Vector Database解決策with NetApp

NetApp Solutions

NetApp
May 03, 2024

This PDF was generated from https://docs.netapp.com/ja-jp/netapp-solutions/ai/vector-database-introduction.html on May 03, 2024. Always check docs.netapp.com for the latest.

目次

Vector Database解決策with NetApp

Karthikeyan NagalingamとRodrigo Nascimento、NetApp

このドキュメントでは、ネットアップのストレージソリューションを使用して、Milvusなどのベクターデータベースや、オープンソースのPostgreSQL拡張pgvectoの導入と管理について詳しく説明します。NetApp ONTAPとStorageGRIDオブジェクトストレージを使用するためのインフラガイドラインについて詳しく説明し、AWS FSx for NetApp ONTAPでのMilvusデータベースの適用を検証します。このドキュメントでは、ネットアップのファイルオブジェクトの二重性と、ベクター埋め込みをサポートするベクターデータベースやアプリケーションに対するその有用性について説明しています。また、ベクトルデータベースのバックアップとリストア機能を提供し、データの整合性と可用性を確保する、ネットアップのエンタープライズ管理製品であるSnapCenterの機能を強調しています。このドキュメントでは、ネットアップのハイブリッドクラウド解決策についてさらに詳しく解説し、オンプレミス環境とクラウド環境にわたるデータレプリケーションと保護におけるネットアップの役割について解説します。また、NetApp ONTAP上のベクトルデータベースのパフォーマンス検証に関するインサイトも含まれています。最後に、生成型AIに関する2つの実用的なユースケース、LLMを使用したRAGと、ネットアップ社内のChatAIについて説明します。本ドキュメントは、ベクターデータベースの管理にネットアップのストレージソリューションを活用するための包括的なガイドです。

リファレンスアーキテクチャでは、次の点に重点を置いています。

12. "付録C：verify_data_netapp.py"
13. "付録D：docker-compose.yml"

はじめに

はじめに

ベクトルデータベースは、大規模言語モデル(LLM)および生成人工知能(AI)における意味探索の複雑さを処理するように設計されている課題に効果的に対処します。従来のデータ管理システムとは異なり、ベクターデータベースは、画像、ビデオ、テキスト、オーディオ、 ラベルやタグではなく、データ自体のコンテンツを使用することで、他の形式の非構造化データを作成できます。

リレーショナルデータベース管理システム（RDBMS）の限界は十分に文書化されています。特に、AIアプリケーションで一般的に使用される高次元データ表現や非構造化データに関する課題が顕著です。RDBMSでは、多くの場合、データをより管理しやすい構造にフラット化するために時間がかかり、エラーが発生しやすいプロセスが必要となり、検索の遅延や非効率化につながります。しかし、ベクターデータベースはこれらの問題を回避するように設計されており、複雑で高次元のデータを管理および検索するためのより効率的で正確な解決策を提供し、AIアプリケーションの進歩を促進します。

本ドキュメントは、現在ベクターデータベースを使用している、または使用を計画しているお客様向けの包括的なガイドであり、NetApp ONTAP、NetApp StorageGRID、Amazon FSxN for NetApp ONTAP、SnapCenterなどのプラットフォームでベクターデータベースを使用するためのベストプラクティスについて詳しく説明しています。ここでは、次のようなさまざまなトピックについて説明します。

- NetAppストレージがNetApp ONTAPおよびStorageGRIDオブジェクトストレージを介して提供する、Milvusなどのベクターデータベースのインフラガイドライン。
- ファイルとオブジェクトストアを使用したAWS FSx for NetApp ONTAPでのMilvusデータベースの検証。
- ネットアップのファイルオブジェクトとオブジェクトの二重性を詳しく検証し、ベクターデータベースやその他のアプリケーションのデータに対するその有用性を実証します。
- ネットアップのデータ保護管理製品SnapCenterは、ベクターデータベースデータのバックアップとリストアの機能を提供します。
- ネットアップのハイブリッドクラウドは、オンプレミス環境とクラウド環境にわたってデータのレプリケーションと保護を提供します。
- NetApp ONTAP上のMilvusやpgvectorなどのベクターデータベースのパフォーマンス検証に関するインサイトを提供します。
- 2つの具体的なユースケース:大規模言語モデル(LLM)を使用したRetrieval Augmented Generation (RAG)とNetApp ITチームのChatAIで、概説された概念と実践の実践例を提供します。

解決策の概要

解決策の概要

この解決策では、ベクターデータベースのお客様が直面している課題に対処するためにNetAppが提供する独自のメリットと機能を紹介します。NetApp ONTAP、StorageGRID、ネットアップのクラウドソリューション、SnapCenterを活用することで、お客様はビジネスの運用に大きな価値を付加できます。これらのツールは、既存の問題に対処するだけでなく、効率性と生産性を向上させ、ビジネス全体の成長に貢献します。

NetAppを選ぶ理由

- ONTAPやStorageGRIDなどのネットアップのソリューションを使用すると、ストレージとコンピューティングを分離し、特定の要件に基づいてリソース利用率を最適化できます。この柔軟性により、NetAppストレージソリューションを使用してストレージを個別に拡張できます。
- ネットアップのストレージコントローラを活用することで、NFSプロトコルとS3プロトコルを使用してベクトルデータベースに効率的にデータを提供できます。これらのプロトコルを使用すると、顧客データの保存とベクターデータベースインデックスの管理が容易になり、ファイルメソッドやオブジェクトメソッドを使用してデータの複数のコピーにアクセスする必要がなくなります。
- NetApp ONTAPは、AWS、Azure、Google Cloudなどの主要なクラウドサービスプロバイダでNASとオブジェクトストレージをネイティブにサポートします。この幅広い互換性により、シームレスな統合が保証され、お客様のデータモビリティ、グローバルなアクセス性、ディザスタリカバリ、動的な拡張性、および高パフォーマンスが実現します。
- ネットアップの堅牢なデータ管理機能を使用すれば、データが潜在的なリスクや脅威から十分に保護されているため、お客様は安心してデータを管理できます。NetAppはデータセキュリティを優先し、お客様の貴重な情報の安全性と整合性についてお客様に安心を提供します。

ベクターデータベース

ベクターデータベース

ベクターデータベースは、機械学習モデルの埋め込みを使用して非構造化データを処理、索引付け、検索するように設計された特殊なタイプのデータベースです。従来の表形式でデータを整理する代わりに、ベクトル埋め込みとも呼ばれる高次元ベクトルとしてデータを配置します。このユニークな構造により、データベースは複雑な多次元データをより効率的かつ正確に処理することができます。

ベクターデータベースの重要な機能の1つは、生成AIを使用して分析を実行することです。これには、指定された入力のようなデータポイントをデータベースが識別する類似性検索と、ノルムから大幅に逸脱したデータポイントを検出できる異常検出が含まれます。

さらに、ベクターデータベースは時間データ、つまりタイムスタンプされたデータを処理するのに適しています。このタイプのデータは、特定のITシステム内の他のすべてのイベントに関連して、「何」が発生したのか、いつ発生したのかについての情報を順番に提供します。時間データを処理および分析するこの機能により、ベクターデータベースは、時間の経過に伴うイベントの理解を必要とするアプリケーションに特に役立ちます。

MLおよび**AI**向けベクターデータベースの利点：

- 高次元検索:ベクトルデータベースは、AIやMLアプリケーションでよく生成される高次元データの管理と取得に優れています。
- 拡張性：AIプロジェクトやMLプロジェクトの成長と拡張をサポートして、大量のデータを処理できるように効率的に拡張できます。
- 柔軟性:ベクターデータベースは高度な柔軟性を提供し、多様なデータ型や構造に対応できます。
- パフォーマンス：AIとMLの処理速度と効率に欠かせない、ハイパフォーマンスなデータの管理と読み出しを提供します。
- カスタマイズ可能なインデックス作成:ベクターデータベースは、カスタマイズ可能なインデックス作成オプションを提供し、特定のニーズに基づいてデータの整理と取得を最適化します。

ベクターデータベースとユースケース。

このセクションでは、さまざまなベクターデータベースとそのユースケースの詳細について説明します。

FaissとScaNN

これらはベクトル探索の領域で重要なツールとして機能するライブラリです。これらのライブラリは、ベクトルデータの管理と検索に役立つ機能を提供し、データ管理のこの専門分野で非常に貴重なリソースを提供します。

Elasticsearch を指定します

これは広く使用されている検索および分析エンジンであり、最近ベクトル検索機能が組み込まれています。この新機能は、その機能を強化し、ベクターデータをより効率的に処理および検索できるようにします。

パインコーン

これは、独自の機能セットを備えた堅牢なベクターデータベースです。インデックス作成機能で高密度ベクトルとスパースベクトルの両方をサポートしており、柔軟性と適応性が向上しています。その主な強みの1つは、従来の検索手法とAIベースの高密度ベクトル検索を組み合わせることで、両者の長所を活用したハイブリッド検索アプローチを構築できることにあります。

主にクラウドベースのPineconeは、機械学習アプリケーション向けに設計されており、GCP、AWS、Open AI、GPT-3、 GPT-3.5、GPT-4、Catgut Plus、Elasticsearch、Haystack、 その他多数。Pineconeはクローズドソースプラットフォームであり、Software as a Service（SaaS）として提供されていることに注意してください。

高度な機能を備えたPineconeは、その高度な検索機能とハイブリッド検索機能を効果的に活用して脅威を検出して対応できるサイバーセキュリティ業界に特に適しています。

クロマ

これは、4つの主要な関数を持つCore-APIを持つベクターデータベースです。そのうちの1つには、メモリ内のドキュメントベクトルストアが含まれています。また、Face Transformersライブラリを使用してドキュメントをベクトル化し、機能と汎用性を強化します。
Chromaはクラウドとオンプレミスの両方で動作するように設計されており、ユーザーのニーズに応じた柔軟性を提供します。特に、オーディオ関連のアプリケーションに優れており、オーディオベースの検索エンジン、音楽レコメンドシステム、その他のオーディオ関連のユースケースに最適です。

ウィアヴィアート

これは、組み込みモジュールまたはカスタムモジュールのいずれかを使用してコンテンツをベクトル化できる汎用性の高いベクターデータベースであり、特定のニーズに基づいた柔軟性を提供します。フルマネージド型とセルフホスティング型の両方のソリューションを提供し、さまざまな導入設定に対応します。

Weaviateの主な機能の1つは、ベクトルとオブジェクトの両方を格納できることで、データ処理機能が強化されていることです。ERPシステムのセマンティック検索やデータ分類など、さまざまなアプリケーションに広く使用されています。電子商取引分野では、検索および推薦エンジンを強化しています。Weaviateは、画像検索、異常検出、自動データ調和、サイバーセキュリティの脅威分析にも使用され、複数のドメインにわたるその汎用性を示しています。

Redis

Redisは、高速なインメモリストレージで知られている高性能ベクトルデータベースで、読み取り/書き込み処理のレイテンシが低くなります。そのため、迅速なデータアクセスが必要なレコメンドシステム、検索エンジン、データ分析アプリケーションに最適です。

Redisは、リスト、セット、ソートされたセットなど、ベクトルのさまざまなデータ構造をサポートしています。また、ベクトル間の距離の計算や交差と結合の検索などのベクトル操作も提供します。これらの機能は、類似性検索、クラスタリング、およびコンテンツベースのレコメンドシステムで特に役立ちます。

スケーラビリティと可用性の面では、Redisは高スループットワークロードの処理に優れ、データレプリケーションを提供します。また、従来のリレーショナルデータベース（RDBMS）を含む他のデータタイプともうまく統合できます。
Redisには、リアルタイム更新のためのパブリッシュ/サブスクライブ(Pub/Sub)機能が含まれています。これは、リアルタイムベクトルを管理するのに役立ちます。さらに、Redisは軽量で使いやすく、ベクトルデータを管理するためのユーザーフレンドリーな解決策になっています。

ミルバス

MongoDBと同様に、ドキュメントストアのようなAPIを提供する汎用性の高いベクターデータベースです。多種多様なデータ型をサポートしているため、データサイエンスや機械学習の分野で人気があります。

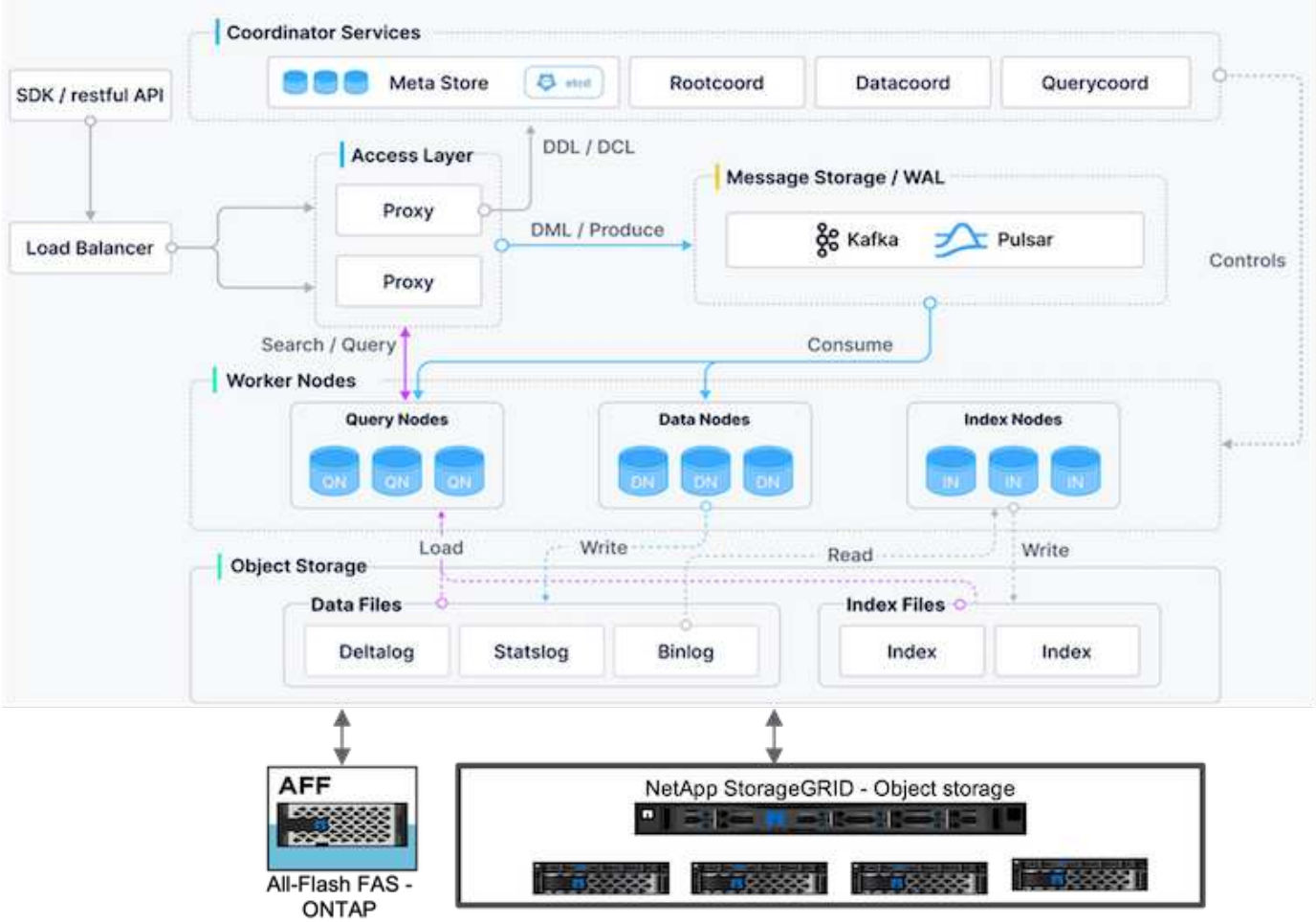
Milvusのユニークな機能の1つは、検索に使用するベクトルの種類を実行時に指定できるマルチベクトル化機能です。さらに、Faissのような他のライブラリの上にあるライブラリKnowwhereを利用して、クエリとベクトル検索アルゴリズム間の通信を管理します。

Milvusは、PyTorchおよびTensorFlowとの互換性により、機械学習ワークフローとのシームレスな統合も提供します。これにより、eコマース、画像およびビデオ分析、オブジェクト認識、画像類似性検索、コンテンツベースの画像検索など、さまざまなアプリケーションに適した優れたツールになります。自然言語処理の分野では、ミルヴァスは文書のクラスタリング、意味検索、質問応答システムに使われている。

この解決策では、解決策検証用にmilvusを選択しました。パフォーマンスについては、milvusとpostgres（pgvecto.rs）の両方を使用しました。

この解決策にミルバスを選んだ理由は何ですか？

- オープンソース: Milvusはオープンソースのベクトルデータベースで、コミュニティ主導の開発と改善を奨励しています。
- AI統合:類似性検索とAIアプリケーションの埋め込みを活用して、ベクターデータベースの機能を強化します。
- 大量処理：Milvusには、ディープニューラルネットワーク（DNN）モデルと機械学習（ML）モデルによって生成された10億個以上の埋め込みベクトルを保存、インデックス化、管理する能力があります。
- ユーザーフレンドリー:使いやすく、セットアップは1分未満で完了します。Milvusは、さまざまなプログラミング言語用のSDKも提供しています。
- スピード:一部の代替製品に比べて最大10倍の高速な読み出し速度を提供します。
- スケーラビリティと可用性: Milvusは拡張性に優れており、必要に応じてスケールアップとスケールアウトのオプションがあります。
- 豊富な機能:さまざまなデータ型、属性フィルタリング、ユーザー定義関数(UDF)のサポート、設定可能な整合性レベル、移動時間をサポートし、さまざまなアプリケーションに対応する汎用性の高いツールです。



このセクションでは、Milvusアーキテクチャで使用されるより高いレバーコンポーネントとサービスを提供します。
*アクセス層–ステートレスプロキシのグループで構成され、システムおよびエンドポイントの最前面層として機能します。
*コーディネータサービス–作業者ノードにタスクを割り当て、システムの頭脳として機能します。ルートコード、データコード、クエリコードの3つのコーディネータタイプがあります。
*ワーカーノード:コーディネータサービスの指示に従い、ユーザートリガー型DML/DDLを実行commands.itには、クエリノード、データノード、インデックスノードの3種類のワーカーノードがあります。
*ストレージ：データの永続性を管理します。メタストレージ、ログブローカー、オブジェクトストレージで構成されます。ONTAPやStorageGRIDなどのNetAppストレージは、顧客データとベクターデータベースデータの両方について、オブジェクトストレージとファイルベースストレージをMilvusに提供します。

技術要件

技術要件

以下に示すハードウェアとソフトウェアの構成は、パフォーマンスを除き、本ドキュメントで実行した検証の大部分に使用されています。これらの構成は、環境のセットアップに役立つガイドラインです。ただし、個々 のお客様の要件によって特定のコンポーネントが異なる場合があることに注意してください。

ハードウェア要件

ハードウェア	詳細
NetApp AFFストレージアレイHAペア	* A800 * ONTAP 9.14.1 * 3.49TB SSD-NVM×48 * 2つのフレキシブルグループボリューム：メタデータとデータ。 *メタデータNFSボリュームには、250GBの永続ボリュームが12個あります。 *データはONTAP NAS S3ボリューム
Fujitsu PRIMERGY RX2540 M4×6	* 64 CPU * Intel（R）Xeon（R）Gold 6142 CPU（2.60GHz） *256 GM物理メモリ * 1 x 100GbEネットワークポート
ネットワーキング	100GbE
StorageGRID	* SG100×1、SGF6024×3 * 3 x 24 x 7.68TB

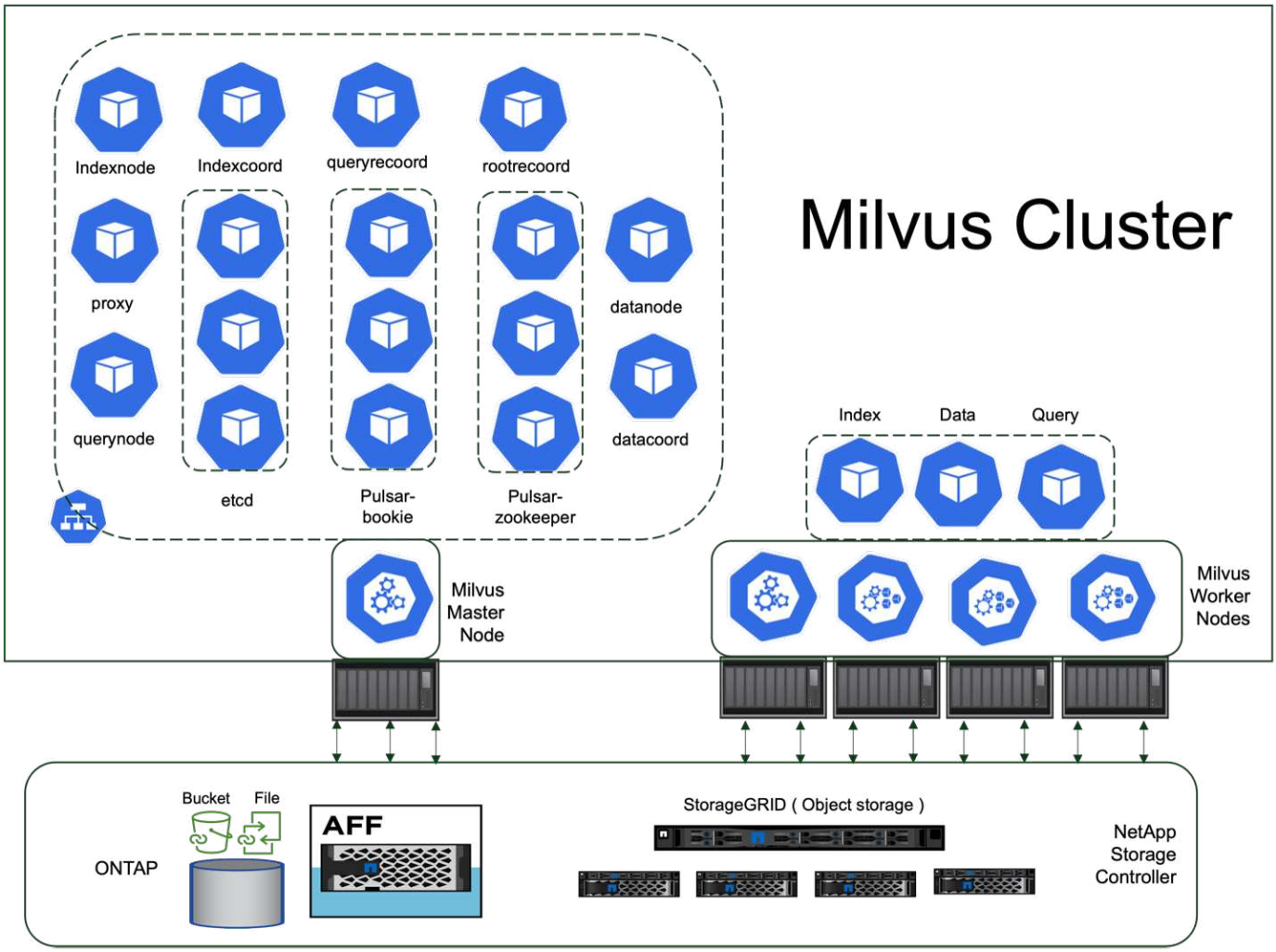
ソフトウェア要件

ソフトウェア	詳細
ミルバスクラスタ	*chart-milvus-4.1.11. *アプリのバージョン–2.3.4 * bookkeeper、zookeeper、pulsar、etcd、proxyなどの依存バンドル querynode、worker
Kubernetes	* 5ノードのKubernetesクラスタ * 1マスターノードと4ワーカーノード *バージョン–1.7.2
Python	*3.10.12.

Deployment手順

Deployment手順

この導入セクションでは、以下のラボセットアップにmilvusベクターデータベースとKubernetesを使用しました。



NetAppストレージは、顧客データとmilvusクラスタデータを保持するためのストレージをクラスタに提供します。

NetAppストレージのセットアップ**–ONTAP**

- ストレージシステムの初期化
- Storage Virtual Machine（SVM）の作成
- 論理ネットワークインターフェイスの割り当て
- NFS、S3の構成とライセンス

NFS（ネットワークファイルシステム）については、次の手順を実行してください。

1. NFSv4用のFlexGroupボリュームを作成します。この検証用のセットアップでは、48本のSSDを使用しました。1本はコントローラのルートボリューム専用のSSD、47本はNFSv4用に分散しています]]。FlexGroupボリュームのNFSエクスポートポリシーに、Kubernetes（Kubernetes）ノードネットワークに対する読み取り/書き込み権限が設定されていることを確認します。これらの権限が設定されていない場合は、Kubernetesノードネットワークに対して読み取り/書き込み（rw）権限を付与してください。
2. すべてのKubernetesノードで、フォルダを作成し、各Kubernetesノードの論理インターフェイス（LIF）を使用してこのフォルダにFlexGroupボリュームをマウントします。

NAS S3（Network Attached Storage Simple Storage Service）については、次の手順を実行してください。

1. NFS用のFlexGroupボリュームを作成します。
2. vserver object-store-server createコマンドを使用して、HTTPを有効にし、管理ステータスを「up」に設定します。HTTPSを有効にし、カスタムリスナーポートを設定することもできます。
3. 「vserver object-store-server user create -user <username>」コマンドを使用して、object-store-serverユーザを作成します。
4. アクセスキーとシークレットキーを取得するには、次のコマンドを実行します。"set diag；vserver object-store-server user show -user <username>"。ただし、今後は、これらのキーはユーザ作成プロセス中に提供されるか、REST API呼び出しを使用して取得できます。
5. 手順2で作成したユーザを使用してオブジェクトストアサーバグループを作成し、アクセスを許可します。この例では、「FullAccess」を提供しています。
6. タイプを「nas」に設定し、NFSv3ボリュームへのパスを指定して、NASバケットを作成します。この目的のためにS3バケットを利用することもできます。

NetAppストレージのセットアップ–StorageGRID

1. StorageGRIDソフトウェアをインストールします。
2. テナントとバケットを作成
3. 必要な権限を持つユーザを作成します。

詳細については、 https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

解決策の概要

当社は、以下の5つの主要分野に焦点を当てた包括的な解決策検証を実施しました。各セクションでは、お客様が直面している課題、NetAppが提供するソリューション、その後お客様にもたらされるメリットについて詳しく説明します。

1. "オンプレミスでのKubernetesを使用したMilvusクラスタセットアップ"
   ストレージとコンピューティングの個別拡張、効果的なインフラ管理、データ管理に関するお客様の課題このセクションでは、クラスタデータと顧客データの両方にNetAppストレージコントローラを利用して、KubernetesにMilvusクラスタをインストールするプロセスを詳しく説明します。
2. "MilvusとAmazon FSxN for NetApp ONTAP–ファイルとオブジェクトの二重性"
   このセクションでは、クラウドにベクターデータベースをデプロイする必要がある理由と、Dockerコンテナ内のAmazon FSxN for NetApp ONTAPにベクターデータベース(milvusスタンドアロン)をデプロイする手順について説明します。
3. "NetApp SnapCenterを使用したベクターデータベース保護。"
   このセクションでは、SnapCenterがONTAPに存在するベクターデータベースデータとMilvusデータをどのように保護するかについて詳しく説明します。この例では、顧客データ用にNFS ONTAPボリューム（vol1）から派生したNASバケット（milvusdbvol1）を使用し、Milvusクラスタ構成データ用に別のNFSボリューム（vectordbpv）を使用しました。
4. "NetApp SnapMirrorを使用したディザスタリカバリ"
   このセクションでは、ベクターデータベースに対するディザスタリカバリ（DR）の重要性と、NetAppディザスタリカバリ製品のSnapMirrorがベクターデータベースに対するDR解決策をどのように提供するかについて説明します。
5. "パフォーマンスの検証"
   このセクションでは、Milvusやpgvecto.RSなどのベクターデータベースのパフォーマンス検証につい

て、LLMライフサイクル内でのRAGおよび推論ワークロードをサポートする際のI/OプロファイルやNetAppストレージコントローラの動作などのストレージパフォーマンス特性に焦点を当てて詳しく説明します。これらのデータベースをONTAPストレージ解決策と組み合わせた場合のパフォーマンスの差別化要因を評価し、特定します。分析は、1秒あたりに処理されるクエリ数（QPS）などの主要パフォーマンス指標に基づいて行われます。

オンプレミスでの**Kubernetes**を使用した**Milvus**クラスタセットアップ

オンプレミスでの**Kubernetes**を使用した**Milvus**クラスタセットアップ

ストレージとコンピューティングの個別拡張、効果的なインフラ管理とデータ管理、Kubernetesデータベースとベクトルデータベースは、大規模なデータ処理を管理するための、強力で拡張性に優れた解決策を形成します。Kubernetesはリソースを最適化してコンテナを管理し、ベクトルデータベースは高次元データや類似性検索を効率的に処理します。この組み合わせにより、大規模なデータセットで複雑なクエリを迅速に処理し、データ量の増大に合わせてシームレスに拡張できるため、ビッグデータアプリケーションやAIワークロードに最適です。

1. このセクションでは、クラスタデータと顧客データの両方にNetAppストレージコントローラを利用して、KubernetesにMilvusクラスタをインストールするプロセスを詳しく説明します。
2. Milvusクラスタをインストールするには、さまざまなMilvusクラスタコンポーネントからのデータを格納するためにPersistent Volumes（PVS）が必要です。これらのコンポーネントには、etcd（3つのインスタンス）、pulsar-bookie-journal（3つのインスタンス）、pulsar-bookie-ledgers（3つのインスタンス）、pulsar-zookeeper-data（3つのインスタンス）があります。

> milvusクラスターでは、Milvusクラスターの信頼性の高いストレージとメッセージストリームのパブリッシュ/サブスクリプションをサポートする基盤となるエンジンにパルサーまたはKafkaのいずれかを使用できます。NFSを使用するKafkaについては、ONTAP 9.12.1以降で改善が行われています。これらの機能強化と、RHEL 8.7または9.1以降でのNFSv4.1およびLinuxの変更に加えて、NFSでKafkaを実行するときに発生する「silly rename」問題を解決してください。NetApp NFS解決策でKafkaを実行する方法の詳細については、https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html。

3. NetApp ONTAPからNFSボリュームを1つ作成し、250GBのストレージを含む12個の永続ボリュームを構築しました。ストレージサイズはクラスタサイズによって異なります。たとえば、各PVが50GBの別のクラスタがあります。詳細については、以下のいずれかのPV YAMLファイルを参照してください。このようなファイルは合計12個ありました。各ファイルでは、storageClassNameが「default」に設定され、ストレージとパスはPVごとに一意です。

```
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. 各PV YAMLファイルに対して「kubectl apply」コマンドを実行して永続ボリュームを作成し、「kubectl get pv」を使用して作成を確認します。

```
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. 顧客データの格納については、MinIO、Azure Blob、S3などのオブジェクトストレージソリューションをサポートしています。このガイドではS3を使用します。次の手順は、ONTAP S3とStorageGRIDオブジェクトストアの両方に適用されます。Helmを使用してMilvusクラスタを導入します。Milvusのダウンロード場所から、設定ファイルvalues.yamlをダウンロードします。このドキュメントで使用したvalues.yamlファイルについては、付録を参照してください。
6. log、etcd、zookeeper、およびbookkeeperのセクションを含め、「storageClass」が各セクションで「default」に設定されていることを確認します。
7. MinIOセクションで、MinIOを無効にします。
8. ONTAPまたはStorageGRIDオブジェクトストレージからNASバケットを作成し、オブジェクトストレージのクレデンシャルを使用して外部S3に追加します。

```
###################################
# 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. Milvusクラスタを作成する前に、PersistentVolumeClaim（PVC）に既存のリソースがないことを確認してください。

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

10. Helmとvalues.yaml設定ファイルを使用して、Milvusクラスタをインストールして起動します。

```
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. PersistentVolumeClaims（PVC）のステータスを確認します。

```
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. ポッドのステータスを確認します。

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

ポッドのステータスが「Running」で正常に機能していることを確認してください

13. MilvusおよびNetAppオブジェクトストレージでデータの書き込みと読み取りをテストします。
    - Pythonプログラム「prepare_data_netapp_new.py」を使用してデータを書き込みます。

```
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:~#
```

- Pythonファイル「verify_data_netapp.py」を使用してデータを読み取ります。

```
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}}]}
```

上記の検証に基づいて、NetAppストレージコントローラを使用したKubernetes上にMilvusクラスタを導入することで実証されているように、Kubernetesとベクトルデータベースの統合により、大規模なデータ操作を管理するための堅牢でスケーラブルで効率的な解決策が提供されます。このセットアップにより、お客様は高次元データを処理し、複雑なクエリを迅速かつ効率的に実行できるようになり、ビッグデータアプリケーションやAIワークロードに最適な解決策になります。さまざまなクラスタコンポーネントにPersistent Volume（PV；永続的ボリューム）を使用して、NetApp ONTAPから単一のNFSボリュームを作成することで、リソース利用率とデータ管理を最適化できます。PersistentVolumeClaims（PVC）とPODのステータスを検証し、データの書き込みと読み取りを

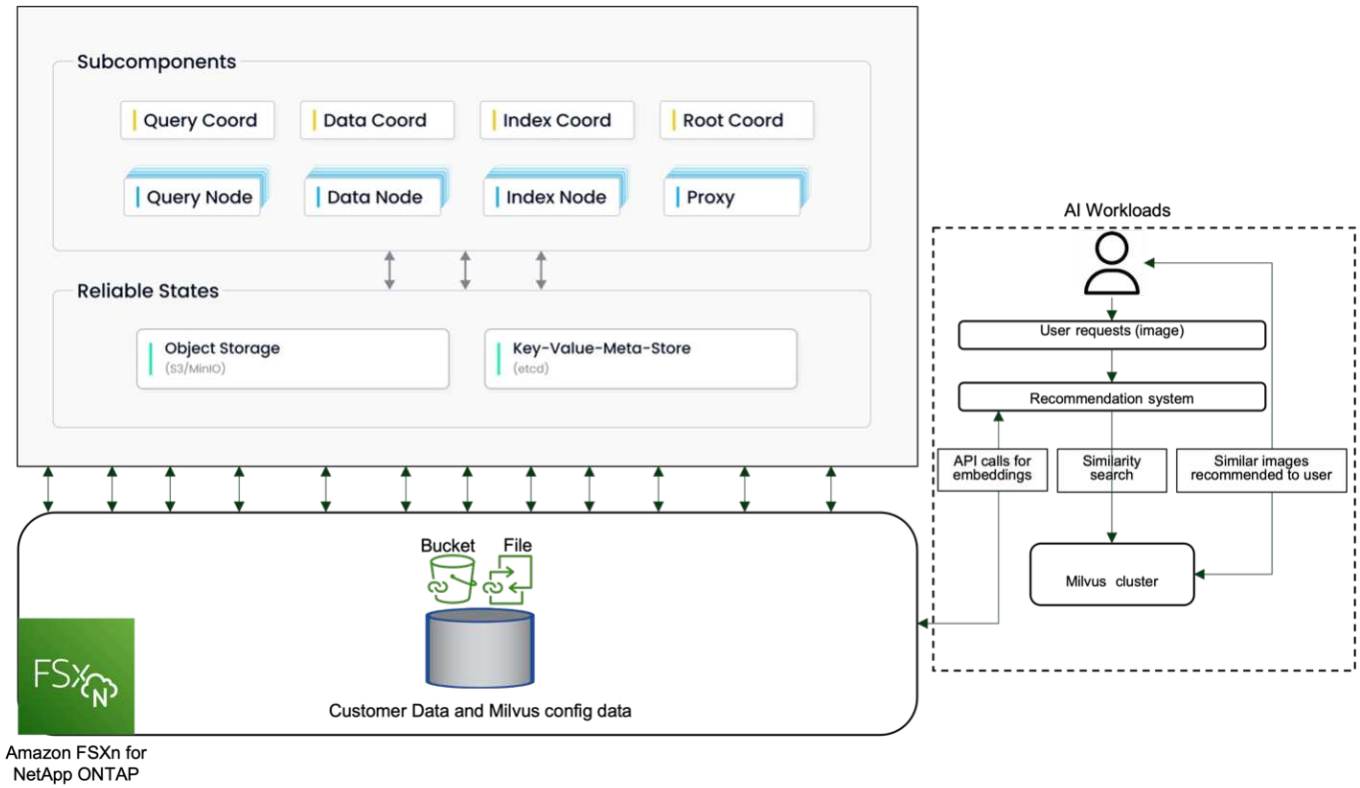
テストするプロセスにより、信頼性が高く一貫したデータ処理が保証されます。お客様のデータにONTAPまたはStorageGRIDオブジェクトストレージを使用すると、データへのアクセス性とセキュリティがさらに向上します。全体的に、このセットアップにより、お客様は、増大するデータニーズに合わせてシームレスに拡張できる耐障害性とパフォーマンスに優れたデータ管理解決策を利用できます。

MilvusとAmazon FSxN for NetApp ONTAP -ファイルとオブジェクトの二重性

MilvusとAmazon FSxN for NetApp ONTAP–ファイルとオブジェクトの二重性

このセクションでは、クラウドにベクターデータベースをデプロイする必要がある理由と、Dockerコンテナ内のAmazon FSxN for NetApp ONTAPにベクターデータベース(milvusスタンドアロン)をデプロイする手順について説明します。

クラウドにベクターデータベースを配置すると、特に高次元データを処理し、類似性検索を実行する必要があるアプリケーションには、いくつかの大きな利点があります。1つ目は、クラウドベースの導入により拡張性が向上し、増大するデータボリュームやクエリの負荷に合わせてリソースを簡単に調整できることです。これにより、データベースは高いパフォーマンスを維持しながら、増加する需要に効率的に対処できます。2つ目は、クラウドを導入することで、地理的に離れた場所にデータをレプリケートできるため、高可用性とディザスタリカバリが実現し、データ損失のリスクが最小限に抑えられ、予期しないイベントが発生しても継続的なサービスが提供されることです。3つ目は、使用したリソースに対してのみ料金が発生するため、対費用効果が高く、必要に応じてスケールアップまたはスケールダウンできるため、ハードウェアへの多額の先行投資が不要です。最後に、クラウドにベクターデータベースを導入すると、どこからでもデータにアクセスして共有できるため、コラボレーションが強化され、チームベースの作業やデータ主導の意思決定が容易になります。この検証で使用したAmazon FSxN for NetApp ONTAPを使用したmilvusスタンドアロンのアーキテクチャを確認してください。



1. Amazon FSxN for NetApp ONTAPインスタンスを作成し、VPC、VPCセキュリティグループ、およびサブネットの詳細をメモします。この情報は、EC2インスタンスを作成するときに必要になります。詳細につ

いては、こちらをご覧ください。 https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create

2. EC2インスタンスを作成し、VPC、セキュリティグループ、およびサブネットがAmazon FSxN for NetApp ONTAPインスタンスのものと一致するようにします。
3. コマンド「apt-get install nfs-common」を使用してnfs-commonをインストールし、「sudo apt-get update」を使用してパッケージ情報を更新します。
4. マウントフォルダを作成し、そのフォルダにAmazon FSxN for NetApp ONTAPをマウントします。

```
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. 'apt-get install'を使用してDockerとDocker Composeをインストールします。
6. Docker-compose.yamlファイルに基づいてMilvusクラスタをセットアップします。このファイルはMilvus Webサイトからダウンロードできます。

```
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. docker-compose.ymlファイルの「volumes」セクションで、NetApp NFSマウントポイントを対応するMilvusコンテナパス（具体的にはetcd、minio、standalone）にマッピングします。Check "付録D：docker-compose.yml" ymlの変更の詳細については、を参照してください。
8. マウントされたフォルダとファイルを確認します。

```
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. docker-compose.ymlファイルが格納されているディレクトリから'docker-compose up -d'を実行します。
10. Milvusコンテナのステータスを確認します。

```
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. VECTORデータベースの読み書き機能とAmazon FSxN for NetApp ONTAPのデータを検証するために、Python Milvus SDKとPyMilvusのサンプルプログラムを使用しました。'apt-get install python3-numpy python3-pip'を使用して必要なパッケージをインストールし、'pip3 install pymilvus'を使用してPyMilvusをインストールします。
12. VECTORデータベースのAmazon FSxN for NetApp ONTAPからのデータの書き込みおよび読み取り操作を

検証します。

```
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
/448789845791611912/448789845791611913/448789845791611939/100/448789845
```

```
91411920
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/100/448789845791
91411920/xl.meta
```

13. verify_data_netapp.pyスクリプトを使用して読み取り操作を確認します。

```
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. お客様が、AIワークロード用にS3プロトコルを介してVECTORデータベースでテストされたNFSデータにアクセスしたい（読み取り）場合は、わかりやすいPythonプログラムを使用して検証できます。この例として、このセクションの冒頭の図で説明したように、別のアプリケーションからの画像の類似性検索があります。

```
root@ip-172-31-29-98:~/pymilvus/examples# sudo python3
/home/ubuntu/milvusvectordb/s3_access.py -i 172.31.255.228 --bucket
milvusnasvol --access-key PY6UF318996I86NBYNDD --secret-key
hoPctr9aD88c1j0SkIYZ2uPa03vlbqKA0c5feK6F
OBJECTS in the bucket milvusnasvol are :
************************************
…
<output content removed to save page space>
```

```
…
bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/0/448789845791411917/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/1/448789845791411918/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/100/448789845791411913/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/101/448789845791411914/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/102/448789845791411915/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/103/448789845791411916/1c48ab6e-1546-4503-9084-28c629216c33/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611920/103/448789845791411916/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/101/448789845791411921/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/102/448789845791411922/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880/448789845791211881/448789845791411889/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880/448789845791211881/448789845791411889/100/448789845791411912/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912/448789845791611913/448789845791611920/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912/448789845791611913/448789845791611920/100/448789845791411919/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912/448789845791611913/448789845791611939/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912/448789845791611913/448789845791611939/100/448789845791411926/xl.meta
************************************
root@ip-172-31-29-98:~/pymilvus/examples#
```

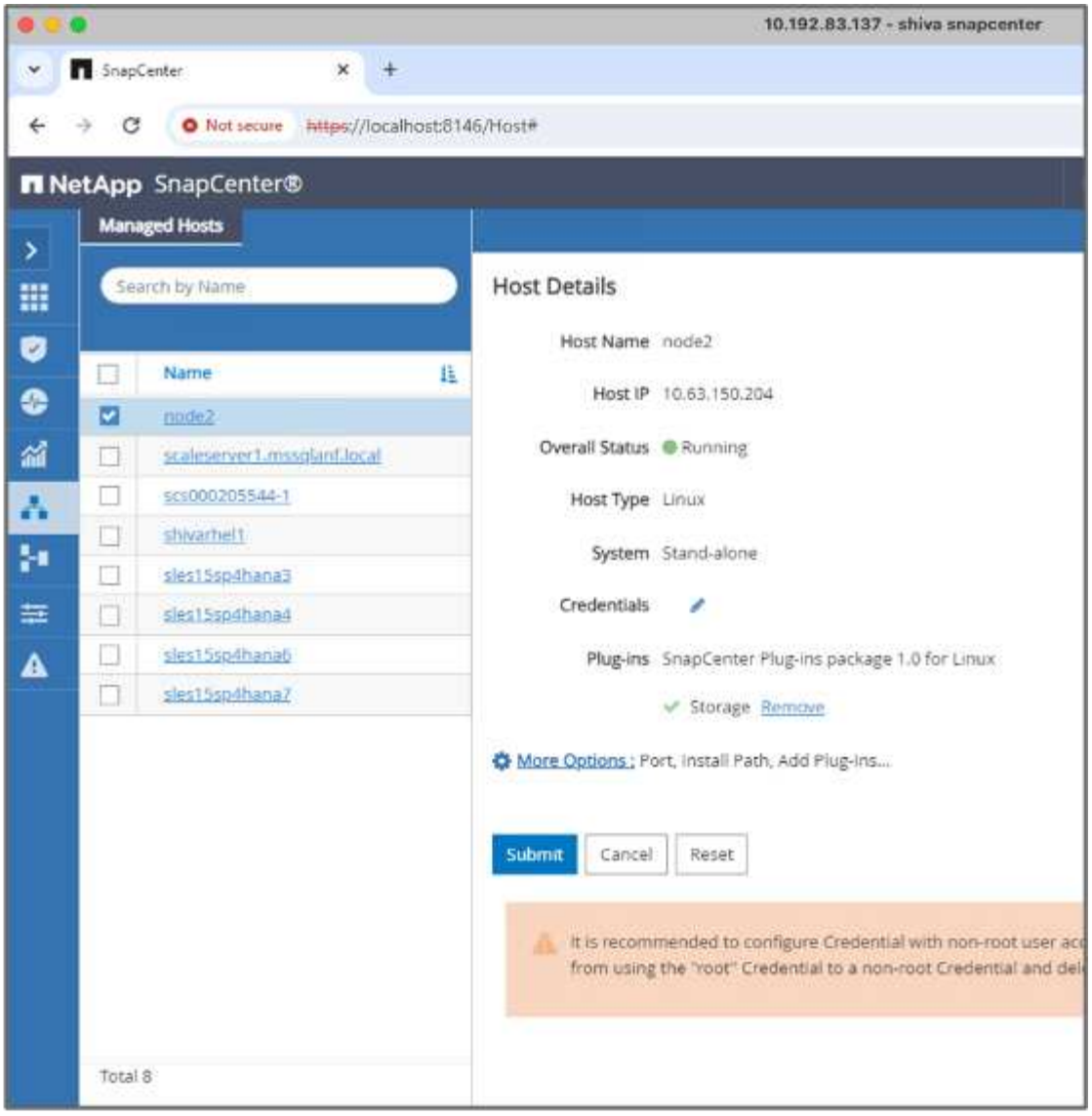
このセクションでは、AmazonのNetApp FSxN for NetApp ONTAPデータストレージを利用して、Dockerコンテナ内にスタンドアロンのMilvusセットアップを導入および運用する方法を効果的に説明します。このセットアップにより、ベクトルデータベースの機能を活用して高次元データを処理し、複雑なクエリを実行できます。これらはすべて、Dockerコンテナのスケーラブルで効率的な環境内で実行できます。Amazon FSxN for NetApp ONTAPインスタンスを作成し、EC2インスタンスを一致させることで、リソース利用率とデータ管理を最適化できます。VECTORデータベースでFSxNからのデータの書き込みおよび読み取り操作を正しく検証することで、信頼性が高く一貫したデータ操作を保証できます。さらに、AIワークロードからS3プロトコルを介してデータをリスト（読み取り）できるため、データへのアクセスが強化されます。したがって、この包括的なプロセスは、AmazonのFSxN for NetApp ONTAPの機能を活用して、大規模なデータ操作を管理するための堅牢で効率的な解決策を顧客に提供します。

SnapCenterを使用したベクターデータベース保護

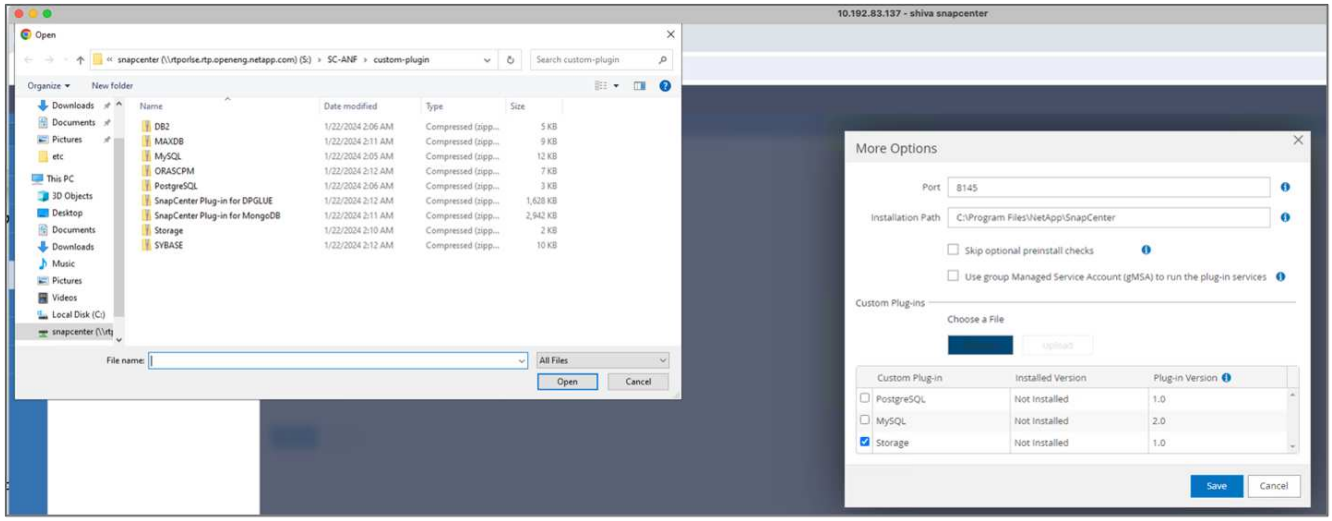
NetApp SnapCenterを使用したベクターデータベース保護。

たとえば、映画制作業界では、顧客はビデオファイルやオーディオファイルなどの重要な埋め込みデータを所有していることがよくあります。ハードドライブの故障などの問題が原因でこのデータが失われると、業務に大きな影響を与え、数百万ドル規模のベンチャーが危険にさらされる可能性があります。ネットアップでは、貴重なコンテンツが失われ、多大な混乱と財務上の損失を引き起こしている事例が発生しています。したがって、この業界では、この重要なデータのセキュリティと整合性を確保することが最も重要です。
このセクションでは、SnapCenterがONTAPに存在するベクターデータベースデータとMilvusデータをどのように保護するかについて詳しく説明します。この例では、顧客データ用にNFS ONTAPボリューム（vol1）から派生したNASバケット（milvusdbvol1）を使用し、Milvusクラスタ構成データ用に別のNFSボリューム（vectordbpv）を使用しました。次を確認してください： "こちらをご覧ください" SnapCenterバックアップワークフロー

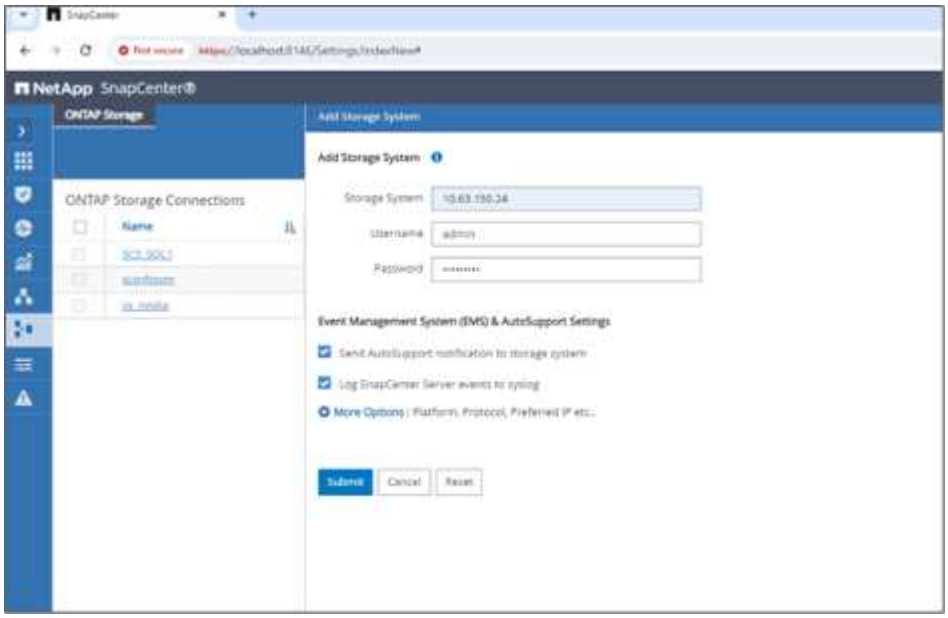
1. SnapCenterコマンドの実行に使用するホストを設定します。



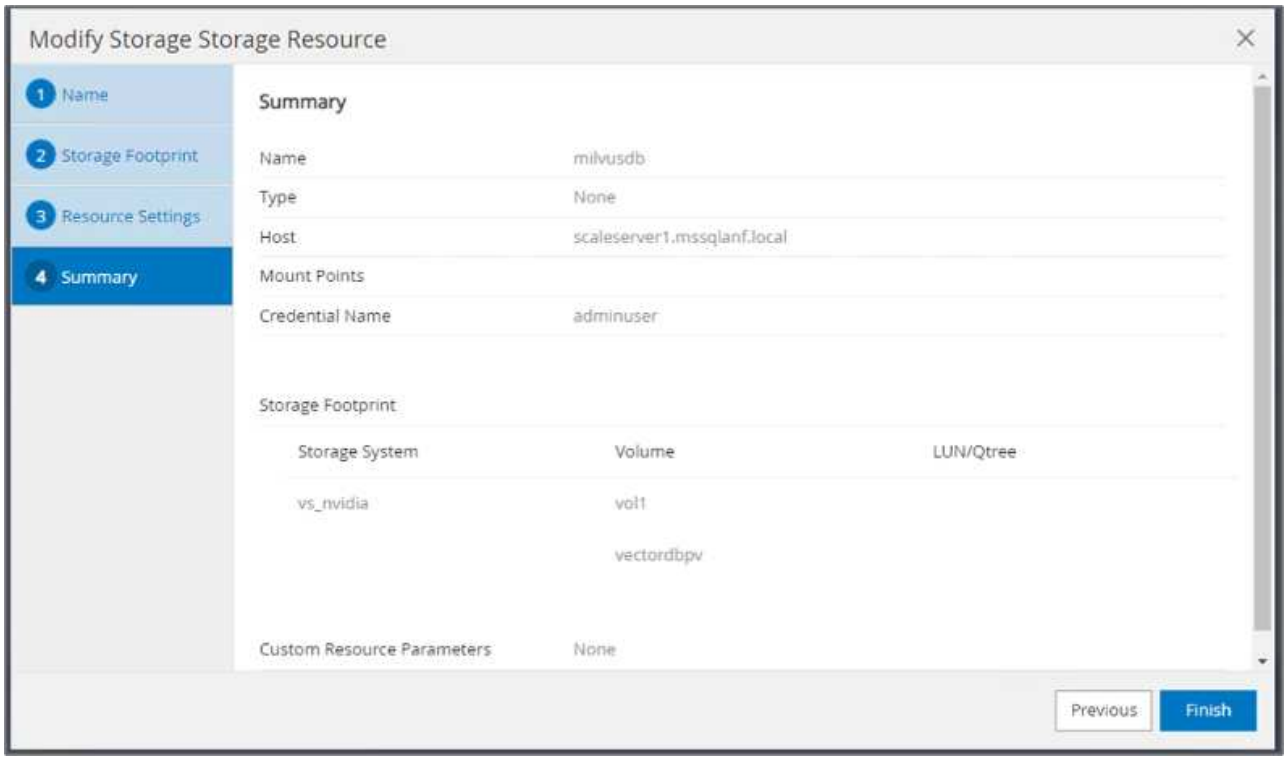
2. ストレージプラグインをインストールして設定します。追加したホストから、[その他のオプション]を選択します。ダウンロードしたストレージプラグインに移動してから選択します。 "NetApp Automation Store の略"。プラグインをインストールし、設定を保存します。



3. ストレージシステムとボリュームをセットアップします。[ストレージシステム]にストレージシステムを追加し、SVM（Storage Virtual Machine）を選択します。この例では、「vs_nvidia」を選択しています。



4. バックアップポリシーとカスタムスナップショット名を組み込んで、ベクターデータベースのリソースを確立します。
   - 整合グループバックアップをデフォルト値で有効にし、ファイルシステムの整合性を伴わないSnapCenterを有効にします。
   - [Storage Footprint]セクションで、ベクターデータベースの顧客データとMilvusクラスタデータに関連付けられているボリュームを選択します。この例では、「vol1」と「vectordbpv」です。
   - ベクターデータベース保護のポリシーを作成し、ポリシーを使用してベクターデータベースリソースを保護します。



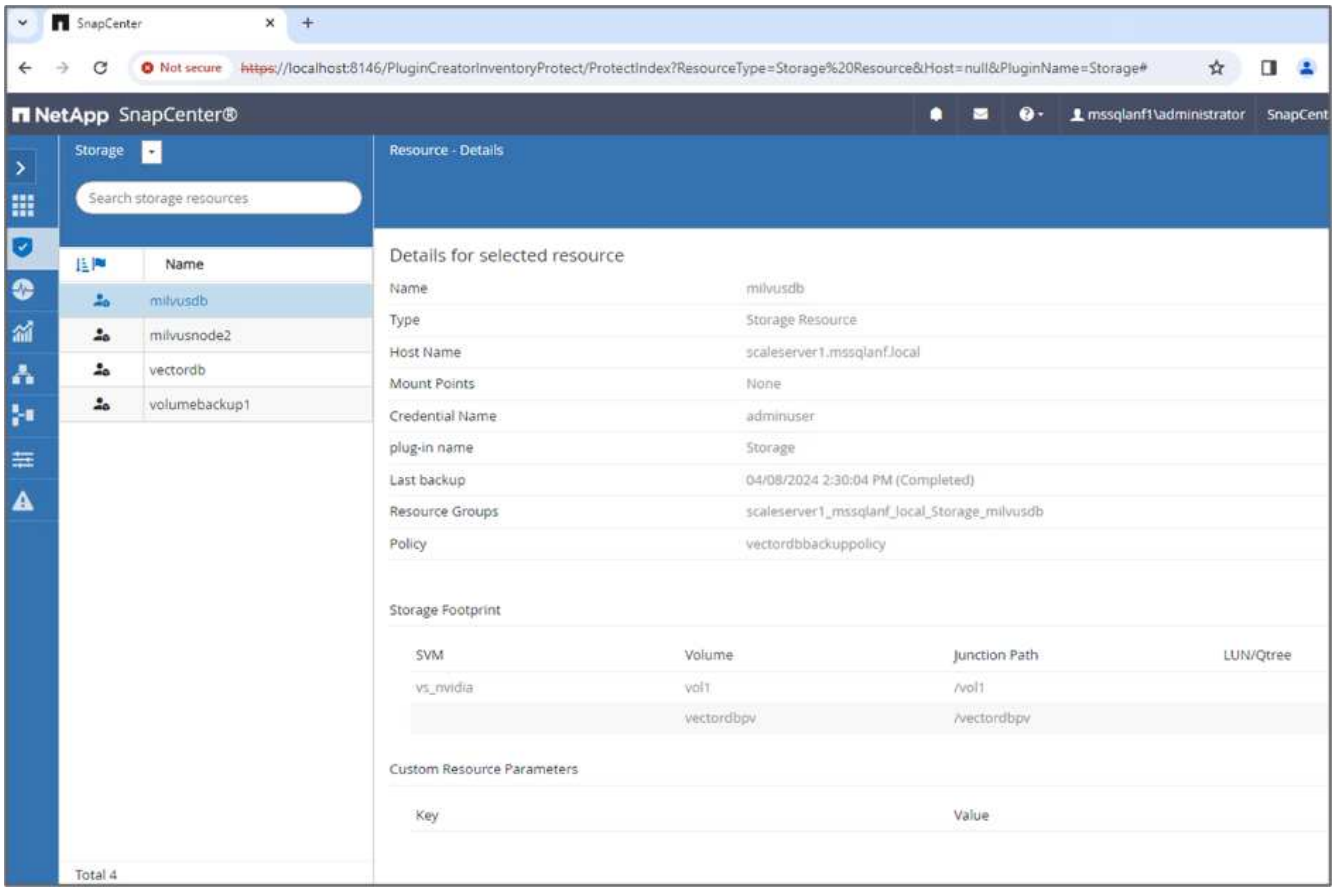
5. Pythonスクリプトを使用してS3 NASバケットにデータを挿入します。今回のケースでは、Milvusが提供するバックアップスクリプト「prepare_data_netapp.py」を変更し、「sync」コマンドを実行して、オペレーティングシステムからデータをフラッシュしました。

```
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. S3 NASバケット内のデータを確認します。この例では、タイムスタンプ「2024-04-08 21：22」のファイルが「prepare_data_netapp.py」スクリプトによって作成されています。

```
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. 「milvusdb」リソースの整合グループ（CG）Snapshotを使用してバックアップを開始する

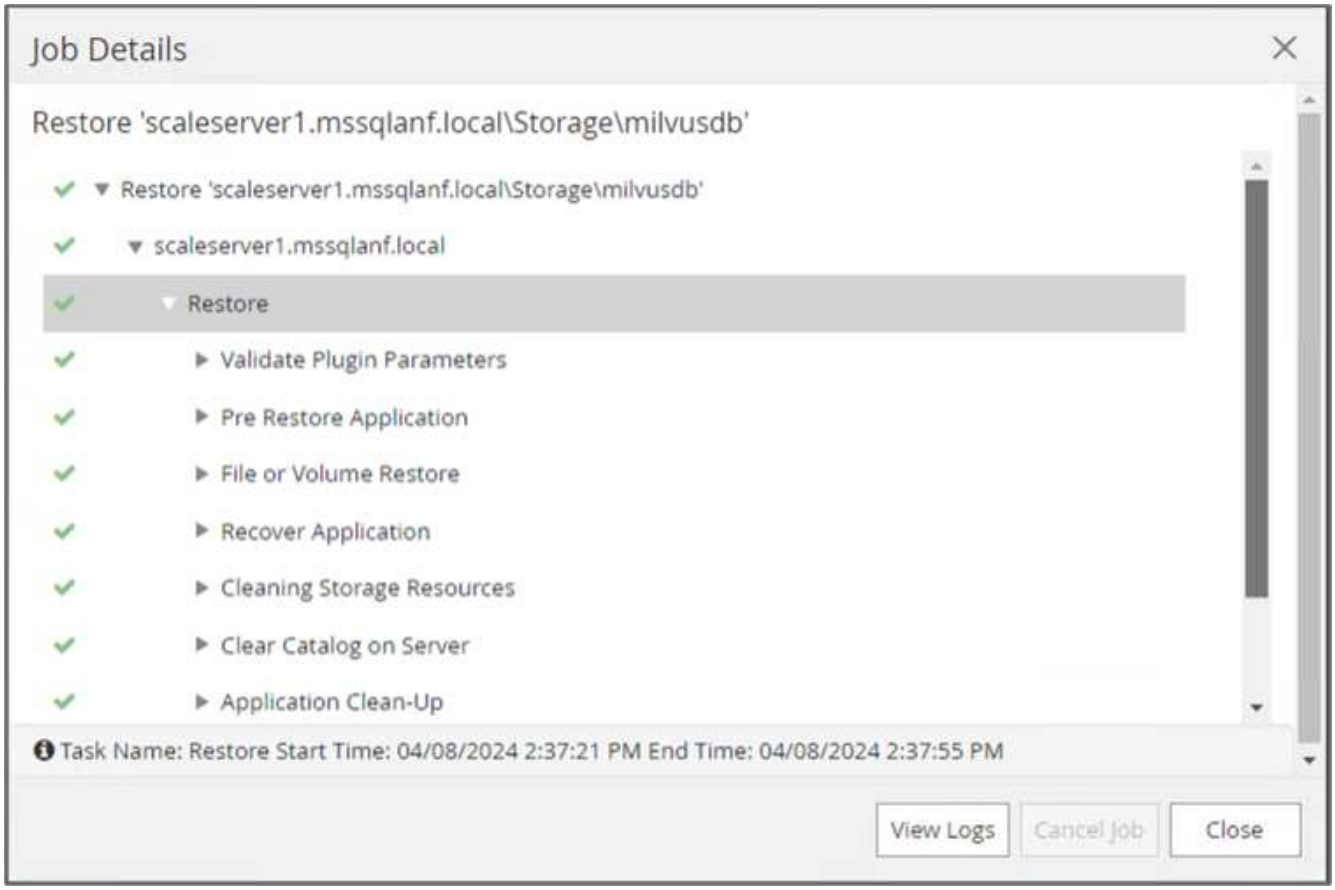


8. バックアップ機能をテストするために、バックアッププロセス後に新しいテーブルを追加するか、NFS（S3 NASバケット）から一部のデータを削除しました。

   このテストでは、バックアップ後に誰かが新しいコレクション、不要なコレクション、または不適切なコレクションを作成したシナリオを想像してください。このような場合は、新しいコレクションが追加される前に、ベクターデータベースをその状態に戻す必要があります。たとえば、「hello_milvus_netapp_sc_testnew」や「hello_milvus_netapp_sc_testnew2」などの新しいコレクションが挿入されています。

```
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. 前のSnapshotからS3 NASバケットのフルリストアを実行します。



10. Pythonスクリプトを使用して、「hello_milvus_netapp_sc_test」コレクションと「hello_milvus_netapp_sc_test2」コレクションのデータを検証します。

```
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. 不要または不適切な収集がデータベースに存在しないことを確認します。

```
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:~#
```

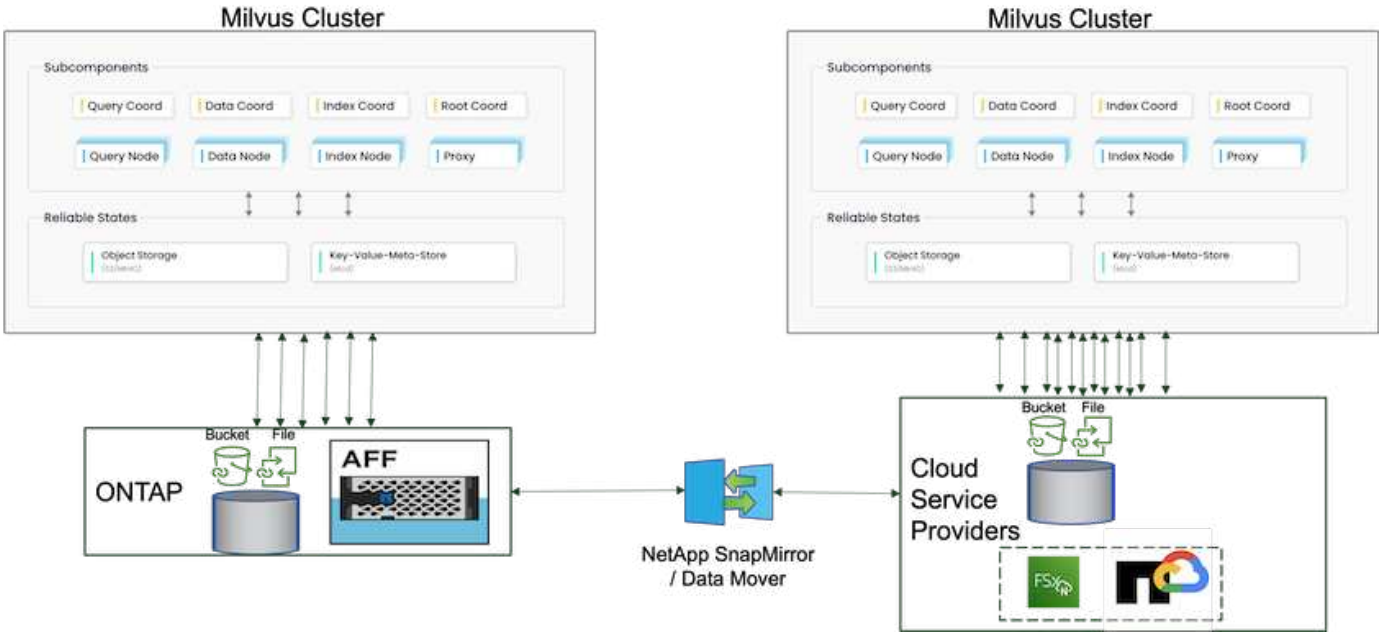
結論として、ネットアップのSnapCenterを使用してベクターデータベースのデータとONTAPにあるMilvusのデータを保護することは、特にデータの整合性が最も重視される業界（映画制作など）で、お客様に大きなメリットをもたらします。SnapCenterでは、整合性のあるバックアップを作成し、完全なデータリストアを実行できるため、組み込みのビデオファイルやオーディオファイルなどの重要なデータを、ハードドライブの障害やその他の問題による損失から確実に保護できます。これにより、業務の中断が防止されるだけでなく、多額の財務上の損失からも保護されます。

このセクションでは、ホストのセットアップ、ストレージプラグインのインストールと構成、カスタムスナップショット名を持つベクターデータベースのリソースの作成など、ONTAPに存在するデータを保護するためにSnapCenterを構成する方法を説明しました。また、整合グループのSnapshotを使用してバックアップを実行し、S3 NASバケット内のデータを検証する方法についても紹介しました。

さらに、バックアップ後に不要または不適切な収集が作成されるシナリオをシミュレートしました。このような場合、SnapCenterで以前のSnapshotからフルリストアを実行すると、新しいコレクションが追加される前の状態にベクターデータベースを戻すことができるため、データベースの整合性が維持されます。特定の時点にデータをリストアするこの機能は、お客様にとって非常に貴重なものであり、データのセキュリティだけでなく、適切に保持されていることを保証します。このように、ネットアップのSnapCenter製品は、データの保護と管理のための堅牢で信頼性の高い解決策をお客様に提供します。

NetApp SnapMirrorを使用したディザスタリカバリ

NetApp SnapMirrorを使用したディザスタリカバリ



ディザスタリカバリは、ベクトルデータベースの整合性と可用性を維持するために非常に重要です。特に、ディザスタリカバリは、高次元データの管理と複雑な類似性検索の実行において重要です。適切に計画され、実装されたディザスタリカバリ戦略により、ハードウェア障害、自然災害、サイバー攻撃などの予期しないインシデントが発生した場合でも、データの損失や侵害が発生しないようにします。これは、データの損失や破損が業務の中断や財務上の損失につながる可能性があるベクターデータベースに依存するアプリケーションで特に重要です。さらに、堅牢なディザスタリカバリ計画により、ダウンタイムを最小限に抑え、サービスの迅速な復旧を可能にすることで、ビジネス継続性も確保されます。これは、地理的に離れた場所にあるNetAppデータ・レプリケーション製品SnapMirror'定期的なバックアップ'フェイルオーバー・メカニズムによって実現されます。したがって、ディザスタリカバリは単なる保護対策ではなく、責任ある効率的なベクターデータベース管理の重要な要素です。

ネットアップのSnapMirrorは、NetApp ONTAPストレージコントローラ間のデータレプリケーションを提供します。主にディザスタリカバリ（DR）ソリューションやハイブリッドソリューションに使用されます。ベクターデータベースのコンテキストでは、オンプレミス環境とクラウド環境の間でスムーズにデータを移行できます。この移行は、データ変換やアプリケーションのリファクタリングを必要としないため、複数のプラットフォームにわたるデータ管理の効率性と柔軟性が向上します。

ベクターデータベースのシナリオでNetApp Hybrid解決策を使用すると、より多くの利点が得られます。

1. 拡張性：ネットアップのハイブリッドクラウド解決策は、要件に応じてリソースを拡張する機能を提供します。ピーク時や予期しない負荷に備えて、通常の予測可能なワークロードや、Amazon FSxN for NetApp ONTAPやGoogle Cloud NetApp Volume（GCNV）などのクラウドリソースにオンプレミスリソースを活用できます。
2. コスト効率：ネットアップのハイブリッドクラウドモデルでは、通常のワークロードにオンプレミスのリソースを使用し、必要なときにのみクラウドリソースに料金を支払うことで、コストを最適化できます。この従量課金制モデルは、NetApp instaclustrサービスを使用すると、非常に対費用効果が高くなります。オンプレミスや大手クラウドサービスプロバイダには、instaclustrがサポートとコンサルティングを提供します。
3. 柔軟性：ネットアップのハイブリッドクラウドなら、データの処理場所を柔軟に選択できます。たとえば、より強力なハードウェアがあり、クラウドでの処理の負荷が低い複雑なベクトル操作をオンプレミスで実行することもできます。
4. ビジネス継続性：災害発生時にデータをNetAppハイブリッドクラウドに配置することで、ビジネス継続性を確保できます。オンプレミスのリソースに影響が出た場合は、迅速にクラウドに切り替えることができ

ます。NetApp SnapMirrorを活用して、オンプレミスとクラウドの間でデータを移動できます。

5. イノベーション：ネットアップのハイブリッドクラウドソリューションは、最先端のクラウドサービスとテクノロジを利用できるようにすることで、イノベーションの加速も可能にします。Amazon FSxN for NetApp ONTAP、Azure NetApp Files、Google Cloud NetApp Volumeなど、クラウドにおけるNetAppのイノベーションは、クラウドサービスプロバイダにとって革新的な製品であり、好まれるNASです。

Vectorデータベースのパフォーマンス検証

パフォーマンスの検証

パフォーマンス検証は、ベクターデータベースとストレージシステムの両方で重要な役割を果たし、最適な運用と効率的なリソース使用率を確保するための重要な要素となります。ベクトルデータベースは、高次元データを処理し、類似性検索を実行することで知られており、複雑なクエリを迅速かつ正確に処理するために、高いパフォーマンスレベルを維持する必要があります。パフォーマンス検証は、ボトルネックを特定し、構成を微調整し、サービスを低下させることなく予想される負荷にシステムが対応できることを確認するのに役立ちます。同様に、ストレージシステムにおいても、レイテンシの問題やシステム全体のパフォーマンスに影響を与えるボトルネックが発生することなく、データを効率的に格納、取得するためには、パフォーマンスの検証が不可欠です。また、ストレージインフラのアップグレードや変更に必要な情報に基づいて意思決定を下すのにも役立ちます。したがって、パフォーマンス検証はシステム管理の重要な側面であり、高いサービス品質、運用効率、およびシステム全体の信頼性の維持に大きく貢献します。

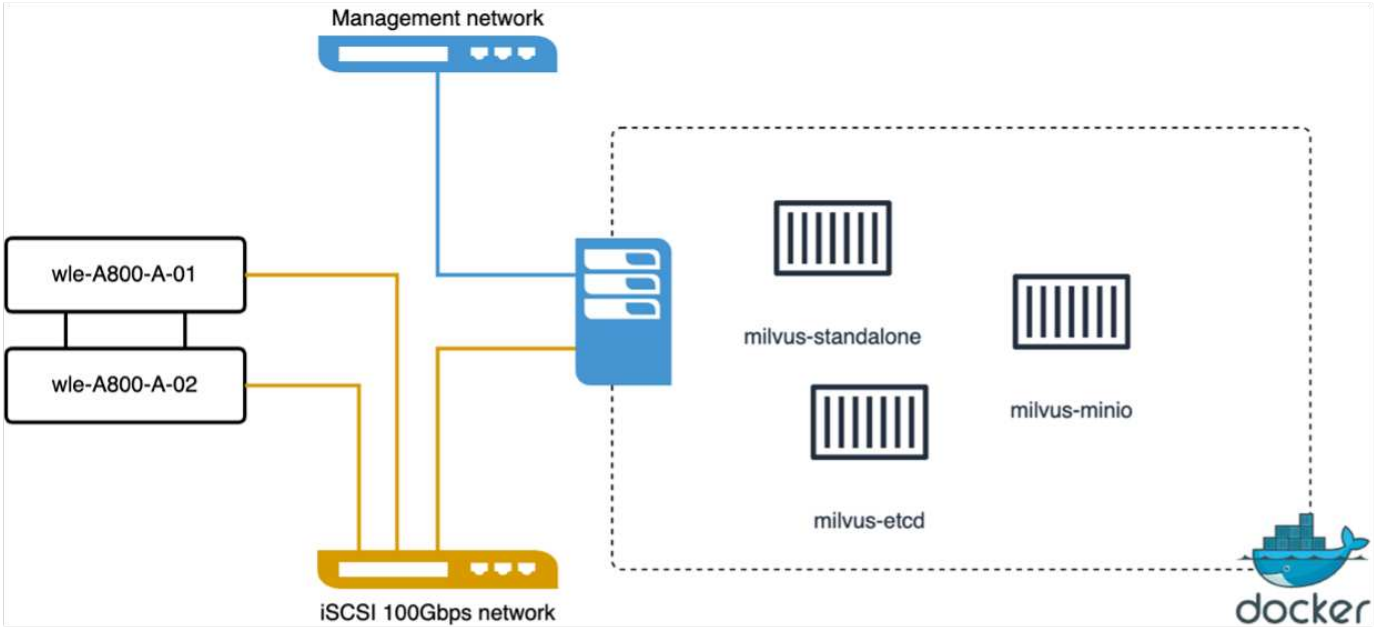
このセクションでは、Milvusやpgvecto.RSなどのベクターデータベースのパフォーマンス検証について、LLMライフサイクル内でのRAGおよび推論ワークロードをサポートする際のI/OプロファイルやNetAppストレージコントローラの動作などのストレージパフォーマンス特性に焦点を当てて詳しく説明します。これらのデータベースをONTAPストレージ解決策と組み合わせた場合のパフォーマンスの差別化要因を評価し、特定します。分析は、1秒あたりに処理されるクエリ数（QPS）などの主要パフォーマンス指標に基づいて行われます。

ミルバスに使用されている方法と進捗状況を以下で確認してください。

詳細	**Milvus**（スタンドアロンおよびクラスタ）	**postgres**（**pgvecto.rs**）
バージョン	2.3.2	0.2.0
ファイルシステム	iSCSI LUN上のXFS	
ワークロードジェネレータ	"VectorDB -ベンチ" –v0.0.5	
テエタセツト	LAIONデータセット *1千万件の埋め込み *768寸法 *最大300GBのデータセットサイズ	

VectorDB - Milvusスタンドアロンクラスタ付きベンチ

vectorDB-Benchを使用したmilvusスタンドアロンクラスタについて、次のパフォーマンス検証を行いました。
milvusスタンドアロンクラスタのネットワーク接続とサーバ接続を以下に示します。



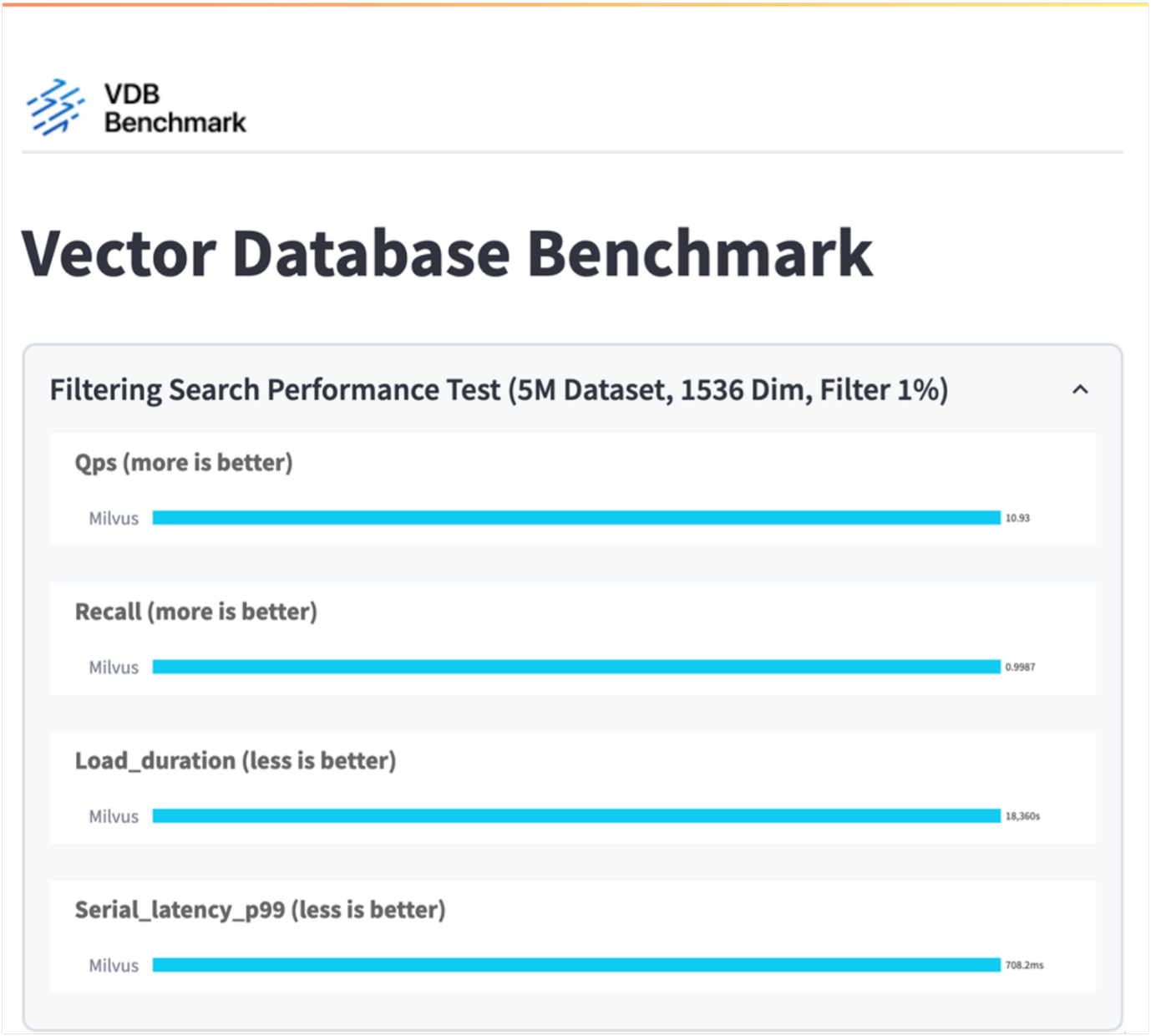
このセクションでは、Milvusスタンドアロンデータベースのテスト結果を共有します。
。 これらのテストのインデックスタイプとしてDiskANNを選択しました。
。 約100GBのデータセットのインデックスの取り込み、最適化、作成には約5時間かかりました。この期間のほとんどで、20コア(ハイパースレッディングが有効な場合は40 vCPUに相当)を搭載したMilvusサーバーは、最大CPU容量100%で動作していました。システムメモリサイズを超える大規模なデータセットでは、DiskANNが特に重要であることがわかりました。
。 クエリフェーズでは、0.9987のリコールで10.93のQueries Per Second (QPS)率が観察されました。クエリの99パーセンタイルレイテンシは708.2ミリ秒で測定されました。

ストレージに関して言えば、データベースは取り込み、挿入後の最適化、インデックス作成の各フェーズで1秒あたり約1、000ops/secの処理を実行していました。クエリフェーズでは、1秒あたり32、000件の処理が必要でした。

次のセクションでは、ストレージパフォーマンスの指標について説明します。

ワークロードフェーズ	メートル法	価値
データの取り込み および ポストインサートの最適化	IOPS	1、000未満
	レイテンシ	400 usecs未満
	ワークロード	読み取り/書き込み混在（主に書き込み）
	IOサイズ	64KB
クエリ	IOPS	最大32、000
	レイテンシ	400 usecs未満
	ワークロード	100%キャッシュ読み取り
	IOサイズ	主に8KB

vectorDB-benchの結果は以下のとおりです。

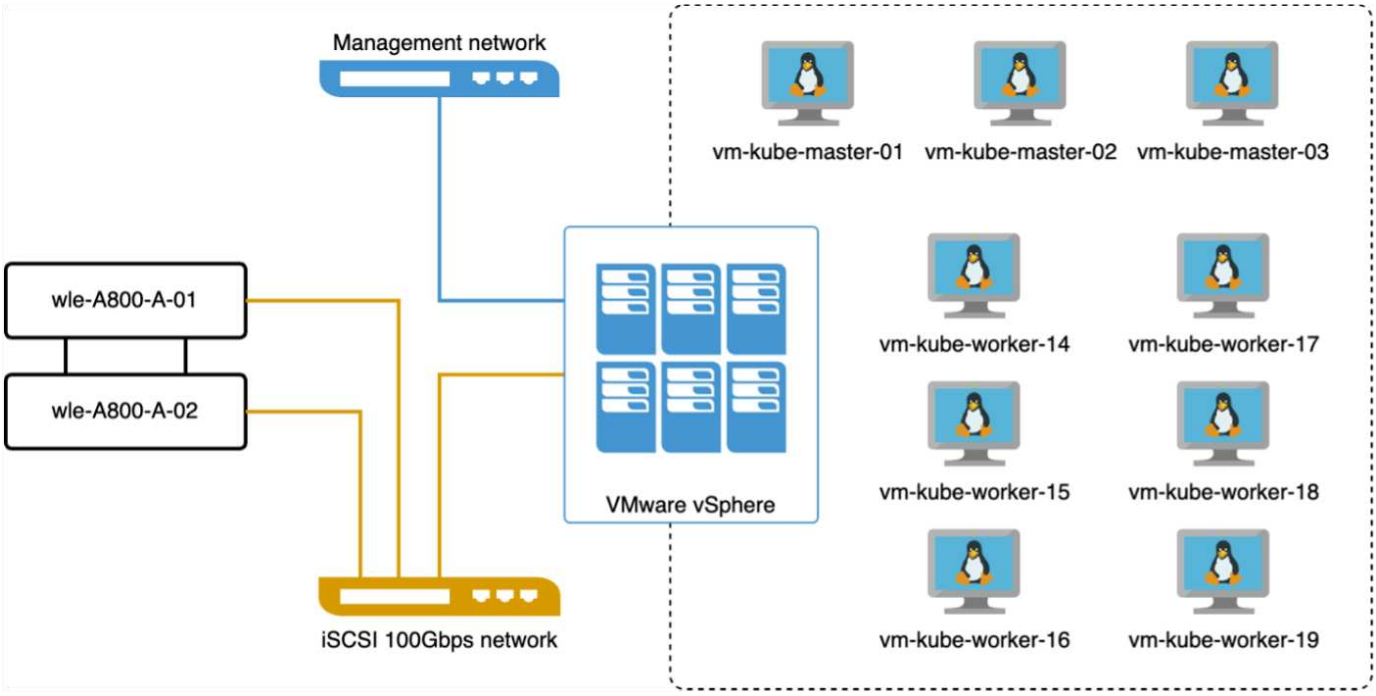


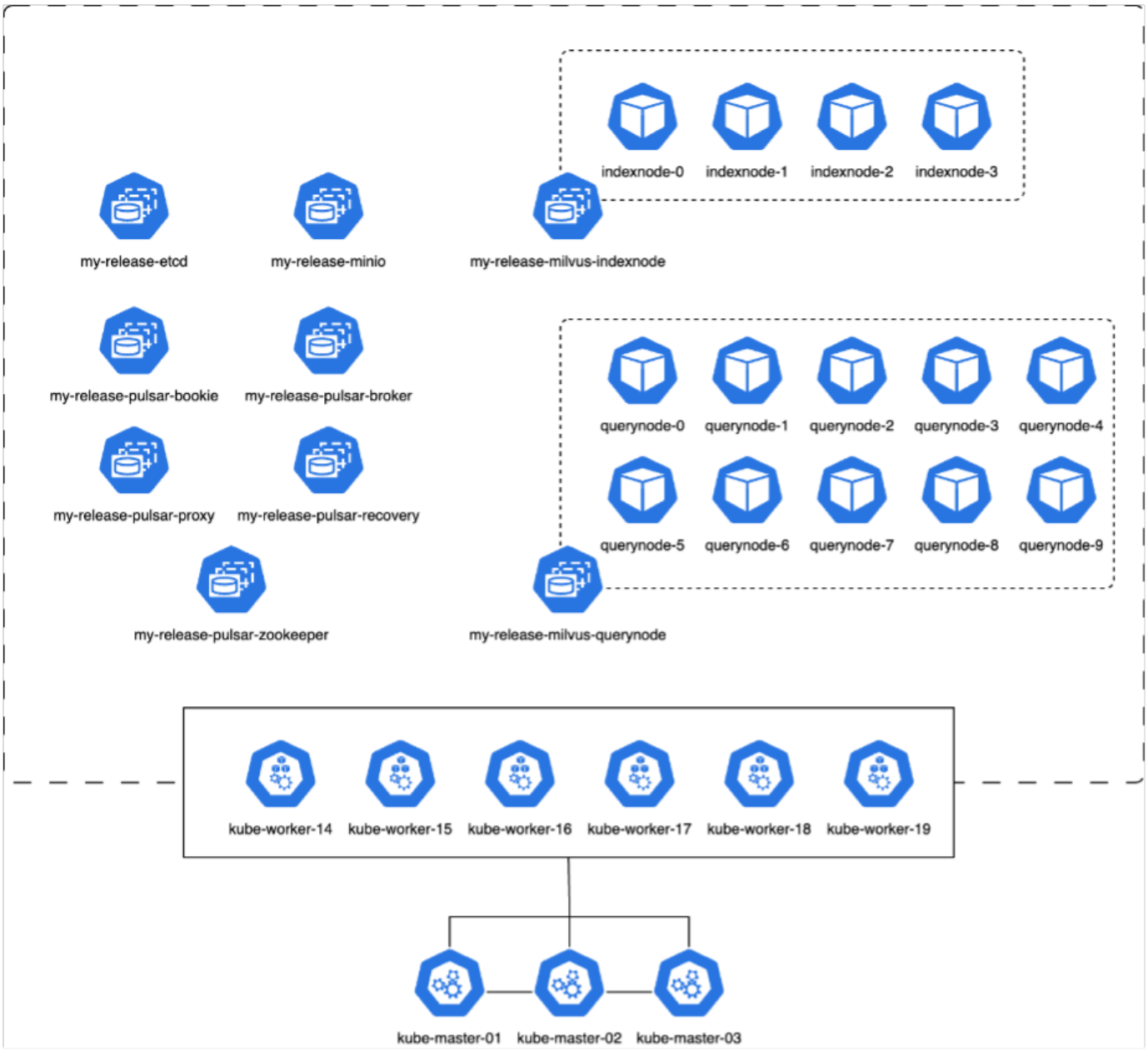
スタンドアロンMilvusインスタンスのパフォーマンス検証から、次元1536の500万ベクトルのデータセットをサポートするには、現在のセットアップでは不十分であることがわかります。ストレージには十分なリソースがあり、システムのボトルネックになっていないことが確認されました。

VectorDB -ミルバスクラスター付きベンチ

このセクションでは、Kubernetes環境内でのMilvusクラスタの導入について説明します。このKubernetesセットアップは、KubernetesのマスターノードとワーカーノードをホストするVMware vSphere環境の上に構築されました。

VMware vSphere環境とKubernetes環境の詳細については、以降のセクションで説明します。


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



このセクションでは、Milvusデータベースをテストした結果と観測結果を紹介します。
*使用されたインデックスタイプはDiskANNです。
*次の表は、次元1536で500万のベクトルを使用する場合の、スタンドアロン配置とクラスタ配置の比較を示しています。クラスタ環境では、データの取り込みと挿入後の最適化にかかる時間が短くなっていることがわかりました。クラスタ環境では、クエリの99パー センタイルレイテンシがスタンドアロンセットアップに比べて6分の1に短縮されました。
* Queries Per Second（QPS；1秒あたりのクエリ数）の割合はクラスタ環境では高くなりましたが、望ましいレベルではありませんでした。

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%

次の図は、ストレージクラスタのレイテンシや合計IOPS（1秒あたりの入出力処理数）など、さまざまなスト

レージ指標を示しています。

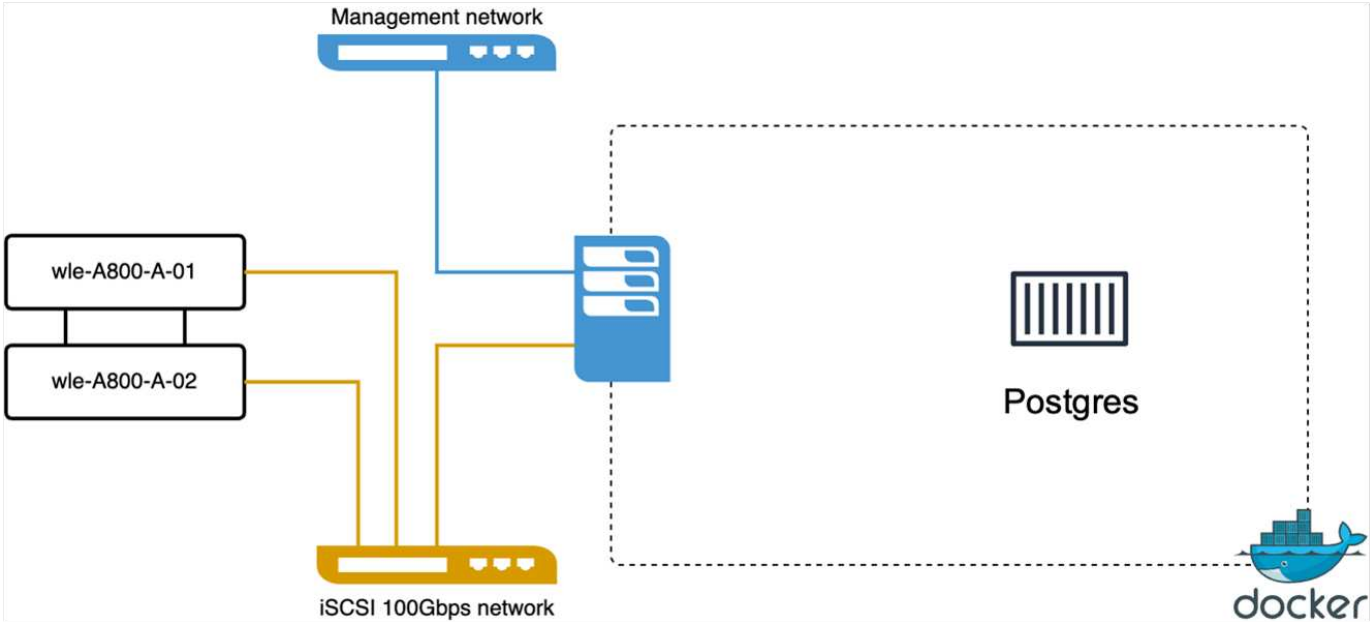
次のセクションでは、ストレージパフォーマンスの主要な指標について説明します。

ワークロードフェーズ	メートル法	価値
データの取り込み および ポストインサートの最適化	IOPS	1、000未満
	レイテンシ	400 usecs未満
	ワークロード	読み取り/書き込み混在（主に書き込み）
	IOサイズ	64KB
クエリ	IOPS	ピーク時147、000
	レイテンシ	400 usecs未満
	ワークロード	100%キャッシュ読み取り
	IOサイズ	主に8KB

スタンドアロンのMilvusクラスタとMilvusクラスタの両方のパフォーマンス検証に基づいて、ストレージI/Oプロファイルの詳細を提示します。
*スタンドアロン環境とクラスタ環境の両方で、I/Oプロファイルが一貫していることが確認されました。
*ピークIOPSの差は、クラスタ環境内のクライアント数が多いことが原因である可能性があります。

vectorDB - Postgresを使用したベンチ(pgvecto.rs)

VectorDB-Benchを使用して、PostgreSQL（pgvecto.rs）に対して次のアクションを実行しました。
PostgreSQL（特にpgvecto.rs）のネットワーク接続とサーバ接続に関する詳細は次のとおりです。

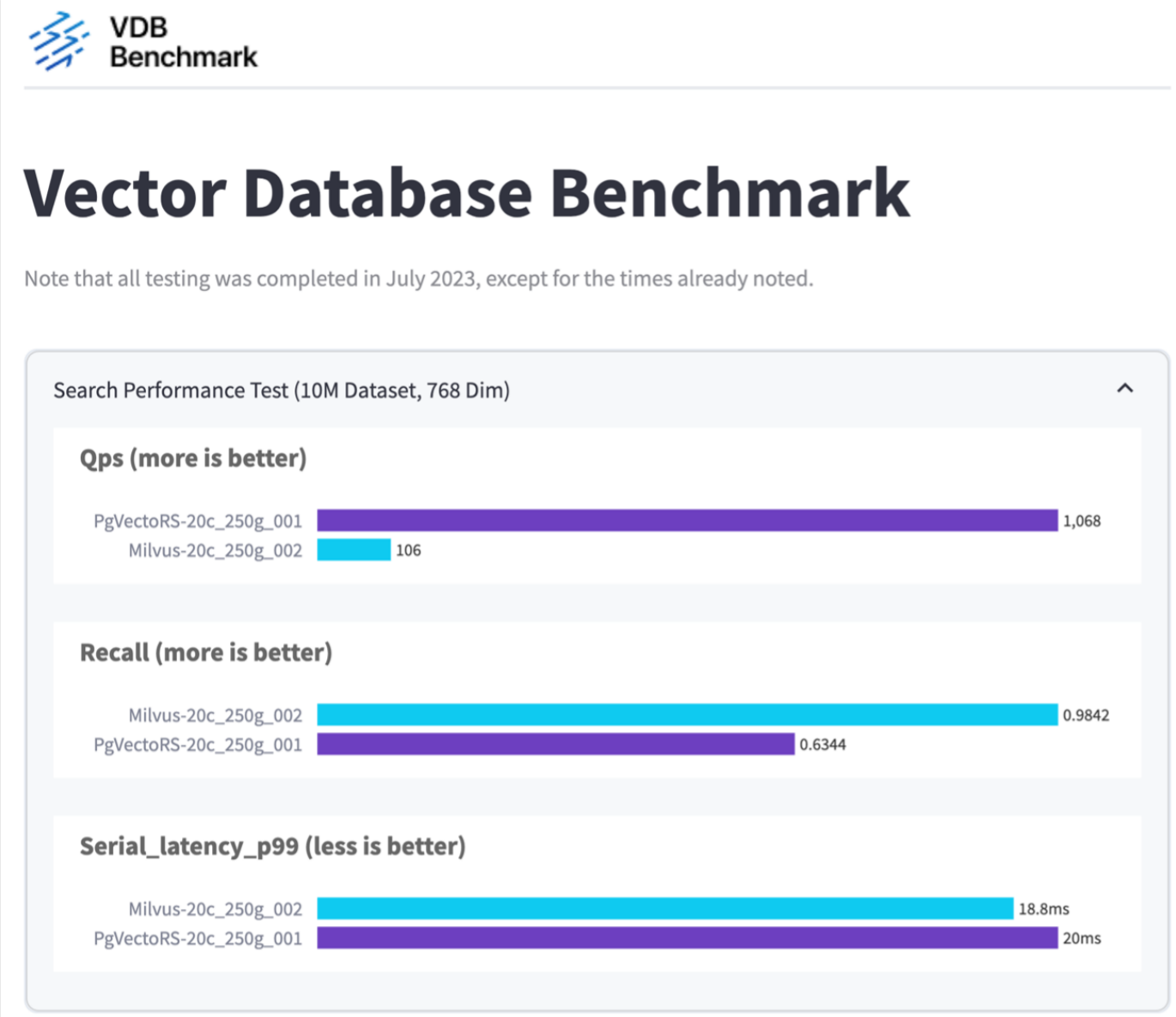


このセクションでは、pgvecto.rsを使用してPostgreSQLデータベースをテストした結果と結果を共有します。
*テストのインデックスタイプとしてHNSWを選択したのは、テスト時にpgvecto.rsでDiskANNを使用できなかったためです。
*データ取り込みフェーズでは、次元768の1000万ベクトルからなるCohereデータセットをロードしました。このプロセスには約4.5時間かかりました。
*クエリフェーズでは、1秒あたりのクエリ数（QPS）は1,068、リコールは0.6344でした。クエリの99パーセンタイルレイテンシは20ミリ秒で測定されました。ランタイムのほとんどで、クライアントCPUは100%の容量で動作していました。

次の図は、ストレージクラスタの合計IOPS（1秒あたりの入出力処理数）など、さまざまなストレージ指標を示しています。


```
 The following section presents the key storage performance metrics.
image:pgvecto_storage_perf_metrics.png["エラー：グラフィックイメージがありません"
]
```

VECTOR DBベンチでの**milvus**と**postgres**のパフォーマンス比較



VectorDBBenchを使用したMilvusおよびPostgreSQLのパフォーマンス検証に基づいて、次のことを確認しました。

- インデックスタイプ：HNSW
- データセット：768次元で1000万ベクトルのコア

pgvecto.rsは0.6344のリコールで1,068のQPSレートを達成し、Milvusは0.9842のリコールで106のQPSレートを達成しました。

クエリの精度が優先される場合、Milvusはpgvecto.rsよりもパフォーマンスが高く、クエリごとに関連する項目の割合が高くなります。ただし、1秒あたりのクエリ数がより重要な要素である場合、pgvecto.rsはMilvusを超えます。ただし、pgvecto.rsを介して取得されるデータの品質は低く、検索結果の約37%が無関係な項目であることに注意する必要があります。

パフォーマンス検証に基づく観察：

パフォーマンスの検証に基づいて、次のことを確認しました。

MilvusのI/Oプロファイルは、OracleのSLOBなどのOLTPワークロードによく似ています。ベンチマークは、データの取り込み、最適化後、クエリの3つのフェーズで構成されています。初期段階は主に64KBの書き込み処理で特徴付けられますが、クエリフェーズでは8KBの読み取りが主に行われます。ONTAPはMilvusのI/O負荷を適切に処理することを期待しています。

PostgreSQLのI/Oプロファイルでは、困難なストレージワークロードは発生しません。メモリ内の実装が現在進行中であるため、クエリフェーズ中にディスクI/Oを確認することはできませんでした。

DiskANNは、ストレージを差別化するための重要なテクノロジーとして登場しています。これにより、システムメモリ境界を越えたベクターDB検索の効率的なスケーリングが可能になります。ただし、HNSWなどのインメモリベクトルDBインデックスを使用して、ストレージパフォーマンスの差別化を確立することはほとんどありません。

また、インデックスタイプがHSNWの場合、クエリフェーズでストレージが重要な役割を果たしないことも注目に値します。HSNWは、RAGアプリケーションをサポートするベクターデータベースで最も重要な操作フェーズです。つまり、ストレージのパフォーマンスがこれらのアプリケーションの全体的なパフォーマンスに大きく影響することはありません。

PostgreSQLを使用したInstaclustrを使用したベクターデータベース: pgvector

PostgreSQLを使用したInstaclustrを使用したベクターデータベース: pgvector

このセクションでは、instaclustr productがpgvector機能でPostgreSQLとどのように統合されるかについて詳しく説明します。「PGVectorとPostgreSQL®を使用してLLMの精度とパフォーマンスを向上させる方法:埋め込みの概要とPGVectorの役割」の例を紹介します。を確認してください "ブログ" をクリックしてください。

Vectorデータベースのユースケース

Vectorデータベースのユースケース

このセクションでは、大規模言語モデルを使用したRetrieval Augmented GenerationとNetApp ITチャットボットの2つのユースケースについて説明します。

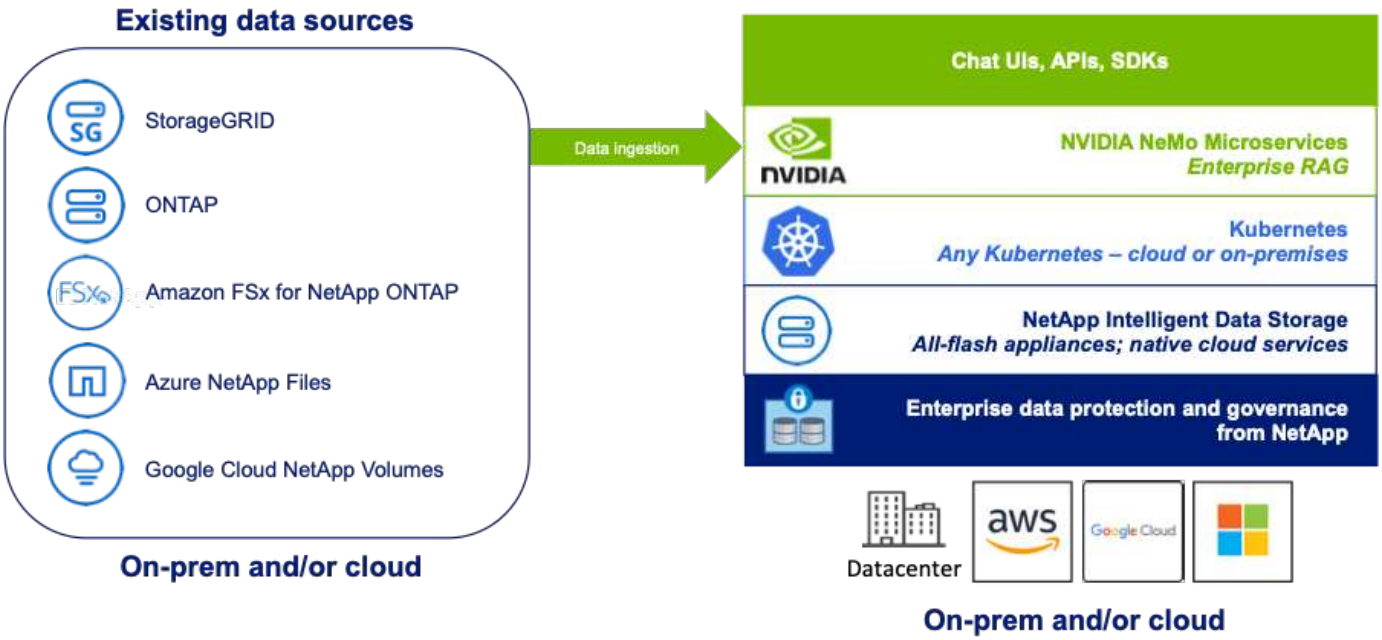
LLM（Large Language Models）を使用したRAG（Retrieval Augmented Generation）

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

NVIDIA Enterprise RAG LLM Operatorは、エンタープライズにRAGを実装するための便利なツールです。このオペレータを使用して、完全なRAGパイプラインを展開できます。RAGパイプラインは、ナレッジベースの埋め込みを格納するためのベクターデータベースとしてMilvusまたはpgvectoのいずれかを利用するようにカスタマイズできます。詳細については、ドキュメントを参照してください。

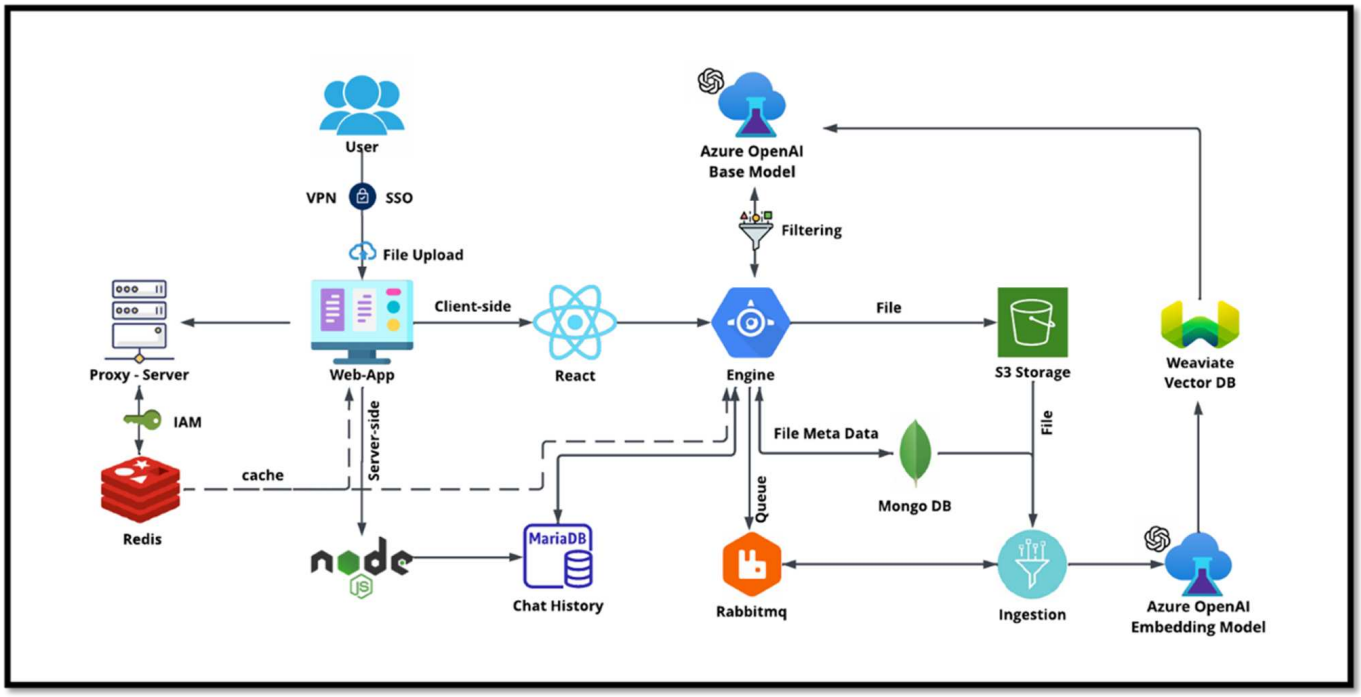
```
NetApp has validated an enterprise RAG architecture powered by the NVIDIA
Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog
post for more information and to see a demo. Figure 1 provides an overview
of this architecture.
```

図1）NVIDIA NemoマイクロサービスとNetAppを基盤とするエンタープライズRAG



NetApp ITチャットボットのユースケース

ネットアップのチャットボットは、ベクターデータベースのもう1つのリアルタイムユースケースとして機能します。この場合、NetAppプライベートOpenAIサンドボックスは、ネットアップの社内ユーザからのクエリを管理するための、効果的でセキュアで効率的なプラットフォームを提供します。厳格なセキュリティプロトコル、効率的なデータ管理システム、高度なAI処理機能を組み込むことで、SSO認証を介して組織内のユーザーの役割と責任に基づいて、ユーザーに対する高品質で正確な応答を保証します。このアーキテクチャは、高度なテクノロジーを統合して、ユーザー重視のインテリジェントなシステムを構築する可能性を強調しています。



ユースケースは、主に4つのセクションに分けることができます。

ユーザ認証と検証：

- ユーザークエリは、最初にNetAppシングルサインオン(SSO)プロセスを実行して、ユーザーのIDを確認します。
- 認証に成功すると、システムはVPN接続をチェックして、安全なデータ転送を保証します。

データ転送と処理：

- VPNが検証されると、データはNetAIChatまたはNetAICreate Webアプリケーションを介してMariaDBに送信されます。MariaDBは、ユーザデータの管理と保存に使用される高速で効率的なデータベースシステムです。
- 次に、MariaDBはその情報をNetApp Azureインスタンスに送信し、Azureインスタンスはユーザデータをを処理ユニットに接続します。

OpenAIとコンテンツフィルタリングとの相互作用：

- Azureインスタンスは、ユーザーの質問をコンテンツフィルタリングシステムに送信します。このシステムはクエリをクリーンアップし、処理の準備を行います。

- その後、クリーンアップされた入力がAzure OpenAIベースモデルに送信され、入力に基づいて応答が生成されます。

応答の生成と管理：

- ベースモデルからの応答は、正確でコンテンツ基準を満たしていることを確認するために最初にチェックされます。
- チェックに合格すると、応答がユーザに返送されます。このプロセスにより、ユーザーはクエリに対して明確で正確で適切な回答を受け取ることができます。

まとめ

まとめ

結論として、このドキュメントでは、NetAppストレージソリューションへのMilvusやpgvectorなどのベクターデータベースの導入と管理の包括的な概要を説明します。NetApp ONTAPとStorageGRIDオブジェクトストレージを活用するためのインフラガイドラインについて話し合い、ファイルとオブジェクトストアを使用してAWS FSx for NetApp ONTAPのMilvusデータベースを検証しました。

ネットアップのファイルオブジェクトとオブジェクトの二重性を検証し、ベクトルデータベース内のデータだけでなく、他のアプリケーションでもその有用性を実証しました。また、ネットアップのエンタープライズ管理製品であるSnapCenterが、ベクターデータベースデータのバックアップ、リストア、クローニングの機能を提供し、データの整合性と可用性を確保する方法についても説明しました。

また、ネットアップのハイブリッドクラウド解決策がオンプレミス環境とクラウド環境にわたってデータのレプリケーションと保護を提供し、シームレスでセキュアなデータ管理を実現する方法についても詳しく説明しています。NetApp ONTAP上のMilvusやpgvectoなどのベクターデータベースのパフォーマンス検証に関するインサイトを提供し、効率とスケーラビリティに関する貴重な情報を提供しました。

最後に、生成型AIの2つのユースケース、LLMを使用したRAGと、ネットアップ社内向けChatAIについて説明しました。これらの実用的な例は、このドキュメントで概説した概念と実践の実際の用途と利点を強調しています。本ドキュメントは、ベクトルデータベースの管理にネットアップの強力なストレージソリューションを活用したいと考えているすべての人を対象とした包括的なガイドです。

謝辞

このホワイトペーパーをNetAppのお客様やNetAppの分野に役立てるために、以下の寄稿者、フィードバックやコメントを提供してくださった方 々 に心から感謝いたします。

1. NetApp ONTAP AI & Analyticsテクニカルマーケティングエンジニアsathish Thyagarajan氏
2. ネットアップテクニカルマーケティングエンジニア、 Mike Oglesby 氏
3. NetAppシニアディレクターAj Mahajan氏
4. NetAppワークロードパフォーマンスエンジニアリングマネージャーJoe Scott氏
5. NetApp製品管理FSxシニアディレクターPuneet Dhawan氏
6. NetApp FSx製品チームシニアプロダクトマネージャーYuval Kalderon氏

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- Milvusドキュメント- https://milvus.io/docs/overview.md
- Milvusスタンドアロンドキュメント- https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- ネットアップの製品マニュアル
  https://www.netapp.com/support-and-training/documentation/
- instaclustr - "Instalclustrノトキユメント"

バージョン履歴

バージョン	日付	ドキュメントのバージョン履歴
バージョン 1.0 以降	2024年4月	初版リリース

付録A：values.yaml

付録A：values.yaml

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

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

# Global node selector
# If set, this will apply to all milvus components
# Individual components can be set to a different node selector
```

```
nodeSelector: {}

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

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

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

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

## Expose the Milvus service to be accessed from outside the cluster
(LoadBalancer service).
## or access it from within the cluster (ClusterIP service). Set the
service type and the port to serve it.
## ref: http://kubernetes.io/docs/user-guide/services/
##
service:
  type: ClusterIP
```

```
  port: 19530
  portName: milvus
  nodePort: ""
  annotations: {}
  labels: {}

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

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

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

serviceAccount:
  create: false
  name:
```

```
  annotations:
  labels:

metrics:
  enabled: true

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

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

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

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

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
```

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

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

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

```
  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
    mountPath: "/var/lib/milvus"
    ## If true, alertmanager will create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: true
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Persistent Volume Storage Class
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ##
      storageClass:
      accessModes: ReadWriteOnce
      size: 50Gi
      subPath: ""

proxy:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  http:
    enabled: true  # whether to enable http rest server
    debugMode:
      enabled: false
  # Mount a TLS secret into proxy pod
  tls:
```

```
    enabled: false
## when enabling proxy.tls, all items below should be uncommented and the
key and crt values should be populated.
#    enabled: true
#    secretName: milvus-tls
## expecting base64 encoded values here: i.e. $(cat tls.crt | base64 -w 0)
and $(cat tls.key | base64 -w 0)
#    key: LS0tLS1CRUdJTiBQU--REDUCT
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

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

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

queryCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
  replicas: 1  # Run Query Coordinator mode with replication disabled
```

```
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for query coordinator

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

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

indexCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
```

```
  replicas: 1   # Run Index Coordinator mode with replication disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for index coordinator

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

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

dataCoordinator:
  enabled: true
  # You can set the number of replicas greater than 1, only if enable
active standby
```

```
  replicas: 1           # Run Data Coordinator mode with replication
disabled
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for data coordinator

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

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

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

```
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  activeStandby:
    enabled: false  # Enable active-standby when you set multiple replicas
for Mixture coordinator

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

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

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

minio:
  enabled: false
  name: minio
  mode: distributed
  image:
    tag: "RELEASE.2023-03-20T20-16-18Z"
    pullPolicy: IfNotPresent
  accessKey: minioadmin
  secretKey: minioadmin
  existingSecret: ""
  bucketName: "milvus-bucket"
  rootPath: file
  useIAM: false
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
  podDisruptionBudget:
    enabled: false
  resources:
    requests:
      memory: 2Gi

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

  service:
    type: ClusterIP
    port: 9000

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

  livenessProbe:
    enabled: true
    initialDelaySeconds: 5
```

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

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

  startupProbe:
    enabled: true
    initialDelaySeconds: 0
    periodSeconds: 10
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 60

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

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

  service:
    type: ClusterIP
    port: 2379
    peerPort: 2380

  auth:
    rbac:
      enabled: false

  persistence:
```

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

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

  readinessProbe:
    enabled: true
    periodSeconds: 20
    timeoutSeconds: 10

  ## Enable auto compaction
  ## compaction by every 1000 revision
  ##
  autoCompactionMode: revision
  autoCompactionRetention: "1000"

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

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

pulsar:
  enabled: true
  name: pulsar

  fullnameOverride: ""
  persistence: true

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

  rbac:
```

```
    enabled: false
    psp: false
    limit_to_namespace: true

  affinity:
    anti_affinity: false

## enableAntiAffinity: no

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

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

  images:
    broker:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    autorecovery:
      repository: apachepulsar/pulsar
      tag: 2.8.2
      pullPolicy: IfNotPresent
    zookeeper:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    bookie:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
      tag: 2.8.2
    proxy:
      repository: apachepulsar/pulsar
      pullPolicy: IfNotPresent
```

```
    tag: 2.8.2
  pulsar_manager:
    repository: apachepulsar/pulsar-manager
    pullPolicy: IfNotPresent
    tag: v0.1.0

zookeeper:
  volumes:
    persistence: true
    data:
      name: data
      size: 20Gi   #SSD Required
      storageClassName: default
  resources:
    requests:
      memory: 1024Mi
      cpu: 0.3
  configData:
    PULSAR_MEM: >
      -Xms1024m
      -Xmx1024m
    PULSAR_GC: >
       -Dcom.sun.management.jmxremote
       -Djute.maxbuffer=10485760
       -XX:+ParallelRefProcEnabled
       -XX:+UnlockExperimentalVMOptions
       -XX:+DoEscapeAnalysis
       -XX:+DisableExplicitGC
       -XX:+PerfDisableSharedMem
       -Dzookeeper.forceSync=no
  pdb:
    usePolicy: false

bookkeeper:
  replicaCount: 3
  volumes:
    persistence: true
    journal:
      name: journal
      size: 100Gi
      storageClassName: default
    ledgers:
      name: ledgers
      size: 200Gi
      storageClassName: default
  resources:
```

```
      requests:
        memory: 2048Mi
        cpu: 1
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+UseG1GC -XX:MaxGCPauseMillis=10
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
        -XX:+PerfDisableSharedMem
        -XX:+PrintGCDetails
      nettyMaxFrameSizeBytes: "104867840"
    pdb:
      usePolicy: false

  broker:
    component: broker
    podMonitor:
      enabled: false
    replicaCount: 1
    resources:
      requests:
        memory: 4096Mi
        cpu: 1.5
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
```

```
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
        -XX:-ResizePLAB
        -XX:+ExitOnOutOfMemoryError
      maxMessageSize: "104857600"
      defaultRetentionTimeInMinutes: "10080"
      defaultRetentionSizeInMB: "-1"
      backlogQuotaDefaultLimitGB: "8"
      ttlDurationDefaultInSeconds: "259200"
      subscriptionExpirationTimeMinutes: "3"
      backlogQuotaDefaultRetentionPolicy: producer_exception
    pdb:
      usePolicy: false

  autorecovery:
    resources:
      requests:
        memory: 512Mi
        cpu: 1

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

  pulsar_manager:
    service:
```

```
      type: ClusterIP

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


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

kafka:
  enabled: false
  name: kafka
  replicaCount: 3
  image:
    repository: bitnami/kafka
    tag: 3.1.0-debian-10-r52
  ## Increase graceful termination for kafka graceful shutdown
  terminationGracePeriodSeconds: "90"
  pdb:
    create: false

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

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

```
    value: "5242880"
  - name: KAFKA_CFG_LOG_ROLL_HOURS
    value: "24"

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

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

    ## Prometheus JMX exporter: exposes the majority of Kafkas metrics
    jmx:
      enabled: false
      image:
        repository: bitnami/jmx-exporter
        tag: 0.16.1-debian-10-r245

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

  service:
    type: ClusterIP
    ports:
      client: 9092

  zookeeper:
    enabled: true
    replicaCount: 3

###################################
# External S3
# - these configs are only used when `externalS3.enabled` is true
###################################
externalS3:
```

```
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
  useSSL: false
  bucketName: "milvusdbvol1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false

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

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

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

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

付録B：prepare_data_netapp_new.py

付録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:~#
```

付録C：verify_data_netapp.py

付録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:~#
```

付録D：docker-compose.yml

付録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
```

著作権に関する情報

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータ ソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用権を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用権については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、http://www.netapp.com/TMに記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。