北京金融科技产业联盟
BEIJING FINTECH INDUSTRY ALLIANCE

金融行业分布式数据库容器化建设需求研究

北京金融科技产业联盟

2024 年 10 月

版权声明

编制委员会

主任：

聂丽琴

编委会成员：

吕俊锋　罗学平　杜志明　田永江　杨维强　汪　洋

编写组成员：

李嵩嵩　杨　旭　刘月然　朱　鹏　秦延涛　曹　伟　刘海波

吕初伟　曹　源　李建国　应珊珊　龙　恒　杨　欣　孙勇福

杨　锐　陈伟红　黄　贵　赵　亮　燕征南　梁海安　骆明顺

路新英　梁广涛　徐雪涛　朱　洁　郎岳樟　李保洋

编审：

黄本涛　张　蕾

参编单位：

北京金融科技产业联盟

招商银行股份有限公司

中国农业银行股份有限公司

杭州云猿生数据有限公司

腾讯云计算（北京）有限责任公司

金篆信科有限责任公司

平安科技（深圳）有限公司

华为云计算技术有限公司

阿里云计算有限公司

上海爱可生信息技术股份有限公司

北京百度网讯科技有限公司

目　录

一、目标

随着金融行业数字化转型的不断深入，金融机构对于数据管理提出了更高的要求。在容器技术和数据库生态系统不断发展的背景下，分布式数据库容器化已经成为一种趋势，凭借其灵活性、可扩展性和高可用性，成为金融行业数据管理的重要选择，可为金融行业带来如下收益：

提高自动化程度，分布式数据库容器化能使其在多云、混合云环境中灵活地部署和管理，利用云平台的资源调度和自动化能力，实现分布式数据库的自动扩缩容和高可用性。

提升部署和运维效率，分布式数据库容器化简化了部署和管理过程。通过使用容器镜像，将配置和依赖项打包在一起，实现一致性和可重复性的部署，简化生产环境的运维工作，降低人为错误的风险。

优化资源利用率，传统的分布式数据库部署通常需要专用的服务器和资源，导致资源利用率较低。而分布式数据库容器化允许在同一台服务器上进行高密度的数据库实例部署，通过容器的隔离性和资源限制功能，可以更好地利用服务器资源，提高资源利用率和成本效益。

数据库混合部署，金融应用通常需要使用多种类型的数据库，容器化使得在同一个环境中运行和管理不同类型的数据库变得更加高效、经济，实现不同数据库的混合部署。

保证数据安全与隔离，在多租户或共享环境中，相对于共享

物理机，分布式数据库容器化提供了更好的数据隔离和安全保障。每个数据库实例运行在独立的容器中，可以减少数据泄露和相互干扰的风险。

本文从金融行业视角，对分布式数据库和容器技术进行了研究，结合实践中的真实运维需求和应用需求给出了分布式数据库容器化建设方案，为云化时代金融机构运用容器技术解决云平台和分布式数据库间兼容匹配的问题提供有效参考。

二、技术分析

（一）分布式数据库技术

分布式数据库是一种在多个物理或逻辑位置存储数据的数据库系统。在金融行业，分布式数据库提供了高可用性、数据一致性和弹性扩缩容能力。金融机构可以利用分布式数据库处理大规模交易数据，实现快速查询和实时分析，从而提高决策效率。随着金融科技的发展，分布式数据库在支持复杂金融产品、风险管理和客户服务等方面发挥着越来越重要的作用。分布式数据库的性能，源自其在数据分布、事务处理和架构上的独特之处。

1.数据分布方式。分布式数据库与单机数据库存储在本地磁盘或者共享磁盘的方式不同，采用 Sharde-Nothing 架构。数据通过多副本保存在不同的数据节点，每个数据节点拥有一部分数据，多个数据节点共同组成完整数据。目前分布式数据库产品的数据分布有两种方式。

指定分片键分布：数据表以指定分片键及分片算法方式，将

整张表的数据打散分布到各个数据节点。常用的分片策略有: 哈希分片（hash）、范围分片（range）、列表分片（list）、复制分片（duplicate）、多级分片等。

根据大小默认分布: 某些分布式数据产品采用默认分片方式，根据固定大小（如 100M）对数据进行物理切割，每个切割单元称为 Region，每个 Region 的内容与数据表无对应关系，一个 Region 可能包含多个不同数据表中的数据。

2.分布式事务。分布式事务指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于分布式系统的不同节点上，实现的目标是将单机数据库的 ACID 理论(即保证事务的原子性、一致性、隔离性、持久性)，延伸到分布式架构。在分布式系统设计中，CAP 理论（Consistency, Availability, Partition tolerance）指出，一个分布式系统在出现分区故障（Partition）时，无法同时保证数据一致性（Consistency）和可用性（Availability），因此分布式系统设计时通常需要在这三者之间做出权衡。业内常用的分布式事务处理有两种方案: a.两阶段提交（2PC）和三阶段提交（3PC）; b.SAGA 事务（事务补偿方案），是一种长活事务模型，用于处理分布式事务，通过补偿操作来恢复失败的事务。在设计分布式系统时，选择合适的一致性模型和算法是非常重要的，这直接影响到系统的性能、可靠性和用户体验。对于分布式事务功能上的主要包含如下三个方面:

(1)数据一致性：不同于单机数据库的事务只在本机一个节点中完成，分布式数据库的事务需要跨越多个数据节点。事务参与节点数量增加，发生故障的概率随之增加。在故障发生时如何保证事务数据一致性、故障发生后失败的事务如何高效回滚处理，通常需要根据具体的应用场景和业务需求来平衡。

(2)分布式事务性能：由于分布式事务提交时涉及多个节点，与单机数据库事务相比性能有所下降。优化分布式事务性能，是分布式数据库产品的重大挑战。

(3)事务隔离级别：目前大多数分布式数据库产品可以实现的隔离级别为 READ COMMIT，少部分产品可支持多种事务隔离级别。若实现单机数据库如 Oracle 的隔离级别及 MVCC 多版本控制，在分布式架构下具有较高难度。目前业界的产品均在此功能上不断提升探索，并不断探索在分布式场景中解决读写并发操作带来的一致性问题。

3.架构。分布式数据库技术架构包括管理模块、计算模块、存储模块 3 部分组成，技术架构如图 1 所示。

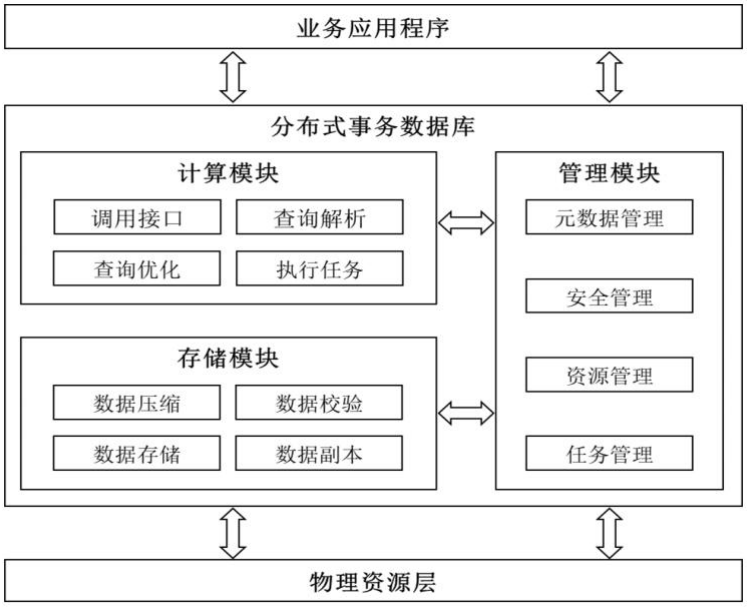

图 1 分布式数据库技术架构图

分布式数据库具有高扩展性、高可用性、分布式事务强一致性等特点，并且部分产品具备数据库云化能力，可为客户提供数据库云服务的标准化交付和快速部署，支持多种部署模式等。经过对主流分布式数据库的架构设计分析，总结出以下几点共同特征:

(1)线性扩展: 分布式数据库软件架构采用分层设计，一般基于 Share-Nothing 架构，采用集群方式部署，实现各组件的灵活扩展，从而提供高性能的数据库服务。同时结合数据动态重分布和读写分离等技术，实现性能的线性扩展。计算节点、数据节点均可横向线性扩展，满足性能及容量的无限扩展需求。

(2)高可用:支持多地多 AZ 组网，任何节点不存在单点故障，可以支持多种组网架构，创新研发数据复制技术，针对不同的业务场景灵活配置不同的策略来满足不同的可用性和可靠性要求，

提高系统吞吐量的同时，实现同城 RPO 为 0。满足数据高可靠、服务高可用要求。支持机房级故障自动切换、支持异地带载演练和异地带压演练。

(3)分布式事务强一致性：具备分布式事务一致性，从数据库层面实现了数据的强一致性，对应用透明，无需应用改造，提升应用迁移效率。

（二）容器技术

云容器技术是一种将应用程序及其依赖项打包在轻量级、可移植的容器中的技术，这些容器可以在任何支持容器运行的环境中快速部署和运行。容器技术通过隔离应用程序及其运行环境，提高了应用的可移植性和可扩展性。在云环境中，容器可以轻松地在不同的云服务和数据中心之间迁移，实现资源的最优配置和成本效益。

1.Pod。Pod 是 K8s 的最小工作单元。每个 Pod 包含一个或多个容器。Pod 中的容器会作为一个整体被 Master 调度到一个 Node 上运行。K8s 引入 Pod 主要基于如下两个目的:

（1）可管理性。有些容器需要紧密联系，Pod 提供了比容器更高层次的抽象，将他们封装到一个部署单元中。K8s 以 Pod 为最小单位进行调度、扩展、共享资源、管理生命周期。

（2）通信和资源共享。Pod 中的所有容器使用同一个网络 namespace，即相同的 IP 地址和 Port 空间。他们可以直接用 localhost 通信，可以共享存储。

2.调度。 K8s 系统的核心任务是创建客户端请求创建的 Pod 对象，并确保其以期望的状态运行。创建 Pod 对象时，调度器为每一个 Pod 资源挑选合适的节点来运行，因此也被称作 Pod 调度器。调度过程中，调度器不会修改 Pod 资源，而是从中读取数据，并根据配置的策略挑选出最适合的节点，而后通过 API 调用将 Pod 绑定至挑选出的节点之上以完成调度过程。

k8s 内建了适合绝大多数场景 Pod 资源调度需求的默认调度器，支持同时使用算法基于原生及可定制的工具来选出集群中最适合运行当前 Pod 资源的一个节点，其核心目标是基于资源可用性将各 Pod 资源公平地分布于集群节点之上。目前，K8s 平台提供的默认调度器也称为“通用调度器”，通过三个步骤完成调度操作：节点预选（Predicate）、节点优先级排序及节点择优。节点预选是基于一系列预选规则对每个节点进行检查，将那些不符合条件的节点过滤掉从而完成节点预选。节点优选是对预选出的节点进行优先级排序，以便选出最适合运行 Pod 对象的节点。最后，从优先级排序结果中挑出优先级最高的节点运行 Pod 象，当此类节点多于一个时，则从中随机选择一个。

3.Service。k8s 的 Service 是一个抽象层，定义了一种访问 Pod 组的方法，通常跨多个容器和节点。Service 为 Pod 组提供固定的 IP 地址和 DNS 名称，以便其他组件可以通过该地址和名称与 Pod 通信，即使背后的 Pod 实例发生变化也不受影响。k8s 提供了几种类型的 Service:

ClusterIP: 这是默认的 Service 类型，为 Service 分配一个内部的 IP 地址，使得 Service 只能在集群内部访问，适用于集群内部通信。

NodePort: 这种类型的 Service 会在集群的所有节点上开放一个端口（NodePort），外部客户端可以通过<NodeIP>:<NodePort>访问 Service。

LoadBalancer: 这种 Service 在 NodePort 的基础上，通过云提供商的负载均衡器向外部暴露一个 Service。云提供商会在其负载均衡器上分配一个外部 IP 地址，流量会通过这个 IP 地址路由到集群中的 Service。

4.存储。k8s 提供了多种存储卷(Volume)类型，用于管理和持久化数据。存储卷可以挂载到 Pod 中，供容器使用。k8s Volume 也支持多种 backend 类型，包括 emptyDir、hostPath、GCE Persistent Disk、AWS Elastic Block Store、NFS、Ceph 等。其中 PersistentVolume 是集群中的一个存储资源，由管理员预先配置或由动态存储供应系统自动供应。PersistentVolume 通常关联到底层的物理存储系统，如 NAS、SAN，云盘，分布式存储或者本地盘。PV 是与 Pod 生命周期独立的，当 Pod 不再存在时，PV 中的数据仍然保持不变。

三、运维需求分析

运维服务能力和运维体系是分布式数据库平稳运行的重要保障。运维体系应该包括完善的监控预警、故障处理、性能优化、

备份恢复、安全防护等服务能力，并形成标准化的操作流程和规范。通过持续完善数据库运维服务能力，建立规范化的运维体系，可以有效提升分布式数据库的稳定性、可用性和性能，降低故障风险，并快速响应和处置故障。

（一）调度

在分布式数据库管理平台中，调度算法和策略直接影响到数据库实例的资源利用效率、业务性能和运维成本。一个高效、智能、灵活的调度系统需要综合考虑多方面的因素和需求，并支持多样化的调度策略和优化手段。应具备如下能力：

1.多维度资源评估与约束控制。调度系统维护节点资源分配表，记录每个节点上已分配的分布式数据库实例及其资源需求(如 CPU 核数、内存大小、存储空间等)，同时通过监控组件实时采集节点的各项资源指标。通过比较节点的资源分配表和实时监控数据，控制计算节点的负载水平和剩余容量。

2.基于分布式数据库服务分级的调度。根据不同的维度，对分布式数据库实例进行分级打标：

（1）业务重要性：根据分布式数据库实例所承载业务的重要程度，划分为核心、重要、一般、长尾等级别。

（2）服务等级协议(SLA)：根据业务对分布式数据库实例的可用性、性能、数据一致性等指标的要求，划分为高保、低保等级别，或者定义更细粒度的 SLA 等级。

（3）业务访问模式：根据分布式数据库实例的读写比例、并发用户数、请求量波动等访问特征，划分为稳态业务(访问平稳)和敏态业务(访问波动大)等。

3.节点密度分级与超卖管理。通过对节点打标区分高密度和低密度节点，并设置不同的超卖比例。根据节点的密度等级和数据库的业务级别，自动匹配最优的部署方案，在资源利用率和业务隔离性之间取得平衡。

4.基于分布式数据库画像的智能混部与负载错峰，通过对分布式数据库实例的资源使用情况进行持续监控和分析，可以自动识别负载类型和高峰期特征，并据此进行智能混部和负载错峰调度，优化节点的资源利用效率。

5.主机组隔离调度，不同的业务负载往往会运行在不同的Node分组中，如经典的数据面和控制面的分离、不同数据库引擎的部署隔离等。

6.Numa Node调度能力，在多Numa Node架构机器下跨Numa访问，对于分布式数据库性能损耗影响很大，需要考虑Numa Node分配和管理能力。

（二）变更

1.配置变更。分布式数据库配置变更是指对参数、设置等进行调整和优化。应符合如下流程:

（1）变更的影响分析和风险评估：评估变更对分布式数据库性能、可用性、安全性等方面的影响，识别潜在的风险和问题，

制定相应的应对措施和回滚方案。通过测试环境和模拟工具，对变更进行预演和验证，确保变更的正确性和有效性，并优化变更的执行方案。

（2）变更流程和审批机制：制定详细的变更执行计划，明确变更的时间窗口、操作步骤、验证标准等，确保变更过程的规范性和一致性。建立标准化的配置变更流程，明确变更的申请、审核、执行、验证等各个环节的职责和要求。对变更进行分级管理，根据变更的影响范围、风险程度和紧急程度，设置不同的审批层级和权限控制。通过变更管理平台和工单系统，实现变更流程的自动化和可视化，确保变更过程的可追溯和可审计。

（3）变更的配置管理和版本控制：建立分布式数据库配置的基线管理和版本控制机制，通过配置管理数据库(CMDB)等工具，实现配置的标准化和可追溯。对不同环境和实例，维护一致的配置模板和参数标准，确保配置的一致性和可重复性。定期对配置进行审计和合规性检查，发现和纠正配置漂移和违规情况，保障分布式数据库的长期稳定运行。

（4）变更的执行和监控：对于需要重启分布式数据库的配置变更，采用滚动升级策略，最小化变更对业务的影响。在变更过程中，对性能指标、资源利用、错误日志等进行实时监控，及时发现和处理异常情况。

（5）变更的验证和回滚机制：根据预定的验证标准（例如冒烟测试用例），对变更后的分布式数据库进行全面的功能、性

能和安全验证，确保变更的正确性和有效性。建立变更的回滚机制和应急预案，在变更出现问题或未达到预期效果时，能够快速回滚到变更前的稳定状态。对变更过程和结果进行详细的记录和总结，形成变更报告和知识库，为后续的变更优化和问题诊断提供参考。

（6）变更的发布和通知：建立变更的发布流程和策略，根据变更的优先级和影响范围，合理安排变更的发布时间和方式。通过邮件、短信、即时通讯等多种渠道，及时向相关人员发送变更通知和进度更新，确保信息的透明和同步。

2.资源变更。资源变更是对分布式数据库所依赖的计算、存储、网络等资源进行调整、优化和扩展，以满足业务增长和性能要求的变化。资源变更应符合如下流程：

（1）资源监控和评估：持续监控分布式数据库的性能指标和资源利用情况，及时发现和预警资源瓶颈和容量不足。定期评估分布式数据库的资源配置和业务需求匹配度，识别优化和扩容的时机。

（2）物理资源评估和决策：评估当前实例所在服务器的资源利用情况，包括 CPU、内存、磁盘、网络等，判断是否有足够的可用资源来满足变更需求。如果当前服务器资源不足，需要评估分布式数据库集群中其他服务器的资源状态，寻找可用资源较为充足的服务器作为迁移目标。综合考虑服务器资源、容灾需求、

数据的重要性和业务等级等因素，制定跨机迁移或集群调度的决策方案。

（3）变更流程和审批机制：制定详细的变更执行计划，明确变更的时间窗口、操作步骤、验证标准等，确保变更过程的规范性和一致性。建立标准化的配置变更流程，明确变更的申请、审核、执行、验证等各个环节的职责和要求。对变更进行分级管理，根据变更的影响范围、风险程度和紧急程度，设置不同的审批层级和权限控制。通过变更管理平台和工单系统，实现变更流程的自动化和可视化，确保变更过程的可追溯和可审计。

（4）跨机迁移和集群调度（可能）：如果需要进行跨机迁移，需要通过集群调度算法，选择集群中资源利用率较低、可用资源较多的服务器作为迁移目标。在迁移过程中，需要确保数据的一致性和完整性，可以采用数据同步、快照、日志传输等技术手段，最小化迁移对业务的影响。

（5）重启带来的副本轮转（可能）：在进行资源变更时，如果需要调整内存等资源限制，可能需要重启数据库进程才能生效。为了最小化重启对业务的影响，可以采用逐个副本轮转的方式，依次对分布式数据库的各个副本进行资源调整和重启。

（6）变更验证和优化机制：监测各项性能指标是否符合预期。基于性能评估的结果，运维人员可以进一步优化资源配置，或者调整变更方案，以实现性能和资源利用率的最佳平衡。如果

变更效果未达预期，或者出现异常情况，需要及时触发回滚机制，将分布式数据库恢复到变更前的状态。

3.版本变更（升级）。版本升级的目的是定期更新分布式数据库软件，以支持新功能、提升性能或修复已知漏洞。不同升级场景下的技术方案和实施流程：

（1）升级策略和风险评估：根据分布式数据库软件的版本发布计划和生命周期，制定长期的升级策略和路线图，平衡新特性引入和稳定性保障的需求。评估不同版本之间的兼容性和差异性，重点关注数据格式、SQL 语法、性能特征、配置参数等方面的变化，以及对现有应用系统的影响。针对每次具体的升级操作，评估其必要性、可行性和风险性，权衡升级的收益、成本和潜在风险，选择最佳的升级时机和方案。在执行升级之前，需要进行一次数据备份，以降低数据损坏的风险，减小发生数据损坏情况下恢复服务的时间窗口。

（2）小版本升级与滚动升级：对于小版本升级(如补丁版本或者次要版本)，通常采用滚动升级的方式，逐个升级分布式数据库实例，以最小化对业务的影响。滚动升级通常可以在分布式数据库正常运行的情况下进行，通过控制升级窗口期和切换时间，最小化业务中断时间。

（3）大版本升级与蓝绿部署：对于大版本升级（如主要版本或者架构变更），由于涉及重大变化和风险，通常采用蓝绿部署的方式，通过业务切换实现平滑升级。蓝绿部署可以最大限度

地降低升级风险，提供回滚保障，但需要额外的硬件资源和较复杂的流量切换机制，需要修改业务连接数据库的 DNS 配置/负载均衡/应用程序配置。

（4）升级验证与回滚机制：无论采用何种升级方式，都需要在升级前后进行严格的验证和测试，确保业务逻辑、数据一致性、性能表现等符合预期。升级验证的主要包括：数据完整性验证、查询结果验证、性能基准测试、业务场景回归测试等，必要时需要进行数据订正或者补偿。建立完善的回滚机制和应急预案，对于升级过程中发现的严重问题或者异常情况，能够及时回退到升级前的稳定状态，保证业务连续性。

（三）切换

切换分为故障自动切换和计划内切换。

1. **故障自动切换（Failover）。**故障自动切换的目标是在 leader（leader 也称主副本）发生故障时，快速、自动地将服务切换到预先准备好的 follower（follower 也称备副本），以最大限度地减少业务中断时间，确保业务连续性。故障自动切换应符合如下流程：

（1）故障检测：通过心跳机制或监控系统，实时监测 leader 的可用性。当 leader 无法访问或响应超时时，触发故障切换流程。通过 follower 或者其他组件二次确认 leader 是否故障，防止误判。

（2）新 leader 选举：根据预设的规则，从可用的 follower 中选择一个作为新的 leader。选举过程考虑 follower 的复制状态、硬件配置、网络延迟、综合负载等因素。确保新 leader 具有最新的数据和最佳的性能。

（3）流量切换：将业务请求从旧 leader 重定向到新 leader。更新数据库连接配置，使应用程序连接到新 leader。确保切换过程快速、平稳，最小化对业务的影响。

（4）恢复复制拓扑：将其他 follower 重新配置为新 leader 的 follower。建立新的复制关系，同步新 leader 的数据更新。监控复制状态，确保数据的一致性和完整性。

（5）故障恢复：对旧 leader 进行故障诊断和修复。根据实际情况，决定是否将其重新加入集群作为 follower。记录故障原因和处理过程，分析原因，优化故障自动切换流程。

2.计划内切换（Switchover）。在分布式数据库运维过程中，有时需要将服务从一个分布式数据库实例切换到另一个实例，以执行维护、升级或配置更改等任务。计划内切换应包括以下流程：

（1）准备阶段：确保备用实例与主实例的数据完全同步。验证备用实例的配置和性能是否满足要求。提交切换计划进行审批（可选）。通知相关人员和系统关于即将进行的切换操作（可选）。

（2）切换阶段：停止向主实例写入新的数据。等待主实例和备用实例之间的数据完全同步。将读取操作切换到备用实例。

将备用实例提升为新的主实例。更新相关系统和客户端的配置，以连接新的主实例。

（3）验证阶段：检查新的主实例是否正常工作。验证数据的完整性和一致性。监控系统性能、查询响应时间、慢查询等指标，确保切换后的稳定性。

（4）回退预案(可选)：如果切换后出现问题，可以将服务切换回原来的主实例。确保回退过程中数据的一致性和完整性。

（四）副本重搭

1.故障副本重搭。分布式数据库副本的健康状态对于整个系统的稳定运行至关重要。由于各种原因，导致复制中断或数据不一致时，需要进行副本重搭操作，以修复受影响的副本。重搭操作应符合如下流程：

（1）问题识别：通过监控系统或人工检查，发现副本出现问题。确定问题的原因和影响范围。

（2）隔离受影响的副本：将出现问题的副本从复制拓扑中移除，避免影响其他节点。停止对该副本的读写操作，防止数据进一步损坏。

（3）重建新副本：创建一个新的副本实例，或清空受影响副本的数据。从 leader 或其他健康的副本进行数据同步，重建副本数据。同步过程中，确保数据的一致性和完整性。重建完成后，将副本加入集群拓扑。

（4）数据同步验证：监控新副本的复制状态和延迟，确保与 leader 库保持一致。对新副本进行数据一致性校验，如比对表结构、数据行数等。进行必要的性能测试，评估新副本的处理能力。

（5）旧副本下线（可选）：进行必要的数据备份或归档。下线旧副本，回收资源或进行后续处理。

2.计划内副本重搭。计划内的副本重搭操作，常见于节点下线、版本升级、硬件更换等场景。与故障引发的非计划性重搭不同，计划内重搭有充分的准备时间，可以更好地控制操作过程和影响范围。计划内副本重搭操作通常包括以下流程：

（1）重搭计划制定：明确重搭的原因、目标和时间安排。评估重搭对业务的影响，制定降低影响的策略。评估是否需要额外的硬件资源。

（2）重建新副本：创建一个新的副本实例，或清空受影响副本的数据。从 leader 库或其他健康的副本进行数据同步，重建副本数据。同步过程中，确保数据的一致性和完整性。重建完成后，将副本加入集群拓扑。

（3）数据同步验证：监控新副本的复制状态和延迟，确保与 leader 保持一致。对新副本进行数据一致性校验，如比对表结构、数据行数等。进行必要的性能测试，评估新副本的处理能力。

（4）服务切换（可选）：选择合适的时间窗口，尽量减少对业务的影响。将读写请求从旧副本切换到新副本。监控切换过程，确保服务的平稳过渡。

（5）旧副本下线（可选）：停止旧副本的复制进程，断开与 leader 的连接。进行必要的数据备份或归档。下线旧副本，回收资源或进行后续处理。

（五）备份恢复

备份可以在分布式数据库故障、人为错误或灾难情况下提供数据恢复的手段。为了确保备份的有效性和可靠性，需要制定完善的备份策略和规范的运维操作流程。

1.数据备份。数据备份主要有自动和手动两种主要方式:

（1）自动备份：制定备份策略，包括备份类型（存储快照/全量/增量/日志）、备份频率、备份保留期限等。创建备份任务，利用分布式数据库厂商提供的备份工具和执行。制定备份数据的管理和归档策略，根据数据的生命周期和法规要求，确定备份数据的保留期限和归档方式。

（2）手动备份： 在特定情况下，如重大变更前、升级迁移前等，需要进行手动备份以确保数据安全。

2.数据库恢复。数据库备份进行恢复的流程:

（1）确定恢复目标：确定需要恢复的目标时间点。确定需要恢复的分布式数据库对象（库、表）和数据范围（行、列）。

（2）选择合适的备份：根据恢复目标时间点，选择最接近且时间点之前的可用备份。考虑备份的类型和完整性，优先选择全量备份，必要时辅以增量备份或日志备份。验证所选备份的完整性和有效性，例如 MD5、CRC 校验码等。

（3）执行恢复操作：创建恢复操作的审计和追踪记录，记录恢复操作的关键事件和操作人员，满足合规性要求。在恢复前，对目标恢复环境进行必要的检查，如是否有足够资源、空间检查、权限配置等。利用分布式数据库厂商提供的备份恢复工具和执行步骤，有序执行恢复操作。监控恢复进度和状态，记录恢复过程中的关键事件和异常情况。

（4）验证恢复结果：执行必要的数据校验和业务验证，确保恢复后的数据库能够正常支持业务运行。生成恢复报告，记录恢复过程、恢复结果和验证情况。

（六）迁移

分布式数据库迁移包括停机迁移和在线迁移。停机迁移适合业务可以接受一定的停机时间、对数据一致性要求较高的场景。在线迁移适合业务连续性要求极高、可以容忍一定的性能影响的场景。分布式数据库迁移应符合如下流程：

1. 迁移准备。 确定迁移的源端和目标端分布式数据库，评估数据量、业务特点、网络条件等因素。检查源端和目标端分布式数据库的配置一致性，包括版本、字符集、网络配置等，避免兼容性问题。选择合适的迁移时间窗口，通常在业务低峰期或维

护窗口进行。制定详细的迁移计划，包括迁移步骤、时间估计、风险评估、回退预案等。

2. 数据导出。使用分布式数据库厂商提供的备份工具，导出源端的全量数据。对导出的数据文件进行压缩和加密，减少数据传输量和保护数据安全。

3. 数据传输。根据网络条件和数据量，选择合适的数据传输方式，如网络流式传输、NAS、对象存储等。使用数据传输工具，将导出的数据文件从源端传输到目标端。监控数据传输进度和网络性能，确保传输的效率和稳定性。

4. 数据导入。在目标端分布式数据库上创建与源端相同的表结构、约束、索引等。使用分布式数据库厂商提供的导入工具，将传输的数据文件导入到目标端分布式数据库。监控数据导入进度和资源消耗，确保导入的效率和可靠性。

5.增量同步。在全量数据导入完成后，启动增量同步进程，实时同步源端分布式数据库的变更到目标端。使用分布式数据库变更捕捉工具，捕获源端增量变更。将捕获的增量变更应用到目标端，保持两端数据的最终一致性。

6.数据校验。在数据导入和增量同步完成后，对目标端分布式数据库进行全面的数据校验。使用数据比对工具，逐表比对源端和目标端的数据行数、校验和等。如果发现数据不一致或丢失，及时进行人工核查和补偿，确保数据的完整性和准确性。

7. 应用切换。在数据校验通过后，准备将业务应用从源端分布式数据库切换到目标端分布式数据库。对源端分布式数据库停写，等待目标端分布式数据库与源端完全一致。修改 DNS/负载均衡/应用的分布式数据库连接配置，将流量切到目标端分布式数据库地址。对业务功能和性能进行监控和验证。如果切换过程中出现问题，及时按照回退预案进行回退操作，保证业务的连续性。

8.迁移完成(可选)。对源端分布式数据库进行一次全量备份下线源端分布式数据库，释放相关的资源。

（七）监控报警

1.监控。在分布式数据库运维中，通过各种监控手段和工具，实时收集和分析数据库的各项性能指标、资源使用情况、SQL 语句执行情况等，可以帮助数据库管理人员全面掌握分布式数据库的运行状态和性能表现。应具备如下能力:

（1）性能指标监控: 监控分布式数据库的关键性能指标，如 CPU 使用率、内存占用、I/O 性能、网络带宽等。跟踪并发连接数、活动会话数、锁等待情况等，评估分布式数据库的并发处理能力和资源利用效率。监测缓存命中率、日志写入速度、checkpoint 频率等，优化内存和持久化性能。

（2）资源使用情况监控: 监控磁盘空间使用情况，包括数据文件、日志文件、备份文件等，确保数据库有足够的存储空间。跟踪数据库的 CPU、内存、I/O 等资源的使用情况，发现资源瓶

颈和异常使用模式。监测分布式数据库的网络连接和带宽使用情况，优化网络配置和减少网络延迟。

（3）SQL 语句执行情况监控：捕获和分析分布式数据库执行的 SQL 语句，了解应用程序的数据访问模式。监控 SQL 语句的执行时间、执行频率、返回结果集大小等，发现性能较差的 SQL 语句和潜在的优化机会。跟踪 SQL 语句的执行计划和资源消耗，如 CPU 时间、逻辑读写、物理读写等，优化 SQL 语句的性能。

（4）异常情况和错误监控：监控分布式数据库的错误日志和告警信息，及时发现和处理异常情况，如死锁、表空间不足、连接异常等。跟踪慢查询日志和诊断日志，分析性能问题的原因，并制定优化方案。监测分布式数据库的安全事件和审计日志，发现和防范潜在的安全威胁和违规操作。

（5）业务监控和关联分析：结合业务系统的关键指标，如响应时间、吞吐量、并发用户数等，分析分布式数据库性能对业务的影响。将分布式数据库监控数据与应用程序监控数据进行关联分析，全面诊断性能问题的根本原因。基于业务的优先级和 SLA 要求，制定监控的策略和告警规则，确保业务的连续性和稳定性。

（6）监控数据的应用和优化：定期分析和挖掘监控数据，利用监控数据建立性能基线，量化评估分布式数据库的性能表现，发现性能的趋势和规律，为容量规划和资源调配提供依据。将监控数据与机器学习算法相结合，实现智能化的异常检测和性能预

测，提高问题识别的准确性和效率。通过对监控数据的深入分析，发现分布式数据库的性能瓶颈和资源利用不平衡，制定针对性的优化方案，如 SQL 调优、索引优化、参数调整等。将分布式数据库监控数据与业务指标相关联，分析业务峰值和增长趋势对数据库性能的影响，提前进行容量评估和资源扩容，确保业务的平稳增长。将监控数据与其他 IT 系统的数据进行整合和关联分析，实现端到端的性能问题诊断和优化。

2.报警。报警是分布式数据库监控的重要延伸和补充，可以帮助运维人员及时发现和处理分布式数据库的各种问题，保障分布式数据库的稳定运行和数据的安全。报警应符合如下功能：

（1）报警规则的设置：根据分布式数据库的类型、业务需求和 SLA 要求，确定需要设置报警的关键指标和阈值，如 CPU 使用率、内存占用、连接数、响应时间等。针对不同的指标和场景，设置不同级别的报警规则，以便区分问题的紧急程度和影响范围。

（2）报警的触发和通知：当分布式数据库的监控指标达到或超过预设的阈值时，自动触发相应级别的报警，并生成详细的报警信息，包括时间、指标、阈值、当前值等。根据报警的级别和影响范围，选择合适的通知方式，确保报警信息能够及时、准确地发送给相关人员。对于严重或紧急的报警，设置多种通知渠道和升级机制，对于严重问题快速升级，确保报警信息的可达性。对于频繁发生的同类报警，可以进行聚合和压缩，在一定时间内

只发送一次汇总报警,减少报警噪音。对于持续时间较长的报警,可以设置递增的报警间隔,避免重复发送对问题处理无益的报警。

(3)报警的关联分析和趋势预测: 结合专家知识和机器学习,构建报警事件的决策树和规则引擎,实现报警的自动分类和风险评估,辅助运维人员的判断和决策。对报警事件进行关联分析,发现不同报警之间的关联性和因果关系,进行根因分析(RCA),全面诊断问题的根本原因。利用机器学习算法和大数据分析技术,对报警数据进行挖掘和建模,预测潜在的问题和风险,提前采取预防措施。结合业务数据和其他监控数据,实现跨层面、跨系统的报警关联分析,提供全局视角下的问题诊断和优化建议。

(4)报警信息的处理和跟踪:建立标准化的报警处理流程,明确不同级别报警的响应时间,确保问题能够得到及时的处理。跟踪和记录报警事件的处理过程和结果,包括原因分析、解决方案、处理时间等,并进行总结和复盘,持续优化报警处理的效率和质量。建立报警反馈机制,收集处理人员对报警有效性的反馈,用于评估和优化报警规则的质量。定期审核和优化报警规则和阈值,根据数据库的运行状况、业务变化和问题反馈,不断完善报警策略,提高报警的准确性和时效性。

(八)数据库访问控制

分布式数据库访问控制通过对数据库的用户、权限、角色等进行精细化管理,确保只有授权的用户才能访问数据库,且只能在其权限范围内进行操作。有效的访问控制可以防止非法用户的

入侵，防止合法用户的越权操作，从而保障数据库的机密性、完整性和可用性。数据库访问控制应遵循如下原则：

1.用户账号管理。根据最小权限原则，为不同的应用系统和运维人员创建独立的分布式数据库用户账号。定期审核和清理过期、废弃、冗余的用户账号，保持账号的精简性和有效性。

2.密码策略管理。制定并执行强密码策略，要求密码符合一定的复杂度要求，如长度、字符类型等。支持定期更换分布式数据库用户密码，防止密码泄露或被暴力破解。保护密码传输和存储的安全，避免明文存储密码。

3.权限与角色管理。基于业务需求和安全要求，设计合理的数据库权限模型和角色层次。遵循权限最小化和职责分离原则，只授予用户必需的操作权限，避免过度授权。通过分布式数据库角色将权限模板化，简化权限管理和分配。定期审核和调整用户权限和角色分配，确保权限配置的准确性和合理性。

4.敏感数据管控。识别和分类分布式数据库中的敏感数据，如个人隐私数据、财务数据等。对敏感数据实施更严格的访问控制，如列级别权限控制、数据脱敏等。启用分布式数据库审计功能，对敏感数据的访问和操作进行记录和监控。

5.访问行为审计。对分布式数据库的访问行为进行全面的审计和记录，包括登录/注销、权限变更、数据操作等。重点关注高危操作和敏感数据的访问，设置实时告警和定期报表。

6.异常访问检测。建立数据库访问行为基线，了解正常的用户访问模式和流量特征。实时监控数据库访问流量，利用机器学习等技术，检测和告警异常访问行为，如暴力破解、注入攻击等。与安全信息和事件管理(SIEM)系统集成，关联分析跨系统的威胁事件。

7.通过白名单限制非法访问。制定明确的分布式数据库白名单策略。在分布式数据库或相关的安全设备(如防火墙、访问控制系统等)中配置白名单规则。定期审核分布式数据库白名单，识别和去除不再需要的白名单项。对数据库白名单的访问进行实时监控，记录访问日志，并设置异常访问的报警规则。

（九）混沌工程

利用混沌工程对分布式数据库缺陷的探索来弥补系统稳定性保障的短板，有助于提前发现和解决分布式数据库系统中的潜在问题，提高系统的可靠性和用户的信任度。分布式数据库混沌工程应具备如下能力:

1.故障场景。支持各种常见类型故障的发起与恢复，覆盖物理机/虚拟机、容器不同底层基础设施。具备如下功能:

（1）支持存储资源故障注入，如: 磁盘填充、磁盘损坏、磁盘速度慢、磁盘不可读、磁盘不可写、文件故障、文件句柄耗尽等;

（2）支持网络资源故障注入，如: 网络抖动、网络丢包、超时、DNS 故障、端口占用等;

（3）支持容器资源故障注入，如：pod、node 等;

（4）支持计算资源故障注入，如：CPU 满载、内存负载故障等;

（5）支持服务、进程资源故障注入，如：进程挂起、进程杀死、服务停止故障、服务器关机、服务器重启等;

（6）支持自定义故障注入，如：Shell 故障注入;

（7）支持典型故障类型注入，如：数据库宕机、数据同步延时、节点故障、突发流量等。

2.实验场景管理。作为分布式数据库混沌平台的核心功能，可对实验进行执行、停止、并行执行、根据业务指标动态控制、超时停止等操作。可利用实验编排可以自动定时实现故障的并行/串行/挂起模拟注入以及故障的自我恢复。通过编排的能力可以构造复杂的故障场景而非只能完成单一故障任务的模拟。支持对实验场景和场景模板的创建、编辑和删除，支持场景库管理、热点场景定义、场景分类、多故障并行组合实验场景、实验爆炸半径定义，及实验环境的创建、恢复等。

3.数据库资源管理。为实验人员提供纳管目标分布式数据库所有环境功能，能够实时获取纳管环境的状态与信息，可对目标数据库资源环境申请、审批；实时展示目标分布式数据库状态、节点信息、部署架构等；具备纳管节点混沌引擎批量安装、卸载、状态查看功能。适配多种架构与类型分布式数据库，支持 X86、C86、ARM 硬件平台，支持 Windows、统信、麒麟等软件平台。

4.混沌实验计划。提供实验计划展示，状态查看，基础故障、故障场景使用与可视化展示，具备如下功能：支持实验计划生命周期管理，如实验计划的创建、编辑、废止等，实验计划的执行时间定义，如立即执行，指定时间执行、周期执行，实验计划优先级设置，手动终止混沌实验，根据基础监控设置防护策略并终止实验，根据系统事件终止实验，历史实验展示可展示全部实验列表、详细信息，复制历史实验直接发起实验。

5.混沌实验监控。实时获取目标分布式数据库监控信息，通过监控界面展示数据库集群主机、数据节点、网关节点、实例四个维度的监控信息。如存活状态、延迟情况、资源利用率等。

（1）支持基础资源监控指标的采集，如：CPU、内存、磁盘、网络等系统指标监控功能。

（2）支持数据库集群状态监控指标采集，如：数据库集群状态、TPS 和 QPS 监控、SOL 平均响应时间统计、锁/等待事件监控、数据库会话连接监控等。支持自定义监控指标设置、故障注入生效验证、故障恢复成功验证、业务稳态数据设置、实施过程中爆炸半径检查。

6.混沌实验结果复盘。实验结束后，可自动生成实验报告，记录整个混沌实验的执行情况，为后续系统优化和改进提供依据。具有实验基础信息、监控指标截图、故障可能原因、解决办法、应急预案、总结建议等。具备实验流程可视化、实验数据图形化，支持实验数据统计分析、报告自动生成等。

（十）智能运维

数据库智能运维利用大数据手段、专家经验引擎快速复制资深数据库管理员的成熟经验，将大量传统手动的分布式数据库运维工作自运维，有效保障数据库服务的安全、稳定及高效运行。

1.实时诊断。实时诊断提供 7*24 小时实时异常诊断与优化服务，利用实时信息进行分析处理，以提高异常发现及处理的效率，可实现定期巡检、主动异常发现和秒级分析优化相结合的分布式数据库健康守护新模式。系统不间断地进行实时诊断，解析分布式数据库问题的根源，并提出优化建议，以便迅速发出告警通知。可支持的诊断能力：锁问题、慢查询、事务问题、性能问题以及高可用性问题等等。实时诊断应具备实时监测、自动诊断、提供建议、高度定制化的能力。

2.SQL 优化。SQL 优化为用户提供一键优化 SQL 语句功能，并给出对应执行计划解析和优化建议。适用于业务优化慢 SQL、代码上线前审查、自检等场景。SQL 优化应具备查询解析和语义理解、性能分析和统计信息、索引选择和创建、可视化执行计划、实时监控和反馈功能。

3.全链路分析。全链路分析收集分布式数据库实例各节点产生的审计日志，通过数据汇集、实时预处理计算，统计/智能分析，给到分析视图与结果建议，并提供了完整剖析链路执行与性能情况的能力，能够协助客户全方位、多维度高效定位问题。

（1）SQL 透视追踪。正向解析：从业务 SQL 到各数据节点，可根据不同条件查找业务 SQL，并查看 SQL 在数据库及集群中的解析过程，以及每一步的性能损耗情况。反向解析：从各数据节点到业务 SQL 的执行过程透视，通过分布式数据库当前 SQL 执行情况，找到性能有损 SQL，并可反向定位业务实际来源。

（2）SQL 聚合分析。业务聚合分析：提取业务 SQL 前缀进行解析，分别送入不同的区域，实现业务编码的快速聚合分析，以及大体量下，业务 SQL 的快速查找。性能统计分析：聚合 SQL 模板分析，业务时段内集群中各项性能指标影响的 SQL 全局排序，并能实时获取模板内 SQL 明细、链路视图、SQL 链路执行过程。

（3）事务聚合分析。业务事务分析：提取第一条事务前缀送入分析，将被拆解后的业务原事务还原，根据业务前缀进行分析与查找。性能统计分析：事务模板化处理，实现时段内集群事务全局性能排序，并能实时获取事务模板内事务的明细，可查看事务内容，也可关联找到分片中被拆解的各个事务。异常事务分析：分辨整个集群中异常的事务，智能分析具体的异常原因以及解决方案。

（4）降本增效助力金融。业务标签管理：当集群数据量体巨大的时候，可以自定义开启部分业务标签名，即可降低集群业务维度数据的聚合量，快速出具结果。图形化及可视化：实时获取链路耗时图、SQL 转义过程及内容。查找及钻取：支持日志全

量字段查找及业务查找、支持业务标签模式及性能模式的数据下钻。

（十一）其他

1.审计日志。分布式数据库通过访问控制可大幅度降低安全风险，但对于具备权限的用户，仍需要审计其操作，防止用户登录信息泄露或访问权限被滥用，审计功能可加强对数据安全、合规的要求。审计日志功能应支持审计日志功能的开启和关闭；审计日志内容包括用户登录、查询、修改、删除等操作类型，及操作时间、客户端来源、执行服务器信息、执行状态、等待时间等。

2.性能容量管理。分布式数据库容器化部署是依托云原生服务模型及管理模式进行管理，为确保数据库系统的服务能力和性能，且兼顾数据库系统的稳定性和业务连续性，需要设计一套技术上可实现的性能容量管理规范。包括性能要求与容量要求等。

（1）性能要求。禁止出现大事务或者长时间不提交、回滚的长事务。禁止联机交易大表出现全表扫描。联机程序谨慎使用JOIN，批处理禁止复杂JOIN（超过3张表）。JOIN操作必须有关联条件，避免笛卡尔积，影响SQL执行效率。禁止使用外键与级联。禁止使用存储过程、自定义函数、触发器、定时作业、视图。系统正式投产前必须评估系统是否需要进行性能压测。尽量用标准SQL语法。表清空建议采用truncate方式。只读应用建议从follower访问。应用分表或采用分区表、多表间并发，避免同一个索引扫描时交叉互锁。避免使用子查询，尽可能改用

JOIN 操作。将复杂 SQL 拆分为多条简单 SQL。表数据量发生较大变化时建议进行统计信息显式收集。

（2）容量要求。合理控制数据库容量，数据库单库、单表容量。单库最大容量建议不超过 500GB，非分区表的单表存储容量(OLTP)建议不超过 20GB，非分区表的单表记录数(OLTP)建议不超过 1000 万行，单库、单表容量过大影响性能、备份恢复效率。如超出此容量规格，建议进行垂直或水平分库分表。建议普通类系统一个实例下不超过 10 个 schema，重要类系统一个实例下不超过 3 个 schema。对于表空间占用较大且持续增长的表必须配置清理策略。联机交易系统当前库禁止保留超过 13 个月的历史数据。禁止在分布式数据库中存储图片，文件等大的二进制数据。

3.资源池管理。分布式数据库容器资源池最佳建设实践为每集群近千台 node 节点级规模，为了充分利用资源池的计算资源和存储资源，且保障分布式数据库系统的绝对可用性，需要设计一套管理完善及技术上可实现的资源池管理标准规范。资源池基线优先级如下：首要保证分布式数据库系统的可用性，其次满足资源治理工作要求，尽可能的保证数据库均匀的分布在分布式数据库容器资源池中。可以设置三种资源池管理基线：蓝线，新架建上限基线，用于保证架建时，机器被充分利用。黄线，资源调平基线，达到该基线，不允许新增任何容器实例。红线，扩容上限，一旦达到该基线，则不允许容器垂直扩容，如有垂直扩容需

求，须先发起跨节点重调度流程，再在调度后的新节点上发起垂直扩容。

4.围绕数据库的 k8s 稳定性。云原生的管控与业务解耦是非常重要的设计原则。对稳定性要求极高的金融应用系统，需要严格控制系统故障的爆炸半径。管控层包括 K8s 底座的问题和故障不会影响到数据服务的正常运行。分布式数据库服务租户之间的故障和性能不能相互影响。K8s 底座上分布式数据库管理系统组件应遵从如下原则：设计上必须兼容 K8s 体系，不可做侵入性改造，避免和定制化底座绑定，APIServer/Etcd/Master/Kubelet 等 K8s 核心组件异常对于正常运行分布式数据库 Pod 不应该批量操作，自定义 Database Operator 做到授权能力，避免代码逻辑问题导致大规模 Pod 对象更新操作。所有行为都需要做到授权验证的，避免误操作发生，需对所有 Database Operator 大规模更新场景充分验证和频率限制。

四、应用需求分析

（一）一致性校验

在分布式数据库高可用架构中，leader 和 follower 之间的数据一致性是保证业务正确性和连续性的基础。然而，由于网络延迟、硬件故障、软件错误等原因，follower 的数据可能与 leader 不一致。这种不一致可能表现为数据丢失、数据重复、数据错乱等问题，严重影响数据的可靠性和完整性。为确保

leader 和 follower 的数据在任意时刻均满足数据一致。副本间一致性校验是关键，应满足如下步骤:

1.制定校验计划。确定校验的范围和频率，选择合适的校验时间窗口。

2.数据导出。如果是在线校验，此步骤可以省略。否则，在 leader 和 follower 上分别导出相同范围的数据，确保导出的数据格式一致，如 SQL 导出或二进制导出，记录导出的时间戳，作为后续对比的基准。

3.数据对比。使用对比工具比较 leader 和 follower 导出(或者在线定义的）的数据。检查数据行数、表结构、索引等是否一致。对于不一致的数据，生成差异报告，记录具体的差异内容。

4.差异分析。分析差异报告,确定不一致的原因和影响范围。判断不一致是由复制延迟、人为操作还是其他因素引起的。评估修复数据不一致的风险和成本。

5.数据修复。根据差异分析结果，提供数据修复方案，如提供修复 sql 等。对于少量的不一致数据，可以手动修改或重新同步。对于大量的不一致数据,可能需要发起计划内副本重搭操作。

（二）容灾

1.容灾实例。分布式数据库实例在 leader 发生故障时能够快速接管业务,保证数据的可用性和连续性。为了应对自然灾害、硬件故障、人为错误等各种潜在风险，与本地高可用不同，容灾实例通常部署在异地数据中心，通过数据同步和网络连接与

leader 保持一致，提供更高级别的容错和灾备能力。搭建数据库容灾实例应满足如下步骤:

（1）异地实例部署。规划异地实例架构，可以是异构（与本地机房不一致的 cpu），也可以是同构。在异地数据中心新建独立的分布式数据库实例。建立异地实例与 leader 之间的网络连接，安全和访问控制策略。

（2）实例备份和恢复。在 leader 上进行全量数据备份（含有数据，日志等），生成一份完整的数据备份。将全量备份传输到异地容灾实例所在的数据中心。传输方法可以通过网络，也可以通过对象存储、NAS 等。在异地容灾实例上进行备份恢复和初始化，确保与 leader 配置一致。

（3）配置数据同步。根据业务需求和技术条件，选择合适的复制模式，如异步复制、半同步复制等。配置 leader 和异地容灾实例之间的复制关系，复制的起始位置应该为全量备份结束的位置和复制方向应该为从 leader 到异地库方向。设置复制的安全认证和传输加密等参数，确保复制的安全性。

（4）延迟复制配置(可选)。根据业务的要求，评估是否需要配置延迟复制。延迟复制可以在 leader 误操作时，使用 follower 数据进行快速恢复。

（5）复制状态监控。实时监控 leader 和异地容灾实例之间的复制状态，如当前主备关系，节点状态，复制时延、复制错误，是否禁止切换等。设置复制时延，节点状态等相关警告和告警阈

值，及时发现和处理复制中的异常情况。定期检查复制的完整性和数据一致性，确保异地容灾实例与 leader 保持同步。

2.容灾切换。在分布式数据库容灾架构中，异地容灾实例是应对主机房故障的重要手段。当主机房发生自然灾害、停电、大面积网络故障等重大故障时，需要快速、可靠地将业务切换到异地容灾实例，确保业务连续性和数据可用性。容灾切换应满足如下步骤:

（1）故障确认与评估。监控系统发现主机房故障，如电力中断、网络断开等。确认故障范围和影响程度，评估主机房恢复时间和数据库受损情况。决定是否启动异地容灾切换预案，评估切换的风险和影响。

（2）启动容灾预案。通知相关团队和人员，包括数据库运维、应用开发等。通过预先准备的控制台（白屏）或者脚本（黑屏）操作步骤，启动切换。

（3）leader 停写（可选）。如果主机房故障未导致 leader 完全不可访问，应断开 leader 与应用的所有连接，将 leader 设置为只读，停止对 leader 的写入操作，避免双写造成数据不一致。如有需要，例如后续有掉电风险，应尝试优雅地关闭 leader 数据库，避免数据损坏。记录 leader 停写的时间点和日志位置，作为后续数据恢复的起点。

（4）切换应用连接。修改 DNS 配置/负载均衡配置/应用程序的连接配置，将流量导向异地容灾实例。

（5）恢复业务运行。验证应用程序与异地容灾实例的连通性，确保切换后的业务可用性。逐步恢复业务应用和服务，监控业务运行状态和性能指标。与业务团队密切沟通，确认业务运行正常，无异常情况发生。

3.回切。在主机房恢复后，应在一定时间范围内，将流量再次切回主机房:

（1）主机房恢复后的数据补全。持续关注主机房的恢复进度，评估恢复时间和数据库损坏情况。当主机房恢复后，启动 leader 数据库， 并对在切换至异地容灾实例之前未能同步的数据部分进行备份。与异地容灾实例建立数据同步关系。将异地容灾实例切换期间的增量数据同步回 leader，确保数据的完整性和一致性。比对 leader 和异地容灾实例的数据，进行必要的数据修复或重建。

（2）切换回主机房。评估 leader 数据库的性能和稳定性，确认其可以承载业务负载。制定切换回主机房的计划，包括停止异地容灾实例写入、切换负载均衡/DNS/应用配置等步骤。在业务低峰期执行切换操作，将业务流量逐步导向 leader。验证业务运行状态和数据完整性，确保切换回主机房后的系统稳定性。

(3)恢复异地容灾实例。切换完成后，恢复 leader 到异地容灾实例的数据同步，确保与 leader 保持一致，以便于后续的容灾切换。

五、建设方案

（一）分层抽象

分布式数据库容器化作为一个通用的模型，旨在为各种分布式数据库在 K8s 中的部署和运维提供一种抽象和统一的表示方法。该模型将容器化分布式数据库的管理映射到位于四个层次的对象上，包括：Cluster、Component、InstanceSet 和 Instance，形成了一个分层的架构。容器化分层图见图 2。

Cluster 层：一个 Cluster 对象代表一个完整的分布式数据库集群，包括数据库的所有组件和服务。

Component 层：Component 表示组成 Cluster 对象的逻辑组件，如元数据管理、数据存储、查询引擎等。每个 Component 对象都有其特定的职责和功能。一个 Cluster 对象里包含一到多个 Component 对象。

InstanceSet 层：InstanceSet 对象管理 Component 对象里多个副本所需的工作负载，感知副本的角色。一个 Component 对象包含一个 InstanceSet 对象。

Instance 层：Instance 对象代表 InstanceSet 对象内的一个实际运行实例，对应 K8s 中的 Pod。一个 InstanceSet 对象可以管理零到多个 Instance 对象。

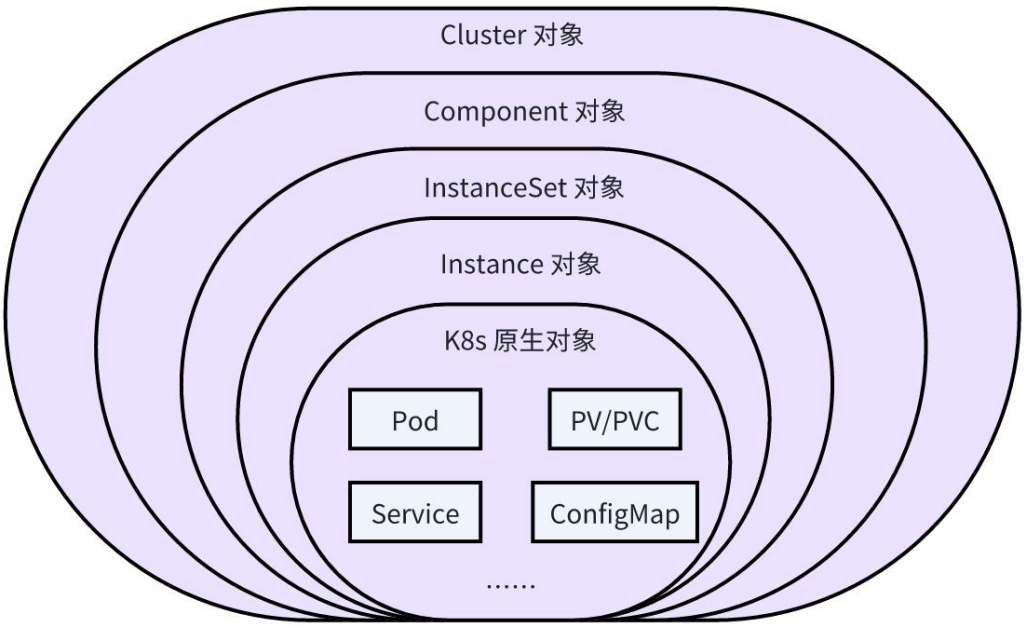

图 2 容器化分层图

1.Cluster 层。Cluster 对象是指一个完整的、可运行的分布式数据库，由多个 Component 对象组成，这些 Component 对象协同工作，分别提供数据存储、计算和管理功能。Cluster 对象封装了分布式数据库的所有 Component 对象及其配置和资源，代表了数据库集群的整体行为和属性。包括：创建、扩缩容、隔离、销毁等完整的生命周期操作，以及通过 Component 对象关联监控、日志、LB、备份、HA 、调度策略等周边属性定义，共同协作完成分布式数据库对象的操作完备性。Cluster 对象的生命周期支持创建、运行、变更、停止、重启、休眠、进入回收站和销毁等运维操作。Cluster 对象的生命周期管理需要根据 Component 对象的拓扑结构（通常是一个有向无环图 DAG）进行编排。各个 Component 对象之间通常存在服务依赖关系，Cluster 对象需要

定义这些 Component 对象之间的依赖关系。在 Cluster 对象进行一致性备份时，需要协调多个 Component 对象的备份过程，以确保整个集群的数据一致性和可恢复性。恢复过程需要按照备份的一致性位点和顺序，对元数据、分片数据等进行有序的恢复和重放。

2.Component 层。Component 是指分布式数据库集群中的一个逻辑组件或服务，每个 Component 对象都承担特定的功能和角色，如数据存储、元数据管理、查询处理等。Component 对象封装了特定功能所需的资源、配置和行为，是 Cluster 对象的基本构建块。多个 Component 对象相互协作，共同提供完整的数据库服务。Component 对象需要管理分布式数据库集群里一个组件所有副本的生命周期。支持创建、重启、升级、修改配置、垂直扩缩容、水平扩缩容、高可用切换、跨节点迁移、休眠、进入回收站和销毁等运维操作。Component 对象创建时，可根据组件的配置、资源需求、调度要求，在适当的节点上创建组件副本的实例。当有新的功能特性、性能优化或者 bug 修复时，Component 对象可进行升级操作。通过修改 Component 对象的配置参数，动态调整多个副本的配置。Component 对象支持多个副本的高可用切换，当节点发生不可恢复故障，或者在进行节点的维护、升级时，可以使用跨节点迁移功能将组件副本从一个节点迁移到另一个节点。Componet 支持相关配置，可选配置项如表 1 所示。

表 1 可选配置项

配置项	说明
容器镜像	设置 Component 对象运行所需的容器镜像。
资源配置	设置 Component 对象所需的 CPU、内存、存储等资源。
挂载卷	设置 Component 对象需要挂载的数据卷。
副本数	设置 Component 对象运行时期望的副本个数。
配置参数	设置 Component 对象的运行时配置，包括配置文件中的参数、全局变量、会话变量等。
调度策略	设置亲和性和反亲和性、指定污点和污点容忍、调度策略以及服务优先级等参数，来控制副本的调度行为和资源分配。
亲和性	允许指定副本与其他副本或节点的调度关系。
反亲和性	与亲和性相反，用于指定副本之间的排斥关系。
污点	是节点的一种特殊标记，表示节点存在某些限制或约束，如专用硬件、网络条件等。
调度策略	决定了副本在节点上的分布方式和优先级。
服务优先级	指定副本的调度优先级，影响其在资源竞争时获得节点资源的顺序。
暴露服务	设置 Component 对象对外提供的网络方式（Hostnetwork/Nodeport/LoadBalancer 等），以及服务端口。
Hostnetwork	使用宿主机的网络命名空间，将容器的端口直接映射到宿主机的网络接口上。
Nodeport	在每个节点上开启一个静态端口，将服务映射到该端口上，通过节点 IP 和端口访问服务。
LoadBalancer	利用云平台或外部负载均衡器,将服务暴露为一个外部可访问的 IP 地址。
配置系统账号	Component 对象之间互相访问，以及执行自动化运维任务，需要通过配置专用的系统账户来实现。
备份配置	设置 Component 对象的备份策略。定义备份的类型、频率、保留期限、存储位置等参数。
监控配置	设置 Component 对象的监控采集和推送方式。

日志采集配置	设置 Component 对象的日志采集和转储规则。

3. InstanceSet 层。 Component 通过可感知角色的 InstanceSet 来管理 Component 对象的多个副本实例，满足数据库多副本场景下的各种需求，提供更加灵活和可控的副本管理能力。例如：副本角色的探测和管理；为不同副本设置不同的资源、配置、调度策略；探测副本和副本之间复制通道的健康状态；升级时的并行和顺序控制；上线、下线指定的副本；管理跨 K8s 集群部署的多个副本。InstanceSet 对象允许用户干预并调整副本的角色，如将一个备节点提升为主节点，或将一个主节点降级为备节点，并自动执行角色调整所需的变更动作，确保副本在新角色下正常工作。InstanceSet 对象提供角色感知的监控功能，会收集副本之间的同步状态、复制延迟，以及副本的可用性、负载等指标，以实现自动修复（auto healing）功能。InstanceSet 对象支持为不同角色的副本指定不同的配置和资源，例如为了确保性能和可靠性，为主节点分配更高的 CPU 和内存。InstanceSet 对象可以精确控制不同角色的副本在重启性变更时的升级顺序，从而最小化系统升级和维护对业务的影响。InstanceSet 对象支持上线、下线指定的副本。在某个节点发生硬件故障时，如在机架、机房裁撤时，需要将这个节点上的副本进行下线。InstanceSet 对象还支持管理跨多个 K8s 集群（地理区域）的副本，以实现同城多机房、两地三中心、三地五副本等容灾形态。

4. Instance 层。Instance 层作为数据库和 K8s 之间的桥梁，将数据库的一个具体副本实例映射到一组 k8s 资源，包括 Pod、PVC、Service、ConfigMap 等。这种映射关系的建立和管理是 Instance 层的核心职责。Instance 对象对应着一个 Pod，每个 Pod 代表集群中运行的一个节点副本，可以是主数据库、从数据库、代理节点或者元数据节点。在 Instance 里对 k8s 的 Pod 功能进行扩展，可以提供更丰富和灵活的数据库副本管理能力，满足数据库场景下的特定需求。Instance 对象支持为数据库副本配置多个独立的网络地址，每个地址对应不同的网络接口和 IP，可以独立配置安全策略和访问控制。Instance 对象允许用户动态修改数据库副本的资源配置，如 CPU、内存等，并进行热更新，无需重启 Pod。为了实现资源的热更新，Instance 需要利用容器运行时的特性，持续监控数据库副本的资源使用情况，如 CPU 利用率、内存占用、I/O 等。

（二）管理平台 API 设计

基于分布式数据库容器化四层抽象模型开发的分布式数据库容器化管理平台需要提供 API，以便用户能够在平台上统一管理各家厂商提供的分布式数据库。对用户（数据库的管理员和使用者）而言，这些 API 应该适用于不同厂商的分布式数据库，并提供一致的操作方式。对于厂商而言，由于不同厂商的分布式数据库具有不同的架构和特性，平台需要提供扩展机制来支持这些差异。分布式数据库容器化管理平台 API 设计图见图 3。

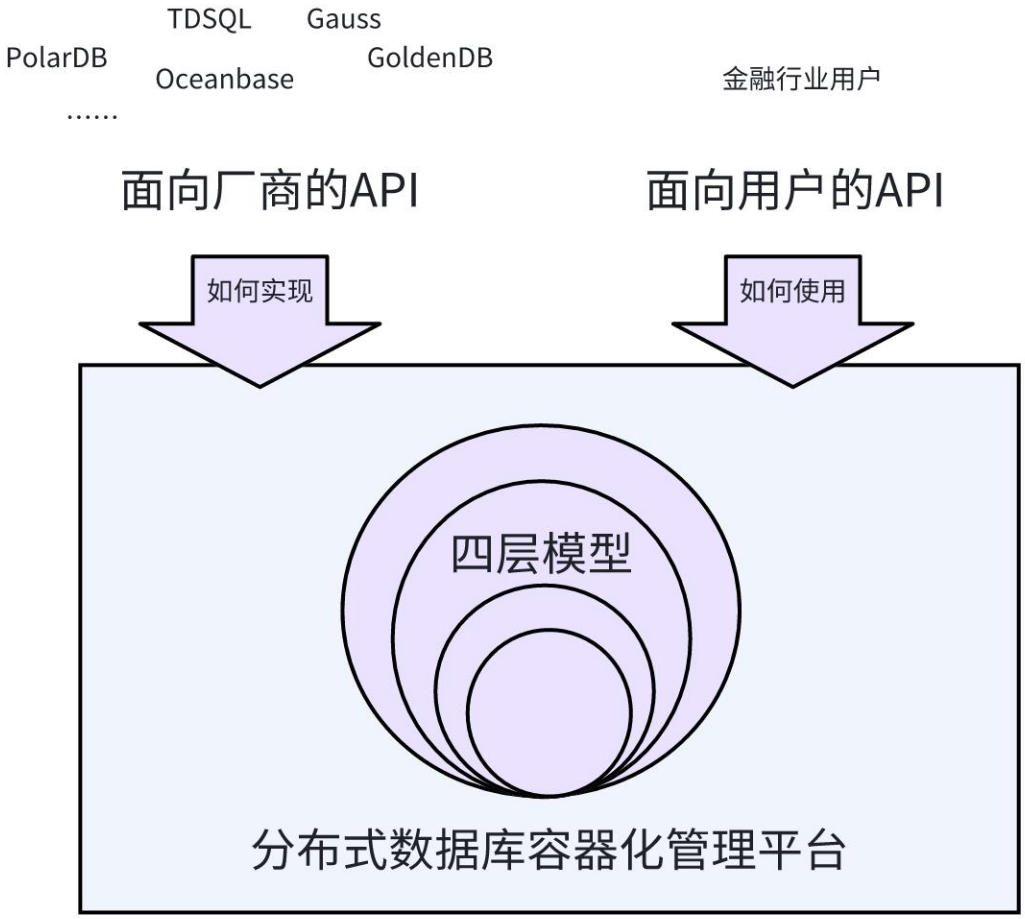

图 3 分布式数据库容器化管理平台 API 设计图

1.面向厂商的 api。有 Component api 和 Cluster api。

（1）Component api。数据库厂商可为其分布式数据库的每个组件创建一个定义文件。内容包含：服务声明，Component 可以提供的服务类型名字，以及版本号。容器镜像，指定该 Component 使用的所有容器的 Docker 镜像，包括镜像名称、版本号、仓库地址等。容器模板，包括注入容器的环境变量、所有挂载的卷等。角色，定义该 Component 的副本支持哪几种角色，

并提供探测副本角色的方法，如执行 SQL 查询。Component 的配置文件模板。脚本，包括 Component 生命周期里会调用的脚本，初始化脚本、启动脚本等。Actions，特定系统事件触发后需要执行的操作。系统账号、网络服务、可定制的监控（如配置 exporter sidecar 容器镜像等）、引用其他组件或服务。

（2）Cluster api。Cluster 文件包含以下内容: 拓扑结构的集合，定义可能的拓扑结构的集合。组件定义，为每个拓扑结构中的组件定义行为。启动顺序，拓扑结构中 Component 的启动顺序，如某些组件需要在其他组件启动后才能启动。相互引用，Component 之间的相互引用和依赖，如一个组件需要访问另一个组件的服务地址和端口。

2.面向用户的 API。用户通过 Cluster API 在一个地方集中配置集群和各组件的属性。Cluster API 内容有: 部署使用的拓扑结构,例如有无代理节点,采用主从结构还是多主对称结构等。分片对应的 component 实例,如果分布式数据库的数据划分为多个水平分片，每个水平分片（Shard）对应一个 Component 实例。为每个 Component 指定所需的副本数量、资源、挂载卷、调度策略等。对外暴露的网络服务地址的形式，例如 hostNetwork，nodePort，loadBalancer。Component 监控、日志采集等的开关设置。备份设置，备份策略、备份时间、备份间隔等。集群结束时释放资源的策略设置，如: 不停机、仅停止计算资源等。

（三）OPENAPI 标准化

各厂家应提供符合行业标准的 OPENAPI，覆盖分布式数据库核心操作。OPENAPI 核心功能至少包括以下关键功能：集群创建、用户管理、备份恢复、故障与主从切换等。OPENAPI 应统一 API 命名，确保易用性和一致性。应为 OPENAPI 标准化建立工作组，定期更新标准以适应技术发展。

六、展望与计划

分布式数据库的容器化为分布式数据库的部署、管理和扩展提供了新的解决方案，同时也带来了新的挑战。随着技术的不断发展，容器化技术在分布式数据库领域得到越来越多的应用。

后续将计划着手分布式数据库容器化相关实践与标准研究，最终形成行业标准，用来指导分布式数据库的容器化应用。在这一进程中，需要进一步加强行业内的协作，包括芯片、操作系统、中间件、云平台等领域厂商的合作，共同打造完整的生态体系，提供一站式的解决方案。