Présentation de la vérification de la solution

NetApp Solutions

NetApp
May 03, 2024

Sommaire

Présentation de la solution

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

1. "Configuration de clusters Milvus avec Kubernetes sur site"
   Challenges des clients : faire évoluer indépendamment les ressources de stockage et de calcul, et gérer efficacement l'infrastructure et les données. Dans cette section, nous détaillons le processus d'installation d'un cluster Milvus sur Kubernetes avec un contrôleur de stockage NetApp pour les données de cluster et les données client.
2. "Milvus avec Amazon FSxN pour NetApp ONTAP : dualité fichier/objet"
   Dans cette section, pourquoi nous devons déployer une base de données vectorielle dans le cloud ainsi que des étapes pour déployer une base de données vectorielle (milvus standalone) dans Amazon FSxN pour NetApp ONTAP dans des conteneurs docker.
3. "Protection de base de données Vector à l'aide de NetApp SnapCenter."
   Dans cette section, nous allons examiner comment SnapCenter protège les données de la base de données vectorielle et les données Milvus résidant dans ONTAP. Dans cet exemple, nous avons utilisé un compartiment NAS (milvusdbvol1) dérivé d'un volume ONTAP NFS (vol1) pour les données clients, et un volume NFS distinct (vectordbpv) pour les données de configuration du cluster Milvus.
4. "Reprise sur incident à l'aide de NetApp SnapMirror"
   Dans cette section, nous parlerons de l'importance de la reprise sur incident pour la base de données vectorielle et de la manière dont snapmirror, produit de reprise sur incident de NetApp, fournit une solution de reprise sur incident pour la base de données vectorielle.
5. "Validation des performances"
   Dans cette section, nous allons examiner la validation des performances des bases de données vectorielles, telles que Milvus et pgvecto.RS, en nous concentrant sur leurs caractéristiques de performances de stockage, telles que le profil d'E/S et le contrôleur de stockage NetApp qui prennent en charge les charges de travail RAG et d'inférence dans le cadre du cycle de vie LLM. Nous évaluerons et identifierons les différences de performances éventuelles lorsque ces bases de données sont combinées à la solution de stockage ONTAP. Notre analyse sera basée sur des indicateurs clés de performance, comme le nombre de requêtes traitées par seconde (QPS).

Configuration de clusters Milvus avec Kubernetes sur site

Configuration de clusters Milvus avec Kubernetes sur site

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

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

instances), pulsar-bookie-journal (trois instances), pulsar-bookie-ledgers (trois instances) et pulsar-zookeeper-data (trois instances).

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

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

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

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

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

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

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

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

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

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

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

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

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

12. Vérifier l'état des pods.

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

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

13. Testez l'écriture et la lecture de données dans Milvus et le stockage objet NetApp.
    - Écrivez les données à l'aide du programme Python « Prepare_Data_netapp_New.py ».

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

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

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

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

=== Start Creating index IVF_FLAT  ===

=== Start loading                  ===

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

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

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

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

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

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

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

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

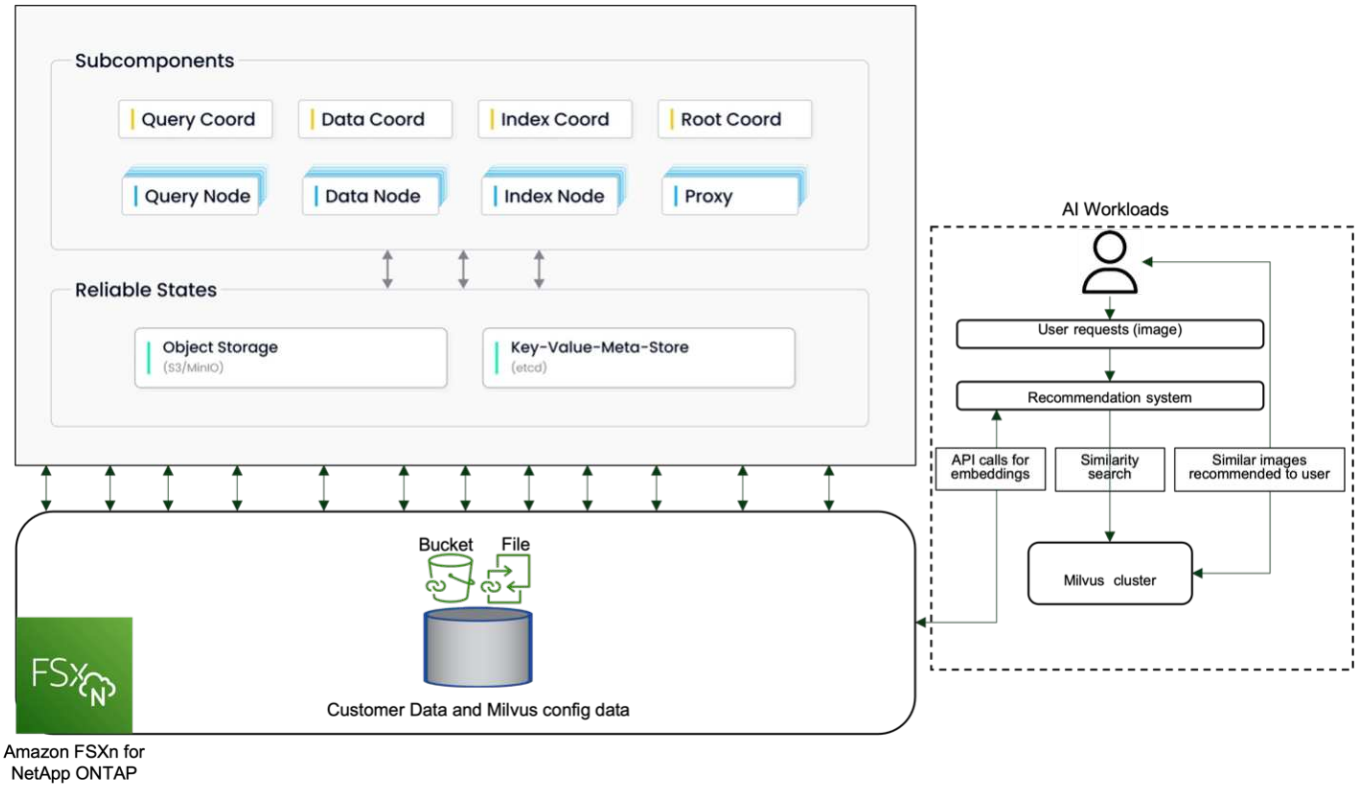
données. Le processus consistant à vérifier l'état des demandes de volume persistant et des pods, ainsi qu'à tester l'écriture et la lecture des données, garantit la fiabilité et la cohérence des opérations de données. L'utilisation du stockage objet ONTAP ou StorageGRID pour les données des clients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

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

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

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

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



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

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

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

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

7. Dans la section « volumes » du fichier docker-compose.yml, mappez le point de montage NFS NetApp sur le chemin du conteneur Milvus correspondant, en particulier dans etcd, minio et standalone.Check "Annexe D : docker-compose.yml" pour plus de détails sur les modifications de 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. Vérifier l'état du conteneur Milvus.

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

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

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

```
/448789845791611912/448789845791611913/448789845791611939/103/448789845791411923/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 (en lecture) aux données NFS testées dans la base de données vectorielle via le protocole S3 pour les workloads d'IA, cela peut être validé à l'aide d'un programme Python simple. Un exemple de ceci pourrait être une recherche de similarité d'images d'une autre application comme mentionné dans l'image qui se trouve au début de cette section.

```
root@ip-172-31-29-98:~/pymilvus/examples# sudo python3
/home/ubuntu/milvusvectordb/s3_access.py -i 172.31.255.228 --bucket
milvusnasvol --access-key PY6UF318996I86NBYNDD --secret-key
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 explique de manière efficace comment les clients peuvent déployer et exploiter une configuration Milvus autonome dans des conteneurs Docker en utilisant NetApp FSxN pour le stockage de données NetApp ONTAP d'Amazon. Cette configuration permet aux clients d'exploiter la puissance des bases de données vectorielles pour gérer les données de grande envergure et exécuter des requêtes complexes, le tout dans l'environnement évolutif et efficace de conteneurs Docker. En créant une instance Amazon FSxN pour NetApp ONTAP et en faisant correspondre l'instance EC2, les clients peuvent assurer une utilisation optimale des ressources et une gestion optimale des données. La validation réussie des opérations d'écriture et de lecture de données de FSxN dans la base de données vectorielle offre aux clients la garantie d'opérations de données fiables et cohérentes. En outre, la possibilité de lister (lire) des données de workloads d'IA via le protocole S3 permet d'améliorer l'accessibilité des données. Ce processus complet offre donc aux clients une solution robuste et efficace pour la gestion de leurs opérations de données à grande échelle, exploitant les fonctionnalités de FSxN pour NetApp ONTAP d'Amazon.

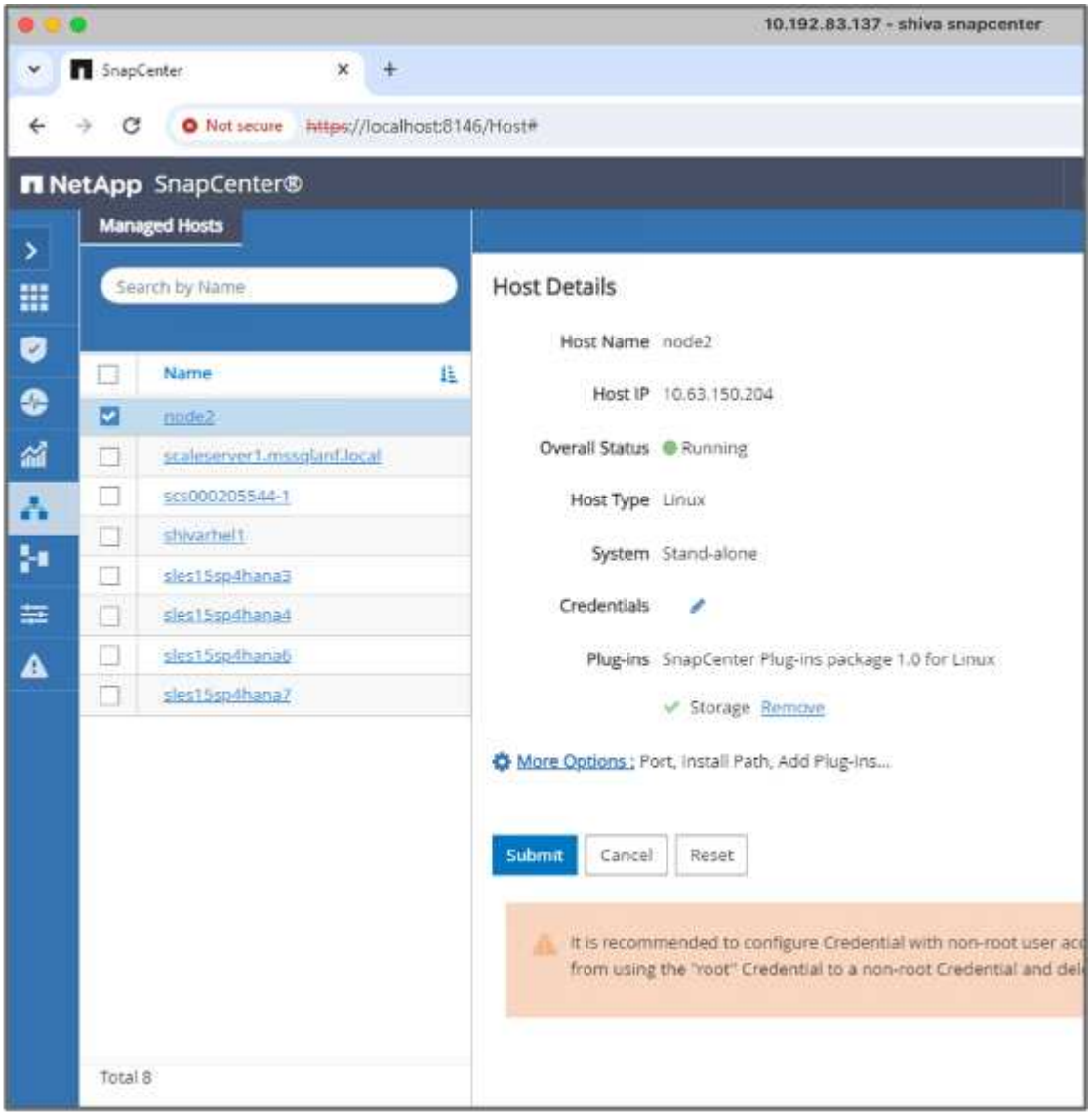
Protection de base de données Vector à l'aide de SnapCenter

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

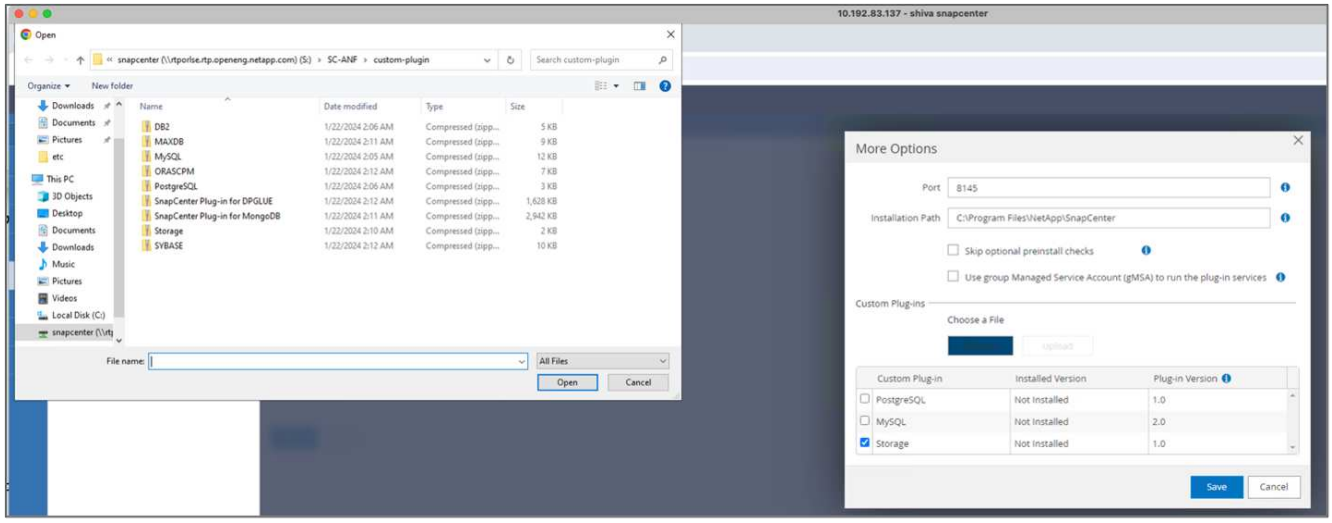
Par exemple, dans le secteur de la production cinématographique, les clients possèdent souvent des données

intégrées essentielles, telles que des fichiers audio et vidéo. La perte de ces données, due à des problèmes tels que les pannes de disque dur, peut avoir un impact significatif sur leurs opérations, ce qui risque de mettre en péril des projets à plusieurs millions de dollars. Nous avons rencontré des cas où du contenu inestimable a été perdu, causant des perturbations importantes et des pertes financières. La sécurité et l'intégrité de ces données essentielles sont donc d'une importance capitale dans ce secteur.
Dans cette section, nous allons examiner comment SnapCenter protège les données de la base de données vectorielle et les données Milvus résidant dans ONTAP. Dans cet exemple, nous avons utilisé un compartiment NAS (milvusdbvol1) dérivé d'un volume ONTAP NFS (vol1) pour les données clients, et un volume NFS distinct (vectordbpv) pour les données de configuration du cluster Milvus. veuillez vérifier le "ici" pour le flux de travail de sauvegarde SnapCenter

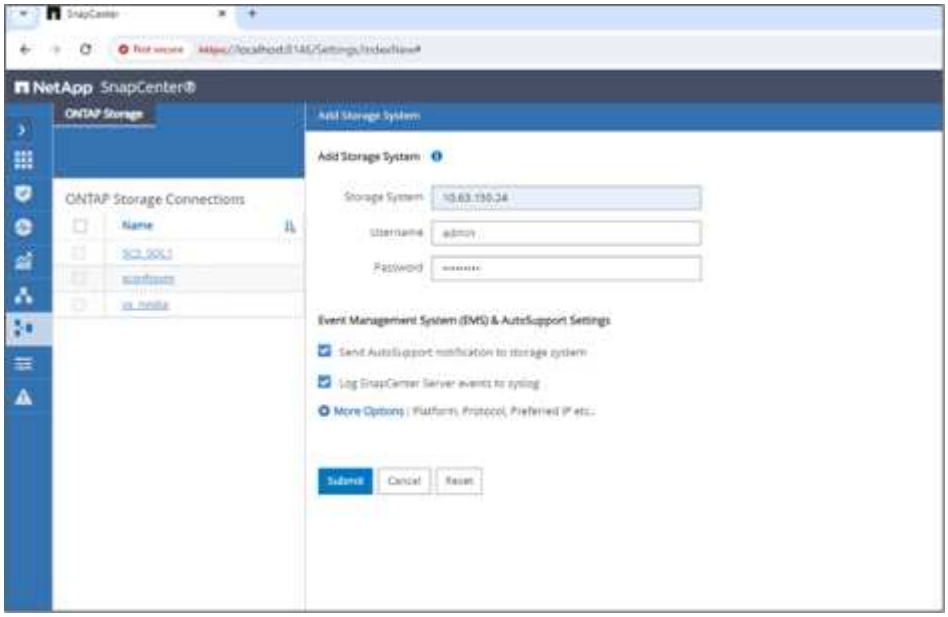
1. Configurez l'hôte qui sera utilisé pour exécuter les commandes SnapCenter.



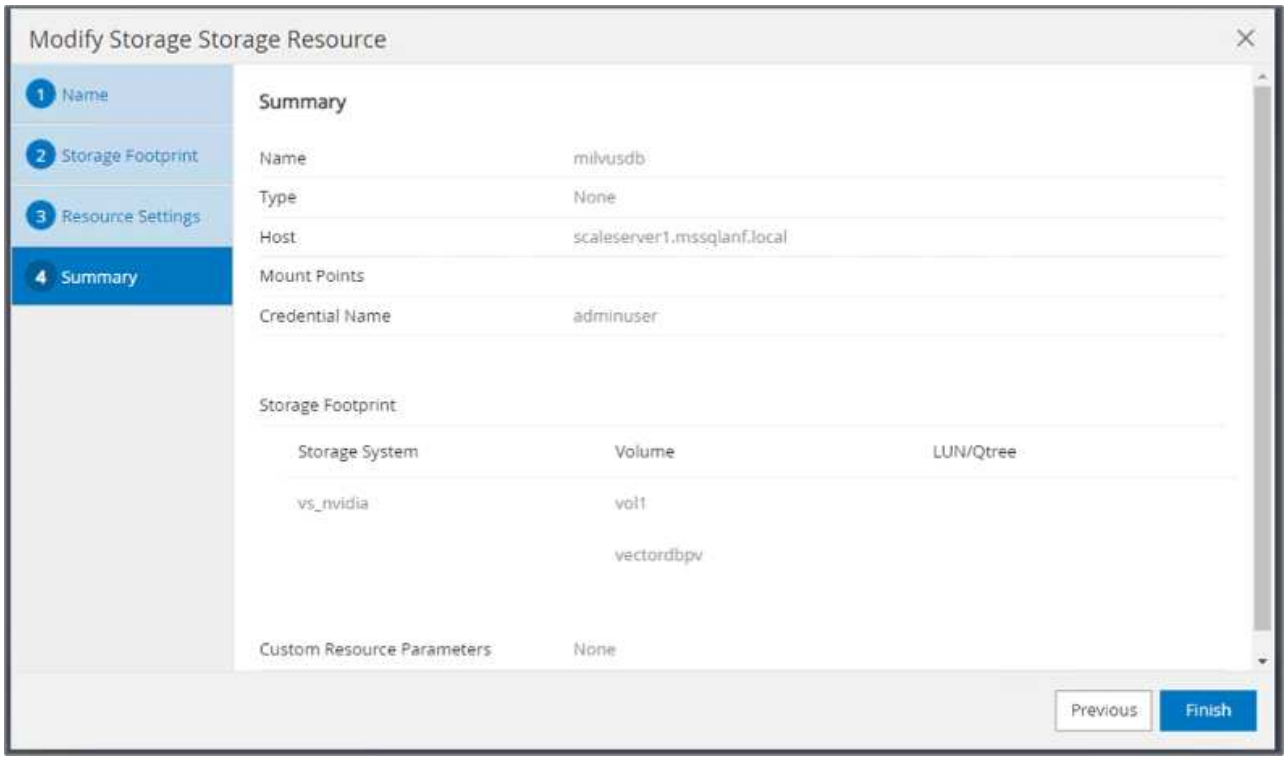
2. Installez et configurez le plug-in de stockage. Dans l'hôte ajouté, sélectionnez « autres options ». Accédez au plug-in de stockage téléchargé et sélectionnez-le dans le "Le site NetApp Automation Store". Installez le plug-in et enregistrez la configuration.



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



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



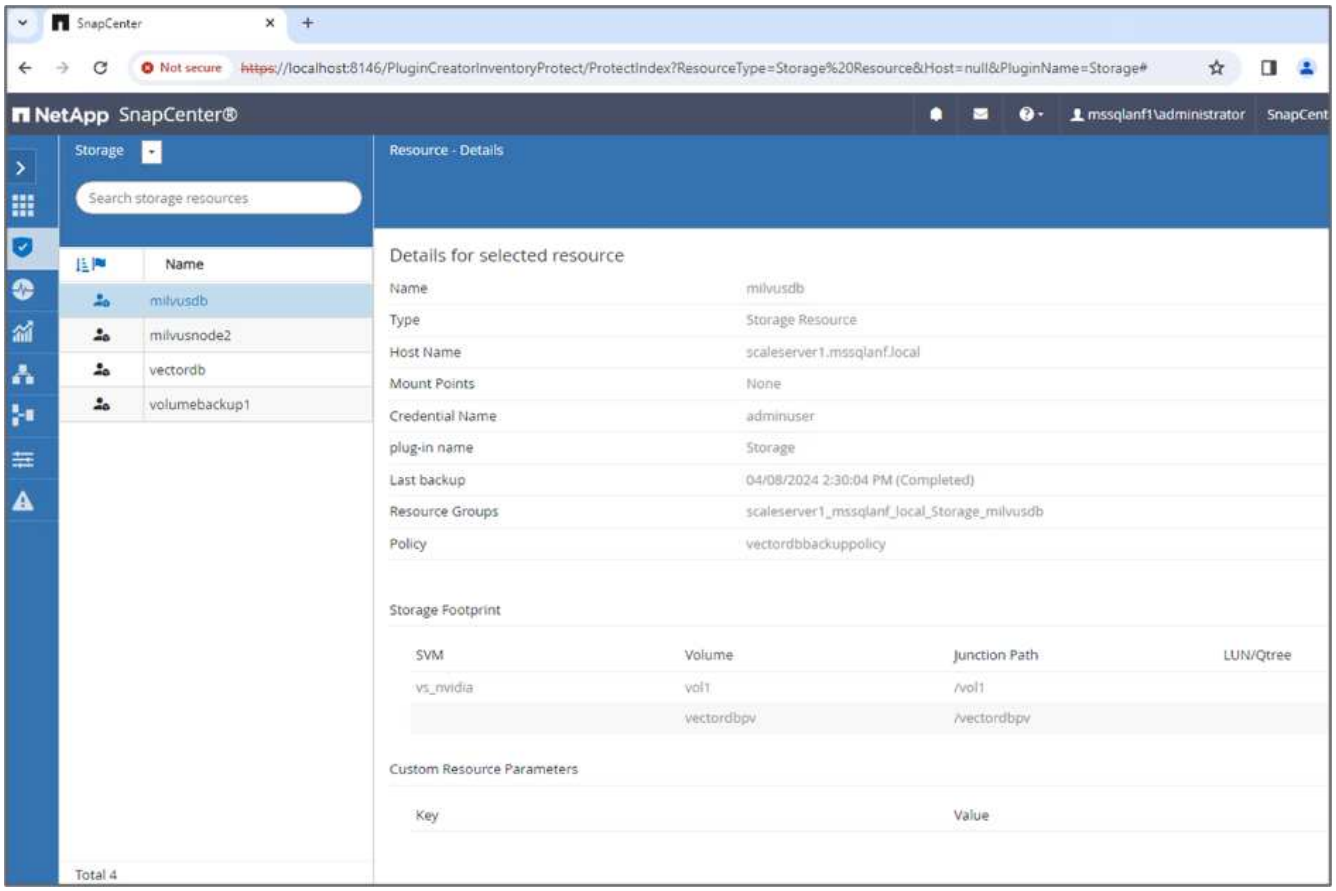
5. Insérez les données dans le compartiment NAS S3 à l'aide d'un script Python. Dans notre cas, nous avons modifié le script de sauvegarde fourni par Milvus, à savoir « prepare_data_netapp.py », et exécuté la commande « sync » pour vider les données du système d'exploitation.

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_test exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_test` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_test: 3000

=== Create collection `hello_milvus_netapp_sc_test2` ===

Number of entities in hello_milvus_netapp_sc_test2: 6000
root@node2:~# for i in 2 3 4 5 6   ; do ssh node$i "hostname; sync; echo
'sync executed';" ; done
node2
sync executed
node3
sync executed
node4
sync executed
node5
sync executed
node6
sync executed
root@node2:~#
```

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

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

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

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

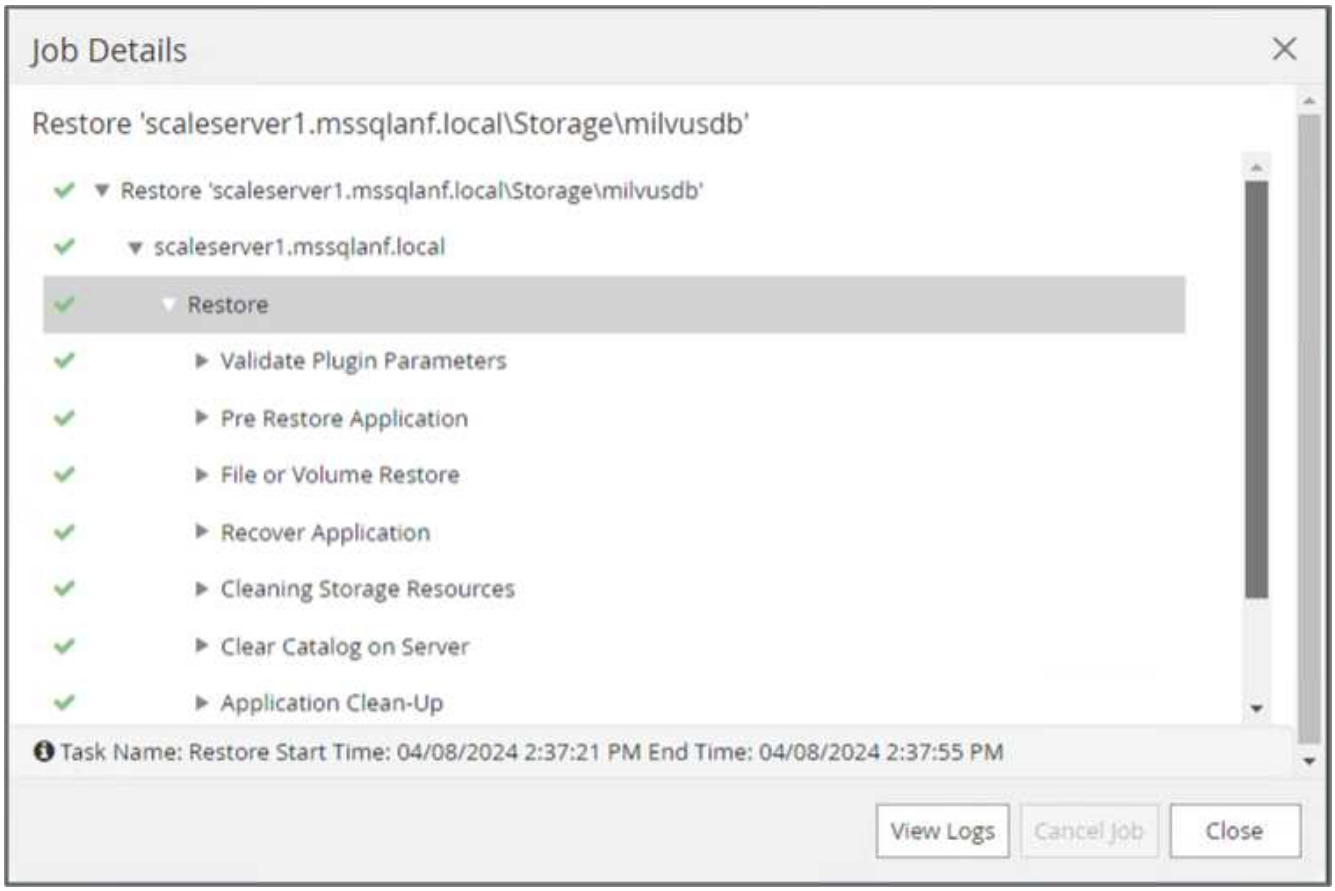


8. Pour tester la fonctionnalité de sauvegarde, nous avons soit ajouté un nouveau tableau après le processus de sauvegarde, soit supprimé certaines données du NFS (compartiment NAS S3).

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

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False

=== Create collection `hello_milvus_netapp_sc_testnew` ===


=== Start inserting entities       ===

Number of entities in hello_milvus_netapp_sc_testnew: 3000

=== Create collection `hello_milvus_netapp_sc_testnew2` ===

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

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



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

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


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

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

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

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

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

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

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

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

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

=== start connecting to Milvus     ===


=== Milvus host: localhost         ===

Does collection hello_milvus_netapp_sc_testnew exist in Milvus: False
Traceback (most recent call last):
  File "/root/verify_data_netapp.py", line 37, in <module>
    recover_collection = Collection(recover_collection_name)
  File "/usr/local/lib/python3.10/dist-
packages/pymilvus/orm/collection.py", line 137, in __init__
    raise SchemaNotReadyException(
pymilvus.exceptions.SchemaNotReadyException: <SchemaNotReadyException:
(code=1, message=Collection 'hello_milvus_netapp_sc_testnew' not exist,
or you can pass in schema to create one.)>
root@node2:~#
```

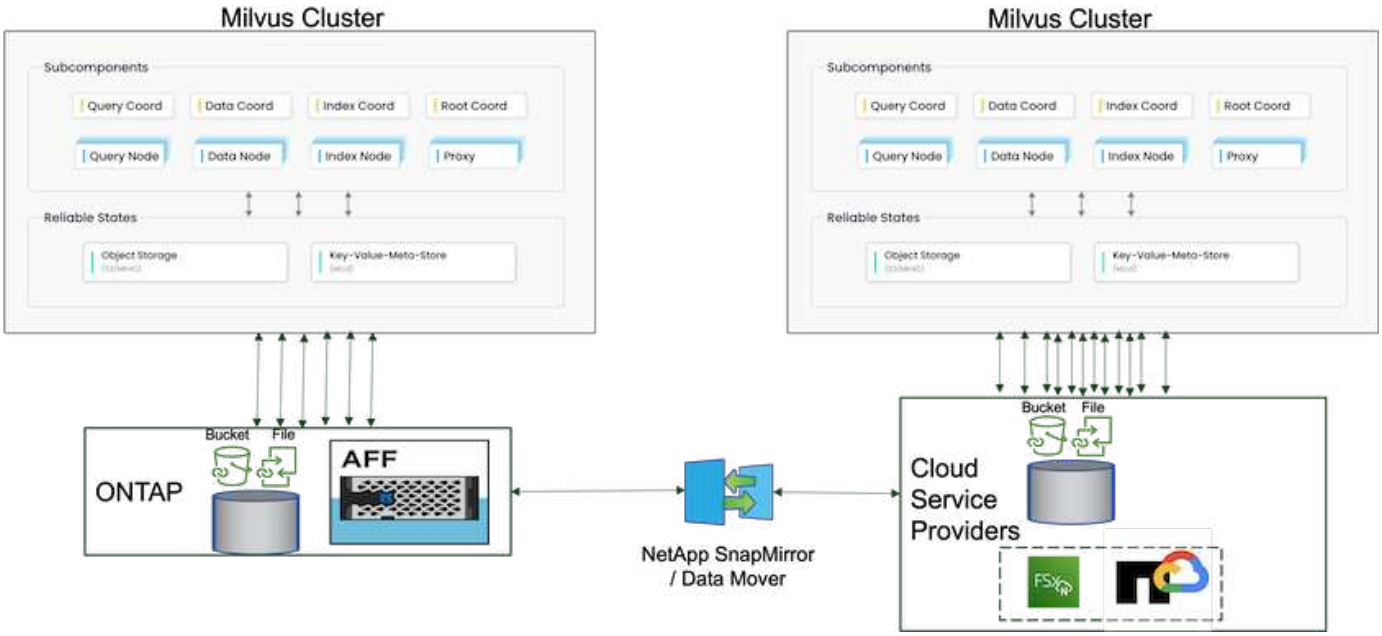
En conclusion, l'utilisation de SnapCenter de NetApp pour protéger les données de base de données Vector et les données Milvus résidant dans ONTAP offre des avantages considérables aux clients, en particulier dans les secteurs où l'intégrité des données est primordiale, tels que la production cinématographique.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

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

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

Reprise sur incident à l'aide de NetApp SnapMirror

Reprise sur incident à l'aide de NetApp SnapMirror



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

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

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

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

puissant et d'opérations moins intensives dans le cloud.

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

Validation des performances de la base de données Vector

Validation des performances

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

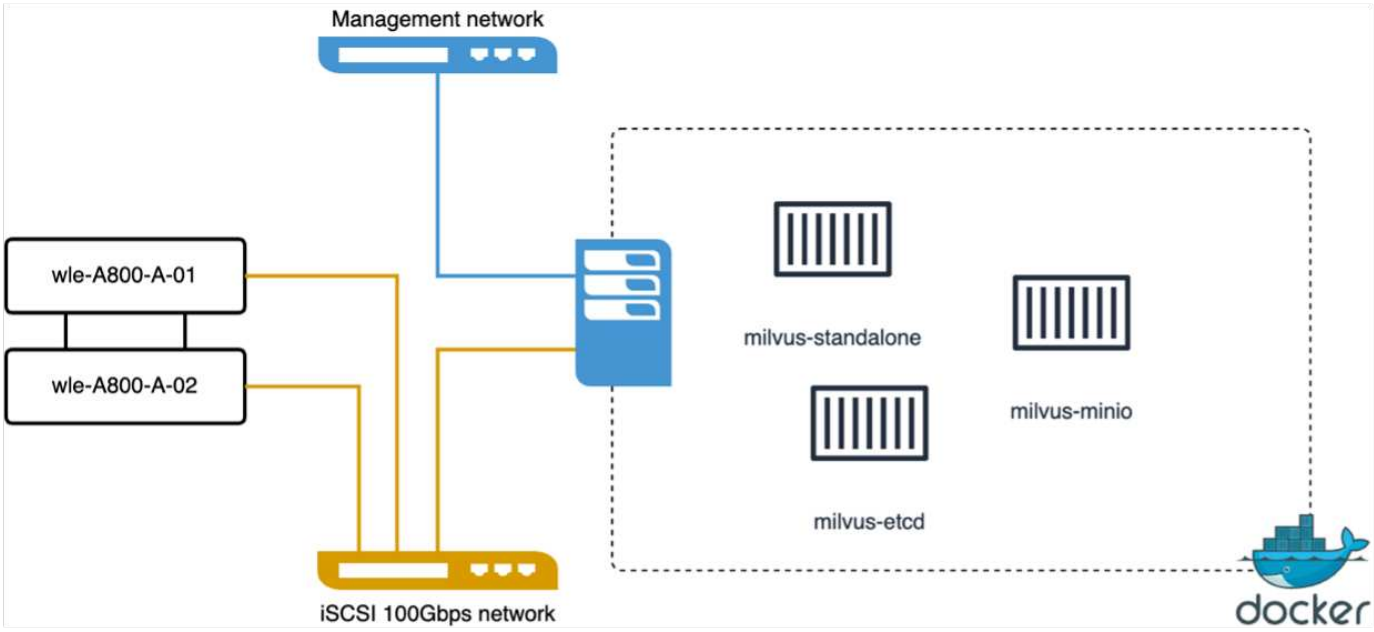
Dans cette section, nous allons examiner la validation des performances des bases de données vectorielles, telles que Milvus et pgvecto.RS, en nous concentrant sur leurs caractéristiques de performances de stockage, telles que le profil d'E/S et le contrôleur de stockage NetApp qui prennent en charge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

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

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

VectorDB-Bench avec cluster autonome Milvus

Nous avons effectué la validation de performances suivante sur un cluster autonome milvus avec vectorDB-Bench.
La connectivité réseau et serveur du cluster autonome milvus est ci-dessous.



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

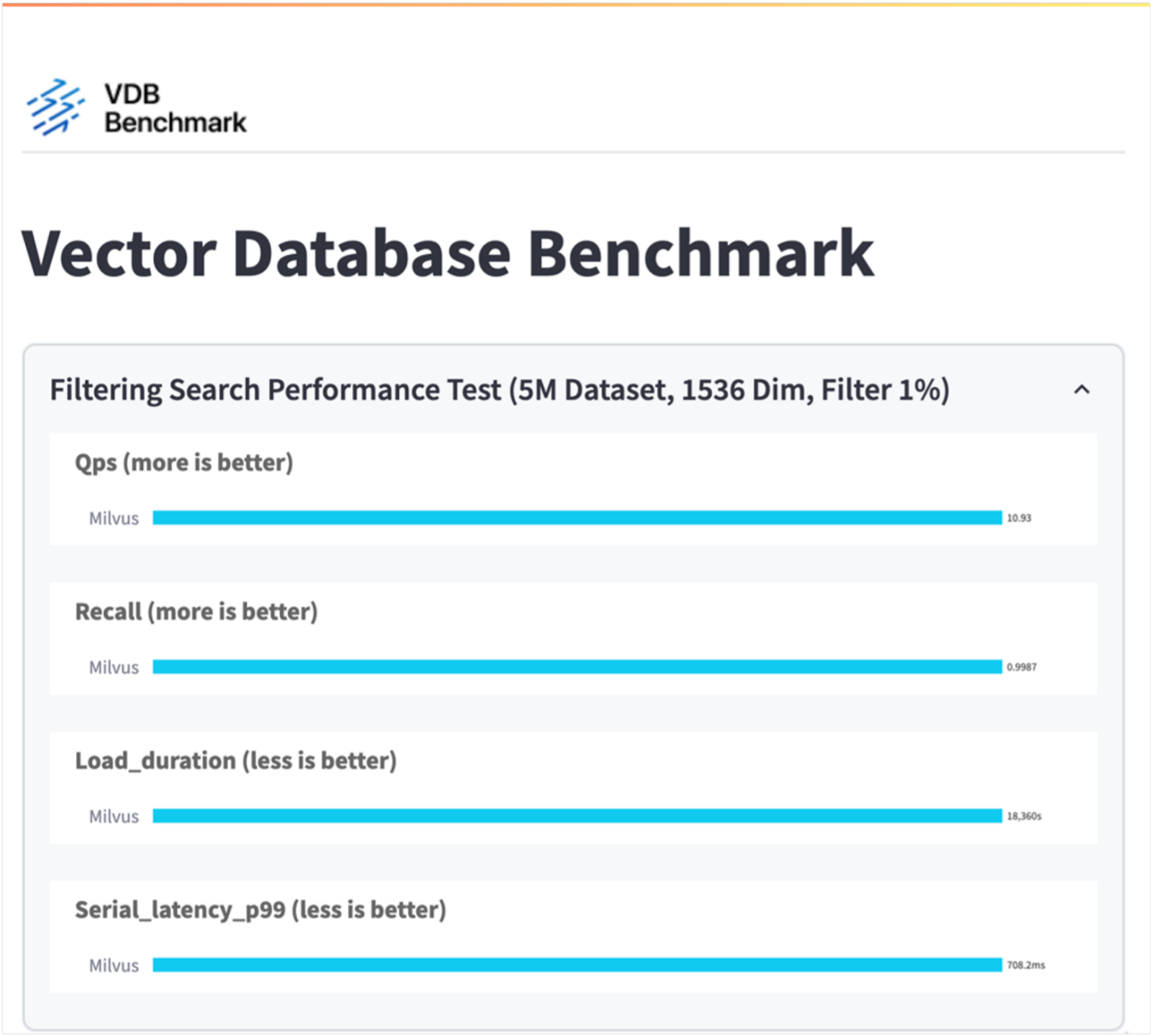
Du point de vue du stockage, la base de données a émis environ 1,000 000 opérations/s durant les phases d'ingestion, d'optimisation après insertion et de création d'index. Dans la phase de requête, il a demandé 32,000 opérations/s.

La section suivante présente les mesures de performances du stockage.

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

Phase du workload	Métrique	Valeur
	Latence	< 400 utilisateurs
	Charge de travail	100 % de lectures mises en cache
	Taille des E/S.	Principalement 8 Ko

Le résultat vectorDB-Bench est inférieur à.



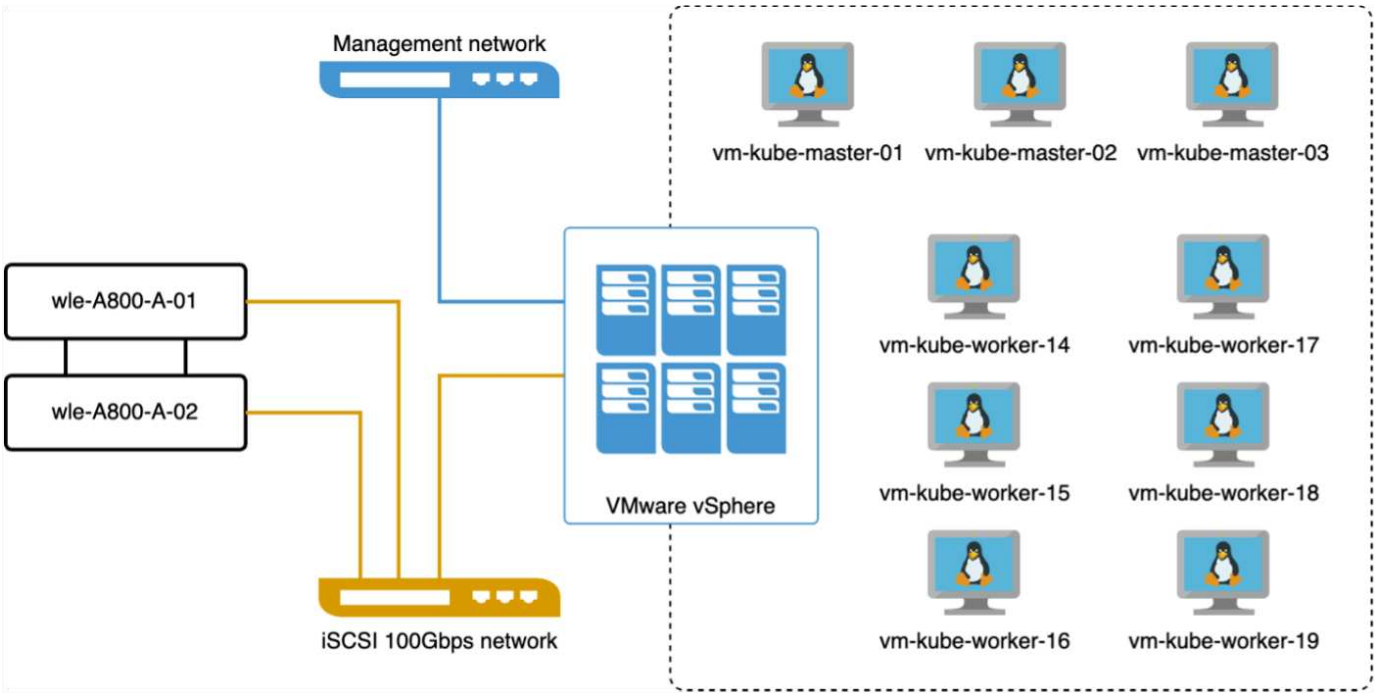
Depuis la validation des performances de l'instance Milvus autonome, il est évident que la configuration actuelle est insuffisante pour prendre en charge un jeu de données de 5 millions de vecteurs avec une dimension de 1536. nous avons déterminé que le stockage possède les ressources adéquates et ne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

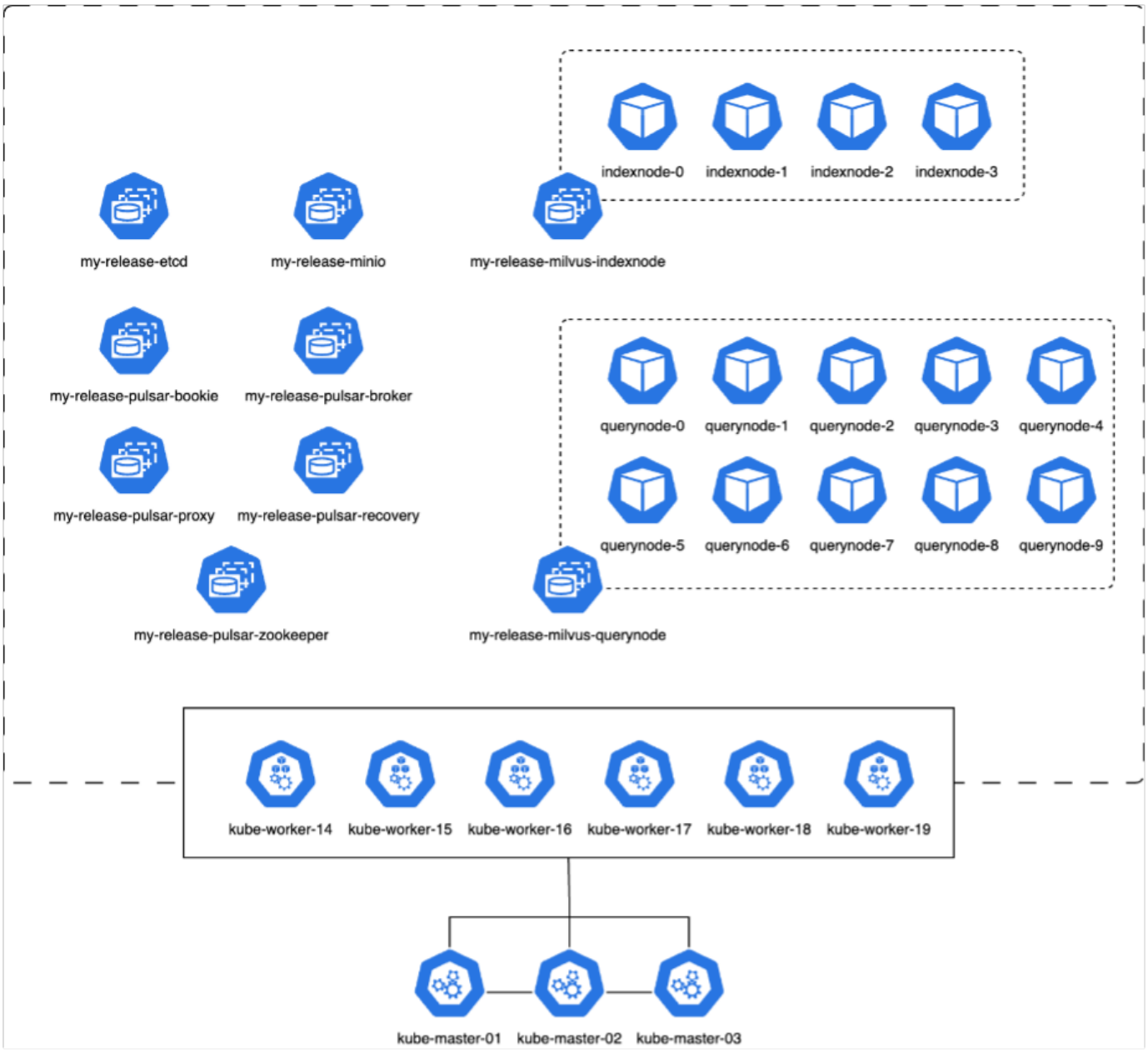
VectorDB-Bench avec cluster Milvus

Cette section traite du déploiement d'un cluster Milvus dans un environnement Kubernetes. Cette configuration

Kubernetes a été construite sur un déploiement VMware vSphere, qui héberge les nœuds maîtres et workers Kubernetes.

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





Dans cette section, nous présentons nos observations et nos résultats des tests de la base de données Milvus.
* Le type d'index utilisé était DiskANN.
* Le tableau ci-dessous fournit une comparaison entre les déploiements autonomes et les déploiements en grappe lorsqu'ils travaillent avec 5 millions de vecteurs à une dimension de 1536. Nous avons observé que le temps nécessaire pour l'ingestion et l'optimisation après insertion des données était plus faible dans le déploiement du cluster. Au cours du déploiement du cluster, la latence du 99e centile pour les requêtes a été divisée par six par rapport à une configuration autonome.
* Bien que le taux de requêtes par seconde (QPS) ait été plus élevé dans le déploiement du cluster, il n'était pas au niveau souhaité.

Metric	Milvus Standalone	Milvus Cluster	Difference
QPS @ Recall	10.93 @ 0.9987	18.42 @ 0.9952	+40%
p99 Latency (less is better)	708.2 ms	117.6 ms	-83%
Load Duration time (less is better)	18,360 secs	12,730 secs	-30%

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

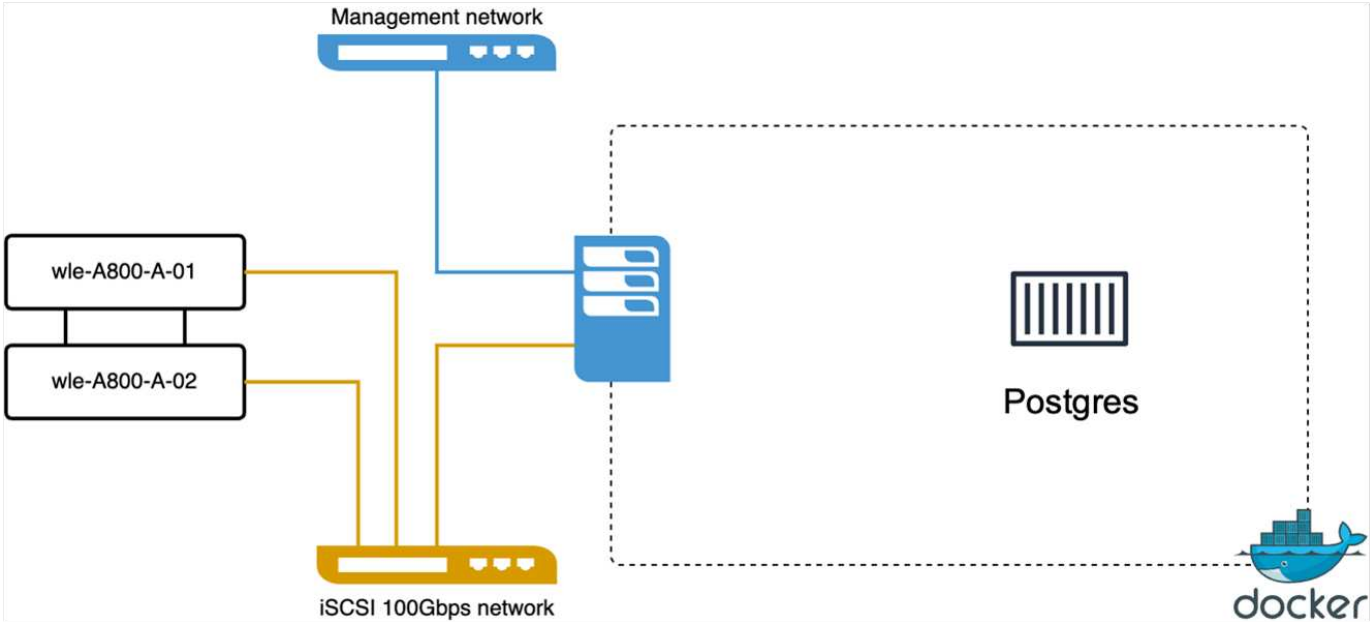
La section suivante présente les principaux metrics de performance du stockage.

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

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

VectorDB-Bench avec Postgres (pgvecto.RS)

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

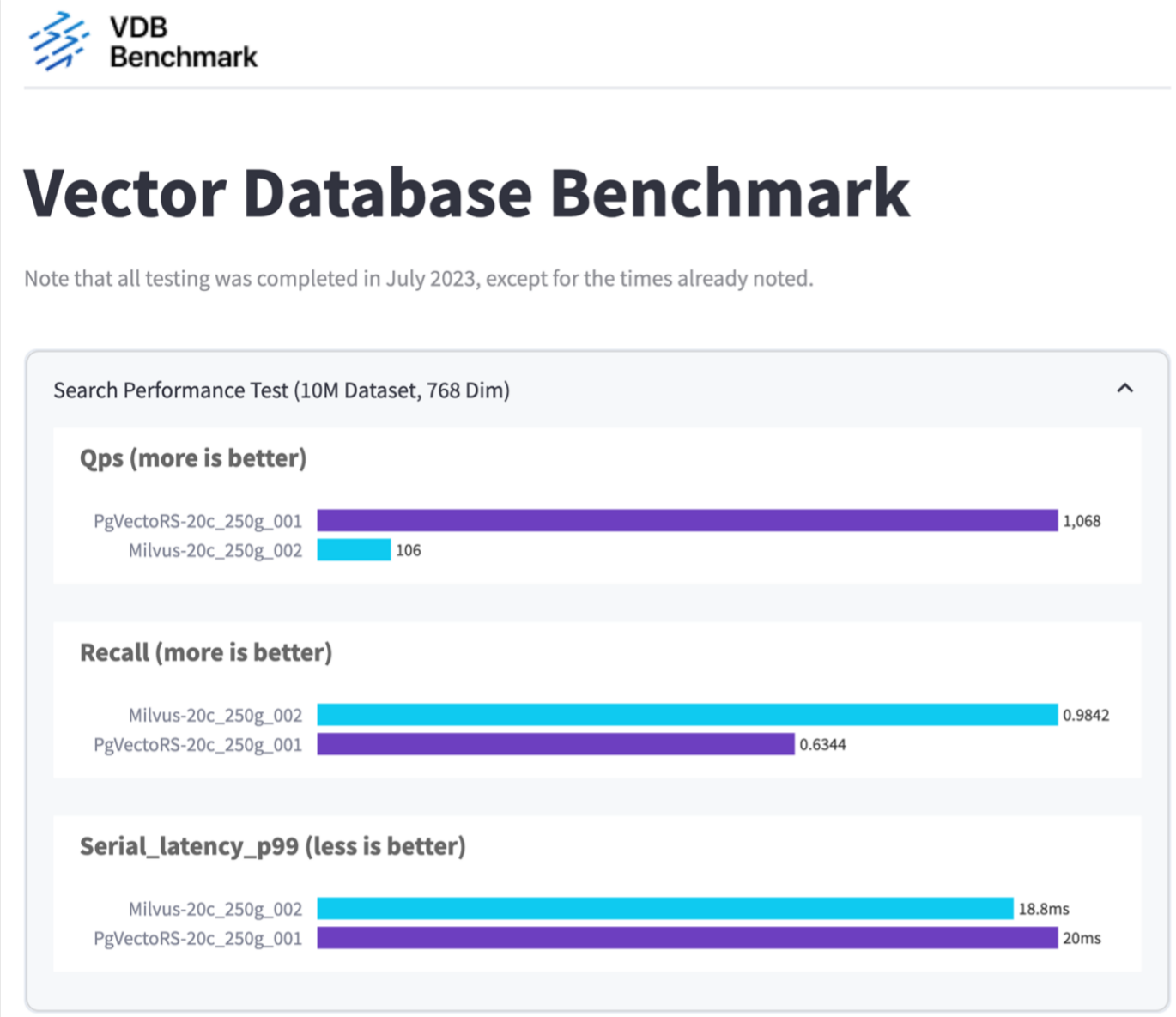


Dans cette section, nous partageons nos observations et nos résultats des tests de la base de données PostgreSQL, en particulier à l'aide de pgvecto.RS.
* Nous avons choisi HNSW comme type d'index pour ces tests parce qu'au moment des tests, DiskANN n'était pas disponible pour pgvecto.RS.
* Pendant la phase d'ingestion des données, nous avons chargé le jeu de données de Cohere, qui se compose de 10 millions de vecteurs à une dimension de 768. Ce processus a pris environ 4.5 heures.
* Dans la phase de requête, nous avons observé un taux de requêtes par seconde (QPS) de 1,068 avec un rappel de 0.6344. La latence du 99e centile pour les requêtes a été mesurée à 20 millisecondes. Pendant la majeure partie de l'exécution, le CPU client fonctionnait à 100 % de sa capacité.

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


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

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



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

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

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

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

Observation basée sur nos validations de performances :

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

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

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

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

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

Informations sur le copyright

Informations sur les marques commerciales