Confluent Kafka 的最佳实践

NetApp artificial intelligence solutions

NetApp
August 18, 2025

目录

Confluent Kafka 的最佳实践

TR-4912： NetApp Confluent Kafka 分层存储的最佳实践指南

Karthikeyan Nagalingam、Joseph Kandatilparambil、 NetApp Rankesh Kumar、Confluence

Apache Kafka 是一个社区分布式事件流平台，每天能够处理数万亿个事件。 Kafka 最初被认为是一个消息队列，基于分布式提交日志的抽象。自 2011 年由 LinkedIn 创建并开源以来，Kafka 已经从一个消息队列发展成为一个成熟的事件流平台。 Confluent 通过 Confluent 平台提供 Apache Kafka 的分发。 Confluent 平台为 Kafka 提供了额外的社区和商业功能，旨在增强大规模生产中运营商和开发人员的流媒体体验。

本文档通过提供以下内容描述了在 NetApp 对象存储产品上使用 Confluent 分层存储的最佳实践指南：

- 使用NetApp对象存储进行汇合验证 – NetApp StorageGRID
- 分层存储性能测试
- NetApp存储系统上 Confluent 的最佳实践指南

为什么选择 **Confluent** 分层存储?

Confluent 已成为许多应用程序的默认实时流媒体平台，尤其是对于大数据、分析和流媒体工作负载。分层存储使用户能够在 Confluent 平台中将计算与存储分开。它使存储数据更具成本效益，使您能够存储几乎无限量的数据并按需扩大（或缩小）工作负载，并使数据和租户重新平衡等管理任务更加容易。 S3 兼容存储系统可以利用所有这些功能，将所有事件集中在一个地方，实现数据民主化，从而无需复杂的数据工程。有关为什么应该为 Kafka 使用分层存储的更多信息，请查看"本文由 Confluent 撰写"。

为什么选择**NetApp StorageGRID**进行分层存储?

StorageGRID是NetApp推出的业界领先的对象存储平台。 StorageGRID是一种软件定义的基于对象的存储解决方案，支持行业标准对象 API，包括 Amazon Simple Storage Service (S3) API。 StorageGRID大规模存储和管理非结构化数据，以提供安全、持久的对象存储。内容被放置在正确的位置、正确的时间和正确的存储层，从而优化工作流程并降低全球分布的富媒体的成本。

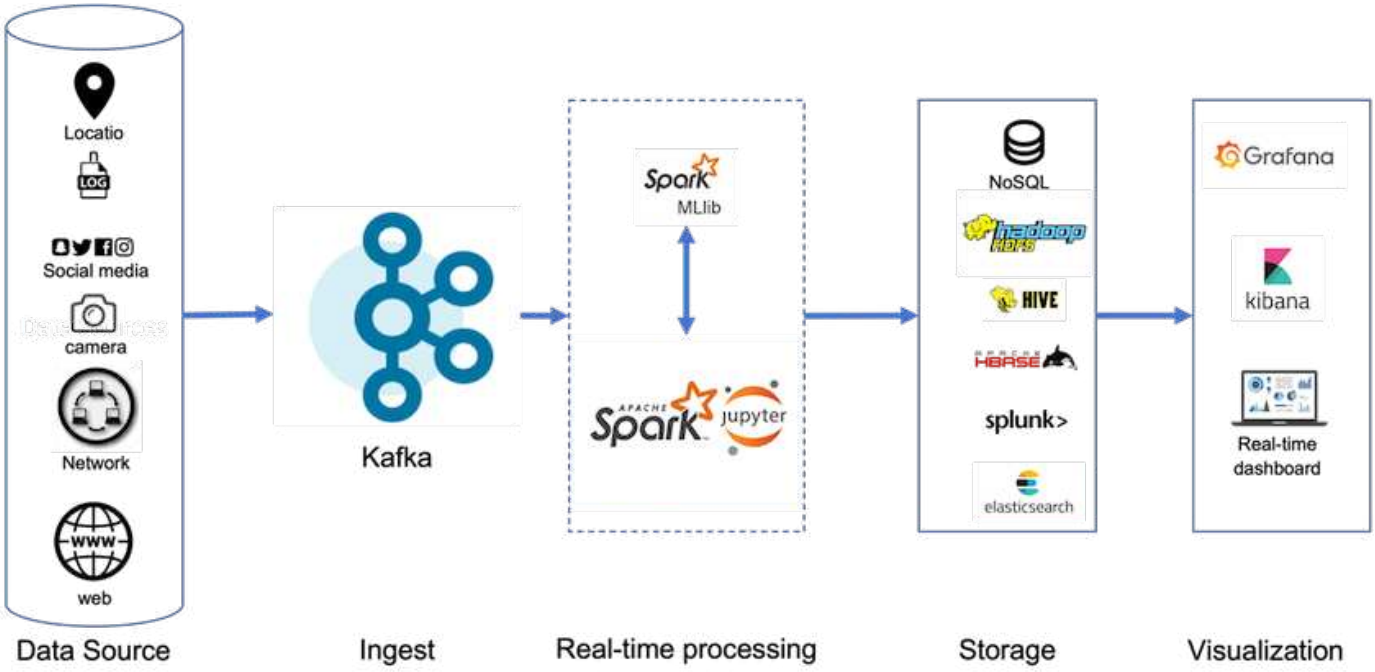
StorageGRID最大的区别在于其信息生命周期管理 (ILM) 策略引擎，它支持策略驱动的数据生命周期管理。策略引擎可以使用元数据来管理数据在其整个生命周期内的存储方式，以便最初优化性能，并随着数据老化自动优化成本和耐用性。

启用 **Confluent** 分层存储

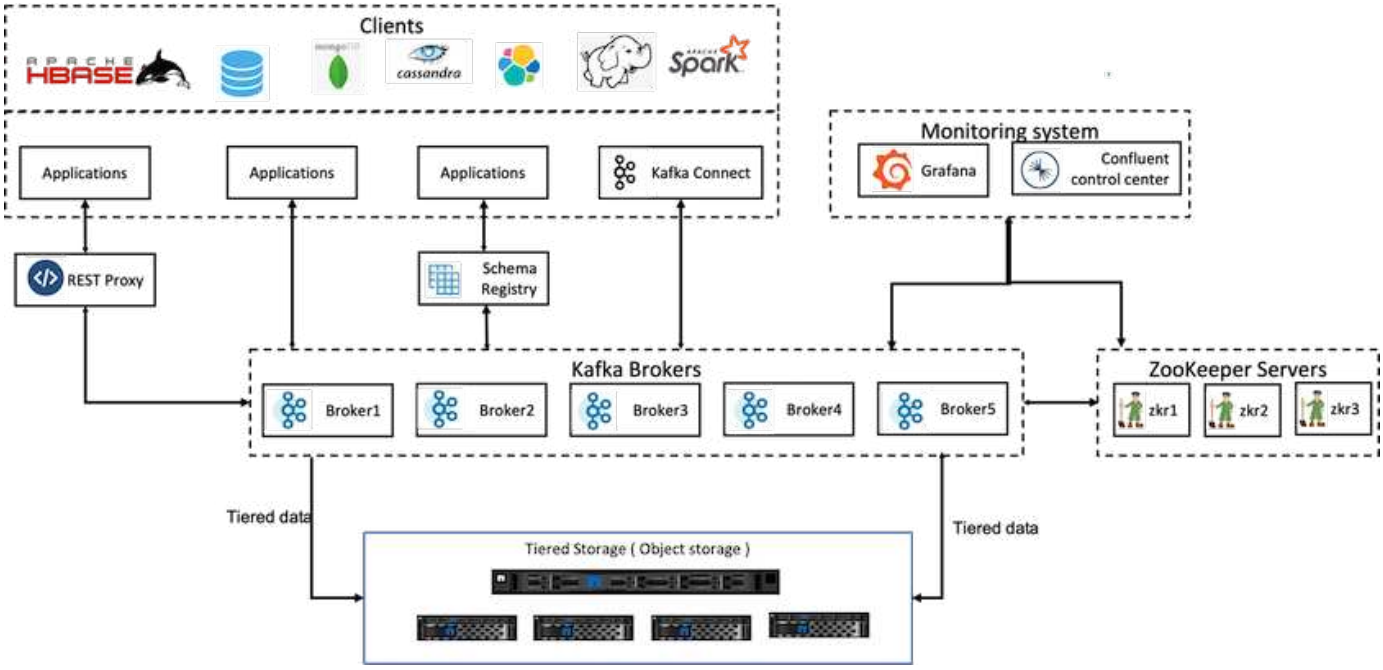
分层存储的基本思想是将数据存储任务与数据处理任务分离。通过这种分离，数据存储层和数据处理层可以更轻松地独立扩展。

Confluent 的分层存储解决方案必须应对两个因素。首先，它必须解决或避免常见的对象存储一致性和可用性属性，例如 LIST 操作中的不一致和偶尔的对象不可用。其次，它必须正确处理分层存储与 Kafka 的复制和容错模型之间的交互，包括僵尸领导者继续分层偏移范围的可能性。 NetApp对象存储提供一致的对象可用性和 HA 模型，使分层存储可用于层偏移范围。 NetApp对象存储提供一致的对象可用性和 HA 模型，使分层存储可用于层偏移范围。

通过分层存储，您可以使用高性能平台在流数据尾部附近进行低延迟读写，还可以使用更便宜、可扩展的对象存储（如NetApp StorageGRID）进行高吞吐量历史读取。我们还为带有 netapp 存储控制器的 Spark 提供了技术解决方案，详细信息请见此处。下图显示了 Kafka 如何融入实时分析管道。



下图描述了NetApp StorageGRID如何作为 Confluent Kafka 的对象存储层。



解决方案架构细节

本节介绍用于 Confluent 验证的硬件和软件。此信息适用于使用NetApp存储的 Confluent Platform 部署。下表涵盖了经过测试的解决方案架构和基本组件。

解决方案组件	详细信息
Confluent Kafka 版本 6.2	• 三位动物园管理员 • 五台经纪商服务器 • 五款工具服务器 • 一个 Grafana • 一个控制中心
Linux（Ubuntu 18.04）	所有服务器
NetApp StorageGRID用于分层存储	• StorageGRID软件 • 1 x SG1000（负载均衡器） • 4 个 SGF6024 • 4 x 24 x 800 SSD • S3 协议 • 4 x 100GbE（代理和StorageGRID实例之间的网络连接）
15台富士通PRIMERGY RX2540服务器	每个配备：* 2 个 CPU，共 16 个物理核心 * Intel Xeon * 256GB 物理内存 * 100GbE 双端口

技术概述

本节介绍此解决方案所使用的技术。

NetAppStorageGRID

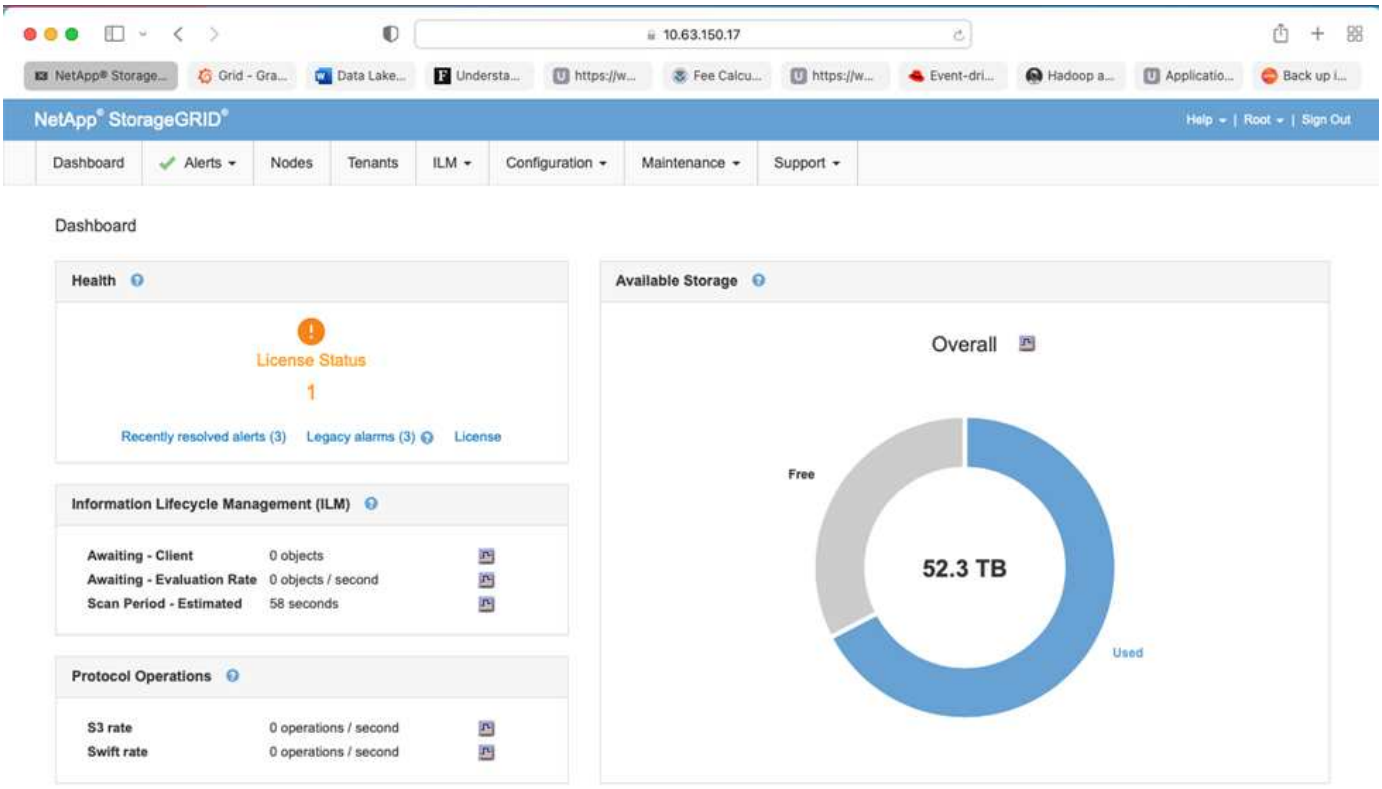
NetApp StorageGRID是一个高性能、经济高效的对象存储平台。通过使用分层存储，Confluent Kafka 上存储在本地存储或代理的 SAN 存储中的大部分数据都被卸载到远程对象存储。此配置可减少重新平衡、扩展或收缩集群或更换故障代理的时间和成本，从而显著改善操作。对象存储在管理驻留在对象存储层的数据方面发挥着重要作用，因此选择正确的对象存储非常重要。

StorageGRID使用分布式、基于节点的网格架构提供智能、策略驱动的全局数据管理。它通过其无处不在的全局对象命名空间与复杂的数据管理功能相结合，简化了 PB 级非结构化数据和数十亿个对象的管理。单次调用对象访问可跨站点扩展，并简化高可用性架构，同时确保持续的对象访问，无论站点或基础设施是否中断。

多租户允许在同一网格内安全地为多个非结构化云和企业数据应用程序提供服务，从而提高NetApp StorageGRID的投资回报率和用例。您可以使用元数据驱动的对象生命周期策略创建多个服务级别，从而优化跨多个地区的耐用性、保护性、性能和位置性。用户可以调整数据管理策略并监控和应用流量限制，以便在不断变化的 IT 环境中随着需求的变化而无中断地重新调整数据格局。

使用网格管理器进行简单管理

StorageGRID Grid Manager 是一个基于浏览器的图形界面，允许您在单一玻璃窗格中配置、管理和监控全球分布位置的StorageGRID系统。



您可以使用StorageGRID Grid Manager 界面执行以下任务：

- 管理全球分布的 PB 级对象存储库，例如图像、视频和记录。
- 监控网格节点和服务以确保对象可用性。
- 使用信息生命周期管理 (ILM) 规则来管理对象数据随时间推移的放置。这些规则控制着对象数据被摄取后会发生什么、如何防止数据丢失、对象数据存储在何处以及存储多长时间。
- 监控系统内的交易、性能和操作。

信息生命周期管理政策

StorageGRID具有灵活的数据管理策略，包括保留对象的副本以及使用 EC（擦除编码）方案（例如 2+1 和 4+2 等）来存储对象，具体取决于特定的性能和数据保护要求。由于工作负载和需求随时间而变化，ILM 策略通常也必须随时间而变化。修改 ILM 策略是一项核心功能，允许StorageGRID客户快速轻松地适应不断变化的环境。

性能

StorageGRID通过添加更多存储节点来扩展性能，这些存储节点可以是虚拟机、裸机或专用设备，例如 "SG5712、SG5760、SG6060 或 SGF6024"。在我们的测试中，我们使用 SGF6024 设备以最小尺寸的三节点网格超越了 Apache Kafka 的关键性能要求。当客户使用额外的代理扩展他们的 Kafka 集群时，他们可以添加更多的存储节点来提高性能和容量。

负载均衡器和端点配置

StorageGRID中的管理节点提供网格管理器 UI（用户界面）和 REST API 端点来查看、配置和管理您的StorageGRID系统，以及审计日志来跟踪系统活动。为了为 Confluent Kafka 分层存储提供高可用性 S3 端点，我们实现了StorageGRID负载均衡器，它作为管理节点和网关节点上的服务运行。此外，负载均衡器还管理本地流量并与 GSLB（全局服务器负载均衡）对话以帮助进行灾难恢复。

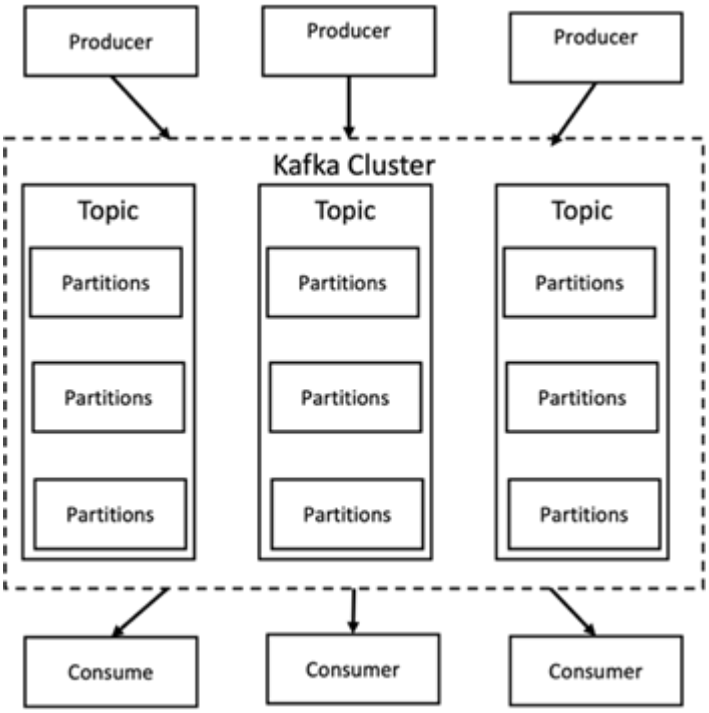
为了进一步增强端点配置， StorageGRID提供了内置于管理节点的流量分类策略，让您监控工作负载流量，并对您的工作负载应用各种服务质量 (QoS) 限制。流量分类策略应用于网关节点和管理节点的StorageGRID负载均衡器服务上的端点。这些策略可以帮助流量整形和监控。

StorageGRID中的流量分类

StorageGRID具有内置 QoS 功能。流量分类策略可以帮助监控来自客户端应用程序的不同类型的 S3 流量。然后，您可以创建并应用策略来根据输入/输出带宽、读/写并发请求数或读/写请求率限制此流量。

阿帕奇卡夫卡

Apache Kafka 是一个用 Java 和 Scala 编写的使用流处理的软件总线的框架实现。其目的是提供一个统一、高吞吐量、低延迟的平台来处理实时数据馈送。 Kafka可以通过Kafka Connect连接外部系统进行数据导出和导入，并提供Java流处理库Kafka Streams。 Kafka 使用基于 TCP 的二进制协议，该协议针对效率进行了优化，并依赖于“消息集”抽象，该抽象自然地将消息组合在一起，以减少网络往返的开销。这使得更大的顺序磁盘操作、更大的网络数据包和连续的内存块成为可能，从而使 Kafka 能够将突发的随机消息写入流转换为线性写入。下图描述了Apache Kafka的基本数据流。



Kafka 存储来自任意数量的进程（称为生产者）的键值消息。数据可以划分到不同主题内的不同分区中。在分区内，消息严格按照其偏移量（消息在分区内的位置）排序，并与时间戳一起索引和存储。其他称为消费者的进程可以从分区读取消息。对于流处理，Kafka 提供了 Streams API，允许编写 Java 应用程序使用来自 Kafka 的数据并将结果写回 Kafka。 Apache Kafka 还可以与外部流处理系统（如 Apache Apex、Apache Flink、Apache Spark、Apache Storm 和 Apache NiFi）协同工作。

Kafka 在一个或多个服务器（称为代理）的集群上运行，所有主题的分区分布在集群节点上。此外，分区被复制到多个代理。这种架构允许 Kafka 以容错方式传递大量消息流，并允许它取代一些传统的消息传递系统，如 Java 消息服务 (JMS)、高级消息队列协议 (AMQP) 等。自 0.11.0.0 版本以来，Kafka 提供事务写入功能，可使用 Streams API 提供一次流处理。

Kafka 支持两种类型的主题：常规主题和压缩主题。常规主题可以配置保留时间或空间界限。如果有记录超过指定的保留时间或超出了分区的空间限制，则允许 Kafka 删除旧数据以释放存储空间。默认情况下，主题的保留时间配置为 7 天，但也可以无限期地存储数据。对于压缩主题，记录不会根据时间或空间界限而过期。相反

，Kafka 将后续消息视为具有相同密钥的旧消息的更新，并保证永远不会删除每个密钥的最新消息。用户可以通过写入具有特定键的空值的所谓墓碑消息来彻底删除消息。

Kafka 中有五个主要 API：

- 生产者 API。允许应用程序发布记录流。
- *消费者 API。*允许应用程序订阅主题并处理记录流。
- 连接器 API。执行可重用的生产者和消费者 API，将主题链接到现有应用程序。
- 流 API。该 API 将输入流转换为输出并产生结果。
- *管理 API。*用于管理 Kafka 主题、代理和其他 Kafka 对象。

消费者和生产者 API 建立在 Kafka 消息传递协议之上，并为 Java 中的 Kafka 消费者和生产者客户端提供参考实现。底层消息传递协议是一种二进制协议，开发人员可以使用任何编程语言编写自己的消费者或生产者客户端。这使得 Kafka 从 Java 虚拟机 (JVM) 生态系统中解放出来。 Apache Kafka wiki 中维护了可用的非 Java 客户端列表。

Apache Kafka 用例

Apache Kafka 最受欢迎的用途是消息传递、网站活动跟踪、指标、日志聚合、流处理、事件源和提交日志记录。

- Kafka 提高了吞吐量、内置分区、复制和容错能力，使其成为大规模消息处理应用程序的良好解决方案。
- Kafka 可以将跟踪管道中的用户活动（页面浏览量、搜索量）重建为一组实时发布-订阅源。
- Kafka 经常用于运营监控数据。这涉及汇总来自分布式应用程序的统计数据以生成集中的操作数据。
- 许多人使用 Kafka 来替代日志聚合解决方案。日志聚合通常从服务器上收集物理日志文件并将它们放在一个中心位置（例如，文件服务器或 HDFS）进行处理。 Kafka 抽象文件细节并将日志或事件数据以消息流的形式提供更清晰的抽象。这允许更低延迟的处理并且更容易支持多个数据源和分布式数据消费。
- 许多 Kafka 用户在由多个阶段组成的处理管道中处理数据，其中从 Kafka 主题中使用原始输入数据，然后聚合、丰富或以其他方式转换为新主题以供进一步使用或后续处理。例如，用于推荐新闻文章的处理管道可能会从 RSS 提要中抓取文章内容并将其发布到“文章”主题。进一步的处理可能会规范化或重复化该内容，并将清理后的文章内容发布到新主题，最后的处理阶段可能会尝试将该内容推荐给用户。这样的处理管道根据各个主题创建实时数据流图。
- 事件溯源是一种应用程序设计风格，其中状态变化被记录为按时间顺序排列的记录序列。 Kafka 对非常大的存储日志数据的支持使其成为以这种风格构建的应用程序的优秀后端。
- Kafka 可以作为分布式系统的一种外部提交日志。日志有助于在节点之间复制数据，并充当故障节点恢复其数据的重新同步机制。 Kafka 中的日志压缩功能有助于支持这种用例。

汇合

Confluent 平台是一个企业级平台，它为 Kafka 提供了先进的功能，旨在帮助加速应用程序开发和连接、通过流处理实现转换、简化企业大规模运营并满足严格的架构要求。 Confluent 由 Apache Kafka 的原始创建者构建，它通过企业级功能扩展了 Kafka 的优势，同时消除了 Kafka 管理或监控的负担。如今，财富 100 强企业中有超过 80% 都采用数据流技术，其中大多数都使用 Confluent。

为什么选择 **Confluent**?

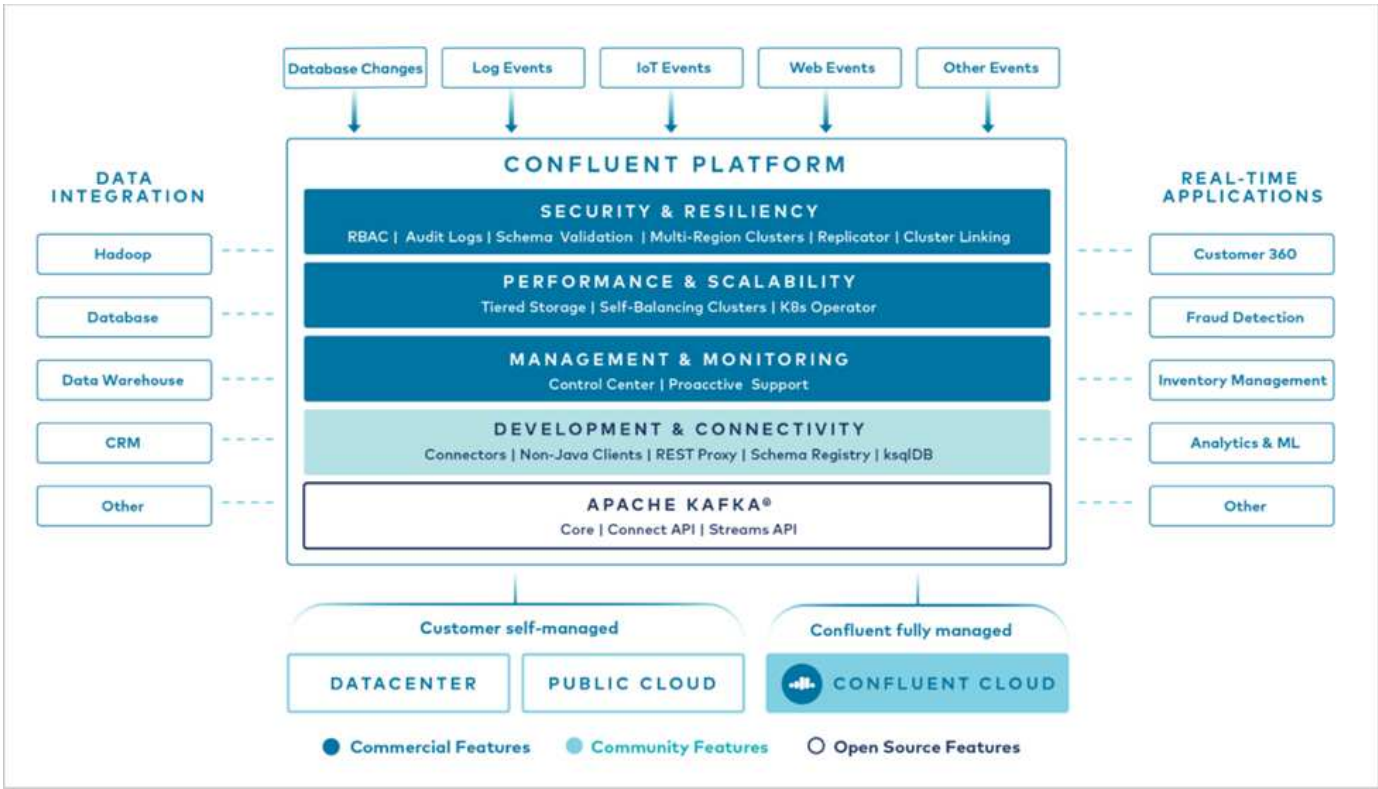
通过将历史数据和实时数据集成到单一的中央事实来源，Confluent 可以轻松构建全新类别的现代事件驱动应用

程序，获得通用数据管道，并解锁具有完全可扩展性、性能和可靠性的强大新用例。

Confluent 的用途是什么?

Confluent 平台让您专注于如何从数据中获取商业价值，而不必担心底层机制，例如如何在不同的系统之间传输或集成数据。具体来说，Confluent Platform 简化了数据源与 Kafka 的连接、流应用程序的构建以及 Kafka 基础设施的保护、监控和管理。如今，Confluent 平台已广泛应用于众多行业，从金融服务、全渠道零售、自动驾驶汽车到欺诈检测、微服务和物联网。

下图显示了 Confluent Kafka 平台组件。



Confluent 事件流技术概述

Confluent 平台的核心是 "阿帕奇卡夫卡"，最受欢迎的开源分布式流媒体平台。 Kafka 的主要功能如下：

- 发布和订阅记录流。
- 以容错的方式存储记录流。
- 处理记录流。

开箱即用的 Confluent Platform 还包括 Schema Registry、REST Proxy、总共 100 多个预构建的 Kafka 连接器和 ksqlDB。

Confluent 平台企业功能概述

- *汇合控制中心。*用于管理和监控 Kafka 的基于 GUI 的系统。它允许您轻松管理 Kafka Connect 以及创建、编辑和管理与其他系统的连接。
- 适用于 **Kubernetes** 的 **Confluent**。 Confluent for Kubernetes 是一个 Kubernetes 操作员。 Kubernetes 操作员通过为特定平台应用程序提供独特的功能和要求来扩展 Kubernetes 的编排功能。对于 Confluent 平台

，这包括大大简化 Kafka 在 Kubernetes 上的部署过程，并自动执行典型的基础设施生命周期任务。

- *汇合连接器至 Kafka。*连接器使用 Kafka Connect API 将 Kafka 连接到其他系统，例如数据库、键值存储、搜索索引和文件系统。 Confluent Hub 具有适用于最流行的数据源和接收器的可下载连接器，包括使用 Confluent Platform 对这些连接器进行全面测试和支持的版本。更多详情请见 "此处"。
- *自平衡集群。*提供自动负载平衡、故障检测和自我修复。它支持根据需要添加或停用代理，无需手动调整。
- *汇合簇连接。*直接将集群连接在一起，并通过链接桥将主题从一个集群镜像到另一个集群。集群链接简化了多数据中心、多集群和混合云部署的设置。
- *汇合自动数据平衡器。*监控集群中的代理数量、分区大小、分区数量以及集群内的领导者数量。它允许您转移数据以在整个集群中创建均匀的工作负载，同时限制重新平衡流量以最大限度地减少重新平衡时对生产工作负载的影响。
- *汇合复制器。*使得在多个数据中心维护多个 Kafka 集群变得比以往更加简单。
- *分层存储。*提供使用您最喜欢的云提供商存储大量 Kafka 数据的选项，从而减少运营负担和成本。通过分层存储，您可以将数据保存在经济高效的对象存储中，并且仅在需要更多计算资源时才扩展代理。
- Confluent JMS 客户端。 Confluent Platform 包含一个与 JMS 兼容的 Kafka 客户端。该 Kafka 客户端实现了 JMS 1.1 标准 API，使用 Kafka 代理作为后端。如果您有使用 JMS 的遗留应用程序并且想要用 Kafka 替换现有的 JMS 消息代理，这将非常有用。
- *Confluent MQTT 代理。*提供一种从 MQTT 设备和网关直接向 Kafka 发布数据的方法，无需中间的 MQTT 代理。
- **Confluent** 安全插件。 Confluent 安全插件用于为各种 Confluent 平台工具和产品添加安全功能。目前，有一个可用于 Confluent REST 代理的插件，可帮助验证传入的请求并将经过验证的主体传播到对 Kafka 的请求。这使得 Confluent REST 代理客户端能够利用 Kafka 代理的多租户安全功能。

汇合验证

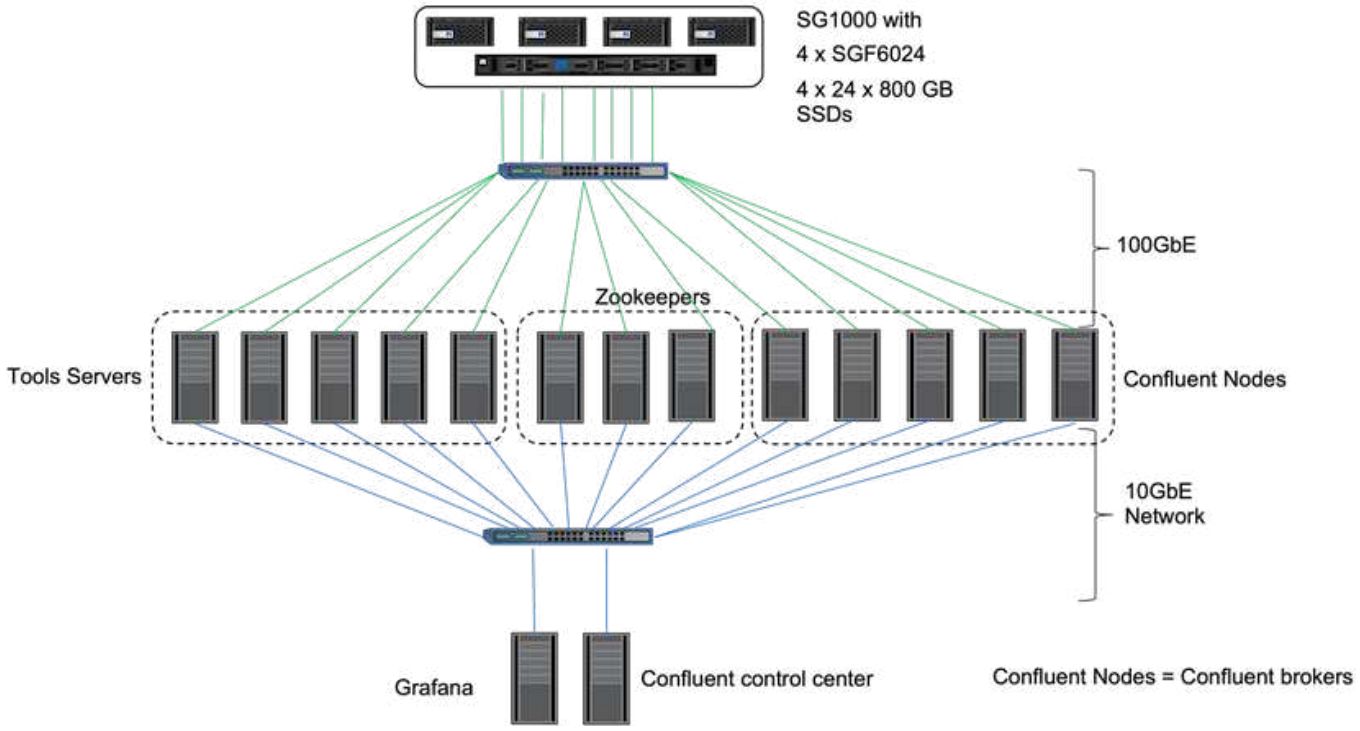
我们使用NetApp StorageGRID中的 Confluent Platform 6.2 Tiered Storage 进行了验证。NetApp和 Confluent 团队共同进行了此次验证，并运行了验证所需的测试用例。

Confluent 平台设置

我们使用以下设置进行验证。

为了验证，我们使用了三个 zookeeper、五个 broker、五个测试脚本执行服务器、命名工具服务器（配备 256GB RAM 和 16 个 CPU）。对于NetApp存储，我们使用了带有四个 SGF6024 的 SG1000 负载均衡器的StorageGRID 。存储和代理通过 100GbE 连接进行连接。

下图显示了用于 Confluent 验证的配置的网络拓扑。



工具服务器充当向 Confluent 节点发送请求的应用程序客户端。

Confluent 分层存储配置

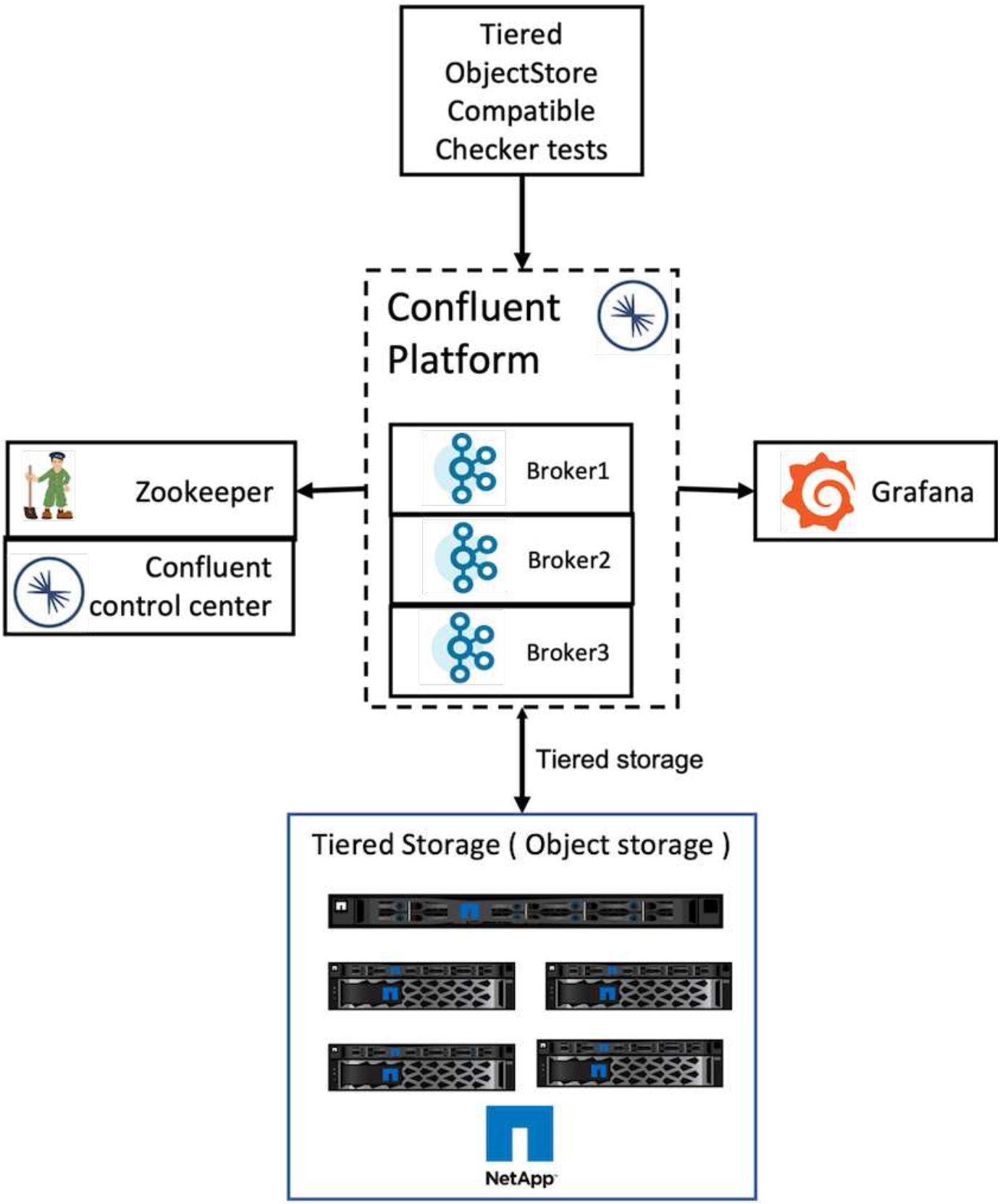
分层存储配置在Kafka中需要以下参数：

```
Confluent.tier.archiver.num.threads=16
confluent.tier.fetcher.num.threads=32
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkasgdbucket1-2
confluent.tier.s3.region=us-west-2
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://kafkasgd.rtpppe.netapp.com:
10444/
confluent.tier.s3.force.path.style.access=true
```

为了验证，我们使用了带有 HTTP 协议的StorageGRID ，但 HTTPS 也可以使用。访问密钥和密钥存储在 `confluent.tier.s3.cred.file.path`范围。

NetApp对象存储 - StorageGRID

我们在StorageGRID中配置了单站点配置以进行验证。



验证测试

我们完成了以下五个测试用例进行验证。这些测试在 Trogdor 框架上执行。前两个是功能测试，其余三个是性能测试。

对象存储正确性测试

此测试确定对象存储 API 上的所有基本操作（例如，获取/放置/删除）是否根据分层存储的需求正常运行。这是每个对象存储服务都应该在后续测试之前通过的基本测试。这是一个要么通过要么失败的断言测试。

分层功能正确性测试

该测试通过或失败的断言测试来确定端到端分层存储功能是否运行良好。该测试创建了一个测试主题，该主题默认配置为启用分层，并且热集大小大大减少。它为新创建的测试主题生成一个事件流，等待代理将段存档到对象存储，然后使用事件流并验证所消耗的流是否与生成的流匹配。向事件流生成的消息数量是可配置的，这使得用户可以根据测试的需要生成足够大的工作量。减少的热集大小可确保活动段之外的消费者提取仅从对象存储中提供；这有助于测试对象存储读取的正确性。我们已经在有和没有对象存储故障注入的情况下执行了此测试。我们通过停止StorageGRID中某个节点的服务管理器服务来模拟节点故障，并验证端到端功能是否与对象存储兼容。

层级获取基准

该测试验证了分层对象存储的读取性能，并检查了基准测试生成的段在高负载下的范围提取读取请求。在这个基准测试中，Confluent 开发了自定义客户端来满足层级获取请求。

生产-消费工作负载基准测试

该测试通过段归档间接生成对象存储上的写入工作负载。当消费者群体获取段时，从对象存储中生成读取工作负载（读取的段）。该工作负载由测试脚本生成。该测试检查了并行线程对对象存储的读写性能。我们对分层功能正确性测试进行了测试，测试了有和没有对象存储故障注入的情况。

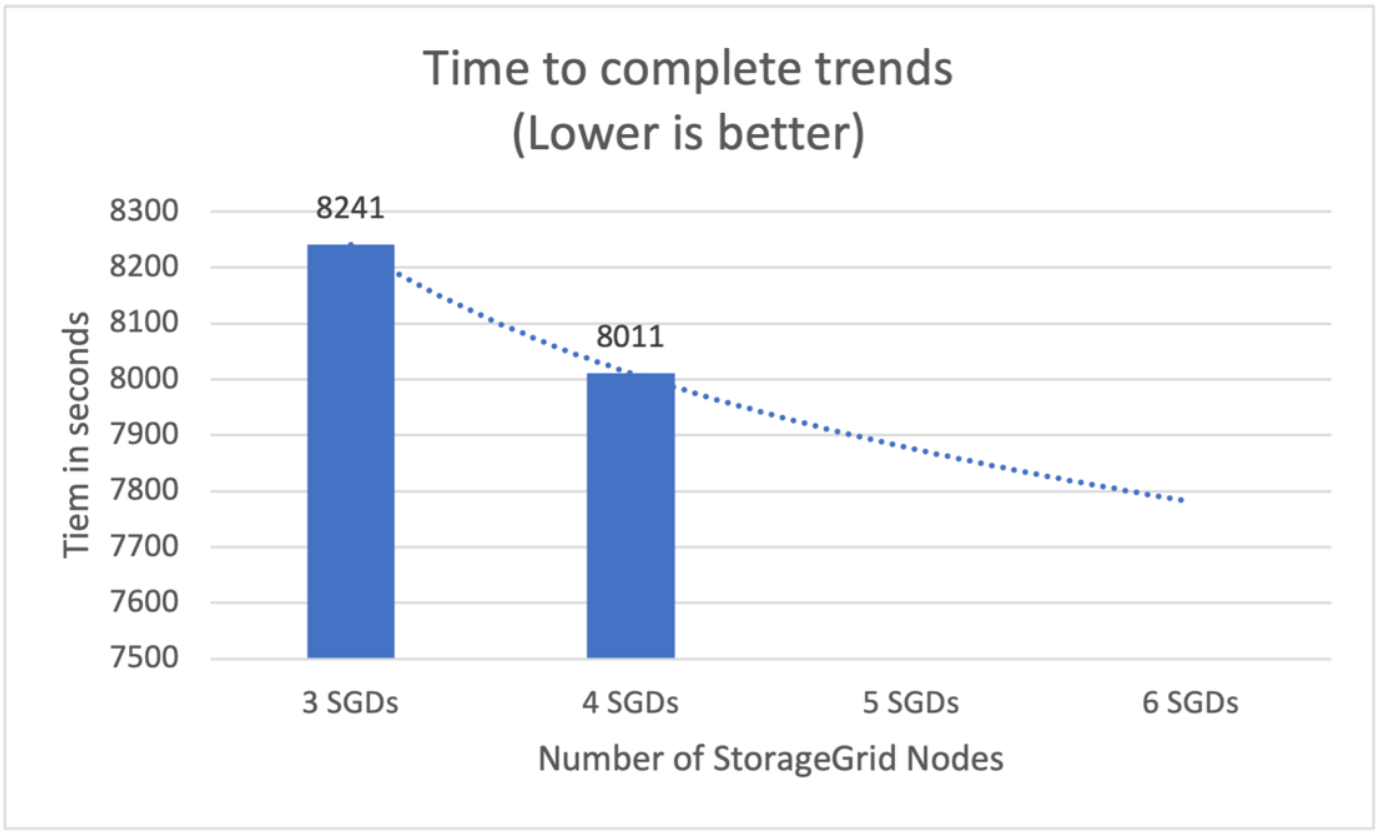
保留工作量基准

该测试检查了对象存储在繁重的主题保留工作负载下的删除性能。保留工作负载是使用测试脚本生成的，该脚本与测试主题并行生成许多消息。测试主题是使用基于大小和基于时间的激进保留设置进行配置，这会导致事件流不断从对象存储中清除。然后将这些片段存档。这导致代理在对象存储中执行大量删除操作，并收集对象存储删除操作的性能。

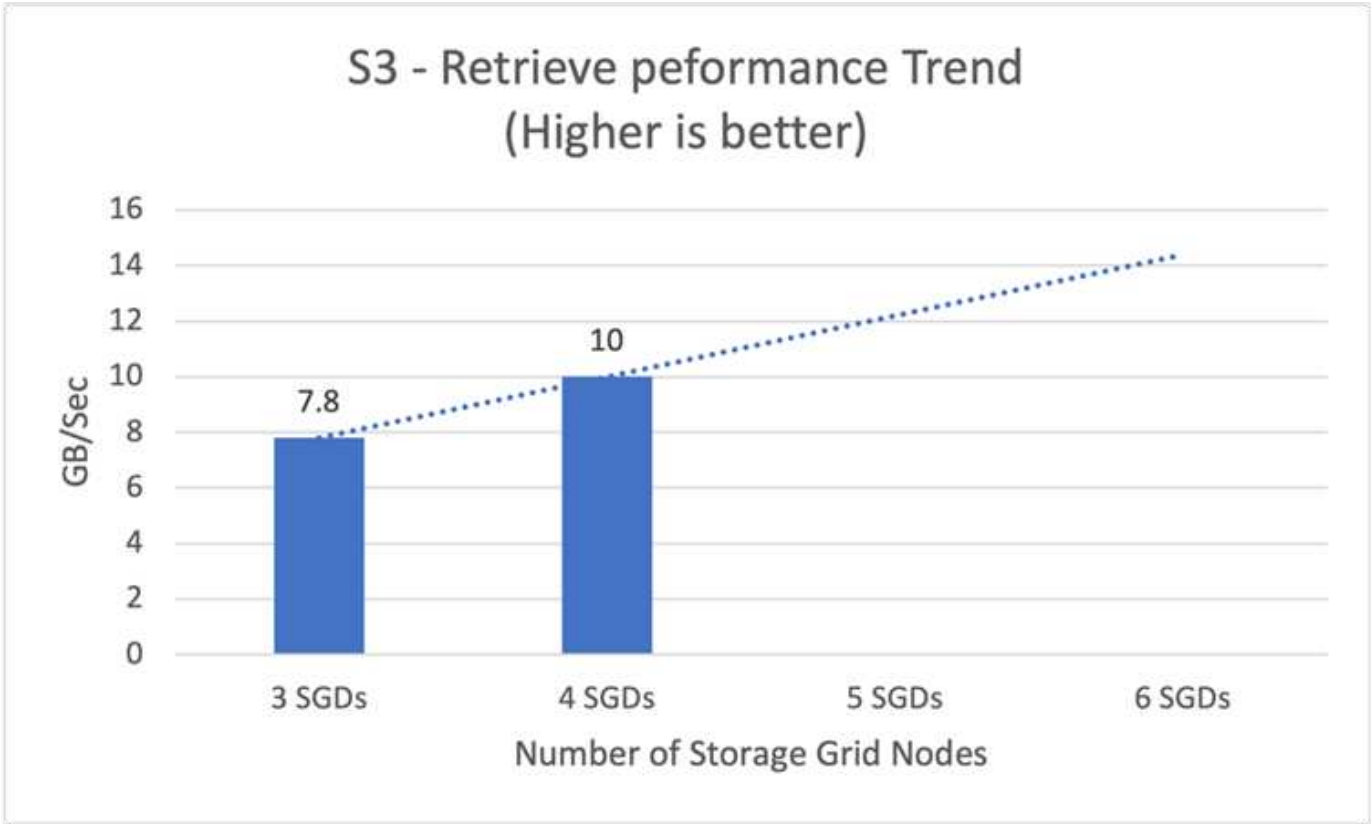
具有可扩展性的性能测试

我们使用NetApp StorageGRID设置对三到四个节点的生产者和消费者工作负载进行了分层存储测试。根据我们的测试，完成时间和性能结果与StorageGRID节点的数量成正比。StorageGRID设置至少需要三个节点。

- 当存储节点数量增加时，完成生产和消费操作的时间呈线性减少。



- s3 检索操作的性能根据StorageGRID节点的数量线性增加。 StorageGRID最多支持 200 个 StorgeGRID 节点。



Confluent S3连接器

Amazon S3 Sink 连接器将数据从 Apache Kafka 主题导出到 Avro、JSON 或 Bytes 格式的 S3 对象。 Amazon S3 接收器连接器定期从 Kafka 轮询数据，然后将其上传到 S3。分区器用于将每个 Kafka 分区的数据分成块。每个数据块都表示为一个 S3 对象。键名对主题、Kafka 分区和该数据块的起始偏移量进行编码。

在此设置中，我们向您展示如何使用 Kafka s3 接收器连接器直接从 Kafka 读取和写入对象存储中的主题。对于此测试，我们使用了独立的 Confluent 集群，但此设置适用于分布式集群。

1. 从 Confluent 网站下载 Confluent Kafka。
2. 将软件包解压到服务器上的文件夹中。
3. 导出两个变量。

```
Export CONFLUENT_HOME=/data/confluent/confluent-6.2.0
export PATH=$PATH:/data/confluent/confluent-6.2.0/bin
```

4. 对于独立的 Confluent Kafka 设置，集群会在 `/tmp`。它还创建 Zookeeper、Kafka、模式注册表、连接、ksql-server 和控制中心文件夹，并从复制它们各自的配置文件 `$CONFLUENT_HOME`。请参见以下示例：

```
root@stlrx2540m1-108:~# ls -ltr /tmp/confluent.406980/
total 28
drwxr-xr-x 4 root root 4096 Oct 29 19:01 zookeeper
drwxr-xr-x 4 root root 4096 Oct 29 19:37 kafka
drwxr-xr-x 4 root root 4096 Oct 29 19:40 schema-registry
drwxr-xr-x 4 root root 4096 Oct 29 19:45 kafka-rest
drwxr-xr-x 4 root root 4096 Oct 29 19:47 connect
drwxr-xr-x 4 root root 4096 Oct 29 19:48 ksql-server
drwxr-xr-x 4 root root 4096 Oct 29 19:53 control-center
root@stlrx2540m1-108:~#
```

5. 配置 Zookeeper。如果使用默认参数，则无需更改任何内容。

```
root@stlrx2540m1-108:~# cat
/tmp/confluent.406980/zookeeper/zookeeper.properties  | grep -iv ^#
dataDir=/tmp/confluent.406980/zookeeper/data
clientPort=2181
maxClientCnxns=0
admin.enableServer=false
tickTime=2000
initLimit=5
syncLimit=2
server.179=controlcenter:2888:3888
root@stlrx2540m1-108:~#
```

在上面的配置中，我们更新了 `server. xxx`财产。默认情况下，您需要三个 Zookeeper 来选择 Kafka 领导者。

6. 我们在 `/tmp/confluent.406980/zookeeper/data`具有唯一 ID：

```
root@stlrx2540m1-108:~# cat /tmp/confluent.406980/zookeeper/data/myid
179
root@stlrx2540m1-108:~#
```

我们将最后一个 IP 地址数字用于 myid 文件。我们对 Kafka、connect、control-center、Kafka、Kafka-rest、ksql-server 和 schema-registry 配置使用了默认值。

7. 启动 Kafka 服务。

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  start
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
ZooKeeper is [UP]
Kafka is [UP]
Schema Registry is [UP]
Kafka REST is [UP]
Connect is [UP]
ksqlDB Server is [UP]
Control Center is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

每个配置都有一个日志文件夹，有助于解决问题。在某些情况下，服务需要更多时间才能启动。确保所有服务均已启动并正在运行。

8. 使用以下方式安装 Kafka connect `confluent-hub` 。

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# ./confluent-
hub install confluentinc/kafka-connect-s3:latest
The component can be installed in any of the following Confluent
Platform installations:
  1. /data/confluent/confluent-6.2.0 (based on $CONFLUENT_HOME)
  2. /data/confluent/confluent-6.2.0 (where this tool is installed)
Choose one of these to continue the installation (1-2): 1
Do you want to install this into /data/confluent/confluent-
6.2.0/share/confluent-hub-components? (yN) y

Component's license:
Confluent Community License
http://www.confluent.io/confluent-community-license
I agree to the software license agreement (yN) y
Downloading component Kafka Connect S3 10.0.3, provided by Confluent,
Inc. from Confluent Hub and installing into /data/confluent/confluent-
6.2.0/share/confluent-hub-components
Do you want to uninstall existing version 10.0.3? (yN) y
Detected Worker's configs:
  1. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  2. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  3. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-distributed.properties
  4. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-standalone.properties
  5. Based on CONFLUENT_CURRENT:
/tmp/confluent.406980/connect/connect.properties
  6. Used by Connect process with PID 15904:
/tmp/confluent.406980/connect/connect.properties
Do you want to update all detected configs? (yN) y
Adding installation directory to plugin path in the following files:
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
standalone.properties
  /tmp/confluent.406980/connect/connect.properties
  /tmp/confluent.406980/connect/connect.properties
```

```
Completed
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

您还可以使用以下方式安装特定版本 `confluent-hub install confluentinc/kafka-connect-s3:10.0.3`。

9. 默认情况下， `confluentinc-kafka-connect-s3`安装在 `/data/confluent/confluent-6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3`。
10. 使用新的 `confluentinc-kafka-connect-s3`。

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/etc/kafka/connect-distributed.properties | grep plugin.path
#
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/co
nnectors,
plugin.path=/usr/share/java,/data/zookeeper/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3
root@stlrx2540m1-108:~#
```

11. 停止 Confluent 服务并重新启动它们。

```
confluent local services  stop
confluent local services  start
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  status
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
Connect is [UP]
Control Center is [UP]
Kafka is [UP]
Kafka REST is [UP]
ksqlDB Server is [UP]
Schema Registry is [UP]
ZooKeeper is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

12. 在 `/root/.aws/credentials`文件。

```
root@stlrx2540m1-108:~# cat /root/.aws/credentials
[default]
aws_access_key_id = xxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxx
root@stlrx2540m1-108:~#
```

13. 验证存储桶是否可访问。

```
root@stlrx2540m4-01:~# aws s3 –endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls kafkasgdbucket1-2
2021-10-29 21:04:18       1388 1
2021-10-29 21:04:20       1388 2
2021-10-29 21:04:22       1388 3
root@stlrx2540m4-01:~#
```

14. 为 s3 和 bucket 配置配置 s3-sink 属性文件。

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-
s3/etc/quickstart-s3.properties | grep -v ^#
name=s3-sink
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=1
topics=s3_testtopic
s3.region=us-west-2
s3.bucket.name=kafkasgdbucket1-2
store.url=http://kafkasgd.rtpppe.netapp.com:10444/
s3.part.size=5242880
flush.size=3
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.avro.AvroFormat
partitioner.class=io.confluent.connect.storage.partitioner.DefaultPartit
ioner
schema.compatibility=NONE
root@stlrx2540m1-108:~#
```

15. 将一些记录导入到 s3 存储桶。

```
kafka-avro-console-producer --broker-list localhost:9092 --topic
s3_topic \
--property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1",
"type":"string"}]}'
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
{"f1": "value4"}
{"f1": "value5"}
{"f1": "value6"}
{"f1": "value7"}
{"f1": "value8"}
{"f1": "value9"}
```

16. 加载 s3-sink 连接器。

```
root@stlrx2540m1-108:~# confluent local services connect connector load
s3-sink  --config /data/confluent/confluent-6.2.0/share/confluent-hub-
components/confluentinc-kafka-connect-s3/etc/quickstart-s3.properties
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "flush.size": "3",
    "format.class": "io.confluent.connect.s3.format.avro.AvroFormat",
    "partitioner.class":
"io.confluent.connect.storage.partitioner.DefaultPartitioner",
    "s3.bucket.name": "kafkasgdbucket1-2",
    "s3.part.size": "5242880",
    "s3.region": "us-west-2",
    "schema.compatibility": "NONE",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "store.url": "http://kafkasgd.rtpppe.netapp.com:10444/",
    "tasks.max": "1",
    "topics": "s3_testtopic",
    "name": "s3-sink"
  },
  "tasks": [],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

17. 检查 s3-sink 状态。

```
root@stlrx2540m1-108:~# confluent local services connect connector
status s3-sink
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "10.63.150.185:8083"
  },
  "tasks": [
    {
      "id": 0,
      "state": "RUNNING",
      "worker_id": "10.63.150.185:8083"
    }
  ],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

18. 检查日志以确保 s3-sink 已准备好接受主题。

```
root@stlrx2540m1-108:~# confluent local services connect log
```

19. 检查 Kafka 中的主题。

```
kafka-topics --list --bootstrap-server localhost:9092
…
connect-configs
connect-offsets
connect-statuses
default_ksql_processing_log
s3_testtopic
s3_topic
s3_topic_new
root@stlrx2540m1-108:~#
```

20. 检查 s3 存储桶中的对象。

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls --recursive kafkasgdbucket1-
2/topics/
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000003.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000006.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000009.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000012.avro
2021-10-29 21:24:09        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000015.avro
root@stlrx2540m1-108:~#
```

21. 要验证内容，请运行以下命令将每个文件从 S3 复制到本地文件系统：

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 cp s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
tes.avro
download: s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro to
./tes.avro
root@stlrx2540m1-108:~#
```

22. 要打印记录，请使用 avro-tools-1.11.0.1.jar（可在 "Apache 档案"）。

```
root@stlrx2540m1-108:~# java -jar /usr/src/avro-tools-1.11.0.1.jar
tojson tes.avro
21/10/30 00:20:24 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
{"f1":"value1"}
{"f1":"value2"}
{"f1":"value3"}
root@stlrx2540m1-108:~#
```

融合自平衡集群

如果您以前管理过 Kafka 集群，那么您可能熟悉手动将分区重新分配给不同代理以确保整个集群的工作负载平衡所带来的挑战。对于部署大量 Kafka 的组织来说，重新整理大量数据可能是一项艰巨、繁琐且有风险的任务，尤其是在集群之上构建关键任务应用程序时。然而，即使对于最小的 Kafka 用例，该过程也很耗时并且容易出现人为错误。

在我们的实验室中，我们测试了 Confluent 自平衡集群功能，该功能可以根据集群拓扑变化或不均匀负载自动重新平衡。 Confluent 重新平衡测试有助于测量当节点发生故障或扩展节点需要在代理之间重新平衡数据时添加新代理的时间。在经典的 Kafka 配置中，需要重新平衡的数据量会随着集群的增长而增长，但是在分层存储中，重新平衡仅限于少量数据。根据我们的验证，在经典的 Kafka 架构中，分层存储中的重新平衡需要几秒或几分钟，并且随着集群的增长而线性增长。

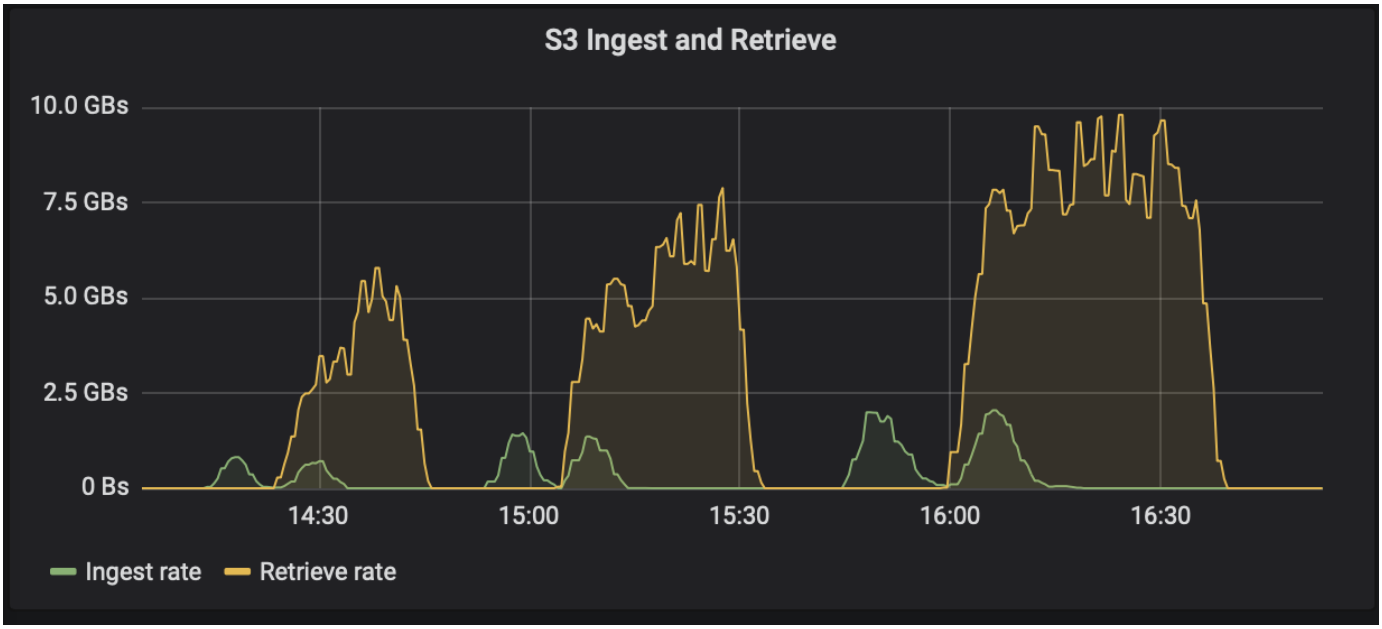
在自平衡集群中，分区重新平衡完全自动化，以优化 Kafka 的吞吐量，加速代理扩展，并减少运行大型集群的运营负担。在稳定状态下，自平衡集群监控代理之间的数据偏差，并不断重新分配分区以优化集群性能。当扩大或缩小平台规模时，自平衡集群会自动识别新代理的存在或旧代理的删除，并触发后续分区重新分配。这使您能够轻松地添加和停用代理，从而使您的 Kafka 集群从根本上更加有弹性。这些好处不需要任何人工干预、复杂的数学运算或分区重新分配通常带来的人为错误风险。因此，数据重新平衡可以在更短的时间内完成，您可以自由地专注于更高价值的事件流项目，而不需要不断监督您的集群。

最佳实践指南

本节介绍了从此次认证中获得的经验教训。

- 根据我们的验证，S3 对象存储最适合 Confluent 保存数据。
- 我们可以使用高吞吐量 SAN（特别是 FC）来保存代理热数据或本地磁盘，因为在 Confluent 分层存储配置中，代理数据目录中保存的数据大小取决于数据移动到对象存储时的段大小和保留时间。
- 当segment.bytes较高时，对象存储提供更好的性能；我们测试了512MB。
- 在 Kafka 中，主题中生成的每个记录的键或值的长度（以字节为单位）由 `length.key.value`范围。对于StorageGRID，S3 对象提取和检索性能提升到更高的值。例如，512 字节提供 5.8GBps 的检索，1024 字节提供 7.5GBps 的 s3 检索，而 2048 字节提供接近 10GBps 的检索。

下图展示了基于 `length.key.value`。



- Kafka 调优。为了提高分层存储的性能，可以增加 TierFetcherNumThreads 和 TierArchiverNumThreads。作为一般准则，您需要增加 TierFetcherNumThreads 以匹配物理 CPU 核心的数量，并将 TierArchiverNumThreads 增加到 CPU 核心数量的一半。例如，在服务器属性中，如果您有一台具有八个物理核心的机器，请设置 confluent.tier.fetcher.num.threads = 8 和 confluent.tier.archiver.num.threads = 4。
- *主题删除的时间间隔。*当主题被删除时，对象存储中的日志段文件的删除不会立即开始。相反，在删除这些文件之前有一个默认值为 3 小时的时间间隔。您可以修改配置 confluent.tier.topic.delete.check.interval.ms 来更改此间隔的值。如果删除主题或集群，您还可以手动删除相应存储桶中的对象。
- *分层存储内部主题的 ACL。*对于内部部署，建议的最佳实践是在用于分层存储的内部主题上启用 ACL 授权器。设置 ACL 规则以限制只有代理用户才能访问此数据。这可以保护内部主题并防止未经授权访问分层存储数据和元数据。

```
kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-
configs.conf \
--add --allow-principal User:<kafka> --operation All --topic "_confluent-
tier-state"
```

替换用户 `<kafka>`与您部署中的实际代理主体一起。

例如，命令 `confluent-tier-state`设置内部主题的 ACL 以进行分层存储。目前，只有一个与分层存储相关的内部主题。该示例创建了一个 ACL，为内部主题上的所有操作提供主要的 Kafka 权限。

浆纱

Kafka 大小调整可以通过四种配置模式进行：简单、粒度、反向和分区。

简单的

简单模式适合首次使用 Apache Kafka 的用户或早期使用案例。对于此模式，您可以提供吞吐量 MBps、读取扇

出、保留和资源利用率百分比（默认值为 60%）等要求。您还可以进入环境，例如本地（裸机、VMware、Kubernetes 或 OpenStack）或云。根据这些信息，Kafka 集群的大小提供了代理、zookeeper、Apache Kafka 连接工作器、模式注册表、REST 代理、ksqlDB 和 Confluent 控制中心所需的服务器数量。

对于分层存储，请考虑使用粒度配置模式来确定 Kafka 集群的大小。粒度模式适合经验丰富的 Apache Kafka 用户或定义明确的用例。本节介绍生产者、流处理器和消费者的大小。

生产者

要描述 Apache Kafka 的生产者（例如本机客户端、REST 代理或 Kafka 连接器），请提供以下信息：

- *姓名。*火花。
- *生产者类型。*应用程序或服务、代理（REST、MQTT、其他）和现有数据库（RDBMS、NOSQL、其他）。您也可以选择“我不知道”。
- *平均吞吐量。*以每秒事件数计算（例如 1,000,000）。
- *峰值吞吐量。*以每秒事件数计算（例如 4,000,000）。
- *平均消息大小。*以字节为单位，未压缩（最大 1MB；例如 1000）。
- *消息格式。*选项包括 Avro、JSON、协议缓冲区、二进制、文本、“我不知道”和其他。
- *复制因子。*选项为 1、2、3（Confluent 建议）、4、5 或 6。
- *保留时间。*有一天（例如）。您希望将数据存储在 Apache Kafka 中多长时间？输入 -1 和任意单位可表示无限时间。计算器假设无限保留的保留时间为 10 年。
- 选中“启用分层存储以减少代理数量并允许无限存储？”复选框。
- 启用分层存储后，保留字段控制在代理本地存储的热数据集。档案保留字段控制数据在档案对象存储中的存储时间。
- *档案存储保留。*一年（例如）。您希望将数据保存在档案存储中多长时间？输入 -1 和任意单位可获得无限持续时间。计算器假设无限保留的保留时间为 10 年。
- 增长乘数。 1（例如）。如果此参数的值基于当前吞吐量，则将其设置为 1。要根据额外增长确定大小，请将此参数设置为增长乘数。
- 生产者实例的数量。 10（例如）。将运行多少个生产者实例？此输入需要将 CPU 负载纳入到尺寸计算中。空白值表示 CPU 负载未纳入计算。

根据此示例输入，尺寸对生产者有以下影响：

- 未压缩字节的平均吞吐量：1GBps。未压缩字节的峰值吞吐量：4GBps。压缩字节的平均吞吐量：400MBps。压缩字节的峰值吞吐量：1.6GBps。这是基于默认的 60% 压缩率（您可以更改此值）。
  - 所需的代理热集存储总量：31,104TB，包括复制和压缩。所需的总代理外存档存储：378,432TB（压缩）。使用"https://fusion.netapp.com"用于StorageGRID大小调整。

流处理器必须描述从 Apache Kafka 使用数据并返回到 Apache Kafka 的应用程序或服务。大多数情况下，这些都是在 ksqlDB 或 Kafka Streams 中构建的。

- *姓名。*火花飘带。
- *处理时间。*该处理器处理一条消息需要多长时间？
  - 1 毫秒（简单、无状态转换）[示例]，10 毫秒（有状态的内存操作）。

- 100ms（有状态网络或磁盘操作），1000ms（第三方 REST 调用）。
- 我已经对这个参数进行了基准测试，并且确切地知道需要多长时间。

- 输出保留。 1天（示例）。流处理器将其输出返回给 Apache Kafka。您希望这些输出数据在 Apache Kafka 中存储多长时间？输入 -1 和任意单位可获得无限持续时间。
- 选中复选框“启用分层存储以减少代理数量并允许无限存储？”
- 档案存储保留。 1年（例如）。您希望将数据保存在档案存储中多长时间？输入 -1 和任意单位可获得无限持续时间。计算器假设无限保留的保留时间为 10 年。
- 输出直通百分比。 100（例如）。流处理器将其输出返回给 Apache Kafka。入站吞吐量的百分比将输出回 Apache Kafka？例如，如果入站吞吐量为 20MBps，且该值为 10，则输出吞吐量将为 2MBps。
- 这是从哪些应用程序读取的？选择“Spark”，这是基于生产者类型的大小调整中使用的名称。根据以上输入，您可以预期流处理器实例和主题分区估计的大小会产生以下影响：
- 该流处理器应用程序需要以下数量的实例。传入的主题可能也需要这么多的分区。联系 Confluent 确认此参数。
  - 1,000 表示平均吞吐量，无增长乘数
  - 峰值吞吐量为 4,000，无增长乘数
  - 1,000 表示平均吞吐量，并带有增长乘数
  - 峰值吞吐量为 4,000，并带有增长乘数

消费者

描述使用来自 Apache Kafka 的数据但不返回到 Apache Kafka 的应用程序或服务；例如，本机客户端或 Kafka 连接器。

- *姓名。*激发消费者。
- *处理时间。*该消费者需要多长时间来处理一条消息？
  - 1ms（例如，像日志记录这样的简单且无状态的任务）
  - 10ms（快速写入数据存储）
  - 100 毫秒（数据存储写入速度慢）
  - 1000ms（第三方 REST 调用）
  - 一些其他已知持续时间的基准测试过程。
- *消费者类型。*应用程序、代理或接收器至现有数据存储（RDBMS、NoSQL 等）。
- 这是从哪些应用程序读取的？将此参数与之前确定的生产者和流大小连接起来。

根据以上输入，您必须确定消费者实例的大小和主题分区估计。消费者应用程序需要以下数量的实例。

- 平均吞吐量为 2,000，无增长乘数
- 峰值吞吐量为 8,000，无增长乘数
- 平均吞吐量为 2,000，包括增长乘数
- 峰值吞吐量为 8,000，包括增长乘数

传入的主题可能也需要这个数量的分区。联系 Confluent 进行确认。

除了对生产者、流处理器和消费者的要求之外，您还必须提供以下额外要求：

- *重建时间。*例如4小时。如果 Apache Kafka 代理主机发生故障，其数据丢失，并且需要配置新主机来替换故障主机，那么这个新主机必须多快重建自身？如果值未知，请将此参数留空。
- *资源利用率目标（百分比）。*例如，60。您希望您的主机在平均吞吐量期间的利用率如何？ Confluent 建议利用率为 60%，除非您使用 Confluent 自平衡集群，在这种情况下利用率可能会更高。

描述你的环境

- *您的集群将在什么环境中运行？*亚马逊网络服务、微软 Azure、谷歌云平台、本地裸机、本地 VMware、本地 OpenStack 还是本地 Kubernates?
- *主人详细信息。*核心数：例如48个，网卡类型（10GbE、40GbE、16GbE、1GbE或其他类型）。
- *存储卷。*主持人：12（例如）。每个主机支持多少个硬盘或 SSD？ Confluent 建议每个主机配备 12 个硬盘。
- 存储容量/卷（以 **GB** 为单位）。 1000（例如）。单个卷可以存储多少 GB 的存储空间？ Confluent 建议使用 1TB 磁盘。
- 存储配置。存储卷如何配置？ Confluent 建议使用 RAID10 来充分利用 Confluent 的所有功能。还支持 JBOD、SAN、RAID 1、RAID 0、RAID 5 和其他类型。
- 单卷吞吐量（**MBps**）。 125（例如）。单个存储卷每秒的读取或写入速度是多少兆字节？ Confluent 推荐使用标准硬盘，其吞吐量通常为 125MBps。
- 内存容量（**GB**）。 64（例如）。

确定环境变量后，选择“Size my Cluster”。根据上面指出的示例参数，我们确定了 Confluent Kafka 的以下大小：

- *Apache Kafka。*经纪人数量：22。您的集群受存储限制。考虑启用分层存储以减少主机数量并允许无限存储。
- Apache ZooKeeper。数量：5；Apache Kafka Connect Workers：数量：2；Schema Registry：数量：2；REST Proxy：数量：2；ksqlDB：数量：2；Confluent Control Center：数量：1。

对于平台团队，请使用反向模式，无需考虑用例。使用分区模式来计算单个主题需要多少个分区。看 https://eventsizer.io根据反向和分区模式进行大小调整。

结束语

本文档提供了将 Confluent 分层存储与NetApp存储结合使用的最佳实践指南，包括验证测试、分层存储性能结果、调整、Confluent S3 连接器和自平衡功能。考虑到 ILM 策略、经过多项性能测试验证的 Confluent 性能以及行业标准的 S3 API， NetApp StorageGRID对象存储是 Confluent 分层存储的最佳选择。

在哪里可以找到更多信息

要了解有关本文档中描述的信息的更多信息，请查看以下文档和/或网站：

- 什么是 Apache Kafka

  "https://www.confluent.io/what-is-apache-kafka/"

- NetApp产品文档

  "https://www.netapp.com/support-and-training/documentation/"

- S3-sink 参数详情

  "https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options"

- 阿帕奇卡夫卡

  "https://en.wikipedia.org/wiki/Apache_Kafka"

- Confluent 平台中的无限存储

  "https://www.confluent.io/blog/infinite-kafka-storage-in-confluent-platform/"

- Confluent 分层存储 - 最佳实践和规模

  "https://docs.confluent.io/platform/current/kafka/tiered-storage.html#best-practices-and-recommendations"

- Confluent 平台的 Amazon S3 接收器连接器

  "https://docs.confluent.io/kafka-connect-s3-sink/current/overview.html"

- Kafka 大小

  "https://eventsizer.io"

- StorageGRID大小调整

  "https://fusion.netapp.com/"

- Kafka 用例

  "https://kafka.apache.org/uses"

- Confluence 平台 6.0 中的自平衡 Kafka 集群

  "https://www.confluent.io/blog/self-balancing-kafka-clusters-in-confluent-platform-6-0/"

  "https://www.confluent.io/blog/confluent-platform-6-0-delivers-the-most-powerful-event-streaming-platform-to-date/"