Carichi di lavoro Apache Kafka con storage NetApp NFS

NetApp artificial intelligence solutions

NetApp
August 18, 2025

Sommario

Carichi di lavoro Apache Kafka con storage NetApp NFS

TR-4947: Carico di lavoro Apache Kafka con storage NetApp NFS - Validazione funzionale e prestazioni

Shantanu Chakole, Karthikeyan Nagalingam e Joe Scott, NetApp

Kafka è un sistema di messaggistica distribuito di tipo publish-subscribe con una coda robusta in grado di accettare grandi quantità di dati di messaggi. Con Kafka, le applicazioni possono scrivere e leggere dati sugli argomenti in modo molto rapido. Grazie alla sua tolleranza agli errori e alla sua scalabilità, Kafka viene spesso utilizzato nel settore dei big data come metodo affidabile per acquisire e spostare rapidamente numerosi flussi di dati. I casi d'uso includono l'elaborazione di flussi, il monitoraggio delle attività del sito web, la raccolta e il monitoraggio delle metriche, l'aggregazione dei log, l'analisi in tempo reale e così via.

Sebbene le normali operazioni Kafka su NFS funzionino bene, il problema assurdo della ridenominazione causa l'arresto anomalo dell'applicazione durante il ridimensionamento o il ripartizionamento di un cluster Kafka in esecuzione su NFS. Si tratta di un problema significativo perché un cluster Kafka deve essere ridimensionato o ripartizionato per scopi di bilanciamento del carico o di manutenzione. Puoi trovare ulteriori dettagli "Qui" .

Il presente documento descrive i seguenti argomenti:

- Il problema della ridenominazione sciocca e la convalida della soluzione
- Riduzione dell'utilizzo della CPU per ridurre il tempo di attesa I/O
- Tempi di recupero più rapidi del broker Kafka
- Prestazioni nel cloud e on-premise

Perché utilizzare l'archiviazione NFS per i carichi di lavoro Kafka?

I carichi di lavoro Kafka nelle applicazioni di produzione possono trasmettere in streaming enormi quantità di dati tra le applicazioni. Questi dati vengono conservati e archiviati nei nodi broker Kafka nel cluster Kafka. Kafka è noto anche per la disponibilità e il parallelismo, che ottiene suddividendo gli argomenti in partizioni e replicando poi tali partizioni in tutto il cluster. Ciò significa che l'enorme quantità di dati che fluisce attraverso un cluster Kafka viene generalmente moltiplicata in termini di dimensioni. NFS rende il ribilanciamento dei dati molto rapido e semplice man mano che cambia il numero di broker. Negli ambienti di grandi dimensioni, il ribilanciamento dei dati su DAS quando cambia il numero di broker richiede molto tempo e, nella maggior parte degli ambienti Kafka, il numero di broker cambia frequentemente.

Altri vantaggi includono quanto segue:

- **Scadenza.** NFS è un protocollo maturo, il che significa che la maggior parte degli aspetti relativi alla sua implementazione, protezione e utilizzo sono ben compresi.
- **Aprire.** NFS è un protocollo aperto e il suo continuo sviluppo è documentato nelle specifiche Internet come protocollo di rete libero e aperto.

- **Economico.** NFS è una soluzione economica per la condivisione di file in rete, facile da configurare perché utilizza l'infrastruttura di rete esistente.
- **Gestione centralizzata.** La gestione centralizzata di NFS riduce la necessità di software e spazio su disco aggiuntivi sui sistemi dei singoli utenti.
- **Distribuito.** NFS può essere utilizzato come file system distribuito, riducendo la necessità di dispositivi di archiviazione su supporti rimovibili.

Perché NetApp per i carichi di lavoro Kafka?

L'implementazione NetApp NFS è considerata uno standard di riferimento per il protocollo e viene utilizzata in innumerevoli ambienti NAS aziendali. Oltre alla credibilità, NetApp offre anche i seguenti vantaggi:

- Affidabilità ed efficienza
- Scalabilità e prestazioni
- Alta disponibilità (partner HA in un cluster NetApp ONTAP)
- Protezione dei dati
  - **Ripristino di emergenza (NetApp SnapMirror).** Il tuo sito non funziona più oppure vuoi ripartire da un sito diverso e riprendere da dove eri rimasto.
  - Gestibilità del sistema di storage (amministrazione e gestione tramite NetApp OnCommand).
  - **Bilanciamento del carico.** Il cluster consente di accedere a volumi diversi da LIF di dati ospitati su nodi diversi.
  - **Operazioni non distruttive.** Gli spostamenti LIF o di volume sono trasparenti per i client NFS.

Soluzione NetApp per un problema di rinomina stupido per carichi di lavoro da NFS a Kafka

Kafka è stato sviluppato partendo dal presupposto che il file system sottostante sia compatibile con POSIX: ad esempio, XFS o Ext4. Il ribilanciamento delle risorse di Kafka rimuove i file mentre l'applicazione li sta ancora utilizzando. Un file system conforme a POSIX consente di procedere con l'annullamento del collegamento. Tuttavia, rimuove il file solo dopo che tutti i riferimenti al file sono scomparsi. Se il file system sottostante è collegato alla rete, il client NFS intercetta le chiamate di unlink e gestisce il flusso di lavoro. Poiché ci sono aperture in sospeso sul file che viene scollegato, il client NFS invia una richiesta di rinomina al server NFS e, all'ultima chiusura del file scollegato, esegue un'operazione di rimozione sul file rinominato. Questo comportamento è comunemente noto come NFS silly rename ed è orchestrato dal client NFS.

Qualsiasi broker Kafka che utilizzi l'archiviazione da un server NFSv3 riscontra problemi a causa di questo comportamento. Tuttavia, il protocollo NFSv4.x dispone di funzionalità per risolvere questo problema consentendo al server di assumersi la responsabilità dei file aperti e non collegati. I server NFS che supportano questa funzionalità opzionale comunicano la capacità di proprietà al client NFS al momento dell'apertura del file. Il client NFS interrompe quindi la gestione dell'unlink quando ci sono aperture in sospeso e consente al server di gestire il flusso. Sebbene la specifica NFSv4 fornisca linee guida per l'implementazione, fino ad ora non esistevano implementazioni di server NFS note che supportassero questa funzionalità opzionale.

Per risolvere il problema assurdo della ridenominazione, sono necessarie le seguenti modifiche al server NFS

e al client NFS:

- **Modifiche al client NFS (Linux).** Al momento dell'apertura del file, il server NFS risponde con un flag, indicando la capacità di gestire lo scollegamento dei file aperti. Le modifiche apportate al lato client NFS consentono al server NFS di gestire la disconnessione in presenza del flag. NetApp ha aggiornato il client NFS Linux open source con queste modifiche. Il client NFS aggiornato è ora generalmente disponibile in RHEL8.7 e RHEL9.1.
- **Modifiche al server NFS.** Il server NFS tiene traccia delle aperture. La disconnessione di un file aperto esistente è ora gestita dal server in modo da corrispondere alla semantica POSIX. Quando l'ultimo file aperto viene chiuso, il server NFS avvia la rimozione effettiva del file, evitando così il noioso processo di rinomina. Il server NFS ONTAP ha implementato questa funzionalità nella sua ultima versione, ONTAP 9.12.1.

Grazie alle modifiche sopra descritte al client e al server NFS, Kafka può sfruttare in tutta sicurezza tutti i vantaggi dell'archiviazione NFS collegata alla rete.

Validazione funzionale - Correzione stupida del cambio di nome

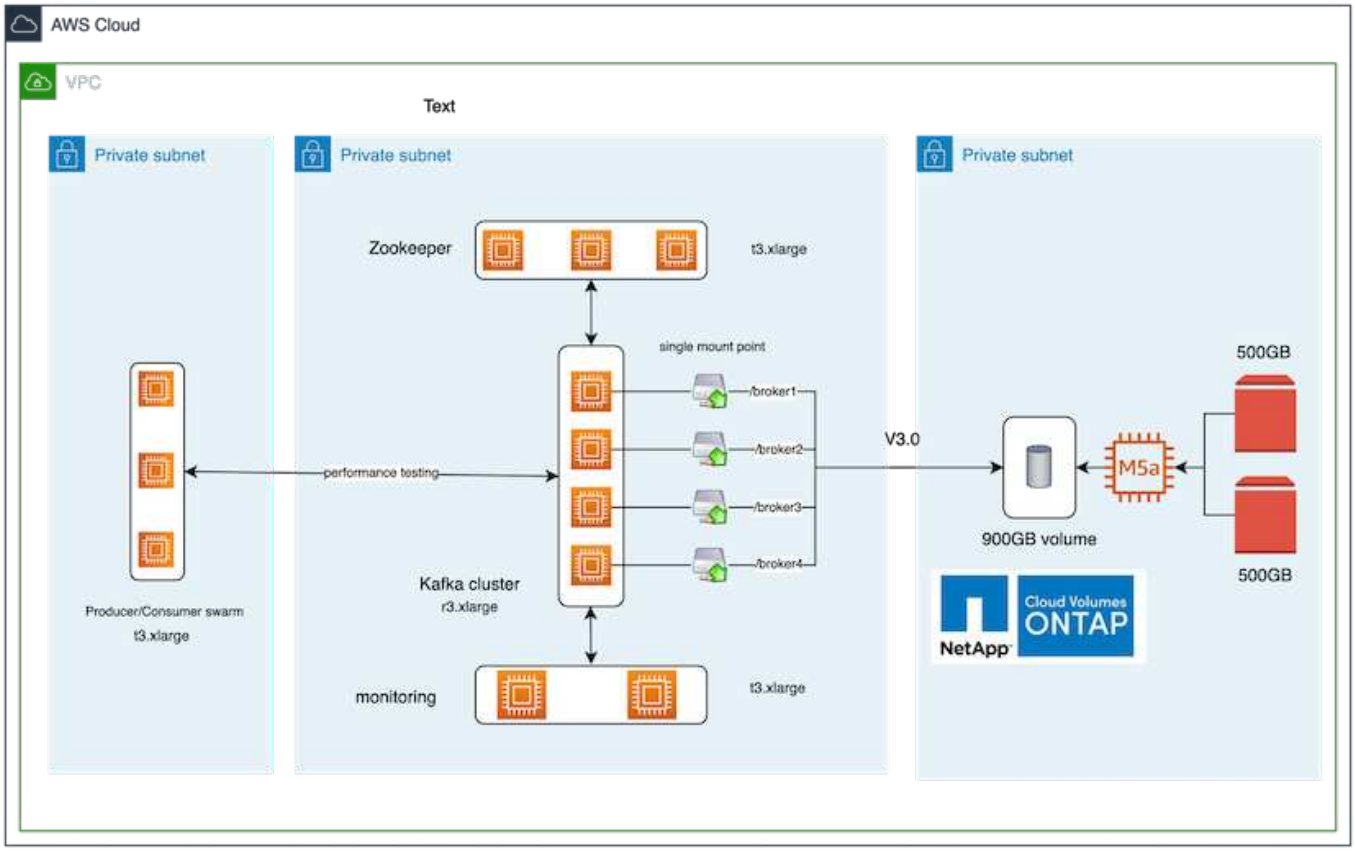
Per la convalida funzionale, abbiamo dimostrato che un cluster Kafka con un mount NFSv3 per l'archiviazione non riesce a eseguire operazioni Kafka come la ridistribuzione delle partizioni, mentre un altro cluster montato su NFSv4 con la correzione può eseguire le stesse operazioni senza interruzioni.

Impostazione di convalida

L'installazione viene eseguita su AWS. Nella tabella seguente sono riportati i diversi componenti della piattaforma e la configurazione ambientale utilizzati per la convalida.

Componente della piattaforma	Configurazione dell'ambiente
Piattaforma Confluent versione 7.2.1	• 3 guardiani dello zoo – t3.xlarge • 4 x server broker – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x centro di controllo – t3.xlarge • 3 x Produttore/consumatore
Sistema operativo su tutti i nodi	RHEL8.7 o successivo
Istanza ONTAP di NetApp Cloud Volumes ONTAP	Istanza a nodo singolo – M5.2xLarge

La figura seguente mostra la configurazione architettonica di questa soluzione.



Flusso architettonico

- **Calcolare.** Abbiamo utilizzato un cluster Kafka a quattro nodi con un ensemble zookeeper a tre nodi in esecuzione su server dedicati.
- **Monitoraggio.** Abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana.
- **Carico di lavoro.** Per generare carichi di lavoro, abbiamo utilizzato un cluster separato a tre nodi in grado di produrre e consumare da questo cluster Kafka.
- **Magazzinaggio.** Abbiamo utilizzato un'istanza ONTAP NetApp Cloud Volumes a nodo singolo con due volumi GP2 AWS-EBS da 500 GB collegati all'istanza. Questi volumi sono stati quindi esposti al cluster Kafka come singoli volumi NFSv4.1 tramite un LIF.

Per tutti i server sono state scelte le proprietà predefinite di Kafka. Lo stesso è stato fatto per lo sciame dei guardiani dello zoo.

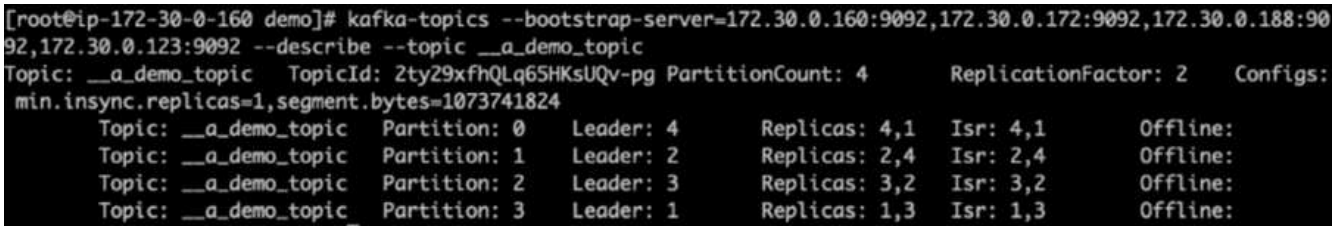
Metodologia di test

1. Aggiornamento `-is-preserve-unlink-enabled true` al volume di Kafka, come segue:

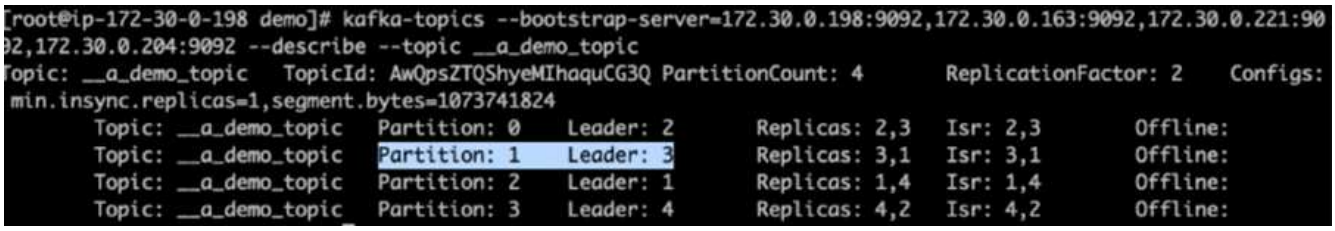
```
aws-shantanclastrecall-aws::*> volume create -vserver kafka_svm -volume
kafka_fg_vol01 -aggregate kafka_aggr -size 3500GB -state online -policy
kafka_policy -security-style unix -unix-permissions 0777 -junction-path
/kafka_fg_vol01 -type RW -is-preserve-unlink-enabled true
[Job 32] Job succeeded: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

2. Sono stati creati due cluster Kafka simili con la seguente differenza:
   - **Gruppo 1.** Il server backend NFS v4.1 che esegue ONTAP versione 9.12.1 pronto per la produzione era ospitato da un'istanza NetApp CVO. Sui broker sono stati installati RHEL 8.7/RHEL 9.1.
   - **Gruppo 2.** Il server NFS backend era un server Linux NFSv3 generico creato manualmente.
3. È stato creato un argomento dimostrativo su entrambi i cluster Kafka.

   Gruppo 1:



   Gruppo 2:

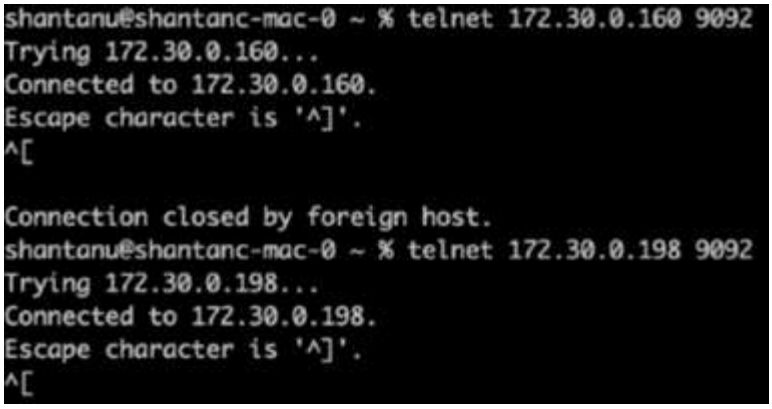


4. I dati sono stati caricati in questi argomenti appena creati per entrambi i cluster. Ciò è stato fatto utilizzando il toolkit producer-perf-test incluso nel pacchetto Kafka predefinito:

```
./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. È stato eseguito un controllo dello stato di salute per broker-1 per ciascuno dei cluster utilizzando telnet:
   - telnet `172.30.0.160 9092`
   - telnet `172.30.0.198 9092`

     Nella schermata successiva viene mostrato un controllo dello stato di salute riuscito per i broker su entrambi i cluster:



6. Per innescare la condizione di errore che causa l'arresto anomalo dei cluster Kafka che utilizzano volumi di archiviazione NFSv3, abbiamo avviato il processo di riassegnazione delle partizioni su entrambi i cluster. La riassegnazione della partizione è stata eseguita utilizzando `kafka-reassign-partitions.sh` . Il processo dettagliato è il seguente:
   a. Per riassegnare le partizioni per un argomento in un cluster Kafka, abbiamo generato la configurazione di riassegnazione proposta in formato JSON (questa operazione è stata eseguita per entrambi i cluster).

```
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. Il JSON di riassegnazione generato è stato quindi salvato in `/tmp/reassignment- file.json` .
   c. Il processo di riassegnazione della partizione effettiva è stato attivato dal seguente comando:

```
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. Dopo alcuni minuti, una volta completata la riassegnazione, un altro controllo dello stato di salute dei broker ha mostrato che il cluster che utilizzava volumi di storage NFSv3 aveva riscontrato un problema di ridenominazione e si era bloccato, mentre il Cluster 1 che utilizzava volumi di storage NetApp ONTAP NFSv4.1 con la correzione ha continuato a funzionare senza interruzioni.

```
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
```

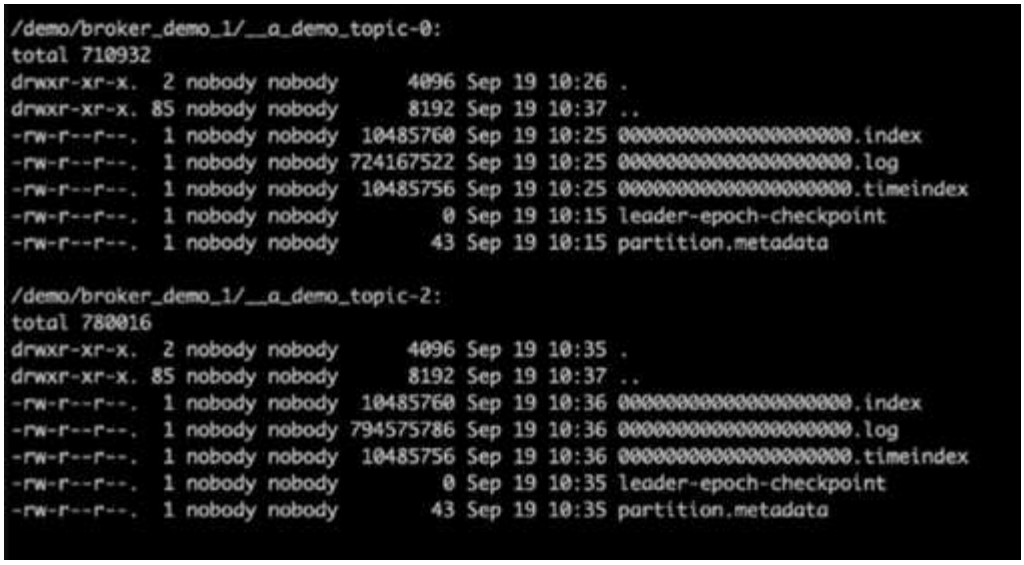
- Cluster1-Broker-1 è attivo.
- Cluster2-broker-1 è morto.

8. Dopo aver controllato le directory del registro di Kafka, è risultato chiaro che il Cluster 1 che utilizzava volumi di storage NetApp ONTAP NFSv4.1 con la correzione aveva un'assegnazione di partizioni pulita, mentre il Cluster 2 che utilizzava storage NFSv3 generico non aveva un'assegnazione di partizioni pulita a causa di stupidi problemi di ridenominazione, che hanno causato l'arresto anomalo. L'immagine seguente mostra il ribilanciamento delle partizioni del Cluster 2, che ha causato un problema di ridenominazione nello storage 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'immagine seguente mostra un ribilanciamento pulito della partizione del Cluster 1 utilizzando lo storage NetApp NFSv4.1.



Perché NetApp NFS per i carichi di lavoro Kafka?

Ora che esiste una soluzione per il problema di ridenominazione nello storage NFS con Kafka, puoi creare distribuzioni robuste che sfruttano lo storage NetApp ONTAP per il tuo carico di lavoro Kafka. Ciò non solo riduce significativamente i costi operativi, ma apporta anche i seguenti vantaggi ai cluster Kafka:

- **Utilizzo ridotto della CPU sui broker Kafka.** L'utilizzo di storage NetApp ONTAP disaggregato separa le operazioni di I/O del disco dal broker, riducendo così l'ingombro della CPU.
- **Tempi di recupero del broker più rapidi.** Poiché lo storage disaggregato NetApp ONTAP è condiviso tra i nodi broker Kafka, una nuova istanza di elaborazione può sostituire un broker non funzionante in qualsiasi momento e in una frazione del tempo rispetto alle distribuzioni Kafka convenzionali, senza dover ricostruire i dati.
- **Efficienza di archiviazione.** Poiché il livello di storage dell'applicazione è ora fornito tramite NetApp ONTAP, i clienti possono usufruire di tutti i vantaggi dell'efficienza di storage offerti da ONTAP, come la compressione dei dati in linea, la deduplicazione e la compattazione.

Questi vantaggi sono stati testati e convalidati in casi di prova che analizzeremo in dettaglio in questa sezione.

Utilizzo ridotto della CPU sul broker Kafka

Abbiamo scoperto che l'utilizzo complessivo della CPU è inferiore rispetto alla controparte DAS quando abbiamo eseguito carichi di lavoro simili su due cluster Kafka separati, identici nelle specifiche tecniche ma diversi nelle tecnologie di archiviazione. Non solo l'utilizzo complessivo della CPU è inferiore quando il cluster Kafka utilizza l'archiviazione ONTAP , ma l'aumento dell'utilizzo della CPU ha mostrato un gradiente più graduale rispetto a un cluster Kafka basato su DAS.

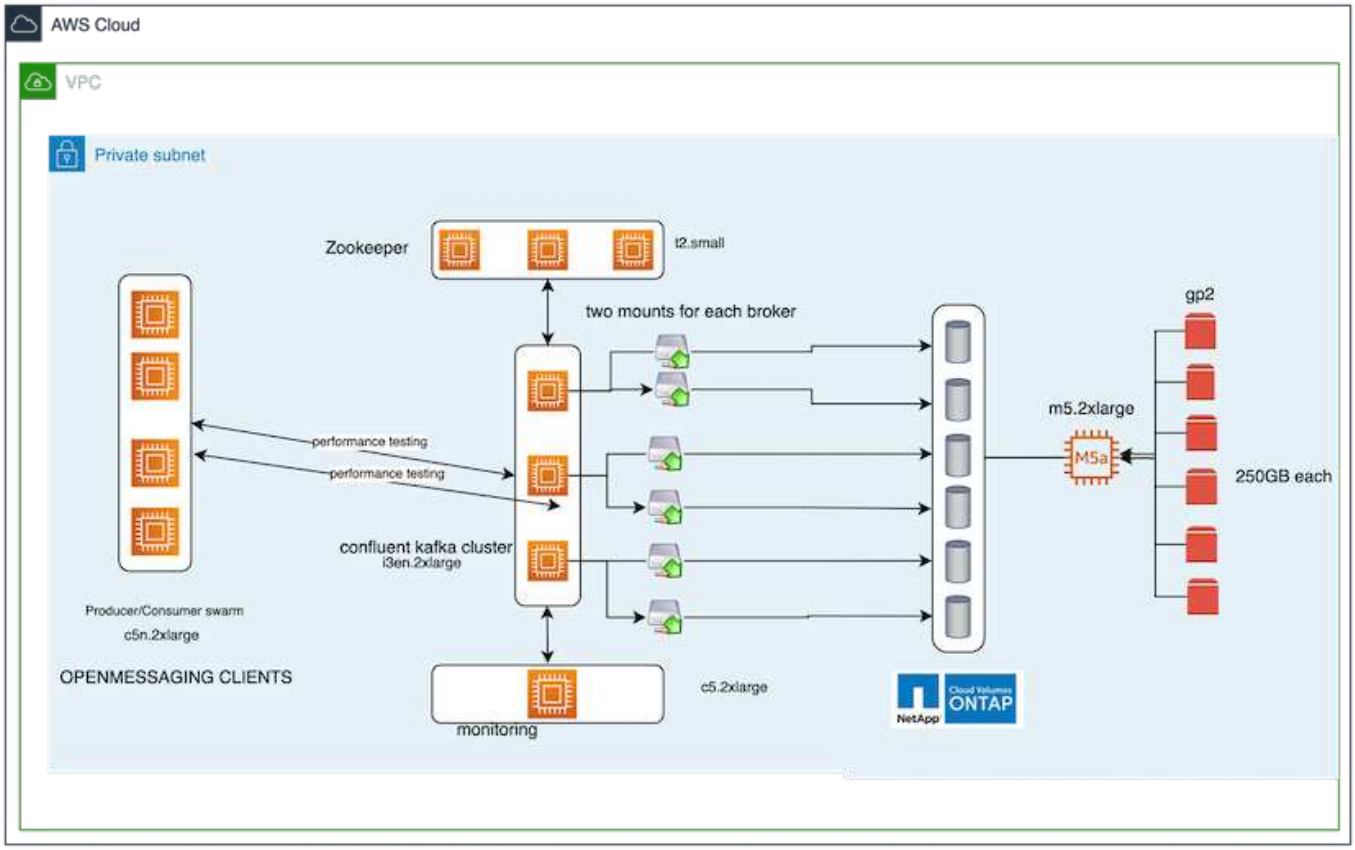
Configurazione architettonica

La tabella seguente mostra la configurazione ambientale utilizzata per dimostrare un utilizzo ridotto della CPU.

Componente della piattaforma	Configurazione dell'ambiente
Strumento di benchmarking di Kafka 3.2.3: OpenMessaging	• 3 guardiani dello zoo – t2.small • 3 server broker – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Produttore/Consumatore — c5n.2xlarge
Sistema operativo su tutti i nodi	RHEL 8.7 o successivo
Istanza ONTAP di NetApp Cloud Volumes ONTAP	Istanza a nodo singolo – M5.2xLarge

Strumento di benchmarking

Lo strumento di benchmarking utilizzato in questo caso di prova è il "OpenMessaging" struttura. OpenMessaging è indipendente dal fornitore e dal linguaggio; fornisce linee guida di settore per finanza, e-commerce, IoT e big data e aiuta a sviluppare applicazioni di messaggistica e streaming su sistemi e piattaforme eterogenei. La figura seguente illustra l'interazione dei client OpenMessaging con un cluster Kafka.



- **Calcolare.** Abbiamo utilizzato un cluster Kafka a tre nodi con un ensemble zookeeper a tre nodi in esecuzione su server dedicati. Ogni broker aveva due punti di montaggio NFSv4.1 su un singolo volume sull'istanza NetApp CVO tramite un LIF dedicato.
- **Monitoraggio.** Abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per generare carichi di lavoro, disponiamo di un cluster separato a tre nodi che può produrre e consumare da questo cluster Kafka.
- **Magazzinaggio.** Abbiamo utilizzato un'istanza ONTAP NetApp Cloud Volumes a nodo singolo con sei volumi GP2 AWS-EBS da 250 GB montati sull'istanza. Questi volumi sono stati quindi esposti al cluster

Kafka come sei volumi NFSv4.1 tramite LIF dedicati.

- **Configurazione.** I due elementi configurabili in questo caso di test erano i broker Kafka e i carichi di lavoro OpenMessaging.
  - **Configurazione del broker.** Per i broker Kafka sono state selezionate le seguenti specifiche. Abbiamo utilizzato un fattore di replicazione pari a 3 per tutte le misurazioni, come evidenziato di seguito.

```
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
```

- **Configurazione del carico di lavoro del benchmark OpenMessaging (OMB).** Sono state fornite le seguenti specifiche. Abbiamo specificato un tasso di produzione target, evidenziato di seguito.

```
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
```

Metodologia di test

1. Sono stati creati due cluster simili, ciascuno con il proprio set di swarm di cluster di benchmarking.
   - **Gruppo 1.** Cluster Kafka basato su NFS.
   - **Gruppo 2.** Cluster Kafka basato su DAS.

2. Utilizzando un comando OpenMessaging, sono stati attivati carichi di lavoro simili su ciascun cluster.

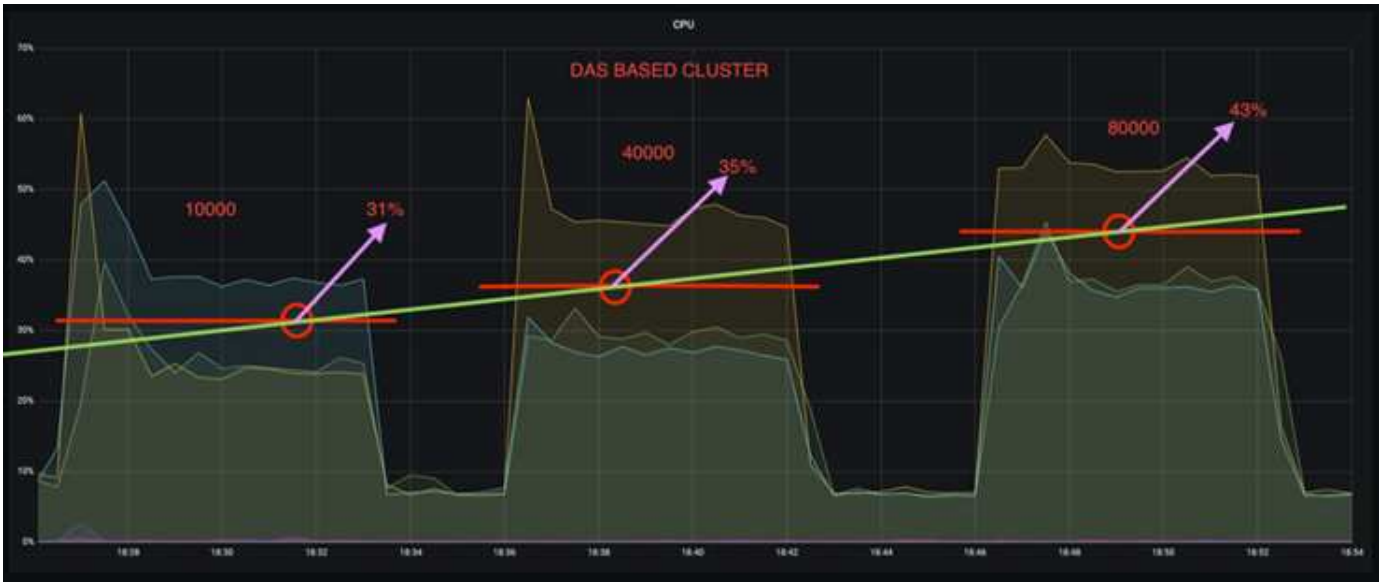
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

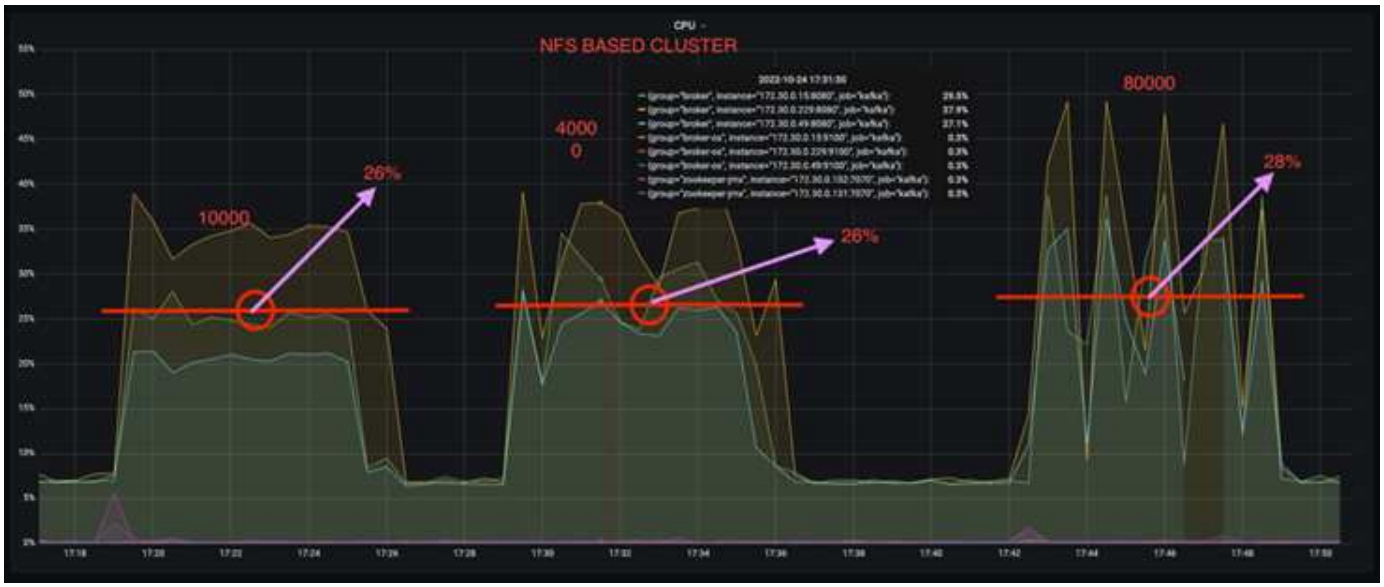
3. La configurazione della velocità di produzione è stata aumentata in quattro iterazioni e l'utilizzo della CPU è stato registrato con Grafana. Il tasso di produzione è stato fissato ai seguenti livelli:
    - 10.000
    - 40.000
    - 80.000
    - 100.000

Osservazione

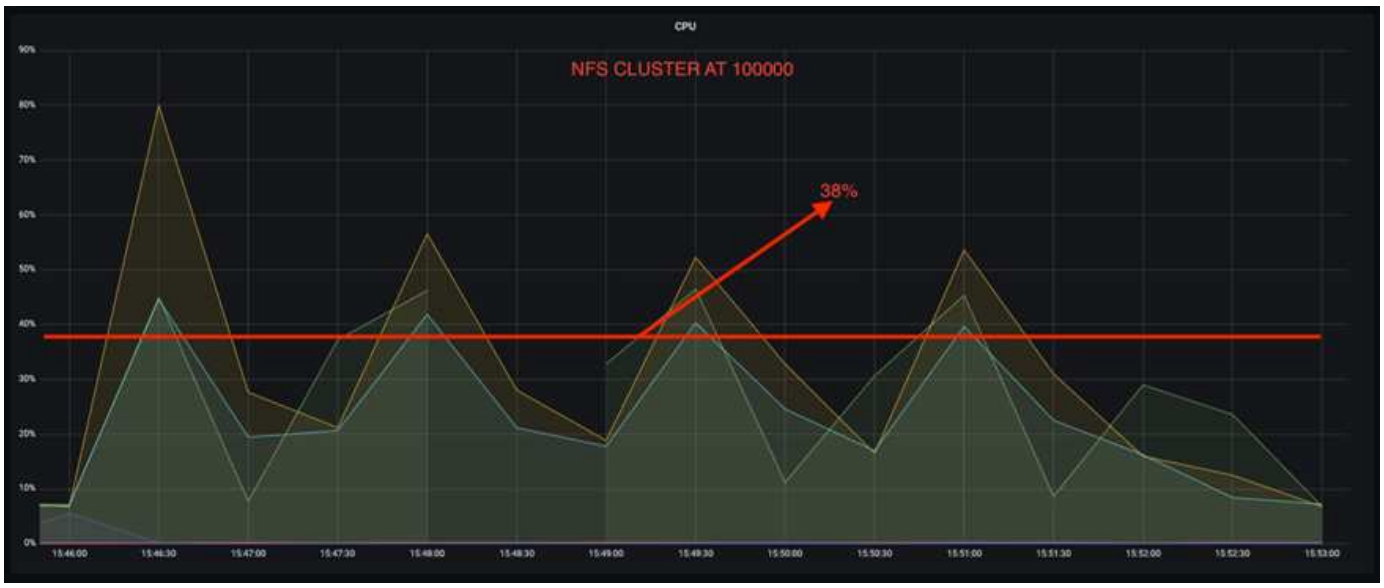
L'utilizzo dello storage NFS NetApp con Kafka offre due vantaggi principali:

- **È possibile ridurre l'utilizzo della CPU di quasi un terzo.** L'utilizzo complessivo della CPU con carichi di lavoro simili è risultato inferiore per NFS rispetto agli SSD DAS; i risparmi vanno dal 5% per tassi di produzione inferiori al 32% per tassi di produzione superiori.
- **Una riduzione tripla dell'utilizzo della CPU a velocità di produzione più elevate.** Come previsto, si è registrato un aumento dell'utilizzo della CPU con l'aumento dei tassi di produzione. Tuttavia, l'utilizzo della CPU sui broker Kafka che utilizzano DAS è aumentato dal 31% per il tasso di produzione più basso al 70% per il tasso di produzione più alto, con un incremento del 39%. Tuttavia, con un backend di archiviazione NFS, l'utilizzo della CPU è aumentato dal 26% al 38%, con un incremento del 12%.





Inoltre, a 100.000 messaggi, DAS mostra un utilizzo della CPU maggiore rispetto a un cluster NFS.



Recupero più rapido del broker

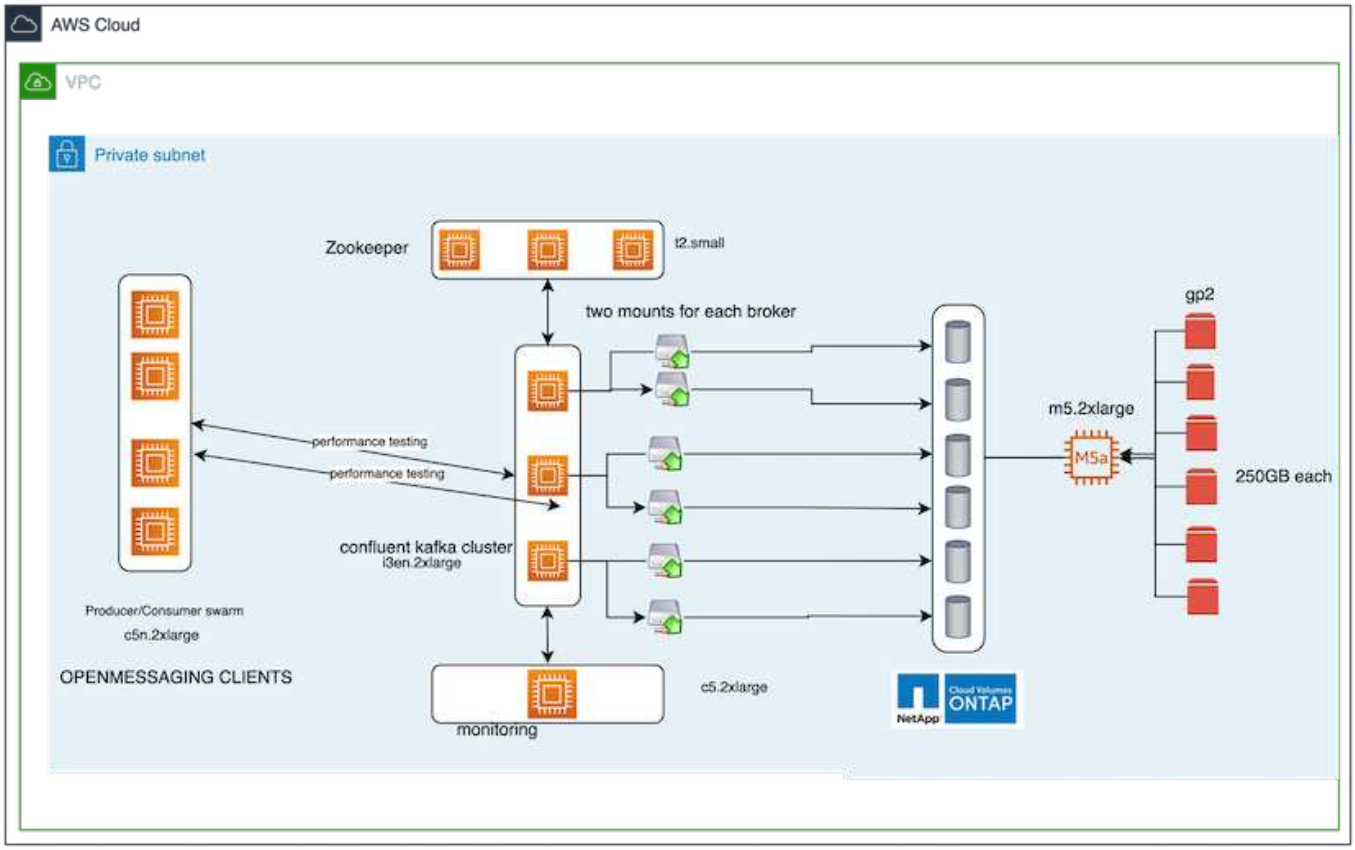
Abbiamo scoperto che i broker Kafka ripristinano più velocemente quando utilizzano lo storage NetApp NFS condiviso. Quando un broker si blocca in un cluster Kafka, può essere sostituito da un broker funzionante con lo stesso ID broker. Dopo aver eseguito questo caso di test, abbiamo scoperto che, nel caso di un cluster Kafka basato su DAS, il cluster ricostruisce i dati su un broker funzionante appena aggiunto, il che richiede molto tempo. Nel caso di un cluster Kafka basato su NetApp NFS, il broker sostitutivo continua a leggere i dati dalla directory di registro precedente e ripristina molto più rapidamente.

Configurazione architettonica

La tabella seguente mostra la configurazione ambientale per un cluster Kafka che utilizza NAS.

Componente della piattaforma	Configurazione dell'ambiente
Kafka 3.2.3	• 3 guardiani dello zoo – t2.small • 3 server broker – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x produttore/consumatore — c5n.2xlarge • 1 x nodo Kafka di backup – i3en.2xlarge
Sistema operativo su tutti i nodi	RHEL8.7 o successivo
Istanza ONTAP di NetApp Cloud Volumes ONTAP	Istanza a nodo singolo – M5.2xLarge

La figura seguente illustra l'architettura di un cluster Kafka basato su NAS.



- **Calcolare.** Un cluster Kafka a tre nodi con un ensemble zookeeper a tre nodi in esecuzione su server dedicati. Ogni broker ha due punti di montaggio NFS su un singolo volume sull'istanza NetApp CVO tramite un LIF dedicato.
- **Monitoraggio.** Due nodi per una combinazione Prometheus-Grafana. Per generare carichi di lavoro, utilizziamo un cluster separato a tre nodi in grado di produrre e consumare dati per questo cluster Kafka.
- **Magazzinaggio.** Un'istanza ONTAP NetApp Cloud Volumes a nodo singolo con sei volumi GP2 AWS-EBS da 250 GB montati sull'istanza. Questi volumi vengono quindi esposti al cluster Kafka come sei volumi NFS tramite LIF dedicati.
- **Configurazione del broker.** L'unico elemento configurabile in questo caso di test sono i broker Kafka. Per i broker Kafka sono state selezionate le seguenti specifiche. IL `replica.lag.time.mx.ms` è impostato su un valore elevato perché determina la velocità con cui un nodo specifico viene rimosso dall'elenco ISR. Quando si passa da nodi danneggiati a nodi sani, non si desidera che l'ID del broker venga escluso dall'elenco 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
```

Metodologia di test

1. Sono stati creati due cluster simili:
   - Un cluster confluente basato su EC2.
   - Un cluster confluente basato su NetApp NFS.
2. È stato creato un nodo Kafka di standby con una configurazione identica ai nodi del cluster Kafka originale.
3. Su ciascuno dei cluster è stato creato un argomento di esempio e sono stati popolati circa 110 GB di dati su ciascun broker.
   - **Cluster basato su EC2.** Una directory di dati del broker Kafka è mappata su `/mnt/data-2` (Nella figura seguente, Broker-1 del cluster1 [terminale sinistro]).
   - * Cluster basato su NetApp NFS.* Una directory di dati del broker Kafka è montata sul punto NFS `/mnt/data` (Nella figura seguente, Broker-1 del cluster2 [terminale destro]).

4. In ciascuno dei cluster, Broker-1 è stato terminato per attivare un processo di ripristino del broker non riuscito.
5. Dopo la chiusura del broker, l'indirizzo IP del broker è stato assegnato come IP secondario al broker standby. Ciò era necessario perché un broker in un cluster Kafka viene identificato da quanto segue:
   - **Indirizzo IP.** Assegnato riassegnando l'IP del broker non riuscito al broker standby.
   - **ID broker.** Questo è stato configurato nel broker standby `server.properties` .
6. Dopo l'assegnazione dell'IP, il servizio Kafka è stato avviato sul broker standby.
7. Dopo un po', sono stati estratti i log del server per verificare il tempo impiegato per creare i dati sul nodo sostitutivo nel cluster.

Osservazione

Il recupero del broker Kafka è stato quasi nove volte più rapido. È stato riscontrato che il tempo impiegato per ripristinare un nodo broker non riuscito è significativamente più rapido quando si utilizza l'archiviazione condivisa NetApp NFS rispetto all'utilizzo di SSD DAS in un cluster Kafka. Per 1 TB di dati di argomento, il tempo di ripristino per un cluster basato su DAS è stato di 48 minuti, rispetto a meno di 5 minuti per un cluster Kafka basato su NetApp-NFS.

Abbiamo osservato che il cluster basato su EC2 ha impiegato 10 minuti per ricostruire i 110 GB di dati sul nuovo nodo broker, mentre il cluster basato su NFS ha completato il ripristino in 3 minuti. Abbiamo anche osservato nei log che gli offset dei consumatori per le partizioni per EC2 erano 0, mentre, sul cluster NFS, gli offset dei consumatori venivano prelevati dal broker precedente.

```
[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 basato su DAS

1. Il nodo di backup è stato avviato alle 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. Il processo di ricostruzione dei dati è terminato alle 09:05:24,860. L'elaborazione di 110 GB di dati ha richiesto circa 10 minuti.

```
[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 basato su NFS

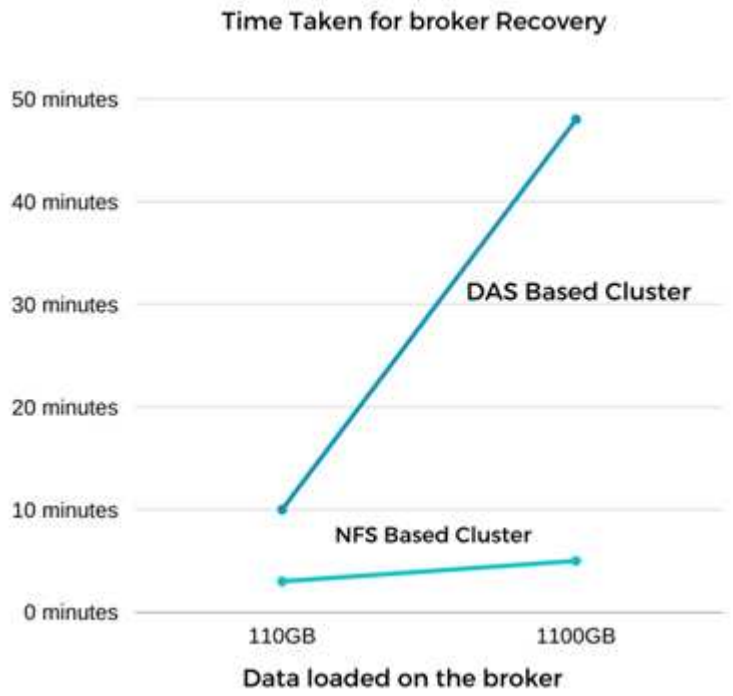
1. Il nodo di backup è stato avviato alle 09:39:17,213. Di seguito è evidenziata la voce di registro iniziale.

```
2 [2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiati
3 [2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.
4 [2022-10-31 09:39:17,213] INFO starting (kafka.server.KafkaServer)
5 [2022-10-31 09:39:17,214] INFO Connecting to zookeeper on 172.30.0.22:2181,172.30.
6 [2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new s
7 [2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a
8 [2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in
```

2. Il processo di ricostruzione dei dati è terminato alle 09:42:29,115. L'elaborazione di 110 GB di dati ha richiesto circa 3 minuti.

```
[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)
```

Il test è stato ripetuto per broker contenenti circa 1 TB di dati, impiegando circa 48 minuti per il DAS e 3 minuti per l'NFS. I risultati sono rappresentati nel grafico seguente.



Efficienza di archiviazione

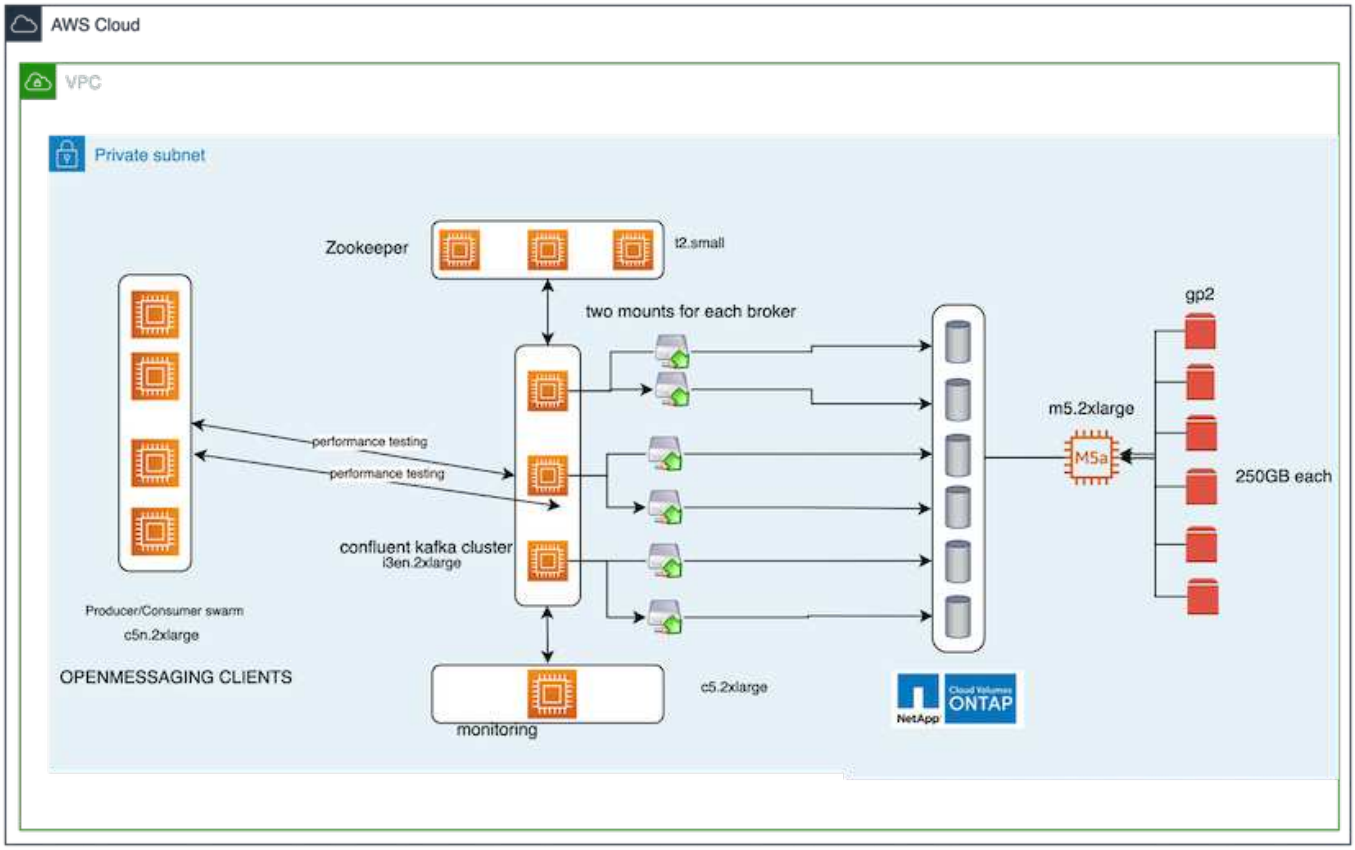
Poiché il livello di archiviazione del cluster Kafka è stato fornito tramite NetApp ONTAP, abbiamo ottenuto tutte le funzionalità di efficienza di archiviazione di ONTAP. Questa soluzione è stata testata generando una quantità significativa di dati su un cluster Kafka con storage NFS fornito su Cloud Volumes ONTAP. Abbiamo potuto constatare che si è verificata una significativa riduzione dello spazio grazie alle funzionalità ONTAP .

Configurazione architettonica

La tabella seguente mostra la configurazione ambientale per un cluster Kafka che utilizza NAS.

Componente della piattaforma	Configurazione dell'ambiente
Kafka 3.2.3	• 3 guardiani dello zoo – t2.small • 3 server broker – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x produttore/consumatore — c5n.2xlarge *
Sistema operativo su tutti i nodi	RHEL8.7 o successivo
Istanza ONTAP di NetApp Cloud Volumes ONTAP	Istanza a nodo singolo – M5.2xLarge

La figura seguente illustra l'architettura di un cluster Kafka basato su NAS.



- **Calcolare.** Abbiamo utilizzato un cluster Kafka a tre nodi con un ensemble zookeeper a tre nodi in esecuzione su server dedicati. Ogni broker aveva due punti di montaggio NFS su un singolo volume sull'istanza NetApp CVO tramite un LIF dedicato.

- **Monitoraggio.** Abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per generare i carichi di lavoro, abbiamo utilizzato un cluster separato a tre nodi in grado di produrre e consumare dati per questo cluster Kafka.
- **Magazzinaggio.** Abbiamo utilizzato un'istanza NetApp Cloud Volumes ONTAP a nodo singolo con sei volumi GP2 AWS-EBS da 250 GB montati sull'istanza. Questi volumi sono stati quindi esposti al cluster Kafka come sei volumi NFS tramite LIF dedicati.
- **Configurazione.** Gli elementi configurabili in questo caso di prova erano i broker Kafka.

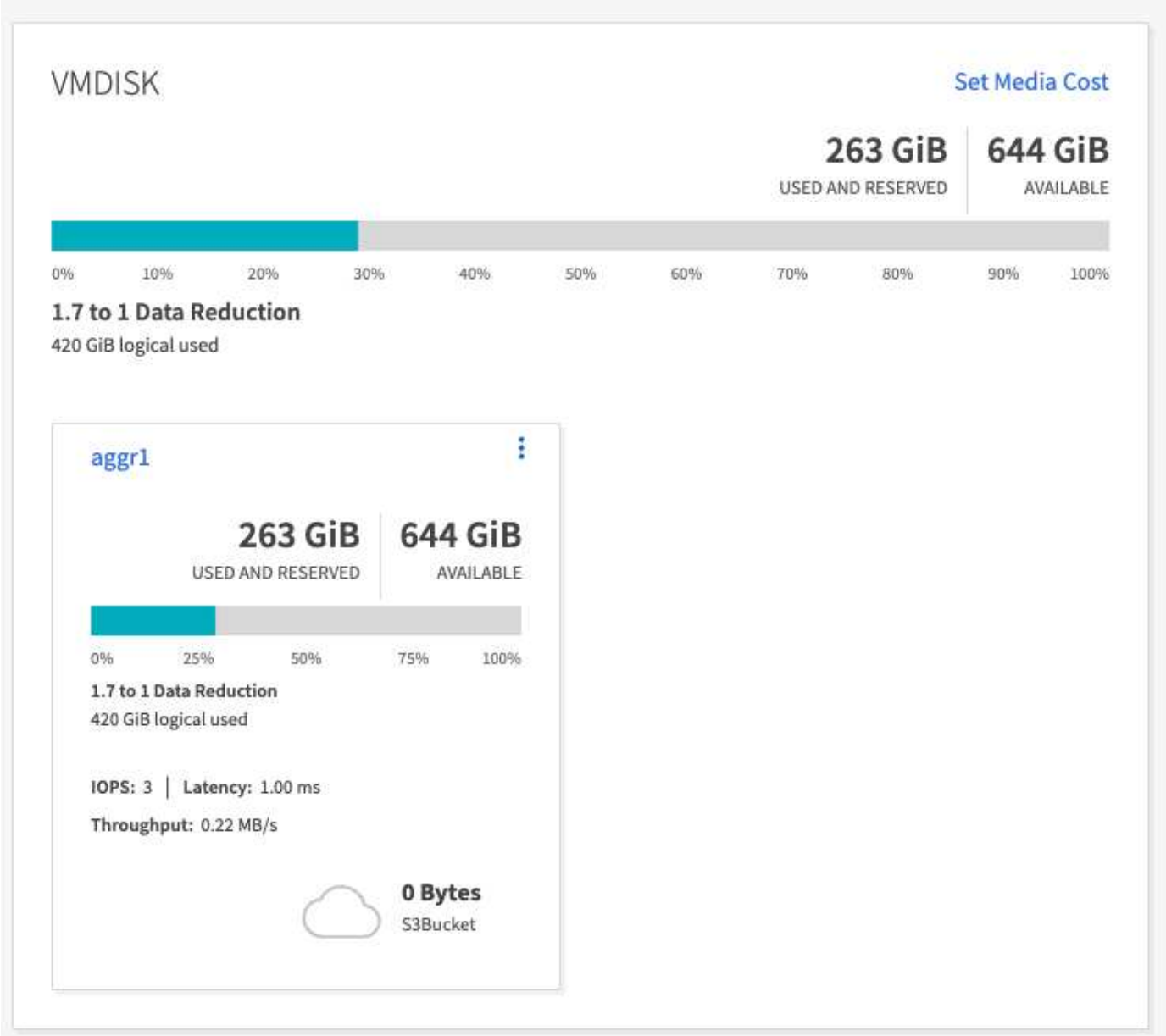
La compressione è stata disattivata dal produttore, consentendogli così di generare un rendimento elevato. L'efficienza dell'archiviazione era invece gestita dal livello di elaborazione.

Metodologia di test

1. È stato predisposto un cluster Kafka con le specifiche sopra menzionate.
2. Sul cluster sono stati prodotti circa 350 GB di dati utilizzando lo strumento OpenMessaging Benchmarking.
3. Una volta completato il carico di lavoro, le statistiche sull'efficienza dell'archiviazione sono state raccolte utilizzando ONTAP System Manager e la CLI.

Osservazione

Per i dati generati utilizzando lo strumento OMB, abbiamo riscontrato un risparmio di spazio di circa il 33% con un rapporto di efficienza di archiviazione di 1,70:1. Come si può vedere nelle figure seguenti, lo spazio logico utilizzato dai dati prodotti era di 420,3 GB e lo spazio fisico utilizzato per contenere i dati era di 281,7 GB.




```
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
```

Panoramica e convalida delle prestazioni in AWS

Un cluster Kafka con il livello di archiviazione montato su NetApp NFS è stato sottoposto a benchmark per le prestazioni nel cloud AWS. Gli esempi di benchmarking sono descritti nelle sezioni seguenti.

Kafka nel cloud AWS con NetApp Cloud Volumes ONTAP (coppia ad alta disponibilità e nodo singolo)

Un cluster Kafka con NetApp Cloud Volumes ONTAP (coppia HA) è stato sottoposto a benchmark per le prestazioni nel cloud AWS. Questo benchmarking è descritto nelle sezioni seguenti.

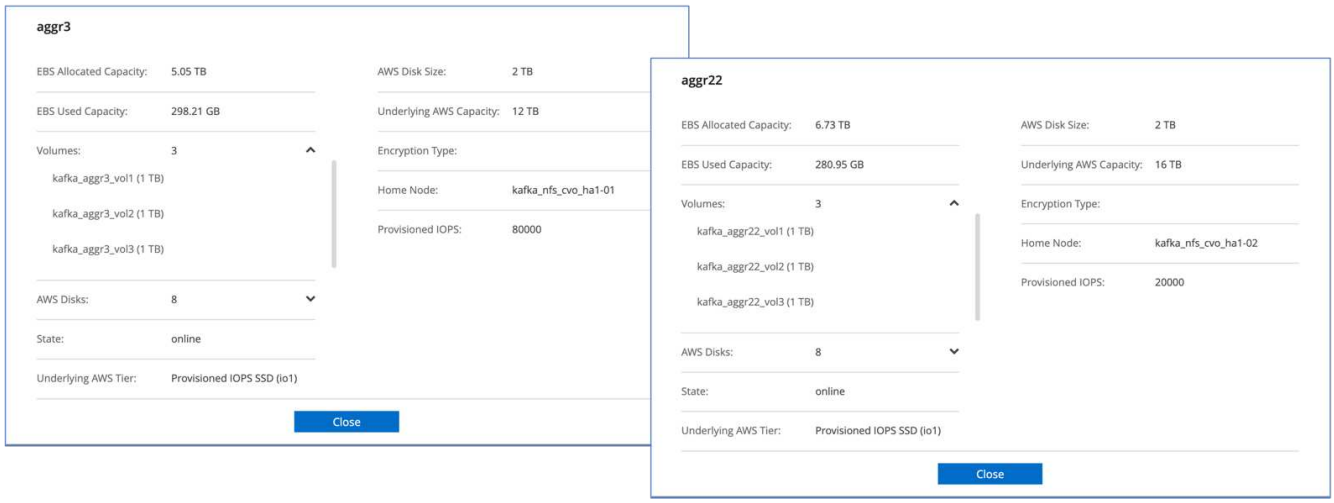
Configurazione architettonica

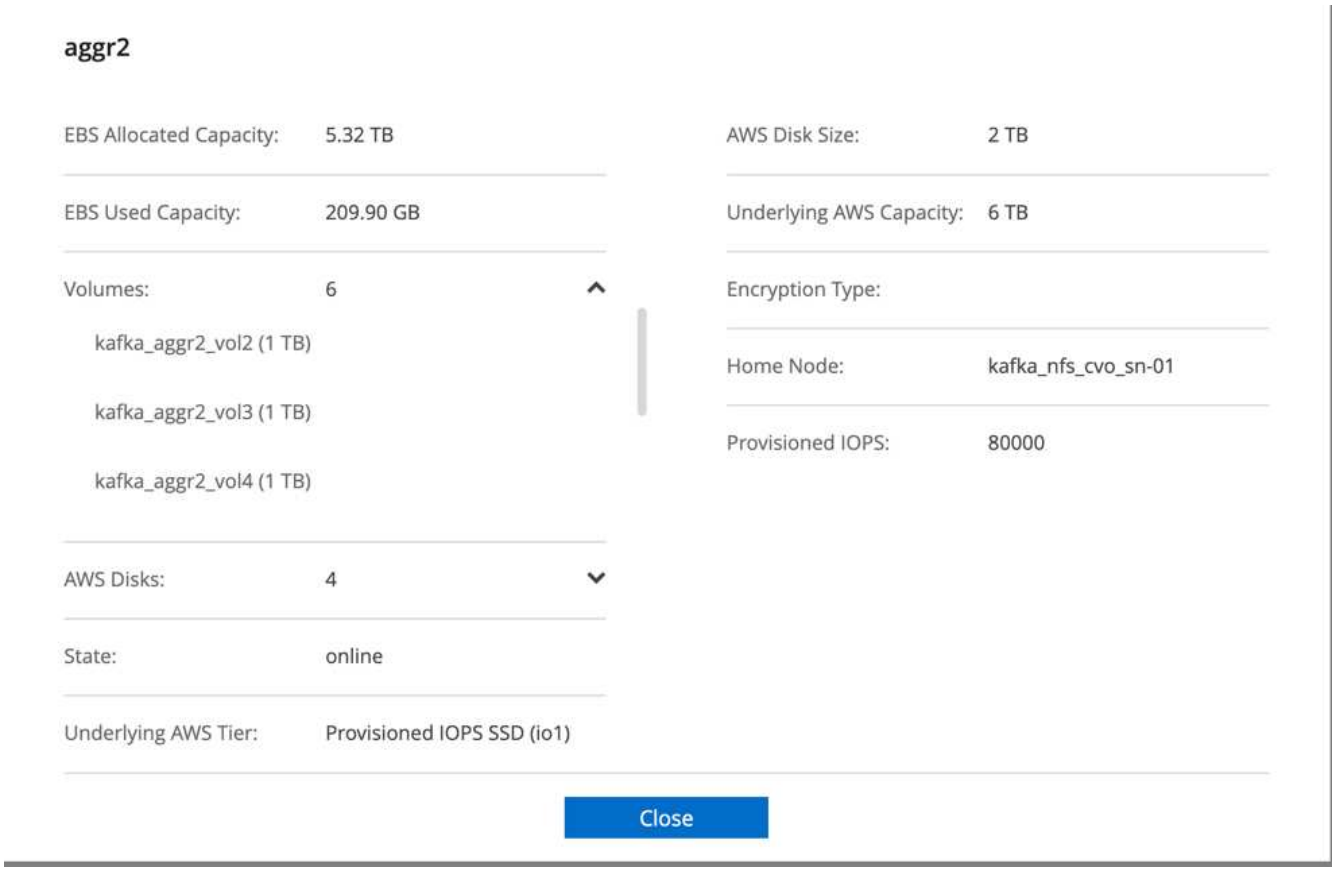
La tabella seguente mostra la configurazione ambientale per un cluster Kafka che utilizza NAS.

Componente della piattaforma	Configurazione dell'ambiente
Kafka 3.2.3	• 3 guardiani dello zoo – t2.small • 3 server broker – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x produttore/consumatore — c5n.2xlarge *
Sistema operativo su tutti i nodi	RHEL8.6
Istanza ONTAP di NetApp Cloud Volumes ONTAP	Istanza di coppia HA – m5dn.12xLarge x 2 nodi Istanza di nodo singolo – m5dn.12xLarge x 1 nodo

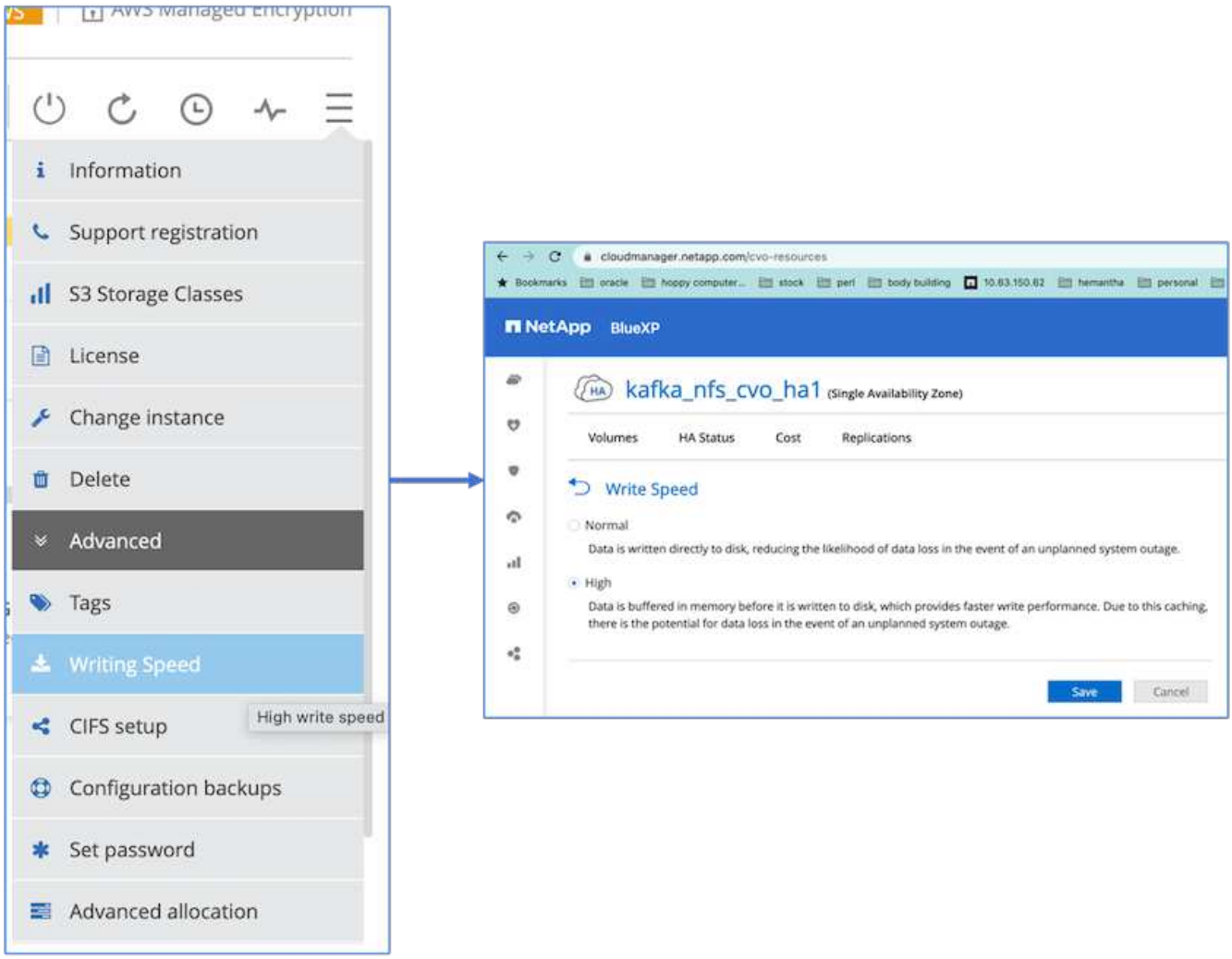
Configurazione ONTAP del volume del cluster NetApp

1. Per la coppia Cloud Volumes ONTAP HA, abbiamo creato due aggregati con tre volumi su ciascun aggregato su ciascun controller di storage. Per il singolo nodo Cloud Volumes ONTAP , creiamo sei volumi in un aggregato.



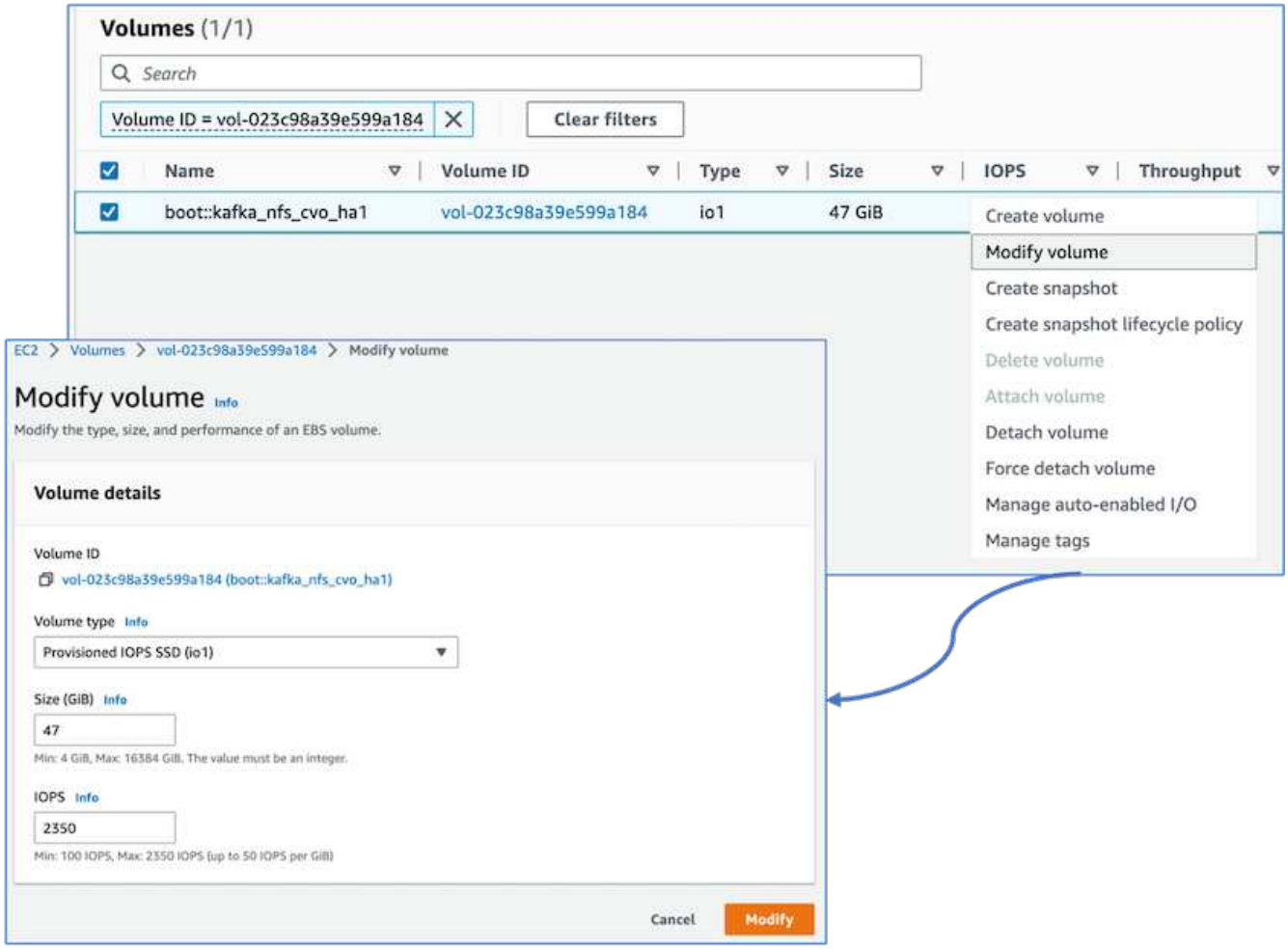


2. Per ottenere migliori prestazioni di rete, abbiamo abilitato la rete ad alta velocità sia per la coppia HA che per il singolo nodo.



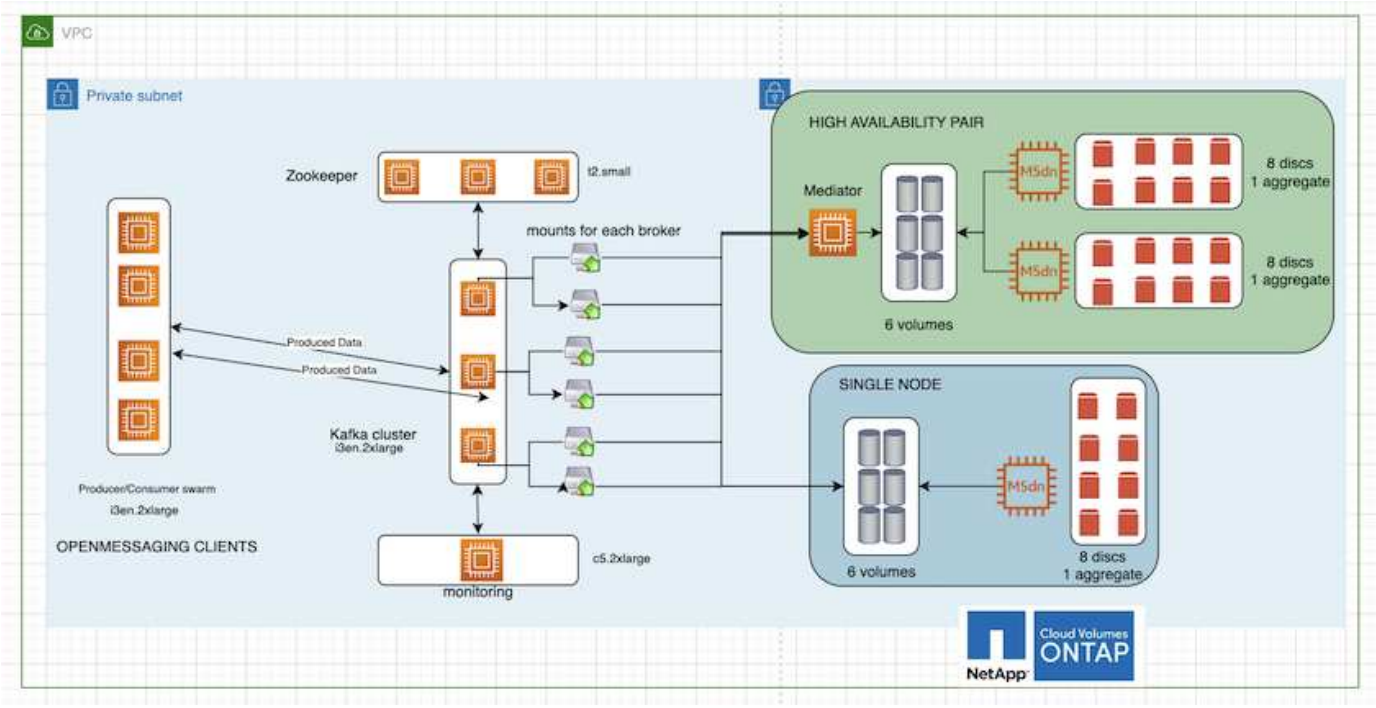
3. Abbiamo notato che la NVRAM ONTAP aveva più IOPS, quindi abbiamo modificato gli IOPS a 2350 per il volume root Cloud Volumes ONTAP . Il disco del volume radice in Cloud Volumes ONTAP aveva una dimensione di 47 GB. Il seguente comando ONTAP è per la coppia HA e lo stesso passaggio è applicabile al singolo nodo.

```
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 figura seguente illustra l'architettura di un cluster Kafka basato su NAS.

- **Calcolare.** Abbiamo utilizzato un cluster Kafka a tre nodi con un ensemble zookeeper a tre nodi in esecuzione su server dedicati. Ogni broker aveva due punti di montaggio NFS su un singolo volume sull'istanza Cloud Volumes ONTAP tramite un LIF dedicato.
- **Monitoraggio.** Abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per generare i carichi di lavoro, abbiamo utilizzato un cluster separato a tre nodi in grado di produrre e consumare dati per questo cluster Kafka.
- **Magazzinaggio.** Abbiamo utilizzato un'istanza ONTAP di volumi Cloud HA-pair con un volume GP3 AWS-EBS da 6 TB montato sull'istanza. Il volume è stato quindi esportato sul broker Kafka con un montaggio NFS.



Configurazioni di benchmarking di OpenMessage

1. Per migliorare le prestazioni NFS, abbiamo bisogno di più connessioni di rete tra il server NFS e il client NFS, che possono essere create utilizzando nconnect. Montare i volumi NFS sui nodi broker con l'opzione nconnect eseguendo il seguente comando:

```
[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. Controllare le connessioni di rete in Cloud Volumes ONTAP. Il seguente comando ONTAP viene utilizzato dal singolo nodo Cloud Volumes ONTAP . Lo stesso passaggio è applicabile alla coppia 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. Utilizziamo il seguente Kafka `server.properties` in tutti i broker Kafka per la coppia Cloud Volumes ONTAP HA. IL `log.dirs` La proprietà è diversa per ogni broker, mentre le restanti proprietà sono comuni a tutti i broker. Per broker1, il `log.dirs` il valore è il seguente:

```
[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 ~]#
```

◦ Per broker2, il `log.dirs` il valore della proprietà è il seguente:

```
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
```

◦ Per broker3, il `log.dirs` il valore della proprietà è il seguente:

```
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. Per il singolo nodo Cloud Volumes ONTAP , The Kafka `servers.properties` è lo stesso della coppia Cloud Volumes ONTAP HA, ad eccezione di `log.dirs` proprietà.
   - Per broker1, il `log.dirs` il valore è il seguente:

```
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
```

   - Per broker2, il `log.dirs` il valore è il seguente:

```
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
```

   - Per broker3, il `log.dirs` il valore della proprietà è il seguente:

```
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. Il carico di lavoro nell'OMB è configurato con le seguenti proprietà: `(/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
```

IL `messageSize` può variare a seconda del caso d'uso. Nel nostro test delle prestazioni abbiamo utilizzato

3K.

Abbiamo utilizzato due driver diversi, Sync o Throughput, di OMB per generare il carico di lavoro sul cluster Kafka.

- Il file yaml utilizzato per le proprietà del driver di sincronizzazione è il seguente (`/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
```

- Il file yaml utilizzato per le proprietà del driver Throughput è il seguente (`/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
```

Metodologia di test

1. Un cluster Kafka è stato predisposto secondo le specifiche descritte sopra utilizzando Terraform e Ansible. Terraform viene utilizzato per creare l'infrastruttura utilizzando istanze AWS per il cluster Kafka, mentre Ansible crea il cluster Kafka su di esse.
2. Un carico di lavoro OMB è stato attivato con la configurazione del carico di lavoro descritta sopra e il driver Sync.

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

3. Un altro carico di lavoro è stato attivato con il driver Throughput con la stessa configurazione del carico di lavoro.

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

Osservazione

Sono stati utilizzati due diversi tipi di driver per generare carichi di lavoro per confrontare le prestazioni di un'istanza Kafka in esecuzione su NFS. La differenza tra i driver è la proprietà di svuotamento del registro.

Per una coppia Cloud Volumes ONTAP HA:

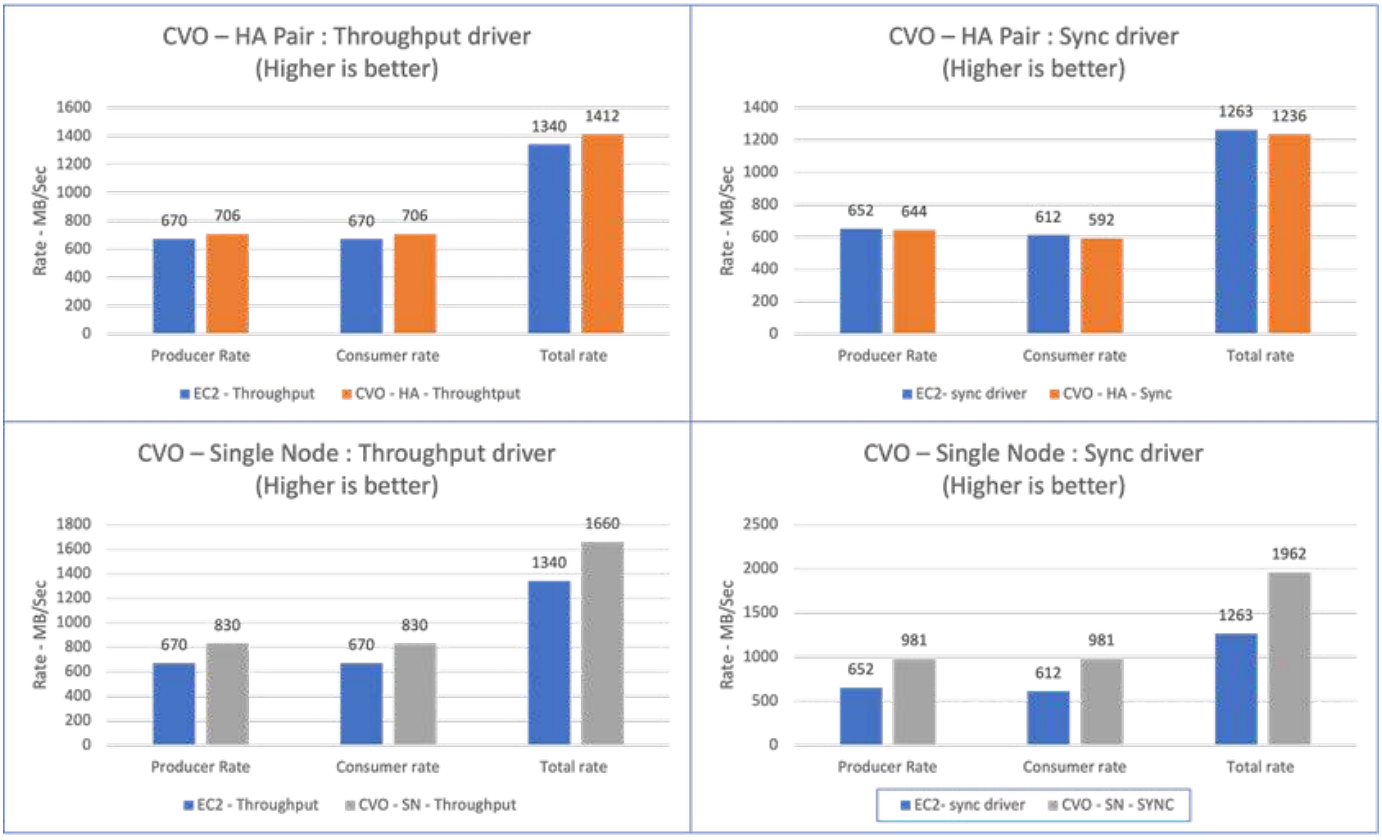
- Velocità totale generata in modo coerente dal driver Sync: ~1236 MBps.
- Throughput totale generato per il driver Throughput: picco ~1412 MBps.

Per un singolo Cloud Volumes ONTAP :

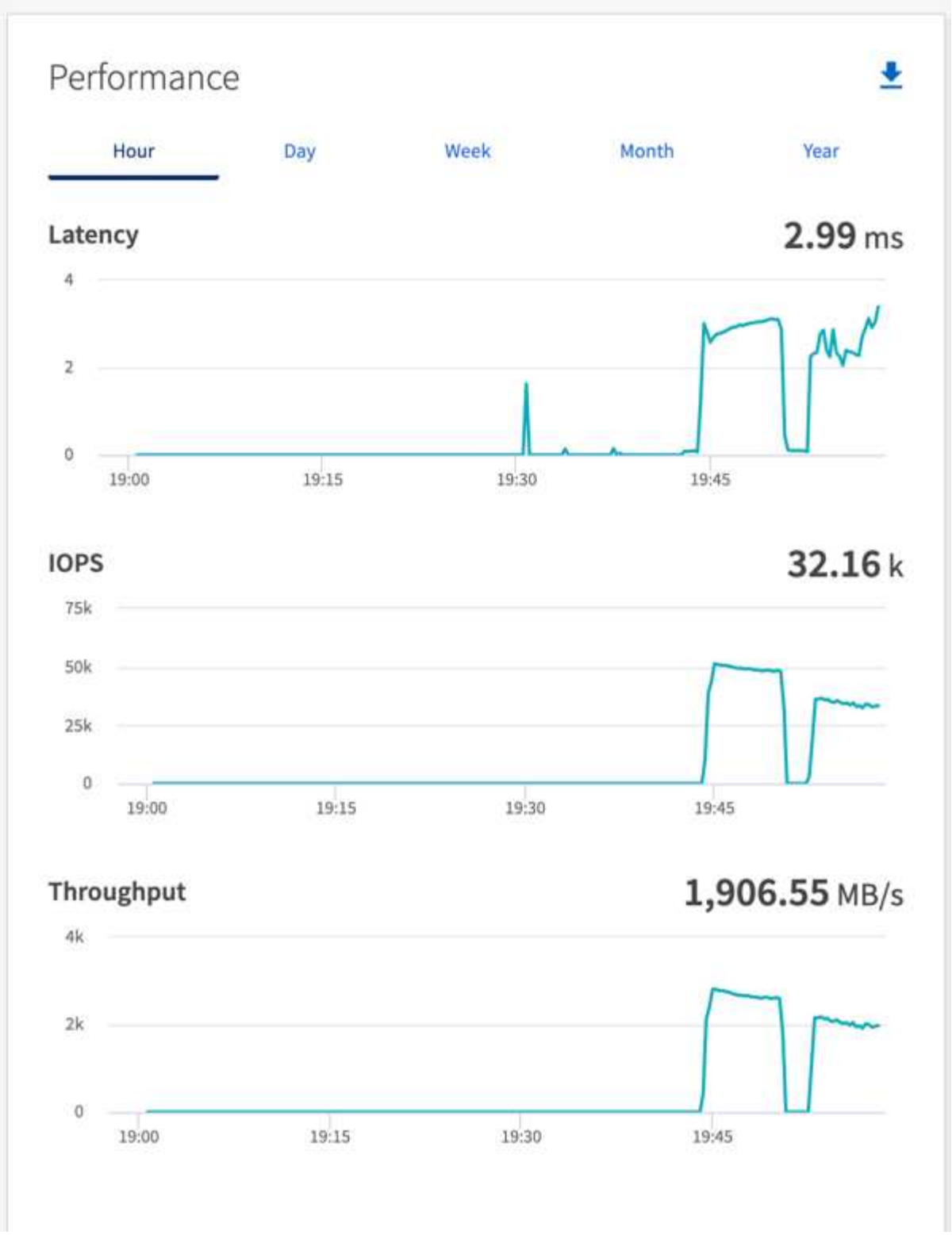
- Velocità totale generata in modo coerente dal driver Sync: ~ 1962 MBps.
- Throughput totale generato dal driver Throughput: picco ~1660MBps

Il driver Sync è in grado di generare un throughput costante poiché i log vengono scaricati sul disco all'istante, mentre il driver Throughput genera picchi di throughput poiché i log vengono salvati sul disco in blocco.

Questi numeri di throughput vengono generati per la configurazione AWS specificata. Per requisiti di prestazioni più elevati, i tipi di istanza possono essere ampliati e ulteriormente ottimizzati per ottenere numeri di throughput migliori. La produttività totale o tasso totale è la combinazione del tasso del produttore e del tasso del consumatore.



Assicurarsi di controllare la velocità di archiviazione quando si esegue il benchmarking della velocità di elaborazione o del driver di sincronizzazione.



Panoramica e convalida delle prestazioni in AWS FSx ONTAP

Un cluster Kafka con il livello di archiviazione montato su NetApp NFS è stato sottoposto a benchmark per le prestazioni in AWS FSx ONTAP. Gli esempi di benchmarking sono descritti nelle sezioni seguenti.

Apache Kafka in AWS FSx ONTAP

Network File System (NFS) è un file system di rete ampiamente utilizzato per archiviare grandi quantità di dati. Nella maggior parte delle organizzazioni i dati vengono sempre più generati da applicazioni di streaming come Apache Kafka. Questi carichi di lavoro richiedono scalabilità, bassa latenza e un'architettura di acquisizione dati solida con moderne capacità di archiviazione. Per consentire analisi in tempo reale e fornire informazioni fruibili, è necessaria un'infrastruttura ben progettata e altamente performante.

Kafka è progettato per funzionare con un file system compatibile con POSIX e si affida al file system per gestire le operazioni sui file, ma quando si archiviano dati su un file system NFSv3, il client NFS del broker Kafka può interpretare le operazioni sui file in modo diverso rispetto a un file system locale come XFS o Ext4. Un esempio comune è la rinomina NFS Silly che causava il fallimento dei broker Kafka durante l'espansione dei cluster e la riallocazione delle partizioni. Per affrontare questa sfida, NetApp ha aggiornato il client NFS Linux open source con modifiche ora generalmente disponibili in RHEL8.7, RHEL9.1 e supportate dall'attuale versione di FSx ONTAP , ONTAP 9.12.1.

Amazon FSx ONTAP fornisce un file system NFS completamente gestito, scalabile e ad alte prestazioni nel cloud. I dati Kafka su FSx ONTAP possono essere scalati per gestire grandi quantità di dati e garantire la tolleranza agli errori. NFS fornisce una gestione centralizzata dell'archiviazione e della protezione dei dati per set di dati critici e sensibili.

Questi miglioramenti consentono ai clienti AWS di sfruttare FSx ONTAP durante l'esecuzione di carichi di lavoro Kafka sui servizi di elaborazione AWS. Questi vantaggi sono: * Riduzione dell'utilizzo della CPU per ridurre il tempo di attesa I/O * Tempo di ripristino più rapido del broker Kafka. * Affidabilità ed efficienza. * Scalabilità e prestazioni. * Disponibilità di zone multi-disponibilità. * Protezione dei dati.

Panoramica e convalida delle prestazioni in AWS FSx ONTAP

Un cluster Kafka con il livello di archiviazione montato su NetApp NFS è stato sottoposto a benchmark per le prestazioni nel cloud AWS. Gli esempi di benchmarking sono descritti nelle sezioni seguenti.

Kafka in AWS FSx ONTAP

Un cluster Kafka con AWS FSx ONTAP è stato sottoposto a benchmark per le prestazioni nel cloud AWS. Questo benchmarking è descritto nelle sezioni seguenti.

Configurazione architettonica

La tabella seguente mostra la configurazione ambientale per un cluster Kafka che utilizza AWS FSx ONTAP.

Componente della piattaforma	Configurazione dell'ambiente
Kafka 3.2.3	• 3 guardiani dello zoo – t2.small • 3 server broker – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x produttore/consumatore — c5n.2xlarge *
Sistema operativo su tutti i nodi	RHEL8.6
AWS FSx ONTAP	Multi-AZ con throughput di 4 GB/sec e 160.000 IOPS

Configurazione NetApp FSx ONTAP

1. Per i nostri test iniziali, abbiamo creato un file system FSx ONTAP con 2 TB di capacità e 40.000 IOPS per una velocità di elaborazione di 2 GB/sec.

```
[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"}
```

   Nel nostro esempio, stiamo distribuendo FSx ONTAP tramite AWS CLI. Sarà necessario personalizzare ulteriormente il comando nel proprio ambiente, a seconda delle necessità. FSx ONTAP può inoltre essere distribuito e gestito tramite la console AWS per un'esperienza di distribuzione più semplice e ottimizzata, con meno input dalla riga di comando.

   Documentazione In FSx ONTAP, il massimo IOPS raggiungibile per un file system con throughput di 2 GB/sec nella nostra regione di test (US-East-1) è di 80.000 IOPS. Il numero massimo totale di IOPS per un file system FSx ONTAP è di 160.000 IOPS, il che richiede un throughput di 4 GB/sec per essere raggiunto, come verrà illustrato più avanti in questo documento.

   Per ulteriori informazioni sulle specifiche delle prestazioni di FSx ONTAP , non esitate a consultare la documentazione di AWS FSx ONTAP qui: https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html .

   La sintassi dettagliata della riga di comando per FSx "create-file-system" è disponibile qui: https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

   Ad esempio, è possibile specificare una chiave KMS specifica anziché la chiave master AWS FSx predefinita utilizzata quando non viene specificata alcuna chiave KMS.

2. Durante la creazione del file system FSx ONTAP , attendi che lo stato "LifeCycle" cambi in "AVAILABLE" nel tuo ritorno JSON dopo aver descritto il tuo file system come segue:

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

3. Convalidare le credenziali effettuando l'accesso a FSx ONTAP SSH con l'utente fsxadmin: Fsxadmin è l'account amministratore predefinito per i file system FSx ONTAP al momento della creazione. La password per fsxadmin è la password configurata durante la prima creazione del file system nella console AWS o tramite AWS CLI, come abbiamo fatto nel passaggio 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. Una volta convalidate le credenziali, creare la macchina virtuale di archiviazione sul file system 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
```

Una Storage Virtual Machine (SVM) è un file server isolato con credenziali amministrative e endpoint propri per l'amministrazione e l'accesso ai dati nei volumi FSx ONTAP e fornisce la multi-tenancy FSx ONTAP .

5. Dopo aver configurato la macchina virtuale di archiviazione primaria, accedi tramite SSH al file system FSx ONTAP appena creato e crea volumi nella macchina virtuale di archiviazione utilizzando il comando di esempio seguente; allo stesso modo, creiamo 6 volumi per questa convalida. In base alla nostra convalida, mantieni il costituente predefinito (8) o meno costituenti, il che fornirà prestazioni migliori a 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. Per i nostri test avremo bisogno di una capacità aggiuntiva nei nostri volumi. Estendi le dimensioni del volume a 2 TB e montalo sul percorso di giunzione.

```
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
```

In FSx ONTAP, i volumi possono essere sottoposti a thin provisioning. Nel nostro esempio, la capacità totale del volume esteso supera la capacità totale del file system, quindi dovremo estendere la capacità totale del file system per sbloccare ulteriore capacità del volume fornito, come illustreremo nel passaggio successivo.

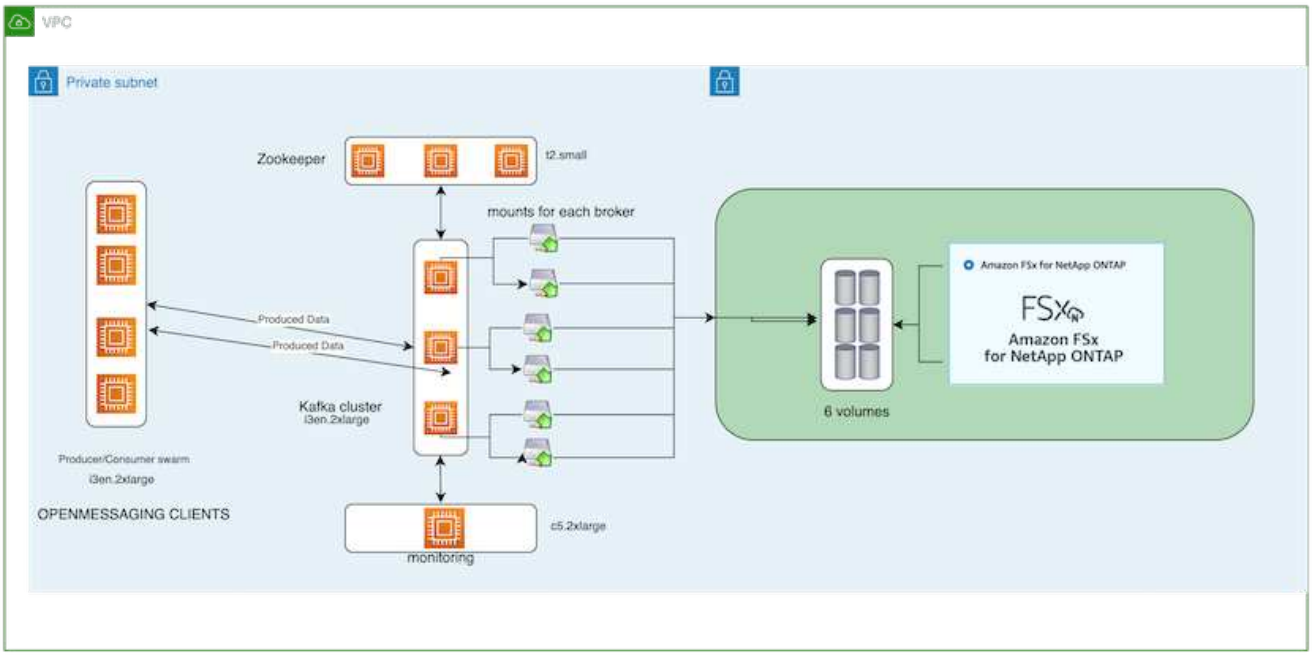
7. Successivamente, per prestazioni e capacità aggiuntive, estendiamo la capacità di throughput di FSx ONTAP da 2 GB/sec a 4 GB/sec e IOPS a 160000 e capacità a 5 TB

```
[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 sintassi dettagliata della riga di comando per FSx "update-file-system" è disponibile qui:https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. I volumi FSx ONTAP vengono montati con le opzioni nconnect e default nei broker Kafka

L'immagine seguente mostra l'architettura finale del nostro cluster Kafka basato su FSx ONTAP :



- Calcolare. Abbiamo utilizzato un cluster Kafka a tre nodi con un ensemble zookeeper a tre nodi in esecuzione su server dedicati. Ogni broker aveva sei punti di montaggio NFS su sei volumi nell'istanza FSx ONTAP .
- Monitoraggio. Abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per generare i carichi di lavoro, abbiamo utilizzato un cluster separato a tre nodi in grado di produrre e consumare dati per questo cluster Kafka.
- Magazzinaggio. Abbiamo utilizzato un FSx ONTAP con sei volumi da 2 TB montati. Il volume è stato quindi esportato nel broker Kafka con un montaggio NFS. I volumi FSx ONTAP vengono montati con 16 sessioni nconnect e opzioni predefinite nei broker Kafka.

Configurazioni di benchmarking di OpenMessage.

Abbiamo utilizzato la stessa configurazione utilizzata per NetApp Cloud Volumes ONTAP e i relativi dettagli sono disponibili qui: xref:./data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

Metodologia di test

1. Un cluster Kafka è stato predisposto secondo le specifiche descritte sopra utilizzando Terraform e Ansible. Terraform viene utilizzato per creare l'infrastruttura utilizzando istanze AWS per il cluster Kafka, mentre Ansible crea il cluster Kafka su di esse.
2. Un carico di lavoro OMB è stato attivato con la configurazione del carico di lavoro descritta sopra e il driver Sync.

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

3. Un altro carico di lavoro è stato attivato con il driver Throughput con la stessa configurazione del carico di lavoro.

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

Osservazione

Sono stati utilizzati due diversi tipi di driver per generare carichi di lavoro per confrontare le prestazioni di un'istanza Kafka in esecuzione su NFS. La differenza tra i driver è la proprietà di svuotamento del registro.

Per un fattore di replicazione Kafka 1 e FSx ONTAP:

- Velocità di trasmissione totale generata in modo coerente dal driver Sync: ~ 3218 MBps e prestazioni di picco in ~ 3652 MBps.
- Throughput totale generato in modo coerente dal driver Throughput: ~ 3679 MBps e prestazioni di picco in ~ 3908 MBps.

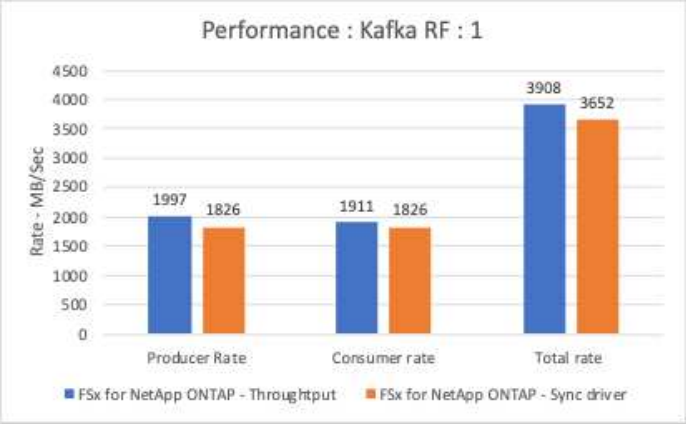
Per Kafka con fattore di replicazione 3 e FSx ONTAP :

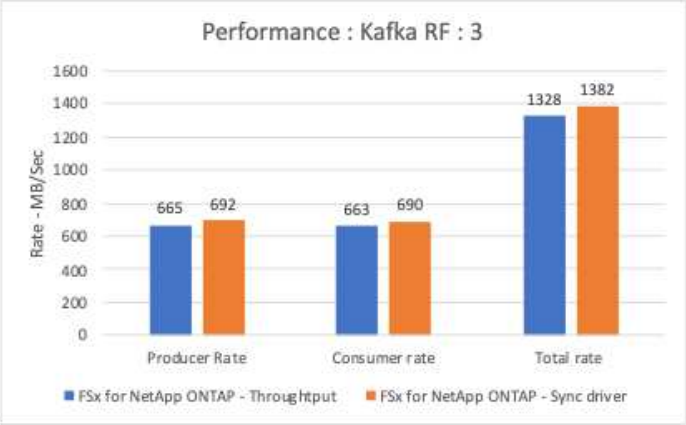
- Velocità di trasmissione totale generata in modo coerente dal driver Sync: ~ 1252 MBps e prestazioni di picco in ~ 1382 MBps.
- Throughput totale generato in modo coerente dal driver Throughput: ~ 1218 MBps e prestazioni di picco in ~ 1328 MBps.

Nel fattore di replicazione Kafka 3, l'operazione di lettura e scrittura è avvenuta tre volte su FSx ONTAP. Nel fattore di replicazione Kafka 1, l'operazione di lettura e scrittura è avvenuta una volta su FSx ONTAP, quindi in entrambe le convalide siamo riusciti a raggiungere la velocità massima di 4 GB/sec.

Il driver Sync è in grado di generare un throughput costante poiché i log vengono scaricati sul disco all'istante, mentre il driver Throughput genera picchi di throughput poiché i log vengono salvati sul disco in blocco.

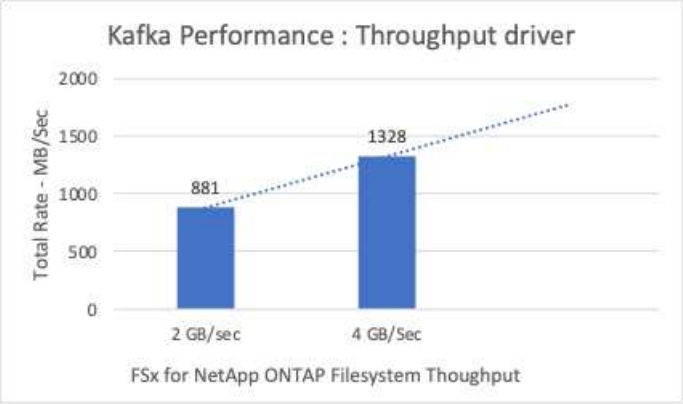
Questi numeri di throughput vengono generati per la configurazione AWS specificata. Per requisiti di prestazioni più elevati, i tipi di istanza possono essere ampliati e ulteriormente ottimizzati per ottenere numeri di throughput migliori. La produttività totale o tasso totale è la combinazione del tasso del produttore e del tasso del consumatore.

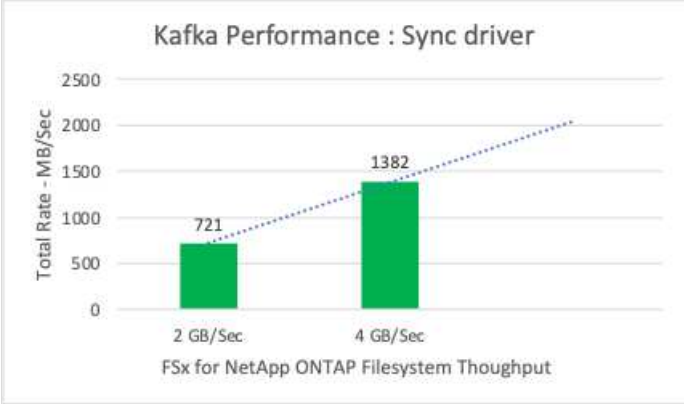




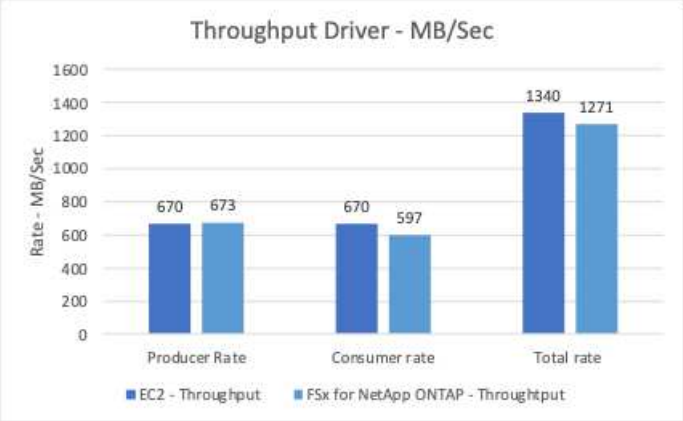
Il grafico sottostante mostra le prestazioni FSx ONTAP a 2 GB/sec e 4 GB/sec per il fattore di replicazione Kafka 3. Il fattore di replicazione 3 esegue l'operazione di lettura e scrittura tre volte sullo storage FSx ONTAP . La velocità totale per il driver di throughput è di 881 MB/sec, che esegue operazioni di lettura e scrittura Kafka a circa 2,64 GB/sec sul file system FSx ONTAP da 2 GB/sec, mentre la velocità totale per il driver di

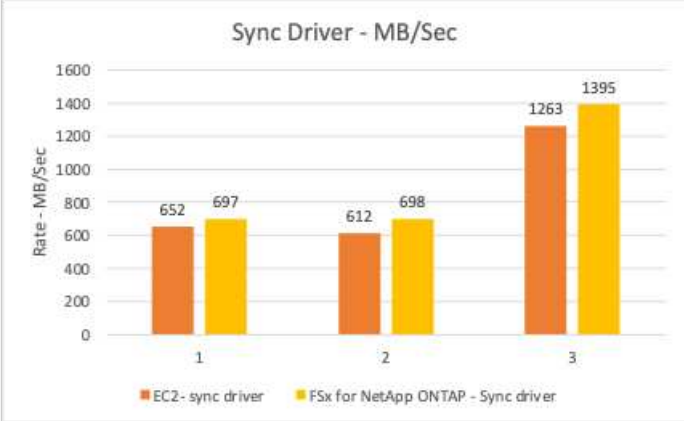
throughput è di 1328 MB/sec, che esegue operazioni di lettura e scrittura Kafka a circa 3,98 GB/sec. Le prestazioni di Kafka sono lineari e scalabili in base alla velocità di elaborazione di FSx ONTAP .





Il grafico sottostante mostra le prestazioni tra l'istanza EC2 e FSx ONTAP (fattore di replicazione Kafka: 3)

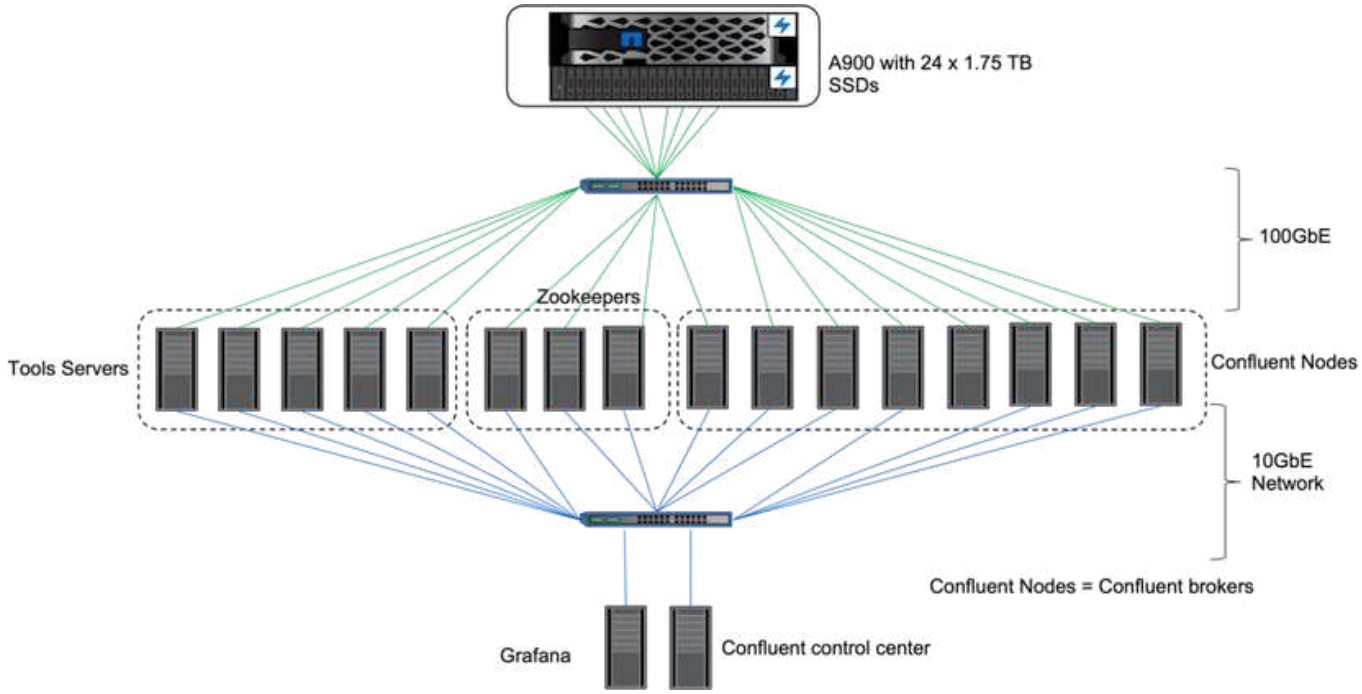




Panoramica e convalida delle prestazioni con AFF A900 in locale

In locale, abbiamo utilizzato il controller di storage NetApp AFF A900 con ONTAP 9.12.1RC1 per convalidare le prestazioni e la scalabilità di un cluster Kafka. Abbiamo utilizzato lo stesso banco di prova delle nostre precedenti best practice di archiviazione a livelli con ONTAP e AFF.

Abbiamo utilizzato Confluent Kafka 6.2.0 per valutare l' AFF A900. Il cluster è composto da otto nodi broker e tre nodi zookeeper. Per i test delle prestazioni abbiamo utilizzato cinque nodi worker OMB.



Configurazione di archiviazione

Abbiamo utilizzato istanze NetApp FlexGroups per fornire un singolo namespace per le directory di registro, semplificando il ripristino e la configurazione. Abbiamo utilizzato NFSv4.1 e pNFS per fornire un accesso diretto al percorso dei dati del segmento di registro.

Ottimizzazione del cliente

Ogni client ha montato l'istanza FlexGroup con il seguente comando.

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

Inoltre, abbiamo aumentato il `max_session_slots` dal valore predefinito `64` A `180` . Corrisponde al limite predefinito dello slot di sessione in ONTAP.

Ottimizzazione del broker Kafka

Per massimizzare la produttività nel sistema sottoposto a test, abbiamo aumentato significativamente i parametri predefiniti per alcuni pool di thread chiave. Consigliamo di seguire le best practice di Confluent Kafka per la maggior parte delle configurazioni. Questa ottimizzazione è stata utilizzata per massimizzare la concorrenza di I/O in sospeso sullo storage. Questi parametri possono essere modificati in base alle risorse di calcolo e agli attributi di archiviazione del tuo broker.

```
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
```

Metodologia di test del generatore di carico di lavoro

Abbiamo utilizzato le stesse configurazioni OMB utilizzate per i test cloud per la configurazione del driver Throughput e dell'argomento.

1. Un'istanza FlexGroup è stata fornita tramite Ansible su 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 è stato abilitato ONTAP SVM.

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

3. Il carico di lavoro è stato attivato con il driver Throughput utilizzando la stessa configurazione del carico di lavoro di Cloud Volumes ONTAP. Vedi la sezione "Prestazioni in stato stazionario " sotto. Il carico di lavoro utilizzava un fattore di replicazione pari a 3, il che significa che tre copie dei segmenti di registro venivano mantenute in NFS.

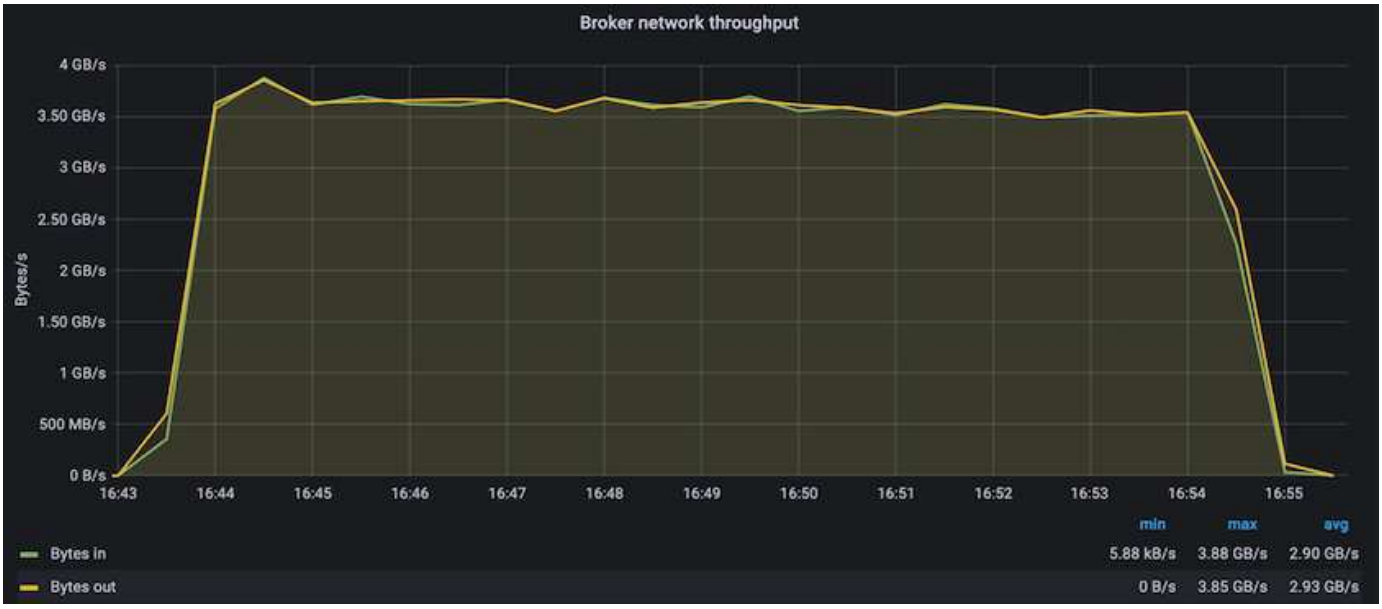
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. Infine, abbiamo completato le misurazioni utilizzando un backlog per misurare la capacità dei consumatori di tenersi aggiornati sui messaggi più recenti. L'OMB crea un backlog mettendo in pausa i consumatori all'inizio di una misurazione. Ciò produce tre fasi distinte: creazione del backlog (traffico riservato ai soli produttori), svuotamento del backlog (una fase in cui i consumatori recuperano gli eventi persi in un argomento) e stato stazionario. Vedi la sezione "Prestazioni estreme ed esplorazione dei limiti di archiviazione " per maggiori informazioni.

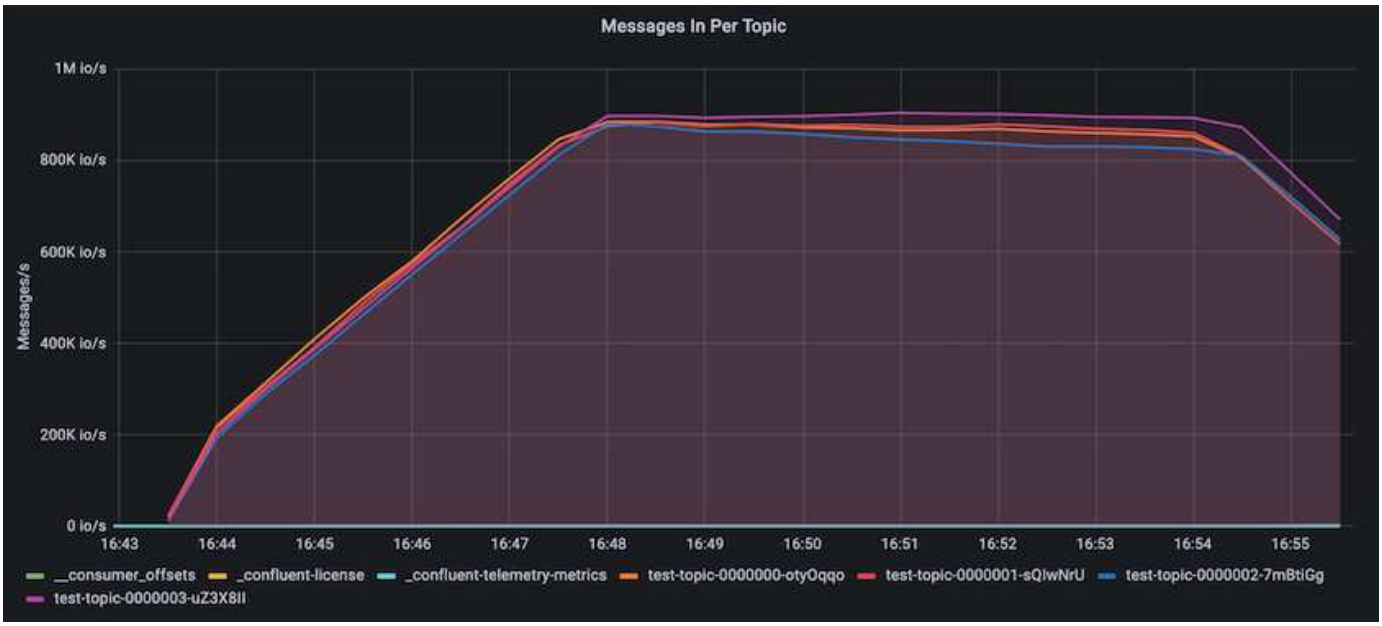
Prestazioni in stato stazionario

Abbiamo valutato l' AFF A900 utilizzando OpenMessaging Benchmark per fornire un confronto simile a quello tra Cloud Volumes ONTAP in AWS e DAS in AWS. Tutti i valori delle prestazioni rappresentano la produttività del cluster Kafka a livello di produttore e consumatore.

Le prestazioni in stato stazionario con Confluent Kafka e AFF A900 hanno raggiunto una velocità di trasmissione media di oltre 3,4 GBps sia per il produttore che per i consumatori. Si tratta di oltre 3,4 milioni di messaggi nel cluster Kafka. Visualizzando la produttività sostenuta in byte al secondo per BrokerTopicMetrics, vediamo le eccellenti prestazioni in stato stazionario e il traffico supportato dall'AFF AFF A900.



Ciò si allinea bene con la visione dei messaggi inviati per argomento. Il grafico seguente fornisce una ripartizione per argomento. Nella configurazione testata abbiamo visto quasi 900.000 messaggi per argomento, suddivisi in quattro argomenti.

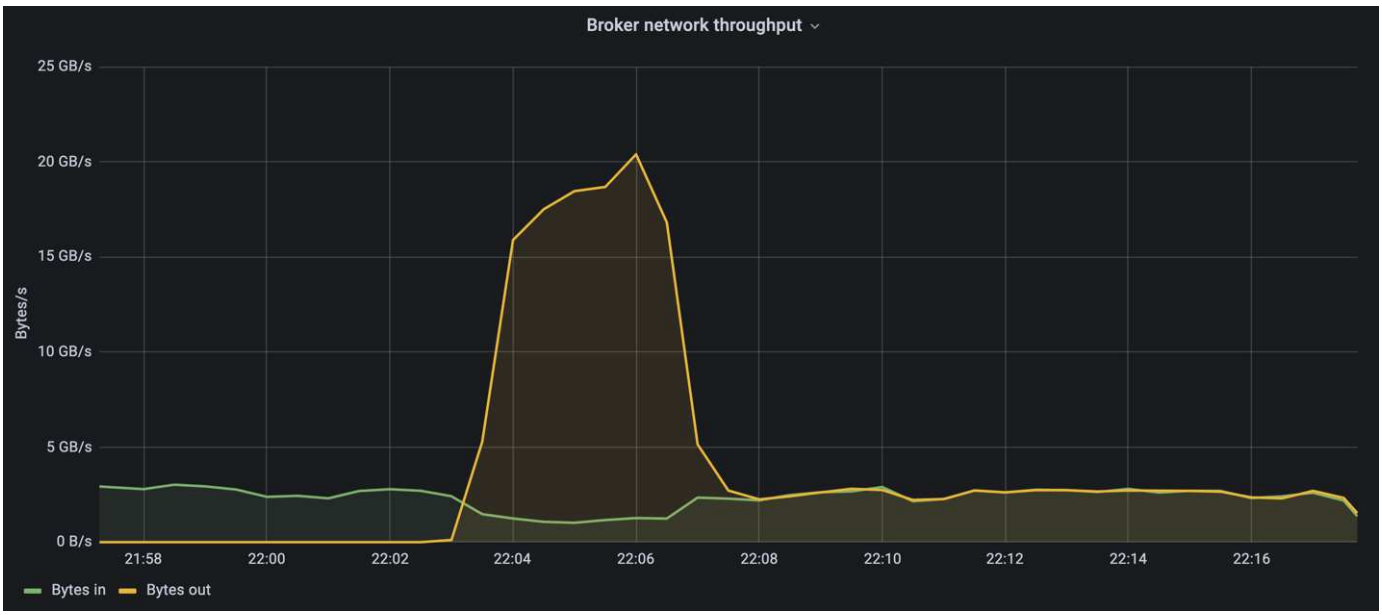


Prestazioni estreme ed esplorazione dei limiti di archiviazione

Per AFF, abbiamo anche effettuato test con OMB utilizzando la funzionalità backlog. La funzionalità di backlog sospende gli abbonamenti dei consumatori mentre viene creato un backlog di eventi nel cluster Kafka. Durante questa fase si verifica solo il traffico del produttore, che genera eventi che vengono salvati nei log. Questa emula più fedelmente i flussi di lavoro di elaborazione batch o di analisi offline; in questi flussi di lavoro, gli abbonamenti dei consumatori vengono avviati e devono leggere i dati storici che sono già stati rimossi dalla cache del broker.

Per comprendere i limiti di archiviazione sulla capacità di elaborazione del consumatore in questa configurazione, abbiamo misurato la fase solo del produttore per capire quanto traffico di scrittura poteva assorbire l'A900. Vedi la sezione successiva "Guida alle taglie " per capire come sfruttare questi dati.

Durante la parte di questa misurazione riservata al solo produttore, abbiamo riscontrato un throughput di picco elevato che ha spinto al limite le prestazioni dell'A900 (quando le altre risorse del broker non erano sature e servivano il traffico del produttore e del consumatore).



Abbiamo aumentato la dimensione del messaggio a 16k per questa misurazione, per limitare i sovraccarichi per messaggio e massimizzare la capacità di archiviazione sui punti di montaggio NFS.

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

Il cluster Confluent Kafka ha raggiunto un throughput di produzione massimo di 4,03 GBps.

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

Dopo che OMB ha completato il popolamento dell'eventbacklog, il traffico dei consumatori è stato riavviato. Durante le misurazioni con drenaggio del backlog, abbiamo osservato un throughput di picco dei consumatori di oltre 20 GBps su tutti gli argomenti. La velocità effettiva combinata del volume NFS che memorizzava i dati del registro OMB si avvicinava a circa 30 GBps.

Guida alle taglie

Amazon Web Services offre un "guida alle taglie" per il dimensionamento e la scalabilità dei cluster Kafka.

Questo dimensionamento fornisce una formula utile per determinare i requisiti di capacità di archiviazione per il cluster Kafka:

Per una produttività aggregata prodotta nel cluster di tcluster con un fattore di replicazione di r, la produttività ricevuta dallo storage del broker è la seguente:

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

La cosa può essere ulteriormente semplificata:

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

Utilizzando questa formula è possibile selezionare la piattaforma ONTAP più adatta alle proprie esigenze di livello caldo Kafka.

La tabella seguente illustra la produttività prevista del produttore per l'A900 con diversi fattori di replicazione:

Fattore di replicazione	Produzione del produttore (GPps)
3 (misurato)	3,4
2	5,1
1	10,2

Conclusione

La soluzione NetApp per il problema assurdo della ridenominazione fornisce una forma di archiviazione semplice, economica e gestita centralmente per carichi di lavoro che in precedenza erano incompatibili con NFS.

Questo nuovo paradigma consente ai clienti di creare cluster Kafka più gestibili, più facili da migrare e da rispecchiare ai fini del disaster recovery e della protezione dei dati. Abbiamo anche visto che NFS offre ulteriori vantaggi, come un utilizzo ridotto della CPU e tempi di ripristino più rapidi, un'efficienza di archiviazione notevolmente migliorata e prestazioni migliori tramite NetApp ONTAP.

Dove trovare ulteriori informazioni

Per saperne di più sulle informazioni descritte nel presente documento, consultare i seguenti documenti e/o siti web:

- Che cos'è Apache Kafka?

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

- Cos'è la rinominazione sciocca?

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

- ONATP è letto per le applicazioni di streaming.

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

- Documentazione del prodotto NetApp

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

- Che cos'è NFS?

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

- Che cos'è la riassegnazione delle partizioni di Kafka?

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

- Che cos'è l'OpenMessaging Benchmark?

  "https://openmessaging.cloud/"

- Come si migra un broker Kafka?

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

- Come si monitora il broker Kafka con Prometheus?

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

- Piattaforma gestita per Apache Kafka

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

- Supporto per Apache Kafka

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

- Servizi di consulenza per Apache Kafka

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

Informazioni sul copyright

Informazioni sul marchio commerciale