Vector Datenbanklösung mit NetApp

NetApp artificial intelligence solutions

NetApp
August 18, 2025

Inhalt

Vector Datenbanklösung mit NetApp

Vector Datenbanklösung mit NetApp

Karthikeyan Nagalingam und Rodrigo Nascimento, NetApp

Dieses Dokument bietet eine umfassende Untersuchung der Bereitstellung und Verwaltung von Vektordatenbanken wie Milvus und pgvecto, einer Open-Source-PostgreSQL-Erweiterung, unter Verwendung der Speicherlösungen von NetApp. Es beschreibt detailliert die Infrastrukturrichtlinien für die Verwendung von NetApp ONTAP und StorageGRID Objektspeicher und validiert die Anwendung der Milvus-Datenbank in AWS FSx ONTAP. Das Dokument erläutert die Datei-Objekt-Dualität von NetApp und ihren Nutzen für Vektordatenbanken und Anwendungen, die Vektoreinbettungen unterstützen. Es betont die Fähigkeiten von SnapCenter, dem Enterprise-Management-Produkt von NetApp, das Sicherungs- und Wiederherstellungsfunktionen für Vektordatenbanken bietet und so die Datenintegrität und -verfügbarkeit sicherstellt. Das Dokument befasst sich eingehender mit der Hybrid-Cloud-Lösung von NetApp und erörtert ihre Rolle bei der Datenreplikation und -sicherung in lokalen und Cloud-Umgebungen. Es enthält Einblicke in die Leistungsvalidierung von Vektordatenbanken auf NetApp ONTAP und schließt mit zwei praktischen Anwendungsfällen zur generativen KI: RAG mit LLM und NetApps internem ChatAI. Dieses Dokument dient als umfassender Leitfaden zur Nutzung der Speicherlösungen von NetApp für die Verwaltung von Vektordatenbanken.

Der Schwerpunkt der Referenzarchitektur liegt auf Folgendem:

12. "Anhang C: verify_data_netapp.py"
13. "Anhang D: docker-compose.yml"

Einführung

Dieser Abschnitt bietet eine Einführung in die Vektordatenbanklösung für NetApp.

Einführung

Vektordatenbanken bewältigen effektiv die Herausforderungen, die für die Bewältigung der Komplexität der semantischen Suche in Large Language Models (LLMs) und generativer künstlicher Intelligenz (KI) entwickelt wurden. Im Gegensatz zu herkömmlichen Datenverwaltungssystemen können Vektordatenbanken verschiedene Datentypen verarbeiten und durchsuchen, darunter Bilder, Videos, Text, Audio und andere Formen unstrukturierter Daten, indem sie den Inhalt der Daten selbst und nicht Beschriftungen oder Tags verwenden.

Die Einschränkungen relationaler Datenbankmanagementsysteme (RDBMS) sind gut dokumentiert, insbesondere ihre Probleme mit hochdimensionalen Datendarstellungen und unstrukturierten Daten, die in KI-Anwendungen üblich sind. RDBMS erfordern häufig einen zeitaufwändigen und fehleranfälligen Prozess zur Vereinfachung der Daten in besser handhabbare Strukturen, was zu Verzögerungen und Ineffizienzen bei der Suche führt. Vektordatenbanken sind jedoch darauf ausgelegt, diese Probleme zu umgehen. Sie bieten eine effizientere und genauere Lösung für die Verwaltung und Suche in komplexen und hochdimensionalen Daten und erleichtern so die Weiterentwicklung von KI-Anwendungen.

Dieses Dokument dient als umfassender Leitfaden für Kunden, die derzeit Vektordatenbanken verwenden oder dies planen. Es beschreibt die Best Practices für die Verwendung von Vektordatenbanken auf Plattformen wie NetApp ONTAP, NetApp StorageGRID, Amazon FSx ONTAP für NetApp ONTAP und SnapCenter. Die hier bereitgestellten Inhalte decken eine Reihe von Themen ab:

- Infrastrukturrichtlinien für Vektordatenbanken wie Milvus, bereitgestellt von NetApp Storage über NetApp ONTAP und StorageGRID Objektspeicher.
- Validierung der Milvus-Datenbank in AWS FSx ONTAP durch Datei- und Objektspeicher.
- Befasst sich eingehend mit der Datei-Objekt-Dualität von NetApp und demonstriert deren Nutzen für Daten in Vektordatenbanken sowie anderen Anwendungen.
- So bietet SnapCenter, das Data Protection Management-Produkt von NetApp, Sicherungs- und Wiederherstellungsfunktionen für Vektordatenbankdaten.
- So bietet die Hybrid Cloud von NetApp Datenreplikation und -schutz in lokalen und Cloud-Umgebungen.
- Bietet Einblicke in die Leistungsvalidierung von Vektordatenbanken wie Milvus und pgvector auf NetApp ONTAP.
- Zwei konkrete Anwendungsfälle: Retrieval Augmented Generation (RAG) mit Large Language Models (LLM) und ChatAI des NetApp IT-Teams, die praktische Beispiele für die beschriebenen Konzepte und Praktiken bieten.

Lösungsübersicht

Dieser Abschnitt bietet einen Überblick über die NetApp Vector-Datenbanklösung.

Lösungsübersicht

Diese Lösung demonstriert die besonderen Vorteile und Fähigkeiten, die NetApp mitbringt, um die Herausforderungen zu bewältigen, vor denen Kunden von Vektordatenbanken stehen. Durch die Nutzung von NetApp ONTAP, StorageGRID, den Cloud-Lösungen von NetApp und SnapCenter können Kunden ihren Geschäftsbetrieb erheblich aufwerten. Diese Tools lösen nicht nur bestehende Probleme, sondern steigern auch die Effizienz und Produktivität und tragen so zum allgemeinen Unternehmenswachstum bei.

Warum NetApp?

- Die Angebote von NetApp, wie ONTAP und StorageGRID, ermöglichen die Trennung von Speicher und Rechenleistung und ermöglichen so eine optimale Ressourcennutzung basierend auf spezifischen Anforderungen. Diese Flexibilität ermöglicht es Kunden, ihren Speicher mithilfe von NetApp Speicherlösungen unabhängig zu skalieren.
- Durch die Nutzung der Speichercontroller von NetApp können Kunden mithilfe der Protokolle NFS und S3 effizient Daten an ihre Vektordatenbank übermitteln. Diese Protokolle erleichtern die Speicherung von Kundendaten und verwalten den Vektordatenbankindex, sodass nicht mehr mehrere Kopien der Daten benötigt werden, auf die über Datei- und Objektmethoden zugegriffen wird.
- NetApp ONTAP bietet native Unterstützung für NAS- und Objektspeicher bei führenden Cloud-Service-Anbietern wie AWS, Azure und Google Cloud. Diese umfassende Kompatibilität gewährleistet eine nahtlose Integration und ermöglicht Kundendatenmobilität, globale Zugänglichkeit, Notfallwiederherstellung, dynamische Skalierbarkeit und hohe Leistung.
- Dank der robusten Datenverwaltungsfunktionen von NetApp können Kunden sicher sein, dass ihre Daten gut vor potenziellen Risiken und Bedrohungen geschützt sind. NetApp legt größten Wert auf Datensicherheit und bietet seinen Kunden die Gewissheit, dass ihre wertvollen Informationen sicher und intakt sind.

Vektordatenbank

Dieser Abschnitt behandelt die Definition und Verwendung einer Vektordatenbank in NetApp KI-Lösungen.

Vektordatenbank

Eine Vektordatenbank ist ein spezialisierter Datenbanktyp, der für die Verarbeitung, Indizierung und Suche unstrukturierter Daten mithilfe von Einbettungen aus Modellen des maschinellen Lernens konzipiert ist. Anstatt Daten in einem herkömmlichen Tabellenformat zu organisieren, werden sie als hochdimensionale Vektoren angeordnet, die auch als Vektoreinbettungen bezeichnet werden. Diese einzigartige Struktur ermöglicht es der Datenbank, komplexe, mehrdimensionale Daten effizienter und genauer zu verarbeiten.

Eine der wichtigsten Fähigkeiten einer Vektordatenbank ist die Verwendung generativer KI zur Durchführung von Analysen. Hierzu gehören Ähnlichkeitssuchen, bei denen die Datenbank Datenpunkte identifiziert, die einer bestimmten Eingabe ähneln, und die Anomalieerkennung, bei der sie Datenpunkte erkennen kann, die erheblich von der Norm abweichen.

Darüber hinaus eignen sich Vektordatenbanken gut für die Verarbeitung zeitlicher Daten oder mit Zeitstempeln versehener Daten. Diese Art von Daten liefert Informationen darüber, „was“ passiert ist und wann es passiert ist, in der Reihenfolge und im Verhältnis zu allen anderen Ereignissen innerhalb eines bestimmten IT-Systems. Diese Fähigkeit, zeitliche Daten zu verarbeiten und zu analysieren, macht Vektordatenbanken besonders nützlich für Anwendungen, die ein Verständnis von Ereignissen im Zeitverlauf erfordern.

Vorteile der Vektordatenbank für ML und KI:

- Hochdimensionale Suche: Vektordatenbanken eignen sich hervorragend zum Verwalten und Abrufen hochdimensionaler Daten, die häufig in KI- und ML-Anwendungen generiert werden.
- Skalierbarkeit: Sie können effizient skaliert werden, um große Datenmengen zu verarbeiten und so das Wachstum und die Expansion von KI- und ML-Projekten zu unterstützen.
- Flexibilität: Vektordatenbanken bieten ein hohes Maß an Flexibilität und ermöglichen die Aufnahme unterschiedlicher Datentypen und -strukturen.
- Leistung: Sie bieten leistungsstarkes Datenmanagement und -abruf, was für die Geschwindigkeit und Effizienz von KI- und ML-Operationen entscheidend ist.
- Anpassbare Indizierung: Vector-Datenbanken bieten anpassbare Indizierungsoptionen, die eine optimierte Datenorganisation und -abfrage basierend auf spezifischen Anforderungen ermöglichen.

Vektordatenbanken und Anwendungsfälle.

Dieser Abschnitt enthält verschiedene Vektordatenbanken und Einzelheiten zu ihren Anwendungsfällen.

Faiss und ScaNN

Es handelt sich um Bibliotheken, die als wichtige Werkzeuge im Bereich der Vektorsuche dienen. Diese Bibliotheken bieten Funktionen, die für die Verwaltung und Suche in Vektordaten von entscheidender Bedeutung sind, und stellen somit unschätzbare Ressourcen in diesem speziellen Bereich der Datenverwaltung dar.

Elasticsearch

Es handelt sich um eine weit verbreitete Such- und Analyse-Engine, die seit kurzem auch über eine Vektorsuchfunktion verfügt. Diese neue Funktion erweitert die Funktionalität und ermöglicht eine effektivere Verarbeitung und Suche in Vektordaten.

Tannenzapfen

Es handelt sich um eine robuste Vektordatenbank mit einem einzigartigen Funktionsumfang. Es unterstützt in seiner Indexierungsfunktionalität sowohl dichte als auch spärliche Vektoren, was seine Flexibilität und Anpassungsfähigkeit verbessert. Eine seiner Hauptstärken liegt in der Fähigkeit, traditionelle Suchmethoden mit KI-basierter dichter Vektorsuche zu kombinieren und so einen hybriden Suchansatz zu schaffen, der das Beste aus beiden Welten nutzt.

Pinecone ist hauptsächlich cloudbasiert, für Anwendungen des maschinellen Lernens konzipiert und lässt sich gut in eine Vielzahl von Plattformen integrieren, darunter GCP, AWS, Open AI, GPT-3, GPT-3.5, GPT-4, Catgut Plus, Elasticsearch, Haystack und mehr. Es ist wichtig zu beachten, dass Pinecone eine Closed-Source-Plattform ist und als Software-as-a-Service-Angebot (SaaS) verfügbar ist.

Aufgrund seiner erweiterten Funktionen eignet sich Pinecone besonders gut für die Cybersicherheitsbranche, wo seine hochdimensionalen Such- und Hybridsuchfunktionen effektiv genutzt werden können, um Bedrohungen zu erkennen und darauf zu reagieren.

Chroma

Es handelt sich um eine Vektordatenbank mit einer Core-API mit vier Hauptfunktionen, von denen eine einen In-Memory-Dokumentenvektorspeicher umfasst. Es nutzt außerdem die Face Transformers-Bibliothek zum Vektorisieren von Dokumenten und verbessert so seine Funktionalität und Vielseitigkeit. Chroma ist für den Betrieb sowohl in der Cloud als auch vor Ort konzipiert und bietet Flexibilität basierend auf den

Benutzeranforderungen. Es zeichnet sich insbesondere bei Audioanwendungen aus und ist daher eine ausgezeichnete Wahl für audiobasierte Suchmaschinen, Musikempfehlungssysteme und andere audiobezogene Anwendungsfälle.

Weaviate

Es handelt sich um eine vielseitige Vektordatenbank, die es Benutzern ermöglicht, ihre Inhalte entweder mithilfe der integrierten oder benutzerdefinierten Module zu vektorisieren und so Flexibilität basierend auf spezifischen Anforderungen zu bieten. Es bietet sowohl vollständig verwaltete als auch selbst gehostete Lösungen und berücksichtigt dabei eine Vielzahl von Bereitstellungspräferenzen.

Eine der Hauptfunktionen von Weaviate ist die Fähigkeit, sowohl Vektoren als auch Objekte zu speichern, wodurch die Datenverarbeitungsfunktionen verbessert werden. Es wird häufig für eine Reihe von Anwendungen verwendet, darunter semantische Suche und Datenklassifizierung in ERP-Systemen. Im E-Commerce-Sektor treibt es Such- und Empfehlungsmaschinen an. Weaviate wird auch für die Bildsuche, Anomalieerkennung, automatisierte Datenharmonisierung und Cybersicherheitsbedrohungsanalyse verwendet und zeigt damit seine Vielseitigkeit in mehreren Bereichen.

Redis

Redis ist eine leistungsstarke Vektordatenbank, die für ihre schnelle In-Memory-Speicherung bekannt ist und eine geringe Latenz für Lese-/Schreibvorgänge bietet. Dies macht es zu einer ausgezeichneten Wahl für Empfehlungssysteme, Suchmaschinen und Datenanalyseanwendungen, die einen schnellen Datenzugriff erfordern.

Redis unterstützt verschiedene Datenstrukturen für Vektoren, einschließlich Listen, Mengen und sortierte Mengen. Es bietet auch Vektoroperationen wie das Berechnen von Abständen zwischen Vektoren oder das Finden von Schnittpunkten und Vereinigungen. Diese Funktionen sind besonders nützlich für die Ähnlichkeitssuche, Clustering und inhaltsbasierte Empfehlungssysteme.

In Bezug auf Skalierbarkeit und Verfügbarkeit zeichnet sich Redis durch die Verarbeitung von Workloads mit hohem Durchsatz aus und bietet Datenreplikation. Es lässt sich auch gut in andere Datentypen integrieren, einschließlich herkömmlicher relationaler Datenbanken (RDBMS). Redis enthält eine Publish/Subscribe-Funktion (Pub/Sub) für Echtzeit-Updates, die für die Verwaltung von Echtzeit-Vektoren von Vorteil ist. Darüber hinaus ist Redis leichtgewichtig und einfach zu verwenden, was es zu einer benutzerfreundlichen Lösung für die Verwaltung von Vektordaten macht.

Milvus

Es handelt sich um eine vielseitige Vektordatenbank, die eine API wie einen Dokumentenspeicher bietet, ähnlich wie MongoDB. Es zeichnet sich durch die Unterstützung einer Vielzahl von Datentypen aus und ist daher eine beliebte Wahl in den Bereichen Datenwissenschaft und maschinelles Lernen.

Eine der einzigartigen Funktionen von Milvus ist die Multivektorisierungsfunktion, die es Benutzern ermöglicht, zur Laufzeit den für die Suche zu verwendenden Vektortyp anzugeben. Darüber hinaus nutzt es Knowwhere, eine Bibliothek, die auf anderen Bibliotheken wie Faiss aufbaut, um die Kommunikation zwischen Abfragen und den Vektorsuchalgorithmen zu verwalten.

Dank seiner Kompatibilität mit PyTorch und TensorFlow bietet Milvus außerdem eine nahtlose Integration in Machine-Learning-Workflows. Dies macht es zu einem hervorragenden Tool für eine Reihe von Anwendungen, darunter E-Commerce, Bild- und Videoanalyse, Objekterkennung, Bildähnlichkeitssuche und inhaltsbasierte Bildabfrage. Im Bereich der natürlichen Sprachverarbeitung wird Milvus für die Dokumentenclusterung, semantische Suche und Frage-Antwort-Systeme verwendet.

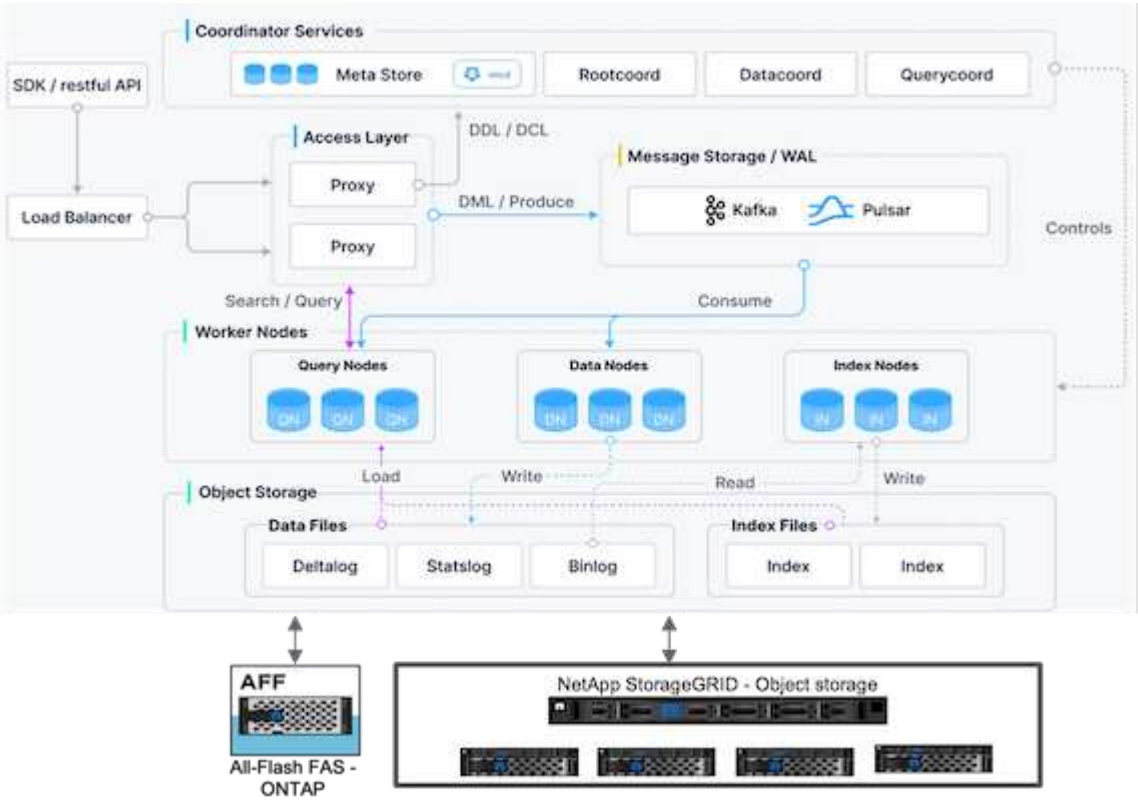
Für diese Lösung haben wir Milvus zur Lösungsvalidierung ausgewählt. Aus Leistungsgründen haben wir

sowohl Milvus als auch Postgres (pgvecto.rs) verwendet.

Warum haben wir uns für diese Lösung für Milvus entschieden?

- Open Source: Milvus ist eine Open-Source-Vektordatenbank, die eine von der Community gesteuerte Entwicklung und Verbesserung fördert.
- KI-Integration: Es nutzt die Einbettung von Ähnlichkeitssuchen und KI-Anwendungen, um die Funktionalität der Vektordatenbank zu verbessern.
- Handhabung großer Datenmengen: Milvus verfügt über die Kapazität, über eine Milliarde Einbettungsvektoren zu speichern, zu indizieren und zu verwalten, die von Deep Neural Networks (DNN) und Machine Learning (ML)-Modellen generiert werden.
- Benutzerfreundlich: Die Verwendung ist einfach, die Einrichtung dauert weniger als eine Minute. Milvus bietet auch SDKs für verschiedene Programmiersprachen an.
- Geschwindigkeit: Es bietet blitzschnelle Abrufgeschwindigkeiten, bis zu 10-mal schneller als einige Alternativen.
- Skalierbarkeit und Verfügbarkeit: Milvus ist hochgradig skalierbar und bietet Optionen zur Skalierung nach oben und unten nach Bedarf.
- Funktionsreich: Es unterstützt verschiedene Datentypen, Attributfilterung, benutzerdefinierte Funktionen (UDF), konfigurierbare Konsistenzstufen und Reisezeiten und ist somit ein vielseitiges Tool für verschiedene Anwendungen.

Milvus-Architekturübersicht



Dieser Abschnitt stellt Komponenten und Dienste höherer Ebene bereit, die in der Milvus-Architektur verwendet werden. * Zugriffsebene – Sie besteht aus einer Gruppe zustandsloser Proxys und dient als Frontebene des Systems und Endpunkt für Benutzer. * Koordinatordienst – er weist die Aufgaben den Arbeitsknoten zu und fungiert als Gehirn des Systems. Es gibt drei Koordinatortypen: Stammkoordinate, Datenkoordinate und Abfragekoordinate. * Worker-Knoten: Es folgt den Anweisungen des Koordinatordienstes

und führt vom Benutzer ausgelöste DML/DDL-Befehle aus. Es gibt drei Arten von Worker-Knoten, nämlich Abfrageknoten, Datenknoten und Indexknoten. * Speicher: Er ist für die Datenpersistenz verantwortlich. Es umfasst Metaspeicher, Log Broker und Objektspeicher. NetApp Speicher wie ONTAP und StorageGRID bieten Milvus Objektspeicher und dateibasierten Speicher sowohl für Kundendaten als auch für Vektordatenbankdaten.

Technologieanforderungen

Dieser Abschnitt bietet einen Überblick über die Anforderungen für die NetApp Vector-Datenbanklösung.

Technologieanforderungen

Für die Mehrzahl der in diesem Dokument durchgeführten Validierungen wurden die unten beschriebenen Hardware- und Softwarekonfigurationen verwendet, mit Ausnahme der Leistung. Diese Konfigurationen dienen als Richtlinie und helfen Ihnen beim Einrichten Ihrer Umgebung. Bitte beachten Sie jedoch, dass die konkreten Komponenten je nach individuellen Kundenanforderungen variieren können.

Hardwareanforderungen

Hardware	Details
NetApp AFF Storage-Array HA-Paar	* A800 * ONTAP 9.14.1 * 48 x 3,49 TB SSD-NVM * Zwei flexible Gruppenvolumes: Metadaten und Daten. * Das Metadaten-NFS-Volume verfügt über 12 persistente Volumes mit 250 GB. * Daten sind ein ONTAP NAS S3-Volume
6 x FUJITSU PRIMERGY RX2540 M4	* 64 CPUs * Intel® Xeon® Gold 6142 CPU @ 2,60 GHz * 256 GM physischer Speicher * 1 x 100GbE Netzwerkanschluss
Vernetzung	100 GbE
StorageGRID	* 1 x SG100, 3 x SGF6024 * 3 x 24 x 7,68 TB

Softwareanforderungen

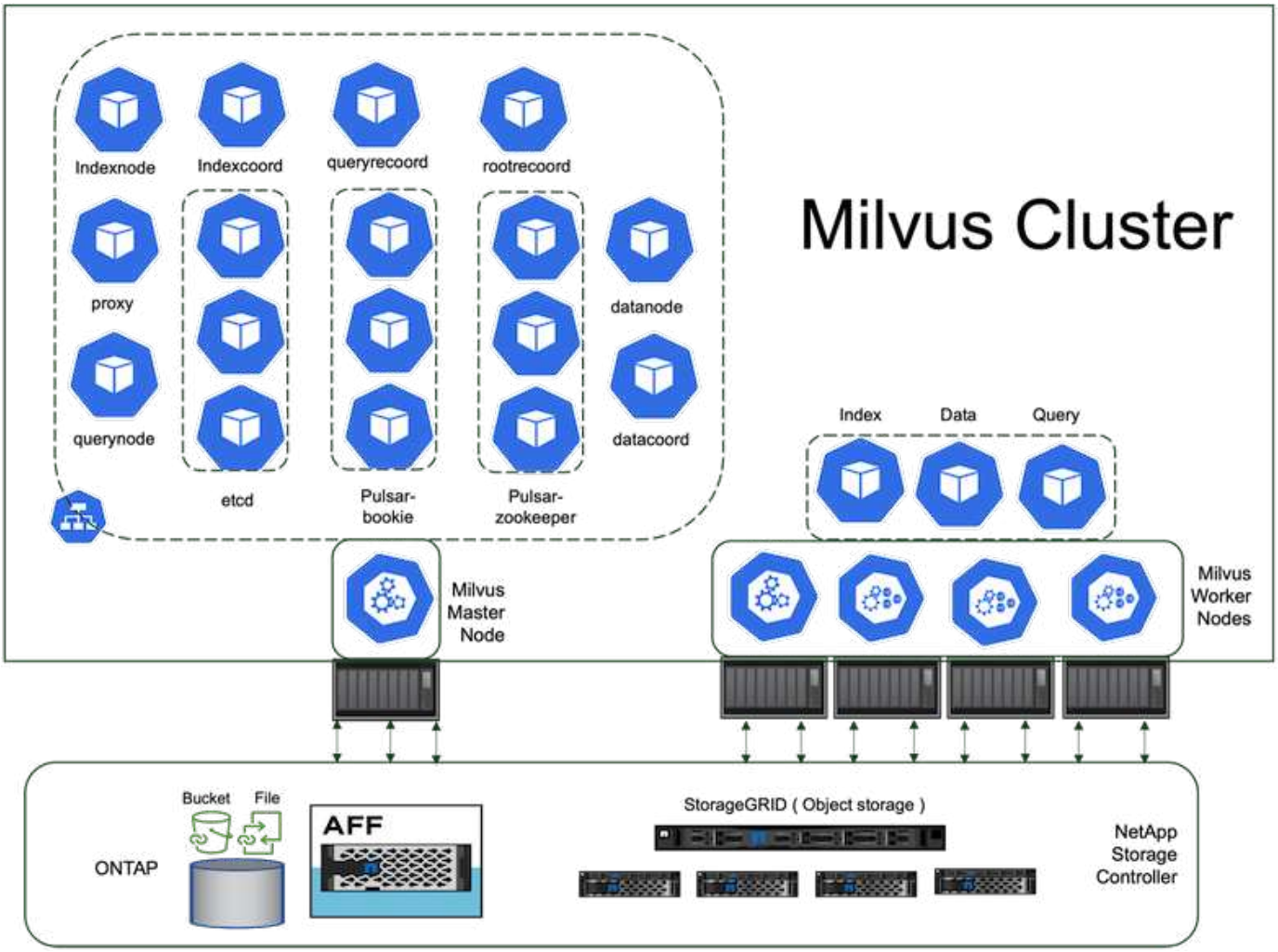
Software	Details
Milvus-Cluster	* DIAGRAMM - milvus-4.1.11. * APP-Version – 2.3.4 * Abhängige Pakete wie Bookkeeper, Zookeeper, Pulsar, etcd, Proxy, Querynode, Worker
Kubernetes	* 5-Knoten-K8s-Cluster * 1 Masterknoten und 4 Workerknoten * Version – 1.7.2
Python	*3.10.12.

Bereitstellungsverfahren

In diesem Abschnitt wird das Bereitstellungsverfahren für die Vektordatenbanklösung für NetApp erläutert.

Bereitstellungsverfahren

In diesem Bereitstellungsabschnitt haben wir die Milvus-Vektordatenbank mit Kubernetes für die Laboreinrichtung wie unten beschrieben verwendet.



Der NetApp-Speicher stellt den Speicherplatz für den Cluster bereit, um Kundendaten und Milvus-Clusterdaten aufzubewahren.

NetApp -Speichereinrichtung – ONTAP

- Initialisierung des Speichersystems
- Erstellen einer virtuellen Speichermaschine (SVM)
- Zuordnung logischer Netzwerkschnittstellen
- NFS, S3-Konfiguration und -Lizenzierung

Bitte befolgen Sie die folgenden Schritte für NFS (Network File System):

1. Erstellen Sie ein FlexGroup -Volume für NFSv4. In unserem Setup für diese Validierung haben wir 48 SSDs verwendet, 1 SSD speziell für das Root-Volume des Controllers und 47 SSDs verteilt für NFSv4]].Überprüfen Sie, ob die NFS-Exportrichtlinie für das FlexGroup -Volume Lese-/Schreibberechtigungen für das Kubernetes-Knotennetzwerk (K8s) hat. Wenn diese Berechtigungen nicht vorhanden sind, erteilen Sie Lese-/Schreibberechtigungen (rw) für das K8s-Knotennetzwerk.

2. Erstellen Sie auf allen K8s-Knoten einen Ordner und mounten Sie das FlexGroup Volume über eine logische Schnittstelle (LIF) auf jedem K8s-Knoten in diesen Ordner.

Bitte befolgen Sie die folgenden Schritte für NAS S3 (Network Attached Storage Simple Storage Service):

1. Erstellen Sie ein FlexGroup -Volume für NFS.
2. Richten Sie mit dem Befehl „vserver object-store-server create“ einen Object-Store-Server mit aktiviertem HTTP und dem Administratorstatus „up“ ein. Sie haben die Möglichkeit, HTTPS zu aktivieren und einen benutzerdefinierten Listener-Port festzulegen.
3. Erstellen Sie einen Object-Store-Server-Benutzer mit dem Befehl „vserver object-store-server user create -user <Benutzername>“.
4. Um den Zugriffsschlüssel und den geheimen Schlüssel zu erhalten, können Sie den folgenden Befehl ausführen: „set diag; vserver object-store-server user show -user <Benutzername>“. In Zukunft werden diese Schlüssel jedoch während des Benutzererstellungsprozesses bereitgestellt oder können mithilfe von REST-API-Aufrufen abgerufen werden.
5. Richten Sie mit dem in Schritt 2 erstellten Benutzer eine Object-Store-Server-Gruppe ein und gewähren Sie Zugriff. In diesem Beispiel haben wir „FullAccess“ bereitgestellt.
6. Erstellen Sie einen NAS-Bucket, indem Sie seinen Typ auf „nas“ festlegen und den Pfad zum NFSv3-Volume angeben. Es ist auch möglich, zu diesem Zweck einen S3-Bucket zu verwenden.

NetApp -Speichereinrichtung – StorageGRID

1. Installieren Sie die storageGRID-Software.
2. Erstellen Sie einen Mandanten und einen Bucket.
3. Erstellen Sie einen Benutzer mit der erforderlichen Berechtigung.

Weitere Einzelheiten finden Sie in https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

Lösungsüberprüfung

Lösungsübersicht

Wir haben eine umfassende Lösungsvalidierung mit Schwerpunkt auf fünf Schlüsselbereichen durchgeführt, deren Einzelheiten im Folgenden aufgeführt sind. Jeder Abschnitt befasst sich eingehend mit den Herausforderungen, vor denen die Kunden stehen, den von NetApp bereitgestellten Lösungen und den daraus resultierenden Vorteilen für den Kunden.

1. "Milvus-Cluster-Setup mit Kubernetes vor Ort"Kunden stehen vor der Herausforderung, Speicher und Rechenleistung unabhängig zu skalieren und die Infrastruktur und Daten effektiv zu verwalten. In diesem Abschnitt beschreiben wir detailliert den Prozess der Installation eines Milvus-Clusters auf Kubernetes unter Verwendung eines NetApp -Speichercontrollers sowohl für Clusterdaten als auch für Kundendaten.
2. Milvus mit Amazon FSx ONTAP für NetApp ONTAP – Datei- und Objektdualität In diesem Abschnitt erfahren Sie, warum wir die Vektordatenbank in der Cloud bereitstellen müssen, sowie die Schritte zur Bereitstellung der Vektordatenbank (Milvus Standalone) in Amazon FSx ONTAP für NetApp ONTAP innerhalb von Docker-Containern.
3. "Vector-Datenbankschutz mit NetApp SnapCenter."In diesem Abschnitt gehen wir näher darauf ein, wie SnapCenter die in ONTAP gespeicherten Vektordatenbankdaten und Milvus-Daten schützt. Für dieses

Beispiel haben wir einen NAS-Bucket (milvusdbvol1) verwendet, der aus einem NFS- ONTAP Volume (vol1) für Kundendaten abgeleitet wurde, und ein separates NFS-Volume (vectordbpv) für Milvus-Cluster-Konfigurationsdaten.

4. "Disaster Recovery mit NetApp SnapMirror"In diesem Abschnitt besprechen wir die Bedeutung der Notfallwiederherstellung (DR) für die Vektordatenbank und wie das NetApp-Notfallwiederherstellungsprodukt SnapMirror eine DR-Lösung für die Vektordatenbank bereitstellt.
5. "Leistungsvalidierung"In diesem Abschnitt möchten wir uns eingehend mit der Leistungsvalidierung von Vektordatenbanken wie Milvus und pgvecto.rs befassen und uns dabei auf ihre Speicherleistungsmerkmale wie E/A-Profil und Verhalten des NetApp-Speichercontrollers zur Unterstützung von RAG- und Inferenz-Workloads innerhalb des LLM-Lebenszyklus konzentrieren. Wir werden alle Leistungsunterschiede bewerten und identifizieren, wenn diese Datenbanken mit der ONTAP Speicherlösung kombiniert werden. Unsere Analyse basiert auf wichtigen Leistungsindikatoren, beispielsweise der Anzahl der pro Sekunde verarbeiteten Abfragen (QPS).

Milvus-Cluster-Setup mit Kubernetes vor Ort

In diesem Abschnitt wird die Einrichtung des Milvus-Clusters für die Vector-Datenbanklösung für NetApp erläutert.

Milvus-Cluster-Setup mit Kubernetes vor Ort

Kunden stehen vor der Herausforderung, Speicher und Rechenleistung unabhängig zu skalieren und eine effektive Infrastruktur- und Datenverwaltung zu gewährleisten. Kubernetes und Vektordatenbanken bilden zusammen eine leistungsstarke, skalierbare Lösung für die Verwaltung großer Datenvorgänge. Kubernetes optimiert Ressourcen und verwaltet Container, während Vektordatenbanken hochdimensionale Daten und Ähnlichkeitssuchen effizient verarbeiten. Diese Kombination ermöglicht die schnelle Verarbeitung komplexer Abfragen großer Datensätze und lässt sich nahtlos mit wachsenden Datenmengen skalieren, was sie ideal für Big-Data-Anwendungen und KI-Workloads macht.

1. In diesem Abschnitt beschreiben wir detailliert den Prozess der Installation eines Milvus-Clusters auf Kubernetes unter Verwendung eines NetApp -Speichercontrollers sowohl für Clusterdaten als auch für Kundendaten.
2. Zur Installation eines Milvus-Clusters sind Persistent Volumes (PVs) zum Speichern von Daten aus verschiedenen Milvus-Clusterkomponenten erforderlich. Zu diesen Komponenten gehören etcd (drei Instanzen), pulsar-bookie-journal (drei Instanzen), pulsar-bookie-ledgers (drei Instanzen) und pulsar-zookeeper-data (drei Instanzen).

> Im Milvus-Cluster können wir entweder Pulsar oder Kafka als zugrunde liegende Engine verwenden, die die zuverlässige Speicherung und Veröffentlichung/Abonnementierung von Nachrichtenströmen im Milvus-Cluster unterstützt. Für Kafka mit NFS hat NetApp Verbesserungen in ONTAP 9.12.1 und höher vorgenommen. Diese Verbesserungen sowie NFSv4.1- und Linux-Änderungen, die in RHEL 8.7 oder 9.1 und höher enthalten sind, beheben das Problem der „dummen Umbenennung“, das beim Ausführen von Kafka über NFS auftreten kann. Wenn Sie an ausführlicheren Informationen zum Ausführen von Kafka mit der NetApp NFS-Lösung interessiert sind, lesen Sie bitte -"dieser Link" .

3. Wir haben ein einzelnes NFS-Volume von NetApp ONTAP erstellt und 12 persistente Volumes mit jeweils 250 GB Speicher eingerichtet. Die Speichergröße kann je nach Clustergröße variieren. Beispielsweise haben wir einen anderen Cluster, bei dem jedes PV über 50 GB verfügt. Weitere Einzelheiten finden Sie unten in einer der PV-YAML-Dateien. Insgesamt hatten wir 12 solcher Dateien. In jeder Datei ist der storageClassName auf „default“ gesetzt und Speicher und Pfad sind für jedes PV eindeutig.

```
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. Führen Sie den Befehl „kubectl apply“ für jede PV-YAML-Datei aus, um die persistenten Volumes zu erstellen, und überprüfen Sie anschließend deren Erstellung mit „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. Zum Speichern von Kundendaten unterstützt Milvus Objektspeicherlösungen wie MinIO, Azure Blob und S3. In dieser Anleitung verwenden wir S3. Die folgenden Schritte gelten sowohl für den ONTAP S3- als auch für den StorageGRID Objektspeicher. Wir verwenden Helm, um den Milvus-Cluster bereitzustellen. Laden Sie die Konfigurationsdatei values.yaml vom Milvus-Download-Speicherort herunter. Die Datei values.yaml, die wir in diesem Dokument verwendet haben, finden Sie im Anhang.
6. Stellen Sie sicher, dass die „storageClass“ in jedem Abschnitt auf „default“ gesetzt ist, einschließlich der Abschnitte für Protokoll, etcd, Zookeeper und Bookkeeper.
7. Deaktivieren Sie MinIO im Abschnitt MinIO.
8. Erstellen Sie einen NAS-Bucket aus dem ONTAP oder StorageGRID Objektspeicher und fügen Sie ihn mit den Objektspeicher-Anmeldeinformationen in ein externes S3 ein.

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

9. Stellen Sie vor dem Erstellen des Milvus-Clusters sicher, dass der PersistentVolumeClaim (PVC) keine bereits vorhandenen Ressourcen enthält.

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

10. Verwenden Sie Helm und die Konfigurationsdatei values.yaml, um den Milvus-Cluster zu installieren und zu starten.

```
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. Überprüfen Sie den Status der PersistentVolumeClaims (PVCs).

```
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. Überprüfen Sie den Status der Pods.

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

Bitte stellen Sie sicher, dass der Pod-Status „läuft“ lautet und wie erwartet funktioniert.

13. Testen Sie das Schreiben und Lesen von Daten im Milvus- und NetApp Objektspeicher.
    - Schreiben Sie Daten mit dem Python-Programm „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:~#
```

- Lesen Sie die Daten mithilfe der Python-Datei „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}}]}
```

Basierend auf der obigen Validierung bietet die Integration von Kubernetes mit einer Vektordatenbank, wie durch die Bereitstellung eines Milvus-Clusters auf Kubernetes unter Verwendung eines NetApp -Speichercontrollers demonstriert, Kunden eine robuste, skalierbare und effiziente Lösung für die Verwaltung umfangreicher Datenvorgänge. Dieses Setup bietet Kunden die Möglichkeit, hochdimensionale Daten zu verarbeiten und komplexe Abfragen schnell und effizient auszuführen, was es zu einer idealen Lösung für Big-Data-Anwendungen und KI-Workloads macht. Die Verwendung von Persistent Volumes (PVs) für verschiedene Clusterkomponenten sowie die Erstellung eines einzelnen NFS-Volumes aus NetApp ONTAP gewährleisten eine optimale Ressourcennutzung und

Datenverwaltung. Der Prozess der Überprüfung des Status von PersistentVolumeClaims (PVCs) und Pods sowie das Testen des Schreibens und Lesens von Daten bietet Kunden die Gewissheit zuverlässiger und konsistenter Datenvorgänge. Die Verwendung von ONTAP oder StorageGRID -Objektspeicher für Kundendaten verbessert die Datenzugänglichkeit und -sicherheit zusätzlich. Insgesamt bietet diese Konfiguration den Kunden eine robuste und leistungsstarke Datenverwaltungslösung, die sich nahtlos an ihren wachsenden Datenbedarf anpassen lässt.

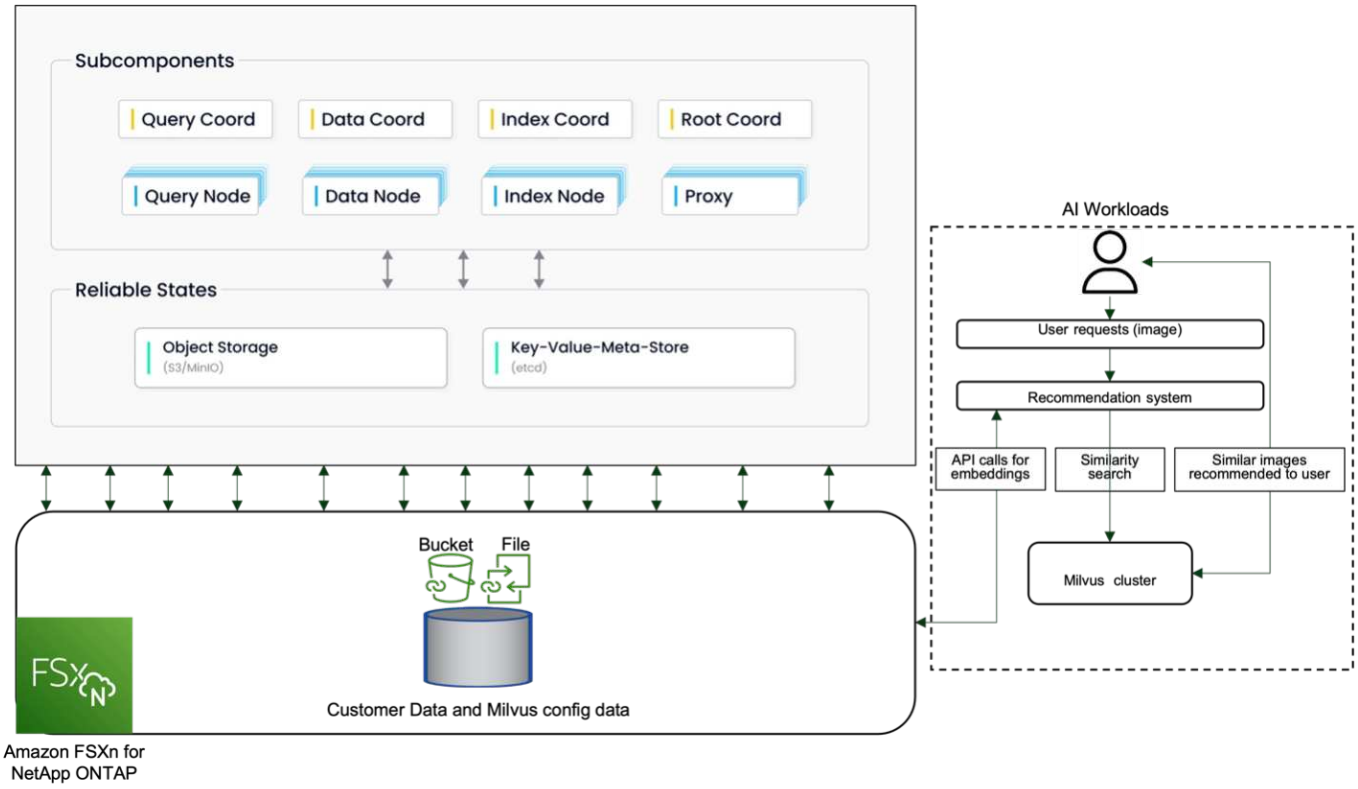
Milvus mit Amazon FSx ONTAP für NetApp ONTAP – Datei- und Objektdualität

In diesem Abschnitt wird die Einrichtung des Milvus-Clusters mit Amazon FSx ONTAP für die Vektordatenbanklösung für NetApp erläutert.

Milvus mit Amazon FSx ONTAP für NetApp ONTAP – Datei- und Objektdualität

In diesem Abschnitt erfahren Sie, warum wir die Vektordatenbank in der Cloud bereitstellen müssen, sowie die Schritte zum Bereitstellen der Vektordatenbank (Milvus Standalone) in Amazon FSx ONTAP für NetApp ONTAP innerhalb von Docker-Containern.

Die Bereitstellung einer Vektordatenbank in der Cloud bietet mehrere bedeutende Vorteile, insbesondere für Anwendungen, bei denen hochdimensionale Daten verarbeitet und Ähnlichkeitssuchen ausgeführt werden müssen. Erstens bietet die Cloud-basierte Bereitstellung Skalierbarkeit und ermöglicht eine einfache Anpassung der Ressourcen an die wachsenden Datenmengen und Abfragelasten. Dadurch wird sichergestellt, dass die Datenbank die erhöhte Nachfrage effizient bewältigen und gleichzeitig eine hohe Leistung aufrechterhalten kann. Zweitens bietet die Cloud-Bereitstellung hohe Verfügbarkeit und Notfallwiederherstellung, da Daten über verschiedene geografische Standorte hinweg repliziert werden können, wodurch das Risiko eines Datenverlusts minimiert und ein kontinuierlicher Dienst auch bei unerwarteten Ereignissen sichergestellt wird. Drittens ist es kosteneffizient, da Sie nur für die Ressourcen zahlen, die Sie tatsächlich nutzen, und je nach Bedarf hoch- oder herunterskalieren können, sodass keine erheblichen Vorabinvestitionen in Hardware erforderlich sind. Und schließlich kann die Bereitstellung einer Vektordatenbank in der Cloud die Zusammenarbeit verbessern, da von überall auf die Daten zugegriffen und diese geteilt werden können, was die teambasierte Arbeit und datengesteuerte Entscheidungsfindung erleichtert. Bitte überprüfen Sie die Architektur des Milvus-Standalone mit Amazon FSx ONTAP für NetApp ONTAP , das bei dieser Validierung verwendet wird.



1. Erstellen Sie eine Amazon FSx ONTAP für NetApp ONTAP Instanz und notieren Sie die Details der VPC, der VPC-Sicherheitsgruppen und des Subnetzes. Diese Informationen werden beim Erstellen einer EC2-Instanz benötigt. Weitere Details finden Sie hier - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. Erstellen Sie eine EC2-Instance und stellen Sie sicher, dass VPC, Sicherheitsgruppen und Subnetz mit denen der Amazon FSx ONTAP für NetApp ONTAP -Instance übereinstimmen.
3. Installieren Sie nfs-common mit dem Befehl „apt-get install nfs-common“ und aktualisieren Sie die Paketinformationen mit „sudo apt-get update“.
4. Erstellen Sie einen Mount-Ordner und mounten Sie Amazon FSx ONTAP für NetApp ONTAP darin.

```
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. Installieren Sie Docker und Docker Compose mit „apt-get install“.
6. Richten Sie einen Milvus-Cluster basierend auf der Datei docker-compose.yaml ein, die von der Milvus-Website heruntergeladen werden kann.

```
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. Ordnen Sie im Abschnitt „Volumes“ der Datei docker-compose.yml den NetApp NFS-Mount-Punkt dem entsprechenden Milvus-Containerpfad zu, insbesondere in etcd, minio und standalone.Check"Anhang D: docker-compose.yml" für Details zu Änderungen in YML
8. Überprüfen Sie die bereitgestellten Ordner und Dateien.

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

9. Führen Sie „docker-compose up -d“ aus dem Verzeichnis aus, das die Datei docker-compose.yml enthält.
10. Überprüfen Sie den Status des Milvus-Containers.

```
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. Um die Lese- und Schreibfunktionalität der Vektordatenbank und ihrer Daten in Amazon FSx ONTAP für NetApp ONTAP zu validieren, haben wir das Python Milvus SDK und ein Beispielprogramm von PyMilvus verwendet. Installieren Sie die erforderlichen Pakete mit „apt-get install python3-numpy python3-pip“ und installieren Sie PyMilvus mit „pip3 install pymilvus“.
12. Validieren Sie Datenschreib- und -lesevorgänge von Amazon FSx ONTAP für NetApp ONTAP in der Vektordatenbank.

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

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

13. Überprüfen Sie den Lesevorgang mit dem Skript 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. Wenn der Kunde für KI-Workloads über das S3-Protokoll auf in der Vektordatenbank getestete NFS-Daten zugreifen (lesen) möchte, kann dies mit einem einfachen Python-Programm validiert werden. Ein Beispiel hierfür könnte eine Ähnlichkeitssuche von Bildern aus einer anderen Anwendung sein, wie im Bild am Anfang dieses Abschnitts erwähnt.

```
root@ip-172-31-29-98:~/pymilvus/examples# sudo python3
/home/ubuntu/milvusvectordb/s3_access.py -i 172.31.255.228 --bucket
milvusnasvol --access-key PY6UF318996I86NBYNDD --secret-key
hoPctr9aD88c1j0SkIYZ2uPa03vlbqKA0c5feK6F
OBJECTS in the bucket milvusnasvol are :
**************************************
…
<output content removed to save page space>
…
bucket/files/insert_log/448789845791611912/448789845791611913/4487898457
91611920/0/448789845791411917/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/1/448789845791411918/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411913/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/101/448789845791411914/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/102/448789845791411915/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/1c48ab6e-
1546-4503-9084-28c629216c33/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
```

```
/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/101/448789845791411921/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/102/448789845791411922/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-
c117-4fba-8256-96cb7557cd6c/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/448789845791411912/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411919/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411926/xl.meta
************************************
root@ip-172-31-29-98:~/pymilvus/examples#
```

Dieser Abschnitt zeigt effektiv, wie Kunden ein eigenständiges Milvus-Setup in Docker-Containern bereitstellen und betreiben können, indem sie Amazons NetApp FSx ONTAP für die NetApp ONTAP Datenspeicherung nutzen. Mit diesem Setup können Kunden die Leistungsfähigkeit von Vektordatenbanken für die Verarbeitung hochdimensionaler Daten und die Ausführung komplexer Abfragen nutzen – und das alles in der skalierbaren und effizienten Umgebung von Docker-Containern. Durch die Erstellung einer Amazon FSx ONTAP für NetApp ONTAP -Instanz und einer passenden EC2-Instanz können Kunden eine optimale Ressourcennutzung und Datenverwaltung sicherstellen. Die erfolgreiche Validierung von Datenschreib- und -lesevorgängen von FSx ONTAP in der Vektordatenbank bietet Kunden die Gewissheit zuverlässiger und konsistenter Datenvorgänge. Darüber hinaus bietet die Möglichkeit, Daten von KI-Workloads über das S3-Protokoll aufzulisten (lesen), eine verbesserte Datenzugänglichkeit. Dieser umfassende Prozess bietet Kunden daher eine robuste und effiziente Lösung für die Verwaltung ihrer groß angelegten Datenvorgänge und nutzt dabei die Funktionen von Amazons FSx ONTAP für NetApp ONTAP.

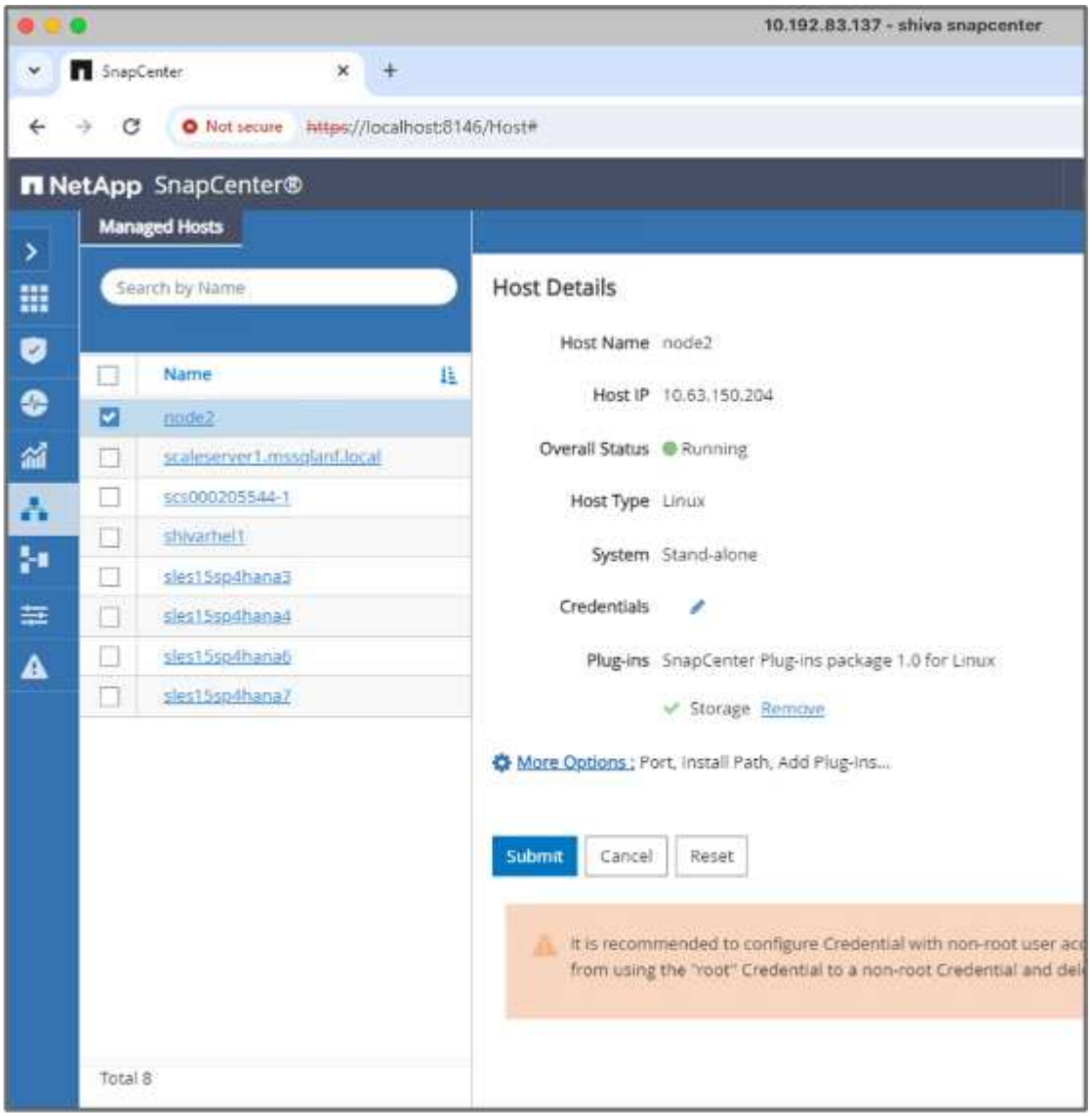
Vector-Datenbankschutz mit SnapCenter

In diesem Abschnitt wird beschrieben, wie Sie mit NetApp SnapCenter Datenschutz für die Vektordatenbank bereitstellen.

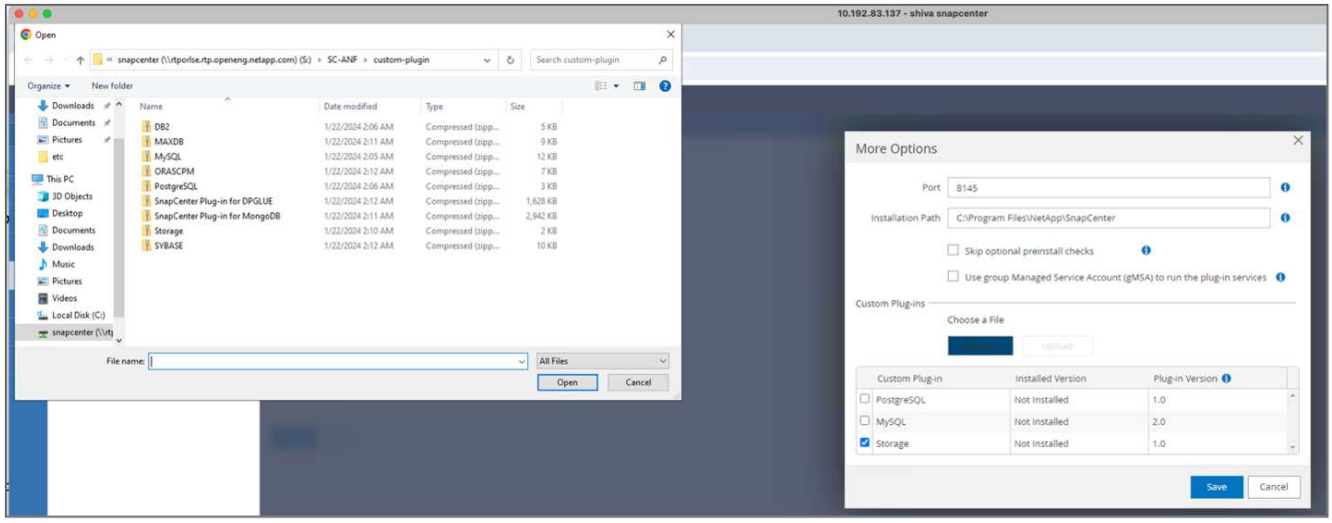
Vector-Datenbankschutz mit NetApp SnapCenter.

In der Filmproduktionsbranche beispielsweise verfügen Kunden häufig über kritische eingebettete Daten wie Video- und Audiodateien. Der Verlust dieser Daten aufgrund von Problemen wie Festplattenausfällen kann erhebliche Auswirkungen auf den Betrieb haben und möglicherweise Multimillionen-Dollar-Projekte gefährden. Wir sind auf Fälle gestoßen, in denen wertvolle Inhalte verloren gingen, was zu erheblichen Störungen und finanziellen Verlusten führte. Die Gewährleistung der Sicherheit und Integrität dieser wichtigen Daten ist daher in dieser Branche von größter Bedeutung. In diesem Abschnitt gehen wir näher darauf ein, wie SnapCenter die in ONTAP gespeicherten Vektordatenbankdaten und Milvus-Daten schützt. Für dieses Beispiel haben wir einen NAS-Bucket (milvusdbvol1) verwendet, der von einem NFS ONTAP Volume (vol1) für Kundendaten abgeleitet wurde, und ein separates NFS-Volume (vectordbpv) für Milvus-Cluster-Konfigurationsdaten. Bitte überprüfen Sie die"hier," für den Snapcenter-Backup-Workflow

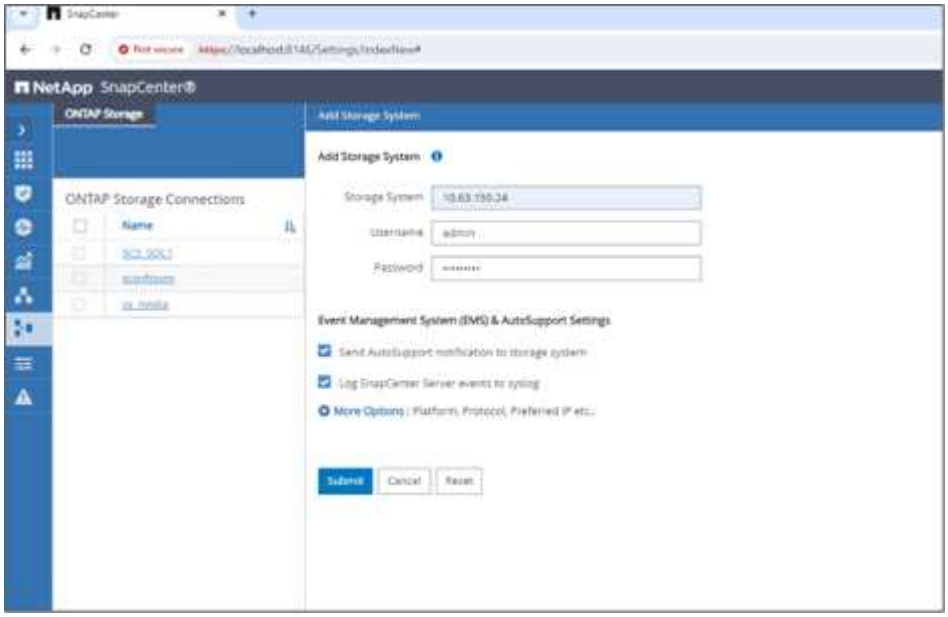
1. Richten Sie den Host ein, der zum Ausführen von SnapCenter -Befehlen verwendet wird.



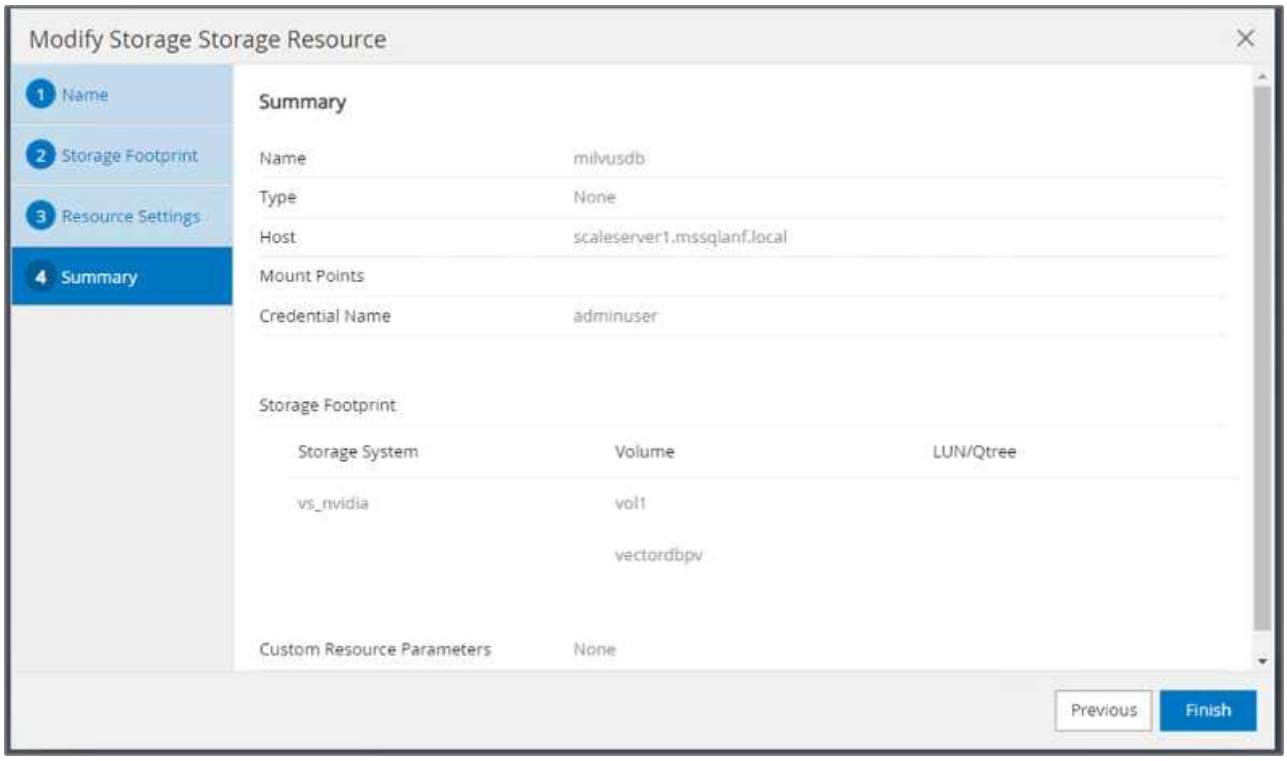
2. Installieren und konfigurieren Sie das Speicher-Plugin. Wählen Sie beim hinzugefügten Host „Weitere Optionen“ aus. Navigieren Sie zum heruntergeladenen Speicher-Plugin und wählen Sie es aus dem"NetApp Automation Store" . Installieren Sie das Plugin und speichern Sie die Konfiguration.



3. Richten Sie das Speichersystem und das Volume ein: Fügen Sie unter „Speichersystem“ das Speichersystem hinzu und wählen Sie die SVM (Storage Virtual Machine) aus. In diesem Beispiel haben wir „vs_nvidia“ gewählt.



4. Richten Sie eine Ressource für die Vektordatenbank ein, die eine Sicherungsrichtlinie und einen benutzerdefinierten Snapshot-Namen enthält.
   - Aktivieren Sie die Konsistenzgruppensicherung mit Standardwerten und aktivieren Sie SnapCenter ohne Dateisystemkonsistenz.
   - Wählen Sie im Abschnitt „Speicherbedarf“ die Volumes aus, die mit den Kundendaten der Vektordatenbank und den Milvus-Clusterdaten verknüpft sind. In unserem Beispiel sind dies „vol1“ und „vectordbpv“.
   - Erstellen Sie eine Richtlinie zum Schutz der Vektordatenbank und schützen Sie die Vektordatenbankressource mithilfe der Richtlinie.



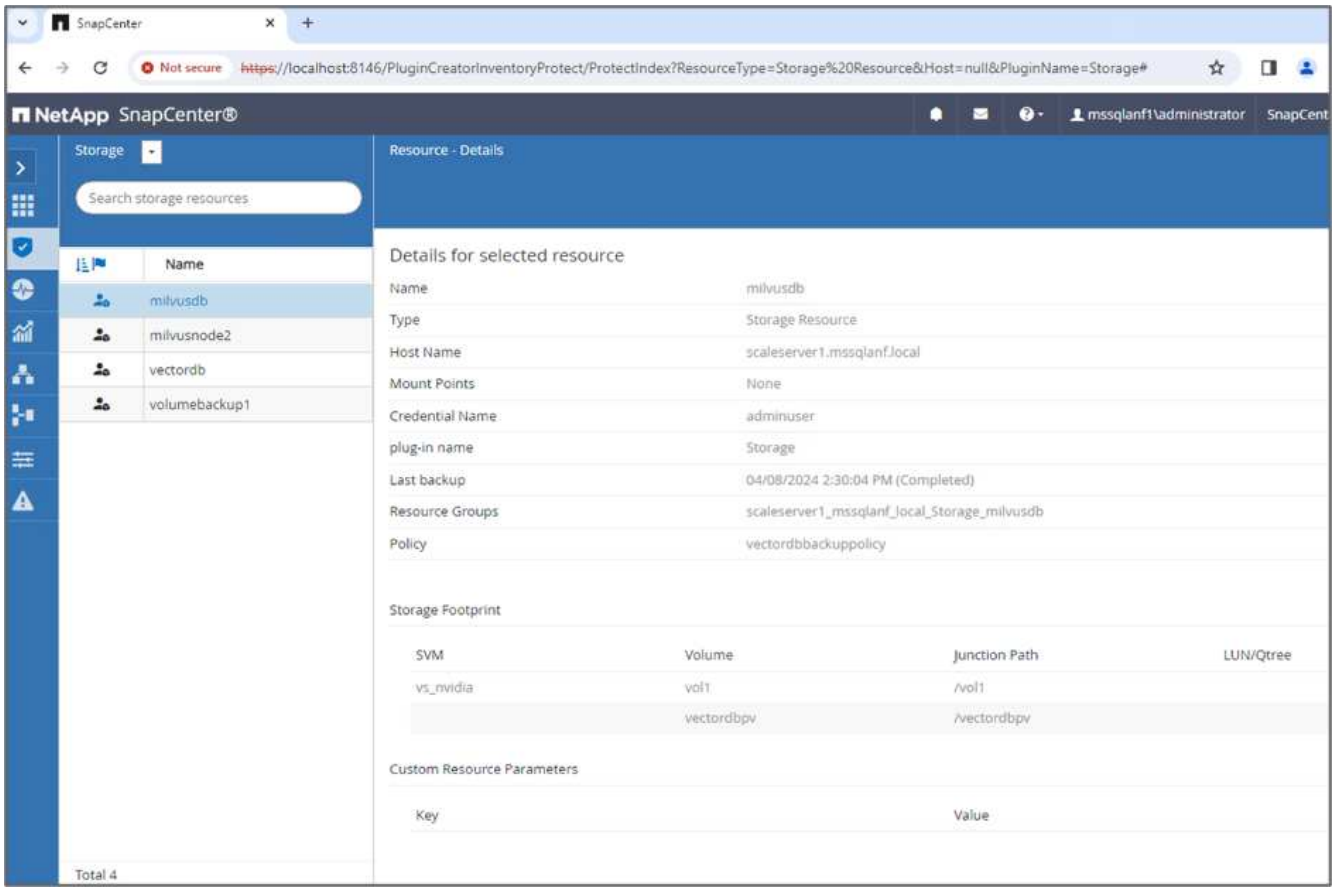
5. Fügen Sie mithilfe eines Python-Skripts Daten in den S3 NAS-Bucket ein. In unserem Fall haben wir das von Milvus bereitgestellte Sicherungsskript „prepare_data_netapp.py“ geändert und den Befehl „sync“ ausgeführt, um die Daten aus dem Betriebssystem zu löschen.

```
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. Überprüfen Sie die Daten im S3 NAS-Bucket. In unserem Beispiel wurden die Dateien mit dem Zeitstempel „2024-04-08 21:22“ vom Skript „prepare_data_netapp.py“ erstellt.

```
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. Starten Sie eine Sicherung mithilfe des Consistency Group (CG)-Snapshots aus der Ressource „milvusdb“.

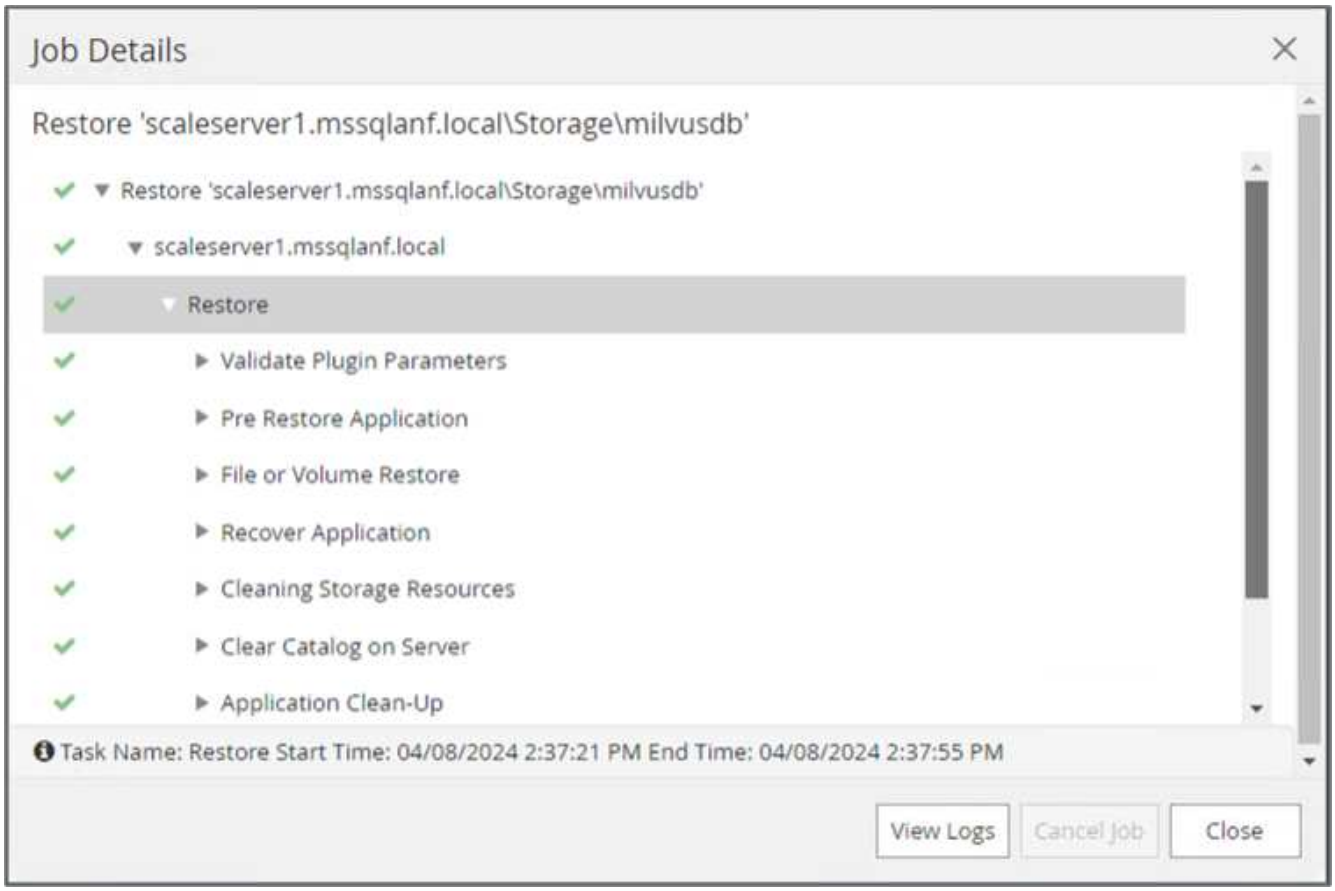


8. Um die Sicherungsfunktionalität zu testen, haben wir nach dem Sicherungsvorgang entweder eine neue Tabelle hinzugefügt oder einige Daten aus dem NFS (S3 NAS-Bucket) entfernt.

   Stellen Sie sich für diesen Test ein Szenario vor, in dem jemand nach der Sicherung eine neue, unnötige oder unangemessene Sammlung erstellt hat. In einem solchen Fall müssten wir die Vektordatenbank auf den Zustand vor dem Hinzufügen der neuen Sammlung zurücksetzen. Beispielsweise wurden neue Sammlungen wie „hello_milvus_netapp_sc_testnew“ und „hello_milvus_netapp_sc_testnew2“ eingefügt.

```
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. Führen Sie eine vollständige Wiederherstellung des S3 NAS-Buckets aus dem vorherigen Snapshot durch.



10. Verwenden Sie ein Python-Skript, um die Daten aus den Sammlungen „hello_milvus_netapp_sc_test“ und „hello_milvus_netapp_sc_test2“ zu überprüfen.

```
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. Stellen Sie sicher, dass die unnötige oder unangemessene Sammlung nicht mehr in der Datenbank vorhanden ist.

```
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:~#
```

Zusammenfassend lässt sich sagen, dass die Verwendung von NetApp SnapCenter zum Schutz von Vektordatenbankdaten und Milvus-Daten in ONTAP den Kunden erhebliche Vorteile bietet, insbesondere in Branchen, in denen die Datenintegrität von größter Bedeutung ist, wie beispielsweise in der Filmproduktion. Die Fähigkeit von SnapCenter, konsistente Backups zu erstellen und vollständige Datenwiederherstellungen durchzuführen, stellt sicher, dass kritische Daten wie eingebettete Video- und Audiodateien vor Verlust durch Festplattenausfälle oder andere Probleme geschützt sind. Dadurch werden nicht nur Betriebsstörungen vermieden, sondern auch erhebliche finanzielle Verluste vermieden.

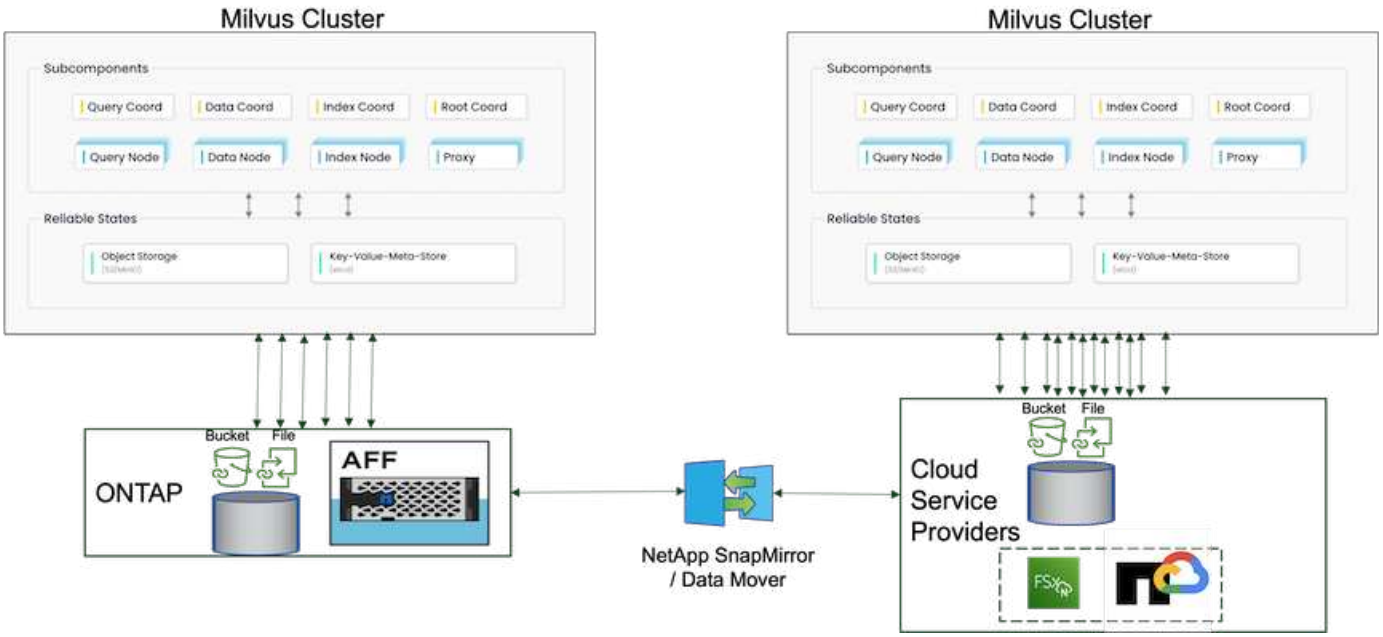
In diesem Abschnitt haben wir gezeigt, wie SnapCenter zum Schutz von in ONTAP gespeicherten Daten konfiguriert werden kann, einschließlich der Einrichtung von Hosts, der Installation und Konfiguration von Speicher-Plugins und der Erstellung einer Ressource für die Vektordatenbank mit einem benutzerdefinierten Snapshot-Namen. Wir haben auch gezeigt, wie man mithilfe des Consistency Group-Snapshots ein Backup durchführt und die Daten im S3 NAS-Bucket überprüft.

Darüber hinaus haben wir ein Szenario simuliert, in dem nach der Sicherung eine unnötige oder unangemessene Sammlung erstellt wurde. In solchen Fällen stellt die Fähigkeit von SnapCenter, eine vollständige Wiederherstellung von einem vorherigen Snapshot durchzuführen, sicher, dass die Vektordatenbank in den Zustand vor dem Hinzufügen der neuen Sammlung zurückgesetzt werden kann, wodurch die Integrität der Datenbank gewahrt bleibt. Diese Möglichkeit, Daten zu einem bestimmten Zeitpunkt wiederherzustellen, ist für Kunden von unschätzbarem Wert, da sie ihnen die Gewissheit gibt, dass ihre Daten nicht nur sicher, sondern auch ordnungsgemäß verwaltet werden. Somit bietet das SnapCenter -Produkt von NetApp Kunden eine robuste und zuverlässige Lösung für Datenschutz und -verwaltung.

Disaster Recovery mit NetApp SnapMirror

In diesem Abschnitt wird DR (Disaster Recovery) mit SnapMirror für die Vektordatenbanklösung für NetApp erläutert.

Disaster Recovery mit NetApp SnapMirror



Die Notfallwiederherstellung ist für die Aufrechterhaltung der Integrität und Verfügbarkeit einer Vektordatenbank von entscheidender Bedeutung, insbesondere angesichts ihrer Rolle bei der Verwaltung hochdimensionaler Daten und der Durchführung komplexer Ähnlichkeitssuchen. Eine gut geplante und implementierte Notfallwiederherstellungsstrategie stellt sicher, dass bei unvorhergesehenen Vorfällen wie Hardwareausfällen, Naturkatastrophen oder Cyberangriffen keine Daten verloren gehen oder gefährdet werden. Dies ist insbesondere für Anwendungen von Bedeutung, die auf Vektordatenbanken basieren, bei denen der Verlust oder die Beschädigung von Daten zu erheblichen Betriebsstörungen und finanziellen Verlusten führen kann. Darüber hinaus gewährleistet ein robuster Notfallwiederherstellungsplan auch die Geschäftskontinuität, indem er Ausfallzeiten minimiert und eine schnelle Wiederherstellung der Dienste ermöglicht. Dies wird durch das NetApp Datenreplikationsprodukt SnapMirrror über verschiedene geografische Standorte, regelmäßige Backups und Failover-Mechanismen erreicht. Daher ist die Notfallwiederherstellung nicht nur eine Schutzmaßnahme, sondern ein entscheidender Bestandteil einer verantwortungsvollen und effizienten Vektordatenbankverwaltung.

SnapMirror von NetApp ermöglicht die Datenreplikation von einem NetApp ONTAP Speichercontroller zu einem anderen und wird hauptsächlich für Disaster Recovery (DR) und Hybridlösungen verwendet. Im Kontext einer Vektordatenbank erleichtert dieses Tool den reibungslosen Datenübergang zwischen lokalen und Cloud-Umgebungen. Dieser Übergang wird ohne die Notwendigkeit von Datenkonvertierungen oder Anwendungsrefactoring erreicht, wodurch die Effizienz und Flexibilität der Datenverwaltung über mehrere Plattformen hinweg verbessert wird.

Die NetApp Hybrid-Lösung in einem Vektordatenbankszenario kann weitere Vorteile bringen:

1. Skalierbarkeit: Die Hybrid-Cloud-Lösung von NetApp bietet die Möglichkeit, Ihre Ressourcen entsprechend Ihren Anforderungen zu skalieren. Sie können lokale Ressourcen für regelmäßige, vorhersehbare Workloads und Cloud-Ressourcen wie Amazon FSx ONTAP für NetApp ONTAP und Google Cloud NetApp Volume (NetApp Volumes) für Spitzenzeiten oder unerwartete Belastungen nutzen.
2. Kosteneffizienz: Mit dem Hybrid-Cloud-Modell von NetApp können Sie Ihre Kosten optimieren, indem Sie lokale Ressourcen für regelmäßige Workloads verwenden und nur dann für Cloud-Ressourcen zahlen, wenn Sie sie benötigen. Dieses Pay-as-you-go-Modell kann mit einem NetApp Instaclustr-Serviceangebot recht kostengünstig sein. Für On-Premise- und große Cloud-Service-Anbieter bietet instaclustr Support und Beratung.

3. Flexibilität: Die Hybrid Cloud von NetApp bietet Ihnen die Flexibilität, den Ort der Datenverarbeitung zu wählen. Sie können sich beispielsweise dafür entscheiden, komplexe Vektoroperationen vor Ort durchzuführen, wo Sie über leistungsfähigere Hardware verfügen, und weniger intensive Operationen in der Cloud.
4. Geschäftskontinuität: Im Katastrophenfall kann die Speicherung Ihrer Daten in einer NetApp Hybrid Cloud die Geschäftskontinuität sicherstellen. Sie können schnell auf die Cloud umsteigen, wenn Ihre lokalen Ressourcen betroffen sind. Wir können NetApp SnapMirror nutzen, um die Daten vor Ort in die Cloud und umgekehrt zu verschieben.
5. Innovation: Die Hybrid-Cloud-Lösungen von NetApp ermöglichen außerdem schnellere Innovationen, indem sie Zugriff auf hochmoderne Cloud-Dienste und -Technologien bieten. NetApp Innovationen in der Cloud wie Amazon FSx ONTAP für NetApp ONTAP, Azure NetApp Files und Google Cloud NetApp Volumes sind innovative Produkte und bevorzugte NAS der Cloud-Service-Anbieter.

Leistungsvalidierung der Vektordatenbank

In diesem Abschnitt wird die Leistungsvalidierung hervorgehoben, die an der Vektordatenbank durchgeführt wurde.

Leistungsvalidierung

Die Leistungsvalidierung spielt sowohl bei Vektordatenbanken als auch bei Speichersystemen eine entscheidende Rolle und ist ein Schlüsselfaktor für die Gewährleistung eines optimalen Betriebs und einer effizienten Ressourcennutzung. Vektordatenbanken, die für die Verarbeitung hochdimensionaler Daten und die Durchführung von Ähnlichkeitssuchen bekannt sind, müssen ein hohes Leistungsniveau aufrechterhalten, um komplexe Abfragen schnell und genau verarbeiten zu können. Mithilfe der Leistungsvalidierung können Engpässe identifiziert und Konfigurationen optimiert werden. Außerdem wird sichergestellt, dass das System die erwartete Belastung ohne Leistungseinbußen bewältigen kann. Ebenso ist bei Speichersystemen eine Leistungsvalidierung unerlässlich, um sicherzustellen, dass Daten effizient gespeichert und abgerufen werden, ohne dass es zu Latenzproblemen oder Engpässen kommt, die die Gesamtleistung des Systems beeinträchtigen könnten. Es hilft auch dabei, fundierte Entscheidungen über notwendige Upgrades oder Änderungen der Speicherinfrastruktur zu treffen. Daher ist die Leistungsvalidierung ein entscheidender Aspekt des Systemmanagements und trägt erheblich zur Aufrechterhaltung einer hohen Servicequalität, Betriebseffizienz und allgemeinen Systemzuverlässigkeit bei.

In diesem Abschnitt möchten wir uns eingehend mit der Leistungsvalidierung von Vektordatenbanken wie Milvus und pgvecto.rs befassen und uns dabei auf ihre Speicherleistungsmerkmale wie E/A-Profil und Verhalten des NetApp-Speichercontrollers zur Unterstützung von RAG- und Inferenz-Workloads innerhalb des LLM-Lebenszyklus konzentrieren. Wir werden alle Leistungsunterschiede bewerten und identifizieren, wenn diese Datenbanken mit der ONTAP Speicherlösung kombiniert werden. Unsere Analyse basiert auf wichtigen Leistungsindikatoren, beispielsweise der Anzahl der pro Sekunde verarbeiteten Abfragen (QPS).

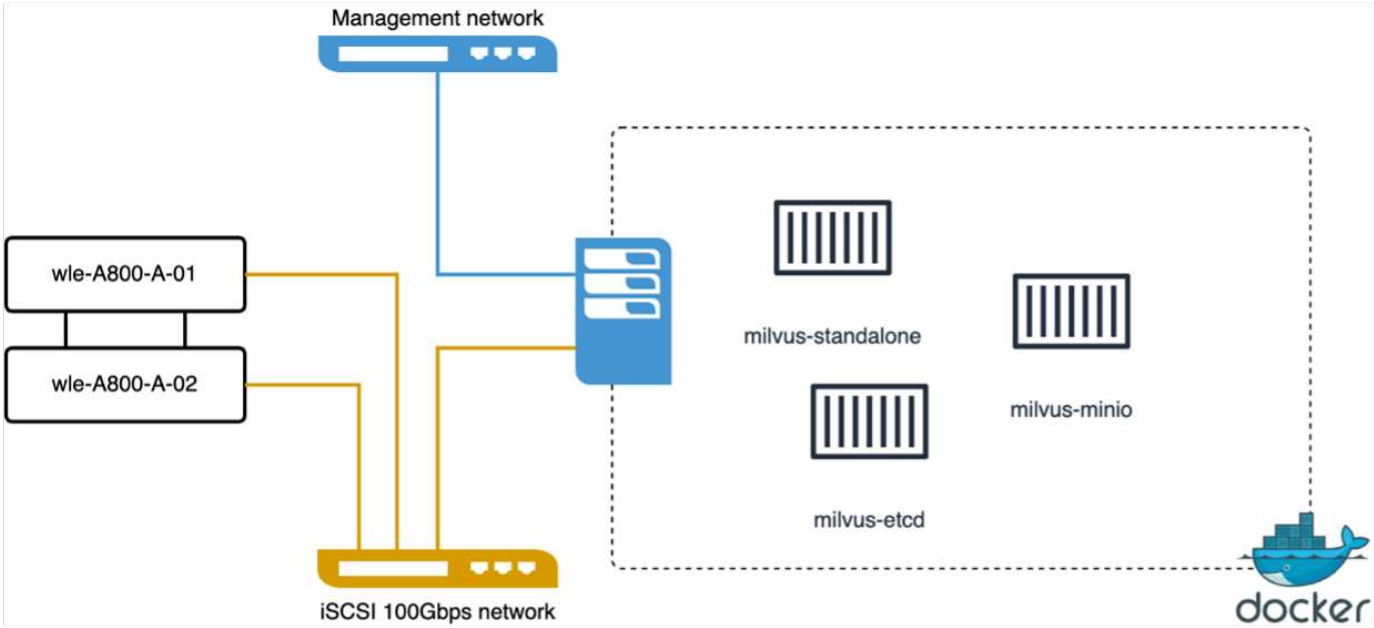
Bitte überprüfen Sie unten die für Milvus und den Fortschritt verwendete Methodik.

Details	Milvus (Standalone und Cluster)	Postgres(pgvecto.rs) #
Version	2.3.2	0.2.0
Dateisystem	XFS auf iSCSI-LUNs	
Arbeitslastgenerator	"VectorDB-Bench"– Version 0.0.5	
Datensätze	LAION-Datensatz * 10 Millionen Einbettungen * 768 Dimensionen * ~300 GB Datensatzgröße	

Speichercontroller	AFF 800 * Version – 9.14.1 * 4 x 100GbE – für Milvus und 2x 100GbE für Postgres * iscsi	

VectorDB-Bench mit Milvus-Standalone-Cluster

Wir haben die folgende Leistungsvalidierung auf dem eigenständigen Milvus-Cluster mit VectorDB-Bench durchgeführt. Die Netzwerk- und Serverkonnektivität des eigenständigen Milvus-Clusters ist unten aufgeführt.



In diesem Abschnitt teilen wir unsere Beobachtungen und Ergebnisse aus dem Testen der eigenständigen Milvus-Datenbank. . Für diese Tests haben wir DiskANN als Indextyp ausgewählt. . Das Aufnehmen, Optimieren und Erstellen von Indizes für einen Datensatz von etwa 100 GB dauerte etwa 5 Stunden. Während des größten Teils dieser Dauer lief der mit 20 Kernen ausgestattete Milvus-Server (was bei aktiviertem Hyper-Threading 40 vcpus entspricht) mit seiner maximalen CPU-Kapazität von 100 %. Wir haben festgestellt, dass DiskANN besonders wichtig für große Datensätze ist, die die Größe des Systemspeichers überschreiten. . In der Abfragephase beobachteten wir eine Abfragerate pro Sekunde (QPS) von 10,93 mit einem Rückruf von 0,9987. Die Latenzzeit für Abfragen im 99. Perzentil wurde mit 708,2 Millisekunden gemessen.

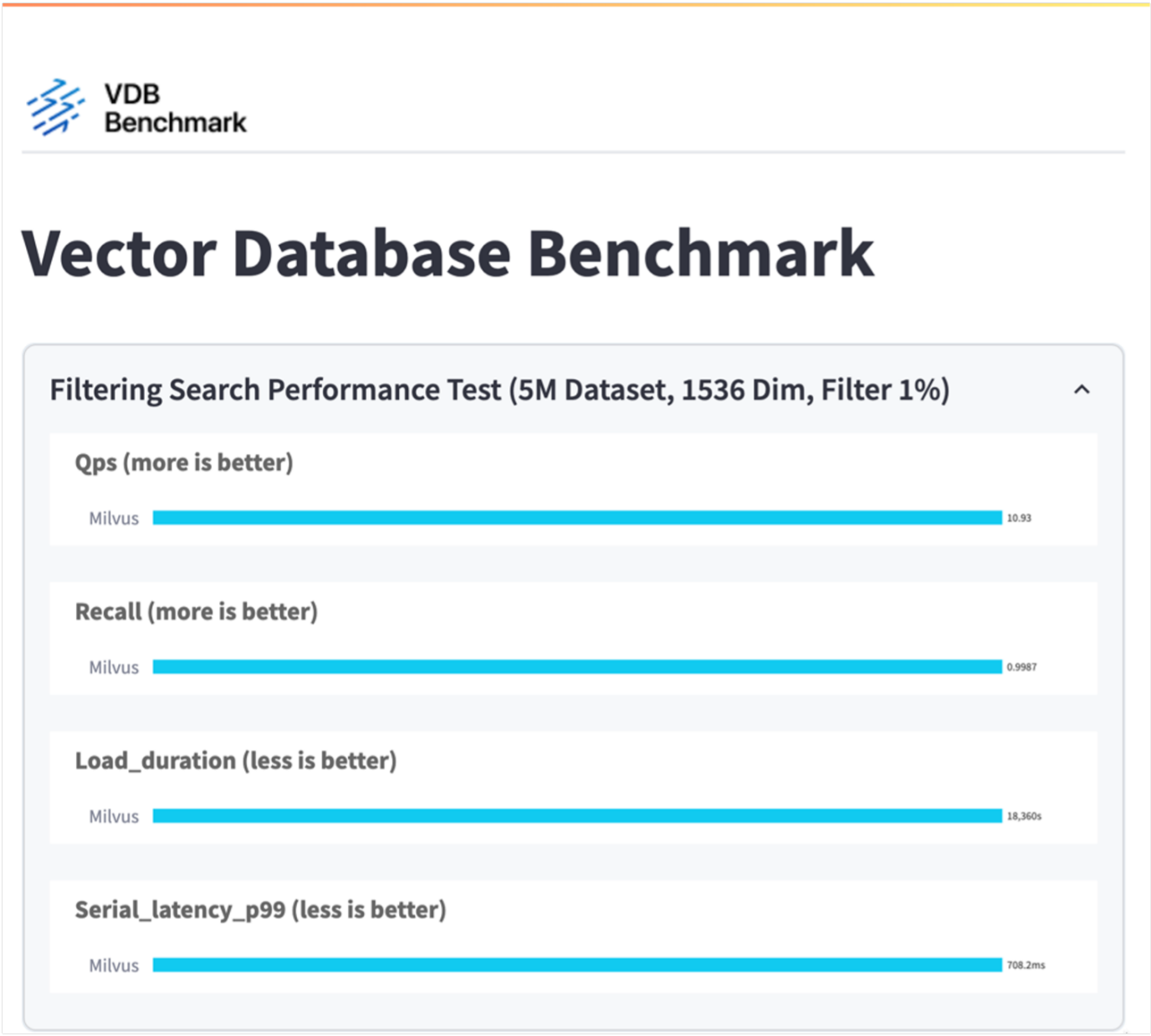
Aus Speichersicht gab die Datenbank während der Aufnahme-, Post-Insert-Optimierungs- und Indexerstellungsphasen etwa 1.000 Operationen/Sekunde aus. In der Abfragephase waren 32.000 Operationen/Sek. erforderlich.

Der folgende Abschnitt stellt die Speicherleistungsmetriken vor.

Arbeitslastphase	**Metrisch**	**Wert**
Datenaufnahme und Optimierung nach dem Einfügen	IOPS	< 1.000
	Latenz	< 400 µs
	Arbeitsbelastung	Lese-/Schreib-Mix, hauptsächlich Schreibvorgänge
	IO-Größe	64 KB

Arbeitslastphase	Metrisch	Wert
Abfrage	IOPS	Höchststand bei 32.000
	Latenz	< 400 µs
	Arbeitsbelastung	100 % zwischengespeicherte Lesevorgänge
	IO-Größe	Hauptsächlich 8 KB

Das VectorDB-Bench-Ergebnis ist unten.

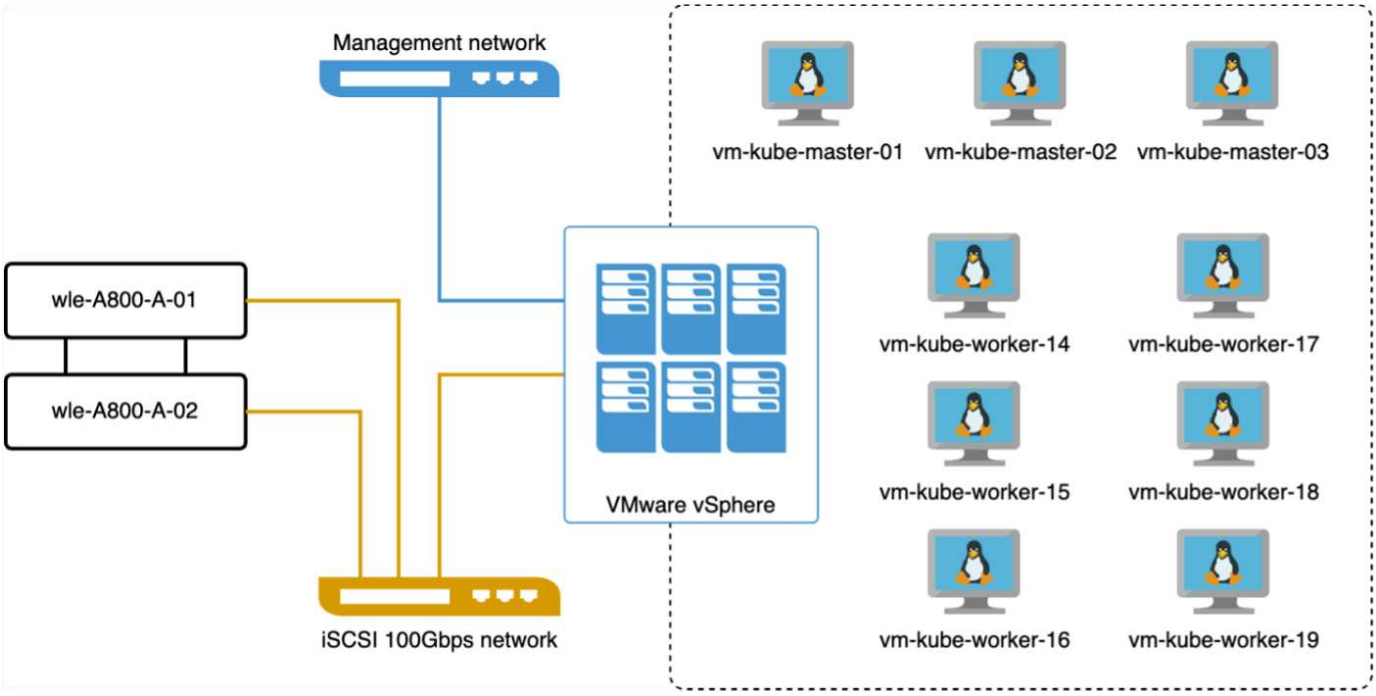


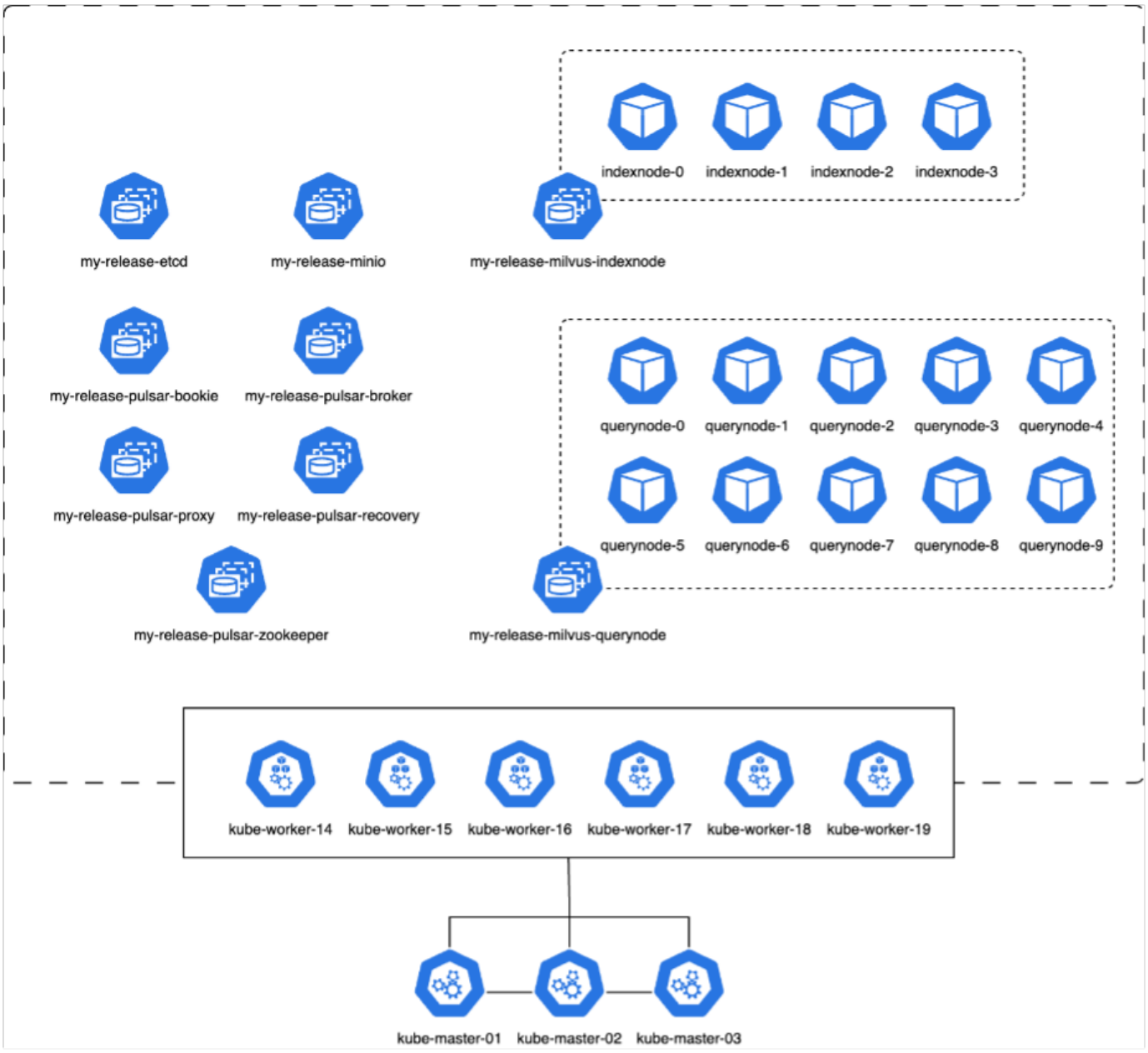
Aus der Leistungsvalidierung der eigenständigen Milvus-Instanz geht hervor, dass die aktuelle Konfiguration nicht ausreicht, um einen Datensatz von 5 Millionen Vektoren mit einer Dimensionalität von 1536 zu unterstützen. Wir haben festgestellt, dass der Speicher über ausreichende Ressourcen verfügt und keinen Engpass im System darstellt.

VectorDB-Bench mit Milvus-Cluster

In diesem Abschnitt besprechen wir die Bereitstellung eines Milvus-Clusters in einer Kubernetes-Umgebung. Dieses Kubernetes-Setup wurde auf einer VMware vSphere-Bereitstellung erstellt, die die Kubernetes-Master- und Worker-Knoten hostete.

Die Details der VMware vSphere- und Kubernetes-Bereitstellungen werden in den folgenden Abschnitten vorgestellt.





In diesem Abschnitt stellen wir unsere Beobachtungen und Ergebnisse aus dem Testen der Milvus-Datenbank vor. * Der verwendete Indextyp war DiskANN. * Die folgende Tabelle bietet einen Vergleich zwischen den Standalone- und Cluster-Bereitstellungen bei der Arbeit mit 5 Millionen Vektoren bei einer Dimensionalität von 1536. Wir haben festgestellt, dass die für die Datenaufnahme und die Optimierung nach dem Einfügen benötigte Zeit bei der Clusterbereitstellung kürzer war. Die Latenzzeit für Abfragen im 99. Perzentil wurde im Cluster-Einsatz im Vergleich zum Standalone-Setup um das Sechsfache reduziert. * Obwohl die Abfragerate pro Sekunde (QPS) bei der Clusterbereitstellung höher war, lag sie nicht auf dem gewünschten Niveau.

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%

Die folgenden Bilder bieten eine Ansicht verschiedener Speichermetriken, einschließlich der Speicherclusterlatenz und der gesamten IOPS (Input/Output Operations Per Second).

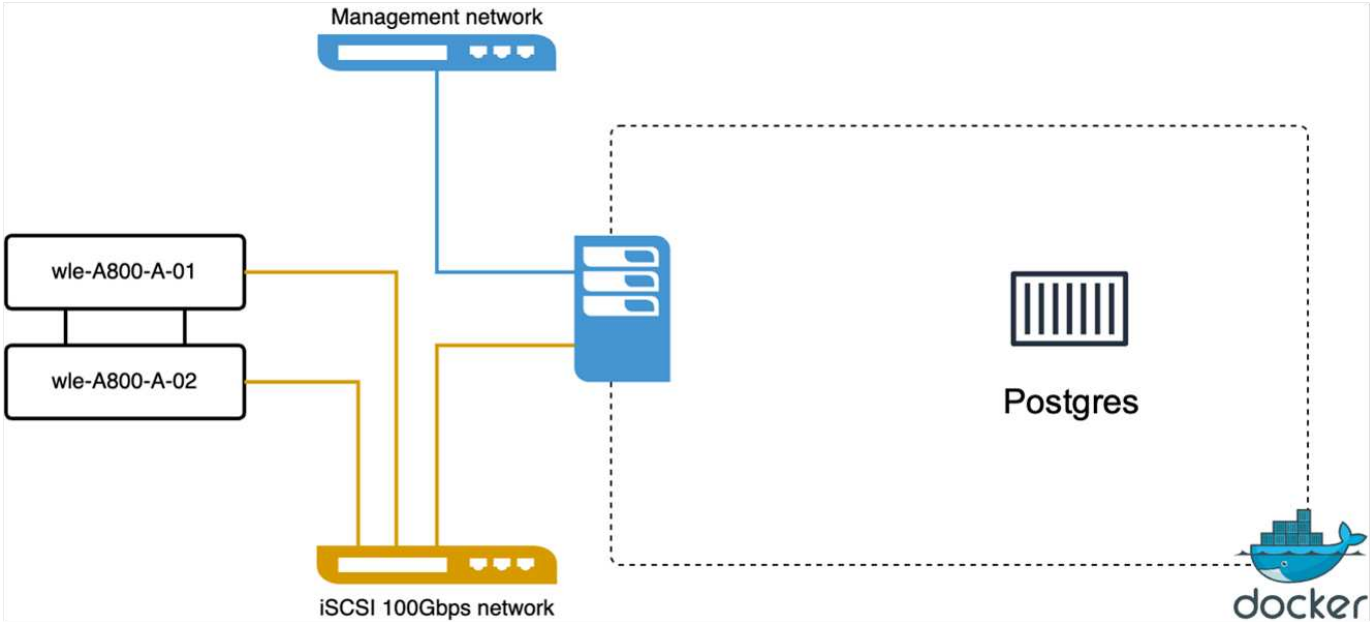
Der folgende Abschnitt stellt die wichtigsten Leistungskennzahlen für den Speicher vor.

Arbeitslastphase	Metrisch	Wert
Datenaufnahme und Optimierung nach dem Einfügen	IOPS	< 1.000
	Latenz	< 400 µs
	Arbeitsbelastung	Lese-/Schreib-Mix, hauptsächlich Schreibvorgänge
	IO-Größe	64 KB
Abfrage	IOPS	Höchststand bei 147.000
	Latenz	< 400 µs
	Arbeitsbelastung	100 % zwischengespeicherte Lesevorgänge
	IO-Größe	Hauptsächlich 8 KB

Basierend auf der Leistungsvalidierung sowohl des eigenständigen Milvus als auch des Milvus-Clusters präsentieren wir die Details des Speicher-E/A-Profils. * Wir haben festgestellt, dass das E/A-Profil sowohl bei eigenständigen als auch bei Cluster-Bereitstellungen konsistent bleibt. * Der beobachtete Unterschied bei den Spitzen-IOPS kann auf die größere Anzahl von Clients in der Clusterbereitstellung zurückgeführt werden.

vectorDB-Bench mit Postgres (pgvecto.rs)

Wir haben die folgenden Aktionen mit VectorDB-Bench an PostgreSQL (pgvecto.rs) durchgeführt: Die Details zur Netzwerk- und Serverkonnektivität von PostgreSQL (insbesondere pgvecto.rs) lauten wie folgt:

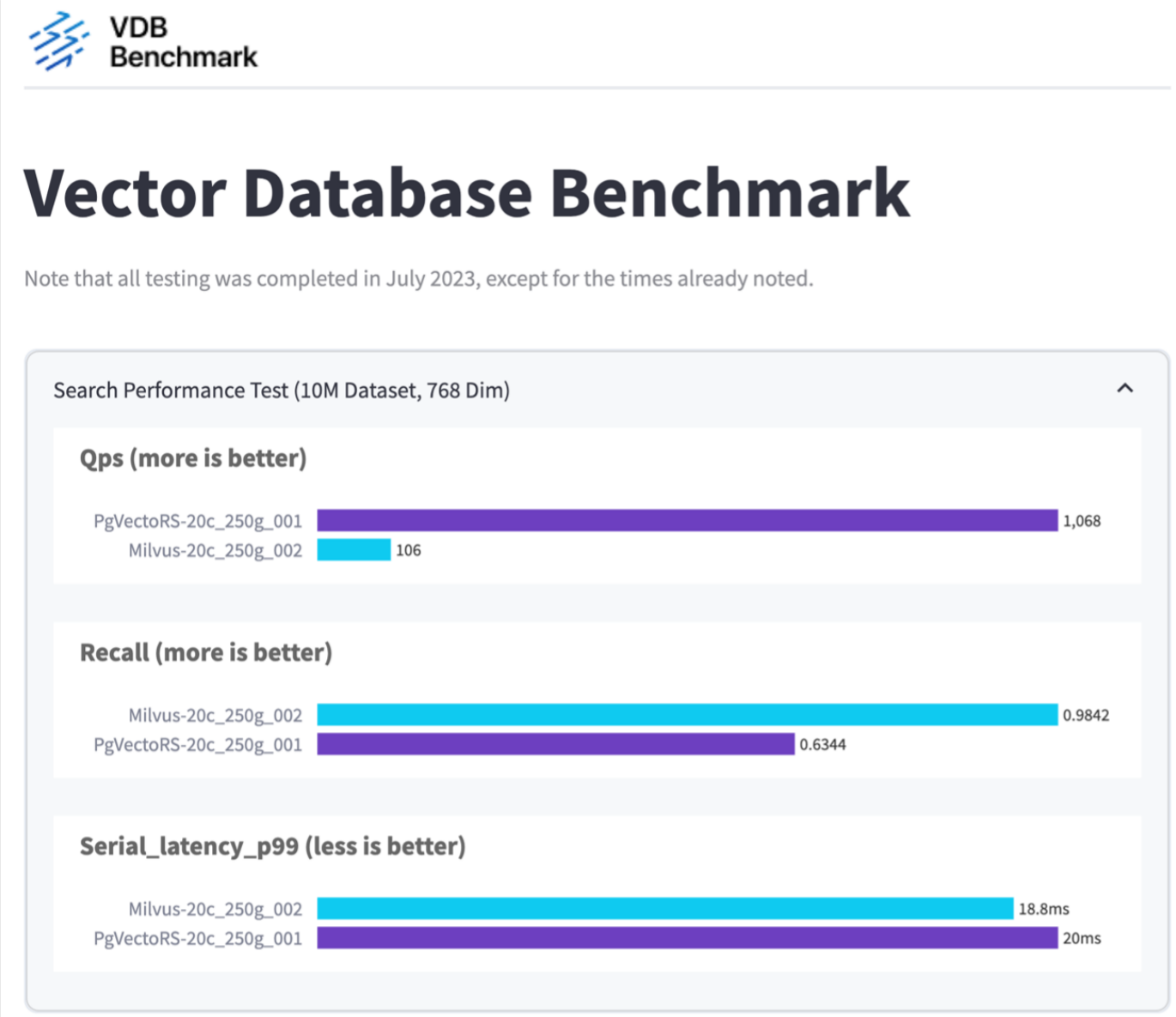


In diesem Abschnitt teilen wir unsere Beobachtungen und Ergebnisse aus dem Testen der PostgreSQL-Datenbank, insbesondere mit pgvecto.rs. * Wir haben HNSW als Indextyp für diese Tests ausgewählt, da DiskANN zum Zeitpunkt des Tests für pgvecto.rs nicht verfügbar war. * Während der Datenaufnahmephase haben wir den Cohere-Datensatz geladen, der aus 10 Millionen Vektoren mit einer Dimensionalität von 768 besteht. Dieser Vorgang dauerte ungefähr 4,5 Stunden. * In der Abfragephase haben wir eine Abfragerate pro Sekunde (QPS) von 1.068 mit einem Rückruf von 0,6344 beobachtet. Die Latenzzeit für Abfragen im 99. Perzentil wurde mit 20 Millisekunden gemessen. Während des größten Teils der Laufzeit war die CPU des Clients zu 100 % ausgelastet.

Die folgenden Bilder bieten eine Ansicht verschiedener Speichermetriken, einschließlich der Gesamt-IOPS (Input/Output Operations Per Second) der Speicherclusterlatenz.


```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-metrics.png["Abbildung, die einen Eingabe-
/Ausgabedialog zeigt oder schriftlichen Inhalt darstellt"]
```

Leistungsvergleich zwischen Milvus und Postgres auf Vector DB Bench



Basierend auf unserer Leistungsvalidierung von Milvus und PostgreSQL mit VectorDBBench haben wir Folgendes beobachtet:

- Indextyp: HNSW
- Datensatz: Cohere mit 10 Millionen Vektoren in 768 Dimensionen

Wir haben festgestellt, dass pgvecto.rs eine Abfragen-pro-Sekunde-Rate (QPS) von 1.068 mit einem Recall von 0,6344 erreichte, während Milvus eine QPS-Rate von 106 mit einem Recall von 0,9842 erreichte.

Wenn hohe Präzision bei Ihren Abfragen Priorität hat, ist Milvus besser als pgvecto.rs, da es einen höheren Anteil relevanter Elemente pro Abfrage abruft. Wenn jedoch die Anzahl der Abfragen pro Sekunde ein entscheidenderer Faktor ist, übertrifft pgvecto.rs Milvus. Es ist jedoch wichtig zu beachten, dass die Qualität der über pgvecto.rs abgerufenen Daten geringer ist, da etwa 37 % der Suchergebnisse irrelevante Elemente sind.

Beobachtung basierend auf unseren Leistungsvalidierungen:

Basierend auf unseren Leistungsvalidierungen haben wir folgende Beobachtungen gemacht:

In Milvus ähnelt das E/A-Profil stark einer OLTP-Workload, wie sie beispielsweise bei Oracle SLOB auftritt. Der Benchmark besteht aus drei Phasen: Datenaufnahme, Nachoptimierung und Abfrage. Die Anfangsphasen sind hauptsächlich durch 64-KB-Schreibvorgänge gekennzeichnet, während die Abfragephase überwiegend 8-KB-Lesevorgänge umfasst. Wir erwarten, dass ONTAP die Milvus-E/A-Last effizient bewältigt.

Das PostgreSQL-E/A-Profil stellt keine anspruchsvolle Speicherarbeitslast dar. Angesichts der derzeit laufenden In-Memory-Implementierung konnten wir während der Abfragephase keine Festplatten-E/A beobachten.

DiskANN erweist sich als entscheidende Technologie zur Speicherdifferenzierung. Es ermöglicht die effiziente Skalierung der Vektor-DB-Suche über die Systemspeichergrenze hinaus. Es ist jedoch unwahrscheinlich, dass mit In-Memory-Vektor-DB-Indizes wie HNSW eine Differenzierung der Speicherleistung erreicht wird.

Es ist auch erwähnenswert, dass der Speicher während der Abfragephase keine kritische Rolle spielt, wenn der Indextyp HSNW ist. Dies ist die wichtigste Betriebsphase für Vektordatenbanken, die RAG-Anwendungen unterstützen. Dies bedeutet, dass die Speicherleistung keinen signifikanten Einfluss auf die Gesamtleistung dieser Anwendungen hat.

Vektordatenbank mit Instaclustr unter Verwendung von PostgreSQL: pgvector

In diesem Abschnitt werden die Einzelheiten der Integration des Instaclustr-Produkts mit PostgreSQL über die Pgvector-Funktionalität in der Vektordatenbanklösung für NetApp erläutert.

Vektordatenbank mit Instaclustr unter Verwendung von PostgreSQL: pgvector

In diesem Abschnitt gehen wir näher auf die Einzelheiten der Integration des Instaclustr-Produkts mit PostgreSQL auf der Grundlage der PgVector-Funktionalität ein. Wir haben ein Beispiel für „So verbessern Sie die Genauigkeit und Leistung Ihres LLM mit PGVector und PostgreSQL: Einführung in Einbettungen und die Rolle von PGVector“. Bitte überprüfen Sie die"Blog" um weitere Informationen zu erhalten.

Anwendungsfälle für Vektordatenbanken

Dieser Abschnitt bietet einen Überblick über die Anwendungsfälle für die NetApp Vector-Datenbanklösung.

Anwendungsfälle für Vektordatenbanken

In diesem Abschnitt besprechen wir zwei Anwendungsfälle, nämlich Retrieval Augmented Generation mit großen Sprachmodellen und NetApp IT-Chatbot.

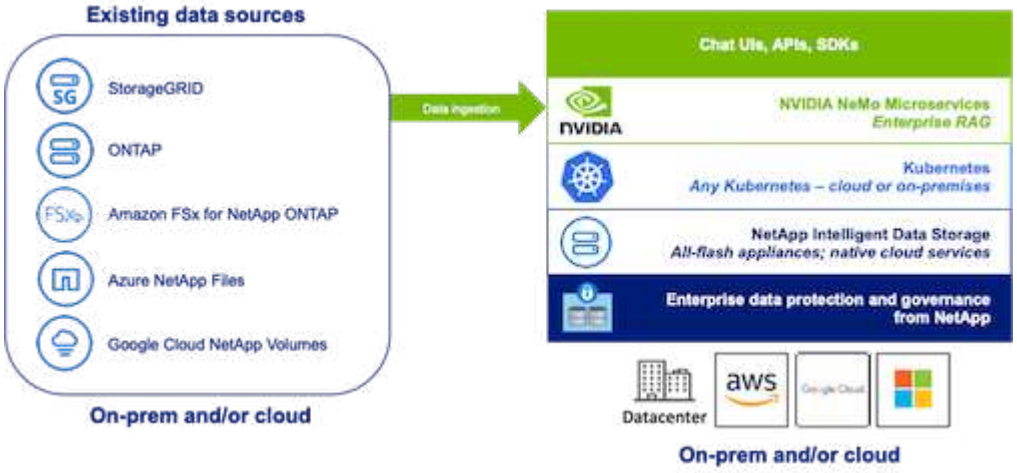
Retrieval Augmented Generation (RAG) mit großen Sprachmodellen (LLMs)

```
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.
```

Der NVIDIA Enterprise RAG LLM Operator ist ein nützliches Tool zur Implementierung von RAG im Unternehmen. Mit diesem Operator kann eine vollständige RAG-Pipeline bereitgestellt werden. Die RAG-Pipeline kann angepasst werden, um entweder Milvus oder pgvecto als Vektordatenbank zum Speichern von Wissensdatenbank-Einbettungen zu verwenden. Weitere Einzelheiten finden Sie in der Dokumentation.

```
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.
```

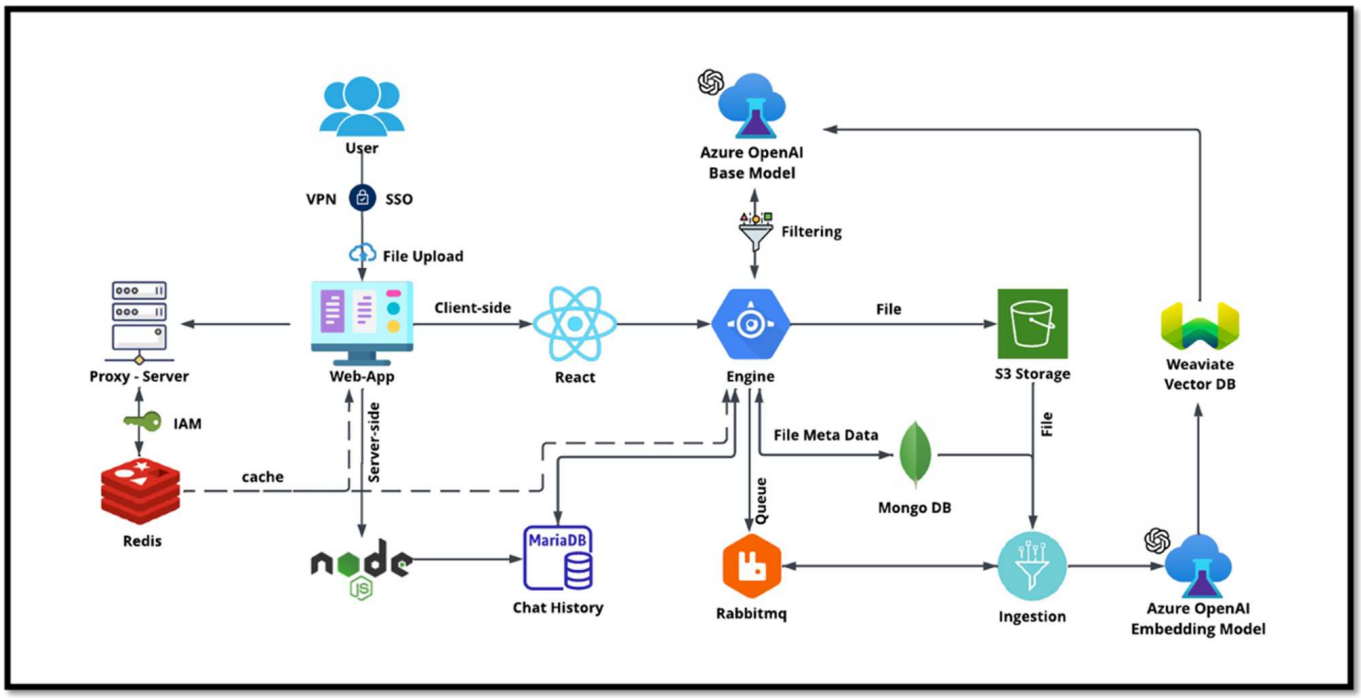
Abbildung 1) Enterprise RAG mit NVIDIA NeMo Microservices und NetApp



Anwendungsfall für den NetApp IT-Chatbot

Der Chatbot von NetApp dient als weiterer Echtzeit-Anwendungsfall für die Vektordatenbank. In diesem Fall bietet die NetApp Private OpenAI Sandbox eine effektive, sichere und effiziente Plattform für die Verwaltung von Abfragen interner NetApp-Benutzer. Durch die Integration strenger Sicherheitsprotokolle, effizienter Datenverwaltungssysteme und ausgefeilter KI-Verarbeitungsfunktionen werden den Benutzern über die SSO-Authentifizierung qualitativ hochwertige und präzise Antworten basierend auf ihren Rollen und Verantwortlichkeiten in der Organisation garantiert. Diese Architektur unterstreicht das Potenzial der

Zusammenführung fortschrittlicher Technologien zur Schaffung benutzerorientierter, intelligenter Systeme.



Der Anwendungsfall kann in vier Hauptabschnitte unterteilt werden.

Benutzerauthentifizierung und -verifizierung:

- Benutzeranfragen durchlaufen zunächst den NetApp Single Sign-On (SSO)-Prozess, um die Identität des Benutzers zu bestätigen.
- Nach erfolgreicher Authentifizierung prüft das System die VPN-Verbindung, um eine sichere Datenübertragung zu gewährleisten.

Datenübertragung und -verarbeitung:

- Sobald das VPN validiert ist, werden die Daten über die Webanwendungen NetAIChat oder NetAICreate an MariaDB gesendet. MariaDB ist ein schnelles und effizientes Datenbanksystem zum Verwalten und Speichern von Benutzerdaten.
- MariaDB sendet die Informationen dann an die NetApp Azure-Instanz, die die Benutzerdaten mit der KI-Verarbeitungseinheit verbindet.

Interaktion mit OpenAI und Inhaltsfilterung:

- Die Azure-Instanz sendet die Fragen des Benutzers an ein Inhaltsfiltersystem. Dieses System bereinigt die Abfrage und bereitet sie für die Verarbeitung vor.
- Die bereinigte Eingabe wird dann an das Azure OpenAI-Basismodell gesendet, das basierend auf der Eingabe eine Antwort generiert.

Antwortgenerierung und -moderation:

- Die Antwort des Basismodells wird zunächst überprüft, um sicherzustellen, dass sie korrekt ist und den Inhaltsstandards entspricht.
- Nach bestandener Prüfung wird die Antwort an den Benutzer zurückgesendet. Dieser Prozess stellt sicher,

dass der Benutzer eine klare, genaue und angemessene Antwort auf seine Anfrage erhält.

Abschluss

Dieser Abschnitt schließt die Vektordatenbanklösung für NetApp ab.

Abschluss

Zusammenfassend bietet dieses Dokument einen umfassenden Überblick über die Bereitstellung und Verwaltung von Vektordatenbanken wie Milvus und pgvector auf NetApp -Speicherlösungen. Wir haben die Infrastrukturrichtlinien für die Nutzung von NetApp ONTAP und StorageGRID Objektspeicher besprochen und die Milvus-Datenbank in AWS FSx ONTAP über Datei- und Objektspeicher validiert.

Wir haben die Datei-Objekt-Dualität von NetApp untersucht und ihren Nutzen nicht nur für Daten in Vektordatenbanken, sondern auch für andere Anwendungen demonstriert. Wir haben auch hervorgehoben, wie SnapCenter, das Enterprise-Management-Produkt von NetApp, Sicherungs-, Wiederherstellungs- und Klonfunktionen für Vektordatenbankdaten bietet und so die Datenintegrität und -verfügbarkeit gewährleistet.

Das Dokument erläutert außerdem, wie die Hybrid Cloud-Lösung von NetApp Datenreplikation und -schutz in lokalen und Cloud-Umgebungen bietet und so ein nahtloses und sicheres Datenmanagement ermöglicht. Wir gaben Einblicke in die Leistungsvalidierung von Vektordatenbanken wie Milvus und pgvecto auf NetApp ONTAP und lieferten wertvolle Informationen zu ihrer Effizienz und Skalierbarkeit.

Abschließend haben wir zwei Anwendungsfälle für generative KI besprochen: RAG mit LLM und die interne ChatAI von NetApp. Diese praktischen Beispiele unterstreichen die realen Anwendungen und Vorteile der in diesem Dokument beschriebenen Konzepte und Praktiken. Insgesamt dient dieses Dokument als umfassender Leitfaden für alle, die die leistungsstarken Speicherlösungen von NetApp für die Verwaltung von Vektordatenbanken nutzen möchten.

Danksagung

Der Autor möchte den unten aufgeführten Mitwirkenden und anderen Personen, die durch ihr Feedback und ihre Kommentare dazu beigetragen haben, dass dieses Dokument für NetApp Kunden und NetApp Bereiche wertvoll ist, herzlich danken.

1. Sathish Thyagarajan, technischer Marketingingenieur, ONTAP AI & Analytics, NetApp
2. Mike Oglesby, Technischer Marketingingenieur, NetApp
3. AJ Mahajan, Senior Director, NetApp
4. Joe Scott, Manager, Workload Performance Engineering, NetApp
5. Puneet Dhawan, Senior Director, Produktmanagement Fsx, NetApp
6. Yuval Kalderon, Senior Product Manager, FSx-Produktteam, NetApp

Wo Sie weitere Informationen finden

Weitere Informationen zu den in diesem Dokument beschriebenen Informationen finden Sie in den folgenden Dokumenten und/oder auf den folgenden Websites:

- Milvus-Dokumentation - https://milvus.io/docs/overview.md
- Eigenständige Milvus-Dokumentation - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- NetApp Produktdokumentationhttps://www.netapp.com/support-and-training/documentation/[]

- instaclustr -"Installclustr-Dokumentation"

Versionsverlauf

Version	Datum	Dokumentversionsverlauf
Version 1.0	April 2024	Erstveröffentlichung

Anhang A: Values.yaml

Dieser Abschnitt enthält Beispiel-YAML-Code für die in der NetApp -Vektordatenbanklösung verwendeten Werte.

Anhang A: Values.yaml

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

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

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

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

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

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

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

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

  ## List of IP addresses at which the Milvus service is available
```

```
  ## Ref: https://kubernetes.io/docs/user-guide/services/#external-ips
  ##
  externalIPs: []
  #   - externalIp1

  # LoadBalancerSourcesRange is a list of allowed CIDR values, which are
combined with ServicePort to
  # set allowed inbound rules on the security group assigned to the master
load balancer
  loadBalancerSourceRanges:
  - 0.0.0.0/0
  # Optionally assign a known public LB IP
  # loadBalancerIP: 1.2.3.4

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

serviceAccount:
  create: false
  name:
  annotations:
  labels:

metrics:
  enabled: true

  serviceMonitor:
```

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

livenessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 30
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

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

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

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Logs Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
```

```
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ## ReadWriteMany access mode required for milvus cluster.
      ##
      storageClass: default
      accessModes: ReadWriteMany
      size: 10Gi
      subPath: ""

## Heaptrack traces all memory allocations and annotates these events with
stack traces.
## See more: https://github.com/KDE/heaptrack
## Enable heaptrack in production is not recommended.
heaptrack:
  image:
    repository: milvusdb/heaptrack
    tag: v0.1.0
    pullPolicy: IfNotPresent

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

  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
    mountPath: "/var/lib/milvus"
    ## If true, alertmanager will create/use a Persistent Volume Claim
    ## If false, use emptyDir
```

```
    ##
    enabled: true
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ##
      storageClass:
      accessModes: ReadWriteOnce
      size: 50Gi
      subPath: ""

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

```
#    key: LS0tLS1CRUdJTiBQU--REDUCT
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

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

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

queryCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Query Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
```

```
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for query coordinator

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

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

indexCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1   # Run Index Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
```

```
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for index coordinator

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

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

dataCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Data Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
```

```
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for data coordinator

  service:
    port: 13333
    annotations: {}
    labels: {}
    clusterIP: ""

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

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

```
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for Mixture coordinator

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

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


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

minio:
  enabled: false
  name: minio
```

```
  mode: distributed
  image:
    tag: "RELEASE.2023-03-20T20-16-18Z"
    pullPolicy: IfNotPresent
  accessKey: minioadmin
  secretKey: minioadmin
  existingSecret: ""
  bucketName: "milvus-bucket"
  rootPath: file
  useIAM: false
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
  podDisruptionBudget:
    enabled: false
  resources:
    requests:
      memory: 2Gi

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

  service:
    type: ClusterIP
    port: 9000

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

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

  readinessProbe:
    enabled: true
```

```
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 1
    successThreshold: 1
    failureThreshold: 5

  startupProbe:
    enabled: true
    initialDelaySeconds: 0
    periodSeconds: 10
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 60

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

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

  service:
    type: ClusterIP
    port: 2379
    peerPort: 2380

  auth:
    rbac:
      enabled: false

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

  ## Change default timeout periods to mitigate zoobie probe process
  livenessProbe:
```

```
    enabled: true
    timeoutSeconds: 10

  readinessProbe:
    enabled: true
    periodSeconds: 20
    timeoutSeconds: 10

  ## Enable auto compaction
  ## compaction by every 1000 revision
  ##
  autoCompactionMode: revision
  autoCompactionRetention: "1000"

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

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

pulsar:
  enabled: true
  name: pulsar

  fullnameOverride: ""
  persistence: true

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

  rbac:
    enabled: false
    psp: false
    limit_to_namespace: true

  affinity:
    anti_affinity: false

## enableAntiAffinity: no
```

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

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

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

zookeeper:
  volumes:
```

```
    persistence: true
    data:
      name: data
      size: 20Gi   #SSD Required
      storageClassName: default
  resources:
    requests:
      memory: 1024Mi
      cpu: 0.3
  configData:
    PULSAR_MEM: >
      -Xms1024m
      -Xmx1024m
    PULSAR_GC: >
       -Dcom.sun.management.jmxremote
       -Djute.maxbuffer=10485760
       -XX:+ParallelRefProcEnabled
       -XX:+UnlockExperimentalVMOptions
       -XX:+DoEscapeAnalysis
       -XX:+DisableExplicitGC
       -XX:+PerfDisableSharedMem
       -Dzookeeper.forceSync=no
  pdb:
    usePolicy: false

bookkeeper:
  replicaCount: 3
  volumes:
    persistence: true
    journal:
      name: journal
      size: 100Gi
      storageClassName: default
    ledgers:
      name: ledgers
      size: 200Gi
      storageClassName: default
  resources:
    requests:
      memory: 2048Mi
      cpu: 1
  configData:
    PULSAR_MEM: >
      -Xms4096m
      -Xmx4096m
      -XX:MaxDirectMemorySize=8192m
```

```
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+UseG1GC -XX:MaxGCPauseMillis=10
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
        -XX:+PerfDisableSharedMem
        -XX:+PrintGCDetails
      nettyMaxFrameSizeBytes: "104867840"
    pdb:
      usePolicy: false

  broker:
    component: broker
    podMonitor:
      enabled: false
    replicaCount: 1
    resources:
      requests:
        memory: 4096Mi
        cpu: 1.5
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
      maxMessageSize: "104857600"
```

```
    defaultRetentionTimeInMinutes: "10080"
    defaultRetentionSizeInMB: "-1"
    backlogQuotaDefaultLimitGB: "8"
    ttlDurationDefaultInSeconds: "259200"
    subscriptionExpirationTimeMinutes: "3"
    backlogQuotaDefaultRetentionPolicy: producer_exception
  pdb:
    usePolicy: false

autorecovery:
  resources:
    requests:
      memory: 512Mi
      cpu: 1

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

pulsar_manager:
  service:
    type: ClusterIP

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

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

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

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

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

  persistence:
    enabled: true
    storageClass:
    accessMode: ReadWriteOnce
```

```
    size: 300Gi

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

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

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

  service:
    type: ClusterIP
    ports:
      client: 9092

  zookeeper:
    enabled: true
    replicaCount: 3

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

```
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false

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

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

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

###################################
# External kafka
# - these configs are only used when `externalKafka.enabled` is true
###################################
externalKafka:
  enabled: false
  brokerList: localhost:9092
```

```
  securityProtocol: SASL_SSL
  sasl:
    mechanisms: PLAIN
    username: ""
    password: ""
root@node2:~#
```

Anhang B: prepare_data_netapp_new.py

Dieser Abschnitt enthält ein Beispiel-Python-Skript zum Vorbereiten von Daten für die Vektordatenbank.

Anhang B: prepare_data_netapp_new.py

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

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

#############################################################################
#######
# 1. connect to Milvus
# Add a new connection alias `default` for Milvus server in
`localhost:19530`
# Actually the "default" alias is a buildin in PyMilvus.
```

```
# If the address of Milvus is the same as `localhost:19530`, you can omit
all
# parameters and call the method as: `connections.connect()`.
#
# Note: the `using` parameter of the following methods is default to
"default".
print(fmt.format("start connecting to Milvus"))

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

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

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

################################################################################
#######
# 2. create collection
# We're going to create a collection with 3 fields.
# +-+------------+------------+------------------
+------------------------------+
# | | field name | field type | other attributes |       field description
|
# +-+------------+------------+------------------
+------------------------------+
# |1|    "pk"    |    Int64   |  is_primary=True |      "primary field"
|
# | |            |            |   auto_id=False  |
|
# +-+------------+------------+------------------
+------------------------------+
# |2|  "random"  |    Double  |                  |      "a double field"
|
# +-+------------+------------+------------------
+------------------------------+
```

```
# |3|"embeddings"| FloatVector|     dim=8        |  "float vector with dim
8"   |
# +-+------------+------------+-----------------
+-----------------------------+
fields = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=False),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]

schema = CollectionSchema(fields, "hello_milvus_ntapnew_update2_sc")

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

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

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

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

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


schema2 = CollectionSchema(fields2, "hello_milvus_ntapnew_update2_sc2")

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

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

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

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

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

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

root@node2:~#
```

Anhang C: verify_data_netapp.py

Dieser Abschnitt enthält ein Python-Beispielskript, das zur Validierung der Vektordatenbank in der NetApp Vektordatenbanklösung verwendet werden kann.

Anhang 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:~#
```

Anhang D: docker-compose.yml

Dieser Abschnitt enthält Beispiel-YAML-Code für die Vektordatenbanklösung für NetApp.

Anhang 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
```