Solution Verification Overview

NetApp Solutions

NetApp
April 26, 2024

Table of Contents

Solution Overview

We have conducted a comprehensive solution validation focused on five key areas, the details of which are outlined below. Each section delves into the challenges faced by customers, the solutions provided by NetApp, and the subsequent benefits to the customer.

1. Milvus cluster setup with Kubernetes in on-premises
   Customer challenges to scale independently on storage and compute, effective infrastructure management and data management. In this section, we detail the process of installing a Milvus cluster on Kubernetes, utilizing a NetApp storage controller for both cluster data and customer data.
2. Milvus with Amazon FSxN for NetApp ONTAP – file and object duality
   In this section, Why we need to deploy vector database in cloud as well as steps to deploy vector database (milvus standalone) in Amazon FSxN for NetApp ONTAP within docker containers.
3. Vector database protection using NetApp SnapCenter.
   In this section, we delve into how SnapCenter safeguards the vector database data and Milvus data residing in ONTAP. For this example, we utilized a NAS bucket (milvusdbvol1) derived from an NFS ONTAP volume (vol1) for customer data, and a separate NFS volume (vectordbpv) for Milvus cluster configuration data.
4. Disaster Recovery using NetApp SnapMirror
   In this section, we discuss about the importance of Disaster recovery(DR) for vector database and how netapp disaster recovery product snapmirror provides DR solution to vector database.
5. Performance validation
   In this section, we aim to delve into the performance validation of vector databases, such as Milvus and pgvecto.rs, focusing on their storage performance characteristics such as I/O profile and netapp storage controller behavious in support of RAG and inference workloads within the LLM Lifecycle. We will evaluate and identify any performance differentiators when these databases are combined with the ONTAP storage solution. Our analysis will be based on key performance indicators, such as the number of queries processed per second(QPS).

Milvus Cluster Setup with Kubernetes in on-premises

Milvus cluster setup with Kubernetes in on-premises

Customer challenges to scale independently on storage and compute, effective infrastructure management and data management,
Kubernetes and vector databases together form a powerful, scalable solution for managing large data operations. Kubernetes optimizes resources and manages containers, while vector databases efficiently handle high-dimensional data and similarity searches. This combination enables swift processing of complex queries on large datasets and seamlessly scales with growing data volumes, making it ideal for big data applications and AI workloads.

1. In this section, we detail the process of installing a Milvus cluster on Kubernetes, utilizing a NetApp storage controller for both cluster data and customer data.
2. To install a Milvus cluster, Persistent Volumes (PVs) are required for storing data from various Milvus cluster components. These components include etcd (three instances), pulsar-bookie-journal (three instances), pulsar-bookie-ledgers (three instances), and pulsar-zookeeper-data (three instances).
3. We created a single NFS volume from NetApp ONTAP and established 12 persistent volumes, each with 250GB of storage. The storage size can vary depending on the cluster size; for instance, we have another cluster where each PV has 50GB. Please refer below to one of the PV YAML files for more details; we had 12 such files in total. In each file, the storageClassName is set to 'default', and the storage and path are

unique to each PV.

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

4. Execute the 'kubectl apply' command for each PV YAML file to create the Persistent Volumes, and then verify their creation using ‘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. For storing customer data, Milvus supports object storage solutions such as MinIO, Azure Blob, and S3. In this guide, we utilize S3. The following steps apply to both ONTAP S3 and StorageGRID object store. We use Helm to deploy the Milvus cluster. Download the configuration file, values.yaml, from the Milvus download location. Please refer to the appendix for the values.yaml file we used in this document.
6. Ensure that the 'storageClass' is set to 'default' in each section, including those for the log, etcd, zookeeper, and bookkeeper.
7. In the MinIO section, disable MinIO.
8. Create a NAS bucket from ONTAP or StorageGRID object storage and include them in an External S3 with the object storage credentials.

```
###################################
# 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. Before creating the Milvus cluster, ensure that the PersistentVolumeClaim (PVC) does not have any pre-existing resources.

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

10. Utilize Helm and the values.yaml configuration file to install and start the Milvus cluster.

```
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. Verify the status of the PersistentVolumeClaims (PVCs).

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

12. Check the status of the 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>
```

Please make sure the pods status are ‘running’ and working as expected

13. Test data writing and reading in Milvus and NetApp object storage.
    - Write data using the "prepare_data_netapp_new.py" Python program.

```
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:~#
```

- Read the data using the "verify_data_netapp.py" Python file.

```
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}}]}
```

Based on the above validation, the integration of Kubernetes with a vector database, as demonstrated through the deployment of a Milvus cluster on Kubernetes using a NetApp storage controller, offers customers a robust, scalable, and efficient solution for managing large-scale data operations. This setup provides customers with the ability to handle high-dimensional data and execute complex queries rapidly and efficiently, making it an ideal solution for big data applications and AI workloads. The use of Persistent Volumes (PVs) for various cluster components, along with the creation of a single NFS volume from NetApp ONTAP, ensures optimal resource utilization and data management. The process of verifying the status of PersistentVolumeClaims (PVCs) and pods, as well as testing data writing and

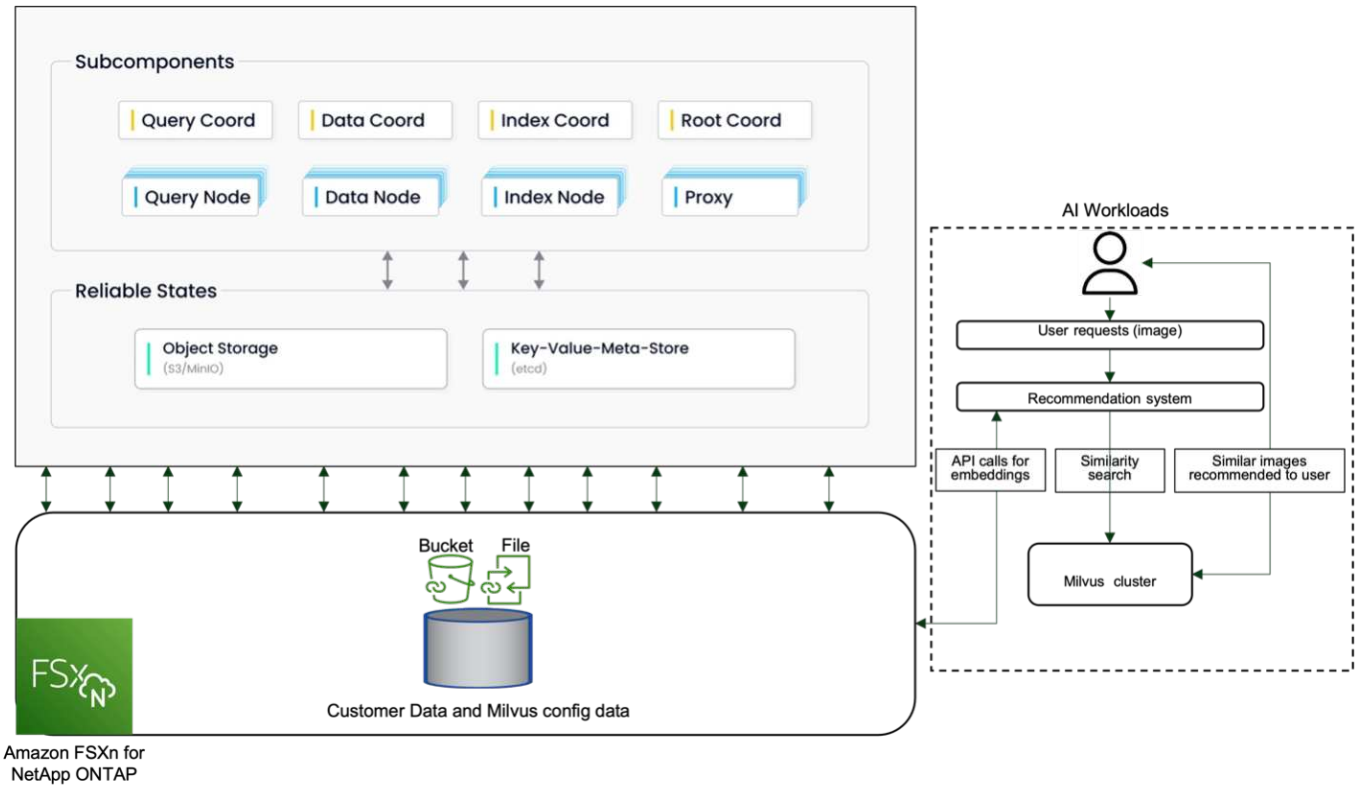
reading, provides customers with the assurance of reliable and consistent data operations. The use of ONTAP or StorageGRID object storage for customer data further enhances data accessibility and security. Overall, this setup empowers customers with a resilient and high-performing data management solution that can seamlessly scale with their growing data needs.

Milvus with Amazon FSxN for NetApp ONTAP - file and object duality

Milvus with Amazon FSxN for NetApp ONTAP – file and object duality

In this section, Why we need to deploy vector database in cloud as well as steps to deploy vector database (milvus standalone) in Amazon FSxN for NetApp ONTAP within docker containers.

Deploying a vector database in the cloud provides several significant benefits, particularly for applications that require handling high-dimensional data and executing similarity searches. First, cloud-based deployment offers scalability, allowing for the easy adjustment of resources to match the growing data volumes and query loads. This ensures that the database can efficiently handle increased demand while maintaining high performance. Second, cloud deployment provides high availability and disaster recovery, as data can be replicated across different geographical locations, minimizing the risk of data loss, and ensuring continuous service even during unexpected events. Third, it provides cost-effectiveness, as you only pay for the resources you use, and can scale up or down based on demand, avoiding the need for substantial upfront investment in hardware. Finally, deploying a vector database in the cloud can enhance collaboration, as data can be accessed and shared from anywhere, facilitating team-based work and data-driven decision making.
Please check the architecture of the milvus standalone with Amazon FSxN for NetApp ONTAP used in this validation.



1. Create an Amazon FSxN for NetApp ONTAP instance and note down the details of the VPC, VPC security groups, and subnet. This information will be required when creating an EC2 instance. You can find more details here - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create

2. Create an EC2 instance, ensuring that the VPC, Security Groups, and subnet match those of the Amazon FSxN for NetApp ONTAP instance.
3. Install nfs-common using the command 'apt-get install nfs-common' and update the package information using 'sudo apt-get update'.
4. Create a mount folder and mount the Amazon FSxN for NetApp ONTAP on it.

```
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. Install Docker and Docker Compose using 'apt-get install'.
6. Set up a Milvus cluster based on the docker-compose.yaml file, which can be downloaded from the Milvus website.

```
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. In the 'volumes' section of the docker-compose.yml file, map the NetApp NFS mount point to the corresponding Milvus container path, specifically in etcd, minio, and standalone.Check Appendix D: docker-compose.yml for details about changes in yml
8. Verify the mounted folders and files.

```
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. Run 'docker-compose up -d' from the directory containing the docker-compose.yml file.
10. Check the status of the Milvus container.

```
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. To validate the read and write functionality of vector database and it's data in Amazon FSxN for NetApp ONTAP, we used the Python Milvus SDK and a sample program from PyMilvus. Install the necessary packages using 'apt-get install python3-numpy python3-pip' and install PyMilvus using 'pip3 install pymilvus'.
12. Validate data writing and reading operations from Amazon FSxN for NetApp ONTAP in the vector database.

```
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. Check the reading operation using the verify_data_netapp.py script.

```
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. If the customer wants to access (read) NFS data tested in the vector database via the S3 protocol for AI workloads, this can be validated using a straightforward Python program. An example of this could be a similarity search of images from another application as mentioned in the picture that is in the beginning of this section.

```
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#
```

This section effectively demonstrates how customers can deploy and operate a standalone Milvus setup within Docker containers, utilizing Amazon's NetApp FSxN for NetApp ONTAP data storage. This setup allows customers to leverage the power of vector databases for handling high-dimensional data and executing complex queries, all within the scalable and efficient environment of Docker containers. By creating an Amazon FSxN for NetApp ONTAP instance and matching EC2 instance, customers can ensure optimal resource utilization and data management. The successful validation of data writing and reading operations from FSxN in the vector database provides customers with the assurance of reliable and consistent data operations. Additionally, the ability to list (read) data from AI workloads via the S3 protocol offers enhanced data accessibility. This comprehensive process, therefore, provides customers with a robust and efficient solution for managing their large-scale data operations, leveraging the capabilities of Amazon's FSxN for NetApp ONTAP.

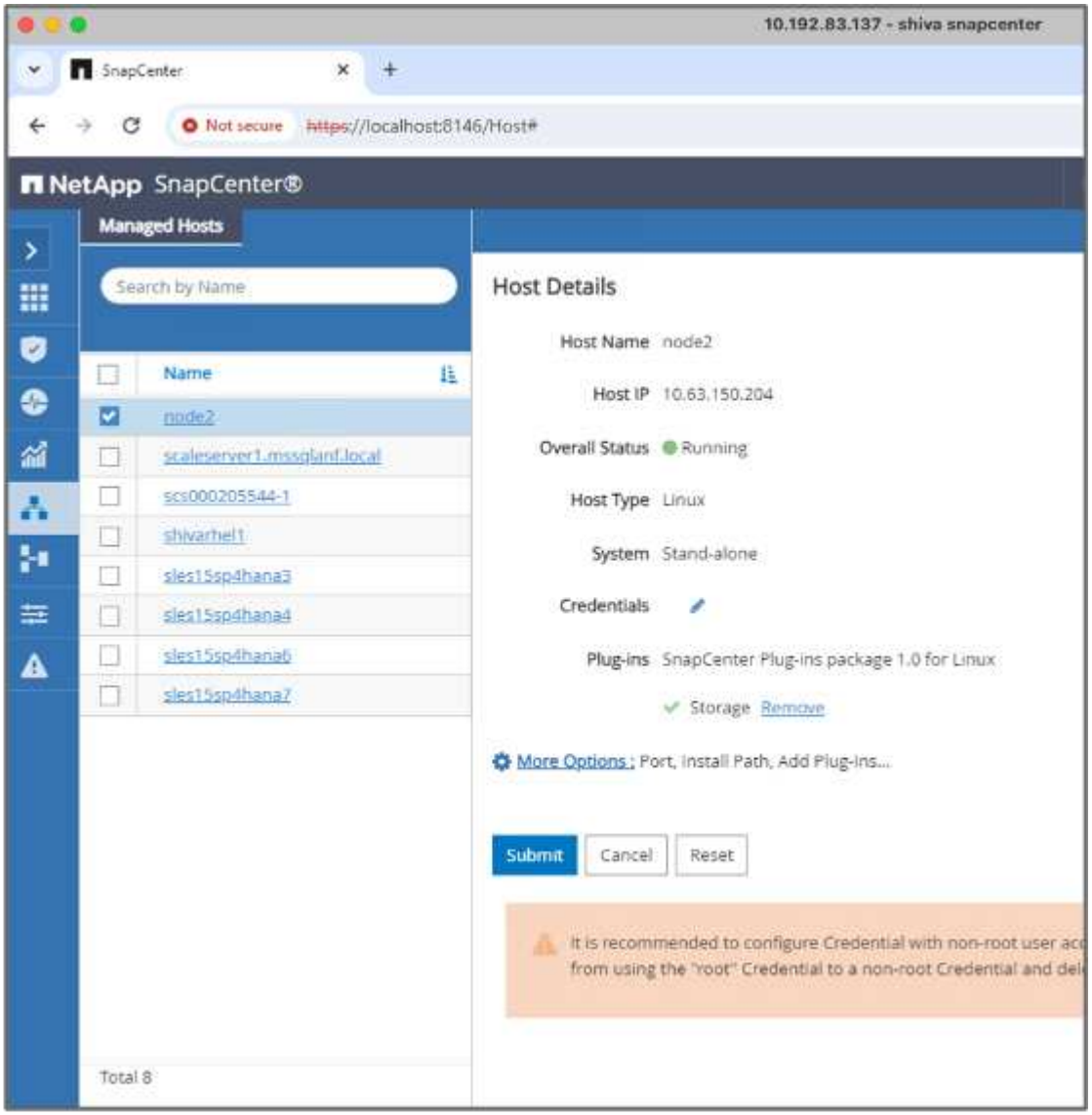
Vector Database Protection using SnapCenter

Vector database protection using NetApp SnapCenter.

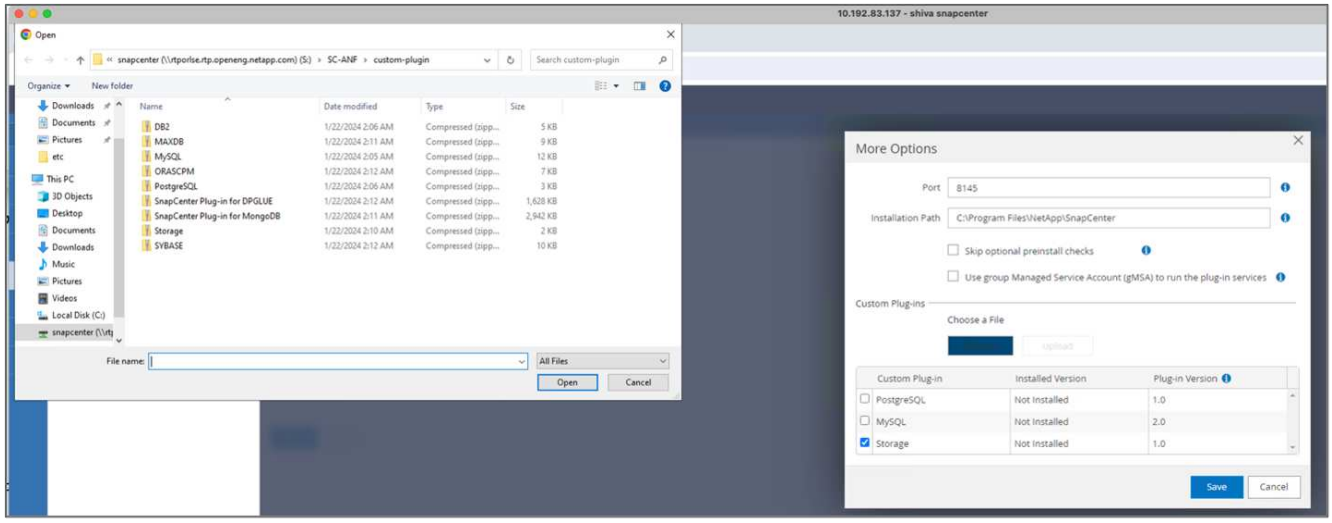
For example, in the film production industry, customers often possess critical embedded data such as video

and audio files. Loss of this data, due to issues like hard drive failures, can have a significant impact on their operations, potentially jeopardizing multimillion-dollar ventures. We have encountered instances where invaluable content was lost, causing substantial disruption and financial loss. Ensuring the security and integrity of this essential data is therefore of paramount importance in this industry.
In this section, we delve into how SnapCenter safeguards the vector database data and Milvus data residing in ONTAP. For this example, we utilized a NAS bucket (milvusdbvol1) derived from an NFS ONTAP volume (vol1) for customer data, and a separate NFS volume (vectordbpv) for Milvus cluster configuration data. please check the here for the snapcenter backup workflow

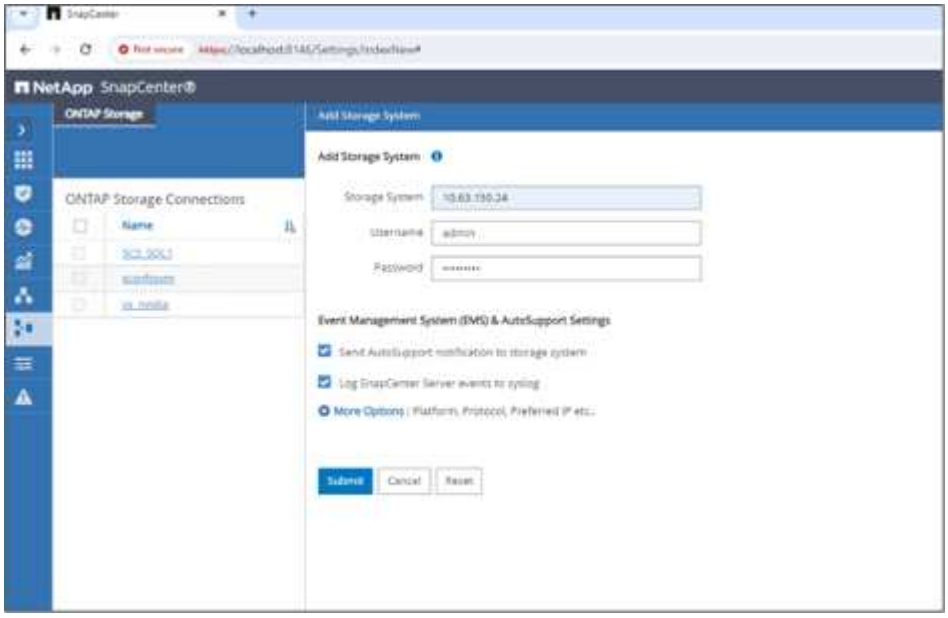
1. Set up the host that will be used to execute SnapCenter commands.



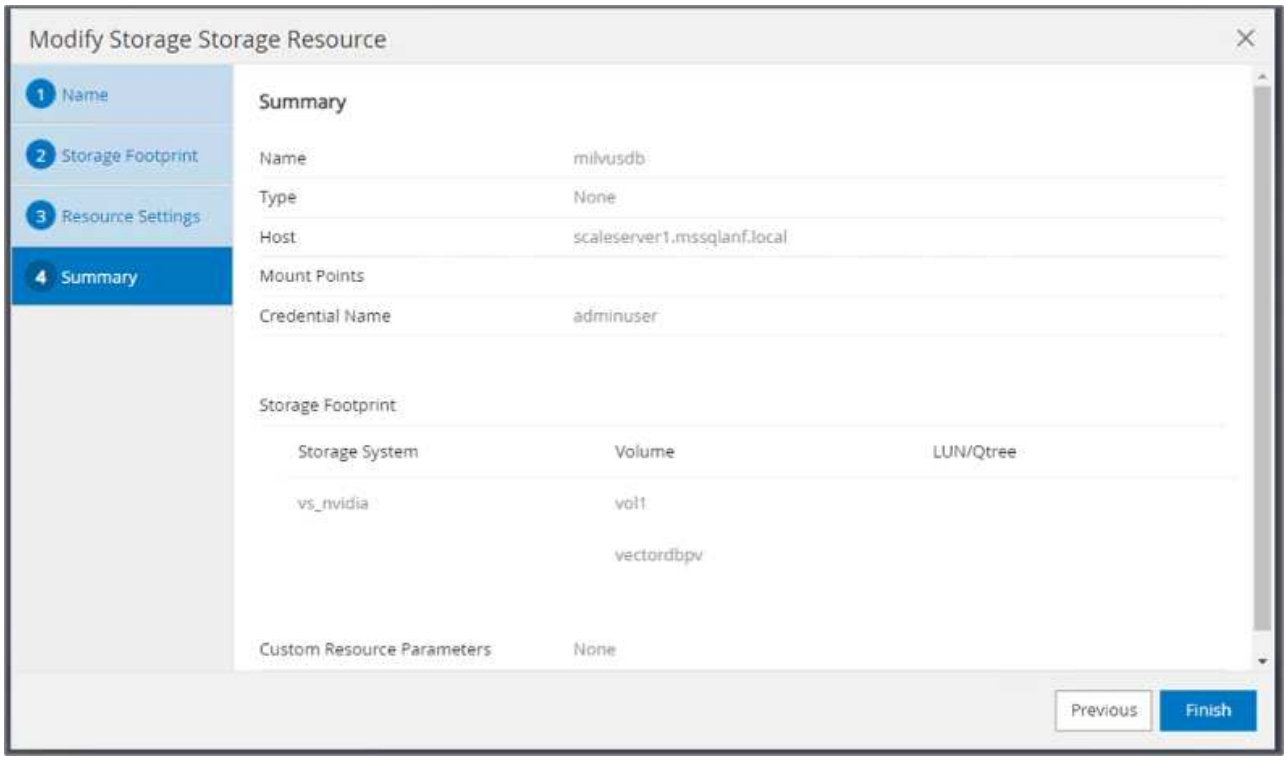
2. Install and configure the storage plugin. From the added host, select "More Options". Navigate to and select the downloaded storage plugin from the NetApp Automation Store. Install the plugin and save the configuration.



3. Set up the storage system and volume: Add the storage system under "Storage System" and select the SVM (Storage Virtual Machine). In this example, we've chosen "vs_nvidia".



4. Establish a resource for the vector database, incorporating a backup policy and a custom snapshot name.
   - Enable Consistency Group Backup with default values and enable SnapCenter without filesystem consistency.
   - In the Storage Footprint section, select the volumes associated with the vector database customer data and Milvus cluster data. In our example, these are "vol1" and "vectordbpv".
   - Create policy for vector database protection and protect vector database resource using the policy.



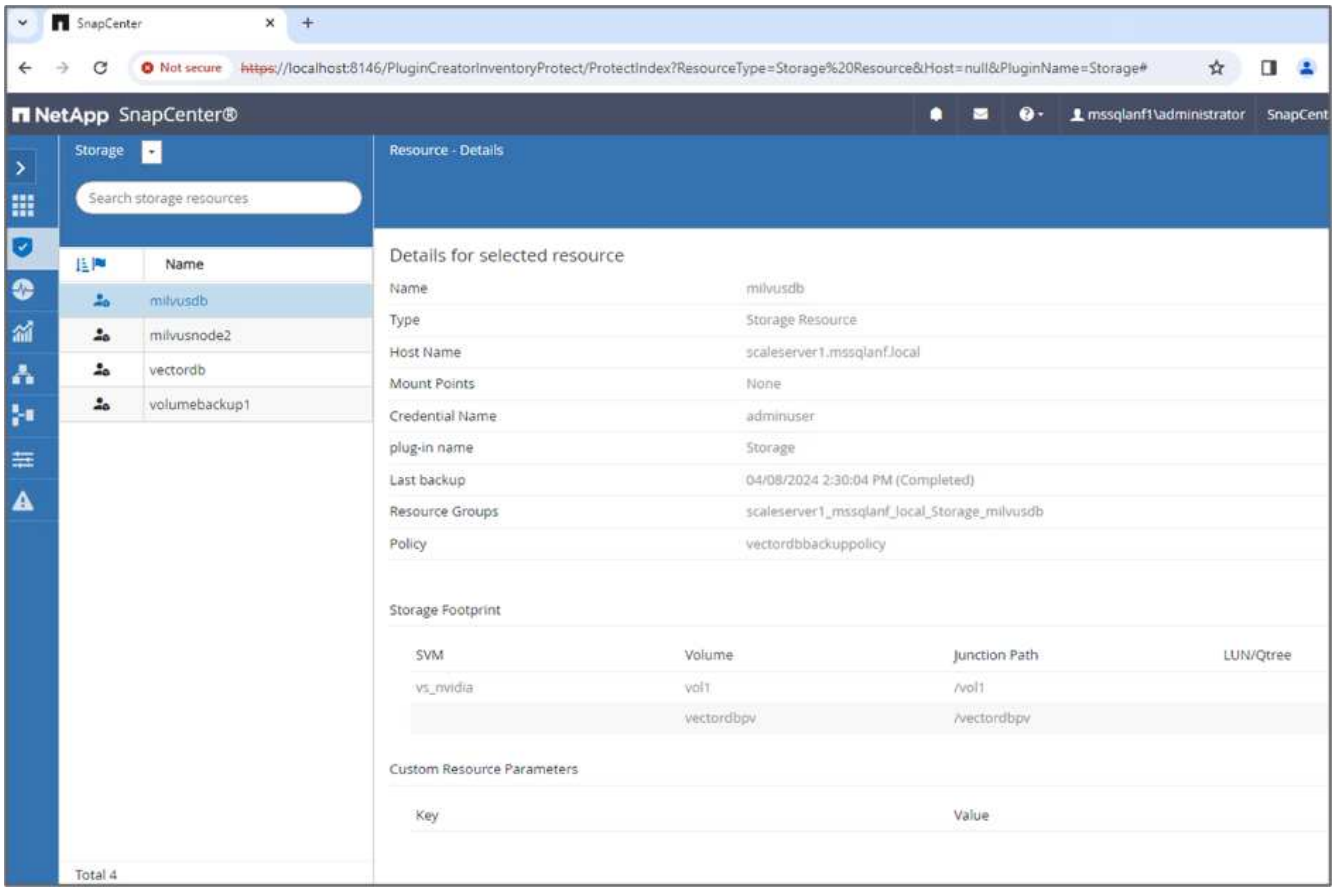
5. Insert data into the S3 NAS bucket using a Python script. In our case, we modified the backup script provided by Milvus, namely 'prepare_data_netapp.py', and executed the 'sync' command to flush the data from the operating system.

```
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. Verify the data in the S3 NAS bucket. In our example, the files with the timestamp '2024-04-08 21:22' were created by the 'prepare_data_netapp.py' script.

```
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. Initiate a backup using the Consistency Group (CG) snapshot from the 'milvusdb' resource

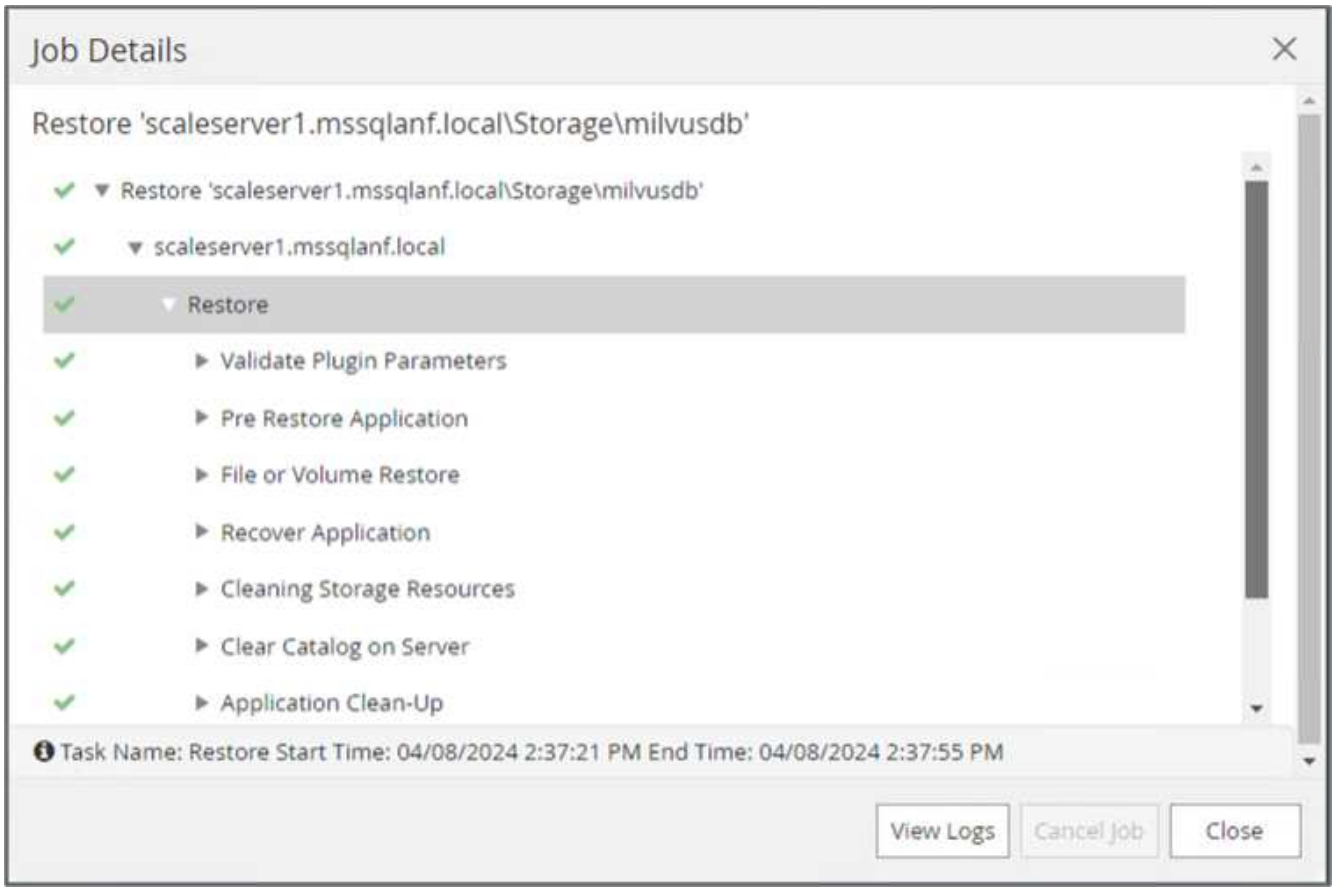


8. To test the backup functionality, we either added a new table after the backup process or removed some data from the NFS (S3 NAS bucket).

   For this test, imagine a scenario where someone created a new, unnecessary, or inappropriate collection after the backup. In such a case, we would need to revert the vector database to its state before the new collection was added. For instance, new collections such as 'hello_milvus_netapp_sc_testnew' and 'hello_milvus_netapp_sc_testnew2' have been inserted.

```
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. Execute a full restore of the S3 NAS bucket from the previous snapshot.



10. Use a Python script to verify the data from the 'hello_milvus_netapp_sc_test' and 'hello_milvus_netapp_sc_test2' collections.

```
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. Verify that the unnecessary or inappropriate collection is no longer present in the database.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

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

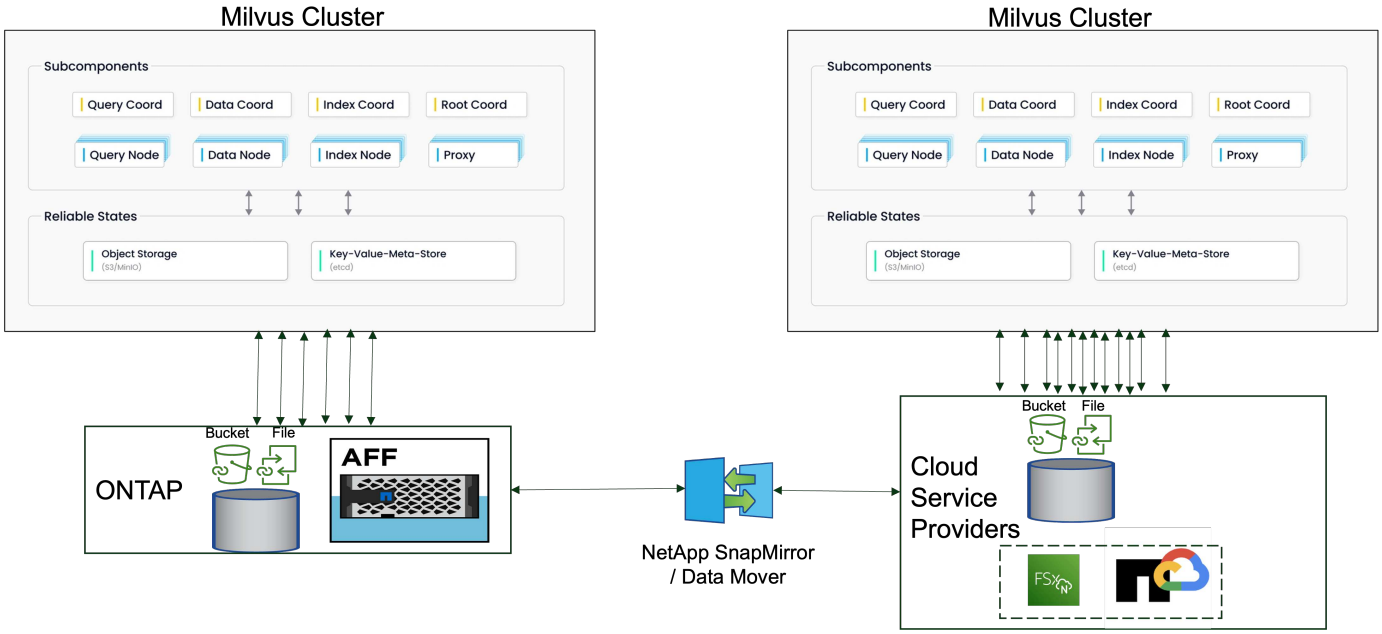
In conclusion, the use of NetApp's SnapCenter to safeguard vector database data and Milvus data residing in ONTAP offers significant benefits to customers, particularly in industries where data integrity is paramount, such as film production. SnapCenter's ability to create consistent backups and perform full data restores ensures that critical data, such as embedded video and audio files, are protected against loss due to hard drive failures or other issues. This not only prevents operational disruption but also safeguards against substantial financial loss.

In this section, we demonstrated how SnapCenter can be configured to protect data residing in ONTAP, including the setup of hosts, installation and configuration of storage plugins, and the creation of a resource for the vector database with a custom snapshot name. We also showcased how to perform a backup using the Consistency Group snapshot and verify the data in the S3 NAS bucket.

Furthermore, we simulated a scenario where an unnecessary or inappropriate collection was created after the backup. In such cases, SnapCenter's ability to perform a full restore from a previous snapshot ensures that the vector database can be reverted to its state before the addition of the new collection, thus maintaining the integrity of the database. This capability to restore data to a specific point in time is invaluable for customers, providing them with the assurance that their data is not only secure but also correctly maintained. Thus, NetApp's SnapCenter product offers customers a robust and reliable solution for data protection and management.

Disaster Recovery using NetApp SnapMirror

Disaster Recovery using NetApp SnapMirror



Disaster recovery is crucial for maintaining the integrity and availability of a vector database, especially given its role in managing high-dimensional data and executing complex similarity searches. A well-planned and implemented disaster recovery strategy ensures that data is not lost or compromised in the event of unforeseen incidents, such as hardware failures, natural disasters, or cyber-attacks. This is particularly significant for applications relying on vector databases, where the loss or corruption of data could lead to significant operational disruptions and financial losses. Moreover, a robust disaster recovery plan also ensures business continuity by minimizing downtime and allowing for the quick restoration of services. This is achieved through NetApp data replication product SnapMirrror across different geographical locations, regular backups, and failover mechanisms. Therefore, disaster recovery is not just a protective measure, but a critical component of responsible and efficient vector database management.

NetApp's SnapMirror provides data replication from one NetApp ONTAP storage controller to another, primarily used for disaster recovery (DR) and hybrid solutions. In the context of a vector database, this tool facilitates the smooth transition of data between on-premises and cloud environments. This transition is achieved without necessitating any data conversions or application refactoring, thereby enhancing the efficiency and flexibility of data management across multiple platforms.

NetApp Hybrid solution in a vector database scenario can bring about more advantages:

1. Scalability: NetApp's hybrid cloud solution offers the ability to scale your resources as per your requirements. You can utilize on-premises resources for regular, predictable workloads and cloud resources such as Amazon FSxN for NetApp ONTAP and Google Cloud NetApp Volume (GCNV) for peak times or unexpected loads.
2. Cost Efficiency: NetApp's hybrid cloud model allows you to optimize your costs by using on-premises resources for regular workloads and only paying for cloud resources when you need them. This pay-as-you-go model can be quite cost-effective with a NetApp instaclustr service offering. For on-prem and major cloud service providers, instaclustr provids support and consultation.
3. Flexibility: NetApp's hybrid cloud gives you the flexibility to choose where to process your data. For example, you might choose to perform complex vector operations on-premises where you have more powerful hardware, and less intensive operations in the cloud.
4. Business Continuity: In the event of a disaster, having your data in a NetApp hybrid cloud can ensure business continuity. You can quickly switch to the cloud if your on-premises resources are affected. We can leverage NetApp SnapMirror to move the data from on-prem to cloud and vice versa.

5. Innovation: NetApp's hybrid cloud solutions can also enable faster innovation by providing access to cutting-edge cloud services and technologies. NetApp innovations in cloud such as Amazon FSxN for NetApp ONTAP, Azure NetApp Files and Google Cloud NetApp Volumes are cloud service providers innovative products and preferred NAS.

Vector Database Performance Validation

Performance validation

Performance validation plays a critical role in both vector databases and storage systems, serving as a key factor in ensuring optimal operation and efficient resource utilization. Vector databases, known for handling high-dimensional data and executing similarity searches, need to maintain high performance levels to process complex queries swiftly and accurately. Performance validation helps identify bottlenecks, fine-tune configurations, and ensure the system can handle expected loads without degradation in service. Similarly, in storage systems, performance validation is essential to ensure data is stored and retrieved efficiently, without latency issues or bottlenecks that could impact overall system performance. It also aids in making informed decisions about necessary upgrades or changes in storage infrastructure. Therefore, performance validation is a crucial aspect of system management, contributing significantly to maintaining high service quality, operational efficiency, and overall system reliability.

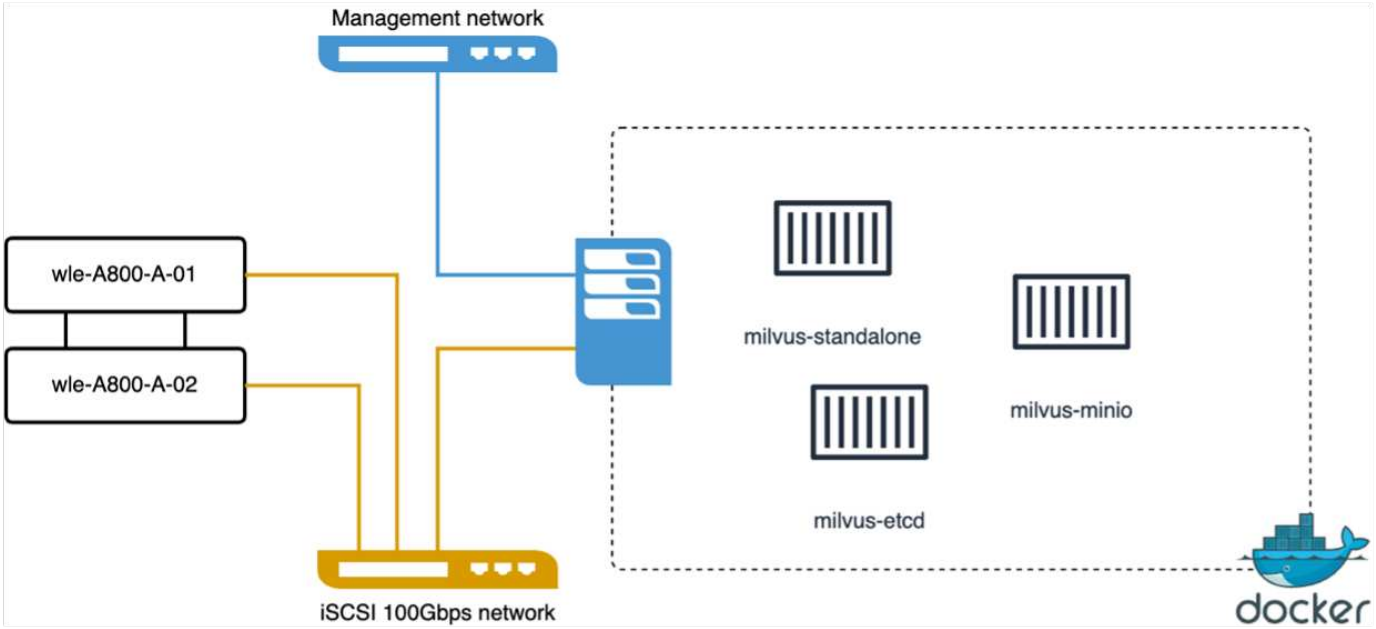
In this section, we aim to delve into the performance validation of vector databases, such as Milvus and pgvecto.rs, focusing on their storage performance characteristics such as I/O profile and netapp storage controller behavious in support of RAG and inference workloads within the LLM Lifecycle. We will evaluate and identify any performance differentiators when these databases are combined with the ONTAP storage solution. Our analysis will be based on key performance indicators, such as the number of queries processed per second(QPS).

Please check the methodology used for milvus and progress below.

Details	Milvus (Standalone and Cluster)	Postgres(pgvecto.rs)
version	2.3.2	0.2.0
Filesystem	XFS on iSCSI LUNs	
Workload Generator	VectorDB-Bench – v0.0.5	
Datasets	LAION Dataset * 10Million Embeddings * 768 Dimensions * ~300GB dataset size	

VectorDB-Bench with Milvus standalone cluster

we did the following performance validation on milvus standalone cluster with vectorDB-Bench.
The network and server connectivity of the milvus standalone cluster is below.



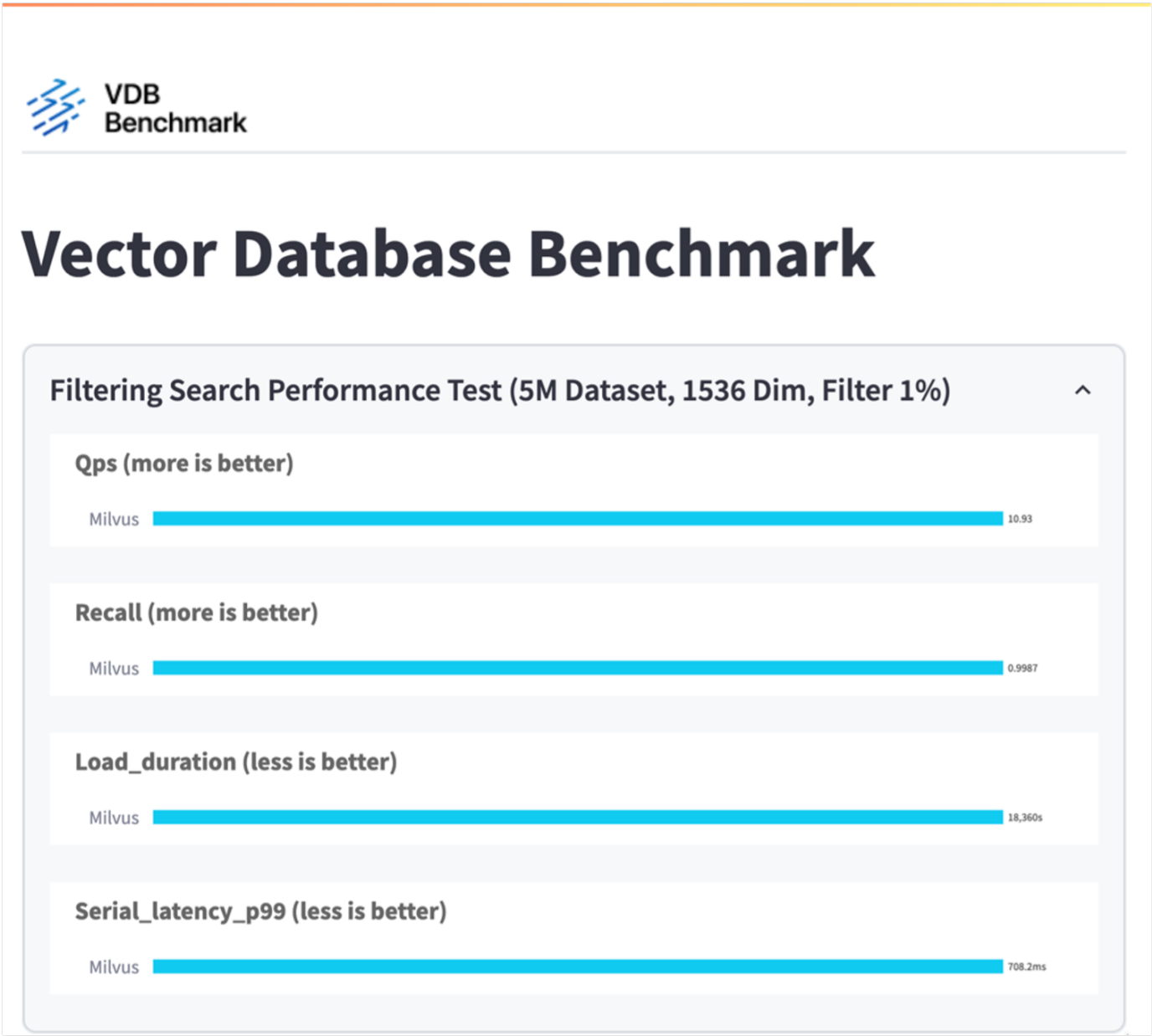
In this section, we share our observations and results from testing the Milvus standalone database.
. We selected DiskANN as the index type for these tests.
. Ingesting, optimizing, and creating indexes for a dataset of approximately 100GB took around 5 hours. For most of this duration, the Milvus server, equipped with 20 cores (which equates to 40 vcpus when Hyper-Threading is enabled), was operating at its maximum CPU capacity of 100%.We found that DiskANN is particularly important for large datasets that exceed the system memory size.
. In the query phase, we observed a Queries per Second (QPS) rate of 10.93 with a recall of 0.9987. The 99th percentile latency for queries was measured at 708.2 milliseconds.

From the storage perspective, the database issued about 1,000 ops/sec during the ingest, post-insert optimization, and index creation phases. In the query phase, it demanded 32,000 ops/sec.

The following section presents the storage performance metrics.

Workload Phase	Metric	Value
Data Ingestion and Post insert optimization	IOPS	< 1,000
	Latency	< 400 usecs
	Workload	Read/Write mix, mostly writes
	IO size	64KB
Query	IOPS	Peak at 32,000
	Latency	< 400 usecs
	Workload	100% cached read
	IO size	Mainly 8KB

The vectorDB-bench result is below.

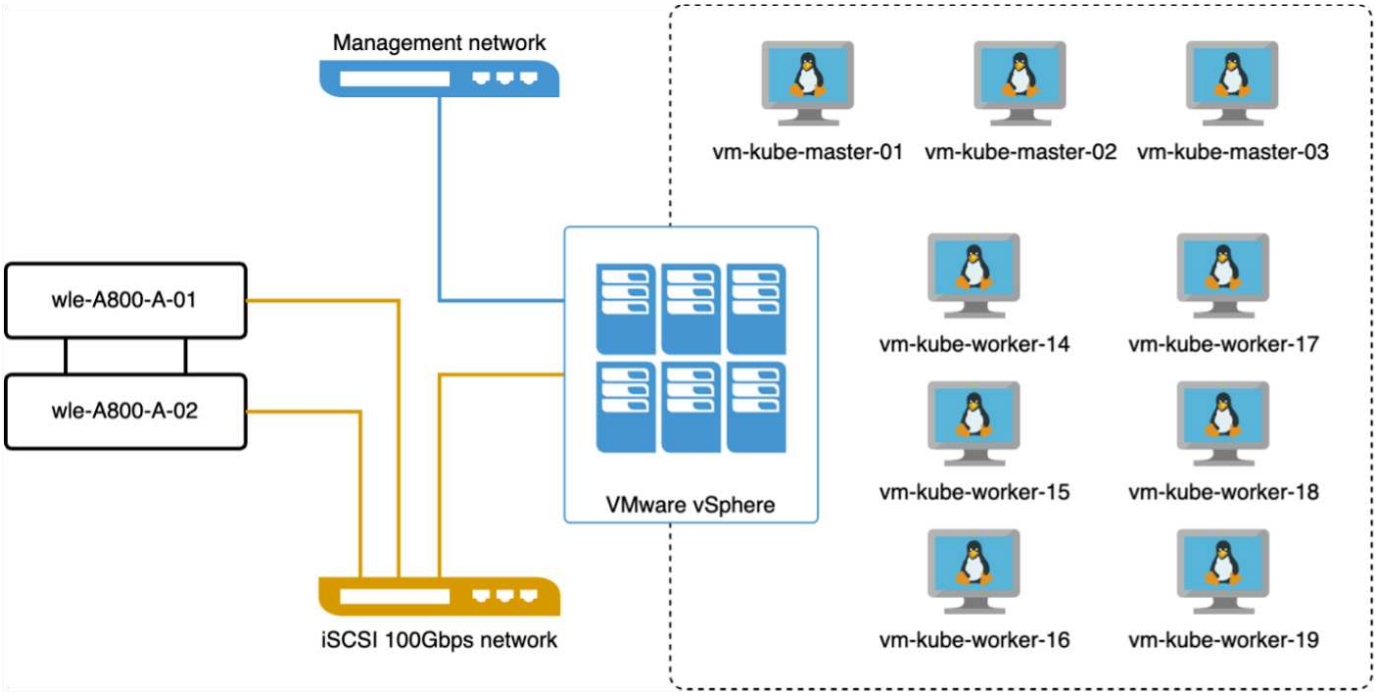


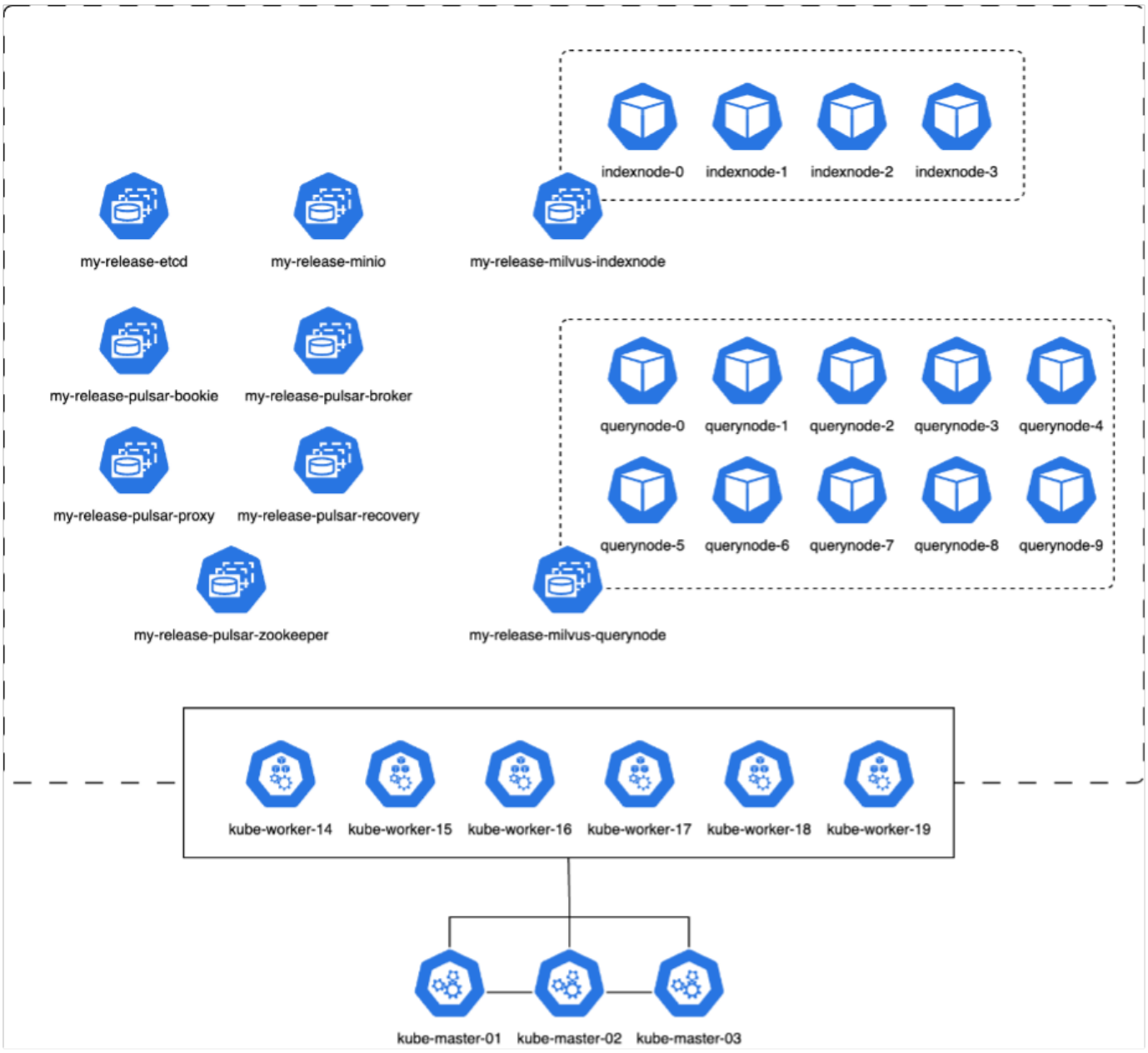
From the performance validation of the standalone Milvus instance, it's evident that the current setup is insufficient to support a dataset of 5 million vectors with a dimensionality of 1536. we've determined that the storage possesses adequate resources and does not constitute a bottleneck in the system.

VectorDB-Bench with milvus cluster

In this section, we discuss the deployment of a Milvus cluster within a Kubernetes environment. This Kubernetes setup was constructed atop a VMware vSphere deployment, which hosted the Kubernetes master and worker nodes.

The details of the VMware vSphere and Kubernetes deployments are presented in the following sections.


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



In this section, we present our observations and results from testing the Milvus database.
* The index type used was DiskANN.
* The table below provides a comparison between the standalone and cluster deployments when working with 5 million vectors at a dimensionality of 1536. We observed that the time taken for data ingestion and post-insert optimization was lower in the cluster deployment. The 99th percentile latency for queries was reduced by six times in the cluster deployment compared to the standalone setup.
* Although the Queries per Second (QPS) rate was higher in the cluster deployment, it was not at the desired level.

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%

The images below provide a view of various storage metrics, including storage cluster latency and total IOPS

(Input/Output Operations Per Second).

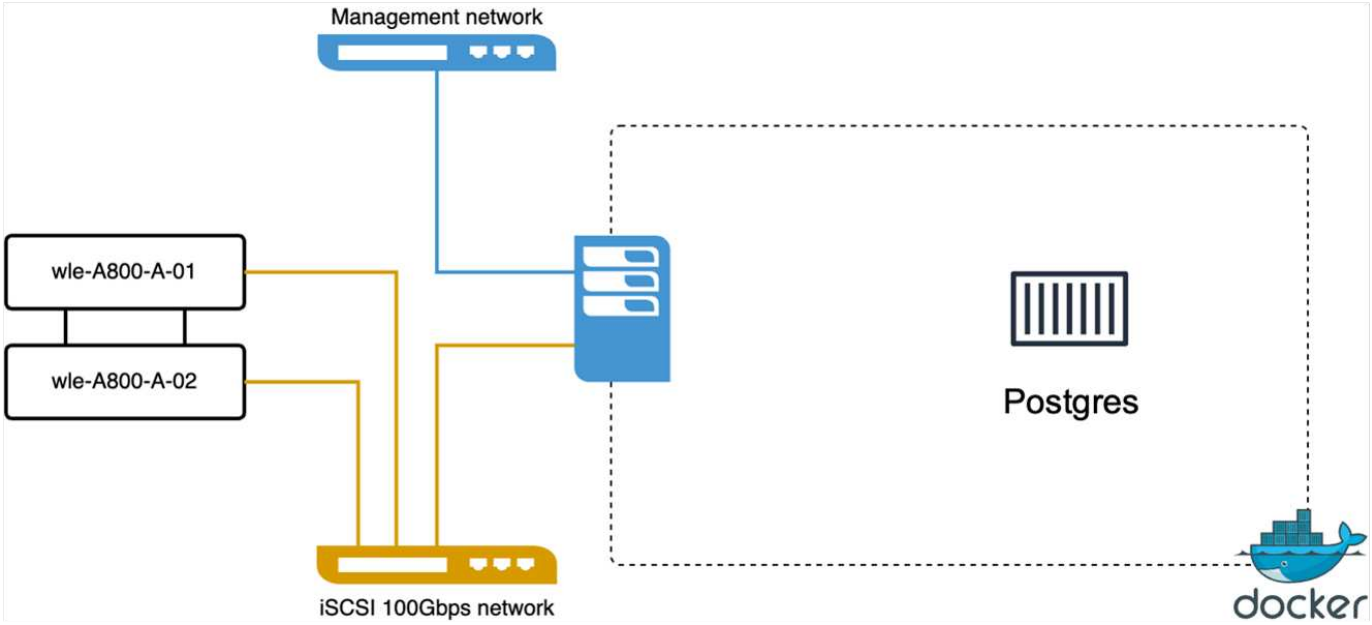
The following section presents the key storage performance metrics.

Workload Phase	Metric	Value
Data Ingestion and Post insert optimization	IOPS	< 1,000
	Latency	< 400 usecs
	Workload	Read/Write mix, mostly writes
	IO size	64KB
Query	IOPS	Peak at 147,000
	Latency	< 400 usecs
	Workload	100% cached read
	IO size	Mainly 8KB

Based on the performance validation of both the standalone Milvus and the Milvus cluster, we present the details of the storage I/O profile.
* We observed that the I/O profile remains consistent across both standalone and cluster deployments.
* The observed difference in peak IOPS can be attributed to the larger number of clients in the cluster deployment.

vectorDB-Bench with Postgres (pgvecto.rs)

We conducted the following actions on PostgreSQL(pgvecto.rs) using VectorDB-Bench:
The details regarding the network and server connectivity of PostgreSQL (specifically, pgvecto.rs) are as follows:



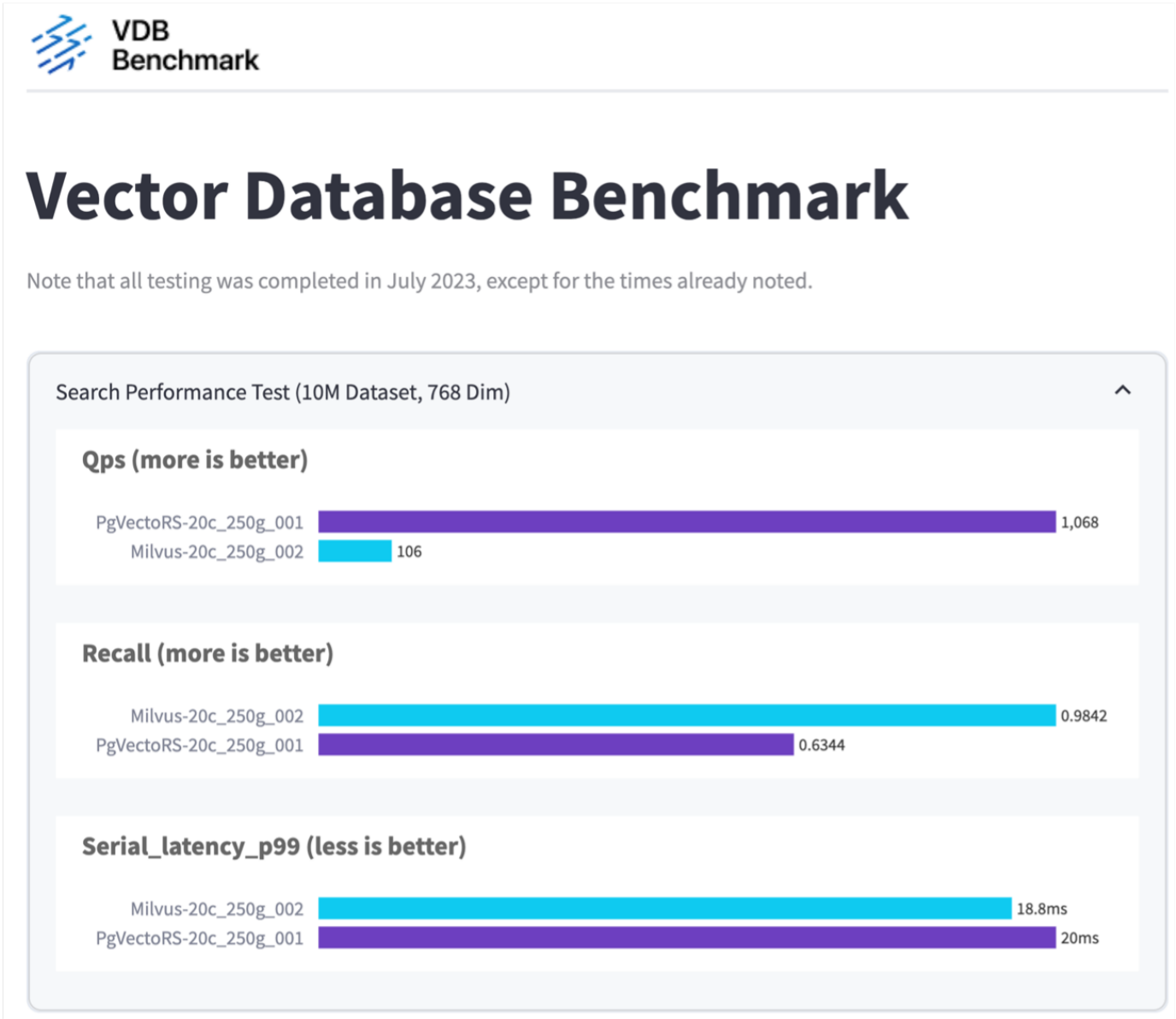
In this section, we share our observations and results from testing the PostgreSQL database, specifically using pgvecto.rs.
* We selected HNSW as the index type for these tests because at the time of testing, DiskANN wasn't available for pgvecto.rs.
* During the data ingestion phase, we loaded the Cohere dataset, which consists of 10 million vectors at a dimensionality of 768. This process took approximately 4.5 hours.
* In the query phase, we observed a Queries per Second (QPS) rate of 1,068 with a recall of 0.6344. The 99th percentile latency for queries was measured at 20 milliseconds. Throughout most of the runtime, the client CPU was operating at 100% capacity.

The images below provide a view of various storage metrics, including storage cluster latency total IOPS (Input/Output Operations Per Second).


```
The following section presents the key storage performance metrics.
```

Workload Phase	Metric	Milvus Standalone	Milvus Cluster
Data Ingestion and Post-Optimization	IOPS	< 1,000	< 1,000
	Latency	< 400 usecs	< 400 usecs
	Workload Mix	Read/Write mix, mostly writes	Read/Write mix, mostly writes
	IO Size	64 KB	64 KB
Query	IOPS	Peak at 32,000	Peak at 147,000
	Latency	< 400 usecs	< 400 usecs
	Workload Mix	100% cache reads	100% cache reads
	IO Size	Mainly 8KB	Mainly 8KB

Performance comparison between milvus and postgres on vector DB Bench



Based on our performance validation of Milvus and PostgreSQL using VectorDBBench, we observed the following:

- Index Type: HNSW
- Dataset: Cohere with 10 million vectors at 768 dimensions

We found that pgvecto.rs achieved a Queries per Second (QPS) rate of 1,068 with a recall of 0.6344, while Milvus achieved a QPS rate of 106 with a recall of 0.9842.

If high precision in your queries is a priority, Milvus outperforms pgvecto.rs as it retrieves a higher proportion of relevant items per query. However, if the number of queries per second is a more crucial factor, pgvecto.rs exceeds Milvus. It's important to note, though, that the quality of the data retrieved via pgvecto.rs is lower, with around 37% of the search results being irrelevant items.

Observation based on our performance validations:

Based on our performance validations, we have made the following observations:

In Milvus, the I/O profile closely resembles an OLTP workload, such as that seen with Oracle SLOB. The benchmark consists of three phases: Data Ingestion, Post-Optimization, and Query. The initial stages are primarily characterized by 64KB write operations, while the query phase predominantly involves 8KB reads. We expect ONTAP to handle the Milvus I/O load proficiently.

The PostgreSQL I/O profile does not present a challenging storage workload. Given the in-memory implementation currently in progress, we didn't observe any disk I/O during the query phase.

DiskANN emerges as a crucial technology for storage differentiation. It enables the efficient scaling of vector DB search beyond the system memory boundary. However, it's unlikely to establish storage performance differentiation with in-memory vector DB indices such as HNSW.

It's also worth noting that storage does not play a critical role during the query phase when the index type is HSNW, which is the most important operating phase for vector databases supporting RAG applications. The implication here is that the storage performance does not significantly impact the overall performance of these applications.

Copyright information

Trademark information