



Soluzione database vettoriale con NetApp

NetApp Solutions

NetApp
May 03, 2024

Sommario

| | |
|--|----|
| Soluzione database vettoriale con NetApp | 1 |
| Introduzione | 2 |
| Panoramica della soluzione | 2 |
| Database vettoriale | 3 |
| Requisiti tecnologici | 6 |
| Procedura di implementazione | 7 |
| Panoramica della soluzione | 9 |
| Database vettoriale con Instaclustr utilizzando PostgreSQL: Pgvector | 44 |
| Casi di utilizzo del database vettoriale | 44 |
| Conclusione | 47 |
| Appendice A: Values.yaml | 48 |
| Appendice B: prepare_data_netapp_new.py | 69 |
| Appendice C: verify_data_netapp.py | 72 |
| Appendice D: docker-compose.yml | 75 |

Soluzione database vettoriale con NetApp

Karthikeyan Nagalingam e Rodrigo Nascimento, NetApp

Questo documento offre un'esplorazione approfondita dell'implementazione e della gestione di database vettoriali come Milvus e pgvector un'estensione PostgreSQL open-source, utilizzando le soluzioni storage di NetApp. Illustra in modo dettagliato le linee guida dell'infrastruttura per l'utilizzo dello storage a oggetti NetApp ONTAP e StorageGRID e convalida l'applicazione del database Milvus in AWS FSX per NetApp ONTAP. Il documento delinea il dualismo degli oggetti file di NetApp e la sua utility per database vettoriali e applicazioni che supportano le embedding vettoriali. Enfatizza le funzionalità di SnapCenter, il prodotto per la gestione aziendale di NetApp, nell'offerta di funzionalità di backup e ripristino per database vettoriali, garantendo l'integrità e la disponibilità dei dati. Il documento analizza ulteriormente la soluzione di cloud ibrido di NetApp, discutendo il suo ruolo nella replica e protezione dei dati negli ambienti on-premise e cloud. Include approfondimenti sulla convalida delle performance dei database vettoriali su NetApp ONTAP e si conclude con due casi d'utilizzo pratici su ai generativa: RAG con LLM e Chatai interno di NetApp. Il presente documento costituisce una guida completa all'utilizzo delle soluzioni storage di NetApp per la gestione di database vettoriali.

L'architettura di riferimento si concentra su quanto segue:

1. ["Introduzione"](#)
2. ["Panoramica della soluzione"](#)
3. ["Database vettoriale"](#)
4. ["Requisiti tecnologici"](#)
5. ["Procedura di implementazione"](#)
6. ["Panoramica di verifica della soluzione"](#)
 - ["Setup del cluster Milvus con Kubernetes on-premise"](#)
 - ["Milvus con Amazon FSxN per NetApp ONTAP – dualismo di file e oggetti"](#)
 - ["Protezione del database vettoriale tramite NetApp SnapCenter."](#)
 - ["Disaster recovery con NetApp SnapMirror"](#)
 - ["Convalida delle performance"](#)
7. ["Database vettoriale con Instaclustr utilizzando PostgreSQL: Pgvector"](#)
8. ["Casi di utilizzo del database vettoriale"](#)
9. ["Conclusione"](#)
10. ["Appendice A: Values.yaml"](#)
11. ["Appendice B: prepare_data_netapp_new.py"](#)
12. ["Appendice C: verify_data_netapp.py"](#)
13. ["Appendice D: docker-compose.yml"](#)

Introduzione

Introduzione

I database vettoriali risolvono efficacemente le sfide progettate per gestire le complessità della ricerca semantica nei modelli di linguaggio grande (LLMs) e nell'intelligenza artificiale generativa (ai). 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 piuttosto che etichette o tag.

I limiti dei sistemi di gestione di database relazionali (RDBMS) sono ben documentati, in particolare i loro problemi con le rappresentazioni di dati ad alta dimensione e i dati non strutturati comuni nelle applicazioni ai. RDBMS richiede spesso un processo dispendioso in termini di tempo e soggetto a errori per appiattare 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 dimensione, facilitando così il progresso delle applicazioni ai.

Questo documento è una guida completa per i clienti che utilizzano o prevedono di utilizzare i database vettoriali, descrivendo in dettaglio le Best practice per l'utilizzo dei database vettoriali su piattaforme quali NetApp ONTAP, NetApp StorageGRID, Amazon FSx for NetApp ONTAP e SnapCenter. Il contenuto qui fornito copre una vasta gamma di argomenti:

- Linee guida per l'infrastruttura per i database vettoriali, come Milvus, fornite dallo storage NetApp attraverso lo storage a oggetti NetApp ONTAP e StorageGRID.
- Validazione del database Milvus in AWS FSX per NetApp ONTAP tramite archivio di file e oggetti.
- Analizza il dualismo degli oggetti file di NetApp, dimostrando la sua utilità per i dati in database vettoriali e in altre applicazioni.
- In che modo SnapCenter, il prodotto NetApp per la gestione della protezione dei dati, offre funzionalità di backup e ripristino per i dati dei database vettoriali.
- Come offre replica e protezione dei dati nel cloud ibrido di NetApp in ambienti on-premise e cloud.
- Fornisce informazioni dettagliate sulla convalida delle prestazioni di database vettoriali come Milvus e pgvector su NetApp ONTAP.
- Due casi d'utilizzo specifici: Retrieval Augmented Generation (RAG) con modelli di linguaggio DI grandi dimensioni (LLM) e ChatAI del team IT di NetApp, offrendo così esempi pratici dei concetti e delle pratiche delineati.

Panoramica della soluzione

Panoramica della soluzione

Questa soluzione illustra i vantaggi e le funzionalità distintivi offerti da NetApp per affrontare le sfide affrontate dai clienti di database vettoriali. Sfruttando NetApp ONTAP, StorageGRID, le soluzioni cloud di NetApp e SnapCenter, i clienti possono aggiungere valore significativo alle loro operazioni di business. Questi strumenti non solo affrontano i problemi esistenti ma migliorano anche l'efficienza e la produttività, contribuendo in tal modo alla crescita complessiva dell'azienda.

Perché NetApp?

- Le offerte di NetApp, come ONTAP e StorageGRID, permettono la separazione di storage e calcolo, consentendo un utilizzo ottimale delle risorse sulla base di requisiti specifici. Questa flessibilità consente ai

clienti di scalare lo storage in maniera indipendente utilizzando le soluzioni di storage NetApp.

- Sfruttando gli storage controller NetApp, i clienti possono distribuire in modo efficiente i dati al database vettoriale utilizzando i protocolli NFS e S3. Questi protocolli facilitano lo storage 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 a file e a oggetti.
- NetApp ONTAP offre supporto nativo per NAS e storage a oggetti nei cloud service provider leader 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 prestazioni elevate.
- Grazie alle solide funzioni 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 tranquillità ai clienti in merito alla sicurezza e all'integrità delle loro preziose informazioni.

Database vettoriale

Database vettoriale

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

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

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

Vantaggi del database vettoriale per ML e ai:

- Ricerca high-dimensionale: I database vettoriali sono eccellenti nella gestione e nel recupero di dati ad alta dimensione, spesso generati in applicazioni ai e ML.
- Scalabilità: Possono scalare in modo efficiente per gestire grandi volumi di dati, supportando la crescita e l'espansione dei progetti ai e ML.
- Flessibilità: Le banche dati vettoriali offrono un elevato grado di flessibilità, consentendo l'adattamento di diversi tipi di dati e strutture.
- Performance: Offrono gestione e recupero dei dati dalle performance elevate, elementi critici per la velocità e l'efficienza delle operazioni ai e ML.
- Indicizzazione personalizzabile: I database vettoriali offrono opzioni di indicizzazione personalizzabili, consentendo l'organizzazione e il recupero dei dati ottimizzati in base alle esigenze specifiche.

Database vettoriali e casi di utilizzo.

In questa sezione vengono forniti vari database vettoriali e i relativi dettagli sui casi di utilizzo.

Faiss e Scann

Si tratta di librerie che fungono da strumenti cruciali nel campo della ricerca vettoriale. Queste librerie forniscono funzionalità che sono fondamentali per la gestione e la ricerca dei dati vettoriali, rendendoli risorse preziose in questa area specializzata della gestione dei dati.

Elasticsearch

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

Pinecone

Si tratta di un solido database vettoriale con un unico set di funzioni. Supporta vettori sia densi che sparse nella sua funzionalità di indicizzazione, che ne migliora la flessibilità e l'adattabilità. Uno dei suoi punti di forza principali risiede nella sua capacità di combinare i metodi di ricerca tradizionali con la ricerca di vettori densi basata su ai, creando un approccio di ricerca ibrida che sfrutta il meglio di entrambi i mondi.

Principalmente basato su cloud, Pinecone è progettato per applicazioni di machine learning e si integra perfettamente con una varietà di piattaforme, tra cui GCP, AWS, Open ai, GPT-3, GPT-3,5, GPT-4, Catgut Plus, Elasticsearch, Haystack, e molto altro ancora. È importante notare che Pinecone è una piattaforma chiusa ed è disponibile come offerta Software as a Service (SaaS).

Grazie alle sue funzionalità avanzate, Pinecone è particolarmente adatto per il settore della cybersicurezza, dove le sue funzionalità di ricerca ad alta dimensione e ibrida possono essere sfruttate in modo efficace per rilevare e rispondere alle minacce.

Cromia

Si tratta di un database vettoriale che ha un Core-API con quattro funzioni principali, una delle quali include un archivio vettoriale di documenti in memoria. Utilizza inoltre la libreria Face Transformers per vettorizzare i documenti, migliorandone la funzionalità e la versatilità.

Chroma è progettato per operare sia nel cloud che on-premise, offrendo flessibilità sulla base delle esigenze dell'utente. In particolare, eccelle nelle applicazioni audio-correlate, rendendolo una scelta eccellente per i motori di ricerca audio-basati, i sistemi di raccomandazione di musica e altri casi di utilizzo audio-correlati.

Weaviate

Si tratta di un database vettoriale versatile che consente agli utenti di vettorizzare i propri contenuti utilizzando moduli integrati o moduli personalizzati, fornendo flessibilità in base a esigenze specifiche. Offre soluzioni completamente gestite e in hosting indipendente, adatte a una vasta gamma di preferenze di implementazione.

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 motori di ricerca e di raccomandazione. Weaviate viene inoltre utilizzato per la ricerca di immagini, il rilevamento di anomalie, l'armonizzazione automatizzata dei dati e l'analisi delle minacce alla sicurezza informatica, evidenziandone la versatilità in più domini.

Redis

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

dati.

Redis supporta varie strutture di dati per i vettori, inclusi elenchi, set e set ordinati. Fornisce anche operazioni vettoriali come il calcolo delle distanze tra vettori o la ricerca di intersezioni e unioni. Queste funzioni 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 è eccellente nella gestione dei carichi di lavoro a throughput elevato e offre la replica dei dati. Si integra bene anche con altri tipi di dati, inclusi i tradizionali database relazionali (RDBMS).

Redis include una funzione Publish/Subscribe (Pub/Sub) per gli aggiornamenti in tempo reale, utile per la gestione dei vettori in tempo reale. Inoltre, Redis è leggero e semplice da utilizzare, il che lo rende una soluzione di facile utilizzo per la gestione dei dati vettoriali.

Milvus

Si tratta di un database vettoriale versatile che offre un'API come un archivio documenti, proprio come MongoDB. Si distingue per il supporto di un'ampia varietà di tipi di dati, che la rende una scelta popolare nel campo della data science 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 runtime il tipo di vettore da utilizzare per la ricerca. Inoltre, utilizza Knowwhere, una libreria che si trova in cima ad altre librerie come Faiss, per gestire la comunicazione tra le query e gli algoritmi di ricerca vettoriale.

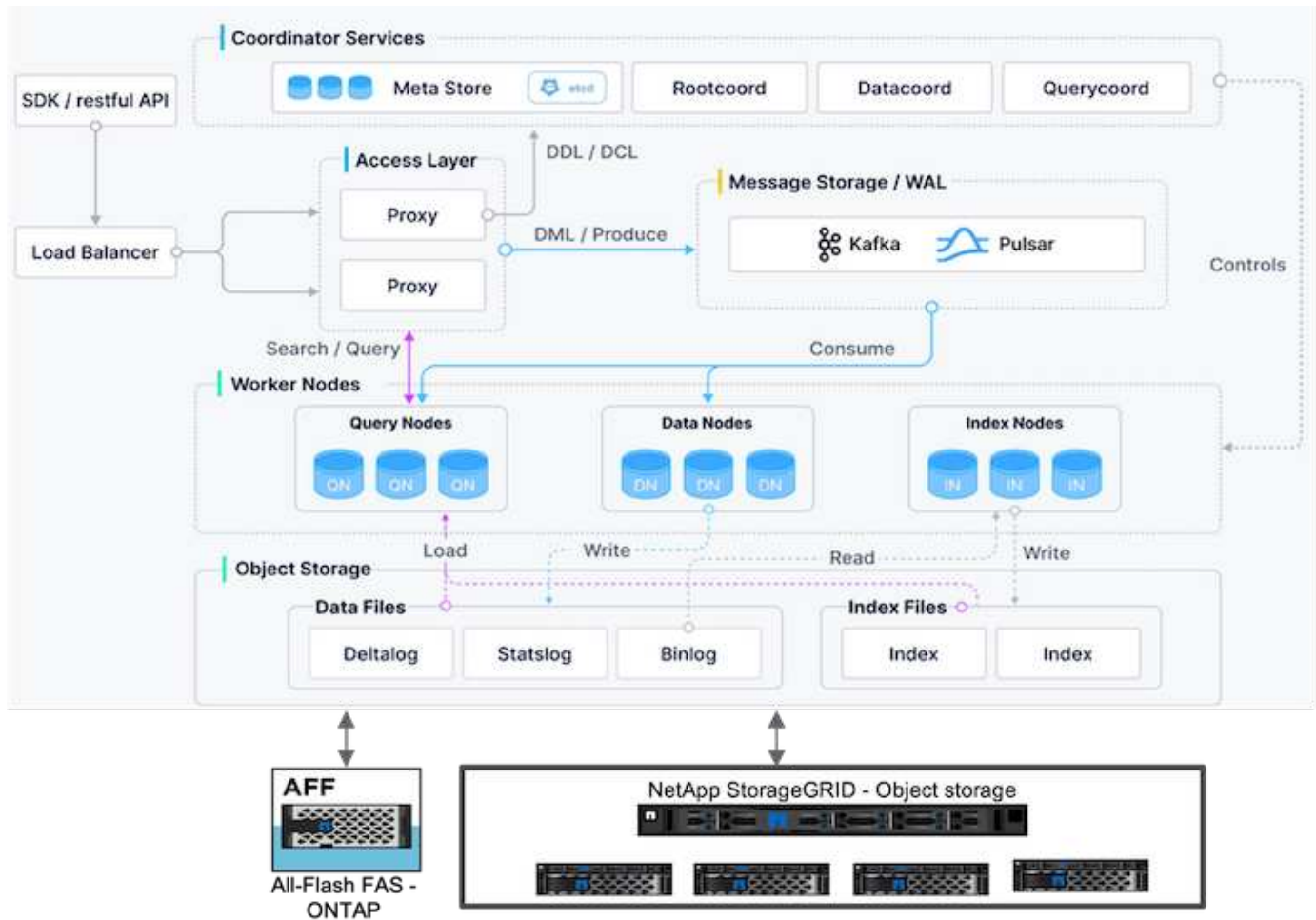
Milvus offre inoltre una perfetta integrazione con i flussi di lavoro di apprendimento automatico, grazie alla sua compatibilità con PyTorch e TensorFlow. Questo lo rende uno strumento eccellente per una vasta gamma di applicazioni, tra cui e-commerce, analisi di immagini e video, riconoscimento di oggetti, ricerca di similarità delle immagini e recupero di immagini basato sul contenuto. Nell'ambito dell'elaborazione del linguaggio naturale, Milvus viene utilizzato per il clustering dei 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(pgvector).

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 di applicazioni di ricerca della similarità e di intelligenza artificiale per migliorare le funzionalità del database vettoriale.
- **Gestione di grandi volumi:** Milvus ha la capacità di memorizzare, indicizzare e gestire oltre un miliardo di vettori di incorporazione generati dai modelli Deep Neural Networks (DNN) e Machine Learning (ML).
- **Intuitivo:** Facile da utilizzare e in meno di un minuto di tempo. Milvus offre anche SDK per diversi linguaggi di programmazione.
- **Velocità:** Offre velocità di recupero incredibilmente elevate, fino a 10 volte più veloci di alcune alternative.
- **Scalabilità e disponibilità:** Milvus è altamente scalabile e offre opzioni di scale-up e scale-out in base alle necessità.
- **Ricco di funzioni:** Supporta diversi tipi di dati, filtro degli attributi, supporto UDF (User-Defined Function), livelli di coerenza configurabili e tempi di viaggio, il che lo rende uno strumento versatile per varie applicazioni.

Panoramica dell'architettura di Milvus



In questa sezione vengono forniti componenti e servizi di livello superiore della leva utilizzati nell'architettura Milvus.

* Livello di accesso: È composto da un gruppo di proxy stateless e funge da livello anteriore del sistema e dell'endpoint per gli utenti.

* Servizio Coordinatore: Assegna le attività ai nodi di lavoro e agisce come cervello di un sistema. Ha tre tipi di coordinatore: La coord di root, la coord di dati e la coord di query.

* Nodi di lavoro : segue l'istruzione del servizio coordinatore ed esegue DML/DDC commands.it attivato dall'utente ha tre tipi di nodi di lavoro come nodo di query, nodo di dati e nodo di indice.

* Storage: È responsabile della persistenza dei dati. Comprende metadata storage, broker dei log e storage a oggetti. Lo storage NetApp come ONTAP e StorageGRID offre storage a oggetti e basato su file a Milvus, sia per i dati dei clienti che per i dati del database vettoriale.

Requisiti tecnologici

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 aiutarvi a configurare il vostro ambiente. Tuttavia, tenere presente che i componenti specifici possono variare in base alle esigenze dei singoli clienti.

Requisiti hardware

| Hardware | Dettagli |
|--|--|
| Coppia ha dello storage array NetApp AFF | <ul style="list-style-type: none">* A800* ONTAP 9.14.1* 48 SSD-NVM DA 3,49TB GB* Due volumi di gruppo flessibili: Metadati e dati.* Il volume NFS metadati ha 12 volumi persistenti con 250GB.* I dati sono un volume NAS S3 di ONTAP |
| 6 X FUJITSU PRIMERGY RX2540 M4 | <ul style="list-style-type: none">* 64 CPU* CPU Intel® Xeon® Gold 6142 a 2,60GHz GHZ* 256 MB di memoria fisica GM* Porta di rete 1 x 100GbE |
| Networking | 100 GbE |
| StorageGRID | <ul style="list-style-type: none">* 1 x SG100, 3xSGF6024* 3 x 24 x 7,68TB |

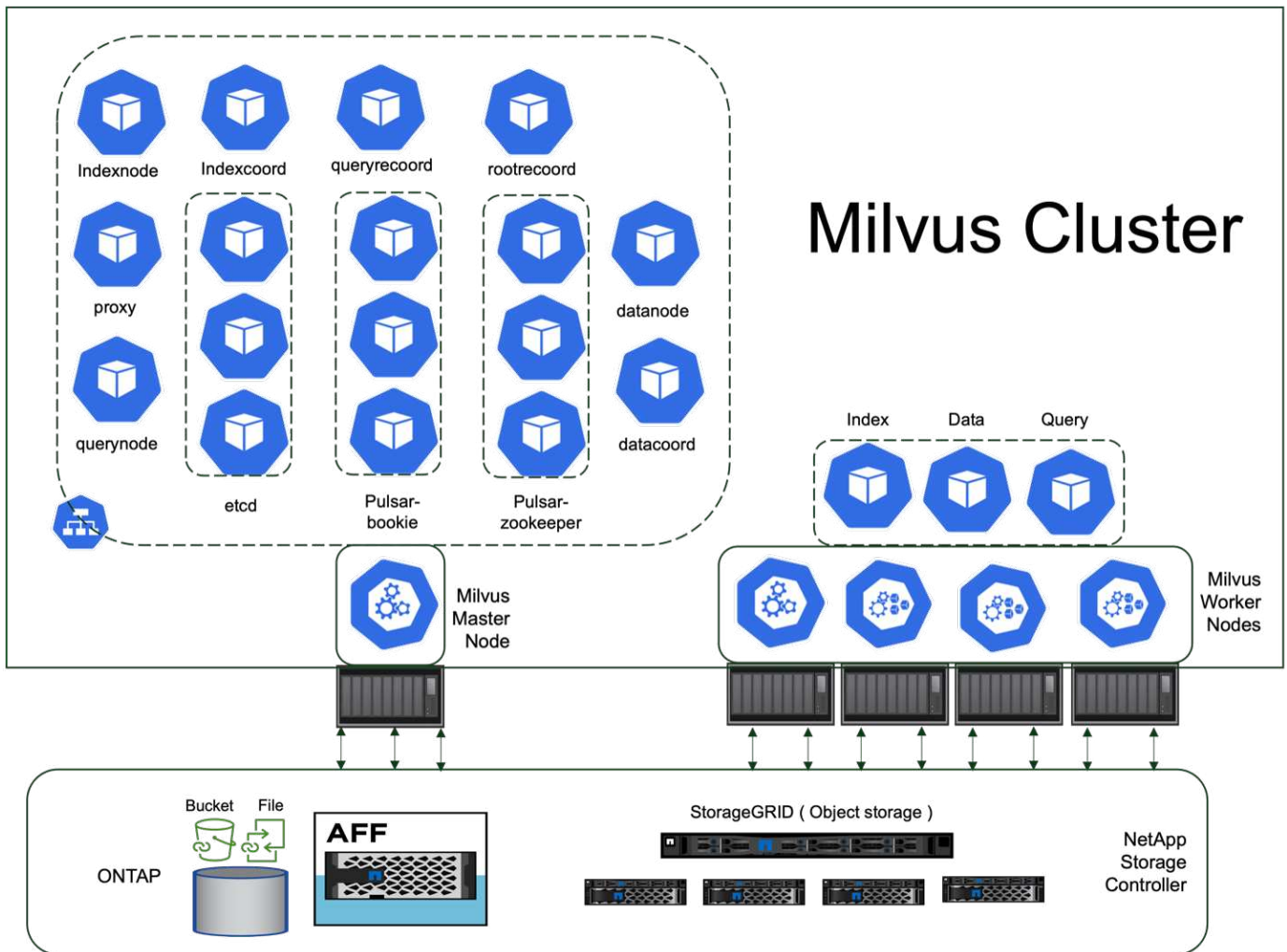
Requisiti software

| Software | Dettagli |
|----------------|---|
| Cluster Milvus | <ul style="list-style-type: none">* GRAFICO - milvus-4,1.11.* Versione APP – 2.3.4* Pacchetti dipendenti come bookkeeper, zookeeper, Pulsar, etcd, proxy, querynode, worker |
| Kubernetes | <ul style="list-style-type: none">* Cluster K8s a 5 nodi* 1 nodi master e 4 nodi Worker* Versione – 1.7.2 |
| Python | *3.10.12. |

Procedura di implementazione

Procedura di distribuzione

In questa sezione abbiamo utilizzato il database vettoriale milvus con Kubernetes per la configurazione di laboratorio, come indicato di seguito.



Lo storage NetApp fornisce lo storage per il cluster, al fine di mantenere i dati dei clienti e dei dati del cluster Milvus.

Setup dello storage NetApp: ONTAP

- Inizializzazione del sistema di archiviazione
- Creazione di una Storage Virtual Machine (SVM)
- Assegnazione di interfacce di rete logiche
- Configurazione e licensing di NFS, S3

Per NFS (Network file System), attenersi alla seguente procedura:

1. Crea un volume FlexGroup per NFSv4. Nella nostra configurazione per questa convalida, abbiamo utilizzato 48 SSD, 1 SSD dedicati al volume root del controller e 47 SSD distribuiti per NFSv4]]. Verifica che la policy di esportazione NFS per il volume FlexGroup disponga di permessi di lettura/scrittura per la rete di nodi Kubernetes (K8s). Se queste autorizzazioni non sono disponibili, concedere autorizzazioni di lettura/scrittura (rw) per la rete a K8s nodi.
2. Su tutti i K8s nodi, crea una cartella e monta il volume FlexGroup in questa cartella tramite un'interfaccia logica (LIF) su ogni K8s nodi.

Attendersi alla procedura riportata di seguito per NAS S3 (Network Attached Storage Simple Storage Service):

1. Crea un volume FlexGroup per NFS.
2. Impostare un server-archivio-oggetti con HTTP attivato e lo stato di amministrazione impostato su 'UP' utilizzando il comando "server-archivio-oggetti-server create". È possibile attivare HTTPS e impostare una porta del listener 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 comando seguente: "Set diag; vserver object-store-server user show -user <username>". Tuttavia, in futuro, questi tasti verranno forniti durante il processo di creazione dell'utente o possono essere recuperati utilizzando le chiamate API REST.
5. Stabilire un gruppo di server di archiviazione oggetti 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 fornendo il percorso al volume NFSv3. A tale scopo è anche possibile utilizzare un bucket S3.

Setup dello storage NetApp: StorageGRID

1. Installazione del software StorageGRID.
2. Creare un tenant e un bucket.
3. Creare un utente con l'autorizzazione richiesta.

Per ulteriori dettagli, consultare <https://docs.netapp.com/us-en/storagegrid-116/primer/index.html>

Panoramica della soluzione

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

1. **"Setup del cluster Milvus con Kubernetes on-premise"**
I clienti affrontano la sfida di scalare in maniera indipendente su storage e calcolo, un'efficace gestione dei dati e dell'infrastruttura. In questa sezione, descriviamo in dettaglio il processo di installazione di un cluster Milvus su Kubernetes, che utilizza uno storage controller NetApp sia per i dati del cluster sia per i dati del cliente.
2. **"Milvus con Amazon FSxN per NetApp ONTAP – dualismo di file e oggetti"**
In questa sezione, perché è necessario distribuire il database vettoriale nel cloud e i passaggi per distribuire il database vettoriale (milvus standalone) in Amazon FSxN per NetApp ONTAP all'interno dei contenitori docker.
3. **"Protezione del database vettoriale tramite NetApp SnapCenter."**
In questa sezione, analizzeremo come SnapCenter protegge i dati del database vettoriale e i dati Milvus che risiedono in ONTAP. Per questo esempio, abbiamo utilizzato un bucket NAS (milvusdbvol1) derivato da un volume NFS ONTAP (vol1) per i dati dei clienti e un volume NFS separato (vectordbpv) per i dati di configurazione del cluster Milvus.
4. **"Disaster recovery con NetApp SnapMirror"**
In questa sezione, verranno descritte l'importanza del ripristino di emergenza (DR) per database vettoriali e il modo in cui il prodotto di ripristino di emergenza NetApp snapmirror fornisce la soluzione di DR al database vettoriale.
5. **"Convalida delle performance"**
In questa sezione, il nostro obiettivo è analizzare la convalida delle prestazioni dei database vettoriali,

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

Setup cluster Milvus con Kubernetes on-premise

Setup del cluster Milvus con Kubernetes on-premise

Le sfide dei clienti di scalare in maniera indipendente su storage e calcolo, un'efficace gestione dell'infrastruttura e i dati

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

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



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

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

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

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

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

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

```
#####
# External S3
# - these configs are only used when `externalS3.enabled` is true
#####
externalS3:
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
  useSSL: false
  bucketName: "milvusdbvoll1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
```

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

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

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

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

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

```

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

```

12. Controllare lo stato dei pod.

```

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

```

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

13. Testare la scrittura e la lettura dei dati nello storage a oggetti Milvus e NetApp.

- Scrivere i dati utilizzando il programma Python "Prepare_data_netapp_new.py".


```

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

```

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

```

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

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

=== Start Creating index IVF_FLAT   ===

=== Start loading                   ===

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

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

```

```

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

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

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

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

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

```

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

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

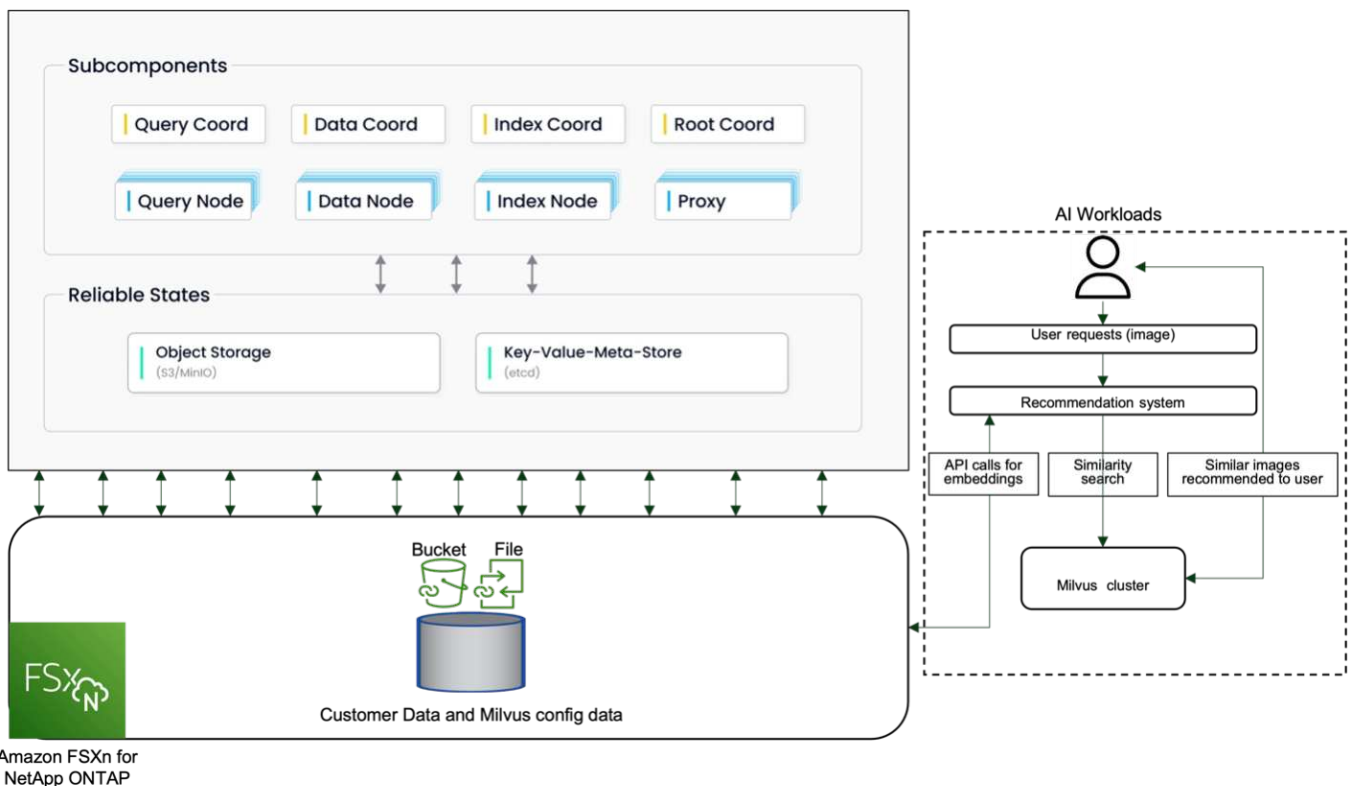
Milvus con Amazon FSxN per NetApp ONTAP - dualità di file e oggetti

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

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

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

Controllare l'architettura del milvus standalone con Amazon FSxN per NetApp ONTAP utilizzato in questa convalida.



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

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

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

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

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

```

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

```

9. Eseguire 'docker-compite up -d' dalla directory contenente il file docker-compite.yml.

10. Controllare lo stato del contenitore Milvus.

```

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

```

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

12. Convalidare le operazioni di scrittura e lettura dei dati da Amazon FSxN per NetApp ONTAP nel database

vettoriale.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3
prepare_data_netapp_new.py
=== start connecting to Milvus      ===
=== Milvus host: localhost          ===
Does collection hello_milvus_ntapnew_sc exist in Milvus: True
=== Drop collection - hello_milvus_ntapnew_sc ===
=== Drop collection - hello_milvus_ntapnew_sc2 ===
=== Create collection `hello_milvus_ntapnew_sc` ===
=== Start inserting entities        ===
Number of entities in hello_milvus_ntapnew_sc: 9000
root@ip-172-31-29-98:~/pymilvus/examples# find
/home/ubuntu/milvusvectordb/
...
<removed content to save page space >
...
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/b3def25f-c117-4fba-8256-96cb7557cd6c
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/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
/448789845791611912/448789845791611913/448789845791611939/0/448789845791
411924
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/0/448789845791
411924/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
/448789845791611912/448789845791611913/448789845791611939/1/448789845791
411925
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/1/448789845791
411925/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
/448789845791611912/448789845791611913/448789845791611939/100/4487898457
```

91411920

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

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

```
root@ip-172-31-29-98:~/pymilvus/examples# python3 verify_data_netapp.py  
=== start connecting to Milvus ===  
  
=== Milvus host: localhost ===  
  
Does collection hello_milvus_ntapnew_sc exist in Milvus: True  
{'auto_id': False, 'description': 'hello_milvus_ntapnew_sc', 'fields':  
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,  
'is_primary': True, 'auto_id': False}, {'name': 'random', 'description':  
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',  
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},  
{'name': 'embeddings', 'description': '', 'type': <DataType.  
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':  
False}  
Number of entities in Milvus: hello_milvus_ntapnew_sc : 9000  
  
=== Start Creating index IVF_FLAT ===  
  
=== Start loading ===  
  
=== Start searching based on vector similarity ===  
  
hit: id: 2248, distance: 0.0, entity: {'random': 0.2777646777746381},  
random field: 0.2777646777746381  
hit: id: 4837, distance: 0.07805602252483368, entity: {'random':  
0.6451650959930306}, random field: 0.6451650959930306  
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':  
0.6141351712303128}, random field: 0.6141351712303128  
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},  
random field: 0.7434908973629817  
hit: id: 830, distance: 0.05628090724349022, entity: {'random':  
0.8544487225667627}, random field: 0.8544487225667627  
hit: id: 8562, distance: 0.07971227169036865, entity: {'random':  
0.4464554280115878}, random field: 0.4464554280115878  
search latency = 0.1266s  
  
=== Start querying with `random > 0.5` ===
```

```

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

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

hit: id: 4837, distance: 0.07805602252483368, entity: {'random':
0.6451650959930306}, random field: 0.6451650959930306
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':
0.6141351712303128}, random field: 0.6141351712303128
hit: id: 515, distance: 0.09590047597885132, entity: {'random':
0.8013175797590888}, random field: 0.8013175797590888
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},
random field: 0.7434908973629817
hit: id: 830, distance: 0.05628090724349022, entity: {'random':
0.8544487225667627}, random field: 0.8544487225667627
hit: id: 1627, distance: 0.08096684515476227, entity: {'random':
0.9302397069516164}, random field: 0.9302397069516164
search latency = 0.2674s
Does collection hello_milvus_ntapnew_sc2 exist in Milvus: True
{'auto_id': True, 'description': 'hello_milvus_ntapnew_sc2', 'fields':
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,
'is_primary': True, 'auto_id': True}, {'name': 'random', 'description':
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},
{'name': 'embeddings', 'description': '', 'type': <DataType.
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':
False}

```

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

```

root@ip-172-31-29-98:~/pymilvus/examples# sudo python3
/home/ubuntu/milvusvectordb/s3_access.py -i 172.31.255.228 --bucket
milvusnasvol --access-key PY6UF318996I86NBYNDD --secret-key
hoPctr9aD88c1j0SkIYZ2uPa03v1bqKA0c5feK6F
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 dimostra in modo efficace come i clienti possono implementare e utilizzare un setup Milvus

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

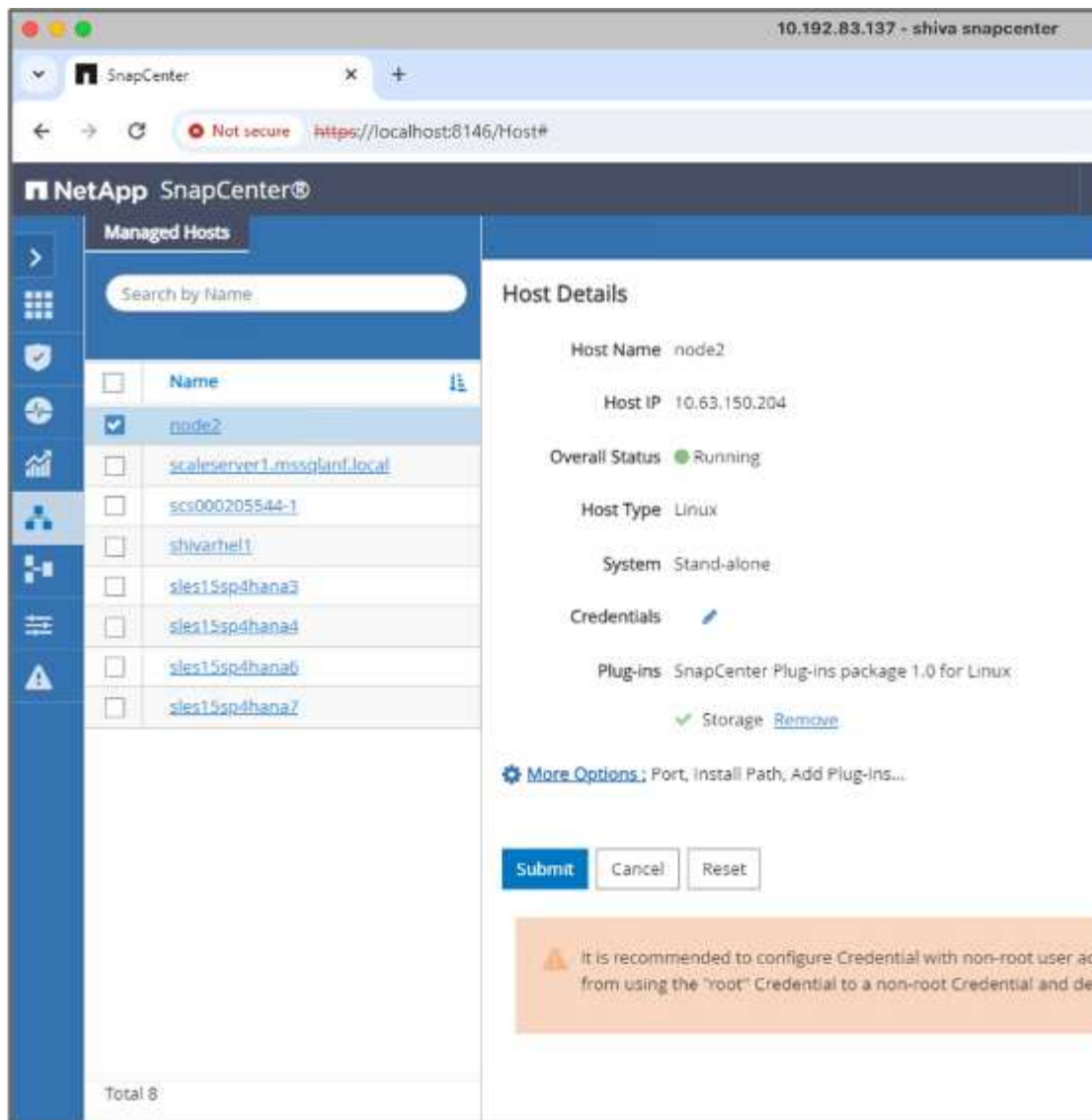
Protezione database vettoriale con SnapCenter

Protezione del database vettoriale tramite NetApp SnapCenter.

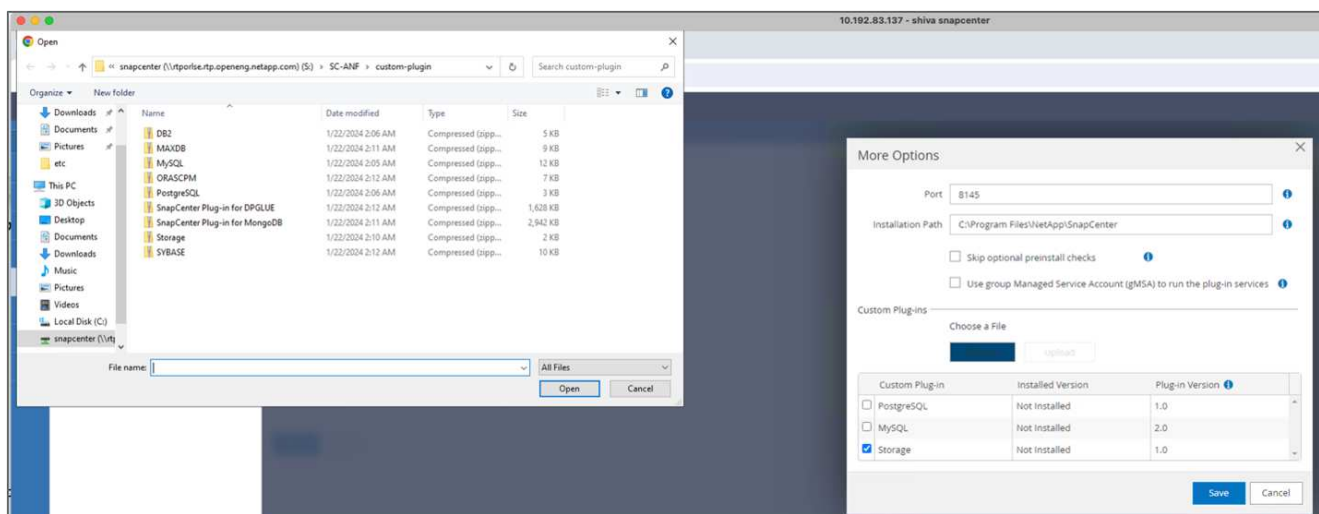
Ad esempio, nel settore della produzione cinematografica, i clienti spesso possiedono dati integrati critici come file video e audio. La perdita di questi dati, a causa di problemi come i guasti del disco rigido, può avere un impatto significativo sulle loro operazioni, potenzialmente compromettendo le iniziative multimilionarie. Abbiamo riscontrato casi in cui si sono persi contenuti di valore inestimabile, causando notevoli interruzioni e perdite finanziarie. Garantire la sicurezza e l'integrità di questi dati essenziali è pertanto di fondamentale importanza in questo settore.

In questa sezione, analizzeremo come SnapCenter protegge i dati del database vettoriale e i dati Milvus che risiedono in ONTAP. Per questo esempio, abbiamo utilizzato un bucket NAS (milvusdbvol1) derivato da un volume NFS ONTAP (vol1) per i dati dei clienti e un volume NFS separato (vectordbvp) per i dati di configurazione del cluster Milvus. controllare ["qui"](#) per flusso di lavoro del backup SnapCenter

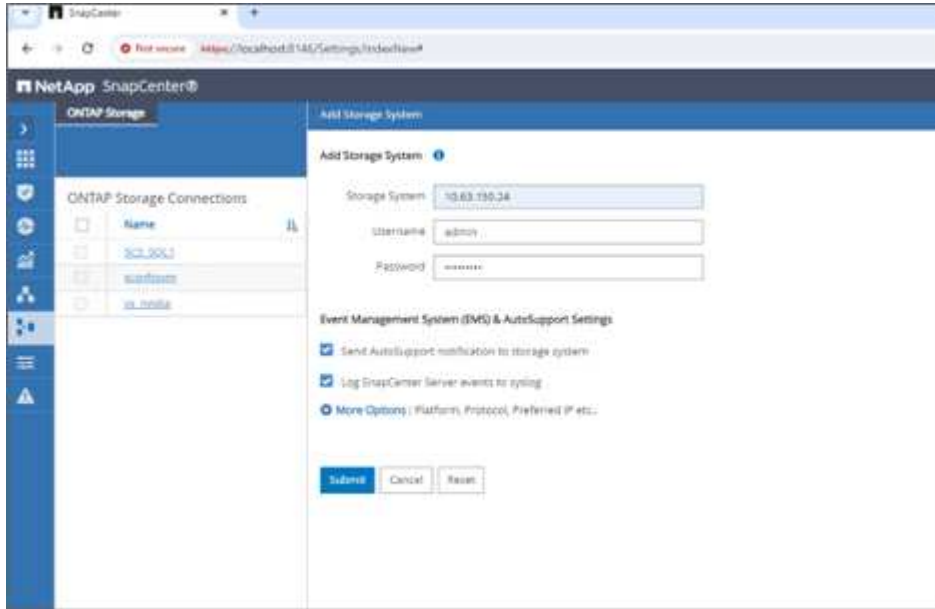
1. Impostare l'host che verrà utilizzato per eseguire i comandi SnapCenter.



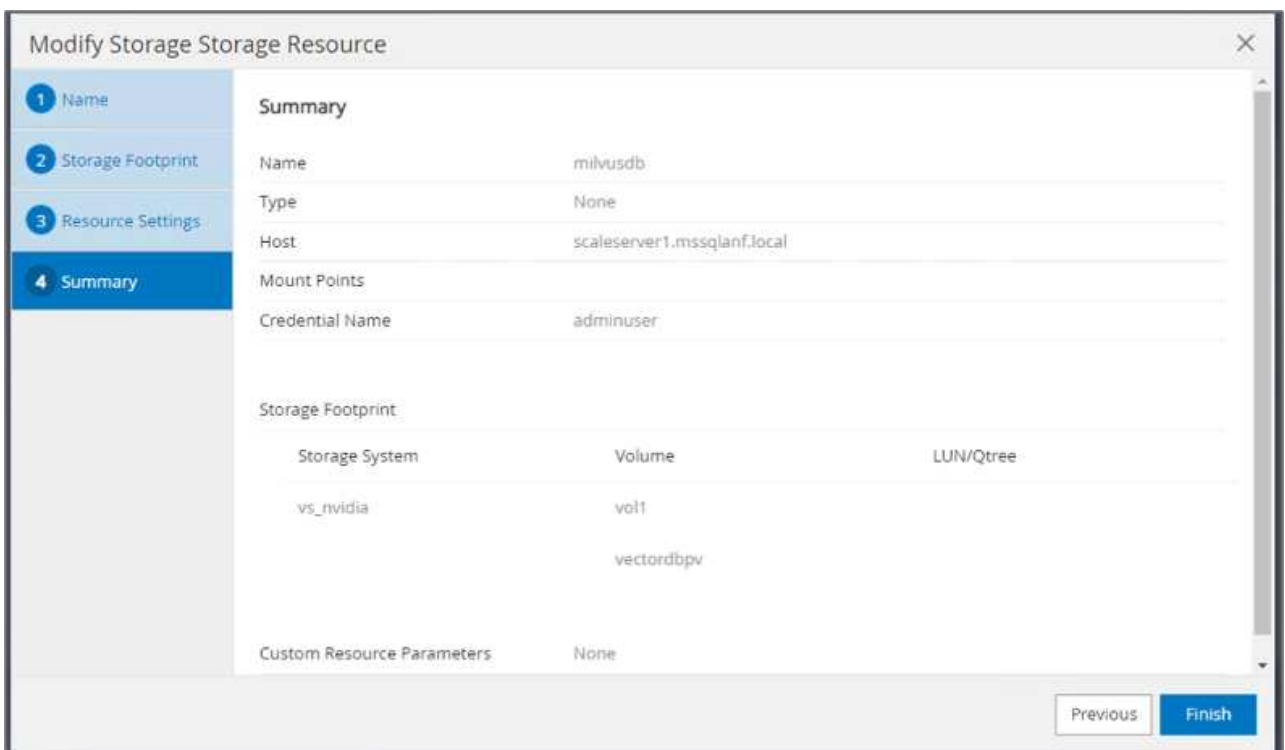
2. Installare e configurare il plug-in di archiviazione. Dall'host aggiunto, selezionare "altre opzioni". Individuare e selezionare il plug-in di archiviazione scaricato dal "NetApp Automation Store". Installare il plugin e salvare la configurazione.



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



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



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

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

=== start connecting to Milvus      ===

=== Milvus host: localhost          ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_test` ===

=== Start inserting entities        ===

Number of entities in hello_milvus_netapp_sc_test: 3000

=== Create collection `hello_milvus_netapp_sc_test2` ===

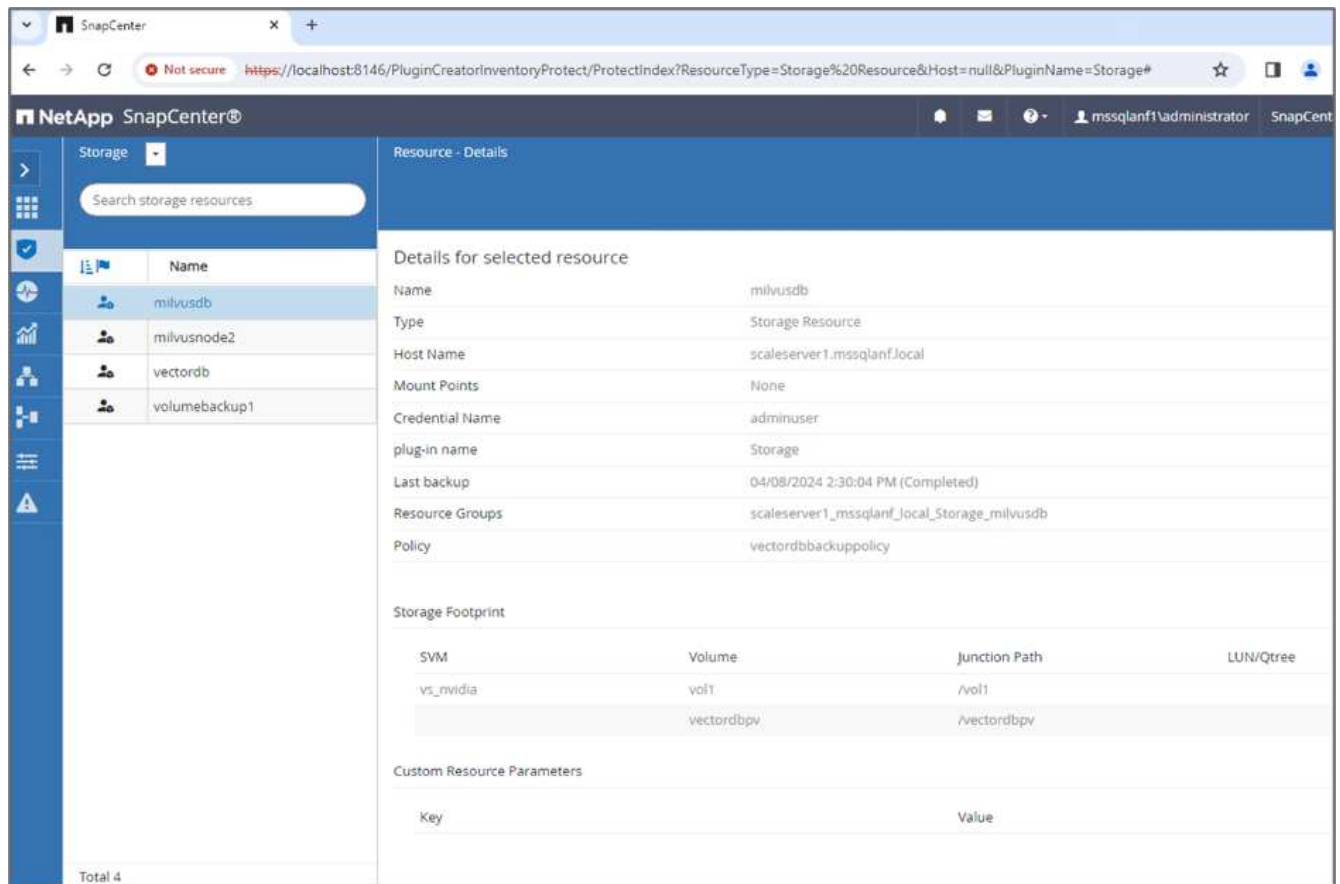
Number of entities in hello_milvus_netapp_sc_test2: 6000
root@node2:~# for i in 2 3 4 5 6    ; do ssh node$i "hostname; sync; echo
'sync executed';" ; done
node2
sync executed
node3
sync executed
node4
sync executed
node5
sync executed
node6
sync executed
root@node2:~#
```

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

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

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

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



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

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

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

=== start connecting to Milvus      ===

=== Milvus host: localhost          ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_testnew` ===

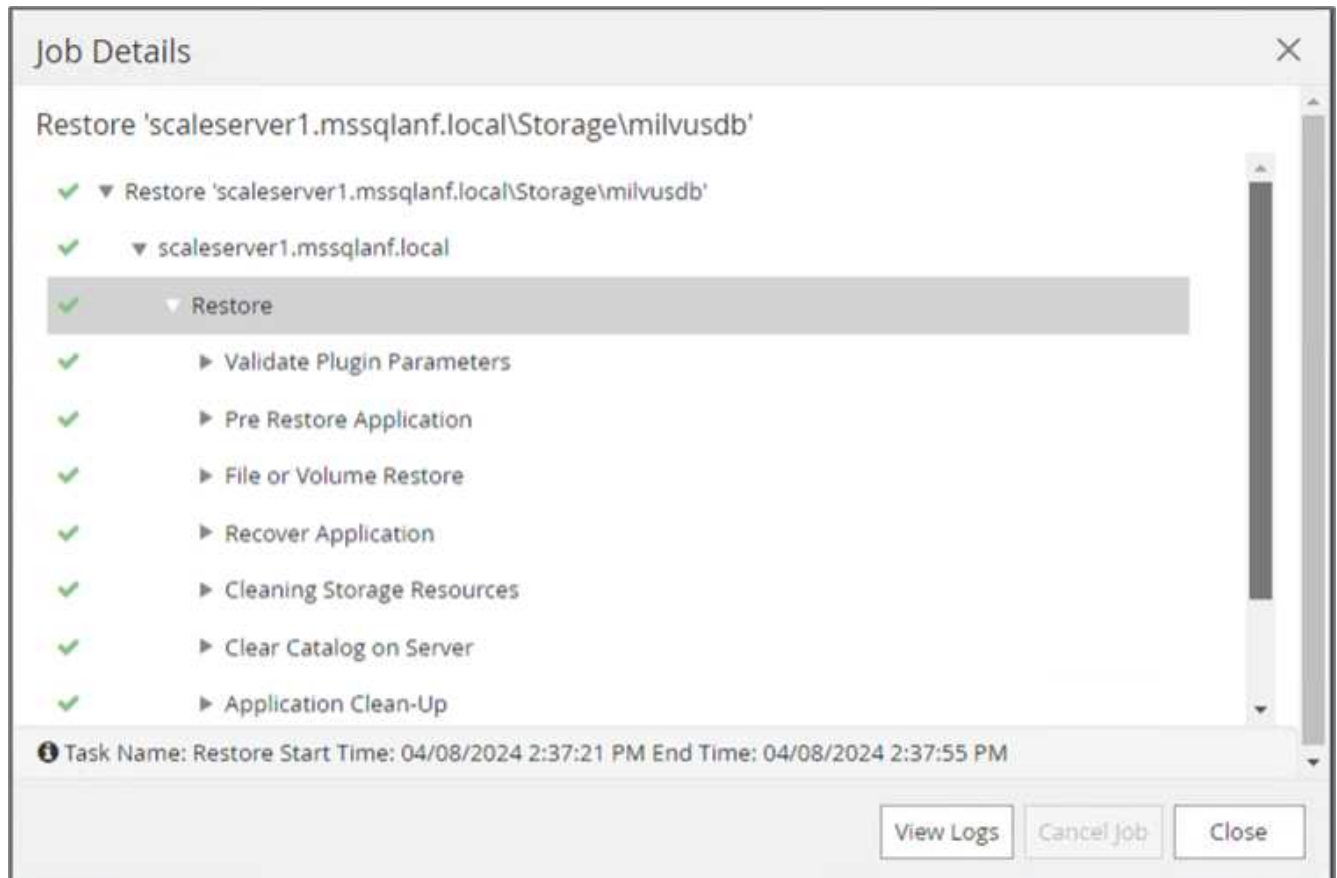
=== Start inserting entities        ===

Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

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

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



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

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

=== start connecting to Milvus ===

=== Milvus host: localhost ===

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

=== Start Creating index IVF_FLAT ===

=== Start loading ===

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

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

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

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

```

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

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

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

=== Start Creating index IVF_FLAT ===

=== Start loading ===

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

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

```

```

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

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

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

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

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

```

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

```

root@node2:~# python3 verify_data_netapp.py

=== start connecting to Milvus      ===

=== Milvus host: localhost         ===

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

```

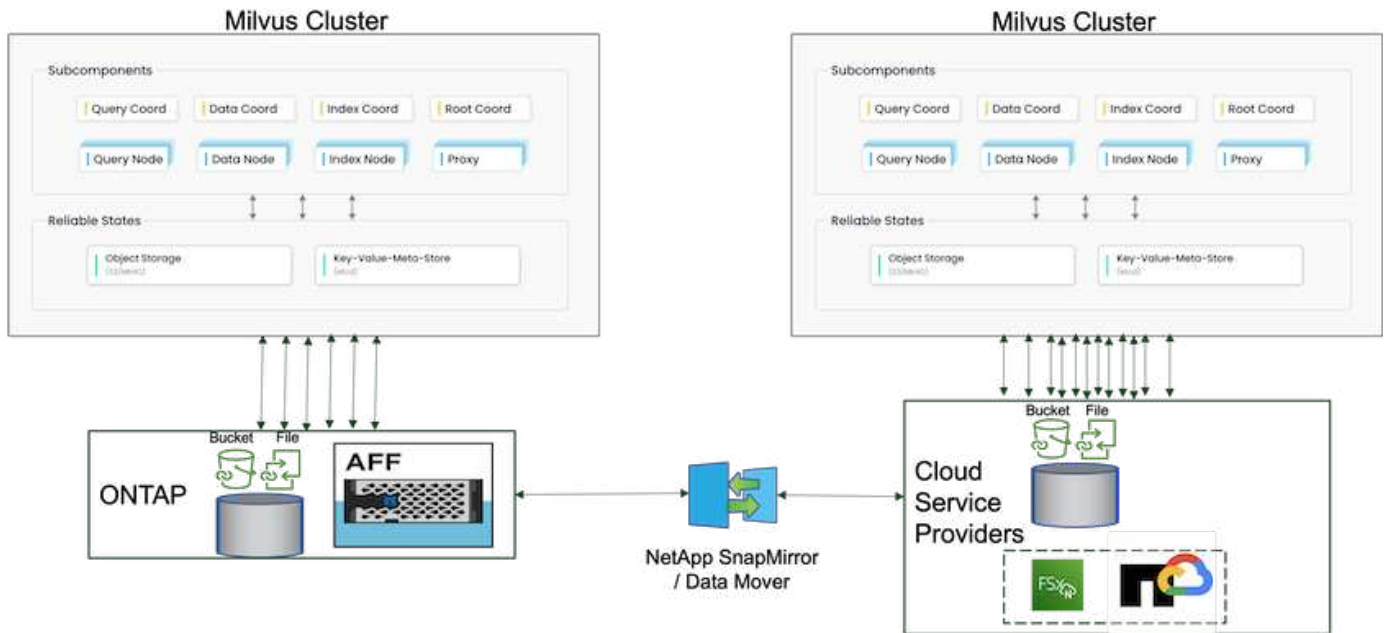
In conclusione, l'utilizzo di SnapCenter di NetApp per la salvaguardia dei dati di database vettoriali e dei dati Milvus che risiedono in ONTAP offre notevoli vantaggi ai clienti, in particolare nei settori in cui l'integrità dei dati è di primaria importanza, come la produzione cinematografica. La capacità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

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

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

Disaster recovery con NetApp SnapMirror

Disaster recovery con NetApp SnapMirror



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

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

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

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

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

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

Convalida delle prestazioni del database vettoriale

Convalida delle performance

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

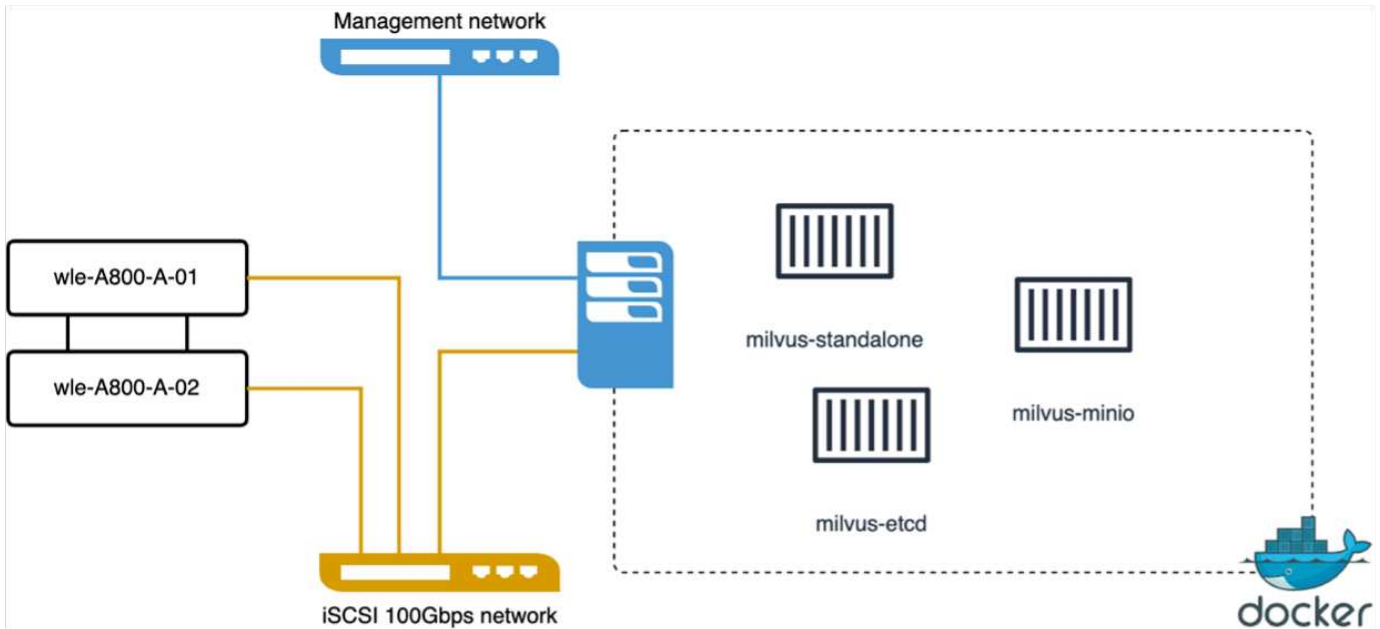
In questa sezione, il nostro obiettivo è analizzare la convalida delle prestazioni dei database vettoriali, come Milvus e pgvecto.rs, concentrandoci sulle caratteristiche delle prestazioni di storage, come il profilo i/o e il comportamento dello storage controller NetApp a supporto di RAG e carichi di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

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

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

VectorDB-Bench con cluster standalone Milvus

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



In questa sezione, condividiamo le nostre osservazioni e i risultati dei test del database autonomo Milvus.

. Per questi test è stato selezionato DiskANN come tipo di indice.

. L'acquisizione, l'ottimizzazione e la creazione di indici per un set di dati di circa 100GB TB ha richiesto circa 5 ore. Per la maggior parte di questa durata, il server Milvus, dotato di 20 core (che equivale a 40 vcpus quando Hyper-Threading è abilitato), funzionava alla sua capacità massima della CPU del 100%.abbiamo scoperto che DiskANN è particolarmente importante per grandi set di dati che superano le dimensioni della memoria del sistema.

. Nella fase di query, abbiamo osservato un tasso di query al secondo (QPS) di 10,93 con un richiamo di 0,9987. La latenza percentile di 99th ms per le query è stata misurata a 708,2 millisecondi.

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

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

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

Il risultato del vectorDB-bench è riportato di seguito.

Vector Database Benchmark

Filtering Search Performance Test (5M Dataset, 1536 Dim, Filter 1%)

Qps (more is better)

Milvus  10.93

Recall (more is better)

Milvus  0.9987

Load_duration (less is better)

Milvus  18,360s

Serial_latency_p99 (less is better)

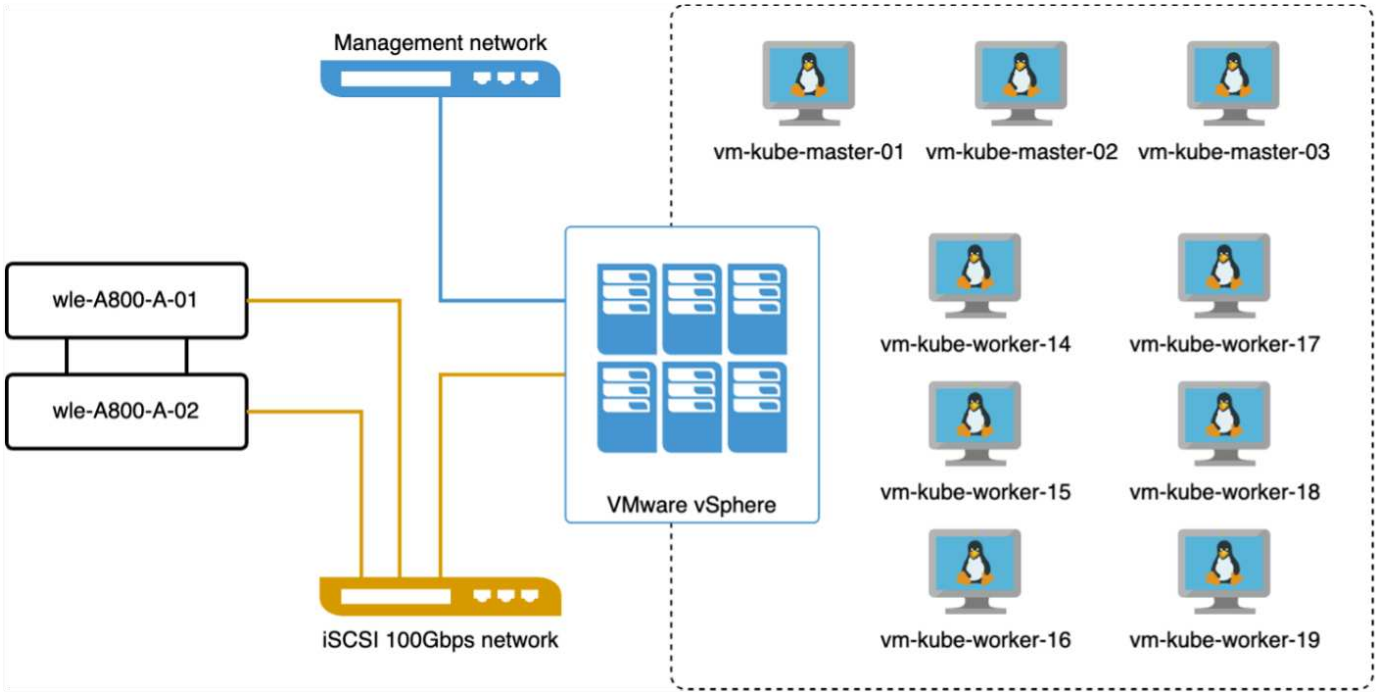
Milvus  708.2ms

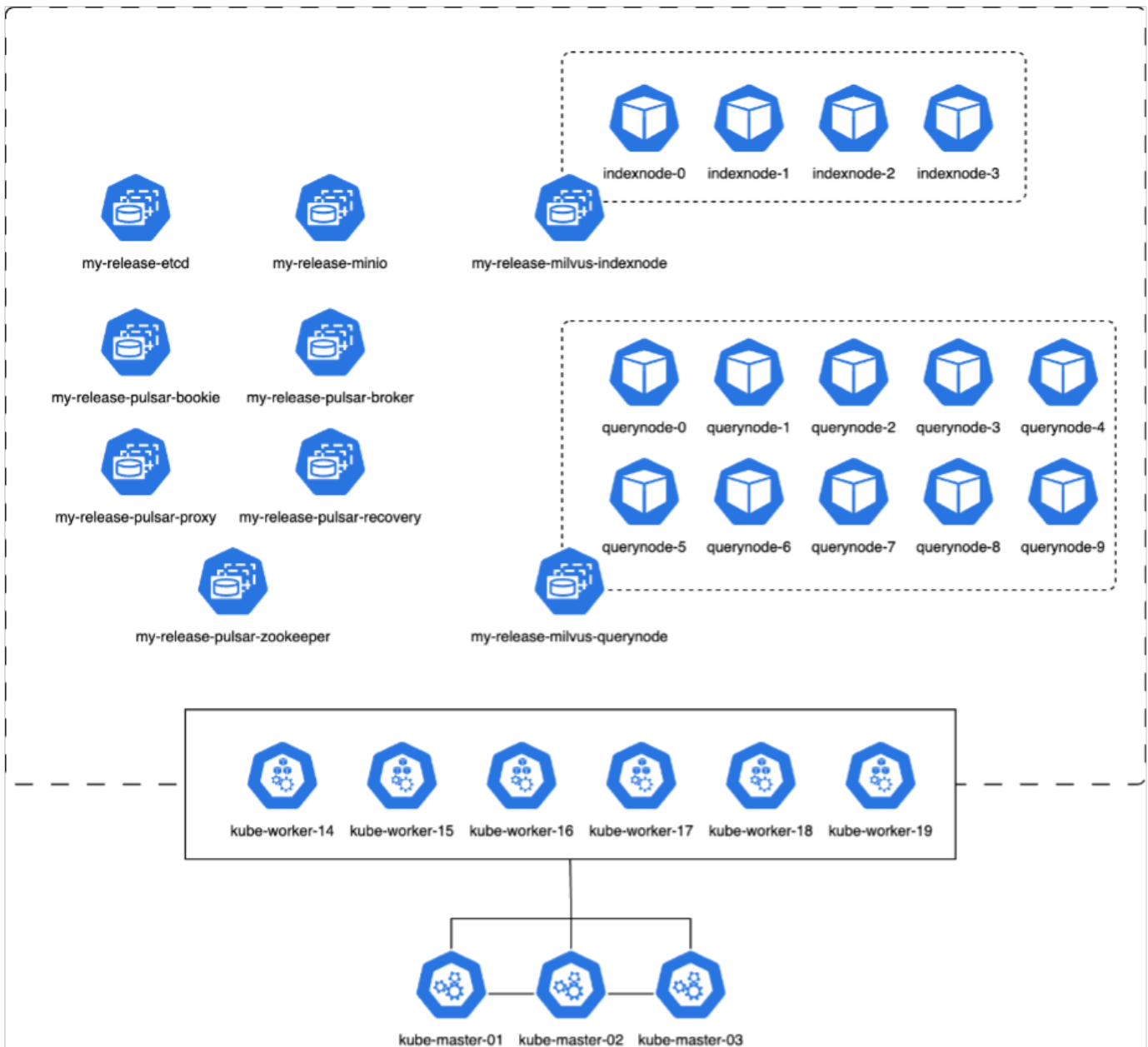
Dalla convalida delle prestazioni dell'istanza autonoma di Milvus, è evidente che l'impostazione corrente non è sufficiente a supportare un set di dati di 5 milioni di vettori con una dimensione di 1536. abbiamo determinato che lo storage dispone di risorse adeguate e non costituisce un collo di bottiglia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

VectorDB-Bench con cluster milvus

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

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





In questa sezione, presentiamo le nostre osservazioni e i risultati dei test del database Milvus.

* Il tipo di indice utilizzato era DiskANN.

* La tabella riportata di seguito fornisce un confronto tra le distribuzioni standalone e cluster quando si utilizzano 5 milioni di vettori con una dimensione di 1536. Abbiamo osservato che il tempo richiesto per l'acquisizione dei dati e l'ottimizzazione post-inserimento è stato inferiore nell'implementazione del cluster. La latenza percentile di 99th ms per le query è stata ridotta di sei volte nell'implementazione del cluster rispetto alla configurazione standalone.

* Sebbene il tasso di query al secondo (QPS) fosse più elevato nella distribuzione del cluster, non era al livello desiderato.

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

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



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

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

Presentiamo i dettagli del profilo i/o dello storage in base alla convalida delle performance sia dello storage Milvus che del cluster Milvus.

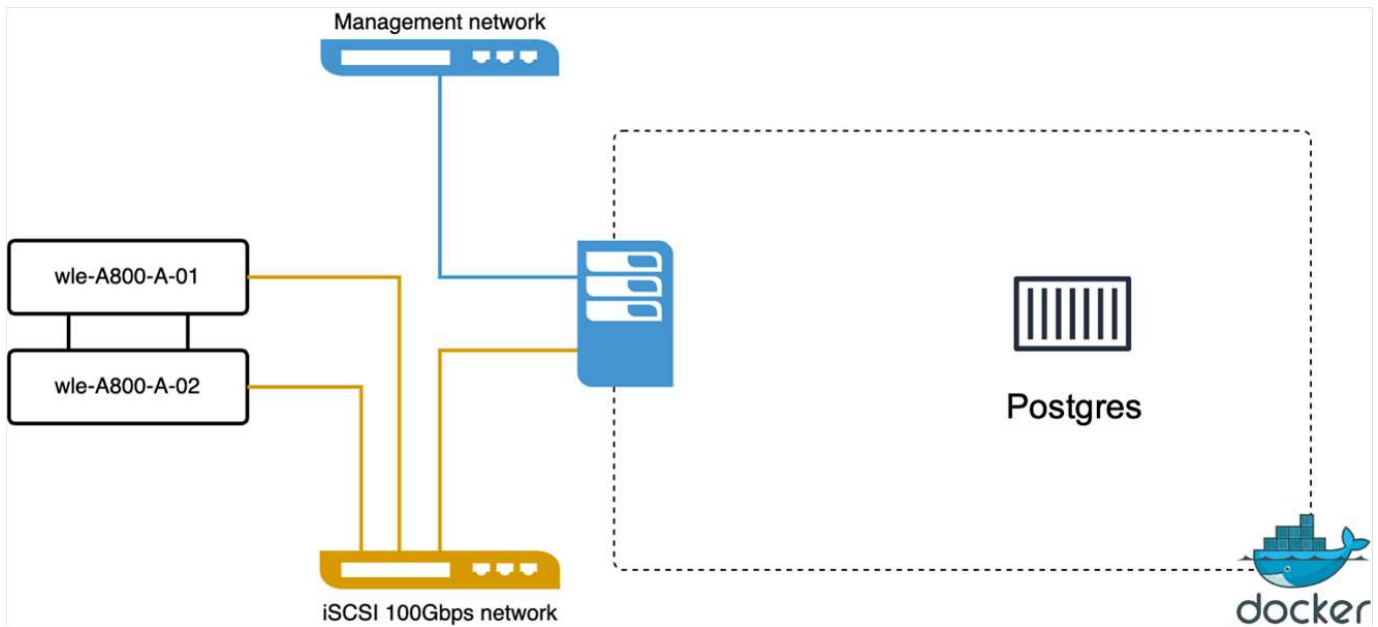
* Abbiamo osservato che il profilo di i/o rimane coerente sia nelle implementazioni standalone che in cluster.

* La differenza osservata negli IOPS di picco può essere attribuita al maggior numero di client nell'implementazione del cluster.

VettorDB-Bench con Postgres (pgvecto.rs)

Abbiamo eseguito le seguenti azioni su PostgreSQL(pgvecto.rs) utilizzando VectorDB-Bench:

I dettagli relativi alla connettività di rete e server di PostgreSQL (in particolare pgvecto.rs) sono i seguenti:



In questa sezione, condividiamo le nostre osservazioni e i risultati dei test del database PostgreSQL, in particolare utilizzando pgvecto.rs.

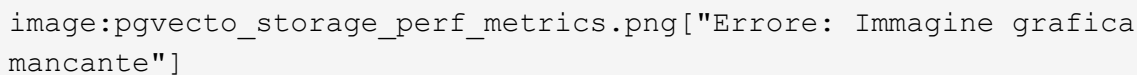
* Abbiamo selezionato HNSW come tipo di indice per questi test perché al momento del test, DiskANN non era disponibile per pgvecto.rs.

* Durante la fase di acquisizione dei dati, è stato caricato il set di dati Cohere, che consiste di 10 milioni di vettori con una dimensione di 768. Questo processo ha richiesto circa 4,5 ore.

* Nella fase di query, abbiamo osservato un tasso di query al secondo (QPS) di 1.068 con un richiamo di 0,6344. La latenza percentile di 99th ms per le query è stata misurata a 20 millisecondi. Per la maggior parte del runtime, la CPU del client funzionava al 100% della capacità.

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

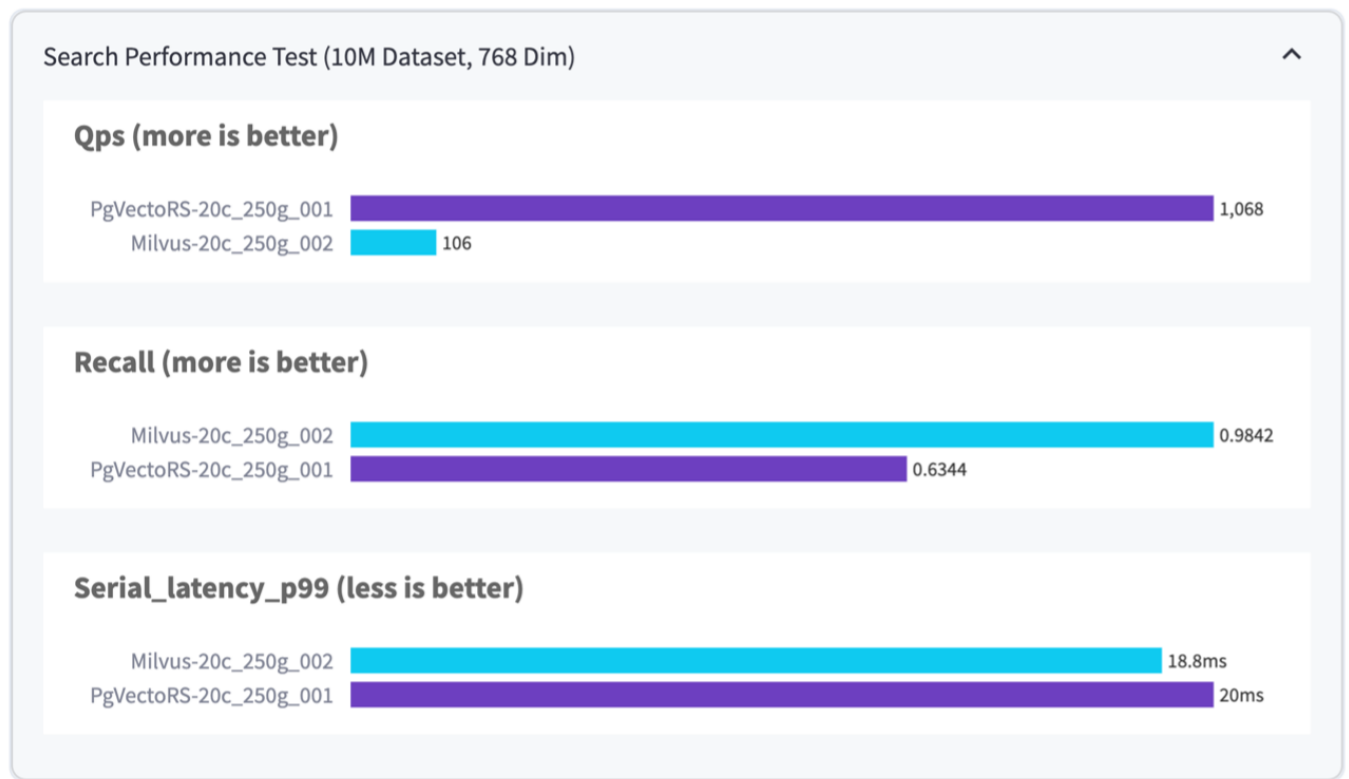


The following section presents the key storage performance metrics.


Confronto delle prestazioni tra milvus e postgres su DB Bench vettoriale

Vector Database Benchmark

Note that all testing was completed in July 2023, except for the times already noted.



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

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

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

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

Osservazione basata sulle nostre convalide delle prestazioni:

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

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

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

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

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

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

Database vettoriale con Instaclustr utilizzando PostgreSQL: Pgvector

Database vettoriale con Instaclustr utilizzando PostgreSQL: Pgvector

In questa sezione, approfondiremo le specifiche di come il prodotto instaclustr si integra con PostgreSQL sulla funzionalità pgvector. Abbiamo un esempio di "come migliorare la vostra accuratezza e le prestazioni LLM con PGVector e PostgreSQL®: Introduzione alle incorporazioni e il ruolo di PGVector". Controllare ["blog"](#) per ottenere ulteriori informazioni.

Casi di utilizzo del database vettoriale

Casi di utilizzo del database vettoriale

In questa sezione, vengono illustrati due casi di utilizzo, come il recupero della generazione aumentata con modelli di linguaggio di grandi dimensioni e il chatbot NetApp IT.

Retrieval Augmented Generation (RAG) con modelli di linguaggio grande (LLMs)

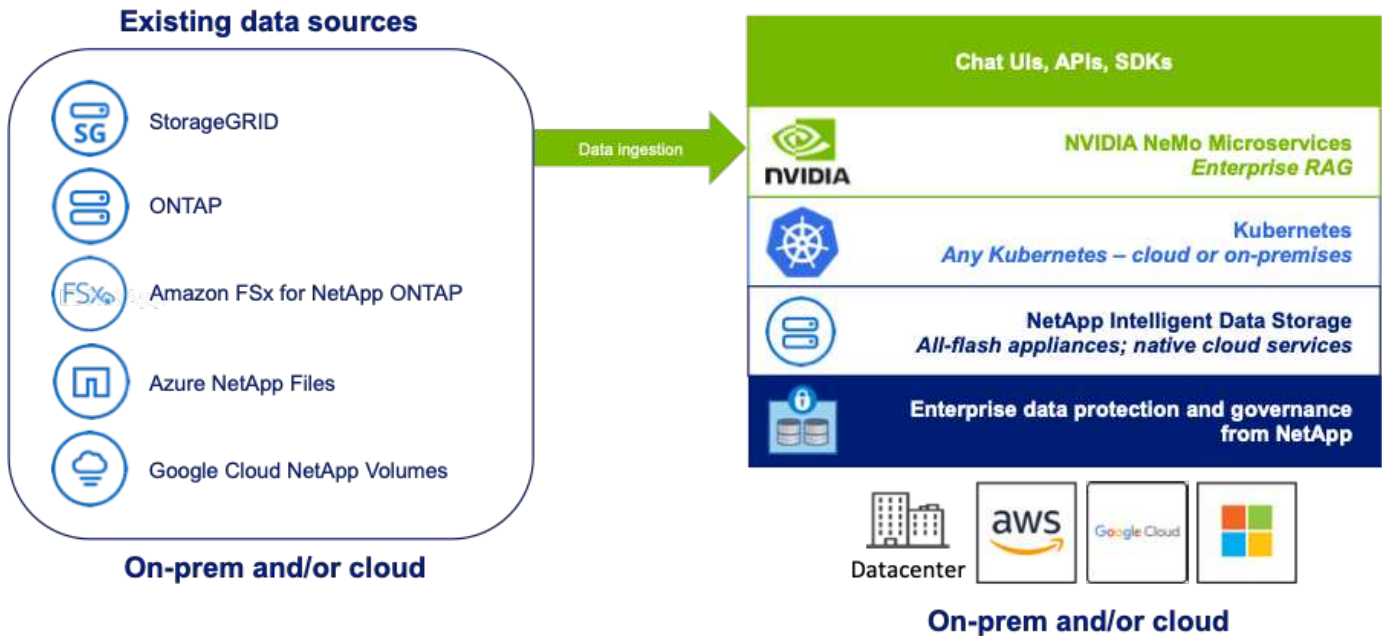
```
Retrieval-augmented generation, or RAG, is a technique for enhancing the accuracy and reliability of Large Language Models, or LLMs, by augmenting prompts with facts fetched from external sources. In a traditional RAG deployment, vector embeddings are generated from an existing dataset and then stored 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 in modo da utilizzare Milvus o pgvecto come database vettoriale per la memorizzazione di embeddings della knowledgebase. Per ulteriori informazioni, consultare la documentazione.

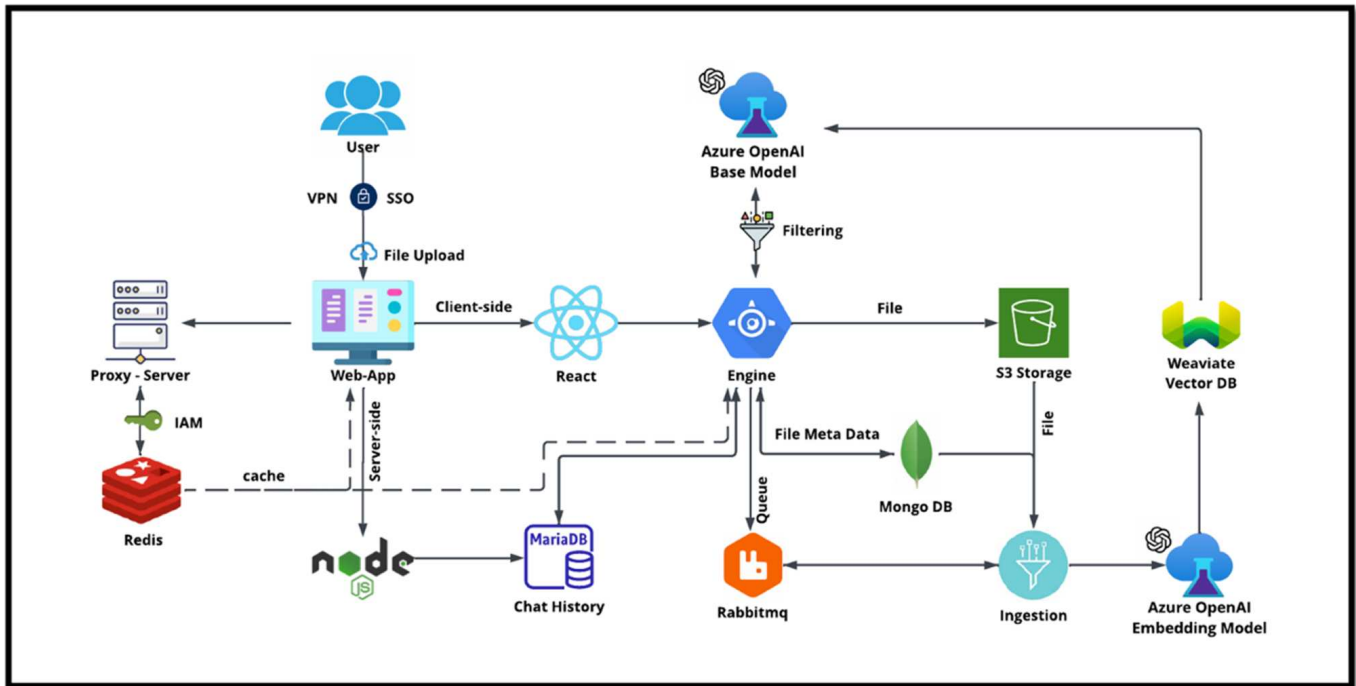
NetApp has validated an enterprise RAG architecture powered by the NVIDIA Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog post for more information and to see a demo. Figure 1 provides an overview of this architecture.

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



Caso d'utilizzo di NetApp IT chatbot

Il chatbot di NetApp funge da altro caso di utilizzo in tempo reale per il database vettoriale. In questo esempio, NetApp Private Openai Sandbox fornisce una piattaforma efficace, sicura ed efficiente per la gestione delle query da parte degli utenti interni di NetApp. Incorporando rigorosi protocolli di sicurezza, sistemi di gestione dei dati efficienti e sofisticate funzionalità di elaborazione dell'ai, garantisce risposte precise e di alta qualità agli utenti in base al loro ruolo e alle loro responsabilità nell'organizzazione tramite l'autenticazione SSO. Questa architettura mette in evidenza il potenziale della fusione di tecnologie avanzate per creare sistemi intelligenti e orientati all'utente.



Il caso di utilizzo può essere suddiviso in quattro sezioni principali.

Autenticazione e verifica utente:

- Le richieste degli utenti vengono inoltrate tramite il processo SSO (NetApp Single Sign-on) per confermare l'identità dell'utente.
- Una volta eseguita l'autenticazione, il sistema controlla la connessione VPN per garantire una trasmissione dati sicura.

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 memorizzare i dati degli utenti.
- MariaDB invia quindi le informazioni all'istanza di NetApp Azure, che connette 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 della risposta:

- La risposta del modello di base viene controllata per garantire che sia accurata e soddisfi gli standard di contenuto.
- Dopo aver superato il controllo, la risposta viene rinviata all'utente. Questo processo assicura che l'utente riceva una risposta chiara, precisa e appropriata alla sua domanda.

Conclusione

Conclusione

In conclusione, questo documento fornisce una panoramica completa sulla distribuzione e la gestione di database vettoriali, come Milvus e pgvector, su soluzioni di storage NetApp. Abbiamo discusso delle linee guida per l'infrastruttura relative all'utilizzo dello storage a oggetti NetApp ONTAP e StorageGRID e validato il database Milvus in AWS FSX per NetApp ONTAP tramite l'archivio di file e oggetti.

Abbiamo esplorato il dualismo degli oggetti file di NetApp, dimostrando la sua utility non solo per i dati in database vettoriali ma anche per altre applicazioni. Abbiamo inoltre evidenziato come SnapCenter, il prodotto per la gestione aziendale di NetApp, offra funzionalità di backup, ripristino e clonazione per dati di database vettoriali, garantendo integrità e disponibilità dei dati.

Questo documento spiega anche in che modo la soluzione di cloud ibrido NetApp offre replica e protezione dei dati in ambienti on-premise e cloud, offrendo un'esperienza di gestione dei dati perfetta e sicura. Abbiamo fornito informazioni dettagliate sulla convalida delle prestazioni di database vettoriali come Milvus e pgvector su NetApp ONTAP, offrendo informazioni preziose sulla loro efficienza e scalabilità.

Infine, abbiamo discusso di due casi d'utilizzo ai generativi: RAG with LLM e Chatai interno di NetApp. Questi esempi pratici sottolineano le applicazioni reali e i vantaggi dei concetti e delle pratiche delineati in questo documento. Nel complesso, questo documento è una guida completa per tutti coloro che desiderano sfruttare le potenti soluzioni storage di NetApp per la gestione di database vettoriali.

Ringraziamenti

L'autore gradisce i ringraziamenti sentiti ai seguenti contributori, altri che hanno fornito il loro feedback e commenti per rendere questo documento importante ai clienti di NetApp e ai campi di NetApp.

1. Sathish Thyagarajan, Product marketing manager, ONTAP ai & Analytics, NetApp
2. Mike Oglesby, Technical Marketing Engineer, NetApp
3. AJ Mahajan, Direttore Generale, NetApp
4. Joe Scott, Manager, Ingegneria delle prestazioni del carico di lavoro, NetApp
5. Puneet Dhawan, Direttore Senior, Gestione prodotti FSX, NetApp
6. Yuval Kalderon, Senior Product Manager, team FSX Product, NetApp

Dove trovare ulteriori informazioni

Per ulteriori informazioni sulle informazioni descritte in questo documento, consultare i seguenti documenti e/o siti Web:

- Documentazione Milvus - <https://milvus.io/docs/overview.md>
- Documentazione autonoma Milvus - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- Documentazione sui prodotti NetApp
<https://www.netapp.com/support-and-training/documentation/>
- istaclustr - "documentazione di installazione"

Cronologia delle versioni

| Versione | Data | Cronologia delle versioni del documento |
|--------------|-------------|---|
| Versione 1.0 | Aprile 2024 | Release iniziale |

Appendice A: Values.yaml

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

```

```

# LoadBalancerSourceRange 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: LS0tLS1CRUdJTtBQU--REDUCT
#   crt: LS0tLS1CRUdJTtBDR--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 zookeeper 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: apache/pulsar
    pullPolicy: IfNotPresent
    tag: 2.8.2
  autorecovery:
    repository: apache/pulsar
    tag: 2.8.2
    pullPolicy: IfNotPresent
  zookeeper:
    repository: apache/pulsar
    pullPolicy: IfNotPresent
    tag: 2.8.2
  bookie:
    repository: apache/pulsar
    pullPolicy: IfNotPresent
    tag: 2.8.2
  proxy:
    repository: apache/pulsar
    pullPolicy: IfNotPresent
    tag: 2.8.2
  pulsar_manager:
    repository: apache/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: apache/pulsar/pulsar
    tag: 2.8.2

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

```

kafka:
  enabled: false
  name: kafka
  replicaCount: 3
  image:
    repository: bitnami/kafka
    tag: 3.1.0-debian-10-r52
  ## Increase graceful termination for kafka graceful shutdown
  terminationGracePeriodSeconds: "90"
  pdb:
    create: false

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

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

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

  metrics:
    ## Prometheus Kafka exporter: exposes complimentary metrics to JMX
  exporter
    kafka:

```

```

    enabled: false
    image:
      repository: bitnami/kafka-exporter
      tag: 1.4.2-debian-10-r182

## Prometheus JMX exporter: exposes the majority of Kafkas metrics
jmx:
  enabled: false
  image:
    repository: bitnami/jmx-exporter
    tag: 0.16.1-debian-10-r245

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

service:
  type: ClusterIP
  ports:
    client: 9092

zookeeper:
  enabled: true
  replicaCount: 3

#####
# External S3
# - these configs are only used when `externalS3.enabled` is true
#####
externalS3:
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
  useSSL: false
  bucketName: "milvusdbvoll1"
  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

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 builtin 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

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:~#

```

Appendix D: docker-compose.yml

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

Copyright © 2024 NetApp, Inc. Tutti i diritti riservati. Stampato negli Stati Uniti d'America. Nessuna porzione di questo documento soggetta a copyright può essere riprodotta in qualsiasi formato o mezzo (grafico, elettronico o meccanico, inclusi fotocopie, registrazione, nastri o storage in un sistema elettronico) senza previo consenso scritto da parte del detentore del copyright.

Il software derivato dal materiale sottoposto a copyright di NetApp è soggetto alla seguente licenza e dichiarazione di non responsabilità:

IL PRESENTE SOFTWARE VIENE FORNITO DA NETAPP "COSÌ COM'È" E SENZA QUALSIVOGLIA TIPO DI GARANZIA IMPLICITA O ESPRESSA FRA CUI, A TITOLO ESEMPLIFICATIVO E NON ESAUSTIVO, GARANZIE IMPLICITE DI COMMERCIALIZZABILITÀ E IDONEITÀ PER UNO SCOPO SPECIFICO, CHE VENGONO DECLINATE DAL PRESENTE DOCUMENTO. NETAPP NON VERRÀ CONSIDERATA RESPONSABILE IN ALCUN CASO PER QUALSIVOGLIA DANNO DIRETTO, INDIRETTO, ACCIDENTALE, SPECIALE, ESEMPLARE E CONSEGUENZIALE (COMPRESI, A TITOLO ESEMPLIFICATIVO E NON ESAUSTIVO, PROCUREMENT O SOSTITUZIONE DI MERCI O SERVIZI, IMPOSSIBILITÀ DI UTILIZZO O PERDITA DI DATI O PROFITTI OPPURE INTERRUZIONE DELL'ATTIVITÀ AZIENDALE) CAUSATO IN QUALSIVOGLIA MODO O IN RELAZIONE A QUALUNQUE TEORIA DI RESPONSABILITÀ, SIA ESSA CONTRATTUALE, RIGOROSA O DOVUTA A INSOLVENZA (COMPRESA LA NEGLIGENZA O ALTRO) INSORTA IN QUALSIASI MODO ATTRAVERSO L'UTILIZZO DEL PRESENTE SOFTWARE ANCHE IN PRESENZA DI UN PREAVVISO CIRCA L'EVENTUALITÀ DI QUESTO TIPO DI DANNI.

NetApp si riserva il diritto di modificare in qualsiasi momento qualunque prodotto descritto nel presente documento senza fornire alcun preavviso. NetApp non si assume alcuna responsabilità circa l'utilizzo dei prodotti o materiali descritti nel presente documento, con l'eccezione di quanto concordato espressamente e per iscritto da NetApp. L'utilizzo o l'acquisto del presente prodotto non comporta il rilascio di una licenza nell'ambito di un qualche diritto di brevetto, marchio commerciale o altro diritto di proprietà intellettuale di NetApp.

Il prodotto descritto in questa guida può essere protetto da uno o più brevetti degli Stati Uniti, esteri o in attesa di approvazione.

LEGENDA PER I DIRITTI SOTTOPOSTI A LIMITAZIONE: l'utilizzo, la duplicazione o la divulgazione da parte degli enti governativi sono soggetti alle limitazioni indicate nel sottoparagrafo (b)(3) della clausola Rights in Technical Data and Computer Software del DFARS 252.227-7013 (FEB 2014) e FAR 52.227-19 (DIC 2007).

I dati contenuti nel presente documento riguardano un articolo commerciale (secondo la definizione data in FAR 2.101) e sono di proprietà di NetApp, Inc. Tutti i dati tecnici e il software NetApp forniti secondo i termini del presente Contratto sono articoli aventi natura commerciale, sviluppati con finanziamenti esclusivamente privati. Il governo statunitense ha una licenza irrevocabile limitata, non esclusiva, non trasferibile, non cedibile, mondiale, per l'utilizzo dei Dati esclusivamente in connessione con e a supporto di un contratto governativo statunitense in base al quale i Dati sono distribuiti. Con la sola esclusione di quanto indicato nel presente documento, i Dati non possono essere utilizzati, divulgati, riprodotti, modificati, visualizzati o mostrati senza la previa approvazione scritta di NetApp, Inc. I diritti di licenza del governo degli Stati Uniti per il Dipartimento della Difesa sono limitati ai diritti identificati nella clausola DFARS 252.227-7015(b) (FEB 2014).

Informazioni sul marchio commerciale

NETAPP, il logo NETAPP e i marchi elencati alla pagina <http://www.netapp.com/TM> sono marchi di NetApp, Inc. Gli altri nomi di aziende e prodotti potrebbero essere marchi dei rispettivi proprietari.