Why NetApp NFS for Kafka workloads?

NetApp Solutions

NetApp
October 20, 2023

Table of Contents

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.

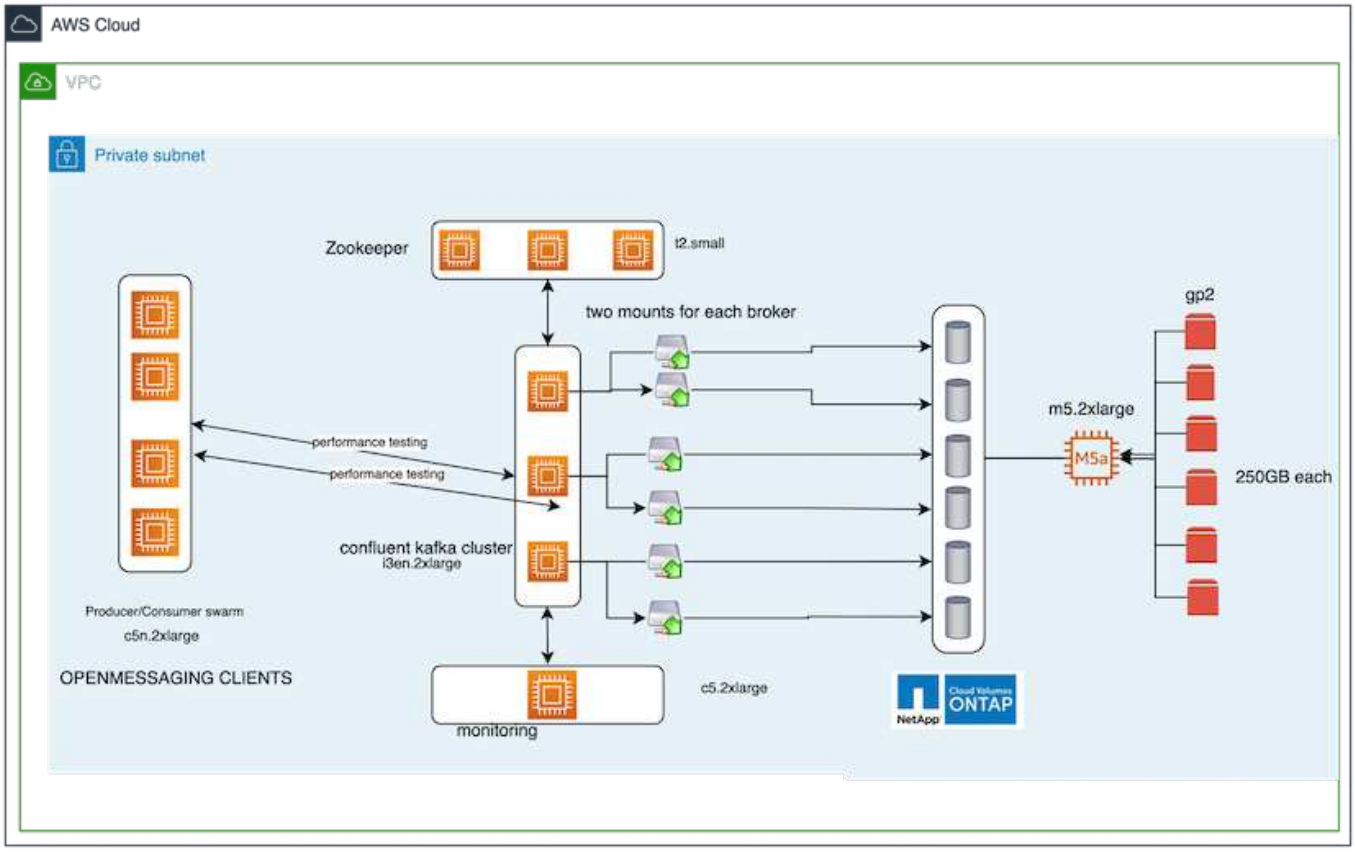
Architectural setup

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.

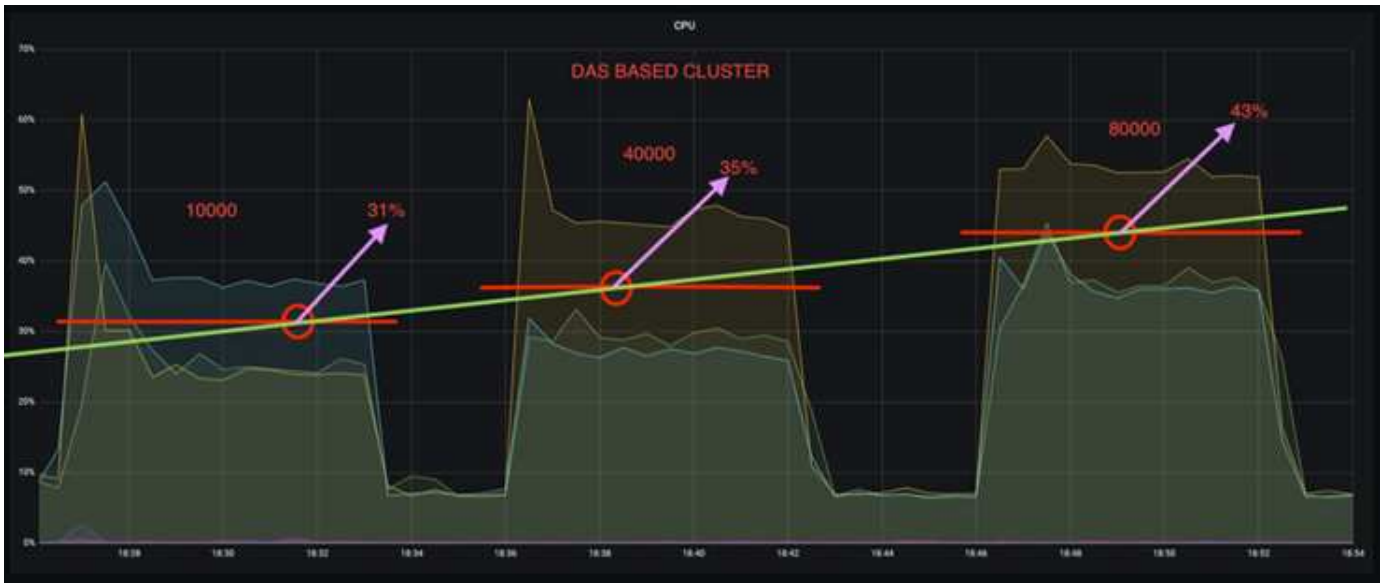
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

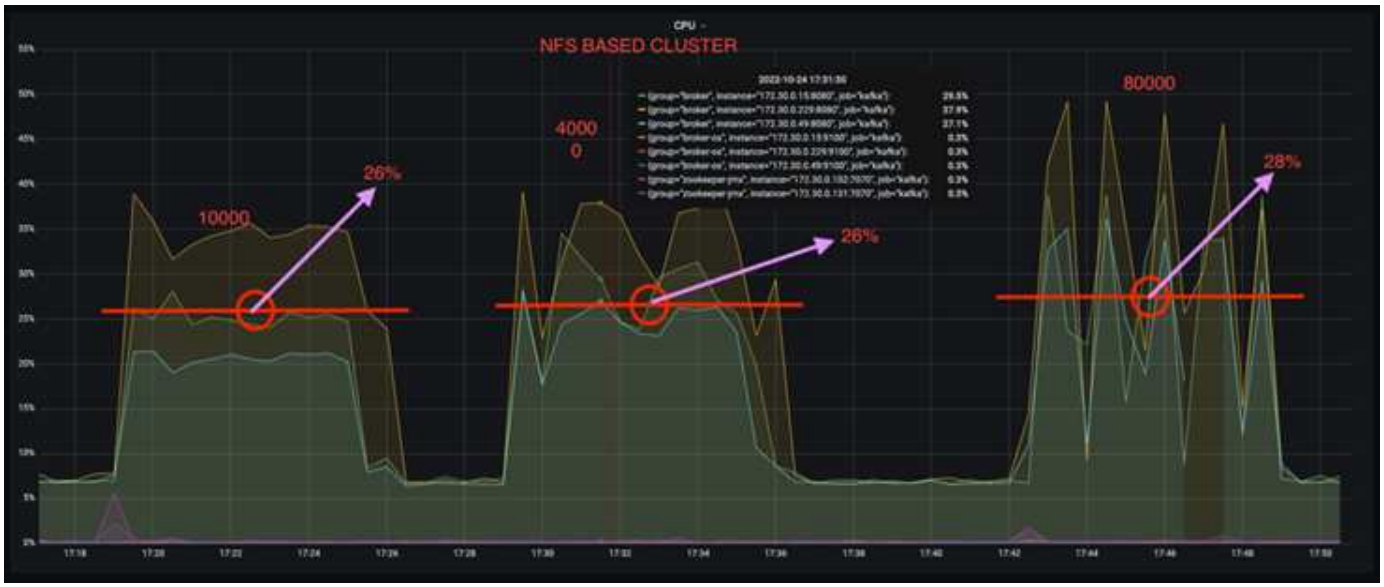
3. The produce rate configuration was increased in four iterations, and CPU utilization was recorded with Grafana. The produce rate was set to the following levels:
    - 10,000
    - 40,000
    - 80,000
    - 100,000

Observation

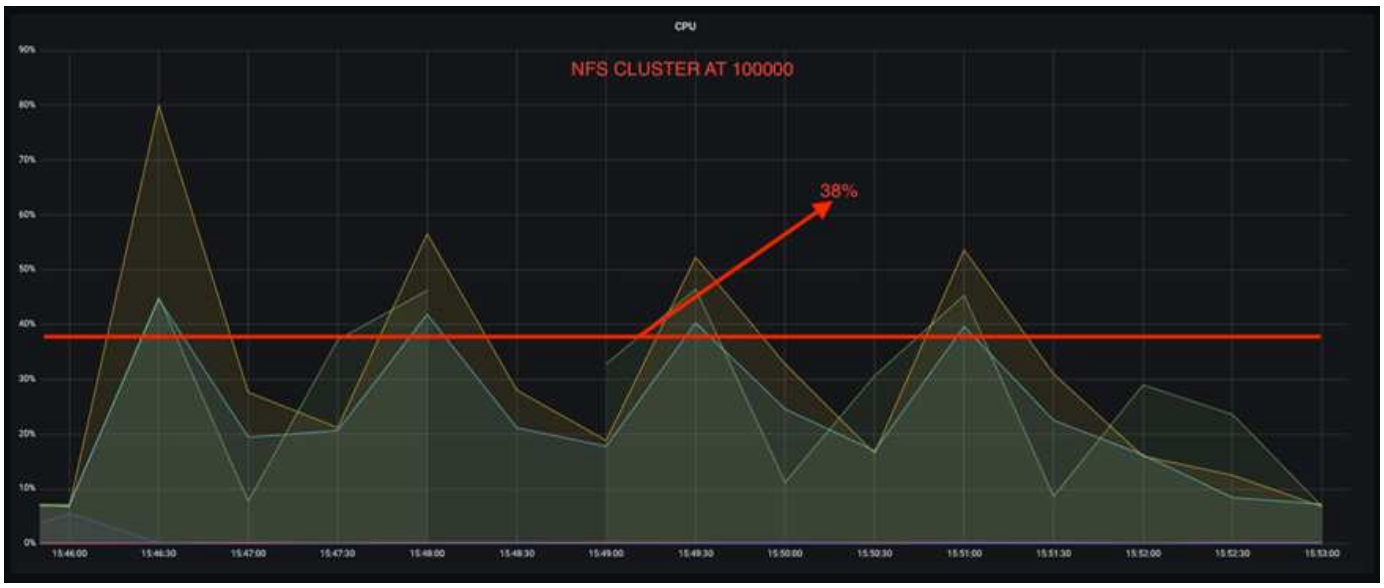
There are two primary benefits of using NetApp NFS storage with Kafka:

- **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

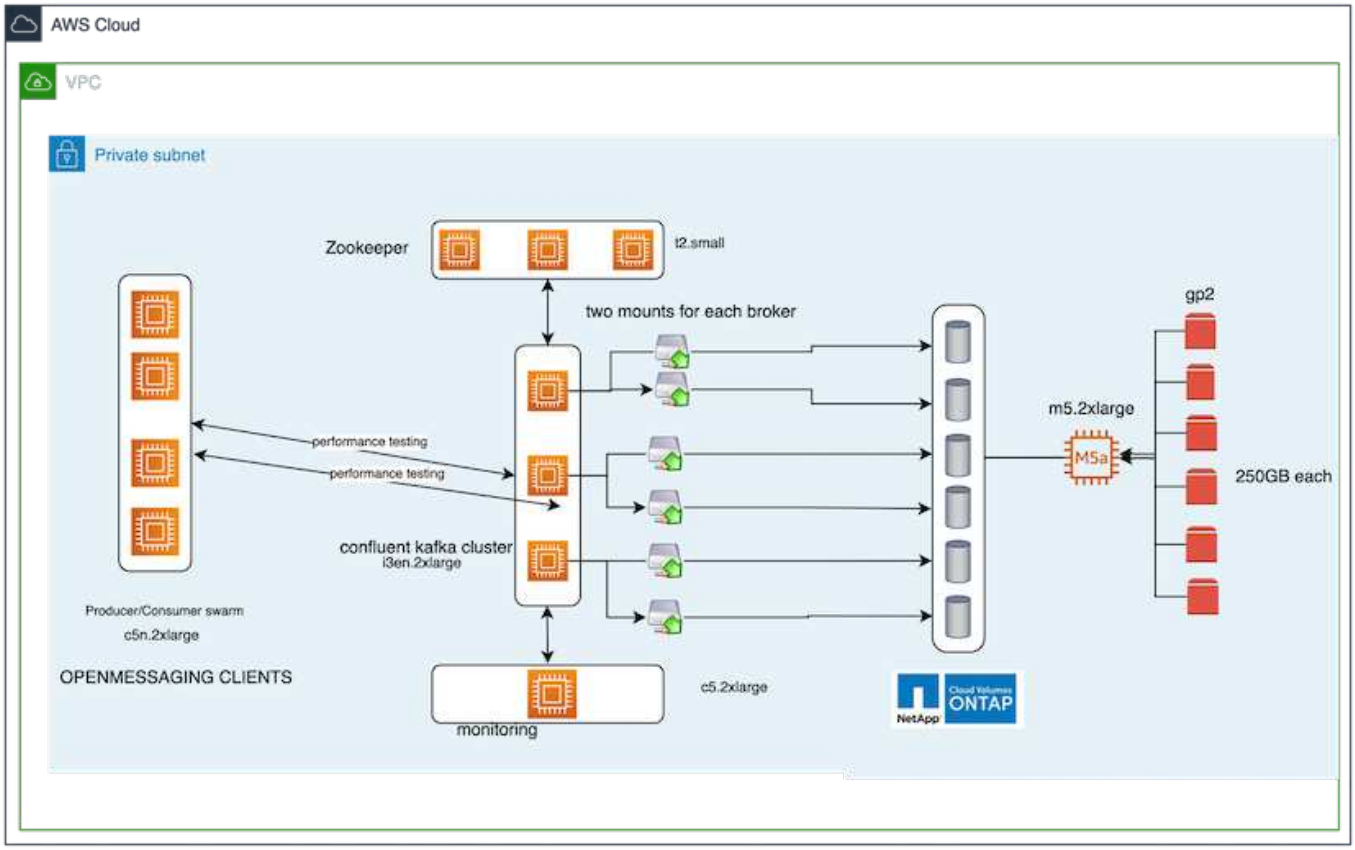
We discovered that Kafka brokers recover faster when they are using shared NetApp NFS storage. When a broker crashes in a Kafka cluster, this broker can be replaced by a healthy broker with a same broker ID. Upon performing this test case, we found that, in the case of a DAS-based Kafka cluster, the 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

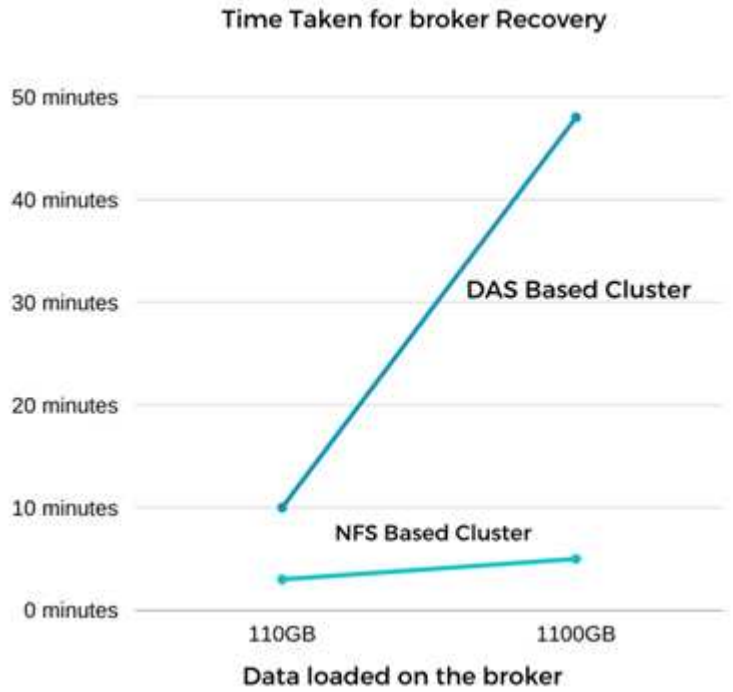
1. The backup node was started at 09:39:17,213. The starting log entry is highlighted below.

```
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

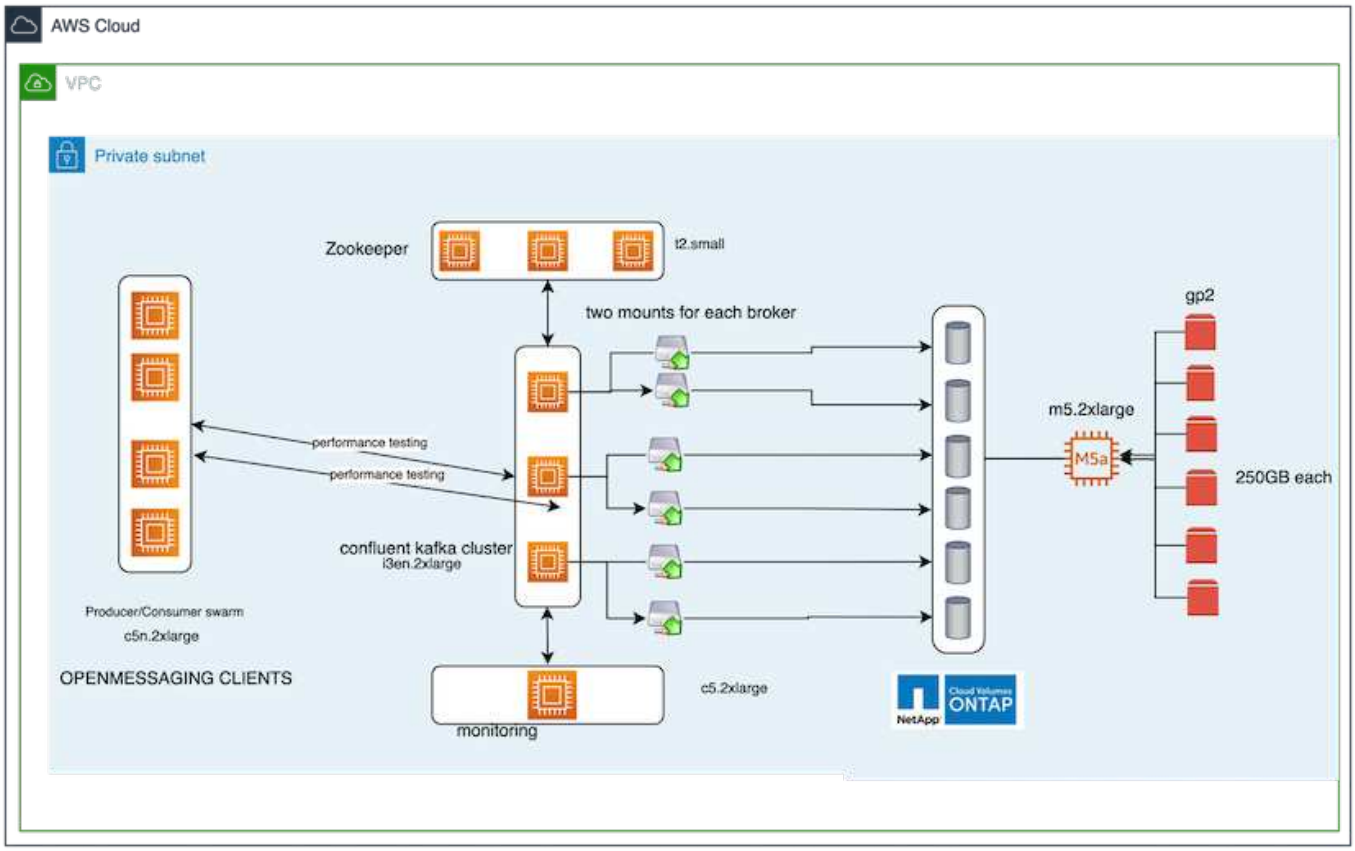
Because the storage layer of the Kafka cluster was provisioned through NetApp ONTAP, we got all the storage efficiency capabilities of ONTAP. This was tested by generating a significant amount of data on a Kafka cluster with NFS storage provisioned on Cloud Volumes ONTAP. We could see that there was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

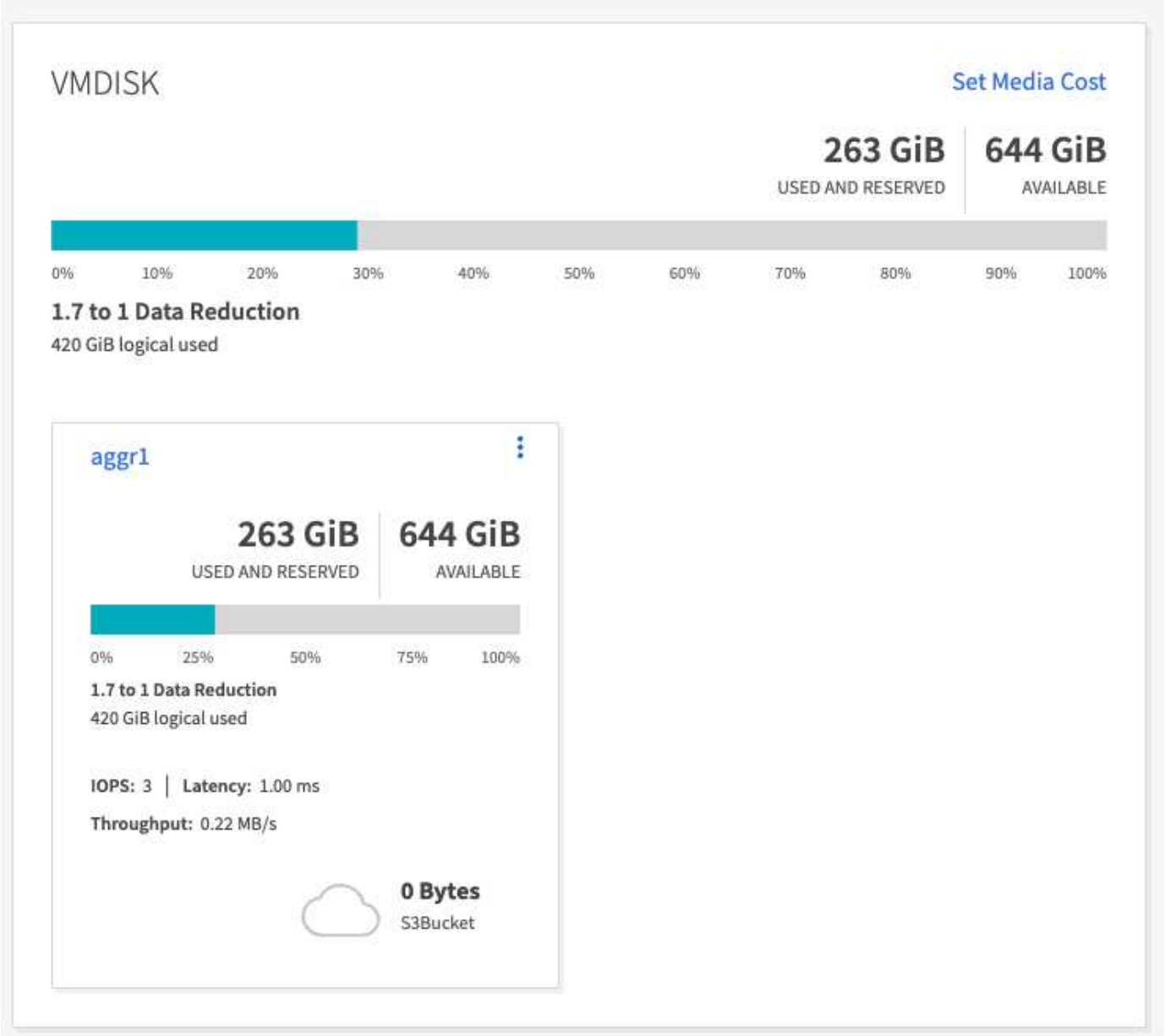
Compression was switched off on the producer's end, thus enabling producers to generate high throughput. Storage efficiency was instead handled by the compute layer.

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.

Copyright information

Trademark information