Confluent Kafka 的最佳實踐

NetApp artificial intelligence solutions

NetApp
February 12, 2026

目錄

Confluent Kafka 的最佳實踐

TR-4912： NetApp Confluent Kafka 分層儲存最佳實務指南

Karthikeyan Nagalingam、Joseph Kandatilparambil、 NetApp Rankesh Kumar、Confluence

Apache Kafka 是一個社群分散式事件流平台，每天都能夠處理數兆個事件。 Kafka 最初被認為是一個訊息佇列，基於分散式提交日誌的抽象化。自 2011 年由 LinkedIn 創建並開源以來，Kafka 已經從一個訊息佇列發展成為一個成熟的事件流平台。 Confluent 透過 Confluent 平台提供 Apache Kafka 的分發。 Confluent 平台為 Kafka 提供了額外的社群和商業功能，旨在增強大規模生產中營運商和開發人員的串流體驗。

本文檔透過提供以下內容描述了在 NetApp 物件儲存產品上使用 Confluent 分層儲存的最佳實踐指南：

- 使用NetApp物件儲存進行匯合驗證 – NetApp StorageGRID
- 分層儲存效能測試
- NetApp儲存系統上 Confluent 的最佳實務指南

為什麼選擇 **Confluent** 分層儲存？

Confluent 已成為許多應用程式的預設即時串流平台，尤其是對於大數據、分析和串流媒體工作負載。分層儲存使用戶能夠在 Confluent 平台中將運算與儲存分開。它使儲存資料更具成本效益，使您能夠儲存幾乎無限量的資料並按需擴大（或縮小）工作負載，並使資料和租戶重新平衡等管理任務更加容易。 S3 相容儲存系統可以利用所有這些功能，將所有事件集中在一個地方，實現資料民主化，因此無需複雜的資料工程。有關為什麼應該為 Kafka 使用分層存儲的更多信息，請查看"本文由 Confluent 撰寫"。

NetApp instaclustr 從 3.8.1 版本開始也支援 Kafka 分層儲存。請點擊此處查看更多詳情 "使用 Kafka 分層儲存的 Instaclust"

為什麼選擇**NetApp StorageGRID**進行分層儲存？

StorageGRID是NetApp推出的業界領先的物件儲存平台。 StorageGRID是一種軟體定義的基於物件的儲存解決方案，支援業界標準物件 API，包括 Amazon Simple Storage Service (S3) API。 StorageGRID大規模儲存和管理非結構化數據，以提供安全、持久的物件儲存。內容被放置在正確的位置、正確的時間和正確的儲存層，從而優化工作流程並降低全球分佈的富媒體的成本。

StorageGRID最大的差異在於其資訊生命週期管理 (ILM) 策略引擎，它支援策略驅動的資料生命週期管理。策略引擎可以使用元資料來管理資料在其整個生命週期內的儲存方式，以便最初優化效能，並隨著資料老化自動優化成本和耐用性。

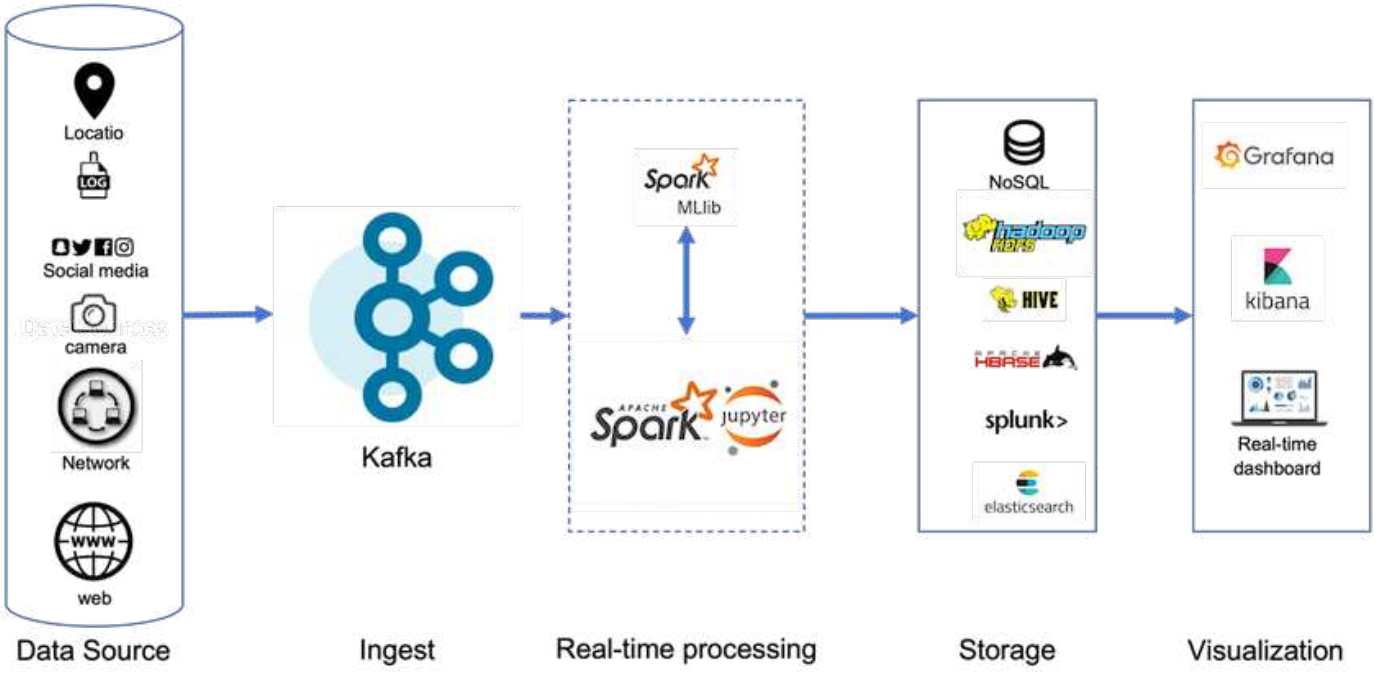
啟用 **Confluent** 分層存儲

分層儲存的基本概念是將資料儲存任務與資料處理任務分開。透過這種分離，資料儲存層和資料處理層可以更輕鬆地獨立擴展。

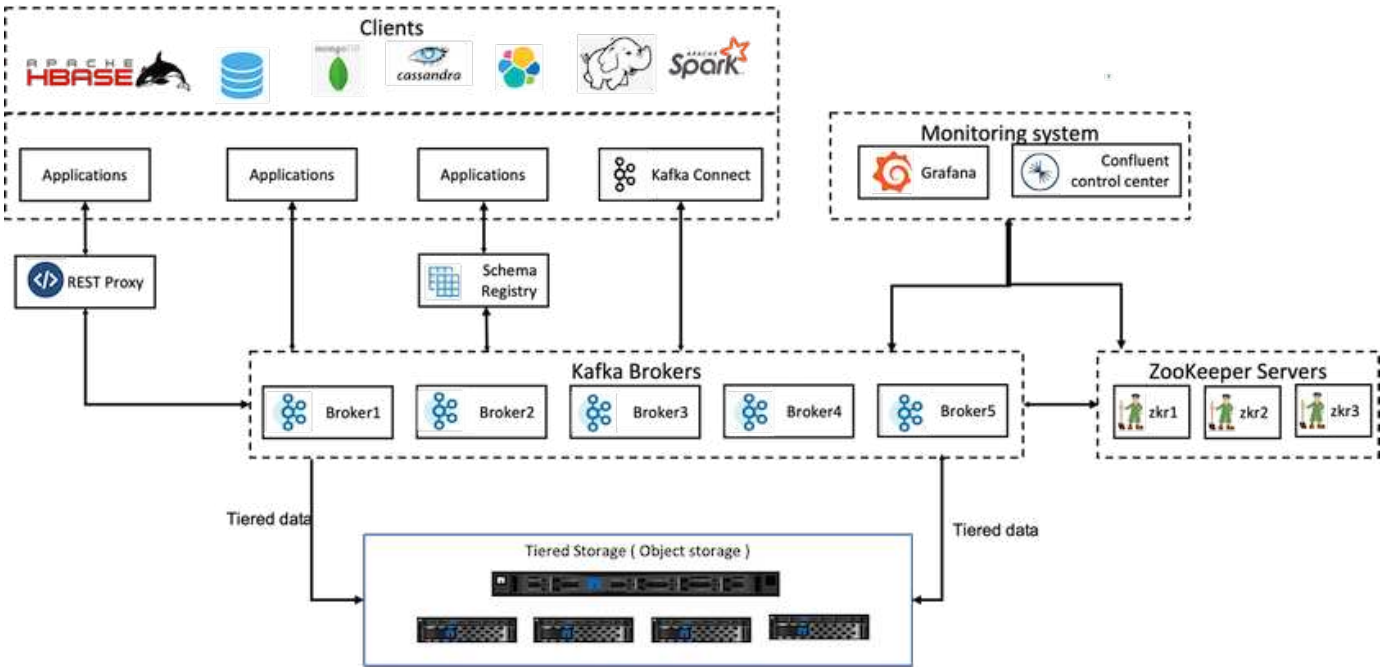
Confluent 的分層儲存解決方案必須處理兩個因素。首先，它必須解決或避免常見的物件儲存一致性和可用性屬性，例如 LIST 操作中的不一致和偶爾的物件不可用。其次，它必須正確處理分層儲存與 Kafka 的複製和容錯模型之間的交互，包括殭屍領導者繼續分層偏移範圍的可能性。 NetApp物件儲存提供一致的物件可用性和 HA 模

型，使分層儲存可用於層偏移範圍。 NetApp物件儲存提供一致的物件可用性和 HA 模型，使分層儲存可用於層偏移範圍。

透過分層存儲，您可以使用高效能平台在串流資料尾部附近進行低延遲讀寫，還可以使用更便宜、可擴展的物件儲存（如NetApp StorageGRID ）進行高吞吐量歷史讀取。我們也為具有 netapp 儲存控制器的 Spark 提供了技術解決方案，詳細資訊請見此處。下圖顯示了 Kafka 如何融入即時分析管道。



下圖描述了NetApp StorageGRID如何作為 Confluent Kafka 的物件儲存層。



解決方案架構細節

本節介紹用於 Confluent 驗證的硬體和軟體。此資訊適用於使用NetApp儲存的 Confluent

Platform 部署。下表涵蓋了經過測試的解決方案架構和基本組件。

解決方案組件	細節
Confluent Kafka 版本 6.2	• 三位動物園管理員 • 五台經紀商伺服器 • 五款工具伺服器 • 一個 Grafana • 一個控制中心
Linux（Ubuntu 18.04）	所有伺服器
NetApp StorageGRID用於分層存儲	• StorageGRID軟體 • 1 x SG1000（負載平衡器） • 4 個 SGF6024 • 4 x 24 x 800 SSD • S3 協議 • 4 x 100GbE（代理程式和StorageGRID實例之間的網路連線）
15台富士通PRIMERGY RX2540伺服器	每個配備：* 2 個 CPU，共 16 個實體核心 * Intel Xeon * 256GB 實體記憶體 * 100GbE 雙端口

技術概述

本節介紹此解決方案所使用的技術。

NetAppStorageGRID

NetApp StorageGRID是一個高效能、經濟高效的物件儲存平台。透過使用分層存儲，Confluent Kafka 上儲存在本機儲存或代理程式的 SAN 儲存中的大部分資料都被卸載到遠端物件儲存。此配置可減少重新平衡、擴展或收縮群集或更換故障代理的時間和成本，從而顯著改善操作。物件儲存在管理駐留在物件儲存層的資料方面發揮著重要作用，因此選擇正確的物件儲存非常重要。

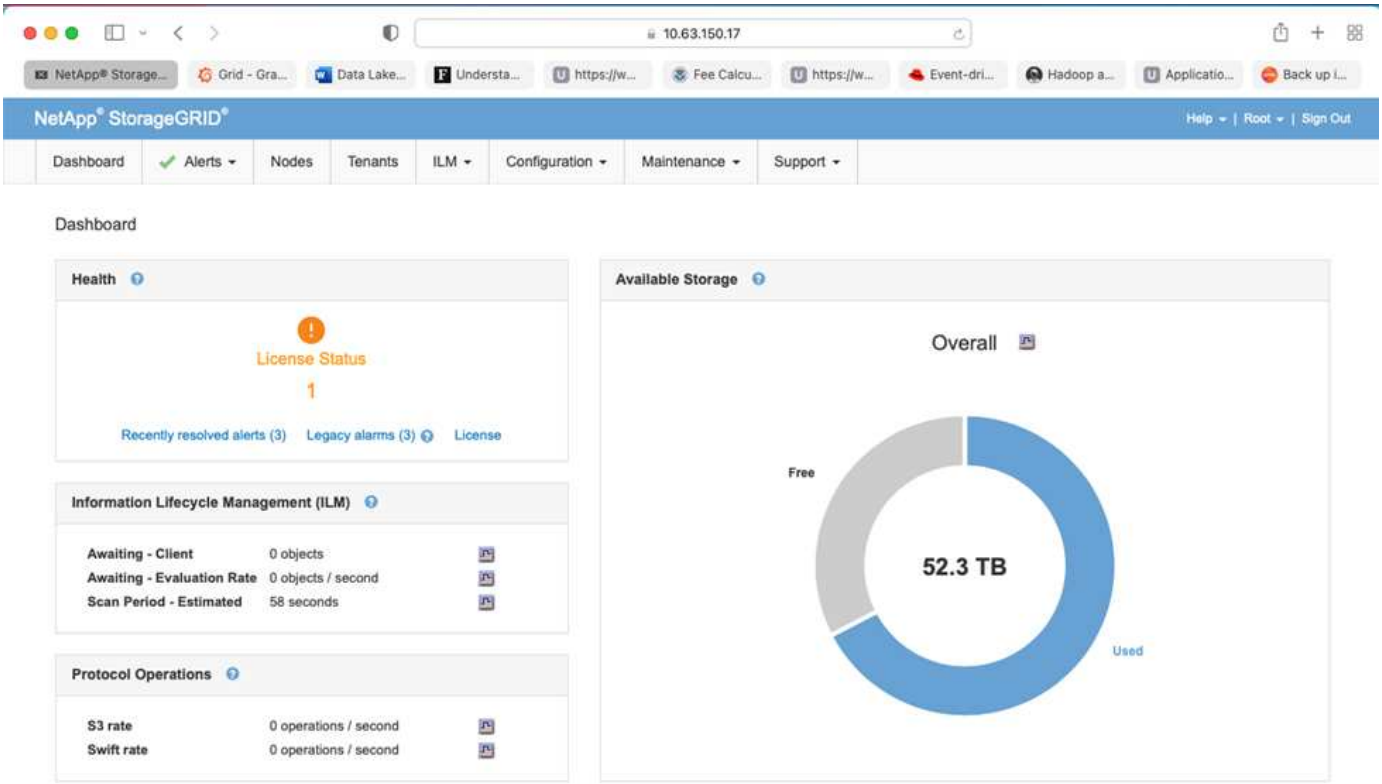
StorageGRID使用分散式、基於節點的網格架構提供智慧、策略驅動的全域資料管理。它透過其無所不在的全域物件命名空間與複雜的資料管理功能結合，簡化了 PB 級非結構化資料和數十億個物件的管理。單次呼叫物件存取可跨站點擴展，並簡化高可用性架構，同時確保持續的物件訪問，無論站點或基礎設施是否中斷。

多租戶允許在同一網格內安全地為多個非結構化雲端和企業資料應用程式提供服務，從而提高NetApp StorageGRID的投資報酬率和用例。您可以使用元資料驅動的物件生命週期策略建立多個服務級別，從而優化跨多個地區的耐用性、保護性、效能和位置性。使用者可以調整資料管理策略並監控和應用流量限制，以便在不斷變化的 IT 環境中隨著需求的變化而無中斷地重新調整資料格局。

使用網格管理器進行簡單管理

StorageGRID Grid Manager 是一個基於瀏覽器的圖形介面，可讓您在單一玻璃窗格中設定、管理和監控全球分

散位置的StorageGRID系統。



您可以使用StorageGRID Grid Manager 介面執行下列任務：

- 管理全球分佈的 PB 級物件儲存庫，例如影像、影片和記錄。
- 監控網格節點和服務以確保物件可用性。
- 使用資訊生命週期管理 (ILM) 規則來管理物件資料隨時間推移的放置。這些規則控制著物件資料被攝取後會發生什麼、如何防止資料遺失、物件資料儲存在何處以及儲存多長時間。
- 監控系統內的交易、效能和操作。

資訊生命週期管理政策

StorageGRID具有靈活的資料管理策略，包括保留對象的副本以及使用 EC（擦除編碼）方案（例如 2+1 和 4+2 等）來儲存對象，具體取決於特定的效能和資料保護要求。由於工作負載和需求隨時間而變化，ILM 策略通常也必須隨時間而變化。修改 ILM 策略是一項核心功能，可讓StorageGRID客戶快速輕鬆地適應不斷變化的環境。

表現

StorageGRID透過增加更多儲存節點來擴展效能，這些節點可以是虛擬機器、裸機或專用設備，例如"SG5712、SG5760、SG6060 或 SGF6024"。在我們的測試中，我們使用 SGF6024 設備以最小尺寸的三節點網格超越了 Apache Kafka 的關鍵效能要求。當客戶使用額外的代理來擴展他們的 Kafka 叢集時，他們可以添加更多的儲存節點來提高效能和容量。

負載平衡器和端點配置

StorageGRID中的管理節點提供網格管理器 UI（使用者介面）和 REST API 端點來檢視、設定和管理您的StorageGRID系統，以及稽核日誌來追蹤系統活動。為了為 Confluent Kafka 分層儲存提供高可用性 S3 端點，我們實作了StorageGRID負載平衡器，它作為管理節點和網關節點上的服務運作。此外，負載平衡器還管理本

地流量並與 GSLB（全域伺服器負載平衡）對話以協助進行災難復原。

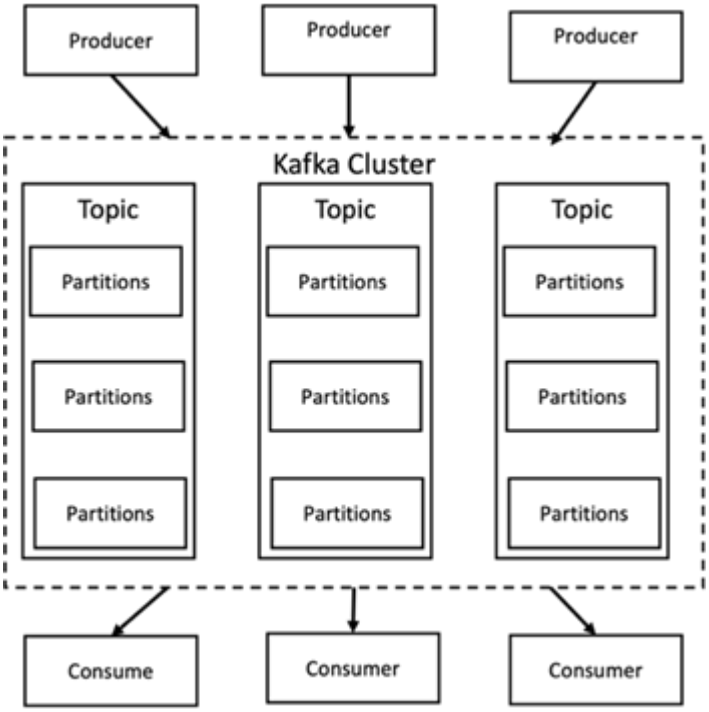
為了進一步增強端點配置， StorageGRID提供了內建於管理節點的流量分類策略，讓您監控工作負載流量，並對您的工作負載套用各種服務品質 (QoS) 限制。流量分類策略應用於網關節點和管理節點的StorageGRID負載平衡器服務上的端點。這些策略可以幫助流量整形和監控。

StorageGRID中的流量分類

StorageGRID內建 QoS 功能。流量分類策略可以幫助監控來自客戶端應用程式的不同類型的 S3 流量。然後，您可以建立並套用策略來根據輸入/輸出頻寬、讀取/寫入並發請求數或讀取/寫入請求率限制此流量。

阿帕契卡夫卡

Apache Kafka 是一個用 Java 和 Scala 編寫的使用流處理的軟體匯流排的框架實作。其目的是提供一個統一、高吞吐量、低延遲的平台來處理即時資料饋送。 Kafka可以透過Kafka Connect連接外部系統進行資料匯出和匯入，並提供Java流處理庫Kafka Streams。 Kafka 使用針對效率進行了最佳化的二進位、基於 TCP 的協議，並依賴「訊息集」抽象，該抽象自然地將訊息組合在一起，以減少網路往返的開銷。這使得更大的順序磁碟操作、更大的網路封包和連續的記憶體區塊成為可能，從而使 Kafka 能夠將突發的隨機訊息寫入流轉換為線性寫入。下圖描述了Apache Kafka的基本資料流。



Kafka 儲存來自任意數量的進程（稱為生產者）的鍵值訊息。資料可以劃分到不同主題內的不同分區。在分區內，訊息嚴格按照其偏移量（訊息在分區內的位置）排序，並與時間戳一起索引和儲存。其他稱為消費者的進程可以從分區讀取訊息。對於流程處理，Kafka 提供了 Streams API，允許編寫 Java 應用程式使用來自 Kafka 的資料並將結果寫回 Kafka。 Apache Kafka 還可以與外部串流處理系統（如 Apache Apex、Apache Flink、Apache Spark、Apache Storm 和 Apache NiFi）協同工作。

Kafka 在一個或多個伺服器（稱為代理）的叢集上運行，所有主題的分區分佈在叢集節點上。此外，分區被複製到多個代理程式。這種架構使得 Kafka 能夠以容錯的方式傳遞大量訊息流，並使其能夠取代一些傳統的訊息傳遞系統，如 Java 訊息服務 (JMS)、高階訊息佇列協定 (AMQP) 等。自 0.11.0.0 版本以來，Kafka 提供事務寫入功能，可使用 Streams API 提供一次串流處理。

Kafka 支援兩種類型的主題：常規主題和壓縮主題。常規主題可以配置保留時間或空間界限。如果有記錄超過指

定的保留時間或超出了分割區的空間限制，則允許 Kafka 刪除舊資料以釋放儲存空間。預設情況下，主題的保留時間配置為 7 天，但也可以無限期地儲存資料。對於壓縮主題，記錄不會根據時間或空間界限而過期。相反，Kafka 將後續訊息視為具有相同金鑰的舊訊息的更新，並保證永遠不會刪除每個金鑰的最新訊息。使用者可以透過寫入具有特定鍵的空值的所謂墓碑訊息來徹底刪除訊息。

Kafka 中有五個主要 API：

- 生產者 API。允許應用程式發布記錄流。
- *消費者 API。 *允許應用程式訂閱主題並處理記錄流。
- 連接器 API。執行可重複使用的生產者和消費者 API，將主題連結到現有應用程式。
- 流 API。此 API 將輸入流轉換為輸出並產生結果。
- *管理 API。 *用於管理 Kafka 主題、代理人和其他 Kafka 物件。

消費者和生產者 API 建立在 Kafka 訊息傳遞協定之上，並為 Java 中的 Kafka 消費者和生產者用戶端提供參考實作。底層訊息傳遞協定是一種二進位協議，開發人員可以使用任何程式語言編寫自己的消費者或生產者用戶端。這使得 Kafka 從 Java 虛擬機器 (JVM) 生態系統中解放出來。 Apache Kafka wiki 中維護了可用的非 Java 用戶端清單。

Apache Kafka 用例

Apache Kafka 最受歡迎的用途是訊息傳遞、網站活動追蹤、指標、日誌聚合、流處理、事件來源和提交日誌記錄。

- Kafka 提高了吞吐量、內建分區、複製和容錯能力，使其成為大規模訊息處理應用程式的良好解決方案。
- Kafka 可以將追蹤管道中的使用者活動（頁面瀏覽量、搜尋量）重建為一組即時發布-訂閱源。
- Kafka 經常用於營運監控數據。這涉及匯總來自分散式應用程式的統計資料以產生集中的操作資料。
- 許多人使用 Kafka 來取代日誌聚合解決方案。日誌聚合通常從伺服器上收集實體日誌檔案並將它們放在一個中心位置（例如，檔案伺服器或 HDFS）進行處理。 Kafka 抽象檔案詳細資料並將日誌或事件資料以訊息流的形式提供更清晰的抽象。這允許更低延遲的處理並且更容易支援多個資料來源和分散式資料消費。
- 許多 Kafka 使用者在由多個階段組成的處理管道中處理數據，其中從 Kafka 主題中使用原始輸入數據，然後聚合、豐富或以其他方式轉換為新主題以供進一步使用或後續處理。例如，用於推薦新聞文章的處理管道可能會從 RSS 提要中抓取文章內容並將其發佈到「文章」主題。進一步的處理可能會規範化或重複化該內容，並將清理後的文章內容發佈到新主題，最後的處理階段可能會嘗試將該內容推薦給使用者。這樣的處理管道根據各個主題創建即時資料流程圖。
- 事件溯源是一種應用程式設計風格，其中狀態變化被記錄為按時間順序排列的記錄序列。 Kafka 對非常大的儲存日誌資料的支援使其成為以這種風格構建的應用程式的優秀後端。
- Kafka 可以作為分散式系統的一種外部提交日誌。日誌有助於在節點之間複製數據，並充當故障節點恢復其數據的重新同步機制。 Kafka 中的日誌壓縮功能有助於支援這種用例。

匯合

Confluent 平台是一個企業級平台，它為 Kafka 提供了先進的功能，旨在幫助加速應用程式開發和連接、透過串流處理實現轉換、簡化企業大規模營運並滿足嚴格的架構要求。 Confluent 由 Apache Kafka 的原始創建者構建，它透過企業級功能擴展了 Kafka 的優勢，同時消除了 Kafka 管理或監控的負擔。如今，財富 100 強企業中有超過 80% 都採用資料流技術，其中大多數都使用 Confluent。

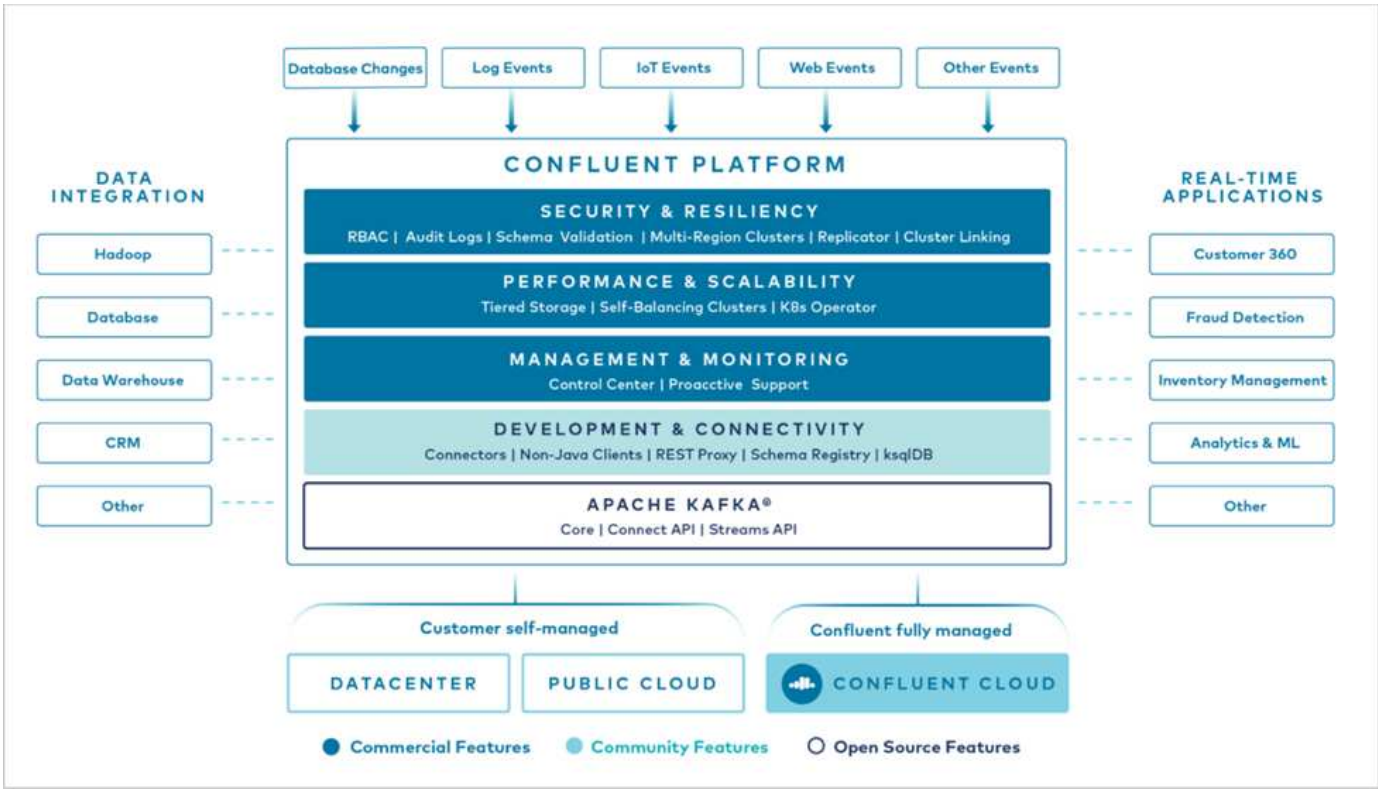
為什麼選擇 **Confluent**？

透過將歷史資料和即時資料整合到單一的中央事實來源，Confluent 可以輕鬆建立全新類別的現代事件驅動應用程序，獲得通用資料管道，並解鎖具有完全可擴展性、效能和可靠性的強大新用例。

Confluent 的用途是什麼？

Confluent 平台讓您專注於如何從資料中獲得商業價值，而不必擔心底層機制，例如如何在不同的系統之間傳輸或整合資料。具體來說，Confluent Platform 簡化了資料來源與 Kafka 的連接、串流應用程式的建置以及 Kafka 基礎設施的保護、監控和管理。如今，Confluent 平台已廣泛應用於眾多產業，從金融服務、全通路零售、自動駕駛汽車到詐欺偵測、微服務和物聯網。

下圖顯示了 Confluent Kafka 平台元件。



Confluent 事件流技術概述

Confluent 平台的核心是 "阿帕契卡夫卡"，最受歡迎的開源分散式串流平台。 Kafka 的主要功能如下：

- 發布和訂閱記錄流。
- 以容錯的方式儲存記錄流。
- 處理記錄流程。

開箱即用的 Confluent Platform 還包括 Schema Registry、REST Proxy、總共 100 多個預先建置的 Kafka 連接器和 ksqlDB。

Confluent 平台企業功能概述

- *匯合控制中心。 *用於管理和監控 Kafka 的基於 GUI 的系統。它允許您輕鬆管理 Kafka Connect 以及建

立、編輯和管理與其他系統的連接。

- *適用於 Kubernetes 的 Confluent。 * Confluent for Kubernetes 是一位 Kubernetes 操作員。 Kubernetes 操作員透過為特定平台應用程式提供獨特的功能和要求來擴展 Kubernetes 的編排功能。對於 Confluent 平台，這包括大幅簡化 Kafka 在 Kubernetes 上的部署流程，並自動執行典型的基礎架構生命週期任務。
- *將連接器匯合至 Kafka。 *連接器使用 Kafka Connect API 將 Kafka 連接到其他系統，例如資料庫、鍵值儲存、搜尋索引和檔案系統。 Confluent Hub 具有適用於最受歡迎的資料來源和接收器的可下載連接器，包括使用 Confluent Platform 對這些連接器進行全面測試和支援的版本。更多詳情請見 "這裡"。
- *自平衡集群。 *提供自動負載平衡、故障偵測和自我修復。它支援根據需要新增或停用代理，無需手動調整。
- *匯合簇連接。 *直接將集群連接在一起，並透過連結橋將主題從一個集群鏡像到另一個集群。集群連結簡化了多資料中心、多集群和混合雲部署的設定。
- *匯合自動數據平衡器。 *監控叢集中的代理數量、分區大小、分區數量以及叢集內的領導者數量。它允許您轉移資料以在整個叢集中創建均勻的工作負載，同時限制重新平衡流量以最大限度地減少重新平衡時對生產工作負載的影響。
- *匯合複製器。 *讓在多個資料中心維護多個 Kafka 叢集變得比以往更加簡單。
- *分層儲存。 *提供使用您最喜歡的雲端供應商儲存大量 Kafka 資料的選項，從而減少營運負擔和成本。透過分層存儲，您可以將資料保存在經濟高效的物件儲存中，並且僅在需要更多運算資源時才擴展代理程式。
- Confluent JMS 用戶端。 Confluent Platform 包含一個與 JMS 相容的 Kafka 用戶端。此 Kafka 用戶端實作了 JMS 1.1 標準 API，使用 Kafka 代理作為後端。如果您有使用 JMS 的遺留應用程式並且想要用 Kafka 取代現有的 JMS 訊息代理，這將非常有用。
- *Confluent MQTT 代理。 *提供一種從 MQTT 設備和網關直接向 Kafka 發布資料的方法，無需中間的 MQTT 代理。
- *Confluent 安全插件。 * Confluent 安全外掛程式用於為各種 Confluent 平台工具和產品添加安全功能。目前，有一個可用於 Confluent REST 代理程式的插件，可協助驗證傳入的請求並將經過驗證的主體傳播到對 Kafka 的請求。這使得 Confluent REST 代理客戶端能夠利用 Kafka 代理的多租戶安全功能。

匯合驗證

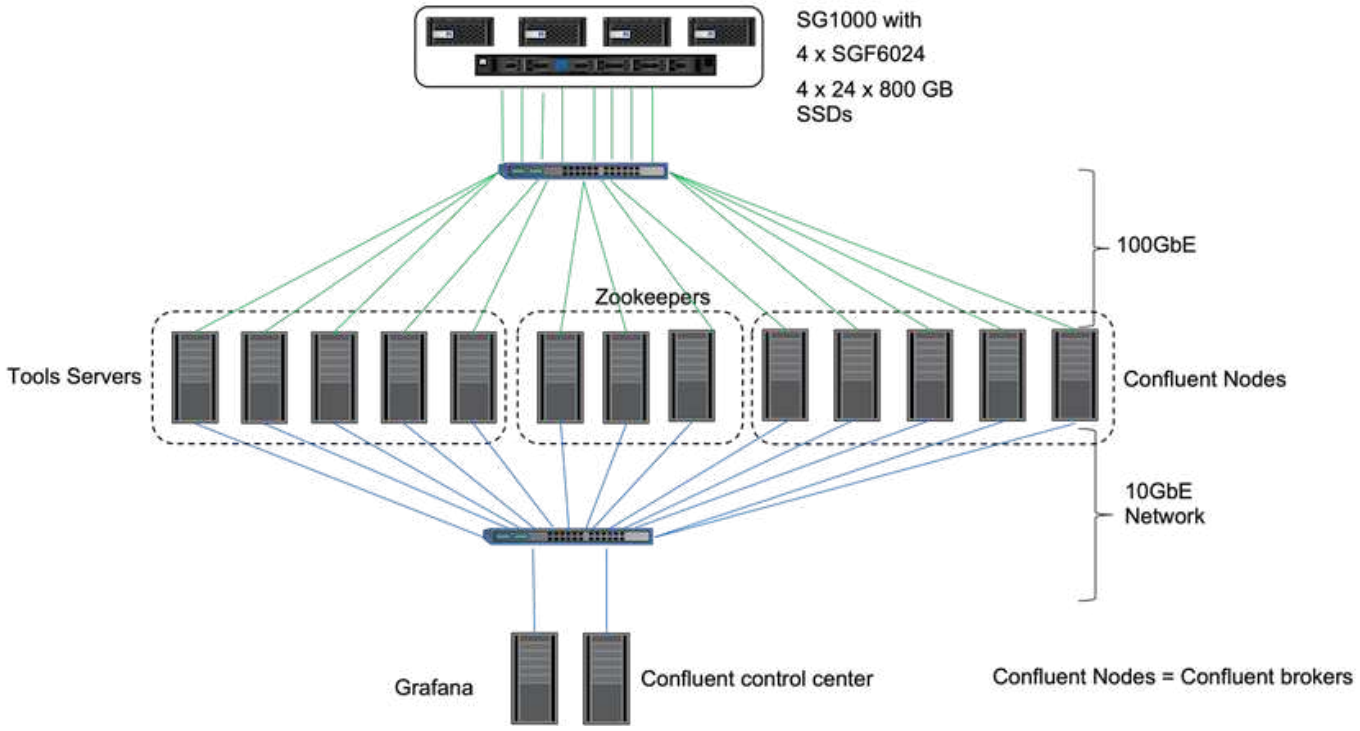
我們使用NetApp StorageGRID中的 Confluent Platform 6.2 Tiered Storage 進行了驗證。NetApp和 Confluent 團隊共同進行了此次驗證，並運行了驗證所需的測試案例。

Confluent 平台設置

我們使用以下設定進行驗證。

為了驗證，我們使用了三個 zookeeper、五個 broker、五個測試腳本執行伺服器、命名工具伺服器（配備 256GB RAM 和 16 個 CPU）。對於NetApp存儲，我們使用了具有四個 SGF6024 的 SG1000 負載平衡器的StorageGRID 。儲存和代理程式透過 100GbE 連線進行連線。

下圖顯示了用於 Confluent 驗證的配置的網路拓撲。



工具伺服器充當向 Confluent 節點發送請求的應用程式用戶端。

Confluent 分層儲存配置

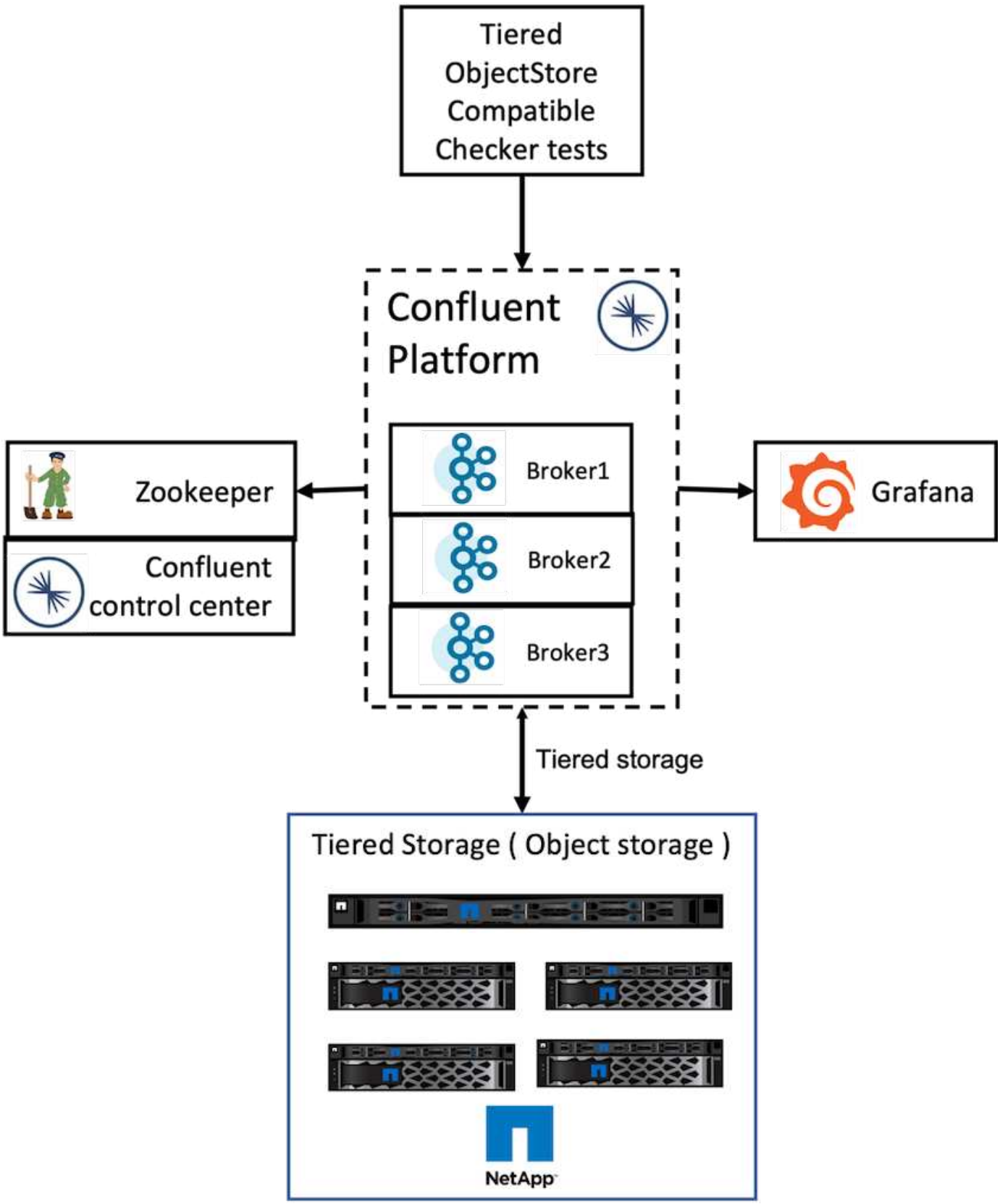
分層儲存配置在Kafka中需要以下參數：

```
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
```

為了驗證，我們使用了具有 HTTP 協定的StorageGRID ，但 HTTPS 也可以使用。存取密鑰和密鑰儲存在 `confluent.tier.s3.cred.file.path`範圍。

NetApp物件儲存 - StorageGRID

我們在StorageGRID中配置了單一站點配置以進行驗證。



驗證測試

我們完成了以下五個測試案例進行驗證。這些測試在 Trogdor 框架上執行。前兩個是功能測試，其餘三個是效能測試。

物件儲存正確性測試

此測試確定物件儲存 API 上的所有基本操作（例如，取得/放置/刪除）是否根據分層儲存的需求正常運作。這是每個物件儲存服務都應該在後續測試之前通過的基本測試。這是一個要么通過要么失敗的斷言測試。

分層功能正確性測試

此測試通過或失敗的斷言測試來確定端到端分層儲存功能是否運作良好。該測試創建了一個測試主題，該主題預設配置為啟用分層，並且熱集大小大大減少。它為新建立的測試主題產生一個事件流，等待代理將段存檔到物件存儲，然後使用事件流並驗證所消耗的流是否與產生的流相符。向事件流產生的訊息數量是可配置的，這使得使用者可以根據測試的需要產生足夠大的工作量。減少的熱集大小可確保活動段之外的消費者提取僅從物件儲存中提供；這有助於測試物件儲存讀取的正確性。我們已經在有和沒有物件儲存故障注入的情況下執行了此測試。我們透過停止StorageGRID中某個節點的服務管理器服務來模擬節點故障，並驗證端對端功能是否與物件儲存相容。

層級獲取基準

此測試驗證了分層物件儲存的讀取效能，並檢查了基準測試產生的段在高負載下的範圍提取讀取請求。在這個基準測試中，Confluent 開發了自訂客戶端來滿足層級取得請求。

生產-消費性工作負載基準測試

此測試透過段歸檔間接產生物件儲存上的寫入工作負載。當消費者群體取得段時，從物件儲存產生讀取工作負載（讀取的段）。此工作負載由測試腳本產生。該測試檢查了並行執行緒對物件儲存的讀寫效能。我們對分層功能正確性測試進行了測試，測試了有和沒有物件儲存故障注入的情況。

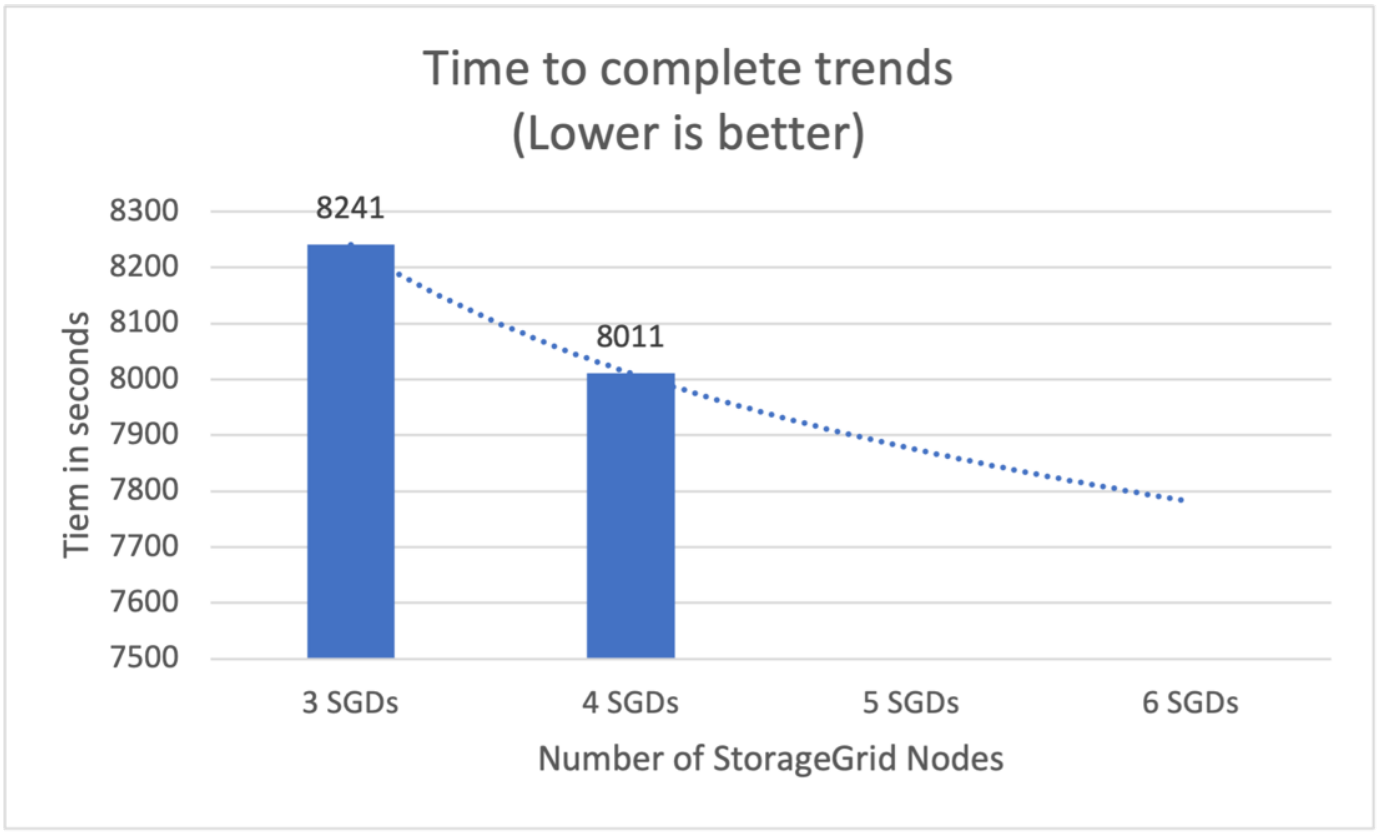
保留工作量基準

此測試檢查了物件儲存在繁重的主題保留工作負載下的刪除效能。保留工作負載是使用測試腳本產生的，該腳本與測試主題並行產生許多訊息。測試主題是使用基於大小和基於時間的激進保留設定進行配置，這會導致事件流不斷從物件儲存中清除。然後將這些片段存檔。這導致代理在物件儲存中執行大量刪除操作，並收集物件儲存刪除操作的效能。

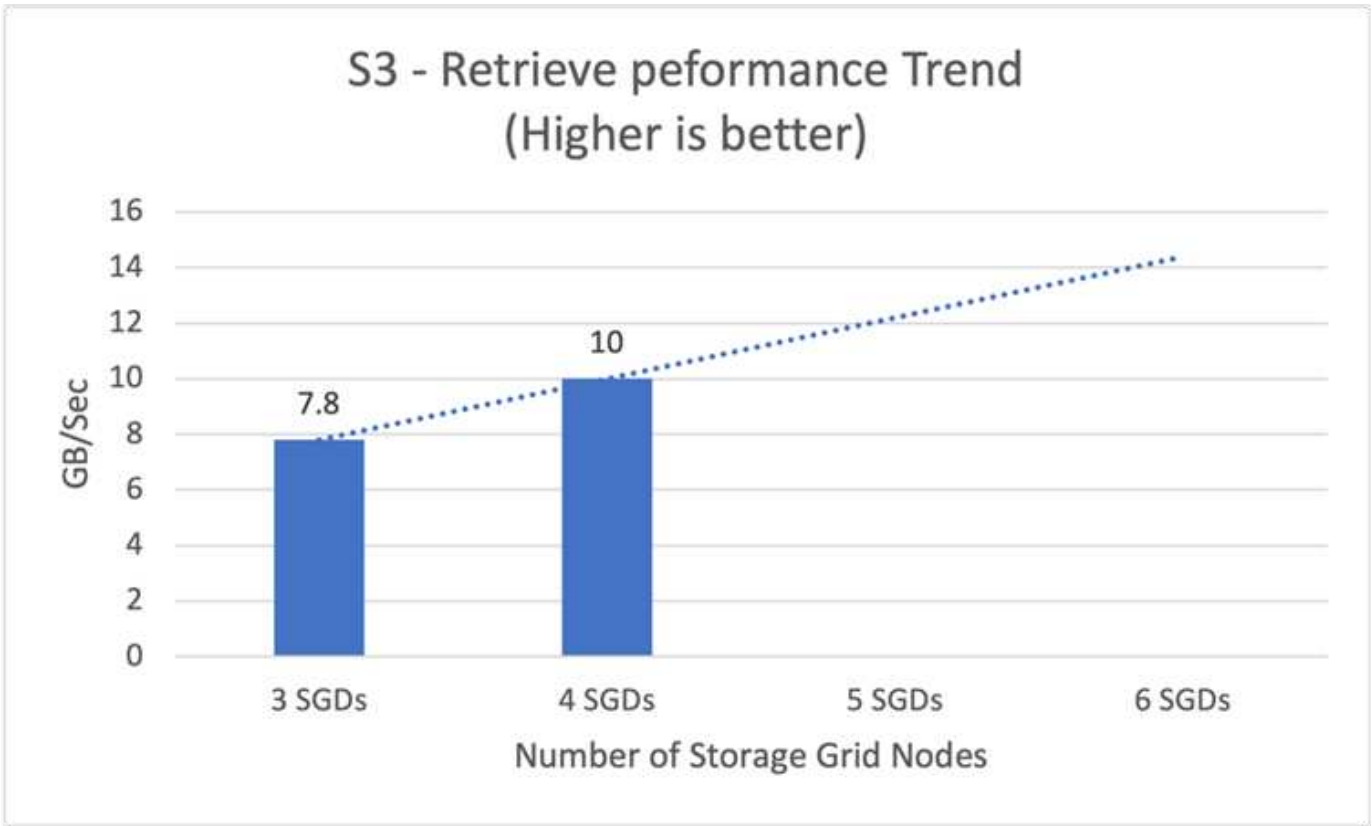
具有可擴展性的效能測試

我們使用NetApp StorageGRID設定對三到四個節點的生產者和消費者工作負載進行了分層儲存測試。根據我們的測試，完成時間和效能結果與StorageGRID節點的數量成正比。StorageGRID設定至少需要三個節點。

- 當儲存節點數量增加時，完成生產和消費作業的時間呈線性減少。



- s3 檢索操作的效能根據StorageGRID節點的數量線性增加。 StorageGRID支援最多 200 個 StorgeGRID 節點。



Confluent S3連接器

Amazon S3 Sink 連接器將資料從 Apache Kafka 主題匯出到 Avro、JSON 或 Bytes 格式的 S3 物件。 Amazon S3 接收器連接器定期從 Kafka 輪詢數據，然後將其上傳到 S3。分區器用於將每個 Kafka 分區的資料分成區塊。每個資料塊都表示為一個 S3 物件。鍵名會對主題、Kafka 分區和該資料區塊的起始偏移量進行編碼。

在此設定中，我們向您展示如何使用 Kafka s3 接收器連接器直接從 Kafka 讀取和寫入物件儲存中的主題。對於此測試，我們使用了獨立的 Confluent 集群，但此設定適用於分散式集群。

1. 從 Confluent 網站下載 Confluent Kafka。
2. 將包解壓縮到伺服器上的資料夾。
3. 導出兩個變數。

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

4. 對於獨立的 Confluent Kafka 設置，叢集會在 `/tmp`。它還創建 Zookeeper、Kafka、模式註冊表、連接、ksql-server 和控制中心資料夾，並從複製它們各自的配置文件 `$CONFLUENT_HOME`。請參閱以下範例：

```
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。如果使用預設參數，則無需更改任何內容。

```
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，我們更新了 `server. xxx`財產。預設情況下，您需要三個 Zookeeper 來選擇 Kafka 領導者。

6. 我們在 `/tmp/confluent.406980/zookeeper/data`具有唯一 ID：

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

我們將最後一個 IP 位址數用於 myid 檔案。我們對 Kafka、connect、control-center、Kafka、Kafka-rest、ksql-server 和 schema-registry 配置使用了預設值。

7. 啟動 Kafka 服務。

```
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#
```

每個配置都有一個日誌資料夾，有助於解決問題。在某些情況下，服務需要更多時間才能啟動。確保所有服務均已啟動並正在運行。

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#
```

您也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. 預設情況下，`confluentinc-kafka-connect-s3`安裝在 `/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 服務並重新啟動它們。

```
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. 驗證儲存桶是否可存取。

```
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. 為 s3 和 bucket 配置配置 s3-sink 屬性檔。

```
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. 將一些記錄匯入到 s3 儲存桶。

```
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. 載入 s3-sink 連接器。

```
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. 檢查 s3-sink 狀態。

```
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. 檢查日誌以確保 s3-sink 已準備好接受主題。

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

19. 檢查 Kafka 中的主題。

```
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. 檢查 s3 儲存桶中的物件。

```
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. 若要驗證內容，請執行以下命令將每個檔案從 S3 複製到本機檔案系統：

```
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. 若要列印記錄，請使用 avro-tools-1.11.0.1.jar（可在 "Apache 檔案"）。

```
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:~#
```

Instaclustr Kafka Connect 連接器

Instaclustr 支援 Kafka Connect 連接器及其詳細資訊 - "更多詳情"。Instaclustr 提供額外的連接器 "他們的詳細信息"

融合自平衡集群

如果您以前曾管理過 Kafka 集群，那麼您可能熟悉手動將分區重新分配給不同代理以確保整個集群的工作負載平衡所帶來的挑戰。對於部署大量 Kafka 的組織來說，重新整理大量資料可能是一項艱鉅、繁瑣且有風險的任務，尤其是在叢集之上建立關鍵任務應用程式時。然而，即使對於最小的 Kafka 用例，該過程也很耗時且容易出現人為錯誤。

在我們的實驗室中，我們測試了 Confluent 自平衡叢集功能，該功能可以根據叢集拓撲變化或不均勻負載自動重新平衡。 Confluent 重新平衡測試有助於測量節點發生故障或擴展節點需要在代理之間重新平衡資料時新增代理的時間。在經典的 Kafka 配置中，需要重新平衡的資料量會隨著叢集的成長而成長，但在分層儲存中，重新平衡僅限於少量資料。根據我們的驗證，在經典的 Kafka 架構中，分層儲存中的重新平衡需要幾秒鐘或幾分鐘，並且隨著叢集的增長而線性增長。

在自平衡叢集中，分區重新平衡完全自動化，以優化 Kafka 的吞吐量，加速代理擴展，並減少運行大型叢集的營運負擔。在穩定狀態下，自平衡叢集監控代理之間的資料偏差，並不斷重新分配分區以優化叢集效能。當擴大或縮小平台規模時，自平衡群集會自動識別新代理的存在或舊代理的刪除，並觸發後續分區重新分配。這使您能夠輕鬆地新增和停用代理，從而使您的 Kafka 叢集從根本上更加有彈性。這些好處不需要任何人工幹預、複雜的數學運算或分區重新分配通常帶來的人為錯誤風險。因此，資料重新平衡可以在更短的時間內完成，您可以自由地專注於更高價值的事件流項目，而不需要不斷監督您的叢集。

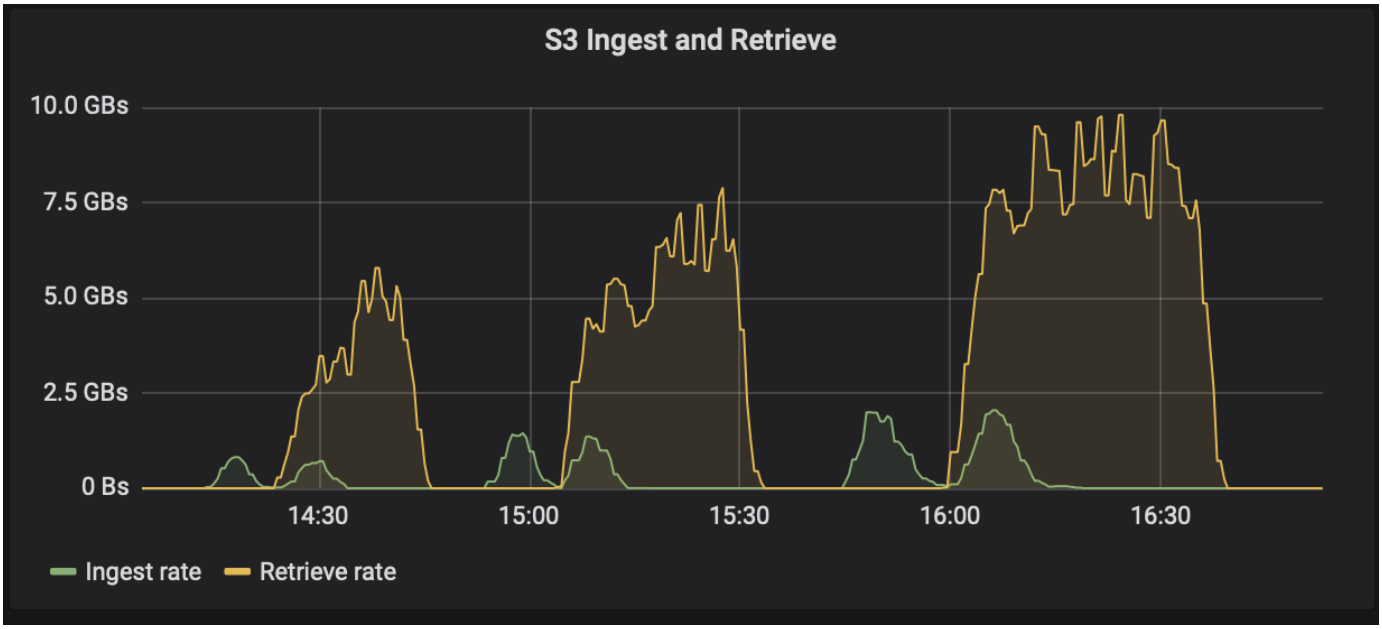
Instaclustr 還支援自我再平衡功能，並且已為多個客戶實施。

最佳實踐指南

本節介紹了從此次認證中獲得的經驗教訓。

- 根據我們的驗證，S3 物件儲存最適合 Confluent 保存資料。
- 我們可以使用高吞吐量 SAN（特別是 FC）來保存代理熱資料或本機磁碟，因為在 Confluent 分層儲存配置中，代理資料目錄中儲存的資料大小取決於資料移至物件儲存時的段大小和保留時間。
- 當segment.bytes較高時，物件儲存提供更好的效能；我們測試了512MB。
- 在 Kafka 中，主題中產生的每個記錄的鍵或值的長度（以位元組為單位）由 `length.key.value`範圍。對於StorageGRID，S3 物件提取和檢索效能提升到更高的值。例如，512 位元組提供 5.8GBps 的檢索，1024 位元組提供 7.5GBps 的 s3 檢索，而 2048 位元組提供接近 10GBps 的檢索。

下圖展示了基於 `length.key.value`。



- Kafka 調優。為了提高分層儲存的效能，可以增加 TierFetcherNumThreads 和 TierArchiverNumThreads。作為一般準則，您需要增加 TierFetcherNumThreads 以匹配實體 CPU 核心的數量，並將 TierArchiverNumThreads 增加到 CPU 核心數量的一半。例如，在伺服器屬性中，如果您有一台具有八個實體核心的機器，請設定 confluent.tier.fetcher.num.threads = 8 和 confluent.tier.archiver.num.threads = 4。
- *主題刪除的時間間隔*當主題被刪除時，物件儲存中的日誌段檔案的刪除不會立即開始。相反，在刪除這些文件之前有一個預設值為 3 小時的時間間隔。您可以修改配置 confluent.tier.topic.delete.check.interval.ms 來變更此間隔的值。如果刪除主題或集群，您也可以手動刪除相應儲存桶中的物件。
- *分層儲存內部主題的 ACL。 *對於內部部署，建議的最佳實踐是在用於分層儲存的內部主題上啟用 ACL 授權器。設定 ACL 規則以限制只有代理使用者才能存取此資料。這可以保護內部主題並防止未經授權存取分層儲存資料和元資料。

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

替換用戶 `<kafka>`與您部署中的實際代理主體一起。

例如，命令 `confluent-tier-state`設定內部主題的 ACL 以進行分層儲存。目前，只有一個與分層儲存相關的內部主題。此範例建立了一個 ACL，為內部主題上的所有操作提供主要的 Kafka 權限。

漿紗

Kafka 大小調整可以透過四種組態模式進行：簡單、粒度、反向和分割。

簡單的

簡單模式適合首次使用 Apache Kafka 的使用者或早期使用案例。對於此模式，您可以提供吞吐量 MBps、讀取扇出、保留和資源利用率百分比（預設值為 60%）等要求。您也可以進入環境，例如本機（裸機、VMware

、Kubernetes 或 OpenStack）或雲端。根據這些信息，Kafka 叢集的大小提供了代理、zookeeper、Apache Kafka 連接工作器、模式註冊表、REST 代理、ksqlDB 和 Confluent 控制中心所需的伺服器數量。

對於分層存儲，請考慮使用粒度配置模式來確定 Kafka 叢集的大小。粒度模式適合經驗豐富的 Apache Kafka 使用者或定義明確的用例。本節介紹生產者、流處理器和消費者的大小。

生產者

若要描述 Apache Kafka 的生產者（例如本機用戶端、REST 代理程式或 Kafka 連接器），請提供以下資訊：

- *姓名。 *火花。
- *生產者類型。 *應用程式或服務、代理（REST、MQTT、其他）和現有資料庫（RDBMS、NOSQL、其他）。您也可以選擇“我不知道”。
- *平均吞吐量。 *以每秒事件數計算（例如 1,000,000）。
- *峰值吞吐量。 *以每秒事件數計算（例如 4,000,000）。
- *平均訊息大小。 *以位元組為單位，未壓縮（最大 1MB；例如 1000）。
- *訊息格式。 *選項包括 Avro、JSON、協議緩衝區、二進制、文字、“我不知道”和其他。
- *複製因子。 *選項為 1、2、3（Confluent 建議）、4、5 或 6。
- *保留時間。 *有一天（例如）。您希望將資料儲存在 Apache Kafka 中多久？輸入 -1 和任意單位可表示無限時間。計算器假設無限保留的保留時間為 10 年。
- 選取「啟用分層儲存以減少代理數量並允許無限儲存？」複選框。
- 啟用分層儲存後，保留欄位控制在代理本地儲存的熱資料集。檔案保留欄位控制資料在檔案物件儲存中的儲存時間。
- *檔案存放保留。 *一年（例如）。您希望將資料保存在檔案儲存中多久？輸入 -1 和任意單位可獲得無限持續時間。計算器假設無限保留的保留時間為 10 年。
- *增長乘數。 * 1（例如）。如果此參數的值是基於目前吞吐量，則將其設為 1。若要根據額外成長確定大小，請將此參數設定為成長乘數。
- 生產者實例的數量。 10（例如）。將運行多少個生產者實例？此輸入需要將 CPU 負載納入尺寸計算。空白值表示 CPU 負載未納入計算。

根據此範例輸入，尺寸對生產者有以下影響：

- 未壓縮位元組的平均吞吐量：1GBps。未壓縮位元組的峰值吞吐量：4GBps。壓縮位元組的平均吞吐量：400MBps。壓縮位元組的峰值吞吐量：1.6GBps。這是基於預設的 60% 壓縮率（您可以更改此值）。
  - 所需的代理熱集儲存總量：31,104TB，包括複製和壓縮。所需的總代理外存檔儲存：378,432TB（壓縮）。使用"https://fusion.netapp.com"用於StorageGRID大小調整。

流處理器必須描述從 Apache Kafka 使用資料並返回 Apache Kafka 的應用程式或服務。大多數情況下，這些都是在 ksqlDB 或 Kafka Streams 中建構的。

- *姓名。 *火花飄帶。
- *處理時間。 *該處理器處理一條訊息需要多長時間？
  - 1 毫秒（簡單、無狀態轉換）[範例]，10 毫秒（有狀態的記憶體操作）。
  - 100ms（有狀態網路或磁碟操作），1000ms（第三方 REST 呼叫）。

    - 我已經對這個參數進行了基準測試，並且確切地知道需要多長時間。

- *輸出保留。 * 1天（範例）。流處理器將其輸出傳回給 Apache Kafka。您希望這些輸出資料在 Apache Kafka 中儲存多久？輸入 -1 和任意單位可獲得無限持續時間。
- 選取核取方塊“啟用分層儲存以減少代理數量並允許無限儲存？”
- *檔案存放保留。 * 1年（例如）。您希望將資料保存在檔案儲存中多久？輸入 -1 和任意單位可獲得無限持續時間。計算器假設無限保留的保留時間為 10 年。
- *輸出直通百分比。 * 100（例如）。流處理器將其輸出傳回給 Apache Kafka。入站吞吐量的百分比將輸出回 Apache Kafka？例如，如果入站吞吐量為 20MBps，且該值為 10，則輸出吞吐量將為 2MBps。
- 這是從哪些應用程式讀取的？選擇“Spark”，這是基於生產者類型的大小調整中使用的名稱。根據上述輸入，您可以預期流處理器執行個體和主題分區估計的大小會產生以下影響：
- 此流處理器應用程式需要以下數量的實例。傳入的主題可能也需要這麼多的分區。聯絡 Confluent 確認此參數。
    - 1,000 表示平均吞吐量，無成長乘數
    - 峰值吞吐量為 4,000，無成長乘數
    - 1,000 表示平均吞吐量，並帶有成長乘數
    - 峰值吞吐量為 4,000，並帶有增長乘數

消費者

描述使用來自 Apache Kafka 的資料但不傳回 Apache Kafka 的應用程式或服務；例如，本機用戶端或 Kafka 連接器。

- *姓名。 *激發消費者。
- *處理時間。 *該消費者需要多長時間來處理一則訊息？
    - 1ms（例如，像日誌記錄這樣的簡單且無狀態的任務）
    - 10ms（快速寫入資料儲存）
    - 100 毫秒（資料儲存寫入速度慢）
    - 1000ms（第三方 REST 呼叫）
    - 一些其他已知持續時間的基準測試過程。
- *消費者類型。 *應用程式、代理程式或接收器至現有資料儲存（RDBMS、NoSQL 等）。
- 這是從哪些應用程式讀取的？將此參數與先前確定的生產者和流量大小連接起來。

根據上述輸入，您必須確定消費者實例的大小和主題分區估計。消費者應用程式需要以下數量的實例。

- 平均吞吐量為 2,000，無成長乘數
- 峰值吞吐量為 8,000，無成長乘數
- 平均吞吐量為 2,000，包括成長乘數
- 峰值吞吐量為 8,000，包括成長乘數

傳入的主題可能也需要這個數量的分區。聯絡 Confluent 進行確認。

除了對生產者、流處理器和消費者的要求之外，您還必須提供以下額外要求：

- *重建時間。 *例如4小時。如果 Apache Kafka 代理主機發生故障，其資料遺失，並且需要配置新主機來取代故障主機，那麼這個新主機必須多快重建自身？如果值未知，請將此參數留空。
- *資源利用率目標（百分比）。 *例如，60。您希望您的主機在平均吞吐量期間的使用率為何？ Confluent 建議利用率為 60%，除非您使用 Confluent 自平衡集群，在這種情況下利用率可能會更高。

描述你的環境

- *您的叢集將在什麼環境中運作？ *亞馬遜網路服務、微軟 Azure、Google雲端平台、本地裸機、本地 VMware、本地 OpenStack 還是本地 Kubernates？
- *主人詳細資料。 *核心數：例如48個，網卡類型（10GbE、40GbE、16GbE、1GbE或其他類型）。
- *儲存磁碟區。 *主持人：12（例如）。每個主機支援多少個硬碟或 SSD？ Confluent 建議每台主機配備 12 個硬碟。
- *儲存容量/磁碟區（以 GB 為單位）。 * 1000（例如）。單一磁碟區可以儲存多少 GB 的儲存空間？ Confluent 建議使用 1TB 磁碟。
- 儲存配置。儲存卷如何配置？ Confluent 建議使用 RAID10 來充分利用 Confluent 的所有功能。也支援 JBOD、SAN、RAID 1、RAID 0、RAID 5 和其他類型。
- *單卷吞吐量（MBps）。 * 125（例如）。單一儲存卷每秒的讀取或寫入速度是多少兆位元組？ Confluent 建議使用標準硬碟，其吞吐量通常為 125MBps。
- *記憶體容量（GB）。 * 64（例如）。

確定環境變數後，選擇“Size my Cluster”。根據上面指出的範例參數，我們確定了 Confluent Kafka 的以下大小：

- *Apache Kafka。 *經紀人數量：22。您的叢集受儲存限制。考慮啟用分層儲存以減少主機數量並允許無限儲存。
- Apache ZooKeeper。數量：5；Apache Kafka Connect Workers：數量：2；Schema Registry：數量：2；REST Proxy：數量：2；ksqlDB：數量：2；Confluent Control Center：數量：1。

對於平台團隊，請使用反向模式，無需考慮用例。使用分區模式來計算單一主題需要多少個分區。看 https://eventsizer.io根據反向和分割模式進行大小調整。

結論

本文檔提供了將 Confluent 分層儲存與NetApp儲存體結合使用的最佳實務指南，包括驗證測試、分層儲存效能結果、調整、Confluent S3 連接器和自平衡功能。考慮到 ILM 策略、經過多項效能測試驗證的 Confluent 效能以及業界標準的 S3 API， NetApp StorageGRID 物件儲存是 Confluent 分層儲存的最佳選擇。

在哪裡可以找到更多信息

要了解有關本文檔中描述的信息的更多信息，請查看以下文檔和/或網站：

- 什麼是 Apache Kafka

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

- NetApp產品文檔

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

- S3-sink 參數詳情

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

- 阿帕契卡夫卡

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

- Confluent 平台中的無限存儲

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

- Confluent 分層儲存 - 最佳實踐與規模

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

- Confluent 平台的 Amazon S3 接收器連接器

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

- Kafka 大小

  "https://eventsizer.io"

- StorageGRID大小調整

  "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/"

- Instaclustr客戶範例及其用例詳情

  https://www.instaclustr.com/blog/netapp-and-pegasystems-open-source-support-package/，https://www.instaclustr.com/wp-content/uploads/Insta_Case_Study_Pegasystems_1_21sep25.pdf

  https://www.instaclustr.com/resources/customer-case-study-pubnub/

  https://www.instaclustr.com/resources/customer-case-study-tesouro/