Übersicht Über Die Lösungsüberprüfung

NetApp Solutions

NetApp
May 03, 2024

Inhalt

Lösungsüberblick

Wir haben eine umfassende Lösungsvalidierung durchgeführt, die sich auf fünf Kernbereiche konzentriert. Die Details werden im Folgenden erläutert. Jeder Abschnitt geht auf die Herausforderungen von Kunden, die von NetApp bereitgestellten Lösungen und die daraus folgenden Vorteile für den Kunden ein.

1. "Milvus Cluster-Einrichtung mit Kubernetes vor Ort"
   Für Kunden stellt es eine Herausforderung dar, nach Bedarf unabhängig nach Storage- und Computing-Ressourcen, effektivem Infrastrukturmanagement und Datenmanagement zu skalieren. In diesem Abschnitt wird der Prozess der Installation eines Milvus Clusters auf Kubernetes beschrieben, der einen NetApp Storage Controller für Cluster- und Kundendaten verwendet.
2. "Milvus mit Amazon FSxN für NetApp ONTAP – Datei- und Objektdualität"
   In diesem Abschnitt, Warum wir brauchen, um Vektor-Datenbank in der Cloud sowie Schritte zur Bereitstellung von Vektor-Datenbank (milvus Standalone) in Amazon FSxN für NetApp ONTAP in Docker-Containern bereitzustellen.
3. "Sicherung von Vector Datenbanken mit NetApp SnapCenter."
   In diesem Abschnitt befassen wir uns eingehend damit, wie SnapCenter die Vektordatenbank-Daten und Milvus-Daten in ONTAP schützt. Für dieses Beispiel wurde ein NAS-Bucket (milvusdbvol1) verwendet, der von einem NFS-ONTAP-Volume (vol1) für Kundendaten abgeleitet wurde, und ein separates NFS-Volume (vectordbpv) für Milvus-Cluster-Konfigurationsdaten.
4. "Disaster Recovery mit NetApp SnapMirror"
   In diesem Abschnitt wird die Bedeutung von Disaster Recovery (DR) für Vektordatenbanken erläutert und es wird erläutert, wie das Disaster Recovery-Produkt von NetApp snapmirror die DR-Lösung für die Vektordatenbank bereitstellt.
5. "Performance-Validierung"
   In diesem Abschnitt möchten wir uns mit der Performance-Validierung von Vektordatenbanken wie Milvus und pgvecto.rs beschäftigen. Dabei konzentrieren wir uns auf die Speicherleistungsmerkmale wie I/O-Profil und das Verhalten des NetApp-Speichercontrollers zur Unterstützung von RAG- und Inferenzarbeitslasten innerhalb des LLM-Lebenszyklus. Wir bewerten und ermitteln jegliche Performance-Unterscheidungsmerkmale, wenn diese Datenbanken mit der ONTAP Storage-Lösung kombiniert werden. Unsere Analyse basiert auf wichtigen Leistungsindikatoren, wie der Anzahl der pro Sekunde verarbeiteten Abfragen (QPS).

Milvus Cluster Setup mit Kubernetes vor Ort

Milvus Cluster-Einrichtung mit Kubernetes vor Ort

Kundenherausforderungen bei der unabhängigen Skalierung nach Storage- und Computing-Ressourcen, effektivem Infrastrukturmanagement und Datenmanagement,
Kubernetes- und Vektordatenbanken bilden zusammen eine leistungsstarke, skalierbare Lösung für das Management großer Datenvorgänge. Kubernetes optimiert Ressourcen und managt Container. Vektordatenbanken können hochdimensionale Daten und Ähnlichkeitssuchen effizient verarbeiten. Diese Kombination ermöglicht die schnelle Verarbeitung komplexer Abfragen auf großen Datensätzen und lässt sich nahtlos an wachsende Datenvolumen skalieren. Dadurch eignet sie sich ideal für Big-Data-Applikationen und KI-Workloads.

1. In diesem Abschnitt wird der Prozess der Installation eines Milvus Clusters auf Kubernetes beschrieben, der einen NetApp Storage Controller für Cluster- und Kundendaten verwendet.
2. Für die Installation eines Milvus-Clusters sind Persistent Volumes (PVs) erforderlich, um Daten aus verschiedenen Milvus-Clusterkomponenten zu speichern. Zu diesen Komponenten gehören etcd (drei

Instanzen), Pulsar-bookie-Journal (drei Instanzen), Pulsar-bookie-Ledgers (drei Instanzen) und Pulsar-Zookeeper-Daten (drei Instanzen).

> Im milvus-Cluster können wir entweder Pulsar oder kafka für die zugrunde liegende Engine verwenden, die die zuverlässige Speicherung und Veröffentlichung/Abonnement von Nachrichtenströmen des Milvus-Clusters unterstützt. Im Falle von Kafka mit NFS hat NetApp Verbesserungen in ONTAP 9.12.1 und höher vorgenommen. Diese Verbesserungen sowie Änderungen an NFSv4.1 und Linux, die in RHEL 8.7 oder 9.1 und höher enthalten sind, lösen das Problem des „dummen Umbenennungen“, das beim Ausführen von Kafka über NFS auftreten kann. Wenn Sie ausführlichere Informationen zum Thema kafka mit NetApp-NFS laufen wünschen, lesen Sie bitte - https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html.

3. Wir haben aus NetApp ONTAP ein einzelnes NFS Volume erstellt und 12 persistente Volumes eingerichtet, jedes mit 250 GB Storage. Die Storage-Größe kann je nach Cluster-Größe variieren; wir haben z. B. einen weiteren Cluster, in dem jeder PV 50 GB hat. Bitte beachten Sie unten eine der PV YAML-Dateien für weitere Details; wir hatten insgesamt 12 solche Dateien. In jeder Datei wird der storageClassName auf 'default' gesetzt, und der Speicher und der Pfad sind für jedes PV eindeutig.

```
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. Führen Sie den Befehl 'kubectl apply' für jede PV YAML-Datei aus, um die Persistent Volumes zu erstellen, und überprüfen Sie dann ihre Erstellung mit ‘kubectl get pv’

```
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. Zum Speichern von Kundendaten unterstützt Milvus Objekt-Storage-Lösungen wie Minio, Azure Blob und S3. In diesem Leitfaden verwenden wir S3. Die folgenden Schritte gelten sowohl für ONTAP S3 als auch für StorageGRID Objektspeicher. Wir verwenden Helm zur Bereitstellung des Milvus Clusters. Laden Sie die Konfigurationsdatei Values.yaml vom Milvus Download-Speicherort herunter. Die Datei Values.yaml, die wir in diesem Dokument verwendet haben, finden Sie im Anhang.
6. Stellen Sie sicher, dass die 'StorageClass' in jedem Abschnitt auf 'Standard' gesetzt ist, einschließlich derjenigen für das Protokoll, etc., Zookeeper und Buchhalter.
7. Deaktivieren Sie Minio im Abschnitt Minio.
8. Erstellen Sie einen NAS-Bucket aus ONTAP oder StorageGRID-Objekt-Storage und fügen Sie sie mit den Zugangsdaten für den Objekt-Storage in ein externes S3-System ein.

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

9. Stellen Sie vor dem Erstellen des Milvus-Clusters sicher, dass das PersistentVolumeClaim (PVC) keine bereits vorhandenen Ressourcen hat.

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

10. Verwenden Sie Helm und die Konfigurationsdatei Values.yaml, um den Milvus-Cluster zu installieren und zu starten.

```
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. Überprüfen Sie den Status der PersistentVolumeClaims (VES).

```
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. Überprüfen Sie den Status der Pods.

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

Stellen Sie sicher, dass der PODs-Status ‘Running’ lautet und wie erwartet funktioniert

13. Testen Sie das Schreiben und Lesen von Daten in Milvus und NetApp Objekt-Storage.
    - Schreiben Sie Daten mit dem Python-Programm „Prepare_Data_netapp_New.py“.

```
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:~#
```

- Lesen Sie die Daten mit der Python-Datei „verify_Data_netapp.py“.

```
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}}]}
```

Basierend auf der oben genannten Validierung bietet die Integration von Kubernetes in eine Vektordatenbank, wie die Implementierung eines Milvus Clusters auf Kubernetes über einen NetApp Storage-Controller demonstriert, Kunden eine robuste, skalierbare und effiziente Lösung für das Management großer Datenoperationen. Diese Einrichtung ermöglicht es Kunden, hochdimensionale Daten schnell und effizient zu verarbeiten und komplexe Abfragen auszuführen. Dadurch ist sie die ideale Lösung für Big-Data-Applikationen und KI-Workloads. Der Einsatz von persistenten Volumes (PVS) für verschiedene Cluster-Komponenten stellt zusammen mit der Erstellung eines einzigen NFS-Volumes aus NetApp ONTAP eine optimale Ressourcenauslastung und ein optimales

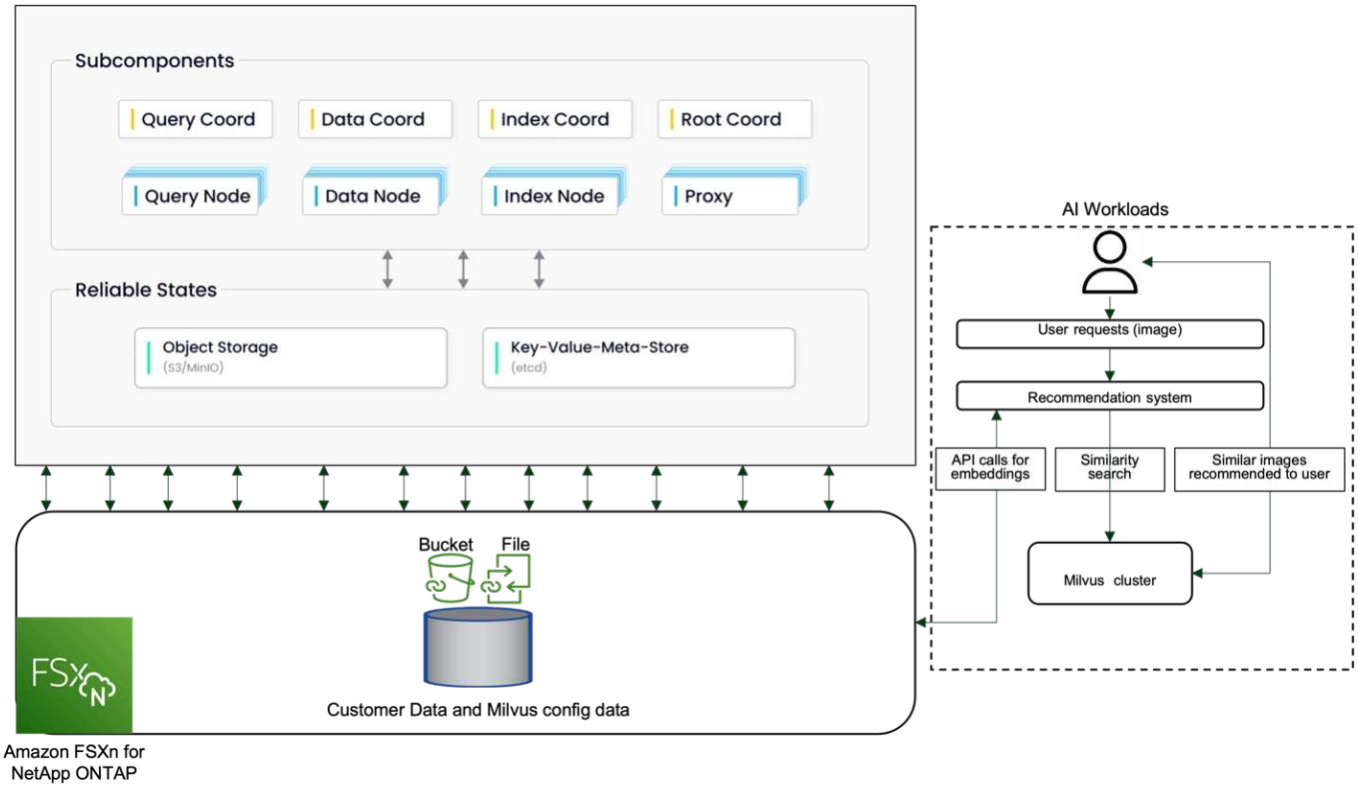
Datenmanagement sicher. Der Prozess der Überprüfung des Status von PersistentVolumeClaims (PVCs) und Pods sowie der Prüfung von Daten Schreiben und Lesen bietet Kunden die Sicherheit zuverlässiger und konsistenter Datenoperationen. Die Nutzung von ONTAP oder StorageGRID Objekt-Storage für Kundendaten verbessert die Verfügbarkeit und Sicherheit von Daten noch weiter. Kunden erhalten durch diese Einrichtung eine robuste und hochperformante Datenmanagement-Lösung, die sich nahtlos an ihre steigenden Datenanforderungen anpassen lässt.

Milvus mit Amazon FSxN für NetApp ONTAP - Datei und Objekt Dualität

Milvus mit Amazon FSxN für NetApp ONTAP – Datei- und Objektdualität

In diesem Abschnitt, Warum wir Vektordatenbank in der Cloud bereitstellen müssen, sowie Schritte zur Bereitstellung von Vektordatenbank (milvus Standalone) in Amazon FSxN für NetApp ONTAP in Docker Containern.

Die Bereitstellung einer Vektordatenbank in der Cloud bietet einige signifikante Vorteile, insbesondere für Anwendungen, die die Verarbeitung von hochdimensionalen Daten und die Ausführung von Ähnlichkeitssuchen erfordern. Erstens bietet die Cloud-basierte Implementierung Skalierbarkeit, die eine einfache Anpassung der Ressourcen an die wachsenden Datenmengen und die Abfragelasten ermöglicht. Auf diese Weise wird sichergestellt, dass die Datenbank den gestiegenen Bedarf effizient bewältigen kann und gleichzeitig eine hohe Performance erhält. Zweitens bietet Cloud-Bereitstellung Hochverfügbarkeit und Disaster Recovery, da Daten an verschiedenen geografischen Standorten repliziert werden können, um das Risiko von Datenverlusten zu minimieren und einen kontinuierlichen Service auch bei unerwarteten Ereignissen sicherzustellen. Drittens bietet sie Kosteneffizienz, da Sie nur für die tatsächlich genutzten Ressourcen zahlen und je nach Bedarf vertikal skalieren können. Dadurch vermeiden Sie erhebliche Vorabinvestitionen in Hardware. Schließlich kann die Bereitstellung einer Vektordatenbank in der Cloud die Zusammenarbeit verbessern, da Daten von überall aus abgerufen und freigegeben werden können. Dies erleichtert Team-basiertes Arbeiten und datengestützte Entscheidungsfindung.
Bitte prüfen Sie die Architektur des eigenständigen milvus mit Amazon FSxN für NetApp ONTAP, die bei dieser Validierung verwendet wird.



1. Erstellen Sie eine Amazon FSxN for NetApp ONTAP-Instanz und notieren Sie sich die Details zu VPC, VPC-Sicherheitsgruppen und dem Subnetz. Diese Informationen sind erforderlich, wenn eine EC2-Instanz erstellt wird. Weitere Details finden Sie hier - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. Erstellen Sie eine EC2-Instanz, um sicherzustellen, dass VPC, Sicherheitsgruppen und Subnetz mit denen der Amazon FSxN für NetApp ONTAP-Instanz übereinstimmen.
3. Installieren Sie nfs-common mit dem Befehl 'apt-get install nfs-common' und aktualisieren Sie die Paketinformationen mit 'udo apt-get Update'.
4. Erstellen Sie einen Mount-Ordner, und mounten Sie den Amazon FSxN für NetApp ONTAP darauf.

```
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. Installieren Sie Docker und Docker Compose mit apt-get install.
6. Richten Sie einen Milvus-Cluster basierend auf der Datei Docker-compose.yaml ein, die von der Milvus-Website heruntergeladen werden kann.

```
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. Ordnen Sie im Abschnitt „Volumes“ der Datei Docker-compose.yml den NetApp-NFS-Bereitstellungspunkt dem entsprechenden Milvus-Containerpfad zu, insbesondere in etcd, minio und Standalone.Check "Anhang D: docker-compose.yml" Für Details über Änderungen in yml
8. Überprüfen Sie die gemounteten Ordner und Dateien.

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

9. Führen Sie 'docker-compose up -d' aus dem Verzeichnis aus, das die Datei docker-compose.yml enthält.
10. Überprüfen Sie den Status des Milvus Containers.

```
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. Um die Lese- und Schreibfunktionalität der Vektordatenbank und ihrer Daten in Amazon FSxN for NetApp ONTAP zu validieren, haben wir das Python Milvus SDK und ein Beispielprogramm von PyMilvus verwendet. Installieren Sie die benötigten Pakete mit 'apt-get install python3-numpy python3-Pip' und installieren Sie PyMilvus mit 'pip3 install pymilvus'.
12. Validieren der Schreib- und Lesevorgänge von Amazon FSxN für NetApp ONTAP in der Vektordatenbank

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

13. Überprüfen Sie den Lesevorgang mit dem Skript verify_data_netapp.py.

```
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. Wenn der Kunde auf NFS-Daten zugreifen möchte, die in der Vektordatenbank über das S3-Protokoll für KI-Workloads getestet wurden (lesen), kann dies mit einem einfachen Python-Programm validiert werden. Ein Beispiel hierfür könnte eine Ähnlichkeitssuche von Bildern aus einer anderen Anwendung sein, wie im Bild zu Beginn dieses Abschnitts erwähnt.

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

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

In diesem Abschnitt wird effektiv gezeigt, wie Kunden mithilfe von Amazon NetApp FSxN für NetApp ONTAP Storage eine eigenständige Einrichtung von Milvus in Docker Containern implementieren und betreiben können. Mit dieser Einrichtung können Kunden die Leistungsfähigkeit von Vektordatenbanken für die Verarbeitung von hochdimensionalen Daten und die Ausführung komplexer Abfragen nutzen – all dies in der skalierbaren und effizienten Umgebung von Docker Containern. Durch Erstellung einer Amazon FSxN for NetApp ONTAP Instanz und Anpassung der EC2-Instanz können Kunden für eine optimale Ressourcenauslastung und Datenmanagement sorgen. Die erfolgreiche Validierung von Datenschreibvorgängen und -Lesevorgängen von FSxN in der Vektordatenbank bietet Kunden die Sicherheit zuverlässiger und konsistenter Datenoperationen. Darüber hinaus bietet die Möglichkeit, Daten aus KI-Workloads über das S3-Protokoll aufzulisten (zu lesen), eine verbesserte Datenverfügbarkeit. Durch diesen umfassenden Prozess erhalten Kunden eine robuste und effiziente Lösung für das Management ihrer umfangreichen Datenabläufe und dabei Nutzung der Funktionen von Amazon FSxN für NetApp ONTAP.

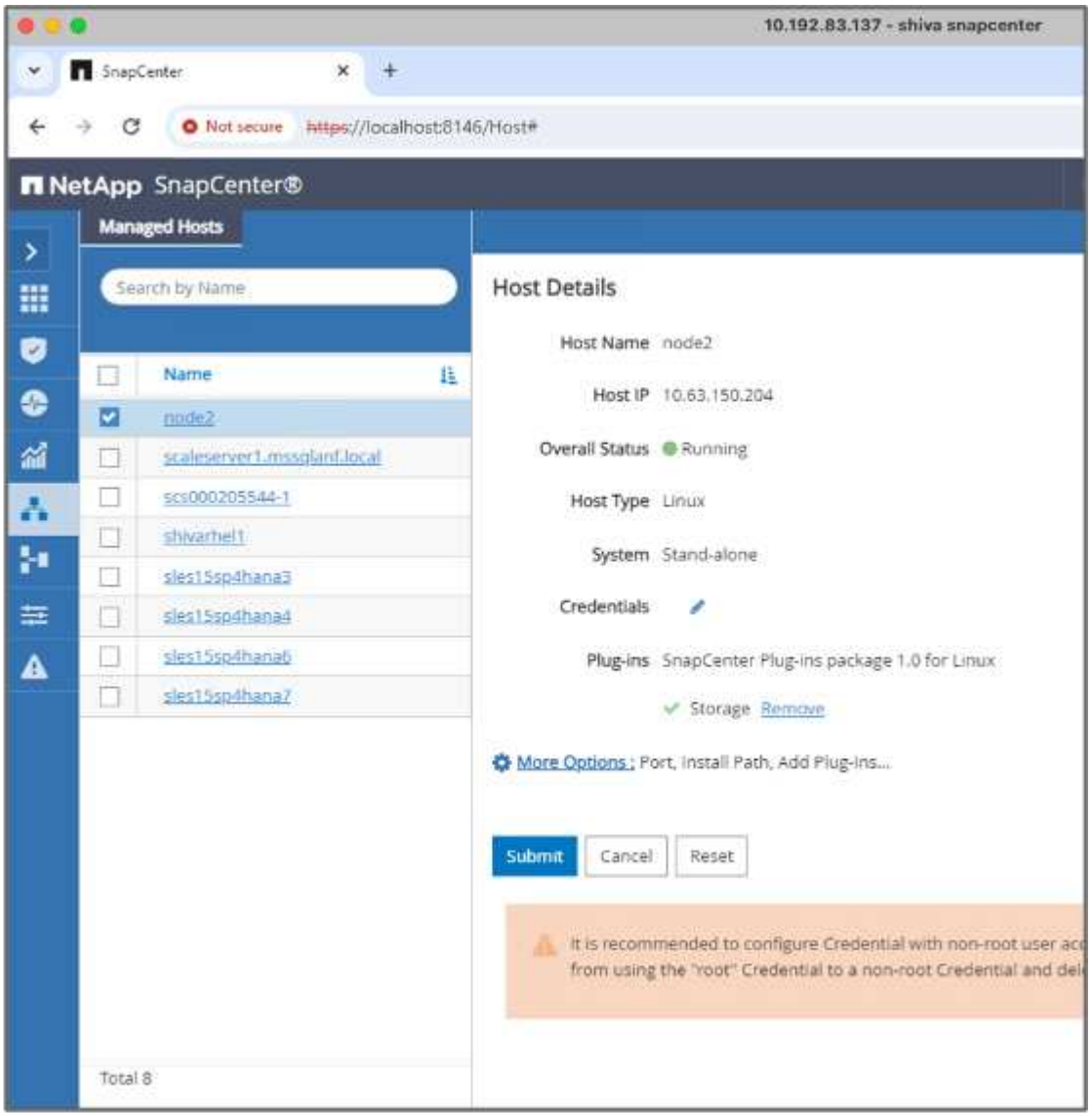
Schutz von Vektordatenbanken mithilfe von SnapCenter

Sicherung von Vector Datenbanken mit NetApp SnapCenter.

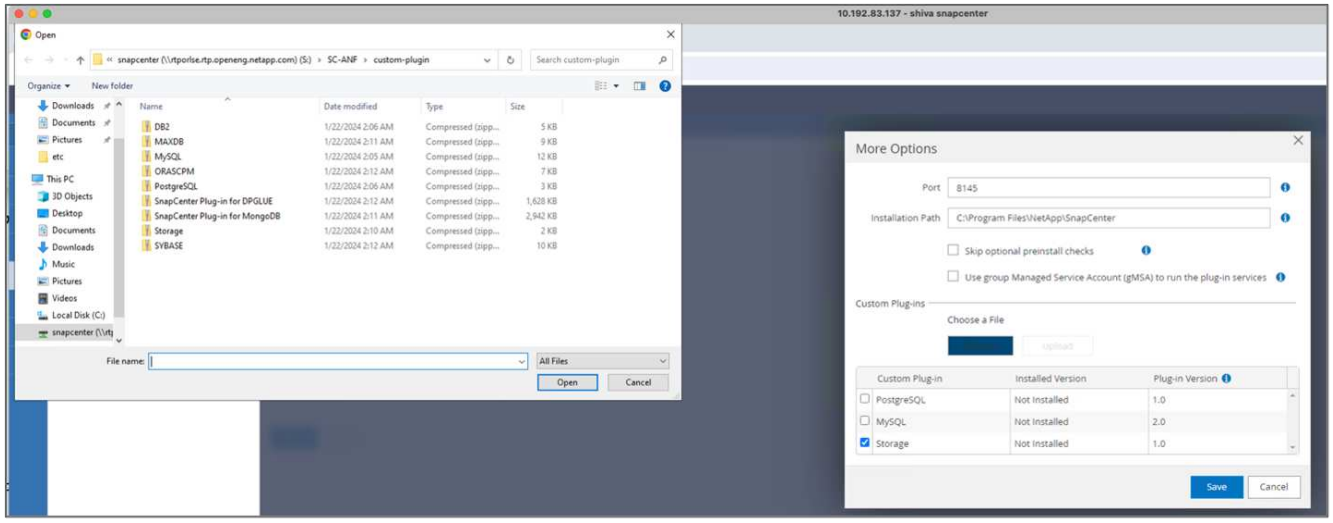
In der Filmproduktionsbranche verfügen Kunden beispielsweise oft über kritische eingebettete Daten wie Video- und Audiodateien. Der Verlust dieser Daten kann aufgrund von Problemen wie Festplattenausfällen erhebliche Auswirkungen auf ihren Betrieb haben und millionenschwere Projekte gefährden. Wir sind auf Fälle gestoßen, in denen unschätzbare Inhalte verloren gegangen sind, was zu erheblichen Störungen und

finanziellen Verlusten führte. Daher ist es in dieser Branche von höchster Bedeutung, die Sicherheit und Integrität dieser wichtigen Daten zu gewährleisten.
In diesem Abschnitt befassen wir uns eingehend damit, wie SnapCenter die Vektordatenbank-Daten und Milvus-Daten in ONTAP schützt. Für dieses Beispiel wurde ein NAS-Bucket (milvusdbvol1) verwendet, der von einem NFS-ONTAP-Volume (vol1) für Kundendaten abgeleitet wurde, und ein separates NFS-Volume (vectordbpv) für Milvus-Cluster-Konfigurationsdaten. Bitte prüfen Sie die "Hier" Des SnapCenter Backup Workflows

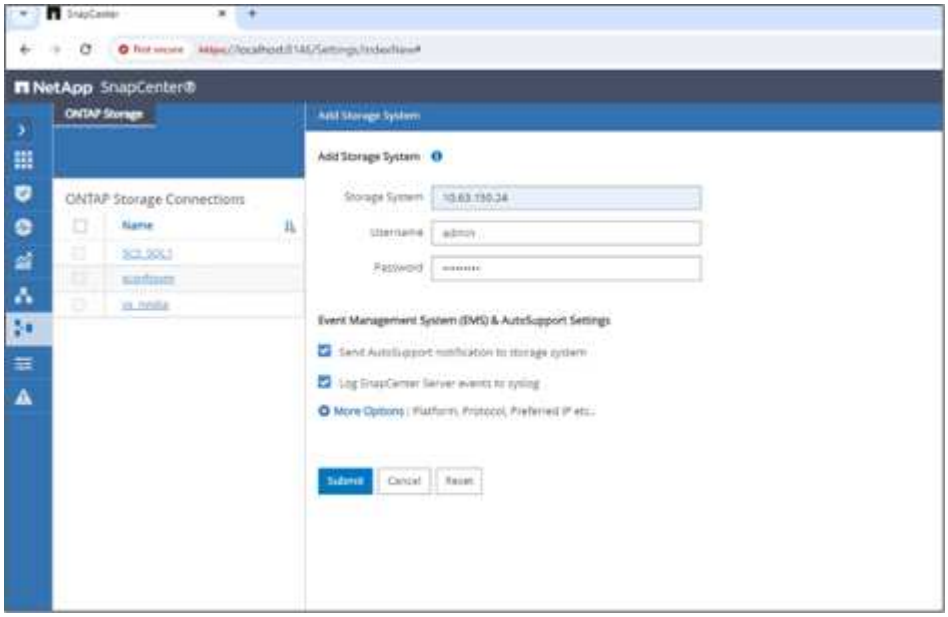
1. Richten Sie den Host ein, der zur Ausführung von SnapCenter-Befehlen verwendet wird.



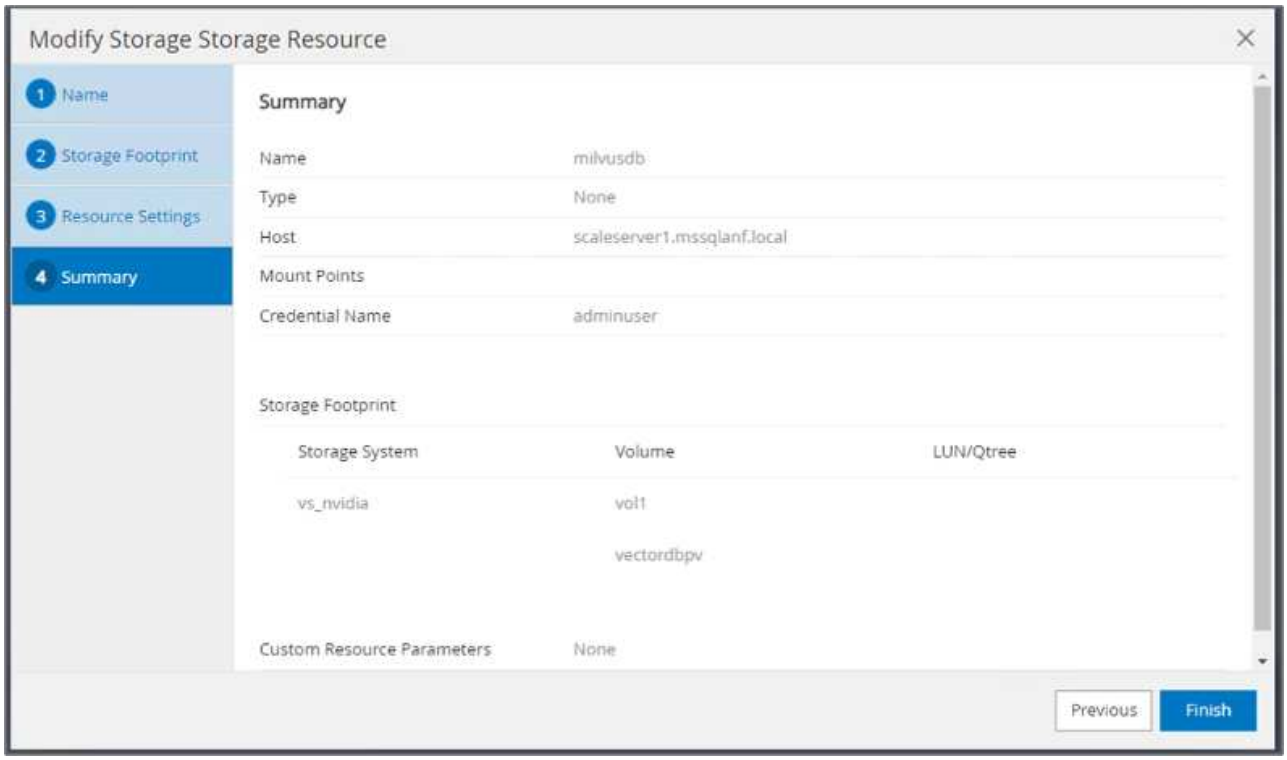
2. Installieren und konfigurieren Sie das Speicher-Plug-in. Wählen Sie aus dem hinzugefügten Host "More Options". Navigieren Sie zum heruntergeladenen Speicher-Plug-in, und wählen Sie es aus dem aus "NetApp Automation Store". Installieren Sie das Plugin und speichern Sie die Konfiguration.



3. Storage-System und Volume einrichten: Fügen Sie das Storage-System unter „Storage System“ hinzu und wählen Sie die SVM (Storage Virtual Machine) aus. In diesem Beispiel haben wir uns für „vs_nvidia“ entschieden.



4. Erstellen Sie eine Ressource für die Vektordatenbank, die eine Sicherungsrichtlinie und einen benutzerdefinierten Snapshot-Namen enthält.
   - Aktivieren Sie Consistency Group Backup mit Standardwerten und aktivieren Sie SnapCenter ohne Dateisystemkonsistenz.
   - Wählen Sie im Abschnitt Storage Footprint die Volumes aus, die mit den Kundendaten der Vektordatenbank und den Milvus-Clusterdaten verknüpft sind. In unserem Beispiel sind dies "vol1" und "vectordbpv".
   - Erstellen Sie eine Richtlinie für den Schutz der Vektordatenbank, und schützen Sie die Vektordatenbankressource mithilfe der Richtlinie.



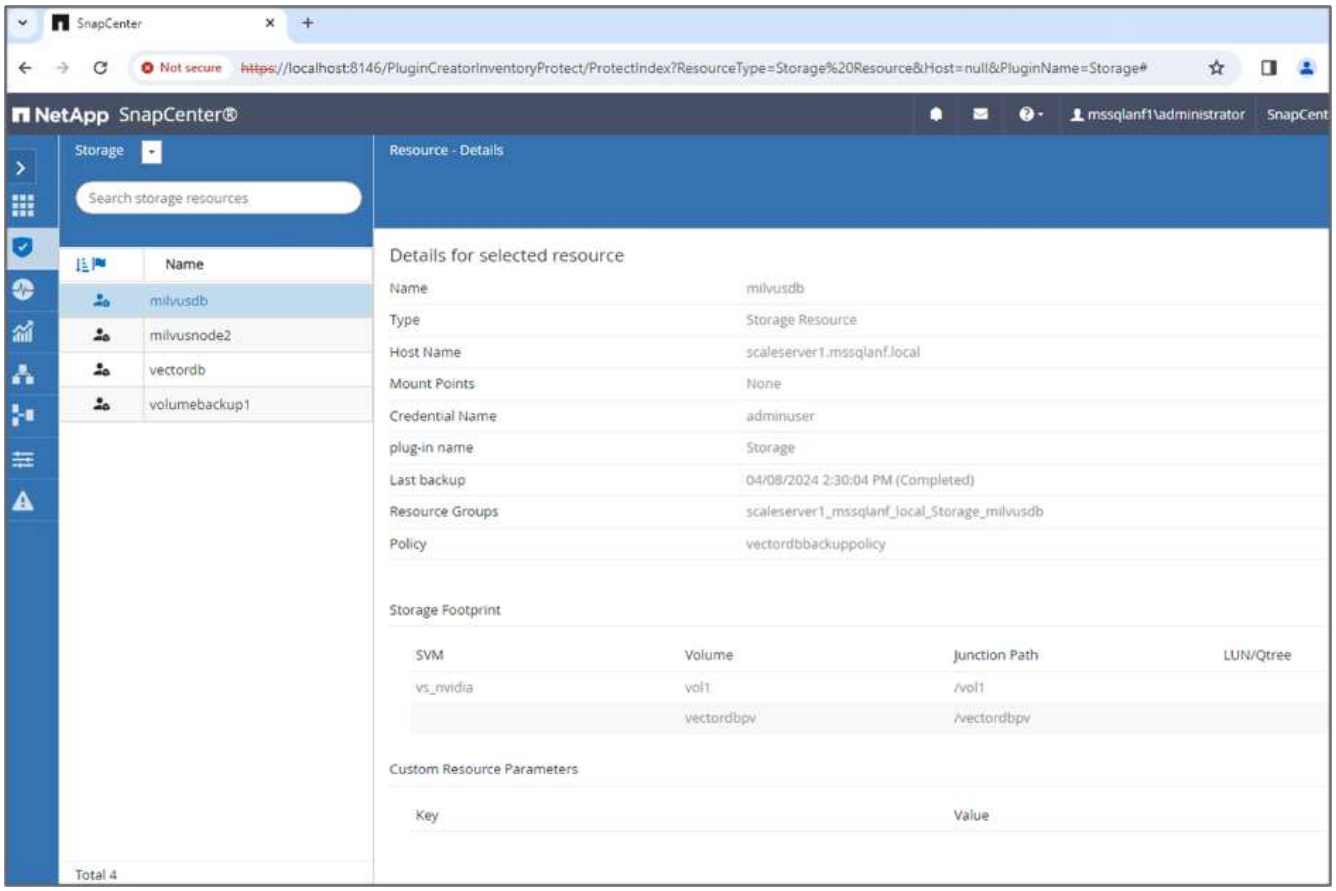
5. Fügen Sie Daten mithilfe eines Python-Skripts in den S3-NAS-Bucket ein. In unserem Fall haben wir das von Milvus bereitgestellte Backup-Skript 'prepare_Data_netapp.py' geändert und den 'sync'-Befehl ausgeführt, um die Daten vom Betriebssystem zu löschen.

```
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. Überprüfen der Daten im S3-NAS-Bucket In unserem Beispiel wurden die Dateien mit dem Zeitstempel '2024-04-08 21:22' vom Skript 'Prepare_Data_netapp.py' erstellt.

```
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. Initiieren Sie ein Backup mit dem CG-Snapshot (Consistency Group) von der Ressource 'milvusdb'

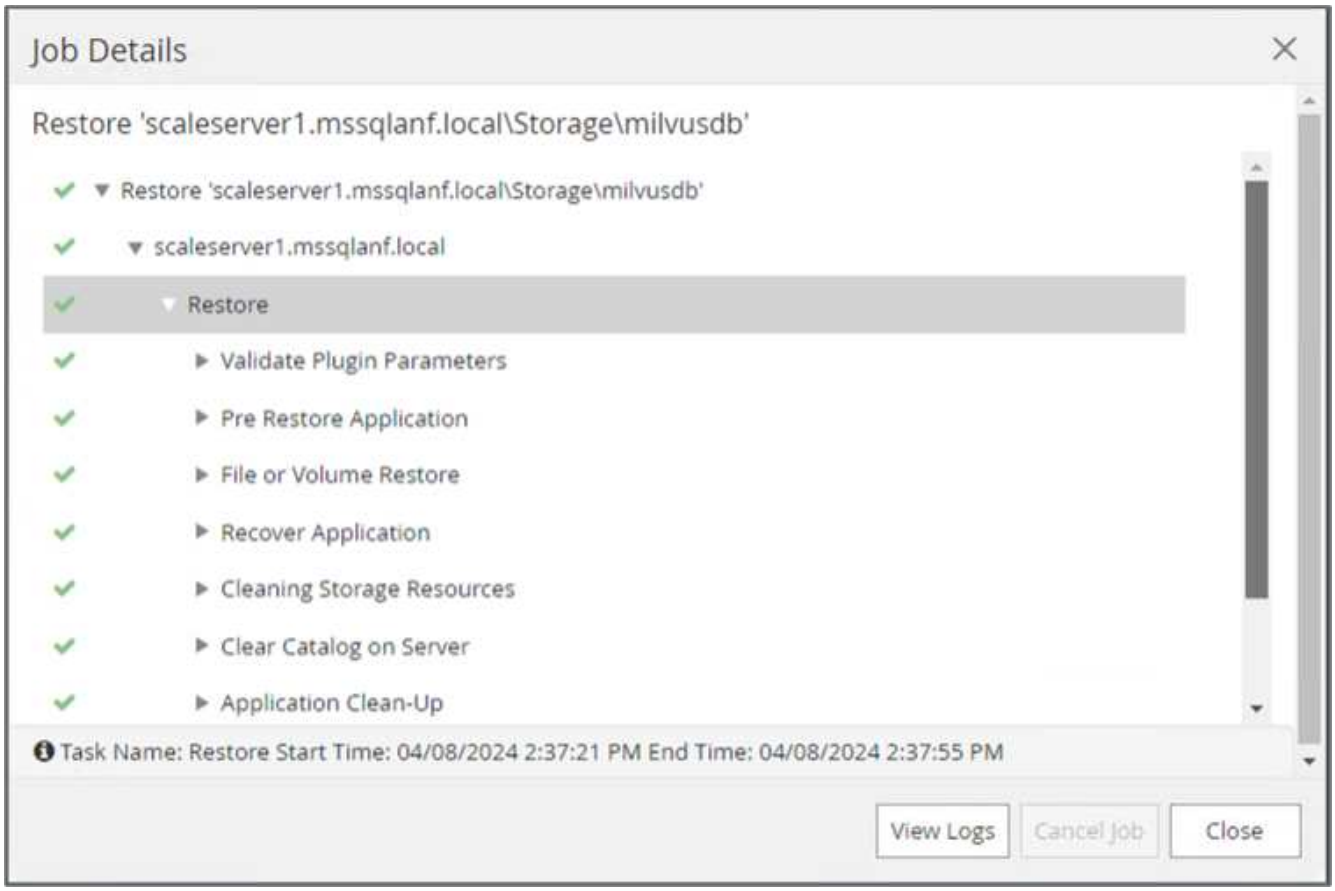


8. Zum Testen der Backup-Funktion haben wir nach dem Backup-Prozess entweder eine neue Tabelle hinzugefügt oder einige Daten aus dem NFS entfernt (S3 NAS Bucket).

   Stellen Sie sich für diesen Test ein Szenario vor, in dem nach dem Backup eine neue, unnötige oder unangemessene Sammlung erstellt wurde. In einem solchen Fall müssten wir die Vektordatenbank auf ihren Zustand zurücksetzen, bevor die neue Sammlung hinzugefügt wurde. So wurden beispielsweise neue Sammlungen wie 'Hello_milvus_netapp_sc_testnew' und 'Hello_milvus_netapp_sc_testnew2' eingefügt.

```
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. Führen Sie eine vollständige Wiederherstellung des S3-NAS-Buckets aus dem vorherigen Snapshot durch.



10. Verwenden Sie ein Python-Skript, um die Daten aus den Sammlungen „Hello_milvus_netapp_sc_Test“ und „Hello_milvus_netapp_sc_test2“ zu überprüfen.

```
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. Vergewissern Sie sich, dass die unnötige oder unangemessene Sammlung nicht mehr in der Datenbank vorhanden ist.

```
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:~#
```

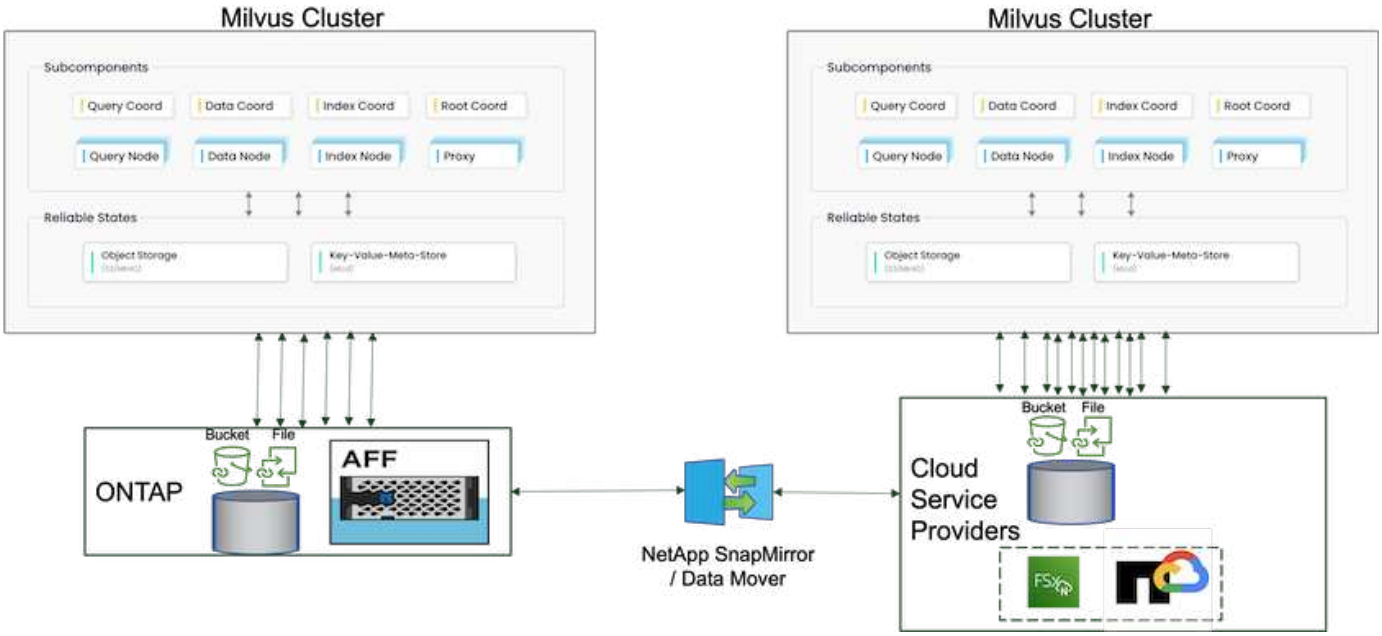
Zusammenfassend lässt sich feststellen, dass die Verwendung von NetApp SnapCenter zur Sicherung von Vektordatenbankdaten und Milvus-Daten in ONTAP erhebliche Vorteile für Kunden bietet, insbesondere in Branchen, in denen die Datenintegrität oberste Priorität hat, wie etwa der Filmproduktion. Durch die Fähigkeit von SnapCenter, konsistente Backups zu erstellen und vollständige Daten-Restores durchzuführen, wird sichergestellt, dass wichtige Daten wie integrierte Video- und Audiodateien vor Verlust durch Festplattenausfälle oder andere Probleme geschützt sind. Auf diese Weise werden nicht nur Betriebsunterbrechungen verhindert, sondern auch erhebliche finanzielle Verluste verhindert.

In diesem Abschnitt haben wir gezeigt, wie SnapCenter für den Schutz von in ONTAP gespeicherten Daten konfiguriert werden kann. Dazu gehören die Einrichtung von Hosts, die Installation und Konfiguration von Storage-Plug-ins sowie die Erstellung einer Ressource für die Vektordatenbank mit einem benutzerdefinierten Snapshot-Namen. Außerdem wurde gezeigt, wie ein Backup mit dem Snapshot der Consistency Group durchgeführt und die Daten im S3-NAS-Bucket verifiziert werden können.

Darüber hinaus haben wir ein Szenario simuliert, in dem nach dem Backup eine unnötige oder unangemessene Sammlung erstellt wurde. In solchen Fällen stellt die Möglichkeit von SnapCenter, eine vollständige Wiederherstellung aus einem früheren Snapshot durchzuführen, sicher, dass die Vektordatenbank vor dem Hinzufügen der neuen Sammlung in ihren Zustand zurückgesetzt werden kann. So wird die Integrität der Datenbank gewahrt. Diese Möglichkeit zur Wiederherstellung der Daten zu einem bestimmten Zeitpunkt ist für Kunden von unschätzbarem Wert und gibt ihnen die Gewissheit, dass ihre Daten nicht nur sicher sind, sondern auch ordnungsgemäß gepflegt werden. Somit bietet das SnapCenter-Produkt von NetApp Kunden eine robuste und zuverlässige Lösung für Datensicherung und -Management.

Disaster Recovery mit NetApp SnapMirror

Disaster Recovery mit NetApp SnapMirror



Disaster Recovery ist entscheidend für die Aufrechterhaltung der Integrität und Verfügbarkeit einer Vektordatenbank, insbesondere angesichts ihrer Rolle bei der Verwaltung von hochdimensionalen Daten und der Ausführung komplexer Ähnlichkeitssuchen. Eine gut geplante und implementierte Disaster-Recovery-Strategie stellt sicher, dass Daten im Falle von unvorhergesehenen Zwischenfällen, wie Hardwareausfällen, Naturkatastrophen oder Cyberangriffen nicht verloren gehen oder beschädigt werden. Dies gilt insbesondere für Applikationen, die auf Vektordatenbanken basieren und bei denen der Verlust oder die Beschädigung von Daten zu erheblichen Betriebsunterbrechungen und finanziellen Verlusten führen kann. Darüber hinaus sorgt ein robuster Disaster Recovery-Plan durch Minimierung der Ausfallzeiten für Business Continuity und ermöglicht eine schnelle Wiederherstellung von Services. Erreicht wird dies durch das NetApp Datenreplizierungsprodukt SnapMirror über verschiedene geografische Standorte, regelmäßige Backups und Failover-Mechanismen. Desaster Recovery ist daher nicht nur eine Schutzmaßnahme, sondern eine wichtige Komponente eines verantwortungsvollen und effizienten Vektordatenbankmanagements.

SnapMirror von NetApp ermöglicht die Datenreplizierung von einem NetApp ONTAP Storage Controller zu einem anderen, der in erster Linie für Disaster Recovery (DR) und Hybridlösungen verwendet wird. Im Kontext einer Vektordatenbank ermöglicht dieses Tool die reibungslose Transition der Daten zwischen On-Premises- und Cloud-Umgebungen. Diese Transition erfolgt ohne Datenkonvertierungen oder Refactoring von Applikationen, wodurch die Effizienz und Flexibilität des Datenmanagements über mehrere Plattformen hinweg verbessert wird.

Die NetApp Hybridlösung in einer Vektordatenbank kann weitere Vorteile bringen:

1. Skalierbarkeit: Die Hybrid-Cloud-Lösung von NetApp bietet die Möglichkeit, Ressourcen nach Bedarf zu skalieren. On-Premises-Ressourcen können für regelmäßige, vorhersehbare Workloads und Cloud-Ressourcen wie Amazon FSxN für NetApp ONTAP und Google Cloud NetApp Volume (GCNV) für Spitzenlasten oder unerwartete Workloads genutzt werden.
2. Kosteneffizienz: Das Hybrid-Cloud-Modell von NetApp ermöglicht eine Kostenoptimierung. Hierbei werden On-Premises-Ressourcen für normale Workloads genutzt und Cloud-Ressourcen nur nach Bedarf bezahlt. Dieses „Pay-as-you-go“-Modell kann mit dem Serviceangebot von NetApp Instaclustr recht kostengünstig sein. Für On-Premises- und große Cloud-Service-Provider bietet instaclustr Support und Beratung.
3. Flexibilität: Die Hybrid Cloud von NetApp gibt Ihnen die Flexibilität, den Speicherort der Daten frei zu wählen. Sie können beispielsweise komplexe Vektoroperationen vor Ort durchführen, bei denen Sie über eine leistungsstärkere Hardware und weniger intensive Vorgänge in der Cloud verfügen.

4. Business Continuity: Wenn Ihre Daten in einer NetApp Hybrid Cloud verfügbar sind, kann bei einem Ausfall die Business Continuity gewährleistet werden. Wenn Ihre On-Premises-Ressourcen betroffen sind, können Sie schnell zur Cloud wechseln. Mit NetApp SnapMirror können wir die Daten von On-Premises-Systemen in die Cloud verschieben und umgekehrt.
5. Innovation: Die Hybrid-Cloud-Lösungen von NetApp ermöglichen durch Zugriff auf hochmoderne Cloud-Services und -Technologien außerdem schnellere Innovationen. NetApp-Innovationen in der Cloud, wie Amazon FSxN für NetApp ONTAP, Azure NetApp Files und Google Cloud NetApp Volumes, sind innovative Produkte von Cloud-Service-Providern und bevorzugtes NAS.

Performance-Validierung Der Vector Datenbank

Performance-Validierung

Die Performance-Validierung spielt sowohl in Vektordatenbanken als auch in Storage-Systemen eine wichtige Rolle und dient als ein wichtiger Faktor für den optimalen Betrieb und die effiziente Ressourcenauslastung. Vector-Datenbanken, die für die Verarbeitung von hochdimensionalen Daten und die Ausführung von Ähnlichkeitssuchen bekannt sind, müssen ein hohes Leistungsniveau aufweisen, um komplexe Abfragen schnell und präzise verarbeiten zu können. Durch die Performance-Validierung werden Engpässe identifiziert und Konfigurationen feinangepasst. Außerdem wird sichergestellt, dass das System mit erwarteten Workloads umgehen kann, ohne dass der Service beeinträchtigt wird. Ebenso ist in Storage-Systemen die Performance-Validierung wichtig, um sicherzustellen, dass Daten effizient gespeichert und abgerufen werden, ohne Latenzprobleme oder Engpässe, die die Systemperformance insgesamt beeinträchtigen könnten. Darüber hinaus unterstützt sie fundierte Entscheidungen bei erforderlichen Upgrades oder Änderungen der Storage-Infrastruktur. Daher ist die Performance-Validierung ein entscheidender Aspekt des Systemmanagements und trägt erheblich zur Aufrechterhaltung einer hohen Servicequalität, Betriebseffizienz und der allgemeinen Systemzuverlässigkeit bei.

In diesem Abschnitt möchten wir uns mit der Performance-Validierung von Vektordatenbanken wie Milvus und pgvecto.rs beschäftigen. Dabei konzentrieren wir uns auf die Speicherleistungsmerkmale wie I/O-Profil und das Verhalten des NetApp-Speichercontrollers zur Unterstützung von RAG- und Inferenzarbeitslasten innerhalb des LLM-Lebenszyklus. Wir bewerten und ermitteln jegliche Performance-Unterscheidungsmerkmale, wenn diese Datenbanken mit der ONTAP Storage-Lösung kombiniert werden. Unsere Analyse basiert auf wichtigen Leistungsindikatoren, wie der Anzahl der pro Sekunde verarbeiteten Abfragen (QPS).

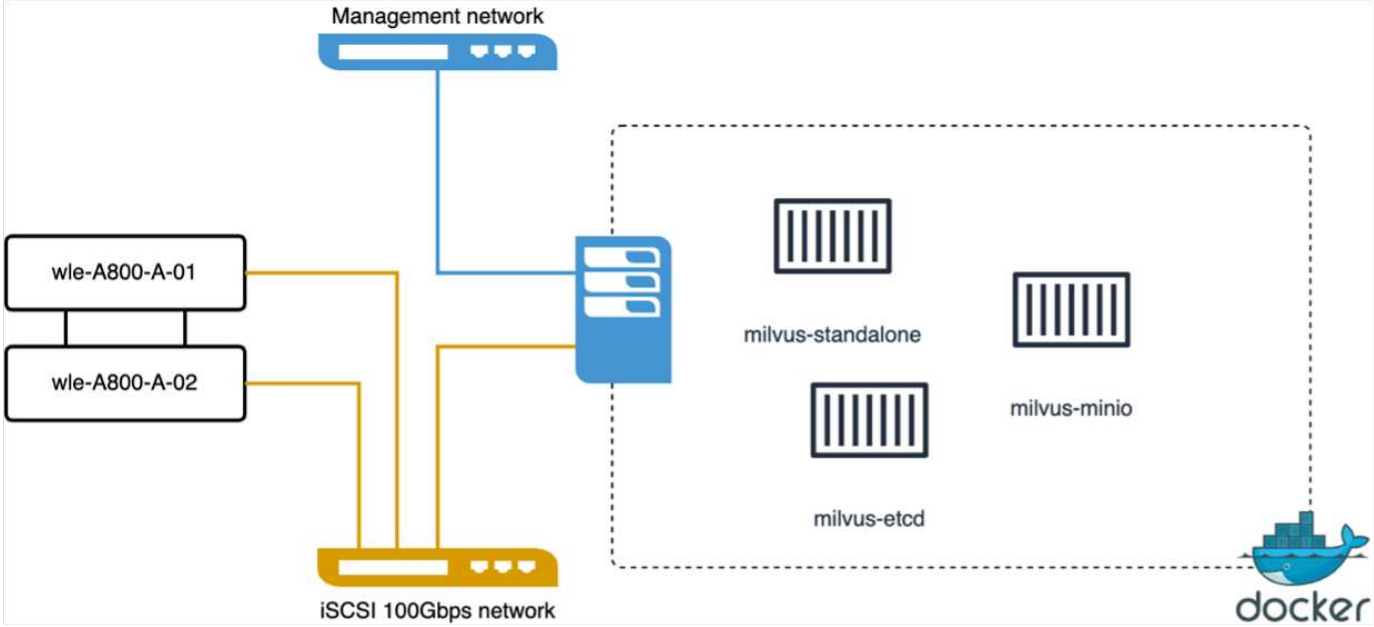
Bitte überprüfen Sie die für milvus verwendete Methode und den Fortschritt unten.

Details	Milvus (Standalone und Cluster)	Postgres(pgvecto.rs)
Version	2.3.2	0.2.0
Dateisystem	XFS auf iSCSI-LUNs	
Workload Generator	"VectorDB-Bank" – V0.0.5	
Datensätze	LAION-Datensatz * 10 Millionen Einbettungen * 768 Abmessungen * ~300 GB Datensatz Größe	

VectorDB-Bench mit Milvus Standalone Cluster

Wir haben die folgende Performance-Validierung auf milvus Standalone Cluster mit vectorDB-Bench durchgeführt.

Die Netzwerk- und Serverkonnektivität des eigenständigen milvus-Clusters ist unten angegeben.



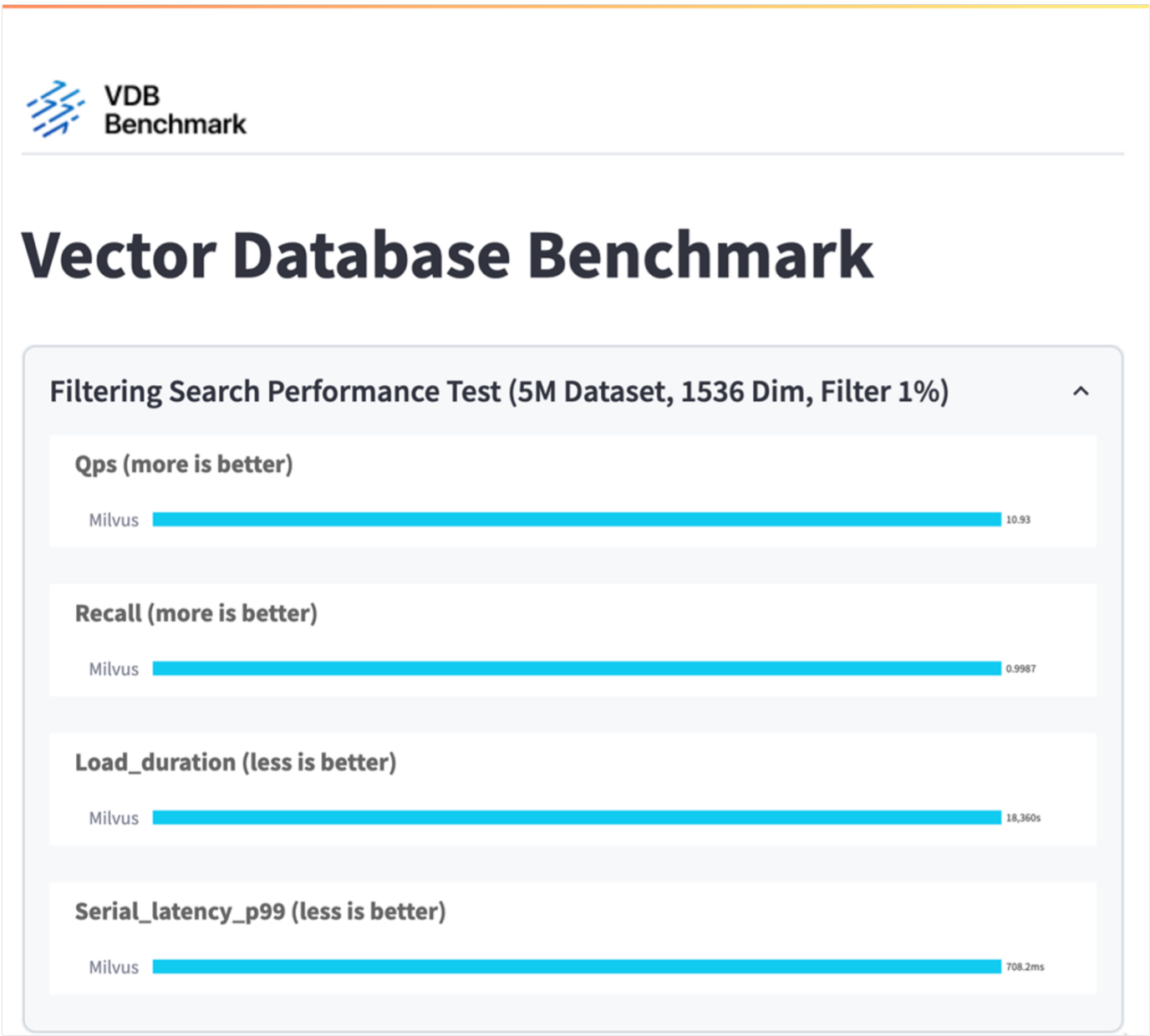
In diesem Abschnitt geben wir unsere Beobachtungen und Ergebnisse aus dem Testen der eigenständigen Datenbank von Milvus weiter.
. Wir haben DiskANN als Indextyp für diese Tests ausgewählt.
. Die Aufnahme, Optimierung und Erstellung von Indizes für einen ungefähr 100 GB großen Datensatz dauerte etwa 5 Stunden. Während der meisten dieser Dauer war der Milvus Server, ausgestattet mit 20 Kernen (was 40 vcpus entspricht, wenn Hyper-Threading aktiviert ist), mit seiner maximalen CPU-Kapazität von 100% betrieben.Wir fanden, dass DiskANN besonders wichtig ist für große Datensätze, die die Größe des Systemspeichers überschreiten.
. In der Abfragephase beobachteten wir eine Rate von Abfragen pro Sekunde (QPS) von 10.93 mit einem Rückruf von 0.9987. Die 99. Perzentillatenz für Abfragen wurde bei 708.2 Millisekunden gemessen.

Aus Sicht des Storage hat die Datenbank während der Aufnahme, der Optimierung nach dem Einfügen und der Indexerstellung ca. 1,000 OPs/s ausgegeben. In der Abfragephase wurden 32,000 OPs/s gefordert

Im folgenden Abschnitt werden die Storage-Performance-Kennzahlen vorgestellt.

Workload-Phase	Metrisch	Wert
Datenaufnahme Und Optimierung nach dem Einfügen	IOPS	< 1,000
	Latenz	< 400 usecs
	Workload	Lese-/Schreib-Kombination, überwiegend Schreibzugriffe
	I/O-Größe	64 KB
Abfrage	IOPS	Spitze bei 32,000
	Latenz	< 400 usecs
	Workload	100 % gecachte Lesevorgänge
	I/O-Größe	Hauptsächlich 8 KB

Das vectorDB-Bench-Ergebnis ist unten.

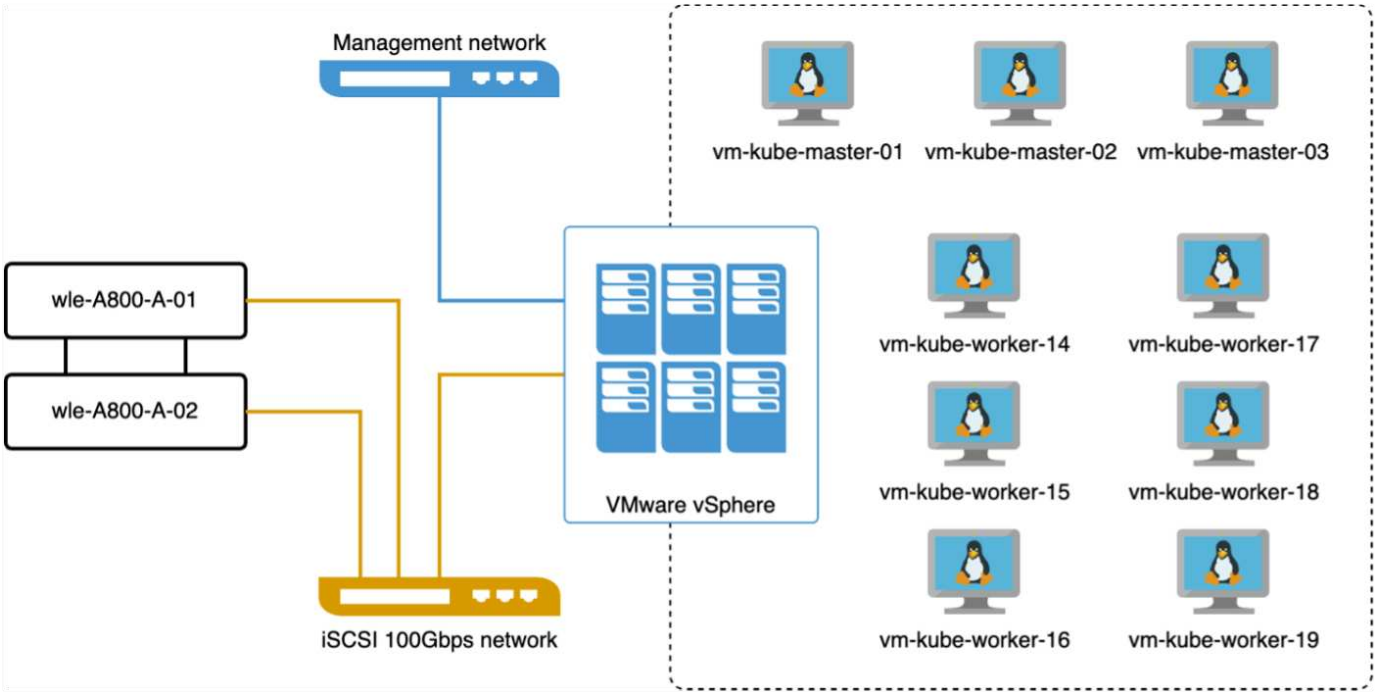


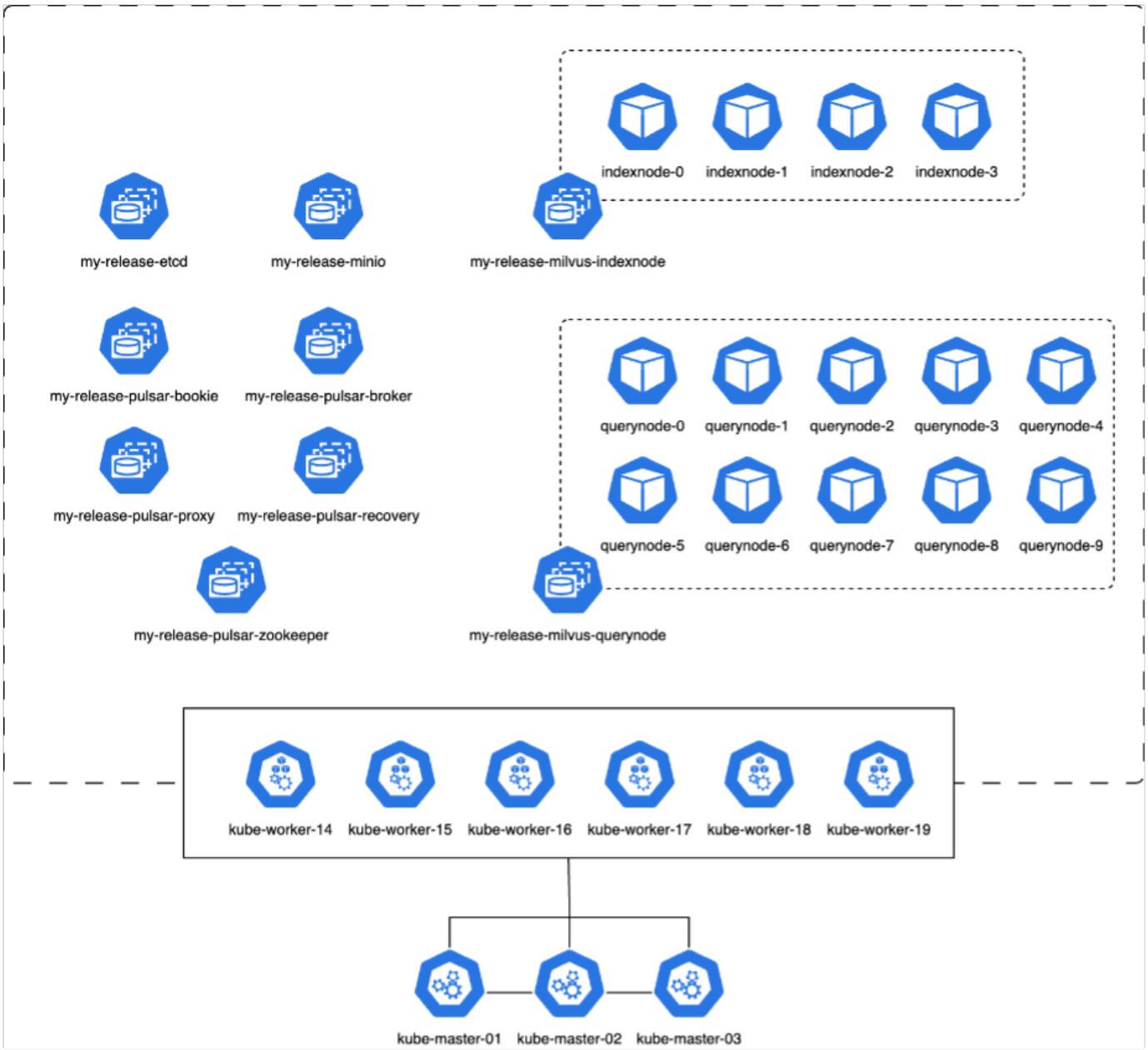
Aus der Performance-Validierung der eigenständigen Milvus-Instanz geht hervor, dass das aktuelle Setup nicht ausreicht, um einen Datensatz von 5 Millionen Vektoren mit einer Dimensionalität von 1536 zu unterstützen. Wir haben festgestellt, dass der Storage über ausreichende Ressourcen verfügt und keinen Engpass im System darstellt.

VectorDB-Bank mit milvus Cluster

In diesem Abschnitt wird die Implementierung eines Milvus Clusters in einer Kubernetes-Umgebung behandelt. Diese Kubernetes-Einrichtung wurde auf einer VMware vSphere Implementierung aufgebaut, in der die Kubernetes-Master- und -Worker-Nodes gehostet wurden.

Details zu Implementierungen von VMware vSphere und Kubernetes finden Sie in den folgenden Abschnitten.


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



In diesem Abschnitt stellen wir unsere Beobachtungen und Ergebnisse aus dem Testen der Milvus-Datenbank vor.
* Der verwendete Indextyp war DiskANN.
* Die folgende Tabelle bietet einen Vergleich zwischen den Standalone- und Cluster-Bereitstellungen bei der Arbeit mit 5 Millionen Vektoren bei einer Dimensionalität von 1536. Dabei stellten wir fest, dass die Zeit für die Datenaufnahme und die Optimierung nach dem Einfügen in das Cluster geringer war. Die 99. Perzentillatenz bei Abfragen wurde in der Cluster-Implementierung im Vergleich zum Standalone-Setup um das Sechsfache verringert.
* Obwohl die Rate für Abfragen pro Sekunde (QPS) in der Cluster-Bereitstellung höher war, war sie nicht auf dem gewünschten Niveau.

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%

Die nachfolgenden Abbildungen bieten eine Übersicht über verschiedene Storage-Kennzahlen, einschließlich der Storage-Cluster-Latenz und der gesamten IOPS (Input/Output Operations per Second).

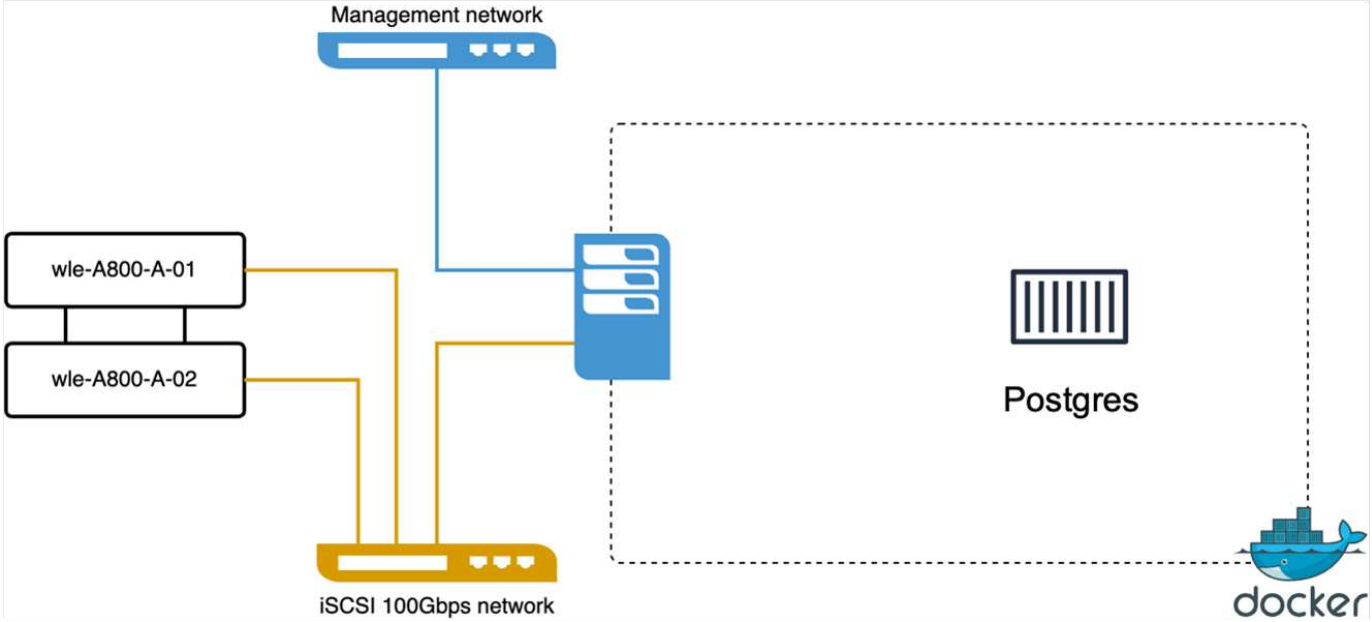
Im folgenden Abschnitt werden die wichtigsten Kennzahlen zur Storage-Performance aufgeführt.

Workload-Phase	Metrisch	Wert
Datenaufnahme Und Optimierung nach dem Einfügen	IOPS	< 1,000
	Latenz	< 400 usecs
	Workload	Lese-/Schreib-Kombination, überwiegend Schreibzugriffe
	I/O-Größe	64 KB
Abfrage	IOPS	Spitze bei 147,000
	Latenz	< 400 usecs
	Workload	100 % gecachte Lesevorgänge
	I/O-Größe	Hauptsächlich 8 KB

Basierend auf der Performance-Validierung des eigenständigen Milvus- und des Milvus-Clusters werden die Details des Storage-I/O-Profils dargestellt.
* Wir stellten fest, dass das I/O-Profil sowohl bei Standalone- als auch bei Cluster-Implementierungen konsistent bleibt.
* Der beobachtete Unterschied bei den IOPS-Spitzenwerten kann auf die größere Anzahl von Clients in der Cluster-Bereitstellung zurückgeführt werden.

VektorDB-Bank mit Postgres (pgvecto.rs)

Wir haben folgende Aktionen auf PostgreSQL(pgvecto.rs) mit VectorDB-Bench durchgeführt:
Die Details bezüglich der Netzwerk- und Serveranbindung von PostgreSQL (insbesondere pgvecto.rs) lauten wie folgt:

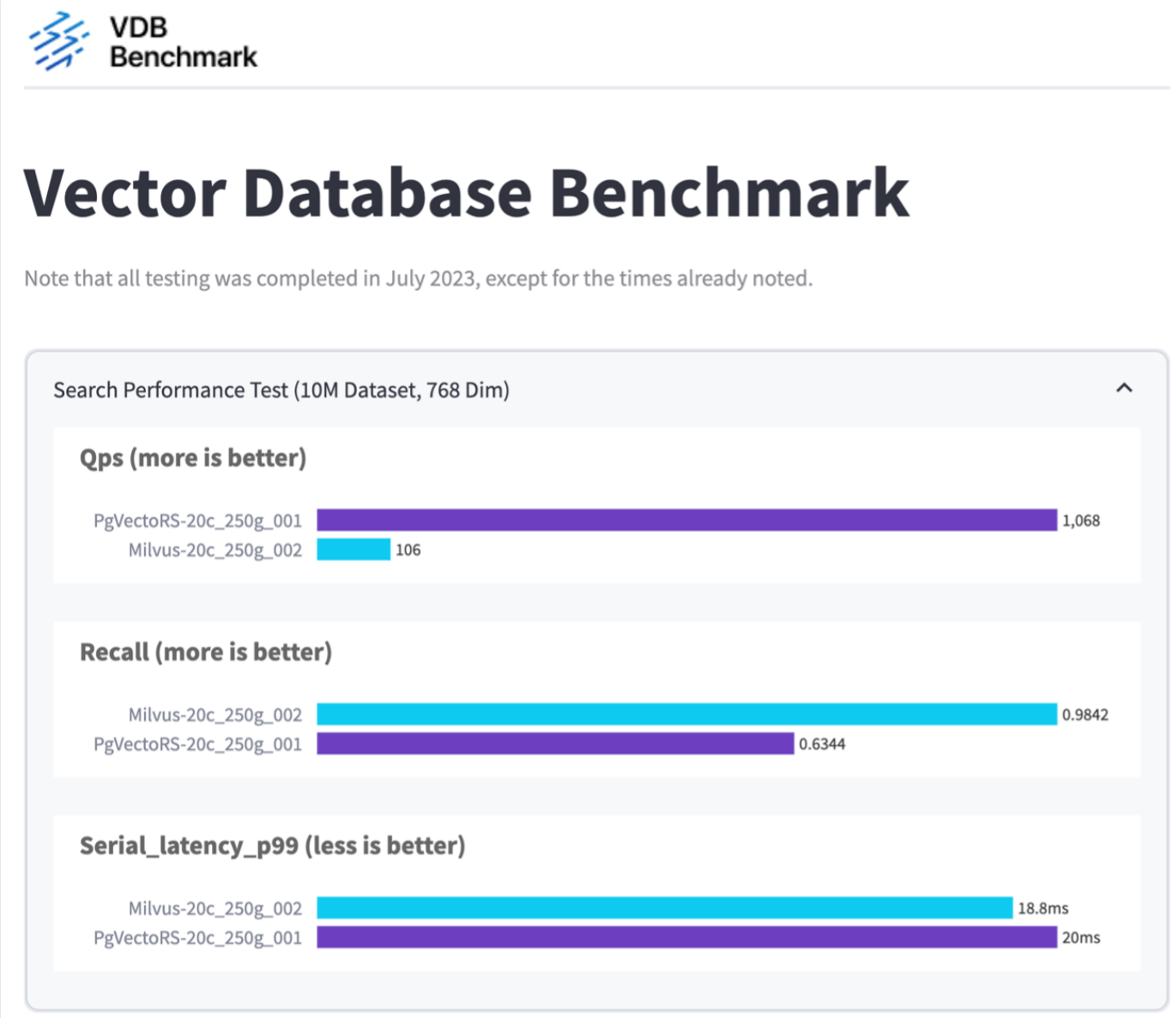


In diesem Abschnitt stellen wir unsere Beobachtungen und Ergebnisse aus dem Testen der PostgreSQL-Datenbank vor, insbesondere mit pgvecto.rs.
* Wir haben HNSW als Indextyp für diese Tests ausgewählt, weil zum Zeitpunkt des Tests DiskANN für pgvecto.rs nicht verfügbar war.
* Während der Datenaufnahme haben wir den Cohe-Datensatz geladen, der aus 10 Millionen Vektoren bei einer Dimensionalität von 768 besteht. Dieser Vorgang dauerte ungefähr 4.5 Stunden.
* In der Abfragephase beobachteten wir eine Rückruffunktrate von 1,068 Abfragen pro Sekunde (QPS) mit einem Rückruffunktsatz von 0.6344. Die 99. Perzentillatenz für Abfragen wurde bei 20 Millisekunden gemessen. Während der meisten Laufzeit wurde die Client-CPU mit 100 % Kapazität betrieben.

Die folgenden Abbildungen bieten eine Übersicht über verschiedene Storage-Kennzahlen, einschließlich der Gesamt-IOPS für die Storage-Cluster-Latenz (Input/Output Operations per Second).


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["Fehler: Fehlendes Grafikbild"]
```

Leistungsvergleich zwischen milvus und Postgres auf der Vektor-DB-Bank



Basierend auf unserer Leistungsvalidierung von Milvus und PostgreSQL mit VectorDBBench konnten wir Folgendes beobachten:

- Indextyp: HNSW
- Datensatz: Cohere mit 10 Millionen Vektoren bei 768 Dimensionen

Wir fanden heraus, dass pgvecto.rs mit einem Rückruf von 0.6344 eine Queries per second (QPS)-Rate von 1,068 erreichte, während Milvus mit einem Rückruf von 0.9842 eine QPS-Rate von 106 erreichte.

Wenn hohe Präzision in Ihren Abfragen Priorität hat, übertrifft Milvus pgvecto.rs, da es einen höheren Anteil relevanter Elemente pro Abfrage abruft. Wenn jedoch die Anzahl der Abfragen pro Sekunde ein entscheidender Faktor ist, übersteigt pgvecto.rs Milvus. Es ist jedoch wichtig zu beachten, dass die Qualität der über pgvecto.rs abgerufenen Daten niedriger ist, wobei etwa 37% der Suchergebnisse irrelevante Elemente sind.

Beobachtung basierend auf unseren Leistungsvalidierungen:

Basierend auf unseren Leistungsvalidierungen haben wir folgende Beobachtungen gemacht:

In Milvus ähnelt das I/O-Profil einem OLTP-Workload, beispielsweise in Oracle SLOB. Der Benchmark besteht aus drei Phasen: Datenaufnahme, Post-Optimierung und Abfrage. Die Anfangsphasen sind in erster Linie durch 64-KB-Schreibvorgänge gekennzeichnet, während die Abfragephase überwiegend 8-KB-Lesevorgänge umfasst. Wir erwarten, dass ONTAP die E/A-Last von Milvus kompetent verarbeitet.

Das PostgreSQL-I/O-Profil stellt keinen anspruchsvollen Storage Workload dar. Angesichts der aktuell laufenden in-Memory-Implementierung haben wir während der Abfragephase keine Festplatten-I/O beobachtet.

DiskANN entwickelt sich zu einer entscheidenden Technologie zur Differenzierung von Storage. Es ermöglicht die effiziente Skalierung der Vektor-DB-Suche über die Systemspeichergrenze hinaus. Es ist jedoch unwahrscheinlich, dass sich die Storage-Performance durch in-Memory-Vektor-DB-Indizes wie HNSW differenziert.

Es ist auch erwähnenswert, dass die Speicherung während der Abfragephase keine wichtige Rolle spielt, wenn der Indextyp HSNW ist, die wichtigste Betriebsphase für Vektordatenbanken, die RAG-Anwendungen unterstützen. Die Folge ist, dass die Storage Performance diese Applikationen nicht wesentlich beeinträchtigt.