Apache Kafka-Workloads mit NetApp NFS-Storage

NetApp Solutions

NetApp
April 26, 2024

Inhalt

Apache Kafka-Workloads mit NetApp NFS-Storage

TR-4947: Apache Kafka Workload mit NetApp NFS-Storage – Funktionsvalidierung und Performance

Shantanu Chakole, Karthikeyan Nagalingam und Joe Scott, NetApp

Kafka ist ein verteiltes Veröffentlichungsabonnensystem mit einer robusten Warteschlange, die große Mengen an Nachrichtendaten aufnehmen kann. Mit Kafka können Anwendungen Daten sehr schnell zu Themen schreiben und lesen. Aufgrund seiner Fehlertoleranz und Skalierbarkeit wird Kafka im Big Data-Bereich häufig als zuverlässige Möglichkeit genutzt, viele Datenströme sehr schnell aufzunehmen und zu verschieben. Anwendungsfälle sind beispielsweise die Stream-Verarbeitung, die Verfolgung der Website-Aktivität, die Erfassung und Überwachung von Kennzahlen, die Protokollaggregation, Echtzeitanalysen usw.

Obwohl normale Kafka-Vorgänge in NFS gut funktionieren, ist die "Blöde Umbenennung" Problem stürzt die Anwendung während der Größenänderung oder Neupartitionierung eines Kafka-Clusters ab, der auf NFS ausgeführt wird. Dies ist ein bedeutender Aspekt, da die Größe eines Kafka-Clusters zum Lastausgleich oder zur Wartung angepasst oder neu partitioniert werden muss. Weitere Details finden Sie hier "Hier".

In diesem Dokument werden die folgenden Themen beschrieben:

- Das Problem mit der Silly-Umbenennung und die Lösungsvalidierung
- Verringerung der CPU-Auslastung, um die I/O-Wartezeit zu verringern
- Kürzere Recovery-Zeit für Kafka-Broker
- Performance in der Cloud und lokal

Gründe für NFS-Storage für Kafka-Workloads

Kafka-Workloads in Produktionsapplikationen können riesige Datenmengen zwischen Applikationen streamen. Diese Daten werden in den Kafka-Broker-Nodes im Kafka-Cluster aufbewahrt und gespeichert. Kafka ist auch für Verfügbarkeit und Parallelität bekannt, was erreicht wird, indem Themen in Partitionen unterteilt und diese Partitionen dann im gesamten Cluster repliziert werden. Das bedeutet, dass die riesige Datenmenge, die durch ein Kafka-Cluster fließt, sich in der Regel in der Größe multipliziert. NFS macht die Ausbalancierung der Daten, da sich die Anzahl der Broker sehr schnell und einfach ändert. In großen Umgebungen erfolgt die Umverteilung von Daten innerhalb von das, wenn die Anzahl der Broker sich sehr ändert und in den meisten Kafka-Umgebungen ändert sich häufig die Anzahl der Broker.

Weitere Vorteile:

- **Maturity.** NFS ist ein ausgereiftes Protokoll, was bedeutet, dass die meisten Aspekte der Implementierung, Sicherung und Nutzung gut verstanden sind.
- **Open.** NFS ist ein offenes Protokoll, dessen Weiterentwicklung in den Internet-Spezifikationen als freies und offenes Netzwerkprotokoll dokumentiert wird.
- **Kosteneffektiv.** NFS ist eine kostengünstige Lösung für die gemeinsame Nutzung von Netzwerkdateien, die einfach eingerichtet werden kann, da sie die vorhandene Netzwerkinfrastruktur nutzt.

- **Zentral verwaltet.** die zentrale Verwaltung von NFS verringert den Bedarf an zusätzlicher Software und Speicherplatz auf einzelnen Benutzersystemen.
- **Distributed.** NFS kann als verteiltes Dateisystem verwendet werden, wodurch der Bedarf an Wechseldatenträgern reduziert wird.

Warum NetApp für Kafka-Workloads?

Die NetApp NFS-Implementierung gilt als Gold-Standard für das Protokoll und wird in unzähligen Enterprise-NAS-Umgebungen eingesetzt. Neben der Glaubwürdigkeit von NetApp bietet es zudem folgende Vorteile:

- Zuverlässigkeit und Effizienz
- Skalierbarkeit und Performance
- Hochverfügbarkeit (HA-Partner in einem NetApp ONTAP Cluster)
- Datensicherung
  - **Disaster Recovery (NetApp SnapMirror).** Ihre Site ist ausgefallen oder Sie möchten an einem anderen Standort starten und an der Stelle fortfahren, an der Sie aufgehört hatten.
  - Einfaches Management Ihres Storage-Systems (Administration und Management mit NetApp OnCommand).
  - **Load Balancing.** 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 Problem, Daten bei NFS- zu Kafka-Workloads umbenennen

Kafka wird mit der Annahme erstellt, dass das zugrunde liegende Dateisystem POSIX-kompatibel ist: Zum Beispiel XFS oder Ext4. Beim Ausbalancieren von Ressourcen in Kafka werden Dateien entfernt, während die Anwendung sie weiterhin verwendet. Ein POSIX-konformes Dateisystem ermöglicht das Aufheben der Verknüpfung zum Fortfahren. Allerdings wird die Datei erst entfernt, nachdem alle Verweise auf die Datei verschwunden sind. Wenn das zugrunde liegende Dateisystem mit einem Netzwerk verbunden ist, fängt der NFS-Client die Aufheben der Verknüpfung ab und verwaltet den Workflow. Da für die nicht verknüpfte Datei noch offene Öffnungen vorhanden sind, sendet der NFS-Client eine Umbenennungsanforderung an den NFS-Server und führt beim letzten Schließen der nicht verknüpften Datei einen Entnahmierungsvorgang für die umbenannte Datei aus. Dieses Verhalten wird allgemein als „dummer über NFS“ bezeichnet und wird vom NFS-Client orchestriert.

Jeder Kafka-Broker, der Storage von einem NFSv3-Server verwendet, läuft aufgrund dieses Verhaltens mit Problemen zusammen. Das NFSv4.x-Protokoll verfügt jedoch über Funktionen, um dieses Problem zu beheben, indem es dem Server erlaubt, die Verantwortung für die geöffneten, nicht verknüpften Dateien zu übernehmen. NFS-Server, die diese optionale Funktion unterstützen, kommunizieren zum Zeitpunkt des Dateiöffnens die Eigentumsfähigkeit an den NFS-Client. Der NFS-Client beendet dann das Aufheben der Verbindung, wenn offene Verbindungen vorhanden sind, und ermöglicht dem Server, den Fluss zu verwalten. Obwohl die NFSv4-Spezifikation Richtlinien für die Implementierung bietet, gab es bisher keine bekannten NFS-Server-Implementierungen, die diese optionale Funktion unterstützen.

Die folgenden Änderungen sind für den NFS-Server und den NFS-Client erforderlich, um das Problem der unsinnigen Umbenennung zu lösen:

- **Änderungen am NFS-Client (Linux).** zum Zeitpunkt des Dateieröffnens antwortet der NFS-Server mit einem Flag, der die Fähigkeit anzeigt, das Aufheben der Verknüpfung von geöffneten Dateien zu handhaben. Durch clientseitige NFS-Änderungen kann der NFS-Server das Aufheben der Verknüpfung in Gegenwart des Flags bearbeiten. 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 geöffneten Daten. Das Aufheben der Verknüpfung zu einer vorhandenen offenen Datei wird nun vom Server verwaltet, um POSIX-Semantik zu entsprechen. Wenn das letzte Öffnen geschlossen wird, initiiert der NFS-Server dann das eigentliche Entfernen der Datei und vermeidet so den dummen Umbenennungsprozess. In der neuesten Version, ONTAP 9.12.1, hat der ONTAP-NFS-Server diese Funktion implementiert.

Durch die oben genannten Änderungen am NFS-Client und -Server kann Kafka sicher von allen Vorteilen von Network-Attached NFS-Storage profitieren.

Funktionale Validierung – Korrektur von Silly-Umbenennung

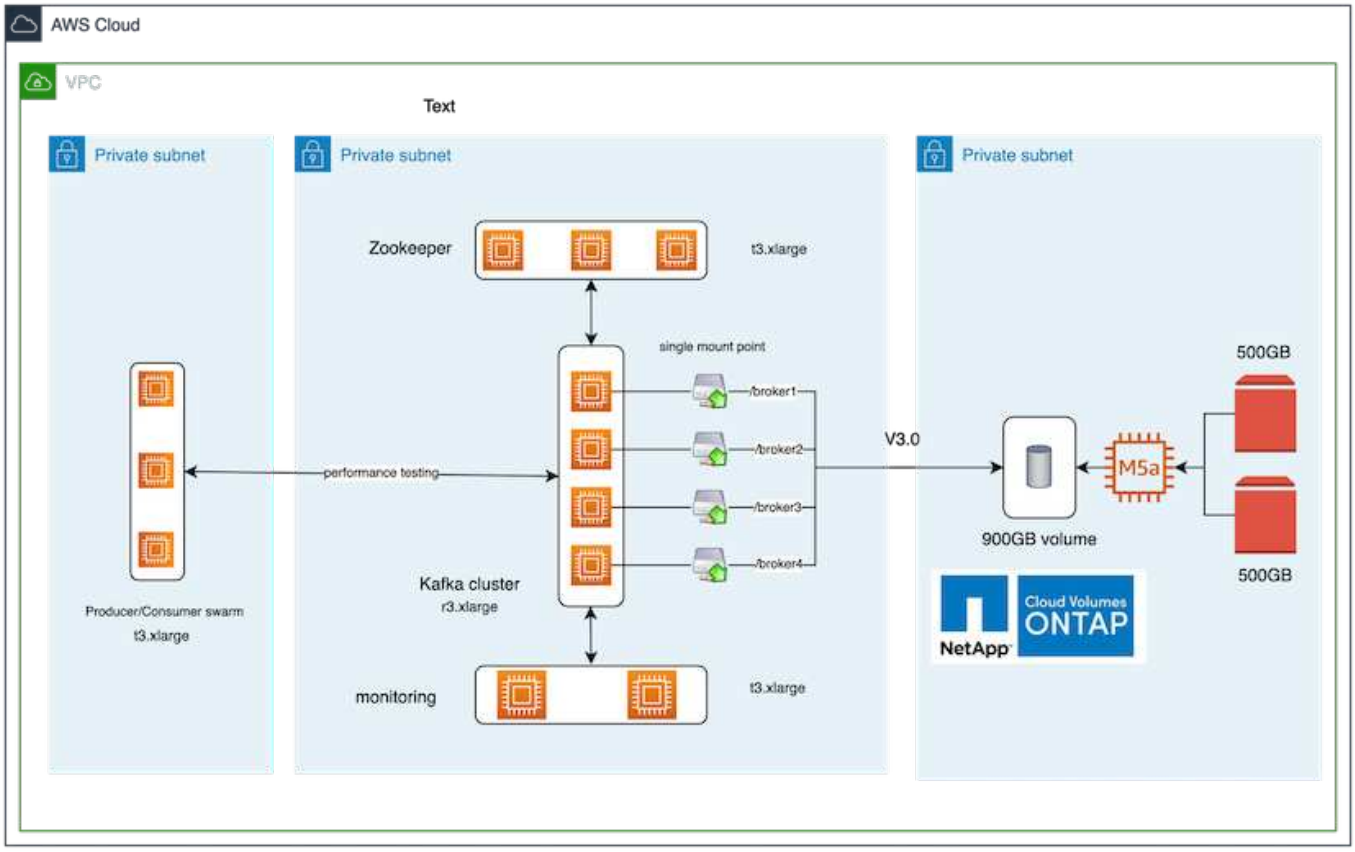
Für die funktionale Validierung haben wir gezeigt, dass ein Kafka-Cluster mit einem NFSv3-Mount für Storage Kafka-Vorgänge wie Partitionsumverteilung nicht ausführen kann, während ein anderer Cluster, der auf NFSv4 mit der Korrektur gemountet ist, die gleichen Operationen ohne Unterbrechungen durchführen kann.

Einrichtung der Validierung

Das Setup läuft auf AWS. Die folgende Tabelle zeigt die verschiedenen Plattformkomponenten und Umgebungskonfigurationen, die für die Validierung verwendet werden.

Plattformkomponente	Umgebungskonfiguration
Confluent Platform Version 7.2.1	• 3 x Zookeeper – t3.xlarge • 4 x Broker-Server – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x Leitstelle – t3.xlarge • 3 x Hersteller/Verbraucher
Betriebssystem auf allen Knoten	RHEL8.7oder höher
NetApp Cloud Volumes ONTAP Instanz	Single-Node-Instanz – M5.2xLarge

Die folgende Abbildung zeigt die Architekturkonfiguration dieser Lösung.



Architekturfluss

- **Compute.** Wir nutzten einen vier-Knoten-Kafka-Cluster mit einem drei-Knoten-Zookeeper-Ensemble, das auf dedizierten Servern lief.
- **Monitoring.** für eine Prometheus-Grafana-Kombination haben wir zwei Knoten verwendet.
- **Workload.** zur Generierung von Workloads haben wir ein separates Cluster mit drei Nodes verwendet, das die Produktion und Nutzung aus diesem Kafka-Cluster ermöglicht.
- **Speicher.** Wir verwendeten eine NetApp Cloud Volumes ONTAP-Instanz mit einem einzigen Node, wobei zwei 500 GB GP2 AWS-EBS Volumes an die Instanz angeschlossen waren. Diese Volumes wurden dann dem Kafka Cluster als einzelnes NFSv4.1-Volume über eine LIF offengelegt.

Die Standardeigenschaften von Kafka wurden für alle Server ausgewählt. Das gleiche wurde für den Zookeeper Schwarm getan.

Methodik des Testens

1. Aktualisierung `-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. Zwei ähnliche Kafka-Cluster wurden mit dem folgenden Unterschied erstellt:
   - **Cluster 1.** der Backend-NFS v4.1-Server mit produktionsbereitem ONTAP Version 9.12.1 wurde von einer NetApp CVO-Instanz gehostet. RHEL 8.7/RHEL 9.1 wurden auf den Brokern 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:

```
[root@ip-172-30-0-160 demo]# kafka-topics --bootstrap-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:90
92,172.30.0.123:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: 2ty29xfhQLq65HKsUQv-pg PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 4       Replicas: 4,1   Isr: 4,1        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 2       Replicas: 2,4   Isr: 2,4        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 3       Replicas: 3,2   Isr: 3,2        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 1       Replicas: 1,3   Isr: 1,3        Offline:
```

   Cluster 2:

```
[root@ip-172-30-0-198 demo]# kafka-topics --bootstrap-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:90
92,172.30.0.204:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: AwQpsZTQShyeMIhaquCG3Q PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 2       Replicas: 2,3   Isr: 2,3        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 3       Replicas: 3,1   Isr: 3,1        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 1       Replicas: 1,4   Isr: 1,4        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 4       Replicas: 4,2   Isr: 4,2        Offline:
```

4. In diese neu erstellten Themen wurden für beide Cluster Daten geladen. Dies wurde mit dem Producer-perf-Test Toolkit durchgeführt, das im Standard-Kafka-Paket 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 eine Integritätsprüfung über Telnet 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:

```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

Connection closed by foreign host.
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.198 9092
Trying 172.30.0.198...
Connected to 172.30.0.198.
Escape character is '^]'.
^[
```

6. Um die Fehlerbedingung auszulösen, die einen Absturz von Kafka-Clustern mit NFSv3-Storage-Volumes verursacht, haben wir bei beiden Clustern den Prozess der Partitionsumzuordnung initiiert. Die Partitionsumverteilung wurde mit durchgeführt `kafka-reassign-partitions.sh`. Der detaillierte Prozess sieht wie folgt aus:
   a. Um die Partitionen für ein Thema in einem Kafka-Cluster neu zuzuweisen, haben wir die vorgeschlagene Neuzuweisung von config JSON 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. Die generierte Neuzuweisung JSON wurde dann in gespeichert `/tmp/reassignment- file.json`.
   c. Der eigentliche Vorgang der Partitionsumverteilung 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. Nach ein paar Minuten, als die Neuzuweisung abgeschlossen war, zeigte eine weitere Integritätsprüfung der Broker, dass Cluster mit NFSv3-Storage-Volumes ein albernes Problem hatten und abgestürzt waren, während Cluster 1 mit NetApp ONTAP NFSv4.1-Storage-Volumes mit der Korrektur 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. Bei der Überprüfung der Kafka-Protokollverzeichnisse wurde deutlich, dass Cluster 1 mit NetApp ONTAP NFSv4.1-Storage-Volumes mit der Korrektur eine saubere Partitionszuweisung hatte, während Cluster 2 mit generischem NFSv3-Storage nicht auf alberne Probleme beim Umbenennen zurückzuführen war, was zum Absturz führte. Das folgende Bild zeigt den Partitionsausgleich von Cluster 2, was zu einem dummen Umbenennungsproblem auf NFSv3 Storage 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 Partition Rebalancing von Cluster 1 mit NetApp NFSv4.1 Storage.

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

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

Warum NetApp NFS für Kafka-Workloads?

Da es jetzt in NFS-Storage mit Kafka eine Lösung für das Problem des unsinnigen Umbenennungen gibt, können Sie robuste Implementierungen erstellen, die NetApp ONTAP-Storage für Ihren Kafka-Workload nutzen. Dies verringert nicht nur den betrieblichen Overhead, sondern bringt auch die folgenden Vorteile für Ihre Kafka-Cluster mit sich:

- **Geringere CPU-Auslastung bei Kafka-Brokern** mittels disaggregiertem NetApp ONTAP Storage werden Festplatten-I/O-Operationen vom Broker getrennt und damit der CPU-Bedarf reduziert.
- **Schnellere Recovery-Zeit für Broker.** Da disaggregierter NetApp ONTAP-Storage über Kafka-Broker-Nodes hinweg gemeinsam genutzt wird, kann eine neue Compute-Instanz einen fehlerhaften Broker jederzeit und in einem Bruchteil der Zeit ersetzen, die in herkömmlichen Kafka-Implementierungen üblich ist, ohne die Daten neu zu erstellen.
- **Storage-Effizienz.** Da die Storage-Ebene der Applikation jetzt über NetApp ONTAP bereitgestellt wird, profitieren Kunden von allen Vorteilen der Storage-Effizienz, die ONTAP bietet, wie Inline-Datenkomprimierung, Deduplizierung und Data-Compaction.

Diese Vorteile wurden in Testfällen, die wir in diesem Abschnitt näher erläutern, getestet und validiert.

Reduzierte CPU-Auslastung beim Kafka-Broker

Wir stellten fest, dass die CPU-Auslastung insgesamt niedriger ist als die des Pendants von das, als wir ähnliche Workloads auf zwei separaten Kafka Clustern ausgeführt hatten, die in den technischen Spezifikationen identisch waren, sich aber in ihren Storage-Technologien unterschieden. Wenn Kafka Cluster ONTAP-Storage verwendet, ist nicht nur die CPU-Auslastung insgesamt geringer, sondern die Steigerung der CPU-Auslastung zeigte auch einen sanfteren Verlauf als in einem das-basierten Kafka Cluster.

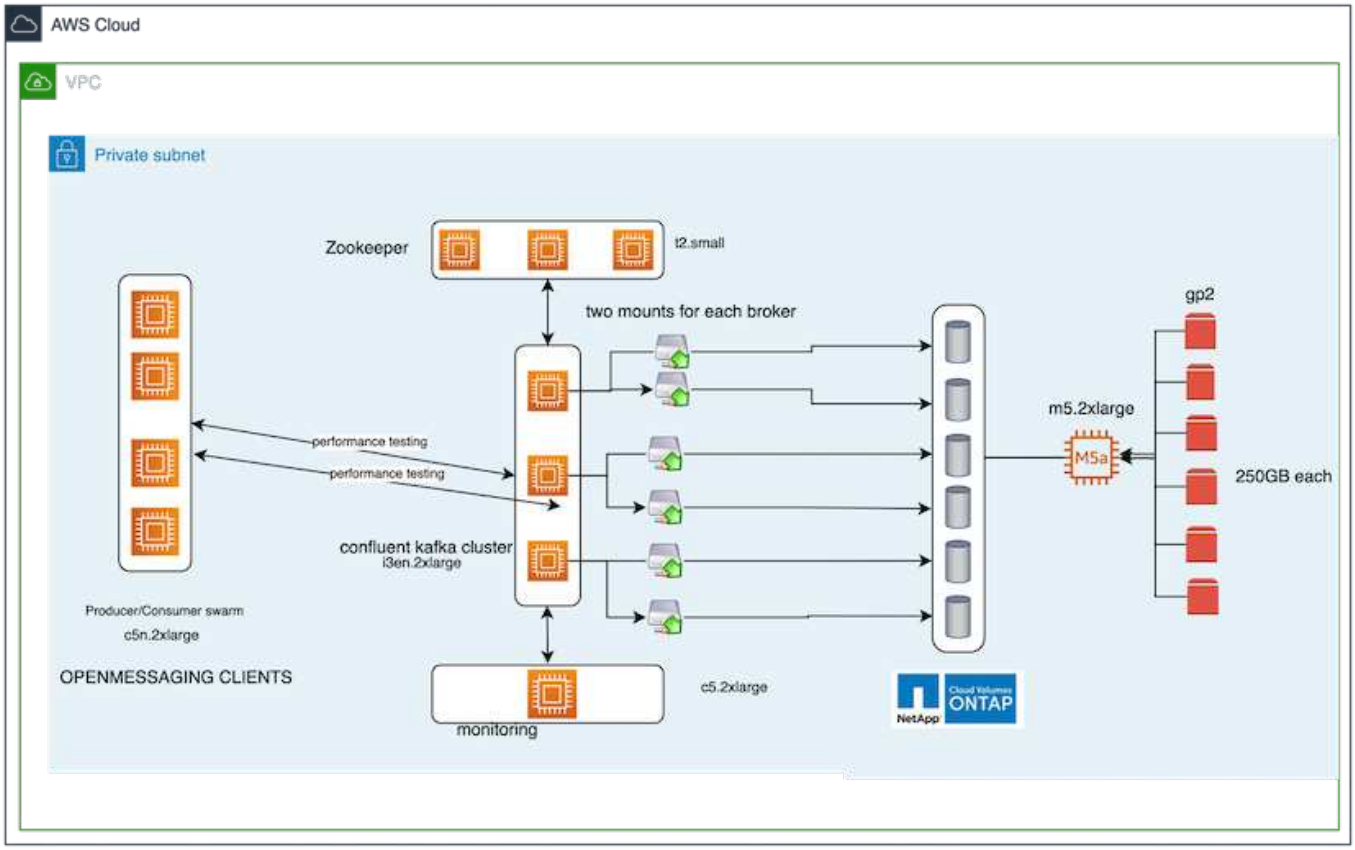
Einrichtung der Architektur

In der folgenden Tabelle ist die Umgebungskonfiguration aufgeführt, mit der die Verringerung der CPU-Auslastung demonstriert wird.

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

Benchmark-Tool

Das in diesem Testfall verwendete Benchmark-Tool ist das "OpenMessaging" Framework: OpenMessaging ist herstellerneutral 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.



- **Compute.** Wir benutzten einen drei-Knoten-Kafka-Cluster mit einem drei-Knoten-Zookeeper-Ensemble, das auf dedizierten Servern läuft. Jeder Broker hatte zwei NFSv4.1-Mount-Punkte auf ein einzelnes Volume in der NetApp CVO-Instanz über eine dedizierte LIF.
- **Monitoring.** für eine Prometheus-Grafana-Kombination haben wir zwei Knoten verwendet. Zur Generierung von Workloads verfügen wir über ein separates Cluster mit drei Nodes, das in diesen Kafka-Cluster erzeugt und genutzt werden kann.

- **Storage.** Wir verwendeten eine NetApp Cloud Volumes ONTAP-Instanz mit einem Node, auf der sechs 250 GB GP2 AWS-EBS Volumes gemountet wurden. Diese Volumes wurden dann über dedizierte LIFs dem Kafka-Cluster als sechs NFSv4.1-Volumes offengelegt.
- **Konfiguration.** die beiden konfigurierbaren Elemente in diesem Testfall waren Kafka-Broker und OpenMessaging-Workloads.
  - **Broker config.** folgende Spezifikationen wurden für die Kafka-Broker ausgewählt. Wir haben den Replikationsfaktor 3 für alle Messungen verwendet, wie unten hervorgehoben 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
```

- **OpenMessaging Benchmark (OMB) Workload-Konfiguration.** die folgenden Spezifikationen wurden bereitgestellt. Wir haben einen angestrebten Erzeugertarif angegeben, der unten hervorgehoben wurde.

```
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
```

Methodik des Testens

1. Es wurden zwei ähnliche Cluster mit jeweils eigenen Benchmark-Cluster-Schwärmen erstellt.
   - **Cluster 1.** NFS-basierter Kafka-Cluster.

    - **Cluster 2.** das-basierter Kafka-Cluster.
2. Mit einem OpenMessaging-Befehl wurden auf jedem Cluster ähnliche Workloads ausgelöst.

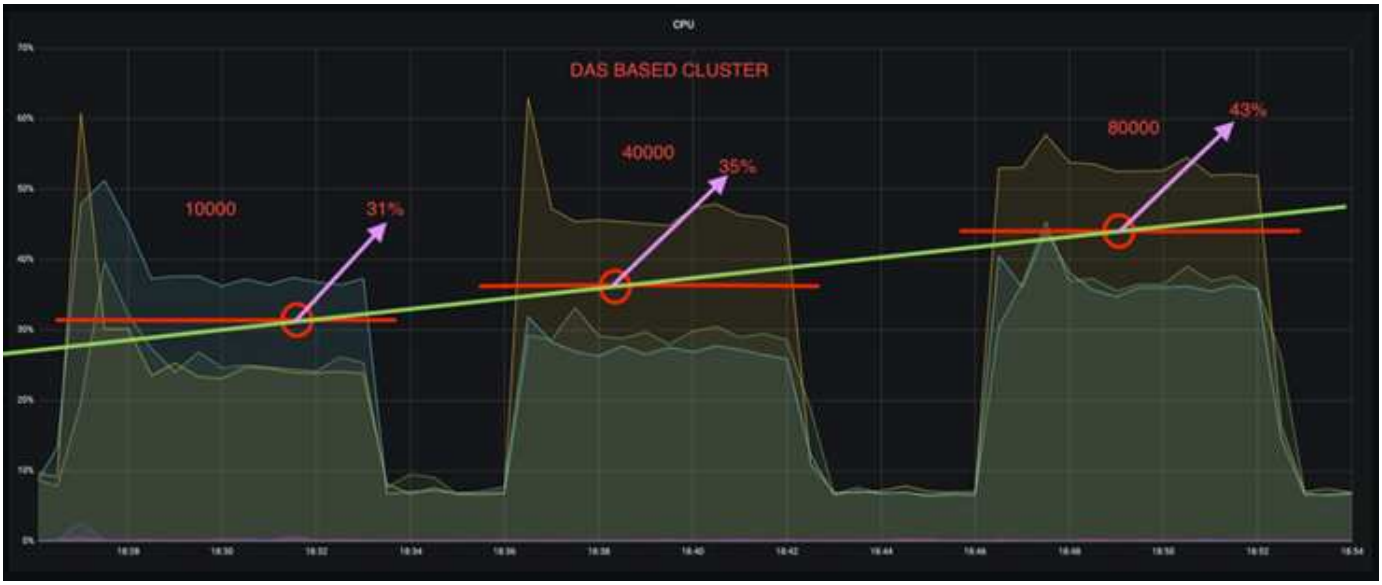
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

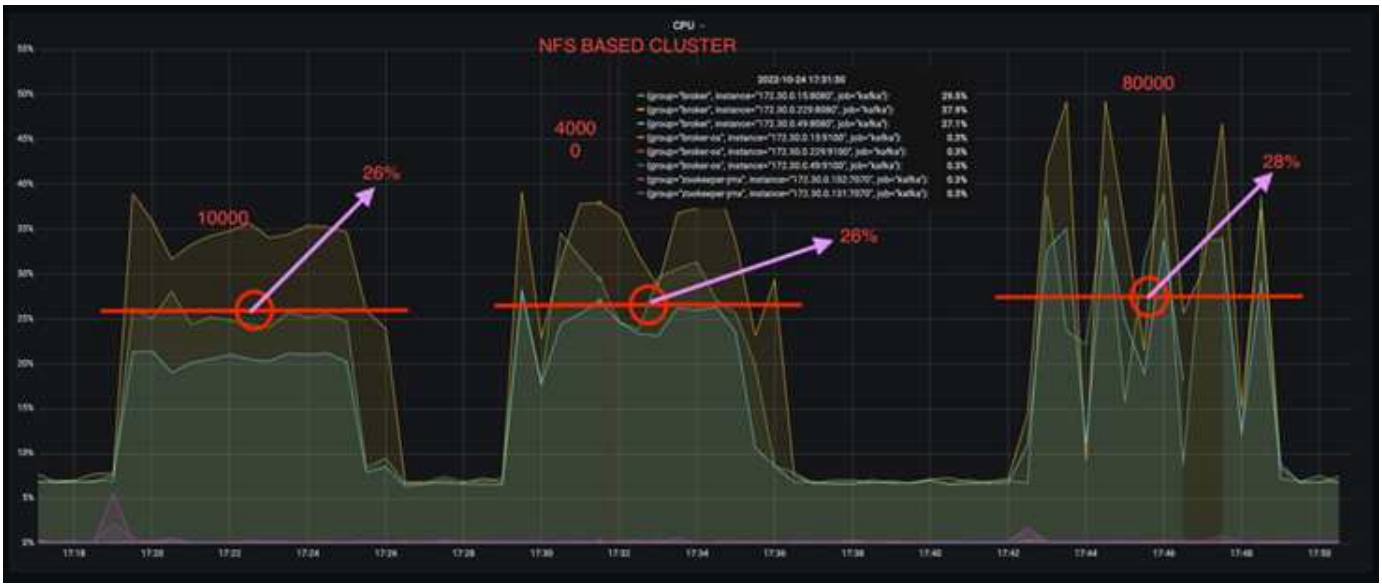
3. Die Konfiguration der Erzeugungsrate wurde in vier Iterationen erhöht, und die CPU-Auslastung wurde mit Grafana aufgezeichnet. Die Erzeugungsrate wurde auf die folgenden Stufen eingestellt:
    - 10,000
    - 40,000
    - 80,000
    - 100,000

Beobachtung

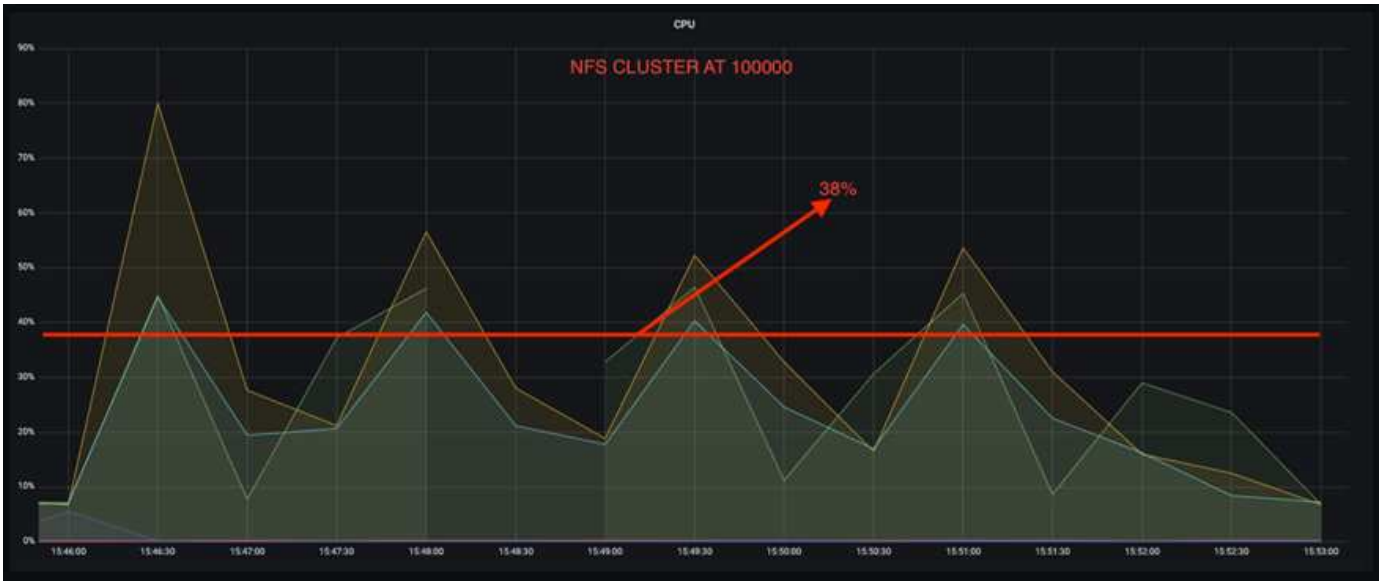
Es gibt zwei Hauptvorteile bei der Nutzung von NetApp NFS-Storage mit Kafka:

- **Sie können die CPU-Nutzung um fast ein Drittel reduzieren.** die CPU-Auslastung bei ähnlichen Workloads war bei NFS im Vergleich zu das SSDs niedriger; die Einsparungen reichen von 5 % bei geringeren Produktraten bis zu 32 % bei höheren Produktraten.
- **Eine Verdreifachung der CPU-Auslastung bei höheren Erzeugungsraten.** wie erwartet gab es eine Aufwärtsbewegung für die Erhöhung der CPU-Auslastung, da die Erzeugungsraten erhöht wurden. Die CPU-Auslastung bei Kafka-Brokern mit das ist jedoch von 31 % bei der niedrigeren Produktzrate auf 70 % bei der höheren Produktzrate gestiegen, was einer Steigerung um 39 % entspricht. Bei einem NFS Storage Backend stieg die CPU-Auslastung von 26% auf 38%, was einer Steigerung von 12% entspricht.





Außerdem zeigt das bei 100,000 Meldungen eine höhere CPU-Auslastung als ein NFS-Cluster.



Schnellere Broker Recovery

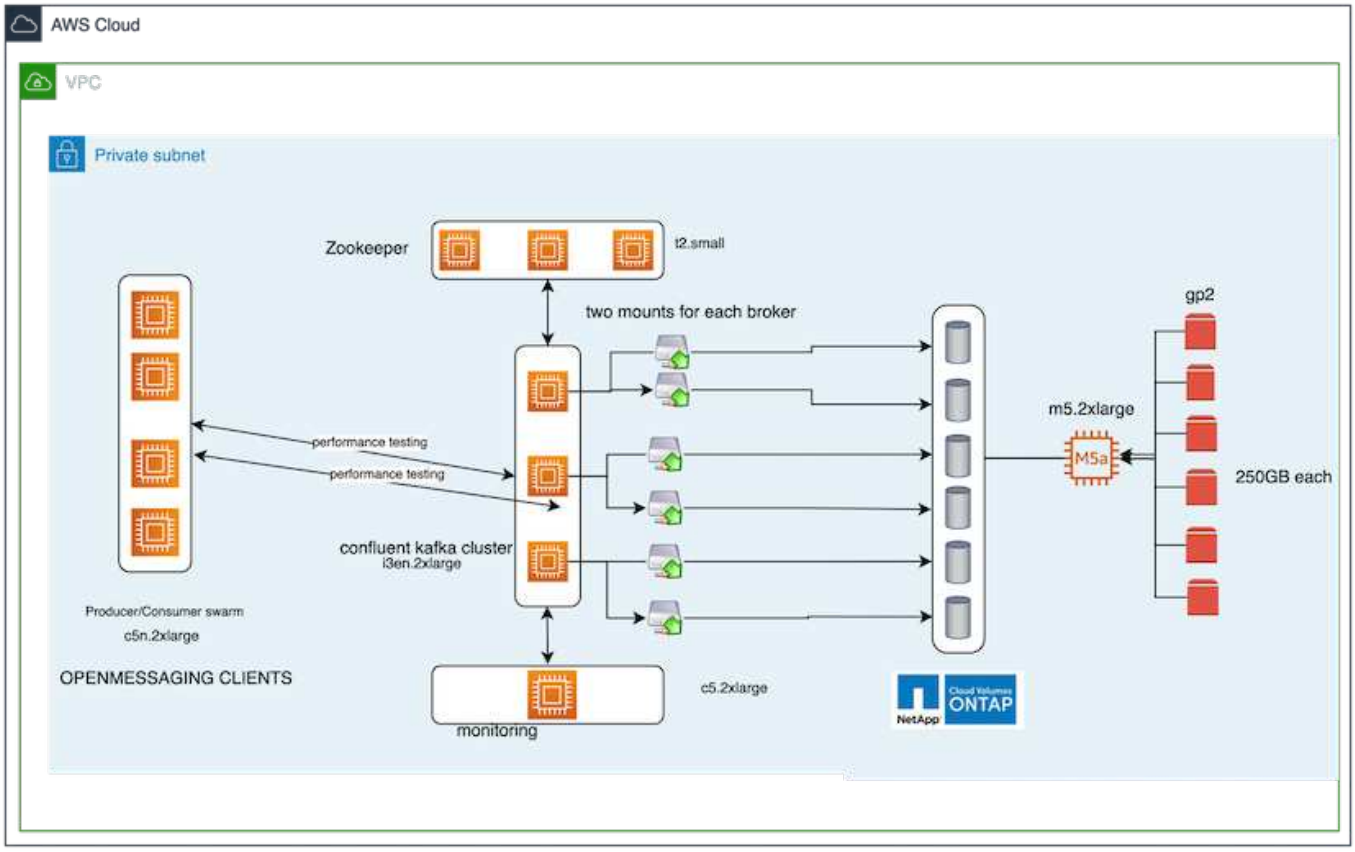
Wir haben herausgefunden, dass Kafka-Broker sich mit Shared-NetApp-NFS-Storage schneller wiederherstellen können. Wenn ein Broker in einem Kafka-Cluster abstürzt, kann dieser Broker durch einen gesunden Broker mit derselben Broker-ID ersetzt werden. Bei diesem Testfall stellten wir fest, dass im Falle eines das-basierten Kafka-Clusters die Daten auf einem neu hinzugefügten fehlerfreien Broker neu erstellt werden, was sehr zeitaufwendig ist. Bei einem NFS-basierten Kafka Cluster von NetApp liest der ersetzende Broker weiterhin Daten aus dem vorherigen Log-Verzeichnis und stellt damit eine wesentlich schnellere Wiederherstellung her.

Einrichtung der Architektur

In der folgenden Tabelle wird die Umgebungskonfiguration für ein Kafka-Cluster mithilfe von NAS gezeigt.

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

In der folgenden Abbildung ist die Architektur eines NAS-basierten Kafka-Clusters dargestellt.



- **Compute.** Ein Kafka-Cluster mit drei Knoten mit einem Zookeeper-Ensemble, das auf dedizierten Servern läuft. Jeder Broker verfügt über zwei NFS-Mount-Punkte zu einem einzelnen Volume in der NetApp CVO-Instanz über eine dedizierte LIF.
- **Monitoring.** zwei Knoten für eine Prometheus-Grafana Kombination. Zur Generierung von Workloads verwenden wir ein separates Cluster mit drei Nodes, das diesen Kafka-Cluster produzieren und nutzen kann.
- **Storage.** Eine NetApp Cloud Volumes ONTAP-Instanz mit einem Node, auf der sechs 250-GB-GP2-AWS-EBS-Volumes gemountet sind. Diese Volumes werden dann über dedizierte LIFs dem Kafka-Cluster als sechs NFS-Volumes offengelegt.
- **Broker-Konfiguration.** das einzige konfigurierbare Element in diesem Testfall sind Kafka-Broker. Für die Kafka-Broker wurden folgende Spezifikationen ausgewählt. Der `replica.lag.time.mx.ms` Wird auf einen hohen Wert gesetzt, da dadurch festgelegt wird, wie schnell ein bestimmter Knoten aus der ISR-Liste entfernt wird. Wenn Sie zwischen schlechten und gesunden 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
```

Methodik des Testens

1. Es wurden zwei ähnliche Cluster erstellt:
   - Ein EC2-basiertes, konfluent Cluster.
   - Ein konfluent NetApp NFS-basiertes Cluster.
2. Ein Standby-Kafka-Node wurde mit einer identischen Konfiguration wie die Nodes aus dem ursprünglichen Kafka-Cluster erstellt.
3. Auf jedem der Cluster wurde ein Beispielthema erstellt und ungefähr 110 GB Daten wurden von jedem der Broker aufgefüllt.
   - **EC2-basierter Cluster.** Ein Kafka-Broker-Datenverzeichnis ist zugeordnet `/mnt/data-2` (In der folgenden Abbildung: Broker-1 von cluster1 [left Terminal]).
   - **NetApp NFS-basierter Cluster.** Ein Kafka Broker Datenverzeichnis ist auf NFS Point montiert `/mnt/data` (In der folgenden Abbildung, Broker-1 von cluster2 [rechtes Terminal]).

4. In jedem Cluster wurde Broker-1 beendet, um einen fehlgeschlagenen Recovery-Prozess für Broker 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 wie folgt identifiziert wird:
   - **IP-Adresse.** wird zugewiesen, indem die fehlgeschlagene Broker-IP dem Standby-Broker neu zugewiesen wird.
   - **Broker-ID.** Diese wurde im Standby-Broker konfiguriert `server.properties`.
6. Bei IP-Zuweisung wurde der Kafka-Dienst auf dem Standby-Broker gestartet.
7. Nach einer Weile wurden die Serverprotokolle abgerufen, um die Zeit zu prüfen, die für das Erstellen von Daten auf dem Ersatz-Node im Cluster erforderlich war.

Beobachtung

Die Recovery des Brokers Kafka war nahezu neunmal schneller. Die Wiederherstellung eines ausgefallenen Broker-Nodes dauerte bei der Nutzung von NetApp NFS Shared Storage erheblich schneller als bei der Nutzung das-SSDs in einem Kafka Cluster. Bei 1 TB Themdaten betrug die Recovery-Zeit für ein das-basiertes Cluster 48 Minuten. Bei einem Kafka Cluster auf NetApp NFS-Basis dauerte die Recovery weniger als 5 Minuten.

Wir beobachteten, dass der EC2-basierte Cluster 10 Minuten benötigt, um die Wiederherstellung der 110 GB Daten auf dem neuen Broker Node durchzuführen, während der NFS-basierte Cluster die Recovery innerhalb von 3 Minuten abgeschlossen hat. Wir beobachteten auch in den Protokollen, dass Verbraucheroffsets für die Partitionen für EC2 0 waren, während auf dem NFS-Cluster Verbraucheroffsets vom vorherigen Broker abgeholt 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 Backup-Knoten wurde um 08:55:53,730 gestartet.

```
2  [2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
3  [2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (org
4  [2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
5  [2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.30
6  [2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. Der Datenneuerstellungsprozess endete um 09:05:24,860. Die Verarbeitung von 110 GB Daten dauerte ca. 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

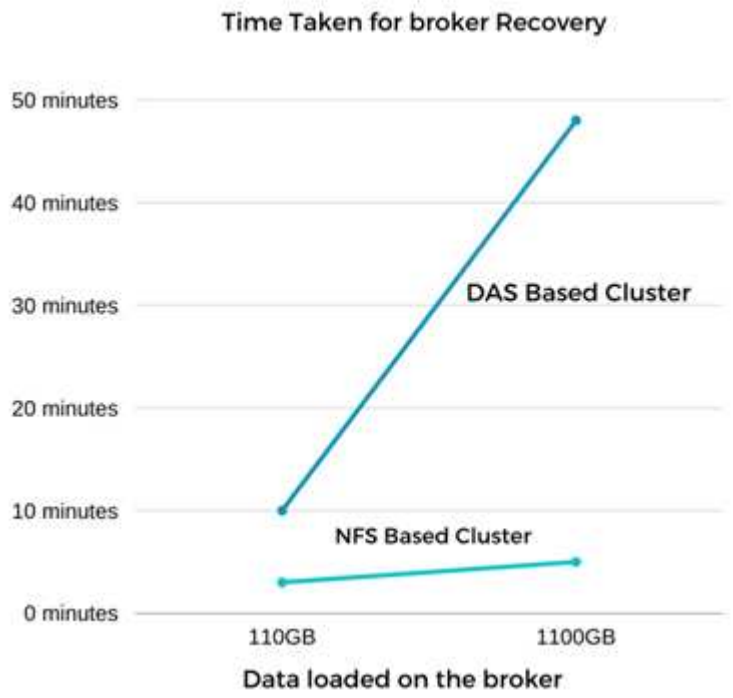
1. Der Backup-Knoten wurde um 09:39:17,213 gestartet. Der Startprotokolleintrag ist unten hervorgehoben.

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

2. Der Datenneuerstellungsvorgang endete um 09:42:29,115. Die Verarbeitung von 110 GB Daten dauerte ca. 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 bei Vermittlern mit etwa 1 TB Daten wiederholt, sodass für das etwa 48 Minuten und für NFS 3 Minuten in Anspruch genommen wurden. Die Ergebnisse sind im folgenden Diagramm dargestellt.



Storage-Effizienz

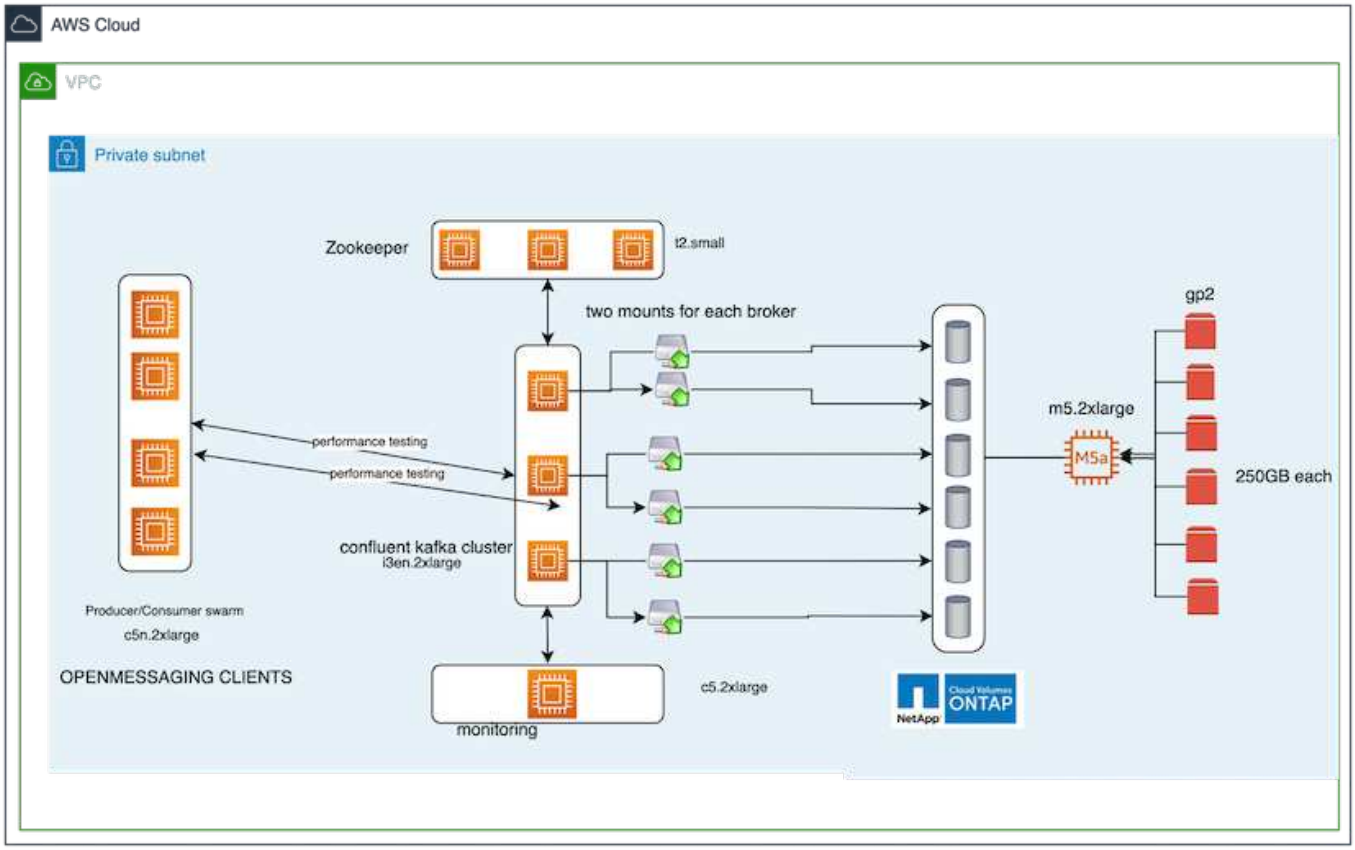
Da die Storage-Ebene des Kafka Clusters über NetApp ONTAP bereitgestellt wurde, verfügen wir über alle Storage-Effizienzfunktionen von ONTAP. Dazu wurde eine beträchtliche Datenmenge in einem Kafka-Cluster mit NFS-Storage, der auf Cloud Volumes ONTAP bereitgestellt wurde, erzeugt. Die ONTAP Funktionen ermöglichten eine deutliche Speicherplatzreduzierung.

Einrichtung der Architektur

In der folgenden Tabelle wird die Umgebungskonfiguration für ein Kafka-Cluster mithilfe von NAS gezeigt.

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

In der folgenden Abbildung ist die Architektur eines NAS-basierten Kafka-Clusters dargestellt.



- **Compute.** Wir benutzten einen drei-Knoten-Kafka-Cluster mit einem drei-Knoten-Zookeeper-Ensemble, das auf dedizierten Servern läuft. Jeder Broker hatte zwei NFS-Mount-Punkte zu einem einzelnen Volume auf der NetApp CVO-Instanz über eine dedizierte LIF.
- **Monitoring.** für eine Prometheus-Grafana-Kombination haben wir zwei Knoten verwendet. Zur Generierung von Workloads haben wir ein separates Cluster mit drei Nodes verwendet, das für diesen Kafka-Cluster erzeugt und genutzt werden kann.
- **Storage.** Wir verwendeten eine NetApp Cloud Volumes ONTAP-Instanz mit einem Node, auf der sechs 250 GB GP2 AWS-EBS Volumes gemountet wurden. Diese Volumes wurden dann über dedizierte LIFs dem Kafka-Cluster als sechs NFS-Volumes offengelegt.
- **Konfiguration.** die konfigurierbaren Elemente in diesem Testfall waren die Kafka-Broker.

Die Kompression wurde am Produktionsende ausgeschaltet, wodurch die Produzenten einen hohen Durchsatz

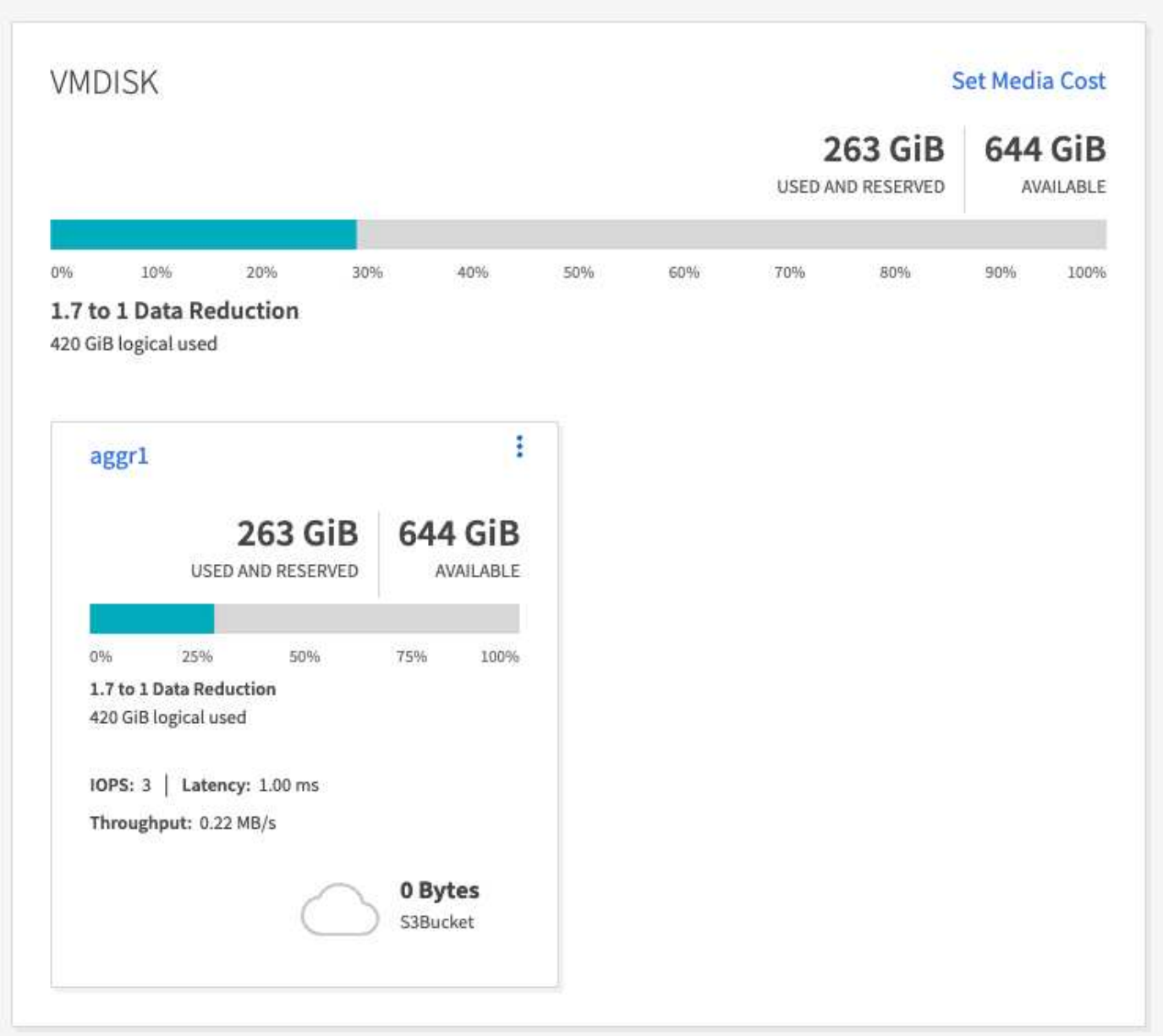
erzielen konnten. Die Storage-Effizienz wurde stattdessen von der Computing-Schicht abgefangen.

Methodik des Testens

1. Ein Kafka-Cluster wurde mit den oben genannten Spezifikationen bereitgestellt.
2. Auf dem Cluster wurden etwa 350 GB Daten mit dem OpenMessaging Benchmarking-Tool erzeugt.
3. Nach Abschluss des Workloads wurden mithilfe von ONTAP System Manager und der CLI die Statistiken zur Storage-Effizienz erfasst.

Beobachtung

Bei Daten, die mit dem OMB-Tool generiert wurden, erzielten wir eine Speicherersparnis von ~33 % bei einem Storage-Effizienzverhältnis von 1.70:1. Wie in den folgenden Abbildungen zu sehen, betrug der logische Speicherplatz der erzeugten Daten 420,3 GB und der physische Speicherplatz für die Daten 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
```

Performance-Übersicht und Validierung in AWS

Ein Kafka-Cluster mit einer auf NetApp NFS gemounteten Storage-Ebene wurde in der AWS Cloud hinsichtlich Performance getestet. Die Benchmarking-Beispiele sind in den folgenden Abschnitten beschrieben.

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

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

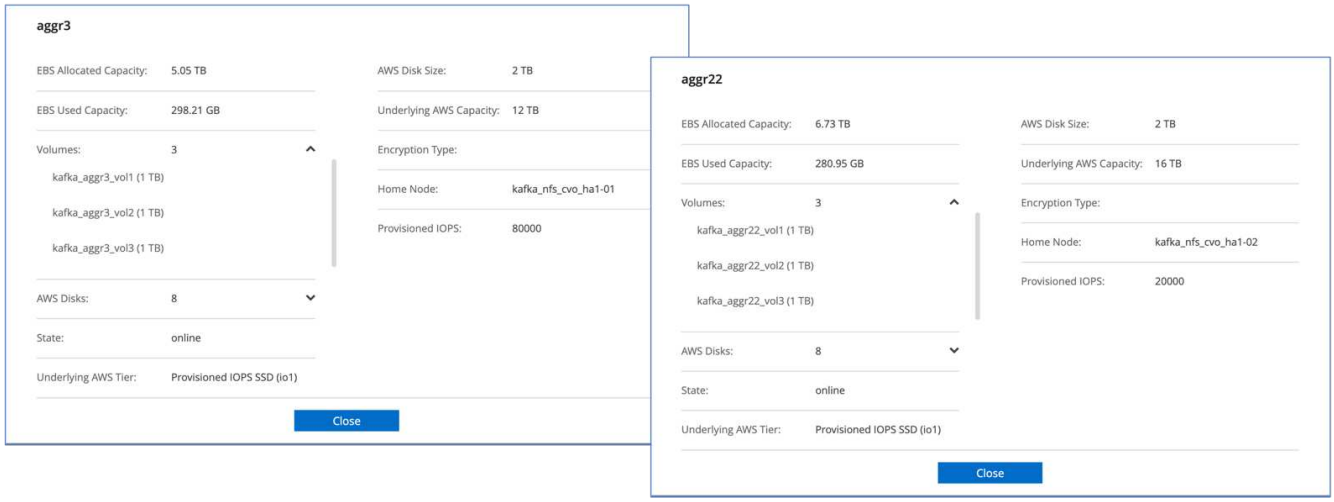
Einrichtung der Architektur

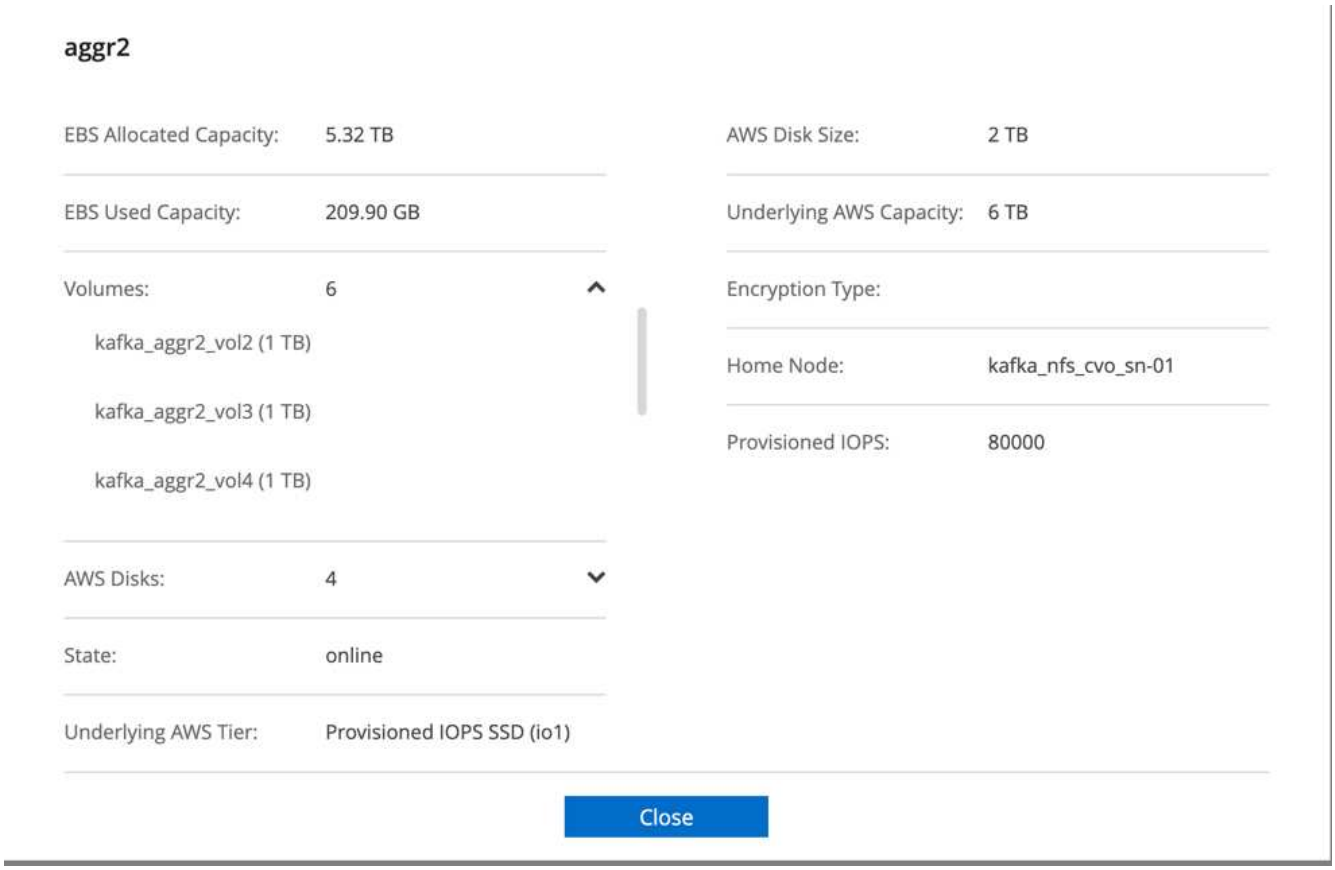
In der folgenden Tabelle wird die Umgebungskonfiguration für ein Kafka-Cluster mithilfe von NAS gezeigt.

Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3	• 3 x Zookeeper – t2.small • 3 x Broker Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Hersteller/Verbraucher — c5n.2xlarge *
Betriebssystem auf allen Knoten	RHEL8.6
NetApp Cloud Volumes ONTAP Instanz	INSTANZ EINES HA-Paars – m5dn.12xLarge x 2-Node-Instanz eines einzelnen Knotens – m5dn.12xLarge x 1 Node

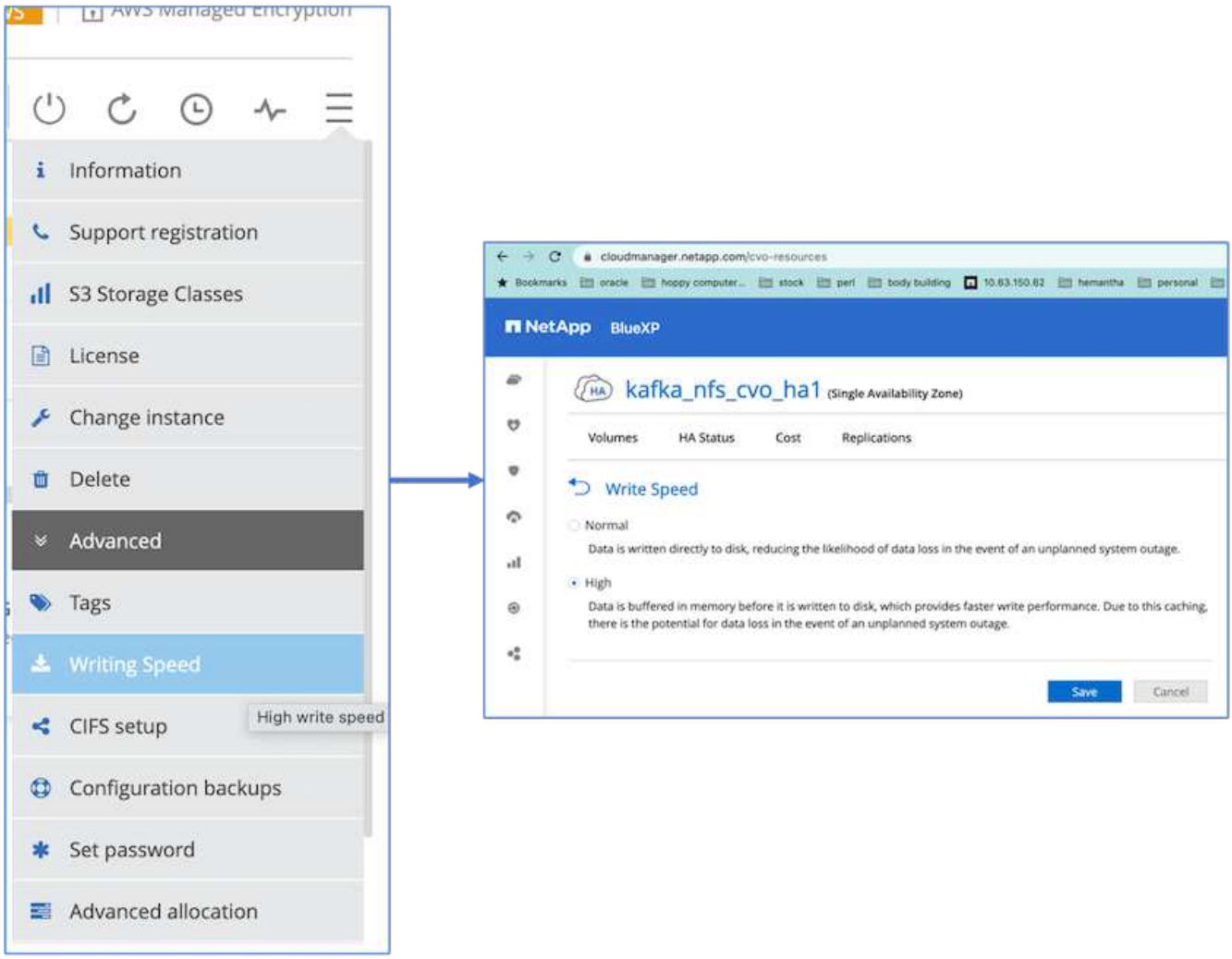
Einrichtung des NetApp Cluster Volume ONTAP

1. Für das Cloud Volumes ONTAP HA-Paar erstellten wir zwei Aggregate mit drei Volumes auf jedem Aggregat auf jedem Storage Controller. Für einen einzelnen Cloud Volumes ONTAP Node erstellen wir sechs Volumes in einem Aggregat.



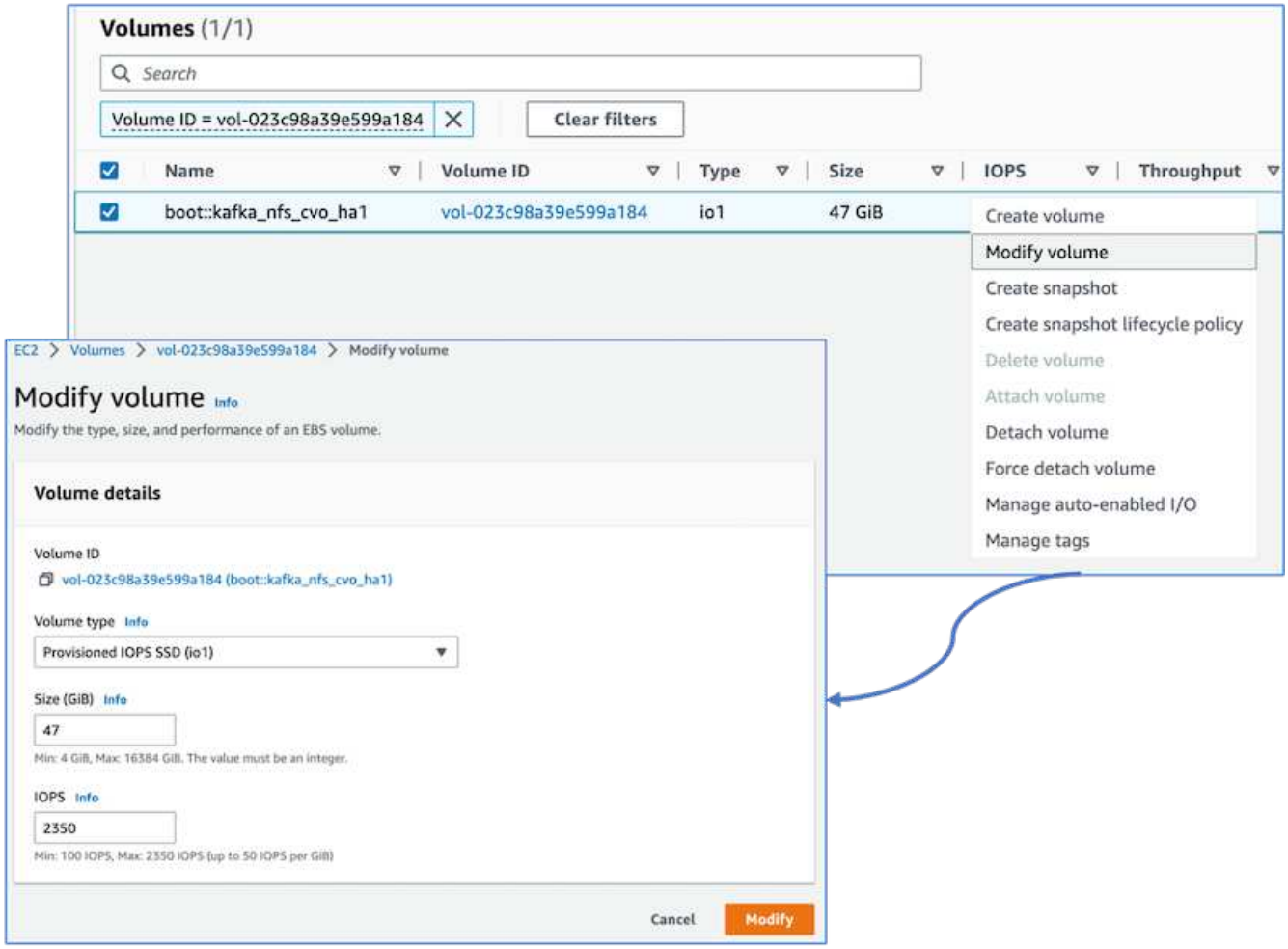


2. Um eine bessere Netzwerk-Performance zu erzielen, haben wir sowohl für das HA-Paar als auch für den einzelnen Node High-Speed-Networking ermöglicht.



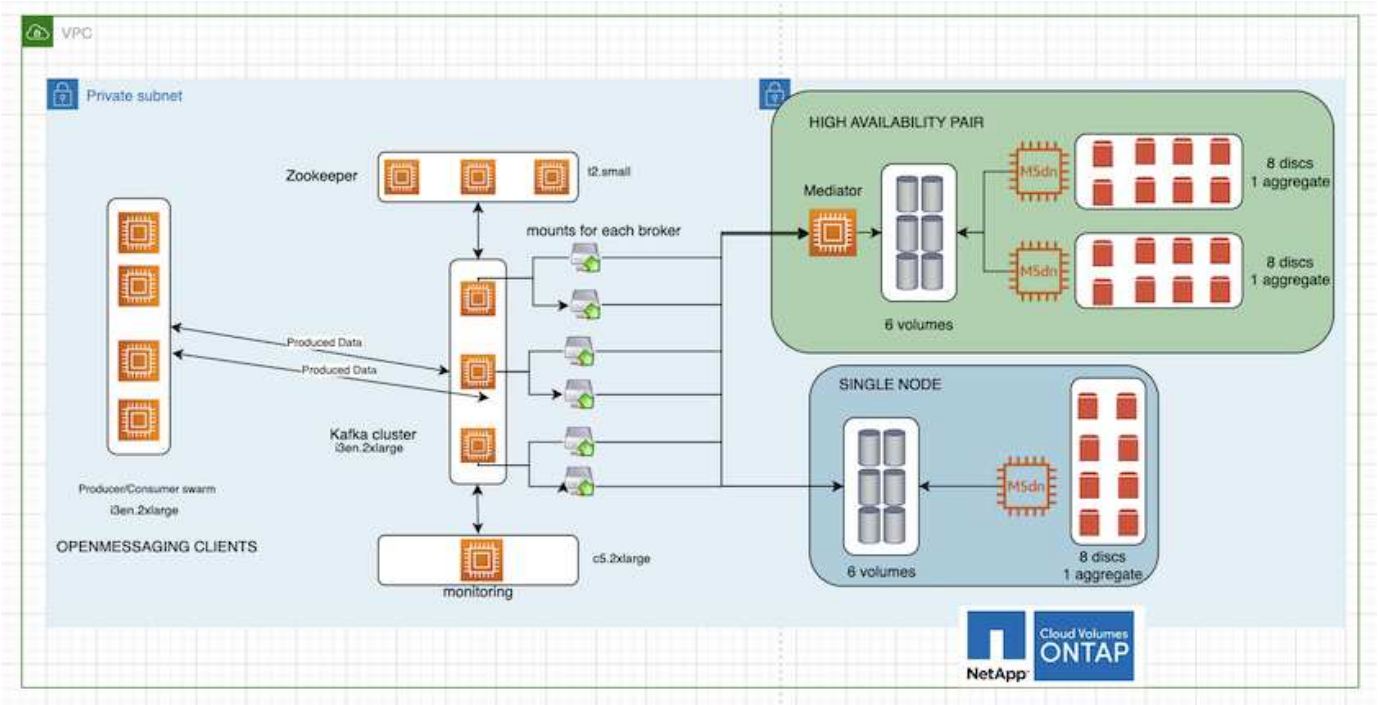
3. Wir bemerkten, dass der ONTAP NVRAM mehr IOPS hatte, also änderten wir die IOPS für das Cloud Volumes ONTAP Root-Volume auf 2350. Die Root-Datenträger in Cloud Volumes ONTAP waren 47 GB groß. Der folgende ONTAP-Befehl gilt für das HA-Paar und derselbe Schritt gilt für den einzelnen Node.

```
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::*>
```



In der folgenden Abbildung ist die Architektur eines NAS-basierten Kafka-Clusters dargestellt.

- **Compute.** Wir benutzten einen drei-Knoten-Kafka-Cluster mit einem drei-Knoten-Zookeeper-Ensemble, das auf dedizierten Servern läuft. Jeder Broker hatte zwei NFS-Mount-Punkte zu einem einzelnen Volume auf der Cloud Volumes ONTAP Instanz über eine dedizierte LIF.
- **Monitoring.** für eine Prometheus-Grafana-Kombination haben wir zwei Knoten verwendet. Zur Generierung von Workloads haben wir ein separates Cluster mit drei Nodes verwendet, das für diesen Kafka-Cluster erzeugt und genutzt werden kann.
- **Storage.** Wir verwendeten eine HA-Paar-Cloud Volumes ONTAP-Instanz mit einem 6 TB GP3 AWS-EBS Volume auf der Instanz gemountet. Das Volume wurde dann mit einem NFS-Mount in den Kafka-Broker exportiert.



OpenMessage Benchmarking-Konfigurationen

1. Für eine bessere NFS Performance benötigen wir mehr Netzwerkverbindungen zwischen dem NFS Server und dem NFS Client, die man mit nconnect erstellen kann. Mit dem folgenden Befehl mounten Sie die NFS-Volumes auf den Broker-Nodes mit der Option nconnect:

```
[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 über den einzelnen Cloud Volumes ONTAP Node 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 benutzen 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 restlichen Eigenschaften sind für Broker üblich. Für Broker1 ist die `log.dirs` Der Wert ist 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 ist 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 ist 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-Node ist die Kafka `servers.properties` Ist das gleiche wie für das Cloud Volumes ONTAP HA-Paar mit Ausnahme des `log.dirs` Eigenschaft.
   - Für Broker1 ist die `log.dirs` Der Wert ist 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 ist 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 ist 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. Der Workload 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
   consumerBbacklogSizeGB: 0
   testDurationMinutes: 5
   ```

   Der `messageSize` Kann je nach Anwendungsfall variieren. In unserem Performance-Test haben wir 3.000

verwendet.

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

- Die yaml-Datei, die für die Eigenschaften des Sync-Treibers verwendet wird, ist 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 yaml-Datei, die für die Eigenschaften des Durchsatztreibers verwendet wird, ist 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
```

Methodik des Testens

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

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

3. Ein anderer Workload wurde mit dem Durchsatztreiber mit derselben Workload-Konfiguration ausgelöst.

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

Beobachtung

Es wurden zwei unterschiedliche Treibertypen verwendet, mit denen Workloads für die Performance einer Kafka-Instanz generiert werden, die auf NFS ausgeführt wird. Der Unterschied zwischen den Treibern ist die Eigenschaft log flush.

Bei einem Cloud Volumes ONTAP HA-Paar:

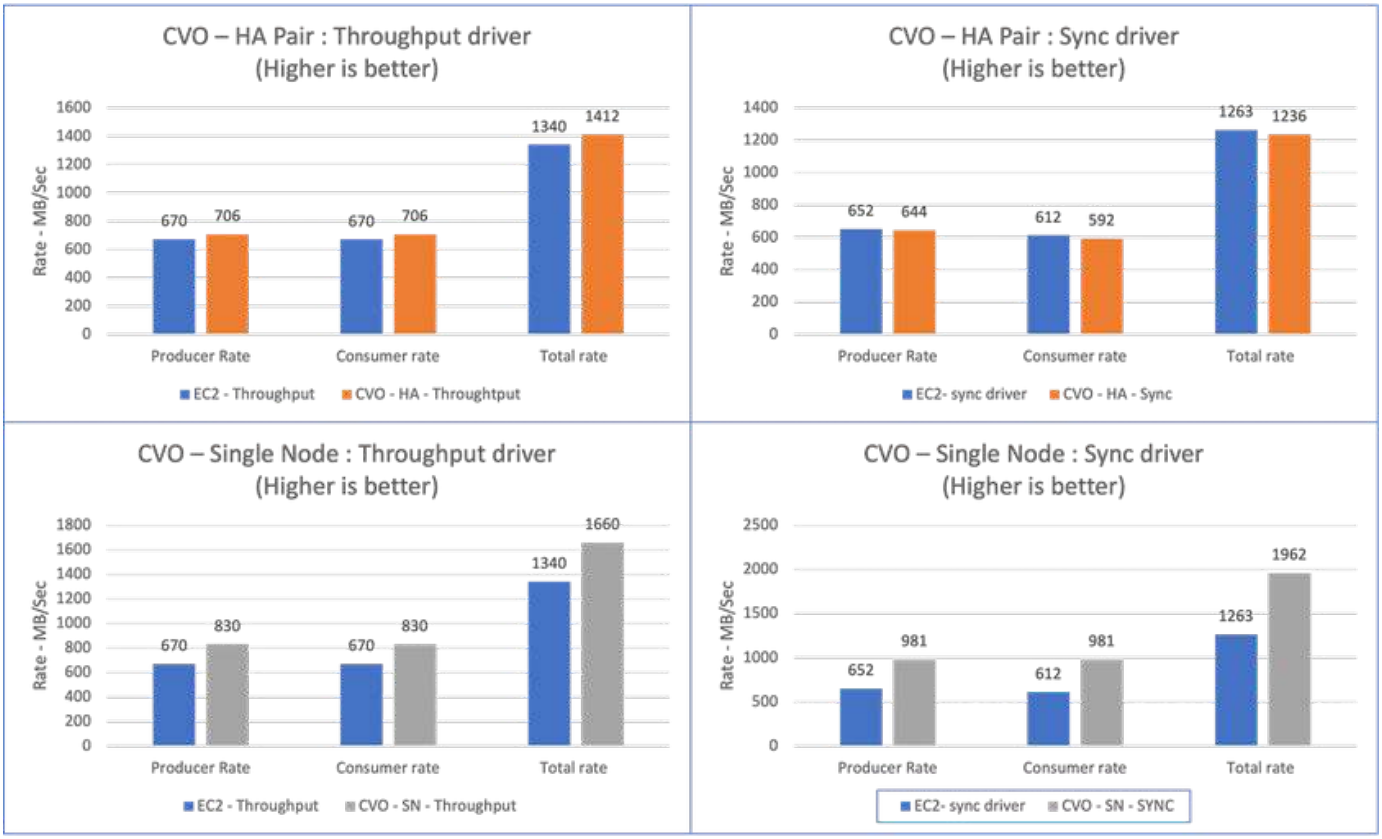
- Der Gesamtdurchsatz, der konsistent vom Sync-Treiber generiert wird: ~1236 Mbps.
- Gesamtdurchsatz für den Durchsatztreiber: Spitze ~1412 Mbps.

Für einen einzelnen Cloud Volumes ONTAP-Node:

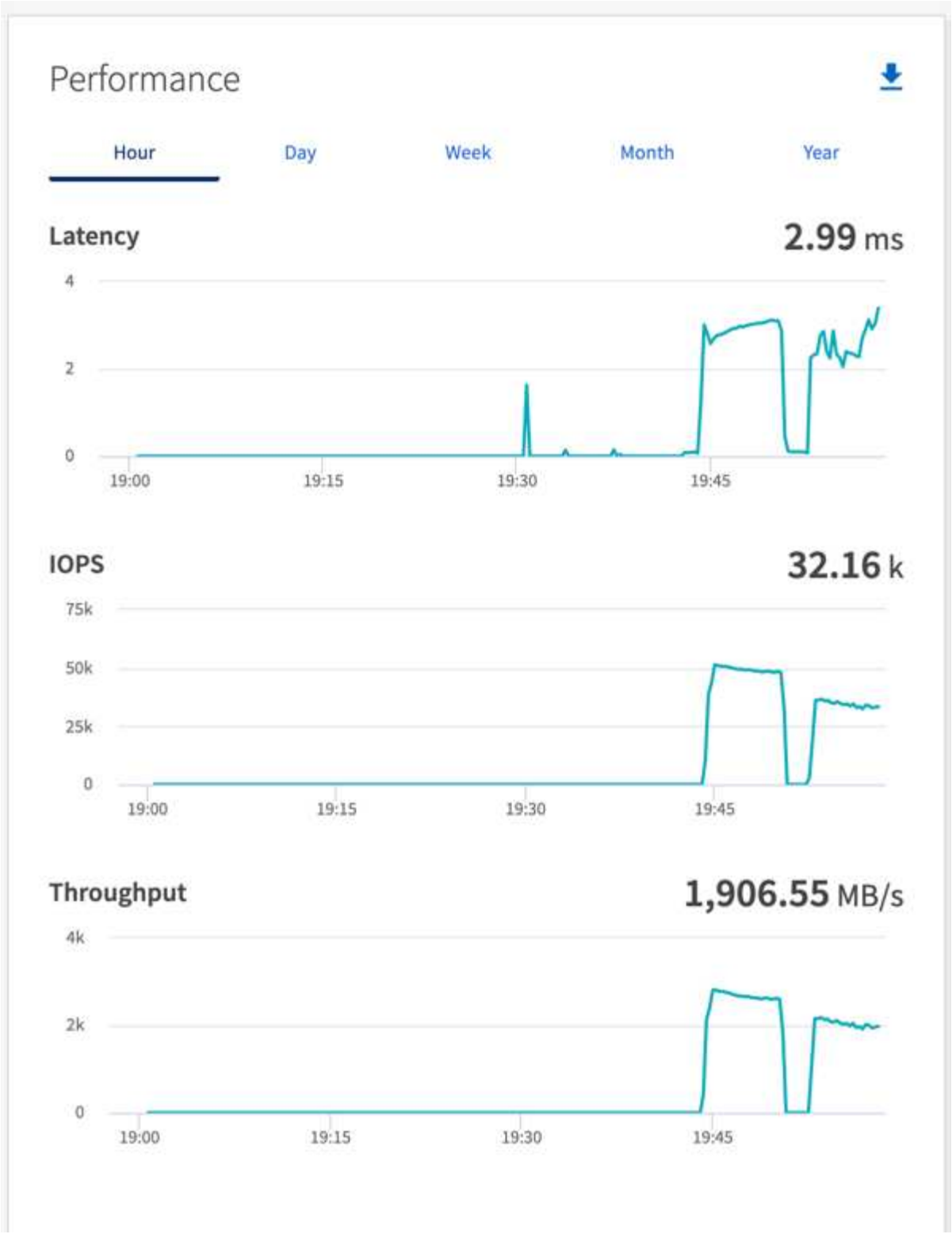
- Der Gesamtdurchsatz, der vom Sync-Treiber konsistent generiert wird: ~ 1962MBps.
- Gesamtdurchsatz des Durchsatztreibers: Spitze ~1660 MB/s

Der Sync-Treiber kann einen konsistenten Durchsatz generieren, da die Protokolle umgehend auf die Festplatte gespeichert werden, während der Durchsatztreiber bei der umfangreichen Protokollüberweise auf die Festplatte führt.

Diese Durchsatzwerte werden für die jeweilige AWS-Konfiguration generiert. Um höhere Performance-Anforderungen zu erfüllen, können die Instanztypen vertikal skaliert und weiter optimiert werden, um einen besseren Durchsatz zu erzielen. Der Gesamtdurchsatz oder die Gesamtrate ist die Kombination von Erzeugerrate und Verbraucherrate.



Prüfen Sie den Storage-Durchsatz, wenn Sie ein Benchmarking des Durchsatzes oder der Synchronisationstreiber durchführen.



Performance-Übersicht und Validierung in AWS FSX for NetApp ONTAP

Ein Kafka-Cluster, bei dem die Storage-Ebene auf NetApp NFS gemountet ist, wurde in AWS FSX for NetApp ONTAP in Bezug auf die Performance gemessen. Die Benchmarking-Beispiele sind in den folgenden Abschnitten beschrieben.

Apache Kafka in AWS FSX for NetApp ONTAP

Network File System (NFS) ist ein weit verbreitetes Netzwerkdateisystem zur Speicherung großer Datenmengen. In den meisten Unternehmen werden Daten zunehmend durch Streaming-Applikationen wie Apache Kafka generiert. Für diese Workloads sind Skalierbarkeit, niedrige Latenz und eine robuste Datenaufnahmearchitektur mit modernen Storage-Funktionen erforderlich. Für Echtzeitanalysen und zur Bereitstellung verwertbarer Erkenntnisse ist eine gut konzipierte und äußerst leistungsfähige Infrastruktur erforderlich.

Kafka funktioniert nach dem Design mit POSIX-konformem Filesystem und verarbeitet Dateivorgänge mithilfe des Filesystems. Beim Speichern von Daten auf einem NFSv3-Filesystem kann der Kafka-Broker NFS-Client Dateivorgänge jedoch anders als ein lokales Dateisystem wie XFS oder Ext4 interpretieren. Ein häufiges Beispiel ist die dumme NFS-Umbenennung, die Kafka-Broker fehlschlagen ließ, wenn sie Cluster erweitern und Partitionen neu zuordnen. Um dieser Herausforderung zu begegnen, hat NetApp den Open-Source-Linux-NFS-Client mit Änderungen aktualisiert, die nun allgemein in RHEL8.7, RHEL9.1 verfügbar sind und von der aktuellen FSX for NetApp ONTAP-Version, ONTAP 9.12.1, unterstützt werden.

Amazon FSX für NetApp ONTAP bietet ein vollständig gemanagtes, skalierbares und hochleistungsfähiges NFS-Dateisystem in der Cloud. Kafka-Daten auf FSX für NetApp ONTAP können für die Verarbeitung großer Datenmengen skaliert werden und Fehlertoleranz gewährleisten. NFS bietet zentralisiertes Storage-Management und Datensicherung für kritische und sensible Datensätze.

Durch diese Verbesserungen können AWS-Kunden die Vorteile von FSX for NetApp ONTAP nutzen, wenn sie Kafka-Workloads auf AWS-Computing-Services ausführen. Diese Vorteile sind:
* Verringerung der CPU-Auslastung zur Verringerung der I/O-Wartezeit
* Schnellere Recovery-Zeit für Kafka Broker.
* Zuverlässigkeit und Effizienz.
* Skalierbarkeit und Performance.
* Multi-Availability Zone Verfügbarkeit.
* Datenschutz.

Performance-Übersicht und Validierung in AWS FSX for NetApp ONTAP

Ein Kafka-Cluster mit einer auf NetApp NFS gemounteten Storage-Ebene wurde in der AWS Cloud hinsichtlich Performance getestet. Die Benchmarking-Beispiele sind in den folgenden Abschnitten beschrieben.

Kafka in AWS FSX for NetApp ONTAP

Ein Kafka-Cluster mit AWS FSX for NetApp ONTAP wurde hinsichtlich der Performance in der AWS-Cloud gemessen. Dieses Benchmarking wird in den folgenden Abschnitten beschrieben.

Einrichtung der Architektur

In der folgenden Tabelle wird die Umgebungskonfiguration für ein Kafka-Cluster mithilfe von AWS FSX für NetApp ONTAP gezeigt.

Plattformkomponente	Umgebungskonfiguration
Kafka 3.2.3	• 3 x Zookeeper – t2.small • 3 x Broker Server – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Hersteller/Verbraucher — c5n.2xlarge *

Plattformkomponente	Umgebungskonfiguration
Betriebssystem auf allen Knoten	RHEL8.6
AWS FSX für NetApp ONTAP	Multi-AZ mit 4 GB/s Durchsatz und 160000 IOPS

Einrichtung von NetApp FSX für NetApp ONTAP

1. Für unsere ersten Tests haben wir ein FSX für NetApp ONTAP-Dateisystem mit 2 TB Kapazität und 40000 IOPS für 2 GB/s Durchsatz 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 implementieren wir FSX for NetApp ONTAP über die AWS CLI. Sie müssen den Befehl in Ihrer Umgebung je nach Bedarf weiter anpassen. FSX for NetApp ONTAP kann zusätzlich über die AWS-Konsole implementiert und verwaltet werden. Dies erleichtert und optimiert die Implementierung mit weniger Befehlszeileneingaben.

   Dokumentation in FSX für NetApp ONTAP beträgt die für ein 2-GB/s-Durchsatzdateisystem in unserer Testregion (US-East-1) erreichbare maximale IOPS 80,000 iops. Der maximale IOPS-Wert für ein FSX für NetApp ONTAP-Dateisystem beträgt insgesamt 160,000 iops, wofür eine 4-GB/s-Durchsatzbereitstellung erforderlich ist, die wir später in diesem Dokument demonstrieren werden.

   Weitere Informationen zu FSX for NetApp ONTAP-Performance-Spezifikationen finden Sie in der Dokumentation zu AWS FSX for NetApp ONTAP hier: https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html .

   Detaillierte Kommandozeilen-Syntax 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 anstelle des standardmäßigen AWS FSX-Master-Schlüssels angeben, der verwendet wird, wenn kein KMS-Schlüssel angegeben wird.

2. Warten Sie beim Erstellen des FSX für NetApp ONTAP-Dateisystems, bis sich der „Lebenszyklus“-Status in Ihrer JSON-Rückkehr in „VERFÜGBAR“ 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. Validieren Sie die Zugangsdaten, indem Sie sich bei FSX for NetApp ONTAP SSH mit dem Benutzer fsxadmin anmelden:
   Fsxadmin ist das Standard-Administratorkonto für FSX für NetApp ONTAP-Dateisysteme bei der Erstellung. Das Passwort für fsxadmin ist das Passwort, das beim ersten Erstellen des Dateisystems entweder in der AWS-Konsole oder mit der AWS-CLI konfiguriert wurde, wie in Schritt 1 angegeben.

```
[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 Zugangsdaten validiert sind, erstellen Sie die Storage Virtual Machine im FSX for NetApp 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 File-Server mit eigenen administrativen Anmeldeinformationen und Endpunkten für die Administration und den Zugriff auf Daten in FSX für NetApp ONTAP Volumes und FSX für NetApp ONTAP-Mandantenfähigkeit.

5. Sobald Sie Ihre primäre Storage Virtual Machine konfiguriert haben, SSH in das neu erstellte FSX for NetApp ONTAP Dateisystem und erstellen Volumes in Storage Virtual Machine mit nachstehendem Beispielbefehl und ähnlich erstellen wir 6 Volumen für diese Validierung. Behalten Sie ausgehend von unserer Validierung die Standardkomponente (8) oder weniger Komponenten bei, was kafka eine bessere Performance bietet.

```
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 werden wir zusätzliche Kapazität in unseren Volumes benötigen. Erweitern Sie das Volume auf 2 TB und mounten Sie es über den 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 für NetApp ONTAP können Volumes über Thin Provisioning bereitgestellt werden. In unserem Beispiel übersteigt die gesamte erweiterte Volume-Kapazität die gesamte Dateisystemkapazität, sodass wir die gesamte Dateisystemkapazität erweitern müssen, um zusätzliche bereitgestellte Volume-Kapazität freizuschalten, die wir im nächsten Schritt demonstrieren werden.

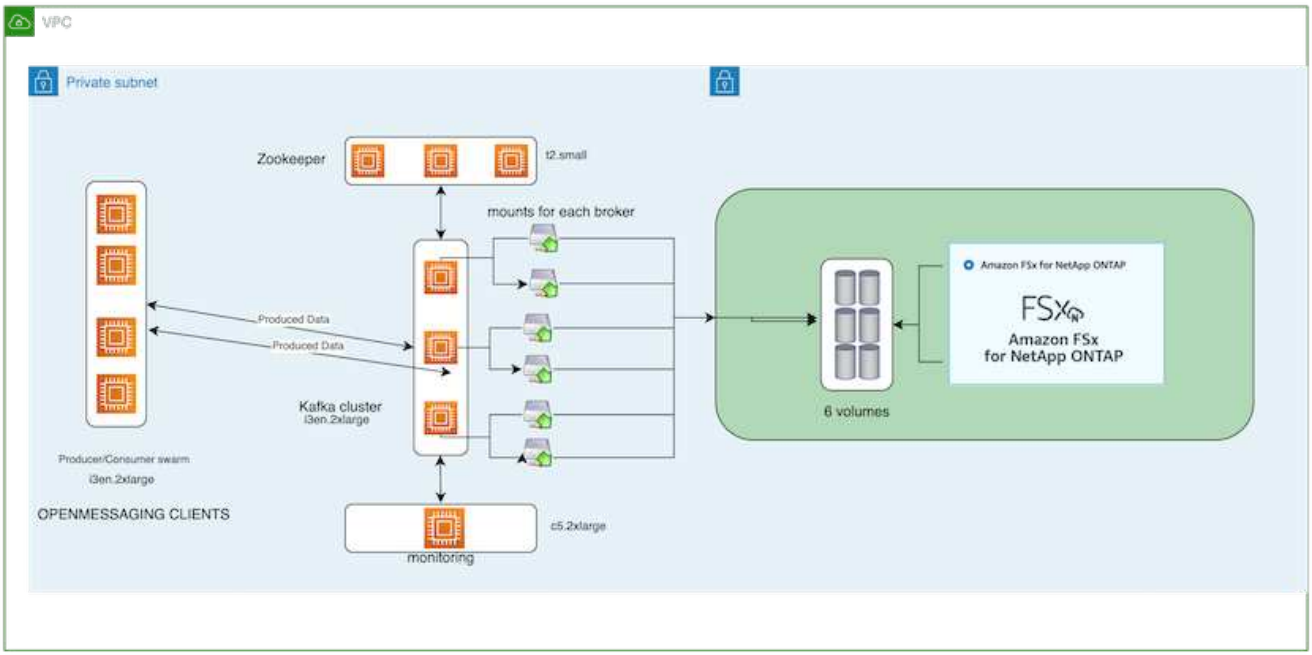
7. Anschließend erweitern wir die FSX for NetApp ONTAP-Durchsatzkapazität von 2 GB/s auf 4 GB/s und IOPS auf 160000 und die Kapazität auf 5 TB, um zusätzliche Performance und Kapazität zu erhalten

```
[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
```

Detaillierte Kommandozeilen-Syntax für FSX „Update-file-System“ finden Sie hier: https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html

8. Die FSX für NetApp ONTAP-Volumes sind mit nconnect und Standardopionen in Kafka-Brokern gemountet

Das folgende Bild zeigt unsere letzte Architektur unseres FSX für NetApp ONTAP-basierten Kafka-Clusters:



- Computing: Wir nutzten einen drei-Knoten-Kafka-Cluster mit einem drei-Knoten-Zookeeper-Ensemble, das auf dedizierten Servern lief. Jeder Broker hatte sechs NFS-Mount-Punkte auf sechs Volumes auf der FSX für NetApp ONTAP-Instanz.
- Monitoring: Wir haben zwei Knoten für eine Prometheus-Grafana Kombination verwendet. Zur Generierung von Workloads haben wir ein separates Cluster mit drei Nodes verwendet, das für diesen Kafka-Cluster erzeugt und genutzt werden kann.
- Storage: Wir verwendeten ein FSX für NetApp ONTAP, in dem sechs 2-TB-Volumes gemountet sind. Das Volume wurde dann mit einem NFS-Mount in den Kafka-Broker exportiert.die FSX für NetApp ONTAP-Volumes sind mit 16 nconnect-Sessions und Standardoptionen in Kafka-Broker gemountet.

OpenMessage Benchmarking-Konfigurationen.

Wir haben dieselbe Konfiguration verwendet wie für das NetApp Cloud Volumes ONTAP und ihre Details sind hier - https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

Methodik des Testens

1. Ein Kafka-Cluster wurde gemäß der oben beschriebenen Spezifikation mit Terraform und ansible bereitgestellt. Terraform wird verwendet, um die Infrastruktur mit AWS-Instanzen für den Kafka-Cluster zu erstellen, und ansible baut auf diesen den Kafka-Cluster.
2. Ein OMB-Workload wurde mit der oben beschriebenen Workload-Konfiguration und dem Sync-Treiber ausgelöst.

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

3. Ein anderer Workload wurde mit dem Durchsatztreiber mit derselben Workload-Konfiguration ausgelöst.

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

Beobachtung

Es wurden zwei unterschiedliche Treibertypen verwendet, mit denen Workloads für die Performance einer Kafka-Instanz generiert werden, die auf NFS ausgeführt wird. Der Unterschied zwischen den Treibern ist die Eigenschaft log flush.

Für einen Kafka-Replizierungsfaktor 1 und den FSX für NetApp ONTAP:

- Gesamtdurchsatz, der konsistent vom Sync-Treiber generiert wird: ~ 3218 Mbit/s und Spitzenleistung in ~ 3652 Mbit/s.
- Gesamtdurchsatz, der konsistent vom Durchsatztreiber generiert wird: ~ 3679 Mbit/s und Spitzenleistung in ~ 3908 Mbit/s.

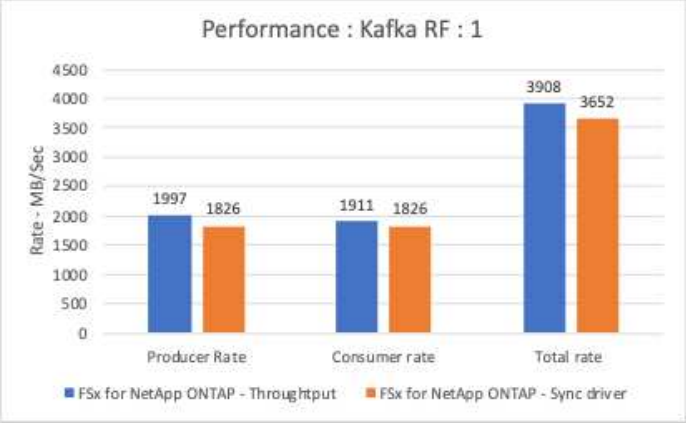
Für Kafka mit Replikationsfaktor 3 und dem FSX für NetApp ONTAP:

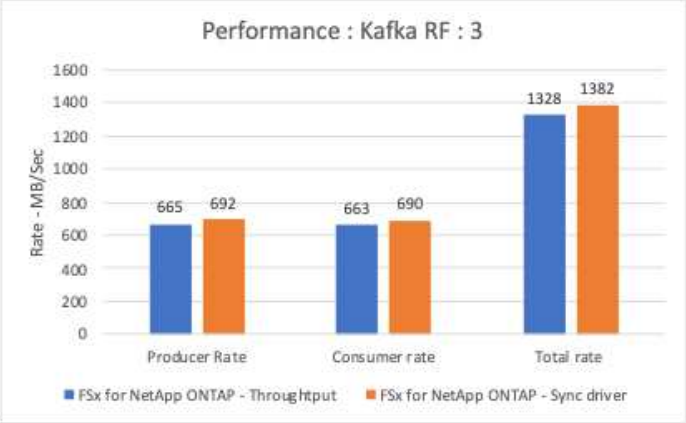
- Gesamtdurchsatz, der konsistent vom Sync-Treiber generiert wird: ~ 1252 Mbit/s und Spitzenleistung in ~ 1382 Mbit/s.
- Gesamtdurchsatz, der konsistent vom Durchsatztreiber generiert wird: ~ 1218 Mbit/s und Spitzenleistung in ~ 1328 Mbit/s.

In Kafka-Replizierungsfaktor 3 fand der Lese- und Schreibvorgang dreimal in FSX für NetApp ONTAP statt, in Kafka-Replizierungsfaktor 1 ist der Lese- und Schreibvorgang einmalig in FSX für NetApp ONTAP, so dass in beiden Validierungen, Wir konnten den maximalen Durchsatz von 4 GB/s erreichen.

Der Sync-Treiber kann einen konsistenten Durchsatz generieren, da die Protokolle umgehend auf die Festplatte gespeichert werden, während der Durchsatztreiber bei der umfangreichen Protokollüberweise auf die Festplatte führt.

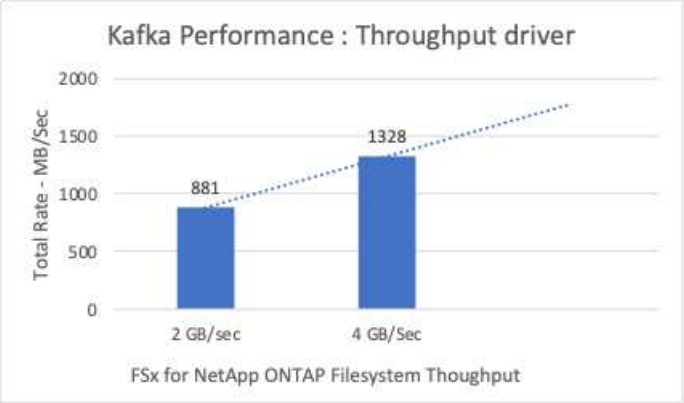
Diese Durchsatzwerte werden für die jeweilige AWS-Konfiguration generiert. Um höhere Performance-Anforderungen zu erfüllen, können die Instanztypen vertikal skaliert und weiter optimiert werden, um einen besseren Durchsatz zu erzielen. Der Gesamtdurchsatz oder die Gesamtrate ist die Kombination von Erzeugerrate und Verbraucherrate.

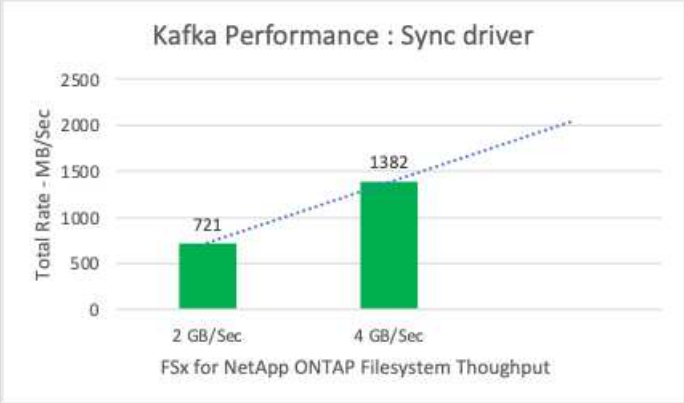




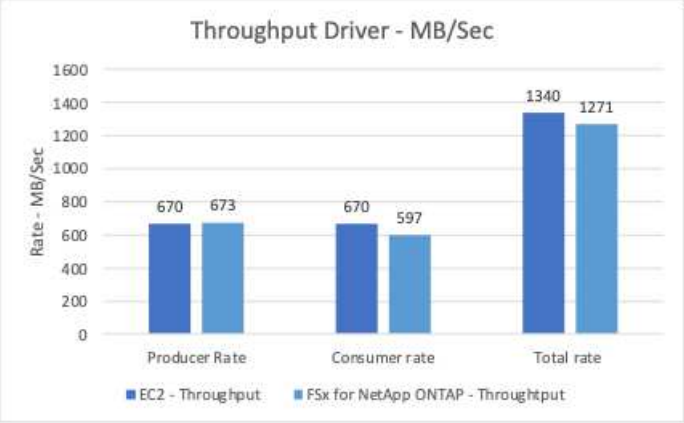
Das folgende Diagramm zeigt den 2-GB/s-FSX für NetApp ONTAP und 4-GB/s-Performance für Kafka-Replizierungsfaktor 3. Der Replizierungsfaktor 3 führt den Lese- und Schreibvorgang dreimal auf dem FSX für

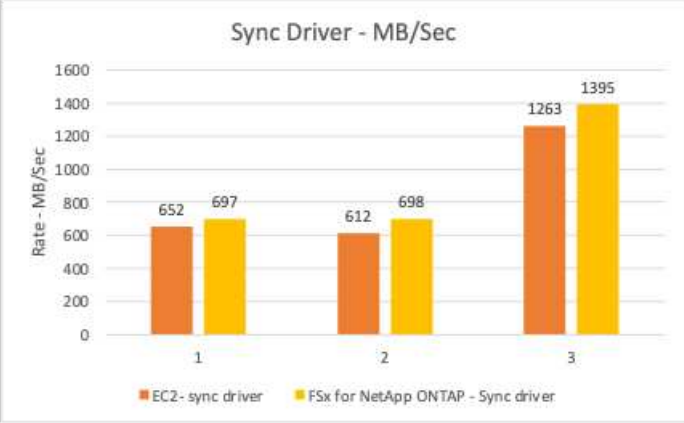
NetApp ONTAP-Storage durch. Die Gesamtrate für den Durchsatztreiber beträgt 881 MB/sec, was den Kafka-Betrieb ungefähr 2.64 GB/sec auf dem 2GB/sec FSX für NetApp ONTAP-Dateisystem liest und schreibt. Die Gesamtrate für den Durchsatztreiber beträgt 1328 MB/sec, was den kafka-Betrieb ungefähr 3.98 GB/sec liest und schreibt. Unsere Kafka-Performance ist linear und skalierbar basierend auf dem FSX for NetApp ONTAP Throughput.





Das folgende Diagramm zeigt die Performance zwischen EC2-Instanz und FSX für NetApp ONTAP (Kafka-Replizierungsfaktor: 3)

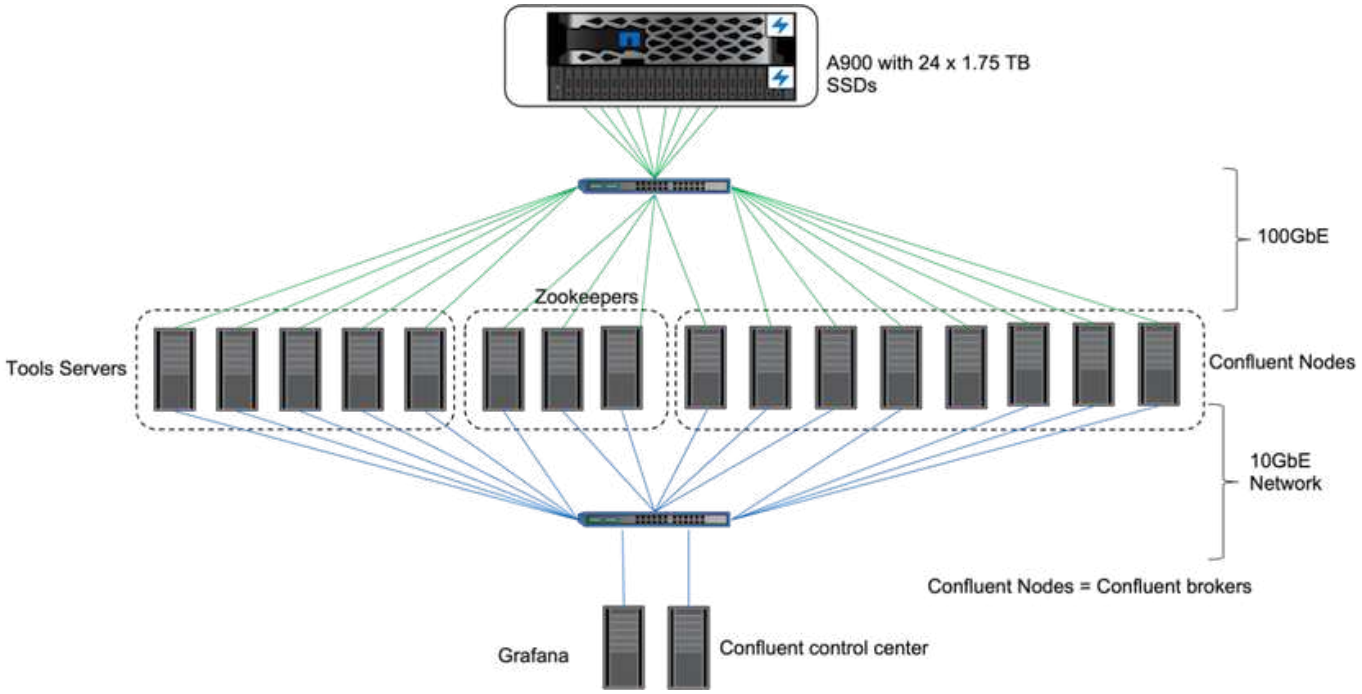




Performance-Übersicht und Validierung mit AFF A900 vor Ort

Vor Ort haben wir den NetApp AFF A900 Storage-Controller mit ONTAP 9.12.1RC1 verwendet, um die Performance und Skalierung eines Kafka Clusters zu validieren. Wir verwendeten dieselbe Testumgebung wie in unseren früheren Best Practices für Tiered Storage mit ONTAP und AFF.

Wir haben Confluent Kafka 6.2.0 verwendet, um die AFF A900 zu evaluieren. Der Cluster verfügt über acht Broker-Nodes und drei Zookeeper-Nodes. Für Performance-Tests haben wir fünf OMB-Arbeitsknoten verwendet.



Storage-Konfiguration

Wir verwendeten NetApp FlexGroups Instanzen, um einen einzelnen Namespace für Protokollverzeichnisse bereitzustellen und so das Recovery und die Konfiguration zu vereinfachen. Wir verwendeten NFSv4.1 und pNFS, um direkten Pfadzugriff auf Protokollsegmentdaten 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 haben wir die `max_session_slots` Aus der Standardeinstellung `64` Bis `180`. Dies entspricht dem Standardlimit für Sitzungsslot in ONTAP.

Kafka-Broker-Tuning

Um den Durchsatz im zu testenden System zu maximieren, haben wir die Standardparameter für bestimmte Schlüsselthread-Pools deutlich erhöht. Für die meisten Konfigurationen empfehlen wir die folgenden Confluent Kafka Best Practices. Dieses Tuning wurde verwendet, um die Parallelität von ausstehender I/O zum Storage zu maximieren. Diese Parameter können an die Computing-Ressourcen und Storage-Attribute Ihres Vermittlers angepasst werden.

```
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 Generator

Für den Durchsatztreiber und die Themenkonfiguration haben wir 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 der ONTAP SVM aktiviert.

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

3. Der Workload wurde mit dem Durchsatztreiber unter Verwendung derselben Workload-Konfiguration wie für Cloud Volumes ONTAP ausgelöst. Siehe Abschnitt „Stabile Performance„ Unten. Für den Workload wurde ein Replizierungsfaktor von 3 verwendet, d. h., es wurden drei Kopien von Protokollsegmenten in NFS aufbewahrt.

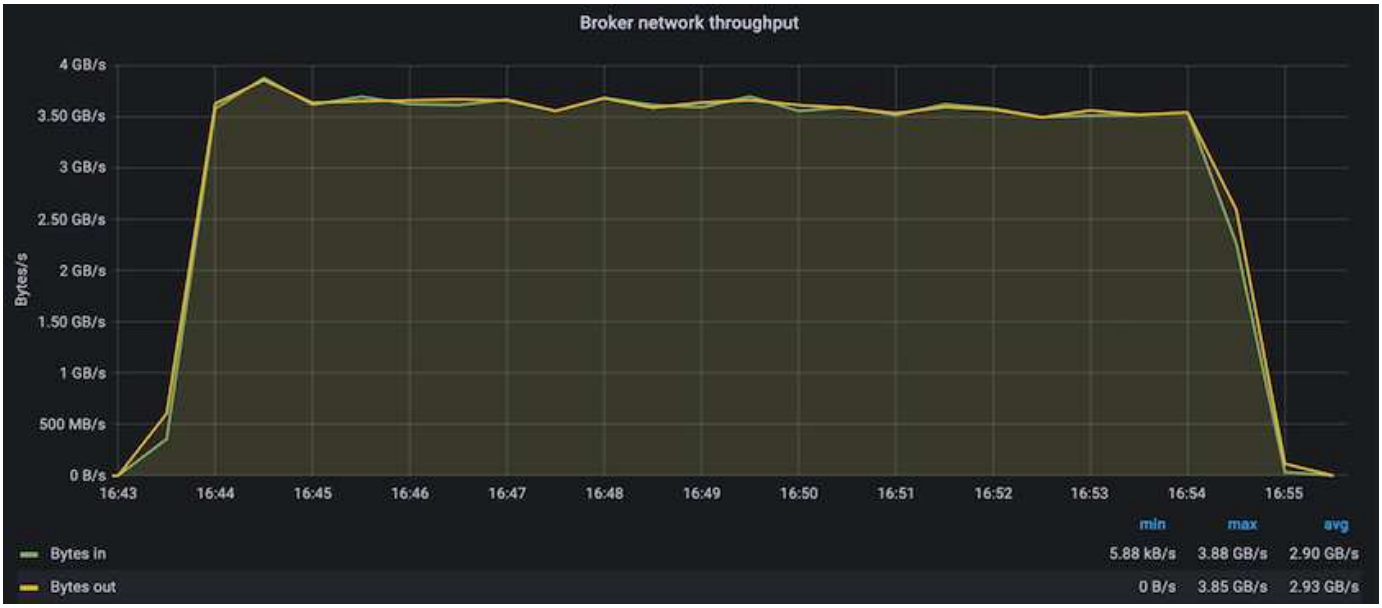
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. Schließlich haben wir Messungen mit einem Rückstand durchgeführt, um die Fähigkeit der Verbraucher zu messen, die neuesten Nachrichten zu erhalten. OMB baut einen Rückstand auf, indem die Verbraucher zu Beginn einer Messung angehalten werden. Dies führt zu drei unterschiedlichen Phasen: Der Schaffung von Rückstand (Traffic nur für Hersteller), der Entleerung von Rückständen (eine verbraucherlastige Phase, in der Verbraucher verpasste Ereignisse in einem Thema aufholen) und der stetige Zustand. Siehe Abschnitt „Speichergrenzen werden untersucht„ Weitere Informationen.

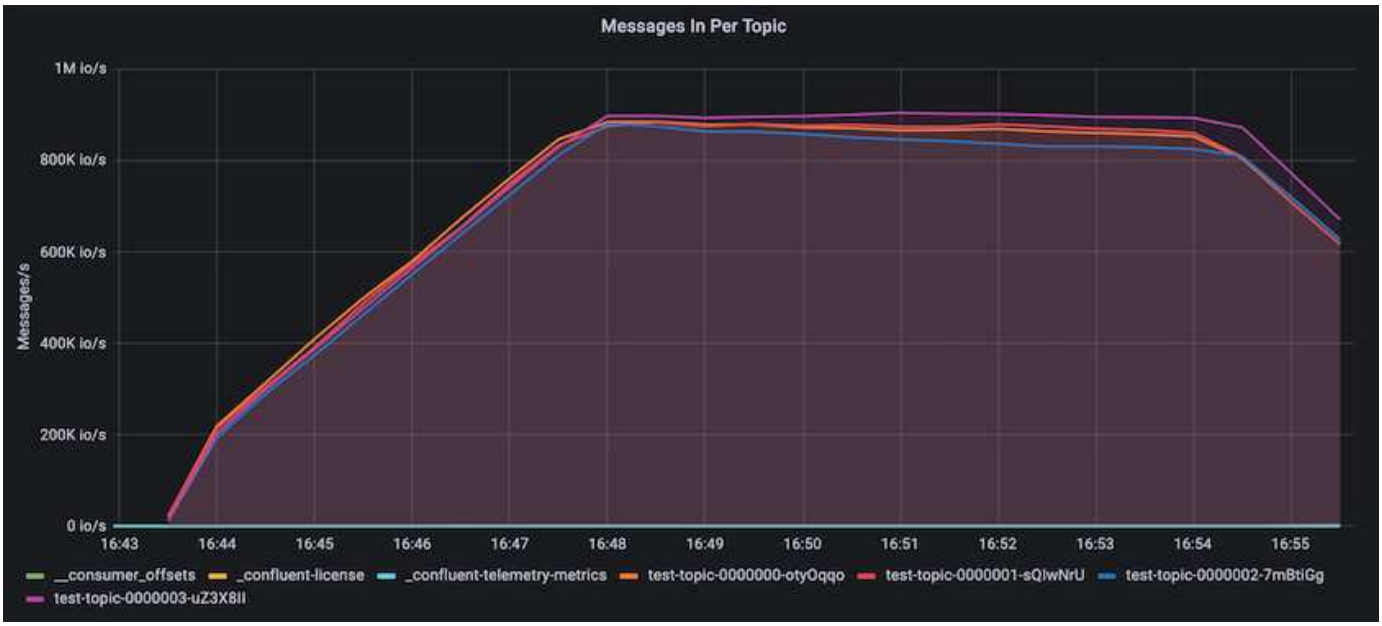
Stabile Performance

Wir haben den AFF A900 anhand des OpenMessaging Benchmarks evaluiert, um einen ähnlichen Vergleich wie für Cloud Volumes ONTAP in AWS und das in AWS zu liefern. Alle Performance-Werte stellen den Kafka-Cluster-Durchsatz auf Produzenten- und Verbraucherebene dar.

Die stabile Performance mit Confluent Kafka und dem AFF A900 erreichte einen durchschnittlichen Durchsatz von 3,4 GB/s sowohl bei Herstellern als auch bei Verbrauchern. Das sind über 3.4 Millionen Nachrichten im Kafka-Cluster. Durch die Visualisierung des kontinuierlichen Durchsatzes in Byte pro Sekunde für BrokerTopicMetrics sehen wir die hervorragende Steady State Performance und den vom AFF A900 unterstützten Datenverkehr.



Dies entspricht gut der Ansicht der Nachrichten, die pro Thema übermittelt werden. Die folgende Grafik zeigt eine Aufschlüsselung pro Thema. Bei der getesteten Konfiguration sahen wir fast 900.000 Nachrichten pro Thema für vier Themen.

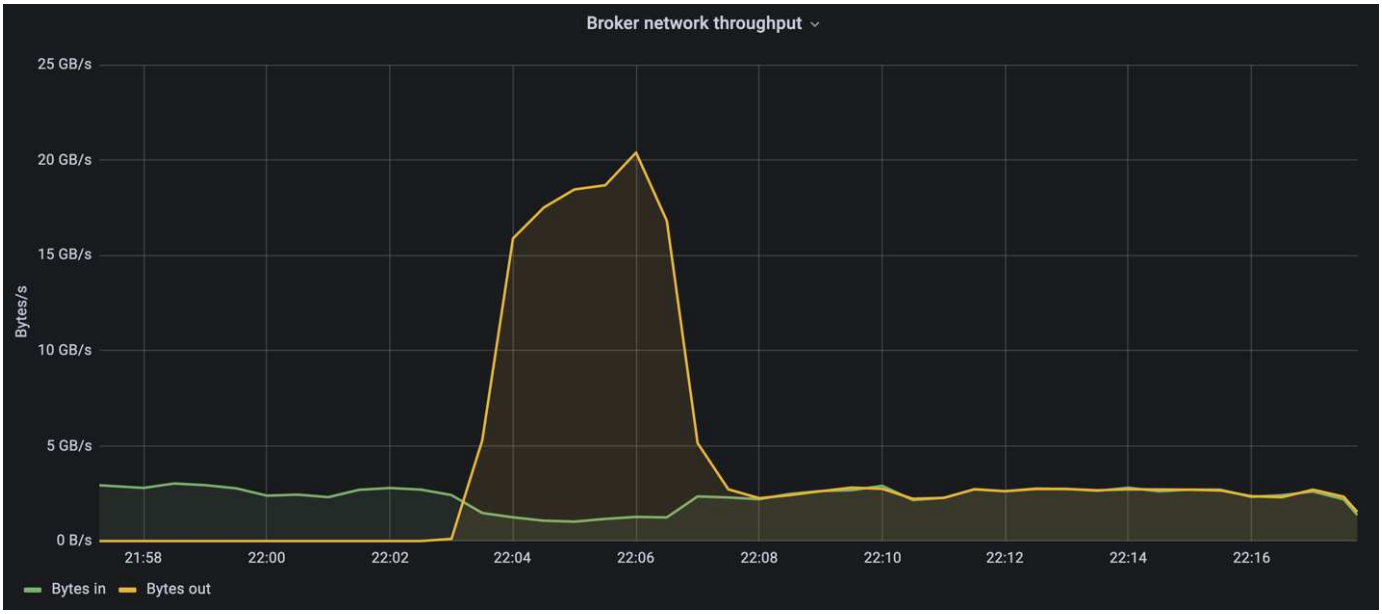


Extreme Performance und Speichergrenzen untersucht werden

Für AFF haben wir auch mit OMB unter Verwendung der Backlog-Funktion getestet. Die Backlog-Funktion hält Verbraucher-Abonnements an, während im Kafka-Cluster ein Rückstand von Ereignissen aufgebaut wird. In dieser Phase tritt nur der Produzentenverkehr auf, der Ereignisse generiert, die in die Protokolle übertragen werden. Dies emuliert die Batch-Verarbeitung oder die Offline-Analyse-Workflows am genauesten. In diesen Workflows werden Kundenabonnements gestartet und müssen historische Daten lesen, die bereits aus dem Broker-Cache entfernt wurden.

Um die Storage-Einschränkungen für den Verbraucherdurchsatz in dieser Konfiguration zu verstehen, haben wir die reine Produzentenphase gemessen, um zu verstehen, wie viel Schreibverkehr das A900 aufnehmen könnte. Siehe den nächsten Abschnitt „Anleitung zur GrößenbemessungUm zu verstehen, wie man diese Daten nutzt.

Während des reinen Produzententeils dieser Messung konnten wir einen hohen Spitzendurchsatz beobachten, der die Grenzen der A900-Leistung überstieg (wenn andere Broker-Ressourcen nicht für den Produzenten- und Verbraucherverkehr gesättigt waren).



Wir haben die Nachrichtengröße für diese Messung auf 16.000 erhöht, um den Overhead pro Nachricht zu begrenzen und den Storage-Durchsatz auf NFS-Bereitstellungspunkte zu maximieren.

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

Der Confluent Kafka Cluster erzielte einen Spitzendurchsatz von 4.03GB/s.

```
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 den Eventstau ausgefüllt hat, wurde der Consumer Traffic neu gestartet. Bei Messungen mit einer Entleerung des Rückstands konnten wir einen Spitzendurchsatz von über 20 GB/s bei allen Themen beobachten. Der kombinierte Durchsatz zum NFS-Volume, auf dem die OMB-Protokolldaten gespeichert werden, wurde auf ~30 GBit/s gesteigert.

Anleitung zur Größenbemessung

Amazon Web Services bietet eine "Leitfaden zur Größenanpassung" Ideal zum Skalieren und Skalieren von Kafka Clustern.

Diese Größenbestimmung bietet eine nützliche Formel zum Bestimmen der Anforderungen an den Storage-Durchsatz für Ihren Kafka-Cluster:

Bei einem aggregierten Durchsatz, der mit einem Replizierungsfaktor r in den Cluster von tcluster erzeugt wird, beträgt der vom Broker Storage erhaltene Durchsatz wie folgt:

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

Das lässt sich noch weiter vereinfachen:

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

Mit dieser Formel können Sie die entsprechende ONTAP-Plattform für die Anforderungen Ihres Kafka-Hot-Tier auswählen.

In der folgenden Tabelle wird der erwartete Producer Throughput für den A900 mit unterschiedlichen Replikationsfaktoren erläutert:

Replizierungsfaktor	Producer Throughput (GPPS)
3 (gemessen)	3.4
2	5.1

Replizierungsfaktor	Producer Throughput (GPPS)
1	10.2

Schlussfolgerung

Die NetApp Lösung für das „albern“-Problem bietet eine einfache, kostengünstige und zentral gemanagte Storage-Form für Workloads, die zuvor mit NFS nicht kompatibel waren.

Mit diesem neuen Paradigma erstellen Sie leichter managebare Kafka-Cluster, die sich einfacher migrieren und für Disaster Recovery und Datensicherung spiegeln lassen.
Zudem konnten wir feststellen, dass NFS zusätzliche Vorteile bietet, wie z. B. geringere CPU-Auslastung, schnellere Recovery-Zeiten, deutlich höhere Storage-Effizienz und eine bessere Performance durch NetApp ONTAP.

Wo Sie weitere Informationen finden

Sehen Sie sich die folgenden Dokumente und/oder Websites an, um mehr über die in diesem Dokument beschriebenen Informationen zu erfahren:

- Was ist Apache Kafka?

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

- Was ist 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/"

- Dumm- Benennen Sie Problem mit Kafka.

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

- NetApp Produktdokumentation

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

- Was ist NFS?

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

- Was ist die Kafka-Partitionsumverteilung?

  "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 migrieren Sie 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/

- Managed Platform für Apache Kafka

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

- Unterstützung für Apache Kafka

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

- Beratungsdienstleistungen für Apache Kafka

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