



솔루션 검증 개요

NetApp Solutions

NetApp
May 03, 2024

목차

솔루션 개요	1
온프레미스에서 Kubernetes를 사용한 Milvus 클러스터 설정	1
NetApp ONTAP 파일 및 객체 이중성을 위한 Amazon FSxN을 사용하는 Milvus	8
SnapCenter를 사용한 벡터 데이터베이스 보호	14
NetApp SnapMirror를 사용한 재해 복구	25
Vector 데이터베이스 성능 검증	27

솔루션 개요

당사는 5가지 주요 영역에 중점을 두고 포괄적인 솔루션 검증을 수행했으며, 자세한 내용은 아래에 나와 있습니다. 각 섹션에서는 고객이 직면한 과제, NetApp에서 제공하는 솔루션, 그리고 고객이 누릴 수 있는 이점에 대해 알아봅니다.

1. "온프레미스에서 Kubernetes를 사용하여 Milvus 클러스터 설정"

스토리지와 컴퓨팅, 효과적인 인프라 관리 및 데이터 관리에 대해 독립적으로 확장해야 하는 고객의 당면 과제 이 섹션에서는 클러스터 데이터 및 고객 데이터 모두에 NetApp 스토리지 컨트롤러를 활용하여 Kubernetes에 Milvus 클러스터를 설치하는 프로세스를 자세히 설명합니다.

2. "NetApp ONTAP용 Amazon FSxN을 사용하는 Milvus – 파일 및 오브젝트 이중화"

이 섹션에서는 클라우드에 벡터 데이터베이스를 배포해야 하는 이유와 Docker 컨테이너 내의 Amazon FSxN for NetApp ONTAP에 벡터 데이터베이스(milvus 독립형)를 배포하는 단계를 설명합니다.

3. "NetApp SnapCenter를 사용한 벡터 데이터베이스 보호."

이 섹션에서는 SnapCenter가 ONTAP에 상주하는 벡터 데이터베이스 데이터와 Milvus 데이터를 보호하는 방법을 알아봅니다. 이 예에서는 고객 데이터에 NFS ONTAP 볼륨(vol1)에서 파생된 NAS 버킷(milvusdbvol1)을 활용하고 Milvus 클러스터 구성 데이터에 별도의 NFS 볼륨(vectordbvp)을 사용했습니다.

4. "NetApp SnapMirror를 사용한 재해 복구"

이 섹션에서는 벡터 데이터베이스에 대한 DR(재해 복구)의 중요성과 NetApp 재해 복구 제품 SnapMirror가 벡터 데이터베이스에 DR 솔루션을 제공하는 방법에 대해 설명합니다.

5. "성능 검증"

이 섹션에서는 Milvus 및 pgvecto.RS와 같은 벡터 데이터베이스의 성능 검증을 자세히 살펴보고, LLM 수명주기 내의 RAG 및 추론 워크로드를 지원하는 I/O 프로파일 및 NetApp 스토리지 컨트롤러 같은 스토리지 성능 특성에 초점을 맞추고자 합니다. 이들 데이터베이스가 ONTAP 스토리지 솔루션과 결합될 때 성능 차별화 요소를 평가하고 식별합니다. 당사의 분석은 QPS(초당 처리된 쿼리 수)와 같은 주요 성능 지표를 기반으로 합니다.

온프레미스에서 Kubernetes를 사용한 Milvus 클러스터 설정

온프레미스에서 Kubernetes를 사용하여 Milvus 클러스터 설정

고객의 당면 과제: 스토리지와 컴퓨팅, 효과적인 인프라 관리 및 데이터 관리

Kubernetes와 벡터 데이터베이스는 함께 대규모 데이터 작업 관리를 위한 강력하고 확장 가능한 솔루션을 형성합니다. Kubernetes는 리소스를 최적화하고 컨테이너를 관리하는 한편, 벡터 데이터베이스는 고밀도 데이터 및 유사성 검색을 효율적으로 처리합니다. 이 결합을 사용하면 대규모 데이터 세트에서 복잡한 쿼리를 신속하게 처리할 수 있고 증가하는 데이터 볼륨으로 원활하게 확장할 수 있으므로 빅데이터 애플리케이션과 AI 워크로드에 이상적입니다.

1. 이 섹션에서는 클러스터 데이터 및 고객 데이터 모두에 NetApp 스토리지 컨트롤러를 활용하여 Kubernetes에 Milvus 클러스터를 설치하는 프로세스를 자세히 설명합니다.

2. Milvus 클러스터를 설치하려면 다양한 Milvus 클러스터 구성 요소의 데이터를 저장하기 위해 영구 볼륨(PVS)이 필요합니다. 이러한 구성 요소에는 etcd(3개 인스턴스), Pulsar-bookie-journal(3개 인스턴스), Pulsar-bookie-ledgers(3개 인스턴스), Pulsar-zookeeper-data(3개 인스턴스) 등이 있습니다.



Milvus 클러스터에서는 Milvus 클러스터의 안정적인 저장 및 메시지 스트림의 게시/구독을 지원하는 기본 엔진에 Pulsar 또는 Kafka를 사용할 수 있습니다. NFS를 사용하는 Kafka의 경우 NetApp은 ONTAP 9.12.1 이상에서 성능을 향상했으며, RHEL 8.7 또는 9.1 이상에 포함된 NFSv4.1 및 Linux 변경 사항과 함께 이러한 기능 향상을 통해 NFS에서 Kafka를 실행할 때 발생할 수 있는 "silly rename" 문제를 해결합니다. NetApp NFS 솔루션과 함께 Kafka 실행 주제에 관한 더 자세한 정보는 을 참조하십시오. <https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html>.

- 우리는 NetApp ONTAP에서 단일 NFS 볼륨을 생성하고 각각 250GB의 스토리지를 가진 12개의 영구 볼륨을 생성했습니다. 스토리지 크기는 클러스터 크기에 따라 다를 수 있습니다. 예를 들어 각 PV에 50GB가 있는 또 다른 클러스터가 있습니다. 자세한 내용은 아래 PV YAML 파일 중 하나를 참조하십시오. 총 12개의 파일이 있습니다. 각 파일에서 storageClassName은 'default'로 설정되고 스토리지 및 경로는 각 PV에 대해 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:~#
```

- 각 PV YAML 파일에 대해 'kubectl apply' 명령을 실행하여 영구 볼륨을 생성한 다음 'kubectl get PV'를 사용하여 생성된 볼륨을 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. 고객 데이터 저장을 위해 Milvus는 MinIO, Azure Blob 및 S3와 같은 오브젝트 스토리지 솔루션을 지원합니다. 이 가이드에서는 S3를 활용합니다. 다음 단계는 ONTAP S3 및 StorageGRID 오브젝트 저장소 모두에 적용됩니다. 우리는 Helm을 사용하여 Milvus 클러스터를 구축합니다. Milvus 다운로드 위치에서 구성 파일 Values.yaml을 다운로드합니다. 이 문서에서 사용한 .yaml 값 파일은 부록을 참조하십시오.
6. 로그, etcd, zookeeper 및 bookkeeper를 포함하여 각 섹션에서 'storageClass'가 'default'로 설정되어 있는지 확인합니다.
7. MinIO 섹션에서 MinIO를 비활성화합니다.
8. ONTAP 또는 StorageGRID 오브젝트 스토리지에서 NAS 버킷을 생성하고 오브젝트 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: "milvusdbvoll1"  
  rootPath: ""  
  useIAM: false  
  cloudProvider: "aws"  
  iamEndpoint: ""  
  region: ""  
  useVirtualHost: false
```

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

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

10. Helm 및 value.yaml 구성 파일을 사용하여 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. Pod의 상태를 확인합니다.

```

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

```

Pod 상태가 '실행 중'이고 예상대로 작동하는지 확인하십시오

13. Milvus 및 NetApp 오브젝트 스토리지에서 데이터 쓰기 및 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}}]}

```

NetApp 스토리지 컨트롤러를 사용하여 Kubernetes에 Milvus 클러스터를 구축하여 설명한 대로 위의 검증을 기반으로 Kubernetes와 벡터 데이터베이스를 통합하면 대규모 데이터 운영 관리를 위한 강력하고 확장 가능하며 효율적인 솔루션을 고객에게 제공합니다. 이 설정은 고객이 높은 차원 데이터를 처리하고 복잡한 쿼리를 신속하고 효율적으로 실행할 수 있도록 하여 빅 데이터 애플리케이션 및 AI 워크로드에 이상적인 솔루션입니다. 다양한 클러스터 구성 요소에 PVS(영구 볼륨)를 사용하고 NetApp ONTAP에서 단일 NFS 볼륨을 생성하면 최적의 리소스 활용도와 데이터 관리가 보장됩니다. PersistentVolumeClaims(PVC) 및 Pod의 상태를 확인하고 데이터 쓰기 및 읽기 테스트를 통해 고객은 안정적이고 일관된 데이터 작업을 보장할 수 있습니다. ONTAP 또는 StorageGRID 오브젝트 스토리지를 고객 데이터에 사용하면 데이터 접근성과 보안이

더욱 강화됩니다. 이 설정을 통해 고객은 증가하는 데이터 요구사항에 맞춰 원활하게 확장할 수 있는 복원력을 갖춘 고성능 데이터 관리 솔루션을 확보할 수 있습니다.

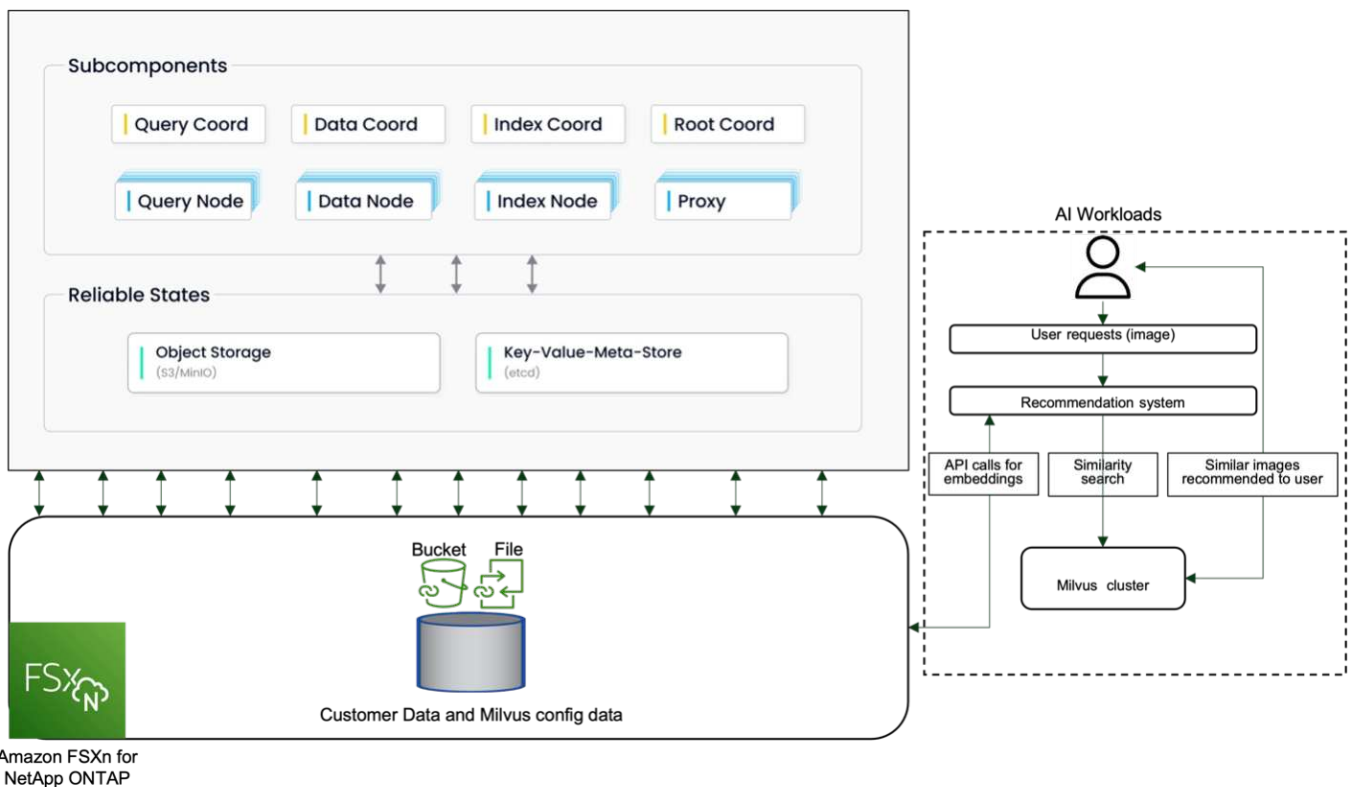
NetApp ONTAP 파일 및 객체 이중성을 위한 Amazon FSxN을 사용하는 Milvus

NetApp ONTAP용 Amazon FSxN을 사용하는 Milvus – 파일 및 오브젝트 이중화

이 섹션에서는 클라우드에 벡터 데이터베이스를 배포해야 하는 이유와 Docker 컨테이너 내의 Amazon FSxN for NetApp ONTAP에 벡터 데이터베이스(milvus 독립형)를 배포하는 단계를 설명합니다.

클라우드에 벡터 데이터베이스를 배포하면 특히 높은 차원 데이터를 처리하고 유사성 검색을 실행해야 하는 애플리케이션의 경우 몇 가지 중요한 이점을 얻을 수 있습니다. 첫째, 클라우드 기반 배포는 확장성을 제공하므로 증가하는 데이터 볼륨 및 쿼리 로드에도 맞게 리소스를 쉽게 조정할 수 있습니다. 이를 통해 데이터베이스에서 고성능을 유지하면서 증가하는 수요를 효율적으로 처리할 수 있습니다. 둘째, 클라우드 배포는 데이터를 서로 다른 지리적 위치에 복제할 수 있어 고가용성 및 재해 복구를 제공하며, 데이터 손실 위험을 최소화하고 예기치 않은 이벤트에도 지속적인 서비스를 보장합니다. 셋째, 사용하는 리소스에 대해서만 비용을 지불하므로 비용 효율성이 향상되고 필요에 따라 확장 또는 축소할 수 있으므로 하드웨어에 대한 초기 투자 비용을 크게 줄일 필요가 없습니다. 마지막으로 클라우드에 벡터 데이터베이스를 배포하면 어디서나 데이터에 액세스하고 공유할 수 있으므로 팀 기반 작업과 데이터 기반 의사 결정을 용이하게 할 수 있으므로 협업을 개선할 수 있습니다.

이 검증에 사용된 NetApp ONTAP용 Amazon FSxN과 함께 Milvus 독립 실행형 아키텍처를 확인하십시오.



1. NetApp ONTAP용 Amazon FSxN 인스턴스를 생성하고 VPC, VPC 보안 그룹 및 서브넷의 세부 정보를 기록합니다. 이 정보는 EC2 인스턴스를 생성할 때 필요합니다. 자세한 내용은 여기 에서 확인할 수 있습니다. - <https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create>
2. EC2 인스턴스를 생성하여 VPC, 보안 그룹 및 서브넷이 NetApp ONTAP용 Amazon FSxN 인스턴스의 인스턴스와 일치하는지 확인합니다.

- 'apt-get install nfs-common' 명령을 사용하여 nfs-common을 설치하고 'SUDO apt-get update'를 사용하여 패키지 정보를 업데이트합니다.
- 마운트 폴더를 생성하고 NetApp ONTAP용 Amazon FSxN을 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:~$
```

- 'apt-get install'을 사용하여 Docker 및 Docker Compose를 설치합니다.
- Milvus 웹 사이트에서 다운로드할 수 있는 docker-composition.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>
```

- docker-configuration.yaml 파일의 'volumes' 섹션에서 NetApp NFS 마운트 지점을 해당 Milvus 컨테이너 경로, 특히 etcd, minio 및 standalone.Check에 매핑합니다 "부록 D: docker-composition.yaml" YML의 변화에 대한 자세한 내용은
- 마운트된 폴더 및 파일을 확인합니다.

```
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:~$
```

- docker-composition.yaml 파일이 들어 있는 디렉토리에서 'docker-composure up-d'를 실행합니다.

10. Milvus 컨테이너의 상태를 확인합니다.

```
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. 벡터 데이터베이스의 읽기 및 쓰기 기능을 검증하기 위해 Amazon FSxN for NetApp ONTAP에서는 Python Milvus SDK와 PyMilvus의 샘플 프로그램을 사용했습니다. 'apt-get install python3-numpy python3-pip'을 사용하여 필요한 패키지를 설치하고 'pip3 install pymilvus'를 사용하여 PyMilvus를 설치합니다.

12. 벡터 데이터베이스에서 NetApp ONTAP용 Amazon FSxN의 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
```

```

/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/b3def25f-c117-4fba-8256-96cb7557cd6c
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/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
/448789845791611912/448789845791611913/448789845791611939/0/448789845791
411924
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/0/448789845791
411924/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
/448789845791611912/448789845791611913/448789845791611939/1/448789845791
411925
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/1/448789845791
411925/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
/448789845791611912/448789845791611913/448789845791611939/100/4487898457
91411920
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/100/4487898457
91411920/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: 0.07971227169036865, 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 워크로드용 S3 프로토콜을 통해 벡터 데이터베이스에서 테스트된 NFS 데이터에 액세스하려는 경우 간단한 Python 프로그램을 사용하여 검증을 받을 수 있습니다. 예를 들어 이 섹션의 시작 부분에 있는 그림에서 언급했듯이 다른 응용 프로그램에서 이미지를 비슷한 방식으로 검색할 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
hoPctr9aD88c1j0SkIYZ2uPa03v1bqKA0c5feK6F
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 NetApp ONTAP 데이터 스토리지용 Amazon의 NetApp FSxN을 활용하여 고객이 Docker 컨테이너 내에서 독립 실행형 Milvus 설정을 구축 및 운영하는 방법을 효과적으로 보여줍니다. 이 설치를 통해 고객은 Docker 컨테이너의 확장 가능하고 효율적인 환경 내에서 벡터 데이터베이스의 기능을 활용하여 차원 높은 데이터를 처리하고 복잡한 쿼리를 실행할 수 있습니다. NetApp ONTAP 인스턴스용 Amazon FSxN을 생성하고 EC2 인스턴스를 일치시킴으로써 최적의 리소스 활용도와 데이터 관리를 보장할 수 있습니다. 벡터 데이터베이스에서 FSxN의 데이터 쓰기 및 읽기 작업을 성공적으로 검증함으로써 고객은 안정적이고 일관된 데이터 작업을 보장할 수 있습니다. 또한, S3 프로토콜을 통해 AI 워크로드의 데이터를 나열(읽기) 기능은 향상된 데이터 접근성을 제공합니다. 따라서 이 포괄적인 프로세스는 고객에게 NetApp ONTAP용 Amazon의 FSxN의 기능을 활용하여 대규모 데이터 운영을 관리할 수 있는 강력하고 효율적인 솔루션을 제공합니다.

SnapCenter를 사용한 벡터 데이터베이스 보호

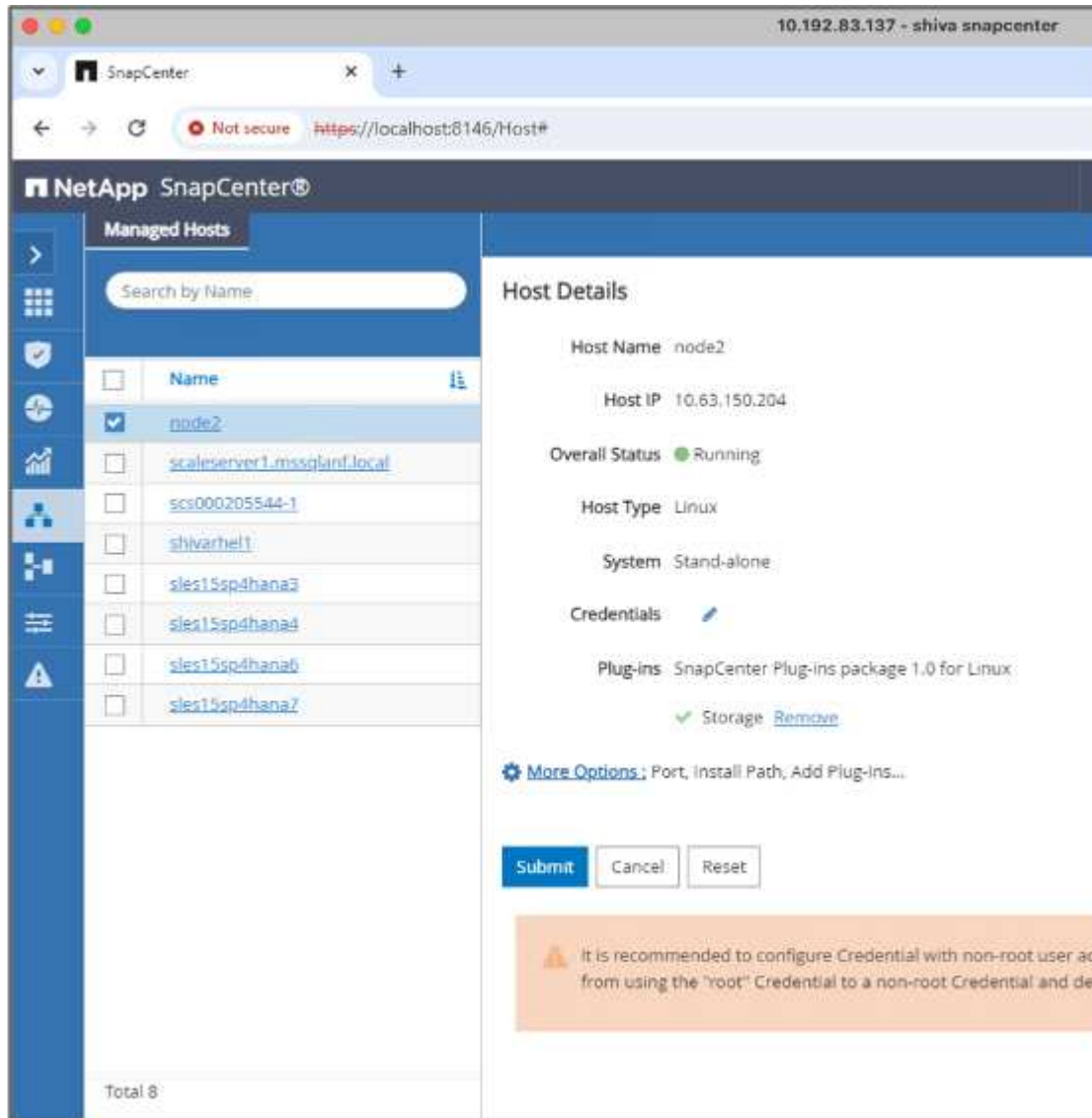
NetApp SnapCenter를 사용한 벡터 데이터베이스 보호.

예를 들어 영화 제작 업계에서 고객은 비디오 및 오디오 파일과 같은 중요한 내장 데이터를 보유하는 경우가 많습니다. 하드 드라이브 장애와 같은 문제로 인해 이 데이터가 손실되면 운영에 큰 영향을 미칠 수 있으며 수백만 달러 규모의 벤처 기업을 위험에 빠뜨릴 수 있습니다. 귀중한 콘텐츠가 유실되어 상당한 혼란과 재정적 손실이 발생하는 경우가 있었습니다. 따라서 이 업계에서는 필수 데이터의 보안과 무결성을 보장하는 것이 매우 중요합니다.

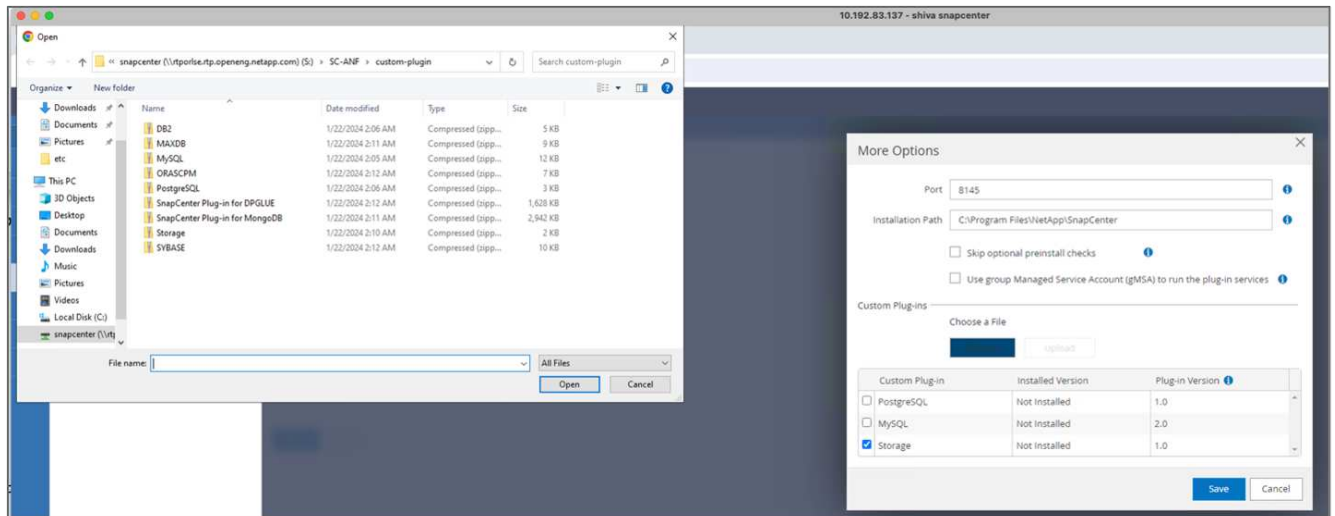
이 섹션에서는 SnapCenter가 ONTAP에 상주하는 벡터 데이터베이스 데이터와 Milvus 데이터를 보호하는 방법을 알아봅니다. 이 예에서는 고객 데이터에 NFS ONTAP 볼륨(vol1)에서 파생된 NAS 버킷(milvusdbvol1)을 활용하고 Milvus 클러스터 구성 데이터에 별도의 NFS 볼륨(vectordbpv)을 사용했습니다. 를 확인하십시오 ["여기"](#) SnapCenter

백업 워크플로우의 경우

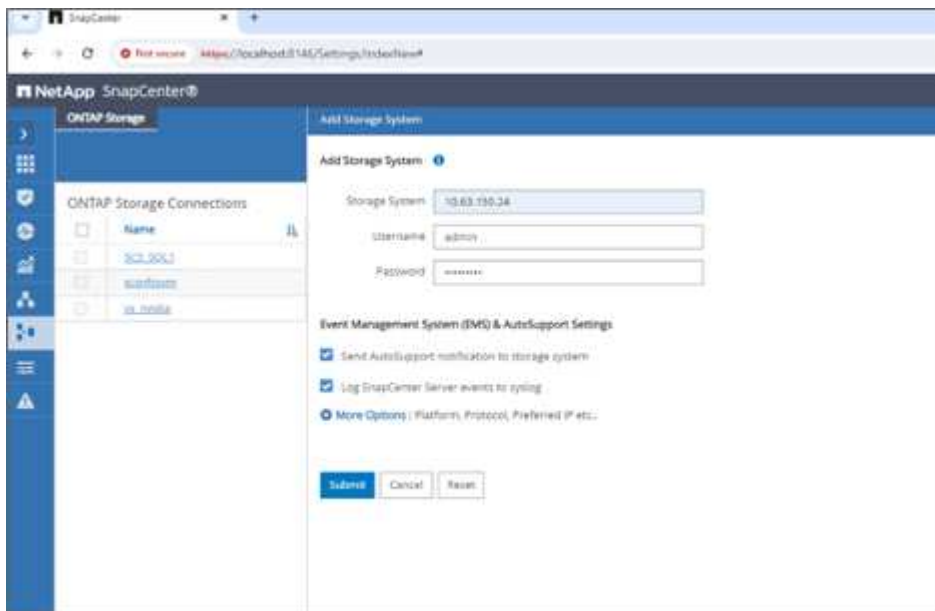
1. SnapCenter 명령을 실행하는 데 사용할 호스트를 설정합니다.



2. 스토리지 플러그인을 설치 및 구성합니다. 추가된 호스트에서 "추가 옵션"을 선택합니다. 에서 다운로드한 스토리지 플러그인을 찾아 선택합니다 "NetApp 자동화 스토어". 플러그인을 설치하고 구성을 저장합니다.



3. 스토리지 시스템 및 볼륨 설정: "Storage System"에 스토리지 시스템을 추가하고 SVM(Storage Virtual Machine)을 선택합니다. 이 예에서는 "vs_nvidia"를 선택했습니다.



4. 백업 정책 및 사용자 지정 스냅샷 이름을 통합하여 벡터 데이터베이스에 대한 리소스를 설정합니다.
- 기본값으로 정합성 보장 그룹 백업을 설정하고 파일 시스템 정합성 보장 없이 SnapCenter를 설정합니다.
 - Storage Footprint 섹션에서 벡터 데이터베이스 고객 데이터 및 Milvus 클러스터 데이터와 연결된 볼륨을 선택합니다. 이 예에서는 "vol1"과 "vectordbpv"입니다.
 - 정책을 사용하여 벡터 데이터베이스 보호에 대한 정책을 만들고 벡터 데이터베이스 리소스를 보호합니다.

Modify Storage Storage Resource

1 Name

2 Storage Footprint

3 Resource Settings

4 Summary

Summary

Name milvusdb

Type None

Host scaleserver1.mssqlanf.local

Mount Points

Credential Name adminuser

Storage Footprint

Storage System	Volume	LUN/Qtree
vs_nvidia	vol1	
	vectordbpv	

Custom Resource Parameters None

Previous Finish

- Python 스크립트를 사용하여 S3 NAS 버킷에 데이터를 삽입합니다. 이 경우 Milvus에서 제공하는 백업 스크립트, 즉 '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:~#

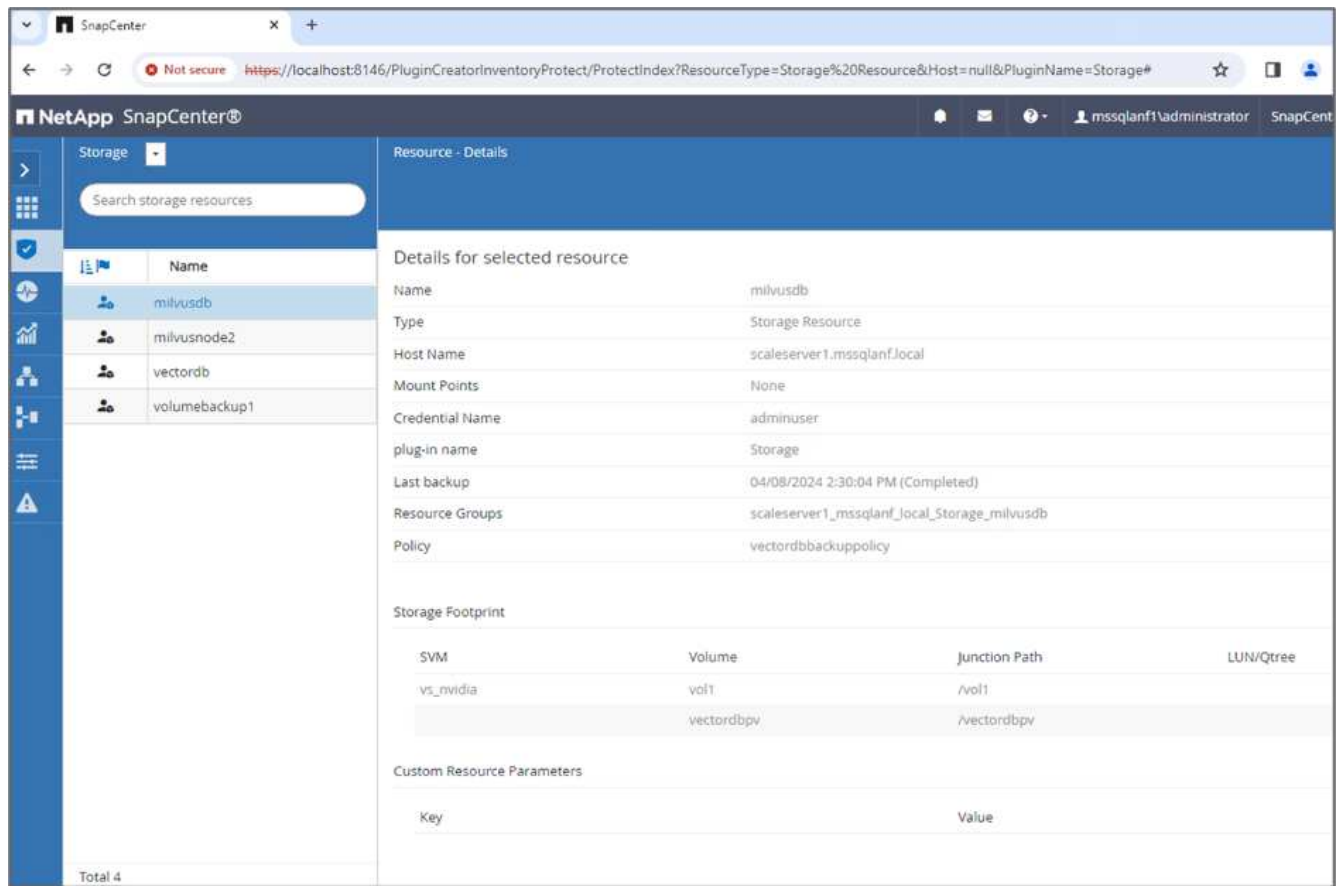
```

6. S3 NAS 버킷에서 데이터를 확인합니다. 이 예에서는 타임 스탬프 '2024-04-08 21:22'를 가진 파일이 '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 CG(정합성 보장 그룹) 스냅샷을 사용하여 백업을 시작합니다



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

이 테스트를 위해 누군가가 백업 후 불필요한 새로운 수집 또는 부적절한 수집을 생성한 시나리오를 생각해 보십시오. 이 경우 새 컬렉션이 추가되기 전에 벡터 데이터베이스를 해당 상태로 되돌려야 합니다. 예를 들어, 'hello_milvus_netapp_sc_testnew', 'hello_milvus_netapp_sc_testnew2'와 같은 새 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        ===

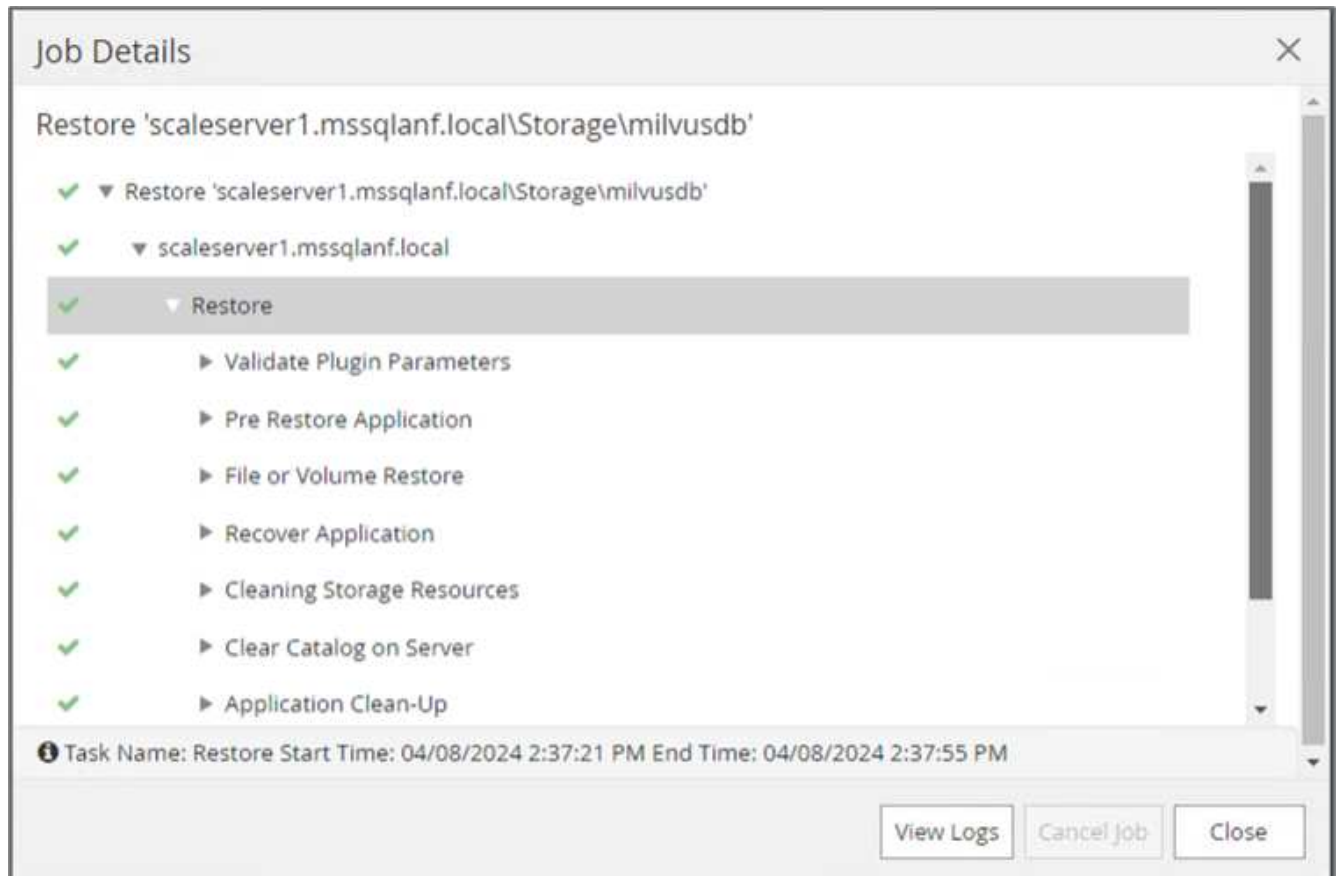
Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== 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 복구를 실행합니다.



10. Python 스크립트를 사용하여 'hello_milvus_netapp_SC_test' 및 'hello_milvus_netapp_SC_test2' 컬렉션에서 데이터를 확인하십시오.

```
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 수집이 데이터베이스에 더 이상 존재하지 않는지 확인합니다.

```

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:~#

```

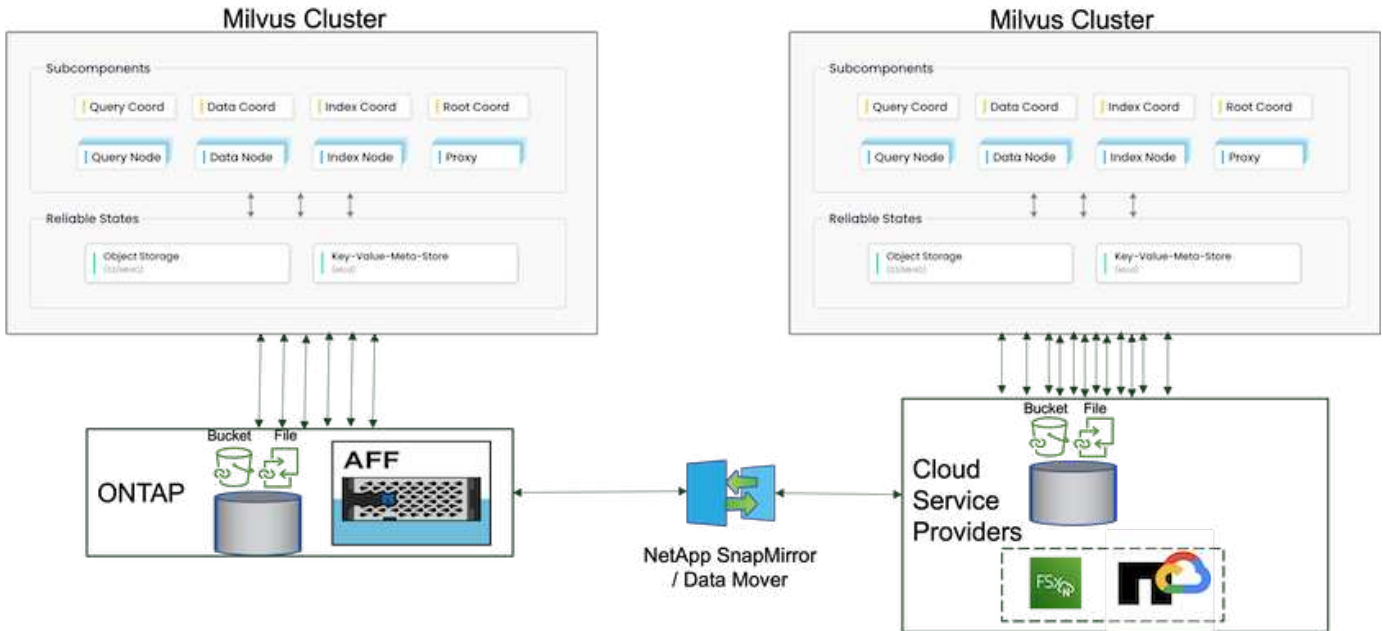
결론적으로, NetApp의 SnapCenter를 사용하여 벡터 데이터베이스 데이터와 ONTAP에 상주하는 Milvus 데이터를 보호하면 고객에게 상당한 혜택을 제공할 수 있으며, 특히 데이터 무결성이 영화 제작과 같이 중요한 산업에서 그 효과를 볼 수 있습니다. SnapCenter는 일관된 백업을 생성하고 전체 데이터 복원을 수행할 수 있으므로 내장 비디오 및 오디오 파일과 같은 중요한 데이터가 하드 드라이브 장애 또는 기타 문제로 인한 손실을 방지합니다. 이는 운영 중단을 방지할 뿐만 아니라 상당한 재정적 손실을 방지합니다.

이 섹션에서는 호스트 설정, 스토리지 플러그인 설치 및 구성, 사용자 지정 스냅샷 이름을 사용하여 벡터 데이터베이스에 대한 리소스 생성을 포함하여 ONTAP에 있는 데이터를 보호하도록 SnapCenter를 구성하는 방법을 살펴보았습니다. 또한 정합성 보장 그룹 스냅샷을 사용하여 백업을 수행하고 S3 NAS 버킷에서 데이터를 확인하는 방법에 대해서도 설명했습니다.

또한 백업 후 불필요하거나 부적절한 수집이 생성된 시나리오를 시뮬레이션했습니다. 이 경우 SnapCenter는 이전 스냅샷에서 전체 복원을 수행할 수 있으므로 새 컬렉션을 추가하기 전의 상태로 벡터 데이터베이스를 되돌릴 수 있으므로 데이터베이스의 무결성을 유지할 수 있습니다. 데이터를 특정 시점으로 복원하는 이러한 기능은 고객이 데이터를 안전하게 보호할 수 있을 뿐만 아니라 올바르게 유지 관리한다는 확신을 바탕으로 고객에게 매우 중요합니다. 따라서 NetApp의 SnapCenter 제품은 데이터 보호 및 관리를 위한 강력하고 안정적인 솔루션을 고객에게 제공합니다.

NetApp SnapMirror를 사용한 재해 복구

NetApp SnapMirror를 사용한 재해 복구



재해 복구는 벡터 데이터베이스의 무결성과 가용성을 유지하는 데 매우 중요합니다. 특히, 높은 차원 데이터를 관리하고 복잡한 유사성 검색을 실행하는 데 그 역할을 합니다. 재해 복구 전략을 잘 계획하고 구현하면 하드웨어 장애, 자연 재해 또는 사이버 공격과 같은 예상치 못한 사고가 발생할 경우 데이터가 손실되거나 손상되지 않습니다. 이는 특히 데이터 손실 또는 손상이 중대한 운영 중단 및 재정적 손실을 초래할 수 있는 벡터 데이터베이스를 사용하는 애플리케이션에 중요합니다. 또한 강력한 재해 복구 계획을 통해 다운타임을 최소화하고 신속한 서비스 복원을 가능하게 하여 비즈니스 연속성을 보장합니다. 이 기능은 서로 다른 지리적 위치, 정기적인 백업 및 장애 조치 메커니즘에 걸쳐 NetApp 데이터 복제 제품 SnapMirror를 통해 구현됩니다. 따라서 재해 복구는 단순한 보호 수단이 아니라 책임감 있고 효율적인 벡터 데이터베이스 관리의 중요한 구성 요소입니다.

NetApp의 SnapMirror는 한 NetApp ONTAP 스토리지 컨트롤러에서 다른 컨트롤러로 데이터 복제를 제공하고, 이는 DR(재해 복구) 및 하이브리드 솔루션에 주로 사용됩니다. 벡터 데이터베이스의 컨텍스트에서 이 툴을 사용하면 온프레미스와 클라우드 환경 간에 데이터를 원활하게 전환할 수 있습니다. 이러한 전환은 데이터 변환 또는 애플리케이션 리팩토링 없이 수행되므로 여러 플랫폼에서 데이터 관리의 효율성과 유연성을 향상할 수 있습니다.

벡터 데이터베이스 시나리오의 NetApp 하이브리드 솔루션은 다음과 같은 점에서 더 많은 이점을 제공합니다.

1. 확장성: NetApp의 하이브리드 클라우드 솔루션은 고객의 요구사항에 따라 리소스를 확장할 수 있는 기능을 제공합니다. 온프레미스 리소스를 NetApp ONTAP용 Amazon FSxN, Google Cloud NetApp Volume(GCNV)과 같은 정기적인 예측 가능한 워크로드 및 클라우드 리소스를 활용하여 사용량이 가장 많은 시간 또는 예상치 못한 부하에 대응할 수 있습니다.
2. 비용 효율성: NetApp의 하이브리드 클라우드 모델을 활용하면 일반 워크로드에 온프레미스 리소스를 사용하고 필요할 때만 클라우드 리소스에 대한 비용을 지불함으로써 비용을 최적화할 수 있습니다. NetApp instaclustr 서비스 오퍼링을 사용하면 이 용량제 모델을 비용 효율적으로 사용할 수 있습니다. instaclustr는 온프레미스 및 주요 클라우드 서비스 공급자를 위해 지원과 상담을 제공합니다.
3. 유연성: NetApp의 하이브리드 클라우드를 사용하면 데이터를 처리할 위치를 유연하게 선택할 수 있습니다. 예를 들어, 클라우드에서 보다 강력한 하드웨어와 덜 집중적인 작업을 수행하는 복잡한 벡터 작업을 사내에서 수행하도록 선택할 수 있습니다.
4. 비즈니스 연속성: 재해 발생 시 NetApp 하이브리드 클라우드에 데이터를 저장하면 비즈니스 연속성을 보장할 수 있습니다. 사내 리소스에 영향을 미치는 경우 클라우드로 빠르게 전환할 수 있습니다. NetApp SnapMirror를 활용하여 사내에서 클라우드로 또는 그 반대로 데이터를 이동할 수 있습니다.
5. 혁신: NetApp의 하이브리드 클라우드 솔루션은 최첨단 클라우드 서비스 및 기술에 대한 액세스를 제공하여 더 빠르게 혁신을 지원할 수 있습니다. NetApp ONTAP, Azure NetApp Files, Google Cloud NetApp Volumes용

Amazon FSxN과 같은 클라우드의 NetApp 혁신은 클라우드 서비스 공급자의 혁신적인 제품과 선호하는 NAS입니다.

Vector 데이터베이스 성능 검증

성능 검증

성능 검증은 벡터 데이터베이스와 스토리지 시스템 모두에서 중요한 역할을 하며 최적의 운영과 효율적인 리소스 활용도를 보장하는 핵심 요소로 작용합니다. 높은 차원 데이터를 처리하고 유사성 검색을 실행하는 것으로 알려진 벡터 데이터베이스는 복잡한 쿼리를 신속하고 정확하게 처리하기 위해 높은 성능 수준을 유지해야 합니다. 성능 검증은 병목 현상을 식별하고 구성을 세밀하게 조정하고 시스템이 서비스 성능 저하 없이 예상 로드를 처리할 수 있도록 하는 데 도움이 됩니다. 마찬가지로 스토리지 시스템에서 성능 검증은 전체 시스템 성능에 영향을 줄 수 있는 지연 시간 문제나 병목 현상 없이 데이터를 효율적으로 저장하고 검색하는 데 필수적입니다. 또한 정보에 근거하여 스토리지 인프라의 필요한 업그레이드 또는 변경 결정을 내릴 수 있도록 지원합니다. 따라서 성능 검증은 시스템 관리의 중요한 측면으로, 높은 서비스 품질, 운영 효율성 및 전반적인 시스템 안정성을 유지하는 데 크게 기여합니다.

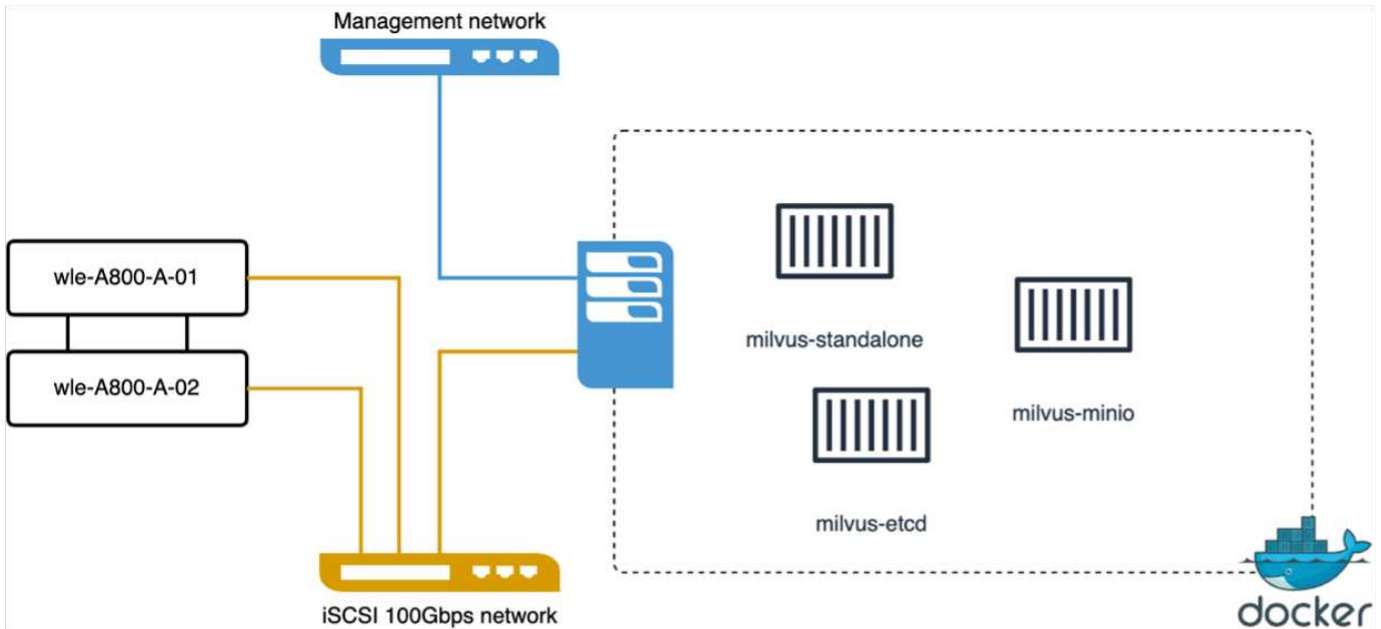
이 섹션에서는 Milvus 및 pgvecto.RS와 같은 벡터 데이터베이스의 성능 검증을 자세히 살펴보고, LLM 수명주기 내의 RAG 및 추론 워크로드를 지원하는 I/O 프로파일 및 NetApp 스토리지 컨트롤러 같은 스토리지 성능 특성에 초점을 맞추고자 합니다. 이들 데이터베이스가 ONTAP 스토리지 솔루션과 결합될 때 성능 차별화 요소를 평가하고 식별합니다. 당사의 분석은 QPS(초당 처리된 쿼리 수)와 같은 주요 성능 지표를 기반으로 합니다.

아래 Milvus에 사용되는 방법론과 진행 상황을 확인하십시오.

세부 정보	Milvus(독립 실행형 및 클러스터)	Postgres(pgvecto.rs)
버전	2.3.2	0.2.0
파일 시스템	iSCSI LUN의 XFS	
워크로드 생성기	"벡터DB-벤치" -v0.0.5	
데이터 세트	LAION 데이터 세트 * 10,000,000개의 침구 * 768 치수 * ~ 300GB 데이터 세트 크기	

VectorDB - Milvus 독립 실행형 클러스터를 사용한 벤치

vectorDB-Bench를 사용하는 milvus 독립 실행형 클러스터에 대해 다음과 같은 성능 검증을 수행했습니다. milvus 독립 실행형 클러스터의 네트워크 및 서버 연결은 다음과 같습니다.



이 섹션에서는 Milvus 독립 실행형 데이터베이스 테스트의 관찰 결과 및 결과를 공유합니다.

. 이러한 테스트의 인덱스 유형으로 DiskANN을 선택했습니다.

. 약 100GB의 데이터 세트에 대한 인덱스를 수집, 최적화 및 생성하는 데 약 5시간이 걸렸습니다. 대부분의 기간 동안 20개의 코어(하이퍼 스레딩을 사용할 경우 40개의 vCPU와 동일)를 장착한 Milvus 서버는 최대 CPU 용량인 100%에서 작동하고 있었습니다. DiskANN은 특히 시스템 메모리 크기를 초과하는 대규모 데이터세트에 매우 중요합니다.

. 조회 단계에서 0.9987을 리콜한 상태에서 QPS(Queries per second)가 10.93인 것을 확인했습니다. 쿼리의 99번째 백분위수 지연 시간은 708.2밀리초로 측정되었습니다.

스토리지 관점에서 볼 때 데이터베이스는 수집, 삽입 후 최적화, 인덱스 생성 단계에서 약 1,000 ops/sec를 기록했습니다. 쿼리 단계에서는 32,000 ops/초가 필요했습니다

다음 섹션에서는 스토리지 성능 메트릭에 대해 설명합니다.

워크로드 단계	미터	값
데이터 수집 및 삽입 후 최적화	IOPS	1,000명 이하
	지연 시간	400개 이상의 활용
	워크로드	읽기/쓰기 조합, 주로 쓰기입니다
	IO 크기입니다	64KB
쿼리	IOPS	32,000에서 최고치
	지연 시간	400개 이상의 활용
	워크로드	100% 캐시된 읽기
	IO 크기입니다	주로 8KB

vectorDB-bench 결과는 다음과 같습니다.

Vector Database Benchmark

Filtering Search Performance Test (5M Dataset, 1536 Dim, Filter 1%) ^

Qps (more is better)

Milvus  10.93

Recall (more is better)

Milvus  0.9987

Load_duration (less is better)

Milvus  18.360s

Serial_latency_p99 (less is better)

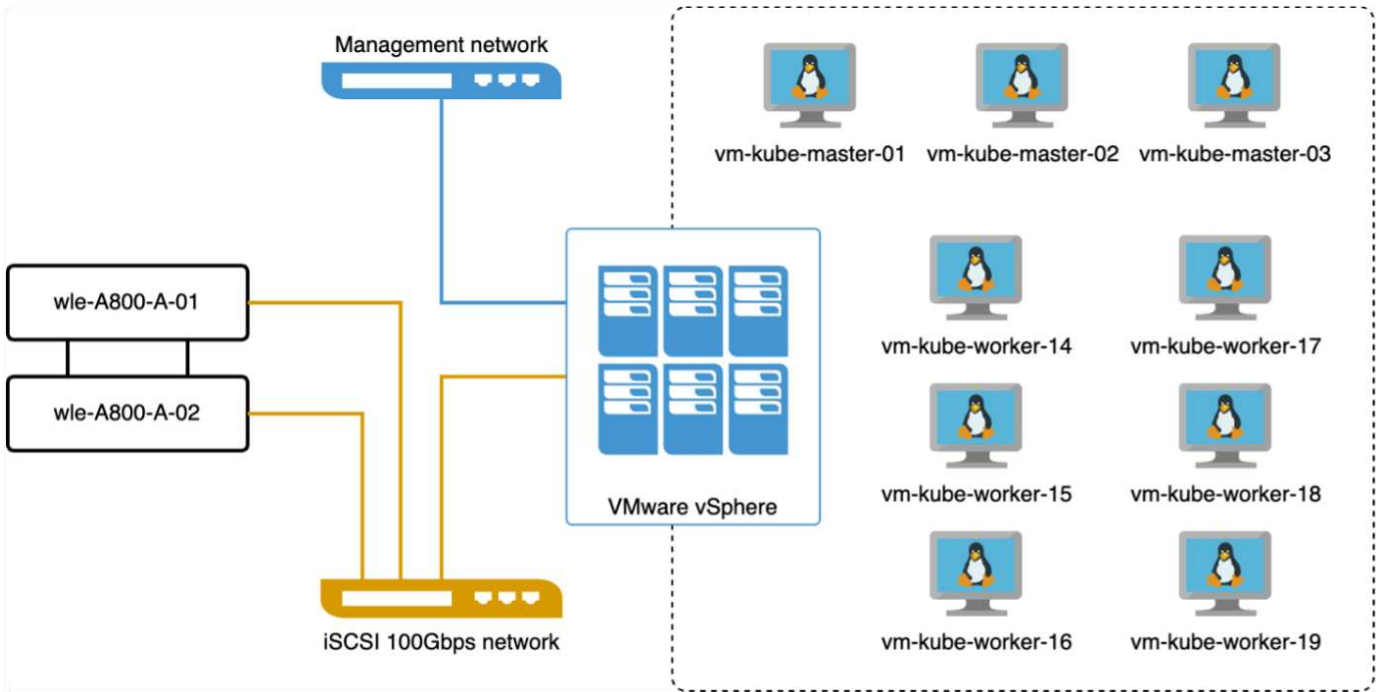
Milvus  708.2ms

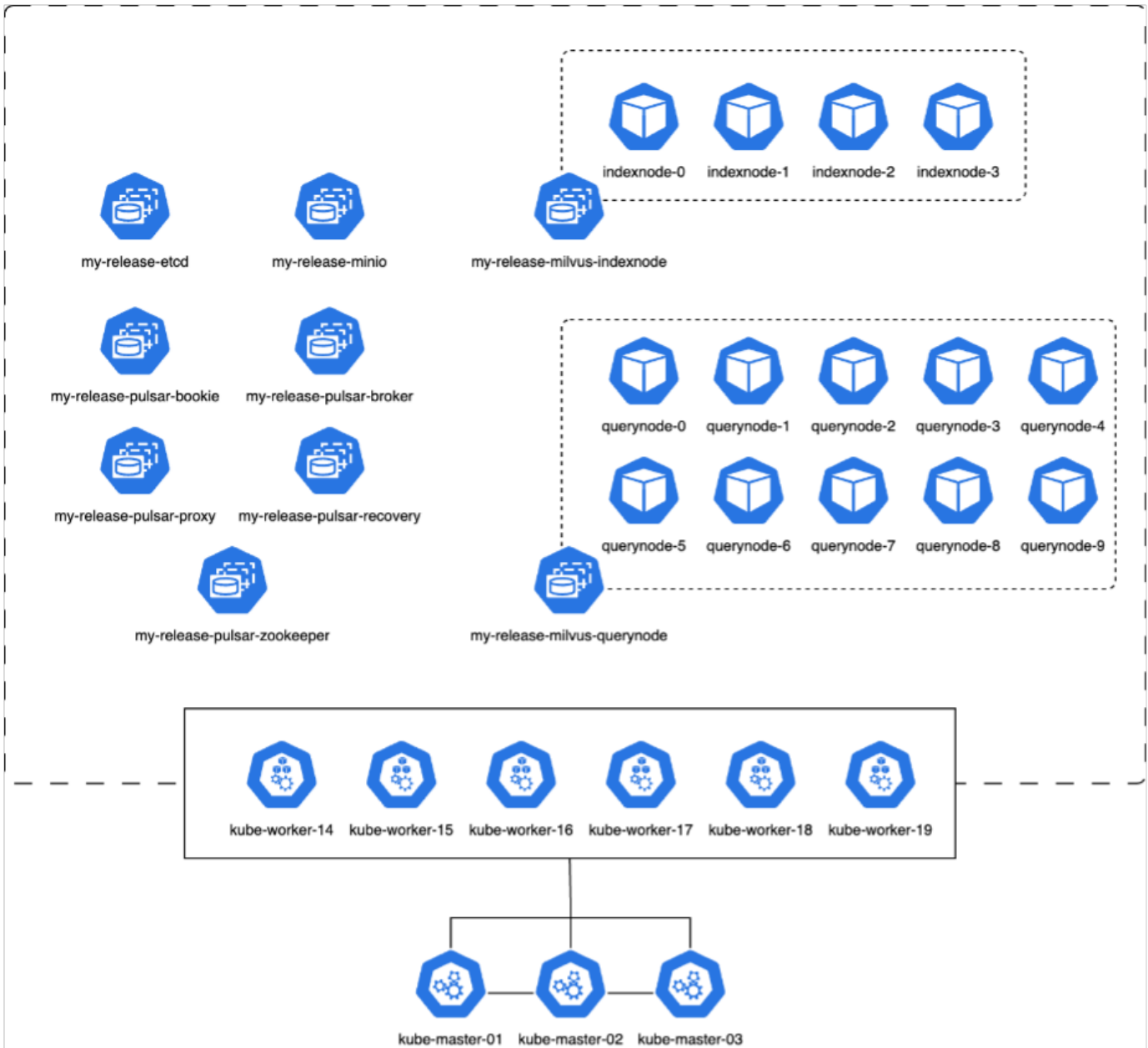
독립 실행형 Milvus 인스턴스의 성능 검증을 통해 현재 설정이 치수 1536의 5백만 벡터로 구성된 데이터 세트를 지원하기에는 불충분하다는 것을 알 수 있습니다. 스토리지에 적절한 리소스가 있으며 시스템에 병목 현상이 발생하지 않는 것으로 확인되었습니다.

VectorDB - 밀버스 클러스터를 사용한 벤치

이 섹션에서는 Kubernetes 환경 내에 Milvus 클러스터를 구축하는 방법에 대해 설명합니다. 이 Kubernetes 설정은 Kubernetes 마스터 및 작업자 노드를 호스팅하는 VMware vSphere 배포를 바탕으로 구성되었습니다.

VMware vSphere 및 Kubernetes 구축에 대한 자세한 내용은 다음 섹션에서 설명합니다.





이 섹션에서는 Milvus 데이터베이스 테스트에서 얻은 관찰 내용과 결과를 설명합니다.

*사용된 인덱스 유형은 DiskANN입니다.

*아래 표는 1536의 치수에서 5백만 벡터로 작업할 때 독립형 배포와 클러스터 배포를 비교한 것입니다. 우리는 데이터 수집 및 삽입 후 최적화에 걸리는 시간이 클러스터 구축에서 더 낮은 것으로 확인되었습니다. 독립 실행형 설정에 비해 클러스터 구축 시 쿼리의 99번째 백분위수 지연 시간이 6배 감소했습니다.

*클러스터 배포에서 QPS(Queries per Second) 비율이 높았지만 원하는 수준에서는 그렇지 않았습니다.

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 시간 및 총 IOPS(초당 입출력 작업 수)를 포함한 다양한 스토리지 메트릭을 보여줍니다.



다음 섹션에서는 주요 스토리지 성능 메트릭에 대해 설명합니다.

워크로드 단계	미터	값
데이터 수집 및 삽입 후 최적화	IOPS	1,000명 이하
	지연 시간	400개 이상의 활용
	워크로드	읽기/쓰기 조합, 주로 쓰기입니다
	IO 크기입니다	64KB
쿼리	IOPS	147,000에서 최고점
	지연 시간	400개 이상의 활용
	워크로드	100% 캐시된 읽기
	IO 크기입니다	주로 8KB

독립 실행형 Milvus 및 Milvus 클러스터의 성능 검증을 토대로 스토리지 I/O 프로필에 대한 세부 정보를 제공합니다.

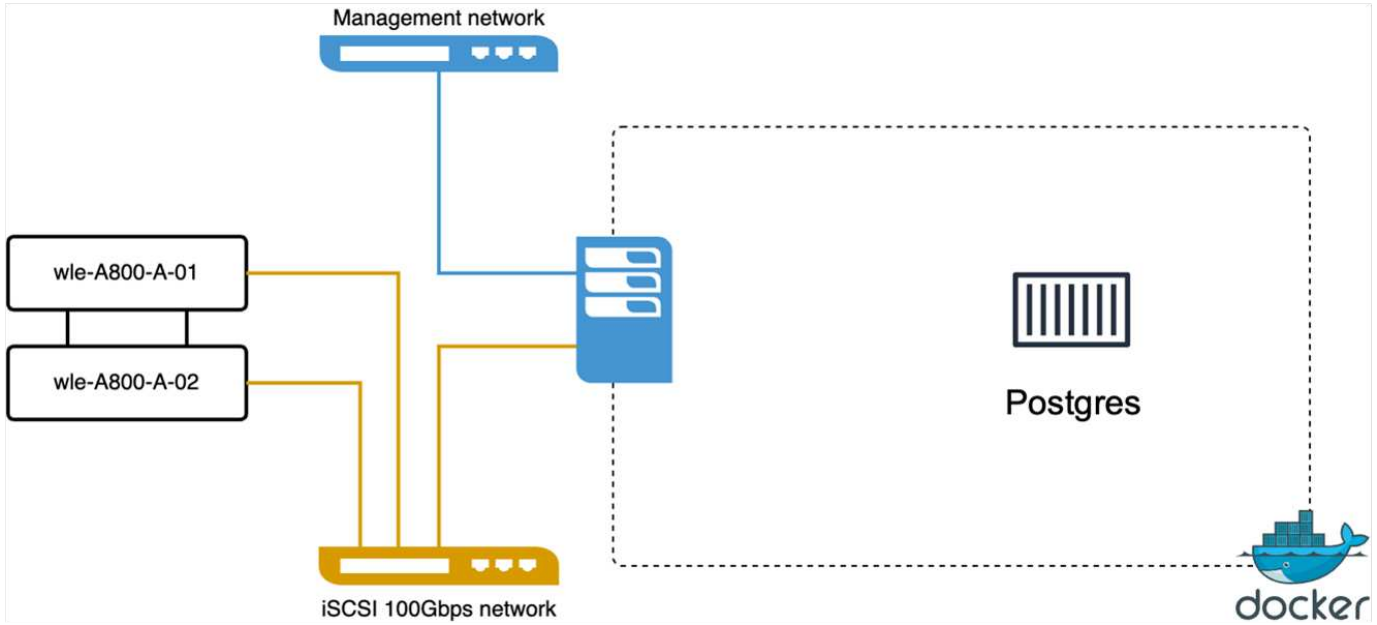
* 독립 실행형 및 클러스터 구축 모두에서 I/O 프로필이 일관되게 유지되는 것을 확인했습니다.

* 피크 IOPS에서 관찰된 차이는 클러스터 구축 시 클라이언트 수가 많기 때문입니다.

Postgres가 있는 vectorDB-벤치(pgvector.rs)

VectorDB-Bench를 사용하여 PostgreSQL(pgvector.rs)에 대해 다음 작업을 수행했습니다.

PostgreSQL(특히 pgvector.rs)의 네트워크 및 서버 연결에 대한 세부 정보는 다음과 같습니다.



이 섹션에서는 특히 pgvecto.rs를 사용하여 PostgreSQL 데이터베이스를 테스트한 결과 및 관찰 결과를 공유합니다.

- * 테스트 당시 DiskANN은 pgvecto.RS에 사용할 수 없었기 때문에 이러한 테스트의 인덱스 유형으로 HNSW를 선택했습니다.
- * 데이터 수집 단계 동안, 우리는 768의 치수에서 천만 벡터로 구성된 COHERE 데이터셋을 로드했습니다. 이 과정은 약 4.5시간이 걸렸습니다.
- * 쿼리 단계에서 0.6344를 리콜하여 1,068의 QPS(Queries per Second)를 확인했습니다. 쿼리의 99번째 백분위수 지연 시간은 20밀리초로 측정되었습니다. 대부분의 런타임 동안 클라이언트 CPU는 100% 용량으로 작동했습니다.

아래 이미지는 스토리지 클러스터 지연 시간 총 IOPS(초당 입출력 작업 수)를 포함한 다양한 스토리지 메트릭을 보여 줍니다.

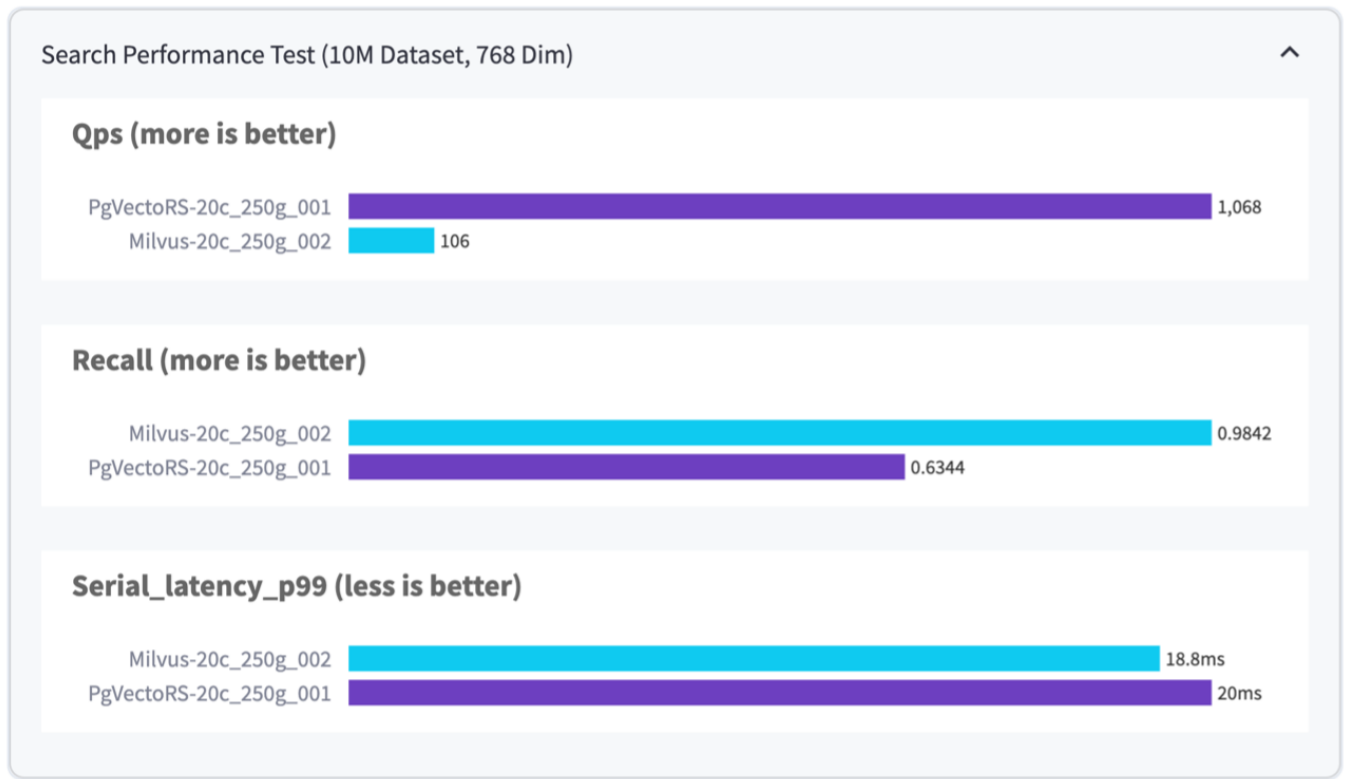


The following section presents the key storage performance metrics.
 image:pgvecto_storage_perf_metrics.png["오류: 그래픽 이미지가 없습니다"]

벡터 DB 벤치의 밀버스와 포스트그레스의 성능 비교

Vector Database Benchmark

Note that all testing was completed in July 2023, except for the times already noted.



VectorDBBench를 사용한 Milvus 및 PostgreSQL의 성능 검증을 토대로 다음과 같은 점을 관찰했습니다.

- 인덱스 유형: HNSW
- 데이터 세트: 768차원으로 1,000만 벡터를 사용하는 COHERE

우리는 pgvecto.RS가 0.6344의 리콜로 1,068의 QPS(Queries per Second)를 달성했으며, Milvus는 0.9842의 리콜로 106의 QPS 속도를 달성했습니다.

쿼리의 높은 정밀도가 우선 순위인 경우 Milvus는 쿼리당 관련 항목의 비율이 더 높기 때문에 pgvecto.rs보다 성능이 뛰어납니다. 그러나 초당 쿼리 수가 더 중요한 요소인 경우 pgvecto.RS는 Milvus를 초과합니다. 하지만 pgvecto.rs를 통해 검색된 데이터의 품질이 낮고 검색 결과의 약 37%가 관련 없는 항목이라는 점을 유의해야 합니다.

성능 검증에 따른 관찰:

성능 검증을 토대로 다음과 같이 관찰했습니다.

Milvus의 I/O 프로파일은 Oracle SLOB에서 볼 수 있는 OLTP 워크로드와 비슷합니다. 벤치마크는 데이터 수집, 사후 최적화 및 쿼리의 세 단계로 구성됩니다. 초기 단계는 주로 64KB 쓰기 작업이 특징이며, 쿼리 단계에는 대개 8KB

읽기가 포함됩니다. ONTAP는 Milvus I/O 로드를 능숙하게 처리할 것으로 기대하고 있습니다.

PostgreSQL 입출력 프로파일은 까다로운 스토리지 워크로드를 제공하지 않습니다. 현재 인메모리 구현이 진행 중이라는 점을 감안할 때 쿼리 단계에서 디스크 입출력을 관찰하지 못했습니다.

DiskANN은 스토리지 차별화를 위한 중요한 기술로 등장했습니다. 시스템 메모리 경계를 넘어 벡터 DB 검색의 효율적인 확장을 지원합니다. 그러나 HNSW와 같은 인메모리 벡터 DB 인덱스와 스토리지 성능 차이를 구별할 가능성은 거의 없습니다.

또한 인덱스 유형이 RAG 애플리케이션을 지원하는 벡터 데이터베이스의 가장 중요한 작동 단계인 HSNW인 경우 쿼리 단계에서 스토리지가 중요한 역할을 수행하지 않는다는 점도 주목할 필요가 있습니다. 여기서 중요한 점은 스토리지 성능이 이러한 애플리케이션의 전체 성능에 크게 영향을 미치지 않는다는 것입니다.

저작권 정보

Copyright © 2024 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 라이선스와 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 라이선스도 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 라이선스를 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 라이선스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

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