Vector Datenbanklösung mit NetApp

NetApp Solutions

NetApp
April 26, 2024

Inhalt

Vector Datenbanklösung mit NetApp

Karthikeyan Nagalingam und Rodrigo Nascimento, NetApp

Dieses Dokument bietet eine eingehende Erforschung der Implementierung und des Managements von Vektordatenbanken wie Milvus und pgvecto einer Open-Source-PostgreSQL-Erweiterung unter Verwendung von NetApp Storage-Lösungen. Es enthält Details zu den Infrastrukturrichtlinien für die Verwendung von NetApp ONTAP und StorageGRID Objekt-Storage und validiert die Anwendung der Milvus Datenbank in AWS FSX für NetApp ONTAP. Dieses Dokument dokumentiert die Dualität von NetApp in Bezug auf File-Objekte und sein Utility für Vektordatenbanken und Applikationen, die Vektoreinbetten unterstützen. Er hebt die Fähigkeiten von SnapCenter hervor, dem Enterprise Management-Produkt von NetApp, bei dem Backup- und Restore-Funktionen für Vektordatenbanken angeboten werden, die für Datenintegrität und Verfügbarkeit sorgen. In diesem Dokument wird die Hybrid-Cloud-Lösung von NetApp näher erörtert, in der die Rolle der Lösung für Datenreplizierung und -Sicherung in On-Premises- und Cloud-Umgebungen erörtert wird. Sie enthält Einblicke in die Performance-Validierung von Vektordatenbanken auf NetApp ONTAP und schließt mit zwei praktischen Anwendungsfällen zu generativer KI ab: RAG mit LLM und dem internen ChatAI von NetApp. Dieses Dokument dient als umfassender Leitfaden zur Nutzung von NetApp Storage-Lösungen für das Management von Vektordatenbanken.

Die Referenzarchitektur konzentriert sich auf Folgendes:

Einführung

Einführung

Vektordatenbanken bewältigen effektiv die Herausforderungen, die für die komplexe semantische Suche in

Large Language Models (LLMs) und generativer künstlicher Intelligenz (KI) entwickelt wurden. Im Gegensatz zu herkömmlichen Datenmanagementsystemen sind Vektordatenbanken in der Lage, verschiedene Datentypen zu verarbeiten und zu durchsuchen, darunter Bilder, Videos, Text, Audio und anderen Formen unstrukturierter Daten, indem der Inhalt der Daten selbst statt anhand von Etiketten oder Tags verwendet wird.

Die Grenzen von Relational Database Management Systems (RDBMS) sind gut dokumentiert, insbesondere ihre Probleme mit hochdimensionalen Datendarstellungen und unstrukturierten Daten, wie sie in KI-Anwendungen üblich sind. RDBMS erfordert oft einen zeitaufwändigen und fehleranfälligen Prozess der Abflachung von Daten in besser verwaltbare Strukturen, was zu Verzögerungen und Ineffizienzen bei der Suche führt. Vektor-Datenbanken sind jedoch darauf ausgelegt, diese Probleme zu umgehen. Sie bieten eine effizientere und genauere Lösung für das Management und die Suche durch komplexe und hochdimensionale Daten und erleichtern so den Fortschritt von KI-Applikationen.

Dieses Dokument dient als umfassender Leitfaden für Kunden, die derzeit Vektordatenbanken verwenden oder deren Verwendung planen. Darin werden die Best Practices für die Nutzung von Vektordatenbanken auf Plattformen wie NetApp ONTAP, NetApp StorageGRID, Amazon FSxN für NetApp ONTAP und SnapCenter erläutert. Die hier bereitgestellten Inhalte decken eine Reihe von Themen ab:

- Richtlinien für die Infrastruktur von Vektordatenbanken wie Milvus, die von NetApp Storage über NetApp ONTAP und StorageGRID Objekt-Storage bereitgestellt werden.
- Validierung der Milvus-Datenbank in AWS FSX für NetApp ONTAP durch Datei- und Objektspeicher.
- Geht auf die Dualität von File-Objekten von NetApp ein und demonstriert seine Nützlichkeit für Daten in Vektordatenbanken und anderen Applikationen.
- SnapCenter, das NetApp Produkt für Datensicherungsmanagement, bietet Backup- und Restore-Funktionen für Vektordatenbankdaten.
- Wie die Hybrid Cloud von NetApp Datenreplizierung und -Schutz in On-Premises- und Cloud-Umgebungen bietet.
- Bietet Einblicke in die Performance-Validierung von Vektordatenbanken wie Milvus und pgvektor auf NetApp ONTAP.
- Zwei spezifische Anwendungsfälle: Retrieval Augmented Generation (RAG) mit Large Language Models (LLM) und ChatAI des NetApp IT Teams. Damit werden praktische Beispiele für die beschriebenen Konzepte und Praktiken vorgestellt.

Lösungsüberblick

Lösungsüberblick

Diese Lösung veranschaulicht die besonderen Vorteile und Funktionen, die NetApp zur Bewältigung der Herausforderungen von Vektordatenbank-Kunden bietet. Durch die Nutzung von NetApp ONTAP, StorageGRID, den Cloud-Lösungen von NetApp und SnapCenter können Kunden ihren Geschäftsbetrieb wesentlich voranbringen. Diese Tools decken nicht nur bestehende Probleme ab, sondern steigern auch die Effizienz und Produktivität und tragen so zum Wachstum des gesamten Unternehmens bei.

Warum NetApp?

- Die Angebote von NetApp wie ONTAP und StorageGRID ermöglichen die Trennung von Storage und Computing und ermöglichen somit eine optimale Ressourcenauslastung je nach spezifischen Anforderungen. Dank dieser Flexibilität sind Kunden in der Lage, ihren Storage mithilfe von NetApp Storage-Lösungen unabhängig zu skalieren.
- Mithilfe der Storage Controller von NetApp können Kunden ihre Vektordatenbank mithilfe der NFS- und S3-Protokolle effizient mit Daten versorgen. Diese Protokolle erleichtern die Speicherung der Kundendaten

und das Management des Vektordatenbankindexes, sodass nicht mehr mehrere Kopien von Daten benötigt werden, auf die über Datei- und Objektmethoden zugegriffen wird.

- NetApp ONTAP bietet nativen Support für NAS und Objekt-Storage bei führenden Cloud-Service-Providern wie AWS, Azure und Google Cloud. Diese breite Kompatibilität gewährleistet eine nahtlose Integration und ermöglicht damit die Datenmobilität der Kunden, globale Verfügbarkeit, Disaster Recovery, dynamische Skalierbarkeit und hohe Performance.
- Dank der robusten Datenmanagement-Funktionen von NetApp können sich Kunden darauf verlassen, dass ihre Daten vor potenziellen Risiken und Bedrohungen gut geschützt sind. NetApp legt den Schwerpunkt auf Datensicherheit und bietet Kunden Sicherheit in Bezug auf die Sicherheit und Integrität ihrer wertvollen Informationen.

Vektordatenbank

Vektordatenbank

Eine Vektordatenbank ist eine spezielle Art von Datenbank, die für die Verarbeitung, Indizierung und Suche unstrukturierter Daten mithilfe von Einbetten aus Machine-Learning-Modellen entwickelt wurde. Anstatt Daten in einem herkömmlichen Tabellenformat zu organisieren, werden Daten als hochdimensionale Vektoren, auch als Vektoreinbetten bezeichnet, angeordnet. Diese einzigartige Struktur ermöglicht es der Datenbank, komplexe, mehrdimensionale Daten effizienter und genauer zu verarbeiten.

Eine der Schlüsselfunktionen einer Vektordatenbank ist die Verwendung generativer KI zur Durchführung von Analysen. Dazu gehören Ähnlichkeitssuchen, bei denen die Datenbank Datenpunkte identifiziert, die wie eine bestimmte Eingabe sind, und Anomalieerkennung, bei der es Datenpunkte erkennen kann, die erheblich von der Norm abweichen.

Darüber hinaus sind Vektordatenbanken gut geeignet, um zeitliche Daten oder zeitgestempelte Daten zu verarbeiten. Diese Art von Daten liefert Informationen über ‘was’ passiert ist und wann es passiert ist, in der Reihenfolge und in Bezug auf alle anderen Ereignisse innerhalb eines gegebenen IT-Systems. Diese Fähigkeit, zeitliche Daten zu verarbeiten und zu analysieren, macht Vektordatenbanken besonders nützlich für Anwendungen, die ein Verständnis von Ereignissen im Laufe der Zeit erfordern.

Vorteile der Vektordatenbank für ML und KI:

- High-Dimensional Search: Vektordatenbanken zeichnen sich durch ihre hervorragende Leistung bei der Verwaltung und dem Abruf von hochdimensionalen Daten aus, die häufig in KI- und ML-Applikationen generiert werden.
- Skalierbarkeit: Effiziente Skalierung zur Verarbeitung großer Datenmengen, um das Wachstum und die Erweiterung von KI- und ML-Projekten zu unterstützen
- Flexibilität: Vector Datenbanken bieten ein hohes Maß an Flexibilität und ermöglichen die Unterbringung unterschiedlicher Datentypen und Strukturen.
- Performance: Sie ermöglichen hochperformantes Datenmanagement und -Abruf, der für die Geschwindigkeit und Effizienz von KI- und ML-Operationen entscheidend ist.
- Anpassbare Indizierung: Vector-Datenbanken bieten anpassbare Indizierungsoptionen, die eine optimierte Datenorganisation und optimierte Datenabfrage je nach Bedarf ermöglichen.

Vektordatenbanken und Anwendungsfälle.

In diesem Abschnitt werden unterschiedliche Vektordatenbanken und deren Anwendungsfalldetails beschrieben.

Faiss und scann

Sie sind Bibliotheken, die als wichtige Werkzeuge im Bereich der Vektorsuche dienen. Diese Bibliotheken bieten Funktionen, die entscheidend für das Management und die Suche durch Vektordaten sind und so wertvolle Ressourcen in diesem speziellen Bereich des Datenmanagements machen.

Elasticsearch

Es ist eine weit verbreitete Such-und Analytics-Engine, hat vor kurzem integriert Vektor-Suchfunktionen. Diese neue Funktion verbessert die Funktionalität, sodass Vektordaten effektiver verarbeitet und durchsucht werden können.

Pinecon

Es handelt sich um eine robuste Vektordatenbank mit einzigartigen Funktionen. Es unterstützt sowohl dichte als auch spärliche Vektoren in seiner Indexierungsfunktionalität, was seine Flexibilität und Anpassungsfähigkeit erhöht. Eine seiner Stärken liegt in der Kombination herkömmlicher Suchmethoden mit KI-basierter Dense-Vector-Suche – so entsteht ein hybrider Suchansatz, der das Beste aus beiden Welten vereint.

Pinecone basiert hauptsächlich auf der Cloud und wurde für Applikationen für Machine Learning entwickelt. Es lässt sich gut in eine Vielzahl von Plattformen wie GCP, AWS, Open AI, GPT-3 integrieren. GPT-3.5, GPT-4, Catgut Plus, Elasticsearch, Haystack, und vieles mehr. Beachten Sie, dass Pinecone eine geschlossene Plattform ist und als Software-als-Service-Angebot (SaaS) erhältlich ist.

Aufgrund seiner fortschrittlichen Fähigkeiten ist Pinecon besonders gut für die Cybersecurity-Branche geeignet, wo seine hochdimensionalen Such- und Hybridsuchfunktionen effektiv genutzt werden können, um Bedrohungen zu erkennen und darauf zu reagieren.

Chroma

Es ist eine Vektor-Datenbank, die eine Core-API mit vier primären Funktionen hat, von denen eine einen in-Memory-Dokument-Vektor-Speicher enthält. Darüber hinaus nutzt es die Face Transformers-Bibliothek, um Dokumente zu vektorisieren und damit seine Funktionalität und Vielseitigkeit zu verbessern.
Chroma ist sowohl für den Betrieb in der Cloud als auch vor Ort konzipiert und bietet Flexibilität basierend auf den Benutzeranforderungen. Insbesondere bei Audio-bezogenen Anwendungen ist es eine hervorragende Wahl für Audio-basierte Suchmaschinen, Musikempfehlungen und andere Audio-bezogene Anwendungsfälle.

Weaviate

Es ist eine vielseitige Vektor-Datenbank, die es Benutzern ermöglicht, ihre Inhalte mit Hilfe der integrierten Module oder benutzerdefinierten Module zu vektorisieren und bietet Flexibilität auf der Grundlage spezifischer Anforderungen. Es bietet sowohl vollständig gemanagte als auch selbst gehostete Lösungen, die eine Vielzahl von Implementierungseinstellungen berücksichtigen.

Eine der wichtigsten Funktionen von Weaviate ist die Möglichkeit, sowohl Vektoren als auch Objekte zu speichern und so die Funktionen zur Datenverarbeitung zu verbessern. Es wird häufig für eine Reihe von Anwendungen verwendet, einschließlich semantischer Suche und Datenklassifizierung in ERP-Systemen. Im E-Commerce-Bereich werden die Suchmaschinen von IT-Abteilungen unterstützt. Weaviate wird auch für die Bildsuche, Anomalieerkennung, automatisierte Datenharmonisierung und Cybersecurity Threat Analysis verwendet und zeigt seine Vielseitigkeit in mehreren Domänen.

Redis

Redis ist eine leistungsstarke Vektordatenbank, die für ihren schnellen in-Memory-Speicher bekannt ist und eine niedrige Latenz für Lese- und Schreibvorgänge bietet. Dies macht es zu einer ausgezeichneten Wahl für Empfehlungssysteme, Suchmaschinen und Datenanalyseanwendungen, die einen schnellen Datenzugriff

benötigen.

Redis unterstützt verschiedene Datenstrukturen für Vektoren, einschließlich Listen, Sets und sortierte Sätze. Es bietet auch Vektoroperationen wie das Berechnen von Abständen zwischen Vektoren oder das Suchen von Kreuzungen und Verbindungen. Diese Funktionen sind besonders nützlich für Systeme zur Ähnlichkeitssuche, für Clustering und für inhaltsbasierte Empfehlungssysteme.

Redis zeichnet sich durch Skalierbarkeit und Verfügbarkeit bei Workloads mit hohem Durchsatz aus und bietet Datenreplikation. Sie lässt sich auch gut in andere Datentypen integrieren, darunter herkömmliche relationale Datenbanken (RDBMS).
Redis enthält eine Funktion zum Veröffentlichen/Abonnieren (Pub/Sub) für Echtzeit-Updates, die für die Verwaltung von Echtzeit-Vektoren von Vorteil ist. Darüber hinaus ist Redis leicht und einfach zu bedienen, was es zu einer benutzerfreundlichen Lösung für die Verwaltung von Vektordaten macht.

Milvus

Diese vielseitige Vektordatenbank bietet eine API wie ein Dokumentspeicher, ähnlich wie MongoDB. Sie zeichnet sich insbesondere durch die Unterstützung einer Vielzahl von Datentypen aus und wird damit zu einer beliebten Wahl in den Bereichen Data Science und Machine Learning.

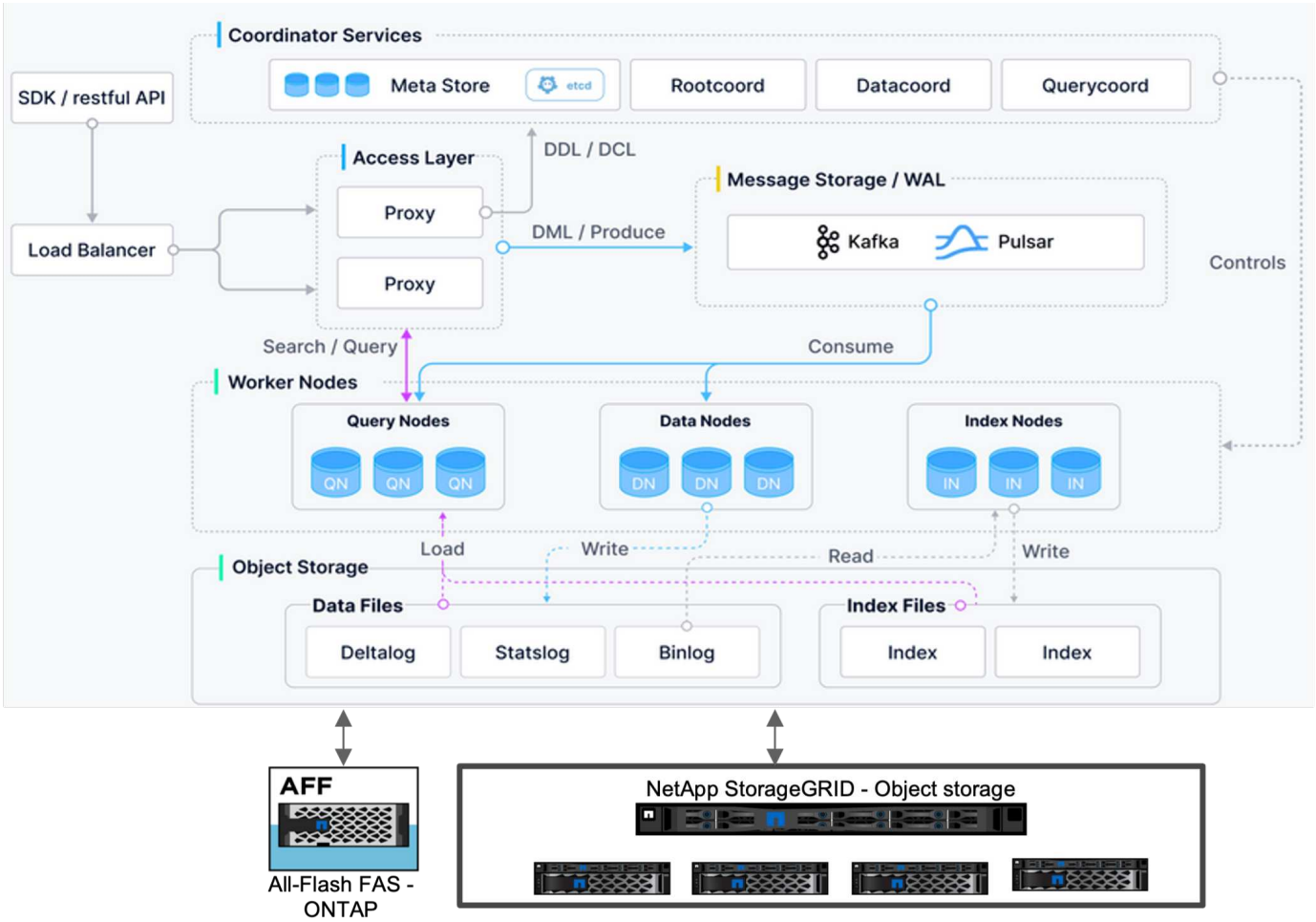
Eine der einzigartigen Funktionen von Milvus ist seine Multi-Vektorisierung, die es Benutzern ermöglicht, zur Laufzeit den für die Suche zu verwendenden Vektortyp anzugeben. Darüber hinaus nutzt es Knowwhere, eine Bibliothek, die auf anderen Bibliotheken wie Faiss sitzt, um die Kommunikation zwischen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

Dank seiner Kompatibilität mit PyTorch und TensorFlow bietet Milvus auch eine nahtlose Integration in Workflows für maschinelles Lernen. Dies macht es zu einem hervorragenden Werkzeug für eine Reihe von Anwendungen, einschließlich E-Commerce, Bild-und Videoanalyse, Objekterkennung, Bildähnlichkeitssuche und Content-basierte Bildabfrage. Im Bereich der natürlichen Sprachverarbeitung wird Milvus für die Dokumentenclustering-, semantische Suche- und Question-Answering-Systeme eingesetzt.

Für diese Lösung haben wir uns bei der Lösungsvalidierung für milvus entschieden. Für die Performance haben wir sowohl milvus als auch postgres(pgvecto.rs) verwendet.

Warum haben wir uns bei dieser Lösung für milvus entschieden?

- Open-Source: Milvus ist eine Open-Source-Vektordatenbank, die die Entwicklung und Verbesserungen durch die Community fördert.
- KI-Integration: Die Einbindung von Ähnlichkeitssuche und KI-Anwendungen optimiert die Funktionalität der Vektordatenbank.
- Handhabung großer Volumen: Milvus kann mehr als eine Milliarde Embedding-Vektoren speichern, indizieren und verwalten, die von Deep Neural Networks (DNN)- und Machine Learning (ML)-Modellen generiert werden.
- Benutzerfreundlich: Die Einrichtung dauert weniger als eine Minute. Milvus bietet auch SDKs für verschiedene Programmiersprachen an.
- Geschwindigkeit: Es bietet extrem schnelle Abrufgeschwindigkeiten, bis zu 10-mal schneller als einige Alternativen.
- Skalierbarkeit und Verfügbarkeit: Milvus ist äußerst skalierbar und bietet Optionen für eine bedarfsgerechte Skalierung.
- Funktionsreich: Es unterstützt verschiedene Datentypen, Attributfilter, UDF-Unterstützung (User-Defined Function), konfigurierbare Konsistenzstufen und Reisezeiten und ist damit ein vielseitiges Werkzeug für verschiedene Anwendungen.



In diesem Abschnitt finden Sie höhere Hebelkomponenten und Dienste, die in der Milvus-Architektur verwendet werden.
* Zugriffsebene – Es besteht aus einer Gruppe von zustandslosen Proxys und dient als die Front-Schicht des Systems und Endpunkt für Benutzer.
* Koordinatorendienst – es ordnet die Aufgaben den Arbeiterknoten zu und fungiert als Gehirn eines Systems. Es hat drei Koordinatortypen: Root-Koord, Data-Coord und Query-Coord.
* Worker Nodes : Es folgt der Anweisung vom Coordinator Service und Execute User getriggert DML/DDL commands.it hat drei Arten von Worker-Knoten wie Abfrage-Knoten, Daten-Knoten und Index-Knoten.
* Storage: Sie ist für die Datenpersistenz verantwortlich. Sie umfasst Meta-Storage, Log-Broker und Objekt-Storage. NetApp-Storage wie ONTAP und StorageGRID bietet Milvus Objekt-Storage und dateibasierten Storage für Kundendaten sowie Vektordatenbankdaten.

Technologieanforderungen

Technologieanforderungen

Die unten aufgeführten Hardware- und Softwarekonfigurationen wurden mit Ausnahme der Performance für die meisten in diesem Dokument durchgeführten Validierungen verwendet. Diese Konfigurationen dienen als Richtlinie zur Einrichtung Ihrer Umgebung. Bitte beachten Sie jedoch, dass die einzelnen Komponenten je nach Kundenanforderungen variieren können.

Hardwareanforderungen

Trennt	Details
NetApp AFF Storage-Array HA-Paar	* A800 * ONTAP 9.14.1 * 48 X 3,49 TB SSD-NVM * Zwei flexible Gruppen-Volumes: Metadaten und Daten. * Metadaten NFS Volume hat 12 x 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 physikalischer Speicher * 1 x 100-GbE-Netzwerk-Port
Netzwerkbetrieb	100 GbE
StorageGRID	* 1 x SG100, 3 x SGF6024 * 3 x 24 x 7,68 TB

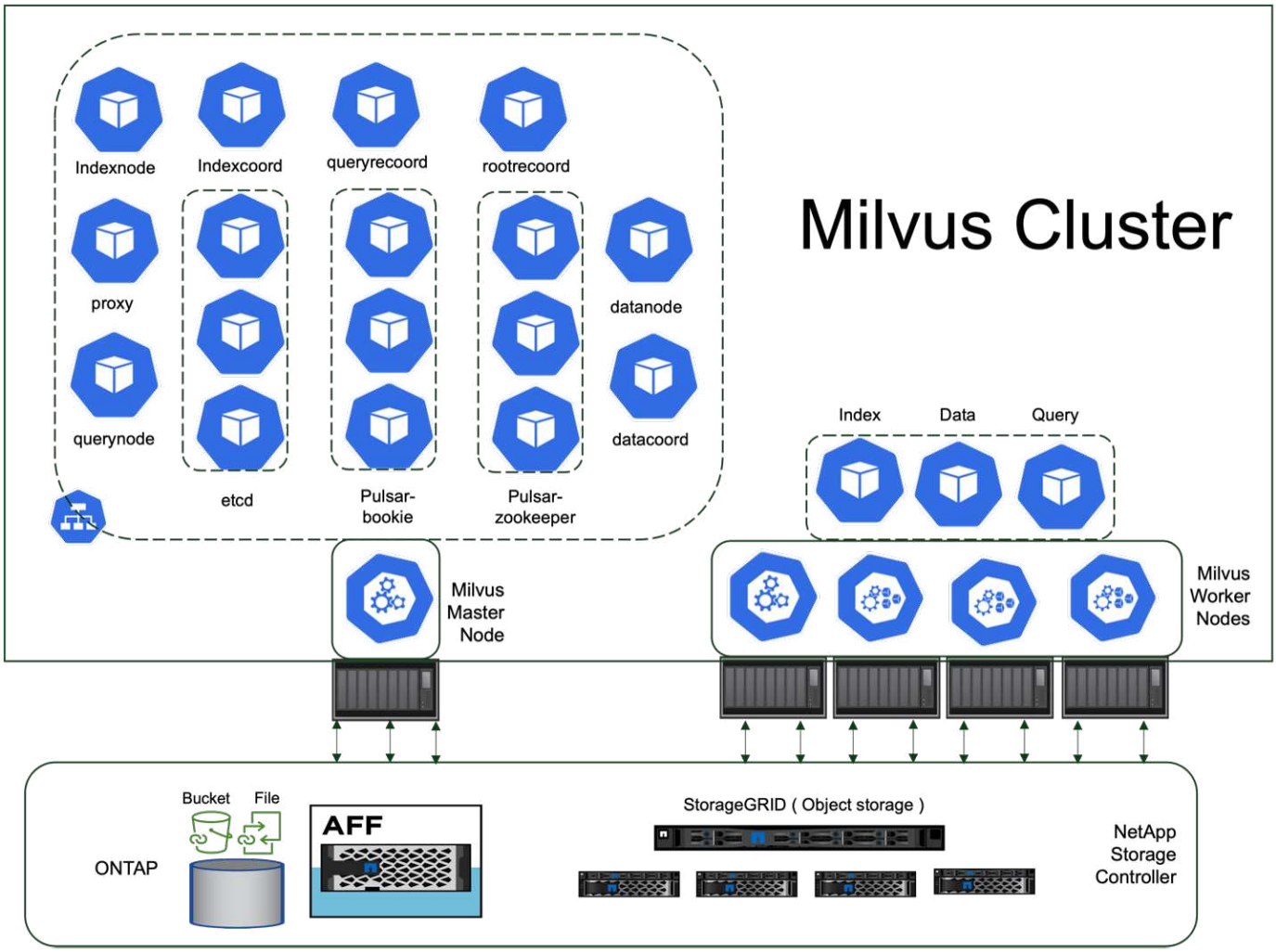
Softwareanforderungen

Software	Details
Milvus-Cluster	* DIAGRAMM - milvus-4.1.11. * APP-Version – 2.3.4 * Abhängige Bündel wie Buchhalter, Zookeeper, Pulsar, etc., Proxy, querynode, worker
Kubernetes	* K8s Cluster mit 5 Nodes * 1 Master-Knoten und 4 Worker-Knoten * Version – 1.7.2
Python	*3.10.12.

Bereitstellungsverfahren

Bereitstellungsverfahren

In diesem Abschnitt zur Implementierung haben wir wie unten gezeigt die milvus Vektor-Datenbank mit Kubernetes für die Lab-Einrichtung verwendet.



Der NetApp-Speicher stellt den Speicher für das Cluster zur Verfügung, um Kundendaten und milvus-Clusterdaten zu behalten.

Einrichtung von NetApp Storage – ONTAP

- Initialisierung des Speichersystems
- Erstellung von Storage Virtual Machines (SVM)
- Zuweisung logischer Netzwerkschnittstellen
- NFS-, S3-Konfiguration und -Lizenzierung

Folgen Sie den Schritten unten für NFS (Network File System):

1. Erstellen Sie ein FlexGroup-Volume für NFSv4. In unserer Einrichtung für diese Validierung haben wir 48 SSDs verwendet, 1 SSD dediziert 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 (K8s) Node-Netzwerk besitzt. Wenn diese Berechtigungen nicht vorhanden sind, erteilen Sie Lese-/Schreibberechtigungen (rw) für das K8s-Knoten-Netzwerk.
2. Erstellen Sie auf allen K8s-Nodes einen Ordner und mounten Sie das FlexGroup-Volume über eine logische Schnittstelle (Logical Interface, LIF) auf jedem K8s-Node in diesem Ordner.

Folgen Sie den Schritten unten für NAS S3 (Network Attached Storage Simple Storage Service):

1. Erstellen Sie ein FlexGroup Volume für NFS.
2. Richten Sie einen Object-Store-Server mit aktiviertem HTTP ein, und der Admin-Status wird mit dem Befehl „vserver Object-Store-Server create“ auf „up“ gesetzt. Sie haben die Möglichkeit, HTTPS zu aktivieren und einen benutzerdefinierten Listener-Port festzulegen.
3. Erstellen Sie mit dem Befehl „vserver object-Store-Server user create -user <username>“ einen Object-Store-Server-Benutzer.
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 <username>". Diese Schlüssel werden jedoch während der Benutzererstellung bereitgestellt oder können über REST-API-Aufrufe abgerufen werden.
5. Erstellen Sie eine Object-Store-Server-Gruppe mit dem in Schritt 2 erstellten Benutzer und gewähren Sie Zugriff. In diesem Beispiel haben wir „FullAccess“ bereitgestellt.
6. Erstellen Sie einen NAS-Bucket, indem Sie seinen Typ auf „nas“ setzen und den Pfad zum NFSv3 Volume bereitstellen. Es ist auch möglich, einen S3-Bucket zu diesem Zweck zu verwenden.

Einrichtung von NetApp Storage – StorageGRID

1. Installieren Sie die StorageGRID-Software.
2. Erstellen von Mandanten und Buckets
3. Benutzer mit der erforderlichen Berechtigung erstellen.

Bitte überprüfen Sie mehr Details in https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

Lösungsüberblick

Wir haben eine umfassende Lösungsvalidierung durchgeführt, die sich auf fünf Kernbereiche konzentriert. Die Details werden im Folgenden erläutert. Jeder Abschnitt geht auf die Herausforderungen von Kunden, die von NetApp bereitgestellten Lösungen und die daraus folgenden Vorteile für den Kunden ein.

1. "Milvus Cluster-Einrichtung mit Kubernetes vor Ort"
   Für Kunden stellt es eine Herausforderung dar, nach Bedarf unabhängig nach Storage- und Computing-Ressourcen, effektivem Infrastrukturmanagement und Datenmanagement zu skalieren. In diesem Abschnitt wird der Prozess der Installation eines Milvus Clusters auf Kubernetes beschrieben, der einen NetApp Storage Controller für Cluster- und Kundendaten verwendet.
2. "Milvus mit Amazon FSxN für NetApp ONTAP – Datei- und Objektdualität"
   In diesem Abschnitt, Warum wir brauchen, um Vektor-Datenbank in der Cloud sowie Schritte zur Bereitstellung von Vektor-Datenbank (milvus Standalone) in Amazon FSxN für NetApp ONTAP in Docker-Containern bereitzustellen.
3. "Sicherung von Vector Datenbanken mit NetApp SnapCenter."
   In diesem Abschnitt befassen wir uns eingehend damit, wie SnapCenter die Vektordatenbank-Daten und Milvus-Daten in ONTAP schützt. Für dieses Beispiel wurde ein 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.
4. "Disaster Recovery mit NetApp SnapMirror"
   In diesem Abschnitt wird die Bedeutung von Disaster Recovery (DR) für Vektordatenbanken erläutert und es wird erläutert, wie das Disaster Recovery-Produkt von NetApp snapmirror die DR-Lösung für die Vektordatenbank bereitstellt.
5. "Performance-Validierung"

In diesem Abschnitt möchten wir uns mit der Performance-Validierung von Vektordatenbanken wie Milvus und pgvecto.rs beschäftigen. Dabei konzentrieren wir uns auf die Speicherleistungsmerkmale wie I/O-Profil und das Verhalten des NetApp-Speichercontrollers zur Unterstützung von RAG- und Inferenzarbeitslasten innerhalb des LLM-Lebenszyklus. Wir bewerten und ermitteln jegliche Performance-Unterscheidungsmerkmale, wenn diese Datenbanken mit der ONTAP Storage-Lösung kombiniert werden. Unsere Analyse basiert auf wichtigen Leistungsindikatoren, wie der Anzahl der pro Sekunde verarbeiteten Abfragen (QPS).

Milvus Cluster Setup mit Kubernetes vor Ort

Milvus Cluster-Einrichtung mit Kubernetes vor Ort

Kundenherausforderungen bei der unabhängigen Skalierung nach Storage- und Computing-Ressourcen, effektivem Infrastrukturmanagement und Datenmanagement,
Kubernetes- und Vektordatenbanken bilden zusammen eine leistungsstarke, skalierbare Lösung für das Management großer Datenvorgänge. Kubernetes optimiert Ressourcen und managt Container. Vektordatenbanken können hochdimensionale Daten und Ähnlichkeitssuchen effizient verarbeiten. Diese Kombination ermöglicht die schnelle Verarbeitung komplexer Abfragen auf großen Datensätzen und lässt sich nahtlos an wachsende Datenvolumen skalieren. Dadurch eignet sie sich ideal für Big-Data-Applikationen und KI-Workloads.

1. In diesem Abschnitt wird der Prozess der Installation eines Milvus Clusters auf Kubernetes beschrieben, der einen NetApp Storage Controller für Cluster- und Kundendaten verwendet.
2. Für die Installation eines Milvus-Clusters sind Persistent Volumes (PVs) erforderlich, um Daten aus verschiedenen Milvus-Clusterkomponenten zu speichern. Zu diesen Komponenten gehören etcd (drei Instanzen), Pulsar-bookie-Journal (drei Instanzen), Pulsar-bookie-Ledgers (drei Instanzen) und Pulsar-Zookeeper-Daten (drei Instanzen).
3. Wir haben aus NetApp ONTAP ein einzelnes NFS Volume erstellt und 12 persistente Volumes eingerichtet, jedes mit 250 GB Storage. Die Storage-Größe kann je nach Cluster-Größe variieren; wir haben z. B. einen weiteren Cluster, in dem jeder PV 50 GB hat. Bitte beachten Sie unten eine der PV YAML-Dateien für weitere Details; wir hatten insgesamt 12 solche Dateien. In jeder Datei wird der storageClassName auf 'default' gesetzt, und der Speicher und der 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 Persistent Volumes zu erstellen, und überprüfen Sie dann ihre 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 Objekt-Storage-Lösungen wie Minio, Azure Blob und S3. In diesem Leitfaden verwenden wir S3. Die folgenden Schritte gelten sowohl für ONTAP S3 als auch für StorageGRID Objektspeicher. Wir verwenden Helm zur Bereitstellung des Milvus Clusters. 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 'Standard' gesetzt ist, einschließlich derjenigen für das Protokoll, etc., Zookeeper und Buchhalter.
7. Deaktivieren Sie Minio im Abschnitt Minio.
8. Erstellen Sie einen NAS-Bucket aus ONTAP oder StorageGRID-Objekt-Storage und fügen Sie sie mit den Zugangsdaten für den Objekt-Storage in ein externes S3-System 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 das PersistentVolumeClaim (PVC) keine bereits vorhandenen Ressourcen hat.

```
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 (VES).

```
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>
```

Stellen Sie sicher, dass der PODs-Status ‘Running’ lautet und wie erwartet funktioniert

13. Testen Sie das Schreiben und Lesen von Daten in Milvus und NetApp Objekt-Storage.
    - 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 mit 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 oben genannten Validierung bietet die Integration von Kubernetes in eine Vektordatenbank, wie die Implementierung eines Milvus Clusters auf Kubernetes über einen NetApp Storage-Controller demonstriert, Kunden eine robuste, skalierbare und effiziente Lösung für das Management großer Datenoperationen. Diese Einrichtung ermöglicht es Kunden, hochdimensionale Daten schnell und effizient zu verarbeiten und komplexe Abfragen auszuführen. Dadurch ist sie die ideale Lösung für Big-Data-Applikationen und KI-Workloads. Der Einsatz von persistenten Volumes (PVS) für verschiedene Cluster-Komponenten stellt zusammen mit der Erstellung eines einzigen NFS-Volumes aus NetApp ONTAP eine optimale Ressourcenauslastung und ein optimales

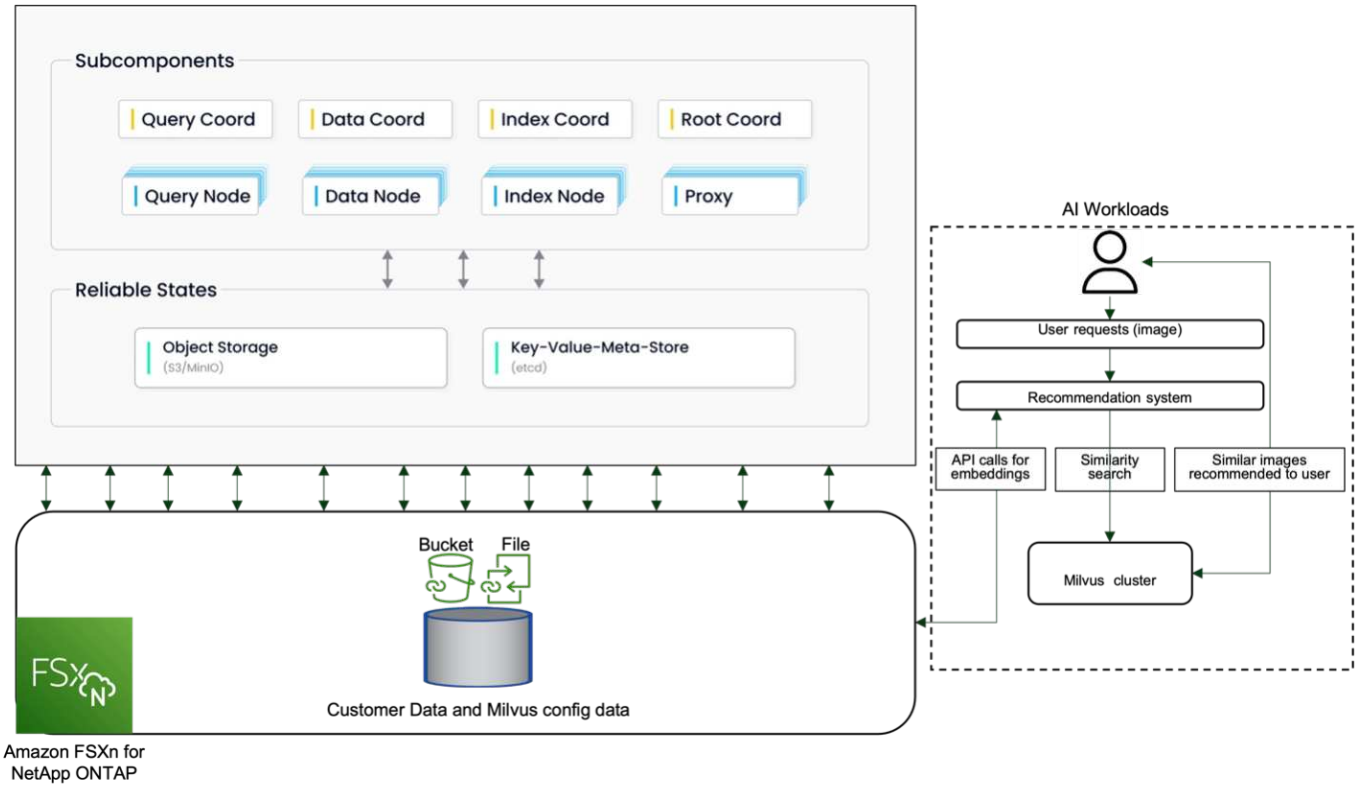
Datenmanagement sicher. Der Prozess der Überprüfung des Status von PersistentVolumeClaims (PVCs) und Pods sowie der Prüfung von Daten Schreiben und Lesen bietet Kunden die Sicherheit zuverlässiger und konsistenter Datenoperationen. Die Nutzung von ONTAP oder StorageGRID Objekt-Storage für Kundendaten verbessert die Verfügbarkeit und Sicherheit von Daten noch weiter. Kunden erhalten durch diese Einrichtung eine robuste und hochperformante Datenmanagement-Lösung, die sich nahtlos an ihre steigenden Datenanforderungen anpassen lässt.

Milvus mit Amazon FSxN für NetApp ONTAP - Datei und Objekt Dualität

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

In diesem Abschnitt, Warum wir Vektordatenbank in der Cloud bereitstellen müssen, sowie Schritte zur Bereitstellung von Vektordatenbank (milvus Standalone) in Amazon FSxN für NetApp ONTAP in Docker Containern.

Die Bereitstellung einer Vektordatenbank in der Cloud bietet einige signifikante Vorteile, insbesondere für Anwendungen, die die Verarbeitung von hochdimensionalen Daten und die Ausführung von Ähnlichkeitssuchen erfordern. Erstens bietet die Cloud-basierte Implementierung Skalierbarkeit, die eine einfache Anpassung der Ressourcen an die wachsenden Datenmengen und die Abfragelasten ermöglicht. Auf diese Weise wird sichergestellt, dass die Datenbank den gestiegenen Bedarf effizient bewältigen kann und gleichzeitig eine hohe Performance erhält. Zweitens bietet Cloud-Bereitstellung Hochverfügbarkeit und Disaster Recovery, da Daten an verschiedenen geografischen Standorten repliziert werden können, um das Risiko von Datenverlusten zu minimieren und einen kontinuierlichen Service auch bei unerwarteten Ereignissen sicherzustellen. Drittens bietet sie Kosteneffizienz, da Sie nur für die tatsächlich genutzten Ressourcen zahlen und je nach Bedarf vertikal skalieren können. Dadurch vermeiden Sie erhebliche Vorabinvestitionen in Hardware. Schließlich kann die Bereitstellung einer Vektordatenbank in der Cloud die Zusammenarbeit verbessern, da Daten von überall aus abgerufen und freigegeben werden können. Dies erleichtert Team-basiertes Arbeiten und datengestützte Entscheidungsfindung.
Bitte prüfen Sie die Architektur des eigenständigen milvus mit Amazon FSxN für NetApp ONTAP, die bei dieser Validierung verwendet wird.



1. Erstellen Sie eine Amazon FSxN for NetApp ONTAP-Instanz und notieren Sie sich die Details zu VPC, VPC-Sicherheitsgruppen und dem Subnetz. Diese Informationen sind erforderlich, wenn eine EC2-Instanz erstellt wird. 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-Instanz, um sicherzustellen, dass VPC, Sicherheitsgruppen und Subnetz mit denen der Amazon FSxN für NetApp ONTAP-Instanz übereinstimmen.
3. Installieren Sie nfs-common mit dem Befehl 'apt-get install nfs-common' und aktualisieren Sie die Paketinformationen mit 'udo apt-get Update'.
4. Erstellen Sie einen Mount-Ordner, und mounten Sie den Amazon FSxN für NetApp ONTAP darauf.

```
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-Bereitstellungspunkt dem entsprechenden Milvus-Containerpfad zu, insbesondere in etcd, minio und Standalone.Check "Anhang D: docker-compose.yml" Für Details über Änderungen in yml
8. Überprüfen Sie die gemounteten 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 FSxN for NetApp ONTAP zu validieren, haben wir das Python Milvus SDK und ein Beispielprogramm von PyMilvus verwendet. Installieren Sie die benötigten Pakete mit 'apt-get install python3-numpy python3-Pip' und installieren Sie PyMilvus mit 'pip3 install pymilvus'.
12. Validieren der Schreib- und Lesevorgänge von Amazon FSxN 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 auf NFS-Daten zugreifen möchte, die in der Vektordatenbank über das S3-Protokoll für KI-Workloads getestet wurden (lesen), 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 zu Beginn 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#
```

In diesem Abschnitt wird effektiv gezeigt, wie Kunden mithilfe von Amazon NetApp FSxN für NetApp ONTAP Storage eine eigenständige Einrichtung von Milvus in Docker Containern implementieren und

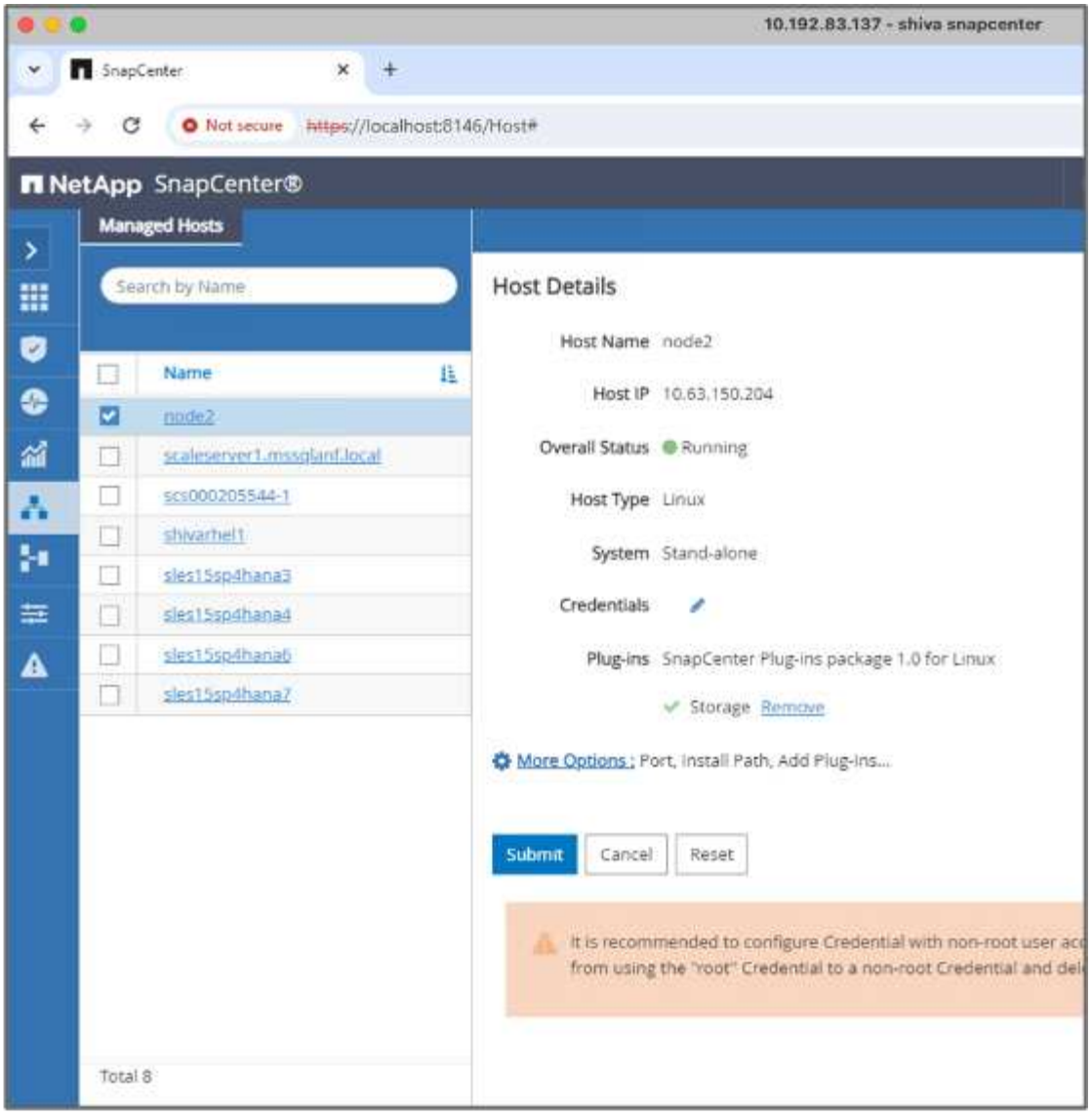
betreiben können. Mit dieser Einrichtung können Kunden die Leistungsfähigkeit von Vektordatenbanken für die Verarbeitung von hochdimensionalen Daten und die Ausführung komplexer Abfragen nutzen – all dies in der skalierbaren und effizienten Umgebung von Docker Containern. Durch Erstellung einer Amazon FSxN for NetApp ONTAP Instanz und Anpassung der EC2-Instanz können Kunden für eine optimale Ressourcenauslastung und Datenmanagement sorgen. Die erfolgreiche Validierung von Datenschreibvorgängen und -Lesevorgängen von FSxN in der Vektordatenbank bietet Kunden die Sicherheit zuverlässiger und konsistenter Datenoperationen. Darüber hinaus bietet die Möglichkeit, Daten aus KI-Workloads über das S3-Protokoll aufzulisten (zu lesen), eine verbesserte Datenverfügbarkeit. Durch diesen umfassenden Prozess erhalten Kunden eine robuste und effiziente Lösung für das Management ihrer umfangreichen Datenabläufe und dabei Nutzung der Funktionen von Amazon FSxN für NetApp ONTAP.

Schutz von Vektordatenbanken mithilfe von SnapCenter

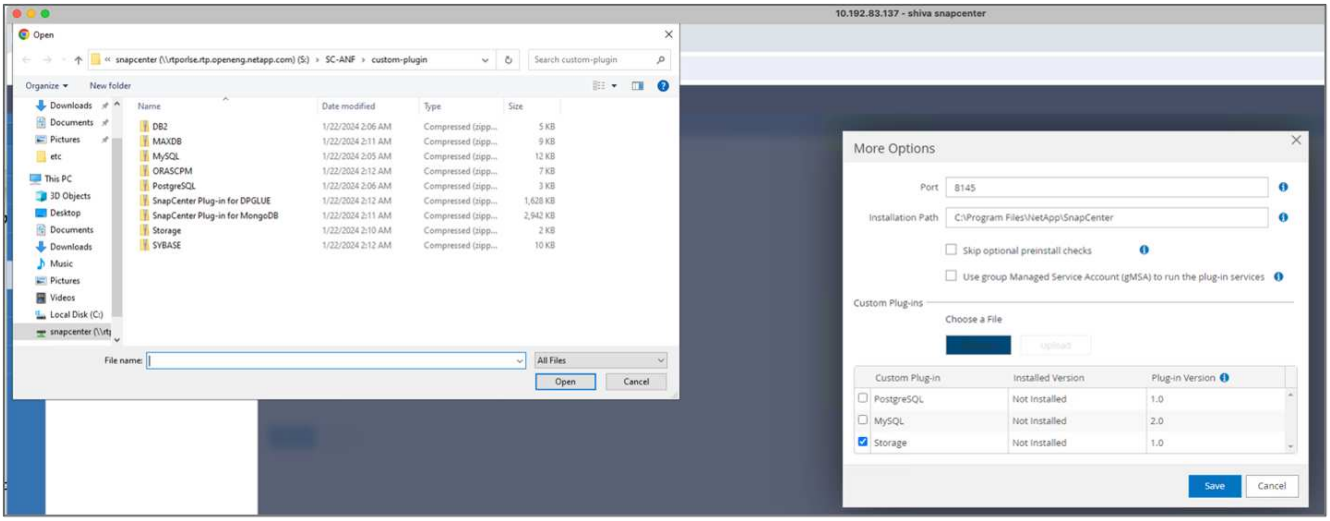
Sicherung von Vector Datenbanken mit NetApp SnapCenter.

In der Filmproduktionsbranche verfügen Kunden beispielsweise oft über kritische eingebettete Daten wie Video- und Audiodateien. Der Verlust dieser Daten kann aufgrund von Problemen wie Festplattenausfällen erhebliche Auswirkungen auf ihren Betrieb haben und millionenschwere Projekte gefährden. Wir sind auf Fälle gestoßen, in denen unschätzbare Inhalte verloren gegangen sind, was zu erheblichen Störungen und finanziellen Verlusten führte. Daher ist es in dieser Branche von höchster Bedeutung, die Sicherheit und Integrität dieser wichtigen Daten zu gewährleisten.
In diesem Abschnitt befassen wir uns eingehend damit, wie SnapCenter die Vektordatenbank-Daten und Milvus-Daten in ONTAP schützt. Für dieses Beispiel wurde ein 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 prüfen Sie die "Hier" Des SnapCenter Backup Workflows

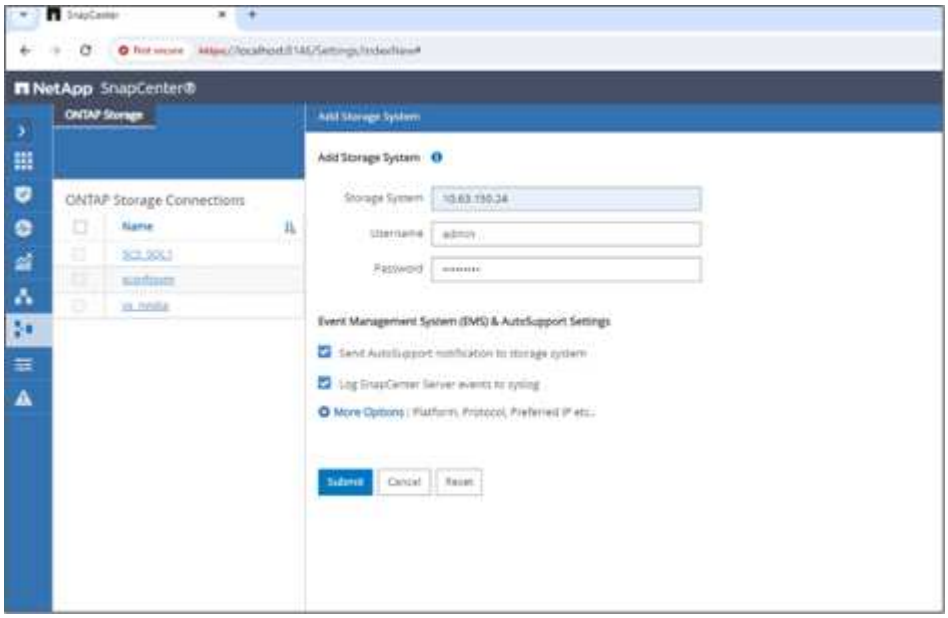
1. Richten Sie den Host ein, der zur Ausführung von SnapCenter-Befehlen verwendet wird.



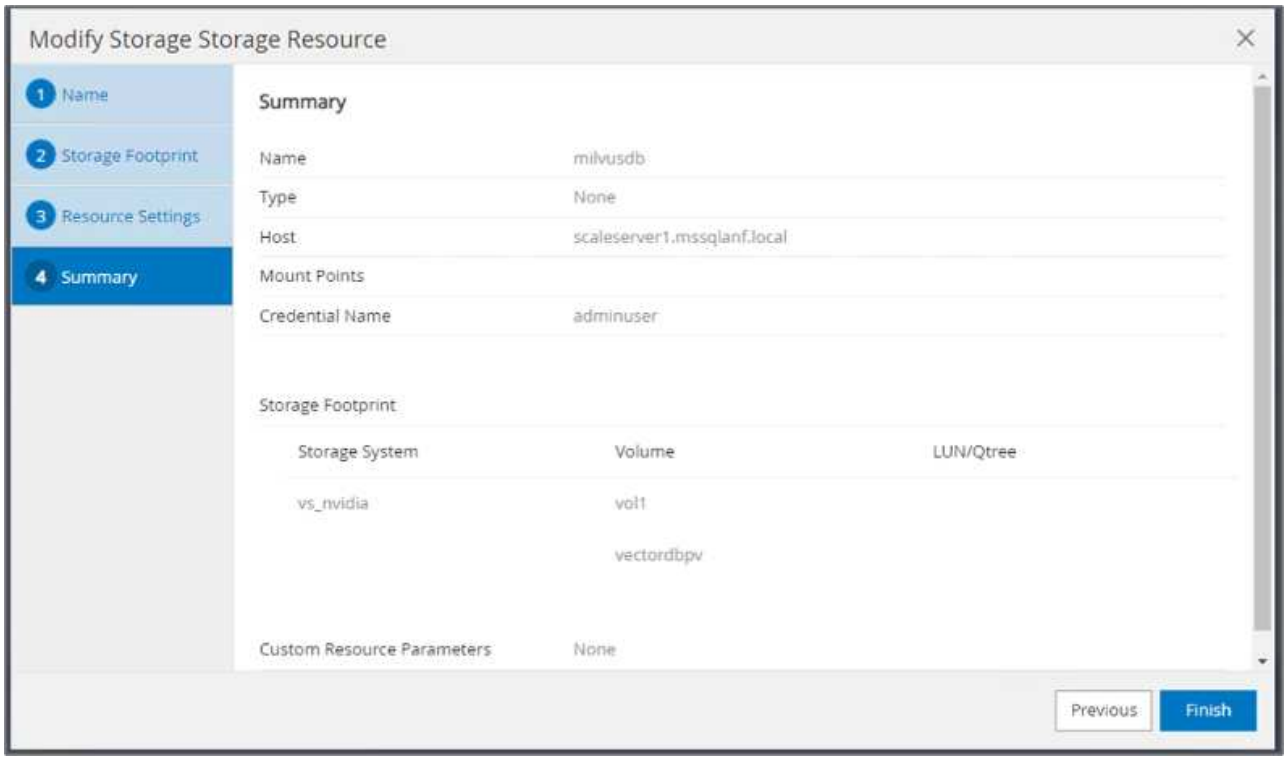
2. Installieren und konfigurieren Sie das Speicher-Plug-in. Wählen Sie aus dem hinzugefügten Host "More Options". Navigieren Sie zum heruntergeladenen Speicher-Plug-in, und wählen Sie es aus dem aus "NetApp Automation Store". Installieren Sie das Plugin und speichern Sie die Konfiguration.



3. Storage-System und Volume einrichten: Fügen Sie das Storage-System unter „Storage System“ hinzu und wählen Sie die SVM (Storage Virtual Machine) aus. In diesem Beispiel haben wir uns für „vs_nvidia“ entschieden.



4. Erstellen Sie eine Ressource für die Vektordatenbank, die eine Sicherungsrichtlinie und einen benutzerdefinierten Snapshot-Namen enthält.
   - Aktivieren Sie Consistency Group Backup mit Standardwerten und aktivieren Sie SnapCenter ohne Dateisystemkonsistenz.
   - Wählen Sie im Abschnitt Storage Footprint 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 für den Schutz der Vektordatenbank, und schützen Sie die Vektordatenbankressource mithilfe der Richtlinie.



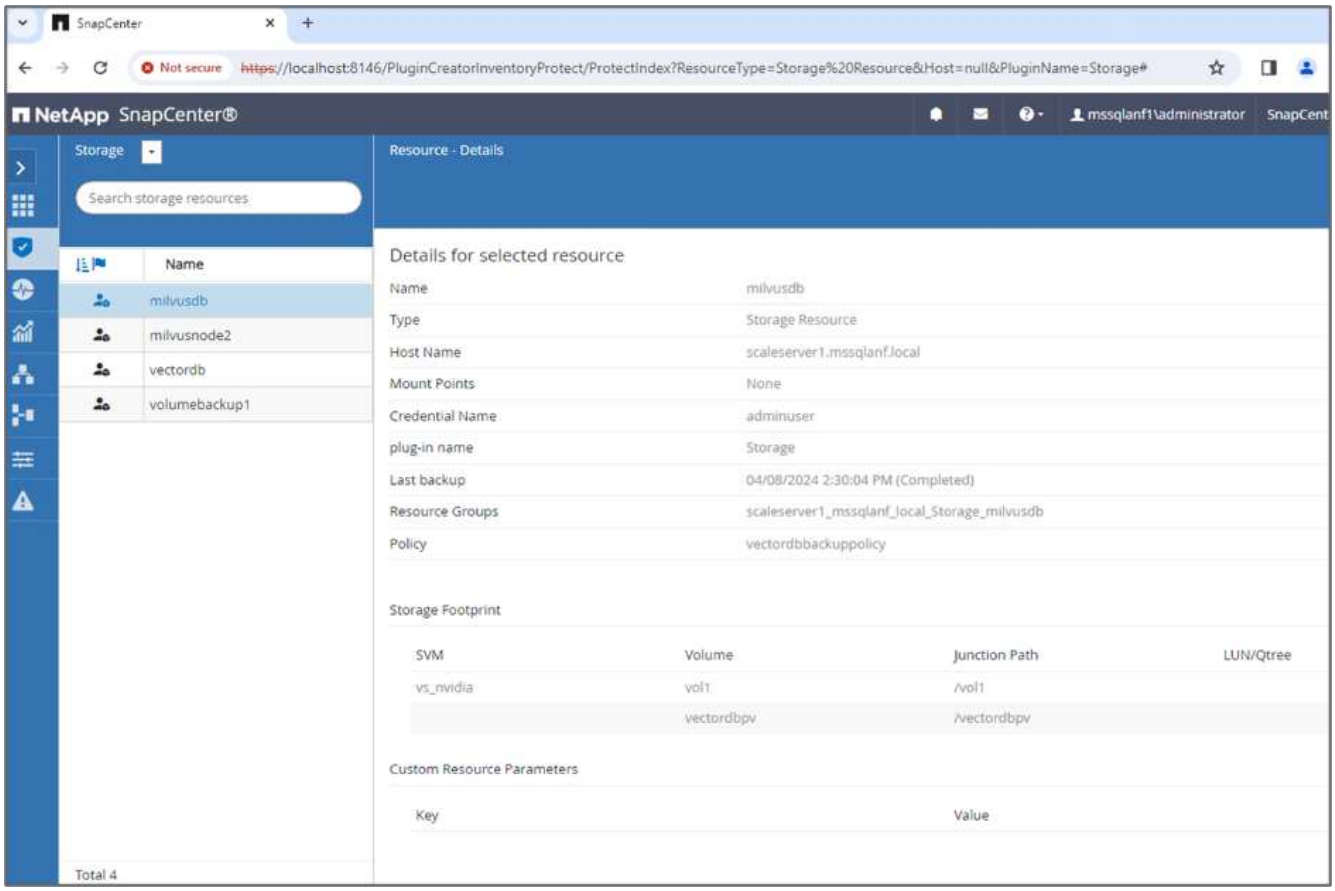
5. Fügen Sie Daten mithilfe eines Python-Skripts in den S3-NAS-Bucket ein. In unserem Fall haben wir das von Milvus bereitgestellte Backup-Skript 'prepare_Data_netapp.py' geändert und den 'sync'-Befehl ausgeführt, um die Daten vom Betriebssystem zu löschen.

```
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 der 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. Initiieren Sie ein Backup mit dem CG-Snapshot (Consistency Group) von der Ressource 'milvusdb'

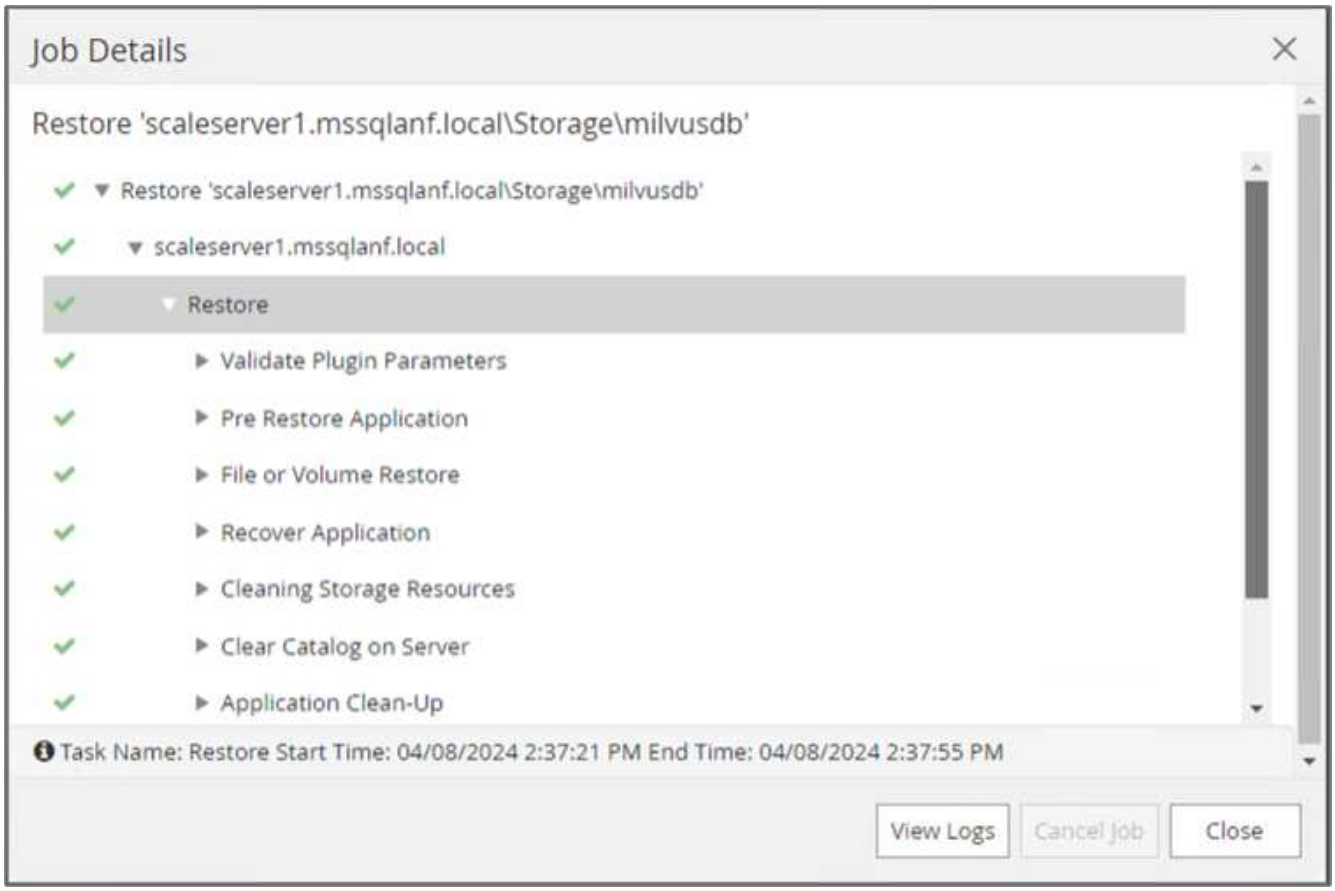


8. Zum Testen der Backup-Funktion haben wir nach dem Backup-Prozess entweder eine neue Tabelle hinzugefügt oder einige Daten aus dem NFS entfernt (S3 NAS Bucket).

   Stellen Sie sich für diesen Test ein Szenario vor, in dem nach dem Backup eine neue, unnötige oder unangemessene Sammlung erstellt wurde. In einem solchen Fall müssten wir die Vektordatenbank auf ihren Zustand zurücksetzen, bevor die neue Sammlung hinzugefügt wurde. So wurden beispielsweise 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. Vergewissern Sie sich, 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:~#
```

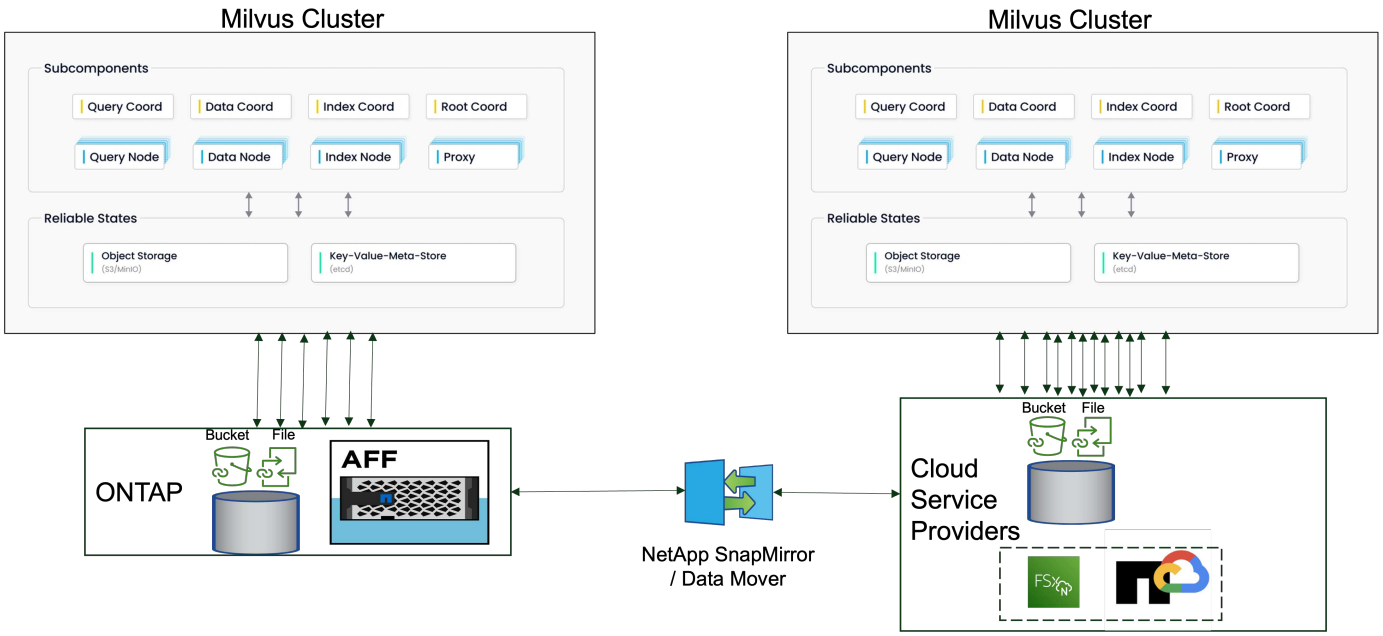
Zusammenfassend lässt sich feststellen, dass die Verwendung von NetApp SnapCenter zur Sicherung von Vektordatenbankdaten und Milvus-Daten in ONTAP erhebliche Vorteile für Kunden bietet, insbesondere in Branchen, in denen die Datenintegrität oberste Priorität hat, wie etwa der Filmproduktion. Durch die Fähigkeit von SnapCenter, konsistente Backups zu erstellen und vollständige Daten-Restores durchzuführen, wird sichergestellt, dass wichtige Daten wie integrierte Video- und Audiodateien vor Verlust durch Festplattenausfälle oder andere Probleme geschützt sind. Auf diese Weise werden nicht nur Betriebsunterbrechungen verhindert, sondern auch erhebliche finanzielle Verluste verhindert.

In diesem Abschnitt haben wir gezeigt, wie SnapCenter für den Schutz von in ONTAP gespeicherten Daten konfiguriert werden kann. Dazu gehören die Einrichtung von Hosts, die Installation und Konfiguration von Storage-Plug-ins sowie die Erstellung einer Ressource für die Vektordatenbank mit einem benutzerdefinierten Snapshot-Namen. Außerdem wurde gezeigt, wie ein Backup mit dem Snapshot der Consistency Group durchgeführt und die Daten im S3-NAS-Bucket verifiziert werden können.

Darüber hinaus haben wir ein Szenario simuliert, in dem nach dem Backup eine unnötige oder unangemessene Sammlung erstellt wurde. In solchen Fällen stellt die Möglichkeit von SnapCenter, eine vollständige Wiederherstellung aus einem früheren Snapshot durchzuführen, sicher, dass die Vektordatenbank vor dem Hinzufügen der neuen Sammlung in ihren Zustand zurückgesetzt werden kann. So wird die Integrität der Datenbank gewahrt. Diese Möglichkeit zur Wiederherstellung der Daten zu einem bestimmten Zeitpunkt ist für Kunden von unschätzbarem Wert und gibt ihnen die Gewissheit, dass ihre Daten nicht nur sicher sind, sondern auch ordnungsgemäß gepflegt werden. Somit bietet das SnapCenter-Produkt von NetApp Kunden eine robuste und zuverlässige Lösung für Datensicherung und -Management.

Disaster Recovery mit NetApp SnapMirror

Disaster Recovery mit NetApp SnapMirror



Disaster Recovery ist entscheidend für die Aufrechterhaltung der Integrität und Verfügbarkeit einer Vektordatenbank, insbesondere angesichts ihrer Rolle bei der Verwaltung von hochdimensionalen Daten und der Ausführung komplexer Ähnlichkeitssuchen. Eine gut geplante und implementierte Disaster-Recovery-Strategie stellt sicher, dass Daten im Falle von unvorhergesehenen Zwischenfällen, wie Hardwareausfällen, Naturkatastrophen oder Cyberangriffen nicht verloren gehen oder beschädigt werden. Dies gilt insbesondere für Applikationen, die auf Vektordatenbanken basieren und bei denen der Verlust oder die Beschädigung von Daten zu erheblichen Betriebsunterbrechungen und finanziellen Verlusten führen kann. Darüber hinaus sorgt ein robuster Disaster Recovery-Plan durch Minimierung der Ausfallzeiten für Business Continuity und ermöglicht eine schnelle Wiederherstellung von Services. Erreicht wird dies durch das NetApp Datenreplizierungsprodukt SnapMirror über verschiedene geografische Standorte, regelmäßige Backups und Failover-Mechanismen. Desaster Recovery ist daher nicht nur eine Schutzmaßnahme, sondern eine wichtige Komponente eines verantwortungsvollen und effizienten Vektordatenbankmanagements.

SnapMirror von NetApp ermöglicht die Datenreplizierung von einem NetApp ONTAP Storage Controller zu einem anderen, der in erster Linie für Disaster Recovery (DR) und Hybridlösungen verwendet wird. Im Kontext einer Vektordatenbank ermöglicht dieses Tool die reibungslose Transition der Daten zwischen On-Premises- und Cloud-Umgebungen. Diese Transition erfolgt ohne Datenkonvertierungen oder Refactoring von Applikationen, wodurch die Effizienz und Flexibilität des Datenmanagements über mehrere Plattformen hinweg verbessert wird.

Die NetApp Hybridlösung in einer Vektordatenbank kann weitere Vorteile bringen:

1. Skalierbarkeit: Die Hybrid-Cloud-Lösung von NetApp bietet die Möglichkeit, Ressourcen nach Bedarf zu skalieren. On-Premises-Ressourcen können für regelmäßige, vorhersehbare Workloads und Cloud-Ressourcen wie Amazon FSxN für NetApp ONTAP und Google Cloud NetApp Volume (GCNV) für Spitzenlasten oder unerwartete Workloads genutzt werden.
2. Kosteneffizienz: Das Hybrid-Cloud-Modell von NetApp ermöglicht eine Kostenoptimierung. Hierbei werden On-Premises-Ressourcen für normale Workloads genutzt und Cloud-Ressourcen nur nach Bedarf bezahlt. Dieses „Pay-as-you-go“-Modell kann mit dem Serviceangebot von NetApp Instaclustr recht kostengünstig sein. Für On-Premises- und große Cloud-Service-Provider bietet instaclustr Support und Beratung.
3. Flexibilität: Die Hybrid Cloud von NetApp gibt Ihnen die Flexibilität, den Speicherort der Daten frei zu wählen. Sie können beispielsweise komplexe Vektoroperationen vor Ort durchführen, bei denen Sie über eine leistungsstärkere Hardware und weniger intensive Vorgänge in der Cloud verfügen.

4. Business Continuity: Wenn Ihre Daten in einer NetApp Hybrid Cloud verfügbar sind, kann bei einem Ausfall die Business Continuity gewährleistet werden. Wenn Ihre On-Premises-Ressourcen betroffen sind, können Sie schnell zur Cloud wechseln. Mit NetApp SnapMirror können wir die Daten von On-Premises-Systemen in die Cloud verschieben und umgekehrt.
5. Innovation: Die Hybrid-Cloud-Lösungen von NetApp ermöglichen durch Zugriff auf hochmoderne Cloud-Services und -Technologien außerdem schnellere Innovationen. NetApp-Innovationen in der Cloud, wie Amazon FSxN für NetApp ONTAP, Azure NetApp Files und Google Cloud NetApp Volumes, sind innovative Produkte von Cloud-Service-Providern und bevorzugtes NAS.

Performance-Validierung Der Vector Datenbank

Performance-Validierung

Die Performance-Validierung spielt sowohl in Vektordatenbanken als auch in Storage-Systemen eine wichtige Rolle und dient als ein wichtiger Faktor für den optimalen Betrieb und die effiziente Ressourcenauslastung. Vector-Datenbanken, die für die Verarbeitung von hochdimensionalen Daten und die Ausführung von Ähnlichkeitssuchen bekannt sind, müssen ein hohes Leistungsniveau aufweisen, um komplexe Abfragen schnell und präzise verarbeiten zu können. Durch die Performance-Validierung werden Engpässe identifiziert und Konfigurationen feinangepasst. Außerdem wird sichergestellt, dass das System mit erwarteten Workloads umgehen kann, ohne dass der Service beeinträchtigt wird. Ebenso ist in Storage-Systemen die Performance-Validierung wichtig, um sicherzustellen, dass Daten effizient gespeichert und abgerufen werden, ohne Latenzprobleme oder Engpässe, die die Systemperformance insgesamt beeinträchtigen könnten. Darüber hinaus unterstützt sie fundierte Entscheidungen bei erforderlichen Upgrades oder Änderungen der Storage-Infrastruktur. Daher ist die Performance-Validierung ein entscheidender Aspekt des Systemmanagements und trägt erheblich zur Aufrechterhaltung einer hohen Servicequalität, Betriebseffizienz und der allgemeinen Systemzuverlässigkeit bei.

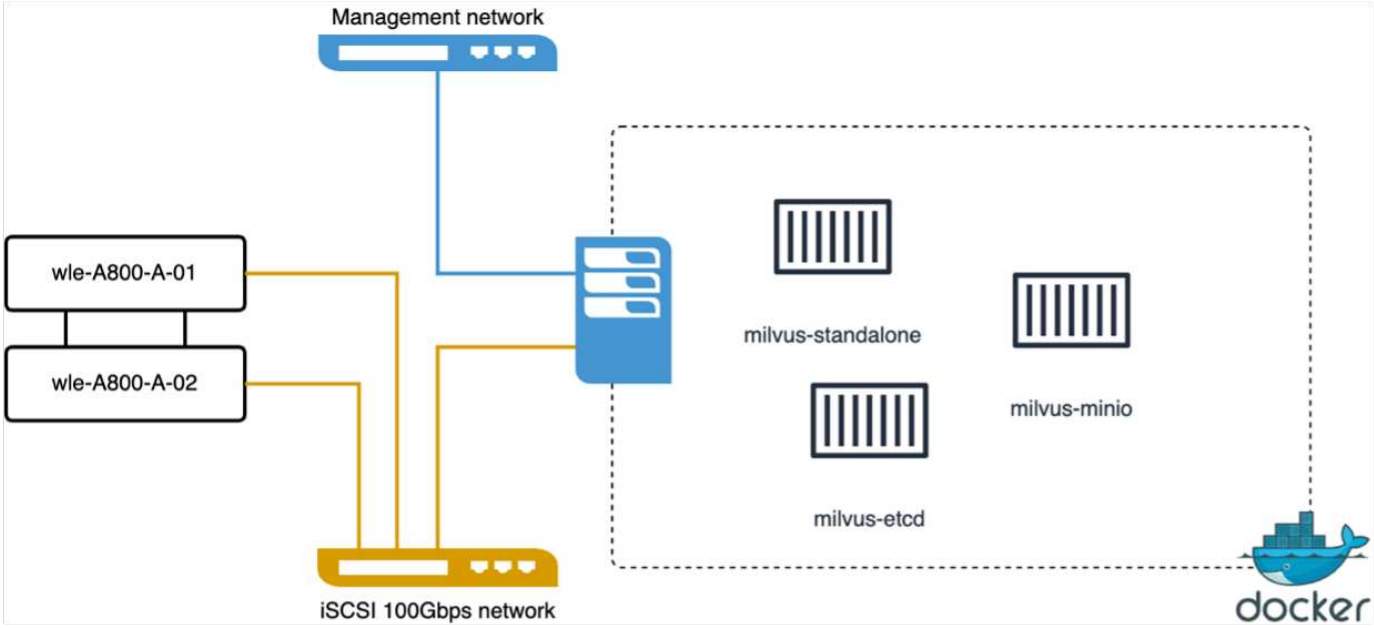
In diesem Abschnitt möchten wir uns mit der Performance-Validierung von Vektordatenbanken wie Milvus und pgvecto.rs beschäftigen. Dabei konzentrieren wir uns auf die Speicherleistungsmerkmale wie I/O-Profil und das Verhalten des NetApp-Speichercontrollers zur Unterstützung von RAG- und Inferenzarbeitslasten innerhalb des LLM-Lebenszyklus. Wir bewerten und ermitteln jegliche Performance-Unterscheidungsmerkmale, wenn diese Datenbanken mit der ONTAP Storage-Lösung kombiniert werden. Unsere Analyse basiert auf wichtigen Leistungsindikatoren, wie der Anzahl der pro Sekunde verarbeiteten Abfragen (QPS).

Bitte überprüfen Sie die für milvus verwendete Methode und den Fortschritt unten.

Details	Milvus (Standalone und Cluster)	Postgres(pgvecto.rs)
Version	2.3.2	0.2.0
Dateisystem	XFS auf iSCSI-LUNs	
Workload Generator	"VectorDB-Bank" – V0.0.5	
Datensätze	LAION-Datensatz * 10 Millionen Einbettungen * 768 Abmessungen * ~300 GB Datensatz Größe	

VectorDB-Bench mit Milvus Standalone Cluster

Wir haben die folgende Performance-Validierung auf milvus Standalone Cluster mit vectorDB-Bench durchgeführt.
Die Netzwerk- und Serverkonnektivität des eigenständigen milvus-Clusters ist unten angegeben.



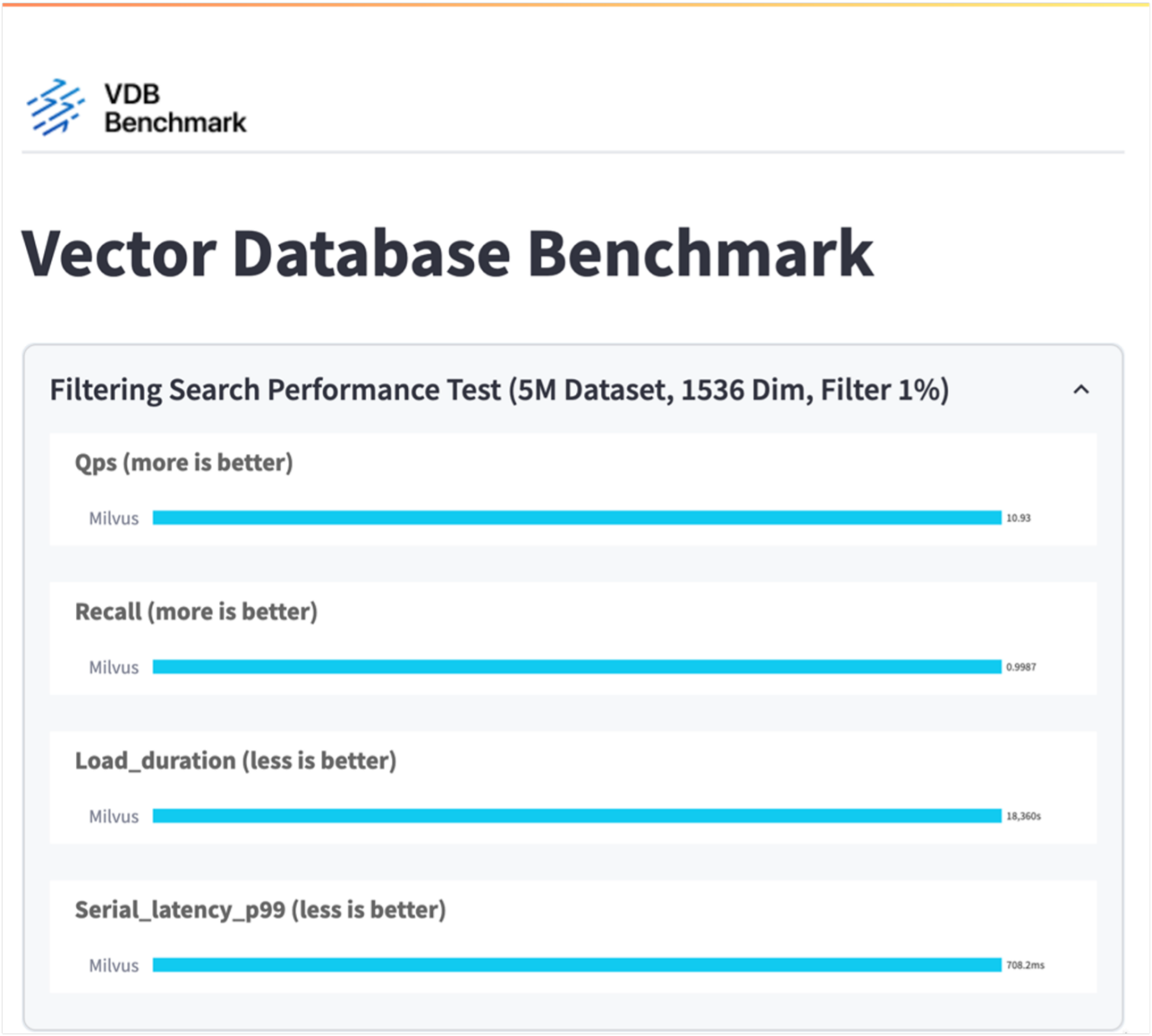
In diesem Abschnitt geben wir unsere Beobachtungen und Ergebnisse aus dem Testen der eigenständigen Datenbank von Milvus weiter.
. Wir haben DiskANN als Indextyp für diese Tests ausgewählt.
. Die Aufnahme, Optimierung und Erstellung von Indizes für einen ungefähr 100 GB großen Datensatz dauerte etwa 5 Stunden. Während der meisten dieser Dauer war der Milvus Server, ausgestattet mit 20 Kernen (was 40 vcpus entspricht, wenn Hyper-Threading aktiviert ist), mit seiner maximalen CPU-Kapazität von 100% betrieben.Wir fanden, dass DiskANN besonders wichtig ist für große Datensätze, die die Größe des Systemspeichers überschreiten.
. In der Abfragephase beobachteten wir eine Rate von Abfragen pro Sekunde (QPS) von 10.93 mit einem Rückruf von 0.9987. Die 99. Perzentillatenz für Abfragen wurde bei 708.2 Millisekunden gemessen.

Aus Sicht des Storage hat die Datenbank während der Aufnahme, der Optimierung nach dem Einfügen und der Indexerstellung ca. 1,000 OPs/s ausgegeben. In der Abfragephase wurden 32,000 OPs/s gefordert

Im folgenden Abschnitt werden die Storage-Performance-Kennzahlen vorgestellt.

Workload-Phase	Metrisch	Wert
Datenaufnahme Und Optimierung nach dem Einfügen	IOPS	< 1,000
	Latenz	< 400 usecs
	Workload	Lese-/Schreib-Kombination, überwiegend Schreibzugriffe
	I/O-Größe	64 KB
Abfrage	IOPS	Spitze bei 32,000
	Latenz	< 400 usecs
	Workload	100 % gecachte Lesevorgänge
	I/O-Größe	Hauptsächlich 8 KB

Das vectorDB-Bench-Ergebnis ist unten.

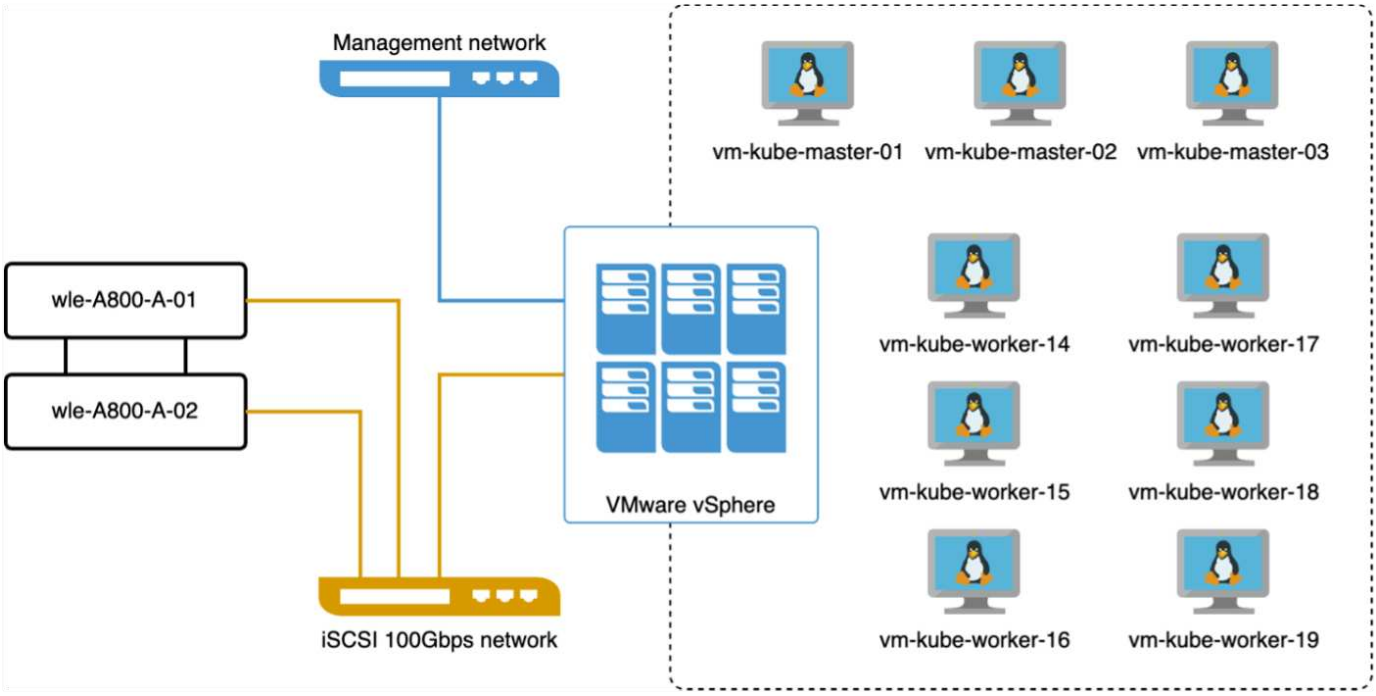


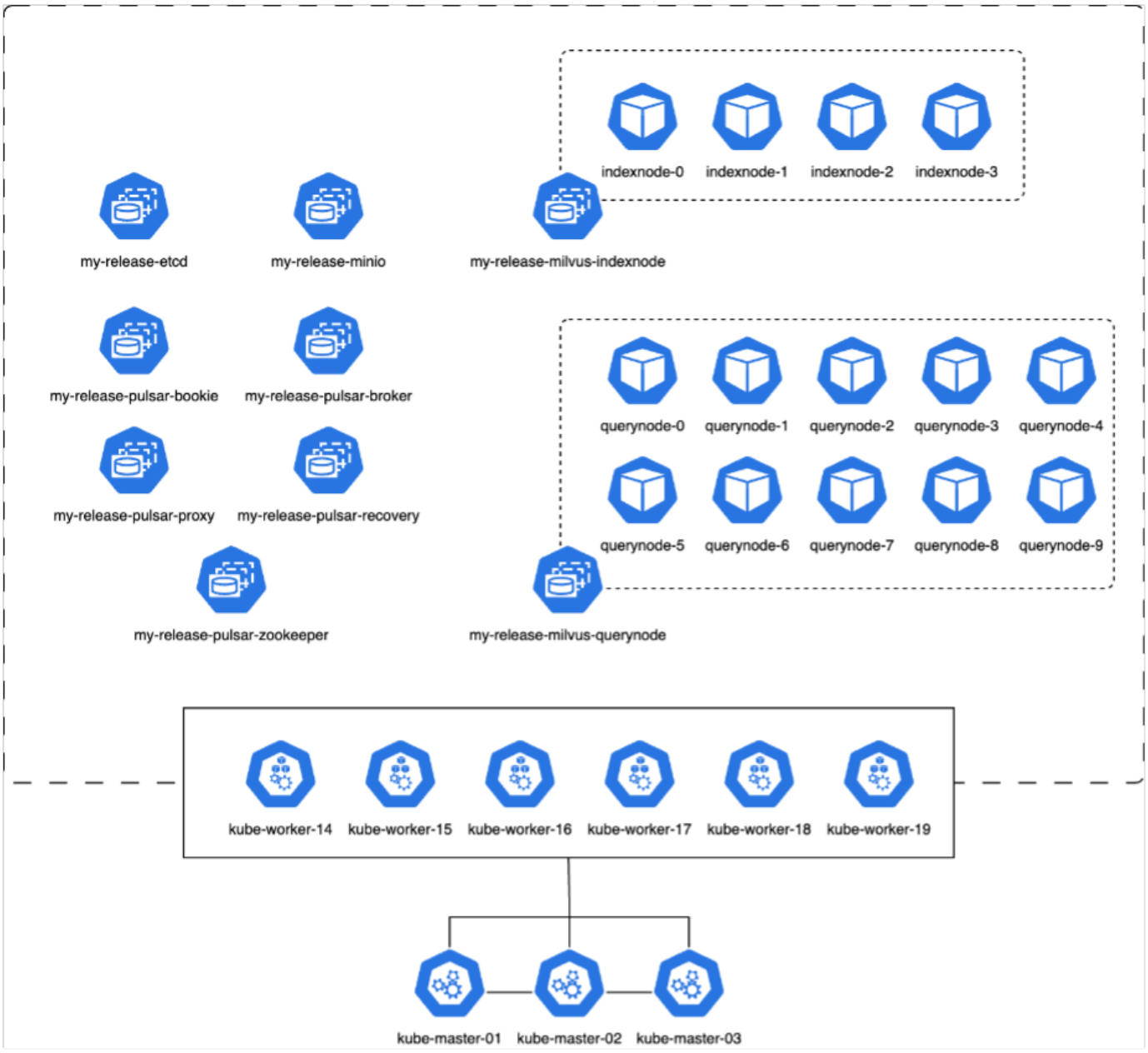
Aus der Performance-Validierung der eigenständigen Milvus-Instanz geht hervor, dass das aktuelle Setup nicht ausreicht, um einen Datensatz von 5 Millionen Vektoren mit einer Dimensionalität von 1536 zu unterstützen. Wir haben festgestellt, dass der Storage über ausreichende Ressourcen verfügt und keinen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

VectorDB-Bank mit milvus Cluster

In diesem Abschnitt wird die Implementierung eines Milvus Clusters in einer Kubernetes-Umgebung behandelt. Diese Kubernetes-Einrichtung wurde auf einer VMware vSphere Implementierung aufgebaut, in der die Kubernetes-Master- und -Worker-Nodes gehostet wurden.

Details zu Implementierungen von VMware vSphere und Kubernetes finden Sie in den folgenden Abschnitten.


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



In 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. Dabei stellten wir fest, dass die Zeit für die Datenaufnahme und die Optimierung nach dem Einfügen in das Cluster geringer war. Die 99. Perzentillatenz bei Abfragen wurde in der Cluster-Implementierung im Vergleich zum Standalone-Setup um das Sechsfache verringert.
* Obwohl die Rate für Abfragen pro Sekunde (QPS) in der Cluster-Bereitstellung höher war, war 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 nachfolgenden Abbildungen bieten eine Übersicht über verschiedene Storage-Kennzahlen, einschließlich der Storage-Cluster-Latenz und der gesamten IOPS (Input/Output Operations per Second).

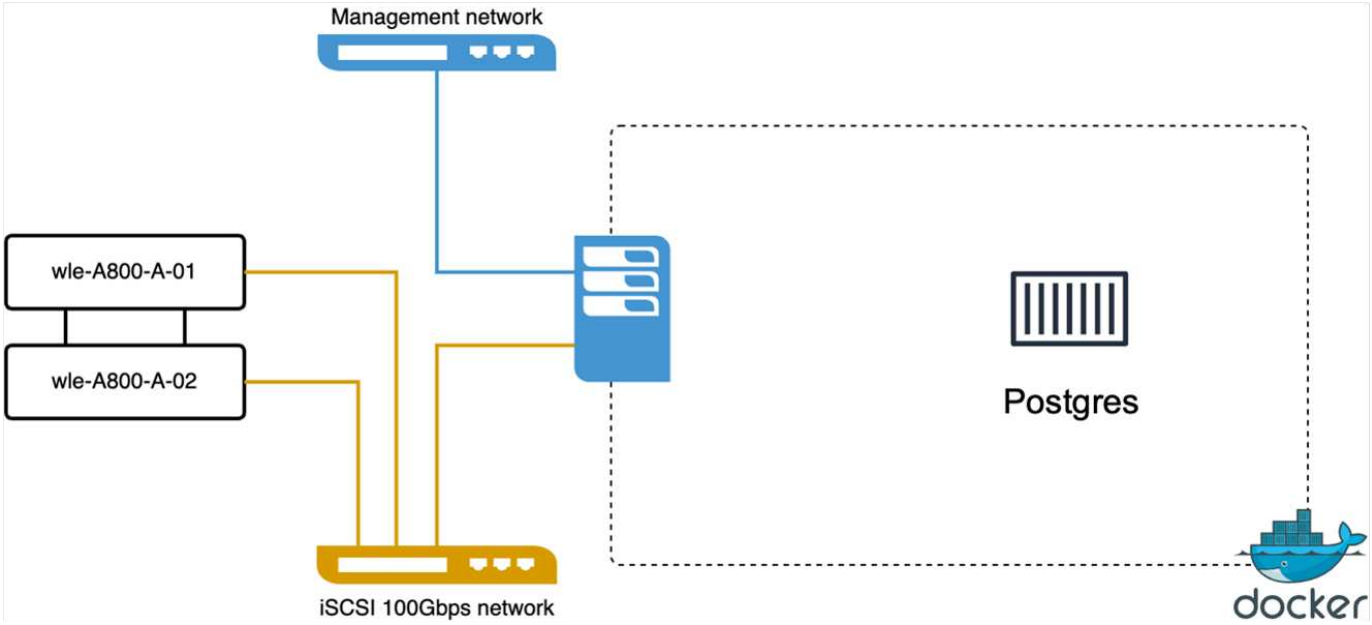
Im folgenden Abschnitt werden die wichtigsten Kennzahlen zur Storage-Performance aufgeführt.

Workload-Phase	Metrisch	Wert
Datenaufnahme Und Optimierung nach dem Einfügen	IOPS	< 1,000
	Latenz	< 400 usecs
	Workload	Lese-/Schreib-Kombination, überwiegend Schreibzugriffe
	I/O-Größe	64 KB
Abfrage	IOPS	Spitze bei 147,000
	Latenz	< 400 usecs
	Workload	100 % gecachte Lesevorgänge
	I/O-Größe	Hauptsächlich 8 KB

Basierend auf der Performance-Validierung des eigenständigen Milvus- und des Milvus-Clusters werden die Details des Storage-I/O-Profils dargestellt.
* Wir stellten fest, dass das I/O-Profil sowohl bei Standalone- als auch bei Cluster-Implementierungen konsistent bleibt.
* Der beobachtete Unterschied bei den IOPS-Spitzenwerten kann auf die größere Anzahl von Clients in der Cluster-Bereitstellung zurückgeführt werden.

VektorDB-Bank mit Postgres (pgvecto.rs)

Wir haben folgende Aktionen auf PostgreSQL(pgvecto.rs) mit VectorDB-Bench durchgeführt:
Die Details bezüglich der Netzwerk- und Serveranbindung von PostgreSQL (insbesondere pgvecto.rs) lauten wie folgt:

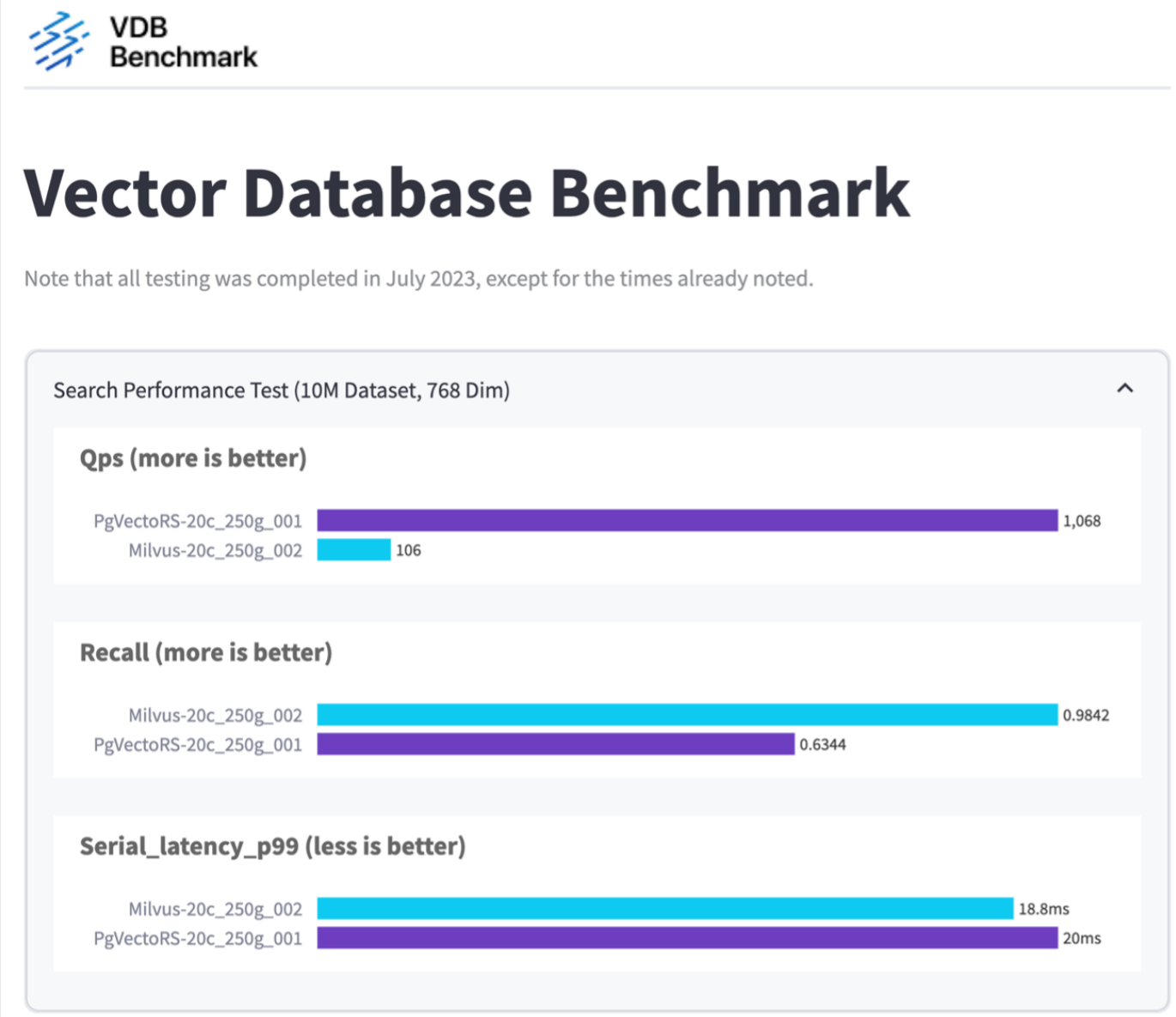


In diesem Abschnitt stellen wir unsere Beobachtungen und Ergebnisse aus dem Testen der PostgreSQL-Datenbank vor, insbesondere mit pgvecto.rs.
* Wir haben HNSW als Indextyp für diese Tests ausgewählt, weil zum Zeitpunkt des Tests DiskANN für pgvecto.rs nicht verfügbar war.
* Während der Datenaufnahme haben wir den Cohe-Datensatz geladen, der aus 10 Millionen Vektoren bei einer Dimensionalität von 768 besteht. Dieser Vorgang dauerte ungefähr 4.5 Stunden.
* In der Abfragephase beobachteten wir eine Rückruffunktrate von 1,068 Abfragen pro Sekunde (QPS) mit einem Rückruffunktsatz von 0.6344. Die 99. Perzentillatenz für Abfragen wurde bei 20 Millisekunden gemessen. Während der meisten Laufzeit wurde die Client-CPU mit 100 % Kapazität betrieben.

Die folgenden Abbildungen bieten eine Übersicht über verschiedene Storage-Kennzahlen, einschließlich der Gesamt-IOPS für die Storage-Cluster-Latenz (Input/Output Operations per Second).


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["Fehler: Fehlendes Grafikbild"]
```

Leistungsvergleich zwischen milvus und Postgres auf der Vektor-DB-Bank



Basierend auf unserer Leistungsvalidierung von Milvus und PostgreSQL mit VectorDBBench konnten wir Folgendes beobachten:

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

Wir fanden heraus, dass pgvecto.rs mit einem Rückruf von 0.6344 eine Queries per second (QPS)-Rate von 1,068 erreichte, während Milvus mit einem Rückruf von 0.9842 eine QPS-Rate von 106 erreichte.

Wenn hohe Präzision in Ihren Abfragen Priorität hat, übertrifft Milvus pgvecto.rs, da es einen höheren Anteil relevanter Elemente pro Abfrage abruft. Wenn jedoch die Anzahl der Abfragen pro Sekunde ein entscheidender Faktor ist, übersteigt pgvecto.rs Milvus. Es ist jedoch wichtig zu beachten, dass die Qualität der über pgvecto.rs abgerufenen Daten niedriger ist, wobei 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 I/O-Profil einem OLTP-Workload, beispielsweise in Oracle SLOB. Der Benchmark besteht aus drei Phasen: Datenaufnahme, Post-Optimierung und Abfrage. Die Anfangsphasen sind in erster Linie durch 64-KB-Schreibvorgänge gekennzeichnet, während die Abfragephase überwiegend 8-KB-Lesevorgänge umfasst. Wir erwarten, dass ONTAP die E/A-Last von Milvus kompetent verarbeitet.

Das PostgreSQL-I/O-Profil stellt keinen anspruchsvollen Storage Workload dar. Angesichts der aktuell laufenden in-Memory-Implementierung haben wir während der Abfragephase keine Festplatten-I/O beobachtet.

DiskANN entwickelt sich zu einer entscheidenden Technologie zur Differenzierung von Storage. Es ermöglicht die effiziente Skalierung der Vektor-DB-Suche über die Systemspeichergrenze hinaus. Es ist jedoch unwahrscheinlich, dass sich die Storage-Performance durch in-Memory-Vektor-DB-Indizes wie HNSW differenziert.

Es ist auch erwähnenswert, dass die Speicherung während der Abfragephase keine wichtige Rolle spielt, wenn der Indextyp HSNW ist, die wichtigste Betriebsphase für Vektordatenbanken, die RAG-Anwendungen unterstützen. Die Folge ist, dass die Storage Performance diese Applikationen nicht wesentlich beeinträchtigt.

Vector Database mit Instaclustr unter Verwendung von PostgreSQL: Pgvektor

Vector Database mit Instaclustr unter Verwendung von PostgreSQL: Pgvektor

In diesem Abschnitt befassen wir uns mit den Besonderheiten der Integration von instaclustr in PostgreSQL auf pgvektorieller Funktionalität. Wir haben ein Beispiel für „wie Sie Ihre LLM-Genauigkeit und -Leistung mit PGVector und PostgreSQL® verbessern können: Einführung in die Einbettung und die Rolle von PGVector“. Bitte prüfen Sie das "Blog" Um weitere Informationen zu erhalten.

Anwendungsfälle Für Vector Database

Anwendungsfälle Für Vector Database

In diesem Abschnitt besprechen wir zwei Anwendungsfälle wie Retrieval Augmented Generation with Large Language Models und NetApp IT Chatbot.

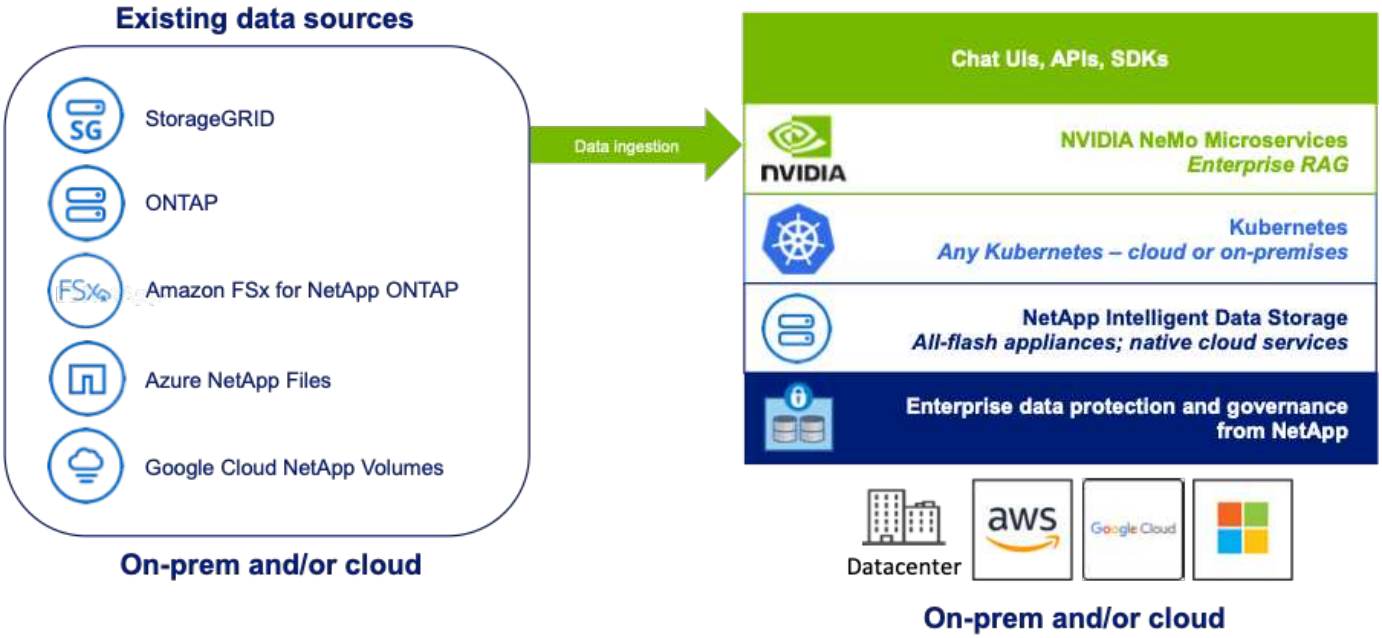
Retrieval Augmented Generation (RAG) mit Large Language Models (LLMs)

```
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 für die Implementierung von RAG in Unternehmen. Mit diesem Operator kann eine vollständige RAG-Pipeline bereitgestellt werden. Die RAG-Pipeline kann so angepasst werden, dass sie entweder Milvus oder pgvecto als Vektordatenbank zum Speichern von Knowledge-Base-Embeddings verwendet. Weitere Informationen finden Sie in der Dokumentation.

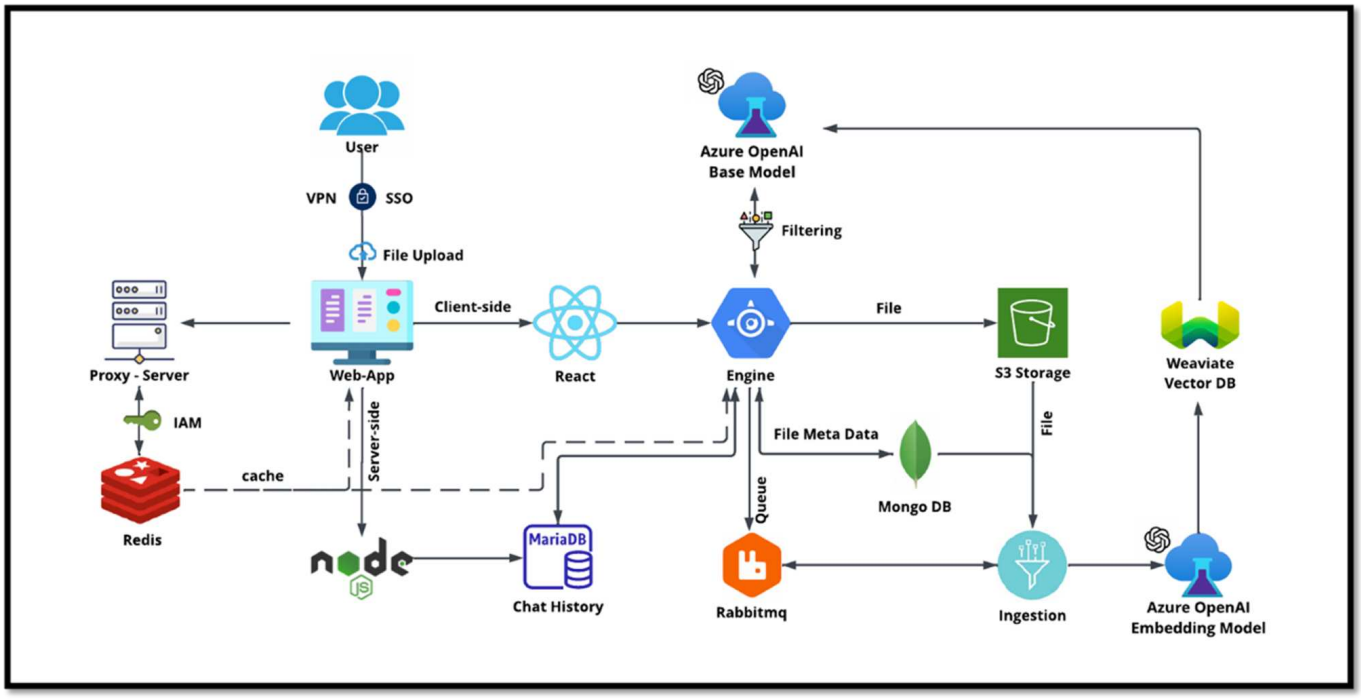
```
NetApp has validated an enterprise RAG architecture powered by the NVIDIA
Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog
post for more information and to see a demo. Figure 1 provides an overview
of this architecture.
```

Abbildung 1) RAG der Enterprise-Klasse mit NVIDIA Nemo Microservices und NetApp



Anwendungsfall mit NetApp IT-Chatbot

Der Chatbot von NetApp dient als weiterer Anwendungsfall in Echtzeit für die Vektordatenbank. Hier bietet die NetApp Private OpenAI Sandbox eine effektive, sichere und effiziente Plattform für das Management von Anfragen interner Benutzer von NetApp. Durch die Integration strenger Sicherheitsprotokolle, effizienter Datenmanagementsysteme und ausgefeilter KI-Verarbeitungsfunktionen garantiert sie hochwertige, präzise Reaktionen auf Benutzer basierend auf ihren Rollen und Verantwortlichkeiten im Unternehmen mittels SSO-Authentifizierung. Diese Architektur unterstreicht das Potenzial der Zusammenführung fortschrittlicher Technologien, um benutzerorientierte, intelligente Systeme zu schaffen.



Der Anwendungsfall kann in vier Hauptabschnitte unterteilt werden.

Benutzerauthentifizierung und -Verifizierung:

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

Datenübertragung und -Verarbeitung:

- Sobald das VPN validiert ist, werden die Daten über NetAIChat oder NetAICreate Web-Anwendungen an MariaDB gesendet. MariaDB ist ein schnelles und effizientes Datenbanksystem zur Verwaltung und Speicherung von Benutzerdaten.
- Anschließend sendet MariaDB die Informationen an die Instanz NetApp Azure, die die Benutzerdaten mit der AI-Verarbeitungseinheit verbindet.

Interaktion mit OpenAI und Content Filtering:

- Die Azure-Instanz sendet die Fragen der Anwender an ein Content-Filtersystem. Dieses System bereinigt die Abfrage und bereitet sie für die Verarbeitung vor.
- Der bereinigte Input wird dann an das Azure OpenAI-Basismodell gesendet, das auf Basis der Eingaben

eine Antwort generiert.

Reaktionserzeugung und Moderation:

- Die Antwort des Basismodells wird zunächst überprüft, um sicherzustellen, dass sie korrekt ist und den Content-Standards entspricht.
- Nach Bestehen der 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.

Schlussfolgerung

Schlussfolgerung

Abschließend bietet dieses Dokument einen umfassenden Überblick über die Implementierung und das Management von Vektordatenbanken wie Milvus und pgvektor auf NetApp Storage-Lösungen. Wir haben die Infrastrukturrichtlinien für die Nutzung von NetApp ONTAP und StorageGRID Objekt-Storage besprochen und die Milvus Datenbank in AWS FSX for NetApp ONTAP durch Datei- und Objektspeicher validiert.

Wir haben die Dualität von NetApp in Bezug auf Datei- und Objektdaten untersucht und dabei demonstriert, wie sich das Dienstprogramm nicht nur für Daten in Vektordatenbanken eignet, sondern auch für andere Applikationen. Außerdem haben wir erläutert, wie SnapCenter, das NetApp Produkt für Enterprise Management, Backup-, Restore- und Klonfunktionen für Vektordatenbankdaten bietet, um Datenintegrität und Verfügbarkeit zu gewährleisten.

Außerdem wird erläutert, wie die Hybrid Cloud-Lösung von NetApp Datenreplizierung und -Schutz in On-Premises- und Cloud-Umgebungen bietet und damit ein nahtloses und sicheres Datenmanagement ermöglicht. Wir haben Einblicke in die Performance-Validierung von Vektordatenbanken wie Milvus und pgvecto auf NetApp ONTAP gegeben und wertvolle Informationen zu deren Effizienz und Skalierbarkeit bereitgestellt.

Zum Abschluss diskutierten wir zwei generative KI-Anwendungsfälle: RAG mit LLM und das interne ChatAI von NetApp. Diese Praxisbeispiele 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 Storage-Lösungen von NetApp zum Management von Vektordatenbanken nutzen möchten.

Danksagungen

Der Autor möchte sich herzlich bei den unten genannten Beitragenden bedanken, anderen, die ihr Feedback und ihre Kommentare abgegeben haben, um dieses Dokument für NetApp-Kunden und NetApp Fields wertvoll zu machen.

1. Sathish Thyagarajan, Technical Marketing Engineer, ONTAP AI & Analytics, NetApp
2. Mike Oglesby, Technical Marketing Engineer, NetApp
3. AJ Mahajan, Senior Director, NetApp
4. Joe Scott, Manager, Workload Performance Engineering, NetApp
5. Puneet Dhawan, Senior Director, Product Management FSX, NetApp
6. Yuval Kalderon, Senior Product Manager, FSX Product Team, NetApp

Wo Sie weitere Informationen finden

Sehen Sie sich die folgenden Dokumente und/oder Websites an, um mehr über die in diesem Dokument beschriebenen Informationen zu erfahren:

- Milvus Dokumentation - https://milvus.io/docs/overview.md
- Eigenständige Dokumentation von Milvus – https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- NetApp Produktdokumentation
  https://www.netapp.com/support-and-training/documentation/
- Instaclustr - "Instalclusterdokumentation"

Versionsverlauf

Version	Datum	Versionsverlauf des Dokuments
Version 1.0	April 2024	Erste Version

Anhang A: Values.yaml

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

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

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

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
```