NetAppによるベクトルデータベースソリューション

NetApp artificial intelligence solutions

NetApp
August 18, 2025

目次

NetAppによるベクトルデータベースソリューション

NetAppによるベクトル データベース ソリューション

NetApp 、Karthikeyan Nagalingam 氏と Rodrigo Nascimento 氏

このドキュメントでは、NetApp 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 拡張機能である pgvecto の導入と管理について詳しく説明します。 NetApp ONTAPおよびStorageGRIDオブジェクトストレージを使用するためのインフラストラクチャガイドラインの詳細を説明し、AWS FSx 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、NetApp のファイルとオブジェクトの二重性と、ベクトル データベースおよびベクトル埋め込みをサポートするアプリケーションに対するその有用性について説明します。これは、ベクター データベースのバックアップと復元機能を提供し、データの整合性と可用性を確保する、NetApp のエンタープライズ管理製品であるSnapCenterの機能に重点を置いています。このドキュメントでは、NetApp のハイブリッド クラウド ソリューションについてさらに詳しく説明し、オンプレミスとクラウド環境にわたるデータのレプリケーションと保護におけるその役割について説明します。これには、 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洞察が含まれており、生成 AI に関する 2 つの実用的な使用例 (LLM を使用した RAG と NetApp の内部 ChatAI) で締めくくられています。このドキュメントは、NetApp のストレージ ソリューションを活用してベクター データベースを管理するための包括的なガイドとして役立ちます。

リファレンス アーキテクチャは、次の点に重点を置いています。

1. "はじめに"
2. "ソリューションの概要"
3. "ベクターデータベース"
4. "技術要件"
5. "展開手順"
6. "ソリューション検証の概要"
    - "オンプレミスでの Kubernetes を使用した Milvus クラスターのセットアップ"
    - Milvus with Amazon FSx ONTAP for NetApp ONTAP – ファイルとオブジェクトの二重性
    - "NetApp SnapCenterを使用したベクトル データベース保護。"
    - "NetApp SnapMirrorを使用した災害復旧"
    - "パフォーマンス検証"
7. "PostgreSQL を使用した Instaclustr によるベクターデータベース: pgvector"
8. "ベクターデータベースのユースケース"

9. "まとめ"
10. "付録A: values.yaml"
11. "付録 B: prepare_data_netapp_new.py"
12. "付録 C: verify_data_netapp.py"
13. "付録D: docker-compose.yml"

はじめに

このセクションでは、 NetAppのベクトル データベース ソリューションについて紹介します。

はじめに

ベクター データベースは、大規模言語モデル (LLM) と生成型人工知能 (AI) におけるセマンティック検索の複雑さを処理するために設計された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、ラベルやタグではなくデータ自体のコンテンツを使用して、画像、ビデオ、テキスト、オーディオ、その他の非構造化データなど、さまざまな種類のデータを処理および検索できます。

リレーショナル データベース管理システム (RDBMS) の限界、特に AI アプリケーションで一般的な高次元データ表現や非構造化データに関する問題点は、十分に文書化されています。 RDBMS では、データをより管理しやすい構造にフラット化するという時間がかかり、エラーが発生しやすいプロセスが必要になることが多く、検索の遅延や非効率につながります。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ソリューションを提供し、AI アプリケーションの進歩を促進します。

このドキュメントは、現在ベクター データベースを使用しているか、使用を計画しているお客様向けの包括的なガイドとして機能し、 NetApp ONTAP、 NetApp StorageGRID、 Amazon FSx ONTAP for NetApp ONTAP、 SnapCenterなどのプラットフォームでベクター データベースを活用するためのベスト プラクティスを詳しく説明します。ここで提供されるコンテンツは、さまざまなトピックをカバーしています。

- NetApp ONTAPおよびStorageGRIDオブジェクト ストレージを通じてNetAppストレージによって提供される、Milvus などのベクター データベース向けのインフラストラクチャ ガイドライン。
- ファイルおよびオブジェクト ストアを介して AWS FSx ONTAPの Milvus データベースを検証します。
- NetApp のファイルとオブジェクトの二重性を詳しく調べ、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示します。
- NetApp のデータ保護管理製品SnapCenter が、ベクター データベース データのバックアップと復元機能をどのように提供するかについて説明します。
- NetApp のハイブリッド クラウドがオンプレミスとクラウド環境全体でデータ レプリケーションと保護を提供する仕組みについて説明します。
- NetApp ONTAP上の Milvus や pgvector などのベクトル データベースのパフォーマンス検証に関する洞察を提供します。
- 2 つの具体的な使用例: 大規模言語モデル (LLM) を使用した検索拡張生成 (RAG) とNetApp IT チームの ChatAI。これにより、概説した概念と実践の実際的な例が提供されます。

ソリューションの概要

このセクションでは、 NetAppベクトル データベース ソリューションの概要を説明します。

ソリューションの概要

このソリューションは、ベクター データベースのお客様が直面する課題に対処するためにNetAppが提供する独自の利点と機能を紹介します。 NetApp ONTAP、 StorageGRID、NetApp のクラウド ソリューション、SnapCenterを活用することで、お客様はビジネス運営に大きな価値を加えることができます。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 NetApp の製品は、ストレージとコンピューティングの分離を可能にし、特定の要件に基づいて最適なリソース利用を実現します。この柔軟性により、お客様はNetAppストレージ ソリューションを使用してストレージを独自に拡張できるようになります。
- NetApp のストレージ コントローラを活用することで、顧客は NFS および S3 プロトコルを使用してベクター データベースにデータを効率的に提供できます。これらのプロトコルは、顧客データの保存を容易にし、ベクター データベース インデックスを管理するため、ファイルやオブジェクトの方法を通じてアクセスされるデータの複数のコピーが不要になります。
- NetApp ONTAP は、AWS、Azure、Google Cloud などの主要なクラウド サービス プロバイダー全体で NAS およびオブジェクト ストレージのネイティブ サポートを提供します。この幅広い互換性により、シームレスな統合が保証され、顧客データのモビリティ、グローバルなアクセス性、災害復旧、動的なスケーラビリティ、および高パフォーマンスが実現します。
- NetApp の強力なデータ管理機能により、お客様は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ことを確信できます。 NetApp はデータ セキュリティを最優先し、お客様の貴重な情報の安全性と整合性に関して安心を提供します。

ベクターデータベース

このセクションでは、 NetApp AI ソリューションにおけるベクター データベースの定義と使用について説明します。

ベクターデータベース

ベクター データベースは、機械学習モデルからの埋め込みを使用して非構造化データを処理、インデックス作成、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。この独自の構造により、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できるようになります。

ベクター データベースの主な機能の 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、データベースが特定の入力に類似するデータ ポイントを識別する類似性検索や、標準から大きく逸脱するデータ ポイントを見つけ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付きデータの処理に適しています。このタイプのデータは、特定の IT システム内で何がいつ起こったか、その順序や他のすべてのイベントとの関係についての情報を提供します。この時間的データの処理と分析の機能により、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** 向けベクトル データベースの利点**:**

- 高次元検索: ベクター データベースは、AI および ML アプリケーションで生成されることが多い高次元データの管理と取得に優れています。
- スケーラビリティ: 大量のデータを効率的に処理して処理できるため、AI および ML プロジェクトの成長と拡大をサポートします。
- 柔軟性: ベクター データベースは高度な柔軟性を備えており、さまざまなデータ タイプと構造に対応できます。
- パフォーマンス: AI および ML 操作の速度と効率に不可欠な、高パフォーマンスのデータ管理と取得を提供します。
- カスタマイズ可能なインデックス作成: Vector 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最適化されたデータの編成と取得を可能にします。

ベクター データベースと使用例。

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使用例の詳細について説明します。

ファイスと**ScaNN**

これらは、ベクトル検索の分野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、ベクター データの管理と検索に役立つ機能を提供するため、データ管理のこの専門分野では非常に貴重なリソースとなります。

Elasticsearch

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、最近ベクター検索機能が組み込まれました。この新しい機能により機能が強化され、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なります。

松ぼっくり

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。インデックス機能では密ベクトルと疎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ます。その主な強みの 1 つは、従来の検索方法と AI ベースの高密度ベクトル検索を組み合わせて、両方の長所を活用したハイブリッド検索アプローチを作成できることです。

Pinecone は主にクラウドベースで、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 などのさまざまなプラットフォームと適切に統合されます。 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) サービスとして利用できることに注意することが重要です。

Pinecone は高度な機能を備えているため、高次元検索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、対応できるため、サイバーセキュリティ業界に特に適してい ます。

彩度

これは、4 つの主な機能を持つ Core-API を備えたベクター データベースであり、その 1 つにメモリ内ドキュメント ベクター ストアが含まれています。また、Face Transformers ライブラリを利用してドキュメントをベクトル化し、機能性と汎用性を高めています。 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基づい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いるため、オーディオベースの検索エンジン、音楽推奨システム、その他のオーディオ関連のユースケースに最適です。

ウィービエイト

これは多目的ベクター データベースであり、ユーザーは組み込みモジュールまたはカスタム モジュールを使用してコンテンツをベクター化することができ、特定のニーズに基づいた柔軟性が得られます。さまざまな展開設定に対応し、完全に管理されたソリューションとセルフホスト型ソリューションの両方を提供します。

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保存して、データ処理機能を強化することです。 ERP システムにおける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で広く使用されています。電子商取引分野では、検索エンジンや推奨エンジンを強化します。 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脅威分析にも使用されており、複数のドメインにわたる汎用性を示しています。

レディス

Redis は、高速なインメモリ ストレージで知られる高性能なベクター データベースであり、読み取り/書き込み操作のレイテンシが低いことが特長です。そのため、迅速なデータ アクセスを必要とする推奨システム、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、ベクトル間の距離を計算したり、交差や結合を見つけたりするなどのベクトル演算も提供します。これらの機能は、類似性検索、クラスタリング、コンテンツベースの推奨システムに特に役立ちます。

スケーラビリティと可用性の面では、Redis は高スループットのワークロードの処理に優れており、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 タイプとも適切に統合されます。 Redis には、リアルタイム更新のための Publish/Subscribe (Pub/Sub) 機能が含まれており、リアルタイム ベクトルの管理に役立ちます。さらに、Redis は軽量で使い方が簡単なため、ベクター データを管理するためのユーザーフレンドリーなソリューションになります。

ミルバス

これは、MongoDB と同様に、ドキュメント ストアのような API を提供する多目的ベクター データベースです。幅広いデータ タイプをサポートしていることが特徴で、データ サイエンスや機械学習の分野で人気のある選択肢となっています。

Milvus のユニークな機能の 1 つは、マルチベクトル化機能です。この機能により、ユーザーは実行時に検索に使用するベクトルの種類を指定できます。さらに、Faiss などの他のライブラリの上位にあるライブラリである Knowwhere を使用して、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、電子商取引、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最適なツールになります。自然言語処理の分野では、Milvus はドキュメントのクラスタリング、セマンティック検索、質問応答システムに使用されます。

このソリューションでは、ソリューション検証に milvus を選択しました。パフォーマンスのために、milvus と postgres(pgvecto.rs) の両方を使用しました。

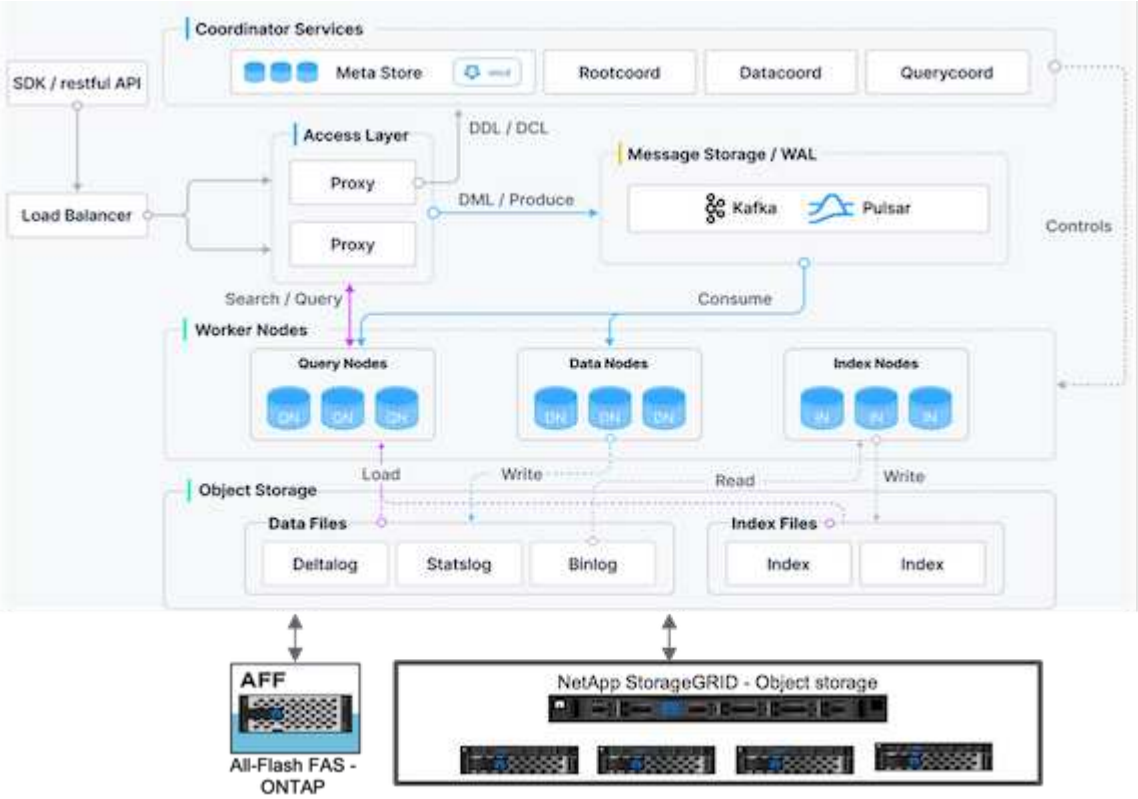
このソリューションに **Milvus** を選んだ理由は何ですか**?**

- オープンソース: Milvus はオープンソースのベクター データベースであり、コミュニティ主導の開発と改善を促進します。
- AI 統合: 埋め込み類似性検索と AI アプリケーションを活用して、ベクター データベースの機能を強化します。
- 大容量処理: Milvus には、ディープ ニューラル ネットワーク (DNN) および機械学習 (ML) モデルによって

生成された 10 億を超える埋め込みベクトルを保存、インデックス作成、管理する機能があります。

- ユーザーフレンドリー: セットアップは 1 分もかからず簡単に使用できます。 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。
- 速度: 他のいくつかの代替品よりも最大 10 倍高速な、非常に高速な取得速度を提供します。
- スケーラビリティと可用性: Milvus は非常にスケーラブルで、必要に応じてスケールアップおよびスケールアウトするオプションがあります。
- 機能が豊富: さまざまなデータ タイプ、属性フィルタリング、ユーザー定義関数 (UDF) のサポート、構成可能な一貫性レベル、移動時間をサポートしており、さまざまなアプリケーションに使用できる多用途のツールです。

Milvusアーキテクチャの概要



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レベルのコンポーネントとサービスについて説明します。 * 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、システムのフロント層およびユーザーへのエンドポイントとして機能します。 * コーディネーター サービス - タスクをワーカー ノードに割り当て、システムの頭脳として機能します。コーディネーターの種類には、ルート コーディネーター、データ コーディネーター、クエリ コーディネーターの 3 つがあります。 * ワーカー ノード: コーディネーター サービスからの指示に従い、ユーザーがトリガーした DML/DDL コマンドを実行します。クエリ ノード、データ ノード、インデックス ノードなど、3 種類のワーカー ノードがあります。 * ストレージ: データの永続性を維持します。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 データの両方に対して、Milvus にオブジェクト ストレージとファイル ベースのストレージを提供します。

技術要件

このセクションでは、 NetAppベクトル データベース ソリューションの要件の概要を説

明します。

技術要件

パフォーマンスを除き、このドキュメントで実行された検証の大部分では、以下に概説するハードウェアおよびソフトウェア構成が使用されました。これらの構成は、環境を設定する際に役立つガイドラインとして機能します。ただし、具体的なコンポーネントは個々の顧客の要件に応じて異なる場合があることにご注意ください。

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 * 48 x 3.49TB SSD-NVM * 2 つの柔軟なグループ 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、250 GB の永続ボリュームが 12 個あります。 * データはONTAP NAS S3 ボリュームです
富士通 PRIMERGY RX2540 M4 x 6	* 64個のCPU * Intel® Xeon® Gold 6142 CPU @ 2.60GHz * 256 GM物理メモリ * 1 x 100GbEネットワークポート
ネットワーク	100ギガビットイーサネット
StorageGRID	* 1 x SG100、3xSGF6024 * 3 x 24 x 7.68TB

ソフトウェア要件

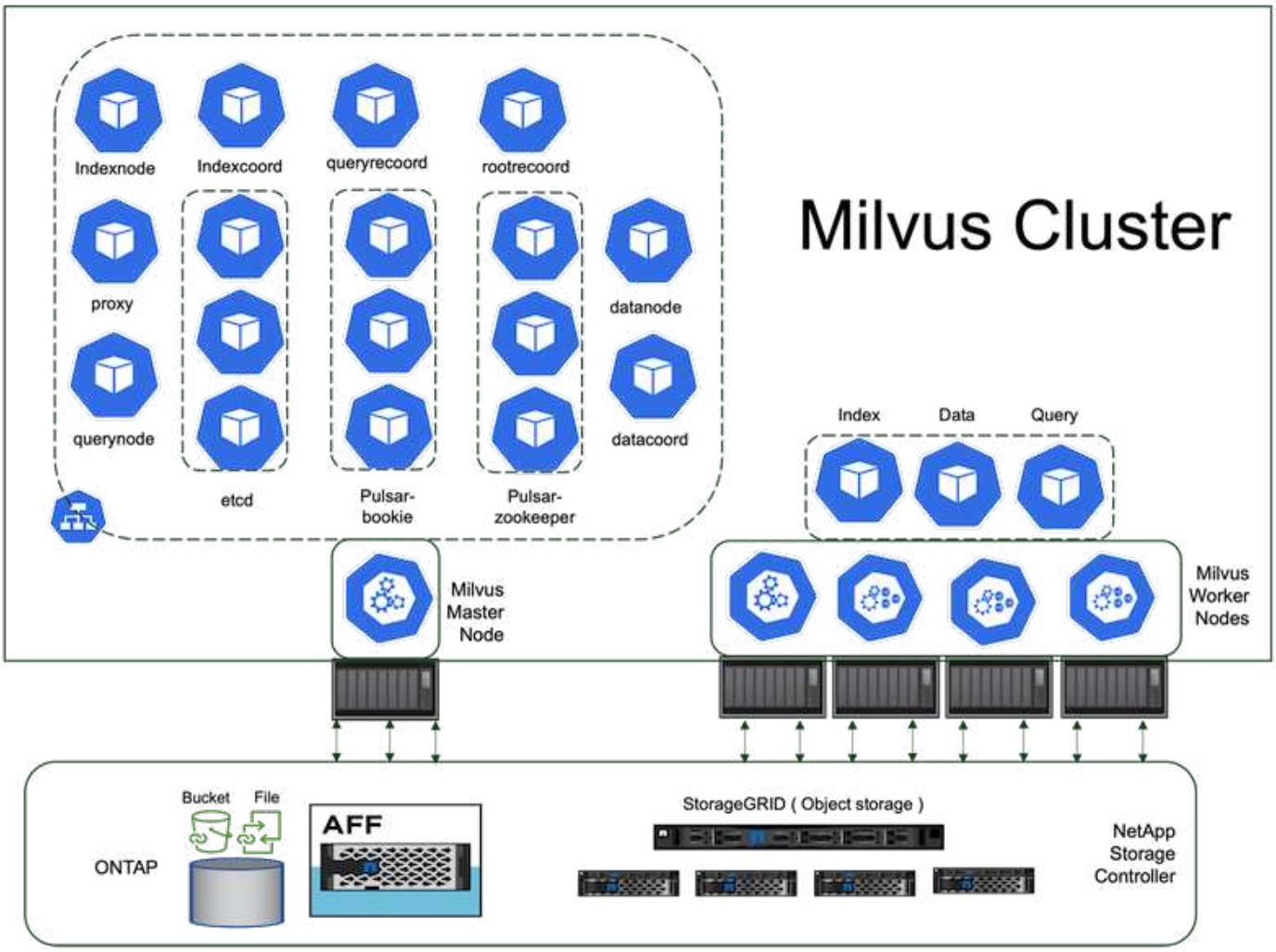
ソフトウェア	詳細
ミルバス星団	* チャート - milvus-4.1.11。 * APPバージョン – 2.3.4 * Bookkeeper、Zookeeper、Pulsar、etcd、Proxy 、QueryNode、Workerなどの依存バンドル
Kubernetes	* 5ノードのK8sクラスター * 1つのマスターノードと4つのワーカーノード * バージョン – 1.7.2
Python	*3.10.12.

展開手順

このセクションでは、 NetAppのベクトル データベース ソリューションの導入手順について説明します。

展開手順

このデプロイメント セクションでは、以下のようにラボのセットアップに Kubernetes を使用した milvus ベクター データベースを使用しました。



NetApp ストレージは、顧客データと Milvus クラスター データを保存するためのクラスター用のストレージを提供します。

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
- ストレージ仮想マシン（SVM）の作成
- 論理ネットワークインターフェースの割り当て
- NFS、S3 の設定とライセンス

NFS (ネットワーク ファイル システム) の場合は、次の手順に従ってください。

1. NFSv4 用のFlexGroupボリュームを作成します。この検証のセットアップでは、48 台の SSD を使用しました。1 台の SSD はコントローラのルート ボリューム専用で、47 台の SSD は NFSv4 用に分散されています。FlexGroup ボリュームの NFS エクスポート ポリシーに、Kubernetes (K8s) ノード ネットワークの読み取り/書き込み権限があることを確認FlexGroupます。これらの権限が設定されていない場合は、K8s ノード ネットワークに読み取り/書き込み (rw) 権限を付与します。
2. すべての K8s ノードでフォルダーを作成し、各 K8s ノードの論理インターフェイス (LIF) を介してこのフォルダーにFlexGroupボリュームをマウントします。

NAS S3 (Network Attached Storage Simple Storage Service) の場合は、以下の手順に従ってください。

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 が有効で管理ステータスが「up」に設定されたオブジェクト ストア サーバーをセットアップします。 HTTPS を有効にしてカスタム リスナー ポートを設定するオプションがあります。
3. 「vserver object-store-server user create -user <username>」コマンドを使用して、オブジェクト ストア サーバー ユーザーを作成します。
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ー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るようになります。
5. 手順 2 で作成したユーザーを使用して object-store-server グループを確立し、アクセスを許可します。この例では、「FullAccess」を指定しました。
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します。
3. 必要な権限を持つユーザーを作成します。

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

ソリューション検証

ソリューションの概要

当社では、5 つの主要領域に重点を置いた包括的なソリューション検証を実施しました。その詳細は以下に記載されています。各セクションでは、顧客が直面している課題、 NetAppが提供するソリューション、そしてそれによって顧客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 クラスターのセットアップ"顧客は、ストレージとコンピューティング、効果的なインフラストラクチャ管理、およびデータ管理に基づいて独立して拡張するという課題に直面しています。このセクションでは、クラスター データと顧客データの両方にNetAppストレージ コントローラーを利用して、Kubernetes に Milvus クラスターをインストールするプロセスについて詳しく説明します。
2. Milvus とAmazon FSx ONTAP 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、クラウドにベクトル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 FSx ONTAP for NetApp ONTAPにベクトルデータベース (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を使用したベクトル データベース保護。"このセクションでは、 SnapCenter がONTAPにあるベクター 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を使用した災害復旧"このセクションでは、ベクター データベースの災害復旧 (DR) の重要性と、NetApp の災害復旧製品 SnapMirror がベクター データベースに DR ソリューションを提供する方法について説明します。
5. "パフォーマンス検証"このセクションでは、Milvus や pgvecto.rs などのベクトル データベースのパフォーマンス検証を詳しく検討し、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 I/O プロファイルやネットアップ 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ます。これらのデータベースをONTAPストレージ ソリューション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。私たちの分析は、1 秒あたりに処理されるクエリ数 (QPS) などの主要な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** クラスターのセットアップ

このセクションでは、 NetAppのベクトル データベース ソリューション用の milvus クラスターのセットアップについて説明します。

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

ストレージとコンピューティングを独立して拡張し、効果的なインフラストラクチャ管理とデータ管理を行うという顧客の課題に対し、Kubernetes とベクター データベースを組み合わせることで、大規模なデータ操作を管理するための強力でスケーラブルなソリューションが実現します。 Kubernetes はリソースを最適化し、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と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に対する複雑なクエリを迅速に処理し、増大するデータ量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、クラスター データと顧客データの両方にNetAppストレージ コントローラーを利用して、Kubernetes に Milvus クラスターをインストールするプロセスについて詳しく説明します。
2. Milvus クラスターをインストールするには、さまざまな Milvus クラスター コンポーネントからのデータを保存するための永続ボリューム (PV) が必要です。これらのコンポーネントには、etcd (インスタンス 3 つ)、pulsar-bookie-journal (インスタンス 3 つ)、pulsar-bookie-ledgers (インスタンス 3 つ)、pulsar-zookeeper-data (インスタンス 3 つ) が含まれます。

> 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 ストリームの公開/サブスクリプションをサポートする基盤エンジンとして、Pulsar または Kafka のいずれかを使用できます。 NFS を使用した Kafka については、 NetApp はONTAP 9.12.1 以降で改善を行っており、これらの機能強化と、RHEL 8.7 または 9.1 以降に含まれる NFSv4.1 および Linux の変更により、NFS 経由で Kafka を実行するときに発生する可能性がある「silly rename」問題が解決されています。NetApp NFS ソリューションを使用した Kafka の実行に関する詳細情報については、以下を参照してください。"このリンク" 。

3. NetApp ONTAPから単一の NFS ボリュームを作成し、それぞれ 250 GB のストレージを持つ 12 個の永続ボリュームを確立しました。ストレージ サイズはクラスターのサイズに応じて異なります。たとえば、各 PV が 50 GB である別のクラスターがあります。詳細については、以下の PV YAML ファイルの 1 つを参照してください。このようなファイルは合計 12 個あります。各ファイルでは、storageClassName は「default」に設定されており、ストレージとパスは各 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. Milvus は、顧客データを保存するために、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 を利用します。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。 Milvus クラスターをデプロイするには Helm を使用します。 Milvus のダウンロード場所から構成ファイル 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. ログ、etcd、zookeeper、bookkeeper などの各セクションで、「storageClass」が「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 ストレージ認証情報を使用して外部 S3 に含めます。

```
###################################
# 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 クラスターを作成する前に、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 構成ファイルを使用して、Milvus クラスター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「実行中」であり、期待どおりに動作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 ストレージでのデータの書き込みと読み取りをテストします。
    - 「prepare_data_netapp_new.py」Python プログラム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:~#
```

- 「verify_data_netapp.py」Python ファイル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}}]}
```

上記の検証に基づき、 NetAppストレージ コントローラを使用して Kubernetes 上に Milvus クラスタを展開することで実証されているように、Kubernetes とベクトル データベースを統合すると、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かつ効率的なソリューションが顧客に提供されます。このセットアップにより、顧客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なソリューションになります。さまざまなクラスター コンポーネントに永続ボリューム (PV) を使用し、 NetApp ONTAPから単一の NFS ボリュームを作成することで、最適なリソース使用率とデータ管理が保証されます。 PersistentVolumeClaims (PVC) とポッド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、信頼性が高く一貫性のあるデータ操作が保証されます。顧客データにONTAPまたはStorageGRIDオブジェクト ストレージを使用すると、データのアクセス性とセキュリティがさらに強化されます。全体として、この設定により、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、回復力に優れた高性能なデータ管理ソリューションが顧客に提供されます。

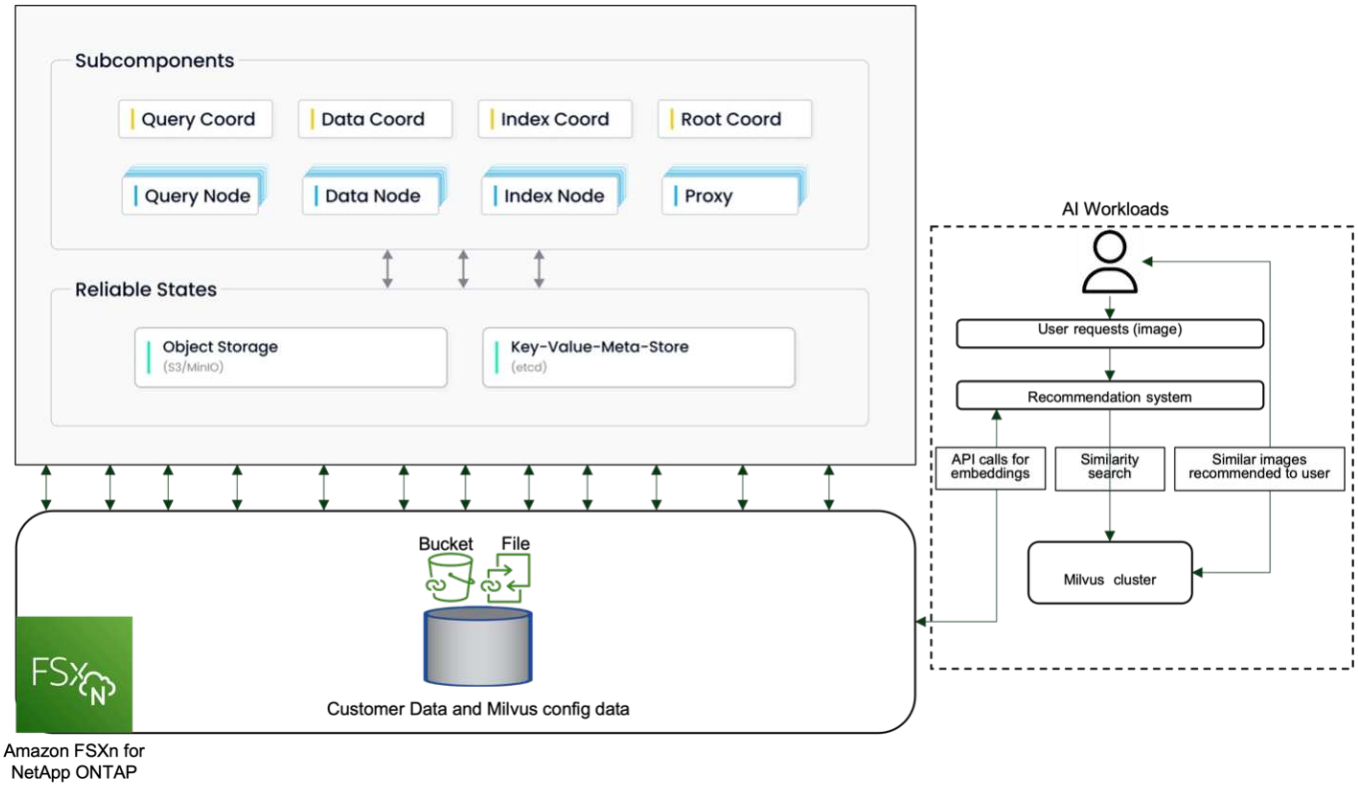
Milvus とAmazon FSx ONTAP for NetApp ONTAP - ファイルとオブジェクトの二重性

このセクションでは、 NetAppのベクトルデータベースソリューション用のAmazon FSx ONTAPを使用した milvus クラスターのセットアップについて説明します。

Milvus とAmazon FSx ONTAP 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 FSx ONTAP 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 データベースを展開すると、特に高次元データの処理や類似性検索の実行を必要とするアプリケーションにとって、いくつかの大きなメリットが得られます。まず、クラウドベースの展開ではスケーラビリティが提供され、増大するデータ量やクエリ負荷に合わせてリソースを簡単に調整できます。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、増加した需要を効率的に処理できるようになります。 2 番目に、クラウドの導入により、地理的に異なる場所にデータを複製できるため、高可用性と災害復旧が可能になり、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た場合でも継続的なサービスが保証されます。 3 番目に、使用したリソースに対してのみ料金を支払い、需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となり、コスト効率が向上します。最後に、クラウドにベクター データベースを展開すると、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とデータに基づく意思決定が容易になります。この検証で使用したAmazon FSx ONTAP for NetApp ONTAPを搭載した 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 FSx ONTAP 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、サブネットの詳細を書き留めます。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、サブネットがAmazon FSx ONTAP for NetApp ONTAPインスタンスのものと一致していることを確認します。
3. 'apt-get install nfs-common' コマンドを使用して 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、そこにAmazon FSx ONTAP 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. Milvus Web サイトからダウンロードできる docker-compose.yaml ファイルに基づいて、Milvus クラスターをセットアップします。

```
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、etcd 、minio、およびスタンドアロンの対応するMilvusコンテナパスにマッピングします。"付録D: docker-compose.yml" ymlの変更の詳細については
8. マウントされたフォルダー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 ファイルを含むディレクトリから '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. Amazon FSx ONTAP for NetApp ONTAPのベクトルデータベースとそのデータの読み取りおよび書き込み機能を検証するために、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. ベクターデータベース内のAmazon FSx ONTAP for NetApp ONTAPからのデータ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
/448789845791611912/448789845791611913/448789845791611939/100/4487898457
```

```
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. 顧客が AI ワークロード用の S3 プロトコルを介してベクトル データベースでテストされた NFS データにアクセス (読み取り) したい場合、簡単な Python プログラムを使用してこれを検証できます。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との類似性検索がその一例です。

```
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、 NetApp ONTAPデータ ストレージに Amazon のNetApp FSx ONTAPを使用し

て、Docker コンテナ内でスタンドアロンの Milvus セットアップを展開および操作する方法を効果的に示します。このセットアップにより、顧客は、スケーラブルで効率的な Docker コンテナ環境内で、ベクトル データベースのパワー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るようになります。 Amazon FSx ONTAP for NetApp ONTAPインスタンスと対応する EC2 インスタンスを作成することで、お客様は最適なリソース使用率とデータ管理を実現できます。ベクトル データベースでの FSx ONTAPからのデータ書き込みおよび読み取り操作の検証が成功したことで、信頼性が高く一貫性のあるデータ操作が保証されます。さらに、S3 プロトコルを介して AI ワークロードからデータを一覧表示する (読み取る) 機能により、データ アクセス性が向上します。したがって、この包括的なプロセスにより、Amazon の FSx ONTAP 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ソリューションが顧客に提供されます。

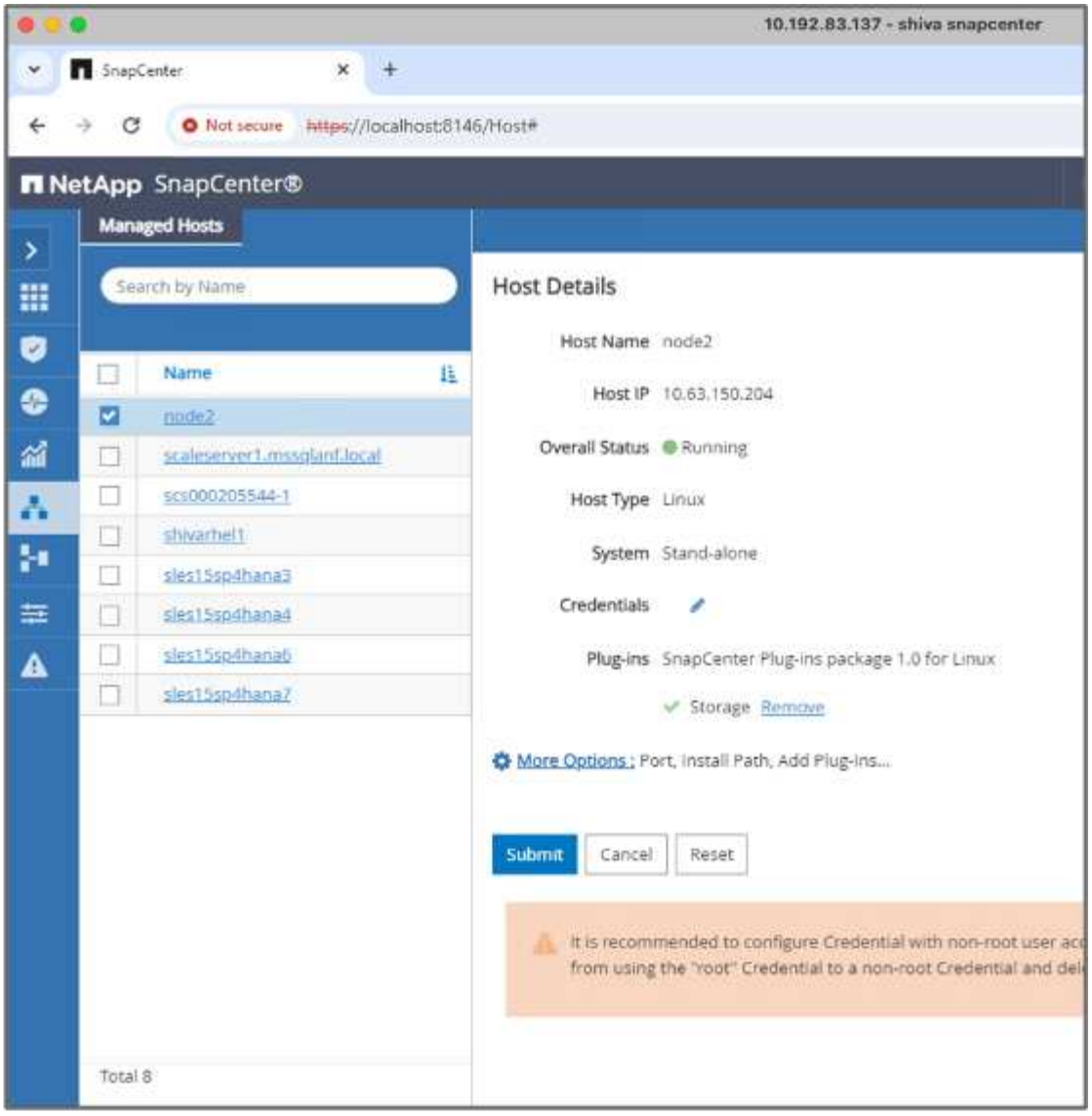
SnapCenterを使用したベクトル データベース保護

このセクションでは、 NetApp SnapCenterを使用してベクター データベースのデータ保護を提供する方法について説明します。

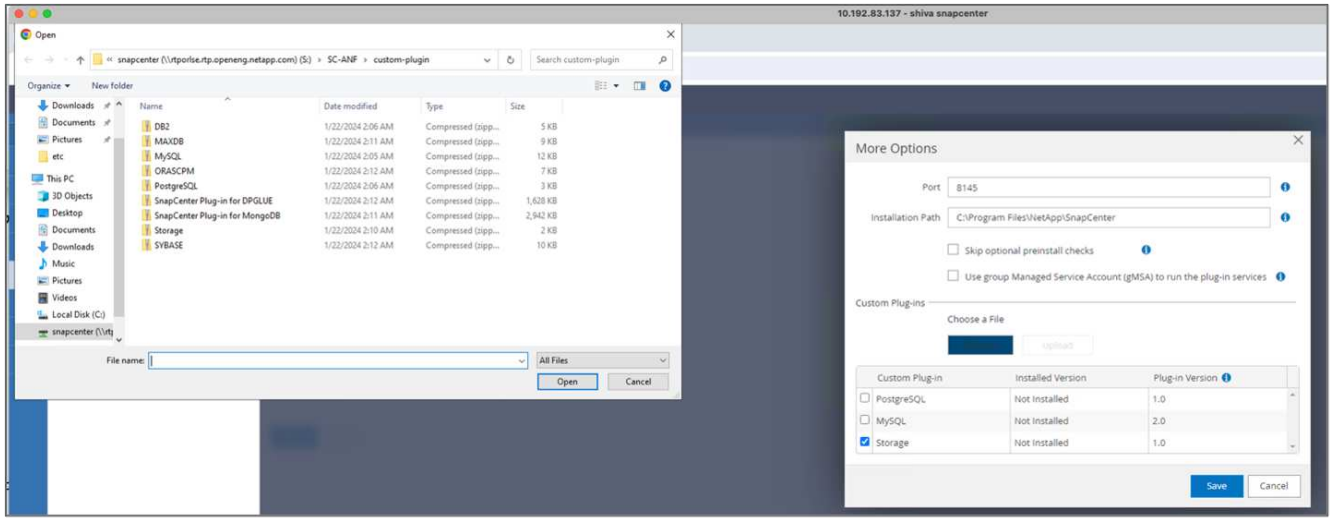
NetApp SnapCenterを使用したベクトル データベース保護。

たとえば、映画制作業界では、顧客がビデオ 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によりこのデータが失われると、業務に大きな影響が及び、数百万ドル規模の事業が危険にさらされる可能性があります。貴重なコンテンツが失われ、大きな混乱と経済的損失が発生する事例がありました。したがって、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は、この業界にとって最も重要です。このセクションでは、 SnapCenter がONTAPにあるベクター 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）と、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。"ここをクリックしてください。" SnapCenterバックアップワークフロー

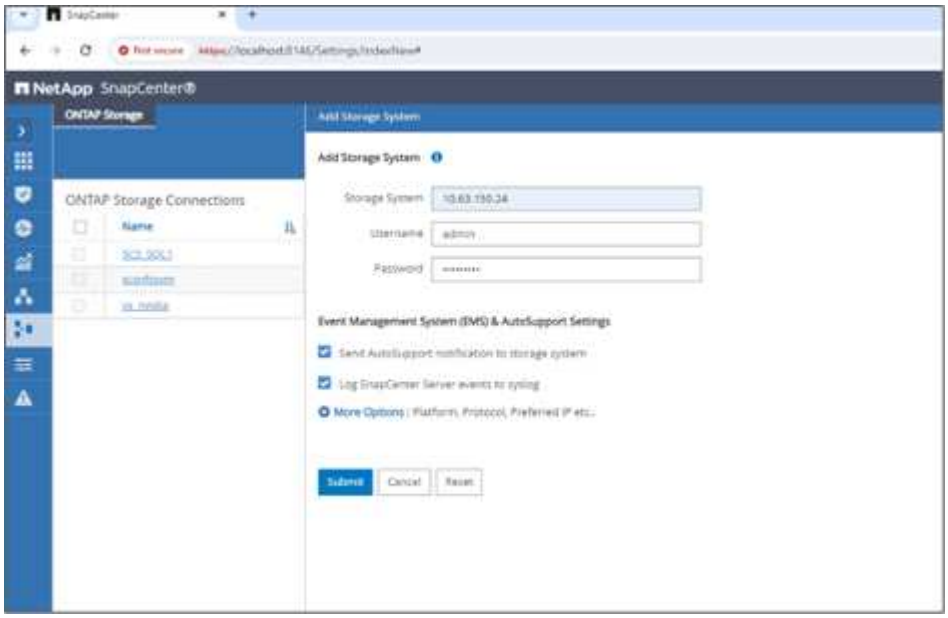
1. SnapCenterコマンドを実行するために使用するホストを設定します。



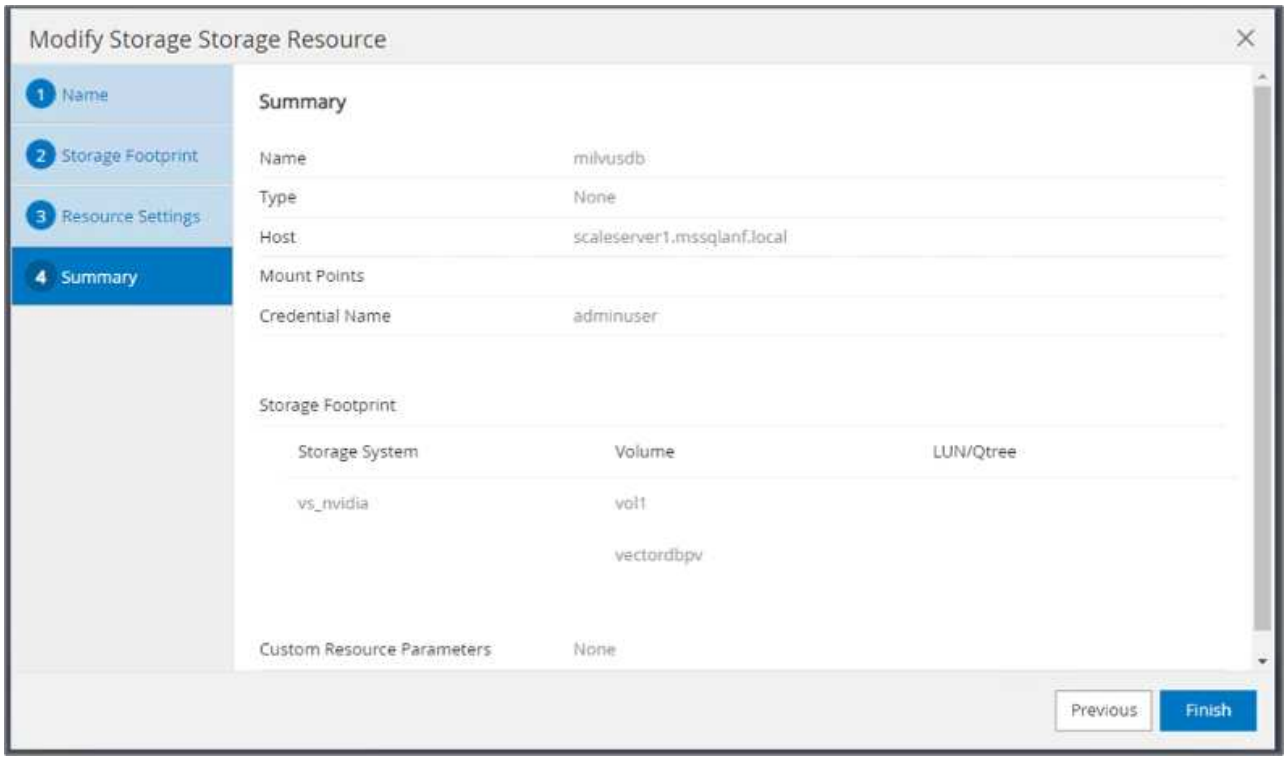
2. ストレージ プラグインをインストールして構成します。追加されたホストから、「その他のオプション」を選択します。ダウンロードしたストレージプラグインを選択して、"NetAppオートメーションストア" 。プラグインをインストールし、設定を保存します。



3. ストレージ システムとボリュームを設定します。「ストレージ システム」の下にストレージ システムを追加し、SVM (ストレージ仮想マシン) を選択します。この例では、「vs_nvidia」を選択しました。



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だベクター データベースのリソースを確立します。
   - デフォルト値で整合性グループ バックアップを有効にし、ファイルシステムの整合性なしでSnapCenterを有効にします。
   - [ストレージ フットプリント] セクションで、ベクター データベースの顧客データと Milvus クラスター 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、これらは「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、その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。



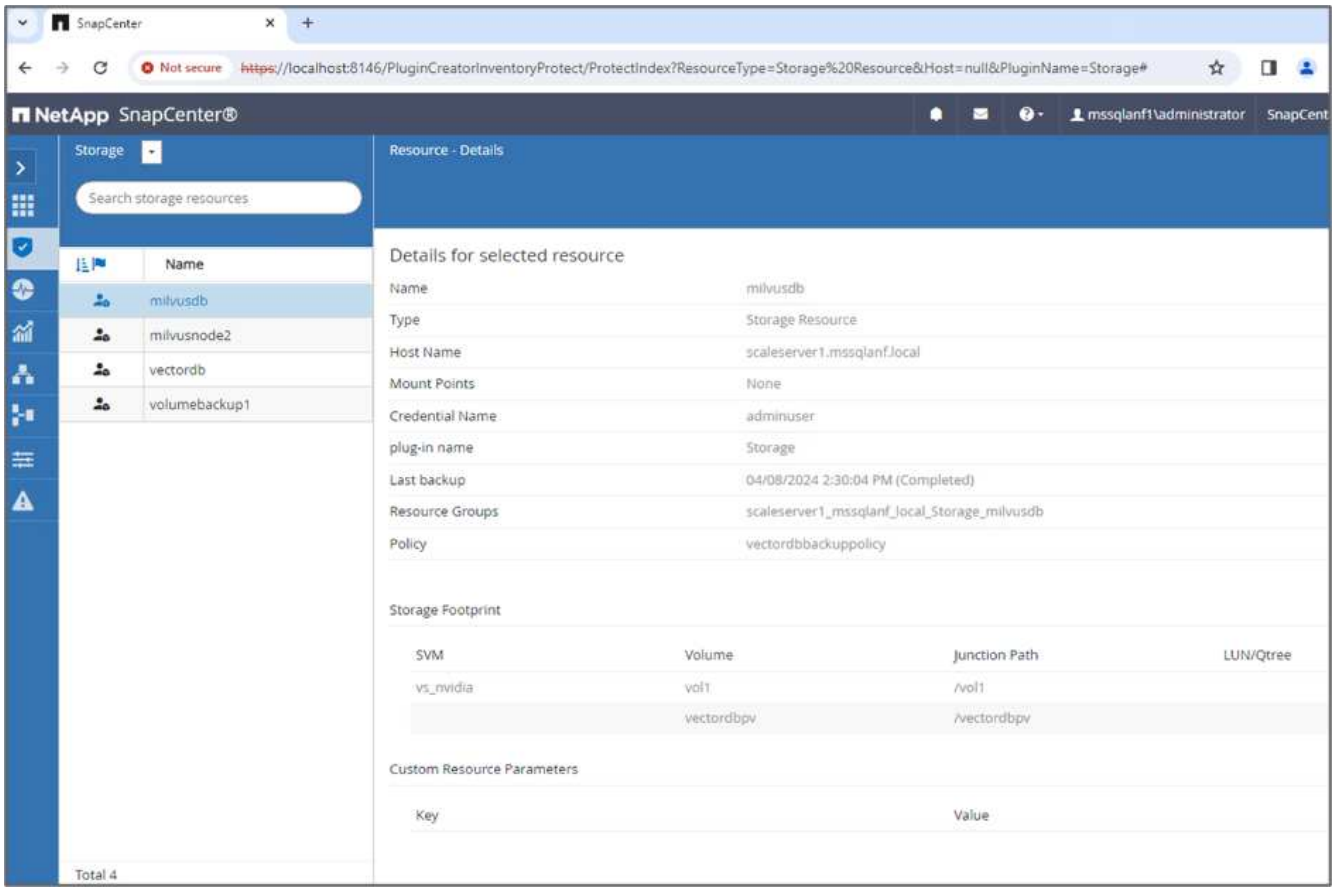
5. Python スクリプトを使用して S3 NAS バケットにデータを挿入します。私たちの場合、Milvus が提供するバックアップ スクリプト「prepare_data_netapp.py」を変更し、「sync」コマンドを実行してオペレーティング システムからデータをフラッシュしました。

```
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検証します。この例では、タイムスタンプが「2024-04-08 21:22」のファイルは、「prepare_data_netapp.py」スクリプトによって作成されました。

```
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'リソースからコンシステンシグループ（CG）スナップショットを使用してバックアップを開始します。

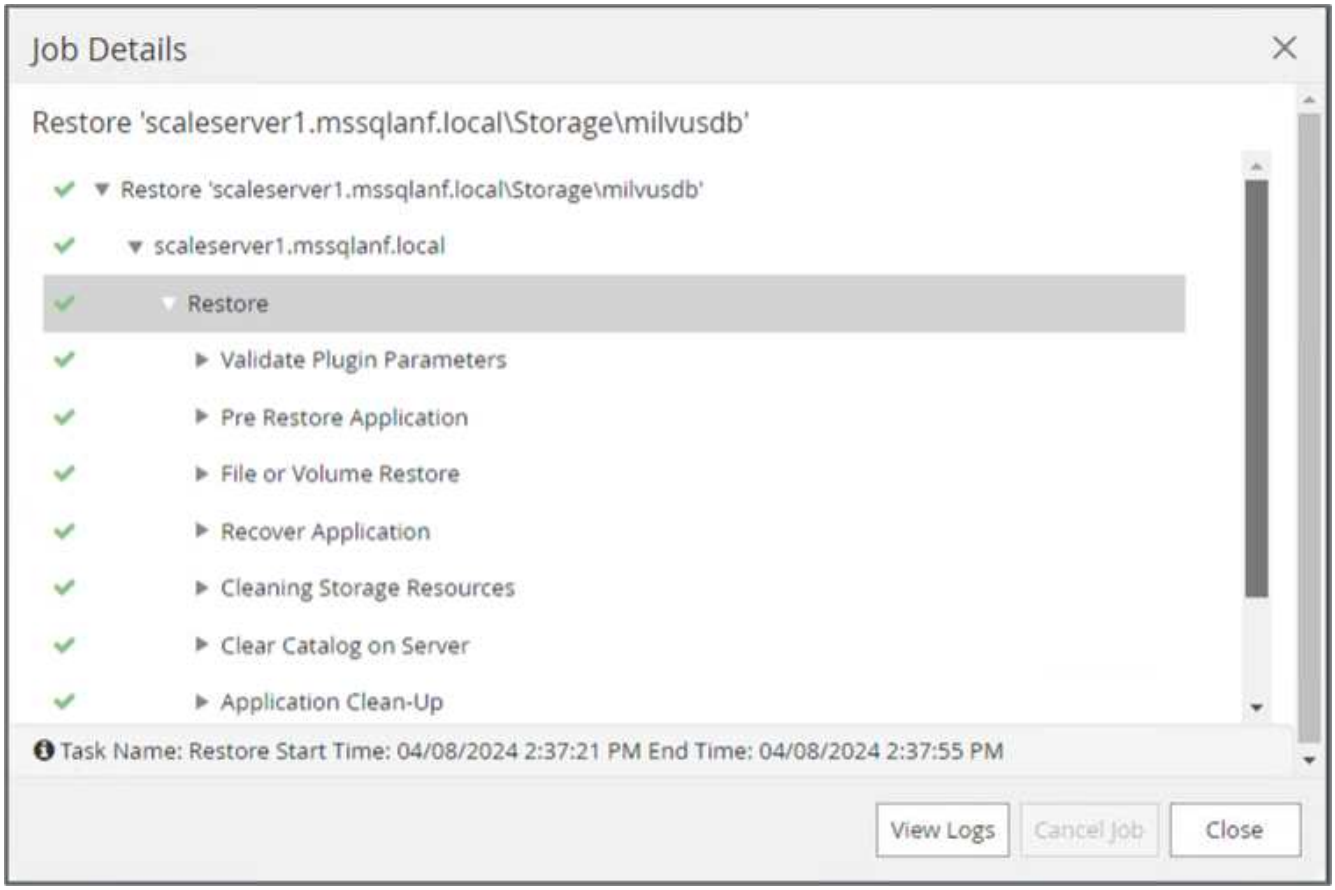


8. バックアップ機能をテストするために、バックアップ プロセス後に新しいテーブルを追加するか、NFS (S3 NAS バケット) から一部のデータを削除しました。

   このテストでは、バックアップ後に誰かが新しい、不要な、または不適切なコレクションを作成したシナリオを想定します。このような場合、ベクター データベースを、新しいコレクションが追加される前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. 以前のスナップショットから S3 NAS バケットの完全復元を実行します。



10. Python スクリプトを使用して、「hello_milvus_netapp_sc_test」および「hello_milvus_netapp_sc_test2」コレクションから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コレクション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、NetApp のSnapCenterを使用してONTAPに存在するベクトル データベース データと Milvus データを保護すると、特に映画制作など、データの整合性が最も重要となる業界の顧客に大きなメリットがもたらされます。 SnapCenter は一貫性のあるバックアップを作成し、完全なデータ復元を実行する機能を備えているため、埋め込まれたビデオ ファイルやオーディオ ファイルなどの重要なデータがハード ドライブの障害やその他の問題による損失から保護されます。これにより、業務の中断が防止されるだけでなく、多大な経済的損失も防ぐことができます。

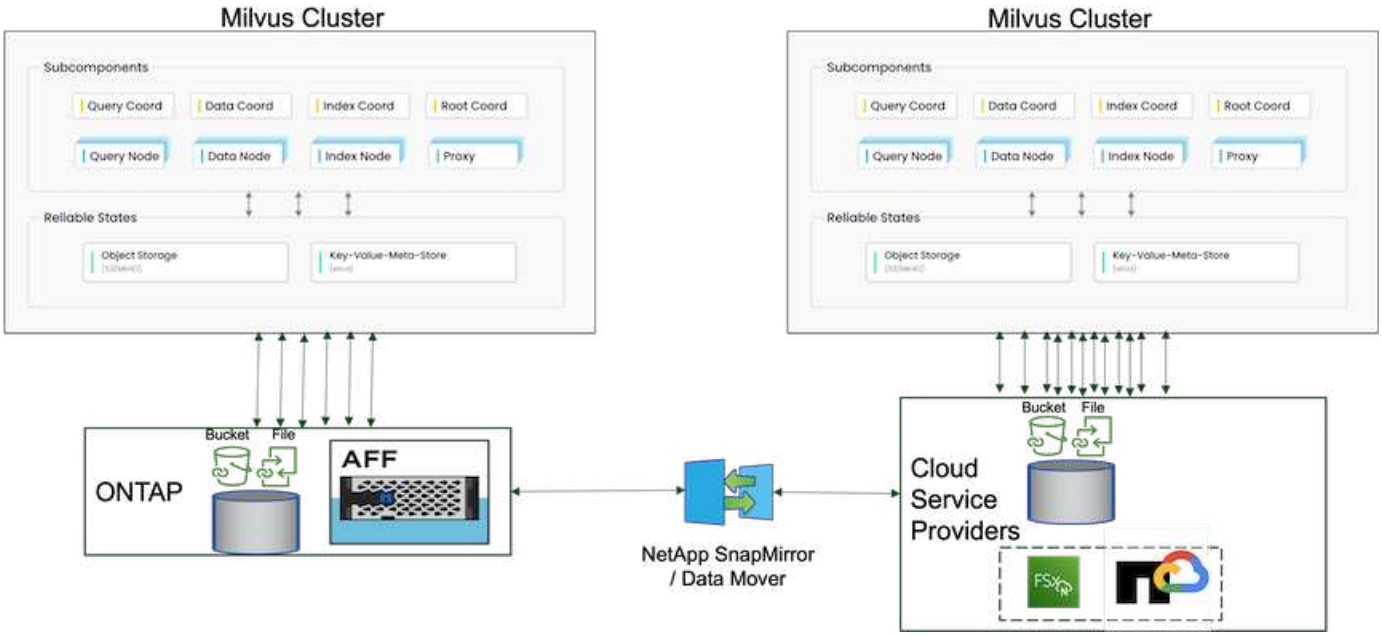
このセクションでは、ホストのセットアップ、ストレージ プラグインのインストールと構成、カスタム スナップショット名を持つベクター データベースのリソースの作成など、 ONTAPに存在するデータを保護するためにSnapCenterを構成する方法について説明しました。また、Consistency Group スナップショットを使用してバックアップを実行し、S3 NAS バケット内のデータを検証する方法も紹介しました。

さらに、バックアップ後に不要または不適切なコレクション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 は以前のスナップショットから完全な復元を実行できるため、ベクター データベースを新しいコレクションを追加する前の状態に戻すことができ、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復元するこの機能は顧客にとって非常に貴重であり、データが安全であるだけでなく、正しく維持されているという保証も提供します。したがって、NetApp のSnapCenter製品は、データ保護と管理のための堅牢で信頼性の高いソリューションを顧客に提供します。

NetApp SnapMirrorを使用した災害復旧

このセクションでは、 NetAppのベクトル データベース ソリューションにおけるSnapMirrorを使用した DR (災害復旧) について説明します。

NetApp SnapMirrorを使用した災害復旧



災害復旧は、特に高次元データの管理と複雑な類似性検索の実行という役割を考えると、ベクター 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。適切に計画され実装された災害復旧戦略により、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、データが失われたり侵害されたりすることがなくなります。これは、データの損失や破損が重大な運用中断や経済的損失につながる可能性があるベクトル データベースに依存するアプリケーションにとって特に重要です。さらに、堅牢な災害復旧計画により、ダウンタイムを最小限に抑え、サービスの迅速な復旧が可能になり、ビジネスの継続性も確保されます。これは、さまざまな地理的な場所、定期的なバックアップ、およびフェイルオーバーメカニズムにわたるNetAppデータ レプリケーション製品 SnapMirror を通じて実現されます。したがって、災害復旧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。

NetApp のSnapMirror は、あるNetApp ONTAPストレージ コントローラから別の NetApp ONTAP ストレージ コントローラへのデータ レプリケーションを提供し、主に災害復旧 (DR) およびハイブリッド 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、このツールはオンプレミス環境とクラウド環境間でのデータのスムーズな移行を容易にします。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せずに実現されるため、複数のプラットフォームにわたるデータ管理の効率と柔軟性が向上します。

ベクトル データベース シナリオにおけるNetAppハイブリッド ソリューションは、さらに多くの利点をもたらします。

1. スケーラビリティ: NetApp のハイブリッド クラウド ソリューションは、要件に応じてリソースを拡張する機能を提供します。定期的な予測可能なワークロードにはオンプレミスのリソースを活用し、ピーク時や予期しない負荷にはAmazon FSx ONTAP for NetApp ONTAPや Google Cloud NetApp Volumes (NetApp Volumes) などのクラウド リソースを活用できます。
2. コスト効率: NetApp 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は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 リソースの料金を支払うことで、コストを最適化できます。この従量課金モデルは、 NetApp instaclustr サービス オファリングを使用すると、非常にコスト効率が高くなります。オンプレミスおよび大手クラウド サービス プロバイダー向けに、instaclustr はサポートとコンサルティングを提供します。
3. 柔軟性: NetApp のハイブリッド クラウドでは、データを処理する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を備えたオンプレミスで複雑なベクトル演算を実行し、クラウドではそれほど負荷のかからない演算を実行することを選択できます。
4. ビジネス継続性: 災害が発生した場合でも、データをNetAppハイブリッド クラウドに保存することで、ビ

ジネスの継続性を確保できます。オンプレミスのリソースが影響を受ける場合は、すぐにクラウドに切り替えることができます。 NetApp SnapMirrorを活用することで、オンプレミスからクラウドへ、あるいはその逆にデータを移動できます。

5. イノベーション: NetApp 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ーへのアクセスを提供することで、イノベーションの迅速化も実現します。 Amazon FSx ONTAP for NetApp ONTAP、 Azure NetApp Files 、 Google Cloud NetApp Volumesなどのクラウドにおけるの NetApp のイノベーションは、クラウド サービス プロバイダーの革新的な製品であり、推奨される NAS です。

ベクターデータベースのパフォーマンス検証

このセクションでは、ベクター データベースで実行されたパフォーマンス検証について説明します。

パフォーマンス検証

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、最適な操作と効率的なリソース利用を保証するための重要な要素となります。高次元データの処理や類似性検索の実行で知られるベクター データベースでは、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、サービスの低下なしにシステムが予想される負荷を処理できることを保証するのに役立ちます。同様に、ストレージ システムでは、全体的なシステム パフォーマンスに影響を与える可能性のある遅延の問題やボトルネックが発生することなく、データが効率的に保存および取得されるようにするために、パフォーマンス検証が不可欠です。また、ストレージ インフラストラクチャの必要なアップグレードや変更について、情報に基づいた意思決定を行うのにも役立ちます。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。

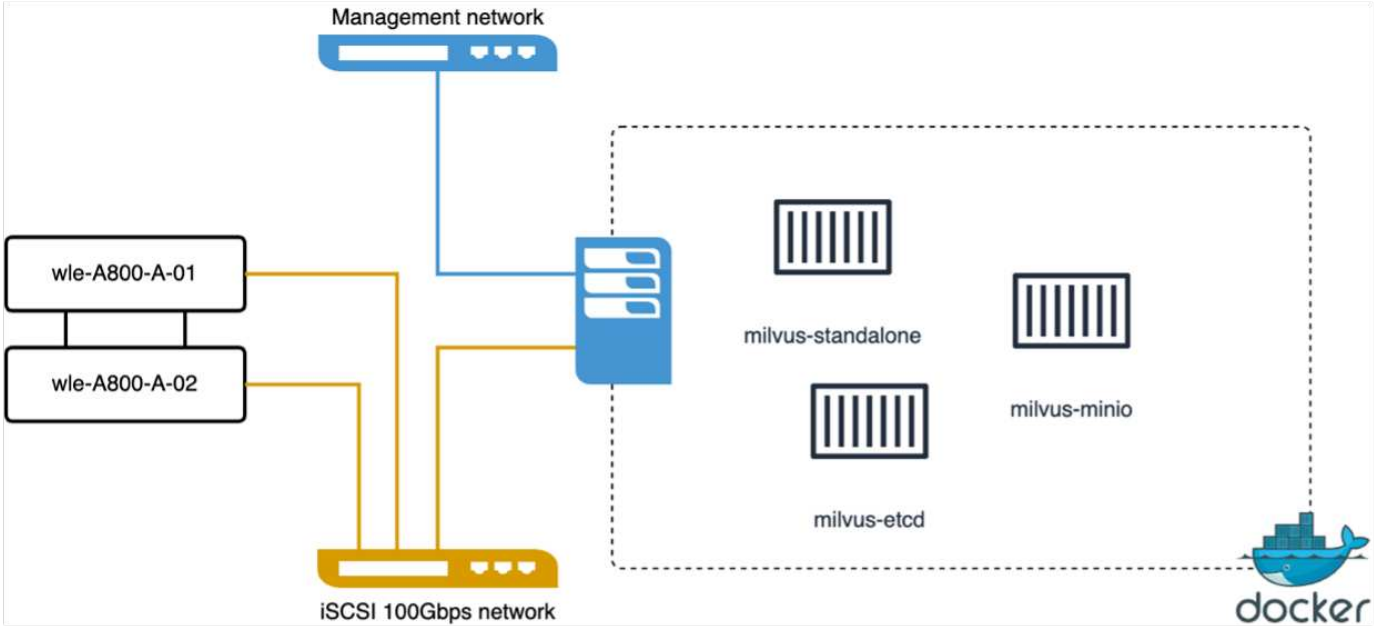
このセクションでは、Milvus や pgvecto.rs などのベクトル データベースのパフォーマンス検証を詳しく検討し、LLM ライフサイクル内での RAG および推論ワークロードをサポートする I/O プロファイルやネットアップ ストレージ コントローラの動作などのストレージ パフォーマンス特性に焦点を当てます。これらのデータベースをONTAPストレージ ソリューションと組み合わせた場合のパフォーマンスの差別化要因を評価し、特定します。私たちの分析は、1 秒あたりに処理されるクエリ数 (QPS) などの主要なパフォーマンス指標に基づいて行われます。

milvus と進捗状況に使用された方法論については以下を確認してください。

詳細	Milvus (スタンドアロンおよびクラスター)	Postgres(pgvecto.rs) #
version	2.3.2	0.2.0
Filesystem	iSCSI LUN 上の XFS	
ワークロードジェネレーター	"VectorDBベンチ"– v0.0.5	
データセット	LAIONデータセット * 1000万個の埋め込み * 768次元 * 約300GBのデータセットサイズ	
ストレージ コントローラ	AFF 800 * バージョン – 9.14.1 * 4 x 100GbE – milvus用、2x 100GbE （postgres用） * iscsi	

Milvus スタンドアロン クラスターを使用した **VectorDB-Bench**

VectorDB-Bench を使用して、milvus スタンドアロン クラスターで次のパフォーマンス検証を実行しました。milvus スタンドアロン クラスターのネットワークとサーバー接続は次のとおりです。



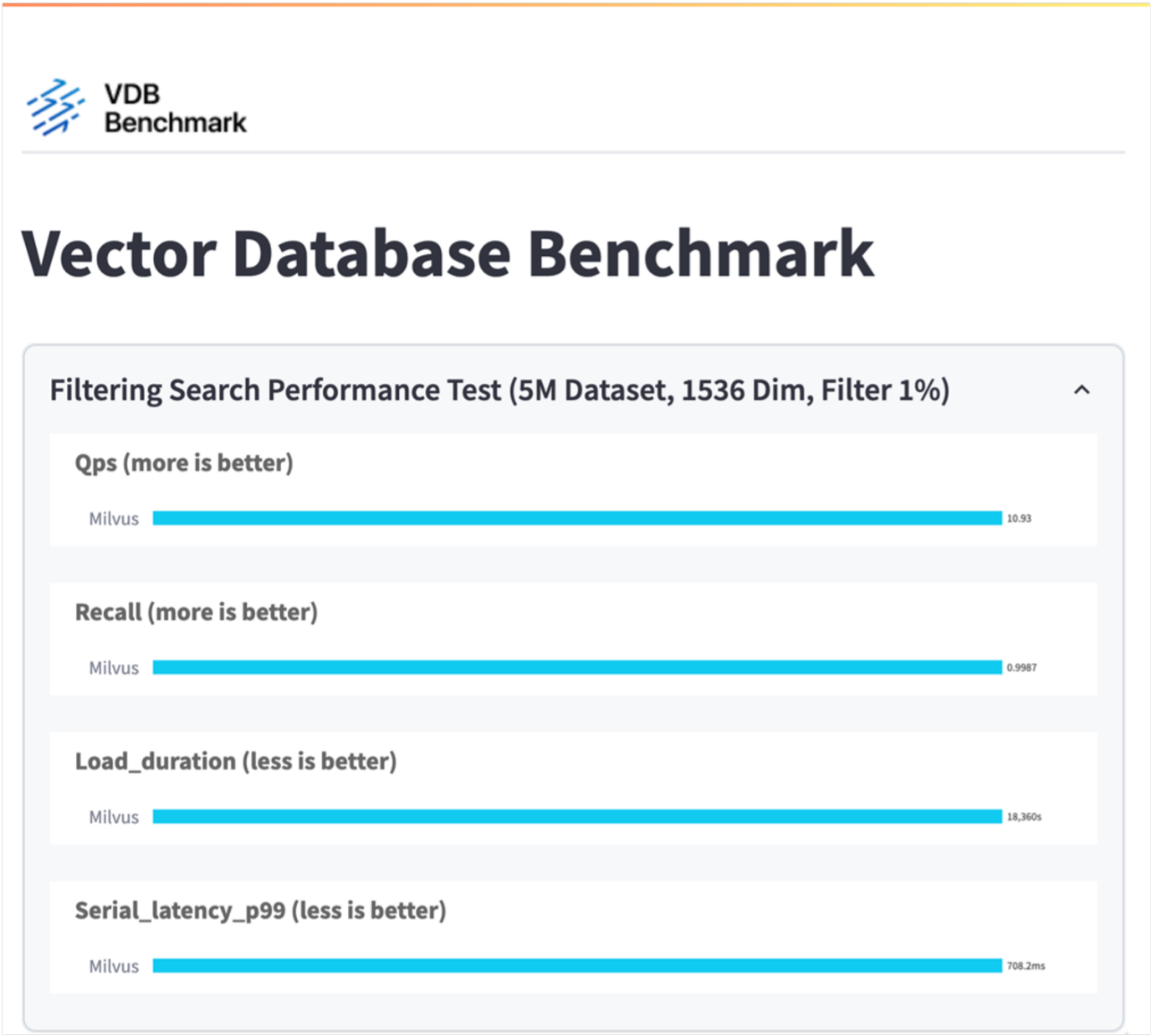
このセクションでは、Milvus スタンドアロン データベースのテストから得た観察結果を共有します。 。これらのテストのインデックス タイプとして DiskANN を選択しました。 。約 100 GB のデータセットの取り込み、最適化、インデックス作成には約 5 時間かかりました。この期間の大半では、20 個のコア (ハイパースレッディングを有効にすると 40 個の vCPU に相当) を搭載した Milvus サーバーが最大 CPU 容量の 100% で動作していました。DiskANN は、システム メモリ サイズを超える大規模なデータセットで特に重要であることがわかりました。 。クエリフェーズでは、1 秒あたりのクエリ数 (QPS) レートが 10.93、リコールが 0.9987 でした。クエリの 99 パーセンタイル レイテンシは 708.2 ミリ秒と測定されました。

ストレージの観点から見ると、データベースは、取り込み、挿入後の最適化、およびインデックス作成の各フェーズで約 1,000 オペレーション/秒を発行しました。クエリフェーズでは、32,000 オペレーション/秒が要求されました。

次のセクションでは、ストレージ パフォーマンス メトリックを示します。

ワークロードフェーズ	指標	**Value**
データの取り込みと挿入後の最適化	IOPS	1,000未満
	レイテンシー	400 マイクロ秒未満
	ワークロード	読み取り/書き込みの混合、主に書き込み
	IOサイズ	64 KB
クエリ	IOPS	ピーク時32,000
	レイテンシー	400 マイクロ秒未満
	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の結果は以下の通りです。

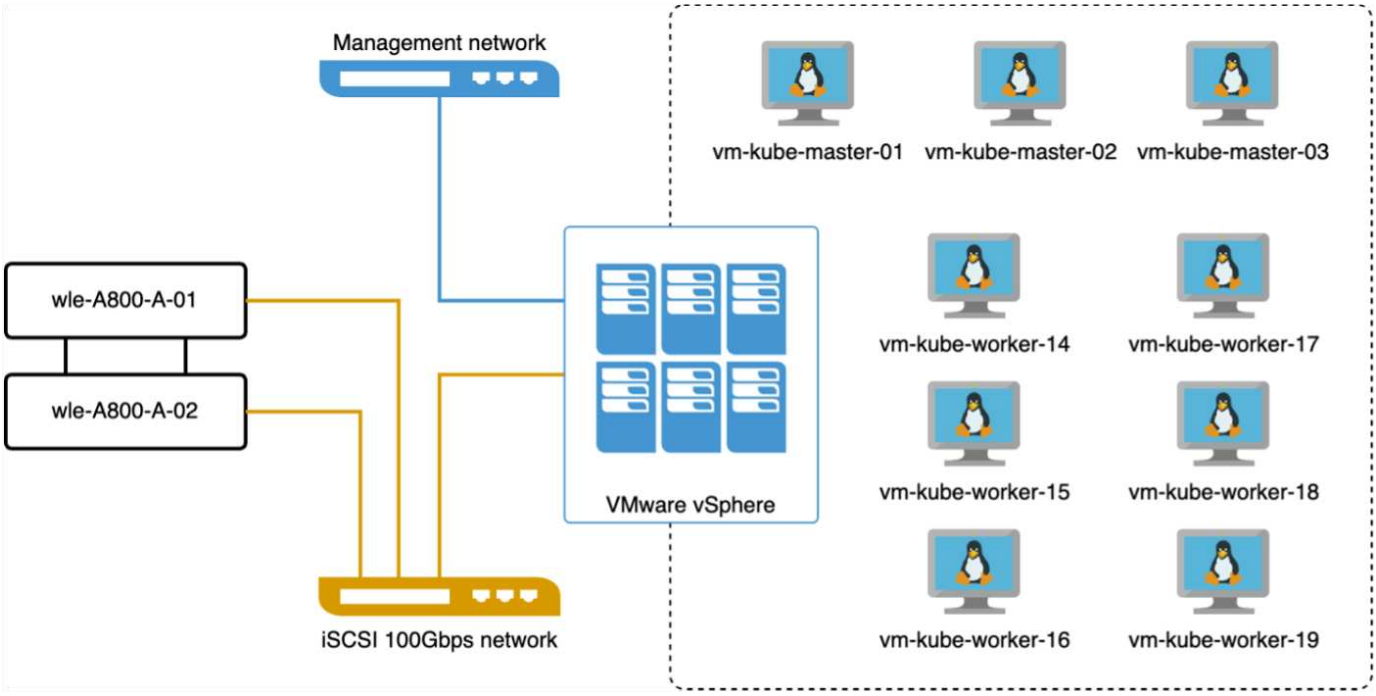


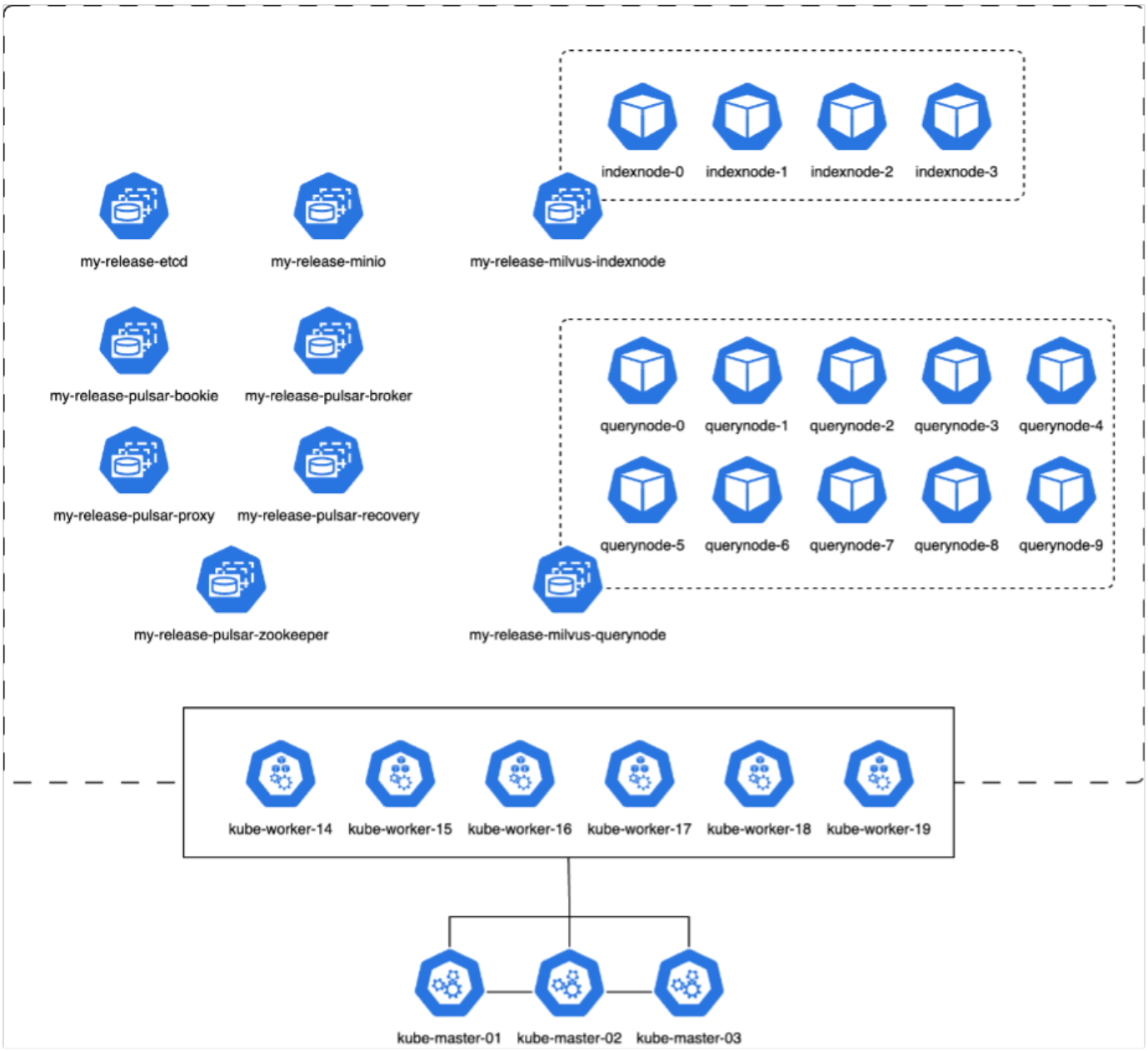
スタンドアロン Milvus インスタンスのパフォーマンス検証から、現在の設定では次元数が 1536 の 500 万ベクトルのデータセットをサポートするには不十分であることが明らかです。ストレージには十分なリソースがあり、システムのボトルネックにはならないと判断しました。

milvus クラスターを使用した VectorDB-Bench

このセクションでは、Kubernetes 環境内での Milvus クラスターの展開について説明します。この Kubernetes セットアップは、Kubernetes マスター ノードとワーカー ノードをホストする VMware vSphere デプロイメント上に構築されました。

VMware vSphere および Kubernetes のデプロイメントの詳細については、次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 データベースのテストから得られた観察結果を紹介します。 * 使用されたインデックス タイプは DiskANN です。 * 以下の表は、1536 次元で 500 万のベクトルを扱う場合のスタンドアロン展開とクラスター展開の比較を示しています。クラスター展開では、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ることがわかりました。スタンドアロン設定と比較して、クラスター展開ではクエリの 99 パーセンタイル レイテンシが 6 分の 1 に削減されました。 * クラスター展開では 1 秒あたりのクエリ数 (QPS) レートは高くなりましたが、望ましいレベルには達しませんでした。

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%

以下の画像は、ストレージ クラスターのレイテンシや合計 IOPS (1 秒あたりの入出力操作数) など、さまざまなストレージ メトリックを示しています。

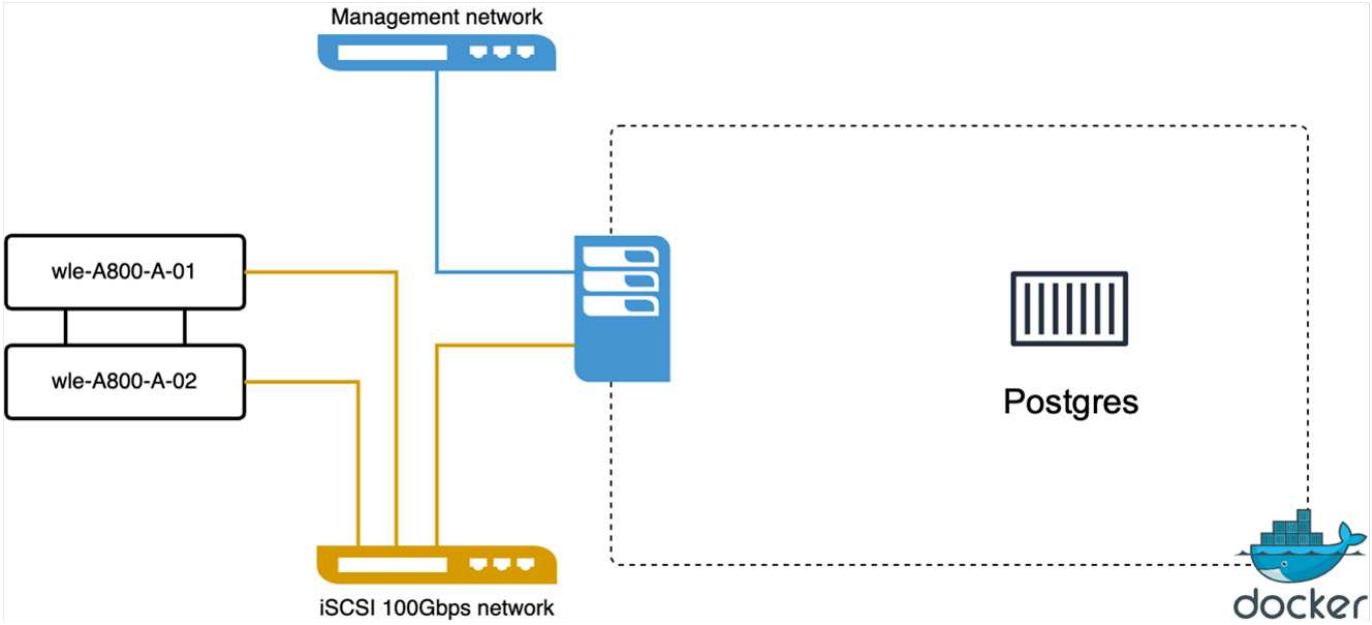
次のセクションでは、主要なストレージ パフォーマンス メトリックを示します。

ワークロードフェーズ	指標	**Value**
データの取り込みと挿入後の最適化	IOPS	1,000未満
	レイテンシー	400 マイクロ秒未満
	ワークロード	読み取り/書き込みの混合、主に書き込み
	IOサイズ	64 KB
クエリ	IOPS	ピーク時14万7000人
	レイテンシー	400 マイクロ秒未満
	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 Milvus と Milvus クラスター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示します。 * I/O プロファイルは、スタンドアロン展開とクラスター展開の両方で一貫していることがわかりました。 * ピーク IOPS で観測された差は、クラスター展開内のクライアント数が多いことに起因します。

Postgres を使用した **VectorDB** 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 (具体的には pgvecto.rs) のネットワークとサーバー接続に関する詳細は次のとおりです。

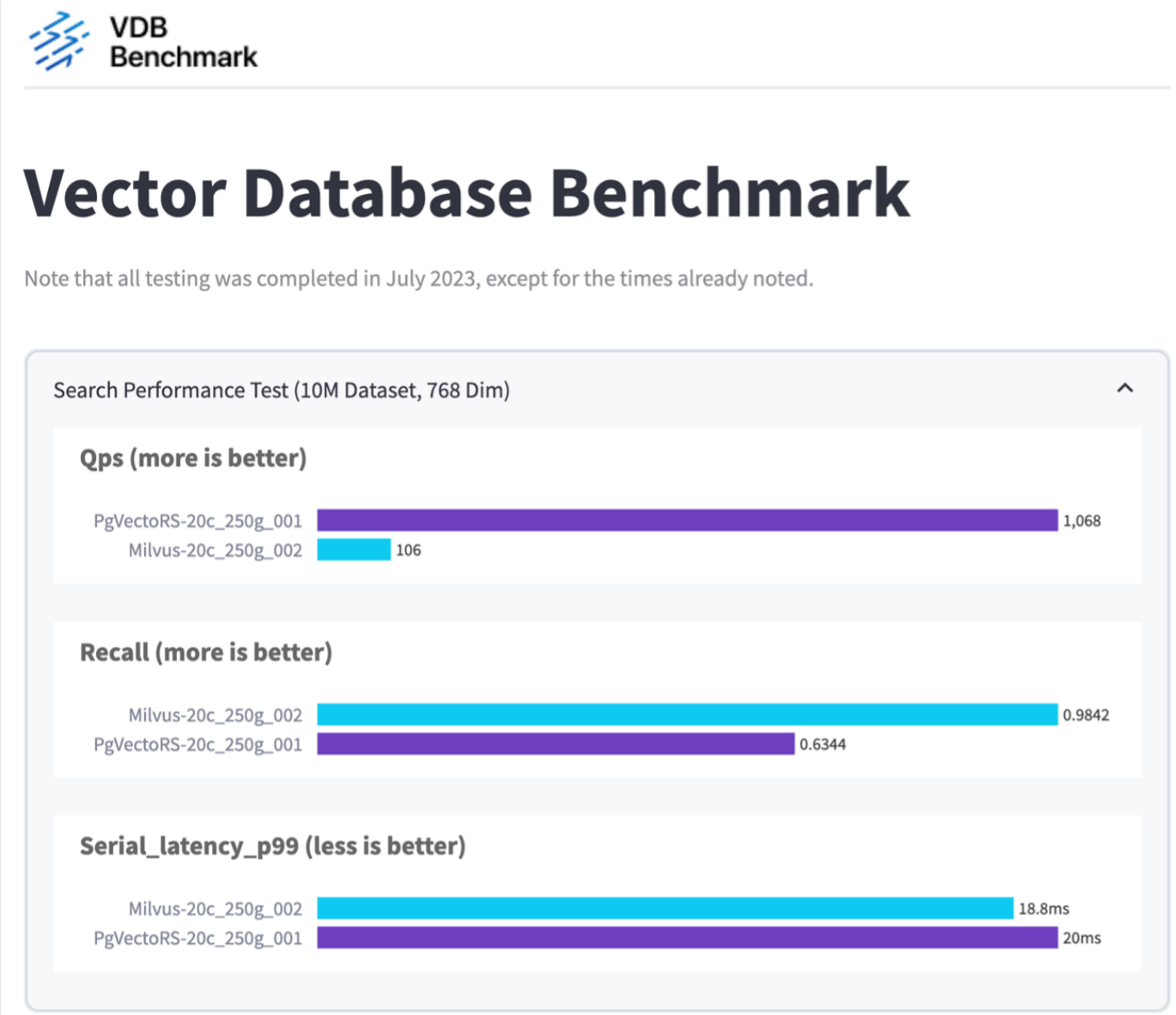


このセクションでは、特に pgvecto.rs を使用して PostgreSQL データベースをテストした結果と観察結果を共有します。 * テスト時点では DiskANN が pgvecto.rs で利用できなかったため、これらのテストのインデックス タイプとして HNSW を選択しました。 * データ取り込みフェーズでは、768 次元の 1,000 万ベクトルで構成される Cohere データセットをロードしました。このプロセスには約 4.5 時間かかりました。 * クエリフェーズでは、1 秒あたりのクエリ数 (QPS) が 1,068、リコールが 0.6344 でした。クエリの 99 パーセンタイル レイテンシは 20 ミリ秒と測定されました。実行時間のほとんどを通じて、クライアントの CPU は 100% の能力で動作していました。

以下の画像は、ストレージ クラスターのレイテンシ合計 IOPS (1 秒あたりの入出力操作数) など、さまざまなストレージ メトリックを示しています。


```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-
metrics.png["入出力ダイアログまたは書かれたコンテンツを示す図"]
```

ベクター**DB**ベンチにおける**milvus**と**postgres**のパフォーマンス比較



VectorDBBench を使用した Milvus と PostgreSQL のパフォーマンス検証に基づいて、次のことがわかりました。

- インデックスタイプ: HNSW
- データセット: 768次元の1000万ベクトルを含むCohere

pgvecto.rs は 1 秒あたりのクエリ数 (QPS) が 1,068、リコールが 0.6344 であったのに対し、Milvus は QPS が 106、リコールが 0.9842 であったことがわかりました。

クエリの高精度が優先される場合、Milvus はクエリごとに関連アイテムをより高い割合で取得するため、pgvecto.rs よりも優れています。しかし、1 秒あたりのクエリ数の方が重要な要素である場合、pgvecto.rs は Milvus を上回ります。ただし、pgvecto.rs 経由で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ことが重要です。

パフォーマンス検証に基づく観察:

パフォーマンス検証に基づいて、次のことが観察されました。

Milvus では、I/O プロファイルは、Oracle SLOB で見られるような OLTP ワークロードによく似ています。ベンチマークは、データ取り込み、事後最適化、クエリの 3 つのフェーズで構成されます。初期段階では主に 64 KB の書き込み操作が特徴で、クエリ段階では主に 8 KB の読み取りが行われます。 ONTAP がMilvus I/O 負荷を適切に処理することが期待されます。

PostgreSQL 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。現在進行中のメモリ内実装を考慮すると、クエリフェーズ中にディスク I/O は発生しませんでした。

DiskANN は、ストレージの差別化に重要なテクノロジーとして登場しました。システム メモリの境界を超えたベクトル 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 などのメモリ内ベクトル DB インデックスでは、ストレージ パフォーマンスの差別化を確立できる可能性は低くなります。

また、インデックス タイプが HSNW の場合、RAG アプリケーションをサポートするベクター データベースにとって最も重要な操作フェーズであるクエリ フェーズでは、ストレージが重要な役割を果たしないことにも注目に値します。ここでの意味は、ストレージ 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な影響を与えないということです。

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

このセクションでは、 NetAppのベクトル データベース ソリューションの pgvector 機能で instaclustr 製品が postgreSQL と統合される具体的な方法について説明します。

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

このセクションでは、instaclustr 製品が pgvector 機能で postgreSQL とどのように統合されるかを詳細に説明します。 「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」の例があります。ご確認ください"ブログ"さらに詳しい情報を入手します。

ベクターデータベースのユースケース

このセクションでは、 NetAppベクトル データベース ソリューションの使用例の概要を説明します。

ベクター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検索拡張生成とNetApp IT チャットボットなどの 2 つのユースケースについて説明します。

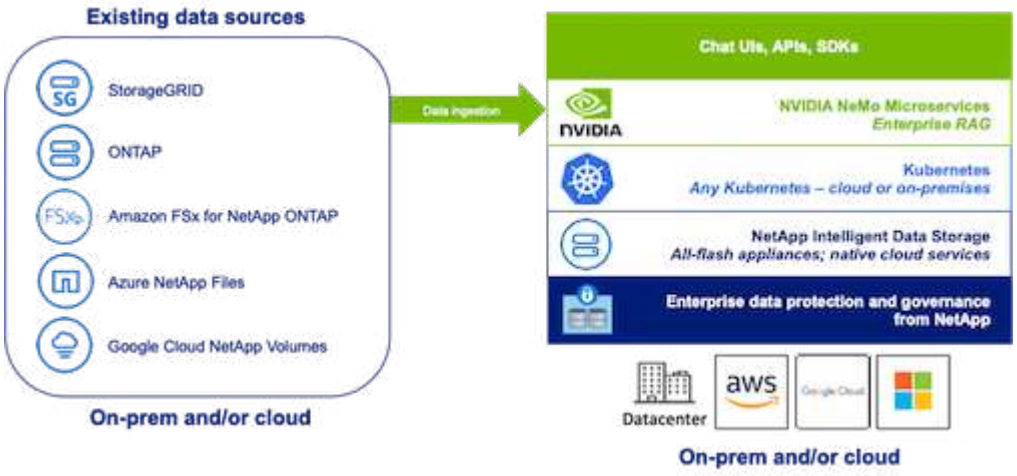
大規模言語モデル（LLM）を用いた検索拡張生成（RAG）

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

NVIDIA Enterprise RAG LLM Operator は、企業内で RAG を実装するための便利なツールです。このオペレーターは、完全な RAG パイプラインをデプロイするために使用できます。 RAG パイプラインは、知識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。

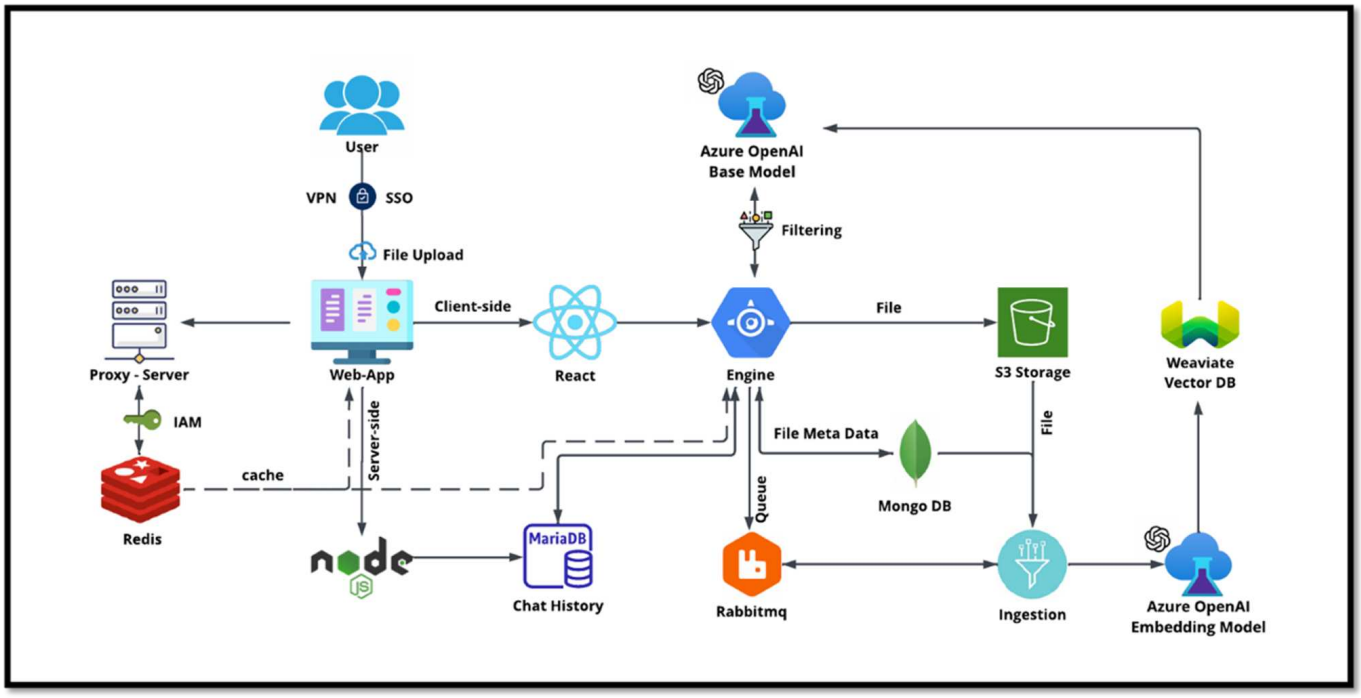
```
NetApp has validated an enterprise RAG architecture powered by the NVIDIA
Enterprise RAG LLM Operator alongside NetApp storage. Refer to our blog
post for more information and to see a demo. Figure 1 provides an overview
of this architecture.
```

図1) NVIDIA NeMo MicroservicesとNetAppを搭載したエンタープライズRAG



NetApp IT チャットボットのユースケース

NetApp のチャットボットは、ベクター データベースのもう 1 つのリアルタイム使用例として機能します。この例では、 NetApp Private OpenAI Sandbox は、NetApp の社内ユーザーからのクエリを管理するための効果的で安全かつ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 認証を通じて組織内の役割と責任に基づいて、ユーザーへの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中心のインテリジェントなシステムを作成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られます。

ユーザー認証と検証:

- ユーザー クエリは、まずNetAppシングル サインオン (SSO) プロセスを経て、ユーザーの ID を確認します。
- 認証が成功すると、システムは VPN 接続をチェックし、安全なデータ転送を確保します。

データの転送と処理:

- 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通じて MariaDB に送信されます。 MariaDB は、ユーザー データを管理および保存するため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。
- その後、MariaDB は情報をNetApp Azure インスタンスに送信し、NetApp Azure インスタンスはユーザー データを AI 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、処理の準備をします。
- クリーンアップされた入力は 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。

レスポンスの生成とモデレーション:

- まず、ベース モデルからの応答がチェックされ、正確であり、コンテンツ標準を満たしているかどうかが確認されます。
- チェックに合格すると、応答がユーザーに返されます。このプロセスにより、ユーザーは質問に対して明確かつ正確で適切な回答を受け取ることができます。

まとめ

このセクションでは、 NetAppのベクトル データベース ソリューションについて説明します。

まとめ

最後に、このドキュメントでは、Milvus や pgvector などのベクトル データベースをNetAppストレージ ソリューションに導入および管理する方法の包括的な概要を示します。 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ストラクチャガイドラインについて説明し、ファイルとオブジェクトストアを通じて AWS FSx ONTAPの Milvus データベースを検証しました。

私たちは、NetApp のファイルとオブジェクトの二重性について調査し、ベクター データベース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、NetApp 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、クローン機能を提供し、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。

このドキュメントでは、NetApp のハイブリッド クラウド ソリューションがオンプレミスとクラウド環境全体でデータのレプリケーションと保護を提供し、シームレスで安全なデータ管理エクスペリエンスを実現する方法についても詳しく説明します。 NetApp ONTAP上の Milvus や pgvecto などのベクトル データベースのパフォーマンス検証に関する洞察を提供し、その効率性と拡張性に関する貴重な情報を提供しました。

最後に、LLM を使用した RAG と NetApp の内部 ChatAI という 2 つの生成 AI の使用例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されている概念と実践の実際の応用と利点を強調しています。全体として、このドキュメントは、NetApp の強力なストレージ ソリューションを活用してベクター データベースを管理しようとしているすべての人にとって包括的なガイドとして役立ちます。

謝辞

著者は、このホワイト ペーパーをNetApp の顧客とNetApp の分野にとって価値あるものにするためにフィードバックとコメントを提供してくれた以下の協力者およびその他の関係者に心から感謝の意を表します。

1. Sathish Thyagarajan 氏、 NetApp 、 ONTAP AI & Analytics、テクニカル マーケティング エンジニア
2. マイク・オグルスビー、 NetAppテクニカル マーケティング エンジニア
3. AJ マハジャン、 NetAppシニア ディレクター
4. Joe Scott、 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
5. Puneet Dhawan、 NetApp 、FSX製品管理担当シニアディレクター
6. Yuval Kalderon、 NetApp FSx 製品チーム シニア プロダクト マネージャー

詳細情報の入手方法

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、次のドキュメントや 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
- NetApp製品ドキュメントhttps://www.netapp.com/support-and-training/documentation/[]

- インスタクラスタ -"installcluster ドキュメント"

バージョン履歴

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

付録A: Values.yaml

このセクションでは、 NetAppベクトル データベース ソリューションで使用される値のサンプル YAML コードを示します。

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


## 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

このセクションでは、ベクター データベース用のデータを準備するために使用されるサンプル Python スクリプトを示します。

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

このセクションには、 NetAppベクター データベース ソリューション内のベクター データベースを検証するために使用できるサンプル Python スクリプトが含まれています。

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

このセクションには、 NetAppのベクトル データベース ソリューションのサンプル YAML コードが含まれています。

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

商標に関する情報