Apache Kafka workloads with NetApp NFS storage

NetApp Solutions

NetApp
May 31, 2023

Table of Contents

Apache Kafka workloads with NetApp NFS storage

TR-4947: Apache Kafka workload with NetApp NFS storage - Functional validation and performance

Shantanu Chakole, Karthikeyan Nagalingam, and Joe Scott, NetApp

Kafka is a distributed publish-subscribe messaging system with a robust queue that can accept large amounts of message data. With Kafka, applications can write and read data to topics in a very fast manner. Because of its fault tolerance and scalability, Kafka is often used in the big data space as a reliable way to ingest and move many data streams very quickly. Use cases include stream processing, website-activity tracking, metrics collection and monitoring, log aggregation, real time analytics, and so on.

Although normal Kafka operations on NFS work well, the silly rename issue crashes the application during the resizing or repartitioning of a Kafka cluster running on NFS. This is a significant issue because a Kafka cluster must be resized or repartitioned for load-balancing or maintenance purposes. You can find additional details here.

This document describes the following subjects:

- The silly-rename problem and solution validation
- Reducing CPU utilization to reduce the I/O wait time
- Faster Kafka broker recovery time
- Performance in the cloud and on-premises

Why use NFS storage for Kafka workloads?

Kafka workloads in production applications can stream huge amounts of data between applications. This data is held and stored in the Kafka broker nodes in the Kafka cluster. Kafka is also known for availability and parallelism, which it achieves by breaking topics into partitions and then replicating those partitions throughout the cluster. This eventually means that the huge amount of data that flows through a Kafka cluster is generally multiplied in size. NFS makes rebalancing data as the number of brokers changes very quick and easy. For large environments, rebalancing data across DAS when the number of brokers changes is very time consuming, and, in most Kafka environments, the number of brokers changes frequently.

Other benefits include the following:

- **Maturity.** NFS is a mature protocol, which means most aspects of implementing, securing, and using it are well understood.
- **Open.** NFS is an open protocol, and its continued development is documented in internet specifications as a free and open network protocol.
- **Cost-effective.** NFS is a low-cost solution for network file sharing that is easy to set up because it uses the existing network infrastructure.
- **Centrally managed.** Centralized management of NFS decreases the need for added software and disk space on individual user systems.

- **Distributed.** NFS can be used as a distributed file system, reducing the need for removable media storage devices.

Why NetApp for Kafka workloads?

The NetApp NFS implementation is considered a gold standard for the protocol and is used in countless enterprise NAS environments. In addition to the credibility of NetApp, it also offers the following benefits:

- Reliability and efficiency
- Scalability and performance
- High availability (HA partner in a NetApp ONTAP cluster)
- Data protection
  - **Disaster recovery (NetApp SnapMirror).** Your site goes down or you want to jump start at a different site and continue from where you left off.
  - Manageability of your storage system (administration and management using NetApp OnCommand).
  - **Load balancing.** The cluster allows you to access different volumes from data LIFs hosted on different nodes.
  - **Nondisruptive operations.** LIFs or volume moves are transparent to the NFS clients.

Next: NetApp solution for silly rename issue for NFS to Kafka workloads.

NetApp solution for silly rename issue for NFS to Kafka workloads

Previous: Introduction.

Kafka is built with the assumption that the underlying filesystem is POSIX compliant: for example, XFS or Ext4. Kafka resource rebalancing removes files while the application is still using them. A POSIX-compliant file system allows unlink to proceed. However, it only removes the file after all references to the file are gone. If the underlying filesystem is network attached, then the NFS client intercepts the unlink calls and manages the workflow. Because there are pending opens on the file being unlinked, the NFS client sends a rename request to the NFS server and, on the last close of the unlinked file, issues a remove operation on the renamed file. This behavior is commonly referred to as NFS silly rename, and it is orchestrated by the NFS client.

Any Kafka broker using storage from an NFSv3 server runs into issues because of this behavior. However, the NFSv4.x protocol has features to address this issue by allowing the server to take responsibility for the opened, unlinked files. NFS servers supporting this optional feature communicate the ownership capability to the NFS client at the time of file opening. The NFS client then ceases the unlink management when there are opens pending and allows the server to manage the flow. Although the NFSv4 specification provides guidelines for implementation, until now, there were not any known NFS server implementations that supported this optional feature.

The following changes are required for the NFS server and the NFS client to address the silly rename issue:

- **Changes to NFS client (Linux).** At the time of file opening, the NFS server responds with a flag, indicating the capability to handle the unlinking of opened files. NFS client-side changes allow the NFS server to

handle the unlinking in the presence of the flag. NetApp has updated the open-source Linux NFS client with these changes. The updated NFS client is now generally available in RHEL8.7 and RHEL9.1.

- **Changes to NFS server.** The NFS server keeps track of opens. Unlinking on an existing open file is now managed by the server to match POSIX semantics. When the last open is closed, The NFS server then initiates the actual removal of the file and thus avoids the silly rename process. The ONTAP NFS server has implemented this capability in its latest release, ONTAP 9.12.1.

With the above changes to the NFS client and server, Kafka can safely reap all the benefits of network-attached NFS storage.

Next: Functional validation - Silly rename fix.

Functional validation - Silly rename fix

Previous: NetApp solution for silly rename issue in NFS to Kafka workload.

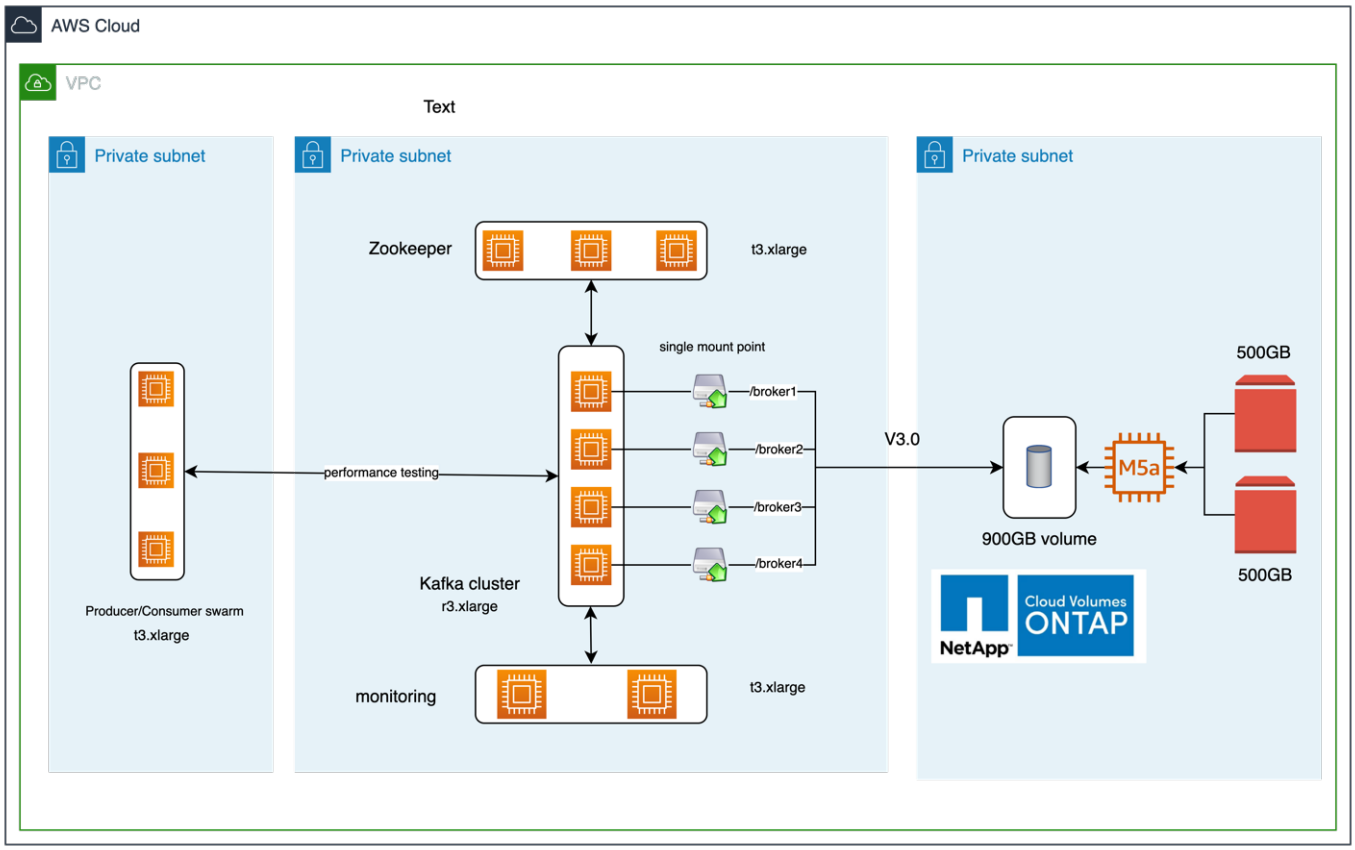
For the functional validation, we showed that a Kafka cluster with an NFSv3 mount for storage fails to perform Kafka operations like partition redistribution, whereas another cluster mounted on NFSv4 with the fix can perform the same operations without any disruptions.

Validation setup

The setup is run on AWS. The following table shows the different platform components and environmental configuration used for the validation.

Platform component	Environment configuration
Confluent Platform version 7.2.1	• 3 x zookeepers – t3.xlarge • 4 x broker servers – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x control center – t3.xlarge • 3 x Producer/consumer
Operating system on all nodes	RHEL8.7or later
NetApp Cloud Volumes ONTAP instance	Single-node instance – M5.2xLarge

The following figure show the architectural configuration for this solution.



Architectural flow

- **Compute.** We used a four-node Kafka cluster with a three-node zookeeper ensemble running on dedicated servers.
- **Monitoring.** We used two nodes for a Prometheus-Grafana combination.
- **Workload.** For generating workloads, we used a separate three-node cluster that can produce to and consume from this Kafka cluster.
- **Storage.** We used a single-node NetApp Cloud volumes ONTAP instance with two 500GB GP2 AWS-EBS volumes attached to the instance. These volumes were then exposed to the Kafka cluster as single NFSv4.1 volume through a LIF.

The default properties of Kafka were chosen for all servers. The same was done for the zookeeper swarm.

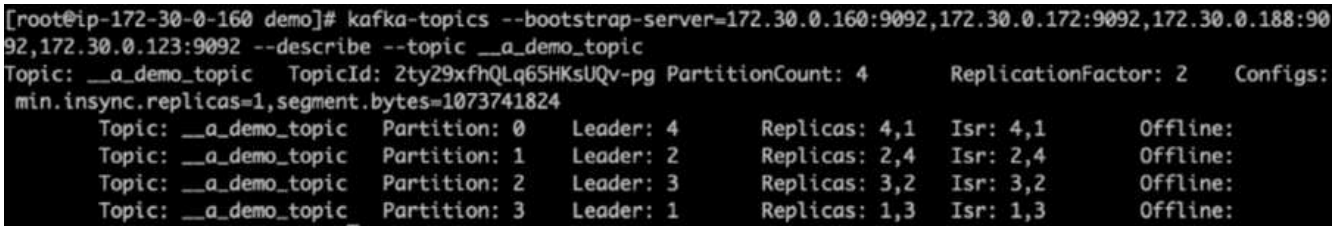
Methodology of testing

1. Update `-is-preserve-unlink-enabled true` to the kafka volume, as follows:

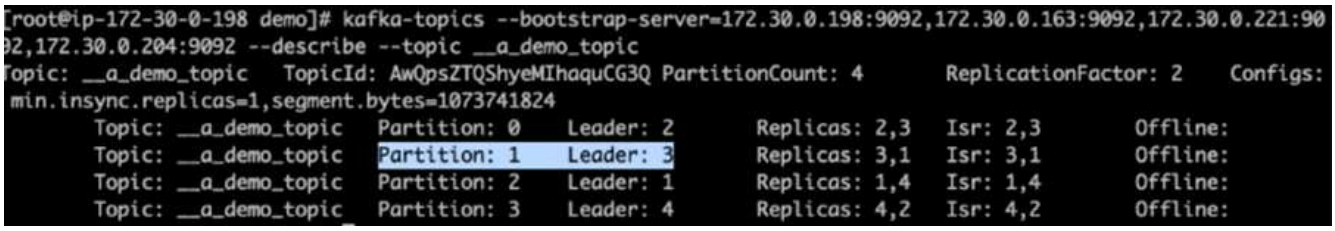
```
aws-shantanclastrecall-aws::*> volume create -vserver kafka_svm -volume
kafka_fg_vol01 -aggregate kafka_aggr -size 3500GB -state online -policy
kafka_policy -security-style unix -unix-permissions 0777 -junction-path
/kafka_fg_vol01 -type RW -is-preserve-unlink-enabled true
[Job 32] Job succeeded: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

2. Two similar Kafka clusters were created with the following difference:
   - **Cluster 1.** The backend NFS v4.1 server running production-ready ONTAP version 9.12.1 was hosted by a NetApp CVO instance. RHEL 8.7/RHEL 9.1 were installed on the brokers.
   - **Cluster 2.** The backend NFS server was a manually created generic Linux NFSv3 server.
3. A demo topic was created on both the Kafka clusters.

   Cluster 1:



   Cluster 2:



4. Data was loaded into these newly created topics for both clusters. This was done using the producer-perf-test toolkit that comes in the default Kafka package:

```
./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. A health check was performed for broker-1 for each of the clusters using telnet:
   - telnet `172.30.0.160 9092`
   - telnet `172.30.0.198 9092`

     A successful health check for brokers on both clusters is shown in the next screenshot:

```
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. To trigger the failure condition that causes Kafka clusters using NFSv3 storage volumes to crash, we initiated the partition reassignment process on both clusters. Partition reassignment was performed using `kafka-reassign-partitions.sh`. The detailed process is as follows:
   a. To reassign the partitions for a topic in a Kafka cluster, we generated the proposed reassignment config JSON (this was performed for both the clusters).

```
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. The generated reassignment JSON was then saved in `/tmp/reassignment- file.json`.
   c. The actual partition reassignment process was triggered by the following command:

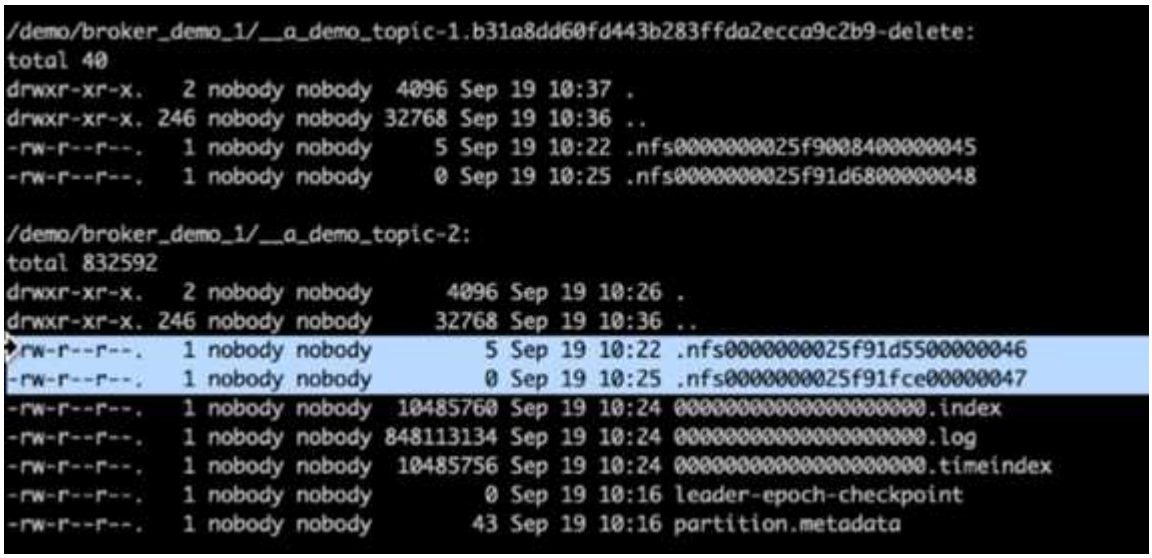
```
kafka-reassign-partitions --bootstrap
-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:9092,172.30.
0.204:9092 --reassignment-json-file /tmp/reassignment-file.json
-execute
```

7. After a few minutes when the reassignment was completed, another health check on the brokers showed that cluster using NFSv3 storage volumes had run into a silly rename issue and had crashed, whereas Cluster 1 using NetApp ONTAP NFSv4.1 storage volumes with the fix continued operations without any disruptions.

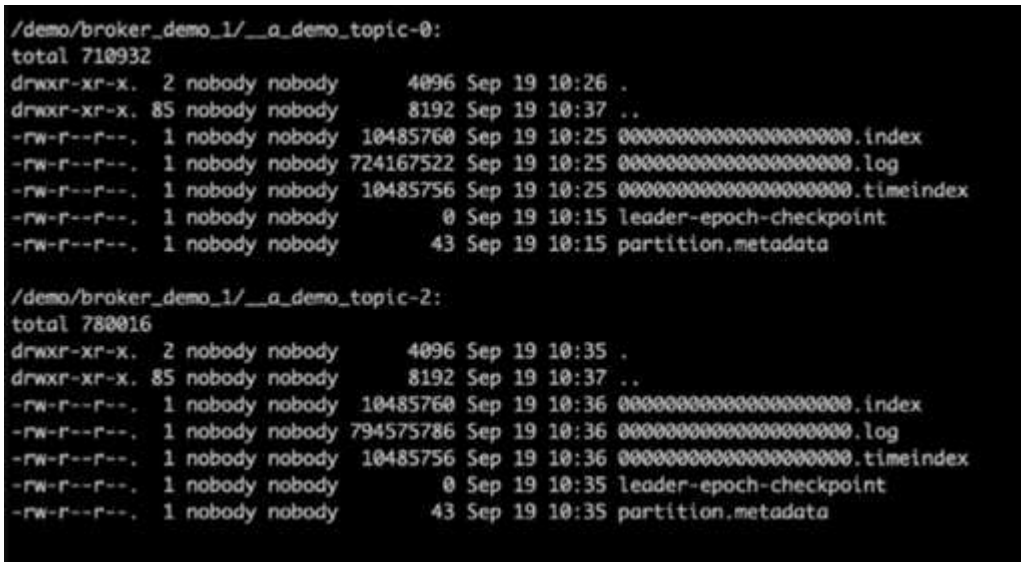
```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

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 is alive.
- Cluster2-broker-1 is dead.

8. Upon checking the Kafka log directories, it was clear that Cluster 1 using NetApp ONTAP NFSv4.1 storage volumes with the fix had clean partition assignment, while Cluster 2 using generic NFSv3 storage did not due to silly rename issues, which led to the crash. The following picture shows partition rebalancing of Cluster 2, which resulted in a silly rename issue on NFSv3 storage.



The following picture shows a clean partition rebalancing of Cluster 1 using NetApp NFSv4.1 storage.



Next: Why NetApp NFS for Kafka workloads?

Why NetApp NFS for Kafka workloads?

Previous: Functional validation - Silly rename fix.

Now that there is a solution for the silly rename issue in NFS storage with Kafka, you can create robust deployments that leverage NetApp ONTAP storage for your Kafka

workload. Not only does this significantly reduce operational overhead, it also brings the following benefits to your Kafka clusters:

- **Reduced CPU utilization on Kafka brokers.** Using disaggregated NetApp ONTAP storage separates disk I/O operations from the broker and thus reduces its CPU footprint.
- **Faster broker recovery-time.** Since disaggregated NetApp ONTAP storage is shared across Kafka broker nodes, a new compute instance can replace a bad broker at any point in a fraction of the time compared to conventional Kafka deployments without rebuilding the data.
- **Storage efficiency.** As the storage layer of the application is now provisioned through NetApp ONTAP, customers can avail all the benefits of storage efficiency that comes with ONTAP, such as in-line data compression, deduplication, and compaction.

These benefits were tested and validated in test cases that we discuss in detail in this section.

Reduced CPU utilization on Kafka broker

We discovered that overall CPU utilization is lower than its DAS counterpart when we ran similar workloads on two sperate Kafka clusters that were identical in their technical specifications but differed in their storage technologies. Not only is the overall CPU utilization lower when Kafka cluster is using ONTAP storage, but the increase in the CPU utilization demonstrated a gentler gradient than in a DAS-based Kafka cluster.

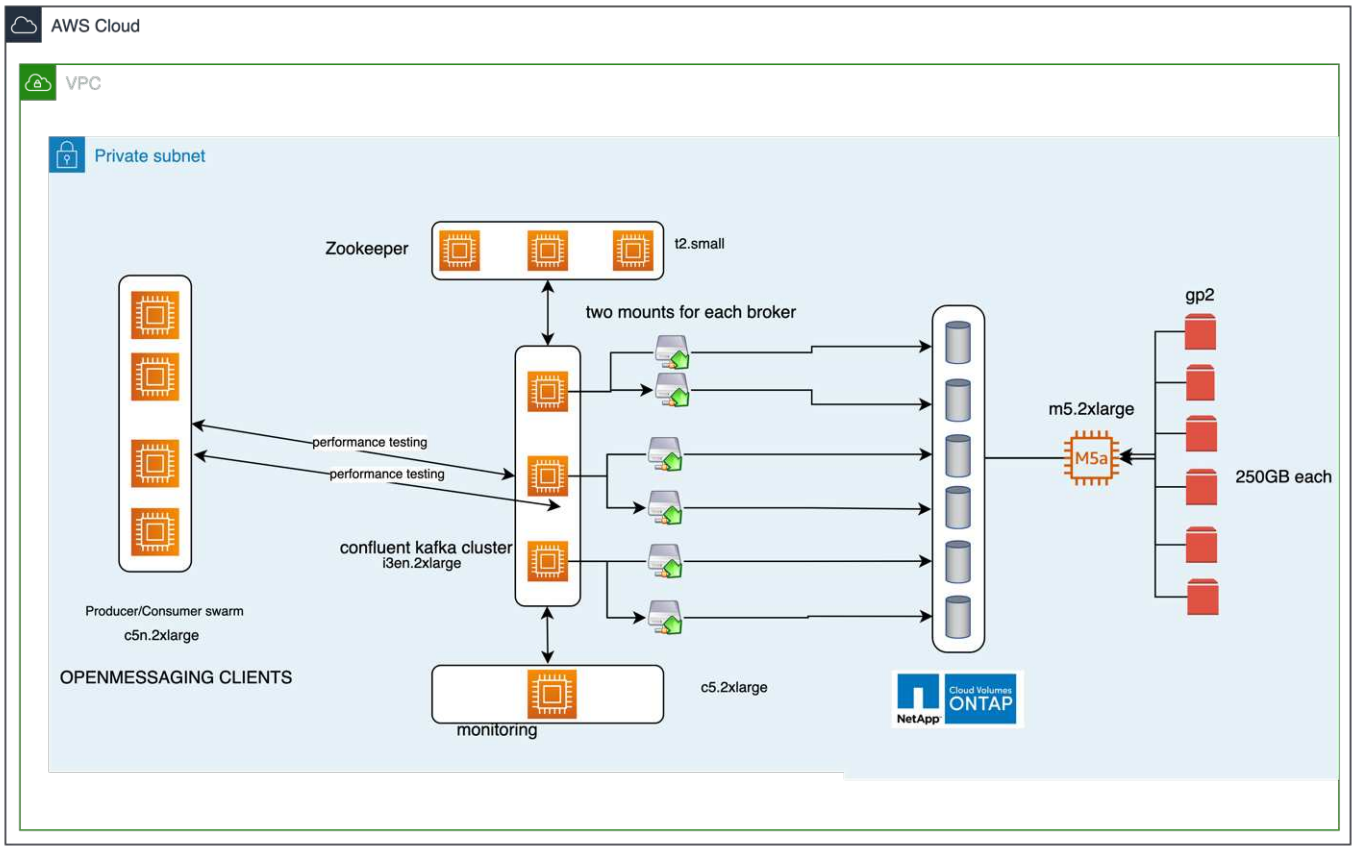
Architectural setup

The following table shows the environmental configuration used to demonstrate reduced CPU utilization.

Platform component	Environment configuration
Kafka 3.2.3 Benchmarking tool: OpenMessaging	• 3 x zookeepers – t2.small • 3 x broker servers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x Producer/Consumer — c5n.2xlarge
Operating system on all nodes	RHEL 8.7 or later
NetApp Cloud Volumes ONTAP instance	Single Node Instance – M5.2xLarge

Benchmarking tool

The benchmarking tool used in this test case is the OpenMessaging framework. OpenMessaging is vendor-neutral and language-independent; it provides industry guidelines for finance, e-commerce, IoT, and big-data; and it helps develop messaging and streaming applications across heterogeneous systems and platforms. The following figure depicts the interaction of OpenMessaging clients with a Kafka cluster.



- **Compute.** We used a three-node Kafka cluster with a three-node zookeeper ensemble running on dedicated servers. Each broker had two NFSv4.1 mount points to a single volume on the NetApp CVO instance through a dedicated LIF.
- **Monitoring.** We used two nodes for a Prometheus-Grafana combination. For generating workloads, we have a separate three-node cluster that can produce to and consume from this Kafka cluster.
- **Storage.** We used a single-node NetApp Cloud volumes ONTAP instance with six 250GB GP2 AWS-EBS volumes mounted on the instance. These volumes were then exposed to the Kafka cluster as six NFSv4.1 volumes through dedicated LIFs.
- **Configuration.** The two configurable elements in this test case were Kafka brokers and OpenMessaging workloads.
    - **Broker config.** The following specifications were selected for the Kafka brokers. We used replication factor of 3 for all measurements, as is highlighted below.

```
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 config.** The following specifications were provided. We specified a target producer rate, highlighted below.

```
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
```

Methodology of testing

1. Two similar clusters were created, each having its own set of benchmarking cluster swarms.
   - **Cluster 1.** NFS-based Kafka cluster.
   - **Cluster 2.** DAS-based Kafka cluster.
2. Using an OpenMessaging command, similar workloads were triggered on each cluster.

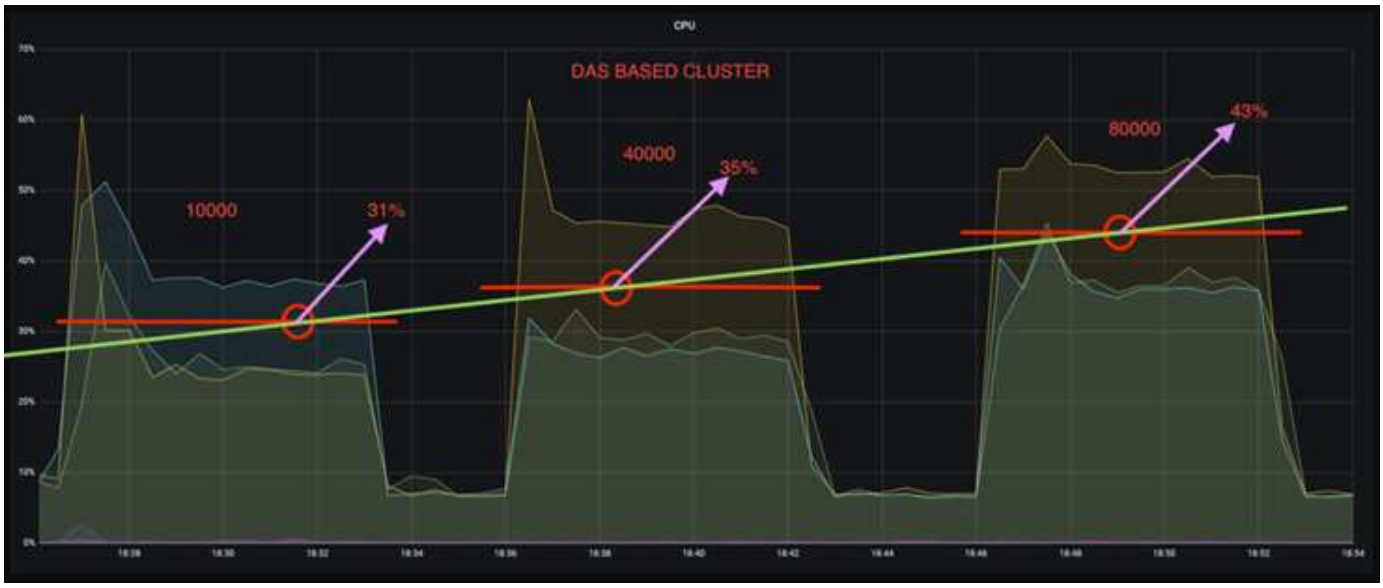
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

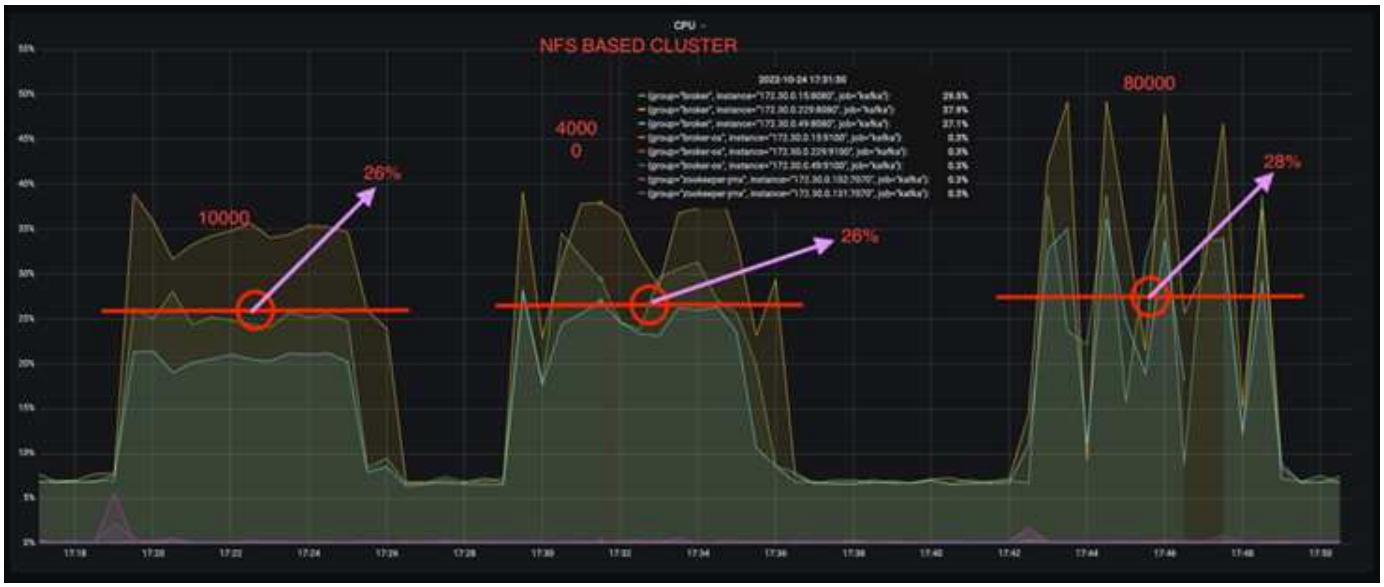
3. The produce rate configuration was increased in four iterations, and CPU utilization was recorded with Grafana. The produce rate was set to the following levels:
    - 10,000
    - 40,000
    - 80,000
    - 100,000

Observation

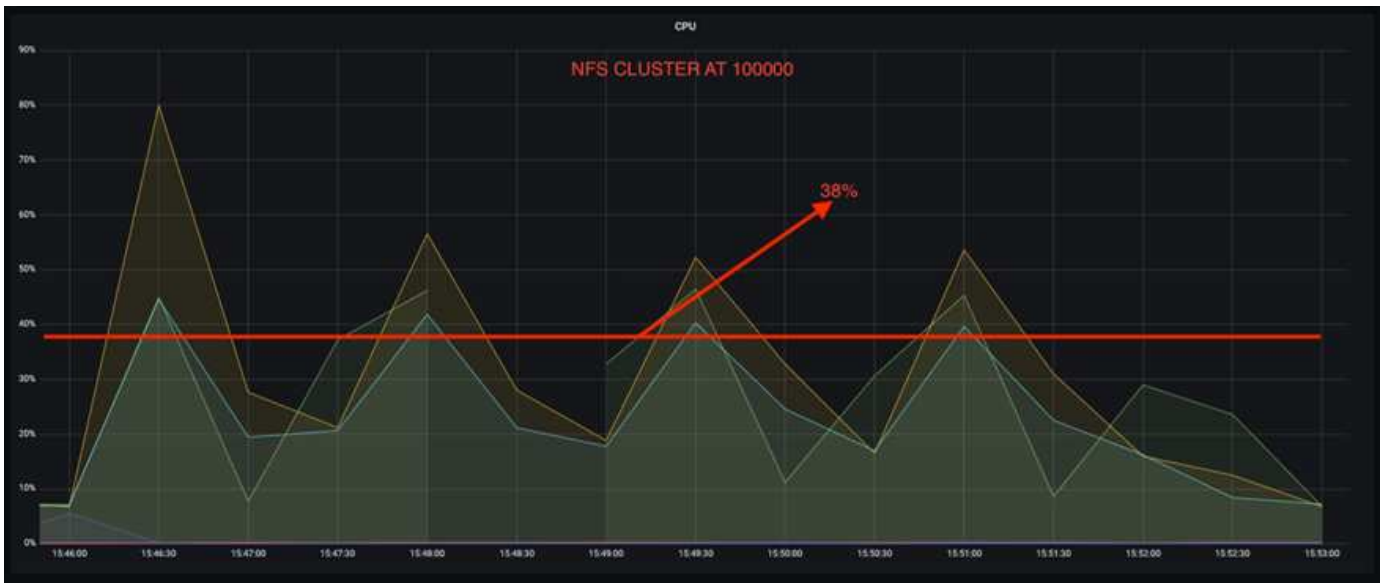
There are two primary benefits of using NetApp NFS storage with Kafka:

- **You can reduce CPU usage by almost one-third.** The overall CPU usage under similar workloads was lower for NFS compared to DAS SSDs; savings range from 5% for lower produce rates to 32% for higher produce rates.
- **A three-fold reduction in CPU utilization drift at higher produce rates.** As expected, there was an upward drift for the increase in CPU utilization as the produce rates were increased. However, CPU utilization on Kafka brokers using DAS went up from 31% for the lower produce rate to 70% for the higher produce rate, a 39% increase. However, with an NFS storage backend, the CPU utilization went up from 26% to 38%, a 12% increase.





Also, at 100,000 messages, DAS shows more CPU utilization than an NFS cluster.



Faster broker recovery

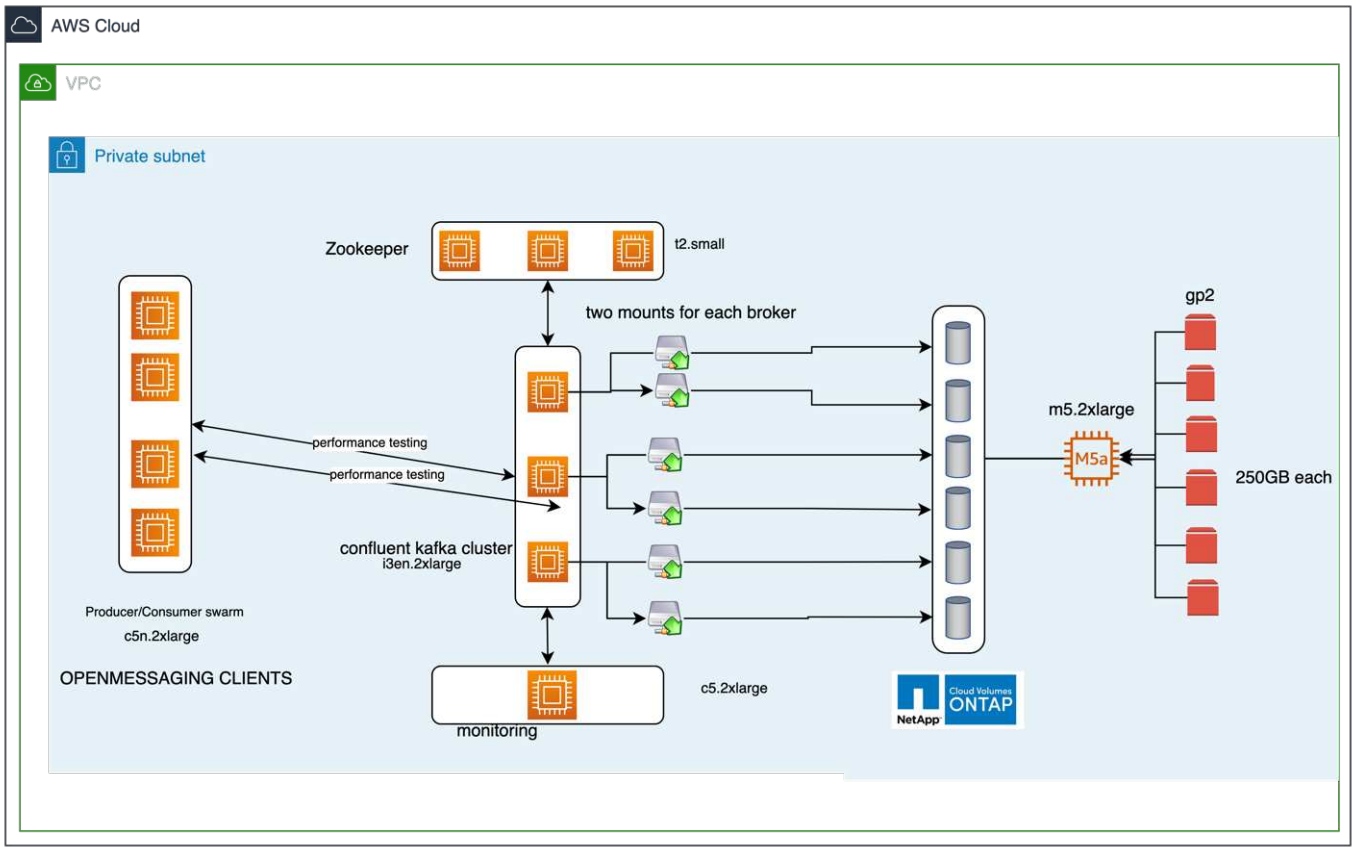
We discovered that Kafka brokers recover faster when they are using shared NetApp NFS storage. When a broker crashes in a Kafka cluster, this broker can be replaced by a healthy broker with a same broker ID. Upon performing this test case, we found that, in the case of a DAS-based Kafka cluster, the cluster rebuilds the data on a newly added healthy broker, which is time consuming. In the case of a NetApp NFS-based Kafka cluster, the replacing broker continues to read data from the previous log directory and recovers much faster.

Architectural setup

The following table shows the environmental configuration for a Kafka cluster using NAS.

Platform component	Environment configuration
Kafka 3.2.3	• 3 x zookeepers – t2.small • 3 x broker servers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producer/consumer — c5n.2xlarge • 1 x backup Kafka node – i3en.2xlarge
Operating system on all nodes	RHEL8.7 or later
NetApp Cloud Volumes ONTAP instance	Single-node instance – M5.2xLarge

The following figure depicts the architecture of an NAS-based Kafka cluster.



- **Compute.** A three-node Kafka cluster with a three-node zookeeper ensemble running on dedicated servers. Each broker has two NFS mount points to a single volume on the NetApp CVO instance via a dedicated LIF.
- **Monitoring.** Two nodes for a Prometheus-Grafana combination. For generating workloads, we use a separate three-node cluster that can produce and consume to this Kafka cluster.
- **Storage.** A single-node NetApp Cloud volumes ONTAP instance with six 250GB GP2 AWS-EBS volumes mounted on the instance. These volumes are then exposed to the Kafka cluster as six NFS volume through dedicated LIFs.
- **Broker configuration.** The one configurable element in this test case are Kafka brokers. The following specifications were selected for the Kafka brokers. The `replica.lag.time.mx.ms` is set to a high value because this determines how fast a particular node is taken out of the ISR list. When you switch between bad and healthy nodes, you don't want that broker ID to be excluded from the ISR list.

```
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
```

Methodology of testing

1. Two similar clusters were created:
   - An EC2-based confluent cluster.
   - A NetApp NFS-based confluent cluster.
2. One standby Kafka node was created with a configuration identical to the nodes from the original Kafka cluster.
3. On each of the clusters, a sample topic was created, and approximately 110GB of data was populated on each of the brokers.
   - **EC2-based cluster.** A Kafka broker data directory is mapped on `/mnt/data-2` (In the following figure, Broker-1 of cluster1 [left terminal]).
   - **NetApp NFS-based cluster.** A Kafka broker data directory is mounted on NFS point `/mnt/data` (In the following figure, Broker-1 of cluster2 [right terminal]).

4. In each of the clusters, Broker-1 was terminated to trigger a failed broker recovery process.
5. After the broker was terminated, the broker IP address was assigned as a secondary IP to the standby broker. This was necessary because a broker in a Kafka cluster is identified by the following:
   - **IP address.** Assigned by reassigning the failed broker IP to the standby broker.
   - **Broker ID.** This was configured in the standby broker `server.properties`.
6. Upon IP assignment, the Kafka service was started on the standby broker.
7. After a while, the server logs were pulled to check the time taken to build data on the replacement node in the cluster.

Observation

Kafka broker recovery was almost nine times faster. The time it took to recover a failed broker node was found to be significantly faster when using NetApp NFS shared storage compared to using DAS SSDs in a Kafka cluster. For 1TB of topic data, the recovery time for a DAS-based cluster was 48 minutes, compared to less than 5 minutes for a NetApp-NFS based Kafka cluster.

We observed that the EC2-based cluster took 10 minutes to rebuild the 110GB of data on the new broker node, whereas the NFS- based cluster completed the recovery in 3 minutes. We also observed in the in logs that consumer offsets for the partitions for EC2 were 0, while, on the NFS cluster, consumer offsets were picked up from the previous broker.

```
[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-based cluster

1. The backup node started at 08:55:53,730.

```
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 (or
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 172.30.0.17:2181,172.3
6 [2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. The data rebuilding process ended at 09:05:24,860. Processing 110GB of data required approximately 10 minutes.

```
[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-based cluster

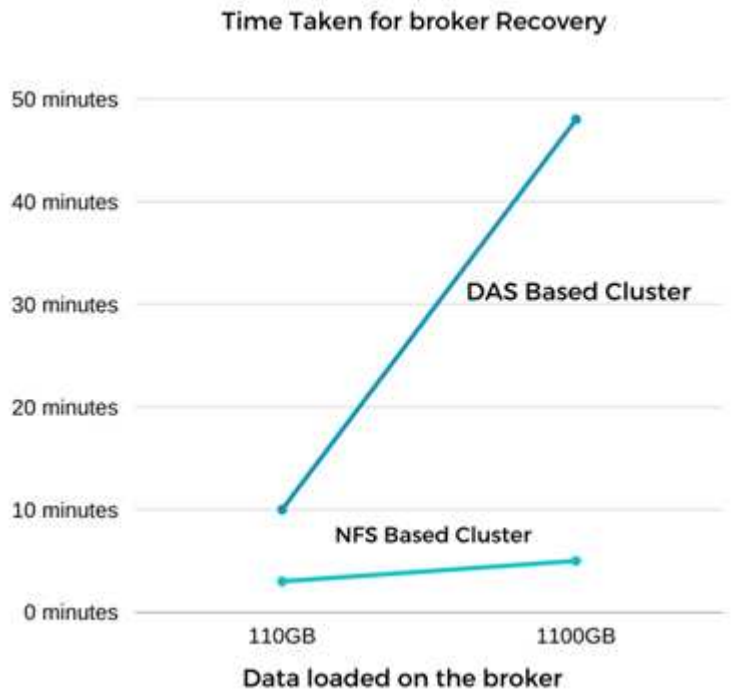
1. The backup node was started at 09:39:17,213. The starting log entry is highlighted below.

```
2 [2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiati
3 [2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.
4 [2022-10-31 09:39:17,213] INFO starting (kafka.server.KafkaServer)
5 [2022-10-31 09:39:17,214] INFO Connecting to zookeeper on 172.30.0.22:2181,172.30.
6 [2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new s
7 [2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a
8 [2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in
```

2. The data rebuild process ended at 09:42:29,115. Processing 110GB of data required approximately 3 minutes.

```
[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)
```

The test was repeated for brokers containing around 1TB data, which took approximately 48 minutes for the DAS and 3 min for NFS. The results are depicted in the following graph.



Storage efficiency

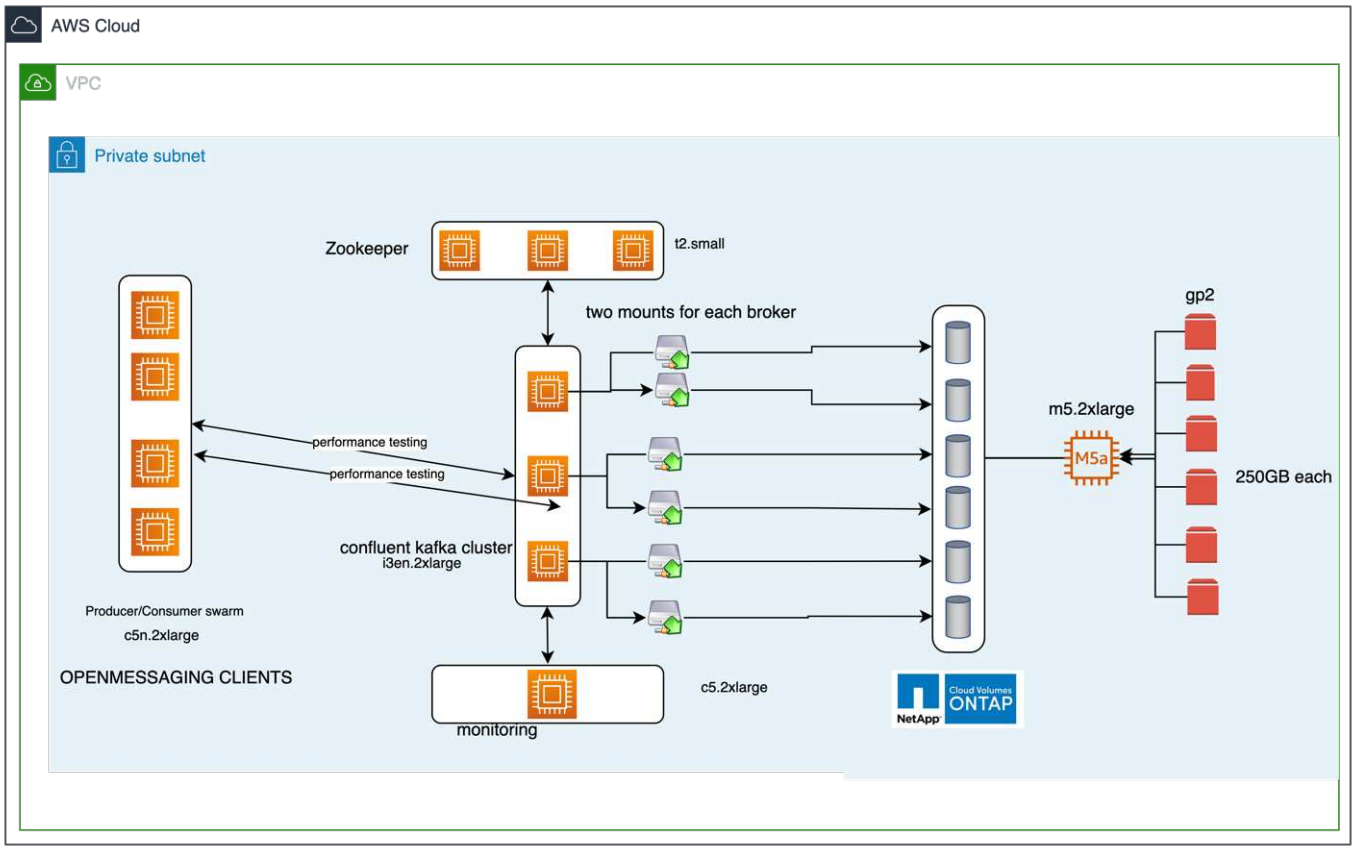
Because the storage layer of the Kafka cluster was provisioned through NetApp ONTAP, we got all the storage efficiency capabilities of ONTAP. This was tested by generating a significant amount of data on a Kafka cluster with NFS storage provisioned on Cloud Volumes ONTAP. We could see that there was a significant space reduction due to ONTAP capabilities.

Architectural setup

The following table shows the environmental configuration for a Kafka cluster using NAS.

Platform component	Environment configuration
Kafka 3.2.3	• 3 x zookeepers – t2.small • 3 x broker servers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producer/consumer — c5n.2xlarge *
Operating system on all nodes	RHEL8.7 or later
NetApp Cloud Volumes ONTAP instance	Single node instance – M5.2xLarge

The following figure depicts the architecture of an NAS-based Kafka cluster.



- **Compute.** We used a three-node Kafka cluster with a three-node zookeeper ensemble running on dedicated servers. Each broker had two NFS mount points to a single volume on the NetApp CVO instance

via a dedicated LIF.

- **Monitoring.** We used two nodes for a Prometheus-Grafana combination. For generating workloads, we used a separate three-node cluster that could produce and consume to this Kafka cluster.
- **Storage.** We used a single-node NetApp Cloud Volumes ONTAP instance with six 250GB GP2 AWS-EBS volumes mounted on the instance. These volumes were then exposed to the Kafka cluster as six NFS volumes through dedicated LIFs.
- **Configuration.** The configurable elements in this test case were the Kafka brokers.

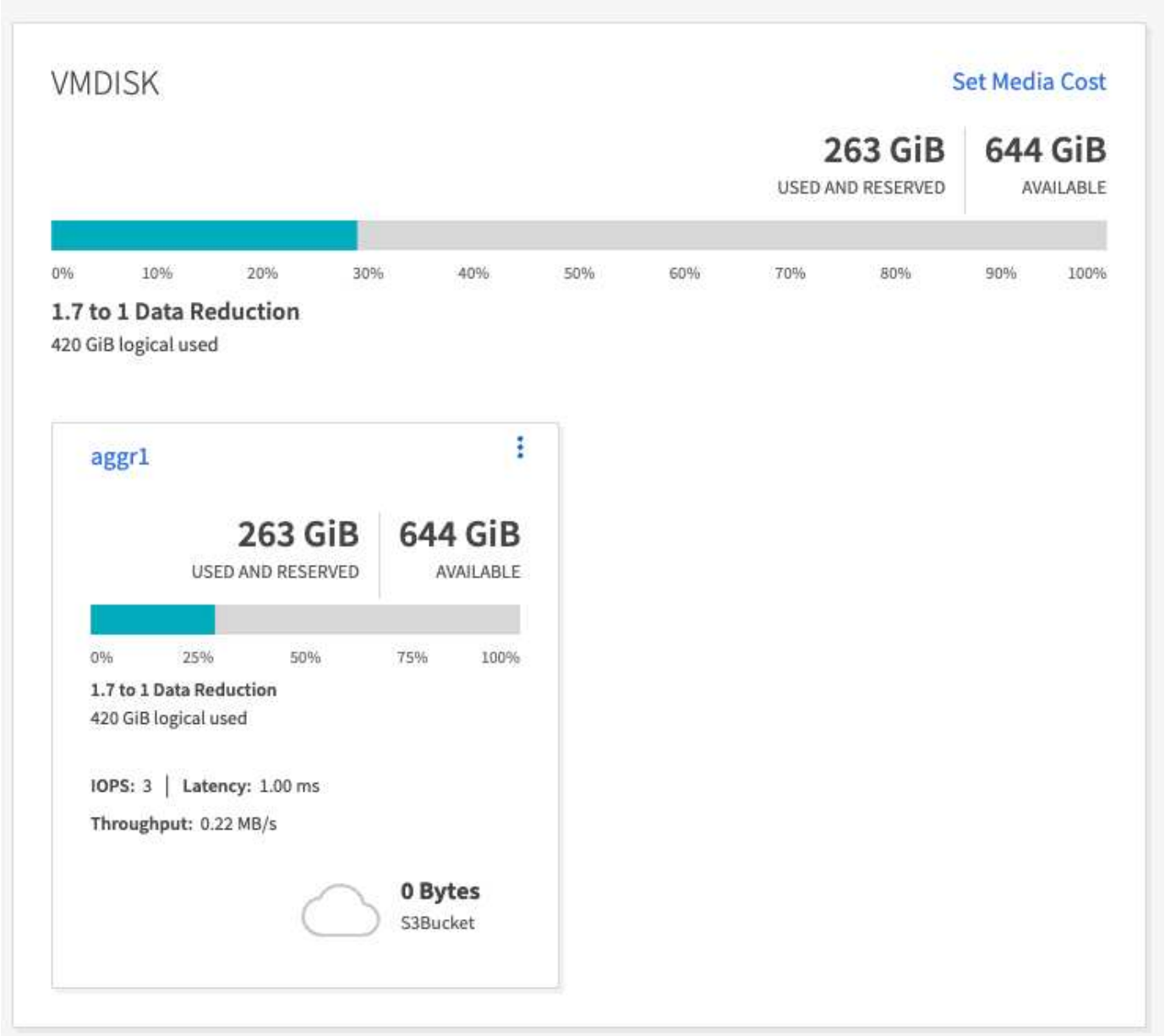
Compression was switched off on the producer's end, thus enabling producers to generate high throughput. Storage efficiency was instead handled by the compute layer.

Methodology of testing

1. A Kafka cluster was provisioned with the specifications mentioned above.
2. On the cluster, about 350GB data was produced using the OpenMessaging Benchmarking tool.
3. After the workload was completed, the storage efficiency statistics were collected using ONTAP System Manager and the CLI.

Observation

For data that was generated using the OMB tool, we saw space savings of ~33% with a storage efficiency ratio of 1.70:1. As seen in the following figures, the logical space used by the data produced was 420.3GB and the physical space used to hold the data was 281.7GB.




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

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

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

Next: Performance overview and validation in AWS.

Performance overview and validation in AWS

Previous: Why NetApp NFS for Kafka workloads?.

A Kafka cluster with the storage layer mounted on NetApp NFS was benchmarked for performance in the AWS cloud. The benchmarking examples are described in the following sections.

Kafka in AWS cloud with NetApp Cloud Volumes ONTAP (high-availability pair and single node)

A Kafka cluster with NetApp Cloud Volumes ONTAP (HA pair) was benchmarked for performance in the AWS cloud. This benchmarking is described in the following sections.

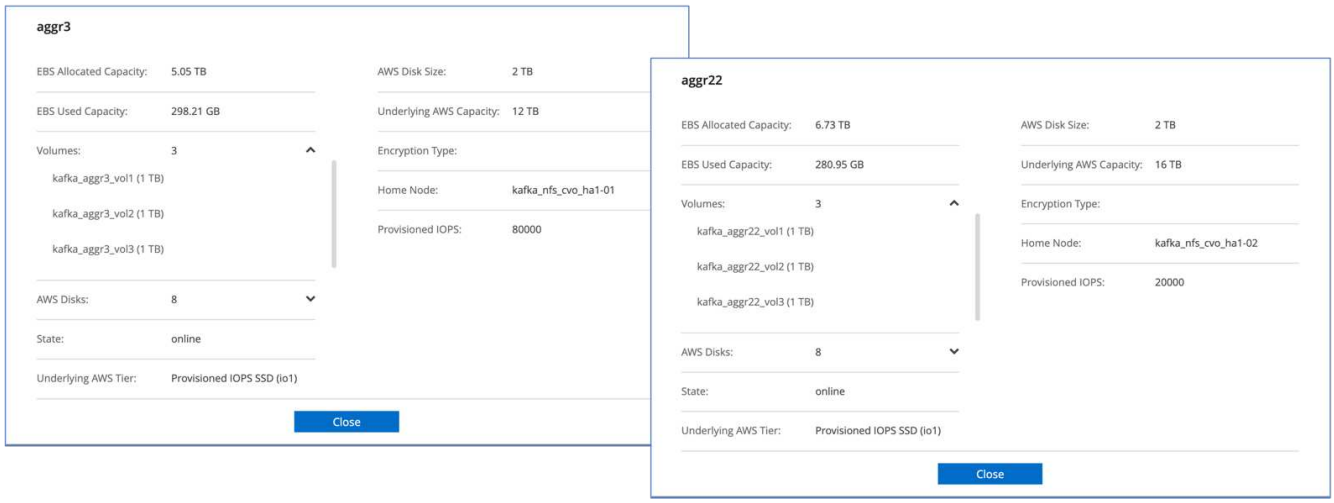
Architectural setup

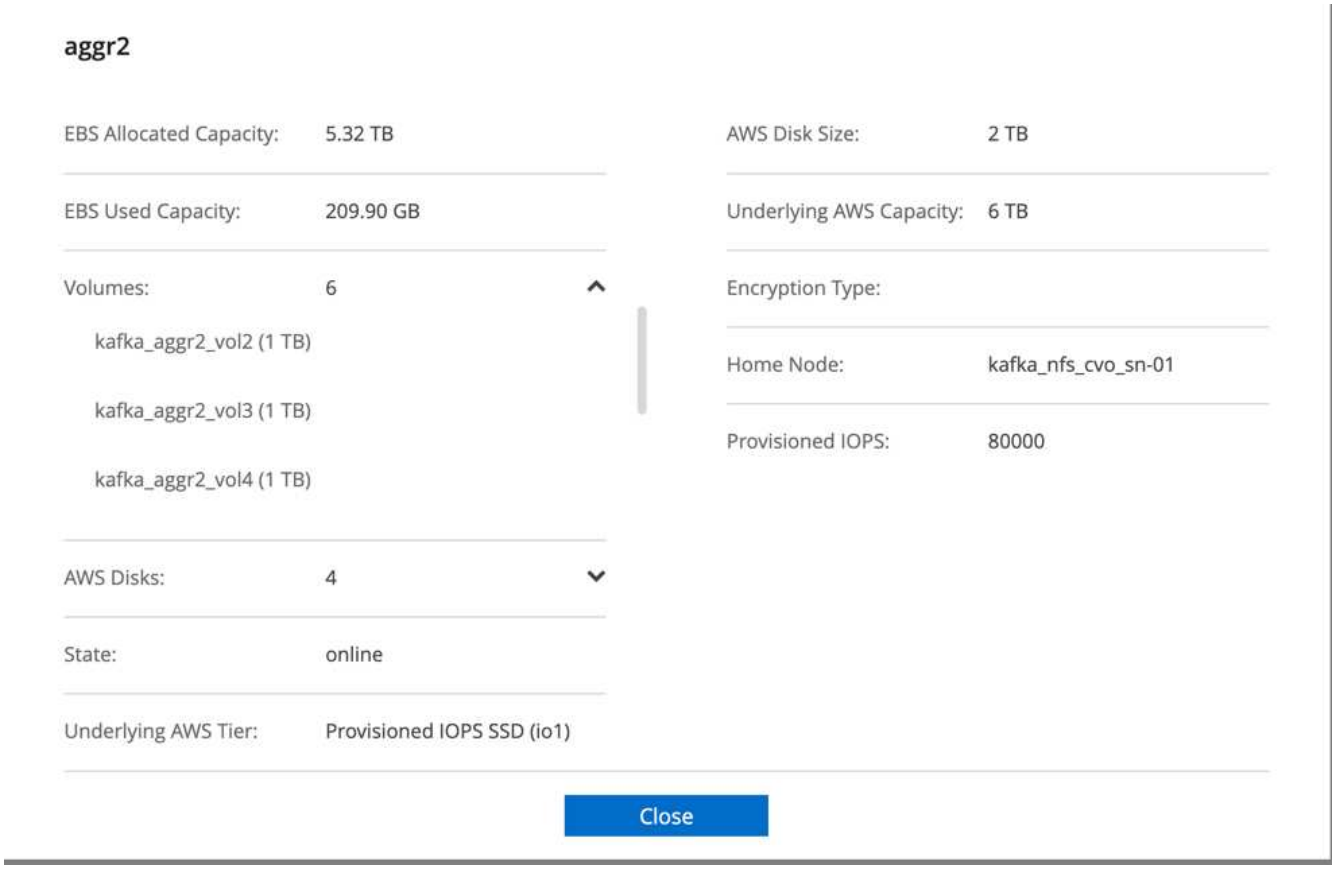
The following table shows the environmental configuration for a Kafka cluster using NAS.

Platform component	Environment configuration
Kafka 3.2.3	• 3 x zookeepers – t2.small • 3 x broker servers – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x producer/consumer — c5n.2xlarge *
Operating system on all nodes	RHEL8.6
NetApp Cloud Volumes ONTAP instance	HA pair instance – m5dn.12xLarge x 2node Single Node Instance - m5dn.12xLarge x 1 node

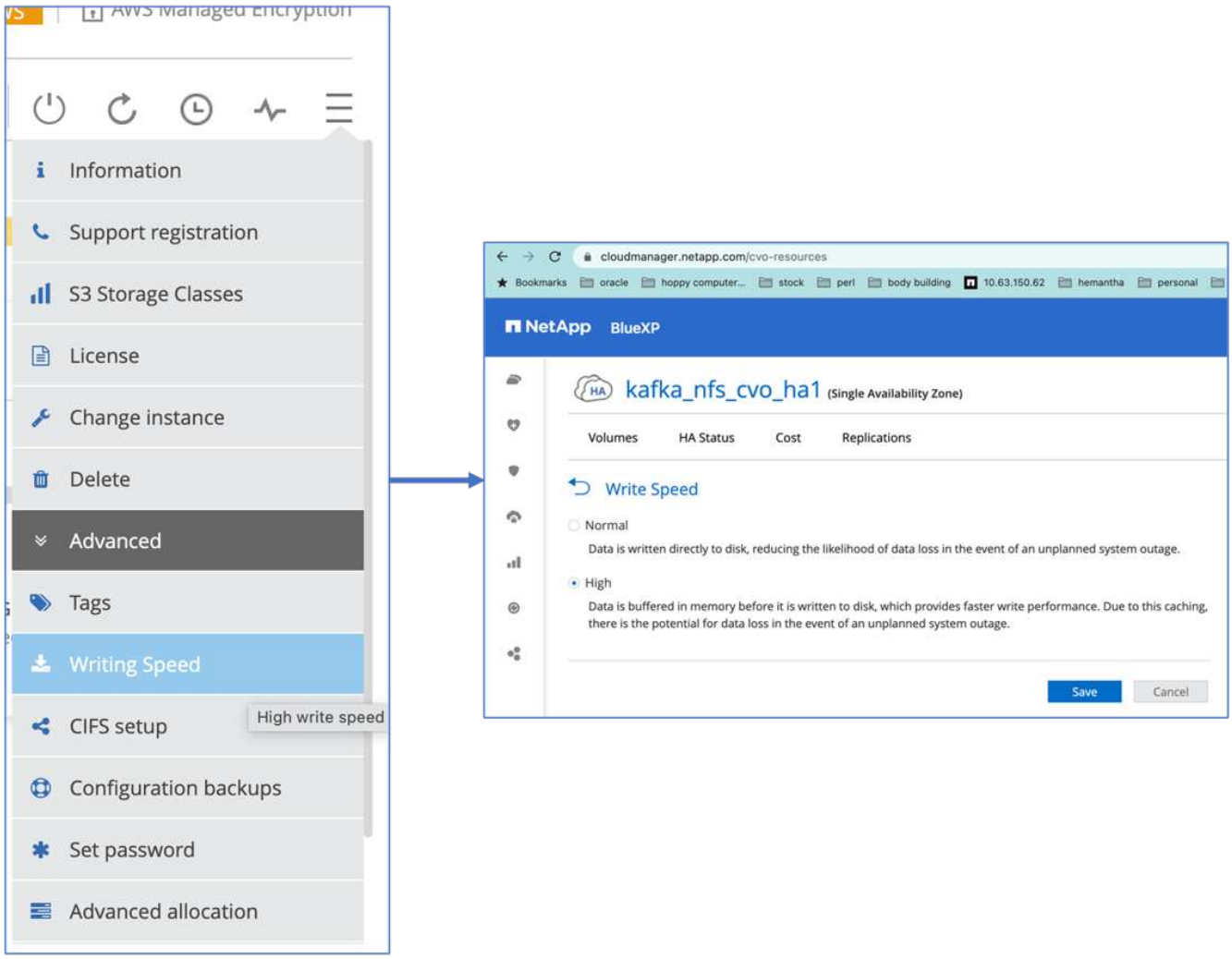
NetApp cluster volume ONTAP setup

1. For the Cloud Volumes ONTAP HA pair, we created two aggregates with three volumes on each aggregate on each storage controller. For the single Cloud Volumes ONTAP node, we create six volumes in an aggregate.



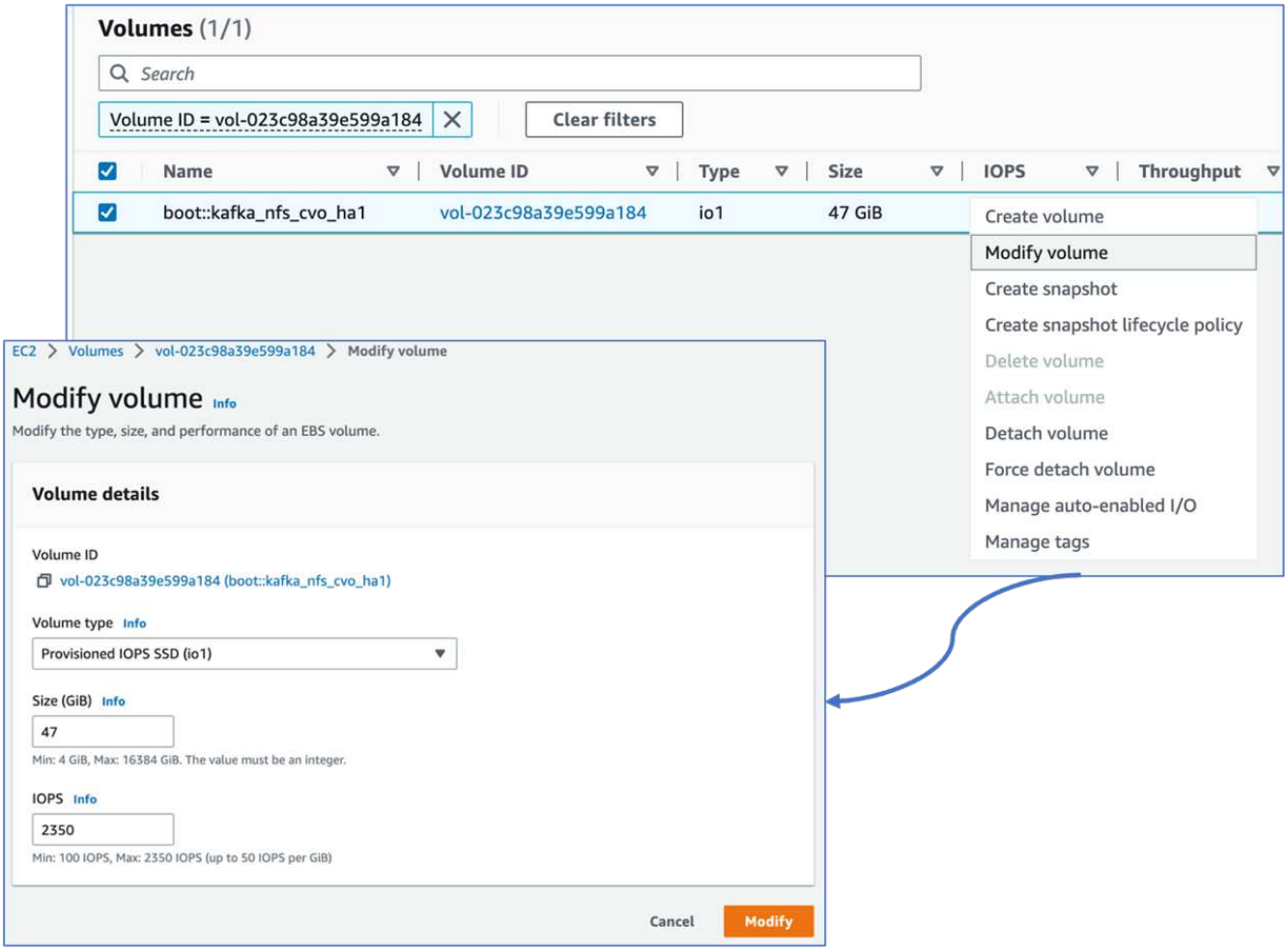


2. To achieve better network performance, we enabled high speed networking for both the HA pair and the single node.



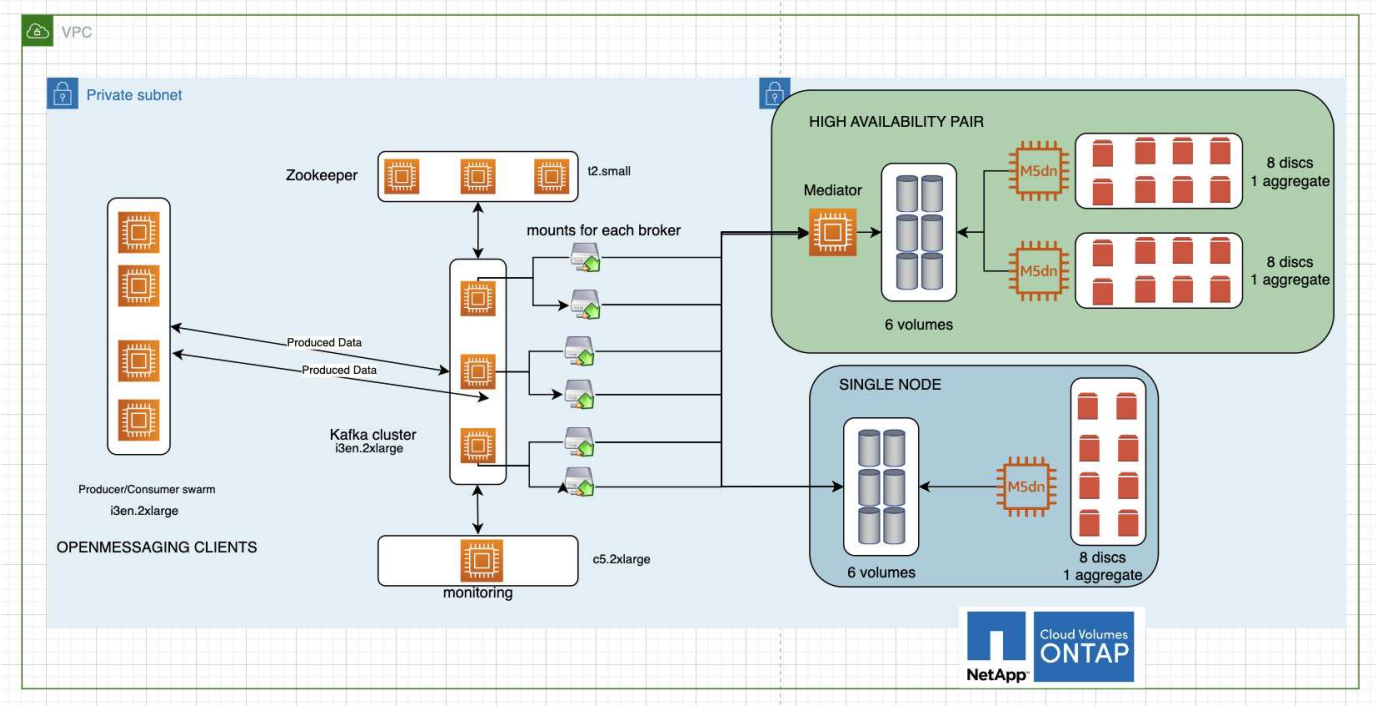
3. We noticed that the ONTAP NVRAM had more IOPS so we changed the IOPS to 2350 for the Cloud Volumes ONTAP root volume. The root volume disk in Cloud Volumes ONTAP was 47GB in size. The following ONTAP command is for the HA pair, and the same step is applicable for the single node.

```
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::*>
```



The following figure depicts the architecture of an NAS-based Kafka cluster.

- **Compute.** We used a three-node Kafka cluster with a three-node zookeeper ensemble running on dedicated servers. Each broker had two NFS mount points to a single volume on the Cloud Volumes ONTAP instance through a dedicated LIF.
- **Monitoring.** We used two nodes for a Prometheus-Grafana combination. For generating workloads, we used a separate three-node cluster that could produce and consume to this Kafka cluster.
- **Storage.** We used an HA-pair Cloud volumes ONTAP instance with one 6TB GP3 AWS-EBS volume mounted on the instance. The volume was then exported to the Kafka broker with an NFS mount.



OpenMessage Benchmarking configurations

1. For better NFS performance, we need more network connections between the NFS server and the NFS client, which can be created using nconnect. Mount the NFS volumes on the broker nodes with the nconnect option by running the following command:

```
[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. Check the network connections in Cloud Volumes ONTAP. The following ONTAP command is used from the single Cloud Volumes ONTAP node. The same step is applicable to the Cloud Volumes ONTAP HA pair.

```
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. We use the following Kafka `server.properties` in all Kafka brokers for the Cloud Volumes ONTAP HA pair. The `log.dirs` property is different for each broker, and the remaining properties are common for brokers. For broker1, the `log.dirs` value is as follows:

```
[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 ~]#
```

   - For broker2, the `log.dirs` property value is as follows:

```
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
```

   - For broker3, the `log.dirs` property value is as follows:

```
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. For the single Cloud Volumes ONTAP node, The Kafka `servers.properties` is the same as for the Cloud Volumes ONTAP HA pair except for the `log.dirs` property.
   - For broker1, the `log.dirs` value is as follows:

```
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
```

   - For broker2, the `log.dirs` value is as follows:

```
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
```

   - For broker3, the `log.dirs` property value is as follows:

```
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. The workload in the OMB is configured with the following properties: `(/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
```

   The `messageSize` can vary for each use case. In our performance test, we used 3K.

We used two different drivers, Sync or Throughput, from OMB to generate the workload on the Kafka cluster.

- The yaml file used for Sync driver properties is as follows `(/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
```

- The yaml file used for the Throughput driver properties is as follows `(/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
```

Methodology of testing

1. A Kafka cluster was provisioned as per the specification described above using Terraform and Ansible. Terraform is used to build the infrastructure using AWS instances for the Kafka cluster and Ansible builds the Kafka cluster on them.
2. An OMB workload was triggered with the workload configuration described above and the Sync driver.

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

3. Another workload was triggered with the Throughput driver with same workload configuration.

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

Observation

Two different types of drivers were used to generate workloads to benchmark the performance of a Kafka instance running on NFS. The difference between the drivers is the log flush property.

For a Cloud Volumes ONTAP HA pair:

- Total throughput generated consistently by the Sync driver: ~1236 MBps.
- Total throughput generated for the Throughput driver: peak ~1412 MBps.

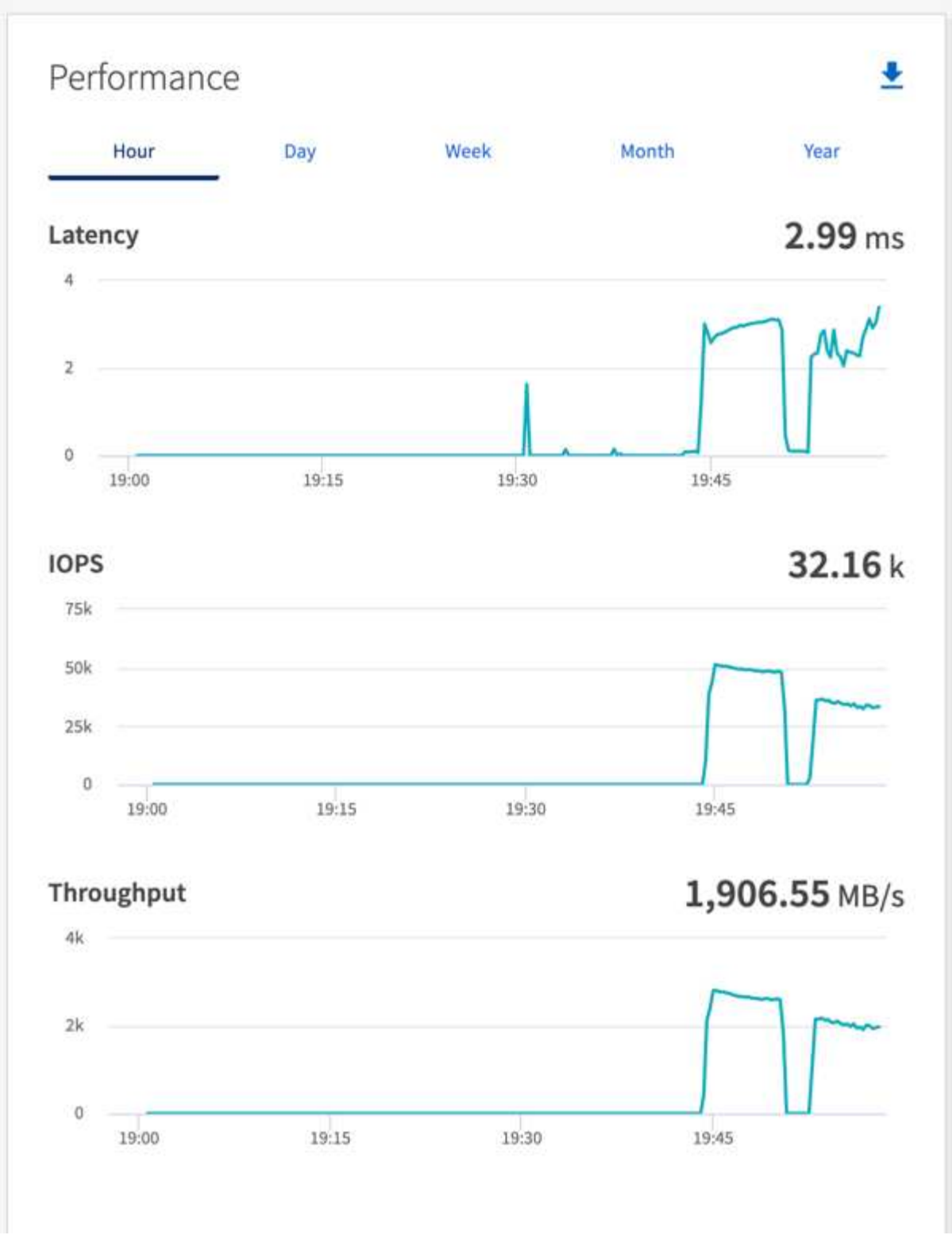
For a single Cloud Volumes ONTAP node:

- Total throughput generated consistently by the Sync driver: ~ 1962MBps.
- Total throughput generated by the Throughput driver: peak ~1660MBps

The Sync driver can generate consistent throughput as logs are flushed to the disk instantly, whereas the Throughput driver generates bursts of throughput as logs are committed to disk in bulk.

These throughput numbers are generated for the given AWS configuration. For higher performance requirements, the instance types can be scaled up and tuned further for better throughput numbers. The total throughput or total rate is the combination of both producer and consumer rate.

Be sure to check the storage throughput when performing throughput or sync driver benchmarking.



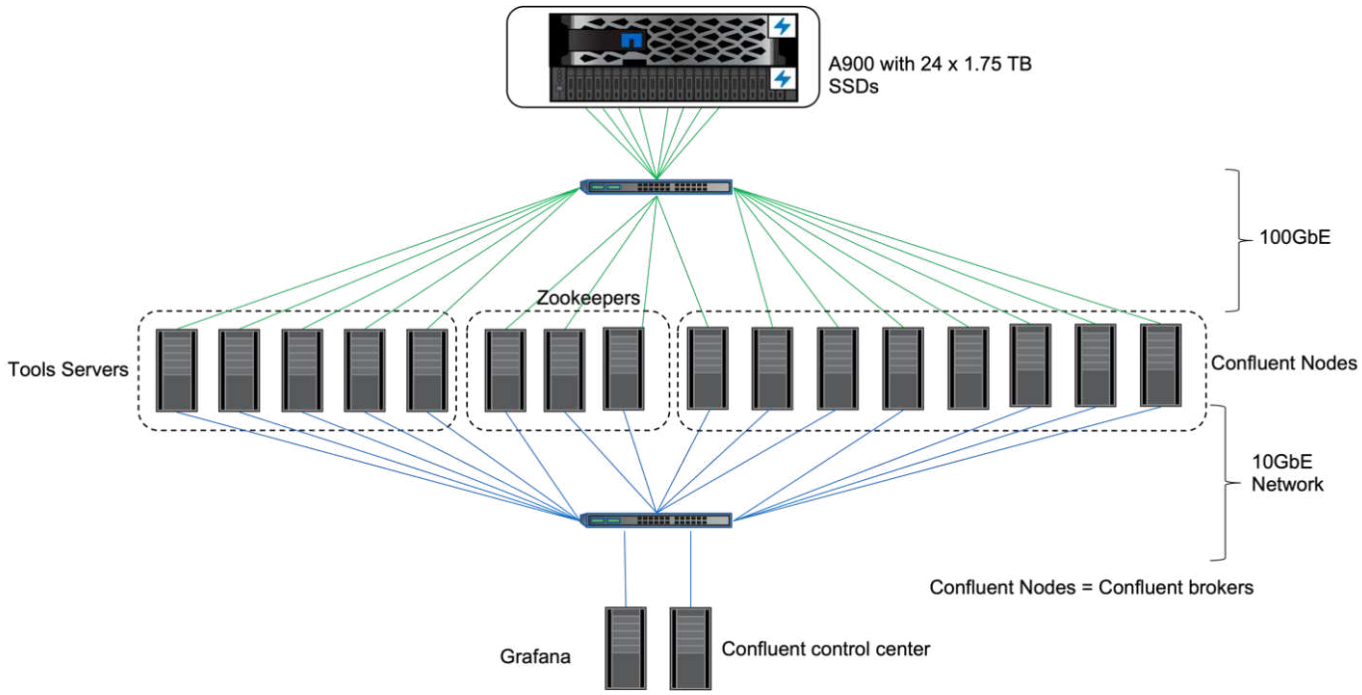
Next: Performance overview and validation with AFF on-premises.

Performance overview and validation with AFF A900 on-premises

Previous: Performance overview and validation in AWS.

On-premises, we used the NetApp AFF A900 storage controller with ONTAP 9.12.1RC1 to validate the performance and scaling of a Kafka cluster. We used the same testbed as in our previous tiered storage best practices with ONTAP and AFF.

We used Confluent Kafka 6.2.0 to evaluate the AFF A900. The cluster features eight broker nodes and three zookeeper nodes. For performance testing, we used five OMB worker nodes.



Storage configuration

We used NetApp FlexGroups instances to provide a single namespace for log directories, simplifying recovery and configuration. We used NFSv4.1 and pNFS to provide direct path access to log segment data.

Client tuning

Each client mounted the FlexGroup instance with the following command.

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

In addition, we increased the `max_session_slots` from the default `64` to `180`. This matches the default session slot limit in ONTAP.

Kafka broker tuning

To maximize throughput in the system under test, we significantly increased the default parameters for certain key thread pools. We recommend following Confluent Kafka best practices for most configurations. This tuning was used to maximize the concurrency of outstanding I/O to storage. These parameters can be adjusted to match your broker's compute resources and storage attributes.

```
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
```

Workload generator testing methodology

We used the same OMB configurations as for cloud testing for the Throughput driver and topic configuration.

1. A FlexGroup instance was provisioned using Ansible on an AFF cluster.

```
---
- 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 was enabled on the ONTAP SVM.

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

3. The workload was triggered with the Throughput driver using with same workload configuration as for Cloud Volumes ONTAP. See the section “Steady state performance” below. The workload used a replication factor of 3, meaning three copies of log segments were maintained in NFS.

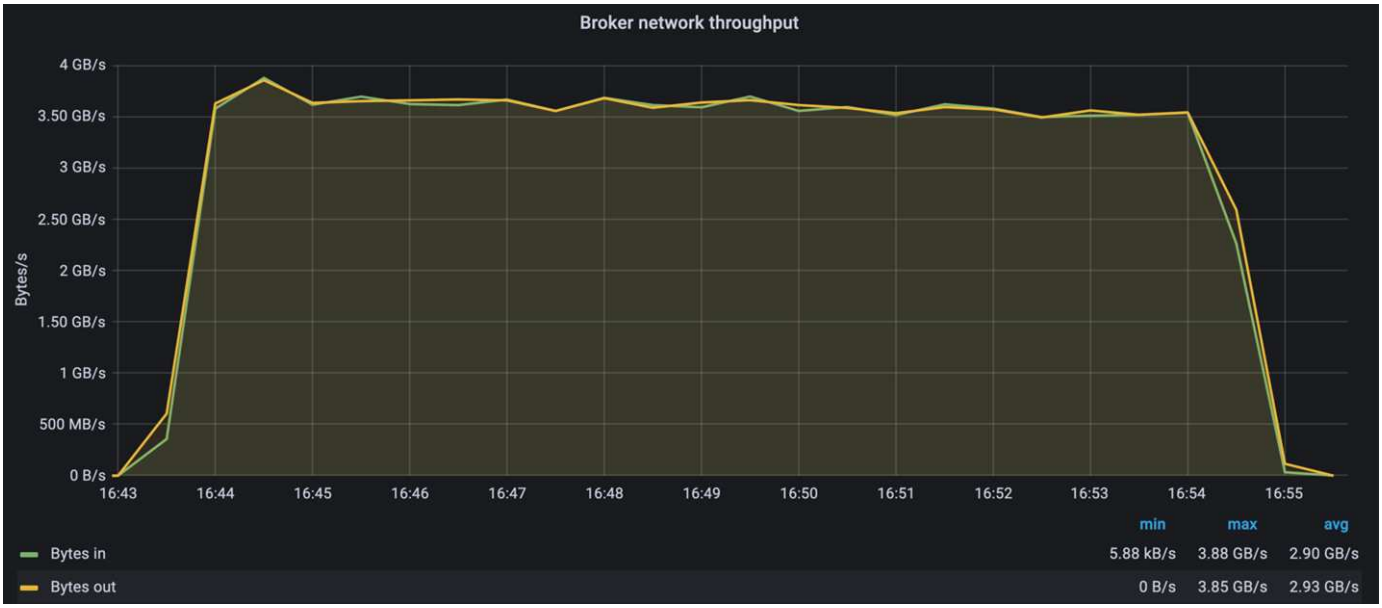
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. Finally, we completed measurements using a backlog to measure the ability of consumers to catch up to the latest messages. OMB constructs a backlog by pausing consumers during the beginning of a measurement. This produces three distinct phases: backlog creation (producer-only traffic), backlog draining (a consumer-heavy phase in which consumers catch up on missed events in a topic), and the steady state. See the section “exploring storage limits” for more information.

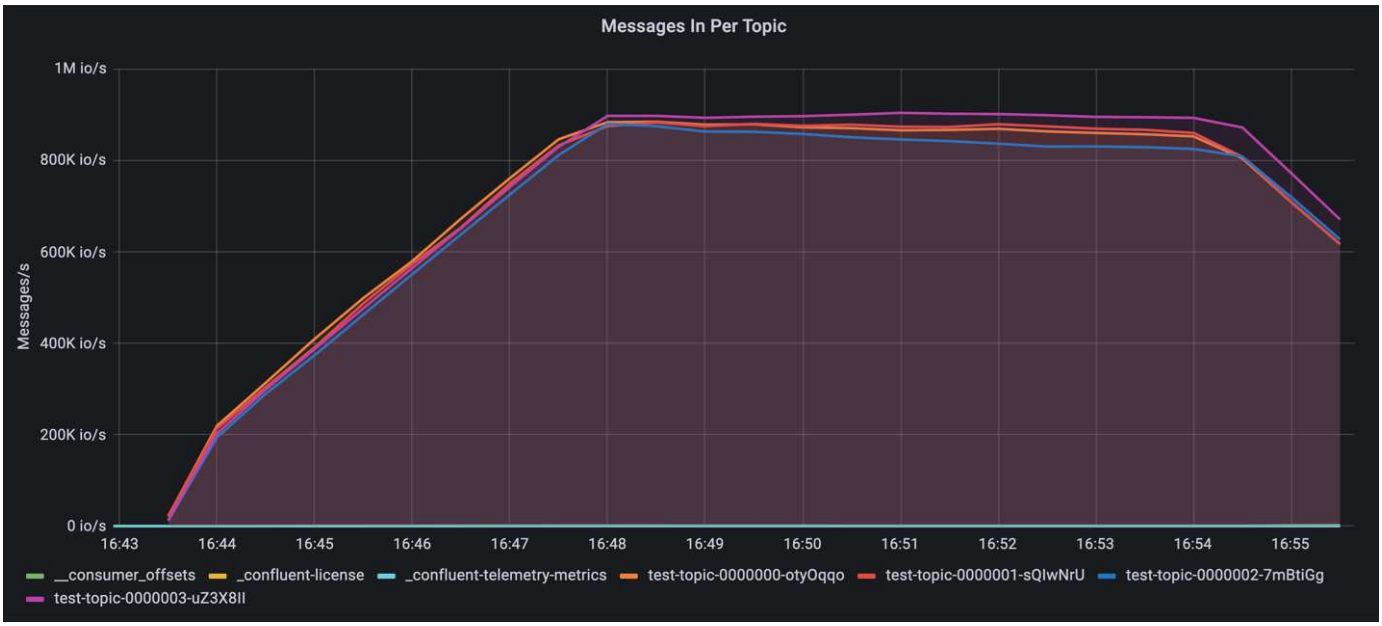
Steady state performance

We evaluated the AFF A900 using the OpenMessaging Benchmark to provide a similar comparison as for Cloud Volumes ONTAP in AWS and DAS in AWS. All performance values represent Kafka-cluster throughput at the producer and consumer level.

Steady state performance with Confluent Kafka and the AFF A900 achieved over 3.4GBps average throughput for both producer and consumers. This is over 3.4 million messages across the Kafka cluster. By visualizing the sustained throughput in bytes per second for BrokerTopicMetrics, we see the excellent steady state performance and traffic supported by the AFF A900.



This aligns well with the view of messages delivered per topic. The following graph provides a per-topic breakdown. In the configuration tested we saw nearly 900k messages per topic across four topics.

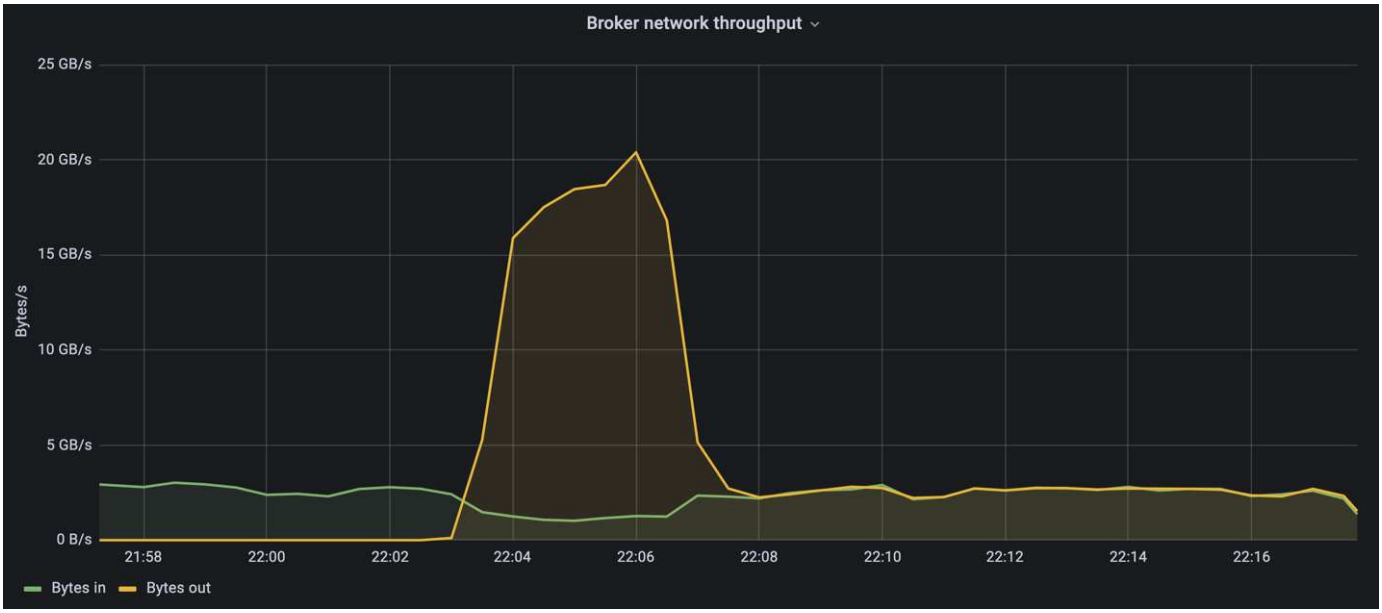


Extreme performance and exploring storage limits

For AFF, we also tested with OMB using the backlog feature. The backlog feature pauses consumer subscriptions while a backlog of events is built up in the Kafka cluster. During this phase, only producer traffic occurs, which generates events that are committed to logs. This most closely emulates batch processing or offline analytics workflows; in these workflows, consumer subscriptions are started and must read historical data that has already been evicted from the broker cache.

To understand the storage limitations on consumer throughput in this configuration, we measured the producer-only phase to understand how much write traffic the A900 could absorb. See the next section “Sizing guidance” to understand how to leverage this data.

During the producer-only part of this measurement, we saw high peak throughput that pushed the limits of A900 performance (when other broker resources were not saturated serving producer and consumer traffic).



We increased the message size to 16k for this measurement to limit per-message overheads and maximize storage throughput to NFS mount points.

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

The Confluent Kafka cluster achieved a peak producer throughput of 4.03GBps.

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

After OMB completed populating the eventbacklog, consumer traffic was restarted. During measurements with backlog draining, we observed peak consumer throughput of over 20GBps across all topics. The combined throughput to the NFS volume storing the OMB log data approached ~30GBps.

Sizing guidance

Amazon Web Services offers a sizing guide for Kafka cluster sizing and scaling.

This sizing provides a useful formula for determining storage throughput requirements for your Kafka cluster:

For an aggregated throughput produced into the cluster of tcluster with a replication factor of r, the throughput received by the broker storage is as follows:

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

This can be simplified even further:

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

Using this formula allows you to select the appropriate ONTAP platform for your Kafka hot tier needs.

The following table explains the anticipated producer throughput for the A900 with different replication factors:

Replication factor	Producer throughput (GPps)
3 (measured)	3.4
2	5.1
1	10.2

Next: Conclusion.

Conclusion

Previous: Performance overview and validation with AFF on-premises.

The NetApp solution for the silly rename problem provides a simple, inexpensive, and centrally managed form of storage for workloads that were previously incompatible with NFS. This new paradigm enables customers to create more manageable Kafka clusters that are easier to migrate and mirror for the purpose of disaster recovery and data protection. We have also seen that NFS provides additional benefits such as reduced CPU utilization and a faster recovery time, dramatically improved storage efficiency, and better performance through NetApp ONTAP.

Next: Where to find additional information.

Where to find additional information

Previous: Conclusion.

To learn more about the information that is described in this document, review the following documents and/or websites:

- What is Apache Kafka?

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

- What is silly rename?

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

- ONATP is read for streaming applications.

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

- Silly- rename issue with Kafka.

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

- NetApp product documentation

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

- What is NFS?

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

- What is Kafka partition reassignment?

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

- What is the OpenMessaging Benchmark?

  https://openmessaging.cloud/

- How do you migrate a Kafka broker?

  https://medium.com/@sanchitbansal26/how-to-migrate-kafka-cluster-with-no-downtime-58c216129058
- How do you monitor Kafka broker with Prometheus?

  https://www.confluent.io/blog/monitor-kafka-clusters-with-prometheus-grafana-and-confluent/
- Managed Platform for Apache Kafka

  https://www.instaclustr.com/platform/managed-apache-kafka/
- Support for Apache Kafka

  https://www.instaclustr.com/support-solutions/kafka-support/
- Consulting services for Apache Kafka

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

Version history

Version	Date	Document version history
Version 1.0	March 2023	Initial release.

Copyright information

Trademark information