# Descripción general de la verificación de la solución

## NetApp Solutions

NetApp
May 03, 2024

# Tabla de contenidos

# 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

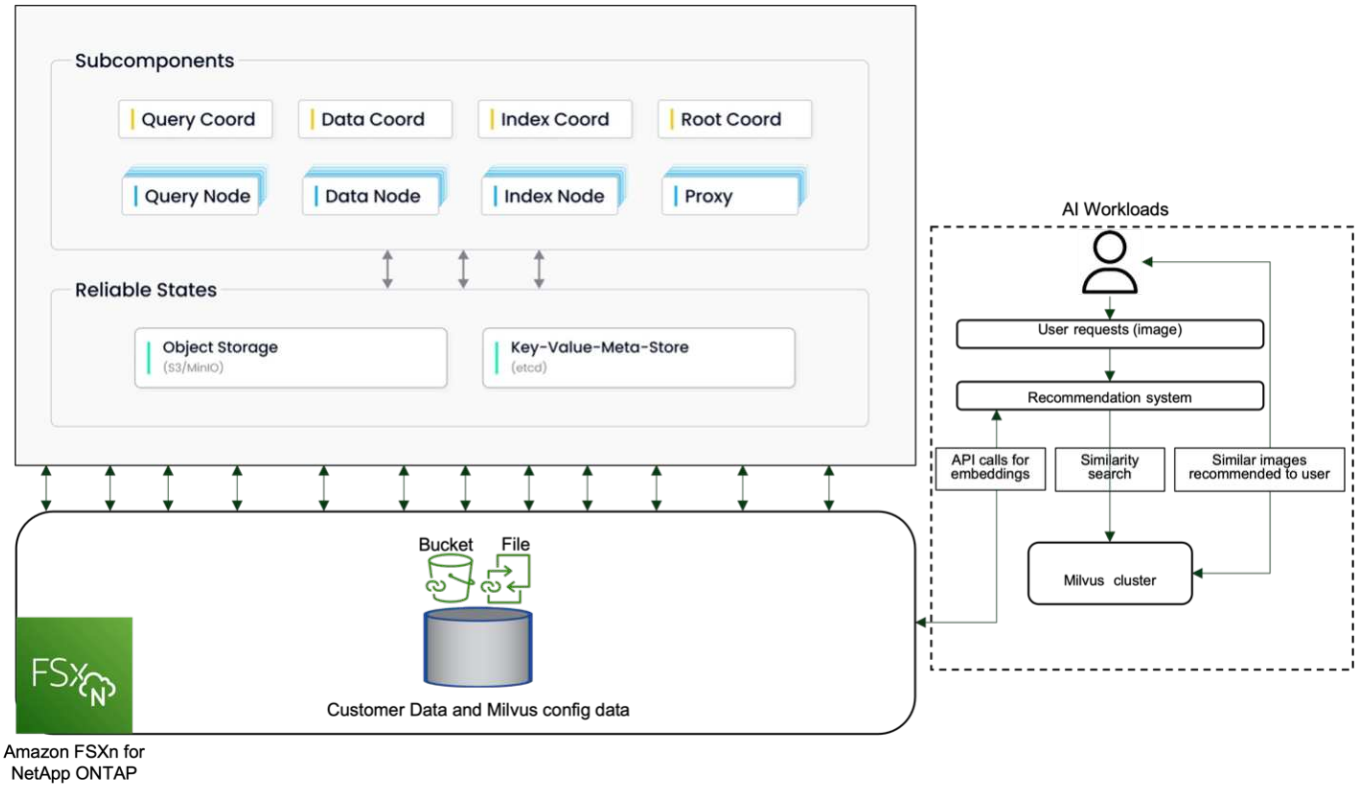
de los recursos y una gestión de datos. El proceso de verificación del estado de PersistentVolumeClaims (RVP) y Pods, así como la realización de pruebas de escritura y lectura de datos, proporciona a los clientes la garantía de realizar operaciones de datos fiables y coherentes. El uso del almacenamiento 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.

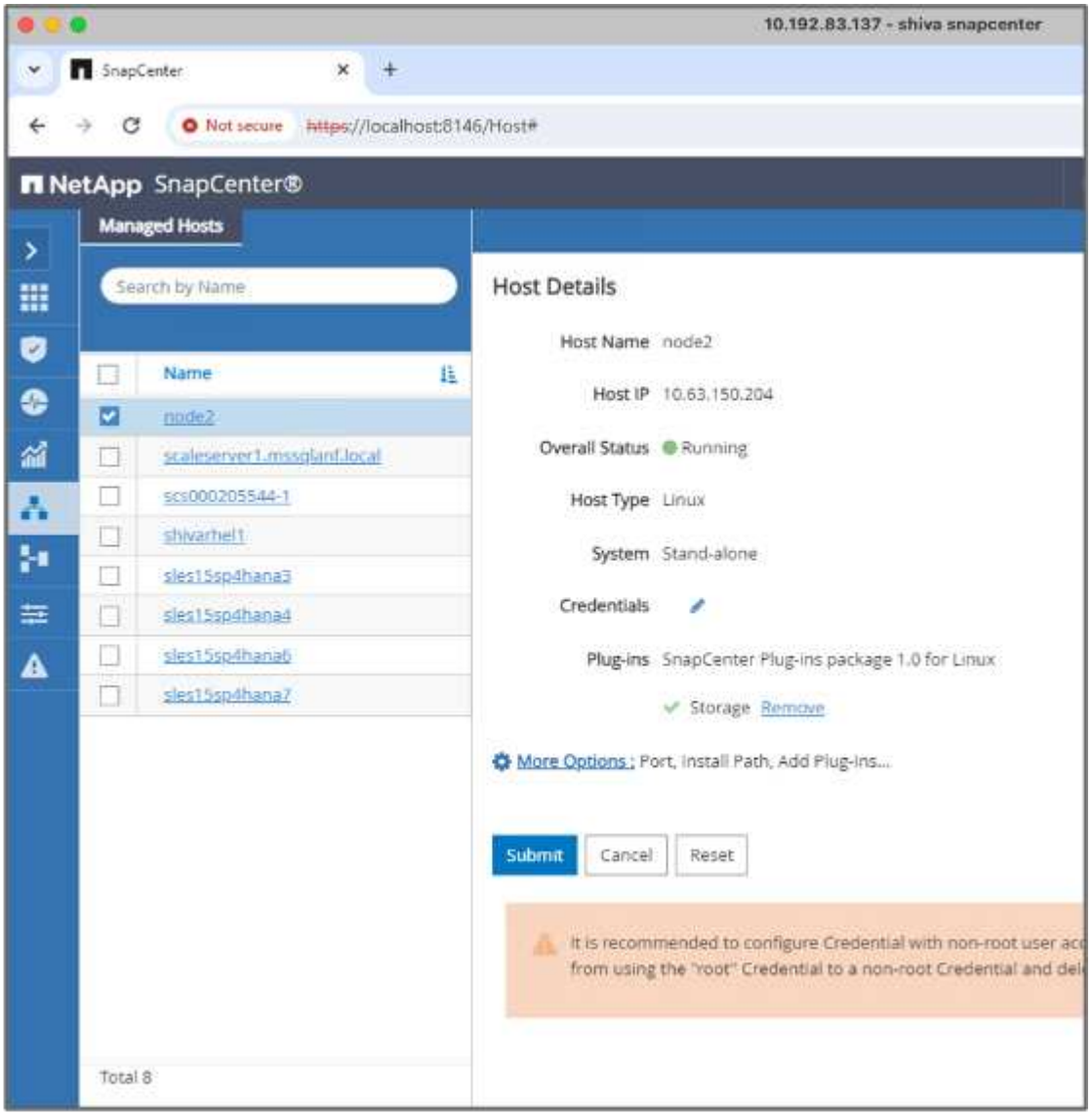
# Protección de bases de datos vectoriales mediante SnapCenter

## Protección de bases de datos vectoriales mediante NetApp SnapCenter.

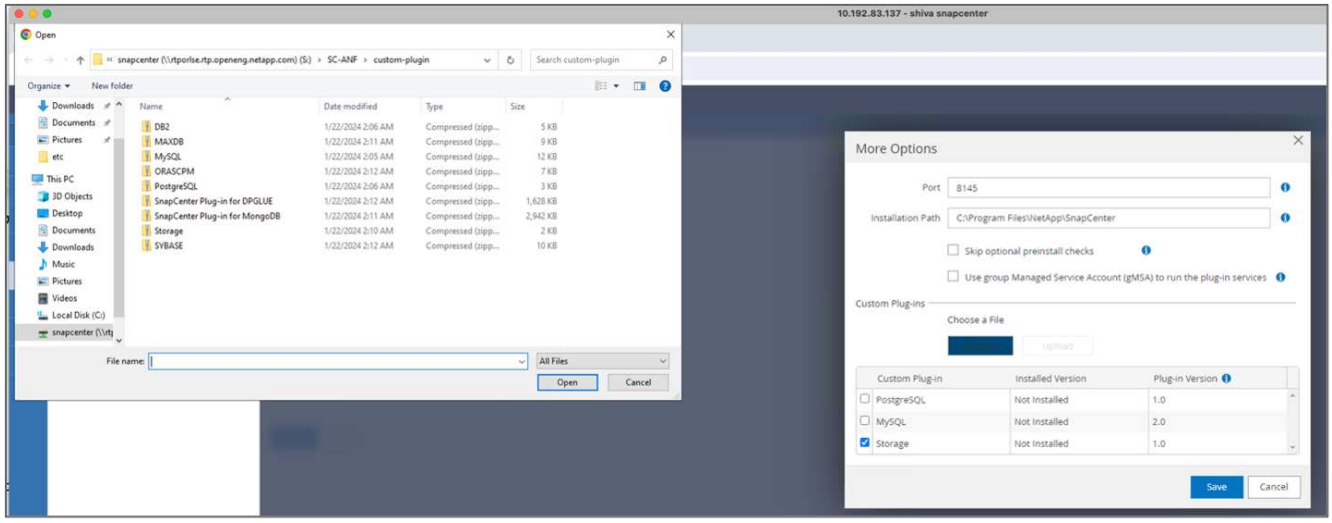
Por ejemplo, en el sector de producción cinematográfica, los clientes suelen poseer datos incrustados

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

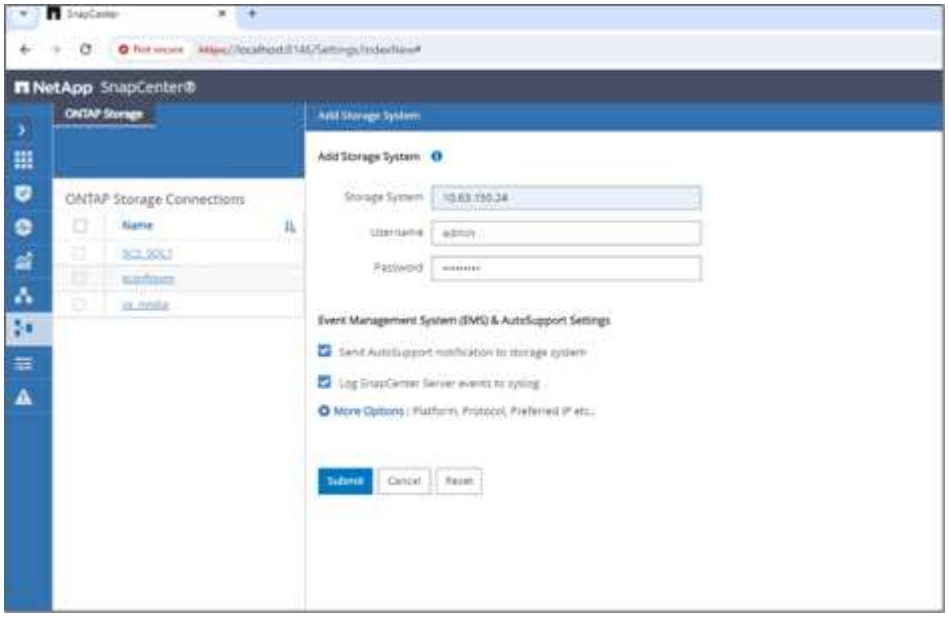
1. Configure el host que se utilizará para ejecutar comandos SnapCenter.



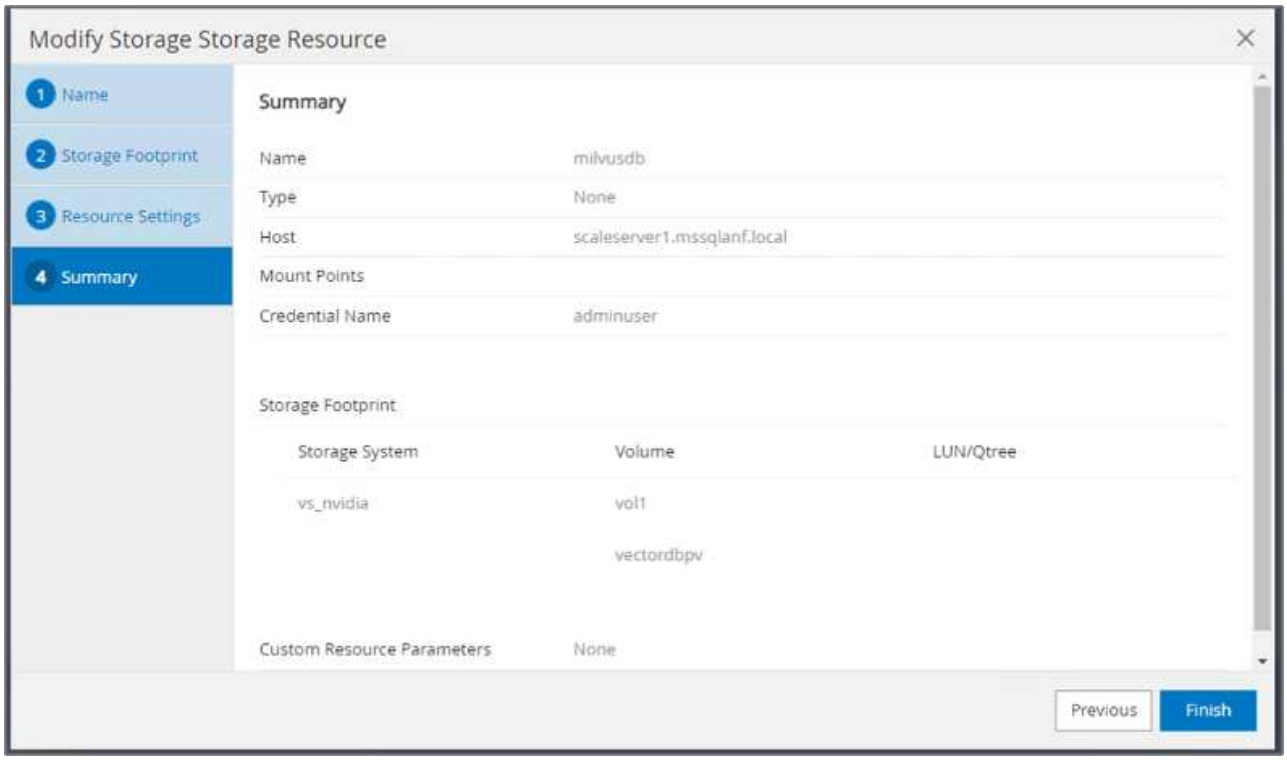

2. Instale y configure el complemento de almacenamiento. En el host agregado, seleccione “Más opciones”. Navegue hasta y seleccione el plugin de almacenamiento descargado en "Automation Store de NetApp". Instale el plugin y guarde la configuración.




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.



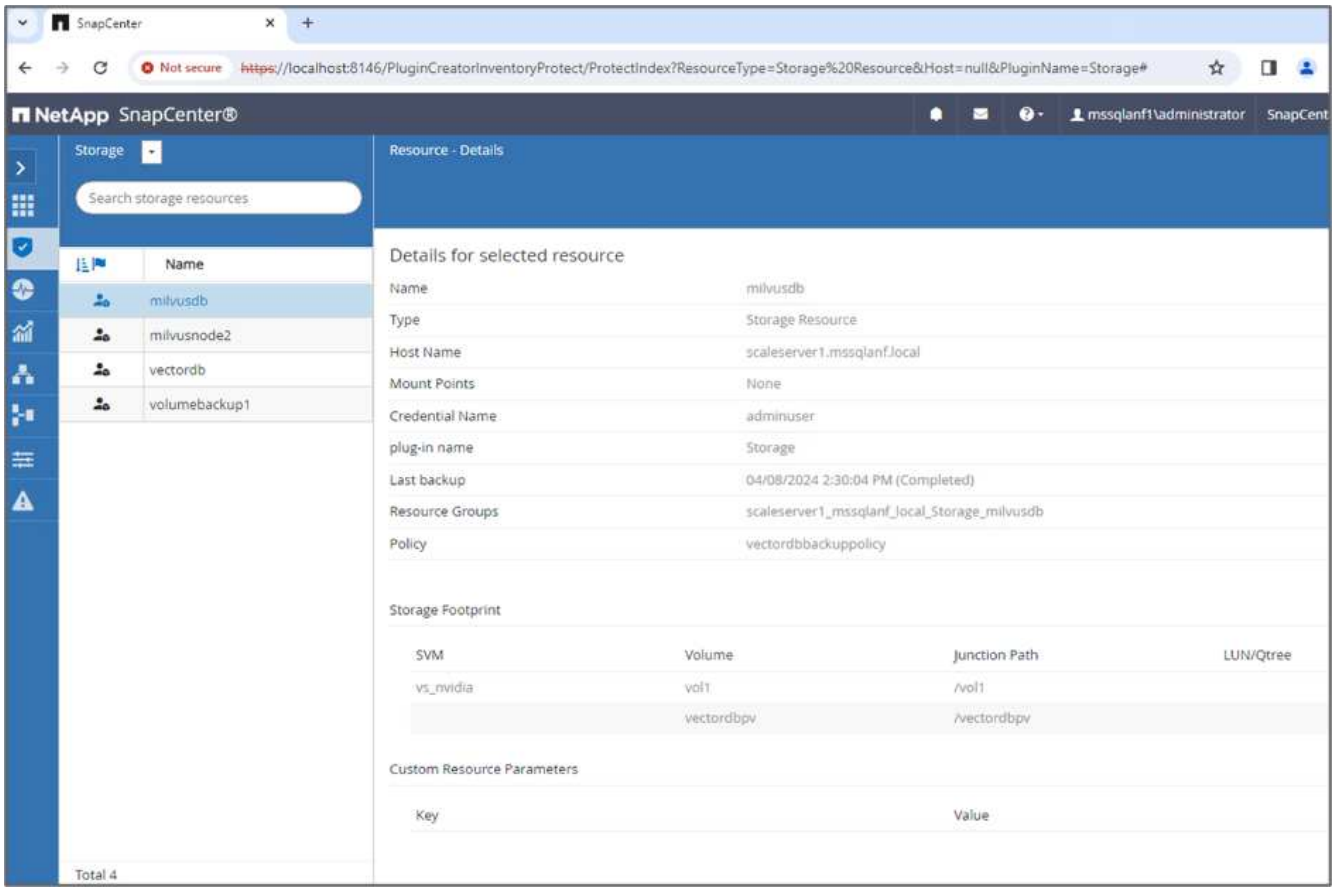

5. Inserte los datos en el bloque NAS de S3 mediante un script de Python. En nuestro caso, modificamos el script de backup proporcionado por Milvus, a saber, 'prepare_data_netapp.py', y ejecutamos el comando 'ync' para vaciar los datos del sistema operativo.

```
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'

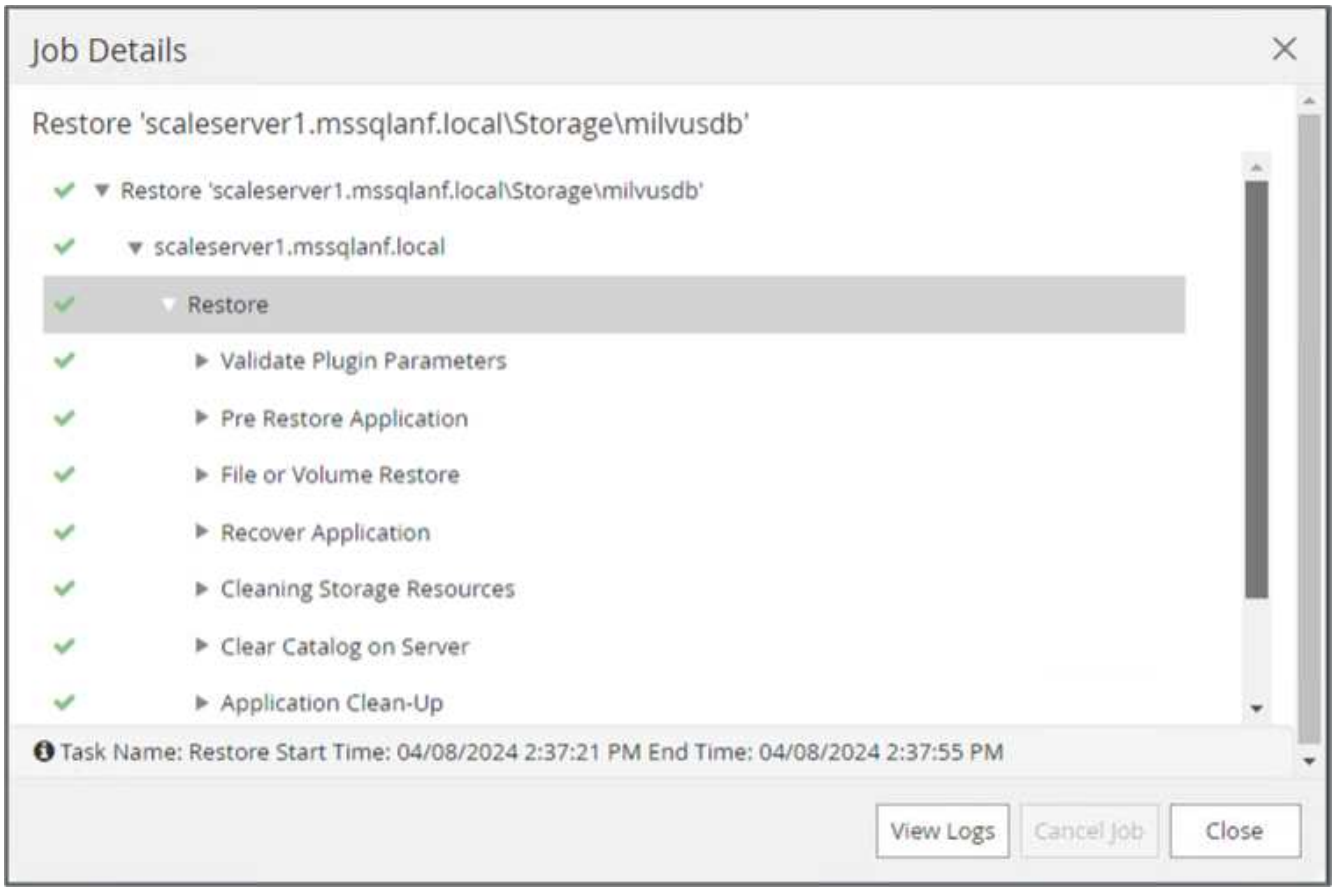



8. Para probar la funcionalidad de copia de seguridad, hemos agregado una nueva tabla después del proceso de copia de seguridad o hemos eliminado algunos datos del NFS (S3 bucket NAS).

   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:~#
```

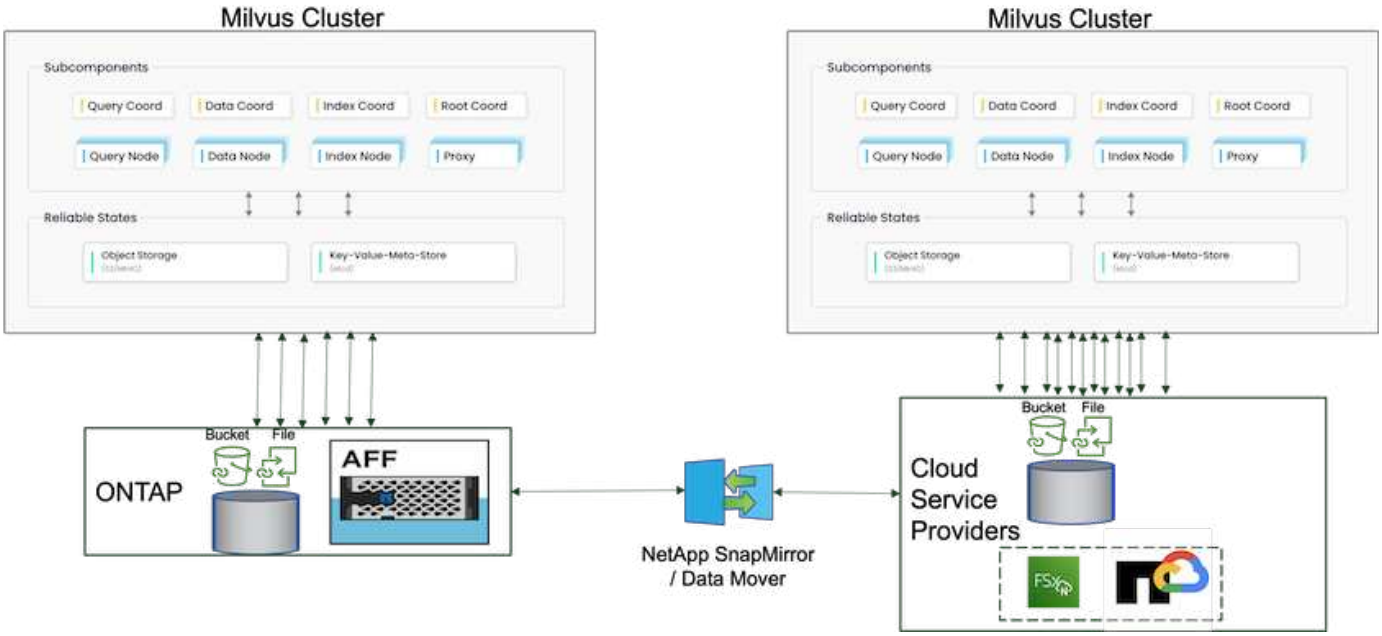
En conclusión, el uso de SnapCenter de NetApp para proteger los datos de bases de datos vectoriales y los datos de Milvus que residen en ONTAP ofrece importantes ventajas a los clientes, en particular en sectores donde la integridad de datos es primordial, como la producción cinematográfica. La capacidad 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.

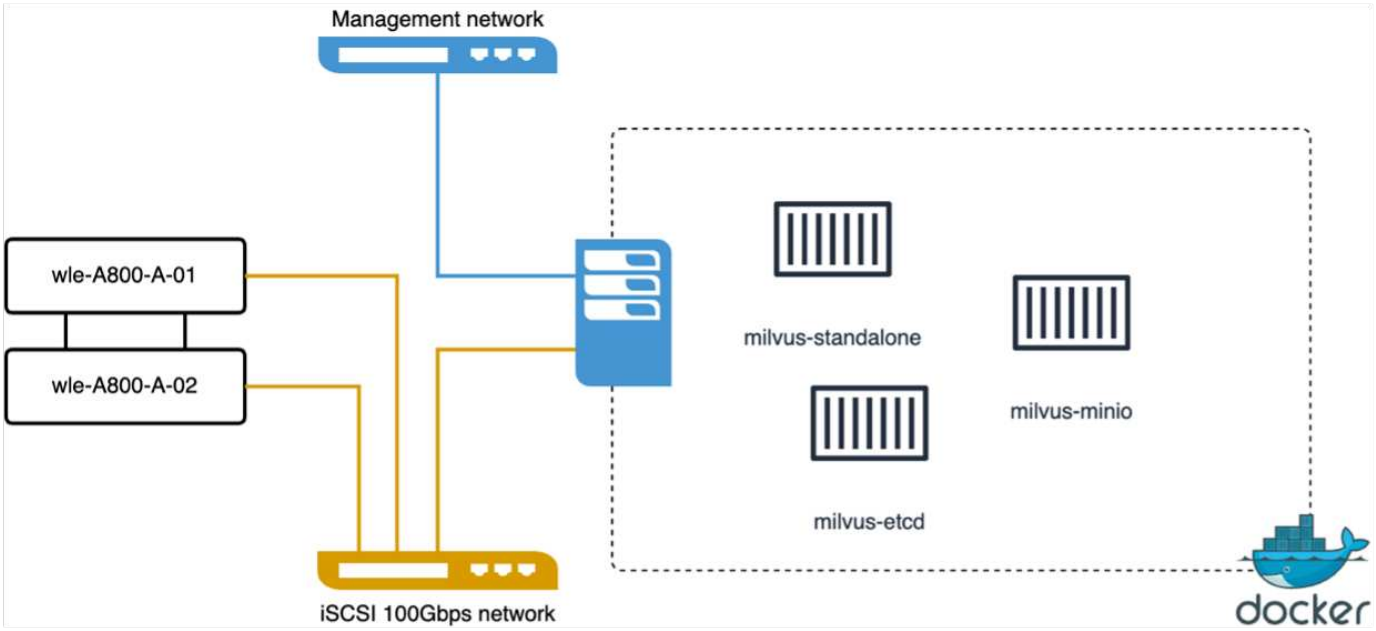
En este apartado, el objetivo es profundizar en la validación del rendimiento de las bases de datos vectoriales, como Milvus y pgvecto.rs, centrándose en sus características de rendimiento del almacenamiento, como el perfil de I/O y el comportamiento de la controladora de almacenamiento NetApp, para 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<br>* 10Million incrustaciones<br>* 768 Dimensiones<br>* ~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.

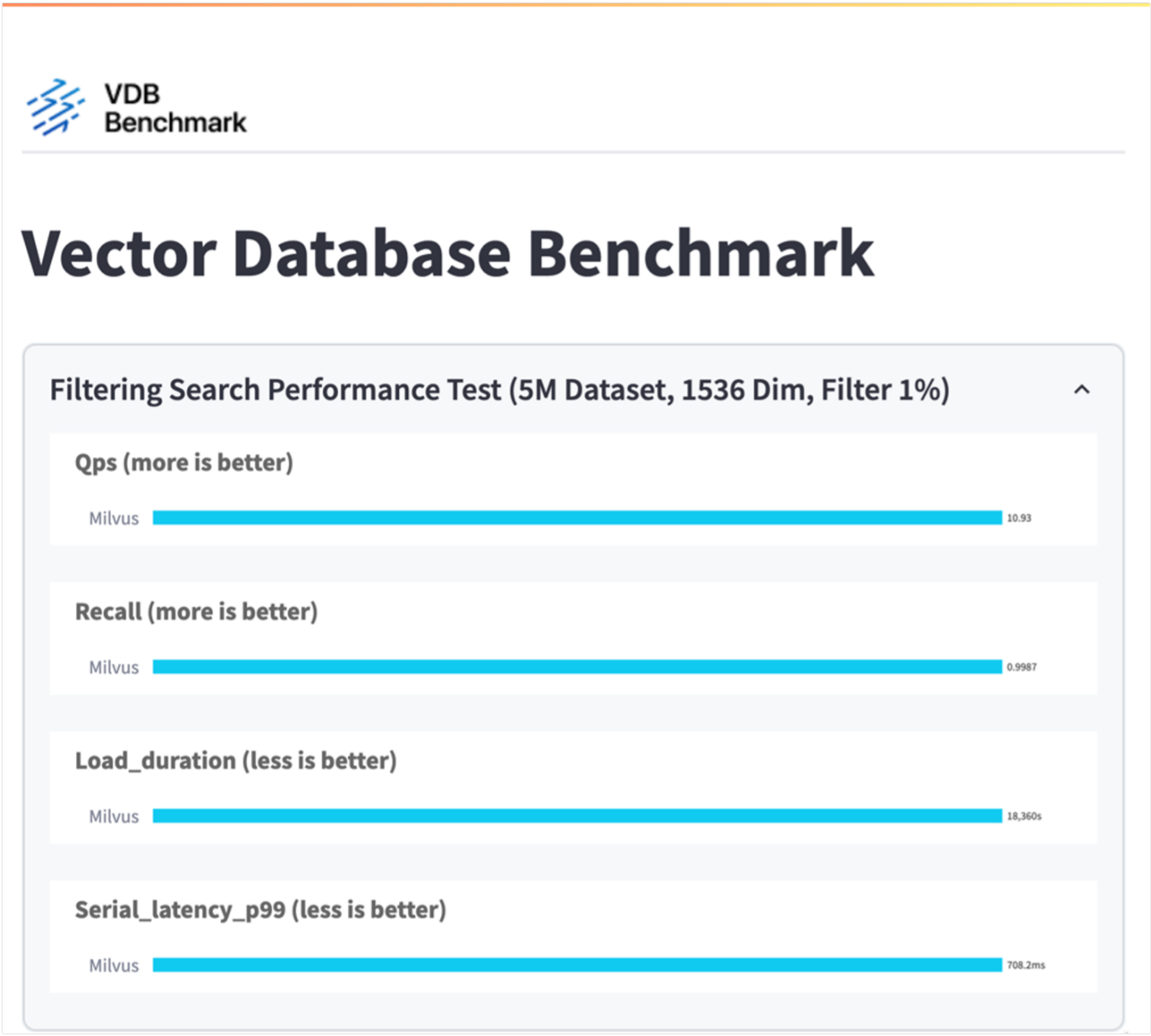
Desde el punto de vista del almacenamiento, la base de datos emitió unas 1.000 operaciones por segundo durante las fases de procesamiento, optimización posterior a la inserción y creación de índices. En la fase de consulta, demandó 32.000 ops/s.

El siguiente apartado presenta las métricas de rendimiento del almacenamiento.

| Fase de carga de trabajo | Métrico | Valor |
|---|---|---|
| Ingesta de datos<br>y..<br>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.



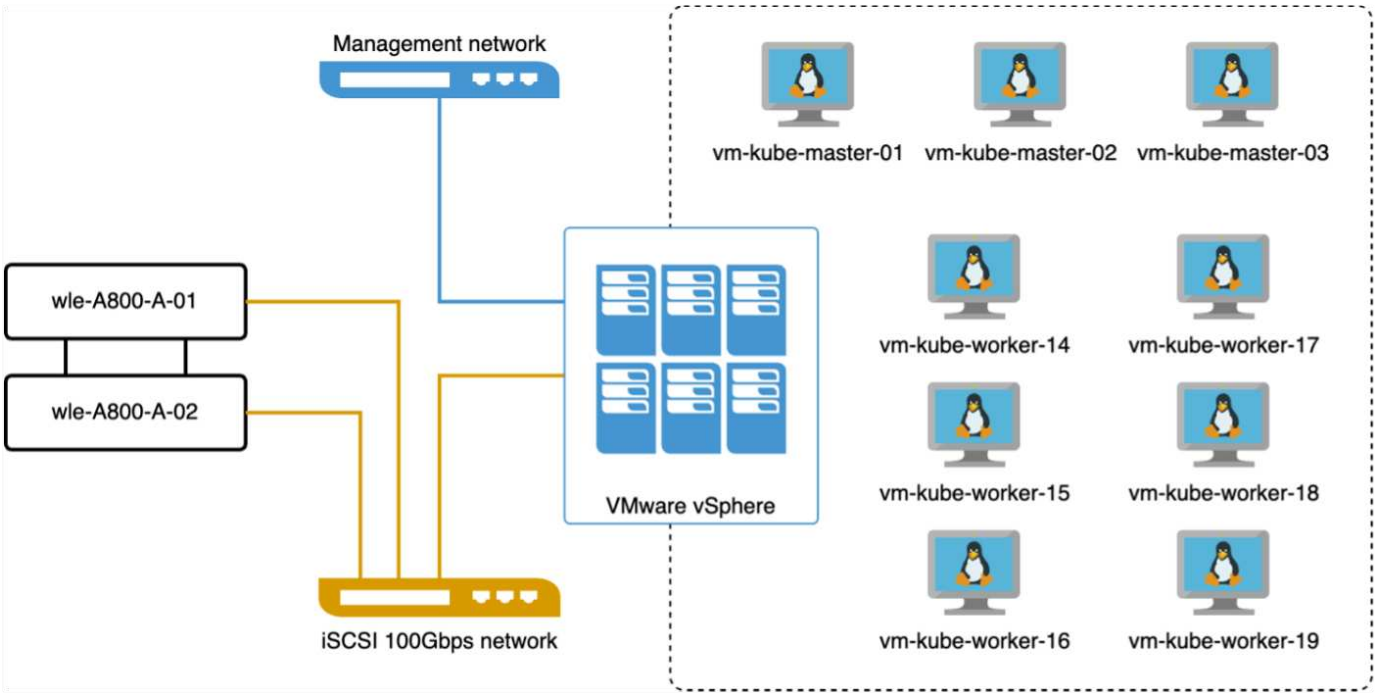

A partir de la validación del rendimiento de la instancia independiente de Milvus, es evidente que la configuración actual es insuficiente para admitir un conjunto de datos de 5 millones de vectores con una dimensionalidad de 1536. hemos determinado que el almacenamiento posee los recursos adecuados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

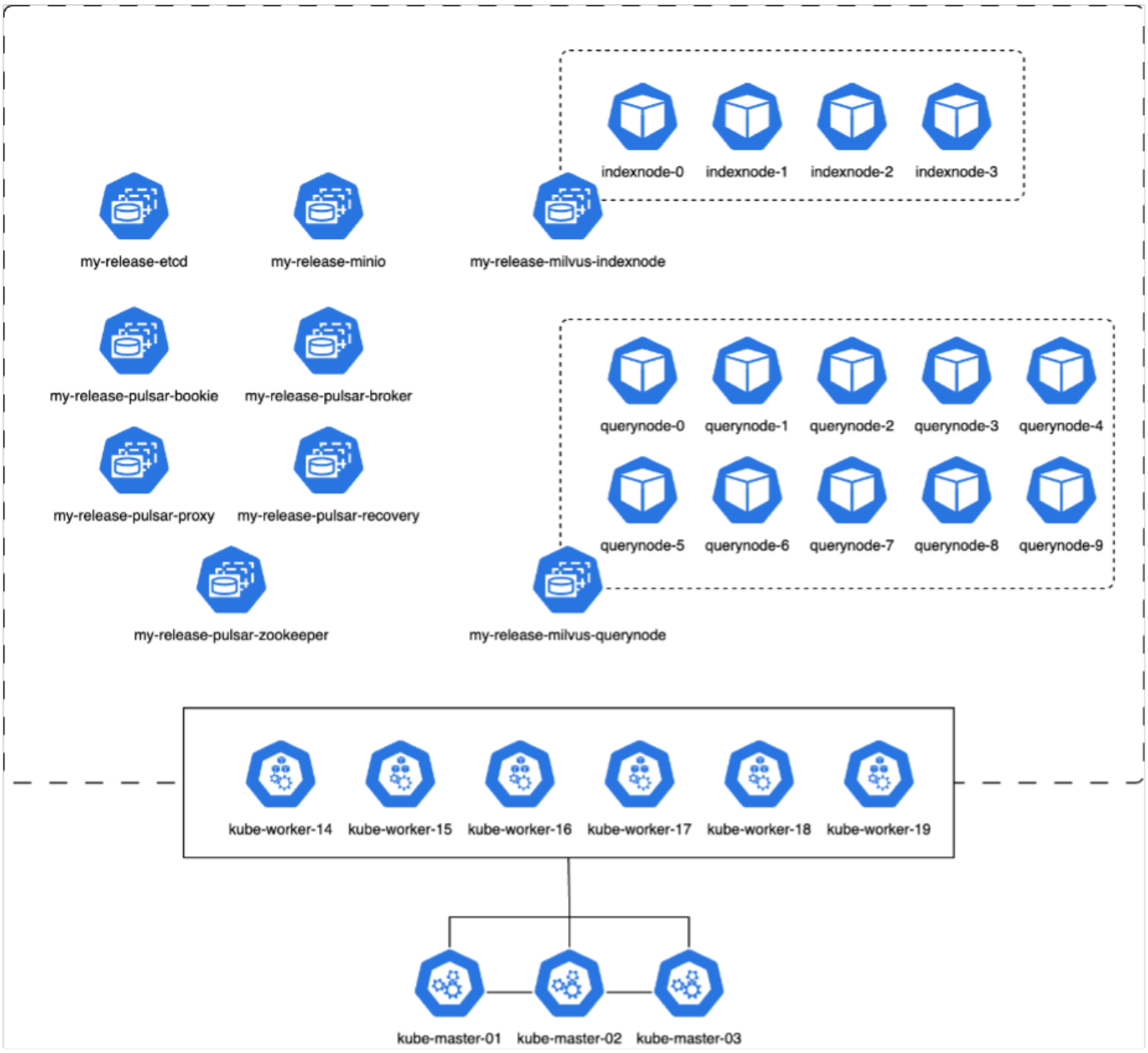
#### VectorDB-Banco con clúster milvus

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).

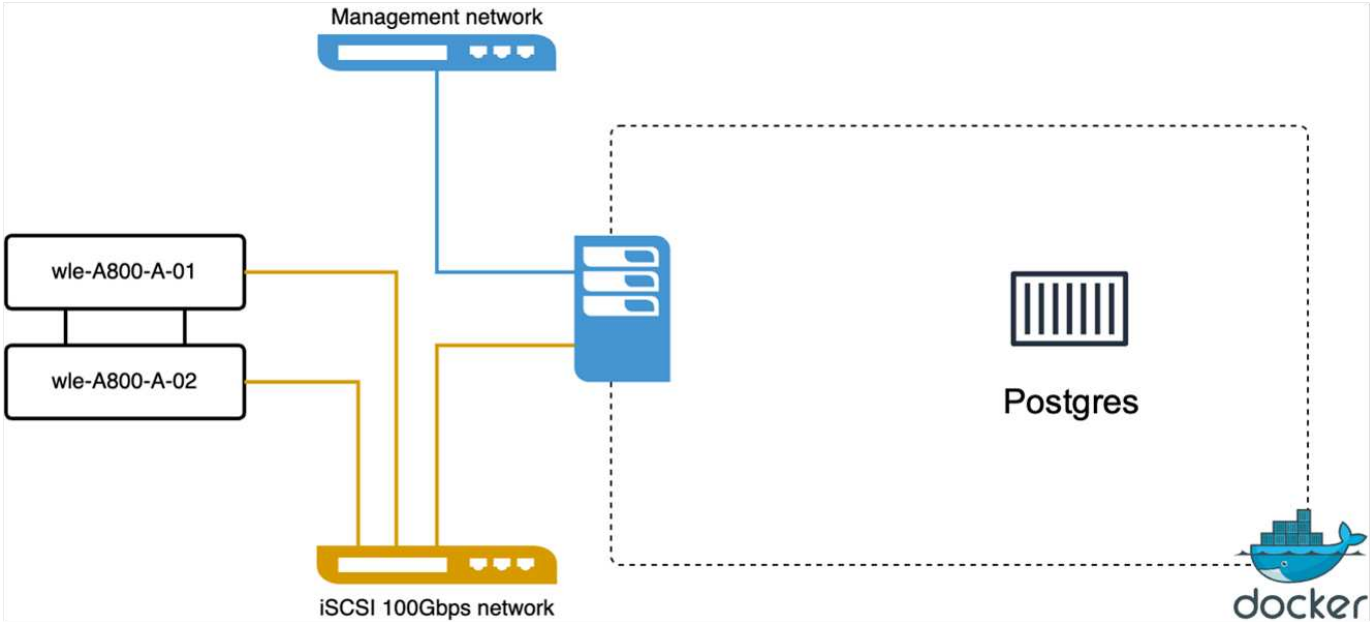

La siguiente sección presenta las métricas clave de rendimiento del almacenamiento.

| Fase de carga de trabajo | Métrico | Valor |
|---|---|---|
| Ingesta de datos<br>y..<br>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:

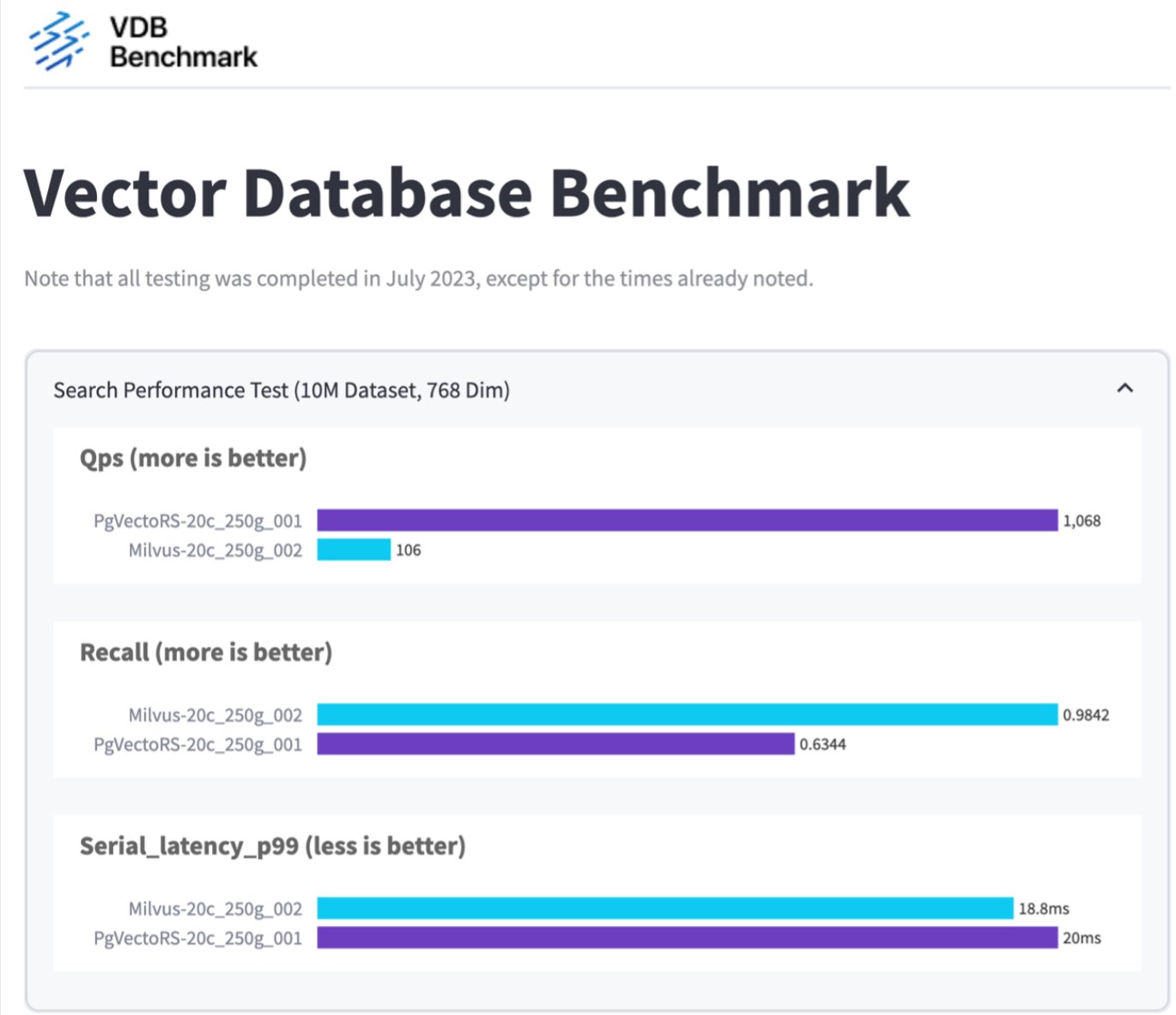



En esta sección, compartimos nuestras observaciones y resultados de la prueba de la base de datos PostgreSQL, específicamente usando pgvecto.rs.
* Seleccionamos HNSW como el tipo de índice para estas pruebas porque en el momento de las pruebas, DiskANN no estaba disponible para pgvecto.rs.
* Durante 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.

## Información de copyright



## Información de la marca comercial