Soluzione di database vettoriale con NetApp

NetApp artificial intelligence solutions

NetApp
August 18, 2025

Sommario

Soluzione di database vettoriale con NetApp

Soluzione di database vettoriale con NetApp

Karthikeyan Nagalingam e Rodrigo Nascimento, NetApp

Questo documento fornisce un'analisi approfondita dell'implementazione e della gestione di database vettoriali, come Milvus e pgvecto, un'estensione open source di PostgreSQL, utilizzando le soluzioni di storage di NetApp. Descrive dettagliatamente le linee guida dell'infrastruttura per l'utilizzo di NetApp ONTAP e StorageGRID Object Storage e convalida l'applicazione del database Milvus in AWS FSx ONTAP. Il documento illustra la dualità file-oggetto di NetApp e la sua utilità per database vettoriali e applicazioni che supportano incorporamenti vettoriali. Sottolinea le capacità di SnapCenter, il prodotto di gestione aziendale di NetApp, nell'offrire funzionalità di backup e ripristino per database vettoriali, garantendo l'integrità e la disponibilità dei dati. Il documento approfondisce ulteriormente la soluzione cloud ibrida di NetApp, discutendone il ruolo nella replicazione e protezione dei dati negli ambienti on-premise e cloud. Include approfondimenti sulla convalida delle prestazioni dei database vettoriali su NetApp ONTAP e si conclude con due casi d'uso pratici sull'intelligenza artificiale generativa: RAG con LLM e ChatAI interno di NetApp. Questo documento costituisce una guida completa per sfruttare al meglio le soluzioni di storage di NetApp per la gestione dei database vettoriali.

L'architettura di riferimento si concentra sui seguenti punti:

13. "Appendice D: docker-compose.yml"

Introduzione

Questa sezione fornisce un'introduzione alla soluzione di database vettoriale per NetApp.

Introduzione

I database vettoriali affrontano in modo efficace le sfide progettate per gestire le complessità della ricerca semantica nei Large Language Models (LLM) e nell'intelligenza artificiale generativa (IA). A differenza dei tradizionali sistemi di gestione dei dati, i database vettoriali sono in grado di elaborare e ricercare vari tipi di dati, tra cui immagini, video, testo, audio e altre forme di dati non strutturati, utilizzando il contenuto dei dati stessi anziché etichette o tag.

I limiti dei sistemi di gestione di database relazionali (RDBMS) sono ben documentati, in particolare le difficoltà con le rappresentazioni di dati ad alta dimensionalità e i dati non strutturati comuni nelle applicazioni di intelligenza artificiale. Gli RDBMS spesso richiedono un processo lungo e soggetto a errori per appiattire i dati in strutture più gestibili, con conseguenti ritardi e inefficienze nelle ricerche. I database vettoriali, tuttavia, sono progettati per aggirare questi problemi, offrendo una soluzione più efficiente e accurata per la gestione e la ricerca di dati complessi e ad alta dimensionalità, facilitando così il progresso delle applicazioni di intelligenza artificiale.

Questo documento costituisce una guida completa per i clienti che attualmente utilizzano o intendono utilizzare database vettoriali, illustrando dettagliatamente le best practice per l'utilizzo di database vettoriali su piattaforme quali NetApp ONTAP, NetApp StorageGRID, Amazon FSx ONTAP per NetApp ONTAP e SnapCenter. I contenuti forniti nel presente documento coprono una vasta gamma di argomenti:

- Linee guida infrastrutturali per database vettoriali, come Milvus, fornite da NetApp Storage tramite NetApp ONTAP e StorageGRID Object Storage.
- Validazione del database Milvus in AWS FSx ONTAP tramite archivio di file e oggetti.
- Approfondisce la dualità file-oggetto di NetApp, dimostrandone l'utilità per i dati nei database vettoriali e in altre applicazioni.
- In che modo il prodotto Data Protection Management di NetApp, SnapCenter, offre funzionalità di backup e ripristino per i dati dei database vettoriali.
- In che modo l'Hybrid Cloud di NetApp offre replicazione e protezione dei dati negli ambienti on-premise e cloud.
- Fornisce approfondimenti sulla convalida delle prestazioni di database vettoriali come Milvus e pgvector su NetApp ONTAP.
- Due casi d'uso specifici: Retrieval Augmented Generation (RAG) con Large Language Models (LLM) e ChatAI del team IT di NetApp , che offrono esempi pratici dei concetti e delle pratiche delineati.

Panoramica della soluzione

Questa sezione fornisce una panoramica della soluzione di database vettoriale NetApp .

Panoramica della soluzione

Questa soluzione mette in mostra i vantaggi e le capacità distintive che NetApp offre per affrontare le sfide che i clienti dei database vettoriali si trovano ad affrontare. Sfruttando NetApp ONTAP, StorageGRID, le soluzioni cloud di NetApp e SnapCenter, i clienti possono aggiungere un valore significativo alle loro operazioni

aziendali. Questi strumenti non solo risolvono i problemi esistenti, ma migliorano anche l'efficienza e la produttività, contribuendo così alla crescita aziendale complessiva.

Perché NetApp?

- Le offerte di NetApp, come ONTAP e StorageGRID, consentono la separazione tra storage ed elaborazione, consentendo un utilizzo ottimale delle risorse in base a requisiti specifici. Questa flessibilità consente ai clienti di scalare in modo indipendente il proprio storage utilizzando le soluzioni di storage NetApp .
- Sfruttando i controller di storage di NetApp, i clienti possono fornire in modo efficiente i dati al proprio database vettoriale utilizzando i protocolli NFS e S3. Questi protocolli facilitano l'archiviazione dei dati dei clienti e gestiscono l'indice del database vettoriale, eliminando la necessità di più copie dei dati a cui si accede tramite metodi di file e oggetti.
- NetApp ONTAP fornisce supporto nativo per NAS e storage di oggetti attraverso i principali provider di servizi cloud come AWS, Azure e Google Cloud. Questa ampia compatibilità garantisce un'integrazione perfetta, consentendo la mobilità dei dati dei clienti, l'accessibilità globale, il ripristino di emergenza, la scalabilità dinamica e le prestazioni elevate.
- Grazie alle solide funzionalità di gestione dei dati di NetApp, i clienti possono stare tranquilli sapendo che i loro dati sono ben protetti da potenziali rischi e minacce. NetApp dà priorità alla sicurezza dei dati, offrendo ai clienti la tranquillità di sapere che le loro preziose informazioni sono al sicuro e integre.

Database vettoriale

Questa sezione tratta la definizione e l'uso di un database vettoriale nelle soluzioni di intelligenza artificiale NetApp .

Database vettoriale

Un database vettoriale è un tipo specializzato di database progettato per gestire, indicizzare e ricercare dati non strutturati utilizzando incorporamenti di modelli di apprendimento automatico. Invece di organizzare i dati in un formato tabellare tradizionale, li organizza come vettori ad alta dimensionalità, noti anche come incorporamenti vettoriali. Questa struttura unica consente al database di gestire dati complessi e multidimensionali in modo più efficiente e accurato.

Una delle funzionalità principali di un database vettoriale è l'utilizzo dell'intelligenza artificiale generativa per eseguire analisi. Ciò include ricerche di similarità, in cui il database identifica punti dati simili a un dato input, e rilevamento di anomalie, in cui può individuare punti dati che si discostano significativamente dalla norma.

Inoltre, i database vettoriali sono adatti a gestire dati temporali, ovvero dati con timestamp. Questo tipo di dati fornisce informazioni su "cosa" è successo e quando è successo, in sequenza e in relazione a tutti gli altri eventi all'interno di un dato sistema IT. Questa capacità di gestire e analizzare dati temporali rende i database vettoriali particolarmente utili per le applicazioni che richiedono la comprensione degli eventi nel tempo.

Vantaggi del database vettoriale per ML e AI:

- Ricerca ad alta dimensionalità: i database vettoriali eccellono nella gestione e nel recupero di dati ad alta dimensionalità, spesso generati nelle applicazioni di intelligenza artificiale e apprendimento automatico.
- Scalabilità: possono scalare in modo efficiente per gestire grandi volumi di dati, supportando la crescita e l'espansione dei progetti di intelligenza artificiale e apprendimento automatico.
- Flessibilità: i database vettoriali offrono un elevato grado di flessibilità, consentendo di gestire diversi tipi di dati e strutture.

- Prestazioni: garantiscono una gestione e un recupero dei dati ad alte prestazioni, fondamentali per la velocità e l'efficienza delle operazioni di intelligenza artificiale e apprendimento automatico.
- Indicizzazione personalizzabile: i database vettoriali offrono opzioni di indicizzazione personalizzabili, consentendo un'organizzazione e un recupero dei dati ottimizzati in base a esigenze specifiche.

Database vettoriali e casi d'uso.

Questa sezione fornisce vari database vettoriali e i dettagli sui loro casi d'uso.

Faiss e ScaNN

Si tratta di librerie che rappresentano strumenti essenziali nel campo della ricerca vettoriale. Queste librerie forniscono funzionalità fondamentali per la gestione e la ricerca nei dati vettoriali, il che le rende risorse inestimabili in questo settore specializzato della gestione dei dati.

Elasticsearch

È un motore di ricerca e analisi ampiamente utilizzato, che ha recentemente incorporato funzionalità di ricerca vettoriale. Questa nuova funzionalità ne migliora le funzionalità, consentendo di gestire e ricercare i dati vettoriali in modo più efficace.

Pigna

Si tratta di un solido database vettoriale con un set di funzionalità unico. Supporta sia vettori densi che sparsi nella sua funzionalità di indicizzazione, il che ne aumenta la flessibilità e l'adattabilità. Uno dei suoi punti di forza principali risiede nella capacità di combinare i metodi di ricerca tradizionali con la ricerca vettoriale densa basata sull'intelligenza artificiale, creando un approccio di ricerca ibrido che sfrutta il meglio di entrambi i mondi.

Basato principalmente sul cloud, Pinecone è progettato per applicazioni di apprendimento automatico e si integra bene con una varietà di piattaforme, tra cui GCP, AWS, Open AI, GPT-3, GPT-3.5, GPT-4, Catgut Plus, Elasticsearch, Haystack e altre ancora. È importante notare che Pinecone è una piattaforma closed-source ed è disponibile come offerta Software as a Service (SaaS).

Grazie alle sue capacità avanzate, Pinecone è particolarmente adatto al settore della sicurezza informatica, dove le sue capacità di ricerca ad alta dimensione e di ricerca ibrida possono essere sfruttate efficacemente per rilevare e rispondere alle minacce.

Croma

Si tratta di un database vettoriale dotato di una Core-API con quattro funzioni principali, una delle quali include un archivio di documenti vettoriali in memoria. Utilizza inoltre la libreria Face Transformers per vettorializzare i documenti, migliorandone la funzionalità e la versatilità. Chroma è progettato per funzionare sia nel cloud che in locale, offrendo flessibilità in base alle esigenze degli utenti. In particolare, eccelle nelle applicazioni audio, il che lo rende una scelta eccellente per motori di ricerca basati sull'audio, sistemi di raccomandazione musicale e altri casi d'uso audio-correlati.

Tessitura

Si tratta di un database vettoriale versatile che consente agli utenti di vettorializzare i propri contenuti utilizzando moduli integrati o moduli personalizzati, garantendo flessibilità in base a esigenze specifiche. Offre soluzioni sia completamente gestite che self-hosted, soddisfacendo una varietà di preferenze di distribuzione.

Una delle caratteristiche principali di Weaviate è la sua capacità di memorizzare sia vettori che oggetti, migliorando le sue capacità di gestione dei dati. È ampiamente utilizzato per una vasta gamma di applicazioni,

tra cui la ricerca semantica e la classificazione dei dati nei sistemi ERP. Nel settore dell'e-commerce, alimenta i motori di ricerca e di raccomandazione. Weaviate viene utilizzato anche per la ricerca di immagini, il rilevamento di anomalie, l'armonizzazione automatica dei dati e l'analisi delle minacce alla sicurezza informatica, dimostrando la sua versatilità in più ambiti.

Redis

Redis è un database vettoriale ad alte prestazioni, noto per la sua rapida archiviazione in memoria, che offre bassa latenza per le operazioni di lettura-scrittura. Ciò lo rende una scelta eccellente per sistemi di raccomandazione, motori di ricerca e applicazioni di analisi dei dati che richiedono un rapido accesso ai dati.

Redis supporta varie strutture dati per i vettori, tra cui elenchi, set e set ordinati. Fornisce inoltre operazioni vettoriali come il calcolo delle distanze tra vettori o la ricerca di intersezioni e unioni. Queste funzionalità sono particolarmente utili per la ricerca di similarità, il clustering e i sistemi di raccomandazione basati sui contenuti.

In termini di scalabilità e disponibilità, Redis eccelle nella gestione di carichi di lavoro ad alta produttività e offre la replica dei dati. Si integra bene anche con altri tipi di dati, compresi i database relazionali tradizionali (RDBMS). Redis include una funzionalità Pubblica/Sottoscrivi (Pub/Sub) per aggiornamenti in tempo reale, utile per la gestione dei vettori in tempo reale. Inoltre, Redis è leggero e semplice da usare, il che lo rende una soluzione intuitiva per la gestione dei dati vettoriali.

Milvus

Si tratta di un database vettoriale versatile che offre un'API simile a un archivio di documenti, molto simile a MongoDB. Si distingue per il supporto di un'ampia varietà di tipi di dati, il che lo rende una scelta popolare nei settori della scienza dei dati e dell'apprendimento automatico.

Una delle caratteristiche uniche di Milvus è la sua capacità di multi-vettorizzazione, che consente agli utenti di specificare in fase di esecuzione il tipo di vettore da utilizzare per la ricerca. Inoltre, utilizza Knowwhere, una libreria che si basa su altre librerie come Faiss, per gestire la comunicazione tra le query e gli algoritmi di ricerca vettoriale.

Milvus offre inoltre un'integrazione perfetta con i flussi di lavoro di apprendimento automatico, grazie alla compatibilità con PyTorch e TensorFlow. Ciò lo rende uno strumento eccellente per una vasta gamma di applicazioni, tra cui e-commerce, analisi di immagini e video, riconoscimento di oggetti, ricerca di similarità di immagini e recupero di immagini basato sui contenuti. Nell'ambito dell'elaborazione del linguaggio naturale, Milvus viene utilizzato per il clustering di documenti, la ricerca semantica e i sistemi di risposta alle domande.

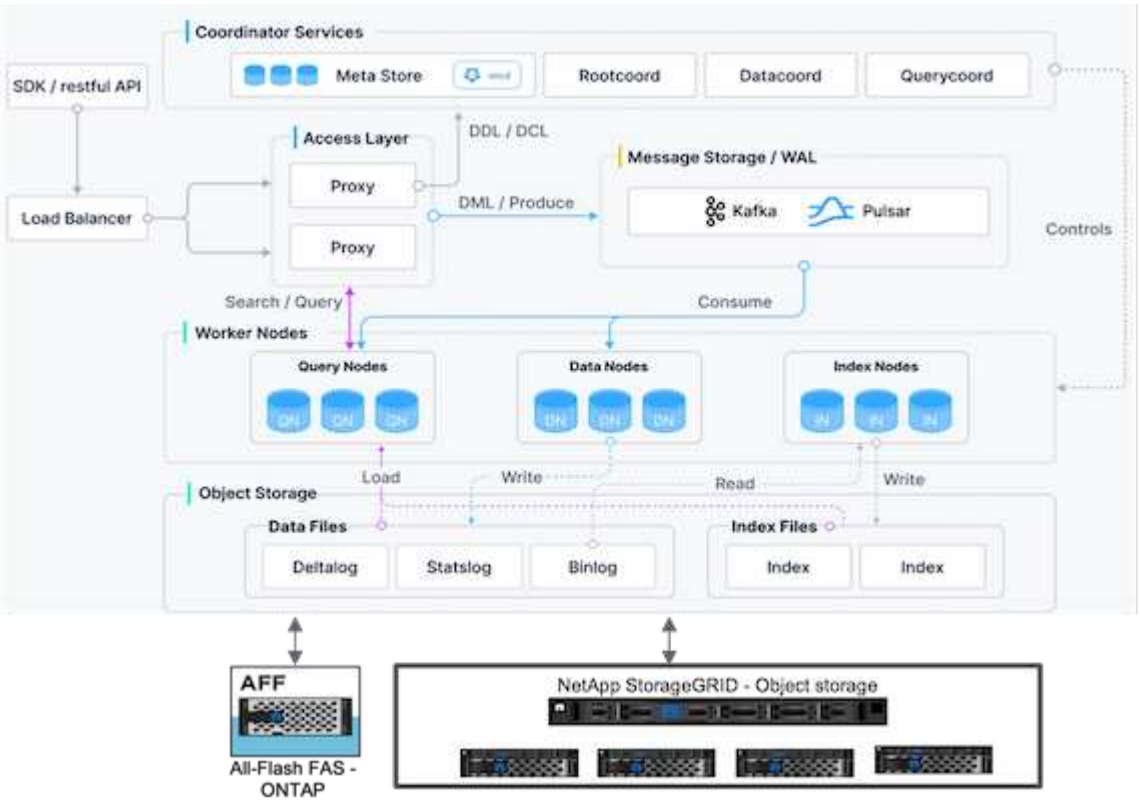
Per questa soluzione abbiamo scelto Milvus per la convalida della soluzione. Per le prestazioni, abbiamo utilizzato sia milvus che postgres(pgvecto.rs).

Perché abbiamo scelto Milvus per questa soluzione?

- Open-source: Milvus è un database vettoriale open-source che incoraggia lo sviluppo e i miglioramenti guidati dalla comunità.
- Integrazione AI: sfrutta l'integrazione della ricerca di similarità e delle applicazioni AI per migliorare la funzionalità del database vettoriale.
- Gestione di grandi volumi: Milvus è in grado di archiviare, indicizzare e gestire oltre un miliardo di vettori di incorporamento generati da modelli di reti neurali profonde (DNN) e apprendimento automatico (ML).
- Facile da usare: è facile da usare e la configurazione richiede meno di un minuto. Milvus offre anche SDK per diversi linguaggi di programmazione.
- Velocità: offre velocità di recupero incredibilmente elevate, fino a 10 volte superiori rispetto ad alcune alternative.

- Scalabilità e disponibilità: Milvus è altamente scalabile, con opzioni di scalabilità verticale e orizzontale in base alle esigenze.
- Ricco di funzionalità: supporta diversi tipi di dati, filtraggio degli attributi, supporto delle funzioni definite dall'utente (UDF), livelli di coerenza configurabili e tempi di percorrenza, il che lo rende uno strumento versatile per varie applicazioni.

Panoramica dell'architettura di Milvus



Questa sezione illustra i componenti e i servizi di livello superiore utilizzati nell'architettura Milvus. * Livello di accesso: è composto da un gruppo di proxy stateless e funge da livello frontale del sistema e da endpoint per gli utenti. * Servizio di coordinamento: assegna i compiti ai nodi worker e funge da cervello del sistema. Ha tre tipi di coordinatore: root coord, data coord e query coord. * Nodi worker: seguono le istruzioni del servizio coordinatore ed eseguono i comandi DML/DDL attivati dall'utente. Hanno tre tipi di nodi worker: nodo query, nodo dati e nodo indice. * Archiviazione: è responsabile della persistenza dei dati. Comprende meta-archiviazione, log broker e archiviazione di oggetti. Le soluzioni di storage NetApp , come ONTAP e StorageGRID, forniscono a Milvus storage di oggetti e storage basato su file sia per i dati dei clienti che per i dati dei database vettoriali.

Requisiti tecnologici

Questa sezione fornisce una panoramica dei requisiti per la soluzione di database vettoriale NetApp .

Requisiti tecnologici

Le configurazioni hardware e software descritte di seguito sono state utilizzate per la maggior parte delle convalide eseguite in questo documento, ad eccezione delle prestazioni. Queste configurazioni servono come linee guida per aiutarti a configurare il tuo ambiente. Tuttavia, si prega di notare che i componenti specifici

possono variare a seconda delle esigenze individuali del cliente.

Requisiti hardware

Hardware	Dettagli
Coppia HA di array di storage NetApp AFF	* A800 * ONTAP 9.14.1 * 48 x 3,49 TB SSD-NVM * Due volumi di gruppo flessibili: metadati e dati. * Il volume NFS dei metadati ha 12 volumi persistenti da 250 GB. * I dati sono un volume ONTAP NAS S3
6 x FUJITSU PRIMERGY RX2540 M4	* 64 CPU * CPU Intel® Xeon® Gold 6142 a 2,60 GHz * 256 GM di memoria fisica * 1 porta di rete da 100 GbE
Networking	100 GbE
StorageGRID	* 1 x SG100, 3xSGF6024 * 3 x 24 x 7,68 TB

Requisiti software

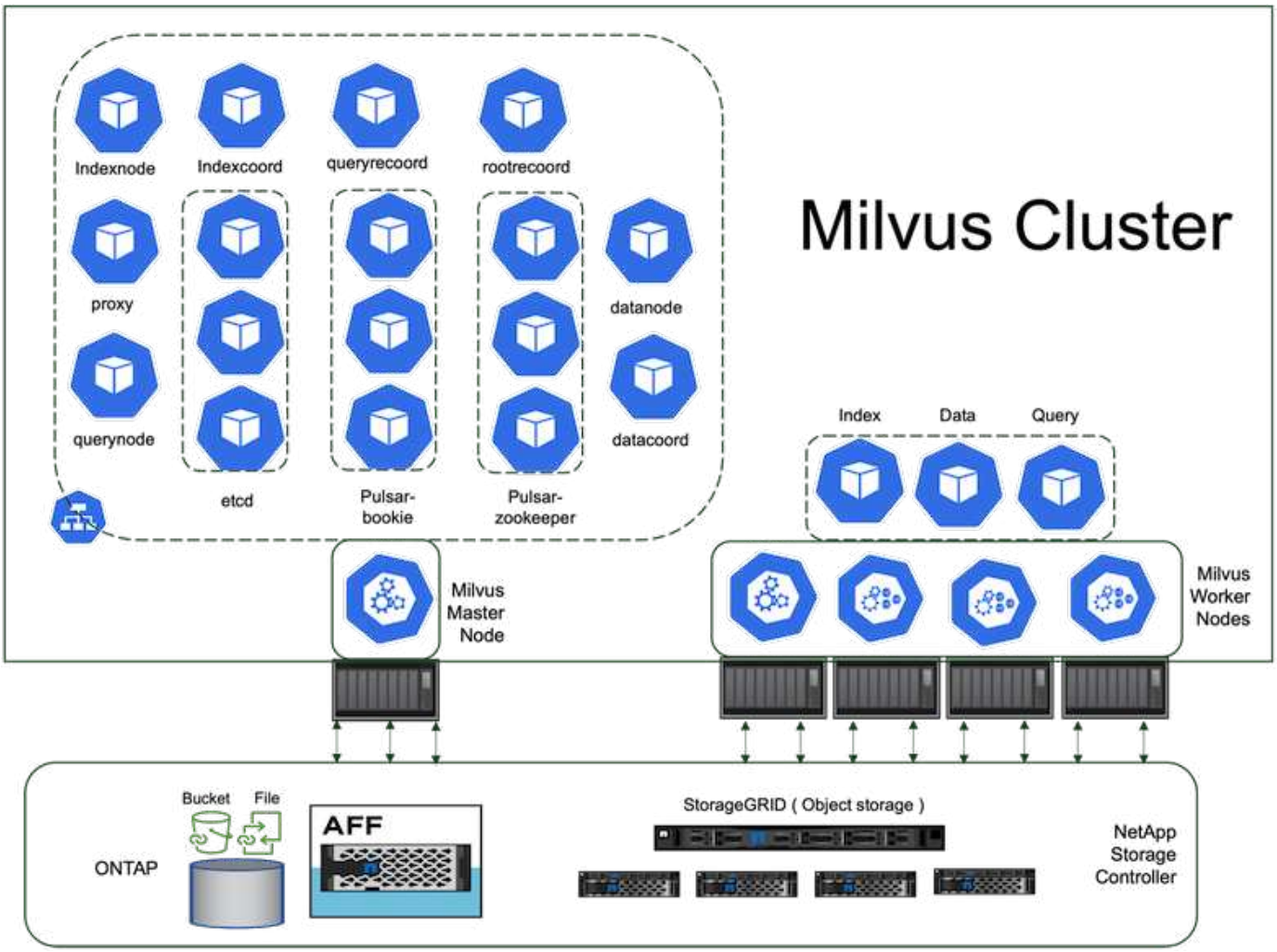
Software	Dettagli
Ammasso di Milvus	* GRAFICO - milvus-4.1.11. * Versione APP – 2.3.4 * Bundle dipendenti come bookkeeper, zookeeper, pulsar, etcd, proxy, querynode, worker
Kubernetes	* Cluster K8s a 5 nodi * 1 nodo master e 4 nodi worker * Versione – 1.7.2
Pitone	*3.10.12.

Procedura di distribuzione

In questa sezione viene illustrata la procedura di distribuzione per la soluzione di database vettoriale per NetApp.

Procedura di distribuzione

In questa sezione di distribuzione, abbiamo utilizzato il database vettoriale Milvus con Kubernetes per la configurazione del laboratorio come di seguito.



NetApp Storage fornisce lo spazio di archiviazione per il cluster in cui conservare i dati dei clienti e i dati del cluster Milvus.

Configurazione dello storage NetApp – ONTAP

- Inizializzazione del sistema di archiviazione
- Creazione di una macchina virtuale di archiviazione (SVM)
- Assegnazione delle interfacce di rete logiche
- Configurazione e licenza NFS, S3

Per NFS (Network File System) seguire i passaggi sottostanti:

1. Creare un volume FlexGroup per NFSv4. Nella nostra configurazione per questa convalida, abbiamo utilizzato 48 SSD, 1 SSD dedicato al volume root del controller e 47 SSD distribuiti per NFSv4. Verificare che la policy di esportazione NFS per il volume FlexGroup disponga di autorizzazioni di lettura/scrittura per la rete dei nodi Kubernetes (K8). Se queste autorizzazioni non sono disponibili, concedere autorizzazioni di lettura/scrittura (rw) per la rete dei nodi K8s.
2. Su tutti i nodi K8s, creare una cartella e montare il volume FlexGroup su questa cartella tramite un'interfaccia logica (LIF) su ciascun nodo K8s.

Per NAS S3 (Network Attached Storage Simple Storage Service), seguire i passaggi indicati di seguito:

1. Creare un volume FlexGroup per NFS.
2. Impostare un object-store-server con HTTP abilitato e lo stato di amministrazione impostato su "attivo" utilizzando il comando "vserver object-store-server create". Hai la possibilità di abilitare HTTPS e impostare una porta di ascolto personalizzata.
3. Creare un utente object-store-server utilizzando il comando "vserver object-store-server user create -user <username>".
4. Per ottenere la chiave di accesso e la chiave segreta, è possibile eseguire il seguente comando: "set diag; vserver object-store-server user show -user <nomeutente>". Tuttavia, in futuro, queste chiavi verranno fornite durante il processo di creazione dell'utente oppure potranno essere recuperate tramite chiamate API REST.
5. Creare un gruppo object-store-server utilizzando l'utente creato nel passaggio 2 e concedere l'accesso. In questo esempio abbiamo fornito "FullAccess".
6. Creare un bucket NAS impostandone il tipo su "nas" e specificando il percorso al volume NFSv3. A questo scopo è anche possibile utilizzare un bucket S3.

Configurazione dell'archiviazione NetApp – StorageGRID

1. Installare il software storageGRID.
2. Crea un tenant e un bucket.
3. Crea un utente con l'autorizzazione richiesta.

Per maggiori dettagli consultare https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

Verifica della soluzione

Panoramica della soluzione

Abbiamo condotto una convalida completa della soluzione incentrata su cinque aree chiave, i cui dettagli sono descritti di seguito. Ogni sezione approfondisce le sfide affrontate dai clienti, le soluzioni fornite da NetApp e i conseguenti vantaggi per il cliente.

1. "Configurazione del cluster Milvus con Kubernetes in locale"Le sfide dei clienti per scalare in modo indipendente su storage e calcolo, gestione efficace dell'infrastruttura e gestione dei dati. In questa sezione, descriviamo in dettaglio il processo di installazione di un cluster Milvus su Kubernetes, utilizzando un controller di archiviazione NetApp sia per i dati del cluster che per i dati dei clienti.
2. Milvus con Amazon FSx ONTAP per NetApp ONTAP – dualità file e oggetto In questa sezione, spiegheremo perché è necessario distribuire un database vettoriale nel cloud e i passaggi per distribuire un database vettoriale (milvus standalone) in Amazon FSx ONTAP per NetApp ONTAP all'interno di container Docker.
3. "Protezione del database vettoriale tramite NetApp SnapCenter."In questa sezione approfondiamo il modo in cui SnapCenter salvaguarda i dati del database vettoriale e i dati Milvus residenti in ONTAP. Per questo esempio, abbiamo utilizzato un bucket NAS (milvusdbvol1) derivato da un volume NFS ONTAP (vol1) per i dati dei clienti e un volume NFS separato (vectordbpv) per i dati di configurazione del cluster Milvus.
4. "Ripristino di emergenza tramite NetApp SnapMirror"In questa sezione, discuteremo dell'importanza del Disaster Recovery (DR) per il database vettoriale e di come il prodotto di Disaster Recovery di NetApp Snapmirror fornisca una soluzione DR per il database vettoriale.
5. "Validazione delle prestazioni"In questa sezione, ci proponiamo di approfondire la convalida delle prestazioni dei database vettoriali, come Milvus e pgvecto.rs, concentrandoci sulle caratteristiche delle

prestazioni di storage, come il profilo I/O e il comportamento del controller di storage NetApp a supporto dei carichi di lavoro RAG e di inferenza all'interno del ciclo di vita LLM. Valuteremo e identificheremo eventuali fattori differenzianti nelle prestazioni quando questi database saranno combinati con la soluzione di archiviazione ONTAP . La nostra analisi si baserà su indicatori chiave di prestazione, come il numero di query elaborate al secondo (QPS).

Configurazione del cluster Milvus con Kubernetes in locale

Questa sezione illustra la configurazione del cluster Milvus per la soluzione di database vettoriale per NetApp.

Configurazione del cluster Milvus con Kubernetes in locale

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

1. In questa sezione, descriviamo in dettaglio il processo di installazione di un cluster Milvus su Kubernetes, utilizzando un controller di archiviazione NetApp sia per i dati del cluster che per i dati dei clienti.
2. Per installare un cluster Milvus, sono necessari volumi persistenti (PV) per archiviare i dati provenienti da vari componenti del cluster Milvus. Questi componenti includono etcd (tre istanze), pulsar-bookie-journal (tre istanze), pulsar-bookie-ledgers (tre istanze) e pulsar-zookeeper-data (tre istanze).

> ⓘ Nel cluster Milvus, possiamo utilizzare sia Pulsar che Kafka come motore sottostante che supporta l'archiviazione affidabile e la pubblicazione/sottoscrizione dei flussi di messaggi del cluster Milvus. Per Kafka con NFS, NetApp ha apportato miglioramenti in ONTAP 9.12.1 e versioni successive, e questi miglioramenti, insieme alle modifiche di NFSv4.1 e Linux incluse in RHEL 8.7 o 9.1 e versioni successive, risolvono il problema della "rinomina stupida" che può verificarsi quando si esegue Kafka su NFS. Se sei interessato a informazioni più approfondite sull'esecuzione di Kafka con la soluzione NetApp NFS, consulta:"questo collegamento" .

3. Abbiamo creato un singolo volume NFS da NetApp ONTAP e stabilito 12 volumi persistenti, ciascuno con 250 GB di storage. La dimensione dello storage può variare a seconda delle dimensioni del cluster; ad esempio, abbiamo un altro cluster in cui ogni PV ha 50 GB. Per maggiori dettagli fare riferimento a uno dei file PV YAML qui sotto; in totale avevamo 12 file di questo tipo. In ogni file, storageClassName è impostato su "default" e lo storage e il percorso sono univoci per ogni PV.

```
root@node2:~# cat sai_nfs_to_default_pv1.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: karthik-pv1
spec:
  capacity:
    storage: 250Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: default
  local:
    path: /vectordbsc/milvus/milvus1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node2
          - node3
          - node4
          - node5
          - node6
root@node2:~#
```

4. Eseguire il comando 'kubectl apply' per ogni file YAML PV per creare i volumi persistenti, quindi verificare la loro creazione utilizzando 'kubectl get pv'

```
root@node2:~# for i in $( seq 1 12 ); do kubectl apply -f
sai_nfs_to_default_pv$i.yaml; done
persistentvolume/karthik-pv1 created
persistentvolume/karthik-pv2 created
persistentvolume/karthik-pv3 created
persistentvolume/karthik-pv4 created
persistentvolume/karthik-pv5 created
persistentvolume/karthik-pv6 created
persistentvolume/karthik-pv7 created
persistentvolume/karthik-pv8 created
persistentvolume/karthik-pv9 created
persistentvolume/karthik-pv10 created
persistentvolume/karthik-pv11 created
persistentvolume/karthik-pv12 created
root@node2:~#
```

5. Per l'archiviazione dei dati dei clienti, Milvus supporta soluzioni di archiviazione di oggetti come MinIO, Azure Blob e S3. In questa guida utilizziamo S3. I passaggi seguenti si applicano sia all'archivio oggetti ONTAP S3 che a StorageGRID . Utilizziamo Helm per distribuire il cluster Milvus. Scarica il file di configurazione, values.yaml, dalla posizione di download di Milvus. Fare riferimento all'appendice per il file values.yaml utilizzato in questo documento.
6. Assicurarsi che 'storageClass' sia impostato su 'default' in ogni sezione, comprese quelle per log, etcd, zookeeper e bookkeeper.
7. Nella sezione MinIO, disabilitare MinIO.
8. Creare un bucket NAS dall'archiviazione di oggetti ONTAP o StorageGRID e includerli in un S3 esterno con le credenziali dell'archiviazione di oggetti.

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

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

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

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

```
root@node2:~# helm upgrade --install my-release milvus/milvus --set
global.storageClass=default  -f values.yaml
Release "my-release" does not exist. Installing it now.
NAME: my-release
LAST DEPLOYED: Thu Mar 14 15:00:07 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
root@node2:~#
```

11. Verificare lo stato dei PersistentVolumeClaims (PVC).

```
root@node2:~# kubectl get pvc
NAME                                                                    STATUS
VOLUME         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-my-release-etcd-0                                                  Bound
karthik-pv8    250Gi      RWO            default        3s
data-my-release-etcd-1                                                  Bound
karthik-pv5    250Gi      RWO            default        2s
data-my-release-etcd-2                                                  Bound
karthik-pv4    250Gi      RWO            default        3s
my-release-pulsar-bookie-journal-my-release-pulsar-bookie-0      Bound
karthik-pv10   250Gi      RWO            default        3s
my-release-pulsar-bookie-journal-my-release-pulsar-bookie-1      Bound
karthik-pv3    250Gi      RWO            default        3s
my-release-pulsar-bookie-journal-my-release-pulsar-bookie-2      Bound
karthik-pv1    250Gi      RWO            default        3s
my-release-pulsar-bookie-ledgers-my-release-pulsar-bookie-0      Bound
karthik-pv2    250Gi      RWO            default        3s
my-release-pulsar-bookie-ledgers-my-release-pulsar-bookie-1      Bound
karthik-pv9    250Gi      RWO            default        3s
my-release-pulsar-bookie-ledgers-my-release-pulsar-bookie-2      Bound
karthik-pv11   250Gi      RWO            default        3s
my-release-pulsar-zookeeper-data-my-release-pulsar-zookeeper-0   Bound
karthik-pv7    250Gi      RWO            default        3s
root@node2:~#
```

12. Controllare lo stato dei baccelli.

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

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

13. Scrittura e lettura dei dati di prova in Milvus e nell'archiviazione di oggetti NetApp .
    - Scrivere i dati utilizzando il programma Python "prepare_data_netapp_new.py".

```
root@node2:~# date;python3 prepare_data_netapp_new.py ;date
Thu Apr  4 04:15:35 PM UTC 2024
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===
Does collection hello_milvus_ntapnew_update2_sc exist in Milvus:
False
=== Drop collection - hello_milvus_ntapnew_update2_sc ===
=== Drop collection - hello_milvus_ntapnew_update2_sc2 ===
=== Create collection `hello_milvus_ntapnew_update2_sc` ===
=== Start inserting entities       ===
Number of entities in hello_milvus_ntapnew_update2_sc: 3000
Thu Apr  4 04:18:01 PM UTC 2024
root@node2:~#
```

- Leggere i dati utilizzando il file Python "verify_data_netapp.py".

```
root@node2:~# python3 verify_data_netapp.py
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===

Does collection hello_milvus_ntapnew_update2_sc exist in Milvus: True
{'auto_id': False, 'description': 'hello_milvus_ntapnew_update2_sc',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64:
5>, 'is_primary': True, 'auto_id': False}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 16}}]}
Number of entities in Milvus: hello_milvus_ntapnew_update2_sc : 3000

=== Start Creating index IVF_FLAT  ===

=== Start loading                  ===

=== Start searching based on vector similarity ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 2600, distance: 0.602496862411499, entity: {'random':
0.3098157043984633}, random field: 0.3098157043984633
hit: id: 1831, distance: 0.6797959804534912, entity: {'random':
0.6331477114129169}, random field: 0.6331477114129169
hit: id: 2999, distance: 0.0, entity: {'random':
0.02316334456872482}, random field: 0.02316334456872482
hit: id: 2524, distance: 0.5918987989425659, entity: {'random':
```

```
0.285283165889066}, random field: 0.285283165889066
hit: id: 264, distance: 0.7254047393798828, entity: {'random':
0.3329096143562196}, random field: 0.3329096143562196
search latency = 0.4533s

=== Start querying with `random > 0.5` ===

query result:
-{'random': 0.6378742006852851, 'embeddings': [0.20963514,
0.39746657, 0.12019053, 0.6947492, 0.9535575, 0.5454552, 0.82360446,
0.21096309, 0.52323616, 0.8035404, 0.77824664, 0.80369574, 0.4914803,
0.8265614, 0.6145269, 0.80234545], 'pk': 0}
search latency = 0.4476s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 1831, distance: 0.6797959804534912, entity: {'random':
0.6331477114129169}, random field: 0.6331477114129169
hit: id: 678, distance: 0.7351570129394531, entity: {'random':
0.5195484662306603}, random field: 0.5195484662306603
hit: id: 2644, distance: 0.8620758056640625, entity: {'random':
0.9785952878381153}, random field: 0.9785952878381153
hit: id: 1960, distance: 0.9083120226860046, entity: {'random':
0.6376039340439571}, random field: 0.6376039340439571
hit: id: 106, distance: 0.9792704582214355, entity: {'random':
0.9679994241326673}, random field: 0.9679994241326673
search latency = 0.1232s
Does collection hello_milvus_ntapnew_update2_sc2 exist in Milvus:
True
{'auto_id': True, 'description': 'hello_milvus_ntapnew_update2_sc2',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64:
5>, 'is_primary': True, 'auto_id': True}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 16}}]}
```

Sulla base della convalida di cui sopra, l'integrazione di Kubernetes con un database vettoriale, come dimostrato attraverso l'implementazione di un cluster Milvus su Kubernetes utilizzando un controller di storage NetApp , offre ai clienti una soluzione solida, scalabile ed efficiente per la gestione di operazioni sui dati su larga scala. Questa configurazione offre ai clienti la possibilità di gestire dati ad alta dimensionalità ed eseguire query complesse in modo rapido ed efficiente, rendendola una soluzione ideale per applicazioni big data e carichi di lavoro di intelligenza artificiale. L'utilizzo di volumi persistenti (PV) per vari componenti del cluster, insieme alla creazione di un singolo volume NFS da NetApp ONTAP, garantisce un utilizzo ottimale delle risorse e una gestione dei dati. Il processo di

verifica dello stato dei PersistentVolumeClaim (PVC) e dei pod, nonché il test di scrittura e lettura dei dati, garantiscono ai clienti operazioni sui dati affidabili e coerenti. L'utilizzo dell'archiviazione di oggetti ONTAP o StorageGRID per i dati dei clienti migliora ulteriormente l'accessibilità e la sicurezza dei dati. Nel complesso, questa configurazione fornisce ai clienti una soluzione di gestione dei dati resiliente e ad alte prestazioni, in grado di adattarsi senza problemi alle crescenti esigenze di dati.

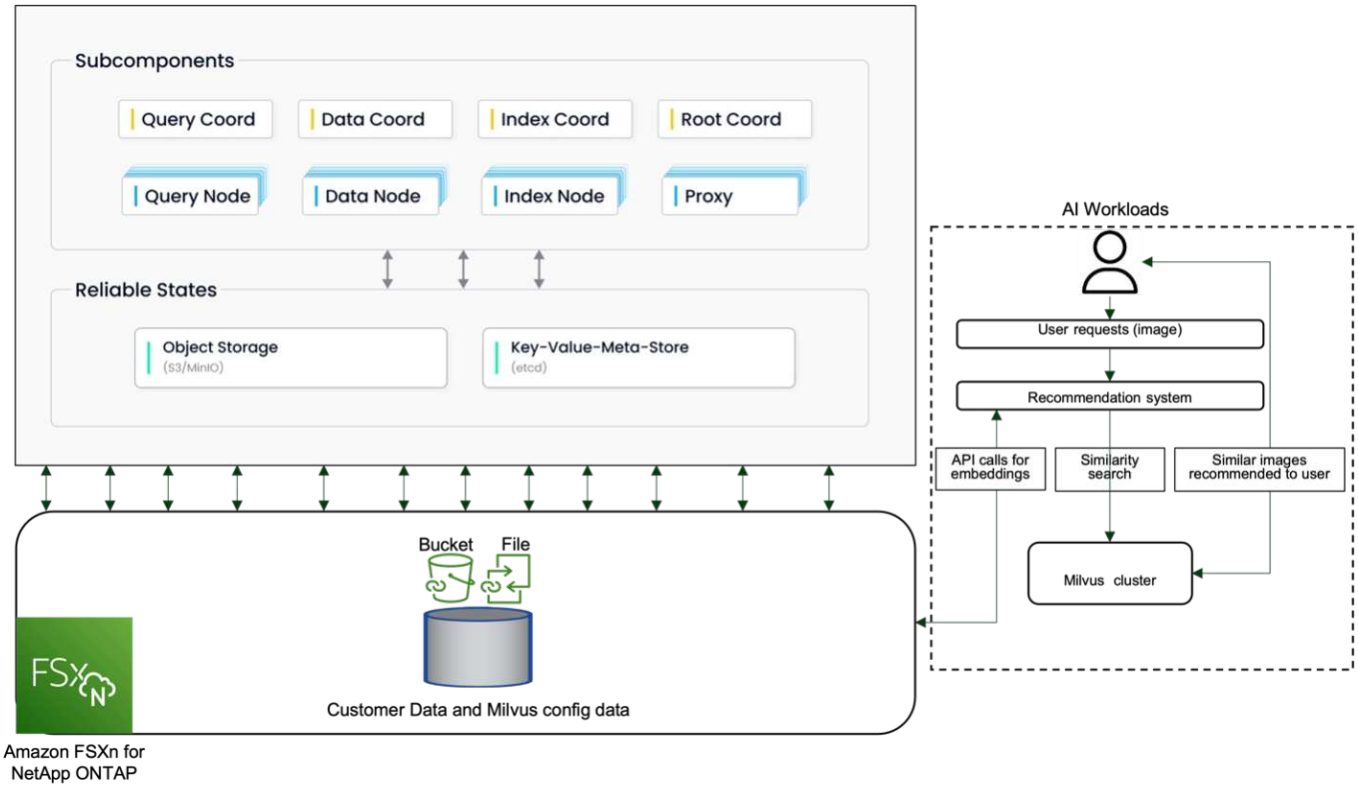
Milvus con Amazon FSx ONTAP per NetApp ONTAP : dualità file e oggetto

Questa sezione illustra la configurazione del cluster Milvus con Amazon FSx ONTAP per la soluzione di database vettoriale per NetApp.

Milvus con Amazon FSx ONTAP per NetApp ONTAP : dualità di file e oggetti

In questa sezione, spiegheremo perché è necessario distribuire un database vettoriale nel cloud e i passaggi per distribuire un database vettoriale (Milvus standalone) in Amazon FSx ONTAP per NetApp ONTAP all'interno di container Docker.

L'implementazione di un database vettoriale nel cloud offre diversi vantaggi significativi, in particolare per le applicazioni che richiedono la gestione di dati ad alta dimensionalità e l'esecuzione di ricerche di similarità. In primo luogo, l'implementazione basata su cloud offre scalabilità, consentendo di adattare facilmente le risorse in base ai crescenti volumi di dati e carichi di query. Ciò garantisce che il database possa gestire in modo efficiente l'aumento della domanda, mantenendo al contempo prestazioni elevate. In secondo luogo, l'implementazione del cloud garantisce elevata disponibilità e ripristino in caso di emergenza, poiché i dati possono essere replicati in diverse posizioni geografiche, riducendo al minimo il rischio di perdita di dati e garantendo un servizio continuo anche in caso di eventi imprevisti. In terzo luogo, garantisce un buon rapporto qualità-prezzo, poiché si paga solo per le risorse utilizzate e si può aumentare o diminuire la scala in base alla domanda, evitando così la necessità di ingenti investimenti iniziali in hardware. Infine, l'implementazione di un database vettoriale nel cloud può migliorare la collaborazione, poiché i dati possono essere consultati e condivisi da qualsiasi luogo, facilitando il lavoro di squadra e il processo decisionale basato sui dati. Verificare l'architettura di milvus standalone con Amazon FSx ONTAP per NetApp ONTAP utilizzato in questa convalida.



1. Creare un'istanza Amazon FSx ONTAP per NetApp ONTAP e annotare i dettagli della VPC, dei gruppi di sicurezza VPC e della subnet. Queste informazioni saranno necessarie durante la creazione di un'istanza EC2. Puoi trovare maggiori dettagli qui - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. Creare un'istanza EC2, assicurandosi che la VPC, i gruppi di sicurezza e la subnet corrispondano a quelli dell'istanza Amazon FSx ONTAP per NetApp ONTAP .
3. Installare nfs-common utilizzando il comando 'apt-get install nfs-common' e aggiornare le informazioni sul pacchetto utilizzando 'sudo apt-get update'.
4. Crea una cartella di montaggio e montaci Amazon FSx ONTAP per NetApp ONTAP .

```
ubuntu@ip-172-31-29-98:~$ mkdir /home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$ sudo mount 172.31.255.228:/vol1
/home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$ df -h /home/ubuntu/milvusvectordb
Filesystem            Size  Used Avail Use% Mounted on
172.31.255.228:/vol1  973G  126G  848G  13% /home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$
```

5. Installa Docker e Docker Compose utilizzando 'apt-get install'.
6. Configurare un cluster Milvus in base al file docker-compose.yaml, scaricabile dal sito web di Milvus.

```
root@ip-172-31-22-245:~# wget https://github.com/milvus-
io/milvus/releases/download/v2.0.2/milvus-standalone-docker-compose.yml
-O docker-compose.yml
--2024-04-01 14:52:23--  https://github.com/milvus-
io/milvus/releases/download/v2.0.2/milvus-standalone-docker-compose.yml
<removed some output to save page space>
```

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

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

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

```
ubuntu@ip-172-31-29-98:~$ sudo docker-compose ps
      Name                     Command                  State
Ports
-------------------------------------------------------------------------------
----------
milvus-etcd         etcd -advertise-client-url ...   Up (healthy)
2379/tcp, 2380/tcp
milvus-minio        /usr/bin/docker-entrypoint ...   Up (healthy)
0.0.0.0:9000->9000/tcp,:::9000->9000/tcp, 0.0.0.0:9001-
>9001/tcp,:::9001->9001/tcp
milvus-standalone   /tini -- milvus run standalone   Up (healthy)
0.0.0.0:19530->19530/tcp,:::19530->19530/tcp, 0.0.0.0:9091-
>9091/tcp,:::9091->9091/tcp
ubuntu@ip-172-31-29-98:~$
ubuntu@ip-172-31-29-98:~$ ls -ltrh /home/ubuntu/milvusvectordb/volumes/
total 12K
drwxr-xr-x 3 root root 4.0K Apr  4 20:21 etcd
drwxr-xr-x 4 root root 4.0K Apr  4 20:21 minio
drwxr-xr-x 5 root root 4.0K Apr  4 20:21 milvus
ubuntu@ip-172-31-29-98:~$
```

11. Per convalidare la funzionalità di lettura e scrittura del database vettoriale e dei suoi dati in Amazon FSx ONTAP per NetApp ONTAP, abbiamo utilizzato Python Milvus SDK e un programma di esempio di PyMilvus. Installa i pacchetti necessari usando 'apt-get install python3-numpy python3-pip' e installa PyMilvus usando 'pip3 install pymilvus'.
12. Convalida le operazioni di scrittura e lettura dei dati da Amazon FSx ONTAP per NetApp ONTAP nel database vettoriale.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3
prepare_data_netapp_new.py
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===
Does collection hello_milvus_ntapnew_sc exist in Milvus: True
=== Drop collection - hello_milvus_ntapnew_sc ===
=== Drop collection - hello_milvus_ntapnew_sc2 ===
=== Create collection `hello_milvus_ntapnew_sc` ===
=== Start inserting entities       ===
Number of entities in hello_milvus_ntapnew_sc: 9000
root@ip-172-31-29-98:~/pymilvus/examples# find
/home/ubuntu/milvusvectordb/
…
<removed content to save page space >
…
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
```

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

13. Controllare l'operazione di lettura utilizzando lo script verify_data_netapp.py.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3 verify_data_netapp.py
=== start connecting to Milvus     ===

=== Milvus host: localhost         ===

Does collection hello_milvus_ntapnew_sc exist in Milvus: True
{'auto_id': False, 'description': 'hello_milvus_ntapnew_sc', 'fields':
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,
'is_primary': True, 'auto_id': False}, {'name': 'random', 'description':
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',
```

```
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},
{'name': 'embeddings', 'description': '', 'type': <DataType.
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':
False}
Number of entities in Milvus: hello_milvus_ntapnew_sc : 9000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: id: 2248, distance: 0.0, entity: {'random': 0.2777646777746381},
random field: 0.2777646777746381
hit: id: 4837, distance: 0.07805602252483368, entity: {'random':
0.6451650959930306}, random field: 0.6451650959930306
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':
0.6141351712303128}, random field: 0.6141351712303128
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},
random field: 0.7434908973629817
hit: id: 830, distance: 0.05628090724349022, entity: {'random':
0.8544487225667627}, random field: 0.8544487225667627
hit: id: 8562, distance: 0.07971227169036865, entity: {'random':
0.4464554280115878}, random field: 0.4464554280115878
search latency = 0.1266s

=== Start querying with `random > 0.5` ===

query result:
-{'random': 0.6378742006852851, 'embeddings': [0.3017092, 0.74452263,
0.8009826, 0.4927033, 0.12762444, 0.29869467, 0.52859956, 0.23734547],
'pk': 0}
search latency = 0.3294s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 4837, distance: 0.07805602252483368, entity: {'random':
0.6451650959930306}, random field: 0.6451650959930306
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':
0.6141351712303128}, random field: 0.6141351712303128
hit: id: 515, distance: 0.09590047597885132, entity: {'random':
0.8013175797590888}, random field: 0.8013175797590888
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},
random field: 0.7434908973629817
hit: id: 830, distance: 0.05628090724349022, entity: {'random':
```

```
0.8544487225667627}, random field: 0.8544487225667627
hit: id: 1627, distance: 0.08096684515476227, entity: {'random':
0.9302397069516164}, random field: 0.9302397069516164
search latency = 0.2674s
Does collection hello_milvus_ntapnew_sc2 exist in Milvus: True
{'auto_id': True, 'description': 'hello_milvus_ntapnew_sc2', 'fields':
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,
'is_primary': True, 'auto_id': True}, {'name': 'random', 'description':
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},
{'name': 'embeddings', 'description': '', 'type': <DataType.
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':
False}
```

14. Se il cliente desidera accedere (leggere) i dati NFS testati nel database vettoriale tramite il protocollo S3 per i carichi di lavoro di intelligenza artificiale, può convalidarli utilizzando un semplice programma Python. Un esempio potrebbe essere una ricerca di similarità di immagini provenienti da un'altra applicazione, come indicato nell'immagine all'inizio di questa sezione.

```
root@ip-172-31-29-98:~/pymilvus/examples# sudo python3
/home/ubuntu/milvusvectordb/s3_access.py -i 172.31.255.228 --bucket
milvusnasvol --access-key PY6UF318996I86NBYNDD --secret-key
hoPctr9aD88c1j0SkIYZ2uPa03vlbqKA0c5feK6F
OBJECTS in the bucket milvusnasvol are :
**************************************
…
<output content removed to save page space>
…
bucket/files/insert_log/448789845791611912/448789845791611913/4487898457
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#
```

Questa sezione illustra in modo efficace come i clienti possono distribuire e gestire una configurazione Milvus autonoma all'interno di container Docker, utilizzando NetApp FSx ONTAP di Amazon per l'archiviazione dei dati NetApp ONTAP . Questa configurazione consente ai clienti di sfruttare la potenza dei database vettoriali per gestire dati ad alta dimensionalità ed eseguire query complesse, il tutto all'interno dell'ambiente scalabile ed efficiente dei container Docker. Creando un'istanza Amazon FSx ONTAP per NetApp ONTAP e un'istanza EC2 corrispondente, i clienti possono garantire un utilizzo ottimale delle risorse e una gestione dei dati. La validazione riuscita delle operazioni di scrittura e lettura dei dati da FSx ONTAP nel database vettoriale offre ai clienti la garanzia di operazioni sui dati affidabili e coerenti. Inoltre, la possibilità di elencare (leggere) i dati dai carichi di lavoro di intelligenza artificiale tramite il protocollo S3 offre una migliore accessibilità ai dati. Questo processo completo, pertanto, fornisce ai clienti una soluzione solida ed efficiente per la gestione delle loro operazioni sui dati su larga scala, sfruttando le capacità di FSx ONTAP di Amazon per NetApp ONTAP.

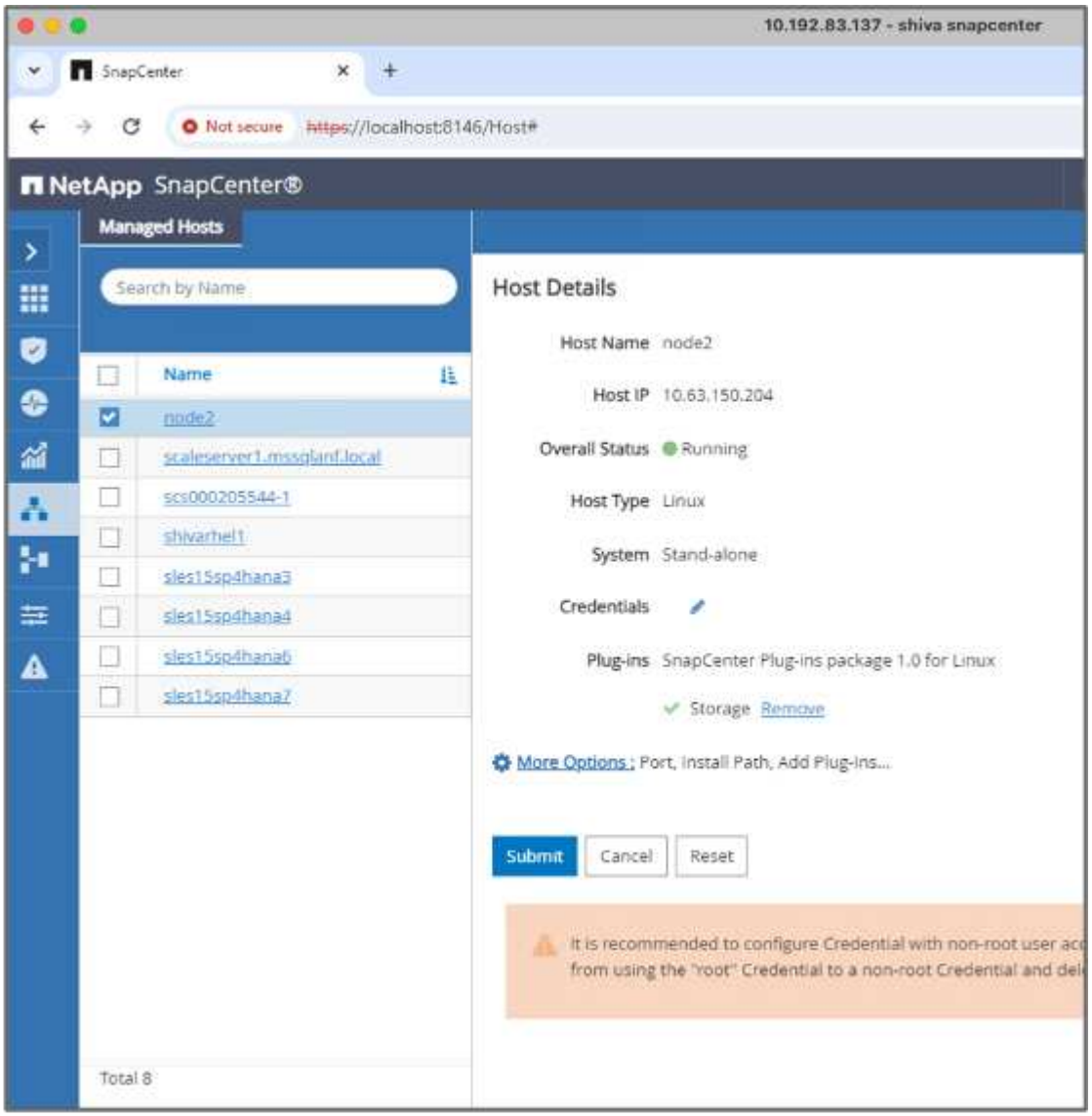
Protezione del database vettoriale tramite SnapCenter

Questa sezione descrive come fornire protezione dei dati per il database vettoriale utilizzando NetApp SnapCenter.

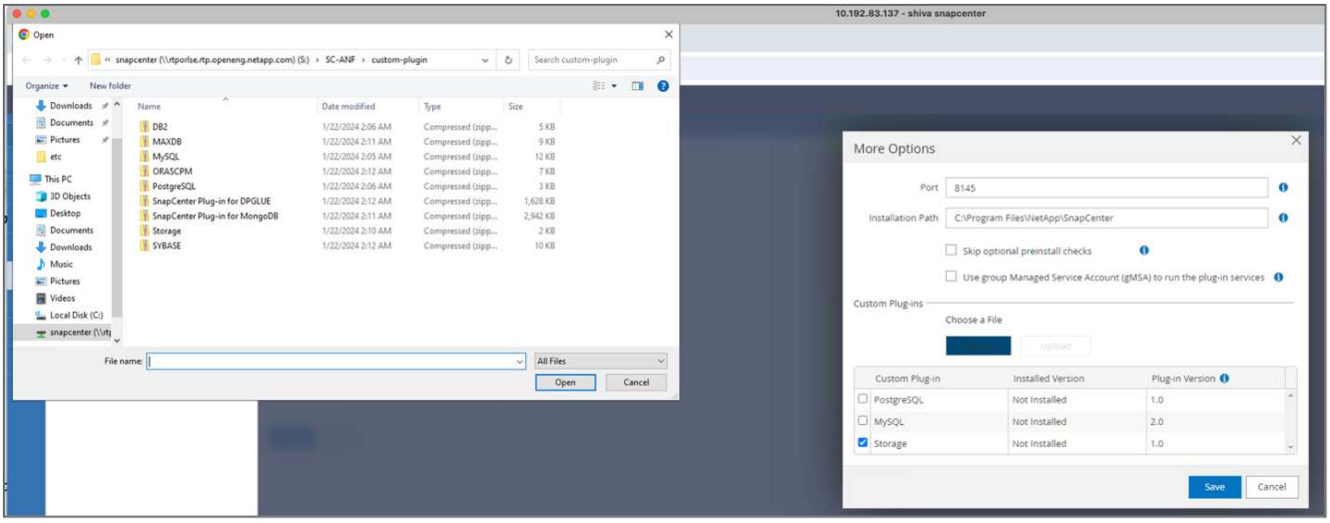
Protezione del database vettoriale tramite NetApp SnapCenter.

Ad esempio, nel settore della produzione cinematografica, i clienti spesso possiedono dati critici incorporati, come file video e audio. La perdita di questi dati, dovuta a problemi come guasti al disco rigido, può avere un impatto significativo sulle loro attività, mettendo potenzialmente a repentaglio iniziative multimilionarie. Ci siamo imbattuti in casi in cui contenuti di inestimabile valore sono andati persi, causando notevoli disagi e perdite finanziarie. Garantire la sicurezza e l'integrità di questi dati essenziali è quindi di fondamentale importanza in questo settore. In questa sezione approfondiamo il modo in cui SnapCenter salvaguarda i dati del database vettoriale e i dati Milvus residenti in ONTAP. Per questo esempio, abbiamo utilizzato un bucket NAS (milvusdbvol1) derivato da un volume NFS ONTAP (vol1) per i dati dei clienti e un volume NFS separato (vectordbpv) per i dati di configurazione del cluster Milvus. Si prega di controllare"Qui" per il flusso di lavoro di backup di SnapCenter

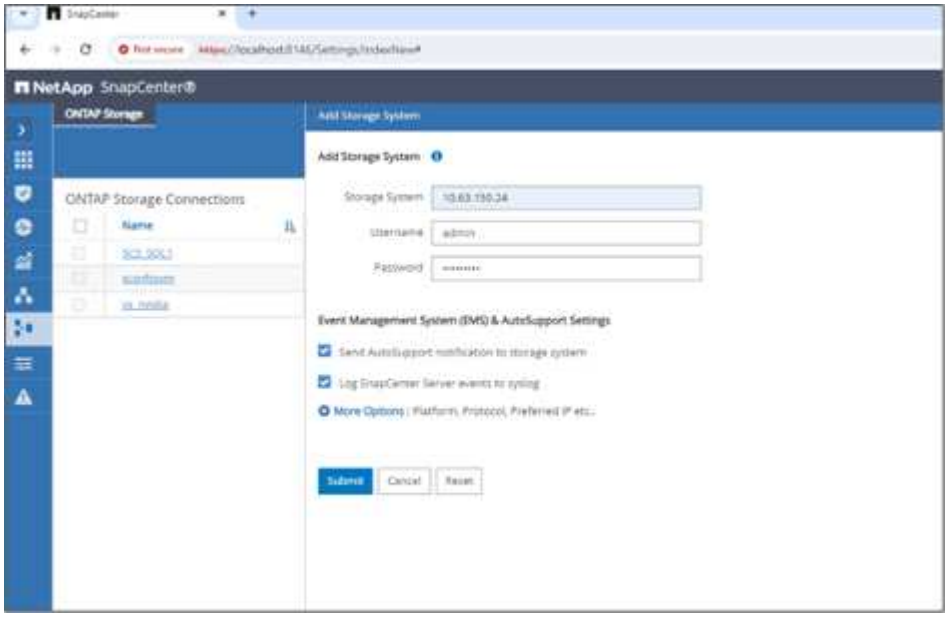
1. Impostare l'host che verrà utilizzato per eseguire i comandi SnapCenter .



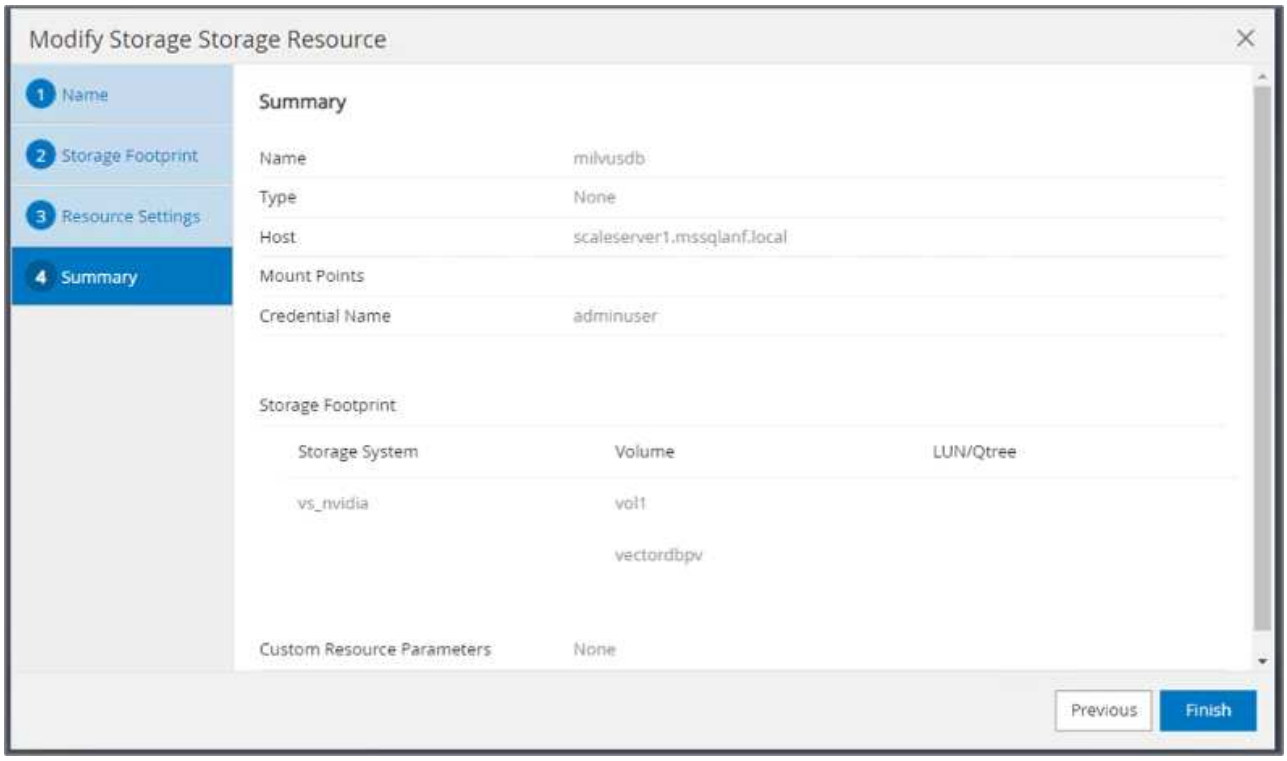
2. Installa e configura il plugin di archiviazione. Dall'host aggiunto, seleziona "Altre opzioni". Passare e selezionare il plug-in di archiviazione scaricato da"Negozio di automazione NetApp" . Installa il plugin e salva la configurazione.



3. Impostare il sistema di archiviazione e il volume: aggiungere il sistema di archiviazione in "Sistema di archiviazione" e selezionare SVM (Storage Virtual Machine). In questo esempio abbiamo scelto "vs_nvidia".



4. Stabilire una risorsa per il database vettoriale, incorporando una politica di backup e un nome di snapshot personalizzato.
   - Abilita il backup del gruppo di coerenza con i valori predefiniti e abilita SnapCenter senza coerenza del file system.
   - Nella sezione Impronta di archiviazione, selezionare i volumi associati ai dati dei clienti del database vettoriale e ai dati del cluster Milvus. Nel nostro esempio, si tratta di "vol1" e "vectordbpv".
   - Creare una policy per la protezione del database vettoriale e proteggere le risorse del database vettoriale utilizzando la policy.



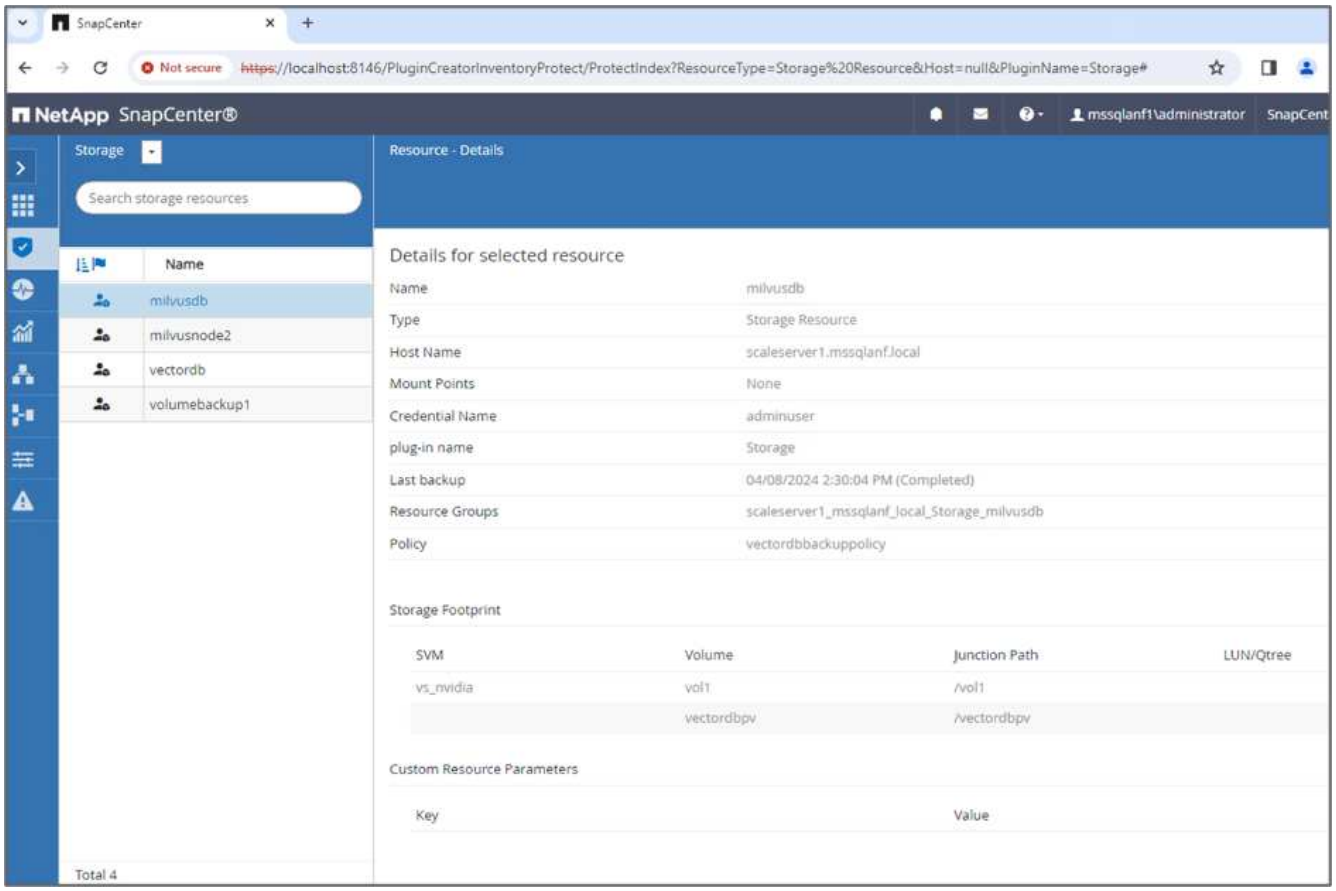
5. Inserire i dati nel bucket NAS S3 utilizzando uno script Python. Nel nostro caso, abbiamo modificato lo script di backup fornito da Milvus, ovvero 'prepare_data_netapp.py', ed eseguito il comando 'sync' per eliminare i dati dal sistema operativo.

```
root@node2:~# python3 prepare_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_test` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_test: 3000

=== Create collection `hello_milvus_netapp_sc_test2` ===

Number of entities in hello_milvus_netapp_sc_test2: 6000
root@node2:~# for i in 2 3 4 5 6   ; do ssh node$i "hostname; sync; echo
'sync executed';" ; done
node2
sync executed
node3
sync executed
node4
sync executed
node5
sync executed
node6
sync executed
root@node2:~#
```

6. Verificare i dati nel bucket NAS S3. Nel nostro esempio, i file con timestamp '2024-04-08 21:22' sono stati creati dallo script 'prepare_data_netapp.py'.

```
root@node2:~# aws s3 ls --profile ontaps3  s3://milvusdbvol1/
--recursive | grep '2024-04-08'

<output content removed to save page space>
2024-04-08 21:18:14       5656
stats_log/448950615991000809/448950615991000810/448950615991001854/100/1
2024-04-08 21:18:12       5654
stats_log/448950615991000809/448950615991000810/448950615991001854/100/4
48950615990800869
2024-04-08 21:18:17       5656
stats_log/448950615991000809/448950615991000810/448950615991001872/100/1
2024-04-08 21:18:15       5654
stats_log/448950615991000809/448950615991000810/448950615991001872/100/4
48950615990800876
2024-04-08 21:22:46       5625
stats_log/448950615991003377/448950615991003378/448950615991003385/100/1
2024-04-08 21:22:45       5623
stats_log/448950615991003377/448950615991003378/448950615991003385/100/4
48950615990800899
2024-04-08 21:22:49       5656
stats_log/448950615991003408/448950615991003409/448950615991003416/100/1
2024-04-08 21:22:47       5654
stats_log/448950615991003408/448950615991003409/448950615991003416/100/4
48950615990800906
2024-04-08 21:22:52       5656
stats_log/448950615991003408/448950615991003409/448950615991003434/100/1
2024-04-08 21:22:50       5654
stats_log/448950615991003408/448950615991003409/448950615991003434/100/4
48950615990800913
root@node2:~#
```

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

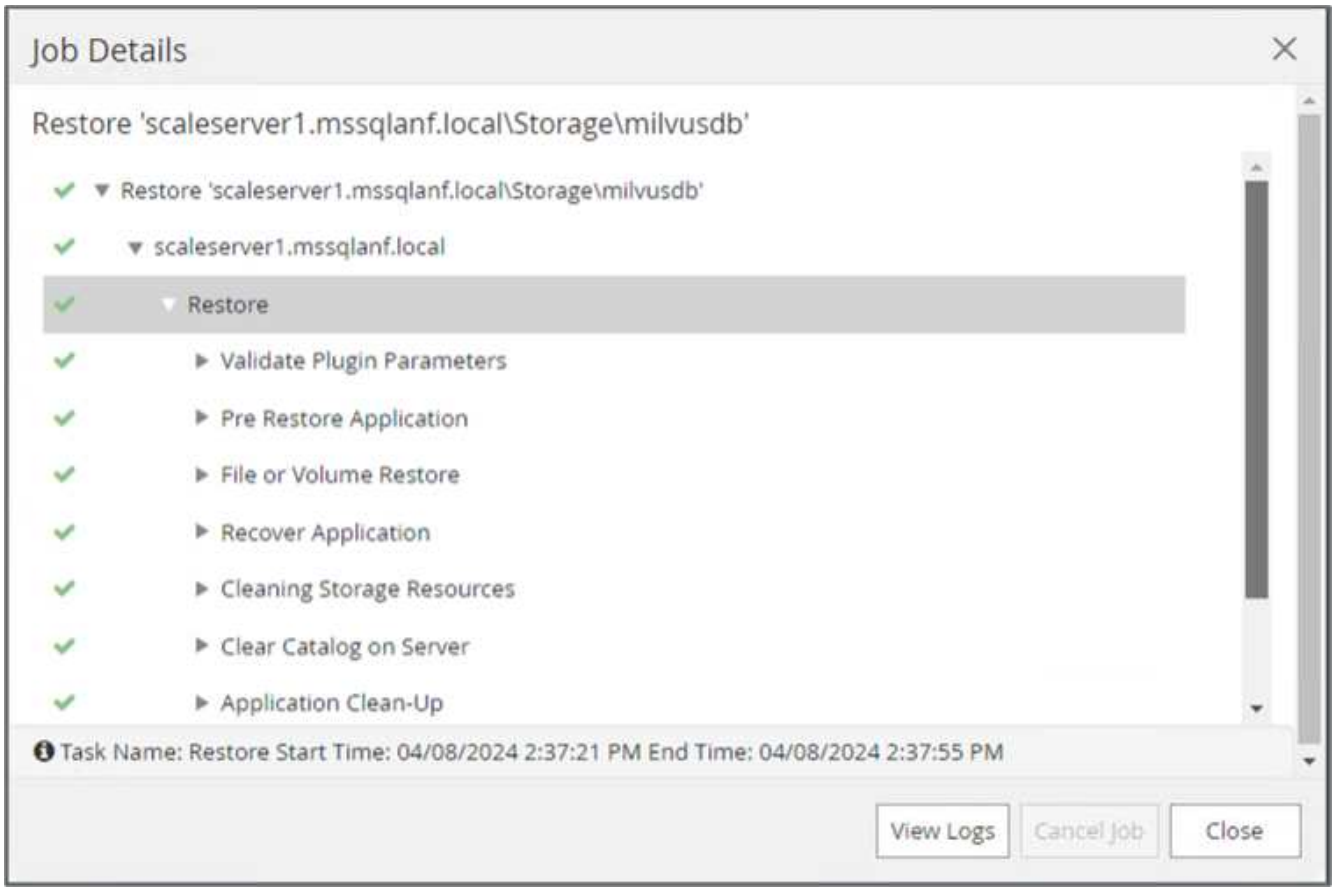


8. Per testare la funzionalità di backup, abbiamo aggiunto una nuova tabella dopo il processo di backup oppure abbiamo rimosso alcuni dati dall'NFS (bucket NAS S3).

   Per questo test, immagina uno scenario in cui qualcuno ha creato una nuova raccolta non necessaria o inappropriata dopo il backup. In tal caso, dovremmo ripristinare il database vettoriale allo stato in cui si trovava prima dell'aggiunta della nuova raccolta. Ad esempio, sono state inserite nuove raccolte come 'hello_milvus_netapp_sc_testnew' e 'hello_milvus_netapp_sc_testnew2'.

```
root@node2:~# python3 prepare_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_testnew` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

Number of entities in hello_milvus_netapp_sc_testnew2: 6000
root@node2:~#
```

9. Eseguire un ripristino completo del bucket NAS S3 dallo snapshot precedente.



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

```
root@node2:~# python3 verify_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: True
{'auto_id': False, 'description': 'hello_milvus_netapp_sc_test',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5
>, 'is_primary': True, 'auto_id': False}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 8}}]}
Number of entities in Milvus: hello_milvus_netapp_sc_test : 3000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 1262, distance: 0.08883658051490784, entity: {'random':
0.2978858685751561}, random field: 0.2978858685751561
hit: id: 1265, distance: 0.09590047597885132, entity: {'random':
0.3042039939240304}, random field: 0.3042039939240304
hit: id: 2999, distance: 0.0, entity: {'random': 0.02316334456872482},
random field: 0.02316334456872482
hit: id: 1580, distance: 0.05628091096878052, entity: {'random':
0.3855988746044062}, random field: 0.3855988746044062
hit: id: 2377, distance: 0.08096685260534286, entity: {'random':
0.8745922204004368}, random field: 0.8745922204004368
search latency = 0.2832s

=== Start querying with `random > 0.5` ===

query result:
-{'random': 0.6378742006852851, 'embeddings': [0.20963514, 0.39746657,
```

```
0.12019053, 0.6947492, 0.9535575, 0.5454552, 0.82360446, 0.21096309],
'pk': 0}
search latency = 0.2257s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 2998, distance: 0.0, entity: {'random': 0.9728033590489911},
random field: 0.9728033590489911
hit: id: 747, distance: 0.14606499671936035, entity: {'random':
0.5648774800635661}, random field: 0.5648774800635661
hit: id: 2527, distance: 0.1530652642250061, entity: {'random':
0.8928974315571507}, random field: 0.8928974315571507
hit: id: 2377, distance: 0.08096685260534286, entity: {'random':
0.8745922204004368}, random field: 0.8745922204004368
hit: id: 2034, distance: 0.20354536175727844, entity: {'random':
0.5526117606328499}, random field: 0.5526117606328499
hit: id: 958, distance: 0.21908017992973328, entity: {'random':
0.6647383716417955}, random field: 0.6647383716417955
search latency = 0.5480s
Does collection hello_milvus_netapp_sc_test2 exist in Milvus: True
{'auto_id': True, 'description': 'hello_milvus_netapp_sc_test2',
'fields': [{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5
>, 'is_primary': True, 'auto_id': True}, {'name': 'random',
'description': '', 'type': <DataType.DOUBLE: 11>}, {'name': 'var',
'description': '', 'type': <DataType.VARCHAR: 21>, 'params':
{'max_length': 65535}}, {'name': 'embeddings', 'description': '',
'type': <DataType.FLOAT_VECTOR: 101>, 'params': {'dim': 8}}]}
Number of entities in Milvus: hello_milvus_netapp_sc_test2 : 6000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: id: 448950615990642008, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990645009, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990640618, distance: 0.13562293350696564, entity:
{'random': 0.7864676926688837}, random field: 0.7864676926688837
hit: id: 448950615990642314, distance: 0.10414951294660568, entity:
{'random': 0.2209597460821181}, random field: 0.2209597460821181
hit: id: 448950615990645315, distance: 0.10414951294660568, entity:
```

```
{'random': 0.2209597460821181}, random field: 0.2209597460821181
hit: id: 448950615990640004, distance: 0.11571306735277176, entity:
{'random': 0.7765521996186631}, random field: 0.7765521996186631
search latency = 0.2381s

=== Start querying with `random > 0.5` ===

query result:
-{'embeddings': [0.15983285, 0.72214717, 0.7414838, 0.44471496,
0.50356466, 0.8750043, 0.316556, 0.7871702], 'pk': 448950615990639798,
'random': 0.7820620141382767}
search latency = 0.3106s

=== Start hybrid searching with `random > 0.5` ===

hit: id: 448950615990642008, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990645009, distance: 0.07805602252483368, entity:
{'random': 0.5326684390871348}, random field: 0.5326684390871348
hit: id: 448950615990640618, distance: 0.13562293350696564, entity:
{'random': 0.7864676926688837}, random field: 0.7864676926688837
hit: id: 448950615990640004, distance: 0.11571306735277176, entity:
{'random': 0.7765521996186631}, random field: 0.7765521996186631
hit: id: 448950615990643005, distance: 0.11571306735277176, entity:
{'random': 0.7765521996186631}, random field: 0.7765521996186631
hit: id: 448950615990640402, distance: 0.13665105402469635, entity:
{'random': 0.9742541034109935}, random field: 0.9742541034109935
search latency = 0.4906s
root@node2:~#
```

11. Verificare che la raccolta non necessaria o inappropriata non sia più presente nel database.

```
root@node2:~# python3 verify_data_netapp.py

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False
Traceback (most recent call last):
  File "/root/verify_data_netapp.py", line 37, in <module>
    recover_collection = Collection(recover_collection_name)
  File "/usr/local/lib/python3.10/dist-
packages/pymilvus/orm/collection.py", line 137, in __init__
    raise SchemaNotReadyException(
pymilvus.exceptions.SchemaNotReadyException: <SchemaNotReadyException:
(code=1, message=Collection 'hello_milvus_netapp_sc_testnew' not exist,
or you can pass in schema to create one.)>
root@node2:~#
```

In conclusione, l'utilizzo di SnapCenter di NetApp per salvaguardare i dati del database vettoriale e i dati Milvus residenti in ONTAP offre notevoli vantaggi ai clienti, in particolare nei settori in cui l'integrità dei dati è fondamentale, come la produzione cinematografica. La capacità di SnapCenter di creare backup coerenti ed eseguire ripristini completi dei dati garantisce che i dati critici, come i file video e audio incorporati, siano protetti da perdite dovute a guasti del disco rigido o altri problemi. Ciò non solo previene interruzioni operative, ma tutela anche da perdite finanziarie sostanziali.

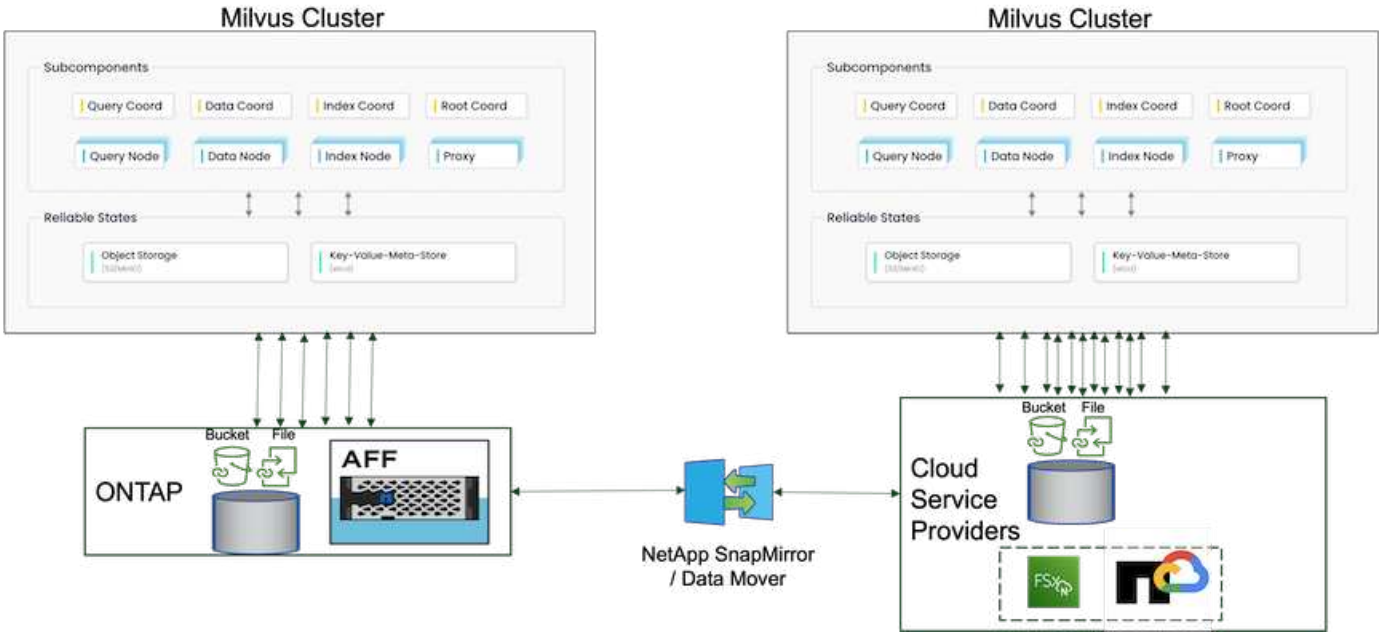
In questa sezione abbiamo dimostrato come configurare SnapCenter per proteggere i dati residenti in ONTAP, inclusa la configurazione degli host, l'installazione e la configurazione dei plugin di archiviazione e la creazione di una risorsa per il database vettoriale con un nome snapshot personalizzato. Abbiamo anche mostrato come eseguire un backup utilizzando lo snapshot del Consistency Group e verificare i dati nel bucket NAS S3.

Inoltre, abbiamo simulato uno scenario in cui è stata creata una raccolta non necessaria o inappropriata dopo il backup. In questi casi, la capacità di SnapCenter di eseguire un ripristino completo da uno snapshot precedente garantisce che il database vettoriale possa essere ripristinato allo stato in cui si trovava prima dell'aggiunta della nuova raccolta, mantenendo così l'integrità del database. Questa capacità di ripristinare i dati a un punto specifico nel tempo è di inestimabile valore per i clienti, poiché fornisce loro la garanzia che i loro dati non solo sono protetti, ma anche correttamente mantenuti. Pertanto, il prodotto SnapCenter di NetApp offre ai clienti una soluzione solida e affidabile per la protezione e la gestione dei dati.

Ripristino di emergenza tramite NetApp SnapMirror

Questa sezione illustra il DR (disaster recovery) con SnapMirror per la soluzione di database vettoriale per NetApp.

Ripristino di emergenza tramite NetApp SnapMirror



Il ripristino dopo un disastro è fondamentale per mantenere l'integrità e la disponibilità di un database vettoriale, soprattutto considerando il suo ruolo nella gestione di dati ad alta dimensionalità e nell'esecuzione di ricerche di similarità complesse. Una strategia di disaster recovery ben pianificata e implementata garantisce che i dati non vengano persi o compromessi in caso di incidenti imprevisti, come guasti hardware, calamità naturali o attacchi informatici. Ciò è particolarmente significativo per le applicazioni che si basano su database vettoriali, in cui la perdita o il danneggiamento dei dati potrebbe causare notevoli interruzioni operative e perdite finanziarie. Inoltre, un solido piano di disaster recovery garantisce la continuità aziendale riducendo al minimo i tempi di inattività e consentendo il rapido ripristino dei servizi. Ciò è possibile grazie al prodotto di replicazione dei dati NetApp SnapMirrror in diverse posizioni geografiche, backup regolari e meccanismi di failover. Pertanto, il disaster recovery non è solo una misura protettiva, ma una componente fondamentale per una gestione responsabile ed efficiente dei database vettoriali.

SnapMirror di NetApp consente la replica dei dati da un controller di storage NetApp ONTAP a un altro, ed è utilizzato principalmente per soluzioni ibride e di disaster recovery (DR). Nel contesto di un database vettoriale, questo strumento facilita la transizione fluida dei dati tra ambienti on-premise e cloud. Questa transizione viene realizzata senza richiedere alcuna conversione di dati o refactoring delle applicazioni, migliorando così l'efficienza e la flessibilità della gestione dei dati su più piattaforme.

La soluzione NetApp Hybrid in uno scenario di database vettoriale può apportare ulteriori vantaggi:

1. Scalabilità: la soluzione cloud ibrida di NetApp offre la possibilità di scalare le risorse in base alle proprie esigenze. È possibile utilizzare risorse on-premise per carichi di lavoro regolari e prevedibili e risorse cloud come Amazon FSx ONTAP per NetApp ONTAP e Google Cloud NetApp Volume (NetApp Volumes) per periodi di punta o carichi imprevisti.
2. Efficienza dei costi: il modello di cloud ibrido di NetApp consente di ottimizzare i costi utilizzando risorse on-premise per carichi di lavoro regolari e pagando le risorse cloud solo quando ne hai bisogno. Questo modello di pagamento a consumo può rivelarsi piuttosto conveniente con un'offerta di servizi NetApp Instaclustr. Instaclustr offre supporto e consulenza ai principali fornitori di servizi cloud e on-premise.
3. Flessibilità: il cloud ibrido di NetApp ti offre la flessibilità di scegliere dove elaborare i tuoi dati. Ad esempio, potresti scegliere di eseguire operazioni vettoriali complesse in locale, dove hai hardware più potente, e operazioni meno intensive nel cloud.
4. Continuità aziendale: in caso di disastro, avere i dati in un cloud ibrido NetApp può garantire la continuità aziendale. Se le tue risorse locali sono interessate, puoi passare rapidamente al cloud. Possiamo sfruttare NetApp SnapMirror per spostare i dati da locale a cloud e viceversa.

5. Innovazione: le soluzioni cloud ibride di NetApp possono anche favorire un'innovazione più rapida, offrendo accesso a servizi e tecnologie cloud all'avanguardia. Le innovazioni NetApp nel cloud, come Amazon FSx ONTAP per NetApp ONTAP, Azure NetApp Files e Google Cloud NetApp Volumes, sono prodotti innovativi e NAS preferiti dai provider di servizi cloud.

Validazione delle prestazioni del database vettoriale

Questa sezione evidenzia la convalida delle prestazioni eseguita sul database vettoriale.

Validazione delle prestazioni

La convalida delle prestazioni svolge un ruolo fondamentale sia nei database vettoriali che nei sistemi di archiviazione, fungendo da fattore chiave per garantire un funzionamento ottimale e un utilizzo efficiente delle risorse. I database vettoriali, noti per la gestione di dati ad alta dimensionalità e l'esecuzione di ricerche di similarità, devono mantenere elevati livelli di prestazioni per elaborare query complesse in modo rapido e accurato. La convalida delle prestazioni aiuta a identificare i colli di bottiglia, a perfezionare le configurazioni e a garantire che il sistema possa gestire i carichi previsti senza degradazione del servizio. Allo stesso modo, nei sistemi di archiviazione, la convalida delle prestazioni è essenziale per garantire che i dati vengano archiviati e recuperati in modo efficiente, senza problemi di latenza o colli di bottiglia che potrebbero influire sulle prestazioni complessive del sistema. Aiuta inoltre a prendere decisioni consapevoli sugli aggiornamenti o le modifiche necessarie all'infrastruttura di storage. Pertanto, la convalida delle prestazioni è un aspetto cruciale della gestione del sistema, contribuendo in modo significativo al mantenimento di un'elevata qualità del servizio, dell'efficienza operativa e dell'affidabilità complessiva del sistema.

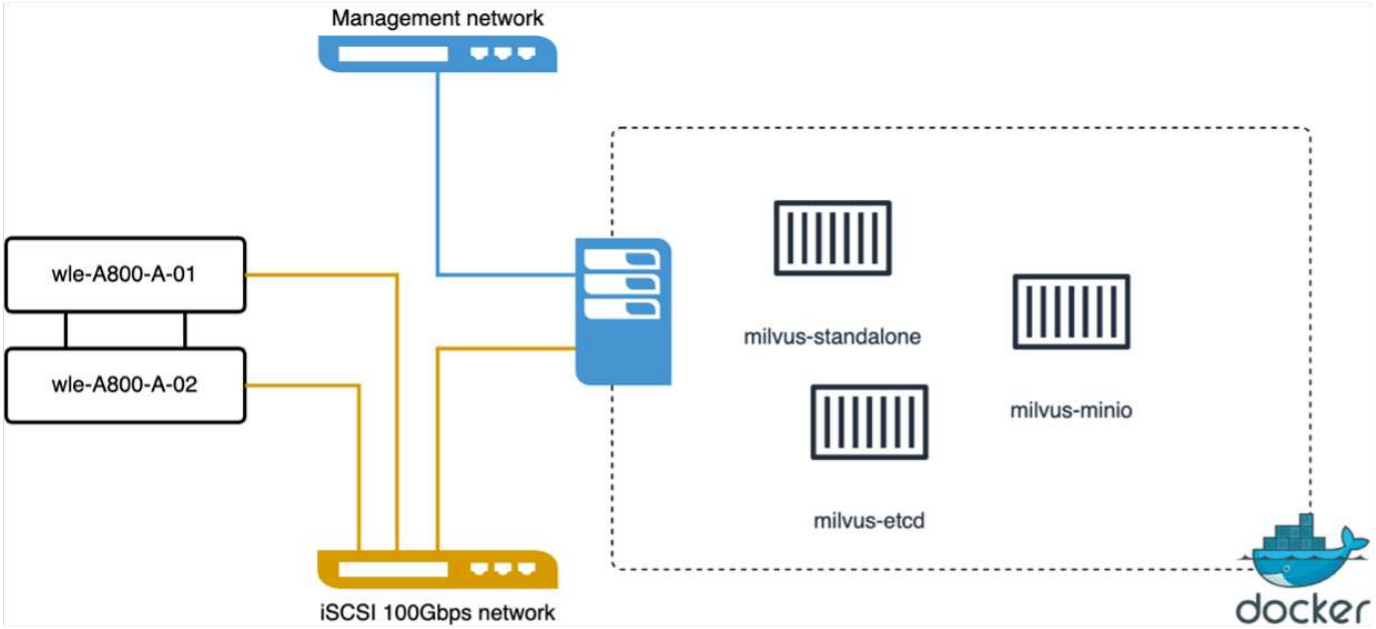
In questa sezione, ci proponiamo di approfondire la convalida delle prestazioni dei database vettoriali, come Milvus e pgvecto.rs, concentrandoci sulle caratteristiche delle prestazioni di storage, come il profilo I/O e il comportamento del controller di storage NetApp a supporto dei carichi di lavoro RAG e di inferenza all'interno del ciclo di vita LLM. Valuteremo e identificheremo eventuali fattori differenzianti nelle prestazioni quando questi database saranno combinati con la soluzione di archiviazione ONTAP . La nostra analisi si baserà su indicatori chiave di prestazione, come il numero di query elaborate al secondo (QPS).

Si prega di controllare la metodologia utilizzata per milvus e i progressi di seguito.

Dettagli	Milvus (autonomo e cluster)	Postgres(pgvecto.rs) #
versione	2.3.2	0.2.0
File system	XFS su LUN iSCSI	
Generatore di carico di lavoro	"VectorDB-Bench"– v0.0.5	
Set di dati	Dataset LAION * 10 milioni di incorporamenti * 768 dimensioni * dimensione del dataset ~300 GB	
Controllore di archiviazione	AFF 800 * Versione – 9.14.1 * 4 x 100GbE – per milvus e 2x 100GbE per postgres * iscsi	

VectorDB-Bench con cluster autonomo Milvus

abbiamo eseguito la seguente convalida delle prestazioni sul cluster autonomo Milvus con vectorDB-Bench. Di seguito è riportata la connettività di rete e server del cluster autonomo Milvus.



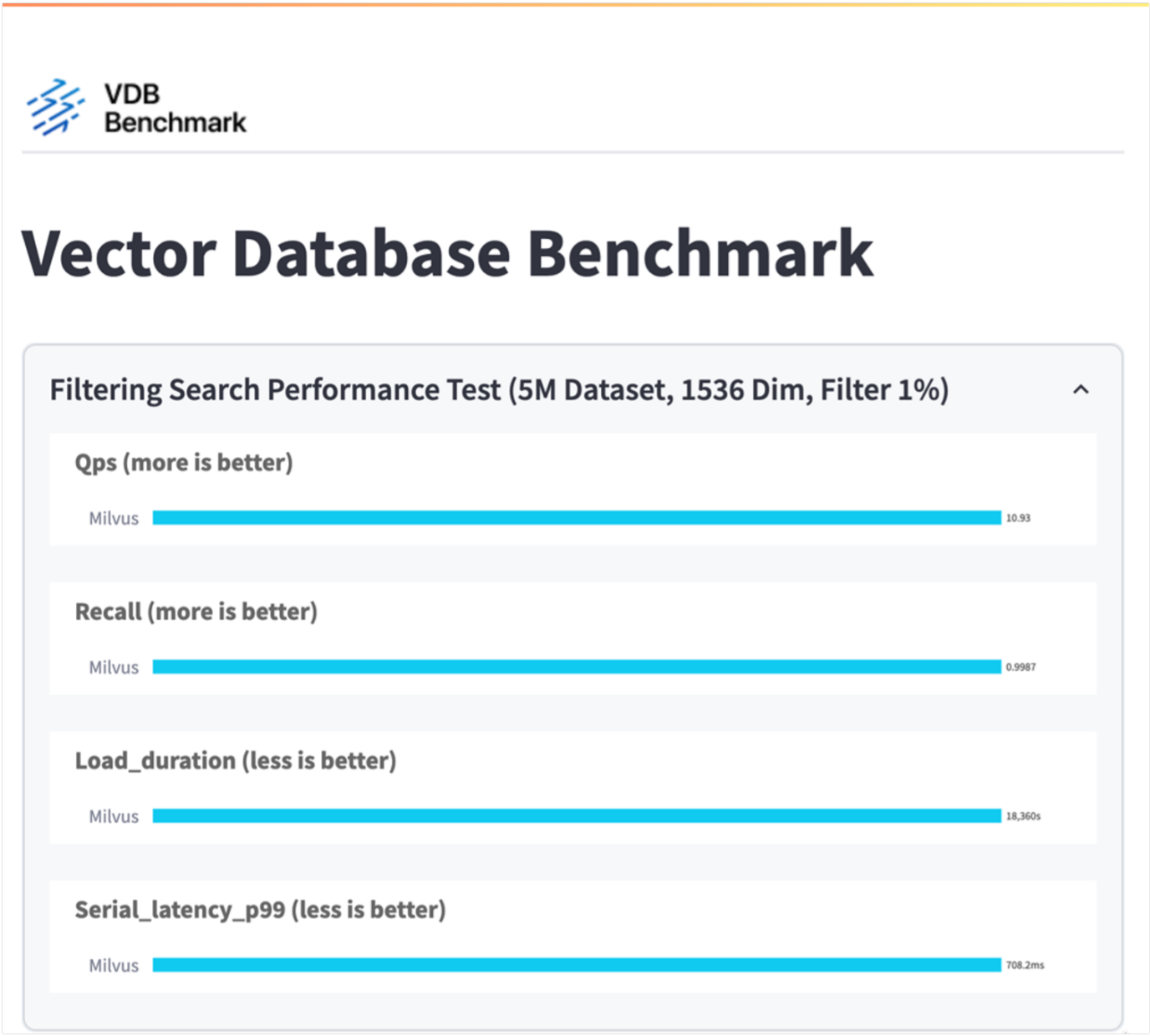
In questa sezione condividiamo le nostre osservazioni e i risultati ottenuti testando il database autonomo Milvus. . Abbiamo selezionato DiskANN come tipo di indice per questi test. . L'acquisizione, l'ottimizzazione e la creazione di indici per un set di dati di circa 100 GB hanno richiesto circa 5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

Dal punto di vista dell'archiviazione, il database ha eseguito circa 1.000 operazioni al secondo durante le fasi di acquisizione, ottimizzazione post-inserimento e creazione dell'indice. Nella fase di query, sono state richieste 32.000 operazioni al secondo.

Nella sezione seguente vengono presentate le metriche delle prestazioni di archiviazione.

Fase di carico di lavoro	Metrico	Valore
Inserimento dei dati e ottimizzazione post-inserimento	IOPS	< 1.000
	Latenza	< 400 usecs
	Carico di lavoro	Mix di lettura/scrittura, per lo più scrive
	dimensione IO	64 KB
Domanda	IOPS	Picco a 32.000
	Latenza	< 400 usecs
	Carico di lavoro	Lettura cache al 100%
	dimensione IO	Principalmente 8 KB

Di seguito è riportato il risultato di vectorDB-bench.

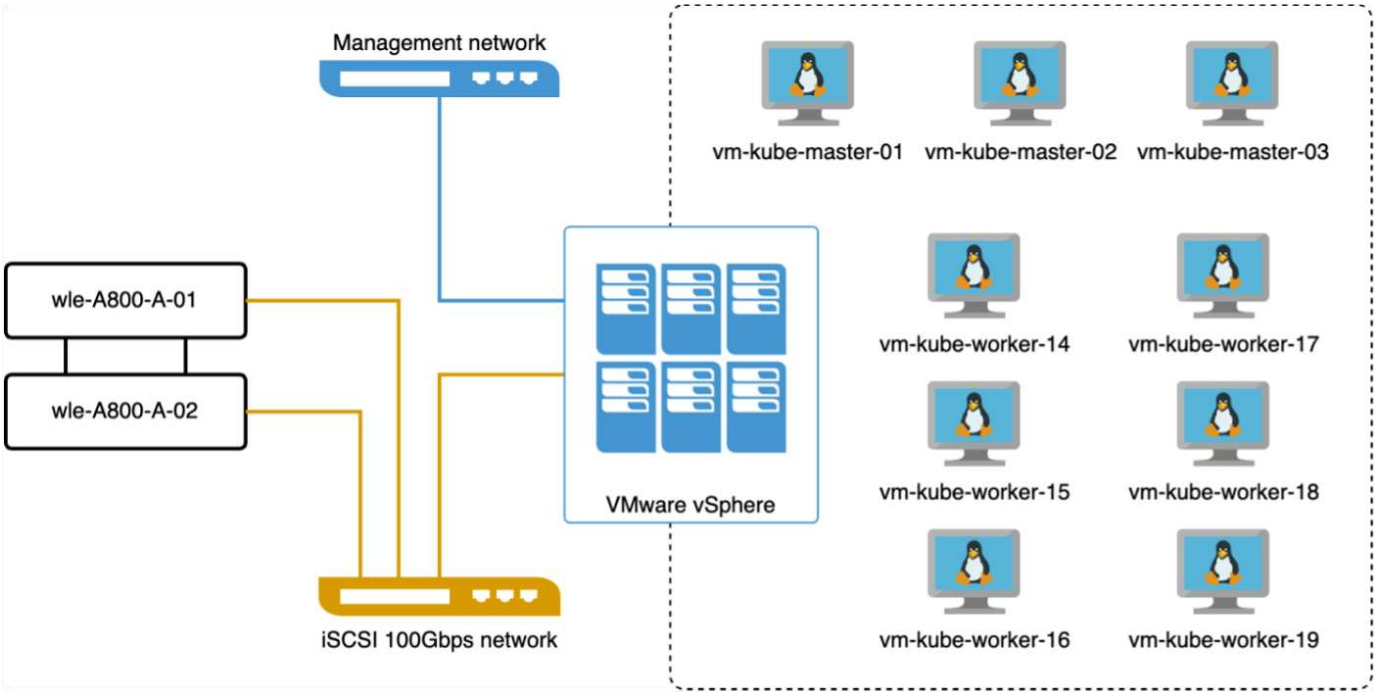


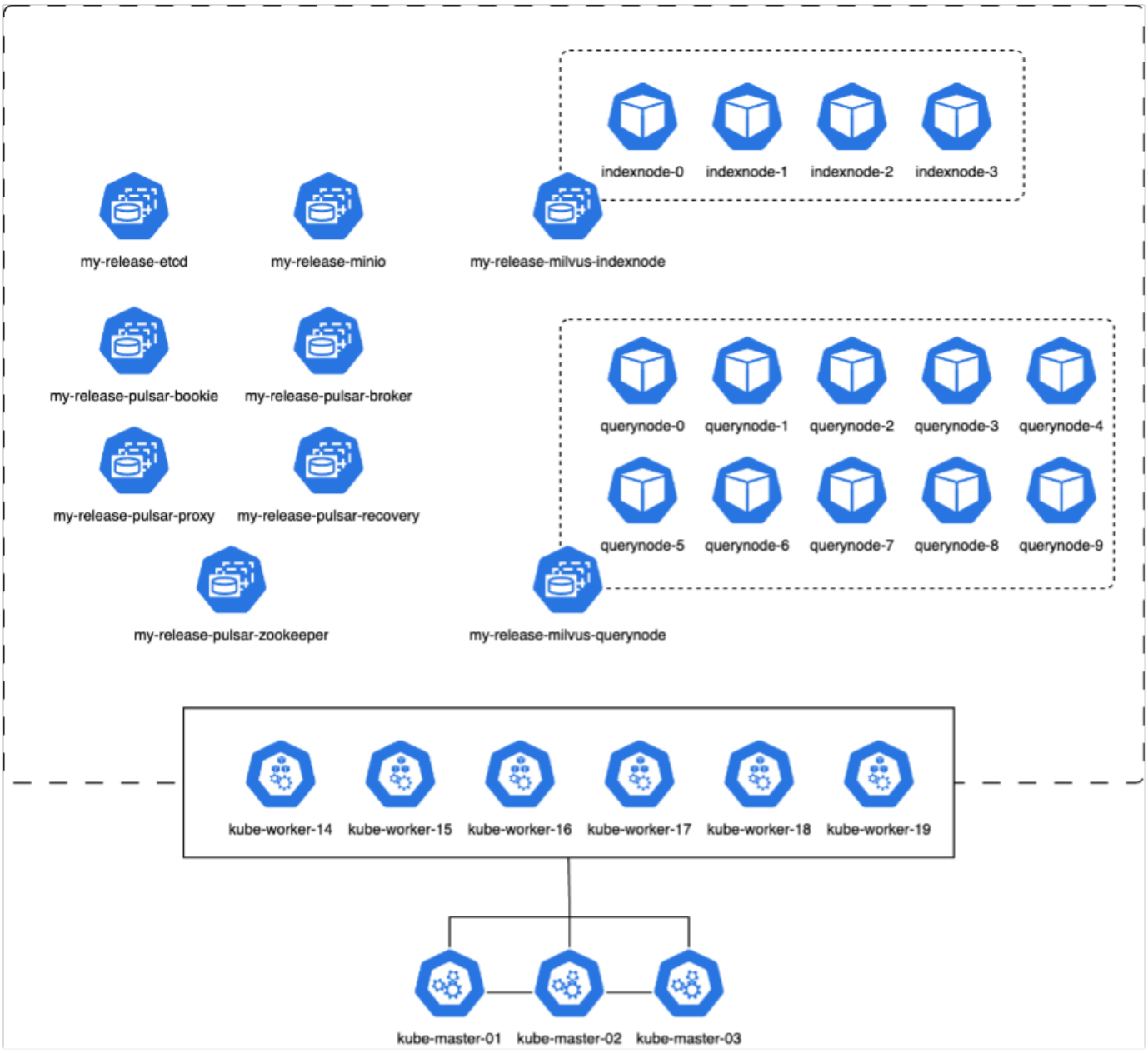
Dalla convalida delle prestazioni dell'istanza Milvus autonoma, è evidente che la configurazione attuale non è sufficiente a supportare un set di dati di 5 milioni di vettori con una dimensionalità di 1536. Abbiamo stabilito che lo storage dispone di risorse adeguate e non costituisce un collo di bottiglia nel sistema.

VectorDB-Bench con cluster Milvus

In questa sezione, discuteremo l'implementazione di un cluster Milvus all'interno di un ambiente Kubernetes. Questa configurazione di Kubernetes è stata realizzata su una distribuzione VMware vSphere, che ospitava i nodi master e worker di Kubernetes.

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


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



In questa sezione presentiamo le nostre osservazioni e i risultati ottenuti testando il database Milvus. * Il tipo di indice utilizzato era DiskANN. * La tabella seguente fornisce un confronto tra le distribuzioni standalone e cluster quando si lavora con 5 milioni di vettori con una dimensionalità di 1536. Abbiamo osservato che il tempo impiegato per l'acquisizione dei dati e l'ottimizzazione post-inserimento era inferiore nella distribuzione del cluster. La latenza del 99° percentile per le query è stata ridotta di sei volte nella distribuzione del cluster rispetto alla configurazione autonoma. * Sebbene la frequenza delle query al secondo (QPS) fosse più elevata nella distribuzione del cluster, non era al livello desiderato.

Metric	Milvus Standalone	Milvus Cluster	Difference
QPS @ Recall	10.93 @ 0.9987	18.42 @ 0.9952	+40%
p99 Latency (less is better)	708.2 ms	117.6 ms	-83%
Load Duration time (less is better)	18,360 secs	12,730 secs	-30%

Le immagini sottostanti forniscono una panoramica di varie metriche di archiviazione, tra cui la latenza del cluster di archiviazione e gli IOPS totali (operazioni di input/output al secondo).

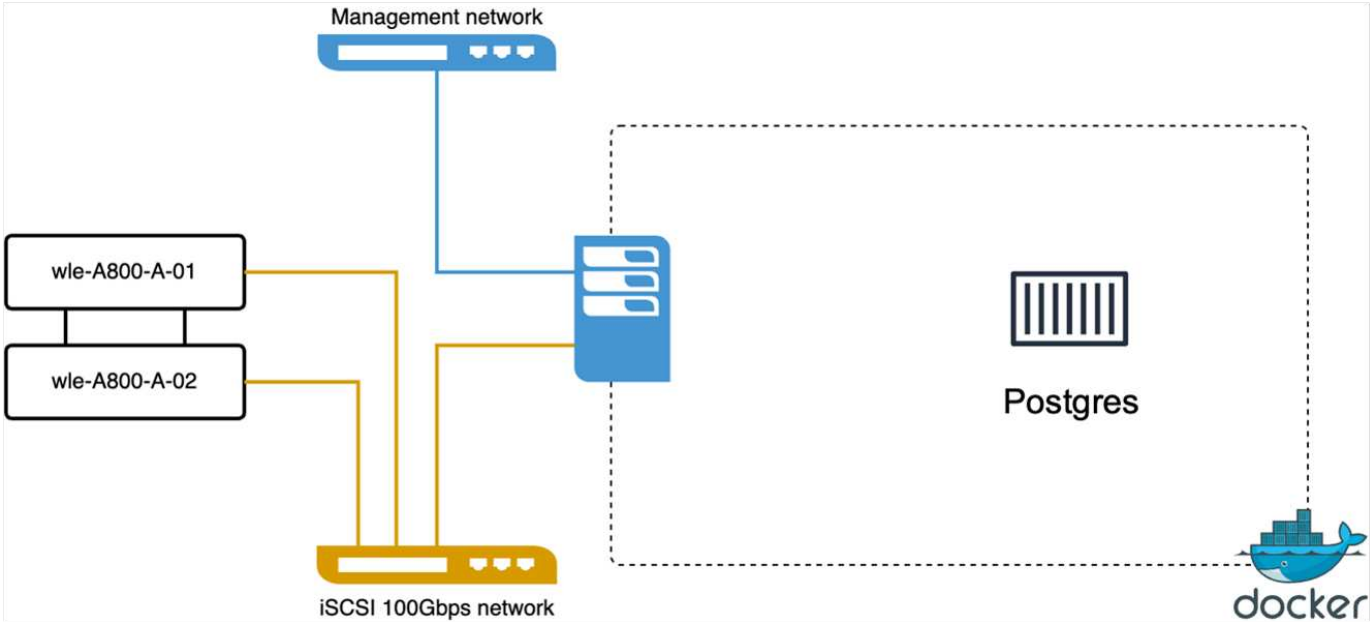
Nella sezione seguente vengono presentate le principali metriche relative alle prestazioni di archiviazione.

Fase di carico di lavoro	Metrico	Valore
Inserimento dei dati e ottimizzazione post-inserimento	IOPS	< 1.000
	Latenza	< 400 usecs
	Carico di lavoro	Mix di lettura/scrittura, per lo più scrive
	dimensione IO	64 KB
Domanda	IOPS	Picco a 147.000
	Latenza	< 400 usecs
	Carico di lavoro	Lettura cache al 100%
	dimensione IO	Principalmente 8 KB

Sulla base della convalida delle prestazioni sia del Milvus autonomo che del cluster Milvus, presentiamo i dettagli del profilo I/O di archiviazione. * Abbiamo osservato che il profilo I/O rimane coerente sia nelle distribuzioni autonome che in quelle cluster. * La differenza osservata nei picchi di IOPS può essere attribuita al numero maggiore di client nella distribuzione del cluster.

vectorDB-Bench con Postgres (pgvecto.rs)

Abbiamo eseguito le seguenti azioni su PostgreSQL (pgvecto.rs) utilizzando VectorDB-Bench: i dettagli riguardanti la connettività di rete e del server di PostgreSQL (in particolare, pgvecto.rs) sono i seguenti:

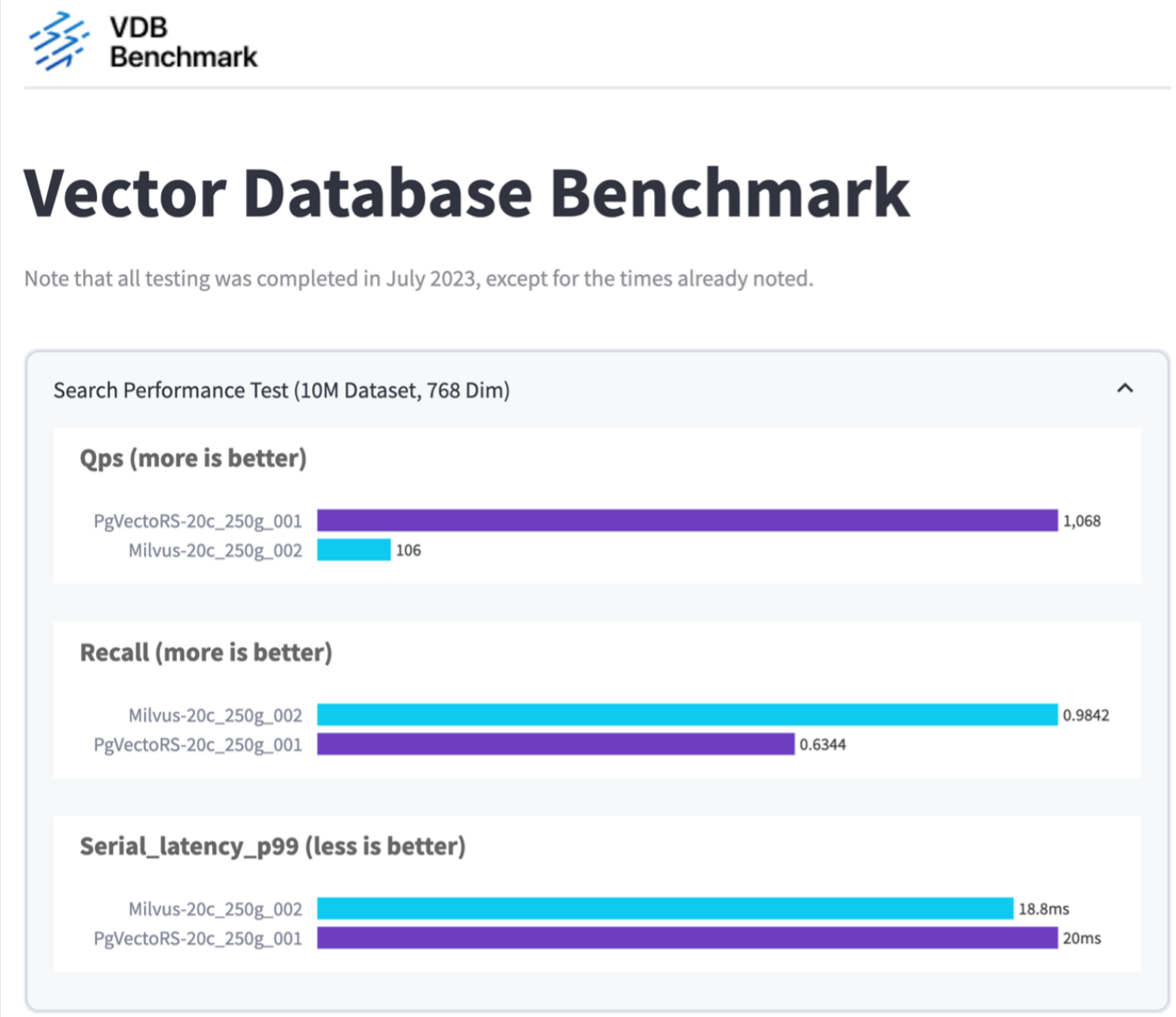


In questa sezione condividiamo le nostre osservazioni e i risultati ottenuti testando il database PostgreSQL, in particolare utilizzando pgvecto.rs. * Abbiamo selezionato HNSW come tipo di indice per questi test perché al momento del test, DiskANN non era disponibile per pgvecto.rs. * Durante la fase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

Le immagini sottostanti forniscono una panoramica di varie metriche di archiviazione, tra cui la latenza totale del cluster di archiviazione IOPS (operazioni di input/output al secondo).


```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-metrics.png["Figura che mostra il dialogo di
input/output o che rappresenta il contenuto scritto"]
```

Confronto delle prestazioni tra Milvus e Postgres su Vector DB Bench



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

- Tipo di indice: HNSW
- Dataset: Cohere con 10 milioni di vettori a 768 dimensioni

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

Se l'elevata precisione nelle tue query è una priorità, Milvus supera pgvecto.rs in quanto recupera una percentuale maggiore di elementi pertinenti per query. Tuttavia, se il numero di query al secondo è un fattore più cruciale, pgvecto.rs supera Milvus. È importante notare, tuttavia, che la qualità dei dati recuperati tramite pgvecto.rs è inferiore, con circa il 37% dei risultati di ricerca costituiti da elementi irrilevanti.

Osservazione basata sulle nostre convalide delle prestazioni:

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

In Milvus, il profilo I/O assomiglia molto a un carico di lavoro OLTP, come quello visto con Oracle SLOB. Il

benchmark è composto da tre fasi: acquisizione dei dati, post-ottimizzazione e query. Le fasi iniziali sono caratterizzate principalmente da operazioni di scrittura da 64 KB, mentre la fase di query prevede prevalentemente letture da 8 KB. Ci aspettiamo che ONTAP gestisca in modo efficiente il carico I/O Milvus.

Il profilo I/O di PostgreSQL non presenta un carico di lavoro di archiviazione impegnativo. Considerata l'implementazione in memoria attualmente in corso, non abbiamo osservato alcun I/O su disco durante la fase di query.

DiskANN emerge come una tecnologia cruciale per la differenziazione dello storage. Consente di ridimensionare in modo efficiente la ricerca nel database vettoriale oltre i limiti della memoria di sistema. Tuttavia, è improbabile che si possa stabilire una differenziazione delle prestazioni di archiviazione con indici DB vettoriali in memoria come HNSW.

Vale anche la pena notare che l'archiviazione non gioca un ruolo critico durante la fase di query quando il tipo di indice è HSNW, che è la fase operativa più importante per i database vettoriali che supportano le applicazioni RAG. Ciò implica che le prestazioni di archiviazione non hanno un impatto significativo sulle prestazioni complessive di queste applicazioni.

Database vettoriale con Instaclustr utilizzando PostgreSQL: pgvector

Questa sezione illustra le specifiche di come il prodotto instaclustr si integra con postgreSQL sulla funzionalità pgvector nella soluzione di database vettoriale per NetApp.

Database vettoriale con Instaclustr utilizzando PostgreSQL: pgvector

In questa sezione approfondiamo i dettagli di come il prodotto instaclustr si integra con postgreSQL sulla funzionalità pgvector. Abbiamo un esempio di "Come migliorare l'accuratezza e le prestazioni del tuo LLM con PGVector e PostgreSQL: introduzione agli incorporamenti e al ruolo di PGVector". Si prega di controllare il"blog" per ottenere maggiori informazioni.

Casi d'uso del database vettoriale

Questa sezione fornisce una panoramica dei casi d'uso per la soluzione di database vettoriale NetApp .

Casi d'uso del database vettoriale

In questa sezione, discuteremo di due casi d'uso quali Retrieval Augmented Generation con modelli linguistici di grandi dimensioni e NetApp IT chatbot.

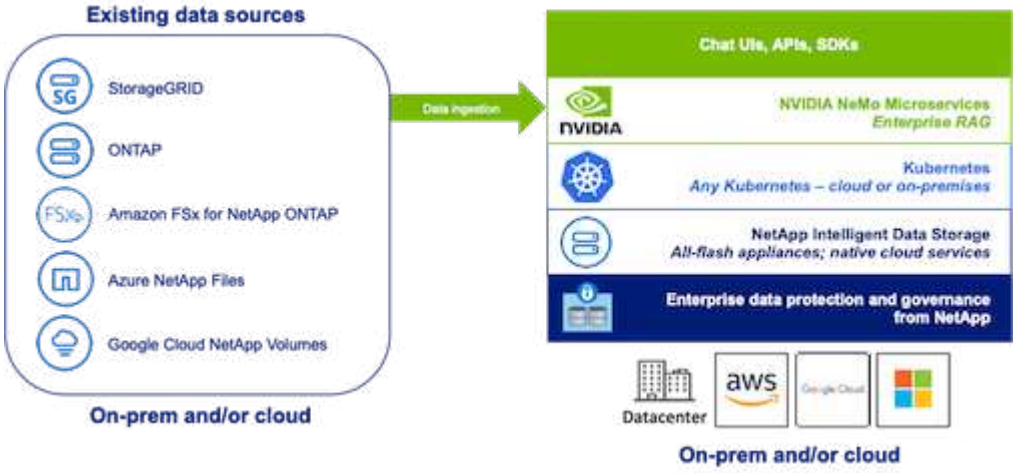
Generazione aumentata del recupero (RAG) con modelli linguistici di grandi dimensioni (LLM)

```
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 è uno strumento utile per implementare RAG in azienda. Questo operatore può essere utilizzato per distribuire una pipeline RAG completa. La pipeline RAG può essere personalizzata per utilizzare Milvus o pgvecto come database vettoriale per l'archiviazione degli incorporamenti della knowledge base. Per i dettagli, fare riferimento alla documentazione.

```
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.
```

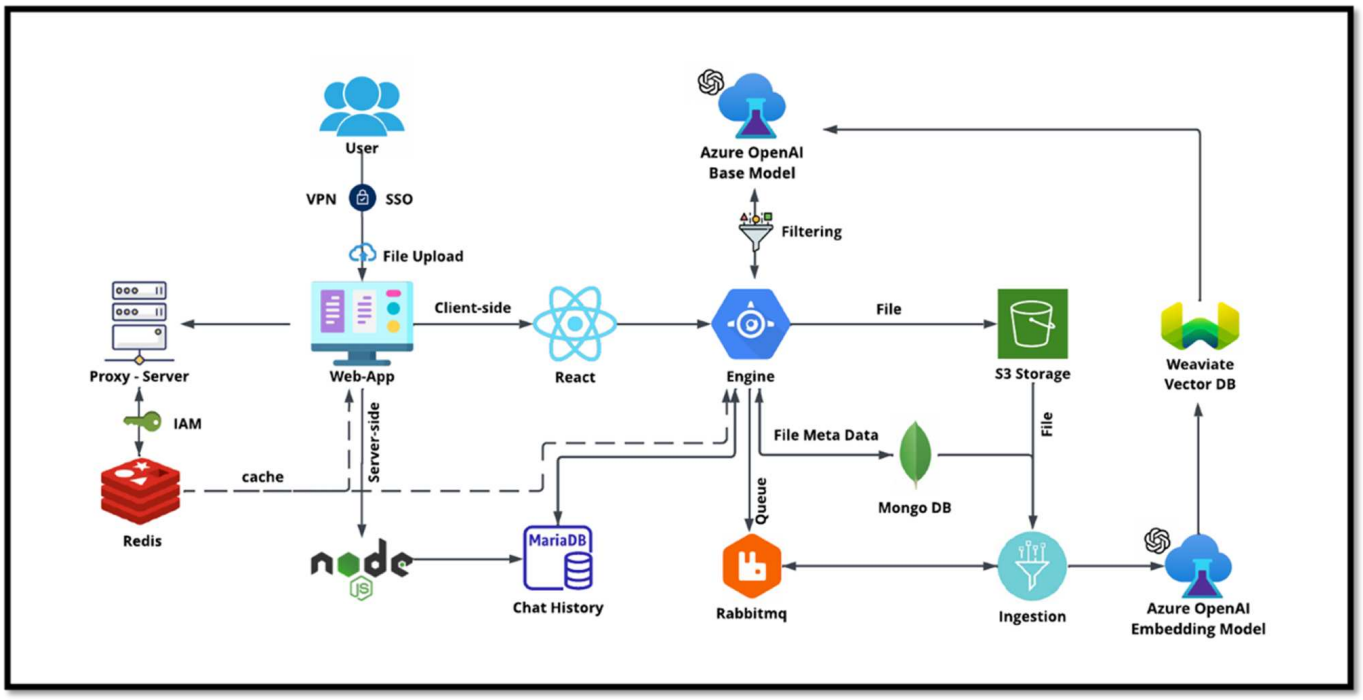
Figura 1) Enterprise RAG basato su NVIDIA NeMo Microservices e NetApp



Caso d'uso del chatbot IT NetApp

Il chatbot di NetApp rappresenta un ulteriore caso d'uso in tempo reale per il database vettoriale. In questo caso, NetApp Private OpenAI Sandbox fornisce una piattaforma efficace, sicura ed efficiente per la gestione delle query degli utenti interni di NetApp. Integrando rigorosi protocolli di sicurezza, efficienti sistemi di gestione dei dati e sofisticate capacità di elaborazione dell'intelligenza artificiale, garantisce risposte precise e di alta qualità agli utenti in base ai loro ruoli e responsabilità nell'organizzazione tramite autenticazione SSO. Questa architettura mette in luce il potenziale dell'unione di tecnologie avanzate per creare sistemi intelligenti e

incentrati sull'utente.



Il caso d'uso può essere suddiviso in quattro sezioni principali.

Autenticazione e verifica dell'utente:

- Le query degli utenti vengono prima sottoposte al processo NetApp Single Sign-On (SSO) per confermare l'identità dell'utente.
- Dopo l'autenticazione avvenuta con successo, il sistema controlla la connessione VPN per garantire una trasmissione sicura dei dati.

Trasmissione ed elaborazione dei dati:

- Una volta convalidata la VPN, i dati vengono inviati a MariaDB tramite le applicazioni web NetAIChat o NetAICreate. MariaDB è un sistema di database veloce ed efficiente utilizzato per gestire e archiviare i dati degli utenti.
- MariaDB invia quindi le informazioni all'istanza NetApp Azure, che collega i dati dell'utente all'unità di elaborazione AI.

Interazione con OpenAI e filtraggio dei contenuti:

- L'istanza di Azure invia le domande dell'utente a un sistema di filtraggio dei contenuti. Questo sistema pulisce la query e la prepara per l'elaborazione.
- L'input ripulito viene quindi inviato al modello base di Azure OpenAI, che genera una risposta in base all'input.

Generazione e moderazione delle risposte:

- La risposta del modello base viene prima verificata per garantire che sia accurata e soddisfi gli standard di contenuto.
- Dopo aver superato il controllo, la risposta viene inviata all'utente. Questo processo garantisce che l'utente

riceva una risposta chiara, accurata e appropriata alla sua domanda.

Conclusione

Questa sezione conclude la soluzione del database vettoriale per NetApp.

Conclusione

In conclusione, questo documento fornisce una panoramica completa sull'implementazione e la gestione di database vettoriali, come Milvus e pgvector, sulle soluzioni di storage NetApp . Abbiamo discusso le linee guida dell'infrastruttura per sfruttare l'archiviazione di oggetti NetApp ONTAP e StorageGRID e convalidato il database Milvus in AWS FSx ONTAP tramite l'archiviazione di file e oggetti.

Abbiamo esplorato la dualità file-oggetto di NetApp, dimostrandone l'utilità non solo per i dati nei database vettoriali, ma anche per altre applicazioni. Abbiamo anche evidenziato come SnapCenter, il prodotto di gestione aziendale di NetApp, offra funzionalità di backup, ripristino e clonazione per i dati dei database vettoriali, garantendone l'integrità e la disponibilità.

Il documento approfondisce anche il modo in cui la soluzione Hybrid Cloud di NetApp offre replicazione e protezione dei dati in ambienti on-premise e cloud, garantendo un'esperienza di gestione dei dati fluida e sicura. Abbiamo fornito approfondimenti sulla convalida delle prestazioni di database vettoriali come Milvus e pgvecto su NetApp ONTAP, offrendo informazioni preziose sulla loro efficienza e scalabilità.

Infine, abbiamo discusso due casi d'uso dell'intelligenza artificiale generativa: RAG con LLM e ChatAI interna di NetApp. Questi esempi pratici sottolineano le applicazioni e i vantaggi concreti dei concetti e delle pratiche delineati nel presente documento. Nel complesso, questo documento costituisce una guida completa per chiunque voglia sfruttare le potenti soluzioni di storage di NetApp per la gestione di database vettoriali.

Ringraziamenti

L'autore desidera ringraziare sentitamente i collaboratori indicati di seguito, coloro che hanno fornito feedback e commenti per rendere questo documento utile ai clienti NetApp e ai settori NetApp .

1. Sathish Thyagarajan, Ingegnere tecnico di marketing, ONTAP AI e analisi, NetApp
2. Mike Oglesby, ingegnere tecnico di marketing, NetApp
3. AJ Mahajan, Direttore Senior, NetApp
4. Joe Scott, Responsabile, Ingegneria delle prestazioni del carico di lavoro, NetApp
5. Puneet Dhawan, Direttore senior, Gestione prodotti Fsx, NetApp
6. Yuval Kalderon, Senior Product Manager, FSx Product Team, NetApp

Dove trovare ulteriori informazioni

Per saperne di più sulle informazioni descritte nel presente documento, consultare i seguenti documenti e/o siti web:

- Documentazione Milvus - https://milvus.io/docs/overview.md
- Documentazione autonoma di Milvus - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- Documentazione del prodotto NetApphttps://www.netapp.com/support-and-training/documentation/[]
- instaclustr -"documentazione di instalclustr"

Cronologia delle versioni

Versione	Data	Cronologia delle versioni del documento
Versione 1.0	Aprile 2024	Versione iniziale

Appendice A: Values.yaml

Questa sezione fornisce un codice YAML di esempio per i valori utilizzati nella soluzione del database vettoriale NetApp .

Appendice 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
```

```yaml
## 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:~#
```

Appendice B: prepare_data_netapp_new.py

Questa sezione fornisce un esempio di script Python utilizzato per preparare i dati per il database vettoriale.

Appendice 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:~#
```

Appendice C: verify_data_netapp.py

Questa sezione contiene uno script Python di esempio che può essere utilizzato per convalidare il database vettoriale nella soluzione di database vettoriale NetApp .

Appendice 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:~#
```

Appendice D: docker-compose.yml

Questa sezione include un codice YAML di esempio per la soluzione di database vettoriale per NetApp.

Appendice 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
```

Informazioni sul copyright

Informazioni sul marchio commerciale