# Carichi di lavoro Apache Kafka con storage NetApp NFS

NetApp Solutions

NetApp
April 26, 2024

# Sommario

Carichi di lavoro Apache Kafka con storage NetApp NFS . . . 1
TR-4947: Carico di lavoro Apache Kafka con storage NetApp NFS - convalida funzionale e performance . . . 1
Soluzione NetApp per problemi di ridenominazione sciocco per carichi di lavoro da NFS a Kafka . . . 2
Convalida funzionale - correzione del ridenominazione senza problemi . . . 3
Perché scegliere NetApp NFS per i workload Kafka? . . . 8
Panoramica delle performance e validazione in AWS . . . 20
Panoramica e convalida delle performance in AWS FSX per NetApp ONTAP . . . 33
Panoramica e validazione delle performance con AFF A900 on-premise . . . 41
Conclusione . . . 48
Dove trovare ulteriori informazioni . . . 48

# Carichi di lavoro Apache Kafka con storage NetApp NFS

## TR-4947: Carico di lavoro Apache Kafka con storage NetApp NFS - convalida funzionale e performance

Shantanu Chakole, Karthikeyan Nagalingam e Joe Scott, NetApp

Kafka è un sistema di messaggistica distribuito publish-subscribe con una solida coda in grado di accettare grandi quantità di dati dei messaggi. Con Kafka, le applicazioni possono scrivere e leggere i dati su argomenti in modo molto rapido. A causa della sua tolleranza agli errori e della sua scalabilità, Kafka viene spesso utilizzato nel grande spazio di dati come un modo affidabile per acquisire e spostare molti flussi di dati molto rapidamente. I casi di utilizzo includono elaborazione dello streaming, monitoraggio delle attività del sito Web, raccolta e monitoraggio delle metriche, aggregazione dei log, analisi in tempo reale e così via.

Anche se le normali operazioni di Kafka su NFS funzionano bene, il "ridenominazione sciocco" Il problema blocca l'applicazione durante il ridimensionamento o la partizione di un cluster Kafka in esecuzione su NFS. Si tratta di un problema significativo perché un cluster Kafka deve essere ridimensionato o ripartizionato a scopo di bilanciamento del carico o di manutenzione. Ulteriori dettagli sono disponibili "qui".

Questo documento descrive i seguenti argomenti:

- Il problema di ridenominazione e la convalida della soluzione
- Riduzione dell'utilizzo della CPU per ridurre i tempi di attesa i/O.
- Tempi di recovery più rapidi per i broker Kafka
- Performance nel cloud e on-premise

### Perché utilizzare lo storage NFS per i workload Kafka?

I carichi di lavoro Kafka nelle applicazioni di produzione possono eseguire lo streaming di enormi quantità di dati tra le applicazioni. Questi dati vengono conservati e memorizzati nei nodi del broker Kafka nel cluster Kafka. Kafka è nota anche per la disponibilità e il parallelismo, che si ottiene suddividendo gli argomenti in partizioni e replicando le partizioni in tutto il cluster. Ciò significa che l'enorme quantità di dati che scorre attraverso un cluster Kafka è generalmente moltiplicata per dimensioni. NFS rende il ribilanciamento dei dati con il variare del numero di broker molto rapido e semplice. Per ambienti di grandi dimensioni, ribilanciamento dei dati in DAS quando il numero di broker cambia è molto lungo e, nella maggior parte degli ambienti Kafka, il numero di broker cambia frequentemente.

Altri vantaggi includono:

- **Maturità.** NFS è un protocollo maturo, il che significa che la maggior parte degli aspetti dell'implementazione, della protezione e dell'utilizzo sono ben noti.
- **Open.** NFS è un protocollo aperto e il suo continuo sviluppo è documentato nelle specifiche Internet come protocollo di rete libero e aperto.
- * Conveniente.* NFS è una soluzione a basso costo per la condivisione di file di rete facile da configurare perché utilizza l'infrastruttura di rete esistente.

- **Gestione centralizzata.** la gestione centralizzata di NFS riduce la necessità di aggiungere software e spazio su disco nei singoli sistemi utente.
- **Distributed.** NFS può essere utilizzato come file system distribuito, riducendo la necessità di dispositivi di storage su supporti rimovibili.

### Perché scegliere NetApp per i workload 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 di NetApp, offre anche i seguenti vantaggi:

- Affidabilità ed efficienza
- Scalabilità e performance
- Alta disponibilità (partner ha in un cluster NetApp ONTAP)
- Protezione dei dati
  - **Disaster Recovery (NetApp SnapMirror).** il tuo sito non funziona o vuoi partire da un altro sito e continuare da dove hai interrotto.
  - Gestibilità del sistema storage (amministrazione e gestione con NetApp OnCommand).
  - **Bilanciamento del carico.** il cluster consente di accedere a volumi diversi da file di dati LIF ospitati su nodi diversi.
  - **Operazioni senza interruzioni.** le LIF o gli spostamenti dei volumi sono trasparenti per i client NFS.

## Soluzione NetApp per problemi di ridenominazione sciocco per carichi di lavoro da NFS a Kafka

Kafka si basa sul presupposto che il file system sottostante sia conforme a POSIX, ad esempio XFS o Ext4. Il ribilanciamento delle risorse 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, il file viene rimosso solo dopo che tutti i riferimenti al file sono stati rimossi. Se il file system sottostante è collegato in rete, il client NFS intercetta le chiamate di sconnessione e gestisce il flusso di lavoro. Poiché il file non è collegato, il client NFS invia una richiesta di ridenominazione al server NFS e, all'ultima chiusura del file non collegato, effettua un'operazione di rimozione del file rinominato. Questo comportamento viene comunemente definito come ridenominazione sciocco di NFS ed è orchestrato dal client NFS.

Qualsiasi broker Kafka che utilizza lo storage da un server NFSv3 si verifica in 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 proprietà al client NFS al momento dell'apertura del file. Il client NFS interrompe quindi la gestione di unlink quando vengono aperte le porte 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 server NFS note che supportassero questa funzionalità opzionale.

Le seguenti modifiche sono necessarie per consentire al server NFS e al client NFS di risolvere il problema del ridenominazione sciocco:

- **Modifiche al client NFS (Linux).** al momento dell'apertura del file, il server NFS risponde con un flag, che

indica la capacità di gestire lo scollegamento dei file aperti. Le modifiche al lato client NFS consentono al server NFS di gestire lo scollegamento 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 dell'apertura. Lo scollegamento su un file aperto esistente è ora gestito dal server per corrispondere alla semantica POSIX. Quando l'ultima apertura viene chiusa, il server NFS avvia la rimozione effettiva del file ed evita così il processo di ridenominazione sciocco. Il server NFS di ONTAP ha implementato questa funzionalità nella sua ultima versione, ONTAP 9.12.1.

Con le suddette modifiche al client e al server NFS, Kafka può sfruttare in tutta sicurezza tutti i vantaggi dello storage NFS collegato alla rete.

# Convalida funzionale - correzione del ridenominazione senza problemi

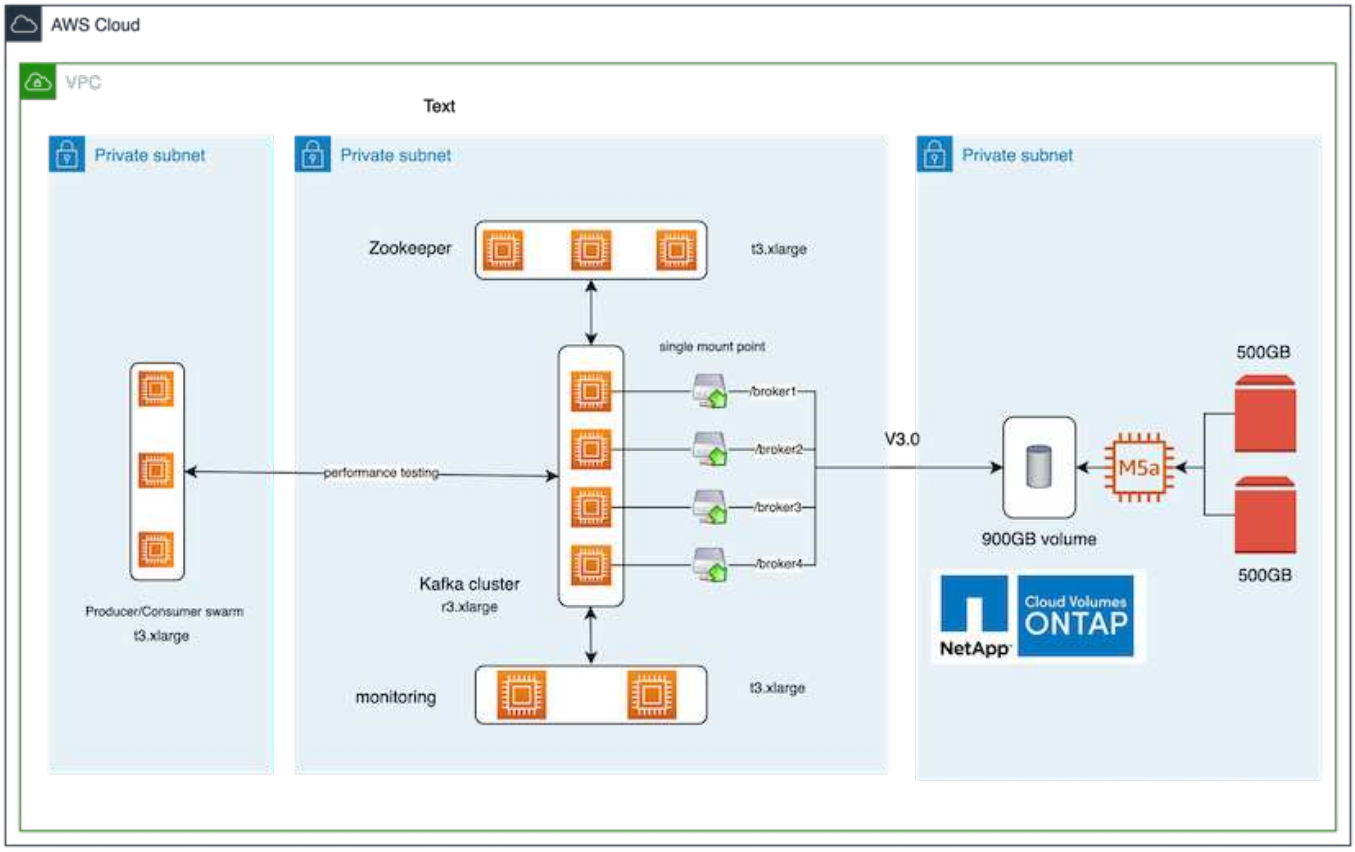
Per la convalida funzionale, abbiamo mostrato che un cluster Kafka con un montaggio NFSv3 per lo storage non esegue operazioni Kafka come la ridistribuzione delle partizioni, mentre un altro cluster montato su NFSv4 con la correzione può eseguire le stesse operazioni senza interruzioni.

## Configurazione della convalida

La configurazione viene eseguita su AWS. La seguente tabella mostra i diversi componenti della piattaforma e la configurazione ambientale utilizzati per la convalida.

| Componente della piattaforma | Configurazione dell'ambiente |
|---|---|
| Confluent Platform versione 7.2.1 | • 3 zookeeper – t3.xlarge<br>• 4 server di broker – r3.xlarge<br>• 1 x Grafana – t3.xlarge<br>• 1 centro di controllo – t3.xlarge<br>• 3 x Produttore/consumatore |
| Sistema operativo su tutti i nodi | RHEL8.7o versione successiva |
| Istanza di NetApp Cloud Volumes ONTAP | Istanza a nodo singolo – M5.2xLarge |

La figura seguente mostra la configurazione architetturale per questa soluzione.




## Flusso architettonico

- **Compute.** abbiamo utilizzato un cluster Kafka a quattro nodi con un ensemble di zookeeper a tre nodi in esecuzione su server dedicati.
- **Monitoring.** abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana.
- **Workload.** per la generazione dei workload, abbiamo utilizzato un cluster a tre nodi separato in grado di produrre e consumare da questo cluster Kafka.
- **Storage.** abbiamo utilizzato un'istanza NetApp Cloud Volumes ONTAP a nodo singolo con due volumi AWS-EBS GP2 da 500 GB collegati all'istanza. Questi volumi sono stati quindi esposti al cluster Kafka come singolo volume NFSv4.1 attraverso un LIF.

Le proprietà predefinite di Kafka sono state scelte per tutti i server. Lo stesso è stato fatto per lo sciame dello zooteeper.

## Metodologia di test

1. Aggiornare `-is-preserve-unlink-enabled true` al volume 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:
    - **Cluster 1.** il server NFS v4.1 di back-end con ONTAP versione 9.12.1 pronto per la produzione è stato ospitato da un'istanza CVO di NetApp. RHEL 8.7/RHEL 9.1 è stato installato sui broker.
    - **Cluster 2.** il server NFS back-end era un server Linux NFSv3 generico creato manualmente.
3. È stato creato un argomento dimostrativo su entrambi i cluster Kafka.

   Cluster 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:
```

   Cluster 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. I dati sono stati caricati in questi nuovi argomenti creati per entrambi i cluster. Questa operazione è stata eseguita utilizzando il toolkit Producer-perf-test incluso nel pacchetto predefinito di Kafka:

```
./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 il 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 dei broker su entrambi i cluster:

```
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...
Connected to 172.30.0.198.
Escape character is '^]'.
^[
```

6. Per attivare la condizione di errore che causa il crash dei cluster Kafka che utilizzano i volumi di storage NFSv3, abbiamo avviato il processo di riassegnazione delle partizioni su entrambi i cluster. La riassegnazione delle partizioni 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 JSON (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 effettiva delle partizioni 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 di completamento della riassegnazione, un altro controllo dello stato di salute dei broker ha dimostrato che il cluster che utilizza volumi di storage NFSv3 ha riscontrato un problema di ridenominazione sciocco e si è bloccato, mentre il cluster 1 utilizza volumi di storage NetApp ONTAP NFSv4.1 con la correzione delle operazioni 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 di log di Kafka, era chiaro che il cluster 1 che utilizzava i volumi di storage NetApp ONTAP NFSv4.1 con la correzione aveva un'assegnazione pulita delle partizioni, mentre il cluster 2 utilizzava lo storage NFSv3 generico non era dovuto a problemi di ridenominazione sciocco, che hanno portato al crash. La figura seguente mostra il ribilanciamento delle partizioni del cluster 2, che ha causato un problema di ridenominazione sciocco sullo 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
```

La figura seguente mostra un ribilanciamento pulito delle partizioni del cluster 1 utilizzando lo storage NetApp NFSv4.1.

```
/demo/broker_demo_1/__a_demo_topic-0:
total 710932
drwxr-xr-x.  2 nobody nobody      4096 Sep 19 10:26 .
drwxr-xr-x. 85 nobody nobody      8192 Sep 19 10:37 ..
-rw-r--r--.  1 nobody nobody  10485760 Sep 19 10:25 00000000000000000000.index
-rw-r--r--.  1 nobody nobody 724167522 Sep 19 10:25 00000000000000000000.log
-rw-r--r--.  1 nobody nobody  10485756 Sep 19 10:25 00000000000000000000.timeindex
-rw-r--r--.  1 nobody nobody         0 Sep 19 10:15 leader-epoch-checkpoint
-rw-r--r--.  1 nobody nobody        43 Sep 19 10:15 partition.metadata

/demo/broker_demo_1/__a_demo_topic-2:
total 780016
drwxr-xr-x.  2 nobody nobody      4096 Sep 19 10:35 .
drwxr-xr-x. 85 nobody nobody      8192 Sep 19 10:37 ..
-rw-r--r--.  1 nobody nobody  10485760 Sep 19 10:36 00000000000000000000.index
-rw-r--r--.  1 nobody nobody 794575786 Sep 19 10:36 00000000000000000000.log
-rw-r--r--.  1 nobody nobody  10485756 Sep 19 10:36 00000000000000000000.timeindex
-rw-r--r--.  1 nobody nobody         0 Sep 19 10:35 leader-epoch-checkpoint
-rw-r--r--.  1 nobody nobody        43 Sep 19 10:35 partition.metadata
```

# Perché scegliere NetApp NFS per i workload Kafka?

Ora che esiste una soluzione per il problema del ridenominazione sciocco nello storage NFS con Kafka, è possibile creare solide implementazioni che sfruttano lo storage NetApp ONTAP per il carico di lavoro Kafka. Questo non solo riduce significativamente l'overhead operativo, ma offre anche i seguenti vantaggi ai cluster Kafka:

- **Utilizzo ridotto della CPU per i broker Kafka.** l'utilizzo dello storage NetApp ONTAP disaggregato separa le operazioni di i/o dei dischi dal broker e riduce così l'impatto della CPU.
- **Tempi di recovery più rapidi per i broker.** poiché lo storage NetApp ONTAP disaggregato è condiviso tra i nodi dei broker Kafka, una nuova istanza di calcolo può sostituire un broker difettoso in qualsiasi momento in una frazione del tempo rispetto alle implementazioni Kafka convenzionali senza ricostruire i dati.
- **Efficienza dello storage.** con il provisioning del layer di storage dell'applicazione tramite NetApp ONTAP, i clienti possono sfruttare tutti i vantaggi dell'efficienza dello storage forniti con ONTAP, come compressione dei dati in linea, deduplica e compaction.

Questi vantaggi sono stati testati e validati in casi di test di cui discutiamo in dettaglio in questa sezione.

## Riduzione dell'utilizzo 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 che erano identici nelle specifiche tecniche ma differivano nelle tecnologie di storage. Non solo l'utilizzo complessivo della CPU è inferiore quando il cluster Kafka utilizza lo storage ONTAP, ma l'aumento dell'utilizzo della CPU ha dimostrato un gradiente più delicato rispetto a un cluster Kafka basato su DAS.

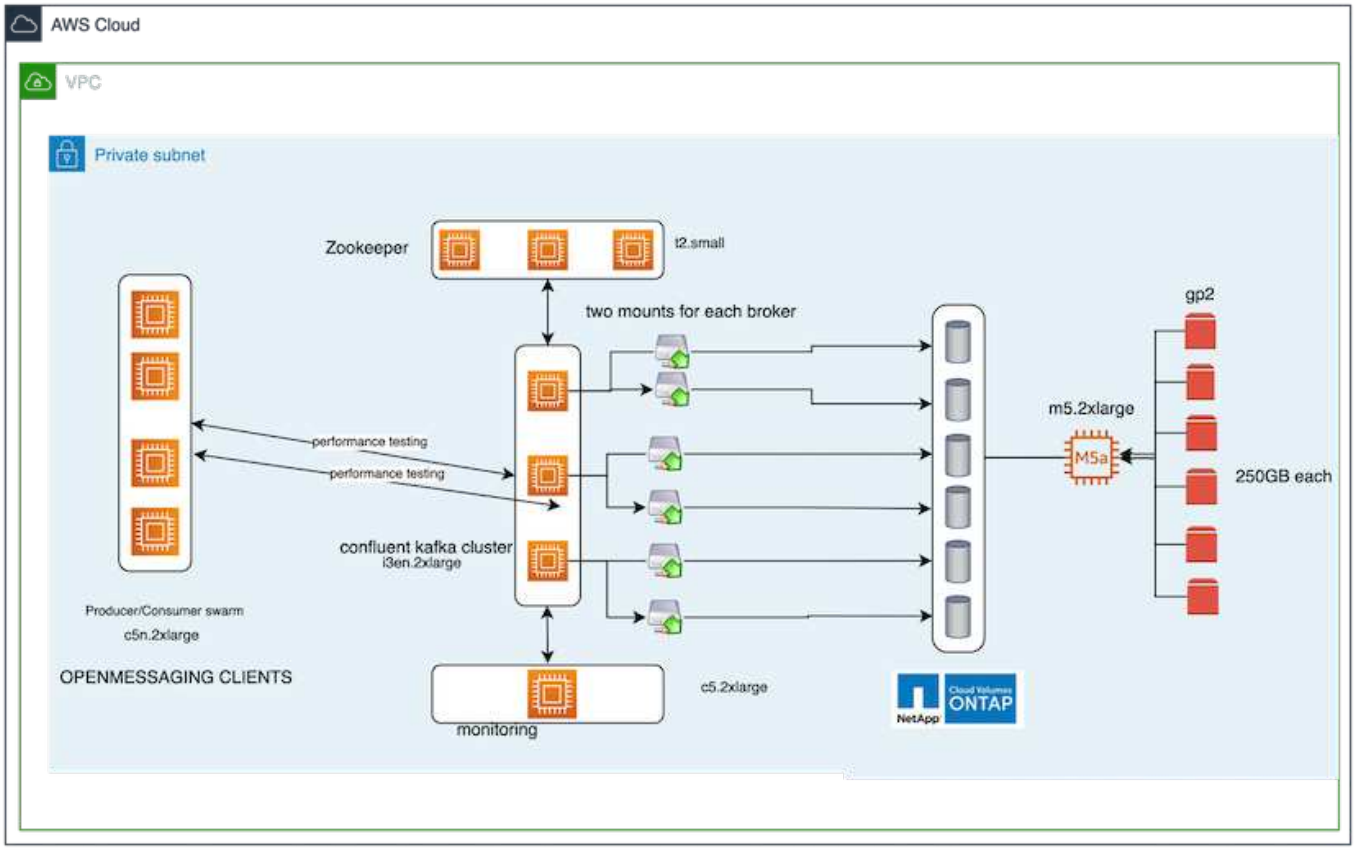
### Configurazione architetturale

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

| Componente della piattaforma | Configurazione dell'ambiente |
|---|---|
| Tool di benchmarking Kafka 3.2.3: OpenMessaging | • 3 zookeeper – t2.small<br>• 3 server di broker – i3en.2xlarge<br>• 1 x Grafana – c5n.2xlarge<br>• 4 x Produttore/Consumer — c5n.2xlarge |
| Sistema operativo su tutti i nodi | RHEL 8.7 o versione successiva |
| Istanza di NetApp Cloud Volumes ONTAP | Istanza a nodo singolo – M5.2xLarge |

### Tool di benchmarking

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




- **Compute.** abbiamo utilizzato un cluster Kafka a tre nodi con un ensemble di zookeeper a tre nodi in esecuzione su server dedicati. Ciascun broker disponeva di due mount point NFSv4.1 su un singolo volume sull'istanza CVO di NetApp attraverso una LIF dedicata.
- **Monitoring.** abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per la generazione dei carichi di lavoro, abbiamo un cluster a tre nodi separato che può produrre e consumare da questo cluster Kafka.

- **Storage.** abbiamo utilizzato un'istanza NetApp Cloud Volumes ONTAP a nodo singolo con sei volumi AWS-EBS GP2 da 250 GB montati sull'istanza. Questi volumi sono stati quindi esposti al cluster Kafka come sei volumi NFSv4.1 attraverso LIF dedicate.
- **Configurazione.** i due elementi configurabili in questo test case erano i broker Kafka e i carichi di lavoro OpenMessaging.
  - **Broker config.** per i broker Kafka sono state selezionate le seguenti specifiche. Abbiamo utilizzato il fattore di replica di 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
```

- **OpenMessaging benchmark (OMB) workload config.** 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 un proprio set di benchmark di sciami di cluster.
   - Cluster 1.* cluster Kafka basato su NFS.

- Cluster 2.* cluster Kafka basato su DAS.

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

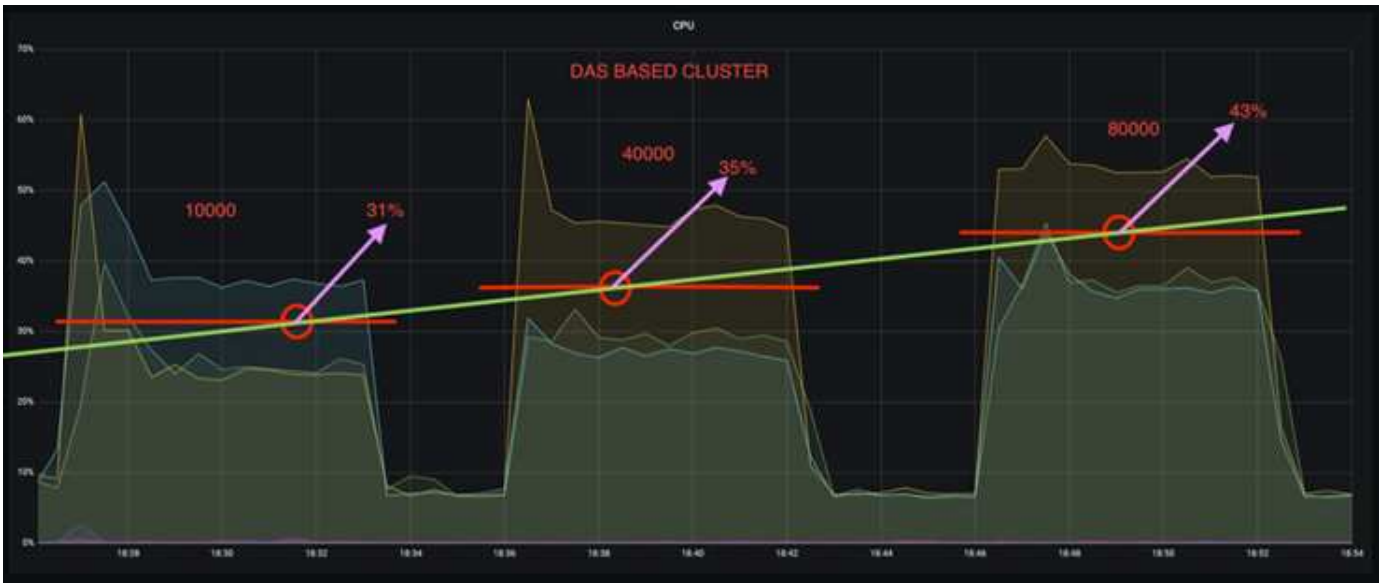
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

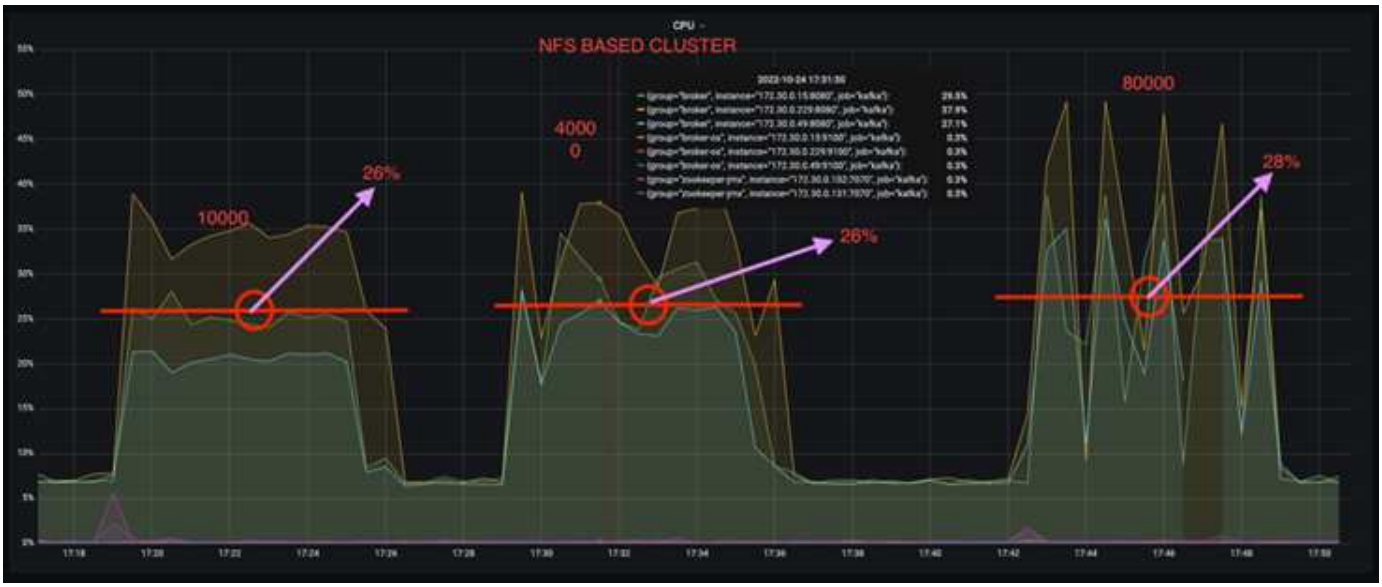
3. La configurazione del tasso di produzione è stata aumentata in quattro iterazioni e l'utilizzo della CPU è stato registrato con Grafana. Il tasso di produzione è stato impostato sui seguenti livelli:
   - 10,000
   - 40,000
   - 80,000
   - 100,000

**Osservazione**

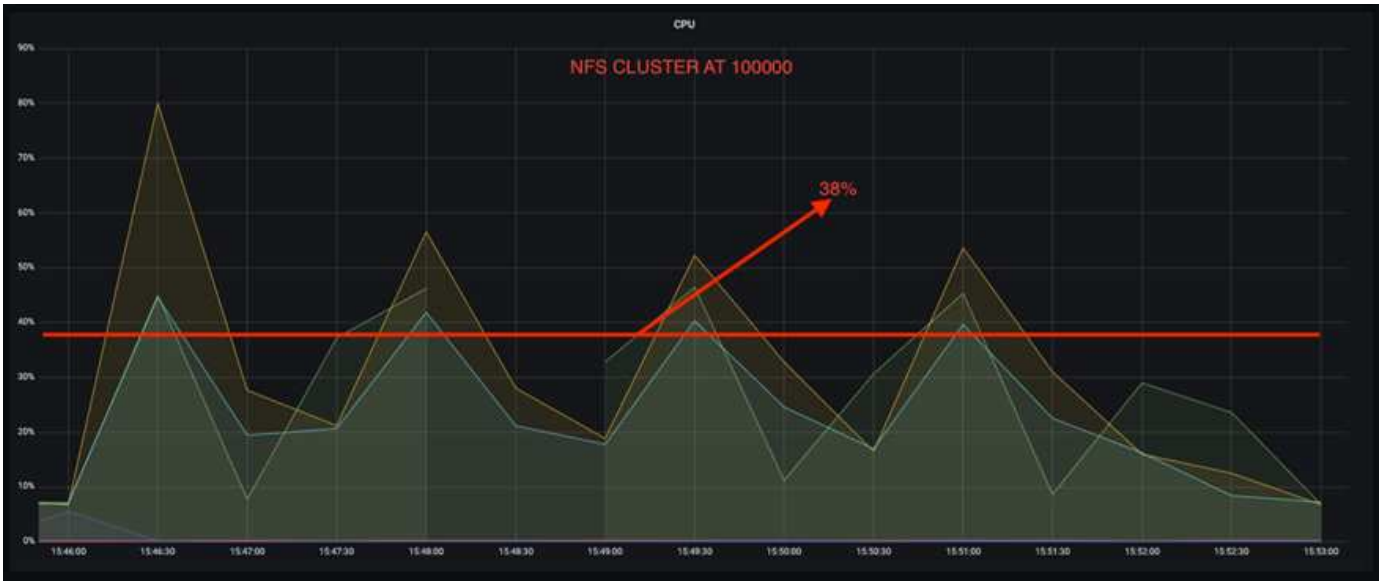
L'utilizzo dello storage NetApp NFS con Kafka offre due vantaggi principali:

- **È possibile ridurre l'utilizzo della CPU di quasi un terzo.** l'utilizzo complessivo della CPU con carichi di lavoro simili è stato inferiore per NFS rispetto agli SSD DAS; i risparmi variano dal 5% per velocità di produzione inferiori al 32% per velocità di produzione superiori.
- **Una riduzione di tre volte nella deriva di utilizzo della CPU a velocità di produzione più elevate.** come previsto, si è verificata una deriva verso l'alto per l'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 inferiore al 70% per il tasso di produzione più elevato, un aumento del 39%. Tuttavia, con un backend di storage NFS, l'utilizzo della CPU è aumentato dal 26% al 38%, con un aumento del 12%.






Inoltre, con 100,000 messaggi, DAS mostra un utilizzo della CPU maggiore rispetto a un cluster NFS.

## Recupero più rapido del broker

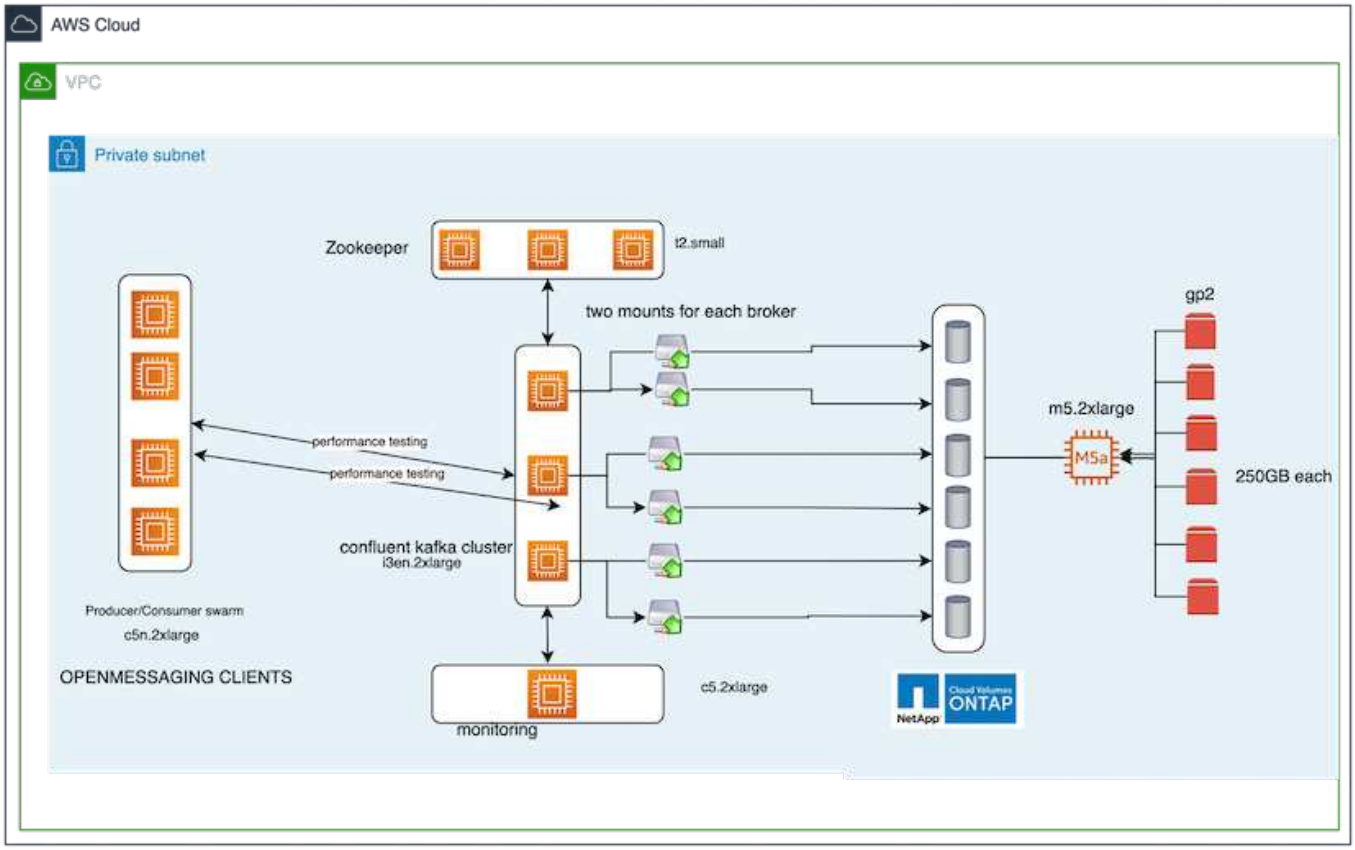
Abbiamo scoperto che i broker Kafka si ripristinano più velocemente quando utilizzano lo storage NetApp NFS condiviso. Quando un broker si blocca in un cluster Kafka, questo broker può essere sostituito da un broker sano con lo stesso ID broker. Dopo aver eseguito questo test case, abbiamo scoperto che,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

### Configurazione architetturale

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

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

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



- **Compute.** un cluster Kafka a tre nodi con un ensemble di zookeeper a tre nodi in esecuzione su server dedicati. Ciascun broker dispone di due punti di montaggio NFS per un singolo volume sull'istanza NetApp CVO tramite un LIF dedicato.
- **Monitoring.** due nodi per una combinazione Prometheus-Grafana. Per la generazione dei carichi di lavoro, utilizziamo un cluster a tre nodi separato in grado di produrre e utilizzare questo cluster Kafka.
- **Storage.** un'istanza NetApp Cloud Volumes ONTAP 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 attraverso LIF dedicate.
- **Configurazione Broker.** l'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 alto perché questo determina la velocità con cui un determinato nodo viene estratto dall'elenco ISR. Quando si passa da un nodo cattivo a un nodo integro, non si desidera che l'ID broker sia 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 ciascuno dei broker.
   - **Cluster basato su EC2.** Su Cui è mappata Una directory di dati del broker Kafka `/mnt/data-2` (Nella figura seguente, Broker-1 del cluster1 [terminale sinistro]).
   - **Cluster NetApp basato su NFS.** Una directory di dati del broker Kafka è montata su NFS point `/mnt/data` (Nella figura seguente, Broker-1 del cluster2 [terminale destro]).


4. In ciascuno dei cluster, il broker-1 è stato terminato per attivare un processo di recovery del broker non riuscito.
5. Una volta terminato il broker, l'indirizzo IP del broker è stato assegnato come IP secondario al broker di standby. Ciò era necessario perché un broker in un cluster Kafka è identificato da quanto segue:
   - **Indirizzo IP.** assegnato riassegnando l'IP del broker guasto al broker di standby.
   - **Broker ID.** questa opzione è stata configurata nel broker di standby `server.properties`.
6. Al momento dell'assegnazione IP, il servizio Kafka è stato avviato sul broker di standby.
7. Dopo un po', i log del server sono stati estratti per controllare il tempo impiegato per creare i dati sul nodo sostitutivo nel cluster.

### Osservazione

Il recupero del broker Kafka è stato quasi nove volte più veloce. Il tempo necessario per ripristinare un nodo broker guasto è risultato notevolmente più veloce quando si utilizza lo storage condiviso NetApp NFS rispetto all'utilizzo di SSD DAS in un cluster Kafka. Per 1 TB di dati su argomenti, 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 del broker, mentre il cluster basato su NFS ha completato il ripristino in 3 minuti. Abbiamo anche osservato nei log che gli offset consumer per le partizioni EC2 erano 0, mentre nel cluster NFS gli offset consumer sono stati rilevati 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 iniziato 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 richiede 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**

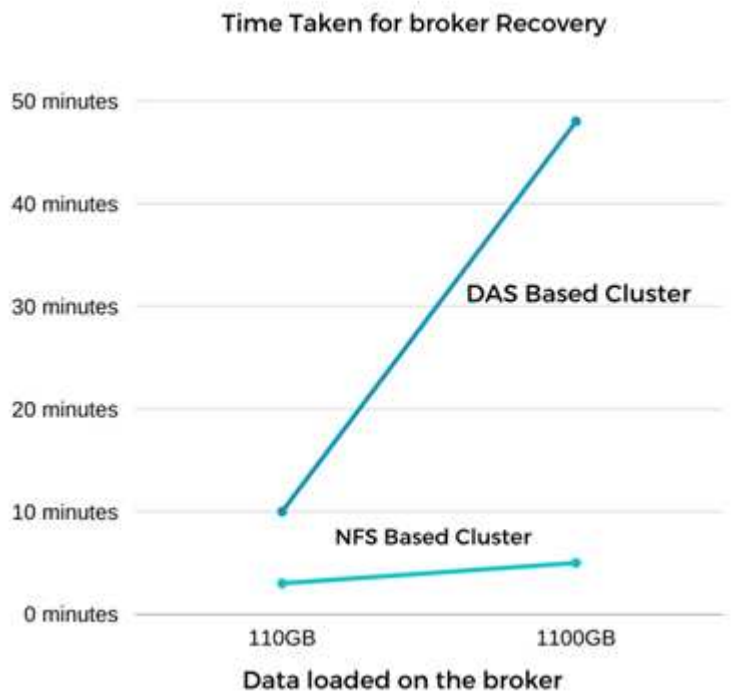
1. Il nodo di backup è stato avviato alle 09:39:17,213. La voce del registro di avvio viene evidenziata di seguito.

```
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
9  [2022-10-31 09:39:17,244] INFO Client environment:java.version=11.0.17 (org.apache
```

2. Il processo di ricostruzione dei dati è terminato alle 09:42:29,115. L'elaborazione di 110 GB di dati richiede 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 i broker contenenti circa 1 TB di dati, che hanno richiesto circa 48 minuti per il DAS e 3 minuti per NFS. I risultati sono illustrati nel seguente grafico.



## Efficienza dello storage

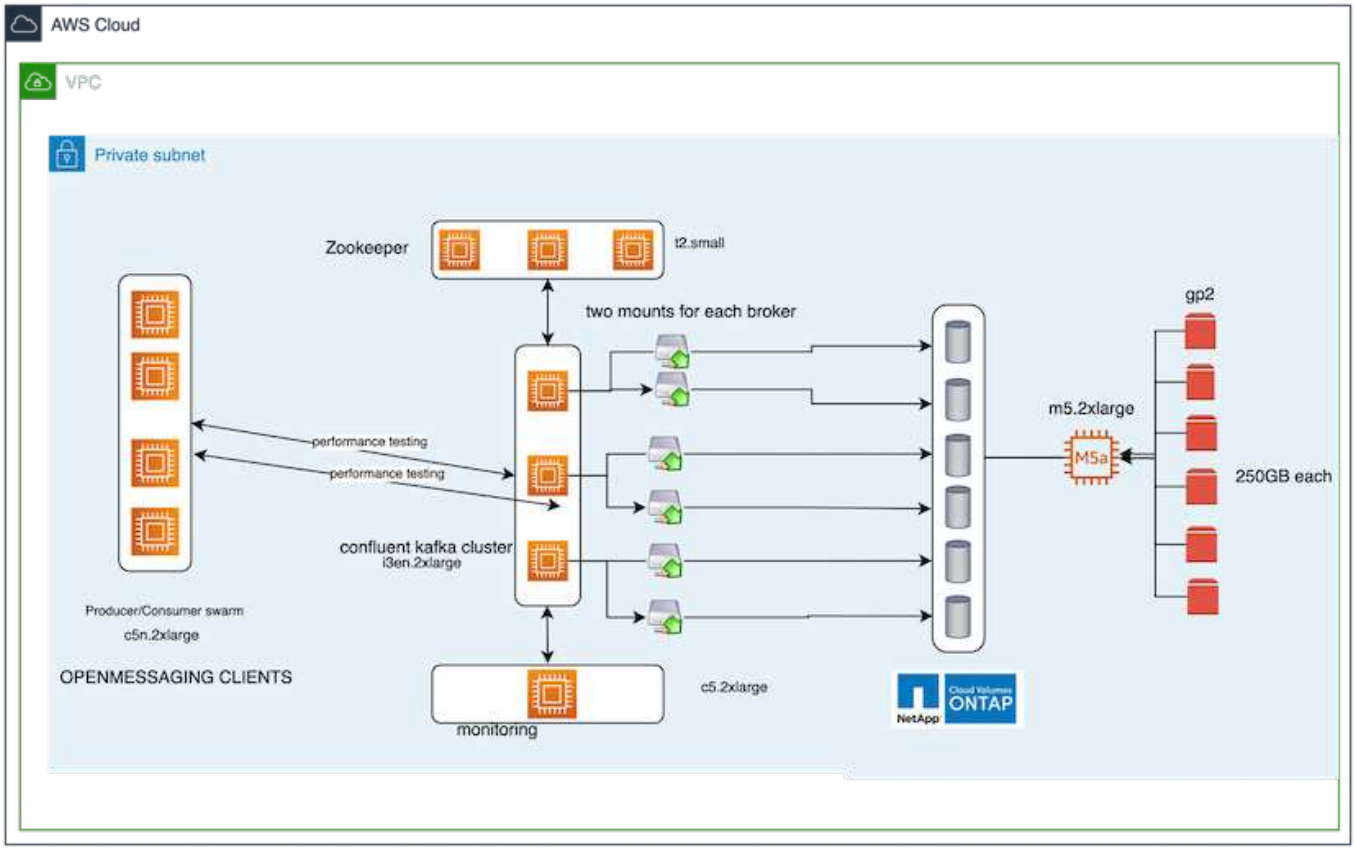
Poiché il provisioning del layer di storage del cluster Kafka è stato eseguito tramite NetApp ONTAP, abbiamo ottenuto tutte le funzionalità di efficienza dello storage di ONTAP. Questo è stato testato generando una quantità significativa di dati su un cluster Kafka con storage NFS fornito su Cloud Volumes ONTAP. Abbiamo potuto constatare che le funzionalità di ONTAP hanno ridotto notevolmente lo spazio.

### Configurazione architetturale

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

| Componente della piattaforma | Configurazione dell'ambiente |
|---|---|
| Kafka 3.2.3 | • 3 zookeeper – t2.small<br>• 3 server di broker – i3en.2xlarge<br>• 1 x Grafana – c5n.2xlarge<br>• 4 x produttore/consumatore — c5n.2xlargo * |
| Sistema operativo su tutti i nodi | RHEL8.7 o versione successiva |
| Istanza di NetApp Cloud Volumes ONTAP | Istanza a nodo singolo – M5.2xLarge |

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




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

- **Monitoring.** abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per la generazione dei carichi di lavoro, abbiamo utilizzato un cluster a tre nodi separato in grado di produrre e utilizzare questo cluster Kafka.
- **Storage.** abbiamo utilizzato un'istanza NetApp Cloud Volumes ONTAP a nodo singolo con sei volumi AWS-EBS GP2 da 250 GB montati sull'istanza. Questi volumi sono stati quindi esposti al cluster Kafka come sei volumi NFS attraverso LIF dedicate.
- **Configurazione.** gli elementi configurabili in questo test case erano i broker Kafka.

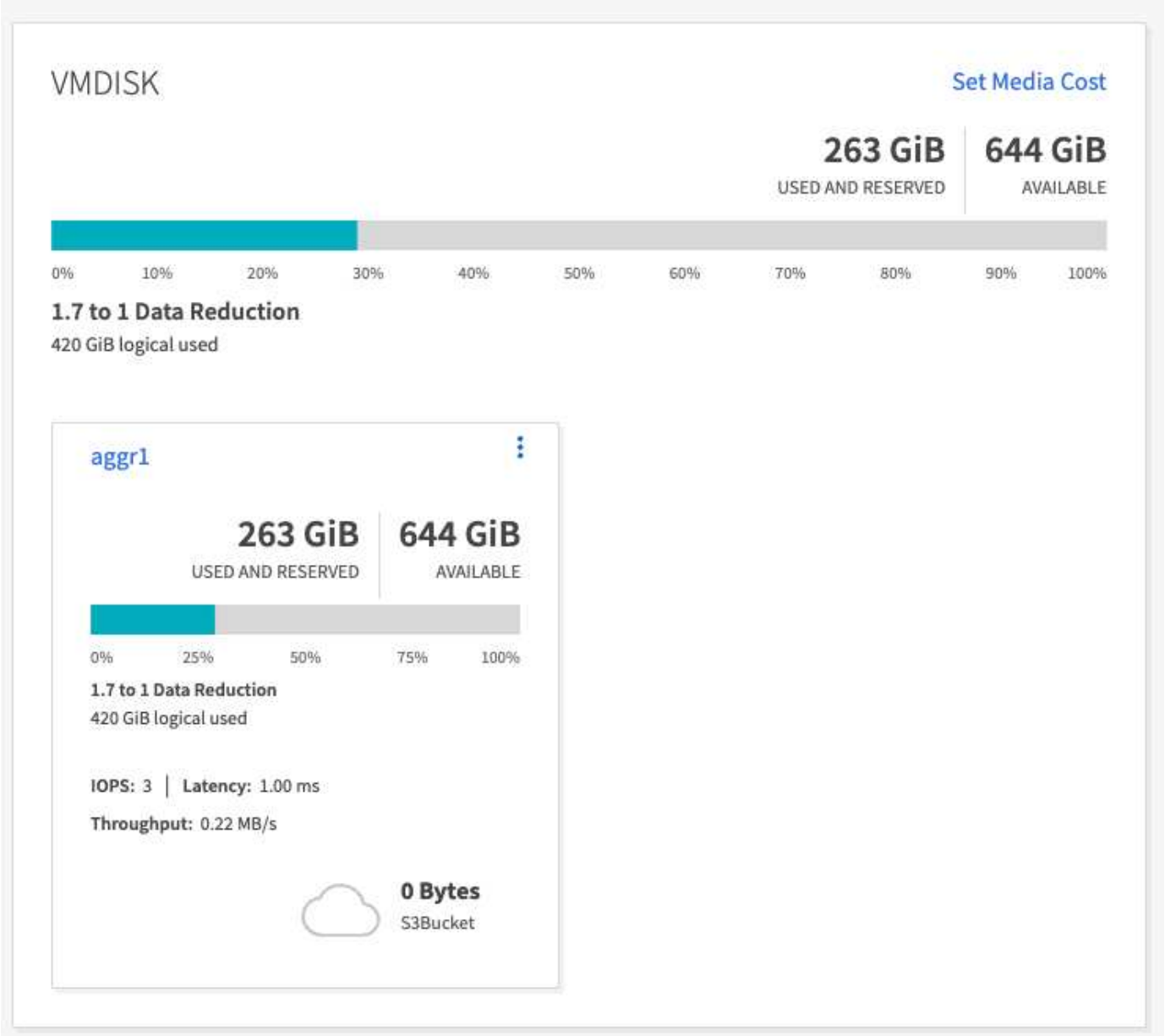
La compressione è stata disattivata alla fine del produttore, consentendo così ai produttori di generare un throughput elevato. L'efficienza dello storage è stata invece gestita dal livello di elaborazione.

### Metodologia di test

1. È stato eseguito il provisioning di un cluster Kafka con le specifiche indicate in precedenza.
2. Sul cluster, sono stati prodotti circa 350 GB di dati utilizzando il tool OpenMessaging Benchmarking.
3. Una volta completato il carico di lavoro, le statistiche sull'efficienza dello storage sono state raccolte utilizzando Gestione di sistema di ONTAP e l'interfaccia CLI.

### Osservazione

Per i dati generati con lo strumento OMB, abbiamo registrato un risparmio di spazio di ~33% con un rapporto di efficienza dello storage di 1.70:1. Come mostrato 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 delle performance e validazione in AWS

Un cluster Kafka con il layer di storage montato su NetApp NFS è stato sottoposto a benchmark per le performance 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 performance nel cloud AWS. Questo benchmarking è descritto nelle sezioni seguenti.

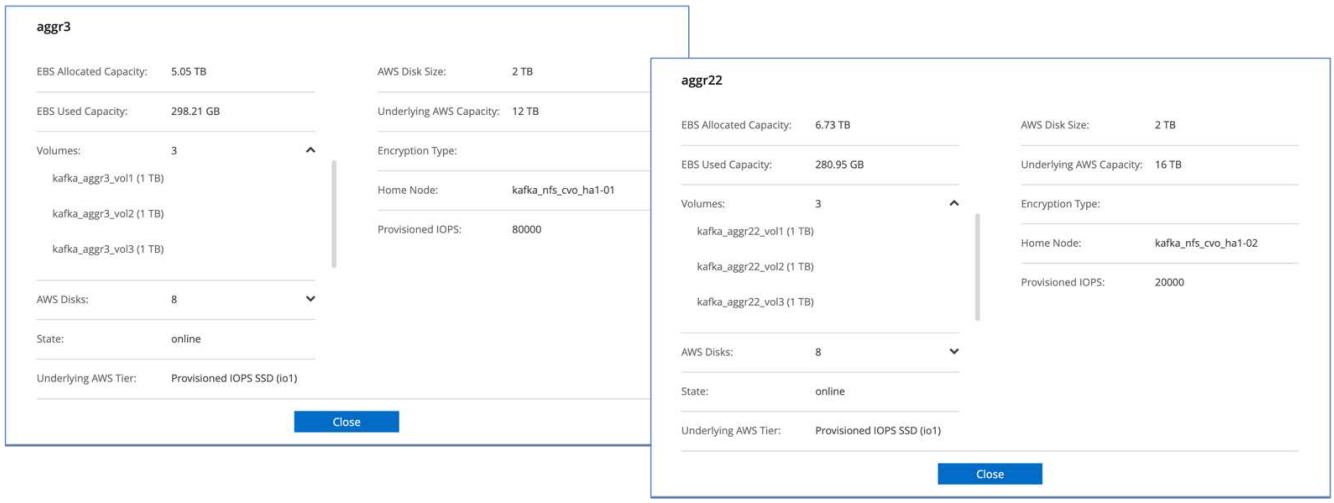
### Configurazione architetturale

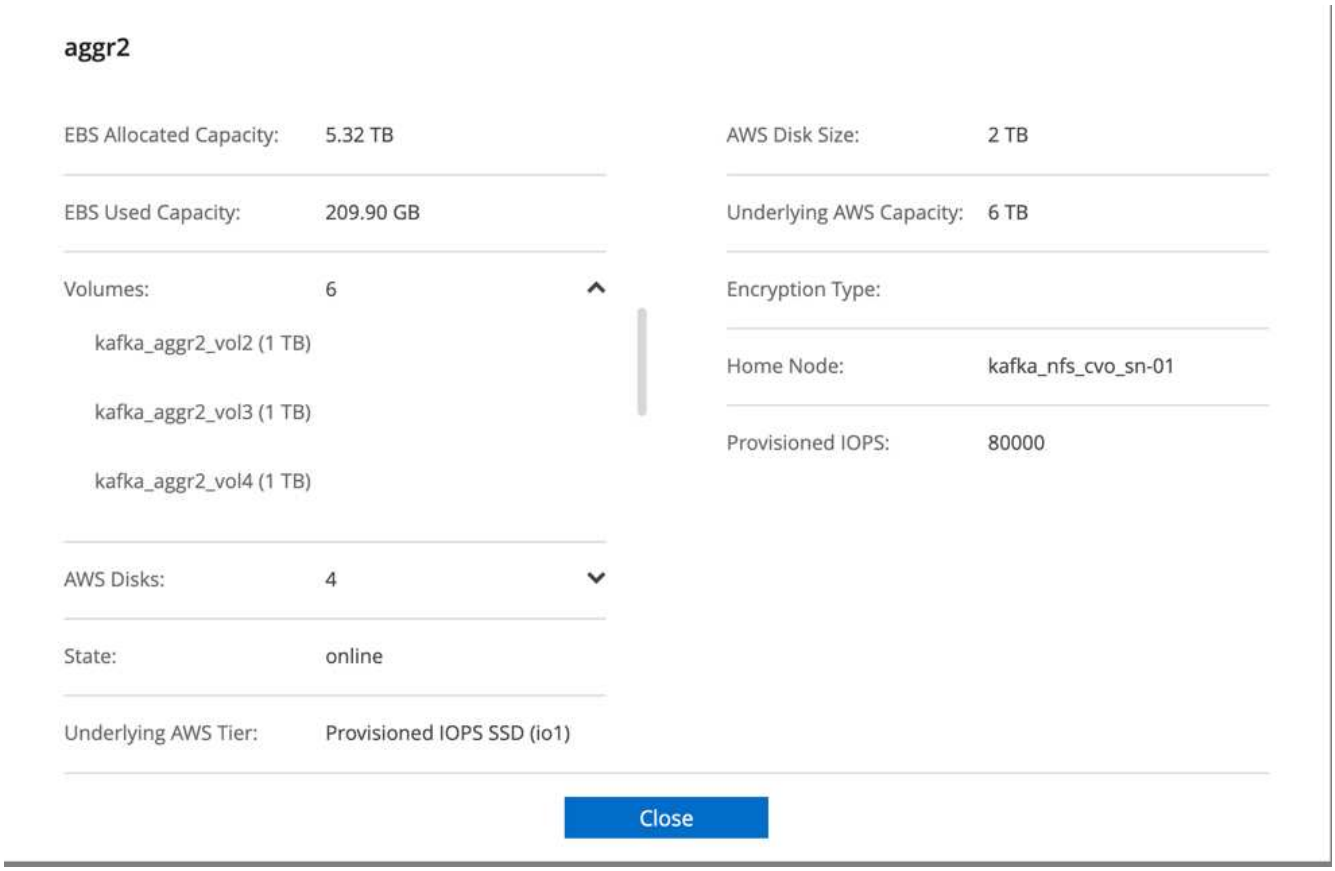
La seguente tabella mostra la configurazione ambientale per un cluster Kafka che utilizza NAS.

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

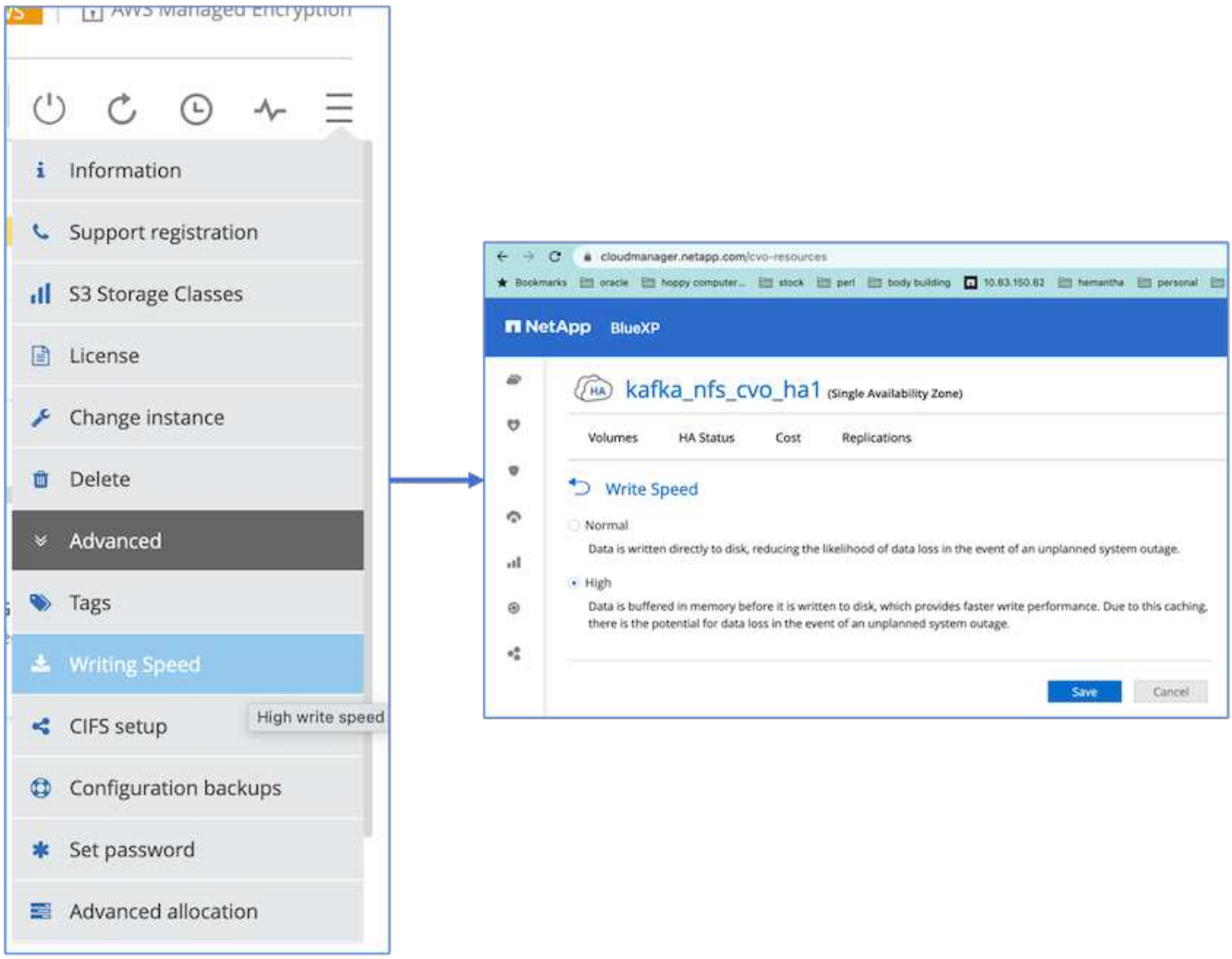
### Configurazione di NetApp Cluster Volume ONTAP

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.



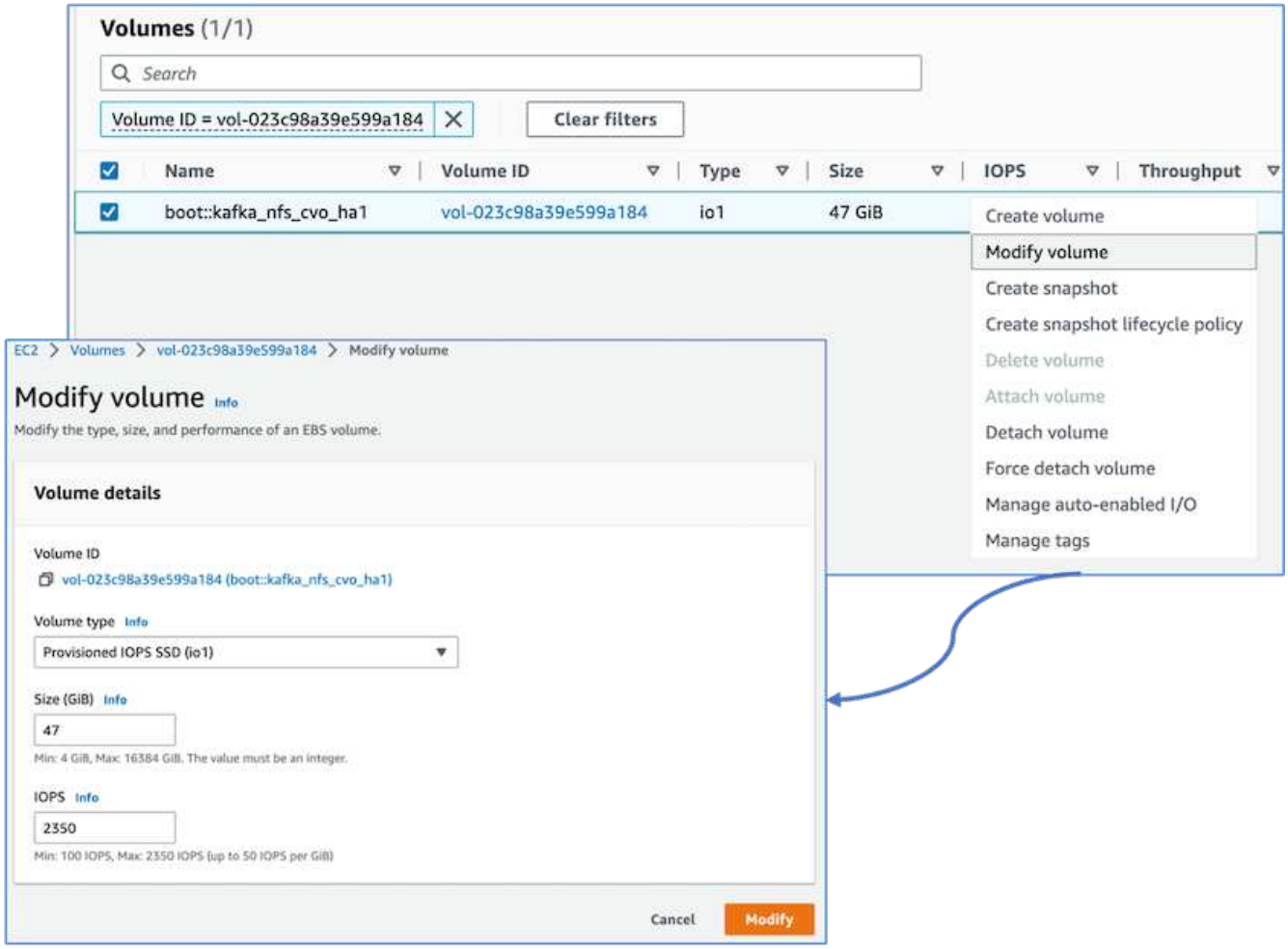



2. Per ottenere performance di rete migliori, abbiamo attivato il networking ad alta velocità sia per la coppia ha che per il singolo nodo.



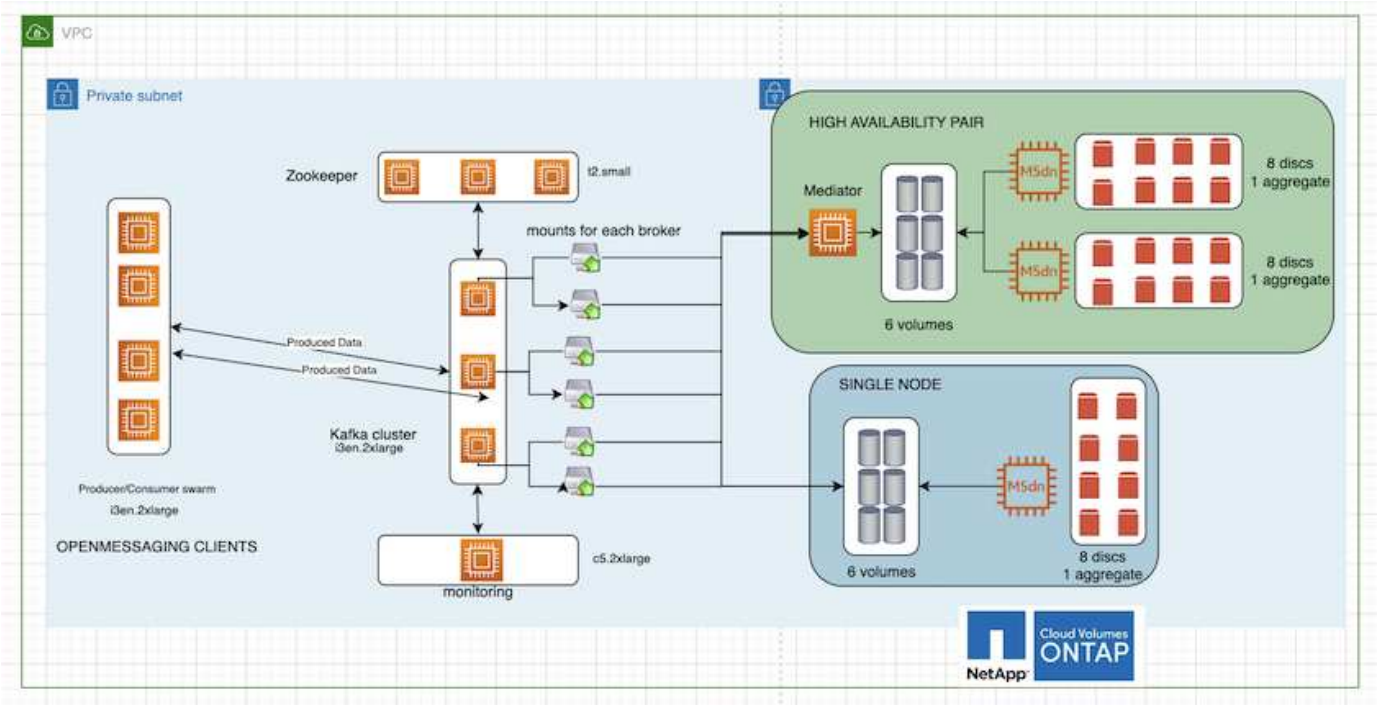

3. Abbiamo notato che la NVRAM ONTAP aveva più IOPS, quindi abbiamo modificato gli IOPS a 2350 per il volume root Cloud Volumes ONTAP. Il disco del volume root in Cloud Volumes ONTAP aveva una dimensione di 47 GB. Il seguente comando ONTAP è per la coppia ha e lo stesso passo è applicabile per il singolo 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 mostra l'architettura di un cluster Kafka basato su NAS.

- **Compute.** abbiamo utilizzato un cluster Kafka a tre nodi con un ensemble di zookeeper a tre nodi in esecuzione su server dedicati. Ciascun broker disponeva di due punti di montaggio NFS su un singolo volume nell'istanza di Cloud Volumes ONTAP tramite un LIF dedicato.
- **Monitoring.** abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per la generazione dei carichi di lavoro, abbiamo utilizzato un cluster a tre nodi separato in grado di produrre e utilizzare questo cluster Kafka.
- **Storage.** abbiamo utilizzato un'istanza di ha-Pair Cloud Volumes ONTAP con un volume AWS-EBS GP3 da 6 TB montato sull'istanza. Il volume è stato quindi esportato nel broker Kafka con un montaggio NFS.




### Configurazioni di benchmarking di OpenMessage

1. Per migliorare le performance 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 di 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. Verificare le connessioni di rete in Cloud Volumes ONTAP. Il seguente comando ONTAP viene utilizzato dal singolo nodo Cloud Volumes ONTAP. Lo stesso passaggio si applica 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 e le proprietà rimanenti sono comuni per gli broker. Per il 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 il 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 il 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, Kafka `servers.properties` È uguale alla coppia Cloud Volumes ONTAP ha, ad eccezione di `log.dirs` proprietà.
   - Per il 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 il 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 il 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 viene 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 in base al caso di utilizzo. Nel nostro test delle performance, abbiamo

utilizzato 3K.

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

- Il file yaml utilizzato per le proprietà del driver Sync 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 di 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. È stato eseguito il provisioning di un cluster Kafka secondo le specifiche descritte in precedenza utilizzando Terraform e Ansible. Il terraform viene utilizzato per costruire l'infrastruttura utilizzando istanze AWS per il cluster Kafka e Ansible crea il cluster Kafka su di essi.
2. È stato attivato un carico di lavoro OMB 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. È stato attivato un altro carico di lavoro con il driver di 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 performance di un'istanza di Kafka in esecuzione su NFS. La differenza tra i driver è la proprietà di scaricamento dei log.

Per una coppia Cloud Volumes ONTAP ha:

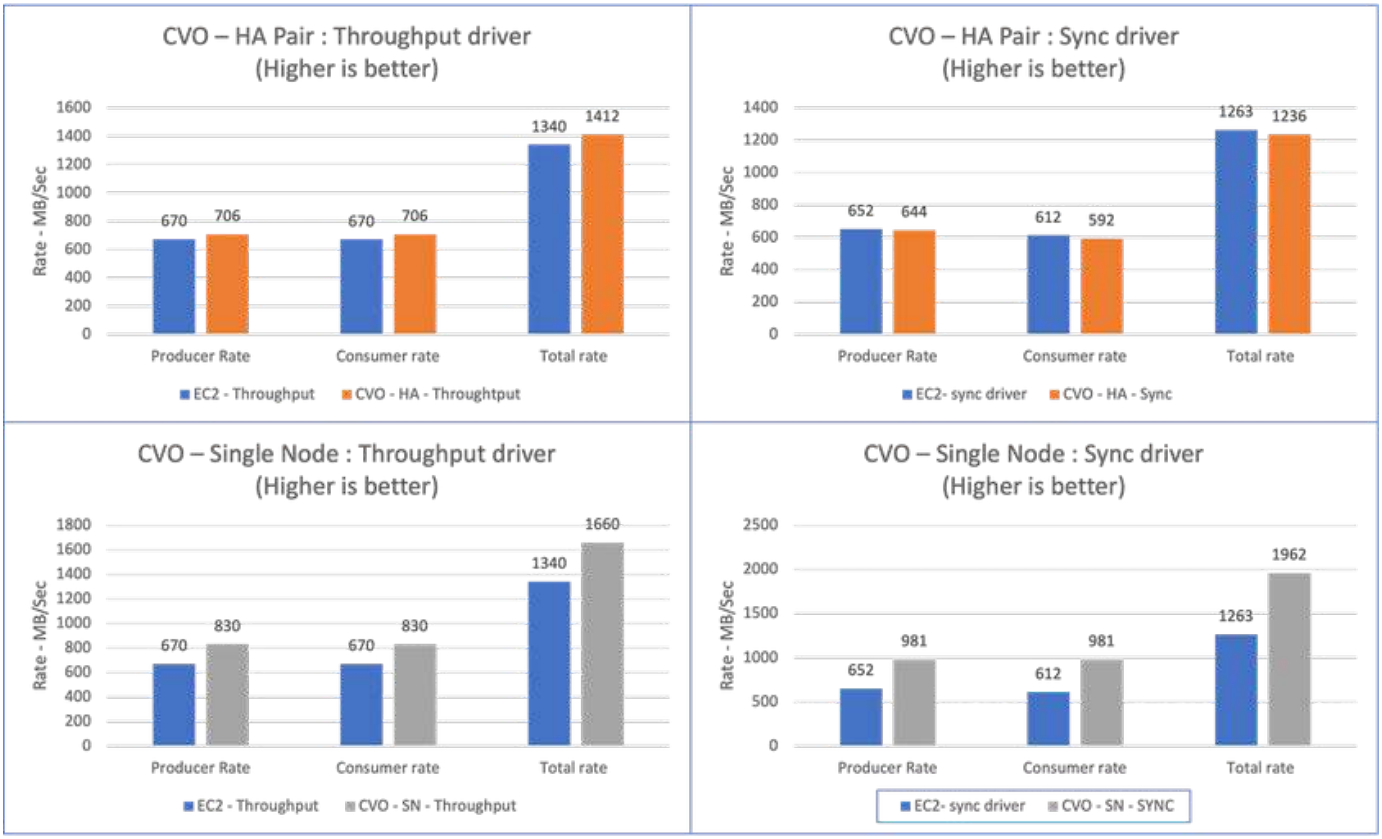
- Throughput totale generato in modo coerente dal driver Sync: ~1236 Mbps.
- Throughput totale generato per il driver di throughput: Picco ~1412 Mbps.

Per un singolo nodo Cloud Volumes ONTAP:

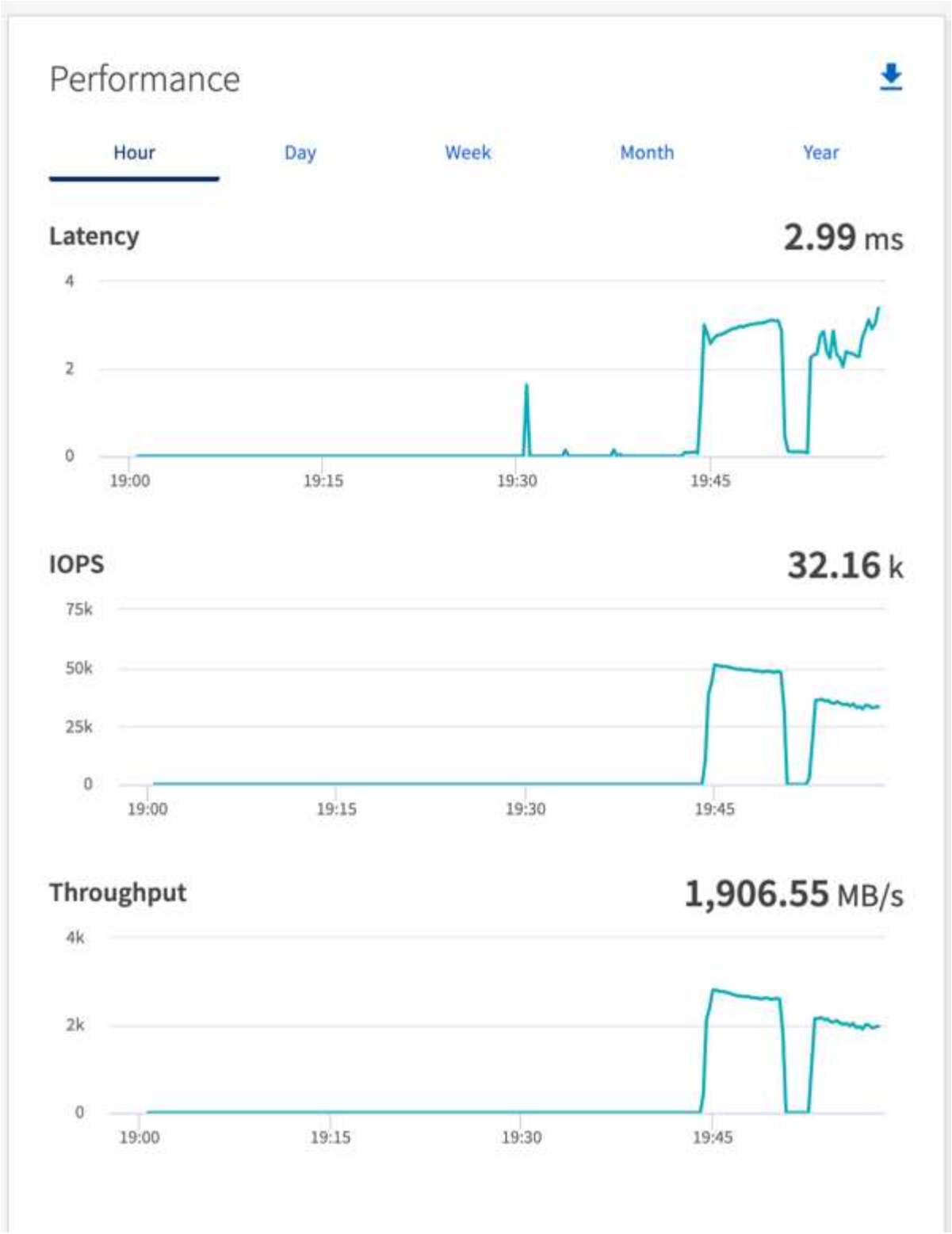
- Throughput totale generato in modo coerente dal driver Sync: ~ 1962 MBps.
- Throughput totale generato dal driver di throughput: Picco ~1660 MBps

Il driver Sync è in grado di generare un throughput coerente quando i log vengono trasferiti istantaneamente sul disco, mentre il driver di throughput genera burst di throughput quando i log vengono impegnati su disco in massa.

Questi numeri di throughput vengono generati per la configurazione AWS specificata. Per requisiti di performance più elevati, i tipi di istanze possono essere scalati e ottimizzati ulteriormente per ottenere numeri di throughput migliori. Il throughput totale o il tasso totale è la combinazione di un tasso di produttore e di consumo.




Verificare il throughput dello storage durante l'esecuzione del benchmarking del throughput o del driver di sincronizzazione.




## Panoramica e convalida delle performance in AWS FSX per NetApp ONTAP

Un cluster Kafka con il layer di storage montato su NFS NetApp è stato sottoposto a benchmark per le performance in AWS FSX per NetApp ONTAP. Gli esempi di benchmarking sono descritti nelle sezioni seguenti.

## Apache Kafka in AWS FSX per NetApp ONTAP

Network file System (NFS) è un file system di rete ampiamente utilizzato per la memorizzazione di 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 una solida architettura di acquisizione dei dati con moderne funzionalità di storage. Per consentire l'analisi in tempo reale e fornire informazioni utili, è necessaria un'infrastruttura ben progettata e dalle performance elevate.

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

Amazon FSX per NetApp ONTAP offre un file system NFS completamente gestito, scalabile e dalle performance elevate nel cloud. I dati Kafka su FSX per NetApp ONTAP possono scalare per gestire grandi quantità di dati e garantire la tolleranza agli errori. NFS offre gestione dello storage centralizzata e protezione dei dati per set di dati critici e sensibili.

Questi miglioramenti consentono ai clienti AWS di sfruttare FSX per NetApp ONTAP quando eseguono carichi di lavoro Kafka su servizi di calcolo AWS. Questi vantaggi sono:
* Riduzione dell'utilizzo della CPU per ridurre i tempi di attesa i/O.
* Tempi di recovery più rapidi per i broker Kafka.
* Affidabilità ed efficienza.
* Scalabilità e performance.
* Disponibilità multi-Availability zone.
* Protezione dei dati.

### Panoramica e convalida delle performance in AWS FSX per NetApp ONTAP

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

#### Kafka in AWS FSX per NetApp ONTAP

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

##### Configurazione architetturale

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

| Componente della piattaforma | Configurazione dell'ambiente |
|---|---|
| Kafka 3.2.3 | • 3 zookeeper – t2.small<br>• 3 server di broker – i3en.2xlarge<br>• 1 x Grafana – c5n.2xlarge<br>• 4 x produttore/consumatore — c5n.2xlargo * |
| Sistema operativo su tutti i nodi | RHEL8.6 |

| Componente della piattaforma | Configurazione dell'ambiente |
|---|---|
| AWS FSX per NetApp ONTAP | Multi-AZ con throughput di 4 GB/sec e 160000 IOPS |

#### Configurazione di NetApp FSX per NetApp ONTAP

1. Per il test iniziale, abbiamo creato un file system FSX per NetApp ONTAP con 2 TB di capacità e 40000 IOPS per un throughput 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 implementando FSX per NetApp ONTAP attraverso l'interfaccia CLI AWS. Sarà necessario personalizzare ulteriormente il comando nell'ambiente in base alle esigenze. FSX per NetApp ONTAP può inoltre essere implementato e gestito tramite la console AWS per un'esperienza di implementazione più semplice e ottimizzata con meno input dalla riga di comando.

   Documentazione in FSX per NetApp ONTAP, il numero massimo di IOPS ottenibili per un file system con throughput di 2 GB/sec nella nostra area di test (US-Est-1) è 80,000 iops. Il totale massimo di iops per un file system FSX per NetApp ONTAP è di 160,000 iops, che richiede un'implementazione di throughput di 4 GB/sec per ottenere il risultato che verrà dimostrato più avanti in questo documento.

   Per ulteriori informazioni sulle specifiche delle prestazioni di FSX per NetApp ONTAP, visita la documentazione di AWS FSX per NetApp 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 invece della chiave master AWS FSX predefinita utilizzata quando non viene specificata alcuna chiave KMS.

2. Durante la creazione del file system FSX per NetApp ONTAP, attendere che lo stato "Lifecycle" (ciclo di vita) passi a "AVAILABLE" (DISPONIBILE) nel ritorno JSON dopo aver descritto il 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 per SSH NetApp ONTAP con l'utente fsxadmin: Fsxadmin è l'account admin predefinito per FSX per i filesystem NetApp ONTAP al momento della creazione. La password per fsxadmin è la password che è stata configurata durante la prima creazione del file system nella console AWS o con l'interfaccia CLI AWS, come è stato completato nella fase 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 storage sul file system FSX per NetApp 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 macchina virtuale per lo storage (SVM) è un file server isolato con le proprie credenziali amministrative ed endpoint per l'amministrazione e l'accesso ai dati in FSX per i volumi NetApp ONTAP e fornisce FSX per il multi-tenancy NetApp ONTAP.

5. Una volta configurata la macchina virtuale di storage primaria, SSH nel nuovo file system FSX per NetApp ONTAP e creare volumi nella macchina virtuale di storage utilizzando il comando di esempio riportato di seguito. Analogamente, creiamo 6 volumi per questa convalida. In base alla nostra convalida, mantenere il costituente predefinito (8) o un numero inferiore di costituenti che forniranno 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, abbiamo bisogno di capacità aggiuntiva nei nostri volumi. Estendere le dimensioni del volume a 2 TB e montarlo 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 per NetApp ONTAP, è possibile eseguire il thin provisioning dei volumi. Nel nostro esempio, la capacità totale del volume esteso supera la capacità totale del file system, quindi sarà necessario estendere la capacità totale del file system per sbloccare la capacità aggiuntiva del volume sottoposto a provisioning, come illustrato nella fase successiva.

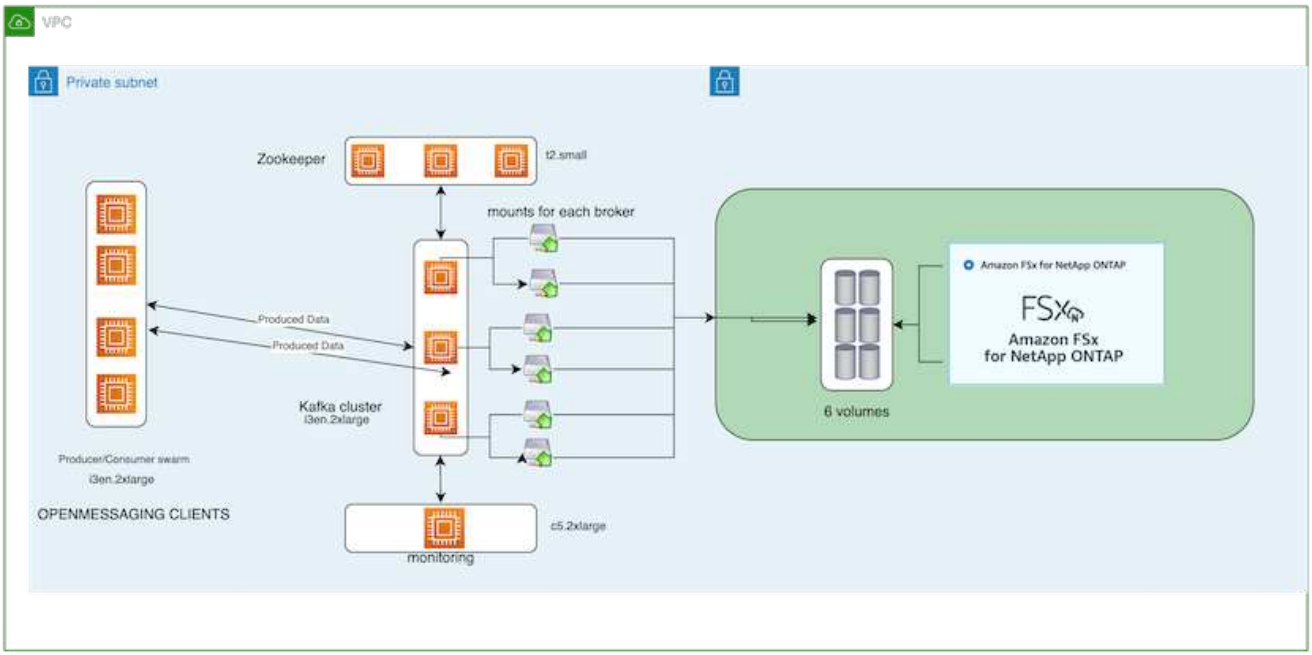
7. Inoltre, per ottenere maggiori performance e capacità, estendiamo la capacità di throughput FSX per NetApp ONTAP da 2 GB/sec a 4 GB/sec e IOPS a 160000 e la capacità a 5 TB

```
[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 per NetApp ONTAP sono montati con opzioni nconnect e predefinite nei broker Kafka

La seguente immagine mostra l'architettura finale di un cluster Kafka basato su FSX per NetApp ONTAP:




- Calcolo. Abbiamo utilizzato un cluster Kafka a tre nodi con un gruppo di zookeeper a tre nodi in esecuzione su server dedicati. Ciascun broker disponeva di sei punti di montaggio NFS su sei volumi nell'istanza FSX per NetApp ONTAP.
- Monitoraggio. Abbiamo utilizzato due nodi per una combinazione Prometheus-Grafana. Per la generazione dei carichi di lavoro, abbiamo utilizzato un cluster a tre nodi separato in grado di produrre e utilizzare questo cluster Kafka.
- Storage. Abbiamo utilizzato un FSX per NetApp ONTAP con sei volumi da 2 TB montati. Il volume è stato quindi esportato nel broker Kafka con un montaggio NFS. I volumi FSX per NetApp ONTAP sono 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 qui - https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

**Metodologia di test**

1. È stato eseguito il provisioning di un cluster Kafka in base alle specifiche descritte in precedenza utilizzando Terraform e ansible. Il terraform viene utilizzato per costruire l'infrastruttura utilizzando istanze AWS per il cluster Kafka e ansible crea il cluster Kafka su di essi.
2. È stato attivato un carico di lavoro OMB 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. È stato attivato un altro carico di lavoro con il driver di 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 performance di un'istanza di Kafka in esecuzione su NFS. La differenza tra i driver è la proprietà di scaricamento dei log.

Per un fattore di replica Kafka 1 e FSX per NetApp ONTAP:

- Throughput totale generato in modo coerente dal driver Sync: ~ 3218 Mbps e performance di picco in ~ 3652 Mbps.
- Throughput totale generato in modo coerente dal driver di throughput: ~ 3679 Mbps e performance di picco in ~ 3908 Mbps.

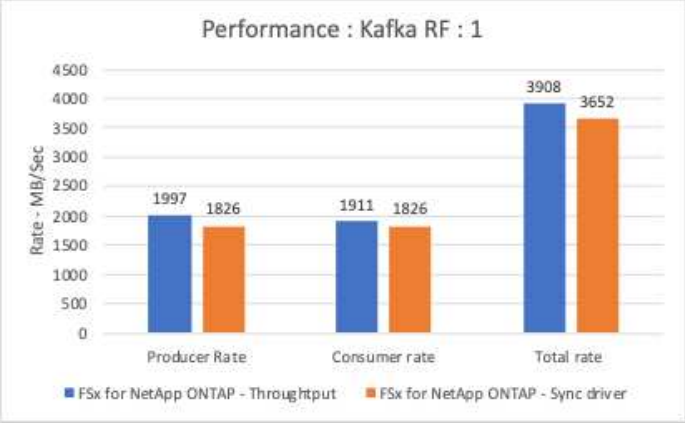
Per Kafka con fattore di replica 3 e FSX per NetApp ONTAP :

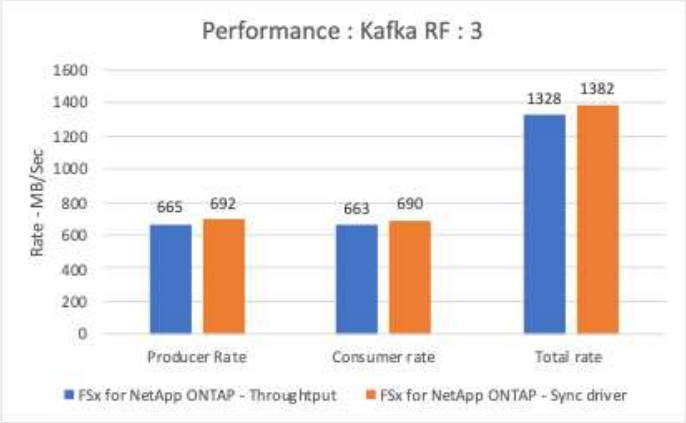
- Throughput totale generato in modo coerente dal driver Sync: ~ 1252 Mbps e performance di picco in ~ 1382 Mbps.
- Throughput totale generato in modo coerente dal driver di throughput: ~ 1218 Mbps e performance di picco in ~ 1328 Mbps.

Nel fattore 3 di replica di Kafka, l'operazione di lettura e scrittura è stata eseguita tre volte su FSX per NetApp ONTAP, nel fattore 1 di replica di Kafka, l'operazione di lettura e scrittura è una volta su FSX per NetApp ONTAP, quindi in entrambe le procedure di convalida, Siamo in grado di raggiungere il throughput massimo di 4 GB/sec.

Il driver Sync è in grado di generare un throughput coerente quando i log vengono trasferiti istantaneamente sul disco, mentre il driver di throughput genera burst di throughput quando i log vengono impegnati su disco in massa.

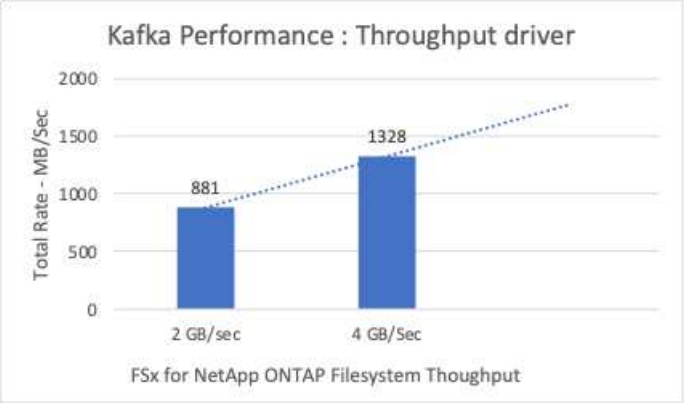
Questi numeri di throughput vengono generati per la configurazione AWS specificata. Per requisiti di performance più elevati, i tipi di istanze possono essere scalati e ottimizzati ulteriormente per ottenere numeri di throughput migliori. Il throughput totale o il tasso totale è la combinazione di un tasso di produttore e di consumo.

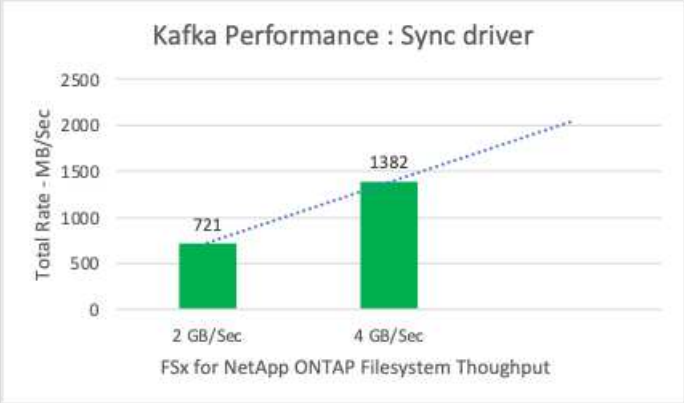


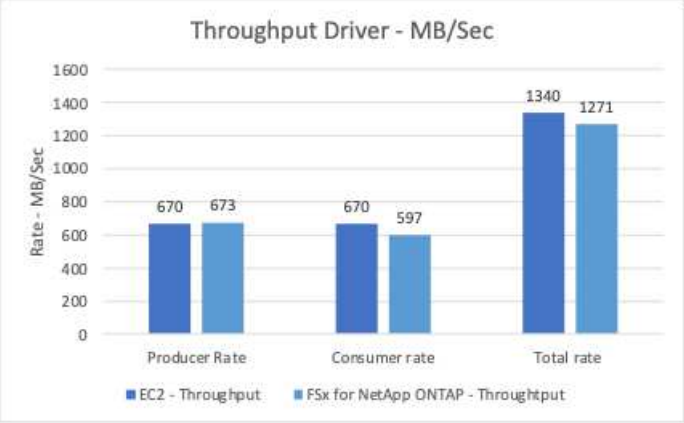

Il grafico riportato di seguito mostra le prestazioni FSX da 2 GB/sec per NetApp ONTAP e da 4 GB/sec per il fattore di replica Kafka 3. Il fattore di replica 3 esegue tre volte l'operazione di lettura e scrittura su FSX per lo

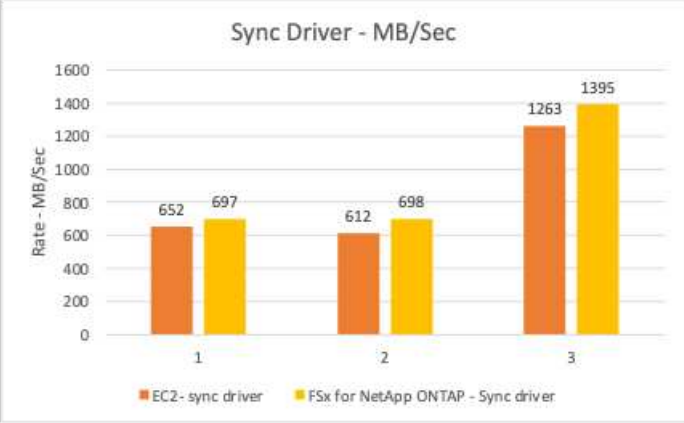
storage NetApp ONTAP. La velocità totale per il driver di throughput è di 881 MB/sec, che esegue operazioni di lettura e scrittura Kafka di circa 2.64 GB/sec sul file system FSX da 2 GB/sec per NetApp ONTAP, mentre la velocità totale per il driver di throughput è di 1328 MB/sec che esegue operazioni di lettura e scrittura kafka di circa 3.98 GB/sec. Le performance di Kafka sono lineari e scalabili in base al throughput FSX per NetApp ONTAP.




Il grafico seguente mostra le performance tra l'istanza EC2 e FSX per NetApp ONTAP (fattore di replica Kafka: 3)

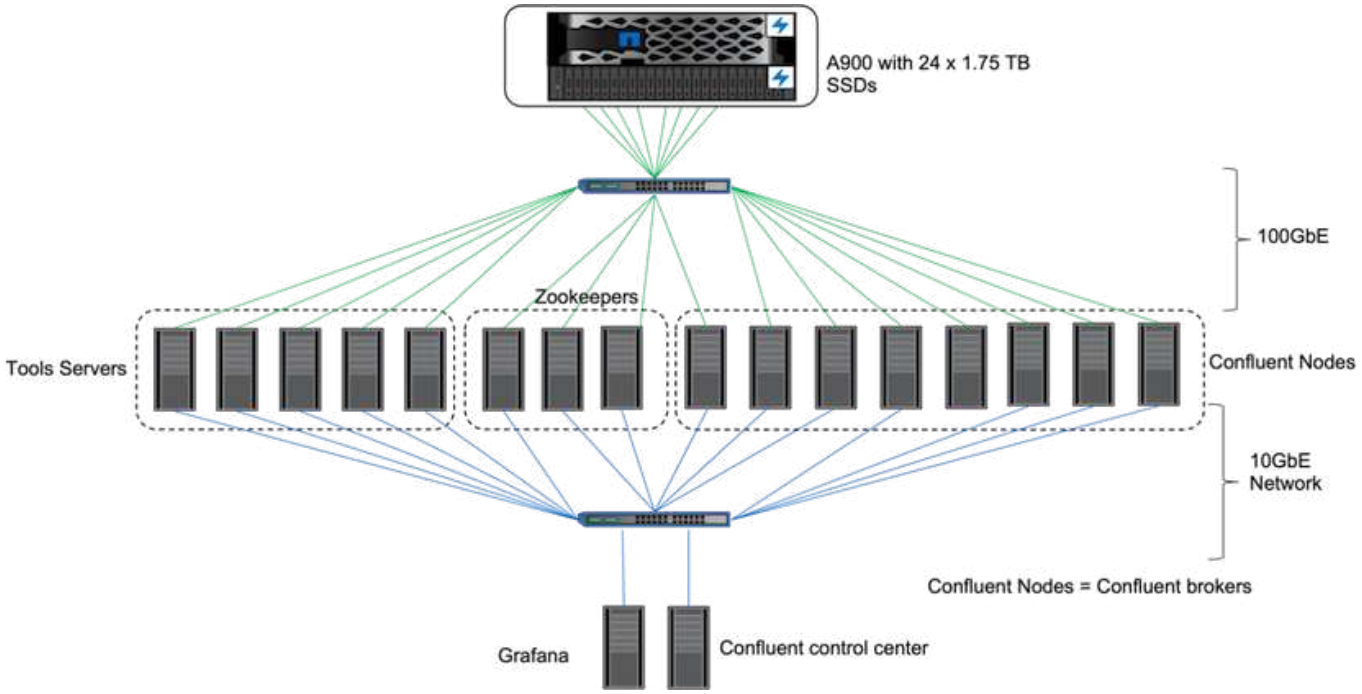



# Panoramica e validazione delle performance con AFF A900 on-premise

On-premise, abbiamo utilizzato il controller di storage NetApp AFF A900 con ONTAP 9.12.1RC1 per convalidare le performance e la scalabilità di un cluster Kafka. Abbiamo utilizzato lo stesso tested delle Best practice per lo storage a più livelli precedenti con ONTAP e AFF.

Abbiamo utilizzato Confluent Kafka 6.2.0 per valutare AFF A900. Il cluster include otto nodi di broker e tre nodi di zookeeper. Per il test delle performance, abbiamo utilizzato cinque nodi di lavoro OMB.




## Configurazione dello storage

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

## Tuning del client

Ogni client ha montato l'istanza di 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 di slot di sessione in ONTAP.

## Messa a punto del broker Kafka

Per massimizzare il throughput nel sistema sottoposto a test, abbiamo aumentato significativamente i parametri predefiniti per alcuni pool di thread chiave. Si consiglia di seguire le Best practice di Confluent Kafka per la maggior parte delle configurazioni. Questo tuning è stato utilizzato per massimizzare la concorrenza tra i/o in sospeso e storage. Questi parametri possono essere regolati in modo da corrispondere alle risorse di calcolo e agli attributi di storage del 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 workload

Abbiamo utilizzato le stesse configurazioni OMB utilizzate per i test cloud per il driver di throughput e la configurazione degli argomenti.

1. È stato eseguito il provisioning di un’istanza di FlexGroup utilizzando 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 attivato su 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 di throughput utilizzando la stessa configurazione del carico di lavoro di Cloud Volumes ONTAP. Vedere la sezione "Performance a stato stazionario" sotto. Il carico di lavoro utilizzava un fattore di replica di 3, il che significa che in NFS sono state mantenute tre copie di segmenti di log.

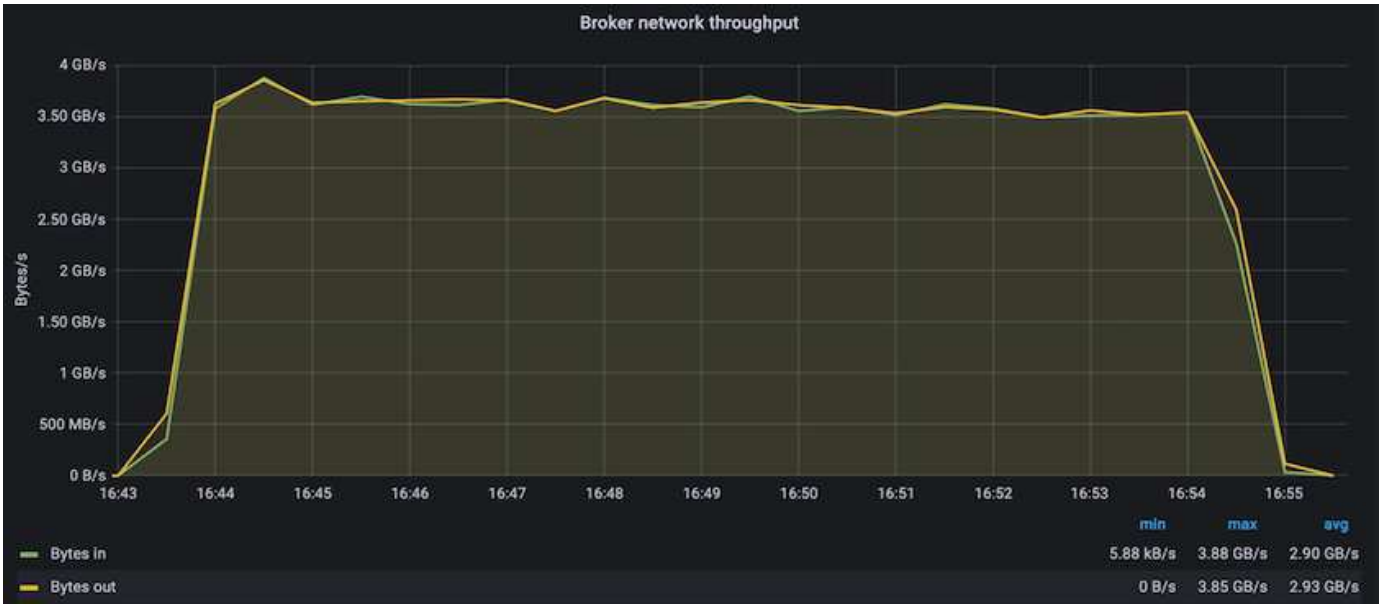
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. Infine, abbiamo completato le misurazioni utilizzando un backlog per misurare la capacità dei consumatori di recuperare i messaggi più recenti. OMB crea un backlog mettendo in pausa i consumatori durante l'inizio di una misurazione. Ciò produce tre fasi distinte: Creazione di backlog (traffico solo produttore), eliminazione dei backlog (una fase pesante per i consumatori in cui i consumatori si mettono al passo con gli eventi persi in un argomento) e lo stato stazionario. Vedere la sezione "analisi dei limiti dello storage" per ulteriori informazioni.

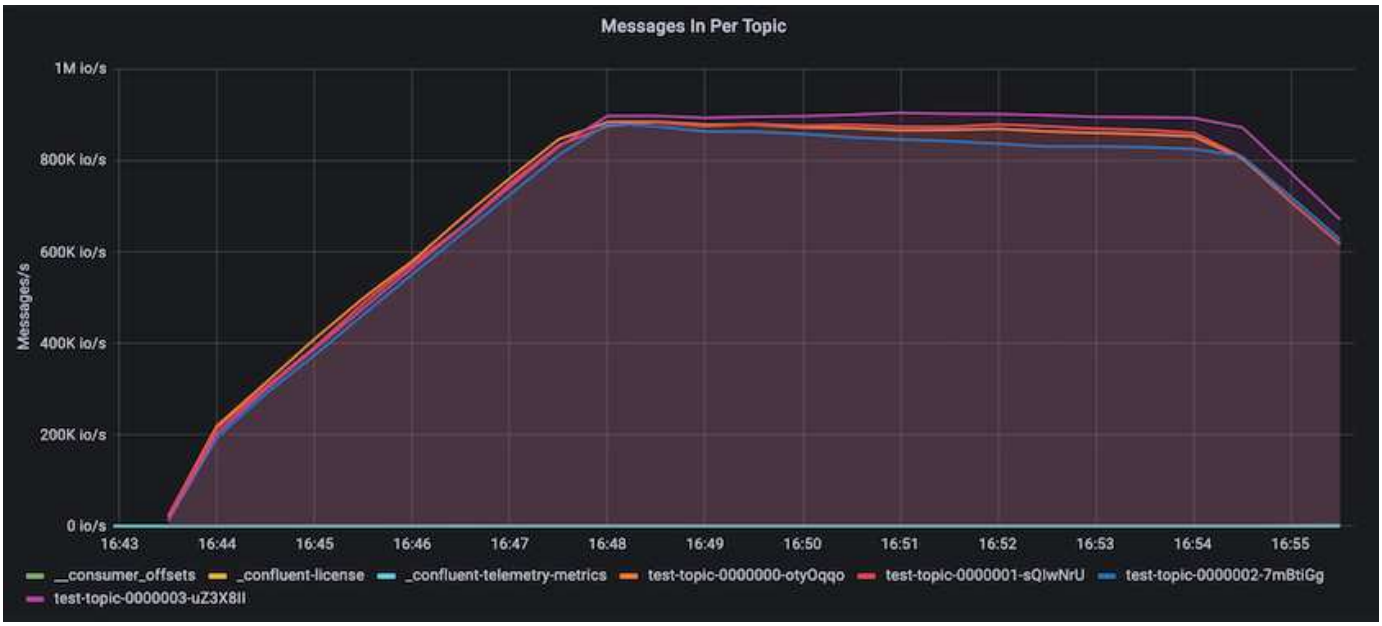
**Performance a stato stazionario**

Abbiamo valutato AFF A900 utilizzando il benchmark OpenMessaging per fornire un confronto simile a Cloud Volumes ONTAP in AWS e DAS in AWS. Tutti i valori delle performance rappresentano il throughput del cluster Kafka a livello di produttore e consumatore.

Le performance a stato stazionario con Confluent Kafka e AFF A900 hanno raggiunto un throughput medio di oltre 3,4 Gbps sia per i produttori che per i consumatori. Si tratta di oltre 3.4 milioni di messaggi nel cluster Kafka. Visualizzando il throughput sostenuto in byte al secondo per BrokerTopicMetrics, vediamo le eccellenti performance di stato stazionario e il traffico supportato da AFF A900.




Questo si allinea perfettamente con la visualizzazione dei messaggi inviati per argomento. Il seguente grafico fornisce una descrizione dettagliata per argomento. Nella configurazione testata abbiamo visto quasi 900.000 messaggi per argomento in quattro argomenti.

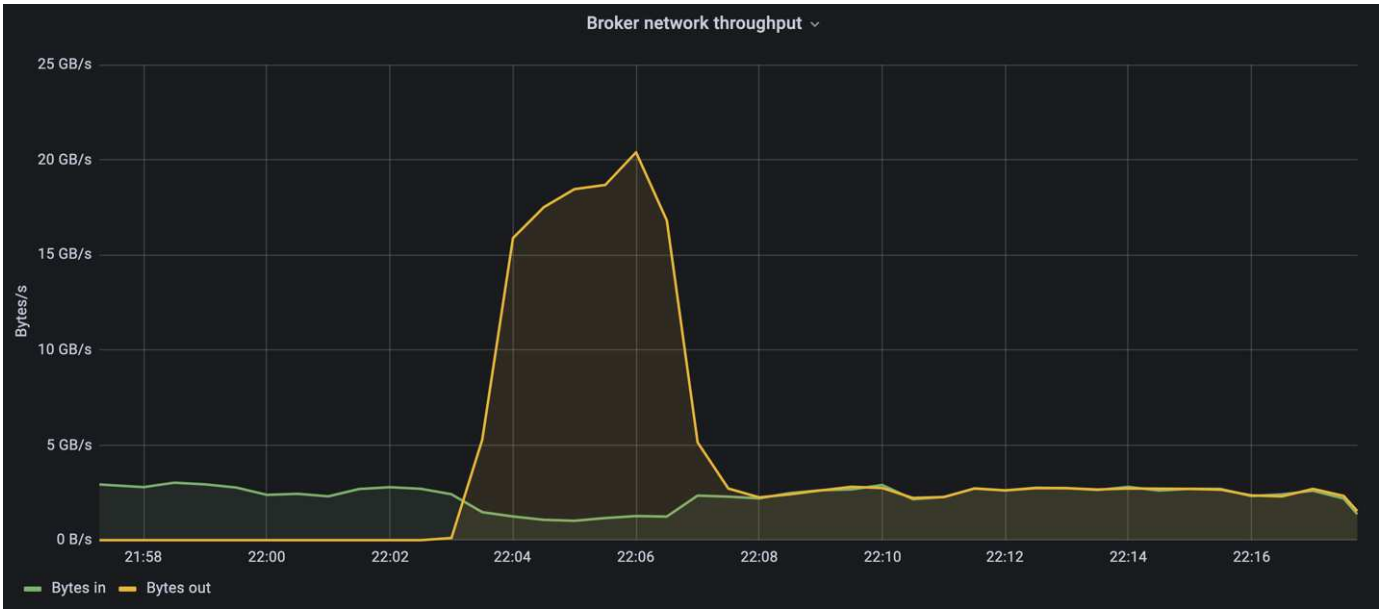



## Performance estreme e analisi dei limiti dello storage

Per AFF, abbiamo anche testato con OMB utilizzando la funzionalità di backlog. La funzionalità di backlog mette in pausa gli abbonamenti consumer mentre nel cluster Kafka viene creato un backlog di eventi. Durante questa fase, si verifica solo il traffico del produttore, che genera eventi che vengono impegnati nei registri. In questo modo si emulano più da vicino i flussi di lavoro di elaborazione batch o di analisi offline; in questi flussi di lavoro, le sottoscrizioni consumer vengono avviate e devono leggere i dati storici che sono già stati rimossi dalla cache del broker.

Per comprendere le limitazioni dello storage sul throughput consumer in questa configurazione, abbiamo misurato la fase solo produttore per capire quanto traffico di scrittura potrebbe assorbire l'A900. Vedere la sezione successiva "Guida al dimensionamento" per capire come sfruttare questi dati.

Durante la parte solo produttore di questa misurazione, abbiamo riscontrato un throughput elevato che ha spinto i limiti delle performance di A900 (quando le altre risorse di broker non erano sature e il traffico consumer e dei produttori).



Abbiamo aumentato le dimensioni del messaggio a 16.000 per questa misurazione per limitare le spese generali per messaggio e massimizzare il throughput dello storage ai punti di montaggio NFS.

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

Il cluster Confluent Kafka ha raggiunto un picco di throughput dei produttori 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 consumer è stato riavviato. Durante le misurazioni con il deflusso del backlog, abbiamo osservato un throughput dei consumatori di oltre 20 Gbps in tutti gli argomenti. Il throughput combinato per il volume NFS che memorizza i dati di log OMB si avvicinava a ~30 Gbps.

## Guida al dimensionamento

Amazon Web Services offre un "guida al dimensionamento" Per il dimensionamento e la scalabilità dei cluster Kafka.

Questo dimensionamento fornisce una formula utile per determinare i requisiti di throughput dello storage per il cluster Kafka:

Per un throughput aggregato prodotto nel cluster di tcluster con un fattore di replica di r, il throughput ricevuto dallo storage del broker è il seguente:

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

Questo può essere ulteriormente semplificato:

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

Questa formula consente di selezionare la piattaforma ONTAP appropriata per le tue esigenze di hot Tier Kafka.

La seguente tabella illustra il throughput previsto dal produttore per l'A900 con diversi fattori di replica:

| Fattore di replica | Throughput produttore (GPPS) |
|---|---|
| 3 (misurato) | 3.4 |
| 2 | 5.1 |
| 1 | 10.2 |

# Conclusione

La soluzione NetApp per il problema del ridenominazione sciocco offre una forma di storage 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 che siano più facili da migrare e eseguire il mirroring ai fini del disaster recovery e della protezione dei dati.
Abbiamo anche visto che NFS offre ulteriori vantaggi, come la riduzione dell'utilizzo della CPU e un tempo di recovery più rapido, un notevole miglioramento dell'efficienza dello storage e migliori performance grazie a NetApp ONTAP.

# Dove trovare ulteriori informazioni

Per ulteriori informazioni sulle informazioni descritte in questo documento, consultare i seguenti documenti e/o siti Web:

- Che cos'è Apache Kafka?

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

- Che cos'è un ridenominazione sciocco?

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

- ONATP viene letto per le applicazioni di streaming.

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

- Stupido: Rinominare il problema con Kafka.

  "https://sbg.technology/2018/07/10/kafka-nfs/"

- Documentazione sui prodotti NetApp

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

- Che cos'è NFS?

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

- Cos'è la riassegnazione delle partizioni Kafka?

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

- Che cos'è il benchmark OpenMessaging?

  "https://openmessaging.cloud/"

- Come migrare un broker Kafka?

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

- Come monitorate 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/