使用NetApp的Vector数据库解决方案

NetApp Solutions

NetApp
May 03, 2024

目录

使用NetApp的Vector数据库解决方案

Karthiskeyan Nagalingam和NetApp的RODARGO NAsciamEN托

本文档全面探讨了如何使用NetApp的存储解决方案部署和管理矢量数据库、例如Milvus和pgvector、这是一个开源PostgreSQL扩展。其中详细介绍了使用NetApp ONTAP和StorageGRID对象存储的基础架构准则、并验证了Milvus数据库在AWS FSx for NetApp ONTAP中的应用。本文档阐述了NetApp的文件-对象双重性及其对支持矢量内置的矢量数据库和应用程序的实用程序。它强调了NetApp企业管理产品SnapCenter在为矢量数据库提供备份和还原功能、确保数据完整性和可用性方面的功能。本文档进一步深入探讨了NetApp的混合云解决方案、讨论了它在内部环境和云环境中的数据复制和保护方面的作用。其中深入介绍了NetApp ONTAP上向量数据库的性能验证、最后介绍了两个有关生成性AI的实际用例：RAG with LLM和NetApp的内部ChatAI。本文档是利用NetApp存储解决方案管理矢量数据库的综合指南。

参考架构侧重于以下方面：

简介

简介

向量数据库可以有效地应对旨在处理大型语言模型(LLM)和生成式人工智能(AI)中复杂语义搜索的挑战。与传统的数据管理系统不同，矢量数据库能够处理和搜索各种类型的数据，包括图像、录像、文字、音频、 和其他形式的非结构化数据、使用数据本身的内容、而不是标签或标记。

关系数据库管理系统(Relational Database Management Systems、RDBMS)的局限性已有充分的记录、尤其是它们在处理AI应用程序中常见的高维度数据表示和非结构化数据方面的困难。RDBMS通常需要一个耗时且容易出错的过程、将数据合并为更易于管理的结构、从而导致搜索延迟和效率低下。然而、矢量数据库旨在规避这些问题、提供更高效、更准确的解决方案来管理和搜索复杂的高维度数据、从而促进AI应用程序的发展。

本文档为当前正在使用或计划使用向量数据库的客户提供了全面的指导、其中详细介绍了在NetApp ONTAP 、NetApp StorageGRID、Amazon FSxN for NetApp ONTAP和SnapCenter等平台上使用向量数据库的最佳实践。本文档提供的内容涵盖一系列主题：

- NetApp存储通过NetApp ONTAP和StorageGRID对象存储提供的矢量数据库(如Milvus)基础架构准则。
- 通过文件和对象存储验证AWS FSx for NetApp ONTAP中的Milvus数据库。
- 深入了解NetApp的文件-对象双重性、展示其对矢量数据库以及其他应用程序中数据的实用程序。
- NetApp的数据保护管理产品SnapCenter如何为矢量数据库数据提供备份和还原功能。
- NetApp的混合云如何跨内部环境和云环境提供数据复制和保护。
- 深入了解NetApp ONTAP上的Milvus和pgvector等向量数据库的性能验证。
- 两个特定用例：使用大型语言模型(LLM)的检索增强型生成(RAG)和NetApp IT团队的ChatAI、从而提供所述概念和实践的实际示例。

解决方案概述

解决方案概述

本解决方案展示了NetApp为应对向量数据库客户所面临的挑战而带来的独特优势和功能。通过利用NetApp ONTAP、StorageGRID、NetApp的云解决方案和SnapCenter、客户可以为其业务运营带来显著价值。这些工具不仅可以解决现有问题、还可以提高效率和生产力、从而促进整体业务增长。

为什么选择**NetApp**?

- NetApp的产品(例如ONTAP和StorageGRID)支持将存储和计算分离、从而根据特定需求实现最佳资源利用率。这种灵活性使客户能够使用NetApp存储解决方案独立扩展其存储。
- 通过利用NetApp的存储控制器、客户可以使用NFS和S3协议高效地向其矢量数据库提供数据。这些协议有助于客户存储数据并管理向量数据库索引、从而无需通过文件和对象方法访问多个数据副本。
- NetApp ONTAP为AWS、Azure和Google Cloud等领先云服务提供商的NAS和对象存储提供本机支持。这种广泛的兼容性可确保无缝集成、从而实现客户数据移动性、全局可访问性、灾难恢复、动态可扩展性和高性能。
- 借助NetApp强大的数据管理功能、客户可以放心地知道他们的数据受到了良好的保护、可以抵御潜在风险和威胁。NetApp优先考虑数据安全、让客户高枕无忧、因为客户可以放心地了解其宝贵信息的安全性和完整性。

向量数据库

向量数据库

向量数据库是一种专用的数据库类型、旨在使用机器学习模型中的内推数据处理、索引和搜索非结构化数据。它不是以传统的表格形式组织数据、而是将数据安排为高维向量、也称为向量内置。这种独特的结构使数据库能够更高效、更准确地处理复杂的多维数据。

矢量数据库的关键功能之一是使用生成型AI执行分析。这包括相似性搜索(数据库可识别与给定输入类似的数据点)和异常检测(可发现明显偏离标准的数据点)。

此外、向量数据库非常适合处理时间数据或带时间戳的数据。此类数据提供有关'发生了什么'和发生时间、发生顺序以及给定IT系统中所有其他事件的相关信息。这种处理和分析时间数据的能力使向量数据库对于需要了解随时间变化的事件的应用程序特别有用。

ML和**AI**的矢量数据库的优势：

- 高维搜索：矢量数据库在管理和检索高维数据方面表现出色、这些数据通常在AI和ML应用程序中生成。
- 可扩展性：它们可以高效扩展以处理大量数据、从而支持AI和ML项目的增长和扩展。
- 灵活性：向量数据库提供高度的灵活性、允许容纳不同类型的数据和结构。
- 性能：它们提供高性能数据管理和检索、对于提高AI和ML操作的速度和效率至关重要。
- 可自定义索引编制：向量数据库提供可自定义的索引编制选项、可根据特定需求优化数据组织和检索。

向量数据库和用例。

本节提供了不同的矢量数据库及其使用情形详细信息。

和**ScaNN**

它们是在矢量搜索领域中用作关键工具的库。这些库提供的功能有助于管理和搜索矢量数据、使其成为这一数据管理专业领域的宝贵资源。

Elasticsearch

它是一个广泛使用的搜索和分析引擎、最近加入了矢量搜索功能。这一新功能增强了其功能，使其能够更有效地处理和搜索矢量数据。

针酮

它是一个强大的矢量数据库，具有一组独特的功能。它的索引功能既支持密集向量、也支持稀疏向量、从而提高了其灵活性和适应性。其主要优势之一在于、它能够将传统搜索方法与基于AI的密集矢量搜索相结合、从而创建一种混合搜索方法、充分利用这两种环境的优势。

Pinecone主要基于云、专为机器学习应用程序而设计、可与各种平台完美集成、包括GCP、AWS、Open AI、GPT-3、 GPT-3.5、GPT-4、Catgt Plus、ElatcSearch、HayStack、 等等。需要注意的是、Pinecone是一个闭源平台、可作为软件即服务(Software as a Service、SaaS)产品提供。

鉴于Pinecone的高级功能、它特别适合网络安全行业、在该行业中、可以有效地利用其高维度搜索和混合搜索功能来检测和应对威胁。

色度

它是一个矢量数据库、具有具有四个主要功能的Core-API、其中一个功能包括内存文档矢量存储。它还利用Face Transformers库对文档进行向量化、从而增强其功能和多功能性。
Chroma设计用于在云端和内部环境中运行、可根据用户需求提供灵活性。特别是、它在音频相关应用程序中表现出色、是基于音频的搜索引擎、音乐推荐系统和其他音频相关用例的绝佳选择。

韦维瓦特

它是一个多功能矢量数据库、允许用户使用其内置模块或自定义模块对其内容进行矢量化、从而根据特定需求提供灵活性。它既提供完全托管的解决方案、也提供自行托管的解决方案、可满足各种部署首选项的要求。

We勤 威的主要功能之一是能够存储向量和对象、从而增强了数据处理能力。它广泛用于各种应用程序、包括ERP系统中的语法搜索和数据分类。在电子商务领域，它为搜索和推荐引擎提供支持。此外、Weanate还可用于图像搜索、异常检测、自动化数据协调和网络安全威胁分析、展示其在多个域中的多功能性。

Redis

Redis是一种高性能矢量数据库、以其快速内存存储而闻名、可为读写操作提供低延迟。因此、对于需要快速访问数据的推荐系统、搜索引擎和数据分析应用程序来说、它是一个绝佳的选择。

Redis支持向量的各种数据结构、包括列表、集和排序集。它还提供矢量操作，如计算矢量之间的距离或查找交点和联合。这些功能对于相似性搜索、集群和基于内容的建议系统尤其有用。

在可扩展性和可用性方面、Redis在处理高吞吐量工作负载方面表现出色、并提供数据复制功能。它还可以与其他数据类型(包括传统关系数据库(RDBMS))完美集成。
Redis包括用于实时更新的发布/订阅(发布/订阅)功能、这对于管理实时向量非常有用。此外、Redis的重量轻且易于使用、使其成为用于管理矢量数据的用户友好型解决方案。

Milvus

它是一个多功能向量数据库、提供类似于文档存储的API、与MongoDB非常相似。它因支持多种数据类型而脱颖而出、成为数据科学和机器学习领域的热门选择。

Milvus的独特功能之一是其多矢量化功能，它允许用户在运行时指定用于搜索的矢量类型。此外、它还利用位于其他库(如法斯库)之上的Knowwhere库来管理查询和向量搜索算法之间的通信。

由于与PyTorch和TensorFlow兼容、Milvus还可以与机器学习工作流无缝集成。这使得它成为一系列应用程序的出色工具、包括电子商务、图像和视频分析、对象识别、图像相似性搜索和基于内容的图像检索。在自然语言处理领域、Milvus用于文档集群、语义搜索和问题解答系统。

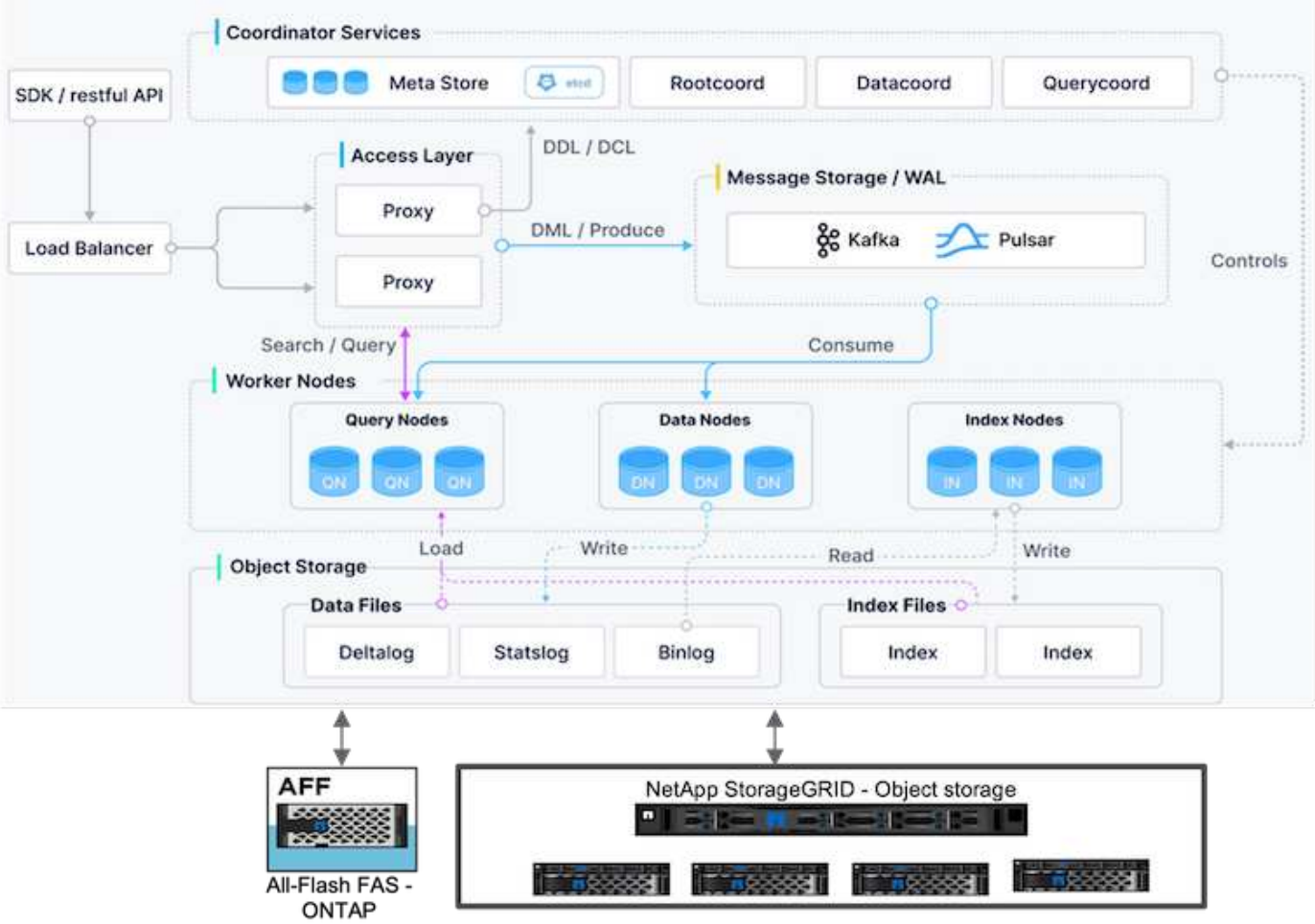
对于此解决方案、我们选择了Milvus进行解决方案验证。为了提高性能、我们使用了milvus和postgres (pgvedost.rs)。

为什么选择**Milvus**作为此解决方案?

- 开源：Milvus是一个开源向量数据库、鼓励社区驱动的开发和改进。
- AI集成：它利用嵌入的相似性搜索和AI应用程序来增强矢量数据库功能。
- 大容量处理：Milvus能够存储、索引和管理由深度神经网络(DNN)和机器学习(ML)模型生成的超过十亿个嵌入矢量。
- 用户友好型：易于使用、不到一分钟即可完成设置。Milvus还为不同的编程语言提供了SDK。

- 速度：它提供极快的检索速度、速度是某些替代产品的10倍。
- 可扩展性和可用性：Milvus具有高度可扩展性、可根据需要进行纵向和横向扩展。
- 功能丰富：支持不同的数据类型、属性筛选、用户定义功能(UDF)支持、可配置的一致性级别和行程时间、使其成为适用于各种应用程序的通用工具。

Milvus架构概述



本节介绍了Milvus架构中使用的更高级别的组件和服务。
*访问层—由一组无状态代理组成、充当系统的前端层和用户端点。
*协调员服务——它将任务分配给工作节点并充当系统的大脑。它有三种协调者类型：根协调者、数据协调者和查询协调者。
*工作节点:它遵循协调者服务的指示并执行用户触发的DML/DDL commands.it有三种类型的工作节点,如查询节点、数据节点和索引节点。
*存储：它负责数据的持久性。它由元数据存储、日志代理和对象存储组成。ONTAP和StorageGRID等NetApp存储为Milvus提供对象存储和基于文件的存储、用于存储客户数据和矢量数据库数据。

技术要求

技术要求

本文档中执行的大多数验证均使用了下面概述的硬件和软件配置、但性能除外。这些配置可作为指导原则、帮助您设置环境。但是、请注意、具体组件可能会因客户要求而异。

硬件要求

硬件	详细信息
NetApp AFF存储阵列HA对	* A800 * ONTAP 9.14.1. * 48个3.49 TB SSD -NVM *两个灵活组卷：元数据和数据。 *元数据NFS卷具有12个250 GB的永久性卷。 *数据是一个ONTAP NAS S3卷
6个Fujitsu PRIMERGY RX2540 M4	* 64个CPU *英特尔®至强®金牌6142 CPU @ 2.60GHz * 256 GM物理内存 * 1个100GbE网络端口
网络	100 GbE
StorageGRID	* 1个SG100、3个SGF6024 * 3 x 24 x 7.68 TB

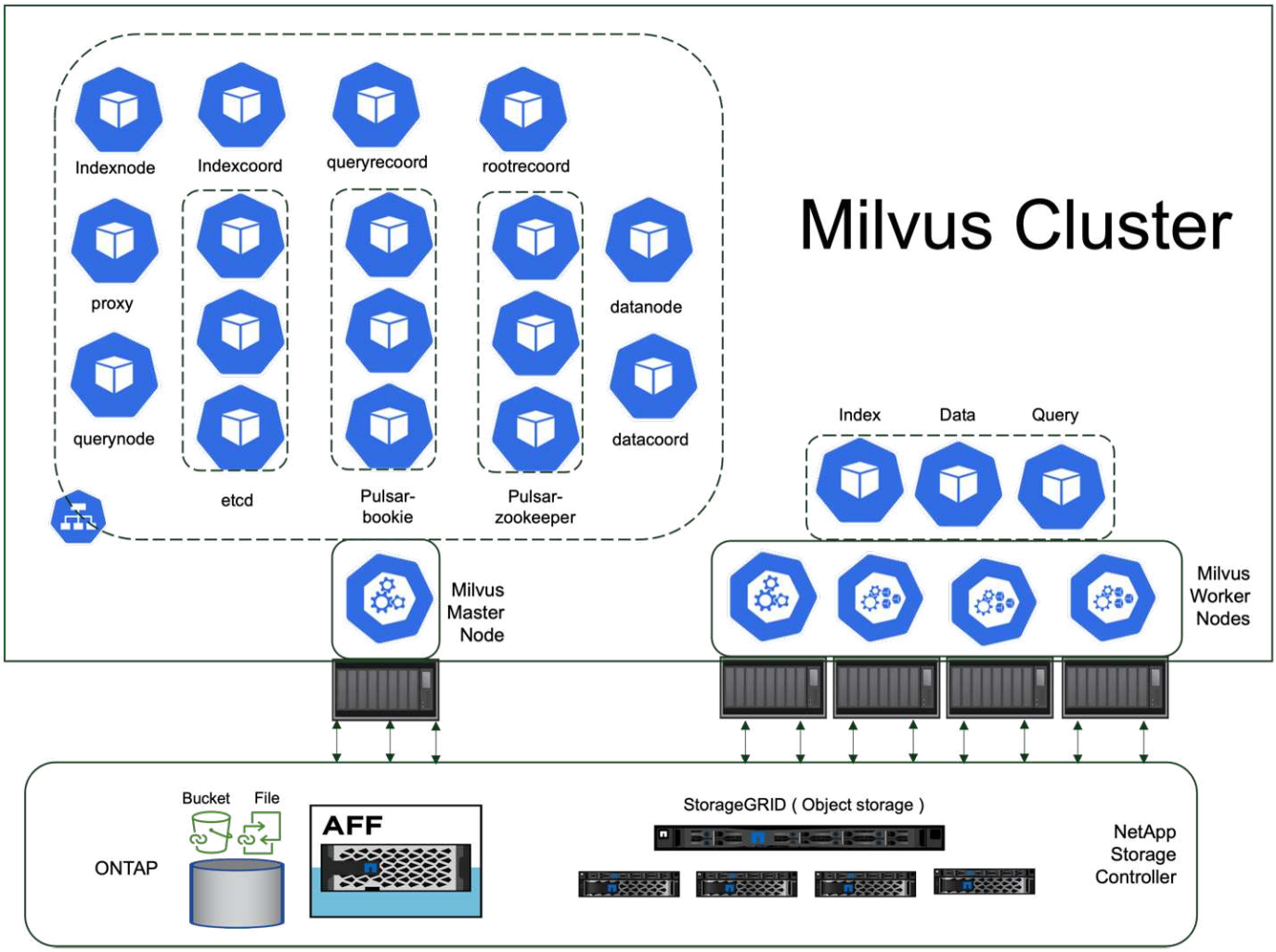
软件要求

软件	详细信息
Milvus集群	*图表- Milvus-4.1.11. *应用程序版本–2.3.4 *依赖包、如bookkeeper、zookeeper、pulsar、etic、proxy" querynnode、worker
Kubernetes	* 5节点K8s集群 * 1个主节点和4个工作节点 *版本–1.7.2
Python	*3.10.12.

部署操作步骤

部署操作步骤

在本部署部分中、我们将Milvus向量数据库与Kubbernetes一起用于实验室设置、如下所示。



NetApp存储可为集群提供存储、以保留客户数据和Milvus集群数据。

NetApp存储设置—ONTAP

- 存储系统初始化
- 创建Storage Virtual Machine (SVM)
- 分配逻辑网络接口
- NFS、S3配置和许可

对于NFS (网络文件系统)、请遵循以下步骤：

1. 为NFSv4创建FlexGroup卷。在此验证设置中、我们使用了48个SSD、1个SSD专用于控制器的根卷、47个SSD分布于NFSv4]]。验证FlexGroup卷的NFS导出策略是否具有Kubernetes (K8s)节点网络的读/写权限。如果未设置这些权限、请为K8s节点网络授予读/写(RW)权限。
2. 在所有K8s节点上、创建一个文件夹、并通过每个K8s节点上的逻辑接口(Logical Interface、LIF)将FlexGroup卷挂载到此文件夹中。

对于NAS S3 (网络连接存储简单存储服务)、请遵循以下步骤：

1. 为NFS创建FlexGroup卷。

2. 使用"vserver object-store-server creation"命令设置启用了HTTP且管理状态设置为"UP "的对象存储服务器。您可以选择启用HTTPS并设置自定义侦听器端口。
3. 使用vserver object-store-server user create -user <username>命令创建object-store-server用户。
4. 要获取访问密钥和机密密钥、可以运行以下命令：set diag；vserver object-store-server user show -user <username>。但是、接下来、这些密钥将在用户创建过程中提供、也可以使用REST API调用来检索。
5. 使用在步骤2中创建的用户建立一个对象存储服务器组并授予访问权限。在此示例中、我们提供了"FullAccess"。
6. 通过将NAS分段的类型设置为"NAS "并提供NFS3卷的路径来创建NAS分段。也可以使用S3存储分段来实现此目的。

NetApp存储设置—StorageGRID

1. 安装StorageGRID软件。
2. 创建租户和存储分段。
3. 创建具有所需权限的用户。

请在中查看更多详细信息 https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

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

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- https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html。

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}}]}
```

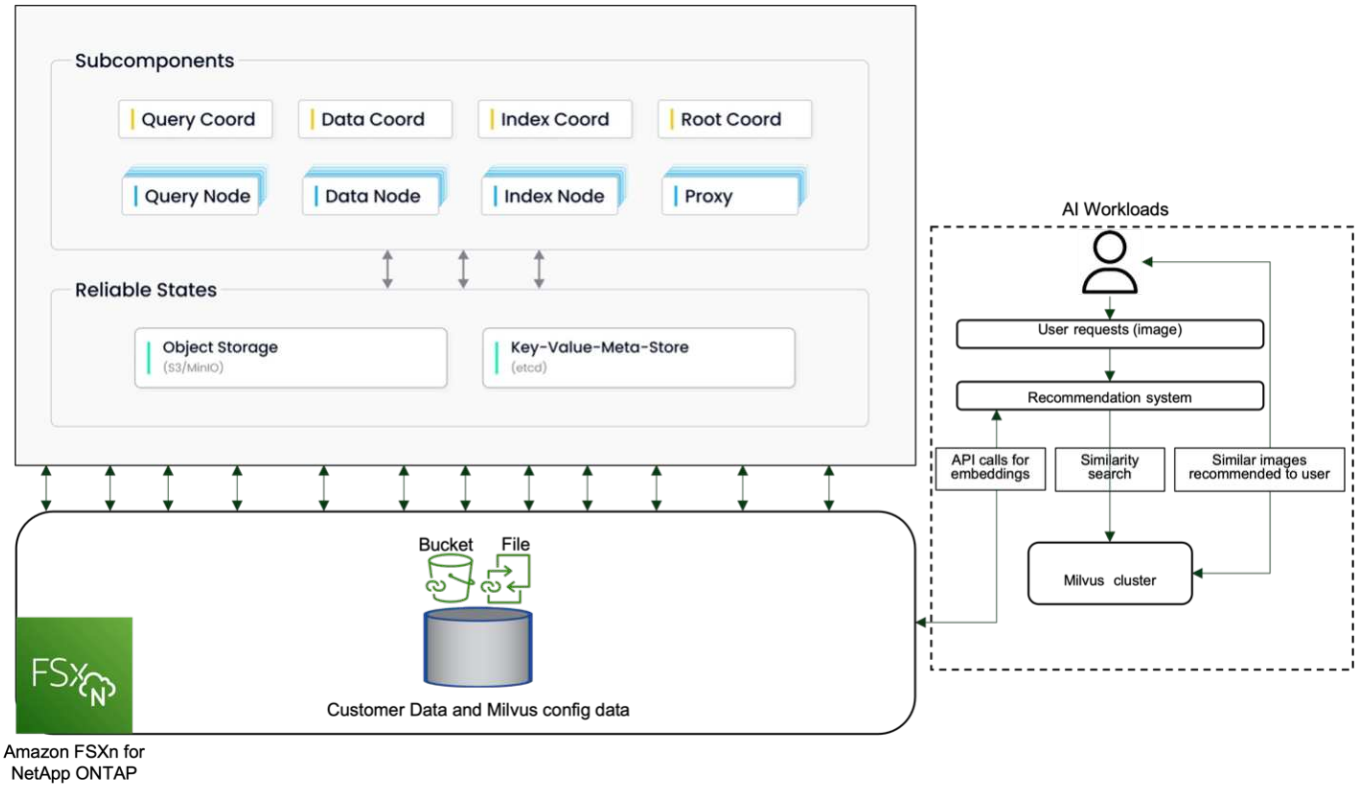
根据上述验证、通过使用NetApp存储控制器在Kubbernetes上部署Milvus集群、Kubbernetes与向量数据库的集成为客户提供了一个强大、可扩展且高效的解决方案、用于管理大规模数据操作。这种设置使客户能够处理高维度数据并快速高效地执行复杂查询、使其成为大数据应用程序和AI工作负载的理想解决方案。对各种集群组件使用永久性卷(PV)、以及从NetApp ONTAP创建单个NFS卷、可确保最佳的资源利用率和数据管理。验证持久卷声明(PVC)和Pod状态以及测试数据写入和读取的过程、为客户提供了可靠且一致的数据操作保证。将ONTAP或StorageGRID对象存储用于客户数据可进一步增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

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/448789845791611912/448789845791611913/4487898457
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#
```

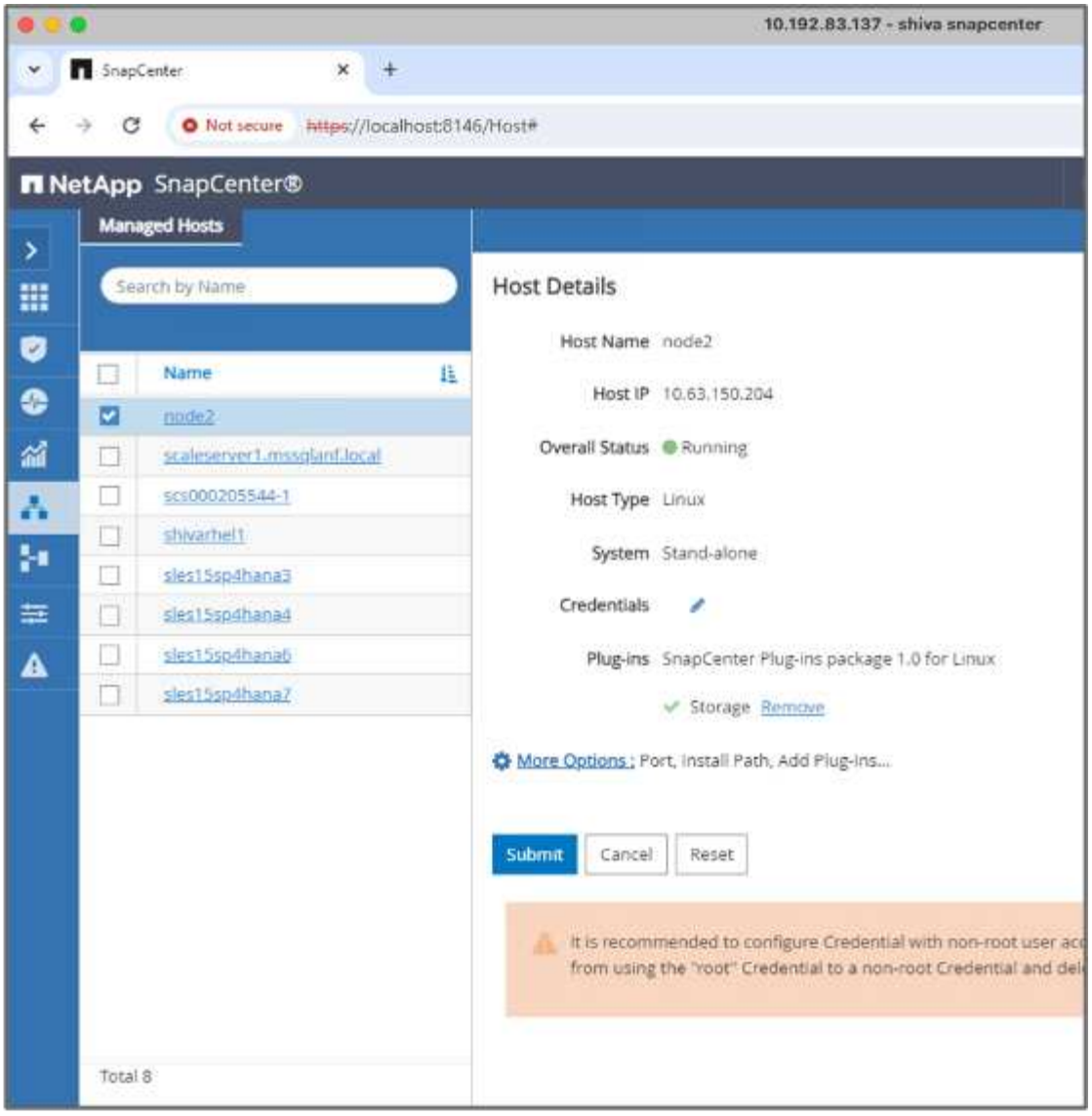
本节有效地演示了客户如何在Docker容器中部署和运行独立的Milvus设置、并利用Amazon的NetApp FSxN来存储NetApp ONTAP数据。通过这种设置、客户可以在可扩展且高效的Docker容器环境中利用向量数据库的强大功能来处理高维数据和执行复杂查询。通过为NetApp ONTAP实例创建Amazon FSxN并匹配EC2实例、客户可以确保最佳资源利用率和数据管理。成功验证了向量数据库中FSxN的数据写入和读取操作、为客户提供了可靠且一致的数据操作保障。此外、通过S3协议列出(读取) AI工作负载中的数据、还增强了数据可访问性。因此、这一全面的流程为客户提供了一个强大而高效的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

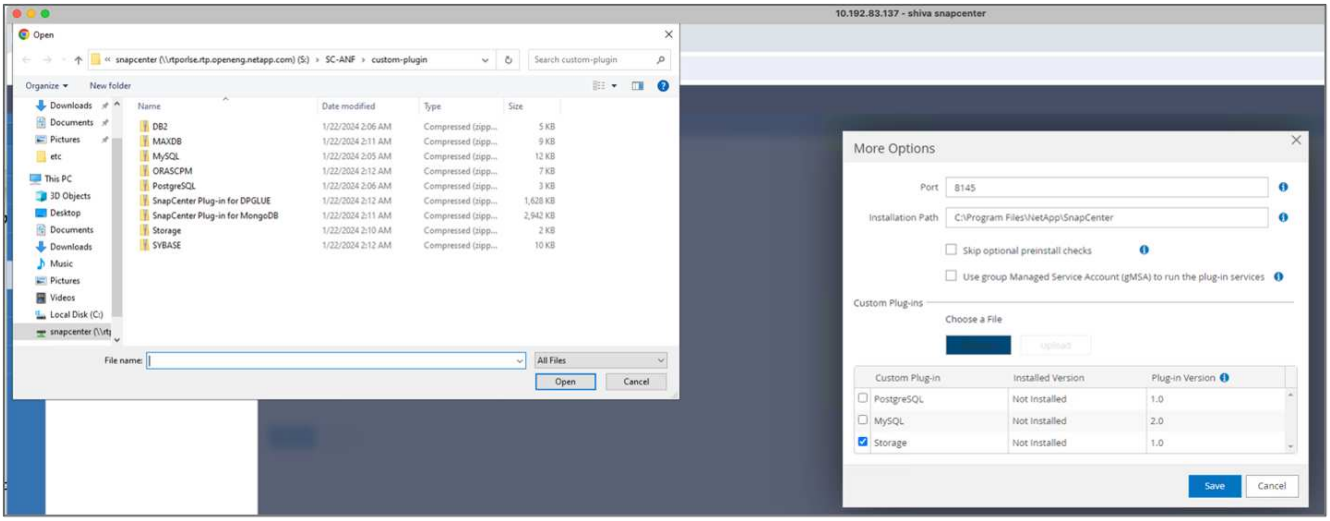
使用**NetApp SnapCenter**保护向量数据库。

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。

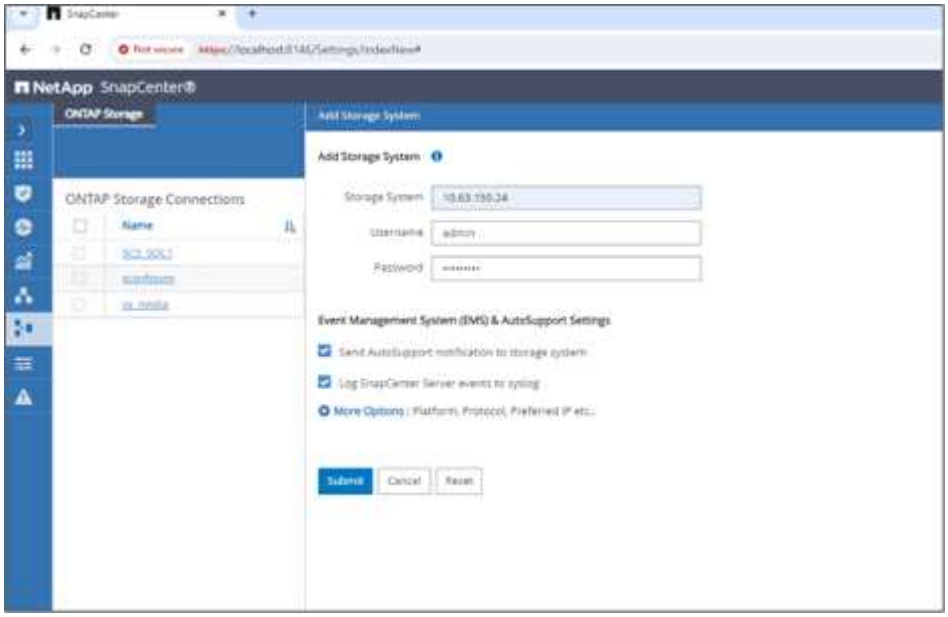


2. 安装和配置存储插件。从添加的主机中、选择"More Options (更多选项)"。导航到并从中选择已下载的存储插件 "NetApp 自动化商店"。安装插件并保存配置。

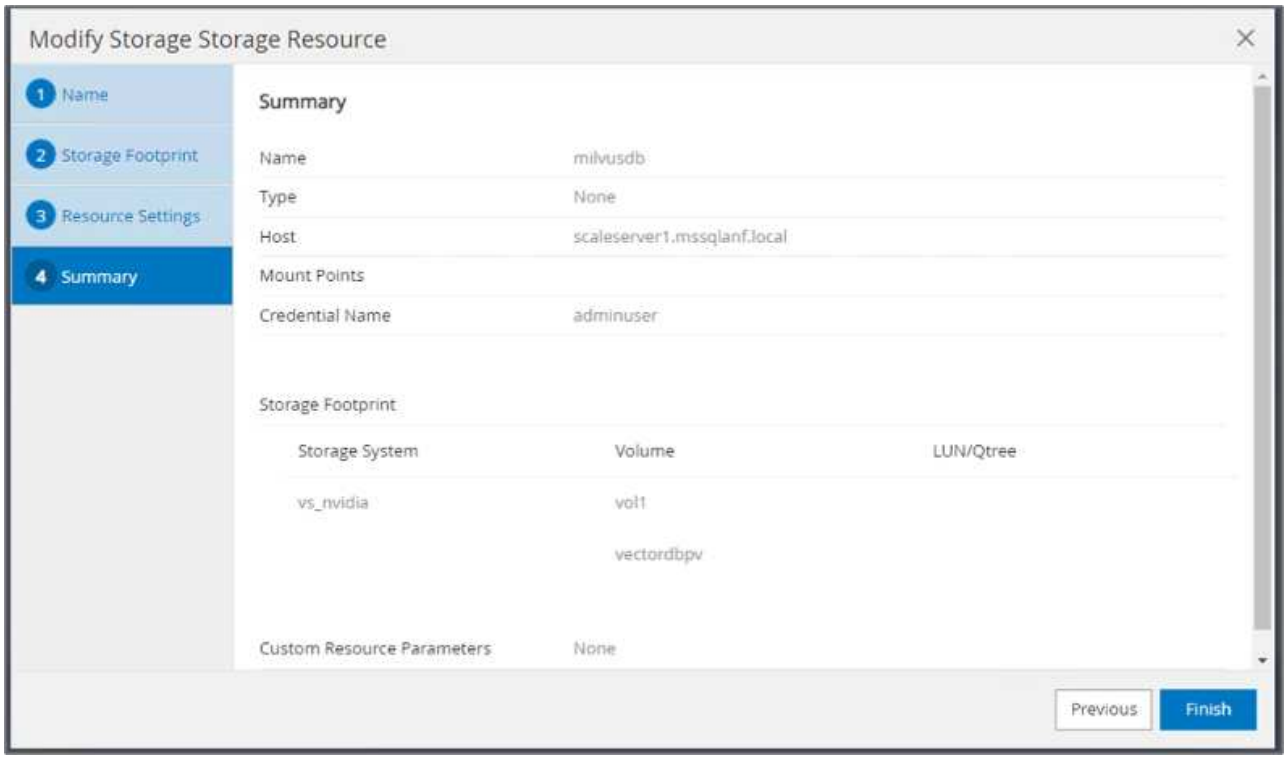


3. 设置存储系统和卷：在"存储系统"下添加存储系统、然后选择SVM (Storage Virtual Machine)。在此示例

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。



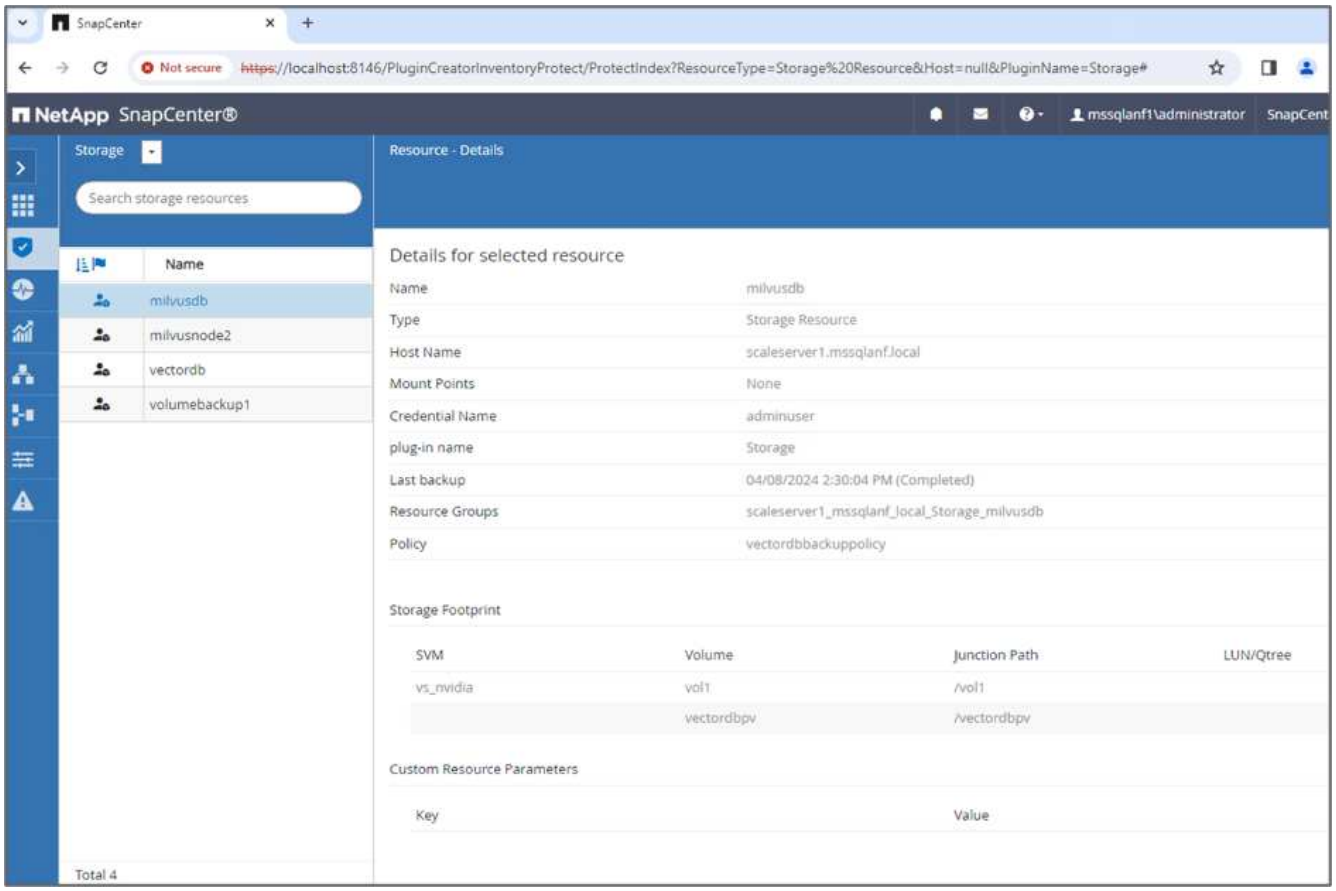
5. 使用Python脚本将数据插入S3 NAS分段。在本示例中、我们修改了Milvus提供的备份脚本、即"prepy_data_NetApp.py"、并执行了"ync"命令从操作系统中刷新数据。

```
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

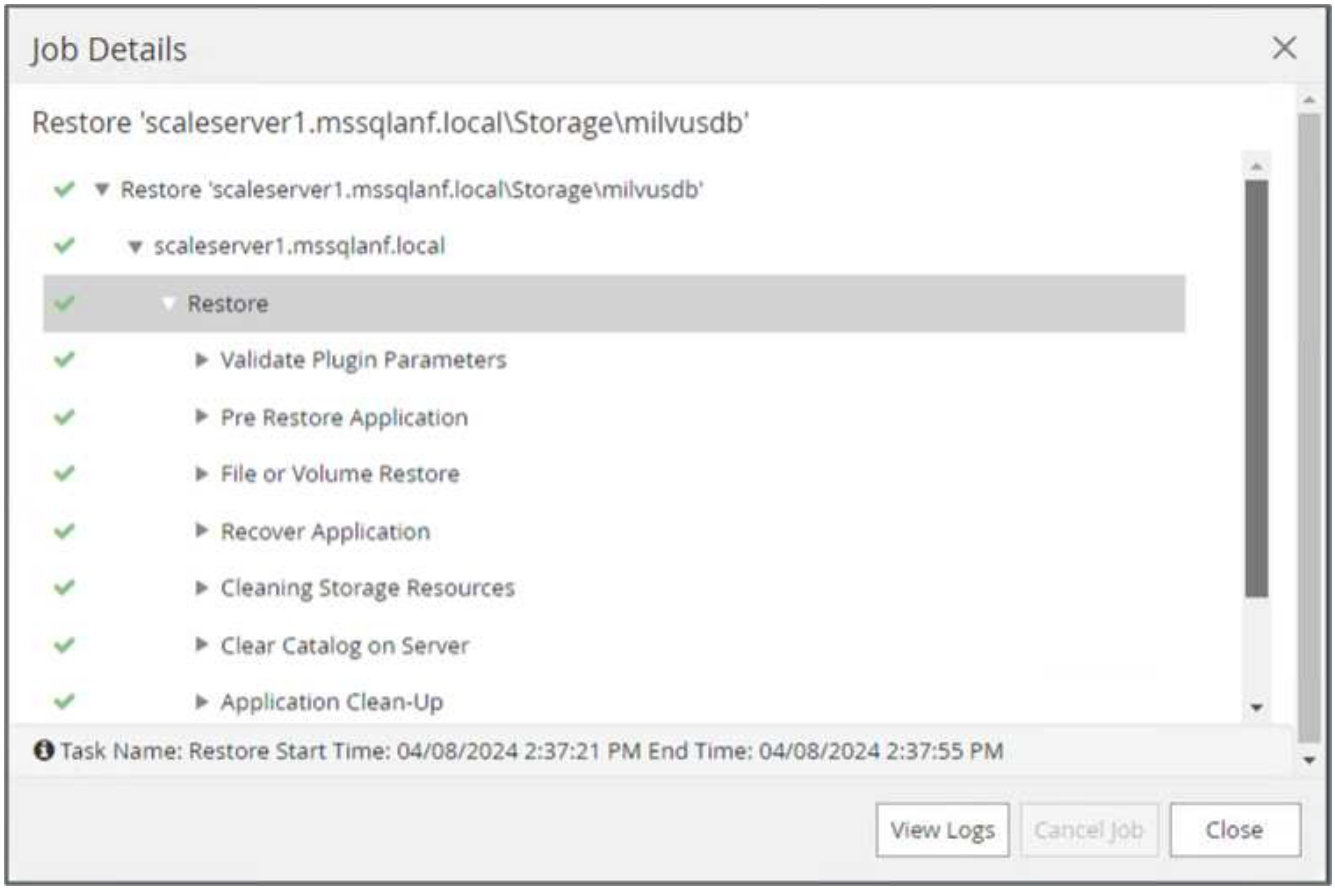


8. 为了测试备份功能、我们会在备份过程后添加一个新表、或者从NFS (S3 NAS存储分段)中删除一些数据。

   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:~#
```

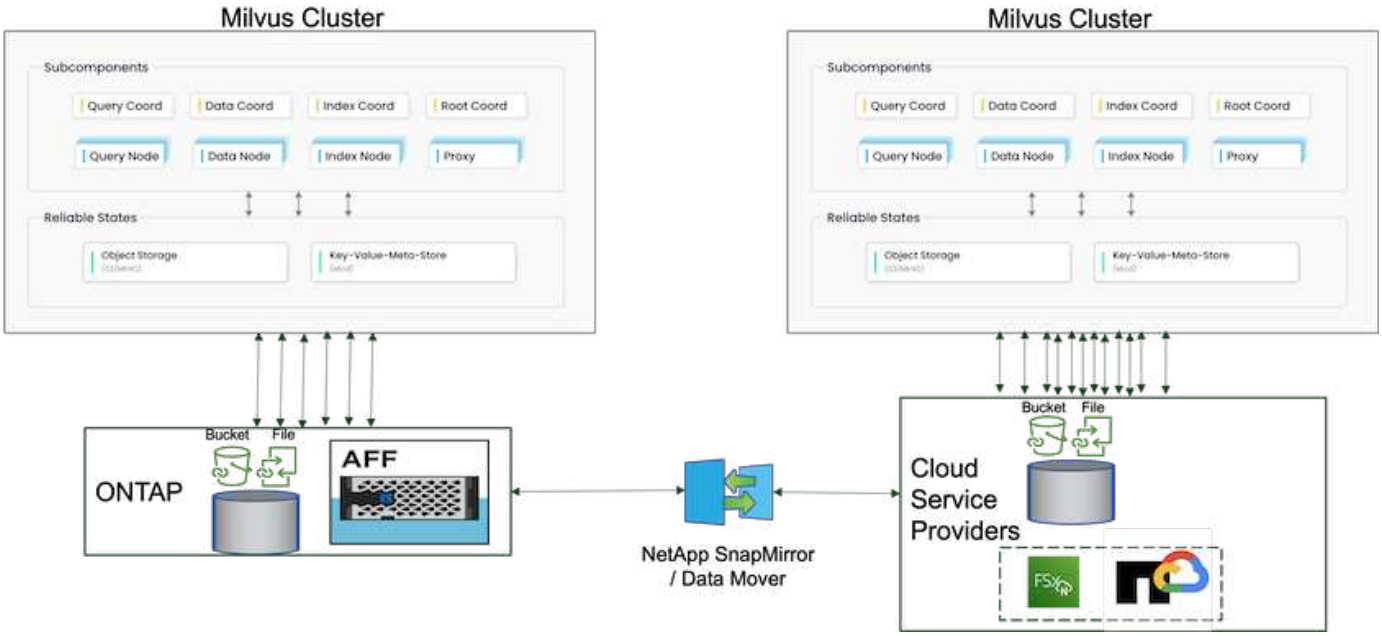
总之、使用NetApp的SnapCenter保护矢量数据库数据以及驻留在ONTAP中的Milvus数据为客户带来了巨大的优势、尤其是在数据完整性至关重要的行业、例如电影制作。SnapCenter能够创建一致的备份并执行完整数据恢复、从而确保关键数据(例如嵌入式视频和音频文件)不会因硬盘故障或其他问题而丢失。这不仅可以防止运营中断、还可以防止出现重大财务损失。

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

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

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。

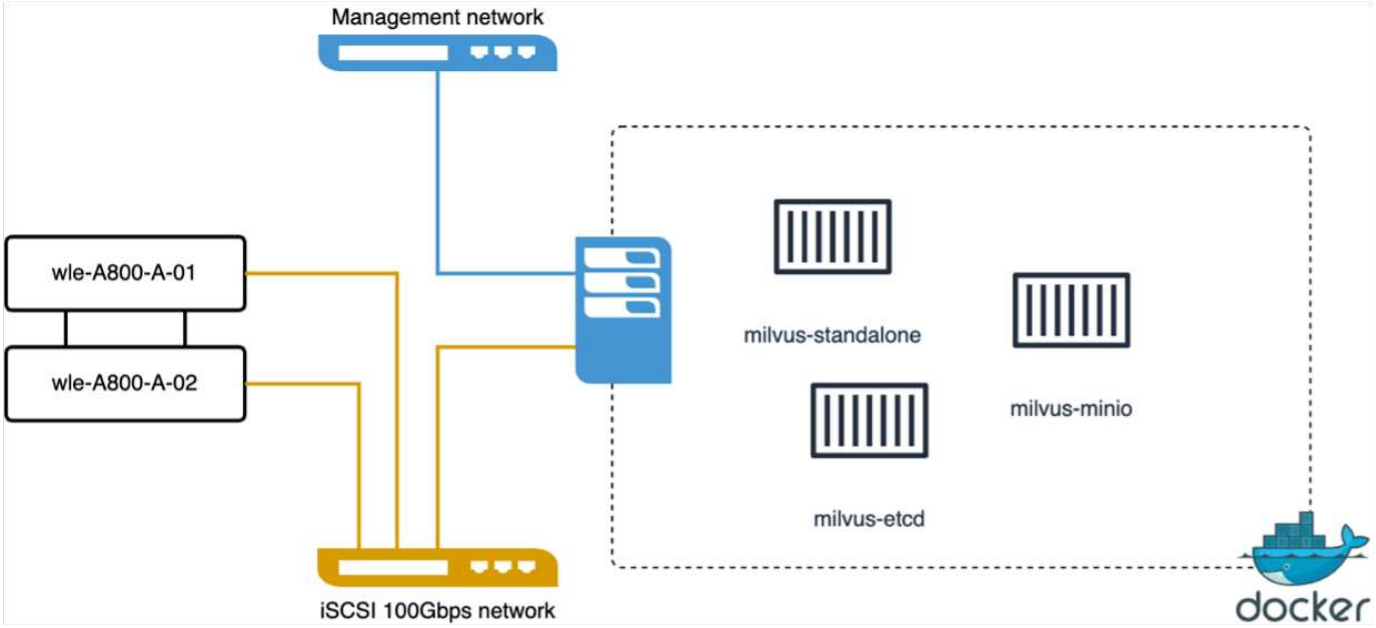
在本节中、我们将深入探讨Milvus和pgvector.rs等向量数据库的性能验证、重点介绍其存储性能特征、例如I/O配置文件和NetApp存储控制器在LLM生命周期内支持RAG和推导工作负载时的行为。我们将评估并确定将这些数据库与ONTAP存储解决方案结合使用时的任何性能差异。我们的分析将基于关键绩效指标、例如每秒处理的查询数量(QPS)。

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	

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。

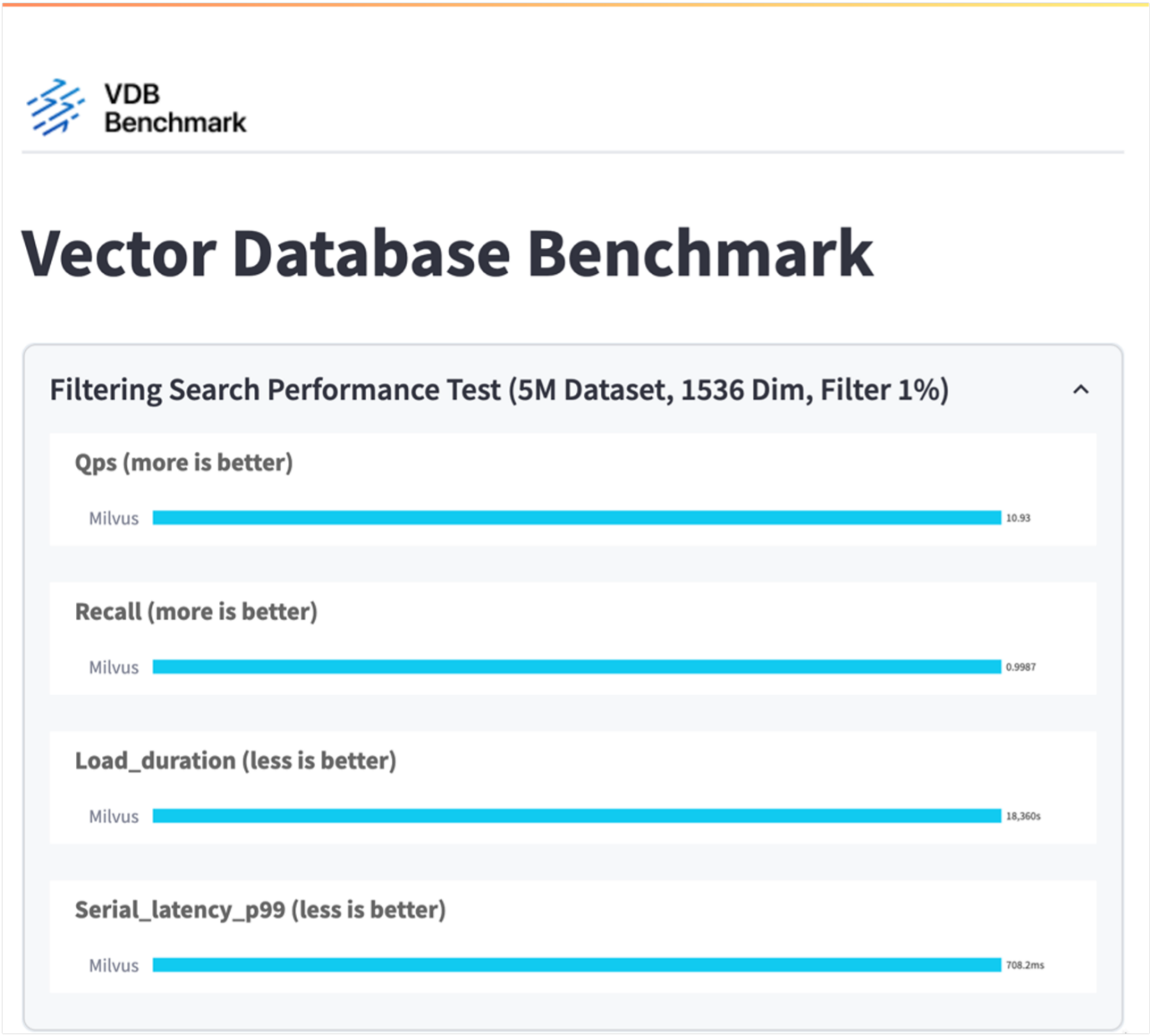
。 为大约100 GB的数据集执行数据导入、优化和创建索引大约需要5小时。在这段时间内的大部分时间内、配备20个核心(启用超线程时相当于40个vCPU)的Milvus服务器以其100%的最大CPU容量运行。我们发现、DiskANN对于超过系统内存大小的大型数据集尤为重要。
。 在查询阶段、我们观察到每秒查询数(Queries Per Second、QPS)比率为10.93、而调用率为0.9987。查询延迟的第99个百位点在708.2毫秒处测量。

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。

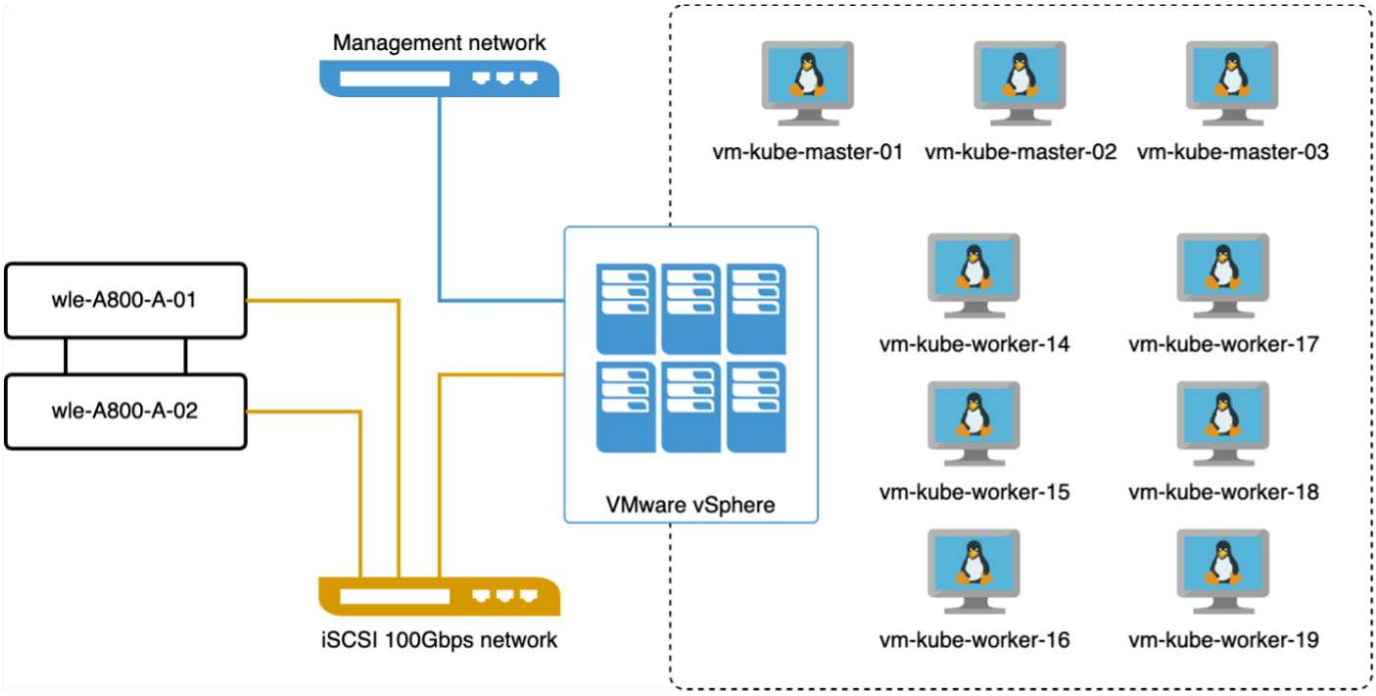


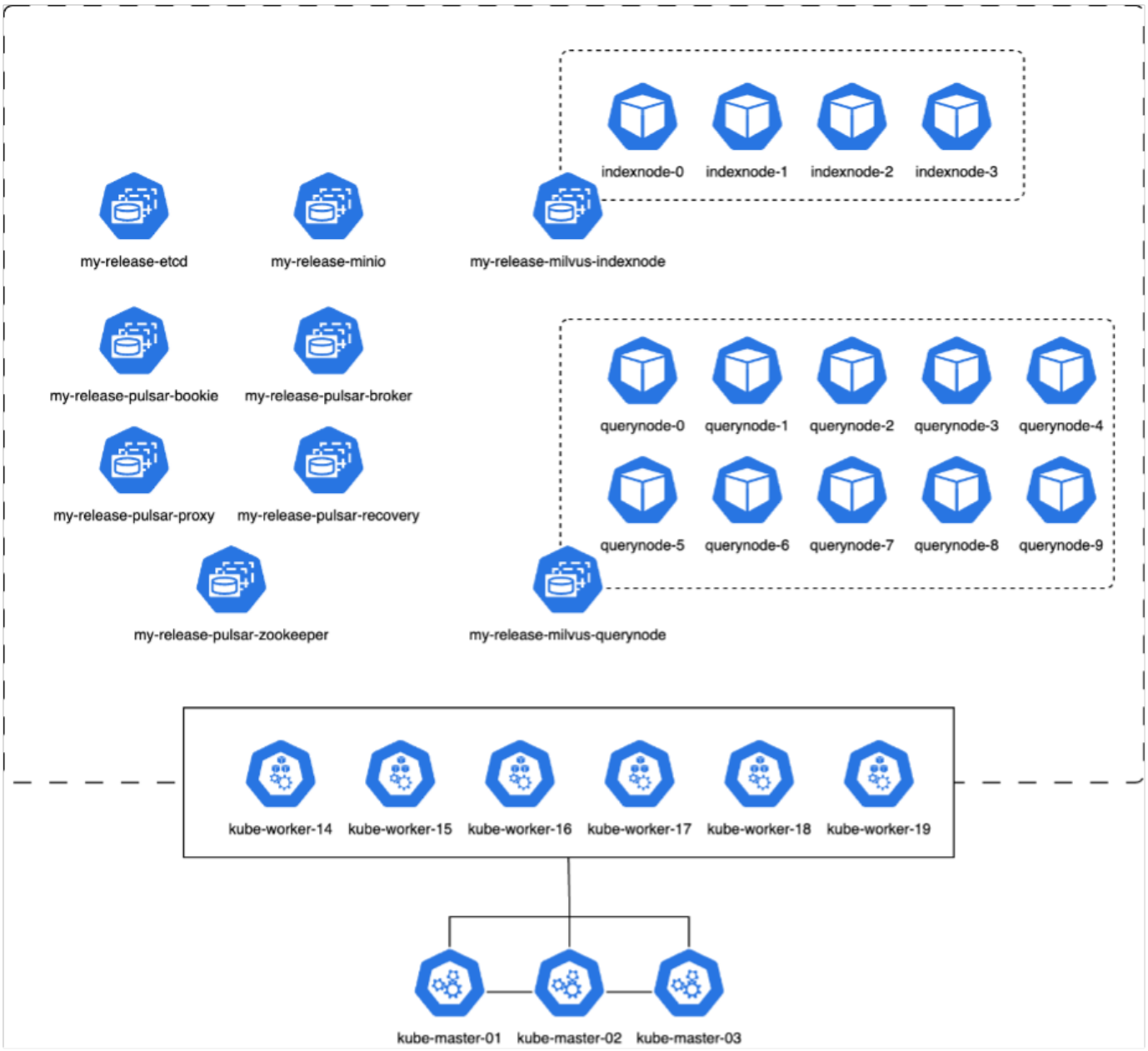
从独立Milvus实例的性能验证来看、很明显、当前设置不足以支持一个包含500万向量且维度为1536的数据集。我们已确定存储具有足够的资源、不会在系统中构成瓶颈。

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)。

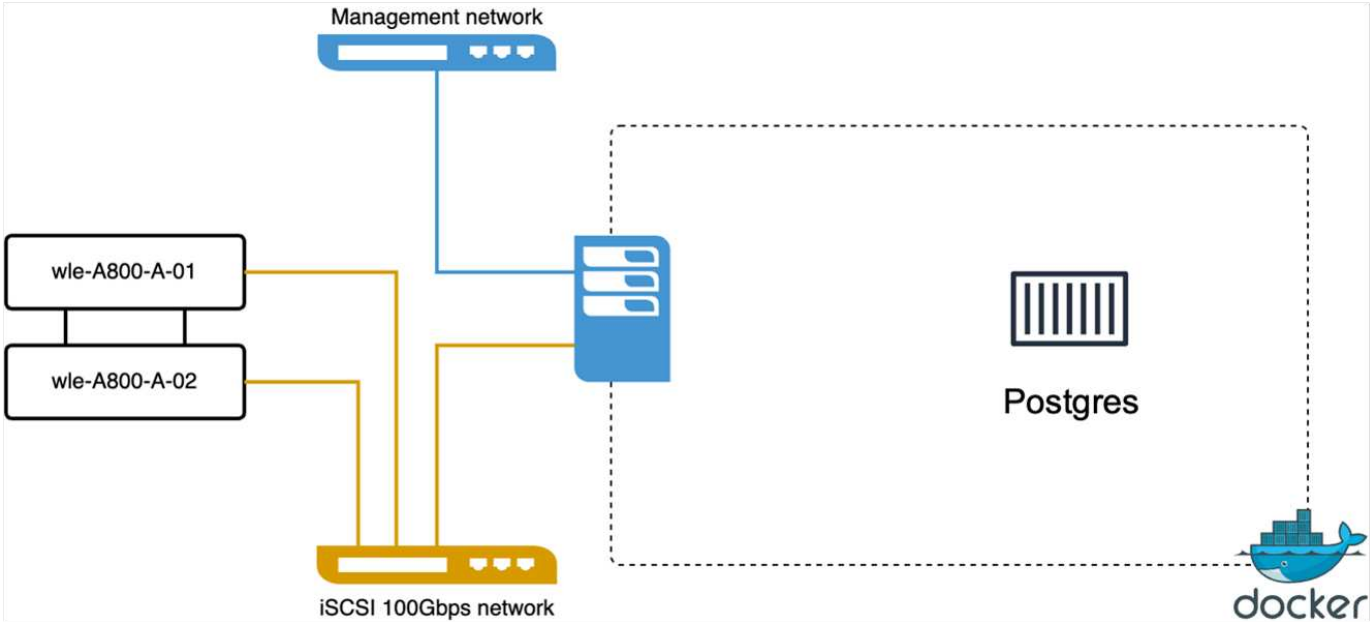
下一节介绍了主要的存储性能指标。

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：

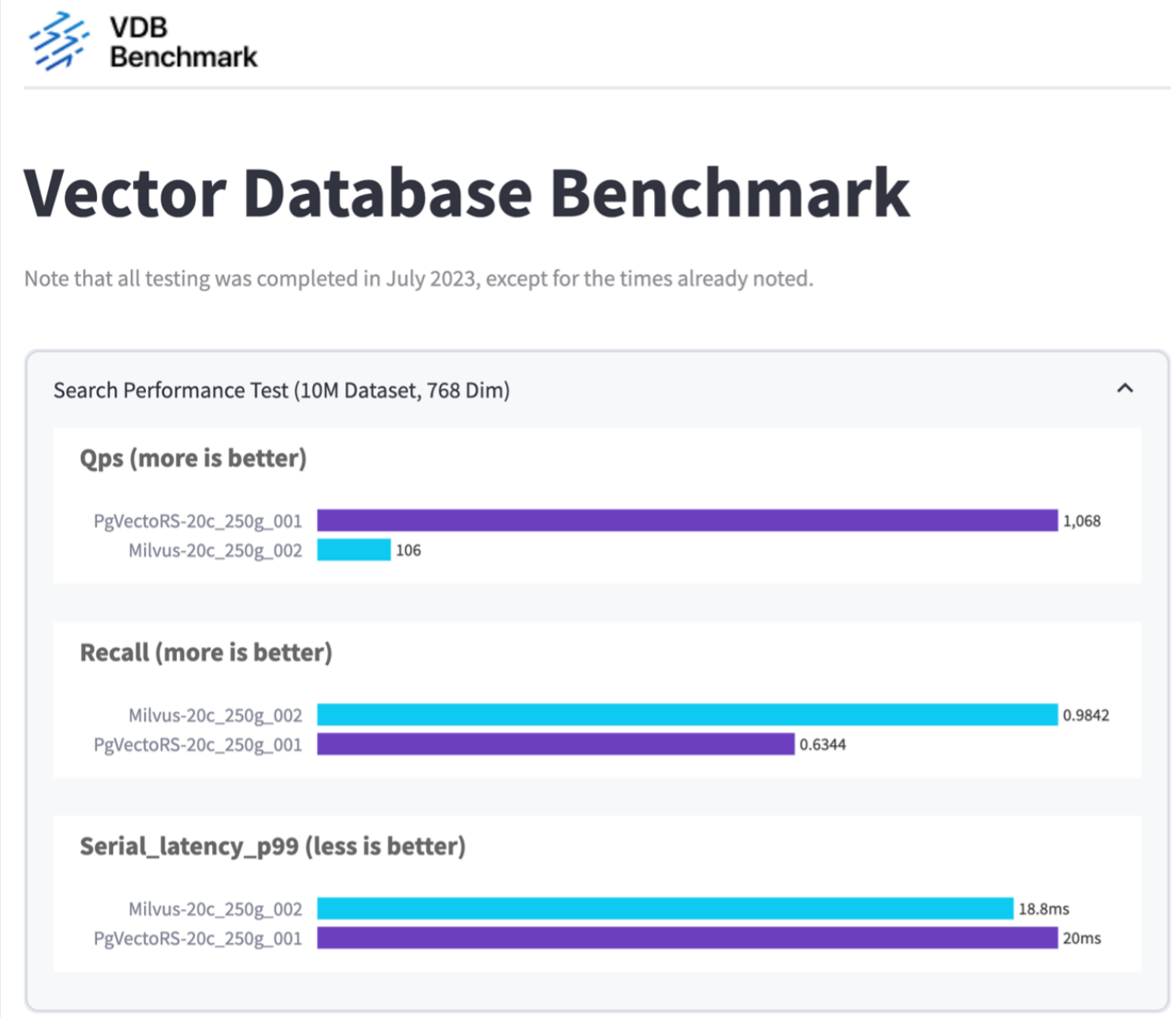


在本节中、我们将分享对PostgreSQL数据库(尤其是使用pg向 量.rs)进行测试后观察到的结果。
*我们选择HNSW作为这些测试的索引类型、因为在测试时、DiskANN不适用于pg向 量.rs。
*在数据载入阶段、我们加载了cothere数据集、该数据集由1、000万个向量组成、维数为768。此过程大约需要4.5小时。
*在查询阶段、我们观察到每秒查询数(Queries Per Second、QPS)比率为1、068、而调用率为0.6344。查询延迟的第99个百位点在20毫秒处测量。在运行时的大部分时间内、客户端CPU都以100%的容量运行。

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["错误：缺少图形映像"]
```

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。

带有使用PostgreSQL的Instaclosts的向量数据库：pgvector

带有使用PostgreSQL的Instaclosts的向量数据库：pgvector

在本节中、我们将详细介绍instaclosts产品如何在向量功能上与PostgreSQL集成。我们有一个示例，“如何使用PGVector和PostgreSQL®提高LLM的准确性和性能：嵌入简介和PGVector的作用”。请检查 "博客" 以获取更多信息。

向量数据库用例

向量数据库用例

在本节中、我们将讨论两个用例、例如使用大型语言模型的"恢复增强型生成"和NetApp IT聊天机器人。

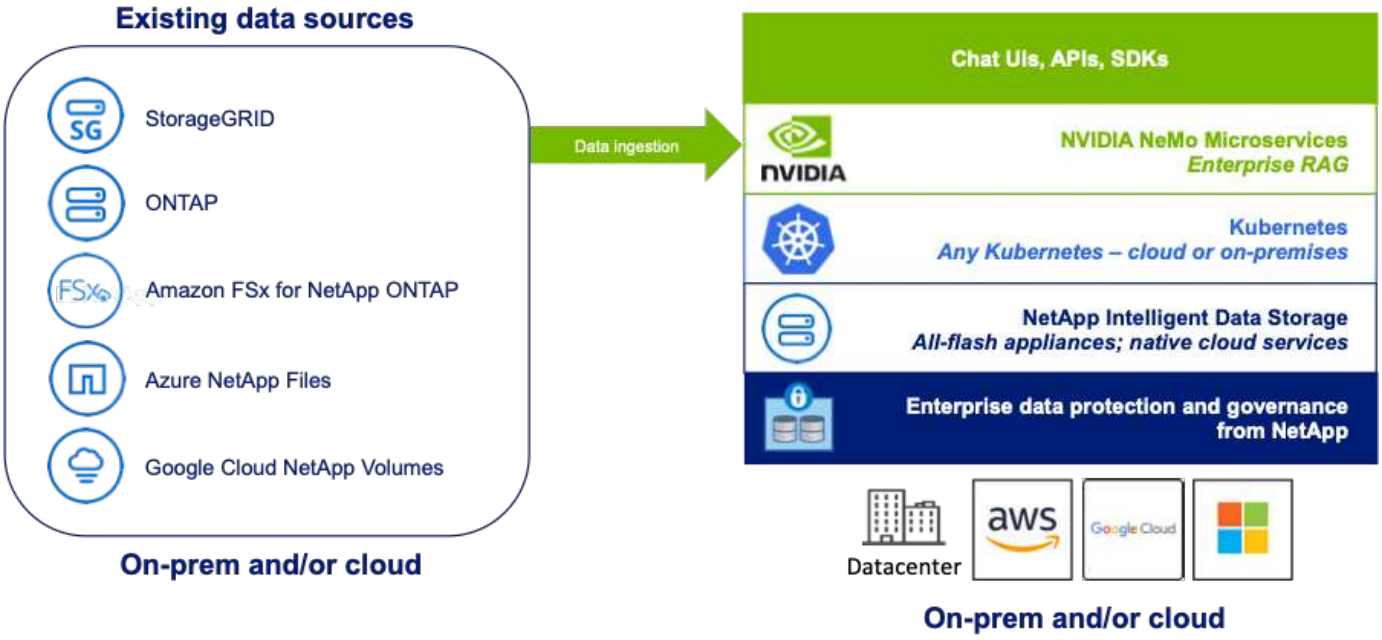
使用大型语言模型**(LLM)**的检索增强型生成**(RAG)**

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

NVIDIA Enterprise RAG LLM Operator是在企业中实施RAG的有用工具。此操作员可用于部署完整的RAG管道。可以自定义RAG管道、以使用Milvus或pgvector作为存储库内包的矢量数据库。有关详细信息、请参见文档。

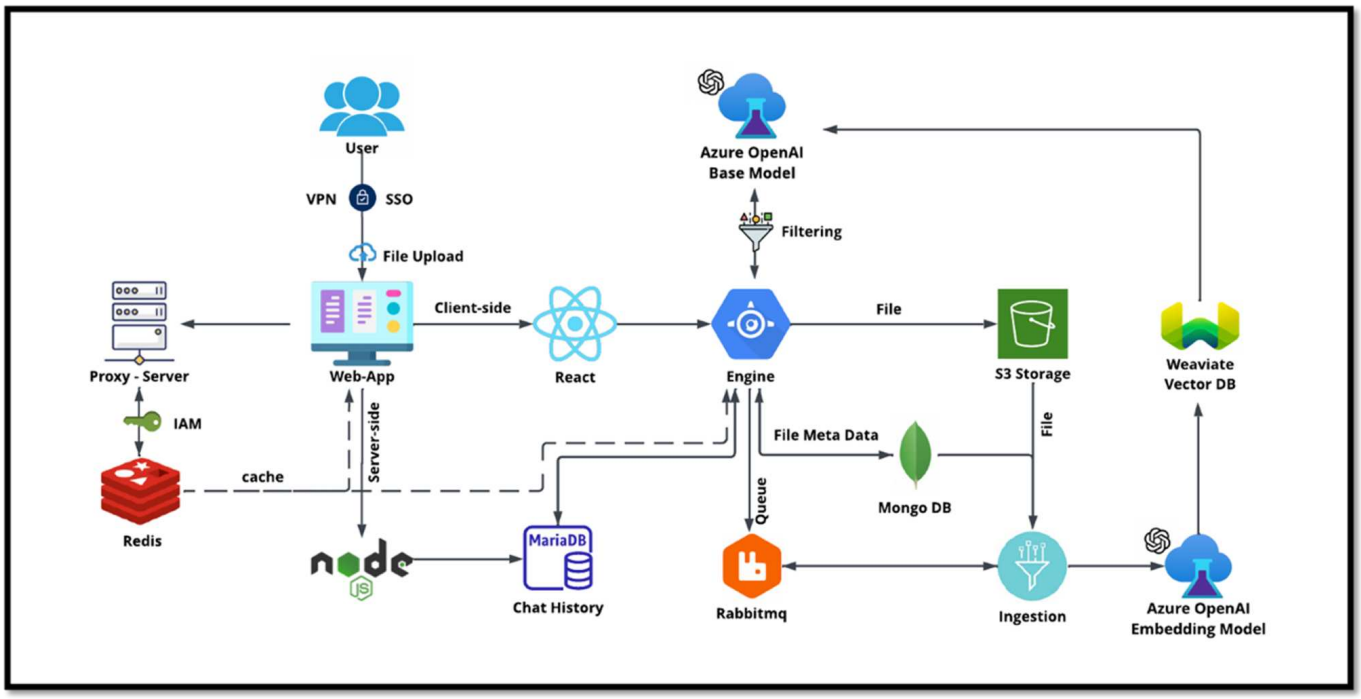
```
NetApp has validated an enterprise RAG architecture powered by the NVIDIA
Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog
post for more information and to see a demo. Figure 1 provides an overview
of this architecture.
```

图1)由NVIDIA Nemo Microservices和NetApp提供支持的企业级RAG



NetApp IT聊天机器人用例

NetApp的聊天机器人是向量数据库的另一个实时用例。在这种情况下、NetApp Private OpenAI沙盒可提供一个有效、安全且高效的平台、用于管理NetApp内部用户的查询。通过整合严格的安全协议、高效的数据管理系统和成熟的AI处理功能、它可以根据用户在组织中的角色和职责、通过SSO身份验证保证为用户提供高质量、精确的响应。此架构突出了融合高级技术以创建以用户为中心的智能系统的潜力。



此使用情形可分为四个主要部分。

用户身份验证和验证：

- 用户查询首先会通过NetApp单点登录(SSO)过程来确认用户的身份。
- 成功进行身份验证后、系统会检查VPN连接以确保安全的数据传输。

数据传输和处理：

- VPN通过验证后、数据将通过NetAIChat或NetAICREAT Web应用程序发送到MariaDB。MariaDB是一个快速高效的数据库系统、用于管理和存储用户数据。
- 然后、MariaDB会将此信息发送到NetApp Azure实例、该实例会将用户数据连接到AI处理单元。

与**OpenAI**和内容筛选交互：

- Azure实例将用户的问题发送到内容筛选系统。此系统会清理查询并为处理该查询做好准备。
- 然后、清理的输入将发送到Azure OpenAI基础模型、该模型将根据输入生成响应。

响应生成和审核：

- 首先检查基本模型的响应、以确保其准确且符合内容标准。
- 通过检查后、响应将发送回用户。此过程可确保用户收到清晰、准确且适合其查询的问题解答。

结论

结论

总之、本文档全面概述了如何在NetApp存储解决方案上部署和管理矢量数据库、例如Milvus和vgvector。我们讨

论了利用NetApp ONTAP和StorageGRID对象存储的基础架构准则、并通过文件和对象存储在AWS FSx for NetApp ONTAP中验证了Milvus数据库。

我们探讨了NetApp的文件-对象双重性、展示了它不仅对矢量数据库中的数据有用、而且对其他应用程序有用。我们还重点介绍了NetApp的企业管理产品SnapCenter如何为矢量数据库数据提供备份、还原和克隆功能、从而确保数据完整性和可用性。

本文档还深入探讨了NetApp的混合云解决方案如何跨内部环境和云环境提供数据复制和保护、从而提供无缝、安全的数据管理体验。我们深入分析了NetApp ONTAP上的Milvus和pgvector等向量数据库的性能验证、并提供了有关其效率和可扩展性的宝贵信息。

最后、我们讨论了两种生成性AI用例：RAG with LLM和NetApp的内部ChatAI。这些实际示例强调了本文档所述概念和实践的实际应用和优势。总之、对于希望利用NetApp强大的存储解决方案来管理矢量数据库的任何人、本文档都是一个全面的指南。

致谢

作者衷心感谢以下贡献者、以及为使本白皮书对NetApp客户和NetApp Fields更有价值而提供反馈和意见的其他人员。

1. NetApp ONTAP AI和分析技术营销工程师Sathish Thyagarajan
2. NetApp 技术营销工程师 Mike Oglesby
3. NetApp高级主管，阿希·马哈詹
4. NetApp工作负载性能工程经理Joe Scott
5. NetApp FSx产品管理高级总监Puneet
6. NetApp FSx产品团队高级产品经理YuVal Kalderon

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请查看以下文档和 / 或网站：

- Milvus文档- https://milvus.io/docs/overview.md
- Milvus独立文档- https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- NetApp 产品文档
  https://www.netapp.com/support-and-training/documentation/
- instaclinst. "instal佐证 文件"

版本历史记录

version	Date	文档版本历史记录
版本 1.0	2024年4月	初始版本。

附录A：values.yaml

附录A：values.yaml

```
root@node2:~# cat  values.yaml
## Enable or disable Milvus Cluster mode
cluster:
  enabled: true

image:
  all:
    repository: milvusdb/milvus
    tag: v2.3.4
    pullPolicy: IfNotPresent
    ## Optionally specify an array of imagePullSecrets.
    ## Secrets must be manually created in the namespace.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/pull-
image-private-registry/
    ##
    # pullSecrets:
    #   - myRegistryKeySecretName
  tools:
    repository: milvusdb/milvus-config-tool
    tag: v0.1.2
    pullPolicy: IfNotPresent

# Global node selector
# If set, this will apply to all milvus components
# Individual components can be set to a different node selector
nodeSelector: {}

# Global tolerations
# If set, this will apply to all milvus components
# Individual components can be set to a different tolerations
tolerations: []

# Global affinity
# If set, this will apply to all milvus components
# Individual components can be set to a different affinity
affinity: {}

# Global labels and annotations
# If set, this will apply to all milvus components
labels: {}
annotations: {}

# Extra configs for milvus.yaml
# If set, this config will merge into milvus.yaml
```

```
# Please follow the config structure in the milvus.yaml
# at https://github.com/milvus-io/milvus/blob/master/configs/milvus.yaml
# Note: this config will be the top priority which will override the
config
# in the image and helm chart.
extraConfigFiles:
  user.yaml: |+
    #    For example enable rest http for milvus proxy
    #    proxy:
    #      http:
    #        enabled: true
    ##  Enable tlsMode and set the tls cert and key
    #  tls:
    #    serverPemPath: /etc/milvus/certs/tls.crt
    #    serverKeyPath: /etc/milvus/certs/tls.key
    #   common:
    #     security:
    #       tlsMode: 1

## Expose the Milvus service to be accessed from outside the cluster
(LoadBalancer service).
## or access it from within the cluster (ClusterIP service). Set the
service type and the port to serve it.
## ref: http://kubernetes.io/docs/user-guide/services/
##
service:
  type: ClusterIP
  port: 19530
  portName: milvus
  nodePort: ""
  annotations: {}
  labels: {}

  ## List of IP addresses at which the Milvus service is available
  ## Ref: https://kubernetes.io/docs/user-guide/services/#external-ips
  ##
  externalIPs: []
  #   - externalIp1

  # LoadBalancerSourcesRange is a list of allowed CIDR values, which are
combined with ServicePort to
  # set allowed inbound rules on the security group assigned to the master
load balancer
  loadBalancerSourceRanges:
  - 0.0.0.0/0
  # Optionally assign a known public LB IP
```

```
  # loadBalancerIP: 1.2.3.4

ingress:
  enabled: false
  annotations:
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: GRPC
    nginx.ingress.kubernetes.io/listen-ports-ssl: '[19530]'
    nginx.ingress.kubernetes.io/proxy-body-size: 4m
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  labels: {}
  rules:
    - host: "milvus-example.local"
      path: "/"
      pathType: "Prefix"
    # - host: "milvus-example2.local"
    #   path: "/otherpath"
    #   pathType: "Prefix"
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - milvus-example.local

serviceAccount:
  create: false
  name:
  annotations:
  labels:

metrics:
  enabled: true

  serviceMonitor:
    # Set this to `true` to create ServiceMonitor for Prometheus operator
    enabled: false
    interval: "30s"
    scrapeTimeout: "10s"
    # Additional labels that can be used so ServiceMonitor will be
discovered by Prometheus
    additionalLabels: {}

livenessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 30
```

```
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

readinessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

log:
  level: "info"
  file:
    maxSize: 300    # MB
    maxAge: 10    # day
    maxBackups: 20
  format: "text"    # text/json

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Logs Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ## ReadWriteMany access mode required for milvus cluster.
      ##
      storageClass: default
      accessModes: ReadWriteMany
      size: 10Gi
      subPath: ""

## Heaptrack traces all memory allocations and annotates these events with
stack traces.
```

```
## See more: https://github.com/KDE/heaptrack
## Enable heaptrack in production is not recommended.
heaptrack:
  image:
    repository: milvusdb/heaptrack
    tag: v0.1.0
    pullPolicy: IfNotPresent

standalone:
  replicas: 1  # Run standalone mode with replication disabled
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  disk:
    enabled: true
    size:
      enabled: false  # Enable local storage size limit
  profiling:
    enabled: false  # Enable live profiling

  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
    mountPath: "/var/lib/milvus"
    ## If true, alertmanager will create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: true
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
```

```
      ##   set, choosing the default provisioner.
      ##
      storageClass:
      accessModes: ReadWriteOnce
      size: 50Gi
      subPath: ""

proxy:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  http:
    enabled: true  # whether to enable http rest server
    debugMode:
      enabled: false
  # Mount a TLS secret into proxy pod
  tls:
    enabled: false
## when enabling proxy.tls, all items below should be uncommented and the
key and crt values should be populated.
#    enabled: true
#    secretName: milvus-tls
## expecting base64 encoded values here: i.e. $(cat tls.crt | base64 -w 0)
and $(cat tls.key | base64 -w 0)
#    key: LS0tLS1CRUdJTiBQU--REDUCT
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

rootCoordinator:
  enabled: true
```

```
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Root Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for root coordinator

  service:
    port: 53100
    annotations: {}
    labels: {}
    clusterIP: ""

queryCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Query Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for query coordinator

  service:
    port: 19531
    annotations: {}
    labels: {}
    clusterIP: ""
```

```
queryNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  disk:
    enabled: true  # Enable querynode load disk index, and search on disk
index
    size:
      enabled: false  # Enable local storage size limit
  profiling:
    enabled: false  # Enable live profiling

indexCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1   # Run Index Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for index coordinator

  service:
    port: 31000
    annotations: {}
    labels: {}
    clusterIP: ""
```

```
indexNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  # Set local storage size in resources
  # limits:
  #    ephemeral-storage: 100Gi
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  disk:
    enabled: true  # Enable index node build disk vector index
    size:
      enabled: false  # Enable local storage size limit

dataCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Data Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for data coordinator

  service:
    port: 13333
    annotations: {}
    labels: {}
```

```
    clusterIP: ""

dataNode:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling

## mixCoordinator contains all coord
## If you want to use mixcoord, enable this and disable all of other
coords
mixCoordinator:
  enabled: false
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Mixture Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for Mixture coordinator

  service:
    annotations: {}
    labels: {}
    clusterIP: ""

attu:
  enabled: false
```

```
  name: attu
  image:
    repository: zilliz/attu
    tag: v2.2.8
    pullPolicy: IfNotPresent
  service:
    annotations: {}
    labels: {}
    type: ClusterIP
    port: 3000
    # loadBalancerIP: ""
  resources: {}
  podLabels: {}
  ingress:
    enabled: false
    annotations: {}
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    labels: {}
    hosts:
      - milvus-attu.local
    tls: []
    #  - secretName: chart-attu-tls
    #    hosts:
    #      - milvus-attu.local


## Configuration values for the minio dependency
## ref: https://github.com/minio/charts/blob/master/README.md
##

minio:
  enabled: false
  name: minio
  mode: distributed
  image:
    tag: "RELEASE.2023-03-20T20-16-18Z"
    pullPolicy: IfNotPresent
  accessKey: minioadmin
  secretKey: minioadmin
  existingSecret: ""
  bucketName: "milvus-bucket"
  rootPath: file
  useIAM: false
  iamEndpoint: ""
  region: ""
```

```
  useVirtualHost: false
  podDisruptionBudget:
    enabled: false
  resources:
    requests:
      memory: 2Gi

  gcsgateway:
    enabled: false
    replicas: 1
    gcsKeyJson: "/etc/credentials/gcs_key.json"
    projectId: ""

  service:
    type: ClusterIP
    port: 9000

  persistence:
    enabled: true
    existingClaim: ""
    storageClass:
    accessMode: ReadWriteOnce
    size: 500Gi

  livenessProbe:
    enabled: true
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 5

  readinessProbe:
    enabled: true
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 1
    successThreshold: 1
    failureThreshold: 5

  startupProbe:
    enabled: true
    initialDelaySeconds: 0
    periodSeconds: 10
    timeoutSeconds: 5
    successThreshold: 1
```

```
    failureThreshold: 60

## Configuration values for the etcd dependency
## ref: https://artifacthub.io/packages/helm/bitnami/etcd
##

etcd:
  enabled: true
  name: etcd
  replicaCount: 3
  pdb:
    create: false
  image:
    repository: "milvusdb/etcd"
    tag: "3.5.5-r2"
    pullPolicy: IfNotPresent

  service:
    type: ClusterIP
    port: 2379
    peerPort: 2380

  auth:
    rbac:
      enabled: false

  persistence:
    enabled: true
    storageClass: default
    accessMode: ReadWriteOnce
    size: 10Gi

  ## Change default timeout periods to mitigate zoobie probe process
  livenessProbe:
    enabled: true
    timeoutSeconds: 10

  readinessProbe:
    enabled: true
    periodSeconds: 20
    timeoutSeconds: 10

  ## Enable auto compaction
  ## compaction by every 1000 revision
  ##
  autoCompactionMode: revision
```

```
  autoCompactionRetention: "1000"

  ## Increase default quota to 4G
  ##
  extraEnvVars:
  - name: ETCD_QUOTA_BACKEND_BYTES
    value: "4294967296"
  - name: ETCD_HEARTBEAT_INTERVAL
    value: "500"
  - name: ETCD_ELECTION_TIMEOUT
    value: "2500"

## Configuration values for the pulsar dependency
## ref: https://github.com/apache/pulsar-helm-chart
##

pulsar:
  enabled: true
  name: pulsar

  fullnameOverride: ""
  persistence: true

  maxMessageSize: "5242880"  # 5 * 1024 * 1024 Bytes, Maximum size of each
message in pulsar.

  rbac:
    enabled: false
    psp: false
    limit_to_namespace: true

  affinity:
    anti_affinity: false

## enableAntiAffinity: no

  components:
    zookeeper: true
    bookkeeper: true
    # bookkeeper - autorecovery
    autorecovery: true
    broker: true
    functions: false
    proxy: true
    toolset: false
    pulsar_manager: false
```

```
  monitoring:
    prometheus: false
    grafana: false
    node_exporter: false
    alert_manager: false

  images:
    broker:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    autorecovery:
      repository: apachepulsar/pulsar
      tag: 2.8.2
      pullPolicy: IfNotPresent
    zookeeper:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    bookie:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    proxy:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    pulsar_manager:
      repository: apachepulsar/pulsar-manager
      pullPolicy: IfNotPresent
      tag: v0.1.0

  zookeeper:
    volumes:
      persistence: true
      data:
        name: data
        size: 20Gi   #SSD Required
        storageClassName: default
    resources:
      requests:
        memory: 1024Mi
        cpu: 0.3
    configData:
      PULSAR_MEM: >
        -Xms1024m
```

```
        -Xmx1024m
      PULSAR_GC: >
         -Dcom.sun.management.jmxremote
         -Djute.maxbuffer=10485760
         -XX:+ParallelRefProcEnabled
         -XX:+UnlockExperimentalVMOptions
         -XX:+DoEscapeAnalysis
         -XX:+DisableExplicitGC
         -XX:+PerfDisableSharedMem
         -Dzookeeper.forceSync=no
    pdb:
      usePolicy: false

  bookkeeper:
    replicaCount: 3
    volumes:
      persistence: true
      journal:
        name: journal
        size: 100Gi
        storageClassName: default
      ledgers:
        name: ledgers
        size: 200Gi
        storageClassName: default
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+UseG1GC -XX:MaxGCPauseMillis=10
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
```

```
        -XX:+ExitOnOutOfMemoryError
        -XX:+PerfDisableSharedMem
        -XX:+PrintGCDetails
      nettyMaxFrameSizeBytes: "104867840"
    pdb:
      usePolicy: false

  broker:
    component: broker
    podMonitor:
      enabled: false
    replicaCount: 1
    resources:
      requests:
        memory: 4096Mi
        cpu: 1.5
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
      maxMessageSize: "104857600"
      defaultRetentionTimeInMinutes: "10080"
      defaultRetentionSizeInMB: "-1"
      backlogQuotaDefaultLimitGB: "8"
      ttlDurationDefaultInSeconds: "259200"
      subscriptionExpirationTimeMinutes: "3"
      backlogQuotaDefaultRetentionPolicy: producer_exception
    pdb:
      usePolicy: false

  autorecovery:
    resources:
      requests:
```

```
        memory: 512Mi
        cpu: 1

  proxy:
    replicaCount: 1
    podMonitor:
      enabled: false
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    service:
      type: ClusterIP
    ports:
      pulsar: 6650
    configData:
      PULSAR_MEM: >
        -Xms2048m -Xmx2048m
      PULSAR_GC: >
        -XX:MaxDirectMemorySize=2048m
      httpNumThreads: "100"
    pdb:
      usePolicy: false

  pulsar_manager:
    service:
      type: ClusterIP

  pulsar_metadata:
    component: pulsar-init
    image:
      # the image used for running `pulsar-cluster-initialize` job
      repository: apachepulsar/pulsar
      tag: 2.8.2


## Configuration values for the kafka dependency
## ref: https://artifacthub.io/packages/helm/bitnami/kafka
##

kafka:
  enabled: false
  name: kafka
  replicaCount: 3
  image:
    repository: bitnami/kafka
```

```
    tag: 3.1.0-debian-10-r52
  ## Increase graceful termination for kafka graceful shutdown
  terminationGracePeriodSeconds: "90"
  pdb:
    create: false

  ## Enable startup probe to prevent pod restart during recovering
  startupProbe:
    enabled: true

  ## Kafka Java Heap size
  heapOpts: "-Xmx4096m -Xms4096m"
  maxMessageBytes: _10485760
  defaultReplicationFactor: 3
  offsetsTopicReplicationFactor: 3
  ## Only enable time based log retention
  logRetentionHours: 168
  logRetentionBytes: _-1
  extraEnvVars:
  - name: KAFKA_CFG_MAX_PARTITION_FETCH_BYTES
    value: "5242880"
  - name: KAFKA_CFG_MAX_REQUEST_SIZE
    value: "5242880"
  - name: KAFKA_CFG_REPLICA_FETCH_MAX_BYTES
    value: "10485760"
  - name: KAFKA_CFG_FETCH_MESSAGE_MAX_BYTES
    value: "5242880"
  - name: KAFKA_CFG_LOG_ROLL_HOURS
    value: "24"

  persistence:
    enabled: true
    storageClass:
    accessMode: ReadWriteOnce
    size: 300Gi

  metrics:
    ## Prometheus Kafka exporter: exposes complimentary metrics to JMX
exporter
    kafka:
      enabled: false
      image:
        repository: bitnami/kafka-exporter
        tag: 1.4.2-debian-10-r182

    ## Prometheus JMX exporter: exposes the majority of Kafkas metrics
```

```
    jmx:
      enabled: false
      image:
        repository: bitnami/jmx-exporter
        tag: 0.16.1-debian-10-r245

    ## To enable serviceMonitor, you must enable either kafka exporter or
jmx exporter.
    ## And you can enable them both
    serviceMonitor:
      enabled: false

  service:
    type: ClusterIP
    ports:
      client: 9092

  zookeeper:
    enabled: true
    replicaCount: 3

###################################
# External S3
# - these configs are only used when `externalS3.enabled` is true
###################################
externalS3:
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
  useSSL: false
  bucketName: "milvusdbvol1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false

###################################
# GCS Gateway
# - these configs are only used when `minio.gcsgateway.enabled` is true
###################################
externalGcs:
  bucketName: ""
```

```
###################################
# External etcd
# - these configs are only used when `externalEtcd.enabled` is true
###################################
externalEtcd:
  enabled: false
  ## the endpoints of the external etcd
  ##
  endpoints:
    - localhost:2379

###################################
# External pulsar
# - these configs are only used when `externalPulsar.enabled` is true
###################################
externalPulsar:
  enabled: false
  host: localhost
  port: 6650
  maxMessageSize: "5242880"  # 5 * 1024 * 1024 Bytes, Maximum size of each
message in pulsar.
  tenant: public
  namespace: default
  authPlugin: ""
  authParams: ""

###################################
# External kafka
# - these configs are only used when `externalKafka.enabled` is true
###################################
externalKafka:
  enabled: false
  brokerList: localhost:9092
  securityProtocol: SASL_SSL
  sasl:
    mechanisms: PLAIN
    username: ""
    password: ""
root@node2:~#
```

附录B：prepare_data_netapp_new.py

附录B：prepare_data_netapp_new.py

```
```

```
root@node2:~# cat prepare_data_netapp_new.py
# hello_milvus.py demonstrates the basic operations of PyMilvus, a Python
SDK of Milvus.
# 1. connect to Milvus
# 2. create collection
# 3. insert data
# 4. create index
# 5. search, query, and hybrid search on entities
# 6. delete entities by PK
# 7. drop collection
import time
import os
import numpy as np
from pymilvus import (
    connections,
    utility,
    FieldSchema, CollectionSchema, DataType,
    Collection,
)

fmt = "\n=== {:30} ===\n"
search_latency_fmt = "search latency = {:.4f}s"
#num_entities, dim = 3000, 8
num_entities, dim = 3000, 16

################################################################################
#######
# 1. connect to Milvus
# Add a new connection alias `default` for Milvus server in
`localhost:19530`
# Actually the "default" alias is a buildin in PyMilvus.
# If the address of Milvus is the same as `localhost:19530`, you can omit
all
# parameters and call the method as: `connections.connect()`.
#
# Note: the `using` parameter of the following methods is default to
"default".
print(fmt.format("start connecting to Milvus"))

host = os.environ.get('MILVUS_HOST')
if host == None:
    host = "localhost"
print(fmt.format(f"Milvus host: {host}"))
#connections.connect("default", host=host, port="19530")
connections.connect("default", host=host, port="27017")
```

```
has = utility.has_collection("hello_milvus_ntapnew_update2_sc")
print(f"Does collection hello_milvus_ntapnew_update2_sc exist in Milvus:
{has}")

#drop the collection
print(fmt.format(f"Drop collection - hello_milvus_ntapnew_update2_sc"))
utility.drop_collection("hello_milvus_ntapnew_update2_sc")
#drop the collection
print(fmt.format(f"Drop collection - hello_milvus_ntapnew_update2_sc2"))
utility.drop_collection("hello_milvus_ntapnew_update2_sc2")

################################################################################
#######
# 2. create collection
# We're going to create a collection with 3 fields.
# +-+------------+------------+------------------
+------------------------------+
# | | field name | field type | other attributes |       field description
|
# +-+------------+------------+------------------
+------------------------------+
# |1|    "pk"    |    Int64   |  is_primary=True |      "primary field"
|
# | |            |            |   auto_id=False  |
|
# +-+------------+------------+------------------
+------------------------------+
# |2|  "random"  |    Double  |                  |      "a double field"
|
# +-+------------+------------+------------------
+------------------------------+
# |3|"embeddings"| FloatVector|     dim=8        |  "float vector with dim
8"   |
# +-+------------+------------+------------------
+------------------------------+
fields = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=False),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]

schema = CollectionSchema(fields, "hello_milvus_ntapnew_update2_sc")

print(fmt.format("Create collection `hello_milvus_ntapnew_update2_sc`"))
```

```
hello_milvus_ntapnew_update2_sc = Collection
("hello_milvus_ntapnew_update2_sc", schema, consistency_level="Strong")

################################################################################
######
# 3. insert data
# We are going to insert 3000 rows of data into
`hello_milvus_ntapnew_update2_sc`
# Data to be inserted must be organized in fields.
#
# The insert() method returns:
# - either automatically generated primary keys by Milvus if auto_id=True
in the schema;
# - or the existing primary key field from the entities if auto_id=False
in the schema.

print(fmt.format("Start inserting entities"))
rng = np.random.default_rng(seed=19530)
entities = [
    # provide the pk field because `auto_id` is set to False
    [i for i in range(num_entities)],
    rng.random(num_entities).tolist(),  # field random, only supports list
    [str(i) for i in range(num_entities)],
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

insert_result = hello_milvus_ntapnew_update2_sc.insert(entities)
hello_milvus_ntapnew_update2_sc.flush()
print(f"Number of entities in hello_milvus_ntapnew_update2_sc:
{hello_milvus_ntapnew_update2_sc.num_entities}")  # check the num_entites

# create another collection
fields2 = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id
=True),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="var", dtype=DataType.VARCHAR, max_length=65535),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=dim)
]


schema2 = CollectionSchema(fields2, "hello_milvus_ntapnew_update2_sc2")

print(fmt.format("Create collection `hello_milvus_ntapnew_update2_sc2`"))
hello_milvus_ntapnew_update2_sc2 = Collection
```

```
("hello_milvus_ntapnew_update2_sc2", schema2, consistency_level="Strong")

entities2 = [
    rng.random(num_entities).tolist(),  # field random, only supports list
    [str(i) for i in range(num_entities)],
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

insert_result2 = hello_milvus_ntapnew_update2_sc2.insert(entities2)
hello_milvus_ntapnew_update2_sc2.flush()
insert_result2 = hello_milvus_ntapnew_update2_sc2.insert(entities2)
hello_milvus_ntapnew_update2_sc2.flush()

# index_params = {"index_type": "IVF_FLAT", "params": {"nlist": 128},
"metric_type": "L2"}
# hello_milvus_ntapnew_update2_sc.create_index("embeddings", index_params)
#
hello_milvus_ntapnew_update2_sc2.create_index(field_name="var",index_name=
"scalar_index")

# index_params2 = {"index_type": "Trie"}
# hello_milvus_ntapnew_update2_sc2.create_index("var", index_params2)

print(f"Number of entities in hello_milvus_ntapnew_update2_sc2:
{hello_milvus_ntapnew_update2_sc2.num_entities}")  # check the num_entites

root@node2:~#
```

附录C：verify_data_netapp.py

附录C：verify_data_netapp.py

```
root@node2:~# cat verify_data_netapp.py
import time
import os
import numpy as np
from pymilvus import (
    connections,
    utility,
    FieldSchema, CollectionSchema, DataType,
    Collection,
)

fmt = "\n=== {:30} ===\n"
```

```
search_latency_fmt = "search latency = {:.4f}s"
num_entities, dim = 3000, 16
rng = np.random.default_rng(seed=19530)
entities = [
    # provide the pk field because `auto_id` is set to False
    [i for i in range(num_entities)],
    rng.random(num_entities).tolist(),  # field random, only supports list
    rng.random((num_entities, dim)),    # field embeddings, supports
numpy.ndarray and list
]

################################################################################
######
# 1. get recovered collection hello_milvus_ntapnew_update2_sc
print(fmt.format("start connecting to Milvus"))
host = os.environ.get('MILVUS_HOST')
if host == None:
    host = "localhost"
print(fmt.format(f"Milvus host: {host}"))
#connections.connect("default", host=host, port="19530")
connections.connect("default", host=host, port="27017")

recover_collections = ["hello_milvus_ntapnew_update2_sc",
"hello_milvus_ntapnew_update2_sc2"]

for recover_collection_name in recover_collections:
    has = utility.has_collection(recover_collection_name)
    print(f"Does collection {recover_collection_name} exist in Milvus:
{has}")
    recover_collection = Collection(recover_collection_name)
    print(recover_collection.schema)
    recover_collection.flush()

    print(f"Number of entities in Milvus: {recover_collection_name} :
{recover_collection.num_entities}")  # check the num_entites


################################################################################
######
    # 4. create index
    # We are going to create an IVF_FLAT index for
hello_milvus_ntapnew_update2_sc collection.
    # create_index() can only be applied to `FloatVector` and
`BinaryVector` fields.
    print(fmt.format("Start Creating index IVF_FLAT"))
    index = {
```

```
        "index_type": "IVF_FLAT",
        "metric_type": "L2",
        "params": {"nlist": 128},
    }

    recover_collection.create_index("embeddings", index)


########################################################################
######
    # 5. search, query, and hybrid search
    # After data were inserted into Milvus and indexed, you can perform:
    # - search based on vector similarity
    # - query based on scalar filtering(boolean, int, etc.)
    # - hybrid search based on vector similarity and scalar filtering.
    #

    # Before conducting a search or a query, you need to load the data in
`hello_milvus` into memory.
    print(fmt.format("Start loading"))
    recover_collection.load()

    #
------------------------------------------------------------------------
---
    # search based on vector similarity
    print(fmt.format("Start searching based on vector similarity"))
    vectors_to_search = entities[-1][-2:]
    search_params = {
        "metric_type": "L2",
        "params": {"nprobe": 10},
    }

    start_time = time.time()
    result = recover_collection.search(vectors_to_search, "embeddings",
search_params, limit=3, output_fields=["random"])
    end_time = time.time()

    for hits in result:
        for hit in hits:
            print(f"hit: {hit}, random field: {hit.entity.get('random')}")
    print(search_latency_fmt.format(end_time - start_time))

    #
------------------------------------------------------------------------
---
```

```
    # query based on scalar filtering(boolean, int, etc.)
    print(fmt.format("Start querying with `random > 0.5`"))

    start_time = time.time()
    result = recover_collection.query(expr="random > 0.5", output_fields=
["random", "embeddings"])
    end_time = time.time()

    print(f"query result:\n-{result[0]}")
    print(search_latency_fmt.format(end_time - start_time))

    #
-------------------------------------------------------------------------
---
    # hybrid search
    print(fmt.format("Start hybrid searching with `random > 0.5`"))

    start_time = time.time()
    result = recover_collection.search(vectors_to_search, "embeddings",
search_params, limit=3, expr="random > 0.5", output_fields=["random"])
    end_time = time.time()

    for hits in result:
        for hit in hits:
            print(f"hit: {hit}, random field: {hit.entity.get('random')}")
    print(search_latency_fmt.format(end_time - start_time))


###########################################################################
#####
    # 7. drop collection
    # Finally, drop the hello_milvus, hello_milvus_ntapnew_update2_sc
collection

    #print(fmt.format(f"Drop collection {recover_collection_name}"))
    #utility.drop_collection(recover_collection_name)

root@node2:~#
```

附录D：dkder-compose．yml

附录D：dkder-compose．yml

```
version: '3.5'
```

```
services:
  etcd:
    container_name: milvus-etcd
    image: quay.io/coreos/etcd:v3.5.5
    environment:
      - ETCD_AUTO_COMPACTION_MODE=revision
      - ETCD_AUTO_COMPACTION_RETENTION=1000
      - ETCD_QUOTA_BACKEND_BYTES=4294967296
      - ETCD_SNAPSHOT_COUNT=50000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/etcd:/etcd
    command: etcd -advertise-client-urls=http://127.0.0.1:2379 -listen
-client-urls http://0.0.0.0:2379 --data-dir /etcd
    healthcheck:
      test: ["CMD", "etcdctl", "endpoint", "health"]
      interval: 30s
      timeout: 20s
      retries: 3

  minio:
    container_name: milvus-minio
    image: minio/minio:RELEASE.2023-03-20T20-16-18Z
    environment:
      MINIO_ACCESS_KEY: minioadmin
      MINIO_SECRET_KEY: minioadmin
    ports:
      - "9001:9001"
      - "9000:9000"
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/minio:/minio_data
    command: minio server /minio_data --console-address ":9001"
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost:9000/minio/health/live"]
      interval: 30s
      timeout: 20s
      retries: 3

  standalone:
    container_name: milvus-standalone
    image: milvusdb/milvus:v2.4.0-rc.1
    command: ["milvus", "run", "standalone"]
    security_opt:
    - seccomp:unconfined
    environment:
      ETCD_ENDPOINTS: etcd:2379
```

```
      MINIO_ADDRESS: minio:9000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/milvus:/var/lib/milvus
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:9091/healthz"]
      interval: 30s
      start_period: 90s
      timeout: 20s
      retries: 3
    ports:
      - "19530:19530"
      - "9091:9091"
    depends_on:
      - "etcd"
      - "minio"

networks:
  default:
    name: milvus
```