백서

퓨어스토리지 플래시블레이드 (FlashBlade) 기반의 분산형 Elasticsearch: 참조 아키텍처

퓨어스토리지 플래시블레이드에서 대규모 Elasticsearch® 구현을 위한 프레임워크

목차

소개

이 문서는 플래시블레이드(FlashBlade) 플랫폼에서 Elasticsearch를 설계 및 구축하고자 하는 시스템 관리자, 스토리지 관리자, IT 관리자, 시스템 아키텍트, 영업 엔지니어, 현장 컨설턴트, 전문 서비스 및 파트너를 위해 마련된 것입니다. Elasticsearch, Linux®, 서버, 스토리지 및 네트워크에 대한 실무 지식이 있다고 가정하지만 이 문서를 읽는 데 반드시 필요하지는 않습니다.

참조 아키텍처에서 다뤄질 내용은 다음과 같습니다 :

- 분산형 직접 연결 스토리지(DDAS)와 기타 네트워크 연결 스토리지(NAS)를 사용하는 기존 Elasticsearch 대비, 플래시블레이드에 최신 Elasticsearch 아키텍처를 구축할 경우 얻을 수 있는 이점에 대해 설명합니다.
- 선형적인 확장성, 핫(hot) 쿼리와 임시(ad-hoc) 쿼리에 대한 DDAS 초과 성능, 간단한 클러스터 관리 및 업그레이드 등의 이점을 입증하는 테스트 결과를 살펴봅니다

플래시블레이드 기반의 확장 가능한 분산형 Elasticsearch 구현

최신 앱, 사물인터넷(IoT), 데브옵스 및 마이크로서비스의 확산으로 인해 시스템에서 생성되는 로그 데이터가 폭발적으로 증가하고 있습니다. 기업 입장에서는 이제 인프라를 연중무휴 24시간 운영해야 할 필요성이 증대되고 있습니다. 모든 인프라 구성 요소(서버, 스토리지, 네트워크, 방화벽, 소프트웨어 등)를 모니터링하는 것이 점점 더 중요한 과제가 되어 가고 있습니다. 문제가 발생하면 가장 보편적인 해결 방법은 로그를 검사하는 것입니다. Elasticsearch는 분산된 인프라에서 발생하는 로그를 분석하는 중앙 저장소를 통해 기업이 복잡한 로그 관리와 보안 문제를 해결할 수 있도록 지원합니다.

기존에는 DDAS 아키텍처에 구축된 Elasticsearch 클러스터가 높은 가용성을 제공했지만, 인프라가 확장됨에 따라 운영상의 문제가 발생하기 시작했습니다. 예측 불가능한 검색 성능, 인프라 복잡성 및 운영 오버헤드, 유연하지 못한 사일로로 인한 낮은 리소스 활용도가 이러한 문제에 포함됩니다.

Elasticsearch와 퓨어스토리지는 기존 DDAS 아키텍처를 통해 이러한 도전과제를 해결해주기 때문에, 기업은 인프라 상면을 줄이고 고객 경험에만 집중할 수 있습니다. 퓨어스토리지는 올플래시 스토리지 어레이로 스토리지 인프라의 복잡성과 비효율성을 제거해주는 업계 선두 기업입니다. 플래시블레이드는 DDAS 솔루션과 동일한 인제스트 및 핫 쿼리 성능을 제공하는 동시에 컴퓨팅과 스토리지를 분리하여 간소화합니다.

이 문서에서는 플래시블레이드를 사용해 Elasticsearch를 구현하기 위한 참조 아키텍처를 제공합니다. 플래시블레이드는 이러한 분산형 솔루션을 통해 Elasticsearch를 위한 주요 이점을 제공합니다.

- **모든 데이터에 대해 더 빠르게 인사이트 확보:** 여러 스토리지 계층을 제거하고 '콜드(cold)' 계층까지 모든 데이터에 대해 일관된 성능을 제공합니다.
- **운영 간소화:** 컴퓨팅과 스토리지를 분리하여 운영을 간소화를 제공하고 독립적으로 확장할 수 있는 역량을 제공합니다.
- **다양한 워크로드 요구사항에 대한 민첩성 향상:** 여러 탄력적인 워크로드를 가상화하고 서비스형 기능을 통해 민첩성을 제공합니다.

- **총소유비용(TCO) 절감**: 인프라 활용도를 최적화하고 분산된 DAS 구현 대비 컴퓨팅과 스토리지 요구 사항을 낮춥니다.

이 참조 아키텍처를 검증하기 위해 두 가지 중요한 사항을 입증하도록 측정했습니다. 하나는 플래시블레이드로 문서를 인덱싱하여 인제스트 성능을 측정했고, 다른 하나는 플래시블레이드에서 로그 데이터를 가져와 검색 성능을 측정했습니다.

Elasticsearch 7.2와 CENTOS 7이 각각 데이터 노드와 마스터 노드로 실행되도록 구성된 40대의 서버로 테스트를 수행했습니다. 또한 Rally 벤치마크 툴을 사용한 로드 생성에 6대의 서버를 사용했으며, 각각 서버 1대가 Kibana, Apache Bench 및 Apache JMeter를 실행하도록 구성했습니다. 각 Elasticsearch 노드에는 플래시블레이드(/mnt/esdata)에 단일 마운트 지점을 구성해 Elasticsearch 인덱스를 저장하도록 했습니다. 로드 생성기 노드에는 다른 플래시블레이드(/mnt/rallydata)에 단일 마운트 지점을 생성해 대량 인제스트 테스트를 위해 원시 데이터를 읽도록 했습니다.

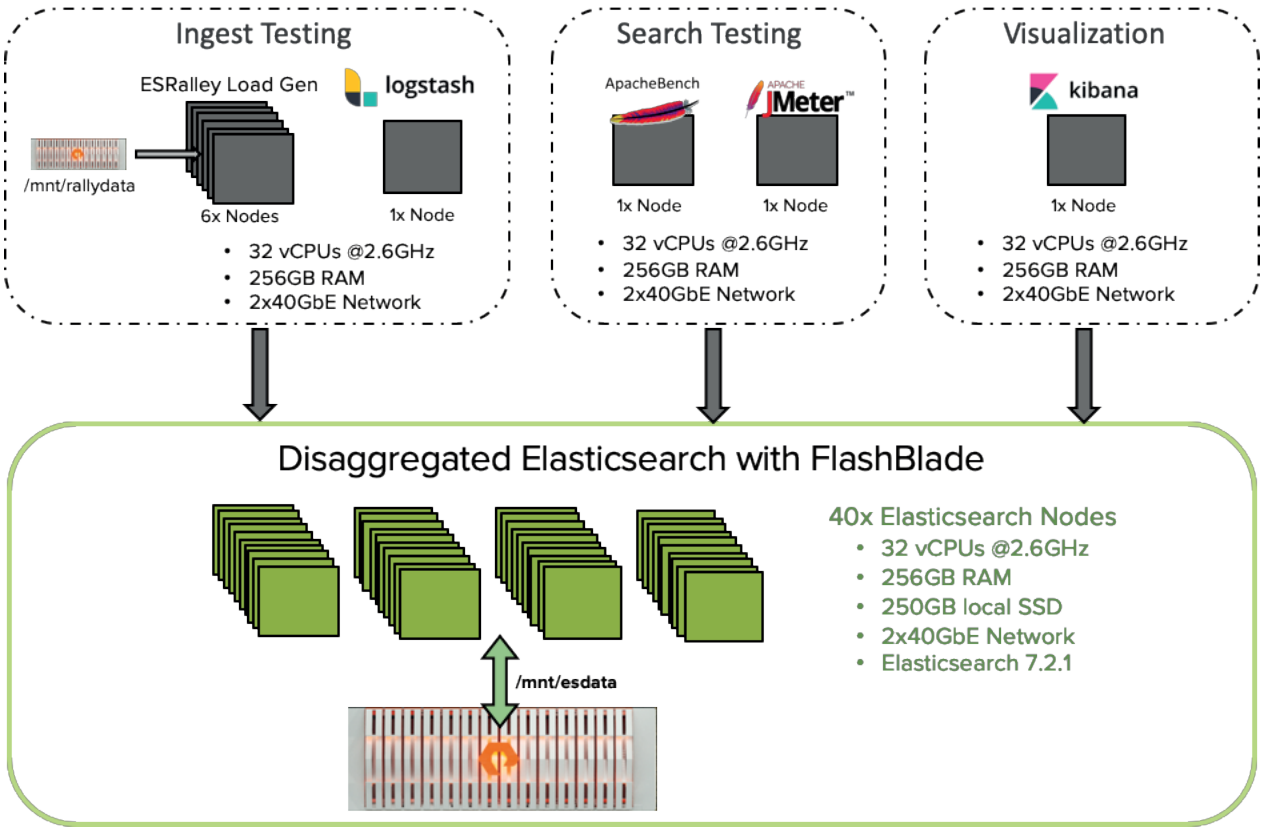

그림 1: 테스트 설정

테스트를 위해 맞춤화된 NYC 택시 데이터 세트와 Apache Logs에 기반한 데이터 세트 두 가지를 사용했습니다. NYC 택시는 정형 데이터 세트(Rally 벤치마크의 일부)로, 더 긴 실행 시간을 통해 보다 정확한 인제스트 측정값을 얻기 위해 8배의 데이터 크기가 생성되도록 지정했습니다. Apache Logs를 기반으로 하는 비정형 데이터 세트는 16TB의 원시 로그 데이터를 생성하도록 지정됐습니다.

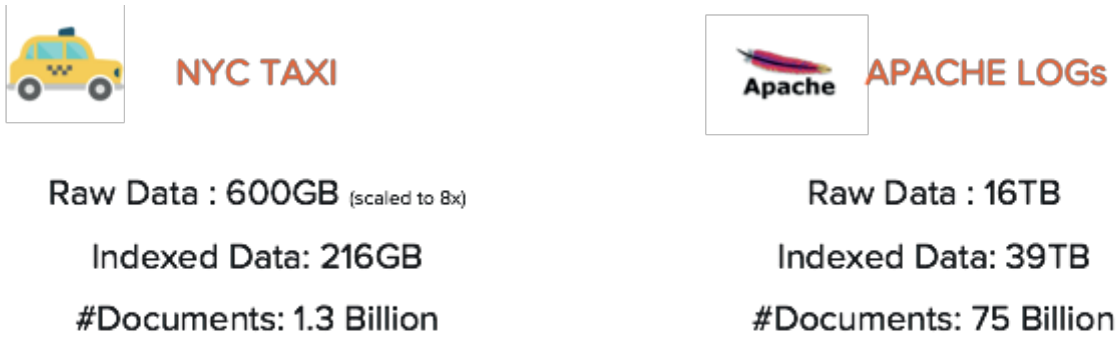

그림 2: 데이터세트 테스트

아래 그림 3은 Elasticsearch 노드의 확장에 따라 DDAS와 플래시블레이드의 인제스트 성능이 어떻게 달라지는지를 보여 줍니다. 그래프에는 두 가지 사항이 강조 표시되어 있습니다. 첫째, 플래시블레이드는 분산 DAS 클러스터와 동일하거나 더 뛰어난 성능을 제공합니다. 둘째, 플래시블레이드는 Elasticsearch 노드의 수가 증가하더라도 로그 데이터 인제스트를 위한 선형적 확장성을 제공합니다. 따라서 플래시블레이드는 Elasticsearch 클러스터를 구현하는 기업에 성능 저하 없이 분리로 인한 간소화를 제공합니다.

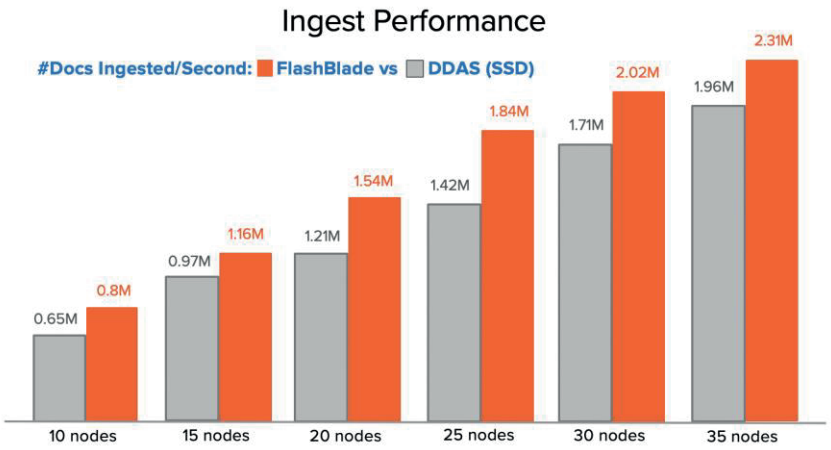

그림 3: DDAS와 플래시블레이드에 대한 인제스트 노드 성능 비교

그림 4는 핫 쿼리(서버의 DRAM에서 쿼리된 Elasticsearch의 "핫" 데이터)와 임시 쿼리(플래시블레이드에서 쿼리된 Elasticsearch의 "콜드" 데이터)의 성능을 보여 줍니다. 이미지의 윗부분은 핫 쿼리와 임시 쿼리를 모두 실행할 때의 성능을 보여 줍니다. 이미지 하단은 쿼리가 실행되는 동안 플래시블레이드 스토리지 성능을 보여줍니다. 핫 쿼리는 로컬 서버에서 액세스되므로, 임시 쿼리의 배치 3개를 실행하는 동안에만 스토리지 트래픽이 표시됩니다. 그래프에서 임시 쿼리를 실행해도 핫 쿼리 성능에 거의 또는 전혀 영향을 미치지 않는 것을 확실하게 볼 수 있습니다. 또한 콜드 데이터는 올플래시 플래시블레이드의 전체 읽기 대역폭을 활용할 수 있으므로 임시 쿼리와 핫 쿼리의 성능이 비슷합니다.

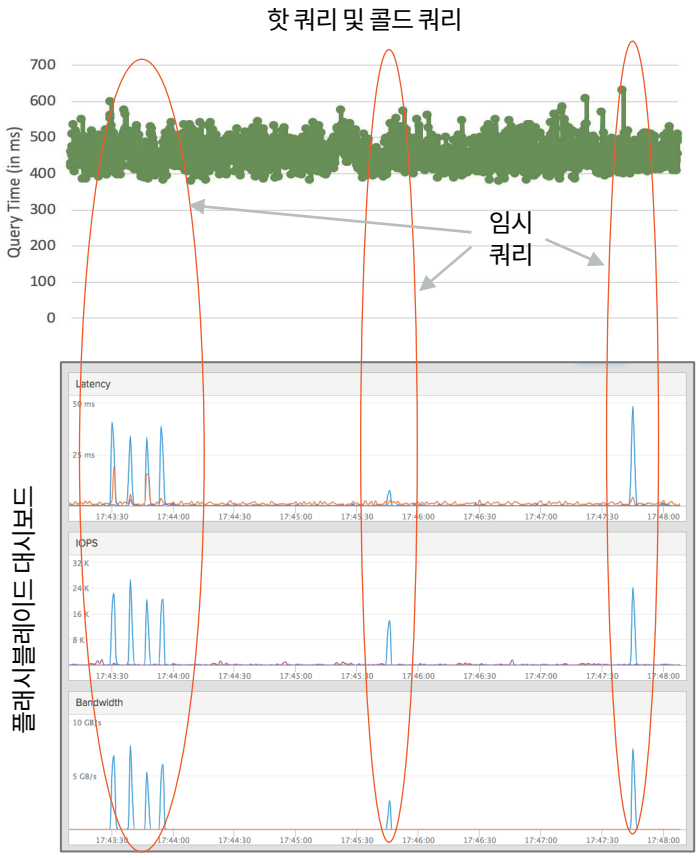

그림 4: 핫 쿼리 및 임시 쿼리 성능

보다 자세한 참조 아키텍처 설계는 부록 A를, 테스트 결과는 부록 B를 참조하십시오.

결론

Elasticsearch는 시장 점유율과 사용자 수가 지속적으로 늘고 있는 선도적인 검색 공급업체입니다. 빠른 인제스트와 긴 데이터 보존이 필요한 모든 활용 사례에서 Elasticsearch를 구현하는 경우가 크게 증가하고 있습니다. 이런 경우, DDAS 아키텍처의 근본적인 한계점이 강조되고 Elasticsearch의 효율성은 감소됩니다. 플래시블레이드에 기반한 Elasticsearch는 간단하고 확장 가능한 아키텍처를 제공하여 기업이 목표를 더 빨리 달성할 수 있도록 지원합니다.

이 솔루션은 플래시블레이드의 확장성과 간소성이 Elasticsearch의 아키텍처를 어떻게 보완해주는지 보여 줍니다. 플래시블레이드는 스토리지 계층을 제거하여 사용 빈도가 높은(핫) 데이터와 사용 빈도가 낮은(콜드) 데이터를 모두 신속하게 파악하게 합니다. 계층 제거를 하면 1개의 복제본 샤드를 사용해 스토리지 사용률을 두 배로 높여주는 동시에, 데이터 보호 기능은 동일하게 제공됩니다. 올플래시 읽기 성능으로 인해 임시 쿼리도 핫 계층과 같은 성능을 제공합니다.

부록 A: 세부적인 참조 아키텍처 디자인

이 참조 아키텍처를 구현하기 위한 기본 원칙은 다음과 같습니다.

- **간소화** – 불필요하거나 복잡한 구성 또는 기술을 피하여 즉시 사용 가능한 일반적인 환경에서 보다 더 나은 결과를 얻을 수 있습니다.
- **반복성** – 모든 고객 사이트에서 쉽게 복제할 수 있는 확장 가능한 빌딩 블록을 생성합니다. 테스트 중인 펌웨어 버전을 게시하고 고객이 솔루션을 구현하기 전에 랩에서 문제를 확인할 수 있습니다.
- **가용성** – 가용성이 높은 Elasticsearch 아키텍처를 퓨어스토리지의 무중단 업그레이드(NDU) 기능으로 보완합니다.
- **효율성** – Elasticsearch의 확장 역량을 활용하며 퓨어스토리지의 낮은 레이턴시와 높은 IOPS와 처리량, 공간, 전력 및 냉각 효율성을 확보할 수 있습니다.
- **확장성** – 플래시블레이드에 구현된 Elasticsearch는 선형적 확장이 가능합니다.

플래시블레이드와 Elasticsearch가 해결해주는 문제

Elasticsearch는 분산형 RESTful 검색 및 분석 엔진으로, 점점 더 많은 사용 사례에 적용되고 있습니다. 이 솔루션은 모든 규모의 기업들이 복잡한 데이터 관리와 네트워크 보안 문제를 해결하고 비즈니스를 확장할 수 있도록 지원합니다. 광범위하고 정교한 API를 통해 액세스할 수 있는 Elasticsearch는 데이터 검색 애플리케이션을 지원하는 매우 빠른 검색 서비스를 제공할 수 있습니다.

일반적으로 Elasticsearch는 스케일 아웃 DAS 기반의 상용 서버를 통해 간소화를 제공합니다. 그러나 데이터 세트가 100TB 규모로 증가하면 구현에 특정 서버 구성이 필요합니다. 성능 문제를 해결하기 위해 기존의 DAS 기반 아키텍처를 사용하는 기업은 확장함에 따라 비용이 늘어나는 문제에 직면하게 됩니다. 또한 시간이 지남에 따라 검색 성능이 저하되고 일관성이 없어집니다.

이러한 한계를 해결하려면 컴퓨팅과 스토리지를 분리하는 것이 확실한 선택인 듯하지만, 이로 인해 발생하는 레이턴시가 Elasticsearch의 성능에 영향을 미칩니다. 퓨어스토리지의 플래시블레이드를 사용하면 Elasticsearch의 확장성 문제를 해소하는 동시에 검색 쿼리에 최고 성능과 최저 레이턴시를 제공할 수 있습니다.

기술 개요

플래시블레이드

퓨어스토리지는 데이터 기반 비즈니스의 스토리지 요구 사항을 충족하기 위해 플래시블레이드 아키텍처를 개발했습니다. 플래시블레이드는 비정형 데이터의 저장 및 처리에 최적화된 올플래시 시스템입니다. 플래시블레이드 시스템은 수천 개의 클라이언트를 위해 여러 파일 시스템과 멀티 테넌트 오브젝트 저장소를 동시에 호스팅할 수 있습니다.

플래시블레이드 시스템의 성능, 용량, 연결 확장 역량은 다음과 같은 5가지 주요 혁신 기술을 기반으로 합니다.

- **고성능 스토리지 장치**
  플래시블레이드는 데이터를 스토리지 장치에 저장하고, 기존의 회전식 디스크나 솔리드 스테이트 드라이브 같이 레이턴시가 높은 스토리지 미디어를 제거함으로써 올플래시 아키텍처의 이점을 극대화합니다. 확장 가능한 NVRAM이 각 스토리지 장치에 통합되어, 시스템에 새 블레이드를 추가하면 성능과 용량을 비례적으로 확장할 수 있습니다.

- **통합 네트워크**
  플래시블레이드 시스템은 클라이언트와 내부 관리 호스트 간의 높은 통신 트래픽을 이더넷 링크를 통해 최대 100Gb/s의 IPv4 및 IPv6 클라이언트 액세스를 모두 지원하는 신뢰할 수 있는 단일 고성능 네트워크로 통합합니다.
- **퓨리티//FB(Purity//FB) 스토리지 운영 체제**
  플래시블레이드의 패브릭 모듈에서 대칭적으로 운영 체제를 실행하는 퓨리티//FB는 플래시블레이드의 블레이드 간에 모든 클라이언트 작업 요청을 균등하게 분산시켜 워크로드 밸런싱 문제를 최소화합니다.
- **파일 및 오브젝트를 위한 공통된 미디어 아키텍처 설계**
  플래시블레이드의 단일 기저 미디어 아키텍처는 전체 플래시블레이드 구성에서 NFSv3, NFS over HTTP, SMB(Samba 수준 기능 사용) 등의 다양한 프로토콜과 Amazon S3를 통해 각각 파일과 오브젝트에 대한 동시 액세스를 지원합니다.
- **간편한 사용**
  플래시블레이드의 퓨리티//FB는 일상적인 관리 작업을 자율적으로 수행함으로써 스토리지 운영을 간소화하고 시스템 관리 부담을 줄여줍니다. 견고한 운영 체제를 갖춘 플래시블레이드는 구성 요소에 장애가 발생할 경우 자동으로 튜닝을 수행하고 시스템 알림을 제공합니다.

플래시블레이드 하드웨어

완전한 플래시블레이드 시스템 구성은 2개의 외부 패브릭 모듈(XFM)로의 고속 링크를 통해 상호 연결된 최대 5개의 독립형 랙 마운트 섀시로 구성됩니다. 각 섀시의 후면에는 고속 이더넷을 통한 TCP/IP를 사용해 블레이드, 다른 섀시 및 클라이언트를 상호 연결하기 위한 2개의 온보드 패브릭 모듈이 있습니다. 두 개의 패브릭 모듈은 서로 연결되어 있으며 각각 컨트롤 프로세서와 이더넷 스위치 ASIC를 포함합니다. 안정성을 위해 각 섀시에는 예비 전원 공급 장치와 냉각 팬이 장착되어 있습니다.

각 섀시의 전면에는 데이터 연산과 저장을 위해 최대 15개의 블레이드가 장착됩니다. 각 블레이드 어셈블리는 프로세서, 통신 인터페이스 및 영구 데이터 저장을 위한 17TB 또는 52TB의 플래시 메모리를 갖춘 독립형 컴퓨팅 모듈입니다.

현재 플래시블레이드 시스템은 초당 150만여 개의 NFSv3 getattrs를 지원할 수 있습니다. 또는 15개의 블레이드가 포함된 단일 4U 섀시에서 3:1로 압축 가능한 데이터 세트에 대한 512KiB 읽기를 최소 17GiB/초에, 512KiB 덮어쓰기를 최소 8GiB/초에 수행할 수 있으며, 75개의 블레이드가 있는 최대 5 x 4U 섀시까지 컴퓨팅과 성능을 확장할 수 있습니다.

플래시블레이드 기술 사양에 대해 자세히 알아보십시오.

Elasticsearch

Elasticsearch는 Apache Lucene 검색 엔진을 기반으로 NoSQL 데이터 저장소에 정보를 인덱싱하고 저장하는 오픈소스 소프트웨어입니다. Elasticsearch는 Elastic Stack(ELK)의 핵심으로, Logstash를 사용해 데이터를 처리하고 Elastic 인덱서로 스트리밍하며, Kibana를 사용해 결과를 쿼리하고 사용자 친화적인 대시보드에 표시합니다.

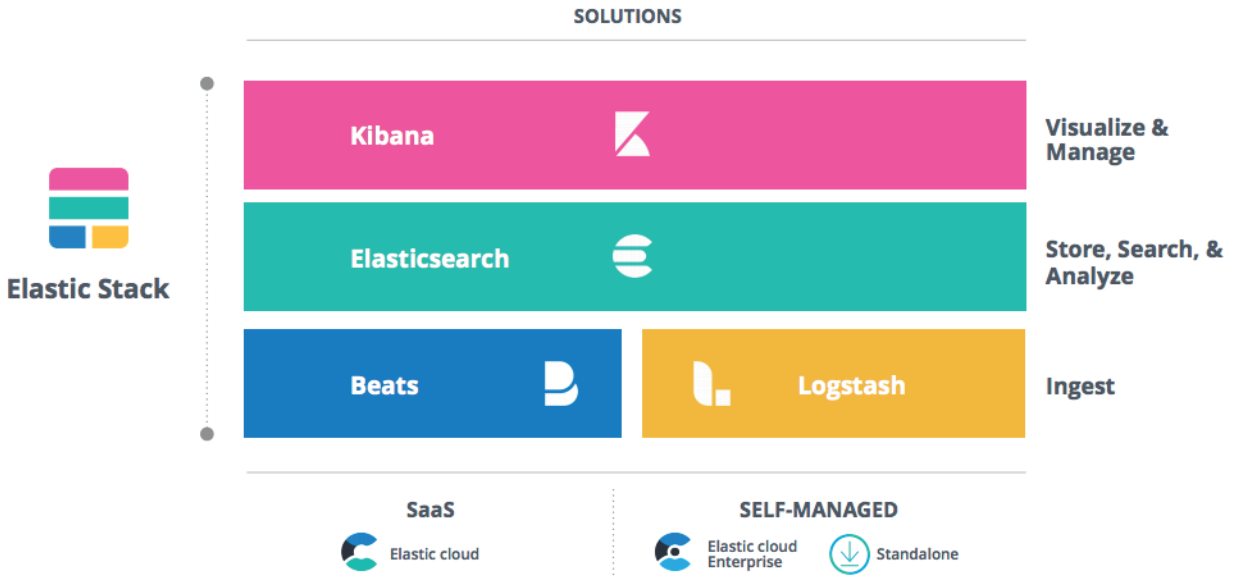

그림 5: Elastic 스택의 부품

Elasticsearch는 다음과 같은 주요 이점을 제공합니다.

- **특히 규모가 크고 증가하는 데이터베이스 환경에 적합한** Elasticsearch 분산 아키텍처는 수천 개의 노드까지 확장할 수 있으며 클러스터 솔루션은 페타바이트 규모의 데이터를 수용할 수 있도록 확장이 가능합니다. Elastic Common Schema(ECS)는 이러한 환경에서 멀티 테넌시를 지원합니다. 분산 인덱스는 각 샤드를 위한 복제본을 지원하는 샤드로 나눠집니다. 라우팅 및 리밸런싱은 IT 부담을 덜어주기 위해 자동으로 수행됩니다.
- **Elasticsearch는 검색을 간소화하여** 다양한 활용 사례(예: 기존의 전체 텍스트 검색, 분석, 자동 완성, 맞춤법 검사기, 알림 엔진 및 범용 데이터 저장소)에 적용할 수 있습니다. 다양한 프로그래밍 언어와의 손쉬운 통합을 지원하기 위해 API 및 쿼리 DSLS를 사용해 검색 기능을 확장합니다. Elasticsearch는 텍스트 분할, 맞춤화된 형태소 분석, 다차원적 검색, 전체 텍스트 검색, 자동 완성, 빠른 검색, 퍼지 검색 및 제안 등을 지원합니다. 문서 지향적으로 복잡한 데이터 구조를 json 형식으로 저장하고 기본적으로 모든 필드를 인덱싱합니다.
- **고성능** 결과를 RESTful API 및 캐시된 검색 필터를 사용해 얻을 수 있으며, 트랜스로그(translog)의 변경 사항을 캡처하여 데이터 무결성을 유지할 수 있습니다. 최신 버전에는 머신러닝(ML) 모듈이 통합되어 있습니다.

Elastic 데이터 스토리지

Elasticsearch 패러다임에서 문서는 인덱스 내에 기본 스토리지 단위로 저장된 JSON 오브젝트입니다. 각 문서 내에는 키와 값으로 구성된 필드가 있습니다.

매핑은 특정 데이터 유형의 문서에 대한 필드를 정의하고 해당 필드를 인덱싱하며 Elasticsearch에서 저장하는 방법을 결정합니다.

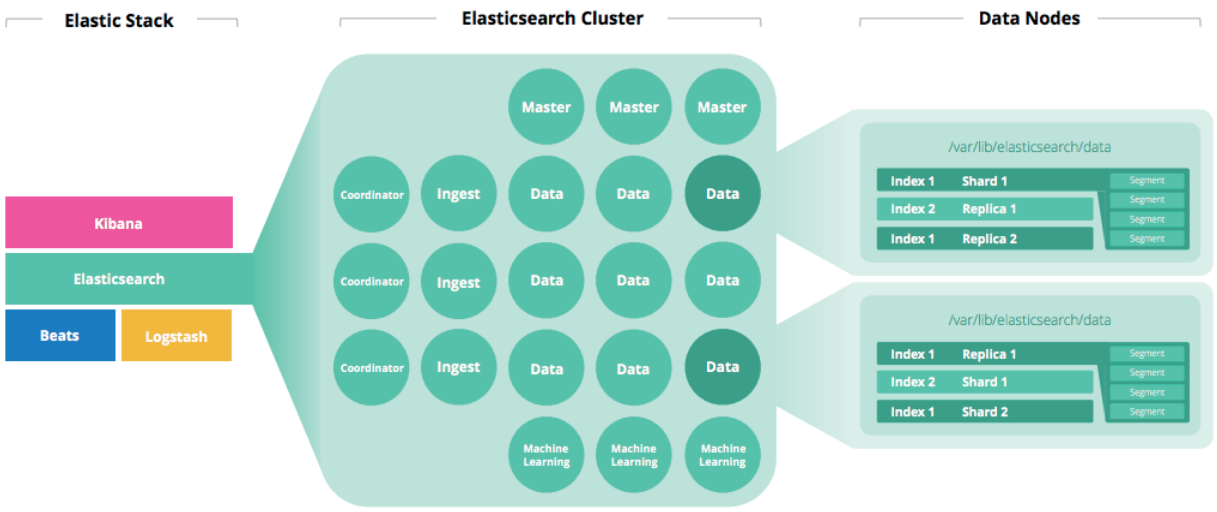

그림 6: Elasticsearch 아키텍처

하나의 샤드는 Elasticsearch의 빌딩 블록을 구성하고 확장성을 지원하는 단일한 Lucene 인덱스입니다. 인덱스는 Elasticsearch에서 가장 큰 데이터 단위로, 샤드처럼 문서의 논리적 파티션입니다.

인덱스를 생성하는 동안 사용자는 샤드 수를 정의합니다. 각 샤드는 클러스터의 어느 위치에서나 호스팅 될 수 있는 독립적인 Lucene 인덱스입니다. 복제본 샤드는 인덱스의 1차 샤드의 복사본입니다. 노드 충돌 시 백업 및 복원에 사용됩니다.

Elasticsearch 노드는 하나 이상의 역할을 수행할 수 있지만, 노드가 리소스를 두고 경합하지 않도록 노드당 하나의 역할을 할당하고 각 역할을 위한 하드웨어를 최적화하는 것이 좋습니다.

데이터 노드는 데이터를 저장하고 검색과 집계 같은 데이터 관련 연산을 실행합니다.

마스터 노드는 클러스터 전체의 관리 및 구성 작업(예: 노드 추가 및 제거)을 처리하고 다른 노드가 해당 리소스를 사용하지 못하도록 방지하여 클러스터의 안정성을 보장합니다.

클라이언트 노드는 클러스터 요청을 마스터 노드로 전달하고 데이터 관련 요청을 데이터 노드로 전달합니다.

인제스트 노드가 여러 파이프라인을 실행하거나 여러 프로세서를 사용하는 경우 추가 컴퓨팅 리소스가 필요합니다.

머신러닝이 많은 작업을 실행하거나 분할, 버킷 또는 복잡한 집계를 사용하는 경우 추가 메모리 및 컴퓨팅 리소스가 필요합니다.

전용 코디네이터 노드는 지속적으로 인덱싱되는 데이터 노드에서 검색의 병합 단계를 오프로드하여 하이브리드 활용 사례에 도움이 될 수 있습니다.

노드 데이터:

```
$ tree data
data
└── elasticsearch
    └── nodes
        └── 0
            ├── _state
            |   └── global-0.st
            └── node.lock
```

그림 7: Elastic 노드를 시작할 때 비어 있는 데이터 디렉터리

`node.lock` 파일은 한 번에 하나의 Elasticsearch 설치만 단일 데이터 디렉터리에서 읽기/쓰기를 수행할 수 있도록 합니다.

인덱스 데이터:

인덱스를 생성하면 데이터가 다양한 폴더에 저장됩니다.

```
$ tree -h data
data
└── [ 102]  elasticsearch
    └── [ 102]  nodes
        └── [ 170]  0
            ├── [ 102]  _state
            |   └── [ 109]  global-0.st
            ├── [ 102]  indices
            |   └── [ 136]  foo
            |       ├── [ 170]  0
            |       |   ├── .....
            |       └── [ 102]  _state
            |           └── [ 256]  state-0.st
            └── [   0]  node.lock
```

그림 8: 다양한 폴더에 저장된 데이터의 인덱싱

샤드 데이터:

샤드 데이터 디렉터리에는 샤드가 기본 샤드로 간주되는지, 아니면 복제본으로 간주되는지에 대한 정보와 버전 정보 등 샤드의 상태 파일이 포함되어 있습니다.

```
$ tree -h data/elasticsearch/nodes/0/indices/foo/0
data/elasticsearch/nodes/0/indices/foo/0
├── [ 102]  _state
│   └── [  81]  state-0.st
├── [ 170]  index
│   ├── [  36]  segments.gen
│   ├── [  79]  segments_1
│   └── [   0]  write.lock
└── [ 102]  translog
    └── [  17]  translog-1429697028120
```

그림 9: 샤드 데이터 디렉터리 구조

Elasticsearch 트랜잭션 로그를 사용하면 모든 문서에 대해 하위 레벨 Lucene 커밋을 수행할 필요없이 데이터를 Elasticsearch에 안전하게 인덱싱할 수 있습니다. Lucene 인덱스를 커밋하면 Lucene 레벨에서 fsync()로 생성된 새 세그먼트가 생성되어 성능에 영향을 미치는 상당한 양의 디스크 I/O가 발생합니다.

들어오는 문서는 처음에 트랜스로그에 저장된 다음 세그먼트에 저장됩니다. 여러 번의 병합 작업이 세그먼트에 발생할 수 있습니다.

인덱싱에는 충분히 낮고 일정한 레이턴시와 높은 쓰기 처리량이 필요합니다. 보유한 데이터에 대한 광범위한 테스트를 수행하길 권합니다.

분리는 로컬 드라이브로는 불가능한 서비스 제공과 확장 모델을 지원합니다.

기업들의 도전과제

코로케이션 스토리지(Collocated storage)

Elasticsearch 클러스터 구현은 높은 데이터 가용성과 충실도를 제공하지만 상당한 비용 문제를 야기하는 분산 스케일 아웃 모델을 기반으로 합니다. Elastic 노드는 샤드를 복제하도록 구성되어 있어, 데이터 손실을 방지하고 노드 장애 시 검색을 용이하게 합니다. 이러한 모델은 서버와 스토리지 간의 근접성에 의존해 고성능을 실현하는 Hadoop 같은 이전의 빅데이터 기술에 적합합니다.

분산 스케일 아웃 모델에서 모든 Elastic 데이터 노드는 주로 직접 연결 스토리지(DAS)를 통해 핫/웜 및 콜드 계층용과 유사한 스토리지 사이즈가 포함되도록 구성됩니다. 이 모델은 과거에 필요한 데이터 볼륨을 처리하는 데 효과적이었습니다. 그러나 데이터가 증가하면서, 스토리지와 컴퓨팅 요구 사항은 선형적으로 확장되지 않습니다. 스토리지 요구 사항을 해결하기 위해 컴퓨팅과 스토리지에 같은 유형의 노드를 추가하는 것은 최적의 성과를 내지 못하고 비용도 많이 듭니다.

그림 10: DDAS를 사용한 기존 Elasticsearch

데이터 사용, 계층화 및 보존 요구 사항

분산 스케일 아웃 모델의 주요 요구 사항은 복제본 샤드가 복제 요소에 기반해 클러스터의 추가 노드에 저장된다는 것입니다. 따라서 클러스터 환경에서 높은 수준의 복제를 수행하면 스토리지 요구 사항이 크게 증가하게 됩니다.

용량 계획

효율적인 운영 성능을 위해 확장을 계획하고 용량을 프로비저닝하는 것이 중요합니다. 이는 Elasticsearch 클러스터가 검색 또는 인덱스 작업의 급증 상황을 수용할 수 있는 용량으로 노드를 가동 및 실행할 수 있게 설계되어야 한다는 것을 의미합니다. 동시에 메모리가 통제 범위 밖으로 확장되지 않도록 하고 예기치 않은 작업/이벤트가 노드를 중단시키는 것을 방지해야 합니다.

용량 계획을 위한 공식이 정해져 있진 않지만, 구체적인 모범 사례는 도움이 될 수 있습니다.

- 실제 활용 사례를 시뮬레이션하여, 노드를 설정하고 샤드가 견딜 수 없을 때까지 실제 데이터 세트로 채웁니다.
- 샤드 크기, 대량 인덱싱 클라이언트, 대량 크기를 다양하게 조합해 봅니다.
- 최적의 수치를 결정하고 정확한 샤드 크기 및 갱신 간격으로 인덱스를 구성합니다.

샤드의 용량을 정의한 후에는 전체 인덱스에 적용할 수 있습니다. 테스트 프로세스 중에 최적의 리소스 활용도를 파악하여 노드 및 JVM 힙 공간을 위한 적절한 RAM 용량을 결정하는 것이 중요합니다.

솔루션

이 문서에서 설명하는 솔루션은 분산 노드를 위한 스토리지 계층으로 플래시블레이드를 사용하는 Elasticsearch로 구성되어 있습니다.

플래시블레이드는 퓨어스토리지의 혁신적인 스케일 아웃 올플래시 파일 및 오브젝트 스토리지 시스템으로, 전체 데이터 사일로를 통합하는 동시에 Elasticsearch와 같은 애플리케이션을 사용하여 시스템 데이터에서 실시간 인사이트를 가속화하도록 설계되었습니다.

Centos 7.4를 실행하는 물리적 시스템이 Elastic 노드에 사용되었습니다.

CentOS 버전 7.4는 Red Hat Enterprise Linux(RHEL) 소스에서 파생된 배포판으로, 엔터프라이즈급 컴퓨팅 플랫폼 기능을 무료로 제공합니다.

현재 플래시블레이드 시스템은 최대 17Gbps의 읽기 속도와 4.5Gbps의 쓰기 속도를 지원할 수 있습니다. 15개의 블레이드가 갖춘 단일 4U 섀시에서 75개의 블레이드를 갖추고 최대 85Gbps의 읽기 속도와 20Gbps의 쓰기 속도를 지원하는 5 x 4U 섀시로 확장할 수 있습니다.

퓨어스토리지 플래시블레이드를 통한 Elasticsearch 가속화

기존 DAS 환경에서는 스토리지 리소스가 CPU 주기를 사용하며, 그렇지 않은 경우 인덱서 노드에 사용할 수 있습니다. 따라서 오버헤드가 추가되고 데이터 서비스가 없습니다. 반면 플래시블레이드 환경에서는 압축을 통해 Elasticsearch 클러스터의 스토리지 요구 사항을 크게 줄일 수 있습니다.

다음 다이어그램은 아키텍처의 스토리지 구성 요소로서 고성능 네트워크 파일 시스템(NFS)을 갖춘 Elasticsearch를 보여줍니다.

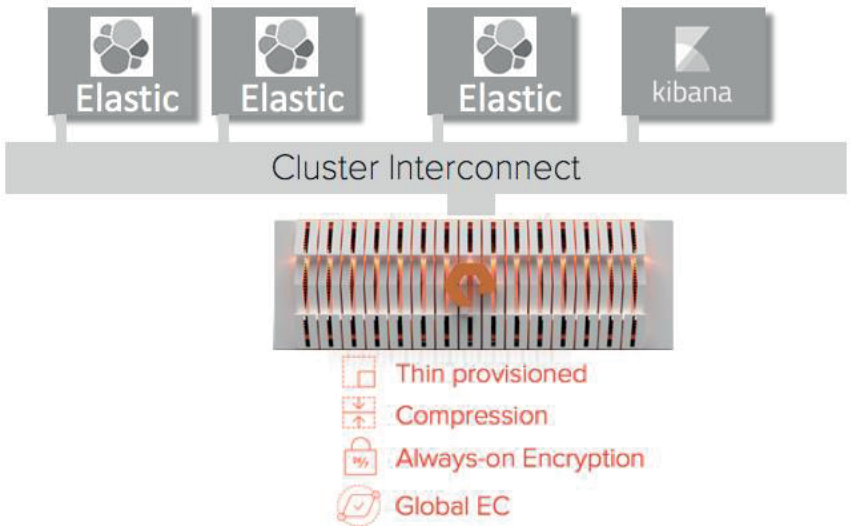

그림 11: NFS를 갖춘 Elasticsearch

RAID가 포함된 플래시블레이드는 데이터 보호 기능을 강화하며, 삭제 인코딩은 데이터 노드를 추가할 때마다 RAID 볼륨을 결정하고 생성해야 하는 시스템 관리자의 운영 부담을 줄여줍니다. 또한 플래시블레이드는 저장 데이터를 암호화된 상태로 유지하는 상시(always-on) 암호화 서비스를 제공합니다.

플래시블레이드는 DAS에 비해 Elasticsearch의 핵심 기능, 데이터 인제스트 및 검색에 더 높은 성능을 지원합니다. 플래시블레이드의 기술 덕분에, 이제 콜드 계층의 성능 특성이 핫/웜 계층의 성능 특성과 동일해 졌습니다. 핫/웜 및 콜드 계층에 존재하는 모든 데이터를 검색할 때 유사한 성능을 확보할 수 있습니다.

인덱스 프로세스:

인덱스 프로세스 중에, 샤드/Lucene에서 변경되는 사항은 Lucene 커밋 동안에만 스토리지에 유지됩니다. 이는 상대적으로 비용이 많이 드는 작업이어서 모든 인덱스나 삭제 작업 후에 수행할 수 없습니다.

Lucene 커밋은 개별 변경 시 수행하기에는 너무 비용이 많이 들기 때문에, 각 샤드 복사본에는 관련된 트랜스로그로 알려진 트랜잭션 로그도 있습니다. 모든 인덱스 및 삭제 작업은 내부 Lucene 인덱스에 의해 처리된 후 인지되기 전에 트랜스로그에 기록됩니다. Elasticsearch 플러시(flush)는 Lucene 커밋을 수행하고 새 트랜스로그를 시작하는 프로세스입니다.

플러시는 백그라운드에서 자동으로 수행되어 트랜스로그가 너무 커지지 않도록 합니다. 트랜스로그의 데이터는 트랜스로그가 fsync되고 커밋될 때만 디스크에 유지됩니다.

Elastic에서 일반적으로 사용되는 스토리지 유형은 NIO FS 유형으로, 이 유형은 NIO를 사용하여 파일 시스템(Lucene NIOFSDirectory에 매핑)에 샤드 인덱스를 저장합니다. 이는 여러 스레드가 동일한 파일에서 동시에 읽을 수 있도록 합니다.

그림 12: Elasticsearch에서의 인덱싱

종합해볼 때, 퓨어스토리지 시스템의 Elasticsearch는

- 확장성이 뛰어난 스토리지 솔루션을 제공합니다.
- 모든 스토리지 계층에 올플래시 성능을 제공합니다.
- 컴퓨팅 및 스토리지를 독립적으로 추가함으로써 동적 클러스터 확장을 가능케합니다.
- 압축을 통해 공간 사용량을 더 줄입니다.
- 암호화를 통해 저장된 Elastic 인덱스 데이터를 보호합니다.

솔루션 디자인

설계 토폴로지

이 섹션에서는 랩에서 테스트한 퓨어스토리지 시스템의 Elasticsearch 설계 토폴로지에 대해 설명합니다.

이 솔루션에는 Elasticsearch 노드를 호스팅하는 40대의 Intel CPU 기반 서버가 포함됩니다. 로드 생성기와 벤치마크 툴을 실행하는 데 별도로 6개의 노드가 사용되었습니다. 이 솔루션에는 데이터 스토리지 계층을 호스팅하는 15개의 블레이드가 장착된 퓨어스토리지 플래시블레이드가 포함되어 있습니다.

플래시블레이드 구성

플래시블레이드는 스토리지 레벨에서 압축 및 암호화된 Elasticsearch 인덱스 데이터를 호스팅합니다. 공유 NFS 파일 시스템은 이더넷을 통해 연결되고 Elastic 노드에 하드 마운트된 각 데이터 노드의 하위 폴더에 사용되었습니다.

구성 요소	설명
플래시블레이드	15 x 17TB 블레이드
용량	원시 용량 240TB
연결성	가용 용량 162.46TB(데이터 절감 안함)
외부 사양	4 x 40Gb/s 이더넷(데이터)
소프트웨어	2 x 1Gb/초 중복 이더넷(관리 포트)

운영 체제 및 소프트웨어 구성

OS 및 소프트웨어	설명
Linux	CentOS 7.4(64비트)
Elasticsearch	Elastic 7.2.1
Rally Benchmark	버전 1.1

물리적 토폴로지

퓨어스토리지에 기반한 Elasticsearch 솔루션은 하드웨어(컴퓨팅, 스토리지, 네트워크) 및 소프트웨어(Elasticsearch, CentOS 리눅스)가 결합된 스택으로 구성됩니다.

구성 요소	설명
Elastic 노드	아래 사양의 Intel 기반 서버 40대: · 2 x Intel Xeon Gold 6138 @ 2GHz(20코어) · 256GB 메모리
벤치마크 노드	아래 사양의 Intel 기반 서버 6대: · Intel Xeon 프로세서 E5-2670 v3 CPU 2개(코어 12개) · 256GB 메모리
Logstash	아래 사양의 Intel 기반 서버 6대: · Intel Xeon 프로세서 E5-2609 v4 CPU 2개(코어 8개) · 64GB 메모리
네트워킹	TOR 2x Cisco Nexus 9372PX 이더넷 스위치
가상 인터페이스 카드:	Cisco UCS VIC 1340의 40Gbps 통합 I/O 포트

논리적 토폴로지

매일 수 테라바이트의 데이터를 수집하면서 수많은 동시 검색을 지원해야 하는 엔터프라이즈급 Elasticsearch 구현의 경우, Elastic 클러스터의 분산 구현을 권장합니다.

랩 테스트 환경에서는 다음을 사용했습니다.

- Elastic 데이터 노드 40개
- 벤치마크 코디네이터 노드 1개
- 벤치마크 로드 생성기 노드 5개
- 동일한 벤치마크 노드의 Logstash 노드 6개

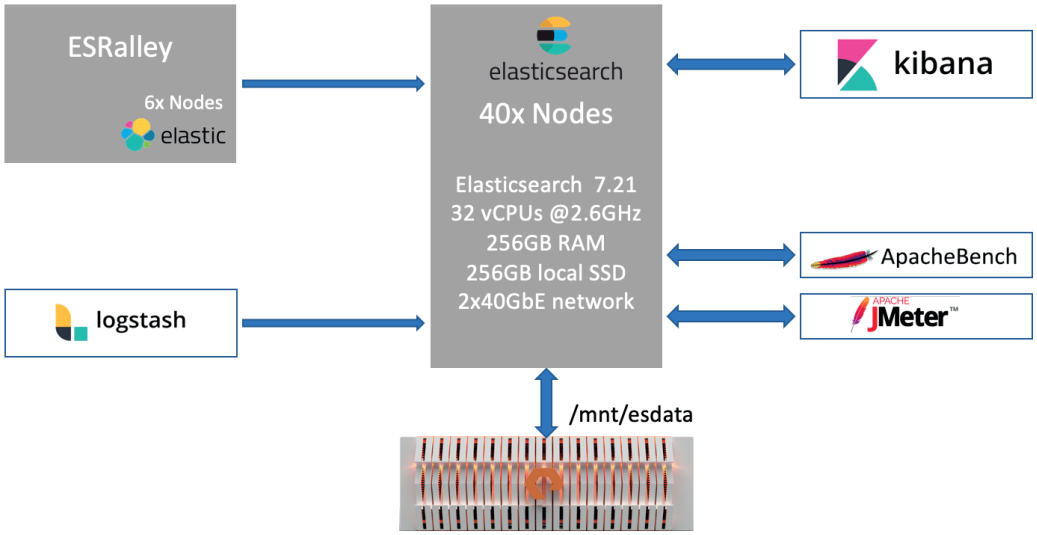

그림 13: 벤치마크 설정

설계 고려 사항

Elasticsearch 구현은 일반적으로 다음과 같은 요소를 기반으로 합니다.

- **일일 인제스트 속도 또는 인덱싱 볼륨** - 인덱스 속도가 높을수록 더 많은 인덱스 노드가 필요합니다.
- **검색 수 및 유형** - 수많은 동시 검색 또는 리소스 집약적인 검색(예: 고밀도 검색)은 검색 헤드와 인덱서에 부담을 줄 수 있습니다.
- **동시 사용자 수** - 대시보드를 보거나 검색을 하는 사용자가 많은 경우 더 많은 검색 헤드가 필요하며, 검색 헤드 클러스터가 이상적입니다.
- **데이터 충실도** - 페일오버 시나리오를 처리하기 위한 복제 계수를 결정합니다.
- **데이터 가용성** - 데이터 노드만 인덱스 데이터의 전체 집합을 검색하고 액세스할 수 있습니다.
- **재해 복구 요구 사항** - 두 데이터센터 사이에 분산된 멀티사이트 Elastic 클러스터는 동일한 데이터 세트를 유지하고 신속하게 복구할 수 있도록 합니다.

Elasticsearch는 클러스터 전체에 분산된 샤드를 통해 인덱스 데이터를 복제하여 소프트웨어 계층에서 데이터 충실도, 가용성 및 재해 복구 요구 사항을 관리합니다.

성능

분산된 Elastic 클러스터 환경에서 검색 및 인덱스 성능을 지원하기 위한 시스템 리소스 및 대역폭 계획은 인덱싱되는 총 데이터 볼륨과 활성 동시 검색 수(예약되었거나 기타의 경우 포함)를 고려해야 합니다.

다음 단계에 따라 적절한 크기를 산정합니다.

1. 해당되는 각 계층(핫, 웜, 프로즌)에 대해 다음 중 가장 큰 크기를 결정합니다.
   a. 데이터 볼륨
   b. 샤드 볼륨
   c. 인덱싱 처리량
   d. 검색 처리량
2. 각 계층의 크기 결합
3. 마스터, 코디네이터, 인제스트, ML 등 모든 전용 노드에서 결정을 내립니다.

부록 B: 솔루션 검증 및 테스트

플래시블레이드에 기반한 Elasticsearch 솔루션은 Elasticsearch의 세 가지 주요 기능인 데이터 인제스트, 검색 및 클러스터 작업을 테스트하여 검증되었습니다.

- 데이터 인제스트
- 검색 동작
- 운영 효율성

데이터 인제스트

Elasticsearch는 데이터 인제스트를 위한 몇 가지 옵션을 제공하지만, Bulk Index API, Logstash 및 Beats가 보편적으로 사용됩니다. 아래 다이어그램은 인덱스 작업 중 데이터 처리 플로우를 보여 줍니다.

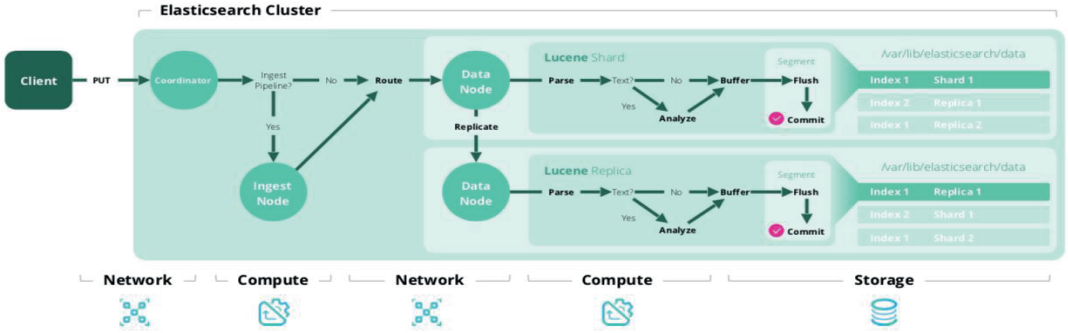

그림 14: Elasticsearch의 데이터 처리 플로우

인제스트 테스트 개요:

정형 데이터와 비정형 데이터를 모두 인제스트할 때 다음 두 가지 테스트를 수행하여 최대 성능을 확인했습니다.

- Elastic 랠리 벤치마크 툴을 사용하여 대규모 데이터 세트를 인제스트하고, 내장된 데이터 세트의 복제본을 만들어 총 용량을 늘렸습니다. 주로 NYC 택시 데이터세트를 사용했습니다.
- 자체 개발한 툴을 사용해 16TB의 원시 데이터로 구성된 Apache 로그를 생성했습니다.

인제스트 테스트 설정:

인덱싱 설정은 Elastic 랠리 벤치마크 툴을 기반으로 했습니다. 5개의 로드 생성기와 1개의 코디네이터를 구성했습니다. 인덱싱을 위한 소스 데이터가 다른 플래시블레이드에 마운트된 드라이브에 저장되었습니다.

소스 데이터세트:

정형 데이터세트:
NYC 택시(ES 랠리 툴의 일부) 원시
원시 데이터: 600GB(8배로 확장)
문서 문서 수: 13억개

비정형 데이터세트:
Apache 로그 – 내부 툴을 사용해 사용자 정의로 생성
원시 데이터: 16TB
문서 수: 750억개

Linux 노드의 시스템 마운트 구성:

/etc/fstab을 편집하여 마운트 구성 추가

```
10.21.152.245:/esdata /mnt/esdatavip nfs rw,bg,nointr,hard,tcp,vers=3 0 0
```

확인하려면:

```
$df -h /mnt/esdatavip
Filesystem Size Used Avail Use% Mounted on 10.21.152.245:/esdata 80T 47T 34T 58%
/mnt/esdatavip1
```

플래시블레이드 스토리지를 위한 Elasticsearch 구성:

Elastic 구성 파일 $ES_HOME/config/elasticsearch.yml 편집

```
cluster.name: cks-es-40n
node.name: ${HOSTNAME}-0
path.data: /mnt/esdatavip/${node.name}/n1
path.logs: /var/log/elasticsearch/${node.name}
bootstrap.memory_lock: true
node.master: true
node.data: true
```

인제스트 테스트 결과

대량 크기(Bulk size) 확장 테스트

대량 삽입(Bulk inserting)은 단일 요청 또는 API 호출에서 여러 문서를 Elasticsearch에 추가하는 방법입니다. 이 방법을 사용하면 개별 문서를 위한 연결을 열고 닫을 필요가 없어 작업 속도가 향상됩니다. 대량 규모는 데이터, 분석 및 클러스터 구성에 따라 결정되지만, 적절한 시작점은 1K입니다. 다양한 대량 크기에 대한 인제스트 성능을 측정하기 위한 일련의 테스트에서, 해당 데이터 세트의 **양호한 대량 크기는 100K**였습니다.

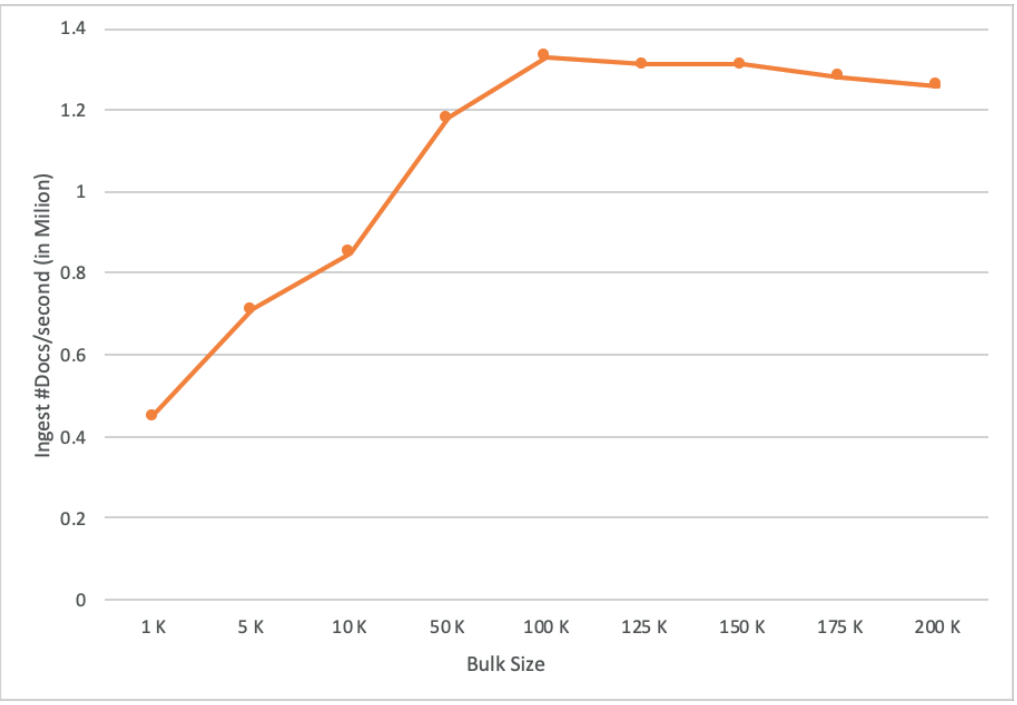

그림 15: 플래시블레이드를 사용한 대량 문서 삽입 성능

샤드 확장

샤딩(sharding)을 사용하면 데이터를 노드 클러스터 전체에 분산할 수 있는 더 작은 청크로 분할하여 스케일 아웃할 수 있습니다. 복제본 샤드는 데이터 손실을 방지하는 데 사용되는 샤드의 복제본입니다. 처음부터 수평으로 **확장할 수 있도록** 설계되었기 때문에, 클러스터에 노드를 더 추가하면 클러스터의 용량을 늘릴 수 있습니다. 이 테스트에서는 대량 문서의 크기를 100K로 구성하고 40개의 인덱스 노드를 사용했습니다. 그림 16은 특정 데이터 세트에서 **노드당 80~100개의 샤드**를 사용할 때 인제스트 성능이 최적이었음을 보여줍니다.

그림 16: 인제스트 성능과 플래시블레이드의 샤드 수 비교

대량 인덱스 클라이언트 확장

대량 인덱스 클라이언트는 대량 인덱싱을 위한 동시 스레드/연결 수입니다. 클라이언트 10개로 시작하면 적절합니다. 여러 테스트를 수행하며 대용량 인덱스 클라이언트를 늘려본 결과, **80~100개의 대량 인덱스 클라이언트**가 40개의 샤드/인덱스로 구성된 특정 데이터세트에 대해 우수한 성능을 보여 주는 것으로 확인되었습니다.

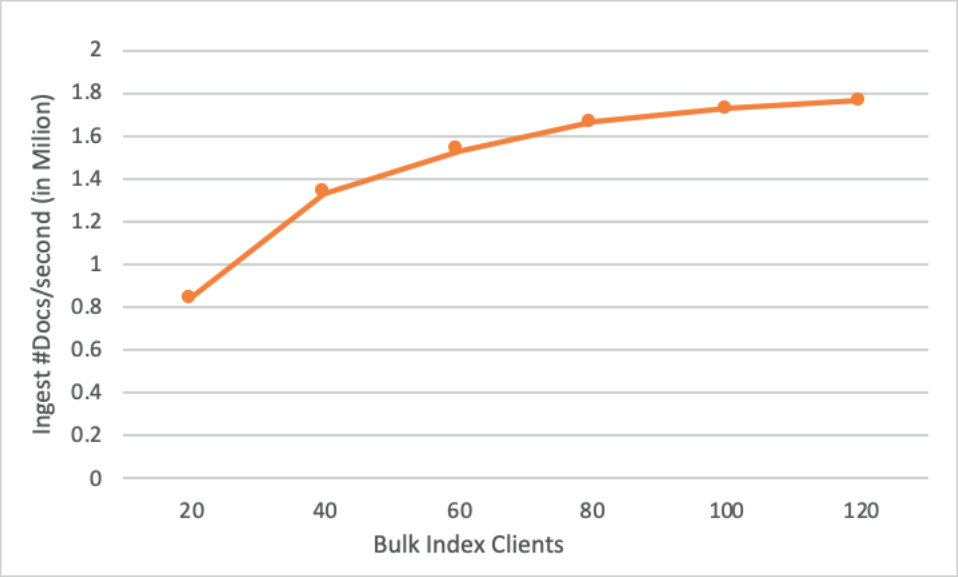

그림 17: 인제스트 성능과 플래시블레이드의 인덱스 클라이언트 수 비교

인덱스 벤치마크 중 플래시블레이드 활용률:

모든 인덱스 벤치마크에서 플래시블레이드는 약 2~3Gb/초의 쓰기 성능을 달성했습니다. 그림 18은 모든 테스트에서 캡처된 플래시블레이드 성능 메트릭입니다.

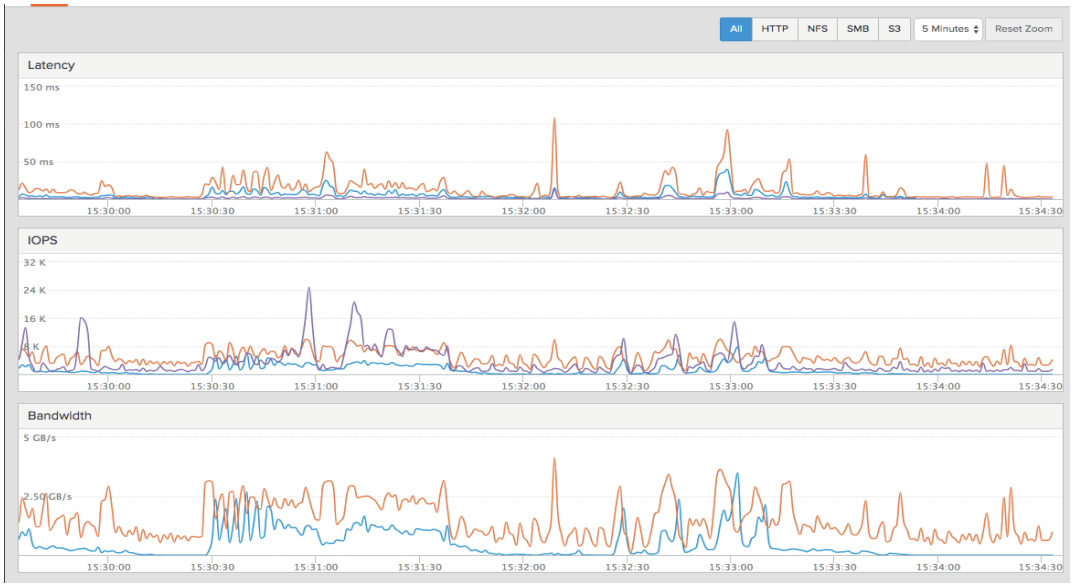

그림 18: 인덱싱 벤치마크를 위한 플래시블레이드 성능 메트릭

DAS SSD와의 비교

플래시블레이드를 로컬 SSD(DAS)를 사용하는 기존 아키텍처와 비교했습니다. 두 구성 모두 선형적으로 확장되지만, 플래시블레이드 구성은 DAS보다 인제스트 성능이 약간 더 높습니다. 40노드 구성에서는 15개 블레이드가 포함된 플래시블레이드 시스템이 성능 제한지점까지 정기적으로 푸시되므로 이 범위를 초과하면 추가 블레이드가 필요합니다.

DAS SSD를 사용하여 동일한 40개 노드에서 테스트한 결과, 플래시블레이드와 동일하거나 약간 더 우수한 성능이 확인되었습니다.

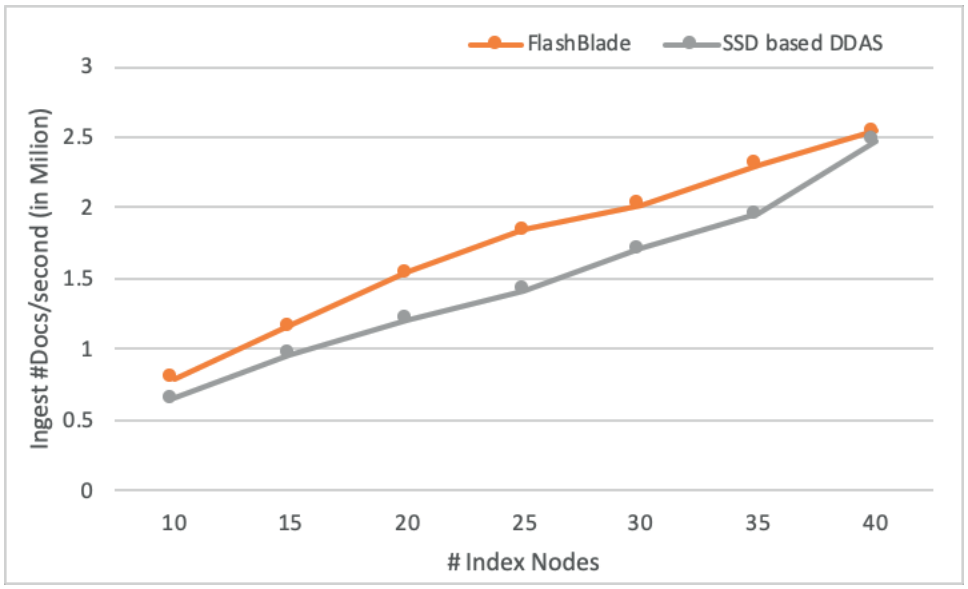

그림 19: 플래시블레이드와 SSD DDAS를 사용한 노드 확장 비교

검색 동작

Elasticsearch는 데이터 인제스트를 위한 몇 가지 옵션을 제공하지만, Bulk Index API, Logstash 및 Beats가 보편적으로 사용됩니다. 아래 다이어그램은 인덱스 작업 중 데이터 처리 플로우를 보여 줍니다.

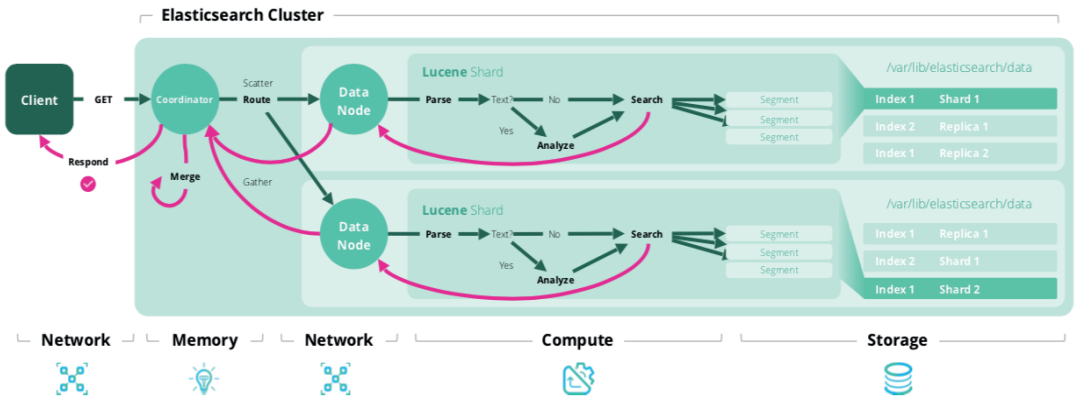

그림 20: Elasticsearch 검색 처리 플로우

검색 테스트 개요:

검색 설정은 동일한 40개 노드 Elastic 클러스터를 기반으로 합니다. 검색 쿼리에 사용자 지정 생성된 16TB의 Apache 로그 데이터를 사용하고 인제스트 테스트의 일부로 동일한 인덱스 데이터를 사용했습니다.

소스 데이터세트:

비정형 데이터세트:
Apache 로그 – 내부 툴을 사용해 사용자 정의로 생성
원시 데이터: 16TB
문서 수: 750억개

동시 검색 테스트

Apache Bench는 동시 세션 수를 설정할 수 있는 기능과 함께 간단한 http 기반 검색 요청 테스트를 제공했습니다. 이러한 쿼리는 동일하지만 기본적인 동시 검색을 테스트하는 데 적합한 것으로 고려됐습니다.

Apache JMeter는 동시 검색 결과를 제공할 뿐 아니라, 가변 쿼리 데이터로 테스트할 수 있는 방법을 제공했습니다. 또한 결과는 Kibana에서 쿼리 및 분석을 할 수 있도록 Elasticsearch의 다른 인스턴스에 입력되었습니다.

검색 테스트 결과:
인덱싱 여부에 기반해 동시 쿼리 검색 결과를 비교할 때 성능 차이는 미미했습니다. 이 그림은 솔루션의 확장 기능을 보여 줍니다.

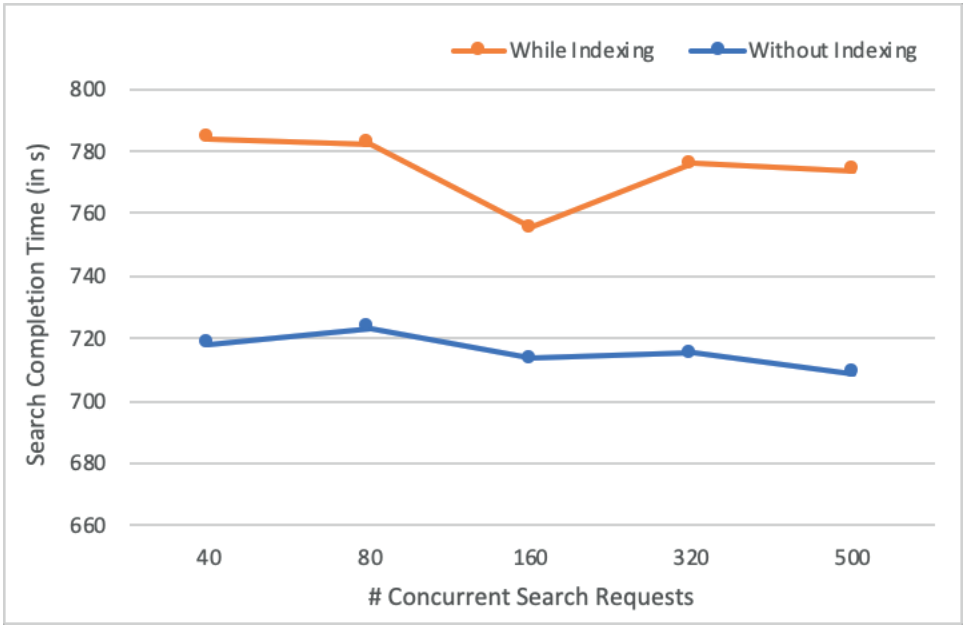

그림 21: 플래시블레이드에서 인덱싱 여부에 따른 검색 성능

핫 쿼리와 콜드 쿼리로 검색:

플래시블레이드가 IO 및 CPU 주기가 필요한 임시 쿼리를 처리할 수 있는지 확인하기 위해 특별한 테스트를 실시했습니다. 무한 루프에서 핫 쿼리를 실행하는 동안, 임시 콜드 쿼리를 도입한 것입니다. 핫 쿼리는 하나의 검색어와 최근 날짜 범위라는 점에서 최소 IO와 연관됐습니다. 이러한 쿼리는 자주 발생하는 쿼리이므로, Elasticsearch는 이러한 쿼리를 클러스터에 지능적으로 캐싱합니다.

몇 분 후 엔터프라이즈 분석 검색 시나리오를 시뮬레이션하기 위해 기존 데이터 범위에 대해 소량의 임시 쿼리 배치를 한 개 및 두 개 검색 용어의 조합으로 실행했습니다. 이러한 쿼리는 더 오래 걸리고 더 많은 IO 및 CPU 주기를 소비할 것으로 예상됩니다. 아래 차트는 랜덤 임시 콜드 쿼리(4세트)가 일정한 간격으로 실행될 때 무한 루프에서 실행되는 동시 핫 쿼리의 영향을 보여줍니다. 쿼리 성능에는 거의 변화가 없습니다. 이미지 하단은 플래시블레이드 대시보드의 성능 메트릭을 보여줍니다.

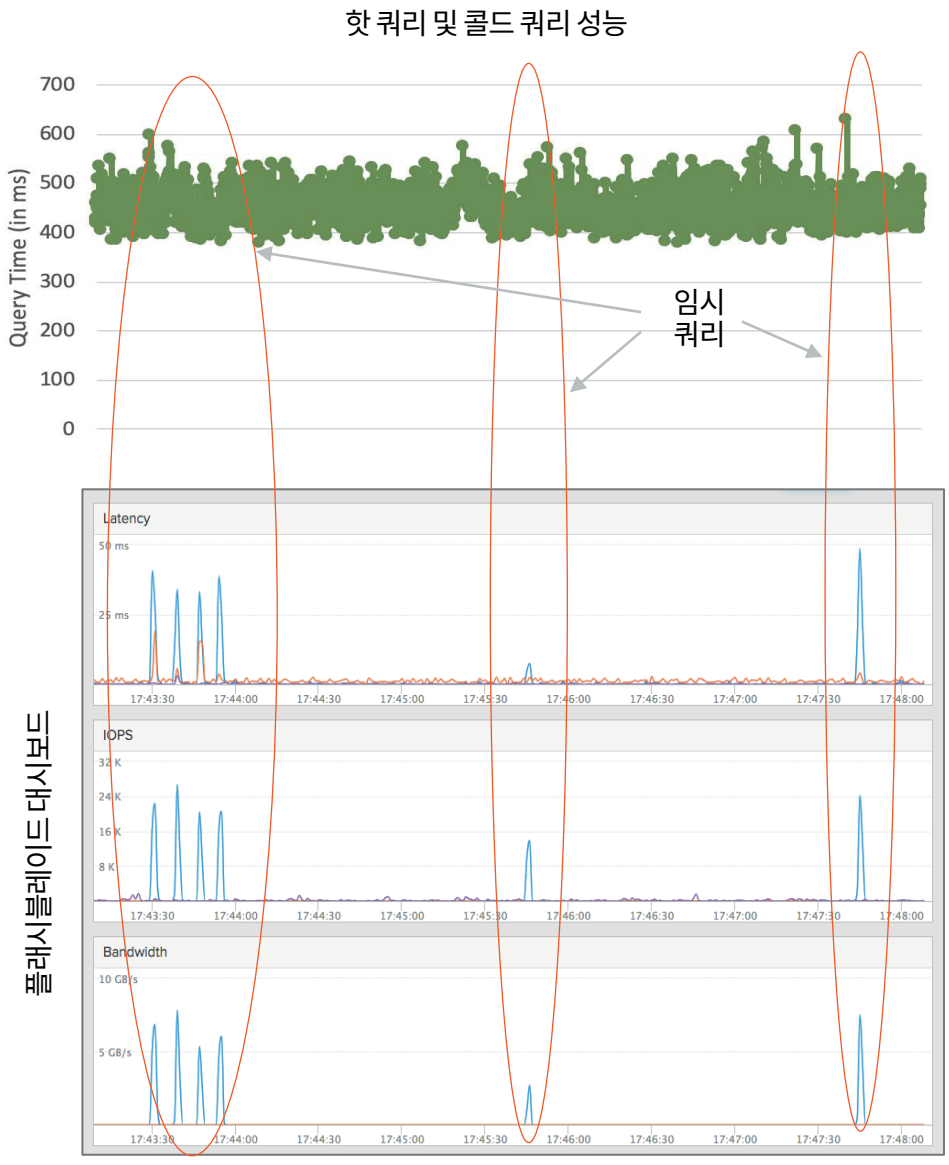

그림 22: 플래시블레이드를 사용한 핫 쿼리 및 콜드 쿼리 성능

운영 효율성

스토리지와 컴퓨팅을 분리하는 것이 관리를 간소화한다는 것을 확인하기 위해 장애 시나리오를 테스트했습니다.

장애 시나리오

- 이 단계에서는 노드 장애 발생 시 Elastic 클러스터의 성능을 테스트합니다:
  복제본 1개를 사용하여 NYC 택시 데이터세트와 동일한 데이터 세트를 사용했습니다. 장애 시나리오를 시뮬레이션하기 위해 Elastic 노드 중 하나의 전원을 껐습니다. Kibana 대시보드는 즉시 클러스터의 상태를 빨간색으로 표시했습니다. 하지만 몇 분 안에 클러스터가 녹색으로 다시 바뀌며 복제본 샤드가 다른 노드의 기본 샤드로 다시 생성되었습니다.
- 복제본 샤드가 없는 경우의 노드 장애:
  이 시나리오에서는 복제본 샤드로 인덱싱되지 않은 Apache 로그 데이터세트를 사용했습니다. 따라서 노드 장애 시나리오를 다시 만들 때, 해당 샤드의 데이터가 손실된 것처럼 보였습니다. 그러나 노드를 다시 백업하면, 마운트된 NAS 폴더에서 데이터 손실 없이 자동으로 데이터를 가져옵니다.

스토리지 공간 부족 시나리오

- 라이브 클러스터에서의 스토리지 확장성
  이 시나리오에서는 플래시블레이드에서 3%만 남아 있는 할당된 스토리지를 사용했습니다. Elasticsearch의 저장 공간이 부족하기 때문에 Elastic의 상태는 처음에는 노란색, 그 다음에는 빨간색으로 나타납니다. 플래시블레이드 구성 화면에서 할당을 변경하여 몇 분 내에 이 문제를 해결할 수 있었습니다. Linux 노드 및 Elastic 노드에 즉시 반영되었으며, 영향을 받은 인덱스는 녹색으로 표시됩니다.

부록 C: 모범 사례 및 추가 리소스

모범 사례

퓨어스토리지 플래시블레이드

플래시블레이드 파일 시스템

- 플래시블레이드에서 단일 NFS 파일 시스템을 생성하고 각 데이터 노드에 하나의 고유한 하위 폴더를 마운트 지점으로 사용합니다. 이를 통해 플래시블레이드에서 파일 시스템별로 공간과 성능을 추적할 수 있습니다
- 플래시블레이드 파일 시스템은 항상 씬 프로비저닝되므로, Elastic 관리자는 데이터 증가를 충족하기 위해 사이즈를 업데이트하지 않아도 되도록 대규모 파일 시스템을 프로비저닝할 수 있습니다.
- 파일 시스템 크기에 대한 고정된 제한값 매개 변수를 설정하지 않는 것이 좋습니다. 그러면 필요할 때 공간을 더 추가할 수 있는 유연성이 제한되기 때문입니다.
- 클러스터의 모든 인덱서에 대한 모든 NFS 파일 시스템을 동일한 크기로 유지합니다.

Linux 마운트 옵션

다음 마운트 옵션을 사용하여 데이터 노드의 인덱서 노드에 NFS 파일 시스템을 마운트합니다.
`rw,bg,nointr,hard,tcp,vers=3`

호스트가 플래시블레이드에서 제공하는 기본 크기(512K)를 가져올 수 있으므로 wsize 옵션을 지정하지 마세요.

리눅스 노드 튜닝

Elastic 노드 튜닝:

1. 메모리 맵 수(count) 설정
cli에서 일시적으로 변경하는 경우: $sysctl -w vm.max_map_count=262144
/etc/sysctl.conf add vm.max_map_count=262144를 편집합니다.
확인하려면:
$sysctl vm.max_map_count를 체크합니다.

2. 스와핑 비활성
`$sudo swapoff -a`
==> `/etc/fstab`에서 스왑을 사용해 모든 행을 주석으로 처리합니다
`$sysctl vm.swappiness=1`

3. Memorylock 설정
Elasticsearch를 시작하기 전에 `ulimit -l unlimited`를 `root`로 설정하거나,
`/etc/security/limits.conf`에서 `memlock`을 `unlimited`로 설정합니다.
$ulimit -l unlimited

```
$ES_HOME/config/elasticsearch.yml file:
bootstrap.memory_lock: true
```

확인하려면:

```
curl -X GET “hostname:9200/_nodes?filter_path=**.mlockall”
```

4. 파일 디스크립터

```
$ulimit -n 65535
```

또는 `/etc/security/limits.conf`에서 nodfile을 65535로 설정합니다.

확인하려면:

```
curl -X GET "10.21.236.35:9200/_nodes/stats/process?filter_path=**.max_file_descriptors"
```

5. nfs 툴 설치

각 노드에 nfs툴을 설치합니다.

```
yum install -y nfs-utils
systemctl enable nfs
systemctl restart nfs
```

복제본 노드 - 리밸런싱:

노드에 장애가 발생하면 리밸런싱 작업이 자동으로 시작되지만 설정을 통해 중지할 수 있습니다

```
Cluster.routing.enable = none
```

장애가 발생한 노드와 연결된 데이터를 플래시블레이드에서 계속 사용할 수 있기 때문에 이 작업이 필요합니다. 테스트 중에 노드를 정상 상태로 복구하면, 데이터를 리밸런싱하지 않고 사용할 수 있었습니다.

복제본 노드 - 리밸런싱:

- **튜닝 가능:** 데이터세트에 이상적인 매개변수(예: 대량 크기, 샤드 수 및 대량 인덱싱 클라이언트)를 결정합니다.
- **새로 고침 간격:** 새로 고침 간격을 1~30초에서 변경하는 것이 좋습니다.
- **사전 인덱싱:** 범위 집계의 경우, 필요한 데이터를 사전 인덱싱하여 오버헤드를 줄입니다.
- **스레드 풀 설정:** 대량 인덱싱에 대해 스레드 풀 속성을 사용합니다

추가 자료

중요 Elasticsearch 구성

https://www.elastic.co/guide/en/elasticsearch/reference/current/important-settings.html

인덱싱 속도 조정

https://www.elastic.co/guide/en/elasticsearch/reference/master/tune-for-indexing-speed.html

검색 속도 조정

https://www.elastic.co/guide/en/elasticsearch/reference/master/tune-for-search-speed.html

랠리 벤치마크 툴

https://esrally.readthedocs.io/en/stable/

부록 D: esrally 벤치마크 툴 설치 방법

Rally는 Elasticsearch를 벤치마크할 수 있도록 Elastic이 제공하는 공용 툴입니다. 이 설정의 목표는 esrally를 실행하여 기존 클러스터를 벤치마크하는 것입니다.

esrally를 위해 노드 몇 개의 설정을 계획합니다. 일례로 5개의 노드를 설정할 수 있습니다. 노드 10.5.5.5 - 10.5.5.10 10.5.5.11 – 10.5.5.15에서 실행되는 Elastic 노드가 이미 있습니다. 이는 벤치마크 후보 시스템입니다.

첫 번째 시스템에서는 랠리를 사용자 Elastic으로 설치하고, 이를 벤치마크 코디네이터라고 부릅니다.

esrally daemon을 시작합니다.

```
$esrally start –node-ip=10.5.5.5 –coordinator-ip=10.5.5.5
```

다른 노드는 랠리 드라이버 노드입니다. 다른 모든 노드에 사용자 Elastic으로 랠리를 설치합니다.

이러한 각 노드에서 esrally daemon을 시작합니다.

```
$esrally start –node-ip=10.5.5.6 –coordinator-ip=10.5.5.5
$esrally start –node-ip=10.5.5.10 –coordinator-ip=10.5.5.5
```

벤치마크를 실행하려면:

```
esrally --track=nyc_taxis --target-hosts=10.5.5.11:9200,10.5.5.12:9200,10.5.5.13:9200,1
0.5.5.14:9200,10.5.5.15:9200 --pipeline=benchmark-only
```

Esrally daemon은 명령을 사용하여 중지할 수 있습니다.

```
$esrallyd stop
```

부록 E: 용량 및 규모 산정

모든 Elastic 스택의 운영 구현은 Elasticsearch의 용량 계획에 기반해 이뤄져야 합니다. 로그, 메트릭, 트레이스 또는 검색에 사용하는 것이든, 직접 또는 클라우드에서 실행하는 것이든, 구현 상태와 성능을 보장하려면, Elasticsearch의 인프라 및 구성을 계획해야 합니다.

용량 계획은 Elasticsearch 구현을 운영하는 데 필요한 리소스의 유형과 양을 추정하는 것입니다. 주요 고려 사항은 다음과 같습니다.

- 기본 컴퓨팅 리소스
- Elasticsearch의 아키텍처, 동작 및 리소스 요구 사항
- Elasticsearch 구현의 요구 사항을 예측하는 방법

네 가지 기본 컴퓨팅 리소스는 데이터가 유지되는 스토리지, 데이터가 버퍼링되는 메모리, 데이터가 처리되는 컴퓨팅, 데이터가 전송되는 네트워크입니다.

고성능 NAS(플래시블레이드)는 스토리지 요구사항을 충족합니다.

JVM 힙은 클러스터, 인덱스, 샤드, 세그먼트 및 파일 데이터의 메타데이터를 저장합니다. 가비지 수집을 방지하려면 사용 가능한 RAM의 최대 50%(최대 31GB RAM)을 사용하는 것이 좋습니다. Elasticsearch는 사용 가능한 나머지 메모리를 사용하여 데이터를 캐시하며, 전체 텍스트 검색, 문서 값 집계 및 정렬 시 디스크 읽기를 방지함으로써 성능을 크게 향상시킵니다.

Elasticsearch 노드에는 사용 가능한 컴퓨팅 리소스를 활용하는 **스레드 풀**과 **스레드 대기열(thread queues)**이 있습니다. CPU 코어의 양과 성능은 Elasticsearch의 데이터 작업에 대한 평균 속도와 최대 **처리량**을 좌우합니다.

대규모 구현의 경우, 노드 간 인제스트, 검색 또는 복제를 위한 데이터 전송량이 **네트워크 포화**를 유발할 수 있습니다. 이러한 경우 네트워크 연결을 더 빠른 속도로 업그레이드하거나, Elasticsearch 구현을 둘 이상의 클러스터로 분할한 다음 **Cross-Cluster Search(CCS)**를 사용하여 단일 논리 유닛으로 검색할 수 있습니다.

Elasticsearch의 주요 활용 사례에 적용될 수 있는 두 가지 기본 규모 산정 방법이 있습니다.

- 볼륨: 클러스터의 각 계층에 대해 예상되는 데이터 및 샤드 양을 저장하는 데 필요한 스토리지 및 메모리 리소스를 추정합니다.
- 처리량: 각 작업과 클러스터의 각 계층에 대해 예상 레이턴시와 처리량으로 예상 작업을 처리하는 데 필요한 메모리, 컴퓨팅 및 네트워크 리소스를 추정합니다.

볼륨 규모 산정: 데이터 볼륨

파악을 위한 질문:

- 하루에 인덱싱할 원시 데이터 양(GB)은 얼마입니까?
- 데이터를 며칠 동안 보관할 예정입니까?
- 몇 개의 복제 샤드를 시행하려고 합니까?
- 데이터 노드당 얼마나 많은 메모리를 할당할 예정입니까?

또한 장애를 처리할 데이터 노드에 해당되는 용량을 비축해는 것이 좋습니다.

총 데이터(GB) = 일별 원시 데이터(GB) * 보존 기간(일) * 순 확장 계수 * (복제본 수 + 1)

총 스토리지(GB) = 총 데이터(GB) * (1 + 0.15 디스크 워터마크 임계값 + 0.05 오차 한계)

총 데이터 노드 수 =ROUNDUP(총 스토리지(GB)/데이터 노드당 메모리/메모리:데이터 비율) + 페일오버 용량에 대한 데이터 노드 1개

압축:

Elasticsearch는 디스크에 기록하기 전에 항상 데이터를 압축합니다. 압축에는 두 가지 모드가 있습니다. 기본 옵션은 성능에 최소한의 영향을 미치며, *best_compression* 은 더 높은 효율성을 제공하지만 CPU 주기를 많이 소모하고 인덱싱 성능에 영향을 줄 수 있습니다. 스토리지를 최적화하려면 *best_compression* 활성화를 권장합니다.

볼륨 규모 산정: 샤드 볼륨

파악을 위한 질문:

- 인덱스 패턴을 몇 개 생성할 예정입니까?
- 몇 개의 기본 및 복제본 샤드를 구성할 예정입니까?
- 순환한다면, 어떤 시간 간격으로 인덱스를 순환할 예정입니까?
- 인덱스는 얼마나 오래 유지할 예정입니까?
- 데이터 노드당 얼마나 많은 메모리를 할당할 예정입니까?

일반적으로 샤드당 40~50GB를 초과하지 않는 것이 좋습니다.

팁: 소규모 일별 인텍스를 주간 또는 월간 인텍스로 통합하여 샤드 수를 줄입니다. 40GB 이상의 큰 일별 인덱스를 시간별 인덱스로 분할하거나 기본 샤드 수를 늘립니다.

총 샤드 수 = 인덱스 패턴 수 * 기본 샤드 수 * (복제본 샤드 수 + 1) * 총 보존 간격

총 데이터 노드 수 = ROUNDUP(총 샤드 수/(데이터 노드당 20 * 메모리))

처리량 규모 산정: 검색 작업

검색 활용 사례에는 스토리지 용량 외에도 검색 응답 시간과 검색 처리량에 대한 타깃이 있습니다. 이러한 타깃에는 더 많은 메모리와 컴퓨팅 리소스가 필요할 수 있습니다.

변수가 너무 많으면 검색 응답 시간에 영향을 주어 특정 용량 계획이 미치는 영향을 예측하기 어렵습니다. 검색 응답 시간을 경험적으로 테스트하고 예상 검색 처리량을 추정함으로써, 클러스터에 필요한 리소스를 예측하여 요구 사항을 충족할 수 있습니다.

파악을 위한 질문:

- 초당 최대 검색 수는 몇 회입니까?
- 평균 검색 응답 시간(밀리초)은 얼마입니까?
- 데이터 노드에 코어 및 코어당 스레드 수가 몇 개입니까?

접근 방법 이론

리소스가 검색 속도에 미치는 영향을 파악하는 대신, 계획된 하드웨어에서 검색 속도를 측정하여 검색 속도를 상수로 사용하세요. 그런 다음 예상되는 최대 검색량을 처리하기 위해 클러스터에 필요한 코어 수를 결정합니다. 궁극적으로는 스레드 풀 대기열(thread pool queue)이 소비되는 것보다 더 빠르게 증가하는 것을 방지하는 것이 목표입니다. 컴퓨팅 리소스가 부족하면 검색 요청이 누락될 위험이 있습니다.

최대 스레드 크기 총 데이터 노드 수 = ROUNDUP(초당 최대 검색 수 * 평균 검색 응답 시간(밀리초)/1000밀리초)

스레드 풀 = ROUNDUP((노드당 물리적 코어 * 코어당 스레드 * 3/2) + 1)

총 데이터 노드 수 = ROUNDUP(최대 스레드/스레드 풀 크기)

Pure Storage, Inc.
650 Castro Street, #400
Mountain View, CA 94041

PS1023-01