



NetApp NFSストレージを使用したApache Kafkaワークロード

NetApp Solutions

NetApp
April 10, 2024

目次

NetApp NFSストレージを使用したApache Kafkaワークロード	1
TR-4947 : 『Apache Kafka workload with NetApp NFS storage - Functional Validation and performance』	1
NetApp解決策 for silly rename問題 for NFS to Kafka workloads.	2
機能検証- silly rename fix	3
ネットアップのNFSがKafkaワークロードに最適な理由	8
AWSでのパフォーマンスの概要と検証	20
AWS FSx for NetApp ONTAPでのパフォーマンスの概要と検証	33
オンプレミスのAFF A900によるパフォーマンスの概要と検証	41
まとめ	48
追加情報の参照先	48

NetApp NFSストレージを使用したApache Kafkaワークロード

TR-4947 : 『Apache Kafka workload with NetApp NFS storage - Functional Validation and performance』

ネットアップShantanu Chakole、Karthikeyan Nagalingam、Joe Scott

Kafkaは、大量のメッセージデータを受け入れることができる堅牢なキューを備えた分散型パブリッシュサブスクライブメッセージングシステムです。Kafkaを使用すると、アプリケーションは非常に高速な方法でトピックへのデータの書き込みと読み取りを行うことができます。フォールトトレランスと拡張性を備えているため、多くのデータストリームを迅速に取り込み、移動するための信頼性の高い方法として、ビッグデータ分野でよく使用されています。ユースケースには、ストリーム処理、Webサイトアクティビティの追跡、指標の収集と監視、ログの集約、リアルタイム分析などがあります。

NFSでの通常のKafka操作は問題なく機能しますが "[変な名前だな](#)" NFSで実行されているKafkaクラスタのサイズ変更中または再パーティション中に、問題 がアプリケーションをクラッシュさせます。これは、負荷分散やメンテナンスのためにKafkaクラスタのサイズを変更または再パーティションする必要があるため、重要な問題 です。詳細については、こちらを参照してください "[こちらをご覧ください](#)"。

このドキュメントでは、次の項目について説明します。

- silly-renameの問題と解決策 の検証
- CPU使用率を削減してI/O待機時間を短縮します
- Kafkaブローカーのリカバリ時間の短縮
- クラウドとオンプレミスでのパフォーマンス

KafkaワークロードにNFSストレージを使用する理由

本番アプリケーションのKafkaワークロードでは、アプリケーション間で大量のデータをストリーミングすることができます。このデータは、Kafkaクラスタ内のKafkaブローカーノードに保持され、格納されます。Kafkaは可用性と並列処理でも知られています。Kafkaは、トピックをパーティションに分割し、それらのパーティションをクラスタ全体に複製することで実現します。これは、最終的には、Kafkaクラスタを通過する膨大な量のデータのサイズが一般的に倍増されることを意味します。NFSを使用すると、ブローカーの数が非常に迅速かつ簡単に変更されるため、データのリバランシングが可能になります。大規模な環境では、ブローカーの数が変更されたときにDAS間でデータをリバランシングするのは非常に時間がかかり、ほとんどのKafka環境ではブローカーの数が頻繁に変更されます。

その他にも、次のようなメリットがあります。

- 成熟度。 NFSは成熟したプロトコルであり、実装、セキュリティ保護、使用のほとんどの側面がよく理解されています。
- オープン。 NFSはオープンプロトコルであり、その継続的な開発はインターネット仕様で無料でオープンなネットワークプロトコルとして文書化されています。

- コスト効率に優れています。NFSは、ネットワークファイル共有用の低コストの解決策であり、既存のネットワークインフラストラクチャを使用しているため、セットアップが簡単です。
- 一元管理 NFSの一元管理により'個のユーザー・システムでソフトウェアやディスク・スペースを追加する必要がなくなります
- 分散。NFSを分散ファイルシステムとして使用できるため、リムーバブルメディアストレージデバイスの必要性が軽減されます。

ネットアップがKafkaワークロードに最適な理由

ネットアップのNFS実装はプロトコルのゴールドスタンダードとみなされ、無数のエンタープライズNAS環境で使用されています。ネットアップの信頼性に加えて、次のようなメリットもあります。

- 信頼性と効率性
- 拡張性とパフォーマンス
- ハイアベイラビリティ（NetApp ONTAP クラスタ内のHAパートナー）
- データ保護
 - *ディザスタリカバリ（NetApp SnapMirror）。*サイトがダウンしたか、別のサイトから始めて中断したところから作業を続行したい。
 - ストレージシステムの管理性（NetApp OnCommand を使用した管理）。
 - *ロードバランシング。*クラスタでは、異なるノードでホストされているデータLIFから異なるボリュームにアクセスできます。
 - ノンストップオペレーション。LIFやボリュームの移動はNFSクライアントに対して透過的です。

NetApp解決策 for silly rename問題 for NFS to Kafka workloads.

Kafkaは、基盤となるファイルシステムがPOSIXに準拠していることを前提に構築されています。たとえば、XFSやext4です。Kafkaリソースのリバランシングは、アプリケーションがまだファイルを使用している間にファイルを削除します。POSIX準拠のファイルシステムでは、リンク解除を続行できます。ただし、ファイルへのすべての参照が失われた後にのみ、ファイルが削除されます。基盤となるファイルシステムがネットワークに接続されている場合、NFSクライアントはunlink呼び出しを代行受信してワークフローを管理します。リンク解除されているファイルでは保留中のオープンがあるため、NFSクライアントはNFSサーバに名前変更要求を送信し、リンク解除されたファイルの最後のクローズ時に、名前変更されたファイルに対して削除操作を実行します。この動作は、一般にNFSのsilly renameと呼ばれ、NFSクライアントによってオーケストレーションされます。

NFSv3サーバのストレージを使用するKafkaブローカーでは、この動作が原因で問題が発生します。ただし、NFSv4.xプロトコルには、リンクされていない開いたファイルをサーバが処理できるようにすることで、この問題に対処する機能があります。このオプション機能をサポートするNFSサーバは、ファイルを開いたときに所有権機能をNFSクライアントに通知します。その後、オープン保留中の状態があるとNFSクライアントはリンク解除の管理を中止し、サーバがフローを管理できるようにします。NFSv4の仕様には実装に関するガイドラインが規定されていますが、これまでこのオプション機能をサポートする既知のNFSサーバの実装は

ありませんでした。

NFSサーバとNFSクライアントで、silly rename問題 に対処するには、次の変更が必要です。

- * NFSクライアント（Linux）への変更。*ファイルを開くときに、NFSサーバは、開いているファイルのリンクを解除する機能を示すフラグを返します。NFSクライアント側の変更により、フラグが指定された状態でNFSサーバがリンク解除を処理できるようになります。ネットアップでは、これらの変更でオープンソースのLinux NFSクライアントを更新しました。更新されたNFSクライアントがRHEL8.7およびRHEL9.1で一般提供されるようになりました。
- * NFSサーバへの変更。* NFSサーバはオープンを追跡します。既存の開いているファイルのリンク解除は、POSIXセマンティクスと一致するようにサーバによって管理されるようになりました。最後に開いていたファイルが閉じると、NFSサーバによって実際のファイルの削除が開始されるため、名前の変更が不要になります。この機能は、ONTAP NFSサーバの最新リリースであるONTAP 9.12.1で実装されています。

NFSクライアントとNFSサーバに対する上記の変更により、Kafkaはネットワーク接続型NFSストレージのすべてのメリットを安全に享受できます。

機能検証- silly rename fix

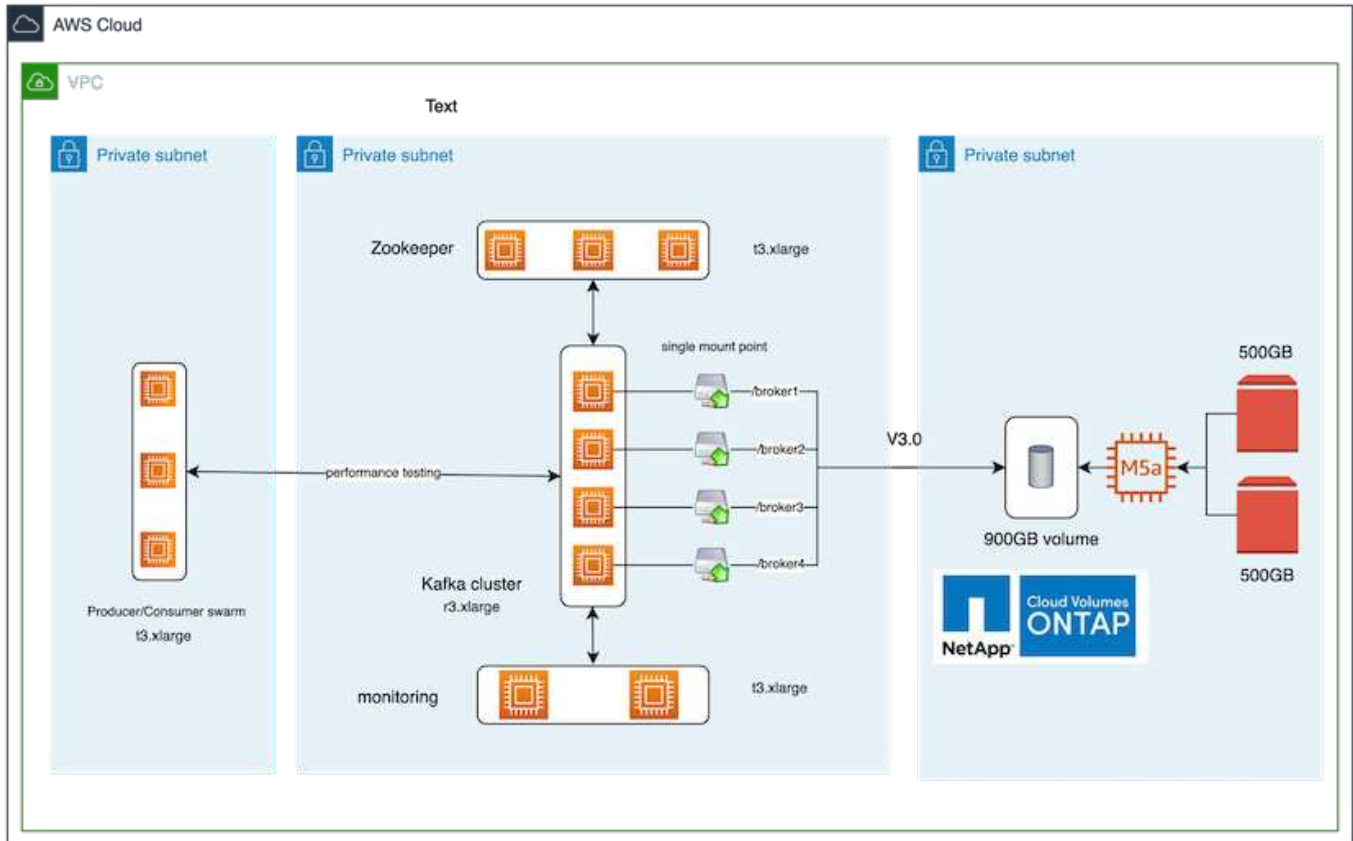
機能検証の結果、ストレージ用にNFSv3マウントを使用したKafkaクラスタではパーティションの再配置などのKafka処理を実行できませんが、修正後にNFSv4にマウントされた別のクラスタでは、システムを停止することなく同じ処理を実行できることがわかりました。

検証のセットアップ

セットアップはAWSで実行されます。次の表に、この検証で使用したプラットフォームコンポーネントと環境構成を示します。

プラットフォームコンポーネント	環境の構成
Confluent Platformバージョン7.2.1	<ul style="list-style-type: none">• 3 x動物飼育係-T3.xlarge• ブローカーサーバ×4-r3.xlarge• Grafana-T3.xlarge×1• 1 xコントロールセンター-T3.xlarge• 3 xプロデューサー/コンシューマー
すべてのノード上のオペレーティングシステム	RHEL8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス-M5.2xLarge

次の図は、この解決策 のアーキテクチャ構成を示しています。



アーキテクチャの流れ

- コンピューティング。4ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。
- 監視。Prometheus-Grafanaの組み合わせには2つのノードを使用しました。
- *ワークロード。*ワークロードの生成には、このKafkaクラスタとの間でやり取り可能な3ノードクラスタを使用しました。
- *ストレージ。*シングルノードのNetApp Cloud Volumes ONTAP インスタンスを使用し、2つの500GB gp2 AWS-EBSボリュームをインスタンスに接続しました。これらのボリュームは、LIFを介して単一のNFSv4.1ボリュームとしてKafkaクラスタに公開されました。

Kafkaのデフォルトプロパティは、すべてのサーバーに対して選択されています。動物園の群れについても同じことが行われました。

テストの方法論

1. 更新 `-is-preserve-unlink-enabled true` 次のようにKafkaボリュームに追加します。

```
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. 似たような2つのKafkaクラスタが作成されましたが、次の違いがあります。

- *クラスタ1。*本番環境対応のONTAP バージョン9.12.1を実行しているバックエンドNFS v4.1サーバは、NetApp CVOインスタンスによってホストされていました。ブローカーにRHEL 8.7 / RHEL 9.1をインストールしました。
- *クラスタ2。*バックエンドNFSサーバは、手動で作成した汎用Linux NFSv3サーバです。

3. 両方のKafkaクラスタでデモトピックを作成しました。

クラスタ1：

```
[root@ip-172-30-0-160 demo]# kafka-topics --bootstrap-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,172.30.0.123:9092 --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:
```

クラスタ2：

```
[root@ip-172-30-0-198 demo]# kafka-topics --bootstrap-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:9092,172.30.0.204:9092 --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. 両方のクラスタについて、新しく作成されたこれらのトピックにデータがロードされました。これには、デフォルトのKafkaパッケージに含まれているproducer-perf-testツールキットを使用しました。

```
./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. telnetを使用して、各クラスタのbroker-1の健全性チェックを実行しました。

- Telnet 172.30.0.160 9092
- Telnet 172.30.0.198 9092

次のスクリーンショットには、両方のクラスタのブローカーの健全性チェックが成功したことが示されています。

```
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. NFSv3ストレージボリュームを使用するKafkaクラスタがクラッシュする障害状態をトリガーするために、両方のクラスタでパーティションの再割り当てプロセスを開始しました。パーティションの再割り当てはを使用して実行されました `kafka-reassign-partitions.sh`。詳細なプロセスは次のとおりです。

- a. Kafkaクラスタでトピックのパーティションを再割り当てするために、提案された再割り当て構成JSONを生成しました（これは両方のクラスタで実行しました）。

```
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. 生成された再割り当てJSONがに保存されました `/tmp/reassignment- file.json`。
c. 実際のパーティション再割り当てプロセスは、次のコマンドによって開始されました。

```
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. 再割り当てが完了してから数分後、ブローカーで別の健全性チェックを実行したところ、NFSv3ストレージボリュームを使用するクラスタで誤った名前の問題が発生してクラッシュしたことがわかりました。一方、クラスタ1では、NetApp ONTAP NFSv4.1ストレージボリュームを使用しています。この問題は修正され、システムが停止することなく継続的に処理


```
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がアクティブです。
- cluster2-broker-1は停止しています。

8. Kafkaログディレクトリを確認したところ、修正済みのNetApp ONTAP NFSv4.1ストレージボリュームを使用しているクラスタ1ではパーティションがクリーンに割り当てられているのに対し、汎用のNFSv3ストレージを使用しているクラスタ2では異常な名前変更の問題が原因でクラッシュが発生したことがわかりました。次の図は、クラスタ2のパーティションのリバランシングを示しています。これにより、NFSv3ストレージで問題 名が誤って変更されました。

```
/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 .nfs0000000025f90084000000045
-rw-r--r--.  1 nobody nobody    0 Sep 19 10:25 .nfs0000000025f91d68000000048

/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 .nfs0000000025f91d55000000046
-rw-r--r--.  1 nobody nobody    0 Sep 19 10:25 .nfs0000000025f91fce000000047
-rw-r--r--.  1 nobody nobody 10485760 Sep 19 10:24 0000000000000000000000000000.index
-rw-r--r--.  1 nobody nobody 848113134 Sep 19 10:24 0000000000000000000000000000.log
-rw-r--r--.  1 nobody nobody 10485756 Sep 19 10:24 0000000000000000000000000000.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
```

次の図は、NetApp NFSv4.1ストレージを使用したクラスタ1のパーティションのクリーンなリバランシングを示しています。

```

/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

```

ネットアップのNFSがKafkaワークロードに最適な理由

Kafkaを使用するNFSストレージの問題 名を変更するための解決策 が追加されたので、KafkaワークロードにNetApp ONTAP ストレージを活用する堅牢な環境を構築できます。これにより、運用オーバーヘッドが大幅に削減されるだけでなく、Kafkaクラスタに次のメリットがもたらされます。

- * KafkaブローカーのCPU利用率を削減*分離されたネットアップONTAP ストレージを使用すると、ディスクI/O処理がブローカーから分離されるため、CPUフットプリントが削減されます。
- *ブローカーのリカバリ時間が短縮されます。*分離型のNetApp ONTAP ストレージはKafkaブローカーのノード間で共有されるため、データを再構築することなく、従来のKafka環境と比較して、任意の時点で新しいコンピューティングインスタンスを使用して不良ブローカーに置き換えることができます。
- * Storage Efficiency。*アプリケーションのストレージレイヤがNetApp ONTAP でプロビジョニングされるようになったため、インラインデータ圧縮、重複排除、コンパクションなど、ONTAP に備わっているStorage Efficiencyのメリットをすべて活用できます。

これらの利点は、このセクションで詳しく説明するテストケースでテストおよび検証されました。

KafkaブローカーのCPU使用率の低下

技術仕様は同一だがストレージ技術が異なる2つの精子Kafkaクラスタで同様のワークロードを実行した場合、全体的なCPU利用率がDASに比べて低いことがわかりました。KafkaクラスタがONTAP ストレージを使用している場合、全体的なCPU利用率は低くなるだけでなく、CPU利用率の増加はDASベースのKafkaクラスタよりも緩やかな勾配を示しています。

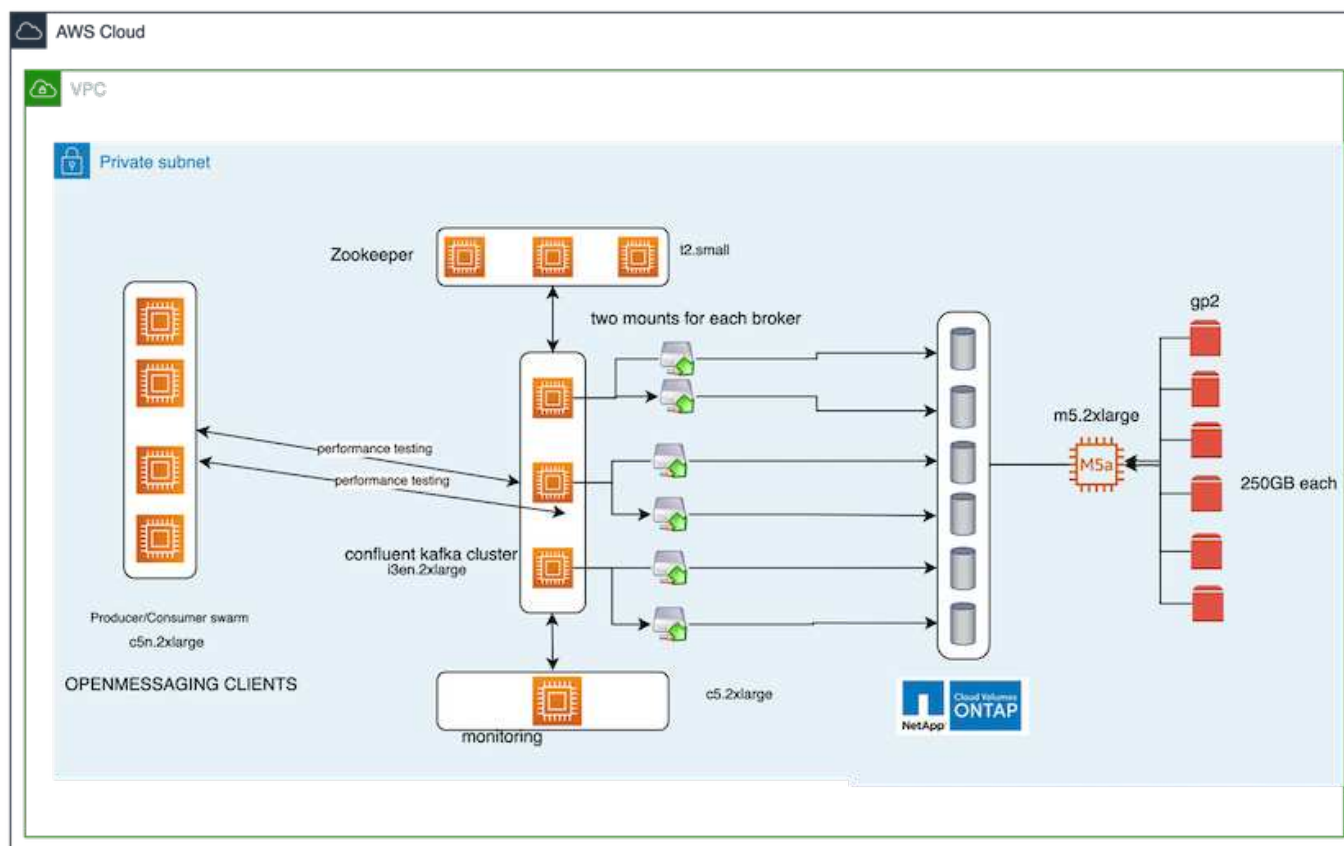
アーキテクチャのセットアップ

次の表に、CPU利用率の低下を実証するために使用する環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3ベンチマークツール: OpenMessaging	<ul style="list-style-type: none"> • 3 x動物飼育係-T2.small • ブローカーサーバ×3-i3en.2xlarge • Grafana-c5n.2xlarge×1 • 4 xプロデューサー/コンシューマー-- c5n.2xlarge
すべてのノード上のオペレーティングシステム	RHEL 8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス-M5.2xLarge

ベンチマークツール

このテストケースで使用されているベンチマークツールはです ["OpenMessagingの略"](#) フレームワーク：OpenMessagingは、ベンダーに依存せず、言語に依存しません。金融、eコマース、IoT、ビッグデータに関する業界ガイドラインを提供し、異種システムやプラットフォーム間でメッセージングやストリーミングアプリケーションを開発するのに役立ちます。次の図は、OpenMessagingクライアントとKafkaクラスタの相互作用を示しています。



- コンピューティング。3ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。各ブローカーには、専用のLIFを介してNetApp CVOインスタンス上の単一のボリュームへのNFSv4.1マウントポイントが2つありました。
- 監視。Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成については、このKafkaクラスタとの間でデータを生成したり消費したりできる3ノードクラスタを別途用意しています。

- *ストレージ。*シングルノードのNetApp Cloud Volumes ONTAP インスタンスを使用し、250GB gp2 AWS-EBSボリュームを6個マウントしました。その後、これらのボリュームは、専用のLIFを介して6つのNFSv4.1ボリュームとしてKafkaクラスタに公開されました。
- *設定。*このテストケースで設定可能な2つの要素は、KafkaブローカーとOpenMessagingワークロードでした。
 - ブローカー設定 Kafkaブローカーには以下の仕様が選択されています。以下に示すように、すべての測定値にレプリケーション係数3を使用しました。

```
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ベンチマーク（OMB）のワークロード構成。*次の仕様が提供されました。以下で強調されている目標生産者率を指定しました。

```
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
```

テストの方法論

1. 2つの類似したクラスタが作成され、それぞれに独自のベンチマーククラスタ群があります。

- クラスタ1。 NFSベースのKafkaクラスタ。
- クラスタ2。 DASベースのKafkaクラスタ。

2. OpenMessagingコマンドを使用すると、各クラスタで同様のワークロードがトリガーされます。

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

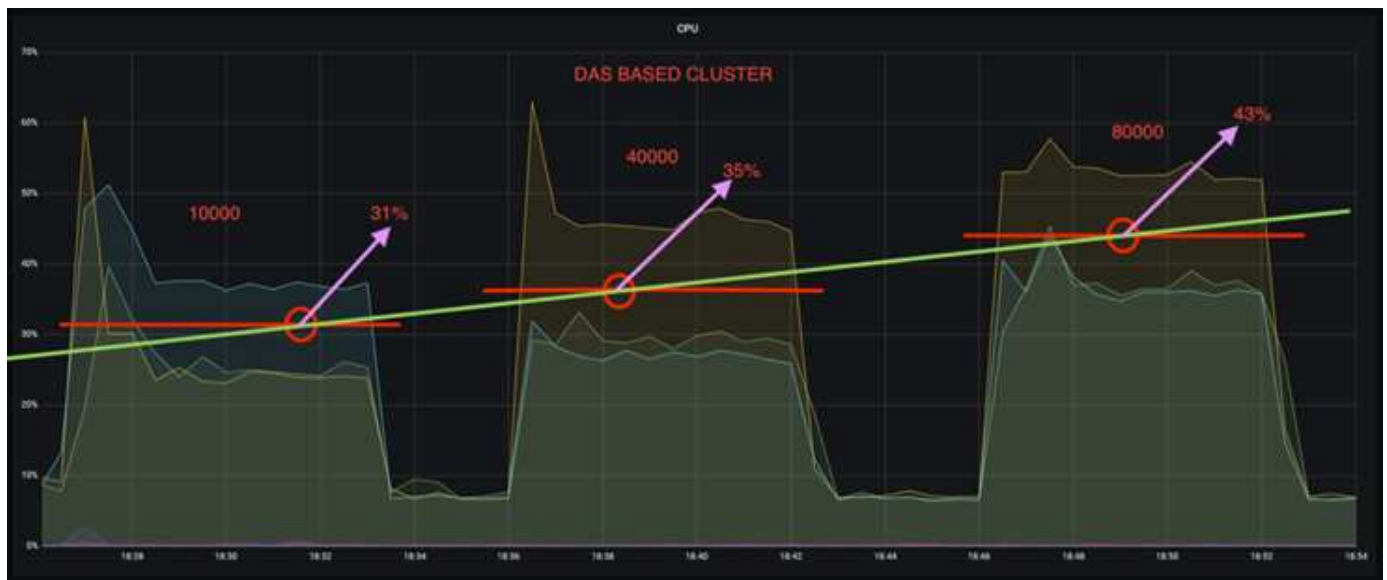
3. プロデュースレートの設定は4回の繰り返して増加し、CPU使用率はGrafanaで記録されました。生産率は次のレベルに設定されました。

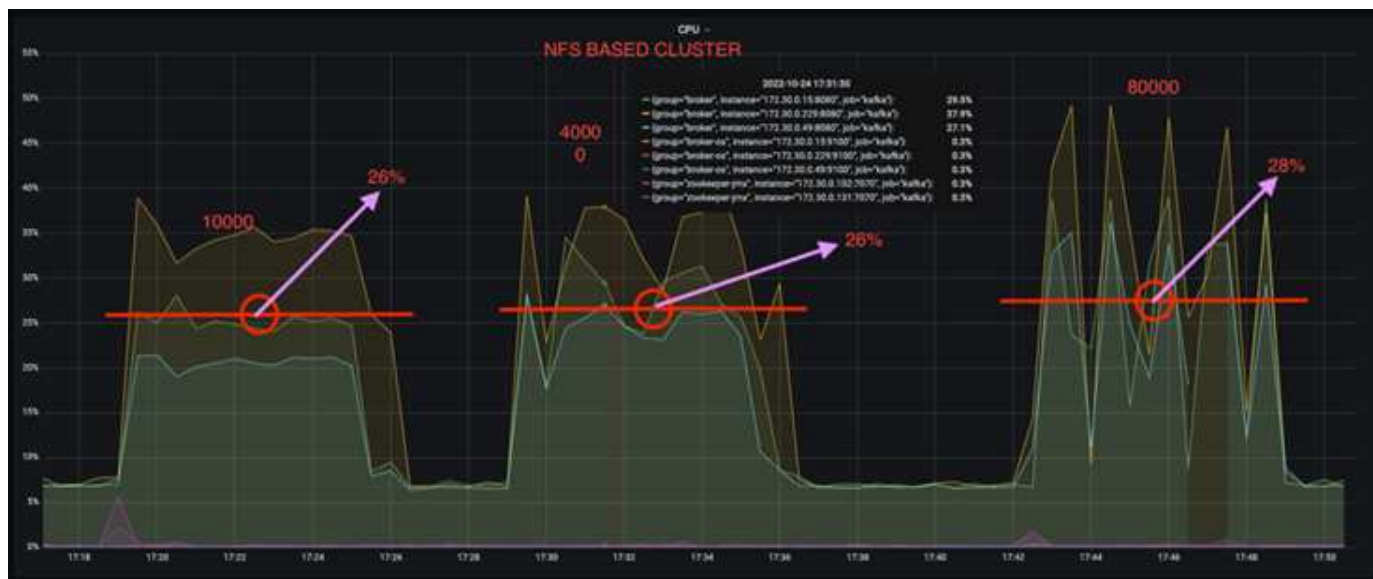
- 1万だ
- 40,000
- 8万だ
- 10万だ

観察

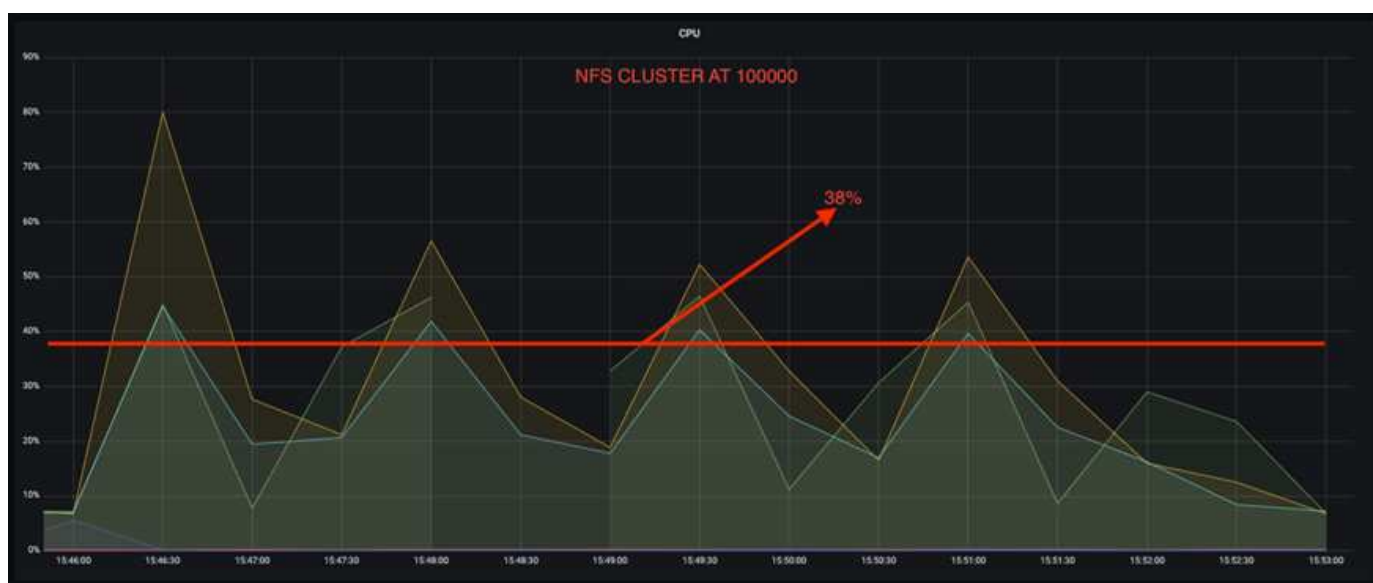
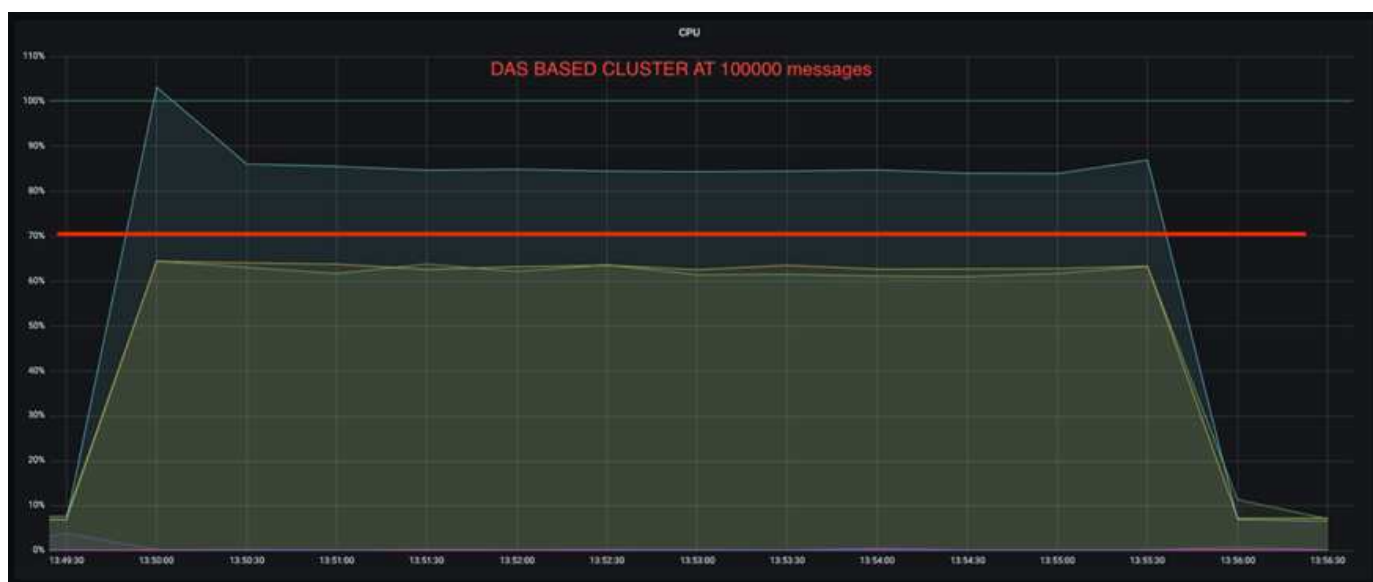
KafkaでNetApp NFSストレージを使用する主なメリットは2つあります。

- * CPU使用率をほぼ3分の1に削減できます。*同様のワークロードでの全体的なCPU使用率は、DAS SSDと比較してNFSでは低く、削減率は低い場合は5%、高い場合は32%です。
- ※プロデュース率が高い場合、CPU使用率のドリフトが3倍に減少※プロデュース率の上昇に伴い、CPU使用率の上昇は予想通り上昇しました。しかし、DASを使用するKafkaブローカーのCPU使用率は、低い生産率では31%から高い生産率では70%に上昇し、39%の増加となりました。一方、NFSストレージバックエンドでは、CPU利用率が26%から38%に上昇し、12%も上昇しました。





また、メッセージ数が100,000の場合、DASのCPU利用率はNFSクラスタよりも高くなります。



ブローカーの迅速なリカバリ

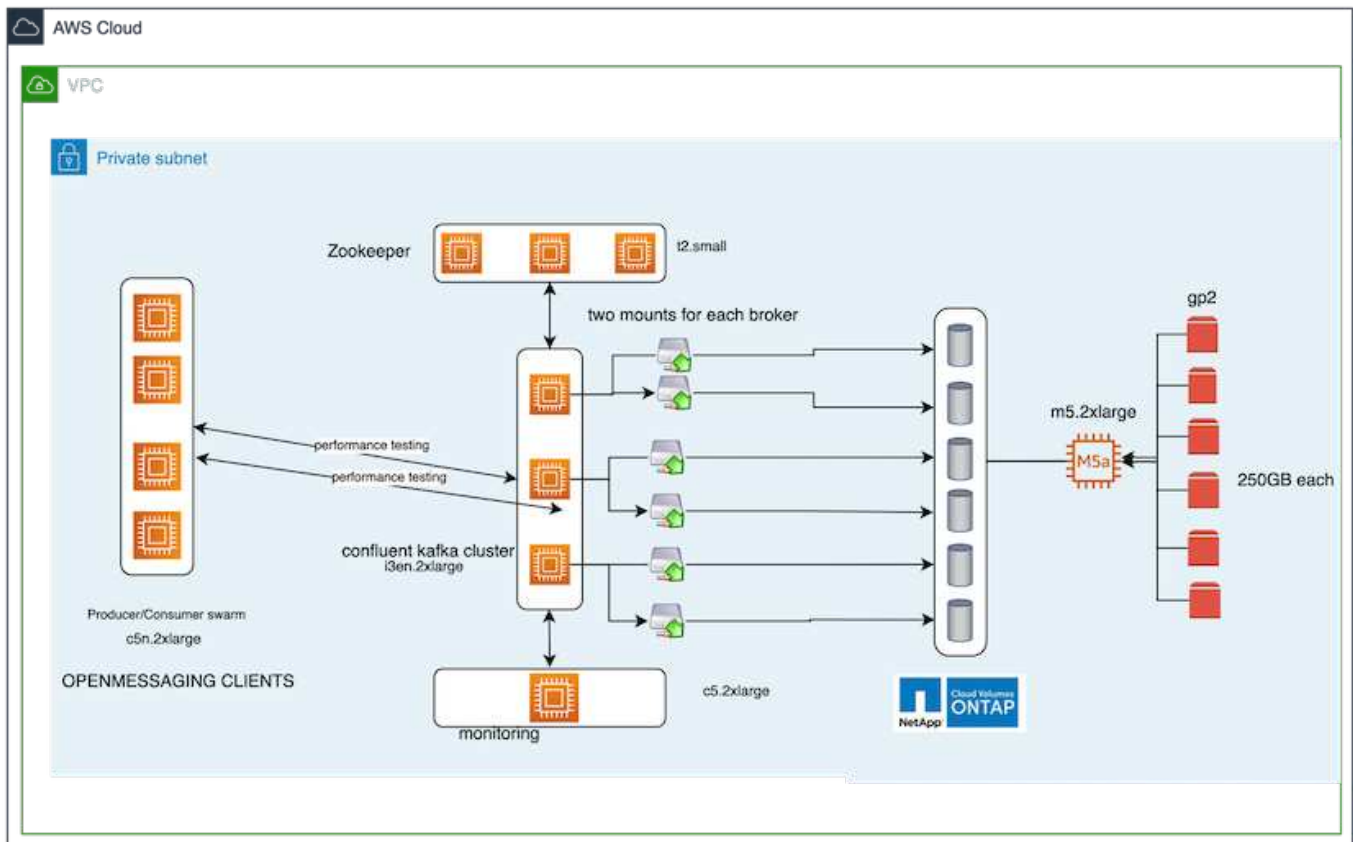
ネットアップの共有NFSストレージを使用すると、Kafkaブローカーのリカバリ時間が短縮されることがわかりました。Kafkaクラスタでブローカーがクラッシュした場合、このブローカーは同じブローカーIDを持つ正常なブローカーに置き換えることができます。このテストケースを実行したところ、DASベースのKafkaクラスタでは、新しく追加された正常なブローカーにデータが再構築されるため、時間がかかることがわかりました。NetApp NFSベースのKafkaクラスタの場合、交換後のブローカーは以前のログディレクトリから引き続きデータを読み取り、はるかに高速にリカバリします。

アーキテクチャのセットアップ

次の表に、NASを使用するKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	<ul style="list-style-type: none">• 3 x動物飼育係-T2.small• ブローカーサーバ×3-i3en.2xlarge• Grafana-c5n.2xlarge×1• 4 x producer/consumer — c5n.2xlarge• 1 xバックアップKafkaノード-i3en.2xlarge
すべてのノード上のオペレーティングシステム	RHEL8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス-M5.2xLarge

次の図は、NASベースのKafkaクラスタのアーキテクチャを示しています。



- **コンピューティング。** 3ノードのZookeeperアンサンブルを専用サーバー上で実行する3ノードのKafkaクラスタ。各ブローカーには、専用のLIFを介してNetApp CVOインスタンス上の単一のボリュームへのNFSマウントポイントが2つあります。
- **監視。** PrometheusとGrafanaの組み合わせでは2ノード。ワークロードの生成には、このKafkaクラスタを生成して使用できる3ノードクラスタを別々に使用します。
- ***ストレージ。*** シングルノードのNetApp Cloud Volumes ONTAP インスタンス。250GB gp2 AWS-EBSボリュームが6個マウントされています。これらのボリュームは、専用のLIFを介して6つのNFSボリュームとしてKafkaクラスタに提供されます。
- ***ブローカーの設定*** このテストケースで設定可能な要素の1つはKafkaブローカーです。Kafkaブローカーのために以下の仕様が選択されました。。 `replica.lag.time.mx.ms` は、特定のノードがISRリストから削除される速度を決定するため、高い値に設定されます。不良ノードと正常ノードを切り替える場合、そのブローカーIDがISRリストから除外されないようにします。

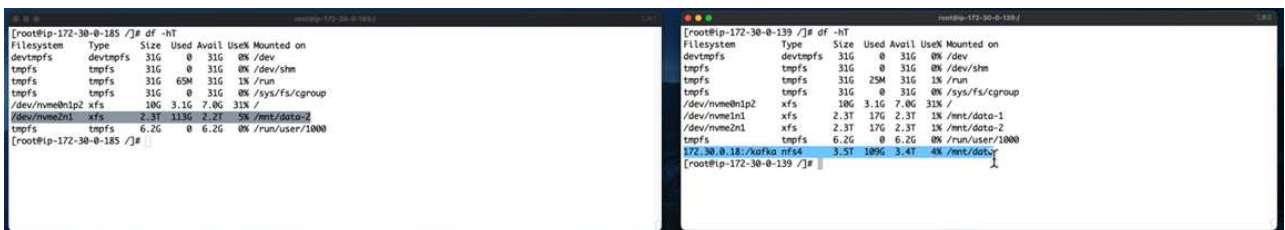

```

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

```

テストの方法論

- 同様の2つのクラスタが作成されました。
 - EC2ベースのコンフルエントクラスタ。
 - NetApp NFSベースのコンフルエントクラスタ。
- 1つのスタンバイKafkaノードが、元のKafkaクラスタのノードと同じ構成で作成されました。
- 各クラスタでサンプルトピックを作成し、各ブローカーに約110GBのデータが読み込まれました。
 - * EC2ベースのクラスタ。* Kafkaブローカーのデータディレクトリがにマッピングされています /mnt/data-2（次の図では、cluster1のBroker-1（左側のターミナル））。
 - * NetApp NFSベースのクラスタ。* KafkaブローカーのデータディレクトリがNFSポイントにマウントされている /mnt/data（次の図では、cluster2のBroker-1（右側の端末））。



- 各クラスタで、Broker-1が終了し、ブローカーのリカバリプロセスが失敗しました。
- ブローカーが終了した後、ブローカーのIPアドレスがセカンダリIPとしてスタンバイブローカーに割り当てられました。これは、Kafkaクラスタ内のブローカーが次のように識別されるために必要でした。
 - * IPアドレス。*障害が発生したブローカーのIPをスタンバイブローカーに再割り当てすることによって割り当てられます。
 - *ブローカーID。*これはスタンバイブローカーで設定されました server.properties。
- IP割り当て時に、スタンバイブローカーでKafkaサービスが開始されました。
- しばらくすると、サーバログがプルされ、クラスタ内の交換用ノードでデータを構築するのにかった時間が確認されました。

観察

Kafkaブローカーの回復はほぼ9倍速くなりました。NetApp NFS共有ストレージを使用すると、KafkaクラスタでDAS SSDを使用する場合と比較して、障害が発生したブローカーノードのリカバリにかかる時間が大幅に短縮されることがわかりました。1TBのトピックデータの場合、DASベースのクラスタのリカバリ時間は48分でしたが、NetApp-NFSベースのKafkaクラスタのリカバリ時間は5分未満でした。

EC2ベースのクラスタで110GBのデータを新しいブローカーノードにリビルドするのに10分かかったのに対し、NFSベースのクラスタでは3分でリカバリが完了しました。また、ログでは、EC2のパーティションのコンシューマオフセットが0であり、NFSクラスタではコンシューマオフセットが前のブローカーから取得されていることがわかりました。

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

DASベースのクラスタ

1. バックアップノードは08:55:53、730に開始されました。

```
2 [2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true (org.apache.kafka.common.config.KafkaConfig$)
3 [2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.config.KafkaConfig$)
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 172.30.0.17:2181,172.30.0.18:2181 (kafka.zookeeper.ZooKeeperClient)
6 [2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new session (kafka.zookeeper.ZooKeeperClient)
```

2. データの再構築プロセスは09:05:24,860に終了しました。110GBのデータの処理には約10分かかります。

```
[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)
```

NFSベースのクラスタ

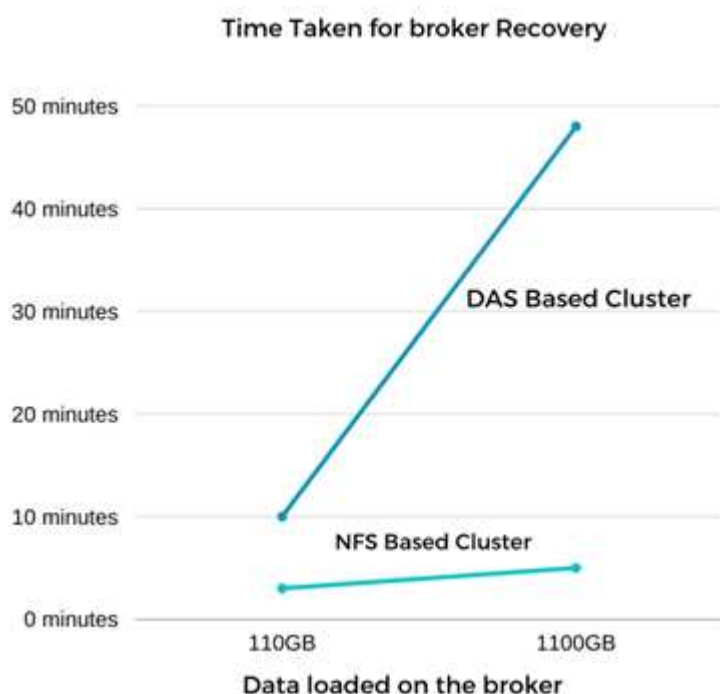
1. バックアップノードは09:39:17、213に開始されました。開始ログエントリは以下のように強調表示されます。

```
2 [2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true (org.apache.kafka.common.config.KafkaConfig$)
3 [2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.config.KafkaConfig$)
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 172.30.0.22:2181,172.30.0.23:2181 (kafka.zookeeper.ZooKeeperClient)
6 [2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new session (kafka.zookeeper.ZooKeeperClient)
7 [2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a (org.apache.zookeeper.ZooKeeperClient)
8 [2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in (org.apache.zookeeper.ZooKeeperClient)
9 [2022-10-31 09:39:17,244] INFO Client environment:java.version=11.0.17 (org.apache.zookeeper.ZooKeeperClient)
```

2. データの再構築プロセスは09:42:29,115に終了しました。110GBのデータの処理には約3分かかります。

```
[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)
```

このテストを、約1TBのデータを含むブローカーに対して繰り返しました。DASでは約48分、NFSでは約3分かかりました。結果を次のグラフに示します。



ストレージ効率

KafkaクラスタのストレージレイヤはNetApp ONTAP を介してプロビジョニングされていたため、ONTAP のすべてのStorage Efficiency機能を利用できました。このテストでは、Cloud Volumes ONTAP でNFSストレージをプロビジョニングしたKafkaクラスタで大量のデータを生成しました。ONTAP 機能により、スペースが大幅に削減されたことがわかりました。

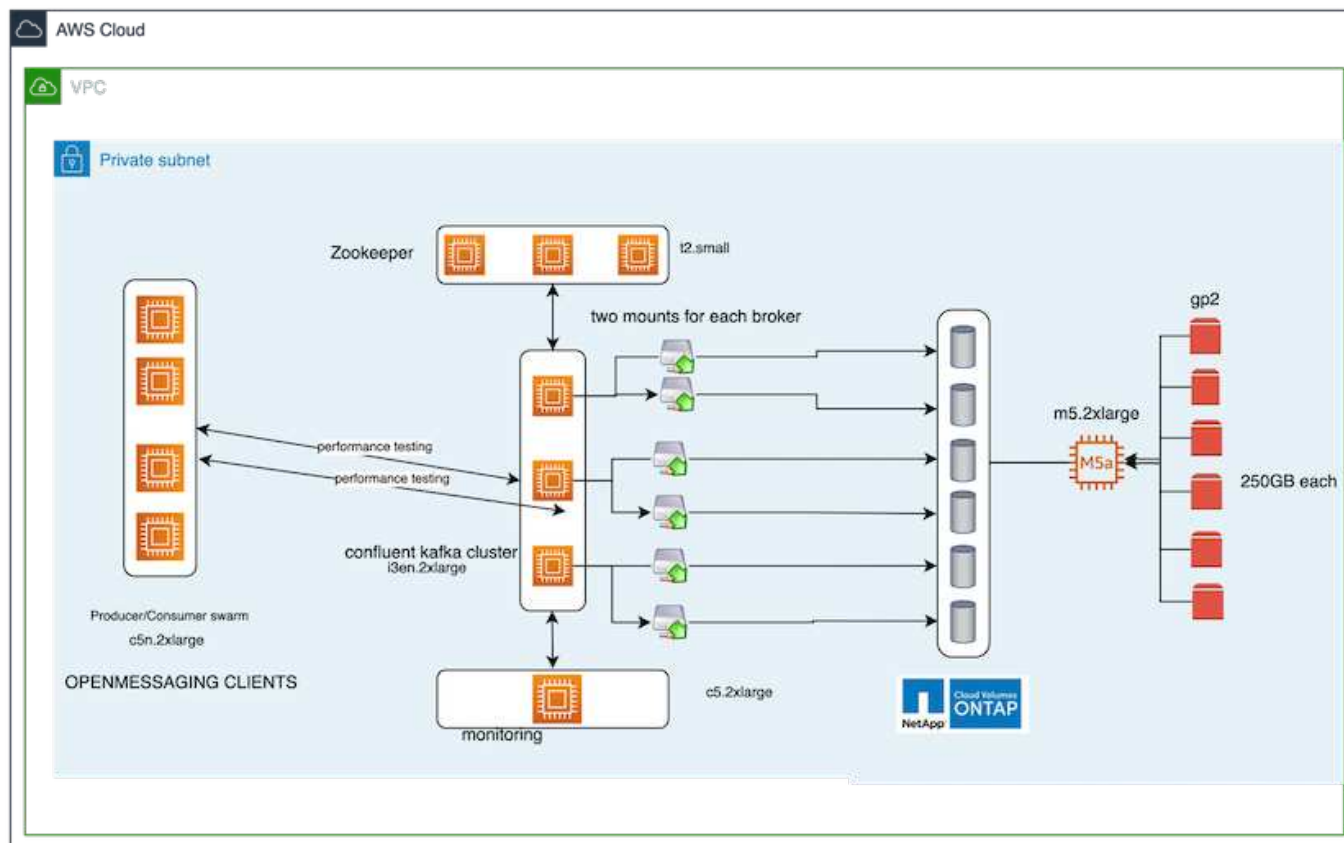
アーキテクチャのセットアップ

次の表に、NASを使用するKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	<ul style="list-style-type: none">• 3 x動物飼育係-T2.small• ブローカーサーバ×3-i3en.2xlarge• Grafana-c5n.2xlarge×1• 4 x producer/consumer — c5n.2xlarge *

プラットフォームコンポーネント	環境の構成
すべてのノード上のオペレーティングシステム	RHEL8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス-M5.2xLarge

次の図は、NASベースのKafkaクラスタのアーキテクチャを示しています。



- **コンピューティング。** 3ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。各ブローカーには、専用のLIFを介してNetApp CVOインスタンス上の単一のボリュームへのNFSマウントポイントが2つありました。
- **監視。** Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成には、独立した3ノードクラスタを使用し、このKafkaクラスタを生成して使用しました。
- ***ストレージ。*** シングルノードのNetApp Cloud Volumes ONTAP インスタンスを使用し、250GB gp2 AWS-EBSボリュームを6個マウントしました。その後、これらのボリュームは、専用のLIFを介して6つのNFSボリュームとしてKafkaクラスタに公開されました。
- ***構成*** このテストケースの構成要素はKafkaブローカーです。

プロデューサー側で圧縮がオフになっているため、プロデューサーは高いスループットを生成できません。Storage Efficiencyは、代わりにコンピューティングレイヤで処理されました。

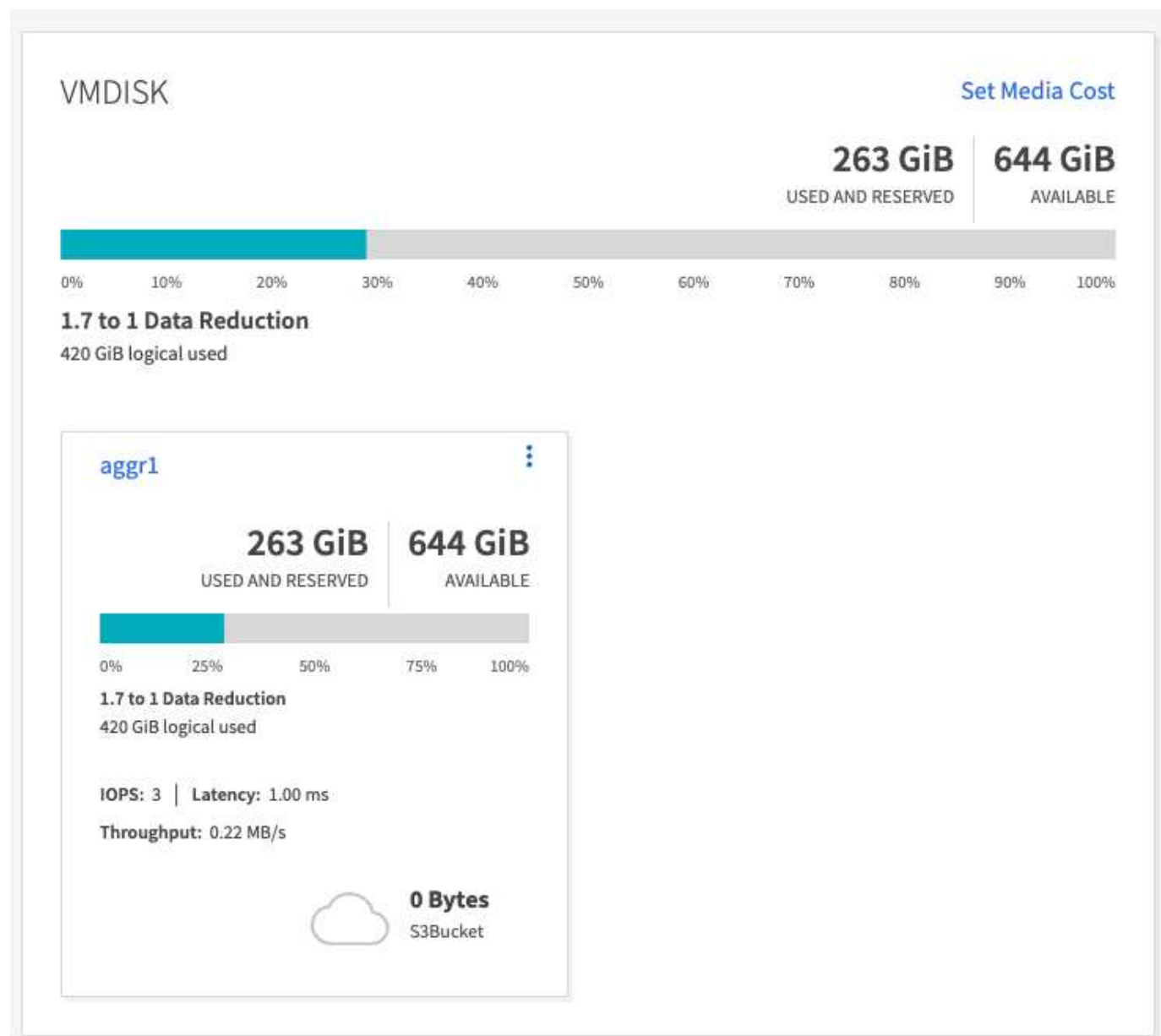
テストの方法論

1. 上記の仕様でKafkaクラスタがプロビジョニングされました。
2. クラスタでは、OpenMessaging Benchmarkingツールを使用して約350GBのデータが生成されました。

3. ワークロードの完了後、ONTAP System ManagerとCLIを使用してStorage Efficiencyの統計を収集しました。

観察

OMBツールを使用して生成したデータでは、ストレージ容量削減比率が1.70：1で約33%削減されました。次の図に示すように、生成されたデータに使用された論理スペースは420.3GB、データの保持に使用された物理スペースは281.7GBです。



```
shantanuCV0instancenew:> 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%	shantanuCV0instancenew-01
/vol/kafka_vol/	281GB	138GB	33%	138GB	33%	0B	0%	svm_shantanuCV0instancenew
/vol/svm_shantanuCV0instancenew_root/	660KB	0B	0%	0B	0%	0B	0%	svm_shantanuCV0instancenew

3 entries were displayed.

```

Name of the Aggregate: aggr1
Node where Aggregate Resides: shantanuCV0instancenew-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でのパフォーマンスの概要と検証

ストレージレイヤをNetApp NFS上にマウントしたKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークテストを実施しました。ベンチマークの例については、次のセクションで説明します。

AWSクラウドのKafkaとNetApp Cloud Volumes ONTAP（ハイアベイラビリティペアとシングルノード）

NetApp Cloud Volumes ONTAP（HAペア）を使用したKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークを実施しました。このベンチマークについては、以降のセクションで説明します。

アーキテクチャのセットアップ

次の表に、NASを使用するKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	<ul style="list-style-type: none"> • 3 x動物飼育係-T2.small • ブローカーサーバ×3-i3en.2xlarge • Grafana-c5n.2xlarge×1 • 4 x producer/consumer — c5n.2xlarge *
すべてのノード上のオペレーティングシステム	RHEL8.6
NetApp Cloud Volumes ONTAP インスタンス	HAペアインスタンス-m5dn.12xLarge x 2ノード シングルノードインスタンス- m5dn.12xLarge x 1ノード

ネットアップクラスタボリュームのONTAP セットアップ

1. Cloud Volumes ONTAP HAペアについては、各ストレージコントローラの各アグリゲートに3つのボリュームを含む2つのアグリゲートを作成しました。単一のCloud Volumes ONTAP ノードの場合は、アグリゲートに6つのボリュームを作成します。

aggr3

EBS Allocated Capacity: 5.05 TB

EBS Used Capacity: 298.21 GB

Volumes: 3

kafka_aggr3_vol1 (1 TB)
kafka_aggr3_vol2 (1 TB)
kafka_aggr3_vol3 (1 TB)

AWS Disks: 8

State: online

Underlying AWS Tier: Provisioned IOPS SSD (io1)

Close

AWS Disk Size: 2 TB

Underlying AWS Capacity: 12 TB

Encryption Type:

Home Node: kafka_nfs_cvo_ha1-01

Provisioned IOPS: 80000

aggr22

EBS Allocated Capacity: 6.73 TB

EBS Used Capacity: 280.95 GB

Volumes: 3

kafka_aggr22_vol1 (1 TB)
kafka_aggr22_vol2 (1 TB)
kafka_aggr22_vol3 (1 TB)

AWS Disks: 8

State: online

Underlying AWS Tier: Provisioned IOPS SSD (io1)

Close

AWS Disk Size: 2 TB

Underlying AWS Capacity: 16 TB

Encryption Type:

Home Node: kafka_nfs_cvo_ha1-02

Provisioned IOPS: 20000

aggr2

EBS Allocated Capacity: 5.32 TB

EBS Used Capacity: 209.90 GB

Volumes: 6

kafka_aggr2_vol2 (1 TB)
kafka_aggr2_vol3 (1 TB)
kafka_aggr2_vol4 (1 TB)

AWS Disks: 4

State: online

Underlying AWS Tier: Provisioned IOPS SSD (io1)

Close

AWS Disk Size: 2 TB

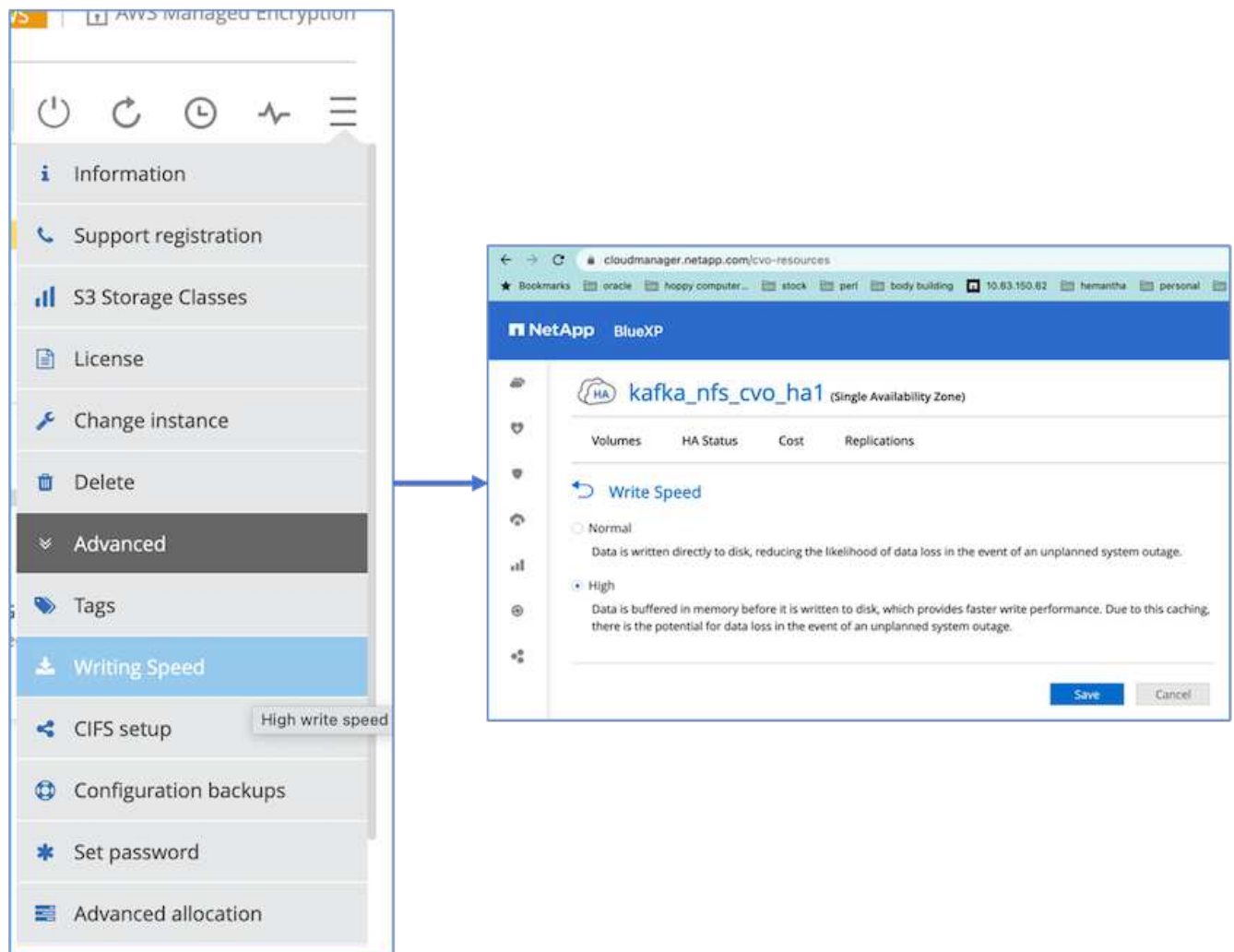
Underlying AWS Capacity: 6 TB

Encryption Type:

Home Node: kafka_nfs_cvo_sn-01

Provisioned IOPS: 80000

- ネットワークパフォーマンスを高めるために、HAペアとシングルノードの両方で高速ネットワークを有効にしました。

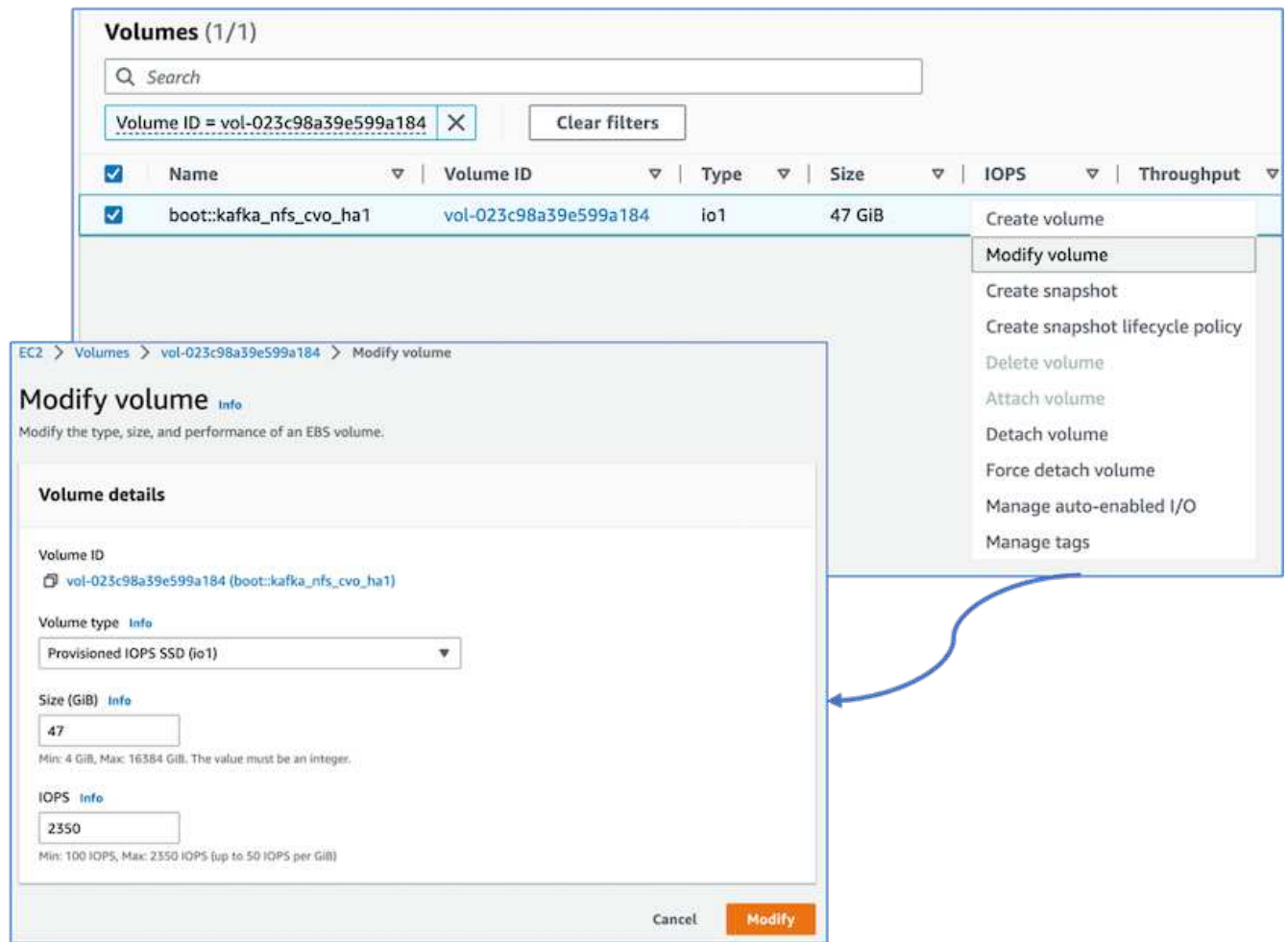


3. ONTAP NVRAMのIOPSが多いことに気付いたので、Cloud Volumes ONTAP ルートボリュームのIOPSを2350に変更しました。Cloud Volumes ONTAP のルートボリュームディスクのサイズは47GBでした。次のONTAP コマンドはHAペア用で、同じ手順をシングルノードにも適用します。


```

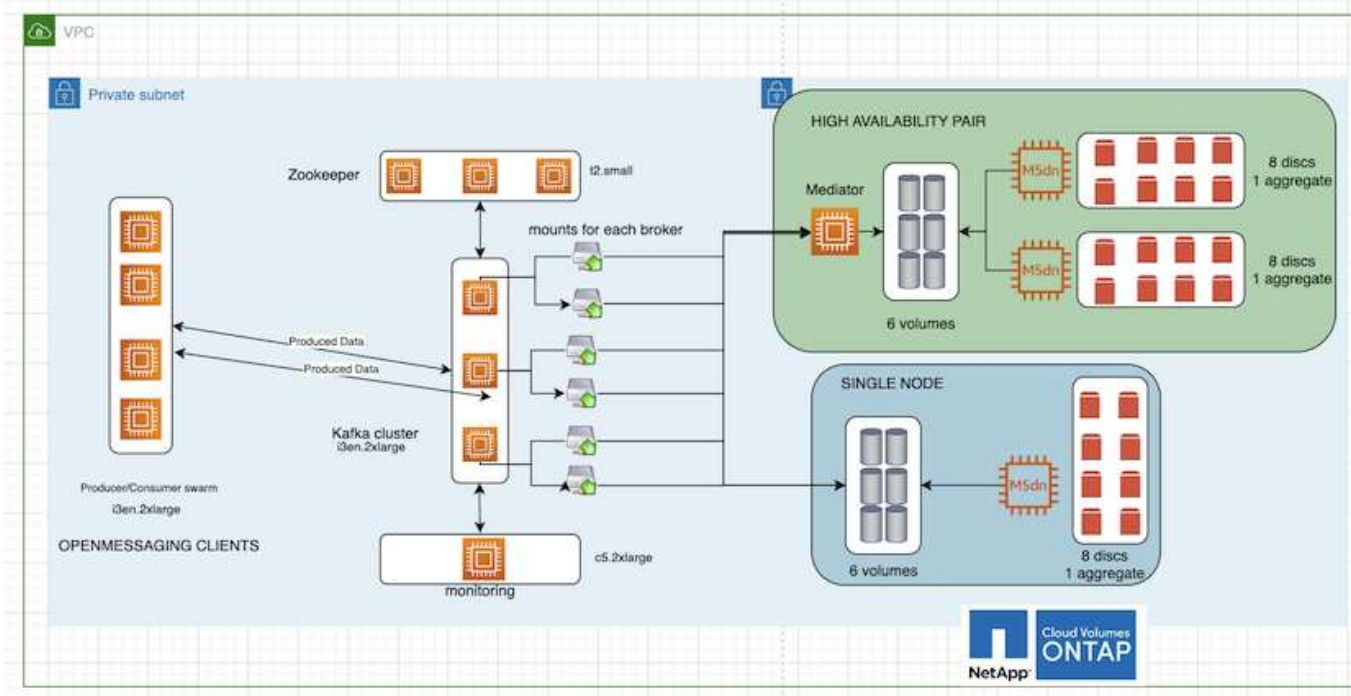
statistics start -object vnvram -instance vnvram -counter
backing_store_iops -sample-id sample_555
kafka_nfs_cvo_ha1:*> statistics show -sample-id sample_555
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-01
  Counter                                                    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:*>

```



次の図は、NASベースのKafkaクラスタのアーキテクチャを示しています。

- コンピューティング。3ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。各ブローカーには、専用のLIFを介してCloud Volumes ONTAP インスタンス上の単一のボリュームへのNFSマウントポイントが2つあります。
- 監視。Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成には、独立した3ノードクラスタを使用し、このKafkaクラスタを生成して使用しました。
- ストレージ。HAペアCloud Volumes ONTAP インスタンスを使用し、インスタンスに6TB gp3 AWS-EBS ボリュームを1つマウントしました。その後、ボリュームはNFSマウントを使用してKafkaブローカーにエクスポートされました。



OpenMessageベンチマーク設定

1. NFSのパフォーマンスを向上させるには、NFSサーバとNFSクライアントの間のネットワーク接続を増やす必要があります。この接続は、nconnectを使用して作成できます。次のコマンドを実行して、nconnectオプションを使用して、ブローカーノードにNFSボリュームをマウントします。

```
[root@ip-172-30-0-121 ~]# cat /etc/fstab
UUID=eaalf38e-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. Cloud Volumes ONTAP でネットワーク接続を確認します。次のONTAP コマンドは、単一のCloud Volumes ONTAP ノードから使用します。同じ手順をCloud Volumes ONTAP HAペアにも適用できます。

```
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 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. 以下のKafkaを使用します server.properties Cloud Volumes ONTAP HAペアのすべてのKafkaブローカー。log.dirs プロパティはブローカーごとに異なり、残りのプロパティはブローカーに共通です。broker1の場合は log.dirs 値は次のとおりです。

```
[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_aggr3_vol3/broker1,/kafka_aggr22_vol1/broker1,/kafka_aggr22_vol2/broker1,/kafka_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 ~]#
```

- 仲介業者2の場合は、log.dirs プロパティ値は次のとおりです。

```
log.dirs=/kafka_aggr3_vol1/broker2,/kafka_aggr3_vol2/broker2,/kafka_aggr3_vol3/broker2,/kafka_aggr22_vol1/broker2,/kafka_aggr22_vol2/broker2,/kafka_aggr22_vol3/broker2
```

- 仲介業者3の場合は、log.dirs プロパティ値は次のとおりです。

```
log.dirs=/kafka_aggr3_vol1/broker3,/kafka_aggr3_vol2/broker3,/kafka_aggr3_vol3/broker3,/kafka_aggr22_vol1/broker3,/kafka_aggr22_vol2/broker3,/kafka_aggr22_vol3/broker3
```

4. 単一のCloud Volumes ONTAP ノードの場合は、Kafka `servers.properties` は、を除き、Cloud Volumes ONTAP HAペアと同じです `log.dirs` プロパティ。

◦ broker1の場合は `log.dirs` 値は次のとおりです。

```
log.dirs=/kafka_aggr2_vol1/broker1,/kafka_aggr2_vol2/broker1,/kafka_aggr2_vol3/broker1,/kafka_aggr2_vol4/broker1,/kafka_aggr2_vol5/broker1,/kafka_aggr2_vol6/broker1
```

◦ 仲介業者2の場合は、 `log.dirs` 値は次のとおりです。

```
log.dirs=/kafka_aggr2_vol1/broker2,/kafka_aggr2_vol2/broker2,/kafka_aggr2_vol3/broker2,/kafka_aggr2_vol4/broker2,/kafka_aggr2_vol5/broker2,/kafka_aggr2_vol6/broker2
```

◦ 仲介業者3の場合は、 `log.dirs` プロパティ値は次のとおりです。

```
log.dirs=/kafka_aggr2_vol1/broker3,/kafka_aggr2_vol2/broker3,/kafka_aggr2_vol3/broker3,/kafka_aggr2_vol4/broker3,/kafka_aggr2_vol5/broker3,/kafka_aggr2_vol6/broker3
```

5. OMB内のワークロードには、次のプロパティが設定されます。 (`/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
```

。messageSize はユースケースごとに異なる場合があります。パフォーマンステストでは、3Kを使用しました。

OMBのSyncまたはThroughputという2つのドライバを使用して、Kafkaクラスタでワークロードを生成しました。

- 。同期ドライバのプロパティに使用されるYAMLファイルは次のとおりです
(/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=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:9092
2
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

- 。スループットドライバのプロパティに使用されるYAMLファイルは次のとおりです
(/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

```

テストの方法論

1. Kafkaクラスタは、前述の仕様に従ってTerraformとAnsibleを使用してプロビジョニングされました。Terraformを使用して、Kafkaクラスタ用のAWSインスタンスを使用してインフラを構築し、Ansibleを使用してKafkaクラスタを構築します。
2. 上記のワークロード構成とSyncドライバでOMBワークロードがトリガーされました。

```

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

```

3. 同じワークロード構成でスループットドライバを使用して別のワークロードがトリガーされました。

```

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

```

観察

NFSで実行されるKafkaインスタンスのパフォーマンスをベンチマークするために、2種類のドライバを使用してワークロードを生成しました。ドライバの違いは、log flushプロパティです。

Cloud Volumes ONTAP HAペアの場合：

- Syncドライバによって一貫して生成される合計スループット：最大1236 Mbps
- スループットドライバに対して生成された合計スループット：ピーク時最大1412 Mbps

単一のCloud Volumes ONTAP ノードの場合：

- Syncドライバで一貫して生成される合計スループット：約1962MBps
- スループットドライバによって生成される合計スループット：最大1660MBps

Syncドライバはログが即座にディスクにフラッシュされるときに一貫したスループットを生成できますが、Throughputドライバはログがディスクに一括コミットされるときにスループットのバーストを生成します。

これらのスループット値は、指定されたAWS構成に対して生成されます。より高いパフォーマンス要件に対応するには、インスタンスタイプをスケールアップしてさらに調整し、スループットを向上させることができます。総スループットまたは総レートは、生産者と消費者の両方のレートの組み合わせです。



スループットまたは同期ドライバのベンチマークを実行するときは、必ずストレージスループットを確認してください。



AWS FSx for NetApp ONTAPでのパフォーマンスの概要と検証

ストレージレイヤをNetApp NFS上にマウントしたKafkaクラスタは、AWS FSx for NetApp ONTAPでパフォーマンスのベンチマークを実施しました。ベンチマークの例については、次のセクションで説明します。

AWS FSx for NetApp ONTAPでのApache Kafkaの使用

Network File System（NFS；ネットワークファイルシステム）は、大量のデータを格納するために広く使用されているネットワークファイルシステムです。ほとんどの組織では、データがApache Kafkaなどのストリーミングアプリケーションによって生成されるようになっています。このようなワークロードには、拡張性、低レイテンシ、最新のストレージ機能を備えた堅牢なデータ取り込みアーキテクチャが必要です。リアルタイム分析を実現し、実用的な分析情報を提供するには、適切に設計されたパフォーマンスの高いインフラが必要です。

Kafkaは、設計上POSIX準拠のファイルシステムと連携し、ファイル操作をファイルシステムに依存して処理しますが、NFSv3ファイルシステムにデータを格納する場合、KafkaブローカーのNFSクライアントは、ファイル操作をXFSやext4などのローカルファイルシステムとは異なる方法で解釈できます。一般的な例としては、NFSのsilly renameがあり、クラスタの拡張時やパーティションの再割り当て時にKafkaブローカーでエラーが発生していました。この課題に対処するため、NetAppはオープンソースのLinux NFSクライアントを更新し、RHEL8.7とRHEL9.1で一般提供されるようになりました。また、最新のFSx for NetApp ONTAPリリースであるONTAP 9.12.1からサポートされるようになりました。

Amazon FSx for NetApp ONTAPは、拡張性とパフォーマンスに優れたフルマネージドのNFSファイルシステムをクラウドで提供します。FSx for NetApp ONTAPのKafkaデータは、大量のデータを処理し、フォールトトレランスを確保するように拡張できます。NFSは、重要で機密性の高いデータセットに対して一元的なストレージ管理とデータ保護を提供します。

これらの機能強化により、AWSのお客様はAWSコンピューティングサービスでKafkaワークロードを実行する際にFSx for NetApp ONTAPを活用できるようになります。次のようなメリットがあります。

- * CPU使用率を削減して、I/O待機時間を短縮します
- * Kafkaブローカーのリカバリ時間の短縮。
- * 信頼性と効率性。
- * 拡張性とパフォーマンス。
- * マルチアベイラビリティゾーンの可用性。
- * データ保護：

AWS FSx for NetApp ONTAPでのパフォーマンスの概要と検証

ストレージレイヤをNetApp NFS上にマウントしたKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークテストを実施しました。ベンチマークの例については、次のセクションで説明します。

AWS FSx for NetApp ONTAPのKafka

AWS FSx for NetApp ONTAPを使用したKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークを実施しました。このベンチマークについては、以降のセクションで説明します。

アーキテクチャのセットアップ

次の表に、AWS FSx for NetApp ONTAPを使用したKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	<ul style="list-style-type: none">• 3 x動物飼育係–T2.small• ブローカーサーバ×3–i3en.2xlarge• Grafana–c5n.2xlarge×1• 4 x producer/consumer — c5n.2xlarge *

プラットフォームコンポーネント	環境の構成
すべてのノード上のオペレーティングシステム	RHEL8.6
AWS FSx for NetApp ONTAP	マルチアストラゼネカ、4GB/秒のスループット、160000 IOPS

NetApp FSx for NetApp ONTAPのセットアップ

1. 最初のテストでは、2TBの容量と40000のIOPSで2GB/秒のスループットを実現するFSx for NetApp ONTAPファイルシステムを作成しました。

```
[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}"
```

この例では、AWS CLIを使用してFSx for NetApp ONTAPを導入しています。必要に応じて、環境内でコマンドをさらにカスタマイズする必要があります。FSx for NetApp ONTAPはAWSコンソールからさらに導入、管理できるため、コマンドラインの入力が少なく、導入がより簡単かつ合理的になります。

ドキュメントFSx for NetApp ONTAPでは、テストリージョン（US-East-1）の2GB/秒スループットファイルシステムで達成可能な最大IOPSは8万IOPSです。FSx for NetApp ONTAPファイルシステムの合計最大IOPSは16万IOPSですが、そのためには4GB/秒のスループット導入が必要です。このことについては、このドキュメントの後半で説明します。

FSx for NetApp ONTAPのパフォーマンス仕様の詳細については、AWS FSx for NetApp ONTAPのドキュメントを参照してください。 <https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html>。

FSx「create-file-system」のコマンドライン構文の詳細については、以下を参照してください。 <https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html>

たとえば、KMSキーが指定されていない場合に使用されるデフォルトのAWS FSxマスターキーとは異なり、特定のKMSキーを指定できます。

2. FSx for NetApp ONTAPファイルシステムを作成するときは、ファイルシステムを次のように定義したら、JSONで「lifecycle」ステータスが「available」に変わるまで待ちます。

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

3. fsxadminユーザを使用してFSx for NetApp ONTAP SSHにログインし、クレデンシャルを検証します。Fsxadminは、作成時にFSx for NetApp ONTAPファイルシステムのデフォルトの管理者アカウントです。fsxadminのパスワードは、手順1で完了したように、AWSコンソールまたはAWS CLIでファイルシステムを最初に作成したときに設定したパスワードです。

```
[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. クレデンシャルの検証が完了したら、FSx for NetApp ONTAPファイルシステムにSVMを作成

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

Storage Virtual Machine (SVM) は分離されたファイルサーバで、FSx for NetApp ONTAPボリューム内のデータを管理およびアクセスするための独自の管理クレデンシャルとエンドポイントを備え、FSx for NetApp ONTAPマルチテナンシーを提供します。

5. プライマリSVMの設定が完了したら、新しく作成したFSx for NetApp ONTAPファイルシステムにSSHで接続し、以下のサンプルコマンドを使用してSVMにボリュームを作成します。同様に、この検証用に6つのボリュームを作成します。Kafkaのパフォーマンスが向上するように、デフォルトのコンスティチュエント（8個）以下のコンスティチュエントを使用してください。

```
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. テスト用にボリュームに追加の容量が必要になります。ボリュームのサイズを2TBに拡張し、ジャンクションパスにマウントします。

```
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
```

FSx for NetApp ONTAPでは、ボリュームをシンプロビジョニングできます。この例では、拡張されたボリュームの合計容量がファイルシステムの合計容量を超えているため、プロビジョニングされた追加のボリューム容量のロックを解除するには、ファイルシステムの合計容量を拡張する必要があります。これについては、次の手順で説明します。

- 次に、パフォーマンスと容量を強化するために、FSx for NetApp ONTAPのスループット容量を2GB/秒から4GB/秒、IOPSから160000、容量を5TBに拡張しました。

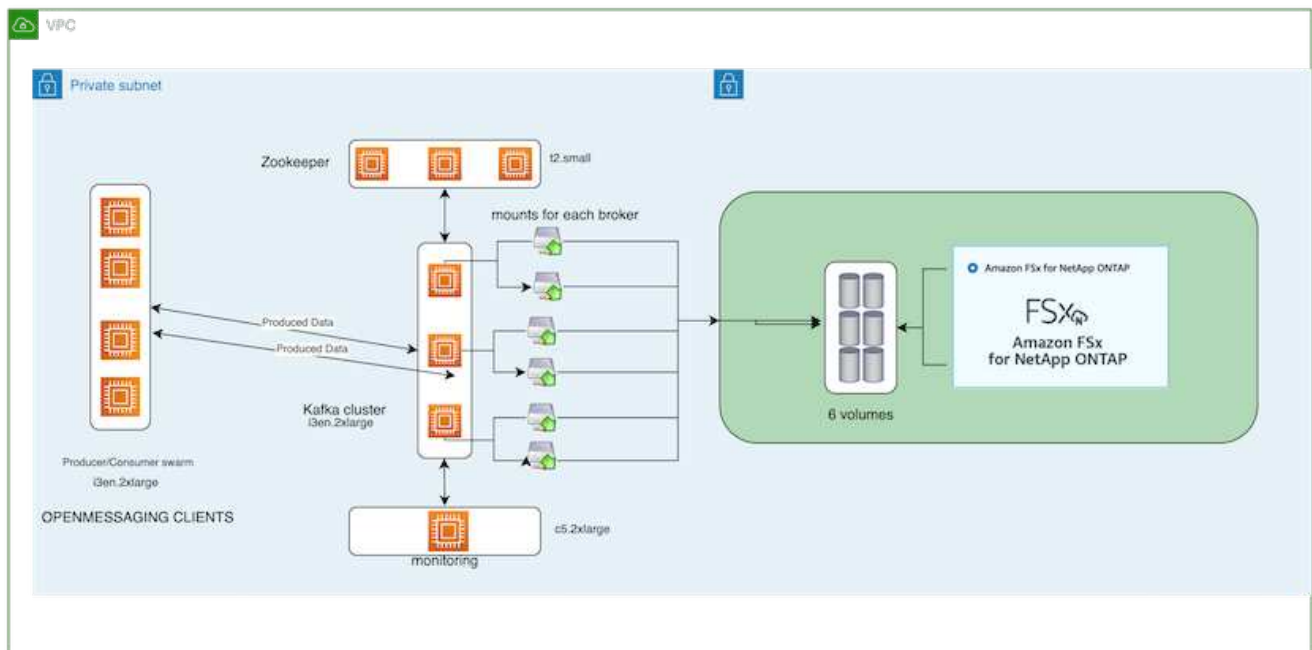
```
[root@ip-172-31-33-69 ~]# aws fsx update-file-system --region us-east-1  
--storage-capacity 5120 --ontap-configuration  
'ThroughputCapacity=4096,DiskIopsConfiguration={Mode=USER_PROVISIONED,Iops=160000}' --file-system-id fs-02ff04bab5ce01c7c
```

FSx 「update-file-system」の詳細なコマンドライン構文は、次のとおりです。

<https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html>

- FSx for NetApp ONTAPボリュームは、nconnectおよびデフォルトのオプションを使用してKafkaブローカーにマウントされます。

次の図は、FSx for NetApp ONTAPベースのKafkaクラスタの最終的なアーキテクチャを示しています。



- ・コンピューティング：3ノードのKafkaクラスタを使用し、専用サーバで3ノードのZookeeperアンサンブルを実行しました。各ブローカーには、FSx for NetApp ONTAPインスタンス上の6つのボリュームに対するNFSマウントポイントが6つありました。
- ・監視：Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成には、独立した3ノードクラスタを使用し、このKafkaクラスタを生成して使用しました。
- ・ストレージ：FSx for NetApp ONTAPを使用し、2TBのボリュームを6個マウントしました。その後、NFSマウントを使用してボリュームをKafkaブローカーにエクスポートしました。FSx for NetApp ONTAPボリュームは、16のnconnectセッションとKafkaブローカーのデフォルトオプションでマウントされます。

OpenMessageベンチマーク設定。

NetApp Cloud Volumes ONTAPと同じ構成を使用しました。詳細はこちらをご覧ください。

<https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup>

テストの方法論

1. Kafkaクラスタは、前述の仕様に従ってterraformとAnsibleを使用してプロビジョニングされました。Terraformを使用して、Kafkaクラスタ用のAWSインスタンスを使用してインフラを構築し、Ansibleを使用してKafkaクラスタを構築します。
2. 上記のワークロード構成とSyncドライバでOMBワークロードがトリガーされました。

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

3. 同じワークロード構成でスループットドライバを使用して別のワークロードがトリガーされました。

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

観察

NFSで実行されるKafkaインスタンスのパフォーマンスをベンチマークするために、2種類のドライバを使用してワークロードを生成しました。ドライバの違いは、log flushプロパティです。

Kafkaレプリケーションファクタ1とFSx for NetApp ONTAPの場合：

- Syncドライバで一貫して生成された総スループット：最大3218 Mbps、最大パフォーマンス（最大3652 Mbps）
- スループットドライバによって一貫して生成された総スループット：最大3679 Mbps、最大3908 Mbpsのパフォーマンス。

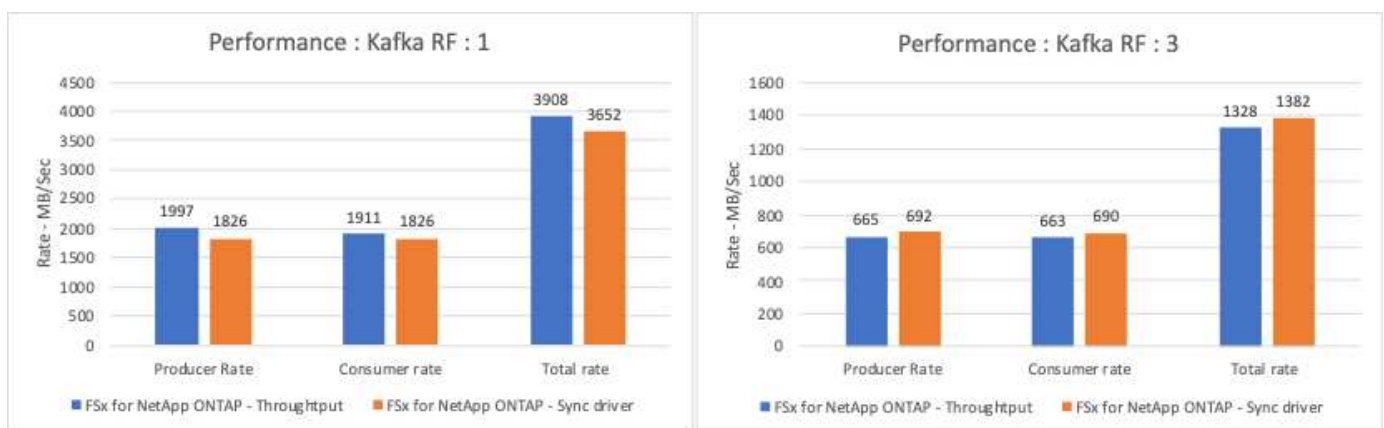
レプリケーションファクタ3のKafkaとFSx for NetApp ONTAPの場合：

- Syncドライバで一貫して生成される合計スループット：最大1252 Mbps、最大1382 Mbps。
- スループットドライバによって一貫して生成される総スループット：最大1218 Mbps、最大パフォーマンス（最大1328 Mbps）

Kafkaレプリケーションファクタ3では、FSx for NetApp ONTAPで読み取りと書き込みの処理が3回行われました。Kafkaレプリケーションファクタ1では、読み取りと書き込みの処理がFSx for NetApp ONTAPで1回行われたため、どちらの検証でも、4GB/秒の最大スループットに到達できました。

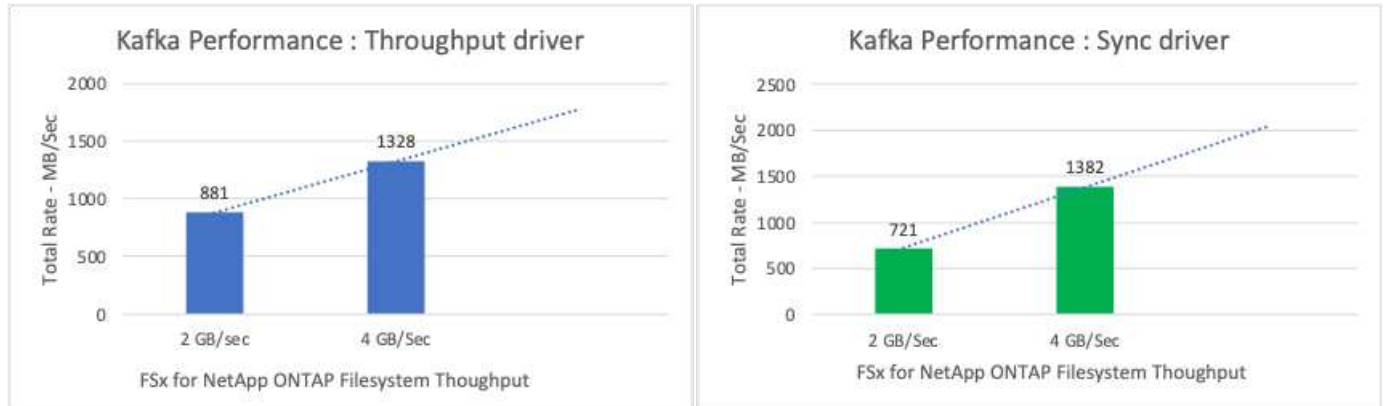
Syncドライバはログが即座にディスクにフラッシュされるときに一貫したスループットを生成できますが、Throughputドライバはログがディスクに一括コミットされるときにスループットのバーストを生成します。

これらのスループット値は、指定されたAWS構成に対して生成されます。より高いパフォーマンス要件に対応するには、インスタンスタイプをスケールアップしてさらに調整し、スループットを向上させることができます。総スループットまたは総レートは、生産者と消費者の両方のレートの組み合わせです。

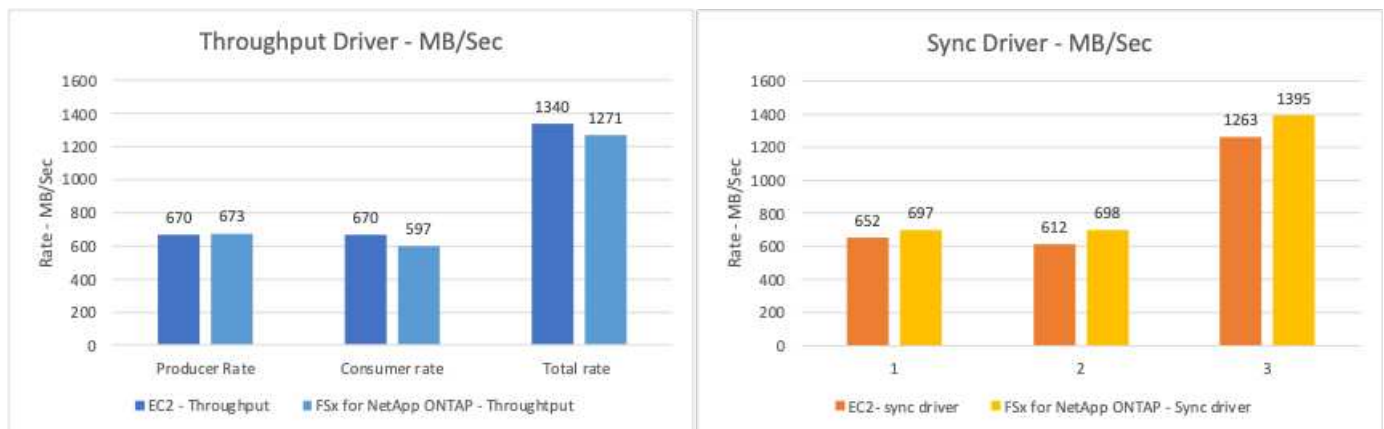


以下の表は、Kafkaレプリケーションファクタ3で2GB/秒のFSx for NetApp ONTAPと4GB/秒のパフォーマンスを示しています。レプリケーションファクタ3は、FSx for NetApp ONTAPストレージで読み取りと書き込みの処理を3回行います。スループットドライバの合計レートは881 MB/秒で、2GB/秒のFSx for NetApp ONTAPファイルシステムではKafkaの読み取りと書き込みを行います。スループットドライバの合計レートは1328

MB/秒で、Kafkaの読み取りと書き込みを行います。Kafkaのパフォーマンスは、FSx for NetApp ONTAPのスループットに基づいてリニアで拡張性に優れています。



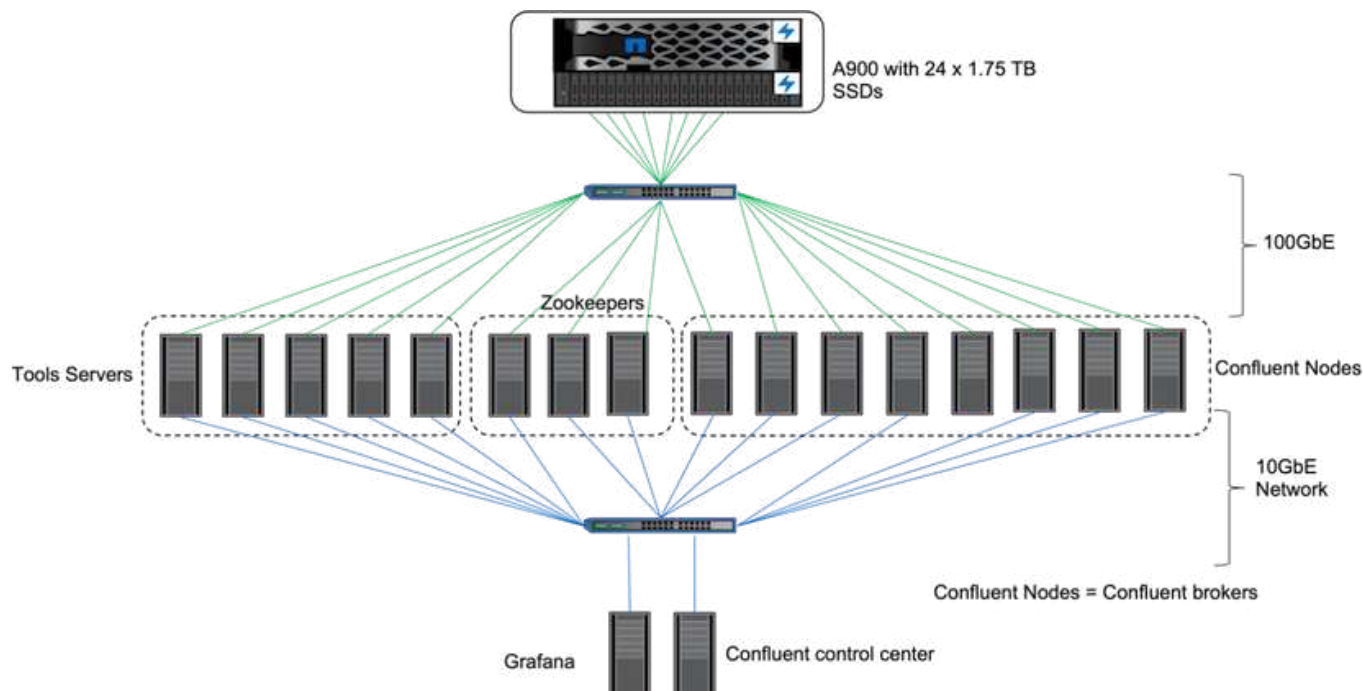
以下のグラフは、EC2インスタンスとFSx for NetApp ONTAPのパフォーマンスを示しています（Kafkaレプリケーション係数：3）。



オンプレミスのAFF A900によるパフォーマンスの概要と検証

オンプレミスでは、NetApp AFF A900ストレージコントローラとONTAP 9.12.1RC1を使用して、Kafkaクラスタのパフォーマンスと拡張性を検証しました。ONTAP およびAFFでは、以前の階層型ストレージのベストプラクティスと同じテストベッドを使用しました。

AFF A900の評価にはConfluent Kafka 6.2.0を使用しました。このクラスタには、8つのブローカーノードと3つのzookeeperノードがあります。パフォーマンステストでは、5つのOMBワーカーノードを使用しました。



ストレージ構成

ネットアップのFlexGroupインスタンスを使用してログディレクトリに単一のネームスペースを提供し、リカバリと設定を簡易化しました。NFSv4.1とpNFSを使用して、ログセグメントデータへの直接アクセスを提供しました。

クライアントの調整

各クライアントは、次のコマンドを使用してFlexGroup インスタンスをマウントしました。

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

さらに、を増やしました `max_session_slots` デフォルトから変更します 64 終了： 180。これは、ONTAP のデフォルトのセッションスロット制限と一致します。

Kafkaブローカーチューニング

テスト対象のシステムのスループットを最大化するために、特定のキースレッドプールのデフォルトパラメータを大幅に増やしました。ほとんどの構成では、Kafkaのベストプラクティスに準拠することを推奨します。この調整は、ストレージに対する未処理のI/Oの同時処理率を最大化するために使用されました。これらのパラメータは、ブローカーのコンピューティングリソースとストレージ属性に合わせて調整できます。

```
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
```

ワークロードジェネレータのテスト方法

スループットドライバとトピック構成には、クラウドテストと同じOMB構成を使用しました。

1. AFF クラスタでAnsibleを使用してFlexGroup インスタンスをプロビジョニングしました。

```

---
- name: Set up kafka broker processes
  hosts: localhost
  vars:
    ntap_hostname: 'hostname'
    ntap_username: 'user'
    ntap_password: 'password'
    size: 10
    size_unit: tb
    vs1: 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 }}"
        vs1: "{{ vs1 }}"
        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でpNFSが有効になりました。

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

3. Cloud Volumes ONTAP と同じワークロード構成でを使用してスループットドライバでワークロードがトリガーされました。「」を参照してください[\[安定した状態でのパフォーマンス\]](#)を参照してください。このワークロードではレプリケーションファクタ3を使用しました。つまり、NFSでログセグメントのコピーが3つ保持されていました。

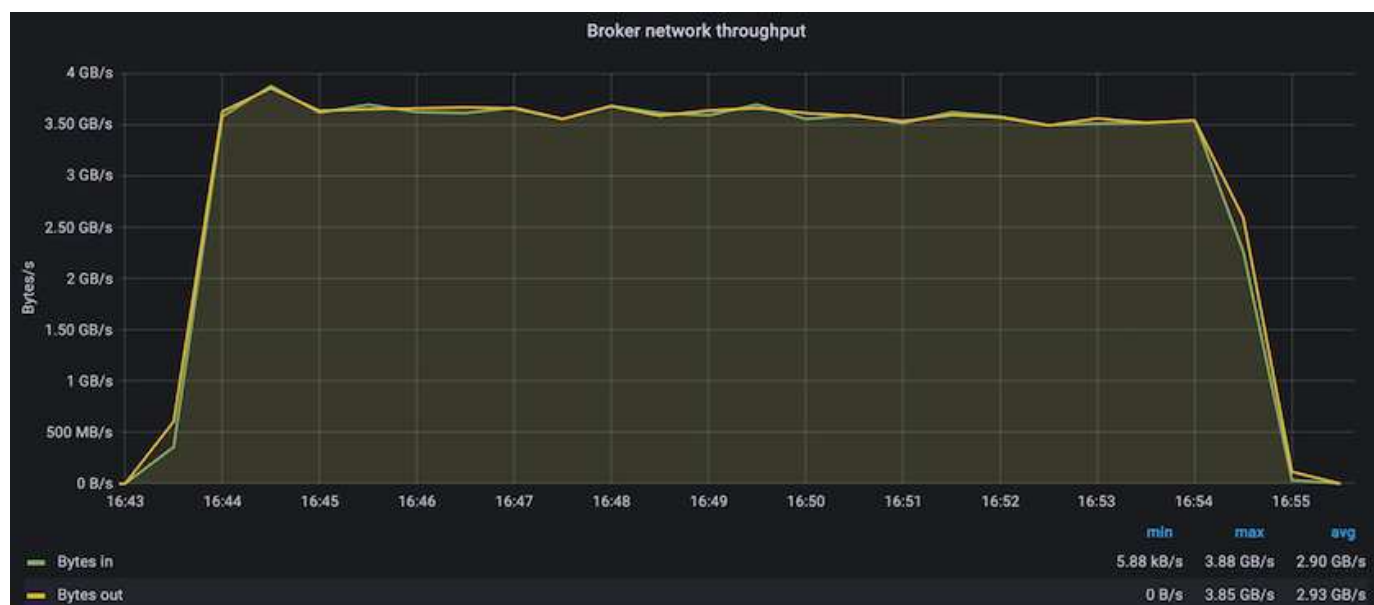
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最後に、バックログを使用して測定を完了し、消費者が最新のメッセージに追いつく能力を測定しました。OMBは、測定の開始時にコンシューマを一時停止することでバックログを作成します。これにより、バックログの作成（生産者のみのトラフィック）、バックログの排出（消費者がトピックで見逃したイベントに追いつくための消費者の多いフェーズ）、および定常状態の3つの異なるフェーズが生成されます。「」を参照してください[ストレージの制限を確認する](#)を参照してください。

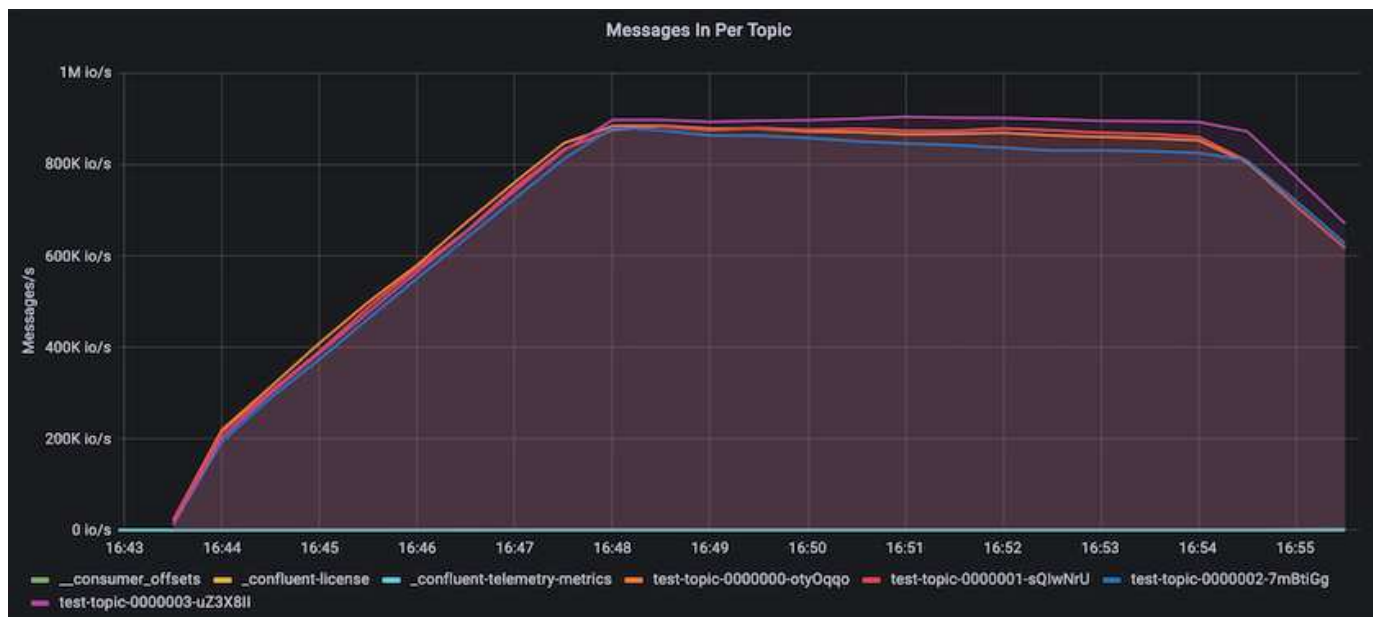
安定した状態でのパフォーマンス

AFF A900は、OpenMessagingベンチマークを使用して評価し、AWSのCloud Volumes ONTAP とAWSのDASと同様の比較を行いました。すべてのパフォーマンス値は、プロデューサーレベルとコンシューマレベルでのKafkaクラスタのスループットを表します。

Confluent KafkaとAFF A900を使用した定常状態のパフォーマンスは、生産者と消費者の両方で平均3.4GBps以上のスループットを達成しました。これは、Kafkaクラスタ全体で340万件を超えるメッセージです。BrokerTopicMetricsの持続スループット（バイト/秒）を視覚化することで、AFF A900がサポートする優れた安定状態のパフォーマンスとトラフィックを確認できます。



これは、トピックごとに配信されるメッセージのビューとよく一致しています。次のグラフは、トピックごとの内訳を示しています。テストした構成では、4つのトピックにわたってトピックごとに約9万件のメッセージが表示されました。

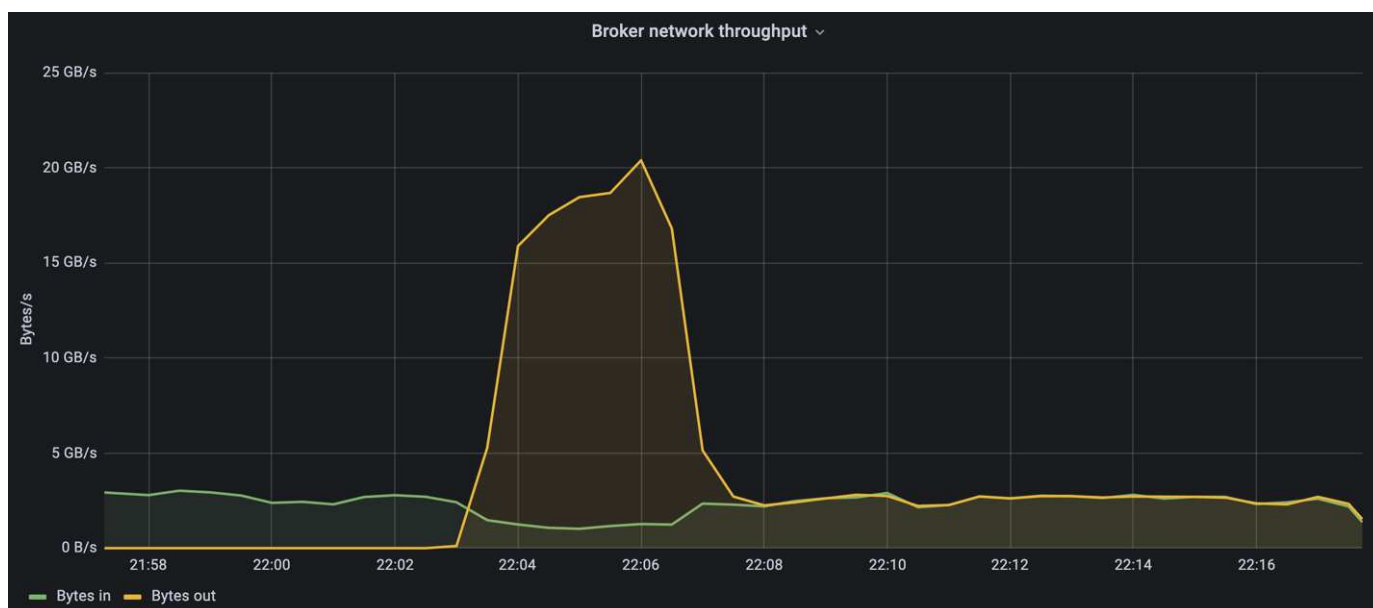


卓越したパフォーマンスを発揮し、ストレージの限界を探る

AFF では、バックログ機能を使用してOMBでテストしました。バックログ機能は、Kafkaクラスタでイベントのバックログが作成されている間、コンシューマサブスクリプションを一時停止します。このフェーズでは、プロデューサトラフィックのみが発生し、ログにコミットされたイベントが生成されます。これは、バッチ処理またはオフライン分析ワークフローを最も厳密にエミュレートします。これらのワークフローでは、コンシューマサブスクリプションが開始され、ブローカーキャッシュからすでに削除されている履歴データを読み取る必要があります。

この構成で利用者のスループットに関するストレージの制限を把握するために、Producer-Onlyフェーズを測定して、A900がどの程度の書き込みトラフィックを吸収できるかを調べました。次のセクションを参照してください[\[サイジングガイド\]](#)を参照してください。

この測定のプロデューサーのみの部分では、ピークスループットが高く、A900のパフォーマンスの限界を押し上げていることがわかりました（他のブローカーリソースがプロデューサーおよびコンシューマトラフィックに対応していない場合）。





この測定では、メッセージあたりのオーバーヘッドを制限し、NFSマウントポイントに対するストレージスループットを最大化するために、メッセージサイズを16kに増やしました。

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

Confluent Kafkaクラスターは、生産者のピークスループット4.03GBpsを達成しました。

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

OMBによるイベントバックログの入力が完了すると、コンシューマトラフィックが再開されました。バックログドレーンを使用した測定では、すべてのトピックで消費者のピークスループットが20Gbpsを超えることが確認されました。OMBログデータを格納するNFSボリュームの合計スループットは約30Gbpsに達しました。

サイジングガイド

Amazon Web Servicesは、を提供します ["サイジングガイド"](#) Kafkaクラスターのサイジングと拡張に使用できます。

このサイジングは、Kafkaクラスターのストレージスループット要件を決定するのに便利な計算式を提供します。

tclusterのクラスター内で生成される集約スループット（レプリケーション係数r）の場合、ブローカストレージが受け取るスループットは次のとおりです。

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

これはさらに単純化することができます。

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

この式を使用すると、Kafkaホットティアのニーズに適したONTAP プラットフォームを選択できます。

次の表に、A900の予測される生産者スループットと、さまざまなレプリケーション要因を示します。

レプリケーションファクタ	生産者スループット（GPPS）
3（実測値）	3.4.
2.	5.1
1.	10.2

まとめ

NetApp解決策 for the silly rename problemは、これまでNFSと互換性がなかったワークロード向けに、シンプルで安価な一元管理されたストレージを提供します。

この新しいパラダイムにより、ディザスタリカバリやデータ保護を目的とした移行やミラーリングが容易な、管理しやすいKafkaクラスタを作成できます。

また、NFSは、CPU利用率の削減とリカバリ時間の短縮、ストレージ効率の大幅な向上、NetApp ONTAP によるパフォーマンスの向上など、追加のメリットをもたらすこともわかりました。

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- Apache Kafkaとは

["https://www.confluent.io/what-is-apache-kafka/"](https://www.confluent.io/what-is-apache-kafka/)

- 愚かな名前変更とは何ですか？

["https://linux-nfs.org/wiki/index.php/Server-side_silly_rename"](https://linux-nfs.org/wiki/index.php/Server-side_silly_rename)

- ONATPはストリーミングアプリケーション用に読み取られます。

["https://www.netapp.com/blog/ontap-ready-for-streaming-applications/"](https://www.netapp.com/blog/ontap-ready-for-streaming-applications/)

- Silly -問題 の名前をKafkaに変更します。

["https://sbg.technology/2018/07/10/kafka-nfs/"](https://sbg.technology/2018/07/10/kafka-nfs/)

- ネットアップの製品マニュアル

["https://www.netapp.com/support-and-training/documentation/"](https://www.netapp.com/support-and-training/documentation/)

- NFSとは

["https://en.wikipedia.org/wiki/Network_File_System"](https://en.wikipedia.org/wiki/Network_File_System)

- Kafkaパーティションの再割り当てとは何ですか。

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

- OpenMessagingベンチマークとは

["https://openmessaging.cloud/"](https://openmessaging.cloud/)

- Kafkaブローカーはどのように移行しますか？

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

- PrometheusでKafkaブローカーをどのように監視していますか？

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

- Apache Kafka向けマネージドプラットフォーム

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

- Apache Kafkaのサポート

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

- Apache Kafkaのコンサルティングサービス

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

著作権に関する情報

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S. このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータ ソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用权を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用权については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。