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

NetApp artificial intelligence solutions

NetApp
February 12, 2026

目次

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

TR-4947: NetApp NFSストレージを使用したApache Kafkaワークロード - 機能検証とパフォーマンス

Shantanu Chakole、Karthikeyan Nagalingam、Joe Scott、 NetApp

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のパブリッシュ/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トピックに非常に高速にデータを書き込んだり読み取ったりできます。 Kafka は、そのフォールト トレランスとスケーラビリティにより、多数のデータ ストリームを非常に高速に取り込んで移動する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、メトリックの収集と監視、ログの集約、リアルタイム分析などがあります。

NFS 上の通常の Kafka 操作は正常に動作しますが、NFS 上で実行されている Kafka クラスターのサイズ変更または再パーティション化中に、名前変更の問題によりアプリケーションがクラッシュします。負荷分散やメンテナンスの目的で Kafka クラスターのサイズを変更したり、再パーティション化したりする必要があるため、これは重大な問題です。詳細は以下をご覧ください "ここをクリックしてください。"。

このドキュメントでは、次の内容について説明します。

- 馬鹿げた名前変更問題と解決策の検証
- CPU使用率を減らしてI/O待機時間を短縮する
- Kafkaブローカーの回復時間の短縮
- 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は何ですか?

実稼働アプリケーションの Kafka ワークロードは、アプリケーション間で膨大な量のデータをストリーミングできます。このデータは、Kafka クラスター内の Kafka ブローカー ノードに保持され、保存されます。 Kafka は可用性と並列性でも知られており、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ー全体に複製することでこれを実現します。これは、最終的に、Kafka クラスターを流れる膨大な量のデータのサイズが通常倍増することを意味します。 NFS を使用すると、ブローカーの数の変化に応じてデータの再バランスが非常に迅速かつ簡単に行えます。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 間でデータを再調整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は頻繁に変更されます。

その他の利点は次のとおりです。

- 成熟。 NFS は成熟したプロトコルであり、その実装、セキュリティ保護、および使用のほとんどの側面が十分に理解されていることを意味します。
- 開ける。 NFS はオープン プロトコルであり、その継続的な開発は、無料かつオープンなネットワーク プロトコルとしてインターネット仕様に文書化されています。

- コスト効率が良い。 NFS は、既存のネットワーク インフラストラクチャを使用するためセットアップが簡単な、低コストのネットワーク ファイル共有ソリューションです。
- 集中管理 NFS を集中管理することで、個々のユーザー システムに追加のソフトウェアやディスク領域が必要になることが減ります。
- 配布済み NFS は分散ファイル 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。

Kafka ワークロードにNetApp を選ぶ理由

NetApp 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ており、数え切れないほどのエンタープライズ NAS 環境で使用されています。NetAppの信頼性に加えて、次のような利点もあります。

- 信頼性と効率性
- スケーラビリティとパフォーマンス
- 高可用性（ NetApp ONTAPクラスタの HA パートナー）
- データ保護
  - *災害復旧 (NetApp SnapMirror)*サイトがダウンした場合、または別のサイトで再開して中断したところから続行する必要がある場合。
  - ストレージ システムの管理性 (NetApp OnCommandを使用した管理)。
  - *負荷分散*クラスターを使用すると、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。
  - 中断のない運用。 LIF または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。

NFS から Kafka へのワークロードの名前変更に関する問題に対するNetApp のソリューション

Kafka は、基盤となるファイルシステムが POSIX 準拠 (たとえば、XFS または Ext4) であることを前提に構築されています。 Kafka リソースの再バランス調整により、アプリケーションがまだ使用しているファイルが削除されます。 POSIX 準拠のファイル システムでは、unlink を続行できます。ただし、ファイルへの参照がすべてなくなった後にのみファイルが削除されます。基礎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 クライアントはリンク解除呼び出しをインターセプトし、ワークフローを管理します。リンク解除されるファイルには保留中のオープンがあるため、NFS クライアントは NFS サーバーに名前変更要求を送信し、リンク解除されたファイルが最後に閉じられるときに、名前が変更されたファイルに対して削除操作を発行します。この動作は一般に NFS の奇妙な名前変更と呼ばれ、NFS クライアントによって調整されます。

この動作により、NFSv3 サーバーのストレージを使用する Kafka ブローカーで問題が発生します。ただし、NFSv4.x プロトコルには、開かれたリンクされていないファイルに対してサーバーが責任を持つようにすることで、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 サーバーは、ファイルを開くときに所有権機能を NFS クライアントに伝えます。 NFS クライアントは、保留中のオープンがある場合にリンク解除の管理を停止し、サーバーがフローを管理できるようにします。 NFSv4 仕様では実装のガイドラインが提供されていますが、これまでこのオプション機能をサポートする NFS サーバー実装は

知られていませんでした。

不合理な名前変更の問題を解決するには、NFS サーバーと NFS クライアントに次の変更が必要です。

- *NFS クライアント (Linux) への変更。*ファイルが開かれるときに、NFS サーバーは開かれたファイルのリンク解除を処理できることを示すフラグで応答します。 NFS クライアント側の変更により、フラグが存在する場合に NFS サーバーがリンク解除を処理できるようになります。 NetApp は、これらの変更をオープンソースの Linux NFS クライアントに反映しました。更新された NFS クライアントは、RHEL8.7 および RHEL9.1 で一般公開されました。
- **NFS** サーバーへの変更。 NFS サーバー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 セマンティクスに合わせてサーバーによって管理されるようになりました。最後のオープンが閉じられると、NFS サーバーはファイルの実際の削除を開始し、無駄な名前変更プロセスを回避します。 ONTAP NFS サーバは、最新リリースのONTAP 9.12.1 でこの機能を実装しました。

NFS クライアントとサーバーに上記の変更を加えることで、Kafka はネットワーク接続された NFS ストレージの利点をすべて安全に享受できるようになります。

機能検証 - ばかげた名前変更の修正

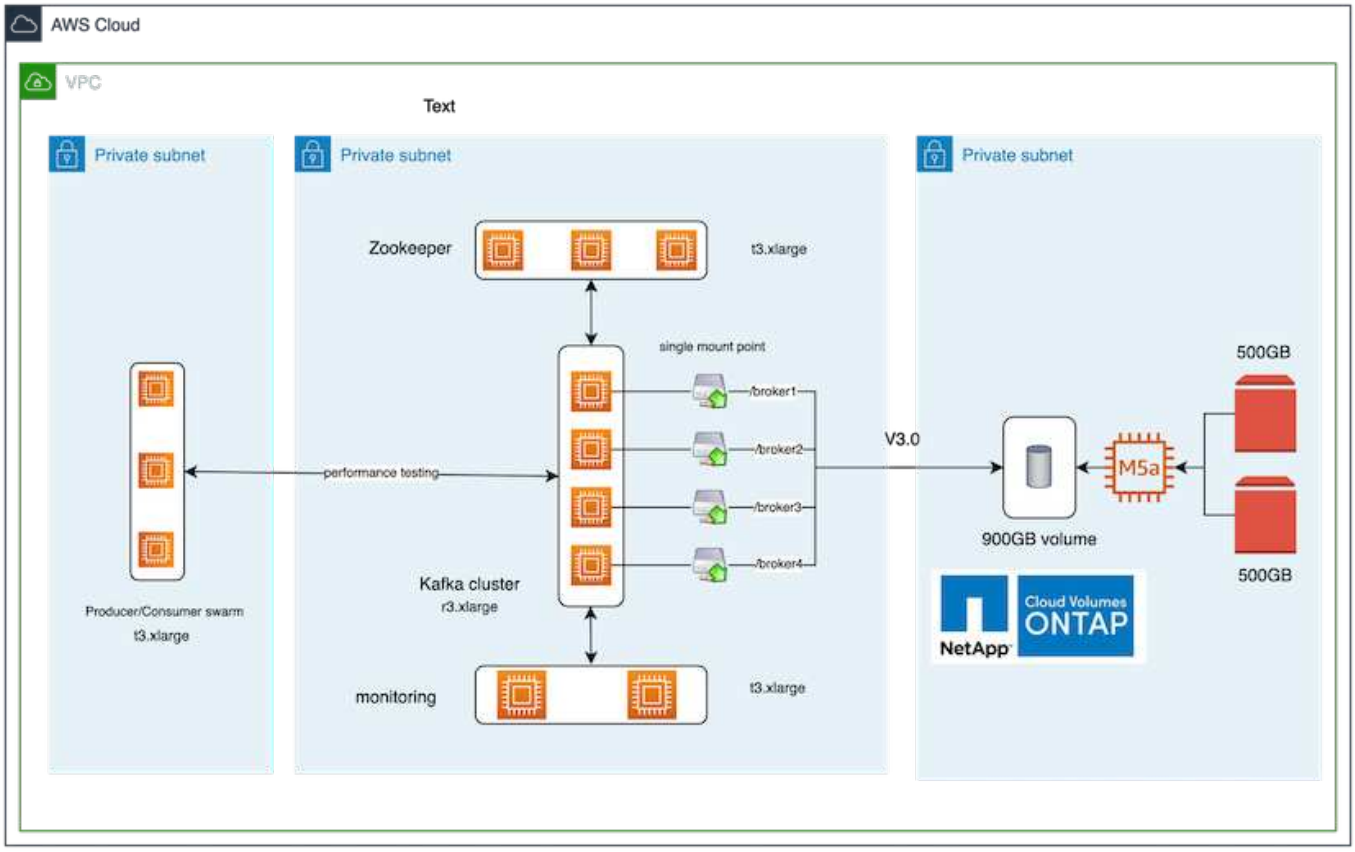
機能検証では、ストレージに NFSv3 をマウントした Kafka クラスターはパーティションの再配分などの Kafka 操作を実行できないのに対し、修正を適用した NFSv4 にマウントされた別のクラスターは中断なく同じ操作を実行できることを示しました。

検証設定

セットアップは AWS 上で実行されます。次の表は、検証に使用されたさまざまなプラットフォーム コンポーネントと環境構成を示しています。

プラットフォームコンポーネント	環境設定
Confluent Platform バージョン 7.2.1	• 飼育員3人 – t3.xlarge • 4台のブローカーサーバー – r3.xlarge • 1 x Grafana – t3.xlarge • コントロールセンター x 1 – t3.xlarge • 3 x 生産者/消費者
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、このソリューションのアーキテクチャ構成を示しています。



建築の流れ

- *計算します。*専用サーバーで実行される 3 ノードの Zookeeper アンサンブルを備えた 4 ノードの Kafka クラスターを使用しました。
- 監視。 Prometheus と Grafana の組み合わせには 2 つのノードを使用しました。
- *作業量。*ワークロードを生成するために、この Kafka クラスターにデータを生成したり、この Kafka クラスターからデータを消費したりできる、別の 3 ノード クラスターを使用しました。
- *ストレージ。*インスタンスに 2 つの 500 GB GP2 AWS-EBS ボリュームが接続された、単一ノードのNetApp Cloud Volumes ONTAPインスタンスを使用しました。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 クラスターに公開されました。

すべてのサーバーに対して、Kafka のデフォルトのプロパティが選択されました。動物園の飼育員の群れにも同じことを行いました。

テストの方法論

1. アップデート `-is-preserve-unlink-enabled true`kafka ボリュームに次のように追加します。

```
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. 次の違いを持つ 2 つの類似した Kafka クラスターが作成されました。
   - *クラスター1*実稼働対応のONTAPバージョン 9.12.1 を実行するバックエンド NFS v4.1 サーバーは、 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 がインストールされました。
   - *クラスター2*バックエンド NFS サーバーは、手動で作成された汎用 Linux NFSv3 サーバーでした。
3. 両方の Kafka クラスターにデモ トピックが作成されました。

   クラスター 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:
```

   クラスター 2:

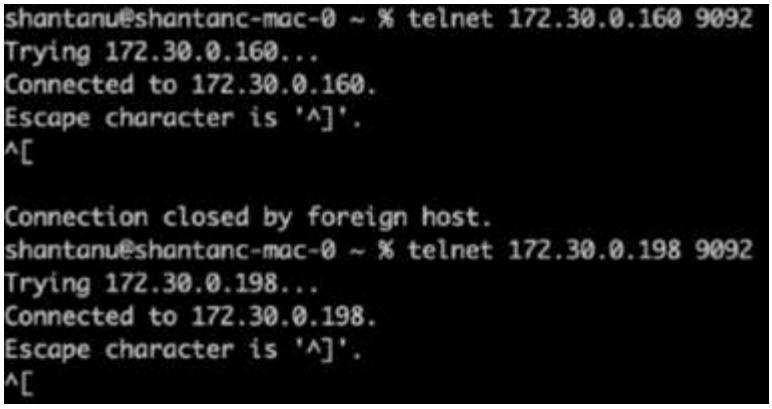
```
[root@ip-172-30-0-198 demo]# kafka-topics --bootstrap-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:90
92,172.30.0.204:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: AwQpsZTQShyeMIhaquCG3Q PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        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. 両方のクラスターの新しく作成されたトピックにデータがロードされました。これは、デフォルトの Kafka パッケージに付属する produced-perf-test ツールキットを使用して実行されました。

```
./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、各クラスターのブローカー 1 のヘルス チェックが実行されました。
   - テルネット `172.30.0.160 9092`
   - テルネット `172.30.0.198 9092`

     次のスクリーンショットは、両方のクラスター上のブローカーのヘルスチェックが成功したことを示しています。



6. NFSv3 ストレージ ボリュームを使用する Kafka クラスターがクラッシュする障害状態をトリガーするために、両方のクラスター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、 `kafka-reassign-partitions.sh` 。詳細なプロセスは次のとおりです。

   a. Kafka クラスター内の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ーに対して実行されました)。

   ```
   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は、 `/tmp/reassignment- file.json` 。

   c. 実際のパーティションの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、ブローカーの別のヘルス チェックで、NFSv3 ストレージ ボリュームを使用しているクラスターが不合理な名前変更の問題に遭遇してクラッシュした一方で、修正が適用され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ー 1 は中断することなく操作を継続していることが示されました。

```
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
```

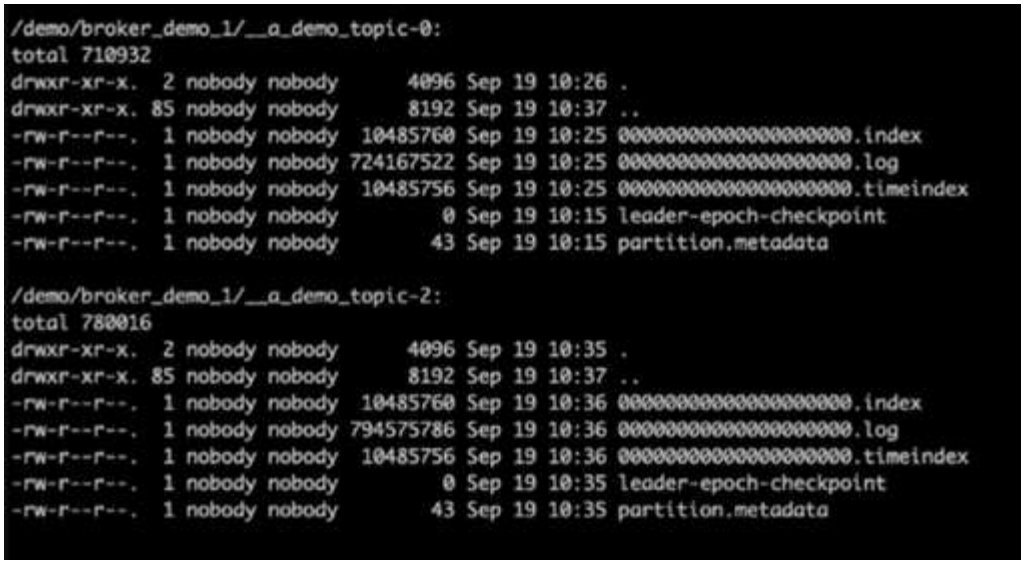
- Cluster1-Broker-1 は稼働しています。
- Cluster2-broker-1 は停止しています。

8. Kafka のログ ディレクトリを確認すると、修正が適用されたNetApp ONTAP NFSv4.1 ストレージ ボリュームを使用する Cluster 1 ではパーティション割り当てが適切に行われていたのに対し、汎用 NFSv3 ストレージを使用する Cluster 2 では、名前変更の問題によってパーティション割り当てが適切に行われず、クラッシュが発生していたことが明らかになりました。次の図は、クラスター 2 のパーティションの再バランスを示しています。これにより、NFSv3 ストレージで名前変更の問題が発生しました。

```
/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 .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
```

次の図は、 NetApp NFSv4.1 ストレージを使用したクラスター 1 のクリーン パーティションの再バランスを示しています。



Kafka ワークロードにNetApp NFS を使用する理由

Kafka を使用した NFS ストレージの無意味な名前変更の問題に対する解決策ができたので、Kafka ワークロードにNetApp ONTAPストレージを活用する堅牢なデプロイメントを作成できます。これにより、運用上のオーバーヘッドが大幅に削減されるだけでなく、Kafka クラスターに次のような利点がもたらされます。

- *Kafka ブローカーの CPU 使用率が削減されました。*分散型NetApp ONTAPストレージを使用すると、ディスク I/O 操作がブローカーから分離され、CPU フットプリントが削減されます。
- *ブローカーの回復時間が短縮されます。*分散されたNetApp ONTAPストレージは Kafka ブローカー ノード間で共有されるため、従来の Kafka デプロイメントと比較して、データを再構築することなく、いつでも短時間で新しいコンピューティング インスタンスが不良ブローカーを置き換えることができます。
- *ストレージ効率。*アプリケーションのストレージ層がNetApp ONTAPを通じてプロビジョニングされるようになったため、顧客はインライン データ圧縮、重複排除、圧縮など、 ONTAPに備わっているストレージ効率の利点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ています。

Kafka ブローカーの CPU 使用率の削減

技術仕様は同一だが、ストレージ テクノロジが異なる 2 つの別々の Kafka クラスターで同様のワークロードを実行したところ、全体的な CPU 使用率が DAS よりも低いことがわかりました。 Kafka クラスターがONTAPストレージを使用している場合、全体的な CPU 使用率が低くなるだけでなく、CPU 使用率の増加は DAS ベースの Kafka クラスターよりも緩やかな勾配を示しました。

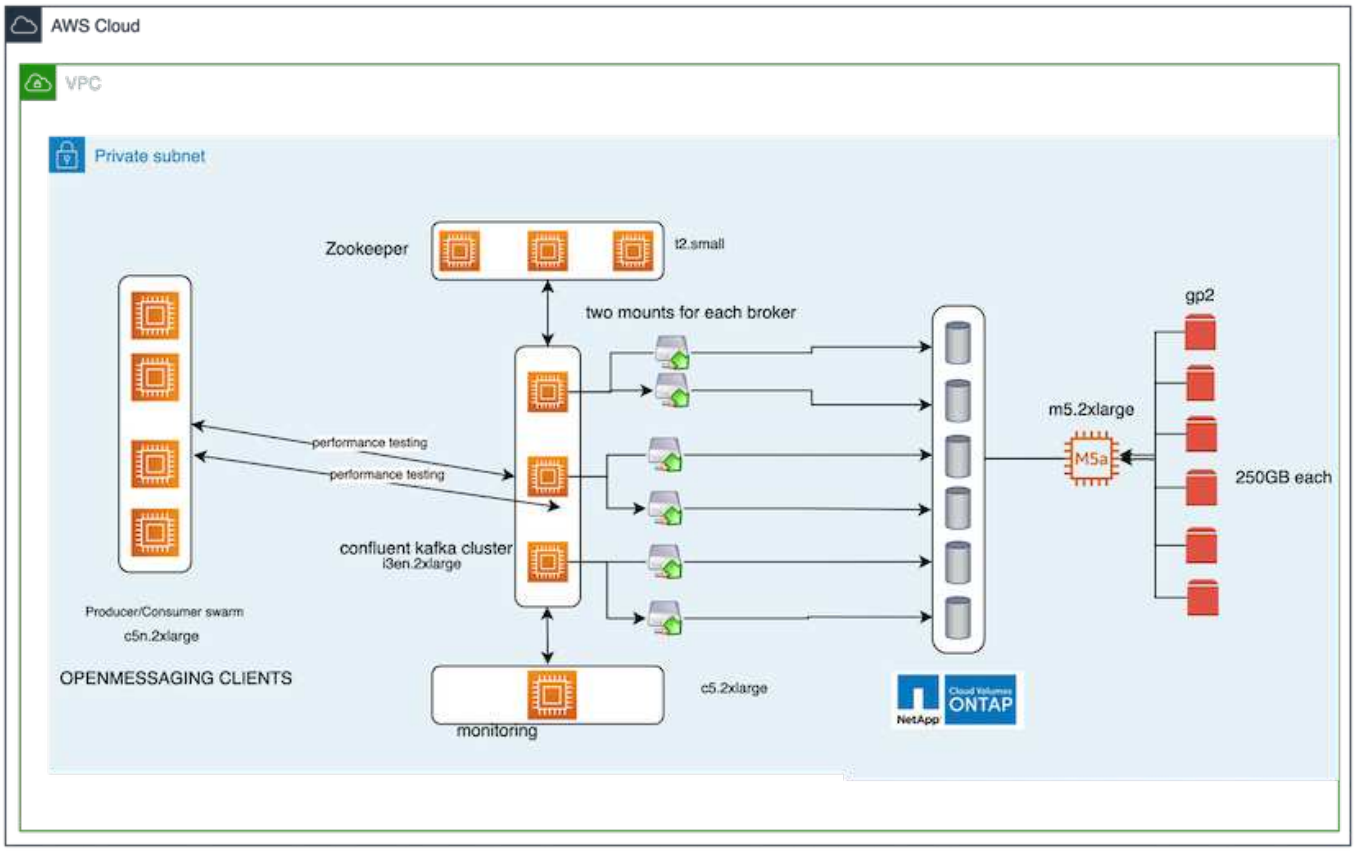
建築のセットアップ

次の表は、CPU 使用率の削減を示すために使用された環境構成を示しています。

プラットフォームコンポーネント	環境設定
Kafka 3.2.3 ベンチマークツール: OpenMessaging	• 飼育員×3 – t2.small • ブローカーサーバー x 3 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 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したベンチマークツールは "オープンメッセージング"フレームワーク。OpenMessaging はベンダー中立かつ言語に依存しません。金融、電子商取引、IoT、ビッグデータに関する業界ガイドラインを提供し、異機種システムやプラットフォーム間でのメッセージングおよびストリーミングアプリケーションの開発に役立ちます。次の図は、OpenMessaging クライアントと Kafka クラスターの相互作用を示しています。



- *計算します。*専用サーバーで実行される 3 ノードの Zookeeper アンサンブルを備えた 3 ノードの Kafka クラスターを使用しました。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ボリュームへの 2 つの NFSv4.1 マウント ポイントがありました。
- 監視。 Prometheus と Grafana の組み合わせには 2 つのノードを使用しました。ワークロードを生成するために、この Kafka クラスターにワークロードを生成したり、この Kafka クラスターからワークロードを

消費したりできる、独立した 3 ノード クラスターがあります。

- *ストレージ。*インスタンスにマウントされた 6 つの 250 GB GP2 AWS-EBS ボリュームを備えた単一ノードのNetApp Cloud Volumes ONTAPインスタンスを使用しました。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 クラスターに公開されました。
- *構成。*このテスト ケースで構成可能な 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 ブローカーには次の仕様が選択されました。以下で強調されているように、すべての測定に複製係数 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) ワークロード構成。*以下の仕様が提供されました。以下に強調表示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ーが作成され、それぞれ独自のベンチマーク クラスター スウォームのセットを持ちました。
   - クラスター**1** NFS ベースの Kafka クラスター。
   - クラスター**2** DAS ベースの Kafka クラスター。
2. OpenMessaging コマンドを使用して、各クラスターで同様のワークロードがトリガーされました。

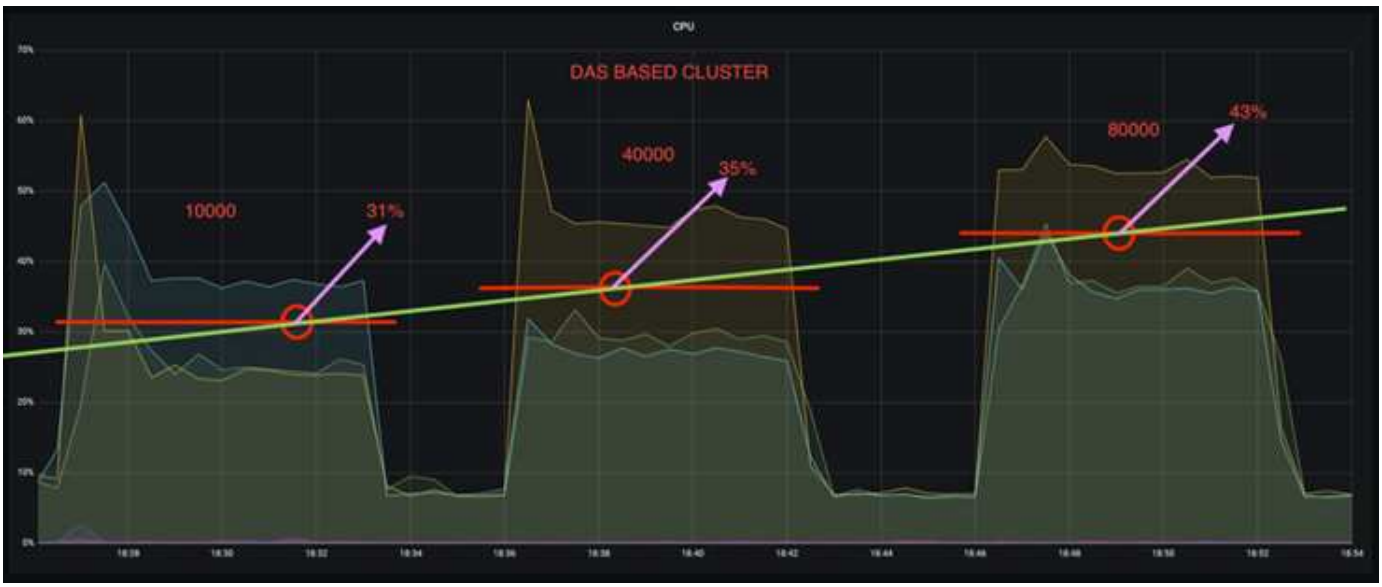
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

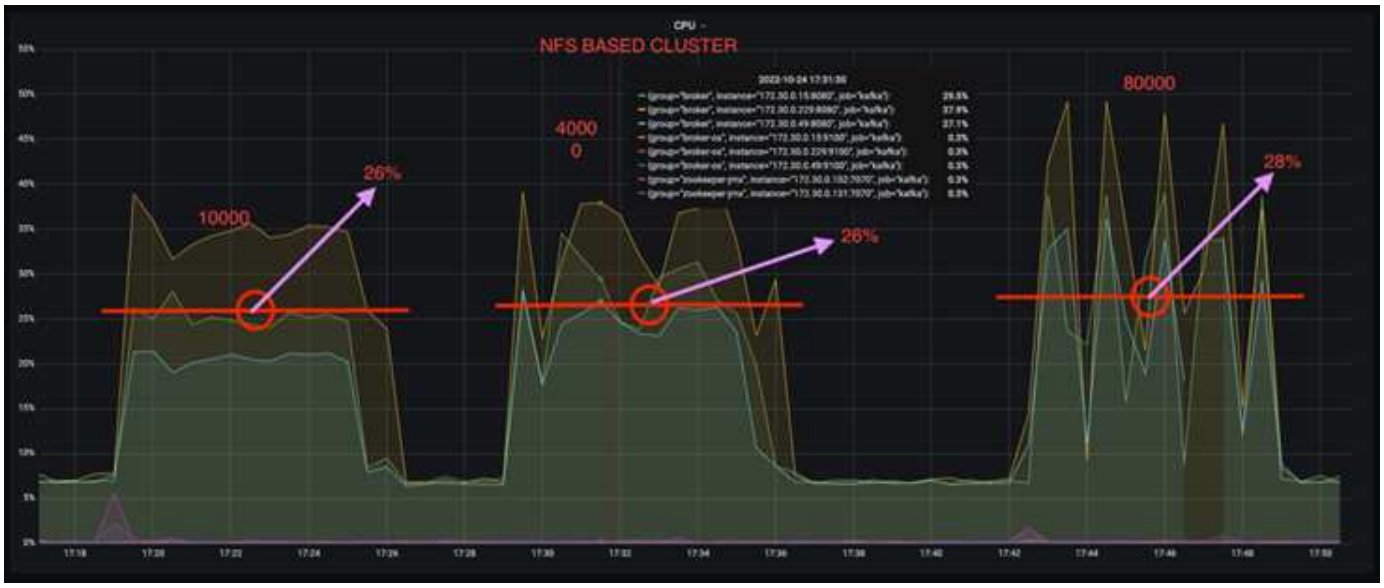
3. 生成率の設定は 4 回の反復で増加され、CPU 使用率は Grafana で記録されました。生産率は次のレベルに設定されました。
   - 10,000
   - 40,000
   - 80,000
   - 100,000

観察

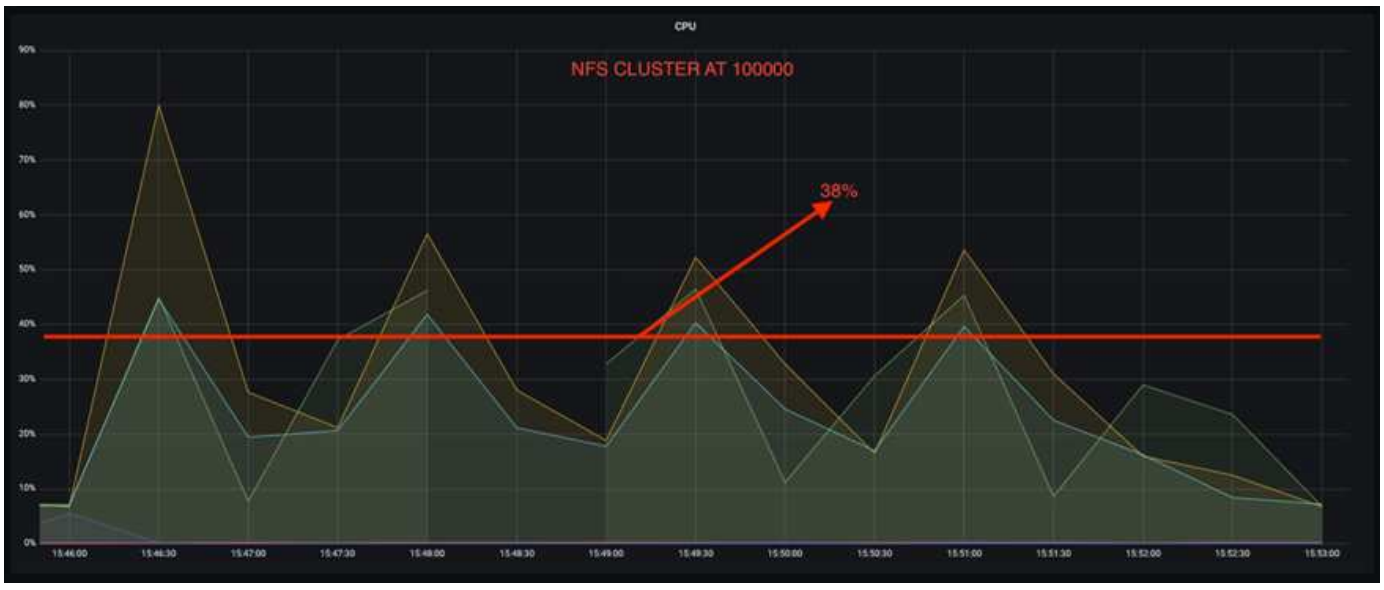
Kafka でNetApp NFS ストレージを使用すると、主に 2 つの利点があります。

- *CPU 使用率を約 3 分の 1 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 の方が低く、節約幅は生成率が低い場合は 5%、生成率が高い場合は 32% です。
- *生産率が高い場合の CPU 使用率のドリフトが 3 分の 1 に減少します。*予想どおり、生産率が上がるにつれて、CPU 使用率の増加は上向きに推移しました。ただし、DAS を使用する Kafka ブローカーの CPU 使用率は、低い生成率の場合の 31% から高い生成率の場合の 70% に上昇し、39% 増加しました。ただし、NFS ストレージ バックエンドでは、CPU 使用率は 26% から 38% に上昇し、12% 増加しました。





また、100,000 件のメッセージでは、DAS は NFS クラスターよりも CPU 使用率が高くなります。



ブローカーの回復が速い

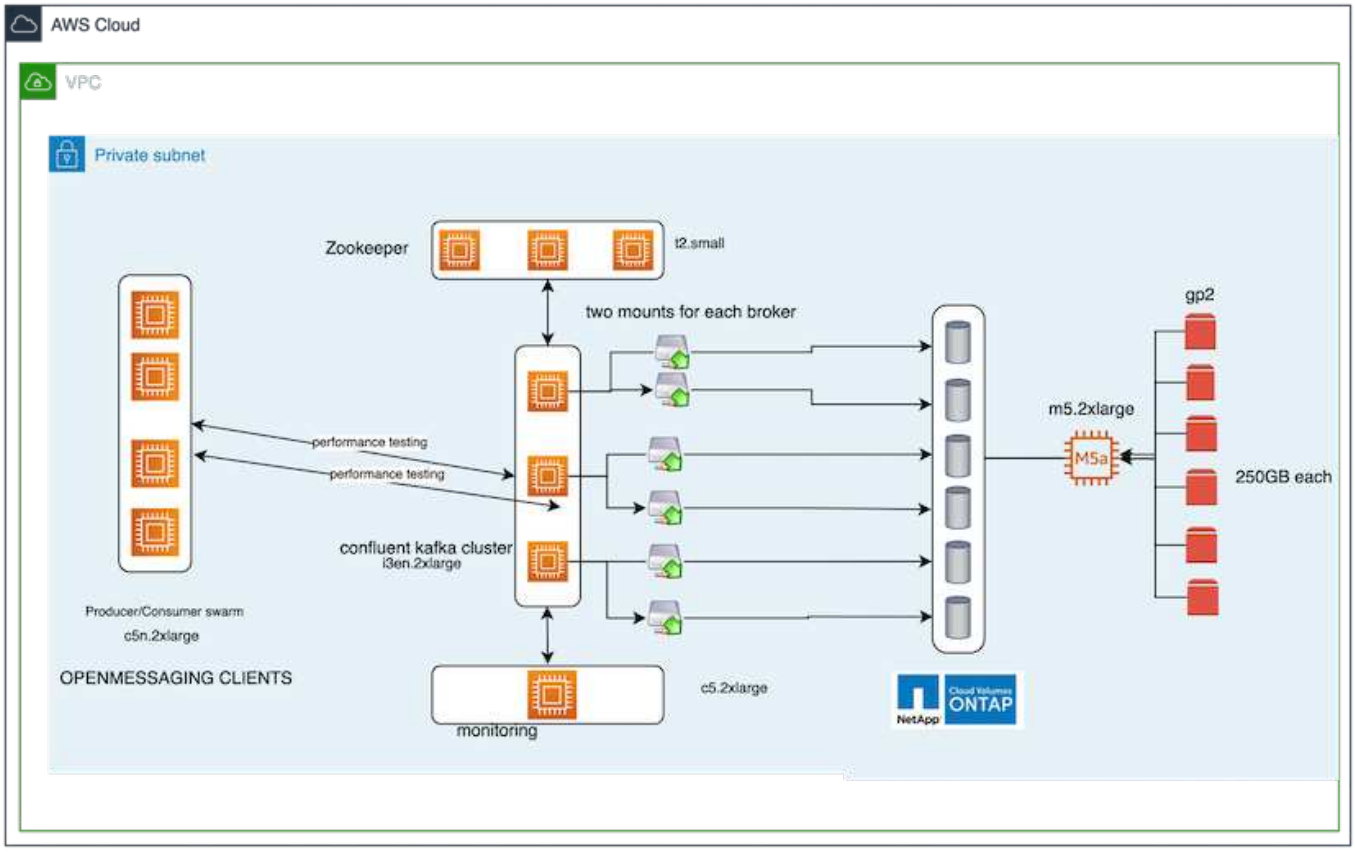
共有NetApp NFS ストレージを使用すると、Kafka ブローカーの回復が速くなることがわかりました。 Kafka クラスターでブローカーがクラッシュした場合、このブローカーは同じブローカー ID を持つ正常なブローカーに置き換えられます。このテストケースを実行すると、DAS ベースの Kafka クラスターの場合、クラスターは新しく追加された正常なブローカー上でデータを再構築するため、時間がかかることがわかりました。 NetApp NFS ベースの Kafka クラスターの場合、置き換えたブローカーは以前のログ ディレクトリからデータを読み取り続けるため、回復がはるかに速くなります。

建築のセットアップ

次の表は、NAS を使用した Kafka クラスターの環境構成を示しています。

プラットフォームコンポーネント	環境設定
カフカ 3.2.3	• 飼育員×3 – t2.small • ブローカーサーバー x 3 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x プロデューサー/コンシューマー — 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 クラスターのアーキテクチャを示しています。



- *計算します。*専用サーバー上で実行される 3 ノードの Zookeeper アンサンブルを備えた 3 ノードの Kafka クラスター。各ブローカーには、専用 LIF を介してNetApp CVO インスタンス上の単一ボリュームへの 2 つの NFS マウント ポイントがあります。
- 監視。 Prometheus と Grafana の組み合わせの 2 つのノード。ワークロードを生成するために、この Kafka クラスターに対して生成および消費できる別の 3 ノード クラスターを使用します。
- *ストレージ。*インスタンスにマウントされた 6 つの 250 GB GP2 AWS-EBS ボリュームを持つ単一ノードのNetApp Cloud Volumes ONTAPインスタンス。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 クラスターに公開されます。
- *ブローカーの構成*このテスト ケースで構成可能な唯一の要素は Kafka ブローカーです。 Kafka ブローカーには次の仕様が選択されました。その `replica.lag.time.mx.ms`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。不良ノードと正常な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する必要があります。

```
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

1. 2 つの類似したクラスターが作成されました。
   - EC2 ベースの合流クラスター。
   - NetApp NFS ベースの合流クラスター。
2. 元の Kafka クラスターのノードと同一の構成で、スタンバイ Kafka ノードが 1 つ作成されました。
3. 各クラスターでサンプルトピックが作成され、ブローカーごとに約 110 GB のデータが入力されました。
   - **EC2** ベースのクラスター。 Kafkaブローカーデータディレクトリは、 `/mnt/data-2` (次の図では、cluster1 の Broker-1 [左端末])。
   - * NetApp NFS ベースのクラスター。* KafkaブローカーデータディレクトリはNFSポイントにマウントされます `/mnt/data`(次の図では、cluster2 の Broker-1 [右端末])。

4. 各クラスターで、Broker-1 が終了され、失敗したブローカーの回復プロセスがトリガーされました。
5. ブローカーが終了した後、ブローカーの IP アドレスがスタンバイ ブローカーのセカンダリ IP として割り当てられました。これは、Kafka クラスター内のブローカーが次のように識別されるため必要でした。
   - *IPアドレス*障害が発生したブローカー 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`。
6. IP が割り当てられると、スタンバイ ブローカーで Kafka サービスが開始されました。
7. しばらくして、サーバー ログを取得して、クラスター内の置換ノードでデータを構築するのにかかった時間をチェックしました。

観察

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

EC2 ベースのクラスターでは新しいブローカー ノードで 110 GB のデータを再構築するのに 10 分かかりましたが、NFS ベースのクラスターでは 3 分でリカバリを完了しました。また、ログでは、EC2 のパーティションのコンシューマー オフセットが 0 である一方、NFS クラスターではコンシューマー オフセットが以前のブローカーから取得されていることも確認しました。

```
[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ベースのクラスター

1. バックアップ ノードは 08:55:53,730 に開始されました。

```
[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 (or
[2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
[2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.3
[2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

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

```
[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

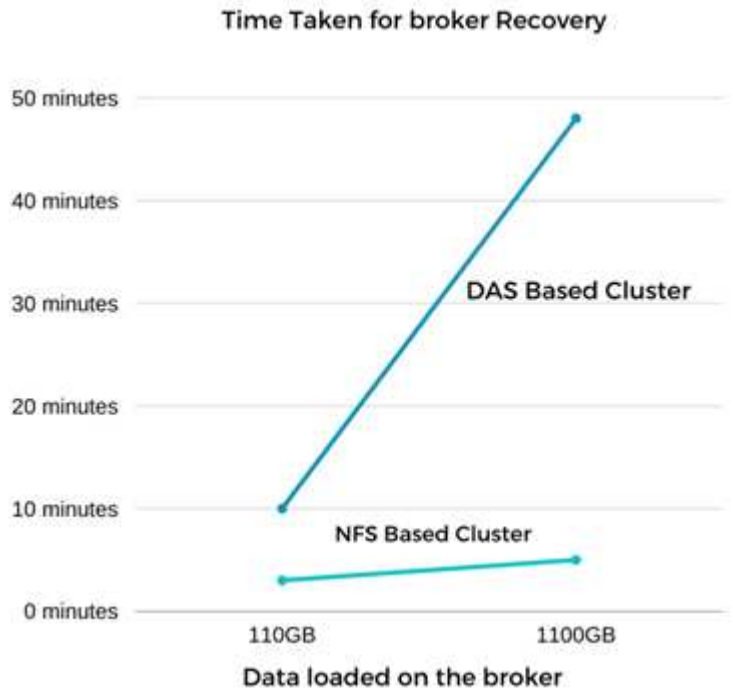
1. バックアップ ノードは 09:39:17,213 に開始されました。開始ログエントリは以下に強調表示されています。

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

```
[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)
```

約 1 TB のデータを含むブローカーに対してテストを繰り返しましたが、DAS の場合は約 48 分、NFS の場合は約 3 分かかりました。結果は次のグラフに示されています。



ストレージ効率

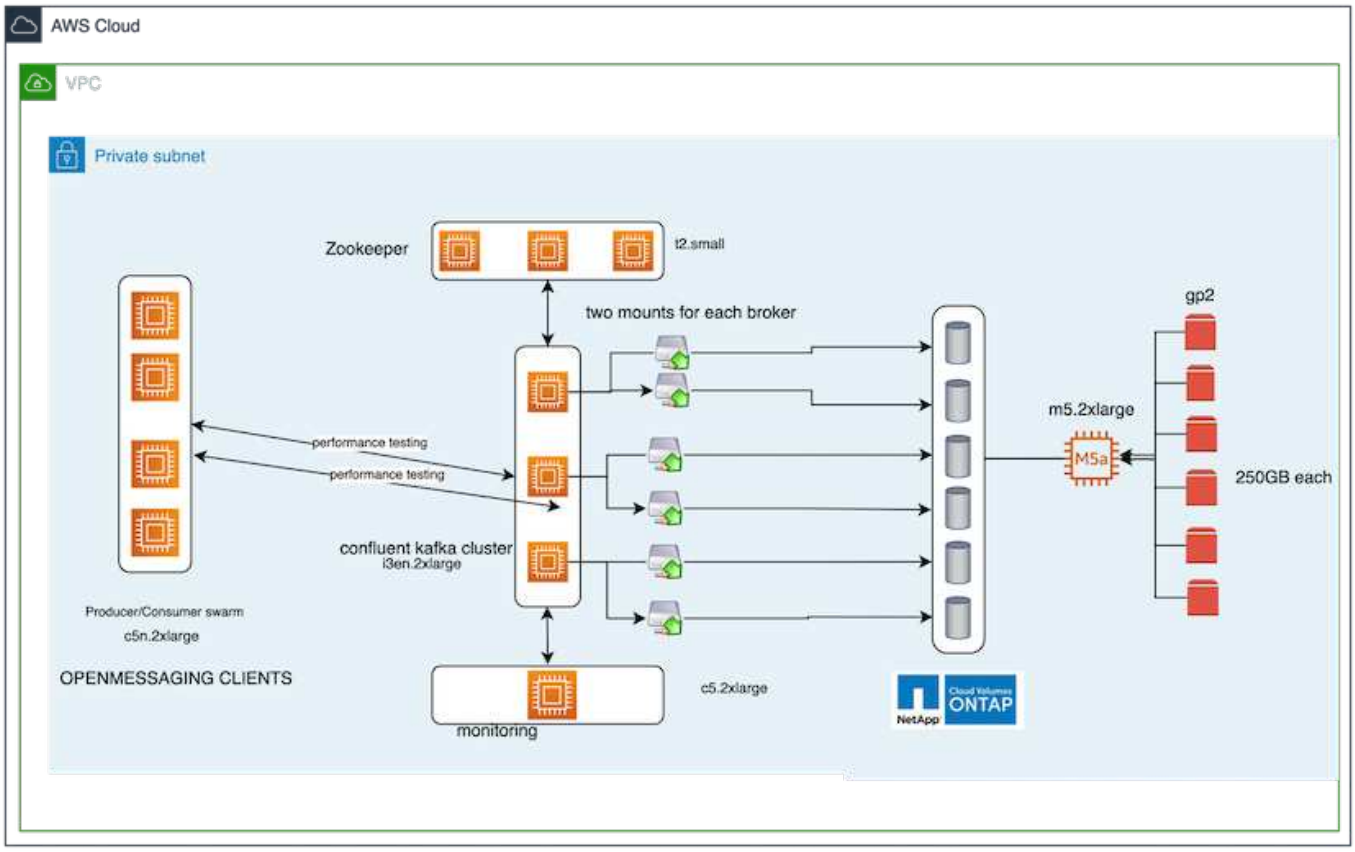
Kafka クラスターのストレージ層はNetApp ONTAPを通じてプロビジョニングされたため、 ONTAPのすべてのストレージ効率機能を利用できました。これは、Cloud Volumes ONTAPでプロビジョニングされた NFS ストレージを使用して Kafka クラスターで大量のデータを生成することによってテストされました。 ONTAP の機能により、スペースが大幅に削減されたことがわかりました。

建築のセットアップ

次の表は、NAS を使用した Kafka クラスターの環境構成を示しています。

プラットフォームコンポーネント	環境設定
カフカ 3.2.3	• 飼育員×3 – t2.small • ブローカーサーバー x 3 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 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	RHEL8.7以降
NetApp Cloud Volumes ONTAPインスタンス	単一ノードインスタンス – M5.2xLarge

次の図は、NAS ベースの Kafka クラスターのアーキテクチャを示しています。



- *計算します。*専用サーバーで実行される 3 ノードの Zookeeper アンサンブルを備えた 3 ノードの Kafka クラスターを使用しました。各ブローカーには、専用 LIF を介してNetApp CVO インスタンス上の単一ボリュームへの 2 つの NFS マウント ポイントがありました。
- 監視。 Prometheus と Grafana の組み合わせには 2 つのノードを使用しました。ワークロードを生成するために、この Kafka クラスターに対して生成と消費が可能な別の 3 ノード クラスターを使用しました。
- *ストレージ。*インスタンスにマウントされた 6 つの 250 GB GP2 AWS-EBS ボリュームを備えた単一ノードのNetApp Cloud Volumes ONTAPインスタンスを使用しました。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 クラスターに公開されました。
- *構成。*このテスト ケースで構成可能な要素は、Kafka ブローカーでした。

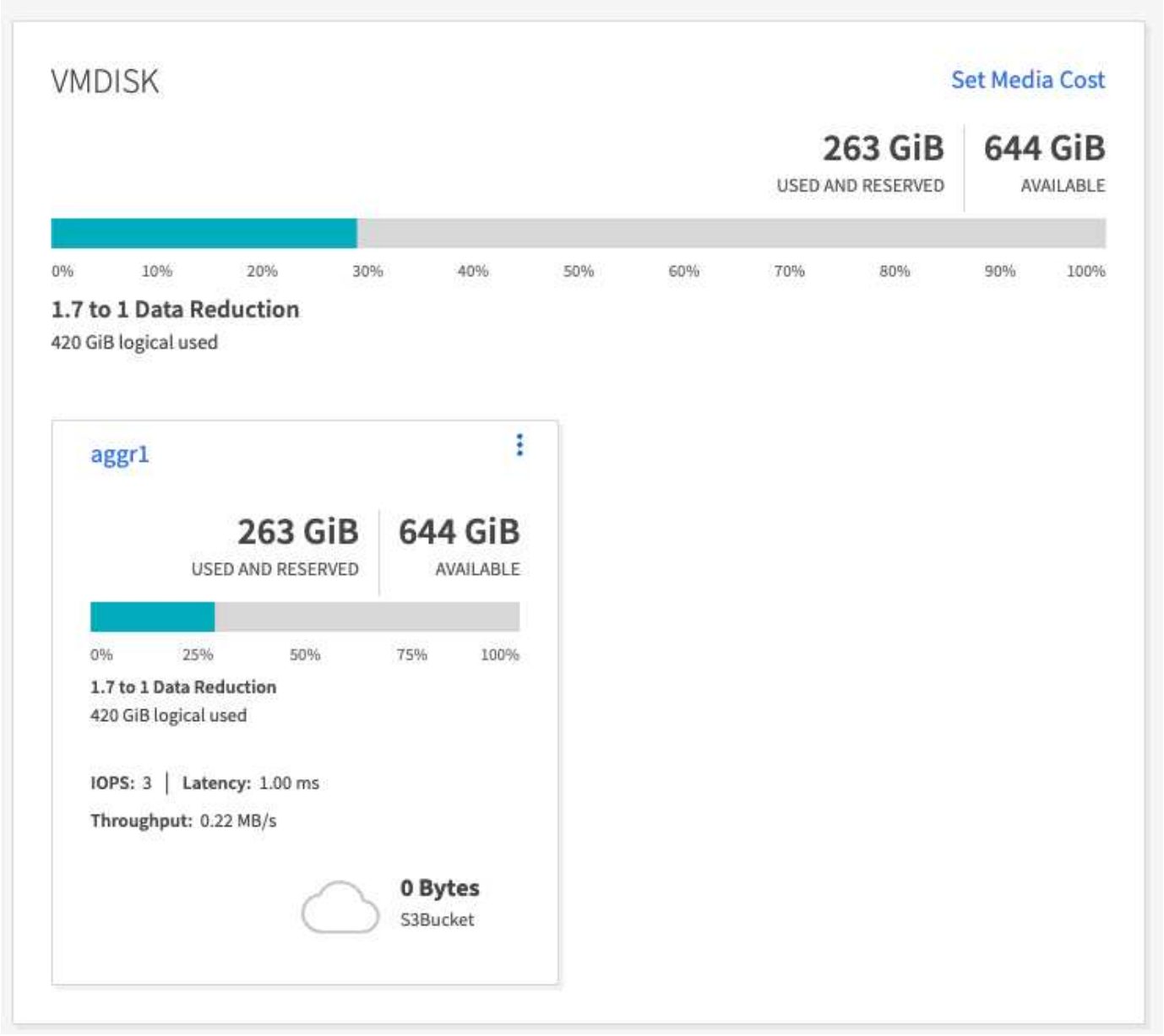
圧縮はプロデューサー側でオフにされたため、プロデューサーは高いスループットを生成できるようになりました。代わりに、ストレージ効率はコンピューティング層によって処理されました。

テストの方法論

1. 上記の仕様で Kafka クラスターがプロビジョニングされました。
2. クラスターでは、OpenMessaging ベンチマーク ツールを使用して約 350 GB のデータが生成されました。
3. ワークロードが完了した後、 ONTAP System Manager と CLI を使用してストレージ効率統計が収集されました。

観察

OMB ツールを使用して生成されたデータでは、ストレージ効率比が 1.70:1 で、スペースが約 33% 節約されました。次の図に示すように、生成されたデータによって使用された論理スペースは 420.3 GB で、データを保持するために使用された物理スペースは 281.7 GB でした。



```
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 でのパフォーマンスの概要と検証

NetApp NFS にマウントされたストレージ層を持つ Kafka クラスターのパフォーマンスを AWS クラウドでベンチマークしました。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。

NetApp Cloud Volumes ONTAPを使用した AWS クラウドでの Kafka (高可用性ペアと単一ノード)

NetApp Cloud Volumes ONTAP (HA ペア) を搭載した Kafka クラスターのパフォーマンスを AWS クラウドでベンチマークしました。このベンチマークについては、次のセクションで説明します。

建築のセットアップ

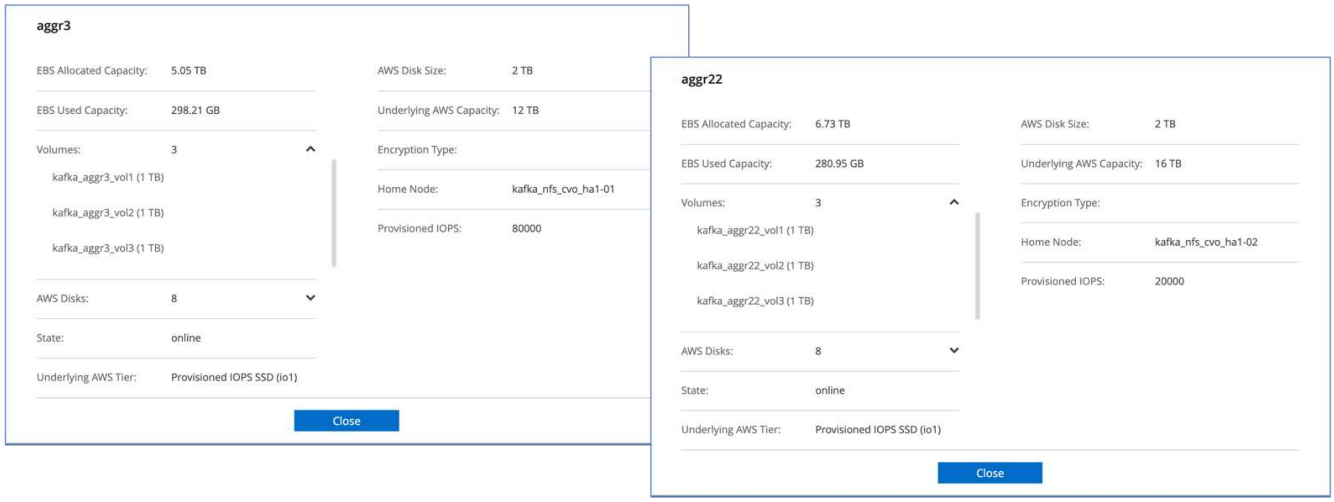
次の表は、NAS を使用した Kafka クラスターの環境構成を示しています。

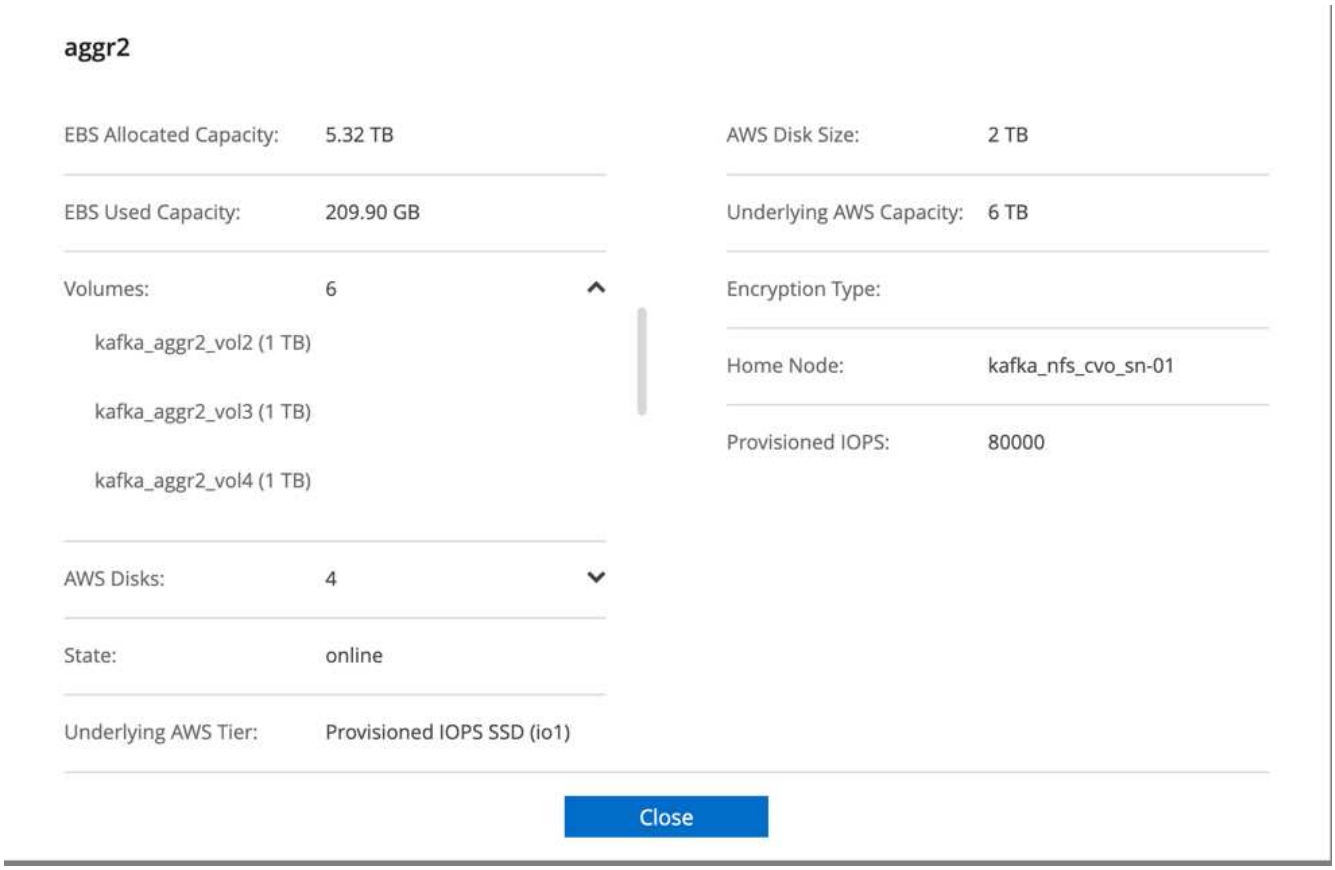
プラットフォームコンポーネント	環境設定
カフカ 3.2.3	• 飼育員×3 – t2.small • ブローカーサーバー x 3 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x プロデューサー/コンシューマー — c5n.2xlarge *
すべてのノード上のオペレーティング システム	RHEL8.6
NetApp Cloud Volumes ONTAPインスタンス	HAペアインスタンス - m5dn.12xLarge x 2ノード シングルノードインスタンス - m5dn.12xLarge x 1ノード

NetAppクラスタボリューム**ONTAP**セットアップ

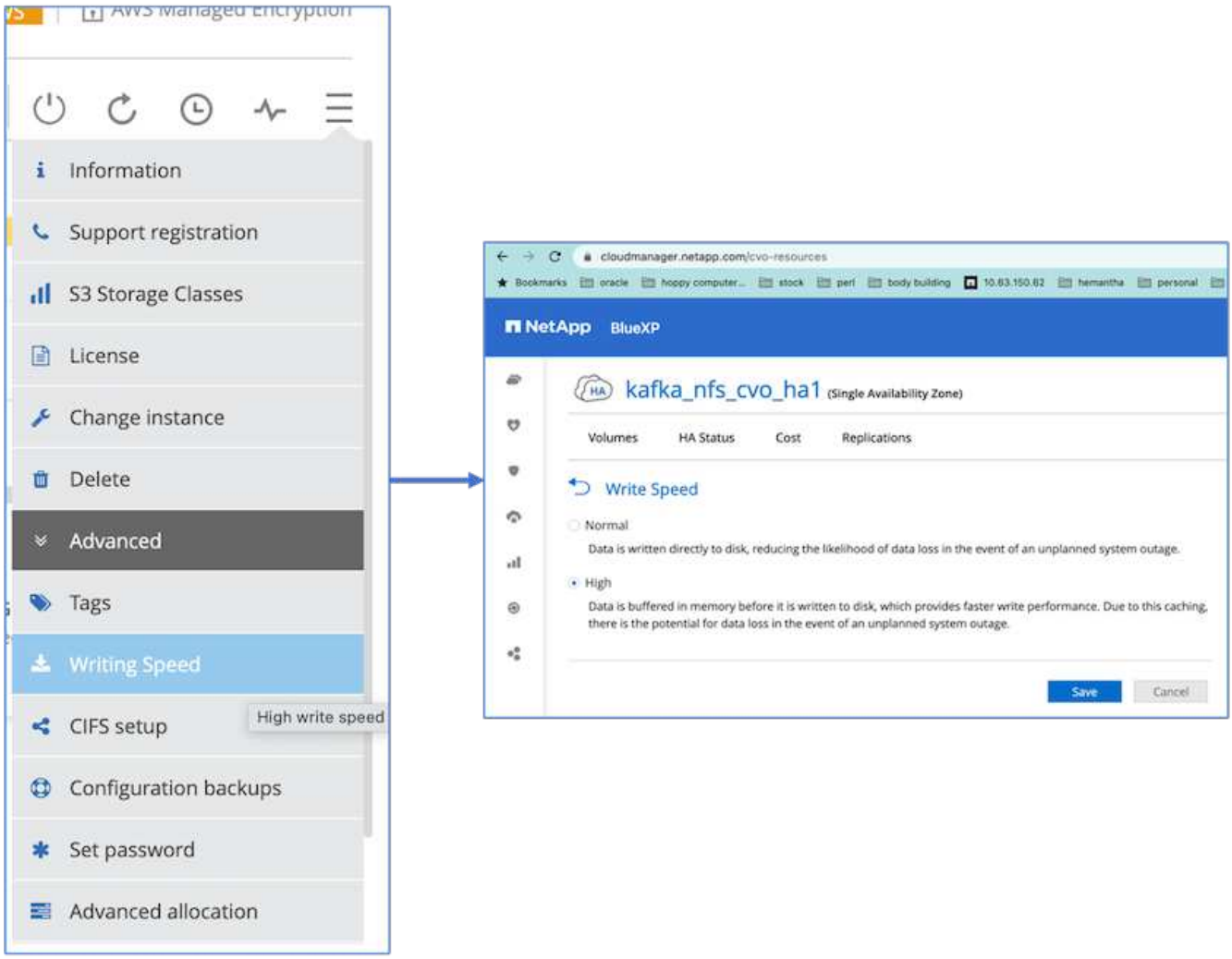
1. Cloud Volumes ONTAP HA ペアの場合、各ストレージ コントローラ上の各アグリゲートに 3 つのボリュ

ームを持つ 2 つのアグリゲートを作成しました。単一のCloud Volumes ONTAPノードに対して、アグリゲート内に 6 つのボリュームを作成します。



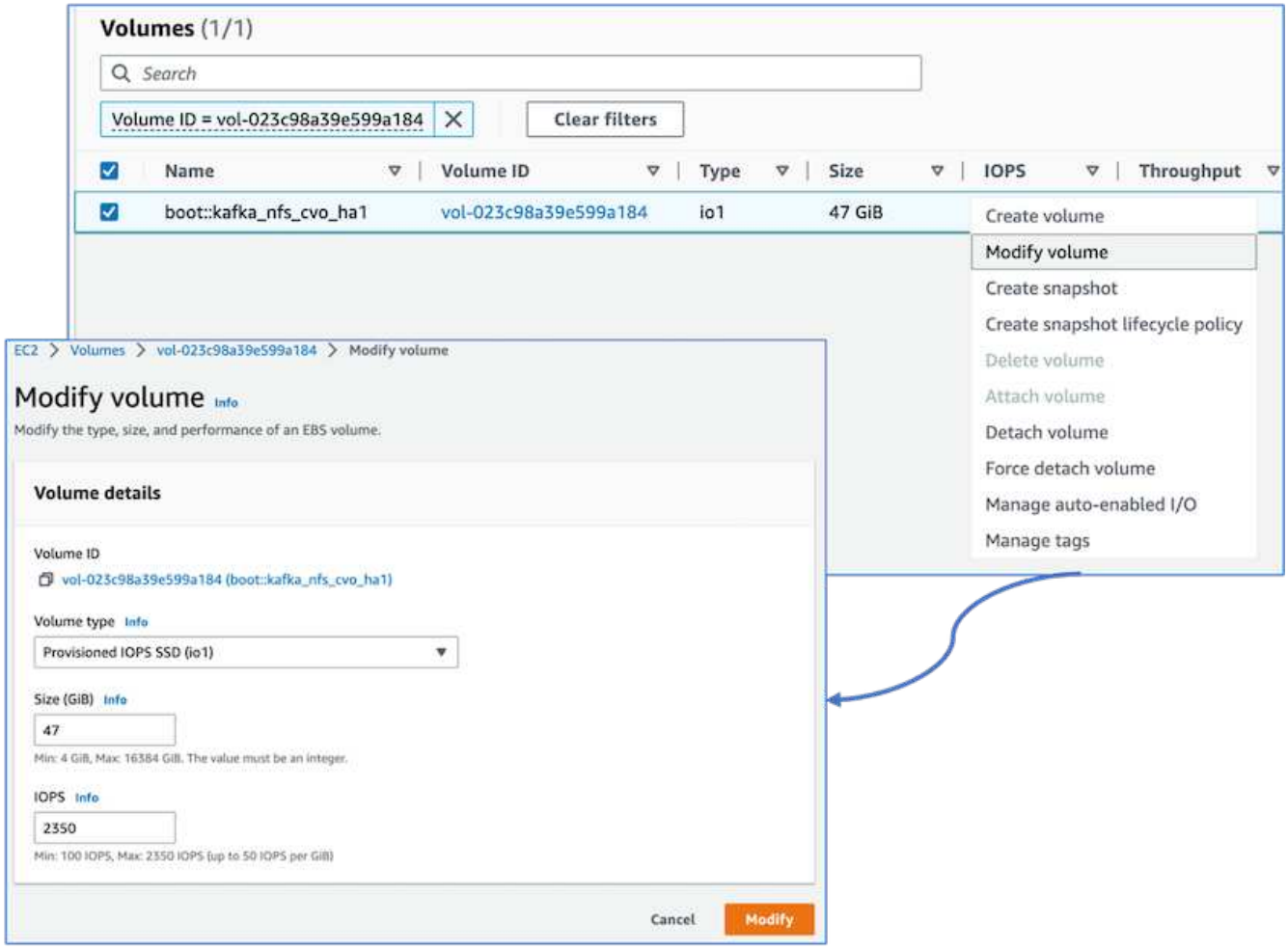


2. より優れたネットワーク パフォーマンスを実現するために、HA ペアと単一ノードの両方で高速ネットワークを有効にしました。



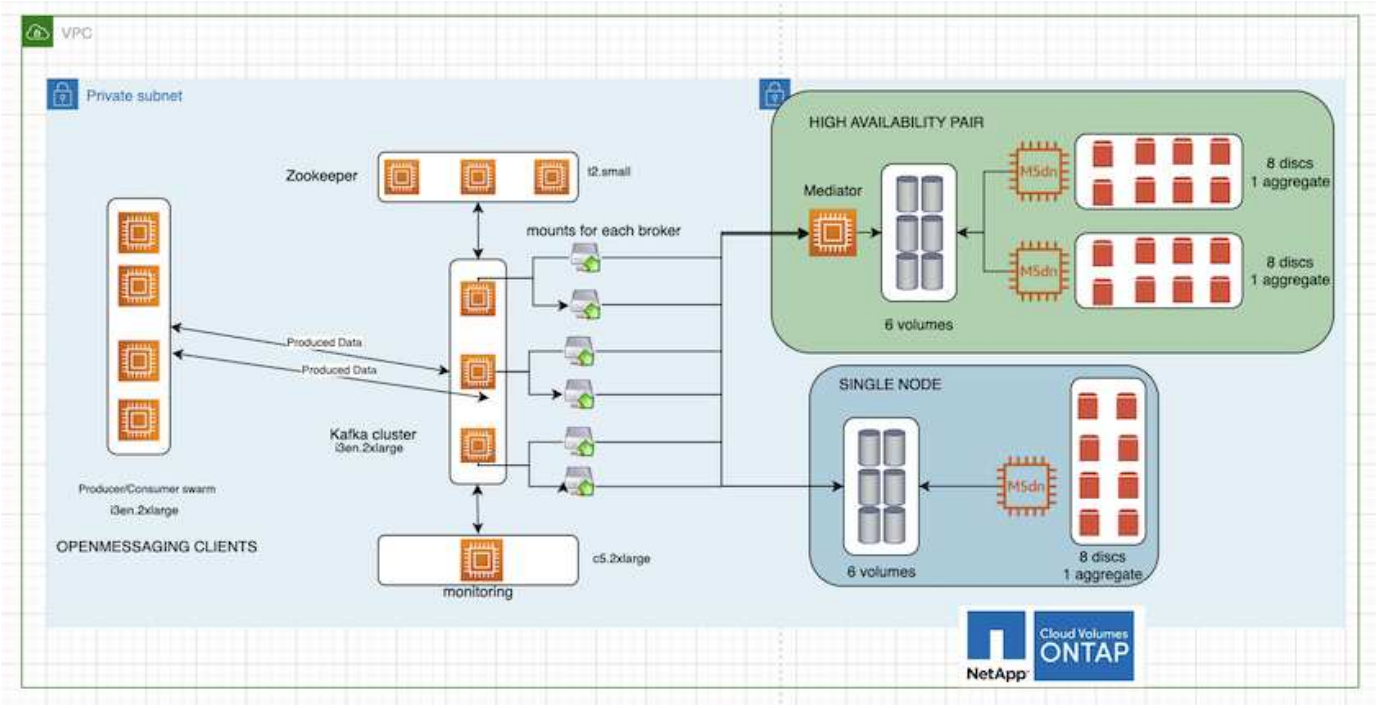
3. ONTAP NVRAM のIOPS が高いことに気づいたので、 Cloud Volumes ONTAPルート ボリュームの IOPS を 2350 に変更しました。 Cloud Volumes ONTAPのルート ボリューム ディスクのサイズは 47 GB でした。次のONTAPコマンドは HA ペア用ですが、同じ手順が単一ノードにも適用されます。

```
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 クラスターのアーキテクチャを示しています。

- *計算します。*専用サーバーで実行される 3 ノードの Zookeeper アンサンブルを備えた 3 ノードの Kafka クラスターを使用しました。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 2 つの NFS マウント ポイントがありました。
- 監視。 Prometheus と Grafana の組み合わせには 2 つのノードを使用しました。ワークロードを生成するために、この Kafka クラスターに対して生成と消費が可能な別の 3 ノード クラスターを使用しました。
- *ストレージ。*インスタンスにマウントされた 6TB GP3 AWS-EBS ボリューム 1 つを備えた HA ペアの Cloud Volumes ONTAPインスタンスを使用しました。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ベンチマーク構成

1. NFS のパフォーマンスを向上させるには、NFS サーバーと NFS クライアント間のネットワーク接続を増やす必要があります。これは、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=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. 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ノードから使用されます。同じ手順がCloud Volumes ONTAP HA ペアにも適用されます。

```
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 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. 以下のKafkaを使用します `server.properties`Cloud Volumes ONTAP HA ペアのすべての Kafka ブローカーで。その `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。ブローカー1の場合、 `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_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 ~]#
```

- ブローカー2の場合、 `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_a
ggr3_vol3/broker2,/kafka_aggr22_vol1/broker2,/kafka_aggr22_vol2/broke
r2,/kafka_aggr22_vol3/broker2
```

- ブローカー3の場合、 `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_a
ggr3_vol3/broker3,/kafka_aggr22_vol1/broker3,/kafka_aggr22_vol2/broke
r3,/kafka_aggr22_vol3/broker3
```

4. 単一のCloud Volumes ONTAPノードの場合、Kafka `servers.properties` Cloud Volumes ONTAP HAペアの場合と同じですが、 `log.dirs`財産。
   - ブローカー1の場合、 `log.dirs`値は次のとおりです。

```
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
```

   - ブローカー2の場合、 `log.dirs`値は次のとおりです。

```
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
```

   - ブローカー3の場合、 `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_a
ggr2_vol3/broker3,/kafka_aggr2_vol4/broker3,/kafka_aggr2_vol5/broker3
,/kafka_aggr2_vol6/broker3
```

5. OMB 内のワークロードは、次のプロパティで構成されます。 `(/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
```

その `messageSize`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。

Kafka クラスターでワークロードを生成するために、OMB の Sync または Throughput という 2 つの異なるドライバーを使用しました。

- 同期ドライバー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=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

- スループットドライバー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. 上記の仕様に従って、Terraform と Ansible を使用して Kafka クラスターがプロビジョニングされました。 Terraform は、Kafka クラスターの AWS インスタンスを使用してインフラストラクチャを構築するために使用され、Ansible はそれら上に Kafka クラスターを構築します。
2. 上記のワークロード構成と同期ドライバを使用して、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のスループット ドライバーで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 上で実行されている 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のワークロードを生成するために、2 つの異なるタイプのドライバーが使用されました。ドライバー間の違いは、ログフラッシュプロパティです。

Cloud Volumes ONTAP HA ペアの場合:

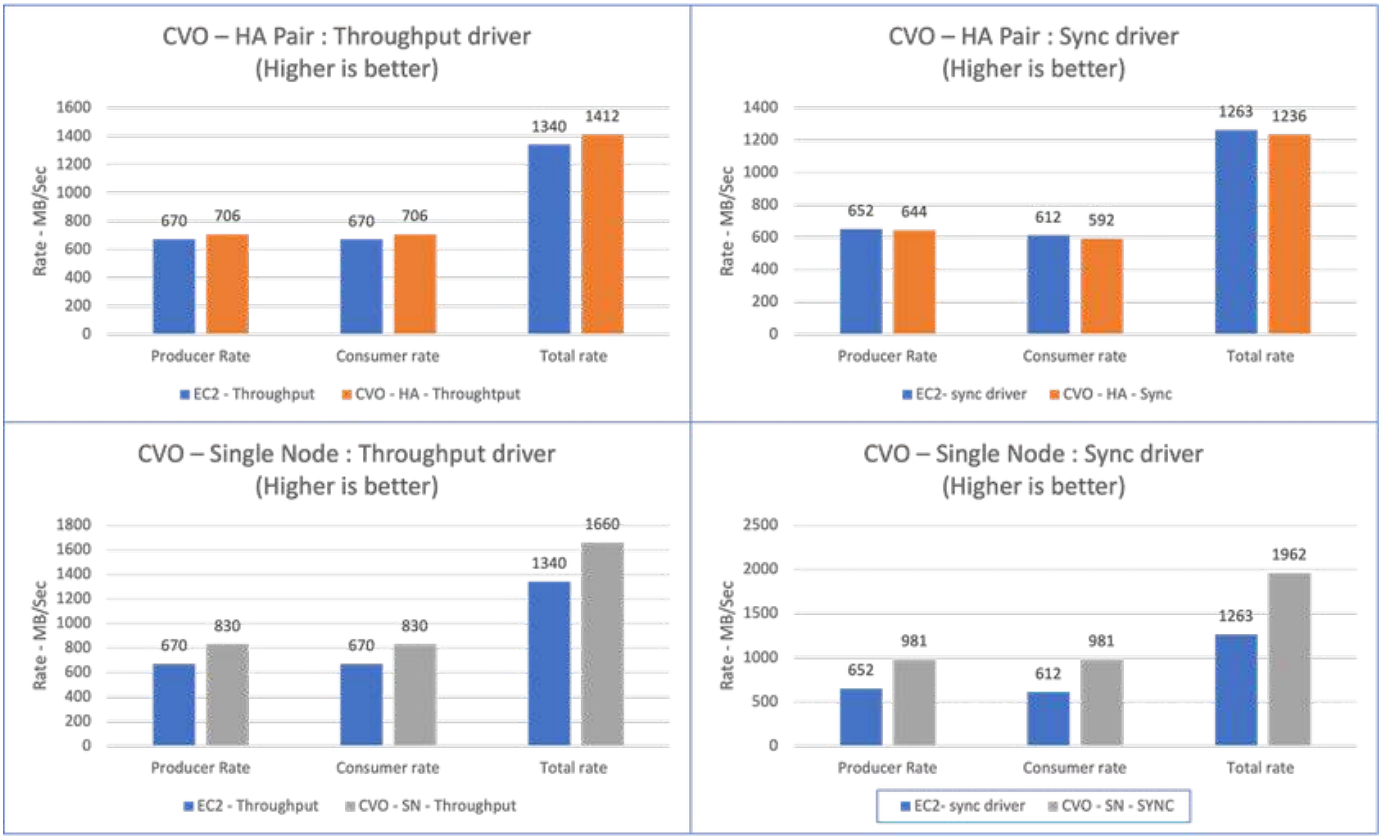
- Sync ドライバーによって一貫して生成される合計スループット: ~1236 MBps。
- スループット ドライバーに対して生成された合計スループット: ピーク時 ~1412 MBps。

単一のCloud Volumes ONTAPノードの場合:

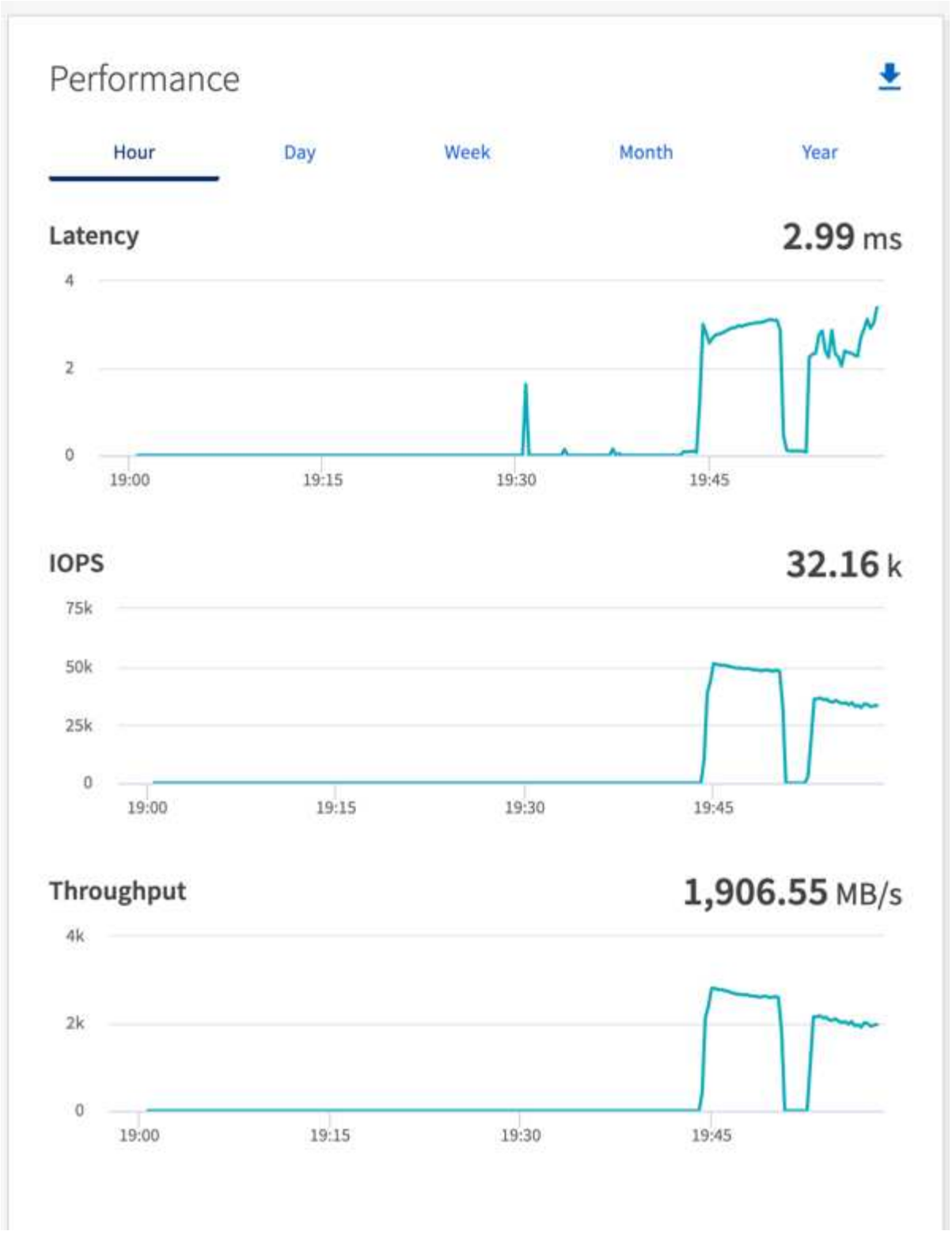
- Sync ドライバーによって一貫して生成される合計スループット: ~ 1962MBps。
- スループットドライバーによって生成される合計スループット: ピーク時約 1660 MBps

Sync ドライバーは、ログがディスクに瞬時にフラッシュされるため一貫したスループットを生成できますが、Throughput ドライバーは、ログが一括してディスクにコミットされるため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数値は、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の場合、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、スループット数値を向上させることができます。合計スループットまたは合計レートは、プロデューサー レートとコンシューマー レートの両方の組み合わせです。



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



AWS FSx ONTAPのパフォーマンスの概要と検証

NetApp NFS にマウントされたストレージ層を持つ Kafka クラスターのパフォーマンスを AWS FSx ONTAPでベンチマークしました。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 ONTAPでの Apache Kafka

ネットワーク ファイル システム (NFS) は、大量のデータを保存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、Apache Kafka などのストリーミング アプリケーションによってデータが生成されることが増えています。これらのワークロードには、スケーラビリティ、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可能にし、実用的な洞察を提供するには、適切に設計された高性能なインフラストラクチャが必要です。

Kafka は設計上、POSIX 準拠のファイル システムで動作し、ファイル操作の処理にはファイル システムに依存しますが、NFSv3 ファイル システムにデータを保存する場合、Kafka ブローカー NFS クライアントは、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ファイル操作を解釈する場合があります。一般的な例としては、クラスターを拡張してパーティションを再割り当てするときに Kafka ブローカーが失敗する原因となった NFS Silly の名前変更が挙げられます。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、現在 RHEL8.7、RHEL9.1 で一般公開されており、現在の FSx ONTAPリリースであるONTAP 9.12.1 からサポートされている変更を加えました。

Amazon FSx ONTAP は、クラウド内で完全に管理され、スケーラブルで高性能な NFS ファイルシステムを提供します。 FSx ONTAP上の 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ために拡張できます。 NFS は、重要な機密データセットの集中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ときに FSx ONTAPを活用できるようになります。これらの利点は次のとおりです。 * CPU 使用率を削減して I/O 待機時間を短縮します。 * Kafka ブローカーの回復時間が短縮されます。 * 信頼性と効率性。 * スケーラビリティとパフォーマンス。 * 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 ONTAPのパフォーマンスの概要と検証

NetApp NFS にマウントされたストレージ層を持つ Kafka クラスターのパフォーマンスを AWS クラウドでベンチマークしました。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 ONTAPでの Kafka

AWS FSx ONTAPを搭載した Kafka クラスターの AWS クラウドでのパフォーマンスをベンチマークしました。このベンチマークについては、次のセクションで説明します。

建築のセットアップ

次の表は、AWS FSx ONTAPを使用した Kafka クラスターの環境構成を示しています。

プラットフォームコンポーネント	環境設定
カフカ 3.2.3	• 飼育員×3 – t2.small • ブローカーサーバー x 3 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 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	RHEL8.6
AWS FSx ONTAP	4GB/秒のスループットと160000 IOPSを備えたマルチAZ

1. 最初のテストでは、2TB の容量と 2GB/秒のスループットで 40000 IOP の FSx 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 を介して FSx ONTAPをデプロイしています。必要に応じて、環境に合わせてコマンドをさらにカスタマイズする必要があります。さらに、FSx ONTAP はAWS コンソールから導入および管理できるため、コマンドライン入力が少なくなり、より簡単かつ合理化された導入エクスペリエンスを実現できます。

ドキュメント FSx ONTAPでは、テスト リージョン (US-East-1) の 2GB/秒スループットのファイル システムで達成可能な最大 IOPS は 80,000 iops です。 FSx ONTAPファイルシステムの合計最大 IOPS は 160,000 IOPS であり、これを実現するには 4GB/秒のスループットの展開が必要です。これについては、このドキュメントの後半で説明します。

FSx ONTAP のパフォーマンス仕様の詳細については、AWS FSx ONTAP のドキュメントをご覧ください。 https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html 。

FSx「create-file-system」の詳細なコマンドライン構文については、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 マスターキーではなく、特定の KMS キーを指定できます。

2. FSx ONTAPファイルシステムを作成するときに、次のようにファイルシステムを記述した後、JSON の戻り値で「LifeCycle」ステータスが「AVAILABLE」に変わるまで待機します。

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

3. fsxadmin ユーザーで FSx ONTAP SSH にログインして資格情報を検証します。Fsxadmin は、作成時の FSx 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、ステップ 1 で完了したように、AWS コンソールまたは AWS CLI を使用して最初にファイルシステムを作成したときに設定されたパスワードです。

```
[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. 資格情報が検証されたら、FSx ONTAPファイルシステム上にストレージ仮想マシンを作成します。

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

ストレージ仮想マシン (SVM) は、FSx 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資格情報とエンドポイントを備えた分離されたファイル サーバーであり、FSx ONTAPマルチテナント機能を提供します。

5. プライマリ ストレージ仮想マシンを構成したら、新しく作成された FSx 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ストレージ仮想マシン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。私たちの検証に基づいて、デフォルトの構成要素 (8) 以下を維持すると、Kafka のパフォーマンスが向上します。

```
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. テストのため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 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、追加のプロビジョニング済み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。

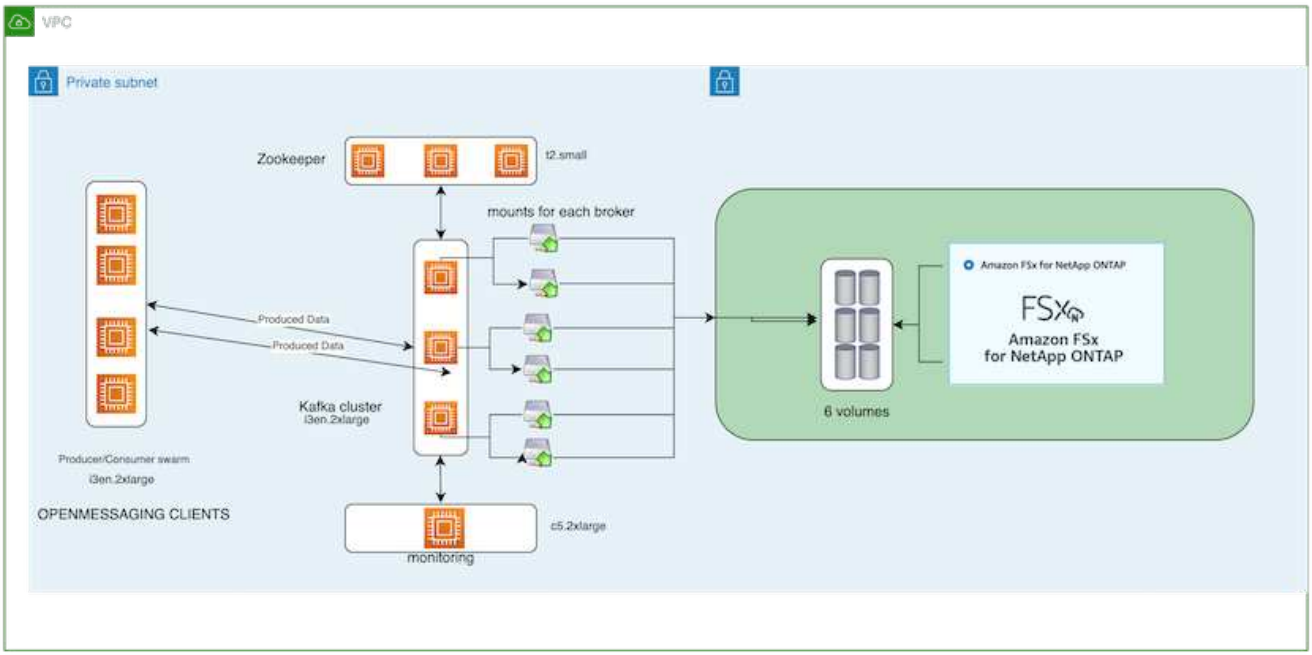
7. 次に、パフォーマンスと容量をさらに向上させるために、FSx ONTAPのスループット容量を2GB/秒から4GB/秒に、IOPSを160000に、容量を5TBに拡張しました。

```
[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,Io
ps=160000}' --file-system-id fs-02ff04bab5ce01c7c
```

FSx「update-file-system」の詳細なコマンドライン構文については、以下を参照してください。https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. FSx ONTAPボリュームは、Kafkaブローカーのnconnectおよびデフォルトオプションでマウントされます。

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



- 計算します。専用サーバーで実行される 3 ノードの Zookeeper アンサンブルを備えた 3 ノードの Kafka クラスターを使用しました。各ブローカーには、FSx ONTAPインスタンス上の 6 つのボリュームへの 6 つの NFS マウント ポイントがありました。
- 監視。 Prometheus と Grafana の組み合わせには 2 つのノードを使用しました。ワークロードを生成するために、この Kafka クラスターに対して生成と消費が可能な別の 3 ノード クラスターを使用しました。
- ストレージ。 2TB ボリュームを 6 つマウントした FSx ONTAPを使用しました。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。FSx ONTAPボリュームは、16 個の nconnect セッションと Kafka ブローカーのデフォルト オプションを使用してマウントされます。

OpenMessage ベンチマーク構成。

NetApp Cloud Volumes ONTAPに使用したのと同じ構成を使用しました。詳細については、こちらを参照してください - xref:./data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

テストの方法論

1. Kafka クラスターは、Terraform と Ansible を使用して、上記の仕様に従ってプロビジョニングされました。 Terraform は、Kafka クラスターの AWS インスタンスを使用してインフラストラクチャを構築するために使用され、Ansible はそれら上に Kafka クラスターを構築します。
2. 上記のワークロード構成と同期ドライバを使用して、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のスループット ドライバーで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 上で実行されている 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のワークロードを生成するために、2 つの異なるタイプのドライバーが使用されました。ドライバー間の違いは、ログフラッシュプロパティです。

Kafka レプリケーション係数 1 および FSx ONTAPの場合:

- Sync ドライバーによって一貫して生成される合計スループット: 約 3218 MBps、ピーク パフォーマンス: 約 3652 MBps。
- スループット ドライバーによって一貫して生成される合計スループット: 約 3679 MBps、ピーク パフォーマンス: 約 3908 MBps。

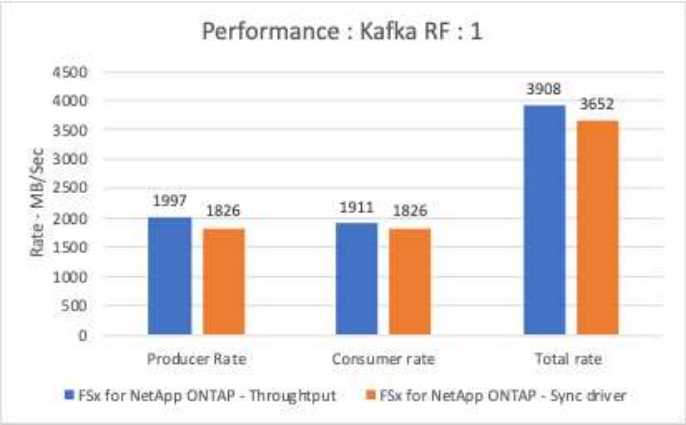
レプリケーション係数 3 および FSx ONTAPを使用した Kafka の場合:

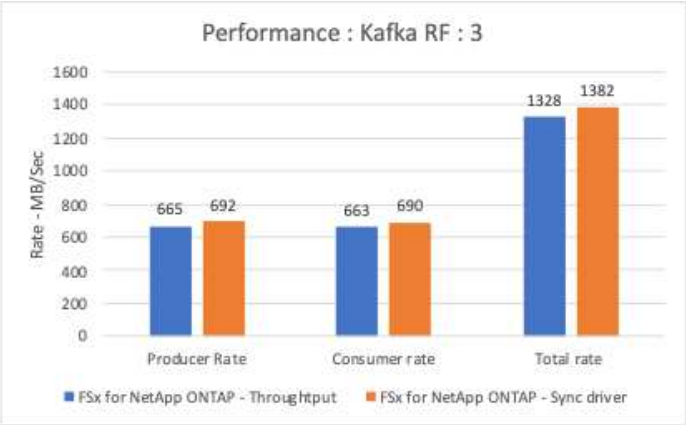
- Sync ドライバーによって一貫して生成される合計スループット: 約 1252 MBps、ピーク パフォーマンス: 約 1382 MBps。
- スループット ドライバーによって一貫して生成される合計スループット: 約 1218 MBps、ピーク パフォーマンス: 約 1328 MBps。

Kafka レプリケーション ファクター 3 では、FSx ONTAPで読み取りおよび書き込み操作が 3 回発生しました。Kafka レプリケーション ファクター 1 では、FSx ONTAPで読み取りおよび書き込み操作が 1 回であるため、両方の検証で最大スループット 4GB/秒に到達できました。

Sync ドライバーは、ログがディスクに瞬時にフラッシュされるため一貫したスループットを生成できますが、Throughput ドライバーは、ログが一括してディスクにコミットされるためスループットのバーストを生成します。

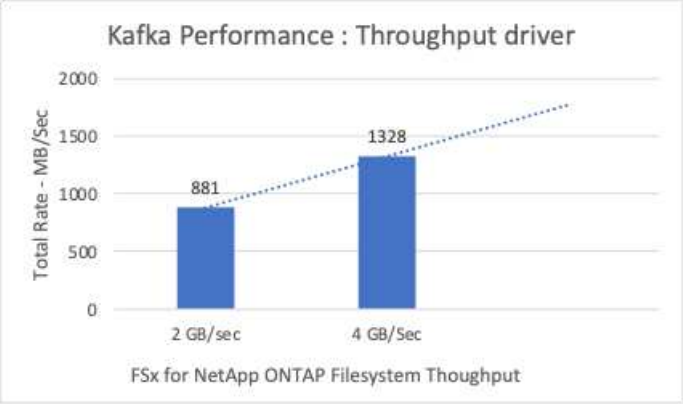
これらのスループット数値は、指定された AWS 構成に対して生成されます。より高いパフォーマンス要件の場合、インスタンス タイプをスケールアップしてさらに調整し、スループット数値を向上させることができます。合計スループットまたは合計レートは、プロデューサー レートとコンシューマー レートの両方の組み合わせです。

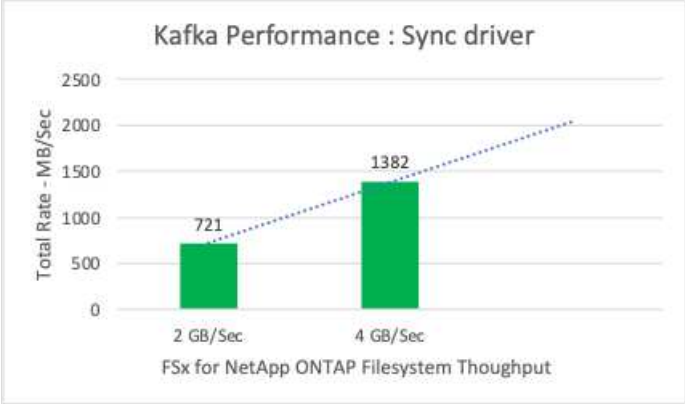




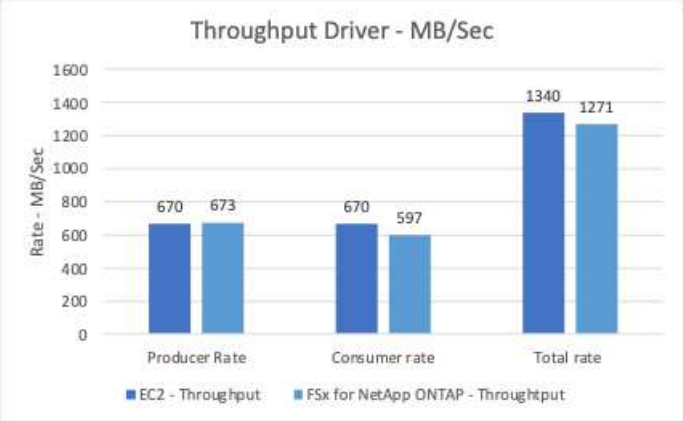
以下のグラフは、Kafka レプリケーション ファクター 3 の 2GB/秒の FSx ONTAPと 4GB/秒のパフォーマンス

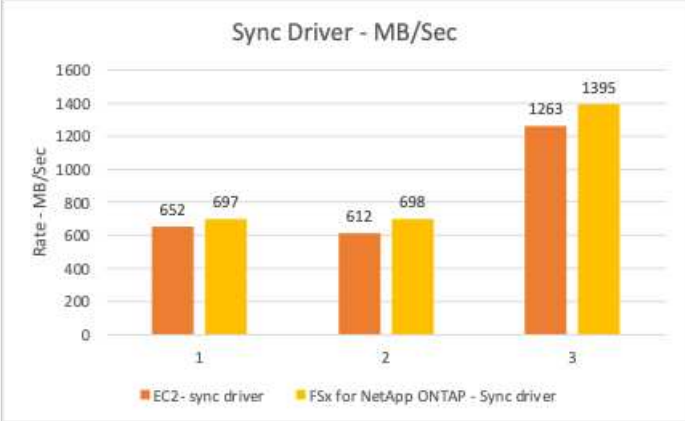
を示しています。レプリケーション係数 3 は、FSx ONTAPストレージで読み取りおよび書き込み操作を 3 回実行します。スループット ドライバーの合計速度は 881 MB/秒で、2 GB/秒の FSx ONTAPファイルシステムで約 2.64 GB/秒の Kafka 操作の読み取りと書き込みを実行します。また、スループット ドライバーの合計速度は 1328 MB/秒で、約 3.98 GB/秒の Kafka 操作の読み取りと書き込みを実行します。 Kafka のパフォーマンスは、FSx ONTAPスループットに基づいて線形かつスケーラブルです。





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

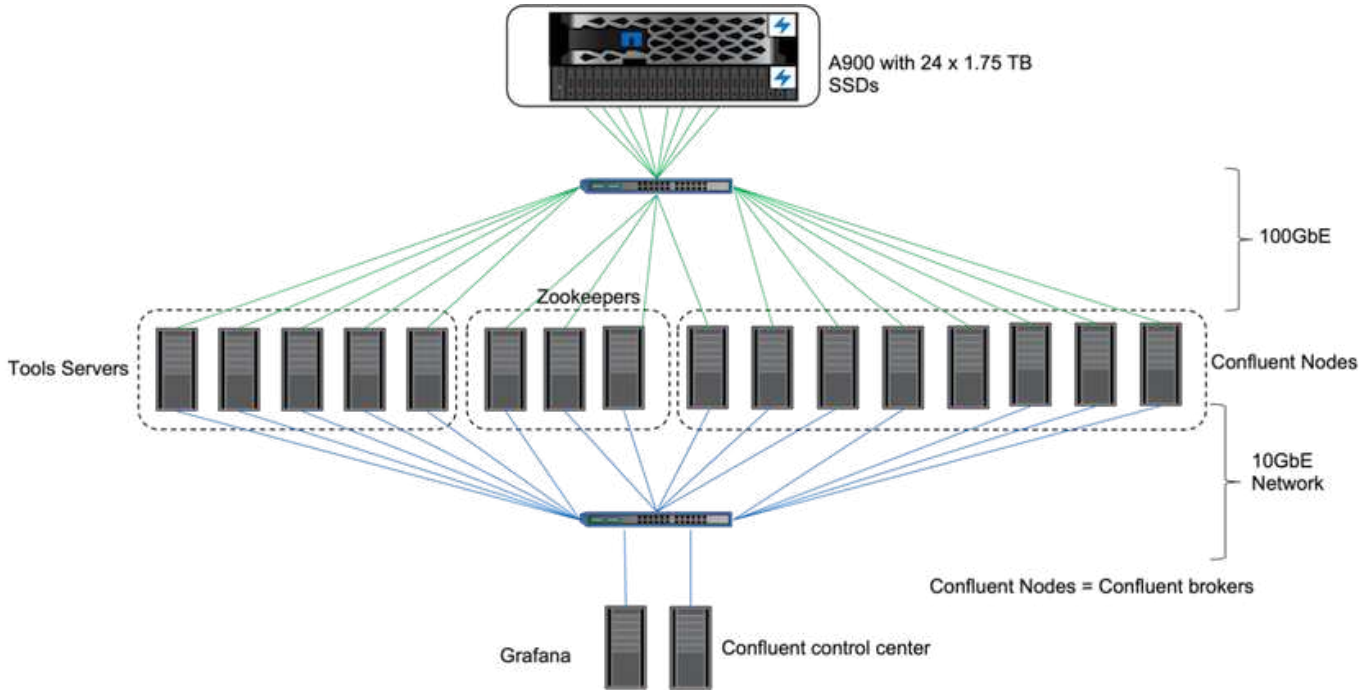




AFF A900オンプレミスのパフォーマンス概要と検証

オンプレミスでは、 ONTAP 9.12.1RC1 を搭載したNetApp AFF A900ストレージ コントローラを使用して、Kafka クラスターのパフォーマンスとスケーリングを検証しました。以前のONTAPおよびAFFを使用した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 を評価するために、Confluent Kafka 6.2.0 を使用しました。クラスターには 8 つのブローカー ノードと 3 つの Zookeeper ノードが含まれます。パフォーマンス テストには、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

NetApp FlexGroups インスタンスを使用して、ログ ディレクトリに単一の名前空間を提供し、リカバリと構成を簡素化しました。ログ セグメント データへの直接パス アクセスを提供するために、NFSv4.1 と pNFS を使用しました。

クライアントのチューニング

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、 `max_session_slots`` デフォルトから `64`に `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ブローカーのチューニング

テスト対象システムのスループットを最大化するために、特定の主要なスレッド プールのデフォルト パラメータを大幅に増加しました。ほとんどの構成では、Confluent Kafka のベスト プラクティスに従うことをお勧めします。このチューニングは、ストレージへの未処理の I/O の同時実行性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論

スループット ドライバー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. FlexGroupインスタンスは、 AFFクラスター上で Ansible を使用してプロビジョニングされました。

```
---
- name: Set up kafka broker processes
  hosts: localhost
  vars:
    ntap_hostname: 'hostname'
    ntap_username: 'user'
    ntap_password: 'password'
    size: 10
    size_unit: tb
    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 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. ワークロードは、Cloud Volumes ONTAPと同じワークロード構成を使用して、スループット ドライバーによってトリガーされました。 「[定常状態のパフォーマンス] " 下に。ワークロードではレプリケーション係数 3 が使用され、ログ セグメントの 3 つのコピーが NFS に保持されました。

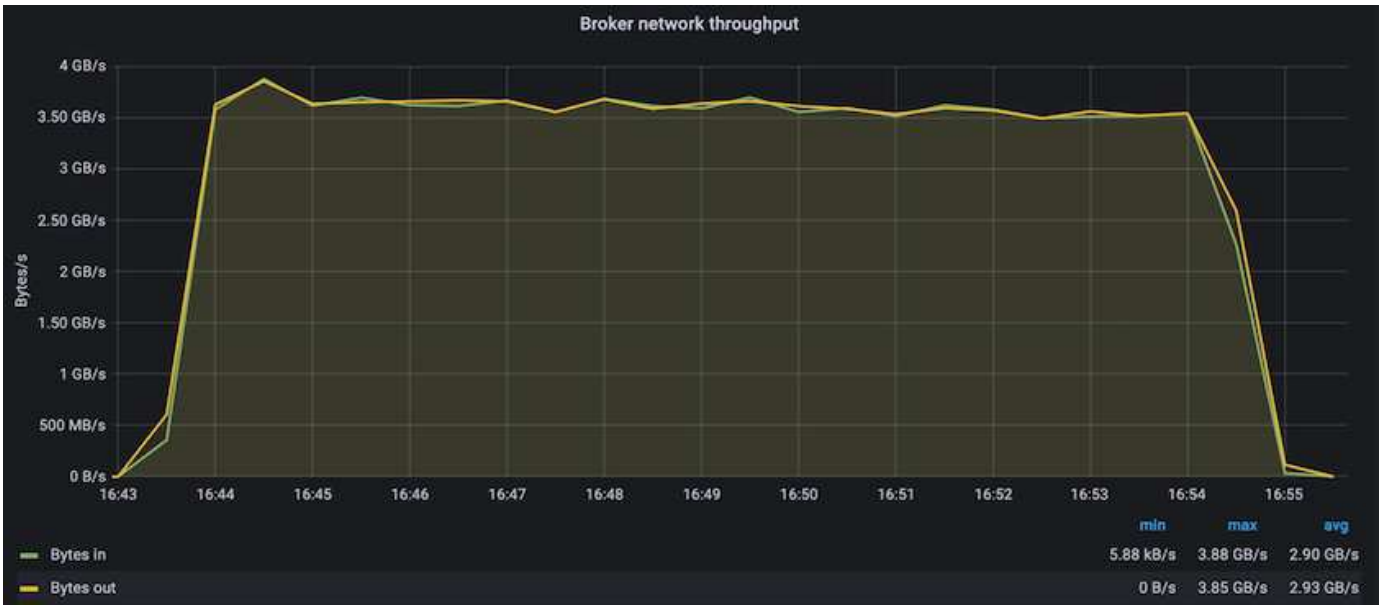
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最後に、バックログを使用して、消費者が最新のメッセージに追いつく能力を測定するための測定を完了しました。 OMB は、測定の開始時にコンシューマーを一時停止することでバックログを構築します。これにより、バックログの作成 (プロデューサーのみのトラフィック)、バックログの排出 (トピック内の見逃されたイベントをコンシューマーが追いつくコンシューマー中心のフェーズ)、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。「[究極のパフォーマンスとストレージの限界の探求]詳細については、「」を参照してください。

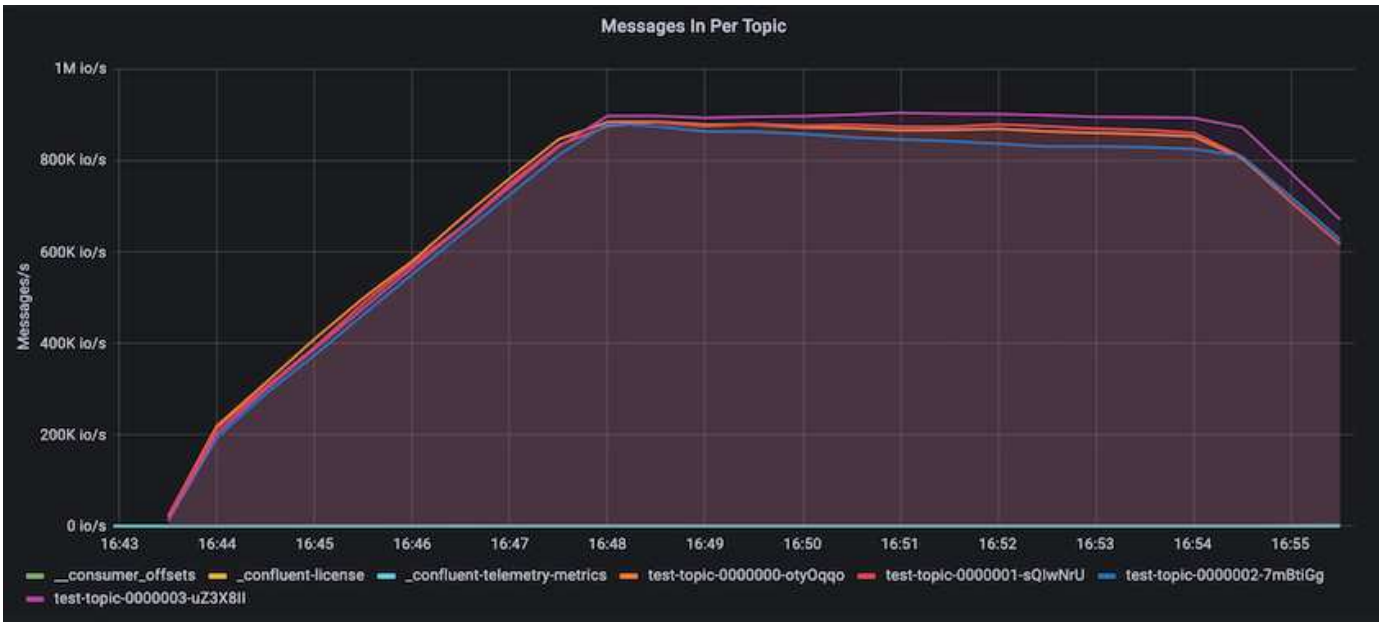
定常状態のパフォーマンス

AWS のCloud Volumes ONTAPと AWS の DAS と同様の比較を行うために、OpenMessaging ベンチマークを使用してAFF A900 を評価しました。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ー レベルでの Kafka クラスター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では、プロデューサーとコンシューマーの両方で 3.4 GBps を超える平均スループットが達成されました。これは、Kafka クラスター全体で 340 万件を超えるメッセージです。 BrokerTopicMetrics の持続スループットをバイト/秒単位で視覚化することで、AFF A900がサポートする優れた安定した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一致します。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 90 万件のメッセージが確認されました。

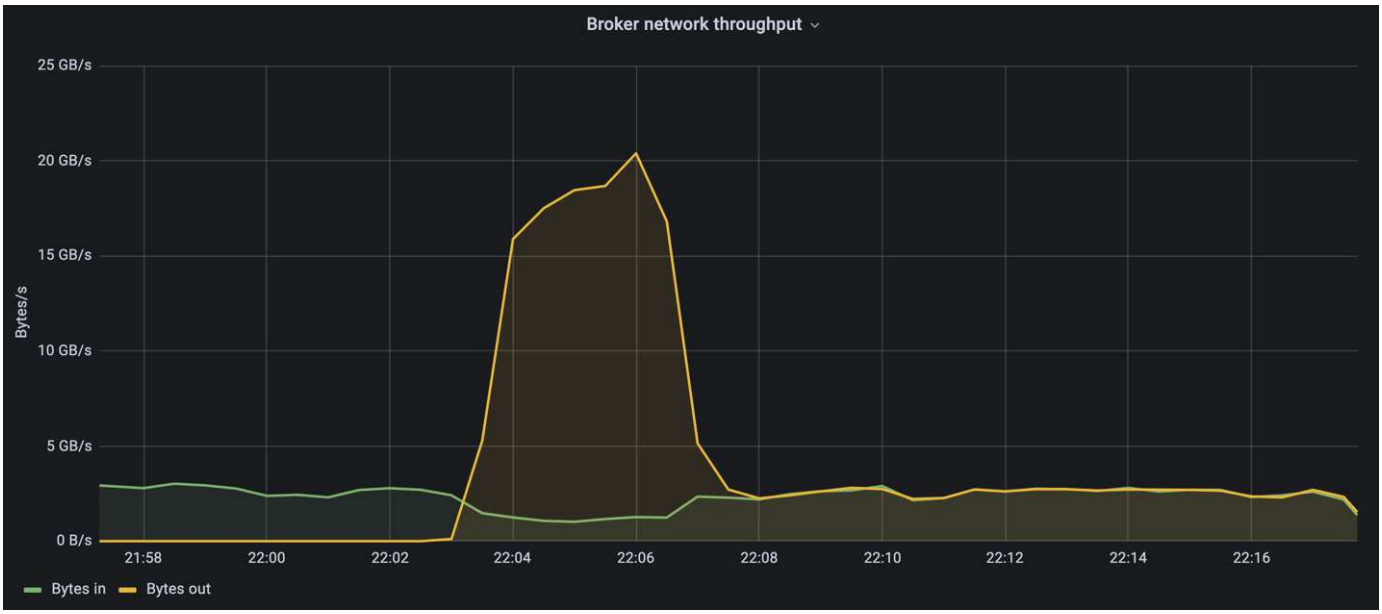


究極のパフォーマンスとストレージの限界の探求

AFFについては、バックログ機能を使用して OMB でもテストしました。バックログ機能は、Kafka クラスターにイベントのバックログが蓄積されている間、コンシューマーの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、プロデューサー トラフィックのみが発生し、ログにコミットされる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よくエミュレートします。これらのワークフローでは、コンシューマー 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。

この構成におけるコンシューマー スループットのストレージ制限を理解するために、プロデューサーのみのフェーズを測定し、A900 が吸収できる書き込みトラフィックの量を把握しました。次のセクションを参照してください。[サイズガイド]このデータをどのように活用するかを理解してください。

この測定のプロデューサーのみの部分では、A900 パフォーマンスの限界を押し上げる高いピーク スループットが確認されました (プロデューサーとコンシューマーのトラフィックを処理する他のブローカー リソースが飽和していない場合)。



この測定では、メッセージごとのオーバーヘッドを制限し、NFS マウント ポイントへのストレージ 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、ピーク時のプロデューサー 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 がイベントバックログの入力を完了すると、コンシューマー トラフィックが再開されました。バックログの排出による測定中に、すべてのトピックにわたって 20 GBps を超えるピーク消費者スループットが観測されました。 OMB ログ データを保存する NFS ボリュームへの合計スループットは、約 30 GBps に近づきました。

サイズガイド

Amazon Web Servicesは "サイズガイド"Kafka クラスターのサイズ設定とスケーリング用。

このサイズ設定は、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ための便利な式を提供します。

レプリケーション係数 r を持つ tcluster のクラスターに生成される集約スループットの場合、ブローカー ストレージが受信するスループットは次のようになります。

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

これをさらに簡略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 ホット層のニーズに適したONTAPプラットフォームを選択できます。

次の表は、さまざまなレプリケーション係数を持つ A900 の予測プロデューサー スループットを示しています。

複製係数	プロデューサースループット（**GPps**）
3（測定値）	3.4
2	5.1
1	10.2

まとめ

厄介な名前変更問題に対するNetAppソリューションは、これまで NFS と互換性がなかったワークロードに対して、シンプルで安価な集中管理型のストレージ形式を提供します。

この新しいパラダイムにより、顧客は災害復旧やデータ保護の目的で移行やミラーリングが容易な、管理しやすい Kafka クラスターを作成できるようになります。また、NFS には、CPU 使用率の低減、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、さらなるメリットがあることもわかりました。

詳細情報の入手方法

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、次のドキュメントや Web サイトを参照してください。

- Apache Kafka とは何ですか?

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

- silly rename とは何ですか?

  "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/"

- NetApp製品ドキュメント

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

- NFS とは何ですか?

  "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"

- OpenMessaging ベンチマークとは何ですか?

  "https://openmessaging.cloud/"

- Kafka ブローカーを移行するにはどうすればよいですか?

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

- Prometheus で Kafka ブローカーを監視するにはどうすればよいですか?

  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

商標に関する情報