使用NetApp NFS 儲存的 Apache Kafka 工作負載

NetApp artificial intelligence solutions

NetApp
February 12, 2026

目錄

使用NetApp NFS 儲存的 Apache Kafka 工作負載

TR-4947：使用NetApp NFS 儲存的 Apache Kafka 工作負載 - 功能驗證與效能

Shantanu Chakole、Karthikeyan Nagalingam 和 Joe Scott， NetApp

Kafka 是一個分散式發布-訂閱訊息系統，具有強大的佇列，可以接受大量訊息資料。使用 Kafka，應用程式可以非常快速地向主題寫入和讀取資料。由於其容錯性和可擴展性，Kafka 經常被用在大數據領域，作為一種可靠的方式來快速提取和移動大量資料流。使用案例包括串流處理、網站活動追蹤、指標收集和監控、日誌聚合、即時分析等。

儘管 NFS 上的正常 Kafka 操作運作良好，但在 NFS 上執行的 Kafka 叢集調整大小或重新分割期間，愚蠢的重命名問題會導致應用程式崩潰。這是一個重大問題，因為必須調整 Kafka 叢集的大小或重新分區以實現負載平衡或維護目的。您可以找到更多詳細信息 "這裡"。

本文檔描述了以下主題：

- 愚蠢的重命名問題和解決方案驗證
- 降低 CPU 使用率以減少 I/O 等待時間
- 更快的 Kafka 代理程式恢復時間
- 雲端和本地的效能

為什麼使用 NFS 儲存來儲存 Kafka 工作負載？

生產應用程式中的 Kafka 工作負載可以在應用程式之間傳輸大量資料。這些資料保存並儲存在 Kafka 叢集中的 Kafka 代理節點中。 Kafka 也以可用性和並行性而聞名，它透過將主題分成多個分區，然後在整個叢集中複製這些分區來實現。這最終意味著流經 Kafka 群集的大量資料通常會倍增。隨著代理數量的變化，NFS 可以非常快速且輕鬆地重新平衡資料。對於大型環境，當代理數量發生變化時，跨 DAS 重新平衡資料非常耗時，並且在大多數 Kafka 環境中，代理數量經常會發生變化。

其他好處包括：

- *到期。 * NFS 是一種成熟的協議，這意味著它的實現、保護和使用的大多數方面都已被很好地理解。
- *打開。 * NFS 是一個開放協議，其持續發展在互聯網規範中被記錄為一個自由開放的網路協議。
- *具有成本效益。 * NFS 是一種低成本的網路檔案共用解決方案，由於它使用現有的網路基礎設施，因此易於設定。
- *集中管理。 * NFS 的集中管理減少了單一使用者係統上新增軟體和磁碟空間的需求。
- *分散式。 * NFS 可用作分散式檔案系統，減少可移動媒體儲存設備的需求。

為什麼選擇NetApp來處理 Kafka 工作負載？

NetApp NFS 實施被認為是該協議的黃金標準，並應用於無數企業 NAS 環境。除了NetApp的信譽之外，它還提供以下優勢：

- 可靠性和效率
- 可擴充性和效能
- 高可用性（ NetApp ONTAP叢集中的 HA 合作夥伴）
- 資料保護
  - 災難復原（NetApp SnapMirror）。 *您的網站癱瘓了，或者您想從另一個網站開始並從上次中斷的地方繼續。
  - 儲存系統的可管理性（使用NetApp OnCommand進行管理）。
  - *負載平衡。 *此叢集可讓您從託管在不同節點上的資料 LIF 存取不同的磁碟區。
  - *無中斷運作。 * LIF 或磁碟區移動對於 NFS 用戶端來說是透明的。

NetApp針對 NFS 到 Kafka 工作負載重新命名問題的解決方案

Kafka 的建置假設底層檔案系統符合 POSIX 標準：例如 XFS 或 Ext4。當應用程式仍在使用檔案時，Kafka 資源重新平衡會刪除這些檔案。符合 POSIX 標準的檔案系統允許取消連結繼續進行。但是，只有在對該文件的所有引用都消失後，它才會刪除該文件。如果底層檔案系統是網路連線的，那麼 NFS 用戶端會攔截取消連結呼叫並管理工作流程。由於正在取消連結的檔案上有待開啟的操作，因此 NFS 用戶端會向 NFS 伺服器發送重新命名請求，並在最後一次關閉取消連結的檔案時，會對重新命名的檔案發出刪除操作。此行為通常稱為 NFS 愚蠢重命名，由 NFS 用戶端精心策劃。

由於這種行為，任何使用 NFSv3 伺服器儲存的 Kafka 代理都會遇到問題。但是，NFSv4.x 協定具有解決此問題的功能，它允許伺服器負責開啟的未連結的檔案。支援此可選功能的 NFS 伺服器在開啟檔案時將所有權能力傳達給 NFS 用戶端。當有待處理的開啟操作時，NFS 用戶端將停止取消連結管理，並允許伺服器管理流程。儘管 NFSv4 規範提供了實作指南，但到目前為止，還沒有任何已知的 NFS 伺服器實作支援此選用功能。

為了解決這個愚蠢的重命名問題，需要對 NFS 伺服器和 NFS 用戶端進行以下更改：

- *對 NFS 用戶端 (Linux) 的變更。 *在開啟檔案時，NFS 伺服器會回應一個標誌，表示有能力處理開啟檔案的取消連結。 NFS 用戶端的變更允許 NFS 伺服器在存在標誌的情況下處理取消連結。 NetApp已根據這些變更更新了開源 Linux NFS 用戶端。更新後的 NFS 用戶端現已在 RHEL8.7 和 RHEL9.1 中普遍可用。
- *對 NFS 伺服器的變更。 * NFS 伺服器追蹤開啟的情況。現在，伺服器管理對現有開啟檔案的取消連結以符合 POSIX 語義。當最後一個開啟的檔案關閉時，NFS 伺服器將啟動檔案的實際刪除，從而避免愚蠢的重命名過程。 ONTAP NFS 伺服器在其最新版本ONTAP 9.12.1 中實作了此功能。

透過對 NFS 用戶端和伺服器進行上述更改，Kafka 可以安全地獲得網路附加 NFS 儲存的所有好處。

功能驗證 - 愚蠢的重命名修復

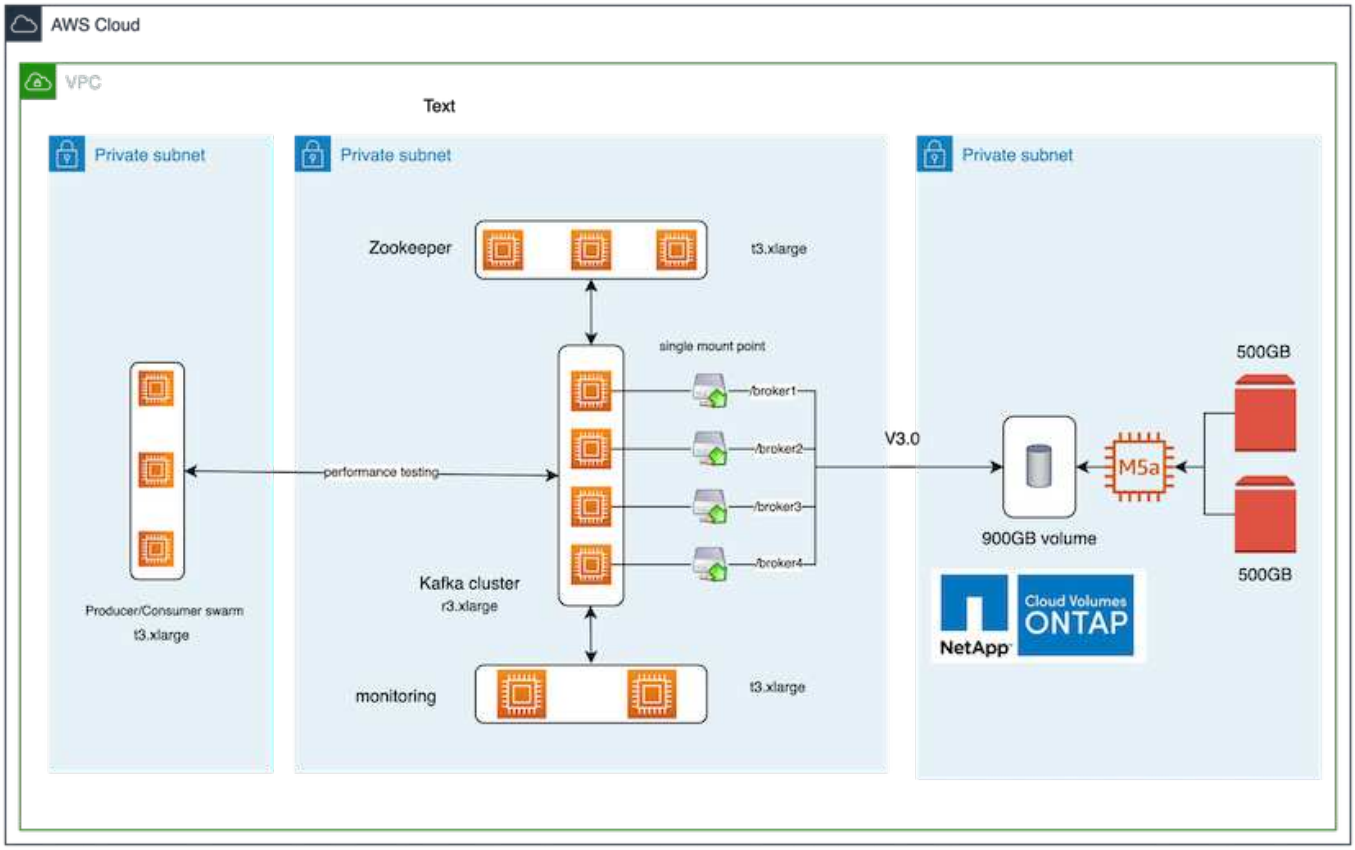
對於功能驗證，我們表明，使用 NFSv3 掛載儲存的 Kafka 叢集無法執行分區重新分配等 Kafka 操作，而使用修復程式掛載在 NFSv4 上的另一個叢集可以執行相同的操作而不會出現任何中斷。

驗證設定

該設定在 AWS 上運行。下表顯示了用於驗證的不同平台組件和環境配置。

平台組件	環境配置
Confluent 平台版本 7.2.1	• 3 個動物園管理員 – t3.xlarge • 4 個代理伺服器 – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x 控制中心 – t3.xlarge • 3 x 生產者/消費者
所有節點上的作業系統	RHEL8.7或更高版本
NetApp Cloud Volumes ONTAP實例	單一節點實例 – M5.2xLarge

下圖顯示了該解決方案的架構配置。



建築流程

- *計算。 *我們使用了四節點 Kafka 集群，並在專用伺服器上運行三節點 zookeeper 集合。
- *監控*我們使用兩個節點來實現 Prometheus-Grafana 組合。
- *工作量*為了產生工作負載，我們使用了一個單獨的三節點集群，該集群可以從該 Kafka 集群中生產和消費。

- *貯存。 *我們使用了單節點NetApp Cloud Volumes ONTAP實例，該實例連接了兩個 500GB GP2 AWS-EBS 磁碟區。然後，這些磁碟區透過 LIF 作為單一 NFSv4.1 磁碟區公開給 Kafka 叢集。

所有伺服器都選擇了 Kafka 的預設屬性。對動物園管理員群也做了同樣的事情。

測試方法

1. 更新 `-is-preserve-unlink-enabled true`到kafka卷，如下：

```
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. 創建了兩個類似的 Kafka 集群，但有以下區別：
   - *集群 1.*執行生產就緒ONTAP版本 9.12.1 的後端 NFS v4.1 伺服器由NetApp CVO 執行個體所託管。代理程式上安裝了 RHEL 8.7/RHEL 9.1。
   - *集群 2.*後端 NFS 伺服器是手動建立的通用 Linux NFSv3 伺服器。
3. 在兩個 Kafka 叢集上都建立了一個示範主題。

   集群 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:
```

   集群 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:
```

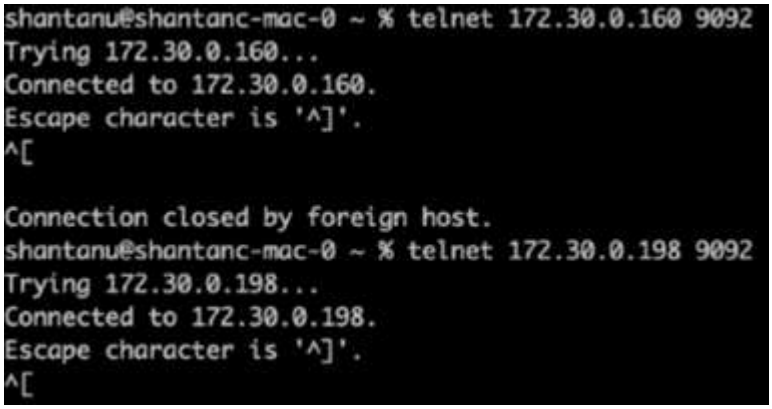
4. 資料被載入到這兩個叢集新建立的主題。這是使用預設 Kafka 包中的生產者效能測試工具包完成的：

```
./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. 使用 telnet 對每個叢集的 broker-1 執行健康檢查：

   - 遠端登入 `172.30.0.160 9092`
   - 遠端登入 `172.30.0.198 9092`

   下圖顯示了兩個集群上的代理的成功健康檢查：



6. 為了觸發導致使用 NFSv3 儲存磁碟區的 Kafka 叢集崩潰的故障條件，我們在兩個叢集上啟動了分區重新分配程序。分區重新分配是使用 `kafka-reassign-partitions.sh`。具體過程如下：

   a. 為了重新指派 Kafka 叢集中某個主題的分區，我們產生了建議的重新指派組態 JSON（對兩個叢集都執行了此操作）。

   ```
   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. 產生的重新分配 JSON 隨後保存在 `/tmp/reassignment- file.json`。

   c. 實際的分區重新分配過程由以下命令觸發：

   ```
   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. 重新分配完成後幾分鐘，對代理進行的另一次健康檢查顯示，使用 NFSv3 存儲卷的集群遇到了一個愚蠢的重命名問題並崩潰了，而使用已修復的NetApp ONTAP NFSv4.1 存儲卷的集群 1 繼續運行而沒有任何中斷。

```
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
```

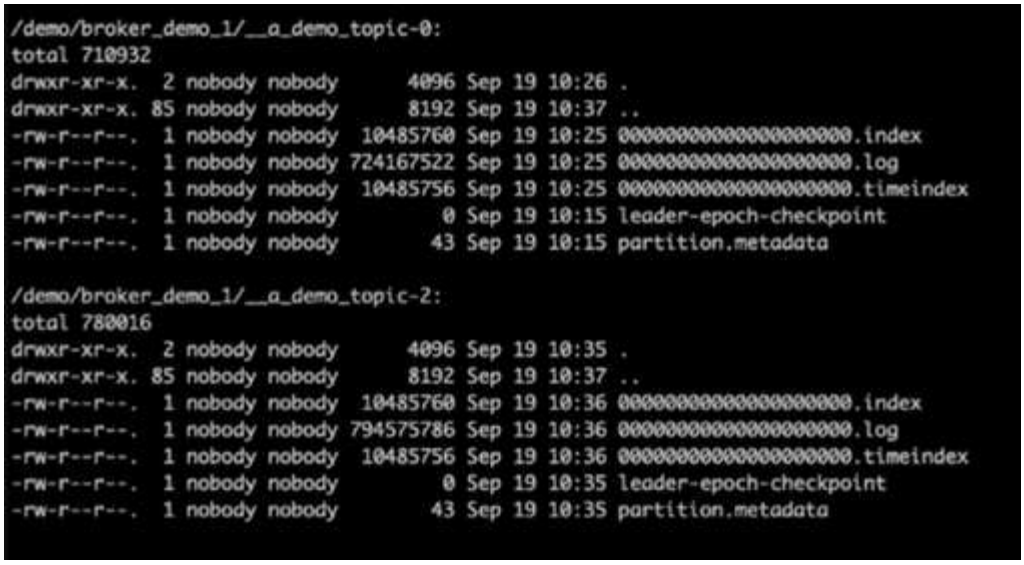
- Cluster1-Broker-1 處於活動狀態。
- Cluster2-broker-1 已死亡。

8. 檢查 Kafka 日誌目錄後，很明顯，使用已修復的NetApp ONTAP NFSv4.1 儲存磁碟區的叢集 1 具有乾淨的分割區分配，而使用通用 NFSv3 儲存的叢集 2 由於愚蠢的重命名問題而沒有，從而導致崩潰。下圖顯示了叢集 2 的分區重新平衡，這導致了 NFSv3 儲存上出現了一個愚蠢的重命名問題。

```
/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
```

下圖顯示了使用NetApp NFSv4.1 儲存對叢集 1 進行乾淨的分區重新平衡。



為什麼選擇NetApp NFS 來支援 Kafka 工作負載？

既然有了針對 Kafka 的 NFS 儲存中愚蠢重命名問題的解決方案，您就可以建立利用NetApp ONTAP儲存來處理 Kafka 工作負載的強大部署。這不僅顯著降低了營運開銷，還為您的 Kafka 叢集帶來以下好處：

- *降低 Kafka 代理程式的 CPU 使用率。 *使用分解的NetApp ONTAP儲存將磁碟 I/O 操作與代理程式分離，從而減少其 CPU 佔用。
- *更快的經紀人恢復時間。 *由於分解的NetApp ONTAP儲存在 Kafka 代理節點之間共享，因此與傳統的 Kafka 部署相比，新的運算實例可以在很短的時間內隨時替換損壞的代理，而無需重建資料。
- *儲存效率。 *由於應用程式的儲存層現在是透過NetApp ONTAP進行配置的，因此客戶可以利用ONTAP帶來的所有儲存效率優勢，例如線內資料壓縮、重複資料刪除和壓縮。

這些好處在本節我們將詳細討論的測試案例中得到了測試和驗證。

降低 **Kafka** 代理的 **CPU** 使用率

我們發現，當我們在兩個技術規格相同但儲存技術不同的獨立 Kafka 叢集上運行類似的工作負載時，整體 CPU 使用率低於其 DAS 對應叢集。當 Kafka 叢集使用ONTAP儲存時，不僅整體 CPU 使用率較低，而且 CPU 使用率的增加比基於 DAS 的 Kafka 叢集表現出更平緩的梯度。

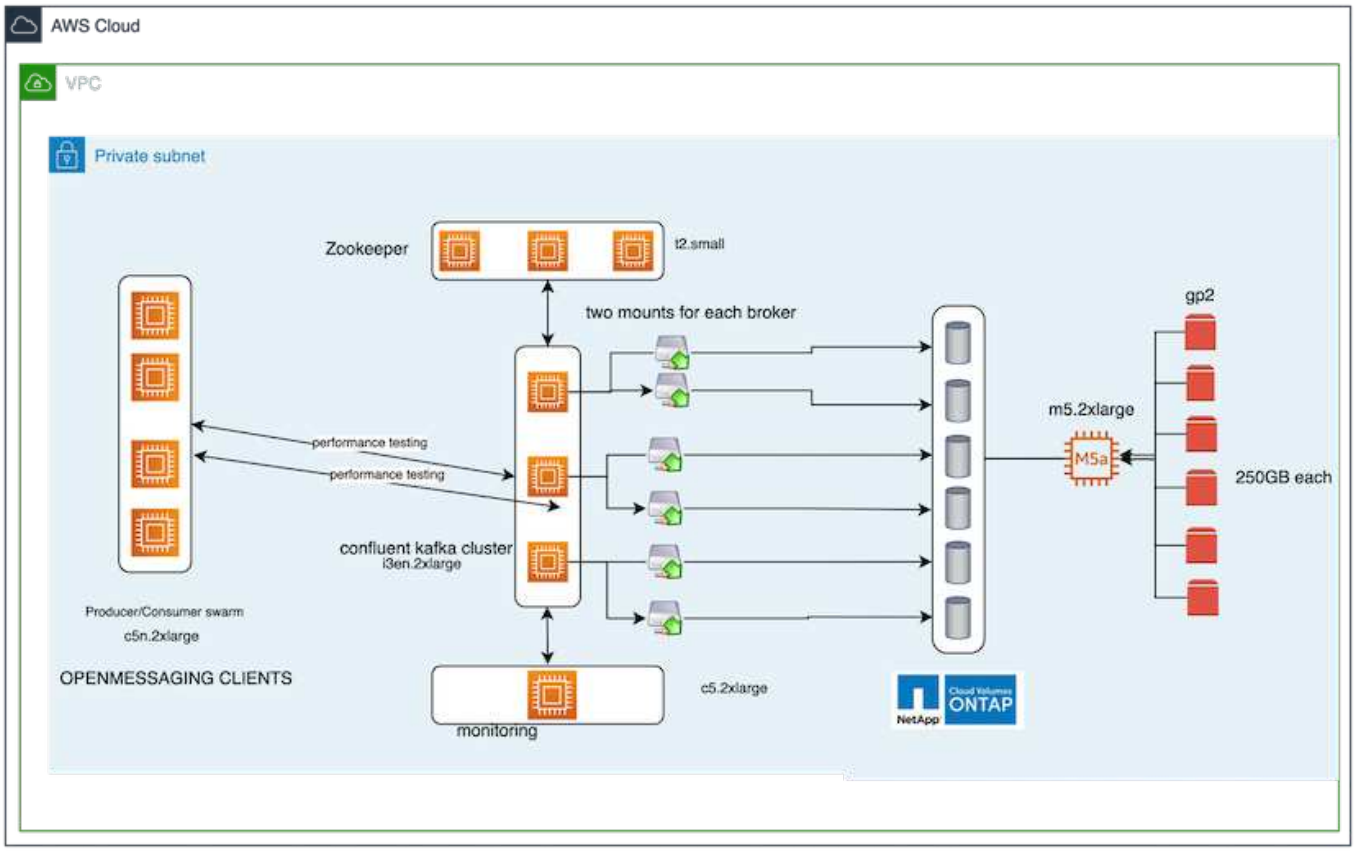
建築設置

下表顯示了用於演示降低 CPU 使用率的環境配置。

平台組件	環境配置
Kafka 3.2.3 基準測試工具：OpenMessaging	• 3 位動物園管理員 – t2.small • 3 個代理伺服器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生產者/消費者 - c5n.2xlarge
所有節點上的作業系統	RHEL 8.7 或更高版本
NetApp Cloud Volumes ONTAP實例	單一節點實例 – M5.2xLarge

基準測試工具

本測試案例中使用的基準測試工具是 "開放訊息傳遞"框架。 OpenMessaging 與供應商無關且與語言無關；它為金融、電子商務、物聯網和大數據提供產業指南；並有助於開發跨異質系統和平台的訊息傳遞和串流應用程式。下圖描述了 OpenMessaging 用戶端與 Kafka 叢集的交互作用。



- *計算。 *我們使用了三節點 Kafka 集群，並在專用伺服器上運行三節點 zookeeper 集合。每個代理程式透過專用 LIF 擁有兩個 NFSv4.1 掛載點，指向NetApp CVO 實例上的單一磁碟區。
- *監控*我們使用兩個節點來實現 Prometheus-Grafana 組合。為了產生工作負載，我們有一個單獨的三節點集群，可以從該 Kafka 集群中生產和消費。
- *貯存。 *我們使用了單節點NetApp Cloud Volumes ONTAP實例，該實例上安裝了六個 250GB GP2 AWS-EBS 磁碟區。然後，這些磁碟區透過專用 LIF 作為六個 NFSv4.1 磁碟區公開給 Kafka 叢集。
- *配置。 *此測試案例中的兩個可設定元素是 Kafka 代理程式和 OpenMessaging 工作負載。

。*經紀人配置。 *為 Kafka 代理選擇了以下規格。我們對所有測量使用了 3 的重複因子，如下所示。

```
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 基準 (OMB) 工作負載配置。 *提供了以下規格。我們指定了目標生產率，如下所示。

```
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
```

測試方法

1. 創建了兩個類似的集群，每個集群都有自己的一組基準集群群。
    - *集群 1.*基於NFS的Kafka叢集。
    - *集群 2.*基於DAS的Kafka叢集。
2. 使用 OpenMessaging 指令，每個叢集上都會觸發類似的工作負載。

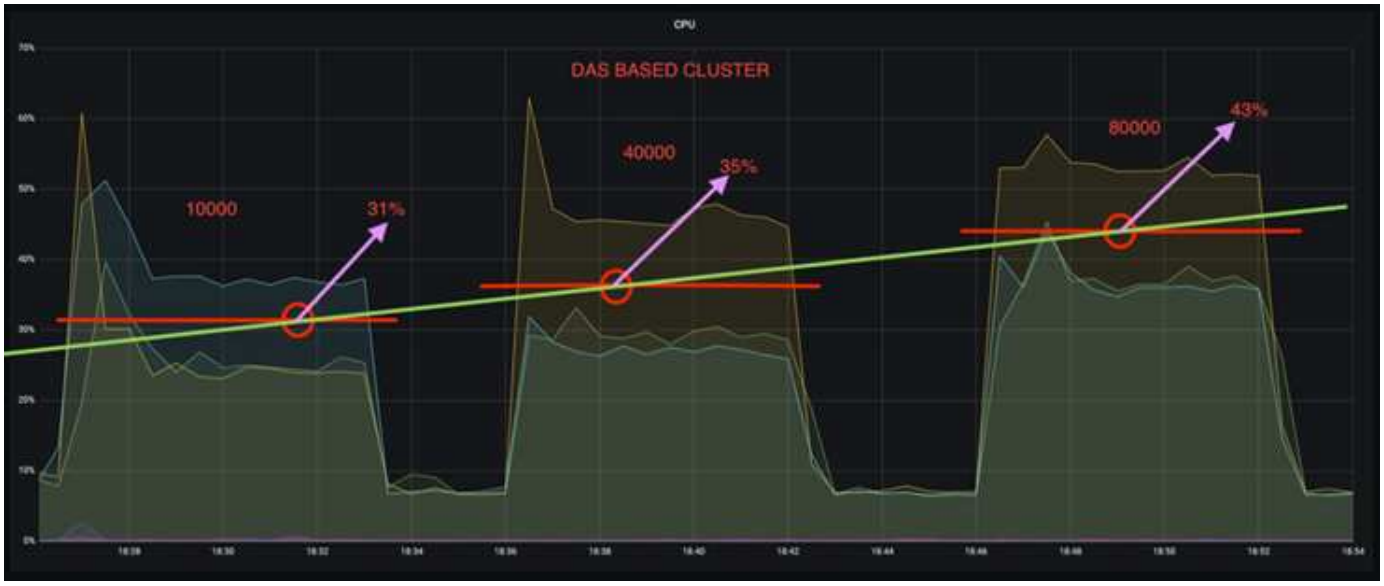
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

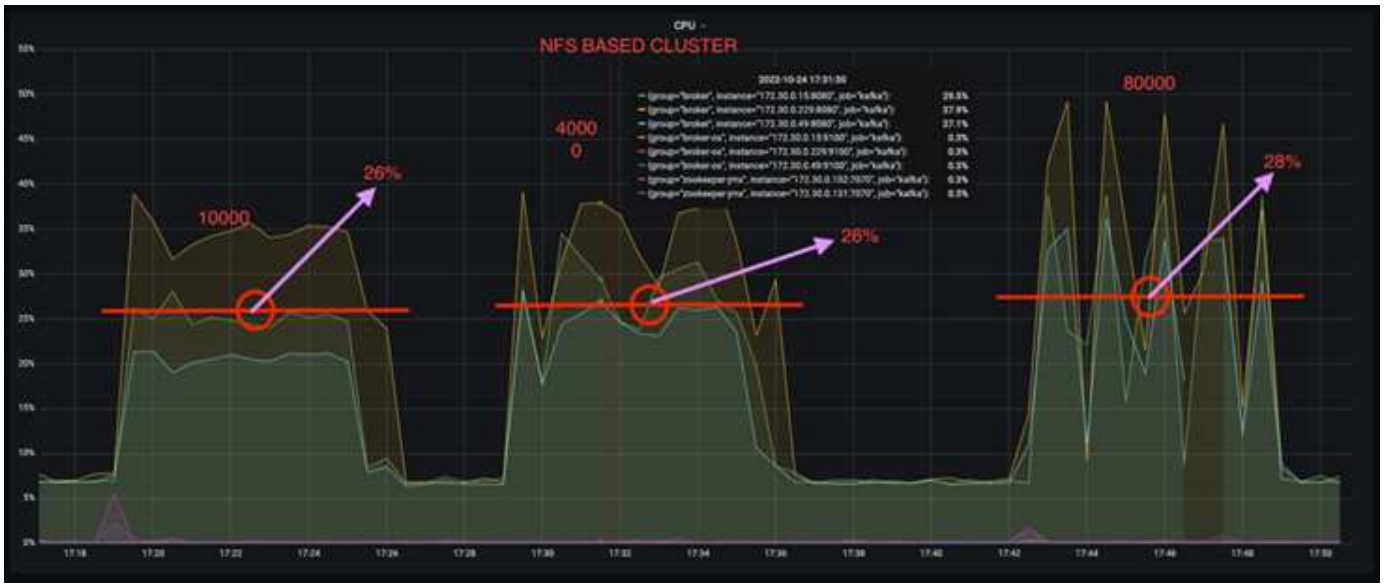
3. 生產率配置在四次迭代中增加，並使用 Grafana 記錄 CPU 使用率。生產力設定為以下水準：
   - 10,000
   - 40,000
   - 80,000
   - 100,000

觀察

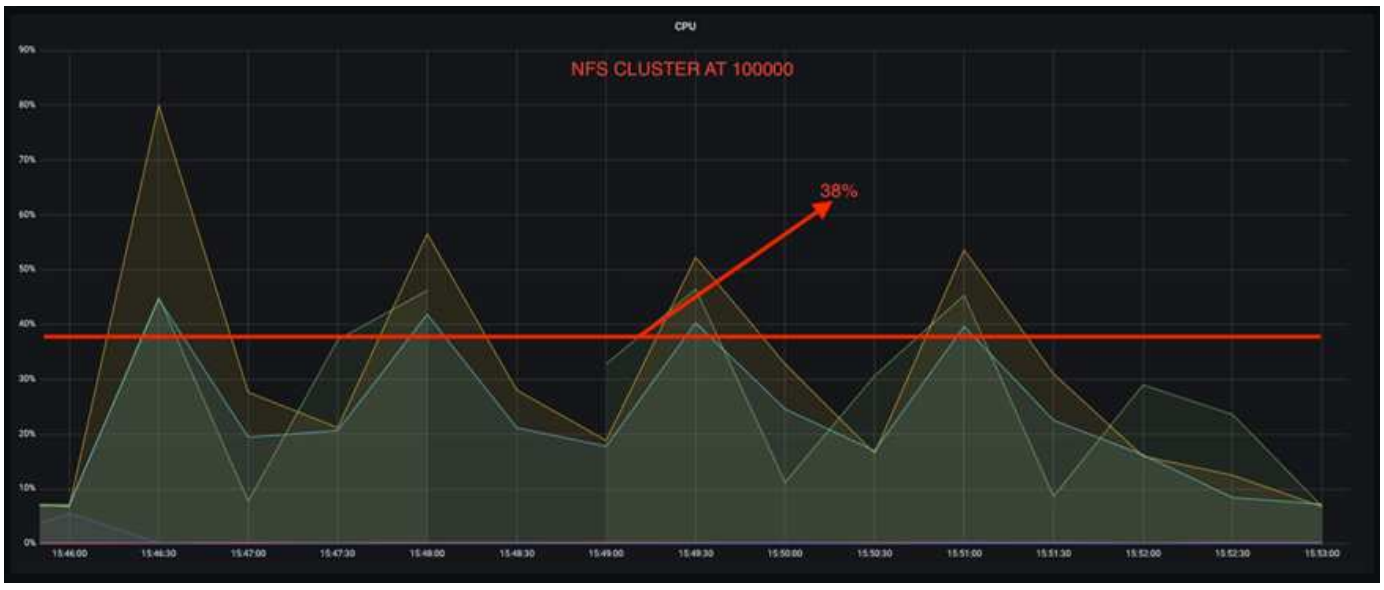
將NetApp NFS 儲存與 Kafka 結合使用有兩個主要好處：

- *您可以將 CPU 使用率降低近三分之一。 *與 DAS SSD 相比，NFS 在類似工作負載下的整體 CPU 使用率較低；節省範圍從較低生產率的 5% 到較高生產率的 32%。
- *在較高的生產率下，CPU 使用率漂移減少了三倍。 *如預期的那樣，隨著生產力的提高，CPU 使用率呈上升趨勢。然而，使用 DAS 的 Kafka 代理的 CPU 使用率從較低生產率時的 31% 上升到較高生產率時的 70%，增幅為 39%。然而，有了 NFS 儲存後端，CPU 使用率從 26% 上升到 38%，增加了 12%。





此外，在 100,000 則訊息時，DAS 顯示的 CPU 使用率比 NFS 叢集更高。



更快的經紀人恢復

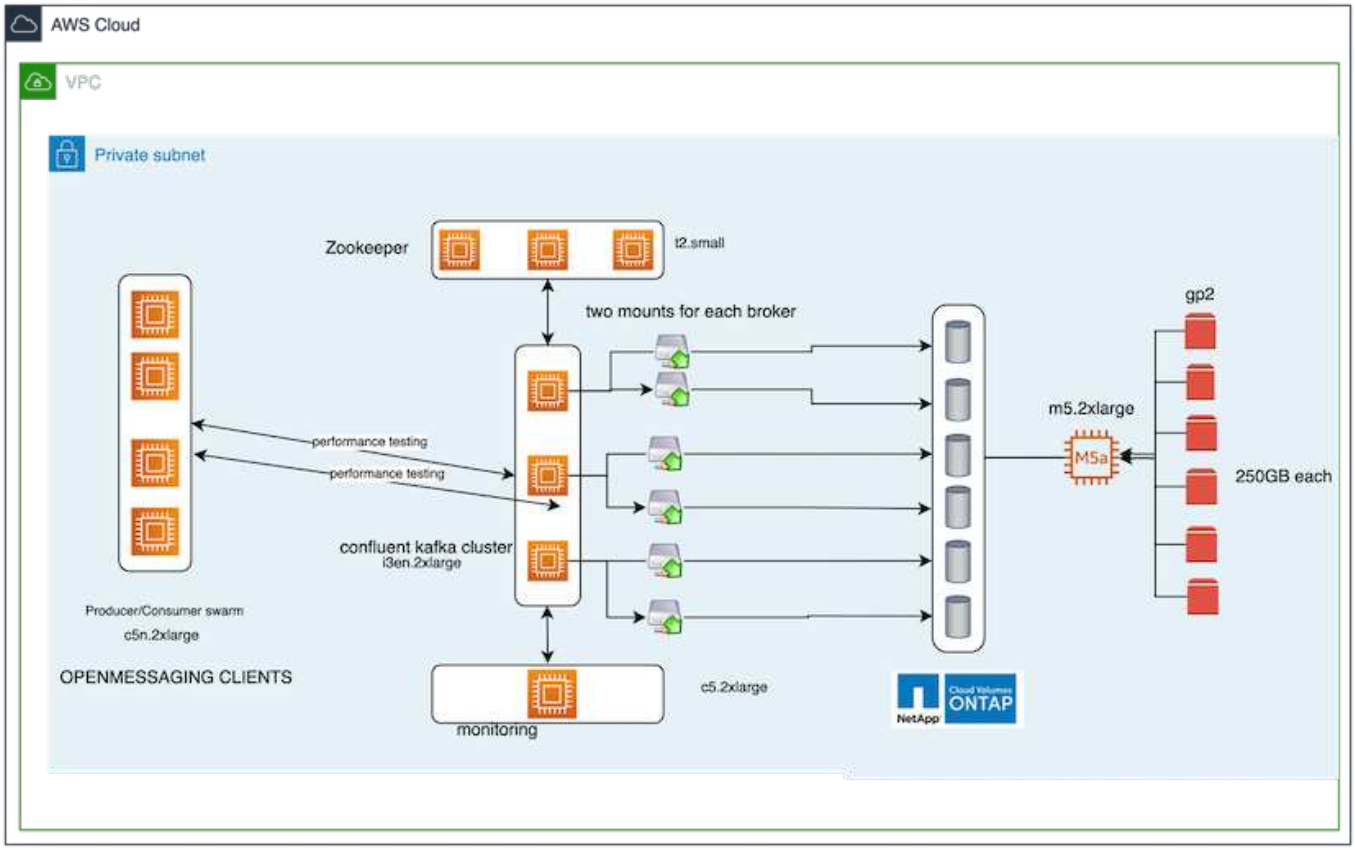
我們發現，當 Kafka 代理程式使用共享NetApp NFS 儲存時，復原速度更快。當 Kafka 群集中某個 Broker 崩潰時，可以用具有相同 Broker ID 的健康 Broker 取代該 Broker。在執行此測試用例時，我們發現，對於基於 DAS 的 Kafka 集群，集群會在新添加的健康 Broker 上重建數據，這非常耗時。對於基於NetApp NFS 的 Kafka 集群，替換代理將繼續從先前的日誌目錄讀取數據，並且恢復速度更快。

建築設置

下表展示了使用NAS的Kafka叢集的環境配置。

平台組件	環境配置
卡夫卡 3.2.3	• 3 位動物園管理員 – t2.small • 3 個代理伺服器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生產者/消費者 - c5n.2xlarge • 1 x 備份 Kafka 節點 – i3en.2xlarge
所有節點上的作業系統	RHEL8.7 或更高版本
NetApp Cloud Volumes ONTAP實例	單一節點實例 – M5.2xLarge

下圖是基於NAS的Kafka叢集架構圖。



- *計算。 *一個三節點 Kafka 集群，帶有一個三節點 zookeeper 集合，在專用伺服器上運行。每個代理程式透過專用 LIF 擁有兩個指向NetApp CVO 實例上的單一磁碟區的 NFS 掛載點。
- 監控 Prometheus-Grafana 組合的兩個節點。為了產生工作負載，我們使用一個單獨的三節點集群，該集群可以為該 Kafka 集群生產和消費。
- *貯存。 *單節點NetApp Cloud Volumes ONTAP實例，實例上安裝了六個 250GB GP2 AWS-EBS 磁碟區。然後，這些磁碟區透過專用 LIF 作為六個 NFS 磁碟區公開給 Kafka 叢集。
- *經紀人配置。 *此測試案例中一個可設定元素是 Kafka 代理。為 Kafka 代理選擇了以下規格。這 `replica.lag.time.mx.ms`設定為較高的值，因為這決定了特定節點從 ISR 清單中取出的速度。當您在壞節點和健康節點之間切換時，您不希望該代理 ID 被排除在 ISR 清單中。

```
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
```

測試方法

1. 創建了兩個類似的集群：
   - 基於 EC2 的匯合集群。
   - 基於NetApp NFS 的匯合叢集。
2. 建立了一個備用 Kafka 節點，其配置與原始 Kafka 叢集中的節點相同。
3. 在每個叢集上，都建立了一個範例主題，並且在每個代理程式上填入了大約 110GB 的資料。
   - *基於 EC2 的集群。 * Kafka 代理程式資料目錄對應到 `/mnt/data-2`（下圖中 cluster1 的 Broker-1[左側終端]）。
   - *基於NetApp NFS 的叢集。 * Kafka 代理程式資料目錄安裝在 NFS 點上 `/mnt/data`（下圖中 cluster2 的 Broker-1【右側終端】）。

4. 在每個叢集中，Broker-1 被終止以觸發失敗的代理恢復過程。

5. 代理終止後，代理 IP 位址被指派作為備用代理的輔助 IP。這是必要的，因為 Kafka 叢集中的代理程式以以下方式標識：
   - *IP 位址。 *透過將發生故障的代理 IP 重新指派給備用代理來進行分配。
   - *經紀人 ID。 *這是在備用代理程式中配置的 `server.properties`。
6. 分配 IP 後，備用代理程式上啟動了 Kafka 服務。
7. 過了一會兒，拉取伺服器日誌來檢查在叢集中的替換節點上建立資料所花費的時間。

觀察

Kafka 代理的恢復速度幾乎提高了 9 倍。與在 Kafka 叢集中使用 DAS SSD 相比，使用NetApp NFS 共用儲存時恢復故障代理節點所需的時間明顯更快。對於 1TB 的主題數據，基於 DAS 的叢集的恢復時間為 48 分鐘，而基於NetApp-NFS 的 Kafka 叢集的恢復時間則不到 5 分鐘。

我們觀察到基於 EC2 的叢集花費 10 分鐘在新代理節點上重建 110GB 數據，而基於 NFS 的叢集在 3 分鐘內完成復原。我們也在日誌中觀察到，EC2 分區的消費者偏移量為 0，而在 NFS 叢集上，消費者偏移量是從前一個代理程式取得的。

```
[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$)
[2022-10-31 08:55:55,170] INFO [LogLoader partition=test-topic-qbVsEZg-
0000-8, dir=/mnt/data-1] Loading producer state till offset 0 with message
format version 2 (kafka.log.UnifiedLog$)
```

基於**DAS**的集群

1. 備份節點於 08:55:53,730 啟動。

```
[2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
[2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (org
[2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
[2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.30
[2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. 資料重建過程於 09:05:24,860 結束。處理 110GB 的資料大約需要 10 分鐘。

```
[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)
```

基於**NFS**的集群

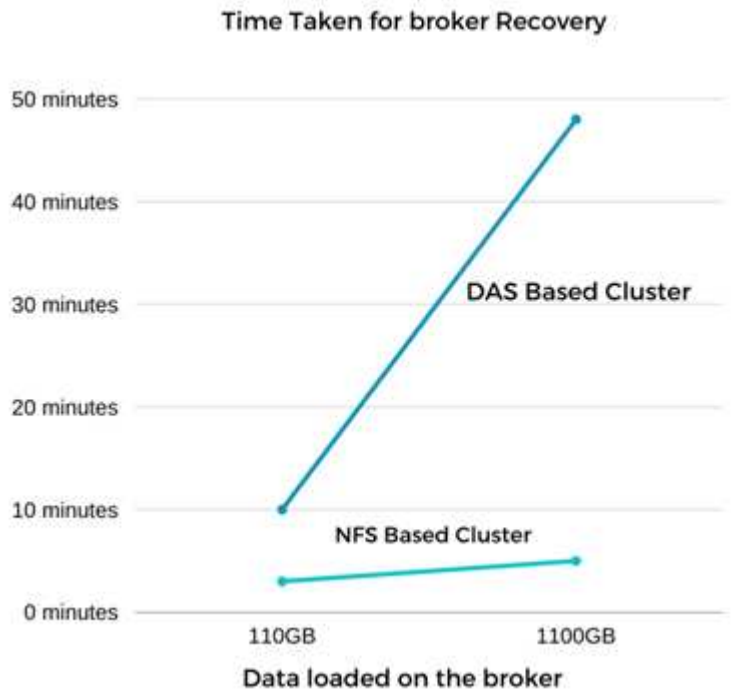
1. 備份節點於 09:39:17,213 啟動。下面突出顯示了起始日誌條目。

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

2. 資料重建過程於 09:42:29,115 結束。處理 110GB 的資料大約需要 3 分鐘。

```
[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)
```

對包含約 1TB 資料的代理程式重複了測試，對於 DAS 大約需要 48 分鐘，對於 NFS 大約需要 3 分鐘。結果如下圖所示。



儲存效率

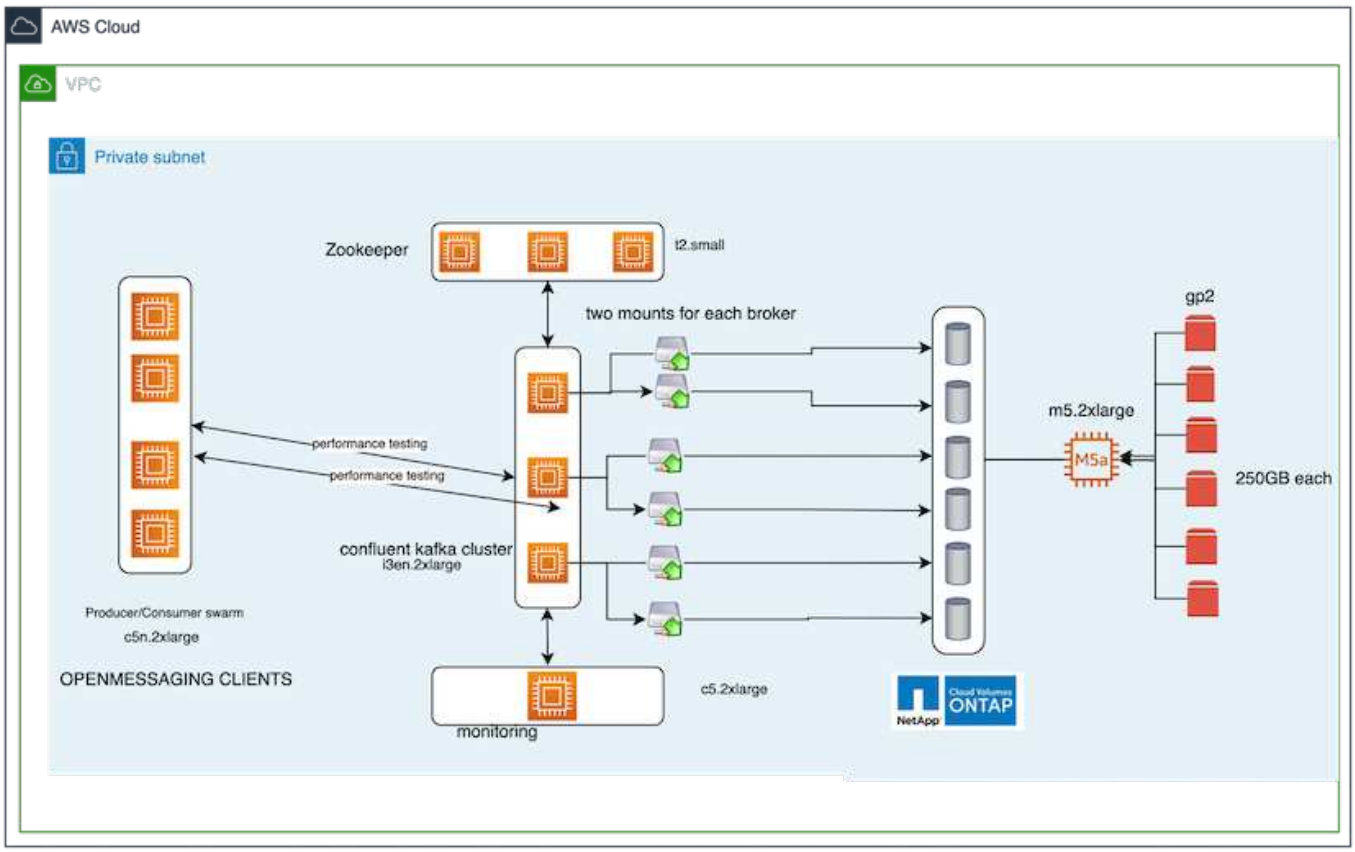
由於 Kafka 叢集的儲存層是透過NetApp ONTAP配置的，因此我們獲得了ONTAP的所有儲存效率功能。這是透過在Cloud Volumes ONTAP上配置 NFS 儲存的 Kafka 叢集上產生大量資料進行的測試。我們可以看到，由於ONTAP功能，空間顯著減少。

建築設置

下表展示了使用NAS的Kafka叢集的環境配置。

平台組件	環境配置
卡夫卡 3.2.3	• 3 位動物園管理員 – t2.small • 3 個代理伺服器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生產者/消費者 — c5n.2xlarge *
所有節點上的作業系統	RHEL8.7 或更高版本
NetApp Cloud Volumes ONTAP實例	單一節點實例 – M5.2xLarge

下圖是基於NAS的Kafka叢集架構圖。



- *計算。 *我們使用了三節點 Kafka 集群，並在專用伺服器上運行三節點 zookeeper 集合。每個代理程式透過專用 LIF 擁有兩個指向NetApp CVO 實例上的單一磁碟區的 NFS 掛載點。
- *監控*我們使用兩個節點來實現 Prometheus-Grafana 組合。為了產生工作負載，我們使用了一個單獨的三節點集群，該集群可以為該 Kafka 集群生產和消費。
- *貯存。 *我們使用了單節點NetApp Cloud Volumes ONTAP實例，該實例上安裝了六個 250GB GP2 AWS-EBS 磁碟區。然後，這些磁碟區透過專用 LIF 作為六個 NFS 磁碟區公開給 Kafka 叢集。
- *配置。 *此測試案例中的可設定元素是 Kafka 代理。

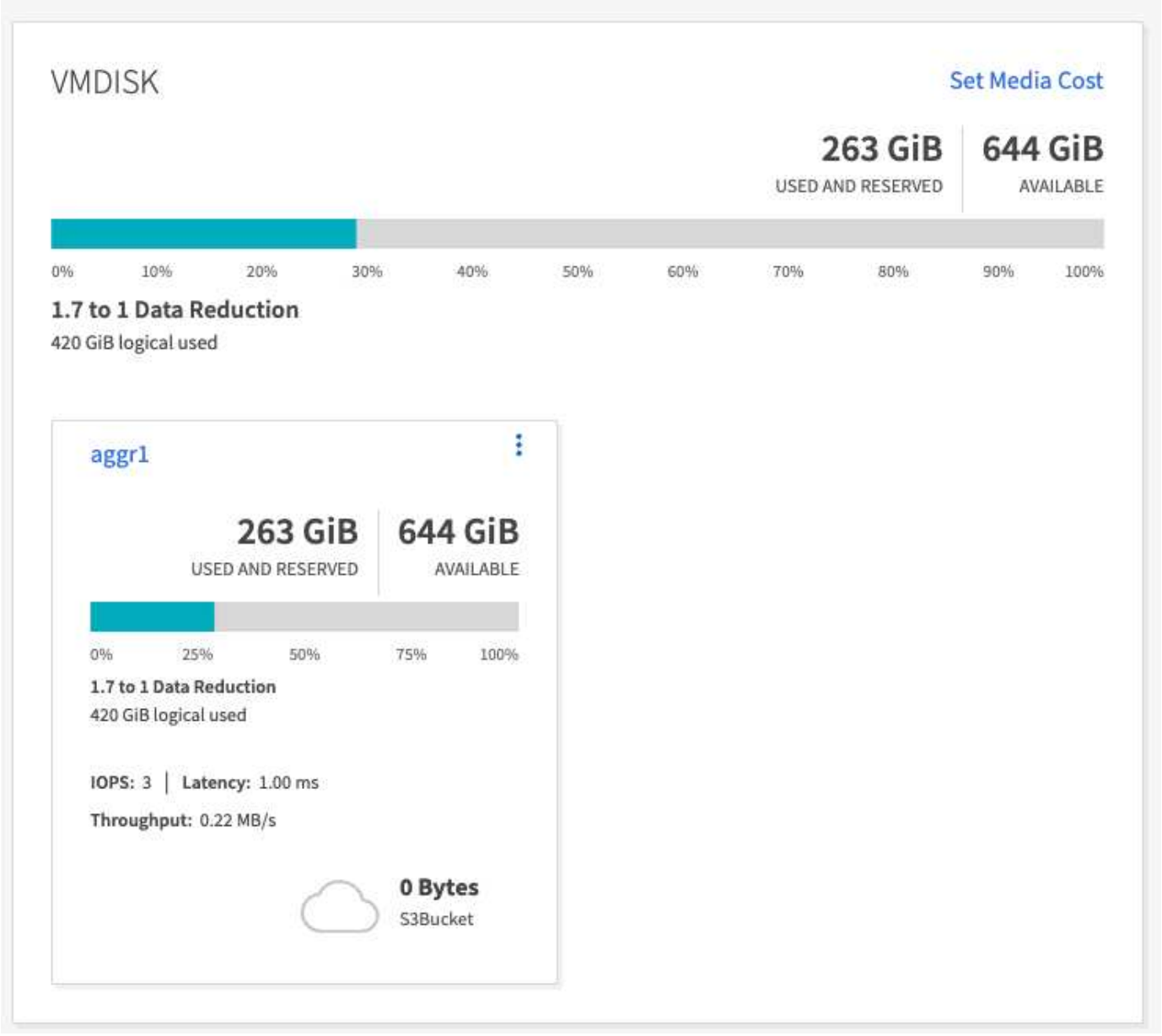
生產者端的壓縮被關閉，從而使生產者能夠產生高吞吐量。儲存效率由運算層處理。

測試方法

1. 已按照上述規格配置了 Kafka 叢集。
2. 在叢集上，使用 OpenMessaging Benchmarking 工具產生了約 350GB 的資料。
3. 工作負載完成後，使用ONTAP系統管理器和 CLI 收集儲存效率統計資料。

觀察

對於使用 OMB 工具產生的數據，我們發現空間節省了約 33%，儲存效率比為 1.70:1。如下圖所示，產生的資料所使用的邏輯空間為420.3GB，用於保存資料的實體空間為281.7GB。



```
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
```

AWS 中的效能概述和驗證

在NetApp NFS 上安裝儲存層的 Kafka 叢集在 AWS 雲端進行了效能基準測試。基準測試範例在以下章節中說明。

AWS 雲端中的 Kafka 與NetApp Cloud Volumes ONTAP （高可用性對和單節點）

對具有NetApp Cloud Volumes ONTAP （HA 對）的 Kafka 叢集在 AWS 雲端中進行了效能基準測試。以下章節描述了此基準測試。

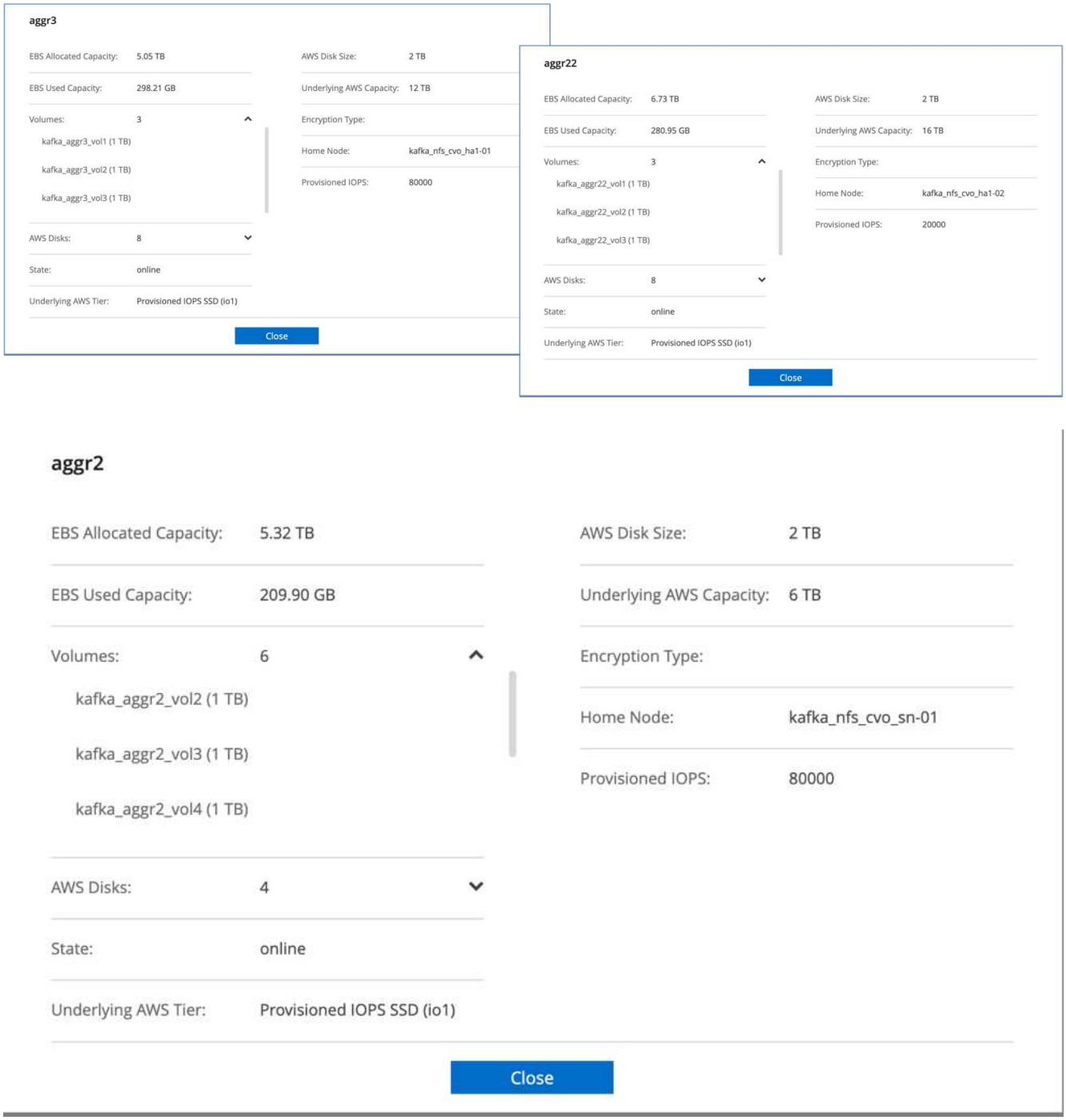
建築設置

下表展示了使用NAS的Kafka叢集的環境配置。

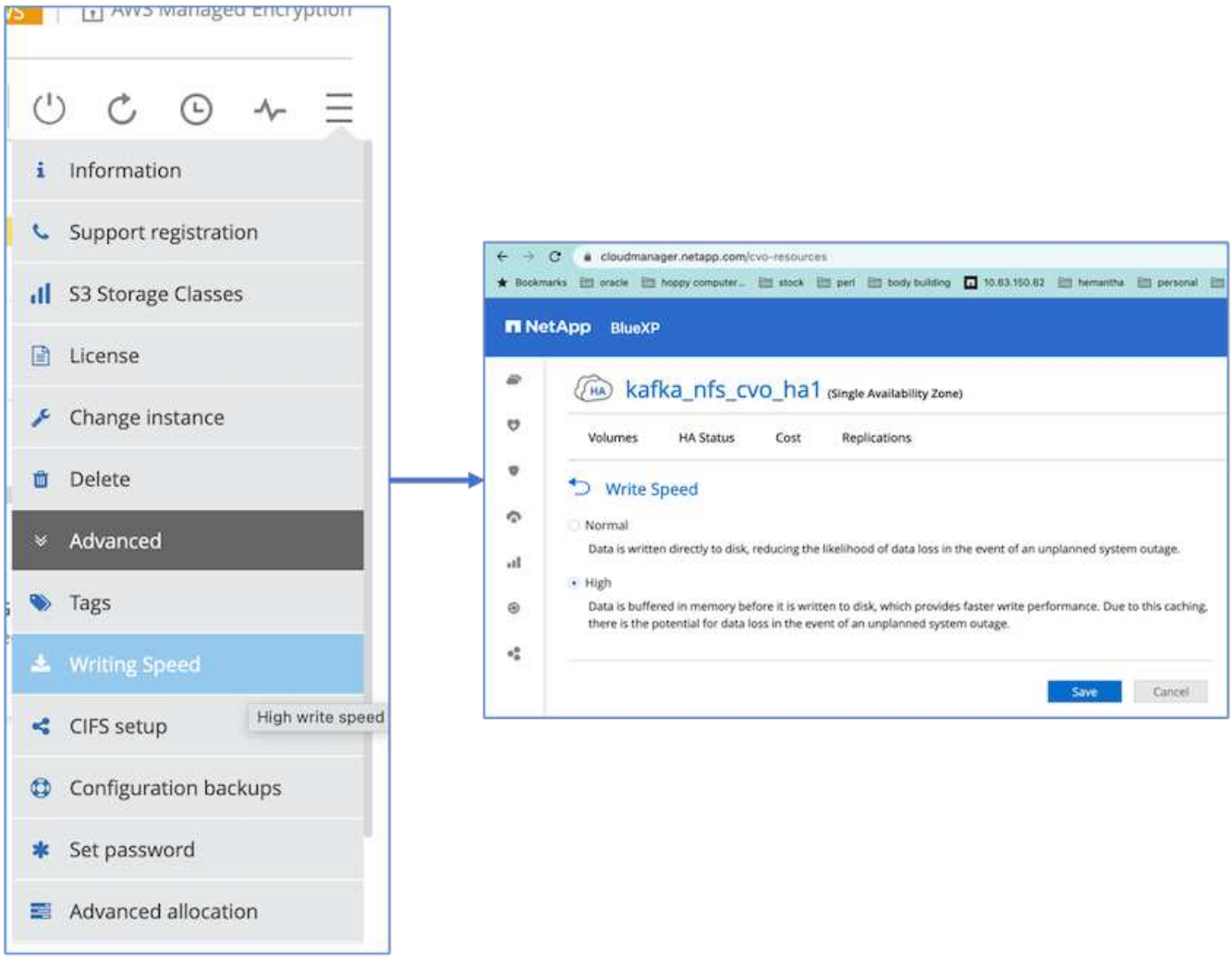
平台組件	環境配置
卡夫卡 3.2.3	• 3 位動物園管理員 – t2.small • 3 個代理伺服器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生產者/消費者 — c5n.2xlarge *
所有節點上的作業系統	RHEL8.6
NetApp Cloud Volumes ONTAP實例	HA 對實例 – m5dn.12xLarge x 2node 單節點實例 - m5dn.12xLarge x 1 節點

NetApp叢集磁碟區**ONTAP**設定

1. 對於Cloud Volumes ONTAP HA 對，我們建立了兩個聚合，每個儲存控制器上的每個聚合上有三個磁碟區。對於單一Cloud Volumes ONTAP節點，我們在一個聚合中建立六個磁碟區。

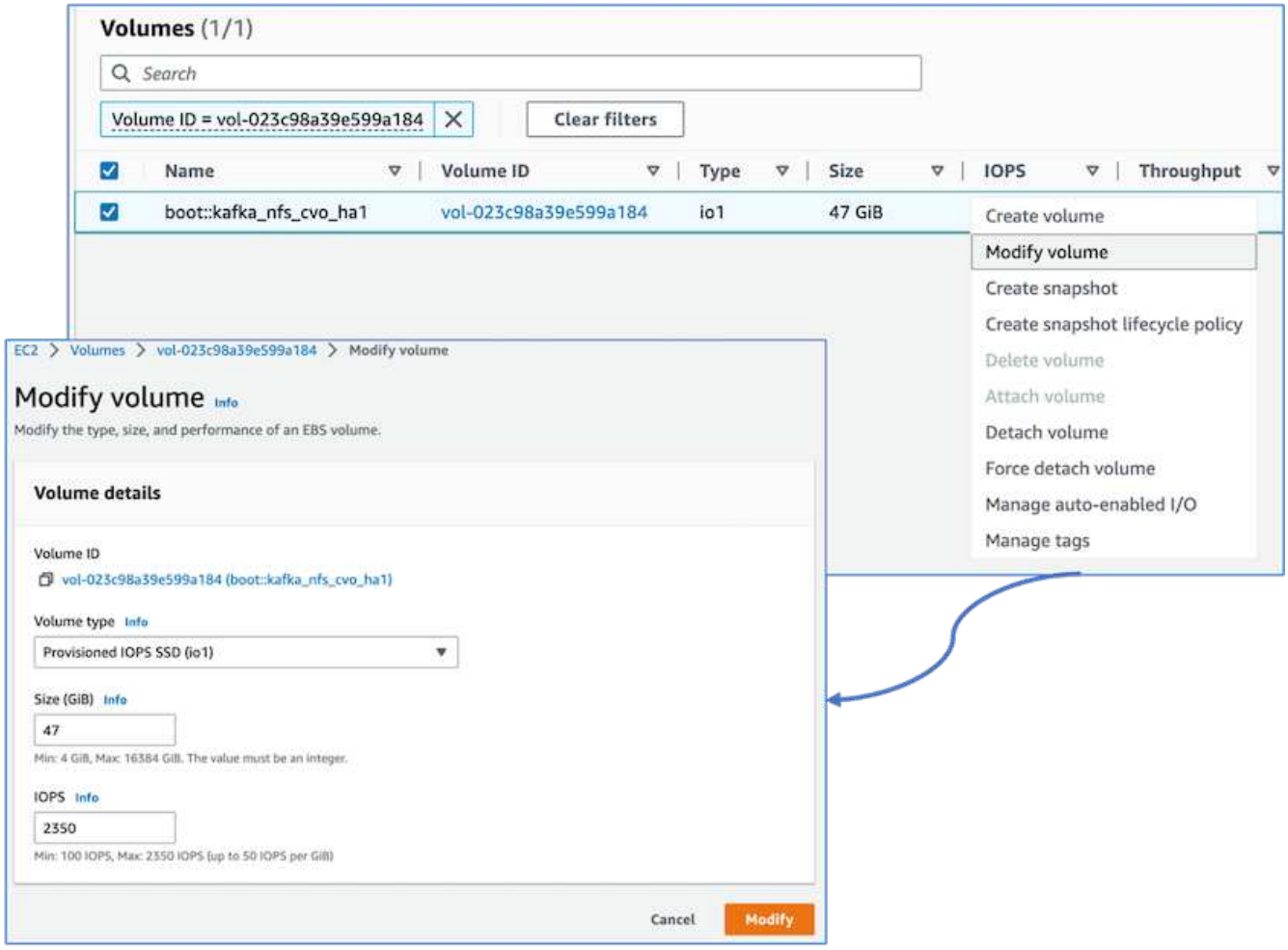


2. 為了實現更好的網路效能，我們為 HA 對和單一節點都啟用了高速網路。



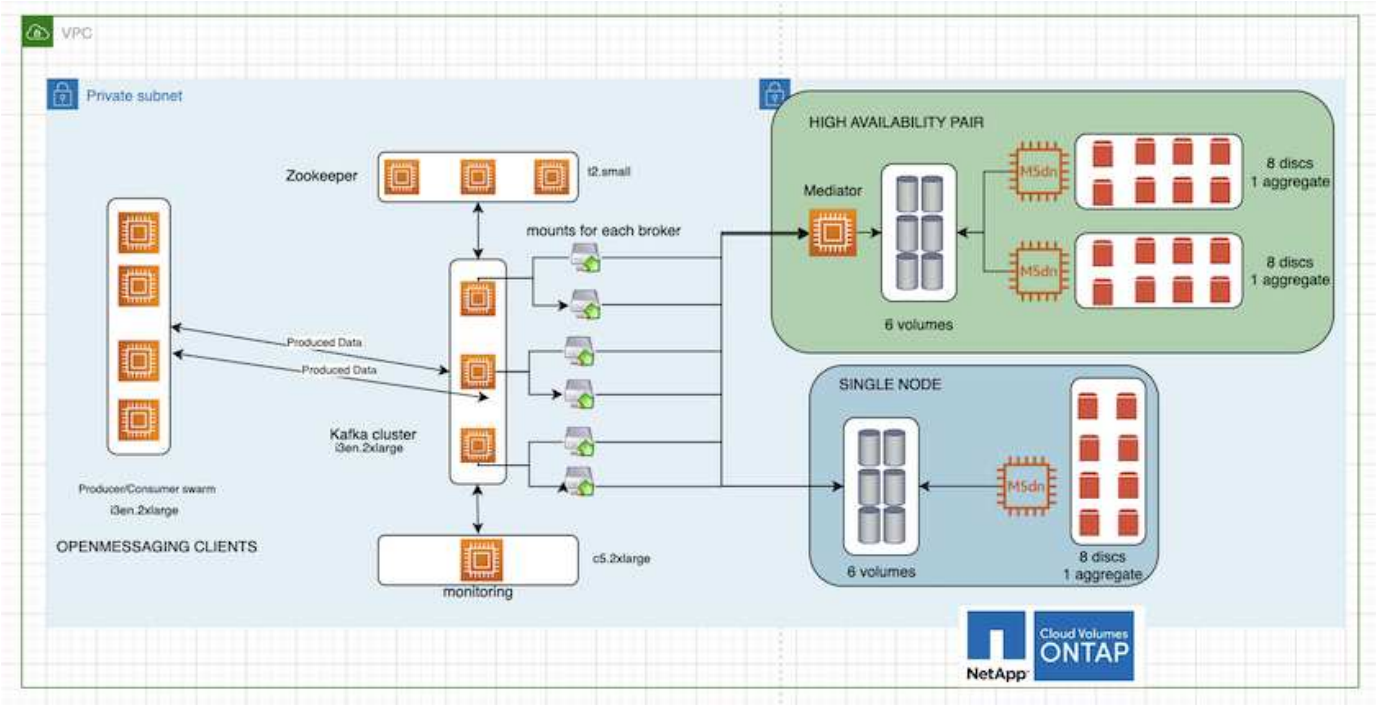
3. 我們注意到ONTAP NVRAM具有更多的 IOPS，因此我們將Cloud Volumes ONTAP根磁碟區的 IOPS 變更為 2350。 Cloud Volumes ONTAP中的根磁碟區大小為 47GB。以下ONTAP指令適用於 HA 對，相同步驟也適用於單一節點。

```
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::*>
```



下圖是基於NAS的Kafka叢集架構圖。

- *計算。 *我們使用了三節點 Kafka 集群，並在專用伺服器上運行三節點 zookeeper 集合。每個代理程式透過專用 LIF 擁有兩個指向Cloud Volumes ONTAP實例上的單一磁碟區的 NFS 掛載點。
- *監控*我們使用兩個節點來實現 Prometheus-Grafana 組合。為了產生工作負載，我們使用了一個單獨的三節點集群，該集群可以為該 Kafka 集群生產和消費。
- *貯存。 *我們使用了 HA 對 Cloud Volumes ONTAP實例，並在該實例上安裝了一個 6TB GP3 AWS-EBS 磁碟區。然後透過 NFS 掛載將該磁碟區匯出到 Kafka 代理程式。



OpenMessage 基準測試配置

1. 為了獲得更好的 NFS 效能，我們需要在 NFS 伺服器和 NFS 用戶端之間建立更多的網路連接，可以使用 nconnect 建立。透過執行下列命令，使用 nconnect 選項將 NFS 磁碟區掛載到代理節點上：

```
[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. 檢查Cloud Volumes ONTAP中的網路連線。下列ONTAP指令從單一Cloud Volumes ONTAP節點使用。相同的步驟適用於Cloud Volumes ONTAP HA 對。

```
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. 我們使用以下 Kafka `server.properties`在Cloud Volumes ONTAP HA 對的所有 Kafka 代理中。這 `log.dirs`屬性對每個經紀人都是不同的，其餘屬性對經紀人來說是共同的。對於 broker1 來說， `log.dirs`值如下：

```
[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 ~]#
```

◦ 對於 broker2， `log.dirs`屬性值如下：

```
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
```

◦ 對於 broker3， `log.dirs`屬性值如下：

```
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. 對於單一Cloud Volumes ONTAP節點，Kafka `servers.properties`與Cloud Volumes ONTAP HA 對相同，但`log.dirs`財產。
   - 對於 broker1 來說， `log.dirs`值如下：

```
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
```

   - 對於 broker2， `log.dirs`值如下：

```
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
```

   - 對於 broker3， `log.dirs`屬性值如下：

```
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. OMB 中的工作負載配置有以下屬性： `(/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
```

這 `messageSize`根據每個用例可能會有所不同。在我們的效能測試中，我們使用了 3K。

我們使用了來自 OMB 的兩個不同的驅動程式（Sync 或 Throughput）來在 Kafka 叢集上產生工作負載。

- 用於同步驅動程式屬性的 yaml 檔案如下 (/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
```

- 用於吞吐量驅動程式屬性的 yaml 檔案如下 (/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
```

測試方法

1. 根據上面描述的規範，使用 Terraform 和 Ansible 配置了 Kafka 叢集。 Terraform 用於使用 AWS 執行個體為 Kafka 叢集建置基礎設施，Ansible 在其上建置 Kafka 叢集。
2. 使用上面描述的工作負載配置和同步驅動程式觸發了 OMB 工作負載。

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

3. 使用具有相同工作負載配置的吞吐量驅動程式觸發了另一個工作負載。

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

觀察

使用兩種不同類型的驅動程式來產生工作負載，以對在 NFS 上執行的 Kafka 執行個體的效能進行基準測試。驅動程式之間的差異在於日誌刷新屬性。

對於Cloud Volumes ONTAP HA 對：

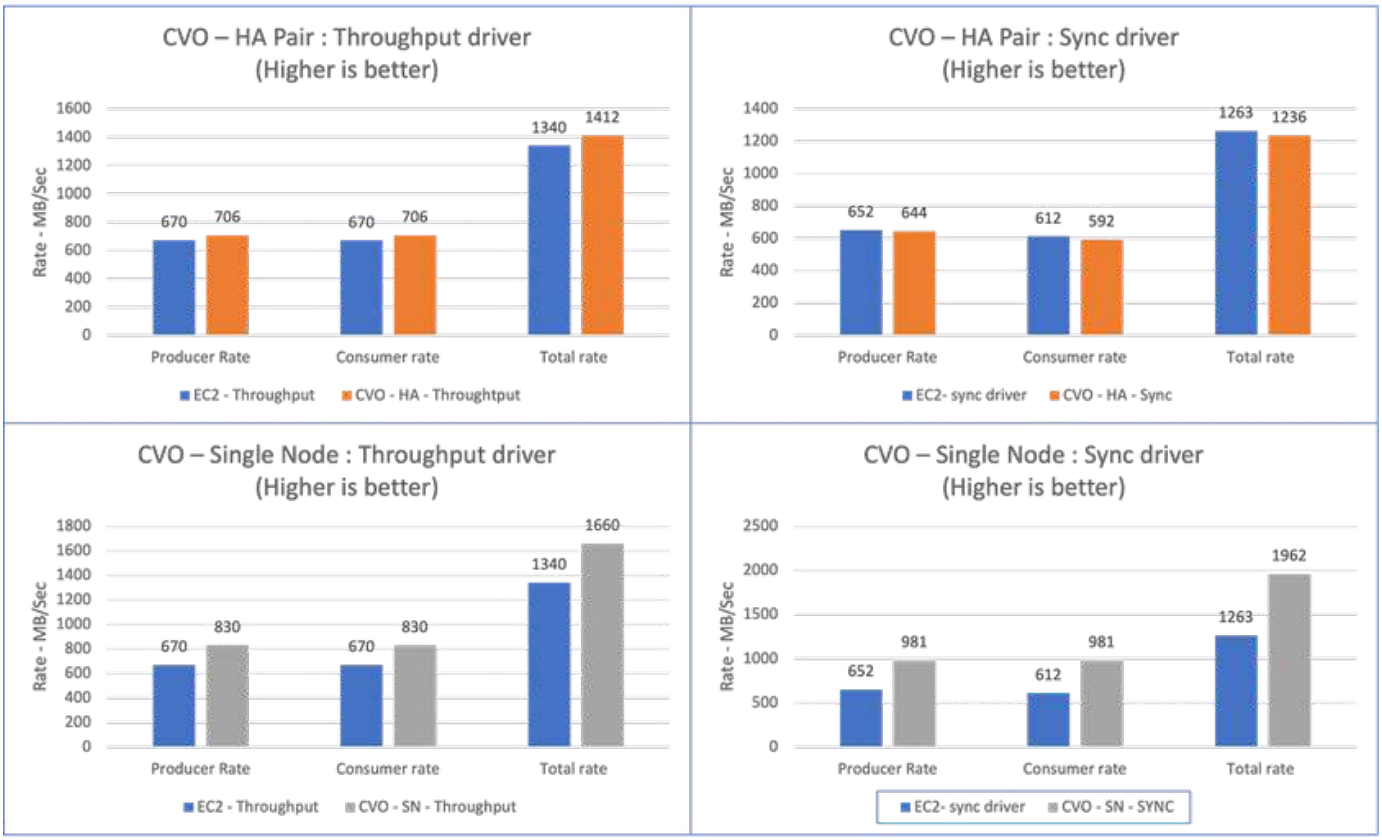
- 同步驅動程式持續產生的總吞吐量：~1236 MBps。
- 吞吐量驅動程式產生的總吞吐量：峰值~1412 MBps。

對於單一Cloud Volumes ONTAP節點：

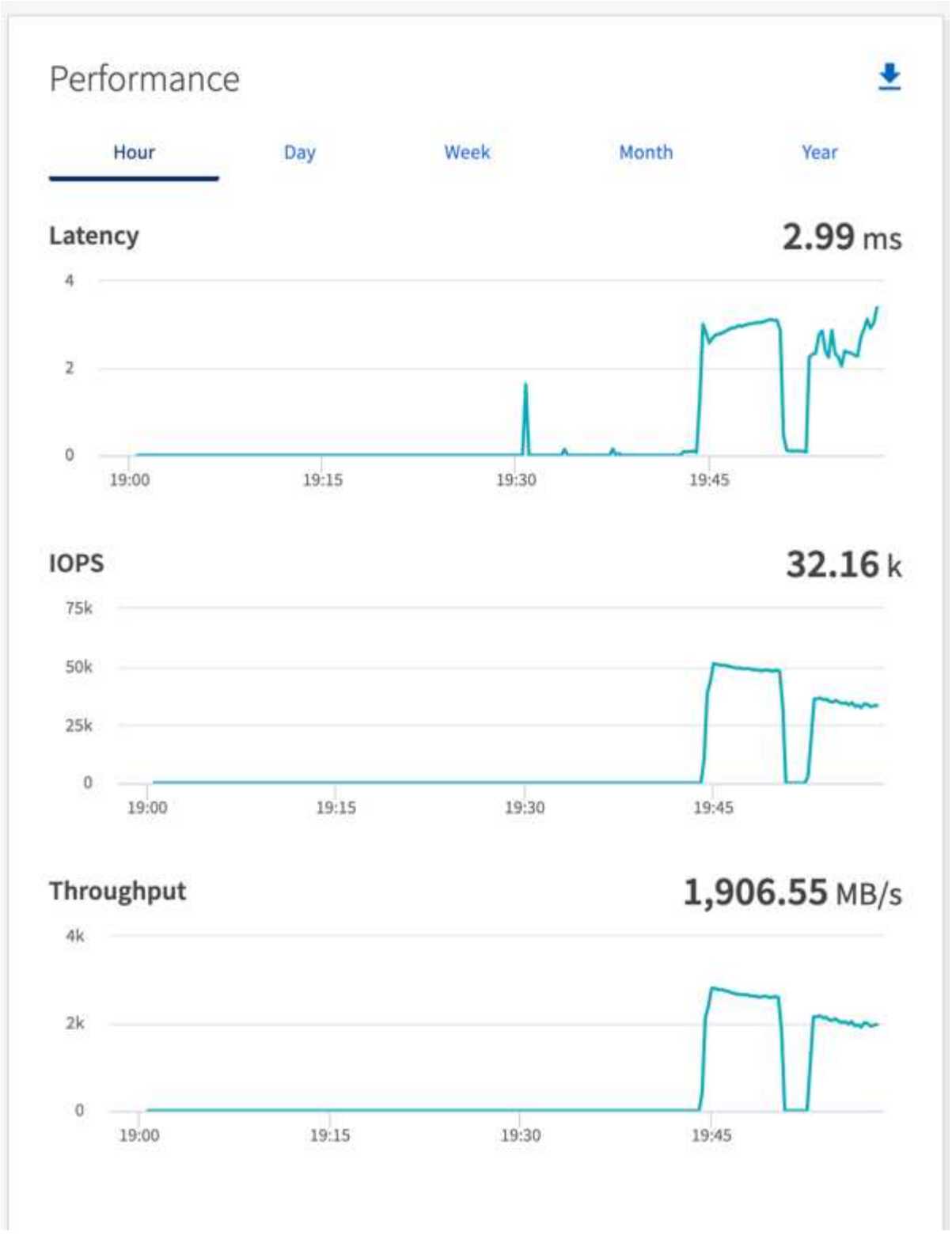
- 同步驅動程式持續產生的總吞吐量：~ 1962MBps。
- 吞吐量驅動程式產生的總吞吐量：峰值~1660MBps

由於日誌會立即刷新到磁碟，因此同步驅動程式可以產生一致的吞吐量，而由於日誌會批次提交到磁碟，因此吞吐量驅動程式會產生突發性的吞吐量。

這些吞吐量數字是針對給定的 AWS 配置產生的。對於更高的效能要求，可以擴大實例類型並進一步調整以獲得更好的吞吐量數字。總吞吐量或總速率是生產者和消費者速率的組合。



在執行吞吐量或同步驅動程式基準測試時，請務必檢查儲存吞吐量。



AWS FSx ONTAP中的效能概述與驗證

在NetApp NFS 上安裝儲存層的 Kafka 叢集在 AWS FSx ONTAP中進行了效能基準測試。基準測試範例在以下章節中說明。

AWS FSx ONTAP中的 Apache Kafka

網路檔案系統 (NFS) 是一種廣泛用於儲存大量資料的網路檔案系統。在大多數組織中，資料越來越多地由 Apache Kafka 等串流應用程式產生。這些工作負載需要可擴展性、低延遲以及具有現代儲存功能的強大資料擷取架構。為了實現即時分析並提供可操作的見解，需要精心設計且高效能的基礎架構。

Kafka 在設計上與 POSIX 相容的檔案系統協同工作，並依賴該檔案系統來處理檔案操作，但是在 NFSv3 檔案系統上儲存資料時，Kafka 代理程式 NFS 用戶端可以以不同於 XFS 或 Ext4 等本機檔案系統的方式解釋檔案操作。一個常見的例子是 NFS Silly 重新命名，它導致 Kafka 代理在擴展叢集和重新分配分區時失敗。為了應對這項挑戰， NetApp更新了開源 Linux NFS 用戶端，這些變更現在已在 RHEL8.7、RHEL9.1 中普遍可用，並且從目前 FSx ONTAP版本ONTAP 9.12.1 開始受支援。

Amazon FSx ONTAP在雲端提供完全託管、可擴展且高效能的 NFS 檔案系統。 FSx ONTAP上的 Kafka 資料可以擴展以處理大量資料並確保容錯能力。 NFS 為關鍵和敏感資料集提供集中儲存管理和資料保護。

這些增強功能使 AWS 客戶能夠在 AWS 運算服務上執行 Kafka 工作負載時利用 FSx ONTAP 。這些好處是：* 降低 CPU 使用率以減少 I/O 等待時間* 更快的 Kafka 代理程式恢復時間。 * 可靠性和效率。 * 可擴展性和效能。 * 多可用區域可用性。 * 資料保護。

AWS FSx ONTAP中的效能概述與驗證

在NetApp NFS 上安裝儲存層的 Kafka 叢集在 AWS 雲端進行了效能基準測試。基準測試範例在以下章節中說明。

AWS FSx ONTAP中的 Kafka

使用 AWS FSx ONTAP 的Kafka 叢集在 AWS 雲端中進行了效能基準測試。以下章節描述了此基準測試。

建築設置

下表顯示了使用 AWS FSx ONTAP 的Kafka 叢集的環境配置。

平台組件	環境配置
卡夫卡 3.2.3	• 3 位動物園管理員 – t2.small • 3 個代理伺服器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生產者/消費者 — c5n.2xlarge *
所有節點上的作業系統	RHEL8.6
AWS FSx ONTAP	多可用區，吞吐量 4GB/秒，IOPS 160000

NetApp FSx ONTAP設定

1. 在我們的初步測試中，我們建立了一個容量為 2TB、吞吐量為 40000 IOP 的 FSx ONTAP檔案系統，吞吐量為 2GB/秒。

```
[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"}
```

在我們的範例中，我們透過 AWS CLI 部署 FSx ONTAP 。您將需要根據需要在您的環境中進一步自訂命令。 FSx ONTAP還可以透過 AWS 控制台進行部署和管理，從而透過更少的命令列輸入獲得更輕鬆、更簡化的部署體驗。

文件在 FSx ONTAP中，我們測試區域（US-East-1）中 2GB/秒吞吐量檔案系統可實現的最大 IOPS 為 80,000 iops。 FSx ONTAP檔案系統的最大總 iops 為 160,000 iops，需要 4GB/秒的吞吐量部署才能實現，我們將在本文檔的後面進行演示。

有關 FSx ONTAP性能規格的更多信息，請隨時訪問此處的 AWS FSx ONTAP文件：https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html 。

FSx「create-file-system」的詳細命令列語法可以在這裡找到：https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

例如，您可以指定特定的 KMS 金鑰，而不是在未指定 KMS 金鑰時使用的預設 AWS FSx 主金鑰。

2. 在建立 FSx ONTAP檔案系統時，請按照以下方式描述您的檔案系統，等待 JSON 返回中的「LifeCycle」狀態變為「AVAILABLE」：

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

3. 透過使用 fsxadmin 使用者登入 FSx ONTAP SSH 來驗證憑證：Fsxadmin 是 FSx ONTAP檔案系統建立時的預設管理員帳戶。 fsxadmin 的密碼是首次在 AWS 主控台或使用 AWS CLI 建立檔案系統時所配置的密碼，如我們在步驟 1 中完成的密碼，如我們在步驟 1 中完成的密碼。

```
[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. 驗證您的憑證後，在 FSx ONTAP檔案系統上建立儲存虛擬機

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

儲存虛擬機器 (SVM) 是一個獨立的檔案伺服器，具有自己的管理憑證和端點，用於管理和存取 FSx ONTAP 磁碟區中的數據，並提供 FSx ONTAP多租用戶。

5. 配置好主儲存虛擬機器後，透過 SSH 進入新建立的 FSx ONTAP檔案系統，並使用下列範例命令在儲存虛擬機器中建立卷，同樣，我們為此驗證會建立 6 個磁碟區。根據我們的驗證，保留預設成分（8）或更少的成分將為 kafka 提供更好的性能。

```
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. 我們的測試需要額外的容量。將磁碟區的大小擴展至 2TB 並安裝在連線路徑上。

```
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
```

在 FSx ONTAP中，磁碟區可以進行精簡配置。在我們的範例中，擴充磁碟區的總容量超過了檔案系統的總容量，因此我們需要擴展檔案系統的總容量以解鎖額外的預配置磁碟區容量，我們將在下一步中示範這一點。

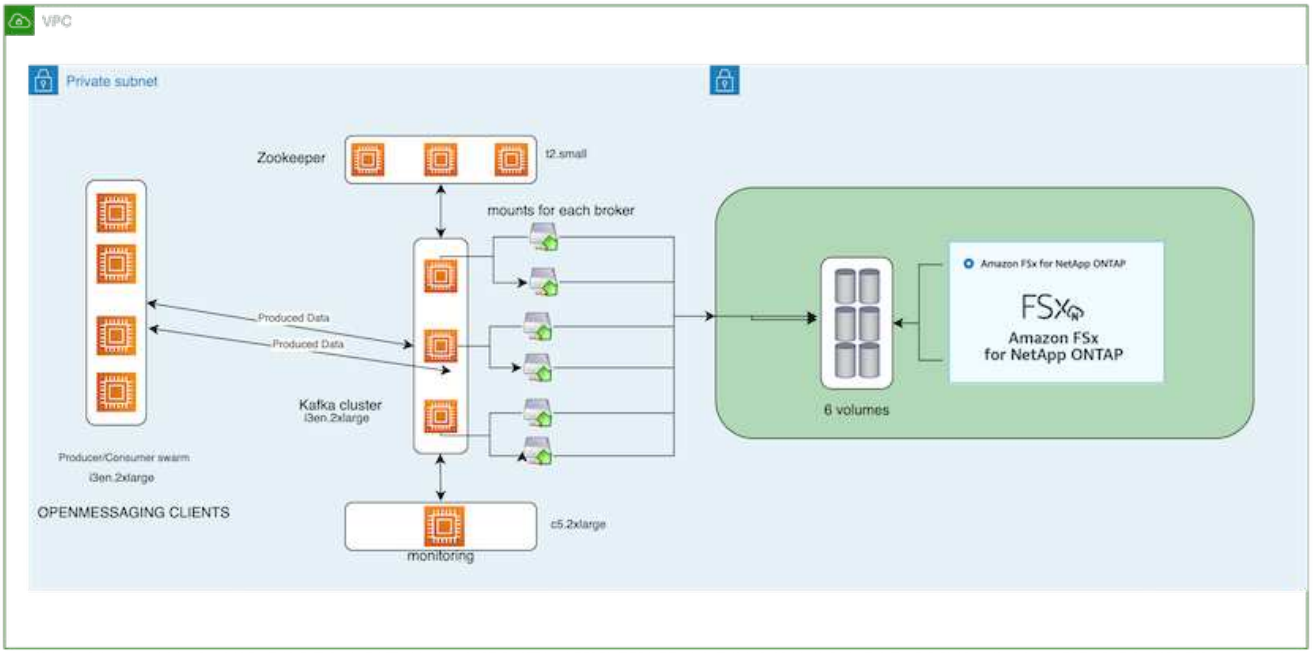
7. 接下來，為了提高效能和容量，我們將 FSx ONTAP吞吐容量從 2GB/秒擴展到 4GB/秒，IOPS 擴展到 160000，容量擴展到 5 TB

```
[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
```

FSx「update-file-system」的詳細命令列語法可以在這裡找到
：https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. FSx ONTAP磁碟區透過 nconnect 和 Kafka 代理程式中的預設選項進行掛載

下圖展示了我們基於 FSx ONTAP 的Kafka 叢集的最終架構：



- 計算。我們使用了三節點 Kafka 集群，並在專用伺服器上運行三節點 zookeeper 集合。每個代理程式有六個 NFS 掛載點，分別指向 FSx ONTAP實例上的六個磁碟區。
- 監控。我們使用兩個節點來實現 Prometheus-Grafana 組合。為了產生工作負載，我們使用了一個單獨的三節點集群，該集群可以為該 Kafka 集群生產和消費。
- 貯存。我們使用了安裝了六個 2TB 磁碟區的 FSx ONTAP 。然後將該磁碟區匯出到具有 NFS 掛載的 Kafka 代理程式。 FSx ONTAP磁碟區在 Kafka 代理程式中安裝了 16 個 nconnect 會話和預設選項。

OpenMessage 基準測試配置。

我們使用了與NetApp Cloud Volumes ONTAP相同的配置，其詳細資訊請參閱此處 - xref:./data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

測試方法

1. 根據上面描述的規範，使用 terraform 和 ansible 配置了 Kafka 叢集。 Terraform 用於使用 AWS 執行個體為 Kafka 叢集建置基礎設施，並且 ansible 在其上建置 Kafka 叢集。
2. 使用上面描述的工作負載配置和同步驅動程式觸發了 OMB 工作負載。

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

3. 使用具有相同工作負載配置的吞吐量驅動程式觸發了另一個工作負載。

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

觀察

使用兩種不同類型的驅動程式來產生工作負載，以對在 NFS 上執行的 Kafka 執行個體的效能進行基準測試。驅動程式之間的差異在於日誌刷新屬性。

對於 Kafka 複製因子 1 和 FSx ONTAP：

- 同步驅動程式持續產生的總吞吐量：~ 3218 MBps，峰值效能約為 3652 MBps。
- 吞吐量驅動程式持續產生的總吞吐量：~ 3679 MBps，峰值效能為 ~ 3908 MBps。

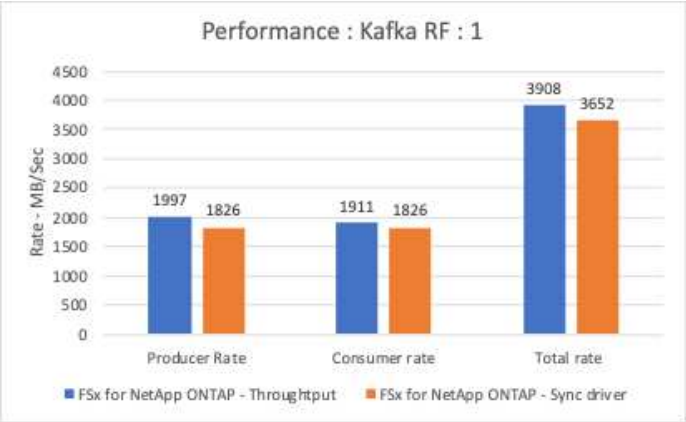
對於複製因子為 3 且具有 FSx ONTAP 的Kafka：

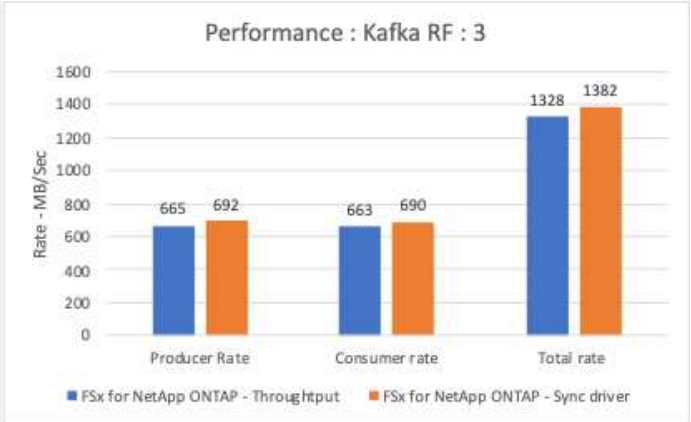
- 同步驅動程式持續產生的總吞吐量：~ 1252 MBps，峰值效能約為 1382 MBps。
- 吞吐量驅動程式持續產生的總吞吐量：~ 1218 MBps，峰值效能約為 1328 MBps。

在 Kafka 複製因子 3 中，讀寫操作在 FSx ONTAP上發生了三次，在 Kafka 複製因子 1 中，讀寫操作在 FSx ONTAP上發生了一次，因此在兩種驗證中，我們都能夠達到 4GB/秒的最大吞吐量。

由於日誌會立即刷新到磁碟，因此同步驅動程式可以產生一致的吞吐量，而由於日誌會批次提交到磁碟，因此吞吐量驅動程式會產生突發性的吞吐量。

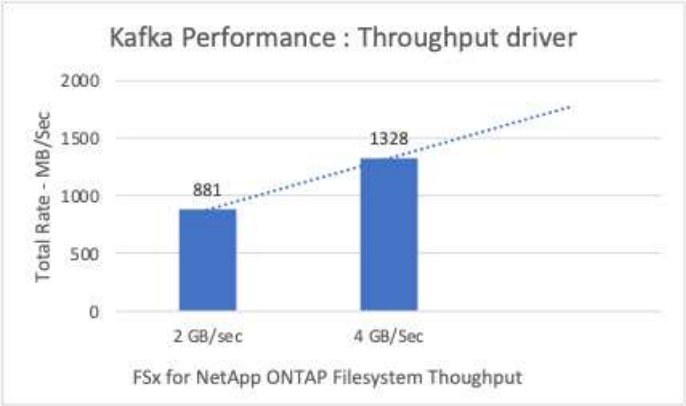
這些吞吐量數字是針對給定的 AWS 配置產生的。對於更高的效能要求，可以擴大實例類型並進一步調整以獲得更好的吞吐量數字。總吞吐量或總速率是生產者和消費者速率的組合。

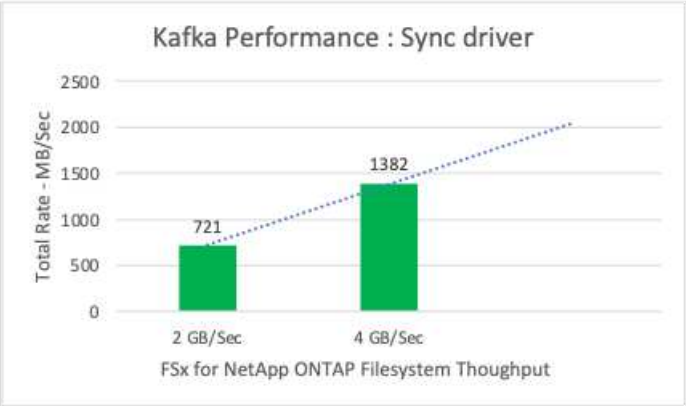




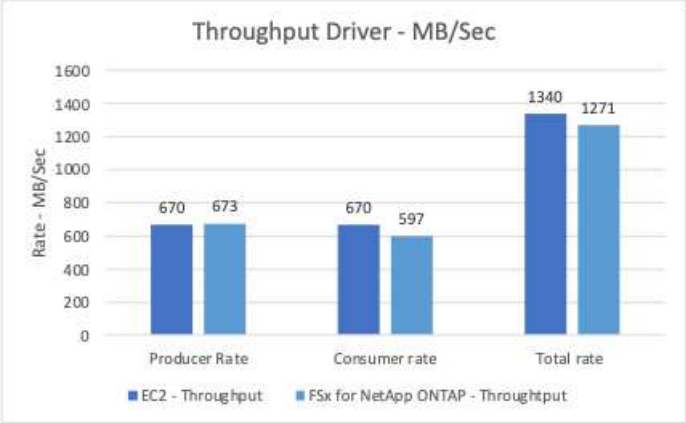
下圖顯示了 Kafka 複製因子 3 的 2GB/秒 FSx ONTAP和 4GB/秒效能。複製因子 3 在 FSx ONTAP儲存上執行三次讀寫操作。吞吐量驅動程式的總速率為 881 MB/秒，在 2GB/秒 FSx ONTAP檔案系統上以大約 2.64 GB/秒的速度讀取和寫入 Kafka 操作，吞吐量驅動程式的總速率為 1328 MB/秒，以大約 3.98 GB/秒的速度讀取和寫入

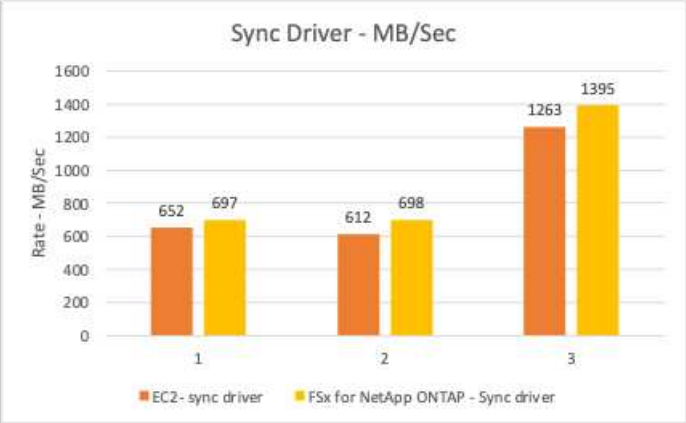
kafka 操作。 Kafka 效能是線性的，並且基於 FSx ONTAP吞吐量可擴充。





下圖顯示了 EC2 執行個體與 FSx ONTAP之間的效能（Kafka 複製因子：3）

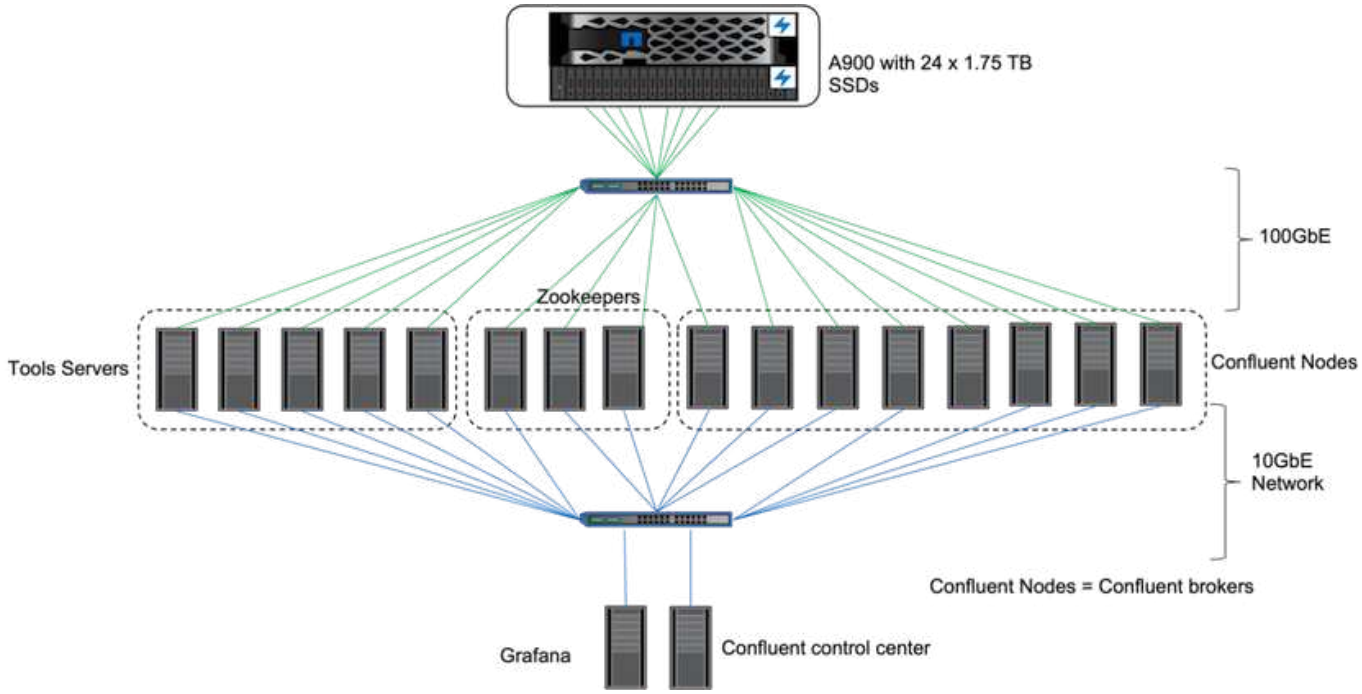




本地AFF A900的效能概述與驗證

在本地，我們使用具有ONTAP 9.12.1RC1 的NetApp AFF A900儲存控制器來驗證 Kafka 叢集的效能和擴展性。我們使用了與先前的ONTAP和AFF分層儲存最佳實踐相同的測試平台。

我們使用 Confluent Kafka 6.2.0 來評估AFF A900。該叢集有八個代理節點和三個 zookeeper 節點。為了進行效能測試，我們使用了五個 OMB 工作節點。



儲存配置

我們使用NetApp FlexGroups 實例為日誌目錄提供單一命名空間，從而簡化復原和配置。我們使用 NFSv4.1 和 pNFS 來提供對日誌段資料的直接路徑存取。

客戶端調優

每個客戶端使用以下命令掛載FlexGroup實例。

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

此外，我們增加了 `max_session_slots`` 從預設 `64`到 `180`。這與ONTAP中的預設會話槽限制相符。

Kafka 代理調優

為了最大限度地提高測試系統的吞吐量，我們顯著增加了某些關鍵線程池的預設參數。對於大多數配置，我們建議遵循 Confluent Kafka 最佳實踐。此調整用於最大化儲存的未完成 I/O 的並發性。這些參數可以進行調整以符合您的代理程式的運算資源和儲存屬性。

```
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
```

工作負載產生器測試方法

我們對吞吐量驅動程式和主題配置使用了與雲端測試相同的 OMB 配置。

1. 使用 Ansible 在AFF叢集上設定了FlexGroup實例。

```
---
- 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. ONTAP SVM 上已啟用 pNFS。

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

3. 工作負載由吞吐量驅動程式觸發，使用與Cloud Volumes ONTAP相同的工作負載配置。請參閱“[穩態性能] “以下。工作負載使用的複製因子為 3，這表示在 NFS 中維護了三個日誌段副本。

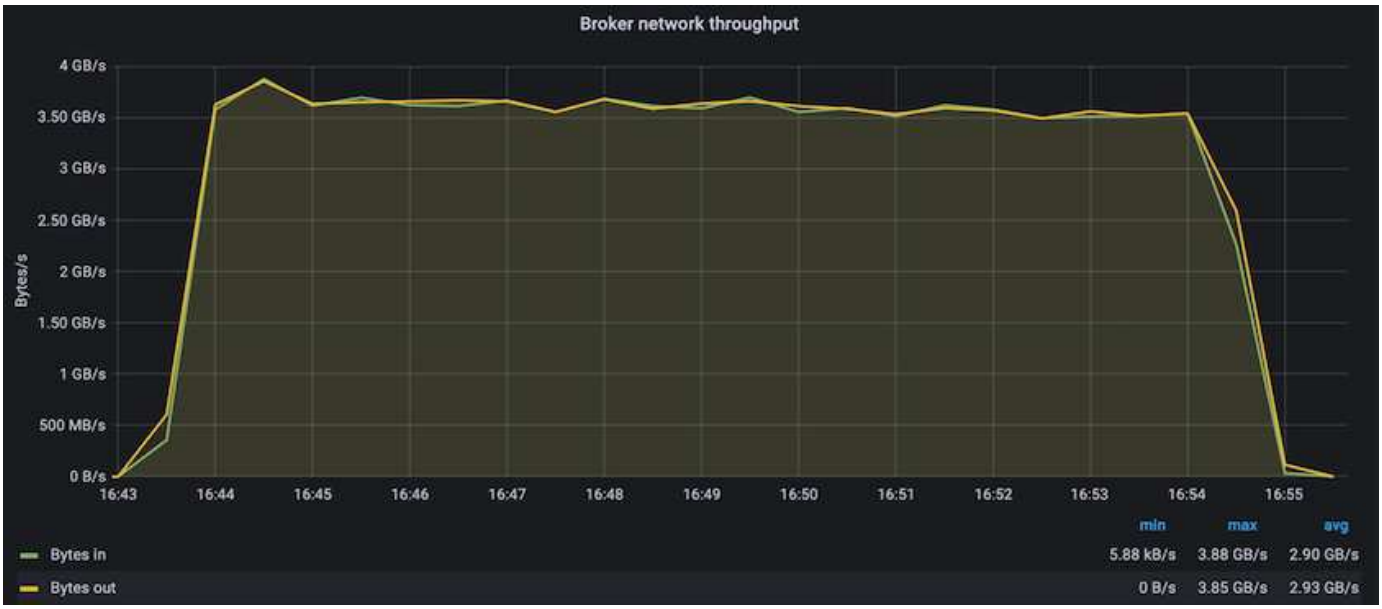
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最後，我們使用積壓完成了測量，以衡量消費者趕上最新消息的能力。 OMB 透過在測量開始時暫停消費者來建立積壓。這會產生三個不同的階段：積壓創建（僅生產者的流量）、積壓消耗（消費者密集的階段，其中消費者追趕主題中錯過的事件）和穩定狀態。請參閱“[極致效能與探索儲存極限] ”獲取更多資訊。

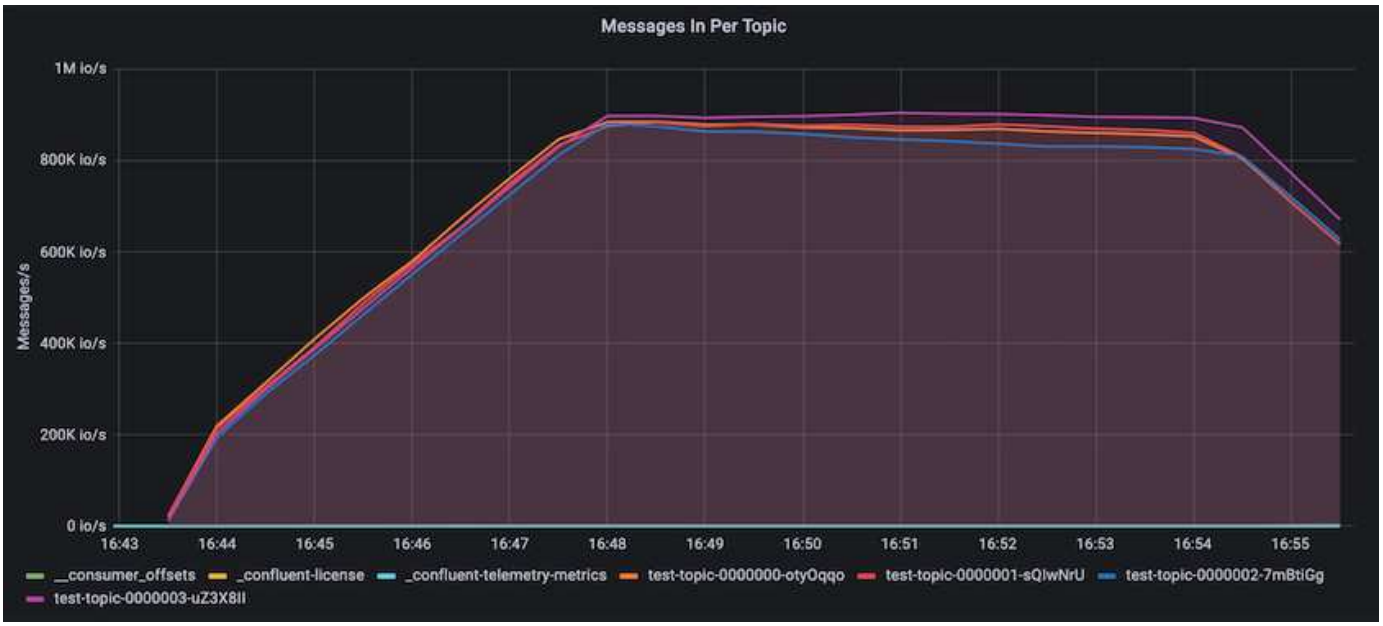
穩態性能

我們使用 OpenMessaging Benchmark 評估了AFF A900 ，以提供與 AWS 中的Cloud Volumes ONTAP和 AWS 中的 DAS 類似的比較。所有效能值代表生產者和消費者層級的 Kafka 叢集吞吐量。

Confluent Kafka 和AFF A900的穩定狀態效能為生產者和消費者實現了超過 3.4GBps 的平均吞吐量。 Kafka 叢集中的訊息超過 340 萬條。透過以每秒位元組數為單位視覺化 BrokerTopicMetrics 的持續吞吐量，我們可以看到AFF A900支援的出色穩定狀態效能和流量。



這與按主題傳遞的訊息的視圖非常一致。下圖提供了按主題的細分情況。在測試的配置中，我們看到四個主題中每個主題有近 90 萬個訊息。

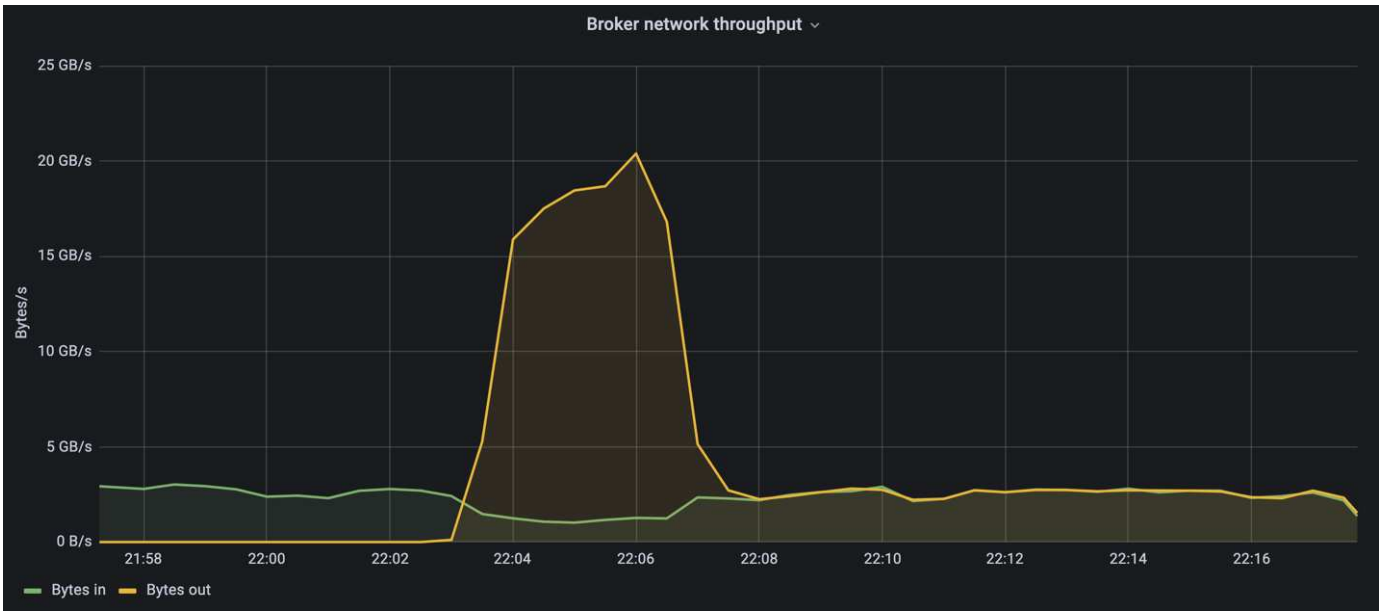


極致效能與探索儲存極限

對於AFF，我們也使用積壓功能對 OMB 進行了測試。當 Kafka 叢集中累積了大量事件積壓時，積壓功能會暫停消費者訂閱。在此階段，僅發生生產者流量，產生提交到日誌的事件。這最接近地模擬了批次或離線分析工作流程；在這些工作流程中，消費者訂閱已啟動，並且必須讀取已從代理快取中逐出的歷史資料。

為了了解此配置中儲存對消費者吞吐量的限制，我們測量了僅生產者階段，以了解 A900 可以吸收多少寫入流量。請參閱下一節“[尺寸指南] 「來了解如何利用這些數據。

在本次測量的僅生產者部分，我們看到了高峰值吞吐量，突破了 A900 效能的極限（當其他代理資源未飽和地為生產者和消費者流量提供服務時）。



我們將此測量的訊息大小增加到 16k，以限制每個訊息的開銷並最大化 NFS 掛載點的儲存吞吐量。

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

Confluent Kafka 叢集實現了 4.03GBps 的峰值生產者吞吐量。

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

OMB 完成填入事件積壓日誌後，消費者流量重新啟動。在積壓工作消耗的測量過程中，我們觀察到所有主題的峰值消費者吞吐量超過 20GBps。儲存 OMB 日誌資料的 NFS 磁碟區的組合吞吐量接近 ~30GBps。

尺寸指南

亞馬遜網路服務提供 "尺寸指南"用於 Kafka 叢集大小調整和擴展。

此大小提供了一個有用的公式來確定 Kafka 叢集的儲存吞吐量需求：

對於複製因子為 r 的 tcluster 叢集產生的聚合吞吐量，代理儲存收到的吞吐量如下：

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

這可以進一步簡化：

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

使用此公式，您可以根據 Kafka 熱層需求選擇合適的ONTAP平台。

下表解釋了具有不同複製因子的 A900 的預期生產者吞吐量：

複製因子	生產者吞吐量 **(GPps)**
3（測量）	3.4
2	5.1
1	10.2

結論

NetApp針對愚蠢重命名問題的解決方案為先前與 NFS 不相容的工作負載提供了一種簡單、廉價且集中管理的儲存形式。

這種新模式使客戶能夠創建更易於管理的 Kafka 集群，這些集群更易於遷移和鏡像，以實現災難復原和資料保護。我們還看到 NFS 提供了其他好處，例如降低 CPU 使用率和加快恢復時間、顯著提高儲存效率以及透

過NetApp ONTAP實現更好的效能。

在哪裡可以找到更多信息

要了解有關本文檔中描述的信息的更多信息，請查看以下文檔和/或網站：

- 什麼是 Apache Kafka？

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

- 什麼是愚蠢的重命名？

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

- ONATP 用於串流應用程式。

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

- NetApp產品文檔

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

- 什麼是 NFS？

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

- 什麼是 Kafka 分區重新分配？

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

- 什麼是 OpenMessaging 基準？

  "https://openmessaging.cloud/"

- 如何遷移 Kafka 代理？

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

- 如何使用 Prometheus 監控 Kafka 代理程式？

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

- Apache Kafka 託管平台

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

- 支援 Apache Kafka

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

- Apache Kafka 諮詢服務

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