Solución de base de datos vectorial con NetApp

NetApp Solutions

NetApp
May 03, 2024

Tabla de contenidos

Solución Vector Database con NetApp

Karthikeyan Nagalingam y Rodrigo Nascimento, NetApp

Este documento proporciona una exploración en profundidad de la puesta en marcha y la gestión de las bases de datos vectoriales, como Milvus y pgvecto, una extensión PostgreSQL de código abierto, mediante las soluciones de almacenamiento de NetApp. Detalla las directrices de infraestructura para usar el almacenamiento de objetos NetApp ONTAP y StorageGRID y valida la aplicación de la base de datos de Milvus en AWS FSx para NetApp ONTAP. El documento elucida la dualidad archivos-objetos de NetApp y su utilidad para aplicaciones y bases de datos vectoriales que admiten la incrustación de vectores. Enfatiza las capacidades de SnapCenter, el producto de gestión empresarial de NetApp, ofreciendo funcionalidades de backup y restauración para bases de datos vectoriales, garantizando así la integridad y la disponibilidad de los datos. En este documento se profundiza en la solución de cloud híbrido de NetApp y se habla de su papel en la replicación y la protección de datos en entornos locales y de cloud. Incluye información sobre la validación del rendimiento de las bases de datos vectoriales en NetApp ONTAP y concluye con dos casos de uso prácticos sobre IA generativa: RAG con LLM y ChatAI interna de NetApp. Este documento sirve como una guía completa para aprovechar las soluciones de almacenamiento de NetApp para la gestión de bases de datos vectoriales.

La arquitectura de referencia se centra en lo siguiente:

Introducción

Introducción

Las bases de datos vectoriales abordan con eficacia los desafíos que están diseñados para manejar las complejidades de la búsqueda semántica en modelos de lenguaje grande (LLMs) y la inteligencia artificial generativa (IA). A diferencia de los sistemas tradicionales de gestión de datos, las bases de datos vectoriales son capaces de procesar y buscar a través de varios tipos de datos, incluyendo imágenes, vídeos, texto, audio, etc. y otras formas de datos no estructurados, usando el contenido de los datos en sí en lugar de etiquetas o etiquetas.

Las limitaciones de los sistemas de gestión de bases de datos relacionales (RDBMS) están bien documentadas, particularmente sus luchas con representaciones de datos de alta dimensión y datos no estructurados comunes en las aplicaciones de IA. Los RDBMS a menudo requieren un proceso largo y propenso a errores de aplanar los datos en estructuras más manejables, lo que conduce a retrasos e ineficiencias en las búsquedas. Las bases de datos vectoriales, sin embargo, están diseñadas para eludir estos problemas, ofreciendo una solución más eficiente y precisa para gestionar y buscar a través de datos complejos y de alta dimensión, facilitando así el avance de las aplicaciones de IA.

Este documento sirve como guía completa para los clientes que actualmente utilizan o planean usar bases de datos vectoriales, y detalla las mejores prácticas para utilizar bases de datos vectoriales en plataformas como NetApp ONTAP, NetApp StorageGRID, Amazon FSxN para NetApp ONTAP y SnapCenter. El contenido proporcionado aquí abarca una variedad de temas:

- Las directrices de infraestructura para bases de datos vectoriales, como Milvus, que proporciona el almacenamiento de NetApp mediante el almacenamiento de objetos de NetApp ONTAP y StorageGRID.
- Validación de la base de datos de Milvus en AWS FSx para NetApp ONTAP mediante almacén de archivos y objetos.
- Profundiza en la dualidad archivo-objeto de NetApp y demuestra su utilidad para datos en bases de datos vectoriales y otras aplicaciones.
- El producto de gestión de la protección de datos de NetApp, SnapCenter, ofrece funcionalidades de backup y restauración para datos de bases de datos vectoriales.
- Cómo el cloud híbrido de NetApp ofrece la replicación y la protección de datos en entornos locales y de cloud.
- Proporciona información sobre la validación del rendimiento de bases de datos vectoriales como Milvus y pgvector en NetApp ONTAP.
- Dos casos de uso específicos: Generación Aumentada de Recuperación (RAG) con Modelos de Lenguaje Grande (LLM) y ChatAI del equipo de TI DE NetApp, ofreciendo así ejemplos prácticos de los conceptos y prácticas esbozados.

Descripción general de la solución

Descripción general de la solución

Esta solución muestra las funciones y las ventajas específicas que NetApp ofrece para hacer frente a los desafíos a los que se enfrentan los clientes de bases de datos vectoriales. Con NetApp ONTAP, StorageGRID, las soluciones cloud de NetApp y SnapCenter, los clientes pueden añadir un valor significativo a sus operaciones empresariales. Estas herramientas no solo abordan los problemas existentes, sino que también mejoran la eficiencia y la productividad, lo que contribuye al crecimiento global del negocio.

¿Por qué elegir NetApp?

- Las ofertas de NetApp, como ONTAP y StorageGRID, permiten la separación del almacenamiento y de la computación, lo que permite una utilización óptima de los recursos en función de requisitos específicos. Esta flexibilidad permite a los clientes escalar de manera independiente su almacenamiento con las soluciones de almacenamiento de NetApp.
- Al aprovechar las controladoras de almacenamiento de NetApp, los clientes pueden servir datos de forma eficiente a su base de datos vectorial mediante los protocolos NFS y S3. Estos protocolos facilitan el almacenamiento de datos de los clientes y gestionan el índice de bases de datos vectoriales, lo que elimina la necesidad de contar con varias copias de los datos a las que se accede mediante métodos de objetos y archivos.
- NetApp ONTAP proporciona compatibilidad nativa para el almacenamiento NAS y de objetos en proveedores de servicios cloud líderes como AWS, Azure y Google Cloud. Esta amplia compatibilidad garantiza una integración perfecta, lo que permite la movilidad de datos del cliente, la accesibilidad global, la recuperación de desastres, la escalabilidad dinámica y el alto rendimiento.
- Con las sólidas capacidades de gestión de datos de NetApp, los clientes pueden estar seguros de que sus datos están bien protegidos frente a posibles riesgos y amenazas. NetApp da prioridad a la seguridad de los datos y ofrece tranquilidad a los clientes en lo que respecta a la seguridad e integridad de su información valiosa.

Base de datos vectorial

Base de datos vectorial

Una base de datos vectorial es un tipo especializado de base de datos diseñada para manejar, indexar y buscar datos no estructurados utilizando incrustaciones de modelos de aprendizaje automático. En lugar de organizar los datos en un formato tabular tradicional, organiza los datos como vectores de alta dimensión, también conocidos como incrustaciones vectoriales. Esta estructura única permite a la base de datos manejar datos complejos y multidimensionales de manera más eficiente y precisa.

Una de las funcionalidades clave de una base de datos vectorial es su uso de la IA generativa para realizar análisis. Esto incluye búsquedas de similitud, donde la base de datos identifica puntos de datos que son como una entrada dada, y detección de anomalías, donde puede detectar puntos de datos que se desvían significativamente de la norma.

Además, las bases de datos vectoriales son adecuadas para manejar datos temporales o datos con marca de tiempo. Este tipo de datos proporciona información sobre “qué” sucedió y cuándo sucedió, en secuencia y en relación con todos los demás eventos dentro de un sistema DE TI dado. Esta capacidad para manejar y analizar datos temporales hace que las bases de datos vectoriales sean particularmente útiles para aplicaciones que requieren una comprensión de los eventos a lo largo del tiempo.

Ventajas de la base de datos vectorial para ML e IA:

- Búsqueda de alta dimensión: Las bases de datos vectoriales destacan en la gestión y recuperación de datos de alta dimensión, que a menudo se generan en aplicaciones de IA y ML.
- Escalabilidad: Pueden escalar de forma eficiente para gestionar grandes volúmenes de datos y dar soporte al crecimiento y la expansión de proyectos de inteligencia artificial y APRENDIZAJE AUTOMÁTICO.
- Flexibilidad: Las bases de datos vectoriales ofrecen un alto grado de flexibilidad, lo que permite el alojamiento de diversos tipos de datos y estructuras.
- Rendimiento: Proporcionan gestión y recuperación de datos de alto rendimiento, vitales para la velocidad y

la eficiencia de las operaciones de IA y APRENDIZAJE AUTOMÁTICO.

- Indexación personalizable: Las bases de datos vectoriales ofrecen opciones de indexación personalizables, lo que permite una organización y recuperación de datos optimizadas en función de necesidades específicas.

Bases de datos vectoriales y casos de uso.

En esta sección se proporcionan varias bases de datos vectoriales y los detalles de sus casos de uso.

Faiss y Scann

Son bibliotecas que sirven como herramientas cruciales en el ámbito de la búsqueda vectorial. Estas bibliotecas proporcionan una funcionalidad que es fundamental en la gestión y búsqueda a través de datos vectoriales, lo que las convierte en recursos invaluables en esta área especializada de gestión de datos.

Elasticsearch

Es un motor de búsqueda y análisis ampliamente utilizado, ha incorporado recientemente capacidades de búsqueda vectorial. Esta nueva función mejora su funcionalidad, lo que permite al departamento de TI manejar y buscar datos vectoriales de forma más efectiva.

Piña

Es una base de datos vectorial robusta con un conjunto único de características. Admite vectores densos y dispersos en su funcionalidad de indexación, lo que mejora su flexibilidad y adaptabilidad. Uno de sus puntos fuertes reside en su capacidad para combinar métodos de búsqueda tradicionales con la búsqueda de vectores densos basada en IA, creando un método de búsqueda híbrido que aprovecha lo mejor de ambos mundos.

Principalmente basado en la nube, Pinecone está diseñado para aplicaciones de aprendizaje automático y se integra bien con una variedad de plataformas, incluyendo GCP, AWS, Open AI, GPT-3, GPT-3,5, GPT-4, Catgut Plus, Elasticsearch, Haystack, y mucho más. Es importante tener en cuenta que Pinecone es una plataforma de código cerrado y está disponible como una oferta de Software como Servicio (SaaS).

Dadas sus capacidades avanzadas, Pinecone es especialmente adecuado para la industria de la ciberseguridad, donde sus capacidades de búsqueda híbrida y de búsqueda de alta dimensión se pueden aprovechar de manera efectiva para detectar y responder a las amenazas.

Croma

Es una base de datos vectorial que tiene una Core-API con cuatro funciones principales, una de las cuales incluye un almacén de vectores de documentos en memoria. También utiliza la biblioteca Face Transformers para vectorizar documentos, mejorando su funcionalidad y versatilidad.
Chroma está diseñado para funcionar tanto en la nube como en las instalaciones, ofreciendo flexibilidad basada en las necesidades del usuario. En particular, destaca en aplicaciones relacionadas con el audio, lo que lo convierte en una excelente opción para motores de búsqueda basados en audio, sistemas de recomendación de música y otros casos de uso relacionados con el audio.

Weaviate

Es una base de datos vectorial versátil que permite a los usuarios vectorizar su contenido utilizando sus módulos integrados o módulos personalizados, proporcionando flexibilidad en función de necesidades específicas. Ofrece tanto soluciones totalmente gestionadas como autoalojadas, atendiendo a una variedad de preferencias de implementación.

Una de las características clave de Weaviate es su capacidad para almacenar tanto vectores como objetos, mejorando sus capacidades de manejo de datos. Es ampliamente utilizado para una gama de aplicaciones, incluyendo búsqueda semántica y clasificación de datos en sistemas ERP. En el sector del comercio electrónico, impulsa motores de búsqueda y recomendación. Weaviate también se utiliza para la búsqueda de imágenes, la detección de anomalías, la armonización automatizada de datos y el análisis de amenazas de ciberseguridad, mostrando su versatilidad en múltiples dominios.

Reedis

Redis es una base de datos vectorial de alto rendimiento conocida por su rápido almacenamiento en memoria, que ofrece baja latencia para operaciones de lectura y escritura. Esto lo convierte en una excelente opción para sistemas de recomendación, motores de búsqueda y aplicaciones de análisis de datos que requieren un acceso rápido a los datos.

Redis admite varias estructuras de datos para vectores, incluidas listas, conjuntos y conjuntos ordenados. También proporciona operaciones vectoriales, como el cálculo de distancias entre vectores o la búsqueda de intersecciones y uniones. Estas características son particularmente útiles para la búsqueda de similitudes, la agrupación en clústeres y los sistemas de recomendación basados en contenido.

En términos de escalabilidad y disponibilidad, Redis destaca en la gestión de cargas de trabajo de alto rendimiento y ofrece replicación de datos. También se integra bien con otros tipos de datos, incluidas las bases de datos relacionales tradicionales (RDBMS).
Redis incluye una función Publicar/Suscribir (Pub/Sub) para actualizaciones en tiempo real, lo que es beneficioso para administrar vectores en tiempo real. Además, Redis es ligero y fácil de usar, lo que lo convierte en una solución fácil de usar para la gestión de datos vectoriales.

Milvus

Es una base de datos vectorial versátil que ofrece una API como un almacén de documentos, al igual que MongoDB. Se destaca por su soporte para una amplia variedad de tipos de datos, por lo que es una opción popular en los campos de la ciencia de datos y el aprendizaje automático.

Una de las características únicas de Milvus es su capacidad de multivectorización, que permite a los usuarios especificar en tiempo de ejecución el tipo de vector a utilizar para la búsqueda. Además, utiliza Knowwhere, una biblioteca que se encuentra sobre otras bibliotecas como Faiss, para gestionar la comunicación entre las consultas y los algoritmos de búsqueda vectorial.

Milvus también ofrece una integración perfecta con flujos de trabajo de aprendizaje automático, gracias a su compatibilidad con PyTorch y TensorFlow. Esto lo convierte en una excelente herramienta para una amplia gama de aplicaciones, incluyendo comercio electrónico, análisis de imágenes y vídeos, reconocimiento de objetos, búsqueda de similitud de imágenes y recuperación de imágenes basada en contenido. En el ámbito del procesamiento del lenguaje natural, Milvus se utiliza para la agrupación de documentos, la búsqueda semántica y los sistemas de respuesta a preguntas.

Para esta solución, elegimos Milvus para la validación de la solución. Para el rendimiento, utilizamos tanto milvus como postgres (pgvecto.rs).

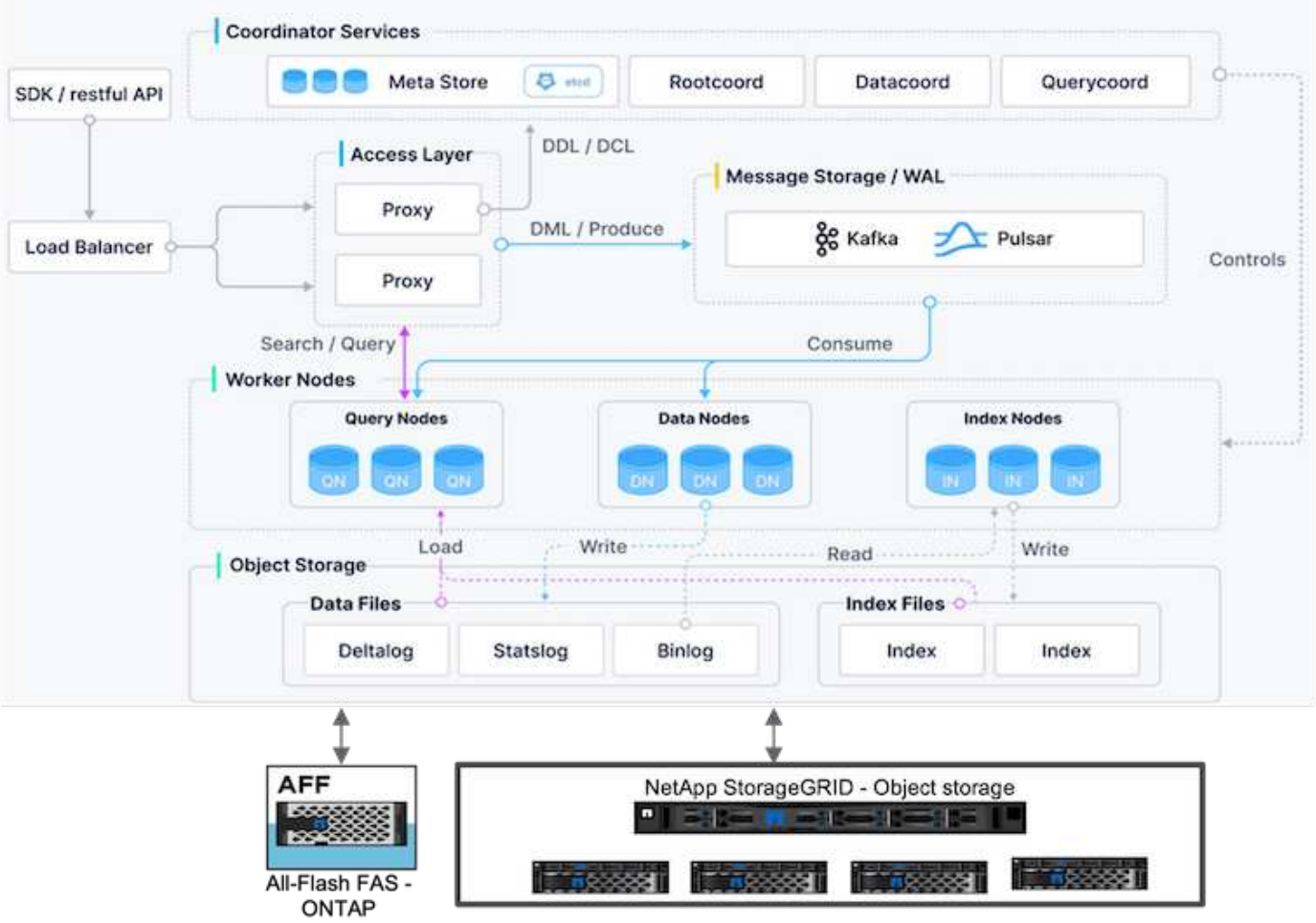
¿Por qué elegimos Milvus para esta solución?

- Código abierto: Milvus es una base de datos vectorial de código abierto que fomenta el desarrollo y las mejoras impulsadas por la comunidad.
- Integración de IA: Aprovecha la integración de aplicaciones de búsqueda de similitud y de IA para mejorar la funcionalidad de las bases de datos vectoriales.
- Manejo de grandes volúmenes: Milvus tiene la capacidad de almacenar, indexar y gestionar más de mil

millones de vectores de incrustación generados por los modelos de redes neuronales profundas (DNN) y aprendizaje automático (ML).

- Fácil de usar: Es fácil de usar, con la configuración que tarda menos de un minuto. Milvus también ofrece SDK para diferentes lenguajes de programación.
- Velocidad: Ofrece velocidades de recuperación increíblemente rápidas, hasta 10 veces más rápidas que algunas alternativas.
- Escalabilidad y disponibilidad: Milvus es altamente escalable, con opciones para escalar horizontal y verticalmente según sea necesario.
- Rico en características: Admite diferentes tipos de datos, filtrado de atributos, soporte de función definida por el usuario (UDF), niveles de consistencia configurables y tiempo de viaje, lo que lo convierte en una herramienta versátil para diversas aplicaciones.

Descripción general de la arquitectura Milvus



Esta sección proporciona componentes de palanca superiores y servicios utilizados en la arquitectura Milvus.
* Capa de acceso: Se compone de un grupo de proxies sin estado y sirve como la capa frontal del sistema y el punto final para los usuarios.
* Servicio de Coordinador: Asigna las tareas a los nodos de los trabajadores y actúa como el cerebro de un sistema. Tiene tres tipos de coordinador: Coord raíz, coord de datos y coord de consulta.
* Nodos de trabajador: Sigue la instrucción del servicio coordinador y ejecuta el usuario disparado DML/DDL commands.it tiene tres tipos de nodos de trabajo, como nodo de consulta, nodo de datos y nodo de índice.
* Almacenamiento: Es responsable de la persistencia de datos. Incluye almacenamiento de metadatos, agente de registro y almacenamiento de objetos. El almacenamiento de NetApp como ONTAP y StorageGRID

proporciona almacenamiento de objetos y almacenamiento basado en archivos a Milvus para datos de clientes y datos de bases de datos vectoriales.

Requisitos tecnológicos

Requisitos tecnológicos

Las configuraciones de hardware y software descritas a continuación se utilizaron para la mayoría de las validaciones realizadas en este documento, a excepción del rendimiento. Estas configuraciones sirven de guía para ayudarle a configurar el entorno. Sin embargo, tenga en cuenta que los componentes específicos pueden variar en función de los requisitos de cada cliente.

Requisitos de hardware

Hardware subyacente	Detalles
Par de alta disponibilidad de la cabina de almacenamiento NetApp AFF	* A800 * ONTAP 9.14.1 * 48 X 3,49TB SSD-NVM * Dos volúmenes de grupo flexibles: Metadatos y datos. * Metadatos El volumen NFS tiene 12 volúmenes persistentes con 250GB. * Los datos son un volumen ONTAP NAS S3
6 SISTEMAS PRIMERGY RX2540 M4 DE FUJITSU	* 64 CPU * Intel® Xeon® Gold 6142 CPU @ 2,60GHz * 256 GB de memoria física de GM * 1 x 100GbE puerto de red
Redes	100 GbE
StorageGRID	* 1 x SG100, 3xSGF6024 * 3 x 24 x 7,68TB

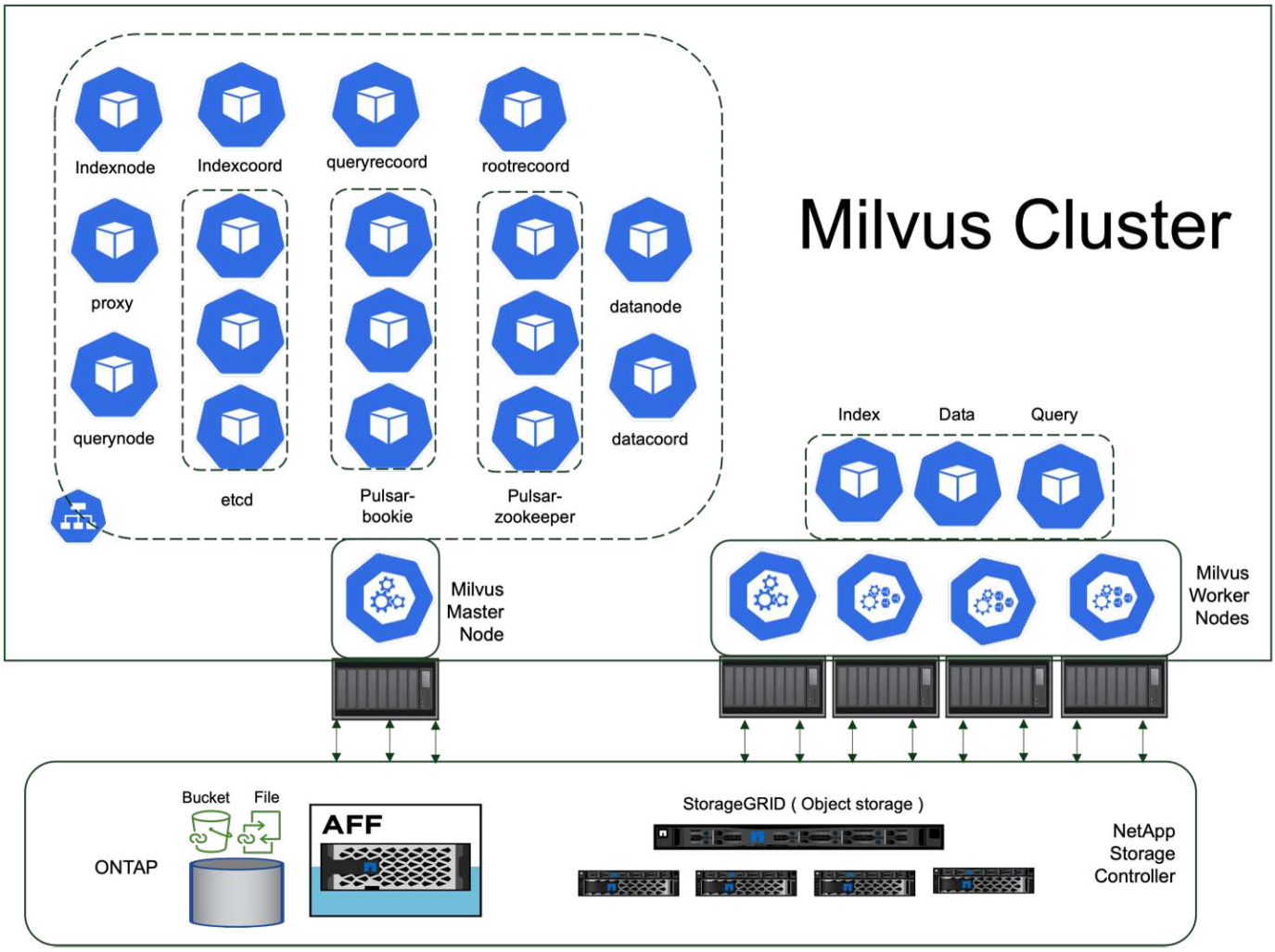
Requisitos de software

De NetApp	Detalles
Clúster de Milvus	* GRÁFICO - milvus-4,1.11. * Versión APP – 2.3.4 * Paquetes dependientes como contador, zookeeper, pulsar, etcd, proxy, querynode, trabajador
Kubernetes	* 5 nodos K8s cluster * 1 Nodo maestro y 4 Nodos de trabajador * Versión – 1.7.2
Python	*3.10.12.

Procedimiento de Despliegue

Procedimiento de despliegue

En esta sección de puesta en marcha, utilizamos una base de datos vectorial milvus con Kubernetes para la configuración del laboratorio del siguiente modo.



El sistema de almacenamiento de NetApp proporciona el almacenamiento al clúster para mantener los datos de los clientes y los datos de clúster de milvus.

Configuración del almacenamiento de NetApp: ONTAP

- Inicialización de sistema de almacenamiento
- Creación de máquinas virtuales de almacenamiento (SVM)
- Asignación de interfaces de red lógicas
- Configuración y licencia de NFS, S3

Siga los pasos que se indican a continuación para NFS (Sistema de archivos de red):

1. Cree un volumen de FlexGroup para NFSv4. En nuestra configuración para esta validación, hemos utilizado 48 SSD, 1 SSD dedicados para el volumen raíz de la controladora y 47 SSD distribuidos en NFSv4]].Verifique que la política de exportación NFS del volumen FlexGroup tenga permisos de lectura/escritura para la red de nodos de Kubernetes (K8s). Si estos permisos no están en su lugar, otorgue permisos de lectura/escritura (rw) para la red de nodos K8s.

2. En todos los nodos K8s, cree una carpeta y monte el volumen FlexGroup en esta carpeta a través de una interfaz lógica (LIF) en cada nodo K8s.

Siga los pasos que se indican a continuación para NAS S3 (Network Attached Storage Simple Storage Service):

1. Cree un volumen de FlexGroup para NFS.
2. Configure un object-store-server con HTTP activado y el estado de administrador definido en 'UP' mediante el comando «Vserver object-store-server create». Tiene la opción de activar HTTPS y definir un puerto de listener personalizado.
3. Cree un usuario object-store-server utilizando el comando «vserver object-store-server user create -user <username>».
4. Para obtener la clave de acceso y la clave secreta, puede ejecutar el siguiente comando: «Set diag; vserver object-store-server user show -user <username>». Sin embargo, en adelante, estas claves se proporcionarán durante el proceso de creación del usuario o se pueden recuperar mediante llamadas a la API de REST.
5. Establezca un grupo object-store-server utilizando el usuario creado en el paso 2 y otorgue acceso. En este ejemplo, hemos proporcionado “FullAccess”.
6. Cree un depósito NAS estableciendo su tipo en “nas” y suministrando la ruta al volumen NFSv3. También es posible utilizar un cubo S3 para este propósito.

Configuración del almacenamiento de NetApp: StorageGRID

1. Instale el software de StorageGRID.
2. Cree un inquilino y un bloque.
3. Cree un usuario con el permiso necesario.

Consulte más información en https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

Descripción general de la solución

Hemos realizado una completa validación de la solución centrada en cinco áreas clave, cuyos detalles se describen a continuación. Cada sección analiza los retos a los que se enfrentan los clientes, las soluciones ofrecidas por NetApp y las ventajas subsiguientes para el cliente.

1. "Configuración de clúster de Milvus con Kubernetes en las instalaciones"
   Retos del cliente para escalar de manera independiente en el almacenamiento y la computación, una gestión de la infraestructura eficaz y la gestión de los datos. En esta sección detallamos el proceso de instalación de un clúster Milvus en Kubernetes, utilizando una controladora de almacenamiento de NetApp tanto para los datos de clúster como para los de cliente.
2. "Milvus con Amazon FSxN para NetApp ONTAP: Dualidad de archivos y objetos"
   En esta sección, ¿Por qué necesitamos implementar la base de datos vectorial en la nube?, así como los pasos para implementar la base de datos vectorial (milvus standalone) en Amazon FSxN para NetApp ONTAP dentro de contenedores docker.
3. "Protección de bases de datos vectoriales mediante NetApp SnapCenter."
   En esta sección, profundizamos en cómo SnapCenter protege los datos de la base de datos vectorial y los datos de Milvus que residen en ONTAP. En este ejemplo, utilizamos un bucket NAS (milvusdbvol1) derivado de un volumen ONTAP NFS (vol1) para los datos de los clientes y un volumen NFS independiente (vectordbpv) para los datos de configuración del cluster Milvus.

4. "Recuperación ante desastres con SnapMirror de NetApp"
   En esta sección trataremos la importancia de la recuperación ante desastres para las bases de datos vectoriales y cómo NetApp el producto de recuperación ante desastres snapmirror proporciona la solución de recuperación ante desastres para la base de datos vectorial.
5. "Validación del rendimiento"
   En este apartado, el objetivo es profundizar en la validación del rendimiento de las bases de datos vectoriales, como Milvus y pgvecto.rs, centrándose en sus características de rendimiento del almacenamiento, como el perfil de I/O y el comportamiento de la controladora de almacenamiento NetApp, para admitir cargas de trabajo de inferencia y RAG dentro del ciclo de vida del LLM. Evaluaremos e identificaremos cualquier diferenciación en el rendimiento cuando estas bases de datos se combinen con la solución de almacenamiento de ONTAP. Nuestro análisis se basará en indicadores clave de rendimiento, como el número de consultas procesadas por segundo (QPS).

Configuración de clústeres de Milvus con Kubernetes en las instalaciones

Configuración de clúster de Milvus con Kubernetes en las instalaciones

Los retos del cliente para escalar de manera independiente en el almacenamiento y la computación, así como en la gestión de datos y la gestión de la infraestructura eficaz.
Kubernetes y las bases de datos vectoriales forman una solución potente y escalable para gestionar grandes operaciones de datos. Kubernetes optimiza los recursos y gestiona los contenedores, mientras que las bases de datos vectoriales manejan de manera eficiente datos de alta dimensión y búsquedas de similitud. Esta combinación permite el procesamiento rápido de consultas complejas en conjuntos de datos de gran tamaño y se escala sin problemas con volúmenes de datos en crecimiento, lo que la hace ideal para las aplicaciones de Big Data y las cargas de trabajo de IA.

1. En esta sección detallamos el proceso de instalación de un clúster Milvus en Kubernetes, utilizando una controladora de almacenamiento de NetApp tanto para los datos de clúster como para los de cliente.
2. Para instalar un clúster Milvus, se requieren volúmenes persistentes (VP) para almacenar datos de varios componentes del clúster Milvus. Estos componentes incluyen etcd (tres instancias), pulsar-bookie-journal (tres instancias), pulsar-bookie-ledgers (tres instancias) y pulsar-zookeeper-data (tres instancias).

   > En milvus cluster, podemos usar pulsar o kafka para el motor subyacente que soporta el almacenamiento confiable del clúster Milvus y la publicación/suscripción de flujos de mensajes. En el caso de Kafka con NFS, NetApp ha realizado mejoras en ONTAP 9.12.1 y versiones posteriores. Además, estas mejoras, junto con los cambios NFSv4,1 y Linux que se incluyen en RHEL 8,7 o 9,1 y posteriores, solucionan el problema «nombre tonto» que puede producirse al ejecutar Kafka a través de NFS. Si desea obtener información más detallada sobre el tema de ejecutar kafka con solución NFS de NetApp, consulte - https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html.

3. Creamos un único volumen NFS en NetApp ONTAP y establecimos 12 volúmenes persistentes, cada uno con 250GB TB de almacenamiento. El tamaño del almacenamiento puede variar dependiendo del tamaño del clúster; por ejemplo, tenemos otro clúster en el que cada VP tiene 50GB TB. Por favor refiérase a continuación a uno de los archivos PV YAML para más detalles; teníamos 12 archivos de este tipo en total. En cada archivo, el storageClassName se establece en 'almacén', y el almacenamiento y la ruta de acceso son únicos para cada VP.

```
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. Ejecute el comando 'kubectl apply' para cada archivo PV YAML para crear los volúmenes persistentes, y luego verifique su creación usando '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. Para almacenar datos de clientes, Milvus admite soluciones de almacenamiento de objetos como MinIO, Azure Blob y S3. En esta guía, utilizamos S3. Los siguientes pasos se aplican tanto al almacén de objetos de ONTAP S3 como a StorageGRID. Utilizamos Helm para desplegar el cluster Milvus. Descargue el archivo de configuración VALUES.yaml desde la ubicación de descarga de Milvus. Consulte el apéndice del archivo VALUES.yaml que utilizamos en este documento.
6. Asegúrese de que 'torageClass' está establecido en 'default' en cada sección, incluidos los correspondientes al log, el ETCD, el zookeeper y el contador.
7. En la sección MinIO, desactive MinIO.
8. Cree un bloque NAS a partir del almacenamiento de objetos ONTAP o StorageGRID e inclúyalos en un S3 externo con las credenciales de almacenamiento de objetos.

```
###################################
# 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. Antes de crear el clúster de Milvus, asegúrese de que PersistentVolumeClaim (RVP) no tenga recursos preexistentes.

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

10. Utilice Helm y el archivo de configuración values.yaml para instalar e iniciar el clúster Milvus.

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

11. Compruebe el estado de las reclamaciones de volúmenes persistentes (RVP).

```
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. Compruebe el estado de los 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>
```

Asegúrese de que el estado de PODS es 'en ejecución' y funciona según lo esperado

13. Prueba de escritura y lectura de datos en el almacenamiento de objetos Milvus y NetApp.
    - Escriba datos con el programa Python «prepare_data_netapp_new.py».

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

- Lea los datos con el archivo Python «verify_data_netapp.py».

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

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

=== Start Creating index IVF_FLAT  ===

=== Start loading                  ===

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

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

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

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

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

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

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

Basada en la validación anterior, la integración de Kubernetes con una base de datos vectorial, como se demuestra mediante la puesta en marcha de un clúster Milvus en Kubernetes mediante una controladora de almacenamiento de NetApp, ofrece a los clientes una solución sólida, escalable y eficiente para la gestión de operaciones de datos a gran escala. Esta configuración proporciona a los clientes la capacidad de manejar datos de alta dimensión y ejecutar consultas complejas de forma rápida y eficiente, lo que la convierte en una solución ideal para las aplicaciones de Big Data y las cargas de trabajo de IA. El uso de volúmenes persistentes (VP) para varios componentes del cluster, junto con la creación de un único volumen NFS desde NetApp ONTAP, garantiza una utilización óptima

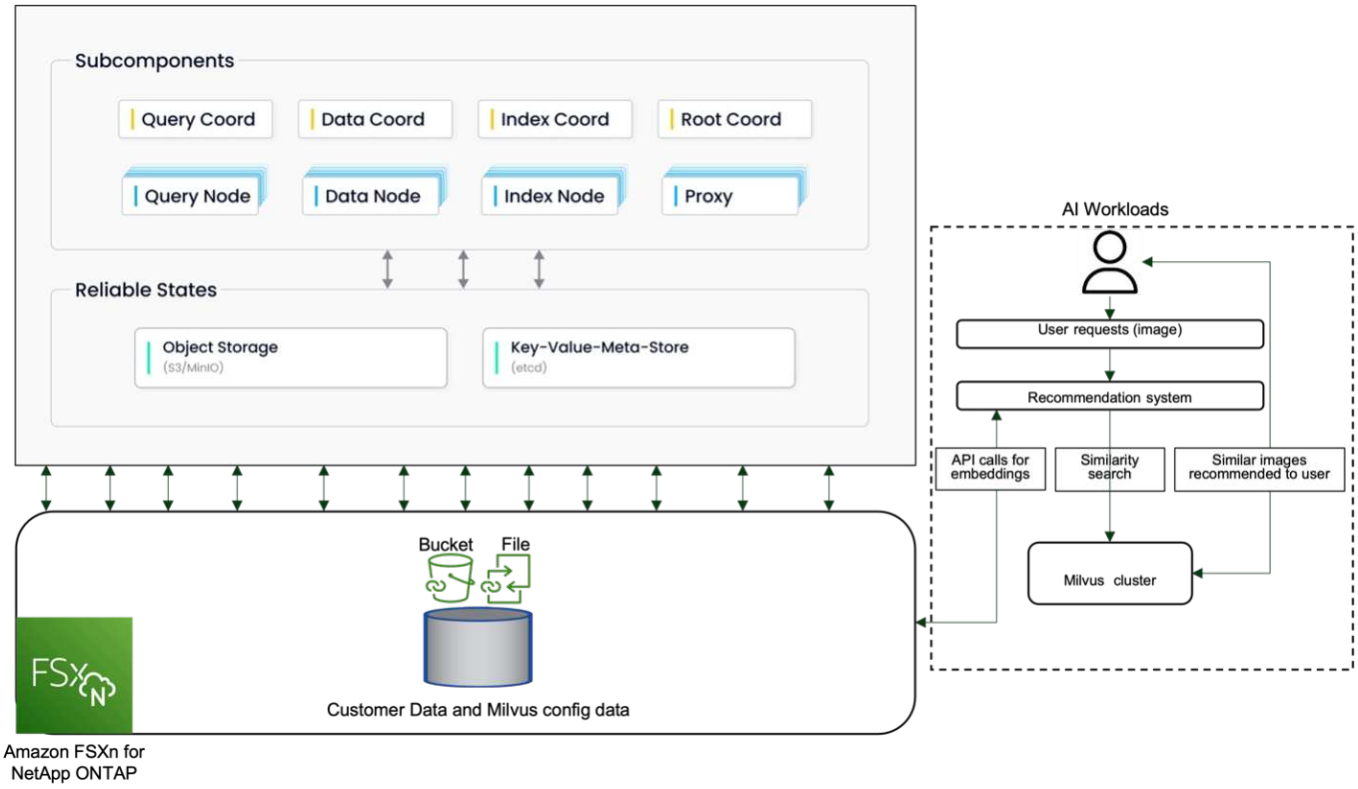
de los recursos y una gestión de datos. El proceso de verificación del estado de PersistentVolumeClaims (RVP) y Pods, así como la realización de pruebas de escritura y lectura de datos, proporciona a los clientes la garantía de realizar operaciones de datos fiables y coherentes. El uso del almacenamiento de objetos de ONTAP o StorageGRID para los datos de clientes mejora aún más la accesibilidad de los datos y la seguridad. En general, esta configuración ofrece a los clientes una solución de gestión de datos resiliente y de alto rendimiento que puede escalarse sin problemas a medida que vayan aumentando sus necesidades relacionadas con datos.

Milvus con Amazon FSxN para NetApp ONTAP - dualidad de archivos y objetos

Milvus con Amazon FSxN para NetApp ONTAP: Dualidad de archivos y objetos

En esta sección, ¿Por qué necesitamos implementar la base de datos vectorial en la nube, así como los pasos para implementar la base de datos vectorial (milvus standalone) en Amazon FSxN para NetApp ONTAP dentro de contenedores docker?

La implementación de una base de datos vectorial en la nube ofrece varias ventajas significativas, especialmente para aplicaciones que requieren manejar datos de alta dimensión y ejecutar búsquedas de similitud. En primer lugar, la implementación basada en la nube ofrece escalabilidad, lo que permite un ajuste fácil de los recursos para que se adapten a los crecientes volúmenes de datos y las cargas de consultas. Esto garantiza que la base de datos pueda gestionar eficientemente el aumento de la demanda mientras mantiene un alto rendimiento. Segundo, la puesta en marcha de cloud proporciona alta disponibilidad y recuperación ante desastres, ya que los datos pueden replicarse entre distintas ubicaciones geográficas, lo que minimiza el riesgo de pérdida de datos y garantiza un servicio continuo incluso durante eventos inesperados. Tercero, proporciona rentabilidad, ya que solo paga por los recursos que utiliza y puede ampliar o reducir verticalmente en función de la demanda, evitando la necesidad de realizar una inversión inicial sustancial en hardware. Por último, la implementación de una base de datos vectorial en la nube puede mejorar la colaboración, ya que se puede acceder a los datos y compartirlos desde cualquier lugar, lo que facilita el trabajo en equipo y la toma de decisiones basadas en datos.
Compruebe la arquitectura del milvus independiente con Amazon FSxN for NetApp ONTAP utilizada en esta validación.



1. Cree una instancia de Amazon FSxN para NetApp ONTAP y anote los detalles de la VPC, los grupos de seguridad de VPC y la subred. Esta información será necesaria al crear una instancia de EC2. Puedes encontrar más detalles aquí - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. Cree una instancia de EC2, asegurándose de que la VPC, los grupos de seguridad y la subred coincidan con los de la instancia de Amazon FSxN para NetApp ONTAP.
3. Instale nfs-common con el comando 'apt-get install nfs-common' y actualice la información del paquete usando 'udo apt-get update'.
4. Cree una carpeta de montaje y monte Amazon FSxN for NetApp ONTAP en ella.

```
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. Instale Docker y Docker Compose usando 'apt-get install'.
6. Configure un clúster Milvus basado en el archivo docker-compose.yaml, que se puede descargar desde el sitio web de Milvus.

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

7. En la sección 'Volumes' del archivo docker-compose.yml, asigne el punto de montaje NFS de NetApp a la ruta correspondiente del contenedor de Milvus, específicamente en etcd, minio y standalone.Check "Apéndice D: docker-compose.yml" para obtener detalles sobre los cambios en yml
8. Verifique las carpetas y los archivos montados.

```
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. Ejecute 'Docker-compose up -d' desde el directorio que contiene el archivo docker-compose.yml.
10. Compruebe el estado del contenedor Milvus.

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

11. Para validar la funcionalidad de lectura y escritura de la base de datos vectorial y sus datos en Amazon FSxN para NetApp ONTAP, utilizamos el SDK de Python Milvus y un programa de muestra de PyMilvus. Instale los paquetes necesarios usando 'apt-get install python3-numpy python3-pip' e instale PyMilvus usando 'pip3 install pymilvus'.
12. Validar las operaciones de escritura y lectura de datos desde Amazon FSxN para NetApp ONTAP en la base de datos vectorial.

```
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. Compruebe la operación de lectura con el script verify_data_netapp.py.

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

=== Milvus host: localhost         ===

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

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

14. Si el cliente quiere acceder (leer) a datos NFS probados en la base de datos vectorial a través del protocolo S3 para cargas de trabajo de IA, se puede validar mediante un programa sencillo de Python. Un ejemplo de esto podría ser una búsqueda de similitud de imágenes de otra aplicación como se menciona en la imagen que está al principio de esta sección.

```
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#
```

En esta sección se muestra de forma eficaz cómo los clientes pueden poner en marcha y operar una configuración independiente de Milvus en contenedores Docker mediante el almacenamiento de datos FSxN de NetApp de Amazon para NetApp ONTAP. Esta configuración permite a los clientes aprovechar la potencia de las bases de datos vectoriales para gestionar datos de alta dimensión y ejecutar consultas complejas, todo ello dentro del entorno escalable y eficiente de los contenedores Docker. Al crear una instancia de Amazon FSxN para NetApp ONTAP y relacionar la instancia de EC2, los clientes pueden garantizar una utilización óptima de los recursos y una gestión de datos. La validación correcta de las operaciones de escritura y lectura de datos de FSxN en la base de datos vectorial proporciona a los clientes la garantía de operaciones de datos fiables y coherentes. Además, la capacidad de mostrar (leer) datos de cargas de trabajo de IA a través del protocolo S3 ofrece accesibilidad a los datos mejorada. Por lo tanto, este completo proceso proporciona a los clientes una solución sólida y eficiente para gestionar sus operaciones de datos a gran escala, aprovechando las funcionalidades de FSxN para NetApp ONTAP de Amazon.

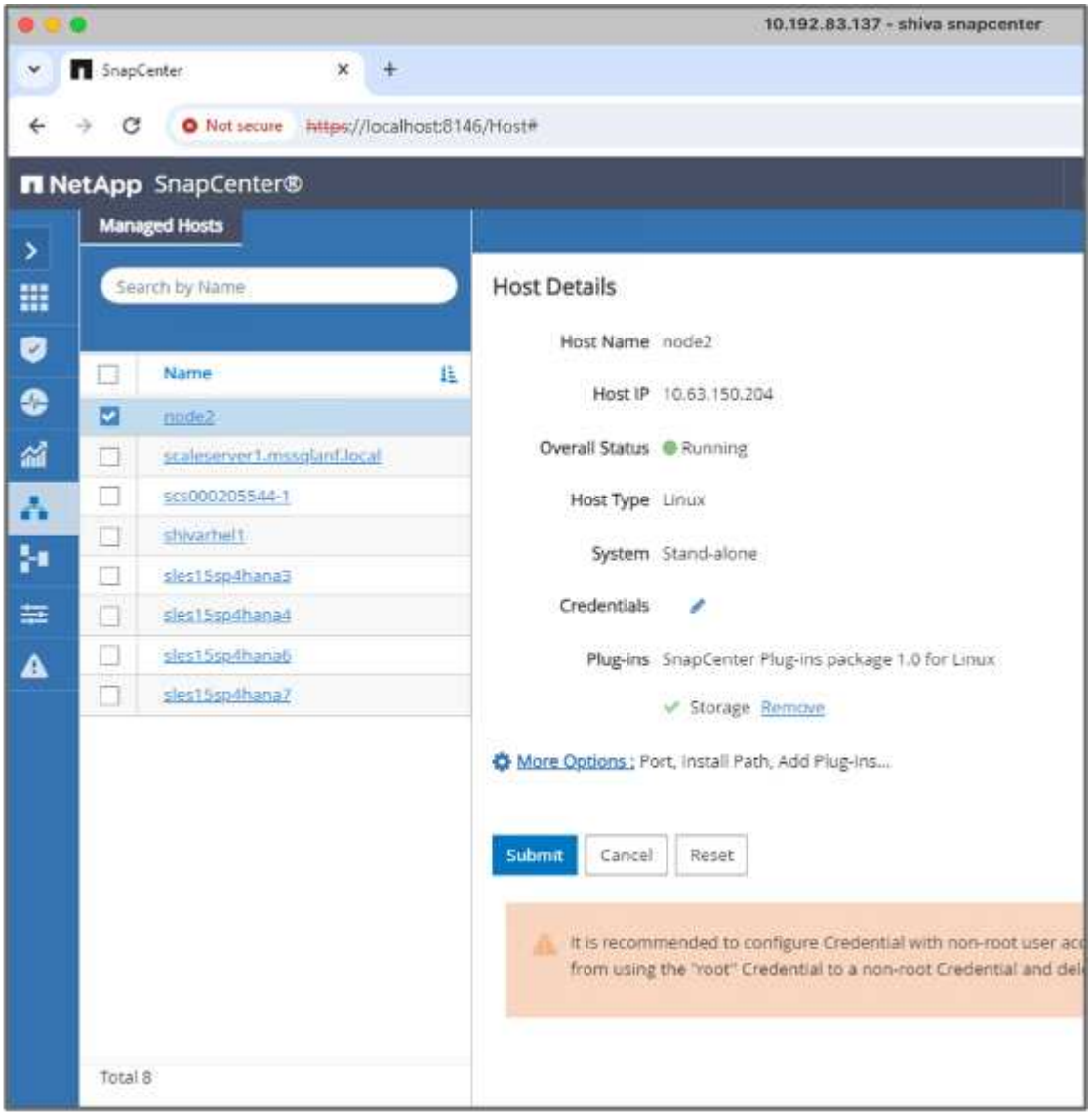
Protección de bases de datos vectoriales mediante SnapCenter

Protección de bases de datos vectoriales mediante NetApp SnapCenter.

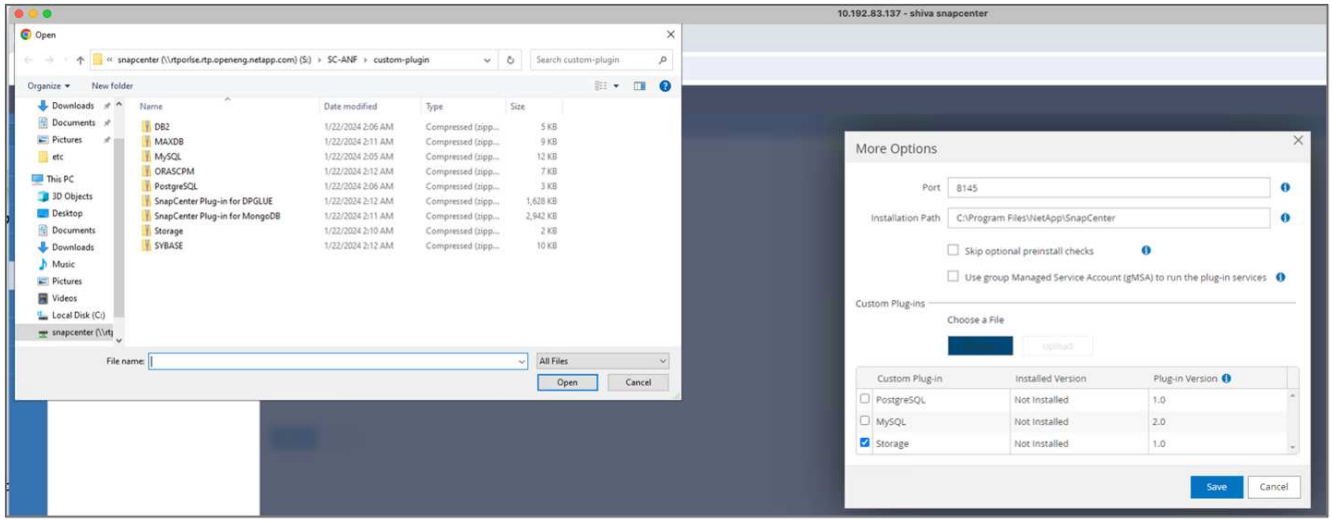
Por ejemplo, en el sector de producción cinematográfica, los clientes suelen poseer datos incrustados cruciales, como archivos de vídeo y audio. La pérdida de estos datos, debido a problemas como fallos en las unidades de disco duro, puede tener un impacto significativo en sus operaciones, lo que puede poner en

peligro empresas de varios millones de dólares. Hemos encontrado casos en los que se perdió contenido inestimable, lo que causó interrupciones sustanciales y pérdidas financieras. Garantizar la seguridad e integridad de estos datos esenciales es, por tanto, de suma importancia en este sector.
En esta sección, profundizamos en cómo SnapCenter protege los datos de la base de datos vectorial y los datos de Milvus que residen en ONTAP. En este ejemplo, utilizamos un bucket NAS (milvusdbvol1) derivado de un volumen ONTAP NFS (vol1) para los datos de los clientes y un volumen NFS independiente (vectordbpv) para los datos de configuración del cluster Milvus. compruebe el "aquí" para el flujo de trabajo de backup de SnapCenter

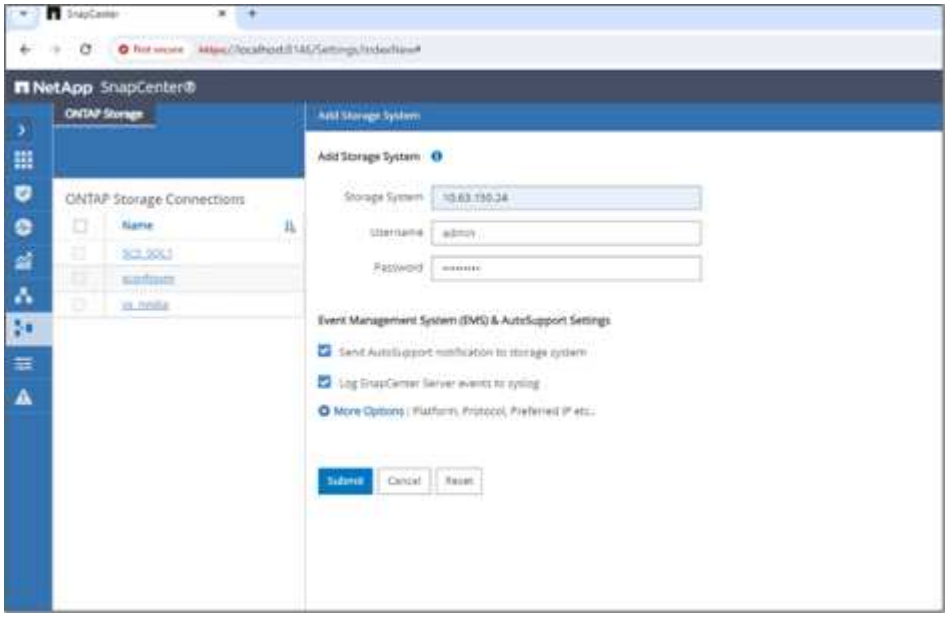
1. Configure el host que se utilizará para ejecutar comandos SnapCenter.



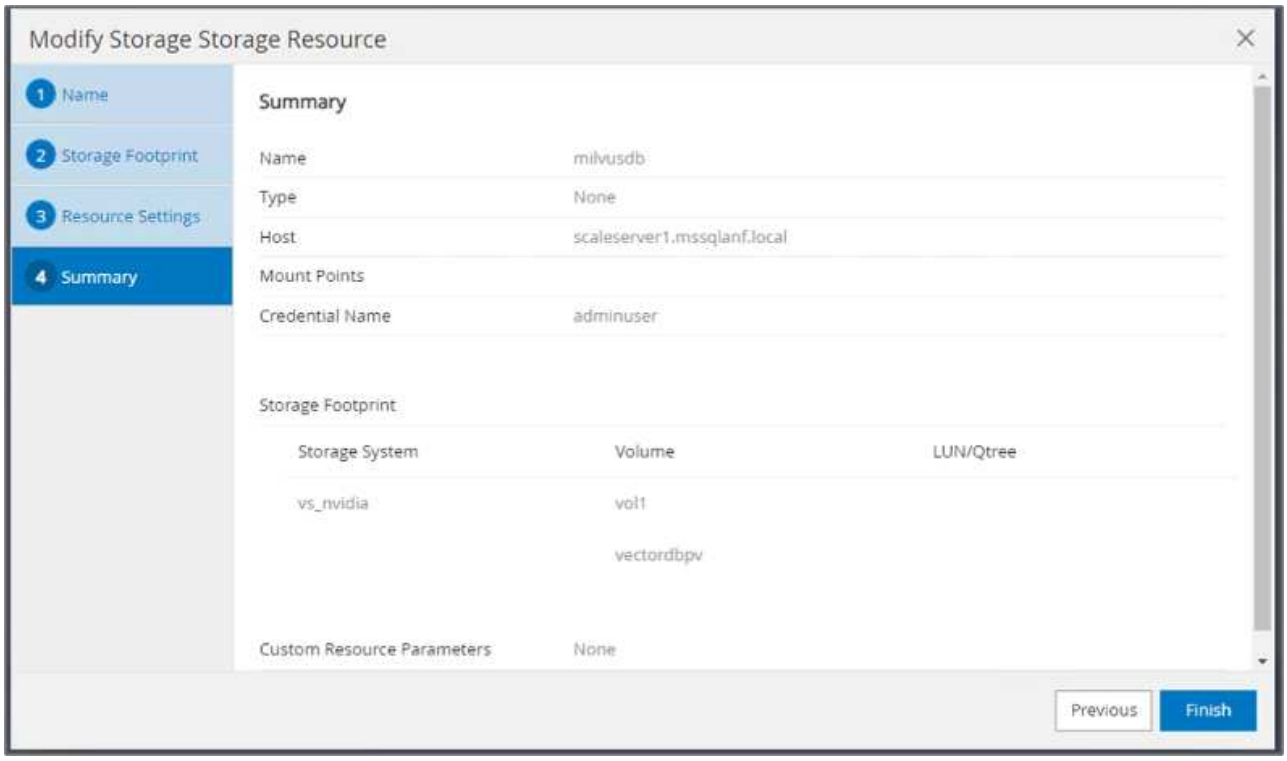
2. Instale y configure el complemento de almacenamiento. En el host agregado, seleccione “Más opciones”. Navegue hasta y seleccione el plugin de almacenamiento descargado en "Automation Store de NetApp". Instale el plugin y guarde la configuración.



3. Configure el sistema de almacenamiento y el volumen: Añada el sistema de almacenamiento en «Storage System» y seleccione la SVM (Storage Virtual Machine). En este ejemplo, hemos elegido «vs_nvidia».



4. Establecer un recurso para la base de datos vectorial, incorporando una política de backup y un nombre de snapshot personalizado.
   - Habilite el backup del grupo de consistencia con valores predeterminados y habilite SnapCenter sin consistencia del sistema de archivos.
   - En la sección Storage Footprint, seleccione los volúmenes asociados a los datos de los clientes de bases de datos vectoriales y los datos de clúster de Milvus. En nuestro ejemplo, estos son «vol1» y «vectordbpv».
   - Cree una política para la protección de bases de datos vectoriales y proteja el recurso de bases de datos vectoriales con la política.



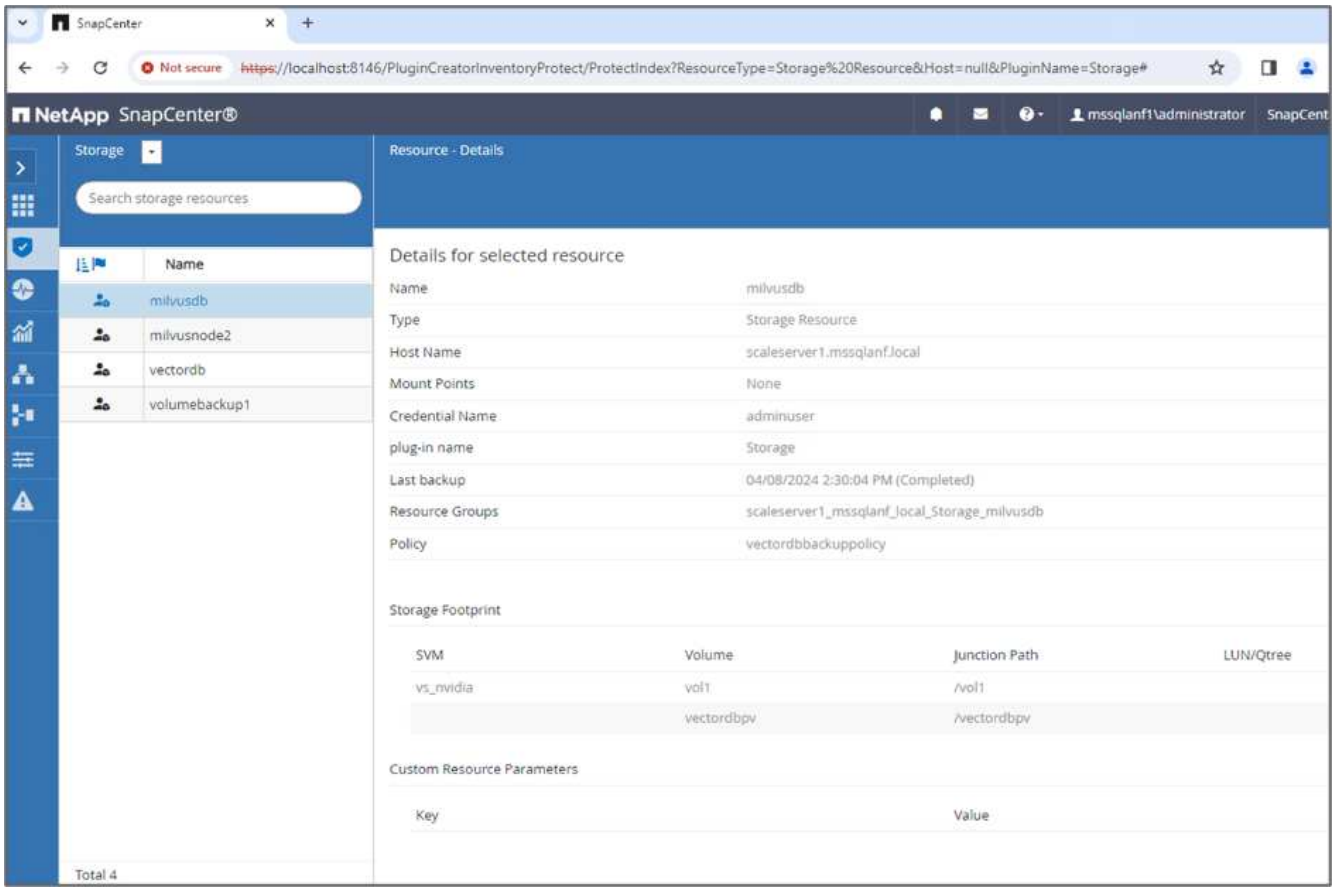
5. Inserte los datos en el bloque NAS de S3 mediante un script de Python. En nuestro caso, modificamos el script de backup proporcionado por Milvus, a saber, 'prepare_data_netapp.py', y ejecutamos el comando 'ync' para vaciar los datos del sistema operativo.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_test` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_test: 3000

=== Create collection `hello_milvus_netapp_sc_test2` ===

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

6. Compruebe los datos en el bloque NAS de S3. En nuestro ejemplo, los archivos con la marca de tiempo '2024-04-08 21:22' se crearon mediante el script 'prepare_data_netapp.py'.

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

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

7. Inicie un backup utilizando la snapshot de grupo de consistencia (CG) desde el recurso 'milvusdb'

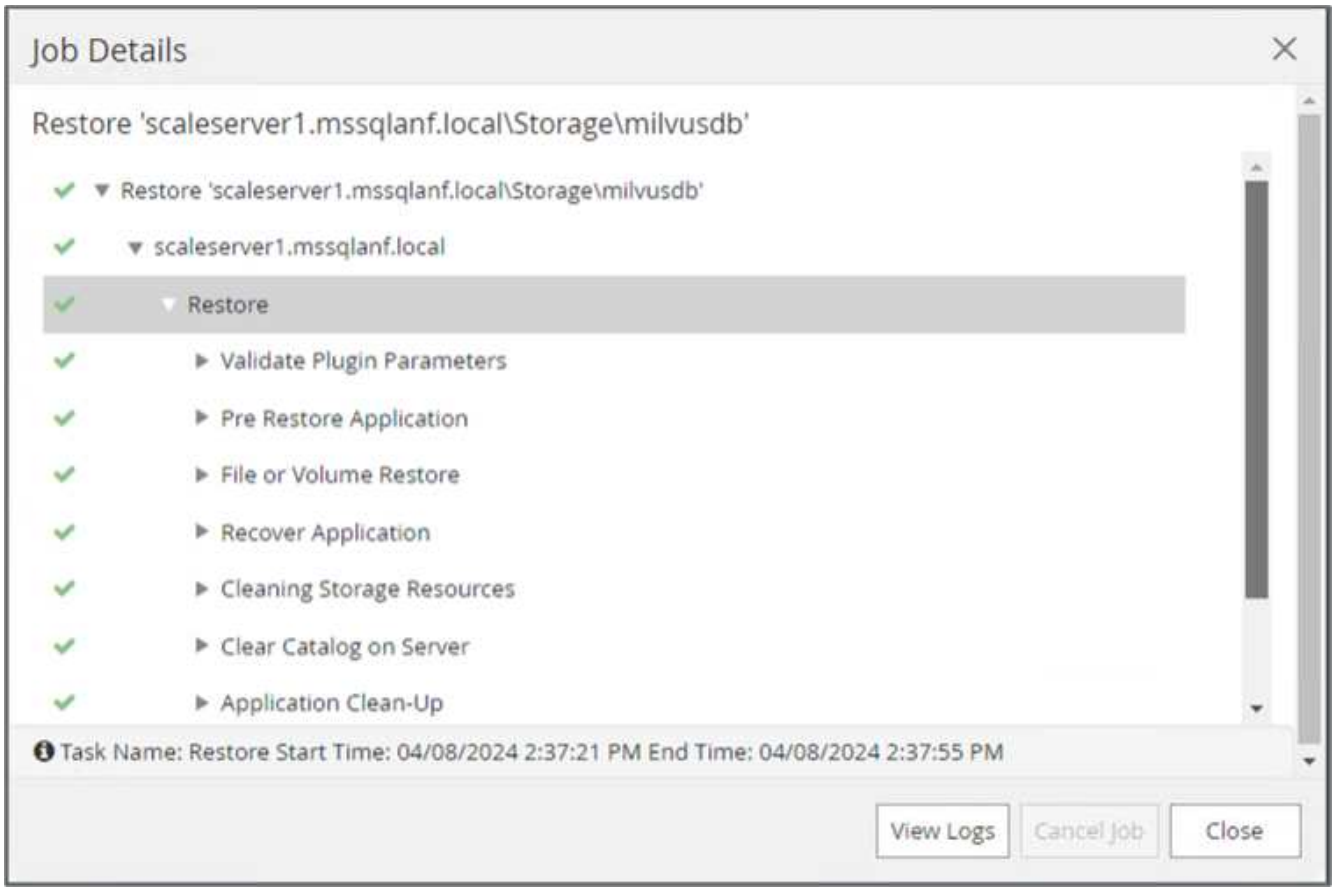


8. Para probar la funcionalidad de copia de seguridad, hemos agregado una nueva tabla después del proceso de copia de seguridad o hemos eliminado algunos datos del NFS (S3 bucket NAS).

   Para esta prueba, imagine un escenario en el que alguien creó una recopilación nueva, innecesaria o inapropiada después de la copia de seguridad. En tal caso, necesitaríamos revertir la base de datos vectorial a su estado antes de que se agregara la nueva colección. Por ejemplo, se han insertado nuevas colecciones, como 'hello_milvus_netapp_sc_testnew' y 'hello_milvus_netapp_sc_testnew2'.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_testnew` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

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

9. Ejecute una restauración completa del depósito de NAS S3 desde la instantánea anterior.



10. Utilice un script de Python para verificar los datos de las colecciones 'hello_milvus_netapp_sc_test' y 'hello_milvus_netapp_sc_test2'.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

11. Compruebe que la recopilación innecesaria o inapropiada ya no está presente en la base de datos.

```
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:~#
```

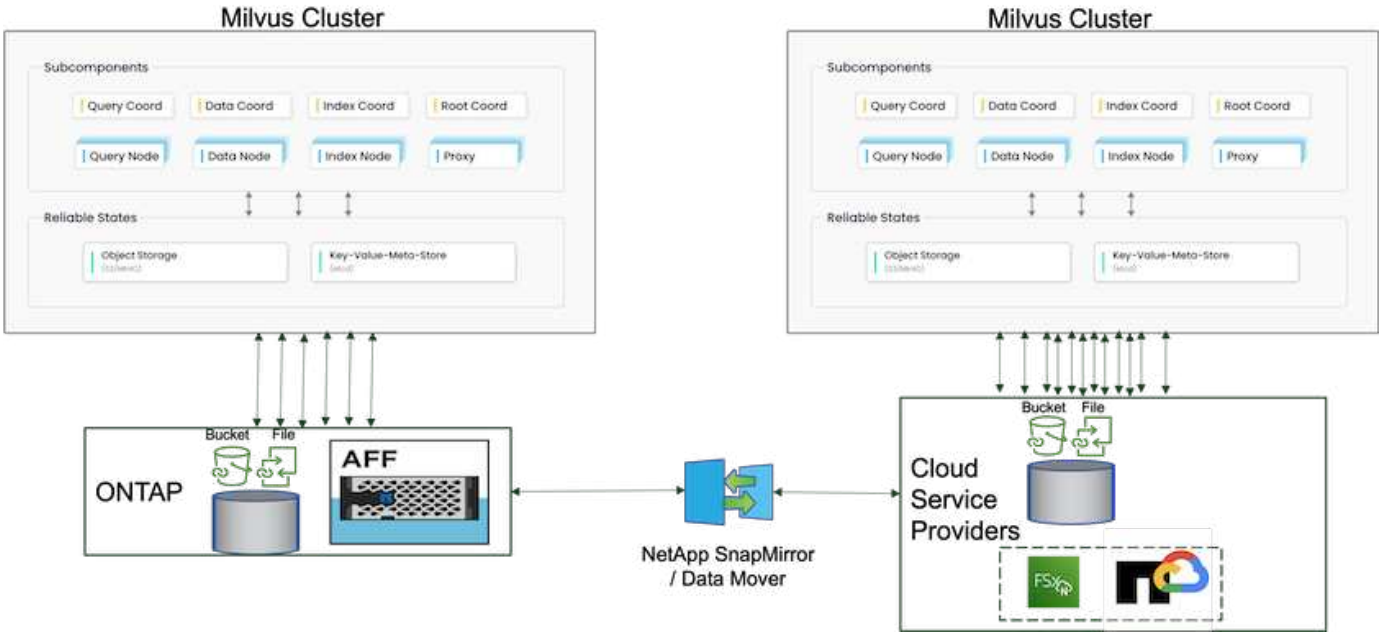
En conclusión, el uso de SnapCenter de NetApp para proteger los datos de bases de datos vectoriales y los datos de Milvus que residen en ONTAP ofrece importantes ventajas a los clientes, en particular en sectores donde la integridad de datos es primordial, como la producción cinematográfica. La capacidad de SnapCenter para crear backups coherentes y realizar restauraciones completas de datos garantiza que los datos cruciales, como los archivos de vídeo y audio integrados, estén protegidos frente a pérdidas causadas por fallos en el disco duro u otros problemas. Esto no solo evita la interrupción operativa, sino que también protege contra pérdidas financieras sustanciales.

En esta sección, demostramos cómo se puede configurar SnapCenter para proteger los datos que residen en ONTAP, incluida la configuración de hosts, la instalación y configuración de complementos de almacenamiento y la creación de un recurso para la base de datos vectorial con un nombre de snapshot personalizado. También hemos mostrado cómo realizar un backup utilizando la snapshot del grupo de consistencia y cómo verificar los datos en el bucket NAS S3.

Además, simulamos un escenario en el que se creó una recopilación innecesaria o inapropiada después de la copia de seguridad. En estos casos, la capacidad de SnapCenter para realizar una restauración completa de una snapshot anterior garantiza que la base de datos vectorial pueda revertirse a su estado antes de que se añada la nueva recogida, manteniendo así la integridad de la base de datos. Esta funcionalidad para restaurar datos a un momento específico es inestimable para los clientes, ya que les garantiza que sus datos no sólo son seguros, sino que también se mantienen de manera correcta. Por ello, el producto SnapCenter de NetApp ofrece a los clientes una solución sólida y fiable para la gestión y la protección de datos.

Recuperación ante desastres con SnapMirror de NetApp

Recuperación ante desastres con SnapMirror de NetApp



La recuperación ante desastres es crucial para mantener la integridad y disponibilidad de una base de datos vectorial, especialmente teniendo en cuenta su papel en la gestión de datos de alta dimensión y la ejecución de búsquedas complejas de similitud. Una estrategia de recuperación ante desastres bien planificada e implementada garantiza que los datos no se pierdan o se vean comprometidos en caso de incidentes imprevistos, como errores de hardware, desastres naturales o ciberataques. Esto es especialmente importante en el caso de aplicaciones que utilizan bases de datos de vectores, en las que la pérdida o el daño de los datos podría provocar interrupciones operativas significativas y pérdidas financieras. Además, contar con un plan sólido de recuperación tras siniestros también garantiza la continuidad del negocio, ya que minimiza el tiempo de inactividad y permite restaurar los servicios con rapidez. Esto se consigue gracias al producto de replicación de datos SnapMirrror de NetApp a través de diferentes ubicaciones geográficas, backups periódicos y mecanismos de recuperación tras fallos. Por lo tanto, la recuperación ante desastres no es solo una medida protectora, sino un componente crítico para la gestión responsable y eficiente de bases de datos vectoriales.

SnapMirror de NetApp proporciona replicación de datos desde una controladora de almacenamiento de NetApp ONTAP a otra, principalmente utilizada para la recuperación ante desastres (DR) y las soluciones híbridas. En el contexto de una base de datos vectorial, esta herramienta facilita una transición de datos fluida entre entornos de cloud y en las instalaciones. Esta transición se consigue sin la necesidad de convertir datos ni refactorizar aplicaciones, lo que mejora la eficiencia y la flexibilidad de la gestión de datos en múltiples plataformas.

La solución híbrida de NetApp en un escenario de base de datos vectorial puede ofrecer más ventajas:

1. Escalabilidad: La solución de cloud híbrido de NetApp ofrece la capacidad de escalar sus recursos según sus requisitos. Puede utilizar los recursos en las instalaciones para cargas de trabajo regulares y previsibles y recursos cloud como Amazon FSxN para NetApp ONTAP y Google Cloud NetApp Volume (GCNV) para tiempos pico o cargas inesperadas.
2. Eficiencia de costes: El modelo de cloud híbrido de NetApp le permite optimizar los costes al usar recursos en las instalaciones para cargas de trabajo normales y pagar solo por recursos cloud cuando los necesite. Este modelo de pago por uso puede ser bastante rentable con una oferta de servicio de Instaclutr de NetApp. Para los principales proveedores de servicios en el cloud y en las instalaciones, instaclustr proporciona soporte y consultoría.
3. Flexibilidad: El cloud híbrido de NetApp le da la flexibilidad de elegir dónde procesar sus datos. Por ejemplo, podría optar por realizar operaciones de vectores complejas en las instalaciones donde dispone

de un hardware más potente y las operaciones menos intensivas en el cloud.

4. Continuidad del negocio: En caso de desastre, tener los datos en un cloud híbrido de NetApp puede garantizar la continuidad del negocio. Puede cambiar rápidamente a la nube si sus recursos en las instalaciones están afectados. Podemos aprovechar SnapMirror de NetApp para mover los datos del on-premises al cloud y viceversa.
5. Innovación: Las soluciones de cloud híbrido de NetApp también hacen posible una innovación más rápida al proporcionar acceso a servicios y tecnologías de cloud de vanguardia. Las innovaciones de NetApp en cloud, como Amazon FSxN para NetApp ONTAP, Azure NetApp Files y Google Cloud NetApp Volumes, son proveedores de servicios cloud de productos innovadores y NAS preferidos.

Validación del rendimiento de la base de datos vectorial

Validación del rendimiento

La validación del rendimiento desempeña un papel fundamental tanto en bases de datos vectoriales como en sistemas de almacenamiento, ya que sirve como factor clave para garantizar un funcionamiento óptimo y una utilización eficiente de los recursos. Las bases de datos vectoriales, conocidas por manejar datos de alta dimensión y ejecutar búsquedas de similitud, necesitan mantener altos niveles de rendimiento para procesar consultas complejas con rapidez y precisión. La validación del rendimiento ayuda a identificar cuellos de botella, ajustar las configuraciones y garantizar que el sistema pueda gestionar las cargas esperadas sin degradación del servicio. Del mismo modo, en los sistemas de almacenamiento, la validación del rendimiento es esencial para garantizar que los datos se almacenan y recuperan de manera eficiente, sin problemas de latencia ni cuellos de botella que puedan afectar al rendimiento general del sistema. También ayuda a tomar decisiones informadas acerca de las actualizaciones o los cambios necesarios en la infraestructura de almacenamiento. Por lo tanto, la validación del rendimiento es un aspecto crucial de la gestión del sistema, ya que contribuye significativamente al mantenimiento de la alta calidad del servicio, la eficiencia operativa y la fiabilidad general del sistema.

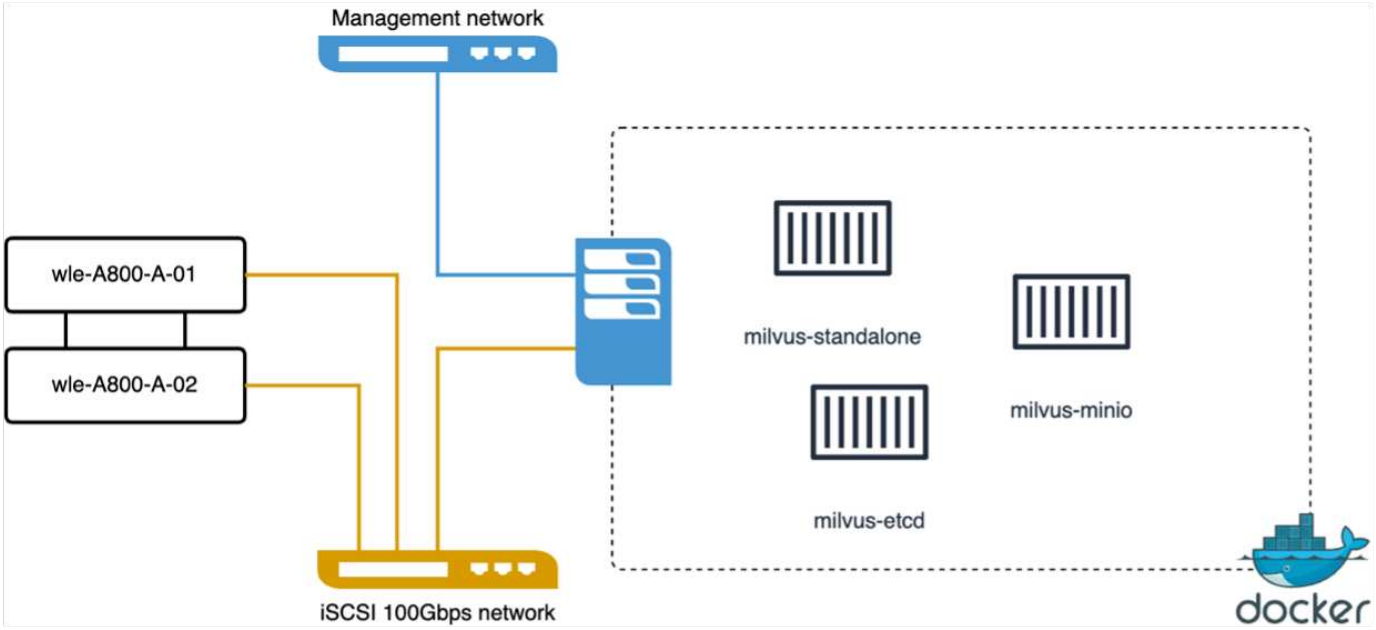
En este apartado, el objetivo es profundizar en la validación del rendimiento de las bases de datos vectoriales, como Milvus y pgvecto.rs, centrándose en sus características de rendimiento del almacenamiento, como el perfil de I/O y el comportamiento de la controladora de almacenamiento NetApp, para admitir cargas de trabajo de inferencia y RAG dentro del ciclo de vida del LLM. Evaluaremos e identificaremos cualquier diferenciación en el rendimiento cuando estas bases de datos se combinen con la solución de almacenamiento de ONTAP. Nuestro análisis se basará en indicadores clave de rendimiento, como el número de consultas procesadas por segundo (QPS).

Compruebe la metodología utilizada para el milvus y el progreso a continuación.

Detalles	Milvus (independiente y clúster)	Postgres(pgvecto.rs)
versión	2.3.2	0.2.0
Sistema de archivos	XFS en LUN iSCSI	
Generador de cargas de trabajo	"VectorDB-Banco" – v0,0.5	
Conjuntos de datos	Conjunto de datos de LAION * 10Million incrustaciones * 768 Dimensiones * ~300GB tamaño del conjunto de datos	

VectorDB-Bench con clúster independiente Milvus

Realizamos la siguiente validación de rendimiento en cluster independiente de Milvus con vectorDB-Bench. La conectividad de red y servidor del clúster independiente de Milvus es inferior.



En esta sección, compartimos nuestras observaciones y resultados de las pruebas de la base de datos independiente de Milvus.
. Seleccionamos DiskANN como el tipo de índice para estas pruebas.
. La ingesta, optimización y creación de índices para un conjunto de datos de aproximadamente 100GB TB llevó alrededor de 5 horas. Durante la mayor parte de esta duración, el servidor Milvus, equipado con 20 núcleos (lo que equivale a 40 vcpu cuando Hyper-Threading está activado), estaba funcionando a su capacidad máxima de CPU del 100%.Se encontró que DiskANN es particularmente importante para grandes conjuntos de datos que exceden el tamaño de la memoria del sistema.
. En la fase de consulta, se observó una tasa de consultas por segundo (QPS) de 10,93 con una recuperación de 0,9987. La latencia de percentil de 99th ms para consultas se midió a 708,2 milisegundos.

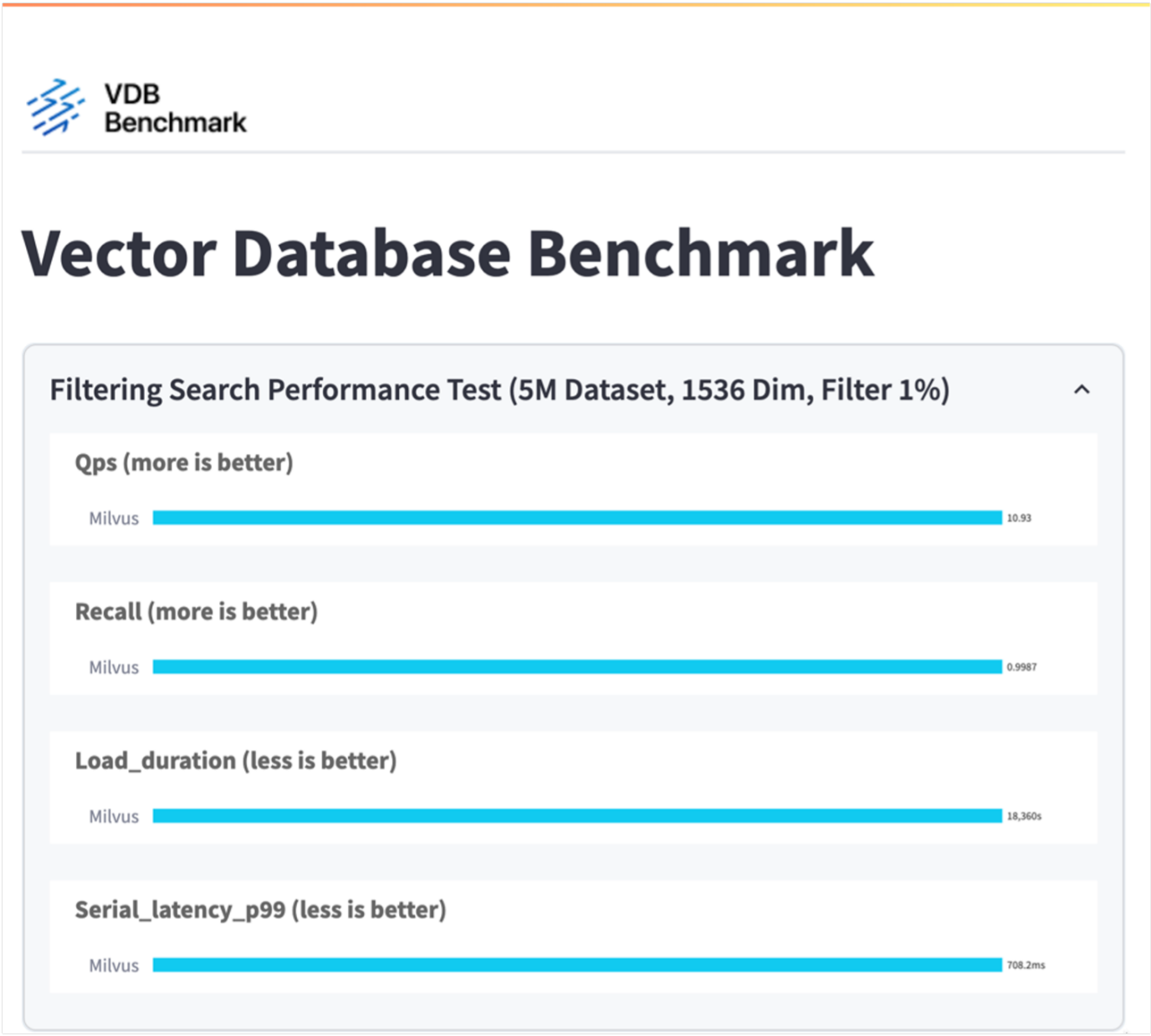
Desde el punto de vista del almacenamiento, la base de datos emitió unas 1.000 operaciones por segundo durante las fases de procesamiento, optimización posterior a la inserción y creación de índices. En la fase de consulta, demandó 32.000 ops/s.

El siguiente apartado presenta las métricas de rendimiento del almacenamiento.

Fase de carga de trabajo	Métrico	Valor
Ingesta de datos y.. Optimización posterior a la inserción	IOPS	< 1.000
	Latencia	< 400 usuarios
	Carga de trabajo	Combinación de lectura/escritura, principalmente escrituras
	Tamaño de I/O.	64KB
Consulta	IOPS	Pico a 32.000

Fase de carga de trabajo	Métrico	Valor
	Latencia	< 400 usuarios
	Carga de trabajo	100 % de lectura en caché
	Tamaño de I/O.	Principalmente 8KB

El resultado vectorDB-BENCH está abajo.



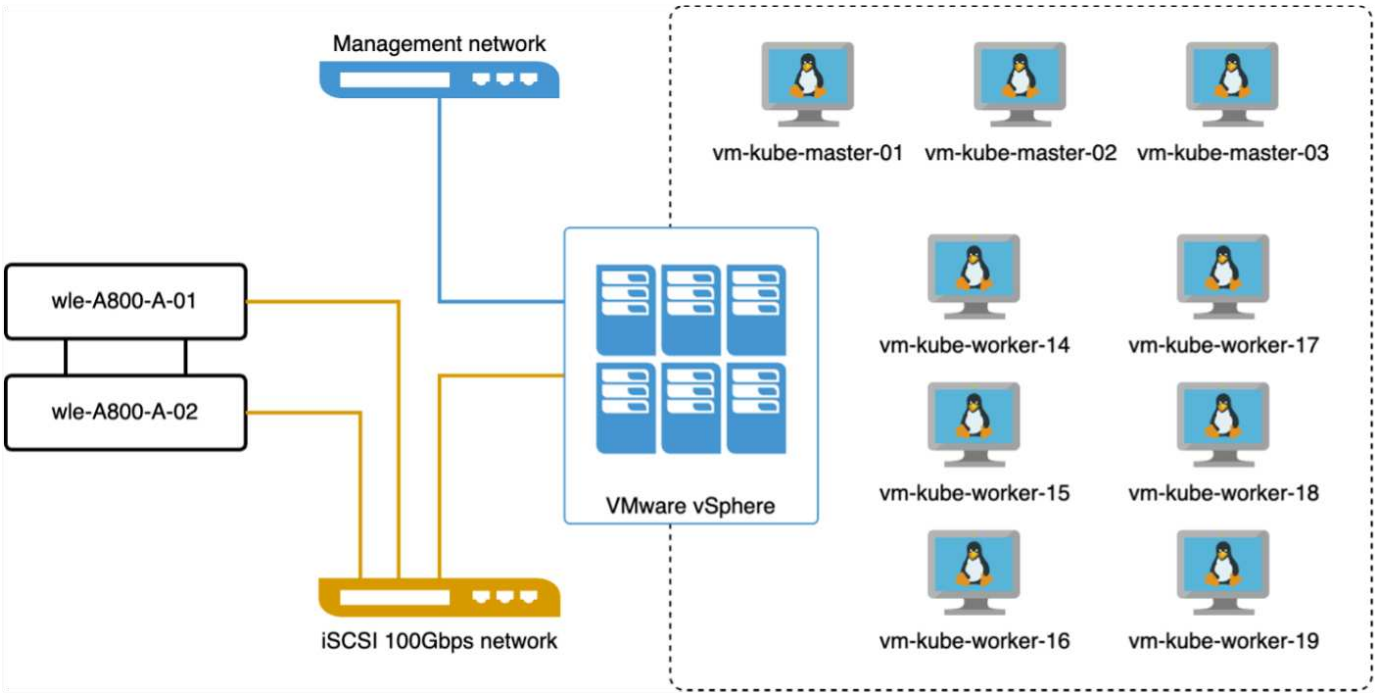
A partir de la validación del rendimiento de la instancia independiente de Milvus, es evidente que la configuración actual es insuficiente para admitir un conjunto de datos de 5 millones de vectores con una dimensionalidad de 1536. hemos determinado que el almacenamiento posee los recursos adecuados y no constituye un cuello de botella en el sistema.

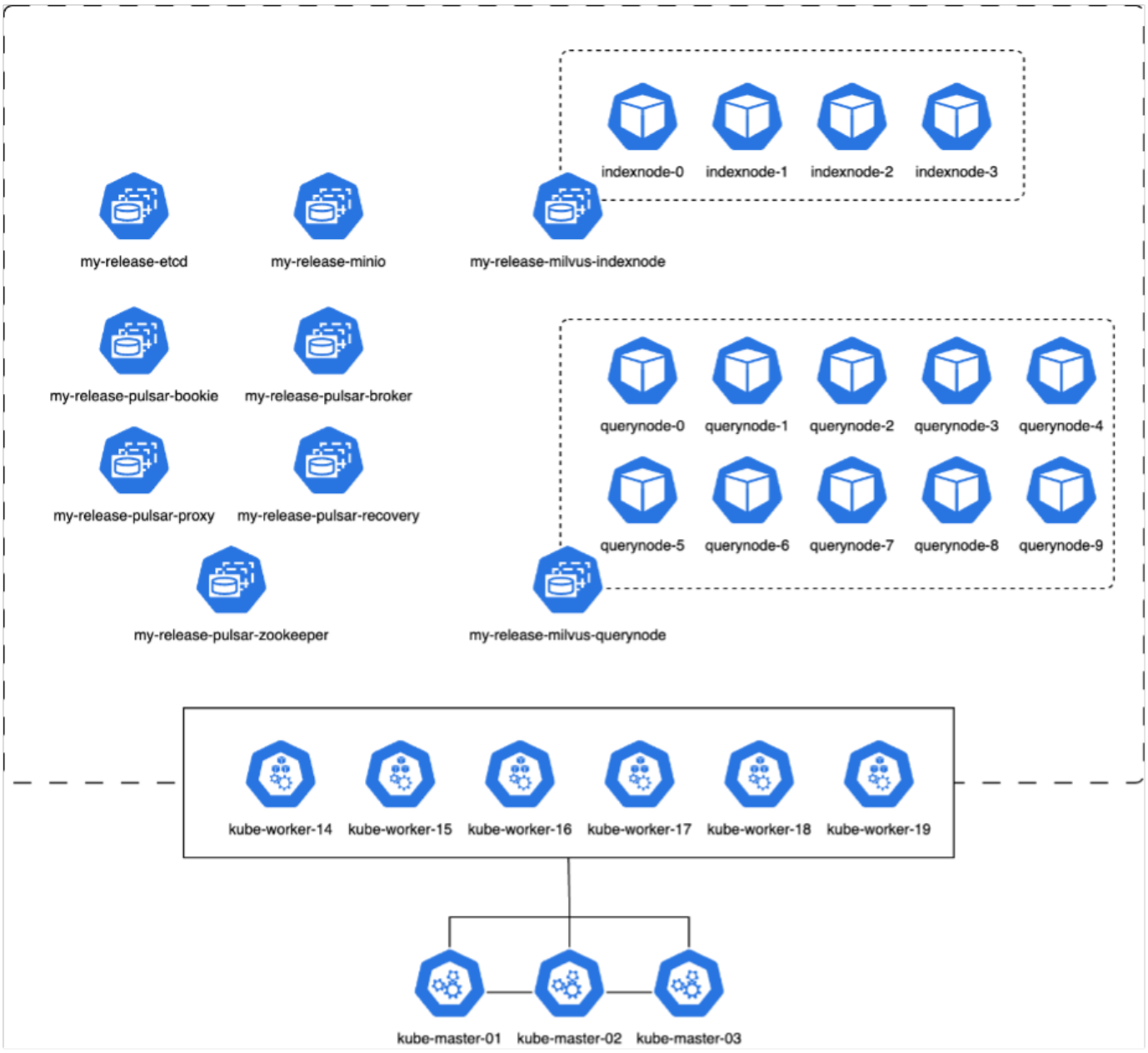
VectorDB-Banco con clúster milvus

En esta sección trataremos la puesta en marcha de un clúster de Milvus en un entorno de Kubernetes. Esta

configuración de Kubernetes se construyó sobre una puesta en marcha de VMware vSphere, que alojaba los nodos maestro y trabajador de Kubernetes.

Los detalles de las implementaciones de VMware vSphere y Kubernetes se presentan en las siguientes secciones.





En esta sección, presentamos nuestras observaciones y resultados de las pruebas de la base de datos Milvus.
* El tipo de índice utilizado fue DiskANN.
* La siguiente tabla proporciona una comparación entre los despliegues independientes y de clúster cuando se trabaja con 5 millones de vectores en una dimensionalidad de 1536. Observamos que el tiempo necesario para la ingesta de datos y la optimización posterior a la inserción era menor en la puesta en marcha del clúster. La latencia de percentil del 99th para consultas se redujo seis veces en la puesta en marcha del clúster en comparación con la configuración independiente.
* Aunque la tasa de consultas por segundo (QPS) era más alta en la implementación del cluster, no estaba en el nivel deseado.

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%

Las siguientes imágenes ofrecen una vista de diversas métricas de almacenamiento, incluida la latencia del clúster de almacenamiento y las IOPS totales (operaciones de entrada/salida por segundo).

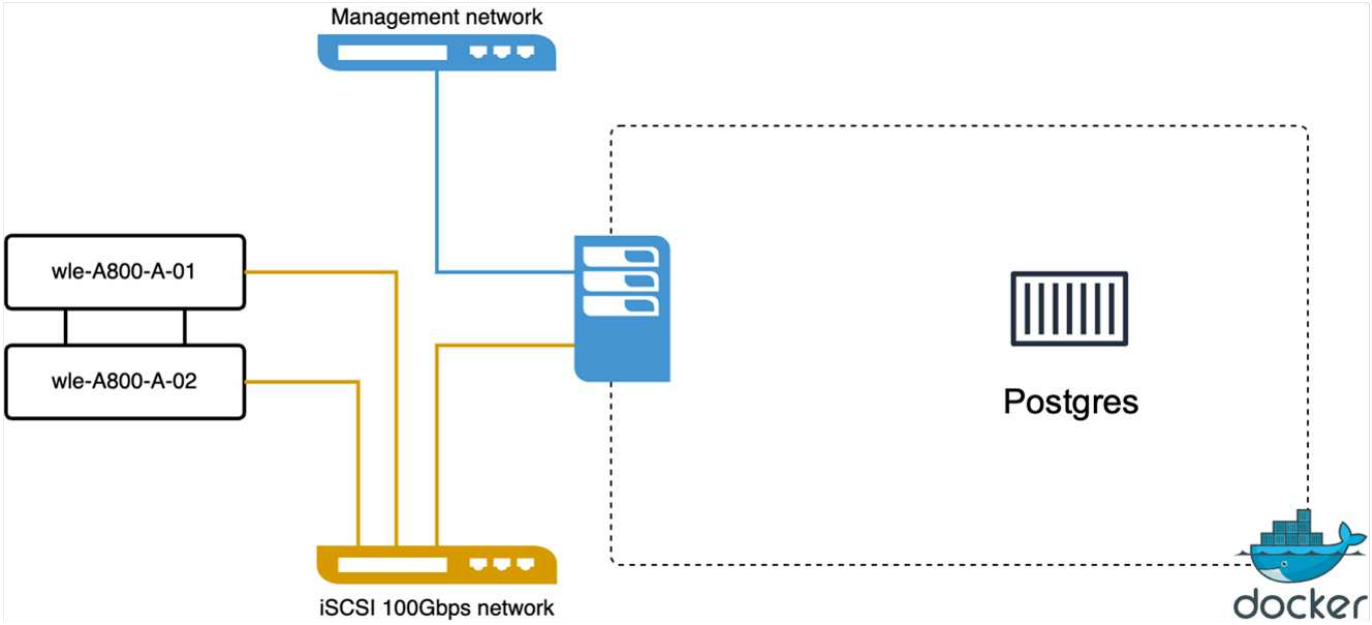
La siguiente sección presenta las métricas clave de rendimiento del almacenamiento.

Fase de carga de trabajo	Métrico	Valor
Ingesta de datos y.. Optimización posterior a la inserción	IOPS	< 1.000
	Latencia	< 400 usuarios
	Carga de trabajo	Combinación de lectura/escritura, principalmente escrituras
	Tamaño de I/O.	64KB
Consulta	IOPS	Pico a 147.000
	Latencia	< 400 usuarios
	Carga de trabajo	100 % de lectura en caché
	Tamaño de I/O.	Principalmente 8KB

Basándonos en la validación del rendimiento tanto del clúster Milvus como del clúster Milvus, presentamos los detalles del perfil de E/S de almacenamiento.
* Observamos que el perfil de E/S permanece consistente tanto en implementaciones independientes como en clusters.
* La diferencia observada en el pico de IOPS se puede atribuir al mayor número de clientes en la implementación del clúster.

Banco vectorDB con Postgres (pgvecto.rs)

Realizamos las siguientes acciones en PostgreSQL(pgvecto.rs) usando VectorDB-Bench:
Los detalles relativos a la conectividad de red y servidor de PostgreSQL (específicamente, pgvecto.rs) son los siguientes:

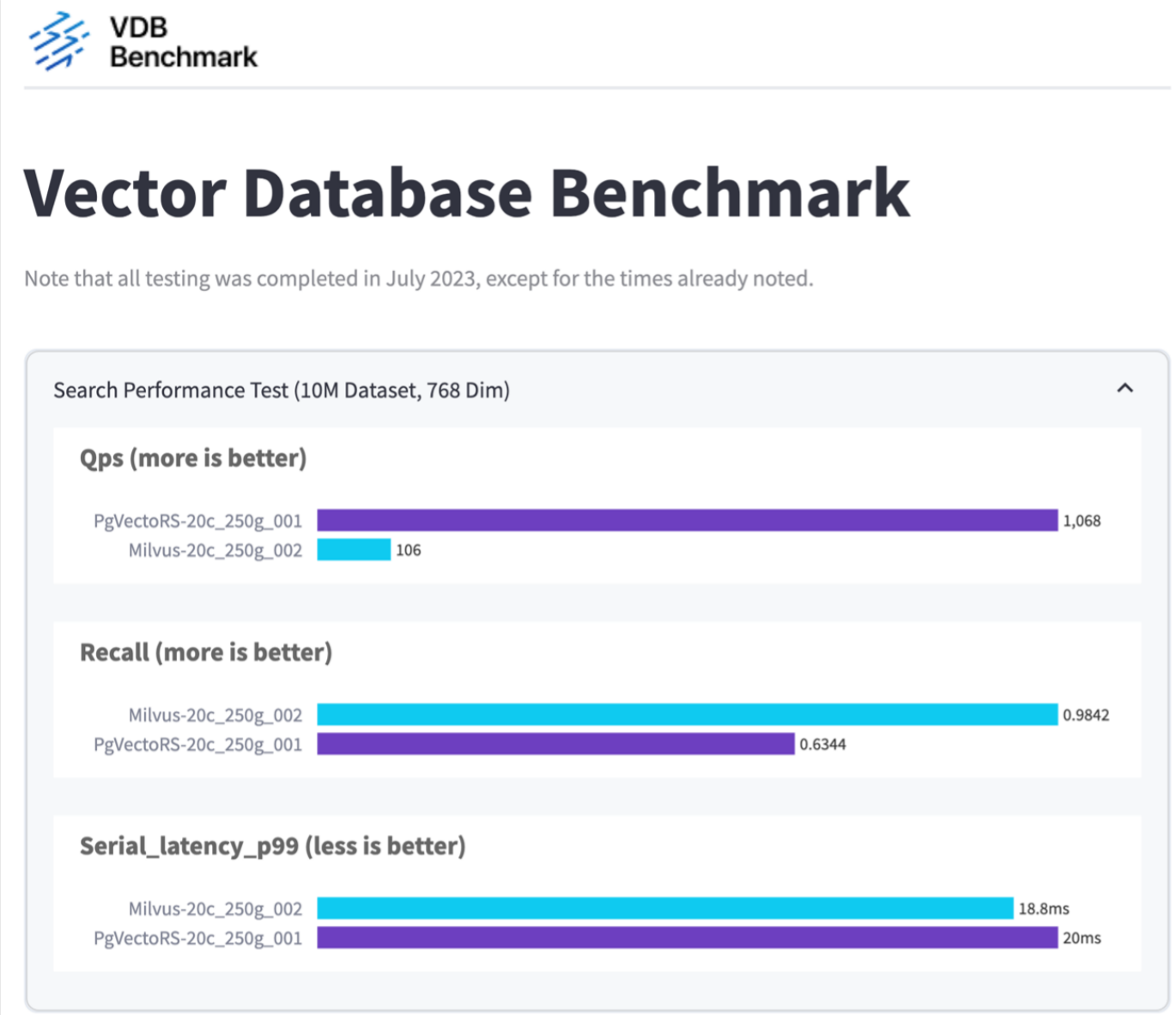


En esta sección, compartimos nuestras observaciones y resultados de la prueba de la base de datos PostgreSQL, específicamente usando pgvecto.rs.
* Seleccionamos HNSW como el tipo de índice para estas pruebas porque en el momento de las pruebas, DiskANN no estaba disponible para pgvecto.rs.
* Durante la fase de ingestión de datos, cargamos el conjunto de datos de cohere, que consta de 10 millones de vectores a una dimensionalidad de 768. Este proceso duró aproximadamente 4,5 horas.
* En la fase de consulta, observamos una tasa de consultas por segundo (QPS) de 1.068 con una recuperación de 0,6344. La latencia de percentil de 99th ms para consultas se midió a 20 milisegundos. Durante la mayor parte del tiempo de ejecución, la CPU del cliente estaba funcionando al 100 % de su capacidad.

Las siguientes imágenes ofrecen una vista de diversas métricas de almacenamiento, incluida la latencia total de IOPS (operaciones de entrada/salida por segundo) del clúster de almacenamiento.


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["Error: Falta la imagen gráfica"]
```

Comparación de rendimiento entre milvus y postgres en vector DB Bench



En base a nuestra validación de rendimiento de Milvus y PostgreSQL usando VectorDBBench, observamos lo siguiente:

- Tipo de índice: HNSW
- Conjunto de datos: Cohere con 10 millones de vectores en 768 dimensiones

Se encontró que pgvecto.rs logró una tasa de consultas por segundo (QPS) de 1.068 con una retirada de 0,6344, mientras que Milvus logró una tasa de QPS de 106 con una retirada de 0,9842.

Si la alta precisión en sus consultas es una prioridad, Milvus supera a pgvecto.rs ya que recupera una mayor proporción de elementos relevantes por consulta. Sin embargo, si el número de consultas por segundo es un factor más crucial, pgvecto.rs excede Milvus. Sin embargo, es importante tener en cuenta que la calidad de los datos recuperados a través de pgvecto.rs es menor, con alrededor del 37% de los resultados de búsqueda siendo elementos irrelevantes.

Observación basada en nuestras validaciones de rendimiento:

Basándonos en nuestras validaciones de rendimiento, hemos realizado las siguientes observaciones:

En Milvus, el perfil de I/O se parece mucho a una carga de trabajo OLTP, como la observada en Oracle SLOB. El punto de referencia consta de tres fases: Ingesta de datos, Post-Optimización y Consulta. Las etapas iniciales se caracterizan principalmente por realizar operaciones de escritura de 64KB KB, mientras que la fase de consulta implica predominantemente lecturas de 8KB KB. Esperamos que ONTAP gestione la carga de I/O de Milvus de manera competente.

El perfil de I/O de PostgreSQL no presenta una carga de trabajo de almacenamiento exigente. Dada la implementación en memoria actualmente en curso, no observamos ninguna E/S de disco durante la fase de consulta.

DiskANN surge como una tecnología vital para la diferenciación del almacenamiento. Permite escalar de forma eficiente la búsqueda de bases de datos vectoriales más allá del límite de memoria del sistema. Sin embargo, es poco probable que establezca una diferenciación del rendimiento del almacenamiento con los índices de bases de datos vectoriales en memoria, como HNSW.

También vale la pena señalar que el almacenamiento no juega un papel crítico durante la fase de consulta cuando el tipo de índice es HSNW, que es la fase operativa más importante para las bases de datos vectoriales que soportan aplicaciones RAG. Lo que implica aquí es que el rendimiento del almacenamiento no afecta significativamente al rendimiento general de estas aplicaciones.

Vector de base de datos con Instaclustr usando PostgreSQL: Pgvector

Vector de base de datos con Instaclustr usando PostgreSQL: Pgvector

En esta sección, profundizamos en los detalles de cómo el producto instaclustr se integra con PostgreSQL en pgvector fuctionality. Tenemos un ejemplo de “Cómo mejorar su precisión y rendimiento de LLM con PGVector y PostgreSQL®: Introducción a las incrustaciones y el papel de PGVector”. Compruebe la "blog" para obtener más información.

Casos de uso de bases de datos vectoriales

Casos de uso de bases de datos vectoriales

En esta sección, discutimos sobre dos casos de uso como la Recuperación de Generación Aumentada con Modelos de Lenguaje Grande y el chatbot NetApp IT.

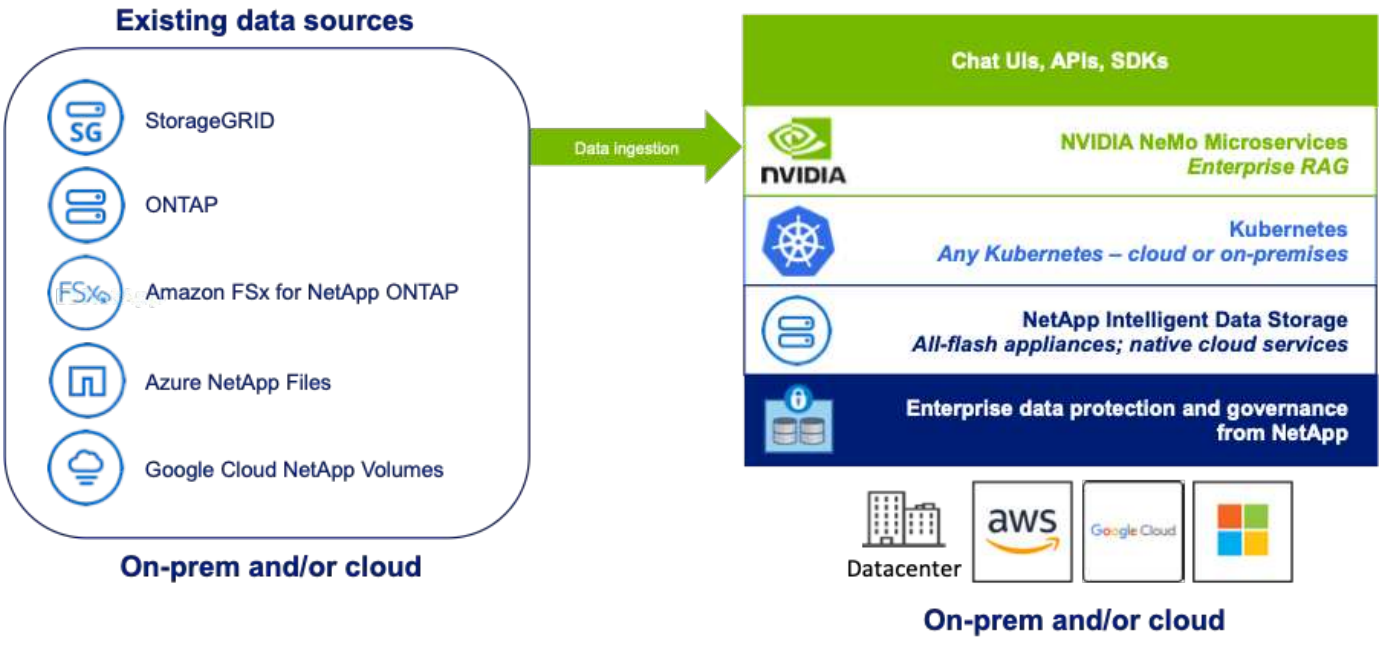
Generación Aumentada de Recuperación (RAG) con Modelos de Lenguaje Grande (LLMs)

```
Retrieval-augmented generation, or RAG, is a technique for enhancing the
accuracy and reliability of Large Language Models, or LLMs, by augmenting
prompts with facts fetched from external sources. In a traditional RAG
deployment, vector embeddings are generated from an existing dataset and
then stored in a vector database, often referred to as a knowledgebase.
Whenever a user submits a prompt to the LLM, a vector embedding
representation of the prompt is generated, and the vector database is
searched using that embedding as the search query. This search operation
returns similar vectors from the knowledgebase, which are then fed to the
LLM as context alongside the original user prompt. In this way, an LLM can
be augmented with additional information that was not part of its original
training dataset.
```

El operador NVIDIA Enterprise RAG LLM es una herramienta útil para implementar RAG en la empresa. Este operador se puede utilizar para desplegar un pipeline RAG completo. La canalización de RAG se puede personalizar para utilizar Milvus o pgvecto como base de datos vectorial para almacenar incrustaciones en la base de conocimientos. Consulte la documentación para obtener más detalles.

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

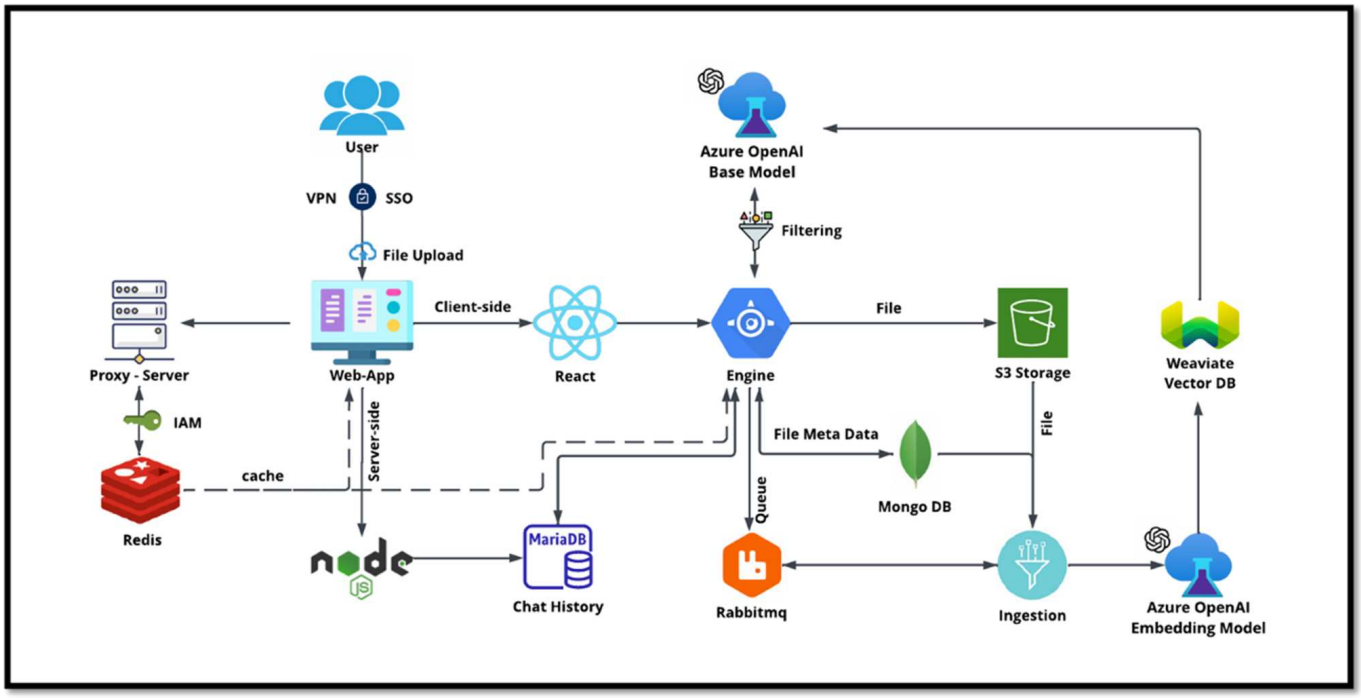
Figura 1) RAG empresarial con la tecnología de microservicios Nemo de NVIDIA y NetApp



Caso de uso de NetApp IT Chatbot

El chatbot de NetApp sirve como otro caso de uso en tiempo real para la base de datos vectorial. En este

ejemplo, el Sandbox de OpenAI Privado de NetApp proporciona una plataforma eficaz, segura y eficiente para gestionar las consultas de los usuarios internos de NetApp. Al incorporar protocolos de seguridad estrictos, sistemas de gestión de datos eficientes y sofisticadas capacidades de procesamiento de IA, garantiza respuestas precisas y de alta calidad a los usuarios en función de sus funciones y responsabilidades en la organización a través de la autenticación SSO. Esta arquitectura destaca el potencial de la fusión de tecnologías avanzadas para crear sistemas inteligentes y centrados en el usuario.



El caso de uso puede dividirse en cuatro secciones primarias.

Autenticación y verificación de usuario:

- Las consultas de usuario primero pasan por el proceso de inicio de sesión único (SSO) de NetApp para confirmar la identidad del usuario.
- Después de una autenticación exitosa, el sistema comprueba la conexión VPN para garantizar una transmisión de datos segura.

Transmisión y procesamiento de datos:

- Una vez validada la VPN, los datos se envían a MariaDB a través de las aplicaciones web NetAIChat o NetAICreate. MariaDB es un sistema de base de datos rápido y eficaz que se utiliza para gestionar y almacenar los datos de los usuarios.
- A continuación, MariaDB envía la información a la instancia de Azure de NetApp, que conecta los datos de usuario a la unidad de procesamiento de IA.

Interacción con OpenAI y filtrado de contenido:

- La instancia de Azure envía las preguntas del usuario a un sistema de filtrado de contenido. Este sistema limpia la consulta y la prepara para su procesamiento.
- La entrada limpiada se envía entonces al modelo base de Azure OpenAI, que genera una respuesta basada en la entrada.

Generación de respuesta y moderación:

- La respuesta del modelo base se comprueba primero para asegurarse de que es precisa y cumple con los estándares de contenido.
- Después de pasar la comprobación, la respuesta se envía de vuelta al usuario. Este proceso garantiza que el usuario reciba una respuesta clara, precisa y adecuada a su consulta.

Conclusión

Conclusión

En conclusión, este documento ofrece una visión general completa para la puesta en marcha y la gestión de bases de datos vectoriales, como Milvus y pgvector, en las soluciones de almacenamiento de NetApp. Hemos tratado las directrices de infraestructura para aprovechar el almacenamiento de objetos NetApp ONTAP y StorageGRID y validado la base de datos de Milvus en AWS FSx para NetApp ONTAP mediante almacén de archivos y objetos.

Exploramos la dualidad archivo-objeto de NetApp, lo cual demuestra su utilidad no solo para datos en bases de datos vectoriales, sino también para otras aplicaciones. También hemos destacado cómo SnapCenter, el producto de gestión empresarial de NetApp, ofrece funcionalidades de backup, restauración y clonado para datos de bases de datos vectoriales, garantizando así la integridad y la disponibilidad de los datos.

El documento también analiza cómo la solución de cloud híbrido de NetApp ofrece la replicación y la protección de datos en entornos en las instalaciones y de cloud, lo que proporciona una experiencia de gestión de datos segura y fluida. Proporcionamos información sobre la validación del rendimiento de bases de datos vectoriales como Milvus y pgvecto en NetApp ONTAP, ofreciendo información valiosa sobre su eficiencia y escalabilidad.

Por último, hemos hablado de dos casos de uso de IA generativos: RAG con LLM y la ChatAI interna de NetApp. Estos ejemplos prácticos subrayan las aplicaciones y beneficios del mundo real de los conceptos y prácticas descritos en este documento. En general, este documento sirve como guía completa para cualquiera que desee aprovechar las potentes soluciones de almacenamiento de NetApp para la gestión de bases de datos vectoriales.

Reconocimientos

Al autor le gusta expresar su más sincero agradecimiento a los colaboradores que aparecen a continuación, a otros que han proporcionado sus comentarios y comentarios para que este documento sea valioso para los clientes de NetApp y los campos de NetApp.

1. Sathish Thyagarajan, Ingeniero de Marketing Técnico, ONTAP AI & Analytics, NetApp
2. Mike Oglesby, Ingeniero Técnico de Marketing de NetApp
3. AJ Mahajan, director sénior de NetApp
4. Joe Scott, director de ingeniería de rendimiento de cargas de trabajo de NetApp
5. Puneet Dhawan, director sénior de gestión de productos FSx, NetApp
6. Yuval Kalderon, director sénior de producto, Equipo de producto de FSx, NetApp

Dónde encontrar información adicional

Si quiere más información sobre el contenido de este documento, consulte los siguientes documentos o sitios web:

- Documentación de Milvus - https://milvus.io/docs/overview.md
- Documentación independiente de Milvus - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- Documentación de productos de NetApp
  https://www.netapp.com/support-and-training/documentation/
- instaclustr - "documentación de instalación"

Historial de versiones

Versión	Fecha	Historial de versiones del documento
Versión 1.0	Abril de 2024	Versión inicial

Apéndice A: Valores.yaml

Apéndice A: Valores.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
##
```

```yaml
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:~#
```

Apéndice B: prepare_data_netapp_new.py

Apéndice 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:~#
```

Apéndice C: verify_data_netapp.py

Apéndice 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:~#
```

Apéndice D: docker-compose.yml

Apéndice 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
```