NetApp 的向量資料庫解決方案

NetApp Solutions

NetApp

May 03, 2024

目錄

NetApp 向量資料庫解決方案 . . . 1
- 簡介 . . . 1
- 解決方案總覽 . . . 2
- 向量資料庫 . . . 3
- 技術需求 . . . 5
- 部署程序 . . . 6
- 解決方案總覽 . . . 8
- 使用 PostgreSQL 的 Instaclustr 向量資料庫： pgvector . . . 42
- 向量資料庫使用案例 . . . 42
- 結論 . . . 44
- 附錄 A ： values.yaml . . . 45
- 附錄 B ： prepare_data_netapp_new.py . . . 66
- 附錄 C ： verify_data_netapp.py . . . 70
- 附錄 D ：泊塢視窗 - 組合 .yml . . . 73

NetApp 向量資料庫解決方案

NetApp 的 Kartheyan Nagalingam 和 Rodrigo Nascimento

本文件使用 NetApp 的儲存解決方案、徹底探索向量資料庫（例如 Milvus ）的部署與管理、以及 pgvecto 開放原始碼 PostgreSQL 延伸。其中詳述使用 NetApp ONTAP 和 StorageGRID 物件儲存設備的基礎架構準則、並驗證 AWS FSX for NetApp ONTAP 中的 Milvus 資料庫應用程式。本文件說明 NetApp 的檔案物件雙重性、以及其用於支援向量嵌入的向量資料庫和應用程式的公用程式。它強調 NetApp 企業管理產品 SnapCenter 的功能、可為向量資料庫提供備份與還原功能、確保資料完整性與可用度。本文件進一步深入探討 NetApp 的混合雲解決方案、討論其在內部部署和雲端環境中的資料複寫與保護角色。其中包括對 NetApp ONTAP 上向量資料庫效能驗證的深入見解、最後以兩個泛用 AI 的實際使用案例做為結論：使用 LLM 和 NetApp 內部 ChatAI 。本文檔是利用 NetApp 的儲存解決方案來管理向量資料庫的完整指南。

參考架構著重於下列項目：

1. "簡介"
2. "解決方案總覽"
3. "向量資料庫"
4. "技術需求"
5. "部署程序"
6. "解決方案驗證總覽"
    - "在內部部署使用 Kubernetes 進行 Milvus 叢集設定"
    - "Milvus 搭配 Amazon FSxN for NetApp ONTAP –檔案和物件雙重性"
    - "使用 NetApp SnapCenter 保護向量資料庫。"
    - "使用 NetApp SnapMirror 進行災難恢復"
    - "效能驗證"
7. "使用 PostgreSQL 的 Instaclustr 向量資料庫： pgvector"
8. "向量資料庫使用案例"
9. "結論"
10. "附錄 A ： values.yaml"
11. "附錄 B ： prepare_data_netapp_new.py"
12. "附錄 C ： verify_data_netapp.py"
13. "附錄 D ：泊塢視窗 - 組合 .yml"

簡介

簡介

向量資料庫可有效因應大型語言模型（ LMS ）和泛用人工智慧（ AI ）中的語義搜尋複雜度所設計的挑戰。與傳統的資料管理系統不同、向量資料庫能夠處理及搜尋各種類型的資料、包括影像、影片、文字、音訊、 使用資料本身的內容、而非標籤或標籤、以及其他形式的非結構化資料。

Relational Database Management Systems （ RDBMS ）的侷限性已詳加記錄、尤其是在 AI 應用程式中、它們與高維度資料呈現和非結構化資料的爭用。RDBMS 通常需要耗時且容易出錯的程序、將資料整合至更容易管理的結構、導致搜尋延遲和效率不彰。然而、向量資料庫的設計旨在規避這些問題、提供更有效率且更準確的解決方案來管理及搜尋複雜的高維度資料、進而協助 AI 應用程式的發展。

本文件為目前使用或計畫使用向量資料庫的客戶提供全方位指南、詳述在 NetApp ONTAP 、 NetApp StorageGRID 、 Amazon FSxN for NetApp ONTAP 和 SnapCenter 等平台上使用向量資料庫的最佳實務做法。此處提供的內容涵蓋多個主題：

- NetApp 儲存設備透過 NetApp ONTAP 和 StorageGRID 物件儲存設備提供的向量資料庫基礎架構準則、例如 Milvus 。
- 透過檔案和物件存放區、驗證 AWS FSX for NetApp ONTAP 中的 Milvus 資料庫。
- 深入瞭解 NetApp 的檔案物件雙重性、展現其在向量資料庫及其他應用程式中的資料實用度。
- 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與還原功能。
- NetApp 的混合雲如何在內部部署和雲端環境中提供資料複寫與保護。
- 深入瞭解 NetApp ONTAP 上的向量資料庫（例如 Milvus 和 pgvector ）的效能驗證。
- 兩種特定使用案例：擷取使用大語言模型（ LLM ）的擴增世代（ RAG ）、以及 NetApp IT 團隊的 ChatAI 、提供所概述概念和實務做法的實際範例。

解決方案總覽

解決方案總覽

此解決方案展現 NetApp 為解決向量資料庫客戶所面臨的挑戰而帶來的獨特優勢與功能。藉由運用 NetApp 雲端解決方案 NetApp ONTAP 、 StorageGRID 和 SnapCenter 、客戶可以為業務營運增加重大價值。這些工具不僅能解決現有的問題、也能提升效率和生產力、進而促進整體業務成長。

為何選擇 **NetApp** ？

- NetApp 的產品（例如 ONTAP 和 StorageGRID ）可分離儲存和運算、根據特定需求提供最佳的資源使用率。這種靈活度可讓客戶使用 NetApp 儲存解決方案來擴充儲存設備。
- 藉由運用 NetApp 的儲存控制器、客戶可以使用 NFS 和 S3 傳輸協定、將資料有效地提供給向量資料庫。這些通訊協定可協助客戶儲存資料並管理向量資料庫索引、不需要透過檔案和物件方法存取多個資料複本。
- NetApp ONTAP 在 AWS 、 Azure 和 Google Cloud 等頂尖雲端服務供應商之間、提供 NAS 和物件儲存的原生支援。這種廣泛的相容性可確保無縫整合、實現客戶資料的移動性、全球存取能力、災難恢復、動態擴充性和高效能。
- 有了 NetApp 強大的資料管理功能、客戶就能放心、因為他們的資料受到妥善保護、不會受到潛在風險和威脅的影響。NetApp 將資料安全性列為優先考量、讓客戶在安全性與完整性方面安心無虞。

向量資料庫

向量資料庫

向量資料庫是一種特殊類型的資料庫、專門設計用來處理、索引及搜尋非結構化資料、並使用機器學習模型的內嵌資料。它不會以傳統的表格格式來組織資料、而是將資料排列成高維向量、也稱為向量嵌入式。這種獨特的結構可讓資料庫更有效率且更準確地處理複雜的多維資料。

向量資料庫的關鍵功能之一、就是使用泛型 AI 來執行分析。這包括相似性搜尋、資料庫可識別資料點、例如指定的輸入、以及異常狀況偵測、藉此找出與正常情況大不相同的資料點。

此外、向量資料庫非常適合處理時間資料或時間戳記資料。這類資料會依序、針對指定 IT 系統內的所有其他事件、提供有關「發生什麼事」及發生時間的資訊。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瞭解一段時間內事件的應用程式特別有用。

ML 和 **AI** 向量資料庫的優點：

- 高維度搜尋：向量資料庫在管理和擷取高維度資料方面表現優異、這通常是在 AI 和 ML 應用程式中產生的。
- 擴充性：可有效擴充以處理大量資料、支援 AI 和 ML 專案的成長與擴充。
- 靈活度：向量資料庫提供高度靈活度、可容納多種資料類型和結構。
- 效能：提供高效能的資料管理與擷取功能、對於 AI 和 ML 作業的速度與效率而言非常重要。
- 可自訂的索引：向量資料庫提供可自訂的索引選項、可根據特定需求來最佳化資料組織和擷取。

向量資料庫和使用案例。

本節提供各種向量資料庫及其使用案例詳細資料。

Faiss 和 ScaNN

這些程式庫是向量搜尋領域中的重要工具。這些程式庫提供的功能有助於管理和搜尋向量資料、讓它們在這個專門的資料管理領域中擁有寶貴的資源。

彈性搜尋

這是一款廣為使用的搜尋與分析引擎、最近整合了向量搜尋功能。這項新功能可強化其功能、讓 IT 更有效地處理及搜尋向量資料。

Pinecone

它是一個強大的向量資料庫、具有一組獨特的功能。它在索引功能中同時支援密集和稀疏的向量、可增強其靈活度和適應能力。其主要優勢之一在於能夠將傳統搜尋方法與 AI 型密集向量搜尋結合、建立混合式搜尋方法、充分發揮兩者的最佳效益。

Pinecone 主要是雲端型、專為機器學習應用程式所設計、並與多種平台完美整合、包括 GCP 、 AWS 、 Open AI 、 GPT -3 、 GPT-3.5 、 GPT-4 、 Catgut Plus 、 Elasticsearch 、 Haystack 、 還有更多。請務必注意、 Pinecone 是一個封閉來源平台、可作為「軟體即服務」（ SaaS ）產品使用。

Pinecone 具備先進功能、特別適合網路安全產業、可有效運用其高維度搜尋和混合式搜尋功能來偵測和回應威脅。

CHROMA

這是一個向量資料庫、具有四個主要功能的核心 API 、其中一個包含記憶體內文件向量儲存區。它也利用 Face Transformers 程式庫來向文件進行向量化、以增強文件的功能和多用途性。
Chroma 可在雲端和內部環境中運作、根據使用者需求提供靈活度。特別是在音訊相關應用程式中、它是音訊型搜尋引擎、音樂推薦系統和其他音訊相關使用案例的最佳選擇。

Weaviate

這是一個多功能的向量資料庫、可讓使用者使用其內建模組或自訂模組、將內容向量化、根據特定需求提供靈活度。它同時提供完全託管和自行代管的解決方案、可因應各種部署偏好。

Weaviate 的其中一項重要功能是能夠同時儲存向量和物件、以增強其資料處理能力。它廣泛用於多種應用程式、包括在 ERP 系統中進行語義搜尋和資料分類。在電子商務領域、它提供搜尋和推薦引擎。Weaviate 也可用於影像搜尋、異常偵測、自動資料協調、以及網路安全威脅分析、顯示其在多個網域中的多功能性。

紅皮

Redis 是高效能的向量資料庫、以其快速的記憶體內儲存設備聞名、提供低延遲的讀寫作業。因此、對於需要快速存取資料的推薦系統、搜尋引擎和資料分析應用程式來說、這是絕佳的選擇。

Redis 支援各種向量資料結構、包括清單、集和排序集。它也提供向量作業、例如計算向量之間的距離、或尋找交叉和聯合。這些功能對於相似性搜尋、叢集和內容型建議系統特別有用。

在擴充性和可用度方面、 Redis 在處理高處理量工作負載方面表現優異、並提供資料複寫功能。它也能與其他資料類型完美整合、包括傳統關聯式資料庫（ RDBMS ）。
Redis 包含發佈 / 訂閱（發佈 / 訂閱（發佈 / 訂閱）（發佈 / 訂閱）（發佈 / 訂閱）功能、可用於即時更新、這對管理即時向量很有幫助。此外、 Redis 不僅重量輕、使用簡易、更是管理向量資料的易用解決方案。

Milvus

這是一個多功能的向量資料庫、提供類似文件儲存區的 API 、就像 MongoDB 一樣。它之所以脫穎而出、是因為它支援多種資料類型、使其成為資料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、可讓使用者在執行階段指定用於搜尋的向量類型。此外、它還利用 KnowWhere 程式庫、它位於其他程式庫（如 Faiss ）之上、來管理查詢與向量搜尋演算法之間的通訊。

由於 Milvus 與 PyTorch 和 TensorFlow 相容、因此也能與機器學習工作流程無縫整合。這使得它成為各種應用程式的絕佳工具、包括電子商務、影像和視訊分析、物件辨識、影像相似度搜尋和內容型影像擷取。在自然語言處理領域、 Milvus 用於文件叢集、語義搜尋和問題解答系統。

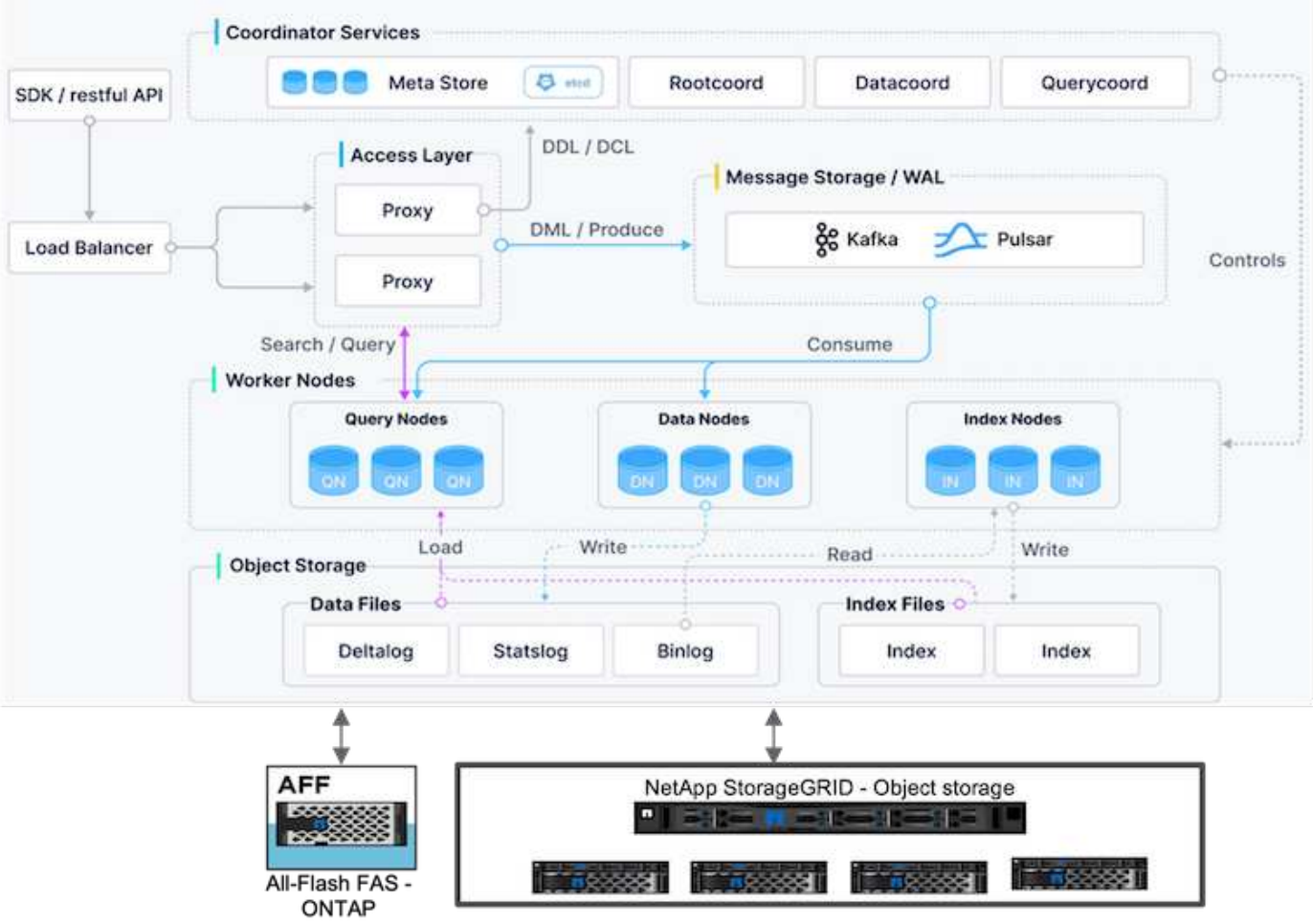
針對此解決方案、我們選擇 Milvus 進行解決方案驗證。為了提升效能、我們同時使用 milvus 和 postgres （ pgveco.RS ）。

為什麼我們選擇 **milvus** 來解決此問題？

- 開放原始碼： Milvus 是開放原始碼向量資料庫、鼓勵社群導向的開發與改善。
- AI 整合：利用內嵌相似性搜尋和 AI 應用程式來增強向量資料庫功能。
- 大容量處理： Milvus 可儲存、索引及管理深度神經網路（ DNN ）和機器學習（ ML ）模式所產生的十多億種內嵌向量。
- 易用：使用方便、設定不到一分鐘。Milvus 也提供適用於不同程式設計語言的 SDK 。
- 速度：提供超快的擷取速度、比某些替代方案快 10 倍。

- 擴充性與可用度： Mivus 具有高度擴充性、並可視需要進行擴充。
- 功能豐富：支援不同的資料類型、屬性篩選、使用者定義功能（ UDF ）支援、可設定的一致性層級和差旅時間、讓它成為各種應用程式的多功能工具。

Milvus 架構總覽



本節提供更高的槓桿元件和服務、用於 Milvus 架構。
* 存取層：由一組無狀態 Proxy 組成、可做為使用者的系統和端點的前層。
* 協調員服務：將工作指派給工作節點、並做為系統的大腦。它有三種協調器類型：根座標、資料座標和查詢座標。
* 工作者節點：它遵循協調器服務的指示、執行使用者觸發的 DML/DDL commands.it 有三種類型的工作者節點、例如查詢節點、資料節點和索引節點。
* 儲存：負責資料持續性。它包含中繼儲存設備、記錄檔代理程式和物件儲存設備。NetApp 儲存設備（例如 ONTAP 和 StorageGRID ）可為客戶資料和向量資料庫資料、提供物件儲存和檔案型儲存設備給 Milvus 。

技術需求

技術需求

除了效能之外、以下所述的硬體和軟體組態已用於本文件中執行的大多數驗證。這些組態是協助您設定環境的準則。不過、請注意、特定元件可能會因個別客戶需求而異。

硬體需求

硬體	詳細資料
NetApp AFF 儲存陣列 HA 配對	* A800 * ONTAP 9.14.1 * 48 x 3.49TB SSD - NVM * 兩個彈性群組磁碟區：中繼資料和資料。 * 中繼資料 NFS 磁碟區有 12 個容量 250 GB 的持續磁碟區。 * 資料是 ONTAP NAS S3 Volume
6 x Fujitsu PRIMERGY RX2540 M4	* 64 個 CPU * Intel （R） Xeon （R） Gold 6142 CPU @ 2.60GHz * 256 GM 實體記憶體 1 個 100GbE 網路連接埠
網路	100 GbE
StorageGRID	* 1 個 SG100 、 3 個 SGF6024 * 3 x 24 x 7.68TB

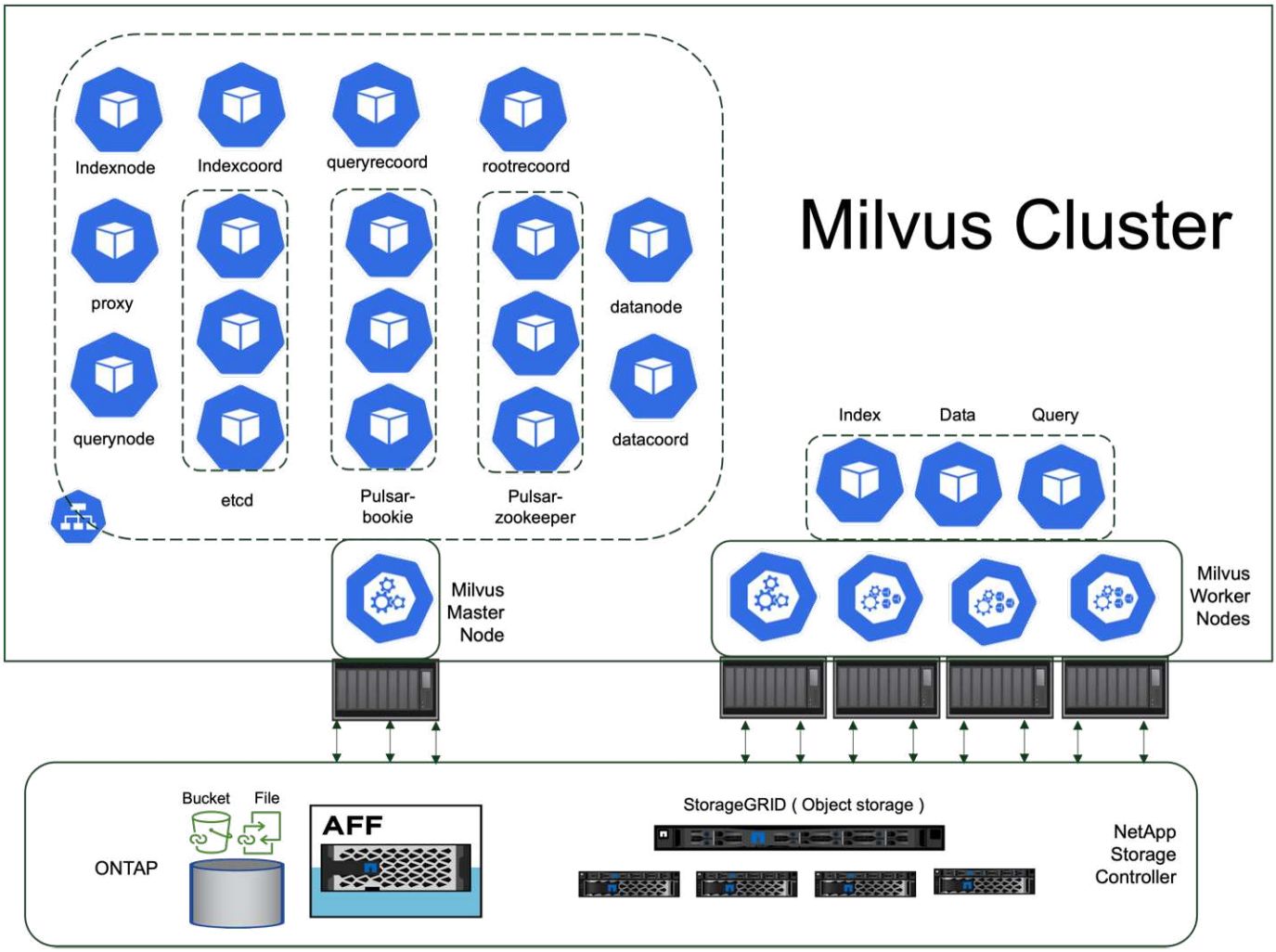
軟體需求

軟體	詳細資料
Milvus 叢集	* 圖表 - milvus-4.1.11 。 * 應用程式版本– 2.3.4 * 相關套裝組合、例如 bookkeeper 、 zookkeeper 、 Pulsar 、 etcd 、 Proxy 、 querynod節點 、 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.

部署程序

部署程序

在本部署區段中、我們將 Milvus 向量資料庫與 Kubernetes 一起用於實驗室設定、如下所示。



NetApp 儲存設備可為叢集提供儲存設備、以保留客戶資料和 milvus 叢集資料。

NetApp 儲存設備設定**– ONTAP**

- 儲存系統初始化
- 建立儲存虛擬機器（ SVM ）
- 邏輯網路介面指派
- NFS 、 S3 組態和授權

請遵循下列 NFS （網路檔案系統）步驟：

1. 為 NFSv4 建立 FlexGroup Volume 。在我們的驗證設定中、我們使用了 48 個 SSD 、 1 個 SSD 專用於控制器的根磁碟區、 47 個 SSD 散佈於 NFSv4]] 。請確認 FlexGroup 磁碟區的 NFS 匯出原則具有 Kubernetes （ K8s ）節點網路的讀取 / 寫入權限。如果沒有這些權限、請為 K8s 節點網路授予讀取 / 寫入（ RW ）權限。
2. 在所有 K8s 節點上、建立資料夾、並透過每個 K8s 節點上的邏輯介面（ LIF ）、將 FlexGroup 磁碟區掛載至此資料夾。

請針對 NAS S3 （網路附加儲存簡易儲存服務）執行下列步驟：

1. 建立 FlexGroup Volume for NFS 。

2. 使用「 vserver object-store-server create 」命令設定物件儲存區伺服器、並將管理狀態設定為「 up 」。您可以選擇啟用 HTTPS 並設定自訂接聽程式連接埠。
3. 使用「 vserver object-store-server user create -user <username> 」命令建立物件儲存伺服器使用者。
4. 若要取得存取金鑰和秘密金鑰、您可以執行下列命令：「 Set diag; vserver object-store-server user show -user <username> 」。不過、在使用者建立程序期間、將會提供這些金鑰、也可以使用 REST API 呼叫來擷取這些金鑰。
5. 使用在步驟 2 中建立的使用者建立物件儲存區伺服器群組、並授予存取權。在此範例中、我們提供了「 FullAccess 」。
6. 將 NAS 貯體的類型設定為「 NAS 」、並提供 NFSv3 Volume 的路徑、以建立 NAS 貯體。您也可以使用 S3 儲存貯體來達到此目的。

NetApp 儲存設備設定**– StorageGRID**

1. 安裝 StorageGRID 軟體。
2. 建立租戶和貯體。
3. 建立具有必要權限的使用者。

如需詳細資訊、請參閱 https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

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}}]}
```

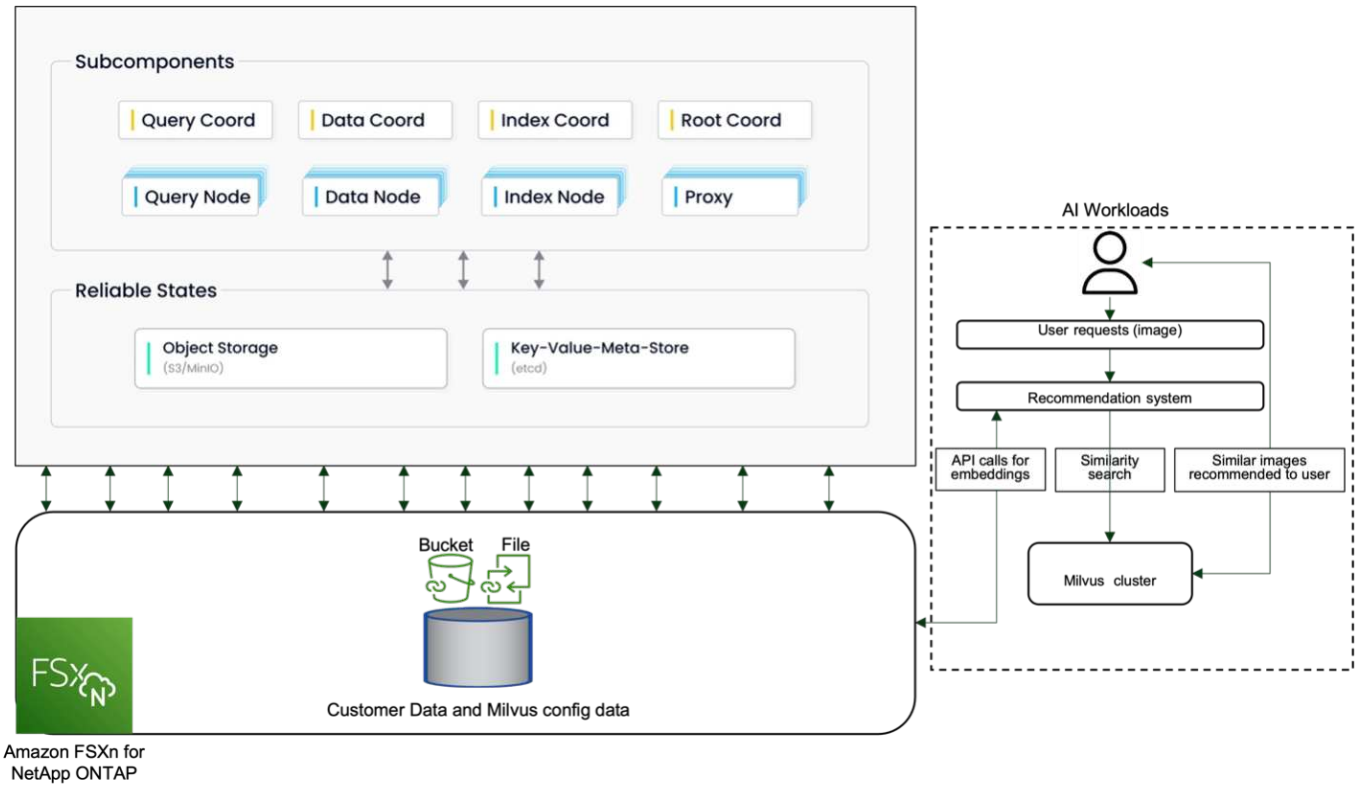
根據上述驗證、 Kubernetes 與向量資料庫的整合、透過在 Kubernetes 上使用 NetApp 儲存控制器部署 Milvus 叢集、為客戶提供強大、可擴充且有效率的解決方案、以管理大規模資料作業。這項設定可讓客戶快速有效地處理高維度資料、並執行複雜查詢、是大型資料應用程式和 AI 工作負載的理想解決方案。將持續磁碟區（ PV ）用於各種叢集元件、以及從 NetApp ONTAP 建立單一 NFS 磁碟區、可確保最佳的資源使用率和資料管理。驗證 PersistentVolume Claims （ PVCS ）和 Pod 狀態的程序、以及測試資料寫入和讀取、可讓客戶確保資料作業可靠且一致。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/448789845791611912/448789845791611913/448789845791611920/0/448789845791411917/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/1/448789845791411918/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/100/448789845791411913/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/101/448789845791411914/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/102/448789845791411915/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/103/448789845791411916/1c48ab6e-1546-4503-9084-28c629216c33/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/103/448789845791411916/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
```

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

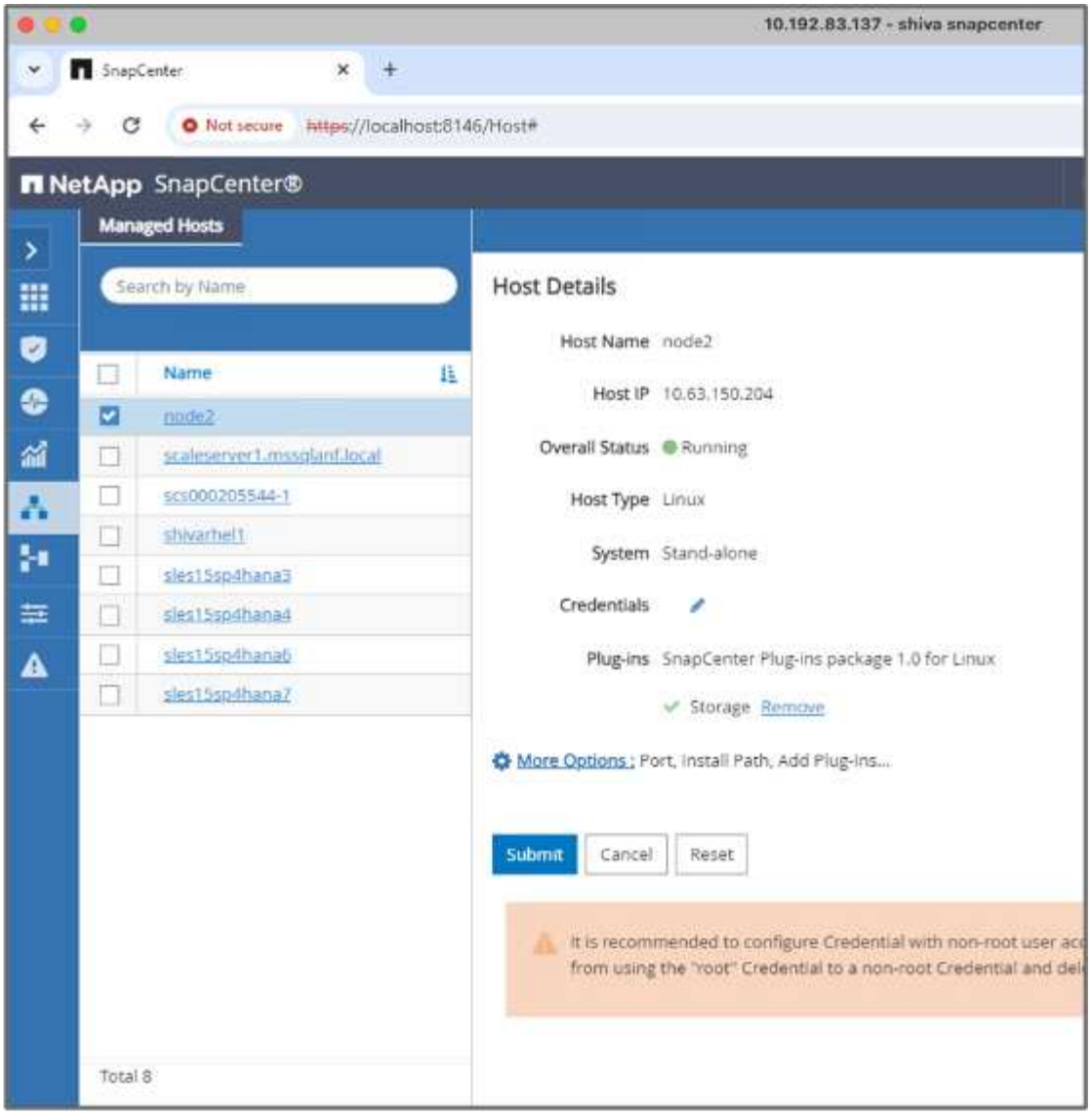
本節有效說明客戶如何在 Docker 容器中部署及操作獨立的 Milvus 設定、並運用 Amazon 的 NetApp FSxN 來儲存 NetApp ONTAP 資料。這項設定可讓客戶運用向量資料庫的強大功能、在 Docker 容器的可擴充且有效率的環境中、處理高維度資料並執行複雜的查詢。透過為 NetApp ONTAP 執行個體建立 Amazon FSxN 並搭配 EC2 執行個體、客戶可以確保最佳的資源使用率和資料管理。成功驗證向量資料庫中 FSxN 的資料寫入與讀取作業、可讓客戶確保資料作業穩定可靠。此外、透過 S3 傳輸協定列出（讀取） AI 工作負載資料的能力、可增強資料存取能力。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

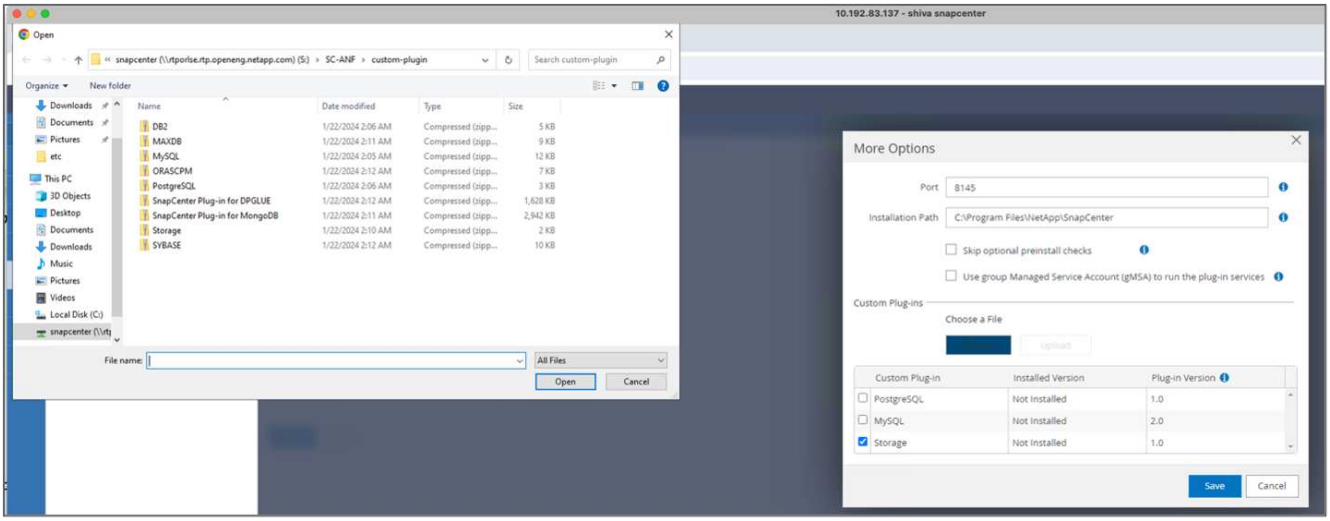
使用 **NetApp SnapCenter** 保護向量資料庫。

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。

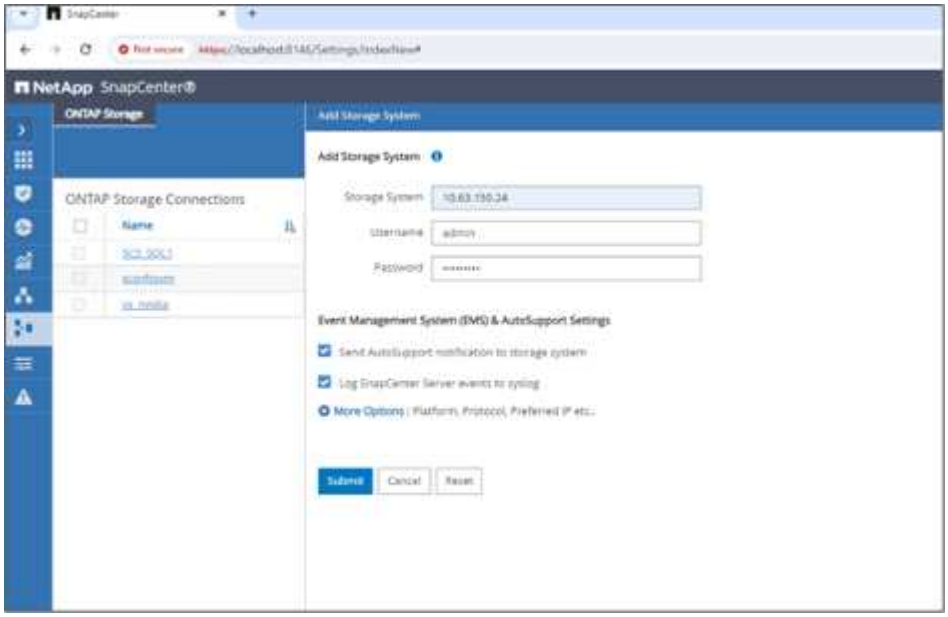


2. 安裝及設定儲存外掛程式。從新增的主機中、選取「更多選項」。瀏覽並從選取下載的儲存外掛程式 "NetApp Automation Store"。安裝外掛程式並儲存組態。

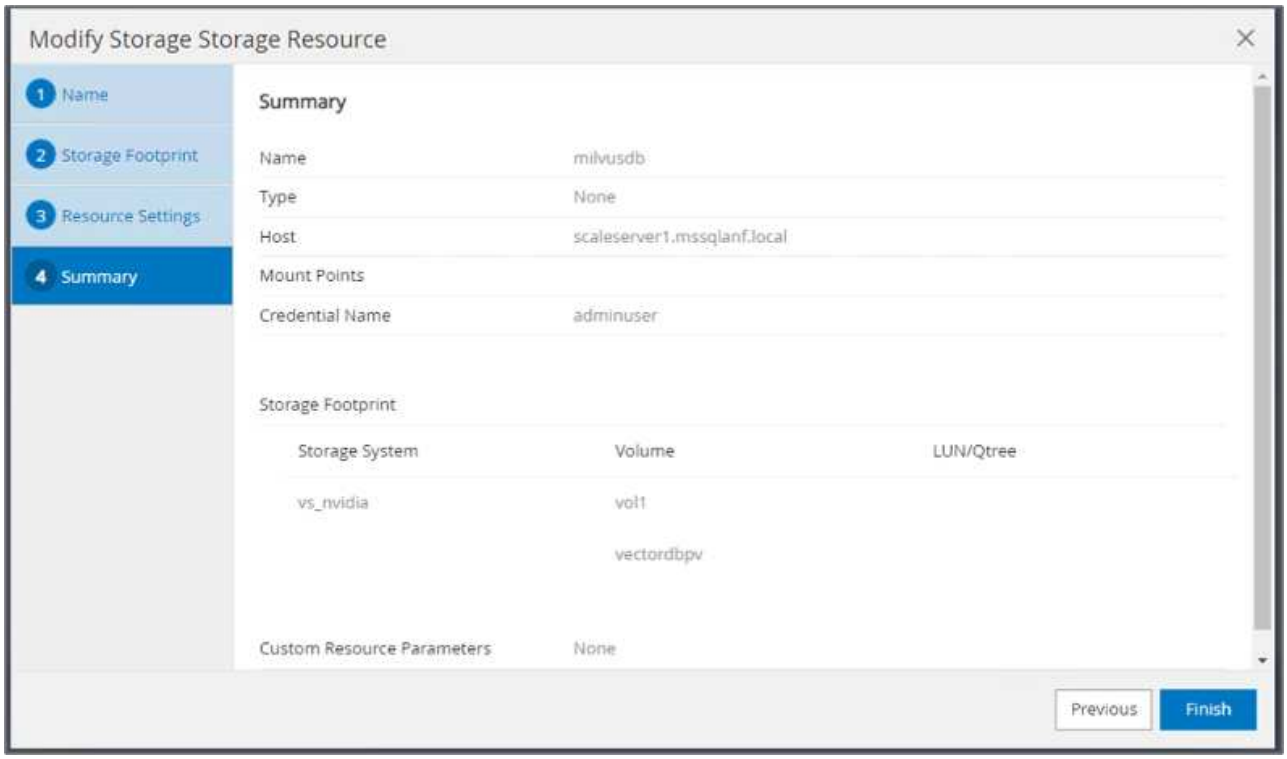


3. 設定儲存系統和磁碟區：在「儲存系統」下新增儲存系統、然後選取 SVM （儲存虛擬機器）。在此範例

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。



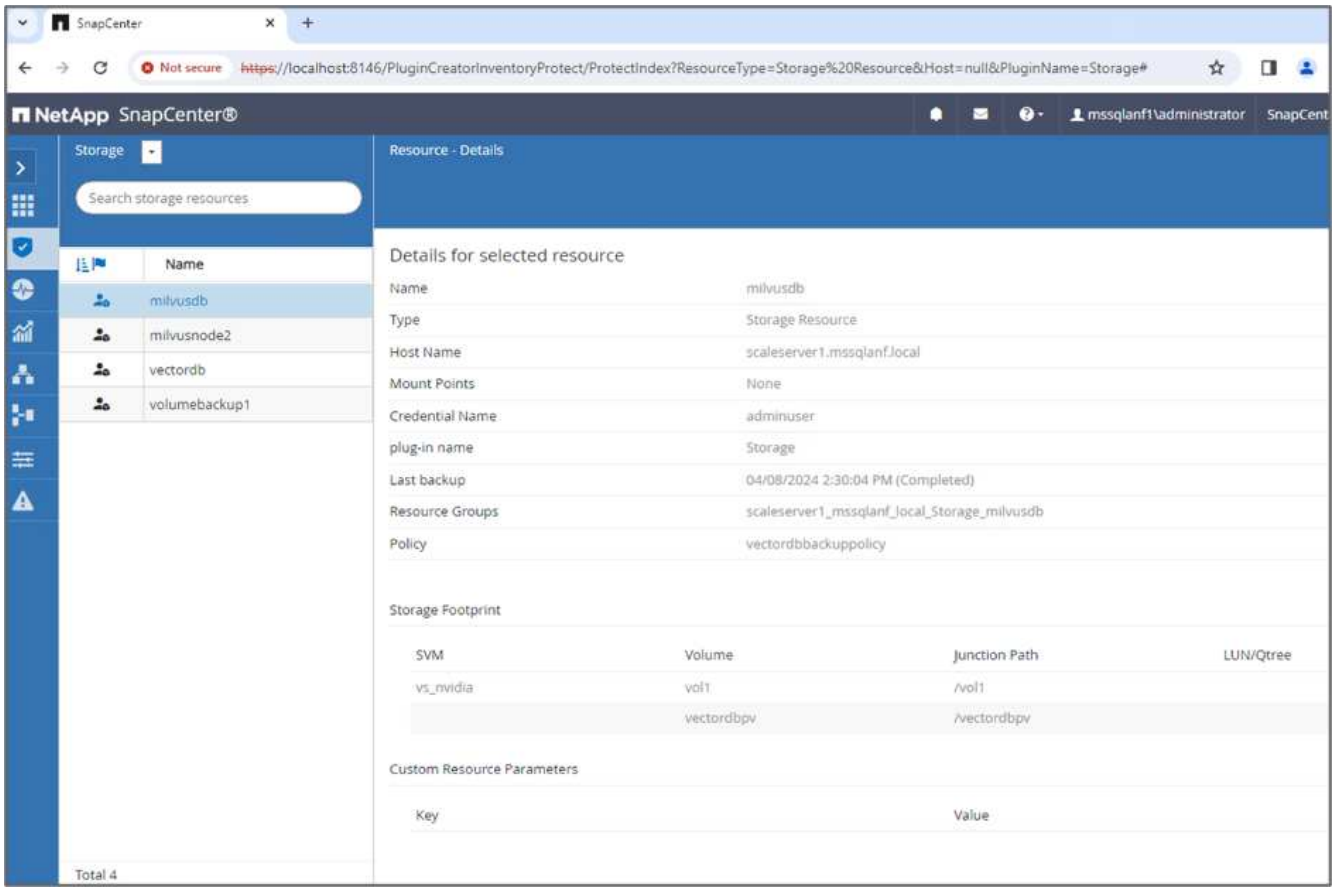
5. 使用 Python 指令碼將資料插入 S3 NAS 貯體。在我們的案例中、我們修改了 Milvus 所提供的備份指令碼、即「 prepy_data_NetApp.py 」、並執行「 Sync 」命令來清除作業系統中的資料。

```
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

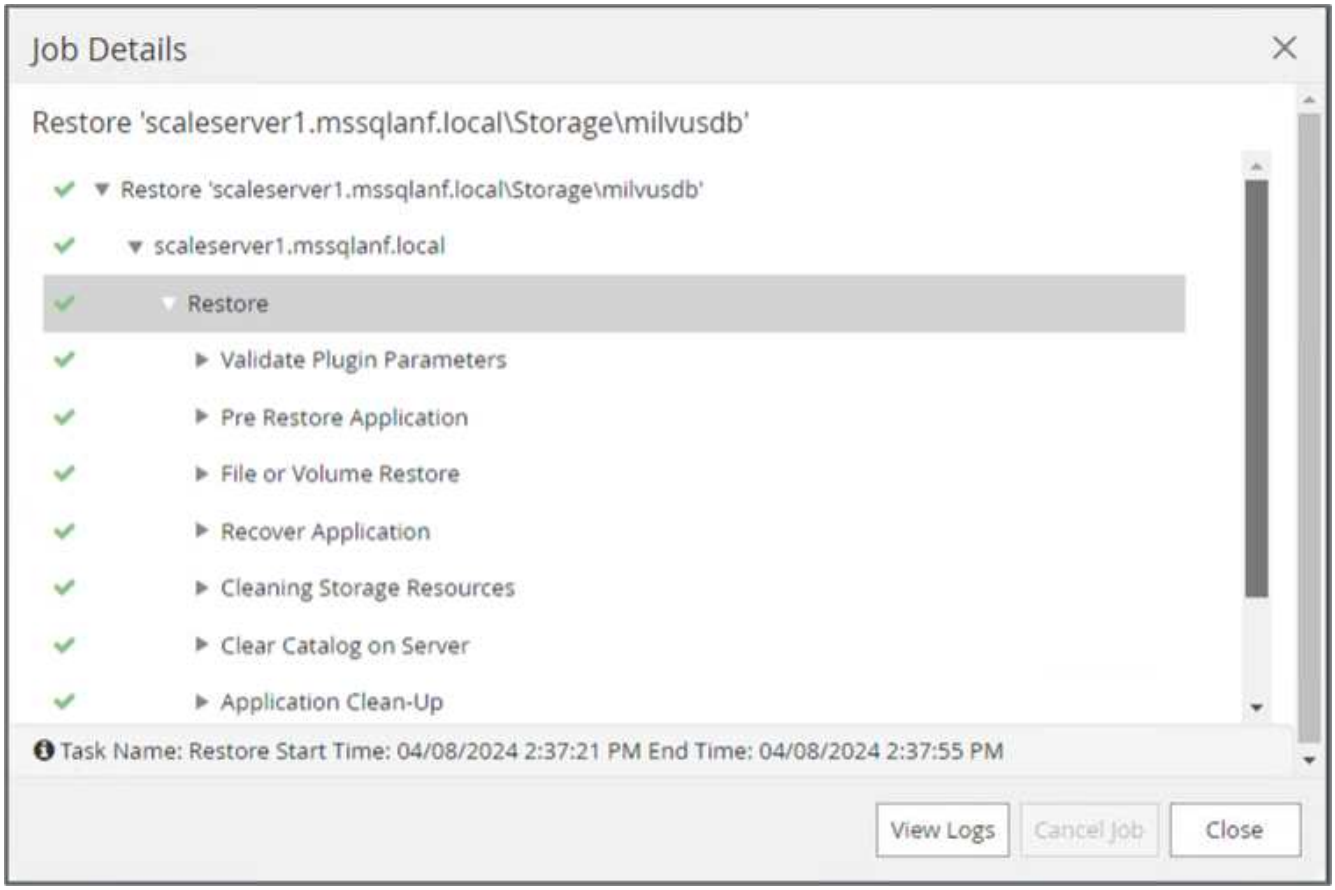


8. 為了測試備份功能、我們會在備份程序之後新增一個表格、或是從 NFS （ S3 NAS 儲存區）移除部分資料。

   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:~#
```

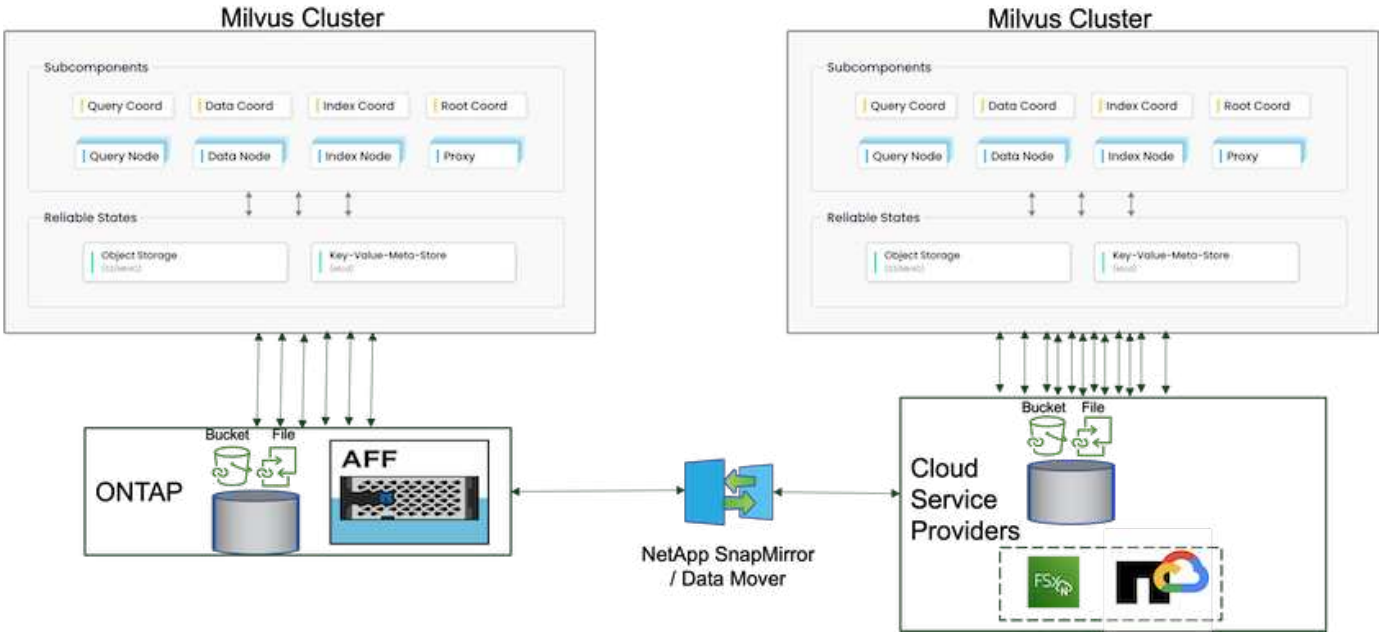
最後、使用 NetApp 的 SnapCenter 來保護向量資料庫資料和位於 ONTAP 的 Milvus 資料、對客戶帶來重大效益、尤其是在資料完整性至關重要的產業、例如電影製作。SnapCenter 能夠建立一致的備份並執行完整資料還原、確保重要資料（例如內嵌視訊和音訊檔案）不會因硬碟故障或其他問題而遺失。這不僅可防止營運中斷、也可防止重大財務損失。

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。

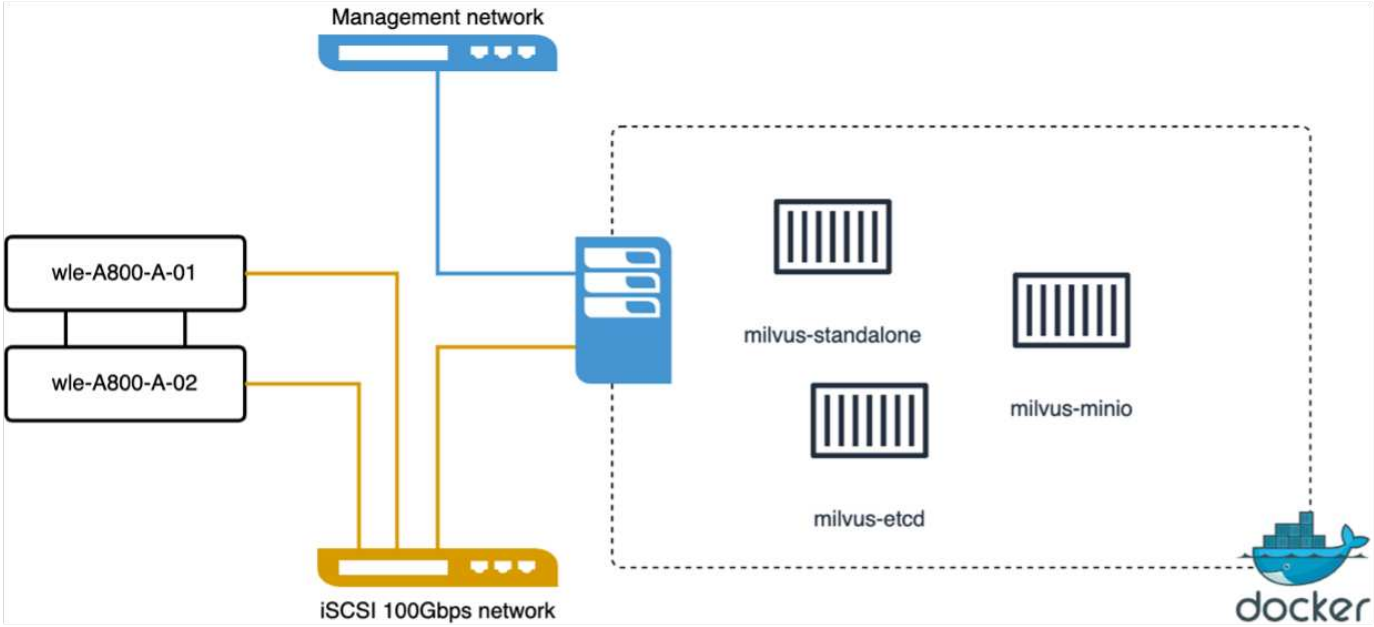
在本節中、我們的目標是深入探討向量資料庫（例如 Milvus 和 pgveco.RS ）的效能驗證、重點在於其儲存效能特性、例如 I/O 設定檔和 NetApp 儲存控制器行為、以支援 LLM 生命週期內的 RAG 和推斷工作負載。當這些資料庫與 ONTAP 儲存解決方案結合使用時、我們會評估並找出任何效能差異。我們的分析將以關鍵效能指標為基礎、例如每秒處理的查詢數（ QPS ）。

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。

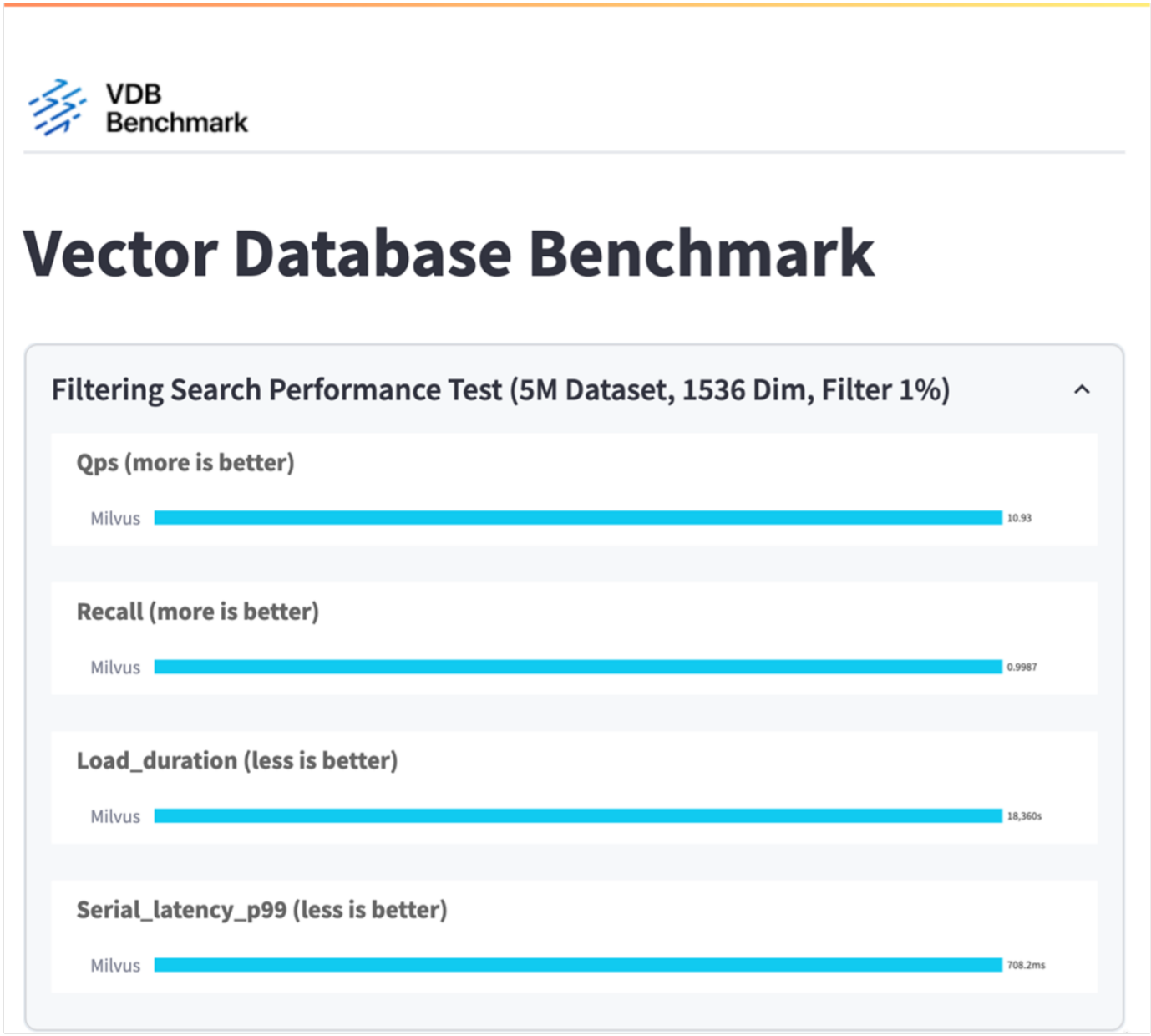
。 擷取、最佳化及建立約 100GB 資料集的索引約需 5 小時。在這段期間中、配備 20 個核心（啟用超執行緒時等於 40 個 vCPU ）的 Milvus 伺服器、其最大 CPU 容量為 100% 。我們發現、對於超過系統記憶體大小的大型資料集而言、 DiskANN 特別重要。
。 在查詢階段中、我們觀察到每秒查詢數（ QPS ）率為 10.93 、而且回收率為 0.9987 。查詢的第 99 個百分位數延遲是以 708.2 毫秒的時間來測量。

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。

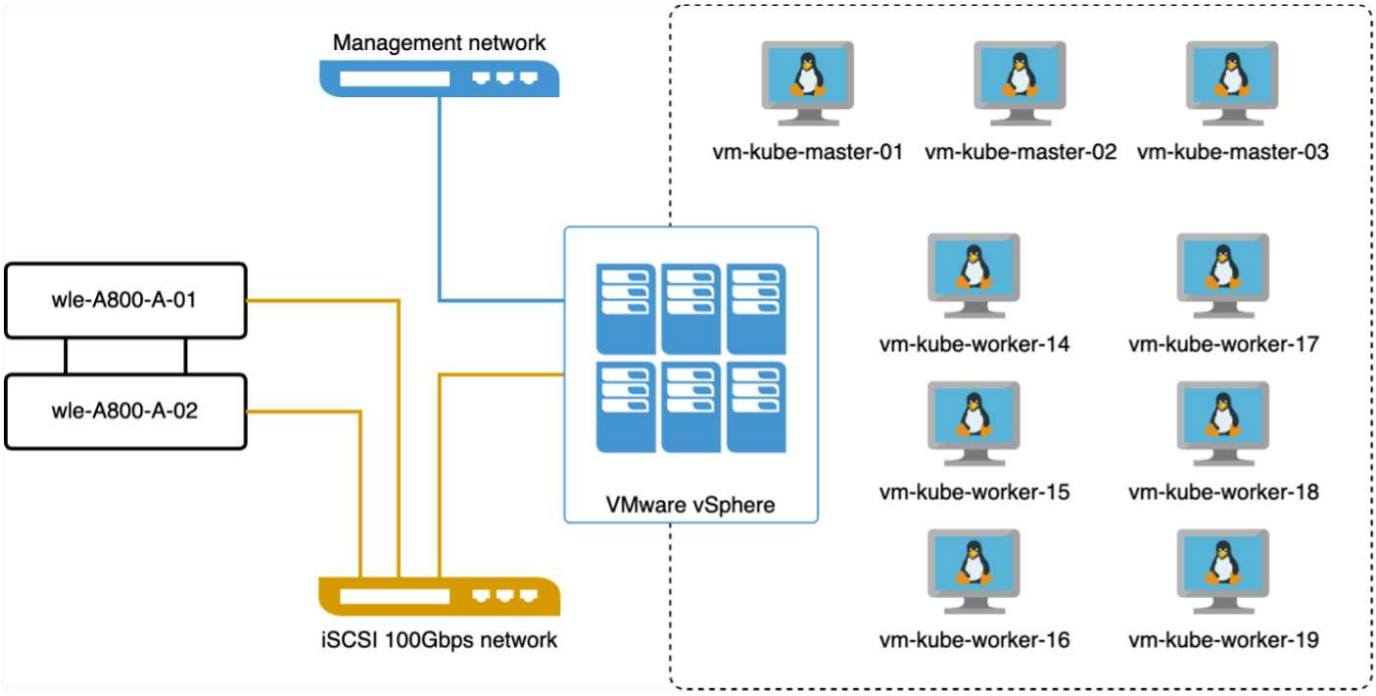


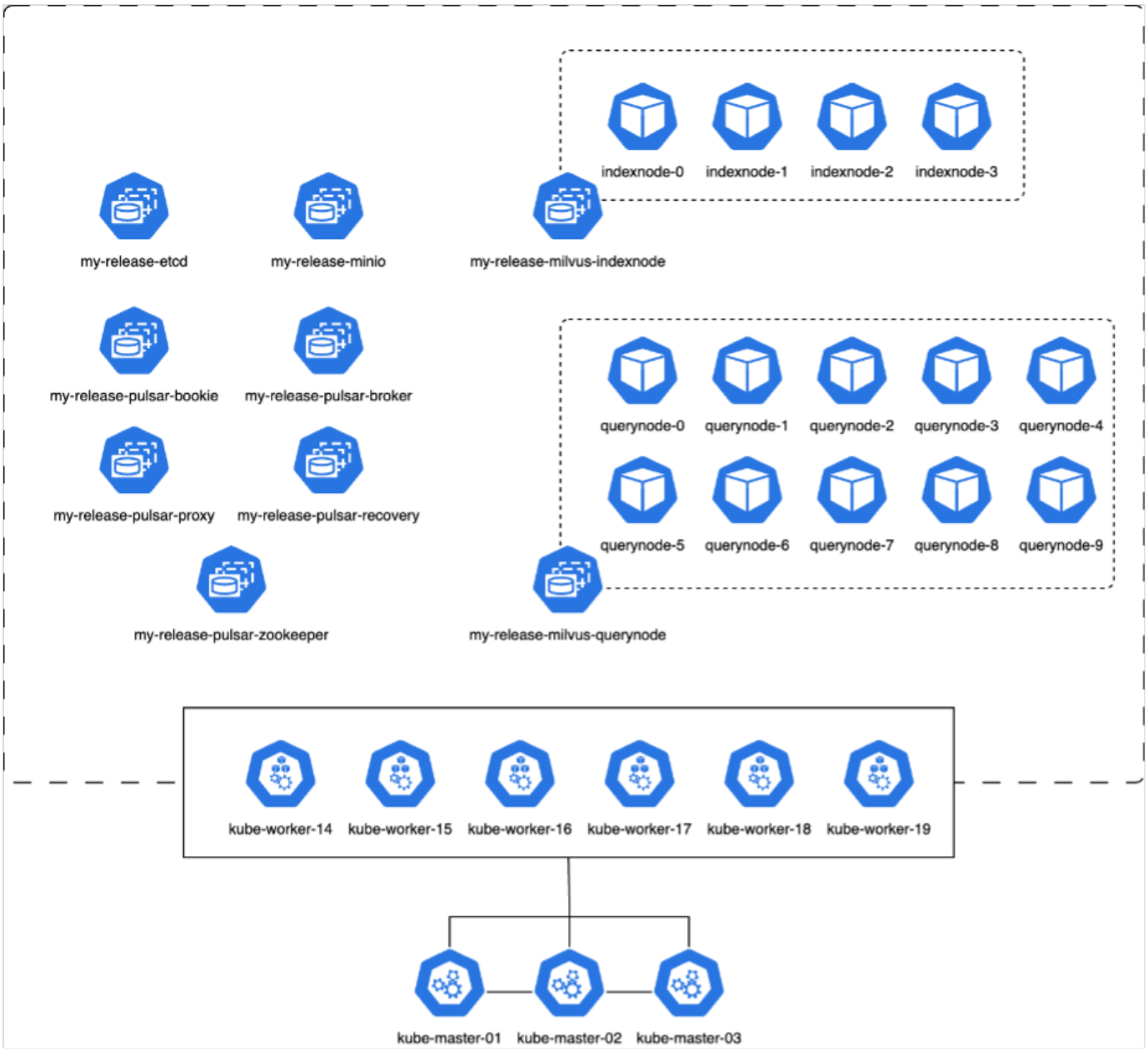
從獨立式 Milvus 執行個體的效能驗證來看、目前的設定顯然不足以支援容量為 1536 的 500 萬向量資料集。我們已確定儲存設備擁有足夠的資源、並不構成系統的瓶頸。

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）。

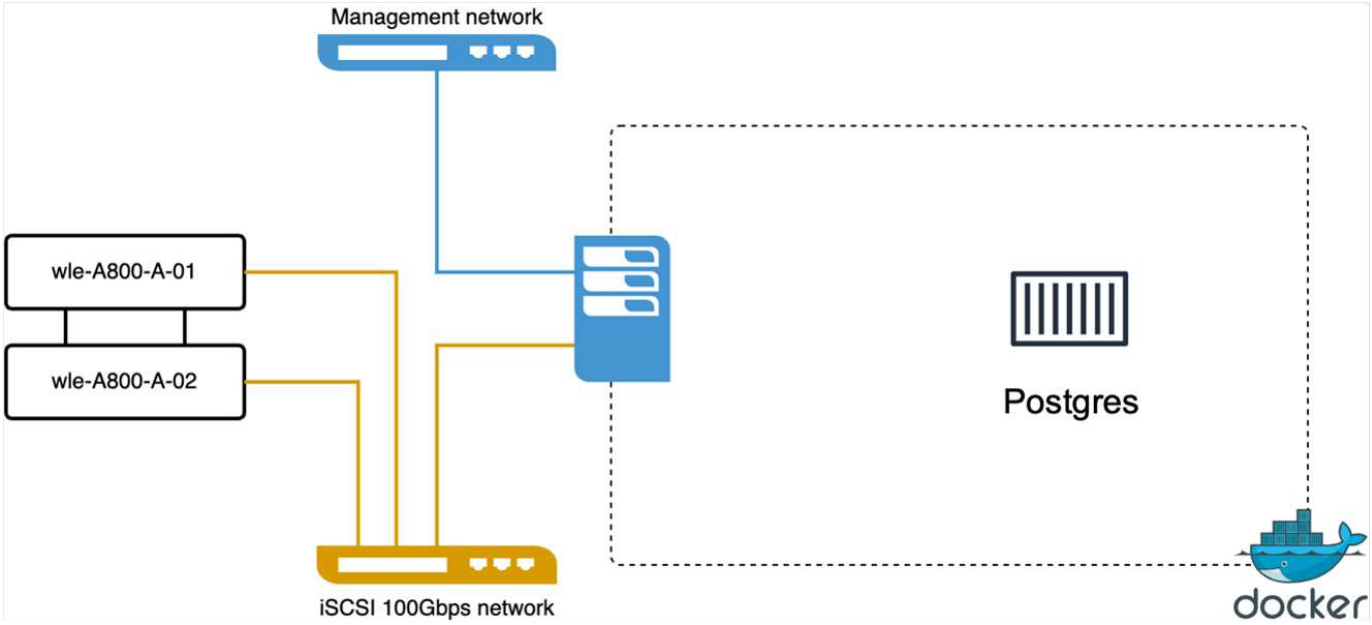
下節將說明主要的儲存效能指標。

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：

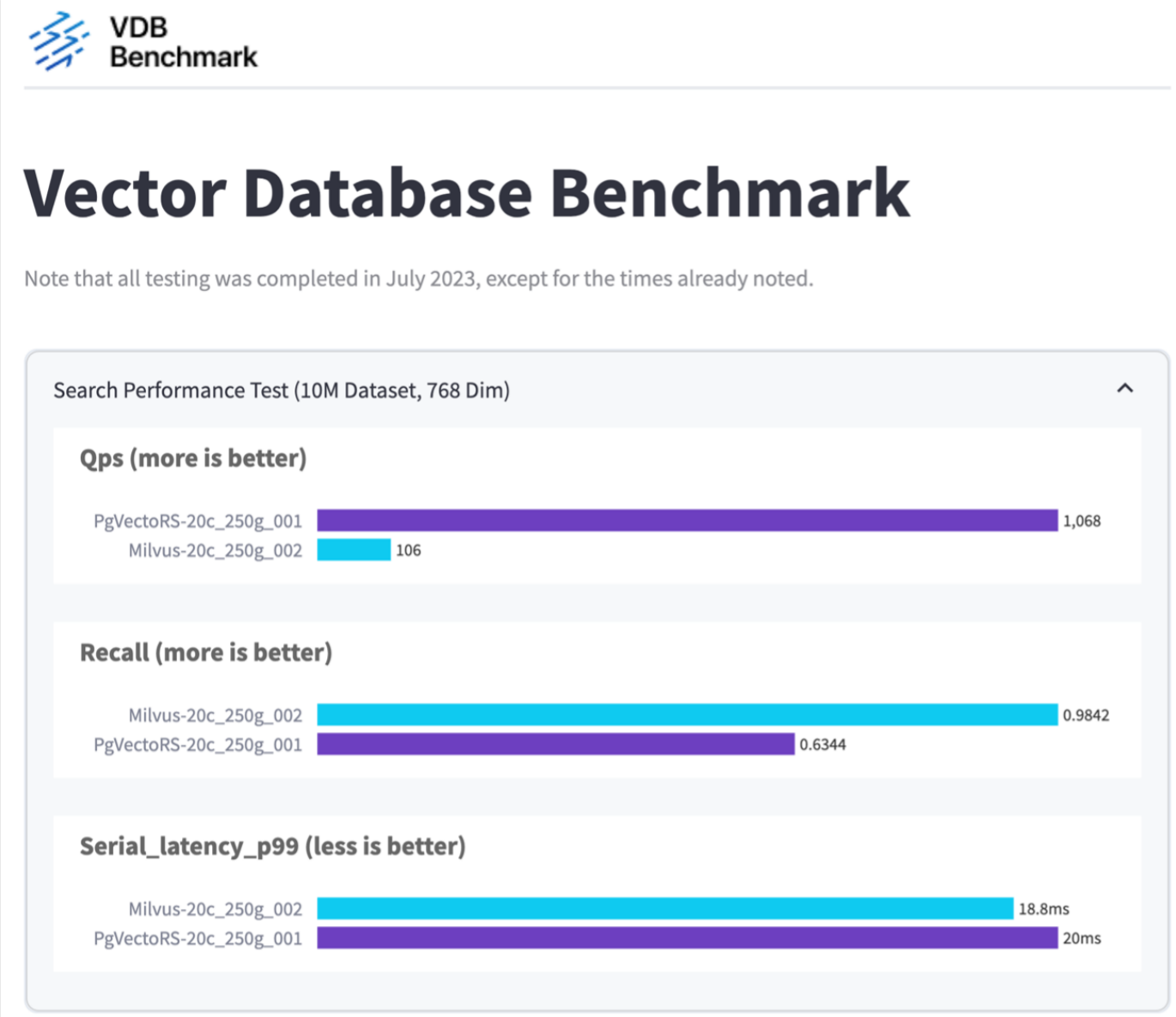


在本節中、我們分享了測試 PostgreSQL 資料庫的觀察結果、特別是使用 pgveco.RS 。
* 我們選擇 HNSW 作為這些測試的索引類型、因為在測試時、 DiskANN 無法用於 pgveco.RS 。
* 在資料擷取階段、我們載入 Cohere 資料集、其中包含 1 、 000 萬個向量、維度為 768 。此程序約需 4.5 小時。
* 在查詢階段、我們觀察到每秒查詢數（ QPS ）為 1 、 068 、召回率為 0.6344 。查詢的第 99 個百分位數延遲是以 20 毫秒為測量單位。在大部分的執行時間中、用戶端 CPU 以 100% 的容量運作。

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。

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

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

在本節中、我們將深入探討 instaclustr 產品如何與 PostgreSQL 在 pgvector 功能上整合的細節。我們提供了一個範例：「如何使用 PGVector 和 PostgreSQL ® 來改善 LLM 的準確度和效能：內嵌與 PGVector 角色簡介」。請檢查 "部落格" 以取得更多資訊。

向量資料庫使用案例

向量資料庫使用案例

在本節中、我們將討論兩個使用案例、例如使用大語言模式擷取擴增代和 NetApp IT 聊天。

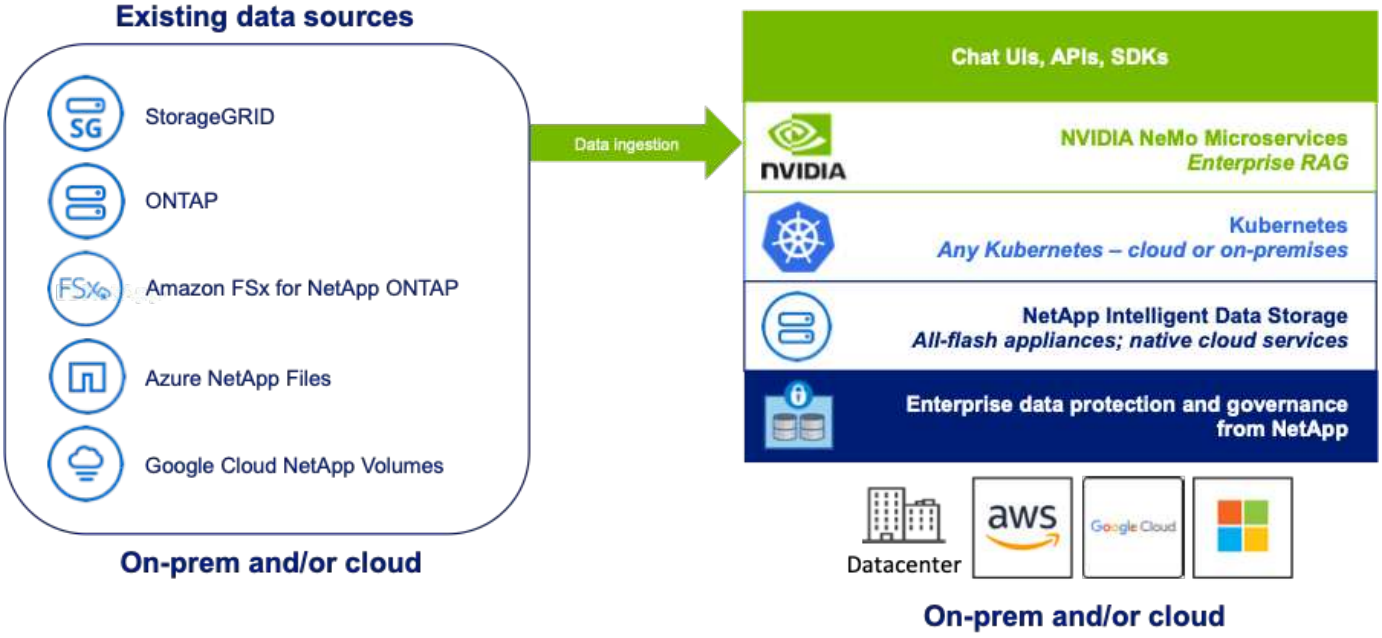
使用大語言模型（ **LRAM** ）擷取擴增產生（ **RAG** ）

```
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 是在企業中實作 RAG 的實用工具。此運算子可用於部署完整的 RAG 管線。您可以自訂 RAG 管道、以使用 Milvus 或 pgvecto 作為儲存知識庫嵌入資料的向量資料庫。如需詳細資訊、請參閱文件。

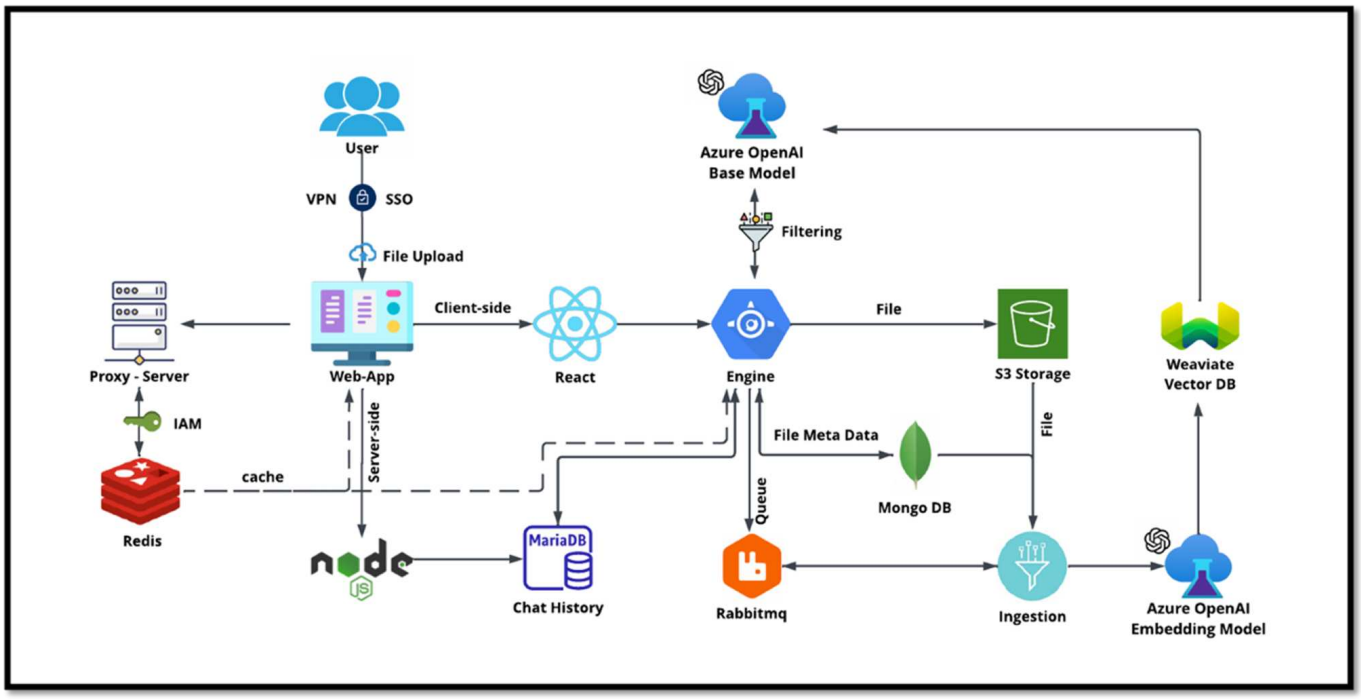
NetApp has validated an enterprise RAG architecture powered by the NVIDIA Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog post for more information and to see a demo. Figure 1 provides an overview of this architecture.

圖 1 ）採用 NVIDIA Nemo Microservices 和 NetApp 技術的企業級 RAG



NetApp IT chatbot 使用案例

NetApp 的 chatbot 是向量資料庫的另一個即時使用案例。在這種情況下、 NetApp Private OpenAI 沙箱提供有效、安全且有效率的平台、可管理 NetApp 內部使用者的查詢。它整合了嚴格的安全性通訊協定、高效率的資料管理系統、以及精密的 AI 處理功能、可根據使用者在組織中的角色和責任、透過 SSO 驗證、確保對使用者做出高品質、精準的回應。此架構強調了合併進階技術以建立以使用者為中心的智慧型系統的潛力。



使用案例可分為四個主要部分。

使用者驗證與驗證：

- 使用者查詢會先完成 NetApp 單一登入（ SSO ）程序、以確認使用者的身分識別。
- 驗證成功後、系統會檢查 VPN 連線、以確保資料傳輸安全無虞。

資料傳輸與處理：

- VPN 驗證完成後、資料會透過 NetAIChat 或 NetAIICureate Web 應用程式傳送至 MariaDB 。MariaDB 是一套快速且有效率的資料庫系統、用於管理及儲存使用者資料。
- 接著 MariaDB 會將資訊傳送至 NetApp Azure 執行個體、該執行個體會將使用者資料連線至 AI 處理單元。

與 **OpenAI** 和內容篩選的互動：

- Azure 執行個體會將使用者的問題傳送至內容篩選系統。此系統會清理查詢、並準備處理。
- 清理後的輸入會傳送至 Azure OpenAI 基礎模型、此模型會根據輸入產生回應。

回應產生與仲裁：

- 首先檢查基礎模型的回應、以確保其準確且符合內容標準。
- 通過檢查後、系統會將回應傳回給使用者。此程序可確保使用者獲得清楚、準確且適當的查詢答案。

結論

結論

最後、本文件提供關於 NetApp 儲存解決方案上部署和管理向量資料庫（例如 Milvus 和 pgvector ）的完整概

觀。我們討論了使用 NetApp ONTAP 和 StorageGRID 物件儲存設備的基礎架構準則、並透過檔案和物件存放區、驗證 AWS FSX for NetApp ONTAP 中的 Milvus 資料庫。

我們探討了 NetApp 的檔案物件雙重性、不僅展示了它在向量資料庫中的資料、也展示了其他應用程式的實用功能。我們也特別強調 NetApp 企業管理產品 SnapCenter 如何為向量資料庫資料提供備份、還原和複製功能、確保資料完整性和可用度。

本文件也說明 NetApp 的混合雲解決方案如何在內部部署和雲端環境中提供資料複寫與保護、提供無縫且安全的資料管理體驗。我們針對 NetApp ONTAP 上的 Milvus 和 pgvecto 等向量資料庫的效能驗證提供深入見解、提供有關其效率和擴充性的寶貴資訊。

最後、我們討論了兩個泛用 AI 使用案例：使用 LLM 和 NetApp 內部 ChatAI 。這些實用範例強調了本文所概述的概念和實務做法的實際應用程式和效益。總的來說、本文件是一份全面性指南、適合任何想要運用 NetApp 強大儲存解決方案來管理向量資料庫的人。

感謝

作者衷心感謝下列貢獻者、其他提供意見回饋和意見的人、讓這篇文章對 NetApp 客戶和 NetApp 領域非常有價值。

1. NetApp ONTAP AI & Analytics 技術行銷工程師 Sathish Thyagarajan
2. Mike Oglesby、NetApp技術行銷工程師
3. NetApp 資深總監 AJ Mahajan
4. NetApp 工作負載效能工程經理 Joe Scott
5. Pueet Dhawan 、 NetApp 產品管理 FSX 資深總監
6. NetApp FSX 產品團隊資深產品經理 Yuval Kalderon

何處可找到其他資訊

若要深入瞭解本文所述資訊、請檢閱下列文件和 / 或網站：

- Milvus 文件 - https://milvus.io/docs/overview.md
- Milvus 獨立式文件 - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- NetApp 產品文件 https://www.netapp.com/support-and-training/documentation/
- instaclustr - "instalragram 說明文件"

版本歷程記錄

版本	日期	文件版本歷程記錄
1.0版	2024 年 4 月	初始版本

附錄 A ： values.yaml

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

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

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 ：泊塢視窗 - 組合 .yml

附錄 D ：泊塢視窗 - 組合 .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
```