NetApp的向量資料庫解決方案

NetApp artificial intelligence solutions

NetApp
February 12, 2026

目錄

NetApp的向量資料庫解決方案

NetApp的向量資料庫解決方案

Karthikeyan Nagalingam 與 Rodrigo Nascimento， NetApp

本文檔深入探討了使用 NetApp 儲存解決方案部署和管理向量資料庫（例如 Milvus 和開源 PostgreSQL 擴充 pgvecto）的方法。詳細介紹了使用NetApp ONTAP和StorageGRID物件儲存的基礎設施指南，並驗證了 Milvus 資料庫在 AWS FSx ONTAP中的應用。該文件闡明了 NetApp 的文件物件二元性及其對支援向量嵌入的向量資料庫和應用程式的實用性。它強調了 NetApp 企業管理產品SnapCenter的功能，為向量資料庫提供備份和復原功能，確保資料的完整性和可用性。該文件進一步深入探討了 NetApp 的混合雲解決方案，討論了其在本機和雲端環境中的資料複製和保護中的作用。它包括對NetApp ONTAP上向量資料庫效能驗證的見解，並總結了生成 AI 的兩個實際用例：帶有 LLM 的 RAG 和 NetApp 的內部 ChatAI。本文檔是利用 NetApp 儲存解決方案管理向量資料庫的綜合指南。

參考架構重點在於以下內容：

介紹

本節介紹NetApp的向量資料庫解決方案。

介紹

向量資料庫有效地解決了旨在處理大型語言模型 (LLM) 和生成人工智慧 (AI) 中的語義搜尋複雜性的挑戰。與傳統的數據管理系統不同，向量資料庫能夠使用數據本身的內容而不是標籤或標記來處理和搜尋各種類型的數據，包括圖像、視訊、文字、音訊和其他形式的非結構化資料。

關聯式資料庫管理系統 (RDBMS) 的限制是有據可查的，尤其是它們在處理人工智慧應用中常見的高維度資料表示和非結構化資料時遇到的困難。 RDBMS 通常需要將資料扁平化為更易於管理的結構，這個過程既耗時又容易出錯，從而導致搜尋延遲和效率低下。向量資料庫的出現解決了這些問題，為複雜高維資料的管理和搜尋提供了更有效率、更準確的解決方案，從而促進了人工智慧應用的發展。

本文檔為目前正在使用或計劃使用向量資料庫的客戶提供全面的指南，詳細介紹了在NetApp ONTAP、 NetApp StorageGRID、 Amazon FSx ONTAP for NetApp ONTAP和SnapCenter等平台上使用向量資料庫的最佳實務。本文提供的內容涵蓋了一系列主題：

- NetApp儲存透過NetApp ONTAP和StorageGRID物件儲存為 Milvus 等向量資料庫提供基礎設施指南。
- 透過檔案和物件儲存驗證 AWS FSx ONTAP中的 Milvus 資料庫。
- 深入研究 NetApp 的檔案物件二元性，展示其對向量資料庫以及其他應用程式中的資料的實用性。
- NetApp 的資料保護管理產品SnapCenter如何為向量資料庫資料提供備份和復原功能。
- NetApp 的混合雲如何在本機和雲端環境中提供資料複製和保護。
- 提供有關NetApp ONTAP上 Milvus 和 pgvector 等向量資料庫的效能驗證的見解。
- 兩個具體的用例：具有大型語言模型 (LLM) 的檢索增強生成 (RAG) 和NetApp IT 團隊的 ChatAI，從而提供所概述的概念和實踐的實際範例。

解決方案概述

本節概述了NetApp向量資料庫解決方案。

解決方案概述

該解決方案展示了NetApp為應對向量資料庫客戶面臨的挑戰所提供的獨特優勢和功能。透過利用NetApp ONTAP、 StorageGRID、NetApp 的雲端解決方案和SnapCenter，客戶可以為其業務營運增加顯著的價值。這些工具不僅解決了現有的問題，還提高了效率和生產力，從而促進了整體業務成長。

為什麼選擇**NetApp**？

- NetApp 的產品（例如ONTAP和StorageGRID）允許分離儲存和運算，從而能夠根據特定需求實現最佳資源利用率。這種靈活性使客戶能夠使用NetApp儲存解決方案獨立擴展其儲存。
- 透過利用 NetApp 的儲存控制器，客戶可以使用 NFS 和 S3 協定有效地向其向量資料庫提供資料。這些協定方便了客戶資料儲存和管理向量資料庫索引，從而無需透過檔案和物件方法存取多個資料副本。
- NetApp ONTAP為 AWS、Azure 和 Google Cloud 等領先的雲端服務供應商提供對 NAS 和物件儲存的原生支援。這種廣泛的兼容性確保了無縫集成，實現了客戶資料移動性、全球可訪問性、災難復原、動態可擴展

性和高效能。

- 借助 NetApp 強大的資料管理功能，客戶可以放心，因為他們的資料受到良好的保護，不會受到潛在風險和威脅。 NetApp優先考慮資料安全，讓客戶對其寶貴資訊的安全性和完整性感到放心。

向量資料庫

本節介紹NetApp AI 解決方案中向量資料庫的定義與使用。

向量資料庫

向量資料庫是一種特殊類型的資料庫，旨在使用機器學習模型的嵌入來處理、索引和搜尋非結構化資料。它不以傳統的表格格式組織數據，而是將數據排列為高維向量，也稱為向量嵌入。這種獨特的結構使得資料庫能夠更有效率、更準確地處理複雜、多維的資料。

向量資料庫的關鍵功能之一是使用生成式人工智慧進行分析。這包括相似性搜索，其中資料庫識別類似於給定輸入的資料點，以及異常檢測，其中它可以發現與常態有顯著偏差的資料點。

此外，向量資料庫非常適合處理時間資料或帶有時間戳記的資料。這種類型的數據提供有關發生了什麼以及何時發生的信息，按順序以及與給定 IT 系統中所有其他事件的關係。這種處理和分析時間資料的能力使得向量資料庫對於需要了解隨時間推移的事件的應用程式特別有用。

向量資料庫對於機器學習和人工智慧的優勢：

- 高維搜尋：向量資料庫擅長管理和檢索高維數據，這些數據通常在 AI 和 ML 應用程式中產生。
- 可擴展性：它們可以有效擴展以處理大量數據，支援 AI 和 ML 項目的成長和擴展。
- 靈活性：向量資料庫具有高度的靈活性，可以適應多種資料類型和結構。
- 效能：它們提供高效能資料管理和檢索，這對於 AI 和 ML 操作的速度和效率至關重要。
- 可自訂的索引：向量資料庫提供可自訂的索引選項，從而能夠根據特定需求優化資料組織和檢索。

向量資料庫和用例。

本節提供各種向量資料庫及其用例詳細資訊。

Faiss和ScaNN

它們是向量搜尋領域中的重要工具庫。這些庫提供的功能有助於管理和搜尋向量數據，使其成為數據管理這一專業領域的寶貴資源。

Elasticsearch

它是一種廣泛使用的搜尋和分析引擎，最近加入了向量搜尋功能。此新功能增強了其功能，使其能夠更有效地處理和搜尋向量資料。

松果

它是一個具有一組獨特功能的強大向量資料庫。它的索引功能同時支援密集和稀疏向量，從而增強了其靈活性和適應性。它的主要優勢之一在於能夠將傳統搜尋方法與基於人工智慧的密集向量搜尋相結合，從而創造出一種兼具兩全其美的混合搜尋方法。

Pinecone 主要基於雲端，專為機器學習應用而設計，可與各種平台良好集成，包括 GCP、AWS、Open AI、GPT-3、GPT-3.5、GPT-4、Catgut Plus、Elasticsearch、Haystack 等。值得注意的是，Pinecone 是一個閉源平台，可作為軟體即服務 (SaaS) 產品使用。

鑑於其先進的功能，Pinecone 特別適合網路安全產業，其高維度搜尋和混合搜尋功能可以有效地利用來檢測和應對威脅。

色度

它是一個向量資料庫，具有包含四個主要功能的核心 API，其中一個功能包括記憶體文件向量儲存。它還利用 Face Transformers 庫來向量化文檔，增強其功能和多功能性。 Chroma 的設計可在雲端和本地運行，可根據使用者需求提供靈活性。特別是在音訊相關應用方面表現出色，使其成為基於音訊的搜尋引擎、音樂推薦系統和其他音訊相關用例的絕佳選擇。

威維特

它是一個多功能向量資料庫，允許用戶使用其內建模組或自訂模組向量化其內容，根據特定需求提供靈活性。它提供完全託管和自託管解決方案，滿足各種部署偏好。

Weaviate 的主要功能之一是它能夠同時儲存向量和對象，從而增強其資料處理能力。它廣泛應用於一系列應用，包括 ERP 系統中的語義搜尋和資料分類。在電子商務領域，它為搜尋和推薦引擎提供支援。 Weaviate 也用於影像搜尋、異常偵測、自動資料協調和網路安全威脅分析，展示了其在多個領域的多功能性。

Redis

Redis 是一種高效能向量資料庫，以其快速的記憶體儲存而聞名，可為讀寫作業提供低延遲。這使其成為需要快速數據存取的推薦系統、搜尋引擎和數據分析應用程式的絕佳選擇。

Redis 支援向量的各種資料結構，包括列表、集合和有序集。它還提供向量運算，例如計算向量之間的距離或尋找交集和並集。這些功能對於相似性搜尋、聚類和基於內容的推薦系統特別有用。

在可擴展性和可用性方面，Redis 擅長處理高吞吐量工作負載並提供資料複製。它還可以與其他資料類型很好地集成，包括傳統的關係資料庫（RDBMS）。 Redis 包含一個用於即時更新的發布/訂閱（Pub/Sub）功能，這有利於管理即時向量。此外，Redis 輕量且易於使用，使其成為管理向量資料的使用者友善解決方案。

Milvus

它是一個多功能的向量資料庫，提供類似文件儲存的 API，非常類似於 MongoDB。它因支援多種數據類型而脫穎而出，成為數據科學和機器學習領域的熱門選擇。

Milvus 的獨特功能之一是其多向量化功能，它允許使用者在運行時指定用於搜尋的向量類型。此外，它利用 Knowwhere（一個位於 Faiss 等其他庫之上的庫）來管理查詢和向量搜尋演算法之間的通訊。

由於與 PyTorch 和 TensorFlow 相容，Milvus 還提供與機器學習工作流程的無縫整合。這使其成為一系列應用的絕佳工具，包括電子商務、圖像和視訊分析、物件識別、圖像相似性搜尋和基於內容的圖像檢索。在自然語言處理領域，Milvus 用於文件聚類、語意搜尋和問答系統。

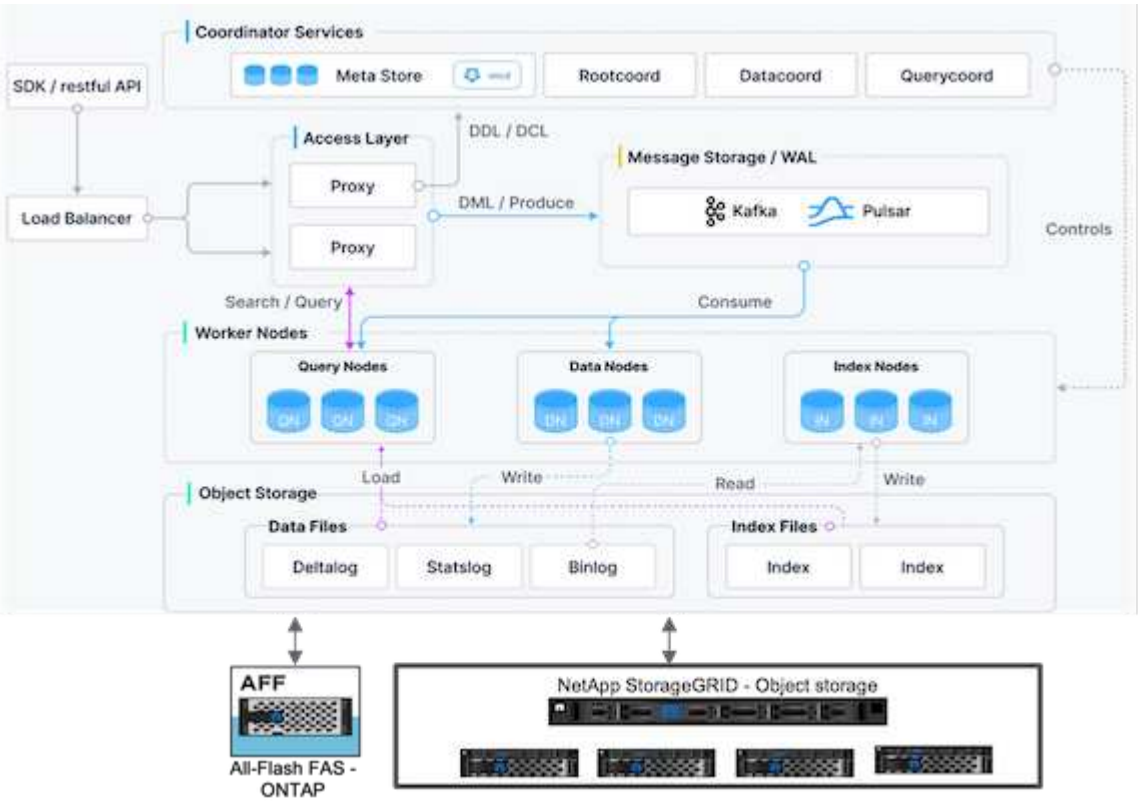
對於這個解決方案，我們選擇了 milvus 進行解決方案驗證。為了提高效能，我們同時使用了 milvus 和 postgres（pgvecto.rs）。

為什麼我們選擇 **milvus** 作為這個解決方案？

- 開源：Milvus 是一個開源向量資料庫，鼓勵社群驅動的開發和改進。

- AI 整合：它利用嵌入相似性搜尋和 AI 應用程式來增強向量資料庫功能。
- 大容量處理：Milvus 有能力儲存、索引和管理由深度神經網路 (DNN) 和機器學習 (ML) 模型產生的超過十億個嵌入向量。
- 使用者友善：易於使用，設定只需不到一分鐘。 Milvus 也為不同的程式語言提供 SDK。
- 速度：它提供極快的檢索速度，比一些替代方案快 10 倍。
- 可擴展性和可用性：Milvus 具有高度可擴展性，可根據需要進行擴展和縮小。
- 功能豐富：它支援不同的資料類型、屬性過濾、使用者定義函數 (UDF) 支援、可配置的一致性等級和旅行時間，使其成為各種應用的多功能工具。

Milvus 架構概述



本節提供 Milvus 架構中使用的更高層級的元件和服務。 * 存取層－由一組無狀態代理程式組成，作為系統的前端層和使用者的端點。 * 協調器服務－它將任務分配給工作節點並充當系統的大腦。它有三種協調器類型：根協調器、資料協調器和查詢協調器。 * 工作節點：它遵循協調服務的指令並執行使用者觸發的DML / DDL命令。它有三種類型的工作節點，例如查詢節點，資料節點和索引節點。 * 儲存：負責資料持久化。它包括元存儲、日誌代理和物件存儲。 NetApp儲存（例如ONTAP和StorageGRID）為 Milvus 提供物件儲存和基於文件的存儲，用於客戶資料和向量資料庫資料。

技術要求

本節概述了NetApp向量資料庫解決方案的要求。

技術要求

除性能外，下面概述的硬體和軟體配置用於本文檔中執行的大部分驗證。這些配置可作為幫助您設定環境的指

南。但請注意，具體組件可能會因個別客戶的要求而有所不同。

硬體需求

硬體	細節
NetApp AFF儲存陣列 HA 對	* A800 * ONTAP 9.14.1 * 48 x 3.49TB SSD-NVM * 兩個靈活組卷：元資料和資料。 * 元資料 NFS 磁碟區有 12 個持久卷，每個磁碟區為 250GB。 * 數據是ONTAP NAS S3 卷
6台富士通PRIMERGY RX2540 M4	* 64 個 CPU * Intel® Xeon® Gold 6142 CPU @ 2.60GHz * 256 GM 實體記憶體 * 1 x 100GbE 網路連接埠
聯網	100 GbE
StorageGRID	* 1 x SG100，3xSGF6024 * 3 x 24 x 7.68TB

軟體需求

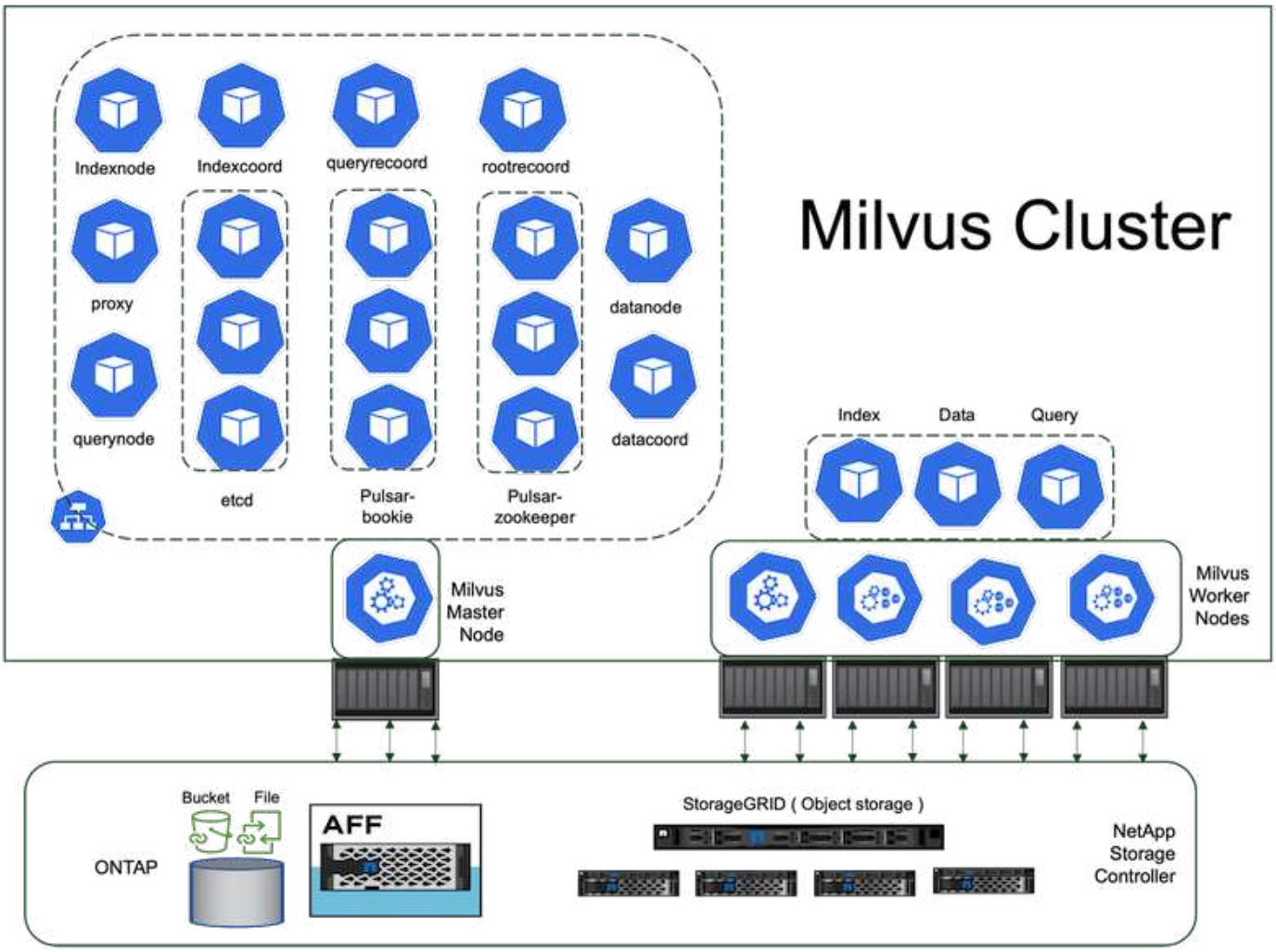
軟體	細節
Milvus 集群	* 圖表 - milvus-4.1.11。 * APP 版本 – 2.3.4 * 依賴的 bundles，例如 bookkeeper、zookeeper、pulsar、etcd、proxy、querynode、worker
Kubernetes	* 5 節點 K8s 叢集 * 1 個主節點和 4 個工作節點 * 版本 – 1.7.2
Python	*3.10.12.

部署流程

本節討論NetApp向量資料庫解決方案的部署過程。

部署流程

在本部署部分中，我們使用 milvus 向量資料庫和 Kubernetes 進行以下實驗設定。



NetApp 儲存為叢集提供存儲，以保存客戶資料和 Milvus 叢集資料。

NetApp儲存設定 **– ONTAP**

- 儲存系統初始化
- 儲存虛擬機器 (SVM) 創建
- 邏輯網路介面的分配
- NFS、S3 配置和許可

對於 NFS（網路檔案系統），請依照下列步驟操作：

1. 為 NFSv4 建立FlexGroup區。在我們為此驗證所做的設定中，我們使用了 48 個 SSD，其中 1 個 SSD 專用於控制器的根卷，另外 47 個 SSD 分佈用於 NFSv4]].驗證FlexGroup卷的 NFS 導出策略是否對 Kubernetes（K8s）節點網路具有讀取/寫入權限。如果沒有這些權限，請授予 K8s 節點網路的讀取/寫入 (rw) 權限。
2. 在所有 K8s 節點上，建立一個資料夾，並透過每個 K8s 節點上的邏輯介面 (LIF) 將FlexGroup磁碟區掛載到該資料夾上。

對於 NAS S3（網路附加儲存簡單儲存服務），請依照下列步驟操作：

1. 為 NFS 建立FlexGroup區。

2. 使用「vserver object-store-server create」指令設定一個啟用 HTTP 的物件儲存伺服器，並將管理狀態設為「up」。您可以選擇啟用 HTTPS 並設定自訂偵聽器連接埠。
3. 使用「vserver object-store-server user create -user <username>」指令建立 object-store-server 使用者。
4. 若要取得存取金鑰和金鑰，可以執行下列指令：「set diag; vserver object-store-server user show -user <username>」。但是，今後這些金鑰將在使用者建立過程中提供，或者可以使用 REST API 呼叫來檢索。
5. 使用步驟 2 中建立的使用者建立物件儲存伺服器群組並授予存取權限。在這個例子中，我們提供了「FullAccess」。
6. 透過將其類型設為「nas」並提供 NFSv3 磁碟區的路徑來建立 NAS 儲存桶。也可以利用 S3 儲存桶來實現此目的。

NetApp儲存設定 – StorageGRID

1. 安裝 storageGRID 軟體。
2. 建立租戶和儲存桶。
3. 建立具有所需權限的使用者。

請查看更多詳細信息 https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

解決方案驗證

解決方案概述

我們針對五個關鍵領域進行了全面的解決方案驗證，具體細節概述如下。每個部分都深入探討了客戶面臨的挑戰、 NetApp提供的解決方案以及隨後為客戶帶來的好處。

1. "在本地使用 Kubernetes 設定 Milvus 集群"客戶面臨的挑戰是獨立擴展儲存和運算、有效的基礎設施管理和資料管理。在本節中，我們詳細介紹了在 Kubernetes 上安裝 Milvus 叢集的過程，並利用NetApp儲存控制器儲存叢集資料和客戶資料。
2. milvus 與 Amazon FSx ONTAP for NetApp ONTAP – 檔案與物件二元性 在本節中，我們將介紹為何ONTAP在雲端部署向量資料庫，以及在NetApp Amazon FSxAmazon FSxONTAPNetAppONTAP（milvus 獨立版）的步驟。
3. "使用NetApp SnapCenter進行向量資料庫保護。"在本節中，我們將深入探討SnapCenter如何保護駐留在ONTAP中的向量資料庫資料和 Milvus 資料。在此範例中，我們利用源自 NFS ONTAP磁碟區（vol1）的 NAS 儲存桶（milvusdbvol1）來儲存客戶數據，並使用單獨的 NFS 磁碟區（vectordbpv）來儲存 Milvus 叢集配置資料。
4. "使用NetApp SnapMirror進行災難復原"在本節中，我們討論災難復原（DR）對於向量資料庫的重要性以及NetApp災難復原產品SnapMirror如何為向量資料庫提供DR解決方案。
5. "性能驗證"在本節中，我們旨在深入研究向量資料庫（例如 Milvus 和 pgvecto.rs）的效能驗證，重點關注它們的儲存效能特徵，例如 I/O 設定檔和 NetApp 儲存控制器在 LLM 生命週期內支援 RAG 和推理工作負載的行為。當這些資料庫與ONTAP儲存解決方案結合時，我們將評估並識別任何效能差異因素。我們的分析將基於關鍵效能指標，例如每秒處理的查詢數（QPS）。

在本地使用 **Kubernetes** 設定 **Milvus** 集群

本節討論針對NetApp的向量資料庫解決方案的 milvus 叢集設定。

在本地使用 **Kubernetes** 設定 **Milvus** 集群

客戶面臨的挑戰是在儲存和運算上獨立擴展、有效的基礎設施管理和資料管理，Kubernetes 和向量資料庫共同構成了管理大數據操作的強大、可擴展的解決方案。 Kubernetes 最佳化資源並管理容器，而向量資料庫則有效率地處理高維度資料和相似性搜尋。這種組合能夠快速處理大型資料集上的複雜查詢，並隨著資料量的增加而無縫擴展，使其成為大數據應用程式和人工智慧工作負載的理想選擇。

1. 在本節中，我們詳細介紹了在 Kubernetes 上安裝 Milvus 叢集的過程，並利用NetApp儲存控制器儲存叢集資料和客戶資料。
2. 要安裝 Milvus 集群，需要持久卷 (PV) 來儲存來自各個 Milvus 集群組件的資料。這些元件包括 etcd（三個實例）、pulsar-bookie-journal（三個實例）、pulsar-bookie-ledgers（三個實例）和 pulsar-zookeeper-data（三個實例）。

> 在 milvus 叢集中，我們可以使用 pulsar 或 kafka 作為支撐 Milvus 叢集可靠儲存以及訊息流發布/訂閱的底層引擎。對於使用 NFS 的 Kafka， NetApp在ONTAP 9.12.1 及更高版本中做出了改進，這些增強功能以及 RHEL 8.7 或 9.1 及更高版本中包含的 NFSv4.1 和 Linux 更改解決了在 NFS 上運行 Kafka 時可能出現的“愚蠢重命名”問題。如果您對使用 NetApp NFS 解決方案運行 Kafka 主題的更多深入資訊感興趣，請查看 -"此連結" 。

3. 我們從NetApp ONTAP建立了一個 NFS 卷，並建立了 12 個持久性卷，每個卷具有 250GB 的儲存空間。儲存大小可能因集群大小而異；例如，我們有另一個集群，其中每個 PV 有 50GB。請參閱下面的 PV YAML 文件之一以了解更多詳細資訊；我們總共有 12 個這樣的文件。在每個檔案中，storageClassName 設定為“default”，並且儲存和路徑對於每個 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. 對每個 PV YAML 檔案執行「kubectl apply」命令來建立持久卷，然後使用「kubectl 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. 為了儲存客戶數據，Milvus 支援物件儲存解決方案，例如 MinIO、Azure Blob 和 S3。在本指南中，我們使用 S3。以下步驟適用於ONTAP S3 和StorageGRID物件儲存。我們使用 Helm 來部署 Milvus 叢集。從 Milvus 下載位置下載設定檔 values.yaml。有關我們在本文檔中使用的 values.yaml 文件，請參閱附錄。
6. 確保每個部分中的“storageClass”設定為“default”，包括日誌、etcd、zookeeper 和 bookkeeper。
7. 在 MinIO 部分，停用 MinIO。
8. 從ONTAP或StorageGRID物件儲存建立 NAS 儲存桶，並使用物件儲存憑證將其包含在外部 S3 中。

```
###################################
# 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，請確保 PersistentVolumeClaim（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. 利用 Helm 和 values.yaml 設定檔安裝並啟動 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 PersistentVolumeClaims (PVC) 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>
```

請確保 Pod 狀態為「正在運作」且如預期運作

13. 測試在 Milvus 和NetApp物件儲存中寫入和讀取資料。
    - 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}}]}
```

基於上述驗證，Kubernetes 與向量資料庫的集成，透過使用NetApp儲存控制器在 Kubernetes 上部署 Milvus 集群，為客戶提供了強大、可擴展且高效的大規模資料操作管理解決方案。此設定為客戶提供了處理高維度資料和快速且有效率地執行複雜查詢的能力，使其成為大數據應用和人工智慧工作負載的理想解決方案。對各種叢集元件使用持久性磁碟區 (PV)，並從NetApp ONTAP建立單一 NFS 卷，可確保最佳資源利用率和資料管理。驗證 PersistentVolumeClaims (PVC) 和 pod 的狀態以及測試資料寫入和讀取的過程為客戶提供了可靠且一致的資料操作的保證。使用ONTAP或StorageGRID物件儲存客戶資料進一步增強了資料的可存取性和安全性。總體而言，這種設定為客戶提供了一種有彈性且高效能的資料管理解決方案，可隨著客戶不斷增長的資料需求而無縫擴展。

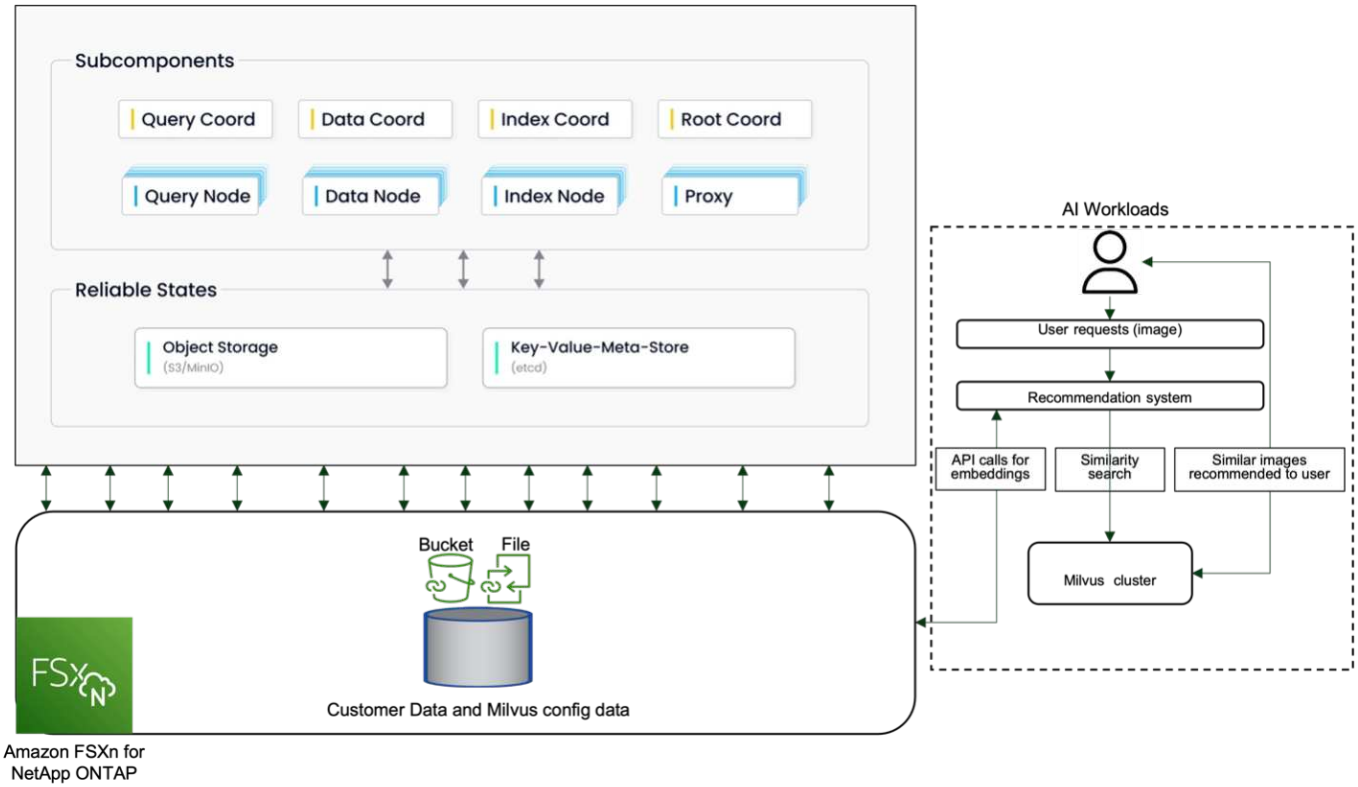
Milvus 與Amazon FSx ONTAP for NetApp ONTAP - 檔案與物件二元性

本節討論使用Amazon FSx ONTAP為NetApp提供向量資料庫解決方案的 milvus 叢集設定。

Milvus 與Amazon FSx ONTAP for NetApp ONTAP – 檔案與物件二元性

在本節中，我們將介紹為什麼需要在雲端部署向量資料庫，以及在 Docker 容器中的Amazon FSx ONTAP for NetApp ONTAP中部署向量資料庫（milvus 獨立版）的步驟。

在雲端部署向量資料庫有幾個顯著的好處，特別是對於需要處理高維度資料和執行相似性搜尋的應用程式。首先，基於雲端的部署提供了可擴展性，允許輕鬆調整資源以適應不斷增長的資料量和查詢負載。這確保資料庫能夠有效地處理增加的需求，同時保持高效能。其次，雲端部署提供了高可用性和災難復原，因為資料可以在不同的地理位置複製，最大限度地降低資料遺失的風險，並確保即使在意外事件期間也能持續提供服務。第三，它具有成本效益，因為您只需為您使用的資源付費，並且可以根據需求擴大或縮小規模，從而無需在硬體上進行大量的前期投資。最後，在雲端部署向量資料庫可以增強協作，因為可以從任何地方存取和共享數據，從而促進基於團隊的工作和數據驅動的決策。請使用Amazon FSx ONTAP for NetApp ONTAP檢查此驗證中所使用的 milvus 獨立架構。



1. 為NetApp ONTAP實例建立Amazon FSx ONTAP ，並記下 VPC、VPC 安全性群組和子網路的詳細資訊。建立 EC2 執行個體時需要此資訊。您可以在此處找到更多詳細資訊 - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. 建立一個 EC2 實例，確保 VPC、安全性群組和子網路與Amazon FSx ONTAP for NetApp ONTAP執行個體的 VPC、安全性群組和子網路相符。
3. 使用指令“apt-get install nfs-common”安裝 nfs-common，並使用“sudo apt-get update”更新套件資訊。
4. 建立一個掛載資料夾並在其上掛載適用於NetApp ONTAP 的Amazon FSx 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 install”安裝 Docker 和 Docker Compose。
6. 根據 docker-compose.yaml 檔案建立 Milvus 集群，該檔案可以從 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 docker-compose.yml 檔案的「volumes」部分中，將NetApp NFS 掛載點對應到對應的 Milvus 容器路徑，具體在 etcd、minio 和 standalone 中。檢查"附錄 D：docker-compose.yml"有關 yml 更改的詳細信息
8. 驗證已安裝的資料夾和檔案。

```
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 docker-compose.yml 檔案的目錄執行「docker-compose up -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Amazon FSx ONTAP for NetApp ONTAP中向量資料庫及其資料的讀寫功能，我們使用了 Python Milvus SDK 和來自 PyMilvus 的範例程式。使用「apt-get install python3-numpy python3-pip」安裝必要的軟體包，並使用「pip3 install pymilvus」安裝 PyMilvus。
12. 驗證向量資料庫中Amazon FSx ONTAP for NetApp ONTAP的資料寫入和讀取操作。

```
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腳本檢查讀取操作。

```
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. 如果客戶想要透過 S3 協定存取（讀取）向量資料庫中測試的 NFS 資料以用於 AI 工作負載，則可以使用簡單的 Python 程式進行驗證。一個例子可以是來自另一個應用程式的圖像的相似性搜索，如本節開頭的圖片中提到的那樣。

```
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#
```

本節有效地展示了客戶如何在 Docker 容器中部署和操作獨立的 Milvus 設置，並利用 Amazon 的NetApp FSx ONTAP進行NetApp ONTAP資料儲存。此設定允許客戶利用向量資料庫的強大功能來處理高維資料和執行複雜查詢，所有這些都可以在可擴展且高效的 Docker 容器環境中完成。透過建立適用於NetApp ONTAP執行個體和符合的 EC2 執行個體的Amazon FSx ONTAP ，客戶可以確保最佳的資源利用率和資料管理。 FSx ONTAP在向量資料庫中資料寫入和讀取操作的成功驗證為客戶提供了可靠、一致的資料操作的保證。此外，透過 S3 協定列出（讀取）來自 AI 工作負載的資料的能力增強了資料可存取性。因此，這項全面的流程為客戶提供了一個強大且高效的解決方案，用於管理他們的大規模資料操作，並利用了 Amazon FSx ONTAP for NetApp ONTAP的功能。

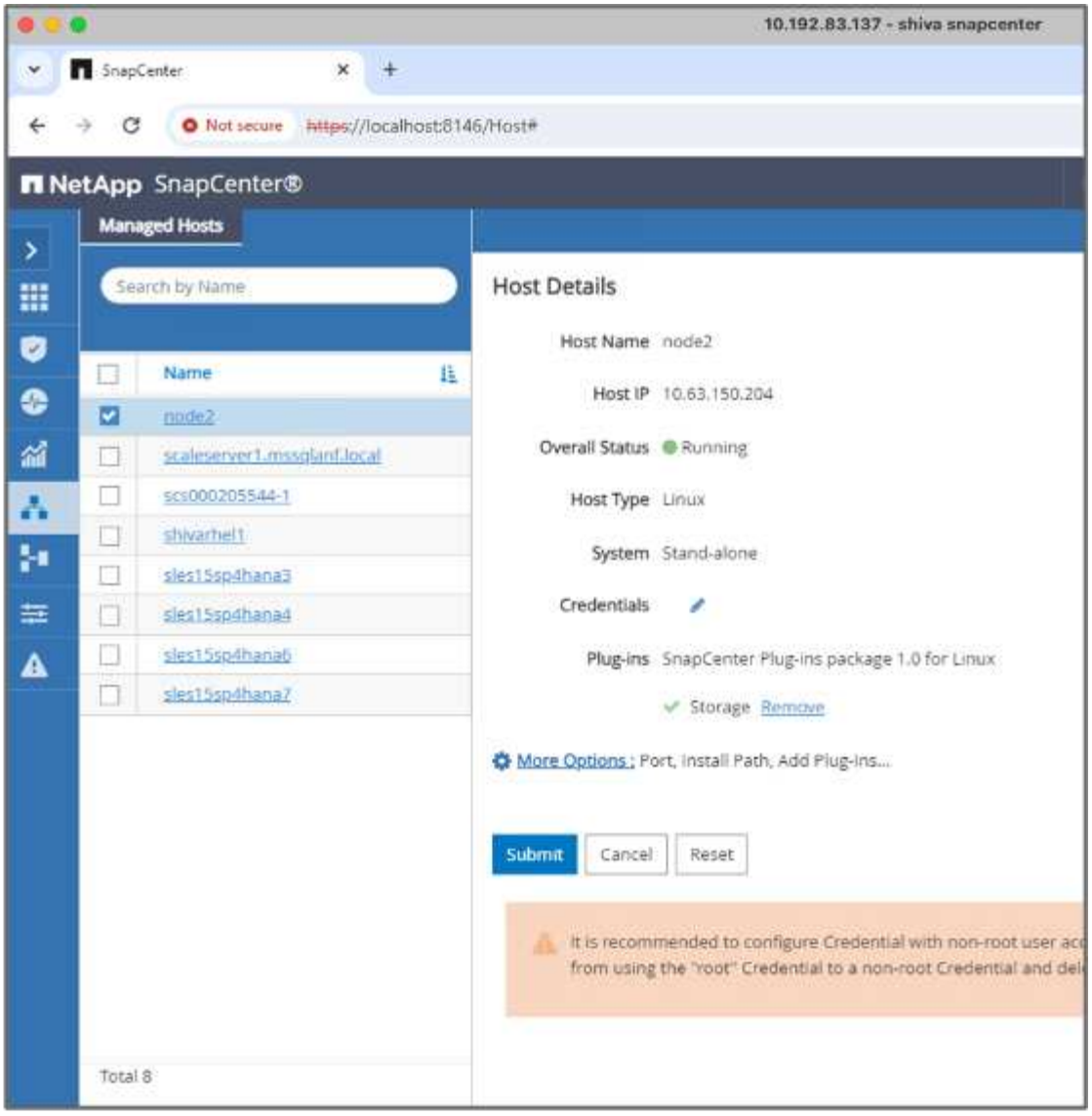
使用**SnapCenter**進行向量資料庫保護

本節介紹如何使用NetApp SnapCenter為向量資料庫提供資料保護。

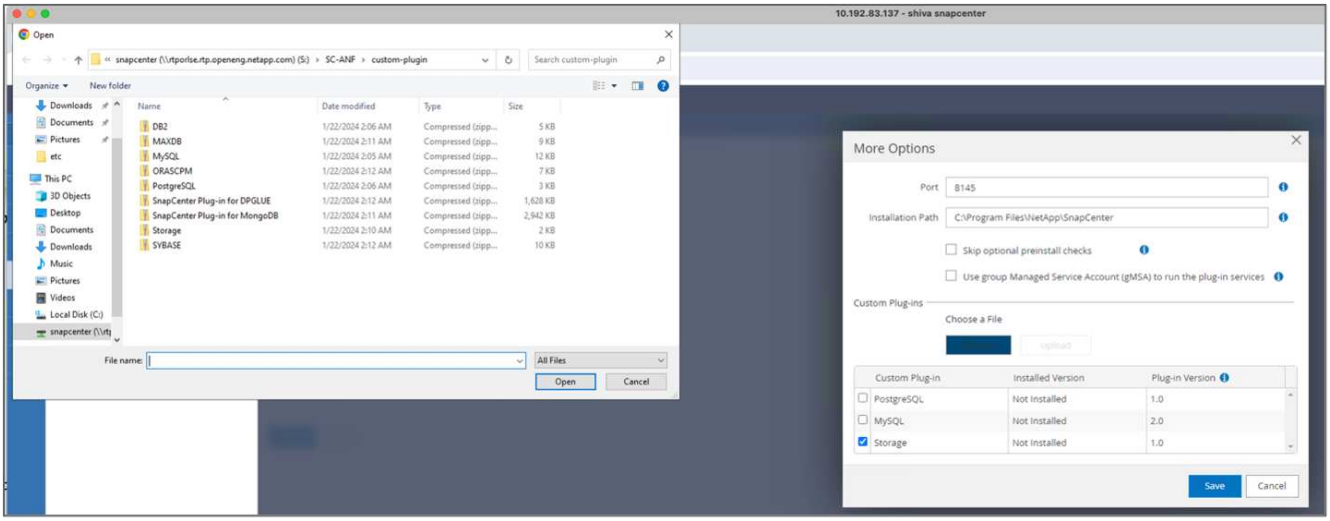
使用**NetApp SnapCenter**進行向量資料庫保護。

例如，在電影製作行業，客戶通常擁有關鍵的嵌入式數據，如視訊和音訊檔案。由於硬碟故障等問題而導致的資料遺失可能會對其營運產生重大影響，甚至可能危及價值數百萬美元的企業。我們曾經遇到寶貴內容遺失的情況，造成嚴重的混亂和經濟損失。因此，確保這些重要數據的安全性和完整性對該行業至關重要。在本節中，我們將深入探討SnapCenter如何保護駐留在ONTAP中的向量資料庫資料和 Milvus 資料。在此範例中，我們使用了從NFS ONTAP磁碟區 (vol1) 衍生的 NAS 儲存桶 (milvusdbvol1) 來儲存客戶數據，並使用了單獨的 NFS 磁碟區 (vectordbpv) 來儲存 Milvus 叢集配置資料。請查看"這裡"Snapcenter 備份工作流程

1. 設定將用於執行SnapCenter指令的主機。

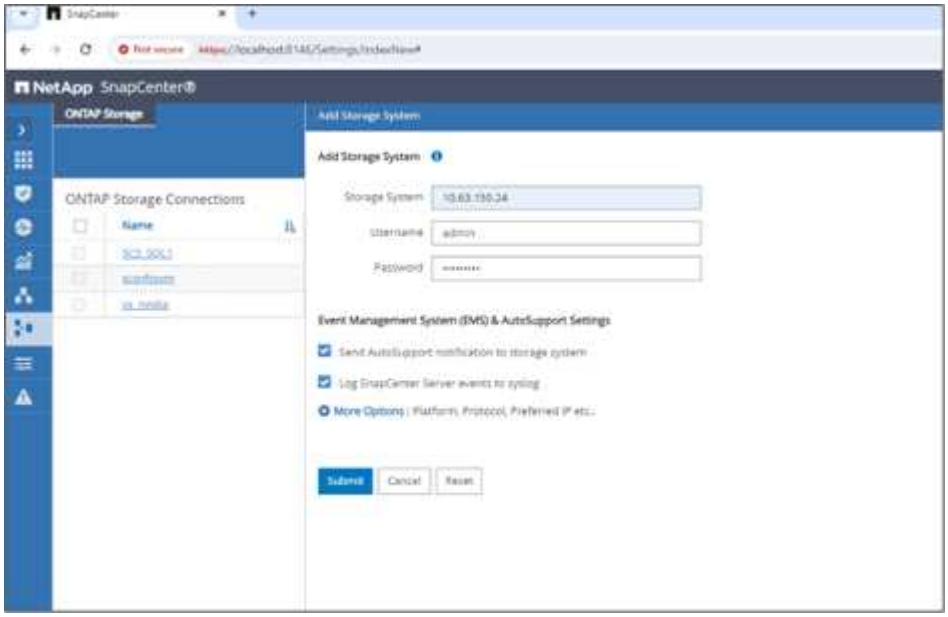


2. 安裝並配置儲存插件。從新增的主機中，選擇「更多選項」。導航到並選擇下載的儲存插件"NetApp自動化商店"。安裝插件並儲存配置。

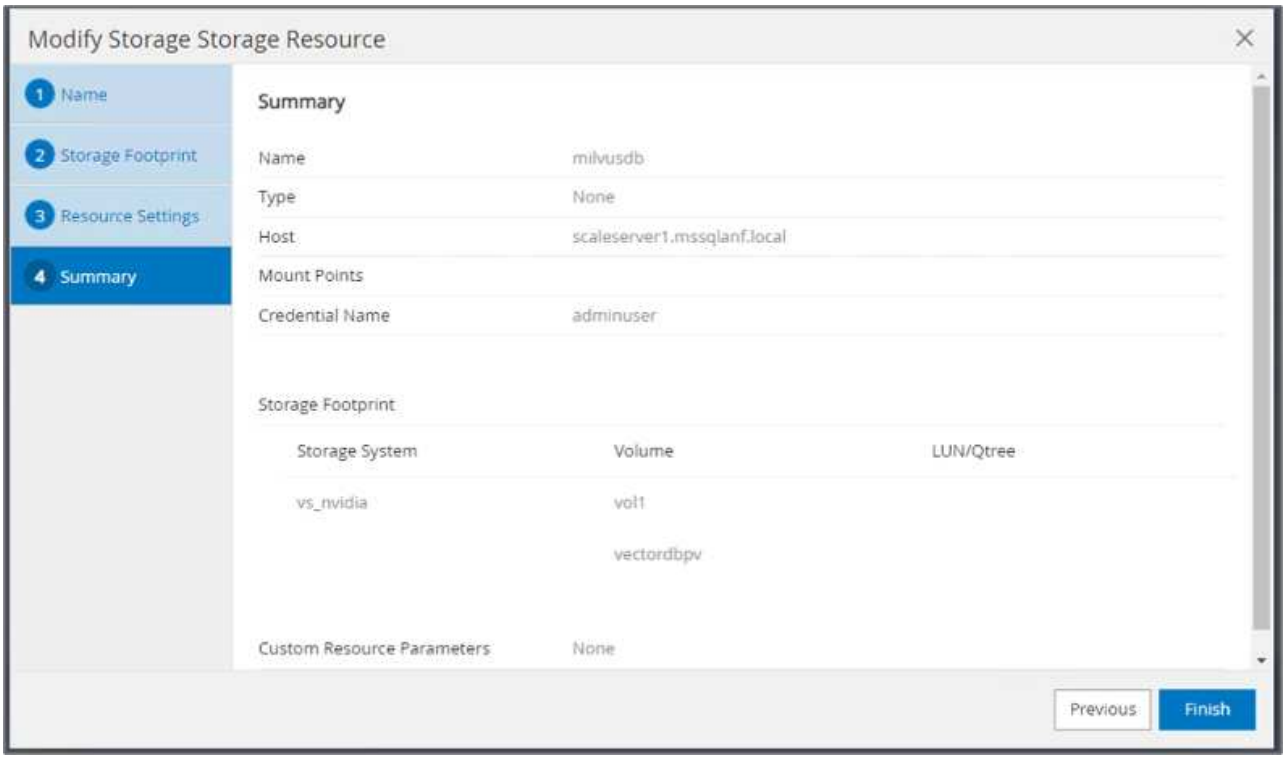


3. 設定儲存系統和磁碟區：在「儲存系統」下新增儲存系統，並選擇SVM（儲存虛擬機器）。在這個例子中，

我們選擇了「vs_nvidia」。



4. 為向量資料庫建立資源，包含備份策略和自訂快照名稱。
   - 使用預設值啟用一致性群組備份，並啟用不具有檔案系統一致性的SnapCenter 。
   - 在儲存佔用空間部分，選擇與向量資料庫客戶資料和 Milvus 叢集資料關聯的磁碟區。在我們的範例中，這些是“vol1”和“vectordbpv”。
   - 建立向量資料庫保護策略，並利用此策略保護向量資料庫資源。



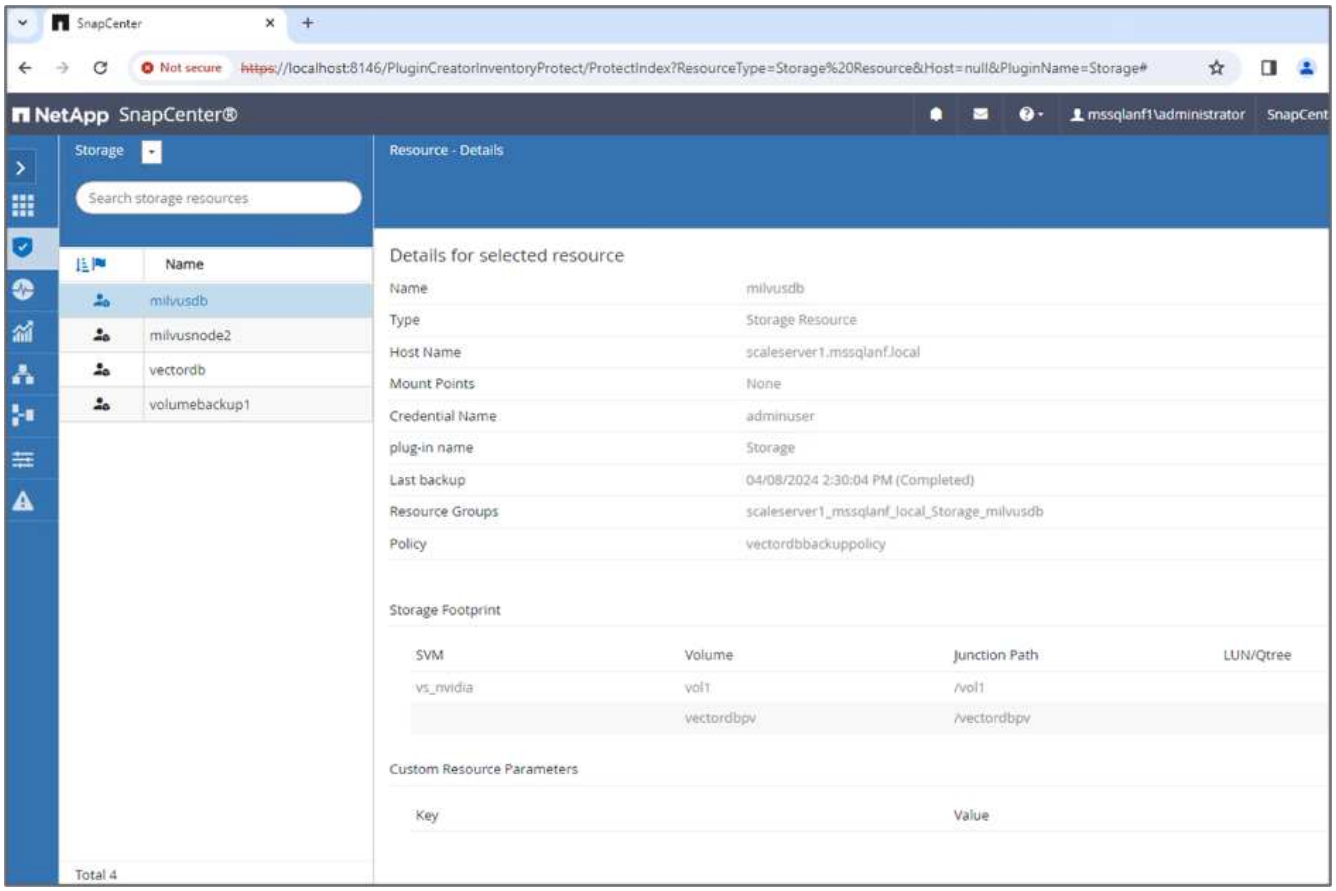
5. 使用 Python 腳本將資料插入 S3 NAS 儲存桶。在我們的案例中，我們修改了 Milvus 提供的備份腳本，即“prepare_data_netapp.py”，並執行“sync”命令從作業系統中刷新資料。

```
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 儲存桶中的資料。在我們的範例中，帶有時間戳記「2024-04-08 21:22」的檔案是由「prepare_data_netapp.py」腳本建立的。

```
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「milvusdb」資源的一致性群組 (CG) 快照啟動備份

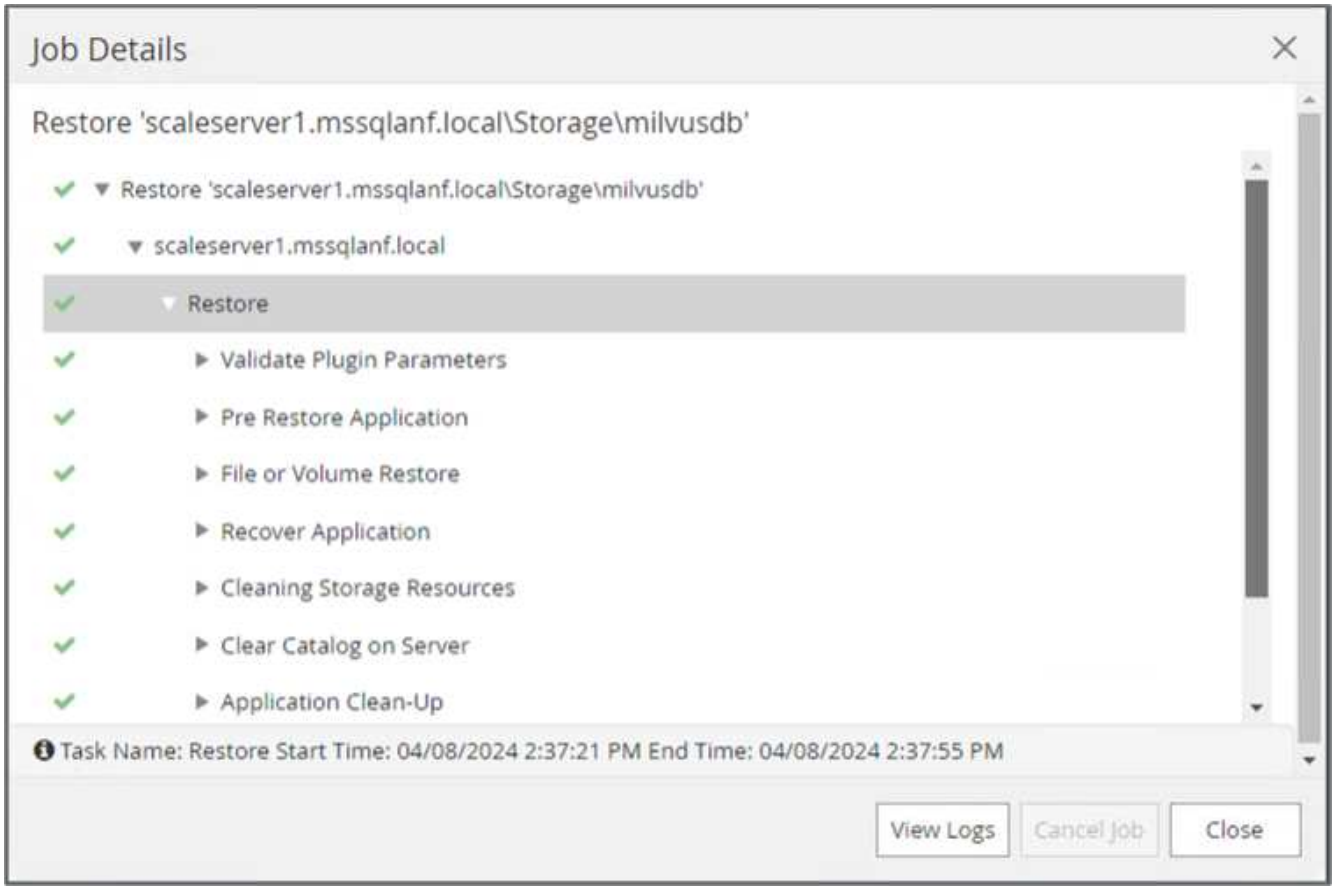


8. 為了測試備份功能，我們在備份過程後新增了一個新表，或從 NFS（S3 NAS 儲存桶）中刪除了一些資料。

   對於此測試，想像一下有人在備份後創建了新的、不必要的或不適當的集合的場景。在這種情況下，我們需要將向量資料庫還原到新增新集合之前的狀態。例如，已插入“hello_milvus_netapp_sc_testnew”和“hello_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. 從上一個快照執行 S3 NAS 儲存桶的完整復原。



10. 使用 Python 腳本驗證來自「hello_milvus_netapp_sc_test」和「hello_milvus_netapp_sc_test2」集合的資料。

```
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. 驗證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:~#
```

總而言之，使用 NetApp 的SnapCenter來保護駐留在ONTAP中的向量資料庫資料和 Milvus 資料可以為客戶帶來顯著的優勢，特別是在資料完整性至關重要的行業，例如電影製作。 SnapCenter 能夠建立一致的備份並執行完整的資料恢復，確保關鍵資料（例如嵌入式視訊和音訊檔案）不會因硬碟故障或其他問題而遺失。這不僅可以防止營運中斷，還可以防止重大財務損失。

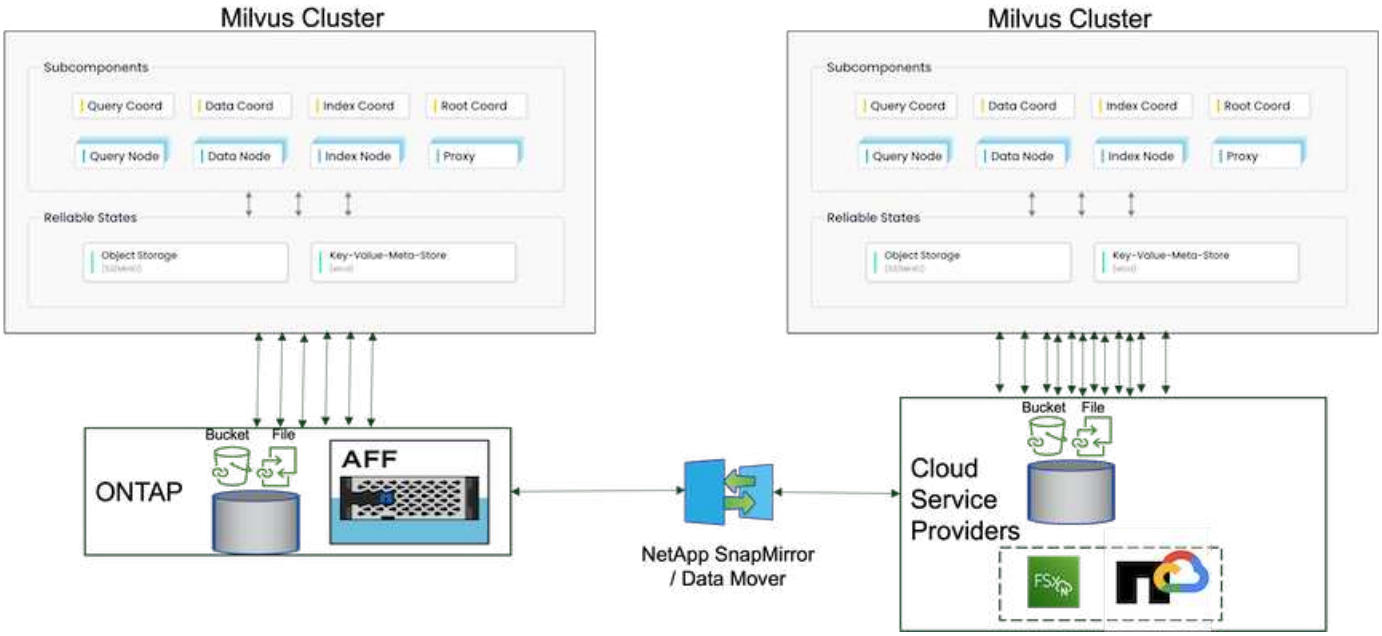
在本節中，我們示範如何配置SnapCenter來保護駐留在ONTAP中的數據，包括主機的設定、儲存插件的安裝和配置，以及使用自訂快照名稱為向量資料庫建立資源。我們也展示如何使用一致性群組快照執行備份並驗證 S3 NAS 儲存桶中的資料。

此外，我們模擬了備份後創建不必要或不適當的集合的情況。在這種情況下，SnapCenter 從先前的快照執行完整復原的能力可確保向量資料庫可以還原到新增集合之前的狀態，從而保持資料庫的完整性。這種將資料恢復到特定時間點的功能對於客戶來說非常寶貴，它為他們提供了保證，確保他們的資料不僅安全，而且得到正確的維護。因此，NetApp 的SnapCenter產品為客戶提供了強大且可靠的資料保護和管理解決方案。

使用NetApp SnapMirror進行災難復原

本節討論使用SnapMirror為NetApp實現向量資料庫解決方案的 DR（災難復原）。

使用NetApp SnapMirror進行災難復原



災難復原對於維護向量資料庫的完整性和可用性至關重要，尤其是考慮到其在管理高維度資料和執行複雜相似性搜尋中的作用。精心規劃和實施的災難復原策略可確保在發生硬體故障、自然災害或網路攻擊等不可預見的事件時資料不會遺失或受到損害。這對於依賴向量資料庫的應用程式尤其重要，因為資料的遺失或損壞可能導致嚴重的營運中斷和財務損失。此外，強大的災難復原計劃還可以最大限度地減少停機時間並允許快速恢復服務，從而確保業務連續性。這是透過NetApp資料複製產品 SnapMirror 跨不同地理位置、定期備份和故障轉移機制實現的。因此，災難復原不僅僅是一種保護措施，而且是負責任、有效率的向量資料庫管理的重要組成部分。

NetApp 的SnapMirror提供從一個NetApp ONTAP儲存控制器到另一個儲存控制器的資料複製，主要用於災難復原 (DR) 和混合解決方案。在向量資料庫的背景下，該工具有助於實現資料在本地和雲端環境之間的平穩過渡。這種轉變無需任何資料轉換或應用程式重構即可實現，從而提高了跨多個平台資料管理的效率和靈活性。

NetApp Hybrid解決方案在向量資料庫場景下可以帶來更多優勢：

1. 可擴展性：NetApp 的混合雲解決方案能夠根據您的需求擴展您的資源。您可以利用本機資源來處理常規、可預測的工作負載，並利用雲端資源（例如Amazon FSx ONTAP for NetApp ONTAP和 Google Cloud NetApp Volume（NetApp Volumes））來應對尖峰時段或意外負載。
2. 成本效益：NetApp 的混合雲模型可讓您透過使用內部資源來處理常規工作負載並僅在需要時支付雲端資源費用來優化成本。這種按需付費模式透過NetApp instaclustr 服務產品可以實現相當高的成本效益。對於本地和主要雲端服務供應商，instaclustr 提供支援和諮詢。
3. 靈活性：NetApp 的混合雲讓您可以靈活地選擇在何處處理資料。例如，您可能選擇在擁有更強大硬體的本地執行複雜的向量操作，而在雲端中執行不太密集的操作。
4. 業務連續性：如果發生災難，將資料保存在NetApp混合雲中可以確保業務連續性。如果您的本地資源受到影響，您可以快速切換到雲端。我們可以利用NetApp SnapMirror將資料從本地端移動到雲端，反之亦然。
5. 創新：NetApp 的混合雲解決方案還可以透過提供對尖端雲端服務和技術的存取來實現更快的創新。 NetApp 在雲端領域的創新，例如Amazon FSx ONTAP 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首選 NAS。

向量資料庫效能驗證

本節重點介紹在向量資料庫上執行的效能驗證。

性能驗證

效能驗證在向量資料庫和儲存系統中都起著至關重要的作用，是確保最佳運作和高效資源利用的關鍵因素。向量資料庫以處理高維度資料和執行相似性搜尋而聞名，需要保持高效能水準才能快速準確地處理複雜查詢。效能驗證有助於識別瓶頸、微調配置並確保系統能夠處理預期負載而不會降低服務品質。同樣，在儲存系統中，效能驗證對於確保資料高效儲存和檢索至關重要，不會出現可能影響整體系統效能的延遲問題或瓶頸。它還有助於對儲存基礎設施的必要升級或變更做出明智的決策。因此，效能驗證是系統管理的重要方面，對維持高服務品質、運作效率和整體系統可靠性有重要貢獻。

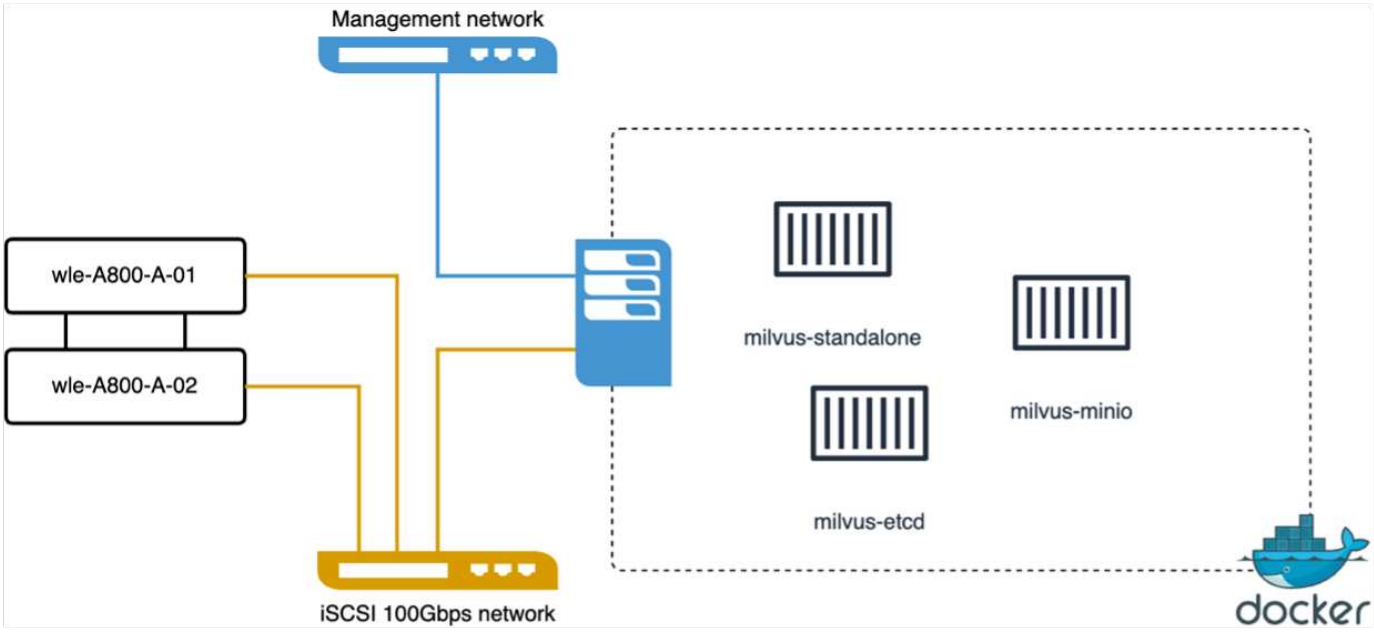
在本節中，我們旨在深入研究向量資料庫（例如 Milvus 和 pgvecto.rs）的效能驗證，重點關注它們的儲存效能特徵，例如 I/O 設定檔和 NetApp 儲存控制器在 LLM 生命週期內支援 RAG 和推理工作負載的行為。當這些資料庫與ONTAP儲存解決方案結合時，我們將評估並識別任何效能差異因素。我們的分析將基於關鍵效能指標，例如每秒處理的查詢數（QPS）。

請檢查下面用於 milvus 和進度的方法。

細節	Milvus（單機和叢集）	Postgres（pgvecto.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	
數據集	LAION 資料集 * 1000 萬個嵌入 * 768 個維度 * 資料集大小約 300GB	
儲存控制器	AFF 800 * 版本 — 9.14.1 * 4 x 100GbE — 用於 milvus，2x 100GbE 用於 postgres * iscsi	

帶有 **Milvus** 獨立叢集的 **VectorDB-Bench**

我們使用vectorDB-Bench在milvus獨立叢集上進行了以下效能驗證。 milvus 獨立叢集的網路和伺服器連線如下。



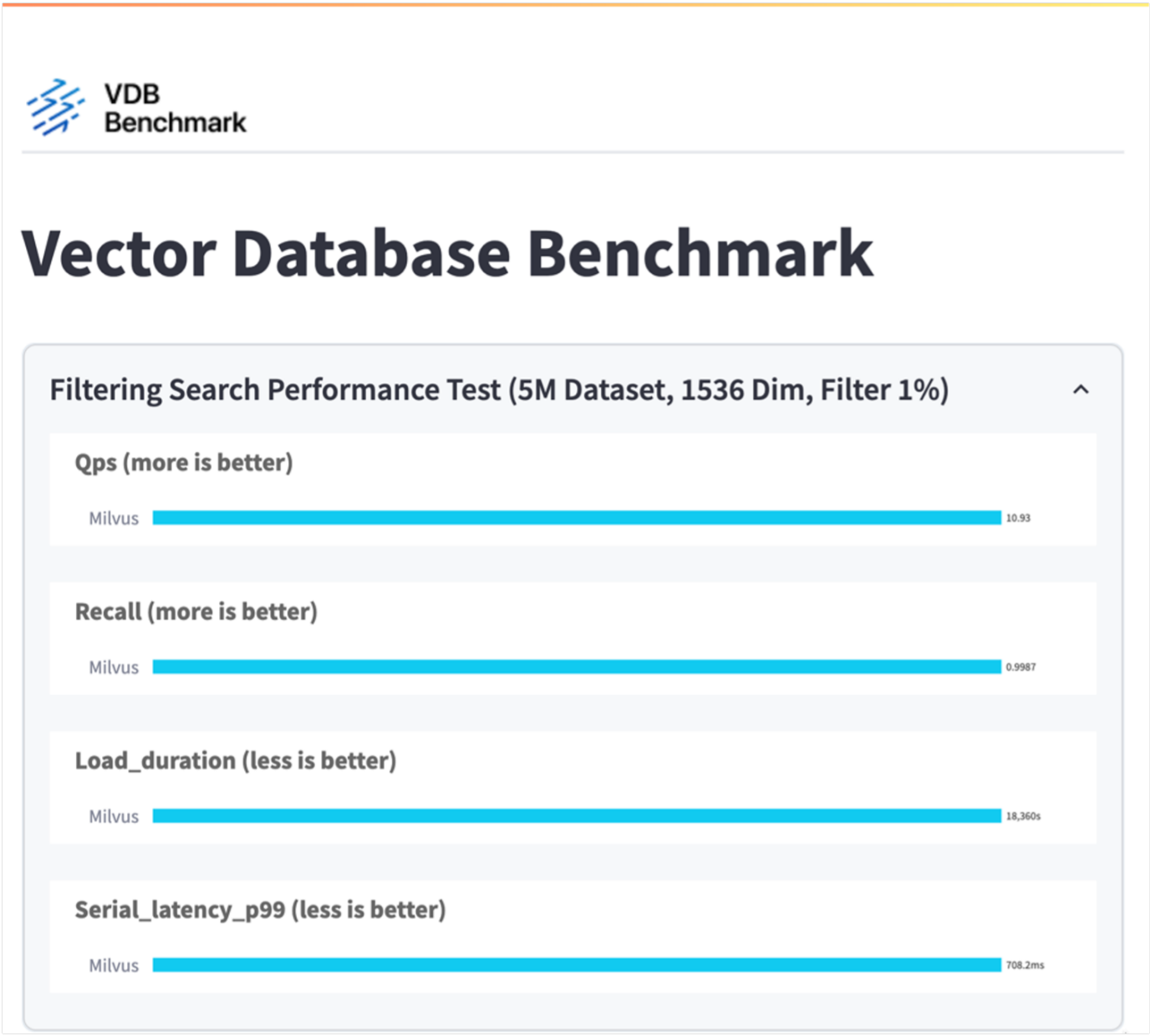
在本節中，我們分享測試 Milvus 獨立資料庫的觀察和結果。。我們選擇 DiskANN 作為這些測試的索引類型。。提取、優化和建立大約 100GB 資料集的索引大約需要 5 個小時。在此持續時間的大部分時間裡，配備 20 個核心（啟用超線程時相當於 40 個 vCPU）的 Milvus 伺服器都以其最大 CPU 容量 100% 運行。我們發現 DiskANN 對於超過系統記憶體大小的大型資料集尤其重要。。在查詢階段，我們觀察到每秒查詢次數 (QPS) 為 10.93，回想率為 0.9987。查詢的第 99 個百分位延遲測量為 708.2 毫秒。

從儲存角度來看，資料庫在攝取、插入後最佳化和索引建立階段發出約 1,000 次操作/秒。在查詢階段，它要求每秒 32,000 次操作。

以下部分介紹儲存效能指標。

工作負載階段	公制	價值
資料擷取和插入後優化	每秒輸入/輸出次數	< 1,000
	延遲	< 400 微秒
	工作量	讀/寫混合，主要是寫入
	IO大小	64KB
詢問	每秒輸入/輸出次數	峰值為32,000
	延遲	< 400 微秒
	工作量	100% 快取讀取
	IO大小	主要為8KB

VectorDB-bench 結果如下。

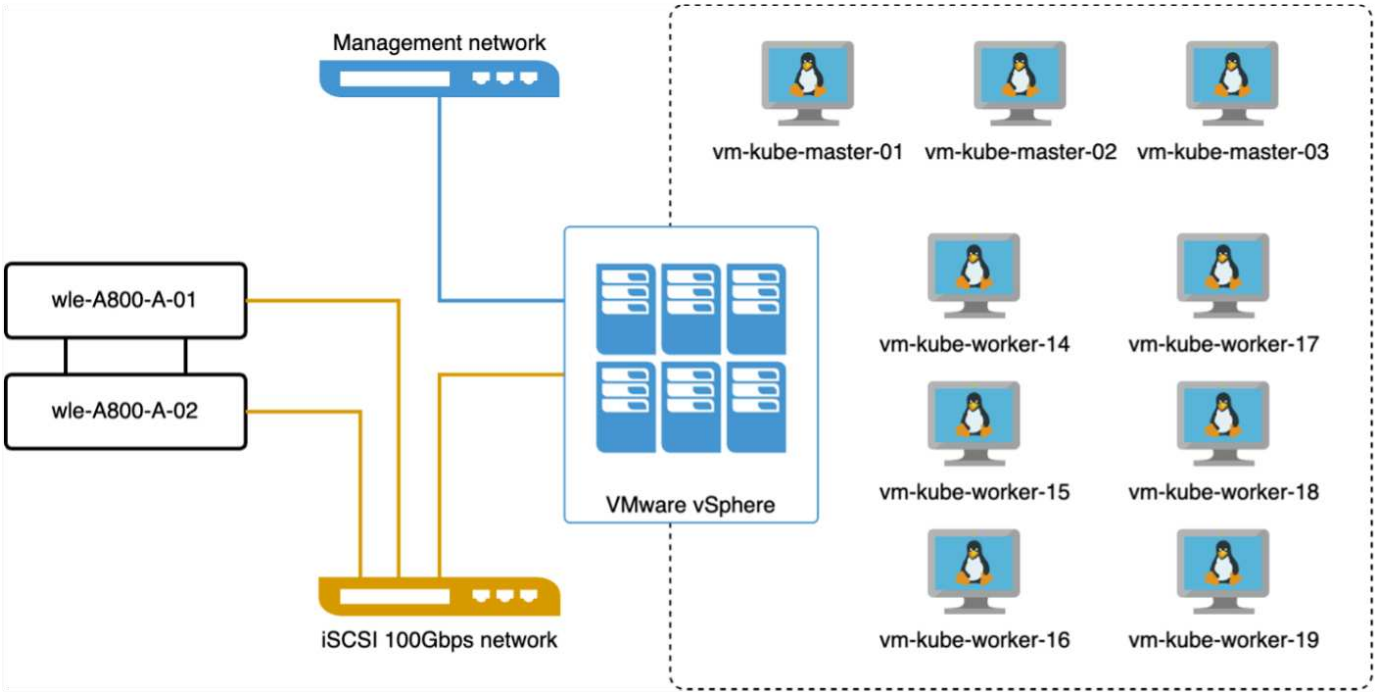


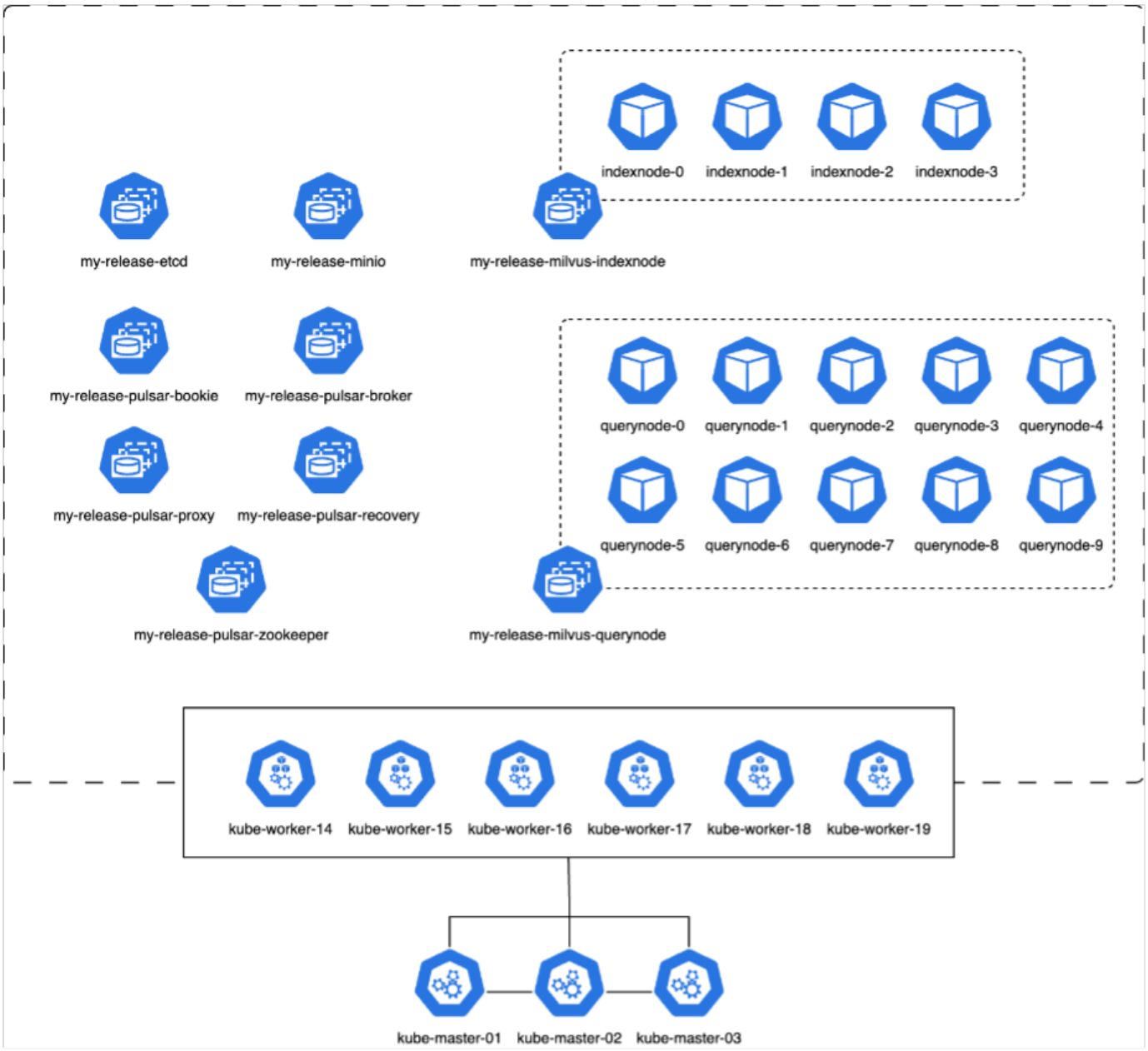
從獨立 Milvus 實例的效能驗證來看，目前的設定不足以支援 500 萬個向量、維度為 1536 的資料集。我們已確定儲存擁有足夠的資源，不會構成系統的瓶頸。

帶有 **milvus** 叢集的 **VectorDB-Bench**

在本節中，我們討論在 Kubernetes 環境中部署 Milvus 叢集。此 Kubernetes 設定建置於 VMware vSphere 部署之上，該部署託管 Kubernetes 主節點和工作節點。

以下部分介紹 VMware vSphere 和 Kubernetes 部署的詳細資訊。


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，我們介紹了測試 Milvus 資料庫的觀察結果和結果。 * 使用的索引類型是 DiskANN。 * 下表比較了在處理 500 萬個向量（維度為 1536）時獨立部署和集群部署的差異。我們觀察到，在叢集部署中，資料擷取和插入後最佳化所需的時間較短。與獨立設定相比，叢集部署中查詢的第 99 個百分位延遲減少了六倍。 * 儘管叢集部署中的每秒查詢數 (QPS) 率較高，但並未達到預期水準。

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%

下圖提供了各種儲存指標的視圖，包括儲存叢集延遲和總 IOPS（每秒輸入/輸出操作）。

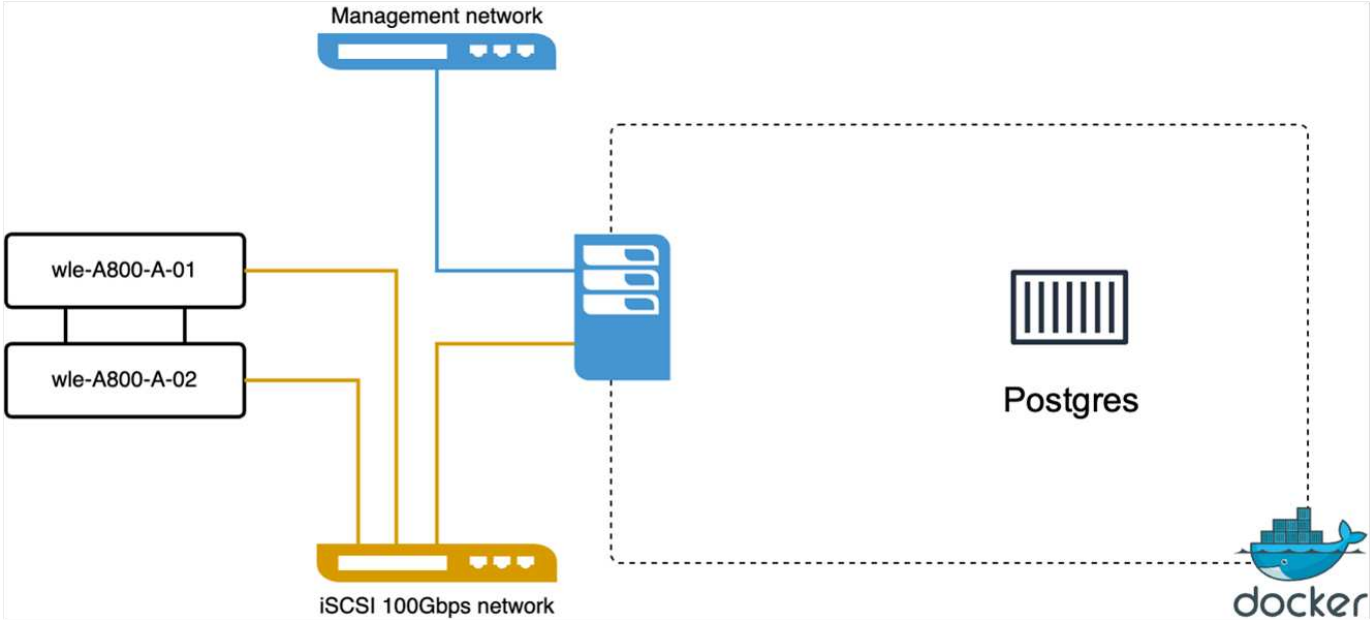
以下部分介紹關鍵的儲存效能指標。

工作負載階段	公制	價值
資料擷取和插入後優化	每秒輸入/輸出次數	< 1,000
	延遲	< 400 微秒
	工作量	讀/寫混合，主要是寫入
	IO大小	64KB
詢問	每秒輸入/輸出次數	峰值為147,000
	延遲	< 400 微秒
	工作量	100% 快取讀取
	IO大小	主要為8KB

基於獨立 Milvus 和 Milvus 叢集的效能驗證，我們展示了儲存 I/O 設定檔的詳細資訊。 * 我們觀察到 I/O 設定檔在獨立部署和叢集部署中保持一致。 * 峰值 IOPS 的觀察到的差異可以歸因於群集部署中的客戶端數量較多。

附有 **Postgres** 的**vectorDB-Bench**（**pgvecto.rs**）

我們使用 VectorDB-Bench 對 PostgreSQL（pgvecto.rs）進行瞭如下操作：PostgreSQL（具體來說，pgvecto.rs）的網路和伺服器連線詳情如下：

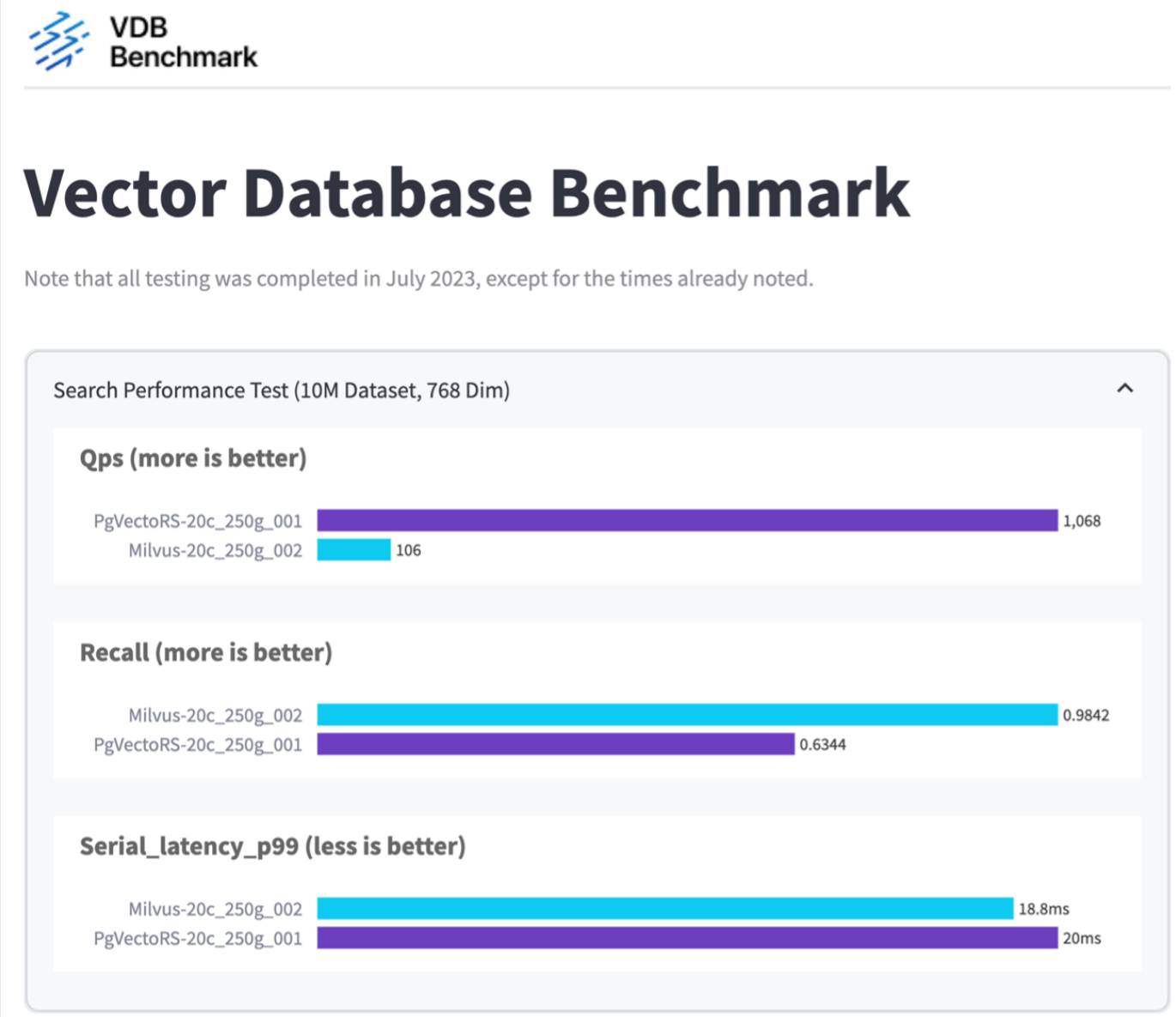


在本節中，我們分享測試 PostgreSQL 資料庫（特別是使用 pgvecto.rs）的觀察和結果。 * 我們選擇 HNSW 作為這些測試的索引類型，因為在測試時，DiskANN 不適用於 pgvecto.rs。 * 在資料擷取階段，我們載入了 Cohere 資料集，該資料集包含 1,000 萬個向量，維度為 768。該過程大約耗時 4.5 小時。 * 在查詢階段，我們觀察到每秒查詢次數 (QPS) 為 1,068，回想率為 0.6344。查詢的第 99 個百分位延遲測量為 20 毫秒。在大部分運行時間內，客戶端 CPU 都以 100% 的容量運作。

下圖提供了各種儲存指標的視圖，包括儲存叢集延遲總 IOPS（每秒輸入/輸出操作）。


```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-metrics.png["此圖顯示輸入/輸出對話框或表示書面內容"]
```

milvus 與 **postgres** 在向量資料庫基準測試上的效能對比



根據我們使用 VectorDBBench 對 Milvus 和 PostgreSQL 進行的效能驗證，我們觀察到以下情況：

- 索引類型：HNSW
- 資料集：包含 768 個維度的 1000 萬個向量

我們發現 pgvecto.rs 的每秒查詢數 (QPS) 達到 1,068，召回率為 0.6344，而 Milvus 的每秒查詢數 (QPS) 達到 106，召回率為 0.9842。

如果您優先考慮查詢的高精度，Milvus 的表現優於 pgvecto.rs，因為它在每個查詢中檢索到更高比例的相關項目。但是，如果每秒查詢次數是一個更關鍵的因素，那麼 pgvecto.rs 就超過了 Milvus。但值得注意的是，透過 pgvecto.rs 檢索的資料品質較低，約 37% 的搜尋結果是不相關的項目。

根據我們的性能驗證得出的觀察結果：

根據我們的性能驗證，我們做出了以下觀察：

在 Milvus 中，I/O 設定檔與 OLTP 工作負載非常相似，例如 Oracle SLOB 中的工作負載。基準測試包括三個階段：資料擷取、後最佳化和查詢。初始階段主要以 64KB 寫入操作為特徵，而查詢階段主要涉及 8KB 讀取。我

們希望ONTAP能夠熟練地處理 Milvus I/O 負載。

PostgreSQL I/O 設定檔不會帶來具有挑戰性的儲存工作負載。鑑於目前正在進行的記憶體實現，我們在查詢階段沒有觀察到任何磁碟 I/O。

DiskANN 成為儲存區分的關鍵技術。它使得向量資料庫搜尋能夠超越系統記憶體邊界進行有效擴展。然而，不太可能透過記憶體中的向量資料庫索引（例如 HNSW）建立儲存效能差異。

另外值得注意的是，當索引類型為 HSNW 時，儲存在查詢階段並不起關鍵作用，而查詢階段是支援 RAG 應用的向量資料庫最重要的操作階段。這裡的含義是儲存效能不會顯著影響這些應用程式的整體效能。

使用 **PostgreSQL** 的 **Instaclustr** 向量資料庫：**pgvector**

本節討論 instaclustr 產品如何與NetApp向量資料庫解決方案中的 postgreSQL 的 pgvector 功能整合的具體細節。

使用 **PostgreSQL** 的 **Instaclustr** 向量資料庫：**pgvector**

在本節中，我們將深入探討 instaclustr 產品如何在 pgvector 功能上與 postgreSQL 整合的具體細節。我們有一個範例「如何使用 PGVector 和 PostgreSQL 來提高 LLM 準確性和效能：嵌入簡介和 PGVector 的作用」。請檢查"部落格"以獲取更多資訊。

向量資料庫用例

本節概述了NetApp向量資料庫解決方案的用例。

向量資料庫用例

在本節中，我們討論兩個用例，例如使用大型語言模型的檢索增強生成和NetApp IT 聊天機器人。

使用大型語言模型 **(LLM)** 進行檢索增強生成 **(RAG)**

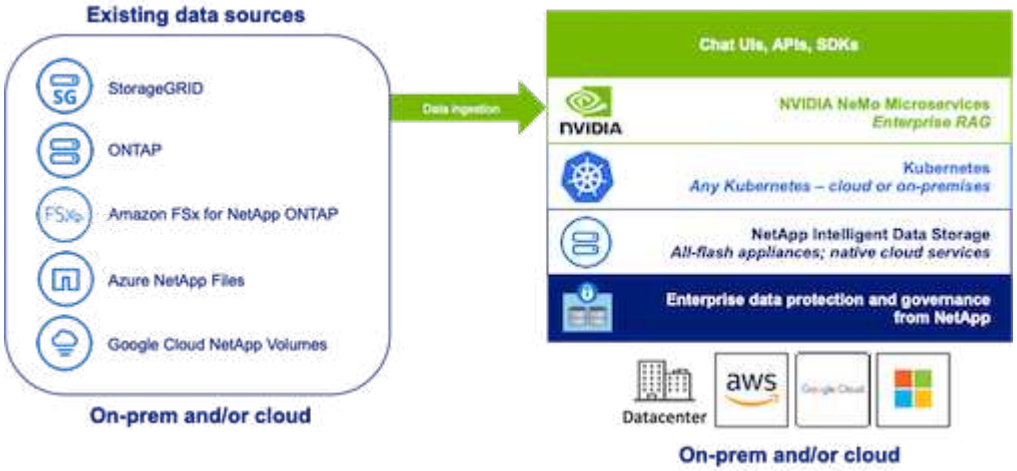
```
Retrieval-augmented generation, or RAG, is a technique for enhancing the
accuracy and reliability of Large Language Models, or LLMs, by augmenting
prompts with facts fetched from external sources. In a traditional RAG
deployment, vector embeddings are generated from an existing dataset and
then stored in a vector database, often referred to as a knowledgebase.
Whenever a user submits a prompt to the LLM, a vector embedding
representation of the prompt is generated, and the vector database is
searched using that embedding as the search query. This search operation
returns similar vectors from the knowledgebase, which are then fed to the
LLM as context alongside the original user prompt. In this way, an LLM can
be augmented with additional information that was not part of its original
training dataset.
```

NVIDIA Enterprise RAG LLM Operator 是企業中實施 RAG 的實用工具。此操作員可用於部署完整的 RAG 管

道。 RAG 管道可以客製化為使用 Milvus 或 pgvecto 作為儲存知識庫嵌入的向量資料庫。有關詳細信息，請參閱文件。

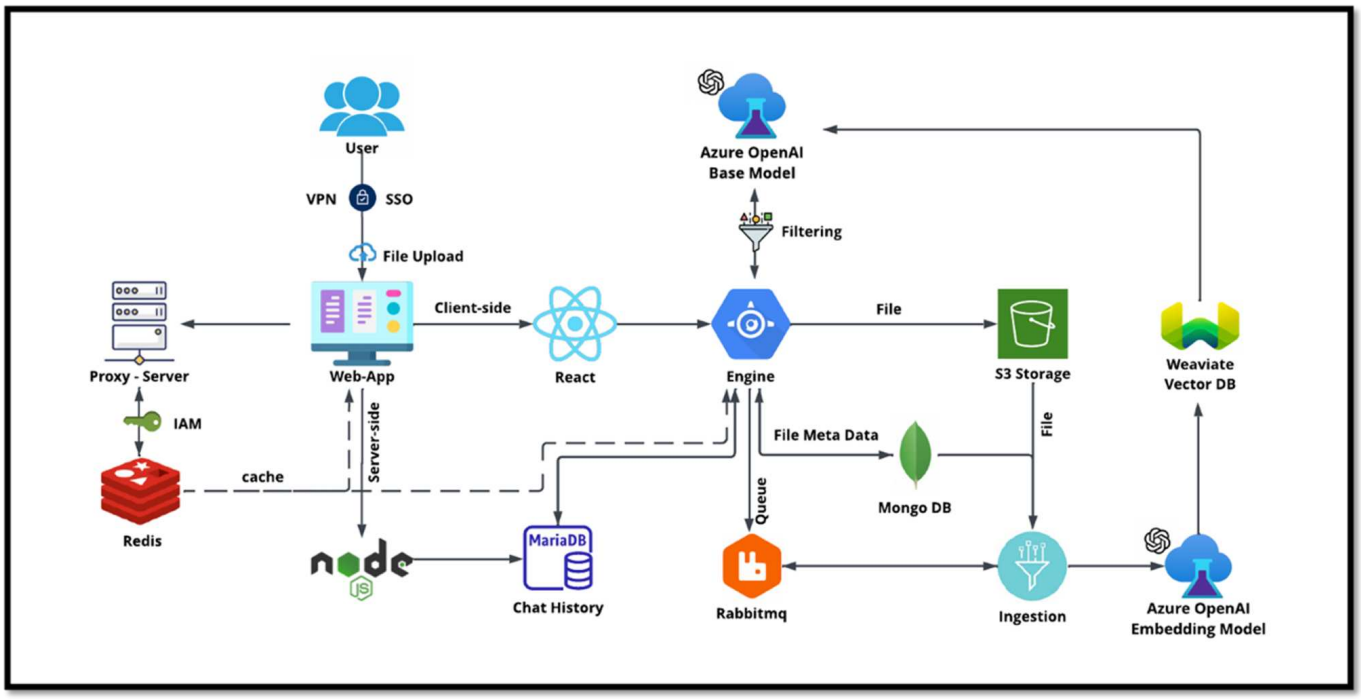
```
NetApp has validated an enterprise RAG architecture powered by the NVIDIA
Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog
post for more information and to see a demo. Figure 1 provides an overview
of this architecture.
```

圖 1) 由NVIDIA NeMo 微服務和NetApp支援的企業 RAG



NetApp IT 聊天機器人用例

NetApp 的聊天機器人是向量資料庫的另一個即時用例。在這種情況下， NetApp Private OpenAI Sandbox 為管理來自 NetApp 內部使用者的查詢提供了一個有效、安全且高效的平台。透過結合嚴格的安全協議、高效的資料管理系統和複雜的人工智慧處理能力，它保證透過 SSO 身份驗證根據組織中使用者的角色和職責為他們提供高品質、精確的回應。這種架構凸顯了融合先進技術以創建以使用者為中心的智慧系統的潛力。



用例可以分為四個主要部分。

使用者身份驗證和驗證：

- 使用者查詢首先經過NetApp單一登入 (SSO) 流程確認使用者的身分。
- 身份驗證成功後，系統會檢查VPN連線以確保資料傳輸的安全。

資料傳輸和處理：

- 一旦 VPN 驗證通過，資料就會透過 NetAIChat 或 NetAICreate Web 應用程式傳送到 MariaDB。 MariaDB 是一個快速且有效率的資料庫系統，用於管理和儲存使用者資料。
- 然後，MariaDB 將資訊傳送到NetApp Azure 實例，該實例將使用者資料連接到 AI 處理單元。

與 **OpenAI** 和內容過濾的交互作用：

- Azure 執行個體將使用者的問題傳送至內容過濾系統。系統清理查詢並準備處理。
- 清理後的輸入隨後被傳送到 Azure OpenAI 基礎模型，該模型根據輸入產生回應。

回應生成和審核：

- 首先檢查基礎模型的回應，以確保其準確性並符合內容標準。
- 檢查通過後，將回應傳回使用者。此流程可確保使用者收到對其查詢的清晰、準確和適當的答案。

結論

本節總結了NetApp的向量資料庫解決方案。

結論

總而言之，本文檔全面概述了在NetApp儲存解決方案上部署和管理向量資料庫（例如 Milvus 和 pgvector）。我們討論了利用NetApp ONTAP和StorageGRID物件儲存的基礎設施指南，並透過檔案和物件儲存驗證了 AWS FSx ONTAP中的 Milvus 資料庫。

我們探索了 NetApp 的檔案物件二元性，證明了它不僅適用於向量資料庫中的數據，也適用於其他應用程式。我們也重點介紹了 NetApp 的企業管理產品SnapCenter如何為向量資料庫資料提供備份、復原和複製功能，確保資料的完整性和可用性。

該文件還深入探討了 NetApp 的混合雲解決方案如何在本地端和雲端環境中提供資料複製和保護，從而提供無縫、安全的資料管理體驗。我們對NetApp ONTAP上 Milvus 和 pgvecto 等向量資料庫的效能驗證提供了見解，並提供了有關其效率和可擴展性的寶貴資訊。

最後，我們討論了兩個生成式 AI 用例：具有 LLM 的 RAG 和 NetApp 的內部 ChatAI。這些實際範例強調了本文檔中概述的概念和實踐的實際應用和好處。總的來說，對於任何希望利用 NetApp 強大的儲存解決方案來管理向量資料庫的人來說，本文檔都是一份全面的指南。

致謝

作者衷心感謝以下貢獻者以及其他提供回饋和評論的人，使本文對NetApp客戶和NetApp領域具有價值。

1. Sathish Thyagarajan， NetApp ONTAP AI 與分析技術行銷工程師
2. NetApp技術行銷工程師 Mike Oglesby
3. NetApp資深總監 AJ Mahajan
4. NetApp工作負載效能工程經理 Joe Scott
5. NetApp Fsx 產品管理資深總監 Puneet Dhawan
6. NetApp FSx 產品團隊資深產品經理 Yuval Kalderon

在哪裡可以找到更多信息

要了解有關本文檔中描述的信息的更多信息，請查看以下文檔和/或網站：

- Milvus 文檔 - https://milvus.io/docs/overview.md
- Milvus 獨立文檔 - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- NetApp產品文檔https://www.netapp.com/support-and-training/documentation/[]
- instaclustr -"installclustr 文檔"

版本歷史記錄

版本	日期	文件版本歷史記錄
版本 1.0	2024年4月	初始版本

附錄 A：Values.yaml

本節提供NetApp向量資料庫解決方案中使用的值的範例 YAML 程式碼。

附錄 A：Values.yaml

```
root@node2:~# cat  values.yaml
## Enable or disable Milvus Cluster mode
cluster:
  enabled: true

image:
  all:
    repository: milvusdb/milvus
    tag: v2.3.4
    pullPolicy: IfNotPresent
    ## Optionally specify an array of imagePullSecrets.
    ## Secrets must be manually created in the namespace.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/pull-
image-private-registry/
    ##
    # pullSecrets:
    #   - myRegistryKeySecretName
  tools:
    repository: milvusdb/milvus-config-tool
    tag: v0.1.2
    pullPolicy: IfNotPresent

# Global node selector
# If set, this will apply to all milvus components
# Individual components can be set to a different node selector
nodeSelector: {}

# Global tolerations
# If set, this will apply to all milvus components
# Individual components can be set to a different tolerations
tolerations: []

# Global affinity
# If set, this will apply to all milvus components
# Individual components can be set to a different affinity
affinity: {}

# Global labels and annotations
# If set, this will apply to all milvus components
labels: {}
annotations: {}

# Extra configs for milvus.yaml
# If set, this config will merge into milvus.yaml
```

```
# Please follow the config structure in the milvus.yaml
# at https://github.com/milvus-io/milvus/blob/master/configs/milvus.yaml
# Note: this config will be the top priority which will override the
config
# in the image and helm chart.
extraConfigFiles:
  user.yaml: |+
    #    For example enable rest http for milvus proxy
    #    proxy:
    #      http:
    #        enabled: true
    ##  Enable tlsMode and set the tls cert and key
    #  tls:
    #    serverPemPath: /etc/milvus/certs/tls.crt
    #    serverKeyPath: /etc/milvus/certs/tls.key
    #   common:
    #     security:
    #       tlsMode: 1

## Expose the Milvus service to be accessed from outside the cluster
(LoadBalancer service).
## or access it from within the cluster (ClusterIP service). Set the
service type and the port to serve it.
## ref: http://kubernetes.io/docs/user-guide/services/
##
service:
  type: ClusterIP
  port: 19530
  portName: milvus
  nodePort: ""
  annotations: {}
  labels: {}

  ## List of IP addresses at which the Milvus service is available
  ## Ref: https://kubernetes.io/docs/user-guide/services/#external-ips
  ##
  externalIPs: []
  #   - externalIp1

  # LoadBalancerSourcesRange is a list of allowed CIDR values, which are
combined with ServicePort to
  # set allowed inbound rules on the security group assigned to the master
load balancer
  loadBalancerSourceRanges:
  - 0.0.0.0/0
  # Optionally assign a known public LB IP
```

```
  # loadBalancerIP: 1.2.3.4

ingress:
  enabled: false
  annotations:
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: GRPC
    nginx.ingress.kubernetes.io/listen-ports-ssl: '[19530]'
    nginx.ingress.kubernetes.io/proxy-body-size: 4m
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  labels: {}
  rules:
    - host: "milvus-example.local"
      path: "/"
      pathType: "Prefix"
    # - host: "milvus-example2.local"
    #   path: "/otherpath"
    #   pathType: "Prefix"
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - milvus-example.local

serviceAccount:
  create: false
  name:
  annotations:
  labels:

metrics:
  enabled: true

  serviceMonitor:
    # Set this to `true` to create ServiceMonitor for Prometheus operator
    enabled: false
    interval: "30s"
    scrapeTimeout: "10s"
    # Additional labels that can be used so ServiceMonitor will be
discovered by Prometheus
    additionalLabels: {}

livenessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 30
```

```
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

readinessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

log:
  level: "info"
  file:
    maxSize: 300    # MB
    maxAge: 10    # day
    maxBackups: 20
  format: "text"    # text/json

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Logs Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ## ReadWriteMany access mode required for milvus cluster.
      ##
      storageClass: default
      accessModes: ReadWriteMany
      size: 10Gi
      subPath: ""

## Heaptrack traces all memory allocations and annotates these events with
stack traces.
```

```
## See more: https://github.com/KDE/heaptrack
## Enable heaptrack in production is not recommended.
heaptrack:
  image:
    repository: milvusdb/heaptrack
    tag: v0.1.0
    pullPolicy: IfNotPresent

standalone:
  replicas: 1  # Run standalone mode with replication disabled
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  disk:
    enabled: true
    size:
      enabled: false  # Enable local storage size limit
  profiling:
    enabled: false  # Enable live profiling

  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
    mountPath: "/var/lib/milvus"
    ## If true, alertmanager will create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: true
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
```

```
      ##   set, choosing the default provisioner.
      ##
      storageClass:
      accessModes: ReadWriteOnce
      size: 50Gi
      subPath: ""

proxy:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  http:
    enabled: true  # whether to enable http rest server
    debugMode:
      enabled: false
  # Mount a TLS secret into proxy pod
  tls:
    enabled: false
## when enabling proxy.tls, all items below should be uncommented and the
key and crt values should be populated.
#    enabled: true
#    secretName: milvus-tls
## expecting base64 encoded values here: i.e. $(cat tls.crt | base64 -w 0)
and $(cat tls.key | base64 -w 0)
#    key: LS0tLS1CRUdJTiBQU--REDUCT
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

rootCoordinator:
  enabled: true
```

```
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Root Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for root coordinator

  service:
    port: 53100
    annotations: {}
    labels: {}
    clusterIP: ""

queryCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Query Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for query coordinator

  service:
    port: 19531
    annotations: {}
    labels: {}
    clusterIP: ""
```

```
queryNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  disk:
    enabled: true  # Enable querynode load disk index, and search on disk
index
    size:
      enabled: false  # Enable local storage size limit
  profiling:
    enabled: false  # Enable live profiling

indexCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1   # Run Index Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for index coordinator

  service:
    port: 31000
    annotations: {}
    labels: {}
    clusterIP: ""
```

```
indexNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  disk:
    enabled: true  # Enable index node build disk vector index
    size:
      enabled: false  # Enable local storage size limit

dataCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Data Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for data coordinator

  service:
    port: 13333
    annotations: {}
    labels: {}
```

```
    clusterIP: ""

dataNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling

## mixCoordinator contains all coord
## If you want to use mixcoord, enable this and disable all of other
coords
mixCoordinator:
  enabled: false
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Mixture Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for Mixture coordinator

  service:
    annotations: {}
    labels: {}
    clusterIP: ""

attu:
  enabled: false
```

```
  name: attu
  image:
    repository: zilliz/attu
    tag: v2.2.8
    pullPolicy: IfNotPresent
  service:
    annotations: {}
    labels: {}
    type: ClusterIP
    port: 3000
    # loadBalancerIP: ""
  resources: {}
  podLabels: {}
  ingress:
    enabled: false
    annotations: {}
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    labels: {}
    hosts:
      - milvus-attu.local
    tls: []
    #  - secretName: chart-attu-tls
    #    hosts:
    #      - milvus-attu.local


## Configuration values for the minio dependency
## ref: https://github.com/minio/charts/blob/master/README.md
##

minio:
  enabled: false
  name: minio
  mode: distributed
  image:
    tag: "RELEASE.2023-03-20T20-16-18Z"
    pullPolicy: IfNotPresent
  accessKey: minioadmin
  secretKey: minioadmin
  existingSecret: ""
  bucketName: "milvus-bucket"
  rootPath: file
  useIAM: false
  iamEndpoint: ""
  region: ""
```

```
  useVirtualHost: false
  podDisruptionBudget:
    enabled: false
  resources:
    requests:
      memory: 2Gi

  gcsgateway:
    enabled: false
    replicas: 1
    gcsKeyJson: "/etc/credentials/gcs_key.json"
    projectId: ""

  service:
    type: ClusterIP
    port: 9000

  persistence:
    enabled: true
    existingClaim: ""
    storageClass:
    accessMode: ReadWriteOnce
    size: 500Gi

  livenessProbe:
    enabled: true
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 5

  readinessProbe:
    enabled: true
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 1
    successThreshold: 1
    failureThreshold: 5

  startupProbe:
    enabled: true
    initialDelaySeconds: 0
    periodSeconds: 10
    timeoutSeconds: 5
    successThreshold: 1
```

```
    failureThreshold: 60

## Configuration values for the etcd dependency
## ref: https://artifacthub.io/packages/helm/bitnami/etcd
##

etcd:
  enabled: true
  name: etcd
  replicaCount: 3
  pdb:
    create: false
  image:
    repository: "milvusdb/etcd"
    tag: "3.5.5-r2"
    pullPolicy: IfNotPresent

  service:
    type: ClusterIP
    port: 2379
    peerPort: 2380

  auth:
    rbac:
      enabled: false

  persistence:
    enabled: true
    storageClass: default
    accessMode: ReadWriteOnce
    size: 10Gi

  ## Change default timeout periods to mitigate zoobie probe process
  livenessProbe:
    enabled: true
    timeoutSeconds: 10

  readinessProbe:
    enabled: true
    periodSeconds: 20
    timeoutSeconds: 10

  ## Enable auto compaction
  ## compaction by every 1000 revision
  ##
  autoCompactionMode: revision
```

```
  autoCompactionRetention: "1000"

  ## Increase default quota to 4G
  ##
  extraEnvVars:
  - name: ETCD_QUOTA_BACKEND_BYTES
    value: "4294967296"
  - name: ETCD_HEARTBEAT_INTERVAL
    value: "500"
  - name: ETCD_ELECTION_TIMEOUT
    value: "2500"

## Configuration values for the pulsar dependency
## ref: https://github.com/apache/pulsar-helm-chart
##

pulsar:
  enabled: true
  name: pulsar

  fullnameOverride: ""
  persistence: true

  maxMessageSize: "5242880"  # 5 * 1024 * 1024 Bytes, Maximum size of each
message in pulsar.

  rbac:
    enabled: false
    psp: false
    limit_to_namespace: true

  affinity:
    anti_affinity: false

## enableAntiAffinity: no

  components:
    zookeeper: true
    bookkeeper: true
    # bookkeeper - autorecovery
    autorecovery: true
    broker: true
    functions: false
    proxy: true
    toolset: false
    pulsar_manager: false
```

```
  monitoring:
    prometheus: false
    grafana: false
    node_exporter: false
    alert_manager: false

  images:
    broker:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    autorecovery:
      repository: apachepulsar/pulsar
      tag: 2.8.2
      pullPolicy: IfNotPresent
    zookeeper:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    bookie:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    proxy:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    pulsar_manager:
      repository: apachepulsar/pulsar-manager
      pullPolicy: IfNotPresent
      tag: v0.1.0

  zookeeper:
    volumes:
      persistence: true
      data:
        name: data
        size: 20Gi   #SSD Required
        storageClassName: default
    resources:
      requests:
        memory: 1024Mi
        cpu: 0.3
    configData:
      PULSAR_MEM: >
        -Xms1024m
```

```
        -Xmx1024m
      PULSAR_GC: >
         -Dcom.sun.management.jmxremote
         -Djute.maxbuffer=10485760
         -XX:+ParallelRefProcEnabled
         -XX:+UnlockExperimentalVMOptions
         -XX:+DoEscapeAnalysis
         -XX:+DisableExplicitGC
         -XX:+PerfDisableSharedMem
         -Dzookeeper.forceSync=no
    pdb:
      usePolicy: false

  bookkeeper:
    replicaCount: 3
    volumes:
      persistence: true
      journal:
        name: journal
        size: 100Gi
        storageClassName: default
      ledgers:
        name: ledgers
        size: 200Gi
        storageClassName: default
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+UseG1GC -XX:MaxGCPauseMillis=10
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
```

```
      -XX:+ExitOnOutOfMemoryError
      -XX:+PerfDisableSharedMem
      -XX:+PrintGCDetails
    nettyMaxFrameSizeBytes: "104867840"
  pdb:
    usePolicy: false

broker:
  component: broker
  podMonitor:
    enabled: false
  replicaCount: 1
  resources:
    requests:
      memory: 4096Mi
      cpu: 1.5
  configData:
    PULSAR_MEM: >
      -Xms4096m
      -Xmx4096m
      -XX:MaxDirectMemorySize=8192m
    PULSAR_GC: >
      -Dio.netty.leakDetectionLevel=disabled
      -Dio.netty.recycler.linkCapacity=1024
      -XX:+ParallelRefProcEnabled
      -XX:+UnlockExperimentalVMOptions
      -XX:+DoEscapeAnalysis
      -XX:ParallelGCThreads=32
      -XX:ConcGCThreads=32
      -XX:G1NewSizePercent=50
      -XX:+DisableExplicitGC
      -XX:-ResizePLAB
      -XX:+ExitOnOutOfMemoryError
    maxMessageSize: "104857600"
    defaultRetentionTimeInMinutes: "10080"
    defaultRetentionSizeInMB: "-1"
    backlogQuotaDefaultLimitGB: "8"
    ttlDurationDefaultInSeconds: "259200"
    subscriptionExpirationTimeMinutes: "3"
    backlogQuotaDefaultRetentionPolicy: producer_exception
  pdb:
    usePolicy: false

autorecovery:
  resources:
    requests:
```

```
        memory: 512Mi
        cpu: 1

  proxy:
    replicaCount: 1
    podMonitor:
      enabled: false
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    service:
      type: ClusterIP
    ports:
      pulsar: 6650
    configData:
      PULSAR_MEM: >
        -Xms2048m -Xmx2048m
      PULSAR_GC: >
        -XX:MaxDirectMemorySize=2048m
      httpNumThreads: "100"
    pdb:
      usePolicy: false

  pulsar_manager:
    service:
      type: ClusterIP

  pulsar_metadata:
    component: pulsar-init
    image:
      # the image used for running `pulsar-cluster-initialize` job
      repository: apachepulsar/pulsar
      tag: 2.8.2


## Configuration values for the kafka dependency
## ref: https://artifacthub.io/packages/helm/bitnami/kafka
##

kafka:
  enabled: false
  name: kafka
  replicaCount: 3
  image:
    repository: bitnami/kafka
```

```
    tag: 3.1.0-debian-10-r52
  ## Increase graceful termination for kafka graceful shutdown
  terminationGracePeriodSeconds: "90"
  pdb:
    create: false

  ## Enable startup probe to prevent pod restart during recovering
  startupProbe:
    enabled: true

  ## Kafka Java Heap size
  heapOpts: "-Xmx4096m -Xms4096m"
  maxMessageBytes: _10485760
  defaultReplicationFactor: 3
  offsetsTopicReplicationFactor: 3
  ## Only enable time based log retention
  logRetentionHours: 168
  logRetentionBytes: _-1
  extraEnvVars:
  - name: KAFKA_CFG_MAX_PARTITION_FETCH_BYTES
    value: "5242880"
  - name: KAFKA_CFG_MAX_REQUEST_SIZE
    value: "5242880"
  - name: KAFKA_CFG_REPLICA_FETCH_MAX_BYTES
    value: "10485760"
  - name: KAFKA_CFG_FETCH_MESSAGE_MAX_BYTES
    value: "5242880"
  - name: KAFKA_CFG_LOG_ROLL_HOURS
    value: "24"

  persistence:
    enabled: true
    storageClass:
    accessMode: ReadWriteOnce
    size: 300Gi

  metrics:
    ## Prometheus Kafka exporter: exposes complimentary metrics to JMX
exporter
    kafka:
      enabled: false
      image:
        repository: bitnami/kafka-exporter
        tag: 1.4.2-debian-10-r182

    ## Prometheus JMX exporter: exposes the majority of Kafkas metrics
```

```
    jmx:
      enabled: false
      image:
        repository: bitnami/jmx-exporter
        tag: 0.16.1-debian-10-r245

    ## To enable serviceMonitor, you must enable either kafka exporter or
jmx exporter.
    ## And you can enable them both
    serviceMonitor:
      enabled: false

  service:
    type: ClusterIP
    ports:
      client: 9092

  zookeeper:
    enabled: true
    replicaCount: 3

###################################
# 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

###################################
# GCS Gateway
# - these configs are only used when `minio.gcsgateway.enabled` is true
###################################
externalGcs:
  bucketName: ""
```

```
###################################
# External etcd
# - these configs are only used when `externalEtcd.enabled` is true
###################################
externalEtcd:
  enabled: false
  ## the endpoints of the external etcd
  ##
  endpoints:
    - localhost:2379

###################################
# External pulsar
# - these configs are only used when `externalPulsar.enabled` is true
###################################
externalPulsar:
  enabled: false
  host: localhost
  port: 6650
  maxMessageSize: "5242880"  # 5 * 1024 * 1024 Bytes, Maximum size of each
message in pulsar.
  tenant: public
  namespace: default
  authPlugin: ""
  authParams: ""

###################################
# External kafka
# - these configs are only used when `externalKafka.enabled` is true
###################################
externalKafka:
  enabled: false
  brokerList: localhost:9092
  securityProtocol: SASL_SSL
  sasl:
    mechanisms: PLAIN
    username: ""
    password: ""
root@node2:~#
```

附錄 B：prepare_data_netapp_new.py

本節提供用於準備向量資料庫資料的Python腳本範例。

附錄 B：prepare_data_netapp_new.py

```
root@node2:~# cat prepare_data_netapp_new.py
# hello_milvus.py demonstrates the basic operations of PyMilvus, a Python
SDK of Milvus.
# 1. connect to Milvus
# 2. create collection
# 3. insert data
# 4. create index
# 5. search, query, and hybrid search on entities
# 6. delete entities by PK
# 7. drop collection
import time
import os
import numpy as np
from pymilvus import (
    connections,
    utility,
    FieldSchema, CollectionSchema, DataType,
    Collection,
)

fmt = "\n=== {:30} ===\n"
search_latency_fmt = "search latency = {:.4f}s"
#num_entities, dim = 3000, 8
num_entities, dim = 3000, 16

#################################################################################
#######
# 1. connect to Milvus
# Add a new connection alias `default` for Milvus server in
`localhost:19530`
# Actually the "default" alias is a buildin in PyMilvus.
# If the address of Milvus is the same as `localhost:19530`, you can omit
all
# parameters and call the method as: `connections.connect()`.
#
# Note: the `using` parameter of the following methods is default to
"default".
print(fmt.format("start connecting to Milvus"))

host = os.environ.get('MILVUS_HOST')
if host == None:
    host = "localhost"
print(fmt.format(f"Milvus host: {host}"))
#connections.connect("default", host=host, port="19530")
```

```
connections.connect("default", host=host, port="27017")

has = utility.has_collection("hello_milvus_ntapnew_update2_sc")
print(f"Does collection hello_milvus_ntapnew_update2_sc exist in Milvus:
{has}")

#drop the collection
print(fmt.format(f"Drop collection - hello_milvus_ntapnew_update2_sc"))
utility.drop_collection("hello_milvus_ntapnew_update2_sc")
#drop the collection
print(fmt.format(f"Drop collection - hello_milvus_ntapnew_update2_sc2"))
utility.drop_collection("hello_milvus_ntapnew_update2_sc2")

###############################################################################
#######
# 2. create collection
# We're going to create a collection with 3 fields.
# +-+------------+------------+------------------
+-----------------------------+
# | | field name | field type | other attributes |       field description
|
# +-+------------+------------+------------------
+-----------------------------+
# |1|    "pk"    |    Int64   |  is_primary=True |      "primary field"
|
# | |            |            |   auto_id=False  |
|
# +-+------------+------------+------------------
+-----------------------------+
# |2|  "random"  |    Double  |                  |      "a double field"
|
# +-+------------+------------+------------------
+-----------------------------+
# |3|"embeddings"| FloatVector|     dim=8        |  "float vector with dim
8"   |
# +-+------------+------------+------------------
+-----------------------------+
fields = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=False),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]

schema = CollectionSchema(fields, "hello_milvus_ntapnew_update2_sc")
```

```
print(fmt.format("Create collection `hello_milvus_ntapnew_update2_sc`"))
hello_milvus_ntapnew_update2_sc = Collection(
"hello_milvus_ntapnew_update2_sc", schema, consistency_level="Strong")

################################################################################
######
# 3. insert data
# We are going to insert 3000 rows of data into
`hello_milvus_ntapnew_update2_sc`
# Data to be inserted must be organized in fields.
#
# The insert() method returns:
# - either automatically generated primary keys by Milvus if auto_id=True
in the schema;
# - or the existing primary key field from the entities if auto_id=False
in the schema.

print(fmt.format("Start inserting entities"))
rng = np.random.default_rng(seed=19530)
entities = [
    # provide the pk field because `auto_id` is set to False
    [i for i in range(num_entities)],
    rng.random(num_entities).tolist(),  # field random, only supports list
    [str(i) for i in range(num_entities)],
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

insert_result = hello_milvus_ntapnew_update2_sc.insert(entities)
hello_milvus_ntapnew_update2_sc.flush()
print(f"Number of entities in hello_milvus_ntapnew_update2_sc:
{hello_milvus_ntapnew_update2_sc.num_entities}")  # check the num_entites

# create another collection
fields2 = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=True),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]


schema2 = CollectionSchema(fields2, "hello_milvus_ntapnew_update2_sc2")
```

```
print(fmt.format("Create collection `hello_milvus_ntapnew_update2_sc2`"))
hello_milvus_ntapnew_update2_sc2 = Collection(
"hello_milvus_ntapnew_update2_sc2", schema2, consistency_level="Strong")

entities2 = [
    rng.random(num_entities).tolist(),  # field random, only supports list
    [str(i) for i in range(num_entities)],
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

insert_result2 = hello_milvus_ntapnew_update2_sc2.insert(entities2)
hello_milvus_ntapnew_update2_sc2.flush()
insert_result2 = hello_milvus_ntapnew_update2_sc2.insert(entities2)
hello_milvus_ntapnew_update2_sc2.flush()

# index_params = {"index_type": "IVF_FLAT", "params": {"nlist": 128},
"metric_type": "L2"}
# hello_milvus_ntapnew_update2_sc.create_index("embeddings", index_params)
#
hello_milvus_ntapnew_update2_sc2.create_index(field_name="var",index_name=
"scalar_index")

# index_params2 = {"index_type": "Trie"}
# hello_milvus_ntapnew_update2_sc2.create_index("var", index_params2)

print(f"Number of entities in hello_milvus_ntapnew_update2_sc2:
{hello_milvus_ntapnew_update2_sc2.num_entities}")  # check the num_entites

root@node2:~#
```

附錄C：verify_data_netapp.py

本節包含一個範例 Python 腳本，可用於驗證NetApp向量資料庫解決方案中的向量資料庫。

附錄C：verify_data_netapp.py

```
root@node2:~# cat verify_data_netapp.py
import time
import os
import numpy as np
from pymilvus import (
    connections,
    utility,
```

```
    FieldSchema, CollectionSchema, DataType,
    Collection,
)

fmt = "\n=== {:30} ===\n"
search_latency_fmt = "search latency = {:.4f}s"
num_entities, dim = 3000, 16
rng = np.random.default_rng(seed=19530)
entities = [
    # provide the pk field because `auto_id` is set to False
    [i for i in range(num_entities)],
    rng.random(num_entities).tolist(),  # field random, only supports list
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

################################################################################
######
# 1. get recovered collection hello_milvus_ntapnew_update2_sc
print(fmt.format("start connecting to Milvus"))
host = os.environ.get('MILVUS_HOST')
if host == None:
    host = "localhost"
print(fmt.format(f"Milvus host: {host}"))
#connections.connect("default", host=host, port="19530")
connections.connect("default", host=host, port="27017")

recover_collections = ["hello_milvus_ntapnew_update2_sc",
"hello_milvus_ntapnew_update2_sc2"]

for recover_collection_name in recover_collections:
    has = utility.has_collection(recover_collection_name)
    print(f"Does collection {recover_collection_name} exist in Milvus:
{has}")
    recover_collection = Collection(recover_collection_name)
    print(recover_collection.schema)
    recover_collection.flush()

    print(f"Number of entities in Milvus: {recover_collection_name} :
{recover_collection.num_entities}")  # check the num_entites


################################################################################
######
    # 4. create index
    # We are going to create an IVF_FLAT index for
```

```
hello_milvus_ntapnew_update2_sc collection.
    # create_index() can only be applied to `FloatVector` and
`BinaryVector` fields.
    print(fmt.format("Start Creating index IVF_FLAT"))
    index = {
        "index_type": "IVF_FLAT",
        "metric_type": "L2",
        "params": {"nlist": 128},
    }

    recover_collection.create_index("embeddings", index)


################################################################################
######
    # 5. search, query, and hybrid search
    # After data were inserted into Milvus and indexed, you can perform:
    # - search based on vector similarity
    # - query based on scalar filtering(boolean, int, etc.)
    # - hybrid search based on vector similarity and scalar filtering.
    #

    # Before conducting a search or a query, you need to load the data in
`hello_milvus` into memory.
    print(fmt.format("Start loading"))
    recover_collection.load()

    #
--------------------------------------------------------------------------------
---
    # search based on vector similarity
    print(fmt.format("Start searching based on vector similarity"))
    vectors_to_search = entities[-1][-2:]
    search_params = {
        "metric_type": "L2",
        "params": {"nprobe": 10},
    }

    start_time = time.time()
    result = recover_collection.search(vectors_to_search, "embeddings",
search_params, limit=3, output_fields=["random"])
    end_time = time.time()

    for hits in result:
        for hit in hits:
            print(f"hit: {hit}, random field: {hit.entity.get('random')}")
```

```
    print(search_latency_fmt.format(end_time - start_time))

    #
-------------------------------------------------------------------------
---
    # query based on scalar filtering(boolean, int, etc.)
    print(fmt.format("Start querying with `random > 0.5`"))

    start_time = time.time()
    result = recover_collection.query(expr="random > 0.5", output_fields=
["random", "embeddings"])
    end_time = time.time()

    print(f"query result:\n-{result[0]}")
    print(search_latency_fmt.format(end_time - start_time))

    #
-------------------------------------------------------------------------
---
    # hybrid search
    print(fmt.format("Start hybrid searching with `random > 0.5`"))

    start_time = time.time()
    result = recover_collection.search(vectors_to_search, "embeddings",
search_params, limit=3, expr="random > 0.5", output_fields=["random"])
    end_time = time.time()

    for hits in result:
        for hit in hits:
            print(f"hit: {hit}, random field: {hit.entity.get('random')}")
    print(search_latency_fmt.format(end_time - start_time))


###########################################################################
#####
    # 7. drop collection
    # Finally, drop the hello_milvus, hello_milvus_ntapnew_update2_sc
collection

    #print(fmt.format(f"Drop collection {recover_collection_name}"))
    #utility.drop_collection(recover_collection_name)

root@node2:~#
```

附錄 D：docker-compose.yml

本節包含NetApp向量資料庫解決方案的範例 YAML 程式碼。

附錄 D：docker-compose.yml

```
version: '3.5'

services:
  etcd:
    container_name: milvus-etcd
    image: quay.io/coreos/etcd:v3.5.5
    environment:
      - ETCD_AUTO_COMPACTION_MODE=revision
      - ETCD_AUTO_COMPACTION_RETENTION=1000
      - ETCD_QUOTA_BACKEND_BYTES=4294967296
      - ETCD_SNAPSHOT_COUNT=50000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/etcd:/etcd
    command: etcd -advertise-client-urls=http://127.0.0.1:2379 -listen
-client-urls http://0.0.0.0:2379 --data-dir /etcd
    healthcheck:
      test: ["CMD", "etcdctl", "endpoint", "health"]
      interval: 30s
      timeout: 20s
      retries: 3

  minio:
    container_name: milvus-minio
    image: minio/minio:RELEASE.2023-03-20T20-16-18Z
    environment:
      MINIO_ACCESS_KEY: minioadmin
      MINIO_SECRET_KEY: minioadmin
    ports:
      - "9001:9001"
      - "9000:9000"
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/minio:/minio_data
    command: minio server /minio_data --console-address ":9001"
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost:9000/minio/health/live"]
      interval: 30s
      timeout: 20s
      retries: 3
```

```
  standalone:
    container_name: milvus-standalone
    image: milvusdb/milvus:v2.4.0-rc.1
    command: ["milvus", "run", "standalone"]
    security_opt:
    - seccomp:unconfined
    environment:
      ETCD_ENDPOINTS: etcd:2379
      MINIO_ADDRESS: minio:9000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/milvus:/var/lib/milvus
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:9091/healthz"]
      interval: 30s
      start_period: 90s
      timeout: 20s
      retries: 3
    ports:
      - "19530:19530"
      - "9091:9091"
    depends_on:
      - "etcd"
      - "minio"

networks:
  default:
    name: milvus
```