Solução de banco de dados vetorial com NetApp

NetApp artificial intelligence solutions

NetApp
August 18, 2025

Índice

Solução de banco de dados vetorial com NetApp

Solução de banco de dados vetorial com NetApp

Karthikeyan Nagalingam e Rodrigo Nascimento, NetApp

Este documento fornece uma exploração completa da implantação e do gerenciamento de bancos de dados vetoriais, como Milvus e pgvecto, uma extensão PostgreSQL de código aberto, usando as soluções de armazenamento da NetApp. Ele detalha as diretrizes de infraestrutura para usar o armazenamento de objetos NetApp ONTAP e StorageGRID e valida a aplicação do banco de dados Milvus no AWS FSx ONTAP. O documento elucida a dualidade arquivo-objeto do NetApp e sua utilidade para bancos de dados vetoriais e aplicativos que oferecem suporte a incorporações vetoriais. Ele enfatiza os recursos do SnapCenter, produto de gerenciamento empresarial da NetApp, ao oferecer funcionalidades de backup e restauração para bancos de dados vetoriais, garantindo a integridade e a disponibilidade dos dados. O documento se aprofunda ainda mais na solução de nuvem híbrida da NetApp, discutindo seu papel na replicação e proteção de dados em ambientes locais e na nuvem. Ele inclui insights sobre a validação de desempenho de bancos de dados vetoriais no NetApp ONTAP e conclui com dois casos de uso prático em IA generativa: RAG com LLM e ChatAI interno da NetApp. Este documento serve como um guia abrangente para aproveitar as soluções de armazenamento da NetApp para gerenciar bancos de dados vetoriais.

A Arquitetura de Referência concentra-se no seguinte:

13. "Apêndice D: docker-compose.yml"

Introdução

Esta seção fornece uma introdução à solução de banco de dados vetorial para NetApp.

Introdução

Os bancos de dados vetoriais abordam efetivamente os desafios projetados para lidar com as complexidades da pesquisa semântica em Grandes Modelos de Linguagem (LLMs) e Inteligência Artificial generativa (IA). Diferentemente dos sistemas tradicionais de gerenciamento de dados, os bancos de dados vetoriais são capazes de processar e pesquisar vários tipos de dados, incluindo imagens, vídeos, texto, áudio e outras formas de dados não estruturados, usando o conteúdo dos dados em si, em vez de rótulos ou tags.

As limitações dos Sistemas de Gerenciamento de Banco de Dados Relacionais (RDBMS) são bem documentadas, particularmente suas dificuldades com representações de dados de alta dimensão e dados não estruturados comuns em aplicativos de IA. Os RDBMS geralmente exigem um processo demorado e sujeito a erros de nivelamento de dados em estruturas mais gerenciáveis, o que leva a atrasos e ineficiências nas pesquisas. No entanto, os bancos de dados vetoriais são projetados para contornar esses problemas, oferecendo uma solução mais eficiente e precisa para gerenciar e pesquisar dados complexos e de alta dimensão, facilitando assim o avanço de aplicações de IA.

Este documento serve como um guia abrangente para clientes que estão usando ou planejam usar bancos de dados vetoriais, detalhando as práticas recomendadas para utilizar bancos de dados vetoriais em plataformas como NetApp ONTAP, NetApp StorageGRID, Amazon FSx ONTAP para NetApp ONTAP e SnapCenter. O conteúdo fornecido aqui abrange uma variedade de tópicos:

- Diretrizes de infraestrutura para bancos de dados vetoriais, como Milvus, fornecidas pelo armazenamento NetApp por meio do NetApp ONTAP e do armazenamento de objetos StorageGRID .
- Validação do banco de dados Milvus no AWS FSx ONTAP por meio de armazenamento de arquivos e objetos.
- Investiga a dualidade arquivo-objeto do NetApp, demonstrando sua utilidade para dados em bancos de dados vetoriais, bem como outros aplicativos.
- Como o produto de gerenciamento de proteção de dados da NetApp, SnapCenter, oferece funcionalidades de backup e restauração para dados de bancos de dados vetoriais.
- Como a Nuvem Híbrida da NetApp oferece replicação e proteção de dados em ambientes locais e na nuvem.
- Fornece insights sobre a validação de desempenho de bancos de dados vetoriais como Milvus e pgvector no NetApp ONTAP.
- Dois casos de uso específicos: Retrieval Augmented Generation (RAG) com Large Language Models (LLM) e o ChatAI da equipe de TI da NetApp , oferecendo assim exemplos práticos dos conceitos e práticas descritos.

Visão geral da solução

Esta seção fornece uma visão geral da solução de banco de dados vetorial da NetApp .

Visão geral da solução

Esta solução mostra os benefícios e recursos diferenciados que a NetApp oferece para enfrentar os desafios enfrentados pelos clientes de bancos de dados vetoriais. Ao aproveitar o NetApp ONTAP, o StorageGRID, as soluções de nuvem da NetApp e o SnapCenter, os clientes podem agregar valor significativo às suas operações comerciais. Essas ferramentas não apenas resolvem problemas existentes, mas também aumentam a eficiência e a produtividade, contribuindo assim para o crescimento geral do negócio.

Por que a NetApp?

- As ofertas da NetApp, como ONTAP e StorageGRID, permitem a separação de armazenamento e computação, possibilitando a utilização ideal de recursos com base em requisitos específicos. Essa flexibilidade permite que os clientes dimensionem seu armazenamento de forma independente usando soluções de armazenamento da NetApp .
- Ao aproveitar os controladores de armazenamento da NetApp, os clientes podem fornecer dados de forma eficiente para seu banco de dados vetorial usando os protocolos NFS e S3. Esses protocolos facilitam o armazenamento de dados do cliente e gerenciam o índice do banco de dados vetorial, eliminando a necessidade de múltiplas cópias de dados acessadas por meio de métodos de arquivo e objeto.
- O NetApp ONTAP fornece suporte nativo para NAS e armazenamento de objetos nos principais provedores de serviços de nuvem, como AWS, Azure e Google Cloud. Essa ampla compatibilidade garante integração perfeita, permitindo mobilidade de dados do cliente, acessibilidade global, recuperação de desastres, escalabilidade dinâmica e alto desempenho.
- Com os recursos robustos de gerenciamento de dados da NetApp, os clientes podem ficar tranquilos sabendo que seus dados estão bem protegidos contra potenciais riscos e ameaças. A NetApp prioriza a segurança de dados, oferecendo tranquilidade aos clientes quanto à segurança e integridade de suas informações valiosas.

Banco de Dados Vetorial

Esta seção aborda a definição e o uso de um banco de dados vetorial em soluções de IA da NetApp .

Banco de Dados Vetorial

Um banco de dados vetorial é um tipo especializado de banco de dados projetado para manipular, indexar e pesquisar dados não estruturados usando incorporações de modelos de aprendizado de máquina. Em vez de organizar dados em um formato tabular tradicional, ele organiza os dados como vetores de alta dimensão, também conhecidos como incorporações de vetores. Essa estrutura exclusiva permite que o banco de dados manipule dados complexos e multidimensionais de forma mais eficiente e precisa.

Um dos principais recursos de um banco de dados vetorial é o uso de IA generativa para realizar análises. Isso inclui pesquisas de similaridade, onde o banco de dados identifica pontos de dados que são semelhantes a uma determinada entrada, e detecção de anomalias, onde ele pode identificar pontos de dados que se desviam significativamente da norma.

Além disso, bancos de dados vetoriais são adequados para lidar com dados temporais ou dados com registro de data e hora. Esse tipo de dado fornece informações sobre "o que" aconteceu e quando aconteceu, em sequência e em relação a todos os outros eventos dentro de um determinado sistema de TI. Essa capacidade de manipular e analisar dados temporais torna os bancos de dados vetoriais particularmente úteis para aplicações que exigem uma compreensão de eventos ao longo do tempo.

Vantagens do banco de dados vetorial para ML e IA:

- Pesquisa de alta dimensão: bancos de dados vetoriais são excelentes no gerenciamento e na recuperação de dados de alta dimensão, que geralmente são gerados em aplicativos de IA e ML.
- Escalabilidade: eles podem ser dimensionados com eficiência para lidar com grandes volumes de dados, dando suporte ao crescimento e à expansão de projetos de IA e ML.
- Flexibilidade: Bancos de dados vetoriais oferecem um alto grau de flexibilidade, permitindo a acomodação de diversos tipos e estruturas de dados.
- Desempenho: eles fornecem gerenciamento e recuperação de dados de alto desempenho, essenciais para a velocidade e eficiência das operações de IA e ML.
- Indexação personalizável: os bancos de dados vetoriais oferecem opções de indexação personalizáveis, permitindo organização e recuperação otimizadas de dados com base em necessidades específicas.

Bancos de dados vetoriais e casos de uso.

Esta seção fornece vários bancos de dados de vetores e detalhes de seus casos de uso.

Faiss e ScaNN

São bibliotecas que servem como ferramentas cruciais no âmbito da pesquisa vetorial. Essas bibliotecas fornecem funcionalidades essenciais para gerenciar e pesquisar dados vetoriais, o que as torna recursos inestimáveis nessa área especializada de gerenciamento de dados.

Elasticsearch

É um mecanismo de busca e análise amplamente utilizado e recentemente incorporou recursos de busca vetorial. Este novo recurso aprimora sua funcionalidade, permitindo que ele manipule e pesquise dados vetoriais de forma mais eficaz.

Pinha

É um banco de dados vetorial robusto com um conjunto exclusivo de recursos. Ele suporta vetores densos e esparsos em sua funcionalidade de indexação, o que aumenta sua flexibilidade e adaptabilidade. Um dos seus principais pontos fortes está na capacidade de combinar métodos de pesquisa tradicionais com pesquisa vetorial densa baseada em IA, criando uma abordagem de pesquisa híbrida que aproveita o melhor dos dois mundos.

Basicamente baseado em nuvem, o Pinecone foi projetado para aplicativos de aprendizado de máquina e se integra bem a uma variedade de plataformas, incluindo GCP, AWS, Open AI, GPT-3, GPT-3.5, GPT-4, Catgut Plus, Elasticsearch, Haystack e muito mais. É importante observar que o Pinecone é uma plataforma de código fechado e está disponível como uma oferta de Software como Serviço (SaaS).

Dadas suas capacidades avançadas, o Pinecone é particularmente adequado para o setor de segurança cibernética, onde seus recursos de pesquisa de alta dimensão e pesquisa híbrida podem ser aproveitados efetivamente para detectar e responder a ameaças.

Croma

É um banco de dados vetorial que tem uma Core-API com quatro funções principais, uma das quais inclui um armazenamento de vetores de documentos na memória. Ele também utiliza a biblioteca Face Transformers para vetorizar documentos, aumentando sua funcionalidade e versatilidade. O Chroma foi projetado para operar na nuvem e no local, oferecendo flexibilidade com base nas necessidades do usuário. Particularmente, ele se destaca em aplicações relacionadas a áudio, o que o torna uma excelente escolha para mecanismos

de busca baseados em áudio, sistemas de recomendação de música e outros casos de uso relacionados a áudio.

Weaviate

É um banco de dados vetorial versátil que permite aos usuários vetorizar seu conteúdo usando seus módulos integrados ou módulos personalizados, proporcionando flexibilidade com base em necessidades específicas. Ela oferece soluções totalmente gerenciadas e auto-hospedadas, atendendo a uma variedade de preferências de implantação.

Um dos principais recursos do Weaviate é sua capacidade de armazenar vetores e objetos, aprimorando suas capacidades de manipulação de dados. É amplamente utilizado para uma variedade de aplicações, incluindo pesquisa semântica e classificação de dados em sistemas ERP. No setor de comércio eletrônico, ele potencializa mecanismos de busca e recomendação. O Weaviate também é usado para pesquisa de imagens, detecção de anomalias, harmonização automatizada de dados e análise de ameaças à segurança cibernética, demonstrando sua versatilidade em vários domínios.

Redis

O Redis é um banco de dados vetorial de alto desempenho conhecido por seu rápido armazenamento na memória, oferecendo baixa latência para operações de leitura e gravação. Isso o torna uma excelente escolha para sistemas de recomendação, mecanismos de busca e aplicativos de análise de dados que exigem acesso rápido aos dados.

O Redis suporta várias estruturas de dados para vetores, incluindo listas, conjuntos e conjuntos classificados. Ele também fornece operações vetoriais, como calcular distâncias entre vetores ou encontrar interseções e uniões. Esses recursos são particularmente úteis para sistemas de pesquisa por similaridade, agrupamento e recomendação baseados em conteúdo.

Em termos de escalabilidade e disponibilidade, o Redis se destaca no tratamento de cargas de trabalho de alto rendimento e oferece replicação de dados. Ele também se integra bem com outros tipos de dados, incluindo bancos de dados relacionais tradicionais (RDBMS). O Redis inclui um recurso Publicar/Assinar (Pub/Sub) para atualizações em tempo real, o que é benéfico para gerenciar vetores em tempo real. Além disso, o Redis é leve e simples de usar, o que o torna uma solução fácil de usar para gerenciar dados vetoriais.

Milvus

É um banco de dados vetorial versátil que oferece uma API como um armazenamento de documentos, muito parecido com o MongoDB. Ele se destaca pelo suporte a uma ampla variedade de tipos de dados, o que o torna uma escolha popular nas áreas de ciência de dados e aprendizado de máquina.

Um dos recursos exclusivos do Milvus é sua capacidade de multivetorização, que permite aos usuários especificar em tempo de execução o tipo de vetor a ser usado para a pesquisa. Além disso, ele utiliza o Knowwhere, uma biblioteca que fica em cima de outras bibliotecas como a Faiss, para gerenciar a comunicação entre consultas e algoritmos de busca vetorial.

O Milvus também oferece integração perfeita com fluxos de trabalho de aprendizado de máquina, graças à sua compatibilidade com PyTorch e TensorFlow. Isso o torna uma excelente ferramenta para uma variedade de aplicações, incluindo comércio eletrônico, análise de imagens e vídeos, reconhecimento de objetos, pesquisa de similaridade de imagens e recuperação de imagens baseada em conteúdo. No campo do processamento de linguagem natural, o Milvus é usado para agrupamento de documentos, pesquisa semântica e sistemas de perguntas e respostas.

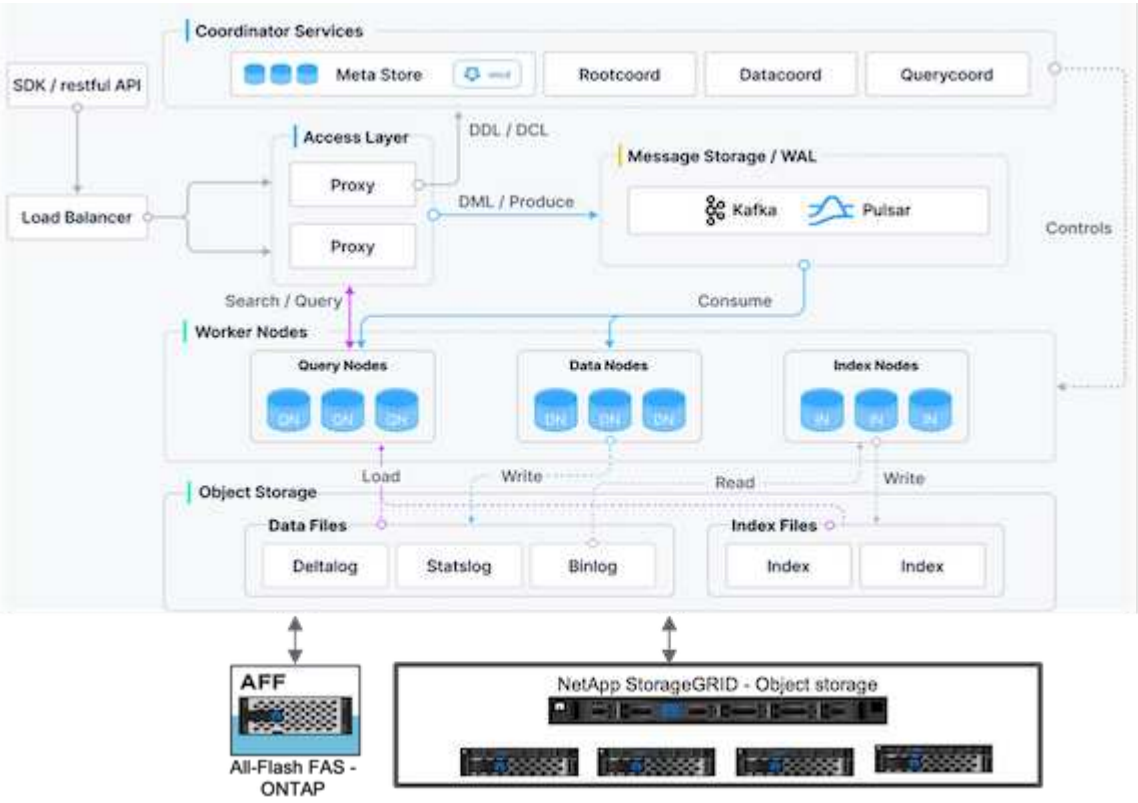
Para esta solução, escolhemos o milvus para a validação da solução. Para desempenho, usamos milvus e

postgres(pgvecto.rs).

Por que escolhemos o milvus para esta solução?

- Código aberto: Milvus é um banco de dados vetorial de código aberto, que incentiva o desenvolvimento e as melhorias conduzidas pela comunidade.
- Integração de IA: aproveita a incorporação de pesquisa de similaridade e aplicativos de IA para aprimorar a funcionalidade do banco de dados vetorial.
- Manuseio de grande volume: o Milvus tem capacidade para armazenar, indexar e gerenciar mais de um bilhão de vetores de incorporação gerados por redes neurais profundas (DNN) e modelos de aprendizado de máquina (ML).
- Fácil de usar: é fácil de usar e a configuração leva menos de um minuto. A Milvus também oferece SDKs para diferentes linguagens de programação.
- Velocidade: Oferece velocidades de recuperação extremamente rápidas, até 10 vezes mais rápidas que algumas alternativas.
- Escalabilidade e disponibilidade: o Milvus é altamente escalável, com opções de expansão vertical e horizontal conforme necessário.
- Rico em recursos: suporta diferentes tipos de dados, filtragem de atributos, suporte a funções definidas pelo usuário (UDF), níveis de consistência configuráveis e tempo de viagem, o que o torna uma ferramenta versátil para várias aplicações.

Visão geral da arquitetura Milvus



Esta seção fornece componentes e serviços de alto nível usados na arquitetura Milvus. * Camada de acesso – É composta por um grupo de proxies sem estado e serve como camada frontal do sistema e ponto final para os usuários. * Serviço de coordenação – ele atribui tarefas aos nós de trabalho e atua como o cérebro do sistema. Ele tem três tipos de coordenadores: coordenada raiz, coordenada de dados e coordenada de consulta. * Nós de trabalho: seguem as instruções do serviço do coordenador e executam comandos

DML/DDL acionados pelo usuário. Possui três tipos de nós de trabalho: nó de consulta, nó de dados e nó de índice. * Armazenamento: é responsável pela persistência dos dados. Ele abrange meta armazenamento, log broker e armazenamento de objetos. O armazenamento da NetApp , como ONTAP e StorageGRID, fornece armazenamento de objetos e armazenamento baseado em arquivos para a Milvus para dados de clientes e dados de banco de dados vetorial.

Requisito de Tecnologia

Esta seção fornece uma visão geral dos requisitos para a solução de banco de dados vetorial NetApp .

Requisito de Tecnologia

As configurações de hardware e software descritas abaixo foram utilizadas para a maioria das validações realizadas neste documento, com exceção do desempenho. Essas configurações servem como um guia para ajudar você a configurar seu ambiente. No entanto, observe que os componentes específicos podem variar dependendo dos requisitos individuais do cliente.

Requisitos de hardware

Hardware	Detalhes
Par de matriz de armazenamento AFF da NetApp HA	* A800 * ONTAP 9.14.1 * 48 x 3,49 TB SSD-NVM * Dois volumes de grupo flexíveis: metadados e dados. * O volume NFS de metadados tem 12 volumes persistentes com 250 GB. * Os dados são um volume ONTAP NAS S3
6 x FUJITSU PRIMERGY RX2540 M4	* 64 CPUs * CPU Intel® Xeon® Gold 6142 a 2,60 GHz * 256 GM de memória física * 1 porta de rede 100GbE
Rede	100 GbE
StorageGRID	* 1 x SG100, 3xSGF6024 * 3 x 24 x 7,68 TB

Requisitos de software

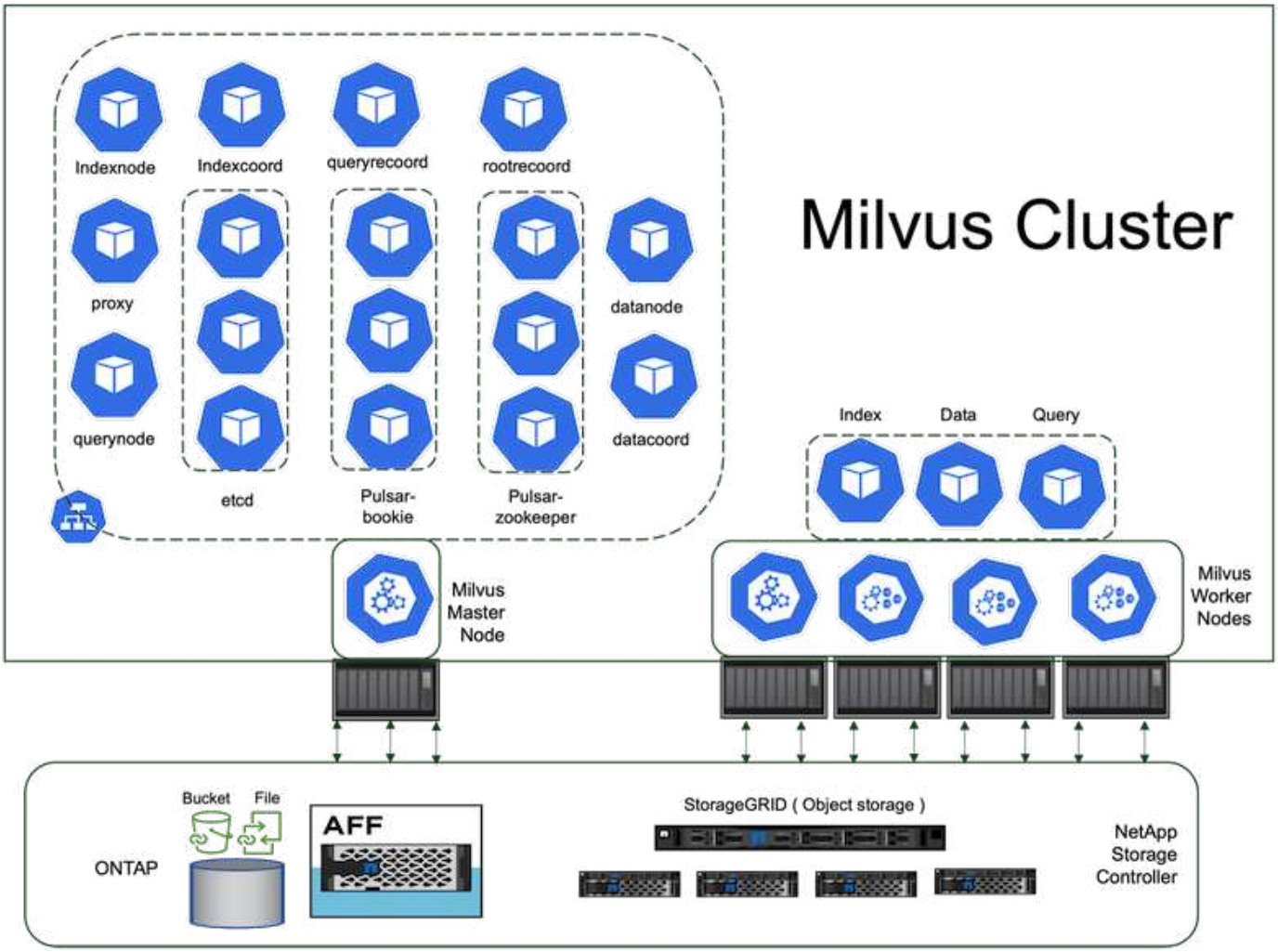
Software	Detalhes
Aglomerado de Milvus	* GRÁFICO - milvus-4.1.11. * Versão do APP – 2.3.4 * Pacotes dependentes como bookkeeper, zookeeper, pulsar, etcd, proxy, querynode, worker
Kubernetes	* Cluster K8s de 5 nós * 1 nó mestre e 4 nós de trabalho * Versão – 1.7.2
Pitão	*3.10.12.

Procedimento de Implantação

Esta seção discute o procedimento de implantação da solução de banco de dados vetorial para NetApp.

Procedimento de implantação

Nesta seção de implantação, usamos o banco de dados vetorial milvus com o Kubernetes para a configuração do laboratório, conforme abaixo.



O armazenamento netapp fornece armazenamento para o cluster para manter os dados dos clientes e os dados do cluster milvus.

Configuração de armazenamento NetApp – ONTAP

- Inicialização do sistema de armazenamento
- Criação de máquina virtual de armazenamento (SVM)
- Atribuição de interfaces de rede lógicas
- NFS, configuração e licenciamento S3

Siga os passos abaixo para NFS (Network File System):

1. Crie um volume FlexGroup para NFSv4. Em nossa configuração para esta validação, usamos 48 SSDs, 1 SSD dedicado para o volume raiz do controlador e 47 SSDs distribuídos para NFSv4]]. Verifique se a política de exportação NFS para o volume FlexGroup tem permissões de leitura/gravação para a rede de nós do Kubernetes (K8s). Se essas permissões não estiverem em vigor, conceda permissões de leitura/gravação (rw) para a rede de nós K8s.

2. Em todos os nós K8s, crie uma pasta e monte o volume FlexGroup nessa pasta por meio de uma Interface Lógica (LIF) em cada nó K8s.

Siga as etapas abaixo para NAS S3 (Network Attached Storage Simple Storage Service):

1. Crie um volume FlexGroup para NFS.
2. Configure um servidor de armazenamento de objetos com HTTP habilitado e o status do administrador definido como 'ativo' usando o comando "vserver object-store-server create". Você tem a opção de habilitar HTTPS e definir uma porta de escuta personalizada.
3. Crie um usuário object-store-server usando o comando "vserver object-store-server user create -user <nome de usuário>".
4. Para obter a chave de acesso e a chave secreta, você pode executar o seguinte comando: "set diag; vserver object-store-server user show -user <nome de usuário>". No entanto, a partir de agora, essas chaves serão fornecidas durante o processo de criação do usuário ou poderão ser recuperadas usando chamadas de API REST.
5. Estabeleça um grupo de objetos-armazenamento-servidor usando o usuário criado na etapa 2 e conceda acesso. Neste exemplo, fornecemos "FullAccess".
6. Crie um bucket NAS definindo seu tipo como "nas" e fornecendo o caminho para o volume NFSv3. Também é possível utilizar um bucket S3 para essa finalidade.

Configuração de armazenamento NetApp – StorageGRID

1. Instale o software storageGRID.
2. Crie um inquilino e um bucket.
3. Crie um usuário com a permissão necessária.

Por favor, verifique mais detalhes em https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

Verificação da solução

Visão geral da solução

Realizamos uma validação abrangente da solução focada em cinco áreas principais, cujos detalhes estão descritos abaixo. Cada seção aborda os desafios enfrentados pelos clientes, as soluções fornecidas pela NetApp e os benefícios subsequentes para o cliente.

1. "Configuração do cluster Milvus com Kubernetes no local"Os desafios do cliente são escalar de forma independente em armazenamento e computação, gerenciamento eficaz de infraestrutura e gerenciamento de dados. Nesta seção, detalhamos o processo de instalação de um cluster Milvus no Kubernetes, utilizando um controlador de armazenamento NetApp para dados do cluster e dados do cliente.
2. Milvus com Amazon FSx ONTAP para NetApp ONTAP – dualidade de arquivo e objeto Nesta seção, por que precisamos implantar o banco de dados vetorial na nuvem, bem como as etapas para implantar o banco de dados vetorial (milvus autônomo) no Amazon FSx ONTAP para NetApp ONTAP em contêineres docker.
3. "Proteção de banco de dados vetorial usando NetApp SnapCenter."Nesta seção, nos aprofundamos em como o SnapCenter protege os dados do banco de dados vetorial e os dados Milvus que residem no ONTAP. Neste exemplo, utilizamos um bucket NAS (milvusdbvol1) derivado de um volume NFS ONTAP (vol1) para dados do cliente e um volume NFS separado (vectordbpv) para dados de configuração do

cluster Milvus.

4. "Recuperação de desastres usando NetApp SnapMirror"Nesta seção, discutiremos a importância da recuperação de desastres (DR) para bancos de dados vetoriais e como o produto de recuperação de desastres Snapmirror da NetApp fornece uma solução de DR para bancos de dados vetoriais.
5. "Validação de desempenho"Nesta seção, pretendemos nos aprofundar na validação de desempenho de bancos de dados vetoriais, como Milvus e pgvecto.rs, com foco em suas características de desempenho de armazenamento, como perfil de E/S e comportamento do controlador de armazenamento netapp em suporte a cargas de trabalho de RAG e inferência dentro do ciclo de vida do LLM. Avaliaremos e identificaremos quaisquer diferenciais de desempenho quando esses bancos de dados forem combinados com a solução de armazenamento ONTAP . Nossa análise será baseada em indicadores-chave de desempenho, como o número de consultas processadas por segundo (QPS).

Configuração do cluster Milvus com Kubernetes no local

Esta seção discute a configuração do cluster milvus para a solução de banco de dados vetorial para NetApp.

Configuração do cluster Milvus com Kubernetes no local

Os desafios do cliente para escalar de forma independente em armazenamento e computação, gerenciamento eficaz de infraestrutura e gerenciamento de dados, Kubernetes e bancos de dados vetoriais juntos formam uma solução poderosa e escalável para gerenciar grandes operações de dados. O Kubernetes otimiza recursos e gerencia contêineres, enquanto bancos de dados vetoriais manipulam com eficiência dados de alta dimensão e pesquisas de similaridade. Essa combinação permite o processamento rápido de consultas complexas em grandes conjuntos de dados e dimensiona perfeitamente com volumes crescentes de dados, tornando-a ideal para aplicativos de big data e cargas de trabalho de IA.

1. Nesta seção, detalhamos o processo de instalação de um cluster Milvus no Kubernetes, utilizando um controlador de armazenamento NetApp para dados do cluster e dados do cliente.
2. Para instalar um cluster Milvus, Volumes Persistentes (PVs) são necessários para armazenar dados de vários componentes do cluster Milvus. Esses componentes incluem etcd (três instâncias), pulsar-bookie-journal (três instâncias), pulsar-bookie-ledgers (três instâncias) e pulsar-zookeeper-data (três instâncias).

> ⓘ No cluster Milvus, podemos usar o Pulsar ou o Kafka como mecanismo subjacente que dá suporte ao armazenamento confiável e à publicação/assinatura de fluxos de mensagens do cluster Milvus. Para o Kafka com NFS, a NetApp fez melhorias no ONTAP 9.12.1 e versões posteriores, e essas melhorias, juntamente com o NFSv4.1 e as alterações do Linux incluídas no RHEL 8.7 ou 9.1 e versões superiores, resolvem o problema de "renomeação absurda" que pode ocorrer ao executar o Kafka sobre NFS. Se você estiver interessado em informações mais detalhadas sobre o tópico de execução do Kafka com a solução NetApp NFS, consulte -"este link" .

3. Criamos um único volume NFS do NetApp ONTAP e estabelecemos 12 volumes persistentes, cada um com 250 GB de armazenamento. O tamanho do armazenamento pode variar dependendo do tamanho do cluster; por exemplo, temos outro cluster onde cada PV tem 50 GB. Consulte abaixo um dos arquivos PV YAML para obter mais detalhes; tínhamos 12 arquivos desse tipo no total. Em cada arquivo, o storageClassName é definido como 'padrão', e o armazenamento e o caminho são exclusivos para cada PV.

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

4. Execute o comando 'kubectl apply' para cada arquivo PV YAML para criar os Volumes Persistentes e, em seguida, verifique sua criação usando 'kubectl get pv'

```
root@node2:~# for i in $( seq 1 12 ); do kubectl apply -f
sai_nfs_to_default_pv$i.yaml; done
persistentvolume/karthik-pv1 created
persistentvolume/karthik-pv2 created
persistentvolume/karthik-pv3 created
persistentvolume/karthik-pv4 created
persistentvolume/karthik-pv5 created
persistentvolume/karthik-pv6 created
persistentvolume/karthik-pv7 created
persistentvolume/karthik-pv8 created
persistentvolume/karthik-pv9 created
persistentvolume/karthik-pv10 created
persistentvolume/karthik-pv11 created
persistentvolume/karthik-pv12 created
root@node2:~#
```

5. Para armazenar dados do cliente, a Milvus oferece suporte a soluções de armazenamento de objetos como MinIO, Azure Blob e S3. Neste guia, utilizamos o S3. As etapas a seguir se aplicam ao ONTAP S3 e ao armazenamento de objetos StorageGRID . Usamos o Helm para implantar o cluster Milvus. Baixe o arquivo de configuração, values.yaml, do local de download do Milvus. Consulte o apêndice para o arquivo values.yaml que usamos neste documento.
6. Certifique-se de que 'storageClass' esteja definido como 'padrão' em cada seção, incluindo aquelas para log, etcd, zookeeper e bookkeeper.
7. Na seção MinIO, desabilite o MinIO.
8. Crie um bucket NAS a partir do armazenamento de objetos ONTAP ou StorageGRID e inclua-os em um S3 externo com as credenciais de armazenamento de objetos.

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

9. Antes de criar o cluster Milvus, certifique-se de que o PersistentVolumeClaim (PVC) não tenha nenhum recurso preexistente.

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

10. Utilize o Helm e o arquivo de configuração values.yaml para instalar e iniciar o cluster Milvus.

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

11. Verifique o status dos PersistentVolumeClaims (PVCs).

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

12. Verifique o status dos pods.

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

Certifique-se de que o status dos pods seja 'em execução' e esteja funcionando conforme o esperado

13. Teste de gravação e leitura de dados no armazenamento de objetos Milvus e NetApp .
    - Grave dados usando o programa Python "prepare_data_netapp_new.py".

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

- Leia os dados usando o arquivo Python "verify_data_netapp.py".

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

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

=== Start Creating index IVF_FLAT  ===

=== Start loading                  ===

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

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

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

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

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

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

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

Com base na validação acima, a integração do Kubernetes com um banco de dados vetorial, conforme demonstrado por meio da implantação de um cluster Milvus no Kubernetes usando um controlador de armazenamento NetApp , oferece aos clientes uma solução robusta, escalável e eficiente para gerenciar operações de dados em larga escala. Essa configuração oferece aos clientes a capacidade de manipular dados de alta dimensão e executar consultas complexas de forma rápida e eficiente, tornando-a uma solução ideal para aplicativos de big data e cargas de trabalho de IA. O uso de Volumes Persistentes (PVs) para vários componentes de cluster, juntamente com a criação de um único volume NFS do NetApp ONTAP, garante a utilização ideal de recursos e o gerenciamento de

dados. O processo de verificação do status de PersistentVolumeClaims (PVCs) e pods, bem como o teste de leitura e gravação de dados, fornece aos clientes a garantia de operações de dados confiáveis e consistentes. O uso do armazenamento de objetos ONTAP ou StorageGRID para dados do cliente melhora ainda mais a acessibilidade e a segurança dos dados. No geral, essa configuração capacita os clientes com uma solução de gerenciamento de dados resiliente e de alto desempenho que pode ser facilmente dimensionada de acordo com suas crescentes necessidades de dados.

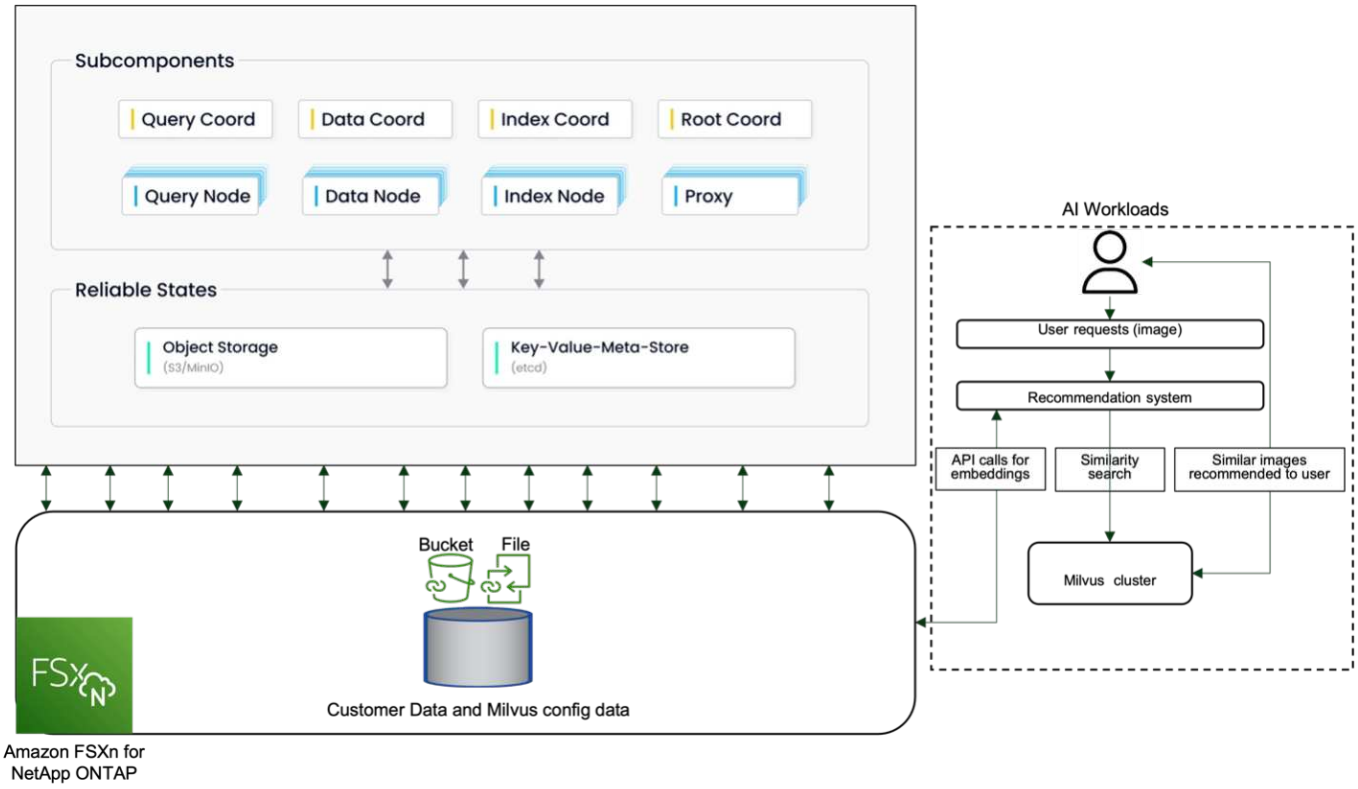
Milvus com Amazon FSx ONTAP para NetApp ONTAP - dualidade de arquivo e objeto

Esta seção discute a configuração do cluster milvus com o Amazon FSx ONTAP para a solução de banco de dados vetorial para NetApp.

Milvus com Amazon FSx ONTAP para NetApp ONTAP – dualidade de arquivo e objeto

Nesta seção, veremos por que precisamos implantar um banco de dados vetorial na nuvem, bem como as etapas para implantar um banco de dados vetorial (milvus autônomo) no Amazon FSx ONTAP para NetApp ONTAP em contêineres do Docker.

A implantação de um banco de dados vetorial na nuvem oferece vários benefícios significativos, especialmente para aplicativos que exigem manipulação de dados de alta dimensão e execução de pesquisas de similaridade. Primeiro, a implantação baseada em nuvem oferece escalabilidade, permitindo o fácil ajuste de recursos para corresponder aos crescentes volumes de dados e cargas de consulta. Isso garante que o banco de dados possa lidar eficientemente com o aumento da demanda, mantendo alto desempenho. Em segundo lugar, a implantação na nuvem oferece alta disponibilidade e recuperação de desastres, pois os dados podem ser replicados em diferentes localizações geográficas, minimizando o risco de perda de dados e garantindo serviço contínuo mesmo durante eventos inesperados. Terceiro, ele oferece boa relação custo-benefício, pois você só paga pelos recursos que usa e pode aumentar ou diminuir conforme a demanda, evitando a necessidade de investimentos iniciais substanciais em hardware. Por fim, a implantação de um banco de dados vetorial na nuvem pode melhorar a colaboração, pois os dados podem ser acessados e compartilhados de qualquer lugar, facilitando o trabalho em equipe e a tomada de decisões baseada em dados. Verifique a arquitetura do milvus standalone com Amazon FSx ONTAP para NetApp ONTAP usado nesta validação.



1. Crie uma instância do Amazon FSx ONTAP para NetApp ONTAP e anote os detalhes da VPC, dos grupos de segurança da VPC e da sub-rede. Essas informações serão necessárias ao criar uma instância do EC2. Você pode encontrar mais detalhes aqui - https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create
2. Crie uma instância do EC2, garantindo que a VPC, os grupos de segurança e a sub-rede correspondam aos da instância do Amazon FSx ONTAP para NetApp ONTAP .
3. Instale o nfs-common usando o comando 'apt-get install nfs-common' e atualize as informações do pacote usando 'sudo apt-get update'.
4. Crie uma pasta de montagem e monte o Amazon FSx ONTAP para NetApp ONTAP nela.

```
ubuntu@ip-172-31-29-98:~$ mkdir /home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$ sudo mount 172.31.255.228:/vol1
/home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$ df -h /home/ubuntu/milvusvectordb
Filesystem            Size  Used Avail Use% Mounted on
172.31.255.228:/vol1  973G  126G  848G  13% /home/ubuntu/milvusvectordb
ubuntu@ip-172-31-29-98:~$
```

5. Instale o Docker e o Docker Compose usando 'apt-get install'.
6. Configure um cluster Milvus com base no arquivo docker-compose.yaml, que pode ser baixado do site Milvus.

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

7. Na seção 'volumes' do arquivo docker-compose.yml, mapeie o ponto de montagem do NetApp NFS para o caminho do contêiner Milvus correspondente, especificamente em etcd, minio e standalone. Verifique"Apêndice D: docker-compose.yml" para detalhes sobre mudanças no yml
8. Verifique as pastas e arquivos montados.

```
ubuntu@ip-172-31-29-98:~/milvusvectordb$ ls -ltrh
/home/ubuntu/milvusvectordb
total 8.0K
-rw-r--r-- 1 root root 1.8K Apr  2 16:35 s3_access.py
drwxrwxrwx 2 root root 4.0K Apr  4 20:19 volumes
ubuntu@ip-172-31-29-98:~/milvusvectordb$ ls -ltrh
/home/ubuntu/milvusvectordb/volumes/
total 0
ubuntu@ip-172-31-29-98:~/milvusvectordb$ cd
ubuntu@ip-172-31-29-98:~$ ls
docker-compose.yml  docker-compose.yml~  milvus.yaml  milvusvectordb
vectordbvol1
ubuntu@ip-172-31-29-98:~$
```

9. Execute 'docker-compose up -d' no diretório que contém o arquivo docker-compose.yml.
10. Verifique o status do contêiner Milvus.

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

11. Para validar a funcionalidade de leitura e gravação do banco de dados vetorial e seus dados no Amazon FSx ONTAP para NetApp ONTAP, usamos o Python Milvus SDK e um programa de exemplo do PyMilvus. Instale os pacotes necessários usando 'apt-get install python3-numpy python3-pip' e instale o PyMilvus usando 'pip3 install pymilvus'.
12. Valide as operações de leitura e gravação de dados do Amazon FSx ONTAP para NetApp ONTAP no banco de dados vetorial.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3
prepare_data_netapp_new.py
=== start connecting to Milvus     ===
=== Milvus host: localhost         ===
Does collection hello_milvus_ntapnew_sc exist in Milvus: True
=== Drop collection - hello_milvus_ntapnew_sc ===
=== Drop collection - hello_milvus_ntapnew_sc2 ===
=== Create collection `hello_milvus_ntapnew_sc` ===
=== Start inserting entities       ===
Number of entities in hello_milvus_ntapnew_sc: 9000
root@ip-172-31-29-98:~/pymilvus/examples# find
/home/ubuntu/milvusvectordb/
…
<removed content to save page space >
…
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
```

```
/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100/448789845791411920
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
```

13. Verifique a operação de leitura usando o script verify_data_netapp.py.

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

=== Milvus host: localhost         ===

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

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

14. Se o cliente quiser acessar (ler) dados NFS testados no banco de dados vetorial por meio do protocolo S3 para cargas de trabalho de IA, isso pode ser validado usando um programa Python simples. Um exemplo disso poderia ser uma busca por similaridade de imagens de outro aplicativo, como mencionado na imagem que está no início desta seção.

```
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#
```

Esta seção demonstra efetivamente como os clientes podem implantar e operar uma configuração Milvus autônoma em contêineres Docker, utilizando o NetApp FSx ONTAP da Amazon para armazenamento de dados NetApp ONTAP . Essa configuração permite que os clientes aproveitem o poder dos bancos de dados vetoriais para manipular dados de alta dimensão e executar consultas complexas, tudo dentro do ambiente escalável e eficiente dos contêineres do Docker. Ao criar uma instância do Amazon FSx ONTAP para NetApp ONTAP e uma instância EC2 correspondente, os clientes podem garantir a utilização ideal de recursos e o gerenciamento de dados. A validação bem-sucedida das operações de leitura e gravação de dados do FSx ONTAP no banco de dados vetorial fornece aos clientes a garantia de operações de dados confiáveis e consistentes. Além disso, a capacidade de listar (ler) dados de cargas de trabalho de IA por meio do protocolo S3 oferece maior acessibilidade aos dados. Portanto, esse processo abrangente fornece aos clientes uma solução robusta e eficiente para gerenciar suas operações de dados em larga escala, aproveitando os recursos do FSx ONTAP da Amazon para NetApp ONTAP.

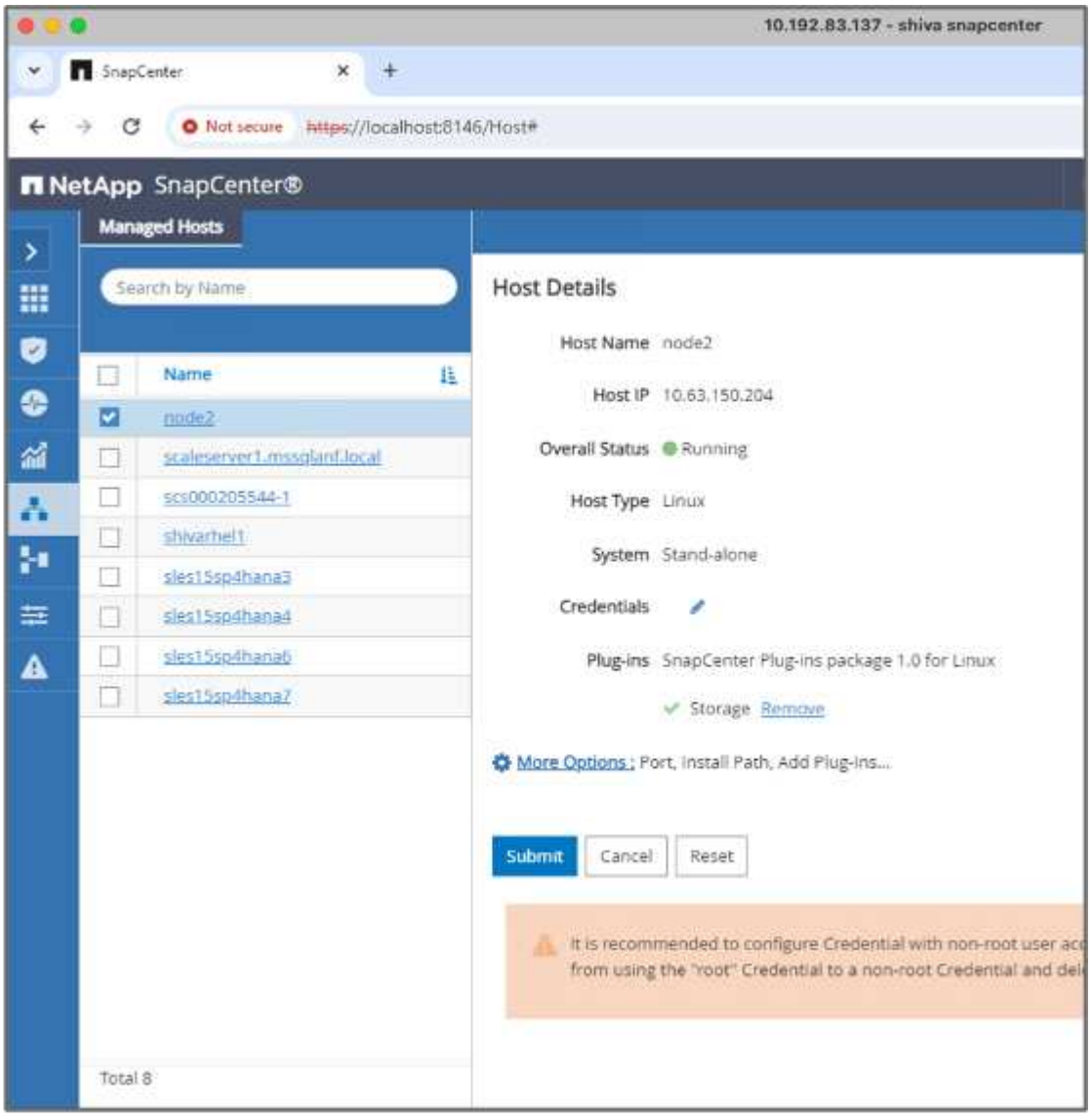
Proteção de banco de dados vetorial usando SnapCenter

Esta seção descreve como fornecer proteção de dados para o banco de dados de vetores usando o NetApp SnapCenter.

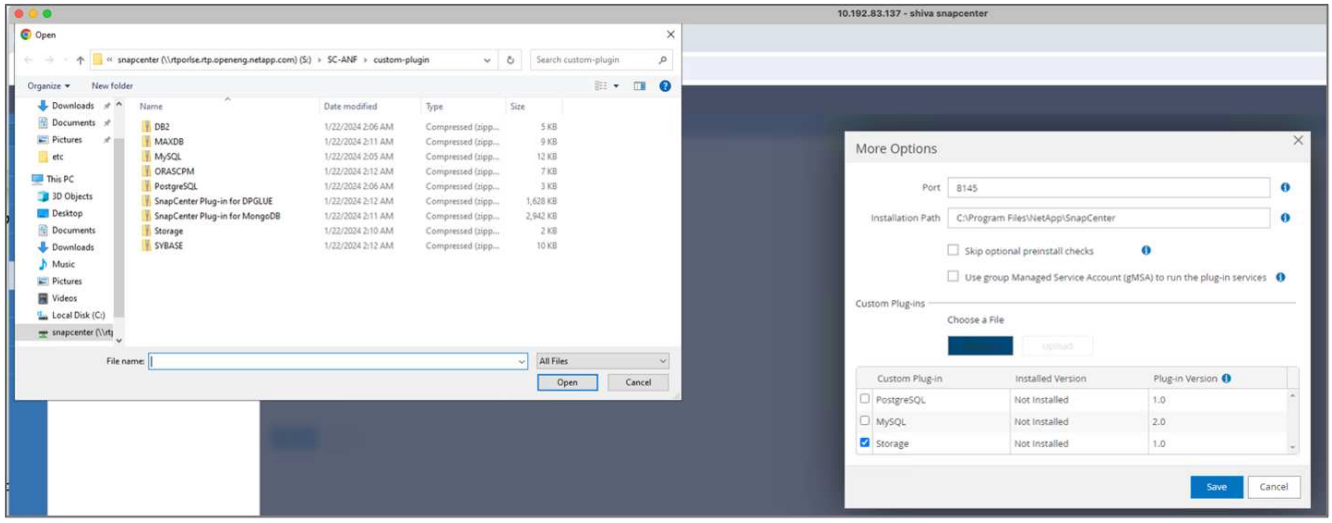
Proteção de banco de dados vetorial usando NetApp SnapCenter.

Por exemplo, na indústria de produção cinematográfica, os clientes geralmente possuem dados incorporados críticos, como arquivos de vídeo e áudio. A perda desses dados, devido a problemas como falhas no disco rígido, pode ter um impacto significativo em suas operações, potencialmente colocando em risco empreendimentos multimilionários. Encontramos casos em que conteúdo inestimável foi perdido, causando interrupções substanciais e perdas financeiras. Garantir a segurança e a integridade desses dados essenciais é, portanto, de suma importância neste setor. Nesta seção, nos aprofundamos em como o SnapCenter protege os dados do banco de dados vetorial e os dados Milvus que residem no ONTAP. Para este exemplo, utilizamos um bucket NAS (milvusdbvol1) derivado de um volume NFS ONTAP (vol1) para dados do cliente e um volume NFS separado (vectordbpv) para dados de configuração do cluster Milvus. verifique o"aqui" para o fluxo de trabalho de backup do snapcenter

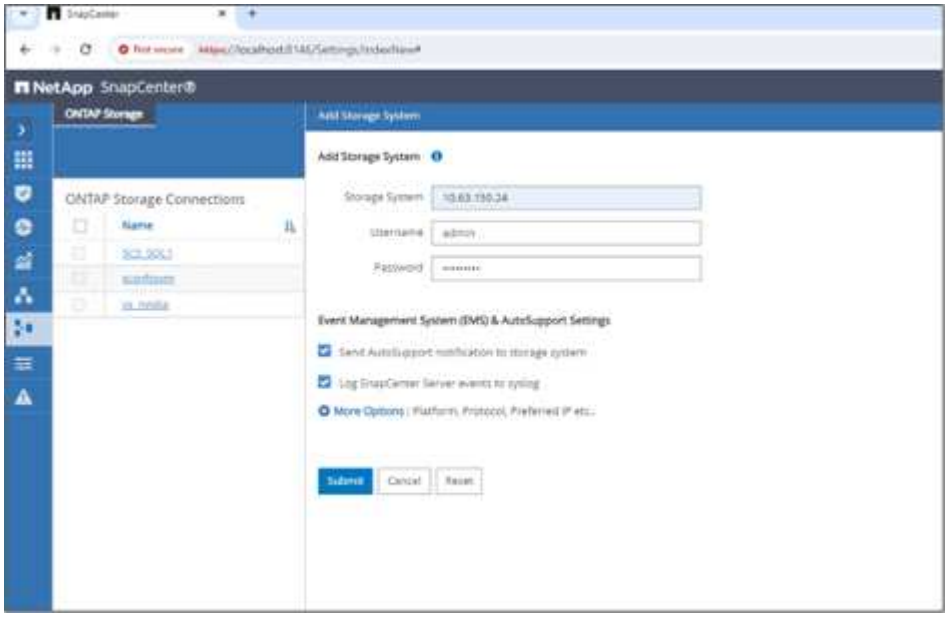
1. Configure o host que será usado para executar comandos do SnapCenter .



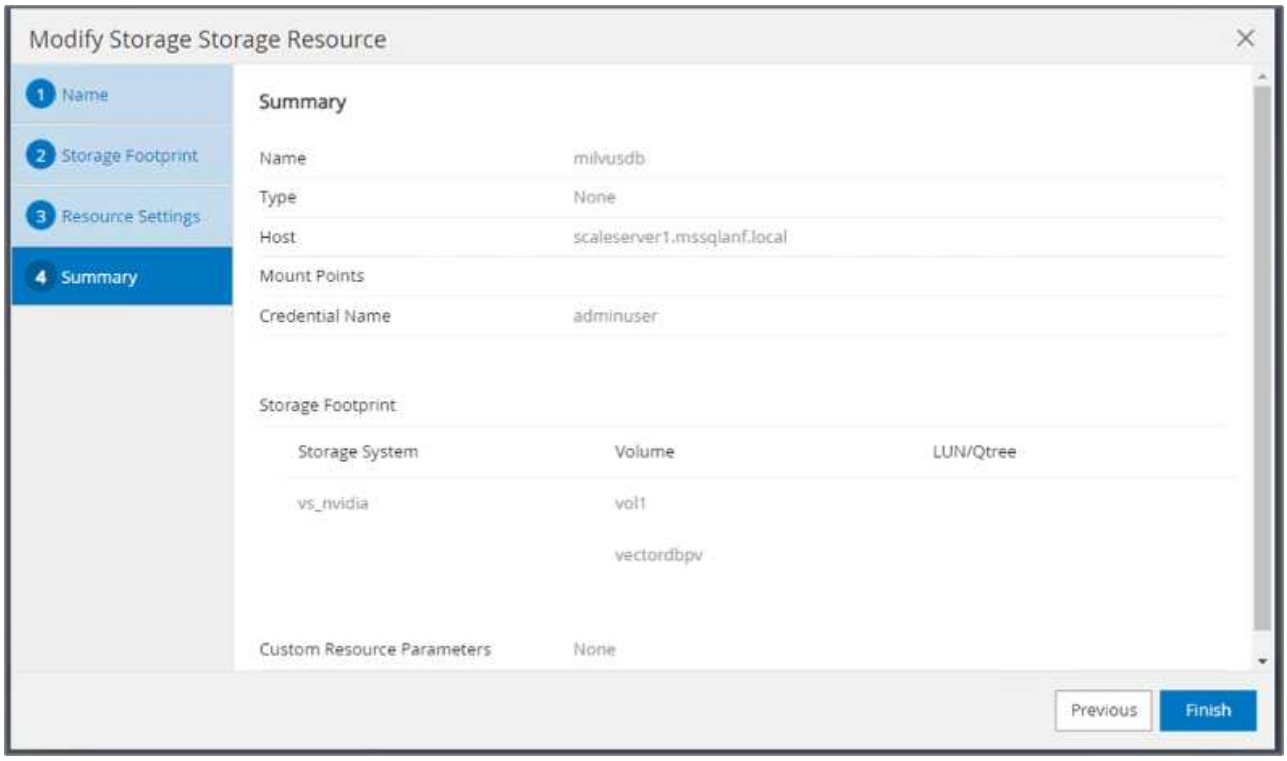
2. Instale e configure o plugin de armazenamento. No host adicionado, selecione "Mais opções". Navegue e selecione o plugin de armazenamento baixado em"Loja de automação NetApp" . Instale o plugin e salve a configuração.



3. Configure o sistema de armazenamento e o volume: adicione o sistema de armazenamento em "Sistema de armazenamento" e selecione a SVM (Máquina Virtual de Armazenamento). Neste exemplo, escolhemos "vs_nvidia".



4. Estabeleça um recurso para o banco de dados vetorial, incorporando uma política de backup e um nome de instantâneo personalizado.
   - Habilite o backup de grupo de consistência com valores padrão e habilite o SnapCenter sem consistência do sistema de arquivos.
   - Na seção Storage Footprint, selecione os volumes associados aos dados do cliente do banco de dados vetorial e aos dados do cluster Milvus. No nosso exemplo, são "vol1" e "vectordbpv".
   - Crie uma política para proteção de banco de dados de vetores e proteja os recursos do banco de dados de vetores usando a política.



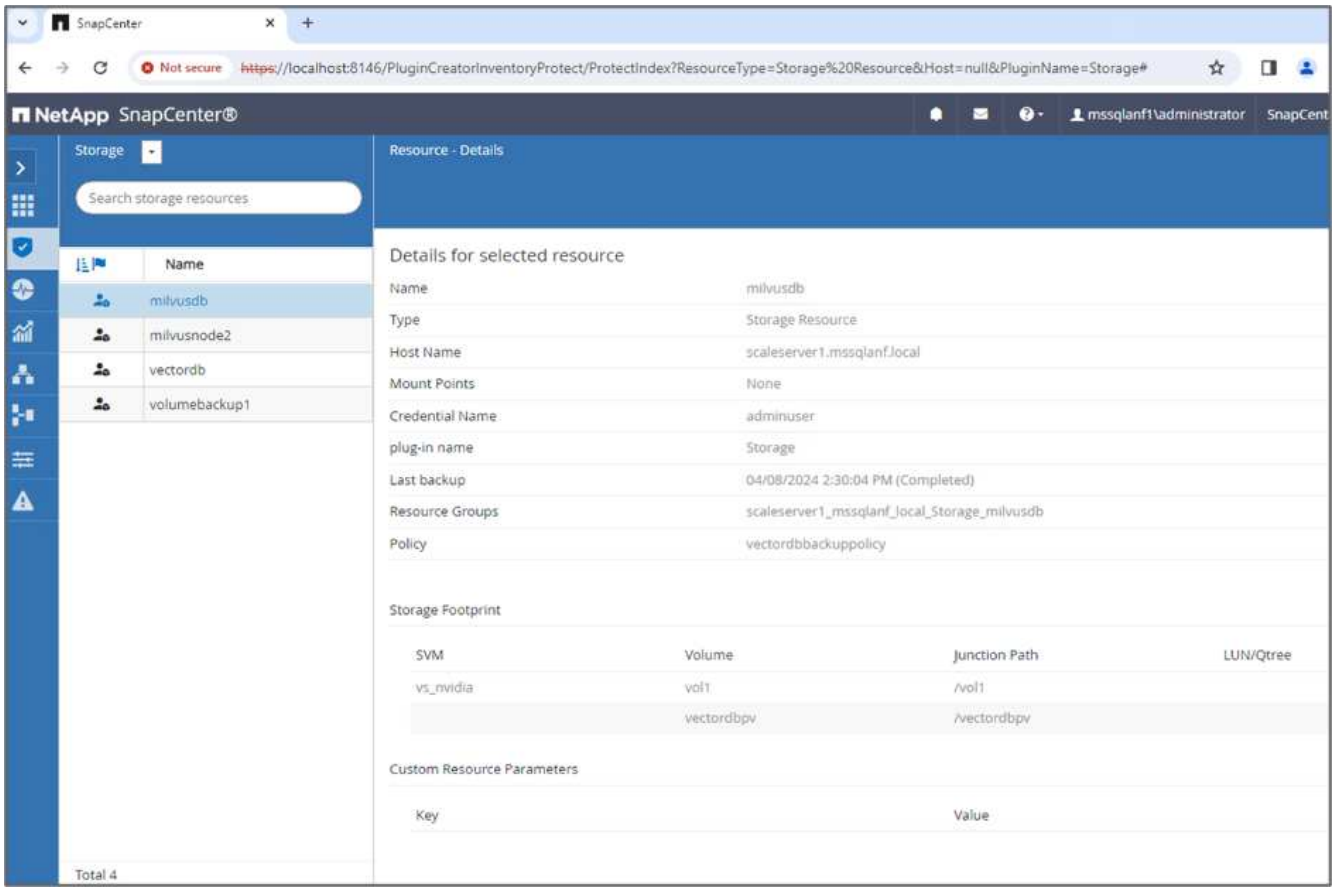
5. Insira dados no bucket S3 NAS usando um script Python. No nosso caso, modificamos o script de backup fornecido pelo Milvus, chamado 'prepare_data_netapp.py', e executamos o comando 'sync' para liberar os dados do sistema operacional.

```
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. Verifique os dados no bucket S3 NAS. Em nosso exemplo, os arquivos com o registro de data e hora '2024-04-08 21:22' foram criados pelo script 'prepare_data_netapp.py'.

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

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

7. Inicie um backup usando o snapshot do Consistency Group (CG) do recurso 'milvusdb'

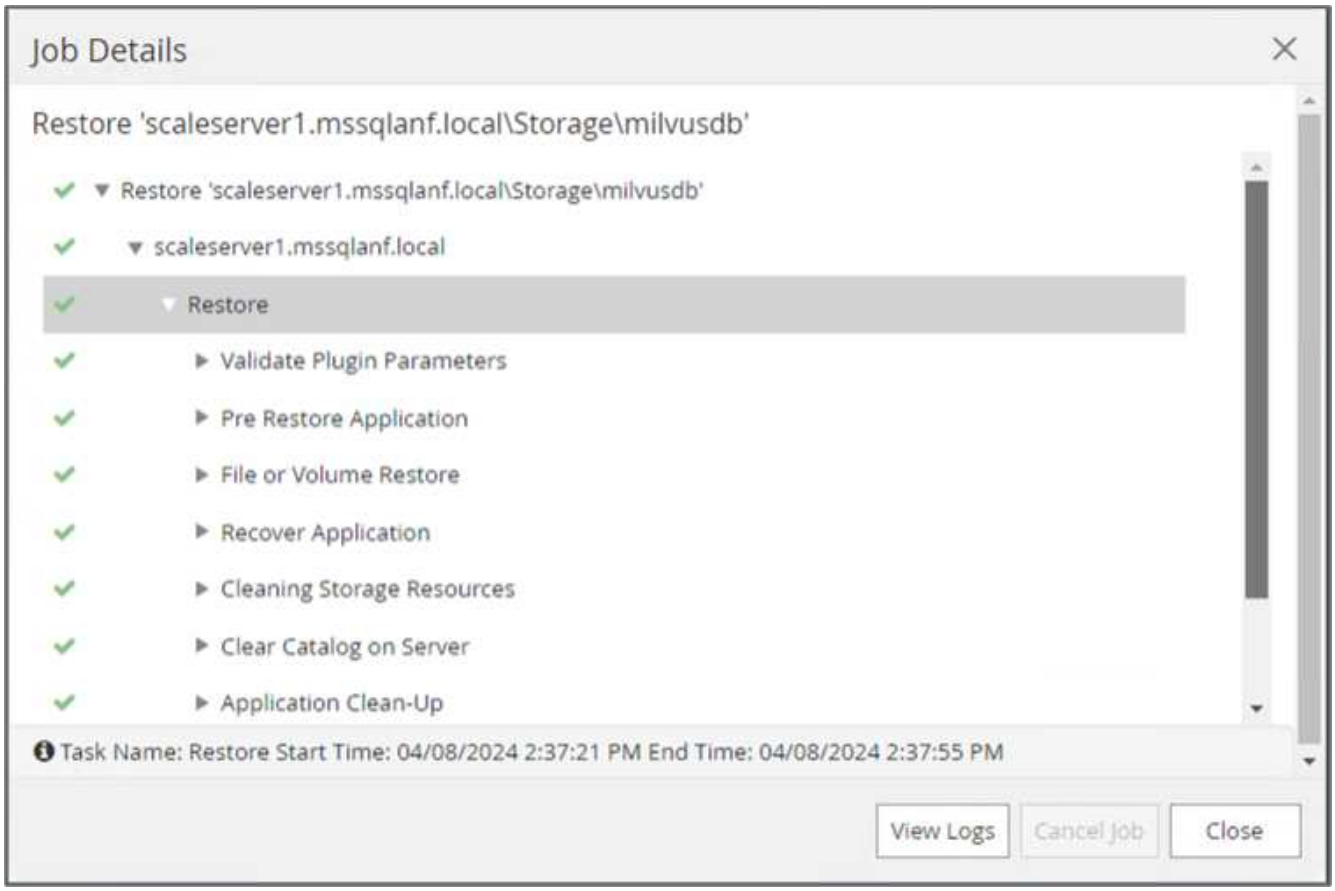


8. Para testar a funcionalidade de backup, adicionamos uma nova tabela após o processo de backup ou removemos alguns dados do NFS (bucket S3 NAS).

   Para este teste, imagine um cenário em que alguém criou uma coleção nova, desnecessária ou inadequada após o backup. Nesse caso, precisaríamos reverter o banco de dados de vetores ao estado anterior à adição da nova coleção. Por exemplo, novas coleções como 'hello_milvus_netapp_sc_testnew' e 'hello_milvus_netapp_sc_testnew2' foram inseridas.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_testnew` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

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

9. Execute uma restauração completa do bucket S3 NAS a partir do snapshot anterior.



10. Use um script Python para verificar os dados das coleções 'hello_milvus_netapp_sc_test' e 'hello_milvus_netapp_sc_test2'.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

11. Verifique se a coleção desnecessária ou inadequada não está mais presente no banco de dados.

```
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:~#
```

Concluindo, o uso do SnapCenter da NetApp para proteger dados de bancos de dados vetoriais e dados Milvus residentes no ONTAP oferece benefícios significativos aos clientes, especialmente em setores onde a integridade dos dados é primordial, como a produção de filmes. A capacidade do SnapCenter de criar backups consistentes e executar restaurações completas de dados garante que dados críticos, como arquivos de vídeo e áudio incorporados, sejam protegidos contra perdas devido a falhas no disco rígido ou outros problemas. Isso não apenas evita interrupções operacionais como também protege contra perdas financeiras substanciais.

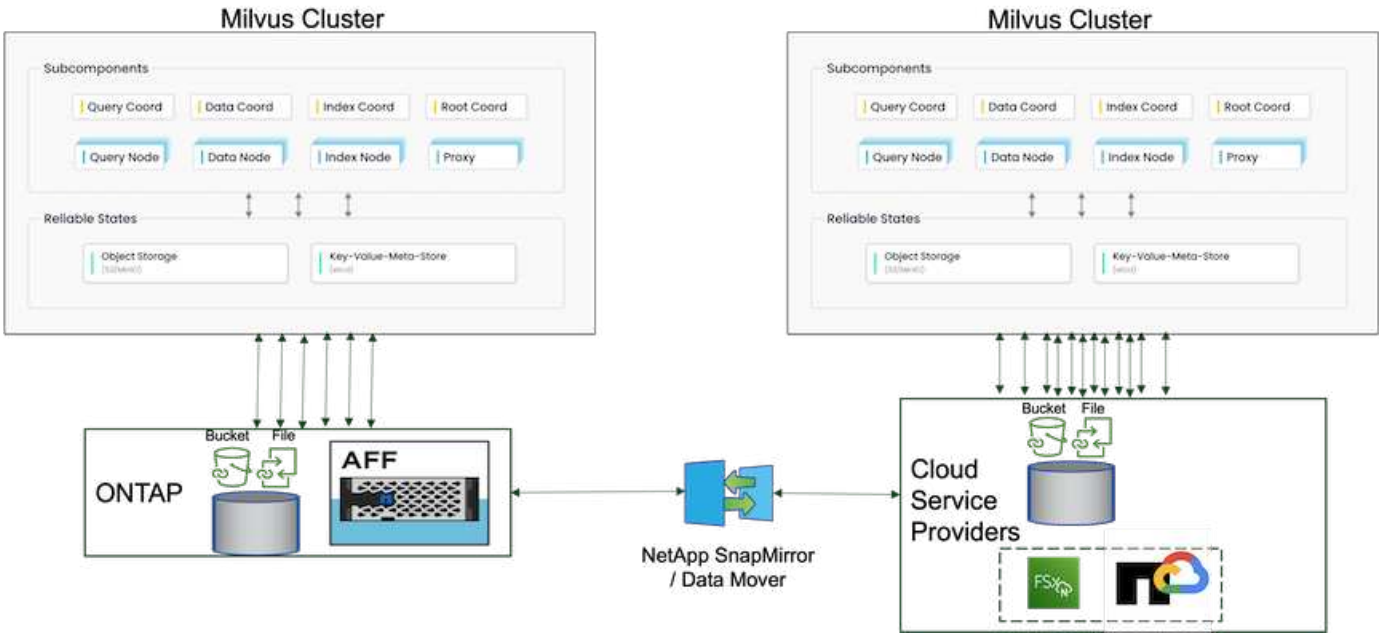
Nesta seção, demonstramos como o SnapCenter pode ser configurado para proteger dados residentes no ONTAP, incluindo a configuração de hosts, instalação e configuração de plug-ins de armazenamento e a criação de um recurso para o banco de dados de vetores com um nome de instantâneo personalizado. Também mostramos como executar um backup usando o snapshot do Consistency Group e verificar os dados no bucket S3 NAS.

Além disso, simulamos um cenário em que uma coleção desnecessária ou inadequada foi criada após o backup. Nesses casos, a capacidade do SnapCenter de executar uma restauração completa de um instantâneo anterior garante que o banco de dados de vetores possa ser revertido ao seu estado anterior à adição da nova coleção, mantendo assim a integridade do banco de dados. Essa capacidade de restaurar dados para um ponto específico no tempo é inestimável para os clientes, fornecendo a eles a garantia de que seus dados não estão apenas seguros, mas também mantidos corretamente. Assim, o produto SnapCenter da NetApp oferece aos clientes uma solução robusta e confiável para proteção e gerenciamento de dados.

Recuperação de desastres usando NetApp SnapMirror

Esta seção discute DR (recuperação de desastres) com SnapMirror para a solução de banco de dados vetorial da NetApp.

Recuperação de desastres usando NetApp SnapMirror



A recuperação de desastres é crucial para manter a integridade e a disponibilidade de um banco de dados vetorial, especialmente devido ao seu papel no gerenciamento de dados de alta dimensão e na execução de pesquisas complexas de similaridade. Uma estratégia de recuperação de desastres bem planejada e implementada garante que os dados não sejam perdidos ou comprometidos em caso de incidentes imprevistos, como falhas de hardware, desastres naturais ou ataques cibernéticos. Isso é particularmente significativo para aplicativos que dependem de bancos de dados vetoriais, onde a perda ou corrupção de dados pode levar a interrupções operacionais significativas e perdas financeiras. Além disso, um plano robusto de recuperação de desastres também garante a continuidade dos negócios, minimizando o tempo de inatividade e permitindo a rápida restauração dos serviços. Isso é obtido por meio do produto de replicação de dados SnapMirror da NetApp em diferentes localizações geográficas, backups regulares e mecanismos de failover. Portanto, a recuperação de desastres não é apenas uma medida de proteção, mas um componente crítico do gerenciamento responsável e eficiente de bancos de dados de vetores.

O SnapMirror da NetApp fornece replicação de dados de um controlador de armazenamento NetApp ONTAP para outro, usado principalmente para recuperação de desastres (DR) e soluções híbridas. No contexto de um banco de dados vetorial, esta ferramenta facilita a transição suave de dados entre ambientes locais e na nuvem. Essa transição é alcançada sem a necessidade de conversões de dados ou refatoração de aplicativos, aumentando assim a eficiência e a flexibilidade do gerenciamento de dados em diversas plataformas.

A solução híbrida da NetApp em um cenário de banco de dados vetorial pode trazer mais vantagens:

1. Escalabilidade: a solução de nuvem híbrida da NetApp oferece a capacidade de dimensionar seus recursos conforme suas necessidades. Você pode utilizar recursos locais para cargas de trabalho regulares e previsíveis e recursos de nuvem, como o Amazon FSx ONTAP para NetApp ONTAP e o Google Cloud NetApp Volume (NetApp Volumes) para horários de pico ou cargas inesperadas.
2. Eficiência de custos: o modelo de nuvem híbrida da NetApp permite que você otimize seus custos usando recursos locais para cargas de trabalho regulares e pagando pelos recursos de nuvem somente quando precisar deles. Esse modelo de pagamento conforme o uso pode ser bastante econômico com uma oferta de serviço NetApp instaclustr. Para provedores de serviços de nuvem locais e principais provedores, a instaclustr fornece suporte e consultoria.
3. Flexibilidade: a nuvem híbrida da NetApp oferece a flexibilidade de escolher onde processar seus dados.

Por exemplo, você pode optar por executar operações vetoriais complexas no local, onde você tem hardware mais potente, e operações menos intensivas na nuvem.

4. Continuidade dos negócios: em caso de desastre, ter seus dados em uma nuvem híbrida da NetApp pode garantir a continuidade dos negócios. Você pode mudar rapidamente para a nuvem se seus recursos locais forem afetados. Podemos aproveitar o NetApp SnapMirror para mover os dados do local para a nuvem e vice-versa.
5. Inovação: as soluções de nuvem híbrida da NetApp também podem permitir inovação mais rápida, fornecendo acesso a serviços e tecnologias de nuvem de ponta. As inovações da NetApp na nuvem, como o Amazon FSx ONTAP para NetApp ONTAP, o Azure NetApp Files e o Google Cloud NetApp Volumes, são produtos inovadores e NAS preferidos dos provedores de serviços de nuvem.

Validação de desempenho do banco de dados vetorial

Esta seção destaca a validação de desempenho que foi realizada no banco de dados de vetores.

Validação de desempenho

A validação de desempenho desempenha um papel fundamental tanto em bancos de dados vetoriais quanto em sistemas de armazenamento, servindo como um fator essencial para garantir a operação ideal e a utilização eficiente de recursos. Bancos de dados vetoriais, conhecidos por manipular dados de alta dimensão e executar pesquisas de similaridade, precisam manter altos níveis de desempenho para processar consultas complexas de forma rápida e precisa. A validação de desempenho ajuda a identificar gargalos, ajustar configurações e garantir que o sistema possa lidar com as cargas esperadas sem degradação do serviço. Da mesma forma, em sistemas de armazenamento, a validação de desempenho é essencial para garantir que os dados sejam armazenados e recuperados de forma eficiente, sem problemas de latência ou gargalos que possam afetar o desempenho geral do sistema. Ele também auxilia na tomada de decisões informadas sobre atualizações ou mudanças necessárias na infraestrutura de armazenamento. Portanto, a validação de desempenho é um aspecto crucial do gerenciamento de sistemas, contribuindo significativamente para manter a alta qualidade do serviço, a eficiência operacional e a confiabilidade geral do sistema.

Nesta seção, pretendemos nos aprofundar na validação de desempenho de bancos de dados vetoriais, como Milvus e pgvecto.rs, com foco em suas características de desempenho de armazenamento, como perfil de E/S e comportamento do controlador de armazenamento netapp em suporte a cargas de trabalho de RAG e inferência dentro do ciclo de vida do LLM. Avaliaremos e identificaremos quaisquer diferenciais de desempenho quando esses bancos de dados forem combinados com a solução de armazenamento ONTAP . Nossa análise será baseada em indicadores-chave de desempenho, como o número de consultas processadas por segundo (QPS).

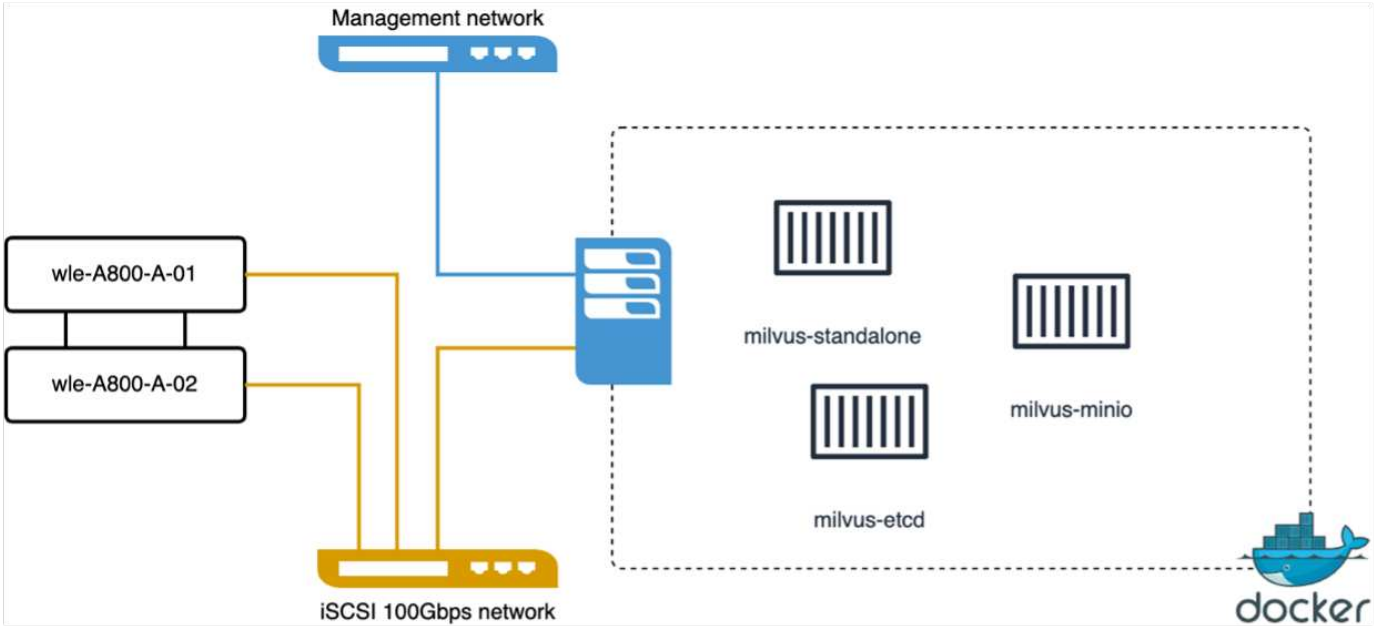
Confira abaixo a metodologia utilizada para o milvus e o progresso.

Detalhes	Milvus (autônomo e cluster)	Postgres(pgvecto.rs) #
versão	2.3.2	0.2.0
Sistema de arquivos	XFS em LUNs iSCSI	
Gerador de carga de trabalho	"Banco VectorDB"– v0.0.5	
Conjuntos de dados	Conjunto de dados LAION * 10 milhões de incorporações * 768 dimensões * tamanho do conjunto de dados de ~300 GB	

Controlador de armazenamento	AFF 800 * Versão – 9.14.1 * 4 x 100GbE – para milvus e 2x 100GbE para postgres * iscsi	

VectorDB-Bench com cluster autônomo Milvus

Fizemos a seguinte validação de desempenho no cluster autônomo milvus com o vectorDB-Bench. A conectividade de rede e servidor do cluster autônomo milvus está abaixo.



Nesta seção, compartilhamos nossas observações e resultados dos testes do banco de dados independente Milvus. . Selecionamos DiskANN como o tipo de índice para esses testes. . A ingestão, otimização e criação de índices para um conjunto de dados de aproximadamente 100 GB levou cerca de 5 horas. Durante a maior parte desse período, o servidor Milvus, equipado com 20 núcleos (o que equivale a 40 vcpus quando o Hyper-Threading está habilitado), estava operando em sua capacidade máxima de CPU de 100%. Descobrimos que o DiskANN é particularmente importante para grandes conjuntos de dados que excedem o tamanho da memória do sistema. . Na fase de consulta, observamos uma taxa de Consultas por Segundo (QPS) de 10,93 com um recall de 0,9987. A latência do 99º percentil para consultas foi medida em 708,2 milissegundos.

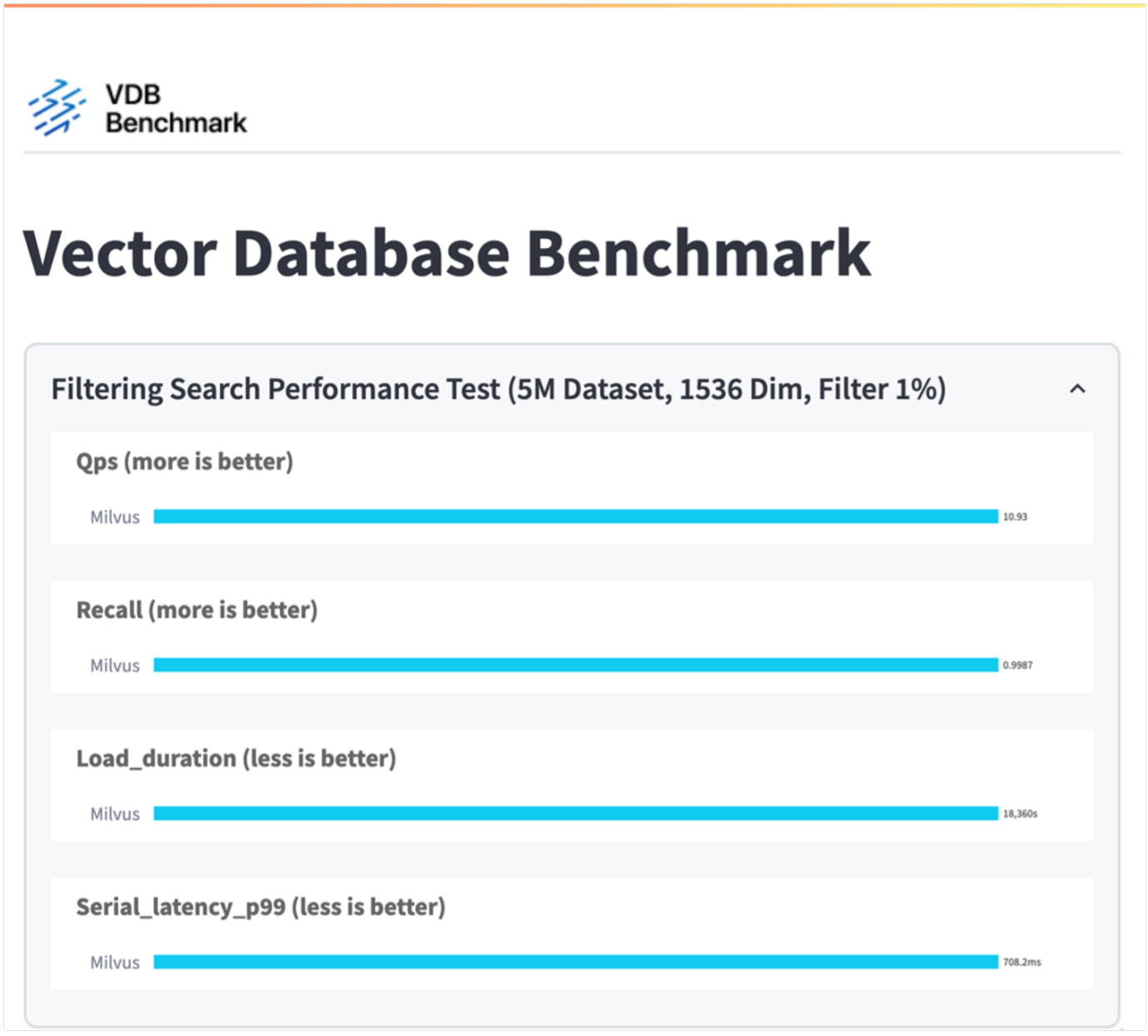
Da perspectiva de armazenamento, o banco de dados emitiu cerca de 1.000 ops/s durante as fases de ingestão, otimização pós-inserção e criação de índice. Na fase de consulta, foram necessárias 32.000 ops/seg.

A seção a seguir apresenta as métricas de desempenho de armazenamento.

Fase de carga de trabalho	Métrica	Valor
Ingestão de dados e otimização pós-inserção	IOPS	< 1.000
	Latência	< 400 usecs
	Carga de trabalho	Mistura de leitura/escrita, principalmente escrita
	Tamanho de E/S	64 KB

Fase de carga de trabalho	Métrica	Valor
Consulta	IOPS	Pico em 32.000
	Latência	< 400 usecs
	Carga de trabalho	100% de leitura em cache
	Tamanho de E/S	Principalmente 8 KB

O resultado do vectorDB-bench está abaixo.

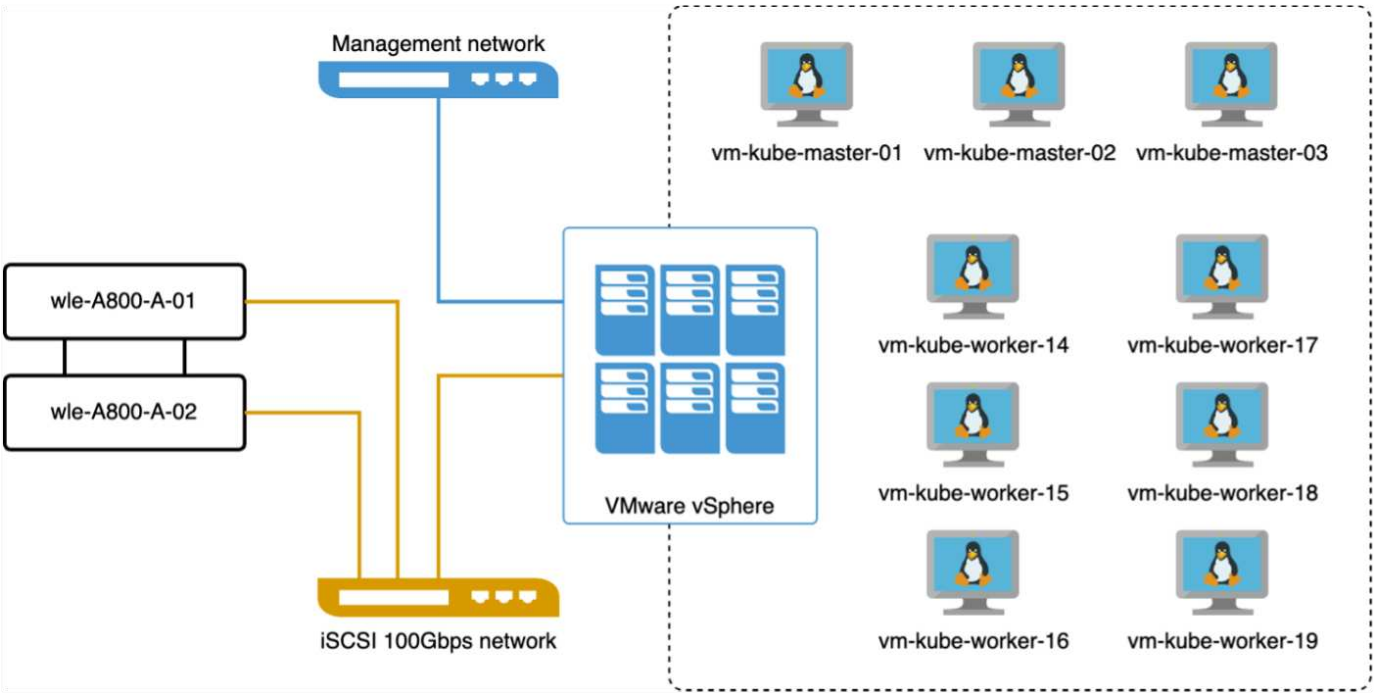


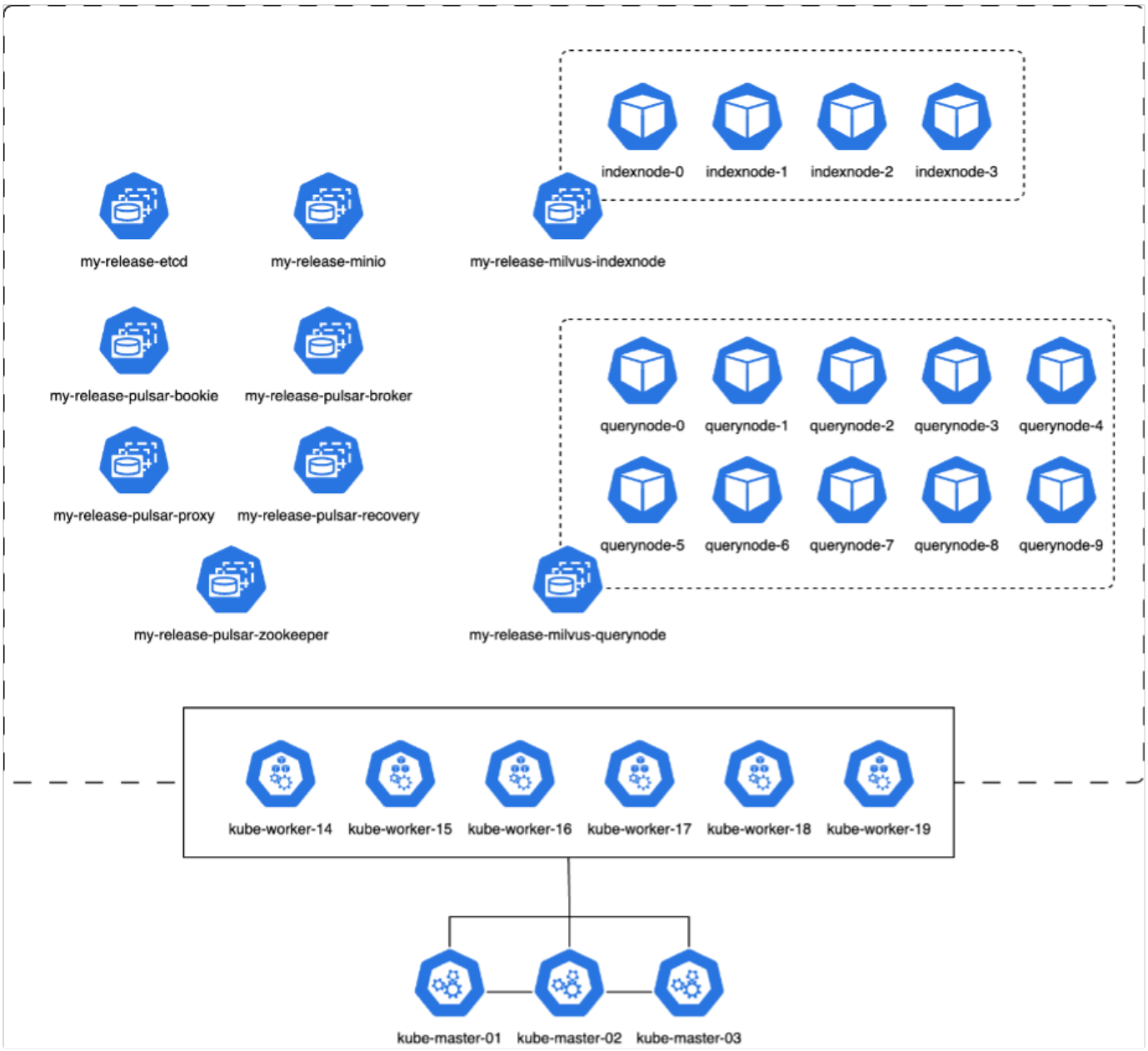
A partir da validação de desempenho da instância autônoma do Milvus, fica evidente que a configuração atual é insuficiente para suportar um conjunto de dados de 5 milhões de vetores com uma dimensionalidade de 1536. Determinamos que o armazenamento possui recursos adequados e não constitui um gargalo no sistema.

VectorDB-Bench com cluster milvus

Nesta seção, discutimos a implantação de um cluster Milvus em um ambiente Kubernetes. Esta configuração do Kubernetes foi construída sobre uma implantação do VMware vSphere, que hospedava os nós mestre e de trabalho do Kubernetes.

Os detalhes das implantações do VMware vSphere e do Kubernetes são apresentados nas seções a seguir.





Nesta seção, apresentamos nossas observações e resultados dos testes do banco de dados Milvus. * O tipo de índice usado foi DiskANN. * A tabela abaixo fornece uma comparação entre as implantações autônomas e em cluster ao trabalhar com 5 milhões de vetores em uma dimensionalidade de 1536. Observamos que o tempo necessário para ingestão de dados e otimização pós-inserção foi menor na implantação do cluster. A latência do 99º percentil para consultas foi reduzida em seis vezes na implantação do cluster em comparação à configuração autônoma. * Embora a taxa de Consultas por Segundo (QPS) tenha sido maior na implantação do cluster, ela não estava no nível desejado.

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%

As imagens abaixo fornecem uma visão de várias métricas de armazenamento, incluindo latência do cluster de armazenamento e IOPS (operações de entrada/saída por segundo) total.

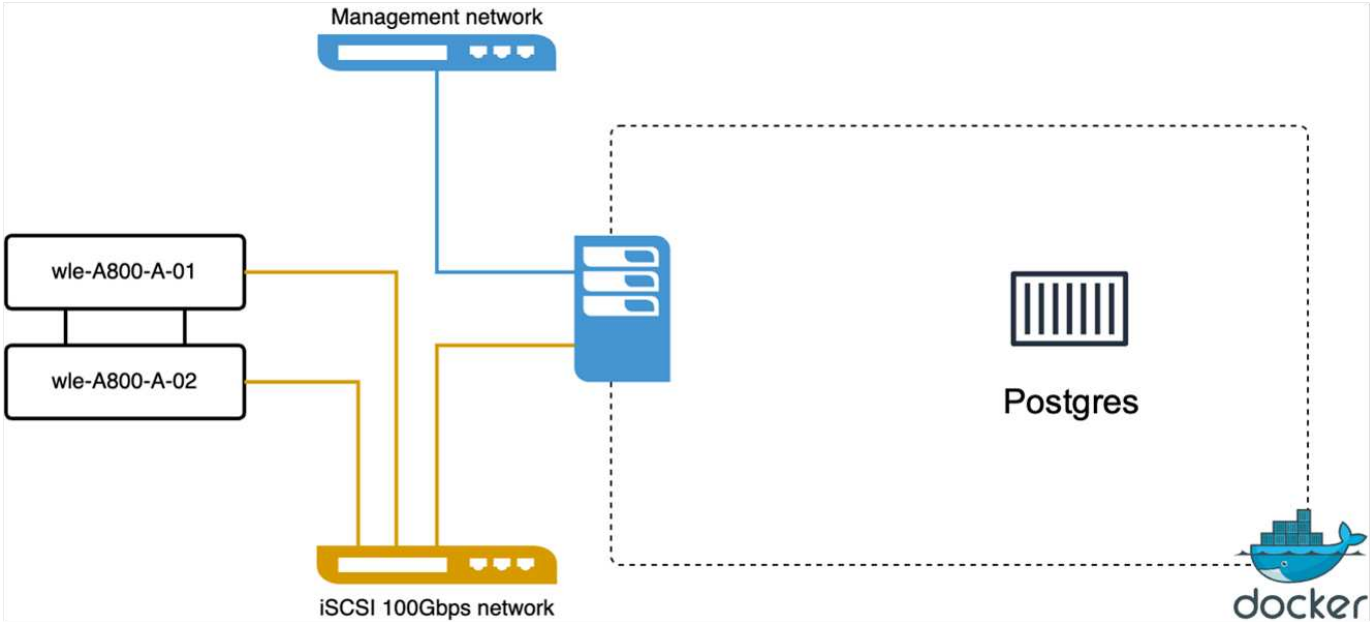
A seção a seguir apresenta as principais métricas de desempenho de armazenamento.

Fase de carga de trabalho	Métrica	Valor
Ingestão de dados e otimização pós-inserção	IOPS	< 1.000
	Latência	< 400 usecs
	Carga de trabalho	Mistura de leitura/escrita, principalmente escrita
	Tamanho de E/S	64 KB
Consulta	IOPS	Pico em 147.000
	Latência	< 400 usecs
	Carga de trabalho	100% de leitura em cache
	Tamanho de E/S	Principalmente 8 KB

Com base na validação de desempenho do Milvus autônomo e do cluster Milvus, apresentamos os detalhes do perfil de E/S de armazenamento. * Observamos que o perfil de E/S permanece consistente em implantações autônomas e em cluster. * A diferença observada no pico de IOPS pode ser atribuída ao maior número de clientes na implantação do cluster.

vectorDB-Bench com Postgres (pgvecto.rs)

Realizamos as seguintes ações no PostgreSQL (pgvecto.rs) usando o VectorDB-Bench: Os detalhes sobre a conectividade de rede e servidor do PostgreSQL (especificamente, pgvecto.rs) são os seguintes:

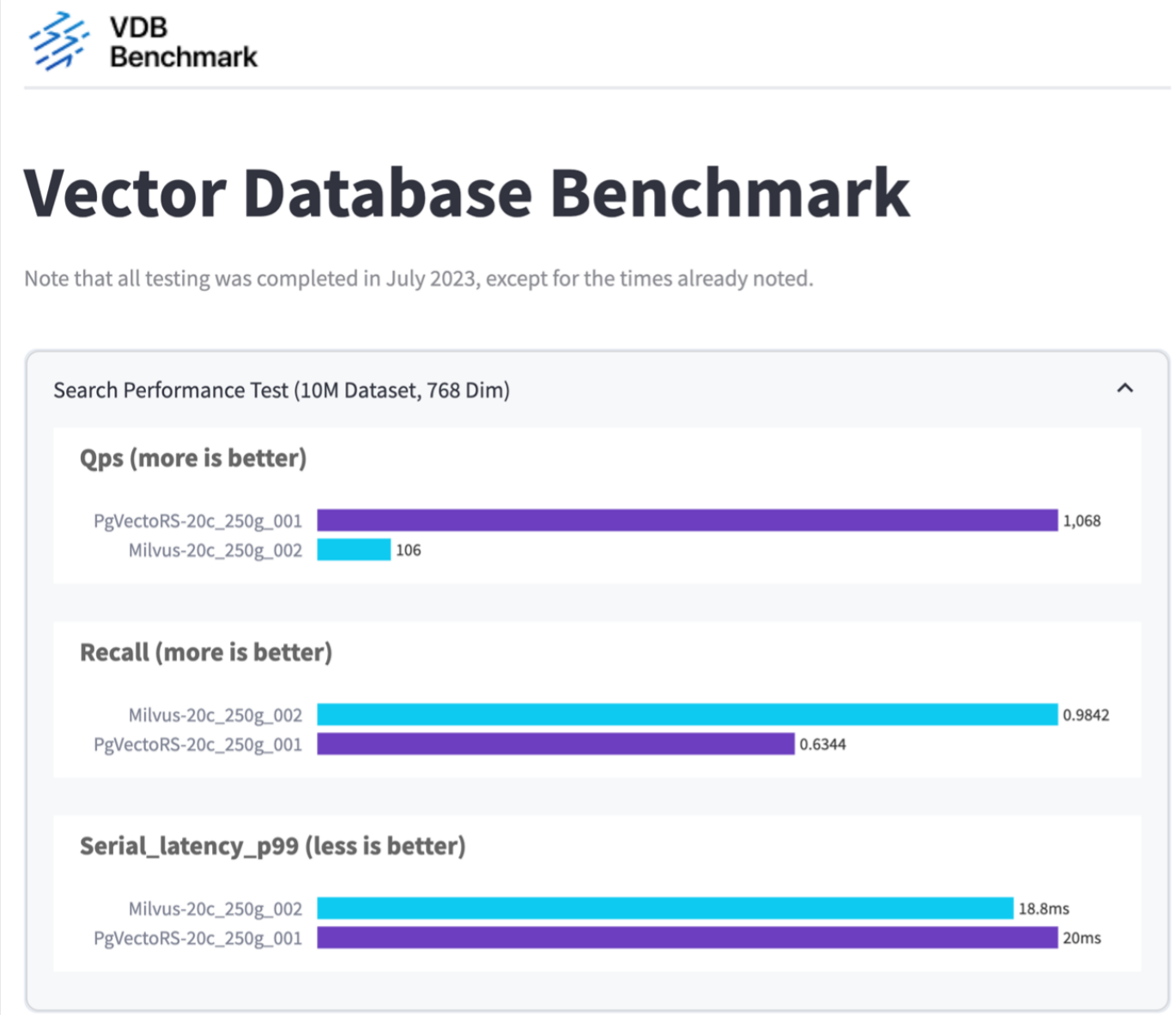


Nesta seção, compartilhamos nossas observações e resultados dos testes do banco de dados PostgreSQL, especificamente usando pgvecto.rs. * Selecionamos HNSW como o tipo de índice para esses testes porque, no momento do teste, o DiskANN não estava disponível para pgvecto.rs. * Durante a fase de ingestão de dados, carregamos o conjunto de dados Cohere, que consiste em 10 milhões de vetores com uma dimensionalidade de 768. Esse processo levou aproximadamente 4,5 horas. * Na fase de consulta, observamos uma taxa de Consultas por Segundo (QPS) de 1.068 com um recall de 0,6344. A latência do 99º percentil para consultas foi medida em 20 milissegundos. Durante a maior parte do tempo de execução, a CPU do cliente estava operando a 100% da capacidade.

As imagens abaixo fornecem uma visão de várias métricas de armazenamento, incluindo latência total do cluster de armazenamento (IOPS) (operações de entrada/saída por segundo).


```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-metrics.png["Figura mostrando diálogo de
entrada/saída ou representando conteúdo escrito"]
```

Comparação de desempenho entre milvus e postgres no banco de dados vetorial Bench



Com base em nossa validação de desempenho do Milvus e PostgreSQL usando VectorDBBench, observamos o seguinte:

- Tipo de índice: HNSW
- Conjunto de dados: Cohere com 10 milhões de vetores em 768 dimensões

Descobrimos que pgvecto.rs atingiu uma taxa de Consultas por Segundo (QPS) de 1.068 com um recall de 0,6344, enquanto Milvus atingiu uma taxa de QPS de 106 com um recall de 0,9842.

Se alta precisão em suas consultas for uma prioridade, o Milvus supera o pgvecto.rs, pois recupera uma proporção maior de itens relevantes por consulta. Entretanto, se o número de consultas por segundo for um fator mais crucial, pgvecto.rs supera Milvus. É importante observar, porém, que a qualidade dos dados recuperados via pgvecto.rs é inferior, com cerca de 37% dos resultados da pesquisa sendo itens irrelevantes.

Observação baseada em nossas validações de desempenho:

Com base em nossas validações de desempenho, fizemos as seguintes observações:

No Milvus, o perfil de E/S se assemelha muito a uma carga de trabalho OLTP, como a vista com o Oracle

SLOB. O benchmark consiste em três fases: ingestão de dados, pós-otimização e consulta. Os estágios iniciais são caracterizados principalmente por operações de gravação de 64 KB, enquanto a fase de consulta envolve predominantemente leituras de 8 KB. Esperamos que o ONTAP lide com a carga de E/S do Milvus com eficiência.

O perfil de E/S do PostgreSQL não apresenta uma carga de trabalho de armazenamento desafiadora. Dada a implementação na memória atualmente em andamento, não observamos nenhuma E/S de disco durante a fase de consulta.

DiskANN surge como uma tecnologia crucial para diferenciação de armazenamento. Ele permite o dimensionamento eficiente da pesquisa de banco de dados vetorial além do limite de memória do sistema. No entanto, é improvável que seja possível estabelecer diferenciação de desempenho de armazenamento com índices de banco de dados vetoriais na memória, como HNSW.

Também vale a pena notar que o armazenamento não desempenha um papel crítico durante a fase de consulta quando o tipo de índice é HSNW, que é a fase operacional mais importante para bancos de dados vetoriais que dão suporte a aplicativos RAG. A implicação aqui é que o desempenho do armazenamento não afeta significativamente o desempenho geral desses aplicativos.

Banco de dados vetorial com Instaclustr usando PostgreSQL: pgvector

Esta seção discute os detalhes específicos de como o produto instaclustr se integra com o PostgreSQL na funcionalidade pgvector na solução de banco de dados vetorial para NetApp.

Banco de dados vetorial com Instaclustr usando PostgreSQL: pgvector

Nesta seção, nos aprofundamos nos detalhes de como o produto instaclustr se integra com o PostgreSQL na funcionalidade do pgvector. Temos um exemplo de "Como melhorar a precisão e o desempenho do seu LLM com PGVector e PostgreSQL: Introdução aos embeddings e ao papel do PGVector". Por favor, verifique o"blog" para obter mais informações.

Casos de uso de banco de dados vetorial

Esta seção fornece uma visão geral dos casos de uso para a solução de banco de dados vetorial da NetApp .

Casos de uso de banco de dados vetorial

Nesta seção, discutimos dois casos de uso, como Geração Aumentada de Recuperação com Grandes Modelos de Linguagem e chatbot de TI da NetApp .

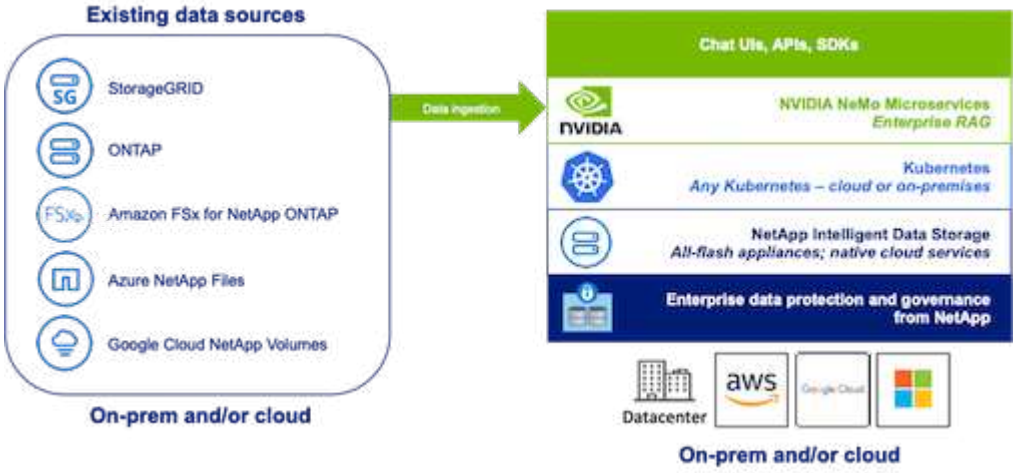
Geração Aumentada de Recuperação (RAG) com Grandes Modelos de Linguagem (LLMs)

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

O NVIDIA Enterprise RAG LLM Operator é uma ferramenta útil para implementar o RAG na empresa. Este operador pode ser usado para implantar um pipeline RAG completo. O pipeline RAG pode ser personalizado para utilizar Milvus ou pgvecto como banco de dados vetorial para armazenar incorporações de base de conhecimento. Consulte a documentação para obter detalhes.

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

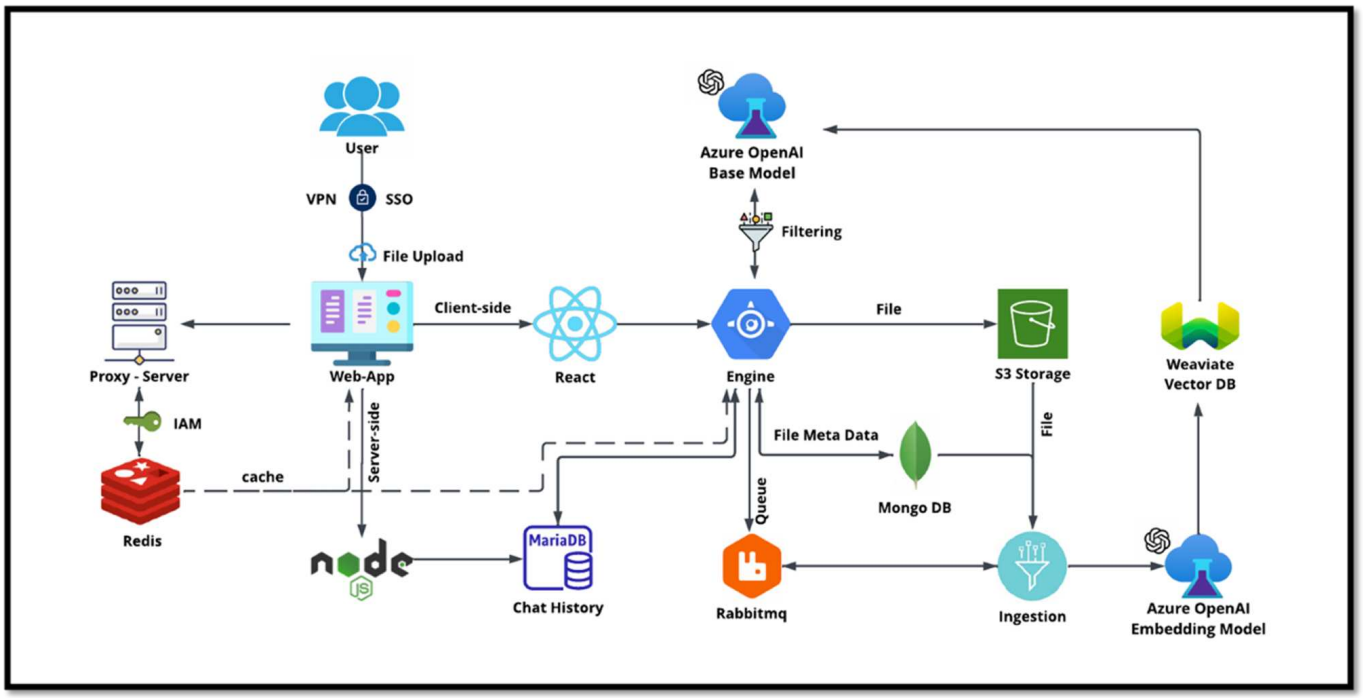
Figura 1) Enterprise RAG com tecnologia NVIDIA NeMo Microservices e NetApp



Caso de uso do chatbot de TI da NetApp

O chatbot da NetApp serve como outro caso de uso em tempo real para o banco de dados vetorial. Neste caso, o NetApp Private OpenAI Sandbox fornece uma plataforma eficaz, segura e eficiente para gerenciar consultas de usuários internos da NetApp. Ao incorporar protocolos de segurança rigorosos, sistemas eficientes de gerenciamento de dados e recursos sofisticados de processamento de IA, ele garante respostas precisas e de alta qualidade aos usuários com base em suas funções e responsabilidades na organização por meio de autenticação SSO. Esta arquitetura destaca o potencial de mesclar tecnologias avançadas para criar

sistemas inteligentes focados no usuário.



O caso de uso pode ser dividido em quatro seções principais.

Autenticação e verificação do usuário:

- As consultas do usuário passam primeiro pelo processo de logon único (SSO) da NetApp para confirmar a identidade do usuário.
- Após a autenticação bem-sucedida, o sistema verifica a conexão VPN para garantir uma transmissão segura de dados.

Transmissão e processamento de dados:

- Depois que a VPN é validada, os dados são enviados ao MariaDB por meio dos aplicativos web NetAIChat ou NetAICreate. MariaDB é um sistema de banco de dados rápido e eficiente usado para gerenciar e armazenar dados do usuário.
- O MariaDB então envia as informações para a instância do NetApp Azure, que conecta os dados do usuário à unidade de processamento de IA.

Interação com OpenAI e filtragem de conteúdo:

- A instância do Azure envia as perguntas do usuário para um sistema de filtragem de conteúdo. Este sistema limpa a consulta e a prepara para processamento.
- A entrada limpa é então enviada ao modelo base do Azure OpenAI, que gera uma resposta com base na entrada.

Geração e moderação de respostas:

- A resposta do modelo base é primeiro verificada para garantir que seja precisa e atenda aos padrões de conteúdo.
- Após passar na verificação, a resposta é enviada de volta ao usuário. Esse processo garante que o

usuário receba uma resposta clara, precisa e apropriada à sua consulta.

Conclusão

Esta seção conclui a solução de banco de dados vetorial para NetApp.

Conclusão

Concluindo, este documento fornece uma visão geral abrangente da implantação e do gerenciamento de bancos de dados vetoriais, como Milvus e pgvector, em soluções de armazenamento da NetApp . Discutimos as diretrizes de infraestrutura para aproveitar o armazenamento de objetos NetApp ONTAP e StorageGRID e validamos o banco de dados Milvus no AWS FSx ONTAP por meio de armazenamento de arquivos e objetos.

Exploramos a dualidade arquivo-objeto do NetApp, demonstrando sua utilidade não apenas para dados em bancos de dados vetoriais, mas também para outros aplicativos. Também destacamos como o SnapCenter, produto de gerenciamento empresarial da NetApp, oferece funcionalidades de backup, restauração e clonagem para dados de bancos de dados vetoriais, garantindo a integridade e a disponibilidade dos dados.

O documento também analisa como a solução de nuvem híbrida da NetApp oferece replicação e proteção de dados em ambientes locais e na nuvem, proporcionando uma experiência de gerenciamento de dados segura e contínua. Fornecemos insights sobre a validação de desempenho de bancos de dados vetoriais como Milvus e pgvecto no NetApp ONTAP, oferecendo informações valiosas sobre sua eficiência e escalabilidade.

Por fim, discutimos dois casos de uso de IA generativa: RAG com LLM e ChatAI interno da NetApp. Esses exemplos práticos ressaltam as aplicações e os benefícios reais dos conceitos e práticas descritos neste documento. No geral, este documento serve como um guia abrangente para qualquer pessoa que queira aproveitar as poderosas soluções de armazenamento da NetApp para gerenciar bancos de dados vetoriais.

Agradecimentos

O autor gostaria de agradecer sinceramente aos colaboradores abaixo, a outros que forneceram feedback e comentários para tornar este artigo valioso para os clientes e os setores da NetApp .

1. Sathish Thyagarajan, engenheiro técnico de marketing, ONTAP AI & Analytics, NetApp
2. Mike Oglesby, engenheiro de marketing técnico, NetApp
3. AJ Mahajan, Diretor Sênior, NetApp
4. Joe Scott, Gerente, Engenharia de Desempenho de Carga de Trabalho, NetApp
5. Puneet Dhawan, Diretor Sênior, Gerenciamento de Produtos Fsx, NetApp
6. Yuval Kalderon, gerente sênior de produtos, equipe de produtos FSx, NetApp

Onde encontrar informações adicionais

Para saber mais sobre as informações descritas neste documento, revise os seguintes documentos e/ou sites:

- Documentação do Milvus - https://milvus.io/docs/overview.md
- Documentação independente do Milvus - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- Documentação do produto NetApphttps://www.netapp.com/support-and-training/documentation/[]
- instaclustr -"documentação do installclustr"

Histórico de versões

Versão	Data	Histórico de versões do documento
Versão 1.0	Abril de 2024	Lançamento inicial

Apêndice A: Valores.yaml

Esta seção fornece código YAML de exemplo para os valores usados na solução de banco de dados vetorial NetApp .

Apêndice A: Valores.yaml

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

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

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

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

# Global affinity
```

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

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

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

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

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

```
  ##
  externalIPs: []
  #   - externalIp1

  # LoadBalancerSourcesRange is a list of allowed CIDR values, which are
combined with ServicePort to
  # set allowed inbound rules on the security group assigned to the master
load balancer
  loadBalancerSourceRanges:
  - 0.0.0.0/0
  # Optionally assign a known public LB IP
  # loadBalancerIP: 1.2.3.4

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

serviceAccount:
  create: false
  name:
  annotations:
  labels:

metrics:
  enabled: true

  serviceMonitor:
    # Set this to `true` to create ServiceMonitor for Prometheus operator
```

```
    enabled: false
    interval: "30s"
    scrapeTimeout: "10s"
    # Additional labels that can be used so ServiceMonitor will be
discovered by Prometheus
    additionalLabels: {}

livenessProbe:
  enabled: true
  initialDelaySeconds: 90
  periodSeconds: 30
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5

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

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

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Logs Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
```

```
spec is
      ##   set, choosing the default provisioner.
      ## ReadWriteMany access mode required for milvus cluster.
      ##
      storageClass: default
      accessModes: ReadWriteMany
      size: 10Gi
      subPath: ""

## Heaptrack traces all memory allocations and annotates these events with
stack traces.
## See more: https://github.com/KDE/heaptrack
## Enable heaptrack in production is not recommended.
heaptrack:
  image:
    repository: milvusdb/heaptrack
    tag: v0.1.0
    pullPolicy: IfNotPresent

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

  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
    mountPath: "/var/lib/milvus"
    ## If true, alertmanager will create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
```

```
    enabled: true
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ##
      storageClass:
      accessModes: ReadWriteOnce
      size: 50Gi
      subPath: ""

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

```
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

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

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

queryCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Query Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
```

```
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for query coordinator

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

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

indexCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1   # Run Index Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
```

```
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for index coordinator

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

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

dataCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1           # Run Data Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
```

```
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for data coordinator

  service:
    port: 13333
    annotations: {}
    labels: {}
    clusterIP: ""

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

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

```
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for Mixture coordinator

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

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


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

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
```

```yaml
## 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:~#
```

Apêndice B: prepare_data_netapp_new.py

Esta seção fornece um exemplo de script Python usado para preparar dados para o banco de dados vetorial.

Apêndice 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:~#
```

Apêndice C: verify_data_netapp.py

Esta seção contém um script Python de exemplo que pode ser usado para validar o banco de dados vetorial na solução de banco de dados vetorial NetApp .

Apêndice 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:~#
```

Apêndice D: docker-compose.yml

Esta seção inclui código YAML de exemplo para a solução de banco de dados vetorial para NetApp.

Apêndice D: docker-compose.yml

```
version: '3.5'

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

  minio:
    container_name: milvus-minio
    image: minio/minio:RELEASE.2023-03-20T20-16-18Z
    environment:
      MINIO_ACCESS_KEY: minioadmin
      MINIO_SECRET_KEY: minioadmin
    ports:
      - "9001:9001"
      - "9000:9000"
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/minio:/minio_data
    command: minio server /minio_data --console-address ":9001"
```

```
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost:9000/minio/health/live"]
      interval: 30s
      timeout: 20s
      retries: 3

  standalone:
    container_name: milvus-standalone
    image: milvusdb/milvus:v2.4.0-rc.1
    command: ["milvus", "run", "standalone"]
    security_opt:
    - seccomp:unconfined
    environment:
      ETCD_ENDPOINTS: etcd:2379
      MINIO_ADDRESS: minio:9000
    volumes:
      - /home/ubuntu/milvusvectordb/volumes/milvus:/var/lib/milvus
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:9091/healthz"]
      interval: 30s
      start_period: 90s
      timeout: 20s
      retries: 3
    ports:
      - "19530:19530"
      - "9091:9091"
    depends_on:
      - "etcd"
      - "minio"

networks:
  default:
    name: milvus
```

Informações sobre direitos autorais

Informações sobre marcas comerciais