Charges de travail Apache Kafka avec stockage NetApp NFS

NetApp artificial intelligence solutions

NetApp
August 18, 2025

Sommaire

Charges de travail Apache Kafka avec stockage NetApp NFS . . . 1
TR-4947 : Charge de travail Apache Kafka avec stockage NetApp NFS – Validation fonctionnelle et performances . . . 1
Pourquoi utiliser le stockage NFS pour les charges de travail Kafka ? . . . 1
Pourquoi NetApp pour les charges de travail Kafka ? . . . 2
Solution NetApp pour le problème de renommage stupide des charges de travail NFS vers Kafka . . . 2
Validation fonctionnelle - Correction d’un changement de nom idiot . . . 3
Configuration de validation . . . 3
Flux architectural . . . 4
Méthodologie des tests . . . 4
Pourquoi NetApp NFS pour les charges de travail Kafka ? . . . 8
Utilisation réduite du processeur sur le courtier Kafka . . . 8
Récupération plus rapide des courtiers . . . 13
Efficacité du stockage . . . 17
Aperçu des performances et validation dans AWS . . . 20
Kafka dans le cloud AWS avec NetApp Cloud Volumes ONTAP (paire haute disponibilité et nœud unique) . . . 20
Méthodologie des tests . . . 31
Observation . . . 31
Présentation et validation des performances dans AWS FSx ONTAP . . . 33
Apache Kafka dans AWS FSx ONTAP . . . 34
Aperçu des performances et validation avec AFF A900 sur site . . . 41
Configuration de stockage . . . 42
Réglage du client . . . 42
Réglage du courtier Kafka . . . 42
Méthodologie de test du générateur de charge de travail . . . 43
Performances extrêmes et exploration des limites de stockage . . . 46
Guide de dimensionnement . . . 47
Conclusion . . . 48
Où trouver des informations supplémentaires . . . 48

Charges de travail Apache Kafka avec stockage NetApp NFS

TR-4947 : Charge de travail Apache Kafka avec stockage NetApp NFS – Validation fonctionnelle et performances

Shantanu Chakole, Karthikeyan Nagalingam et Joe Scott, NetApp

Kafka est un système de messagerie de publication-abonnement distribué avec une file d'attente robuste qui peut accepter de grandes quantités de données de message. Avec Kafka, les applications peuvent écrire et lire des données sur des sujets de manière très rapide. En raison de sa tolérance aux pannes et de son évolutivité, Kafka est souvent utilisé dans l'espace Big Data comme un moyen fiable d'ingérer et de déplacer de nombreux flux de données très rapidement. Les cas d'utilisation incluent le traitement des flux, le suivi de l'activité du site Web, la collecte et la surveillance des mesures, l'agrégation des journaux, les analyses en temps réel, etc.

Bien que les opérations Kafka normales sur NFS fonctionnent bien, le problème de renommage stupide fait planter l'application lors du redimensionnement ou du repartitionnement d'un cluster Kafka exécuté sur NFS. Il s'agit d'un problème important car un cluster Kafka doit être redimensionné ou repartitionné à des fins d'équilibrage de charge ou de maintenance. Vous pouvez trouver des détails supplémentaires "ici" .

Ce document décrit les sujets suivants :

- Le problème du changement de nom idiot et la validation de la solution
- Réduire l'utilisation du processeur pour réduire le temps d'attente des E/S
- Temps de récupération plus rapide du courtier Kafka
- Performances dans le cloud et sur site

Pourquoi utiliser le stockage NFS pour les charges de travail Kafka ?

Les charges de travail Kafka dans les applications de production peuvent diffuser d'énormes quantités de données entre les applications. Ces données sont conservées et stockées dans les nœuds du courtier Kafka dans le cluster Kafka. Kafka est également connu pour sa disponibilité et son parallélisme, qu'il obtient en divisant les sujets en partitions, puis en répliquant ces partitions dans tout le cluster. Cela signifie finalement que l'énorme quantité de données qui circule dans un cluster Kafka est généralement multipliée en taille. NFS permet de rééquilibrer les données en fonction des changements du nombre de courtiers très rapidement et facilement. Pour les environnements de grande taille, le rééquilibrage des données sur DAS lorsque le nombre de courtiers change prend beaucoup de temps et, dans la plupart des environnements Kafka, le nombre de courtiers change fréquemment.

Les autres avantages comprennent les suivants :

- **Maturité.** NFS est un protocole mature, ce qui signifie que la plupart des aspects de sa mise en œuvre, de sa sécurisation et de son utilisation sont bien compris.
- **Ouvrir.** NFS est un protocole ouvert et son développement continu est documenté dans les spécifications Internet en tant que protocole réseau libre et ouvert.

- **Rentable.** NFS est une solution économique de partage de fichiers en réseau, facile à configurer car elle utilise l'infrastructure réseau existante.
- **Gestion centralisée.** La gestion centralisée de NFS réduit le besoin de logiciels et d'espace disque supplémentaires sur les systèmes utilisateur individuels.
- **Distribué.** NFS peut être utilisé comme système de fichiers distribué, réduisant ainsi le besoin de périphériques de stockage amovibles.

Pourquoi NetApp pour les charges de travail Kafka ?

L'implémentation NetApp NFS est considérée comme une référence absolue pour le protocole et est utilisée dans d'innombrables environnements NAS d'entreprise. Outre la crédibilité de NetApp, il offre également les avantages suivants :

- Fiabilité et efficacité
- Évolutivité et performance
- Haute disponibilité (partenaire HA dans un cluster NetApp ONTAP)
- Protection des données
  - **Reprise après sinistre (NetApp SnapMirror).** Votre site tombe en panne ou vous souhaitez démarrer sur un autre site et reprendre là où vous vous êtes arrêté.
  - Facilité de gestion de votre système de stockage (administration et gestion via NetApp OnCommand).
  - **Équilibrage de charge.** Le cluster vous permet d'accéder à différents volumes à partir de LIF de données hébergés sur différents nœuds.
  - **Opérations non perturbatrices.** Les LIF ou les déplacements de volumes sont transparents pour les clients NFS.

Solution NetApp pour le problème de renommage stupide des charges de travail NFS vers Kafka

Kafka est construit avec l'hypothèse que le système de fichiers sous-jacent est compatible POSIX : par exemple, XFS ou Ext4. Le rééquilibrage des ressources Kafka supprime les fichiers pendant que l'application les utilise encore. Un système de fichiers compatible POSIX permet de procéder à la dissociation. Cependant, il ne supprime le fichier qu'une fois que toutes les références au fichier ont disparu. Si le système de fichiers sous-jacent est connecté au réseau, le client NFS intercepte les appels de dissociation et gère le flux de travail. Étant donné qu'il existe des ouvertures en attente sur le fichier en cours de dissociation, le client NFS envoie une demande de changement de nom au serveur NFS et, lors de la dernière fermeture du fichier dissocié, émet une opération de suppression sur le fichier renommé. Ce comportement est communément appelé renommage idiot NFS et il est orchestré par le client NFS.

Tout courtier Kafka utilisant le stockage d'un serveur NFSv3 rencontre des problèmes en raison de ce comportement. Cependant, le protocole NFSv4.x dispose de fonctionnalités permettant de résoudre ce problème en permettant au serveur d'assumer la responsabilité des fichiers ouverts et non liés. Les serveurs NFS prenant en charge cette fonctionnalité facultative communiquent la capacité de propriété au client NFS au moment de l'ouverture du fichier. Le client NFS cesse alors la gestion de la dissociation lorsqu'il y a des ouvertures en attente et permet au serveur de gérer le flux. Bien que la spécification NFSv4 fournisse des

directives pour la mise en œuvre, jusqu'à présent, il n'existait aucune implémentation de serveur NFS connue prenant en charge cette fonctionnalité facultative.

Les modifications suivantes sont nécessaires pour que le serveur NFS et le client NFS résolvent le problème de renommage stupide :

- **Modifications apportées au client NFS (Linux).** Au moment de l'ouverture du fichier, le serveur NFS répond avec un indicateur, indiquant la capacité à gérer la dissociation des fichiers ouverts. Les modifications côté client NFS permettent au serveur NFS de gérer la dissociation en présence de l'indicateur. NetApp a mis à jour le client NFS Linux open source avec ces modifications. Le client NFS mis à jour est désormais généralement disponible dans RHEL8.7 et RHEL9.1.
- **Modifications apportées au serveur NFS.** Le serveur NFS garde une trace des ouvertures. La dissociation d'un fichier ouvert existant est désormais gérée par le serveur pour correspondre à la sémantique POSIX. Lorsque la dernière ouverture est fermée, le serveur NFS lance alors la suppression effective du fichier et évite ainsi le processus de renommage stupide. Le serveur ONTAP NFS a implémenté cette capacité dans sa dernière version, ONTAP 9.12.1.

Grâce aux modifications ci-dessus apportées au client et au serveur NFS, Kafka peut profiter en toute sécurité de tous les avantages du stockage NFS connecté au réseau.

Validation fonctionnelle - Correction d'un changement de nom idiot

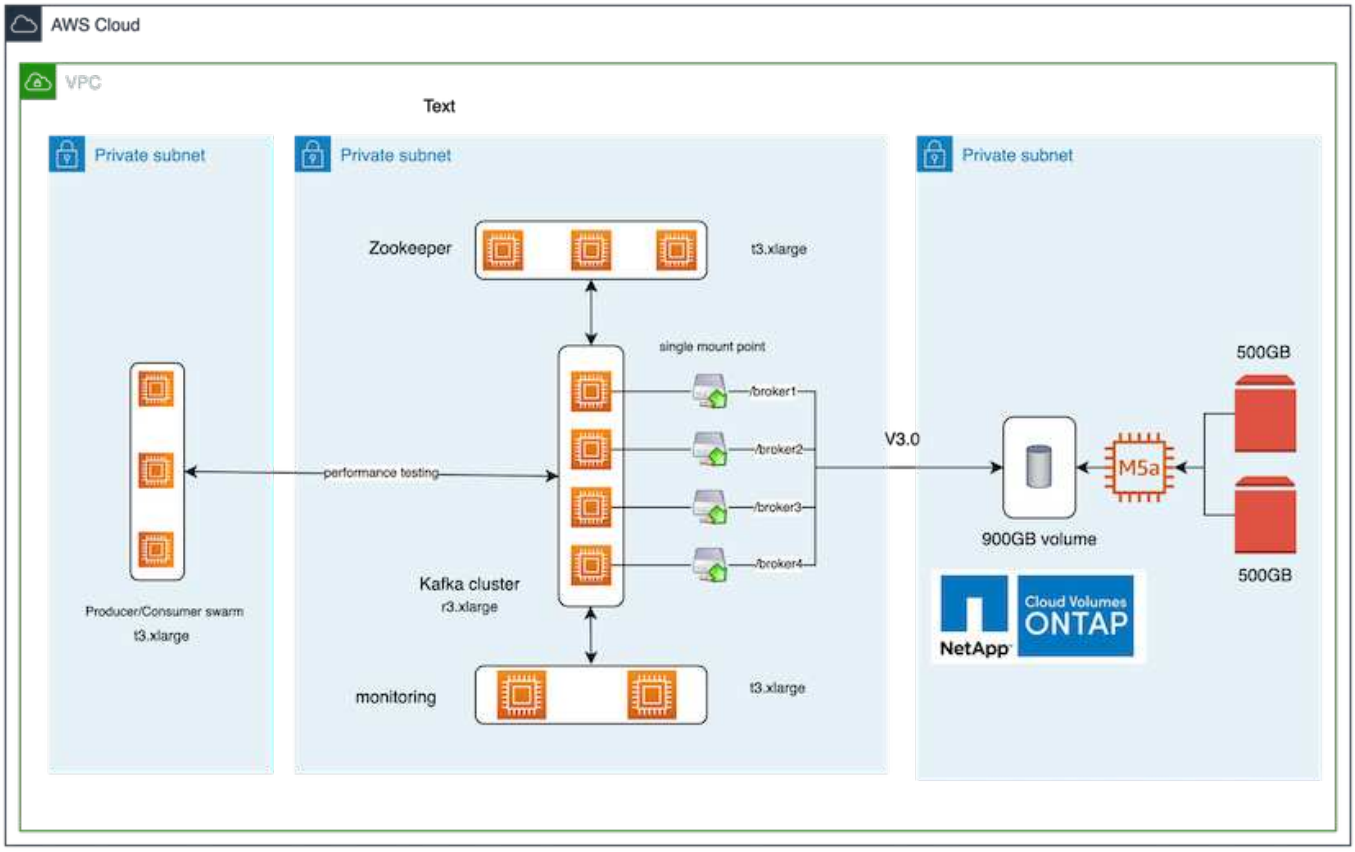
Pour la validation fonctionnelle, nous avons montré qu'un cluster Kafka avec un montage NFSv3 pour le stockage ne parvient pas à effectuer des opérations Kafka comme la redistribution de partition, alors qu'un autre cluster monté sur NFSv4 avec le correctif peut effectuer les mêmes opérations sans aucune interruption.

Configuration de validation

L'installation est exécutée sur AWS. Le tableau suivant présente les différents composants de la plateforme et la configuration environnementale utilisés pour la validation.

Composant de la plateforme	Configuration de l'environnement
Plateforme Confluent version 7.2.1	• 3 x gardiens de zoo – t3.xlarge • 4 serveurs courtiers – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x centre de contrôle – t3.xlarge • 3 x Producteur/consommateur
Système d'exploitation sur tous les nœuds	RHEL 8.7 ou version ultérieure
Instance NetApp Cloud Volumes ONTAP	Instance à nœud unique – M5.2xLarge

La figure suivante montre la configuration architecturale de cette solution.



Flux architectural

- **Calculer.** Nous avons utilisé un cluster Kafka à quatre nœuds avec un ensemble zookeeper à trois nœuds exécuté sur des serveurs dédiés.
- **Surveillance.** Nous avons utilisé deux nœuds pour une combinaison Prometheus-Grafana.
- **Charge de travail.** Pour générer des charges de travail, nous avons utilisé un cluster à trois nœuds distinct qui peut produire et consommer à partir de ce cluster Kafka.
- **Stockage.** Nous avons utilisé une instance ONTAP de volumes NetApp Cloud à nœud unique avec deux volumes AWS-EBS GP2 de 500 Go attachés à l'instance. Ces volumes ont ensuite été exposés au cluster Kafka en tant que volume NFSv4.1 unique via un LIF.

Les propriétés par défaut de Kafka ont été choisies pour tous les serveurs. La même chose a été faite pour l'essaim de gardiens de zoo.

Méthodologie des tests

1. Mise à jour `-is-preserve-unlink-enabled true` au volume Kafka, comme suit :

```
aws-shantanclastrecall-aws::*> volume create -vserver kafka_svm -volume
kafka_fg_vol01 -aggregate kafka_aggr -size 3500GB -state online -policy
kafka_policy -security-style unix -unix-permissions 0777 -junction-path
/kafka_fg_vol01 -type RW -is-preserve-unlink-enabled true
[Job 32] Job succeeded: Successful
```

2. Deux clusters Kafka similaires ont été créés avec la différence suivante :
   - **Groupe 1.** Le serveur backend NFS v4.1 exécutant la version ONTAP 9.12.1 prête pour la production était hébergé par une instance NetApp CVO. RHEL 8.7/RHEL 9.1 ont été installés sur les courtiers.
   - **Groupe 2.** Le serveur NFS principal était un serveur Linux NFSv3 générique créé manuellement.
3. Un sujet de démonstration a été créé sur les deux clusters Kafka.

   Groupe 1 :

```
[root@ip-172-30-0-160 demo]# kafka-topics --bootstrap-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:90
92,172.30.0.123:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: 2ty29xfhQLq65HKsUQv-pg PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 4       Replicas: 4,1   Isr: 4,1        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 2       Replicas: 2,4   Isr: 2,4        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 3       Replicas: 3,2   Isr: 3,2        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 1       Replicas: 1,3   Isr: 1,3        Offline:
```

   Groupe 2 :

```
[root@ip-172-30-0-198 demo]# kafka-topics --bootstrap-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:90
92,172.30.0.204:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: AwQpsZTQShyeMIhaquCG3Q PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 2       Replicas: 2,3   Isr: 2,3        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 3       Replicas: 3,1   Isr: 3,1        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 1       Replicas: 1,4   Isr: 1,4        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 4       Replicas: 4,2   Isr: 4,2        Offline:
```

4. Les données ont été chargées dans ces rubriques nouvellement créées pour les deux clusters. Cela a été fait en utilisant la boîte à outils producer-perf-test fournie dans le package Kafka par défaut :

```
./kafka-producer-perf-test.sh --topic __a_demo_topic --throughput -1
--num-records 3000000 --record-size 1024 --producer-props acks=all
bootstrap.servers=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,
172.30.0.123:9092
```

5. Un contrôle de santé a été effectué pour broker-1 pour chacun des clusters à l'aide de telnet :
   - telnet `172.30.0.160 9092`
   - telnet `172.30.0.198 9092`

     Un contrôle de santé réussi pour les courtiers sur les deux clusters est présenté dans la capture d'écran suivante :

6. Pour déclencher la condition d'échec qui provoque le blocage des clusters Kafka utilisant des volumes de stockage NFSv3, nous avons lancé le processus de réaffectation de partition sur les deux clusters. La réaffectation des partitions a été effectuée à l'aide de `kafka-reassign-partitions.sh` . Le processus détaillé est le suivant :
   a. Pour réaffecter les partitions d'un sujet dans un cluster Kafka, nous avons généré la configuration de réaffectation proposée JSON (cela a été effectué pour les deux clusters).

```
kafka-reassign-partitions --bootstrap
-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,172.30.
0.123:9092 --broker-list "1,2,3,4" --topics-to-move-json-file
/tmp/topics.json --generate
```

   b. Le JSON de réaffectation généré a ensuite été enregistré dans `/tmp/reassignment- file.json` .
   c. Le processus de réaffectation de partition réel a été déclenché par la commande suivante :

```
kafka-reassign-partitions --bootstrap
-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:9092,172.30.
0.204:9092 --reassignment-json-file /tmp/reassignment-file.json
–execute
```

7. Quelques minutes après la fin de la réaffectation, un autre contrôle de santé sur les courtiers a montré que le cluster utilisant des volumes de stockage NFSv3 avait rencontré un problème de renommage stupide et s'était écrasé, tandis que le cluster 1 utilisant des volumes de stockage NetApp ONTAP NFSv4.1 avec le correctif a continué ses opérations sans aucune interruption.

```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

Connection closed by foreign host.
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.198 9092
Trying 172.30.0.198...
telnet: connect to address 172.30.0.198: Connection refused
telnet: Unable to connect to remote host
```

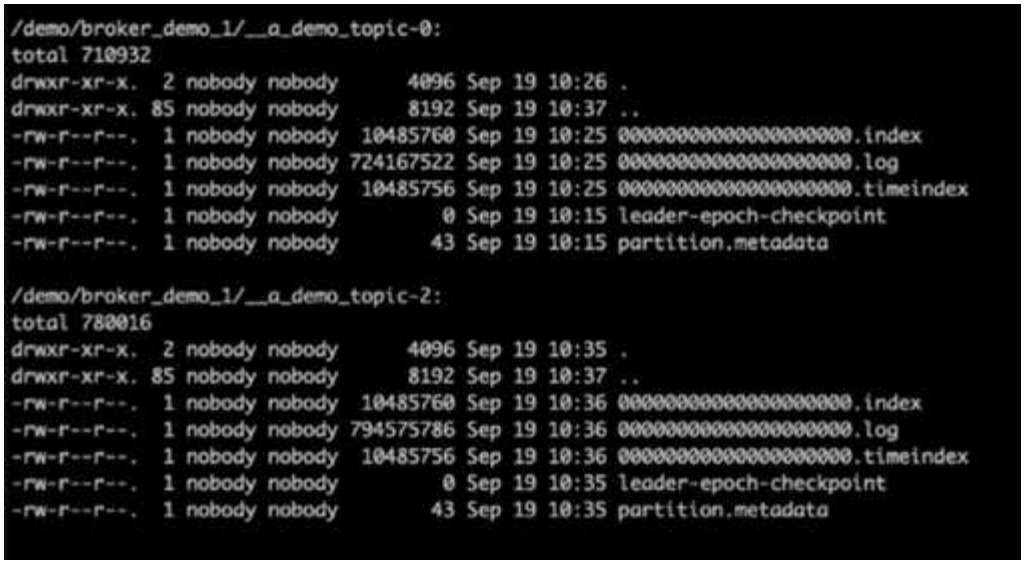
- Cluster1-Broker-1 est actif.
- Cluster2-broker-1 est mort.

8. Après avoir vérifié les répertoires de journaux Kafka, il était clair que le cluster 1 utilisant les volumes de stockage NetApp ONTAP NFSv4.1 avec le correctif avait une attribution de partition propre, tandis que le cluster 2 utilisant le stockage NFSv3 générique ne l'avait pas en raison de problèmes de renommage stupides, ce qui a conduit au crash. L'image suivante montre le rééquilibrage des partitions du cluster 2, qui a entraîné un problème de renommage stupide sur le stockage NFSv3.

```
/demo/broker_demo_1/__a_demo_topic-1.b31a8dd60fd443b283ffda2ecca9c2b9-delete:
total 40
drwxr-xr-x.   2 nobody nobody  4096 Sep 19 10:37 .
drwxr-xr-x. 246 nobody nobody 32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody     5 Sep 19 10:22 .nfs0000000025f9008400000045
-rw-r--r--.   1 nobody nobody     0 Sep 19 10:25 .nfs0000000025f91d6800000048

/demo/broker_demo_1/__a_demo_topic-2:
total 832592
drwxr-xr-x.   2 nobody nobody      4096 Sep 19 10:26 .
drwxr-xr-x. 246 nobody nobody     32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody         5 Sep 19 10:22 .nfs0000000025f91d5500000046
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:25 .nfs0000000025f91fce00000047
-rw-r--r--.   1 nobody nobody  10485760 Sep 19 10:24 00000000000000000000.index
-rw-r--r--.   1 nobody nobody 848113134 Sep 19 10:24 00000000000000000000.log
-rw-r--r--.   1 nobody nobody  10485756 Sep 19 10:24 00000000000000000000.timeindex
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:16 leader-epoch-checkpoint
-rw-r--r--.   1 nobody nobody        43 Sep 19 10:16 partition.metadata
```

L'image suivante montre un rééquilibrage de partition propre du cluster 1 à l'aide du stockage NetApp NFSv4.1.



Pourquoi NetApp NFS pour les charges de travail Kafka ?

Maintenant qu'il existe une solution au problème de renommage stupide dans le stockage NFS avec Kafka, vous pouvez créer des déploiements robustes qui exploitent le stockage NetApp ONTAP pour votre charge de travail Kafka. Non seulement cela réduit considérablement les frais opérationnels, mais cela apporte également les avantages suivants à vos clusters Kafka :

- **Utilisation réduite du processeur sur les courtiers Kafka.** L'utilisation d'un stockage NetApp ONTAP désagrégé sépare les opérations d'E/S de disque du courtier et réduit ainsi son empreinte CPU.
- **Temps de récupération du courtier plus rapide.** Étant donné que le stockage NetApp ONTAP désagrégé est partagé entre les nœuds de courtier Kafka, une nouvelle instance de calcul peut remplacer un courtier défectueux à tout moment en une fraction du temps par rapport aux déploiements Kafka conventionnels sans reconstruire les données.
- **Efficacité de stockage.** Étant donné que la couche de stockage de l'application est désormais provisionnée via NetApp ONTAP, les clients peuvent bénéficier de tous les avantages de l'efficacité du stockage offerts par ONTAP, tels que la compression des données en ligne, la déduplication et le compactage.

Ces avantages ont été testés et validés dans des cas de test que nous discutons en détail dans cette section.

Utilisation réduite du processeur sur le courtier Kafka

Nous avons découvert que l'utilisation globale du processeur est inférieure à celle de son homologue DAS lorsque nous avons exécuté des charges de travail similaires sur deux clusters Kafka distincts qui étaient identiques dans leurs spécifications techniques mais différaient dans leurs technologies de stockage. Non seulement l'utilisation globale du processeur est plus faible lorsque le cluster Kafka utilise le stockage ONTAP , mais l'augmentation de l'utilisation du processeur a démontré un gradient plus doux que dans un cluster Kafka basé sur DAS.

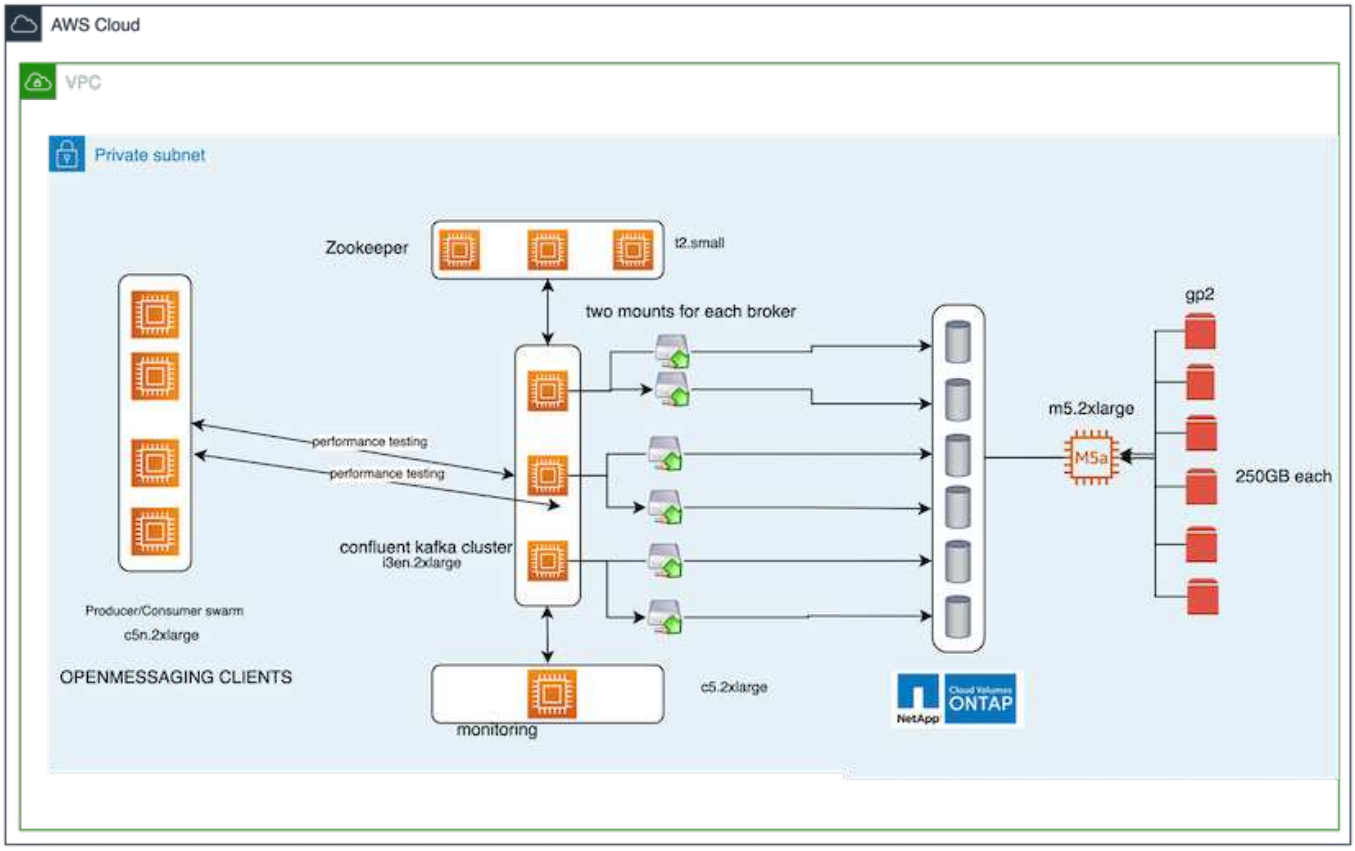
Configuration architecturale

Le tableau suivant montre la configuration environnementale utilisée pour démontrer l'utilisation réduite du processeur.

Composant de la plateforme	Configuration de l'environnement
Outil d'analyse comparative Kafka 3.2.3 : OpenMessaging	• 3 x gardiens de zoo – t2.small • 3 serveurs courtiers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Producteur/Consommateur — c5n.2xlarge
Système d'exploitation sur tous les nœuds	RHEL 8.7 ou version ultérieure
Instance NetApp Cloud Volumes ONTAP	Instance à nœud unique – M5.2xLarge

Outil d'analyse comparative

L'outil d'analyse comparative utilisé dans ce cas de test est le "OpenMessaging" cadre. OpenMessaging est indépendant des fournisseurs et de la langue ; il fournit des directives sectorielles pour la finance, le commerce électronique, l'IoT et le big data ; et il aide à développer des applications de messagerie et de streaming sur des systèmes et des plates-formes hétérogènes. La figure suivante illustre l'interaction des clients OpenMessaging avec un cluster Kafka.



- **Calculer.** Nous avons utilisé un cluster Kafka à trois nœuds avec un ensemble zookeeper à trois nœuds exécuté sur des serveurs dédiés. Chaque courtier disposait de deux points de montage NFSv4.1 sur un seul volume sur l'instance NetApp CVO via un LIF dédié.
- **Surveillance.** Nous avons utilisé deux nœuds pour une combinaison Prometheus-Grafana. Pour générer des charges de travail, nous disposons d'un cluster distinct à trois nœuds qui peut produire et consommer à partir de ce cluster Kafka.

- **Stockage.** Nous avons utilisé une instance ONTAP de volumes NetApp Cloud à nœud unique avec six volumes AWS-EBS GP2 de 250 Go montés sur l'instance. Ces volumes ont ensuite été exposés au cluster Kafka sous forme de six volumes NFSv4.1 via des LIF dédiés.
- **Configuration.** Les deux éléments configurables dans ce cas de test étaient les courtiers Kafka et les charges de travail OpenMessaging.
  - **Configuration du courtier.** Les spécifications suivantes ont été sélectionnées pour les courtiers Kafka. Nous avons utilisé un facteur de réplication de 3 pour toutes les mesures, comme indiqué ci-dessous.

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

- **Configuration de la charge de travail du benchmark OpenMessaging (OMB).** Les spécifications suivantes ont été fournies. Nous avons spécifié un taux de producteur cible, mis en évidence ci-dessous.

```
name: 4 producer / 4 consumers on 1 topic
topics: 1
partitionsPerTopic: 100
messageSize: 1024
payloadFile: "payload/payload-1Kb.data"
subscriptionsPerTopic: 1
consumerPerSubscription: 4
producersPerTopic: 4
producerRate: 40000
consumerBacklogSizeGB: 0
testDurationMinutes: 5
```

Méthodologie des tests

1. Deux clusters similaires ont été créés, chacun disposant de son propre ensemble d'essaims de clusters d'analyse comparative.

- **Groupe 1.** Cluster Kafka basé sur NFS.
- **Groupe 2.** Cluster Kafka basé sur DAS.

2. À l'aide d'une commande OpenMessaging, des charges de travail similaires ont été déclenchées sur chaque cluster.

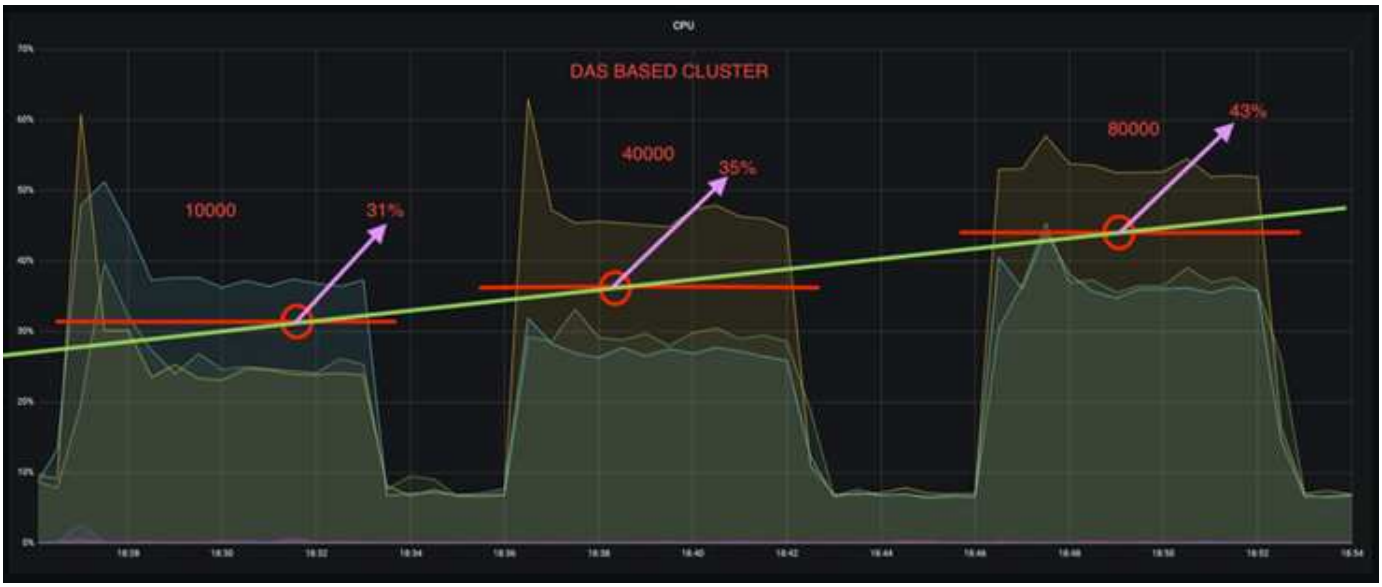
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

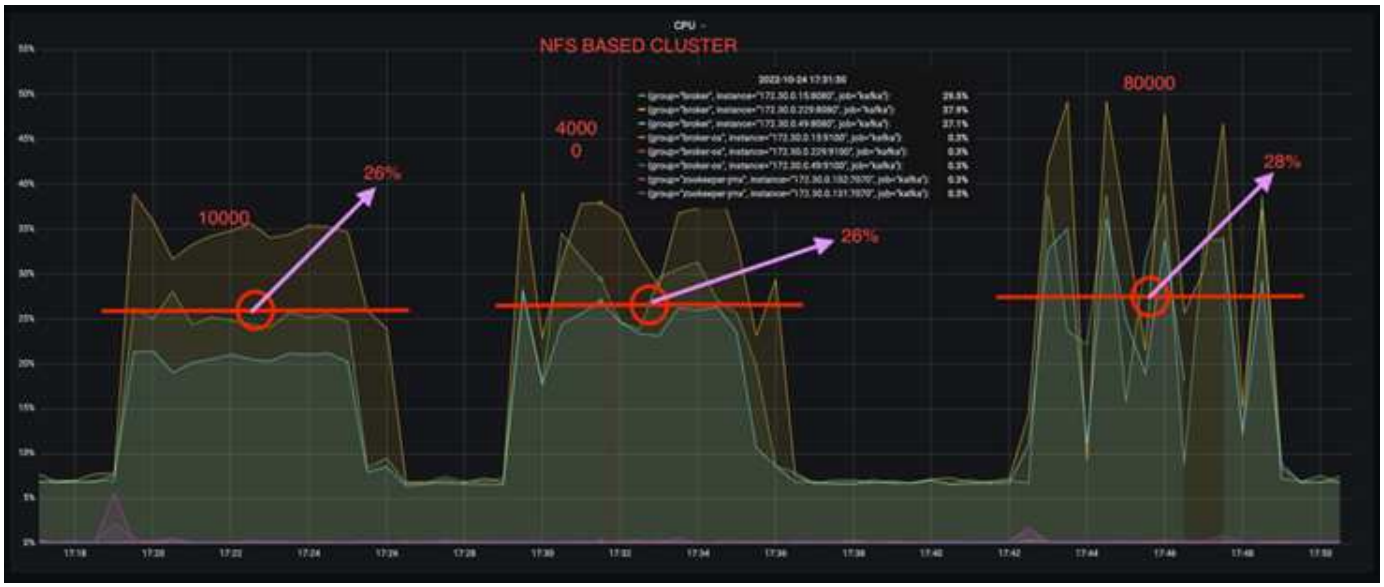
3. La configuration du taux de production a été augmentée en quatre itérations et l'utilisation du processeur a été enregistrée avec Grafana. Le taux de production a été fixé aux niveaux suivants :
   - 10 000
   - 40 000
   - 80 000
   - 100 000

Observation

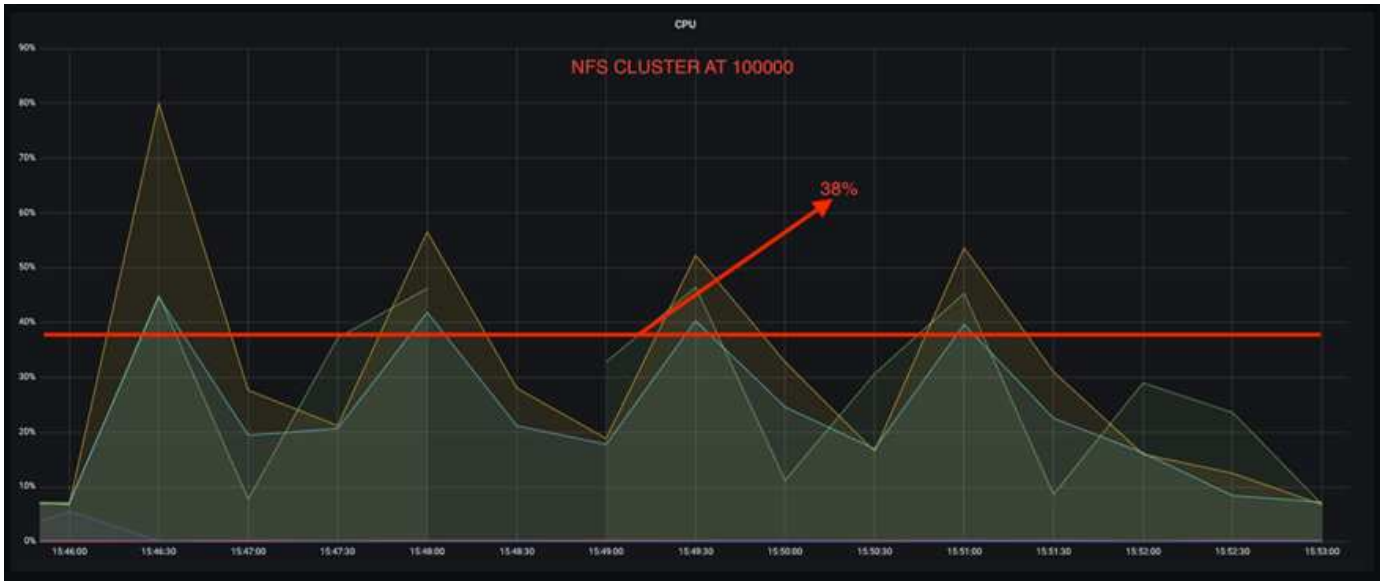
L'utilisation du stockage NetApp NFS avec Kafka présente deux avantages principaux :

- **Vous pouvez réduire l'utilisation du processeur de près d'un tiers.** L'utilisation globale du processeur sous des charges de travail similaires était inférieure pour NFS par rapport aux SSD DAS ; les économies varient de 5 % pour des taux de production inférieurs à 32 % pour des taux de production plus élevés.
- **Une réduction de trois fois de la dérive d'utilisation du processeur à des taux de production plus élevés.** Comme prévu, il y a eu une tendance à la hausse de l'utilisation du processeur à mesure que les taux de production ont augmenté. Cependant, l'utilisation du processeur sur les courtiers Kafka utilisant DAS est passée de 31 % pour le taux de production inférieur à 70 % pour le taux de production supérieur, soit une augmentation de 39 %. Cependant, avec un backend de stockage NFS, l'utilisation du processeur est passée de 26 % à 38 %, soit une augmentation de 12 %.





De plus, à 100 000 messages, DAS affiche une utilisation du processeur supérieure à celle d'un cluster NFS.



Récupération plus rapide des courtiers

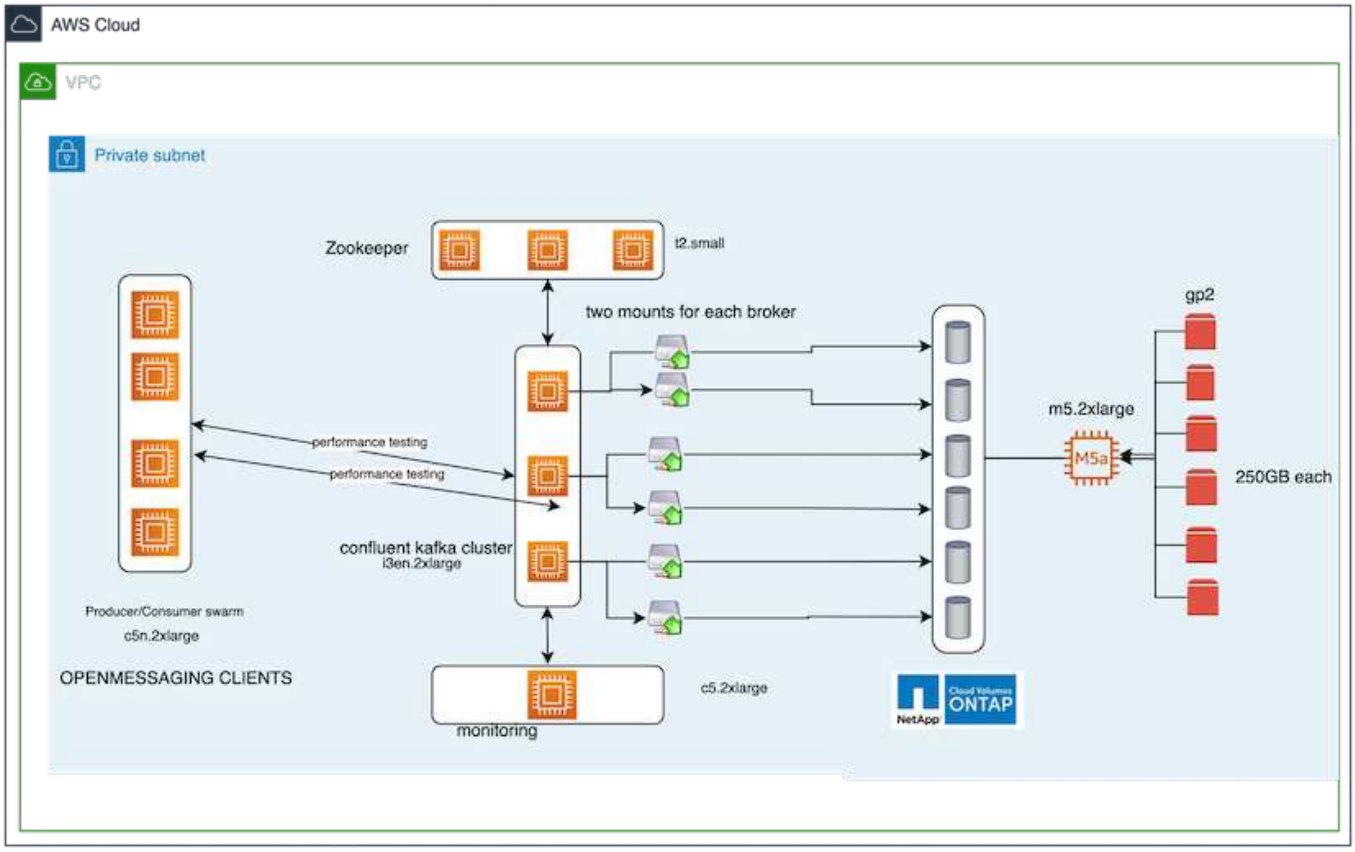
Nous avons découvert que les courtiers Kafka récupèrent plus rapidement lorsqu'ils utilisent un stockage NetApp NFS partagé. Lorsqu'un courtier tombe en panne dans un cluster Kafka, ce courtier peut être remplacé par un courtier sain avec le même ID de courtier. Après avoir effectué ce cas de test, nous avons constaté que, dans le cas d'un cluster Kafka basé sur DAS, le cluster reconstruit les données sur un courtier sain nouvellement ajouté, ce qui prend du temps. Dans le cas d'un cluster Kafka basé sur NetApp NFS, le courtier de remplacement continue de lire les données du répertoire de journaux précédent et récupère beaucoup plus rapidement.

Configuration architecturale

Le tableau suivant montre la configuration environnementale d'un cluster Kafka utilisant NAS.

Composant de la plateforme	Configuration de l'environnement
Kafka 3.2.3	• 3 x gardiens de zoo – t2.small • 3 serveurs courtiers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producteur/consommateur — c5n.2xlarge • 1 x nœud Kafka de sauvegarde – i3en.2xlarge
Système d'exploitation sur tous les nœuds	RHEL8.7 ou version ultérieure
Instance NetApp Cloud Volumes ONTAP	Instance à nœud unique – M5.2xLarge

La figure suivante illustre l'architecture d'un cluster Kafka basé sur NAS.



- **Calculer.** Un cluster Kafka à trois nœuds avec un ensemble zookeeper à trois nœuds exécuté sur des serveurs dédiés. Chaque courtier dispose de deux points de montage NFS sur un seul volume sur l'instance NetApp CVO via un LIF dédié.
- **Surveillance.** Deux nœuds pour une combinaison Prometheus-Grafana. Pour générer des charges de travail, nous utilisons un cluster à trois nœuds distinct qui peut produire et consommer sur ce cluster Kafka.
- **Stockage.** Une instance ONTAP de volumes NetApp Cloud à nœud unique avec six volumes AWS-EBS GP2 de 250 Go montés sur l'instance. Ces volumes sont ensuite exposés au cluster Kafka sous forme de six volumes NFS via des LIF dédiés.
- **Configuration du courtier.** Le seul élément configurable dans ce cas de test sont les courtiers Kafka. Les spécifications suivantes ont été sélectionnées pour les courtiers Kafka. Le `replica.lag.time.mx.ms` est défini sur une valeur élevée car cela détermine la vitesse à laquelle un nœud particulier est retiré de la liste ISR. Lorsque vous basculez entre des nœuds défectueux et sains, vous ne souhaitez pas que cet ID de courtier soit exclu de la liste ISR.

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

Méthodologie des tests

1. Deux clusters similaires ont été créés :
   - Un cluster confluent basé sur EC2.
   - Un cluster confluent basé sur NetApp NFS.
2. Un nœud Kafka de secours a été créé avec une configuration identique aux nœuds du cluster Kafka d'origine.
3. Sur chacun des clusters, un exemple de sujet a été créé et environ 110 Go de données ont été renseignés sur chacun des courtiers.
   - **Cluster basé sur EC2.** Un répertoire de données de courtier Kafka est mappé sur `/mnt/data-2` (Dans la figure suivante, Broker-1 du cluster1 [terminal gauche]).
   - * Cluster basé sur NetApp NFS.* Un répertoire de données de courtier Kafka est monté sur un point NFS `/mnt/data` (Dans la figure suivante, Broker-1 du cluster2 [terminal de droite]).

4. Dans chacun des clusters, Broker-1 a été arrêté pour déclencher un processus de récupération de courtier ayant échoué.
5. Une fois le courtier terminé, l'adresse IP du courtier a été attribuée comme adresse IP secondaire au courtier de secours. Cela était nécessaire car un courtier dans un cluster Kafka est identifié par les éléments suivants :
   - **Adresse IP.** Attribué en réaffectant l'IP du courtier défaillant au courtier de secours.
   - **Identifiant du courtier.** Ceci a été configuré dans le courtier de secours `server.properties` .
6. Lors de l'attribution de l'IP, le service Kafka a été démarré sur le courtier de secours.
7. Après un certain temps, les journaux du serveur ont été extraits pour vérifier le temps nécessaire à la création des données sur le nœud de remplacement du cluster.

Observation

La récupération du courtier Kafka a été presque neuf fois plus rapide. Le temps nécessaire pour récupérer un nœud de courtier défaillant s'est avéré nettement plus rapide lors de l'utilisation du stockage partagé NetApp NFS par rapport à l'utilisation de SSD DAS dans un cluster Kafka. Pour 1 To de données de sujet, le temps de récupération pour un cluster basé sur DAS était de 48 minutes, contre moins de 5 minutes pour un cluster Kafka basé sur NetApp-NFS.

Nous avons observé que le cluster basé sur EC2 a mis 10 minutes pour reconstruire les 110 Go de données sur le nouveau nœud de courtier, tandis que le cluster basé sur NFS a terminé la récupération en 3 minutes. Nous avons également observé dans les journaux que les décalages des consommateurs pour les partitions pour EC2 étaient de 0, tandis que, sur le cluster NFS, les décalages des consommateurs étaient récupérés à partir du courtier précédent.

```
[2022-10-31 09:39:17,747] INFO [LogLoader partition=test-topic-51R3EWs-
0000-55, dir=/mnt/kafka-data/broker2] Reloading from producer snapshot and
rebuilding producer state from offset 583999 (kafka.log.UnifiedLog$)
[2022-10-31 08:55:55,170] INFO [LogLoader partition=test-topic-qbVsEZg-
0000-8, dir=/mnt/data-1] Loading producer state till offset 0 with message
format version 2 (kafka.log.UnifiedLog$)
```

Cluster basé sur DAS

1. Le nœud de sauvegarde a démarré à 08:55:53,730.

```
2 [2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
3 [2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (or
4 [2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
5 [2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.3
6 [2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. Le processus de reconstruction des données s'est terminé à 09:05:24,860. Le traitement de 110 Go de données a nécessité environ 10 minutes.

```
[2022-10-31 09:05:24,860] INFO [ReplicaFetcherManager on broker 1] Removed fetcher for
partitions HashSet(test-topic-qbVsEZg-0000-95, test-topic-qbVsEZg-0000-5,
test-topic-qbVsEZg-0000-41, test-topic-qbVsEZg-0000-23, test-topic-qbVsEZg-0000-11,
test-topic-qbVsEZg-0000-47, test-topic-qbVsEZg-0000-83, test-topic-qbVsEZg-0000-35,
test-topic-qbVsEZg-0000-89, test-topic-qbVsEZg-0000-71, test-topic-qbVsEZg-0000-53,
test-topic-qbVsEZg-0000-29, test-topic-qbVsEZg-0000-59, test-topic-qbVsEZg-0000-77,
test-topic-qbVsEZg-0000-65, test-topic-qbVsEZg-0000-17)
(kafka.server.ReplicaFetcherManager)
```

Cluster basé sur NFS

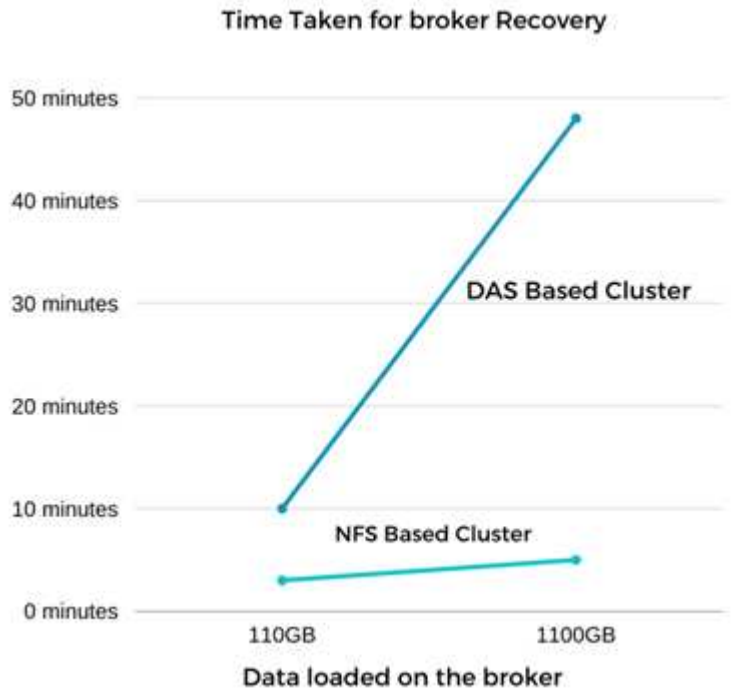
1. Le nœud de sauvegarde a été démarré à 09:39:17,213. L'entrée du journal de départ est mise en évidence ci-dessous.

```
[2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiati
[2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.
[2022-10-31 09:39:17,213] INFO starting (kafka.server.KafkaServer)
[2022-10-31 09:39:17,214] INFO Connecting to zookeeper on 172.30.0.22:2181,172.30.
[2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new s
[2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a
[2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in
```

2. Le processus de reconstruction des données s'est terminé à 09:42:29,115. Le traitement de 110 Go de données a nécessité environ 3 minutes.

```
[2022-10-31 09:42:29,115] INFO [GroupMetadataManager brokerId=1] Finished loading offsets
and group metadata from __consumer_offsets-20 in 28478 milliseconds for epoch 3, of which
28478 milliseconds was spent in the scheduler.
(kafka.coordinator.group.GroupMetadataManager)
```

Le test a été répété pour les courtiers contenant environ 1 To de données, ce qui a pris environ 48 minutes pour le DAS et 3 minutes pour le NFS. Les résultats sont représentés dans le graphique suivant.



Efficacité du stockage

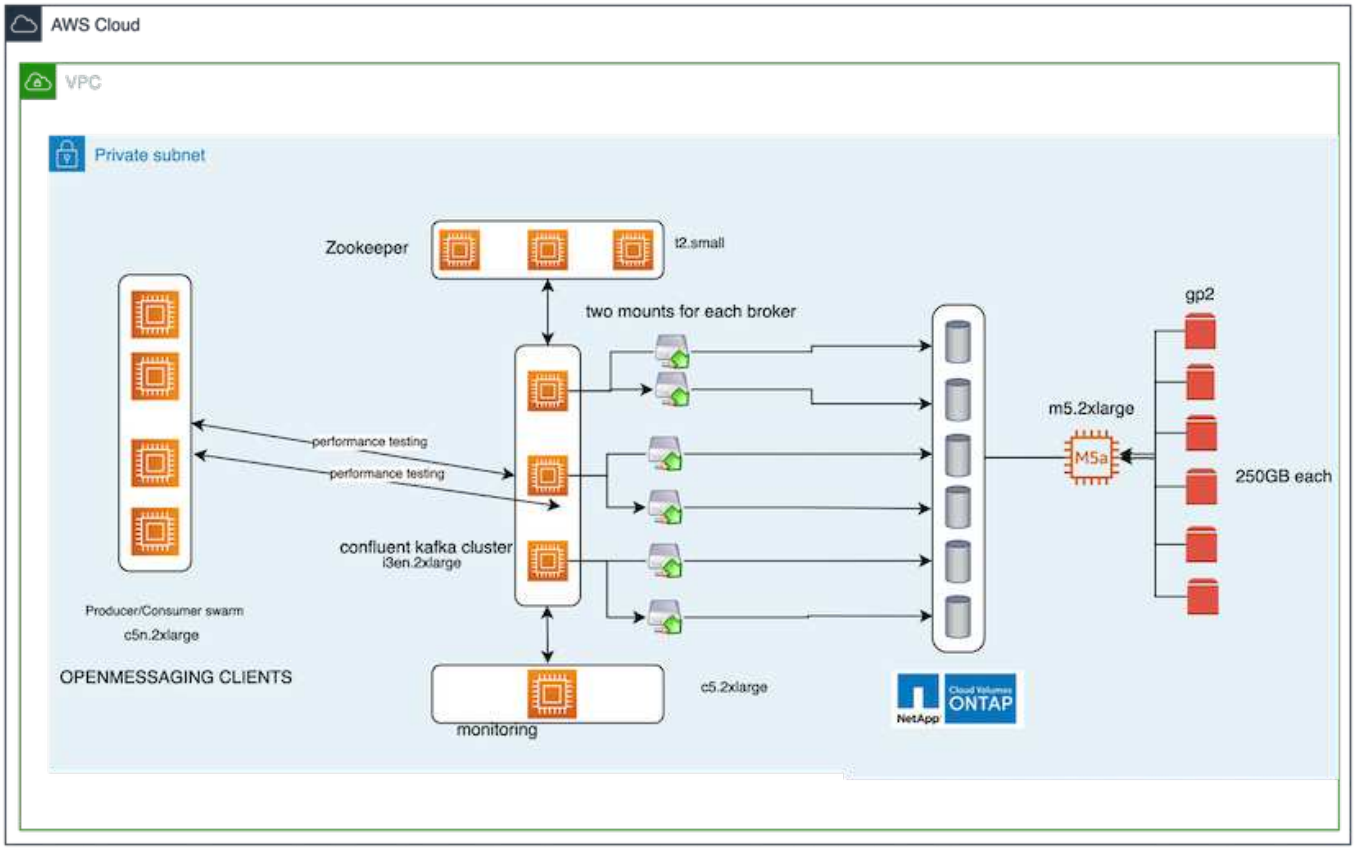
Étant donné que la couche de stockage du cluster Kafka a été provisionnée via NetApp ONTAP, nous avons obtenu toutes les capacités d'efficacité de stockage d' ONTAP. Cela a été testé en générant une quantité importante de données sur un cluster Kafka avec un stockage NFS provisionné sur Cloud Volumes ONTAP. Nous avons pu constater une réduction significative de l'espace grâce aux capacités ONTAP .

Configuration architecturale

Le tableau suivant montre la configuration environnementale d'un cluster Kafka utilisant NAS.

Composant de la plateforme	Configuration de l'environnement
Kafka 3.2.3	• 3 x gardiens de zoo – t2.small • 3 serveurs courtiers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producteur/consommateur — c5n.2xlarge *
Système d'exploitation sur tous les nœuds	RHEL8.7 ou version ultérieure
Instance NetApp Cloud Volumes ONTAP	Instance à nœud unique – M5.2xLarge

La figure suivante illustre l'architecture d'un cluster Kafka basé sur NAS.



- **Calculer.** Nous avons utilisé un cluster Kafka à trois nœuds avec un ensemble zookeeper à trois nœuds exécuté sur des serveurs dédiés. Chaque courtier disposait de deux points de montage NFS sur un seul volume sur l'instance NetApp CVO via un LIF dédié.
- **Surveillance.** Nous avons utilisé deux nœuds pour une combinaison Prometheus-Grafana. Pour générer des charges de travail, nous avons utilisé un cluster à trois nœuds distinct qui pouvait produire et consommer sur ce cluster Kafka.
- **Stockage.** Nous avons utilisé une instance NetApp Cloud Volumes ONTAP à nœud unique avec six volumes AWS-EBS GP2 de 250 Go montés sur l'instance. Ces volumes ont ensuite été exposés au cluster Kafka sous forme de six volumes NFS via des LIF dédiés.
- **Configuration.** Les éléments configurables dans ce cas de test étaient les courtiers Kafka.

La compression a été désactivée du côté du producteur, permettant ainsi aux producteurs de générer un débit

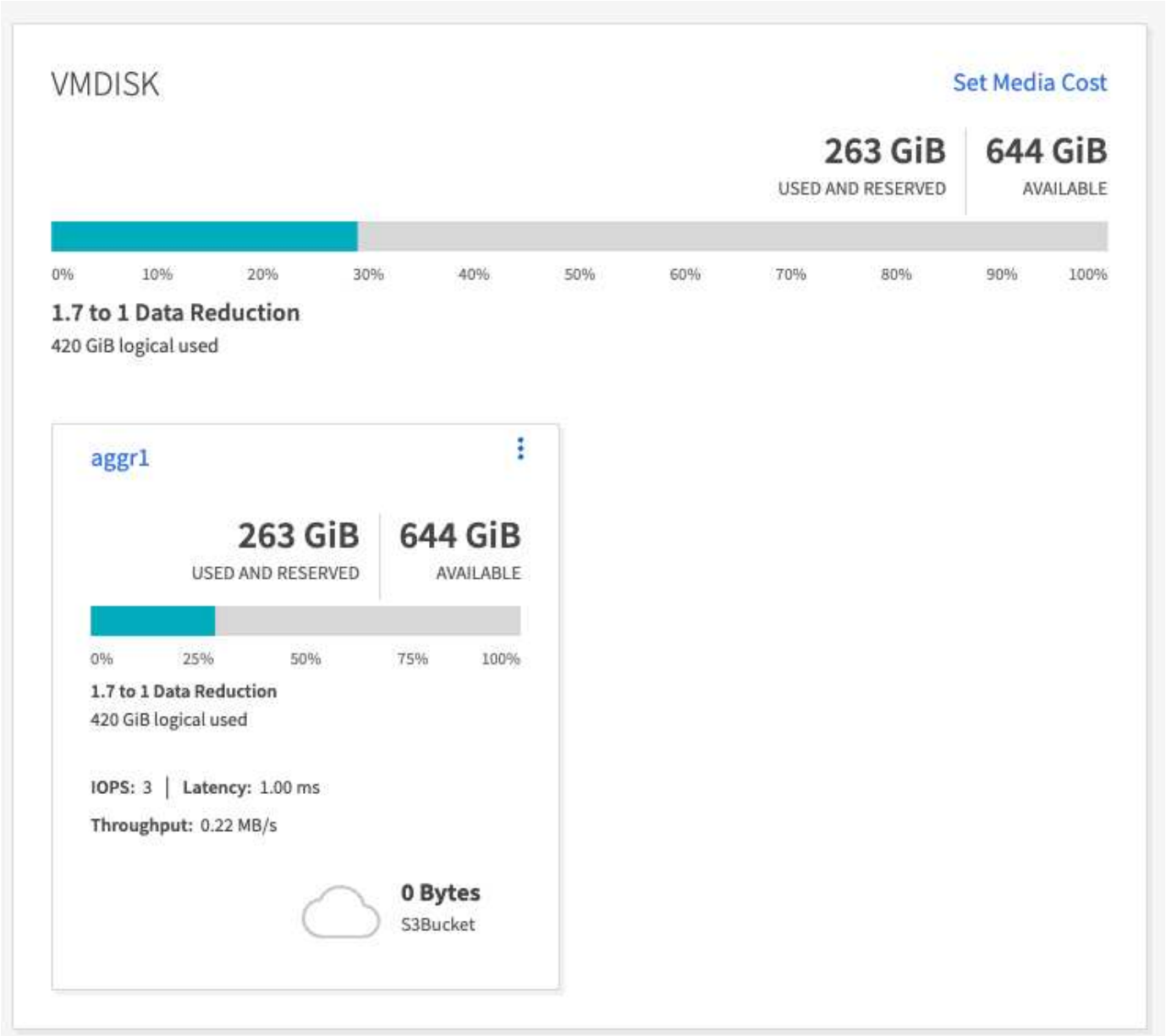
élevé. L'efficacité du stockage était plutôt gérée par la couche de calcul.

Méthodologie des tests

1. Un cluster Kafka a été provisionné avec les spécifications mentionnées ci-dessus.
2. Sur le cluster, environ 350 Go de données ont été produites à l'aide de l'outil OpenMessaging Benchmarking.
3. Une fois la charge de travail terminée, les statistiques d'efficacité du stockage ont été collectées à l'aide ONTAP System Manager et de l'interface de ligne de commande.

Observation

Pour les données générées à l'aide de l'outil OMB, nous avons constaté des économies d'espace d'environ 33 % avec un ratio d'efficacité de stockage de 1,70:1. Comme le montrent les figures suivantes, l'espace logique utilisé par les données produites était de 420,3 Go et l'espace physique utilisé pour contenir les données était de 281,7 Go.



```
shantanuCVOinstancenew::> df -h -S

Warning: The "-S" parameter is deprecated and may be removed in a future release. To show the efficiency ratio use "aggr show-efficiency"
         command.
Filesystem               used      total-saved    %total-saved    deduplicated    %deduplicated    compressed    %compressed  Vserver
/vol/vol0/             7319MB               0B              0%              0B              0%            0B            0%  shantanuCVOinstancenew-01
/vol/kafka_vol/         281GB            138GB             33%           138GB             33%            0B            0%  svm_shantanuCVOinstancenew
/vol/svm_shantanuCVOinstancenew_root/
                        660KB               0B              0%              0B              0%            0B            0%  svm_shantanuCVOinstancenew
3 entries were displayed.
```

```
                    Name of the Aggregate: aggr1
               Node where Aggregate Resides: shantanuCVOinstancenew-01
             Total Storage Efficiency Ratio: 1.70:1
Total Data Reduction Efficiency Ratio Without Snapshots: 1.70:1
Total Data Reduction Efficiency Ratio without snapshots and flexclones: 1.70:1
          Logical Space Used for All Volumes: 420.3GB
         Physical Space Used for All Volumes: 281.7GB
```

Aperçu des performances et validation dans AWS

Un cluster Kafka avec la couche de stockage montée sur NetApp NFS a été évalué pour les performances dans le cloud AWS. Les exemples d'analyse comparative sont décrits dans les sections suivantes.

Kafka dans le cloud AWS avec NetApp Cloud Volumes ONTAP (paire haute disponibilité et nœud unique)

Un cluster Kafka avec NetApp Cloud Volumes ONTAP (paire HA) a été évalué pour ses performances dans le cloud AWS. Cette analyse comparative est décrite dans les sections suivantes.

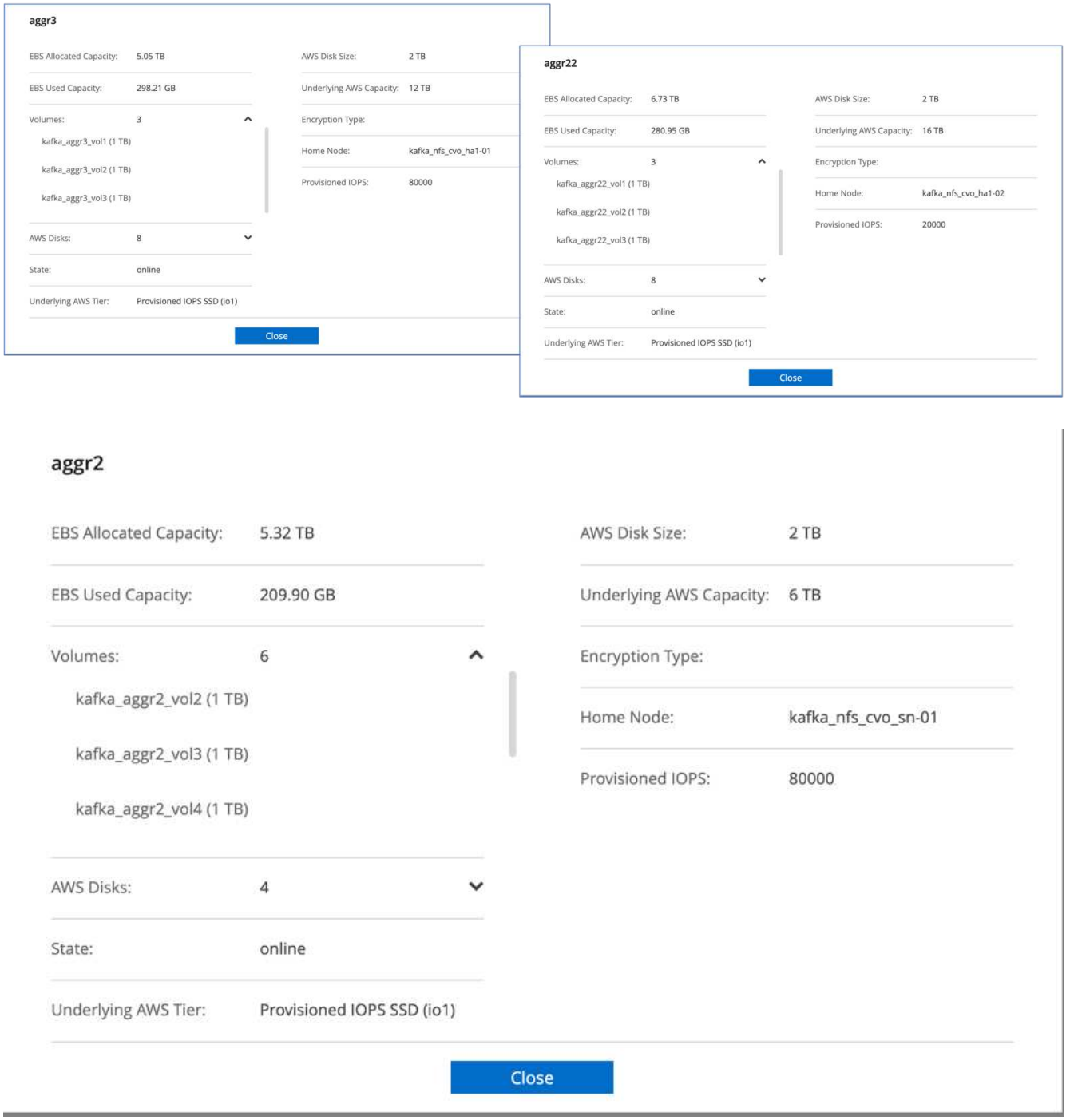
Configuration architecturale

Le tableau suivant montre la configuration environnementale d'un cluster Kafka utilisant NAS.

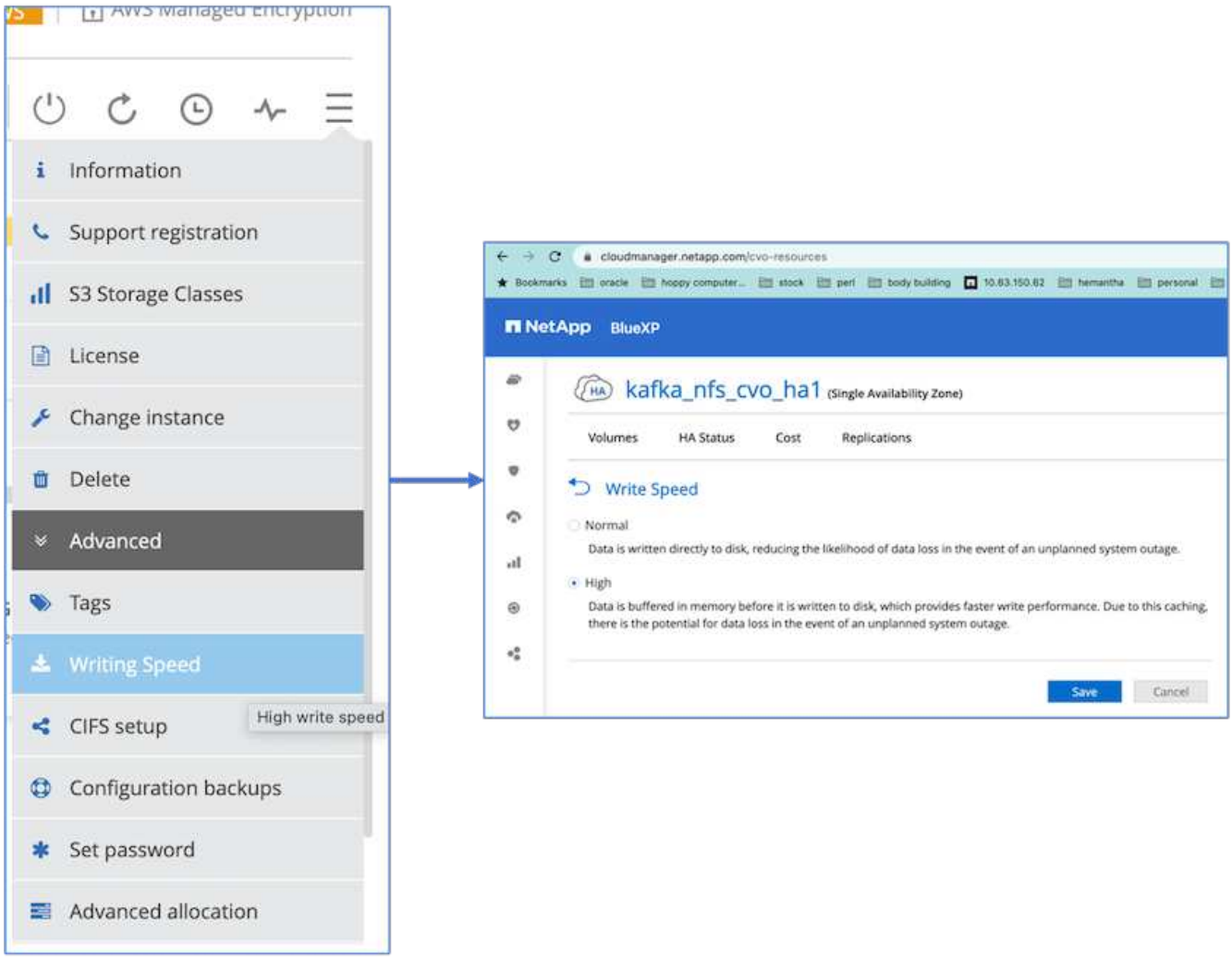
Composant de la plateforme	Configuration de l'environnement
Kafka 3.2.3	• 3 x gardiens de zoo – t2.small • 3 serveurs courtiers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producteur/consommateur — c5n.2xlarge *
Système d'exploitation sur tous les nœuds	RHEL8.6
Instance NetApp Cloud Volumes ONTAP	Instance de paire HA – m5dn.12xLarge x 2node Instance de nœud unique - m5dn.12xLarge x 1 nœud

Configuration du volume de cluster NetApp ONTAP

1. Pour la paire Cloud Volumes ONTAP HA, nous avons créé deux agrégats avec trois volumes sur chaque agrégat sur chaque contrôleur de stockage. Pour le nœud unique Cloud Volumes ONTAP , nous créons six volumes dans un agrégat.

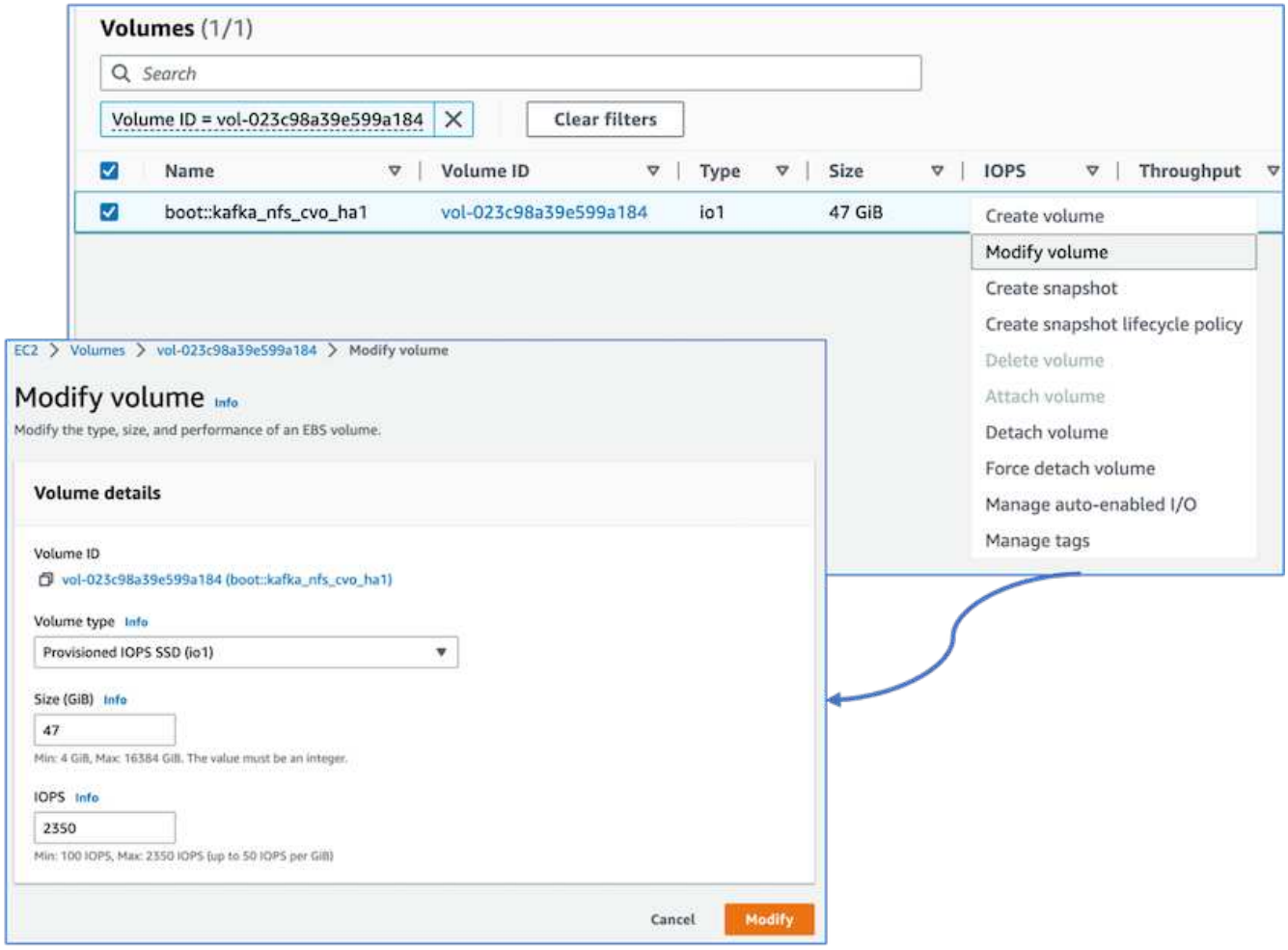


2. Pour obtenir de meilleures performances réseau, nous avons activé la mise en réseau à haut débit pour la paire HA et le nœud unique.



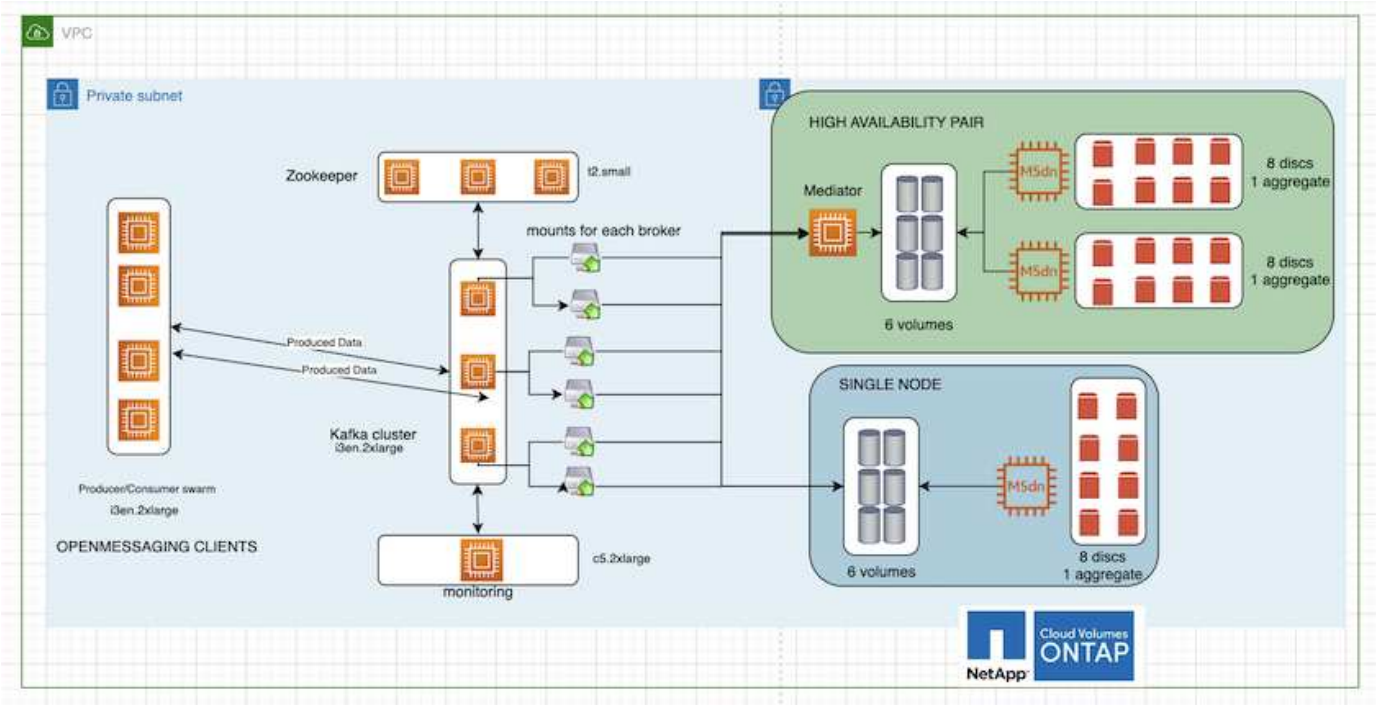
3. Nous avons remarqué que la NVRAM ONTAP avait plus d'IOPS, nous avons donc modifié les IOPS à 2350 pour le volume racine Cloud Volumes ONTAP . Le disque du volume racine dans Cloud Volumes ONTAP avait une taille de 47 Go. La commande ONTAP suivante concerne la paire HA et la même étape s'applique au nœud unique.

```
statistics start -object vnvram -instance vnvram -counter
backing_store_iops -sample-id sample_555
kafka_nfs_cvo_ha1::*> statistics show -sample-id sample_555
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-01
    Counter                                                     Value
    -------------------------------- --------------------------------
    backing_store_iops                                           1479
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-02
    Counter                                                     Value
    -------------------------------- --------------------------------
    backing_store_iops                                           1210
2 entries were displayed.
kafka_nfs_cvo_ha1::*>
```



La figure suivante illustre l'architecture d'un cluster Kafka basé sur NAS.

- **Calculer.** Nous avons utilisé un cluster Kafka à trois nœuds avec un ensemble zookeeper à trois nœuds exécuté sur des serveurs dédiés. Chaque courtier disposait de deux points de montage NFS sur un seul volume sur l'instance Cloud Volumes ONTAP via un LIF dédié.
- **Surveillance.** Nous avons utilisé deux nœuds pour une combinaison Prometheus-Grafana. Pour générer des charges de travail, nous avons utilisé un cluster à trois nœuds distinct qui pouvait produire et consommer sur ce cluster Kafka.
- **Stockage.** Nous avons utilisé une instance ONTAP de volumes Cloud à paire HA avec un volume AWS-EBS GP3 de 6 To monté sur l'instance. Le volume a ensuite été exporté vers le courtier Kafka avec un montage NFS.



Configurations d’analyse comparative d’OpenMessage

1. Pour de meilleures performances NFS, nous avons besoin de plus de connexions réseau entre le serveur NFS et le client NFS, qui peuvent être créées à l’aide de nconnect. Montez les volumes NFS sur les nœuds du courtier avec l’option nconnect en exécutant la commande suivante :

```
[root@ip-172-30-0-121 ~]# cat /etc/fstab
UUID=eaa1f38e-de0f-4ed5-a5b5-2fa9db43bb38/xfsdefaults00
/dev/nvme1n1 /mnt/data-1 xfs defaults,noatime,nodiscard 0 0
/dev/nvme2n1 /mnt/data-2 xfs defaults,noatime,nodiscard 0 0
172.30.0.233:/kafka_aggr3_vol1 /kafka_aggr3_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol2 /kafka_aggr3_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol3 /kafka_aggr3_vol3 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol1 /kafka_aggr22_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol2 /kafka_aggr22_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol3 /kafka_aggr22_vol3 nfs
defaults,nconnect=16 0 0
[root@ip-172-30-0-121 ~]# mount -a
[root@ip-172-30-0-121 ~]# df -h
Filesystem                         Size  Used Avail Use% Mounted on
devtmpfs                            31G     0   31G   0% /dev
tmpfs                               31G  249M   31G   1% /run
tmpfs                               31G     0   31G   0% /sys/fs/cgroup
/dev/nvme0n1p2                      10G  2.8G  7.2G  28% /
/dev/nvme1n1                       2.3T  248G  2.1T  11% /mnt/data-1
/dev/nvme2n1                       2.3T  245G  2.1T  11% /mnt/data-2
172.30.0.233:/kafka_aggr3_vol1     1.0T   12G 1013G   2% /kafka_aggr3_vol1
172.30.0.233:/kafka_aggr3_vol2     1.0T  5.5G 1019G   1% /kafka_aggr3_vol2
172.30.0.233:/kafka_aggr3_vol3     1.0T  8.9G 1016G   1% /kafka_aggr3_vol3
172.30.0.242:/kafka_aggr22_vol1    1.0T  7.3G 1017G   1%
/kafka_aggr22_vol1
172.30.0.242:/kafka_aggr22_vol2    1.0T  6.9G 1018G   1%
/kafka_aggr22_vol2
172.30.0.242:/kafka_aggr22_vol3    1.0T  5.9G 1019G   1%
/kafka_aggr22_vol3
tmpfs                              6.2G     0  6.2G   0% /run/user/1000
[root@ip-172-30-0-121 ~]#
```

2. Vérifiez les connexions réseau dans Cloud Volumes ONTAP. La commande ONTAP suivante est utilisée à partir du nœud Cloud Volumes ONTAP unique. La même étape s'applique à la paire Cloud Volumes ONTAP HA.

```
Last login time: 1/20/2023 00:16:29
kafka_nfs_cvo_sn::> network connections active show -service nfs*
-fields remote-host
node               cid        vserver              remote-host
```

```
------------------ ---------- ------------------- ------------
kafka_nfs_cvo_sn-01 2315762628 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762629 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762630 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762631 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762632 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762633 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762634 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762635 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762636 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762637 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762639 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762640 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762641 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762642 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762643 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762644 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762645 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762646 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762647 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762648 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762649 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762650 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762651 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762652 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762653 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762656 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762657 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762658 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762659 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762660 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762661 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762662 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762663 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762664 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762665 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762666 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762667 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762668 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762669 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762670 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762671 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762672 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762673 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762674 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762676 svm_kafka_nfs_cvo_sn 172.30.0.121
```

```
kafka_nfs_cvo_sn-01 2315762677 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762678 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762679 svm_kafka_nfs_cvo_sn 172.30.0.223
48 entries were displayed.

kafka_nfs_cvo_sn::>
```

3. Nous utilisons le Kafka suivant `server.properties` dans tous les courtiers Kafka pour la paire Cloud Volumes ONTAP HA. Le `log.dirs` la propriété est différente pour chaque courtier, et les propriétés restantes sont communes aux courtiers. Pour broker1, le `log.dirs` la valeur est la suivante :

```
[root@ip-172-30-0-121 ~]# cat /opt/kafka/config/server.properties
broker.id=0
advertised.listeners=PLAINTEXT://172.30.0.121:9092
#log.dirs=/mnt/data-1/d1,/mnt/data-1/d2,/mnt/data-1/d3,/mnt/data-
2/d1,/mnt/data-2/d2,/mnt/data-2/d3
log.dirs=/kafka_aggr3_vol1/broker1,/kafka_aggr3_vol2/broker1,/kafka_aggr
3_vol3/broker1,/kafka_aggr22_vol1/broker1,/kafka_aggr22_vol2/broker1,/ka
fka_aggr22_vol3/broker1
zookeeper.connect=172.30.0.12:2181,172.30.0.30:2181,172.30.0.178:2181
num.network.threads=64
num.io.threads=64
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
num.partitions=1
num.recovery.threads.per.data.dir=1
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
replica.fetch.max.bytes=524288000
background.threads=20
num.replica.alter.log.dirs.threads=40
num.replica.fetchers=20
[root@ip-172-30-0-121 ~]#
```

◦ Pour broker2, le `log.dirs` la valeur de la propriété est la suivante :

```
log.dirs=/kafka_aggr3_vol1/broker2,/kafka_aggr3_vol2/broker2,/kafka_a
ggr3_vol3/broker2,/kafka_aggr22_vol1/broker2,/kafka_aggr22_vol2/broke
r2,/kafka_aggr22_vol3/broker2
```

◦ Pour broker3, le `log.dirs` la valeur de la propriété est la suivante :

```
log.dirs=/kafka_aggr3_vol1/broker3,/kafka_aggr3_vol2/broker3,/kafka_a
ggr3_vol3/broker3,/kafka_aggr22_vol1/broker3,/kafka_aggr22_vol2/broke
r3,/kafka_aggr22_vol3/broker3
```

4. Pour le nœud unique Cloud Volumes ONTAP , Kafka `servers.properties` est le même que pour la paire Cloud Volumes ONTAP HA, à l'exception du `log.dirs` propriété.
   - Pour broker1, le `log.dirs` la valeur est la suivante :

```
log.dirs=/kafka_aggr2_vol1/broker1,/kafka_aggr2_vol2/broker1,/kafka_a
ggr2_vol3/broker1,/kafka_aggr2_vol4/broker1,/kafka_aggr2_vol5/broker1
,/kafka_aggr2_vol6/broker1
```

   - Pour broker2, le `log.dirs` la valeur est la suivante :

```
log.dirs=/kafka_aggr2_vol1/broker2,/kafka_aggr2_vol2/broker2,/kafka_a
ggr2_vol3/broker2,/kafka_aggr2_vol4/broker2,/kafka_aggr2_vol5/broker2
,/kafka_aggr2_vol6/broker2
```

   - Pour broker3, le `log.dirs` la valeur de la propriété est la suivante :

```
log.dirs=/kafka_aggr2_vol1/broker3,/kafka_aggr2_vol2/broker3,/kafka_a
ggr2_vol3/broker3,/kafka_aggr2_vol4/broker3,/kafka_aggr2_vol5/broker3
,/kafka_aggr2_vol6/broker3
```

5. La charge de travail dans l'OMB est configurée avec les propriétés suivantes : `(/opt/benchmark/workloads/1-topic-100-partitions-1kb.yaml)` .

```
topics: 4
partitionsPerTopic: 100
messageSize: 32768
useRandomizedPayloads: true
randomBytesRatio: 0.5
randomizedPayloadPoolSize: 100
subscriptionsPerTopic: 1
consumerPerSubscription: 80
producersPerTopic: 40
producerRate: 1000000
consumerBacklogSizeGB: 0
testDurationMinutes: 5
```

Le `messageSize` peut varier pour chaque cas d'utilisation. Dans notre test de performance, nous avons

utilisé 3K.

Nous avons utilisé deux pilotes différents, Sync ou Throughput, d'OMB pour générer la charge de travail sur le cluster Kafka.

- Le fichier yaml utilisé pour les propriétés du pilote de synchronisation est le suivant (`/opt/benchmark/driver- kafka/kafka-sync.yaml`) :

```
name: Kafka
driverClass:
io.openmessaging.benchmark.driver.kafka.KafkaBenchmarkDriver
# Kafka client-specific configuration
replicationFactor: 3
topicConfig: |
  min.insync.replicas=2
  flush.messages=1
  flush.ms=0
commonConfig: |

bootstrap.servers=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

- Le fichier yaml utilisé pour les propriétés du pilote de débit est le suivant (`/opt/benchmark/driver- kafka/kafka-throughput.yaml`) :

```
name: Kafka
driverClass:
io.openmessaging.benchmark.driver.kafka.KafkaBenchmarkDriver
# Kafka client-specific configuration
replicationFactor: 3
topicConfig: |
  min.insync.replicas=2
commonConfig: |

bootstrap.servers=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
  default.api.timeout.ms=1200000
  request.timeout.ms=1200000
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

Méthodologie des tests

1. Un cluster Kafka a été provisionné conformément à la spécification décrite ci-dessus à l'aide de Terraform et Ansible. Terraform est utilisé pour créer l'infrastructure à l'aide d'instances AWS pour le cluster Kafka et Ansible construit le cluster Kafka sur celles-ci.
2. Une charge de travail OMB a été déclenchée avec la configuration de charge de travail décrite ci-dessus et le pilote Sync.

```
Sudo bin/benchmark –drivers driver-kafka/kafka- sync.yaml workloads/1-
topic-100-partitions-1kb.yaml
```

3. Une autre charge de travail a été déclenchée avec le pilote de débit avec la même configuration de charge de travail.

```
sudo bin/benchmark –drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

Observation

Deux types de pilotes différents ont été utilisés pour générer des charges de travail afin d'évaluer les performances d'une instance Kafka exécutée sur NFS. La différence entre les pilotes est la propriété de vidage

du journal.

Pour une paire Cloud Volumes ONTAP HA :

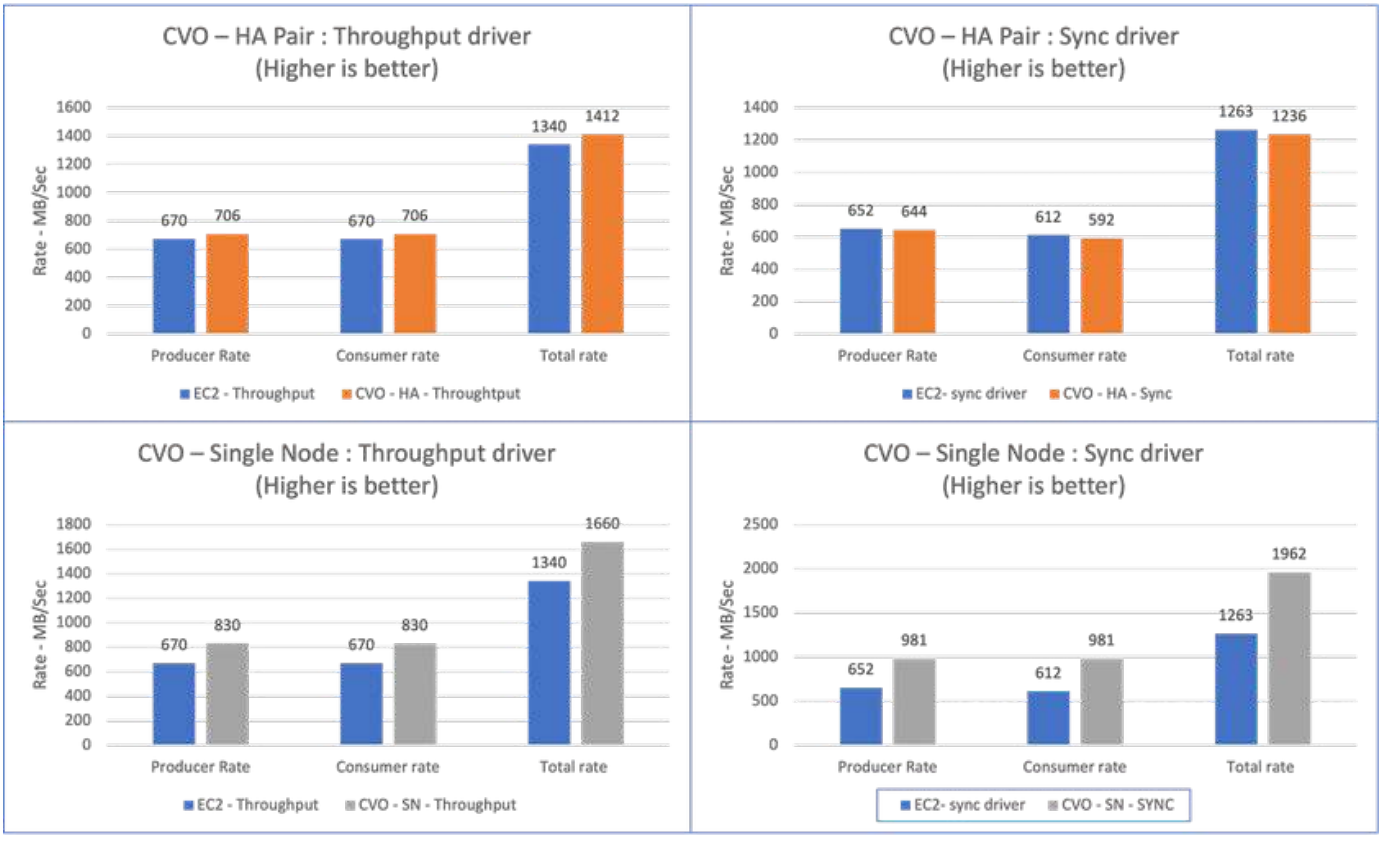
- Débit total généré de manière cohérente par le pilote Sync : environ 1 236 Mo/s.
- Débit total généré pour le pilote de débit : pic ~ 1 412 Mo/s.

Pour un seul nœud Cloud Volumes ONTAP :

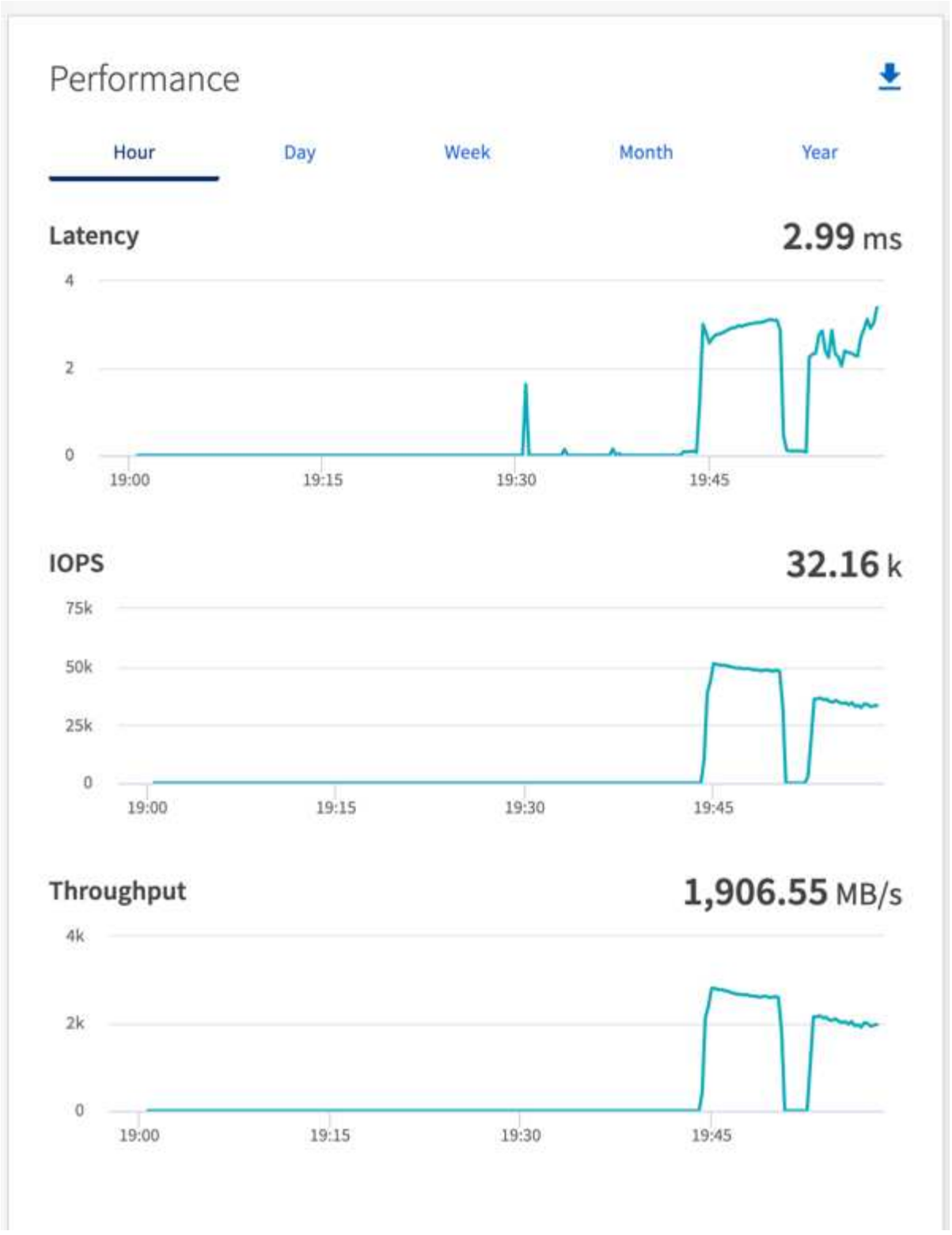
- Débit total généré de manière cohérente par le pilote Sync : ~ 1 962 Mo/s.
- Débit total généré par le pilote de débit : pic ~ 1 660 Mo/s

Le pilote de synchronisation peut générer un débit constant lorsque les journaux sont vidés instantanément sur le disque, tandis que le pilote de débit génère des rafales de débit lorsque les journaux sont validés sur le disque en masse.

Ces numéros de débit sont générés pour la configuration AWS donnée. Pour des exigences de performances plus élevées, les types d'instances peuvent être mis à l'échelle et optimisés davantage pour de meilleurs chiffres de débit. Le débit total ou le taux total est la combinaison du taux du producteur et du taux du consommateur.



Assurez-vous de vérifier le débit de stockage lorsque vous effectuez une analyse comparative du débit ou du pilote de synchronisation.



Présentation et validation des performances dans AWS FSx ONTAP

Un cluster Kafka avec la couche de stockage montée sur NetApp NFS a été évalué pour les performances dans AWS FSx ONTAP. Les exemples d'analyse comparative sont décrits dans les sections suivantes.

Apache Kafka dans AWS FSx ONTAP

Network File System (NFS) est un système de fichiers réseau largement utilisé pour stocker de grandes quantités de données. Dans la plupart des organisations, les données sont de plus en plus générées par des applications de streaming comme Apache Kafka. Ces charges de travail nécessitent une évolutivité, une faible latence et une architecture d'ingestion de données robuste avec des capacités de stockage modernes. Pour permettre des analyses en temps réel et fournir des informations exploitables, une infrastructure bien conçue et hautement performante est nécessaire.

Kafka, de par sa conception, fonctionne avec un système de fichiers compatible POSIX et s'appuie sur le système de fichiers pour gérer les opérations de fichiers, mais lors du stockage de données sur un système de fichiers NFSv3, le client NFS du courtier Kafka peut interpréter les opérations de fichiers différemment d'un système de fichiers local comme XFS ou Ext4. Un exemple courant est le renommage idiot de NFS qui a provoqué l'échec des courtiers Kafka lors de l'extension des clusters et de la réallocation des partitions. Pour relever ce défi, NetApp a mis à jour le client NFS Linux open source avec des modifications désormais généralement disponibles dans RHEL8.7, RHEL9.1 et prises en charge à partir de la version actuelle de FSx ONTAP , ONTAP 9.12.1.

Amazon FSx ONTAP fournit un système de fichiers NFS entièrement géré, évolutif et hautement performant dans le cloud. Les données Kafka sur FSx ONTAP peuvent évoluer pour gérer de grandes quantités de données et garantir la tolérance aux pannes. NFS fournit une gestion centralisée du stockage et une protection des données pour les ensembles de données critiques et sensibles.

Ces améliorations permettent aux clients AWS de profiter de FSx ONTAP lors de l'exécution de charges de travail Kafka sur les services de calcul AWS. Ces avantages sont : * Réduction de l'utilisation du processeur pour réduire le temps d'attente des E/S * Temps de récupération du courtier Kafka plus rapide. * Fiabilité et efficacité. * Évolutivité et performance. * Disponibilité en zone multi-disponibilité. * Protection des données.

Présentation et validation des performances dans AWS FSx ONTAP

Un cluster Kafka avec la couche de stockage montée sur NetApp NFS a été évalué pour les performances dans le cloud AWS. Les exemples d'analyse comparative sont décrits dans les sections suivantes.

Kafka dans AWS FSx ONTAP

Un cluster Kafka avec AWS FSx ONTAP a été évalué pour ses performances dans le cloud AWS. Cette analyse comparative est décrite dans les sections suivantes.

Configuration architecturale

Le tableau suivant présente la configuration environnementale d'un cluster Kafka utilisant AWS FSx ONTAP.

Composant de la plateforme	Configuration de l'environnement
Kafka 3.2.3	• 3 x gardiens de zoo – t2.small • 3 serveurs courtiers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producteur/consommateur — c5n.2xlarge *
Système d'exploitation sur tous les nœuds	RHEL8.6
AWS FSx ONTAP	Multi-AZ avec un débit de 4 Go/s et 160 000 IOPS

Configuration de NetApp FSx ONTAP

1. Pour nos tests initiaux, nous avons créé un système de fichiers FSx ONTAP avec 2 To de capacité et 40 000 IOP pour un débit de 2 Go/s.

```
[root@ip-172-31-33-69 ~]# aws fsx create-file-system --region us-east-2
--storage-capacity 2048 --subnet-ids <desired subnet 1> subnet-<desired
subnet 2> --file-system-type ONTAP --ontap-configuration
DeploymentType=MULTI_AZ_HA_1,ThroughputCapacity=2048,PreferredSubnetId=<
desired primary subnet>,FsxAdminPassword=<new
password>,DiskIopsConfiguration="{Mode=USER_PROVISIONED,Iops=40000"}
```

   Dans notre exemple, nous déployons FSx ONTAP via l'AWS CLI. Vous devrez personnaliser davantage la commande dans votre environnement selon vos besoins. FSx ONTAP peut également être déployé et géré via la console AWS pour une expérience de déploiement plus simple et plus rationalisée avec moins de saisie de ligne de commande.

   Dans FSx ONTAP, le nombre maximal d'IOPS réalisables pour un système de fichiers à débit de 2 Go/s dans notre région de test (US-East-1) est de 80 000 IOPS. Le nombre total d'I/O maximal pour un système de fichiers FSx ONTAP est de 160 000 I/O, ce qui nécessite un déploiement de débit de 4 Go/s pour y parvenir, ce que nous démontrerons plus tard dans ce document.

   Pour plus d'informations sur les spécifications de performances de FSx ONTAP , n'hésitez pas à consulter la documentation AWS FSx ONTAP ici : https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html .

   La syntaxe détaillée de la ligne de commande pour FSx « create-file-system » peut être trouvée ici : https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

   Par exemple, vous pouvez spécifier une clé KMS spécifique par opposition à la clé principale AWS FSx par défaut qui est utilisée lorsqu'aucune clé KMS n'est spécifiée.

2. Lors de la création du système de fichiers FSx ONTAP , attendez que le statut « LifeCycle » passe à « DISPONIBLE » dans votre retour JSON après avoir décrit votre système de fichiers comme suit :

```
[root@ip-172-31-33-69 ~]# aws fsx describe-file-systems  --region us-
east-1 --file-system-ids fs-02ff04bab5ce01c7c
```

3. Validez les informations d'identification en vous connectant à FSx ONTAP SSH avec l'utilisateur fsxadmin : Fsxadmin est le compte administrateur par défaut pour les systèmes de fichiers FSx ONTAP lors de la création. Le mot de passe pour fsxadmin est le mot de passe qui a été configuré lors de la première création du système de fichiers, soit dans la console AWS, soit avec l'AWS CLI, comme nous l'avons fait à l'étape 1.

```
[root@ip-172-31-33-69 ~]# ssh fsxadmin@198.19.250.244
The authenticity of host '198.19.250.244 (198.19.250.244)' can't be
established.
ED25519 key fingerprint is
SHA256:mgCyRXJfWRc2d/jOjFbMBsUcYOWjxoIky0ltHvVDL/Y.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '198.19.250.244' (ED25519) to the list of
known hosts.
(fsxadmin@198.19.250.244) Password:

This is your first recorded login.
```

4. Une fois vos informations d'identification validées, créez la machine virtuelle de stockage sur le système de fichiers FSx ONTAP

```
[root@ip-172-31-33-69 ~]# aws fsx --region us-east-1 create-storage-
virtual-machine --name svmkafkatest --file-system-id fs-
02ff04bab5ce01c7c
```

Une machine virtuelle de stockage (SVM) est un serveur de fichiers isolé avec ses propres informations d'identification administratives et points de terminaison pour administrer et accéder aux données dans les volumes FSx ONTAP et fournit une multilocation FSx ONTAP .

5. Une fois que vous avez configuré votre machine virtuelle de stockage principale, connectez-vous au système de fichiers FSx ONTAP nouvellement créé et créez des volumes dans la machine virtuelle de stockage à l'aide de l'exemple de commande ci-dessous et de la même manière, nous créons 6 volumes pour cette validation. Sur la base de notre validation, conservez le constituant par défaut (8) ou moins de constituants qui offriront de meilleures performances à Kafka.

```
FsxId02ff04bab5ce01c7c::*> volume create -volume kafkafsxN1 -state
online -policy default -unix-permissions ---rwxr-xr-x -junction-active
true -type RW -snapshot-policy none  -junction-path /kafkafsxN1 -aggr
-list aggr1
```

6. Nous aurons besoin de capacités supplémentaires dans nos volumes pour nos tests. Étendez la taille du volume à 2 To et montez-le sur le chemin de jonction.

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN1 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN1" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN2 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN2" size set to 2.10t.
```

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN3 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN3" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN4 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN4" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN5 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN5" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN6 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN6" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume show -vserver svmkafkatest -volume *
Vserver   Volume       Aggregate    State      Type       Size
Available Used%
--------- ------------ ------------ ---------- ---- ----------
---------- -----
svmkafkatest
          kafkafsxN1   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN2   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN3   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN4   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN5   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN6   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          svmkafkatest_root
                       aggr1        online     RW          1GB
968.1MB    0%
7 entries were displayed.

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN1 -junction
-path /kafkafsxN1

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN2 -junction
-path /kafkafsxN2
```

```
FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN3 -junction
-path /kafkafsxN3

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN4 -junction
-path /kafkafsxN4

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN5 -junction
-path /kafkafsxN5

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN6 -junction
-path /kafkafsxN6
```

Dans FSx ONTAP, les volumes peuvent être provisionnés de manière dynamique. Dans notre exemple, la capacité totale du volume étendu dépasse la capacité totale du système de fichiers. Nous devrons donc étendre la capacité totale du système de fichiers afin de débloquer une capacité de volume provisionnée supplémentaire que nous démontrerons dans notre prochaine étape.

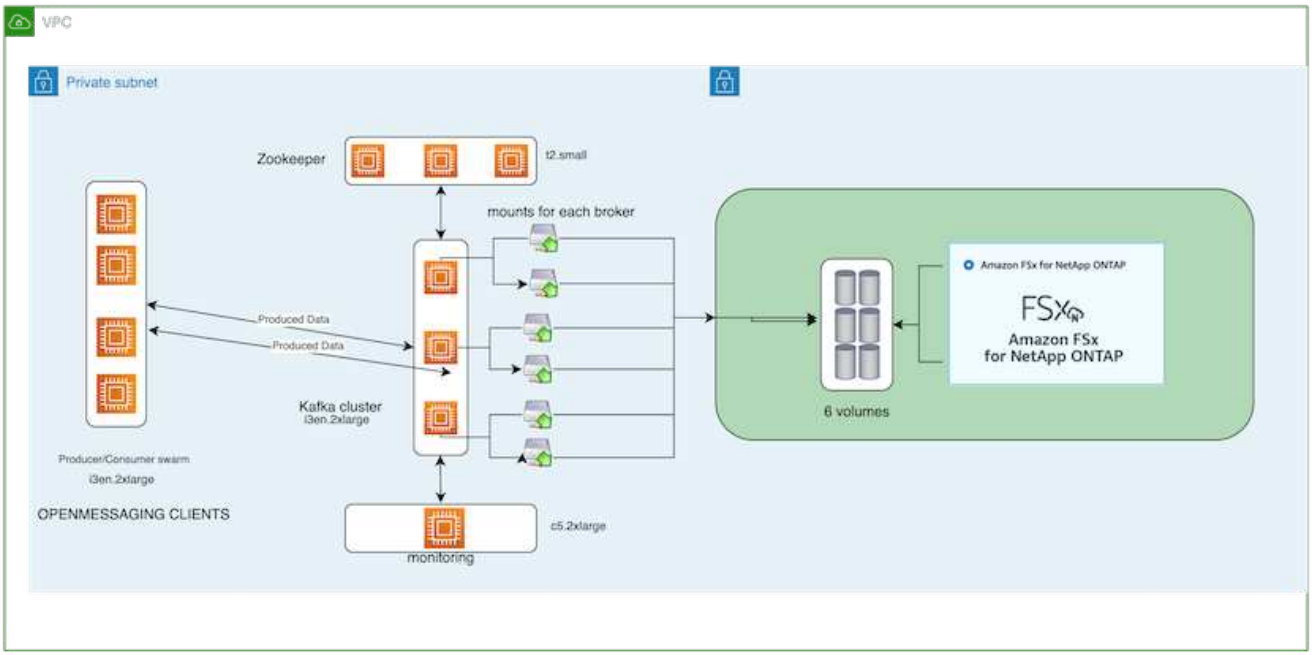
7. Ensuite, pour des performances et une capacité supplémentaires, nous étendons la capacité de débit de FSx ONTAP de 2 Go/s à 4 Go/s et les IOPS à 160 000, et la capacité à 5 To.

```
[root@ip-172-31-33-69 ~]# aws fsx update-file-system --region us-east-1
--storage-capacity 5120 --ontap-configuration
'ThroughputCapacity=4096,DiskIopsConfiguration={Mode=USER_PROVISIONED,Io
ps=160000}' --file-system-id fs-02ff04bab5ce01c7c
```

La syntaxe détaillée de la ligne de commande pour FSx « update-file-system » peut être trouvée ici :https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. Les volumes FSx ONTAP sont montés avec nconnect et les options par défaut dans les courtiers Kafka

L'image suivante montre notre architecture finale de notre cluster Kafka basé sur FSx ONTAP :



- Calculer. Nous avons utilisé un cluster Kafka à trois nœuds avec un ensemble zookeeper à trois nœuds exécuté sur des serveurs dédiés. Chaque courtier disposait de six points de montage NFS sur six volumes sur l'instance FSx ONTAP .
- Surveillance. Nous avons utilisé deux nœuds pour une combinaison Prometheus-Grafana. Pour générer des charges de travail, nous avons utilisé un cluster à trois nœuds distinct qui pouvait produire et consommer sur ce cluster Kafka.
- Stockage. Nous avons utilisé un FSx ONTAP avec six volumes de 2 To montés. Le volume a ensuite été exporté vers le courtier Kafka avec un montage NFS. Les volumes FSx ONTAP sont montés avec 16 sessions nconnect et des options par défaut dans les courtiers Kafka.

Configurations d'analyse comparative OpenMessage.

Nous avons utilisé la même configuration que celle utilisée pour les volumes NetApp Cloud ONTAP et leurs détails sont ici - lien : kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

Méthodologie des tests

1. Un cluster Kafka a été provisionné conformément à la spécification décrite ci-dessus à l'aide de Terraform et d'ansible. Terraform est utilisé pour créer l'infrastructure à l'aide d'instances AWS pour le cluster Kafka et Ansible construit le cluster Kafka sur celles-ci.
2. Une charge de travail OMB a été déclenchée avec la configuration de charge de travail décrite ci-dessus et le pilote Sync.

```
sudo bin/benchmark -drivers driver-kafka/kafka-sync.yaml workloads/1-
topic-100-partitions-1kb.yaml
```

3. Une autre charge de travail a été déclenchée avec le pilote de débit avec la même configuration de charge de travail.

```
sudo bin/benchmark -drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

Observation

Deux types de pilotes différents ont été utilisés pour générer des charges de travail afin d'évaluer les performances d'une instance Kafka exécutée sur NFS. La différence entre les pilotes est la propriété de vidage du journal.

Pour un facteur de réplication Kafka 1 et le FSx ONTAP:

- Débit total généré de manière cohérente par le pilote Sync : ~ 3 218 Mo/s et performances maximales à ~ 3 652 Mo/s.
- Débit total généré de manière cohérente par le pilote de débit : ~ 3 679 Mo/s et performances maximales à ~ 3 908 Mo/s.

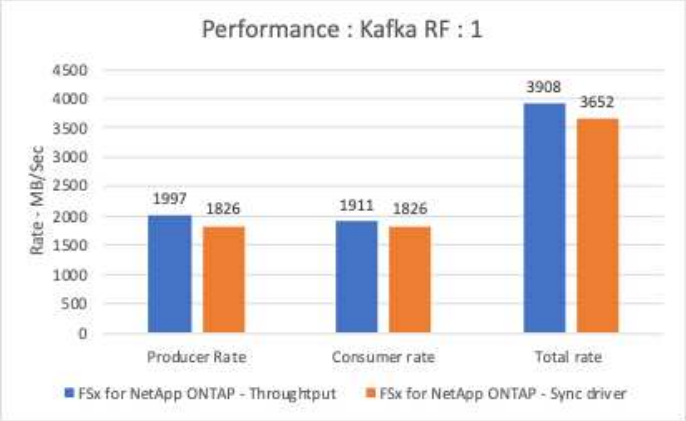
Pour Kafka avec facteur de réplication 3 et FSx ONTAP :

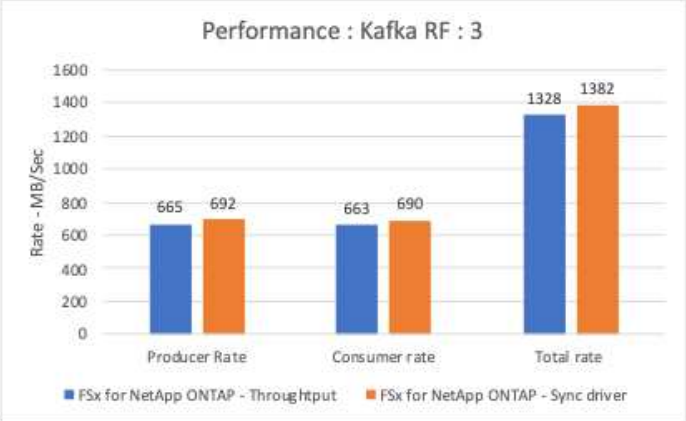
- Débit total généré de manière cohérente par le pilote Sync : ~ 1 252 Mo/s et performances maximales à ~ 1 382 Mo/s.
- Débit total généré de manière cohérente par le pilote de débit : environ 1 218 Mo/s et performances maximales d'environ 1 328 Mo/s.

Dans le facteur de réplication Kafka 3, l'opération de lecture et d'écriture s'est produite trois fois sur le FSx ONTAP. Dans le facteur de réplication Kafka 1, l'opération de lecture et d'écriture s'est produite une fois sur le FSx ONTAP. Ainsi, dans les deux validations, nous avons pu atteindre le débit maximal de 4 Go/s.

Le pilote de synchronisation peut générer un débit constant lorsque les journaux sont vidés instantanément sur le disque, tandis que le pilote de débit génère des rafales de débit lorsque les journaux sont validés sur le disque en masse.

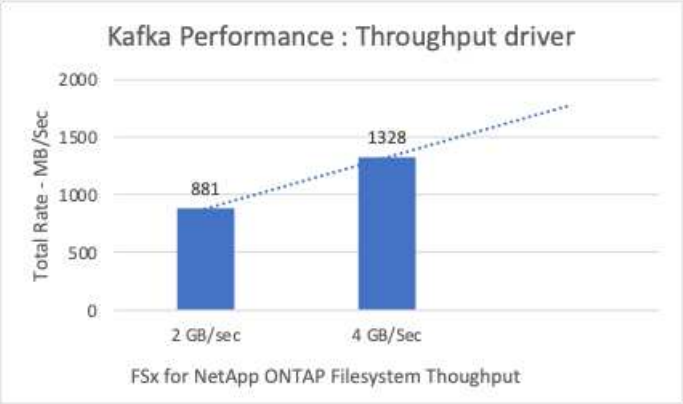
Ces numéros de débit sont générés pour la configuration AWS donnée. Pour des exigences de performances plus élevées, les types d'instances peuvent être mis à l'échelle et optimisés davantage pour de meilleurs chiffres de débit. Le débit total ou le taux total est la combinaison du taux du producteur et du taux du consommateur.

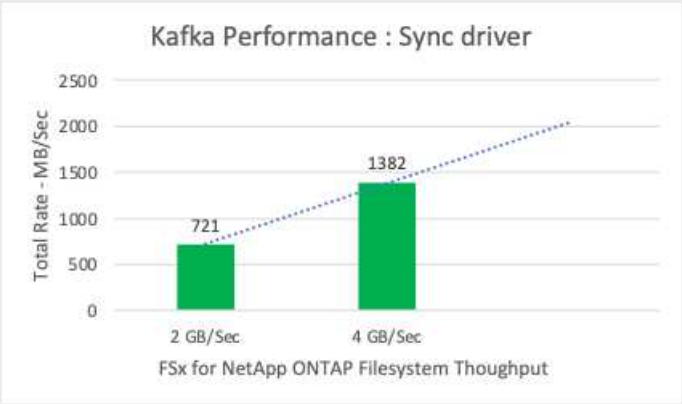




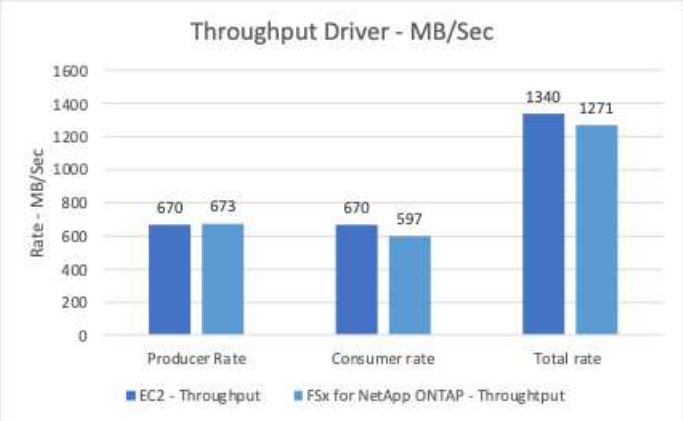
Le graphique ci-dessous montre les performances de FSx ONTAP à 2 Go/s et de 4 Go/s pour le facteur de réplication Kafka 3. Le facteur de réplication 3 effectue l'opération de lecture et d'écriture trois fois sur le

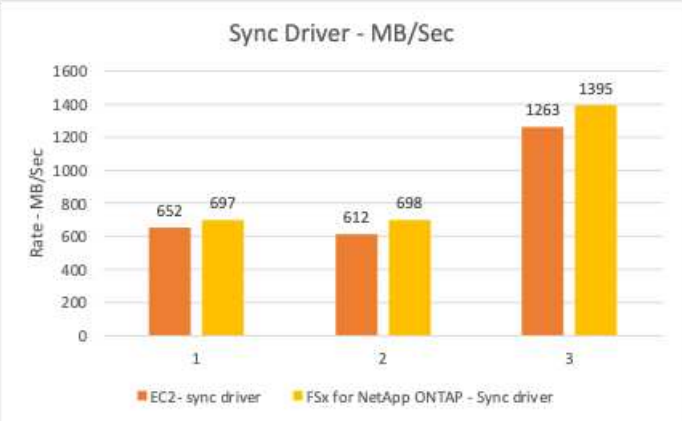
stockage FSx ONTAP . Le débit total du pilote de débit est de 881 Mo/s, ce qui permet de lire et d'écrire l'opération Kafka à environ 2,64 Go/s sur le système de fichiers FSx ONTAP à 2 Go/s et le débit total du pilote de débit est de 1 328 Mo/s, ce qui permet de lire et d'écrire l'opération Kafka à environ 3,98 Go/s. Les performances de Kafka sont linéaires et évolutives en fonction du débit FSx ONTAP .





Le graphique ci-dessous montre les performances entre l'instance EC2 et FSx ONTAP (facteur de réplication Kafka : 3)

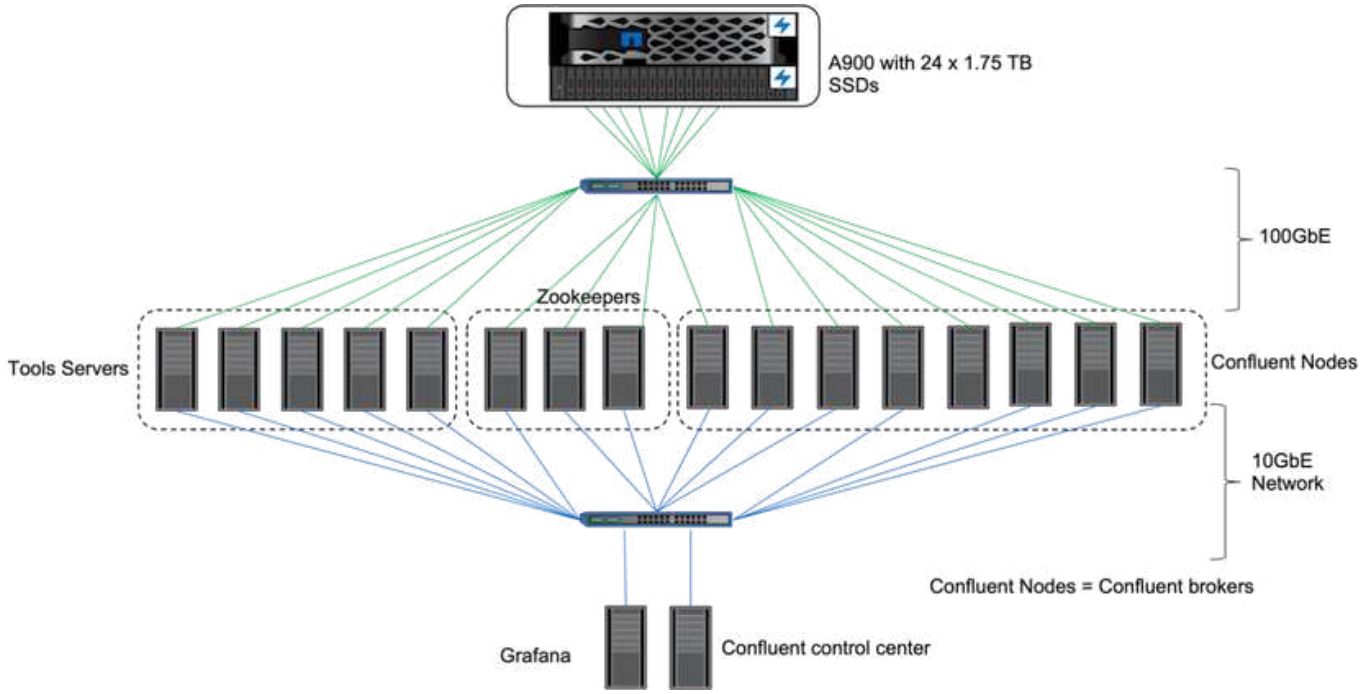




Aperçu des performances et validation avec AFF A900 sur site

Sur site, nous avons utilisé le contrôleur de stockage NetApp AFF A900 avec ONTAP 9.12.1RC1 pour valider les performances et la mise à l'échelle d'un cluster Kafka. Nous avons utilisé le même banc d'essai que dans nos précédentes meilleures pratiques de stockage hiérarchisé avec ONTAP et AFF.

Nous avons utilisé Confluent Kafka 6.2.0 pour évaluer l' AFF A900. Le cluster comprend huit nœuds de courtier et trois nœuds de gardien de zoo. Pour les tests de performances, nous avons utilisé cinq nœuds de travail OMB.



Configuration de stockage

Nous avons utilisé des instances NetApp FlexGroups pour fournir un espace de noms unique pour les répertoires de journaux, simplifiant ainsi la récupération et la configuration. Nous avons utilisé NFSv4.1 et pNFS pour fournir un accès direct aux données des segments de journaux.

Réglage du client

Chaque client a monté l'instance FlexGroup avec la commande suivante.

```
mount -t nfs -o vers=4.1,nconnect=16 172.30.0.121:/kafka_vol01
/data/kafka_vol01
```

De plus, nous avons augmenté le `max_session_slots` à partir de la valeur par défaut `64` à `180` . Cela correspond à la limite d'emplacement de session par défaut dans ONTAP.

Réglage du courtier Kafka

Afin de maximiser le débit du système testé, nous avons considérablement augmenté les paramètres par défaut pour certains pools de threads clés. Nous vous recommandons de suivre les meilleures pratiques de Confluent Kafka pour la plupart des configurations. Ce réglage a été utilisé pour maximiser la concurrence des E/S en attente vers le stockage. Ces paramètres peuvent être ajustés pour correspondre aux ressources de calcul et aux attributs de stockage de votre courtier.

```
num.io.threads=96
num.network.threads=96
background.threads=20
num.replica.alter.log.dirs.threads=40
num.replica.fetchers=20
queued.max.requests=2000
```

Méthodologie de test du générateur de charge de travail

Nous avons utilisé les mêmes configurations OMB que pour les tests cloud pour le pilote de débit et la configuration de rubrique.

1. Une instance FlexGroup a été provisionnée à l'aide d'Ansible sur un cluster AFF .

```
---
- name: Set up kafka broker processes
  hosts: localhost
  vars:
    ntap_hostname: 'hostname'
    ntap_username: 'user'
    ntap_password: 'password'
    size: 10
    size_unit: tb
    vserver: vs1
    state: present
    https: true
    export_policy: default
    volumes:
      - name: kafka_fg_vol01
        aggr: ["aggr1_a", "aggr2_a", "aggr1_b", "aggr2_b"]
        path: /kafka_fg_vol01
  tasks:
    - name: Edit volumes
      netapp.ontap.na_ontap_volume:
        state: "{{ state }}"
        name: "{{ item.name }}"
        aggr_list: "{{ item.aggr }}"
        aggr_list_multiplier: 8
        size: "{{ size }}"
        size_unit: "{{ size_unit }}"
        vserver: "{{ vserver }}"
        snapshot_policy: none
        export_policy: default
        junction_path: "{{ item.path }}"
        qos_policy_group: none
        wait_for_completion: True
        hostname: "{{ ntap_hostname }}"
        username: "{{ ntap_username }}"
        password: "{{ ntap_password }}"
        https: "{{ https }}"
        validate_certs: false
      connection: local
      with_items: "{{ volumes }}"
```

2. pNFS a été activé sur le SVM ONTAP .

```
vserver modify -vserver vs1 -v4.1-pnfs enabled -tcp-max-xfer-size 262144
```

3. La charge de travail a été déclenchée avec le pilote de débit utilisant la même configuration de charge de travail que pour Cloud Volumes ONTAP. Voir la section "Performances à l'état stable " ci-dessous. La charge de travail utilisait un facteur de réplication de 3, ce qui signifie que trois copies de segments de journaux étaient conservées dans NFS.

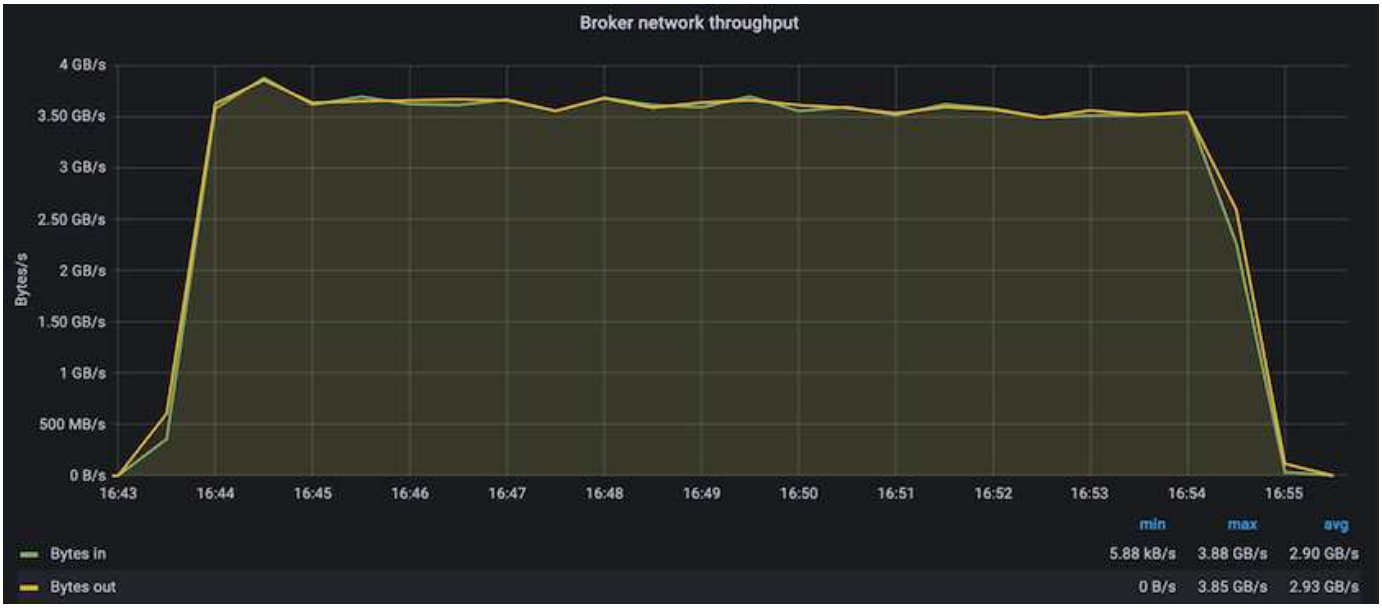
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. Enfin, nous avons réalisé des mesures à l'aide d'un backlog pour évaluer la capacité des consommateurs à rattraper les derniers messages. L'OMB construit un backlog en mettant les consommateurs en pause au début d'une mesure. Cela produit trois phases distinctes : la création du backlog (trafic réservé aux producteurs), l'épuisement du backlog (une phase très axée sur les consommateurs dans laquelle les consommateurs rattrapent les événements manqués dans un sujet) et l'état stable. Voir la section "Performances extrêmes et exploration des limites de stockage " pour plus d'informations.

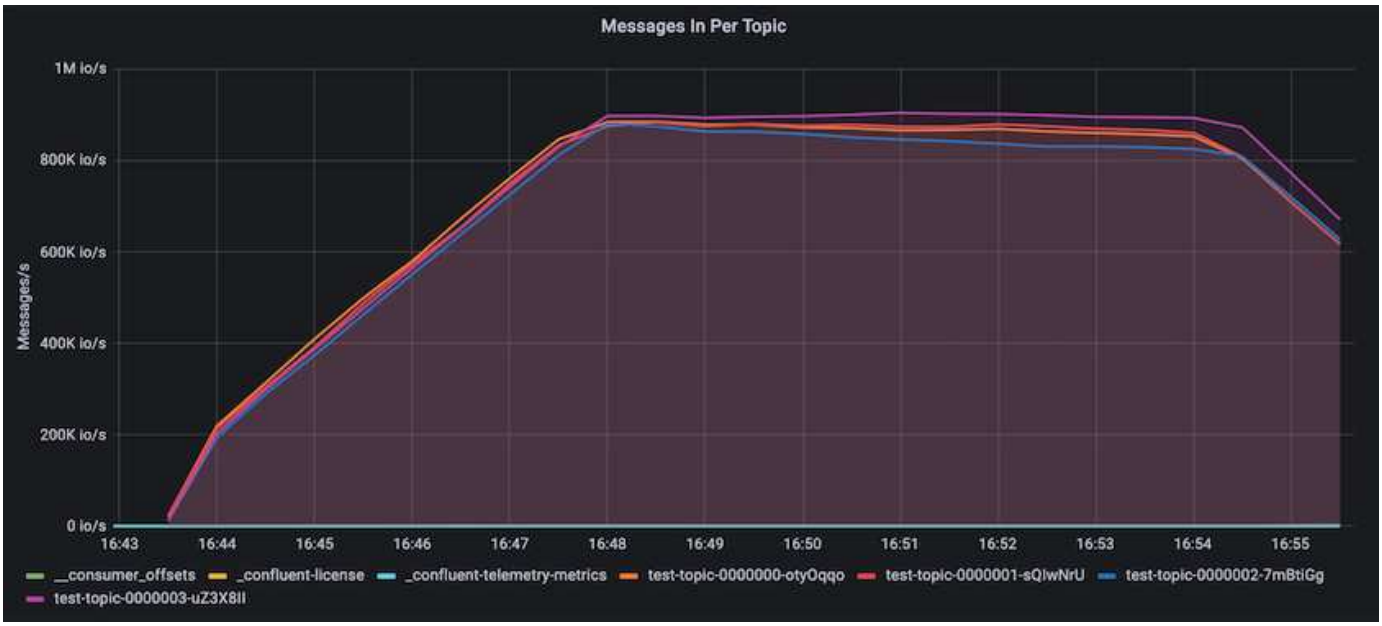
Performances à l'état stable

Nous avons évalué l' AFF A900 à l'aide de l'OpenMessaging Benchmark pour fournir une comparaison similaire à celle de Cloud Volumes ONTAP dans AWS et DAS dans AWS. Toutes les valeurs de performance représentent le débit du cluster Kafka au niveau du producteur et du consommateur.

Les performances en régime permanent avec Confluent Kafka et l' AFF A900 ont atteint un débit moyen de plus de 3,4 Gbit/s pour les producteurs et les consommateurs. Cela représente plus de 3,4 millions de messages sur l'ensemble du cluster Kafka. En visualisant le débit soutenu en octets par seconde pour BrokerTopicMetrics, nous constatons les excellentes performances en régime permanent et le trafic pris en charge par l' AFF A900.



Cela correspond bien à la vue des messages délivrés par sujet. Le graphique suivant fournit une répartition par sujet. Dans la configuration testée, nous avons vu près de 900 000 messages par sujet sur quatre sujets.

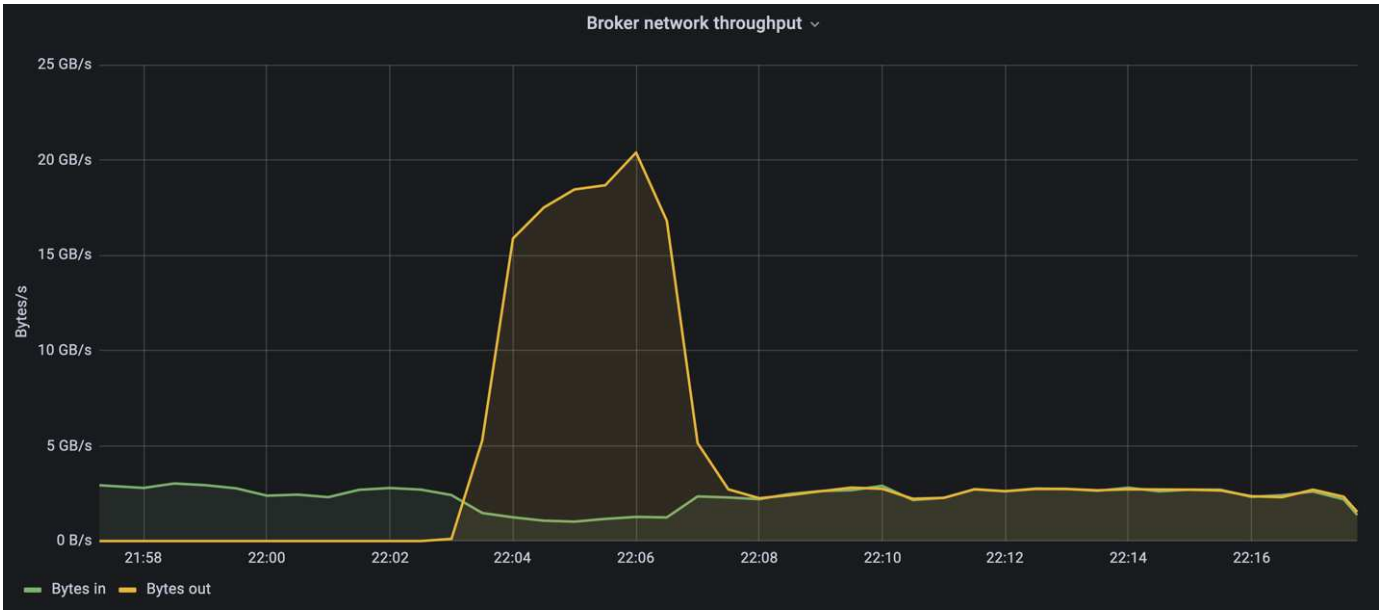


Performances extrêmes et exploration des limites de stockage

Pour AFF, nous avons également testé avec OMB en utilisant la fonctionnalité backlog. La fonctionnalité de backlog suspend les abonnements des consommateurs pendant qu'un backlog d'événements est constitué dans le cluster Kafka. Au cours de cette phase, seul le trafic du producteur se produit, ce qui génère des événements qui sont validés dans les journaux. Cela émule le plus étroitement les flux de travail de traitement par lots ou d'analyse hors ligne ; dans ces flux de travail, les abonnements des consommateurs sont démarrés et doivent lire les données historiques qui ont déjà été expulsées du cache du courtier.

Pour comprendre les limites de stockage sur le débit du consommateur dans cette configuration, nous avons mesuré la phase réservée au producteur pour comprendre la quantité de trafic d'écriture que l'A900 pouvait absorber. Voir la section suivante "Guide de dimensionnement " pour comprendre comment exploiter ces données.

Au cours de la partie réservée aux producteurs de cette mesure, nous avons constaté un débit de pointe élevé qui a repoussé les limites des performances de l'A900 (lorsque les autres ressources du courtier n'étaient pas saturées et ne desservaient pas le trafic des producteurs et des consommateurs).



Nous avons augmenté la taille du message à 16 Ko pour cette mesure afin de limiter les frais généraux par message et de maximiser le débit de stockage vers les points de montage NFS.

```
messageSize: 16384
consumerBacklogSizeGB: 4096
```

Le cluster Confluent Kafka a atteint un débit de production maximal de 4,03 Gbit/s.

```
18:12:23.833 [main] INFO WorkloadGenerator - Pub rate 257759.2 msg/s /
4027.5 MB/s | Pub err     0.0 err/s …
```

Une fois que l'OMB a terminé de remplir le backlog des événements, le trafic consommateur a été redémarré. Lors des mesures avec drainage du backlog, nous avons observé un débit consommateur maximal de plus de 20 Gbit/s sur tous les sujets. Le débit combiné du volume NFS stockant les données du journal OMB approchait environ 30 Gbit/s.

Guide de dimensionnement

Amazon Web Services propose une "guide des tailles" pour le dimensionnement et la mise à l'échelle des clusters Kafka.

Ce dimensionnement fournit une formule utile pour déterminer les besoins en débit de stockage de votre cluster Kafka :

Pour un débit agrégé produit dans le cluster de tcluster avec un facteur de réplication de r, le débit reçu par le stockage du courtier est le suivant :

```
t[storage] = t[cluster]/#brokers + t[cluster]/#brokers * (r-1)
           = t[cluster]/#brokers * r
```

Cela peut être encore simplifié :

```
max(t[cluster]) <= max(t[storage]) * #brokers/r
```

L'utilisation de cette formule vous permet de sélectionner la plate-forme ONTAP appropriée à vos besoins en matière de niveau chaud Kafka.

Le tableau suivant explique le débit de production prévu pour l'A900 avec différents facteurs de réplication :

Facteur de réplication	Débit du producteur (GPps)
3 (mesuré)	3,4
2	5,1
1	10,2

Conclusion

La solution NetApp au problème de renommage stupide fournit une forme de stockage simple, peu coûteuse et gérée de manière centralisée pour les charges de travail qui étaient auparavant incompatibles avec NFS.

Ce nouveau paradigme permet aux clients de créer des clusters Kafka plus faciles à gérer, plus faciles à migrer et à mettre en miroir à des fins de reprise après sinistre et de protection des données. Nous avons également constaté que NFS offre des avantages supplémentaires tels qu'une utilisation réduite du processeur et un temps de récupération plus rapide, une efficacité de stockage considérablement améliorée et de meilleures performances grâce à NetApp ONTAP.

Où trouver des informations supplémentaires

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

- Qu'est-ce qu'Apache Kafka ?

  "https://www.confluent.io/what-is-apache-kafka/"

- Qu'est-ce qu'un renommage idiot ?

  "https://linux-nfs.org/wiki/index.php/Server-side_silly_rename"

- ONATP est conçu pour les applications de streaming.

  "https://www.netapp.com/blog/ontap-ready-for-streaming-applications/"

- Documentation produit NetApp

  "https://www.netapp.com/support-and-training/documentation/"

- Qu'est-ce que NFS ?

  "https://en.wikipedia.org/wiki/Network_File_System"

- Qu'est-ce que la réaffectation de partition Kafka ?

  "https://docs.cloudera.com/runtime/7.2.10/kafka-managing/topics/kafka-manage-cli-reassign-overview.html"

- Qu'est-ce que le benchmark OpenMessaging ?

  "https://openmessaging.cloud/"

- Comment migrer un courtier Kafka ?

  "https://medium.com/@sanchitbansal26/how-to-migrate-kafka-cluster-with-no-downtime-58c216129058"

- Comment surveiller le courtier Kafka avec Prometheus ?

  https://www.confluent.io/blog/monitor-kafka-clusters-with-prometheus-grafana-and-confluent/

- Plateforme gérée pour Apache Kafka

  https://www.instaclustr.com/platform/managed-apache-kafka/

- Prise en charge d'Apache Kafka

  https://www.instaclustr.com/support-solutions/kafka-support/

- Services de conseil pour Apache Kafka

  https://www.instaclustr.com/services/consulting/

Informations sur le copyright

Informations sur les marques commerciales