NetApp 사용한 벡터 데이터베이스 솔루션

NetApp artificial intelligence solutions

NetApp
August 18, 2025

This PDF was generated from https://docs.netapp.com/ko-kr/netapp-solutions-ai/vector-db/ai-vdb-solution-with-netapp.html on August 18, 2025. Always check docs.netapp.com for the latest.

목차

NetApp 사용한 벡터 데이터베이스 솔루션 . . . 1
- NetApp 사용한 벡터 데이터베이스 솔루션 . . . 1
- 소개 . . . 2
  - 소개 . . . 2
- 솔루션 개요 . . . 2
  - 솔루션 개요 . . . 2
- 벡터 데이터베이스 . . . 3
  - 벡터 데이터베이스 . . . 3
- 기술 요구 사항 . . . 6
  - 기술 요구 사항 . . . 6
  - 하드웨어 요구 사항 . . . 6
  - 소프트웨어 요구 사항 . . . 7
- 배포 절차 . . . 7
  - 배포 절차 . . . 7
- 솔루션 검증 . . . 9
  - 솔루션 개요 . . . 9
  - 온프레미스에서 Kubernetes를 사용한 Milvus 클러스터 설정 . . . 10
  - NetApp ONTAP 용 Amazon FSx ONTAP 탑재한 Milvus - 파일 및 객체 이중성 . . . 17
  - SnapCenter 사용한 벡터 데이터베이스 보호 . . . 23
  - NetApp SnapMirror 사용한 재해 복구 . . . 34
  - 벡터 데이터베이스 성능 검증 . . . 36
- PostgreSQL을 사용하는 Instaclustr의 벡터 데이터베이스: pgvector . . . 44
  - PostgreSQL을 사용하는 Instaclustr의 벡터 데이터베이스: pgvector . . . 44
- 벡터 데이터베이스 사용 사례 . . . 44
  - 벡터 데이터베이스 사용 사례 . . . 44
- 결론 . . . 46
  - 결론 . . . 47
- 부록 A: Values.yaml . . . 48
  - 부록 A: Values.yaml . . . 48
- 부록 B: prepare_data_netapp_new.py . . . 69
  - 부록 B: prepare_data_netapp_new.py . . . 69
- 부록 C: verify_data_netapp.py . . . 72
  - 부록 C: verify_data_netapp.py . . . 72
- 부록 D: docker-compose.yml . . . 76
  - 부록 D: docker-compose.yml . . . 76

NetApp 사용한 벡터 데이터베이스 솔루션

NetApp 사용한 벡터 데이터베이스 솔루션

Karthikeyan Nagalingam 및 Rodrigo Nascimento, NetApp

이 문서에서는 NetApp의 스토리지 솔루션을 사용하여 Milvus와 오픈소스 PostgreSQL 확장 프로그램인 pgvecto와 같은 벡터 데이터베이스를 구축하고 관리하는 방법에 대해 자세히 살펴봅니다. NetApp ONTAP 및 StorageGRID 객체 스토리지를 사용하기 위한 인프라 가이드라인을 자세히 설명하고 AWS FSx ONTAP 에서 Milvus 데이터베이스의 적용을 검증합니다. 이 문서에서는 NetApp의 파일-객체 이중성과 벡터 임베딩을 지원하는 벡터 데이터베이스 및 애플리케이션에 대한 유용성을 설명합니다. 이는 NetApp의 엔터프라이즈 관리 제품인 SnapCenter 의 역량을 강조하여 벡터 데이터베이스에 대한 백업 및 복원 기능을 제공하고 데이터 무결성과 가용성을 보장합니다. 이 문서에서는 NetApp의 하이브리드 클라우드 솔루션을 자세히 살펴보고, 온프레미스와 클라우드 환경에서의 데이터 복제 및 보호에 있어서 이 솔루션의 역할을 설명합니다. 여기에는 NetApp ONTAP 에서 벡터 데이터베이스의 성능 검증에 대한 통찰력이 포함되며, 생성 AI에 대한 두 가지 실제 사용 사례인 LLM을 갖춘 RAG와 NetApp의 내부 ChatAI에 대한 내용으로 마무리됩니다. 이 문서는 벡터 데이터베이스를 관리하기 위해 NetApp의 스토리지 솔루션을 활용하는 방법에 대한 포괄적인 가이드입니다.

참조 아키텍처는 다음 사항에 초점을 맞춥니다.

1. "소개"
2. "솔루션 개요"
3. "벡터 데이터베이스"
4. "기술 요구 사항"
5. "배포 절차"
6. "솔루션 검증 개요"
    - "온프레미스에서 Kubernetes를 사용한 Milvus 클러스터 설정"
    - 링크:vector-database-milvus-with-Amazon-FSx ONTAP-for- NetApp- ONTAP .html[NetApp ONTAP 용 Amazon FSx ONTAP 과 함께하는 Milvus - 파일 및 객체 이중성]
    - "NetApp SnapCenter 사용한 벡터 데이터베이스 보호."
    - "NetApp SnapMirror 사용한 재해 복구"
    - "성능 검증"
7. "PostgreSQL을 사용하는 Instaclustr의 벡터 데이터베이스: pgvector"
8. "벡터 데이터베이스 사용 사례"
9. "결론"
10. "부록 A: values.yaml"
11. "부록 B: prepare_data_netapp_new.py"
12. "부록 C: verify_data_netapp.py"

13. "부록 D: docker-compose.yml"

소개

이 섹션에서는 NetApp 용 벡터 데이터베이스 솔루션을 소개합니다.

소개

벡터 데이터베이스는 대규모 언어 모델(LLM)과 생성형 인공 지능(AI)에서 의미 검색의 복잡성을 처리하도록 설계된 과제를 효과적으로 해결합니다. 기존 데이터 관리 시스템과 달리 벡터 데이터베이스는 레이블이나 태그가 아닌 데이터 자체의 내용을 사용하여 이미지, 비디오, 텍스트, 오디오 및 기타 형태의 비정형 데이터를 포함한 다양한 유형의 데이터를 처리하고 검색할 수 있습니다.

관계형 데이터베이스 관리 시스템(RDBMS)의 한계는 잘 알려져 있으며, 특히 AI 애플리케이션에서 흔히 볼 수 있는 고차원 데이터 표현과 비정형 데이터 처리의 어려움에 대한 내용이 많습니다. RDBMS는 데이터를 관리하기 쉬운 구조로 평면화하는 데 시간이 많이 걸리고 오류가 발생하기 쉬운 프로세스를 필요로 하며, 이로 인해 검색이 지연되고 비효율적이 됩니다. 그러나 벡터 데이터베이스는 이러한 문제를 해결하도록 설계되어 복잡하고 고차원 데이터를 관리하고 검색하기 위한 보다 효율적이고 정확한 솔루션을 제공함으로써 AI 애플리케이션의 발전을 촉진합니다.

이 문서는 현재 벡터 데이터베이스를 사용 중이거나 사용할 계획인 고객을 위한 포괄적인 가이드로, NetApp ONTAP, NetApp StorageGRID, Amazon FSx ONTAP for NetApp ONTAP, SnapCenter 등의 플랫폼에서 벡터 데이터베이스를 활용하는 모범 사례를 자세히 설명합니다. 여기에 제공된 내용은 다음과 같은 다양한 주제를 다룹니다.

- NetApp ONTAP 및 StorageGRID 개체 스토리지를 통해 NetApp 스토리지에서 제공하는 Milvus와 같은 벡터 데이터베이스에 대한 인프라 가이드라인입니다.
- 파일 및 객체 저장소를 통해 AWS FSx ONTAP 에서 Milvus 데이터베이스를 검증합니다.
- NetApp의 파일-객체 이중성을 탐구하여 벡터 데이터베이스와 기타 애플리케이션의 데이터에 대한 유용성을 보여줍니다.
- NetApp의 데이터 보호 관리 제품인 SnapCenter 는 벡터 데이터베이스 데이터에 대한 백업 및 복원 기능을 제공합니다.
- NetApp의 하이브리드 클라우드가 온프레미스 및 클라우드 환경 전반에서 데이터 복제 및 보호를 제공하는 방식입니다.
- NetApp ONTAP 에서 Milvus 및 pgvector와 같은 벡터 데이터베이스의 성능 검증에 대한 통찰력을 제공합니다.
- 두 가지 구체적인 사용 사례: 대규모 언어 모델(LLM)을 사용한 검색 증강 생성(RAG)과 NetApp IT 팀의 ChatAI를 통해 설명된 개념과 관행에 대한 실제적 예를 제공합니다.

솔루션 개요

이 섹션에서는 NetApp 벡터 데이터베이스 솔루션에 대한 개요를 제공합니다.

솔루션 개요

이 솔루션은 NetApp 이 벡터 데이터베이스 고객이 겪는 과제를 해결하기 위해 제공하는 독특한 이점과 기능을 보여줍니다. NetApp ONTAP, StorageGRID, NetApp의 클라우드 솔루션 및 SnapCenter 활용함으로써 고객은 비즈니스 운영에 상당한 가치를 더할 수 있습니다. 이러한 도구는 기존 문제를 해결할 뿐만 아니라 효율성과 생산성을 높여 전반적인 비즈니스 성장에 기여합니다.

왜 **NetApp** 인가**?**

- ONTAP 및 StorageGRID 와 같은 NetApp의 제품을 사용하면 스토리지와 컴퓨팅을 분리하여 특정 요구 사항에 따라 최적의 리소스 활용이 가능합니다. 이러한 유연성 덕분에 고객은 NetApp 스토리지 솔루션을 사용하여 스토리지를 독립적으로 확장할 수 있습니다.
- NetApp의 스토리지 컨트롤러를 활용하면 고객은 NFS 및 S3 프로토콜을 사용하여 벡터 데이터베이스에 효율적으로 데이터를 제공할 수 있습니다. 이러한 프로토콜은 고객 데이터 저장을 용이하게 하고 벡터 데이터베이스 인덱스를 관리하며, 파일 및 객체 메서드를 통해 액세스하는 여러 데이터 사본의 필요성을 제거합니다.
- NetApp ONTAP AWS, Azure, Google Cloud 등 주요 클라우드 서비스 공급업체에서 NAS 및 개체 스토리지에 대한 기본 지원을 제공합니다. 이러한 광범위한 호환성은 원활한 통합을 보장하여 고객 데이터 이동성, 글로벌 접근성, 재해 복구, 동적 확장성 및 고성능을 구현합니다.
- NetApp의 강력한 데이터 관리 기능을 통해 고객은 자신의 데이터가 잠재적 위험과 위협으로부터 잘 보호된다는 확신을 가질 수 있습니다. NetApp 데이터 보안을 우선시하여 고객에게 귀중한 정보의 안전성과 무결성에 대한 안심을 제공합니다.

벡터 데이터베이스

이 섹션에서는 NetApp AI 솔루션에서 벡터 데이터베이스의 정의와 사용에 대해 다룹니다.

벡터 데이터베이스

벡터 데이터베이스는 머신 러닝 모델의 임베딩을 사용하여 비정형 데이터를 처리, 인덱싱, 검색하도록 설계된 특수한 유형의 데이터베이스입니다. 데이터를 기존의 표 형식으로 구성하는 대신, 벡터 임베딩이라고도 하는 고차원 벡터로 데이터를 정리합니다. 이 독특한 구조 덕분에 데이터베이스는 복잡하고 다차원적인 데이터를 보다 효율적이고 정확하게 처리할 수 있습니다.

벡터 데이터베이스의 주요 기능 중 하나는 생성적 AI를 사용하여 분석을 수행하는 것입니다. 여기에는 데이터베이스가 주어진 입력과 유사한 데이터 포인트를 식별하는 유사성 검색과, 표준에서 크게 벗어나는 데이터 포인트를 발견할 수 있는 이상 감지가 포함됩니다.

더욱이 벡터 데이터베이스는 시간 데이터나 타임스탬프가 있는 데이터를 처리하는 데 적합합니다. 이러한 유형의 데이터는 IT 시스템 내에서 발생한 '무엇'이 언제 발생했는지에 대한 정보를 순서대로, 그리고 다른 모든 이벤트와 비교하여 제공합니다. 시간적 데이터를 처리하고 분석하는 이러한 기능 덕분에 벡터 데이터베이스는 시간 경과에 따른 이벤트를 이해해야 하는 애플리케이션에 특히 유용합니다.

ML 및 **AI**를 위한 벡터 데이터베이스의 장점**:**

- 고차원 검색: 벡터 데이터베이스는 AI 및 ML 애플리케이션에서 자주 생성되는 고차원 데이터를 관리하고 검색하는 데 탁월합니다.
- 확장성: 대량의 데이터를 처리하도록 효율적으로 확장할 수 있어 AI 및 ML 프로젝트의 성장과 확장을 지원합니다.
- 유연성: 벡터 데이터베이스는 높은 수준의 유연성을 제공하여 다양한 데이터 유형과 구조를 수용할 수 있습니다.
- 성능: AI와 ML 작업의 속도와 효율성에 중요한 고성능 데이터 관리 및 검색 기능을 제공합니다.
- 사용자 정의 가능한 인덱싱: 벡터 데이터베이스는 사용자 정의 가능한 인덱싱 옵션을 제공하여 특정 요구 사항에 따라 최적화된 데이터 구성 및 검색이 가능합니다.

벡터 데이터베이스와 사용 사례**.**

이 섹션에서는 다양한 벡터 데이터베이스와 해당 사용 사례에 대한 세부 정보를 제공합니다.

파이스와 스카**NN**

이들은 벡터 검색 영역에서 중요한 도구 역할을 하는 라이브러리입니다. 이러한 라이브러리는 벡터 데이터를 관리하고 검색하는 데 도움이 되는 기능을 제공하므로 데이터 관리의 특수 분야에서 매우 귀중한 리소스입니다.

엘라스틱서치

널리 사용되는 검색 및 분석 엔진이며, 최근 벡터 검색 기능을 통합했습니다. 이 새로운 기능은 기능성을 향상시켜 벡터 데이터를 보다 효과적으로 처리하고 검색할 수 있게 해줍니다.

솔방울

이는 고유한 기능을 갖춘 강력한 벡터 데이터베이스입니다. 인덱싱 기능에서 밀집 벡터와 희소 벡터를 모두 지원하므로 유연성과 적응성이 향상됩니다. 이 솔루션의 주요 장점 중 하나는 기존 검색 방법과 AI 기반 고밀도 벡터 검색을 결합하여 두 가지의 장점을 모두 활용하는 하이브리드 검색 방식을 만드는 능력입니다.

Pinecone은 주로 클라우드 기반으로, 머신 러닝 애플리케이션용으로 설계되었으며 GCP, AWS, Open AI, GPT-3, GPT-3.5, GPT-4, Catgut Plus, Elasticsearch, Haystack 등 다양한 플랫폼과 잘 통합됩니다. Pinecone은 폐쇄형 소스 플랫폼이며 SaaS(Software as a Service) 형태로 제공된다는 점에 유의하는 것이 중요합니다.

고급 기능을 갖춘 Pinecone은 사이버 보안 산업에 특히 적합합니다. 고차원 검색 및 하이브리드 검색 기능을 효과적으로 활용하여 위협을 탐지하고 대응할 수 있습니다.

크로마

4가지 주요 기능을 갖춘 Core-API를 갖춘 벡터 데이터베이스로, 그 중 하나에는 메모리 내 문서 벡터 저장소가 포함됩니다. 또한, Face Transformers 라이브러리를 활용하여 문서를 벡터화하여 기능성과 다양성을 향상시켰습니다. Chroma는 클라우드와 온프레미스 모두에서 작동하도록 설계되었으며, 사용자 요구 사항에 따라 유연성을 제공합니다. 특히 오디오 관련 애플리케이션에 탁월하여 오디오 기반 검색 엔진, 음악 추천 시스템 및 기타 오디오 관련 사용 사례에 매우 적합한 선택입니다.

위비에이트

사용자가 내장 모듈이나 사용자 정의 모듈을 사용하여 콘텐츠를 벡터화할 수 있는 다용도 벡터 데이터베이스로, 특정 요구 사항에 따라 유연성을 제공합니다. 다양한 배포 환경 설정에 맞춰 완전 관리형 솔루션과 자체 호스팅 솔루션을 모두 제공합니다.

Weaviate의 주요 특징 중 하나는 벡터와 객체를 모두 저장할 수 있는 기능으로, 이를 통해 데이터 처리 기능이 향상됩니다. 이는 ERP 시스템의 의미 검색 및 데이터 분류를 포함한 다양한 응용 분야에 널리 사용됩니다. 전자상거래 부문에서는 검색 및 추천 엔진을 구동합니다. Weaviate는 이미지 검색, 이상 감지, 자동 데이터 조화, 사이버 보안 위협 분석에도 사용되어 여러 도메인에 걸친 다재다능함을 보여줍니다.

레디스

Redis는 빠른 인메모리 스토리지로 유명한 고성능 벡터 데이터베이스로, 읽기-쓰기 작업에 대한 낮은 지연 시간을 제공합니다. 따라서 빠른 데이터 액세스가 필요한 추천 시스템, 검색 엔진, 데이터 분석 애플리케이션에 매우 적합합니다.

Redis는 목록, 집합, 정렬된 집합을 포함하여 벡터에 대한 다양한 데이터 구조를 지원합니다. 또한 벡터 간의 거리를 계산하거나 교집합과 합집합을 찾는 등의 벡터 연산도 제공합니다. 이러한 기능은 특히 유사성 검색, 클러스터링, 콘텐츠 기반 추천 시스템에 유용합니다.

확장성과 가용성 측면에서 Redis는 높은 처리량 작업 부하를 처리하는 데 탁월하며 데이터 복제 기능을 제공합니다. 또한 기존의 관계형 데이터베이스(RDBMS)를 포함한 다른 데이터 유형과도 잘 통합됩니다. Redis에는 실시간

업데이트를 위한 게시/구독(Pub/Sub) 기능이 포함되어 있어 실시간 벡터를 관리하는 데 유용합니다. 게다가 Redis는 가볍고 사용하기 간편하여 벡터 데이터를 관리하는 데 사용하기 편리한 솔루션입니다.

밀버스

MongoDB와 매우 비슷하게 문서 저장소와 같은 API를 제공하는 다용도 벡터 데이터베이스입니다. 다양한 데이터 유형을 지원한다는 점에서 두드러지며, 이로 인해 데이터 과학 및 머신 러닝 분야에서 인기 있는 선택이 되었습니다.

Milvus의 독특한 기능 중 하나는 다중 벡터화 기능으로, 사용자는 런타임에 검색에 사용할 벡터 유형을 지정할 수 있습니다. 더욱이 Faiss와 같은 다른 라이브러리 위에 있는 라이브러리인 Knowwhere를 활용하여 쿼리와 벡터 검색 알고리즘 간의 통신을 관리합니다.

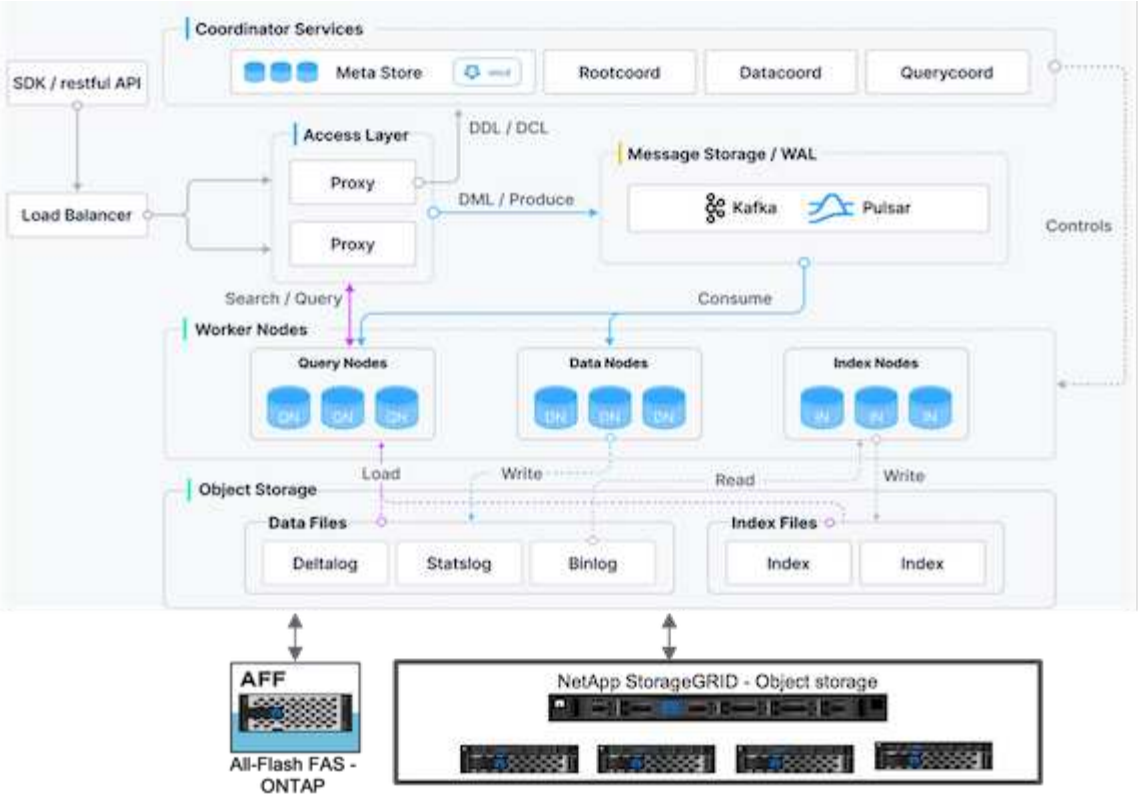
Milvus는 PyTorch 및 TensorFlow와의 호환성 덕분에 머신 러닝 워크플로우와의 원활한 통합을 제공합니다. 이로 인해 이 도구는 전자상거래, 이미지 및 비디오 분석, 객체 인식, 이미지 유사성 검색, 콘텐츠 기반 이미지 검색을 포함한 다양한 응용 분야에 매우 적합합니다. 자연어 처리 분야에서 Milvus는 문서 클러스터링, 의미 검색, 질의응답 시스템에 사용됩니다.

이 솔루션의 경우, 솔루션 검증을 위해 milvus를 선택했습니다. 성능을 위해 milvus와 postgres(pgvecto.rs)를 모두 사용했습니다.

이 솔루션을 위해 왜 **Milvus**를 선택했을까요**?**

- 오픈 소스: Milvus는 커뮤니티 중심의 개발과 개선을 장려하는 오픈 소스 벡터 데이터베이스입니다.
- AI 통합: 유사성 검색과 AI 애플리케이션을 내장하여 벡터 데이터베이스 기능을 강화합니다.
- 대용량 처리: Milvus는 딥 신경망(DNN) 및 머신 러닝(ML) 모델에서 생성된 10억 개 이상의 임베딩 벡터를 저장, 색인화하고 관리할 수 있는 역량을 갖추고 있습니다.
- 사용자 친화적: 사용하기 쉽고, 설정하는 데 1분도 걸리지 않습니다. Milvus는 또한 다양한 프로그래밍 언어에 대한 SDK를 제공합니다.
- 속도: 일부 대안보다 최대 10배 빠른 매우 빠른 검색 속도를 제공합니다.
- 확장성 및 가용성: Milvus는 필요에 따라 확장 및 축소할 수 있는 옵션을 갖추고 있어 확장성이 매우 뛰어납니다.
- 풍부한 기능: 다양한 데이터 유형, 속성 필터링, 사용자 정의 함수(UDF) 지원, 구성 가능한 일관성 수준 및 이동 시간을 지원하므로 다양한 애플리케이션에 적합한 다재다능한 도구입니다.

Milvus 아키텍처 개요



이 섹션에서는 Milvus 아키텍처에서 사용되는 상위 레벨 구성 요소와 서비스를 제공합니다. * 액세스 계층 – 상태 비저장 프록시 그룹으로 구성되며 시스템의 프런트 계층이자 사용자의 엔드포인트 역할을 합니다. * 코디네이터 서비스 – 작업자 노드에 작업을 할당하고 시스템의 두뇌 역할을 합니다. 세 가지 코디네이터 유형이 있습니다: 루트 좌표, 데이터 좌표, 쿼리 좌표. * 워커 노드: 코디네이터 서비스의 지시를 따르고 사용자가 트리거한 DML/DDL 명령을 실행합니다. 쿼리 노드, 데이터 노드, 인덱스 노드와 같이 세 가지 유형의 워커 노드가 있습니다. * 저장소: 데이터 지속성을 담당합니다. 여기에는 메타 스토리지, 로그 브로커, 객체 스토리지가 포함됩니다. ONTAP 및 StorageGRID 와 같은 NetApp 스토리지는 Milvus에 고객 데이터와 벡터 데이터베이스 데이터 모두를 위한 객체 스토리지와 파일 기반 스토리지를 제공합니다.

기술 요구 사항

이 섹션에서는 NetApp 벡터 데이터베이스 솔루션에 대한 요구 사항에 대한 개요를 제공합니다.

기술 요구 사항

아래에 설명된 하드웨어 및 소프트웨어 구성은 성능을 제외하고 이 문서에서 수행된 대부분의 검증에 활용되었습니다. 이러한 구성은 환경을 설정하는 데 도움이 되는 지침으로 사용됩니다. 그러나 구체적인 구성 요소는 개별 고객 요구 사항에 따라 달라질 수 있습니다.

하드웨어 요구 사항

하드웨어	세부
NetApp AFF 스토리지 어레이 HA 쌍	* A800 * ONTAP 9.14.1 * 48 x 3.49TB SSD-NVM * 두 개의 유연한 그룹 볼륨: 메타데이터 및 데이터. * 메타데이터 NFS 볼륨에는 250GB의 영구 볼륨이 12개 있습니다. * 데이터는 ONTAP NAS S3 볼륨입니다.

하드웨어	세부
후지쯔 프라이머지 RX2540 M4 6개	* 64개 CPU * Intel® Xeon® Gold 6142 CPU @ 2.60GHz * 256 GM 물리적 메모리 * 1 x 100GbE 네트워크 포트
네트워킹	100GbE
StorageGRID	* 1 x SG100, 3xSGF6024 * 3 x 24 x 7.68TB

소프트웨어 요구 사항

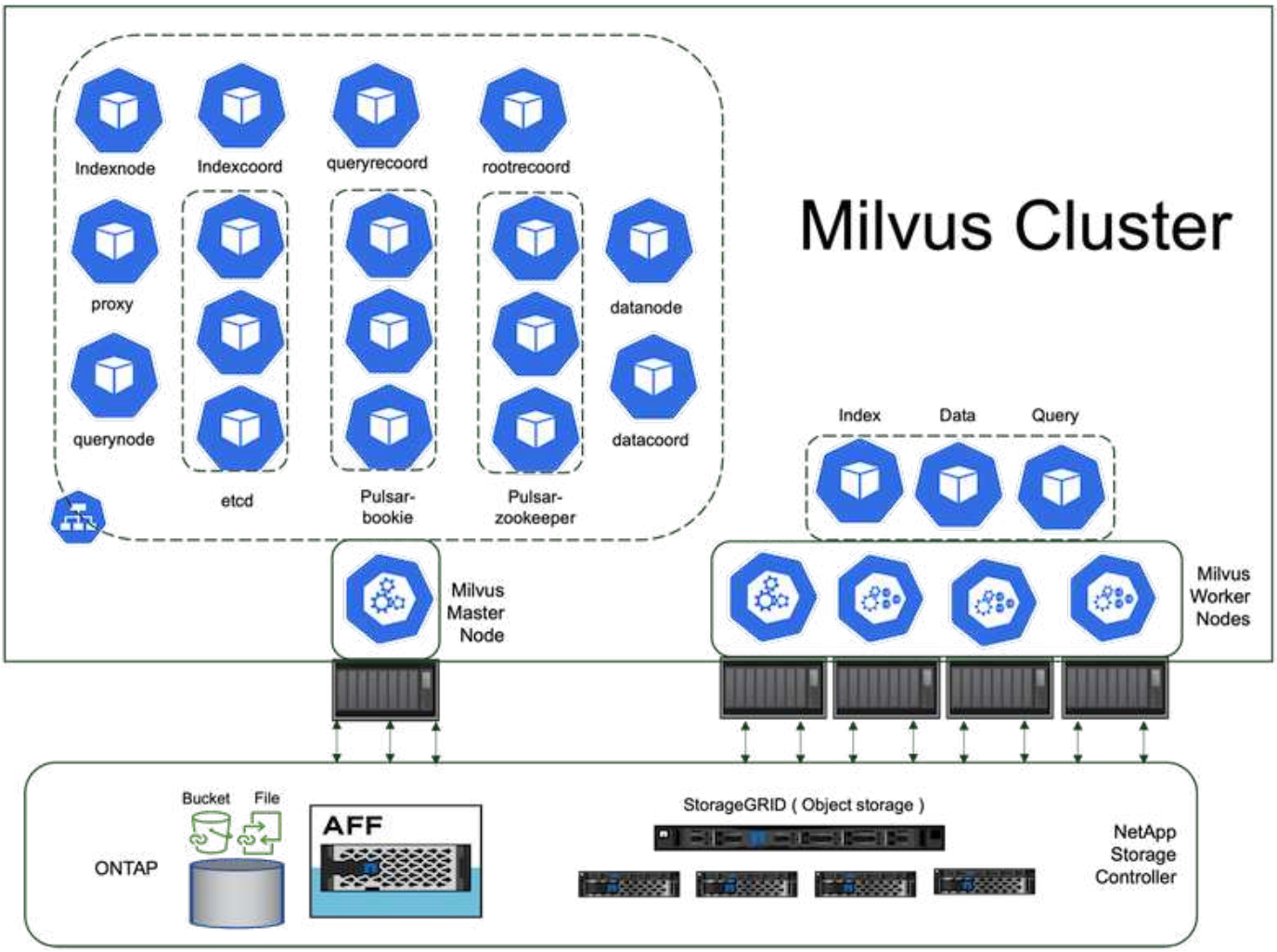
소프트웨어	세부
밀부스 클러스터	* 차트 - milvus-4.1.11. * APP 버전 – 2.3.4 * Bookkeeper, Zookeeper, Pulsar, Etcd, Proxy, QueryNode, Worker와 같은 종속 번들
쿠버네티스	* 5개 노드 K8s 클러스터 * 1개 마스터 노드 및 4개 워커 노드 * 버전 – 1.7.2
파이썬	*3.10.12.

배포 절차

이 섹션에서는 NetApp 용 벡터 데이터베이스 솔루션의 배포 절차에 대해 설명합니다.

배포 절차

이 배포 섹션에서는 아래와 같이 랩 설정을 위해 Kubernetes와 함께 milvus 벡터 데이터베이스를 사용했습니다.



NetApp 스토리지는 클러스터가 고객 데이터와 Milvus 클러스터 데이터를 보관할 수 있는 스토리지를 제공합니다.

NetApp 스토리지 설정 **– ONTAP**

- 스토리지 시스템 초기화
- 스토리지 가상 머신(SVM) 생성
- 논리적 네트워크 인터페이스 할당
- NFS, S3 구성 및 라이센싱

NFS(네트워크 파일 시스템)의 경우 아래 단계를 따르세요.

1. NFSv4에 대한 FlexGroup 볼륨을 생성합니다. 이 검증을 위한 설정에서 48개의 SSD를 사용했습니다. 1개의 SSD는 컨트롤러의 루트 볼륨에 전용으로 사용하고 47개의 SSD는 NFSv4에 분산했습니다. FlexGroup 볼륨에 대한 NFS 내보내기 정책에 Kubernetes(K8s) 노드 네트워크에 대한 읽기/쓰기 권한이 있는지 확인하세요. 이러한 권한이 없는 경우 K8s 노드 네트워크에 대한 읽기/쓰기(rw) 권한을 부여하세요.
2. 모든 K8s 노드에서 폴더를 만들고 각 K8s 노드의 논리 인터페이스(LIF)를 통해 FlexGroup 볼륨을 이 폴더에 마운트합니다.

NAS S3(Network Attached Storage Simple Storage Service)의 경우 아래 단계를 따르세요.

1. NFS용 FlexGroup 볼륨을 생성합니다.

2. "vserver object-store-server create" 명령을 사용하여 HTTP를 활성화하고 관리 상태를 'up'으로 설정한 객체 저장소 서버를 설정합니다. HTTPS를 활성화하고 사용자 정의 리스너 포트를 설정할 수 있습니다.
3. "vserver object-store-server user create -user <username>" 명령을 사용하여 object-store-server 사용자를 생성합니다.
4. 액세스 키와 비밀 키를 얻으려면 다음 명령을 실행하세요: "set diag; vserver object-store-server user show -user <username>". 하지만 앞으로는 이러한 키가 사용자 생성 프로세스 중에 제공되거나 REST API 호출을 사용하여 검색할 수 있습니다.
5. 2단계에서 생성한 사용자를 사용하여 객체-저장소-서버 그룹을 설정하고 액세스 권한을 부여합니다. 이 예에서는 "FullAccess"를 제공했습니다.
6. 유형을 "nas"로 설정하고 NFSv3 볼륨의 경로를 제공하여 NAS 버킷을 만듭니다. 이 목적으로 S3 버킷을 활용하는 것도 가능합니다.

NetApp 스토리지 설정 **– StorageGRID**

1. storageGRID 소프트웨어를 설치하세요.
2. 테넌트와 버킷을 생성합니다.
3. 필요한 권한이 있는 사용자를 생성합니다.

자세한 내용은 에서 확인하세요. https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

솔루션 검증

솔루션 개요

우리는 5가지 핵심 영역에 초점을 맞춘 포괄적인 솔루션 검증을 수행했으며, 자세한 내용은 아래에 설명되어 있습니다. 각 섹션에서는 고객이 직면한 과제, NetApp 이 제공하는 솔루션, 그리고 이를 통해 고객에게 제공되는 이점을 자세히 살펴봅니다.

1. "온프레미스에서 Kubernetes를 사용한 Milvus 클러스터 설정"고객이 스토리지와 컴퓨팅을 독립적으로 확장하고, 인프라를 효과적으로 관리하고, 데이터를 관리해야 하는 과제가 있습니다. 이 섹션에서는 클러스터 데이터와 고객 데이터 모두에 NetApp 스토리지 컨트롤러를 활용하여 Kubernetes에 Milvus 클러스터를 설치하는 과정을 자세히 설명합니다.
2. NetApp ONTAP 용 Amazon FSx ONTAP 과 함께 사용하는 Milvus - 파일과 객체의 이중성 이 섹션에서는 왜 클라우드에 벡터 데이터베이스를 배포해야 하는지, 그리고 Docker 컨테이너 내에서 NetApp ONTAP 용 Amazon FSx ONTAP 에 벡터 데이터베이스(milvus 독립형)를 배포하는 단계에 대해 설명합니다.
3. "NetApp SnapCenter 사용한 벡터 데이터베이스 보호."이 섹션에서는 SnapCenter ONTAP 에 있는 벡터 데이터베이스 데이터와 Milvus 데이터를 어떻게 보호하는지 자세히 살펴보겠습니다. 이 예에서는 고객 데이터에 대한 NFS ONTAP 볼륨(vol1)에서 파생된 NAS 버킷(milvusdbvol1)과 Milvus 클러스터 구성 데이터에 대한 별도의 NFS 볼륨(vectordbpv)을 활용했습니다.
4. "NetApp SnapMirror 사용한 재해 복구"이 섹션에서는 벡터 데이터베이스에 대한 재해 복구(DR)의 중요성과 NetApp 재해 복구 제품인 Snapmirror가 벡터 데이터베이스에 DR 솔루션을 제공하는 방식에 대해 설명합니다.
5. "성능 검증"이 섹션에서는 Milvus 및 pgvecto.rs와 같은 벡터 데이터베이스의 성능 검증을 자세히 살펴보고, LLM 수명 주기 내에서 RAG 및 추론 워크로드를 지원하는 I/O 프로필 및 netapp 스토리지 컨트롤러 동작과 같은 스토리지 성능 특성에 중점을 둡니다. 이러한 데이터베이스를 ONTAP 스토리지 솔루션과 결합하면 성능상의 차이점을 평가하고 파악할 것입니다. 당사의 분석은 초당 처리되는 쿼리 수(QPS)와 같은 핵심 성과 지표를 기반으로 진행됩니다.

온프레미스에서 **Kubernetes**를 사용한 **Milvus** 클러스터 설정

이 섹션에서는 NetApp 의 벡터 데이터베이스 솔루션을 위한 milvus 클러스터 설정에 대해 설명합니다.

온프레미스에서 **Kubernetes**를 사용한 **Milvus** 클러스터 설정

고객이 스토리지와 컴퓨팅을 독립적으로 확장하고, 효과적인 인프라 관리와 데이터 관리를 수행해야 하는 과제에 직면해 있습니다. Kubernetes와 벡터 데이터베이스는 함께 대규모 데이터 작업을 관리하기 위한 강력하고 확장 가능한 솔루션을 형성합니다. 쿠버네티스는 리소스를 최적화하고 컨테이너를 관리하는 반면, 벡터 데이터베이스는 고차원 데이터와 유사성 검색을 효율적으로 처리합니다. 이러한 조합을 통해 대규모 데이터 세트에 대한 복잡한 쿼리를 신속하게 처리할 수 있으며 증가하는 데이터 볼륨에 따라 원활하게 확장되므로 빅데이터 애플리케이션과 AI 워크로드에 이상적입니다.

1. 이 섹션에서는 클러스터 데이터와 고객 데이터 모두에 NetApp 스토리지 컨트롤러를 활용하여 Kubernetes에 Milvus 클러스터를 설치하는 과정을 자세히 설명합니다.
2. Milvus 클러스터를 설치하려면 다양한 Milvus 클러스터 구성 요소의 데이터를 저장하기 위한 영구 볼륨(PV)이 필요합니다. 이러한 구성 요소에는 etcd(인스턴스 3개), pulsar-bookie-journal(인스턴스 3개), pulsar-bookie-ledgers(인스턴스 3개), pulsar-zookeeper-data(인스턴스 3개)가 포함됩니다.

> Milvus 클러스터에서는 Milvus 클러스터의 안정적인 저장과 메시지 스트림의 게시/구독을 지원하는 기본 엔진으로 Pulsar나 Kafka를 사용할 수 있습니다. NFS를 사용하는 Kafka의 경우, NetApp ONTAP 9.12.1 이상에서 개선 사항을 적용했으며, 이러한 개선 사항과 RHEL 8.7 또는 9.1 이상에 포함된 NFSv4.1 및 Linux 변경 사항은 NFS에서 Kafka를 실행할 때 발생할 수 있는 "쓸데없는 이름 바꾸기" 문제를 해결합니다. NetApp NFS 솔루션에서 Kafka 실행에 대한 자세한 내용은 다음을 참조하세요."이 링크" .

3. NetApp ONTAP 에서 단일 NFS 볼륨을 생성하고 각각 250GB의 저장 용량을 갖춘 12개의 영구 볼륨을 설정했습니다. 저장 용량은 클러스터 크기에 따라 달라질 수 있습니다. 예를 들어, 각 PV가 50GB인 다른 클러스터가 있습니다. 자세한 내용은 아래 PV YAML 파일 중 하나를 참조하세요. 총 12개의 파일이 있습니다. 각 파일에서 storageClassName은 'default'로 설정되고, 저장소와 경로는 각 PV마다 고유합니다.

```
root@node2:~# cat sai_nfs_to_default_pv1.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: karthik-pv1
spec:
  capacity:
    storage: 250Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: default
  local:
    path: /vectordbsc/milvus/milvus1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node2
          - node3
          - node4
          - node5
          - node6
root@node2:~#
```

4. 각 PV YAML 파일에 대해 'kubectl apply' 명령을 실행하여 영구 볼륨을 생성한 다음 'kubectl get pv'를 사용하여 생성을 확인합니다.

```
root@node2:~# for i in $( seq 1 12 ); do kubectl apply -f
sai_nfs_to_default_pv$i.yaml; done
persistentvolume/karthik-pv1 created
persistentvolume/karthik-pv2 created
persistentvolume/karthik-pv3 created
persistentvolume/karthik-pv4 created
persistentvolume/karthik-pv5 created
persistentvolume/karthik-pv6 created
persistentvolume/karthik-pv7 created
persistentvolume/karthik-pv8 created
persistentvolume/karthik-pv9 created
persistentvolume/karthik-pv10 created
persistentvolume/karthik-pv11 created
persistentvolume/karthik-pv12 created
root@node2:~#
```

5. Milvus는 고객 데이터를 저장하기 위해 MinIO, Azure Blob, S3와 같은 개체 스토리지 솔루션을 지원합니다. 이 가이드에서는 S3를 활용합니다. 다음 단계는 ONTAP S3와 StorageGRID 개체 저장소 모두에 적용됩니다. Milvus 클러스터를 배포하기 위해 Helm을 사용합니다. Milvus 다운로드 위치에서 구성 파일 values.yaml을 다운로드합니다. 이 문서에서 사용한 values.yaml 파일은 부록을 참조하세요.
6. 로그, etcd, zookeeper, bookkeeper를 포함하여 각 섹션에서 'storageClass'가 'default'로 설정되어 있는지 확인하세요.
7. MinIO 섹션에서 MinIO를 비활성화합니다.
8. ONTAP 또는 StorageGRID 개체 스토리지에서 NAS 버킷을 만들고 개체 스토리지 자격 증명을 사용하여 외부 S3에 포함합니다.

```
##################################
# External S3
# - these configs are only used when `externalS3.enabled` is true
##################################
externalS3:
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
  useSSL: false
  bucketName: "milvusdbvol1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
```

9. Milvus 클러스터를 생성하기 전에 PersistentVolumeClaim(PVC)에 기존 리소스가 없는지 확인하세요.

```
root@node2:~# kubectl get pvc
No resources found in default namespace.
root@node2:~#
```

10. Helm과 values.yaml 구성 파일을 활용하여 Milvus 클러스터를 설치하고 시작합니다.

```
root@node2:~# helm upgrade --install my-release milvus/milvus --set
global.storageClass=default  -f values.yaml
Release "my-release" does not exist. Installing it now.
NAME: my-release
LAST DEPLOYED: Thu Mar 14 15:00:07 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
root@node2:~#
```

11. PersistentVolumeClaims(PVC)의 상태를 확인합니다.

```
root@node2:~# kubectl get pvc
NAME                                                                    STATUS
VOLUME         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-my-release-etcd-0                                                  Bound
karthik-pv8    250Gi      RWO            default        3s
data-my-release-etcd-1                                                  Bound
karthik-pv5    250Gi      RWO            default        2s
data-my-release-etcd-2                                                  Bound
karthik-pv4    250Gi      RWO            default        3s
my-release-pulsar-bookie-journal-my-release-pulsar-bookie-0      Bound
karthik-pv10   250Gi      RWO            default        3s
my-release-pulsar-bookie-journal-my-release-pulsar-bookie-1      Bound
karthik-pv3    250Gi      RWO            default        3s
my-release-pulsar-bookie-journal-my-release-pulsar-bookie-2      Bound
karthik-pv1    250Gi      RWO            default        3s
my-release-pulsar-bookie-ledgers-my-release-pulsar-bookie-0      Bound
karthik-pv2    250Gi      RWO            default        3s
my-release-pulsar-bookie-ledgers-my-release-pulsar-bookie-1      Bound
karthik-pv9    250Gi      RWO            default        3s
my-release-pulsar-bookie-ledgers-my-release-pulsar-bookie-2      Bound
karthik-pv11   250Gi      RWO            default        3s
my-release-pulsar-zookeeper-data-my-release-pulsar-zookeeper-0   Bound
karthik-pv7    250Gi      RWO            default        3s
root@node2:~#
```

12. 포드의 상태를 확인하세요.

```
root@node2:~# kubectl get pods -o wide
NAME                                                  READY   STATUS
RESTARTS        AGE    IP              NODE    NOMINATED NODE
READINESS GATES
<content removed to save page space>
```

포드 상태가 '실행 중'이고 예상대로 작동하는지 확인하세요.

13. Milvus와 NetApp 개체 스토리지에서 데이터 쓰기와 읽기를 테스트합니다.
    - "prepare_data_netapp_new.py" Python 프로그램을 사용하여 데이터를 작성합니다.

```
root@node2:~# date;python3 prepare_data_netapp_new.py ;date
Thu Apr  4 04:15:35 PM UTC 2024
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===
Does collection hello_milvus_ntapnew_update2_sc exist in Milvus:
False
=== Drop collection - hello_milvus_ntapnew_update2_sc ===
=== Drop collection - hello_milvus_ntapnew_update2_sc2 ===
=== Create collection `hello_milvus_ntapnew_update2_sc` ===
=== Start inserting entities       ===
Number of entities in hello_milvus_ntapnew_update2_sc: 3000
Thu Apr  4 04:18:01 PM UTC 2024
root@node2:~#
```

- "verify_data_netapp.py" Python 파일을 사용하여 데이터를 읽습니다.

```
root@node2:~# python3 verify_data_netapp.py
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===

Does collection hello_milvus_ntapnew_update2_sc exist in Milvus: True
{'auto_id': False, 'description': 'hello_milvus_ntapnew_update2_sc',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64:
5>, 'is_primary': True, 'auto_id': False}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 16}}]}
Number of entities in Milvus: hello_milvus_ntapnew_update2_sc : 3000

=== Start Creating index IVF_FLAT  ===

=== Start loading                  ===

=== Start searching based on vector similarity ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 2600, distance: 0.602496862411499, entity: {'random':
0.3098157043984633}, random field: 0.3098157043984633
hit: id: 1831, distance: 0.6797959804534912, entity: {'random':
0.6331477114129169}, random field: 0.6331477114129169
hit: id: 2999, distance: 0.0, entity: {'random':
0.02316334456872482}, random field: 0.02316334456872482
hit: id: 2524, distance: 0.5918987989425659, entity: {'random':
```

```
0.285283165889066}, random field: 0.285283165889066
hit: id: 264, distance: 0.7254047393798828, entity: {'random':
0.3329096143562196}, random field: 0.3329096143562196
search latency = 0.4533s

=== Start querying with `random > 0.5` ===

query result:
-{'random': 0.6378742006852851, 'embeddings': [0.20963514,
0.39746657, 0.12019053, 0.6947492, 0.9535575, 0.5454552, 0.82360446,
0.21096309, 0.52323616, 0.8035404, 0.77824664, 0.80369574, 0.4914803,
0.8265614, 0.6145269, 0.80234545], 'pk': 0}
search latency = 0.4476s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 1831, distance: 0.6797959804534912, entity: {'random':
0.6331477114129169}, random field: 0.6331477114129169
hit: id: 678, distance: 0.7351570129394531, entity: {'random':
0.5195484662306603}, random field: 0.5195484662306603
hit: id: 2644, distance: 0.8620758056640625, entity: {'random':
0.9785952878381153}, random field: 0.9785952878381153
hit: id: 1960, distance: 0.9083120226860046, entity: {'random':
0.6376039340439571}, random field: 0.6376039340439571
hit: id: 106, distance: 0.9792704582214355, entity: {'random':
0.9679994241326673}, random field: 0.9679994241326673
search latency = 0.1232s
Does collection hello_milvus_ntapnew_update2_sc2 exist in Milvus:
True
{'auto_id': True, 'description': 'hello_milvus_ntapnew_update2_sc2',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64:
5>, 'is_primary': True, 'auto_id': True}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 16}}]}
```

위의 검증을 바탕으로 NetApp 스토리지 컨트롤러를 사용하여 Kubernetes에 Milvus 클러스터를 배포하는 방식으로 Kubernetes와 벡터 데이터베이스를 통합하면 고객에게 대규모 데이터 작업을 관리할 수 있는 견고하고 확장 가능하며 효율적인 솔루션을 제공합니다. 이러한 설정은 고객에게 고차원 데이터를 처리하고 복잡한 쿼리를 빠르고 효율적으로 실행할 수 있는 기능을 제공하므로 빅데이터 애플리케이션과 AI 워크로드에 이상적인 솔루션입니다. 다양한 클러스터 구성 요소에 대해 영구 볼륨(PV)을 사용하고 NetApp ONTAP 에서 단일 NFS 볼륨을 생성하면 최적의 리소스 활용과 데이터 관리가 보장됩니다. PersistentVolumeClaims(PVC) 및 Pod 상태를 검증하고, 데이터 쓰기 및 읽기를 테스트하는 프로세스를 통해 고객은 안정적이고 일관된 데이터 작업을 보장받을 수 있습니다. 고객 데이터에 ONTAP 또는 StorageGRID 개체 스토리지를 사용하면 데이터

접근성과 보안이 더욱 향상됩니다. 전반적으로 이러한 설정은 고객에게 증가하는 데이터 요구에 맞춰 원활하게 확장할 수 있는 탄력적이고 고성능 데이터 관리 솔루션을 제공합니다.

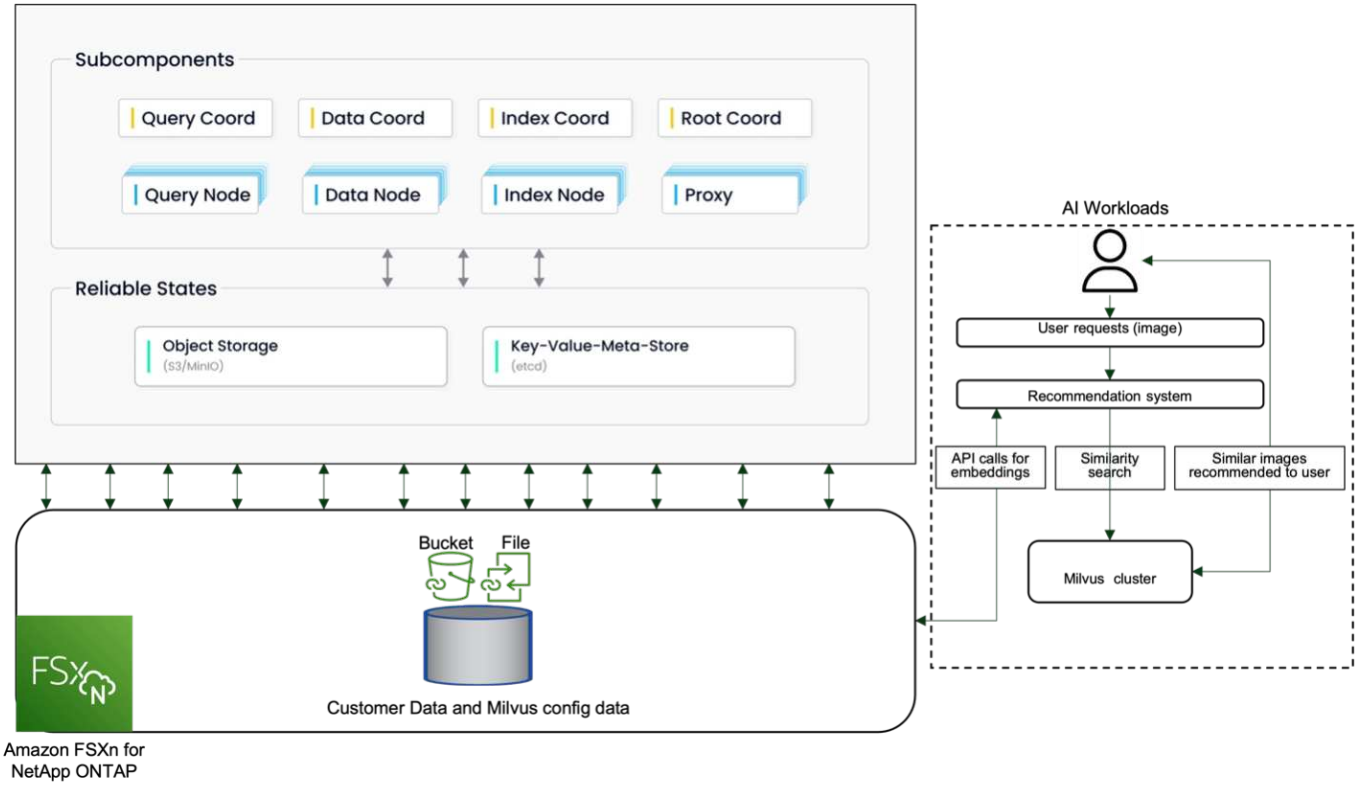
NetApp ONTAP 용 Amazon FSx ONTAP 탑재한 Milvus - 파일 및 객체 이중성

이 섹션에서는 NetApp 의 벡터 데이터베이스 솔루션을 위한 Amazon FSx ONTAP 사용한 milvus 클러스터 설정에 대해 설명합니다.

NetApp ONTAP 용 Amazon FSx ONTAP 탑재한 Milvus – 파일 및 객체 이중성

이 섹션에서는 왜 클라우드에 벡터 데이터베이스를 배포해야 하는지, 그리고 Docker 컨테이너 내 NetApp ONTAP 용 Amazon FSx ONTAP 에 벡터 데이터베이스(milvus standalone)를 배포하는 단계를 알아봅니다.

클라우드에 벡터 데이터베이스를 구축하면 여러 가지 중요한 이점이 있는데, 특히 고차원 데이터를 처리하고 유사성 검색을 실행해야 하는 애플리케이션의 경우 이점이 큽니다. 첫째, 클라우드 기반 배포는 확장성을 제공하므로 증가하는 데이터 볼륨과 쿼리 부하에 맞춰 리소스를 쉽게 조정할 수 있습니다. 이를 통해 데이터베이스는 높은 성능을 유지하는 동시에 증가된 수요를 효율적으로 처리할 수 있습니다. 두 번째로, 클라우드 구축은 여러 지리적 위치에 걸쳐 데이터를 복제할 수 있으므로 높은 가용성과 재해 복구를 제공하고, 데이터 손실 위험을 최소화하며, 예상치 못한 이벤트 발생 시에도 지속적인 서비스를 보장합니다. 셋째, 사용한 리소스에 대해서만 비용을 지불하고 수요에 따라 확장하거나 축소할 수 있으므로 하드웨어에 대한 상당한 사전 투자가 필요 없어 비용 효율성이 높습니다. 마지막으로, 클라우드에 벡터 데이터베이스를 구축하면 어디서나 데이터에 접근하고 공유할 수 있으므로 협업이 향상되고, 팀 기반 작업과 데이터 기반 의사 결정이 용이해집니다. 이 검증에 사용된 NetApp ONTAP 용 Amazon FSx ONTAP 과 함께 사용되는 milvus 독립형의 아키텍처를 확인하세요.



1. NetApp ONTAP 인스턴스를 위한 Amazon FSx ONTAP 생성하고 VPC, VPC 보안 그룹 및 서브넷의 세부 정보를 기록해 둡니다. 이 정보는 EC2 인스턴스를 생성할 때 필요합니다. 자세한 내용은 여기에서 확인할 수 있습니다. https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. VPC, 보안 그룹 및 서브넷이 NetApp ONTAP 인스턴스의 Amazon FSx ONTAP 과 일치하는지 확인하여 EC2

인스턴스를 생성합니다.

3. 'apt-get install nfs-common' 명령어를 사용하여 nfs-common을 설치하고 'sudo apt-get update'를 사용하여 패키지 정보를 업데이트합니다.
4. 마운트 폴더를 만들고 해당 폴더에 NetApp ONTAP 용 Amazon FSx ONTAP 마운트합니다.

```
ubuntu@ip-172-31-29-98:~$ mkdir /home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$ sudo mount 172.31.255.228:/vol1
/home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$ df -h /home/ubuntu/milvusvectordb
Filesystem            Size  Used Avail Use% Mounted on
172.31.255.228:/vol1  973G  126G  848G  13% /home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$
```

5. 'apt-get install'을 사용하여 Docker와 Docker Compose를 설치합니다.
6. Milvus 웹사이트에서 다운로드할 수 있는 docker-compose.yaml 파일을 기반으로 Milvus 클러스터를 설정합니다.

```
root@ip-172-31-22-245:~# wget https://github.com/milvus-
io/milvus/releases/download/v2.0.2/milvus-standalone-docker-compose.yml
-O docker-compose.yml
--2024-04-01 14:52:23--  https://github.com/milvus-
io/milvus/releases/download/v2.0.2/milvus-standalone-docker-compose.yml
<removed some output to save page space>
```

7. docker-compose.yml 파일의 'volumes' 섹션에서 NetApp NFS 마운트 지점을 해당 Milvus 컨테이너 경로(특히 etcd, minio 및 standalone)에 매핑합니다."부록 D: docker-compose.yml" yml 변경 사항에 대한 자세한 내용은
8. 마운트된 폴더와 파일을 확인하세요.

```
ubuntu@ip-172-31-29-98:~/milvusvectordb$ ls -ltrh
/home/ubuntu/milvusvectordb
total 8.0K
-rw-r--r-- 1 root root 1.8K Apr  2 16:35 s3_access.py
drwxrwxrwx 2 root root 4.0K Apr  4 20:19 volumes
ubuntu@ip-172-31-29-98:~/milvusvectordb$ ls -ltrh
/home/ubuntu/milvusvectordb/volumes/
total 0
ubuntu@ip-172-31-29-98:~/milvusvectordb$ cd
ubuntu@ip-172-31-29-98:~$ ls
docker-compose.yml  docker-compose.yml~  milvus.yaml  milvusvectordb
vectordbvol1
ubuntu@ip-172-31-29-98:~$
```

9. docker-compose.yml 파일이 있는 디렉토리에서 'docker-compose up -d'를 실행합니다.

10. Milvus 컨테이너의 상태를 확인하세요.

```
ubuntu@ip-172-31-29-98:~$ sudo docker-compose ps
      Name                     Command                  State
Ports
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
----------
milvus-etcd         etcd -advertise-client-url ...   Up (healthy)
2379/tcp, 2380/tcp
milvus-minio        /usr/bin/docker-entrypoint ...   Up (healthy)
0.0.0.0:9000->9000/tcp,:::9000->9000/tcp, 0.0.0.0:9001-
>9001/tcp,:::9001->9001/tcp
milvus-standalone   /tini -- milvus run standalone   Up (healthy)
0.0.0.0:19530->19530/tcp,:::19530->19530/tcp, 0.0.0.0:9091-
>9091/tcp,:::9091->9091/tcp
ubuntu@ip-172-31-29-98:~$
ubuntu@ip-172-31-29-98:~$ ls -ltrh /home/ubuntu/milvusvectordb/volumes/
total 12K
drwxr-xr-x 3 root root 4.0K Apr  4 20:21 etcd
drwxr-xr-x 4 root root 4.0K Apr  4 20:21 minio
drwxr-xr-x 5 root root 4.0K Apr  4 20:21 milvus
ubuntu@ip-172-31-29-98:~$
```

11. NetApp ONTAP 용 Amazon FSx ONTAP 에서 벡터 데이터베이스와 데이터의 읽기 및 쓰기 기능을 검증하기 위해 Python Milvus SDK와 PyMilvus의 샘플 프로그램을 사용했습니다. 'apt-get install python3-numpy python3-pip'를 사용하여 필요한 패키지를 설치하고 'pip3 install pymilvus'를 사용하여 PyMilvus를 설치합니다.

12. 벡터 데이터베이스에서 NetApp ONTAP 용 Amazon FSx ONTAP 의 데이터 쓰기 및 읽기 작업을 검증합니다.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3
prepare_data_netapp_new.py
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===
Does collection hello_milvus_ntapnew_sc exist in Milvus: True
=== Drop collection - hello_milvus_ntapnew_sc ===
=== Drop collection - hello_milvus_ntapnew_sc2 ===
=== Create collection `hello_milvus_ntapnew_sc` ===
=== Start inserting entities       ===
Number of entities in hello_milvus_ntapnew_sc: 9000
root@ip-172-31-29-98:~/pymilvus/examples# find
/home/ubuntu/milvusvectordb/
…
<removed content to save page space >
…
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
```

```
/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100/448789845791411920
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
```

13. verify_data_netapp.py 스크립트를 사용하여 읽기 작업을 확인합니다.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3 verify_data_netapp.py
=== start connecting to Milvus     ===

=== Milvus host: localhost         ===

Does collection hello_milvus_ntapnew_sc exist in Milvus: True
{'auto_id': False, 'description': 'hello_milvus_ntapnew_sc', 'fields':
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,
'is_primary': True, 'auto_id': False}, {'name': 'random', 'description':
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',
```

```
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},
{'name': 'embeddings', 'description': '', 'type': <DataType.
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':
False}
Number of entities in Milvus: hello_milvus_ntapnew_sc : 9000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: id: 2248, distance: 0.0, entity: {'random': 0.2777646777746381},
random field: 0.2777646777746381
hit: id: 4837, distance: 0.07805602252483368, entity: {'random':
0.6451650959930306}, random field: 0.6451650959930306
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':
0.6141351712303128}, random field: 0.6141351712303128
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},
random field: 0.7434908973629817
hit: id: 830, distance: 0.05628090724349022, entity: {'random':
0.8544487225667627}, random field: 0.8544487225667627
hit: id: 8562, distance: 0.0797122716903686 5, entity: {'random':
0.4464554280115878}, random field: 0.4464554280115878
search latency = 0.1266s

=== Start querying with `random > 0.5` ===

query result:
-{'random': 0.6378742006852851, 'embeddings': [0.3017092, 0.74452263,
0.8009826, 0.4927033, 0.12762444, 0.29869467, 0.52859956, 0.23734547],
'pk': 0}
search latency = 0.3294s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 4837, distance: 0.07805602252483368, entity: {'random':
0.6451650959930306}, random field: 0.6451650959930306
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':
0.6141351712303128}, random field: 0.6141351712303128
hit: id: 515, distance: 0.09590047597885132, entity: {'random':
0.8013175797590888}, random field: 0.8013175797590888
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},
random field: 0.7434908973629817
hit: id: 830, distance: 0.05628090724349022, entity: {'random':
```

```
0.8544487225667627}, random field: 0.8544487225667627
hit: id: 1627, distance: 0.08096684515476227, entity: {'random':
0.9302397069516164}, random field: 0.9302397069516164
search latency = 0.2674s
Does collection hello_milvus_ntapnew_sc2 exist in Milvus: True
{'auto_id': True, 'description': 'hello_milvus_ntapnew_sc2', 'fields':
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,
'is_primary': True, 'auto_id': True}, {'name': 'random', 'description':
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},
{'name': 'embeddings', 'description': '', 'type': <DataType.
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':
False}
```

14. 고객이 AI 워크로드를 위해 S3 프로토콜을 통해 벡터 데이터베이스에서 테스트된 NFS 데이터에 액세스(읽기)하려는 경우 간단한 Python 프로그램을 사용하여 이를 검증할 수 있습니다. 이에 대한 예로는 이 섹션의 시작 부분에 있는 그림에서 언급한 것처럼 다른 애플리케이션의 이미지에 대한 유사성 검색을 들 수 있습니다.

```
root@ip-172-31-29-98:~/pymilvus/examples# sudo python3
/home/ubuntu/milvusvectordb/s3_access.py -i 172.31.255.228 --bucket
milvusnasvol --access-key PY6UF318996I86NBYNDD --secret-key
hoPctr9aD88c1j0SkIYZ2uPa03vlbqKA0c5feK6F
OBJECTS in the bucket milvusnasvol are :
****************************************
…
<output content removed to save page space>
…
bucket/files/insert_log/448789845791611912/448789845791611913/4487898457
91611920/0/448789845791411917/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/1/448789845791411918/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411913/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/101/448789845791411914/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/102/448789845791411915/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/1c48ab6e-
1546-4503-9084-28c629216c33/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
```

```
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/101/448789845791411921/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/102/448789845791411922/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-
c117-4fba-8256-96cb7557cd6c/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/448789845791411912/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411919/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411926/xl.meta
****************************************
root@ip-172-31-29-98:~/pymilvus/examples#
```

이 섹션에서는 고객이 Amazon의 NetApp FSx ONTAP NetApp ONTAP 데이터 스토리지에 활용하여 Docker 컨테이너 내에서 독립형 Milvus 설정을 배포하고 운영하는 방법을 효과적으로 보여줍니다. 이러한 설정을 통해 고객은 Docker 컨테이너의 확장 가능하고 효율적인 환경 내에서 벡터 데이터베이스의 힘을 활용하여 고차원 데이터를 처리하고 복잡한 쿼리를 실행할 수 있습니다. NetApp ONTAP 인스턴스와 일치하는 EC2 인스턴스를 위한 Amazon FSx ONTAP 생성하면 고객은 최적의 리소스 활용과 데이터 관리를 보장할 수 있습니다. FSx ONTAP 에서 벡터 데이터베이스의 데이터 쓰기 및 읽기 작업을 성공적으로 검증함으로써 고객은 안정적이고 일관된 데이터 작업을 보장받을 수 있습니다. 또한 S3 프로토콜을 통해 AI 워크로드의 데이터를 나열(읽기)하는 기능을 통해 데이터 접근성이 향상됩니다. 따라서 이 포괄적인 프로세스는 고객에게 Amazon FSx ONTAP for NetApp ONTAP 의 기능을 활용하여 대규모 데이터 작업을 관리할 수 있는 강력하고 효율적인 솔루션을 제공합니다.

SnapCenter 사용한 벡터 데이터베이스 보호

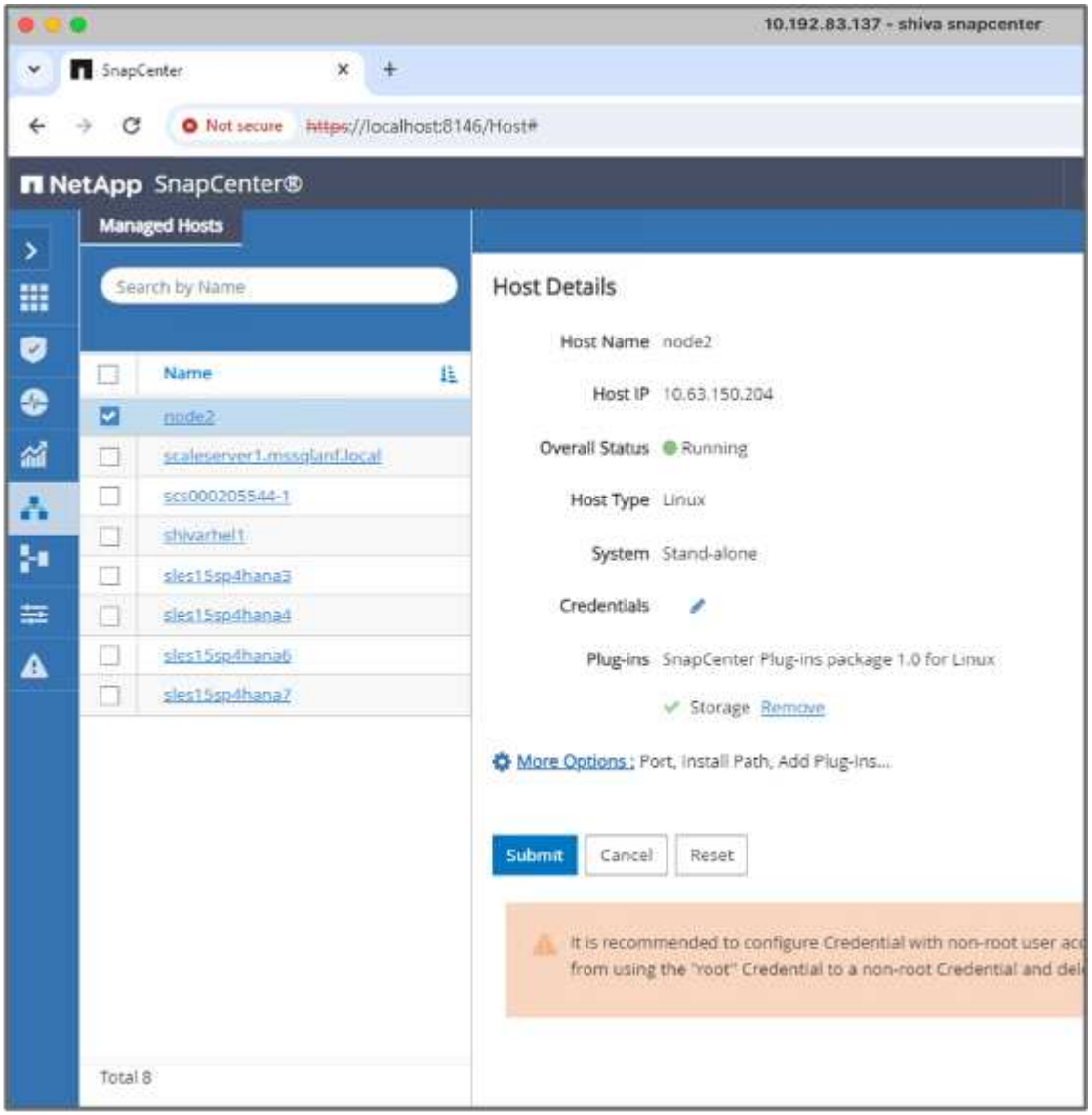
이 섹션에서는 NetApp SnapCenter 사용하여 벡터 데이터베이스에 대한 데이터 보호를 제공하는 방법을 설명합니다.

NetApp SnapCenter 사용한 벡터 데이터베이스 보호.

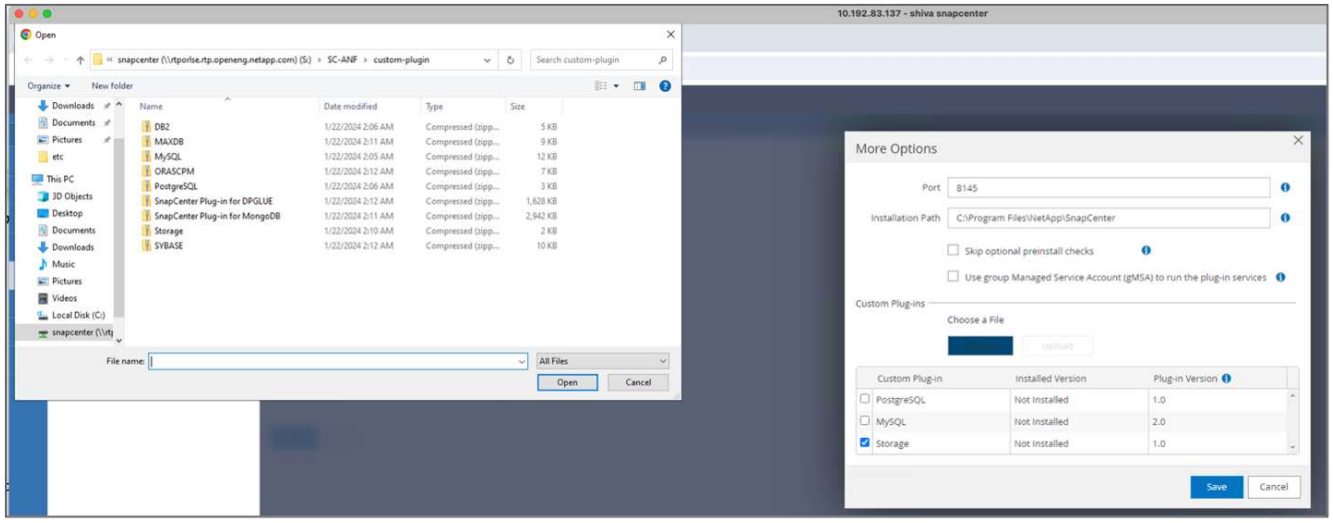
예를 들어, 영화 제작 산업에서 고객은 종종 비디오 및 오디오 파일과 같은 중요한 내장 데이터를 보유하고 있습니다. 하드 드라이브 오류 등의 문제로 인해 이러한 데이터가 손실되면 운영에 상당한 영향을 미칠 수 있으며, 수백만 달러 규모의 사업이 위험에 처할 수도 있습니다. 우리는 귀중한 콘텐츠가 손실되어 상당한 중단과 재정적 손실을 초래하는

사례들을 경험했습니다. 따라서 이 필수 데이터의 보안과 무결성을 보장하는 것이 이 업계에서 가장 중요합니다. 이 섹션에서는 SnapCenter ONTAP 에 있는 벡터 데이터베이스 데이터와 Milvus 데이터를 어떻게 보호하는지 자세히 살펴보겠습니다. 이 예에서는 고객 데이터에 대해 NFS ONTAP 볼륨(vol1)에서 파생된 NAS 버킷(milvusdbvol1)을 활용했으며 Milvus 클러스터 구성 데이터에 대해 별도의 NFS 볼륨(vectordbpv)을 활용했습니다. 다음을 확인하십시오 ."여기" SnapCenter 백업 워크플로우용

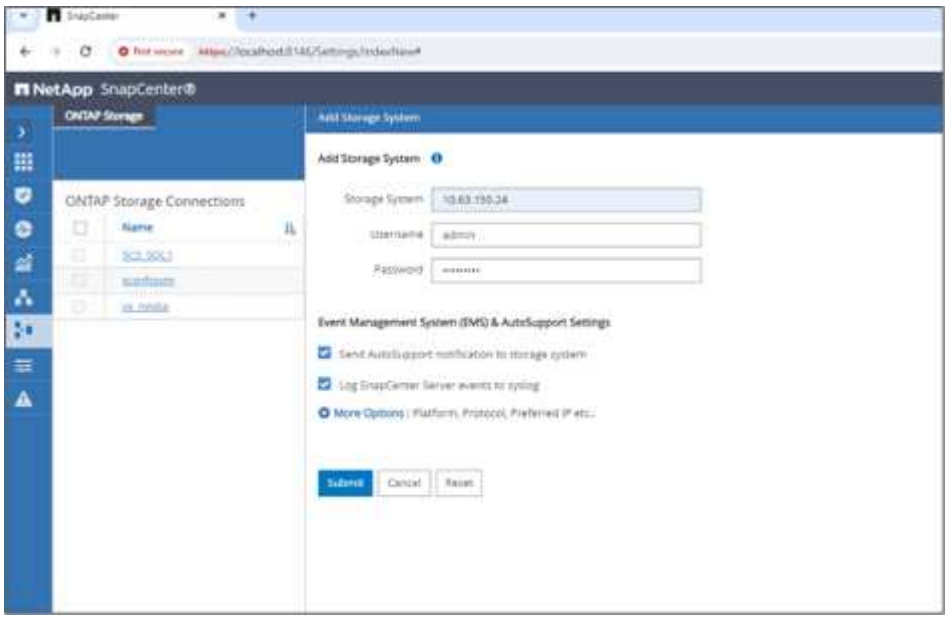
1. SnapCenter 명령을 실행하는 데 사용될 호스트를 설정합니다.



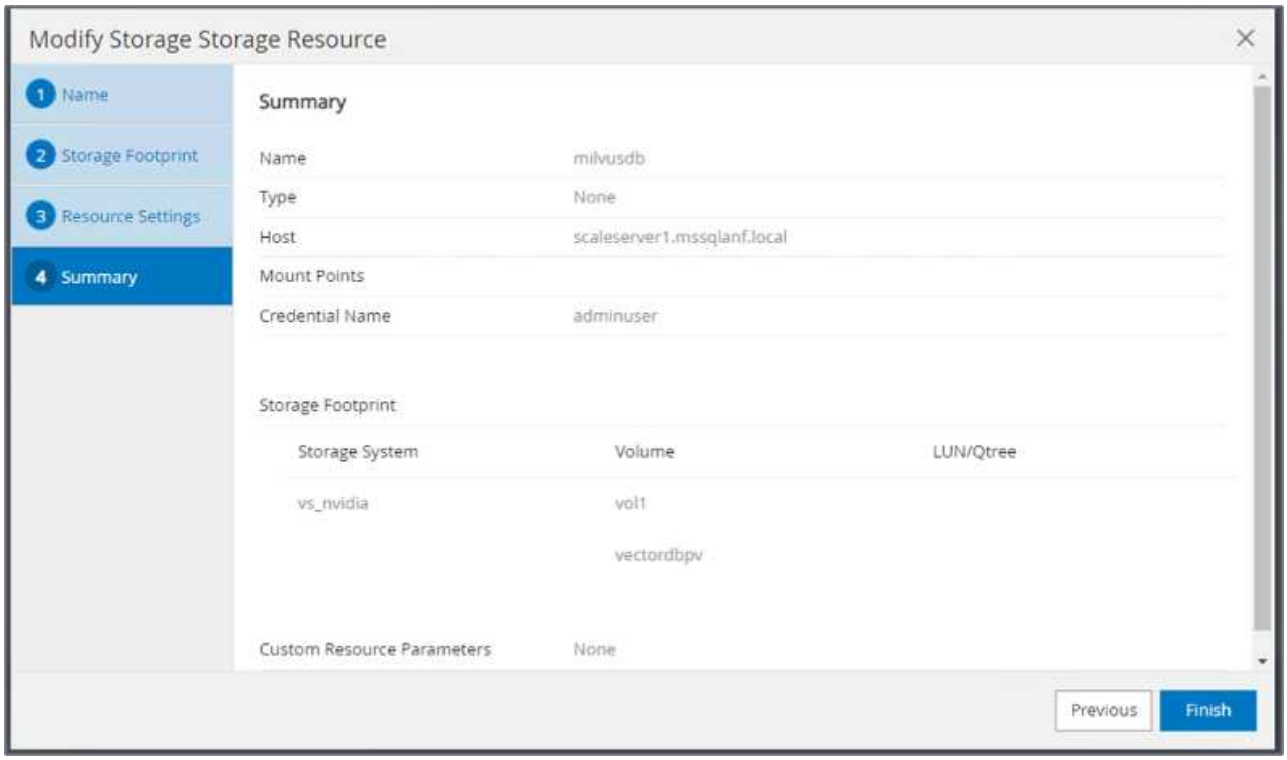
2. 스토리지 플러그인을 설치하고 구성합니다. 추가된 호스트에서 "추가 옵션"을 선택하세요. 다운로드한 저장소 플러그인을 탐색하여 선택하세요."NetApp 자동화 스토어" . 플러그인을 설치하고 구성을 저장합니다.



3. 스토리지 시스템 및 볼륨 설정: "스토리지 시스템"에서 스토리지 시스템을 추가하고 SVM(스토리지 가상 머신)을 선택합니다. 이 예에서는 "vs_nvidia"를 선택했습니다.



4. 백업 정책과 사용자 정의 스냅샷 이름을 통합하여 벡터 데이터베이스에 대한 리소스를 설정합니다.
   - 기본값으로 일관성 그룹 백업을 활성화하고 파일 시스템 일관성 없이 SnapCenter 활성화합니다.
   - 저장소 공간 섹션에서 벡터 데이터베이스 고객 데이터와 Milvus 클러스터 데이터와 연결된 볼륨을 선택합니다. 우리의 예에서는 "vol1"과 "vectordbpv"가 있습니다.
   - 벡터 데이터베이스 보호에 대한 정책을 만들고 해당 정책을 사용하여 벡터 데이터베이스 리소스를 보호합니다.



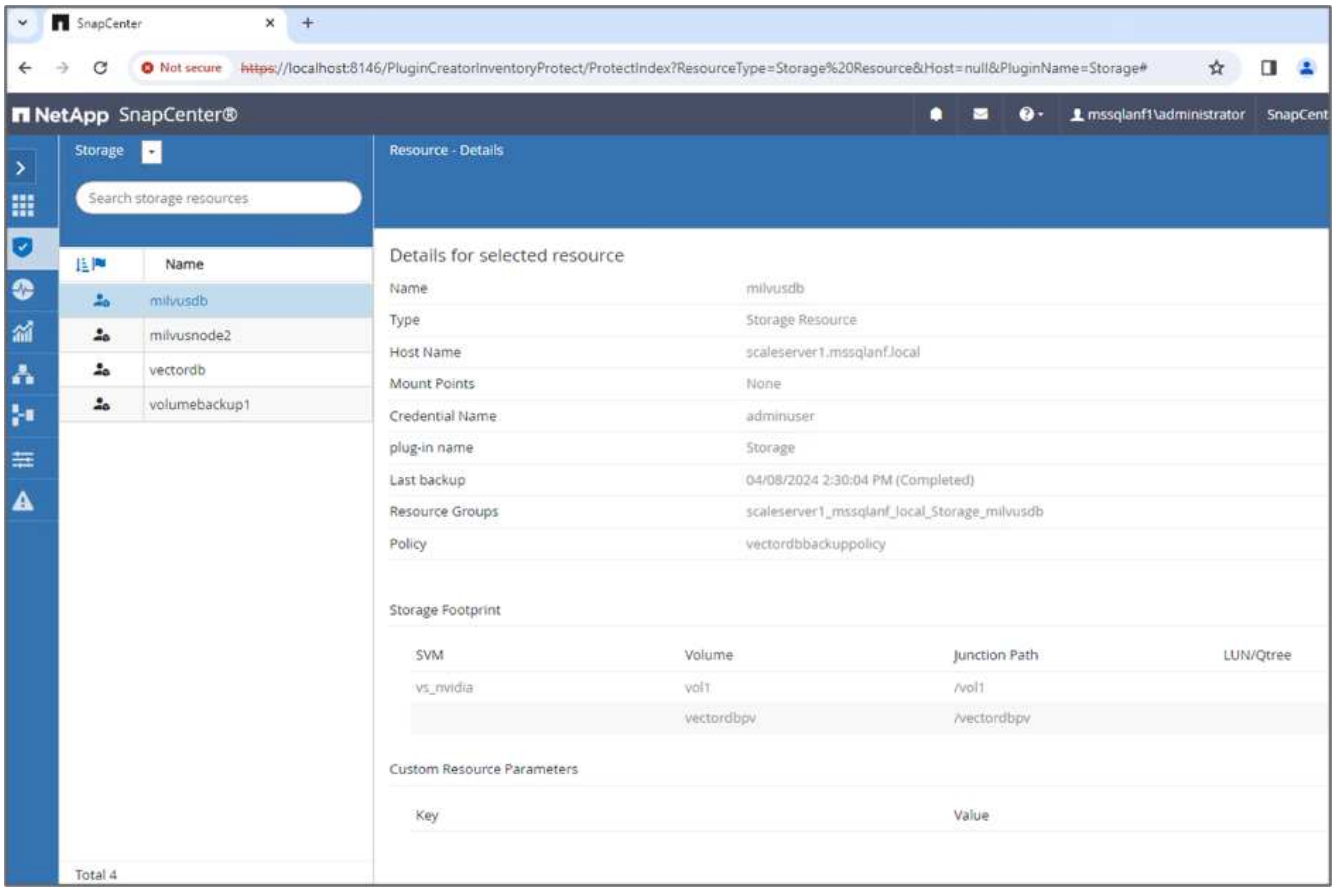
5. Python 스크립트를 사용하여 S3 NAS 버킷에 데이터를 삽입합니다. 우리의 경우 Milvus가 제공한 백업 스크립트인 'prepare_data_netapp.py'를 수정하고 'sync' 명령을 실행하여 운영 체제에서 데이터를 플러시했습니다.

```
root@node2:~# python3 prepare_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_test` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_test: 3000

=== Create collection `hello_milvus_netapp_sc_test2` ===

Number of entities in hello_milvus_netapp_sc_test2: 6000
root@node2:~# for i in 2 3 4 5 6   ; do ssh node$i "hostname; sync; echo
'sync executed';" ; done
node2
sync executed
node3
sync executed
node4
sync executed
node5
sync executed
node6
sync executed
root@node2:~#
```

6. S3 NAS 버킷의 데이터를 확인합니다. 예를 들어, 타임스탬프가 '2024-04-08 21:22'인 파일은 'prepare_data_netapp.py' 스크립트에 의해 생성되었습니다.

```
root@node2:~# aws s3 ls --profile ontaps3  s3://milvusdbvol1/
--recursive | grep '2024-04-08'

<output content removed to save page space>
2024-04-08 21:18:14       5656
stats_log/448950615991000809/448950615991000810/448950615991001854/100/1
2024-04-08 21:18:12       5654
stats_log/448950615991000809/448950615991000810/448950615991001854/100/4
48950615990800869
2024-04-08 21:18:17       5656
stats_log/448950615991000809/448950615991000810/448950615991001872/100/1
2024-04-08 21:18:15       5654
stats_log/448950615991000809/448950615991000810/448950615991001872/100/4
48950615990800876
2024-04-08 21:22:46       5625
stats_log/448950615991003377/448950615991003378/448950615991003385/100/1
2024-04-08 21:22:45       5623
stats_log/448950615991003377/448950615991003378/448950615991003385/100/4
48950615990800899
2024-04-08 21:22:49       5656
stats_log/448950615991003408/448950615991003409/448950615991003416/100/1
2024-04-08 21:22:47       5654
stats_log/448950615991003408/448950615991003409/448950615991003416/100/4
48950615990800906
2024-04-08 21:22:52       5656
stats_log/448950615991003408/448950615991003409/448950615991003434/100/1
2024-04-08 21:22:50       5654
stats_log/448950615991003408/448950615991003409/448950615991003434/100/4
48950615990800913
root@node2:~#
```

7. 'milvusdb' 리소스의 일관성 그룹(CG) 스냅샷을 사용하여 백업을 시작합니다.

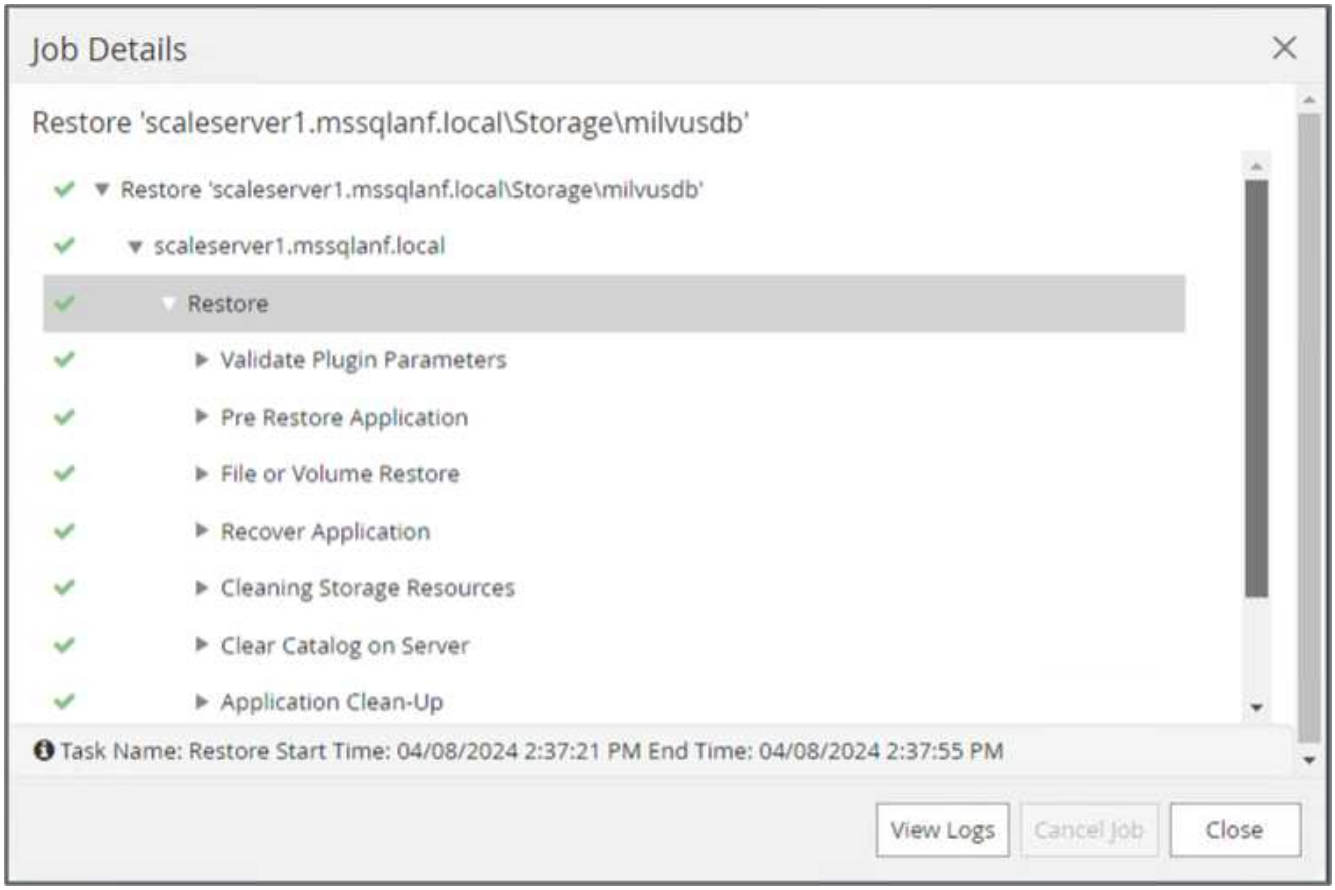


8. 백업 기능을 테스트하기 위해 백업 프로세스 후에 새 테이블을 추가하거나 NFS(S3 NAS 버킷)에서 일부 데이터를 제거했습니다.

   이 테스트에서는 백업 후 누군가가 새롭고 불필요하거나 부적절한 컬렉션을 만든 상황을 상상해 보세요. 이런 경우에는 새로운 컬렉션이 추가되기 전 상태로 벡터 데이터베이스를 되돌려야 합니다. 예를 들어, 'hello_milvus_netapp_sc_testnew' 및 'hello_milvus_netapp_sc_testnew2'와 같은 새로운 컬렉션이 삽입되었습니다.

```
root@node2:~# python3 prepare_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_testnew` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

Number of entities in hello_milvus_netapp_sc_testnew2: 6000
root@node2:~#
```

9. 이전 스냅샷에서 S3 NAS 버킷의 전체 복원을 실행합니다.



10. Python 스크립트를 사용하여 'hello_milvus_netapp_sc_test' 및 'hello_milvus_netapp_sc_test2' 컬렉션의 데이터를 확인합니다.

```
root@node2:~# python3 verify_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: True
{'auto_id': False, 'description': 'hello_milvus_netapp_sc_test',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5
>, 'is_primary': True, 'auto_id': False}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 8}}]}
Number of entities in Milvus: hello_milvus_netapp_sc_test : 3000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 1262, distance: 0.08883658051490784, entity: {'random':
0.2978858685751561}, random field: 0.2978858685751561
hit: id: 1265, distance: 0.09590047597885132, entity: {'random':
0.3042039939240304}, random field: 0.3042039939240304
hit: id: 2999, distance: 0.0, entity: {'random': 0.02316334456872482},
random field: 0.02316334456872482
hit: id: 1580, distance: 0.05628091096878052, entity: {'random':
0.3855988746044062}, random field: 0.3855988746044062
hit: id: 2377, distance: 0.08096685260534286, entity: {'random':
0.8745922204004368}, random field: 0.8745922204004368
search latency = 0.2832s

=== Start querying with `random > 0.5` ===

query result:
-{'random': 0.6378742006852851, 'embeddings': [0.20963514, 0.39746657,
```

```
0.12019053, 0.6947492, 0.9535575, 0.5454552, 0.82360446, 0.21096309],
'pk': 0}
search latency = 0.2257s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 747, distance: 0.14606499671936035, entity: {'random':
0.5648774800635661}, random field: 0.5648774800635661
hit: id: 2527, distance: 0.1530652642250061, entity: {'random':
0.8928974315571507}, random field: 0.8928974315571507
hit: id: 2377, distance: 0.08096685260534286, entity: {'random':
0.8745922204004368}, random field: 0.8745922204004368
hit: id: 2034, distance: 0.20354536175727844, entity: {'random':
0.5526117606328499}, random field: 0.5526117606328499
hit: id: 958, distance: 0.21908017992973328, entity: {'random':
0.6647383716417955}, random field: 0.6647383716417955
search latency = 0.5480s
Does collection hello_milvus_netapp_sc_test2 exist in Milvus: True
{'auto_id': True, 'description': 'hello_milvus_netapp_sc_test2',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5
>, 'is_primary': True, 'auto_id': True}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 8}}]}
Number of entities in Milvus: hello_milvus_netapp_sc_test2 : 6000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: id: 448950615990642008, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990645009, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990640618, distance: 0.13562293350696564, entity:
{'random': 0.7864676926688837}, random field: 0.7864676926688837
hit: id: 448950615990642314, distance: 0.10414951294660568, entity:
{'random': 0.2209597460821181}, random field: 0.2209597460821181
hit: id: 448950615990645315, distance: 0.10414951294660568, entity:
```

```
{'random': 0.2209597460821181}, random field: 0.2209597460821181
hit: id: 448950615990640004, distance: 0.11571306735277176, entity:
{'random': 0.7765521996186631}, random field: 0.7765521996186631
search latency = 0.2381s

=== Start querying with `random > 0.5` ===

query result:
-{'embeddings': [0.15983285, 0.72214717, 0.7414838, 0.44471496,
0.50356466, 0.8750043, 0.316556, 0.7871702], 'pk': 448950615990639798,
'random': 0.7820620141382767}
search latency = 0.3106s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 448950615990642008, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990645009, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990640618, distance: 0.13562293350696564, entity:
{'random': 0.7864676926688837}, random field: 0.7864676926688837
hit: id: 448950615990640004, distance: 0.11571306735277176, entity:
{'random': 0.7765521996186631}, random field: 0.7765521996186631
hit: id: 448950615990643005, distance: 0.11571306735277176, entity:
{'random': 0.7765521996186631}, random field: 0.7765521996186631
hit: id: 448950615990640402, distance: 0.13665105402469635, entity:
{'random': 0.9742541034109935}, random field: 0.9742541034109935
search latency = 0.4906s
root@node2:~#
```

11. 불필요하거나 부적절한 컬렉션이 더 이상 데이터베이스에 존재하지 않는지 확인하세요.

```
root@node2:~# python3 verify_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False
Traceback (most recent call last):
  File "/root/verify_data_netapp.py", line 37, in <module>
    recover_collection = Collection(recover_collection_name)
  File "/usr/local/lib/python3.10/dist-
packages/pymilvus/orm/collection.py", line 137, in __init__
    raise SchemaNotReadyException(
pymilvus.exceptions.SchemaNotReadyException: <SchemaNotReadyException:
(code=1, message=Collection 'hello_milvus_netapp_sc_testnew' not exist,
or you can pass in schema to create one.)>
root@node2:~#
```

결론적으로, ONTAP 에 있는 벡터 데이터베이스 데이터와 Milvus 데이터를 보호하기 위해 NetApp의 SnapCenter 사용하면 고객에게 상당한 이점을 제공하며, 특히 영화 제작과 같이 데이터 무결성이 가장 중요한 산업의 고객에게 큰 이점을 제공합니다. SnapCenter는 일관된 백업을 생성하고 전체 데이터 복원을 수행할 수 있는 기능을 통해 내장된 비디오 및 오디오 파일과 같은 중요한 데이터가 하드 드라이브 오류나 기타 문제로 인한 손실로부터 보호됩니다. 이는 운영 중단을 방지할 뿐만 아니라 상당한 재정적 손실로부터도 보호해줍니다.

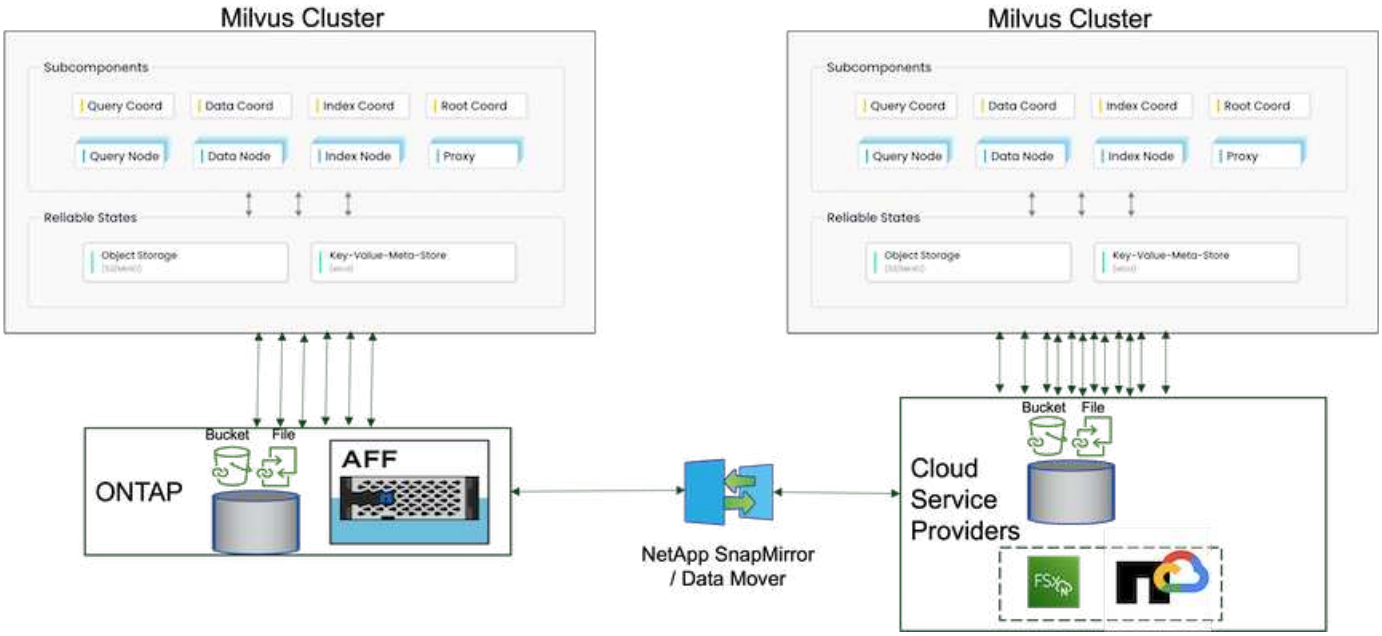
이 섹션에서는 호스트 설정, 스토리지 플러그인 설치 및 구성, 사용자 정의 스냅샷 이름을 사용하여 벡터 데이터베이스에 대한 리소스 생성을 포함하여 ONTAP 에 있는 데이터를 보호하기 위해 SnapCenter 구성하는 방법을 보여주었습니다. 또한 일관성 그룹 스냅샷을 사용하여 백업을 수행하고 S3 NAS 버킷의 데이터를 검증하는 방법도 보여드렸습니다.

더 나아가, 백업 후 불필요하거나 부적절한 컬렉션이 생성되는 시나리오를 시뮬레이션했습니다. 이런 경우 SnapCenter는 이전 스냅샷에서 전체 복원을 수행할 수 있는 기능을 통해 벡터 데이터베이스를 새 컬렉션을 추가하기 전 상태로 되돌릴 수 있으므로 데이터베이스의 무결성을 유지할 수 있습니다. 특정 시점으로 데이터를 복원하는 기능은 고객에게 매우 귀중하며, 데이터가 안전할 뿐만 아니라 올바르게 유지 관리되고 있다는 확신을 제공합니다. 따라서 NetApp의 SnapCenter 제품은 고객에게 데이터 보호 및 관리를 위한 견고하고 안정적인 솔루션을 제공합니다.

NetApp SnapMirror 사용한 재해 복구

이 섹션에서는 NetApp 의 벡터 데이터베이스 솔루션을 위한 SnapMirror 사용한 DR(재해 복구)에 대해 설명합니다.

NetApp SnapMirror 사용한 재해 복구



재해 복구는 벡터 데이터베이스의 무결성과 가용성을 유지하는 데 필수적이며, 특히 고차원 데이터를 관리하고 복잡한 유사성 검색을 실행하는 역할을 감안할 때 더욱 그렇습니다. 잘 계획하고 구현된 재해 복구 전략은 하드웨어 장애, 자연재해, 사이버 공격 등 예상치 못한 사고가 발생하더라도 데이터가 손실되거나 손상되지 않도록 보장합니다. 이는 벡터 데이터베이스에 의존하는 애플리케이션에 특히 중요한데, 데이터가 손실되거나 손상되면 상당한 운영 중단과 재정적 손실로 이어질 수 있기 때문입니다. 더욱이, 강력한 재해 복구 계획은 가동 중지 시간을 최소화하고 서비스를 신속하게 복구할 수 있게 하여 비즈니스 연속성을 보장합니다. 이는 다양한 지리적 위치에 걸친 NetApp 데이터 복제 제품 SnapMirrror, 정기적 백업 및 장애 조치 메커니즘을 통해 달성됩니다. 따라서 재해 복구는 단순한 보호 조치가 아니라 책임감 있고 효율적인 벡터 데이터베이스 관리의 중요한 구성 요소입니다.

NetApp의 SnapMirror 하나의 NetApp ONTAP 스토리지 컨트롤러에서 다른 컨트롤러로 데이터를 복제하는 기능을 제공하며, 주로 재해 복구(DR) 및 하이브리드 솔루션에 사용됩니다. 벡터 데이터베이스의 맥락에서 이 도구는 온프레미스와 클라우드 환경 간의 원활한 데이터 전환을 용이하게 합니다. 이러한 전환은 데이터 변환이나 애플리케이션 리팩토링이 필요 없이 이루어지므로 여러 플랫폼에서 데이터 관리의 효율성과 유연성이 향상됩니다.

벡터 데이터베이스 시나리오에서 NetApp 하이브리드 솔루션은 더 많은 이점을 가져올 수 있습니다.

1. 확장성: NetApp의 하이브리드 클라우드 솔루션은 요구 사항에 따라 리소스를 확장할 수 있는 기능을 제공합니다. 정기적이고 예측 가능한 작업 부하에는 온프레미스 리소스를 활용하고, 피크 시간이나 예상치 못한 부하에는 Amazon FSx ONTAP for NetApp ONTAP 및 Google Cloud NetApp Volume(NetApp Volumes)과 같은 클라우드 리소스를 활용할 수 있습니다.
2. 비용 효율성: NetApp의 하이브리드 클라우드 모델을 사용하면 정기적인 작업에는 온프레미스 리소스를 사용하고 필요할 때만 클라우드 리소스에 대한 비용을 지불하여 비용을 최적화할 수 있습니다. NetApp instaclustr 서비스를 이용하면 이러한 사용량 기반 결제 모델을 상당히 비용 효율적으로 활용할 수 있습니다. 온프레미스 및 주요 클라우드 서비스 공급업체를 위해 instaclustr는 지원과 컨설팅을 제공합니다.
3. 유연성: NetApp의 하이브리드 클라우드는 데이터를 처리할 위치를 선택할 수 있는 유연성을 제공합니다. 예를 들어, 더 강력한 하드웨어를 갖춘 온프레미스에서 복잡한 벡터 작업을 수행하고, 덜 집약적인 작업은 클라우드에서 수행하기로 선택할 수 있습니다.
4. 비즈니스 연속성: 재해 발생 시 NetApp 하이브리드 클라우드에 데이터를 보관하면 비즈니스 연속성을 보장할 수 있습니다. 온프레미스 리소스에 영향을 미치는 경우 신속하게 클라우드로 전환할 수 있습니다. NetApp SnapMirror 활용하면 온프레미스에서 클라우드로 데이터를 이동할 수 있으며, 그 반대의 경우도 가능합니다.
5. 혁신: NetApp의 하이브리드 클라우드 솔루션은 최첨단 클라우드 서비스와 기술에 대한 액세스를 제공함으로써 보다 빠른 혁신을 가능하게 합니다. Amazon FSx ONTAP for NetApp ONTAP, Azure NetApp Files , Google

Cloud NetApp Volumes 와 같은 클라우드의 NetApp 혁신은 클라우드 서비스 제공업체의 혁신적인 제품이자 선호되는 NAS입니다.

벡터 데이터베이스 성능 검증

이 섹션에서는 벡터 데이터베이스에서 수행된 성능 검증을 강조합니다.

성능 검증

성능 검증은 벡터 데이터베이스와 저장 시스템 모두에서 중요한 역할을 하며, 최적의 운영과 효율적인 리소스 활용을 보장하는 핵심 요소로 작용합니다. 고차원 데이터를 처리하고 유사성 검색을 실행하는 것으로 알려진 벡터 데이터베이스는 복잡한 쿼리를 빠르고 정확하게 처리하기 위해 높은 성능 수준을 유지해야 합니다. 성능 검증은 병목 현상을 파악하고, 구성을 미세하게 조정하고, 시스템이 서비스 저하 없이 예상되는 부하를 처리할 수 있는지 확인하는 데 도움이 됩니다. 마찬가지로 저장 시스템에서도 성능 검증은 데이터가 효율적으로 저장되고 검색되며, 전반적인 시스템 성능에 영향을 줄 수 있는 지연 문제나 병목 현상이 발생하지 않도록 하는 데 필수적입니다. 또한, 스토리지 인프라의 필요한 업그레이드나 변경에 대한 정보에 입각한 결정을 내리는 데 도움이 됩니다. 따라서 성능 검증은 시스템 관리의 중요한 측면이며, 높은 서비스 품질, 운영 효율성, 전반적인 시스템 안정성을 유지하는 데 크게 기여합니다.

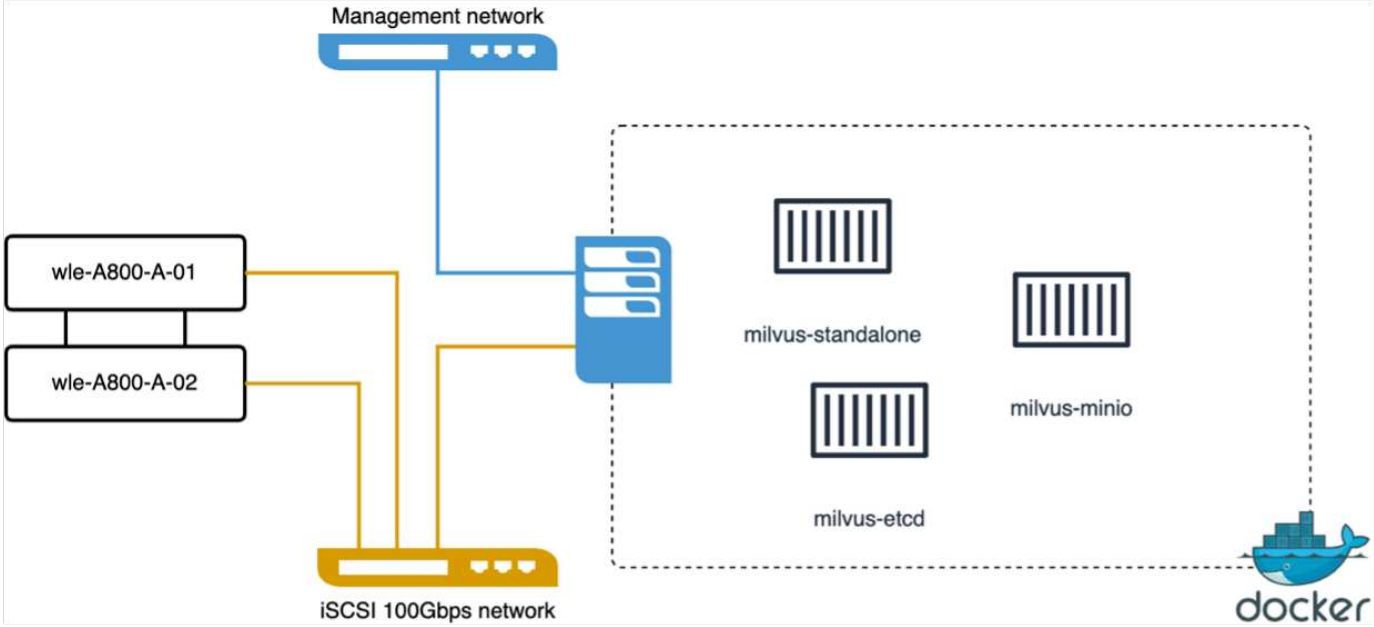
이 섹션에서는 Milvus 및 pgvecto.rs와 같은 벡터 데이터베이스의 성능 검증을 자세히 살펴보고, LLM 수명 주기 내에서 RAG 및 추론 워크로드를 지원하는 I/O 프로필 및 netapp 스토리지 컨트롤러 동작과 같은 스토리지 성능 특성에 중점을 둡니다. 이러한 데이터베이스를 ONTAP 스토리지 솔루션과 결합하면 성능상의 차이점을 평가하고 파악할 것입니다. 당사의 분석은 초당 처리되는 쿼리 수(QPS)와 같은 핵심 성과 지표를 기반으로 진행됩니다.

아래에서 밀부스와 진행 상황에 사용된 방법론을 확인하세요.

세부	Milvus(독립형 및 클러스터)	Postgres(pgvecto.rs) #
버전	2.3.2	0.2.0
파일 시스템	iSCSI LUN의 XFS	
워크로드 생성기	"VectorDB-벤치"– v0.0.5	
데이터 세트	LAION 데이터 세트 * 1,000만 개의 임베딩 * 768개의 차원 * ~300GB 데이터 세트 크기	
스토리지 컨트롤러	AFF 800 * 버전 – 9.14.1 * 4 x 100GbE – milvus용 및 2 x 100GbE – postgres용 * iscsi	

Milvus 독립형 클러스터가 포함된 **VectorDB-Bench**

우리는 vectorDB-Bench를 사용하여 milvus 독립형 클러스터에서 다음과 같은 성능 검증을 수행했습니다. milvus 독립형 클러스터의 네트워크 및 서버 연결성은 아래와 같습니다.



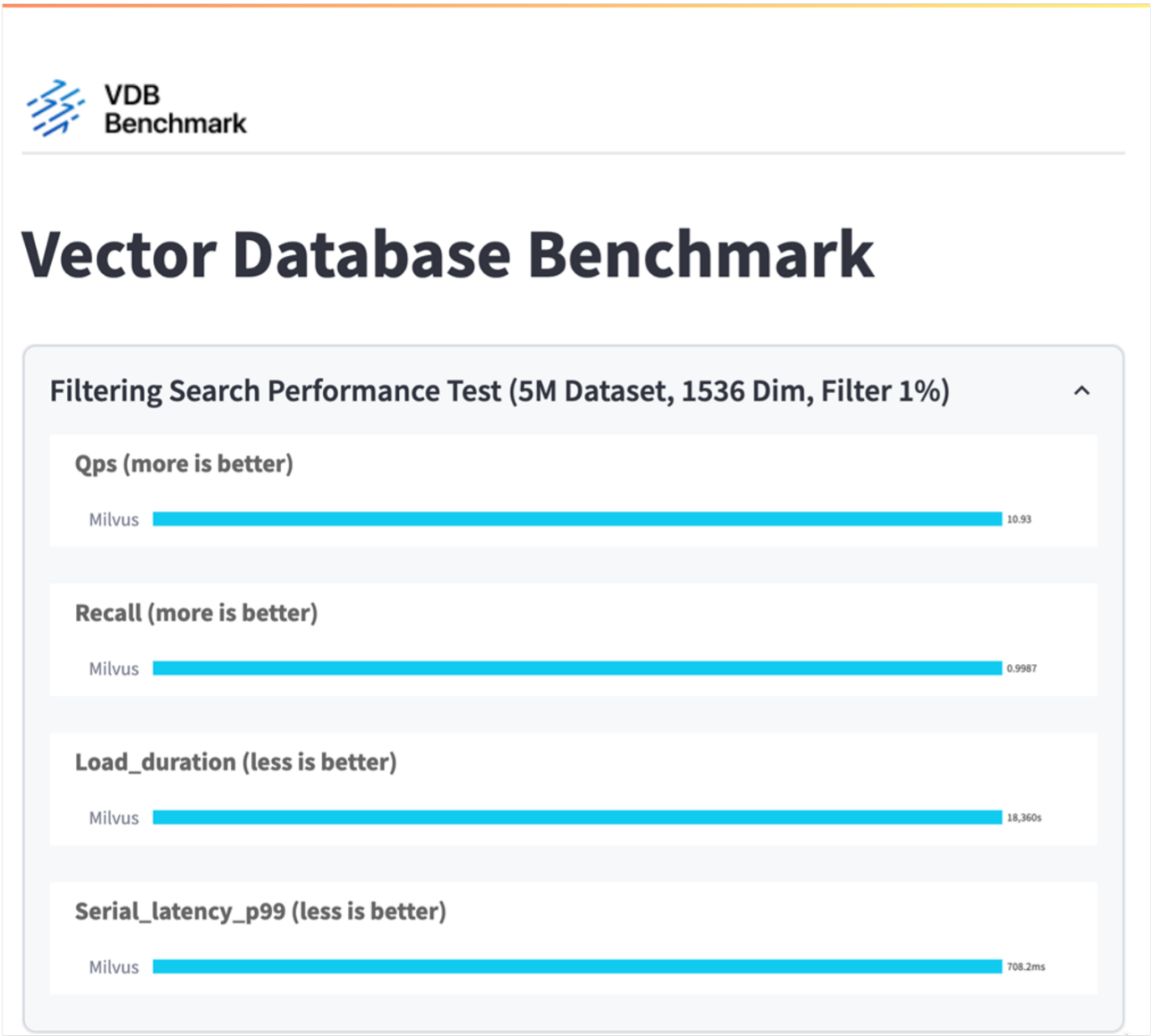
이 섹션에서는 Milvus 독립형 데이터베이스를 테스트하여 얻은 관찰 결과와 결과를 공유합니다. . 우리는 이러한 테스트의 인덱스 유형으로 DiskANN을 선택했습니다. . 약 100GB 규모의 데이터 세트를 수집, 최적화하고 인덱스를 생성하는 데 약 5시간이 걸렸습니다. 이 기간의 대부분 동안, 하이퍼스레딩이 활성화된 경우 40개의 vcpu에 해당하는 20개의 코어를 갖춘 Milvus 서버는 최대 CPU 용량인 100%로 작동했습니다. DiskANN은 시스템 메모리 크기를 초과하는 대규모 데이터 세트에 특히 중요하다는 것을 발견했습니다. . 쿼리 단계에서 우리는 0.9987의 회수율로 초당 쿼리 수(QPS) 10.93을 관찰했습니다. 쿼리에 대한 99번째 백분위수 지연 시간은 708.2밀리초로 측정되었습니다.

저장 관점에서 볼 때, 데이터베이스는 수집, 삽입 후 최적화, 인덱스 생성 단계에서 초당 약 1,000개의 작업을 실행했습니다. 쿼리 단계에서는 초당 32,000개의 작업이 요구되었습니다.

다음 섹션에서는 스토리지 성능 측정 항목을 소개합니다.

작업 부하 단계	미터법	가치
데이터 수집 및 삽입 후 최적화	아이옵스	< 1,000
	숨어 있음	< 400유초
	작업량	읽기/쓰기 혼합, 주로 쓰기
	IO 크기	64KB
질문	아이옵스	최고 32,000
	숨어 있음	< 400유초
	작업량	100% 캐시된 읽기
	IO 크기	대부분 8KB

vectorDB-bench 결과는 아래와 같습니다.

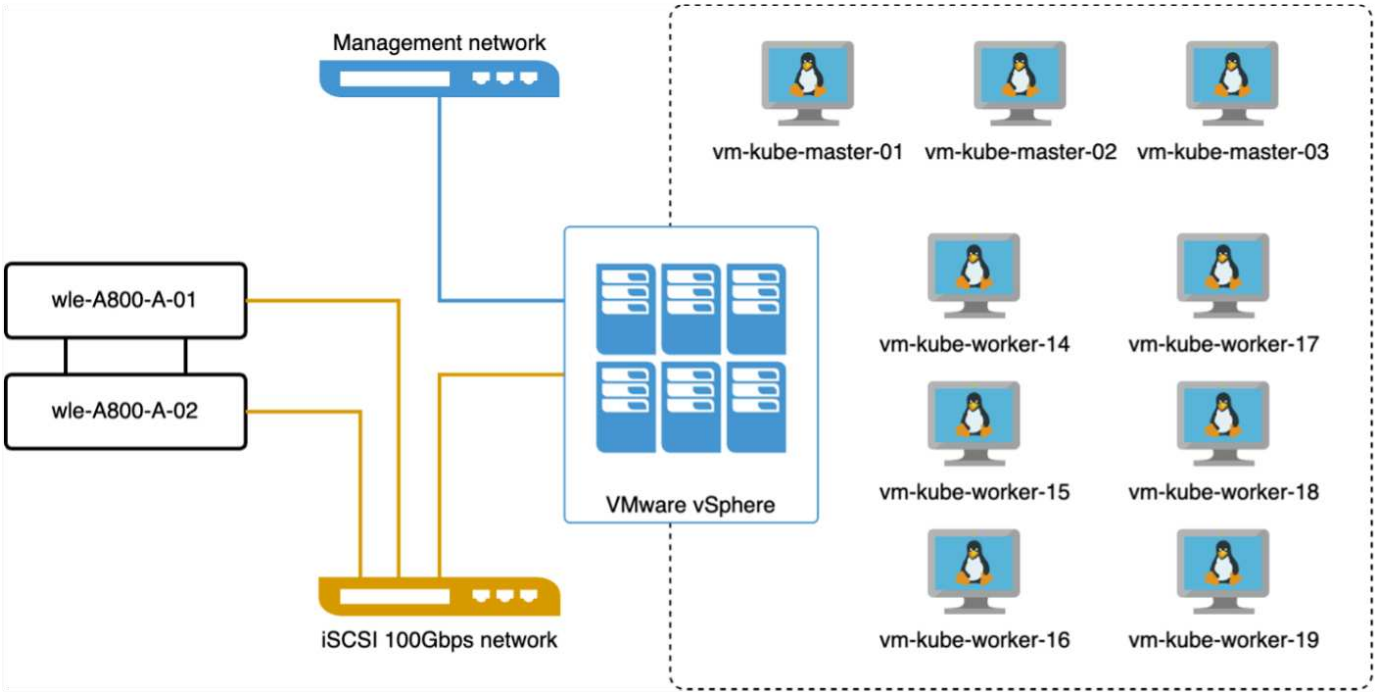


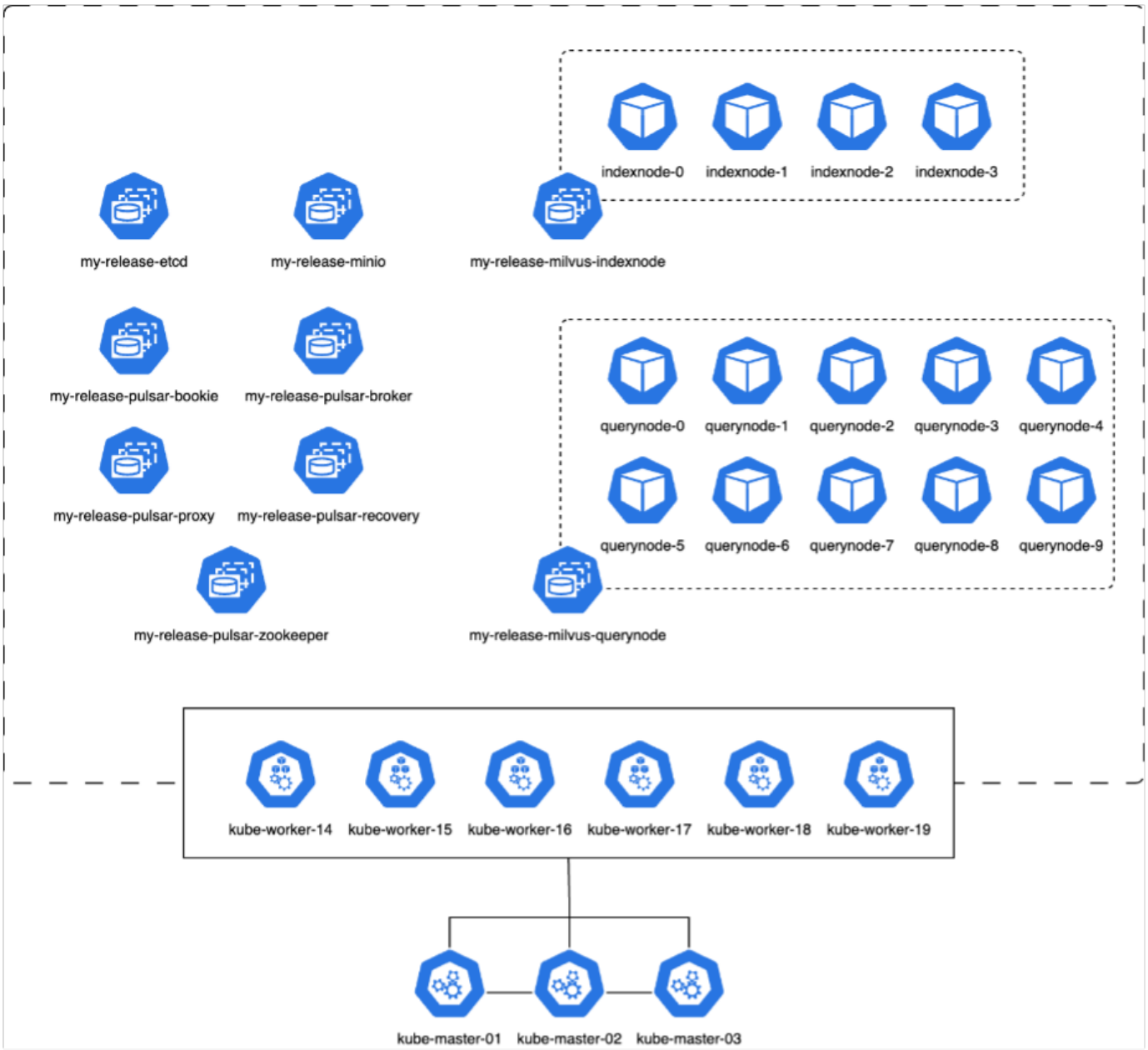
독립형 Milvus 인스턴스의 성능 검증을 통해, 현재 설정은 1536차원의 500만 개 벡터 데이터 세트를 지원하기에 부족하다는 것이 분명해졌습니다. 저장소에 충분한 리소스가 있으며 시스템에 병목 현상을 일으키지 않는다는 것을 확인했습니다.

Milvus 클러스터가 포함된 **VectorDB-Bench**

이 섹션에서는 Kubernetes 환경 내에서 Milvus 클러스터를 배포하는 방법에 대해 설명합니다. 이 Kubernetes 설정은 Kubernetes 마스터 및 워커 노드를 호스팅하는 VMware vSphere 배포를 기반으로 구성되었습니다.

다음 섹션에서는 VMware vSphere 및 Kubernetes 배포에 대한 세부 정보를 제공합니다.


Management network
vm-kube-master-01
vm-kube-master-02
vm-kube-master-03
wle-A800-A-01
wle-A800-A-02
VMware vSphere
vm-kube-worker-14
vm-kube-worker-17
vm-kube-worker-15
vm-kube-worker-18
vm-kube-worker-16
vm-kube-worker-19
iSCSI 100Gbps network



이 섹션에서는 Milvus 데이터베이스 테스트에서 얻은 관찰 결과와 결과를 제시합니다. * 사용된 인덱스 유형은 DiskANN입니다. * 아래 표는 1536차원에서 500만 개의 벡터로 작업할 때 독립 실행형 배포와 클러스터 배포를 비교한 것입니다. 클러스터 배포에서는 데이터 수집과 삽입 후 최적화에 걸리는 시간이 더 짧은 것을 확인했습니다. 클러스터 배포에서는 독립 실행형 설정에 비해 쿼리의 99번째 백분위수 지연 시간이 6배나 단축되었습니다. * 클러스터 배포 시 초당 쿼리 수(QPS)는 더 높았지만 원하는 수준은 아니었습니다.

Metric	Milvus Standalone	Milvus Cluster	Difference
QPS @ Recall	10.93 @ 0.9987	18.42 @ 0.9952	+40%
p99 Latency (less is better)	708.2 ms	117.6 ms	-83%
Load Duration time (less is better)	18,360 secs	12,730 secs	-30%

아래 이미지는 스토리지 클러스터 지연 시간과 총 IOPS(초당 입출력 작업)를 포함한 다양한 스토리지 측정 항목을 보여줍니다.

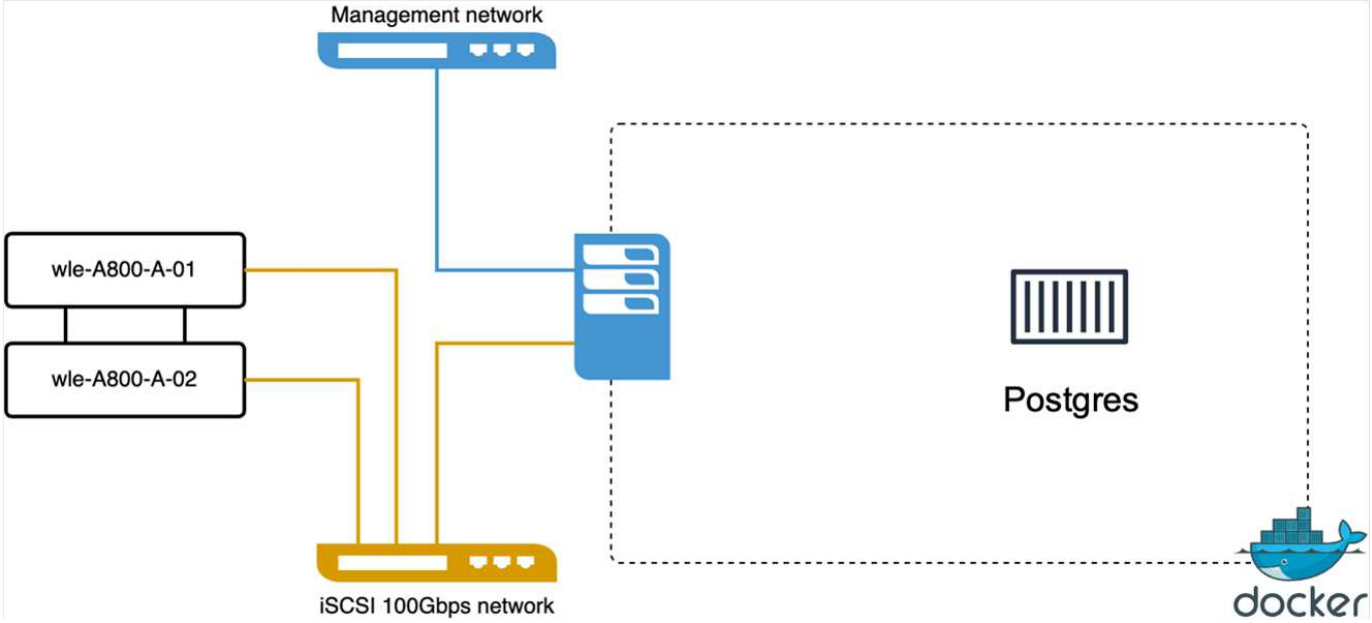
다음 섹션에서는 주요 스토리지 성능 지표를 소개합니다.

작업 부하 단계	미터법	가치
데이터 수집 및 삽입 후 최적화	아이옵스	< 1,000
	숨어 있음	< 400유초
	작업량	읽기/쓰기 혼합, 주로 쓰기
	IO 크기	64KB
질문	아이옵스	최고치 147,000
	숨어 있음	< 400유초
	작업량	100% 캐시된 읽기
	IO 크기	대부분 8KB

독립형 Milvus와 Milvus 클러스터의 성능 검증을 바탕으로 스토리지 I/O 프로필에 대한 세부 정보를 제시합니다. * 독립 실행형과 클러스터 배포 모두에서 I/O 프로필이 일관되게 유지되는 것을 확인했습니다. * 관찰된 최대 IOPS의 차이는 클러스터 배포에 포함된 클라이언트 수가 더 많은 데 기인할 수 있습니다.

Postgres를 사용한 **vectorDB-Bench(pgvecto.rs)**

VectorDB-Bench를 사용하여 PostgreSQL(pgvecto.rs)에서 다음 작업을 수행했습니다. PostgreSQL(특히 pgvecto.rs)의 네트워크 및 서버 연결에 대한 세부 정보는 다음과 같습니다.

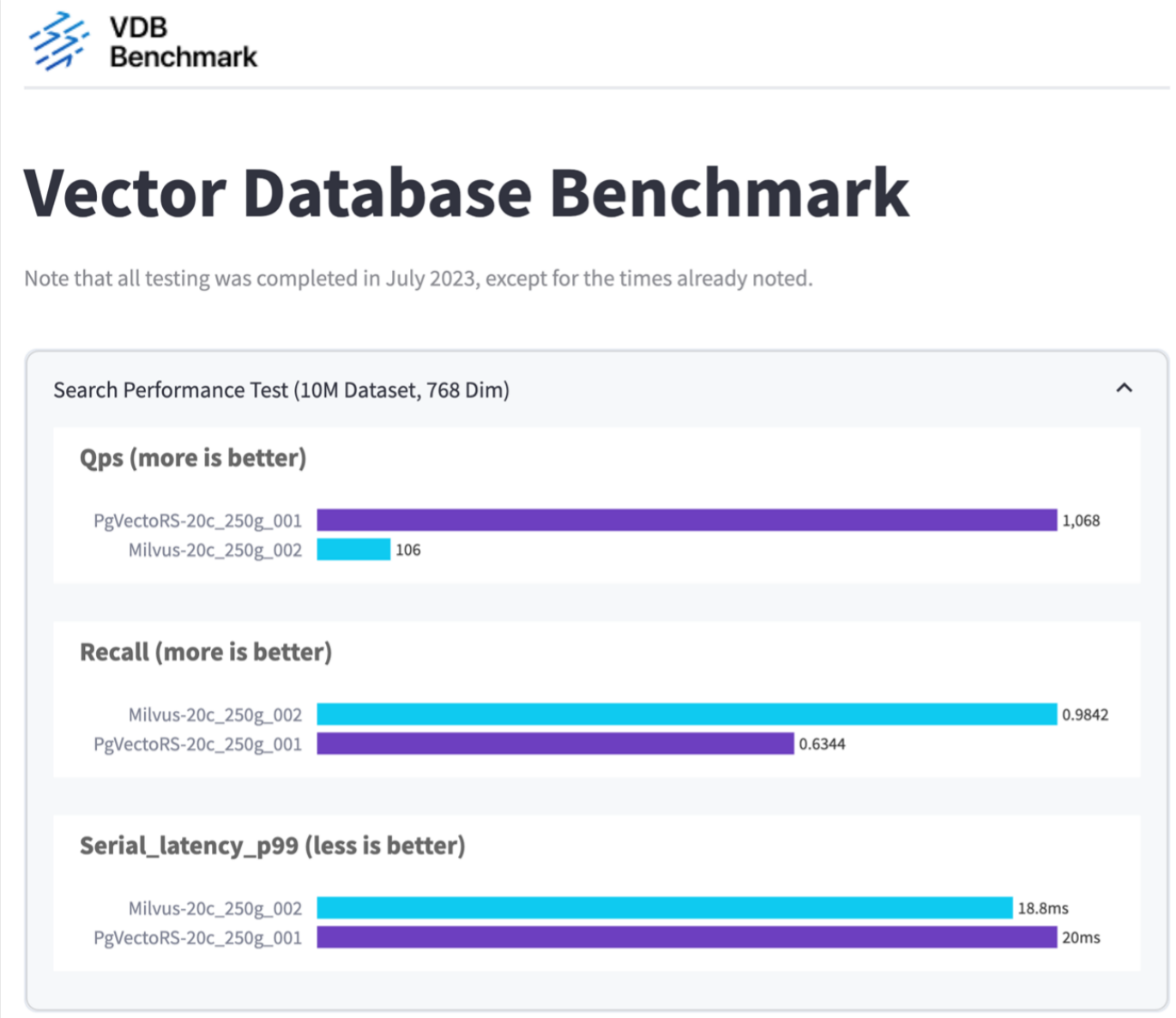


이 섹션에서는 pgvecto.rs를 사용하여 PostgreSQL 데이터베이스를 테스트한 결과와 관찰 내용을 공유합니다. * 테스트 당시 pgvecto.rs에서 DiskANN을 사용할 수 없었기 때문에 이러한 테스트의 인덱스 유형으로 HNSW를 선택했습니다. * 데이터 수집 단계에서는 768차원의 1,000만 개의 벡터로 구성된 Cohere 데이터 세트를 로드했습니다. 이 과정은 약 4.5시간이 걸렸습니다. * 쿼리 단계에서 우리는 0.6344의 회수율로 초당 쿼리 수(QPS) 1,068을 관찰했습니다. 쿼리에 대한 99번째 백분위수 지연 시간은 20밀리초로 측정되었습니다. 대부분의 런타임 동안 클라이언트 CPU는 100% 용량으로 작동했습니다.

아래 이미지는 스토리지 클러스터 지연 총 IOPS(초당 입출력 작업)를 포함한 다양한 스토리지 측정 항목을 보여줍니다.

```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-metrics.png["입력/출력 대화 상자 또는 서면 내용을
나타내는 그림"]
```

벡터 **DB** 벤치에서 **milvus**와 **postgres**의 성능 비교



VectorDBBench를 사용하여 Milvus와 PostgreSQL의 성능을 검증한 결과, 다음과 같은 결과가 관찰되었습니다.

- 인덱스 유형: HNSW
- 데이터 세트: 768차원의 1,000만 개 벡터로 구성된 Cohere

pgvecto.rs는 0.6344의 재현율로 초당 쿼리 수(QPS) 1,068을 달성한 반면, Milvus는 0.9842의 재현율로 초당 쿼리 수 106을 달성한 것으로 나타났습니다.

질의의 정확도를 높이는 것이 우선순위라면 Milvus는 pgvecto.rs보다 우수한 성능을 보이는데, 이는 질의당 관련 항목의 비율이 더 높기 때문입니다. 하지만 초당 쿼리 수가 더 중요한 요소라면 pgvecto.rs가 Milvus를 능가합니다. 하지만 pgvecto.rs를 통해 검색된 데이터의 품질이 낮다는 점을 알아두는 것이 중요합니다. 검색 결과의 약 37%가 관련성이 없는 항목입니다.

성과 검증을 기반으로 한 관찰 결과:

성과 검증을 바탕으로 다음과 같은 관찰 결과를 얻었습니다.

Milvus에서 I/O 프로필은 Oracle SLOB에서 볼 수 있는 것과 같은 OLTP 작업 부하와 매우 유사합니다. 벤치마크는

데이터 수집, 사후 최적화, 쿼리의 세 단계로 구성됩니다. 초기 단계는 주로 64KB 쓰기 작업을 특징으로 하는 반면, 쿼리 단계는 주로 8KB 읽기 작업을 포함합니다. ONTAP 이 Milvus I/O 부하를 효율적으로 처리할 것으로 기대합니다.

PostgreSQL I/O 프로필은 까다로운 스토리지 작업 부하를 나타내지 않습니다. 현재 진행 중인 메모리 내 구현을 고려하면 쿼리 단계에서 디스크 I/O가 관찰되지 않았습니다.

DiskANN은 스토리지 차별화를 위한 핵심 기술로 부상하고 있습니다. 시스템 메모리 경계를 넘어 벡터 DB 검색을 효율적으로 확장할 수 있습니다. 그러나 HNSW와 같은 메모리 내 벡터 DB 인덱스를 사용하여 스토리지 성능 차별화를 확립하기는 어려울 것입니다.

또한 인덱스 유형이 HSNW인 경우 쿼리 단계에서 저장소가 중요한 역할을 하지 않는다는 점도 주목할 만합니다. HSNW는 RAG 애플리케이션을 지원하는 벡터 데이터베이스의 가장 중요한 운영 단계입니다. 여기서 의미하는 바는 저장 성능이 이러한 애플리케이션의 전반적인 성능에 큰 영향을 미치지 않는다는 것입니다.

PostgreSQL을 사용하는 **Instaclustr**의 벡터 데이터베이스**: pgvector**

이 섹션에서는 instaclustr 제품이 NetApp 의 벡터 데이터베이스 솔루션에서 pgvector 기능과 postgreSQL을 통합하는 방법에 대한 세부 사항을 설명합니다.

PostgreSQL을 사용하는 **Instaclustr**의 벡터 데이터베이스**: pgvector**

이 섹션에서는 instaclustr 제품이 pgvector 기능을 통해 postgreSQL과 어떻게 통합되는지 자세히 살펴보겠습니다. "PGVector와 PostgreSQL을 사용하여 LLM 정확도와 성능을 개선하는 방법: 임베딩 소개 및 PGVector의 역할"에 대한 예가 있습니다. 확인해 주세요"블로그" 더 많은 정보를 얻으려면.

벡터 데이터베이스 사용 사례

이 섹션에서는 NetApp 벡터 데이터베이스 솔루션의 사용 사례에 대한 개요를 제공합니다.

벡터 데이터베이스 사용 사례

이 섹션에서는 대규모 언어 모델을 사용한 검색 증강 생성 및 NetApp IT 챗봇과 같은 두 가지 사용 사례에 대해 설명합니다.

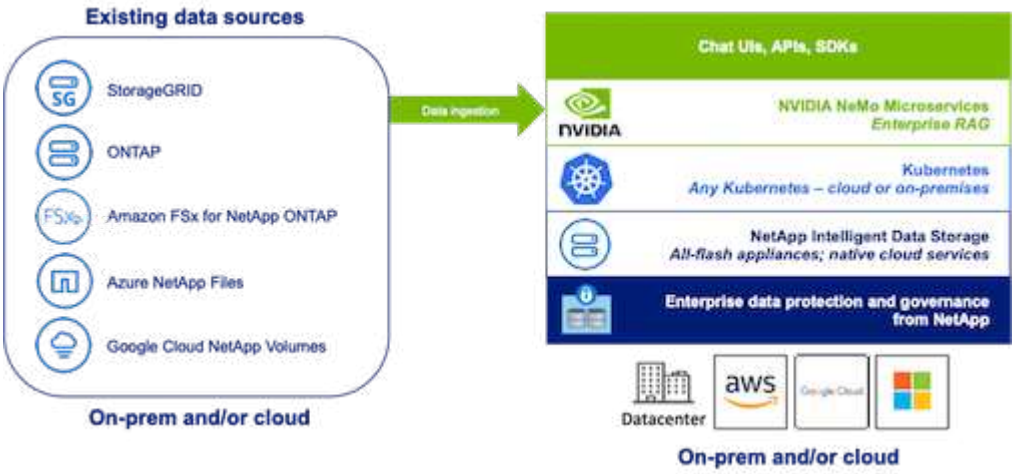
대규모 언어 모델**(LLM)**을 사용한 검색 증강 생성**(RAG)**

```
Retrieval-augmented generation, or RAG, is a technique for enhancing the
accuracy and reliability of Large Language Models, or LLMs, by augmenting
prompts with facts fetched from external sources. In a traditional RAG
deployment, vector embeddings are generated from an existing dataset and
then stored in a vector database, often referred to as a knowledgebase.
Whenever a user submits a prompt to the LLM, a vector embedding
representation of the prompt is generated, and the vector database is
searched using that embedding as the search query. This search operation
returns similar vectors from the knowledgebase, which are then fed to the
LLM as context alongside the original user prompt. In this way, an LLM can
be augmented with additional information that was not part of its original
training dataset.
```

NVIDIA Enterprise RAG LLM Operator는 기업에서 RAG를 구현하는 데 유용한 도구입니다. 이 연산자는 전체 RAG 파이프라인을 배포하는 데 사용할 수 있습니다. RAG 파이프라인은 지식 기반 임베딩을 저장하기 위한 벡터 데이터베이스로 Milvus나 pgvecto를 활용하도록 사용자 정의할 수 있습니다. 자세한 내용은 설명서를 참조하세요.

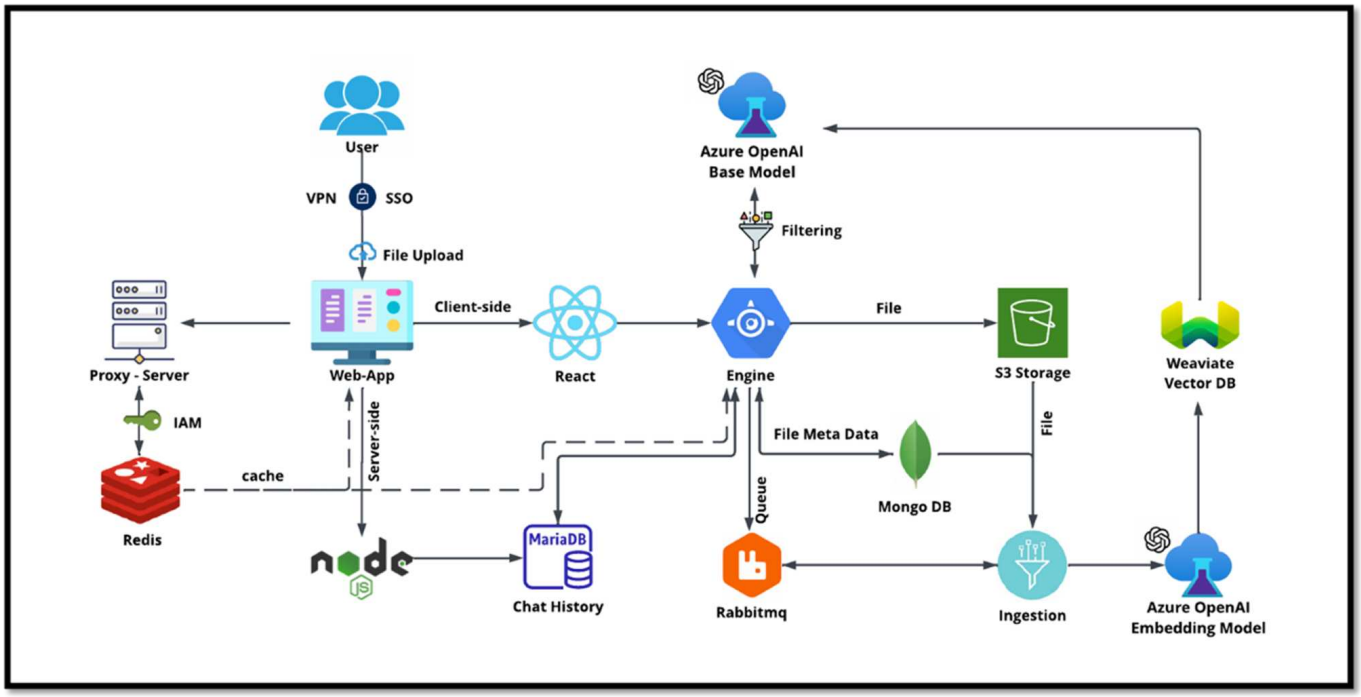
```
NetApp has validated an enterprise RAG architecture powered by the NVIDIA
Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog
post for more information and to see a demo. Figure 1 provides an overview
of this architecture.
```

그림 1) NVIDIA NeMo 마이크로서비스와 NetApp 기반 엔터프라이즈 RAG



NetApp IT 챗봇 사용 사례

NetApp의 챗봇은 벡터 데이터베이스의 또 다른 실시간 사용 사례로 활용됩니다. 이 경우 NetApp Private OpenAI Sandbox는 NetApp 내부 사용자의 쿼리를 관리하기 위한 효과적이고 안전하며 효율적인 플랫폼을 제공합니다. 엄격한 보안 프로토콜, 효율적인 데이터 관리 시스템, 정교한 AI 처리 기능을 통합하여 SSO 인증을 통해 조직 내 사용자의 역할과 책임에 따라 고품질의 정확한 응답을 보장합니다. 이 아키텍처는 고급 기술을 통합하여 사용자 중심의 지능형 시스템을 만드는 잠재력을 강조합니다.



사용 사례는 네 가지 주요 섹션으로 나눌 수 있습니다.

사용자 인증 및 검증:

- 사용자 쿼리는 먼저 NetApp Single Sign-On(SSO) 프로세스를 거쳐 사용자의 신원을 확인합니다.
- 인증이 성공하면 시스템은 VPN 연결을 확인하여 안전한 데이터 전송이 이루어지는지 확인합니다.

데이터 전송 및 처리:

- VPN이 검증되면 데이터는 NetAIChat 또는 NetAICreate 웹 애플리케이션을 통해 MariaDB로 전송됩니다. MariaDB는 사용자 데이터를 관리하고 저장하는 데 사용되는 빠르고 효율적인 데이터베이스 시스템입니다.
- 그런 다음 MariaDB는 해당 정보를 NetApp Azure 인스턴스로 전송하고, 이를 통해 사용자 데이터가 AI 처리 장치에 연결됩니다.

OpenAI 및 콘텐츠 필터링과의 상호 작용:

- Azure 인스턴스는 사용자의 질문을 콘텐츠 필터링 시스템으로 전송합니다. 이 시스템은 쿼리를 정리하고 처리를 위해 준비합니다.
- 정리된 입력은 Azure OpenAI 기본 모델로 전송되고, 이 모델에서는 입력을 기반으로 응답을 생성합니다.

응답 생성 및 조정:

- 기본 모델의 응답은 먼저 정확하고 콘텐츠 표준을 충족하는지 확인하기 위해 검사됩니다.
- 검사를 통과하면 응답이 사용자에게 다시 전송됩니다. 이 과정을 통해 사용자는 자신의 질문에 대해 명확하고 정확하며 적절한 답변을 받을 수 있습니다.

결론

이 섹션에서는 NetApp 의 벡터 데이터베이스 솔루션을 마무리합니다.

결론

결론적으로, 이 문서는 NetApp 스토리지 솔루션에서 Milvus 및 pgvector와 같은 벡터 데이터베이스를 배포하고 관리하는 방법에 대한 포괄적인 개요를 제공합니다. NetApp ONTAP 및 StorageGRID 객체 스토리지를 활용하기 위한 인프라 가이드라인을 논의하고 파일 및 객체 저장소를 통해 AWS FSx ONTAP 에서 Milvus 데이터베이스를 검증했습니다.

우리는 NetApp의 파일-객체 이중성을 살펴보고 벡터 데이터베이스의 데이터뿐만 아니라 다른 애플리케이션에도 유용하다는 것을 보여주었습니다. 또한 NetApp의 엔터프라이즈 관리 제품인 SnapCenter 벡터 데이터베이스 데이터에 대한 백업, 복원 및 복제 기능을 제공하여 데이터 무결성과 가용성을 보장하는 방식도 강조했습니다.

또한 이 문서에서는 NetApp의 하이브리드 클라우드 솔루션이 온프레미스 및 클라우드 환경에서 데이터 복제 및 보호를 제공하여 원활하고 안전한 데이터 관리 환경을 제공하는 방법을 자세히 설명합니다. 우리는 NetApp ONTAP 에서 Milvus와 pgvecto와 같은 벡터 데이터베이스의 성능 검증에 대한 통찰력을 제공하고, 이러한 데이터베이스의 효율성과 확장성에 대한 귀중한 정보를 제공했습니다.

마지막으로 LLM을 활용한 RAG와 NetApp의 내부 ChatAI라는 두 가지 생성 AI 사용 사례에 대해 논의했습니다. 이러한 실제 사례는 이 문서에 설명된 개념과 관행의 실제 적용과 이점을 강조합니다. 전반적으로 이 문서는 벡터 데이터베이스를 관리하기 위해 NetApp의 강력한 스토리지 솔루션을 활용하고자 하는 모든 사람을 위한 포괄적인 가이드 역할을 합니다.

감사의 말

저자는 아래의 기여자들과 NetApp 고객과 NetApp 분야에 귀중한 이 논문을 만들기 위해 피드백과 의견을 제공해주신 분들께 진심으로 감사드리고 싶습니다.

1. Sathish Thyagarajan, NetApp ONTAP AI 및 분석 기술 마케팅 엔지니어
2. Mike Oglesby, NetApp 기술 마케팅 엔지니어
3. AJ Mahajan, NetApp 수석 이사
4. Joe Scott, NetApp 워크로드 성능 엔지니어링 관리자
5. Puneet Dhawan, NetApp Fsx 제품 관리 부문 수석 이사
6. Yuval Kalderon, NetApp FSx 제품 팀 수석 제품 관리자

추가 정보를 찾을 수 있는 곳

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

- Milvus 문서 - https://milvus.io/docs/overview.md
- Milvus 독립형 문서 - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- NetApp 제품 문서https://www.netapp.com/support-and-training/documentation/[]
- 인스타클러스트 -"설치 클러스터 문서"

버전 기록

버전	날짜	문서 버전 기록
버전 1.0	2024년 4월	최초 출시

부록 A: Values.yaml

이 섹션에서는 NetApp 벡터 데이터베이스 솔루션에서 사용되는 값에 대한 샘플 YAML 코드를 제공합니다.

부록 A: Values.yaml

```
root@node2:~# cat  values.yaml
## Enable or disable Milvus Cluster mode
cluster:
  enabled: true

image:
  all:
    repository: milvusdb/milvus
    tag: v2.3.4
    pullPolicy: IfNotPresent
    ## Optionally specify an array of imagePullSecrets.
    ## Secrets must be manually created in the namespace.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/pull-
image-private-registry/
    ##
    # pullSecrets:
    #   - myRegistryKeySecretName
  tools:
    repository: milvusdb/milvus-config-tool
    tag: v0.1.2
    pullPolicy: IfNotPresent

# Global node selector
# If set, this will apply to all milvus components
# Individual components can be set to a different node selector
nodeSelector: {}

# Global tolerations
# If set, this will apply to all milvus components
# Individual components can be set to a different tolerations
tolerations: []

# Global affinity
# If set, this will apply to all milvus components
# Individual components can be set to a different affinity
affinity: {}

# Global labels and annotations
```

```yaml
# If set, this will apply to all milvus components
labels: {}
annotations: {}

# Extra configs for milvus.yaml
# If set, this config will merge into milvus.yaml
# Please follow the config structure in the milvus.yaml
# at https://github.com/milvus-io/milvus/blob/master/configs/milvus.yaml
# Note: this config will be the top priority which will override the
config
# in the image and helm chart.
extraConfigFiles:
  user.yaml: |+
    #    For example enable rest http for milvus proxy
    #    proxy:
    #      http:
    #        enabled: true
    ##  Enable tlsMode and set the tls cert and key
    #  tls:
    #    serverPemPath: /etc/milvus/certs/tls.crt
    #    serverKeyPath: /etc/milvus/certs/tls.key
    #   common:
    #     security:
    #       tlsMode: 1

## Expose the Milvus service to be accessed from outside the cluster
(LoadBalancer service).
## or access it from within the cluster (ClusterIP service). Set the
service type and the port to serve it.
## ref: http://kubernetes.io/docs/user-guide/services/
##
service:
  type: ClusterIP
  port: 19530
  portName: milvus
  nodePort: ""
  annotations: {}
  labels: {}

  ## List of IP addresses at which the Milvus service is available
  ## Ref: https://kubernetes.io/docs/user-guide/services/#external-ips
  ##
  externalIPs: []
  #   - externalIp1

  # LoadBalancerSourcesRange is a list of allowed CIDR values, which are
```

```
combined with ServicePort to
  # set allowed inbound rules on the security group assigned to the master
load balancer
  loadBalancerSourceRanges:
  - 0.0.0.0/0
  # Optionally assign a known public LB IP
  # loadBalancerIP: 1.2.3.4

ingress:
  enabled: false
  annotations:
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: GRPC
    nginx.ingress.kubernetes.io/listen-ports-ssl: '[19530]'
    nginx.ingress.kubernetes.io/proxy-body-size: 4m
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  labels: {}
  rules:
    - host: "milvus-example.local"
      path: "/"
      pathType: "Prefix"
    # - host: "milvus-example2.local"
    #   path: "/otherpath"
    #   pathType: "Prefix"
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - milvus-example.local

serviceAccount:
  create: false
  name:
  annotations:
  labels:

metrics:
  enabled: true

  serviceMonitor:
    # Set this to `true` to create ServiceMonitor for Prometheus operator
    enabled: false
    interval: "30s"
    scrapeTimeout: "10s"
    # Additional labels that can be used so ServiceMonitor will be
discovered by Prometheus
```

```
    additionalLabels: {}

livenessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 30
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

readinessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

log:
  level: "info"
  file:
    maxSize: 300    # MB
    maxAge: 10    # day
    maxBackups: 20
  format: "text"    # text/json

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Logs Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ## ReadWriteMany access mode required for milvus cluster.
      ##
      storageClass: default
```

```yaml
      accessModes: ReadWriteMany
      size: 10Gi
      subPath: ""

## Heaptrack traces all memory allocations and annotates these events with
stack traces.
## See more: https://github.com/KDE/heaptrack
## Enable heaptrack in production is not recommended.
heaptrack:
  image:
    repository: milvusdb/heaptrack
    tag: v0.1.0
    pullPolicy: IfNotPresent

standalone:
  replicas: 1  # Run standalone mode with replication disabled
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  disk:
    enabled: true
    size:
      enabled: false  # Enable local storage size limit
  profiling:
    enabled: false  # Enable live profiling

  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
    mountPath: "/var/lib/milvus"
    ## If true, alertmanager will create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: true
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
```

```
      ## Milvus Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ##
      storageClass:
      accessModes: ReadWriteOnce
      size: 50Gi
      subPath: ""

proxy:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  http:
    enabled: true  # whether to enable http rest server
    debugMode:
      enabled: false
  # Mount a TLS secret into proxy pod
  tls:
    enabled: false
## when enabling proxy.tls, all items below should be uncommented and the
key and crt values should be populated.
#    enabled: true
#    secretName: milvus-tls
## expecting base64 encoded values here: i.e. $(cat tls.crt | base64 -w 0)
and $(cat tls.key | base64 -w 0)
#    key: LS0tLS1CRUdJTiBQU--REDUCT
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
```

```
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

rootCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Root Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for root coordinator

  service:
    port: 53100
    annotations: {}
    labels: {}
    clusterIP: ""

queryCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Query Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for query coordinator
```

```
  service:
    port: 19531
    annotations: {}
    labels: {}
    clusterIP: ""

queryNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  disk:
    enabled: true  # Enable querynode load disk index, and search on disk
index
    size:
      enabled: false  # Enable local storage size limit
  profiling:
    enabled: false  # Enable live profiling

indexCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1   # Run Index Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for index coordinator
```

```
  service:
    port: 31000
    annotations: {}
    labels: {}
    clusterIP: ""

indexNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  disk:
    enabled: true  # Enable index node build disk vector index
    size:
      enabled: false  # Enable local storage size limit

dataCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Data Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
```

```
for data coordinator

  service:
    port: 13333
    annotations: {}
    labels: {}
    clusterIP: ""

dataNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling

## mixCoordinator contains all coord
## If you want to use mixcoord, enable this and disable all of other
coords
mixCoordinator:
  enabled: false
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Mixture Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for Mixture coordinator

  service:
```

```yaml
    annotations: {}
    labels: {}
    clusterIP: ""

attu:
  enabled: false
  name: attu
  image:
    repository: zilliz/attu
    tag: v2.2.8
    pullPolicy: IfNotPresent
  service:
    annotations: {}
    labels: {}
    type: ClusterIP
    port: 3000
    # loadBalancerIP: ""
  resources: {}
  podLabels: {}
  ingress:
    enabled: false
    annotations: {}
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    labels: {}
    hosts:
      - milvus-attu.local
    tls: []
    #  - secretName: chart-attu-tls
    #    hosts:
    #      - milvus-attu.local


## Configuration values for the minio dependency
## ref: https://github.com/minio/charts/blob/master/README.md
##

minio:
  enabled: false
  name: minio
  mode: distributed
  image:
    tag: "RELEASE.2023-03-20T20-16-18Z"
    pullPolicy: IfNotPresent
  accessKey: minioadmin
  secretKey: minioadmin
```

```yaml
  existingSecret: ""
  bucketName: "milvus-bucket"
  rootPath: file
  useIAM: false
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
  podDisruptionBudget:
    enabled: false
  resources:
    requests:
      memory: 2Gi

  gcsgateway:
    enabled: false
    replicas: 1
    gcsKeyJson: "/etc/credentials/gcs_key.json"
    projectId: ""

  service:
    type: ClusterIP
    port: 9000

  persistence:
    enabled: true
    existingClaim: ""
    storageClass:
    accessMode: ReadWriteOnce
    size: 500Gi

  livenessProbe:
    enabled: true
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 5

  readinessProbe:
    enabled: true
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 1
    successThreshold: 1
    failureThreshold: 5
```

```yaml
  startupProbe:
    enabled: true
    initialDelaySeconds: 0
    periodSeconds: 10
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 60

## Configuration values for the etcd dependency
## ref: https://artifacthub.io/packages/helm/bitnami/etcd
##

etcd:
  enabled: true
  name: etcd
  replicaCount: 3
  pdb:
    create: false
  image:
    repository: "milvusdb/etcd"
    tag: "3.5.5-r2"
    pullPolicy: IfNotPresent

  service:
    type: ClusterIP
    port: 2379
    peerPort: 2380

  auth:
    rbac:
      enabled: false

  persistence:
    enabled: true
    storageClass: default
    accessMode: ReadWriteOnce
    size: 10Gi

  ## Change default timeout periods to mitigate zoobie probe process
  livenessProbe:
    enabled: true
    timeoutSeconds: 10

  readinessProbe:
    enabled: true
    periodSeconds: 20
```

```
    timeoutSeconds: 10

  ## Enable auto compaction
  ## compaction by every 1000 revision
  ##
  autoCompactionMode: revision
  autoCompactionRetention: "1000"

  ## Increase default quota to 4G
  ##
  extraEnvVars:
  - name: ETCD_QUOTA_BACKEND_BYTES
    value: "4294967296"
  - name: ETCD_HEARTBEAT_INTERVAL
    value: "500"
  - name: ETCD_ELECTION_TIMEOUT
    value: "2500"

## Configuration values for the pulsar dependency
## ref: https://github.com/apache/pulsar-helm-chart
##

pulsar:
  enabled: true
  name: pulsar

  fullnameOverride: ""
  persistence: true

  maxMessageSize: "5242880"  # 5 * 1024 * 1024 Bytes, Maximum size of each
message in pulsar.

  rbac:
    enabled: false
    psp: false
    limit_to_namespace: true

  affinity:
    anti_affinity: false

## enableAntiAffinity: no

  components:
    zookeeper: true
    bookkeeper: true
    # bookkeeper - autorecovery
    autorecovery: true
```

```
    broker: true
    functions: false
    proxy: true
    toolset: false
    pulsar_manager: false

  monitoring:
    prometheus: false
    grafana: false
    node_exporter: false
    alert_manager: false

  images:
    broker:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    autorecovery:
      repository: apachepulsar/pulsar
      tag: 2.8.2
      pullPolicy: IfNotPresent
    zookeeper:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    bookie:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    proxy:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    pulsar_manager:
      repository: apachepulsar/pulsar-manager
      pullPolicy: IfNotPresent
      tag: v0.1.0

  zookeeper:
    volumes:
      persistence: true
      data:
        name: data
        size: 20Gi   #SSD Required
        storageClassName: default
    resources:
```

```
      requests:
        memory: 1024Mi
        cpu: 0.3
    configData:
      PULSAR_MEM: >
        -Xms1024m
        -Xmx1024m
      PULSAR_GC: >
         -Dcom.sun.management.jmxremote
         -Djute.maxbuffer=10485760
         -XX:+ParallelRefProcEnabled
         -XX:+UnlockExperimentalVMOptions
         -XX:+DoEscapeAnalysis
         -XX:+DisableExplicitGC
         -XX:+PerfDisableSharedMem
         -Dzookeeper.forceSync=no
    pdb:
      usePolicy: false

  bookkeeper:
    replicaCount: 3
    volumes:
      persistence: true
      journal:
        name: journal
        size: 100Gi
        storageClassName: default
      ledgers:
        name: ledgers
        size: 200Gi
        storageClassName: default
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+UseG1GC -XX:MaxGCPauseMillis=10
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
```

```
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
        -XX:+PerfDisableSharedMem
        -XX:+PrintGCDetails
      nettyMaxFrameSizeBytes: "104867840"
    pdb:
      usePolicy: false

  broker:
    component: broker
    podMonitor:
      enabled: false
    replicaCount: 1
    resources:
      requests:
        memory: 4096Mi
        cpu: 1.5
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
      maxMessageSize: "104857600"
      defaultRetentionTimeInMinutes: "10080"
      defaultRetentionSizeInMB: "-1"
      backlogQuotaDefaultLimitGB: "8"
      ttlDurationDefaultInSeconds: "259200"
      subscriptionExpirationTimeMinutes: "3"
      backlogQuotaDefaultRetentionPolicy: producer_exception
```

```
    pdb:
      usePolicy: false

  autorecovery:
    resources:
      requests:
        memory: 512Mi
        cpu: 1

  proxy:
    replicaCount: 1
    podMonitor:
      enabled: false
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    service:
      type: ClusterIP
    ports:
      pulsar: 6650
    configData:
      PULSAR_MEM: >
        -Xms2048m -Xmx2048m
      PULSAR_GC: >
        -XX:MaxDirectMemorySize=2048m
      httpNumThreads: "100"
    pdb:
      usePolicy: false

  pulsar_manager:
    service:
      type: ClusterIP

  pulsar_metadata:
    component: pulsar-init
    image:
      # the image used for running `pulsar-cluster-initialize` job
      repository: apachepulsar/pulsar
      tag: 2.8.2


## Configuration values for the kafka dependency
## ref: https://artifacthub.io/packages/helm/bitnami/kafka
##
```

```yaml
kafka:
  enabled: false
  name: kafka
  replicaCount: 3
  image:
    repository: bitnami/kafka
    tag: 3.1.0-debian-10-r52
  ## Increase graceful termination for kafka graceful shutdown
  terminationGracePeriodSeconds: "90"
  pdb:
    create: false

  ## Enable startup probe to prevent pod restart during recovering
  startupProbe:
    enabled: true

  ## Kafka Java Heap size
  heapOpts: "-Xmx4096m -Xms4096m"
  maxMessageBytes: _10485760
  defaultReplicationFactor: 3
  offsetsTopicReplicationFactor: 3
  ## Only enable time based log retention
  logRetentionHours: 168
  logRetentionBytes: _-1
  extraEnvVars:
  - name: KAFKA_CFG_MAX_PARTITION_FETCH_BYTES
    value: "5242880"
  - name: KAFKA_CFG_MAX_REQUEST_SIZE
    value: "5242880"
  - name: KAFKA_CFG_REPLICA_FETCH_MAX_BYTES
    value: "10485760"
  - name: KAFKA_CFG_FETCH_MESSAGE_MAX_BYTES
    value: "5242880"
  - name: KAFKA_CFG_LOG_ROLL_HOURS
    value: "24"

  persistence:
    enabled: true
    storageClass:
    accessMode: ReadWriteOnce
    size: 300Gi

  metrics:
    ## Prometheus Kafka exporter: exposes complimentary metrics to JMX
exporter
    kafka:
```

```
      enabled: false
      image:
        repository: bitnami/kafka-exporter
        tag: 1.4.2-debian-10-r182

    ## Prometheus JMX exporter: exposes the majority of Kafkas metrics
    jmx:
      enabled: false
      image:
        repository: bitnami/jmx-exporter
        tag: 0.16.1-debian-10-r245

    ## To enable serviceMonitor, you must enable either kafka exporter or
jmx exporter.
    ## And you can enable them both
    serviceMonitor:
      enabled: false

  service:
    type: ClusterIP
    ports:
      client: 9092

  zookeeper:
    enabled: true
    replicaCount: 3

#####################################
# External S3
# - these configs are only used when `externalS3.enabled` is true
#####################################
externalS3:
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
  useSSL: false
  bucketName: "milvusdbvol1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
```

```
#####################################
# GCS Gateway
# - these configs are only used when `minio.gcsgateway.enabled` is true
#####################################
externalGcs:
  bucketName: ""

#####################################
# External etcd
# - these configs are only used when `externalEtcd.enabled` is true
#####################################
externalEtcd:
  enabled: false
  ## the endpoints of the external etcd
  ##
  endpoints:
    - localhost:2379

#####################################
# External pulsar
# - these configs are only used when `externalPulsar.enabled` is true
#####################################
externalPulsar:
  enabled: false
  host: localhost
  port: 6650
  maxMessageSize: "5242880"  # 5 * 1024 * 1024 Bytes, Maximum size of each
message in pulsar.
  tenant: public
  namespace: default
  authPlugin: ""
  authParams: ""

#####################################
# External kafka
# - these configs are only used when `externalKafka.enabled` is true
#####################################
externalKafka:
  enabled: false
  brokerList: localhost:9092
  securityProtocol: SASL_SSL
  sasl:
    mechanisms: PLAIN
    username: ""
    password: ""
root@node2:~#
```

부록 B: prepare_data_netapp_new.py

이 섹션에서는 벡터 데이터베이스의 데이터를 준비하는 데 사용되는 샘플 Python 스크립트를 제공합니다.

부록 B: prepare_data_netapp_new.py

```
root@node2:~# cat prepare_data_netapp_new.py
# hello_milvus.py demonstrates the basic operations of PyMilvus, a Python
SDK of Milvus.
# 1. connect to Milvus
# 2. create collection
# 3. insert data
# 4. create index
# 5. search, query, and hybrid search on entities
# 6. delete entities by PK
# 7. drop collection
import time
import os
import numpy as np
from pymilvus import (
    connections,
    utility,
    FieldSchema, CollectionSchema, DataType,
    Collection,
)

fmt = "\n=== {:30} ===\n"
search_latency_fmt = "search latency = {:.4f}s"
#num_entities, dim = 3000, 8
num_entities, dim = 3000, 16

###########################################################################
#######
# 1. connect to Milvus
# Add a new connection alias `default` for Milvus server in
`localhost:19530`
# Actually the "default" alias is a buildin in PyMilvus.
# If the address of Milvus is the same as `localhost:19530`, you can omit
all
# parameters and call the method as: `connections.connect()`.
#
# Note: the `using` parameter of the following methods is default to
"default".
print(fmt.format("start connecting to Milvus"))
```

```
host = os.environ.get('MILVUS_HOST')
if host == None:
    host = "localhost"
print(fmt.format(f"Milvus host: {host}"))
#connections.connect("default", host=host, port="19530")
connections.connect("default", host=host, port="27017")

has = utility.has_collection("hello_milvus_ntapnew_update2_sc")
print(f"Does collection hello_milvus_ntapnew_update2_sc exist in Milvus:
{has}")

#drop the collection
print(fmt.format(f"Drop collection - hello_milvus_ntapnew_update2_sc"))
utility.drop_collection("hello_milvus_ntapnew_update2_sc")
#drop the collection
print(fmt.format(f"Drop collection - hello_milvus_ntapnew_update2_sc2"))
utility.drop_collection("hello_milvus_ntapnew_update2_sc2")

##########################################################################
#######
# 2. create collection
# We're going to create a collection with 3 fields.
# +-+------------+------------+------------------
+-------------------------------+
# | | field name | field type | other attributes |       field description
|
# +-+------------+------------+------------------
+-------------------------------+
# |1|    "pk"    |    Int64   |  is_primary=True |      "primary field"
|
# | |            |            |   auto_id=False  |
|
# +-+------------+------------+------------------
+-------------------------------+
# |2|  "random"  |    Double  |                  |      "a double field"
|
# +-+------------+------------+------------------
+-------------------------------+
# |3|"embeddings"| FloatVector|     dim=8        |  "float vector with dim
8"   |
# +-+------------+------------+------------------
+-------------------------------+
fields = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=False),
```

```
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]

schema = CollectionSchema(fields, "hello_milvus_ntapnew_update2_sc")

print(fmt.format("Create collection `hello_milvus_ntapnew_update2_sc`"))
hello_milvus_ntapnew_update2_sc = Collection
("hello_milvus_ntapnew_update2_sc", schema, consistency_level="Strong")

##########################################################################
######
# 3. insert data
# We are going to insert 3000 rows of data into
`hello_milvus_ntapnew_update2_sc`
# Data to be inserted must be organized in fields.
#
# The insert() method returns:
# - either automatically generated primary keys by Milvus if auto_id=True
in the schema;
# - or the existing primary key field from the entities if auto_id=False
in the schema.

print(fmt.format("Start inserting entities"))
rng = np.random.default_rng(seed=19530)
entities = [
    # provide the pk field because `auto_id` is set to False
    [i for i in range(num_entities)],
    rng.random(num_entities).tolist(),  # field random, only supports list
    [str(i) for i in range(num_entities)],
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

insert_result = hello_milvus_ntapnew_update2_sc.insert(entities)
hello_milvus_ntapnew_update2_sc.flush()
print(f"Number of entities in hello_milvus_ntapnew_update2_sc:
{hello_milvus_ntapnew_update2_sc.num_entities}")  # check the num_entites

# create another collection
fields2 = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=True),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
```

```
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]


schema2 = CollectionSchema(fields2, "hello_milvus_ntapnew_update2_sc2")

print(fmt.format("Create collection `hello_milvus_ntapnew_update2_sc2`"))
hello_milvus_ntapnew_update2_sc2 = Collection
("hello_milvus_ntapnew_update2_sc2", schema2, consistency_level="Strong")

entities2 = [
    rng.random(num_entities).tolist(),  # field random, only supports list
    [str(i) for i in range(num_entities)],
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

insert_result2 = hello_milvus_ntapnew_update2_sc2.insert(entities2)
hello_milvus_ntapnew_update2_sc2.flush()
insert_result2 = hello_milvus_ntapnew_update2_sc2.insert(entities2)
hello_milvus_ntapnew_update2_sc2.flush()

# index_params = {"index_type": "IVF_FLAT", "params": {"nlist": 128},
"metric_type": "L2"}
# hello_milvus_ntapnew_update2_sc.create_index("embeddings", index_params)
#
hello_milvus_ntapnew_update2_sc2.create_index(field_name="var",index_name=
"scalar_index")

# index_params2 = {"index_type": "Trie"}
# hello_milvus_ntapnew_update2_sc2.create_index("var", index_params2)

print(f"Number of entities in hello_milvus_ntapnew_update2_sc2:
{hello_milvus_ntapnew_update2_sc2.num_entities}")  # check the num_entites

root@node2:~#
```

부록 C: verify_data_netapp.py

이 섹션에는 NetApp 벡터 데이터베이스 솔루션의 벡터 데이터베이스를 검증하는 데 사용할 수 있는 샘플 Python 스크립트가 포함되어 있습니다.

부록 C: verify_data_netapp.py

```
root@node2:~# cat verify_data_netapp.py
```

```
import time
import os
import numpy as np
from pymilvus import (
    connections,
    utility,
    FieldSchema, CollectionSchema, DataType,
    Collection,
)

fmt = "\n=== {:30} ===\n"
search_latency_fmt = "search latency = {:.4f}s"
num_entities, dim = 3000, 16
rng = np.random.default_rng(seed=19530)
entities = [
    # provide the pk field because `auto_id` is set to False
    [i for i in range(num_entities)],
    rng.random(num_entities).tolist(),  # field random, only supports list
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

#######################################################################
######
# 1. get recovered collection hello_milvus_ntapnew_update2_sc
print(fmt.format("start connecting to Milvus"))
host = os.environ.get('MILVUS_HOST')
if host == None:
    host = "localhost"
print(fmt.format(f"Milvus host: {host}"))
#connections.connect("default", host=host, port="19530")
connections.connect("default", host=host, port="27017")

recover_collections = ["hello_milvus_ntapnew_update2_sc",
"hello_milvus_ntapnew_update2_sc2"]

for recover_collection_name in recover_collections:
    has = utility.has_collection(recover_collection_name)
    print(f"Does collection {recover_collection_name} exist in Milvus:
{has}")
    recover_collection = Collection(recover_collection_name)
    print(recover_collection.schema)
    recover_collection.flush()

    print(f"Number of entities in Milvus: {recover_collection_name} :
{recover_collection.num_entities}")  # check the num_entites
```

```
##########################################################################
######
    # 4. create index
    # We are going to create an IVF_FLAT index for
hello_milvus_ntapnew_update2_sc collection.
    # create_index() can only be applied to `FloatVector` and
`BinaryVector` fields.
    print(fmt.format("Start Creating index IVF_FLAT"))
    index = {
        "index_type": "IVF_FLAT",
        "metric_type": "L2",
        "params": {"nlist": 128},
    }

    recover_collection.create_index("embeddings", index)


##########################################################################
######
    # 5. search, query, and hybrid search
    # After data were inserted into Milvus and indexed, you can perform:
    # - search based on vector similarity
    # - query based on scalar filtering(boolean, int, etc.)
    # - hybrid search based on vector similarity and scalar filtering.
    #

    # Before conducting a search or a query, you need to load the data in
`hello_milvus` into memory.
    print(fmt.format("Start loading"))
    recover_collection.load()

    #
---------------------------------------------------------------------------
---
    # search based on vector similarity
    print(fmt.format("Start searching based on vector similarity"))
    vectors_to_search = entities[-1][-2:]
    search_params = {
        "metric_type": "L2",
        "params": {"nprobe": 10},
    }

    start_time = time.time()
    result = recover_collection.search(vectors_to_search, "embeddings",
search_params, limit=3, output_fields=["random"])
```

```
    end_time = time.time()

    for hits in result:
        for hit in hits:
            print(f"hit: {hit}, random field: {hit.entity.get('random')}")
    print(search_latency_fmt.format(end_time - start_time))

    #
---------------------------------------------------------------------------
---
    # query based on scalar filtering(boolean, int, etc.)
    print(fmt.format("Start querying with `random > 0.5`"))

    start_time = time.time()
    result = recover_collection.query(expr="random > 0.5", output_fields=
["random", "embeddings"])
    end_time = time.time()

    print(f"query result:\n-{result[0]}")
    print(search_latency_fmt.format(end_time - start_time))

    #
---------------------------------------------------------------------------
---
    # hybrid search
    print(fmt.format("Start hybrid searching with `random > 0.5`"))

    start_time = time.time()
    result = recover_collection.search(vectors_to_search, "embeddings",
search_params, limit=3, expr="random > 0.5", output_fields=["random"])
    end_time = time.time()

    for hits in result:
        for hit in hits:
            print(f"hit: {hit}, random field: {hit.entity.get('random')}")
    print(search_latency_fmt.format(end_time - start_time))


#########################################################################
#####
    # 7. drop collection
    # Finally, drop the hello_milvus, hello_milvus_ntapnew_update2_sc
collection

    #print(fmt.format(f"Drop collection {recover_collection_name}"))
    #utility.drop_collection(recover_collection_name)
```

```
root@node2:~#
```

부록 D: docker-compose.yml

이 섹션에는 NetApp 의 벡터 데이터베이스 솔루션에 대한 샘플 YAML 코드가 포함되어 있습니다.

부록 D: docker-compose.yml

```yaml
version: '3.5'

services:
  etcd:
    container_name: milvus-etcd
    image: quay.io/coreos/etcd:v3.5.5
    environment:
      - ETCD_AUTO_COMPACTION_MODE=revision
      - ETCD_AUTO_COMPACTION_RETENTION=1000
      - ETCD_QUOTA_BACKEND_BYTES=4294967296
      - ETCD_SNAPSHOT_COUNT=50000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/etcd:/etcd
    command: etcd -advertise-client-urls=http://127.0.0.1:2379 -listen
-client-urls http://0.0.0.0:2379 --data-dir /etcd
    healthcheck:
      test: ["CMD", "etcdctl", "endpoint", "health"]
      interval: 30s
      timeout: 20s
      retries: 3

  minio:
    container_name: milvus-minio
    image: minio/minio:RELEASE.2023-03-20T20-16-18Z
    environment:
      MINIO_ACCESS_KEY: minioadmin
      MINIO_SECRET_KEY: minioadmin
    ports:
      - "9001:9001"
      - "9000:9000"
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/minio:/minio_data
    command: minio server /minio_data --console-address ":9001"
    healthcheck:
      test: ["CMD", "curl", "-f",
```

```
"http://localhost:9000/minio/health/live"]
      interval: 30s
      timeout: 20s
      retries: 3

  standalone:
    container_name: milvus-standalone
    image: milvusdb/milvus:v2.4.0-rc.1
    command: ["milvus", "run", "standalone"]
    security_opt:
    - seccomp:unconfined
    environment:
      ETCD_ENDPOINTS: etcd:2379
      MINIO_ADDRESS: minio:9000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/milvus:/var/lib/milvus
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:9091/healthz"]
      interval: 30s
      start_period: 90s
      timeout: 20s
      retries: 3
    ports:
      - "19530:19530"
      - "9091:9091"
    depends_on:
      - "etcd"
      - "minio"

networks:
  default:
    name: milvus
```

저작권 정보

Copyright © 2025 NetApp, Inc. All Rights Reserved. 미국에서 인쇄된 본 문서의 어떠한 부분도 저작권 소유자의 사전 서면 승인 없이는 어떠한 형식이나 수단(복사, 녹음, 녹화 또는 전자 검색 시스템에 저장하는 것을 비롯한 그래픽, 전자적 또는 기계적 방법)으로도 복제될 수 없습니다.

NetApp이 저작권을 가진 자료에 있는 소프트웨어에는 아래의 라이센스와 고지사항이 적용됩니다.

본 소프트웨어는 NetApp에 의해 '있는 그대로' 제공되며 상품성 및 특정 목적에의 적합성에 대한 명시적 또는 묵시적 보증을 포함하여(이에 제한되지 않음) 어떠한 보증도 하지 않습니다. NetApp은 대체품 또는 대체 서비스의 조달, 사용 불능, 데이터 손실, 이익 손실, 영업 중단을 포함하여(이에 국한되지 않음), 이 소프트웨어의 사용으로 인해 발생하는 모든 직접 및 간접 손해, 우발적 손해, 특별 손해, 징벌적 손해, 결과적 손해의 발생에 대하여 그 발생 이유, 책임론, 계약 여부, 엄격한 책임, 불법 행위(과실 또는 그렇지 않은 경우)와 관계없이 어떠한 책임도 지지 않으며, 이와 같은 손실의 발생 가능성이 통지되었다 하더라도 마찬가지입니다.

NetApp은 본 문서에 설명된 제품을 언제든지 예고 없이 변경할 권리를 보유합니다. NetApp은 NetApp의 명시적인 서면 동의를 받은 경우를 제외하고 본 문서에 설명된 제품을 사용하여 발생하는 어떠한 문제에도 책임을 지지 않습니다. 본 제품의 사용 또는 구매의 경우 NetApp에서는 어떠한 특허권, 상표권 또는 기타 지적 재산권이 적용되는 라이센스도 제공하지 않습니다.

본 설명서에 설명된 제품은 하나 이상의 미국 특허, 해외 특허 또는 출원 중인 특허로 보호됩니다.

제한적 권리 표시: 정부에 의한 사용, 복제 또는 공개에는 DFARS 252.227-7013(2014년 2월) 및 FAR 52.227-19(2007년 12월)의 기술 데이터-비상업적 품목에 대한 권리(Rights in Technical Data -Noncommercial Items) 조항의 하위 조항 (b)(3)에 설명된 제한사항이 적용됩니다.

여기에 포함된 데이터는 상업용 제품 및/또는 상업용 서비스(FAR 2.101에 정의)에 해당하며 NetApp, Inc.의 독점 자산입니다. 본 계약에 따라 제공되는 모든 NetApp 기술 데이터 및 컴퓨터 소프트웨어는 본질적으로 상업용이며 개인 비용만으로 개발되었습니다. 미국 정부는 데이터가 제공된 미국 계약과 관련하여 해당 계약을 지원하는 데에만 데이터에 대한 전 세계적으로 비독점적이고 양도할 수 없으며 재사용이 불가능하며 취소 불가능한 라이센스를 제한적으로 가집니다. 여기에 제공된 경우를 제외하고 NetApp, Inc.의 사전 서면 승인 없이는 이 데이터를 사용, 공개, 재생산, 수정, 수행 또는 표시할 수 없습니다. 미국 국방부에 대한 정부 라이센스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

NETAPP, NETAPP 로고 및 http://www.netapp.com/TM에 나열된 마크는 NetApp, Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.