Apache Kafka-Workloads mit NetApp NFS-Speicher

NetApp artificial intelligence solutions

NetApp
February 12, 2026

Inhalt

Apache Kafka-Workloads mit NetApp NFS-Speicher

TR-4947: Apache Kafka-Workload mit NetApp NFS-Speicher – Funktionale Validierung und Leistung

Shantanu Chakole, Karthikeyan Nagalingam und Joe Scott, NetApp

Kafka ist ein verteiltes Publish-Subscribe-Messaging-System mit einer robusten Warteschlange, die große Mengen an Nachrichtendaten aufnehmen kann. Mit Kafka können Anwendungen Daten sehr schnell in Themen schreiben und daraus lesen. Aufgrund seiner Fehlertoleranz und Skalierbarkeit wird Kafka im Big Data-Bereich häufig als zuverlässige Methode zum schnellen Aufnehmen und Verschieben vieler Datenströme verwendet. Zu den Anwendungsfällen gehören Stream-Verarbeitung, Website-Aktivitätsverfolgung, Erfassung und Überwachung von Metriken, Protokollaggregation, Echtzeitanalysen usw.

Obwohl normale Kafka-Operationen auf NFS gut funktionieren, führt das dumme Umbenennungsproblem beim Ändern der Größe oder Neupartitionieren eines auf NFS laufenden Kafka-Clusters zum Absturz der Anwendung. Dies ist ein erhebliches Problem, da die Größe eines Kafka-Clusters zum Lastenausgleich oder zu Wartungszwecken geändert oder neu partitioniert werden muss. Weitere Details finden Sie "hier," .

In diesem Dokument werden die folgenden Themen beschrieben:

- Das alberne Umbenennungsproblem und die Lösungsvalidierung
- Reduzierung der CPU-Auslastung zur Verkürzung der E/A-Wartezeit
- Schnellere Wiederherstellungszeit des Kafka-Brokers
- Leistung in der Cloud und vor Ort

Warum NFS-Speicher für Kafka-Workloads verwenden?

Kafka-Workloads in Produktionsanwendungen können riesige Datenmengen zwischen Anwendungen streamen. Diese Daten werden in den Kafka-Broker-Knoten im Kafka-Cluster gehalten und gespeichert. Kafka ist außerdem für seine Verfügbarkeit und Parallelität bekannt, die es durch die Aufteilung von Themen in Partitionen und die anschließende Replikation dieser Partitionen im gesamten Cluster erreicht. Dies bedeutet letztendlich, dass sich die enorme Datenmenge, die durch einen Kafka-Cluster fließt, im Allgemeinen vervielfacht. NFS ermöglicht eine Neugewichtung der Daten bei Änderungen der Anzahl der Broker sehr schnell und einfach. Bei großen Umgebungen ist die Neuverteilung der Daten über DAS bei einer Änderung der Brokeranzahl sehr zeitaufwändig, und in den meisten Kafka-Umgebungen ändert sich die Anzahl der Broker häufig.

Zu den weiteren Vorteilen zählen:

- **Reife.** NFS ist ein ausgereiftes Protokoll, was bedeutet, dass die meisten Aspekte seiner Implementierung, Sicherung und Verwendung gut verstanden sind.
- **Offen.** NFS ist ein offenes Protokoll und seine Weiterentwicklung ist in Internetspezifikationen als freies und offenes Netzwerkprotokoll dokumentiert.

- **Kostengünstig.** NFS ist eine kostengünstige Lösung für die gemeinsame Nutzung von Netzwerkdateien, die einfach einzurichten ist, da sie die vorhandene Netzwerkinfrastruktur nutzt.
- **Zentral verwaltet.** Durch die zentrale Verwaltung von NFS verringert sich der Bedarf an zusätzlicher Software und Speicherplatz auf den Systemen einzelner Benutzer.
- **Verteilt.** NFS kann als verteiltes Dateisystem verwendet werden, wodurch der Bedarf an Wechselmedienspeichergeräten reduziert wird.

Warum NetApp für Kafka-Workloads?

Die NetApp NFS-Implementierung gilt als Goldstandard für das Protokoll und wird in zahllosen Enterprise-NAS-Umgebungen verwendet. Neben der Glaubwürdigkeit von NetApp bietet es auch die folgenden Vorteile:

- Zuverlässigkeit und Effizienz
- Skalierbarkeit und Leistung
- Hohe Verfügbarkeit (HA-Partner in einem NetApp ONTAP Cluster)
- Datenschutz
  - **Notfallwiederherstellung (NetApp SnapMirror).** Ihre Site ist ausgefallen oder Sie möchten auf einer anderen Site weitermachen und dort fortfahren, wo Sie aufgehört haben.
  - Verwaltbarkeit Ihres Speichersystems (Administration und Management mit NetApp OnCommand).
  - **Lastausgleich.** Der Cluster ermöglicht Ihnen den Zugriff auf verschiedene Volumes von Daten-LIFs, die auf verschiedenen Knoten gehostet werden.
  - **Unterbrechungsfreier Betrieb.** LIFs oder Volume-Verschiebungen sind für die NFS-Clients transparent.

NetApp -Lösung für das dumme Umbenennungsproblem für NFS-zu-Kafka-Workloads

Kafka wird unter der Annahme erstellt, dass das zugrunde liegende Dateisystem POSIX-kompatibel ist: beispielsweise XFS oder Ext4. Durch die Neuverteilung der Kafka-Ressourcen werden Dateien entfernt, während die Anwendung sie noch verwendet. Ein POSIX-kompatibles Dateisystem ermöglicht die Fortsetzung der Verknüpfungsaufhebung. Die Datei wird jedoch erst entfernt, wenn alle Verweise auf die Datei verschwunden sind. Wenn das zugrunde liegende Dateisystem an das Netzwerk angeschlossen ist, fängt der NFS-Client die Unlink-Aufrufe ab und verwaltet den Workflow. Da für die Datei, deren Verknüpfung aufgehoben wird, noch Öffnungsvorgänge ausstehen, sendet der NFS-Client eine Umbenennungsanforderung an den NFS-Server und führt beim letzten Schließen der aufgehobenen Datei einen Entfernungsvorgang für die umbenannte Datei aus. Dieses Verhalten wird allgemein als „NFS Silly Rename“ bezeichnet und wird vom NFS-Client orchestriert.

Jeder Kafka-Broker, der Speicher von einem NFSv3-Server verwendet, stößt aufgrund dieses Verhaltens auf Probleme. Das NFSv4.x-Protokoll verfügt jedoch über Funktionen zur Behebung dieses Problems, indem es dem Server ermöglicht, die Verantwortung für die geöffneten, nicht verknüpften Dateien zu übernehmen. NFS-Server, die diese optionale Funktion unterstützen, teilen dem NFS-Client beim Öffnen der Datei die Eigentumsrechte mit. Der NFS-Client beendet dann die Verwaltung der Verknüpfungsaufhebung, wenn noch Öffnungen ausstehen, und überlässt dem Server die Verwaltung des Datenflusses. Obwohl die NFSv4-

Spezifikation Richtlinien für die Implementierung bereitstellt, gab es bisher keine bekannten NFS-Serverimplementierungen, die diese optionale Funktion unterstützten.

Um das Problem der dummen Umbenennung zu beheben, sind für den NFS-Server und den NFS-Client die folgenden Änderungen erforderlich:

- **Änderungen am NFS-Client (Linux).** Beim Öffnen der Datei antwortet der NFS-Server mit einem Flag, das die Fähigkeit anzeigt, die Verknüpfung geöffneter Dateien aufzuheben. Durch Änderungen auf der NFS-Clientseite kann der NFS-Server die Aufhebung der Verknüpfung bei Vorhandensein des Flags handhaben. NetApp hat den Open-Source-Linux-NFS-Client mit diesen Änderungen aktualisiert. Der aktualisierte NFS-Client ist jetzt allgemein in RHEL8.7 und RHEL9.1 verfügbar.
- **Änderungen am NFS-Server.** Der NFS-Server verfolgt die Öffnungen. Das Aufheben der Verknüpfung einer vorhandenen geöffneten Datei wird jetzt vom Server verwaltet, um der POSIX-Semantik zu entsprechen. Wenn die letzte Öffnung geschlossen ist, leitet der NFS-Server das eigentliche Entfernen der Datei ein und vermeidet so den albernen Umbenennungsprozess. Der ONTAP NFS-Server hat diese Funktion in seiner neuesten Version, ONTAP 9.12.1, implementiert.

Mit den oben genannten Änderungen am NFS-Client und -Server kann Kafka alle Vorteile des netzwerkgebundenen NFS-Speichers sicher nutzen.

Funktionale Validierung – Dumme Umbenennungskorrektur

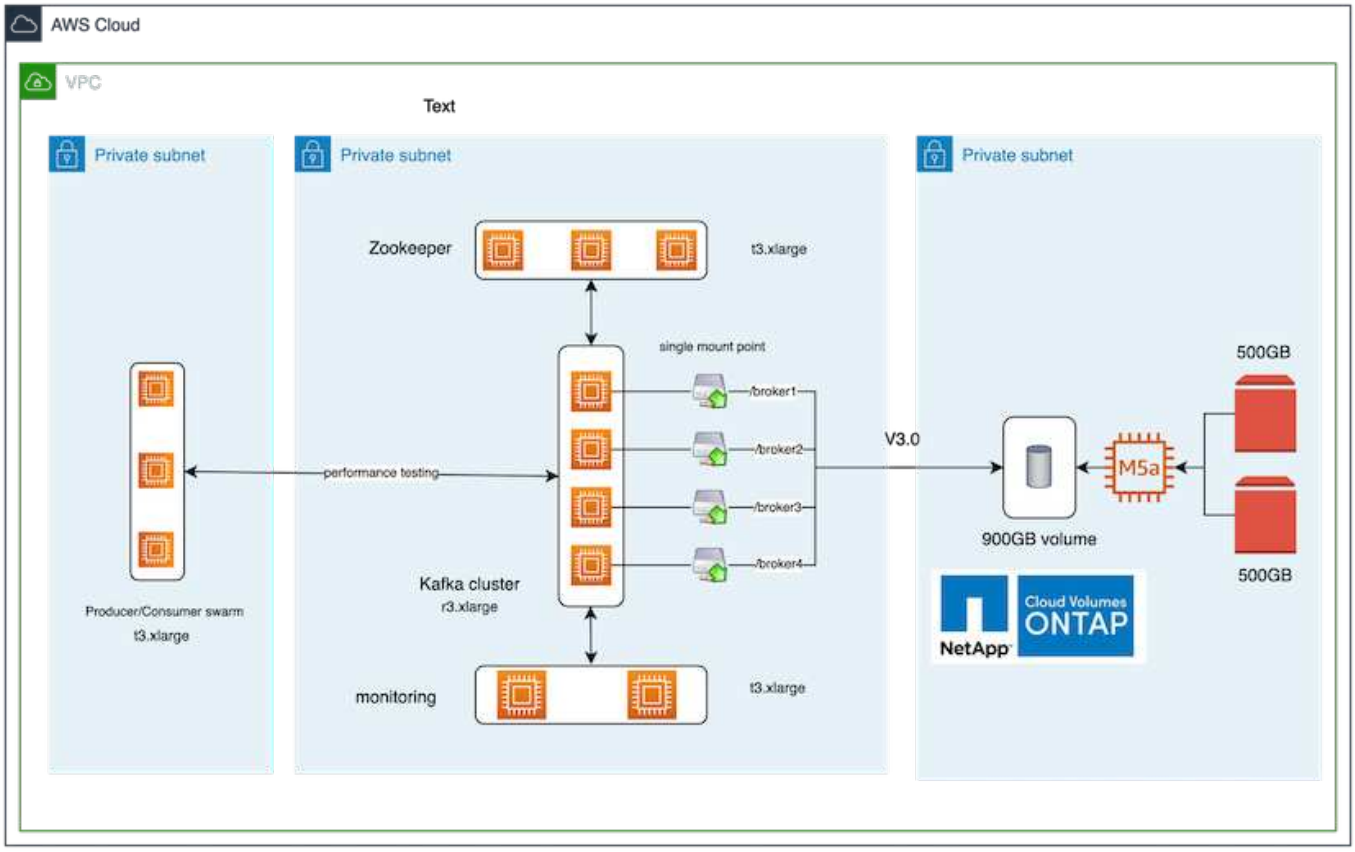
Zur Funktionsvalidierung haben wir gezeigt, dass ein Kafka-Cluster mit einer NFSv3-Einbindung für den Speicher keine Kafka-Operationen wie die Partitionsumverteilung durchführen kann, während ein anderer, mit dem Fix auf NFSv4 eingebundener Cluster dieselben Operationen ohne Unterbrechungen durchführen kann.

Validierungs-Setup

Das Setup wird auf AWS ausgeführt. Die folgende Tabelle zeigt die verschiedenen Plattformkomponenten und Umgebungskonfigurationen, die für die Validierung verwendet wurden.

Plattformkomponente	Umgebungskonfiguration
Confluent Platform Version 7.2.1	• 3 x Tierpfleger – t3.xlarge • 4 x Broker-Server – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x Kontrollzentrum – t3.xlarge • 3 x Produzent/Konsument
Betriebssystem auf allen Knoten	RHEL8.7 oder höher
NetApp Cloud Volumes ONTAP Instanz	Einzelknoteninstanz – M5.2xLarge

Die folgende Abbildung zeigt die Architekturkonfiguration für diese Lösung.



Architektonischer Fluss

- **Berechnen.** Wir haben einen Kafka-Cluster mit vier Knoten und einem Zookeeper-Ensemble mit drei Knoten verwendet, das auf dedizierten Servern ausgeführt wird.
- **Überwachung.** Wir haben zwei Knoten für eine Prometheus-Grafana-Kombination verwendet.
- **Arbeitsbelastung.** Zum Generieren von Workloads haben wir einen separaten Cluster mit drei Knoten verwendet, der für diesen Kafka-Cluster produzieren und von diesem konsumieren kann.
- **Lagerung.** Wir haben eine NetApp Cloud Volumes ONTAP Instanz mit einem Knoten verwendet, an die zwei 500 GB große GP2 AWS-EBS-Volumes angeschlossen waren. Diese Volumes wurden dann über ein LIF als einzelnes NFSv4.1-Volume dem Kafka-Cluster zugänglich gemacht.

Für alle Server wurden die Standardeigenschaften von Kafka gewählt. Dasselbe wurde für den Zoowärterschwarm getan.

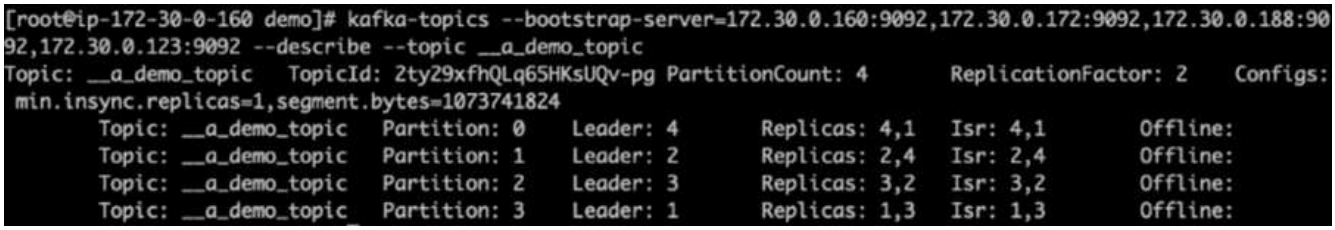
Testmethodik

1. Aktualisieren `-is-preserve-unlink-enabled true` zum Kafka-Band, wie folgt:

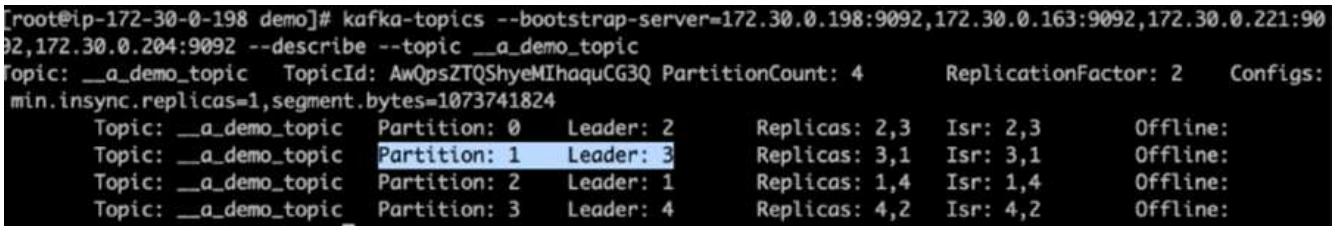
```
aws-shantanclastrecall-aws::*> volume create -vserver kafka_svm -volume
kafka_fg_vol01 -aggregate kafka_aggr -size 3500GB -state online -policy
kafka_policy -security-style unix -unix-permissions 0777 -junction-path
/kafka_fg_vol01 -type RW -is-preserve-unlink-enabled true
[Job 32] Job succeeded: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

2. Es wurden zwei ähnliche Kafka-Cluster mit folgendem Unterschied erstellt:
   - **Cluster 1.** Der Backend-NFS v4.1-Server mit der produktionsbereiten ONTAP Version 9.12.1 wurde von einer NetApp CVO-Instanz gehostet. Auf den Brokern wurden RHEL 8.7/RHEL 9.1 installiert.
   - **Cluster 2.** Der Backend-NFS-Server war ein manuell erstellter generischer Linux-NFSv3-Server.
3. Auf beiden Kafka-Clustern wurde ein Demothema erstellt.

   Cluster 1:



   Cluster 2:



4. In diese neu erstellten Themen wurden für beide Cluster Daten geladen. Dies wurde mithilfe des Producer-Perf-Test-Toolkits durchgeführt, das im Standardpaket von Kafka enthalten ist:

```
./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. Für Broker-1 wurde für jeden der Cluster per Telnet eine Integritätsprüfung durchgeführt:
   - Telnet `172.30.0.160 9092`
   - Telnet `172.30.0.198 9092`

     Eine erfolgreiche Integritätsprüfung für Broker auf beiden Clustern wird im nächsten Screenshot angezeigt:

6. Um den Fehlerzustand auszulösen, der zum Absturz von Kafka-Clustern mit NFSv3-Speichervolumes führt, haben wir den Prozess zur Neuzuweisung der Partitionen auf beiden Clustern eingeleitet. Die Neuzuweisung der Partitionen erfolgte mit `kafka-reassign-partitions.sh` . Der detaillierte Ablauf ist wie folgt:
   a. Um die Partitionen für ein Thema in einem Kafka-Cluster neu zuzuweisen, haben wir die vorgeschlagene JSON-Konfiguration für die Neuzuweisung generiert (dies wurde für beide Cluster durchgeführt).

```
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. Das generierte Neuzuweisungs-JSON wurde dann gespeichert in `/tmp/reassignment-file.json` .
   c. Der eigentliche Partitionsneuzuweisungsprozess wurde durch den folgenden Befehl ausgelöst:

```
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. Einige Minuten nach Abschluss der Neuzuweisung zeigte eine weitere Integritätsprüfung der Broker, dass bei Clustern mit NFSv3-Speichervolumes ein dummes Umbenennungsproblem aufgetreten war und diese abgestürzt waren, während Cluster 1 mit NetApp ONTAP NFSv4.1-Speichervolumes und dem Fix den Betrieb ohne Unterbrechungen fortsetzte.

```
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
```

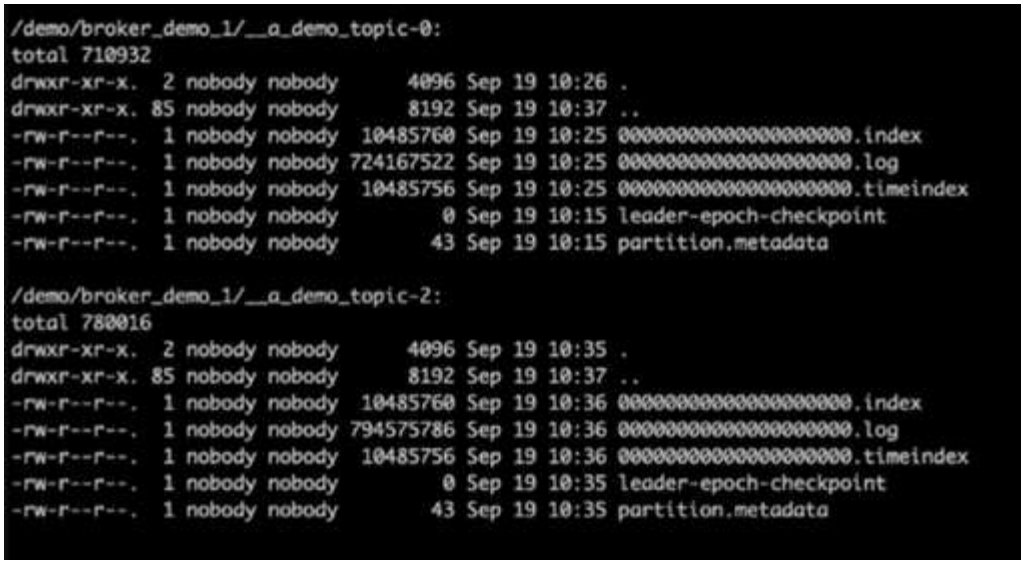
- Cluster1-Broker-1 ist aktiv.
- Cluster2-Broker-1 ist tot.

8. Beim Überprüfen der Kafka-Protokollverzeichnisse war klar, dass Cluster 1, der NetApp ONTAP NFSv4.1-Speichervolumes mit dem Fix verwendet, über eine saubere Partitionszuweisung verfügte, während dies bei Cluster 2, der generischen NFSv3-Speicher verwendet, aufgrund von dummen Umbenennungsproblemen, die zum Absturz führten, nicht der Fall war. Das folgende Bild zeigt die Neuverteilung der Partitionen von Cluster 2, die zu einem dummen Umbenennungsproblem im NFSv3-Speicher führte.

```
/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
```

Das folgende Bild zeigt eine saubere Neuverteilung der Partitionen von Cluster 1 unter Verwendung von NetApp NFSv4.1-Speicher.



Warum NetApp NFS für Kafka-Workloads?

Da es jetzt eine Lösung für das alberne Umbenennungsproblem im NFS-Speicher mit Kafka gibt, können Sie robuste Bereitstellungen erstellen, die NetApp ONTAP -Speicher für Ihre Kafka-Workload nutzen. Dies reduziert nicht nur den Betriebsaufwand erheblich, sondern bringt Ihren Kafka-Clustern auch die folgenden Vorteile:

- **Reduzierte CPU-Auslastung bei Kafka-Brokern.** Durch die Verwendung disaggregierter NetApp ONTAP -Speicher werden Festplatten-E/A-Vorgänge vom Broker getrennt und so dessen CPU-Bedarf reduziert.
- **Schnellere Wiederherstellungszeit des Brokers.** Da der disaggregierte NetApp ONTAP Speicher über alle Kafka-Broker-Knoten hinweg gemeinsam genutzt wird, kann eine neue Compute-Instanz einen fehlerhaften Broker jederzeit in einem Bruchteil der Zeit ersetzen, die bei herkömmlichen Kafka-Bereitstellungen benötigt wird, ohne dass die Daten neu erstellt werden müssen.
- **Speichereffizienz.** Da die Speicherebene der Anwendung jetzt über NetApp ONTAP bereitgestellt wird, können Kunden alle Vorteile der Speichereffizienz von ONTAP nutzen, wie beispielsweise Inline-Datenkomprimierung, Deduplizierung und Kompaktierung.

Diese Vorteile wurden in Testfällen getestet und validiert, die wir in diesem Abschnitt ausführlich besprechen.

Reduzierte CPU-Auslastung auf dem Kafka-Broker

Wir haben festgestellt, dass die allgemeine CPU-Auslastung niedriger ist als beim DAS-Gegenstück, als wir ähnliche Workloads auf zwei separaten Kafka-Clustern ausführten, die in ihren technischen Spezifikationen identisch waren, sich aber in ihren Speichertechnologien unterschieden. Wenn der Kafka-Cluster ONTAP Speicher verwendet, ist nicht nur die allgemeine CPU-Auslastung geringer, sondern auch der Anstieg der CPU-Auslastung weist einen sanfteren Verlauf auf als in einem DAS-basierten Kafka-Cluster.

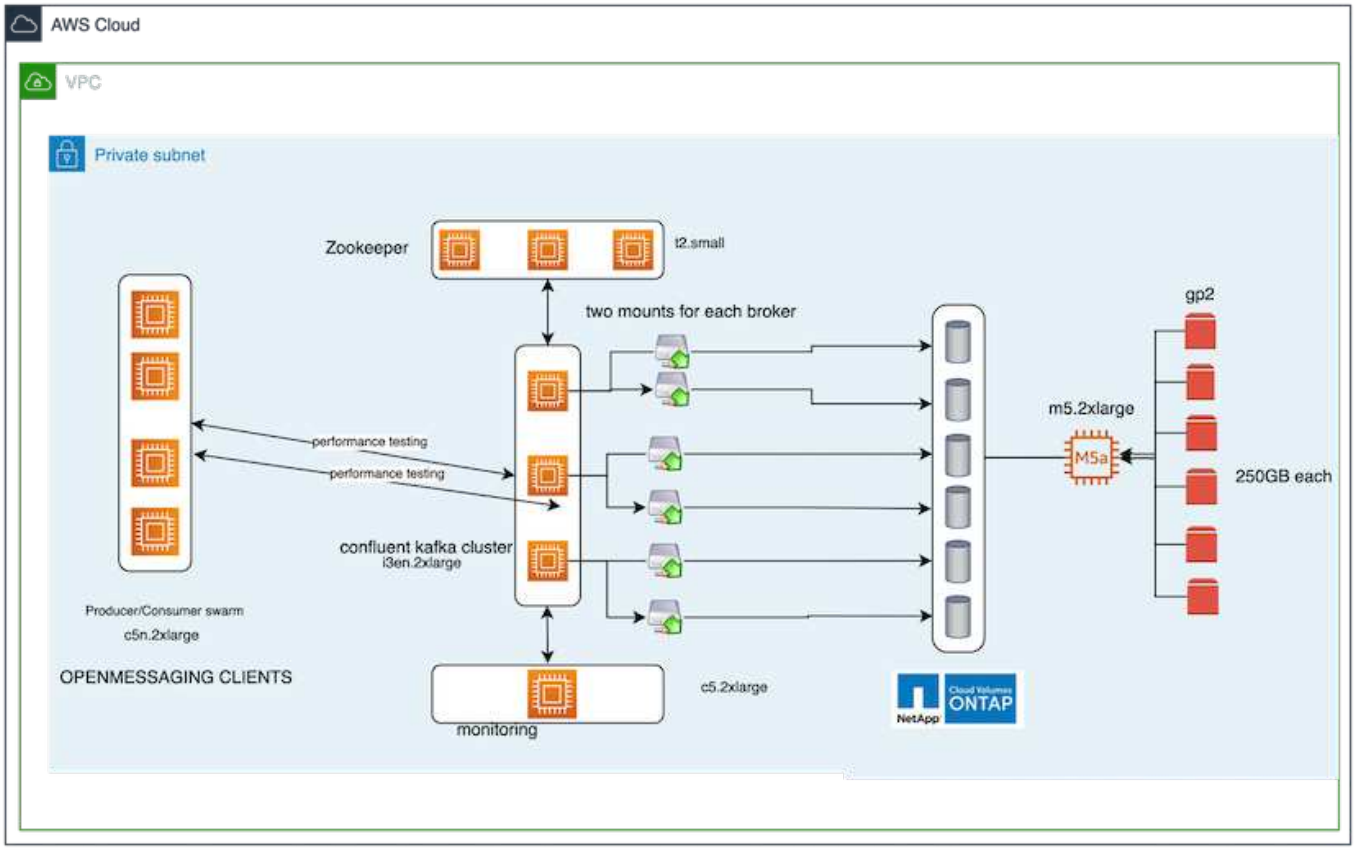
Architektonischer Aufbau

Die folgende Tabelle zeigt die Umgebungskonfiguration, die verwendet wurde, um eine reduzierte CPU-Auslastung zu demonstrieren.

Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3 Benchmarking-Tool: OpenMessaging	• 3 x Tierpfleger – t2.small • 3 x Broker-Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Produzent/Verbraucher — c5n.2xlarge
Betriebssystem auf allen Knoten	RHEL 8.7 oder höher
NetApp Cloud Volumes ONTAP Instanz	Einzelknoteninstanz – M5.2xLarge

Benchmarking-Tool

Das in diesem Testfall verwendete Benchmarking-Tool ist das "OpenMessaging" Rahmen. OpenMessaging ist anbieter- und sprachunabhängig; es bietet Branchenrichtlinien für Finanzen, E-Commerce, IoT und Big Data und unterstützt die Entwicklung von Messaging- und Streaming-Anwendungen über heterogene Systeme und Plattformen hinweg. Die folgende Abbildung zeigt die Interaktion von OpenMessaging-Clients mit einem Kafka-Cluster.



- **Berechnen.** Wir haben einen Kafka-Cluster mit drei Knoten und einem Zookeeper-Ensemble mit drei Knoten verwendet, das auf dedizierten Servern ausgeführt wird. Jeder Broker verfügte über zwei NFSv4.1-Mount-Punkte zu einem einzelnen Volume auf der NetApp CVO-Instanz über ein dediziertes LIF.
- **Überwachung.** Wir haben zwei Knoten für eine Prometheus-Grafana-Kombination verwendet. Zum Generieren von Workloads verfügen wir über einen separaten Cluster mit drei Knoten, der für diesen Kafka-Cluster produzieren und von diesem konsumieren kann.

- **Lagerung.** Wir haben eine NetApp Cloud Volumes ONTAP Instanz mit einem Knoten und sechs auf der Instanz gemounteten 250 GB GP2 AWS-EBS-Volumes verwendet. Diese Volumes wurden dann dem Kafka-Cluster als sechs NFSv4.1-Volumes über dedizierte LIFs zugänglich gemacht.
- **Konfiguration.** Die beiden konfigurierbaren Elemente in diesem Testfall waren Kafka-Broker und OpenMessaging-Workloads.
  - **Broker-Konfiguration.** Für die Kafka-Broker wurden folgende Spezifikationen gewählt. Wir haben für alle Messungen einen Replikationsfaktor von 3 verwendet, wie unten hervorgehoben.

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

- **OpenMessaging-Benchmark (OMB)-Workload-Konfiguration.** Die folgenden Spezifikationen wurden bereitgestellt. Wir haben eine Zielproduzentenrate festgelegt, die unten hervorgehoben ist.

```
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
```

Testmethodik

1. Es wurden zwei ähnliche Cluster erstellt, die jeweils über einen eigenen Satz von Benchmarking-Cluster-Schwärmen verfügten.

- **Cluster 1.** NFS-basierter Kafka-Cluster.
- **Cluster 2.** DAS-basierter Kafka-Cluster.

2. Mithilfe eines OpenMessaging-Befehls wurden auf jedem Cluster ähnliche Workloads ausgelöst.

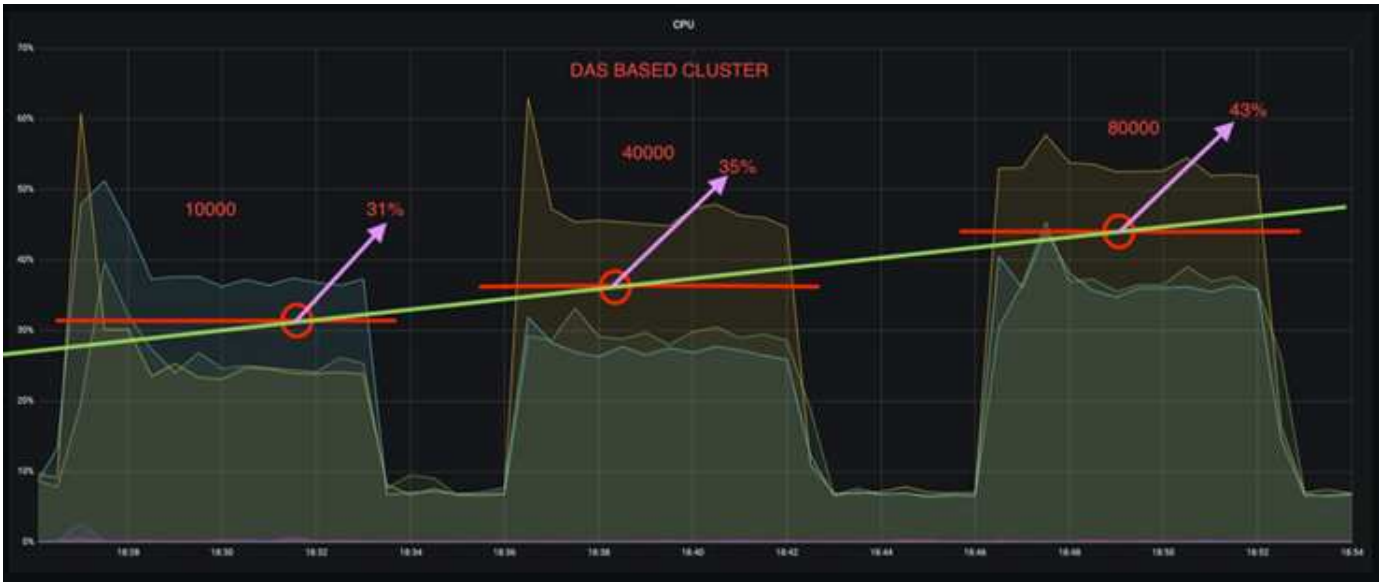
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

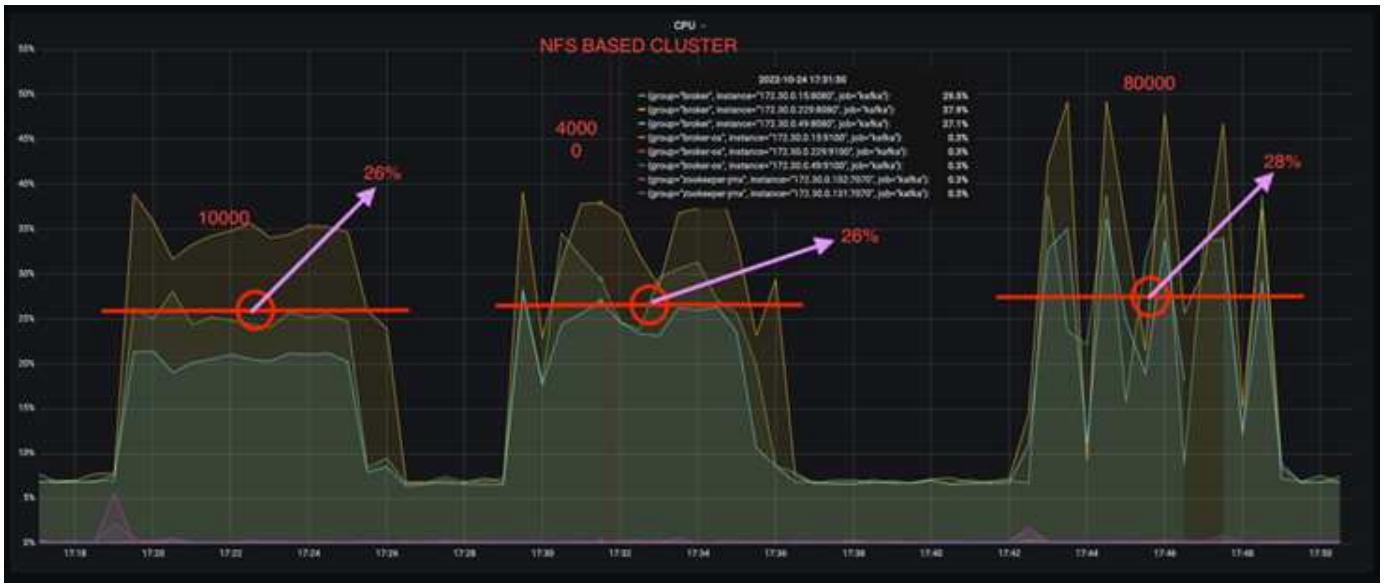
3. Die Produktionsratenkonfiguration wurde in vier Iterationen erhöht und die CPU-Auslastung mit Grafana aufgezeichnet. Die Produktionsrate wurde auf folgende Stufen festgelegt:
   - 10.000
   - 40.000
   - 80.000
   - 100.000

Beobachtung

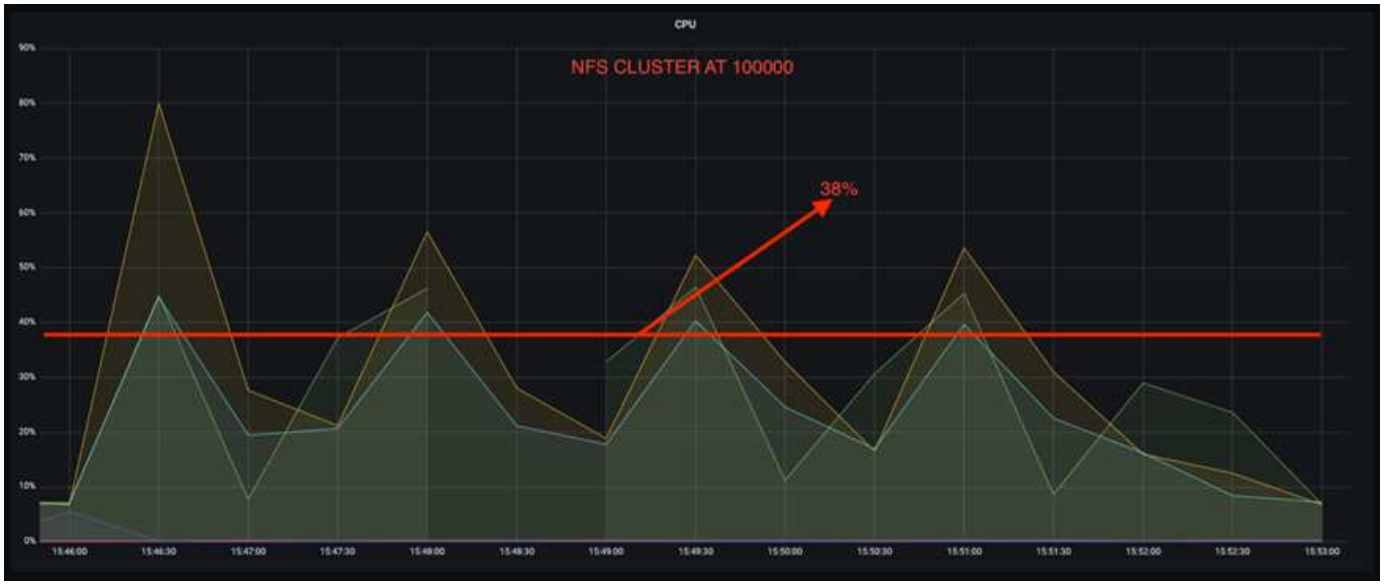
Die Verwendung von NetApp NFS-Speicher mit Kafka bietet zwei Hauptvorteile:

- **Sie können die CPU-Auslastung um fast ein Drittel reduzieren.** Die allgemeine CPU-Auslastung war bei ähnlichen Arbeitslasten bei NFS niedriger als bei DAS-SSDs; die Einsparungen reichen von 5 % bei niedrigeren Produktionsraten bis zu 32 % bei höheren Produktionsraten.
- **Eine dreifache Reduzierung der CPU-Auslastungsabweichung bei höheren Produktionsraten.** Wie erwartet gab es mit der Erhöhung der Produktionsraten einen Aufwärtstrend bei der Erhöhung der CPU-Auslastung. Allerdings stieg die CPU-Auslastung bei Kafka-Brokern, die DAS verwenden, von 31 % bei der niedrigeren Produktionsrate auf 70 % bei der höheren Produktionsrate, also um 39 %. Mit einem NFS-Speicher-Backend stieg die CPU-Auslastung jedoch von 26 % auf 38 %, eine Steigerung um 12 %.





Außerdem weist DAS bei 100.000 Nachrichten eine höhere CPU-Auslastung auf als ein NFS-Cluster.



Schnellere Broker-Wiederherstellung

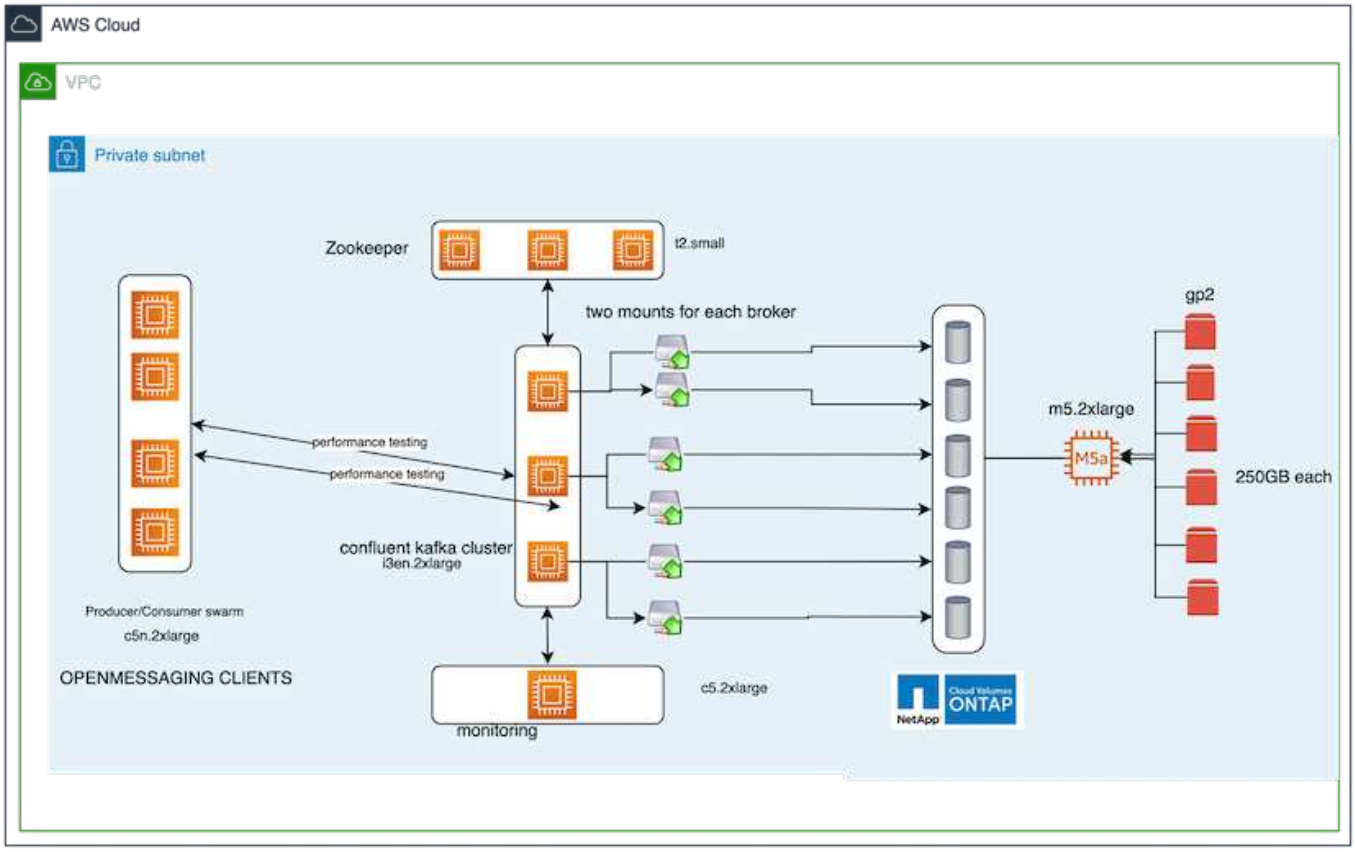
Wir haben festgestellt, dass Kafka-Broker schneller wiederhergestellt werden, wenn sie gemeinsam genutzten NetApp NFS-Speicher verwenden. Wenn ein Broker in einem Kafka-Cluster abstürzt, kann dieser Broker durch einen fehlerfreien Broker mit derselben Broker-ID ersetzt werden. Bei der Durchführung dieses Testfalls stellten wir fest, dass im Fall eines DAS-basierten Kafka-Clusters der Cluster die Daten auf einem neu hinzugefügten, fehlerfreien Broker neu aufbaut, was zeitaufwändig ist. Im Fall eines NetApp NFS-basierten Kafka-Clusters liest der ersetzende Broker weiterhin Daten aus dem vorherigen Protokollverzeichnis und stellt die Daten viel schneller wieder her.

Architektonischer Aufbau

Die folgende Tabelle zeigt die Umgebungskonfiguration für einen Kafka-Cluster mit NAS.

Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3	• 3 x Tierpfleger – t2.small • 3 x Broker-Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Produzent/Verbraucher – c5n.2xlarge • 1 x Backup-Kafka-Knoten – i3en.2xlarge
Betriebssystem auf allen Knoten	RHEL8.7 oder höher
NetApp Cloud Volumes ONTAP Instanz	Einzelknoteninstanz – M5.2xLarge

Die folgende Abbildung zeigt die Architektur eines NAS-basierten Kafka-Clusters.



- **Berechnen.** Ein Kafka-Cluster mit drei Knoten und einem Zookeeper-Ensemble mit drei Knoten, das auf dedizierten Servern ausgeführt wird. Jeder Broker verfügt über zwei NFS-Mount-Punkte zu einem einzelnen Volume auf der NetApp CVO-Instanz über ein dediziertes LIF.
- **Überwachung.** Zwei Knoten für eine Prometheus-Grafana-Kombination. Zum Generieren von Workloads verwenden wir einen separaten Cluster mit drei Knoten, der für diesen Kafka-Cluster produzieren und konsumieren kann.
- **Lagerung.** Eine NetApp Cloud Volumes ONTAP Instanz mit einem Knoten und sechs auf der Instanz gemounteten 250 GB GP2 AWS-EBS-Volumes. Diese Volumes werden dann dem Kafka-Cluster über dedizierte LIFs als sechs NFS-Volumes zur Verfügung gestellt.
- **Broker-Konfiguration.** Das einzige konfigurierbare Element in diesem Testfall sind Kafka-Broker. Für die Kafka-Broker wurden folgende Spezifikationen gewählt. Der `replica.lag.time.mx.ms` wird auf einen hohen Wert eingestellt, da dieser bestimmt, wie schnell ein bestimmter Knoten aus der ISR-Liste entfernt wird. Wenn Sie zwischen fehlerhaften und fehlerfreien Knoten wechseln, möchten Sie nicht, dass diese Broker-ID von der ISR-Liste ausgeschlossen wird.

```
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
```

Testmethodik

1. Es wurden zwei ähnliche Cluster erstellt:
   - Ein EC2-basierter konfluenter Cluster.
   - Ein NetApp NFS-basierter Confluent-Cluster.
2. Es wurde ein Standby-Kafka-Knoten mit einer Konfiguration erstellt, die mit den Knoten des ursprünglichen Kafka-Clusters identisch ist.
3. Auf jedem der Cluster wurde ein Beispielthema erstellt und auf jedem der Broker wurden ungefähr 110 GB Daten gespeichert.
   - **EC2-basierter Cluster.** Ein Kafka-Broker-Datenverzeichnis ist abgebildet auf `/mnt/data-2` (In der folgenden Abbildung Broker-1 von Cluster1 [linkes Terminal]).
   - * NetApp NFS-basierter Cluster.* Ein Kafka-Broker-Datenverzeichnis ist auf einem NFS-Punkt gemountet `/mnt/data` (In der folgenden Abbildung Broker-1 von Cluster2 [rechtes Terminal]).

4. In jedem der Cluster wurde Broker-1 beendet, um einen fehlgeschlagenen Broker-Wiederherstellungsprozess auszulösen.
5. Nachdem der Broker beendet wurde, wurde die Broker-IP-Adresse dem Standby-Broker als sekundäre IP zugewiesen. Dies war notwendig, da ein Broker in einem Kafka-Cluster durch Folgendes identifiziert wird:
   - **IP-Adresse.** Zugewiesen durch Neuzuweisung der ausgefallenen Broker-IP an den Standby-Broker.
   - **Broker-ID.** Dies wurde im Standby-Broker konfiguriert `server.properties` .
6. Bei der IP-Zuweisung wurde der Kafka-Dienst auf dem Standby-Broker gestartet.
7. Nach einer Weile wurden die Serverprotokolle abgerufen, um die zum Erstellen der Daten auf dem Ersatzknoten im Cluster benötigte Zeit zu überprüfen.

Beobachtung

Die Wiederherstellung des Kafka-Brokers war fast neunmal schneller. Die zur Wiederherstellung eines ausgefallenen Broker-Knotens benötigte Zeit war bei Verwendung des gemeinsam genutzten NetApp NFS-Speichers deutlich kürzer als bei Verwendung von DAS-SSDs in einem Kafka-Cluster. Bei 1 TB Themendaten betrug die Wiederherstellungszeit für einen DAS-basierten Cluster 48 Minuten, verglichen mit weniger als 5 Minuten für einen NetApp-NFS-basierten Kafka-Cluster.

Wir haben festgestellt, dass der EC2-basierte Cluster 10 Minuten benötigte, um die 110 GB Daten auf dem neuen Broker-Knoten wiederherzustellen, während der NFS-basierte Cluster die Wiederherstellung in 3 Minuten abschloss. Wir haben in den Protokollen auch festgestellt, dass die Consumer-Offsets für die Partitionen für EC2 0 waren, während im NFS-Cluster die Consumer-Offsets vom vorherigen Broker übernommen wurden.

```
[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-basierter Cluster

1. Der Sicherungsknoten wurde um 08:55:53.730 gestartet.

```
[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. Der Datenwiederherstellungsprozess endete um 09:05:24.860. Die Verarbeitung von 110 GB Daten dauerte ungefähr 10 Minuten.

```
[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-basierter Cluster

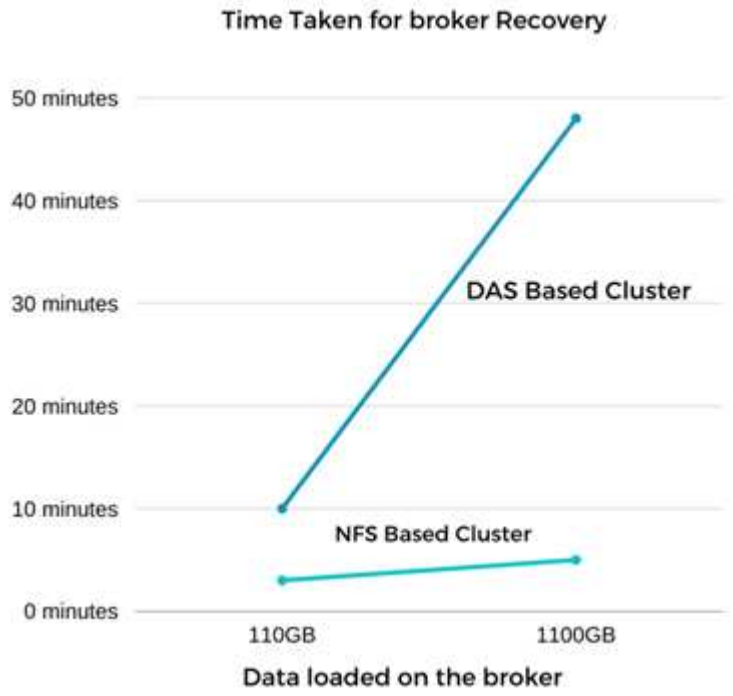
1. Der Backup-Knoten wurde um 09:39:17,213 gestartet. Der Startprotokolleintrag ist unten hervorgehoben.

```
[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. Der Datenwiederherstellungsprozess endete um 09:42:29,115. Die Verarbeitung von 110 GB Daten dauerte ungefähr 3 Minuten.

```
[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)
```

Der Test wurde für Broker mit etwa 1 TB Daten wiederholt, was für das DAS ungefähr 48 Minuten und für NFS 3 Minuten dauerte. Die Ergebnisse sind in der folgenden Grafik dargestellt.



Speichereffizienz

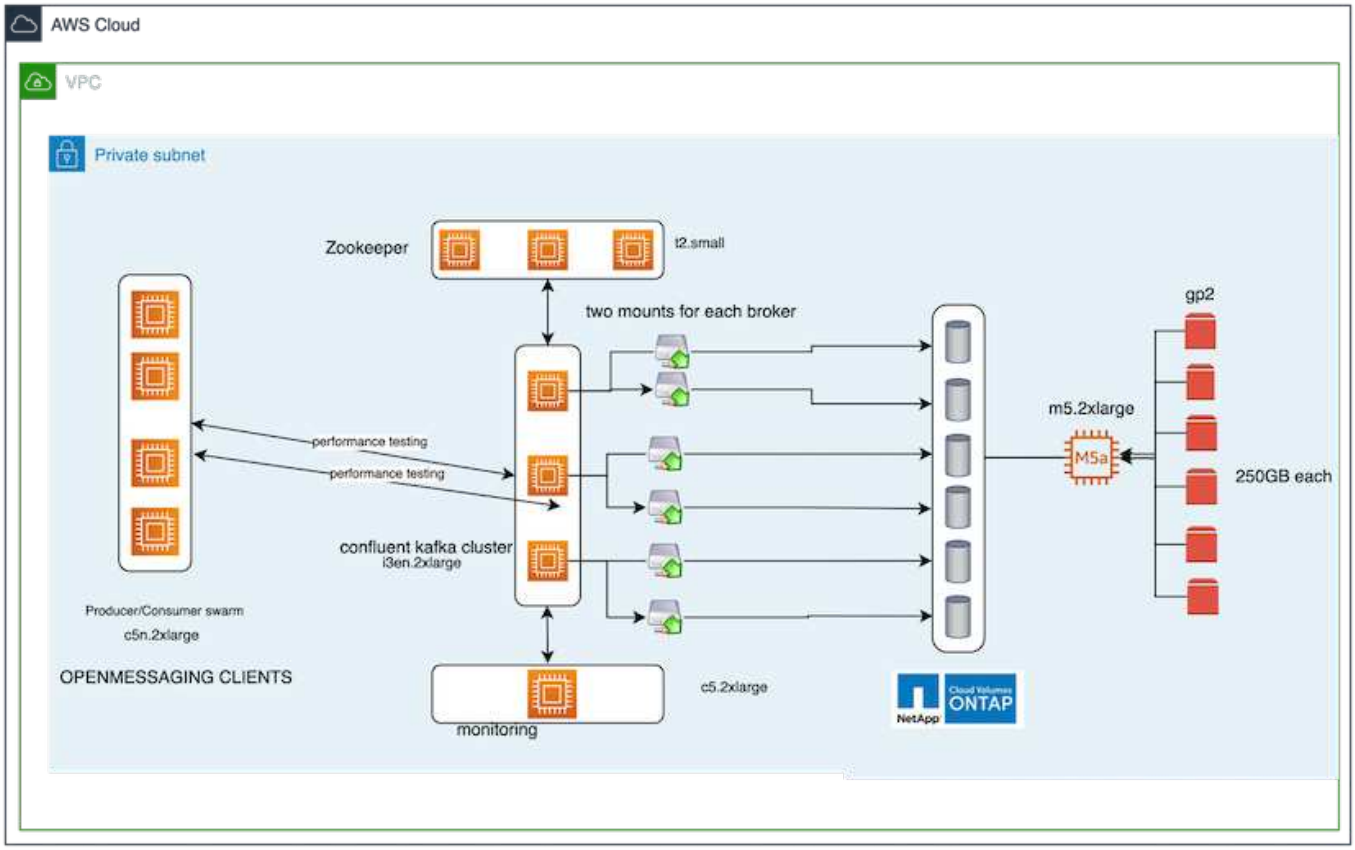
Da die Speicherschicht des Kafka-Clusters über NetApp ONTAP bereitgestellt wurde, konnten wir alle Speichereffizienzfunktionen von ONTAP nutzen. Dies wurde getestet, indem eine erhebliche Datenmenge auf einem Kafka-Cluster mit NFS-Speicher generiert wurde, der auf Cloud Volumes ONTAP bereitgestellt wurde. Wir konnten feststellen, dass es aufgrund der ONTAP -Funktionen zu einer erheblichen Platzreduzierung kam.

Architektonischer Aufbau

Die folgende Tabelle zeigt die Umgebungskonfiguration für einen Kafka-Cluster mit NAS.

Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3	• 3 x Tierpfleger – t2.small • 3 x Broker-Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Produzent/Verbraucher — c5n.2xlarge *
Betriebssystem auf allen Knoten	RHEL8.7 oder höher
NetApp Cloud Volumes ONTAP Instanz	Einzelknoteninstanz – M5.2xLarge

Die folgende Abbildung zeigt die Architektur eines NAS-basierten Kafka-Clusters.



- **Berechnen.** Wir haben einen Kafka-Cluster mit drei Knoten und einem Zookeeper-Ensemble mit drei Knoten verwendet, das auf dedizierten Servern ausgeführt wird. Jeder Broker verfügte über zwei NFS-Mount-Punkte zu einem einzelnen Volume auf der NetApp CVO-Instanz über ein dediziertes LIF.
- **Überwachung.** Wir haben zwei Knoten für eine Prometheus-Grafana-Kombination verwendet. Zum Generieren von Workloads haben wir einen separaten Cluster mit drei Knoten verwendet, der für diesen Kafka-Cluster produzieren und konsumieren konnte.
- **Lagerung.** Wir haben eine NetApp Cloud Volumes ONTAP Instanz mit einem Knoten und sechs auf der Instanz gemounteten 250 GB GP2 AWS-EBS-Volumes verwendet. Diese Volumes wurden dann über dedizierte LIFs als sechs NFS-Volumes dem Kafka-Cluster zugänglich gemacht.
- **Konfiguration.** Die konfigurierbaren Elemente in diesem Testfall waren die Kafka-Broker.

Die Komprimierung wurde auf der Produzentenseite abgeschaltet, wodurch die Produzenten einen hohen

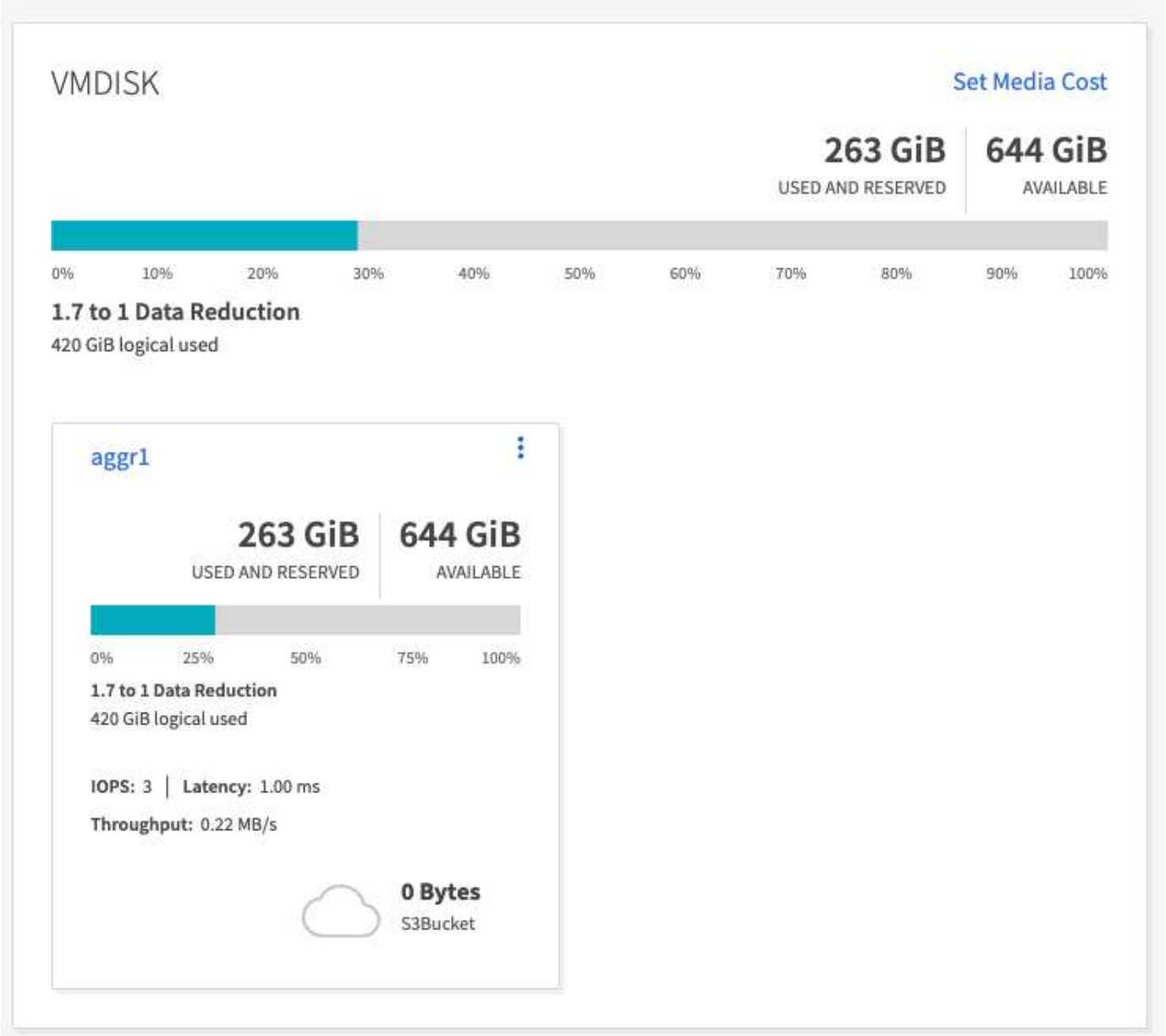
Durchsatz erzielen konnten. Die Speichereffizienz wurde stattdessen von der Rechenschicht übernommen.

Testmethodik

1. Ein Kafka-Cluster wurde mit den oben genannten Spezifikationen bereitgestellt.
2. Auf dem Cluster wurden mithilfe des OpenMessaging Benchmarking-Tools etwa 350 GB Daten erstellt.
3. Nachdem die Arbeitslast abgeschlossen war, wurden die Statistiken zur Speichereffizienz mithilfe von ONTAP System Manager und der CLI erfasst.

Beobachtung

Bei Daten, die mit dem OMB-Tool generiert wurden, konnten wir eine Platzersparnis von ca. 33 % bei einem Speichereffizienzverhältnis von 1,70:1 feststellen. Wie aus den folgenden Abbildungen hervorgeht, betrug der von den erzeugten Daten verwendete logische Speicherplatz 420,3 GB und der zum Speichern der Daten verwendete physische Speicherplatz 281,7 GB.



```
shantanuCVOinstancenew::> df -h -S

Warning: The "-S" parameter is deprecated and may be removed in a future release. To show the efficiency ratio use "aggr show-efficiency"
         command.
Filesystem               used      total-saved    %total-saved    deduplicated    %deduplicated    compressed    %compressed  Vserver
/vol/vol0/             7319MB               0B              0%              0B              0%            0B             0%  shantanuCVOinstancenew-01
/vol/kafka_vol/         281GB            138GB             33%           138GB             33%            0B             0%  svm_shantanuCVOinstancenew
/vol/svm_shantanuCVOinstancenew_root/
                        660KB               0B              0%              0B              0%            0B             0%  svm_shantanuCVOinstancenew
3 entries were displayed.
```

```
                    Name of the Aggregate: aggr1
               Node where Aggregate Resides: shantanuCVOinstancenew-01
             Total Storage Efficiency Ratio: 1.70:1
Total Data Reduction Efficiency Ratio Without Snapshots: 1.70:1
Total Data Reduction Efficiency Ratio without snapshots and flexclones: 1.70:1
          Logical Space Used for All Volumes: 420.3GB
        Physical Space Used for All Volumes: 281.7GB
```

Leistungsübersicht und -validierung in AWS

Ein Kafka-Cluster mit der auf NetApp NFS montierten Speicherschicht wurde hinsichtlich seiner Leistung in der AWS-Cloud getestet. Die Benchmarking-Beispiele werden in den folgenden Abschnitten beschrieben.

Kafka in der AWS-Cloud mit NetApp Cloud Volumes ONTAP (Hochverfügbarkeitspaar und Einzelknoten)

Ein Kafka-Cluster mit NetApp Cloud Volumes ONTAP (HA-Paar) wurde hinsichtlich seiner Leistung in der AWS-Cloud getestet. Dieses Benchmarking wird in den folgenden Abschnitten beschrieben.

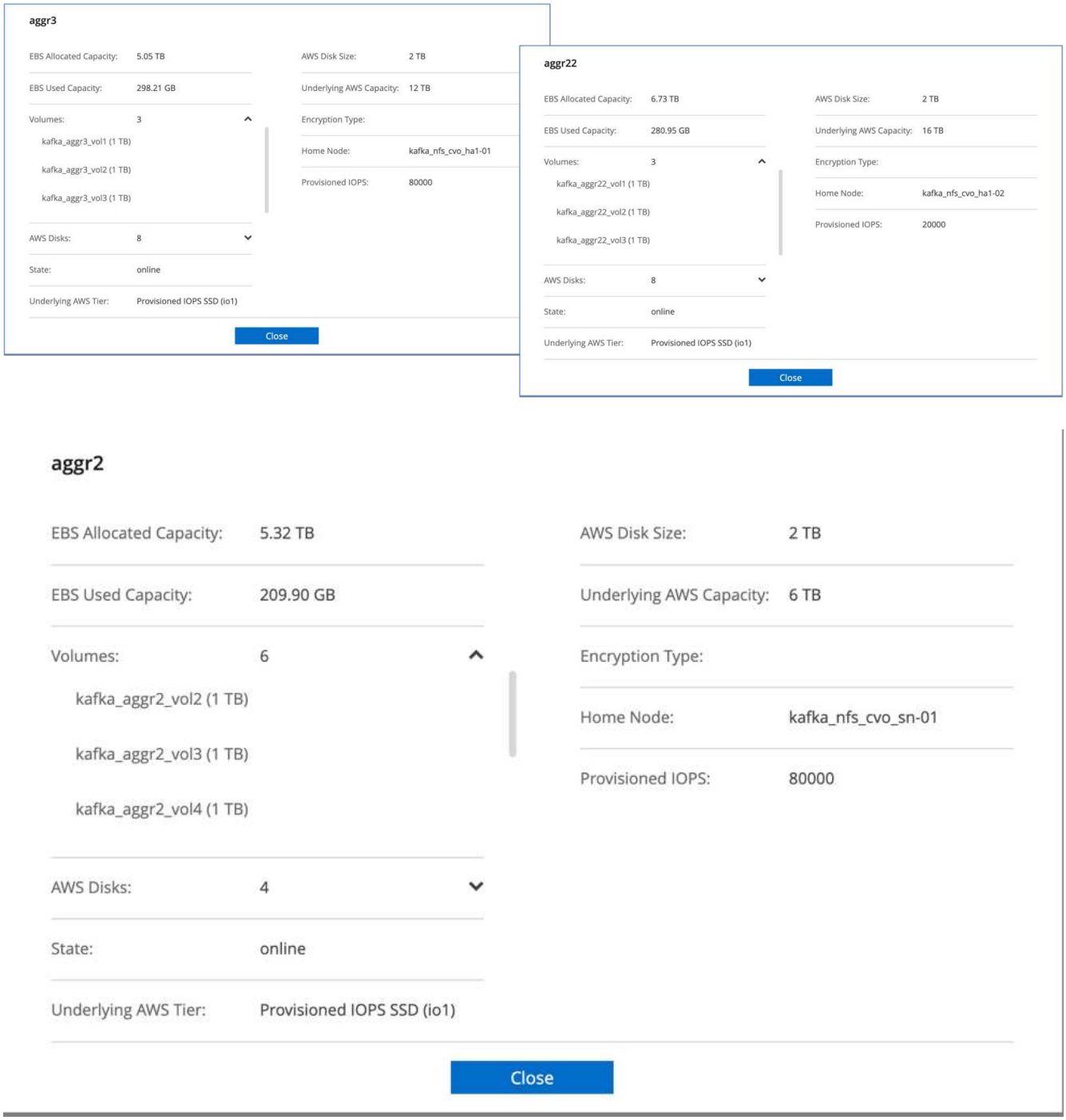
Architektonischer Aufbau

Die folgende Tabelle zeigt die Umgebungskonfiguration für einen Kafka-Cluster mit NAS.

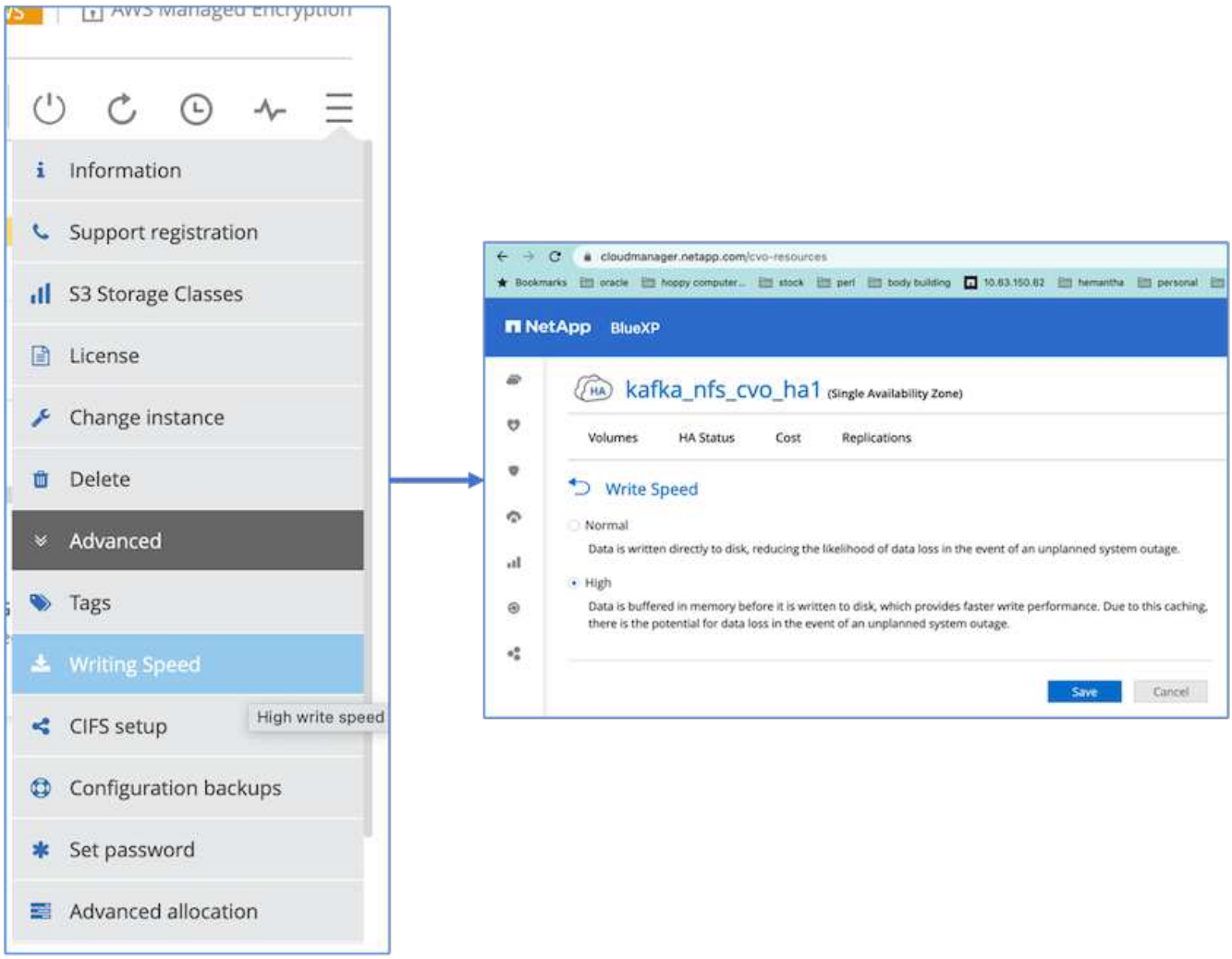
Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3	• 3 x Tierpfleger – t2.small • 3 x Broker-Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Produzent/Verbraucher — c5n.2xlarge *
Betriebssystem auf allen Knoten	RHEL8.6
NetApp Cloud Volumes ONTAP Instanz	HA-Paarinstanz – m5dn.12xLarge x 2 Knoten Einzelknoteninstanz – m5dn.12xLarge x 1 Knoten

NetApp Cluster Volume ONTAP Setup

1. Für das Cloud Volumes ONTAP HA-Paar haben wir zwei Aggregate mit jeweils drei Volumes auf jedem Aggregat auf jedem Speichercontroller erstellt. Für den einzelnen Cloud Volumes ONTAP -Knoten erstellen wir sechs Volumes in einem Aggregat.

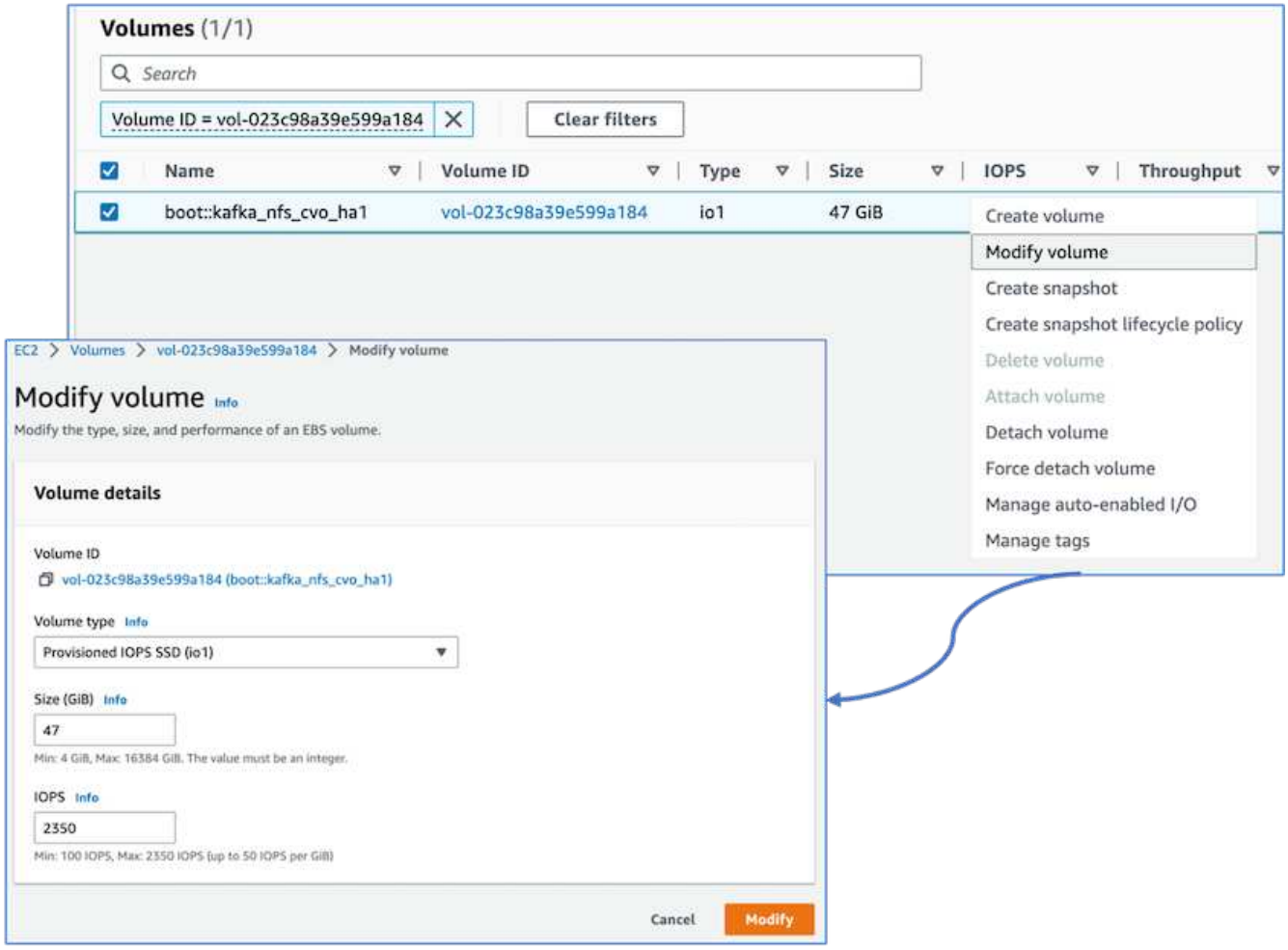


2. Um eine bessere Netzwerkleistung zu erzielen, haben wir Hochgeschwindigkeitsnetzwerke sowohl für das HA-Paar als auch für den einzelnen Knoten aktiviert.



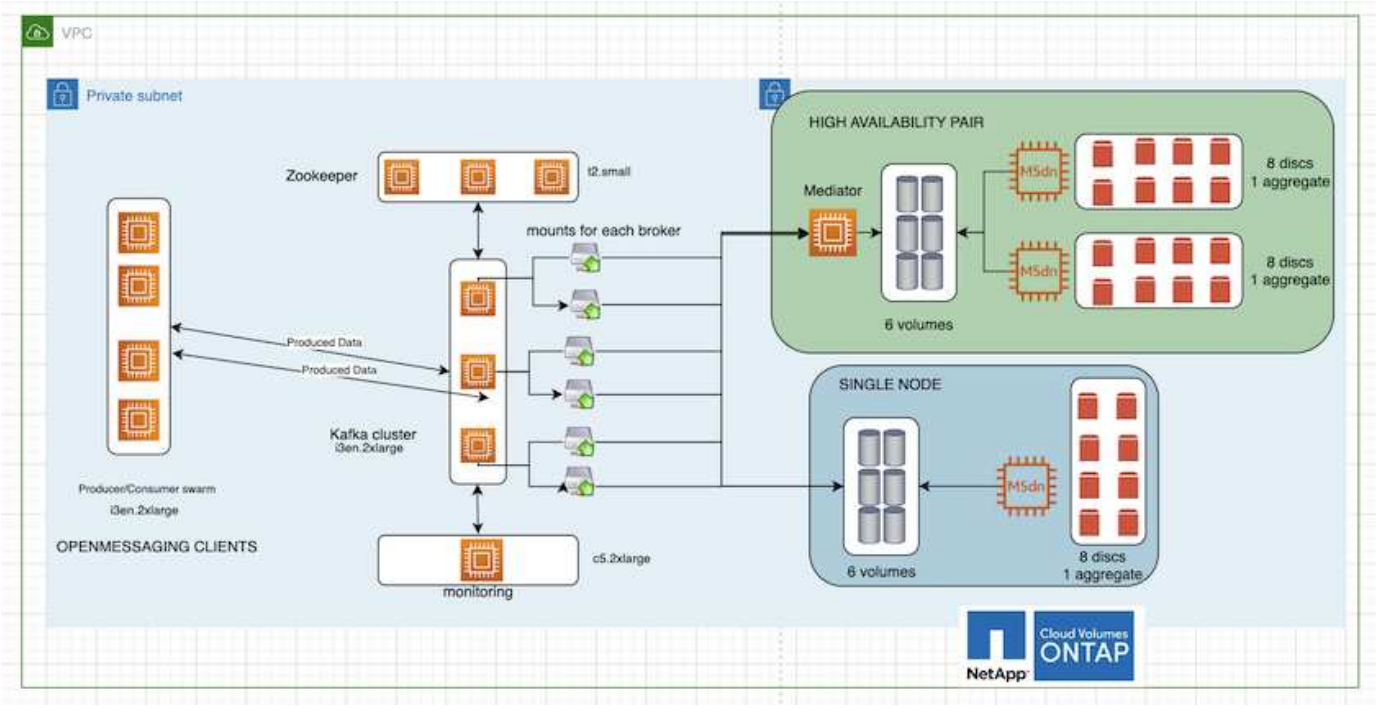
3. Wir haben festgestellt, dass der ONTAP NVRAM mehr IOPS hatte, also haben wir die IOPS für das Cloud Volumes ONTAP Stammvolume auf 2350 geändert. Die Root-Volume-Festplatte in Cloud Volumes ONTAP hatte eine Größe von 47 GB. Der folgende ONTAP -Befehl gilt für das HA-Paar und der gleiche Schritt ist für den einzelnen Knoten anwendbar.

```
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::*>
```



Die folgende Abbildung zeigt die Architektur eines NAS-basierten Kafka-Clusters.

- **Berechnen.** Wir haben einen Kafka-Cluster mit drei Knoten und einem Zookeeper-Ensemble mit drei Knoten verwendet, das auf dedizierten Servern ausgeführt wird. Jeder Broker verfügte über zwei NFS-Mount-Punkte zu einem einzelnen Volume auf der Cloud Volumes ONTAP Instanz über ein dediziertes LIF.
- **Überwachung.** Wir haben zwei Knoten für eine Prometheus-Grafana-Kombination verwendet. Zum Generieren von Workloads haben wir einen separaten Cluster mit drei Knoten verwendet, der für diesen Kafka-Cluster produzieren und konsumieren konnte.
- **Lagerung.** Wir haben eine HA-Pair-Cloud-Volumes ONTAP Instanz mit einem auf der Instanz gemounteten 6-TB-GP3-AWS-EBS-Volume verwendet. Anschließend wurde das Volume mit einem NFS-Mount zum Kafka-Broker exportiert.



OpenMessage Benchmarking-Konfigurationen

1. Für eine bessere NFS-Leistung benötigen wir mehr Netzwerkverbindungen zwischen dem NFS-Server und dem NFS-Client, die mit nconnect erstellt werden können. Mounten Sie die NFS-Volumes auf den Broker-Knoten mit der Option „nconnect“, indem Sie den folgenden Befehl ausführen:

```
[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. Überprüfen Sie die Netzwerkverbindungen in Cloud Volumes ONTAP. Der folgende ONTAP -Befehl wird vom einzelnen Cloud Volumes ONTAP Knoten verwendet. Derselbe Schritt gilt für das Cloud Volumes ONTAP HA-Paar.

```
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. Wir verwenden folgende Kafka `server.properties` in allen Kafka-Brokern für das Cloud Volumes ONTAP HA-Paar. Der `log.dirs` Die Eigenschaft ist für jeden Broker unterschiedlich und die übrigen Eigenschaften sind für alle Broker gleich. Für Broker1 ist die `log.dirs` Der Wert lautet wie folgt:

```
[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 ~]#
```

◦ Für Broker2 ist die `log.dirs` Der Eigenschaftswert lautet wie folgt:

```
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
```

◦ Für Broker3 ist die `log.dirs` Der Eigenschaftswert lautet wie folgt:

```
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. Für den einzelnen Cloud Volumes ONTAP -Knoten: Der Kafka `servers.properties` ist das gleiche wie für das Cloud Volumes ONTAP HA-Paar, außer der `log.dirs` Eigentum.
   - Für Broker1 ist die `log.dirs` Der Wert lautet wie folgt:

```
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
```

   - Für Broker2 ist die `log.dirs` Der Wert lautet wie folgt:

```
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
```

   - Für Broker3 ist die `log.dirs` Der Eigenschaftswert lautet wie folgt:

```
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. Die Arbeitslast im OMB ist mit den folgenden Eigenschaften konfiguriert: `(/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
```

Der `messageSize` kann je nach Anwendungsfall unterschiedlich sein. In unserem Leistungstest haben wir

3K verwendet.

Wir haben zwei verschiedene Treiber, Sync oder Throughput, von OMB verwendet, um die Arbeitslast auf dem Kafka-Cluster zu generieren.

- Die für die Sync-Treibereigenschaften verwendete YAML-Datei lautet wie folgt (`/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
```

- Die für die Durchsatztreibereigenschaften verwendete YAML-Datei lautet wie folgt (`/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
```

Testmethodik

1. Ein Kafka-Cluster wurde gemäß der oben beschriebenen Spezifikation mit Terraform und Ansible bereitgestellt. Terraform wird verwendet, um die Infrastruktur mithilfe von AWS-Instanzen für den Kafka-Cluster aufzubauen, und Ansible baut den Kafka-Cluster darauf auf.
2. Mit der oben beschriebenen Workload-Konfiguration und dem Sync-Treiber wurde eine OMB-Workload ausgelöst.

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

3. Mit dem Throughput-Treiber wurde eine weitere Workload mit derselben Workload-Konfiguration ausgelöst.

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

Beobachtung

Zum Generieren von Workloads wurden zwei verschiedene Treibertypen verwendet, um die Leistung einer auf NFS laufenden Kafka-Instanz zu vergleichen. Der Unterschied zwischen den Treibern liegt in der Log-Flush-Eigenschaft.

Für ein Cloud Volumes ONTAP HA-Paar:

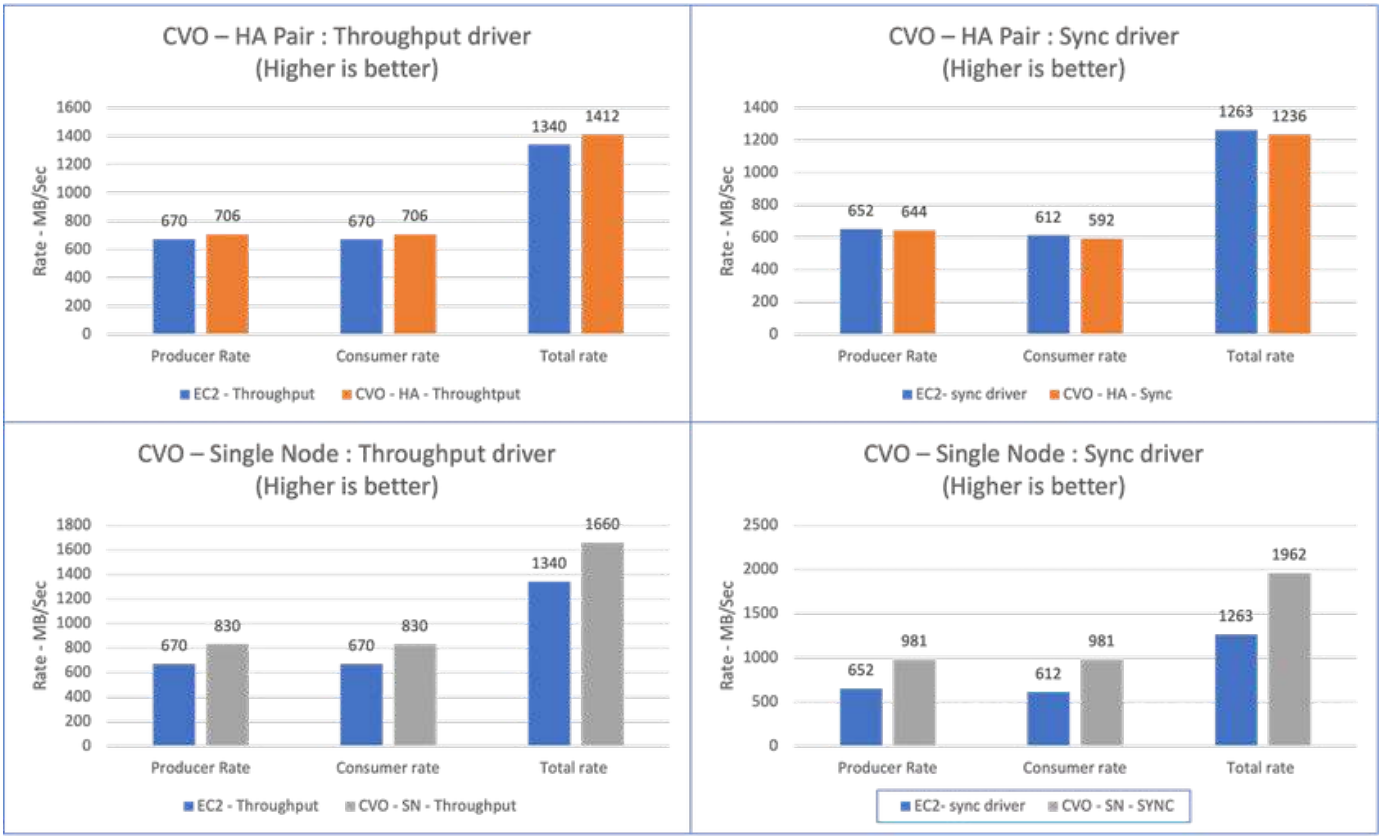
- Gesamtdurchsatz, der durchgängig vom Sync-Treiber generiert wird: ~1236 MBps.
- Gesamtdurchsatz, der für den Durchsatztreiber generiert wurde: Spitze ~1412 MBps.

Für einen einzelnen Cloud Volumes ONTAP Knoten:

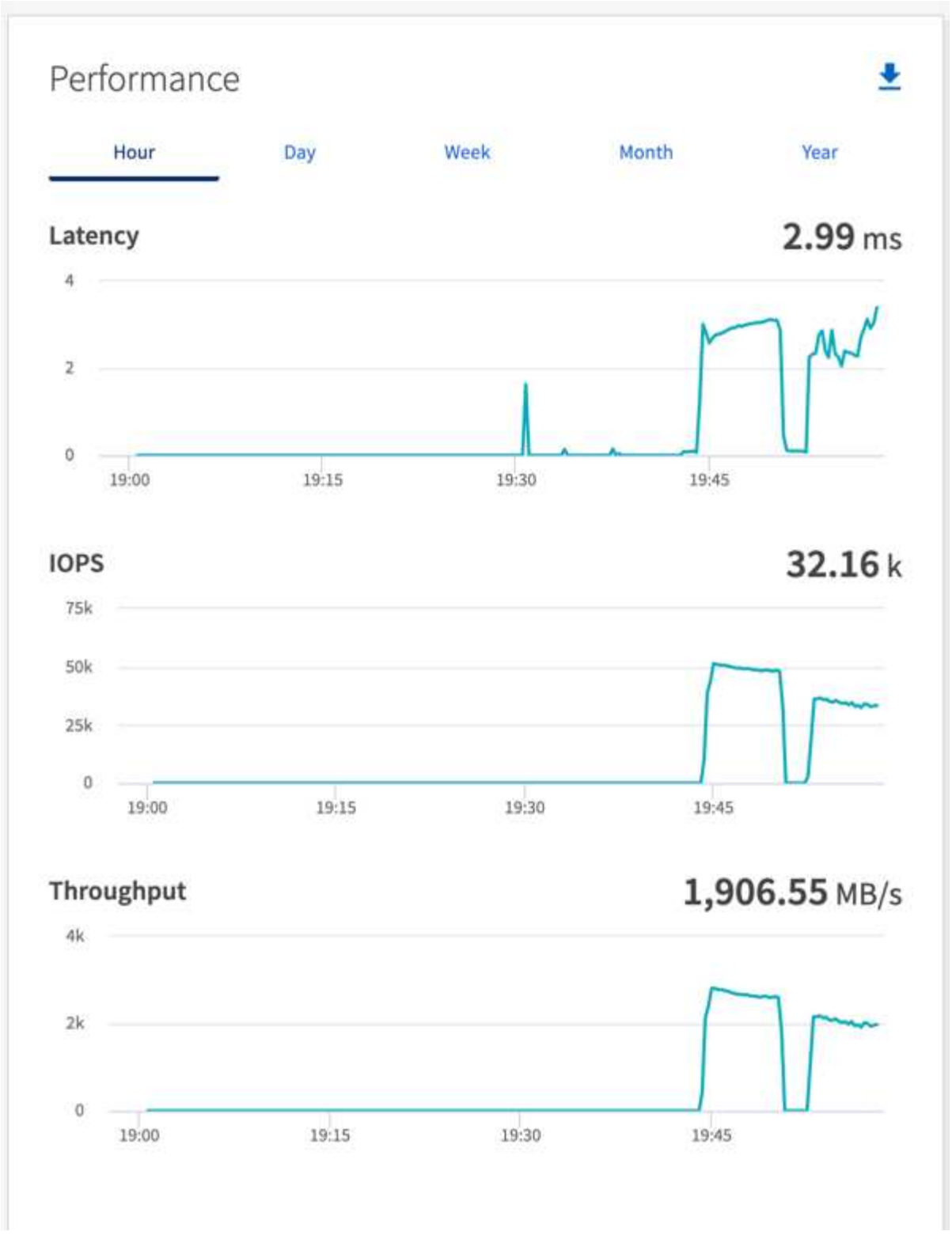
- Gesamtdurchsatz, der durchgängig vom Sync-Treiber generiert wird: ~ 1962 MBps.
- Gesamtdurchsatz, der vom Durchsatztreiber generiert wird: Spitze ~1660 MBps

Der Sync-Treiber kann einen konsistenten Durchsatz generieren, da Protokolle sofort auf die Festplatte geschrieben werden, während der Throughput-Treiber Durchsatzschübe generiert, da Protokolle in großen Mengen auf die Festplatte geschrieben werden.

Diese Durchsatzzahlen werden für die jeweilige AWS-Konfiguration generiert. Bei höheren Leistungsanforderungen können die Instanztypen hochskaliert und für bessere Durchsatzzahlen weiter optimiert werden. Der Gesamtdurchsatz oder die Gesamtrate ist die Kombination aus Produzenten- und Verbraucherrate.



Überprüfen Sie unbedingt den Speicherdurchsatz, wenn Sie ein Durchsatz- oder Synchronisierungstreiber-Benchmarking durchführen.



Leistungsübersicht und -validierung in AWS FSx ONTAP

Ein Kafka-Cluster mit der auf NetApp NFS montierten Speicherschicht wurde hinsichtlich seiner Leistung im AWS FSx ONTAP getestet. Die Benchmarking-Beispiele werden in den folgenden Abschnitten beschrieben.

Apache Kafka in AWS FSx ONTAP

Network File System (NFS) ist ein weit verbreitetes Netzwerkdateisystem zum Speichern großer Datenmengen. In den meisten Organisationen werden Daten zunehmend durch Streaming-Anwendungen wie Apache Kafka generiert. Diese Workloads erfordern Skalierbarkeit, geringe Latenz und eine robuste Datenaufnahmearchitektur mit modernen Speicherfunktionen. Um Echtzeitanalysen zu ermöglichen und umsetzbare Erkenntnisse zu liefern, ist eine gut konzipierte und hochleistungsfähige Infrastruktur erforderlich.

Kafka arbeitet konstruktionsbedingt mit POSIX-kompatiblen Dateisystemen und verlässt sich bei der Verarbeitung von Dateivorgängen auf das Dateisystem. Beim Speichern von Daten auf einem NFSv3-Dateisystem kann der NFS-Client des Kafka-Brokers Dateivorgänge jedoch anders interpretieren als ein lokales Dateisystem wie XFS oder Ext4. Ein häufiges Beispiel ist die NFS Silly-Umbenennung, die zum Ausfall von Kafka-Brokern beim Erweitern von Clustern und Neuzuordnen von Partitionen führte. Um diese Herausforderung zu bewältigen, hat NetApp den Open-Source-Linux-NFS-Client mit Änderungen aktualisiert, die jetzt allgemein in RHEL8.7 und RHEL9.1 verfügbar sind und ab der aktuellen FSx ONTAP Version ONTAP 9.12.1 unterstützt werden.

Amazon FSx ONTAP bietet ein vollständig verwaltetes, skalierbares und leistungsstarkes NFS-Dateisystem in der Cloud. Kafka-Daten auf FSx ONTAP können skaliert werden, um große Datenmengen zu verarbeiten und Fehlertoleranz zu gewährleisten. NFS bietet zentrales Speichermanagement und Datenschutz für kritische und sensible Datensätze.

Diese Verbesserungen ermöglichen es AWS-Kunden, die Vorteile von FSx ONTAP zu nutzen, wenn sie Kafka-Workloads auf AWS-Rechendiensten ausführen. Diese Vorteile sind: * Reduzierung der CPU-Auslastung zur Verkürzung der E/A-Wartezeit * Schnellere Wiederherstellungszeit des Kafka-Brokers. * Zuverlässigkeit und Effizienz. * Skalierbarkeit und Leistung. * Verfügbarkeit in mehreren Verfügbarkeitszonen. * Datenschutz.

Leistungsübersicht und -validierung in AWS FSx ONTAP

Ein Kafka-Cluster mit der auf NetApp NFS montierten Speicherschicht wurde hinsichtlich seiner Leistung in der AWS-Cloud getestet. Die Benchmarking-Beispiele werden in den folgenden Abschnitten beschrieben.

Kafka in AWS FSx ONTAP

Ein Kafka-Cluster mit AWS FSx ONTAP wurde hinsichtlich seiner Leistung in der AWS-Cloud getestet. Dieses Benchmarking wird in den folgenden Abschnitten beschrieben.

Architektonischer Aufbau

Die folgende Tabelle zeigt die Umgebungskonfiguration für einen Kafka-Cluster mit AWS FSx ONTAP.

Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3	• 3 x Tierpfleger – t2.small • 3 x Broker-Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Produzent/Verbraucher — c5n.2xlarge *
Betriebssystem auf allen Knoten	RHEL8.6
AWS FSx ONTAP	Multi-AZ mit 4 GB/Sek. Durchsatz und 160.000 IOPS

NetApp FSx ONTAP Setup

1. Für unsere ersten Tests haben wir ein FSx ONTAP Dateisystem mit 2 TB Kapazität und 40.000 IOPs für einen Durchsatz von 2 GB/s erstellt.

```
[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"}
```

   In unserem Beispiel stellen wir FSx ONTAP über die AWS CLI bereit. Sie müssen den Befehl in Ihrer Umgebung nach Bedarf weiter anpassen. FSx ONTAP kann zusätzlich über die AWS-Konsole bereitgestellt und verwaltet werden, um eine einfachere und optimierte Bereitstellung mit weniger Befehlszeileneingaben zu ermöglichen.

   Dokumentation: In FSx ONTAP beträgt der maximal erreichbare IOPS-Wert für ein Dateisystem mit 2 GB/s Durchsatz in unserer Testregion (US-Ost-1) 80.000 IOPS. Die maximalen Gesamt-IOPS für ein FSx ONTAP -Dateisystem betragen 160.000 IOPS. Um dies zu erreichen, ist eine Bereitstellung mit einem Durchsatz von 4 GB/s erforderlich, was wir später in diesem Dokument demonstrieren werden.

   Weitere Informationen zu den Leistungsspezifikationen von FSx ONTAP finden Sie hier in der AWS FSx ONTAP -Dokumentation: https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html .

   Eine detaillierte Befehlszeilensyntax für FSx „create-file-system“ finden Sie hier: https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

   Sie können beispielsweise einen bestimmten KMS-Schlüssel angeben, im Gegensatz zum standardmäßigen AWS FSx-Hauptschlüssel, der verwendet wird, wenn kein KMS-Schlüssel angegeben ist.

2. Warten Sie beim Erstellen des FSx ONTAP Dateisystems, bis sich der Status „LifeCycle“ in Ihrer JSON-Rückgabe in „AVAILABLE“ ändert, nachdem Sie Ihr Dateisystem wie folgt beschrieben haben:

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

3. Bestätigen Sie die Anmeldeinformationen, indem Sie sich mit dem Benutzer fsxadmin bei FSx ONTAP SSH anmelden: Fsxadmin ist das Standardadministratorkonto für FSx ONTAP Dateisysteme bei der Erstellung. Das Kennwort für fsxadmin ist das Kennwort, das beim ersten Erstellen des Dateisystems entweder in der AWS-Konsole oder mit der AWS CLI konfiguriert wurde, wie wir es in Schritt 1 abgeschlossen haben.

```
[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. Sobald Ihre Anmeldeinformationen validiert wurden, erstellen Sie die virtuelle Speichermaschine auf dem FSx ONTAP Dateisystem

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

Eine Storage Virtual Machine (SVM) ist ein isolierter Dateiserver mit eigenen Administratoranmeldeinformationen und Endpunkten zum Verwalten und Zugreifen auf Daten in FSx ONTAP Volumes und bietet FSx ONTAP Multi-Tenancy.

5. Nachdem Sie Ihre primäre Storage Virtual Machine konfiguriert haben, greifen Sie per SSH auf das neu erstellte FSx ONTAP Dateisystem zu und erstellen Sie Volumes in der Storage Virtual Machine mit dem folgenden Beispielbefehl. Auf ähnliche Weise erstellen wir 6 Volumes für diese Validierung. Behalten Sie basierend auf unserer Validierung die Standardkomponente (8) oder weniger Komponenten bei, was zu einer besseren Leistung von Kafka führt.

```
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. Für unsere Tests benötigen wir zusätzliche Kapazitäten in unseren Volumina. Erweitern Sie die Größe des Volumes auf 2 TB und mounten Sie es auf dem Verbindungspfad.

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN1 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN1" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN2 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN2" size set to 2.10t.
```

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN3 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN3" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN4 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN4" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN5 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN5" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN6 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN6" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume show -vserver svmkafkatest -volume *
Vserver   Volume       Aggregate    State      Type       Size
Available Used%
--------- ------------ ------------ ---------- ---- ----------
---------- -----
svmkafkatest
          kafkafsxN1   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN2   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN3   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN4   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN5   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN6   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          svmkafkatest_root
                       aggr1        online     RW          1GB
968.1MB    0%
7 entries were displayed.

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN1 -junction
-path /kafkafsxN1

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN2 -junction
-path /kafkafsxN2
```

```
FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN3 -junction
-path /kafkafsxN3

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN4 -junction
-path /kafkafsxN4

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN5 -junction
-path /kafkafsxN5

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN6 -junction
-path /kafkafsxN6
```

In FSx ONTAP können Volumes per Thin Provisioning bereitgestellt werden. In unserem Beispiel übersteigt die Gesamtkapazität des erweiterten Volumes die Gesamtkapazität des Dateisystems. Daher müssen wir die Gesamtkapazität des Dateisystems erweitern, um zusätzliche bereitgestellte Volumekapazität freizugeben, was wir im nächsten Schritt demonstrieren werden.

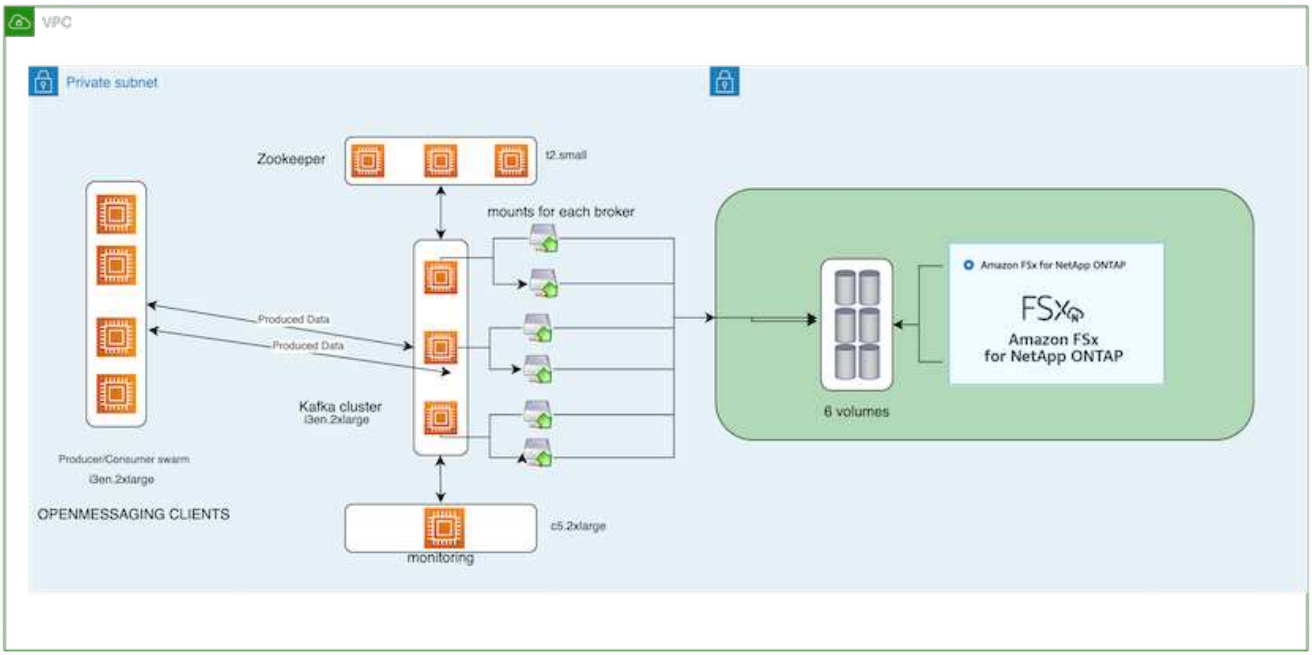
7. Als nächstes erweitern wir für zusätzliche Leistung und Kapazität die FSx ONTAP Durchsatzkapazität von 2 GB/Sek. auf 4 GB/Sek. und IOPS auf 160000 und die Kapazität auf 5 TB

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

Eine detaillierte Befehlszeilensyntax für FSx „update-file-system“ finden Sie hier:https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. Die FSx ONTAP -Volumes werden mit nconnect und Standardoptionen in Kafka-Brokern gemountet

Das folgende Bild zeigt unsere endgültige Architektur eines auf FSx ONTAP basierenden Kafka-Clusters:



- Berechnen. Wir haben einen Kafka-Cluster mit drei Knoten und einem Zookeeper-Ensemble mit drei Knoten verwendet, das auf dedizierten Servern ausgeführt wird. Jeder Broker hatte sechs NFS-Mount-Punkte für sechs Volumes auf der FSx ONTAP Instanz.
- Überwachung. Wir haben zwei Knoten für eine Prometheus-Grafana-Kombination verwendet. Zum Generieren von Workloads haben wir einen separaten Cluster mit drei Knoten verwendet, der für diesen Kafka-Cluster produzieren und verbrauchen konnte.
- Lagerung. Wir haben ein FSx ONTAP mit sechs gemounteten 2-TB-Volumes verwendet. Das Volume wurde dann mit einem NFS-Mount zum Kafka-Broker exportiert. Die FSx ONTAP Volumes werden mit 16 Nconnect-Sitzungen und Standardoptionen in Kafka-Brokern gemountet.

OpenMessage-Benchmarking-Konfigurationen.

Wir haben dieselbe Konfiguration verwendet, die für die NetApp Cloud Volumes ONTAP verwendet wurde. Die Details dazu finden Sie hier: Link:kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

Testmethodik

1. Ein Kafka-Cluster wurde gemäß der oben beschriebenen Spezifikation mithilfe von Terraform und Ansible bereitgestellt. Terraform wird verwendet, um die Infrastruktur mithilfe von AWS-Instanzen für den Kafka-Cluster aufzubauen, und Ansible erstellt den Kafka-Cluster darauf.
2. Mit der oben beschriebenen Workload-Konfiguration und dem Sync-Treiber wurde eine OMB-Workload ausgelöst.

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

3. Mit dem Throughput-Treiber wurde eine weitere Workload mit derselben Workload-Konfiguration ausgelöst.

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

Beobachtung

Zum Generieren von Workloads wurden zwei verschiedene Treibertypen verwendet, um die Leistung einer auf NFS laufenden Kafka-Instanz zu vergleichen. Der Unterschied zwischen den Treibern liegt in der Log-Flush-Eigenschaft.

Für einen Kafka-Replikationsfaktor 1 und den FSx ONTAP:

- Gesamtdurchsatz, der durchgängig vom Sync-Treiber generiert wird: ~ 3218 MBps und Spitzenleistung bei ~ 3652 MBps.
- Gesamtdurchsatz, der durchgängig vom Durchsatztreiber generiert wird: ~ 3679 MBps und Spitzenleistung bei ~ 3908 MBps.

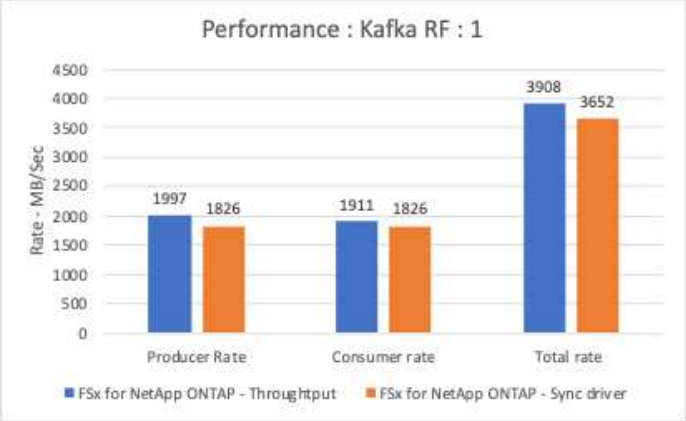
Für Kafka mit Replikationsfaktor 3 und FSx ONTAP :

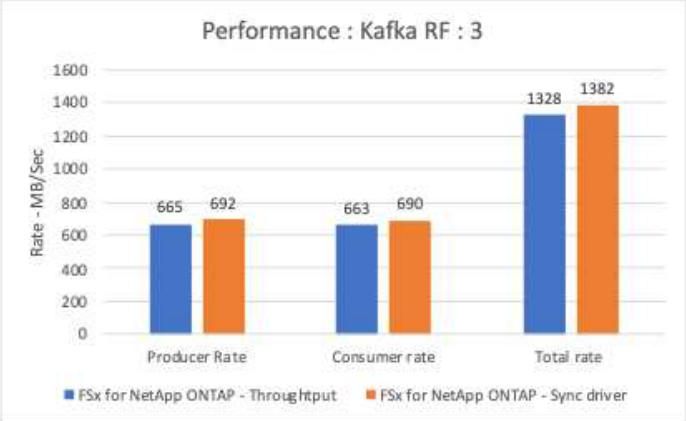
- Gesamtdurchsatz, der durchgängig vom Sync-Treiber generiert wird: ~ 1252 MBps und Spitzenleistung bei ~ 1382 MBps.
- Gesamtdurchsatz, der durch den Durchsatztreiber konstant generiert wird: ~ 1218 MBps und Spitzenleistung bei ~ 1328 MBps.

Beim Kafka-Replikationsfaktor 3 erfolgte der Lese- und Schreibvorgang dreimal auf dem FSx ONTAP. Beim Kafka-Replikationsfaktor 1 erfolgt der Lese- und Schreibvorgang einmal auf dem FSx ONTAP, sodass wir bei beiden Validierungen den maximalen Durchsatz von 4 GB/s erreichen konnten.

Der Sync-Treiber kann einen konsistenten Durchsatz generieren, da Protokolle sofort auf die Festplatte geschrieben werden, während der Throughput-Treiber Durchsatzschübe generiert, da Protokolle in großen Mengen auf die Festplatte geschrieben werden.

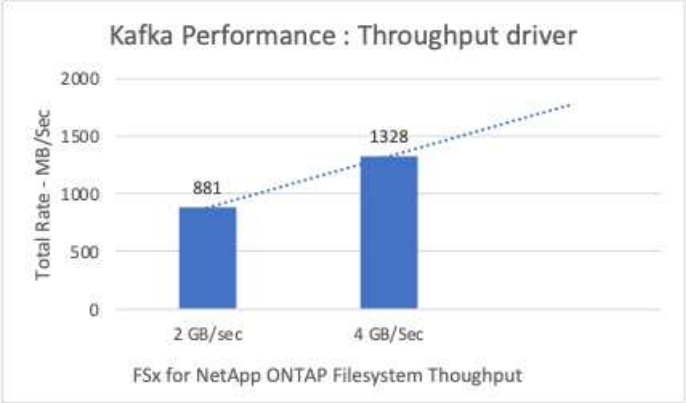
Diese Durchsatzzahlen werden für die jeweilige AWS-Konfiguration generiert. Bei höheren Leistungsanforderungen können die Instanztypen hochskaliert und für bessere Durchsatzzahlen weiter optimiert werden. Der Gesamtdurchsatz oder die Gesamtrate ist die Kombination aus Produzenten- und Verbraucherrate.

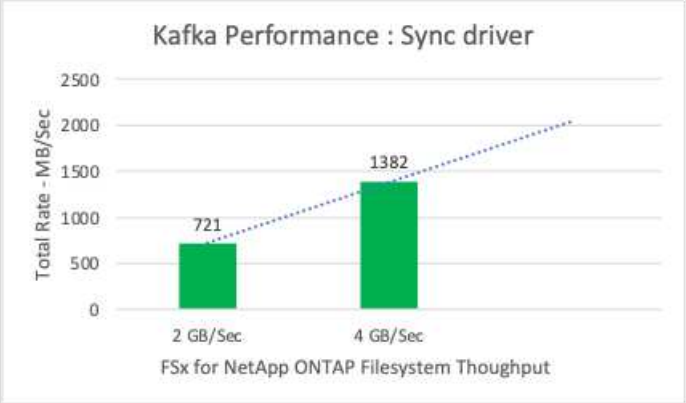




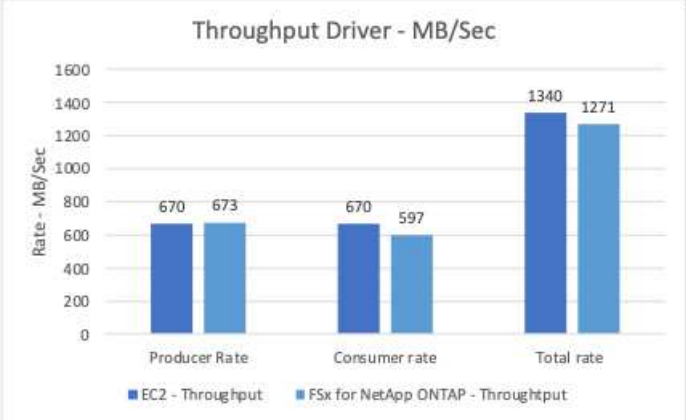
Das folgende Diagramm zeigt die Leistung von 2 GB/s für FSx ONTAP und 4 GB/s für den Kafka-Replikationsfaktor 3. Der Replikationsfaktor 3 führt den Lese- und Schreibvorgang dreimal auf dem FSx

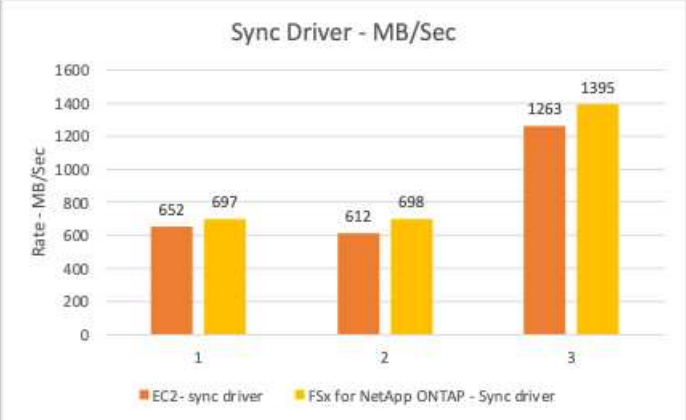
ONTAP Speicher aus. Die Gesamtrate für den Durchsatztreiber beträgt 881 MB/s, was Lese- und Schreibvorgänge für Kafka mit ungefähr 2,64 GB/s auf dem FSx ONTAP Dateisystem mit 2 GB/s ausführt, und die Gesamtrate für den Durchsatztreiber beträgt 1328 MB/s, was Lese- und Schreibvorgänge für Kafka mit ungefähr 3,98 GB/s ausführt. Die Kafka-Leistung ist linear und basierend auf dem FSx ONTAP Durchsatz skalierbar.





Das folgende Diagramm zeigt die Leistung zwischen EC2-Instanz und FSx ONTAP (Kafka-Replikationsfaktor: 3).

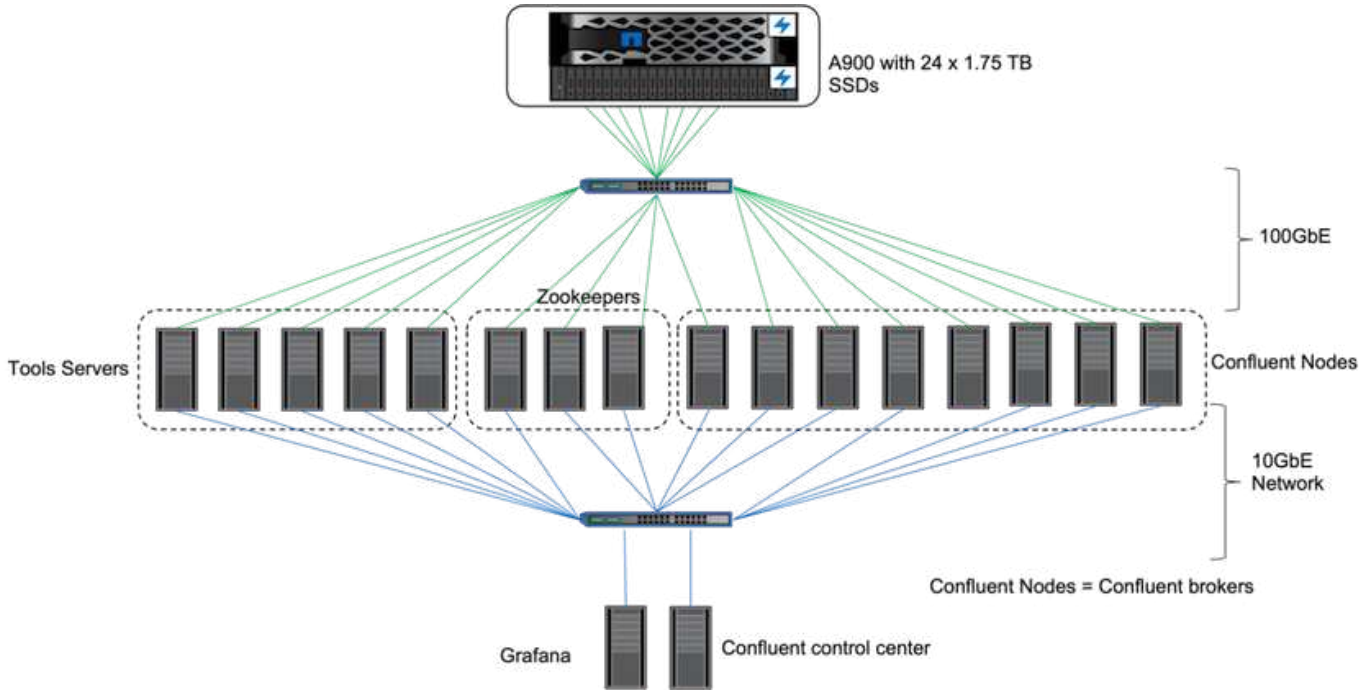




Leistungsübersicht und Validierung mit AFF A900 vor Ort

Vor Ort haben wir den NetApp AFF A900 Speichercontroller mit ONTAP 9.12.1RC1 verwendet, um die Leistung und Skalierung eines Kafka-Clusters zu validieren. Wir haben dasselbe Testbed wie bei unseren vorherigen Best Practices für Tiered Storage mit ONTAP und AFF verwendet.

Zur Evaluierung des AFF A900 haben wir Confluent Kafka 6.2.0 verwendet. Der Cluster verfügt über acht Broker-Knoten und drei Zookeeper-Knoten. Für Leistungstests haben wir fünf OMB-Workerknoten verwendet.



Storage-Konfiguration

Wir haben NetApp FlexGroups-Instanzen verwendet, um einen einzigen Namespace für Protokollverzeichnisse bereitzustellen und so die Wiederherstellung und Konfiguration zu vereinfachen. Wir haben NFSv4.1 und pNFS verwendet, um einen direkten Pfadzugriff auf die Daten des Protokollsegments zu ermöglichen.

Client-Tuning

Jeder Client hat die FlexGroup -Instanz mit dem folgenden Befehl gemountet.

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

Darüber hinaus erhöhten wir die `max_session_slots` von der Standardeinstellung `64` Zu `180` . Dies entspricht dem Standardlimit für Sitzungsslots in ONTAP.

Kafka-Broker-Tuning

Um den Durchsatz im Testsystem zu maximieren, haben wir die Standardparameter für bestimmte wichtige Thread-Pools deutlich erhöht. Wir empfehlen, für die meisten Konfigurationen die Best Practices von Confluent Kafka zu befolgen. Diese Optimierung wurde verwendet, um die Parallelität ausstehender E/A-Vorgänge zum Speicher zu maximieren. Diese Parameter können angepasst werden, um den Rechenressourcen und Speicherattributen Ihres Brokers zu entsprechen.

```
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
```

Testmethodik für Workload-Generatoren

Wir haben für den Durchsatztreiber und die Themenkonfiguration dieselben OMB-Konfigurationen wie für Cloud-Tests verwendet.

1. Eine FlexGroup -Instanz wurde mit Ansible auf einem AFF Cluster bereitgestellt.

```
---
- name: Set up kafka broker processes
  hosts: localhost
  vars:
    ntap_hostname: 'hostname'
    ntap_username: 'user'
    ntap_password: 'password'
    size: 10
    size_unit: tb
    vserver: vs1
    state: present
    https: true
    export_policy: default
    volumes:
      - name: kafka_fg_vol01
        aggr: ["aggr1_a", "aggr2_a", "aggr1_b", "aggr2_b"]
        path: /kafka_fg_vol01
  tasks:
    - name: Edit volumes
      netapp.ontap.na_ontap_volume:
        state: "{{ state }}"
        name: "{{ item.name }}"
        aggr_list: "{{ item.aggr }}"
        aggr_list_multiplier: 8
        size: "{{ size }}"
        size_unit: "{{ size_unit }}"
        vserver: "{{ vserver }}"
        snapshot_policy: none
        export_policy: default
        junction_path: "{{ item.path }}"
        qos_policy_group: none
        wait_for_completion: True
        hostname: "{{ ntap_hostname }}"
        username: "{{ ntap_username }}"
        password: "{{ ntap_password }}"
        https: "{{ https }}"
        validate_certs: false
      connection: local
      with_items: "{{ volumes }}"
```

2. pNFS wurde auf dem ONTAP SVM aktiviert.

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

3. Die Arbeitslast wurde mit dem Throughput-Treiber ausgelöst, wobei dieselbe Arbeitslastkonfiguration wie für Cloud Volumes ONTAP verwendet wurde. Siehe Abschnitt "Steady-State-Leistung " unten. Die Arbeitslast verwendete einen Replikationsfaktor von 3, was bedeutet, dass drei Kopien der Protokollsegmente in NFS verwaltet wurden.

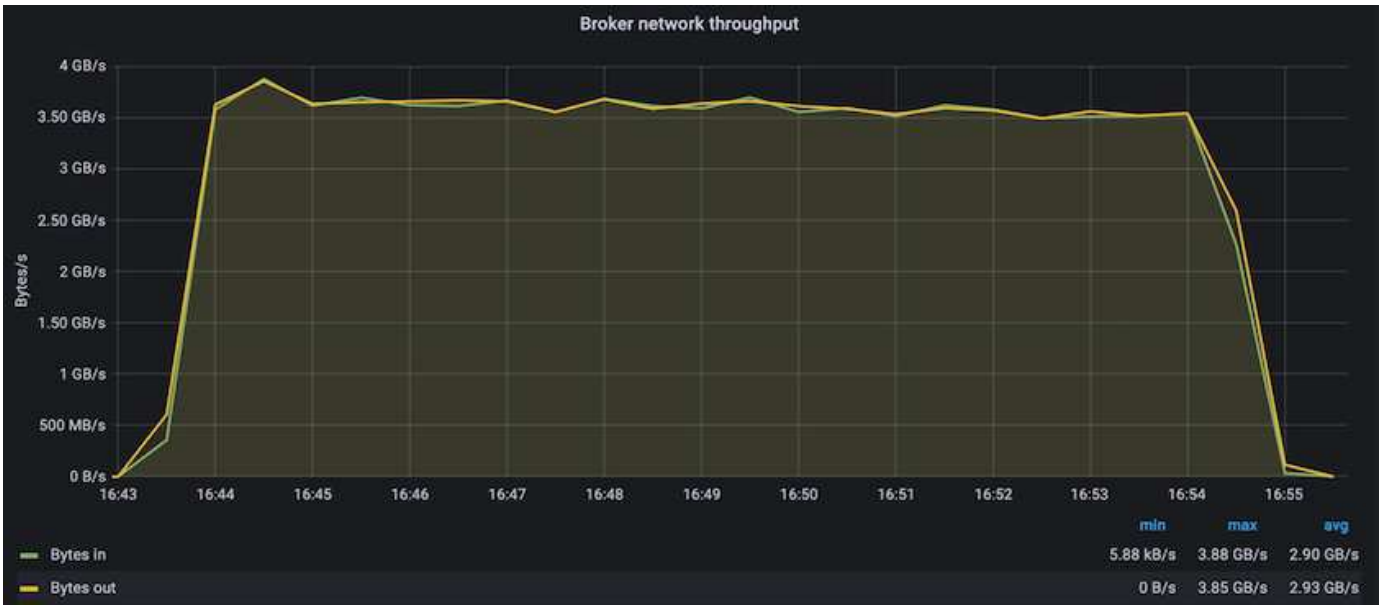
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. Abschließend haben wir Messungen mithilfe eines Rückstands durchgeführt, um die Fähigkeit der Verbraucher zu messen, die neuesten Nachrichten nachzuholen. OMB baut einen Rückstand auf, indem es Verbraucher zu Beginn einer Messung anhält. Dadurch entstehen drei verschiedene Phasen: die Erstellung eines Rückstands (Verkehr nur für Produzenten), der Abbau des Rückstands (eine Phase mit vielen Konsumenten, in der Konsumenten verpasste Ereignisse zu einem Thema nachholen) und der stationäre Zustand. Siehe Abschnitt "Extreme Leistung und Ausloten der Speichergrenzen " für weitere Informationen.

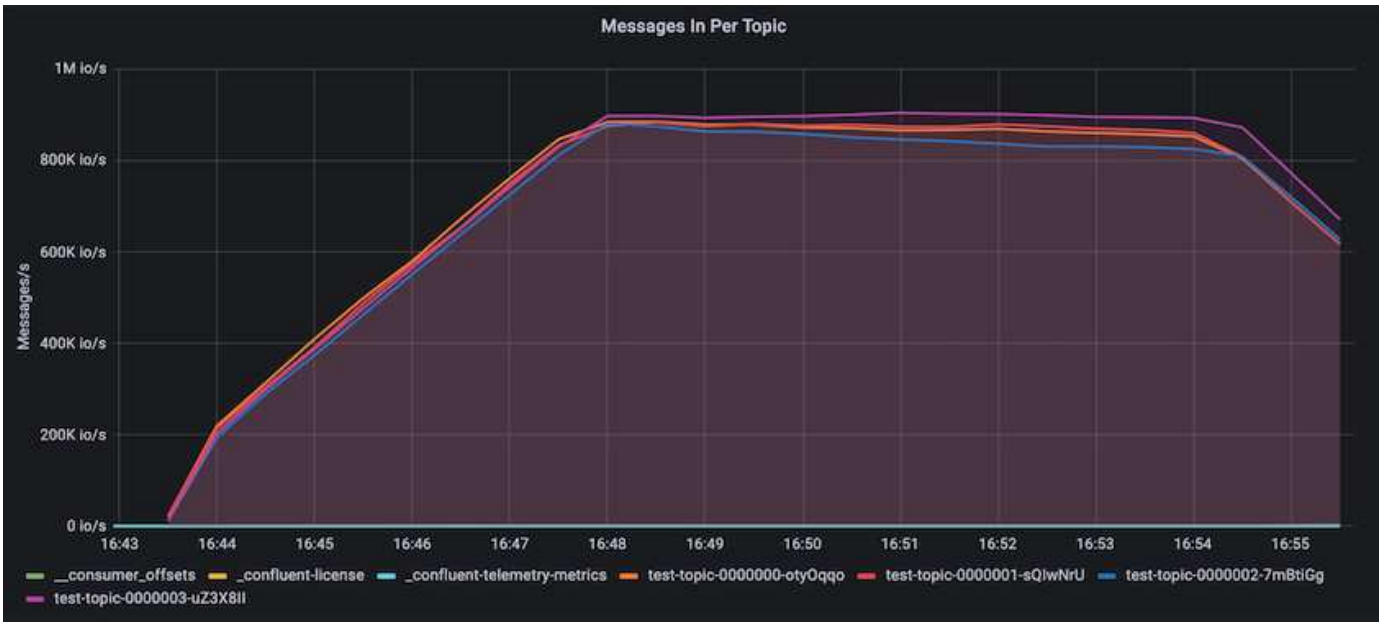
Steady-State-Leistung

Wir haben die AFF A900 mithilfe des OpenMessaging Benchmarks bewertet, um einen ähnlichen Vergleich wie für Cloud Volumes ONTAP in AWS und DAS in AWS zu ermöglichen. Alle Leistungswerte stellen den Durchsatz des Kafka-Clusters auf Produzenten- und Verbraucherebene dar.

Die konstante Leistung mit Confluent Kafka und dem AFF A900 erreichte einen durchschnittlichen Durchsatz von über 3,4 GBps für Produzenten und Verbraucher. Das sind über 3,4 Millionen Nachrichten im gesamten Kafka-Cluster. Durch die Visualisierung des anhaltenden Durchsatzes in Bytes pro Sekunde für BrokerTopicMetrics sehen wir die hervorragende Dauerleistung und den Datenverkehr, die vom AFF A900 unterstützt werden.



Dies passt gut zur Ansicht der pro Thema übermittelten Nachrichten. Das folgende Diagramm bietet eine Aufschlüsselung nach Themen. In der getesteten Konfiguration haben wir fast 900.000 Nachrichten pro Thema in vier Themenbereichen gesehen.

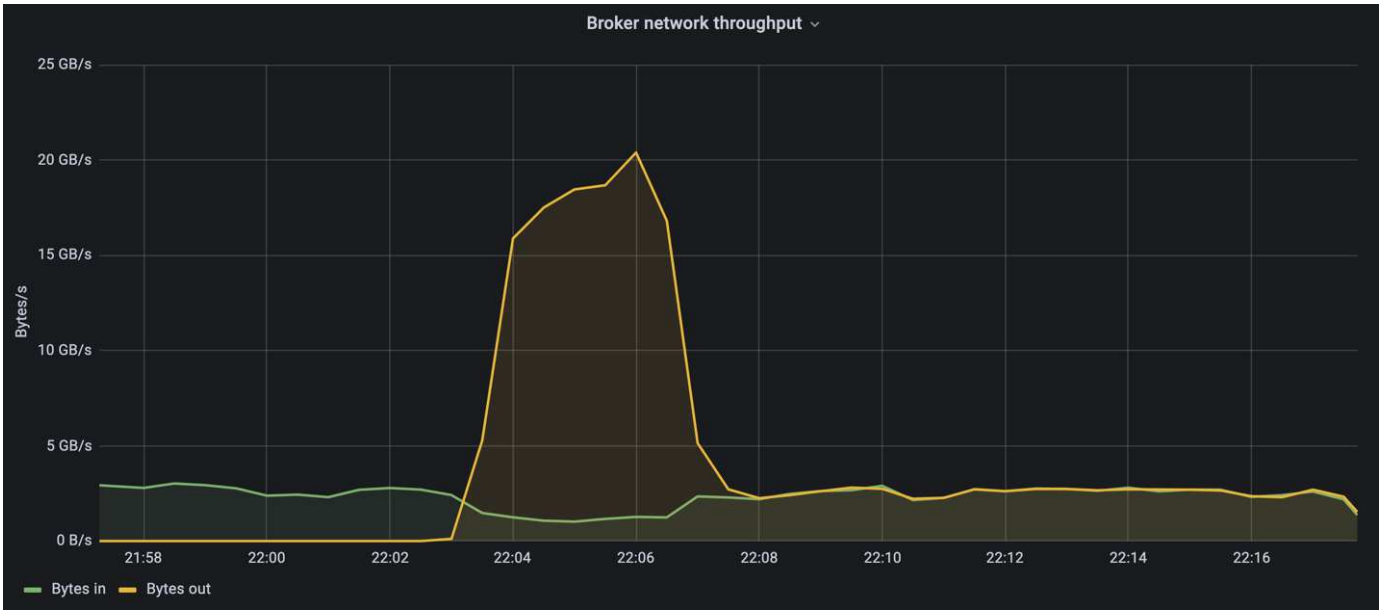


Extreme Leistung und Ausloten der Speichergrenzen

Für AFF haben wir auch mit OMB unter Verwendung der Backlog-Funktion getestet. Die Backlog-Funktion pausiert Verbraucherabonnements, während sich im Kafka-Cluster ein Rückstand an Ereignissen aufbaut. Während dieser Phase tritt nur Produzentenverkehr auf, der Ereignisse generiert, die in Protokollen festgehalten werden. Dies emuliert am ehesten die Stapelverarbeitung oder Offline-Analyse-Workflows. In diesen Workflows werden Verbraucherabonnements gestartet und müssen historische Daten lesen, die bereits aus dem Broker-Cache entfernt wurden.

Um die Speicherbeschränkungen für den Verbraucherdurchsatz in dieser Konfiguration zu verstehen, haben wir die reine Produzentenphase gemessen, um zu verstehen, wie viel Schreibverkehr der A900 aufnehmen kann. Siehe den nächsten Abschnitt "Größenberatung ", um zu verstehen, wie diese Daten genutzt werden können.

Während des Nur-Produzenten-Teils dieser Messung sahen wir einen hohen Spitzendurchsatz, der die Grenzen der A900-Leistung ausreizte (wenn andere Broker-Ressourcen nicht durch die Bedienung des Produzenten- und Verbraucherverkehrs ausgelastet waren).



Wir haben die Nachrichtengröße für diese Messung auf 16 KB erhöht, um den Overhead pro Nachricht zu begrenzen und den Speicherdurchsatz zu NFS-Mountpunkten zu maximieren.

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

Der Confluent Kafka-Cluster erreichte einen Spitzenproduzentendurchsatz von 4,03 GBps.

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

Nachdem OMB das Auffüllen des Eventbacklogs abgeschlossen hatte, wurde der Verbraucherverkehr neu gestartet. Bei Messungen mit Backlog-Drainage konnten wir einen Spitzendurchsatz der Verbraucher von über 20 GBits/s bei allen Themen feststellen. Der kombinierte Durchsatz zum NFS-Volume, auf dem die OMB-Protokolldaten gespeichert sind, lag bei etwa 30 GBits/s.

Größenberatung

Amazon Web Services bietet eine "Größentabelle" zur Größenbestimmung und Skalierung von Kafka-Clustern.

Diese Größenbestimmung bietet eine nützliche Formel zur Bestimmung des Speicherdurchsatzbedarfs für Ihren Kafka-Cluster:

Bei einem aggregierten Durchsatz, der mit einem Replikationsfaktor von r in den Cluster von tcluster erzeugt wird, beträgt der vom Broker-Speicher empfangene Durchsatz:

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

Dies lässt sich noch weiter vereinfachen:

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

Mithilfe dieser Formel können Sie die geeignete ONTAP Plattform für Ihre Kafka-Hot-Tier-Anforderungen auswählen.

Die folgende Tabelle erläutert den erwarteten Herstellerdurchsatz für den A900 mit unterschiedlichen Replikationsfaktoren:

Replikationsfaktor	Produzentendurchsatz (GPps)
3 (gemessen)	3,4
2	5,1
1	10,2

Abschluss

Die NetApp -Lösung für das Silly-Rename-Problem bietet eine einfache, kostengünstige und zentral verwaltete Speicherform für Workloads, die zuvor mit NFS nicht kompatibel waren.

Dieses neue Paradigma ermöglicht es Kunden, besser verwaltbare Kafka-Cluster zu erstellen, die sich zum Zweck der Notfallwiederherstellung und des Datenschutzes einfacher migrieren und spiegeln lassen. Wir haben außerdem festgestellt, dass NFS zusätzliche Vorteile bietet, wie etwa eine geringere CPU-Auslastung und eine schnellere Wiederherstellungszeit, eine deutlich verbesserte Speichereffizienz und eine bessere Leistung durch NetApp ONTAP.

Wo Sie weitere Informationen finden

Weitere Informationen zu den in diesem Dokument beschriebenen Informationen finden Sie in den folgenden Dokumenten und/oder auf den folgenden Websites:

- Was ist Apache Kafka?

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

- Was ist eine dumme Umbenennung?

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

- ONATP wird für Streaming-Anwendungen gelesen.

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

- NetApp Produktdokumentation

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

- Was ist NFS?

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

- Was ist eine Kafka-Partitionsneuzuweisung?

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

- Was ist der OpenMessaging-Benchmark?

  "https://openmessaging.cloud/"

- Wie migriert man einen Kafka-Broker?

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

- Wie überwachen Sie den Kafka-Broker mit Prometheus?

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

- Verwaltete Plattform für Apache Kafka

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

- Unterstützung für Apache Kafka

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

- Beratungsleistungen für Apache Kafka

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