



解決策検証の概要

NetApp Solutions

NetApp
May 03, 2024

目次

解決策の概要	1
オンプレミスでのKubernetesを使用したMilvusクラスタセットアップ	1
MilvusとAmazon FSxN for NetApp ONTAP -ファイルとオブジェクトの二重性	8
SnapCenterを使用したベクターデータベース保護	15
NetApp SnapMirrorを使用したディザスタリカバリ	25
Vectorデータベースのパフォーマンス検証	27

解決策の概要

当社は、以下の5つの主要分野に焦点を当てた包括的な解決策検証を実施しました。各セクションでは、お客様が直面している課題、NetAppが提供するソリューション、その後お客様にもたらされるメリットについて詳しく説明します。

1. **"オンプレミスでのKubernetesを使用したMilvusクラスタセットアップ"**
ストレージとコンピューティングの個別拡張、効果的なインフラ管理、データ管理に関するお客様の課題
このセクションでは、クラスタデータと顧客データの両方にNetAppストレージコントローラを利用して、KubernetesにMilvusクラスタをインストールするプロセスを詳しく説明します。
2. **"MilvusとAmazon FSxN for NetApp ONTAP-ファイルとオブジェクトの二重性"**
このセクションでは、クラウドにベクターデータベースをデプロイする必要がある理由と、Dockerコンテナ内のAmazon FSxN for NetApp ONTAPにベクターデータベース (milvusスタンドアロン) をデプロイする手順について説明します。
3. **"NetApp SnapCenterを使用したベクターデータベース保護。"**
このセクションでは、SnapCenterがONTAPに存在するベクターデータベースデータとMilvusデータをどのように保護するかについて詳しく説明します。この例では、顧客データ用にNFS ONTAPボリューム (vol1) から派生したNASバケット (milvusdbvol1) を使用し、Milvusクラスタ構成データ用に別のNFSボリューム (vectordbvp) を使用しました。
4. **"NetApp SnapMirrorを使用したディザスタリカバリ"**
このセクションでは、ベクターデータベースに対するディザスタリカバリ (DR) の重要性と、NetAppディザスタリカバリ製品のSnapMirrorがベクターデータベースに対するDR解決策をどのように提供するかについて説明します。
5. **"パフォーマンスの検証"**
このセクションでは、Milvusやpgvecto.RSなどのベクターデータベースのパフォーマンス検証について、LLMライフサイクル内でのRAGおよび推論ワークロードをサポートする際のI/OプロファイルやNetAppストレージコントローラの動作などのストレージパフォーマンス特性に焦点を当てて詳しく説明します。これらのデータベースをONTAPストレージ解決策と組み合わせた場合のパフォーマンスの差別化要因を評価し、特定します。分析は、1秒あたりに処理されるクエリ数 (QPS) などの主要パフォーマンス指標に基づいて行われます。

オンプレミスでのKubernetesを使用したMilvusクラスタセットアップ

オンプレミスでのKubernetesを使用したMilvusクラスタセットアップ

ストレージとコンピューティングの個別拡張、効果的なインフラ管理とデータ管理、Kubernetesデータベースとベクトルデータベースは、大規模なデータ処理を管理するための、強力で拡張性に優れた解決策を形成します。Kubernetesはリソースを最適化してコンテナを管理し、ベクトルデータベースは高次元データや類似性検索を効率的に処理します。この組み合わせにより、大規模なデータセットで複雑なクエリを迅速に処理し、データ量の増大に合わせてシームレスに拡張できるため、ビッグデータアプリケーションやAIワークロードに最適です。

1. このセクションでは、クラスタデータと顧客データの両方にNetAppストレージコントローラを利用して、KubernetesにMilvusクラスタをインストールするプロセスを詳しく説明します。
2. Milvusクラスタをインストールするには、さまざまなMilvusクラスタコンポーネントからのデータを格納するためにPersistent Volumes (PVS) が必要です。これらのコンポーネントには、etcd (3つのインスタンス)、pulsar-bookie-journal (3つのインスタンス)、pulsar-bookie-ledgers (3つのインスタンス)

)、pulsar-zookeeper-data (3つのインスタンス) があります。



milvusクラスターでは、Milvusクラスターの信頼性の高いストレージとメッセージストリームのパブリッシュ/サブスクリプションをサポートする基盤となるエンジンにパルサーまたはKafkaのいずれかを使用できます。NFSを使用するKafkaについては、ONTAP 9.12.1以降で改善が行われています。これらの機能強化と、RHEL 8.7または9.1以降でのNFSv4.1およびLinuxの変更に加えて、NFSでKafkaを実行するときに発生する「silly rename」問題を解決してください。NetApp NFS解決策でKafkaを実行する方法の詳細については、<https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html>。

3. NetApp ONTAPからNFSボリュームを1つ作成し、250GBのストレージを含む12個の永続ボリュームを構築しました。ストレージサイズはクラスタサイズによって異なります。たとえば、各PVが50GBの別のクラスタがあります。詳細については、以下のいずれかのPV YAMLファイルを参照してください。このようなファイルは合計12個ありました。各ファイルでは、storageClassNameが「default」に設定され、ストレージとパスはPVごとに一意です。

```
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. 各PV YAMLファイルに対して「kubectl apply」コマンドを実行して永続ボリュームを作成し、「kubectl get pv」を使用して作成を確認します。

```
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. 顧客データの格納については、MinIO、Azure Blob、S3などのオブジェクトストレージソリューションをサポートしています。このガイドではS3を使用します。次の手順は、ONTAP S3とStorageGRIDオブジェクトストアの両方に適用されます。Helmを使用してMilvusクラスタを導入します。Milvusのダウンロード場所から、設定ファイルvalues.yamlをダウンロードします。このドキュメントで使用したvalues.yamlファイルについては、付録を参照してください。
6. log、etcd、zookeeper、およびbookkeeperのセクションを含め、「storageClass」が各セクションで「default」に設定されていることを確認します。
7. MinIOセクションで、MinIOを無効にします。
8. ONTAPまたはStorageGRIDオブジェクトストレージからNASバケットを作成し、オブジェクトストレージのクレデンシャルを使用して外部S3に追加します。

```
#####
# 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クラスタを作成する前に、PersistentVolumeClaim (PVC) に既存のリソースがないことを確認してください。

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

10. Helmとvalues.yaml設定ファイルを使用して、Milvusクラスタをインストールして起動します。

```
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) のステータスを確認します。

```

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. ポッドのステータスを確認します。

```

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

```

ポッドのステータスが「Running」で正常に機能していることを確認してください

13. MilvusおよびNetAppオブジェクトストレージでデータの書き込みと読み取りをテストします。

- Pythonプログラム「prepare_data_netapp_new.py」を使用してデータを書き込みます。

```

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:~#

```

- Pythonファイル「verify_data_netapp.py」を使用してデータを読み取ります。

```

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}}]}

```

上記の検証に基づいて、NetAppストレージコントローラを使用したKubernetes上にMilvusクラスタを導入することで実証されているように、Kubernetesとベクトルデータベースの統合により、大規模なデータ操作を管理するための堅牢でスケラブルで効率的な解決策が提供されます。このセットアップにより、お客様は高次元データを処理し、複雑なクエリを迅速かつ効率的に実行できるようになり、ビッグデータアプリケーションやAIワークロードに最適な解決策になります。さまざまなクラスタコンポーネントにPersistent Volume (PV; 永続的ボリューム) を使用して、NetApp ONTAPから単一のNFSボリュームを作成することで、リソース利用率とデータ管理を最適化できます。PersistentVolumeClaims (PVC) とPODのステータスを検証し、データの書き込みと読み取りを

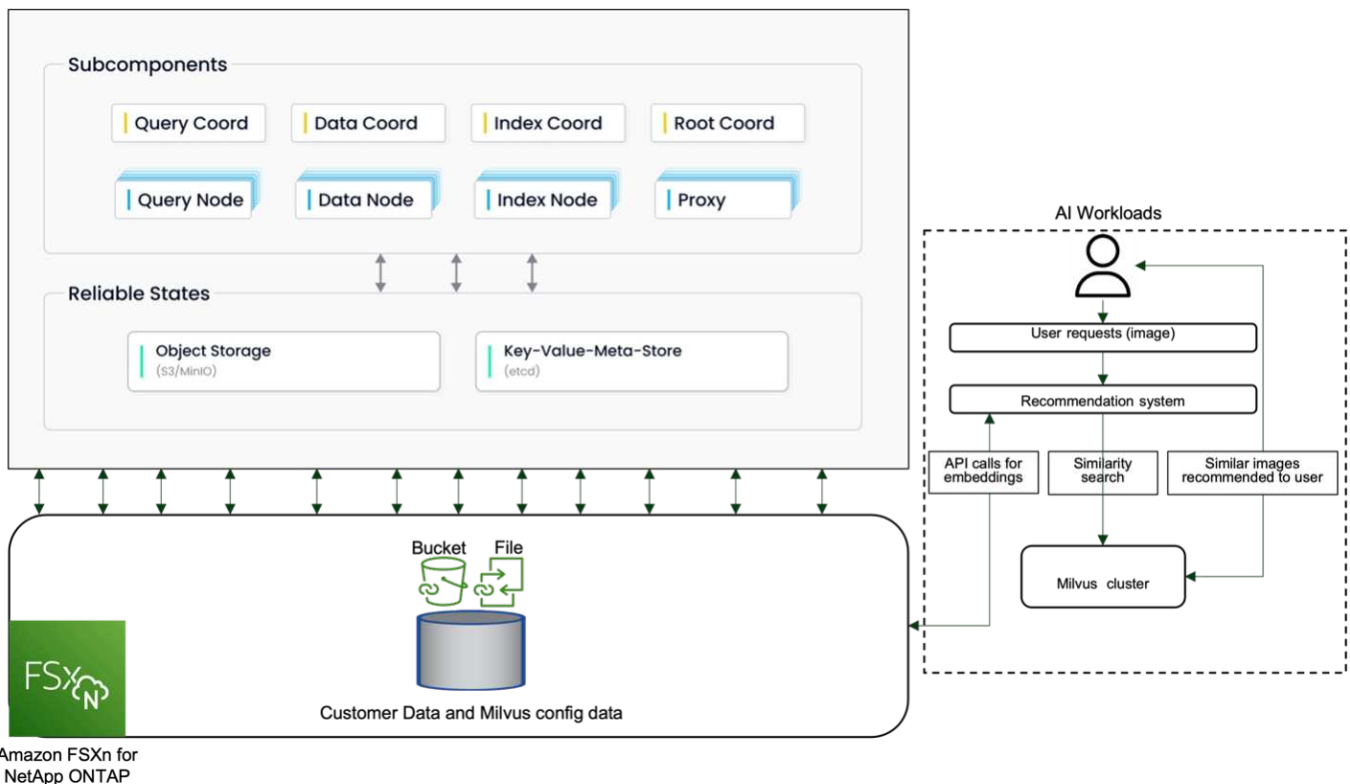
テストするプロセスにより、信頼性が高く一貫したデータ処理が保証されます。お客様のデータにON TAPまたはStorageGRIDオブジェクトストレージを使用すると、データへのアクセス性とセキュリティがさらに向上します。全体的に、このセットアップにより、お客様は、増大するデータニーズに合わせてシームレスに拡張できる耐障害性とパフォーマンスに優れたデータ管理解決策を利用できます。

MilvusとAmazon FSxN for NetApp ONTAP -ファイルとオブジェクトの二重性

MilvusとAmazon FSxN for NetApp ONTAP-ファイルとオブジェクトの二重性

このセクションでは、クラウドにベクターデータベースをデプロイする必要がある理由と、Dockerコンテナ内のAmazon FSxN for NetApp ONTAPにベクターデータベース(milvusスタンドアロン)をデプロイする手順について説明します。

クラウドにベクターデータベースを配置すると、特に高次元データを処理し、類似性検索を実行する必要があるアプリケーションには、いくつかの大きな利点があります。1つ目は、クラウドベースの導入により拡張性が向上し、増大するデータボリュームやクエリの負荷に合わせてリソースを簡単に調整できることです。これにより、データベースは高いパフォーマンスを維持しながら、増加する需要に効率的に対処できます。2つ目は、クラウドを導入することで、地理的に離れた場所にデータをレプリケートできるため、高可用性とディザスタリカバリが実現し、データ損失のリスクが最小限に抑えられ、予期しないイベントが発生しても継続的なサービスが提供されることです。3つ目は、使用したリソースに対してのみ料金が発生するため、対費用効果が高く、必要に応じてスケールアップまたはスケールダウンできるため、ハードウェアへの多額の先行投資が不要です。最後に、クラウドにベクターデータベースを導入すると、どこからでもデータにアクセスして共有できるため、コラボレーションが強化され、チームベースの作業やデータ主導の意思決定が容易になります。この検証で使用したAmazon FSxN for NetApp ONTAPを使用したmilvusスタンドアロンのアーキテクチャを確認してください。



1. Amazon FSxN for NetApp ONTAPインスタンスを作成し、VPC、VPCセキュリティグループ、およびサブネットの詳細をメモします。この情報は、EC2インスタンスを作成するときに必要になります。詳細については、こちらをご覧ください。 <https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create>
2. EC2インスタンスを作成し、VPC、セキュリティグループ、およびサブネットがAmazon FSxN for NetApp ONTAPインスタンスのものと同じようにします。
3. コマンド「`apt-get install nfs-common`」を使用してnfs-commonをインストールし、「`sudo apt-get update`」を使用してパッケージ情報を更新します。
4. マウントフォルダを作成し、そのフォルダにAmazon FSxN for NetApp ONTAPをマウントします。

```
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'を使用してDockerとDocker Composeをインストールします。
6. Docker-compose.ymlファイルに基づいてMilvusクラスタをセットアップします。このファイルはMilvus Webサイトからダウンロードできます。

```
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ファイルの「volumes」セクションで、NetApp NFSマウントポイントに対応するMilvusコンテナパス（具体的にはetcd、minio、standalone）にマッピングします。Check "[付録D : docker-compose.yml](#)" ymlの変更の詳細については、を参照してください。
8. マウントされたフォルダとファイルを確認します。

```

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
vectordbvoll
ubuntu@ip-172-31-29-98:~$

```

9. docker-compose.ymlファイルが格納されているディレクトリから'docker-compose up -d'を実行します。
10. Milvusコンテナのステータスを確認します。

```

ubuntu@ip-172-31-29-98:~$ sudo docker-compose ps

```

Name	Command	State

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. VECTORデータベースの読み書き機能とAmazon FSxN for NetApp ONTAPのデータを検証するために、Python Milvus SDKとPyMilvusのサンプルプログラムを使用しました。'apt-get install python3-numpy python3-pip'を使用して必要なパッケージをインストールし、'pip3 install pymilvus'を使用してPyMilvusをインストールします。
12. VECTORデータベースのAmazon FSxN for NetApp ONTAPからのデータの書き込みおよび読み取り操作を

検証します。

```
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スクリプトを使用して読み取り操作を確認します。

```
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. お客様が、AIワークロード用にS3プロトコルを介してVECTORデータベースでテストされたNFSデータにアクセスしたい（読み取り）場合は、わかりやすいPythonプログラムを使用して検証できます。この例として、このセクションの冒頭の図で説明したように、別のアプリケーションからの画像の類似性検索があります。

```

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#
```


このセクションでは、AmazonのNetApp FSxN for NetApp ONTAPデータストレージを利用して、Dockerコンテナ内にスタンドアロンのMilvusセットアップを導入および運用する方法を効果的に説明します。このセットアップにより、ベクトルデータベースの機能を活用して高次元データを処理し、複雑なクエリを実行できます。これらはすべて、Dockerコンテナのスケーラブルで効率的な環境内で実行できます。Amazon FSxN for NetApp ONTAPインスタンスを作成し、EC2インスタンスを一致させることで、リソース利用率とデータ管理を最適化できます。VECTORデータベースでFSxNからのデータの書き込みおよび読み取り操作を正しく検証することで、信頼性が高く一貫したデータ操作を保証できます。さらに、AIワークロードからS3プロトコルを介してデータをリスト（読み取り）できるため、データへのアクセスが強化されます。したがって、この包括的なプロセスは、AmazonのFSxN for NetApp ONTAPの機能を活用して、大規模なデータ操作を管理するための堅牢で効率的な解決策を顧客に提供します。

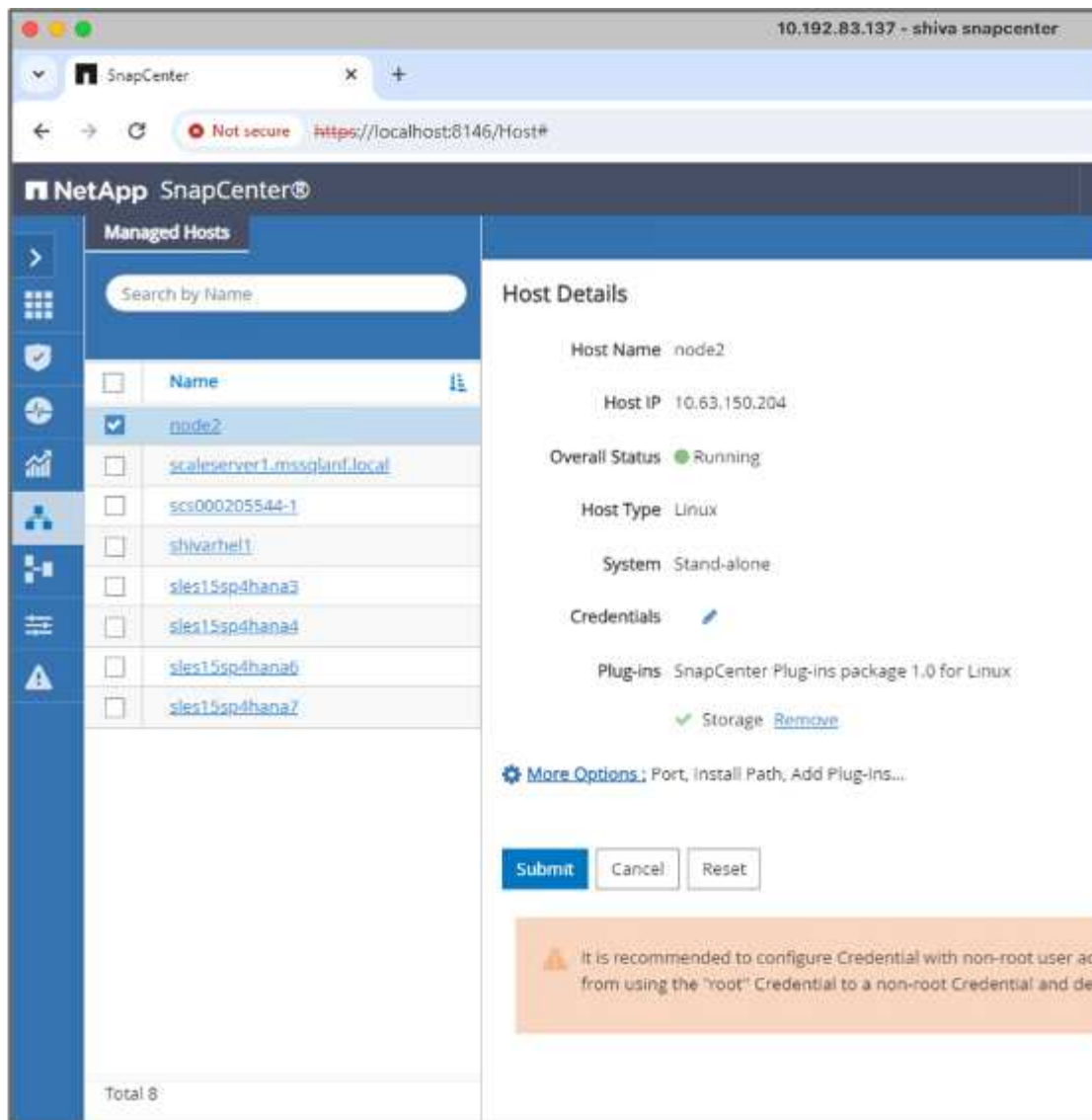
SnapCenterを使用したベクターデータベース保護

NetApp SnapCenterを使用したベクターデータベース保護。

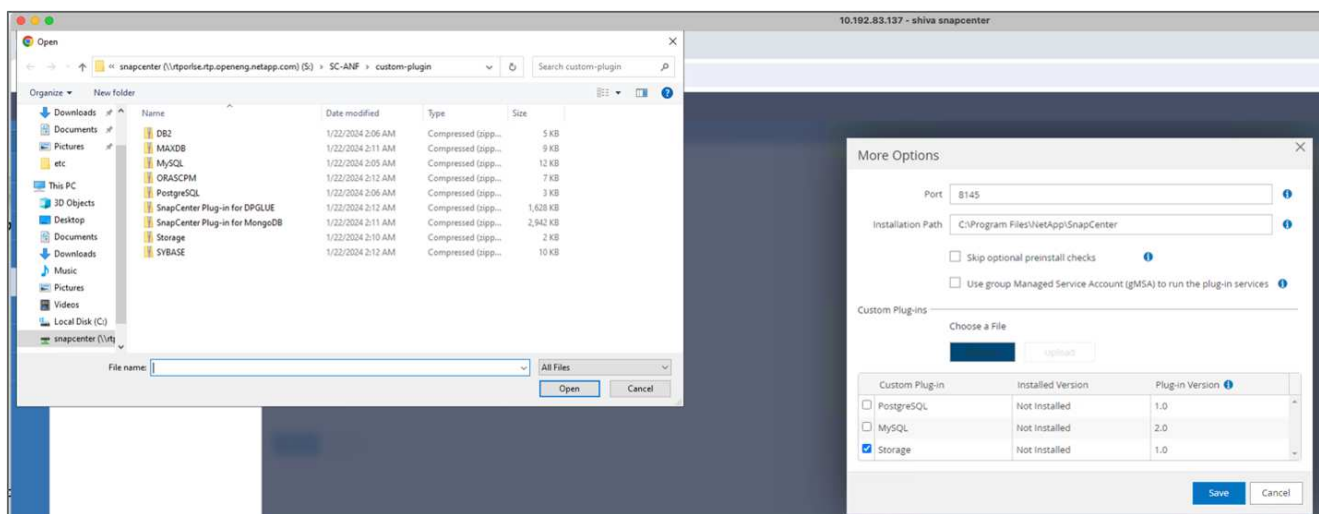
たとえば、映画制作業界では、顧客はビデオファイルやオーディオファイルなどの重要な埋め込みデータを所有していることがよくあります。ハードドライブの故障などの問題が原因でこのデータが失われると、業務に大きな影響を与え、数百万ドル規模のベンチャーが危険にさらされる可能性があります。ネットアップでは、貴重なコンテンツが失われ、多大な混乱と財務上の損失を引き起こしている事例が発生しています。したがって、この業界では、この重要なデータのセキュリティと整合性を確保することが最も重要です。

このセクションでは、SnapCenterがONTAPに存在するベクターデータベースデータとMilvusデータをどのように保護するかについて詳しく説明します。この例では、顧客データ用にNFS ONTAPボリューム（vol1）から派生したNASバケット（milvusdbvol1）を使用し、Milvusクラスタ構成データ用に別のNFSボリューム（vectordbvpv）を使用しました。次を確認してください：["こちらをご覧ください"](#) SnapCenterバックアップワークフロー

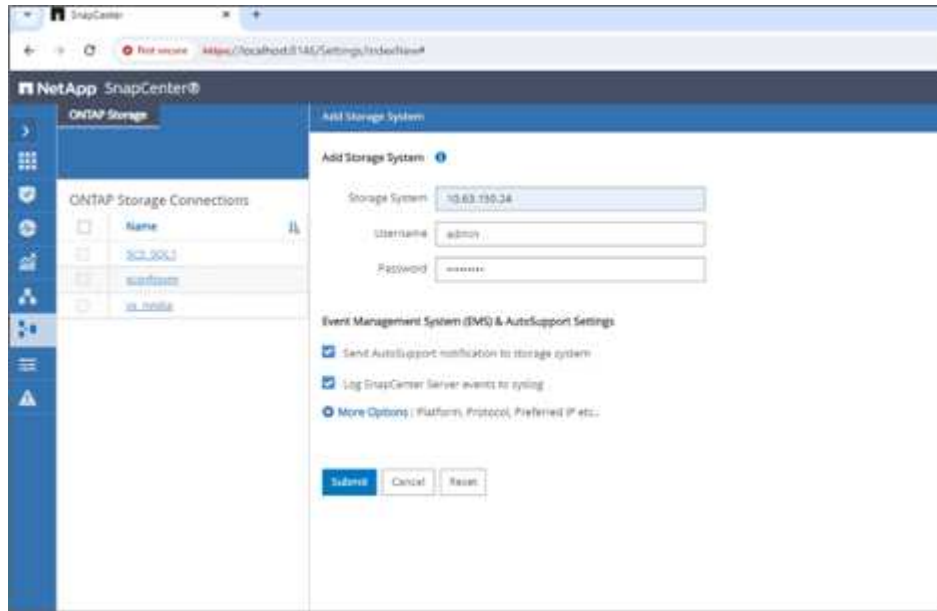
1. SnapCenterコマンドの実行に使用するホストを設定します。



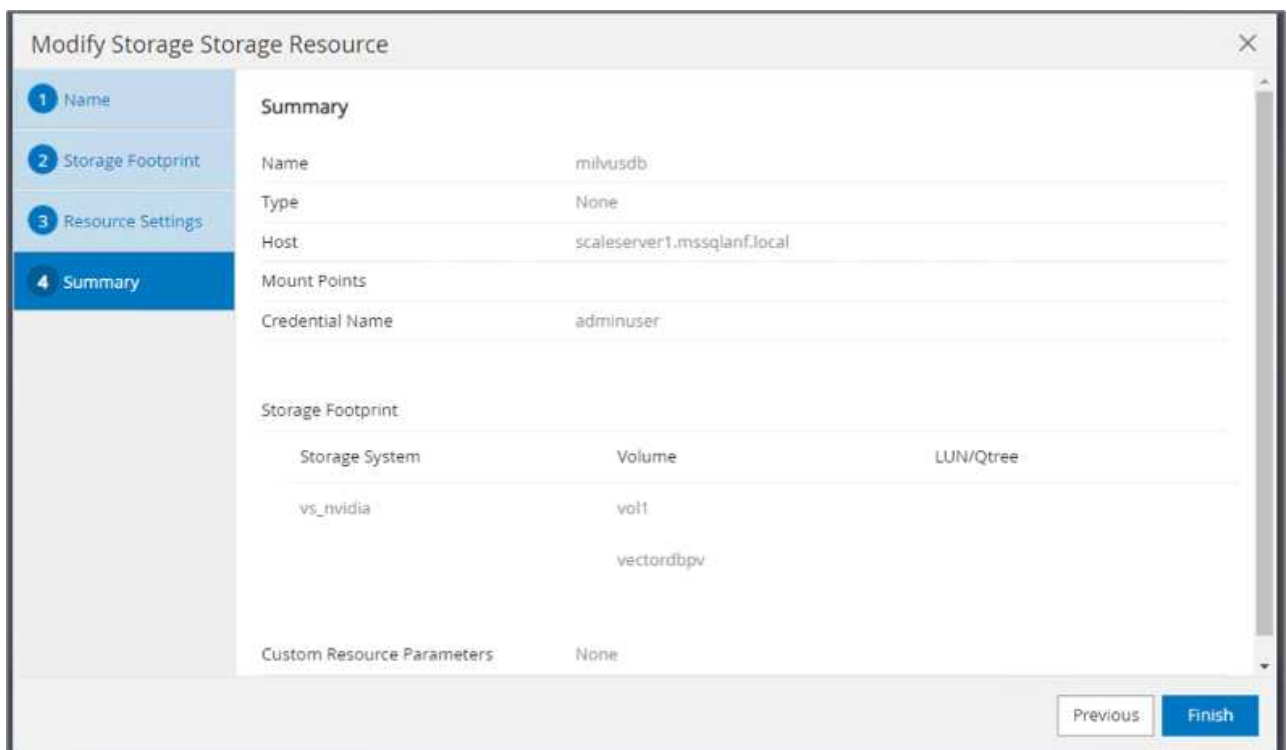
2. ストレージプラグインをインストールして設定します。追加したホストから、[その他のオプション]を選択します。ダウンロードしたストレージプラグインに移動してから選択します。"NetApp Automation Store の略"。プラグインをインストールし、設定を保存します。



3. ストレージシステムとボリュームをセットアップします。[ストレージシステム]にストレージシステムを追加し、SVM (Storage Virtual Machine) を選択します。この例では、「vs_nvidia」を選択しています。



4. バックアップポリシーとカスタムスナップショット名を組み込んで、ベクターデータベースのリソースを確立します。
 - 整合グループバックアップをデフォルト値で有効にし、ファイルシステムの整合性を伴わないSnapCenterを有効にします。
 - [Storage Footprint]セクションで、ベクターデータベースの顧客データとMilvusクラスタデータに関連付けられているボリュームを選択します。この例では、「vol1」と「vectordbvp」です。
 - ベクターデータベース保護のポリシーを作成し、ポリシーを使用してベクターデータベースリソースを保護します。



5. Pythonスクリプトを使用してS3 NASバケットにデータを挿入します。今回のケースでは、Milvusが提供するバックアップスクリプト「prepare_data_netapp.py」を変更し、「sync」コマンドを実行して、オペレーティングシステムからデータをフラッシュしました。

```
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` ===

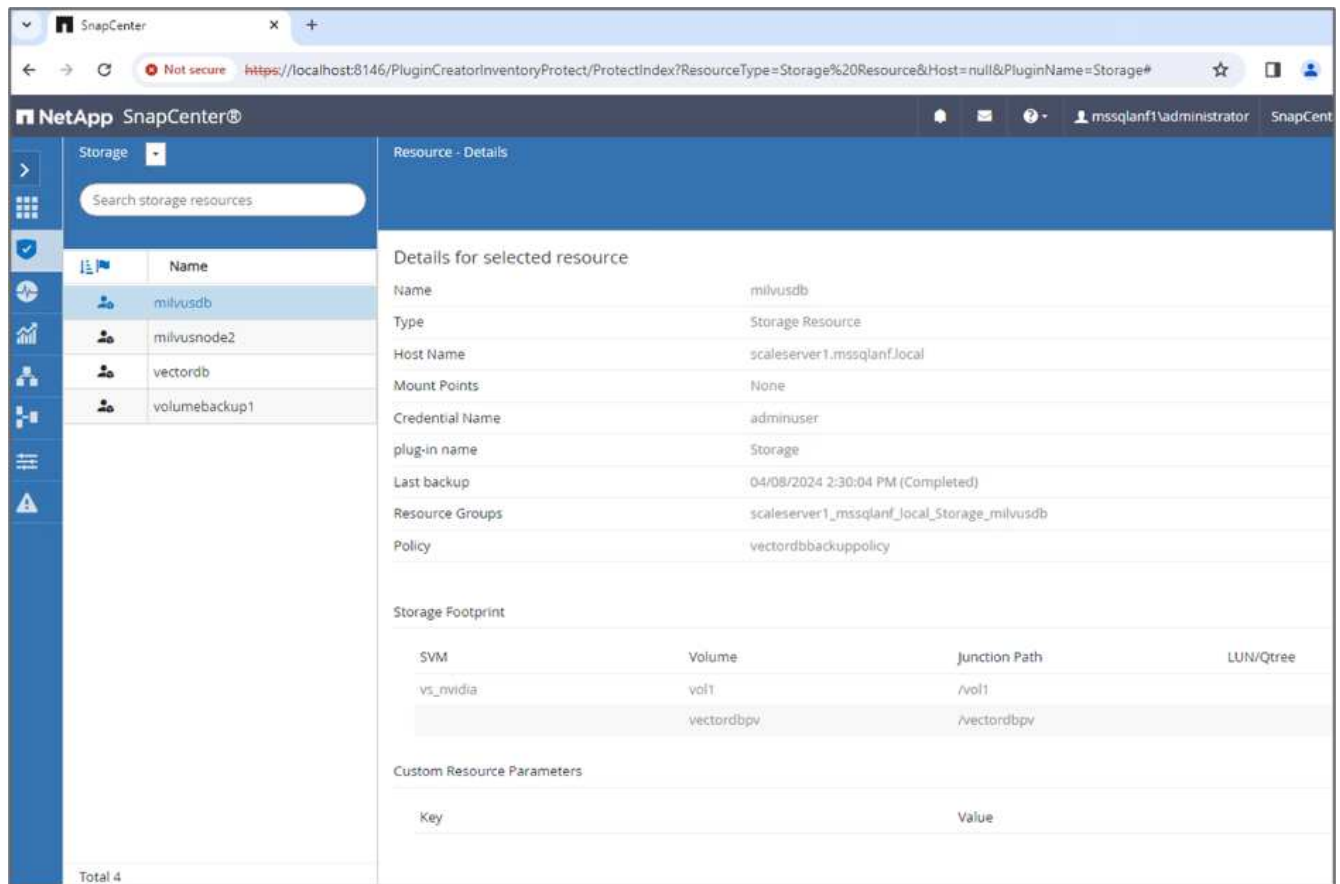
Number of entities in hello_milvus_netapp_sc_test2: 6000
root@node2:~# for i in 2 3 4 5 6 ; do ssh node$i "hostname; sync; echo
'sync executed';" ; done
node2
sync executed
node3
sync executed
node4
sync executed
node5
sync executed
node6
sync executed
root@node2:~#
```

6. S3 NASバケット内のデータを確認します。この例では、タイムスタンプ「2024-04-08 21:22」のファイルが「prepare_data_netapp.py」スクリプトによって作成されています。

```
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」リソースの整合グループ (CG) Snapshotを使用してバックアップを開始する



8. バックアップ機能をテストするために、バックアッププロセス後に新しいテーブルを追加するか、NFS（S3 NASバケット）から一部のデータを削除しました。

このテストでは、バックアップ後に誰かが新しいコレクション、不要なコレクション、または不適切なコレクションを作成したシナリオを想像してください。このような場合は、新しいコレクションが追加される前に、ベクターデータベースをその状態に戻す必要があります。たとえば、「hello_milvus_netapp_sc_testnew」や「hello_milvus_netapp_sc_testnew2」などの新しいコレクションが挿入されています。

```

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       ===

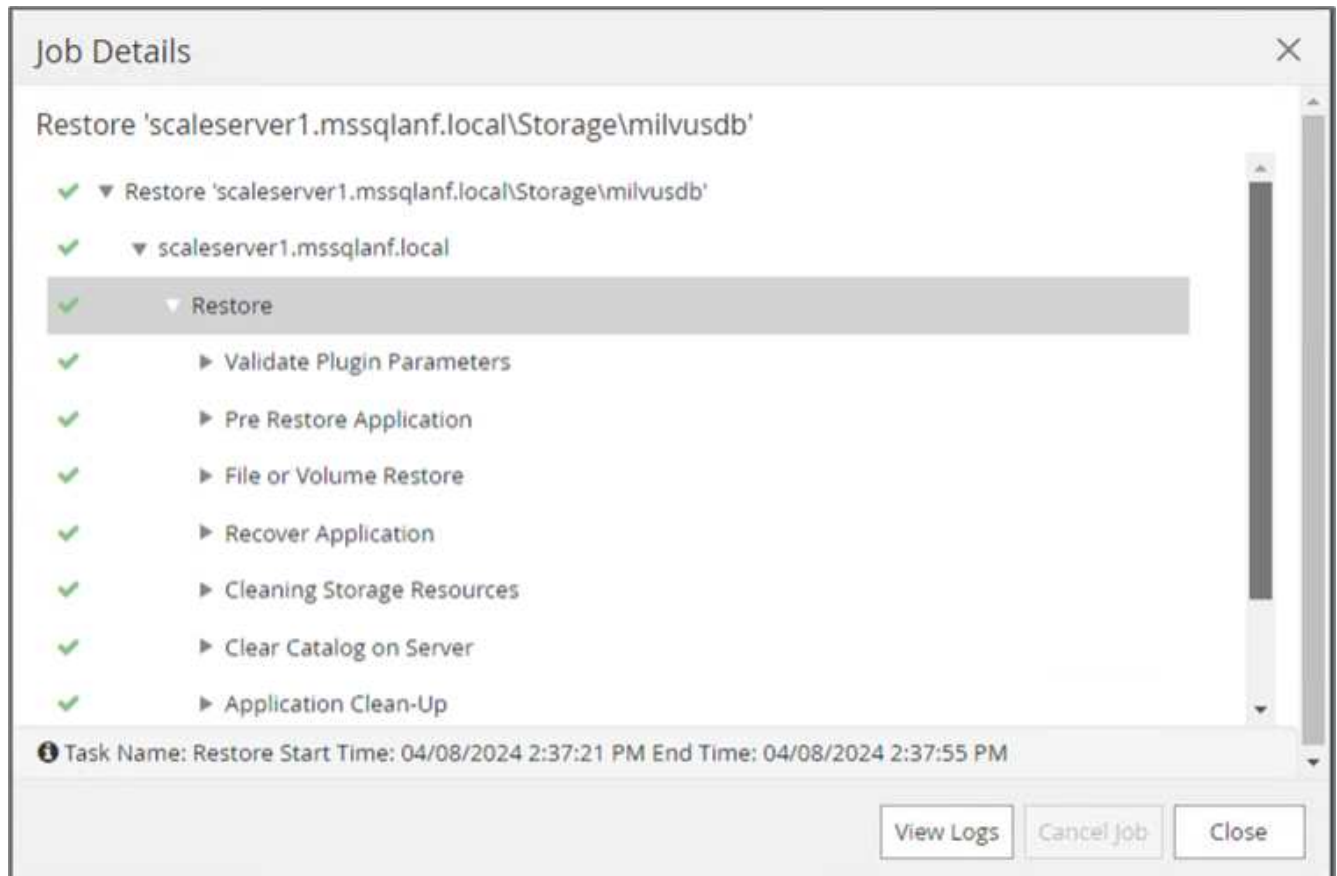
Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

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

```

9. 前のSnapshotからS3 NASバケットのフルリストアを実行します。



10. Pythonスクリプトを使用して、「hello_milvus_netapp_sc_test」コレクションと「hello_milvus_netapp_sc_test2」コレクションのデータを検証します。

```
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. 不要または不適切な収集がデータベースに存在しないことを確認します。

```

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:~#

```

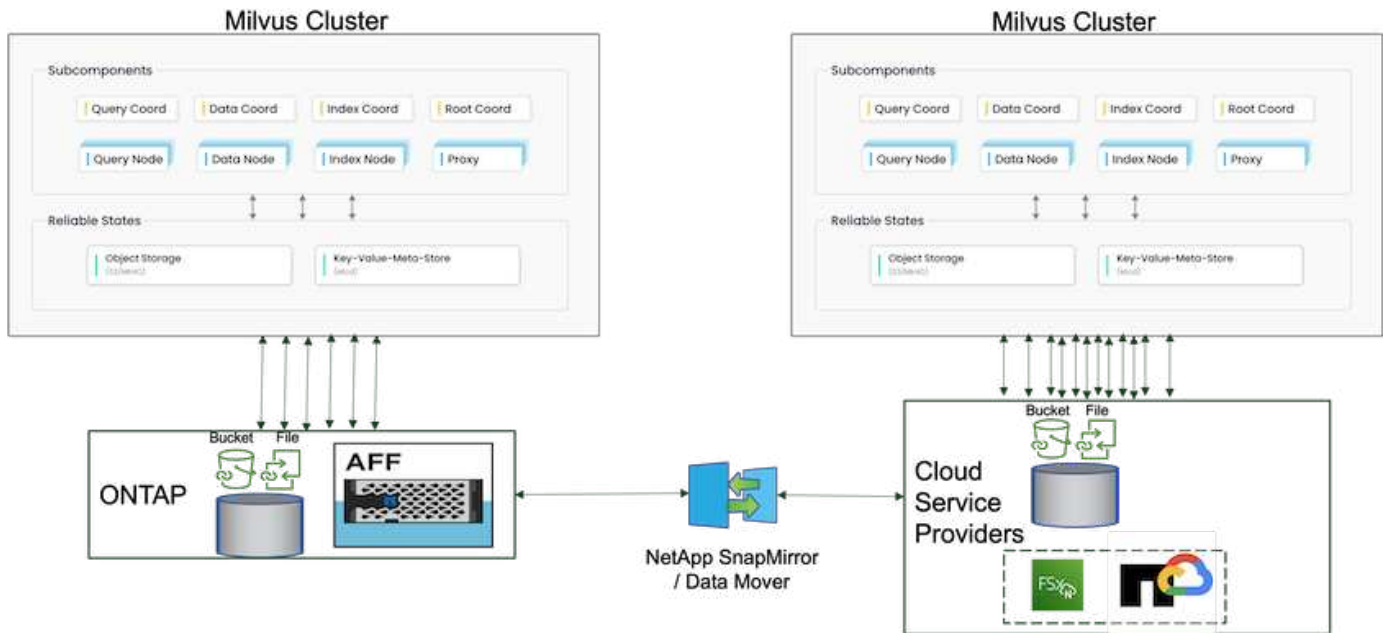
結論として、ネットアップのSnapCenterを使用してベクターデータベースのデータとONTAPにあるMilvusのデータを保護することは、特にデータの整合性が最も重視される業界（映画制作など）で、お客様に大きなメリットをもたらします。SnapCenterでは、整合性のあるバックアップを作成し、完全なデータリストアを実行できるため、組み込みのビデオファイルやオーディオファイルなどの重要なデータを、ハードドライブの障害やその他の問題による損失から確実に保護できます。これにより、業務の中断が防止されるだけでなく、多額の財務上の損失からも保護されます。

このセクションでは、ホストのセットアップ、ストレージプラグインのインストールと構成、カスタムスナップショット名を持つベクターデータベースのリソースの作成など、ONTAPに存在するデータを保護するためにSnapCenterを構成する方法を説明しました。また、整合グループのSnapshotを使用してバックアップを実行し、S3 NASバケット内のデータを検証する方法についても紹介しました。

さらに、バックアップ後に不要または不適切な収集が作成されるシナリオをシミュレートしました。このような場合、SnapCenterで以前のSnapshotからフルリストアを実行すると、新しいコレクションが追加される前の状態にベクターデータベースを戻すことができるため、データベースの整合性が維持されます。特定の時点でデータをリストアするこの機能は、お客様にとって非常に貴重なものであり、データのセキュリティだけでなく、適切に保持されていることを保証します。このように、ネットアップのSnapCenter製品は、データの保護と管理のための堅牢で信頼性の高い解決策をお客様に提供します。

NetApp SnapMirrorを使用したディザスタリカバリ

NetApp SnapMirrorを使用したディザスタリカバリ



ディザスタリカバリは、ベクトルデータベースの整合性と可用性を維持するために非常に重要です。特に、ディザスタリカバリは、高次元データの管理と複雑な類似性検索の実行において重要です。適切に計画され、実装されたディザスタリカバリ戦略により、ハードウェア障害、自然災害、サイバー攻撃などの予期しないインシデントが発生した場合でも、データの損失や侵害が発生しないようにします。これは、データの損失や破損が業務の中断や財務上の損失につながる可能性があるベクターデータベースに依存するアプリケーションで特に重要です。さらに、堅牢なディザスタリカバリ計画により、ダウンタイムを最小限に抑え、サービスの迅速な復旧を可能にすることで、ビジネス継続性も確保されます。これは、地理的に離れた場所にあるNetAppデータ・レプリケーション製品SnapMirror'定期的なバックアップ'フェイルオーバー・メカニズムによって実現されます。したがって、ディザスタリカバリは単なる保護対策ではなく、責任ある効率的なベクターデータベース管理の重要な要素です。

ネットアップのSnapMirrorは、NetApp ONTAPストレージコントローラ間のデータレプリケーションを提供します。主にディザスタリカバリ (DR) ソリューションやハイブリッドソリューションに使用されます。ベクターデータベースのコンテキストでは、オンプレミス環境とクラウド環境の間でスムーズにデータを移行できます。この移行は、データ変換やアプリケーションのリファクタリングを必要としないため、複数のプラットフォームにわたるデータ管理の効率性と柔軟性が向上します。

ベクターデータベースのシナリオでNetApp Hybrid解決策を使用すると、より多くの利点が得られます。

1. 拡張性：ネットアップのハイブリッドクラウド解決策は、要件に応じてリソースを拡張する機能を提供します。ピーク時や予期しない負荷に備えて、通常の予測可能なワークロードや、Amazon FSx for NetApp ONTAPやGoogle Cloud NetApp Volume (GCNV) などのクラウドリソースにオンプレミスリソースを活用できます。
2. コスト効率：ネットアップのハイブリッドクラウドモデルでは、通常のワークロードにオンプレミスのリソースを使用し、必要なときにのみクラウドリソースに料金を支払うことで、コストを最適化できます。この従量課金制モデルは、NetApp instaclustrサービスを使用すると、非常に対費用効果が高くなります。オンプレミスや大手クラウドサービスプロバイダには、instaclustrがサポートとコンサルティングを提供します。
3. 柔軟性：ネットアップのハイブリッドクラウドなら、データの処理場所を柔軟に選択できます。たとえば、より強力なハードウェアがあり、クラウドでの処理の負荷が低い複雑なベクトル操作をオンプレミスで実行することもできます。
4. ビジネス継続性：災害発生時にデータをNetAppハイブリッドクラウドに配置することで、ビジネス継続性を確保できます。オンプレミスのリソースに影響が出た場合は、迅速にクラウドに切り替えることができ

ます。NetApp SnapMirrorを活用して、オンプレミスとクラウドの間でデータを移動できます。

5. イノベーション：ネットアップのハイブリッドクラウドソリューションは、最先端のクラウドサービスとテクノロジーを利用できるようにすることで、イノベーションの加速も可能にします。Amazon FSx for NetApp ONTAP、Azure NetApp Files、Google Cloud NetApp Volumeなど、クラウドにおけるNetAppのイノベーションは、クラウドサービスプロバイダにとって革新的な製品であり、好まれるNASです。

Vectorデータベースのパフォーマンス検証

パフォーマンスの検証

パフォーマンス検証は、ベクターデータベースとストレージシステムの両方で重要な役割を果たし、最適な運用と効率的なリソース使用率を確保するための重要な要素となります。ベクトルデータベースは、高次元データを処理し、類似性検索を実行することで知られており、複雑なクエリを迅速かつ正確に処理するために、高いパフォーマンスレベルを維持する必要があります。パフォーマンス検証は、ボトルネックを特定し、構成を微調整し、サービスを低下させることなく予想される負荷にシステムが対応できることを確認するのに役立ちます。同様に、ストレージシステムにおいても、レイテンシの問題やシステム全体のパフォーマンスに影響を与えるボトルネックが発生することなく、データを効率的に格納、取得するためには、パフォーマンスの検証が不可欠です。また、ストレージインフラのアップグレードや変更に必要な情報に基づいて意思決定を下すのにも役立ちます。したがって、パフォーマンス検証はシステム管理の重要な側面であり、高いサービス品質、運用効率、およびシステム全体の信頼性の維持に大きく貢献します。

このセクションでは、Milvusやpgvecto.RSなどのベクターデータベースのパフォーマンス検証について、LLM ライフサイクル内でのRAGおよび推論ワークロードをサポートする際のI/OプロファイルやNetAppストレージコントローラの動作などのストレージパフォーマンス特性に焦点を当てて詳しく説明します。これらのデータベースをONTAPストレージ解決策と組み合わせた場合のパフォーマンスの差別化要因を評価し、特定します。分析は、1秒あたりに処理されるクエリ数（QPS）などの主要パフォーマンス指標に基づいて行われます。

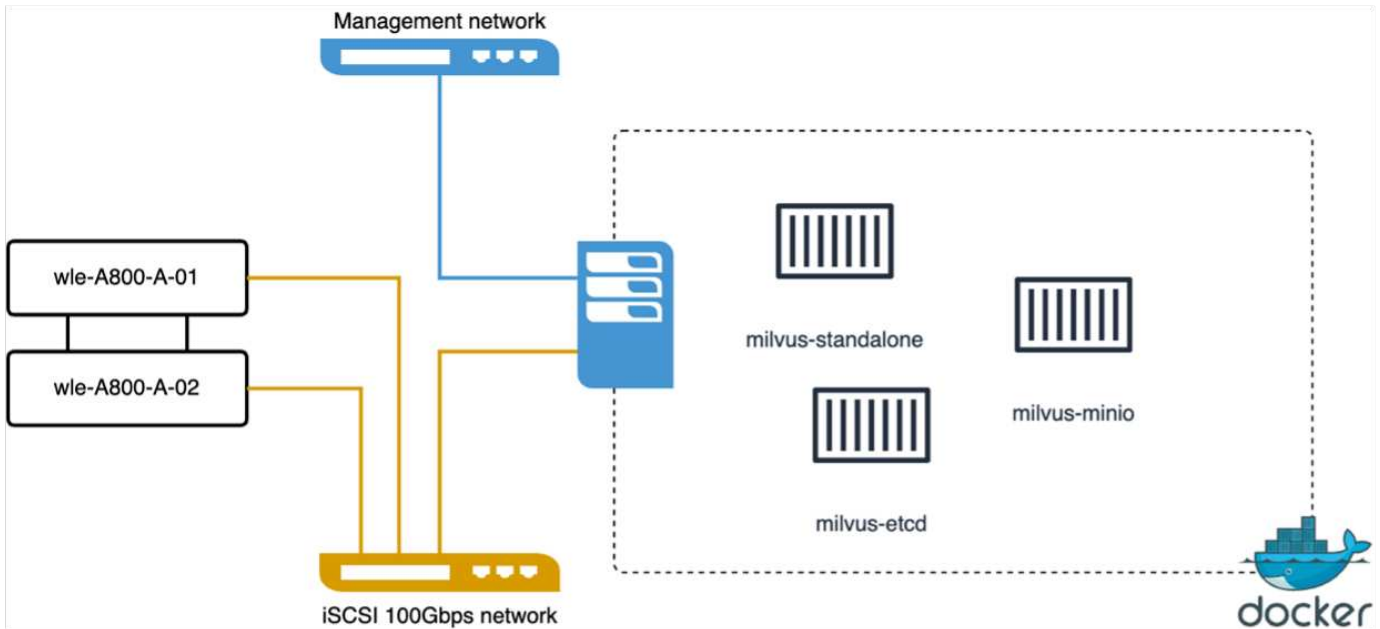
ミルバスに使用されている方法と進捗状況を以下で確認してください。

詳細	Milvus (スタンドアロンおよびクラスタ)	postgres (pgvecto.rs)
バージョン	2.3.2	0.2.0
ファイルシステム	iSCSI LUN上のXFS	
ワークロードジェネレータ	"VectorDB -ベンチ" -v0.0.5	
データセット	LAIONデータセット *1千万件の埋め込み *768寸法 *最大300GBのデータセットサイズ	

VectorDB - Milvusスタンドアロンクラスタ付きベンチ

vectorDB-Benchを使用したmilvusスタンドアロンクラスタについて、次のパフォーマンス検証を行いました。

milvusスタンドアロンクラスタのネットワーク接続とサーバ接続を以下に示します。



このセクションでは、Milvusスタンドアロンデータベースのテスト結果を共有します。

。これらのテストのインデックスタイプとしてDiskANNを選択しました。

。約100GBのデータセットのインデックスの取り込み、最適化、作成には約5時間かかりました。この期間のほとんどで、20コア(ハイパースレッディングが有効な場合は40 vCPUに相当)を搭載したMilvusサーバーは、最大CPU容量100%で動作していました。システムメモリサイズを超える大規模なデータセットでは、DiskANNが特に重要であることがわかりました。

。クエリフェーズでは、0.9987のリコールで10.93のQueries Per Second (QPS)率が観察されました。クエリの99パーセンタイルレイテンシは708.2ミリ秒で測定されました。

ストレージに関して言えば、データベースは取り込み、挿入後の最適化、インデックス作成の各フェーズで1秒あたり約1、000ops/secの処理を実行していました。クエリフェーズでは、1秒あたり32、000件の処理が必要でした。

次のセクションでは、ストレージパフォーマンスの指標について説明します。

ワークロードフェーズ	メートル法	価値
データの取り込み および ポストインサートの最適化	IOPS	1、000未満
	レイテンシ	400 usecs未満
	ワークロード	読み取り/書き込み混在（主に書き込み）
	IOサイズ	64KB
クエリ	IOPS	最大32、000
	レイテンシ	400 usecs未満
	ワークロード	100%キャッシュ読み取り
	IOサイズ	主に8KB

vectorDB-benchの結果は以下のとおりです。

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)

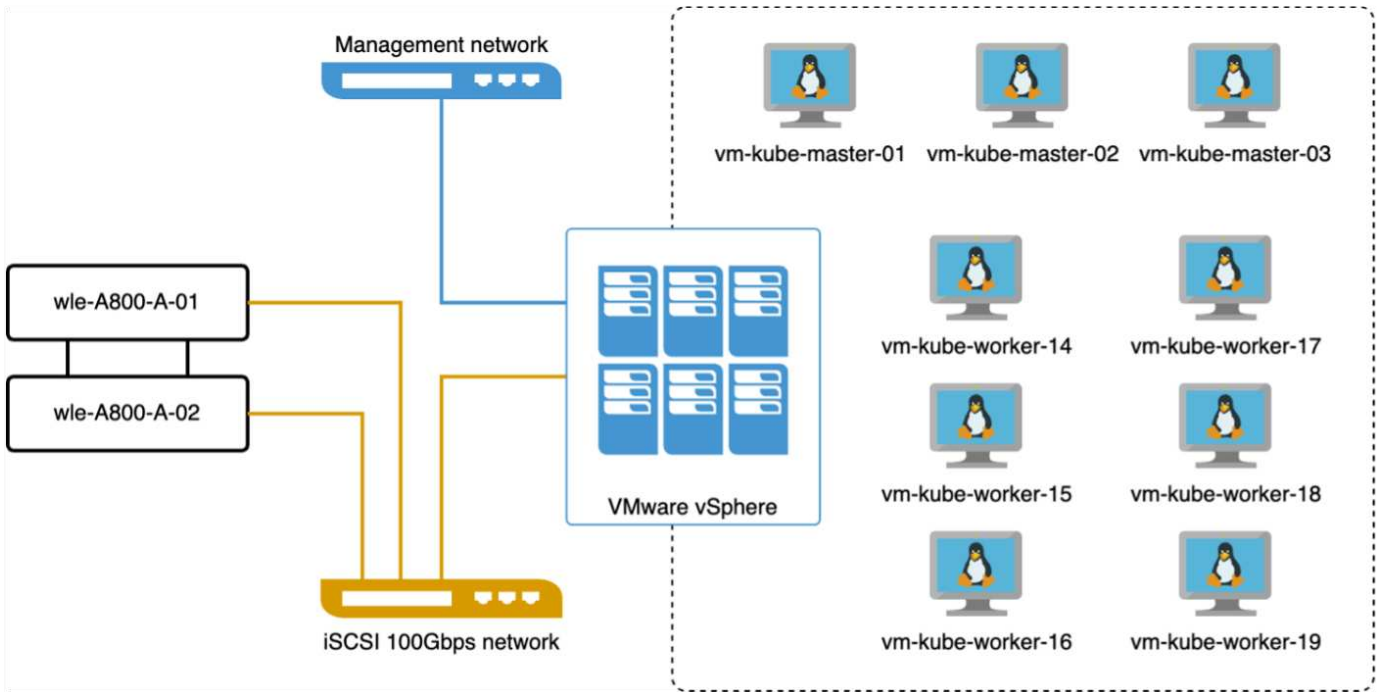
Milvus  708.2ms

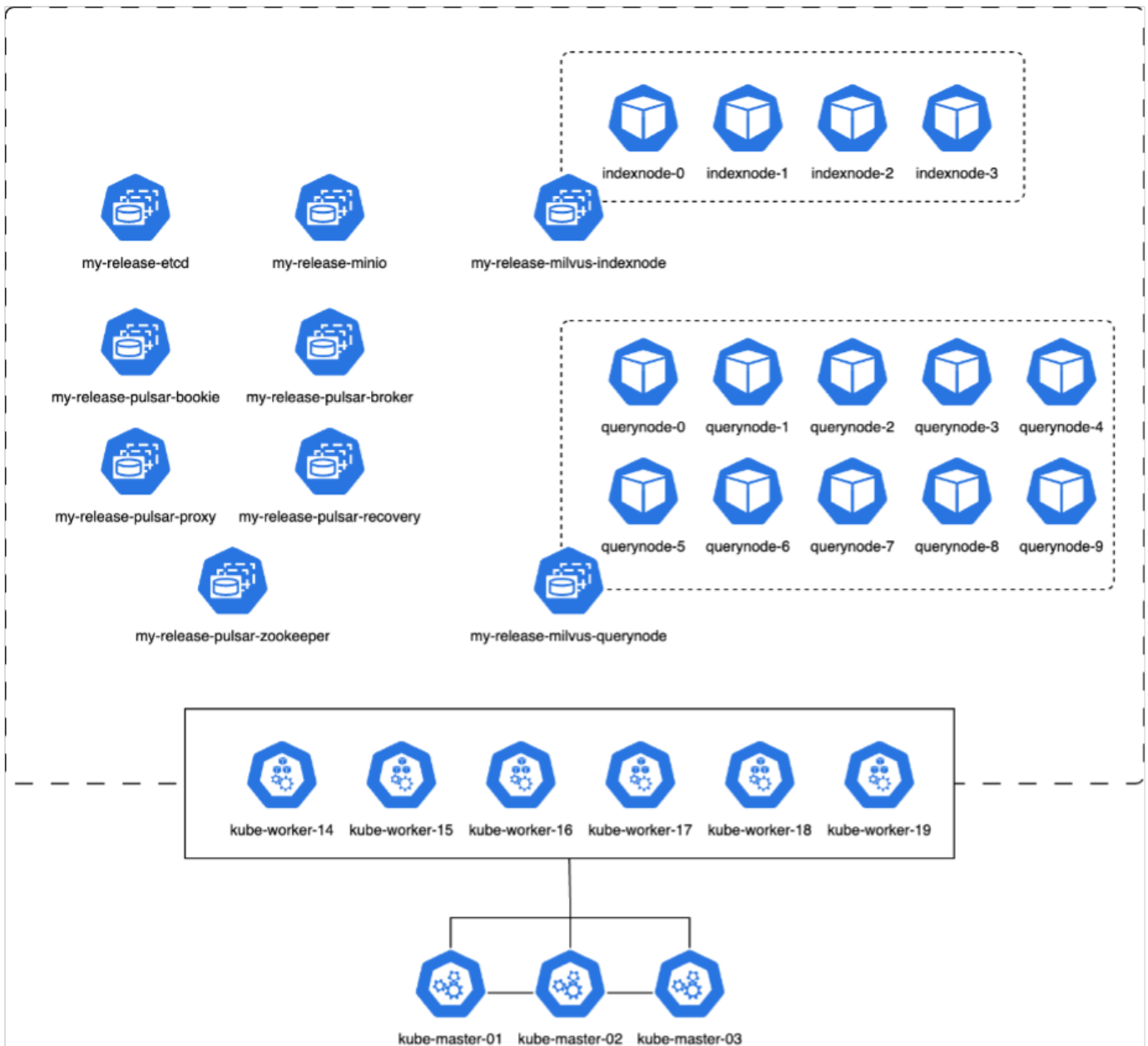
スタンドアロンMilvusインスタンスのパフォーマンス検証から、次元1536の500万ベクトルのデータセットをサポートするには、現在のセットアップでは不十分であることがわかります。ストレージには十分なリソースがあり、システムのボトルネックになっていないことが確認されました。

VectorDB -ミルバスクラスター付きベンチ

このセクションでは、Kubernetes環境内でのMilvusクラスターの導入について説明します。このKubernetesセットアップは、KubernetesのマスターノードとワーカーノードをホストするVMware vSphere環境の上に構築されました。

VMware vSphere環境とKubernetes環境の詳細については、以降のセクションで説明します。





このセクションでは、Milvusデータベースをテストした結果と観測結果を紹介します。

*使用されたインデックスタイプはDiskANNです。

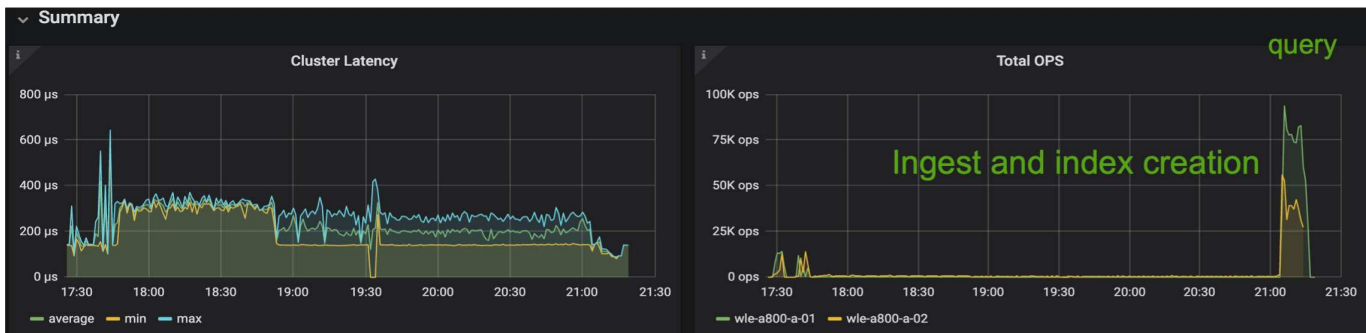
*次の表は、次元1536で500万のベクトルを使用する場合の、スタンドアロン配置とクラスタ配置の比較を示しています。クラスタ環境では、データの取り込みと挿入後の最適化にかかる時間が短くなっていることがわかりました。クラスタ環境では、クエリの99パーセンタイルレイテンシがスタンドアロンセットアップに比べて6分の1に短縮されました。

* Queries Per Second (QPS ; 1秒あたりのクエリ数) の割合はクラスタ環境では高くなりましたが、望ましいレベルではありませんでした。

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%

次の図は、ストレージクラスタのレイテンシや合計IOPS（1秒あたりの入出力処理数）など、さまざまなスト

レージ指標を示しています。



次のセクションでは、ストレージパフォーマンスの主要な指標について説明します。

ワークロードフェーズ	メートル法	価値
データの取り込み および ポストインサートの最適化	IOPS	1、000未満
	レイテンシ	400 usecs未満
	ワークロード	読み取り/書き込み混在（主に書き込み）
	IOサイズ	64KB
クエリ	IOPS	ピーク時147、000
	レイテンシ	400 usecs未満
	ワークロード	100%キャッシュ読み取り
	IOサイズ	主に8KB

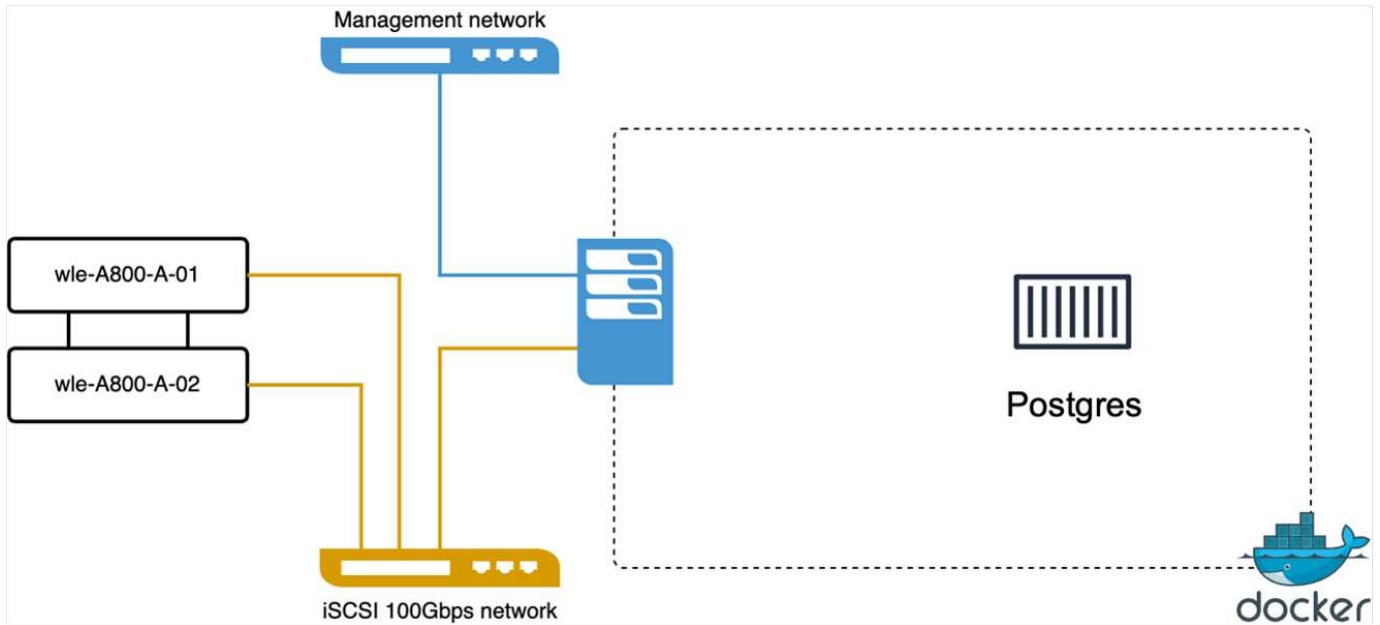
スタンドアロンのMilvusクラスタとMilvusクラスタの両方のパフォーマンス検証に基づいて、ストレージI/Oプロファイルの詳細を提示します。

*スタンドアロン環境とクラスタ環境の両方で、I/Oプロファイルが一貫していることが確認されました。

*ピークIOPSの差は、クラスタ環境内のクライアント数が多いことが原因である可能性があります。

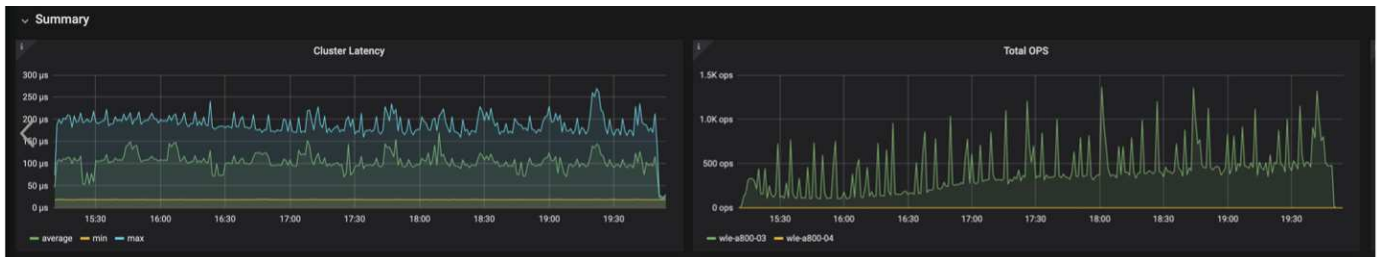
vectorDB - Postgresを使用したベンチ(pgvector.rs)

VectorDB-Benchを使用して、PostgreSQL (pgvector.rs) に対して次のアクションを実行しました。PostgreSQL (特にpgvector.rs) のネットワーク接続とサーバ接続に関する詳細は次のとおりです。



このセクションでは、pgvecto.rsを使用してPostgreSQLデータベースをテストした結果と結果を共有します。
 *テストのインデックスタイプとしてHNSWを選択したのは、テスト時にpgvecto.rsでDiskANNを使用できなかったためです。
 *データ取り込みフェーズでは、次元768の1000万ベクトルからなるCohereデータセットをロードしました。このプロセスには約4.5時間かかりました。
 *クエリフェーズでは、1秒あたりのクエリ数（QPS）は1,068、リコールは0.6344でした。クエリの99パーセントレイテンシは20ミリ秒で測定されました。ランタイムのほとんどで、クライアントCPUは100%の容量で動作していました。

次の図は、ストレージクラスタの合計IOPS（1秒あたりの入出力処理数）など、さまざまなストレージ指標を示しています。



The following section presents the key storage performance metrics.
 image:pgvecto_storage_perf_metrics.png["エラー：グラフィックイメージがありません"]
]

VECTOR DBベンチでのmilvusとpostgresのパフォーマンス比較

Vector Database Benchmark

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

Search Performance Test (10M Dataset, 768 Dim)

Qps (more is better)



Recall (more is better)



Serial_latency_p99 (less is better)



VectorDBBenchを使用したMilvusおよびPostgreSQLのパフォーマンス検証に基づいて、次のことを確認しました。

- インデックスタイプ：HNSW
- データセット：768次元で1000万ベクトルのコア

pgvector.rsは0.6344のリコールで1,068のQPSレートを達成し、Milvusは0.9842のリコールで106のQPSレートを達成しました。

クエリの精度が優先される場合、Milvusはpgvector.rsよりもパフォーマンスが高く、クエリごとに関連する項目の割合が高くなります。ただし、1秒あたりのクエリ数がより重要な要素である場合、pgvector.rsはMilvusを超えます。ただし、pgvector.rsを介して取得されるデータの品質は低く、検索結果の約37%が無関係な項目であることに注意する必要があります。

パフォーマンス検証に基づく観察：

パフォーマンスの検証に基づいて、次のことを確認しました。

MilvusのI/Oプロファイルは、OracleのSLOBなどのOLTPワークロードによく似ています。ベンチマークは、データの取り込み、最適化後、クエリの3つのフェーズで構成されています。初期段階は主に64KBの書き込み処理で特徴付けられますが、クエリフェーズでは8KBの読み取りが主に行われます。ONTAPはMilvusのI/O負荷を適切に処理することを期待しています。

PostgreSQLのI/Oプロファイルでは、困難なストレージワークロードは発生しません。メモリ内の実装が現在進行中であるため、クエリフェーズ中にディスクI/Oを確認することはできませんでした。

DiskANNは、ストレージを差別化するための重要なテクノロジーとして登場しています。これにより、システムメモリ境界を越えたベクターDB検索の効率的なスケーリングが可能になります。ただし、HNSWなどのインメモリベクトルDBインデックスを使用して、ストレージパフォーマンスの差別化を確立することはほとんどありません。

また、インデックスタイプがHNSWの場合、クエリフェーズでストレージが重要な役割を果たさないことも注目に値します。HNSWは、RAGアプリケーションをサポートするベクターデータベースで最も重要な操作フェーズです。つまり、ストレージのパフォーマンスがこれらのアプリケーションの全体的なパフォーマンスに大きく影響することはありません。

著作権に関する情報

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および/または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用权を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用权については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。