Solution de base de données vectorielle avec NetApp

NetApp artificial intelligence solutions

NetApp
February 12, 2026

Sommaire

Solution de base de données vectorielle avec NetApp

Solution de base de données vectorielle avec NetApp

Karthikeyan Nagalingam et Rodrigo Nascimento, NetApp

Ce document fournit une exploration approfondie du déploiement et de la gestion des bases de données vectorielles, telles que Milvus et pgvecto, une extension PostgreSQL open source, en utilisant les solutions de stockage de NetApp. Il détaille les directives d'infrastructure pour l'utilisation du stockage d'objets NetApp ONTAP et StorageGRID et valide l'application de la base de données Milvus dans AWS FSx ONTAP. Le document explique la dualité fichier-objet de NetApp et son utilité pour les bases de données vectorielles et les applications qui prennent en charge les intégrations vectorielles. Il met l'accent sur les capacités de SnapCenter, le produit de gestion d'entreprise de NetApp, en offrant des fonctionnalités de sauvegarde et de restauration pour les bases de données vectorielles, garantissant l'intégrité et la disponibilité des données. Le document approfondit davantage la solution de cloud hybride de NetApp, en discutant de son rôle dans la réplication et la protection des données dans les environnements sur site et dans le cloud. Il comprend des informations sur la validation des performances des bases de données vectorielles sur NetApp ONTAP et se termine par deux cas d'utilisation pratiques sur l'IA générative : RAG avec LLM et ChatAI interne de NetApp. Ce document sert de guide complet pour exploiter les solutions de stockage de NetApp pour la gestion des bases de données vectorielles.

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

Introduction

Cette section fournit une introduction à la solution de base de données vectorielle pour NetApp.

Introduction

Les bases de données vectorielles répondent efficacement aux défis conçus pour gérer les complexités de la recherche sémantique dans les grands modèles de langage (LLM) et l'intelligence artificielle générative (IA). Contrairement aux systèmes de gestion de données traditionnels, les bases de données vectorielles sont capables de traiter et de rechercher différents types de données, notamment 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 (SGBDR) sont bien documentées, en particulier leurs difficultés avec les représentations de données à haute dimension et les données non structurées courantes dans les applications d'IA. Les SGBDR nécessitent souvent un processus long et sujet aux erreurs d'aplatissement des données dans des structures plus faciles à gérer, ce qui entraîne des retards et des inefficacités dans les recherches. Les bases de données vectorielles sont toutefois conçues pour contourner ces problèmes, en offrant une solution plus efficace et plus précise pour la gestion et la recherche de données complexes et de grande dimension, facilitant ainsi l'avancement des applications d'IA.

Ce document sert de guide complet pour les clients qui utilisent actuellement ou prévoient d'utiliser des bases de données vectorielles, détaillant les meilleures pratiques d'utilisation des bases de données vectorielles sur des plates-formes telles que NetApp ONTAP, NetApp StorageGRID, Amazon FSx ONTAP pour NetApp ONTAP et SnapCenter. Le contenu fourni ici couvre une gamme de sujets :

- Directives d'infrastructure pour les bases de données vectorielles, comme Milvus, fournies par le stockage NetApp via NetApp ONTAP et le stockage d'objets StorageGRID .
- Validation de la base de données Milvus dans AWS FSx ONTAP via un magasin de fichiers et d'objets.
- Plonge dans la dualité fichier-objet de NetApp, démontrant son utilité pour les données dans les bases de données vectorielles ainsi que pour d'autres applications.
- Comment le produit de gestion de la protection des données de NetApp, SnapCenter, offre des fonctionnalités de sauvegarde et de restauration pour les données de bases de données vectorielles.
- Comment le cloud hybride de NetApp offre la réplication et la protection des données dans les environnements sur site et cloud.
- Fournit des informations sur la validation des performances des bases de données vectorielles telles que Milvus et pgvector sur NetApp ONTAP.
- Deux cas d'utilisation spécifiques : Retrieval Augmented Generation (RAG) avec Large Language Models (LLM) et ChatAI de l'équipe informatique de NetApp , offrant ainsi des exemples pratiques des concepts et pratiques décrits.

Présentation de la solution

Cette section fournit un aperçu de la solution de base de données vectorielle NetApp .

Présentation de la solution

Cette solution présente les avantages et les capacités distinctifs que NetApp apporte pour relever les défis auxquels sont confrontés les clients de bases de données vectorielles. En tirant parti de NetApp ONTAP, StorageGRID, des solutions cloud de NetApp et de SnapCenter, les clients peuvent ajouter une valeur significative à leurs opérations commerciales. Ces outils non seulement répondent aux problèmes existants, mais améliorent également l'efficacité et la productivité, contribuant ainsi à la croissance globale de l'entreprise.

Pourquoi NetApp?

- Les offres de NetApp, telles que ONTAP et StorageGRID, permettent la séparation du stockage et du calcul, permettant une utilisation optimale des ressources en fonction d'exigences spécifiques. Cette flexibilité permet aux clients de faire évoluer leur stockage de manière indépendante à l'aide des solutions de stockage NetApp .
- En exploitant les contrôleurs de stockage de NetApp, les clients peuvent efficacement fournir des données à leur base de données vectorielle à l'aide des protocoles NFS et S3. Ces protocoles facilitent le stockage des données client et gèrent l'index de la base de données vectorielle, éliminant ainsi le besoin de plusieurs copies de données accessibles via des méthodes de fichiers et d'objets.
- NetApp ONTAP fournit une prise en charge native du stockage NAS et d'objets auprès des principaux fournisseurs de services cloud tels qu'AWS, Azure et Google Cloud. Cette large compatibilité garantit une intégration transparente, permettant la mobilité des données client, l'accessibilité globale, la reprise après sinistre, l'évolutivité dynamique et des performances élevées.
- Grâce aux solides capacités de gestion des données de NetApp, les clients peuvent être assurés que leurs données sont bien protégées contre les risques et menaces potentiels. NetApp accorde la priorité à la sécurité des données, offrant ainsi aux clients la tranquillité d'esprit quant à la sécurité et à l'intégrité de leurs précieuses informations.

Base de données vectorielles

Cette section couvre la définition et l'utilisation d'une base de données vectorielle dans les solutions NetApp AI.

Base de données vectorielles

Une base de données vectorielle est un type de base de données spécialisé conçu pour gérer, indexer et rechercher des données non structurées à l'aide d'intégrations provenant de modèles d'apprentissage automatique. Au lieu d'organiser les données dans un format tabulaire traditionnel, il organise les données sous forme de vecteurs de grande dimension, également appelés plongements vectoriels. Cette structure unique permet à la base de données de gérer des données complexes et multidimensionnelles de manière plus efficace et plus précise.

L'une des principales capacités 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 ressemblent à une entrée donnée, et la détection d'anomalies, où elle peut repérer les points de données qui s'écartent considérablement de la norme.

De plus, les bases de données vectorielles sont bien adaptées pour gérer des données temporelles ou des données horodatées. Ce type de données fournit des informations sur « ce » qui s'est produit et quand cela s'est produit, dans l'ordre et par rapport à tous les autres événements au sein d'un système informatique donné. Cette capacité à gérer et à analyser des 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 à haute dimension : les bases de données vectorielles excellent dans la gestion et la récupération de données à haute dimension, qui sont souvent générées dans les applications d'IA et de ML.
- Évolutivité : ils peuvent évoluer efficacement pour gérer de grands volumes de données, soutenant ainsi la croissance et l'expansion des projets d'IA et de ML.
- Flexibilité : les bases de données vectorielles offrent un haut degré de flexibilité, permettant l'hébergement de divers types et structures de données.
- Performances : Ils offrent une gestion et une récupération de données hautes performances, essentielles à la rapidité et à 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 diverses bases de données vectorielles et les détails de leurs cas d'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 à la gestion et à la recherche de données vectorielles, ce qui en fait des ressources inestimables dans ce domaine spécialisé de la gestion des données.

Elasticsearch

Il s'agit d'un moteur de recherche et d'analyse largement utilisé, qui a récemment intégré des capacités de recherche vectorielle. Cette nouvelle fonctionnalité améliore ses fonctionnalités, lui permettant 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. Il 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 traditionnelles avec une recherche vectorielle dense basée sur l'IA, créant ainsi une approche de recherche hybride qui exploite le meilleur des deux mondes.

Principalement basé sur le cloud, Pinecone est conçu pour les applications d'apprentissage automatique et s'intègre bien à une variété de plates-formes, notamment GCP, AWS, Open AI, GPT-3, GPT-3.5, GPT-4, Catgut Plus, Elasticsearch, Haystack, et plus encore. Il est important de noter que Pinecone est une plate-forme à source fermée et est disponible en tant qu'offre de logiciel en tant que service (SaaS).

Compte tenu de ses capacités avancées, Pinecone est particulièrement bien adapté au secteur de la cybersécurité, où ses capacités de recherche hautement dimensionnelle et de recherche hybride peuvent être

exploitées efficacement pour détecter et répondre aux menaces.

Chroma

Il s'agit d'une base de données vectorielle dotée d'une API principale avec quatre fonctions principales, dont l'une comprend un magasin de vecteurs de documents en mémoire. Il utilise également la bibliothèque Face Transformers pour vectoriser les documents, améliorant ainsi sa fonctionnalité et sa polyvalence. Chroma est conçu pour fonctionner à la fois dans le cloud et sur site, offrant une flexibilité en fonction des besoins des utilisateurs. Il excelle particulièrement dans les applications liées à l'audio, ce qui en fait un excellent choix pour les moteurs de recherche basés sur l'audio, les systèmes de recommandation musicale et d'autres cas d'utilisation liés à l'audio.

Tisser

Il s'agit d'une base de données vectorielle polyvalente qui permet aux utilisateurs de vectoriser leur contenu à l'aide de ses modules intégrés ou de modules personnalisés, offrant une flexibilité en fonction des besoins spécifiques. Il propose des solutions entièrement gérées et auto-hébergées, répondant à une variété de préférences de déploiement.

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

Redis

Redis est une base de données vectorielle hautes performances connue pour son stockage rapide en mémoire, offrant une faible latence pour les opérations de lecture-écriture. Cela en fait un excellent choix pour les systèmes de recommandation, les moteurs de recherche et les applications d'analyse de données qui nécessitent un accès rapide aux données.

Redis prend en charge diverses structures de données pour les vecteurs, notamment 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 d'unions. Ces fonctionnalités sont particulièrement utiles pour la recherche de similarité, le clustering 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 offre la réplication des données. Il s'intègre également bien avec d'autres types de données, y compris les bases de données relationnelles traditionnelles (SGBDR). Redis inclut une fonctionnalité de publication/abonnement (Pub/Sub) pour les mises à jour en temps réel, ce qui est bénéfique pour la gestion des 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 offre une API semblable à un magasin de documents, un peu comme MongoDB. Il se distingue par sa prise en charge d'une grande variété de types de données, ce qui en fait un choix populaire dans les domaines de la science des données et de l'apprentissage automatique.

L'une des fonctionnalités uniques de Milvus est sa capacité de multi-vectorisation, qui permet aux utilisateurs de spécifier au moment de l'exécution le type de vecteur à utiliser pour la recherche. De plus, il utilise

Knowwhere, une bibliothèque qui se trouve au-dessus d'autres bibliothèques comme Faiss, pour gérer la communication entre les requêtes et les algorithmes de recherche vectorielle.

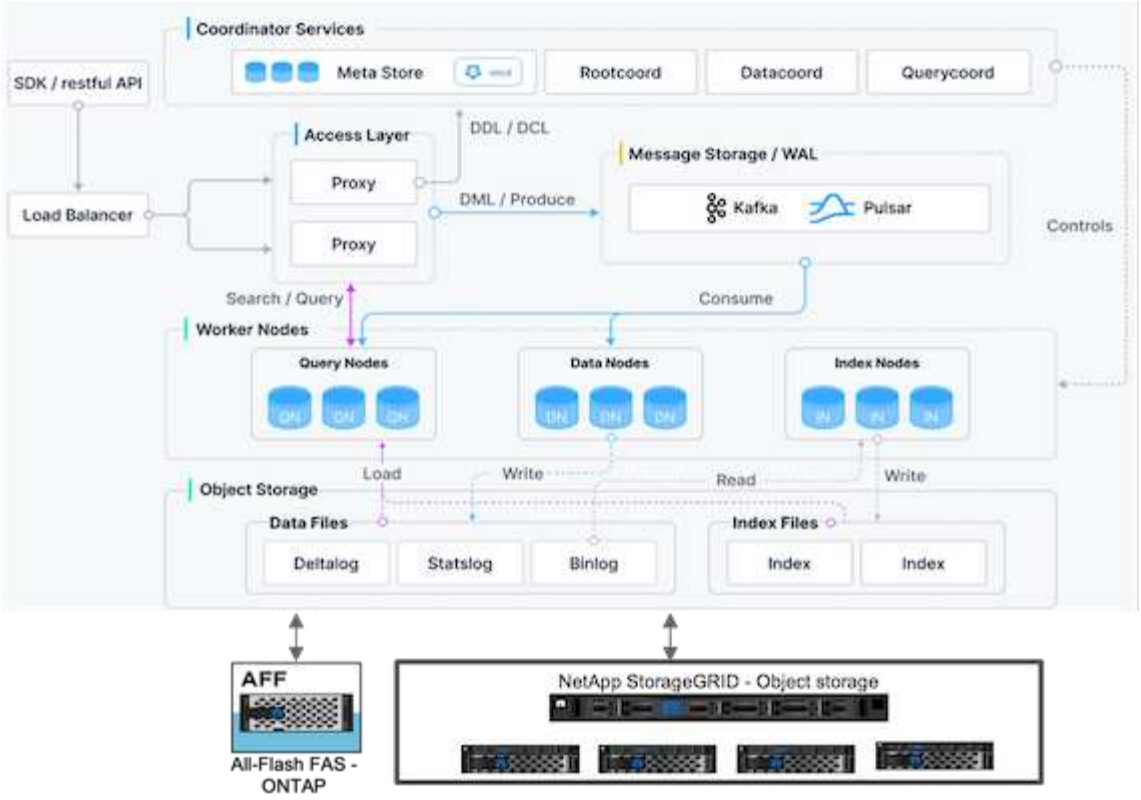
Milvus offre également une intégration transparente avec les workflows d'apprentissage automatique, grâce à sa compatibilité avec PyTorch et TensorFlow. Cela en fait un excellent outil pour une gamme d'applications, notamment le commerce électronique, l'analyse d'images et de vidéos, la reconnaissance 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 le regroupement de documents, la recherche sémantique et les systèmes de réponses aux questions.

Pour cette solution, nous avons choisi Milvus pour la validation de la solution. Pour les performances, nous avons utilisé à la fois milvus et postgres (pgvecto.rs).

Pourquoi avons-nous choisi Milvus pour cette solution ?

- Open-Source : Milvus est une base de données vectorielle open source, encourageant le développement et les améliorations pilotés par la communauté.
- Intégration de l'IA : elle exploite l'intégration de la recherche de similarité et des applications d'IA pour améliorer les fonctionnalités de la base de données vectorielle.
- Gestion de gros volumes : Milvus a la capacité de stocker, d'indexer et de gérer plus d'un milliard de vecteurs d'intégration générés par des modèles de réseaux neuronaux profonds (DNN) et d'apprentissage automatique (ML).
- Convivial : il est facile à utiliser, la configuration prenant moins d'une minute. Milvus propose également des SDK pour différents langages de programmation.
- Vitesse : Il offre des vitesses de récupération ultra-rapides, jusqu'à 10 fois plus rapides que certaines alternatives.
- Évolutivité et disponibilité : Milvus est hautement évolutif, avec des options d'évolutivité et de mise à l'échelle selon les 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 trajet, ce qui en fait un outil polyvalent pour diverses applications.

Présentation de l'architecture Milvus



Cette section fournit des composants et des services de niveau supérieur 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 de coordination – il attribue les tâches aux nœuds de travail et agit comme le cerveau du système. Il dispose de trois types de coordinateurs : coordonnées racine, coordonnées de données et coordonnées de requête. * Nœuds de travail : il suit les instructions du service de coordination et exécute les commandes DML/DDL déclenchées par l'utilisateur. Il dispose de trois types de nœuds de travail tels que le nœud de requête, le nœud de données et le nœud d'index. * Stockage : il est responsable de la persistance des données. Il comprend un stockage méta, un courtier de journaux et un stockage d'objets. Le stockage NetApp tel que ONTAP et StorageGRID fournit un stockage d'objets et un stockage basé sur des fichiers à Milvus pour les données client et les données de base de données vectorielles.

Exigences technologiques

Cette section fournit un aperçu des exigences relatives à la solution de base de données vectorielle NetApp .

Exigences technologiques

Les configurations matérielles et logicielles décrites ci-dessous ont été utilisées pour la majorité des validations effectuées dans ce document, à l'exception des performances. Ces configurations servent de guide pour vous aider à configurer votre environnement. Veuillez toutefois noter que les composants spécifiques peuvent varier en fonction des exigences individuelles des clients.

Configuration matérielle requise

Matériel	Détails
Paire de baies de stockage NetApp AFF HA	* A800 * ONTAP 9.14.1 * 48 x 3,49 To SSD-NVM * Deux volumes de groupe flexibles : métadonnées et données. * Le volume NFS de métadonnées dispose de 12 volumes persistants de 250 Go. * Les données sont un volume ONTAP NAS S3
6 x FUJITSU PRIMERGY RX2540 M4	* 64 processeurs * Processeur Intel® Xeon® Gold 6142 à 2,60 GHz * Mémoire physique de 256 Go * 1 port réseau 100 GbE
Réseautage	100 GbE
StorageGRID	* 1 x SG100, 3xSGF6024 * 3 x 24 x 7,68 To

Configuration logicielle requise

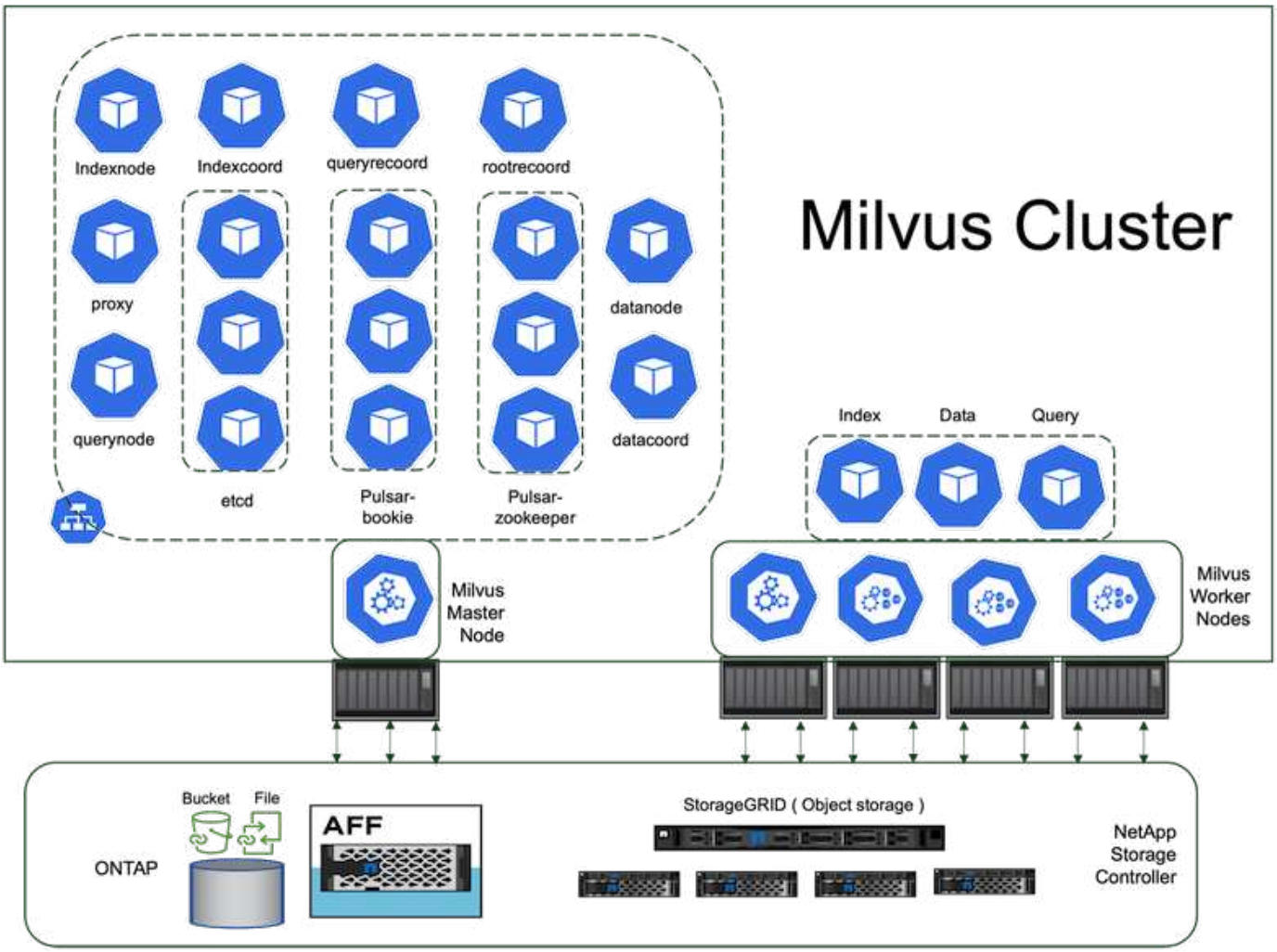
Logiciels	Détails
amas de Milvus	* GRAPHIQUE - milvus-4.1.11. * Version APP – 2.3.4 * Bundles dépendants tels que bookkeeper, zookeeper, pulsar, etcd, proxy, querynode, worker
Kubernetes	* Cluster K8s à 5 nœuds * 1 nœud maître et 4 nœuds travailleurs * Version – 1.7.2
Python	*3.10.12.

Procédure de déploiement

Cette section décrit la procédure de déploiement de la solution de base de données vectorielle pour NetApp.

Procédure de déploiement

Dans cette section de déploiement, nous avons utilisé la base de données vectorielle Milvus avec Kubernetes pour la configuration du laboratoire comme ci-dessous.



Le stockage NetApp fournit le stockage au cluster pour conserver les données des clients et les données du cluster Milvus.

Configuration du stockage NetApp – ONTAP

- Initialisation du système de stockage
- Création d'une machine virtuelle de stockage (SVM)
- Affectation des interfaces réseau logiques
- Configuration et licence NFS, S3

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

1. Créez un volume FlexGroup pour NFSv4. Dans notre configuration pour cette validation, nous avons utilisé 48 SSD, 1 SSD dédié au volume racine du contrôleur et 47 SSD répartis pour NFSv4]].Vérifiez que la politique d'exportation NFS pour le volume FlexGroup dispose d'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) pour le 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.

Veuillez suivre les étapes ci-dessous pour NAS S3 (Network Attached Storage Simple Storage Service) :

1. Créez un volume FlexGroup pour NFS.
2. Configurez un serveur de magasin d'objets avec HTTP activé et le statut d'administrateur 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 à l'aide de 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, à l'avenir, ces clés seront fournies lors du processus de création de l'utilisateur ou pourront être récupérées à l'aide d'appels d'API REST.
5. Créez un groupe object-store-server à l'aide de l'utilisateur créé à l'étape 2 et accordez l'accès. Dans cet exemple, nous avons fourni « FullAccess ».
6. Créez un bucket NAS en définissant son type sur « nas » et en fournissant le chemin d'accès au volume NFSv3. Il est également possible d'utiliser un bucket S3 à cette fin.

Configuration du stockage NetApp – StorageGRID

1. Installez le logiciel storageGRID.
2. Créez un locataire et un bucket.
3. Créez un utilisateur avec l'autorisation requise.

Veuillez vérifier plus de détails dans https://docs.netapp.com/us-en/storagegrid-116/primer/index.html

Vérification de la solution

Présentation de la solution

Nous avons réalisé une validation complète de la solution axée sur cinq domaines clés, dont les détails sont décrits ci-dessous. Chaque section examine les défis auxquels sont confrontés les clients, les solutions fournies par NetApp et les avantages qui en découlent pour le client.

1. "Configuration du cluster Milvus avec Kubernetes sur site"Les défis des clients sont de pouvoir évoluer de manière indépendante en matière de stockage et de calcul, de gestion efficace des infrastructures et de gestion des données. Dans cette section, nous détaillons le processus d'installation d'un cluster Milvus sur Kubernetes, en utilisant un contrôleur de stockage NetApp pour les données du cluster et les données client.
2. lien:vector-database-milvus-with-Amazon-FSx ONTAP-for- NetApp- ONTAP.html[Milvus avec Amazon FSx ONTAP pour NetApp ONTAP – dualité fichier et objet] Dans cette section, pourquoi nous devons déployer une base de données vectorielle dans le cloud ainsi que les étapes pour déployer une base de données vectorielle (milvus autonome) dans Amazon FSx ONTAP pour NetApp ONTAP dans des conteneurs Docker.
3. "Protection de base de données vectorielle à l'aide de NetApp SnapCenter."Dans cette section, nous examinons comment SnapCenter protège les données de la base de données vectorielle et les données Milvus résidant dans ONTAP. Pour cet exemple, nous avons utilisé un bucket NAS (milvusdbvol1) dérivé d'un volume NFS ONTAP (vol1) pour les données client et un volume NFS distinct (vectordbpv) pour les données de configuration du cluster Milvus.
4. "Reprise après sinistre avec NetApp SnapMirror"Dans cette section, nous discutons de l'importance de la reprise après sinistre (DR) pour la base de données vectorielle et de la manière dont le produit de reprise

après sinistre NetApp Snapmirror fournit une solution DR à la base de données vectorielle.

5. "Validation des performances"Dans cette section, nous visons à approfondir 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 comportement du contrôleur de stockage NetApp à la prise en charge des charges de travail RAG et d'inférence au sein du cycle de vie LLM. Nous évaluerons et identifierons les différenciateurs de performances lorsque ces bases de données seront combinées avec la solution de stockage ONTAP . Notre analyse sera basée sur des indicateurs de performance clés, tels que le nombre de requêtes traitées par seconde (QPS).

Configuration du cluster Milvus avec Kubernetes sur site

Cette section décrit la configuration du cluster Milvus pour la solution de base de données vectorielle pour NetApp.

Configuration du cluster Milvus avec Kubernetes sur site

Les défis des clients sont de pouvoir évoluer indépendamment en termes de stockage et de calcul, de gestion efficace de l'infrastructure et de gestion des données. Kubernetes et les bases de données vectorielles forment ensemble 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 données de grande dimension et les recherches de similarité. Cette combinaison permet un traitement rapide de requêtes complexes sur de grands ensembles de données et s'adapte de manière transparente aux volumes de données croissants, ce qui la rend 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, en utilisant un contrôleur de stockage NetApp pour les données du cluster et les données client.
2. Pour installer un cluster Milvus, des volumes persistants (PV) sont nécessaires pour stocker les données de divers composants du cluster Milvus. Ces composants incluent 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 et la publication/l'abonnement des flux de messages du cluster Milvus. Pour Kafka avec NFS, NetApp a apporté des améliorations à ONTAP 9.12.1 et versions ultérieures. Ces améliorations, ainsi que les modifications apportées à NFSv4.1 et Linux incluses dans RHEL 8.7 ou 9.1 et versions ultérieures, résolvent le problème de « renommage inutile » qui peut survenir lors de l'exécution de Kafka sur NFS. Si vous souhaitez obtenir des informations plus détaillées sur l'exécution de Kafka avec la solution NetApp NFS, veuillez consulter :"ce lien" .

3. Nous avons créé un seul volume NFS à partir de NetApp ONTAP et établi 12 volumes persistants, chacun avec 250 Go de stockage. La taille de stockage peut varier en fonction de la taille du cluster ; par exemple, nous avons un autre cluster où chaque PV dispose de 50 Go. Veuillez vous référer ci-dessous à l'un des fichiers PV YAML pour plus de détails ; nous avions 12 de ces fichiers au total. Dans chaque fichier, le storageClassName est défini sur « default », 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 PV YAML 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. Pour stocker les données client, Milvus prend en charge les solutions de stockage d'objets telles que MinIO, Azure Blob et S3. Dans ce guide, nous utilisons S3. Les étapes suivantes s'appliquent à la fois au magasin 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 de Milvus. Veuillez vous référer à l'annexe pour le fichier values.yaml que nous avons utilisé dans ce document.
6. Assurez-vous que la « classe de stockage » est définie sur « par défaut » dans chaque section, y compris celles du journal, etcd, zookeeper et bookkeeper.
7. Dans la section MinIO, désactivez MinIO.
8. Créez un bucket NAS à partir du stockage d'objets ONTAP ou StorageGRID et incluez-les dans un S3 externe avec les informations d'identification de stockage d'objets.

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

9. Avant de créer le cluster Milvus, assurez-vous que le PersistentVolumeClaim (PVC) ne dispose d'aucune ressource préexistante.

```
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 PersistentVolumeClaims (PVC).

```
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. Vérifiez 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>
```

Veuillez vous assurer que l'état des pods est « en cours d'exécution » et fonctionne comme prévu

13. Écriture et lecture de données de test dans le stockage d'objets Milvus et NetApp .
    - Écrivez des 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 démontré par 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 offre aux clients la possibilité de gérer des données de grande dimension et d'exécuter des requêtes complexes rapidement et efficacement, ce qui en fait une solution idéale pour les applications Big Data et les charges de travail d'IA. L'utilisation de volumes persistants (PV) pour divers composants de cluster, ainsi que la création d'un volume NFS unique à partir de NetApp ONTAP, garantissent une utilisation optimale des

ressources et une gestion des données. Le processus de vérification de l'état des PersistentVolumeClaims (PVC) et des pods, ainsi que le test de l'écriture et de la lecture des données, offrent aux clients l'assurance d'opérations de données fiables et cohérentes. L'utilisation du stockage d'objets ONTAP ou StorageGRID pour les données client améliore encore l'accessibilité et la sécurité des données. Dans l'ensemble, cette configuration offre aux clients une solution de gestion de données résiliente et performante, capable de s'adapter de manière transparente à leurs besoins croissants en matière de données.

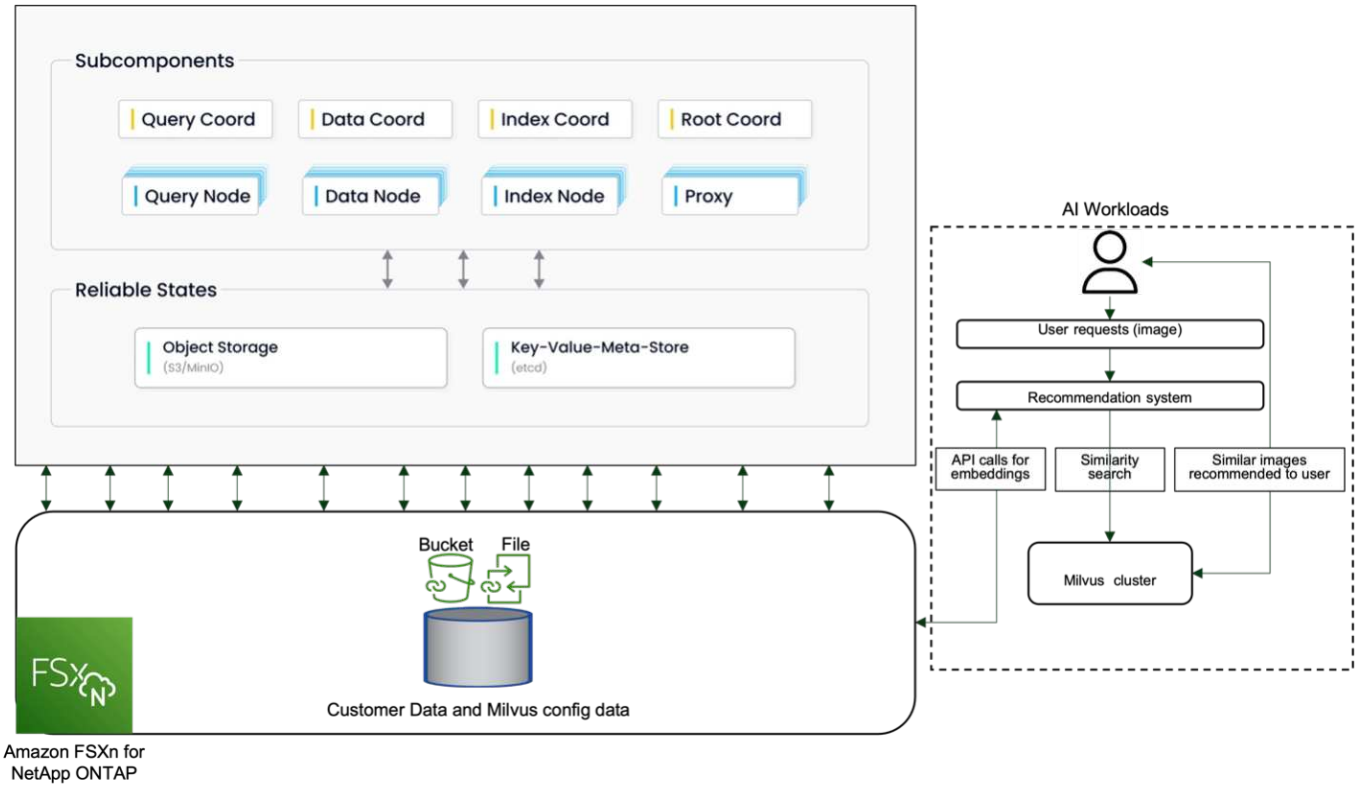
Milvus avec Amazon FSx ONTAP pour NetApp ONTAP - dualité fichier et objet

Cette section décrit la configuration du cluster milvus avec Amazon FSx ONTAP pour la solution de base de données vectorielle pour NetApp.

Milvus avec Amazon FSx ONTAP pour NetApp ONTAP – dualité fichier et objet

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

Le déploiement d'une base de données vectorielle dans le cloud offre plusieurs avantages importants, en particulier pour les applications qui nécessitent la gestion de données de grande dimension et l'exécution de recherches de similarité. Tout d'abord, le déploiement basé sur le cloud offre une évolutivité, permettant un ajustement facile des ressources pour correspondre aux volumes de données et aux charges de requêtes croissants. Cela garantit que la base de données peut gérer efficacement la demande accrue tout en maintenant des performances élevées. Deuxièmement, le déploiement dans le cloud offre une haute disponibilité et une reprise après sinistre, car les données peuvent être répliquées sur différents emplacements géographiques, minimisant ainsi le risque de perte de données et garantissant un service continu même en cas d'événements inattendus. Troisièmement, il offre une rentabilité optimale, car vous ne payez que pour les ressources que vous utilisez et pouvez augmenter ou diminuer votre capacité en fonction de la demande, évitant ainsi le besoin d'un investissement initial substantiel en matériel. Enfin, le déploiement d'une base de données vectorielle dans le cloud peut améliorer la collaboration, car les données peuvent être consultées et partagées de n'importe où, facilitant le travail en équipe et la prise de décision basée sur les données. Veuillez vérifier l'architecture du milvus autonome avec Amazon FSx ONTAP pour NetApp ONTAP utilisé dans cette validation.



1. Créez une instance Amazon FSx ONTAP pour NetApp ONTAP et notez les détails du VPC, des groupes de sécurité VPC et du sous-réseau. Ces informations seront nécessaires lors de la création d'une instance EC2. Vous pouvez trouver 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 FSx ONTAP pour NetApp ONTAP .
3. Installez nfs-common à l'aide de la commande « apt-get install nfs-common » et mettez à jour les informations du package à l'aide de « sudo apt-get update ».
4. Créez un dossier de montage et montez Amazon FSx ONTAP pour NetApp ONTAP dessus.

```
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 basé sur le fichier docker-compose.yaml, qui peut être téléchargé à partir du site Web de 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.yml, mappez le point de montage NetApp NFS au chemin du conteneur Milvus correspondant, en particulier dans etcd, minio et standalone.Check"Annexe D : docker-compose.yml" pour plus de détails sur les changements dans yml
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
vectordbvol1
ubuntu@ip-172-31-29-98:~$
```

9. Exécutez « docker-compose up -d » à partir du répertoire contenant le fichier docker-compose.yml.
10. Vérifiez 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 FSx ONTAP pour NetApp ONTAP, nous avons utilisé le SDK Python Milvus et un exemple de programme de PyMilvus. Installez les packages nécessaires en utilisant « apt-get install python3-numpy python3-pip » et installez PyMilvus en utilisant « pip3 install pymilvus ».
12. Validez les opérations d'écriture et de lecture de données depuis Amazon FSx ONTAP 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
```

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

13. 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 (lire) aux données NFS testées dans la base de données vectorielle via le protocole S3 pour les charges de travail d'IA, cela peut être validé à l'aide d'un programme Python simple. Un exemple de cela pourrait être une recherche de similarité d'images provenant 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
hoPctr9aD88c1j0SkIYZ2uPa03vlbqKA0c5feK6F
OBJECTS in the bucket milvusnasvol are :
**************************************
…
<output content removed to save page space>
…
bucket/files/insert_log/448789845791611912/448789845791611913/4487898457
91611920/0/448789845791411917/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/1/448789845791411918/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411913/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/101/448789845791411914/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/102/448789845791411915/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/1c48ab6e-
1546-4503-9084-28c629216c33/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611920/103/448789845791411916/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/0/448789845791411924/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/1/448789845791411925/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
```

```
/448789845791611913/448789845791611939/100/448789845791411920/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/101/448789845791411921/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/102/448789845791411922/xl.meta
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/b3def25f-
c117-4fba-8256-96cb7557cd6c/part.1
volumes/minio/a-bucket/files/insert_log/448789845791611912
/448789845791611913/448789845791611939/103/448789845791411923/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791211880
/448789845791211881/448789845791411889/100/448789845791411912/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611920/100/448789845791411919/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/1/xl.meta
volumes/minio/a-bucket/files/stats_log/448789845791611912
/448789845791611913/448789845791611939/100/448789845791411926/xl.meta
************************************
root@ip-172-31-29-98:~/pymilvus/examples#
```

Cette section montre efficacement comment les clients peuvent déployer et exploiter une configuration Milvus autonome dans des conteneurs Docker, en utilisant NetApp FSx ONTAP d'Amazon pour le stockage de données NetApp ONTAP . Cette configuration permet aux clients d'exploiter la puissance des bases de données vectorielles pour gérer des données de grande dimension et exécuter des requêtes complexes, le tout dans l'environnement évolutif et efficace des conteneurs Docker. En créant une instance Amazon FSx ONTAP pour NetApp ONTAP et une instance EC2 correspondante, les clients peuvent garantir une utilisation optimale des ressources et une gestion des données. La validation réussie des opérations d'écriture et de lecture de données de FSx ONTAP dans la base de données vectorielle offre aux clients l'assurance d'opérations de données fiables et cohérentes. De plus, la possibilité de répertorier (lire) les données des charges de travail d'IA via le protocole S3 offre une meilleure accessibilité aux données. Ce processus complet offre donc aux clients une solution robuste et efficace pour gérer leurs opérations de données à grande échelle, en exploitant les capacités d'Amazon FSx ONTAP pour NetApp ONTAP.

Protection de la base de données vectorielle à l'aide de SnapCenter

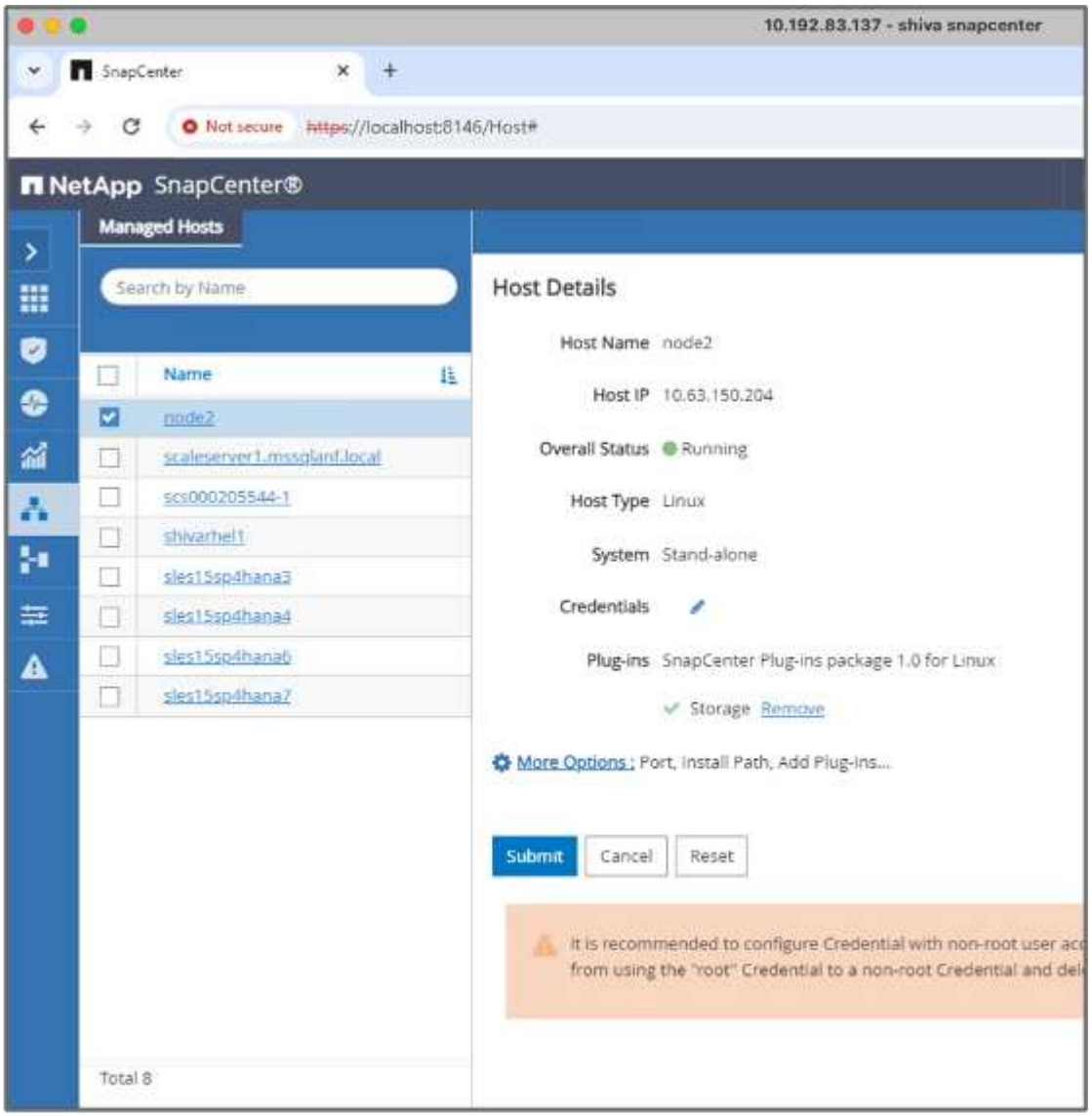
Cette section décrit comment assurer la protection des données pour la base de données vectorielle à l'aide de NetApp SnapCenter.

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

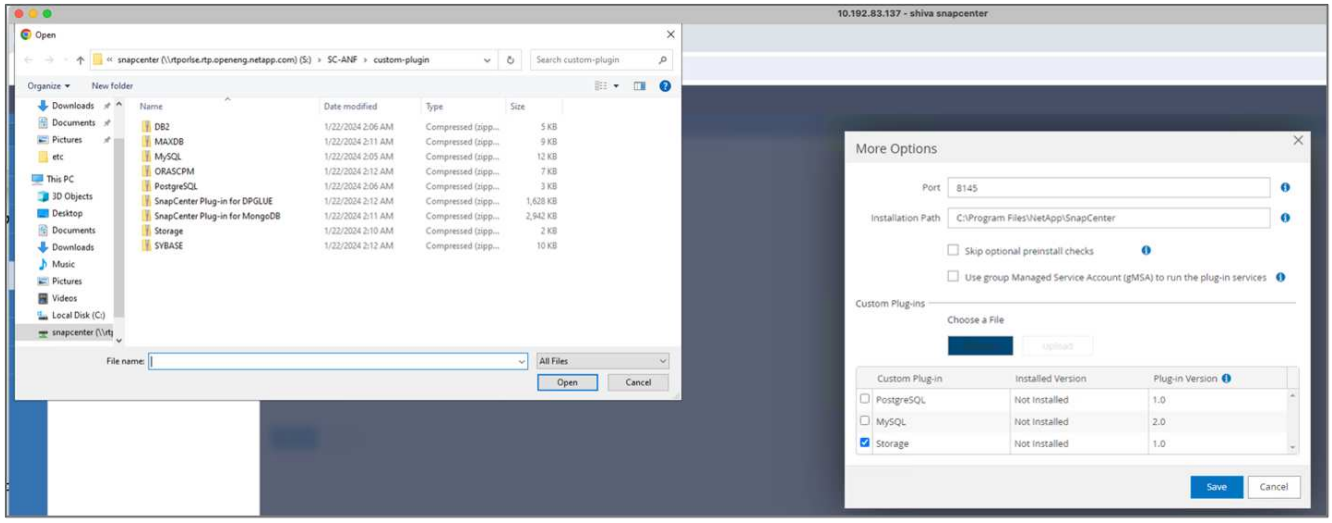
Par exemple, dans l'industrie de la production cinématographique, les clients possèdent souvent des données intégrées critiques telles que des fichiers vidéo et audio. La perte de ces données, due à des problèmes tels

que des pannes de disque dur, peut avoir un impact significatif sur leurs opérations, mettant potentiellement en péril des entreprises de plusieurs millions de dollars. Nous avons rencontré des cas où un contenu inestimable a été perdu, entraînant des perturbations et des pertes financières importantes. Assurer la sécurité et l'intégrité de ces données essentielles est donc d'une importance primordiale dans ce secteur. Dans cette section, nous examinons comment SnapCenter protège les données de la base de données vectorielle et les données Milvus résidant dans ONTAP. Pour cet exemple, nous avons utilisé un bucket NAS (milvusdbvol1) dérivé d'un volume NFS ONTAP (vol1) pour les données client 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

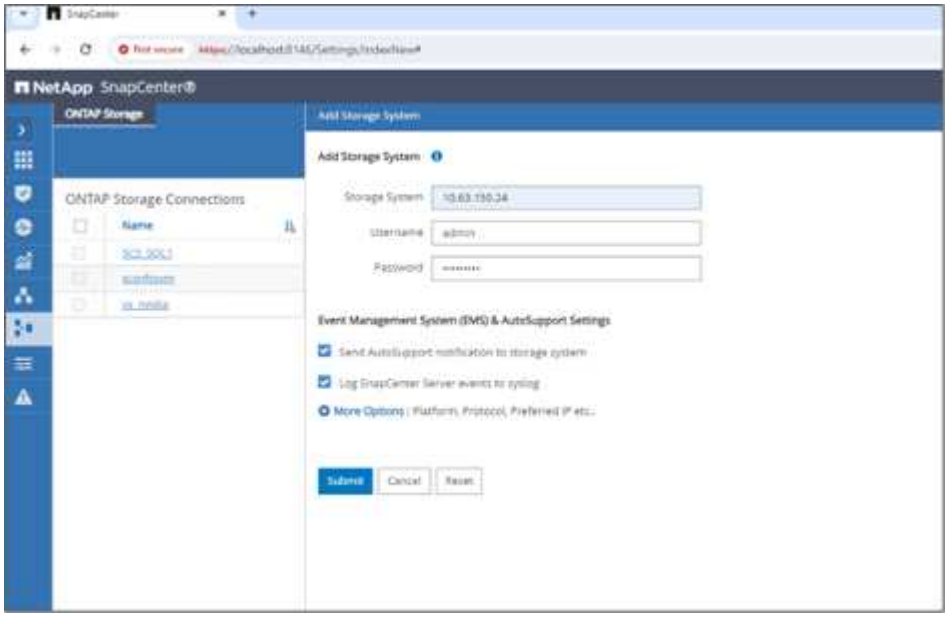
1. Configurez l'hôte qui sera utilisé pour exécuter les commandes SnapCenter .



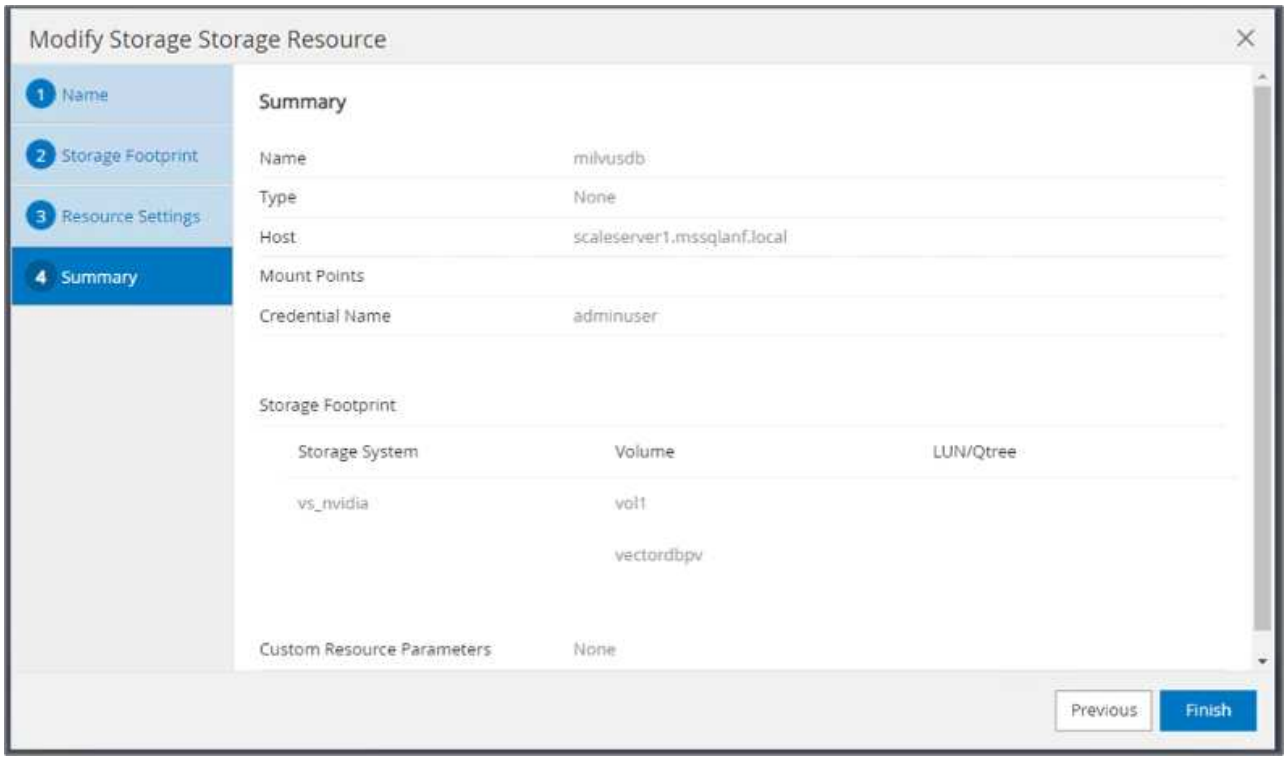
2. Installez et configurez le plugin de stockage. À partir de l'hôte ajouté, sélectionnez « Plus d'options ». Accédez au plugin de stockage téléchargé et sélectionnez-le dans le"Boutique d'automatisation NetApp" . Installez le plugin et enregistrez la configuration.



3. Configurer le système de stockage et le volume : ajoutez le système de stockage sous « Système de stockage » et sélectionnez la SVM (machine virtuelle de stockage). Dans cet exemple, nous avons choisi « vs_nvidia ».



4. Établissez une ressource pour la base de données vectorielle, en intégrant une politique de sauvegarde et un nom d'instantané personnalisé.
   - Activez la sauvegarde du groupe de cohérence avec les valeurs par défaut et activez SnapCenter sans cohérence du système de fichiers.
   - Dans la section Empreinte de stockage, sélectionnez les volumes associés aux données client de la base de données vectorielle et aux données du cluster Milvus. Dans notre exemple, il s'agit de « vol1 » et « vectordbpv ».
   - Créez une politique de protection de la base de données vectorielle et protégez les ressources de la base de données vectorielle à l'aide de la politique.



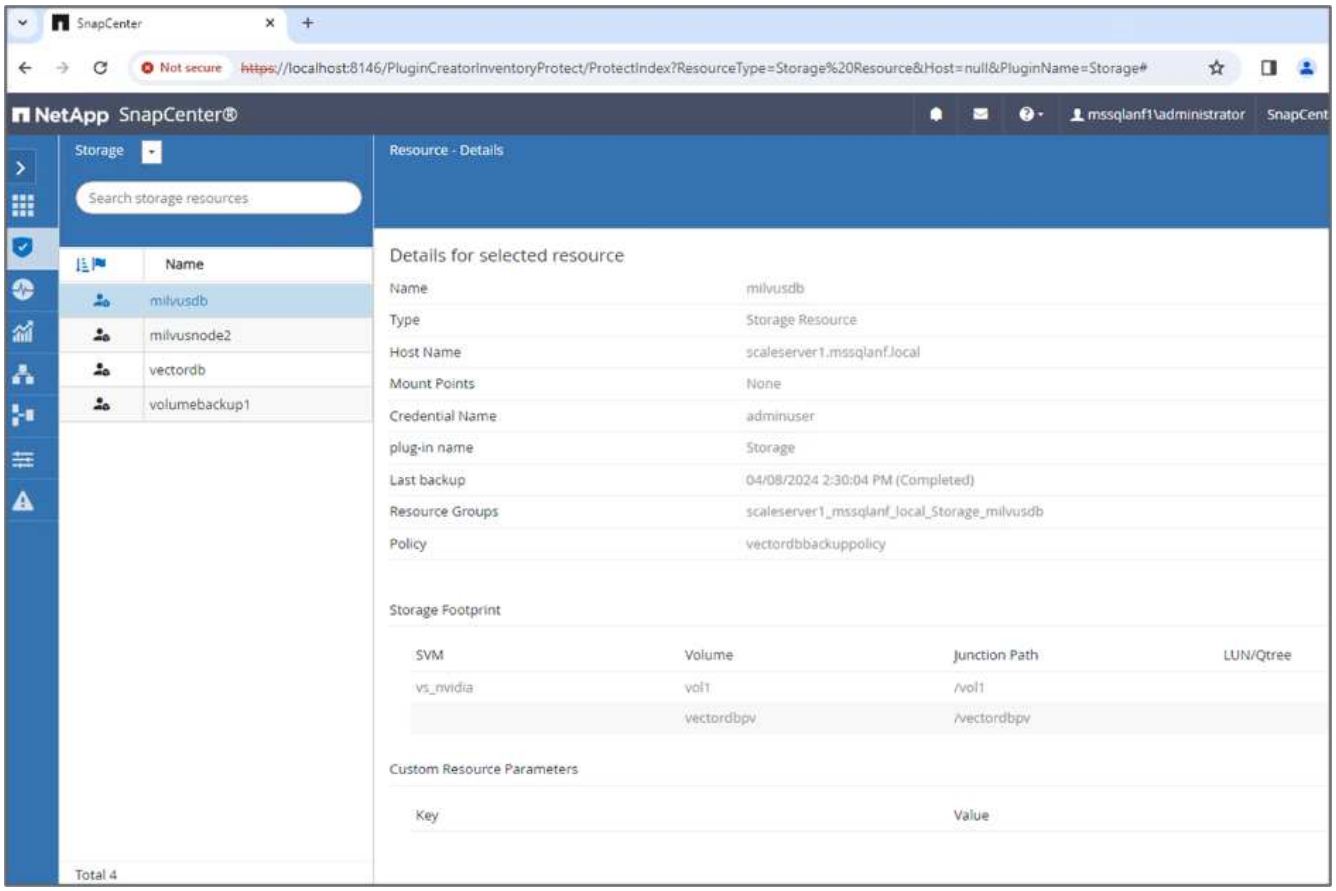
5. Insérez des données dans le bucket NAS S3 à l'aide d'un script Python. Dans notre cas, nous avons modifié le script de sauvegarde fourni par Milvus, à savoir « prepare_data_netapp.py », et exécuté la commande « sync » pour vider les données du système d'exploitation.

```
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 bucket NAS S3. Dans notre exemple, les fichiers avec l'horodatage « 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. Lancer une sauvegarde à l'aide de l'instantané du groupe de cohérence (CG) à partir de la ressource « milvusdb »

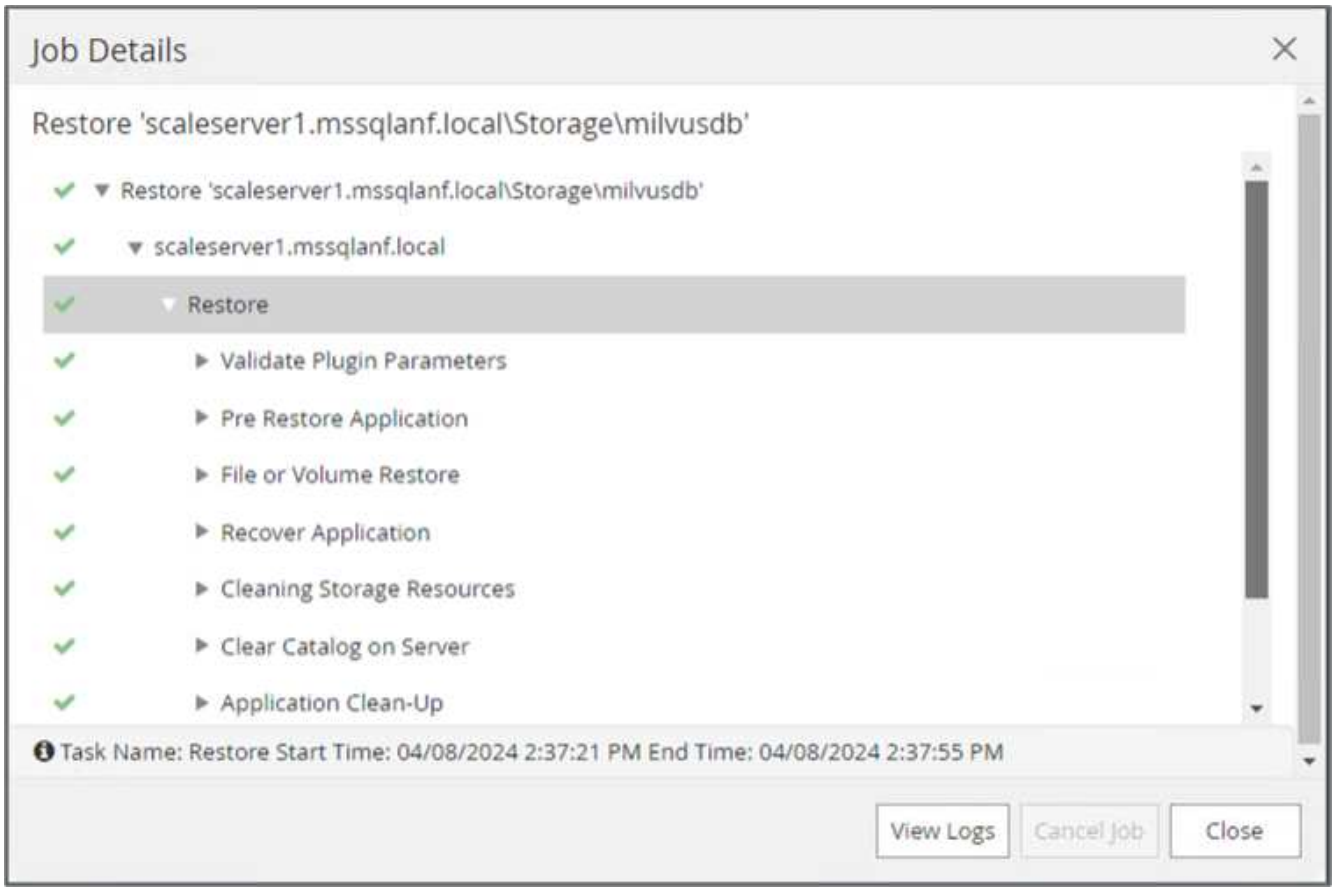


8. Pour tester la fonctionnalité de sauvegarde, nous avons soit ajouté une nouvelle table après le processus de sauvegarde, soit supprimé certaines données du NFS (bucket NAS S3).

   Pour ce test, imaginez un scénario dans lequel quelqu'un a créé une nouvelle collection inutile ou inappropriée après la sauvegarde. Dans un tel cas, nous devrions rétablir la base de données vectorielle à son état antérieur à 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 bucket NAS S3 à partir de l'instantané 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 bases de données vectorielles et les données Milvus résidant dans ONTAP offre des avantages significatifs aux clients, en particulier dans les secteurs où l'intégrité des données est primordiale, comme la production cinématographique. La capacité de SnapCenter à créer des sauvegardes cohérentes et à effectuer des restaurations complètes des données garantit que les données critiques, telles que les fichiers vidéo et audio intégrés, sont protégées contre les pertes dues à des pannes de disque dur ou à d'autres problèmes. Cela permet non seulement d'éviter les perturbations opérationnelles, mais également de se prémunir contre des pertes financières importantes.

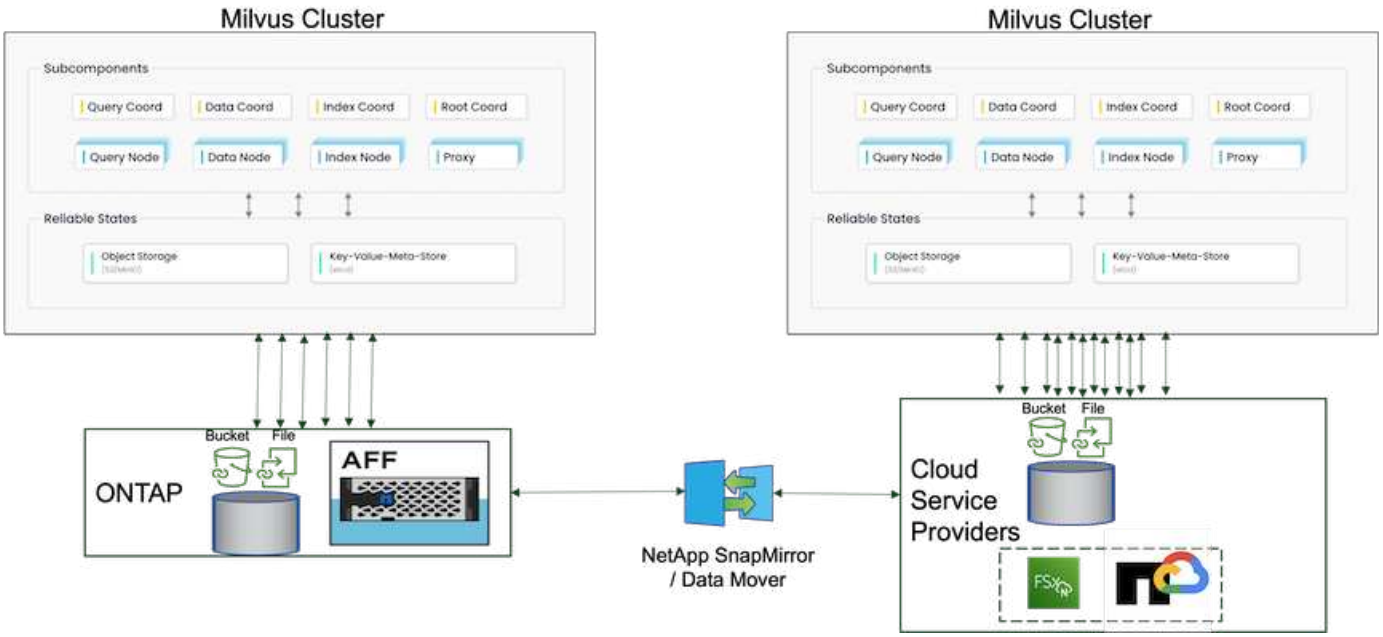
Dans cette section, nous avons démontré comment SnapCenter peut être configuré pour protéger les données résidant dans ONTAP, y compris la configuration des hôtes, l'installation et la configuration des plugins de stockage et la création d'une ressource pour la base de données vectorielle avec un nom d'instantané personnalisé. Nous avons également montré comment effectuer une sauvegarde à l'aide de l'instantané du groupe de cohérence et vérifier les données dans le bucket 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 instantané précédent garantit que la base de données vectorielle peut être rétablie à son état avant l'ajout de la nouvelle collection, préservant ainsi l'intégrité de la base de données. Cette capacité à restaurer des données à un moment précis est inestimable pour les clients, leur offrant l'assurance que leurs données sont non seulement sécurisées mais également correctement conservées. Ainsi, le produit SnapCenter de NetApp offre aux clients une solution robuste et fiable pour la protection et la gestion des données.

Reprise après sinistre avec NetApp SnapMirror

Cette section décrit la reprise après sinistre (DR) avec SnapMirror pour la solution de base de données vectorielle pour NetApp.

Reprise après sinistre avec 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 de données de grande dimension et l'exécution de recherches de similarité complexes. Une stratégie de reprise après sinistre bien planifiée et mise en œuvre garantit 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. Cela est particulièrement important pour les applications s'appuyant sur des bases de données vectorielles, où la perte ou la corruption des données pourrait entraîner des perturbations opérationnelles et des pertes financières importantes. De plus, un plan de reprise après sinistre robuste garantit également la continuité des activités en minimisant les temps d'arrêt et en permettant la restauration rapide des services. Ceci est réalisé grâce au produit de réplication de données NetApp SnapMirrror sur différents emplacements géographiques, à des sauvegardes régulières et à des mécanismes de basculement. Par conséquent, la reprise après sinistre n'est pas seulement une mesure de protection, mais un élément essentiel d'une gestion responsable et efficace des bases de données vectorielles.

SnapMirror de NetApp fournit la réplication des données d'un contrôleur de stockage NetApp ONTAP vers un autre, principalement utilisé pour la reprise après sinistre (DR) et les solutions hybrides. Dans le contexte d'une base de données vectorielle, cet outil facilite la transition fluide des données entre les environnements sur site et cloud. Cette transition est réalisée sans nécessiter de conversion de données ni de refactorisation d'application, améliorant ainsi l'efficacité et la flexibilité de la gestion des données sur plusieurs plates-formes.

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

1. Évolutivité : la solution cloud hybride de NetApp offre la possibilité de faire évoluer vos ressources en fonction de vos besoins. Vous pouvez utiliser des ressources sur site pour des charges de travail régulières et prévisibles et des ressources cloud telles qu'Amazon Amazon FSx ONTAP pour NetApp ONTAP et Google Cloud NetApp Volume (NetApp Volumes) pour les heures de pointe ou les charges inattendues.
2. Efficacité des coûts : le modèle de cloud hybride de NetApp vous permet d'optimiser vos coûts en utilisant des ressources sur site pour les charges de travail régulières et en payant uniquement pour les ressources cloud lorsque vous en avez besoin. Ce modèle de paiement à l'utilisation peut être très rentable avec une offre de service NetApp instaclustr. Pour les fournisseurs de services cloud sur site et majeurs, instaclustr fournit une assistance et des conseils.

3. Flexibilité : le cloud hybride de NetApp vous offre la flexibilité de choisir où traiter vos données. Par exemple, vous pouvez choisir d'effectuer des opérations vectorielles complexes sur site, où vous disposez d'un matériel plus puissant, et des opérations moins intensives dans le cloud.
4. Continuité des activités : en cas de sinistre, le fait de disposer de vos données dans un cloud hybride NetApp peut garantir la continuité des activités. Vous pouvez rapidement passer au cloud si vos ressources sur site sont affectées. Nous pouvons exploiter NetApp SnapMirror pour déplacer les données du site vers le cloud et vice versa.
5. Innovation : les solutions de cloud hybride de NetApp peuvent également permettre une innovation plus rapide en fournissant un accès à des services et technologies cloud de pointe. Les innovations NetApp dans le cloud telles qu'Amazon Amazon FSx ONTAP pour NetApp ONTAP, Azure NetApp Files et Google Cloud NetApp Volumes sont des produits innovants et des NAS préférés des fournisseurs de services cloud.

Validation des performances de la base de données vectorielle

Cette section met en évidence la validation des performances qui a été effectuée sur la base de données vectorielles.

Validation des performances

La validation des performances joue un rôle essentiel dans les bases de données vectorielles et les systèmes de stockage, servant de facteur clé pour garantir un fonctionnement optimal et une utilisation efficace des ressources. Les bases de données vectorielles, connues pour gérer des données de grande dimension et exécuter des recherches de similarité, doivent maintenir des niveaux de performances élevés pour traiter des requêtes complexes rapidement et avec précision. La validation des performances permet d'identifier les goulots d'étranglement, d'affiner les configurations et de garantir que le système peut gérer les charges attendues sans dégradation du service. De même, dans les systèmes de stockage, la validation des performances est essentielle pour garantir que les données sont stockées et récupérées efficacement, sans problèmes de latence ni goulots d'étranglement susceptibles d'avoir un impact sur les performances globales du système. Il permet également de prendre des décisions éclairées sur les mises à niveau ou les modifications nécessaires à l'infrastructure de stockage. Par conséquent, la validation des performances est un aspect crucial de la gestion du système, contribuant de manière significative au maintien d'une qualité de service élevée, de l'efficacité opérationnelle et de la fiabilité globale du système.

Dans cette section, nous visons à approfondir 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 comportement du contrôleur de stockage NetApp à la prise en charge des charges de travail RAG et d'inférence au sein du cycle de vie LLM. Nous évaluerons et identifierons les différenciateurs de performances lorsque ces bases de données seront combinées avec la solution de stockage ONTAP . Notre analyse sera basée sur des indicateurs de performance clés, tels que le nombre de requêtes traitées par seconde (QPS).

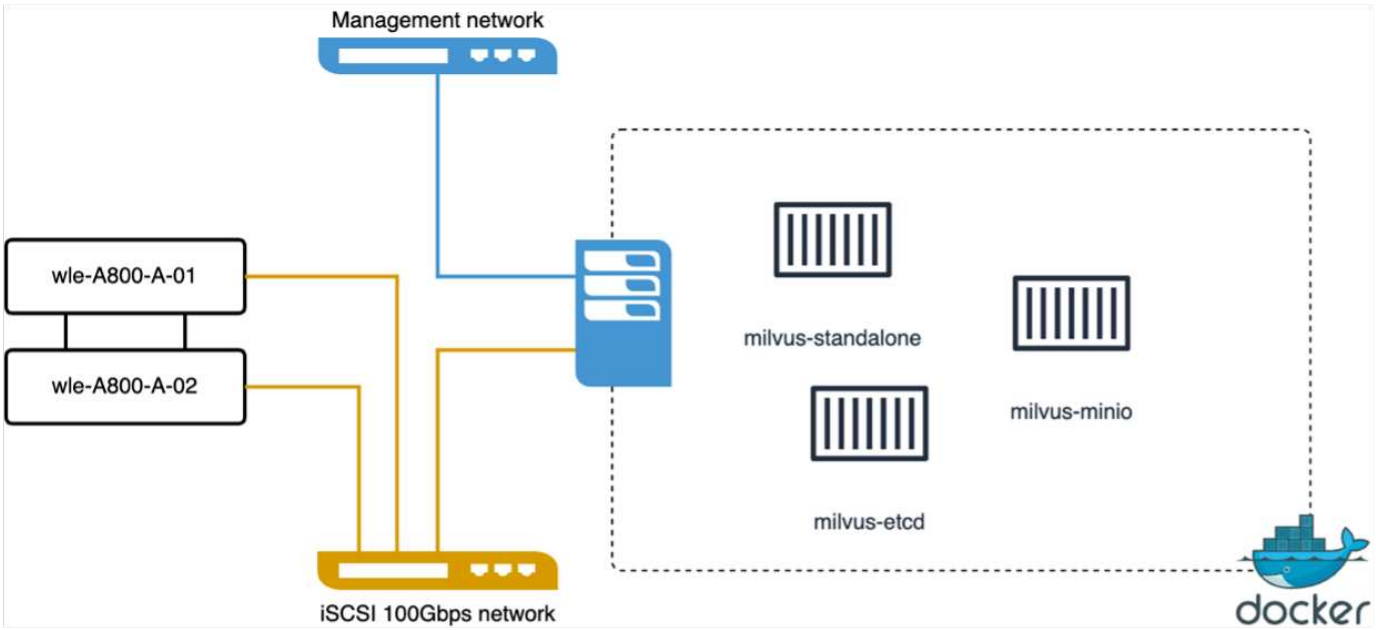
Veuillez consulter la méthodologie utilisée pour milvus et les progrès ci-dessous.

Détails	Milvus (autonome et cluster)	Postgres(pgvecto.rs) #
version	2.3.2	0.2.0
Système de fichiers	XFS sur les LUN iSCSI	
Générateur de charge de travail	"VectorDB-Bench"– v0.0.5	

Ensembles de données	Ensemble de données LAION * 10 millions d'intégrations * 768 dimensions * Taille de l'ensemble de données d'environ 300 Go	
Contrôleur de stockage	AFF 800 * Version – 9.14.1 * 4 x 100GbE – pour milvus et 2x 100GbE pour postgres * iscsi	

VectorDB-Bench avec cluster autonome Milvus

nous avons effectué la validation des performances suivante sur le 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 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 ensemble de données 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 CPU maximale de 100 %. Nous avons constaté que DiskANN est particulièrement important pour les grands ensembles de données 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 percentile pour les requêtes a été mesurée à 708,2 millisecondes.

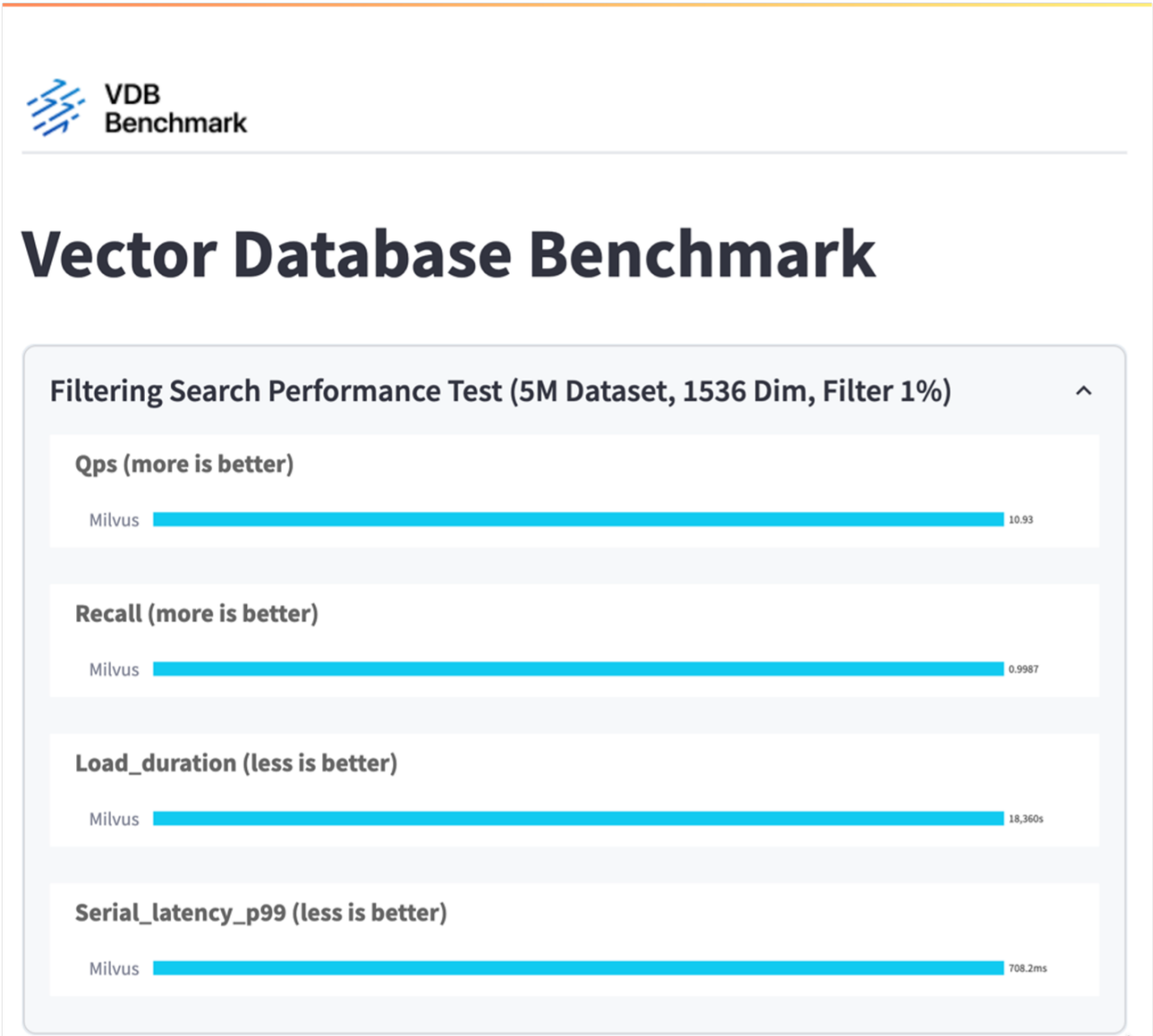
Du point de vue du stockage, la base de données a émis environ 1 000 opérations/s pendant les phases d'ingestion, d'optimisation post-insertion et de création d'index. Dans la phase de requête, il a demandé 32 000 opérations/sec.

La section suivante présente les mesures de performances de stockage.

Phase de charge de travail	Métrique	Valeur
Ingestion de données et optimisation post-insertion	Op E/S par sec	< 1 000

Phase de charge de travail	Métrique	Valeur
	Latence	< 400 unités d'utilisation
	Charge de travail	Mix lecture/écriture, principalement écriture
	Taille des E/S	64 Ko
Requête	Op E/S par sec	Pic à 32 000
	Latence	< 400 unités d'utilisation
	Charge de travail	Lecture à 100 % en cache
	Taille des E/S	Principalement 8 Ko

Le résultat de vectorDB-bench est ci-dessous.



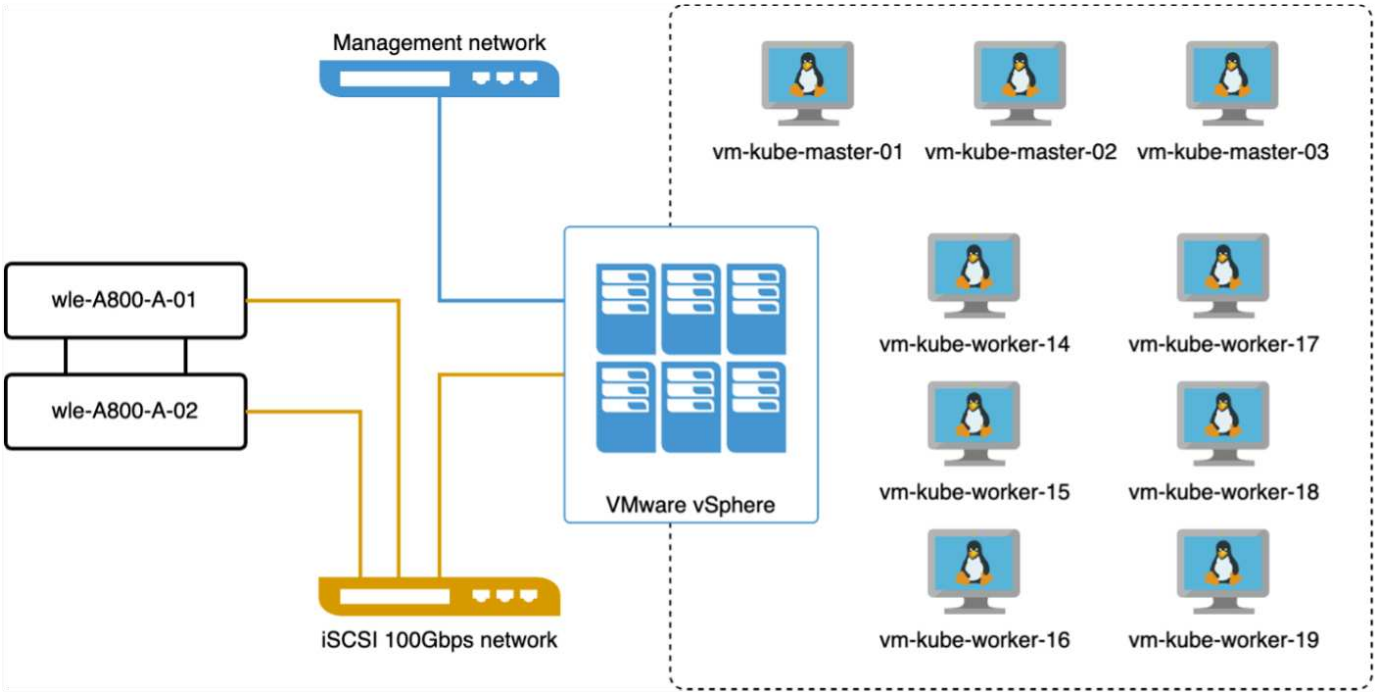
D'après la validation des performances de l'instance autonome Milvus, il est évident que la configuration

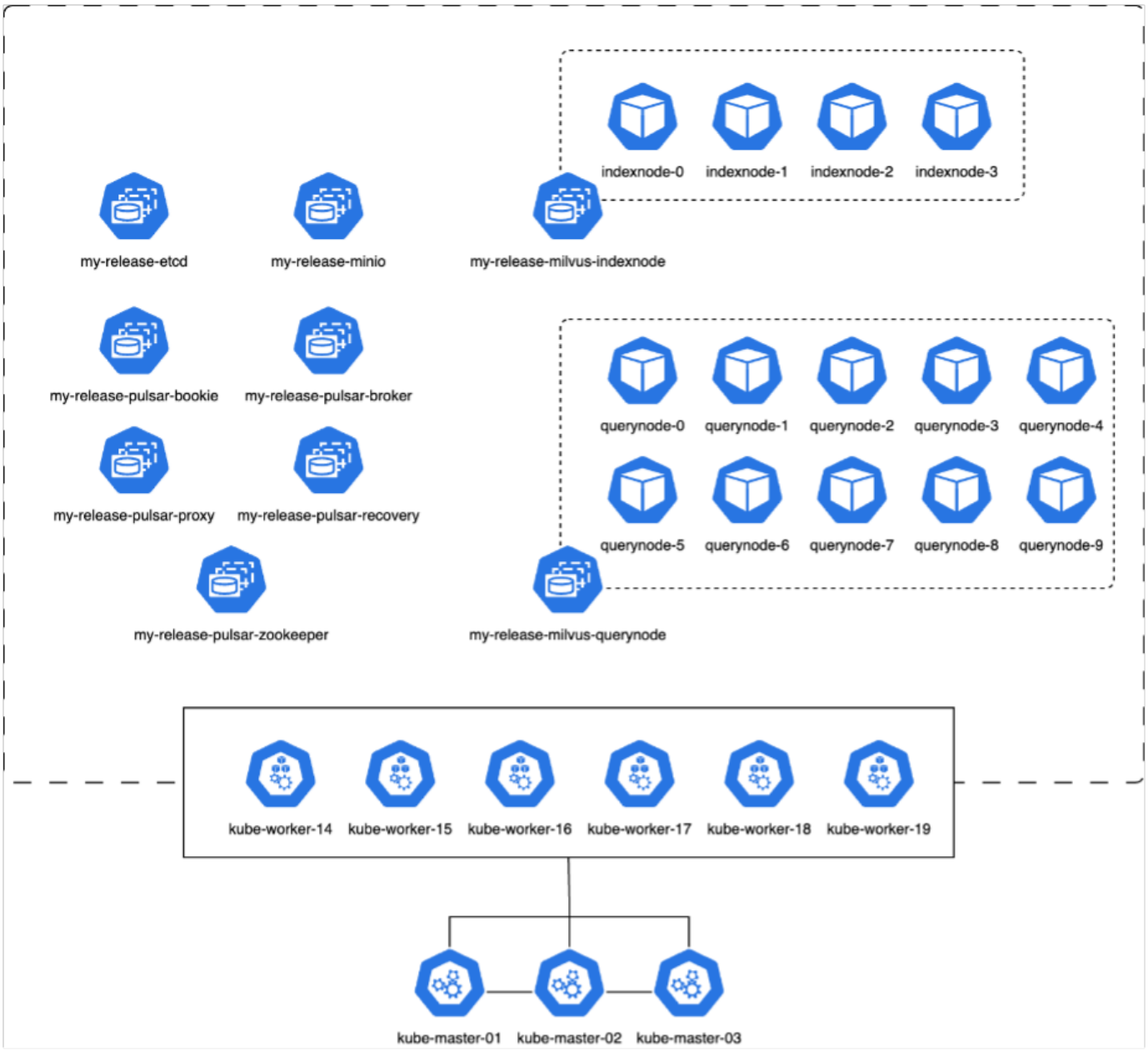
actuelle est insuffisante pour prendre en charge un ensemble de données de 5 millions de vecteurs avec une dimensionnalité de 1536. Nous avons déterminé que le stockage possède des ressources adéquates et ne constitue pas un goulot d'étranglement dans le système.

VectorDB-Bench avec cluster Milvus

Dans cette section, nous discutons du déploiement d'un cluster Milvus dans un environnement Kubernetes. Cette configuration Kubernetes a été construite sur un déploiement VMware vSphere, qui hébergeait les nœuds maître et de travail Kubernetes.

Les détails des déploiements VMware vSphere et Kubernetes sont présentés dans les sections suivantes.





Dans cette section, nous présentons nos observations et résultats issus 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 en cluster lorsque l'on travaille avec 5 millions de vecteurs à une dimensionnalité de 1536. Nous avons observé que le temps nécessaire à l'ingestion des données et à l'optimisation post-insertion était plus faible dans le déploiement en cluster. La latence du 99e percentile pour les requêtes a été réduite de six fois dans le déploiement du cluster par rapport à la configuration autonome. * Bien que le taux de requêtes par seconde (QPS) soit 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 fournissent une vue de diverses mesures de stockage, notamment la latence du cluster

de stockage et le nombre total d'IOPS (opérations d'entrée/sortie par seconde).

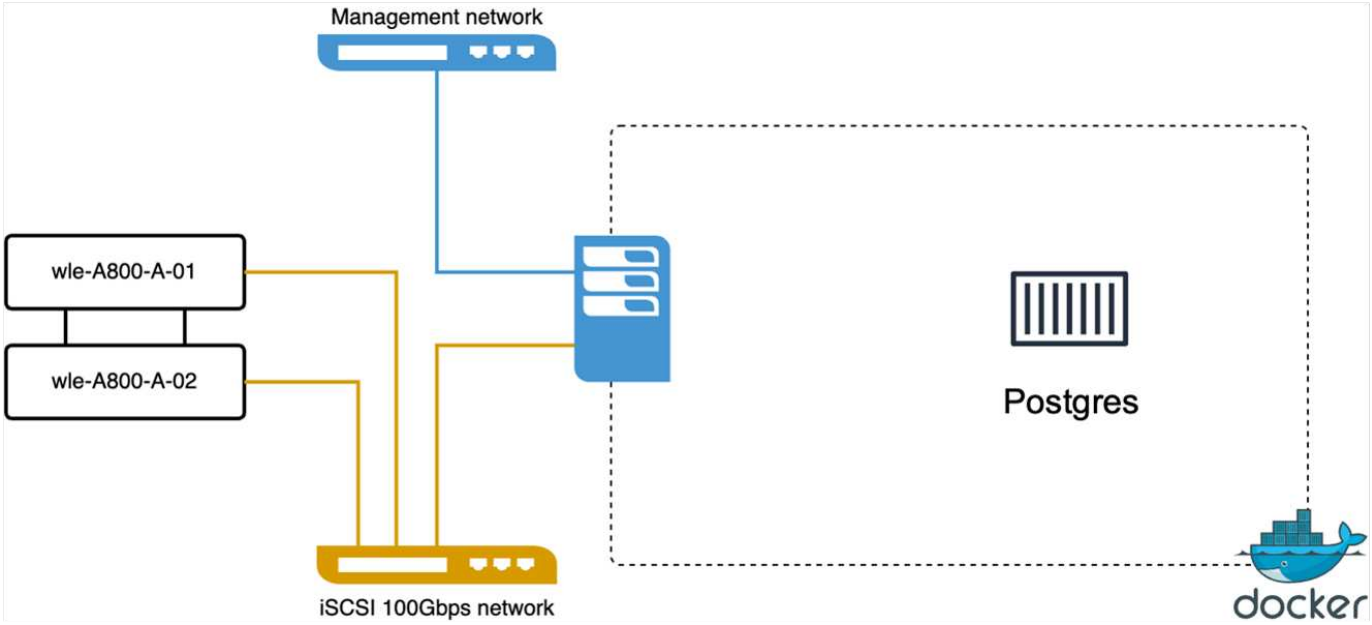
La section suivante présente les principales mesures de performances de stockage.

Phase de charge de travail	Métrique	Valeur
Ingestion de données et optimisation post-insertion	Op E/S par sec	< 1 000
	Latence	< 400 unités d'utilisation
	Charge de travail	Mix lecture/écriture, principalement écriture
	Taille des E/S	64 Ko
Requête	Op E/S par sec	Pic à 147 000
	Latence	< 400 unités d'utilisation
	Charge de travail	Lecture à 100 % en cache
	Taille des E/S	Principalement 8 Ko

Sur la base de la validation des performances du Milvus autonome et du cluster Milvus, nous présentons les détails du profil d'E/S de stockage. * Nous avons observé que le profil d'E/S reste cohérent dans les déploiements autonomes et en cluster. * La différence observée dans les IOPS de pointe peut être attribuée au plus grand nombre de clients dans le déploiement du cluster.

vectorDB-Bench avec Postgres (pgvecto.rs)

Nous avons effectué les actions suivantes sur PostgreSQL (pgvecto.rs) en utilisant VectorDB-Bench : Les détails concernant la connectivité réseau et serveur de PostgreSQL (en particulier, pgvecto.rs) sont les suivants :

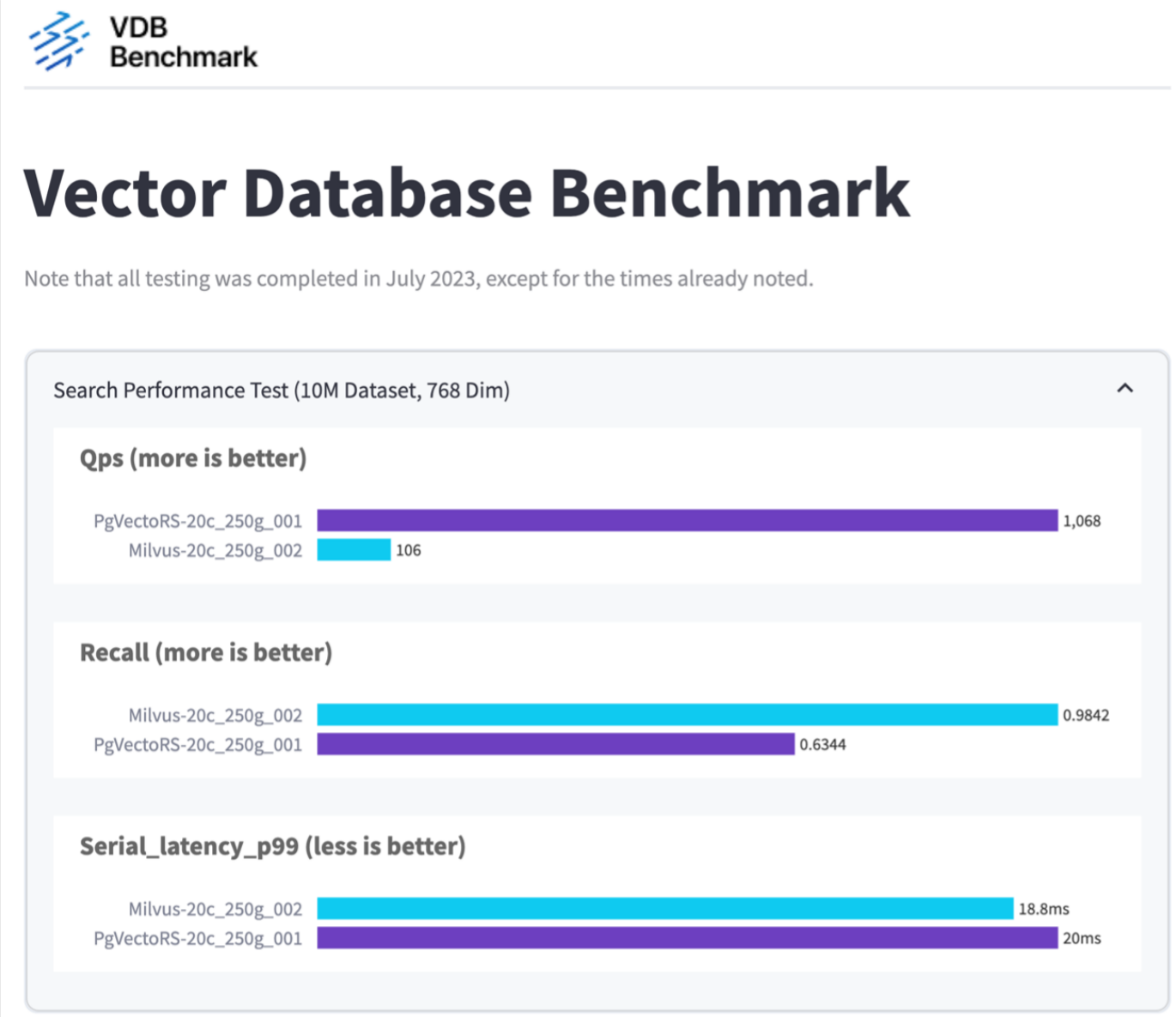


Dans cette section, nous partageons nos observations et résultats des tests de la base de données PostgreSQL, en particulier à l'aide de pgvecto.rs. * Nous avons sélectionné HNSW comme type d'index pour ces tests car au moment des tests, DiskANN n'était pas disponible pour pgvecto.rs. * Au cours de la phase d'ingestion des données, nous avons chargé l'ensemble de données Cohere, qui se compose de 10 millions de vecteurs d'une dimensionnalité 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 percentile pour les requêtes a été mesurée à 20 millisecondes. Pendant la majeure partie de l'exécution, le processeur client fonctionnait à 100 % de sa capacité.

Les images ci-dessous fournissent une vue de diverses mesures de stockage, notamment la latence du cluster de stockage et le nombre total d'IOPS (opérations d'entrée/sortie par seconde).


```
 The following section presents the key storage performance metrics.
image:pgvecto-storage-perf-metrics.png["Figure montrant une boîte de
dialogue d'entrée/sortie ou représentant un contenu écrit"]
```

Comparaison des performances entre Milvus et Postgres sur Vector DB Bench



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'indice : HNSW
- Ensemble de données : Cohere avec 10 millions de vecteurs à 768 dimensions

Nous avons constaté que pgvecto.rs atteignait un taux de requêtes par seconde (QPS) de 1 068 avec un rappel de 0,6344, tandis que Milvus atteignait un taux de QPS de 106 avec un rappel de 0,9842.

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

Observation basée sur nos validations de performance :

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

Dans Milvus, le profil d'E/S ressemble beaucoup à une charge de travail OLTP, telle que celle observée avec Oracle SLOB. Le benchmark se compose de trois phases : l'ingestion des données, la post-optimisation et la requête. Les étapes initiales sont principalement caractérisées par des opérations d'écriture de 64 Ko, tandis que la phase de requête implique principalement des lectures de 8 Ko. Nous nous attendons à ce ONTAP gère efficacement la charge d'E/S Milvus.

Le profil d'E/S PostgreSQL ne présente pas de charge de travail de stockage difficile. Étant donné l'implémentation en mémoire actuellement en cours, nous n'avons observé aucune E/S disque pendant la phase de requête.

DiskANN apparaît 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. Cependant, il est peu probable d'établir une différenciation des performances de stockage avec des indices de base de données vectoriels en mémoire tels que HNSW.

Il convient également 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. L'implication ici est que les performances de stockage n'ont pas d'impact significatif sur les performances globales de ces applications.

Base de données vectorielle avec Instaclustr utilisant PostgreSQL : pgvector

Cette section décrit les spécificités de la manière dont le produit instaclustr s'intègre à la fonctionnalité postgreSQL sur pgvector dans la solution de base de données vectorielle pour NetApp.

Base de données vectorielle avec Instaclustr utilisant PostgreSQL : pgvector

Dans cette section, nous examinons en détail la manière dont le produit instaclustr s'intègre à la fonctionnalité postgreSQL sur pgvector. Nous avons un exemple de « Comment améliorer la précision et les performances de votre LLM avec PGVector et PostgreSQL : Introduction aux intégrations et au rôle de PGVector ». Veuillez vérifier le"blog" pour obtenir plus d'informations.

Cas d'utilisation de bases de données vectorielles

Cette section fournit un aperçu des cas d'utilisation de la solution de base de données vectorielle NetApp .

Cas d'utilisation de bases de données vectorielles

Dans cette section, nous discutons de deux cas d'utilisation tels que la récupération augmentée avec de grands modèles de langage et le chatbot informatique NetApp .

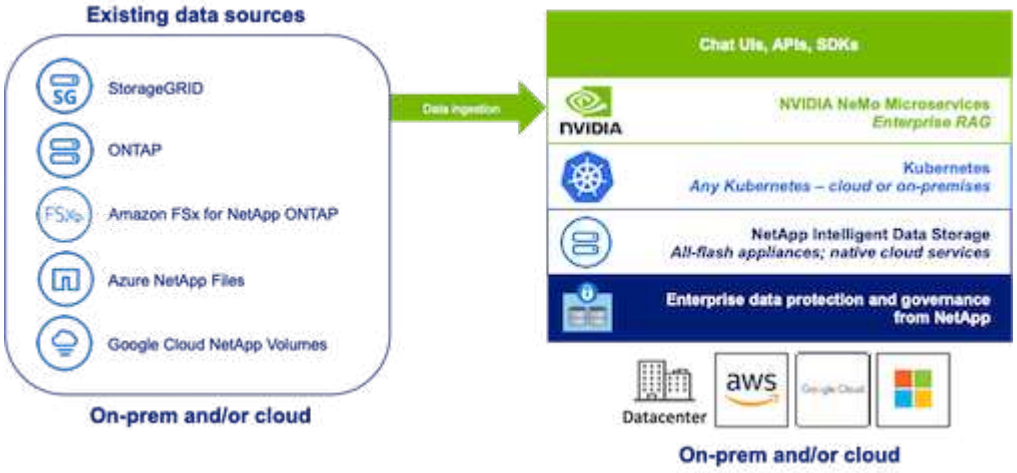
Génération augmentée de récupération (RAG) avec de grands modèles de langage (LLM)

```
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 la mise en œuvre de 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 stocker les intégrations de la base de connaissances. Reportez-vous à la documentation pour plus de détails.

```
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.
```

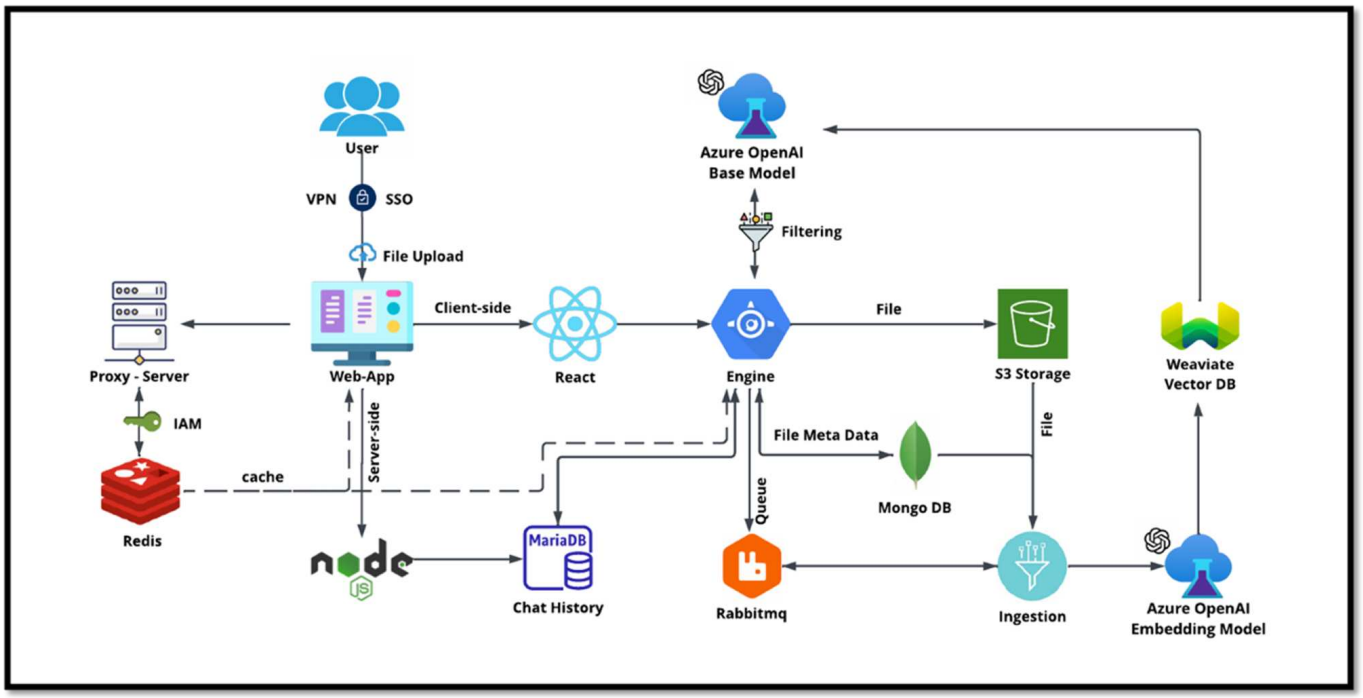
Figure 1) Enterprise RAG optimisé par NVIDIA NeMo Microservices et NetApp



Cas d'utilisation du chatbot informatique NetApp

Le chatbot de NetApp sert d'autre cas d'utilisation en temps réel pour la base de données vectorielle. Dans ce cas, NetApp Private OpenAI Sandbox fournit une plate-forme efficace, sécurisée et efficiente pour gérer les requêtes des utilisateurs internes de NetApp. En intégrant des protocoles de sécurité rigoureux, des systèmes de gestion de données efficaces et des capacités de traitement d'IA sophistiquées, il garantit des réponses précises et de haute qualité aux utilisateurs en fonction de leurs rôles et responsabilités au sein de l'organisation via l'authentification SSO. Cette architecture met en évidence le potentiel de fusion de

technologies avancées pour créer des systèmes intelligents axés sur l'utilisateur.



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

Authentification et vérification de l'utilisateur :

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

Transmission et traitement des données :

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

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 pour le traitement.
- L'entrée nettoyée est ensuite envoyée au modèle de base Azure OpenAI, qui génère une réponse basée sur l'entrée.

Génération et modération des réponses :

- La réponse du modèle de base est d'abord vérifiée pour garantir qu'elle est exacte et qu'elle répond 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

Cette section conclut la solution de base de données vectorielle pour NetApp.

Conclusion

En conclusion, ce document fournit un aperçu complet du déploiement et de la gestion des bases de données vectorielles, telles que Milvus et pgvector, sur les solutions de stockage NetApp . Nous avons discuté des directives d'infrastructure pour exploiter le stockage d'objets NetApp ONTAP et StorageGRID et validé la base de données Milvus dans AWS FSx ONTAP via le magasin de fichiers et d'objets.

Nous avons exploré la dualité fichier-objet de NetApp, démontrant son utilité non seulement pour les données des bases de données vectorielles, mais également pour d'autres applications. Nous avons également souligné comment SnapCenter, le produit de gestion d'entreprise de NetApp, offre des fonctionnalités de sauvegarde, de restauration et de clonage pour les données de bases de données vectorielles, garantissant ainsi l'intégrité et la disponibilité des données.

Le document examine également comment la solution Hybrid Cloud de NetApp offre une réplication et une protection des données dans les environnements sur site et cloud, offrant une expérience de gestion des données transparente et sécurisée. Nous avons fourni des informations sur la validation des performances des bases de données vectorielles telles que Milvus et pgvecto sur NetApp ONTAP, offrant des informations précieuses sur leur efficacité et leur évolutivité.

Enfin, nous avons discuté de deux cas d'utilisation d'IA générative : RAG avec LLM et ChatAI interne de NetApp. Ces exemples pratiques soulignent les applications et les avantages concrets des concepts et des pratiques décrits dans ce document. Dans l'ensemble, ce document sert de guide complet pour quiconque souhaite tirer parti des puissantes solutions de stockage de NetApp pour la gestion des bases de données vectorielles.

Remerciements

L'auteur souhaite remercier sincèrement les contributeurs ci-dessous, ainsi que les autres personnes qui ont fourni leurs commentaires et leurs commentaires pour rendre ce document utile aux clients et aux NetApp .

1. Sathish Thyagarajan, ingénieur marketing technique, ONTAP AI & Analytics, NetApp
2. Mike Oglesby, ingénieur marketing 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 principal, gestion des produits FSX, NetApp
6. Yuval Kalderon, chef de produit senior, équipe produit FSx, NetApp

Où trouver des informations supplémentaires

Pour en savoir plus sur les informations décrites dans ce document, consultez les documents et/ou sites Web suivants :

- Documentation Milvus - https://milvus.io/docs/overview.md
- Documentation autonome Milvus - https://milvus.io/docs/v2.0.x/install_standalone-docker.md

- Documentation produit NetApphttps://www.netapp.com/support-and-training/documentation/[]
- instaclustr -"documentation d'installclustr"

Historique des versions

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

Annexe A : Values.yaml

Cette section fournit un exemple de code YAML pour les valeurs utilisées dans la solution de base de données vectorielle NetApp .

Annexe A : Values.yaml

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

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

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

# Global tolerations
# If set, this will apply to all milvus components
```

```
# Individual components can be set to a different tolerations
tolerations: []

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

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

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

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

```
  labels: {}

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

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

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

serviceAccount:
  create: false
  name:
  annotations:
  labels:

metrics:
```

```
  enabled: true

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

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

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

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

  persistence:
    mountPath: "/milvus/logs"
    ## If true, create/use a Persistent Volume Claim
    ## If false, use emptyDir
    ##
    enabled: false
    annotations:
      helm.sh/resource-policy: keep
    persistentVolumeClaim:
      existingClaim: ""
      ## Milvus Logs Persistent Volume Storage Class
```

```
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic
provisioning
      ## If undefined (the default) or set to null, no storageClassName
spec is
      ##   set, choosing the default provisioner.
      ## ReadWriteMany access mode required for milvus cluster.
      ##
      storageClass: default
      accessModes: ReadWriteMany
      size: 10Gi
      subPath: ""

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

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

  ## Default message queue for milvus standalone
  ## Supported value: rocksmq, natsmq, pulsar and kafka
  messageQueue: rocksmq
  persistence:
```

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

proxy:
  enabled: true
  # You can set the number of replicas to -1 to remove the replicas field
in case you want to use HPA
  replicas: 1
  resources: {}
  nodeSelector: {}
  affinity: {}
  tolerations: []
  extraEnv: []
  heaptrack:
    enabled: false
  profiling:
    enabled: false  # Enable live profiling
  http:
    enabled: true  # whether to enable http rest server
    debugMode:
      enabled: false
  # Mount a TLS secret into proxy pod
  tls:
    enabled: false
## when enabling proxy.tls, all items below should be uncommented and the
key and crt values should be populated.
#    enabled: true
```

```
#    secretName: milvus-tls
## expecting base64 encoded values here: i.e. $(cat tls.crt | base64 -w 0)
and $(cat tls.key | base64 -w 0)
#    key: LS0tLS1CRUdJTiBQU--REDUCT
#    crt: LS0tLS1CRUdJTiBDR--REDUCT
#  volumes:
#  - secret:
#      secretName: milvus-tls
#    name: milvus-tls
#  volumeMounts:
#  - mountPath: /etc/milvus/certs/
#    name: milvus-tls

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

  service:
    type: ClusterIP
    port: 9000

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

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

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

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

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

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

  service:
    type: ClusterIP
    port: 2379
    peerPort: 2380

  auth:
    rbac:
      enabled: false

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

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

  readinessProbe:
    enabled: true
    periodSeconds: 20
    timeoutSeconds: 10

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

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

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

pulsar:
  enabled: true
  name: pulsar

  fullnameOverride: ""
  persistence: true

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

  rbac:
    enabled: false
    psp: false
    limit_to_namespace: true

  affinity:
```

```
    anti_affinity: false

## enableAntiAffinity: no

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

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

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

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

  bookkeeper:
    replicaCount: 3
    volumes:
      persistence: true
      journal:
        name: journal
        size: 100Gi
        storageClassName: default
      ledgers:
        name: ledgers
        size: 200Gi
        storageClassName: default
    resources:
      requests:
        memory: 2048Mi
        cpu: 1
    configData:
      PULSAR_MEM: >
```

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

  broker:
    component: broker
    podMonitor:
      enabled: false
    replicaCount: 1
    resources:
      requests:
        memory: 4096Mi
        cpu: 1.5
    configData:
      PULSAR_MEM: >
        -Xms4096m
        -Xmx4096m
        -XX:MaxDirectMemorySize=8192m
      PULSAR_GC: >
        -Dio.netty.leakDetectionLevel=disabled
        -Dio.netty.recycler.linkCapacity=1024
        -XX:+ParallelRefProcEnabled
        -XX:+UnlockExperimentalVMOptions
        -XX:+DoEscapeAnalysis
        -XX:ParallelGCThreads=32
        -XX:ConcGCThreads=32
        -XX:G1NewSizePercent=50
        -XX:+DisableExplicitGC
```

```
      -XX:-ResizePLAB
      -XX:+ExitOnOutOfMemoryError
    maxMessageSize: "104857600"
    defaultRetentionTimeInMinutes: "10080"
    defaultRetentionSizeInMB: "-1"
    backlogQuotaDefaultLimitGB: "8"
    ttlDurationDefaultInSeconds: "259200"
    subscriptionExpirationTimeMinutes: "3"
    backlogQuotaDefaultRetentionPolicy: producer_exception
  pdb:
    usePolicy: false

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

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

pulsar_manager:
  service:
    type: ClusterIP

pulsar_metadata:
  component: pulsar-init
  image:
```

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


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

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

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

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

  persistence:
```

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

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

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

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

  service:
    type: ClusterIP
    ports:
      client: 9092

  zookeeper:
    enabled: true
    replicaCount: 3

###################################
# External S3
# - these configs are only used when `externalS3.enabled` is true
###################################
externalS3:
  enabled: true
  host: "192.168.150.167"
  port: "80"
  accessKey: "24G4C1316APP2BIPDE5S"
  secretKey: "Zd28p43rgZaU44PX_ftT279z9nt4jBSro97j87Bx"
```

```
  useSSL: false
  bucketName: "milvusdbvol1"
  rootPath: ""
  useIAM: false
  cloudProvider: "aws"
  iamEndpoint: ""
  region: ""
  useVirtualHost: false

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

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

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

###################################
# External kafka
# - these configs are only used when `externalKafka.enabled` is true
###################################
```

```
externalKafka:
  enabled: false
  brokerList: localhost:9092
  securityProtocol: SASL_SSL
  sasl:
    mechanisms: PLAIN
    username: ""
    password: ""
root@node2:~#
```

Annexe B : prepare_data_netapp_new.py

Cette section fournit un exemple de script Python utilisé pour préparer les données de la base de données vectorielle.

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

Cette section contient un exemple de script Python qui peut être utilisé pour valider la base de données vectorielle dans la solution de base de données vectorielle NetApp .

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

Cette section comprend un exemple de code YAML pour la solution de base de données vectorielle pour NetApp.

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
```