解決方案驗證總覽
NetApp Solutions

NetApp
May 03, 2024

目錄

解決方案總覽

我們已針對五個關鍵領域進行全面的解決方案驗證、詳細內容概述如下。每個部分都會深入探討客戶所面臨的挑戰、 NetApp 提供的解決方案、以及後續對客戶的好處。

1. "在內部部署使用 Kubernetes 進行 Milvus 叢集設定"
   客戶在儲存與運算、有效的基礎架構管理與資料管理上、必須自行擴充規模、這是一項挑戰。在本節中、我們將詳細說明在 Kubernetes 上安裝 Milvus 叢集的程序、並使用 NetApp 儲存控制器來處理叢集資料和客戶資料。
2. "Milvus 搭配 Amazon FSxN for NetApp ONTAP –檔案和物件雙重性"
   在本節中、為什麼我們需要在雲端中部署向量資料庫、以及在 Docker 容器內的 Amazon FSxN for NetApp ONTAP 中部署向量資料庫（ milvus 獨立式）的步驟。
3. "使用 NetApp SnapCenter 保護向量資料庫。"
   在本節中、我們將深入探討 SnapCenter 如何保護 ONTAP 中的向量資料庫資料和 Milvus 資料。在此範例中、我們將從 NFS ONTAP Volume （ vol1 ）衍生的 NAS 儲存區（ milvusdbvol1 ）用於客戶資料、並將獨立的 NFS Volume （ vectordbpv ）用於 Milvus 叢集組態資料。
4. "使用 NetApp SnapMirror 進行災難恢復"
   在本節中、我們將討論災難恢復（ DR ）對於向量資料庫的重要性、以及 NetApp 災難恢復產品 SnapMirror 如何為向量資料庫提供災難恢復解決方案。
5. "效能驗證"
   在本節中、我們的目標是深入探討向量資料庫（例如 Milvus 和 pgveco.RS ）的效能驗證、重點在於其儲存效能特性、例如 I/O 設定檔和 NetApp 儲存控制器行為、以支援 LLM 生命週期內的 RAG 和推斷工作負載。當這些資料庫與 ONTAP 儲存解決方案結合使用時、我們會評估並找出任何效能差異。我們的分析將以關鍵效能指標為基礎、例如每秒處理的查詢數（ QPS ）。

在內部部署使用 **Kubernetes** 的 **Milvus** 叢集設定

在內部部署使用 **Kubernetes** 進行 **Milvus** 叢集設定

客戶在儲存與運算上的擴充、有效的基礎架構管理與資料管理、
Kubernetes 和向量資料庫一起形成強大且可擴充的解決方案、可用於管理大型資料作業。Kubernetes 可最佳化資源並管理容器、而向量資料庫則可有效處理高維度資料和相似度搜尋。這項組合可快速處理大型資料集的複雜查詢、並可隨著不斷成長的資料量順暢擴充、因此非常適合巨量資料應用程式和 AI 工作負載。

1. 在本節中、我們將詳細說明在 Kubernetes 上安裝 Milvus 叢集的程序、並使用 NetApp 儲存控制器來處理叢集資料和客戶資料。
2. 若要安裝 Milvus 叢集、儲存來自各種 Milvus 叢集元件的資料時、需要持續磁碟區（ PV ）。這些元件包括 etcd （三個執行個體）、 Pulsar-bootike-journal （三個執行個體）、 Pulsar-bootike-ledgers （三個執行個體）和 Pulsar-zooker-data （三個執行個體）。

   > 在 milvus 叢集中、我們可以使用 Pulsar 或 Kafka 作為基礎引擎、以支援 Milvus 叢集可靠的儲存、以及訊息串流的發佈 / 訂閱。針對 NFS 的 Kafka 、 NetApp 已在 ONTAP 9.12.1 及更新版本中進行改善、這些增強功能以及 RHEL 8.7 或 9.1 或更新版本中所包含的 NFSv4.1 及 Linux 變更、可解決透過 NFS 執行 Kafka 時可能發生的「愚蠢重新命名」問題。如果您想深入瞭解如何使用 NetApp NFS 執行 Kafka 解決方案、請查看： https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html。

3. 我們從 NetApp ONTAP 建立了單一 NFS Volume 、並建立了 12 個持續磁碟區、每個磁碟區都有 250GB 的

儲存容量。儲存容量可能會因叢集大小而異；例如、我們有另一個叢集、其中每個 PV 都有 50GB 。請參閱下列 PV YAML 檔案之一、以取得更多詳細資料；我們總共有 12 個此類檔案。在每個檔案中、storageClassName 會設為「預設」、而儲存設備和路徑對每個 PV 都是唯一的。

```
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. 為每個 PV YAML 檔案執行「 kutbectl apply 」命令、以建立持續磁碟區、然後使用「 kutbectl Get PV 」驗證其建立

```
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. 為了儲存客戶資料、 Milvus 支援 MinIO 、 Azure Blob 和 S3 等物件儲存解決方案。在本指南中、我們使用 S3 。下列步驟同時適用於 ONTAP S3 和 StorageGRID 物件存放區。我們使用 Helm 來部署 Milvus 叢集。從 Milvus 下載位置下載組態檔案 values.yaml 。請參閱附錄以取得本文件所使用的 values.yaml 檔案。
6. 請確定每個區段的「 sorageClass 」都設為「預設」、包括記錄檔、 etcd 、 zookeeper 和 bookkeeper 的「預設類別」。
7. 在 MinIO 區段中、停用 MinIO 。
8. 從 ONTAP 或 StorageGRID 物件儲存區建立 NAS 儲存區、並將其納入具有物件儲存認證的外部 S3 。

```
###################################
# 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. 在建立 Milvus 叢集之前、請確定 PersistentVolume Claim （ PVC ）沒有任何預先存在的資源。

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

10. 使用 Helm 和 values.yaml 組態檔案來安裝和啟動 Milvus 叢集。

```
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. 驗證 PersistentVolume Claims （ PVCS ）的狀態。

```
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. 檢查 Pod 的狀態。

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

請確定 Pod 狀態為「執行中」、並正常運作

13. 在 Milvus 和 NetApp 物件儲存設備中測試資料寫入和讀取。
    - 使用「 Prepare _data_NetApp_new.py 」 Python 程式寫入資料。

```
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:~#
```

- 使用「 VERIFY_data_NetApp.py 」 Python 檔案讀取資料。

```
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}}]}
```

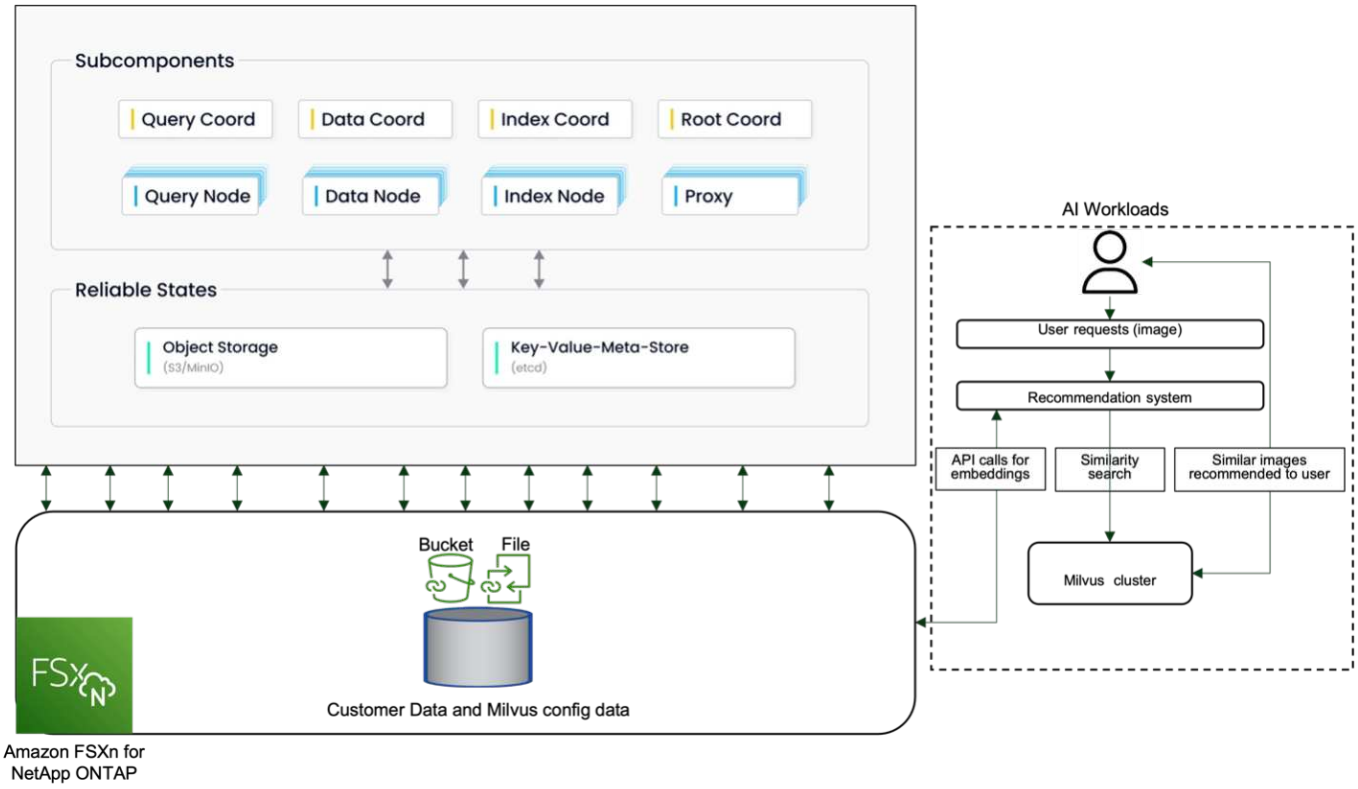
根據上述驗證、 Kubernetes 與向量資料庫的整合、透過在 Kubernetes 上使用 NetApp 儲存控制器部署 Milvus 叢集、為客戶提供強大、可擴充且有效率的解決方案、以管理大規模資料作業。這項設定可讓客戶快速有效地處理高維度資料、並執行複雜查詢、是大型資料應用程式和 AI 工作負載的理想解決方案。將持續磁碟區（ PV ）用於各種叢集元件、以及從 NetApp ONTAP 建立單一 NFS 磁碟區、可確保最佳的資源使用率和資料管理。驗證 PersistentVolume Claims （ PVCS ）和 Pod 狀態的程序、以及測試資料寫入和讀取、可讓客戶確保資料作業可靠且一致。使用 ONTAP 或 StorageGRID 物件儲存設備來儲存客戶資料、可進一步增強資料的存取能力和安全性。整體而言、這項設定可讓客戶擁有彈性且高效能的資料管理解決方案、並可隨著不斷成長的資料需求順暢地擴充。

Milvus 搭配 Amazon FSxN for NetApp ONTAP - 檔案和物件雙重性

Milvus 搭配 Amazon FSxN for NetApp ONTAP –檔案和物件雙重性

在本節中、為什麼我們需要在雲端中部署向量資料庫、以及在 Docker 容器內的 Amazon FSxN for NetApp ONTAP 中部署向量資料庫（ milvus 獨立式）的步驟。

在雲端中部署向量資料庫可提供多項重要效益、特別是對於需要處理高維度資料和執行相似度搜尋的應用程式。首先、雲端型部署提供擴充性、可輕鬆調整資源、以配合不斷成長的資料量和查詢負載。如此可確保資料庫能夠有效處理增加的需求、同時維持高效能。其次、雲端部署可提供高可用度和災難恢復、因為資料可在不同的地理位置上複寫、將資料遺失的風險降至最低、並確保即使發生非預期的事件、仍能持續提供服務。第三、它能提供成本效益、因為您只需支付所使用的資源、並可根據需求進行上下擴充、避免需要大量的硬體前期投資。最後、在雲端部署向量資料庫可加強協同作業、因為資料可以從任何地方存取和共享、有助於團隊工作和資料導向的決策。
請檢查在此驗證中使用的 Milvus 獨立式與 Amazon FSxN for NetApp ONTAP 的架構。



1. 建立 Amazon FSxN for NetApp ONTAP 執行個體、並記下 VPC 、 VPC 安全群組和子網路的詳細資料。建立 EC2 執行個體時、必須提供這項資訊。您可以在這裡找到更多詳細資料 - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. 建立 EC2 執行個體、確保 VPC 、安全性群組和子網路與 Amazon FSxN for NetApp ONTAP 執行個體的相符。
3. 使用命令 'apt-Get install NFS-common' 安裝 NFS-common' 、並使用 'Udo apt-Get update" 更新套件資訊。
4. 建立裝載資料夾、並在其中掛載 Amazon FSxN for 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. 使用「 apt-Get 安裝」安裝 Docker 和 Docker Compose 。
6. 根據泊塢視窗 -compare.yaml 檔案設定 Milvus 叢集、可從 Milvus 網站下載。

```
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. 在泊塢視窗 -compile.yml 檔案的「 Volumes 」（磁碟區）區段中、將 NetApp NFS 掛載點對應至對應的 Milvus 容器路徑、特別是 etcd 、 minio 和 standbedy.Check "附錄 D ：泊塢視窗 - 組合 .yml" 以取得有關 Yml 變更的詳細資訊
8. 驗證掛載的資料夾和檔案。

```
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. 從包含泊塢視窗 -compile.yml 檔案的目錄執行「 ocker-competup -d 」。
10. 檢查 Milvus 容器的狀態。

```
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. 為了驗證向量資料庫的讀寫功能、以及它在 Amazon FSxN for NetApp ONTAP 中的資料、我們使用 Python Milvus SDK 和 PyMilvus 的範例程式。使用 'apt-Get install python3-numpy python3-pip' 安裝必要的套件、並使用 'pip3 install pymilvus' 安裝 PyMilvus 。
12. 驗證向量資料庫中 Amazon FSxN for NetApp ONTAP 的資料寫入和讀取作業。

```
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. 使用 verify_data_netapp.py 指令碼檢查讀取作業。

```
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. 如果客戶想要存取（讀取）透過 S3 傳輸協定在向量資料庫中測試的 AI 工作負載 NFS 資料、則可以使用簡單易懂的 Python 程式來驗證。例如、如本節開頭的圖片所述、從其他應用程式搜尋影像的相似性。

```
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#
```

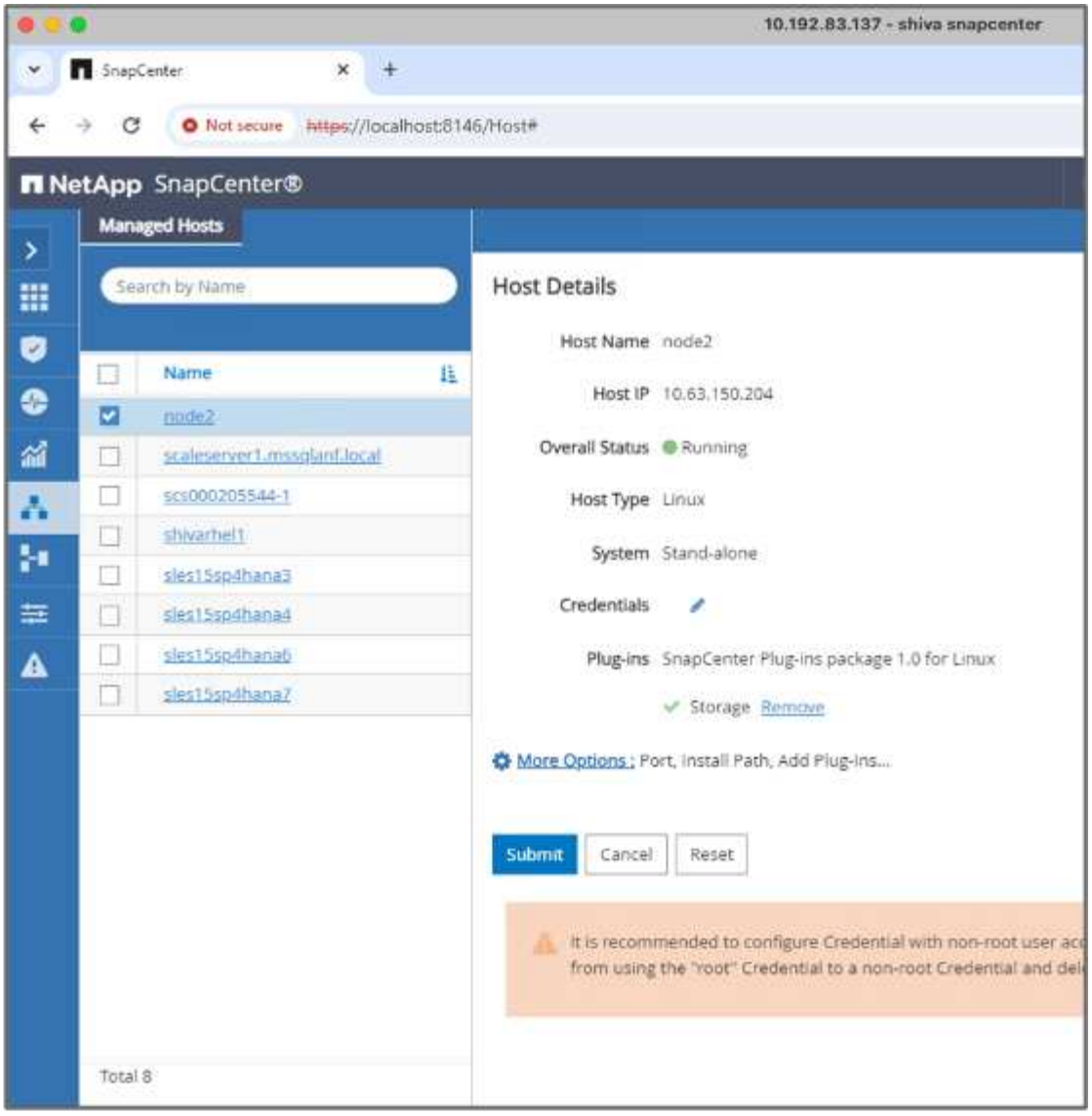
本節有效說明客戶如何在 Docker 容器中部署及操作獨立的 Milvus 設定、並運用 Amazon 的 NetApp FSxN 來儲存 NetApp ONTAP 資料。這項設定可讓客戶運用向量資料庫的強大功能、在 Docker 容器的可擴充且有效率的環境中、處理高維度資料並執行複雜的查詢。透過為 NetApp ONTAP 執行個體建立 Amazon FSxN 並搭配 EC2 執行個體、客戶可以確保最佳的資源使用率和資料管理。成功驗證向量資料庫中 FSxN 的資料寫入與讀取作業、可讓客戶確保資料作業穩定可靠。此外、透過 S3 傳輸協定列出（讀取） AI 工作負載資料的能力、可增強資料存取能力。因此、這項全方位的程序可為客戶提供強大且有效率的解決方案、讓客戶運用 Amazon FSxN for NetApp ONTAP 的功能來管理大規模資料作業。

使用 **SnapCenter** 的向量資料庫保護

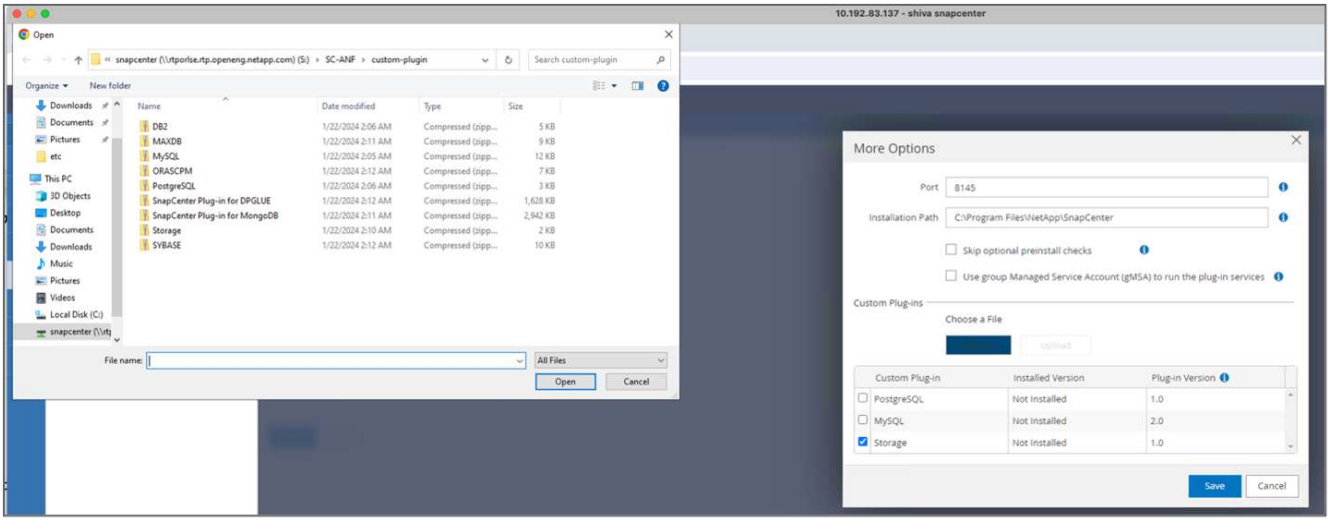
使用 **NetApp SnapCenter** 保護向量資料庫。

例如、在電影製作產業中、客戶通常擁有視訊和音訊檔案等重要的嵌入式資料。由於硬碟故障等問題而導致資料遺失、可能會對其營運造成重大影響、進而可能危及數百萬美元的風險。我們曾遇到過寶貴內容遺失的情況、導致嚴重的中斷和財務損失。因此、確保這些重要資料的安全性和完整性、在這個產業中至關重要。
在本節中、我們將深入探討 SnapCenter 如何保護 ONTAP 中的向量資料庫資料和 Milvus 資料。在此範例中、我們將從 NFS ONTAP Volume （ vol1 ）衍生的 NAS 儲存區（ milvusdbvol1 ）用於客戶資料、並將獨立的 NFS Volume （ vectordbpv ）用於 Milvus 叢集組態資料。請檢查 "請按這裡" 適用於 SnapCenter 備份工作流程

1. 設定用於執行 SnapCenter 命令的主機。

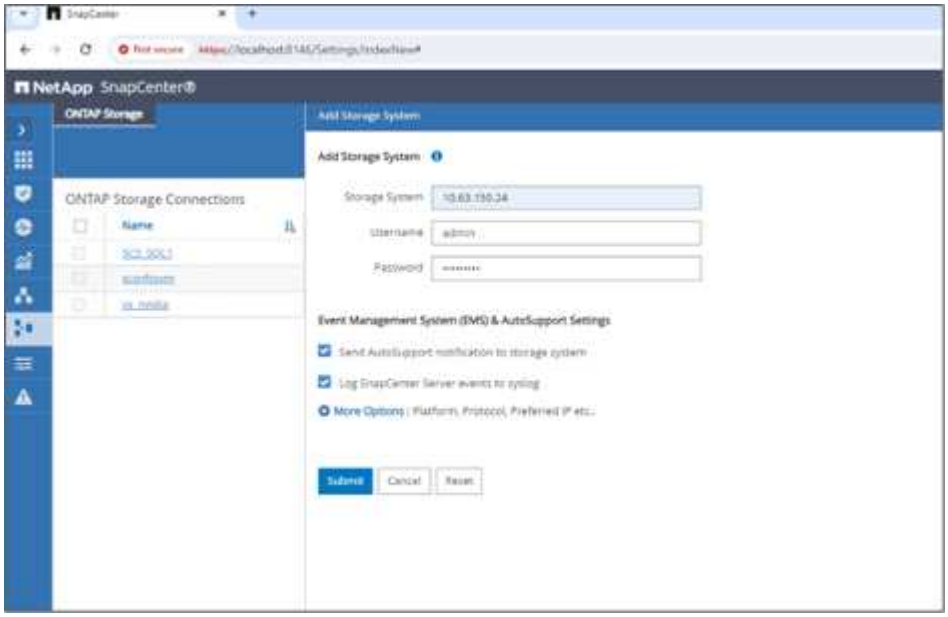


2. 安裝及設定儲存外掛程式。從新增的主機中、選取「更多選項」。瀏覽並從選取下載的儲存外掛程式 "NetApp Automation Store"。安裝外掛程式並儲存組態。

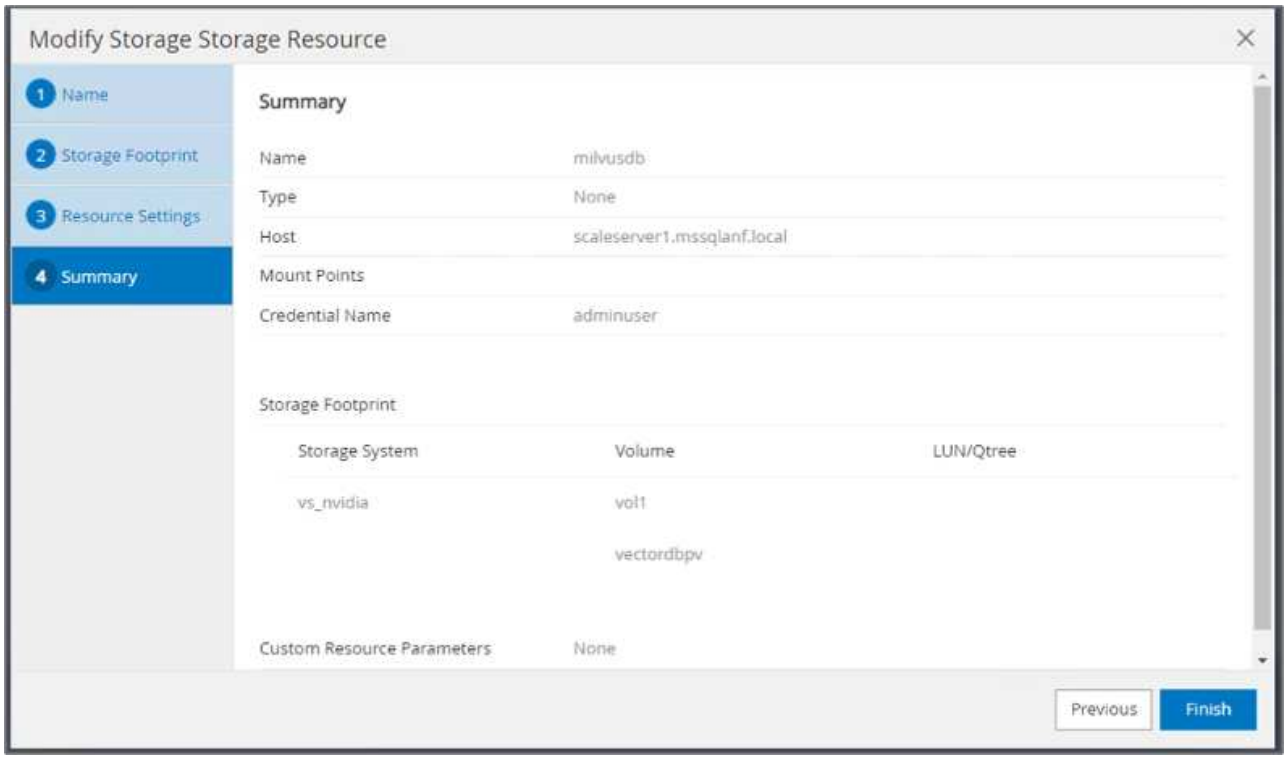


3. 設定儲存系統和磁碟區：在「儲存系統」下新增儲存系統、然後選取 SVM （儲存虛擬機器）。在此範例

中、我們選擇了「 Vs_NVIDIA 」。



4. 為向量資料庫建立資源、並納入備份原則和自訂快照名稱。
   - 啟用預設值的一致性群組備份、並啟用 SnapCenter 而不需檔案系統一致性。
   - 在「儲存空間」區段中、選取與向量資料庫客戶資料和 Milvus 叢集資料相關的磁碟區。在我們的範例中、這些是「 vol1 」和「 vectordbpv 」。
   - 建立向量資料庫保護原則、並使用原則保護向量資料庫資源。



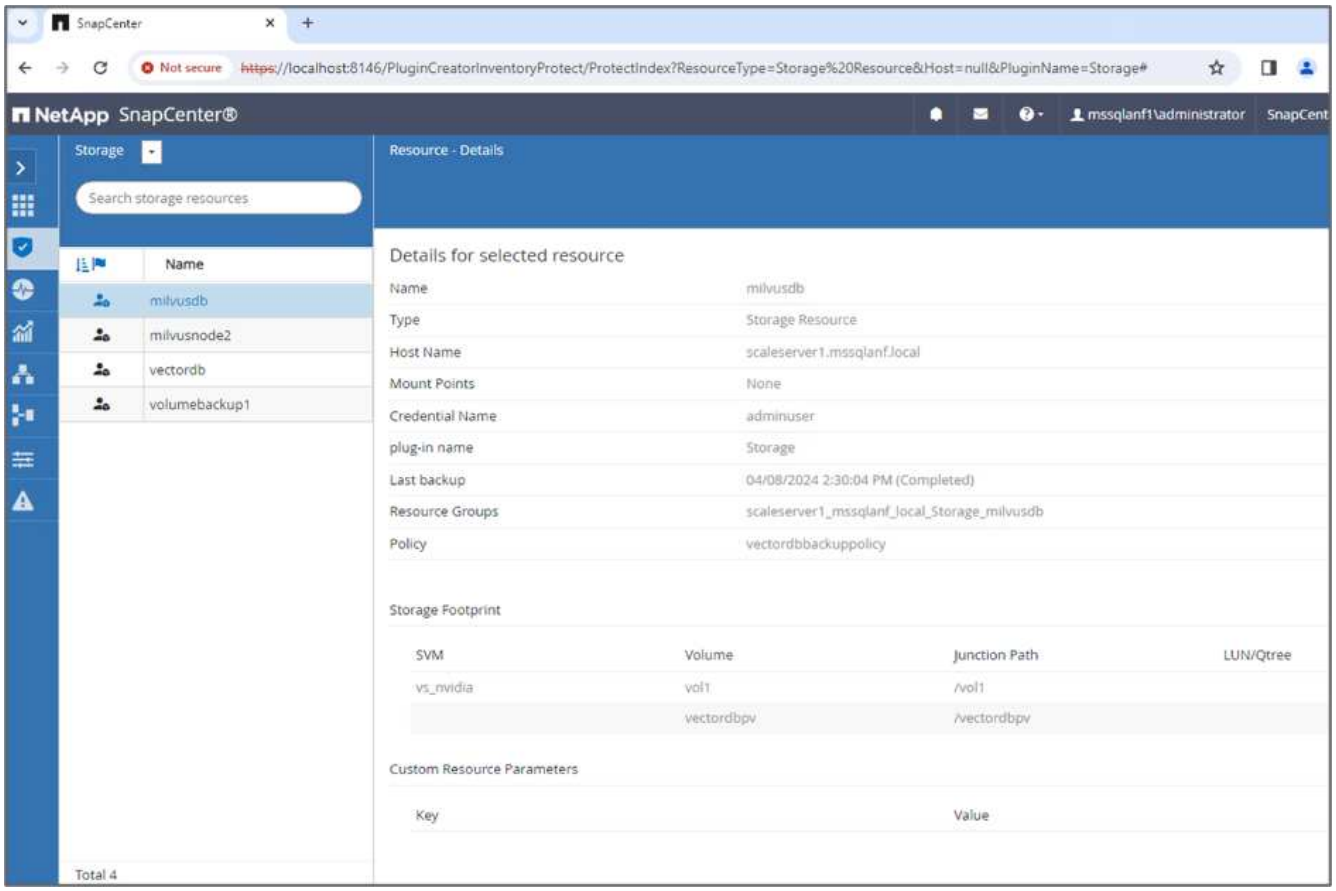
5. 使用 Python 指令碼將資料插入 S3 NAS 貯體。在我們的案例中、我們修改了 Milvus 所提供的備份指令碼、即「 prepy_data_NetApp.py 」、並執行「 Sync 」命令來清除作業系統中的資料。

```
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. 驗證 S3 NAS 貯體中的資料。在我們的範例中、時間戳記為「 2024-04-0821:22 」的檔案是由「 prepy_data_NetApp.py 」指令碼所建立。

```
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. 使用「 ilvusdb 」資源中的一致性群組（ CG ）快照來啟動備份

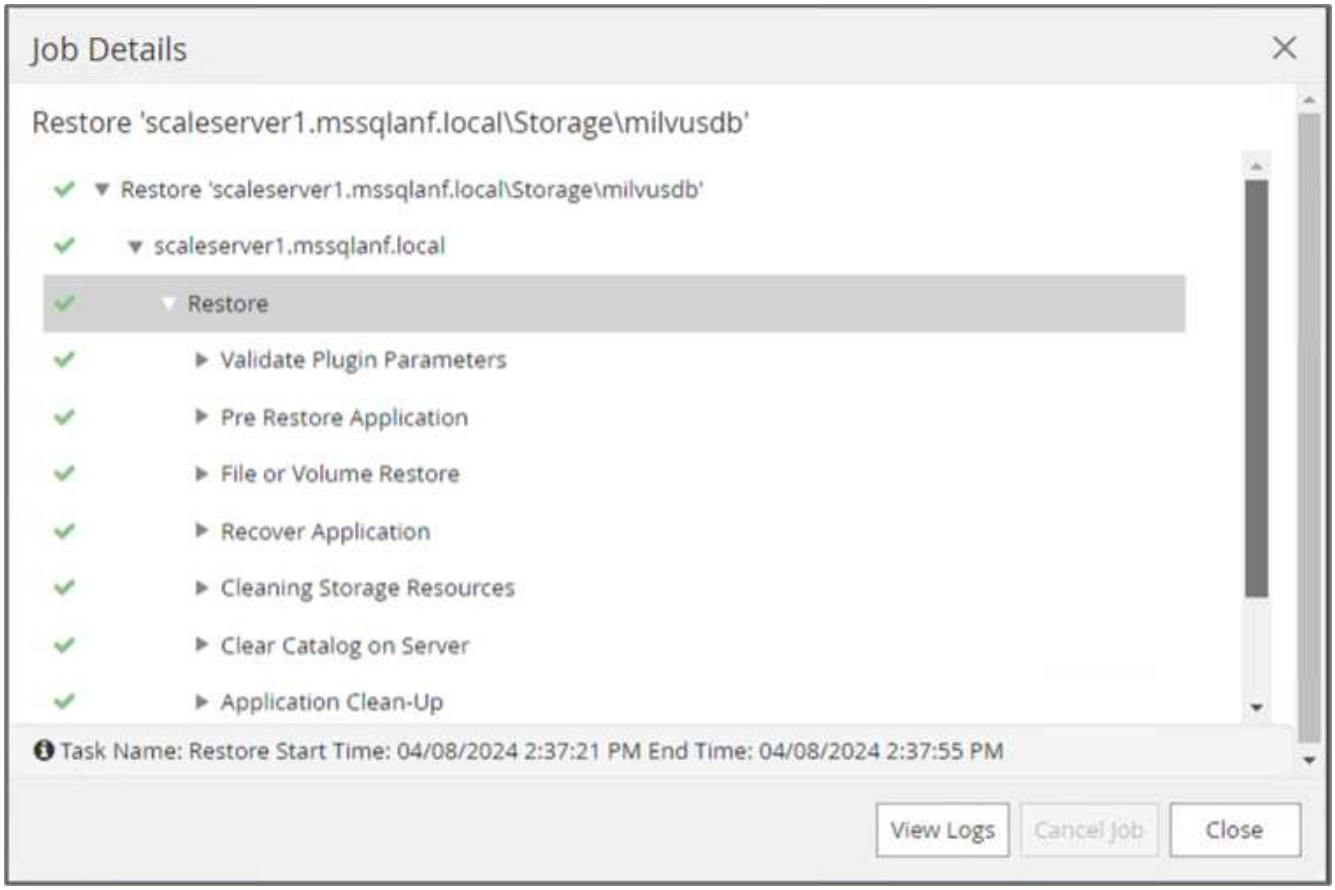


8. 為了測試備份功能、我們會在備份程序之後新增一個表格、或是從 NFS （ S3 NAS 儲存區）移除部分資料。

   在此測試中、假設有人在備份後建立了新的、不必要的或不適當的集合。在這種情況下、我們需要在新增新集合之前、將向量資料庫還原至其狀態。例如、已插入「 hell_milvus_netapp_sc_testnew 」和「 hell_milvus_netapp_sc_testnew2 」等新集合。

```
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. 從先前的快照執行 S3 NAS 儲存區的完整還原。



10. 使用 Python 指令碼來驗證「 hell_milvus_netapp_sc_test 」和「 hell_milvus_netapp_sc_test2 」集合中的資料。

```
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. 確認資料庫中不再存在不必要或不適當的集合。

```
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:~#
```

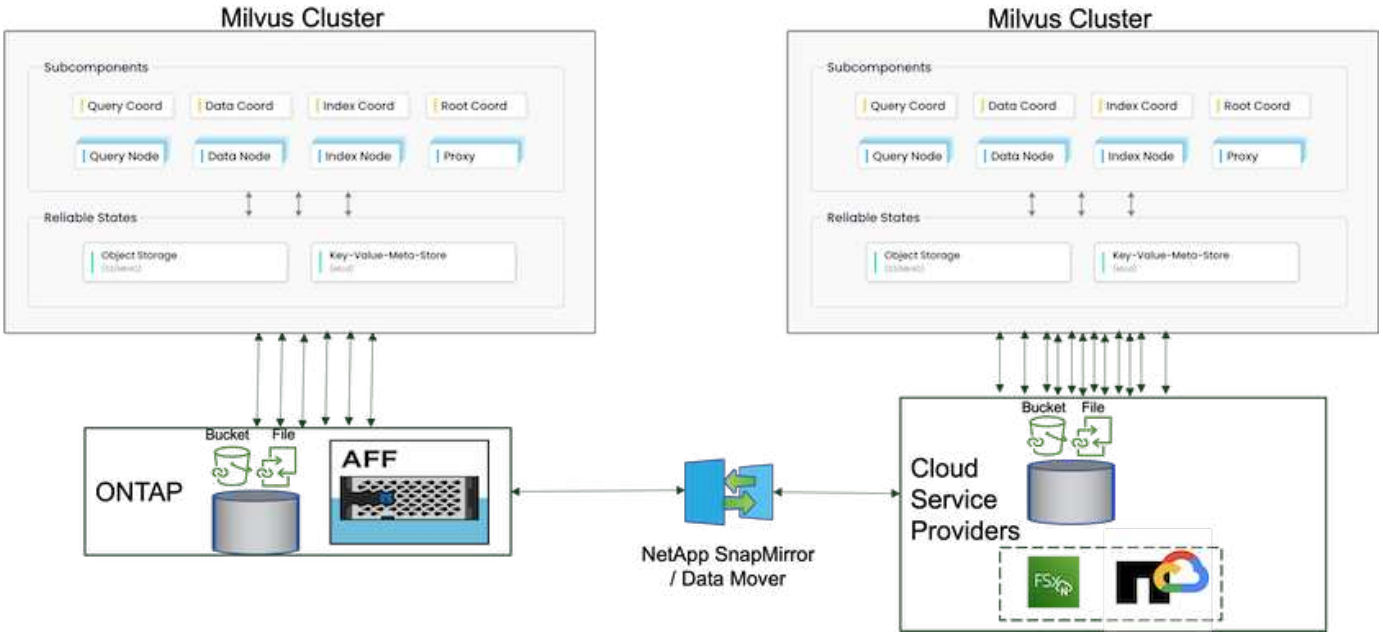
最後、使用 NetApp 的 SnapCenter 來保護向量資料庫資料和位於 ONTAP 的 Milvus 資料、對客戶帶來重大效益、尤其是在資料完整性至關重要的產業、例如電影製作。SnapCenter 能夠建立一致的備份並執行完整資料還原、確保重要資料（例如內嵌視訊和音訊檔案）不會因硬碟故障或其他問題而遺失。這不僅可防止營運中斷、也可防止重大財務損失。

在本節中、我們示範如何設定 SnapCenter 來保護 ONTAP 中的資料、包括主機設定、儲存外掛程式的安裝和組態、以及使用自訂快照名稱建立向量資料庫的資源。我們也展示如何使用一致性群組快照執行備份、並驗證 S3 NAS 儲存區中的資料。

此外、我們模擬的案例是在備份之後建立不必要或不適當的集合。在這種情況下、 SnapCenter 可從先前的快照執行完整還原、確保向量資料庫在新增集合之前、可還原至其狀態、進而維持資料庫的完整性。這項將資料還原到特定時間點的功能對客戶來說非常重要、讓他們能夠保證資料不僅安全、而且能正確維護。因此、 NetApp 的 SnapCenter 產品為客戶提供強大可靠的資料保護與管理解決方案。

使用 NetApp SnapMirror 進行災難恢復

使用 NetApp SnapMirror 進行災難恢復



災難恢復對於維持向量資料庫的完整性和可用度至關重要、尤其是在管理高維度資料和執行複雜的相似性搜尋時、更是如此。妥善規劃且實作的災難恢復策略可確保在發生意外事件（例如硬體故障、自然災害或網路攻擊）時、不會遺失或洩漏資料。這對於仰賴向量資料庫的應用程式而言特別重要、因為資料遺失或毀損可能會導致重大的營運中斷和財務損失。此外、健全的災難恢復計畫也能將停機時間降至最低、並讓服務快速還原、確保業務持續運作。這是透過 NetApp 資料複寫產品鏡射鏡射功能、跨越不同的地理位置、定期備份和容錯移轉機制來達成。因此、災難恢復不僅是一項保護措施、也是負責且有效率的向量資料庫管理的關鍵元件。

NetApp 的 SnapMirror 可將資料從一個 NetApp ONTAP 儲存控制器複寫到另一個儲存控制器、主要用於災難恢復（ DR ）和混合式解決方案。在向量資料庫的環境中、此工具有助於在內部部署環境和雲端環境之間順暢地轉換資料。這項轉換不需要進行任何資料轉換或應用程式重構、因此可提升跨多個平台的資料管理效率與靈活度。

向量資料庫案例中的 NetApp 混合式解決方案可帶來更多優勢：

1. 擴充性： NetApp 的混合雲解決方案可根據您的需求擴充您的資源。您可以將內部部署資源用於一般可預測的工作負載和雲端資源、例如 Amazon FSxN for NetApp ONTAP 和 Google Cloud NetApp Volume （ GCNV ）、以因應尖峰時間或非預期的負載。
2. 成本效益： NetApp 的混合雲模式可讓您將內部部署資源用於一般工作負載、並在需要時僅支付雲端資源的費用、藉此最佳化成本。這種隨用隨付模式可與 NetApp instaclustr 服務產品相較、具有相當高的成本效益。對於內部部署和主要雲端服務供應商、 instaclustr 提供支援和諮詢服務。
3. 靈活度： NetApp 的混合雲可讓您靈活選擇處理資料的位置。例如、您可以選擇在內部環境中執行複雜的向量作業、而內部環境中的硬體功能更強大、而且雲端作業的密集度也更低。
4. 營運不中斷：萬一發生災難、將資料放在 NetApp 混合雲中可確保營運不中斷。如果內部部署資源受到影響、您可以快速切換至雲端。我們可以利用 NetApp SnapMirror 將資料從內部部署移至雲端、反之亦然。
5. 創新： NetApp 的混合雲解決方案也能提供最先進的雲端服務與技術、以加速創新。NetApp 在雲端的創新技術、例如 Amazon FSxN for NetApp ONTAP 、 Azure NetApp Files 和 Google Cloud NetApp Volumes 、都是雲端服務供應商的創新產品和偏好的 NAS 。

向量資料庫效能驗證

效能驗證

效能驗證在向量資料庫和儲存系統中都扮演重要角色、是確保最佳運作和有效資源使用率的關鍵因素。向量資料庫以處理高維度資料和執行相似度搜尋而聞名、因此需要維持高效能層級、才能快速準確地處理複雜的查詢。效能驗證有助於識別瓶頸、微調組態、並確保系統能夠處理預期的負載、而不會降低服務品質。同樣地、在儲存系統中、效能驗證是確保資料儲存及擷取效率的關鍵、而不會產生延遲問題或瓶頸、進而影響整體系統效能。它也有助於在資訊充足的情況下、針對必要的儲存基礎架構升級或變更做出決策。因此、效能驗證是系統管理的關鍵層面、有助於維持高服務品質、營運效率和整體系統可靠性。

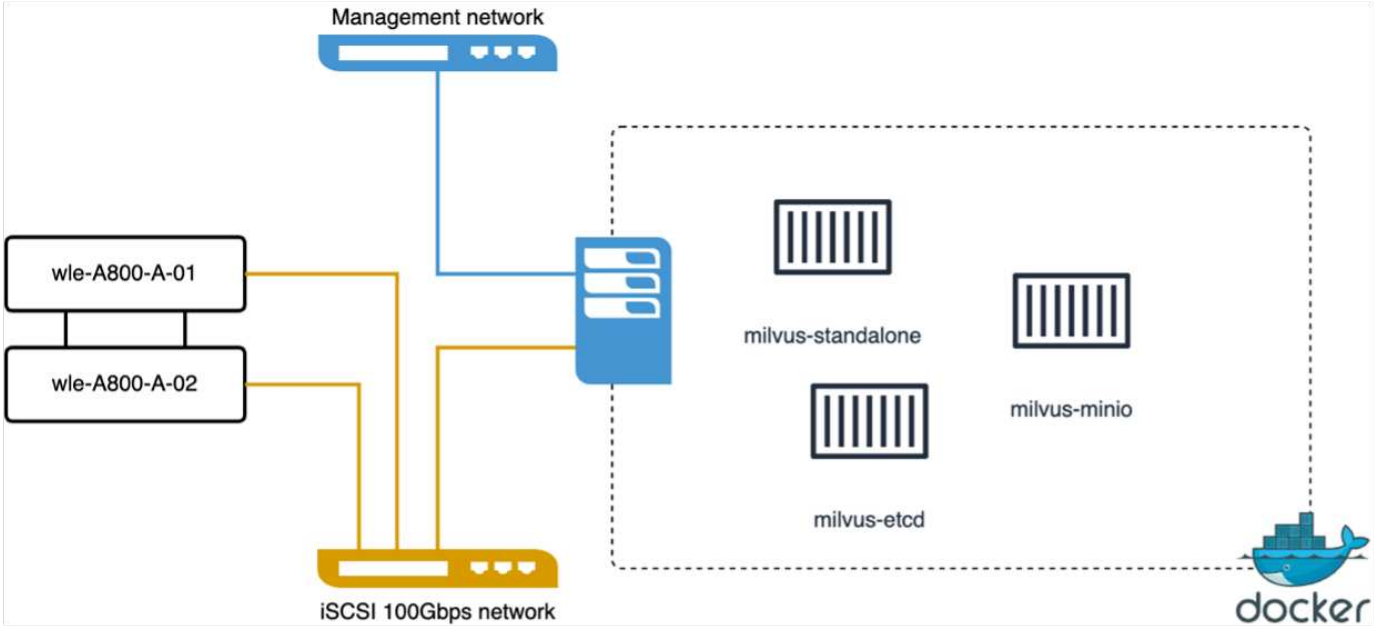
在本節中、我們的目標是深入探討向量資料庫（例如 Milvus 和 pgveco.RS ）的效能驗證、重點在於其儲存效能特性、例如 I/O 設定檔和 NetApp 儲存控制器行為、以支援 LLM 生命週期內的 RAG 和推斷工作負載。當這些資料庫與 ONTAP 儲存解決方案結合使用時、我們會評估並找出任何效能差異。我們的分析將以關鍵效能指標為基礎、例如每秒處理的查詢數（ QPS ）。

請查看以下 Milvus 使用的方法和進度。

詳細資料	**Milvus** （獨立式和叢集）	**Postgres** （ **pgveco.RS** ）
版本	2.3.2.	0.2.0
檔案系統	iSCSI LUN 上的 XFS	
工作負載產生器	"VectorDB-Bench" – v0.0.5.	
資料集	LAION 資料集 * 1 億套嵌入式產品 * 768 尺寸 * ~300GB 資料集大小	

VectorDB-Bench 搭配 **Milvus** 獨立式叢集

我們在採用 vectorDB-Bench 的 mivus 獨立叢集上進行了下列效能驗證。
milvus 獨立叢集的網路和伺服器連線能力如下。



在本節中、我們分享了測試 Milvus 獨立式資料庫的觀察結果和結果。

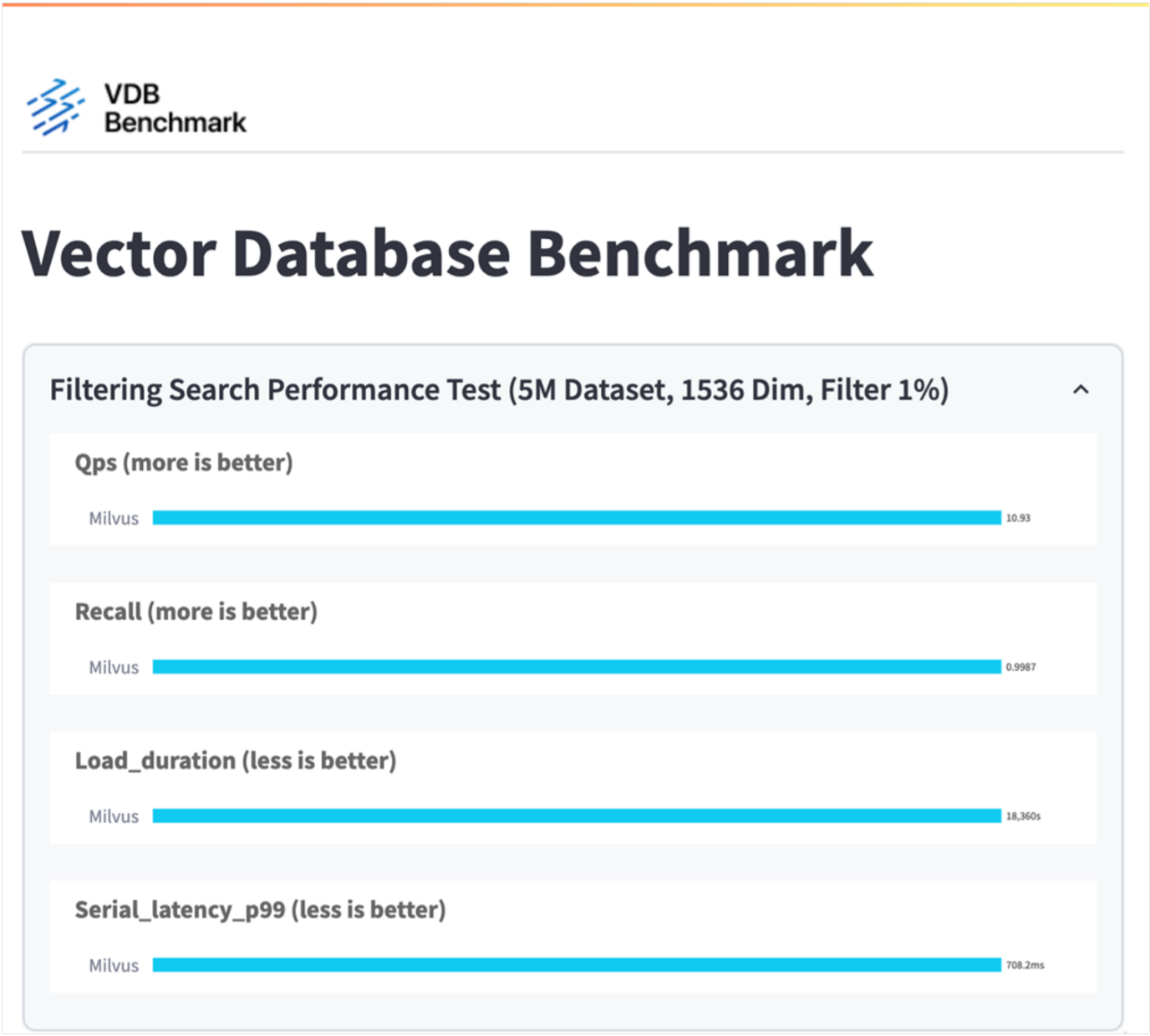
。 我們選擇 DiskANN 作為這些測試的索引類型。
。 擷取、最佳化及建立約 100GB 資料集的索引約需 5 小時。在這段期間中、配備 20 個核心（啟用超執行緒時等於 40 個 vCPU ）的 Milvus 伺服器、其最大 CPU 容量為 100% 。我們發現、對於超過系統記憶體大小的大型資料集而言、 DiskANN 特別重要。
。 在查詢階段中、我們觀察到每秒查詢數（ QPS ）率為 10.93 、而且回收率為 0.9987 。查詢的第 99 個百分位數延遲是以 708.2 毫秒的時間來測量。

從儲存的角度來看、資料庫在擷取、插入後最佳化和索引建立階段中、每秒發行約 1 、 000 次作業。在查詢階段、它需要 32,000 個作業 / 秒

下節將說明儲存效能指標。

工作負載階段	度量	價值
資料擷取 和 插入後最佳化	IOPS	< 1 、 000
	延遲	< 400 美元
	工作負載	讀取 / 寫入混合、大部分是寫入
	IO 大小	64KB
查詢	IOPS	尖峰時間為 32,000
	延遲	< 400 美元
	工作負載	100% 快取讀取
	IO 大小	主要 8KB

vectorDB-bench 結果如下。

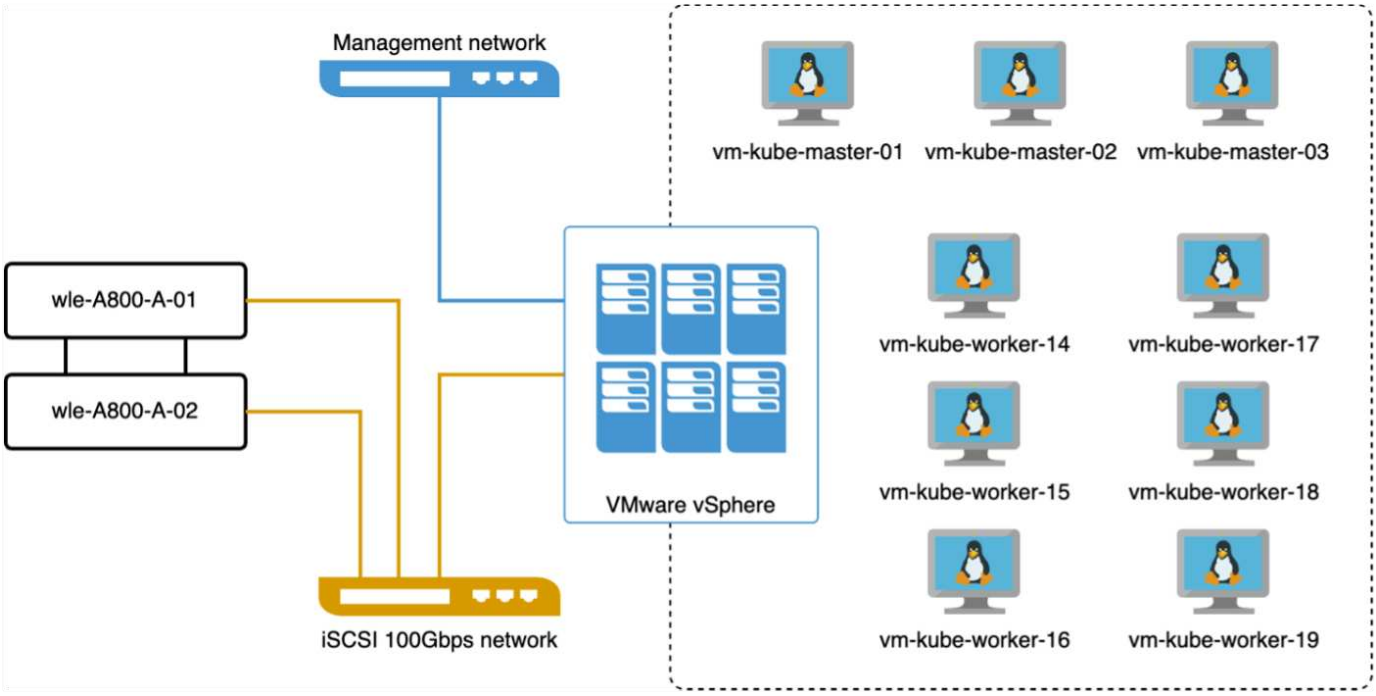


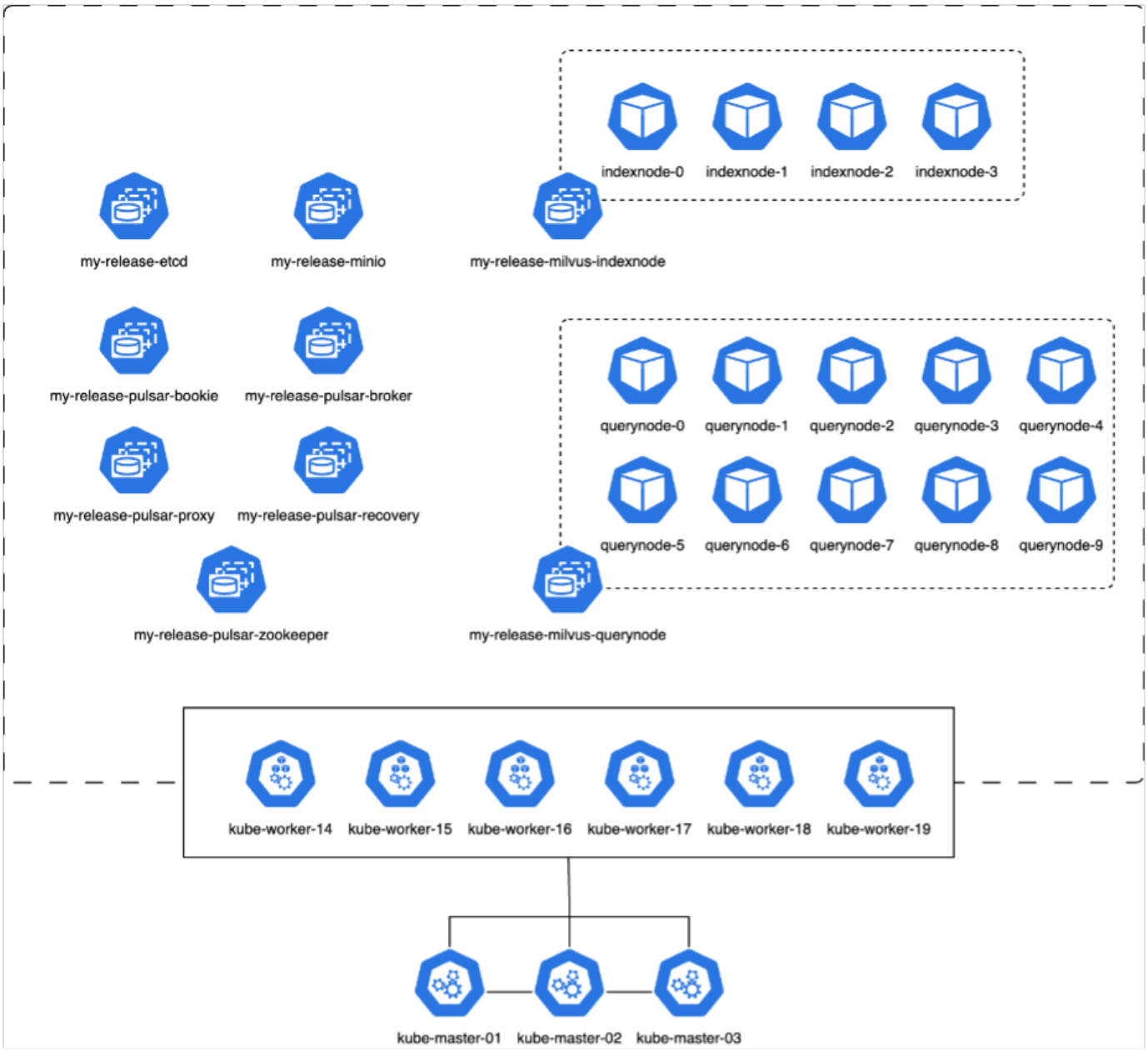
從獨立式 Milvus 執行個體的效能驗證來看、目前的設定顯然不足以支援容量為 1536 的 500 萬向量資料集。我們已確定儲存設備擁有足夠的資源、並不構成系統的瓶頸。

VectorDB-Bench 搭配 **milvus** 叢集

在本節中、我們將討論在 Kubernetes 環境中部署 Milvus 叢集的問題。這項 Kubernetes 設定是在 VMware vSphere 部署的基礎上建構、該部署是 Kubernetes 主節點和工作節點的主節點。

VMware vSphere 和 Kubernetes 部署的詳細資料將在下列各節中說明。


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



在本節中、我們會介紹測試 Milvus 資料庫的觀察結果和結果。
* 使用的索引類型為 DiskANN 。
* 下表提供獨立部署與叢集部署之間的比較、以 1536 的維度處理 500 萬個向量。我們觀察到、叢集部署中的資料擷取和插入後最佳化所需時間較短。與獨立安裝相比、叢集部署中查詢延遲的第 99 百分位數減少了六倍。
* 雖然叢集部署中的每秒查詢數（ QPS ）速率較高、但並未達到所需的層級。

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%

下圖提供各種儲存指標的檢視、包括儲存叢集延遲和 IOPS 總計（每秒輸入 / 輸出作業數）。

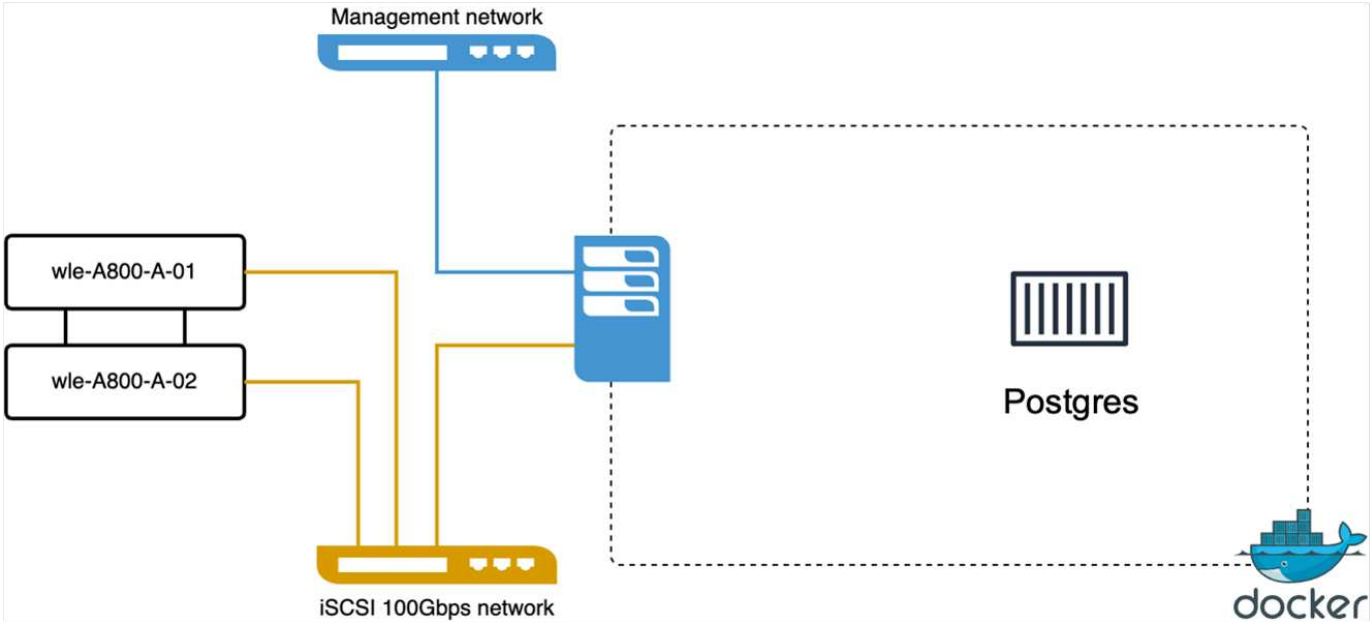
下節將說明主要的儲存效能指標。

工作負載階段	度量	價值
資料擷取 和 插入後最佳化	IOPS	< 1 、 000
	延遲	< 400 美元
	工作負載	讀取 / 寫入混合、大部分是寫入
	IO 大小	64KB
查詢	IOPS	尖峰為 147,000
	延遲	< 400 美元
	工作負載	100% 快取讀取
	IO 大小	主要 8KB

根據獨立式 Milvus 和 Milvus 叢集的效能驗證、我們提供儲存 I/O 設定檔的詳細資料。
* 我們觀察到、在獨立部署和叢集部署中、 I/O 設定檔保持一致。
* 觀察到的尖峰 IOPS 差異、可歸因於叢集部署中的用戶端數量較多。

vectorDB-Bench 搭配 **Postgres** （ **pgvector.RS** ）

我們使用 VectorDB-Bench 在 PostgreSQL （ pgvec托 .RS ）上執行下列動作：
PostgreSQL （特別是 pgveco.RS ）的網路和伺服器連線詳細資料如下：

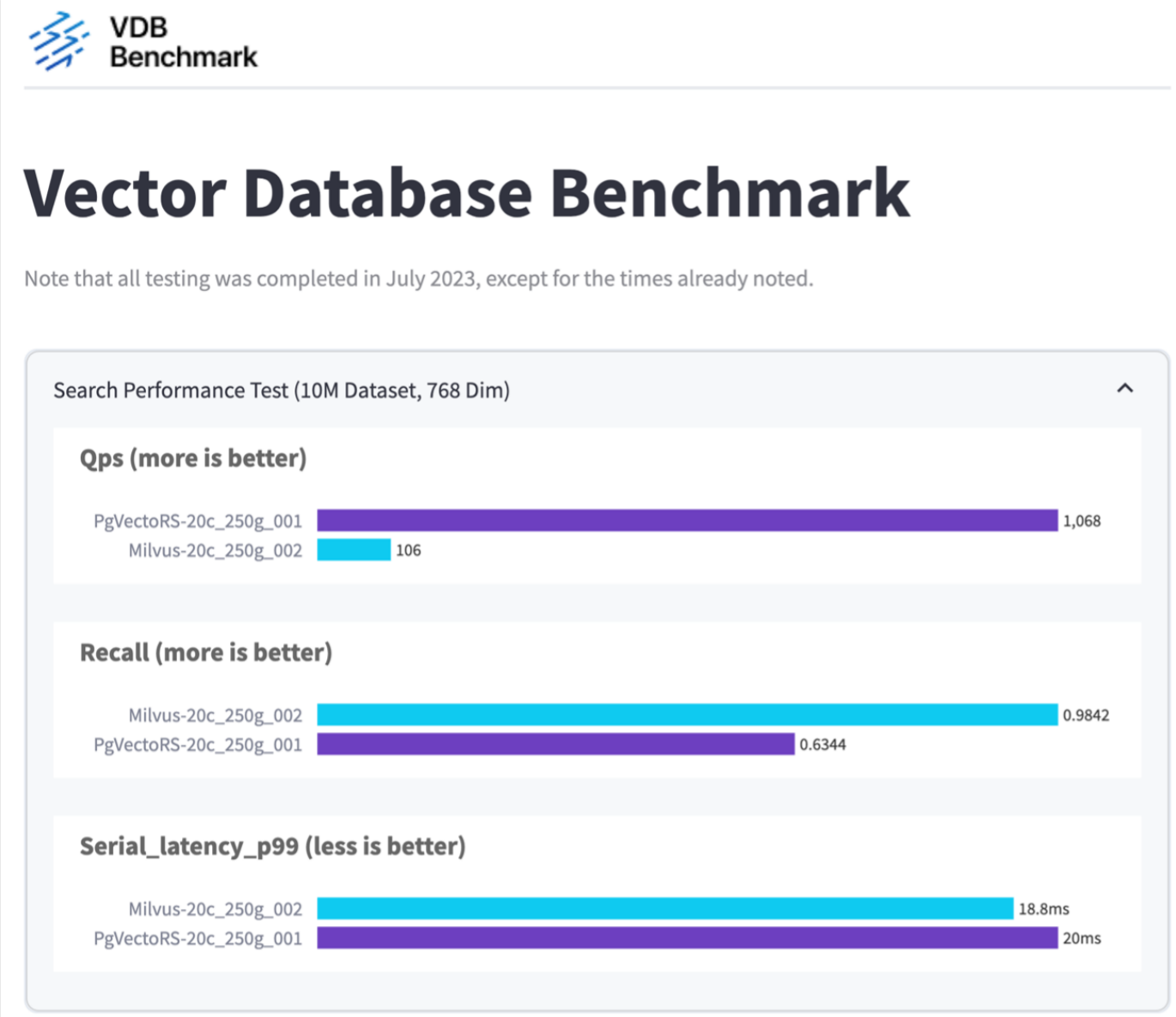


在本節中、我們分享了測試 PostgreSQL 資料庫的觀察結果、特別是使用 pgveco.RS 。
* 我們選擇 HNSW 作為這些測試的索引類型、因為在測試時、 DiskANN 無法用於 pgveco.RS 。
* 在資料擷取階段、我們載入 Cohere 資料集、其中包含 1 、 000 萬個向量、維度為 768 。此程序約需 4.5 小時。
* 在查詢階段、我們觀察到每秒查詢數（ QPS ）為 1 、 068 、召回率為 0.6344 。查詢的第 99 個百分位數延遲是以 20 毫秒為測量單位。在大部分的執行時間中、用戶端 CPU 以 100% 的容量運作。

下圖提供各種儲存指標的檢視、包括儲存叢集延遲總計 IOPS （每秒輸入 / 輸出作業數）。


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["錯誤：缺少圖形影像"]
```

向量 **DB Bench** 上的 **milvus** 與 **postgres** 效能比較



根據我們使用 VectorDBBench 對 Milvus 和 PostgreSQL 的效能驗證、我們觀察到下列事項：

- 索引類型： HNSW
- 資料集： Cohere 提供 1 、 000 萬個向量、尺寸 768

我們發現 pgveco.RS 的每秒查詢數（ QPS ）為 1 、 068 、回收率為 0.6344 、而 Milvus 的 QPS 率為 106 、回收率為 0.9842 。

如果查詢的高精度是優先順序、 Milvus 會比 pgveco.RS 更出色、因為它會擷取每個查詢的相關項目比例更高。不過、如果每秒查詢數是更重要的因素、 pgveco.RS 就會超過 Milvus 。不過、請務必注意、透過 pgvecto 擷取的資料品質較低、其中約 37% 的搜尋結果是不相關的項目。

根據我們的效能驗證進行觀察：

根據我們的績效驗證、我們提出下列觀察：

在 Milvus 中、 I/O 設定檔與 OLTP 工作負載非常相似、例如 Oracle slob 。基準測試包含三個階段：資料擷取、最佳化後及查詢。初始階段的主要特徵是 64KB 寫入作業、而查詢階段則主要涉及 8KB 讀取。我們期望 ONTAP

能以專業的方式處理 Milvus I/O 負載。

PostgreSQL I/O 設定檔並不代表具有挑戰性的儲存工作負載。由於目前正在進行記憶體內建實作、我們在查詢階段並未觀察到任何磁碟 I/O 。

DiskANN 是儲存差異化的關鍵技術。它能有效擴充向量 DB 搜尋、使其超越系統記憶體界限。但是、不太可能利用記憶體內向量 DB 指數（例如 HNSW ）來建立儲存效能差異化。

此外、值得注意的是、當索引類型為 HSNW 時、在查詢階段、儲存設備並不扮演關鍵角色、 HSNW 是支援 RAG 應用程式的向量資料庫最重要的作業階段。這裏的含意是、儲存效能不會對這些應用程式的整體效能造成重大影響。