Panoramica di verifica della soluzione

NetApp Solutions

NetApp
May 14, 2024

Sommario

Panoramica della soluzione

Abbiamo condotto una convalida completa della soluzione incentrata su cinque aree chiave, i dettagli delle quali sono descritti di seguito. Ogni sezione analizza le sfide affrontate dai clienti, le soluzioni fornite da NetApp e i successivi benefici per il cliente.

1. "Setup del cluster Milvus con Kubernetes on-premise"
   I clienti affrontano la sfida di scalare in maniera indipendente su storage e calcolo, un'efficace gestione dei dati e dell'infrastruttura. In questa sezione, descriviamo in dettaglio il processo di installazione di un cluster Milvus su Kubernetes, che utilizza uno storage controller NetApp sia per i dati del cluster sia per i dati del cliente.
2. "Milvus con Amazon FSxN per NetApp ONTAP – dualismo di file e oggetti"
   In questa sezione, perché è necessario distribuire il database vettoriale nel cloud e i passaggi per distribuire il database vettoriale (milvus standalone) in Amazon FSxN per NetApp ONTAP all'interno dei contenitori docker.
3. "Protezione del database vettoriale tramite NetApp SnapCenter."
   In questa sezione, analizzeremo come SnapCenter protegge i dati del database vettoriale e i dati Milvus che risiedono in ONTAP. Per questo esempio, abbiamo utilizzato un bucket NAS (milvusdbvol1) derivato da un volume NFS ONTAP (vol1) per i dati dei clienti e un volume NFS separato (vectordbpv) per i dati di configurazione del cluster Milvus.
4. "Disaster recovery con NetApp SnapMirror"
   In questa sezione, verranno descritte l'importanza del ripristino di emergenza (DR) per database vettoriali e il modo in cui il prodotto di ripristino di emergenza NetApp snapmirror fornisce la soluzione di DR al database vettoriale.
5. "Convalida delle performance"
   In questa sezione, il nostro obiettivo è analizzare la convalida delle prestazioni dei database vettoriali, come Milvus e pgvecto.rs, concentrandoci sulle caratteristiche delle prestazioni di storage, come il profilo i/o e il comportamento dello storage controller NetApp a supporto di RAG e carichi di lavoro di inferenza all'interno del ciclo di vita LLM. Valuteremo e identificheremo eventuali elementi di differenziazione delle performance quando questi database sono combinati con la soluzione di storage ONTAP. La nostra analisi si baserà su indicatori chiave delle prestazioni, ad esempio il numero di query elaborate al secondo (QPS).

Setup cluster Milvus con Kubernetes on-premise

Setup del cluster Milvus con Kubernetes on-premise

Le sfide dei clienti di scalare in maniera indipendente su storage e calcolo, un'efficace gestione dell'infrastruttura e i dati
Kubernetes e i database vettoriali formano insieme una soluzione potente e scalabile per la gestione di operazioni sui dati di grandi dimensioni. Kubernetes ottimizza le risorse e gestisce i container, mentre i database vettoriali gestiscono in modo efficiente dati ad alta dimensione e ricerche di similarità. Questa combinazione consente una rapida elaborazione di query complesse su grandi set di dati e si adatta perfettamente ai volumi di dati in crescita, rendendola ideale per applicazioni big data e workload ai.

1. In questa sezione, descriviamo in dettaglio il processo di installazione di un cluster Milvus su Kubernetes, che utilizza uno storage controller NetApp sia per i dati del cluster sia per i dati del cliente.
2. Per installare un cluster Milvus, sono necessari volumi persistenti (PVS) per memorizzare i dati di vari componenti del cluster Milvus. Questi componenti includono etcd (tre istanze), Pulsar-bookie-journal (tre istanze), Pulsar-bookie-ledgers (tre istanze) e Pulsar-zookeeper-data (tre istanze).

In milvus cluster, possiamo utilizzare sia Pulsar o kafka per il motore sottostante che supporta la memorizzazione affidabile del cluster Milvus e la pubblicazione/sottoscrizione di flussi di messaggi. Per Kafka con NFS, NetApp ha apportato miglioramenti in ONTAP 9.12.1 e versioni successive. Tali miglioramenti, insieme alle modifiche a NFSv4,1 e Linux incluse in RHEL 8,7 o 9,1 e versioni successive, risolvono il problema del "ridenominazione semplice" che si verifica quando si esegue Kafka su NFS. Se siete interessati a informazioni più approfondite sull'argomento dell'esecuzione di kafka con la soluzione NFS NetApp, consultate - "questo link".

3. Abbiamo creato un singolo volume NFS di NetApp ONTAP e abbiamo stabilito 12 volumi persistenti con 250GB TB di storage, ciascuno. Le dimensioni dello storage possono variare in base alle dimensioni del cluster, ad esempio disponiamo di un altro cluster in cui ogni PV ha 50GB TB. Per ulteriori dettagli, fare riferimento a uno dei file YAML PV riportati di seguito; in totale, sono disponibili 12 file. In ogni file, storageClassName è impostato su 'default', e l'archiviazione e il percorso sono univoci per ogni PV.

```
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. Eseguire il comando 'kubectl apply' per ogni file YAML PV per creare i volumi persistenti, quindi verificare la loro creazione utilizzando '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. Per l'archiviazione dei dati dei clienti, Milvus supporta soluzioni di storage a oggetti come MinIO, BLOB di Azure e S3. In questa guida, utilizziamo S3. I seguenti passaggi si applicano sia a ONTAP S3 che all'archivio oggetti StorageGRID. Utilizziamo Helm per implementare il cluster Milvus. Scaricare il file di configurazione, Values.yaml, dal percorso di download di Milvus. Fare riferimento all'appendice per il file Values.yaml utilizzato in questo documento.
6. Assicurarsi che la 'torageClass' sia impostata su 'default' in ogni sezione, compresi quelli per il log, etcd, zoookeeper e bookkeeper.
7. Nella sezione MinIO, disabilitare MinIO.
8. Creare un bucket NAS dallo storage a oggetti ONTAP o StorageGRID e includerli in un S3 esterno con le credenziali di storage a oggetti.

```
##################################
# 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. Prima di creare il cluster Milvus, assicurarsi che il PVC (PersistentVolumeClaim) non disponga di risorse preesistenti.

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

10. Utilizzare Helm e il file di configurazione values.yaml per installare e avviare il cluster Milvus.

```
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. Verificare lo stato delle richieste di verifica del volume di persistenza (PVC).

```
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. Controllare lo stato dei pod.

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

Assicurarsi che lo stato dei pod sia "in esecuzione" e funzioni come previsto

13. Testare la scrittura e la lettura dei dati nello storage a oggetti Milvus e NetApp.
    - Scrivere i dati utilizzando il programma Python "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:~#
```

- Leggere i dati utilizzando il file Python "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}}]}
```

In base alla validazione sopra indicata, l'integrazione di Kubernetes con un database vettoriale, come dimostrata tramite l'implementazione di un cluster Milvus su Kubernetes che utilizza uno storage controller NetApp, offre ai clienti una soluzione solida, scalabile ed efficiente per gestire operazioni su dati su larga scala. Questo setup offre ai clienti la capacità di gestire dati ad alta dimensione ed eseguire query complesse in modo rapido ed efficiente, rendendolo la soluzione ideale per applicazioni big data e workload ai. L'utilizzo dei volumi persistenti (PV) per vari componenti del cluster, insieme alla creazione di un singolo volume NFS da NetApp ONTAP, garantisce un utilizzo ottimale delle risorse e una gestione dei dati. Il processo di verifica dello stato di PersistentVolumeClaims (PVCS) e POD,

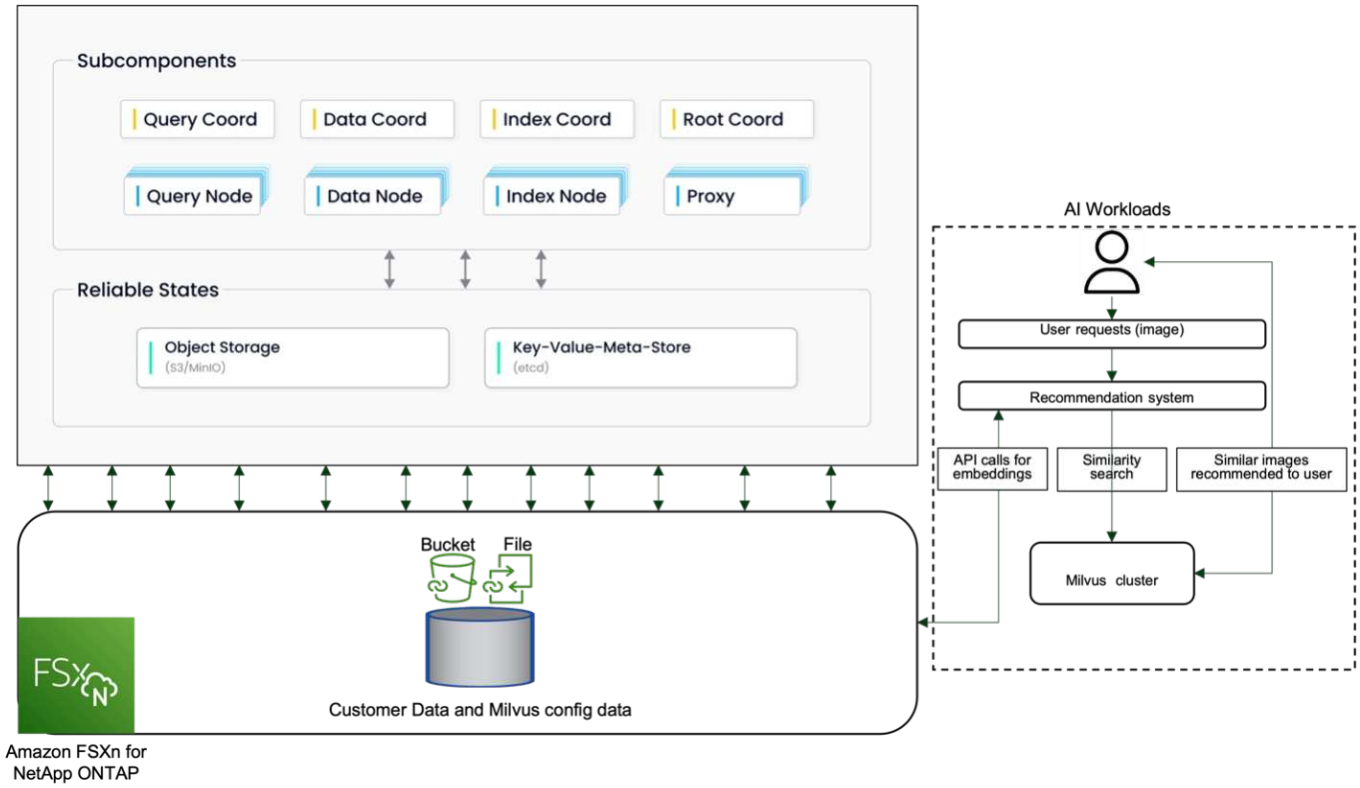
nonché di verifica della scrittura e della lettura dei dati, fornisce ai clienti la garanzia di operazioni di dati affidabili e coerenti. L'utilizzo dello storage a oggetti ONTAP o StorageGRID per i dati dei clienti migliora ulteriormente l'accessibilità e la sicurezza dei dati. Nel complesso, questo setup offre ai clienti una soluzione per la gestione dei dati resiliente e ad alte performance, in grado di scalare perfettamente con le crescenti esigenze in termini di dati.

Milvus con Amazon FSxN per NetApp ONTAP - dualità di file e oggetti

Milvus con Amazon FSxN per NetApp ONTAP – dualismo di file e oggetti

In questa sezione, perché è necessario implementare il database vettoriale nel cloud e i passaggi per distribuire il database vettoriale (milvus standalone) in Amazon FSxN per NetApp ONTAP all'interno dei contenitori docker.

L'implementazione di un database vettoriale nel cloud offre diversi vantaggi significativi, in particolare per le applicazioni che richiedono la gestione di dati ad alta dimensione e l'esecuzione di ricerche di similarità. Innanzitutto, l'implementazione basata sul cloud offre scalabilità, consentendo una facile regolazione delle risorse in base ai volumi di dati in crescita e ai carichi di query. In questo modo, il database può gestire in modo efficiente l'aumento della domanda mantenendo al contempo prestazioni elevate. In secondo luogo, l'implementazione del cloud offre high Availability e disaster recovery, poiché i dati possono essere replicati in diverse posizioni geografiche, riducendo al minimo il rischio di perdita di dati e garantendo un servizio continuo anche in eventi imprevisti. In terzo luogo, garantisce convenienza, poiché si paga solo per le risorse utilizzate e si può scalare in verticale o in orizzontale in base alla domanda, evitando la necessità di investimenti anticipati sostanziali in hardware. Infine, l'implementazione di un database vettoriale nel cloud può migliorare la collaborazione, poiché i dati possono essere accessibili e condivisi da qualsiasi luogo, facilitando il lavoro di gruppo e il processo decisionale basato sui dati.
Controllare l'architettura del milvus standalone con Amazon FSxN per NetApp ONTAP utilizzato in questa convalida.



1. Creare un'istanza di Amazon FSxN for NetApp ONTAP e annotare i dettagli di VPC, gruppi di sicurezza VPC e subnet. Queste informazioni saranno necessarie quando si crea un'istanza EC2. Puoi trovare ulteriori dettagli qui - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. Creare un'istanza EC2, verificando che VPC, gruppi di sicurezza e subnet corrispondano a quelli dell'istanza Amazon FSxN per NetApp ONTAP.
3. Installare nfs-common usando il comando 'apt-get install nfs-common' e aggiornare le informazioni sui pacchetti usando 'sudo apt-get update'.
4. Creare una cartella di montaggio e montare su di essa Amazon FSxN per NetApp ONTAP.

```
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. Installare Docker e Docker Compose utilizzando "apt-get install".
6. Impostare un cluster Milvus basato sul file docker-compose.yaml, che può essere scaricato dal sito Web di Milvus.

```
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. Nella sezione 'volumi' del file docker-compone.yml, mappare il punto di montaggio NFS NetApp al percorso del contenitore Milvus corrispondente, in particolare in etcd, minio e standalone.Check "Appendice D: docker-compose.yml" per i dettagli sulle modifiche in yml
8. Verificare le cartelle e i file montati.

```
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. Eseguire 'docker-compite up -d' dalla directory contenente il file docker-compite.yml.
10. Controllare lo stato del contenitore Milvus.

```
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. Per convalidare la funzionalità di lettura e scrittura del database vettoriale e dei relativi dati in Amazon FSxN per NetApp ONTAP, abbiamo utilizzato l'SDK Python Milvus e un programma di esempio di PyMilvus. Installare i pacchetti necessari usando 'apt-get install python3-numpy python3-pip' e installare PyMilvus usando 'pip3 install pymilvus'.
12. Convalidare le operazioni di scrittura e lettura dei dati da Amazon FSxN per NetApp ONTAP nel database

vettoriale.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3
prepare_data_netapp_new.py
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===
Does collection hello_milvus_ntapnew_sc exist in Milvus: True
=== Drop collection - hello_milvus_ntapnew_sc ===
=== Drop collection - hello_milvus_ntapnew_sc2 ===
=== Create collection `hello_milvus_ntapnew_sc` ===
=== Start inserting entities       ===
Number of entities in hello_milvus_ntapnew_sc: 9000
root@ip-172-31-29-98:~/pymilvus/examples# find
/home/ubuntu/milvusvectordb/
…
<removed content to save page space >
…
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/0
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/100
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/100/448789845
```

```
91411920
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/100/4487898457
91411920/xl.meta
```

13. Controllare l'operazione di lettura utilizzando lo script 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. Se il cliente desidera accedere (leggere) ai dati NFS testati nel database vettoriale tramite il protocollo S3 per i carichi di lavoro ai, questo può essere convalidato tramite un semplice programma Python. Un esempio potrebbe essere la ricerca di similarità di immagini da un'altra applicazione, come indicato nella figura che si trova all'inizio di questa sezione.

```
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/448789845791611912/448789845791611913/448789845791611920/0/448789845791411917/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#
```

Questa sezione dimostra in modo efficace come i clienti possono implementare e utilizzare un setup Milvus

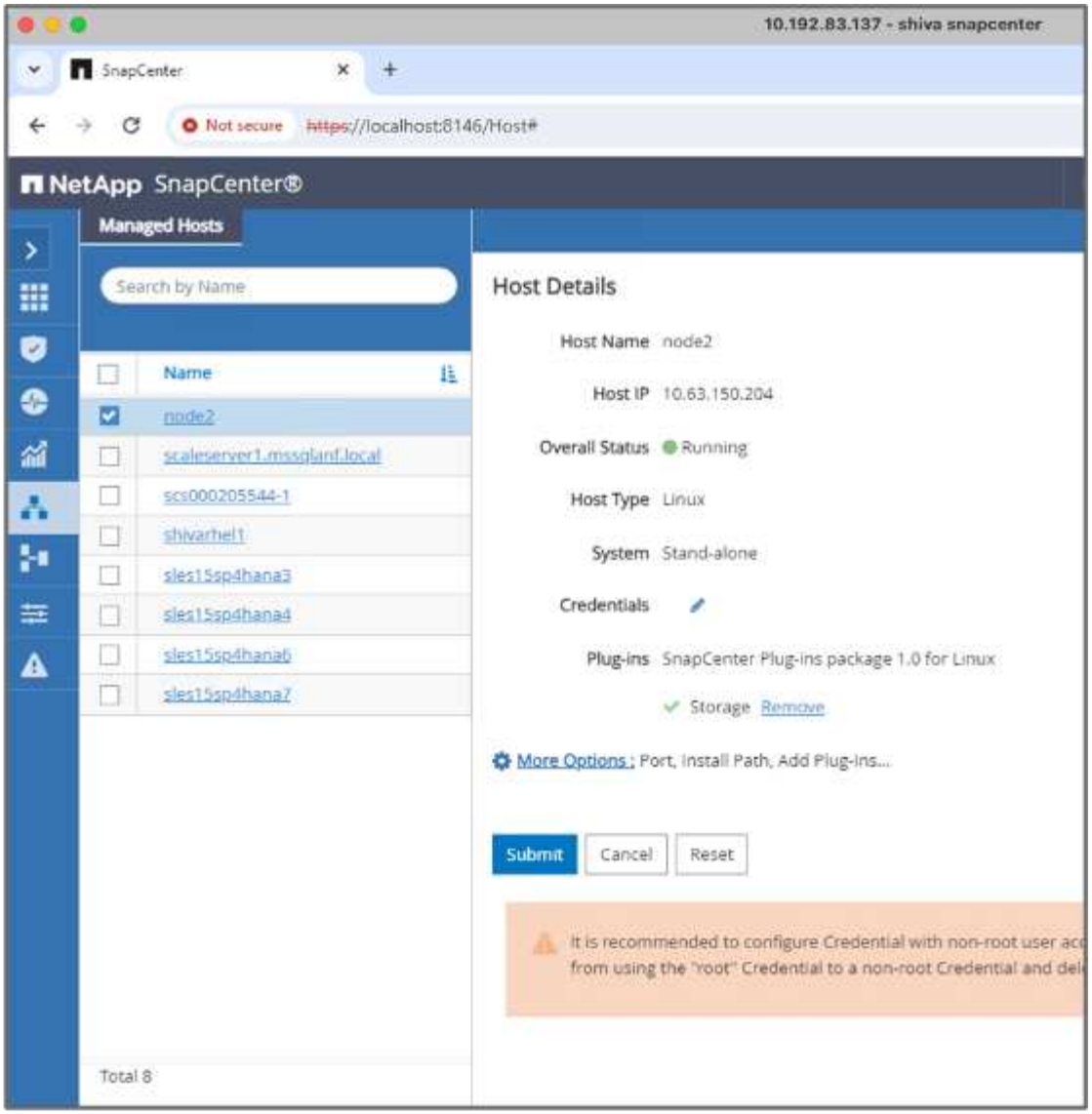
standalone all'interno dei container Docker, utilizzando la funzione FSxN di Amazon NetApp per lo storage dei dati NetApp ONTAP. Il setup consente ai clienti di sfruttare la potenza dei database vettoriali per la gestione dei dati ad alta dimensione e l'esecuzione di query complesse, il tutto all'interno dell'ambiente scalabile ed efficiente dei container Docker. Creando un'istanza di Amazon FSxN per NetApp ONTAP e abbinando l'istanza di EC2, i clienti possono garantire un utilizzo ottimale delle risorse e la gestione dei dati. La convalida riuscita delle operazioni di scrittura e lettura dei dati da parte di FSxN nel database vettoriale fornisce ai clienti la garanzia di operazioni di dati affidabili e coerenti. Inoltre, la possibilità di elencare (leggere) i dati dai workload ai tramite il protocollo S3 offre una migliore accessibilità dei dati. Questo processo completo, pertanto, fornisce ai clienti una soluzione solida ed efficiente per gestire le loro operazioni di dati su larga scala, sfruttando le capacità di FSxN di Amazon per NetApp ONTAP.

Protezione database vettoriale con SnapCenter

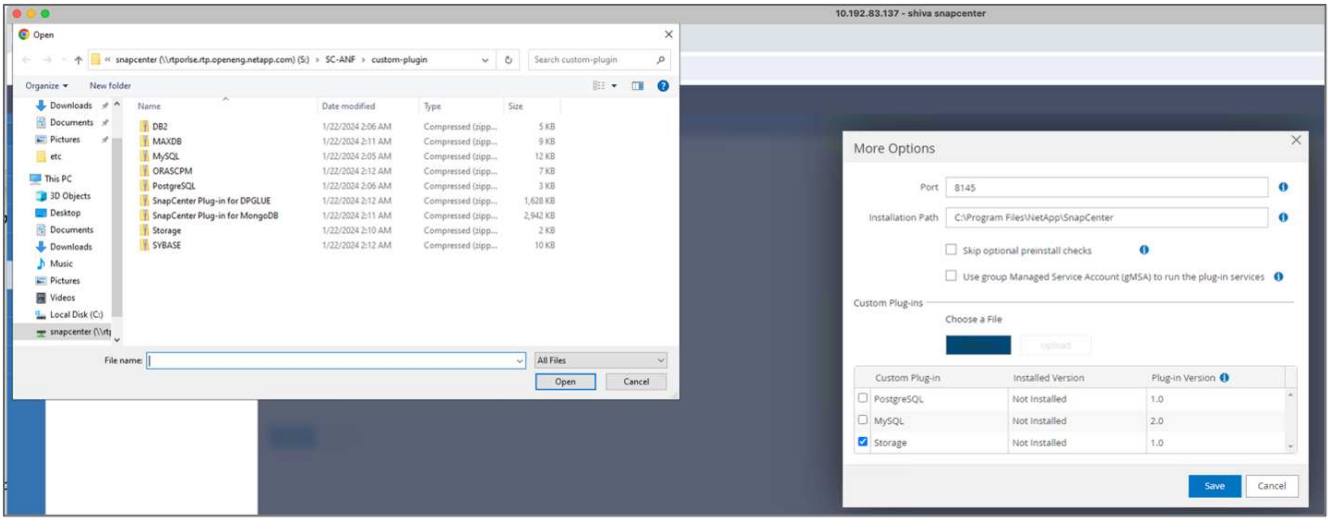
Protezione del database vettoriale tramite NetApp SnapCenter.

Ad esempio, nel settore della produzione cinematografica, i clienti spesso possiedono dati integrati critici come file video e audio. La perdita di questi dati, a causa di problemi come i guasti del disco rigido, può avere un impatto significativo sulle loro operazioni, potenzialmente compromettendo le iniziative multimilionarie. Abbiamo riscontrato casi in cui si sono persi contenuti di valore inestimabile, causando notevoli interruzioni e perdite finanziarie. Garantire la sicurezza e l'integrità di questi dati essenziali è pertanto di fondamentale importanza in questo settore.
In questa sezione, analizzeremo come SnapCenter protegge i dati del database vettoriale e i dati Milvus che risiedono in ONTAP. Per questo esempio, abbiamo utilizzato un bucket NAS (milvusdbvol1) derivato da un volume NFS ONTAP (vol1) per i dati dei clienti e un volume NFS separato (vectordbpv) per i dati di configurazione del cluster Milvus. controllare "qui" per flusso di lavoro del backup SnapCenter

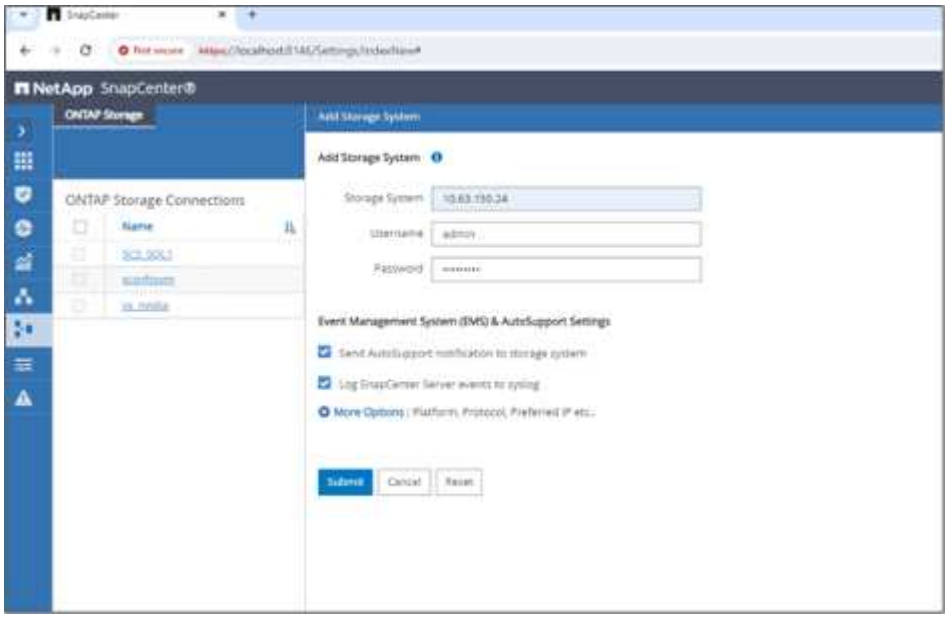
1. Impostare l'host che verrà utilizzato per eseguire i comandi SnapCenter.



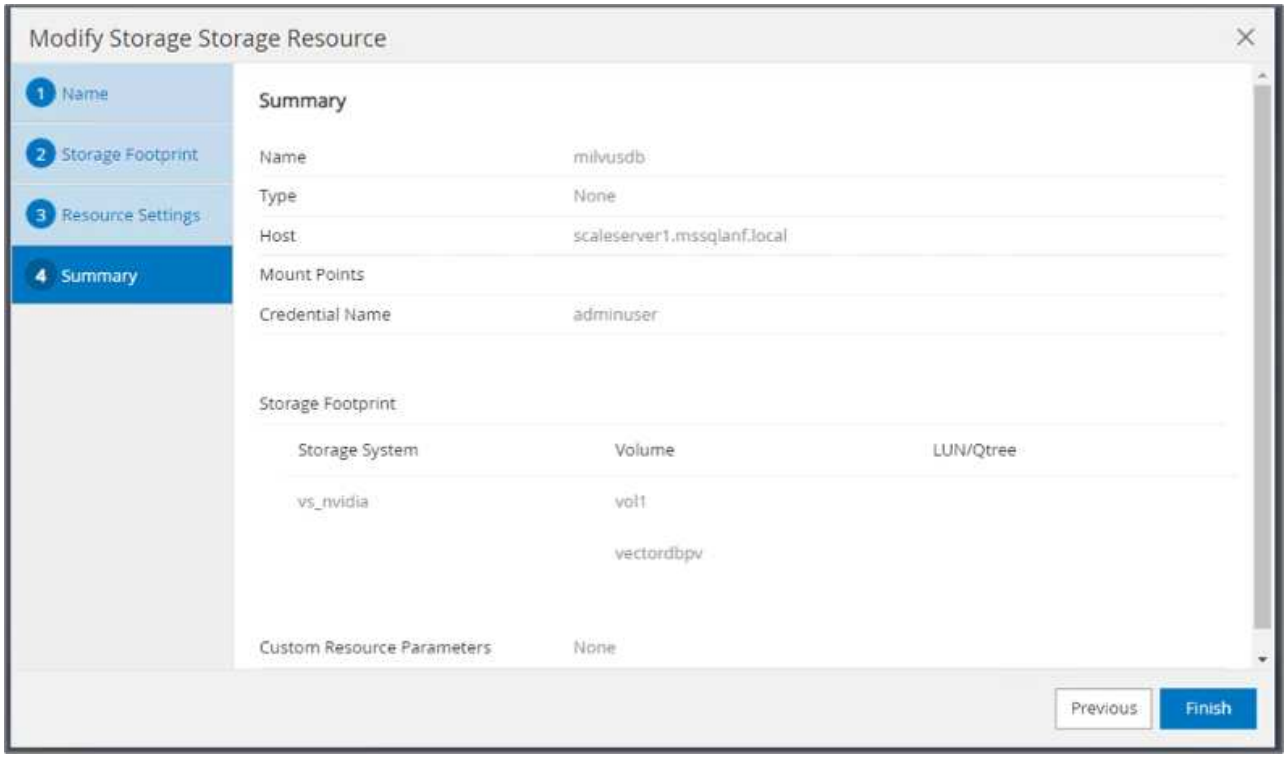
2. Installare e configurare il plug-in di archiviazione. Dall'host aggiunto, selezionare "altre opzioni". Individuare e selezionare il plug-in di archiviazione scaricato dal "NetApp Automation Store". Installare il plugin e salvare la configurazione.



3. Configura il sistema storage e il volume: Aggiungi il sistema storage sotto "Storage System" (sistema storage) e seleziona l'SVM (Storage Virtual Machine). In questo esempio, abbiamo scelto "vs_nvidia".



4. Stabilire una risorsa per il database vettoriale, incorporando un criterio di backup e un nome di snapshot personalizzato.
   - Abilitare il backup del gruppo di coerenza con i valori predefiniti e abilitare SnapCenter senza coerenza del file system.
   - Nella sezione Storage Footprint, selezionare i volumi associati ai dati cliente del database vettoriale e ai dati del cluster Milvus. Nel nostro esempio, questi sono "vol1" e "vectordbpv".
   - Creare un criterio per la protezione del database vettoriale e proteggere la risorsa del database vettoriale utilizzando il criterio.



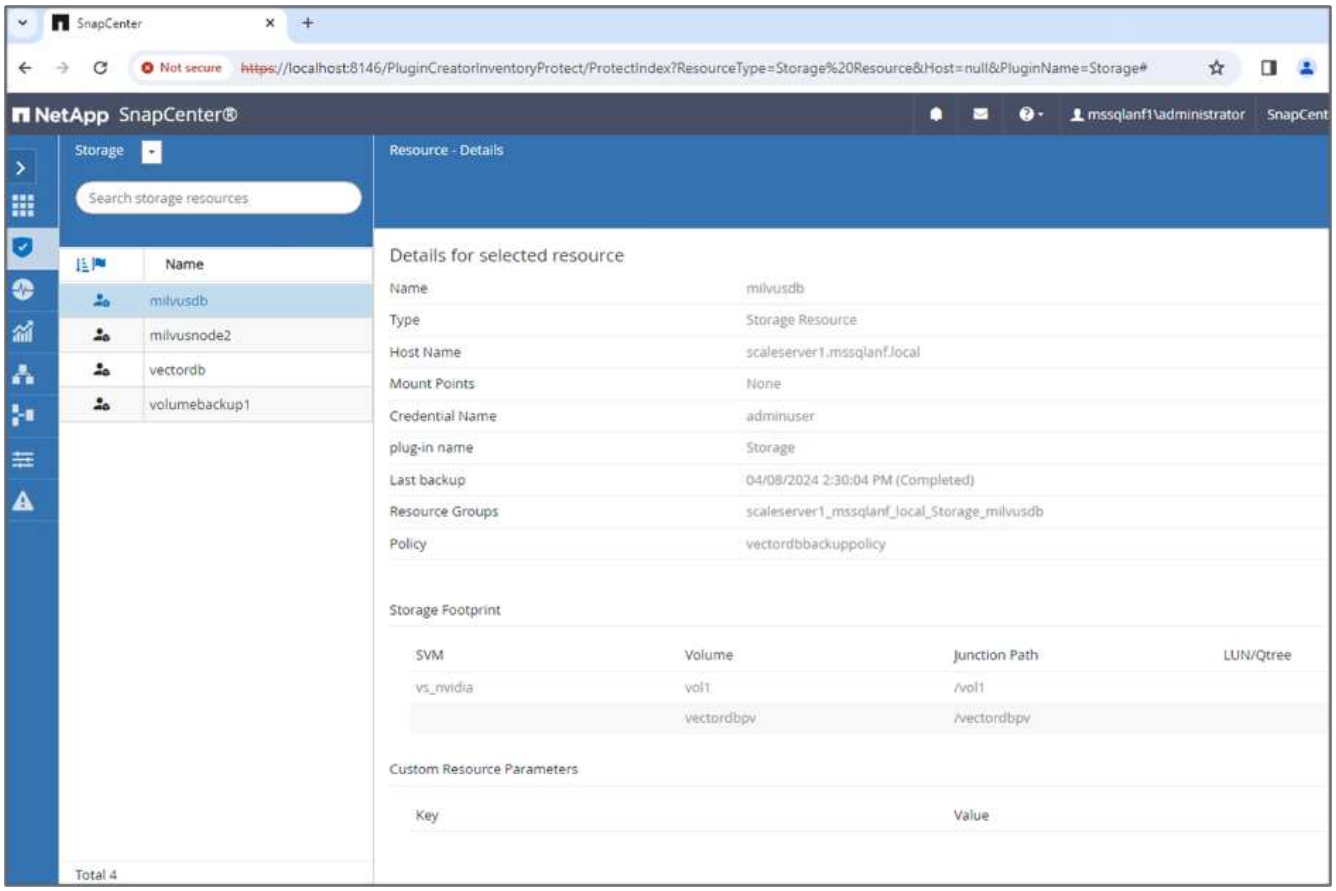
5. Inserire i dati nel bucket S3 NAS utilizzando uno script Python. Nel nostro caso, abbiamo modificato lo script di backup fornito da Milvus, ovvero 'prepare_data_netapp.py', ed eseguito il comando 'sync' per eliminare i dati dal sistema operativo.

```
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. Verifica dei dati nel bucket S3 NAS. Nel nostro esempio, i file con la data e l'ora '2024-04-08 21:22' sono stati creati dallo script 'Prepare_data_netapp.py'.

```
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. Avviare un backup utilizzando lo snapshot del gruppo di coerenza (CG) dalla risorsa 'milvusdb'

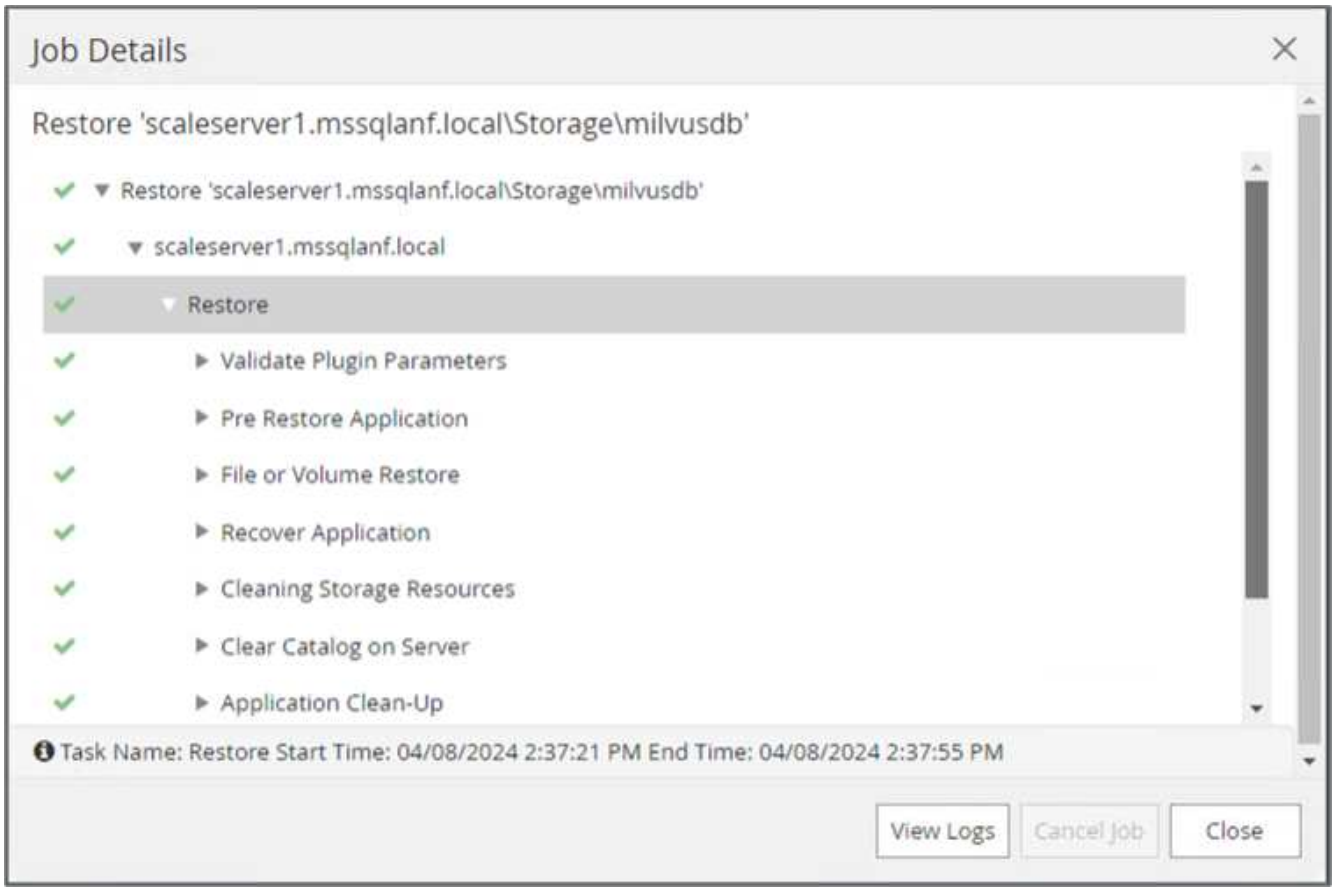


8. Per verificare la funzionalità di backup, abbiamo aggiunto una nuova tabella dopo il processo di backup o rimosso alcuni dati dal NFS (bucket S3 NAS).

   Per questo test, immaginate uno scenario in cui qualcuno ha creato una nuova raccolta non necessaria o non appropriata dopo il backup. In tal caso, sarebbe necessario riportare il database vettoriale al suo stato prima di aggiungere la nuova raccolta. Ad esempio, sono state inserite nuove raccolte come 'hello_milvus_netapp_sc_testnew' e 'hello_milvus_netapp_sc_testnew2'.

```
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. Eseguire un ripristino completo del bucket S3 NAS dalla snapshot precedente.



10. Utilizzare uno script Python per verificare i dati dalle raccolte "hello_milvus_netapp_sc_test" e "hello_milvus_netapp_sc_test2".

```
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. Verificare che la raccolta non necessaria o non appropriata non sia più presente nel database.

```
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:~#
```

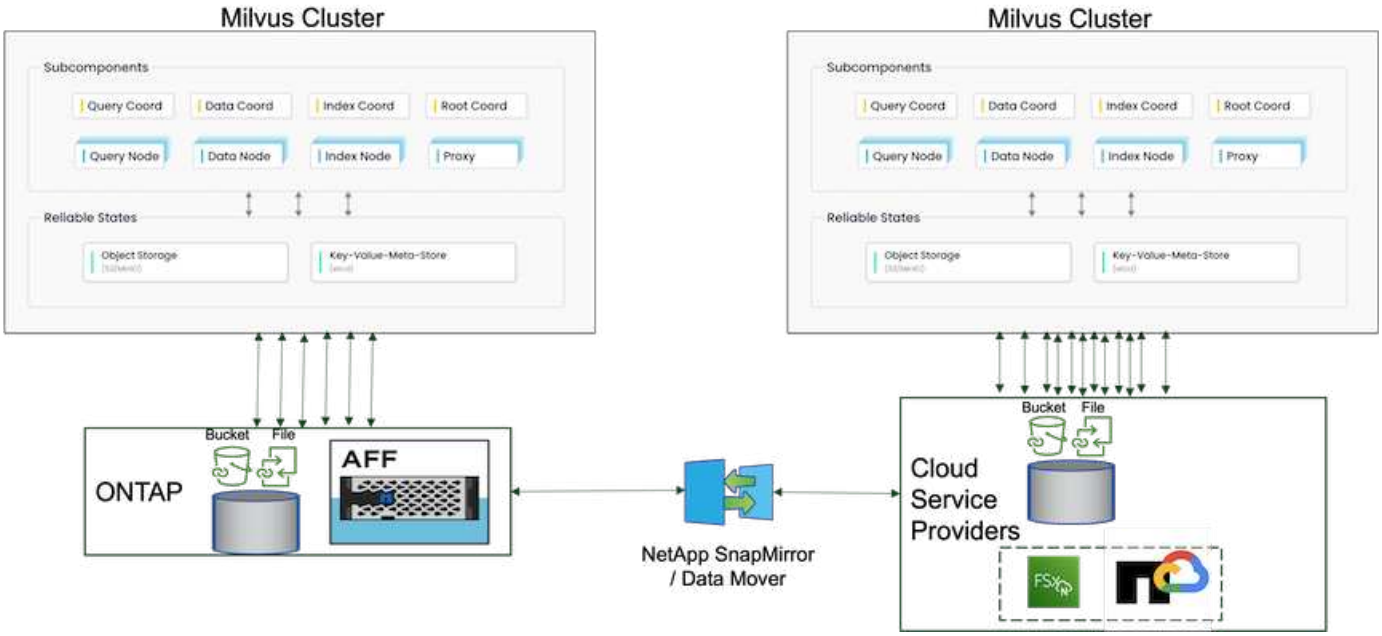
In conclusione, l'utilizzo di SnapCenter di NetApp per la salvaguardia dei dati di database vettoriali e dei dati Milvus che risiedono in ONTAP offre notevoli vantaggi ai clienti, in particolare nei settori in cui l'integrità dei dati è di primaria importanza, come la produzione cinematografica. La capacità di SnapCenter di creare backup coerenti e di eseguire ripristini completi dei dati garantisce che i dati critici, come file audio e video integrati, siano protetti dalle perdite dovute a guasti del disco rigido o ad altri problemi. Ciò non solo impedisce le perturbazioni operative, ma protegge anche da ingenti perdite finanziarie.

In questa sezione, abbiamo dimostrato come SnapCenter possa essere configurato per proteggere i dati che risiedono in ONTAP, inclusa l'installazione degli host, l'installazione e la configurazione dei plug-in di storage e la creazione di una risorsa per il database vettoriale con un nome snapshot personalizzato. Abbiamo inoltre illustrato come eseguire un backup utilizzando la snapshot del gruppo di coerenza e verificare i dati nel bucket NAS S3.

Inoltre, abbiamo simulato uno scenario in cui è stata creata una raccolta non necessaria o inadeguata dopo il backup. In tali casi, la capacità di SnapCenter di eseguire un ripristino completo da una snapshot precedente garantisce che il database vettoriale possa essere riportato al suo stato prima dell'aggiunta della nuova raccolta, mantenendo così l'integrità del database. Questa funzionalità di ripristino dei dati in uno specifico istante temporale è un valore inestimabile per i clienti, con la certezza che i dati non solo sono al sicuro, ma anche mantenuti in maniera corretta. Pertanto, il prodotto SnapCenter di NetApp offre ai clienti una soluzione solida e affidabile per la protezione e la gestione dei dati.

Disaster recovery con NetApp SnapMirror

Disaster recovery con NetApp SnapMirror



Il disaster recovery è fondamentale per mantenere l'integrità e la disponibilità di un database vettoriale, soprattutto in considerazione del suo ruolo nella gestione di dati ad alta dimensione e nell'esecuzione di complesse ricerche di similarità. Una strategia di disaster recovery ben pianificata e implementata garantisce che i dati non vadano persi o compromessi in caso di incidenti imprevisti, come guasti all'hardware, disastri naturali o attacchi informatici. Ciò è particolarmente significativo per le applicazioni che si basano su database vettoriali, dove la perdita o il danneggiamento dei dati potrebbe portare a significative interruzioni operative e perdite finanziarie. Inoltre, un solido piano di disaster recovery garantisce anche la continuità aziendale riducendo al minimo i tempi di inattività e consentendo il rapido ripristino dei servizi. Ciò è possibile grazie al prodotto di replica dei dati NetApp SnapMirroror in diverse posizioni geografiche, backup regolari e meccanismi di failover. Pertanto, il disaster recovery non è solo una misura di protezione, ma un componente critico della gestione responsabile ed efficiente del database vettoriale.

SnapMirror di NetApp fornisce la replica dei dati da un storage controller NetApp ONTAP all'altro, utilizzato principalmente per il disaster recovery (DR) e soluzioni ibride. Nel contesto di un database vettoriale, questo tool facilita la transizione fluida dei dati tra ambienti on-premise e cloud. Questa transizione si ottiene senza richiedere conversioni di dati o refactoring delle applicazioni, migliorando di conseguenza l'efficienza e la flessibilità della gestione dei dati in più piattaforme.

La soluzione ibrida NetApp in uno scenario di database vettoriale può offrire ulteriori vantaggi:

1. Scalabilità: La soluzione di cloud ibrido NetApp offre la possibilità di scalare le risorse in base ai tuoi requisiti. Puoi utilizzare le risorse on-premise per i workload normali e prevedibili e le risorse cloud come Amazon FSxN per NetApp ONTAP e Google Cloud NetApp Volume (GCNV) per i momenti di picco o i carichi imprevisti.
2. Efficienza dei costi: Il modello di cloud ibrido di NetApp ti consente di ottimizzare i costi utilizzando le risorse on-premise per carichi di lavoro regolari e pagando solo per le risorse cloud quando ne hai bisogno. Questo modello pay-as-you-go può risultare piuttosto conveniente con un'offerta di servizio NetApp instaclstr. Per i principali cloud service provider on-premise e quelli on-premise, instaclustr offre supporto e consulenza.
3. Flessibilità: Il cloud ibrido di NetApp ti offre la flessibilità di scegliere dove trattare i tuoi dati. Ad esempio, potresti scegliere di eseguire operazioni vettoriali complesse on-premise, dove disponi di hardware più potente e operazioni meno intensive nel cloud.
4. Continuità del business: In caso di disastro, la disponibilità dei dati in un cloud ibrido NetApp può garantire la continuità del business. In caso di impatto sulle risorse on-premise, puoi passare rapidamente al cloud.

Possiamo sfruttare NetApp SnapMirror per spostare i dati dall'ambiente on-premise al cloud e viceversa.

5. Innovazione: Le soluzioni di cloud ibrido di NetApp possono anche consentire un'innovazione più rapida fornendo l'accesso a tecnologie e servizi cloud all'avanguardia. Le innovazioni di NetApp nel cloud come Amazon FSxN per NetApp ONTAP, Azure NetApp Files e Google Cloud NetApp Volumes sono prodotti innovativi per i cloud service provider e NAS preferiti.

Convalida delle prestazioni del database vettoriale

Convalida delle performance

La convalida delle performance gioca un ruolo critico sia nei database vettoriali che nei sistemi di storage, fungendo da fattore chiave per garantire un funzionamento ottimale e un utilizzo efficiente delle risorse. I database vettoriali, noti per la gestione di dati ad alta dimensione e l'esecuzione di ricerche di similarità, devono mantenere livelli di prestazioni elevati per elaborare query complesse in modo rapido e accurato. La convalida delle performance aiuta a identificare i colli di bottiglia, ottimizzare le configurazioni e garantire che il sistema possa gestire i carichi previsti senza peggiorare il servizio. Analogamente, nei sistemi storage, la convalida delle performance è essenziale per garantire che i dati vengano archiviati e recuperati in modo efficiente, senza problemi di latenza o colli di bottiglia che potrebbero influire sulle performance di sistema complessive. Contribuisce inoltre a prendere decisioni informate in merito ai necessari aggiornamenti o modifiche dell'infrastruttura storage. Pertanto, la convalida delle prestazioni è un aspetto cruciale della gestione del sistema, che contribuisce in modo significativo a mantenere l'elevata qualità del servizio, l'efficienza operativa e l'affidabilità complessiva del sistema.

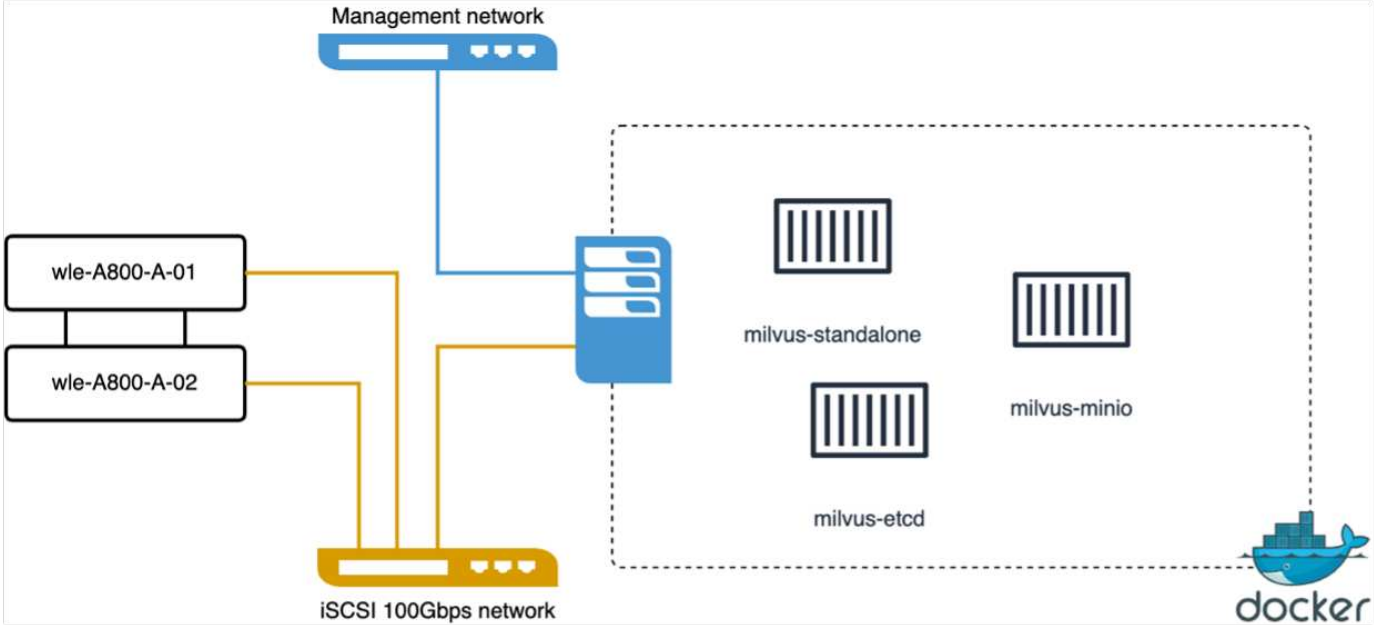
In questa sezione, il nostro obiettivo è analizzare la convalida delle prestazioni dei database vettoriali, come Milvus e pgvecto.rs, concentrandoci sulle caratteristiche delle prestazioni di storage, come il profilo i/o e il comportamento dello storage controller NetApp a supporto di RAG e carichi di lavoro di inferenza all'interno del ciclo di vita LLM. Valuteremo e identificheremo eventuali elementi di differenziazione delle performance quando questi database sono combinati con la soluzione di storage ONTAP. La nostra analisi si baserà su indicatori chiave delle prestazioni, ad esempio il numero di query elaborate al secondo (QPS).

Controllare la metodologia utilizzata per il milvus e procedere di seguito.

Dettagli	Milvus (standalone e cluster)	Postgres(pgvecto.rs)
versione	2.3.2	0.2.0
Filesystem	XFS su LUN iSCSI	
Workload generator	"Banco VectorDB" – v0,0.5	
Set di dati	Set di dati LAION * 10Million incisioni * 768 dimensioni * Dimensione set di dati ~300GB	

VectorDB-Bench con cluster standalone Milvus

Abbiamo eseguito la seguente convalida delle performance su cluster milvus standalone con vectorDB-Bench. La connettività di rete e server del cluster milvus standalone è riportata di seguito.



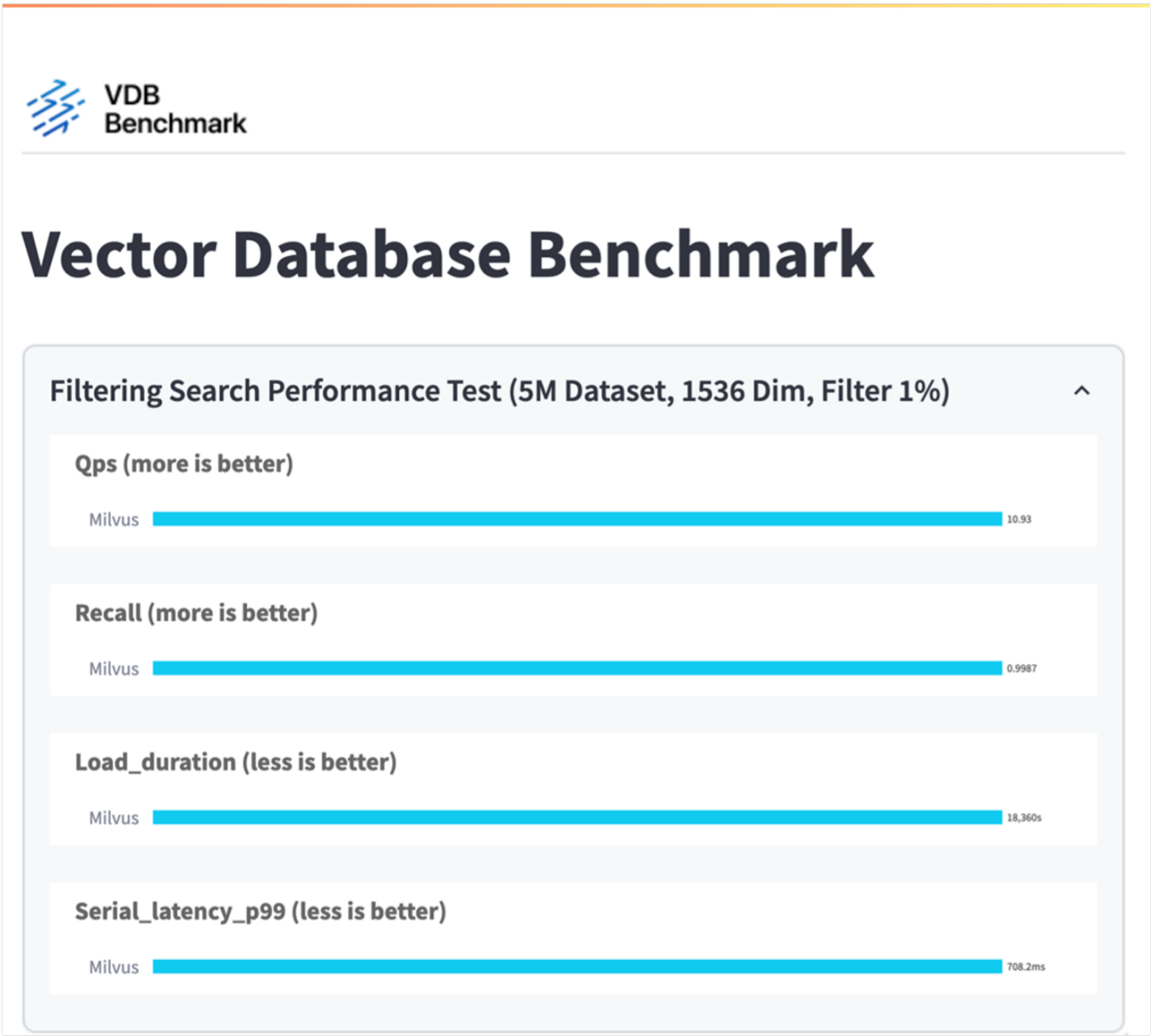
In questa sezione, condividiamo le nostre osservazioni e i risultati dei test del database autonomo Milvus.
. Per questi test è stato selezionato DiskANN come tipo di indice.
. L'acquisizione, l'ottimizzazione e la creazione di indici per un set di dati di circa 100GB TB ha richiesto circa 5 ore. Per la maggior parte di questa durata, il server Milvus, dotato di 20 core (che equivale a 40 vcpus quando Hyper-Threading è abilitato), funzionava alla sua capacità massima della CPU del 100%.abbiamo scoperto che DiskANN è particolarmente importante per grandi set di dati che superano le dimensioni della memoria del sistema.
. Nella fase di query, abbiamo osservato un tasso di query al secondo (QPS) di 10,93 con un richiamo di 0,9987. La latenza percentile di 99th ms per le query è stata misurata a 708,2 millisecondi.

Dal punto di vista dello storage, il database ha emesso circa 1.000 op/sec durante le fasi di acquisizione, ottimizzazione post-inserimento e creazione di indice. Nella fase di query, richiedevano 32.000 op/sec.

La sezione seguente presenta le metriche relative alle performance dello storage.

Fase del carico di lavoro	Metrico	Valore
Acquisizione dei dati e. Ottimizzazione post-inserimento	IOPS	< 1.000
	Latenza	< 400 usecs
	Carico di lavoro	Combinazione di lettura e scrittura, per la maggior parte scritture
	Dimensioni io	64KB
Query	IOPS	Picco alle 32.000:00
	Latenza	< 400 usecs
	Carico di lavoro	100% lettura nella cache
	Dimensioni io	Principalmente 8KB

Il risultato del vectorDB-bench è riportato di seguito.

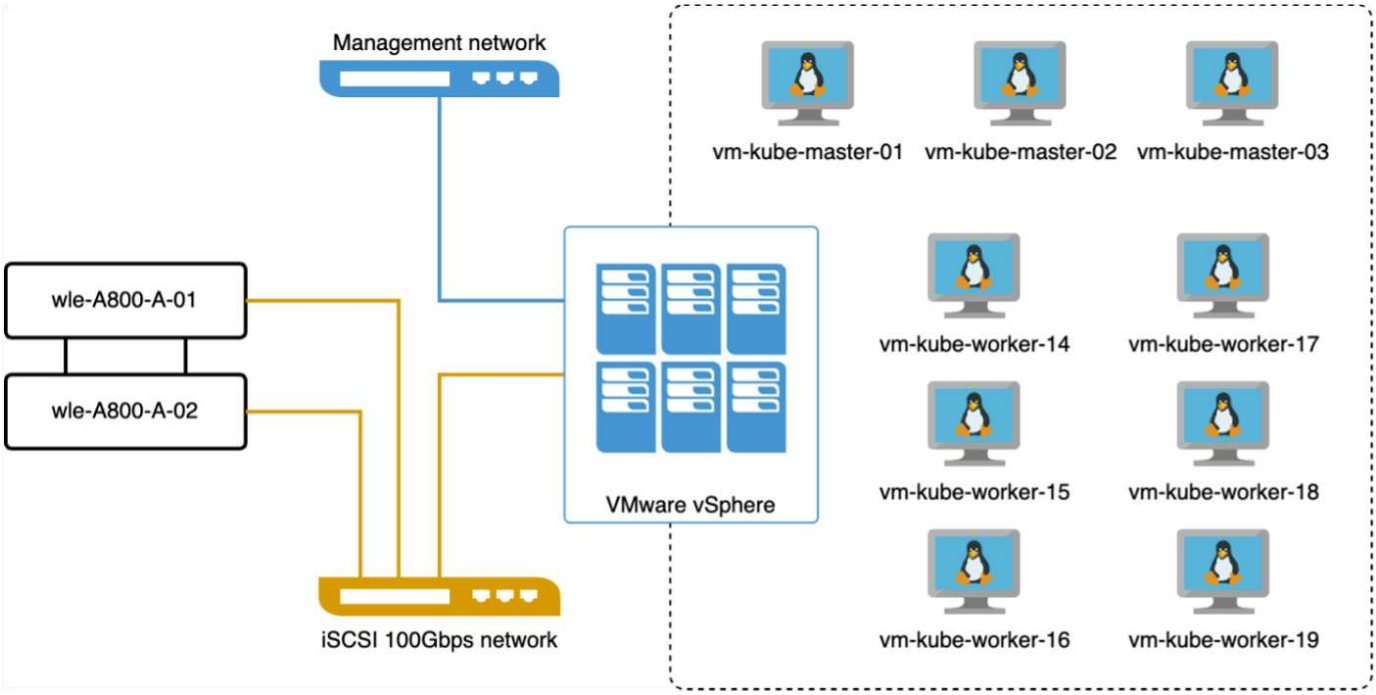


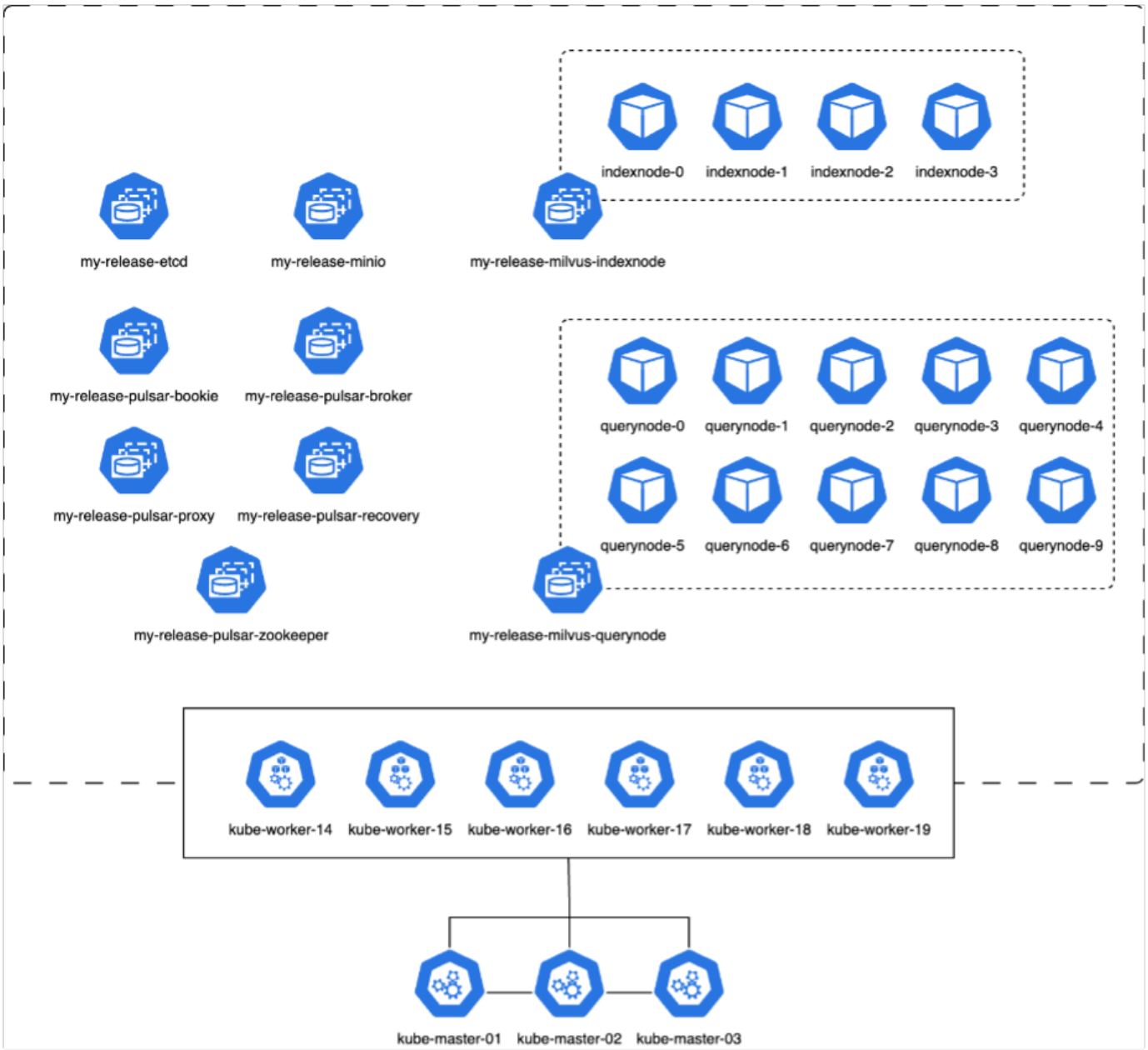
Dalla convalida delle prestazioni dell'istanza autonoma di Milvus, è evidente che l'impostazione corrente non è sufficiente a supportare un set di dati di 5 milioni di vettori con una dimensione di 1536. abbiamo determinato che lo storage dispone di risorse adeguate e non costituisce un collo di bottiglia nel sistema.

VectorDB-Bench con cluster milvus

In questa sezione, parleremo dell'implementazione di un cluster Milvus all'interno di un ambiente Kubernetes. Questo setup di Kubernetes è stato costruito in cima a un'implementazione VMware vSphere, che ospitava i nodi di lavoro e master di Kubernetes.

I dettagli delle implementazioni di VMware vSphere e Kubernetes sono presentati nelle seguenti sezioni.


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



In questa sezione, presentiamo le nostre osservazioni e i risultati dei test del database Milvus.
* Il tipo di indice utilizzato era DiskANN.
* La tabella riportata di seguito fornisce un confronto tra le distribuzioni standalone e cluster quando si utilizzano 5 milioni di vettori con una dimensione di 1536. Abbiamo osservato che il tempo richiesto per l'acquisizione dei dati e l'ottimizzazione post-inserimento è stato inferiore nell'implementazione del cluster. La latenza percentile di 99th ms per le query è stata ridotta di sei volte nell'implementazione del cluster rispetto alla configurazione standalone.
* Sebbene il tasso di query al secondo (QPS) fosse più elevato nella distribuzione del cluster, non era al livello desiderato.

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%

Le immagini seguenti forniscono una vista di varie metriche storage, inclusa la latenza del cluster storage e gli IOPS totali (operazioni di input/output al secondo).

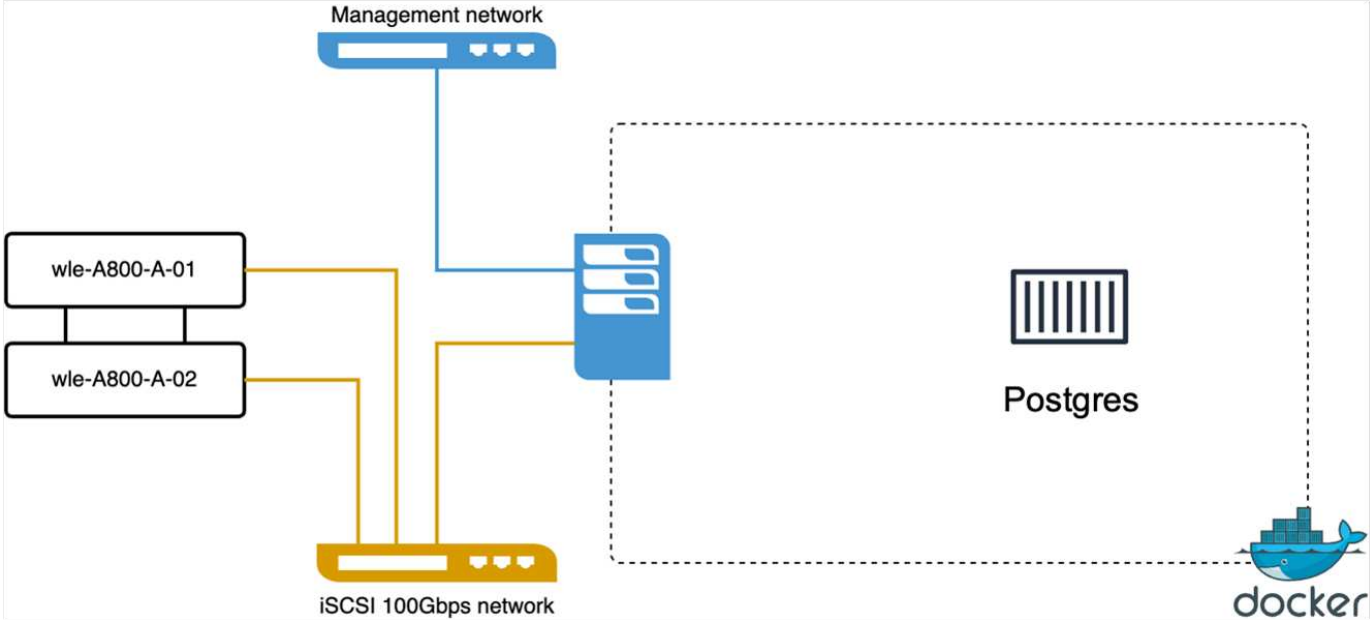
La sezione seguente presenta le metriche chiave delle performance dello storage.

Fase del carico di lavoro	Metrico	Valore
Acquisizione dei dati e. Ottimizzazione post-inserimento	IOPS	< 1.000
	Latenza	< 400 usecs
	Carico di lavoro	Combinazione di lettura e scrittura, per la maggior parte scritture
	Dimensioni io	64KB
Query	IOPS	Picco alle 147.000:00
	Latenza	< 400 usecs
	Carico di lavoro	100% lettura nella cache
	Dimensioni io	Principalmente 8KB

Presentiamo i dettagli del profilo i/o dello storage in base alla convalida delle performance sia dello storage Milvus che del cluster Milvus.
* Abbiamo osservato che il profilo di i/o rimane coerente sia nelle implementazioni standalone che in cluster.
* La differenza osservata negli IOPS di picco può essere attribuita al maggior numero di client nell'implementazione del cluster.

VettorDB-Bench con Postgres (pgvecto.rs)

Abbiamo eseguito le seguenti azioni su PostgreSQL(pgvecto.rs) utilizzando VectorDB-Bench:
I dettagli relativi alla connettività di rete e server di PostgreSQL (in particolare pgvecto.rs) sono i seguenti:

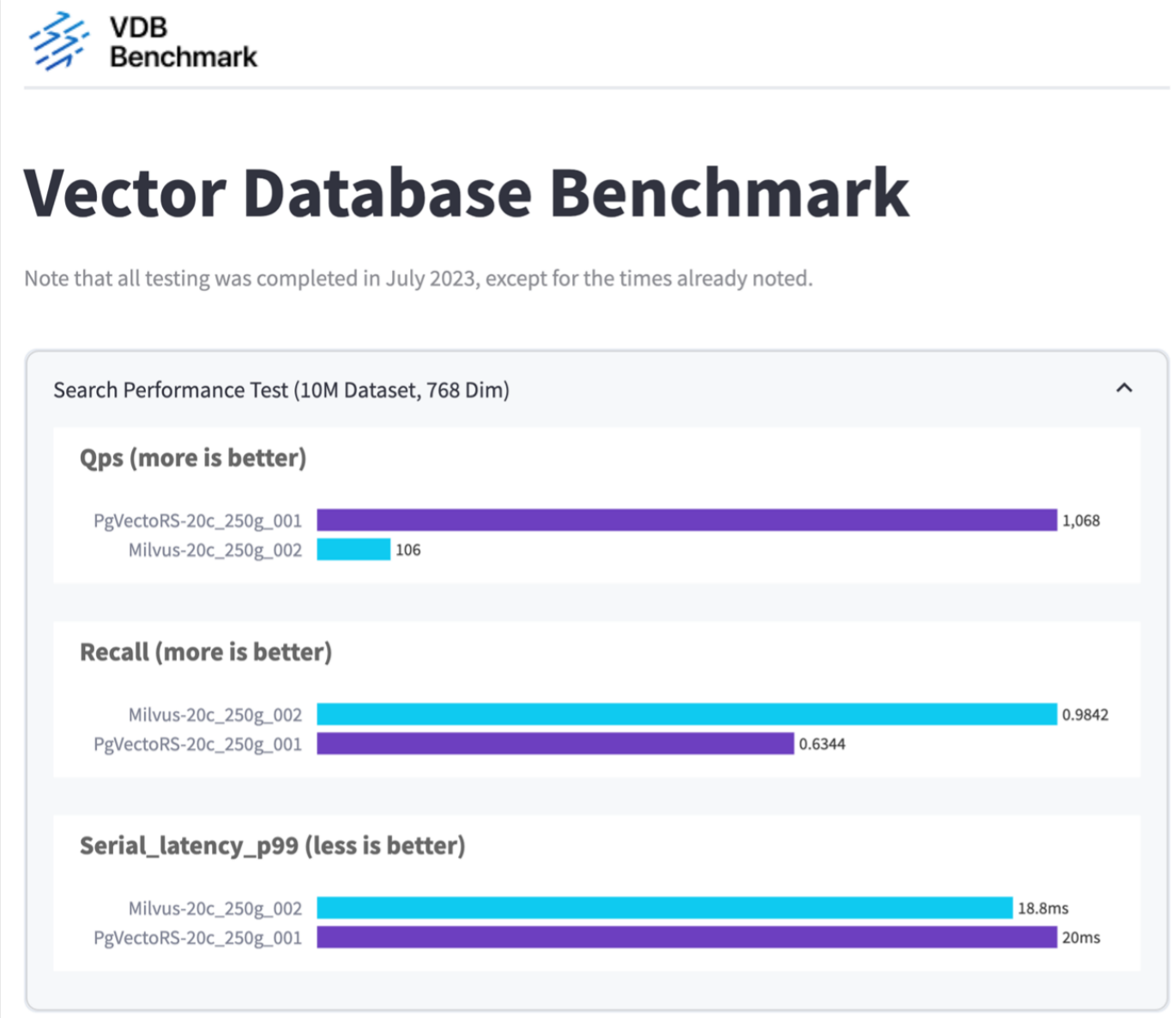


In questa sezione, condividiamo le nostre osservazioni e i risultati dei test del database PostgreSQL, in particolare utilizzando pgvecto.rs.
* Abbiamo selezionato HNSW come tipo di indice per questi test perché al momento del test, DiskANN non era disponibile per pgvecto.rs.
* Durante la fase di acquisizione dei dati, è stato caricato il set di dati Cohere, che consiste di 10 milioni di vettori con una dimensione di 768. Questo processo ha richiesto circa 4,5 ore.
* Nella fase di query, abbiamo osservato un tasso di query al secondo (QPS) di 1.068 con un richiamo di 0,6344. La latenza percentile di 99th ms per le query è stata misurata a 20 millisecondi. Per la maggior parte del runtime, la CPU del client funzionava al 100% della capacità.

Le immagini qui sotto forniscono una vista di varie metriche storage, inclusi gli IOPS totali della latenza del cluster storage (operazioni di input/output al secondo).


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["Errore: Immagine grafica
mancante"]
```

Confronto delle prestazioni tra milvus e postgres su DB Bench vettoriale



In base alla nostra convalida delle prestazioni di Milvus e PostgreSQL utilizzando VectorDBBench, abbiamo osservato quanto segue:

- Tipo di indice: HNSW
- Set di dati: Coqui con 10 milioni di vettori a 768 dimensioni

Abbiamo scoperto che pgvecto.rs ha raggiunto un tasso di query al secondo (QPS) di 1.068 con un richiamo di 0,6344, mentre Milvus ha raggiunto un tasso di QPS di 106 con un richiamo di 0,9842.

Se l'elevata precisione nelle query è una priorità, Milvus supera pgvecto.rs poiché recupera una proporzione maggiore di elementi rilevanti per ogni query. Tuttavia, se il numero di query al secondo è un fattore più cruciale, pgvecto.rs supera Milvus. È importante notare, comunque, che la qualità dei dati recuperati tramite pgvecto.rs è inferiore, con circa il 37% dei risultati di ricerca che sono elementi irrilevanti.

Osservazione basata sulle nostre convalide delle prestazioni:

Sulla base delle nostre convalide delle prestazioni, abbiamo fatto le seguenti osservazioni:

Il profilo di i/o di Milvus assomiglia molto a un carico di lavoro OLTP, come quello offerto dagli SLO Oracle. Il

benchmark è composto da tre fasi: Inserimento dei dati, Post-ottimizzazione e query. Gli stadi iniziali sono caratterizzati principalmente da operazioni di scrittura 64KB, mentre la fase di query riguarda prevalentemente operazioni di lettura 8KB. Ci aspettiamo che ONTAP gestisca con competenza il carico i/o Milvus.

Il profilo i/o di PostgreSQL non presenta carichi di lavoro complessi per lo storage. Data l'implementazione in memoria attualmente in corso, durante la fase di query non è stato rilevato alcun i/o del disco.

DiskANN emerge come una tecnologia cruciale per la differenziazione dello storage. Consente di scalare in modo efficiente la ricerca DB vettoriale oltre il limite della memoria di sistema. Tuttavia, è improbabile che stabilisca la differenziazione delle prestazioni di storage con indici DB vettoriali in memoria come HNSW.

È inoltre opportuno notare che l'archiviazione non svolge un ruolo critico durante la fase di query quando il tipo di indice è HSNW, che è la fase operativa più importante per i database vettoriali che supportano le applicazioni RAG. In questo caso, l'implicazione è che le performance dello storage non hanno un impatto significativo sulle performance complessive di queste applicazioni.