解决方案验证
NetApp Solutions

NetApp
September 26, 2024

目录

解决方案验证

解决方案概述

我们已针对五个关键领域进行了全面的解决方案验证、详细信息如下。每个部分都深入探讨了客户面临的挑战、NetApp提供的解决方案以及后续为客户带来的优势。

1. "在内部部署中使用Kubbernetes设置Milvus集群"
   客户在存储和计算、有效的基础架构管理和数据管理方面面临独立扩展的挑战。在本节中、我们将详细介绍在Kubbernetes上安装Milvus集群的过程、并利用NetApp存储控制器来存储集群数据和客户数据。
2. "Milvus与Amazon FSxN for NetApp ONTAP—文件和对象双重性"
   在本节中、我们为什么需要在云中部署向量数据库、以及在适用于NetApp ONTAP的Amazon FSxN中的Docker容器中部署向量数据库(Milvus standalone)的步骤。
3. "使用NetApp SnapCenter保护向量数据库。"
   在本节中、我们将深入探讨SnapCenter如何保护驻留在ONTAP中的向量数据库数据和Milvus数据。在本示例中、我们会使用从NFS ONTAP卷(vol1)派生的NAS存储分段(milvusdbvol1)存储客户数据、并使用单独的NFS卷(v/tordbpv)存储Milvus集群配置数据。
4. "使用NetApp SnapMirror进行灾难恢复"
   在本节中、我们将讨论向量数据库进行灾难恢复(Disaster Recovery、DR)的重要性、以及NetApp灾难恢复产品SnapMirror如何为向量数据库提供灾难恢复解决方案。
5. "性能验证"
   在本节中、我们将深入探讨Milvus和pgvector.rs等向量数据库的性能验证、重点介绍其存储性能特征、例如I/O配置文件和NetApp存储控制器在LLM生命周期内支持RAG和推导工作负载时的行为。我们将评估并确定将这些数据库与ONTAP存储解决方案结合使用时的任何性能差异。我们的分析将基于关键绩效指标、例如每秒处理的查询数量(QPS)。

在内部部署中使用**Kubbernetes**进行**Milvus**集群设置

本节讨论用于NetApp的向量数据库解决方案的Milvus群集设置。

在内部部署中使用**Kubbernetes**设置**Milvus**集群

客户在存储和计算、有效的基础架构管理和数据管理方面面临独立扩展的挑战、
Kubnetes和向量数据库共同构成一个功能强大的可扩展解决方案、用于管理大型数据操作。Kubarnetes可以优化资源并管理容器、而向量数据库则可以高效地处理高维度数据和相似性搜索。这种组合可以快速处理大型数据集上的复杂查询、并可随着不断增长的数据量无缝扩展、因此非常适合大数据应用程序和AI工作负载。

1. 在本节中、我们将详细介绍在Kubbernetes上安装Milvus集群的过程、并利用NetApp存储控制器来存储集群数据和客户数据。
2. 要安装Milvus集群、需要使用永久性卷(PV)来存储来自各种Milvus集群组件的数据。这些组成部分包括etC2(三个实例)、脉冲星-博彩记(三个实例)、脉冲星-博彩记分类账(三个实例)和脉冲星-祖-数据(三个实例)。

在Milvus集群中、我们可以使用Pulsar或Kafka作为底层引擎、支持Milvus集群可靠地存储和发布/订阅消息流。对于采用NFS的Kafka、NetApp在ONTAP 9.12.1及更高版本中进行了改进、这些增强功能以及RHEL 8.7或9.1及更高版本中包含的NFSv4.1和Linux更改解决了问题描述在基于NFS运行Kafka时可能发生的"愚蠢重命名"NFS问题。如果您对使用NetApp NFS解决方案运行Kafka这一主题的更多深入信息感兴趣、请检查- "此链接。"。

3. 我们从NetApp ONTAP创建了一个NFS卷、并建立了12个永久性卷、每个卷具有250 GB的存储空间。存储大小可能因集群大小而异；例如、我们还有一个集群、其中每个PV具有50 GB。请参阅下面的PV YAML文件之一了解更多详细信息；我们总共有12个此类文件。在每个文件中、storageClassName会设置为"默认"、并且存储和路径对于每个PV都是唯一的。

```
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. 对每个PV YAML文件执行"kubecl Apply"命令以创建永久性卷、然后使用'kubecl get pv'验证其创建情况

```
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. 为了存储客户数据、Milvus支持Minio、Azure Blb和S3等对象存储解决方案。在本指南中、我们将使用S3。以下步骤同时适用于ONTAP S3和StorageGRID对象存储。我们使用Helm部署Milvus集群。从Milvus下载位置下载配置文件values.yaml。请参阅附录、了解我们在本文档中使用的values.yaml文件。
6. 确保每个部分中的"sorageClass"设置为"efault"、包括日志、etd"、Zookekeeper和bookkeeper。
7. 在MinIO部分中、禁用MinIO。
8. 从ONTAP或StorageGRID对象存储创建NAS存储分段、并使用对象存储凭据将其包含在外部S3中。

```
###################################
# 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. 在创建Milvus集群之前、请确保PerbestentVolumeClaim (PVC)不具有任何先前已有的资源。

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

10. 使用Helm和values.yaml配置文件安装和启动Milvus集群。

```
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. 验证PersentVolumeClaims (PVCS)的状态。

```
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. 检查Pod的状态。

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

请确保Pod状态为‘running’(正在运行)且按预期工作

13. 测试Milvus和NetApp对象存储中的数据写入和读取。
    - 使用"prepy_data_NetApp_new.py" Python程序写入数据。

```
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:~#
```

    - 使用"verify_data_NetApp.py" Python文件读取数据。

```
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}}]}
```

根据上述验证、通过使用NetApp存储控制器在Kubbernetes上部署Milvus集群、Kubbernetes与向量数据库的集成为客户提供了一个强大、可扩展且高效的解决方案、用于管理大规模数据操作。这种设置使客户能够处理高维度数据并快速高效地执行复杂查询、使其成为大数据应用程序和AI工作负载的理想解决方案。对各种集群组件使用永久性卷(PV)、以及从NetApp ONTAP创建单个NFS卷、可确保最佳的资源利用率和数据管理。验证持久卷声明(PVC)和Pod状态以及测试数据写入和读取的过程、为客户提供了可靠且一致的数据操作保证。将ONTAP或StorageGRID对象存储用于客户数据可进一步增强数据可访问性和安全性。总之、这种设置为客户提供了一个具有故障恢复能力的高性能数据管理解决方案、可以根据其不断增长的数据需求无缝扩展。

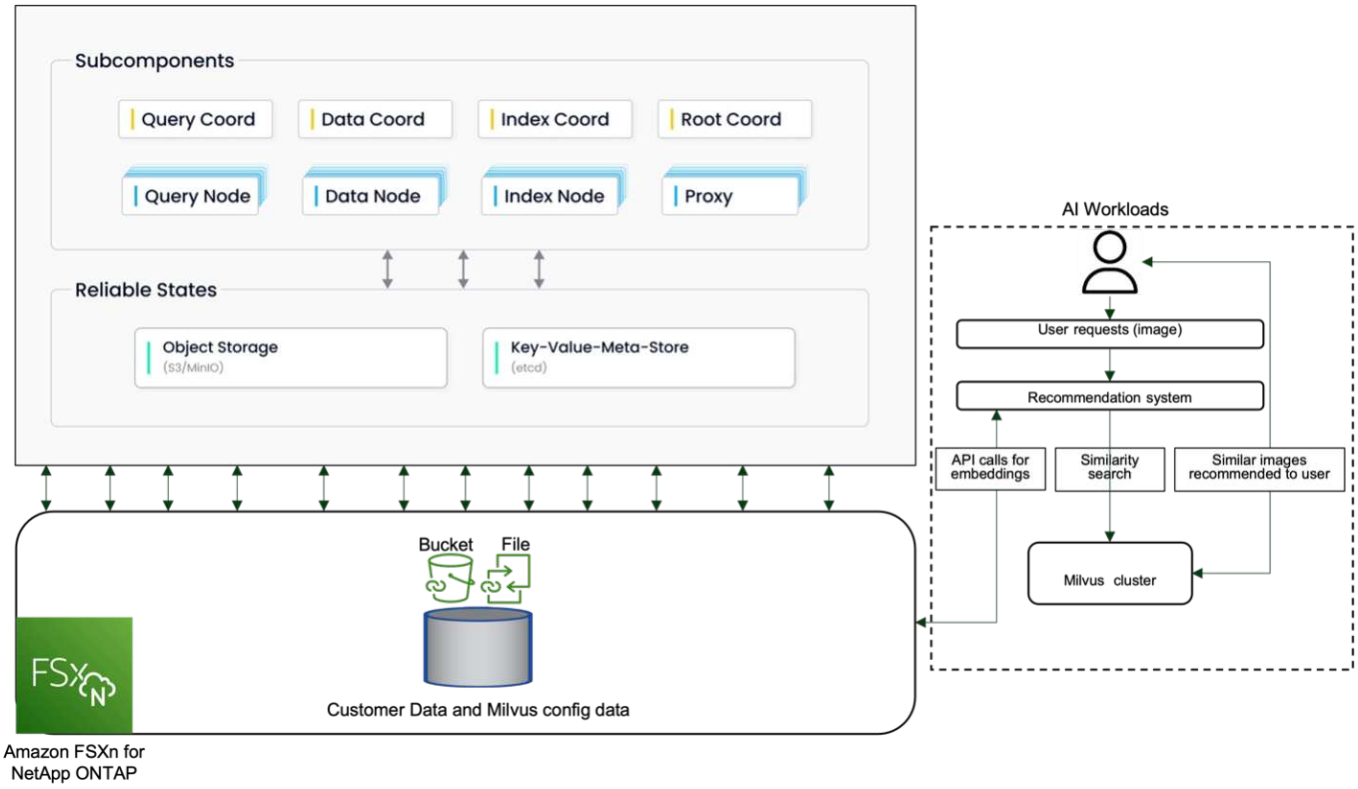
Milvus与Amazon FSxN for NetApp ONTAP—文件和对象双重性

本节讨论使用Amazon FSxN为适用于NetApp的向量数据库解决方案设置Milvus集群。

Milvus与Amazon FSxN for NetApp ONTAP—文件和对象双重性

在本节中、我们需要在云中部署向量数据库的原因以及在Docker容器中的Amazon FSxN for NetApp ONTAP中部署向量数据库(Milvus独立)的步骤。

在云中部署矢量数据库可提供多项显著优势、尤其是对于需要处理高维度数据和执行相似性搜索的应用程序。首先、基于云的部署提供可扩展性、支持轻松调整资源、以满足不断增长的数据量和查询负载的需求。这样可以确保数据库在保持高性能的同时高效地处理不断增长的需求。其次、云部署可提供高可用性和灾难恢复、因为可以在不同地理位置之间复制数据、从而最大限度地降低数据丢失的风险、并确保即使在意外事件发生时也能持续提供服务。第三、它不仅经济高效、因为您只需为所使用的资源付费、而且可以根据需要进行扩展或缩减、从而无需在前期投入大量硬件。最后、在云中部署矢量数据库可以增强协作、因为数据可以从任何位置访问和共享、从而促进基于团队的工作和数据驱动的决策。
请检查在此验证中使用的Milvus独立架构以及Amazon FSxN for NetApp ONTAP。



1. 创建适用于NetApp ONTAP的Amazon FSxN实例、并记下VPC、VPC安全组和子网的详细信息。创建EC2实例时需要此信息。有关详细信息、请单击此处- https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. 创建EC2实例、确保VPC、安全组和子网与Amazon FSxN for NetApp ONTAP实例的VPC、安全组和子网匹配。
3. 使用命令"apt-get install NFS-common"安装NFS-common、并使用"sudo apt-get update"更新软件包信息。
4. 创建一个挂载文件夹、并在此文件夹上挂载Amazon FSxN for NetApp ONTAP。

```
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. 使用"apt-get install"安装Docker和Docker准备。
6. 基于Docker compose．yaml文件设置Milvus群集，该文件可从Milvus网站下载。

```
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. 在Docker compose .yml文件的"volumes"部分中、将NetApp NFS挂载点映射到相应的Milvus容器路径、尤其是etd、minio和standalone.Check中的路径 "附录D：dkder-compose．yml" 有关yml更改的详细信息
8. 验证已挂载的文件夹和文件。

```
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. 从包含docker-compose．yml文件的目录中运行docker-compose up -d。
10. 检查Milvus容器的状态。

```
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. 为了验证向量数据库及其数据在Amazon FSxN for NetApp ONTAP中的读写功能、我们使用了Python Milvus SDK和PyMilvus的示例程序。使用"apt-get install python3-NumPy python3-pip"安装必要的软件包、并使用"pip3 install pymilvus"安装PyMilvus。
12. 验证向量数据库中Amazon FSxN for NetApp ONTAP的数据写入和读取操作。

```
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
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
```

```
91411923/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. 使用verify_data_netapp.py脚本检查读取操作。

```
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. 如果客户希望通过S3协议访问(读取)在矢量数据库中测试的AI工作负载NFS数据、可以使用简单的Python程序进行验证。例如、本节开头的图片中提到的对其他应用程序中的图像进行相似性搜索。

```
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/448789845791611912/448789845791611913/448789845791611920/0/448789845791411917/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#
```

本节有效地演示了客户如何在Docker容器中部署和运行独立的Milvus设置、并利用Amazon的NetApp FSxN来存储NetApp ONTAP数据。通过这种设置、客户可以在可扩展且高效的Docker容器环境中利用向量数据库的强大功能来处理高维数据和执行复杂查询。通过为NetApp ONTAP实例创建Amazon FSxN并匹配EC2实例、客户可以确保最佳资源利用率和数据管理。成功验证了向量数据库中FSxN的数据写入和读取操作、为客户提供了可靠且一致的数据操作保障。此外、通过S3协议列出(读取) AI工作负载中的数据、还增强了数据可访问性。因此、这一全面的流程为客户提供了一个强大而高效的解决方案、用于管理其大规模数据操作、并利用Amazon FSxN for NetApp ONTAP的功能。

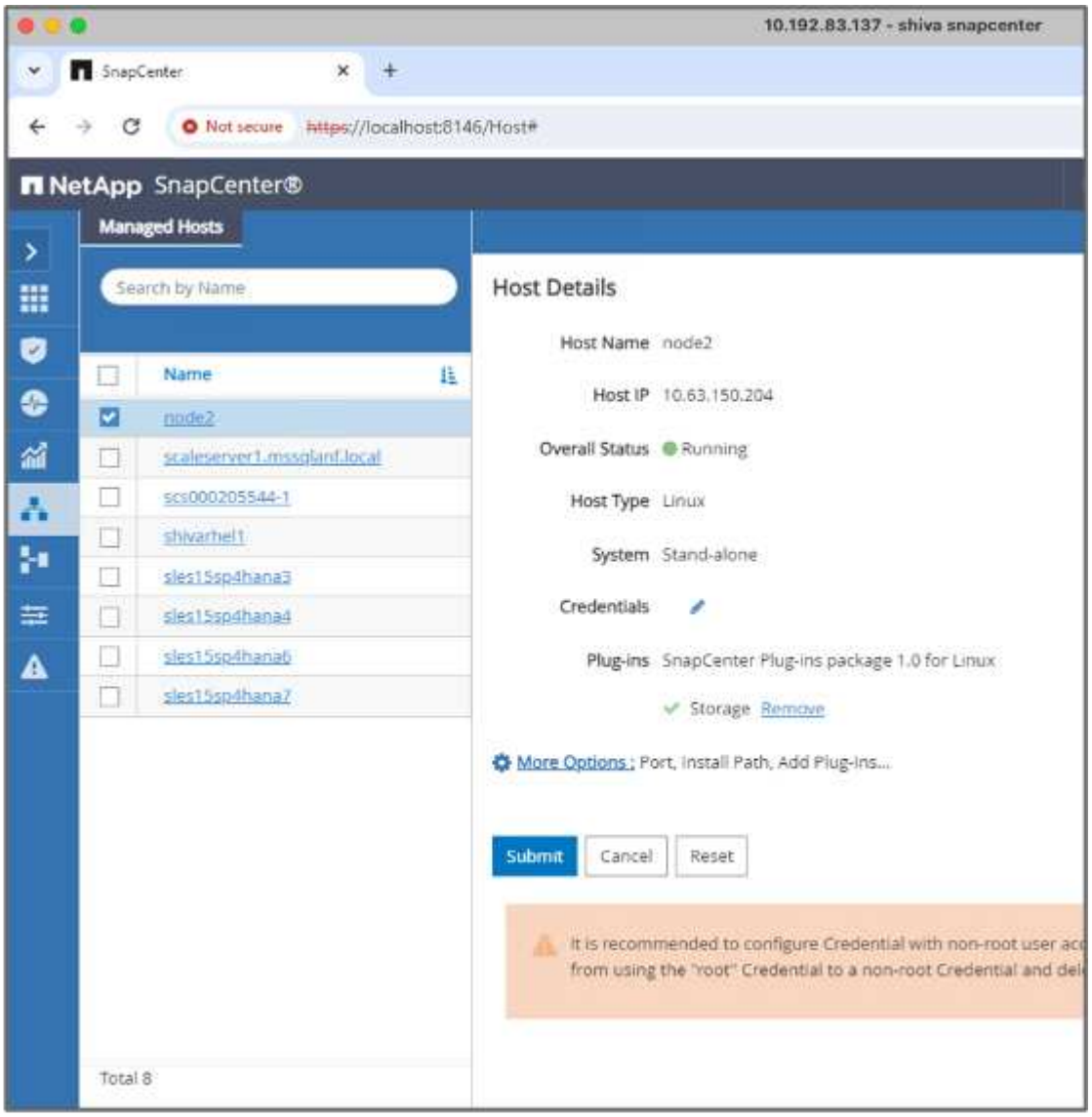
使用SnapCenter的向量数据库保护

本节介绍如何使用NetApp SnapCenter为矢量数据库提供数据保护。

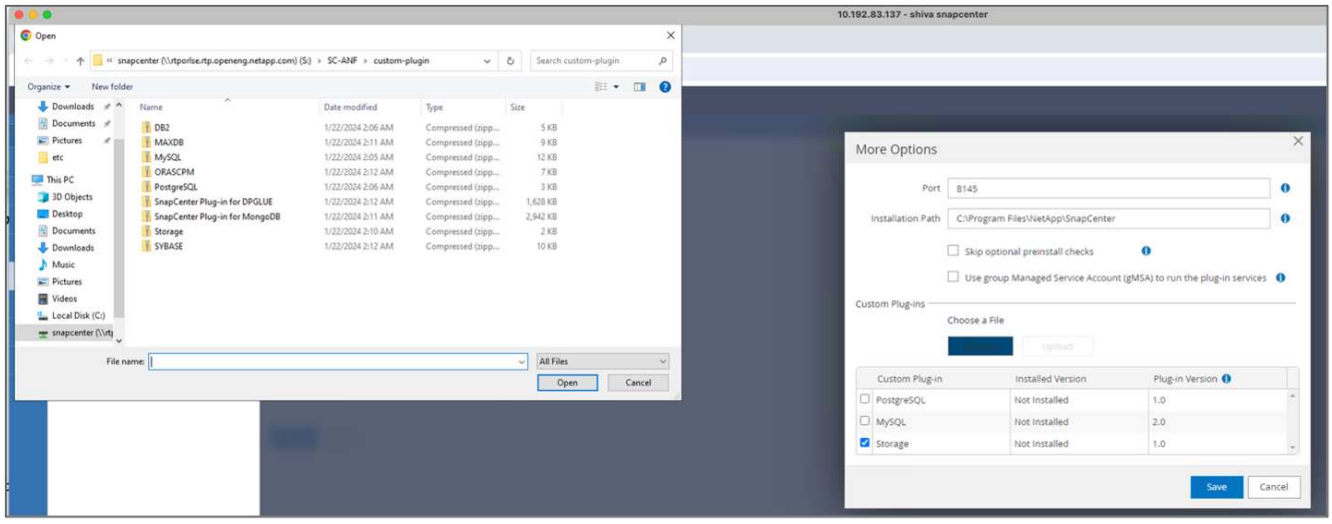
使用NetApp SnapCenter保护向量数据库。

例如、在电影制作行业、客户通常拥有视频和音频文件等关键嵌入式数据。由于硬盘故障等问题而丢失这些数据可能会对其运营产生重大影响、从而可能危及数百万美元的企业。我们遇到过宝贵的内容丢失的情况、导致了重大中断和财务损失。因此、确保这些重要数据的安全性和完整性在该行业中至关重要。
在本节中、我们将深入探讨SnapCenter如何保护驻留在ONTAP中的向量数据库数据和Milvus数据。在本示例中、我们会使用从NFS ONTAP卷(vol1)派生的NAS存储分段(milvusdbvol1)存储客户数据、并使用单独的NFS卷(v/tordbpv)存储Milvus集群配置数据。请检查 "此处" 适用于SnapCenter备份工作流

1. 设置要用于执行SnapCenter命令的主机。

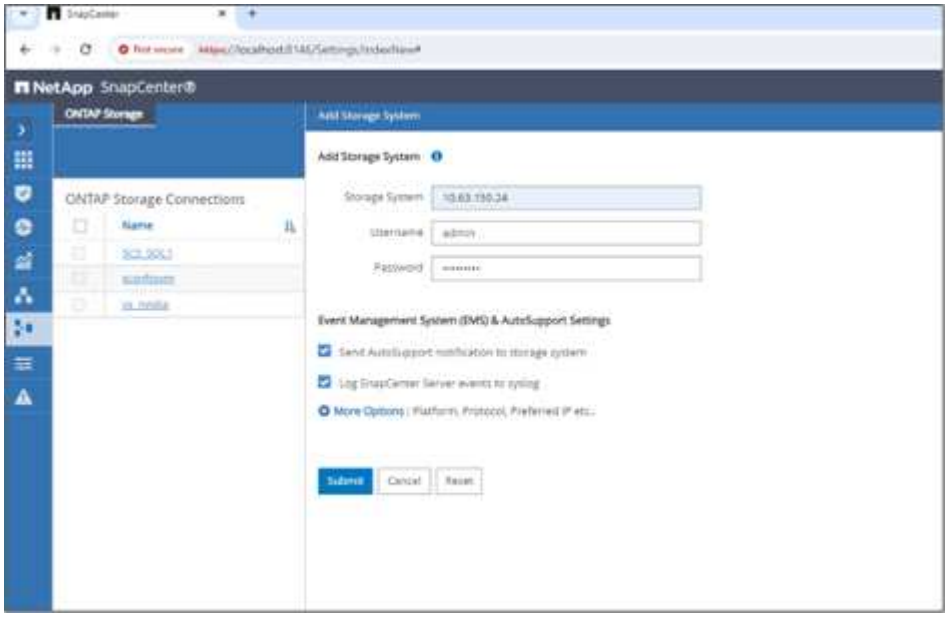


2. 安装和配置存储插件。从添加的主机中、选择"More Options (更多选项)"。导航到并从中选择已下载的存储插件 "NetApp 自动化商店"。安装插件并保存配置。

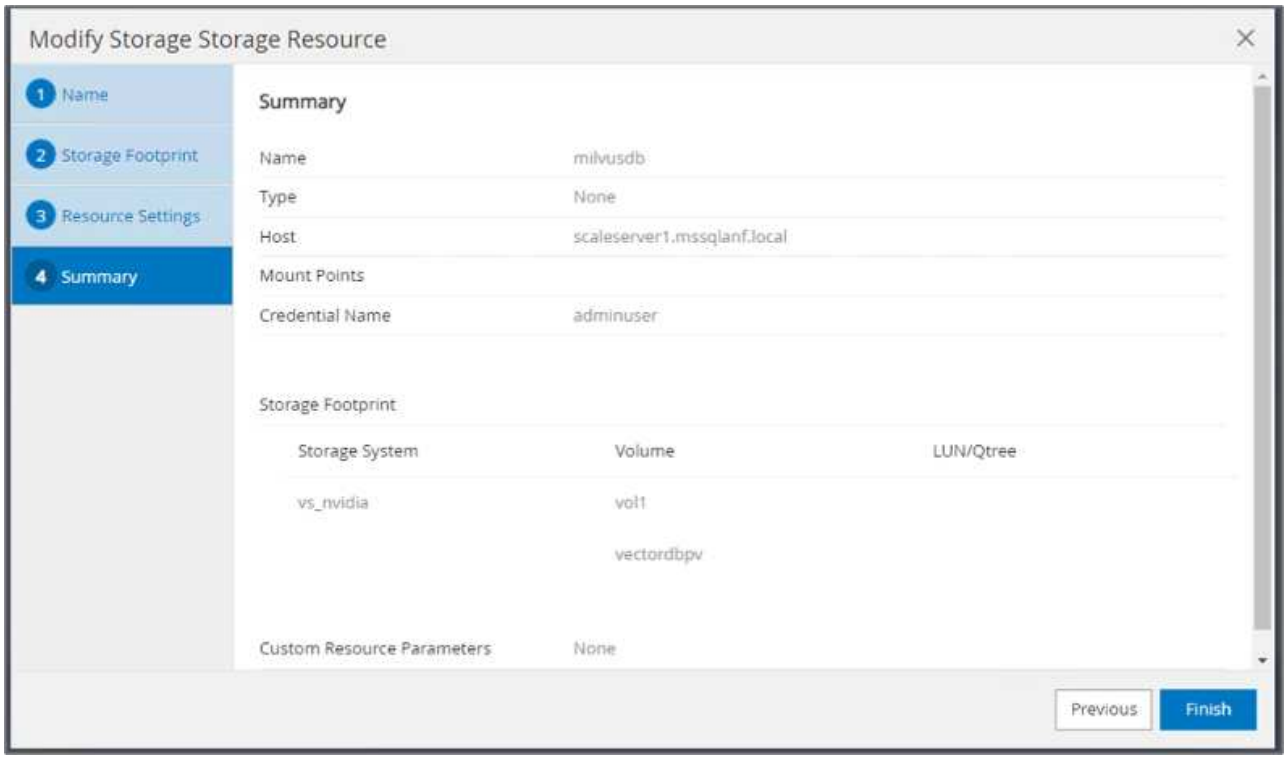


3. 设置存储系统和卷：在"存储系统"下添加存储系统、然后选择SVM (Storage Virtual Machine)。在此示例

中、我们选择了"vs_nidia"。



4. 为向量数据库建立一个资源、其中包含备份策略和自定义快照名称。
   - 使用默认值启用一致性组备份、并在文件系统不一致的情况下启用SnapCenter。
   - 在存储占用空间部分中、选择与向量数据库客户数据和Milvus集群数据关联的卷。在我们的示例中、这些卷分别为"vol1"和"v：v：ordbpv"。
   - 创建用于矢量数据库保护的策略并使用该策略保护矢量数据库资源。



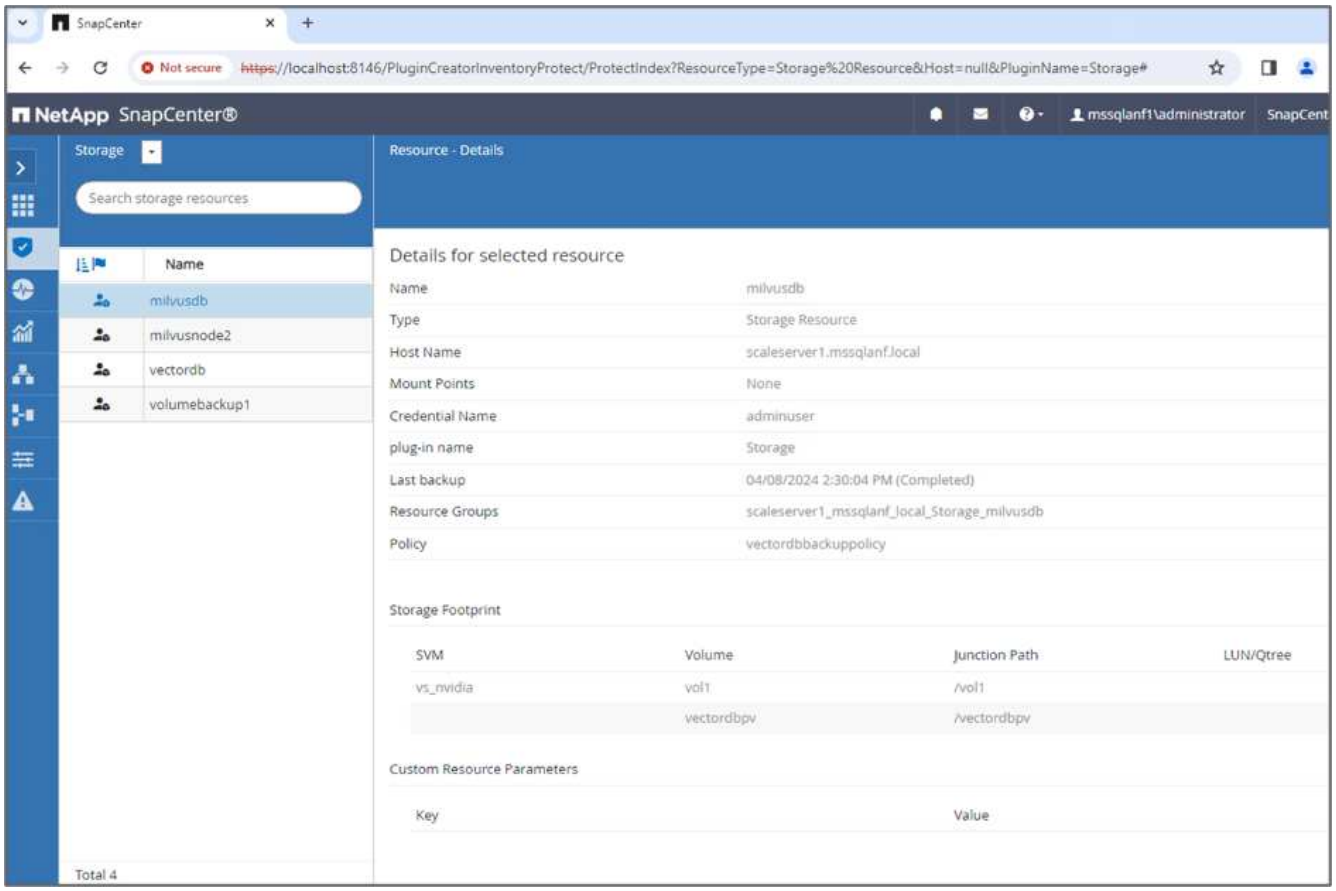
5. 使用Python脚本将数据插入S3 NAS分段。在本示例中、我们修改了Milvus提供的备份脚本、即"prepy_data_NetApp.py"、并执行了"ync"命令从操作系统中刷新数据。

```
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. 验证S3 NAS存储分段中的数据。在本示例中、时间戳为"2024-04-08 21：22"的文件是由"prepy_data_NetApp.py"脚本创建的。

```
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. 使用"ilvusdb"资源中的一致性组(CG)快照启动备份

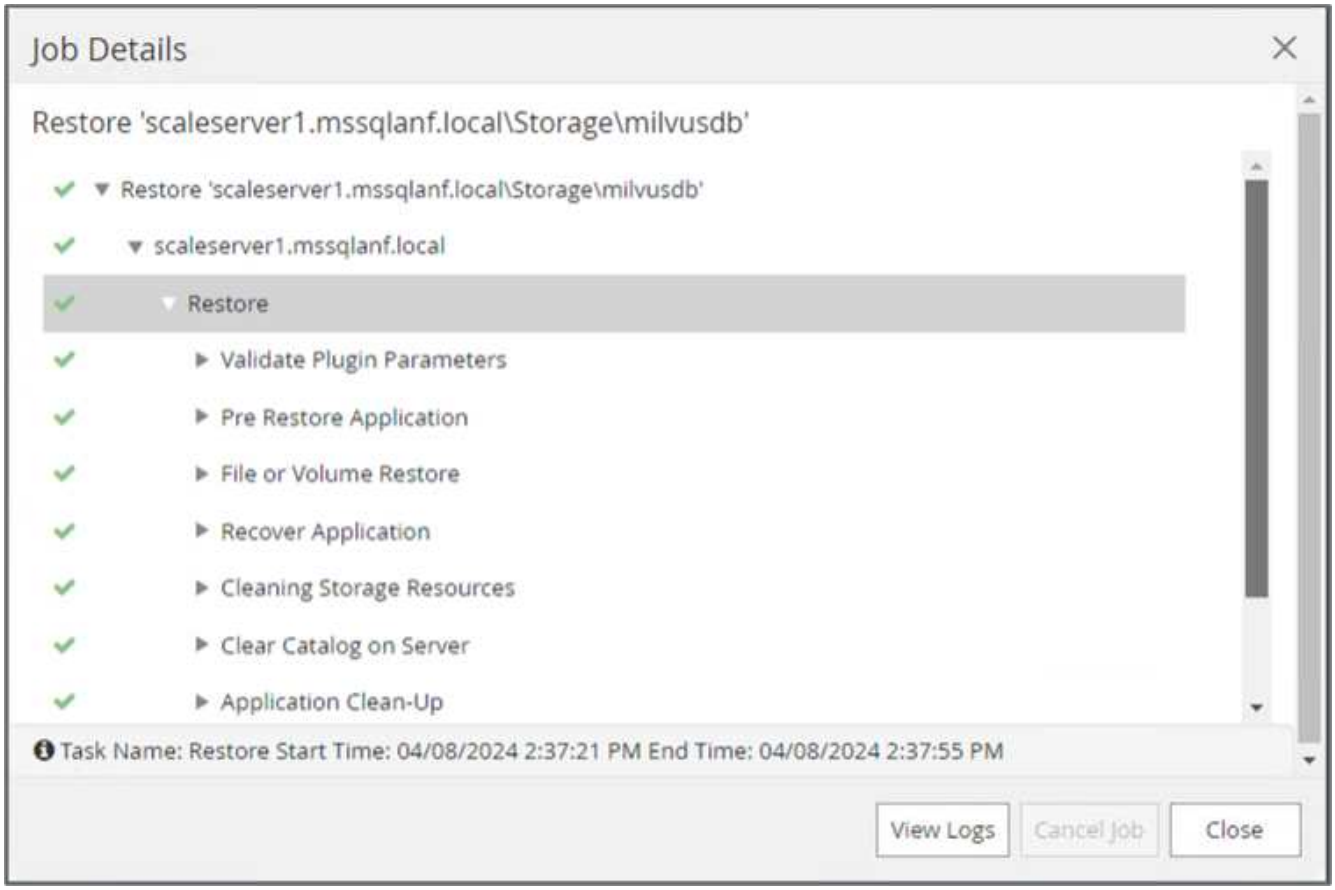


8. 为了测试备份功能、我们会在备份过程后添加一个新表、或者从NFS (S3 NAS存储分段)中删除一些数据。

   在此测试中、假设有人在备份后创建了新的、不必要的或不适当的集合。在这种情况下，我们需要在添加新集合之前将引导程序数据库还原到其状态。例如、已插入新集合、例如"hello milvus_NetApp_sc_testnew"和"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. 从上一个快照执行S3 NAS存储分段的完全还原。



10. 使用Python脚本验证"hello milvus_NetApp_sc_test"和"hello milvus_NetApp_sc_test2"集合中的数据。

```
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. 验证数据库中是否不再存在不必要或不适当的收集。

```
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:~#
```

总之、使用NetApp的SnapCenter保护矢量数据库数据以及驻留在ONTAP中的Milvus数据为客户带来了巨大的优势、尤其是在数据完整性至关重要的行业、例如电影制作。SnapCenter能够创建一致的备份并执行完整数据恢复、从而确保关键数据(例如嵌入式视频和音频文件)不会因硬盘故障或其他问题而丢失。这不仅可以防止运营中断、还可以防止出现重大财务损失。

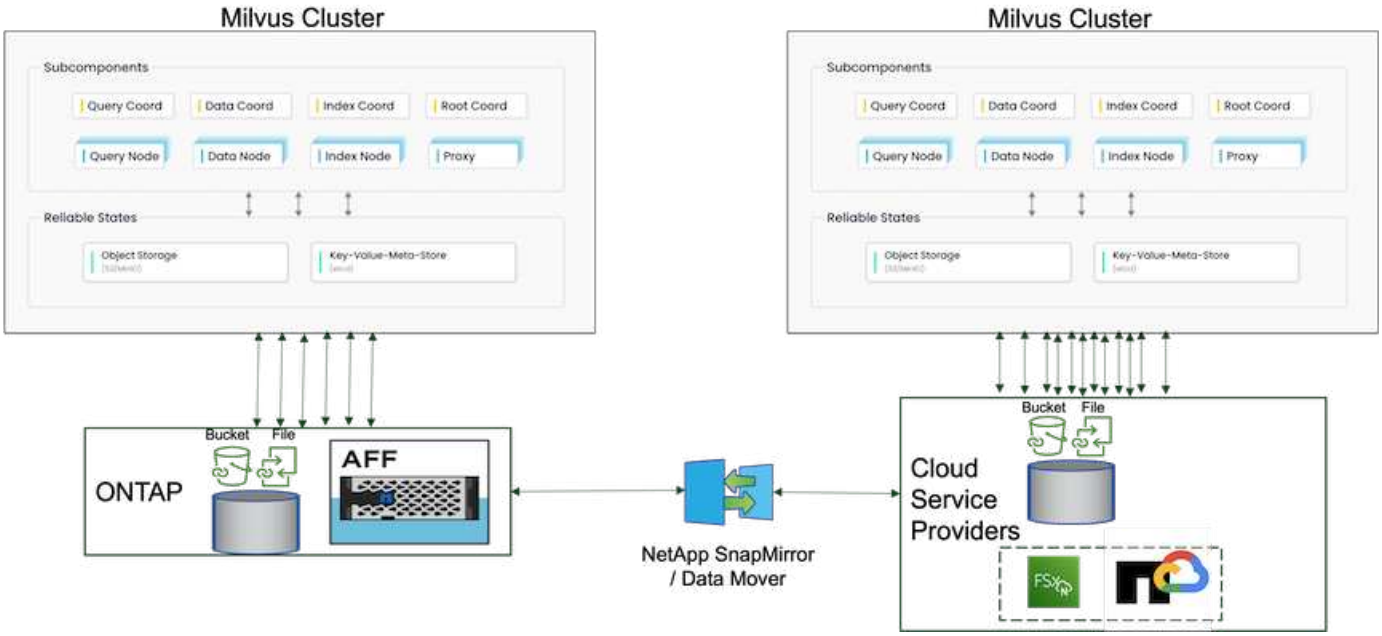
在本节中、我们演示了如何配置SnapCenter以保护驻留在ONTAP中的数据、包括设置主机、安装和配置存储插件以及使用自定义快照名称为矢量数据库创建资源。此外、我们还展示了如何使用一致性组快照执行备份并验证S3 NAS存储分段中的数据。

此外、我们还模拟了备份后创建不必要或不适当的收集的情形。在这种情况下、SnapCenter能够从先前的快照执行完全还原、从而确保向量数据库可以还原到添加新集合之前的状态、从而保持数据库的完整性。这种将数据还原到特定时间点的功能对客户来说非常重要、可以确保他们的数据不仅安全、而且维护正确。因此、NetApp的SnapCenter产品可为客户提供强大可靠的解决方案来实现数据保护和管理。

使用NetApp SnapMirror进行灾难恢复

本节将讨论适用于NetApp的向量数据库解决方案的SnapMirror灾难恢复(灾难恢复)。

使用NetApp SnapMirror进行灾难恢复



灾难恢复对于保持矢量数据库的完整性和可用性至关重要、尤其是考虑到它在管理高维数据和执行复杂的相似性搜索方面的作用。精心规划和实施的灾难恢复策略可确保在发生意外事件(例如硬件故障、自然灾害或网络攻击)时、数据不会丢失或损坏。对于依赖矢量数据库的应用程序来说、这一点尤为重要、因为数据丢失或损坏可能导致严重的运营中断和财务损失。此外、强大的灾难恢复计划还可以最大限度地减少停机时间并快速恢复服务、从而确保业务连续性。这是通过NetApp数据复制产品跨不同地理位置执行SnapMirror、定期备份和故障转移机制实现的。因此、灾难恢复不仅是一种保护措施、而且也是负责任和高效的矢量数据库管理的一个关键组成部分。

NetApp的SnapMirror可提供从一个NetApp ONTAP存储控制器到另一个存储控制器的数据复制、主要用于灾难恢复(Disaster Recovery、DR)和混合解决方案。在矢量数据库环境中、此工具有助于在内部环境和云环境之间顺利过渡数据。这种过渡无需进行任何数据转换或应用程序重构、从而提高了跨多个平台的数据管理效率和灵活性。

向量数据库方案中的NetApp混合解决方案具有更多优势：

1. 可扩展性：NetApp的混合云解决方案能够根据您的需求扩展资源。您可以将内部资源用于常规的可预测工作负载、并将Amazon FSxN for NetApp ONTAP和Google Cloud NetApp Volume (GCNV)等云资源用于高峰时段或意外负载。
2. 成本效益：NetApp的混合云模式支持您将内部资源用于常规工作负载、并且仅在需要时购买云资源、从而优化成本。这种按需购买模式可以通过NetApp instaclinstServer服务产品实现极具成本效益的优势。对于内部和主要云服务提供商、instaclinstor提供支持和咨询服务。
3. 灵活性：NetApp的混合云让您可以灵活地选择在何处处理数据。例如、您可能会选择在内部执行复杂的向量操作、在这些环境中、您的硬件功能更强大、而云中的操作强度更低。
4. 业务连续性：发生灾难时、将数据存储在NetApp混合云中可以确保业务连续性。如果内部资源受到影响、您可以快速切换到云。我们可以利用NetApp SnapMirror将数据从内部迁移到云、反之亦然。
5. 创新：NetApp的混合云解决方案还可以通过提供对尖端云服务和技术的访问来加快创新速度。NetApp在云中的创新技术、例如Amazon FSxN for NetApp ONTAP、Azure NetApp Files和Google Cloud NetApp Volumes、是云服务提供商的创新产品和首选NAS。

向量数据库性能验证

本节重点介绍对向量数据库执行的性能验证。

性能验证

性能验证在矢量数据库和存储系统中都发挥着关键作用、是确保最佳运行和高效利用资源的关键因素。向量数据库因处理高维度数据和执行相似性搜索而闻名、需要保持高性能水平、才能快速准确地处理复杂的查询。性能验证有助于识别瓶颈、微调配置、并确保系统可以处理预期负载而不会降低服务质量。同样、在存储系统中、性能验证对于确保高效存储和检索数据至关重要、不会出现可能影响整体系统性能的延迟问题或瓶颈。它还有助于在存储基础架构的必要升级或更改方面做出明智的决策。因此、性能验证是系统管理的一个关键方面、它可以显著提高服务质量、运营效率和整体系统可靠性。

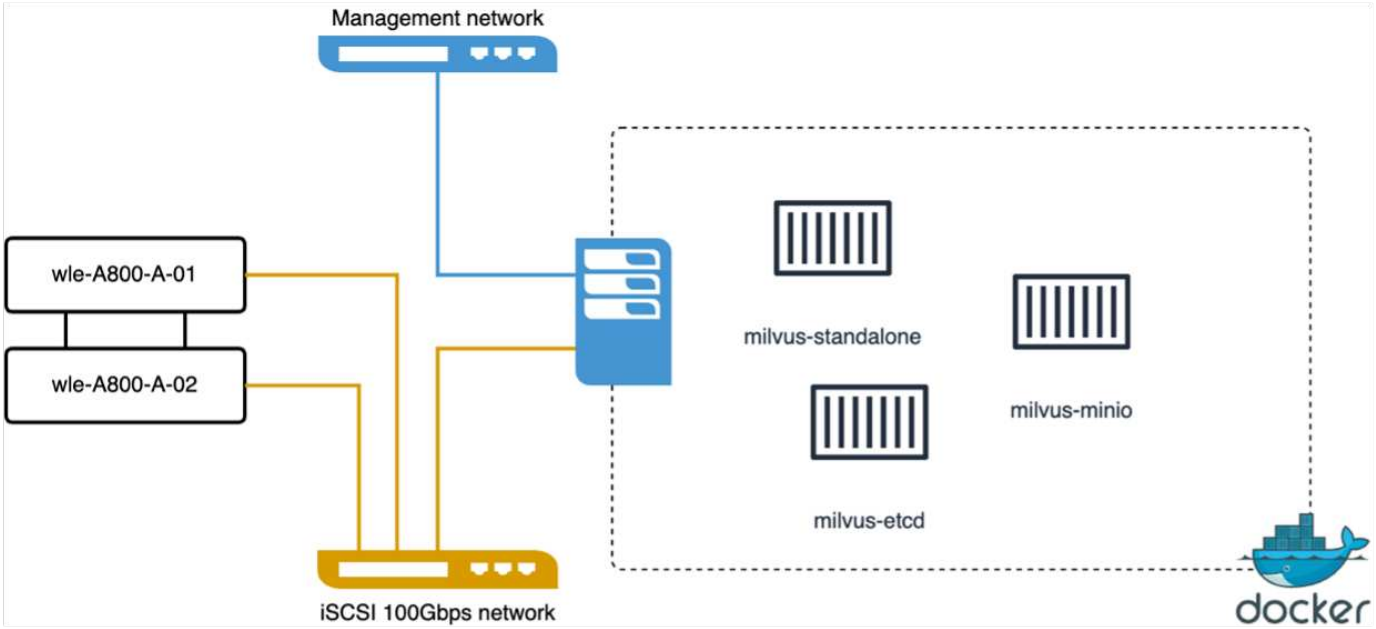
在本节中、我们将深入探讨Milvus和pgvector.rs等向量数据库的性能验证、重点介绍其存储性能特征、例如I/O配置文件和NetApp存储控制器在LLM生命周期内支持RAG和推导工作负载时的行为。我们将评估并确定将这些数据库与ONTAP存储解决方案结合使用时的任何性能差异。我们的分析将基于关键绩效指标、例如每秒处理的查询数量(QPS)。

请在下面检查用于milvus的方法和进度。

详细信息	Milvus (独立和集群)	Postgre (pg向 量.rs)#
version	2.3.2.	0.2.0
文件系统	iSCSI LUN上的XFS	
工作负载生成器	"向数据库平台" v0.0.5	
数据集	LAION数据集 * 1000万个嵌入项 * 768尺寸 *~300 GB数据集大小	
存储控制器	AFF 800 版本**—9.14.1 * 4** 个**100GbE—**适用于**Milvus**、**2** 个**100GbE**适用于后端 iSCSI	

VittorDB-Bench与**Milvus**独立集群

我们使用vittorDB-Bench对Milvus独立集群进行了以下性能验证。
Milvus独立集群的网络和服务器连接如下。



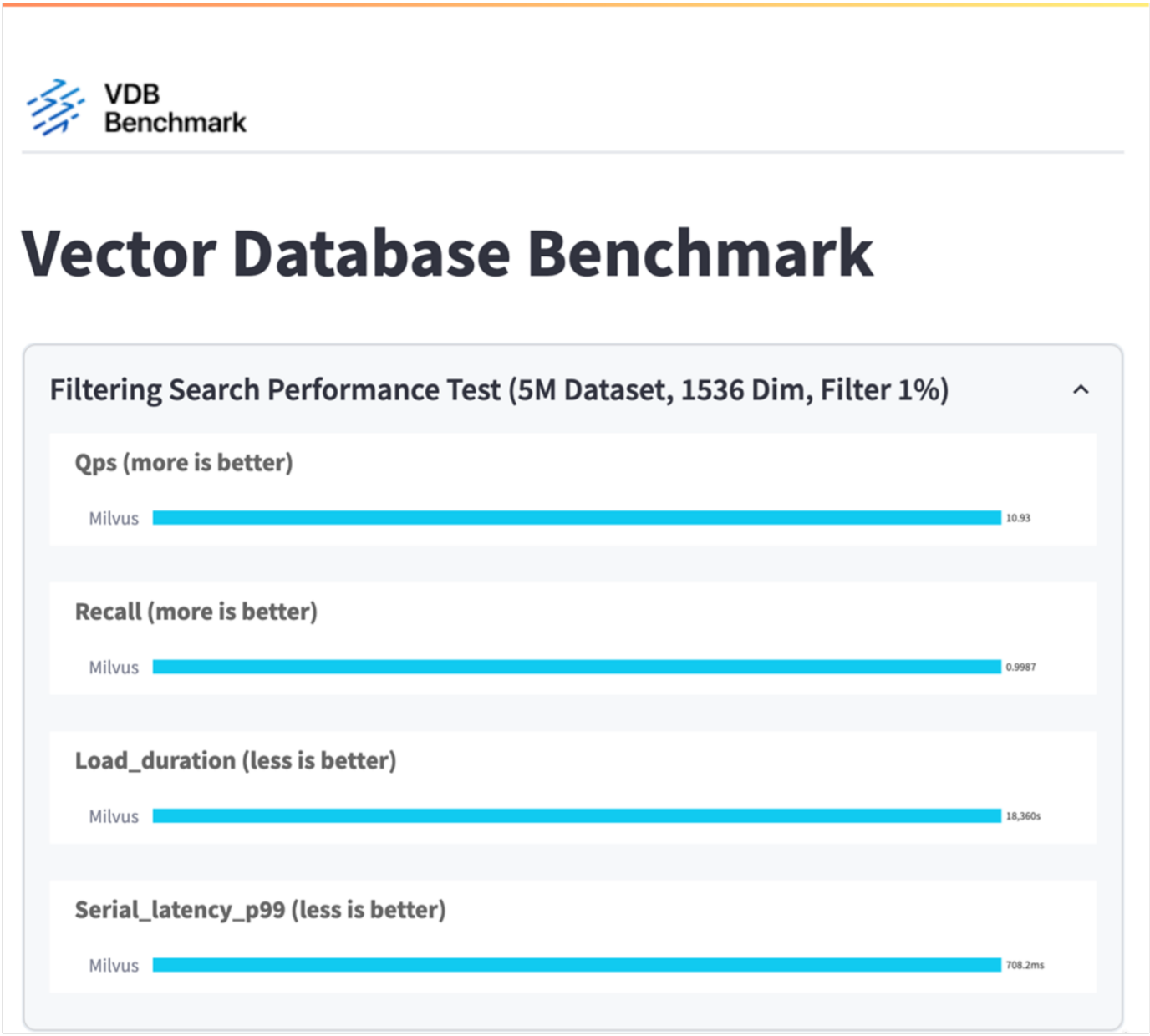
在本节中、我们将分享测试Milvus独立数据库时观察到的结果。
。 我们选择DiskANN作为这些测试的索引类型。
。 为大约100 GB的数据集执行数据导入、优化和创建索引大约需要5小时。在这段时间内的大部分时间内、配备20个核心(启用超线程时相当于40个vCPU)的Milvus服务器以其100%的最大CPU容量运行。我们发现、DiskANN对于超过系统内存大小的大型数据集尤为重要。
。 在查询阶段、我们观察到每秒查询数(Queries Per Second、QPS)比率为10.93、而调用率为0.9987。查询延迟的第99个百位点在708.2毫秒处测量。

从存储角度来看、在加载、插入后优化和索引创建阶段、数据库发出的操作数大约为1、000次/秒。在查询阶段、它需要32,000次操作/秒

下一节介绍了存储性能指标。

工作负载阶段	衡量指标	价值
数据载入 和 刀片后优化	IOPS	< 1、000
	延迟	小于400美元
	工作负载	读/写混合、大多数为写入
	IO大小	64 KB
查询	IOPS	峰值为32、000
	延迟	小于400美元
	工作负载	100%缓存读取
	IO大小	主要为8 KB

以下是以下的bittorDB-bench结果。

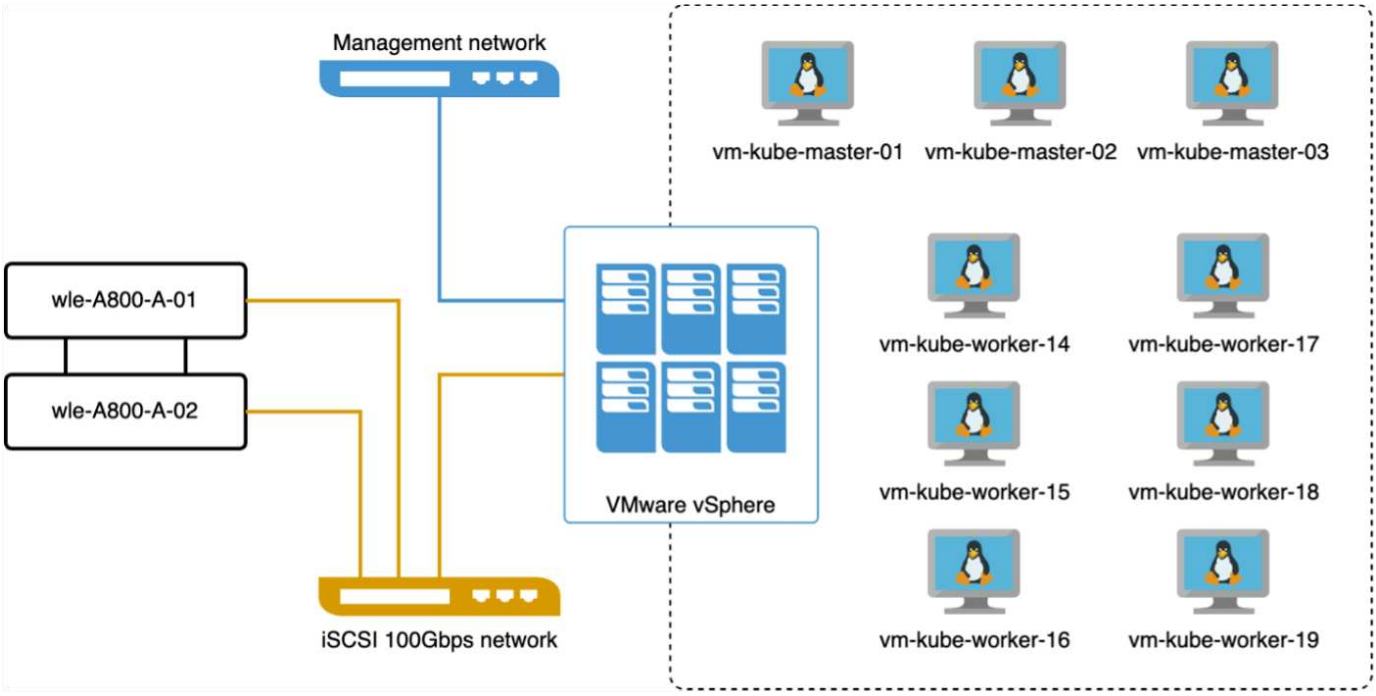


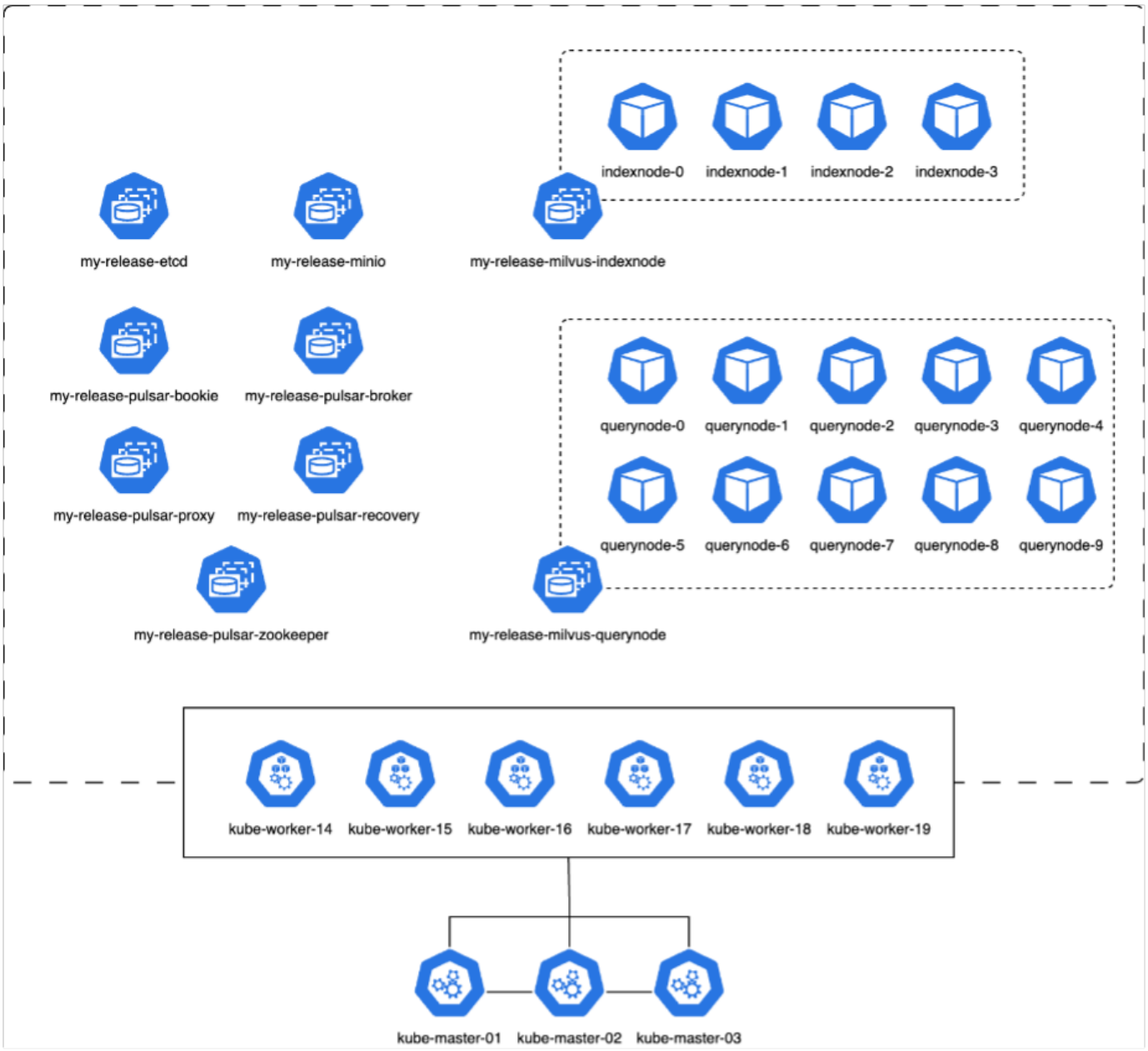
从独立Milvus实例的性能验证来看、很明显、当前设置不足以支持一个包含500万向量且维度为1536的数据集。我们已确定存储具有足够的资源、不会在系统中构成瓶颈。

VittorDB-Bench与Milvus集群

在本节中、我们将讨论如何在Kubbernetes环境中部署Milvus集群。此Kubnetes设置是在VMware vSphere部署基础上构建的、该部署托管Kubnetes主节点和工作节点。

以下各节介绍了VMware vSphere和Kubnetes部署的详细信息。


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



在本节中、我们将介绍测试Milvus数据库时观察到的结果。
*使用的索引类型为DiskANN。
*下表提供了在维数为1536的情况下处理500万向量时独立部署与集群部署之间的比较。我们发现、在集群部署中、数据采集和插入后优化所需的时间较短。与独立设置相比、集群部署中查询延迟的第99个百位数减少了6倍。
*虽然集群部署中的每秒查询数(QPS)率较高、但并未达到所需的级别。

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%

下图提供了各种存储指标的视图、包括存储集群延迟和总IOPS (每秒输入/输出操作数)。

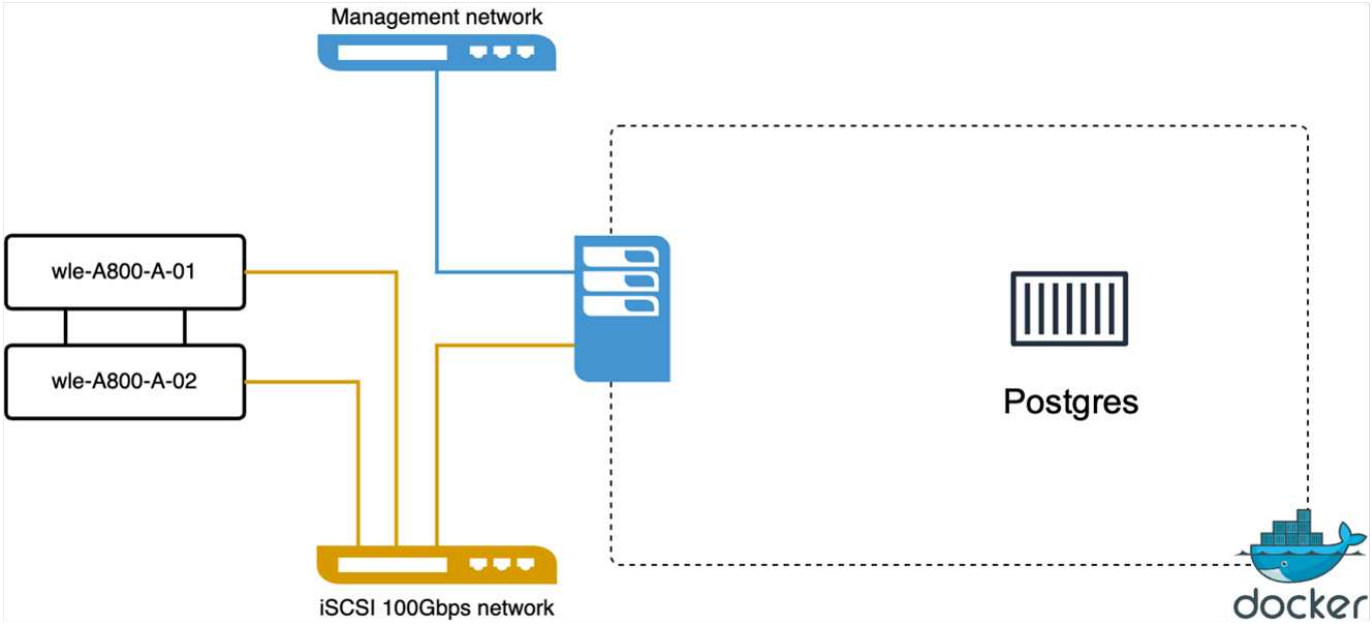
下一节介绍了主要的存储性能指标。

工作负载阶段	衡量指标	价值
数据载入 和 刀片后优化	IOPS	< 1、000
	延迟	小于400美元
	工作负载	读/写混合、大多数为写入
	IO大小	64 KB
查询	IOPS	峰值为147、000
	延迟	小于400美元
	工作负载	100%缓存读取
	IO大小	主要为8 KB

根据独立Milvus和Milvus集群的性能验证、我们将提供存储I/O配置文件的详细信息。
*我们发现、无论是独立部署还是集群部署、I/O配置文件都保持一致。
*峰值IOPS的观察到差异可能是由于集群部署中的客户端数量较多所致。

使用**Postgre**的向量数据库工作台**(pg**向 量**.rs)**

我们使用了向量数据库对PostgreSQL (pg向 量.rs)执行了以下操作：
有关PostgreSQL (特别是pg向 量.rs)的网络和服务器连接的详细信息如下：

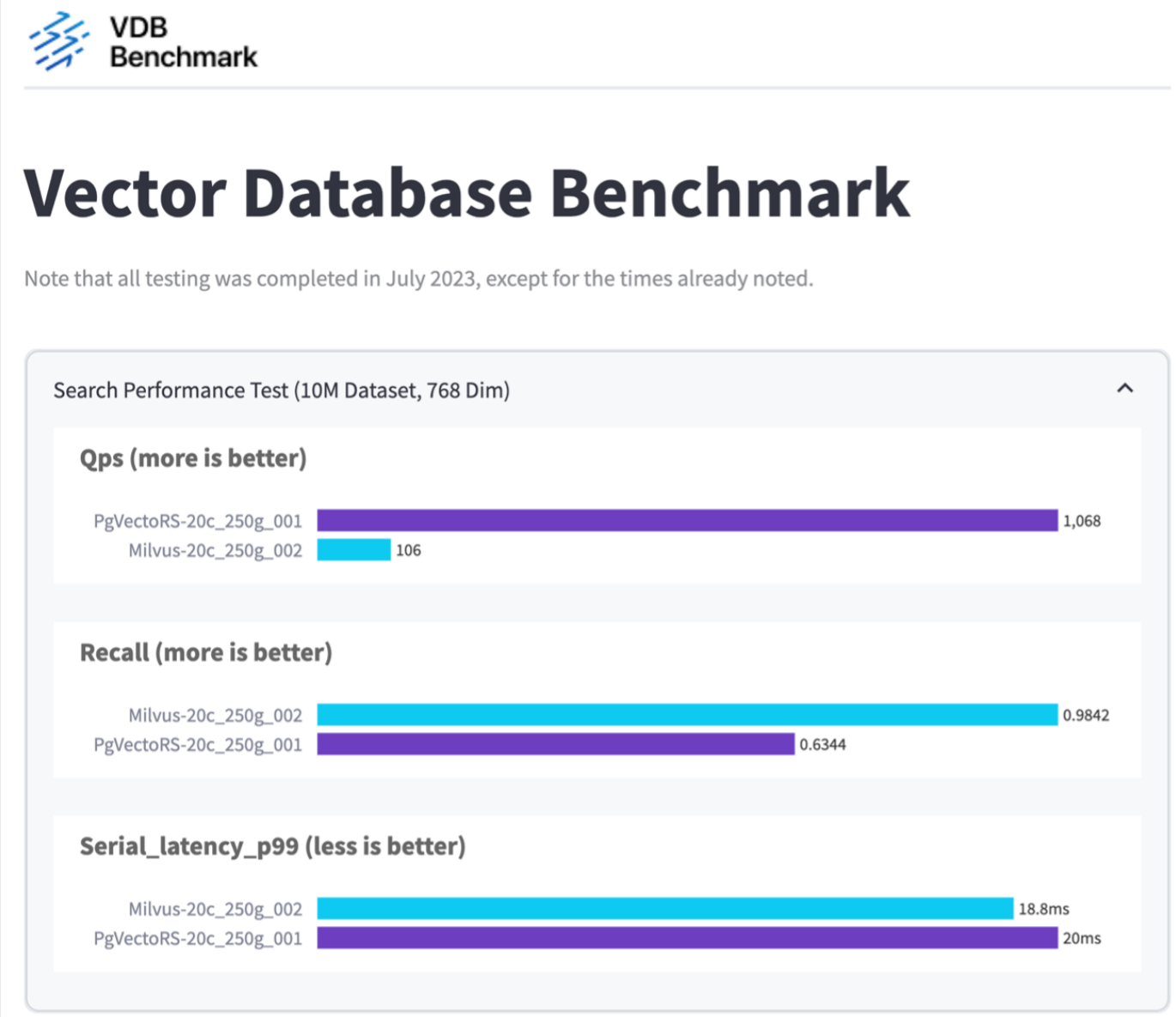


在本节中、我们将分享对PostgreSQL数据库(尤其是使用pg向 量.rs)进行测试后观察到的结果。
*我们选择HNSW作为这些测试的索引类型、因为在测试时、DiskANN不适用于pg向 量.rs。
*在数据载入阶段、我们加载了cothere数据集、该数据集由1、000万个向量组成、维数为768。此过程大约需要4.5小时。
*在查询阶段、我们观察到每秒查询数(Queries Per Second、QPS)比率为1、068、而调用率为0.6344。查询延迟的第99个百位点在20毫秒处测量。在运行时的大部分时间内、客户端CPU都以100%的容量运行。

下图提供了各种存储指标的视图、包括存储集群延迟总IOPS (每秒输入/输出操作数)。


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["图中显示了输入/输出对话框或表示已写入内容
"]
```

Vector DB Bench上的**Milvus**与**postgres**之间的性能比较



根据我们使用VittorDBBench对Milvus和PostgreSQL进行的性能验证、我们观察到以下情况：

- 索引类型：HNSW
- 数据集：具有768个维度的1000万向量

我们发现、pgvrecto .rs的每秒查询数(Queries Per Second、QPS)为1、068、召回率为0.6344、而Milvus的召回率为106、召回率为0.9842。

如果查询的高精度是优先事项、则Milvus的性能会优于pgvitou.rs、因为它会在每个查询中检索更高比例的相关项。但是、如果每秒查询数是一个更关键的因素、则pgvECG.rs将超过Milvus。但是、需要注意的是、通过pg向量.rs检索的数据质量较低、大约37%的搜索结果是不相关的项目。

根据我们的性能验证进行观察：

根据我们的性能验证、我们观察到以下情况：

在Milvus中、I/O配置文件与OLTP工作负载非常相似、例如Oracle slob中的工作负载。基准测试由三个阶段组成：数据采集、优化后和查询。初始阶段的特征主要是64 KB写入操作、而查询阶段主要涉及8 KB读取。我们希

望ONTAP能够出色地处理Milvus I/O负载。

PostgreSQL I/O配置文件不会产生具有挑战性的存储工作负载。鉴于当前正在实施内存、我们在查询阶段未发现任何磁盘I/O。

DiskANN成为实现存储差异化优势的关键技术。它可以高效地将矢量数据库搜索扩展到系统内存边界之外。但是、使用HNSW等内存向量数据库索引不太可能建立存储性能差异。

此外、还需要注意的是、当索引类型为HSNW时、存储在查询阶段并不起关键作用、HSNW是支持RAG应用程序的矢量数据库最重要的操作阶段。此处的含义是、存储性能不会对这些应用程序的整体性能产生显著影响。

版权信息

商标信息