Confluent Kafka 모범 사례

NetApp artificial intelligence solutions

NetApp
February 12, 2026

목차

Confluent Kafka 모범 사례

TR-4912: NetApp 사용한 Confluent Kafka 계층형 스토리지에 대한 모범 사례 가이드라인

Karthikeyan Nagalingam, Joseph Kandatilparambil, NetApp Rankesh Kumar, Confluent

Apache Kafka는 하루에 수조 개의 이벤트를 처리할 수 있는 커뮤니티 분산형 이벤트 스트리밍 플랫폼입니다. 처음에는 메시징 큐로 구상되었지만, 카프카는 분산 커밋 로그의 추상화를 기반으로 합니다. 카프카는 2011년 LinkedIn에서 개발되어 오픈 소스로 공개된 이후 메시지 큐에서 본격적인 이벤트 스트리밍 플랫폼으로 발전했습니다. Confluent는 Confluent Platform을 통해 Apache Kafka를 배포합니다. Confluent 플랫폼은 Kafka에 커뮤니티 및 상용 기능을 추가하여 대규모 프로덕션 환경에서 운영자와 개발자 모두의 스트리밍 경험을 향상시키도록 설계되었습니다.

이 문서에서는 NetApp의 개체 스토리지 제품에서 Confluent Tiered Storage를 사용하기 위한 모범 사례 가이드라인을 설명하며, 다음과 같은 내용을 제공합니다.

- NetApp Object Storage를 통한 Confluent 검증 – NetApp StorageGRID
- 계층형 스토리지 성능 테스트
- NetApp 스토리지 시스템에서 Confluent를 위한 모범 사례 가이드라인

Confluent Tiered Storage를 선택해야 하는 이유는 무엇인가요?

Confluent는 특히 빅데이터, 분석 및 스트리밍 워크로드를 위한 다양한 애플리케이션의 기본 실시간 스트리밍 플랫폼이 되었습니다. 계층형 스토리지를 사용하면 사용자가 Confluent 플랫폼에서 컴퓨팅과 스토리지를 분리할 수 있습니다. 이를 통해 데이터 저장의 비용 효율성이 높아지고, 사실상 무한한 양의 데이터를 저장하고 필요에 따라 작업 부하를 확장(또는 축소)할 수 있으며, 데이터 및 테넌트 재조정과 같은 관리 작업이 더 쉬워집니다. S3 호환 스토리지 시스템은 이러한 모든 기능을 활용하여 모든 이벤트를 한곳에서 관리함으로써 데이터를 민주화하고 복잡한 데이터 엔지니어링의 필요성을 없앨 수 있습니다. Kafka에 계층형 스토리지를 사용해야 하는 이유에 대한 자세한 내용은 다음을 확인하세요 ."Confluent의 이 기사" .

NetApp instaclustr는 3.8.1 버전부터 계층형 스토리지를 사용하는 Kafka도 지원합니다. 자세한 내용은 여기를 확인하세요. "Kafka 계층형 스토리지를 사용하는 Instaclust"

계층형 스토리지에 NetApp StorageGRID 사용해야 하는 이유는 무엇인가요?

StorageGRID 는 NetApp 이 제공하는 업계 최고의 객체 스토리지 플랫폼입니다. StorageGRID Amazon Simple Storage Service(S3) API를 비롯한 업계 표준 객체 API를 지원하는 소프트웨어 정의 객체 기반 스토리지 솔루션입니다. StorageGRID 대규모로 비정형 데이터를 저장하고 관리하여 안전하고 내구성 있는 객체 스토리지를 제공합니다. 콘텐츠는 적절한 위치, 적절한 시간, 적절한 저장 계층에 배치되어 워크플로를 최적화하고 전 세계적으로 분산된 리치 미디어의 비용을 절감합니다.

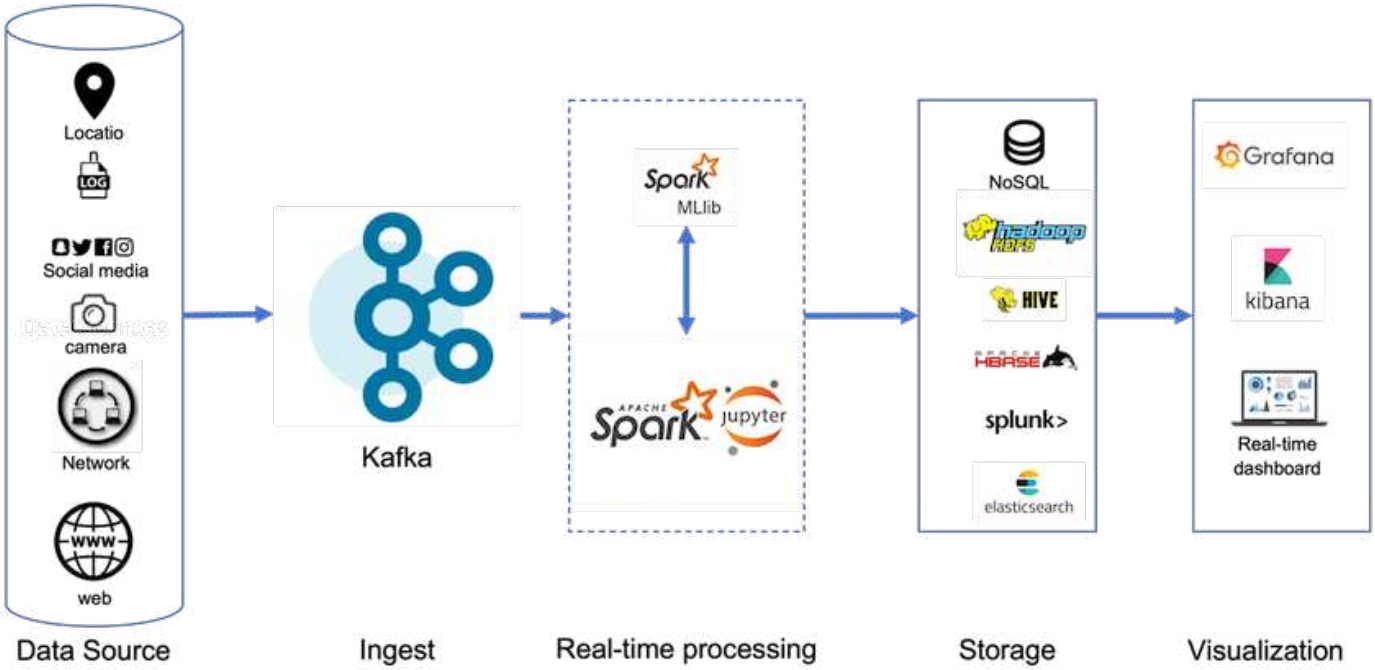
StorageGRID 의 가장 큰 차별화 요소는 정책 기반의 데이터 수명 주기 관리를 지원하는 정보 수명 주기 관리(ILM) 정책 엔진입니다. 정책 엔진은 메타데이터를 사용하여 데이터가 수명 동안 저장되는 방식을 관리하여 초기에는 성능을 최적화하고, 데이터가 오래됨에 따라 비용과 내구성을 자동으로 최적화할 수 있습니다.

Confluent 계층형 스토리지 활성화

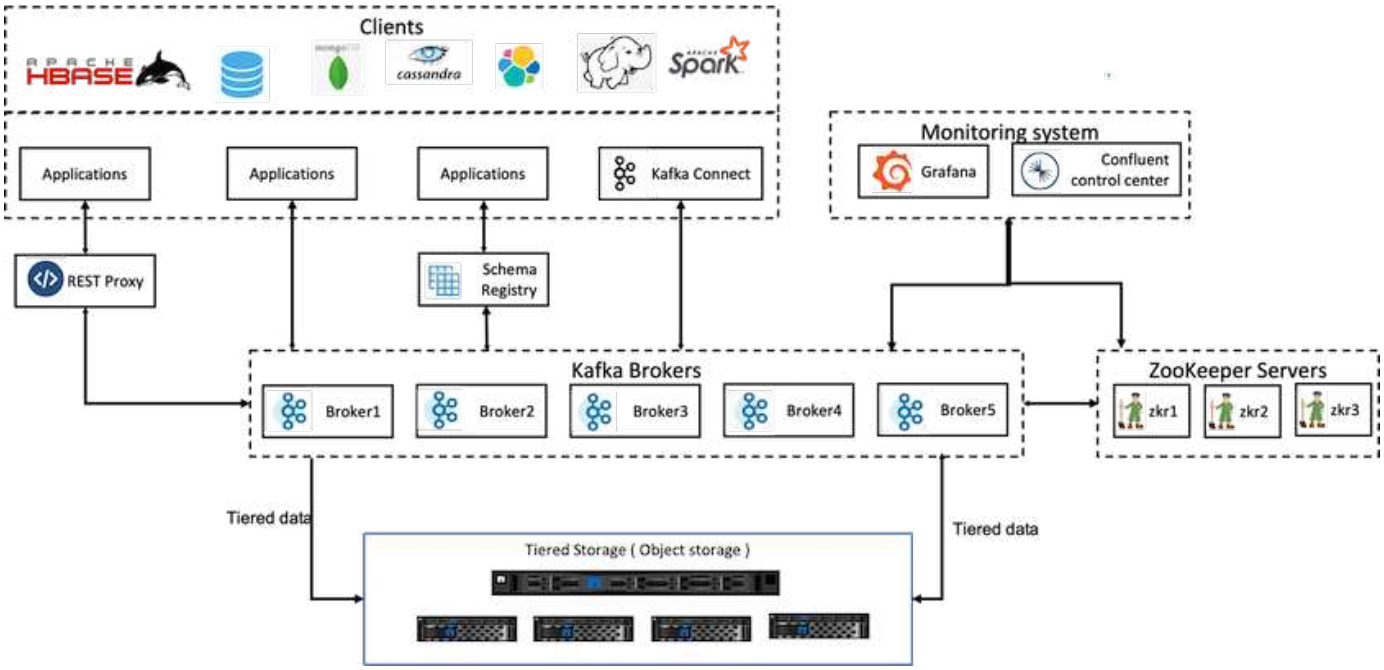
계층형 스토리지의 기본 아이디어는 데이터 저장 작업과 데이터 처리를 분리하는 것입니다. 이러한 분리를 통해 데이터 저장 계층과 데이터 처리 계층이 독립적으로 확장하기가 훨씬 쉬워집니다.

Confluent의 계층형 스토리지 솔루션은 두 가지 요소를 고려해야 합니다. 첫째, LIST 작업의 불일치나 가끔씩 발생하는 객체 사용 불가 등 공통적인 객체 저장소의 일관성 및 가용성 속성을 해결하거나 방지해야 합니다. 둘째, 계층형 스토리지와 카프카의 복제 및 장애 허용 모델 간의 상호 작용을 올바르게 처리해야 하며, 여기에는 좀비 리더가 계속해서 계층 오프셋 범위를 유지할 가능성이 포함됩니다. NetApp 개체 스토리지는 일관된 개체 가용성과 HA 모델을 모두 제공하여 피곤한 스토리지를 계층 오프셋 범위에 사용할 수 있도록 합니다. NetApp 개체 스토리지는 일관된 개체 가용성과 HA 모델을 제공하여 소모된 스토리지를 계층 오프셋 범위에 사용할 수 있도록 합니다.

계층형 스토리지를 사용하면 스트리밍 데이터의 끝부분에 가까운 저지연 읽기 및 쓰기에 고성능 플랫폼을 사용할 수 있으며, 높은 처리량의 과거 읽기에 NetApp StorageGRID 와 같은 저렴하고 확장 가능한 개체 저장소를 사용할 수도 있습니다. 또한 NetApp 스토리지 컨트롤러를 갖춘 Spark에 대한 기술 솔루션도 있으며 자세한 내용은 여기에서 확인하세요. 다음 그림은 Kafka가 실시간 분석 파이프라인에 어떻게 적용되는지 보여줍니다.



다음 그림은 NetApp StorageGRID Confluent Kafka의 객체 스토리지 계층으로 어떻게 적용되는지 보여줍니다.



솔루션 아키텍처 세부 정보

이 섹션에서는 Confluent 검증에 사용되는 하드웨어와 소프트웨어에 대해 설명합니다. 이 정보는 NetApp 스토리지를 사용한 Confluent Platform 배포에 적용됩니다. 다음 표는 테스트된 솔루션 아키텍처와 기본 구성 요소를 보여줍니다.

솔루션 구성 요소	세부
Confluent Kafka 버전 6.2	• 세 명의 동물원 관리인 • 5개의 브로커 서버 • 5개의 도구 서버 • 원 그라파나 • 하나의 제어 센터
리눅스(우분투 18.04)	모든 서버
계층형 스토리지를 위한 NetApp StorageGRID	• StorageGRID 소프트웨어 • 1 x SG1000(로드 밸런서) • 4 x SGF6024 • 4 x 24 x 800 SSD • S3 프로토콜 • 4 x 100GbE(브로커와 StorageGRID 인스턴스 간 네트워크 연결)
15개의 Fujitsu PRIMERGY RX2540 서버	각각 다음이 장착되어 있습니다: * 2개의 CPU, 총 16개의 물리적 코어 * Intel Xeon * 256GB 물리적 메모리 * 100GbE 듀얼 포트

기술 개요

이 섹션에서는 이 솔루션에 사용된 기술에 대해 설명합니다.

NetApp StorageGRID

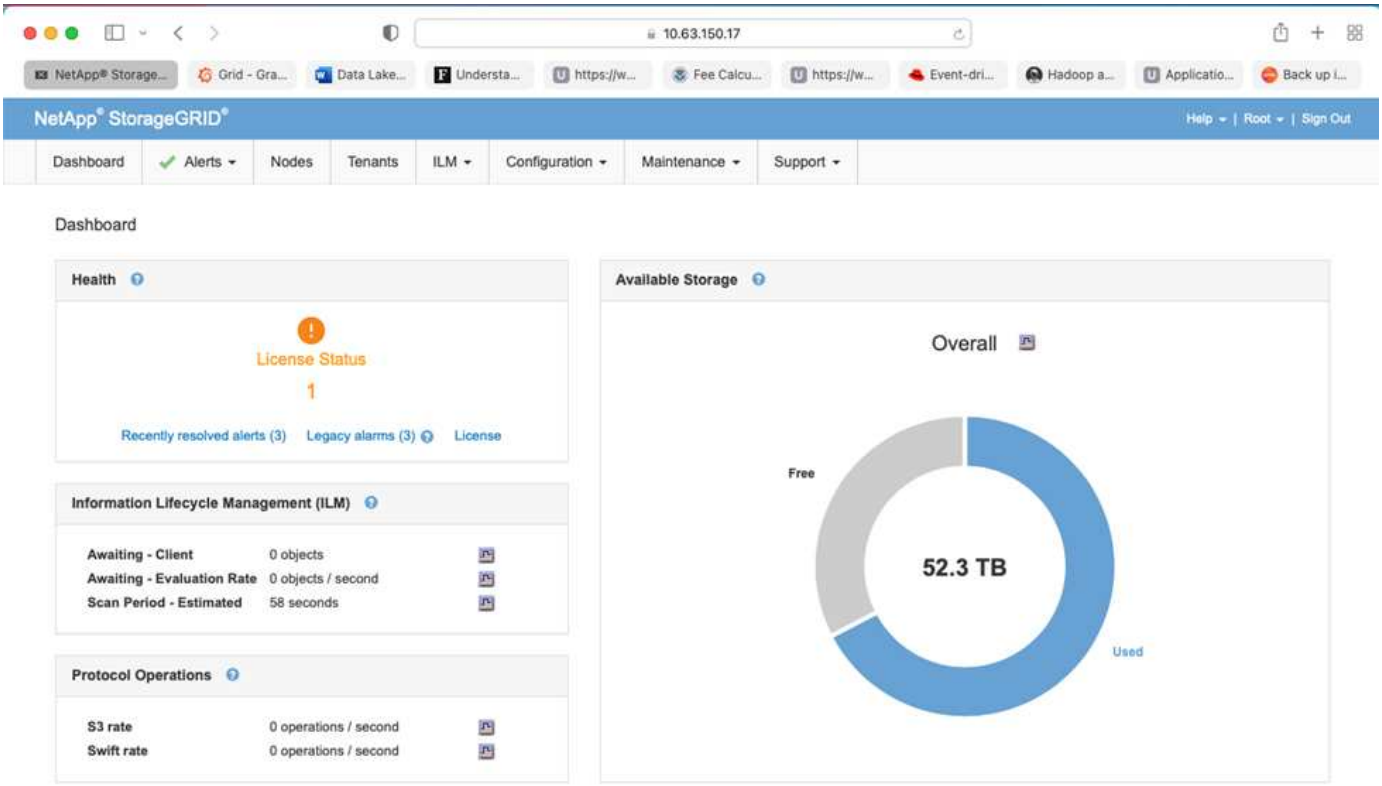
NetApp StorageGRID 는 고성능, 비용 효율적인 객체 스토리지 플랫폼입니다. 계층형 스토리지를 사용하면 브로커의 로컬 스토리지나 SAN 스토리지에 저장된 Confluent Kafka의 대부분 데이터가 원격 개체 저장소로 오프로드됩니다. 이 구성은 클러스터를 재조정, 확장 또는 축소하거나 실패한 브로커를 교체하는 데 드는 시간과 비용을 줄여 운영을 크게 개선합니다. 개체 스토리지는 개체 스토리지 계층에 있는 데이터를 관리하는 데 중요한 역할을 하므로 올바른 개체 스토리지를 선택하는 것이 중요합니다.

StorageGRID 분산된 노드 기반 그리드 아키텍처를 사용하여 지능적이고 정책 기반의 글로벌 데이터 관리를 제공합니다. 유비쿼터스 글로벌 객체 네임스페이스와 정교한 데이터 관리 기능을 결합하여 페타바이트 규모의 비정형 데이터와 수십억 개의 객체를 간편하게 관리할 수 있습니다. 단일 호출 객체 액세스는 여러 사이트로 확장되어 고가용성 아키텍처를 단순화하는 동시에 사이트나 인프라 중단에 관계없이 지속적인 객체 액세스를 보장합니다.

멀티테넌시를 통해 여러 개의 비정형 클라우드 및 엔터프라이즈 데이터 애플리케이션을 동일한 그리드 내에서 안전하게 서비스할 수 있어 NetApp StorageGRID 의 ROI와 사용 사례가 늘어납니다. 메타데이터 기반 개체 수명 주기 정책을 통해 여러 서비스 수준을 생성하여 여러 지역에 걸쳐 내구성, 보호, 성능 및 지역성을 최적화할 수 있습니다. 사용자는 끊임없이 변화하는 IT 환경에서 요구 사항이 바뀌더라도 데이터 관리 정책을 조정하고 트래픽 제한을 모니터링하여 데이터 환경에 맞게 중단 없이 재조정할 수 있습니다.

Grid Manager를 통한 간편한 관리

StorageGRID Grid Manager는 브라우저 기반 그래픽 인터페이스로, 단일 창에서 전 세계에 분산된 위치에 있는 StorageGRID 시스템을 구성, 관리 및 모니터링할 수 있습니다.



StorageGRID Grid Manager 인터페이스를 사용하여 다음 작업을 수행할 수 있습니다.

- 이미지, 비디오, 기록 등의 객체가 담긴 페타바이트 규모의 글로벌 분산 저장소를 관리합니다.
- 그리드 노드와 서비스를 모니터링하여 객체 가용성을 보장합니다.
- 정보 수명 주기 관리(ILM) 규칙을 사용하여 시간 경과에 따른 개체 데이터의 배치를 관리합니다. 이러한 규칙은 객체 데이터가 수집된 후 어떻게 처리되는지, 데이터가 손실로부터 어떻게 보호되는지, 객체 데이터가 어디에 저장되는지, 얼마 동안 저장되는지를 관리합니다.
- 시스템 내에서 거래, 성능 및 운영을 모니터링합니다.

정보 수명 주기 관리 정책

StorageGRID 개체의 복제본을 보관하고 2+1, 4+2 등과 같은 EC(삭제 코딩) 방식을 사용하여 특정 성능 및 데이터 보호 요구 사항에 따라 개체를 저장하는 등 유연한 데이터 관리 정책을 제공합니다. 시간이 지남에 따라 작업 부하와 요구 사항이 바뀌므로 ILM 정책도 시간이 지남에 따라 변경해야 하는 것이 일반적입니다. ILM 정책을 수정하는 것은 StorageGRID 고객이 끊임없이 변화하는 환경에 빠르고 쉽게 적응할 수 있도록 하는 핵심 기능입니다.

성능

StorageGRID VM, 베어 메탈 또는 특수 목적 어플라이언스와 같은 더 많은 스토리지 노드를 추가하여 성능을 확장합니다."SG5712, SG5760, SG6060 또는 SGF6024" . 테스트 결과, SGF6024 어플라이언스를 사용하여 최소 크기의 3노드 그리드로 Apache Kafka의 주요 성능 요구 사항을 충족했습니다. 고객이 추가 브로커를 사용하여 Kafka 클러스터를 확장하면 더 많은 스토리지 노드를 추가하여 성능과 용량을 늘릴 수 있습니다.

로드 밸런서 및 엔드포인트 구성

StorageGRID 의 관리 노드는 StorageGRID 시스템을 보고, 구성하고, 관리할 수 있는 Grid Manager UI(사용자 인터페이스)와 REST API 엔드포인트를 제공하며, 시스템 활동을 추적하기 위한 감사 로그를 제공합니다. Confluent Kafka 계층형 스토리지에 고가용성 S3 엔드포인트를 제공하기 위해 관리 노드와 게이트웨이 노드에서 서비스로 실행되는 StorageGRID 로드 밸런서를 구현했습니다. 또한, 로드 밸런서는 로컬 트래픽을 관리하고 GSLB(글로벌 서버 로드 밸런싱)와 통신하여 재해 복구를 지원합니다.

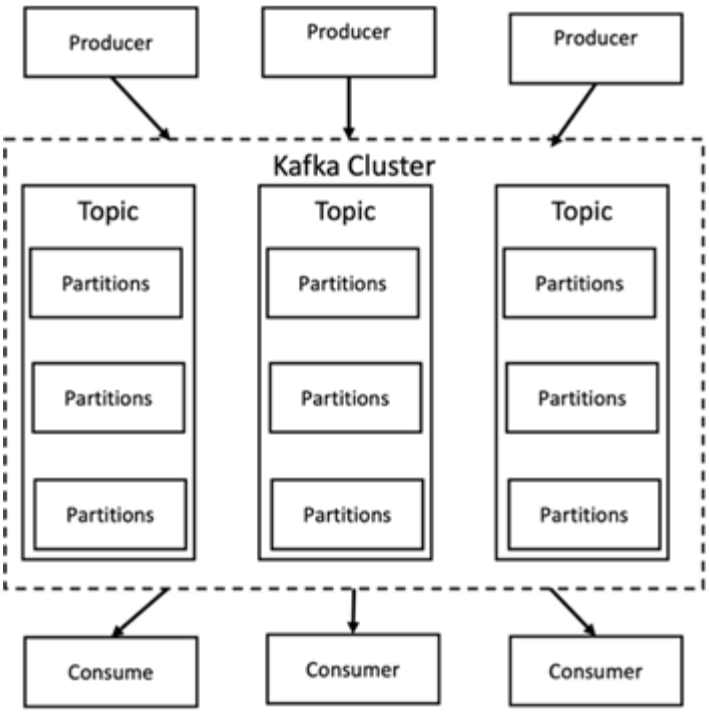
엔드포인트 구성을 더욱 향상시키기 위해 StorageGRID 관리 노드에 내장된 트래픽 분류 정책을 제공하고, 워크로드 트래픽을 모니터링하고, 워크로드에 다양한 서비스 품질(QoS) 제한을 적용할 수 있도록 합니다. 트래픽 분류 정책은 게이트웨이 노드와 관리 노드에 대한 StorageGRID 부하 분산 서비스의 엔드포인트에 적용됩니다. 이러한 정책은 트래픽 조절 및 모니터링에 도움이 될 수 있습니다.

StorageGRID 의 트래픽 분류

StorageGRID 에는 QoS 기능이 내장되어 있습니다. 트래픽 분류 정책은 클라이언트 애플리케이션에서 발생하는 다양한 유형의 S3 트래픽을 모니터링하는 데 도움이 될 수 있습니다. 그런 다음 입출력 대역폭, 동시 읽기/쓰기 요청 수 또는 읽기/쓰기 요청 속도를 기준으로 이 트래픽에 제한을 두는 정책을 만들고 적용할 수 있습니다.

아파치 카프카

Apache Kafka는 Java와 Scala로 작성된 스트림 처리를 사용하는 소프트웨어 버스의 프레임워크 구현입니다. 실시간 데이터 피드를 처리하기 위한 통합적이고 처리량이 높으며 지연 시간이 짧은 플랫폼을 제공하는 것이 목표입니다. Kafka는 Kafka Connect를 통해 외부 시스템에 연결하여 데이터를 내보내고 가져올 수 있으며, Java 스트림 처리 라이브러리인 Kafka 스트림을 제공합니다. 카프카는 효율성을 위해 최적화된 바이너리 TCP 기반 프로토콜을 사용하며, 네트워크 왕복 오버헤드를 줄이기 위해 자연스럽게 메시지를 그룹화하는 "메시지 세트" 추상화에 의존합니다. 이를 통해 더 큰 순차적 디스크 작업, 더 큰 네트워크 패킷, 연속적인 메모리 블록이 가능해져서 Kafka는 무작위 메시지 쓰기의 버스트 스트림을 선형 쓰기로 전환할 수 있습니다. 다음 그림은 Apache Kafka의 기본적인 데이터 흐름을 보여줍니다.



카프카는 프로듀서라고 불리는 임의의 수의 프로세스에서 나온 키-값 메시지를 저장합니다. 데이터는 다양한 주제 내에서 여러 파티션으로 분할될 수 있습니다. 파티션 내에서 메시지는 오프셋(파티션 내 메시지의 위치)에 따라 엄격하게 정렬되고, 타임스탬프와 함께 인덱싱되어 저장됩니다. 소비자라고 불리는 다른 프로세스는 파티션에서 메시지를 읽을 수 있습니다. 스트림 처리를 위해 Kafka는 Kafka에서 데이터를 사용하고 결과를 Kafka로 다시 쓰는 Java 애플리케이션을 작성할 수 있는 Streams API를 제공합니다. Apache Kafka는 Apache Apex, Apache Flink, Apache Spark, Apache Storm, Apache NiFi와 같은 외부 스트림 처리 시스템과도 호환됩니다.

카프카는 하나 이상의 서버(브로커라고 함)로 구성된 클러스터에서 실행되며, 모든 토픽의 파티션은 클러스터 노드에 분산됩니다. 또한 파티션은 여러 브로커에 복제됩니다. 이 아키텍처를 통해 Kafka는 장애 허용 방식으로 대량의 메시지 스트림을 전달할 수 있었고, Java Message Service(JMS), Advanced Message Queuing Protocol(AMQP) 등 기존 메시징 시스템 중 일부를 대체할 수 있었습니다. 0.11.0.0 릴리스 이후 Kafka는 Streams API를 사용하여 정확히 한 번의 스트림 처리를 제공하는 트랜잭션 쓰기 기능을 제공합니다.

카프카는 일반 토픽과 압축 토픽의 두 가지 유형을 지원합니다. 일반 주제는 보존 시간이나 공간 제한을 사용하여 구성할 수 있습니다. 지정된 보존 시간보다 오래된 레코드가 있거나 파티션의 공간 제한을 초과하는 경우 Kafka는 오래된 데이터를 삭제하여 저장 공간을 확보할 수 있습니다. 기본적으로 주제는 7일의 보존 기간으로 구성되지만, 무기한으로 데이터를 저장하는 것도 가능합니다. 압축된 주제의 경우 레코드는 시간이나 공간 경계에 따라 만료되지 않습니다. 대신, 카프카는 이후 메시지를 동일한 키를 가진 이전 메시지의 업데이트로 처리하고 키별로 최신 메시지를 삭제하지 않도록 보장합니다. 사용자는 특정 키에 null 값을 지정하여 소위 '툼스톤' 메시지를 작성하여 메시지를 완전히 삭제할 수 있습니다.

Kafka에는 5가지 주요 API가 있습니다.

- 생산자 **API.** 레코드 스트림을 게시할 수 있는 애플리케이션을 허용합니다.
- 소비자 **API.** 애플리케이션이 주제를 구독하고 레코드 스트림을 처리할 수 있도록 허용합니다.
- 커넥터 **API.** 주제를 기존 애플리케이션에 연결할 수 있는 재사용 가능한 생산자 및 소비자 API를 실행합니다.
- 스트림 **API.** 이 API는 입력 스트림을 출력으로 변환하고 결과를 생성합니다.
- 관리자 **API.** Kafka 토픽, 브로커 및 기타 Kafka 객체를 관리하는 데 사용됩니다.

소비자 및 생산자 API는 Kafka 메시징 프로토콜을 기반으로 구축되었으며 Java로 작성된 Kafka 소비자 및 생산자

클라이언트에 대한 참조 구현을 제공합니다. 기본 메시징 프로토콜은 개발자가 어떤 프로그래밍 언어로든 자체 소비자 또는 생산자 클라이언트를 작성하는 데 사용할 수 있는 바이너리 프로토콜입니다. 이렇게 하면 Kafka가 Java Virtual Machine(JVM) 생태계에서 해제됩니다. 사용 가능한 비Java 클라이언트 목록은 Apache Kafka 위키에서 관리됩니다.

Apache Kafka 사용 사례

Apache Kafka는 메시징, 웹사이트 활동 추적, 메트릭, 로그 집계, 스트림 처리, 이벤트 소싱 및 커밋 로깅에 가장 많이 사용됩니다.

- 카프카는 처리량이 개선되었고, 분할 기능, 복제 기능, 내결함성 등이 내장되어 있어 대규모 메시지 처리 애플리케이션에 적합한 솔루션입니다.
- 카프카는 사용자 활동(페이지 뷰, 검색)을 추적 파이프라인에서 실시간 게시-구독 피드 세트로 재구성할 수 있습니다.
- 카프카는 종종 운영 모니터링 데이터에 사용됩니다. 여기에는 분산된 애플리케이션의 통계를 집계하여 운영 데이터의 중앙 집중식 피드를 생성하는 작업이 포함됩니다.
- 많은 사람들이 로그 집계 솔루션의 대체 솔루션으로 Kafka를 사용합니다. 로그 집계는 일반적으로 서버에서 물리적인 로그 파일을 수집하여 처리를 위해 중앙 장소(예: 파일 서버나 HDFS)에 저장합니다. 카프카는 파일 세부 정보를 추상화하고 로그나 이벤트 데이터를 메시지 스트림으로 더욱 깔끔하게 추상화합니다. 이를 통해 처리 지연 시간을 줄이고 여러 데이터 소스와 분산된 데이터 소비에 대한 지원을 보다 쉽게 할 수 있습니다.
- 많은 카프카 사용자는 여러 단계로 구성된 처리 파이프라인에서 데이터를 처리합니다. 여기서 원시 입력 데이터는 카프카 토픽에서 소비된 후 집계, 강화되거나 추가 소비 또는 후속 처리를 위해 새로운 토픽으로 변환됩니다. 예를 들어, 뉴스 기사를 추천하는 처리 파이프라인은 RSS 피드에서 기사 콘텐츠를 크롤링하여 "기사" 주제에 게시할 수 있습니다. 추가 처리 단계에서는 이 콘텐츠를 정규화하거나 중복을 제거하고 정리된 기사 콘텐츠를 새로운 주제에 게시할 수 있으며, 최종 처리 단계에서는 이 콘텐츠를 사용자에게 추천하려고 시도할 수 있습니다. 이러한 처리 파이프라인은 개별 주제를 기반으로 실시간 데이터 흐름의 그래프를 생성합니다.
- 이벤트 소싱은 상태 변경 사항을 시간순으로 기록하는 애플리케이션 디자인 스타일입니다. Kafka는 매우 큰 규모의 저장된 로그 데이터를 지원하므로 이 스타일로 구축된 애플리케이션에 적합한 백엔드입니다.
- 카프카는 분산 시스템에 대한 일종의 외부 커밋 로그 역할을 할 수 있습니다. 로그는 노드 간에 데이터를 복제하는 데 도움이 되며, 실패한 노드가 데이터를 복원할 수 있는 재동기화 메커니즘 역할을 합니다. Kafka의 로그 압축 기능은 이러한 사용 사례를 지원하는 데 도움이 됩니다.

지류

Confluent Platform은 Kafka를 고급 기능으로 완성하여 애플리케이션 개발 및 연결을 가속화하고, 스트림 처리를 통해 변환을 지원하고, 대규모 기업 운영을 단순화하고, 엄격한 아키텍처 요구 사항을 충족하는 엔터프라이즈급 플랫폼입니다. Apache Kafka의 최초 개발자가 개발한 Confluent는 Kafka의 이점을 엔터프라이즈급 기능으로 확장하는 동시에 Kafka 관리나 모니터링의 부담을 제거합니다. 오늘날 Fortune 100 기업 중 80% 이상이 데이터 스트리밍 기술을 사용하고 있으며, 그 중 대부분은 Confluent를 사용합니다.

왜 **Confluent**를 선택해야 할까요**?**

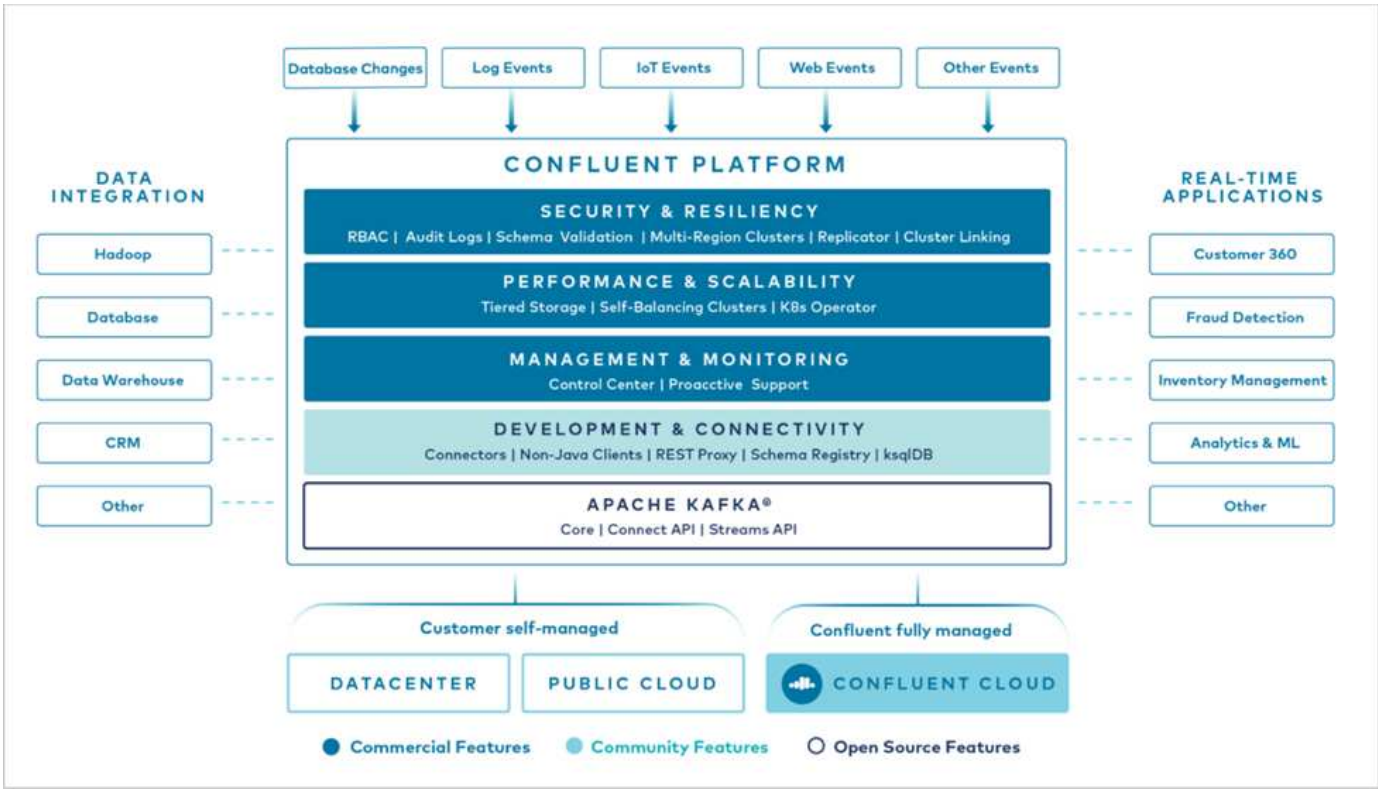
Confluent는 과거 데이터와 실시간 데이터를 단일의 중앙 진실 소스로 통합하여 완전히 새로운 종류의 최신 이벤트 중심 애플리케이션을 쉽게 구축하고, 보편적인 데이터 파이프라인을 확보하고, 완전한 확장성, 성능, 안정성을 갖춘 강력한 새로운 사용 사례를 창출할 수 있도록 지원합니다.

Confluent는 무엇에 사용되나요**?**

Confluent Platform을 사용하면 여러 시스템 간에 데이터가 전송되거나 통합되는 방식과 같은 기본적인 메커니즘에 대해 걱정하는 대신 데이터에서 비즈니스 가치를 도출하는 방법에 집중할 수 있습니다. 특히, Confluent Platform은 Kafka에 대한 데이터 소스 연결, 스트리밍 애플리케이션 구축, Kafka 인프라의 보안, 모니터링 및 관리를 간소화합니다.

오늘날 Confluent Platform은 금융 서비스, 옴니채널 소매, 자율주행차부터 사기 탐지, 마이크로서비스, IoT까지 다양한 산업에 걸쳐 광범위한 사용 사례에 사용되고 있습니다.

다음 그림은 Confluent Kafka 플랫폼 구성 요소를 보여줍니다.



Confluent의 이벤트 스트리밍 기술 개요

Confluent 플랫폼의 핵심은 다음과 같습니다. "아파치 카프카" 가장 인기 있는 오픈소스 분산 스트리밍 플랫폼입니다. 카프카의 주요 기능은 다음과 같습니다.

- 레코드 스트림을 게시하고 구독합니다.
- 장애에 견딜 수 있는 방식으로 레코드 스트림을 저장합니다.
- 레코드 스트림을 처리합니다.

Confluent Platform에는 기본적으로 스키마 레지스트리, REST 프록시, 100개 이상의 사전 구축된 Kafka 커넥터, ksqlDB가 포함되어 있습니다.

Confluent 플랫폼의 엔터프라이즈 기능 개요

- **Confluent** 제어 센터**.** Kafka를 관리하고 모니터링하기 위한 GUI 기반 시스템입니다. Kafka Connect를 쉽게 관리하고 다른 시스템에 대한 연결을 생성, 편집, 관리할 수 있습니다.
- **Kubernetes**용 **Confluent.** Kubernetes용 Confluent는 Kubernetes 운영자입니다. 쿠버네티스 운영자는 특정 플랫폼 애플리케이션에 대한 고유한 기능과 요구 사항을 제공하여 쿠버네티스의 오케스트레이션 기능을 확장합니다. Confluent Platform의 경우, 여기에는 Kubernetes에서 Kafka의 배포 프로세스를 크게 단순화하고 일반적인 인프라 수명 주기 작업을 자동화하는 것이 포함됩니다.
- **Kafka**에 대한 합류 커넥터**.** 커넥터는 Kafka Connect API를 사용하여 Kafka를 데이터베이스, 키-값 저장소, 검색 인덱스, 파일 시스템 등의 다른 시스템에 연결합니다. Confluent Hub에는 가장 인기 있는 데이터 소스와 싱크에

대한 다운로드 가능한 커넥터가 있으며, 여기에는 Confluent Platform에서 이러한 커넥터의 완전히 테스트되고 지원되는 버전이 포함됩니다. 더 자세한 내용은 다음을 참조하세요. "여기" .

- 자체 균형 클러스터. 자동화된 부하 분산, 장애 감지 및 자체 복구 기능을 제공합니다. 필요에 따라 브로커를 추가하거나 해제하는 데 대한 지원을 제공하며, 수동 조정은 필요하지 않습니다.
- 합류 클러스터 연결. 링크 브리지를 통해 클러스터를 직접 연결하고 한 클러스터의 주제를 다른 클러스터로 미러링합니다. 클러스터 연결을 통해 다중 데이터 센터, 다중 클러스터 및 하이브리드 클라우드 배포 설정이 간소화됩니다.
- **Confluent** 자동 데이터 밸런서. 클러스터 내의 브로커 수, 파티션 크기, 파티션 수, 리더 수를 모니터링합니다. 이 기능을 사용하면 클러스터 전체에서 균일한 작업 부하를 생성하기 위해 데이터를 이동할 수 있으며, 재조정하는 동안 프로덕션 작업 부하에 미치는 영향을 최소화하기 위해 재조정 트래픽을 조절할 수 있습니다.
- 합류 복제기. 여러 데이터 센터에서 여러 Kafka 클러스터를 유지 관리하는 것이 그 어느 때보다 쉬워졌습니다.
- 계층화된 스토리지. 선호하는 클라우드 공급업체를 사용하여 대용량의 Kafka 데이터를 저장할 수 있는 옵션을 제공하므로 운영 부담과 비용이 줄어듭니다. 계층형 스토리지를 사용하면 비용 효율적인 개체 스토리지에 데이터를 보관하고, 더 많은 컴퓨팅 리소스가 필요할 때만 확장 브로커를 사용할 수 있습니다.
- **Confluent JMS** 클라이언트. Confluent Platform에는 Kafka용 JMS 호환 클라이언트가 포함되어 있습니다. 이 Kafka 클라이언트는 Kafka 브로커를 백엔드로 사용하여 JMS 1.1 표준 API를 구현합니다. 이 기능은 JMS를 사용하는 레거시 애플리케이션이 있고 기존 JMS 메시지 브로커를 Kafka로 교체하려는 경우에 유용합니다.
- **Confluent MQTT** 프록시. 중간에 MQTT 브로커가 필요 없이 MQTT 장치 및 게이트웨이에서 Kafka로 직접 데이터를 게시하는 방법을 제공합니다.
- **Confluent** 보안 플러그인. Confluent 보안 플러그인은 다양한 Confluent Platform 도구와 제품에 보안 기능을 추가하는 데 사용됩니다. 현재 Confluent REST 프록시에 사용할 수 있는 플러그인이 있는데, 이 플러그인은 들어오는 요청을 인증하고 인증된 주체를 Kafka에 대한 요청에 전파하는 데 도움이 됩니다. 이를 통해 Confluent REST 프록시 클라이언트는 Kafka 브로커의 멀티테넌트 보안 기능을 활용할 수 있습니다.

Confluent 검증

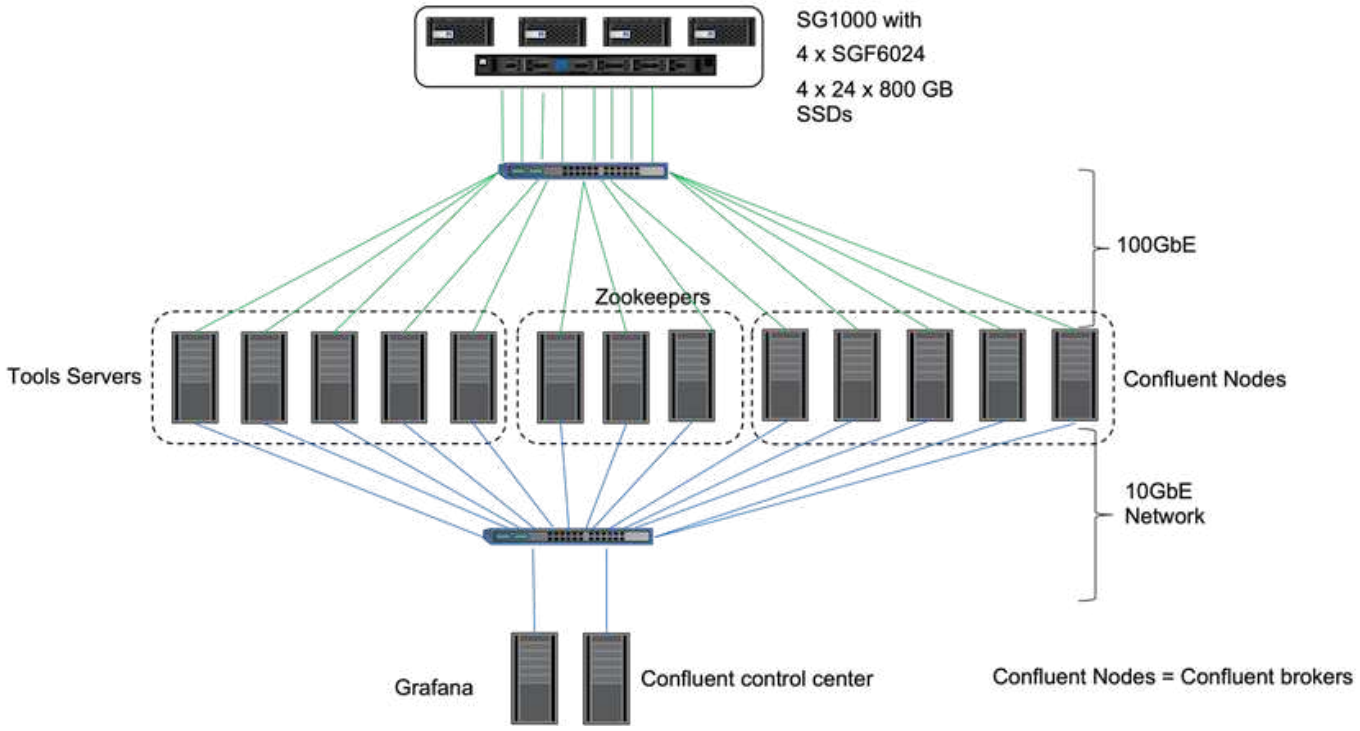
NetApp StorageGRID 의 Confluent Platform 6.2 Tiered Storage를 사용하여 검증을 수행했습니다. NetApp 과 Confluent 팀은 이 검증 작업을 함께 수행했으며 검증에 필요한 테스트 사례를 실행했습니다.

Confluent 플랫폼 설정

우리는 검증을 위해 다음과 같은 설정을 사용했습니다.

검증을 위해 3개의 동물원 관리자, 5개의 브로커, 5개의 테스트 스크립트 실행 서버, 256GB RAM과 16개의 CPU가 장착된 명명된 도구 서버를 사용했습니다. NetApp 스토리지의 경우, 4개의 SGF6024를 탑재한 SG1000 로드 밸런서와 함께 StorageGRID 사용했습니다. 저장소와 브로커는 100GbE 연결을 통해 연결되었습니다.

다음 그림은 Confluent 검증에 사용된 구성의 네트워크 토폴로지를 보여줍니다.



도구 서버는 Confluent 노드에 요청을 보내는 애플리케이션 클라이언트 역할을 합니다.

Confluent 계층형 스토리지 구성

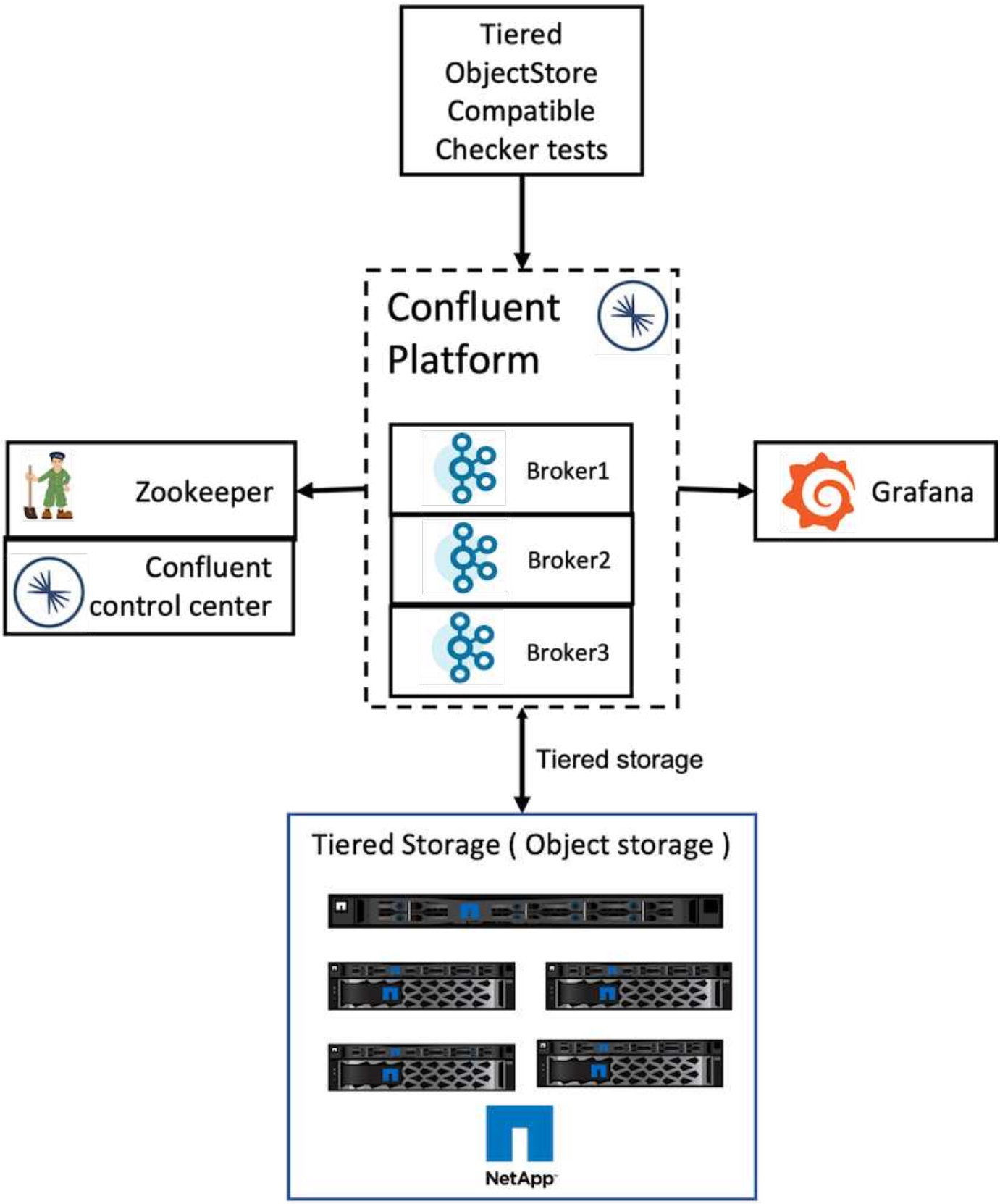
Kafka에서 계층형 스토리지 구성에는 다음 매개변수가 필요합니다.

```
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
```

검증을 위해 HTTP 프로토콜을 사용하는 StorageGRID 사용했지만 HTTPS도 가능합니다. 액세스 키와 비밀 키는 제공된 파일 이름에 저장됩니다. `confluent.tier.s3.cred.file.path` 매개변수.

NetApp 객체 스토리지 - StorageGRID

검증을 위해 StorageGRID 에서 단일 사이트 구성을 구성했습니다.



검증 테스트

우리는 검증을 위해 다음의 5가지 테스트 사례를 완료했습니다. 이러한 테스트는 Trogdor 프레임워크에서 실행됩니다. 처음 두 가지는 기능 테스트였고 나머지 세 가지는 성능 테스트였습니다.

객체 저장소 정확성 테스트

이 테스트는 계층형 스토리지의 요구 사항에 따라 객체 저장소 API의 모든 기본 작업(예: 가져오기/넣기/삭제)이 제대로 작동하는지 여부를 판별합니다. 이는 모든 객체 저장 서비스가 다음 테스트에 앞서 통과해야 하는 기본 테스트입니다. 합격 아니면 불합격이 결정되는 단정적 시험입니다.

계층화 기능 정확성 테스트

이 테스트는 엔드투엔드 계층형 스토리지 기능이 통과하거나 실패하는 단정적 테스트를 통해 잘 작동하는지 확인합니다. 이 테스트는 기본적으로 계층화가 활성화되고 핫셋 크기가 크게 줄어든 테스트 주제를 생성합니다. 새로 생성된 테스트 주제에 대한 이벤트 스트림을 생성하고, 브로커가 세그먼트를 객체 저장소에 보관할 때까지 기다린 다음, 이벤트 스트림을 사용하여 사용된 스트림이 생성된 스트림과 일치하는지 확인합니다. 이벤트 스트림에 생성되는 메시지의 수는 구성 가능하므로 사용자는 테스트 요구 사항에 따라 충분히 큰 작업 부하를 생성할 수 있습니다. 핫셋 크기를 줄이면 활성 세그먼트 외부의 소비자 페치가 객체 저장소에서만 제공되도록 보장됩니다. 이는 읽기에 대한 객체 저장소의 정확성을 테스트하는 데 도움이 됩니다. 우리는 객체 저장소 오류 주입을 적용한 경우와 적용하지 않은 경우로 이 테스트를 수행했습니다. StorageGRID 의 노드 중 하나에서 서비스 관리자 서비스를 중지하고 엔드투엔드 기능이 개체 스토리지에서 작동하는지 검증하여 노드 장애를 시뮬레이션했습니다.

티어 페치 벤치마크

이 테스트는 계층형 개체 스토리지의 읽기 성능을 검증하고 벤치마크에서 생성된 세그먼트에서 높은 부하가 걸리는 범위 페치 읽기 요청을 확인했습니다. 이 벤치마크에서 Confluent는 계층별 페치 요청을 처리하기 위해 사용자 정의 클라이언트를 개발했습니다.

생산-소비 워크로드 벤치마크

이 테스트는 세그먼트 보관을 통해 개체 저장소에 대한 쓰기 작업 부하를 간접적으로 생성했습니다. 소비자 그룹이 세그먼트를 가져올 때 개체 스토리지에서 읽기 작업 부하(세그먼트 읽기)가 생성되었습니다. 이 작업 부하는 테스트 스크립트에 의해 생성되었습니다. 이 테스트는 병렬 스레드에서 개체 스토리지의 읽기 및 쓰기 성능을 확인했습니다. 우리는 계층화 기능 정확성 테스트에서 했던 것처럼 객체 저장소 오류 주입을 적용한 경우와 적용하지 않은 경우를 테스트했습니다.

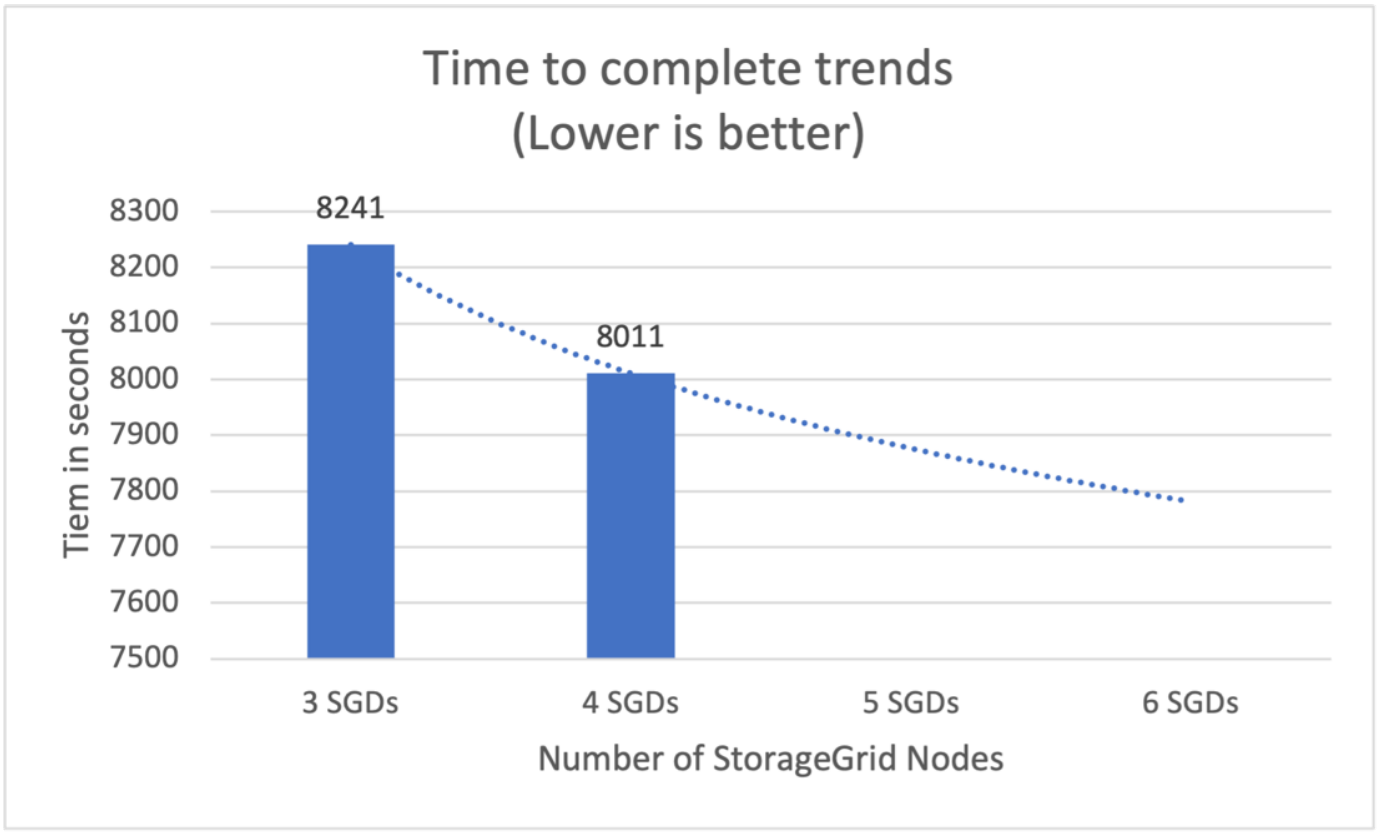
유지 관리 작업 벤치마크

이 테스트는 주제 보존 작업 부하가 큰 상황에서 객체 저장소의 삭제 성능을 확인했습니다. 보존 작업 부하는 테스트 주제와 병렬로 많은 메시지를 생성하는 테스트 스크립트를 사용하여 생성되었습니다. 테스트 주제는 이벤트 스트림이 개체 저장소에서 지속적으로 제거되도록 하는 공격적인 크기 기반 및 시간 기반 보존 설정으로 구성되었습니다. 그런 다음 세그먼트는 보관되었습니다. 이로 인해 브로커에 의한 개체 저장소의 대량 삭제가 발생하고 개체 저장소 삭제 작업의 성능이 저하되었습니다.

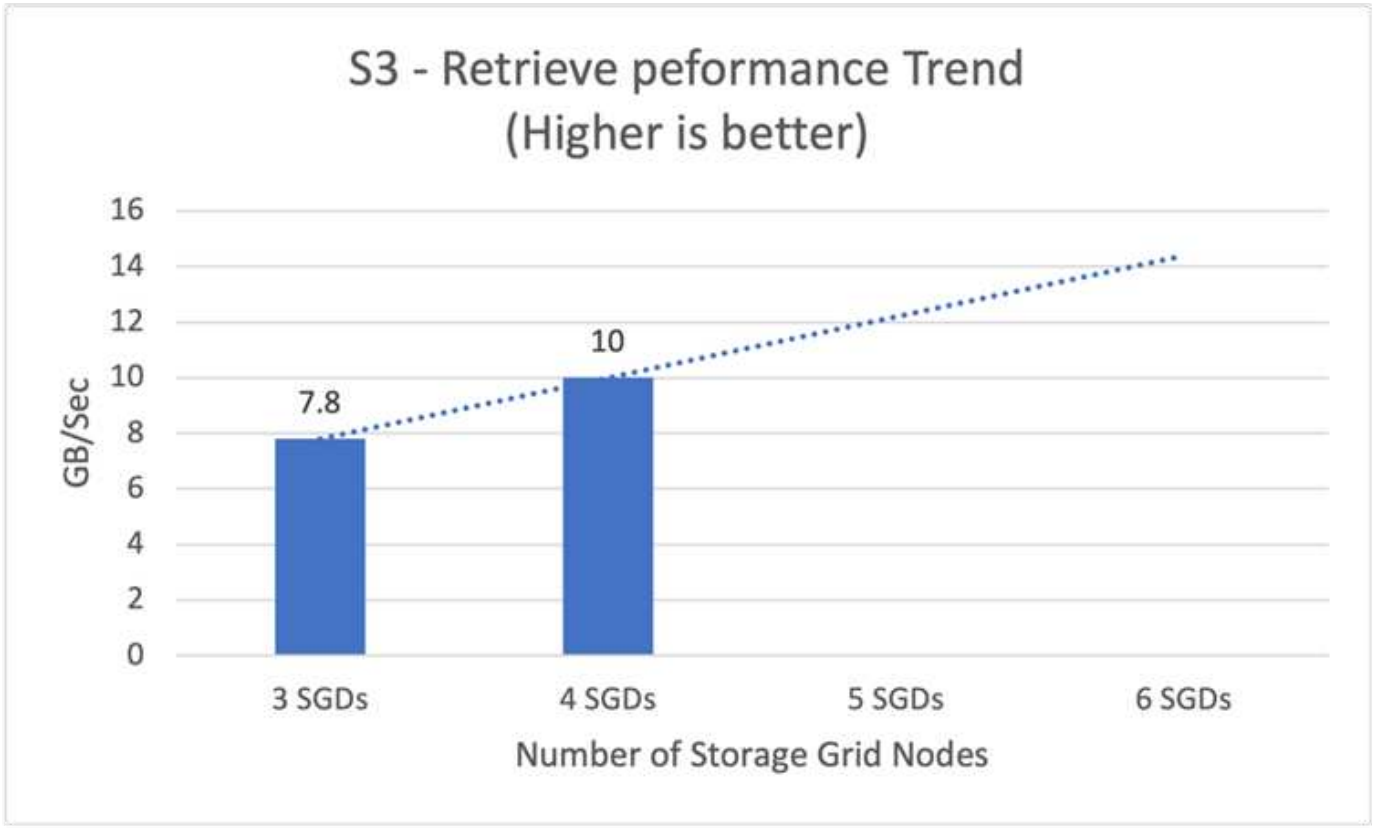
확장성을 갖춘 성능 테스트

NetApp StorageGRID 설정을 사용하여 프로듀서 및 소비자 워크로드에 대한 3~4개 노드로 계층형 스토리지 테스트를 수행했습니다. 테스트에 따르면 완료 시간과 성능 결과는 StorageGRID 노드 수에 직접적으로 비례했습니다. StorageGRID 설정에는 최소 3개의 노드가 필요했습니다.

- 저장 노드 수가 증가함에 따라 생산 및 소비자 작업을 완료하는 데 걸리는 시간이 선형적으로 감소했습니다.



- s3 검색 작업의 성능은 StorageGRID 노드 수에 따라 선형적으로 증가했습니다. StorageGRID 최대 200개의 StorgeGRID 노드를 지원합니다.



Confluent S3 커넥터

Amazon S3 Sink 커넥터는 Apache Kafka 토픽의 데이터를 Avro, JSON 또는 Bytes 형식으로 S3 객체로 내보냅니다. Amazon S3 싱크 커넥터는 주기적으로 Kafka에서 데이터를 폴링하고 이를 다시 S3에 업로드합니다. 파티셔너는 모든 카프카 파티션의 데이터를 청크로 분할하는 데 사용됩니다. 각 데이터 덩어리는 S3 객체로 표현됩니다. 키 이름은 주제, 카프카 파티션, 이 데이터 청크의 시작 오프셋을 인코딩합니다.

이 설정에서는 Kafka S3 싱크 커넥터를 사용하여 Kafka에서 개체 스토리지의 주제를 직접 읽고 쓰는 방법을 보여드립니다. 이 테스트에서는 독립형 Confluent 클러스터를 사용했지만, 이 설정은 분산 클러스터에도 적용할 수 있습니다.

1. Confluent 웹사이트에서 Confluent Kafka를 다운로드하세요.
2. 서버의 폴더에 패키지를 압축 해제합니다.
3. 두 개의 변수를 내보냅니다.

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

4. 독립 실행형 Confluent Kafka 설정의 경우 클러스터는 임시 루트 폴더를 생성합니다. `/tmp` 또한 Zookeeper, Kafka, 스키마 레지스트리, Connect, ksql-server 및 Control-Center 폴더를 생성하고 해당 구성 파일을 다음에서 복사합니다. `$CONFLUENT_HOME` . 다음 예를 참조하세요.

```
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. Zookeeper를 구성합니다. 기본 매개변수를 사용하는 경우 아무것도 변경할 필요가 없습니다.

```
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:~#
```

위의 구성에서 우리는 다음을 업데이트했습니다. `server. xxx` 재산. 기본적으로 Kafka 리더 선택을 위해서는 3명의 Zookeeper가 필요합니다.

6. 우리는 myid 파일을 생성했습니다 `/tmp/confluent.406980/zookeeper/data` 고유 ID로:

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

우리는 myid 파일에 마지막 IP 주소 번호를 사용했습니다. Kafka, connect, control-center, Kafka, Kafka-rest, ksql-server 및 schema-registry 구성에 기본값을 사용했습니다.

7. 카프카 서비스를 시작합니다.

```
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#
```

각 구성에는 문제 해결에 도움이 되는 로그 폴더가 있습니다. 어떤 경우에는 서비스를 시작하는 데 시간이 더 오래 걸립니다. 모든 서비스가 정상적으로 작동하고 있는지 확인하세요.

8. Kafka Connect를 사용하여 설치하세요 `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#
```

다음을 사용하여 특정 버전을 설치할 수도 있습니다. `confluent-hub install confluentinc/kafka-connect-s3:10.0.3` .

9. 기본적으로, `confluentinc-kafka-connect-s3` 에 설치됨 `/data/confluent/confluent-6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3` .
10. 플러그인 경로를 새 것으로 업데이트합니다. `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. Confluent 서비스를 중지했다가 다시 시작합니다.

```
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. 액세스 ID 및 비밀 키를 구성하세요. `/root/.aws/credentials` 파일.

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

13. 버킷에 접근 가능한지 확인하세요.

```
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. S3 및 버킷 구성을 위해 S3-싱크 속성 파일을 구성합니다.

```
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. 몇 개의 레코드를 S3 버킷으로 가져옵니다.

```
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. s3-싱크 커넥터를 로드합니다.

```
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. s3-sink 상태를 확인하세요.

```
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. s3-sink가 토픽을 수락할 준비가 되었는지 확인하려면 로그를 확인하세요.

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

19. 카프카의 주제를 확인하세요.

```
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. s3 버킷의 객체를 확인하세요.

```
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. 내용을 확인하려면 다음 명령을 실행하여 각 파일을 S3에서 로컬 파일 시스템으로 복사합니다.

```
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. 레코드를 인쇄하려면 avro-tools-1.11.0.1.jar(다음에서 사용 가능)를 사용하세요. "아파치 아카이브").

```
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 커넥터

Instaclustr는 Kafka Connect 커넥터 및 관련 세부 정보를 지원합니다. "자세한 내용". Instaclustr는 추가 커넥터를 제공합니다. "그들의 세부 정보"

Confluent 자체 균형 클러스터

이전에 카프카 클러스터를 관리한 적이 있다면 클러스터 전체에서 작업 부하가 균형을 이루도록 여러 브로커에 파티션을 수동으로 재할당하는 데 따르는 과제에 대해 잘 알고 있을 것입니다. 대규모 Kafka 배포를 운영하는 조직의 경우, 대량의 데이터를 재조정하는 일은 어렵고 지루하며 위험할 수 있습니다. 특히, 미션 크리티컬 애플리케이션이 클러스터 위에 구축된 경우 더욱 그렇습니다. 그러나 가장 작은 카프카 사용 사례라 하더라도 이 과정은 시간이 많이 걸리고 인간의 실수가 발생하기 쉽습니다.

저희 연구실에서는 클러스터 토폴로지 변경이나 부하 불균형에 따라 자동으로 재조정하는 Confluent 셀프 밸런싱 클러스터 기능을 테스트했습니다. Confluent 재균형 테스트는 노드 장애가 발생하거나 확장 노드에서 브로커 간에 데이터를 재균형화해야 할 때 새로운 브로커를 추가하는 데 걸리는 시간을 측정하는 데 도움이 됩니다. 클래식 카프카 구성에서는 클러스터가 커짐에 따라 재조정해야 할 데이터 양이 늘어나지만, 계층형 스토리지에서는 재조정이 적은 양의 데이터로 제한됩니다. 검증 결과에 따르면, 클래식 카프카 아키텍처에서 계층형 스토리지의 재조정은 몇 초 또는 몇 분이 걸리고 클러스터가 커짐에 따라 선형적으로 증가합니다.

셀프 밸런싱 클러스터에서는 파티션 재조정이 완전히 자동화되어 Kafka의 처리량을 최적화하고, 브로커 확장을 가속화하고, 대규모 클러스터를 실행하는 데 따른 운영 부담을 줄입니다. 정상 상태에서 자체 균형 클러스터는 브로커 전체의 데이터 불균형을 모니터링하고 클러스터 성능을 최적화하기 위해 지속적으로 파티션을 재할당합니다. 플랫폼을 확장하거나 축소할 때 셀프 밸런싱 클러스터는 자동으로 새로운 브로커의 존재 또는 기존 브로커의 제거를 인식하고 후속 파티션 재할당을 트리거합니다. 이를 통해 브로커를 쉽게 추가하고 해제할 수 있어 Kafka 클러스터가 근본적으로 더 탄력적으로 작동합니다. 이러한 이점은 파티션 재할당에 일반적으로 수반되는 수동 개입, 복잡한 수학 또는 인간 오류의 위험 없이 제공됩니다. 결과적으로 데이터 재조정이 훨씬 짧은 시간 안에 완료되고, 클러스터를 지속적으로 감독할 필요 없이 더 높은 가치의 이벤트 스트리밍 프로젝트에 집중할 수 있습니다.

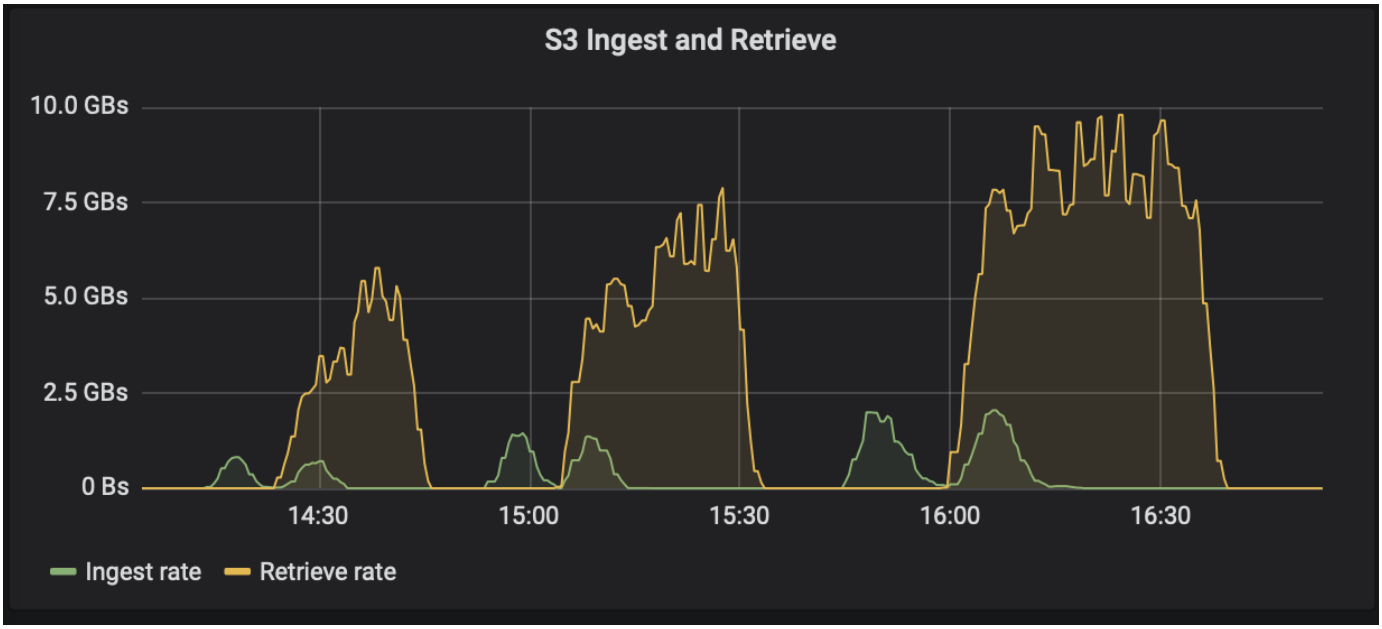
Instaclustr는 자체 재조정 기능도 지원하며 이미 여러 고객사에 도입되었습니다.

모범 사례 지침

이 섹션에서는 이 인증으로부터 얻은 교훈을 소개합니다.

- 검증 결과, Confluent가 데이터를 보관하는 데는 S3 개체 스토리지가 가장 적합합니다.
- Confluent 계층형 스토리지 구성에서 브로커 데이터 디렉토리에 보관되는 데이터 크기는 세그먼트 크기와 데이터가 개체 스토리지로 이동될 때의 보존 시간에 따라 결정되므로, 고처리량 SAN(특히 FC)을 사용하여 브로커의 핫 데이터나 로컬 디스크를 유지할 수 있습니다.
- segment.bytes가 높을수록 개체 저장소의 성능이 더 좋습니다. 테스트에서는 512MB를 사용했습니다.
- Kafka에서는 주제에 생성된 각 레코드의 키 또는 값의 길이(바이트)가 다음에 의해 제어됩니다. `length.key.value` 매개변수. StorageGRID 의 경우 S3 객체 수집 및 검색 성능이 더 높은 값으로 향상되었습니다. 예를 들어, 512바이트는 5.8GBps 검색 속도를 제공하고, 1024바이트는 7.5GBps s3 검색 속도를 제공하며, 2048바이트는 거의 10GBps 검색 속도를 제공합니다.

다음 그림은 S3 객체 수집 및 검색을 기반으로 보여줍니다. `length.key.value` .



- 카프카 튜닝. 계층형 저장소의 성능을 개선하려면 TierFetcherNumThreads와 TierArchiverNumThreads를 늘릴 수 있습니다. 일반적인 지침으로, TierFetcherNumThreads를 실제 CPU 코어 수와 일치하도록 늘리고 TierArchiverNumThreads를 CPU 코어 수의 절반으로 늘리는 것이 좋습니다. 예를 들어, 서버 속성에서 물리적 코어가 8개인 머신이 있는 경우 confluent.tier.fetcher.num.threads = 8, confluent.tier.archiver.num.threads = 4로 설정합니다.
- 주제 삭제 시간 간격. 주제가 삭제되면 개체 스토리지에 있는 로그 세그먼트 파일의 삭제가 즉시 시작되지 않습니다. 대신, 해당 파일이 삭제되기 전에 기본값인 3시간의 시간 간격이 있습니다. confluent.tier.topic.delete.check.interval.ms 구성을 수정하여 이 간격의 값을 변경할 수 있습니다. 주제나 클러스터를 삭제하는 경우 해당 버킷에 있는 객체를 수동으로 삭제할 수도 있습니다.
- 계층화된 스토리지 내부 주제에 대한 **ACL.** 온프레미스 배포에 권장되는 모범 사례는 계층형 저장소에 사용되는 내부 항목에 ACL 권한 부여자를 활성화하는 것입니다. ACL 규칙을 설정하여 이 데이터에 대한 액세스를 브로커 사용자로만 제한합니다. 이를 통해 내부 주제가 보호되고 계층형 스토리지 데이터와 메타데이터에 대한 무단 액세스가 방지됩니다.

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

사용자를 교체하세요 `<kafka>` 배포 시 실제 브로커 주체와 함께.

예를 들어, 명령 `confluent-tier-state` 계층형 저장소에 대한 내부 주제에 ACL을 설정합니다. 현재 계층형 스토리지와 관련된 내부 주제는 단 하나뿐입니다. 이 예제에서는 내부 주제에 대한 모든 작업에 대한 주요 Kafka 권한을 제공하는 ACL을 만듭니다.

사이징

카프카 크기 조정은 간단, 세분화, 역방향, 파티션의 4가지 구성 모드로 수행할 수 있습니다.

단순한

간단 모드는 Apache Kafka를 처음 사용하는 사용자나 초기 상태 사용 사례에 적합합니다. 이 모드에서는 처리량(MBps), 읽기 팬아웃, 보존 및 리소스 활용률(기본값은 60%)과 같은 요구 사항을 제공합니다. 또한 온프레미스(베어메탈, VMware, Kubernetes 또는 OpenStack)나 클라우드와 같은 환경으로 들어갑니다. 이 정보를 기반으로 Kafka 클러스터의 크기를 조정하면 브로커, Zookeeper, Apache Kafka Connect Worker, 스키마 레지스트리, REST 프록시, ksqlDB 및 Confluent 제어 센터에 필요한 서버 수가 제공됩니다.

계층형 스토리지의 경우 Kafka 클러스터 크기를 조정하기 위한 세분화된 구성 모드를 고려하세요. 세분화 모드는 숙련된 Apache Kafka 사용자나 명확하게 정의된 사용 사례에 적합합니다. 이 섹션에서는 생산자, 스트림 프로세서, 소비자의 크기 조정에 대해 설명합니다.

프로듀서

Apache Kafka의 제작자(예: 네이티브 클라이언트, REST 프록시 또는 Kafka 커넥터)를 설명하려면 다음 정보를 제공하세요.

- 이름. 불꽃.
- 생산자 유형. 애플리케이션 또는 서비스, 프록시(REST, MQTT, 기타), 기존 데이터베이스(RDBMS, NOSQL, 기타). "모르겠습니다"를 선택할 수도 있습니다.
- 평균 처리량. 초당 이벤트 수(예: 1,000,000개).
- 최대 처리량. 초당 이벤트 수(예: 4,000,000개).
- 평균 메시지 크기. 압축되지 않은 바이트 단위(최대 1MB, 예: 1000).
- 메시지 형식. 옵션으로는 Avro, JSON, 프로토콜 버퍼, 바이너리, 텍스트, "모르겠습니다" 및 기타가 있습니다.
- 복제 인자. 옵션은 1, 2, 3(Confluent 추천), 4, 5, 6입니다.
- 보존 시간. 어느 날(예를 들어). Apache Kafka에 데이터를 얼마 동안 저장하고 싶으신가요? 무한한 시간을 원하면 -1을 아무 단위나 입력하세요. 계산기는 무한 보존을 위해 보존 기간이 10년이라고 가정합니다.
- "계층형 스토리지를 사용하여 브로커 수를 줄이고 무한한 스토리지를 허용하시겠습니까?" 확인란을 선택하세요.
- 계층형 스토리지가 활성화되면 보존 필드는 브로커에 로컬로 저장된 핫 데이터 세트를 제어합니다. 보관 보존 필드는 데이터가 보관 개체 저장소에 저장되는 기간을 제어합니다.
- 보관 보관 (예를 들어) 1년. 귀하의 데이터를 보관 저장소에 얼마 동안 보관하고 싶으신가요? 무한한 시간을 원하면 -1을 아무 단위나 입력하세요. 계산기는 무한 보존을 위해 10년간 보존한다고 가정합니다.
- 성장 배수. 1 (예를 들어). 이 매개변수의 값이 현재 처리량을 기반으로 하는 경우 1로 설정합니다. 추가적인 성장에 따라 크기를 조정하려면 이 매개변수를 성장 배수로 설정하세요.
- 생산자 인스턴스 수. 10 (예를 들어). 몇 개의 프로듀서 인스턴스가 실행될 예정인가요? 이 입력은 CPU 부하를 크기 계산에 통합하는 데 필요합니다. 빈 값은 CPU 부하가 계산에 포함되지 않음을 나타냅니다.

이 예시 입력을 기준으로 하면 크기 조정은 생산자에게 다음과 같은 영향을 미칩니다.

- 압축되지 않은 바이트의 평균 처리량: 1GBps. 압축되지 않은 바이트의 최대 처리량: 4GBps. 압축된 바이트의 평균 처리량: 400MBps. 압축 바이트의 최대 처리량: 1.6GBps. 이는 기본 60% 압축률을 기준으로 합니다(이 값은 변경할 수 있습니다).
  - 브로커 내 핫셋 스토리지에 필요한 총 용량: 복제를 포함하여 압축된 용량으로 31,104TB입니다. 필요한 총 오프브로커 보관 저장 용량: 압축 시 378,432TB. 사용"https://fusion.netapp.com" StorageGRID 크기 조정을 위해.

스트림 프로세서는 Apache Kafka에서 데이터를 소비하고 Apache Kafka로 다시 생성하는 애플리케이션이나 서비스를 설명해야 합니다. 대부분의 경우 이러한 기능은 ksqlDB 또는 Kafka Streams에 내장되어 있습니다.

- 이름. 스파크 스트리머.
- 처리 시간. 이 프로세서가 단일 메시지를 처리하는 데 얼마나 걸리나요?
  - 1ms(단순, 상태 비저장 변환) [예], 10ms(상태 저장 메모리 내 작업).
  - 100ms(상태 저장 네트워크 또는 디스크 작업), 1000ms(제3자 REST 호출).
  - 저는 이 매개변수를 벤치마킹했고, 얼마나 걸리는지 정확히 알고 있습니다.
- 산출물 보존. 1일(예시). 스트림 프로세서는 Apache Kafka로 출력을 생성합니다. 이 출력 데이터를 Apache Kafka에 얼마 동안 저장하고 싶으신가요? 무한한 시간을 원하면 -1을 아무 단위나 입력하세요.
- "계층형 스토리지를 사용하여 브로커 수를 줄이고 무한한 스토리지를 허용하시겠습니까?" 확인란을 선택하세요.
- 보관 보관 1년(예를 들어). 귀하의 데이터를 보관 저장소에 얼마 동안 보관하고 싶으신가요? 무한한 시간을 원하면 -1을 아무 단위나 입력하세요. 계산기는 무한 보존을 위해 10년간 보존한다고 가정합니다.
- 출력 패스스루 백분율. 100(예를 들어). 스트림 프로세서는 Apache Kafka로 출력을 생성합니다. 인바운드 처리량 중 Apache Kafka로 다시 출력되는 비율은 얼마입니까? 예를 들어, 인바운드 처리량이 20MBps이고 이 값이 10이면 출력 처리량은 2MBps가 됩니다.
- 이것은 어떤 응용프로그램에서 읽어오는 것인가요? 생산자 유형 기반 크기 조정에 사용되는 이름인 "Spark"를 선택합니다. 위의 입력을 기반으로 스트림 프로세서 인스턴스와 토픽 파티션 추정에 대한 크기 조정의 다음과 같은 효과를 기대할 수 있습니다.
- 이 스트림 프로세서 애플리케이션에는 다음과 같은 개수의 인스턴스가 필요합니다. 들어오는 주제에도 이 정도의 파티션이 필요할 가능성이 있습니다. 이 매개변수를 확인하려면 Confluent에 문의하세요.
  - 성장 배수 없이 평균 처리량의 경우 1,000
  - 성장 배수 없이 최대 처리량의 경우 4,000
  - 성장 배수를 적용한 평균 처리량의 경우 1,000
  - 성장 배수를 적용한 최대 처리량의 경우 4,000

소비자들

Apache Kafka에서 데이터를 사용하지만 Apache Kafka로 다시 데이터를 생성하지 않는 애플리케이션이나 서비스를 설명해 주세요. 예를 들어, 네이티브 클라이언트나 Kafka 커넥터가 있습니다.

- 이름. 소비자를 자극합니다.
- 처리 시간. 이 소비자가 단일 메시지를 처리하는 데 얼마나 걸리나요?
  - 1ms(예: 로깅과 같은 간단하고 상태 없는 작업)
  - 10ms(데이터 저장소에 대한 빠른 쓰기)
  - 100ms(데이터 저장소에 대한 느린 쓰기)
  - 1000ms(제3자 REST 호출)
  - 기간이 알려진 다른 벤치마크 프로세스.
- 소비자 유형. 기존 데이터 저장소(RDBMS, NoSQL, 기타)에 대한 애플리케이션, 프록시 또는 싱크입니다.
- 이것은 어떤 응용프로그램에서 읽어오는 것인가요? 이 매개변수를 이전에 결정된 프로듀서 및 스트림 크기에 연결합니다.

위의 입력 내용을 기반으로 소비자 인스턴스 크기와 주제 파티션 추정치를 결정해야 합니다. 소비자 애플리케이션에는 다음과 같은 수의 인스턴스가 필요합니다.

- 평균 처리량의 경우 2,000, 성장 배수 없음
- 최대 처리량의 경우 8,000, 성장 배수 없음
- 성장 배수를 포함한 평균 처리량의 경우 2,000
- 최대 처리량의 경우 8,000(성장 배수 포함)

들어오는 주제에도 이 개수의 파티션이 필요할 가능성이 높습니다. 확인하려면 Confluent에 문의하세요.

생산자, 스트림 프로세서, 소비자에 대한 요구 사항 외에 다음과 같은 추가 요구 사항을 제공해야 합니다.

- 재건 시간입니다**.** 예를 들어, 4시간. Apache Kafka 브로커 호스트에 장애가 발생하여 데이터가 손실되고, 장애가 발생한 호스트를 대체하기 위해 새로운 호스트가 프로비저닝되는 경우, 이 새로운 호스트는 얼마나 빨리 자체적으로 재구축해야 합니까? 값을 알 수 없는 경우 이 매개변수를 비워 두세요.
- 자원 활용 목표**(**백분율**).** 예를 들어, 60. 평균 처리량 동안 호스트를 얼마나 활용하고 싶으신가요? Confluent는 Confluent 자체 균형 클러스터를 사용하지 않는 한 60%의 사용률을 권장하지만, Confluent 자체 균형 클러스터를 사용하는 경우 사용률이 더 높아질 수 있습니다.

주변 환경을 설명하세요

- 클러스터는 어떤 환경에서 실행되나요**?** Amazon Web Services, Microsoft Azure, Google Cloud Platform, 온프레미스 베어메탈, 온프레미스 VMware, 온프레미스 OpenStack, 온프레미스 Kubernates 중 어떤 것을 선택하시겠습니까?
- 호스트 정보**.** 코어 수: 48개(예시), 네트워크 카드 유형(10GbE, 40GbE, 16GbE, 1GbE 또는 다른 유형).
- 저장 용량**.** 호스트: 12(예시) 호스트당 몇 개의 하드 드라이브 또는 SSD가 지원됩니까? Confluent는 호스트당 12개의 하드 드라이브를 권장합니다.
- 저장 용량**/**볼륨**(GB)** 1000(예를 들어). 단일 볼륨은 기가바이트 단위로 얼마나 많은 저장 공간을 저장할 수 있나요? Confluent에서는 1TB 디스크를 권장합니다.
- 저장 구성**.** 저장 볼륨은 어떻게 구성되나요? Confluent에서는 모든 Confluent 기능을 활용하기 위해 RAID10을 권장합니다. JBOD, SAN, RAID 1, RAID 0, RAID 5 및 기타 유형도 지원됩니다.
- 단일 볼륨 처리량**(MBps).** 125 (예를 들어). 단일 저장 볼륨은 초당 메가바이트 단위로 얼마나 빨리 읽거나 쓸 수 있습니까? Confluent는 일반적으로 처리량이 125MBps인 표준 하드 드라이브를 권장합니다.
- 메모리 용량**(GB).** 64(예를 들어).

환경 변수를 결정한 후 클러스터 크기를 선택합니다. 위에 표시된 예시 매개변수를 기반으로 Confluent Kafka에 대한 다음 크기를 결정했습니다.

- 아파치 카프카**.** 브로커 수: 22개. 귀하의 클러스터는 저장소에 제한되어 있습니다. 호스트 수를 줄이고 무한한 저장 공간을 확보하려면 계층형 저장소를 활성화하는 것을 고려하세요.
- 아파치 동물원 관리자**.** 개수: 5; Apache Kafka Connect Workers: 개수: 2; 스키마 레지스트리: 개수: 2; REST 프록시: 개수: 2; ksqlDB: 개수: 2; Confluent Control Center: 개수: 1.

사용 사례가 고려되지 않은 플랫폼 팀의 경우 역방향 모드를 사용합니다. 파티션 모드를 사용하면 단일 주제에 필요한 파티션 수를 계산할 수 있습니다. 보다 https://eventsizer.io 역방향 및 파티션 모드에 따른 크기 조정을 위해.

결론

이 문서에서는 검증 테스트, 계층형 스토리지 성능 결과, 튜닝, Confluent S3 커넥터, 셀프 밸런싱 기능을 포함하여 NetApp 스토리지와 함께 Confluent Tiered Storage를 사용하기 위한 모범 사례 가이드라인을 제공합니다. ILM 정책, 검증을 위한 여러 성능 테스트를 통한 Confluent 성능, 업계 표준 S3 API를 고려할 때 NetApp StorageGRID 개체 스토리지는 Confluent 계층형 스토리지에 최적의 선택입니다.

추가 정보를 찾을 수 있는 곳

이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹사이트를 검토하세요.

- Apache Kafka란 무엇입니까?

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

- NetApp 제품 문서

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

- S3-싱크 매개변수 세부 정보

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

- 아파치 카프카

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

- Confluent 플랫폼의 무한 스토리지

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

- Confluent Tiered Storage - 모범 사례 및 크기 조정

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

- Confluent Platform용 Amazon S3 싱크 커넥터

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

- 카프카 크기 조정

  "https://eventsizer.io"

- StorageGRID 크기 조정

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

- 카프카 사용 사례

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

- Confluent 플랫폼 6.0에서 Kafka 클러스터의 자체 균형 조정

  "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 고객 사례 및 사용 사례 세부 정보

  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/