Best Practices für Confluent Kafka

NetApp artificial intelligence solutions

NetApp
February 12, 2026

Inhalt

Best Practices für Confluent Kafka

TR-4912: Best Practice-Richtlinien für Confluent Kafka Tiered Storage mit NetApp

Karthikeyan Nagalingam, Joseph Kandatilparambil, NetApp Rankesh Kumar, Confluent

Apache Kafka ist eine von der Community verteilte Event-Streaming-Plattform, die Billionen von Ereignissen pro Tag verarbeiten kann. Kafka wurde ursprünglich als Nachrichtenwarteschlange konzipiert und basiert auf der Abstraktion eines verteilten Commit-Protokolls. Seit seiner Erstellung und Open-Source-Veröffentlichung durch LinkedIn im Jahr 2011 hat sich Kafka von einer Nachrichtenwarteschlange zu einer vollwertigen Event-Streaming-Plattform entwickelt. Confluent liefert die Distribution von Apache Kafka mit der Confluent-Plattform. Die Confluent-Plattform ergänzt Kafka um zusätzliche Community- und kommerzielle Funktionen, die das Streaming-Erlebnis von Betreibern und Entwicklern in der Produktion in großem Maßstab verbessern sollen.

Dieses Dokument beschreibt die Best-Practice-Richtlinien für die Verwendung von Confluent Tiered Storage auf einem Object Storage-Angebot von NetApp und stellt die folgenden Inhalte bereit:

- Konfluente Verifizierung mit NetApp Object Storage – NetApp StorageGRID
- Leistungstests für mehrstufigen Speicher
- Best-Practice-Richtlinien für Confluent auf NetApp -Speichersystemen

Warum Confluent Tiered Storage?

Confluent hat sich zur Standard-Echtzeit-Streaming-Plattform für viele Anwendungen entwickelt, insbesondere für Big Data-, Analyse- und Streaming-Workloads. Mit Tiered Storage können Benutzer in der Confluent-Plattform Rechenleistung und Speicher trennen. Es macht die Datenspeicherung kostengünstiger, ermöglicht Ihnen die Speicherung nahezu unbegrenzter Datenmengen und die Skalierung von Arbeitslasten nach Bedarf nach oben (oder unten) und vereinfacht Verwaltungsaufgaben wie die Neuverteilung von Daten und Mandanten. S3-kompatible Speichersysteme können alle diese Funktionen nutzen, um Daten mit allen Ereignissen an einem Ort zu demokratisieren, wodurch die Notwendigkeit einer komplexen Datentechnik entfällt. Weitere Informationen dazu, warum Sie Tiered Storage für Kafka verwenden sollten, finden Sie unter"dieser Artikel von Confluent" .

NetApp instaclustr unterstützt Kafka ab Version 3.8.1 auch mit Tiered Storage. Weitere Details finden Sie hier. "Instaclust nutzt Kafka-Tiered Storage"

Warum NetApp StorageGRID für Tiered Storage?

StorageGRID ist eine branchenführende Objektspeicherplattform von NetApp. StorageGRID ist eine softwaredefinierte, objektbasierte Speicherlösung, die branchenübliche Objekt-APIs unterstützt, einschließlich der Amazon Simple Storage Service (S3)-API. StorageGRID speichert und verwaltet unstrukturierte Daten in großem Umfang, um einen sicheren, dauerhaften Objektspeicher bereitzustellen. Inhalte werden zur richtigen Zeit am richtigen Ort und auf der richtigen Speicherebene platziert, wodurch Arbeitsabläufe optimiert und die Kosten für global verteilte Rich Media gesenkt werden.

Das größte Unterscheidungsmerkmal von StorageGRID ist die Policy-Engine für Information Lifecycle

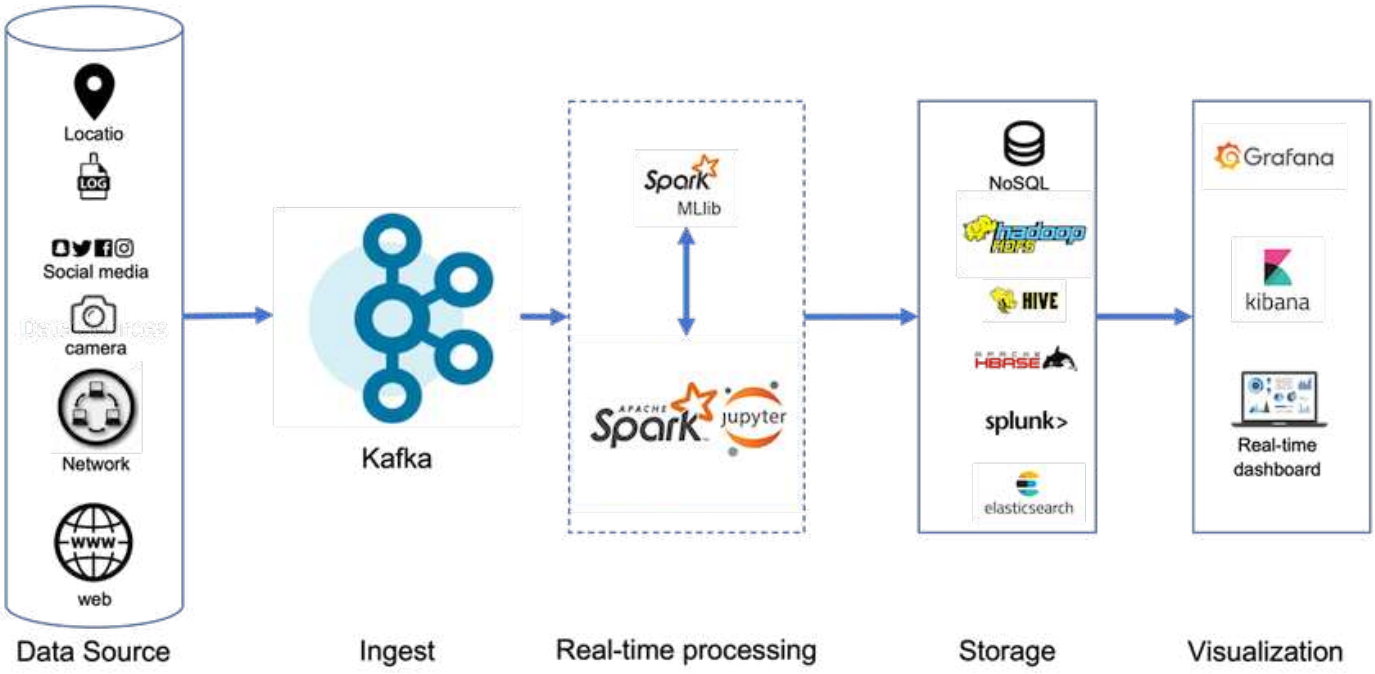
Management (ILM), die ein richtliniengesteuertes Datenlebenszyklusmanagement ermöglicht. Die Richtlinien-Engine kann Metadaten verwenden, um zu verwalten, wie Daten während ihrer gesamten Lebensdauer gespeichert werden, um zunächst die Leistung zu optimieren und mit zunehmendem Alter der Daten automatisch die Kosten und Haltbarkeit zu optimieren.

Aktivieren von Confluent Tiered Storage

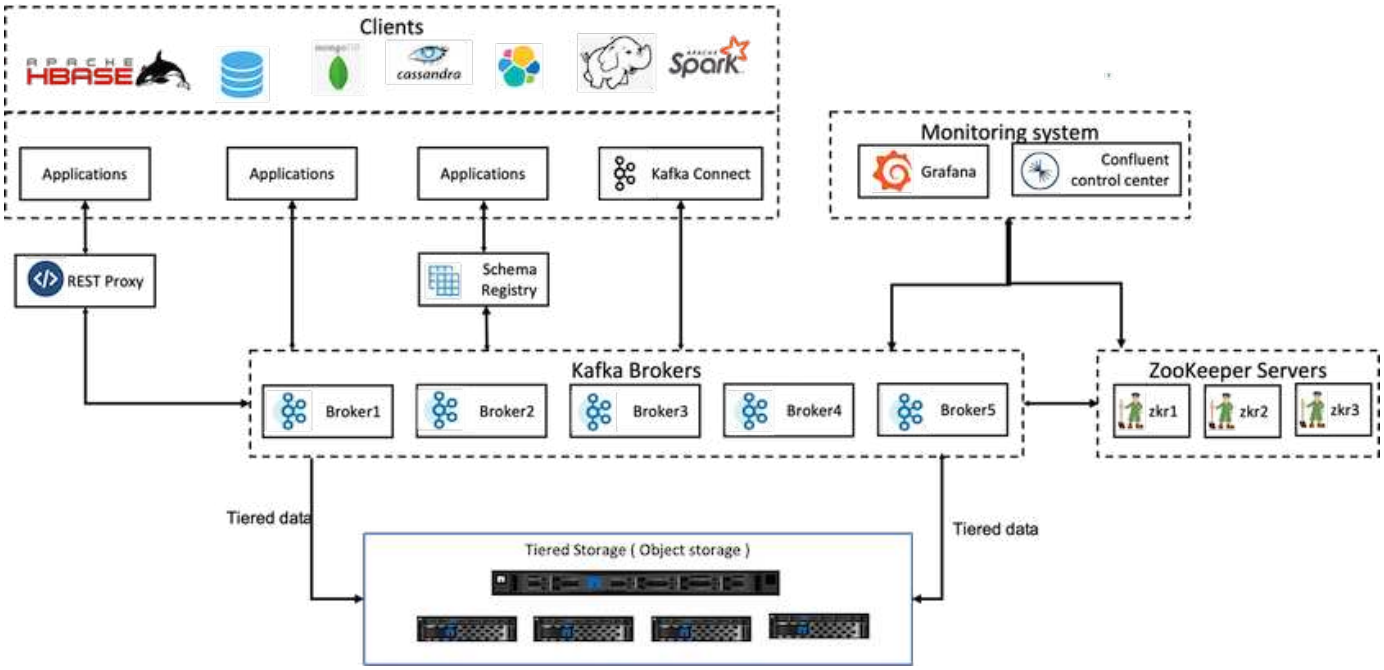
Die Grundidee des Tiered Storage besteht darin, die Aufgaben der Datenspeicherung von denen der Datenverarbeitung zu trennen. Durch diese Trennung wird es für die Datenspeicherungsebene und die Datenverarbeitungsebene wesentlich einfacher, unabhängig voneinander zu skalieren.

Eine mehrstufige Speicherlösung für Confluent muss zwei Faktoren berücksichtigen. Erstens müssen allgemeine Konsistenz- und Verfügbarkeitseigenschaften von Objektspeichern, wie etwa Inkonsistenzen bei LIST-Operationen und gelegentliche Nichtverfügbarkeit von Objekten, umgangen oder vermieden werden. Zweitens muss die Interaktion zwischen mehrstufigem Speicher und dem Replikations- und Fehlertoleranzmodell von Kafka korrekt gehandhabt werden, einschließlich der Möglichkeit, dass Zombie-Leader weiterhin Offset-Bereiche stufen. NetApp Object Storage bietet sowohl die konsistente Objektverfügbarkeit als auch das HA-Modell und stellt den erschöpften Speicher für Tier-Offset-Bereiche zur Verfügung. NetApp Objektspeicher bietet konsistente Objektverfügbarkeit und ein HA-Modell, um den erschöpften Speicher für Tier-Offset-Bereiche verfügbar zu machen.

Mit Tiered Storage können Sie Hochleistungsplattformen für Lese- und Schreibvorgänge mit geringer Latenz am Ende Ihrer Streaming-Daten verwenden. Außerdem können Sie günstigere, skalierbare Objektspeicher wie NetApp StorageGRID für historische Lesevorgänge mit hohem Durchsatz nutzen. Wir haben auch eine technische Lösung für Spark mit NetApp-Speichercontroller. Einzelheiten finden Sie hier. Die folgende Abbildung zeigt, wie Kafka in eine Echtzeit-Analyse-Pipeline passt.



Die folgende Abbildung zeigt, wie NetApp StorageGRID als Objektspeicherebene von Confluent Kafka passt.



Details zur Lösungsarchitektur

Dieser Abschnitt behandelt die für die Confluent-Verifizierung verwendete Hardware und Software. Diese Informationen gelten für die Bereitstellung der Confluent Platform mit NetApp -Speicher. Die folgende Tabelle umfasst die getestete Lösungsarchitektur und die Basiskomponenten.

Lösungskomponenten	Details
Confluent Kafka Version 6.2	• Drei Tierpfleger • Fünf Broker-Server • Fünf Tools-Server • Ein Grafana • Ein Kontrollzentrum
Linux (Ubuntu 18.04)	Alle Server
NetApp StorageGRID für Tiered Storage	• StorageGRID -Software • 1 x SG1000 (Lastverteiler) • 4 x SGF6024 • 4 x 24 x 800 SSDs • S3-Protokoll • 4 x 100 GbE (Netzwerkkonnektivität zwischen Broker- und StorageGRID Instanzen)
15 Fujitsu PRIMERGY RX2540 Server	Jedes ist ausgestattet mit: * 2 CPUs, insgesamt 16 physischen Kernen * Intel Xeon * 256 GB physischem Speicher * 100 GbE Dual-Port

Technologieübersicht

In diesem Abschnitt wird die in dieser Lösung verwendete Technologie beschrieben.

NetApp StorageGRID

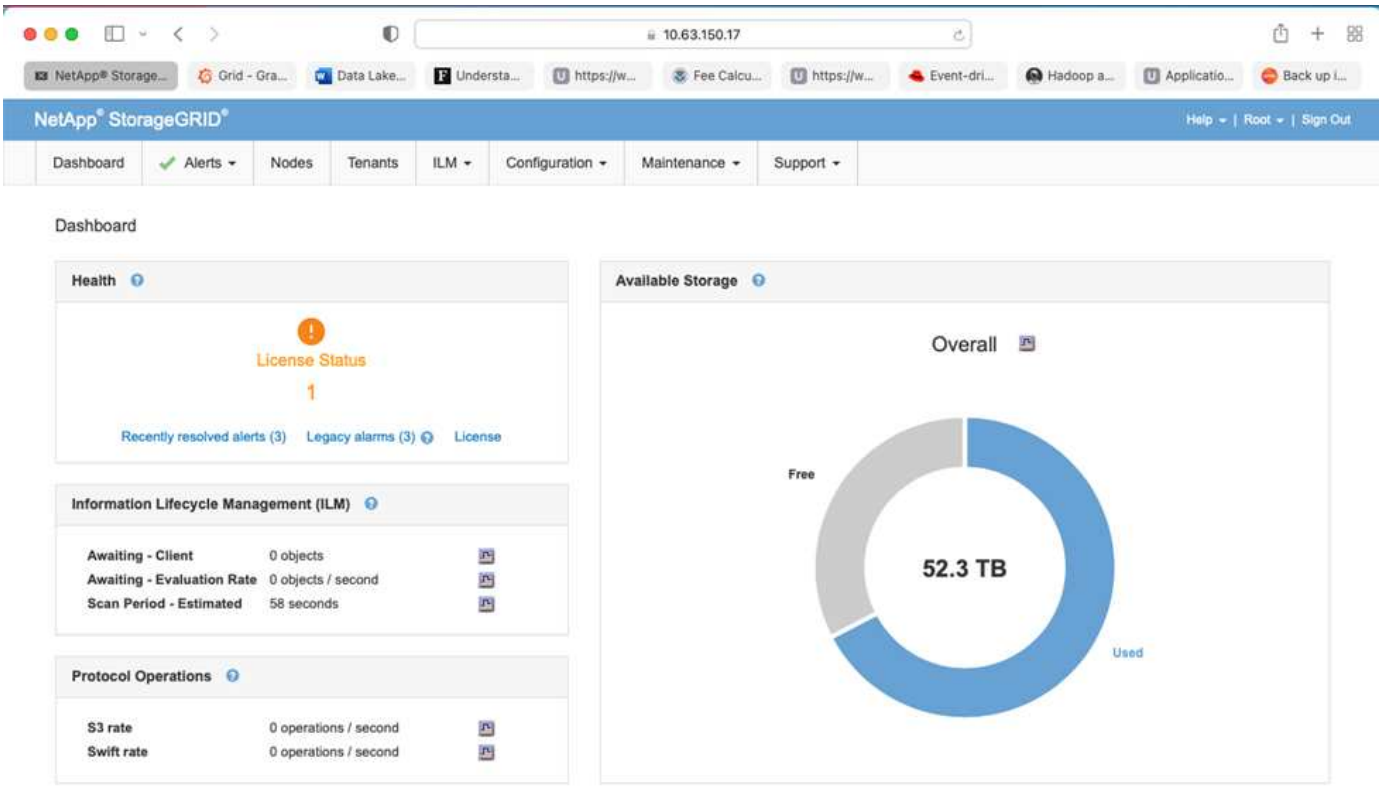
NetApp StorageGRID ist eine leistungsstarke und kostengünstige Objektspeicherplattform. Durch die Verwendung von mehrstufigem Speicher werden die meisten Daten auf Confluent Kafka, die im lokalen Speicher oder im SAN-Speicher des Brokers gespeichert sind, in den Remote-Objektspeicher ausgelagert. Diese Konfiguration führt zu erheblichen Betriebsverbesserungen, da Zeit und Kosten für die Neuausrichtung, Erweiterung oder Verkleinerung von Clustern oder den Austausch eines ausgefallenen Brokers reduziert werden. Der Objektspeicher spielt eine wichtige Rolle bei der Verwaltung von Daten, die sich auf der Objektspeicherebene befinden. Deshalb ist die Auswahl des richtigen Objektspeichers wichtig.

StorageGRID bietet intelligentes, richtliniengesteuertes globales Datenmanagement mithilfe einer verteilten, knotenbasierten Grid-Architektur. Es vereinfacht die Verwaltung von Petabytes unstrukturierter Daten und Milliarden von Objekten durch seinen allgegenwärtigen globalen Objekt-Namespace in Kombination mit ausgefeilten Datenverwaltungsfunktionen. Der Objektzugriff per Einzelaufruf erstreckt sich über mehrere Standorte und vereinfacht Hochverfügbarkeitsarchitekturen, während gleichzeitig ein kontinuierlicher Objektzugriff unabhängig von Standort- oder Infrastrukturausfällen gewährleistet wird.

Durch die Mandantenfähigkeit können mehrere unstrukturierte Cloud- und Unternehmensdatenanwendungen sicher im selben Grid verwaltet werden, wodurch sich der ROI und die Anwendungsfälle für NetApp StorageGRID erhöhen. Sie können mehrere Service-Levels mit metadatengesteuerten Objekt-Lebenszyklusrichtlinien erstellen und so Haltbarkeit, Schutz, Leistung und Lokalität über mehrere geografische Regionen hinweg optimieren. Benutzer können Datenverwaltungsrichtlinien anpassen und Verkehrsbeschränkungen überwachen und anwenden, um sich unterbrechungsfrei an die Datenlandschaft anzupassen, wenn sich ihre Anforderungen in sich ständig verändernden IT-Umgebungen ändern.

Einfache Verwaltung mit Grid Manager

Der StorageGRID Grid Manager ist eine browserbasierte grafische Benutzeroberfläche, mit der Sie Ihr StorageGRID -System an weltweit verteilten Standorten in einer einzigen Fensteransicht konfigurieren, verwalten und überwachen können.



Mit der StorageGRID Grid Manager-Schnittstelle können Sie die folgenden Aufgaben ausführen:

- Verwalten Sie global verteilte Repositories im Petabyte-Bereich mit Objekten wie Bildern, Videos und Aufzeichnungen.
- Überwachen Sie Grid-Knoten und -Dienste, um die Objektverfügbarkeit sicherzustellen.
- Verwalten Sie die Platzierung von Objektdaten im Laufe der Zeit mithilfe von Regeln für das Information Lifecycle Management (ILM). Diese Regeln legen fest, was mit den Daten eines Objekts nach der Aufnahme geschieht, wie sie vor Verlust geschützt werden, wo und wie lange die Objektdaten gespeichert werden.
- Überwachen Sie Transaktionen, Leistung und Vorgänge innerhalb des Systems.

Richtlinien für das Information Lifecycle Management

StorageGRID verfügt über flexible Datenverwaltungsrichtlinien, die das Aufbewahren von Replikatkopien Ihrer Objekte und die Verwendung von EC-Schemata (Erasure Coding) wie 2+1 und 4+2 (unter anderem) zum Speichern Ihrer Objekte umfassen, abhängig von spezifischen Leistungs- und Datenschutzanforderungen. Da sich Arbeitslasten und Anforderungen im Laufe der Zeit ändern, ist es üblich, dass sich auch die ILM-Richtlinien im Laufe der Zeit ändern müssen. Das Ändern von ILM-Richtlinien ist eine Kernfunktion, die es StorageGRID Kunden ermöglicht, sich schnell und einfach an ihre sich ständig ändernde Umgebung anzupassen.

Performance

StorageGRID skaliert die Leistung durch Hinzufügen weiterer Speicherknoten, die VMs, Bare Metal oder speziell entwickelte Geräte wie die"SG5712, SG5760, SG6060 oder SGF6024" . In unseren Tests haben wir die wichtigsten Leistungsanforderungen von Apache Kafka mit einem Drei-Knoten-Raster der Mindestgröße unter Verwendung des SGF6024-Geräts übertroffen. Wenn Kunden ihren Kafka-Cluster mit zusätzlichen Brokern skalieren, können sie weitere Speicherknoten hinzufügen, um Leistung und Kapazität zu erhöhen.

Load Balancer und Endpunktkonfiguration

Admin-Knoten in StorageGRID bieten die Grid Manager-Benutzeroberfläche (Benutzeroberfläche) und den REST-API-Endpunkt zum Anzeigen, Konfigurieren und Verwalten Ihres StorageGRID -Systems sowie Prüfprotokolle zum Verfolgen der Systemaktivität. Um einen hochverfügbaren S3-Endpunkt für den mehrstufigen Confluent Kafka-Speicher bereitzustellen, haben wir den StorageGRID Load Balancer implementiert, der als Dienst auf Admin-Knoten und Gateway-Knoten ausgeführt wird. Darüber hinaus verwaltet der Load Balancer auch den lokalen Datenverkehr und kommuniziert mit dem GSLB (Global Server Load Balancing), um bei der Notfallwiederherstellung zu helfen.

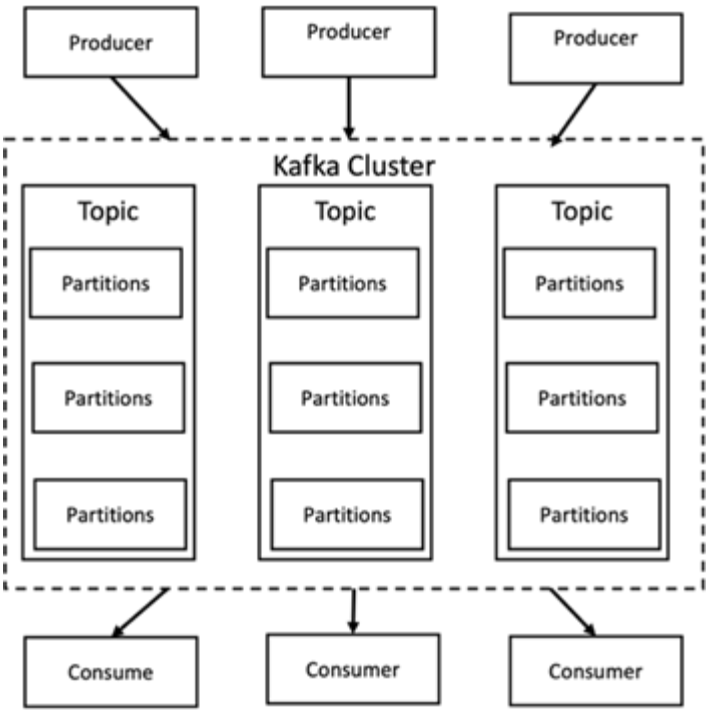
Um die Endpunktkonfiguration weiter zu verbessern, bietet StorageGRID im Admin-Knoten integrierte Richtlinien zur Verkehrsklassifizierung, ermöglicht Ihnen die Überwachung Ihres Workload-Verkehrs und wendet verschiedene Quality-of-Service-Grenzwerte (QoS) auf Ihre Workloads an. Richtlinien zur Verkehrsklassifizierung werden auf Endpunkte des StorageGRID Load Balancer-Dienstes für Gateway-Knoten und Admin-Knoten angewendet. Diese Richtlinien können bei der Verkehrsgestaltung und -überwachung helfen.

Verkehrsklassifizierung in StorageGRID

StorageGRID verfügt über eine integrierte QoS-Funktionalität. Richtlinien zur Verkehrsklassifizierung können dabei helfen, verschiedene Arten von S3-Verkehr zu überwachen, der von einer Clientanwendung kommt. Sie können dann Richtlinien erstellen und anwenden, um diesen Datenverkehr basierend auf der Eingangs-/Ausgangsbandbreite, der Anzahl gleichzeitiger Lese-/Schreibanforderungen oder der Lese-/Schreibanforderungsrate zu begrenzen.

Apache Kafka

Apache Kafka ist eine Framework-Implementierung eines Softwarebusses mit Stream-Verarbeitung, geschrieben in Java und Scala. Ziel ist es, eine einheitliche Plattform mit hohem Durchsatz und geringer Latenz für die Verarbeitung von Echtzeit-Datenfeeds bereitzustellen. Kafka kann über Kafka Connect eine Verbindung zu einem externen System zum Datenexport und -import herstellen und bietet Kafka Streams, eine Java-Stream-Verarbeitungsbibliothek. Kafka verwendet ein binäres, TCP-basiertes Protokoll, das auf Effizienz optimiert ist und auf einer „Nachrichtensatz“-Abstraktion basiert, die Nachrichten auf natürliche Weise gruppiert, um den Overhead des Netzwerk-Roundtrips zu reduzieren. Dies ermöglicht größere sequenzielle Festplattenvorgänge, größere Netzwerkpakete und zusammenhängende Speicherblöcke, wodurch Kafka einen stoßweisen Strom zufälliger Nachrichtenschreibvorgänge in lineare Schreibvorgänge umwandeln kann. Die folgende Abbildung zeigt den grundlegenden Datenfluss von Apache Kafka.



Kafka speichert Schlüssel-Wert-Nachrichten, die von einer beliebigen Anzahl von Prozessen, sogenannten Produzenten, stammen. Die Daten können innerhalb verschiedener Themen in verschiedene Partitionen aufgeteilt werden. Innerhalb einer Partition werden Nachrichten streng nach ihren Offsets (der Position einer Nachricht innerhalb einer Partition) geordnet und zusammen mit einem Zeitstempel indiziert und gespeichert. Andere Prozesse, sogenannte Verbraucher, können Nachrichten aus Partitionen lesen. Für die Stream-Verarbeitung bietet Kafka die Streams-API, die das Schreiben von Java-Anwendungen ermöglicht, die Daten von Kafka nutzen und Ergebnisse zurück an Kafka schreiben. Apache Kafka funktioniert auch mit externen Stream-Verarbeitungssystemen wie Apache Apex, Apache Flink, Apache Spark, Apache Storm und Apache NiFi.

Kafka läuft auf einem Cluster aus einem oder mehreren Servern (Brokern genannt) und die Partitionen aller Themen werden auf die Clusterknoten verteilt. Darüber hinaus werden Partitionen auf mehrere Broker repliziert. Diese Architektur ermöglicht es Kafka, riesige Nachrichtenströme fehlertolerant zu übermitteln und einige der herkömmlichen Nachrichtensysteme wie Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP) usw. zu ersetzen. Seit der Version 0.11.0.0 bietet Kafka transaktionale Schreibvorgänge, die mithilfe der Streams-API eine exakt einmalige Stream-Verarbeitung ermöglichen.

Kafka unterstützt zwei Arten von Themen: regulär und kompakt. Reguläre Themen können mit einer Aufbewahrungsdauer oder einer Platzbegrenzung konfiguriert werden. Wenn Datensätze vorhanden sind, die älter als die angegebene Aufbewahrungszeit sind, oder wenn die Speicherplatzgrenze für eine Partition überschritten wird, darf Kafka alte Daten löschen, um Speicherplatz freizugeben. Standardmäßig sind Themen mit einer Aufbewahrungszeit von 7 Tagen konfiguriert, es ist jedoch auch möglich, Daten unbegrenzt zu speichern. Bei komprimierten Themen verfallen Datensätze nicht aufgrund von Zeit- oder Speicherplatzbeschränkungen. Stattdessen behandelt Kafka spätere Nachrichten als Aktualisierungen älterer Nachrichten mit demselben Schlüssel und garantiert, dass die neueste Nachricht pro Schlüssel niemals gelöscht wird. Benutzer können Nachrichten vollständig löschen, indem sie eine sogenannte Tombstone-Nachricht mit dem Nullwert für einen bestimmten Schlüssel schreiben.

Es gibt fünf wichtige APIs in Kafka:

- **Produzenten-API.** Ermöglicht einer Anwendung, Datensatzströme zu veröffentlichen.
- **Consumer-API.** Ermöglicht einer Anwendung, Themen zu abonnieren und Datensatzströme zu verarbeiten.

- **Connector-API.** Führt die wiederverwendbaren Producer- und Consumer-APIs aus, die die Themen mit den vorhandenen Anwendungen verknüpfen können.
- **Streams-API.** Diese API konvertiert die Eingabeströme in Ausgaben und erzeugt das Ergebnis.
- **Admin-API.** Wird zum Verwalten von Kafka-Themen, Brokern und anderen Kafka-Objekten verwendet.

Die Consumer- und Producer-APIs bauen auf dem Kafka-Messaging-Protokoll auf und bieten eine Referenzimplementierung für Kafka-Consumer- und Producer-Clients in Java. Das zugrunde liegende Nachrichtenprotokoll ist ein Binärprotokoll, das Entwickler verwenden können, um ihre eigenen Consumer- oder Producer-Clients in jeder beliebigen Programmiersprache zu schreiben. Dadurch wird Kafka aus dem Ökosystem der Java Virtual Machine (JVM) entsperrt. Eine Liste der verfügbaren Nicht-Java-Clients wird im Apache Kafka-Wiki verwaltet.

Apache Kafka-Anwendungsfälle

Apache Kafka wird am häufigsten für Messaging, Website-Aktivitätsverfolgung, Metriken, Protokollaggregation, Stream-Verarbeitung, Event Sourcing und Commit-Protokollierung verwendet.

- Kafka verfügt über einen verbesserten Durchsatz, integrierte Partitionierung, Replikation und Fehlertoleranz, was es zu einer guten Lösung für groß angelegte Nachrichtenverarbeitungsanwendungen macht.
- Kafka kann die Aktivitäten eines Benutzers (Seitenaufrufe, Suchvorgänge) in einer Tracking-Pipeline als eine Reihe von Publish-Subscribe-Feeds in Echtzeit wiederherstellen.
- Kafka wird häufig für Betriebsüberwachungsdaten verwendet. Dabei werden Statistiken aus verteilten Anwendungen aggregiert, um zentralisierte Feeds mit Betriebsdaten zu erstellen.
- Viele Leute verwenden Kafka als Ersatz für eine Protokollaggregationslösung. Bei der Protokollaggregation werden in der Regel physische Protokolldateien von Servern gesammelt und zur Verarbeitung an einem zentralen Ort (z. B. einem Dateiserver oder HDFS) abgelegt. Kafka abstrahiert Dateidetails und bietet eine sauberere Abstraktion von Protokoll- oder Ereignisdaten als Nachrichtenstrom. Dies ermöglicht eine Verarbeitung mit geringerer Latenz und eine einfachere Unterstützung mehrerer Datenquellen und einer verteilten Datennutzung.
- Viele Kafka-Benutzer verarbeiten Daten in Verarbeitungspipelines, die aus mehreren Phasen bestehen, in denen Roheingabedaten aus Kafka-Themen verwendet und dann aggregiert, angereichert oder anderweitig in neue Themen zur weiteren Verwendung oder Weiterverarbeitung umgewandelt werden. Beispielsweise könnte eine Verarbeitungspipeline zum Empfehlen von Nachrichtenartikeln Artikelinhalte aus RSS-Feeds crawlen und in einem „Artikel“-Thema veröffentlichen. Bei der weiteren Verarbeitung kann dieser Inhalt normalisiert oder dedupliziert und der bereinigte Artikelinhalt in einem neuen Thema veröffentlicht werden. In einer letzten Verarbeitungsphase kann versucht werden, den Benutzern diesen Inhalt zu empfehlen. Solche Verarbeitungspipelines erstellen Diagramme von Echtzeit-Datenflüssen basierend auf den einzelnen Themen.
- Event Sourcing ist ein Anwendungsdesignstil, bei dem Statusänderungen als zeitlich geordnete Abfolge von Datensätzen protokolliert werden. Die Unterstützung von Kafka für sehr große gespeicherte Protokolldaten macht es zu einem hervorragenden Backend für eine in diesem Stil erstellte Anwendung.
- Kafka kann als eine Art externes Commit-Log für ein verteiltes System dienen. Das Protokoll hilft bei der Replikation von Daten zwischen Knoten und fungiert als Neusynchronisierungsmechanismus für ausgefallene Knoten, um ihre Daten wiederherzustellen. Die Protokollkomprimierungsfunktion in Kafka unterstützt diesen Anwendungsfall.

Zusammenfließend

Confluent Platform ist eine unternehmensreife Plattform, die Kafka um erweiterte Funktionen ergänzt, die die Anwendungsentwicklung und Konnektivität beschleunigen, Transformationen durch Stream-Verarbeitung

ermöglichen, Unternehmensabläufe im großen Maßstab vereinfachen und strenge Architekturanforderungen erfüllen sollen. Confluent wurde von den ursprünglichen Entwicklern von Apache Kafka entwickelt und erweitert die Vorteile von Kafka um Funktionen auf Unternehmensniveau, während es gleichzeitig den Aufwand für die Verwaltung oder Überwachung von Kafka verringert. Heute nutzen über 80 % der Fortune 100-Unternehmen Datenstreaming-Technologie – und die meisten von ihnen verwenden Confluent.

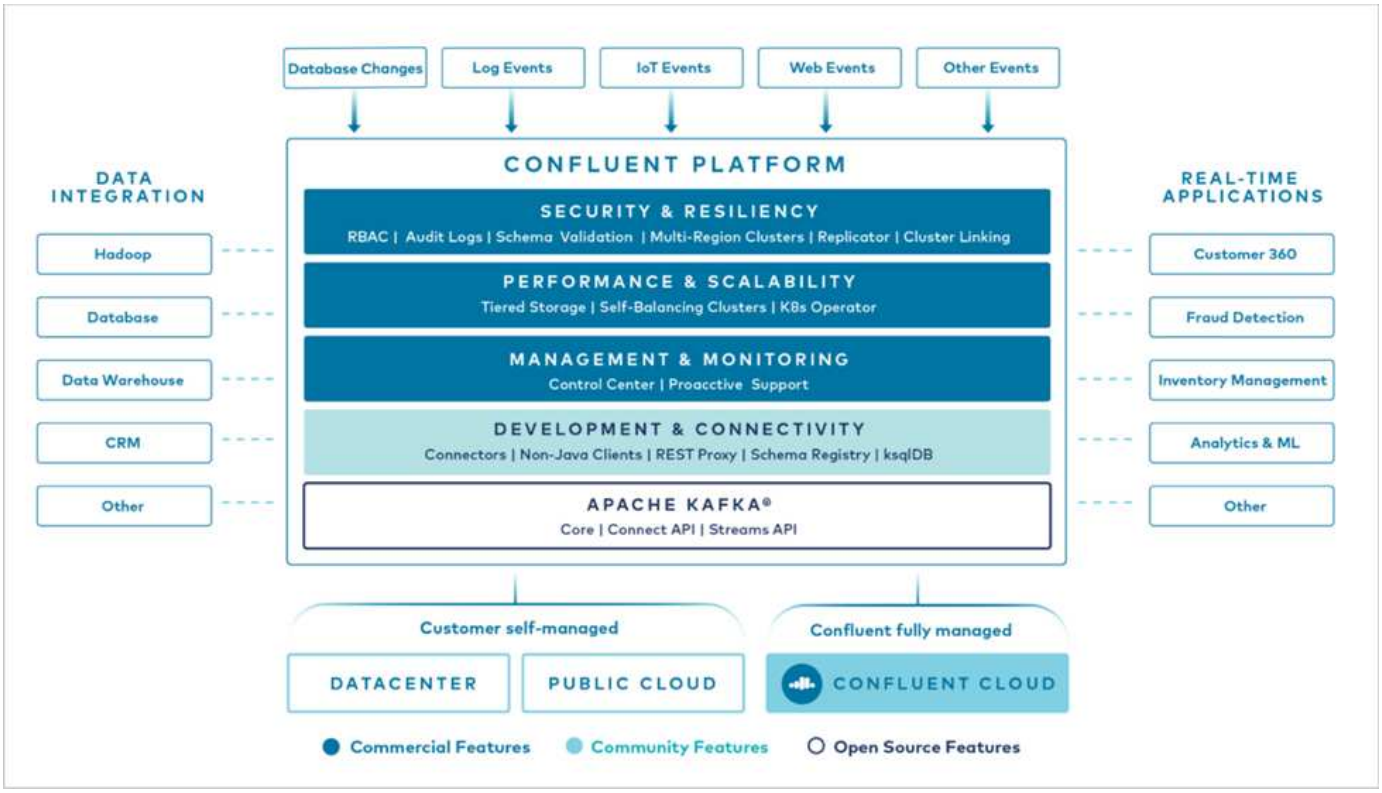
Warum Confluent?

Durch die Integration historischer und Echtzeitdaten in eine einzige, zentrale Quelle der Wahrheit erleichtert Confluent den Aufbau einer völlig neuen Kategorie moderner, ereignisgesteuerter Anwendungen, den Aufbau einer universellen Datenpipeline und die Erschließung leistungsstarker neuer Anwendungsfälle mit voller Skalierbarkeit, Leistung und Zuverlässigkeit.

Wofür wird Confluent verwendet?

Mit der Confluent Platform können Sie sich darauf konzentrieren, wie Sie aus Ihren Daten geschäftlichen Nutzen ziehen, anstatt sich um die zugrunde liegenden Mechanismen zu kümmern, beispielsweise darum, wie Daten zwischen unterschiedlichen Systemen transportiert oder integriert werden. Insbesondere vereinfacht die Confluent Platform die Verbindung von Datenquellen mit Kafka, die Erstellung von Streaming-Anwendungen sowie die Sicherung, Überwachung und Verwaltung Ihrer Kafka-Infrastruktur. Heute wird die Confluent Platform für eine breite Palette von Anwendungsfällen in zahlreichen Branchen eingesetzt, von Finanzdienstleistungen, Omnichannel-Einzelhandel und autonomen Autos bis hin zu Betrugserkennung, Microservices und IoT.

Die folgende Abbildung zeigt die Komponenten der Confluent Kafka-Plattform.



Überblick über die Event-Streaming-Technologie von Confluent

Der Kern der Confluent Platform ist "Apache Kafka" , die beliebteste Open-Source-Plattform für verteiltes Streaming. Die wichtigsten Funktionen von Kafka sind:

- Veröffentlichen und abonnieren Sie Datensatz-Streams.
- Speichern Sie Datensatzströme fehlertolerant.
- Verarbeiten Sie Datensatzströme.

Die Confluent Platform umfasst standardmäßig auch Schema Registry, REST Proxy, insgesamt über 100 vorgefertigte Kafka-Konnektoren und ksqlDB.

Übersicht über die Enterprise-Funktionen der Confluent-Plattform

- **Confluent-Kontrollzentrum.** Ein GUI-basiertes System zur Verwaltung und Überwachung von Kafka. Es ermöglicht Ihnen die einfache Verwaltung von Kafka Connect und das Erstellen, Bearbeiten und Verwalten von Verbindungen zu anderen Systemen.
- **Confluent für Kubernetes.** Confluent für Kubernetes ist ein Kubernetes-Operator. Kubernetes-Operatoren erweitern die Orchestrierungsfunktionen von Kubernetes, indem sie die einzigartigen Funktionen und Anforderungen für eine bestimmte Plattformanwendung bereitstellen. Für die Confluent Platform bedeutet dies eine erhebliche Vereinfachung des Bereitstellungsprozesses von Kafka auf Kubernetes und die Automatisierung typischer Aufgaben im Lebenszyklus der Infrastruktur.
- **Konfluente Konnektoren zu Kafka.** Konnektoren verwenden die Kafka Connect-API, um Kafka mit anderen Systemen wie Datenbanken, Schlüssel-Wert-Speichern, Suchindizes und Dateisystemen zu verbinden. Confluent Hub verfügt über herunterladbare Konnektoren für die gängigsten Datenquellen und -senken, einschließlich vollständig getesteter und unterstützter Versionen dieser Konnektoren mit Confluent Platform. Weitere Details finden Sie "hier," .
- **Selbstausgleichende Cluster.** Bietet automatisierten Lastausgleich, Fehlererkennung und Selbstheilung. Es bietet Unterstützung für das Hinzufügen oder Außerbetriebnehmen von Brokern nach Bedarf, ohne dass eine manuelle Anpassung erforderlich ist.
- **Konfluente Clusterverknüpfung.** Verbindet Cluster direkt miteinander und spiegelt Themen von einem Cluster zum anderen über eine Linkbrücke. Die Clusterverknüpfung vereinfacht die Einrichtung von Multi-Datacenter-, Multi-Cluster- und Hybrid-Cloud-Bereitstellungen.
- **Confluent automatischer Datenausgleich.** Überwacht Ihren Cluster hinsichtlich der Anzahl der Broker, der Größe der Partitionen, der Anzahl der Partitionen und der Anzahl der Leader innerhalb des Clusters. Sie können Daten verschieben, um eine gleichmäßige Arbeitslast in Ihrem Cluster zu erreichen, und gleichzeitig den Datenverkehr drosseln, um die Auswirkungen auf die Produktionsarbeitslasten während der Neuverteilung zu minimieren.
- **Konfluenter Replikator.** Macht es einfacher als je zuvor, mehrere Kafka-Cluster in mehreren Rechenzentren zu verwalten.
- **Stufenspeicher.** Bietet Optionen zum Speichern großer Mengen von Kafka-Daten bei Ihrem bevorzugten Cloud-Anbieter und reduziert so den Betriebsaufwand und die Kosten. Mit Tiered Storage können Sie Daten auf kostengünstigem Objektspeicher aufbewahren und Broker nur dann skalieren, wenn Sie mehr Rechenressourcen benötigen.
- **Confluent JMS-Client.** Confluent Platform enthält einen JMS-kompatiblen Client für Kafka. Dieser Kafka-Client implementiert die JMS 1.1-Standard-API und verwendet Kafka-Broker als Backend. Dies ist nützlich, wenn Sie über ältere Anwendungen verfügen, die JMS verwenden, und Sie den vorhandenen JMS-Nachrichtenbroker durch Kafka ersetzen möchten.
- **Confluent MQTT-Proxy.** Bietet eine Möglichkeit, Daten von MQTT-Geräten und -Gateways direkt an Kafka zu veröffentlichen, ohne dass ein MQTT-Broker dazwischengeschaltet werden muss.
- **Confluent-Sicherheits-Plugins.** Confluent-Sicherheits-Plugins werden verwendet, um verschiedenen Tools und Produkten der Confluent-Plattform Sicherheitsfunktionen hinzuzufügen. Derzeit ist ein Plug-In für den Confluent REST-Proxy verfügbar, das bei der Authentifizierung eingehender Anfragen hilft und den authentifizierten Auftraggeber an Anfragen an Kafka weitergibt. Dadurch können Confluent REST-Proxy-

Clients die Multitenant-Sicherheitsfunktionen des Kafka-Brokers nutzen.

Konfluente Überprüfung

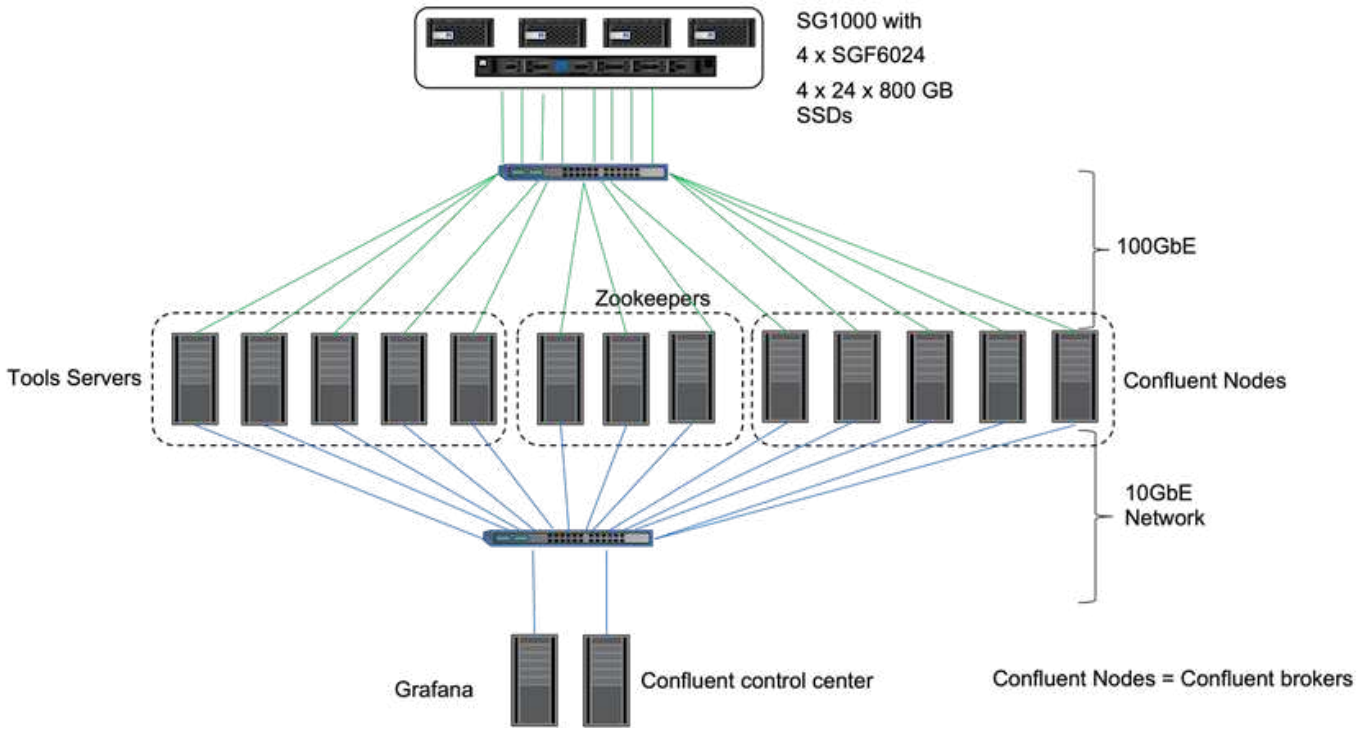
Wir haben die Überprüfung mit Confluent Platform 6.2 Tiered Storage in NetApp StorageGRID durchgeführt. Die Teams von NetApp und Confluent arbeiteten gemeinsam an dieser Verifizierung und führten die für die Verifizierung erforderlichen Testfälle aus.

Einrichtung der Confluent-Plattform

Zur Überprüfung haben wir das folgende Setup verwendet.

Zur Überprüfung verwendeten wir drei Zookeeper, fünf Broker, fünf Testskript-Ausführungsserver, benannte Tool-Server mit 256 GB RAM und 16 CPUs. Für den NetApp -Speicher haben wir StorageGRID mit einem SG1000-Load Balancer mit vier SGF6024 verwendet. Die Speicher und Broker wurden über 100GbE-Verbindungen verbunden.

Die folgende Abbildung zeigt die Netzwerktopologie der für die Confluent-Verifizierung verwendeten Konfiguration.



Die Tool-Server fungieren als Anwendungsclients, die Anfragen an Confluent-Knoten senden.

Confluent-Tiered-Storage-Konfiguration

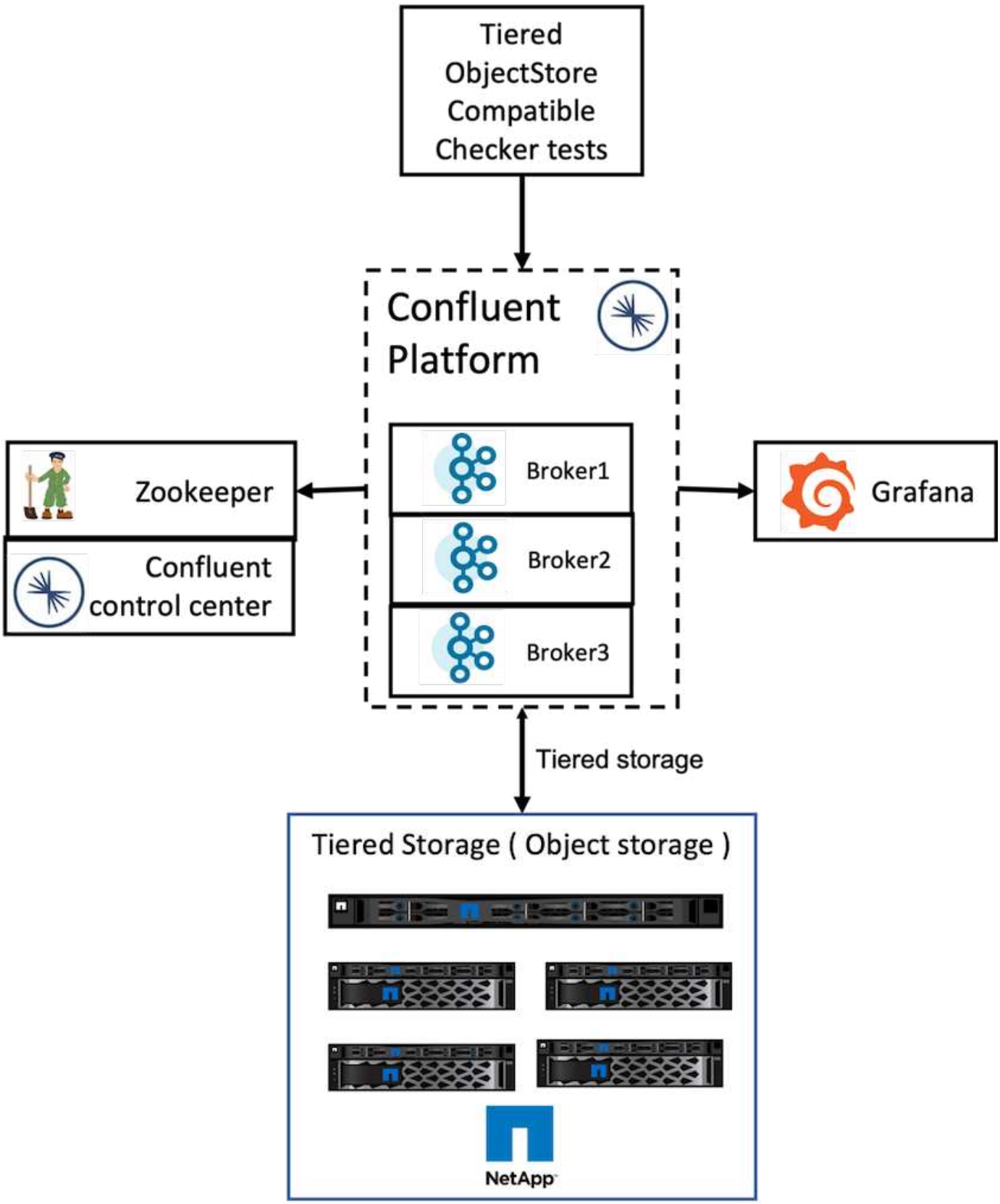
Die mehrstufige Speicherkonfiguration erfordert die folgenden Parameter in Kafka:

```
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
```

Zur Überprüfung haben wir StorageGRID mit dem HTTP-Protokoll verwendet, aber auch HTTPS funktioniert. Der Zugriffsschlüssel und der geheime Schlüssel werden in der Datei mit dem angegebenen Namen gespeichert. `confluent.tier.s3.cred.file.path` Parameter.

NetApp Objektspeicher – StorageGRID

Zur Überprüfung haben wir die Single-Site-Konfiguration in StorageGRID konfiguriert.



Verifizierungstests

Zur Verifizierung haben wir die folgenden fünf Testfälle durchgeführt. Diese Tests werden auf dem Trogdor-Framework ausgeführt. Bei den ersten beiden handelte es sich um Funktionstests und bei den restlichen drei um Leistungstests.

Korrektheitstest des Objektspeichers

Dieser Test ermittelt, ob alle grundlegenden Vorgänge (z. B. Get/Put/Delete) der Objektspeicher-API entsprechend den Anforderungen des mehrstufigen Speichers gut funktionieren. Es handelt sich um einen grundlegenden Test, den jeder Objektspeicherdienst vor den folgenden Tests bestehen sollte. Es handelt sich um einen Aussagetest, der entweder bestanden oder nicht bestanden wird.

Korrektheitstest der Tiering-Funktionalität

Dieser Test ermittelt mit einem Assertivtest, der entweder erfolgreich ist oder fehlschlägt, ob die End-to-End-Tiered-Storage-Funktionalität gut funktioniert. Der Test erstellt ein Testthema, das standardmäßig mit aktiviertem Tiering und stark reduzierter Hotset-Größe konfiguriert ist. Es erzeugt einen Ereignisstrom zum neu erstellten Testthema, wartet darauf, dass die Broker die Segmente im Objektspeicher archivieren, verbraucht dann den Ereignisstrom und überprüft, ob der verbrauchte Strom mit dem erzeugten Strom übereinstimmt. Die Anzahl der an den Ereignisstrom gesendeten Nachrichten ist konfigurierbar, sodass der Benutzer je nach Testbedarf eine ausreichend große Arbeitslast generieren kann. Die reduzierte Hotset-Größe stellt sicher, dass die Abrufe des Verbrauchers außerhalb des aktiven Segments nur aus dem Objektspeicher erfolgen. Dies hilft beim Testen der Richtigkeit des Objektspeichers für Lesevorgänge. Wir haben diesen Test mit und ohne Fehlerinjektion im Objektspeicher durchgeführt. Wir haben einen Knotenausfall simuliert, indem wir den Service Manager-Dienst in einem der Knoten in StorageGRID gestoppt und überprüft haben, ob die End-to-End-Funktionalität mit dem Objektspeicher funktioniert.

Benchmark für den Tier-Abruf

Dieser Test validierte die Leseleistung des mehrstufigen Objektspeichers und überprüfte die Range-Fetch-Leseanforderungen unter hoher Last von Segmenten, die durch den Benchmark generiert wurden. In diesem Benchmark hat Confluent benutzerdefinierte Clients entwickelt, um die Tier-Fetch-Anfragen zu erfüllen.

Benchmark für die Arbeitslast „Produzieren und Konsumieren“

Dieser Test erzeugte durch die Archivierung von Segmenten indirekt eine Schreiblast im Objektspeicher. Die Lesearbeitslast (gelesene Segmente) wurde aus dem Objektspeicher generiert, als Verbrauchergruppen die Segmente abgerufen haben. Diese Arbeitslast wurde durch das Testskript generiert. Dieser Test überprüfte die Leistung beim Lesen und Schreiben im Objektspeicher in parallelen Threads. Wir haben mit und ohne Fehlerinjektion im Objektspeicher getestet, wie wir es für den Korrektheitstest der Tiering-Funktionalität getan haben.

Benchmark für die Aufbewahrungsarbeitslast

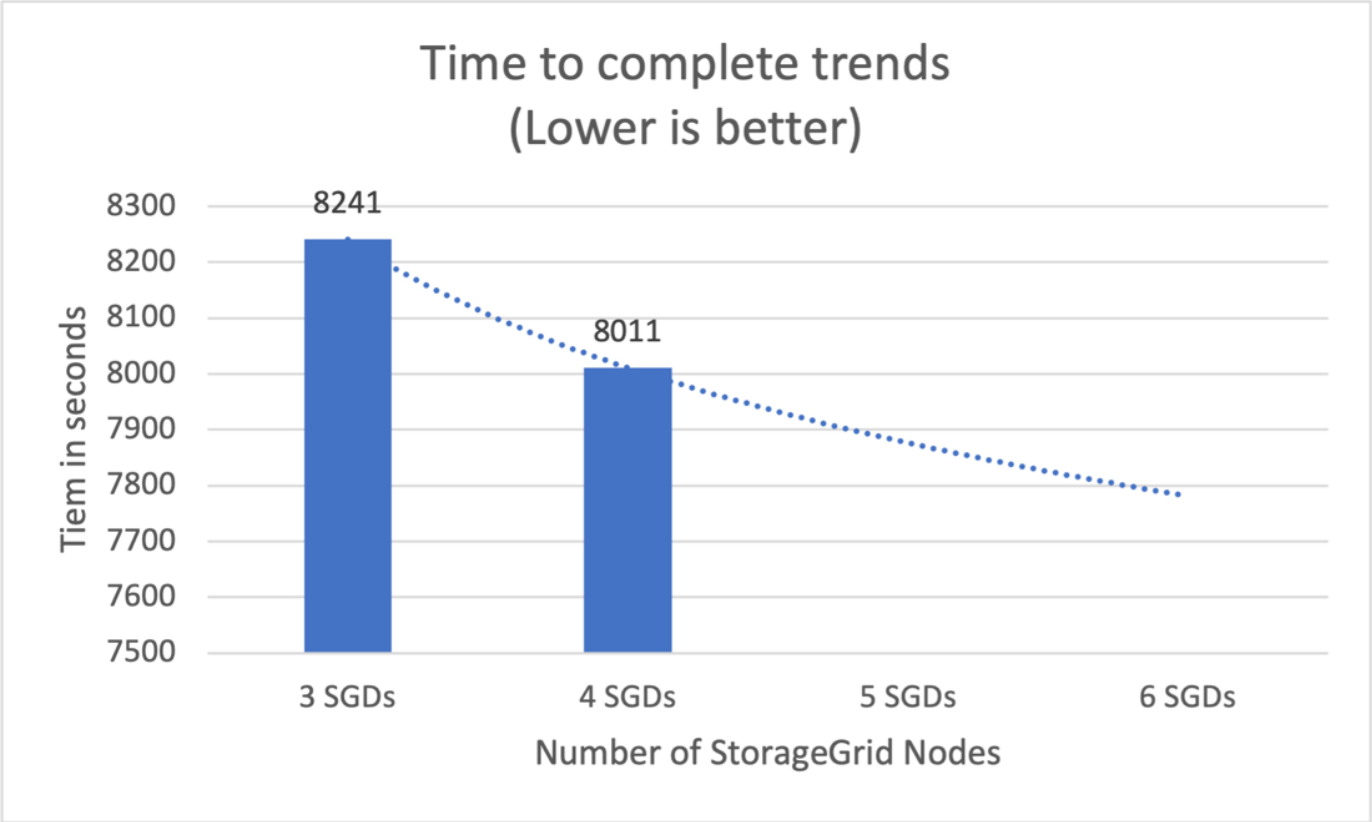
Dieser Test prüfte die Löschleistung eines Objektspeichers unter einer hohen Themenaufbewahrungslast. Der Aufbewahrungsaufwand wurde mithilfe eines Testskripts generiert, das viele Nachrichten parallel zu einem Testthema produziert. Das Testthema war die Konfiguration mit einer aggressiven größen- und zeitbasierten Aufbewahrungseinstellung, die dazu führte, dass der Ereignisstrom kontinuierlich aus dem Objektspeicher gelöscht wurde. Anschließend wurden die Segmente archiviert. Dies führte zu einer großen Anzahl von Löschungen im Objektspeicher durch den Broker und zur Erfassung der Leistung der Löschvorgänge im Objektspeicher.

Leistungstests mit Skalierbarkeit

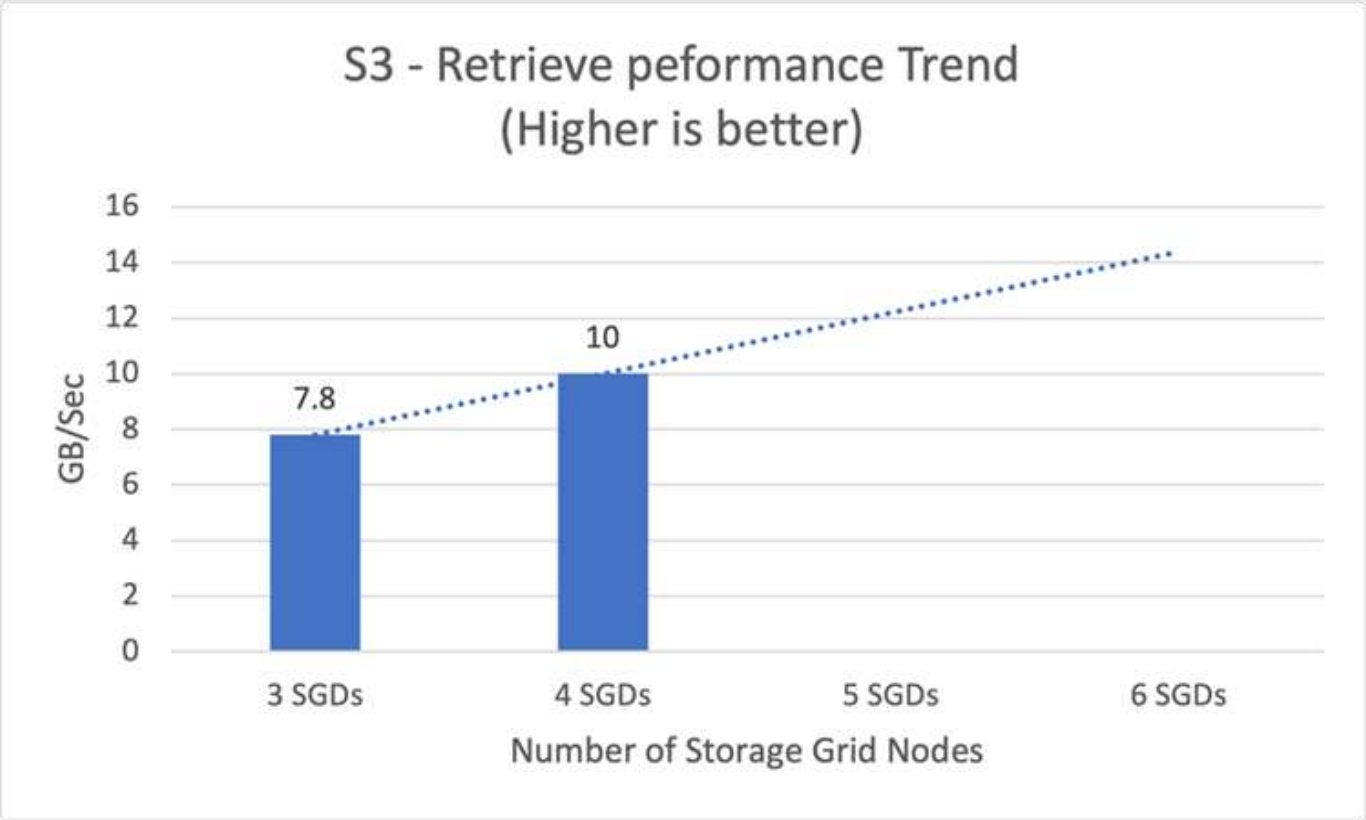
Wir haben die Tiered-Storage-Tests mit drei bis vier Knoten für Producer- und Consumer-Workloads mit dem NetApp StorageGRID Setup durchgeführt. Unseren Tests zufolge waren die Zeit bis zur Fertigstellung und die Leistungsergebnisse direkt proportional zur Anzahl der StorageGRID Knoten. Für die Einrichtung von StorageGRID waren

mindestens drei Knoten erforderlich.

- Die Zeit zum Abschließen des Produktions- und Verbrauchervorgangs verringerte sich linear, wenn die Anzahl der Speicherknoten zunahm.



- Die Leistung für den S3-Abrufvorgang stieg linear basierend auf der Anzahl der StorageGRID Knoten. StorageGRID unterstützt bis zu 200 StorgeGRID-Knoten.



Confluent S3-Anschluss

Der Amazon S3 Sink-Connector exportiert Daten aus Apache Kafka-Themen in S3-Objekte im Avro-, JSON- oder Bytes-Format. Der Amazon S3-Sink-Connector fragt regelmäßig Daten von Kafka ab und lädt sie wiederum zu S3 hoch. Ein Partitionierer wird verwendet, um die Daten jeder Kafka-Partition in Blöcke aufzuteilen. Jeder Datenblock wird als S3-Objekt dargestellt. Der Schlüsselname kodiert das Thema, die Kafka-Partition und den Start-Offset dieses Datenblocks.

In diesem Setup zeigen wir Ihnen, wie Sie Themen im Objektspeicher von Kafka direkt mithilfe des Kafka s3-Sink-Connectors lesen und schreiben. Für diesen Test haben wir einen eigenständigen Confluent-Cluster verwendet, dieses Setup ist jedoch auch auf einen verteilten Cluster anwendbar.

1. Laden Sie Confluent Kafka von der Confluent-Website herunter.
2. Entpacken Sie das Paket in einen Ordner auf Ihrem Server.
3. Exportieren Sie zwei Variablen.

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

4. Für ein eigenständiges Confluent Kafka-Setup erstellt der Cluster einen temporären Stammordner in `/tmp` . Es erstellt außerdem Zookeeper, Kafka, ein Schema-Registry, Connect, einen KSQL-Server und Control-Center-Ordner und kopiert die jeweiligen Konfigurationsdateien von `$CONFLUENT_HOME` . Siehe das folgende Beispiel:

```
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. Konfigurieren Sie Zookeeper. Wenn Sie die Standardparameter verwenden, müssen Sie nichts ändern.

```
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:~#
```

In der obigen Konfiguration haben wir die `server. xxx` Eigentum. Standardmäßig benötigen Sie drei Zookeeper für die Kafka-Leader-Auswahl.

6. Wir haben eine Myid-Datei erstellt in `/tmp/confluent.406980/zookeeper/data` mit einer eindeutigen ID:

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

Wir haben die letzte Nummer der IP-Adressen für die MyID-Datei verwendet. Wir haben Standardwerte für die Konfigurationen Kafka, Connect, Control-Center, Kafka, Kafka-Rest, KSQL-Server und Schema-Registry verwendet.

7. Starten Sie die Kafka-Dienste.

```
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#
```

Für jede Konfiguration gibt es einen Protokollordner, der bei der Fehlerbehebung hilft. In einigen Fällen dauert der Start der Dienste länger. Stellen Sie sicher, dass alle Dienste aktiv sind und ausgeführt werden.

8. Installieren Sie Kafka Connect mit `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#
```

Sie können auch eine bestimmte Version installieren, indem Sie `confluent-hub install confluentinc/kafka-connect-s3:10.0.3` .

9. Standardmäßig `confluentinc-kafka-connect-s3` ist installiert in `/data/confluent/confluent-6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3` .
10. Aktualisieren Sie den Plug-In-Pfad mit dem neuen `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. Stoppen Sie die Confluent-Dienste und starten Sie sie neu.

```
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. Konfigurieren Sie die Zugriffs-ID und den geheimen Schlüssel im `/root/.aws/credentials` Datei.

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

13. Überprüfen Sie, ob der Bucket erreichbar ist.

```
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. Konfigurieren Sie die S3-Sink-Eigenschaftendatei für die S3- und Bucket-Konfiguration.

```
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. Importieren Sie einige Datensätze in den S3-Bucket.

```
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. Laden Sie den S3-Sink-Connector.

```
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. Überprüfen Sie den S3-Sink-Status.

```
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. Überprüfen Sie das Protokoll, um sicherzustellen, dass s3-sink bereit ist, Themen anzunehmen.

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

19. Sehen Sie sich die Themen in Kafka an.

```
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. Überprüfen Sie die Objekte im S3-Bucket.

```
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. Um den Inhalt zu überprüfen, kopieren Sie jede Datei von S3 in Ihr lokales Dateisystem, indem Sie den folgenden Befehl ausführen:

```
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. Um die Datensätze auszudrucken, verwenden Sie avro-tools-1.11.0.1.jar (verfügbar im "Apache-Archive").

```
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 Connectors

Instaclustr unterstützt Kafka Connect Connectors und deren Details - "Weitere Details"Die Instaclustr bietet zusätzliche Konnektoren. "ihre Details"

Konfluente, selbstausgleichende Cluster

Wenn Sie bereits einen Kafka-Cluster verwaltet haben, sind Sie wahrscheinlich mit den Herausforderungen vertraut, die mit der manuellen Neuzuweisung von Partitionen an verschiedene Broker einhergehen, um sicherzustellen, dass die Arbeitslast im gesamten Cluster ausgeglichen ist. Für Organisationen mit großen Kafka-Bereitstellungen kann die Neuordnung großer Datenmengen entmutigend, mühsam und riskant sein, insbesondere wenn unternehmenskritische Anwendungen auf dem Cluster erstellt werden. Allerdings ist der Prozess selbst bei den kleinsten Kafka-Anwendungsfällen zeitaufwändig und anfällig für menschliche Fehler.

In unserem Labor haben wir die Funktion zum selbstausgleichenden Cluster von Confluent getestet, die den Neuausgleich basierend auf Änderungen der Clustertopologie oder ungleichmäßiger Last automatisiert. Der Confluent-Neuausgleichstest hilft dabei, die Zeit zu messen, die zum Hinzufügen eines neuen Brokers benötigt wird, wenn ein Knotenausfall vorliegt oder der Skalierungsknoten einen Neuausgleich der Daten zwischen den Brokern erfordert. In klassischen Kafka-Konfigurationen wächst die Menge der neu auszugleichenden Daten mit dem Wachstum des Clusters, bei mehrstufigem Speicher ist die Neuausgleichung jedoch auf eine kleine Datenmenge beschränkt. Basierend auf unserer Validierung dauert die Neuausrichtung im mehrstufigen Speicher in einer klassischen Kafka-Architektur Sekunden oder Minuten und wächst linear mit dem Wachstum des Clusters.

In selbstausgleichenden Clustern werden Partitionsneuausgleiche vollständig automatisiert, um den Durchsatz von Kafka zu optimieren, die Broker-Skalierung zu beschleunigen und den Betriebsaufwand für den Betrieb eines großen Clusters zu reduzieren. Im stationären Zustand überwachen selbstausgleichende Cluster die Datenabweichung zwischen den Brokern und weisen Partitionen kontinuierlich neu zu, um die Clusterleistung zu optimieren. Beim Hoch- oder Herunterskalieren der Plattform erkennen selbstausgleichende Cluster automatisch das Vorhandensein neuer Broker oder das Entfernen alter Broker und lösen eine anschließende Neuzuweisung der Partition aus. Dadurch können Sie Broker einfach hinzufügen und außer Betrieb nehmen, wodurch Ihre Kafka-Cluster wesentlich elastischer werden. Diese Vorteile ergeben sich ohne manuelle Eingriffe, komplexe Berechnungen oder das Risiko menschlicher Fehler, das bei Partitionsneuzuweisungen normalerweise auftritt. Dadurch werden Datenneuausrichtungen in wesentlich kürzerer Zeit abgeschlossen und Sie können sich auf höherwertige Event-Streaming-Projekte konzentrieren, anstatt Ihre Cluster ständig überwachen zu müssen.

Instaclustr unterstützt außerdem Funktionen zur automatischen Neuausrichtung und wurde bereits für mehrere Kunden implementiert.

Best Practice-Richtlinien

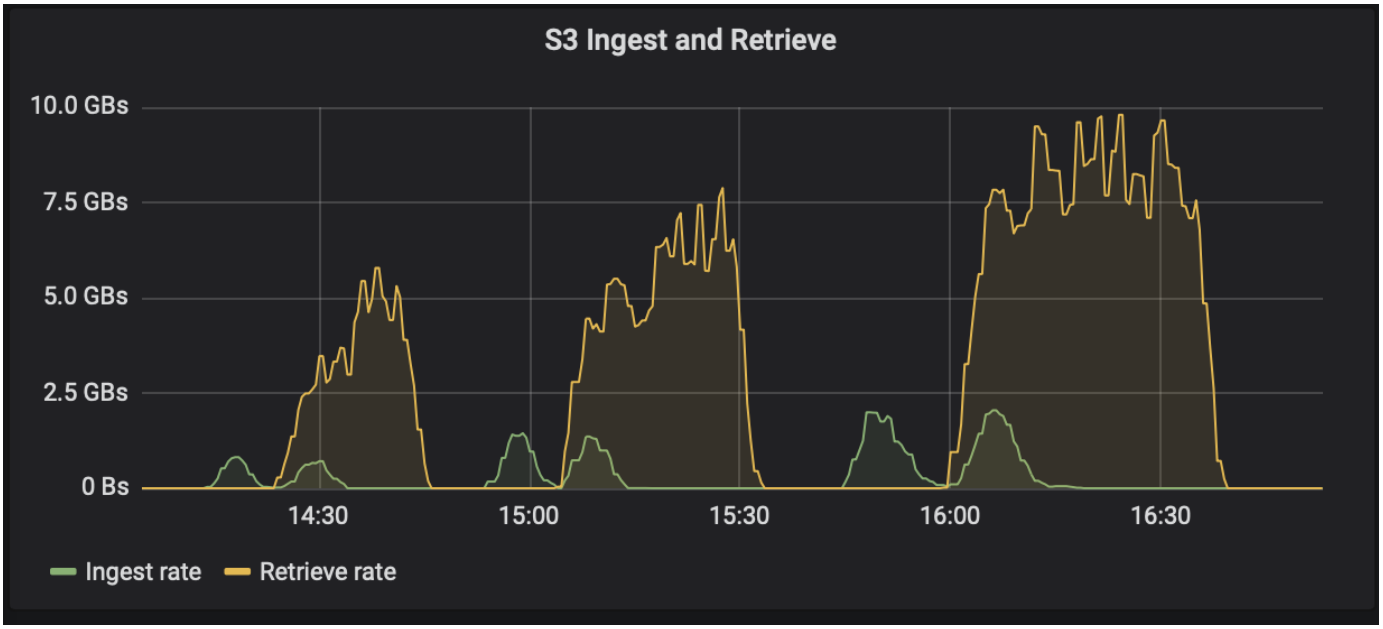
In diesem Abschnitt werden die Erkenntnisse aus dieser Zertifizierung vorgestellt.

- Basierend auf unserer Validierung ist der S3-Objektspeicher für Confluent am besten geeignet, um Daten aufzubewahren.
- Wir können ein Hochdurchsatz-SAN (insbesondere FC) verwenden, um die Hot Data des Brokers oder die lokale Festplatte aufzubewahren, da in der mehrstufigen Speicherkonfiguration von Confluent die Größe der im Datenverzeichnis des Brokers gespeicherten Daten auf der Segmentgröße und der

Aufbewahrungszeit basiert, wenn die Daten in den Objektspeicher verschoben werden.

- Objektspeicher bieten eine bessere Leistung, wenn segment.bytes höher ist; wir haben 512 MB getestet.
- In Kafka wird die Länge des Schlüssels oder Wertes (in Bytes) für jeden zum Thema erstellten Datensatz durch die `length.key.value` Parameter. Für StorageGRID wurde die Leistung beim Aufnehmen und Abrufen von S3-Objekten auf höhere Werte erhöht. Beispielsweise ermöglichten 512 Bytes einen Abruf von 5,8 GBps, 1024 Bytes einen S3-Abruf von 7,5 GBps und 2048 Bytes fast 10 GBps.

Die folgende Abbildung zeigt die Aufnahme und den Abruf von S3-Objekten basierend auf `length.key.value` .



- **Kafka-Tuning.** Um die Leistung des mehrstufigen Speichers zu verbessern, können Sie TierFetcherNumThreads und TierArchiverNumThreads erhöhen. Als allgemeine Richtlinie sollten Sie TierFetcherNumThreads erhöhen, um es an die Anzahl der physischen CPU-Kerne anzupassen, und TierArchiverNumThreads auf die Hälfte der Anzahl der CPU-Kerne erhöhen. Wenn Sie beispielsweise über eine Maschine mit acht physischen Kernen verfügen, legen Sie in den Servereigenschaften confluent.tier.fetcher.num.threads = 8 und confluent.tier.archiver.num.threads = 4 fest.
- **Zeitintervall für Themenlöschungen.** Wenn ein Thema gelöscht wird, beginnt das Löschen der Protokollsegmentdateien im Objektspeicher nicht sofort. Vielmehr gibt es ein Zeitintervall mit einem Standardwert von 3 Stunden, bevor die Löschung dieser Dateien erfolgt. Sie können die Konfiguration confluent.tier.topic.delete.check.interval.ms ändern, um den Wert dieses Intervalls zu ändern. Wenn Sie ein Thema oder einen Cluster löschen, können Sie die Objekte im jeweiligen Bucket auch manuell löschen.
- **ACLs zu internen Themen des Tiered Storage.** Eine empfohlene Best Practice für lokale Bereitstellungen besteht darin, einen ACL-Autorisierer für die internen Themen zu aktivieren, die für mehrstufigen Speicher verwendet werden. Legen Sie ACL-Regeln fest, um den Zugriff auf diese Daten auf den Broker-Benutzer zu beschränken. Dies sichert die internen Themen und verhindert den unbefugten Zugriff auf Tiered-Storage-Daten und Metadaten.

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

Ersetzen Sie den Benutzer `<kafka>` mit dem tatsächlichen Broker-Prinzipal in Ihrer Bereitstellung.

Beispielsweise der Befehl `confluent-tier-state` legt ACLs für das interne Thema für mehrstufigen Speicher fest. Derzeit gibt es nur ein einziges internes Thema zum Thema Tiered Storage. Das Beispiel erstellt eine ACL, die dem Kafka-Prinzip die Berechtigung für alle Vorgänge am internen Thema erteilt.

Größen

Die Kafka-Dimensionierung kann mit vier Konfigurationsmodi durchgeführt werden: einfach, granular, umgekehrt und Partitionen.

Einfach

Der einfache Modus eignet sich für Erstbenutzer von Apache Kafka oder Anwendungsfälle im Frühstadium. Für diesen Modus geben Sie Anforderungen wie Durchsatz-MBps, Lese-Fanout, Aufbewahrung und den Prozentsatz der Ressourcennutzung an (60 % ist der Standard). Sie geben auch die Umgebung ein, z. B. vor Ort (Bare-Metal, VMware, Kubernetes oder OpenStack) oder in der Cloud. Basierend auf diesen Informationen liefert die Dimensionierung eines Kafka-Clusters die Anzahl der Server, die für den Broker, den Zookeeper, die Apache Kafka Connect Worker, das Schema-Register, einen REST-Proxy, ksqlDB und das Confluent-Kontrollzentrum erforderlich sind.

Berücksichtigen Sie bei mehrstufigem Speicher den granularen Konfigurationsmodus zur Größenbestimmung eines Kafka-Clusters. Der Granularmodus eignet sich für erfahrene Apache Kafka-Benutzer oder klar definierte Anwendungsfälle. In diesem Abschnitt wird die Dimensionierung für Produzenten, Stream-Prozessoren und Konsumenten beschrieben.

Produzenten

Um die Produzenten für Apache Kafka zu beschreiben (z. B. einen nativen Client, einen REST-Proxy oder einen Kafka-Connector), geben Sie die folgenden Informationen an:

- **Name.** Funke.
- **Produzententyp.** Anwendung oder Dienst, Proxy (REST, MQTT, andere) und vorhandene Datenbank (RDBMS, NOSQL, andere). Sie können auch „Ich weiß nicht“ auswählen.
- **Durchschnittlicher Durchsatz.** In Ereignissen pro Sekunde (z. B. 1.000.000).
- **Spitzendurchsatz.** In Ereignissen pro Sekunde (z. B. 4.000.000).
- **Durchschnittliche Nachrichtengröße.** In Bytes, unkomprimiert (max. 1 MB; beispielsweise 1000).
- **Nachrichtenformat.** Zu den Optionen gehören Avro, JSON, Protokollpuffer, Binär, Text, „Ich weiß nicht“ und andere.
- **Replikationsfaktor.** Optionen sind 1, 2, 3 (Confluent-Empfehlung), 4, 5 oder 6.
- **Aufbewahrungszeit.** Eines Tages (zum Beispiel). Wie lange sollen Ihre Daten in Apache Kafka gespeichert werden? Geben Sie -1 mit einer beliebigen Einheit für eine unendliche Zeit ein. Der Rechner geht bei unbegrenzter Aufbewahrung von einer Aufbewahrungsdauer von 10 Jahren aus.
- Aktivieren Sie das Kontrollkästchen „Tiered Storage aktivieren, um die Anzahl der Broker zu verringern und unbegrenzten Speicher zu ermöglichen?“
- Wenn die mehrstufige Speicherung aktiviert ist, steuern die Aufbewahrungsfelder den Hotset der Daten, der lokal auf dem Broker gespeichert wird. Die Felder für die Archivaufbewahrung steuern, wie lange Daten

im Archivobjektspeicher gespeichert werden.

- **Archivspeicherung.** Ein Jahr (zum Beispiel). Wie lange sollen Ihre Daten im Archivspeicher aufbewahrt werden? Geben Sie -1 mit einer beliebigen Einheit für eine unendliche Dauer ein. Der Rechner geht bei unbegrenzter Aufbewahrung von einer Aufbewahrungsdauer von 10 Jahren aus.
- **Wachstumsmultiplikator.** 1 (zum Beispiel). Wenn der Wert dieses Parameters auf dem aktuellen Durchsatz basiert, setzen Sie ihn auf 1. Um die Größe auf Grundlage zusätzlichen Wachstums anzupassen, legen Sie diesen Parameter auf einen Wachstumsmultiplikator fest.
- **Anzahl der Produzenteninstanzen.** 10 (zum Beispiel). Wie viele Produzenteninstanzen werden ausgeführt? Diese Eingabe ist erforderlich, um die CPU-Auslastung in die Größenberechnung einzubeziehen. Ein leerer Wert zeigt an, dass die CPU-Auslastung nicht in die Berechnung einbezogen wird.

Basierend auf dieser Beispieleingabe hat die Größenanpassung die folgenden Auswirkungen auf die Hersteller:

- Durchschnittlicher Durchsatz in unkomprimierten Bytes: 1 GBps. Spitzendurchsatz in unkomprimierten Bytes: 4 GB/s. Durchschnittlicher Durchsatz in komprimierten Bytes: 400 MB/s. Spitzendurchsatz in komprimierten Bytes: 1,6 GB/s. Dies basiert auf einer Standardkomprimierungsrate von 60 % (Sie können diesen Wert ändern).
    - Gesamter On-Broker-Hotset-Speicher erforderlich: 31.104 TB, einschließlich Replikation, komprimiert. Gesamter Off-Broker-Archivspeicherbedarf: 378.432 TB, komprimiert. Verwenden"https://fusion.netapp.com" zur StorageGRID -Dimensionierung.

Stream-Prozessoren müssen ihre Anwendungen oder Dienste beschreiben, die Daten von Apache Kafka verbrauchen und wieder in Apache Kafka produzieren. In den meisten Fällen werden diese in ksqlDB oder Kafka Streams erstellt.

- **Name.** Spark-Streamer.
- **Bearbeitungszeit.** Wie lange braucht dieser Prozessor, um eine einzelne Nachricht zu verarbeiten?
    - 1 ms (einfache, zustandslose Transformation) [Beispiel], 10 ms (zustandsbehafteter In-Memory-Vorgang).
    - 100 ms (zustandsbehafteter Netzwerk- oder Festplattenvorgang), 1000 ms (REST-Aufruf eines Drittanbieters).
    - Ich habe diesen Parameter getestet und weiß genau, wie lange es dauert.
- **Ausgabeaufbewahrung.** 1 Tag (Beispiel). Ein Stream-Prozessor gibt seine Ausgabe an Apache Kafka zurück. Wie lange sollen diese Ausgabedaten in Apache Kafka gespeichert werden? Geben Sie -1 mit einer beliebigen Einheit für eine unendliche Dauer ein.
- Aktivieren Sie das Kontrollkästchen „Tiered Storage aktivieren, um die Anzahl der Broker zu verringern und unbegrenzten Speicher zu ermöglichen?“
- **Archivspeicherung.** 1 Jahr (zum Beispiel). Wie lange sollen Ihre Daten im Archivspeicher aufbewahrt werden? Geben Sie -1 mit einer beliebigen Einheit für eine unendliche Dauer ein. Der Rechner geht bei unbegrenzter Aufbewahrung von einer Aufbewahrungsdauer von 10 Jahren aus.
- **Prozentsatz der Ausgabeweiterleitung.** 100 (zum Beispiel). Ein Stream-Prozessor gibt seine Ausgabe an Apache Kafka zurück. Welcher Prozentsatz des eingehenden Durchsatzes wird zurück an Apache Kafka ausgegeben? Wenn beispielsweise der eingehende Durchsatz 20 MBps beträgt und dieser Wert 10 ist, beträgt der Ausgangsdurchsatz 2 MBps.
- Aus welchen Anwendungen wird dies gelesen? Wählen Sie „Spark“, den Namen, der bei der herstellertypbasierten Größenbestimmung verwendet wird. Basierend auf den obigen Eingaben können Sie die folgenden Auswirkungen der Größenanpassung auf Stream-Prozessor-Instanzen und

Themenpartitionsschätzungen erwarten:

- Diese Stream-Prozessor-Anwendung erfordert die folgende Anzahl von Instanzen. Die eingehenden Themen erfordern wahrscheinlich auch diese Anzahl an Partitionen. Wenden Sie sich an Confluent, um diesen Parameter zu bestätigen.
  - 1.000 für durchschnittlichen Durchsatz ohne Wachstumsmultiplikator
  - 4.000 für Spitzendurchsatz ohne Wachstumsmultiplikator
  - 1.000 für durchschnittlichen Durchsatz mit einem Wachstumsmultiplikator
  - 4.000 für Spitzendurchsatz mit Wachstumsmultiplikator

Verbraucher

Beschreiben Sie Ihre Anwendungen oder Dienste, die Daten von Apache Kafka nutzen und nicht wieder in Apache Kafka produzieren; beispielsweise ein nativer Client oder Kafka Connector.

- **Name.** Spark-Verbraucher.
- **Bearbeitungszeit.** Wie lange braucht dieser Verbraucher, um eine einzelne Nachricht zu verarbeiten?
  - 1 ms (z. B. eine einfache und zustandslose Aufgabe wie das Protokollieren)
  - 10 ms (schnelles Schreiben in einen Datenspeicher)
  - 100 ms (langsames Schreiben in einen Datenspeicher)
  - 1000 ms (REST-Aufruf eines Drittanbieters)
  - Ein anderer Benchmark-Prozess mit bekannter Dauer.
- **Verbrauchertyp.** Anwendung, Proxy oder Sink zu einem vorhandenen Datenspeicher (RDBMS, NoSQL, andere).
- Aus welchen Anwendungen wird dies gelesen? Verbinden Sie diesen Parameter mit der zuvor ermittelten Produzenten- und Streamgröße.

Basierend auf den obigen Eingaben müssen Sie die Größe für Verbraucherinstanzen und Themenpartitionsschätzungen bestimmen. Eine Consumer-Anwendung erfordert die folgende Anzahl von Instanzen.

- 2.000 für durchschnittlichen Durchsatz, kein Wachstumsmultiplikator
- 8.000 für Spitzendurchsatz, kein Wachstumsmultiplikator
- 2.000 für durchschnittlichen Durchsatz, einschließlich Wachstumsmultiplikator
- 8.000 für Spitzendurchsatz, einschließlich Wachstumsmultiplikator

Die eingehenden Themen benötigen wahrscheinlich auch diese Anzahl von Partitionen. Wenden Sie sich zur Bestätigung an Confluent.

Zusätzlich zu den Anforderungen für Produzenten, Stream-Prozessoren und Konsumenten müssen Sie die folgenden zusätzlichen Anforderungen erfüllen:

- **Zeit zum Wiederaufbau.** Zum Beispiel 4 Stunden. Wenn ein Apache Kafka-Broker-Host ausfällt, seine Daten verloren gehen und ein neuer Host bereitgestellt wird, um den ausgefallenen Host zu ersetzen, wie schnell muss sich dieser neue Host selbst wiederherstellen? Lassen Sie diesen Parameter leer, wenn der Wert unbekannt ist.
- **Ressourcennutzungsziel (Prozentsatz).** Zum Beispiel 60. Wie ausgelastet sollen Ihre Hosts bei durchschnittlichem Durchsatz sein? Confluent empfiehlt eine Auslastung von 60 %, es sei denn, Sie

verwenden selbstausgleichende Confluent-Cluster. In diesem Fall kann die Auslastung höher sein.

Beschreiben Sie Ihre Umgebung

- **In welcher Umgebung wird Ihr Cluster ausgeführt?** Amazon Web Services, Microsoft Azure, Google Cloud Platform, Bare-Metal vor Ort, VMware vor Ort, OpenStack vor Ort oder Kubernates vor Ort?
- **Hostdetails.** Anzahl der Kerne: 48 (zum Beispiel), Netzwerkkartentyp (10GbE, 40GbE, 16GbE, 1GbE oder ein anderer Typ).
- **Speichervolumes.** Host: 12 (zum Beispiel). Wie viele Festplatten oder SSDs werden pro Host unterstützt? Confluent empfiehlt 12 Festplatten pro Host.
- **Speicherkapazität/-volumen (in GB).** 1000 (zum Beispiel). Wie viel Speicherplatz in Gigabyte kann ein einzelnes Volume speichern? Confluent empfiehlt 1-TB-Festplatten.
- **Speicherkonfiguration.** Wie werden Speichervolumes konfiguriert? Confluent empfiehlt RAID10, um alle Confluent-Funktionen zu nutzen. JBOD, SAN, RAID 1, RAID 0, RAID 5 und andere Typen werden ebenfalls unterstützt.
- **Durchsatz einzelner Datenträger (MBps).** 125 (zum Beispiel). Wie schnell kann ein einzelnes Speichervolumen in Megabyte pro Sekunde lesen oder schreiben? Confluent empfiehlt Standardfestplatten, die normalerweise einen Durchsatz von 125 MB/s haben.
- **Speicherkapazität (GB).** 64 (zum Beispiel).

Nachdem Sie Ihre Umgebungsvariablen ermittelt haben, wählen Sie „Size my Cluster“ (Größe meines Clusters festlegen). Basierend auf den oben angegebenen Beispielparametern haben wir die folgende Dimensionierung für Confluent Kafka ermittelt:

- **Apache Kafka.** Anzahl der Makler: 22. Ihr Cluster ist speichergebunden. Erwägen Sie die Aktivierung von Tiered Storage, um die Anzahl Ihrer Hosts zu verringern und unbegrenzten Speicherplatz zu ermöglichen.
- **Apache ZooKeeper.** Anzahl: 5; Apache Kafka Connect Workers: Anzahl: 2; Schema Registry: Anzahl: 2; REST-Proxy: Anzahl: 2; ksqlDB: Anzahl: 2; Confluent Control Center: Anzahl: 1.

Verwenden Sie den umgekehrten Modus für Plattformteams ohne einen Anwendungsfall im Sinn. Verwenden Sie den Partitionsmodus, um zu berechnen, wie viele Partitionen ein einzelnes Thema benötigt. Sehen https://eventsizer.io zur Größenbestimmung basierend auf den Reverse- und Partitionsmodi.

Abschluss

Dieses Dokument enthält Best-Practice-Richtlinien für die Verwendung von Confluent Tiered Storage mit NetApp -Speicher, einschließlich Verifizierungstests, Leistungsergebnissen für Tiered Storage, Optimierung, Confluent S3-Konnektoren und der Selbstausgleichsfunktion. Unter Berücksichtigung von ILM-Richtlinien, Confluent-Leistung mit mehreren Leistungstests zur Überprüfung und branchenüblichen S3-APIs ist der NetApp StorageGRID Objektspeicher die optimale Wahl für Confluent-Tiered-Storage.

Wo Sie weitere Informationen finden

Weitere Informationen zu den in diesem Dokument beschriebenen Informationen finden Sie in den folgenden Dokumenten und/oder auf den folgenden Websites:

- Was ist Apache Kafka

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

- NetApp Produktdokumentation

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

- S3-Sink-Parameterdetails

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

- Apache Kafka

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

- Unbegrenzter Speicherplatz in der Confluent-Plattform

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

- Confluent Tiered Storage – Best Practices und Dimensionierung

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

- Amazon S3-Sink-Connector für die Confluent-Plattform

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

- Kafka-Dimensionierung

  "https://eventsizer.io"

- StorageGRID -Dimensionierung

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

- Kafka-Anwendungsfälle

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

- Selbstausgleichende Kafka-Cluster in der Confluent-Plattform 6.0

  "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/"

- Beispielhafte Instaclustr-Kunden und Details zu ihren Anwendungsfällen

  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/