



Solution de base de données Vector avec NetApp

NetApp Solutions

NetApp
May 03, 2024

Sommaire

Solution de base de données Vector avec NetApp	1
Introduction	2
Présentation de la solution	2
Base de données vectorielle	3
Exigences technologiques	7
Procédure de déploiement	8
Présentation de la solution	9
Base de données Vector avec Instacluster à l'aide de PostgreSQL: Pgvector	45
Cas d'utilisation de la base de données Vector	45
Conclusion	48
Annexe A : valeurs.yaml	49
Annexe B : prepare_data_netapp_new.py	70
Annexe C : verify_data_netapp.py	74
Annexe D : docker-compose.yml	77

Solution de base de données Vector avec NetApp

Karthikeyan Nagalingam et Rodrigo Nascimento, NetApp

Ce document contient une exploration approfondie du déploiement et de la gestion des bases de données vectorielles, telles que Milvus et pgvector une extension PostgreSQL open source, à l'aide des solutions de stockage NetApp. Il détaille les conseils d'infrastructure pour l'utilisation du stockage objet NetApp ONTAP et StorageGRID et valide l'application de la base de données Milvus dans AWS FSX pour NetApp ONTAP. Il présente la dualité fichier/objet de NetApp et son utilité pour les bases de données vectorielles et les applications prenant en charge les codes vectoriels. Il met l'accent sur les fonctionnalités de sauvegarde et de restauration d'SnapCenter, produit de gestion d'entreprise de NetApp, pour les bases de données vectorielles, qui assurent l'intégrité et la disponibilité des données. Le document approfondit son analyse de la solution de cloud hybride NetApp, et aborde son rôle dans la réplication et la protection des données dans les environnements sur site et dans le cloud. Il comprend un aperçu de la validation des performances des bases de données vectorielles sur NetApp ONTAP et se termine par deux cas d'utilisation pratiques de l'IA générative : RAG avec LLM et le Charai interne de NetApp. Ce document fournit un guide complet sur l'exploitation des solutions de stockage NetApp pour la gestion des bases de données vectorielles.

L'architecture de référence se concentre sur les éléments suivants :

1. ["Introduction"](#)
2. ["Présentation de la solution"](#)
3. ["Base de données vectorielle"](#)
4. ["Exigences technologiques"](#)
5. ["Procédure de déploiement"](#)
6. ["Présentation de la vérification de la solution"](#)
 - ["Configuration de clusters Milvus avec Kubernetes sur site"](#)
 - ["Milvus avec Amazon FSxN pour NetApp ONTAP : dualité fichier/objet"](#)
 - ["Protection de base de données Vector à l'aide de NetApp SnapCenter."](#)
 - ["Reprise sur incident à l'aide de NetApp SnapMirror"](#)
 - ["Validation des performances"](#)
7. ["Base de données Vector avec Instacluster à l'aide de PostgreSQL: Pgvector"](#)
8. ["Cas d'utilisation de la base de données Vector"](#)
9. ["Conclusion"](#)
10. ["Annexe A : valeurs.yaml"](#)
11. ["Annexe B : prepare_data_netapp_new.py"](#)
12. ["Annexe C : verify_data_netapp.py"](#)

Introduction

Introduction

Les bases de données vectorielles répondent efficacement aux défis qui sont conçus pour gérer la complexité de la recherche sémantique dans les modèles de langage large (LLMs) et l'intelligence artificielle générative (ai). Contrairement aux systèmes traditionnels de gestion des données, les bases de données vectorielles sont capables de traiter et de rechercher à travers différents types de données, y compris des images, des vidéos, du texte, de l'audio, et d'autres formes de données non structurées, en utilisant le contenu des données elles-mêmes plutôt que des étiquettes ou des balises.

Les limites des systèmes de gestion de bases de données relationnelles (RDBMS) sont bien documentées, notamment dans leurs difficultés liées aux représentations de données massives et aux données non structurées communes aux applications d'IA. Les SGBDR nécessitent souvent un processus long et source d'erreurs, qui consiste à aplatir les données en structures plus gérables, ce qui entraîne des retards et des inefficacités dans les recherches. Les bases de données vectorielles, en revanche, sont conçues pour contourner ces problèmes et offrir une solution plus efficace et plus précise pour la gestion et la recherche de données complexes et dimensionnelles, facilitant ainsi l'avancement des applications d'IA.

Ce document fournit un guide complet aux clients qui utilisent ou prévoient d'utiliser des bases de données vectorielles, en détaillant les bonnes pratiques d'utilisation des bases de données vectorielles sur des plateformes telles que NetApp ONTAP, NetApp StorageGRID, Amazon FSx pour NetApp ONTAP et SnapCenter. Le contenu fourni dans ce document couvre une grande variété de sujets :

- Recommandations d'infrastructure pour les bases de données vectorielles, telles que Milvus, fournies par le stockage NetApp via NetApp ONTAP et le stockage objet StorageGRID
- Validation de la base de données Milvus dans AWS FSX pour NetApp ONTAP via un magasin de fichiers et d'objets
- S'intègre dans la dualité fichier/objet de NetApp, démontrant ainsi son utilité pour les données dans les bases de données vectorielles et d'autres applications.
- SnapCenter, le produit NetApp de gestion de la protection des données, propose les fonctionnalités de sauvegarde et de restauration des données de bases de données vectorielles.
- Comment le cloud hybride NetApp assure la réplication et la protection des données dans l'ensemble des environnements sur site et cloud.
- Fournit des informations sur la validation des performances de bases de données vectorielles telles que Milvus et pgvector sur NetApp ONTAP.
- Deux cas d'utilisation spécifiques : récupération de la génération augmentée (RAG) avec des modèles de langage large (LLM) et Chai de l'équipe IT de NetApp, offrant ainsi des exemples pratiques des concepts et des pratiques présentés.

Présentation de la solution

Présentation de la solution

Cette solution présente les avantages et les capacités spécifiques de NetApp pour relever les défis auxquels sont confrontés les clients de bases de données vectorielles. Avec NetApp ONTAP, StorageGRID, les solutions cloud de NetApp et SnapCenter, les clients peuvent valoriser considérablement leurs opérations. Non seulement ces outils permettent de résoudre les problèmes existants, mais ils améliorent également l'efficacité

et la productivité, contribuant ainsi à la croissance globale de l'entreprise.

Pourquoi choisir NetApp ?

- Les offres NetApp, telles que ONTAP et StorageGRID, permettent de séparer le stockage et le calcul pour une utilisation optimale des ressources selon des besoins spécifiques. Cette flexibilité permet aux clients de faire évoluer leur stockage indépendamment à l'aide des solutions de stockage NetApp.
- En exploitant les contrôleurs de stockage de NetApp, les clients peuvent transmettre efficacement des données à leur base de données vectorielle à l'aide des protocoles NFS et S3. Ces protocoles facilitent le stockage des données des clients et la gestion de l'index de la base de données Vector, éliminant ainsi la nécessité de plusieurs copies des données accessibles via les méthodes fichier et objet.
- NetApp ONTAP offre une prise en charge native du stockage NAS et objet sur les principaux fournisseurs de services cloud comme AWS, Azure et Google Cloud. Cette compatibilité étendue garantit une intégration transparente, ce qui permet la mobilité des données des clients, l'accessibilité globale, la reprise après incident, l'évolutivité dynamique et de hautes performances.
- Grâce aux puissantes fonctionnalités de gestion des données de NetApp, les clients sont assurés que leurs données sont bien protégées contre les risques et les menaces potentiels. NetApp privilégie la sécurité des données en offrant aux clients la tranquillité d'esprit en matière de sécurité et d'intégrité de leurs informations stratégiques.

Base de données vectorielle

Base de données vectorielle

Une base de données vectorielle est un type spécialisé de base de données conçu pour gérer, indexer et rechercher des données non structurées à l'aide d'encodages issus de modèles de machine learning. Au lieu d'organiser les données dans un format tabulaire traditionnel, il organise les données sous forme de vecteurs à haute dimension, également appelés embeddings vectoriels. Cette structure unique permet à la base de données de traiter des données complexes et multidimensionnelles plus efficacement et avec plus de précision.

L'une des principales fonctionnalités d'une base de données vectorielle est son utilisation de l'IA générative pour effectuer des analyses. Cela inclut les recherches de similarité, où la base de données identifie les points de données qui sont comme une entrée donnée, et la détection d'anomalies, où elle peut détecter les points de données qui dévient de manière significative de la norme.

En outre, les bases de données vectorielles sont bien adaptées pour traiter les données temporelles, ou les données horodatées. Ce type de données fournit des informations sur ce qui s'est passé et quand il s'est produit, dans l'ordre et par rapport à tous les autres événements dans un système INFORMATIQUE donné. Cette capacité à gérer et à analyser les données temporelles rend les bases de données vectorielles particulièrement utiles pour les applications qui nécessitent une compréhension des événements au fil du temps.

Avantages de la base de données vectorielle pour le ML et l'IA :

- Recherche multidimensionnelle : les bases de données vectorielles excellent dans la gestion et la récupération de données de haute dimension, souvent générées dans les applications d'IA et de ML.
- Évolutivité : ces systèmes peuvent évoluer efficacement pour gérer d'importants volumes de données, tout en prenant en charge la croissance et l'extension de vos projets d'IA et de ML.
- Flexibilité : les bases de données vectorielles offrent un haut degré de flexibilité, permettant l'adaptation de divers types et structures de données.

- Performances : elles offrent une gestion et une récupération des données haute performance, essentielles pour la vitesse et l'efficacité des opérations d'IA et de ML.
- Indexation personnalisable : les bases de données vectorielles offrent des options d'indexation personnalisables, permettant une organisation et une récupération optimisées des données en fonction de besoins spécifiques.

Bases de données vectorielles et cas d'utilisation

Cette section fournit des bases de données vectorielles et des détails sur leur utilisation.

Faiss et Scann

Ce sont des bibliothèques qui servent d'outils essentiels dans le domaine de la recherche vectorielle. Ces bibliothèques offrent des fonctionnalités essentielles pour la gestion et la recherche à travers les données vectorielles, ce qui en fait des ressources précieuses dans ce domaine spécialisé de la gestion des données.

ElasticSearch

Il s'agit d'un moteur de recherche et d'analytique largement utilisé, qui a récemment intégré des fonctionnalités de recherche vectorielle. Cette nouvelle fonctionnalité améliore sa fonctionnalité, ce qui lui permet de gérer et de rechercher plus efficacement les données vectorielles.

Pomme de pin

Il s'agit d'une base de données vectorielle robuste dotée d'un ensemble unique de fonctionnalités. Elle prend en charge les vecteurs denses et clairsemés dans sa fonctionnalité d'indexation, ce qui améliore sa flexibilité et son adaptabilité. L'un de ses principaux atouts réside dans sa capacité à combiner des méthodes de recherche classiques avec la recherche vectorielle dense basée sur l'IA pour créer une approche de recherche hybride qui exploite le meilleur des deux mondes.

Principalement basé sur le cloud, Pinecone est conçu pour les applications de machine learning et s'intègre parfaitement avec diverses plateformes, notamment GCP, AWS, Open ai, GPT-3, GPT-3.5, GPT-4, catgut plus, Elasticsearch, Haystack, et bien plus encore. Notez que Pinecone est une plateforme open source disponible en tant que service (SaaS).

Compte tenu de ses fonctionnalités avancées, Pinecone est particulièrement adapté au secteur de la cybersécurité, où ses capacités performantes de recherche et de recherche hybride peuvent être exploitées de manière efficace pour détecter les menaces et y répondre.

Couleur

Il s'agit d'une base de données vectorielle dotée d'une API Core dotée de quatre fonctions principales, dont l'une comprend un magasin de documents vectoriels en mémoire. Il utilise également la bibliothèque des transformateurs de visage pour vectoriser les documents, en améliorant sa fonctionnalité et sa polyvalence. Chroma est conçu pour fonctionner à la fois dans le cloud et sur site, offrant une flexibilité basée sur les besoins des utilisateurs. En particulier, il excelle dans les applications audio, ce qui en fait un excellent choix pour les moteurs de recherche audio, les systèmes de recommandation de musique et d'autres cas d'utilisation audio.

Weaviate

Cette base de données polyvalente permet aux utilisateurs de vectoriser leur contenu à l'aide de modules intégrés ou de modules personnalisés, offrant ainsi une flexibilité adaptée à des besoins spécifiques. Il propose des solutions entièrement gérées et auto-hébergées, répondant à diverses préférences de déploiement.

L'une des caractéristiques principales de Weaviate est sa capacité à stocker à la fois des vecteurs et des objets, améliorant ainsi ses capacités de traitement des données. Il est largement utilisé pour un large éventail d'applications, y compris la recherche sémantique et la classification des données dans les systèmes ERP. Dans le secteur du commerce électronique, il alimente les moteurs de recherche et de recommandation. Weaviate est également utilisé pour la recherche d'images, la détection des anomalies, l'harmonisation automatisée des données et l'analyse des menaces de cybersécurité, afin de démontrer sa polyvalence dans plusieurs domaines.

Redis

Redis est une base de données vectorielle hautes performances, connue pour son stockage en mémoire rapide, qui offre une faible latence pour les opérations de lecture/écriture. Il s'agit donc d'un excellent choix pour les systèmes de recommandation, les moteurs de recherche et les applications d'analytique qui nécessitent un accès rapide aux données.

Redis prend en charge diverses structures de données pour les vecteurs, y compris les listes, les ensembles et les ensembles triés. Il fournit également des opérations vectorielles telles que le calcul des distances entre les vecteurs ou la recherche d'intersections et de raccords. Ces fonctionnalités sont particulièrement utiles pour la recherche par similarité, la mise en grappe et les systèmes de recommandation basés sur le contenu.

En termes d'évolutivité et de disponibilité, Redis excelle dans la gestion des charges de travail à haut débit et propose la réplication des données. Il s'intègre également bien à d'autres types de données, y compris les bases de données relationnelles traditionnelles (SGBDR).

Redis inclut une fonction publier/s'abonner (Pub/Sub) pour les mises à jour en temps réel, ce qui est utile pour gérer les vecteurs en temps réel. De plus, Redis est léger et simple à utiliser, ce qui en fait une solution conviviale pour la gestion des données vectorielles.

Milvus

Il s'agit d'une base de données vectorielle polyvalente qui propose une API comme un magasin de documents, tout comme MongoDB. Elle se distingue par sa prise en charge d'un large éventail de types de données, ce qui en fait un choix populaire dans les domaines de la data science et de l'apprentissage machine.

L'une des caractéristiques uniques de Milvus est sa capacité de multi-vectorisation, qui permet aux utilisateurs de spécifier à l'exécution le type de vecteur à utiliser pour la recherche. De plus, elle utilise Knowhere, une bibliothèque située au sommet d'autres bibliothèques comme Faiss, pour gérer la communication entre les requêtes et les algorithmes de recherche vectorielle.

Milvus offre également une intégration transparente avec les workflows d'apprentissage machine grâce à sa compatibilité avec PyTorch et TensorFlow. Cela en fait un excellent outil pour une gamme d'applications, y compris l'e-commerce, l'analyse d'images et de vidéos, la reconnaissance d'objets, la recherche de similarité d'images et la récupération d'images basée sur le contenu. Dans le domaine du traitement du langage naturel, Milvus est utilisé pour la mise en grappe de documents, la recherche sémantique et les systèmes de réponse aux questions.

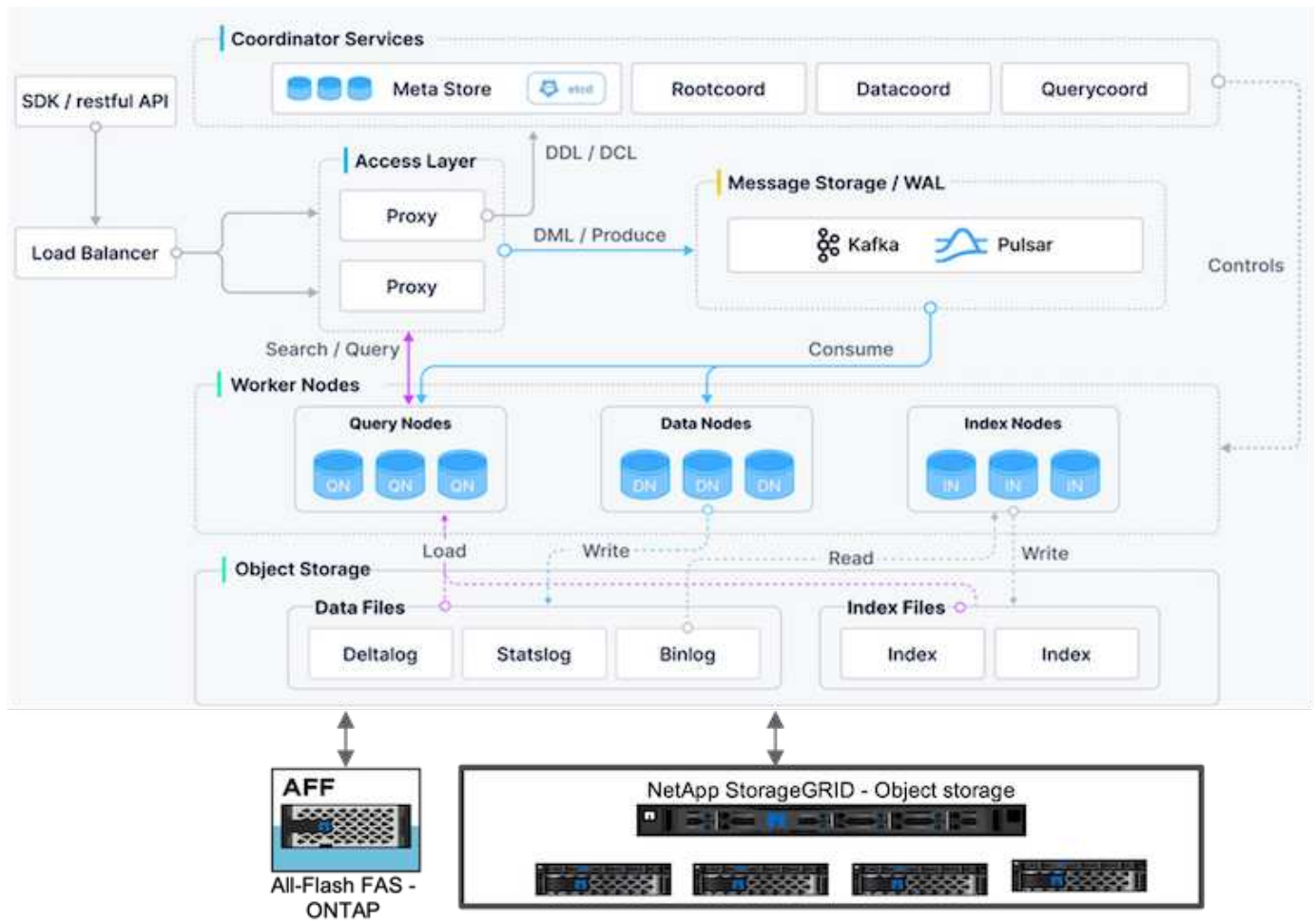
Pour cette solution, nous avons sélectionné milvus pour la validation de la solution. Pour les performances, nous avons utilisé milvus et postgres(pgvector.RS).

Pourquoi avons-nous choisi milvus pour cette solution ?

- Open-Source : Milvus est une base de données vectorielle open-source qui encourage le développement et les améliorations communautaires.
- Intégration IA : cette solution exploite l'intégration d'applications d'IA et de recherche par similarité pour améliorer les fonctionnalités des bases de données vectorielles.

- Gestion de grands volumes : Milvus a la capacité de stocker, d'indexer et de gérer plus d'un milliard de vecteurs d'intégration générés par les modèles de réseaux neuronaux profonds (DNN) et de machine learning (ML).
- Convivial : il est facile à utiliser et sa configuration prend moins d'une minute. Milvus propose également des kits de développement logiciel pour différents langages de programmation.
- Vitesse : elle offre des vitesses de récupération jusqu'à 10 fois supérieures à celles de certaines solutions.
- Évolutivité et disponibilité : Milvus est extrêmement évolutif et offre des options d'évolutivité horizontale et verticale en fonction des besoins.
- Riche en fonctionnalités : il prend en charge différents types de données, le filtrage des attributs, la prise en charge des fonctions définies par l'utilisateur (UDF), les niveaux de cohérence configurables et le temps de déplacement, ce qui en fait un outil polyvalent pour diverses applications.

Présentation de l'architecture Milvus



Cette section fournit des composants de levier supérieur et des services sont utilisés dans l'architecture Milvus.

- * Couche d'accès – elle est composée d'un groupe de proxys sans état et sert de couche frontale du système et de point de terminaison pour les utilisateurs.
- * Service coordinateur – il attribue les tâches aux nœuds de travail et agit comme le cerveau d'un système. Il a trois types de coordinateurs : Coord racine, Coord données et coord requête.
- * Nœuds de travail : il suit l'instruction du service coordinateur et exécute les nœuds de travail déclenchés par l'utilisateur DML/DDL commands. it a trois types de nœuds de travail tels que noeud de requête, noeud de données et noeud d'index.

* Stockage : il est responsable de la persistance des données. Elle comprend le méta-stockage, le courtier en journaux et le stockage objet. Le stockage NetApp tel que ONTAP et StorageGRID fournit à Milvus un stockage objet et un stockage basé sur des fichiers pour les données clients et les données de base de données vectorielles.

Exigences technologiques

Exigences technologiques

Les configurations matérielles et logicielles décrites ci-dessous ont été utilisées pour la majorité des validations réalisées dans ce document, à l'exception des performances. Ces configurations vous aideront à configurer votre environnement. Toutefois, notez que les composants spécifiques peuvent varier en fonction des besoins spécifiques du client.

Configuration matérielle requise

Sous-jacent	Détails
Paire haute disponibilité de la baie de stockage NetApp AFF	<ul style="list-style-type: none"> * A800 * ONTAP 9.14.1 * 48 SSD-NVM 3,49 TO * Deux volumes de groupe flexibles : métadonnées et données. * Le volume NFS de métadonnées a 12 x volumes persistants avec 250 Go. * Les données sont un volume ONTAP NAS S3
6 SERVEURS PRIMERGY RX2540 M4 DE FUJITSU	<ul style="list-style-type: none"> * 64 processeurs * Processeur Intel® Xeon® Gold 6142 à 2,60 GHz * 256 mémoire physique GM * 1 port réseau 100 GbE
Mise en réseau	100 GbE
StorageGRID	<ul style="list-style-type: none"> * 1 x SG100, 3 x SGF6024 * 3 x 24 x 7,68 To

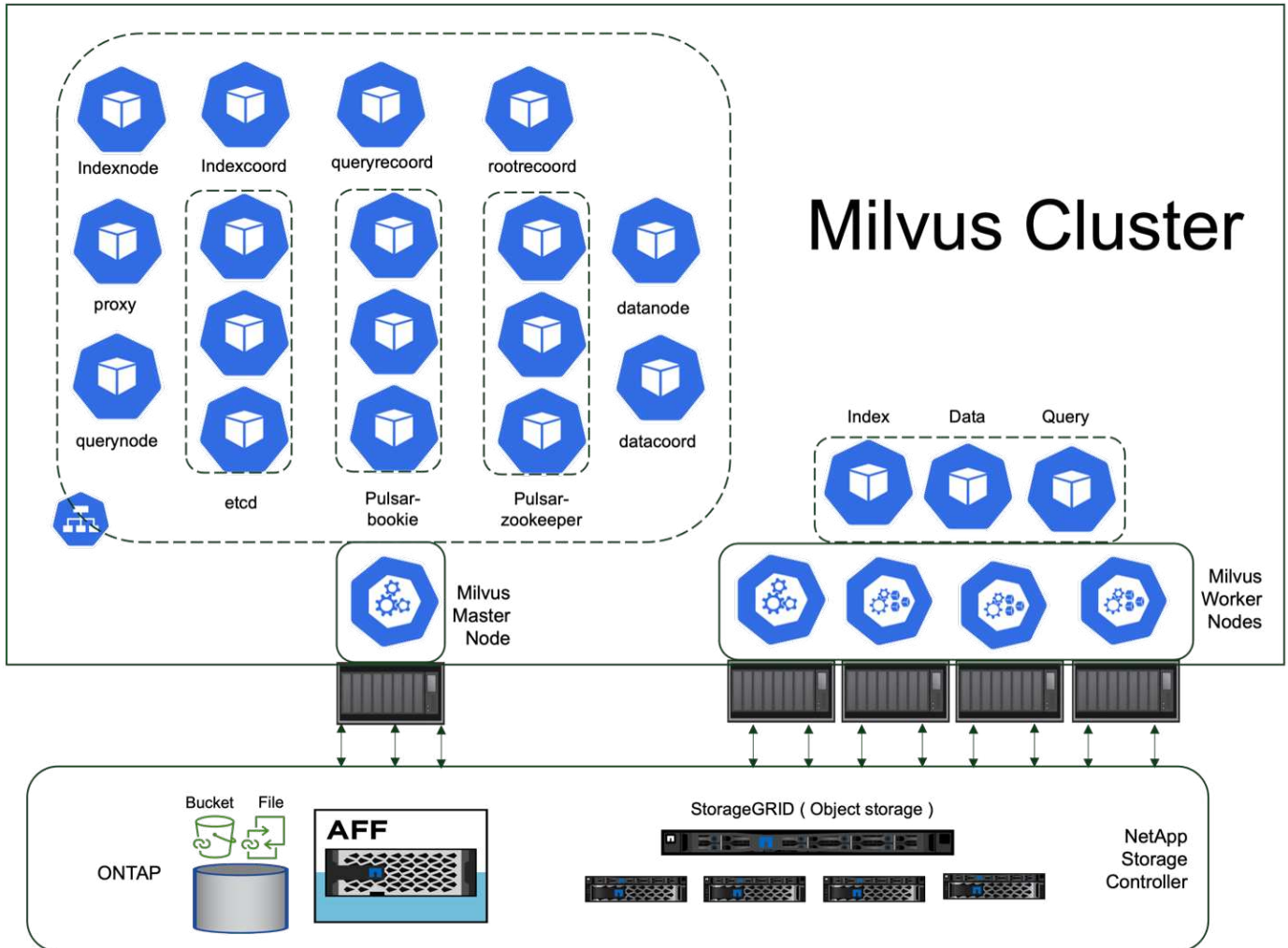
Configuration logicielle requise

Logiciel	Détails
Cluster Milvus	<ul style="list-style-type: none"> * GRAPHIQUE - milvus-4.1.11. * Version de l'APPLICATION – 2.3.4 * Bundles dépendants tels que bookkeeper, zookeeper, pulsar, etcd, proxy, querynode, worker
Kubernetes	<ul style="list-style-type: none"> * Cluster K8s à 5 nœuds * 1 nœud maître et 4 nœuds travailleur * Version – 1.7.2
Python	*3.10.12.

Procédure de déploiement

Procédure de déploiement

Dans cette section de déploiement, nous avons utilisé la base de données vecteur Milvus avec Kubernetes pour la configuration de l'atelier, comme ci-dessous.



Le système de stockage NetApp fournit le stockage pour le cluster afin de conserver les données des clients et les données des clusters.

Configuration du stockage NetApp – ONTAP

- Initialisation du système de stockage
- Création de machines virtuelles de stockage (SVM)
- Affectation d'interfaces réseau logiques
- Configuration et licences NFS et S3

Veuillez suivre les étapes ci-dessous pour NFS (Network File System) :

1. Créez un volume FlexGroup pour NFSv4. Dans notre configuration de cette validation, nous avons utilisé 48 SSD, 1 SSD dédié au volume racine du contrôleur et 47 SSD répartis sur NFSv4. Vérifiez que la règle d'export NFS pour le volume FlexGroup possède des autorisations de lecture/écriture pour le réseau de

nœuds Kubernetes (K8s). Si ces autorisations ne sont pas en place, accordez des autorisations de lecture/écriture (rw) au réseau de nœuds K8s.

2. Sur tous les nœuds K8s, créez un dossier et montez le volume FlexGroup sur ce dossier via une interface logique (LIF) sur chaque nœud K8s.

Suivez les étapes ci-dessous pour NAS S3 (Network Attached Storage simple Storage Service) :

1. Créez un volume FlexGroup pour NFS.
2. Configurez un serveur-stockage-objet avec HTTP activé et l'état admin défini sur « UP » à l'aide de la commande « vserver object-store-Server create ». Vous avez la possibilité d'activer HTTPS et de définir un port d'écoute personnalisé.
3. Créez un utilisateur object-store-Server en utilisant la commande « vserver object-store-Server user create -user <username> ».
4. Pour obtenir la clé d'accès et la clé secrète, vous pouvez exécuter la commande suivante : « set diag; vserver object-store-Server user show -user <username> ». Cependant, ces clés seront fournies lors du processus de création de l'utilisateur ou peuvent être récupérées à l'aide d'appels API REST.
5. Créez un groupe de stockage-serveur d'objets à l'aide de l'utilisateur créé à l'étape 2 et accordez l'accès. Dans cet exemple, nous avons fourni « FullAccess ».
6. Pour créer un compartiment NAS, définissez son type sur « nas » et indiquez le chemin d'accès au volume NFSv3. Il est également possible d'utiliser un compartiment S3 à cet effet.

Configuration du stockage NetApp – StorageGRID

1. Installez le logiciel StorageGRID.
2. Créer un locataire et un compartiment.
3. Créez un utilisateur avec l'autorisation requise.

Pour plus d'informations, consultez <https://docs.netapp.com/us-en/storagegrid-116/primer/index.html>

Présentation de la solution

Nous avons procédé à une validation complète de la solution dans cinq domaines clés dont les détails sont présentés ci-dessous. Chaque section traite des défis auxquels sont confrontés les clients, des solutions fournies par NetApp et des avantages qu'ils en tirent.

1. ["Configuration de clusters Milvus avec Kubernetes sur site"](#)
Challenges des clients : faire évoluer indépendamment les ressources de stockage et de calcul, et gérer efficacement l'infrastructure et les données. Dans cette section, nous détaillons le processus d'installation d'un cluster Milvus sur Kubernetes avec un contrôleur de stockage NetApp pour les données de cluster et les données client.
2. ["Milvus avec Amazon FSxN pour NetApp ONTAP : dualité fichier/objet"](#)
Dans cette section, pourquoi nous devons déployer une base de données vectorielle dans le cloud ainsi que des étapes pour déployer une base de données vectorielle (milvus standalone) dans Amazon FSxN pour NetApp ONTAP dans des conteneurs docker.
3. ["Protection de base de données Vector à l'aide de NetApp SnapCenter."](#)
Dans cette section, nous allons examiner comment SnapCenter protège les données de la base de données vectorielle et les données Milvus résidant dans ONTAP. Dans cet exemple, nous avons utilisé un compartiment NAS (milvusdbvol1) dérivé d'un volume ONTAP NFS (vol1) pour les données clients, et un volume NFS distinct (vectordbpv) pour les données de configuration du cluster Milvus.

4. "Reprise sur incident à l'aide de NetApp SnapMirror"

Dans cette section, nous parlerons de l'importance de la reprise sur incident pour la base de données vectorielle et de la manière dont snapmirror, produit de reprise sur incident de NetApp, fournit une solution de reprise sur incident pour la base de données vectorielle.

5. "Validation des performances"

Dans cette section, nous allons examiner la validation des performances des bases de données vectorielles, telles que Milvus et pgvecto.RS, en nous concentrant sur leurs caractéristiques de performances de stockage, telles que le profil d'E/S et le contrôleur de stockage NetApp qui prennent en charge les charges de travail RAG et d'inférence dans le cadre du cycle de vie LLM. Nous évaluerons et identifierons les différences de performances éventuelles lorsque ces bases de données sont combinées à la solution de stockage ONTAP. Notre analyse sera basée sur des indicateurs clés de performance, comme le nombre de requêtes traitées par seconde (QPS).

Configuration de clusters Milvus avec Kubernetes sur site

Configuration de clusters Milvus avec Kubernetes sur site

Challenges des clients : faire évoluer leurs ressources de stockage et de calcul de manière indépendante, gérer efficacement l'infrastructure et gérer les données

Ensemble, Kubernetes et les bases de données vectorielles forment une solution puissante et évolutive pour la gestion des opérations de données volumineuses. Kubernetes optimise les ressources et gère les conteneurs, tandis que les bases de données vectorielles gèrent efficacement les recherches de similarités et de données de grande taille. Cette combinaison permet de traiter rapidement les requêtes complexes sur des datasets volumineux et d'évoluer de manière transparente avec des volumes de données croissants, ce qui en fait la solution idéale pour les applications Big Data et les charges de travail d'IA.

1. Dans cette section, nous détaillons le processus d'installation d'un cluster Milvus sur Kubernetes avec un contrôleur de stockage NetApp pour les données de cluster et les données client.
2. Pour installer un cluster Milvus, des volumes persistants (PVS) sont nécessaires au stockage des données issues de divers composants du cluster Milvus. Ces composants comprennent les données etcd (trois instances), pulsar-bookie-journal (trois instances), pulsar-bookie-ledgers (trois instances) et pulsar-zookeeper-data (trois instances).



Dans le cluster milvus, nous pouvons utiliser pulsar ou kafka pour le moteur sous-jacent prenant en charge le stockage fiable du cluster Milvus et la publication/l'abonnement des flux de messages. Pour Kafka avec NFS, NetApp a apporté des améliorations à ONTAP 9.12.1 et versions ultérieures. Ces améliorations, ainsi que les modifications NFSv4.1 et Linux incluses dans RHEL 8.7 ou 9.1 et versions ultérieures, résolvent le problème de « changement de nom silly » qui peut survenir lors de l'exécution de Kafka avec NFS. Si vous souhaitez obtenir des informations plus détaillées sur l'exécution de kafka avec la solution NFS NetApp, veuillez consulter - <https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-introduction.html>.

3. Nous avons créé un volume NFS unique à partir de NetApp ONTAP et établi 12 volumes persistants, chacun avec 250 Go de stockage. La taille du stockage peut varier en fonction de la taille du cluster. Par exemple, nous disposons d'un autre cluster où chaque volume persistant dispose de 50 Go. Veuillez vous reporter à l'un des fichiers PV YAML pour plus de détails ; nous avons 12 fichiers de ce type au total. Dans chaque fichier, le nom de classe de stockage est défini sur « par défaut », et le stockage et le chemin sont uniques à chaque PV.

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

4. Exécutez la commande 'kubectl Apply' pour chaque fichier YAML PV pour créer les volumes persistants, puis vérifiez leur création à l'aide de 'kubectl get pv'

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

5. Milvus prend en charge les solutions de stockage objet telles que MiniO, Azure Blob et S3 pour le stockage des données des clients. Dans ce guide, nous utilisons S3. Les étapes suivantes s'appliquent aux magasins d'objets ONTAP S3 et StorageGRID. Nous utilisons Helm pour déployer le cluster Milvus. Téléchargez le fichier de configuration, Values.yaml, à partir de l'emplacement de téléchargement Milvus. Veuillez vous reporter à l'annexe pour le fichier values.yaml que nous avons utilisé dans ce document.
6. Assurez-vous que la 'StorageClass' est définie sur 'par défaut' dans chaque section, y compris celles du journal, de l'ETCD, du zookeeper et du bookkeeper.
7. Dans la section MiniO, désactivez MiniO.
8. Créez un compartiment NAS à partir d'un stockage objet ONTAP ou StorageGRID et ajoutez-les dans un S3 externe avec les identifiants du stockage objet.

```
#####
# 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: "milvusdbvoll1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false
```

9. Avant de créer le cluster Milvus, assurez-vous que la demande de volume persistant ne dispose pas de ressources préexistantes.

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

10. Utilisez Helm et le fichier de configuration Values.yaml pour installer et démarrer le cluster Milvus.

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

11. Vérifiez l'état des demandes de volume persistant.

```

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. Vérifier l'état des pods.

```

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

```

Assurez-vous que l'état des modules est « en cours d'exécution » et qu'ils fonctionnent comme prévu

13. Testez l'écriture et la lecture de données dans Milvus et le stockage objet NetApp.

- Écrivez les données à l'aide du programme Python « Prepare_Data_netapp_New.py ».


```

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

```

- Lisez les données à l'aide du fichier Python « verify_Data_netapp.py ».

```

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

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

=== Start Creating index IVF_FLAT   ===

=== Start loading                    ===

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

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

```

```

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

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

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

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

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

```

Sur la base de la validation ci-dessus, l'intégration de Kubernetes avec une base de données vectorielle, comme l'illustre le déploiement d'un cluster Milvus sur Kubernetes à l'aide d'un contrôleur de stockage NetApp, offre aux clients une solution robuste, évolutive et efficace pour la gestion des opérations de données à grande échelle. Cette configuration permet aux clients de gérer des données à dimension élevée et d'exécuter des requêtes complexes de manière rapide et efficace. Elle constitue ainsi la solution idéale pour les applications Big Data et les workloads d'IA. L'utilisation des volumes persistants (PVS) pour divers composants du cluster, ainsi que la création d'un volume NFS unique à partir de NetApp ONTAP, assurent une utilisation optimale des ressources et une gestion optimale des

données. Le processus consistant à vérifier l'état des demandes de volume persistant et des pods, ainsi qu'à tester l'écriture et la lecture des données, garantit la fiabilité et la cohérence des opérations de données. L'utilisation du stockage objet ONTAP ou StorageGRID pour les données des clients renforce encore l'accessibilité et la sécurité des données. Cette configuration offre une solution de gestion des données résiliente et haute performance qui peut évoluer de manière transparente en fonction de l'évolution de vos besoins en termes de données.

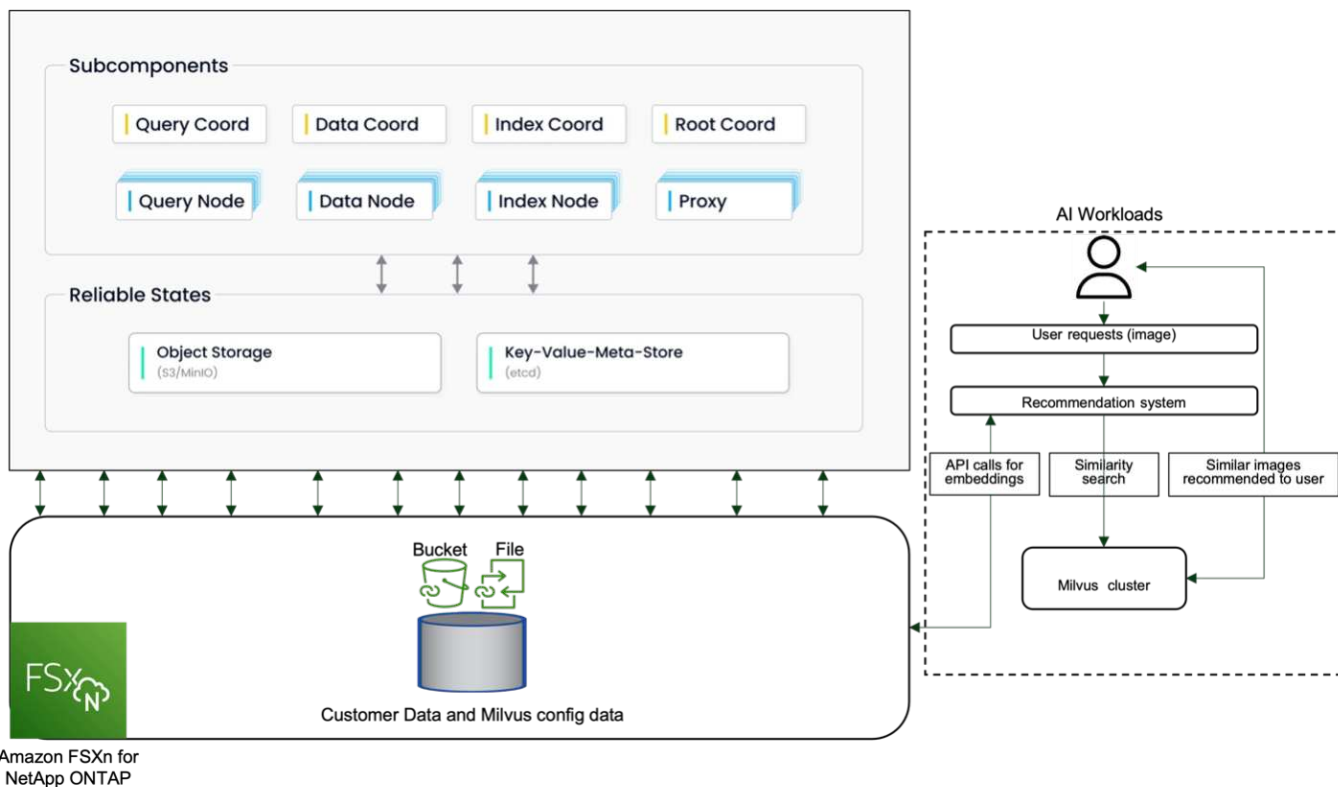
Milvus avec Amazon FSxN pour NetApp ONTAP : la dualité fichier/objet

Milvus avec Amazon FSxN pour NetApp ONTAP : dualité fichier/objet

Dans cette section, pourquoi nous devons déployer une base de données vectorielle dans le cloud ainsi que des étapes pour déployer une base de données vectorielle (milvus autonome) dans Amazon FSxN pour NetApp ONTAP dans des conteneurs docker.

Le déploiement d'une base de données vectorielle dans le cloud offre plusieurs avantages significatifs, notamment pour les applications nécessitant le traitement de données de grande dimension et l'exécution de recherches de similarité. Tout d'abord, le déploiement basé sur le cloud offre une évolutivité qui facilite l'ajustement des ressources pour répondre aux volumes croissants de données et aux charges de requêtes. Ainsi, la base de données peut gérer efficacement l'augmentation de la demande tout en maintenant des performances élevées. Deuxièmement, le déploiement cloud assure la haute disponibilité et la reprise après incident, car les données peuvent être répliquées sur plusieurs sites géographiques, ce qui réduit le risque de perte de données et garantit la continuité du service même en cas d'événements imprévus. Troisièmement, il offre une rentabilité puisque vous ne payez que les ressources que vous utilisez et peut évoluer à la hausse ou à la baisse en fonction de la demande, évitant ainsi un investissement initial substantiel pour le matériel. Enfin, le déploiement d'une base de données vectorielle dans le cloud permet d'améliorer la collaboration, car les données sont accessibles et partagées depuis n'importe où, ce qui facilite le travail en équipe et la prise de décision basée sur les données.

Veuillez vérifier l'architecture du milvus autonome avec Amazon FSxN pour NetApp ONTAP utilisé dans cette validation.



1. Créez une instance Amazon FSxN pour NetApp ONTAP et notez les détails du VPC, des groupes de sécurité VPC et du sous-réseau. Ces informations seront requises lors de la création d'une instance EC2. Vous trouverez plus de détails ici - <https://us-east-1.console.aws.amazon.com/fsx/home?region=us-east-1#file-system-create>
2. Créez une instance EC2, en vous assurant que le VPC, les groupes de sécurité et le sous-réseau correspondent à ceux de l'instance Amazon FSxN pour NetApp ONTAP.
3. Installez nfs-common à l'aide de la commande 'apt-get install nfs-common' et mettez à jour les informations sur le paquet à l'aide de 'udo apt-get update'.
4. Créez un dossier de montage et montez-le sur Amazon FSxN pour NetApp ONTAP.

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

5. Installez Docker et Docker compose à l'aide de 'apt-get install'.
6. Configurez un cluster Milvus à partir du fichier docker-compose.yaml, qui peut être téléchargé à partir du site Web Milvus.

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

7. Dans la section « volumes » du fichier docker-compose.yaml, mappez le point de montage NFS NetApp sur le chemin du conteneur Milvus correspondant, en particulier dans etcd, minio et standalone. Check "[Annexe D : docker-compose.yaml](#)" pour plus de détails sur les modifications de yaml
8. Vérifiez les dossiers et fichiers montés.

```

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
vectordbvoll
ubuntu@ip-172-31-29-98:~$

```

9. Exécutez 'docker-compose up -d' à partir du répertoire contenant le fichier docker-compose.yml.
10. Vérifier l'état du conteneur Milvus.

```

ubuntu@ip-172-31-29-98:~$ sudo docker-compose ps

```

Name	Command	State
Ports		

milvus-etcd	etcd -advertise-client-url ...	Up (healthy)
2379/tcp, 2380/tcp		
milvus-minio	/usr/bin/docker-entrypoint ...	Up (healthy)
0.0.0.0:9000->9000/tcp, :::9000->9000/tcp, 0.0.0.0:9001-		
>9001/tcp, :::9001->9001/tcp		
milvus-standalone	/tini -- milvus run standalone	Up (healthy)
0.0.0.0:19530->19530/tcp, :::19530->19530/tcp, 0.0.0.0:9091-		
>9091/tcp, :::9091->9091/tcp		

```

ubuntu@ip-172-31-29-98:~$
ubuntu@ip-172-31-29-98:~$ ls -ltrh /home/ubuntu/milvusvectordb/volumes/
total 12K
drwxr-xr-x 3 root root 4.0K Apr  4 20:21 etcd
drwxr-xr-x 4 root root 4.0K Apr  4 20:21 minio
drwxr-xr-x 5 root root 4.0K Apr  4 20:21 milvus
ubuntu@ip-172-31-29-98:~$

```

11. Pour valider la fonctionnalité de lecture et d'écriture de la base de données vectorielle et de ses données dans Amazon FSxN pour NetApp ONTAP, nous avons utilisé le kit de développement logiciel Python Milvus et un exemple de programme PyMilvus. Installez les paquets nécessaires à l'aide de 'apt-get install python3-numpy python3-pip' et installez PyMilvus à l'aide de 'pic3 install pymilvus'.
12. Valider les opérations d'écriture et de lecture des données à partir d'Amazon FSxN pour NetApp ONTAP

dans la base de données vectorielle.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3
prepare_data_netapp_new.py
=== start connecting to Milvus      ===
=== Milvus host: localhost          ===
Does collection hello_milvus_ntapnew_sc exist in Milvus: True
=== Drop collection - hello_milvus_ntapnew_sc ===
=== Drop collection - hello_milvus_ntapnew_sc2 ===
=== Create collection `hello_milvus_ntapnew_sc` ===
=== Start inserting entities        ===
Number of entities in hello_milvus_ntapnew_sc: 9000
root@ip-172-31-29-98:~/pymilvus/examples# find
/home/ubuntu/milvusvectordb/
...
<removed content to save page space >
...
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/b3def25f-c117-4fba-8256-96cb7557cd6c
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/b3def25f-c117-4fba-8256-96cb7557cd6c/part.1
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/103/4487898457
91411923/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
/448789845791611912/448789845791611913/448789845791611939/0/448789845791
411924
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/0/448789845791
411924/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
/448789845791611912/448789845791611913/448789845791611939/1/448789845791
411925
/home/ubuntu/milvusvectordb/volumes/minio/a-bucket/files/insert_log
/448789845791611912/448789845791611913/448789845791611939/1/448789845791
411925/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  
/448789845791611912/448789845791611913/448789845791611939/100/4487898457  
91411920/xl.meta
```

13. Vérifiez l'opération de lecture à l'aide du script `verify_data_netapp.py`.

```
root@ip-172-31-29-98:~/pymilvus/examples# python3 verify_data_netapp.py  
=== start connecting to Milvus ===  
  
=== Milvus host: localhost ===  
  
Does collection hello_milvus_ntapnew_sc exist in Milvus: True  
{'auto_id': False, 'description': 'hello_milvus_ntapnew_sc', 'fields':  
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,  
'is_primary': True, 'auto_id': False}, {'name': 'random', 'description':  
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',  
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},  
{'name': 'embeddings', 'description': '', 'type': <DataType.  
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':  
False}  
Number of entities in Milvus: hello_milvus_ntapnew_sc : 9000  
  
=== Start Creating index IVF_FLAT ===  
  
=== Start loading ===  
  
=== Start searching based on vector similarity ===  
  
hit: id: 2248, distance: 0.0, entity: {'random': 0.2777646777746381},  
random field: 0.2777646777746381  
hit: id: 4837, distance: 0.07805602252483368, entity: {'random':  
0.6451650959930306}, random field: 0.6451650959930306  
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':  
0.6141351712303128}, random field: 0.6141351712303128  
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},  
random field: 0.7434908973629817  
hit: id: 830, distance: 0.05628090724349022, entity: {'random':  
0.8544487225667627}, random field: 0.8544487225667627  
hit: id: 8562, distance: 0.07971227169036865, entity: {'random':  
0.4464554280115878}, random field: 0.4464554280115878  
search latency = 0.1266s  
  
=== Start querying with `random > 0.5` ===
```

```

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

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

hit: id: 4837, distance: 0.07805602252483368, entity: {'random':
0.6451650959930306}, random field: 0.6451650959930306
hit: id: 7172, distance: 0.07954417169094086, entity: {'random':
0.6141351712303128}, random field: 0.6141351712303128
hit: id: 515, distance: 0.09590047597885132, entity: {'random':
0.8013175797590888}, random field: 0.8013175797590888
hit: id: 2249, distance: 0.0, entity: {'random': 0.7434908973629817},
random field: 0.7434908973629817
hit: id: 830, distance: 0.05628090724349022, entity: {'random':
0.8544487225667627}, random field: 0.8544487225667627
hit: id: 1627, distance: 0.08096684515476227, entity: {'random':
0.9302397069516164}, random field: 0.9302397069516164
search latency = 0.2674s
Does collection hello_milvus_ntapnew_sc2 exist in Milvus: True
{'auto_id': True, 'description': 'hello_milvus_ntapnew_sc2', 'fields':
[{'name': 'pk', 'description': '', 'type': <DataType.INT64: 5>,
'is_primary': True, 'auto_id': True}, {'name': 'random', 'description':
'', 'type': <DataType.DOUBLE: 11>}, {'name': 'var', 'description': '',
'type': <DataType.VARCHAR: 21>, 'params': {'max_length': 65535}},
{'name': 'embeddings', 'description': '', 'type': <DataType.
FLOAT_VECTOR: 101>, 'params': {'dim': 8}}], 'enable_dynamic_field':
False}

```

14. Si le client souhaite accéder (en lecture) aux données NFS testées dans la base de données vectorielle via le protocole S3 pour les workloads d'IA, cela peut être validé à l'aide d'un programme Python simple. Un exemple de ceci pourrait être une recherche de similarité d'images d'une autre application comme mentionné dans l'image qui se trouve au début de cette section.

```

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
hoPctr9aD88c1j0SkIYZ2uPa03v1bqKA0c5feK6F
OBJECTS in the bucket milvusnasvol are :
*****
...
<output content removed to save page space>
...

```



```
bucket/files/insert_log/448789845791611912/448789845791611913/4487898457
91611920/0/448789845791411917/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/1/448789845791411918/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411913/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/101/448789845791411914/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/102/448789845791411915/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/1c48ab6e-
1546-4503-9084-28c629216c33/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/101/448789845791411921/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/102/448789845791411922/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-
c117-4fba-8256-96cb7557cd6c/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/448789845791411912/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411919/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411926/xl.meta
*****
root@ip-172-31-29-98:~/pymilvus/examples#
```

Cette section explique de manière efficace comment les clients peuvent déployer et exploiter une

configuration Milvus autonome dans des conteneurs Docker en utilisant NetApp FSxN pour le stockage de données NetApp ONTAP d'Amazon. Cette configuration permet aux clients d'exploiter la puissance des bases de données vectorielles pour gérer les données de grande envergure et exécuter des requêtes complexes, le tout dans l'environnement évolutif et efficace de conteneurs Docker. En créant une instance Amazon FSxN pour NetApp ONTAP et en faisant correspondre l'instance EC2, les clients peuvent assurer une utilisation optimale des ressources et une gestion optimale des données. La validation réussie des opérations d'écriture et de lecture de données de FSxN dans la base de données vectorielle offre aux clients la garantie d'opérations de données fiables et cohérentes. En outre, la possibilité de lister (lire) des données de workloads d'IA via le protocole S3 permet d'améliorer l'accessibilité des données. Ce processus complet offre donc aux clients une solution robuste et efficace pour la gestion de leurs opérations de données à grande échelle, exploitant les fonctionnalités de FSxN pour NetApp ONTAP d'Amazon.

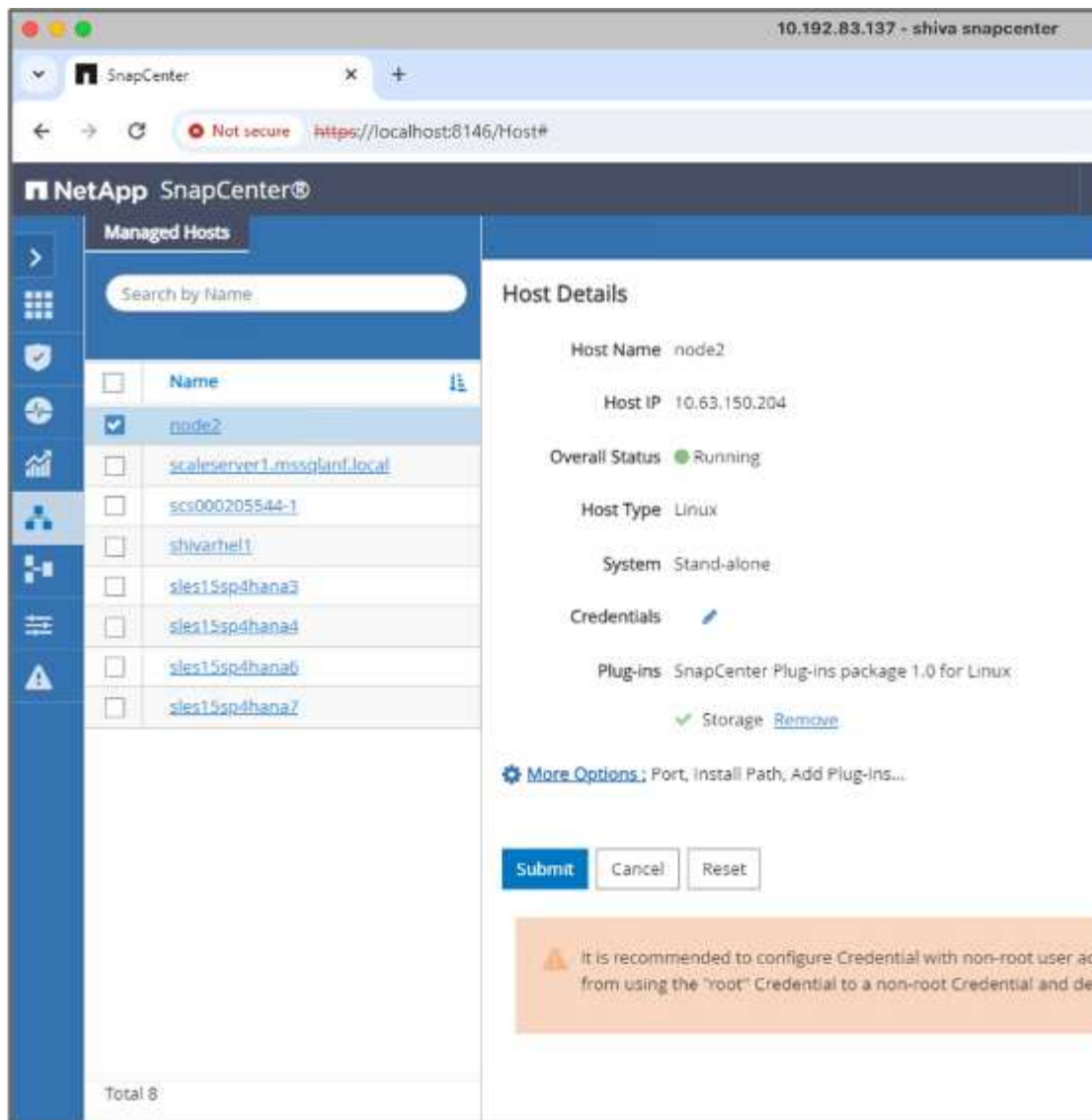
Protection de base de données Vector à l'aide de SnapCenter

Protection de base de données Vector à l'aide de NetApp SnapCenter.

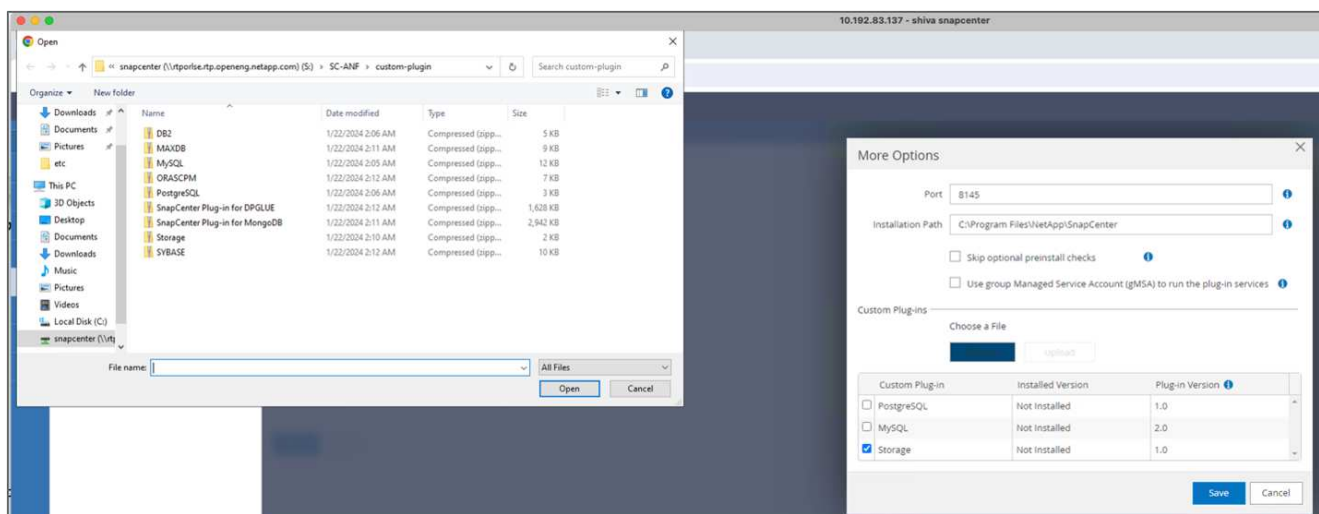
Par exemple, dans le secteur de la production cinématographique, les clients possèdent souvent des données intégrées essentielles, telles que des fichiers audio et vidéo. La perte de ces données, due à des problèmes tels que les pannes de disque dur, peut avoir un impact significatif sur leurs opérations, ce qui risque de mettre en péril des projets à plusieurs millions de dollars. Nous avons rencontré des cas où du contenu inestimable a été perdu, causant des perturbations importantes et des pertes financières. La sécurité et l'intégrité de ces données essentielles sont donc d'une importance capitale dans ce secteur.

Dans cette section, nous allons examiner comment SnapCenter protège les données de la base de données vectorielle et les données Milvus résidant dans ONTAP. Dans cet exemple, nous avons utilisé un compartiment NAS (milvusdbvol1) dérivé d'un volume ONTAP NFS (vol1) pour les données clients, et un volume NFS distinct (vectordbpv) pour les données de configuration du cluster Milvus. veuillez vérifier le "[ici](#)" pour le flux de travail de sauvegarde SnapCenter

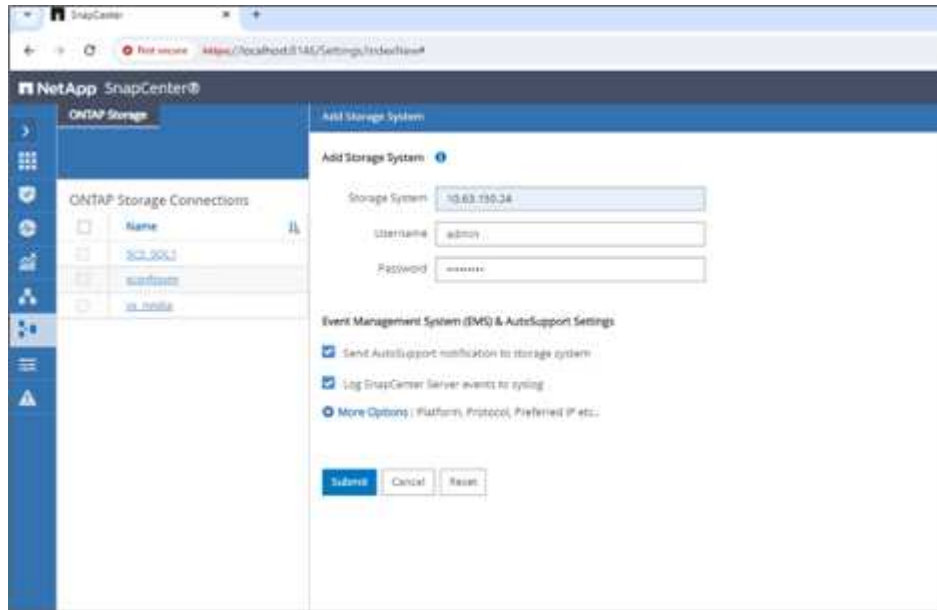
1. Configurez l'hôte qui sera utilisé pour exécuter les commandes SnapCenter.



- Installez et configurez le plug-in de stockage. Dans l'hôte ajouté, sélectionnez « autres options ». Accédez au plug-in de stockage téléchargé et sélectionnez-le dans le "Le site NetApp Automation Store". Installez le plug-in et enregistrez la configuration.



3. Configurer le système et le volume de stockage : ajouter le système de stockage sous « système de stockage » et sélectionner le SVM (Storage Virtual machine). Dans cet exemple, nous avons choisi « vs_nvidia ».



4. Établissez une ressource pour la base de données vectorielle, en incorporant une règle de sauvegarde et un nom de snapshot personnalisé.
- Activer la sauvegarde de groupe de cohérence avec les valeurs par défaut et activer SnapCenter sans cohérence de système de fichiers.
 - Dans la section empreinte du stockage, sélectionnez les volumes associés aux données clients de la base de données vectorielle et aux données du cluster Milvus. Dans notre exemple, il s'agit de "vol1" et de "vectordbpv".
 - Créez une stratégie pour la protection de la base de données vectorielle et protégez la ressource de base de données vectorielle à l'aide de la stratégie.

Modify Storage Storage Resource

1 Name

2 Storage Footprint

3 Resource Settings

4 Summary

Summary

Name milvusdb

Type None

Host scaleserver1.mssqlanf.local

Mount Points

Credential Name adminuser

Storage Footprint

Storage System	Volume	LUN/Qtree
vs_nvidia	vol1	
	vectordbpv	

Custom Resource Parameters None

Previous Finish

5. Insérez les données dans le compartiment NAS S3 à l'aide d'un script Python. Dans notre cas, nous avons modifié le script de sauvegarde fourni par Milvus, à savoir « prepare_data_netapp.py », et exécuté la commande « sync » pour vider les données du système d'exploitation.

```

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:~#

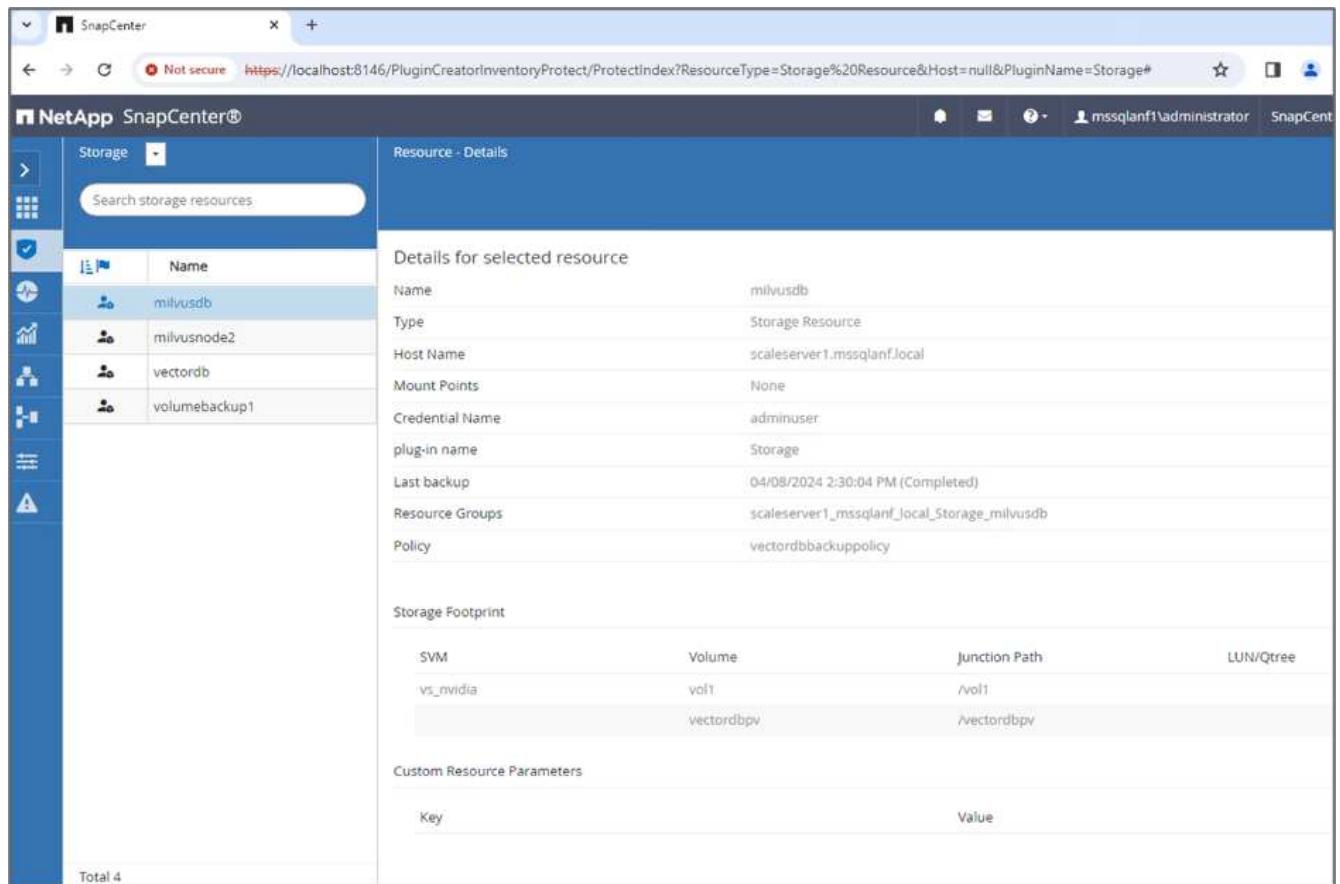
```

6. Vérifiez les données dans le compartiment NAS S3. Dans notre exemple, les fichiers dont l'horodatage est « 2024-04-08 21:22 » ont été créés par le script « prepare_data_netapp.py ».

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

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

7. Lancez une sauvegarde à l'aide du snapshot du groupe de cohérence (CG) à partir de la ressource 'milvusdb'



8. Pour tester la fonctionnalité de sauvegarde, nous avons soit ajouté un nouveau tableau après le processus de sauvegarde, soit supprimé certaines données du NFS (compartiment NAS S3).

Pour ce test, imaginez un scénario dans lequel quelqu'un a créé une nouvelle collection, inutile ou inappropriée après la sauvegarde. Dans ce cas, nous devrions rétablir la base de données vectorielle à son état avant l'ajout de la nouvelle collection. Par exemple, de nouvelles collections telles que « hello_milvus_netapp_sc_testNew » et « hello_milvus_netapp_sc_testnew2 » ont été insérées.


```
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` ===

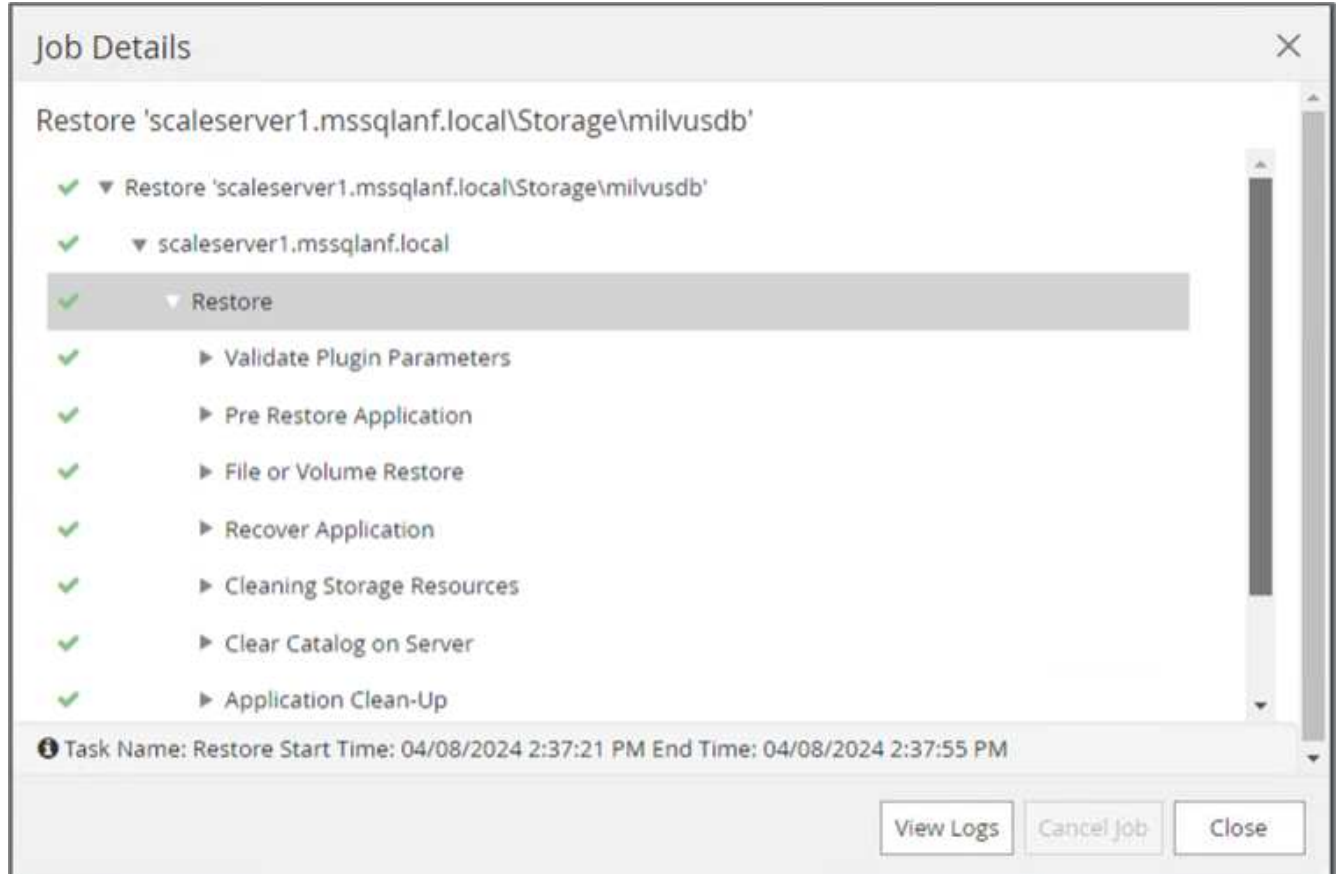
=== Start inserting entities        ===

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. Exécutez une restauration complète du compartiment NAS S3 à partir du snapshot précédent.



10. Utilisez un script Python pour vérifier les données des collections « hello_milvus_netapp_sc_test » et « hello_milvus_netapp_sc_test2 ».

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

=== start connecting to Milvus ===

=== Milvus host: localhost ===

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

=== Start Creating index IVF_FLAT ===

=== Start loading ===

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

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

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

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

```

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

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

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

=== Start Creating index IVF_FLAT ===

=== Start loading ===

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

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

```

```

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

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

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

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

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

```

11. Vérifiez que la collection inutile ou inappropriée n'est plus présente dans la base de données.

```

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:~#

```

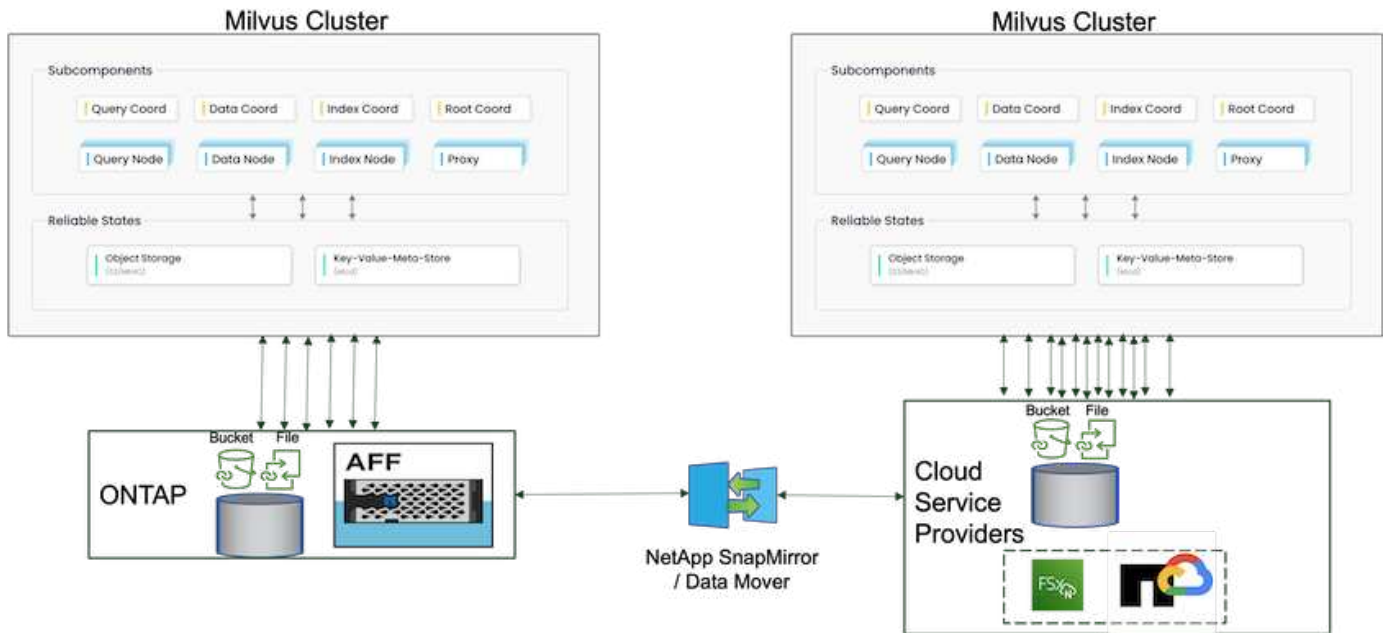
En conclusion, l'utilisation de SnapCenter de NetApp pour protéger les données de base de données Vector et les données Milvus résidant dans ONTAP offre des avantages considérables aux clients, en particulier dans les secteurs où l'intégrité des données est primordiale, tels que la production cinématographique. La capacité de SnapCenter à créer des sauvegardes cohérentes et à restaurer les données complètes garantit que les données stratégiques, telles que les fichiers audio et vidéo intégrés, sont protégées contre les pertes causées par des défaillances de disque dur ou d'autres problèmes. Cela permet non seulement d'éviter les perturbations opérationnelles, mais également d'éviter des pertes financières substantielles.

Dans cette section, nous avons démontré comment configurer SnapCenter pour protéger les données résidant dans ONTAP, notamment la configuration des hôtes, l'installation et la configuration des plug-ins de stockage, et la création d'une ressource pour la base de données Vector avec un nom de snapshot personnalisé. Nous vous avons également présenté comment effectuer une sauvegarde à l'aide du snapshot de groupe de cohérence et vérifier les données dans le compartiment NAS S3.

De plus, nous avons simulé un scénario dans lequel une collection inutile ou inappropriée a été créée après la sauvegarde. Dans de tels cas, la capacité de SnapCenter à effectuer une restauration complète à partir d'un snapshot précédent permet de rétablir l'état de la base de données vectorielle avant l'ajout de la nouvelle collection, préservant ainsi l'intégrité de la base de données. Cette fonctionnalité de restauration des données à un point dans le temps est inestimable pour les clients. Elle leur assure non seulement la sécurité de leurs données, mais aussi la maintenance adéquate. Le produit SnapCenter de NetApp offre ainsi une solution robuste et fiable de protection et de gestion des données.

Reprise sur incident à l'aide de NetApp SnapMirror

Reprise sur incident à l'aide de NetApp SnapMirror



La reprise après sinistre est essentielle pour maintenir l'intégrité et la disponibilité d'une base de données vectorielle, en particulier compte tenu de son rôle dans la gestion des données de grande dimension et l'exécution de recherches complexes de similarité. Une stratégie de reprise d'activité bien planifiée et mise en œuvre permet de s'assurer que les données ne sont pas perdues ou compromises en cas d'incidents imprévus, tels que des pannes matérielles, des catastrophes naturelles ou des cyberattaques. Cette approche est particulièrement importante pour les applications reposant sur des bases de données vectorielles, où toute perte ou corruption de données peut entraîner des interruptions opérationnelles et des pertes financières importantes. En outre, un plan solide de reprise sur incident assure également la continuité de l'activité en minimisant les temps d'arrêt et en permettant la restauration rapide des services. Pour ce faire, nous nous appuyons sur SnapMirror, produit de réplication de données NetApp, sur différents sites géographiques, à des sauvegardes régulières et à des mécanismes de basculement. Par conséquent, la reprise après incident n'est pas seulement une mesure de protection, mais un composant essentiel d'une gestion responsable et efficace de la base de données vectorielle.

NetApp SnapMirror assure la réplication des données d'un contrôleur de stockage NetApp ONTAP vers un autre, principalement utilisé pour la reprise après incident et les solutions hybrides. Dans le contexte d'une base de données vectorielle, cet outil facilite la transition en douceur des données entre les environnements sur site et cloud. Cette transition s'effectue sans conversion de données ni remaniement d'applications, ce qui améliore l'efficacité et la flexibilité de la gestion des données sur plusieurs plateformes.

La solution hybride NetApp dans un scénario de base de données vectorielle peut vous apporter davantage d'avantages :

1. **Évolutivité** : la solution de cloud hybride de NetApp vous permet de faire évoluer vos ressources en fonction de vos besoins. Vous pouvez utiliser les ressources sur site pour des charges de travail régulières et prévisibles ainsi que des ressources cloud comme Amazon FSxN pour NetApp ONTAP et Google Cloud NetApp Volume (GCNV) en cas de pics d'activité ou de charges inattendues.
2. **Maîtrise des coûts** : le modèle de cloud hybride NetApp vous permet d'optimiser les coûts en utilisant des ressources sur site pour les workloads réguliers, et en payant uniquement les ressources cloud que vous utilisez lorsque vous en avez besoin. Ce modèle de paiement basé sur l'utilisation peut être économique avec une offre de service NetApp Instacluster. Pour les principaux fournisseurs de services cloud et sur site, Instacluster fournit un support et un conseil.
3. **La flexibilité** : avec le cloud hybride NetApp, vous pouvez choisir où traiter vos données. Par exemple, vous pouvez choisir d'effectuer des opérations vectorielles complexes sur site où vous disposez de matériel plus

puissant et d'opérations moins intensives dans le cloud.

4. Continuité de l'activité : en cas d'incident, la mise à disposition de vos données dans un cloud hybride NetApp assure la continuité de l'activité. En cas d'impact sur vos ressources sur site, vous pouvez rapidement passer au cloud. Nous pouvons utiliser NetApp SnapMirror pour déplacer les données d'un environnement sur site vers le cloud, et inversement.
5. Innovation : les solutions de cloud hybride de NetApp permettent également d'innover plus rapidement en fournissant un accès à des services et technologies de pointe dans le cloud. Les innovations NetApp dans le cloud, telles qu'Amazon FSxN pour NetApp ONTAP, Azure NetApp Files et Google Cloud NetApp volumes sont des fournisseurs de services cloud qui innovent et préfèrent le NAS.

Validation des performances de la base de données Vector

Validation des performances

La validation des performances joue un rôle critique à la fois dans les bases de données vectorielles et les systèmes de stockage. Elle joue un rôle clé dans la garantie d'un fonctionnement optimal et d'une utilisation efficace des ressources. Les bases de données vectorielles, connues pour le traitement de données de grande dimension et l'exécution de recherches de similarité, doivent maintenir des niveaux de performances élevés pour traiter rapidement et précisément les requêtes complexes. La validation des performances permet d'identifier les goulets d'étranglement, d'ajuster les configurations et de s'assurer que le système peut gérer les charges attendues sans dégradation des services. De même, dans les systèmes de stockage, la validation des performances est essentielle pour garantir le stockage et la récupération efficaces des données, sans problèmes de latence ni goulets d'étranglement susceptibles d'affecter les performances globales du système. Il permet également de prendre des décisions avisées concernant les mises à niveau ou les modifications nécessaires de l'infrastructure de stockage. Par conséquent, la validation des performances est un aspect crucial de la gestion du système et contribue de manière significative au maintien d'un niveau élevé de qualité de service, d'efficacité opérationnelle et de fiabilité globale du système.

Dans cette section, nous allons examiner la validation des performances des bases de données vectorielles, telles que Milvus et pgvecto.RS, en nous concentrant sur leurs caractéristiques de performances de stockage, telles que le profil d'E/S et le contrôleur de stockage NetApp qui prennent en charge les charges de travail RAG et d'inférence dans le cadre du cycle de vie LLM. Nous évaluerons et identifierons les différences de performances éventuelles lorsque ces bases de données sont combinées à la solution de stockage ONTAP. Notre analyse sera basée sur des indicateurs clés de performance, comme le nombre de requêtes traitées par seconde (QPS).

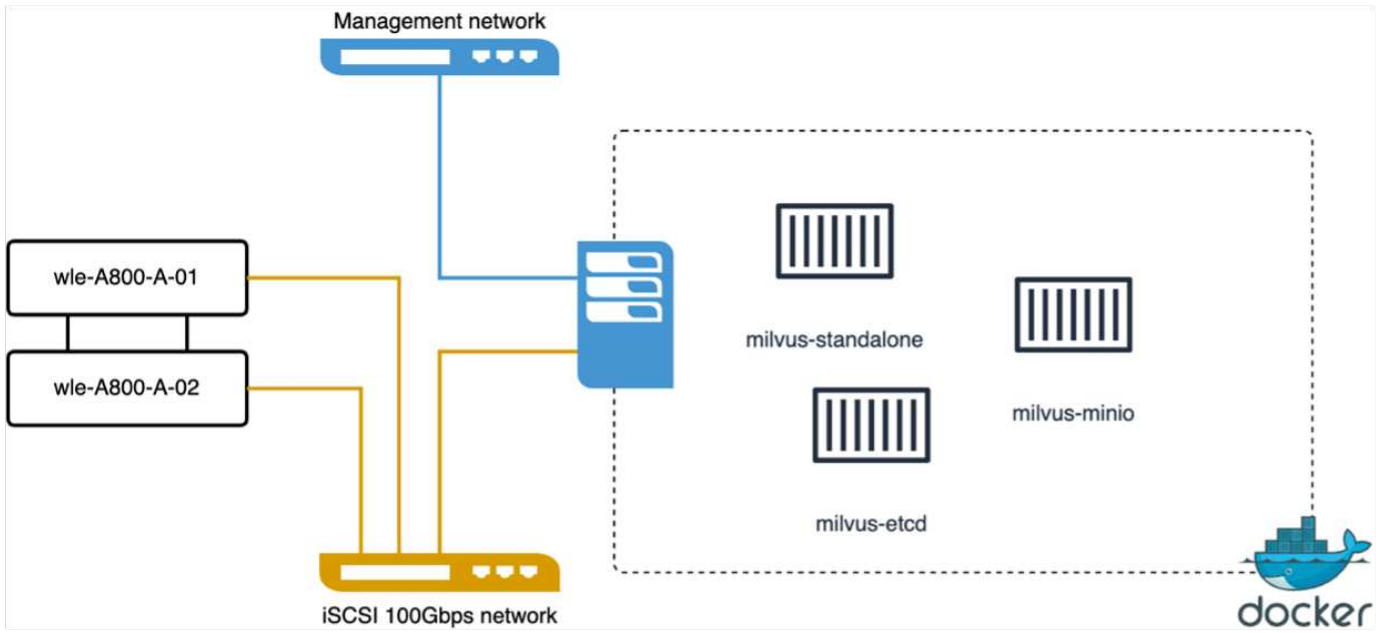
Veuillez vérifier la méthodologie utilisée pour le milvus et la progression ci-dessous.

Détails	Milvus (autonome et cluster)	Postgres(pgvecto.RS)
version	2.3.2	0.2.0
Système de fichiers	XFS sur LUN iSCSI	
Générateur de charges de travail	"VectorDB-Bench" – v0.0.5	
Jeux de données	Jeu de données LAION * 10millions de lits * 768 Dimensions * Taille de Dataset de 300 Go	

VectorDB-Bench avec cluster autonome Milvus

Nous avons effectué la validation de performances suivante sur un cluster autonome milvus avec vectorDB-Bench.

La connectivité réseau et serveur du cluster autonome milvus est ci-dessous.



Dans cette section, nous partageons nos observations et nos résultats des tests de la base de données autonome Milvus.

- . Nous avons sélectionné DiskANN comme type d'index pour ces tests.
- . L'ingestion, l'optimisation et la création d'index pour un dataset d'environ 100 Go ont pris environ 5 heures. Pendant la majeure partie de cette durée, le serveur Milvus, équipé de 20 cœurs (ce qui équivaut à 40 vcpu lorsque Hyper-Threading est activé), fonctionnait à sa capacité de processeur maximale de 100 %. nous avons constaté que DiskANN est particulièrement important pour les jeux de données volumineux qui dépassent la taille de la mémoire système.
- . Dans la phase de requête, nous avons observé un taux de requêtes par seconde (QPS) de 10.93 avec un rappel de 0.9987. La latence du 99e centile pour les requêtes a été mesurée à 708.2 millisecondes.

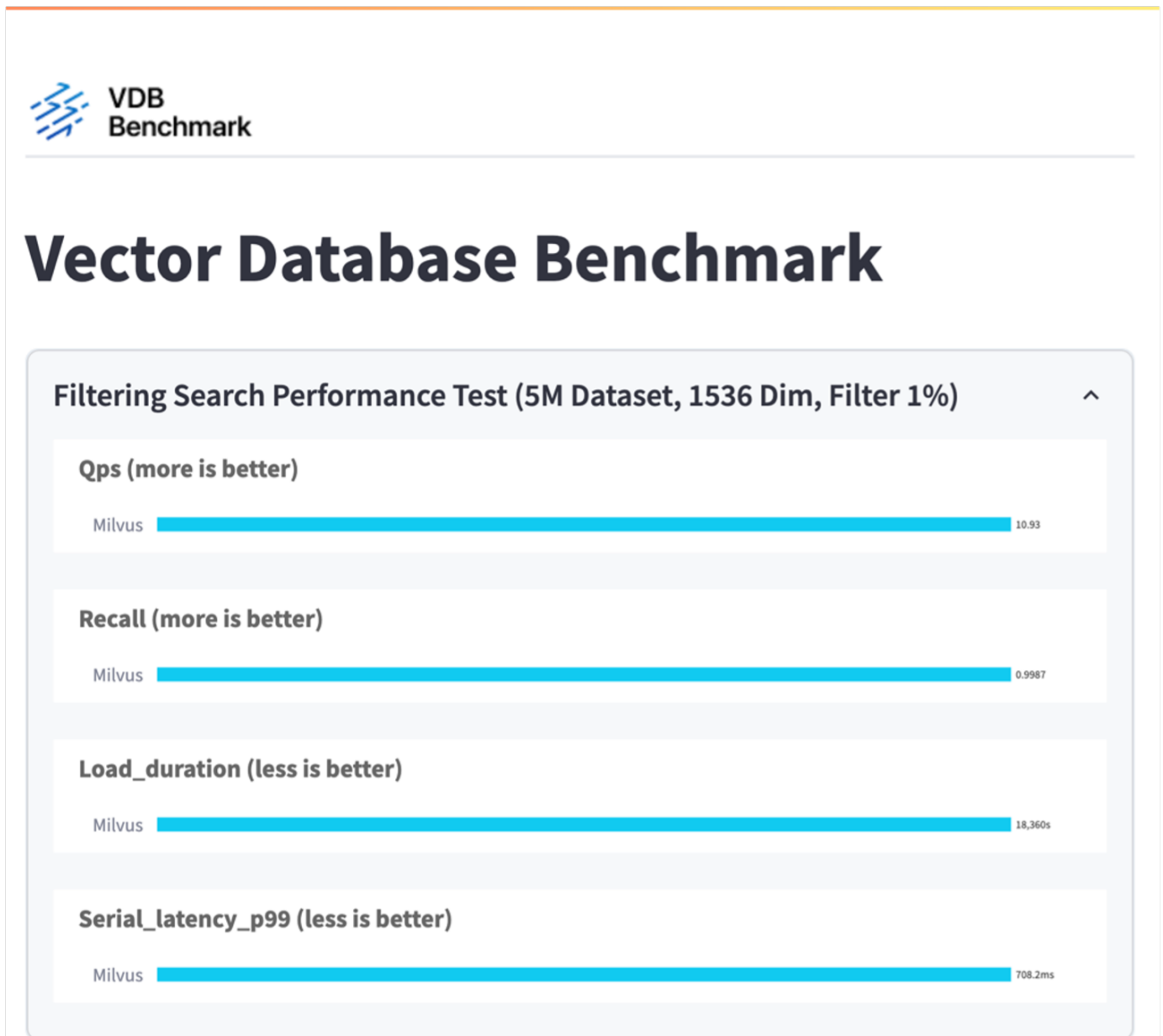
Du point de vue du stockage, la base de données a émis environ 1,000 000 opérations/s durant les phases d'ingestion, d'optimisation après insertion et de création d'index. Dans la phase de requête, il a demandé 32,000 opérations/s.

La section suivante présente les mesures de performances du stockage.

Phase du workload	Métrique	Valeur
Ingestion des données et Optimisation post-insertion	D'IOPS	< 1,000
	Latence	< 400 utilisateurs
	Charge de travail	Proportion de lectures/écritures, principalement des écritures
Requête	Taille des E/S.	64 KO
	D'IOPS	Crête à 32,000
	Latence	< 400 utilisateurs
	Charge de travail	100 % de lectures mises en cache

Phase du workload	Métrique	Valeur
	Taille des E/S.	Principalement 8 Ko

Le résultat vectorDB-Bench est inférieur à.

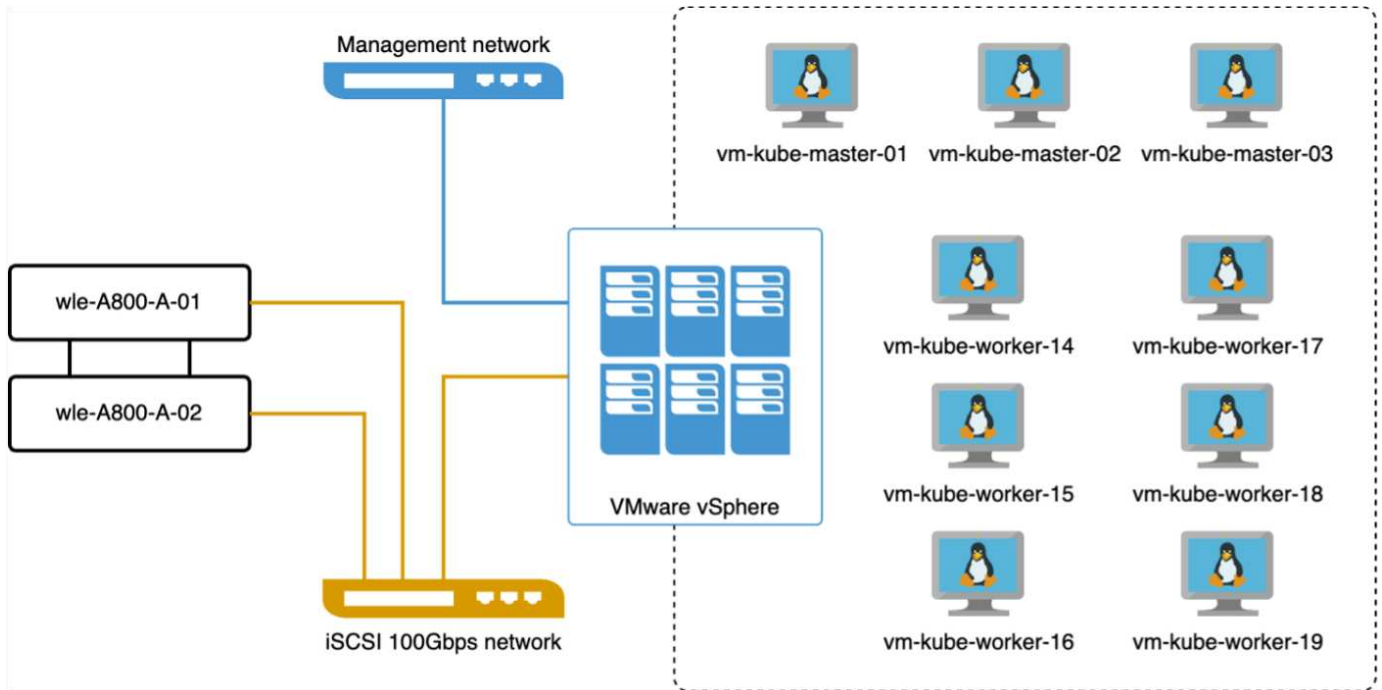


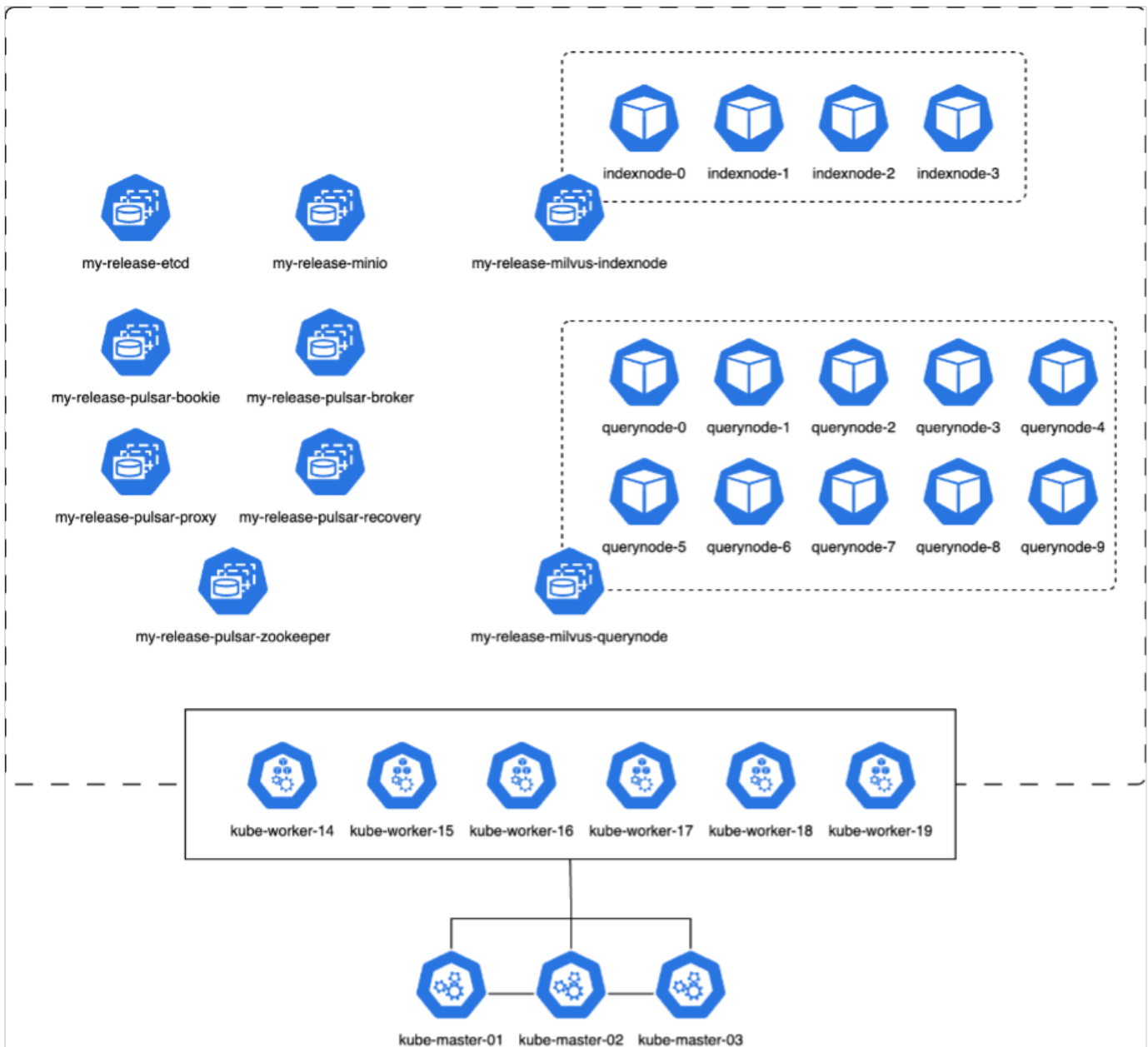
Depuis la validation des performances de l'instance Milvus autonome, il est évident que la configuration actuelle est insuffisante pour prendre en charge un jeu de données de 5 millions de vecteurs avec une dimension de 1536. nous avons déterminé que le stockage possède les ressources adéquates et ne constitue pas un goulet d'étranglement dans le système.

VectorDB-Bench avec cluster Milvus

Cette section traite du déploiement d'un cluster Milvus dans un environnement Kubernetes. Cette configuration Kubernetes a été construite sur un déploiement VMware vSphere, qui héberge les nœuds maîtres et workers Kubernetes.

Les sections suivantes présentent un détail des déploiements VMware vSphere et Kubernetes.





Dans cette section, nous présentons nos observations et nos résultats des tests de la base de données Milvus.

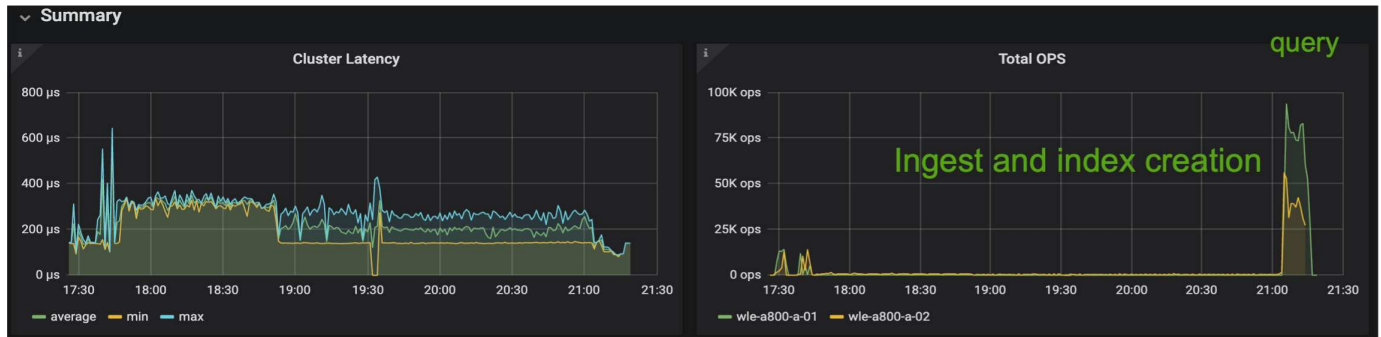
* Le type d'index utilisé était DiskANN.

* Le tableau ci-dessous fournit une comparaison entre les déploiements autonomes et les déploiements en grappe lorsqu'ils travaillent avec 5 millions de vecteurs à une dimension de 1536. Nous avons observé que le temps nécessaire pour l'ingestion et l'optimisation après insertion des données était plus faible dans le déploiement du cluster. Au cours du déploiement du cluster, la latence du 99e centile pour les requêtes a été divisée par six par rapport à une configuration autonome.

* Bien que le taux de requêtes par seconde (QPS) ait été plus élevé dans le déploiement du cluster, il n'était pas au niveau souhaité.

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%

Les images ci-dessous présentent diverses mesures de stockage, notamment la latence du cluster de stockage et les IOPS totales (opérations d'entrée/sortie par seconde).



La section suivante présente les principaux metrics de performance du stockage.

Phase du workload	Métrique	Valeur
Ingestion des données et Optimisation post-insertion	D'IOPS	< 1,000
	Latence	< 400 utilisateurs
	Charge de travail	Proportion de lectures/écritures, principalement des écritures
Requête	Taille des E/S.	64 KO
	D'IOPS	Crête à 147,000
	Latence	< 400 utilisateurs
	Charge de travail	100 % de lectures mises en cache
	Taille des E/S.	Principalement 8 Ko

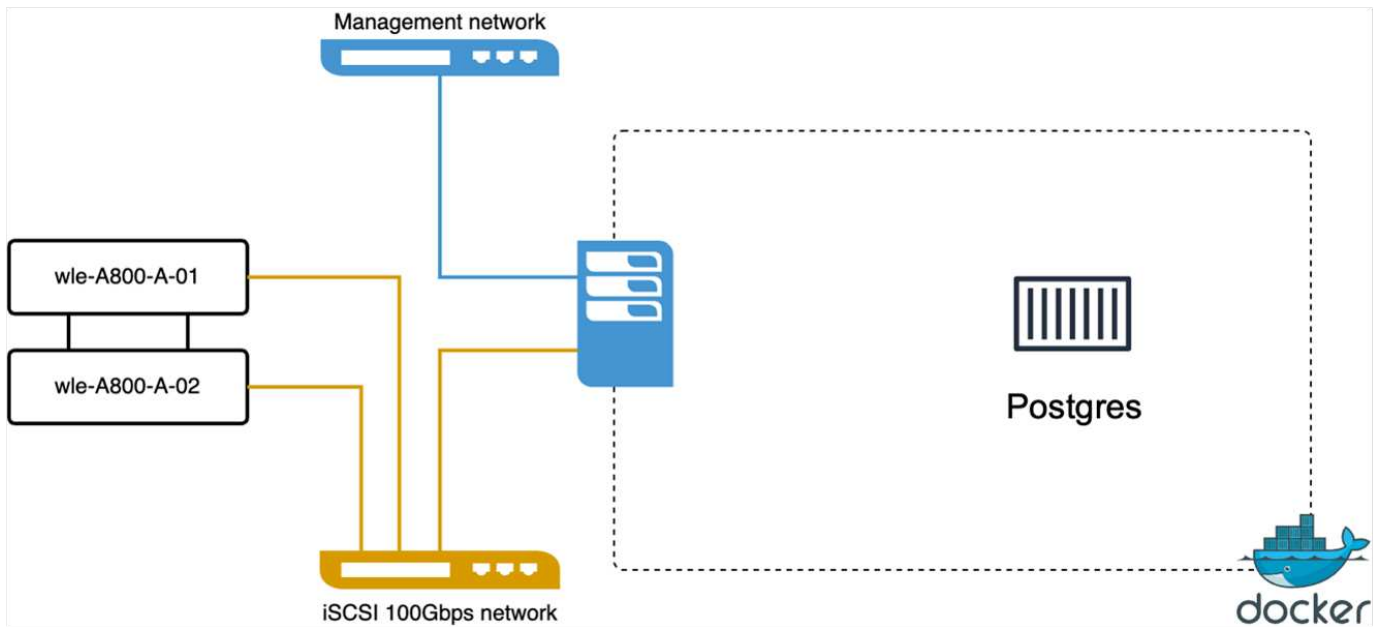
Sur la base de la validation des performances du cluster Milvus autonome et du cluster Milvus, nous présentons les détails du profil d'E/S du stockage.

* Nous avons observé que le profil d'E/S reste cohérent à la fois dans les déploiements autonomes et en cluster.

* La différence observée dans le pic d'IOPS peut être attribuée au plus grand nombre de clients dans le déploiement de cluster.

VectorDB-Bench avec Postgres (pgvecto.RS)

Nous avons effectué les actions suivantes sur PostgreSQL(pgvecto.RS) à l'aide de VectorDB-Bench : Les détails concernant la connectivité réseau et serveur de PostgreSQL (plus précisément, pgvecto.RS) sont les suivants :



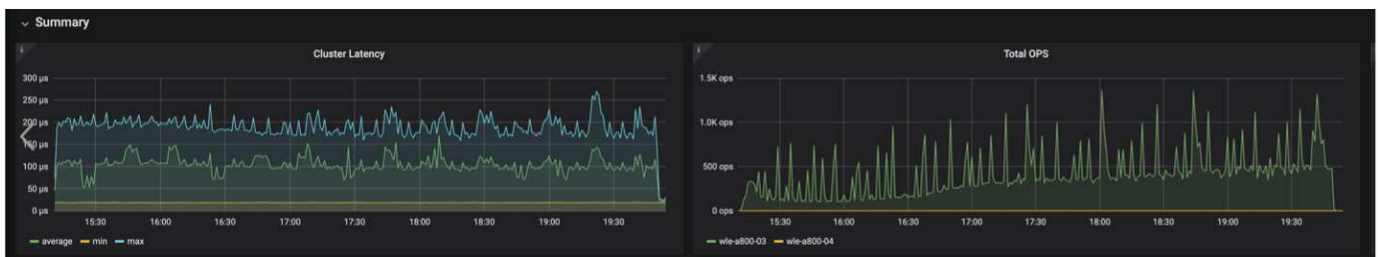
Dans cette section, nous partageons nos observations et nos résultats des tests de la base de données PostgreSQL, en particulier à l'aide de pgvecto.RS.

* Nous avons choisi HNSW comme type d'index pour ces tests parce qu'au moment des tests, DiskANN n'était pas disponible pour pgvecto.RS.

* Pendant la phase d'ingestion des données, nous avons chargé le jeu de données de Cohere, qui se compose de 10 millions de vecteurs à une dimension de 768. Ce processus a pris environ 4.5 heures.

* Dans la phase de requête, nous avons observé un taux de requêtes par seconde (QPS) de 1,068 avec un rappel de 0.6344. La latence du 99e centile pour les requêtes a été mesurée à 20 millisecondes. Pendant la majeure partie de l'exécution, le CPU client fonctionnait à 100 % de sa capacité.

Les images ci-dessous offrent une vue d'ensemble des différentes mesures de stockage, y compris les IOPS totales de latence du cluster de stockage (opérations d'entrée/sortie par seconde).



The following section presents the key storage performance metrics.
 image:pgvecto_storage_perf_metrics.png["Erreur : image graphique manquante"]

Comparaison des performances entre milvus et postgres sur le banc de base de données vectoriel

Vector Database Benchmark

Note that all testing was completed in July 2023, except for the times already noted.



Sur la base de notre validation des performances de Milvus et PostgreSQL à l'aide de VectorDBBench, nous avons observé ce qui suit :

- Type d'index : HNSW
- Dataset : cohérent avec 10 millions de vecteurs à 768 dimensions

Nous avons constaté que pgvecto.RS a atteint un taux de requêtes par seconde (QPS) de 1,068 avec un rappel de 0.6344, tandis que Milvus a atteint un taux QPS de 106 avec un rappel de 0.9842.

Si la haute précision de vos requêtes est une priorité, Milvus surpasse pgvecto.RS car il récupère une proportion plus élevée d'éléments pertinents par requête. Toutefois, si le nombre de requêtes par seconde est un facteur plus important, pgvecto.RS dépasse Milvus. Il est important de noter, cependant, que la qualité des données récupérées via pgvecto.RS est plus faible, avec environ 37% des résultats de recherche étant des éléments non pertinents.

Observation basée sur nos validations de performances :

Sur la base de nos validations de performances, nous avons fait les observations suivantes :

Chez Milvus, le profil d'E/S ressemble beaucoup à une charge de travail OLTP, comme c'est le cas avec Oracle SLOB. Le banc d'essai se compose de trois phases : ingestion des données, post-optimisation et requête. Les étapes initiales sont principalement caractérisées par des opérations d'écriture de 64 Ko, alors que la phase de requête implique principalement des lectures de 8 Ko. Nous pensons que ONTAP devrait gérer la charge d'E/S Milvus avec compétence.

Le profil d'E/S PostgreSQL ne présente pas de charge de travail de stockage complexe. Étant donné que l'implémentation in-memory est en cours, nous n'avons pas observé d'E/S de disque pendant la phase de requête.

DiskANN émerge comme une technologie cruciale pour la différenciation du stockage. Il permet une mise à l'échelle efficace de la recherche de base de données vectorielle au-delà de la limite de la mémoire système. Toutefois, il est peu probable qu'il se démarque des performances de stockage grâce à des indices de base de données vectoriels en mémoire tels que HNSW.

Il est également important de noter que le stockage ne joue pas un rôle critique pendant la phase de requête lorsque le type d'index est HSNW, qui est la phase de fonctionnement la plus importante pour les bases de données vectorielles prenant en charge les applications RAG. Cela signifie que la performance du stockage n'a pas un impact significatif sur les performances globales de ces applications.

Base de données Vector avec Instaclustr à l'aide de PostgreSQL: Pgvector

Base de données Vector avec Instaclustr à l'aide de PostgreSQL: Pgvector

Dans cette section, nous allons examiner les spécificités de l'intégration du produit Instaclustr à PostgreSQL sur la fonctionnalité pgvector. Nous avons un exemple de "Comment améliorer la précision et les performances de votre LLM avec PGVector et PostgreSQL®: Introduction aux embeddings et le rôle de PGVector". Veuillez vérifier le "[Blog](#)" pour en savoir plus.

Cas d'utilisation de la base de données Vector

Cas d'utilisation de la base de données Vector

Cette section aborde deux cas d'utilisation, tels que la récupération de la génération augmentée avec les modèles en langage large et le chatbot INFORMATIQUE NetApp.

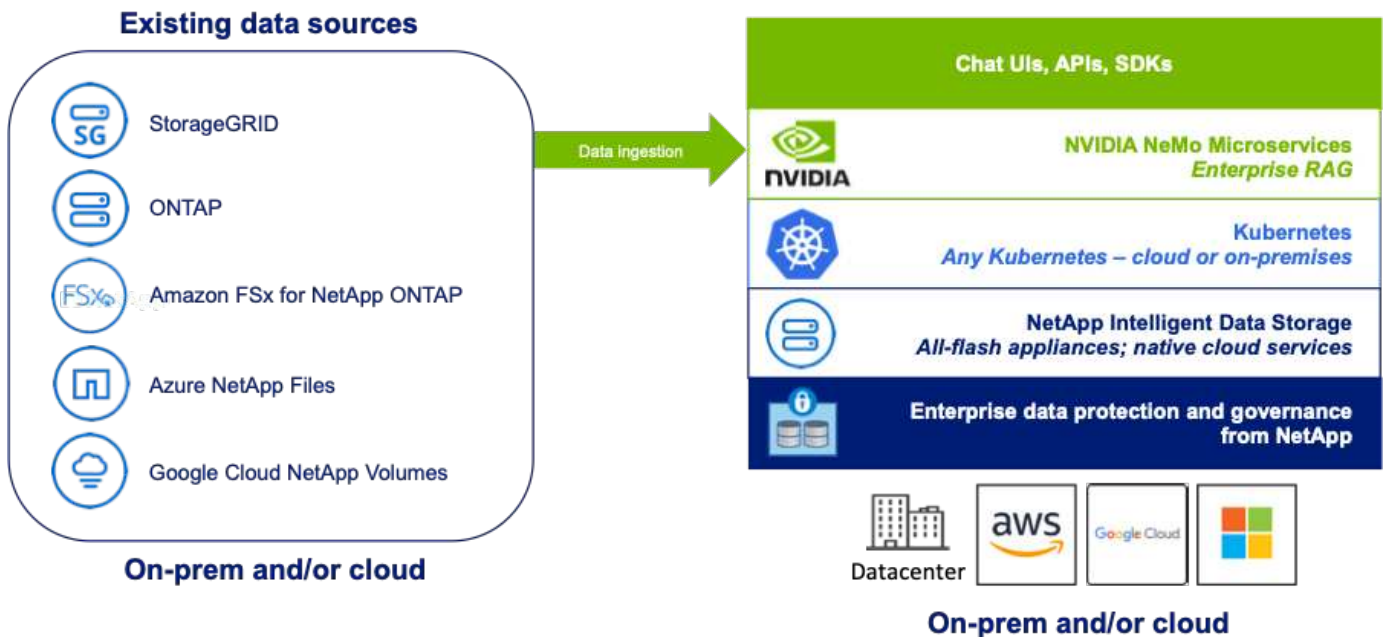
Récupération de la génération augmentée (RAG) avec les modèles de langage large (LLMS)

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

L'opérateur NVIDIA Enterprise RAG LLM est un outil utile pour implémenter RAG dans l'entreprise. Cet opérateur peut être utilisé pour déployer un pipeline RAG complet. Le pipeline RAG peut être personnalisé pour utiliser Milvus ou pgvecto comme base de données vectorielle pour le stockage des codes de base de connaissances. Pour plus de détails, reportez-vous à la documentation.

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.

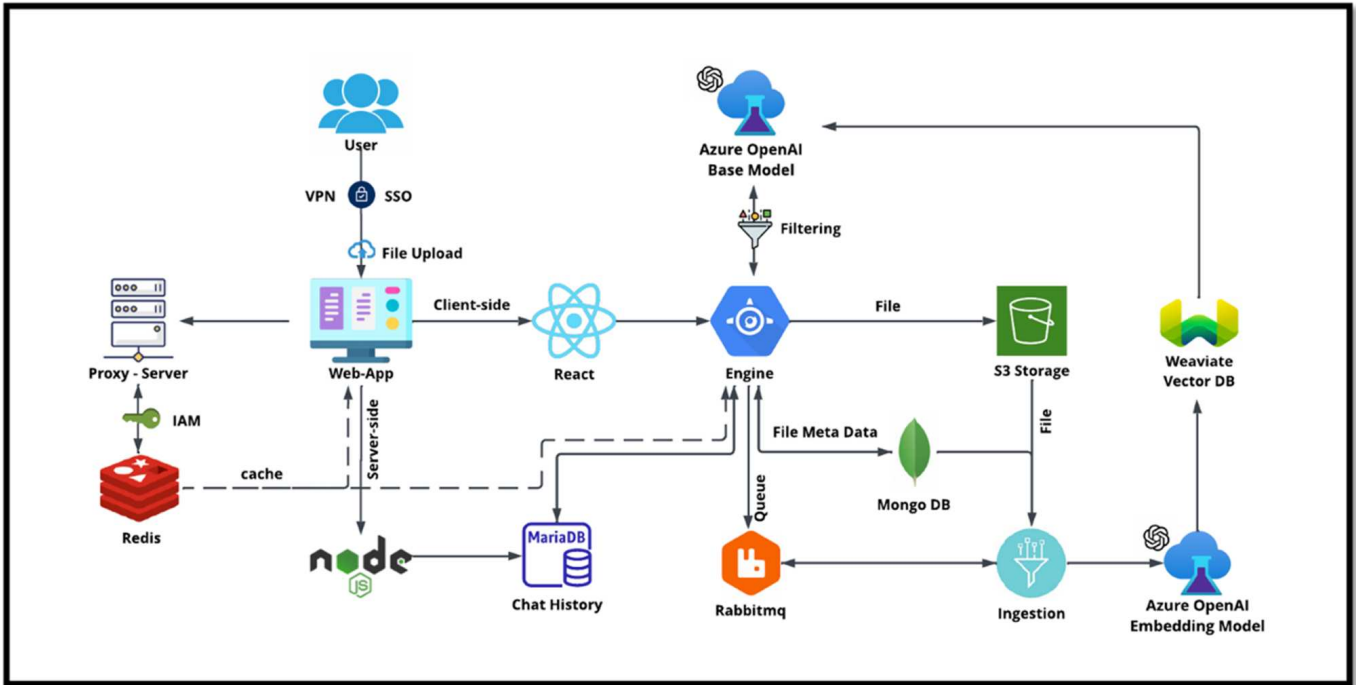
Figure 1) RAG d'entreprise optimisé par NVIDIA Nemo microservices et NetApp



Cas d'utilisation du chatbot INFORMATIQUE NetApp

Le chatbot de NetApp joue un rôle de plus dans l'utilisation en temps réel de la base de données vectorielle.

Dans ce cas, le sandbox NetApp Private OpenAI fournit une plateforme efficace, sécurisée et efficace pour la gestion des requêtes des utilisateurs internes de NetApp. Grâce à l'intégration de protocoles de sécurité stricts, de systèmes de gestion efficaces des données et de fonctionnalités avancées de traitement d'IA, il garantit des réponses précises et de haute qualité aux utilisateurs en fonction de leur rôle et de leurs responsabilités dans l'entreprise via l'authentification SSO. Cette architecture met en évidence le potentiel de fusion de technologies avancées pour créer des systèmes intelligents et centrés sur l'utilisateur.



Le cas d'utilisation peut être divisé en quatre sections principales.

Authentification et vérification de l'utilisateur :

- Les requêtes utilisateur passent d'abord par le processus d'authentification unique (SSO) NetApp pour confirmer l'identité de l'utilisateur.
- Une fois l'authentification réussie, le système vérifie la connexion VPN pour garantir une transmission sécurisée des données.

Transmission et traitement des données :

- Une fois le VPN validé, les données sont envoyées à MariaDB via les applications Web NetAIChat ou NetAIICR. MariaDB est un système de base de données rapide et efficace qui permet de gérer et de stocker les données des utilisateurs.
- MariaDB envoie ensuite les informations à l'instance NetApp Azure, qui connecte les données utilisateur à la machine de traitement ai.

Interaction avec OpenAI et filtrage de contenu :

- L'instance Azure envoie les questions de l'utilisateur à un système de filtrage de contenu. Ce système nettoie la requête et la prépare au traitement.
- L'entrée nettoyée est ensuite envoyée au modèle de base d'Azure OpenAI, qui génère une réponse basée sur l'entrée.

Génération et modération de la réponse :

- La réponse du modèle de base est d'abord vérifiée afin de s'assurer qu'elle est exacte et conforme aux normes de contenu.
- Après avoir passé le contrôle, la réponse est renvoyée à l'utilisateur. Ce processus garantit que l'utilisateur reçoit une réponse claire, précise et appropriée à sa requête.

Conclusion

Conclusion

En conclusion, ce document offre une vue d'ensemble complète du déploiement et de la gestion de bases de données vectorielles, telles que Milvus et pgvector, sur les solutions de stockage NetApp. Nous avons abordé les conseils d'infrastructure pour l'exploitation du stockage objet NetApp ONTAP et StorageGRID et validé la base de données Milvus dans AWS FSX pour NetApp ONTAP via un magasin de fichiers et d'objets.

Nous avons exploré la dualité fichier/objet de NetApp qui démontre son utilité non seulement pour les données dans des bases de données vectorielles, mais aussi pour d'autres applications. Nous avons également souligné comment SnapCenter, le produit de gestion d'entreprise de NetApp, propose des fonctionnalités de sauvegarde, de restauration et de clonage pour les données de bases de données vectorielles, assurant ainsi l'intégrité et la disponibilité des données.

Il examine également comment la solution de cloud hybride NetApp assure la réplication et la protection des données dans les environnements sur site et cloud, assurant ainsi une expérience de gestion des données transparente et sécurisée. Nous avons fourni des informations sur la validation des performances de bases de données vectorielles telles que Milvus et pgvector sur NetApp ONTAP, et nous avons fourni des informations précieuses sur leur efficacité et leur évolutivité.

Enfin, nous avons abordé deux cas d'utilisation d'IA générative : RAG avec LLM et le ChatAI interne de NetApp. Ces exemples pratiques soulignent les applications et les avantages réels des concepts et des pratiques décrits dans ce document. Ce document sert de guide complet à tous ceux qui souhaitent tirer parti des puissantes solutions de stockage de NetApp pour la gestion des bases de données vectorielles.

Remerciements

L'auteur remercie chaleureusement les contributeurs ci-dessous, d'autres personnes qui ont fourni leurs commentaires et commentaires pour rendre ce document précieux aux clients de NetApp et aux champs de NetApp.

1. Satish Thyagarajan, Ingénieur marketing et technique, ONTAP ai & Analytics, NetApp
2. Mike Oglesby, Ingénieur marketing et technique, NetApp
3. AJ Mahajan, Directeur principal, NetApp
4. Joe Scott, responsable de l'ingénierie des performances des charges de travail, NetApp
5. Puneet Dhawan, Directeur senior, solution FSX de gestion de produits, NetApp
6. Yuval Kalderon, responsable produits senior, équipe produit FSX, NetApp

Où trouver des informations complémentaires

Pour en savoir plus sur les informations données dans ce livre blanc, consultez ces documents et/ou sites web :

- Documentation Milvus - <https://milvus.io/docs/overview.md>
- Documentation autonome Milvus - https://milvus.io/docs/v2.0.x/install_standalone-docker.md
- Documentation produit NetApp
<https://www.netapp.com/support-and-training/documentation/>
- instaclustr - "documentation instalclustr"

Historique des versions

Version	Date	Historique des versions du document
Version 1.0	Avril 2024	Version initiale

Annexe A : valeurs.yaml

Annexe A : valeurs.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

# LoadBalancerSourceRange 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 zookeeper 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: apache/pulsar/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: "milvusdbvoll"
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:~#

```

Annexe B : prepare_data_netapp_new.py

Annexe 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:~#

```

Annexe C : verify_data_netapp.py

Annexe 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:~#
```

Annexe D : docker-compose.yml

Annexe 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
```


Informations sur le copyright

Copyright © 2024 NetApp, Inc. Tous droits réservés. Imprimé aux États-Unis. Aucune partie de ce document protégé par copyright ne peut être reproduite sous quelque forme que ce soit ou selon quelque méthode que ce soit (graphique, électronique ou mécanique, notamment par photocopie, enregistrement ou stockage dans un système de récupération électronique) sans l'autorisation écrite préalable du détenteur du droit de copyright.

Les logiciels dérivés des éléments NetApp protégés par copyright sont soumis à la licence et à l'avis de non-responsabilité suivants :

CE LOGICIEL EST FOURNI PAR NETAPP « EN L'ÉTAT » ET SANS GARANTIES EXPRESSES OU TACITES, Y COMPRIS LES GARANTIES TACITES DE QUALITÉ MARCHANDE ET D'ADÉQUATION À UN USAGE PARTICULIER, QUI SONT EXCLUES PAR LES PRÉSENTES. EN AUCUN CAS NETAPP NE SERA TENU POUR RESPONSABLE DE DOMMAGES DIRECTS, INDIRECTS, ACCESSOIRES, PARTICULIERS OU EXEMPLAIRES (Y COMPRIS L'ACHAT DE BIENS ET DE SERVICES DE SUBSTITUTION, LA PERTE DE JOUISSANCE, DE DONNÉES OU DE PROFITS, OU L'INTERRUPTION D'ACTIVITÉ), QUELLES QU'EN SOIENT LA CAUSE ET LA DOCTRINE DE RESPONSABILITÉ, QU'IL S'AGISSE DE RESPONSABILITÉ CONTRACTUELLE, STRICTE OU DÉLICTEUELLE (Y COMPRIS LA NÉGLIGENCE OU AUTRE) DÉCOULANT DE L'UTILISATION DE CE LOGICIEL, MÊME SI LA SOCIÉTÉ A ÉTÉ INFORMÉE DE LA POSSIBILITÉ DE TELS DOMMAGES.

NetApp se réserve le droit de modifier les produits décrits dans le présent document à tout moment et sans préavis. NetApp décline toute responsabilité découlant de l'utilisation des produits décrits dans le présent document, sauf accord explicite écrit de NetApp. L'utilisation ou l'achat de ce produit ne concède pas de licence dans le cadre de droits de brevet, de droits de marque commerciale ou de tout autre droit de propriété intellectuelle de NetApp.

Le produit décrit dans ce manuel peut être protégé par un ou plusieurs brevets américains, étrangers ou par une demande en attente.

LÉGENDE DE RESTRICTION DES DROITS : L'utilisation, la duplication ou la divulgation par le gouvernement sont sujettes aux restrictions énoncées dans le sous-paragraphe (b)(3) de la clause Rights in Technical Data-Noncommercial Items du DFARS 252.227-7013 (février 2014) et du FAR 52.227-19 (décembre 2007).

Les données contenues dans les présentes se rapportent à un produit et/ou service commercial (tel que défini par la clause FAR 2.101). Il s'agit de données propriétaires de NetApp, Inc. Toutes les données techniques et tous les logiciels fournis par NetApp en vertu du présent Accord sont à caractère commercial et ont été exclusivement développés à l'aide de fonds privés. Le gouvernement des États-Unis dispose d'une licence limitée irrévocable, non exclusive, non cessible, non transférable et mondiale. Cette licence lui permet d'utiliser uniquement les données relatives au contrat du gouvernement des États-Unis d'après lequel les données lui ont été fournies ou celles qui sont nécessaires à son exécution. Sauf dispositions contraires énoncées dans les présentes, l'utilisation, la divulgation, la reproduction, la modification, l'exécution, l'affichage des données sont interdits sans avoir obtenu le consentement écrit préalable de NetApp, Inc. Les droits de licences du Département de la Défense du gouvernement des États-Unis se limitent aux droits identifiés par la clause 252.227-7015(b) du DFARS (février 2014).

Informations sur les marques commerciales

NETAPP, le logo NETAPP et les marques citées sur le site <http://www.netapp.com/TM> sont des marques déposées ou des marques commerciales de NetApp, Inc. Les autres noms de marques et de produits sont des marques commerciales de leurs propriétaires respectifs.