# NetApp NFS 스토리지를 사용한 Apache Kafka 워크로드

NetApp artificial intelligence solutions

NetApp
February 12, 2026

# 목차

# NetApp NFS 스토리지를 사용한 Apache Kafka 워크로드

## TR-4947: NetApp NFS 스토리지를 사용한 Apache Kafka 워크로드 - 기능 검증 및 성능

Shantanu Chakole, Karthikeyan Nagalingam 및 Joe Scott, NetApp

카프카는 대량의 메시지 데이터를 수용할 수 있는 강력한 큐를 갖춘 분산형 게시-구독 메시징 시스템입니다. Kafka를 사용하면 애플리케이션이 매우 빠르게 주제에 데이터를 쓰고 읽을 수 있습니다. Kafka는 내결함성과 확장성이 뛰어나서 빅데이터 분야에서 많은 데이터 스트림을 매우 빠르게 수집하고 이동하는 안정적인 방법으로 자주 사용됩니다. 사용 사례로는 스트림 처리, 웹사이트 활동 추적, 메트릭 수집 및 모니터링, 로그 집계, 실시간 분석 등이 있습니다.

NFS에서 일반적인 Kafka 작업은 잘 작동하지만, NFS에서 실행되는 Kafka 클러스터의 크기를 조정하거나 다시 분할하는 동안 어리석은 이름 바꾸기 문제로 인해 애플리케이션이 충돌합니다. 이는 부하 분산이나 유지 관리를 위해 카프카 클러스터의 크기를 조정하거나 다시 분할해야 하기 때문에 중요한 문제입니다. 추가 세부 정보를 찾을 수 있습니다 "여기" .

이 문서에서는 다음 주제에 대해 설명합니다.

- 어리석은 이름 바꾸기 문제와 솔루션 검증
- I/O 대기 시간을 줄이기 위해 CPU 사용률을 줄이세요
- 더 빠른 Kafka 브로커 복구 시간
- 클라우드 및 온프레미스에서의 성능

### Kafka 워크로드에 NFS 스토리지를 사용하는 이유는 무엇입니까?

프로덕션 애플리케이션의 Kafka 워크로드는 애플리케이션 간에 엄청난 양의 데이터를 스트리밍할 수 있습니다. 이 데이터는 Kafka 클러스터의 Kafka 브로커 노드에 보관되고 저장됩니다. 카프카는 가용성과 병렬 처리로도 유명한데, 이는 주제를 파티션으로 나누고 해당 파티션을 클러스터 전체에 복제함으로써 달성됩니다. 결국 이는 카프카 클러스터를 통과하는 엄청난 양의 데이터 크기가 일반적으로 몇 배로 증가한다는 것을 의미합니다. NFS를 사용하면 브로커 수가 변경되어도 데이터를 매우 빠르고 쉽게 재조정할 수 있습니다. 대규모 환경에서 브로커 수가 변경될 때 DAS 전체에서 데이터를 재조정하는 작업은 매우 많은 시간이 소요되며, 대부분의 Kafka 환경에서는 브로커 수가 자주 변경됩니다.

기타 혜택은 다음과 같습니다.

- 성숙함. NFS는 성숙한 프로토콜이므로 구현, 보안 및 사용의 대부분 측면이 잘 이해되어 있습니다.
- 열려 있는. NFS는 개방형 프로토콜이며, 무료 개방형 네트워크 프로토콜로서 지속적인 개발이 인터넷 사양에 문서화되어 있습니다.
- 비용 효율적입니다. NFS는 기존 네트워크 인프라를 활용하기 때문에 설정이 쉬운 저비용 네트워크 파일 공유 솔루션입니다.
- 중앙에서 관리됨. NFS를 중앙에서 관리하면 개별 사용자 시스템에서 추가 소프트웨어와 디스크 공간에 대한 필요성이 줄어듭니다.
- 분배됨. NFS는 분산 파일 시스템으로 사용할 수 있으므로 이동식 미디어 저장 장치의 필요성이 줄어듭니다.

### Kafka 워크로드에 NetApp 하는 이유는 무엇입니까?

NetApp NFS 구현은 프로토콜에 대한 황금 표준으로 간주되며 수많은 기업 NAS 환경에서 사용됩니다. NetApp 의 신뢰성 외에도 다음과 같은 이점이 있습니다.

- 신뢰성과 효율성
- 확장성 및 성능
- 고가용성( NetApp ONTAP 클러스터의 HA 파트너)
- 데이터 보호
  - 재해 복구**(NetApp SnapMirror).** 사이트가 다운되거나 다른 사이트에서 바로 시작해서 중단했던 부분부터 계속하고 싶은 경우.
  - 스토리지 시스템의 관리 용이성( NetApp OnCommand 사용한 관리 및 운영).
  - 부하 분산**.** 클러스터를 사용하면 다양한 노드에 호스팅된 데이터 LIF에서 다양한 볼륨에 액세스할 수 있습니다.
  - 중단 없는 운영**.** LIF 또는 볼륨 이동은 NFS 클라이언트에 투명하게 처리됩니다.

# NFS에서 Kafka 워크로드로의 어리석은 이름 변경 문제에 대한 NetApp 솔루션

카프카는 기본 파일 시스템이 POSIX 호환이라는 가정 하에 구축되었습니다(예: XFS 또는 Ext4). 카프카 리소스 재조정은 애플리케이션이 파일을 계속 사용하는 동안 해당 파일을 제거합니다. POSIX 호환 파일 시스템에서는 unlink가 계속 진행됩니다. 하지만 파일에 대한 모든 참조가 사라진 후에야 파일을 제거합니다. 기본 파일 시스템이 네트워크에 연결되어 있으면 NFS 클라이언트가 연결 해제 호출을 가로채서 워크플로를 관리합니다. 연결 해제되는 파일에 보류 중인 열려 있는 작업이 있기 때문에 NFS 클라이언트는 NFS 서버로 이름 변경 요청을 보내고, 연결 해제된 파일을 마지막으로 닫을 때 이름이 변경된 파일에 대한 제거 작업을 실행합니다. 이러한 동작은 일반적으로 NFS Silly Rename이라고 불리며, NFS 클라이언트에 의해 조정됩니다.

NFSv3 서버의 저장소를 사용하는 모든 Kafka 브로커는 이러한 동작으로 인해 문제에 직면합니다. 그러나 NFSv4.x 프로토콜에는 서버가 열려 있지만 연결되지 않은 파일에 대한 책임을 맡을 수 있도록 하여 이 문제를 해결하는 기능이 있습니다. 이 옵션 기능을 지원하는 NFS 서버는 파일을 열 때 NFS 클라이언트에 소유권 기능을 전달합니다. 그러면 NFS 클라이언트는 보류 중인 열기가 있는 경우 연결 해제 관리를 중단하고 서버가 흐름을 관리하도록 허용합니다. NFSv4 사양은 구현에 대한 지침을 제공하지만 지금까지 이 선택적 기능을 지원하는 알려진 NFS 서버 구현은 없었습니다.

다음은 어리석은 이름 바꾸기 문제를 해결하기 위해 NFS 서버와 NFS 클라이언트에 필요한 변경 사항입니다.

- **NFS** 클라이언트**(Linux)**에 대한 변경 사항**.** 파일을 여는 시점에 NFS 서버는 플래그로 응답하여 열린 파일의 연결을 해제할 수 있는 기능을 나타냅니다. NFS 클라이언트 측 변경을 통해 NFS 서버가 플래그가 있는 경우 연결 해제를 처리할 수 있습니다. NetApp 이러한 변경 사항으로 오픈 소스 Linux NFS 클라이언트를 업데이트했습니다. 업데이트된 NFS 클라이언트는 이제 RHEL8.7 및 RHEL9.1에서 일반적으로 사용할 수 있습니다.
- **NFS** 서버 변경**.** NFS 서버는 오픈을 추적합니다. 기존에 열려 있는 파일의 연결을 해제하는 작업은 이제 POSIX 의미 체계와 일치하도록 서버에서 관리됩니다. 마지막으로 열린 파일이 닫히면 NFS 서버는 파일의 실제 제거를 시작하여 어리석은 이름 바꾸기 과정을 피합니다. ONTAP NFS 서버는 최신 릴리스인 ONTAP 9.12.1에서 이 기능을 구현했습니다.

NFS 클라이언트와 서버에 위와 같은 변경 사항을 적용하면 Kafka는 네트워크에 연결된 NFS 스토리지의 모든 이점을 안전하게 얻을 수 있습니다.

# 기능 검증 - 어리석은 이름 바꾸기 수정

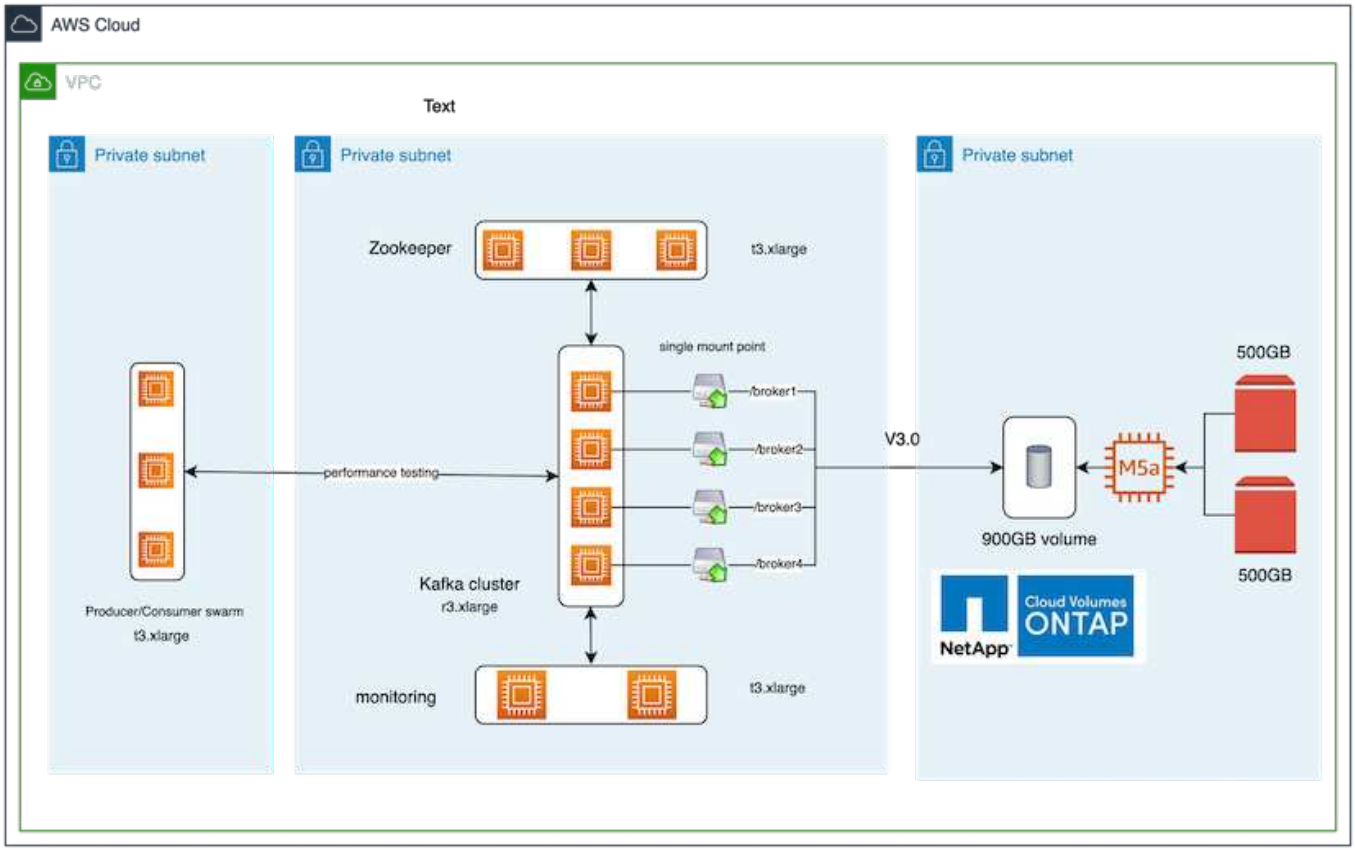
기능적 검증을 위해, 저장소로 NFSv3를 마운트한 Kafka 클러스터는 파티션 재분배와 같은 Kafka 작업을 수행하지 못하는 반면, 수정 사항을 적용한 NFSv4에 마운트된 다른 클러스터는 아무런 중단 없이 동일한 작업을 수행할 수 있음을 보여주었습니다.

## 검증 설정

설정은 AWS에서 실행됩니다. 다음 표는 검증에 사용된 다양한 플랫폼 구성 요소와 환경 구성을 보여줍니다.

| 플랫폼 구성 요소 | 환경 구성 |
| --- | --- |
| Confluent 플랫폼 버전 7.2.1 | • 3 x 동물원 관리인 – t3.xlarge<br>• 4 x 브로커 서버 – r3.xlarge<br>• 1 x Grafana – t3.xlarge<br>• 1 x 제어 센터 – t3.xlarge<br>• 3 x 생산자/소비자 |
| 모든 노드의 운영 체제 | RHEL8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

다음 그림은 이 솔루션의 아키텍처 구성을 보여줍니다.




## 건축 흐름

- 계산합니다. 우리는 전용 서버에서 실행되는 3노드 Zookeeper 앙상블과 함께 4노드 Kafka 클러스터를 사용했습니다.
- 모니터링. 우리는 Prometheus-Grafana 조합을 위해 두 개의 노드를 사용했습니다.
- 작업량. 워크로드를 생성하기 위해 Kafka 클러스터에서 워크로드를 생산하고 소비할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- 저장. 우리는 두 개의 500GB GP2 AWS-EBS 볼륨이 인스턴스에 연결된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스를 사용했습니다. 이러한 볼륨은 LIF를 통해 단일 NFSv4.1 볼륨으로 Kafka 클러스터에 노출되었습니다.

모든 서버에 대해 Kafka의 기본 속성이 선택되었습니다. 동물원 관리인 떼에도 똑같은 일이 이루어졌습니다.

## 테스트 방법론

1. 업데이트 `-is-preserve-unlink-enabled true` 카프카 볼륨에 다음과 같이 표시됩니다.

```
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. 두 개의 유사한 Kafka 클러스터가 다음과 같은 차이점을 가지고 생성되었습니다.
   - 클러스터 **1.** 프로덕션에 바로 사용할 수 있는 ONTAP 버전 9.12.1을 실행하는 백엔드 NFS v4.1 서버는 NetApp CVO 인스턴스에 의해 호스팅되었습니다. 브로커에 RHEL 8.7/RHEL 9.1이 설치되었습니다.
   - 클러스터 **2.** 백엔드 NFS 서버는 수동으로 생성된 일반 Linux NFSv3 서버였습니다.
3. 두 Kafka 클러스터 모두에 데모 주제가 생성되었습니다.

   클러스터 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:
```

   클러스터 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. 두 클러스터 모두의 새로 생성된 주제에 데이터가 로드되었습니다. 이 작업은 기본 Kafka 패키지에 포함된 producer-perf-test 툴킷을 사용하여 수행되었습니다.

```
./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. Telnet을 사용하여 각 클러스터의 브로커-1에 대한 상태 점검을 수행했습니다.
   - 텔넷 `172.30.0.160 9092`
   - 텔넷 `172.30.0.198 9092`

     다음 스크린샷에는 두 클러스터의 브로커에 대한 성공적인 상태 검사가 표시됩니다.

```
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. NFSv3 스토리지 볼륨을 사용하는 Kafka 클러스터가 충돌하는 오류 조건을 트리거하기 위해 두 클러스터 모두에서 파티션 재할당 프로세스를 시작했습니다. 파티션 재할당은 다음을 사용하여 수행되었습니다. `kafka-reassign-partitions.sh` . 자세한 과정은 다음과 같습니다.

   a. Kafka 클러스터의 주제에 대한 파티션을 재할당하기 위해 제안된 재할당 구성 JSON을 생성했습니다(이 작업은 두 클러스터 모두에서 수행되었습니다).

   ```
   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. 생성된 재할당 JSON은 다음에 저장되었습니다. `/tmp/reassignment- file.json` .

   c. 실제 파티션 재할당 프로세스는 다음 명령에 의해 트리거되었습니다.

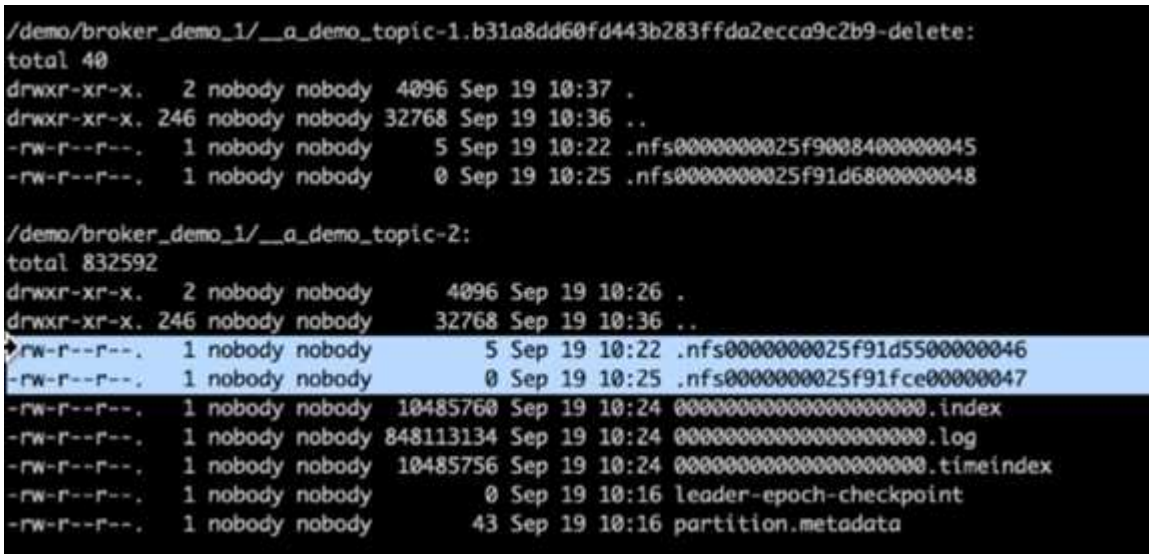
   ```
   kafka-reassign-partitions --bootstrap
   -server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:9092,172.30.
   0.204:9092 --reassignment-json-file /tmp/reassignment-file.json
   -execute
   ```

7. 재할당이 완료된 후 몇 분 후에 브로커에서 실시한 또 다른 상태 점검에서 NFSv3 스토리지 볼륨을 사용하는 클러스터가 어리석은 이름 바꾸기 문제에 부딪혀 충돌했지만, 수정된 NetApp ONTAP NFSv4.1 스토리지 볼륨을 사용하는 클러스터 1은 중단 없이 작업을 계속 진행한 것으로 나타났습니다.

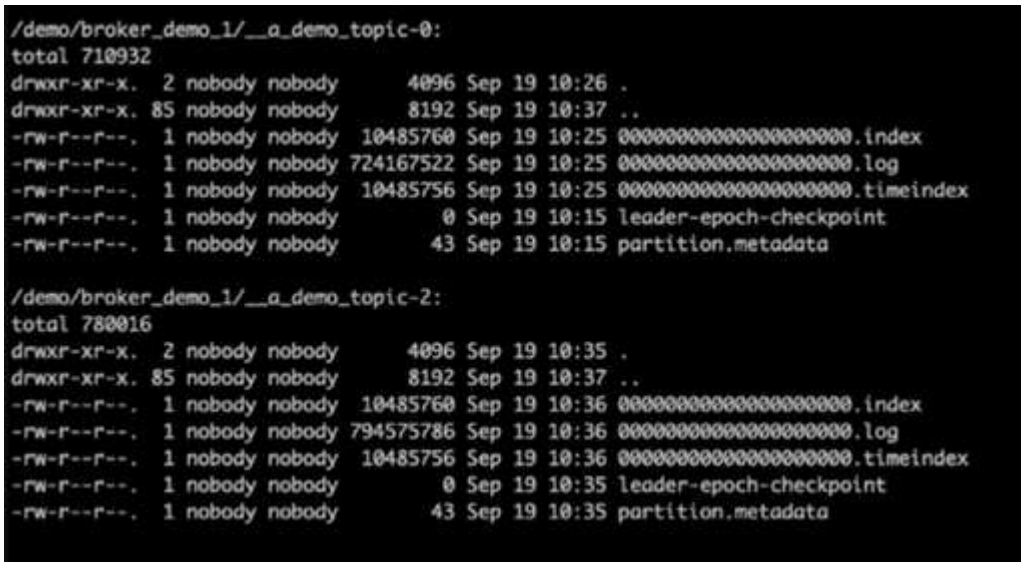
```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

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-1은 활성화되어 있습니다.
- Cluster2-broker-1이 죽었습니다.

8. Kafka 로그 디렉터리를 확인한 결과, NetApp ONTAP NFSv4.1 스토리지 볼륨을 수정하여 사용하는 클러스터 1은 파티션이 깔끔하게 할당되었지만, 일반 NFSv3 스토리지를 사용하는 클러스터 2는 어리석은 이름 변경 문제로 인해 파티션이 깔끔하게 할당되지 않아 충돌이 발생한 것이 분명했습니다. 다음 그림은 클러스터 2의 파티션 재조정을 보여주는데, 이로 인해 NFSv3 스토리지에서 어리석은 이름 변경 문제가 발생했습니다.




다음 그림은 NetApp NFSv4.1 스토리지를 사용하여 클러스터 1의 깔끔한 파티션 재조정을 보여줍니다.




# Kafka 워크로드에 NetApp NFS를 사용해야 하는 이유는 무엇입니까?

이제 Kafka를 사용하여 NFS 스토리지의 어리석은 이름 변경 문제에 대한 솔루션이 있으므로 Kafka 워크로드에 NetApp ONTAP 스토리지를 활용하는 강력한 배포를 만들 수 있습니다. 이렇게 하면 운영 오버헤드가 크게 줄어들 뿐만 아니라, Kafka 클러스터에 다음과 같은 이점도 제공됩니다.

- **Kafka** 브로커의 **CPU** 사용률이 감소했습니다**.** 분산된 NetApp ONTAP 스토리지를 사용하면 디스크 I/O 작업이 브로커에서 분리되어 CPU 사용량이 줄어듭니다.
- 더 빠른 브로커 복구 시간**.** 분산된 NetApp ONTAP 스토리지는 Kafka 브로커 노드 전체에서 공유되므로, 기존 Kafka 배포에 비해 데이터를 다시 빌드하지 않고도 새로운 컴퓨팅 인스턴스가 언제든지 잘못된 브로커를 대체할 수 있는 시간이 훨씬 단축됩니다.
- 저장 효율성**.** 이제 애플리케이션의 스토리지 계층이 NetApp ONTAP 통해 프로비저닝되므로 고객은 인라인 데이터 압축, 중복 제거, 압축과 같은 ONTAP 이 제공하는 모든 스토리지 효율성 이점을 활용할 수 있습니다.

이러한 이점은 이 섹션에서 자세히 설명하는 테스트 사례에서 테스트되고 검증되었습니다.

## Kafka 브로커의 CPU 사용률 감소

기술 사양은 동일하지만 저장 기술이 다른 두 개의 별도의 Kafka 클러스터에서 유사한 작업 부하를 실행했을 때, 전반적인 CPU 사용률이 DAS 대응 제품보다 낮다는 것을 발견했습니다. Kafka 클러스터가 ONTAP 스토리지를 사용하면 전반적인 CPU 사용률이 낮을 뿐만 아니라 CPU 사용률의 증가 폭도 DAS 기반 Kafka 클러스터보다 완만했습니다.

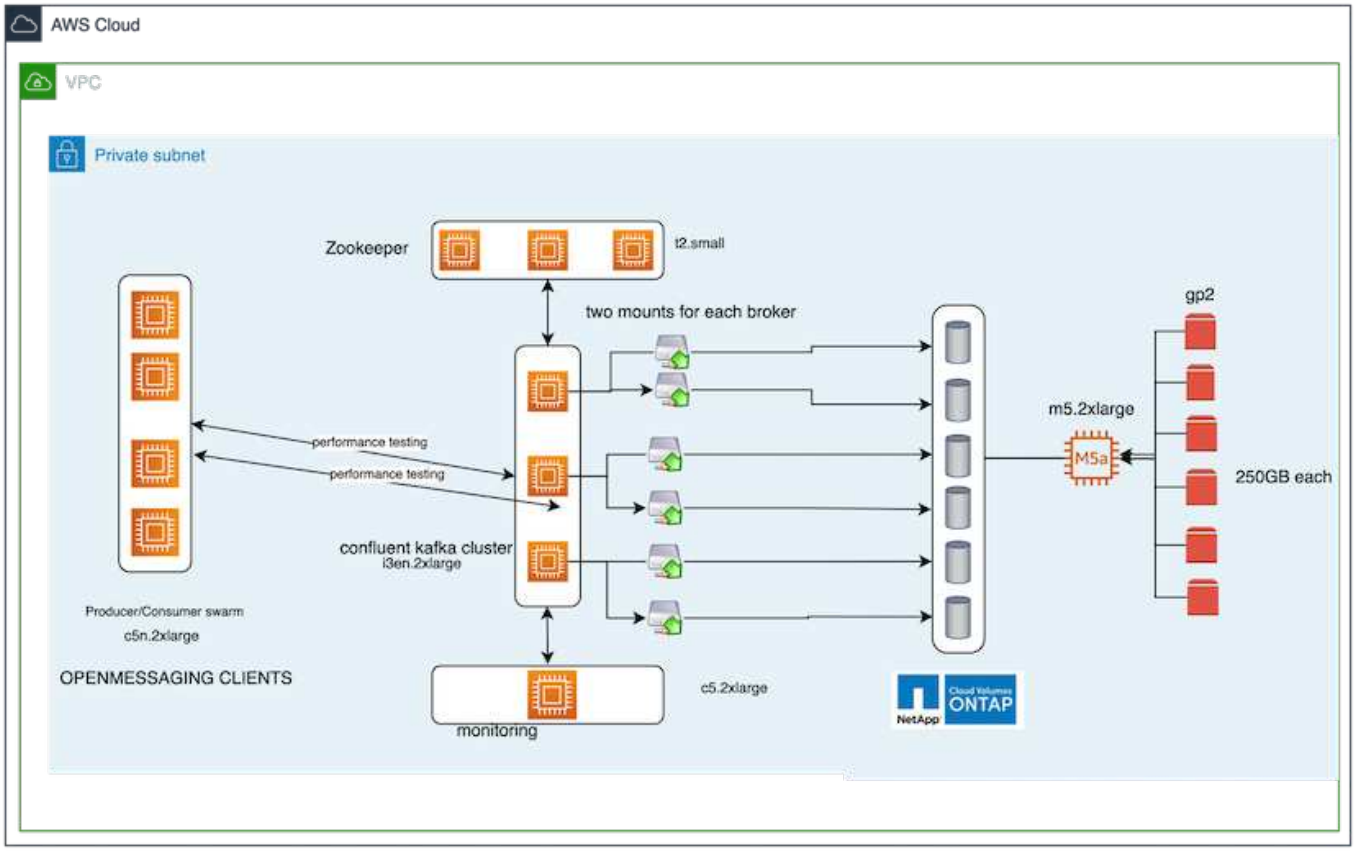
### 건축적 설정

다음 표는 CPU 사용률을 낮추는 데 사용된 환경 구성을 보여줍니다.

| 플랫폼 구성 요소 | 환경 구성 |
| --- | --- |
| Kafka 3.2.3 벤치마킹 도구: OpenMessaging | • 3 x 동물원 관리인 – t2.small<br>• 3개의 브로커 서버 - i3en.2xlarge<br>• 1 x 그라파나 – c5n.2xlarge<br>• 4 x 생산자/소비자 — c5n.2xlarge |
| 모든 노드의 운영 체제 | RHEL 8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

### 벤치마킹 도구

이 테스트 케이스에서 사용된 벤치마킹 도구는 다음과 같습니다. "오픈메시징" 뼈대. OpenMessaging은 공급업체와 언어에 구애받지 않습니다. 금융, 전자상거래, IoT, 빅데이터에 대한 산업 가이드라인을 제공하며, 이기종 시스템과 플랫폼에서 메시징 및 스트리밍 애플리케이션을 개발하는 데 도움이 됩니다. 다음 그림은 OpenMessaging 클라이언트와 Kafka 클러스터의 상호작용을 보여줍니다.




- 계산합니다. 우리는 전용 서버에서 실행되는 3노드 Zookeeper 앙상블과 함께 3노드 Kafka 클러스터를 사용했습니다. 각 브로커는 전용 LIF를 통해 NetApp CVO 인스턴스의 단일 볼륨에 대한 두 개의 NFSv4.1 마운트 포인트를 가졌습니다.
- 모니터링. 우리는 Prometheus-Grafana 조합을 위해 두 개의 노드를 사용했습니다. 워크로드를 생성하기 위해 Kafka 클러스터에서 워크로드를 생산하고 소비할 수 있는 별도의 3노드 클러스터가 있습니다.
- 저장. 6개의 250GB GP2 AWS-EBS 볼륨이 인스턴스에 마운트된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스를 사용했습니다. 이러한 볼륨은 전용 LIF를 통해 6개의 NFSv4.1 볼륨으로 Kafka 클러스터에 노출되었습니다.
- 구성. 이 테스트 사례에서 구성 가능한 두 가지 요소는 Kafka 브로커와 OpenMessaging 워크로드였습니다.
  - 브로커 구성. 다음은 Kafka 브로커에 대해 선택된 사양입니다. 아래에 강조된 것처럼 모든 측정에 대해 복제 계수 3을 사용했습니다.

```
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** 벤치마크**(OMB)** 워크로드 구성**.** 다음과 같은 사양이 제공되었습니다. 우리는 아래에 강조된 목표 생산자 요율을 지정했습니다.

```
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
```

테스트 방법론

1. 두 개의 유사한 클러스터가 생성되었으며, 각각 고유한 벤치마킹 클러스터 군집이 있습니다.
   - 클러스터 **1.** NFS 기반 카프카 클러스터.
   - 클러스터 **2.** DAS 기반 카프카 클러스터.
2. OpenMessaging 명령을 사용하여 각 클러스터에서 유사한 작업 부하가 트리거되었습니다.

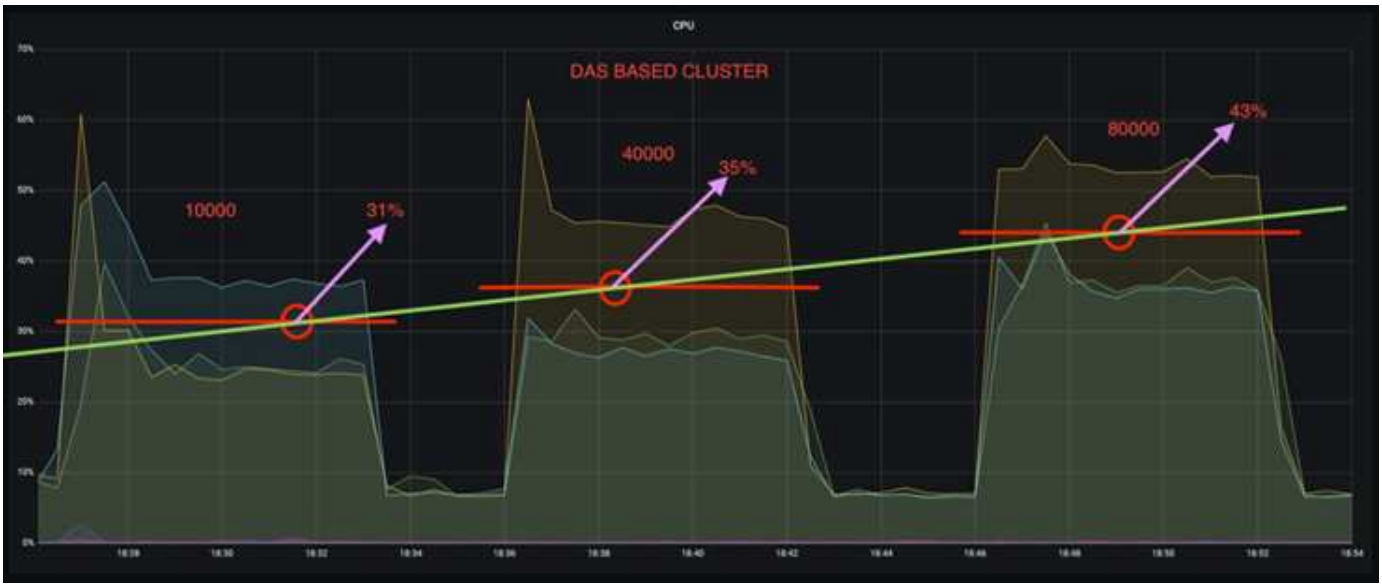
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

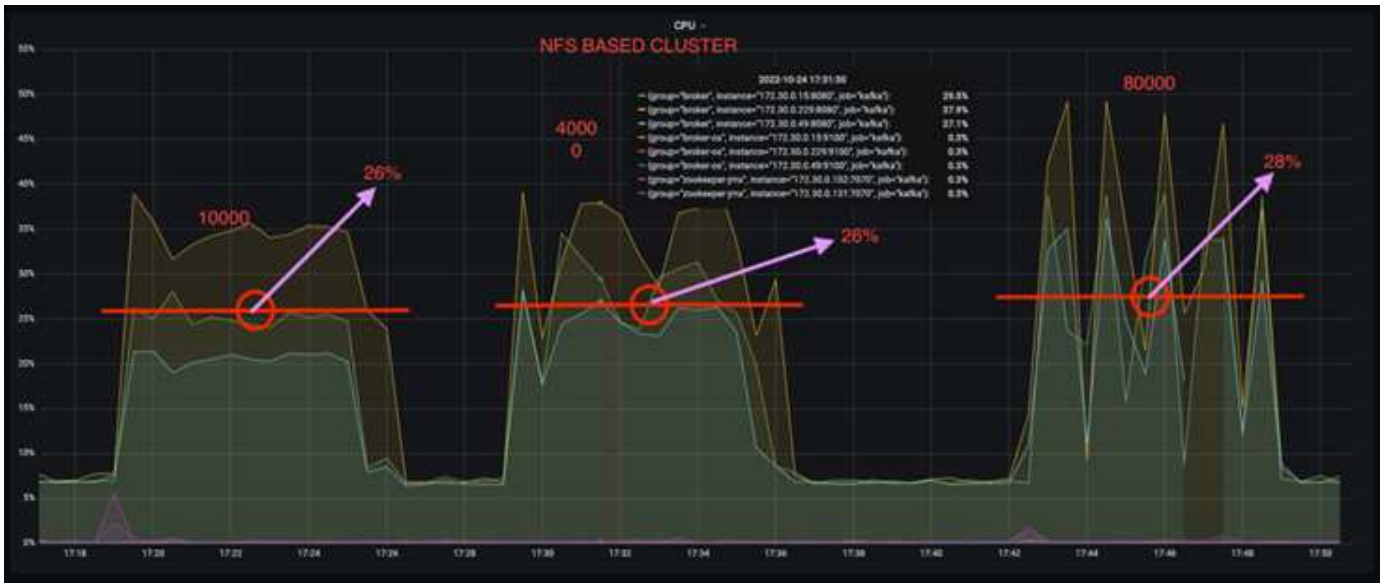
3. 생산율 구성은 4번의 반복을 거쳐 증가하였고, Grafana를 사용하여 CPU 사용률을 기록했습니다. 생산율은 다음 수준으로 설정되었습니다.
    - 10,000
    - 40,000
    - 80,000
    - 100,000

관찰

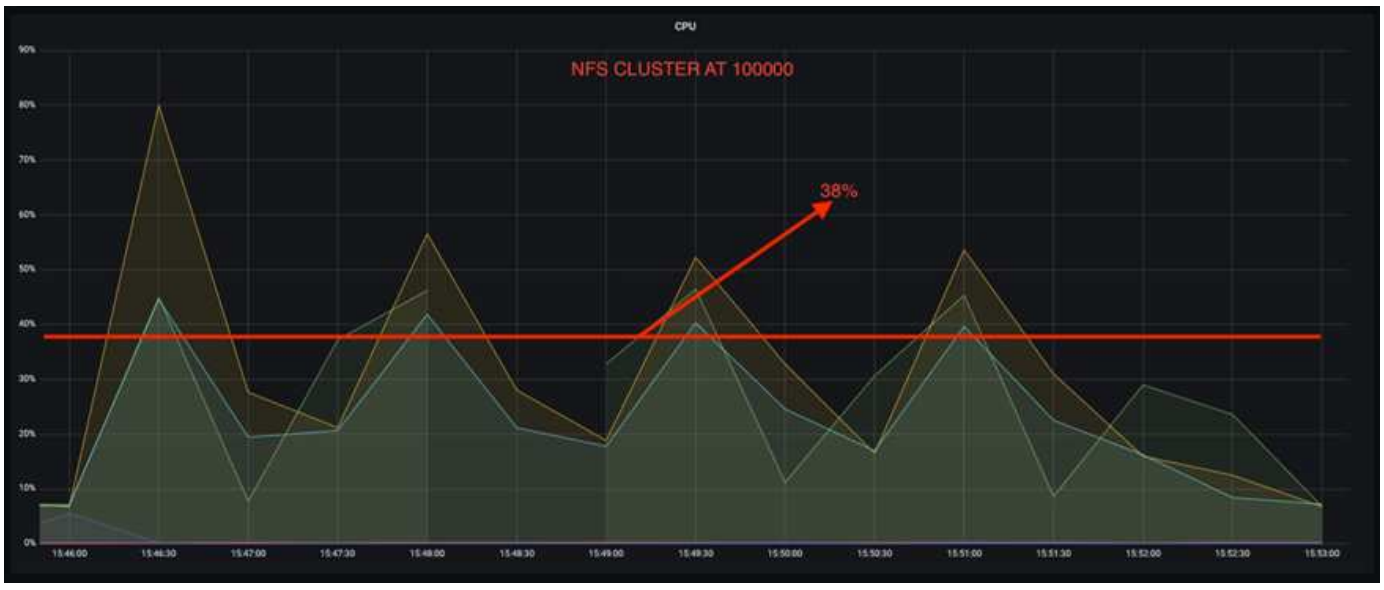
Kafka와 함께 NetApp NFS 스토리지를 사용하면 두 가지 주요 이점이 있습니다.

- **CPU** 사용량을 약 **1/3**까지 줄일 수 있습니다**.** 유사한 작업 부하에서 NFS의 전반적인 CPU 사용량은 DAS SSD보다 낮았습니다. 생성률이 낮을수록 절감 효과가 5%, 생성률이 높을수록 절감 효과가 32%에 달했습니다.
- 생산 속도가 높을수록 **CPU** 사용률 변동이 **3**배 감소합니다**.** 예상대로, 생산율이 증가함에 따라 CPU 활용도가 위쪽으로 증가했습니다. 그러나 DAS를 사용하는 Kafka 브로커의 CPU 사용률은 낮은 생성률의 경우 31%에서 높은 생성률의 경우 70%로 39% 증가했습니다. 하지만 NFS 스토리지 백엔드를 사용하면 CPU 사용률이 26%에서 38%로 12% 증가했습니다.






또한 100,000개의 메시지에서 DAS는 NFS 클러스터보다 CPU 사용률이 더 높습니다.

## 더 빠른 브로커 회복

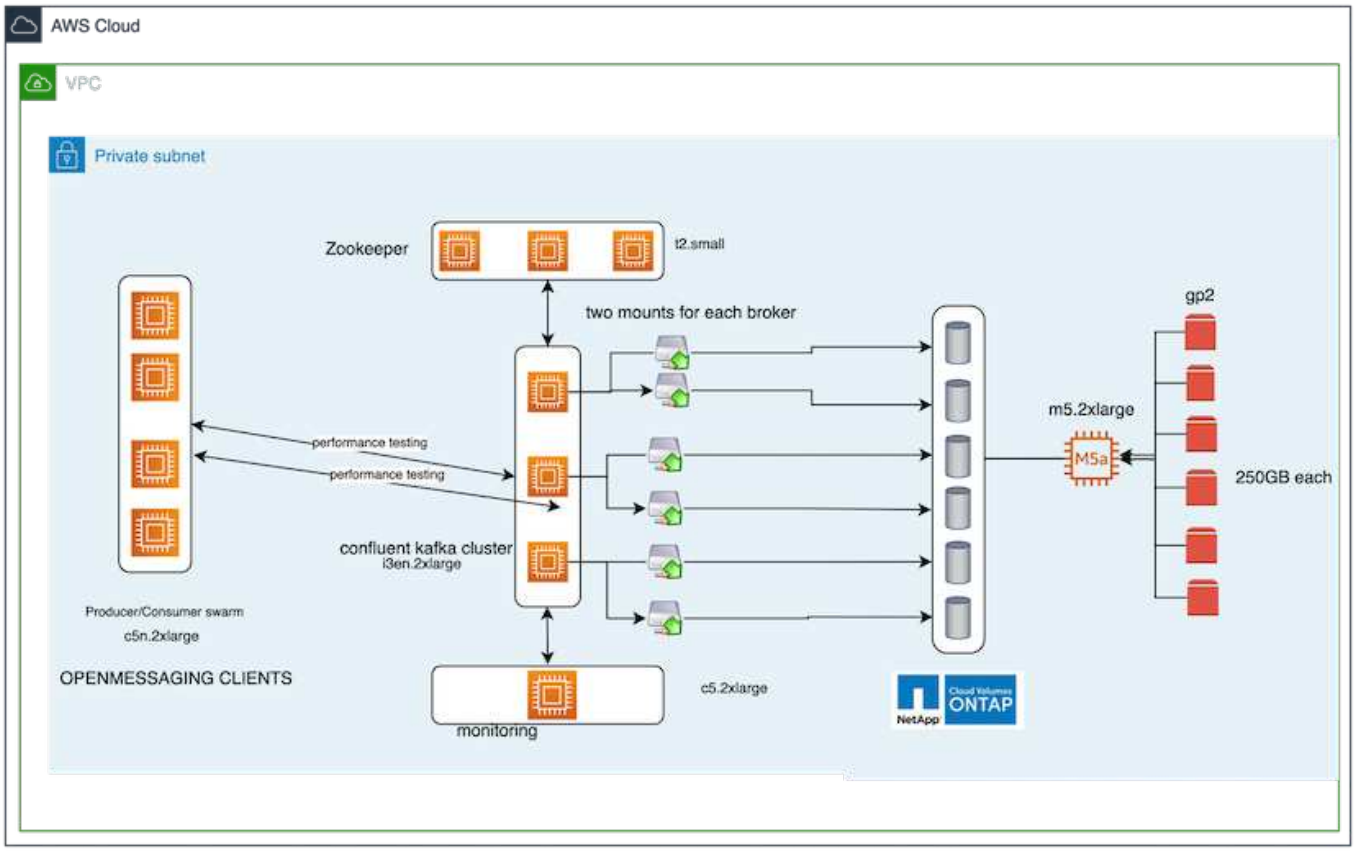
Kafka 브로커가 공유 NetApp NFS 스토리지를 사용할 경우 복구 속도가 더 빨라진다는 것을 발견했습니다. Kafka 클러스터에서 브로커가 충돌하는 경우, 이 브로커는 동일한 브로커 ID를 가진 정상적인 브로커로 교체될 수 있습니다. 이 테스트 사례를 수행한 결과, DAS 기반 Kafka 클러스터의 경우 클러스터가 새로 추가된 정상적인 브로커에서 데이터를 다시 구축하는데, 이는 시간이 많이 소요되는 작업이라는 것을 발견했습니다. NetApp NFS 기반 Kafka 클러스터의 경우, 교체 브로커는 이전 로그 디렉토리에서 데이터를 계속 읽고 훨씬 빠르게 복구합니다.

### 건축적 설정

다음 표는 NAS를 사용하는 Kafka 클러스터의 환경 구성을 보여줍니다.

| 플랫폼 구성 요소 | 환경 구성 |
| --- | --- |
| 카프카 3.2.3 | • 3 x 동물원 관리인 – t2.small<br>• 3개의 브로커 서버 - i3en.2xlarge<br>• 1 x 그라파나 – c5n.2xlarge<br>• 4 x 생산자/소비자 — c5n.2xlarge<br>• 1 x 백업 Kafka 노드 – i3en.2xlarge |
| 모든 노드의 운영 체제 | RHEL8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

다음 그림은 NAS 기반 Kafka 클러스터의 아키텍처를 보여줍니다.



- 계산합니다. 전용 서버에서 실행되는 3노드 Zookeeper 앙상블을 갖춘 3노드 Kafka 클러스터입니다. 각 브로커는 전용 LIF를 통해 NetApp CVO 인스턴스의 단일 볼륨에 대한 두 개의 NFS 마운트 지점을 갖습니다.
- 모니터링. Prometheus-Grafana 조합을 위한 두 개의 노드. 워크로드를 생성하기 위해 이 Kafka 클러스터에서 워크로드를 생성하고 소비할 수 있는 별도의 3노드 클러스터를 사용합니다.
- 저장. 6개의 250GB GP2 AWS-EBS 볼륨이 인스턴스에 마운트된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스입니다. 이러한 볼륨은 전용 LIF를 통해 6개의 NFS 볼륨으로 Kafka 클러스터에 노출됩니다.
- 브로커 구성. 이 테스트 사례에서 구성 가능한 요소 중 하나는 Kafka 브로커입니다. 다음은 Kafka 브로커에 대해 선택된 사양입니다. 그만큼 `replica.lag.time.mx.ms` 특정 노드가 ISR 목록에서 얼마나 빨리 제거되는지를 결정하기 때문에 높은 값으로 설정됩니다. 불량 노드와 정상 노드 사이를 전환할 때 해당 브로커 ID가 ISR 목록에서 제외되는 것을 원하지 않을 것입니다.

```
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
```

테스트 방법론

1. 두 개의 유사한 클러스터가 생성되었습니다.
   - EC2 기반 합류 클러스터.
   - NetApp NFS 기반 합류 클러스터.
2. 원래 Kafka 클러스터의 노드와 동일한 구성을 가진 대기 Kafka 노드 하나가 생성되었습니다.
3. 각 클러스터에서 샘플 토픽이 생성되었고, 각 브로커에 약 110GB의 데이터가 채워졌습니다.
   - **EC2** 기반 클러스터. Kafka 브로커 데이터 디렉토리는 다음에 매핑됩니다. `/mnt/data-2` (다음 그림에서 cluster1의 Broker-1[왼쪽 터미널]).
   - * NetApp NFS 기반 클러스터.* Kafka 브로커 데이터 디렉토리는 NFS 지점에 마운트됩니다. `/mnt/data` (다음 그림에서 클러스터2의 Broker-1[오른쪽 터미널]).


4. 각 클러스터에서 Broker-1이 종료되어 실패한 브로커 복구 프로세스가 시작되었습니다.

5. 브로커가 종료된 후 브로커 IP 주소는 대기 브로커의 보조 IP로 할당되었습니다. 이는 Kafka 클러스터의 브로커가 다음으로 식별되기 때문에 필요했습니다.
   - **IP** 주소**.** 실패한 브로커 IP를 대기 브로커에 재할당하여 할당됩니다.
   - 브로커 **ID.** 이는 대기 브로커에서 구성되었습니다. `server.properties` .
6. IP 할당 시, 대기 브로커에서 Kafka 서비스가 시작되었습니다.
7. 얼마 후, 클러스터의 교체 노드에서 데이터를 빌드하는 데 걸리는 시간을 확인하기 위해 서버 로그를 뽑았습니다.

관찰

카프카 브로커 복구가 거의 9배 더 빨랐습니다. Kafka 클러스터에서 DAS SSD를 사용하는 것보다 NetApp NFS 공유 스토리지를 사용하면 실패한 브로커 노드를 복구하는 데 걸리는 시간이 훨씬 빠른 것으로 나타났습니다. 1TB의 주제 데이터에 대해 DAS 기반 클러스터의 복구 시간은 48분이었고, NetApp-NFS 기반 Kafka 클러스터의 경우 5분 미만이었습니다.

EC2 기반 클러스터는 새로운 브로커 노드에서 110GB의 데이터를 재구축하는 데 10분이 걸렸지만, NFS 기반 클러스터는 3분 만에 복구를 완료했습니다. 또한 EC2의 파티션에 대한 소비자 오프셋이 0인 반면, NFS 클러스터에서는 소비자 오프셋이 이전 브로커에서 수집된 것을 로그에서 확인했습니다.

```
[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** 기반 클러스터

1. 백업 노드는 08:55:53,730에 시작되었습니다.

```
[2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
[2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (or
[2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
[2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.3
[2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. 데이터 재구축 프로세스는 09:05:24,860에 종료되었습니다. 110GB의 데이터를 처리하는 데 약 10분이 걸렸습니다.

```
[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)
```

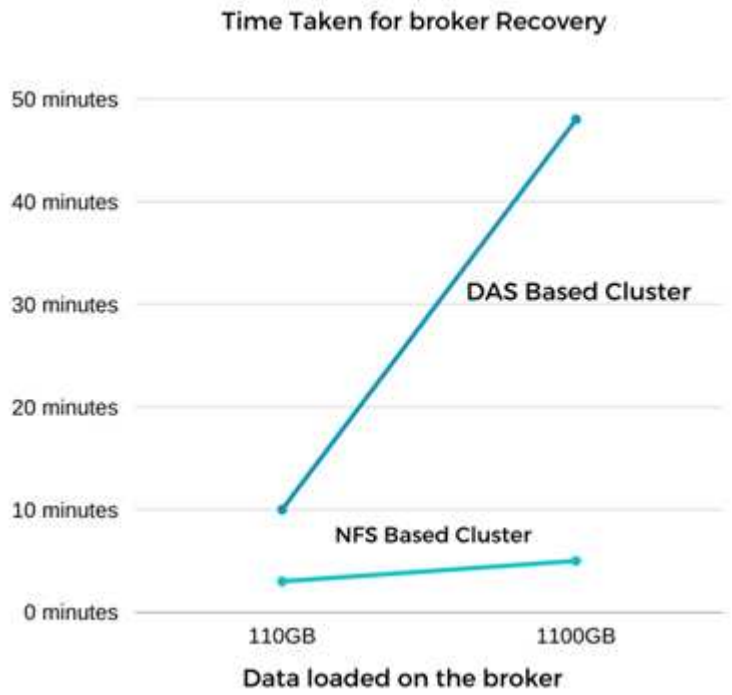
1. 백업 노드는 09:39:17,213에 시작되었습니다. 시작 로그 항목은 아래에 강조 표시되어 있습니다.

```
[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. 데이터 재구축 프로세스는 09:42:29,115에 종료되었습니다. 110GB의 데이터를 처리하는 데 약 3분이 걸렸습니다.

```
[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)
```

약 1TB의 데이터를 담고 있는 브로커에 대해 테스트를 반복한 결과, DAS의 경우 약 48분, NFS의 경우 3분이 걸렸습니다. 결과는 다음 그래프에 나타나 있습니다.




## 스토리지 효율성

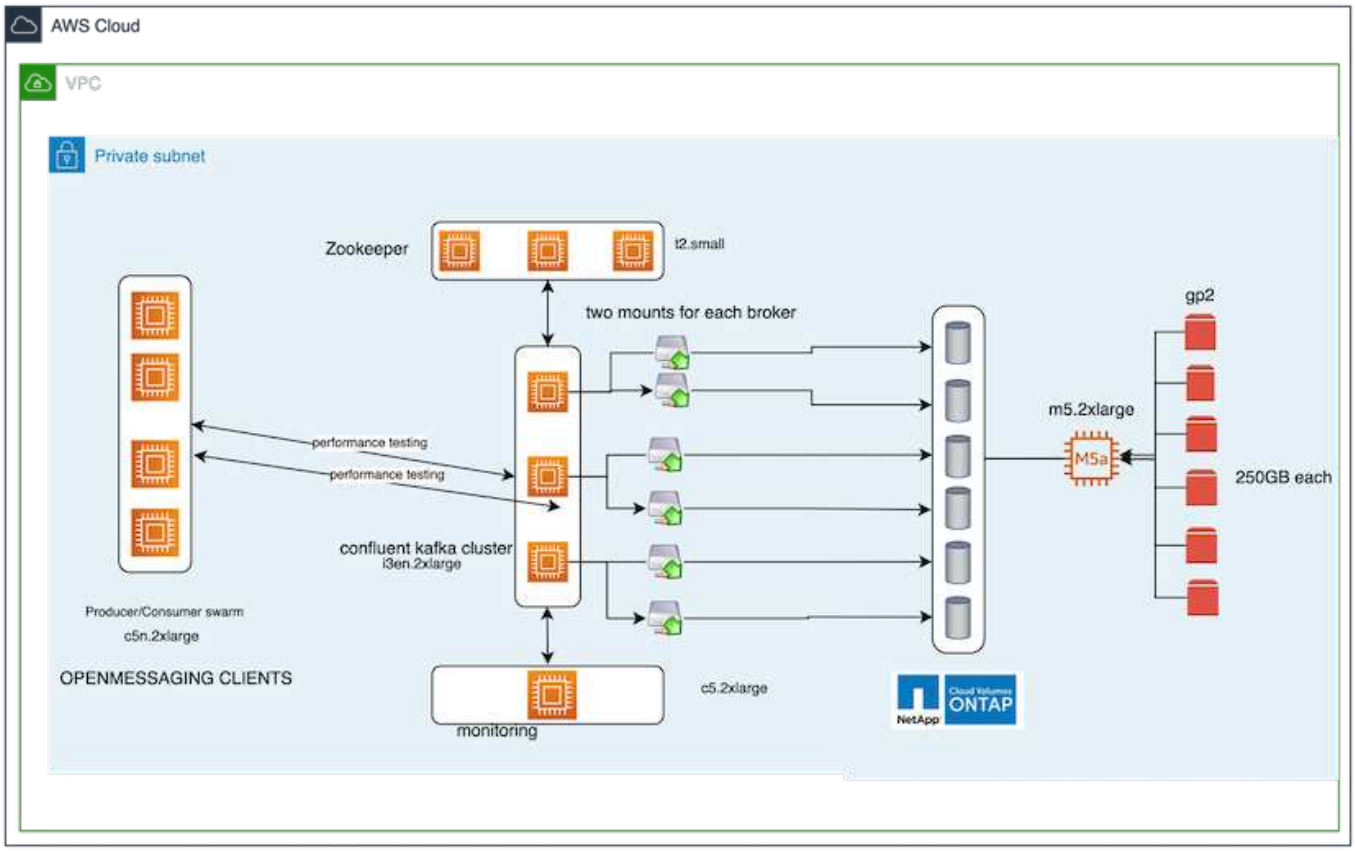
Kafka 클러스터의 스토리지 계층은 NetApp ONTAP 통해 프로비저닝되었기 때문에 ONTAP 의 모든 스토리지 효율성 기능을 얻을 수 있었습니다. 이는 Cloud Volumes ONTAP 에 프로비저닝된 NFS 스토리지를 사용하여 Kafka 클러스터에서 상당한 양의 데이터를 생성하여 테스트되었습니다. ONTAP 기능으로 인해 상당한 공간 감소가 발생했음을 알 수 있었습니다.

건축적 설정

다음 표는 NAS를 사용하는 Kafka 클러스터의 환경 구성을 보여줍니다.

| 플랫폼 구성 요소 | 환경 구성 |
| --- | --- |
| 카프카 3.2.3 | • 3 x 동물원 관리인 – t2.small<br>• 3개의 브로커 서버 - i3en.2xlarge<br>• 1 x 그라파나 – c5n.2xlarge<br>• 4 x 생산자/소비자 — c5n.2xlarge * |
| 모든 노드의 운영 체제 | RHEL8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

다음 그림은 NAS 기반 Kafka 클러스터의 아키텍처를 보여줍니다.




- 계산합니다. 우리는 전용 서버에서 실행되는 3노드 Zookeeper 앙상블과 함께 3노드 Kafka 클러스터를 사용했습니다. 각 브로커는 전용 LIF를 통해 NetApp CVO 인스턴스의 단일 볼륨에 대한 두 개의 NFS 마운트 지점을 갖고 있었습니다.
- 모니터링. 우리는 Prometheus-Grafana 조합을 위해 두 개의 노드를 사용했습니다. 워크로드를 생성하기 위해 이 Kafka 클러스터에서 워크로드를 생성하고 소비할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- 저장. 우리는 6개의 250GB GP2 AWS-EBS 볼륨이 인스턴스에 마운트된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스를 사용했습니다. 이러한 볼륨은 전용 LIF를 통해 6개의 NFS 볼륨으로 Kafka 클러스터에 노출되었습니다.

- 구성. 이 테스트 사례에서 구성 가능한 요소는 Kafka 브로커였습니다.

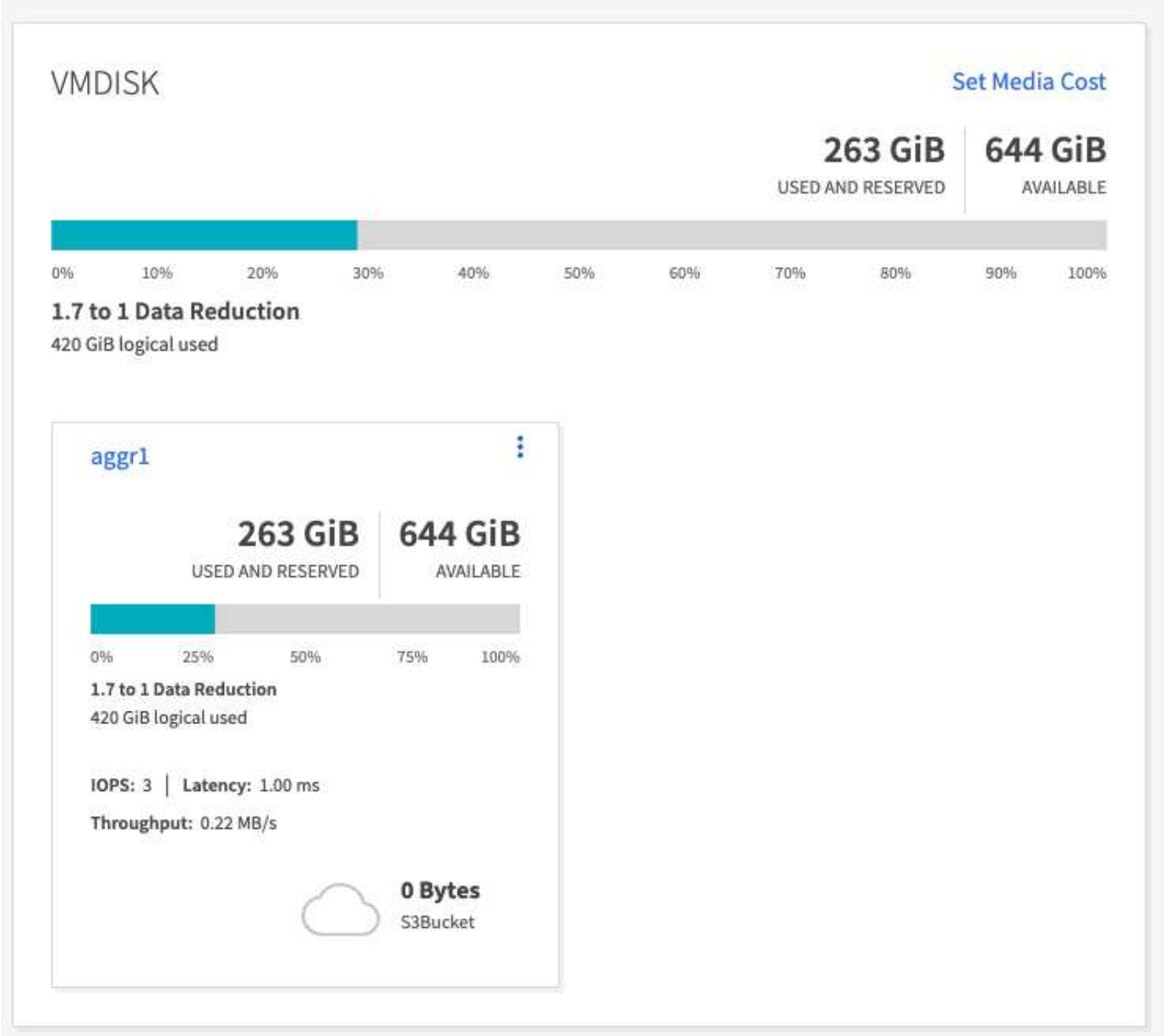
생산자 측에서는 압축을 해제하여 생산자가 높은 처리량을 창출할 수 있게 했습니다. 대신 저장 효율성은 컴퓨팅 계층에서 처리되었습니다.

테스트 방법론

1. 카프카 클러스터는 위에 언급된 사양에 따라 제공되었습니다.
2. 클러스터에서는 OpenMessaging 벤치마킹 도구를 사용하여 약 350GB의 데이터가 생성되었습니다.
3. 작업 부하가 완료된 후 ONTAP System Manager와 CLI를 사용하여 스토리지 효율성 통계를 수집했습니다.

관찰

OMB 도구를 사용하여 생성된 데이터의 경우 저장 효율성 비율이 1.70:1로 약 33%의 공간 절약 효과를 보였습니다. 다음 그림에서 볼 수 있듯이, 생성된 데이터가 사용하는 논리적 공간은 420.3GB이고, 데이터를 보관하는 데 사용된 물리적 공간은 281.7GB였습니다.



```
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
```

# AWS에서의 성능 개요 및 검증

NetApp NFS에 스토리지 계층이 탑재된 Kafka 클러스터의 성능이 AWS 클라우드에서 벤치마킹되었습니다. 벤치마킹 사례는 다음 섹션에 설명되어 있습니다.

## NetApp Cloud Volumes ONTAP (고가용성 쌍 및 단일 노드)을 사용한 AWS 클라우드의 Kafka

NetApp Cloud Volumes ONTAP (HA 쌍)이 포함된 Kafka 클러스터는 AWS 클라우드에서 성능을 벤치마킹했습니다. 다음 섹션에서는 이 벤치마킹에 대해 설명합니다.

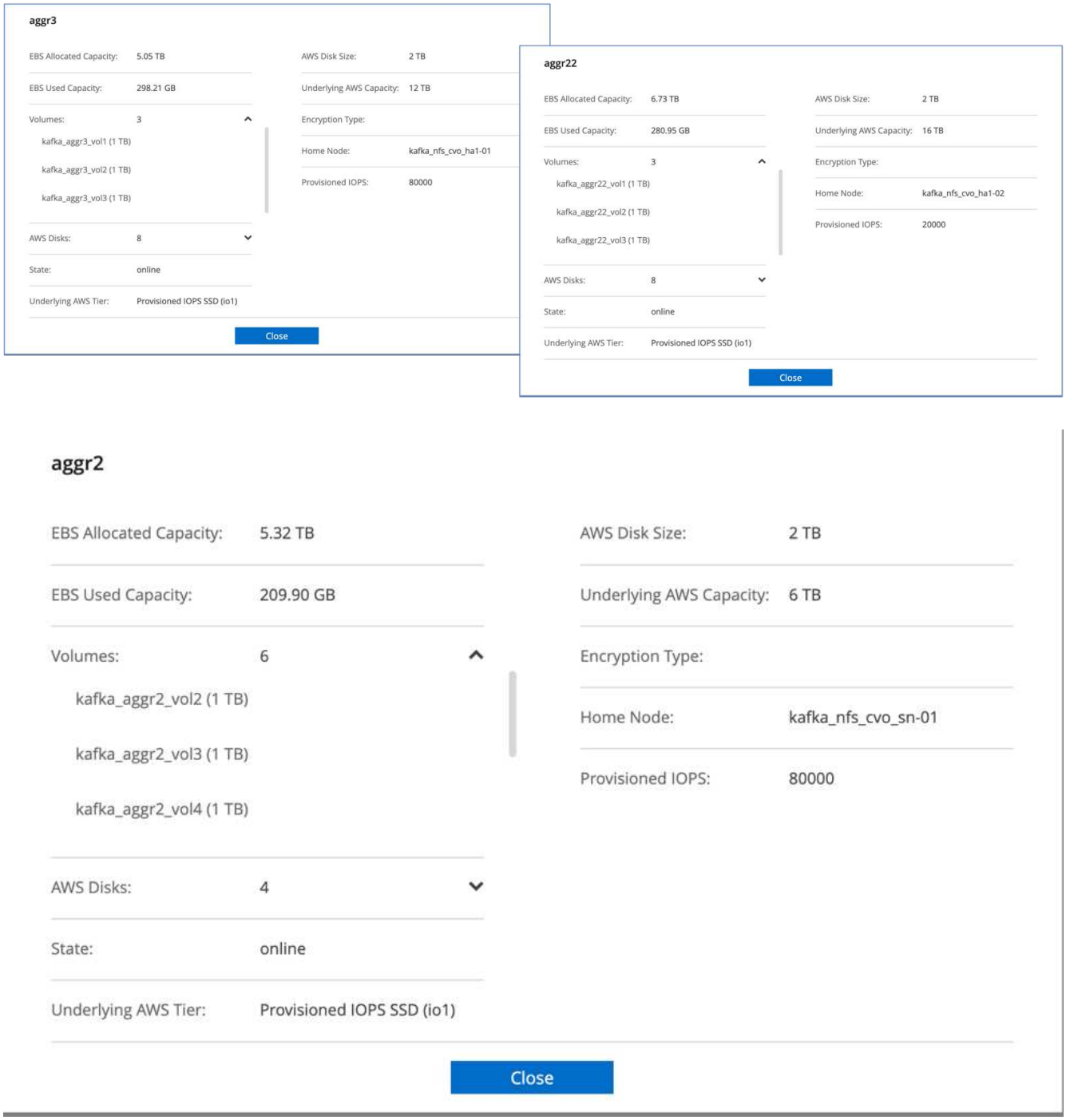
건축적 설정

다음 표는 NAS를 사용하는 Kafka 클러스터의 환경 구성을 보여줍니다.

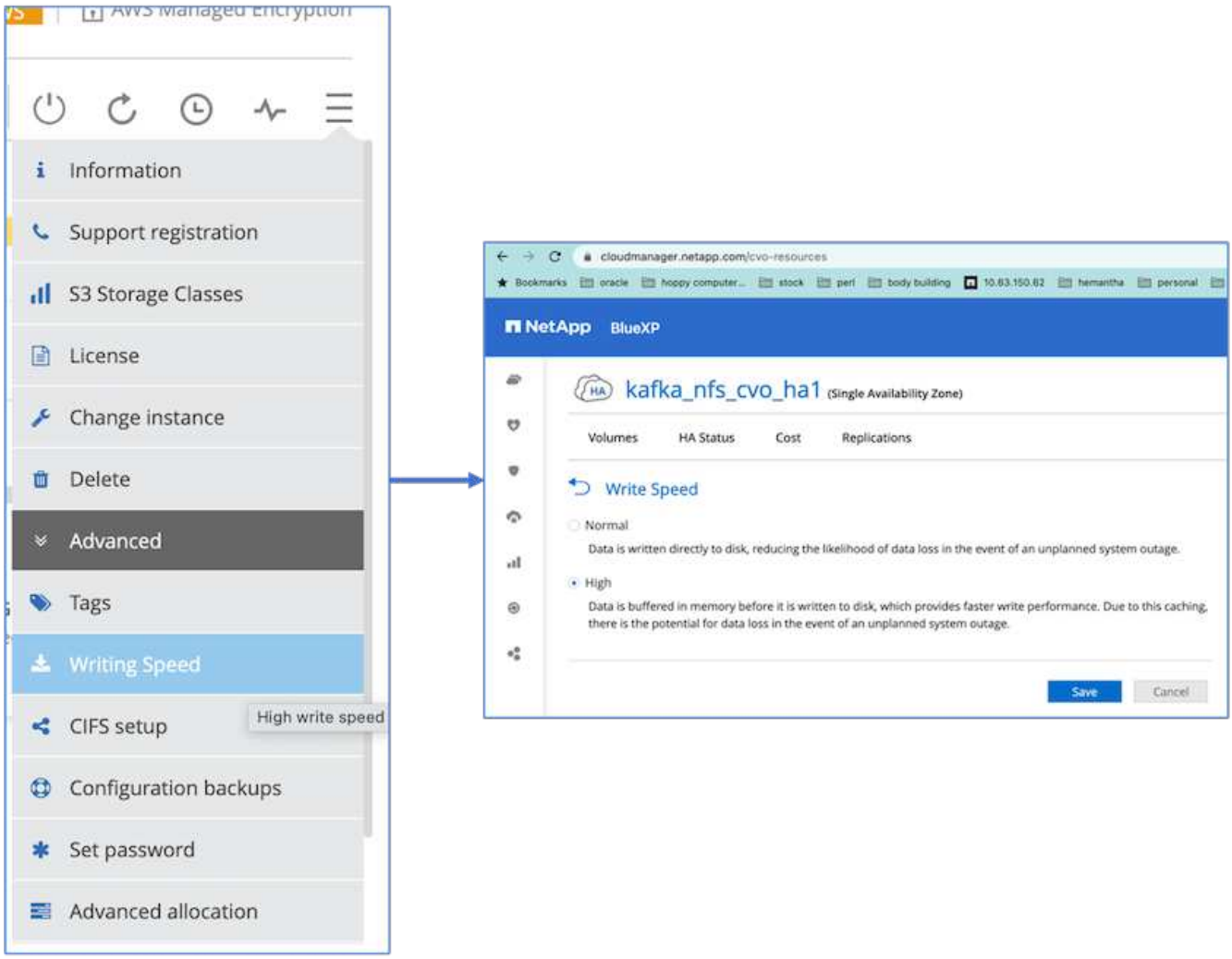
| 플랫폼 구성 요소 | 환경 구성 |
| --- | --- |
| 카프카 3.2.3 | • 3 x 동물원 관리인 – t2.small<br>• 3개의 브로커 서버 - i3en.2xlarge<br>• 1 x 그라파나 – c5n.2xlarge<br>• 4 x 생산자/소비자 — c5n.2xlarge * |
| 모든 노드의 운영 체제 | RHEL8.6 |
| NetApp Cloud Volumes ONTAP 인스턴스 | HA 쌍 인스턴스 - m5dn.12xLarge x 2노드 단일 노드 인스턴스 - m5dn.12xLarge x 1노드 |

**NetApp** 클러스터 볼륨 **ONTAP** 설정

1. Cloud Volumes ONTAP HA 쌍의 경우 각 스토리지 컨트롤러에서 각 집계에 3개의 볼륨이 있는 2개의 집계를 생성했습니다. 단일 Cloud Volumes ONTAP 노드의 경우 집계된 6개의 볼륨을 생성합니다.

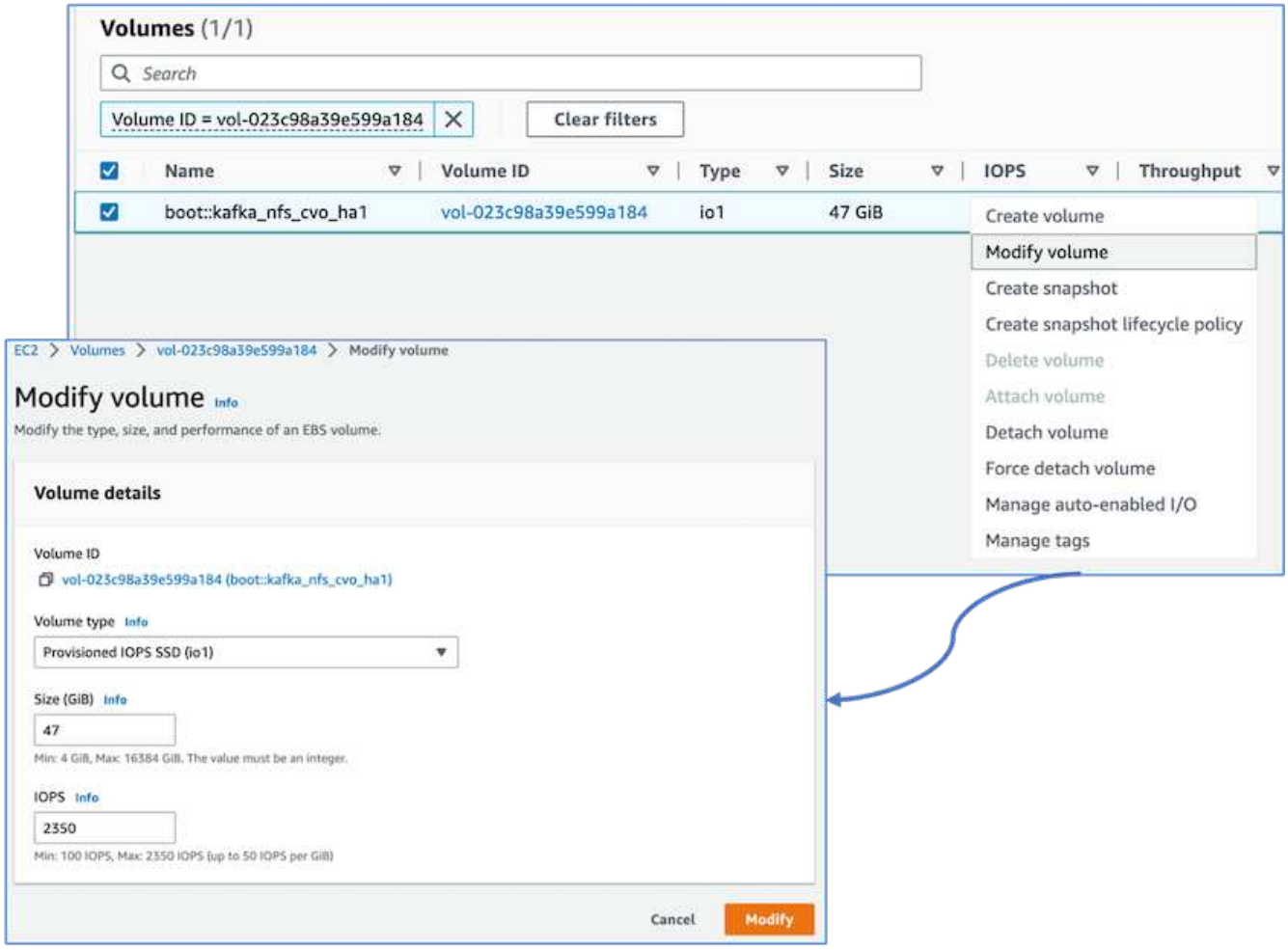



2. 더 나은 네트워크 성능을 달성하기 위해 HA 쌍과 단일 노드 모두에 고속 네트워킹을 구현했습니다.



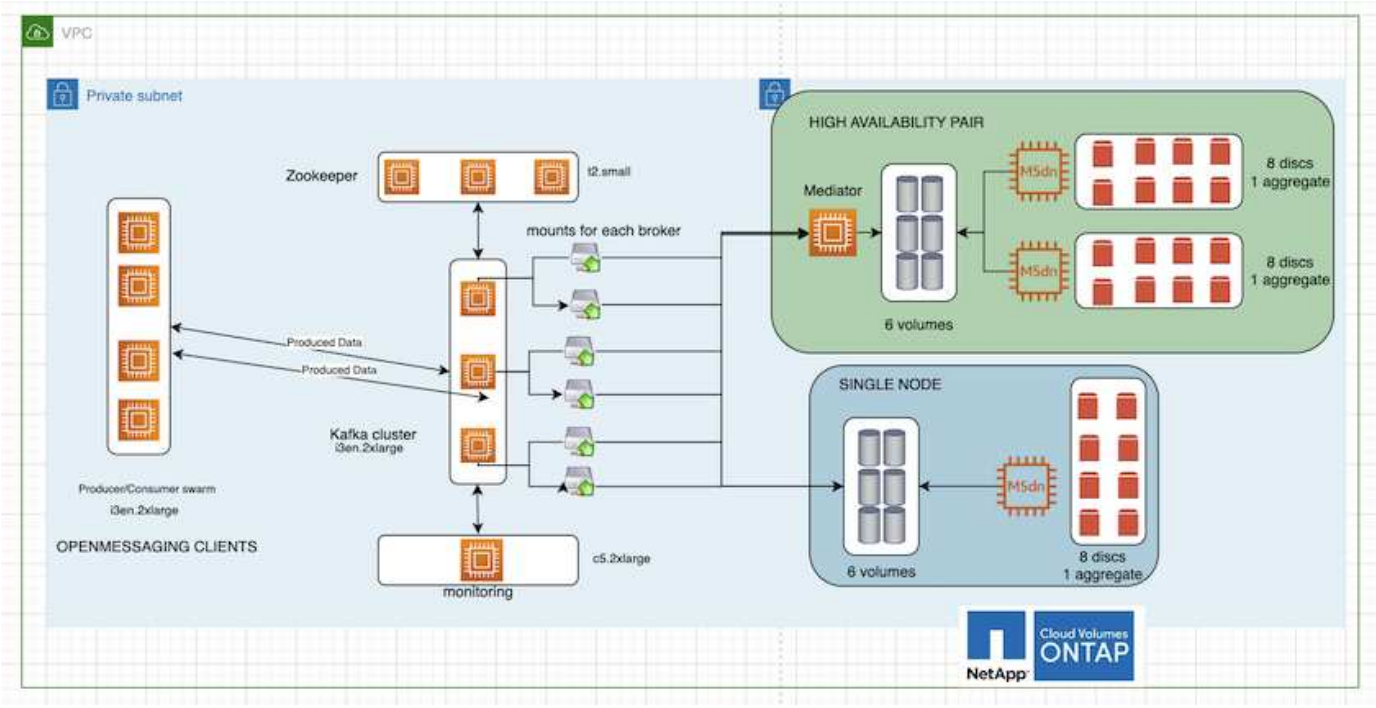

3. ONTAP NVRAM IOPS가 더 높다는 것을 알았으므로 Cloud Volumes ONTAP 루트 볼륨의 IOPS를 2350으로 변경했습니다. Cloud Volumes ONTAP 의 루트 볼륨 디스크 크기는 47GB입니다. 다음 ONTAP 명령은 HA 쌍을 위한 것이며, 동일한 단계가 단일 노드에도 적용됩니다.

```
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::*>
```




다음 그림은 NAS 기반 Kafka 클러스터의 아키텍처를 보여줍니다.

- 계산합니다. 우리는 전용 서버에서 실행되는 3노드 Zookeeper 앙상블과 함께 3노드 Kafka 클러스터를 사용했습니다. 각 브로커는 전용 LIF를 통해 Cloud Volumes ONTAP 인스턴스의 단일 볼륨에 대한 두 개의 NFS 마운트 지점을 가졌습니다.
- 모니터링. 우리는 Prometheus-Grafana 조합을 위해 두 개의 노드를 사용했습니다. 워크로드를 생성하기 위해 이 Kafka 클러스터에서 워크로드를 생성하고 소비할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- 저장. 우리는 인스턴스에 마운트된 6TB GP3 AWS-EBS 볼륨 1개가 있는 HA 쌍 Cloud Volumes ONTAP 인스턴스를 사용했습니다. 그런 다음 볼륨은 NFS 마운트를 통해 Kafka 브로커로 내보내졌습니다.




**OpenMessage** 벤치마킹 구성

1. 더 나은 NFS 성능을 위해서는 NFS 서버와 NFS 클라이언트 사이에 더 많은 네트워크 연결이 필요한데, 이는 nconnect를 사용하여 생성할 수 있습니다. 다음 명령을 실행하여 nconnect 옵션으로 브로커 노드에 NFS 볼륨을 마운트합니다.

```
[root@ip-172-30-0-121 ~]# cat /etc/fstab
UUID=eaa1f38e-de0f-4ed5-a5b5-2fa9db43bb38/xfsdefaults00
/dev/nvme1n1 /mnt/data-1 xfs defaults,noatime,nodiscard 0 0
/dev/nvme2n1 /mnt/data-2 xfs defaults,noatime,nodiscard 0 0
172.30.0.233:/kafka_aggr3_vol1 /kafka_aggr3_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol2 /kafka_aggr3_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol3 /kafka_aggr3_vol3 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol1 /kafka_aggr22_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol2 /kafka_aggr22_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol3 /kafka_aggr22_vol3 nfs
defaults,nconnect=16 0 0
[root@ip-172-30-0-121 ~]# mount -a
[root@ip-172-30-0-121 ~]# df -h
Filesystem                         Size  Used Avail Use% Mounted on
devtmpfs                            31G     0   31G   0% /dev
tmpfs                               31G  249M   31G   1% /run
tmpfs                               31G     0   31G   0% /sys/fs/cgroup
/dev/nvme0n1p2                      10G  2.8G  7.2G  28% /
/dev/nvme1n1                       2.3T  248G  2.1T  11% /mnt/data-1
/dev/nvme2n1                       2.3T  245G  2.1T  11% /mnt/data-2
172.30.0.233:/kafka_aggr3_vol1     1.0T   12G 1013G   2% /kafka_aggr3_vol1
172.30.0.233:/kafka_aggr3_vol2     1.0T  5.5G 1019G   1% /kafka_aggr3_vol2
172.30.0.233:/kafka_aggr3_vol3     1.0T  8.9G 1016G   1% /kafka_aggr3_vol3
172.30.0.242:/kafka_aggr22_vol1    1.0T  7.3G 1017G   1%
/kafka_aggr22_vol1
172.30.0.242:/kafka_aggr22_vol2    1.0T  6.9G 1018G   1%
/kafka_aggr22_vol2
172.30.0.242:/kafka_aggr22_vol3    1.0T  5.9G 1019G   1%
/kafka_aggr22_vol3
tmpfs                              6.2G     0  6.2G   0% /run/user/1000
[root@ip-172-30-0-121 ~]#
```

2. Cloud Volumes ONTAP 에서 네트워크 연결을 확인하세요. 다음 ONTAP 명령은 단일 Cloud Volumes ONTAP 노드에서 사용됩니다. 동일한 단계는 Cloud Volumes ONTAP HA 쌍에도 적용됩니다.

```
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. 우리는 다음의 카프카를 사용합니다 `server.properties` Cloud Volumes ONTAP HA 쌍에 대한 모든 Kafka 브로커에서. 그만큼 `log.dirs` 각 중개인마다 취급하는 부동산의 종류가 다르고, 나머지 부동산은 중개인들이 공통적으로 취급합니다. 브로커1의 경우 `log.dirs` 값은 다음과 같습니다.

```
[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 ~]#
```

◦ 브로커2의 경우 `log.dirs` 재산 가치는 다음과 같습니다.

```
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
```

◦ 브로커3의 경우 `log.dirs` 재산 가치는 다음과 같습니다.

```
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. 단일 Cloud Volumes ONTAP 노드의 경우 Kafka `servers.properties` Cloud Volumes ONTAP HA 쌍의 경우와 동일하지만 `log.dirs` 재산.
   - 브로커1의 경우 `log.dirs` 값은 다음과 같습니다.

```
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
```

   - 브로커2의 경우 `log.dirs` 값은 다음과 같습니다.

```
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
```

   - 브로커3의 경우 `log.dirs` 재산 가치는 다음과 같습니다.

```
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. OMB의 작업 부하는 다음 속성으로 구성됩니다. (`/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
```

그만큼 `messageSize` 각 사용 사례마다 다를 수 있습니다. 성능 테스트에서는 3K를 사용했습니다.

우리는 OMB의 Sync와 Throughput이라는 두 가지 드라이버를 사용하여 Kafka 클러스터에서 작업 부하를 생성했습니다.

- 동기화 드라이버 속성에 사용되는 yaml 파일은 다음과 같습니다. (`/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
```

- Throughput 드라이버 속성에 사용되는 yaml 파일은 다음과 같습니다. (`/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
```

## 테스트 방법론

1. Terraform과 Ansible을 사용하여 위에 설명된 사양에 따라 Kafka 클러스터가 프로비저닝되었습니다. Terraform은 Kafka 클러스터용 AWS 인스턴스를 사용하여 인프라를 구축하는 데 사용되고 Ansible은 해당 인스턴스에 Kafka 클러스터를 구축합니다.
2. 위에 설명된 워크로드 구성과 Sync 드라이버를 사용하여 OMB 워크로드가 트리거되었습니다.

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

3. 동일한 작업 부하 구성을 사용하는 Throughput 드라이버로 또 다른 작업 부하가 트리거되었습니다.

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

## 관찰

NFS에서 실행되는 Kafka 인스턴스의 성능을 벤치마킹하기 위한 워크로드를 생성하기 위해 두 가지 유형의 드라이버가 사용되었습니다. 드라이버 간의 차이점은 로그 플러시 속성입니다.

Cloud Volumes ONTAP HA 쌍의 경우:

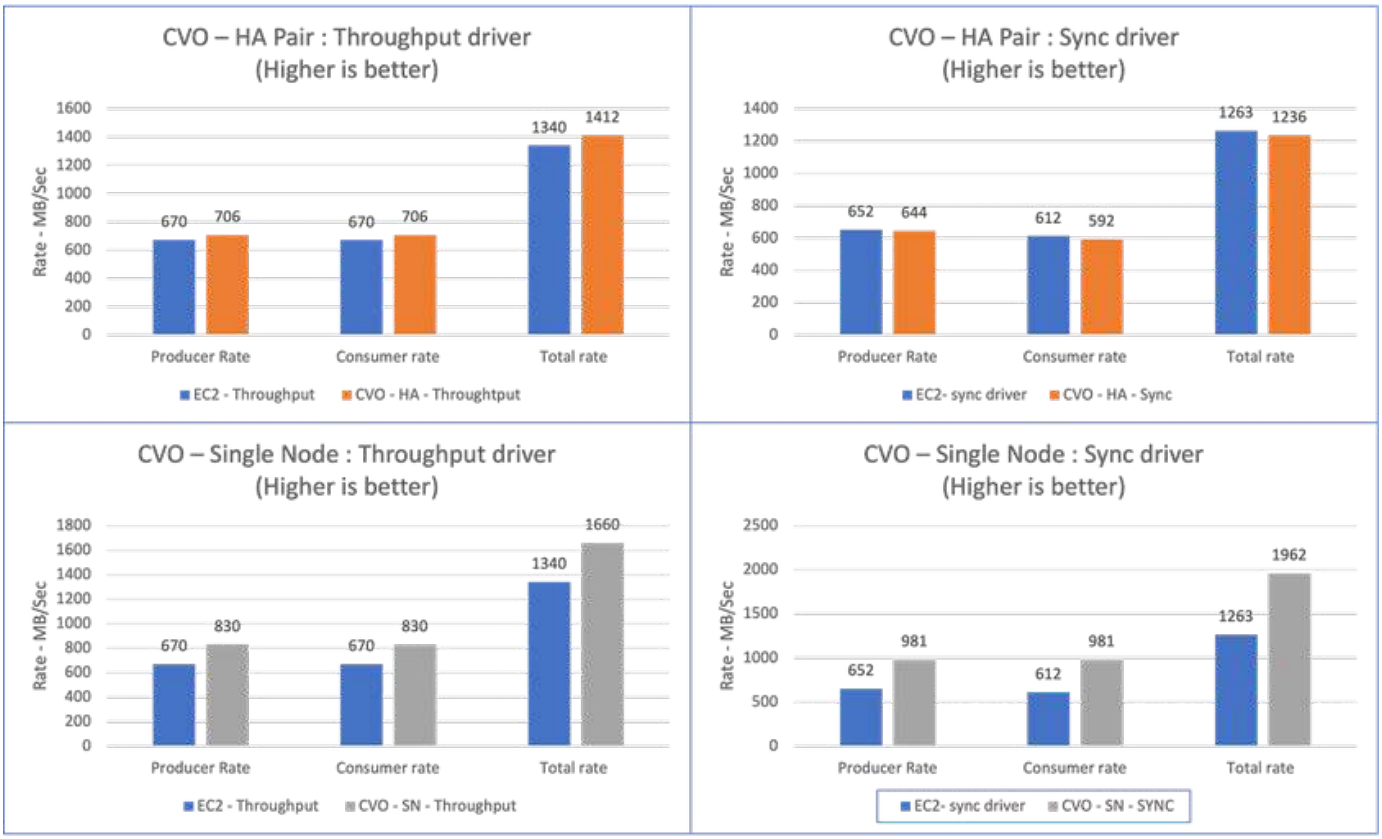
- Sync 드라이버에서 지속적으로 생성된 총 처리량: ~1236MBps.
- Throughput 드라이버에 대해 생성된 총 처리량: 최대 ~1412MBps.

단일 Cloud Volumes ONTAP 노드의 경우:

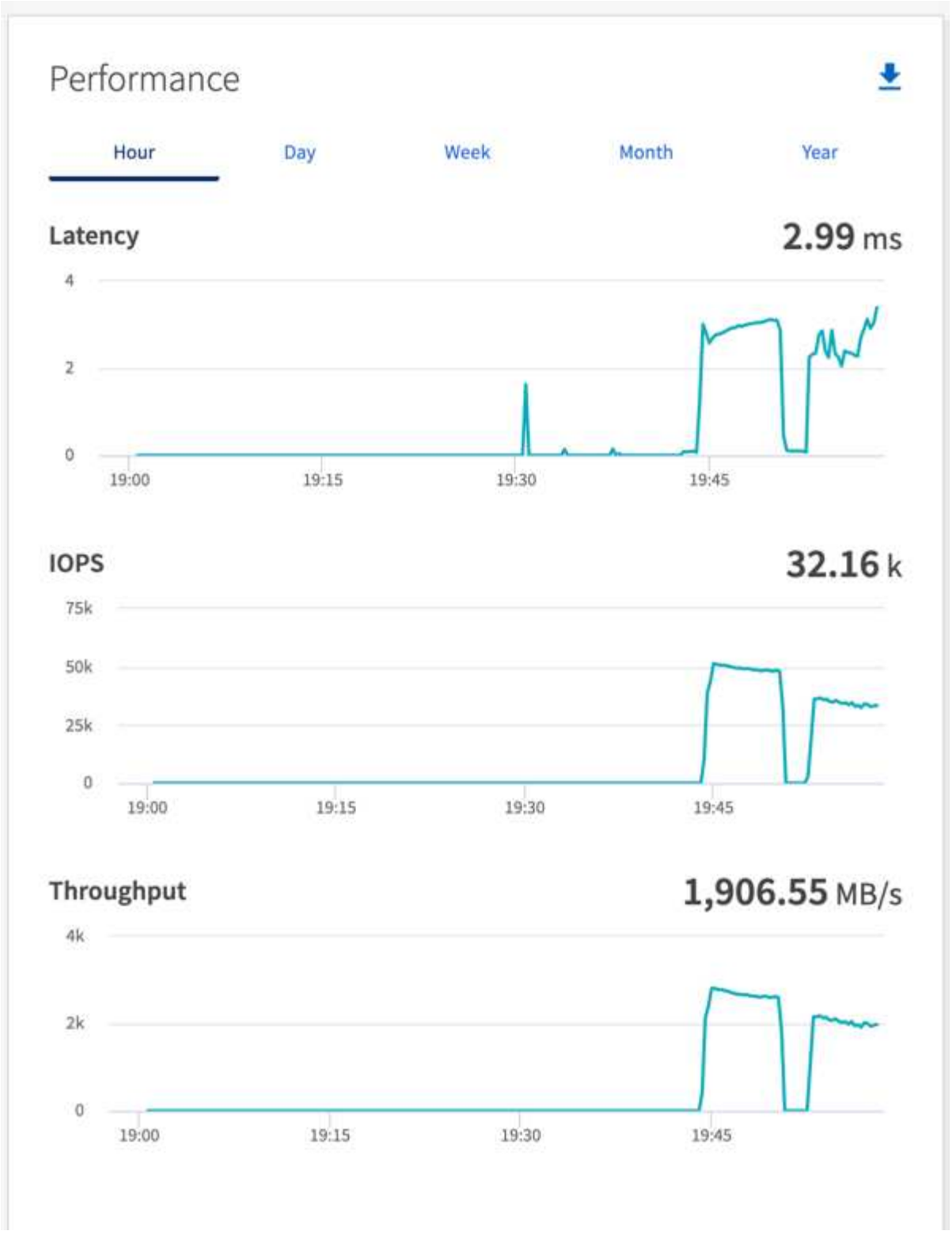
- Sync 드라이버에서 지속적으로 생성된 총 처리량: ~ 1962MBps.
- Throughput 드라이버에서 생성된 총 처리량: 최대 ~1660MBps

Sync 드라이버는 로그가 디스크에 즉시 플러시되므로 일관된 처리량을 생성할 수 있는 반면, Throughput 드라이버는 로그가 대량으로 디스크에 커밋되므로 처리량이 급증합니다.

이러한 처리량 수치는 주어진 AWS 구성에 대해 생성됩니다. 더 높은 성능이 필요한 경우 인스턴스 유형을 확장하고 조정하여 처리량을 더욱 높일 수 있습니다. 총 처리량 또는 총 속도는 생산자 속도와 소비자 속도를 합친 것입니다.




처리량이나 동기화 드라이버 벤치마킹을 수행할 때는 반드시 저장소 처리량을 확인하세요.




# AWS FSx ONTAP 의 성능 개요 및 검증

NetApp NFS에 스토리지 계층이 장착된 Kafka 클러스터는 AWS FSx ONTAP 에서 성능을 벤치마킹했습니다. 벤치마킹 사례는 다음 섹션에 설명되어 있습니다.

# AWS FSx ONTAP 의 Apache Kafka

네트워크 파일 시스템(NFS)은 대량의 데이터를 저장하는 데 널리 사용되는 네트워크 파일 시스템입니다. 대부분의 조직에서는 Apache Kafka와 같은 스트리밍 애플리케이션을 통해 데이터가 생성되는 경우가 점점 늘어나고 있습니다. 이러한 작업에는 확장성, 낮은 지연 시간, 최신 스토리지 기능을 갖춘 강력한 데이터 수집 아키텍처가 필요합니다. 실시간 분석을 가능하게 하고 실행 가능한 통찰력을 제공하려면 잘 설계되고 성능이 뛰어난 인프라가 필요합니다.

Kafka는 설계상 POSIX 호환 파일 시스템에서 작동하며 파일 작업을 처리하기 위해 해당 파일 시스템에 의존하지만, NFSv3 파일 시스템에 데이터를 저장하는 경우 Kafka 브로커 NFS 클라이언트는 XFS나 Ext4와 같은 로컬 파일 시스템과 다르게 파일 작업을 해석할 수 있습니다. 흔한 예로는 NFS Silly 이름 변경으로 인해 클러스터를 확장하고 파티션을 재할당할 때 Kafka 브로커가 실패하는 경우가 있습니다. 이러한 과제를 해결하기 위해 NetApp 오픈 소스 Linux NFS 클라이언트를 업데이트하여 현재 RHEL8.7, RHEL9.1에서 일반적으로 사용할 수 있는 변경 사항을 적용했으며, 현재 FSx ONTAP 릴리스인 ONTAP 9.12.1에서 지원됩니다.

Amazon FSx ONTAP 클라우드에서 완벽하게 관리되고 확장 가능하며 성능이 뛰어난 NFS 파일 시스템을 제공합니다. FSx ONTAP 의 Kafka 데이터는 대량의 데이터를 처리하고 내결함성을 보장하도록 확장될 수 있습니다. NFS는 중요하고 민감한 데이터 세트에 대한 중앙 집중식 스토리지 관리 및 데이터 보호를 제공합니다.

이러한 향상된 기능을 통해 AWS 고객은 AWS 컴퓨팅 서비스에서 Kafka 워크로드를 실행할 때 FSx ONTAP 의 이점을 활용할 수 있습니다. 이러한 이점은 다음과 같습니다. * CPU 사용률을 줄여 I/O 대기 시간을 줄입니다. * Kafka 브로커 복구 시간이 더 빠릅니다. * 신뢰성과 효율성. * 확장성 및 성능. * 다중 이용 가능 구역 이용 가능. * 데이터 보호.

## AWS FSx ONTAP 의 성능 개요 및 검증

NetApp NFS에 스토리지 계층이 탑재된 Kafka 클러스터의 성능이 AWS 클라우드에서 벤치마킹되었습니다. 벤치마킹 사례는 다음 섹션에 설명되어 있습니다.

### AWS FSx ONTAP 의 카프카

AWS FSx ONTAP 탑재된 Kafka 클러스터는 AWS 클라우드에서 성능을 벤치마킹했습니다. 다음 섹션에서는 이 벤치마킹에 대해 설명합니다.

#### 건축적 설정

다음 표는 AWS FSx ONTAP 사용하는 Kafka 클러스터의 환경 구성을 보여줍니다.

| 플랫폼 구성 요소 | 환경 구성 |
| --- | --- |
| 카프카 3.2.3 | • 3 x 동물원 관리인 – t2.small<br>• 3개의 브로커 서버 - i3en.2xlarge<br>• 1 x 그라파나 – c5n.2xlarge<br>• 4 x 생산자/소비자 — c5n.2xlarge * |
| 모든 노드의 운영 체제 | RHEL8.6 |
| AWS FSx ONTAP | 4GB/초 처리량과 160000 IOPS를 갖춘 다중 AZ |

#### NetApp FSx ONTAP 설정

1. 초기 테스트를 위해 2TB 용량과 2GB/초 처리량을 위한 40000 IOPS를 갖춘 FSx ONTAP 파일 시스템을 만들었습니다.

```
[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"}
```

우리의 예에서는 AWS CLI를 통해 FSx ONTAP 배포합니다. 필요에 따라 사용자 환경에서 명령을 추가로 사용자 지정해야 합니다. FSx ONTAP AWS 콘솔을 통해 배포 및 관리할 수 있으므로 명령줄 입력을 줄여 더 쉽고 간소화된 배포 환경을 제공합니다.

FSx ONTAP 에서 테스트 지역(US-East-1)의 처리량 2GB/초 파일 시스템에서 달성 가능한 최대 IOPS는 80,000 iops입니다. FSx ONTAP 파일 시스템의 총 최대 IOPS는 160,000 IOPS이며, 이를 달성하려면 4GB/초의 처리량 배포가 필요합니다. 이에 대해서는 이 문서의 뒷부분에서 설명하겠습니다.

FSx ONTAP 성능 사양에 대한 자세한 내용은 여기에서 AWS FSx ONTAP 설명서를 참조하세요. https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html .

FSx "create-file-system"에 대한 자세한 명령줄 구문은 여기에서 확인할 수 있습니다. https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

예를 들어, KMS 키가 지정되지 않은 경우 사용되는 기본 AWS FSx 마스터 키 대신 특정 KMS 키를 지정할 수 있습니다.

2. FSx ONTAP 파일 시스템을 생성하는 동안 다음과 같이 파일 시스템을 설명한 후 JSON 반환에서 "LifeCycle" 상태가 "AVAILABLE"로 변경될 때까지 기다리세요.

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

3. fsxadmin 사용자로 FSx ONTAP SSH에 로그인하여 자격 증명을 확인합니다. Fsxadmin은 FSx ONTAP 파일 시스템을 생성할 때 사용되는 기본 관리자 계정입니다. fsxadmin의 비밀번호는 1단계에서 완료한 대로 AWS 콘솔이나 AWS CLI를 사용하여 파일 시스템을 처음 생성할 때 구성한 비밀번호입니다.

```
[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. 자격 증명이 검증되면 FSx ONTAP 파일 시스템에 스토리지 가상 머신을 생성합니다.

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

SVM(Storage Virtual Machine)은 FSx ONTAP 볼륨의 데이터를 관리하고 액세스하기 위한 자체 관리 자격 증명과 엔드포인트를 갖춘 격리된 파일 서버이며 FSx ONTAP 다중 테넌시를 제공합니다.

5. 기본 스토리지 가상 머신을 구성한 후 새로 만든 FSx ONTAP 파일 시스템에 SSH를 실행하고 아래 샘플 명령을 사용하여 스토리지 가상 머신에 볼륨을 생성합니다. 마찬가지로 이 검증을 위해 6개의 볼륨을 생성합니다. 우리의 검증에 따르면, 카프카에 더 나은 성능을 제공하는 기본 구성 요소(8) 또는 그 이하의 구성 요소를 유지합니다.

```
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. 테스트를 위해서는 볼륨에 추가 용량이 필요합니다. 볼륨 크기를 2TB로 확장하고 연결 경로에 마운트합니다.

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

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

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

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN4 -new-size +2TB
```

```
vol size: Volume "svmkafkatest:kafkafsxN4" size set to 2.10t.

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

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

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

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

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

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

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

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

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

FSx ONTAP 에서는 볼륨을 씬 프로비저닝할 수 있습니다. 예시에서 총 확장 볼륨 용량이 총 파일 시스템 용량을 초과하므로 추가 프로비저닝 볼륨 용량을 잠금 해제하려면 총 파일 시스템 용량을 확장해야 합니다. 이는 다음 단계에서 보여드리겠습니다.

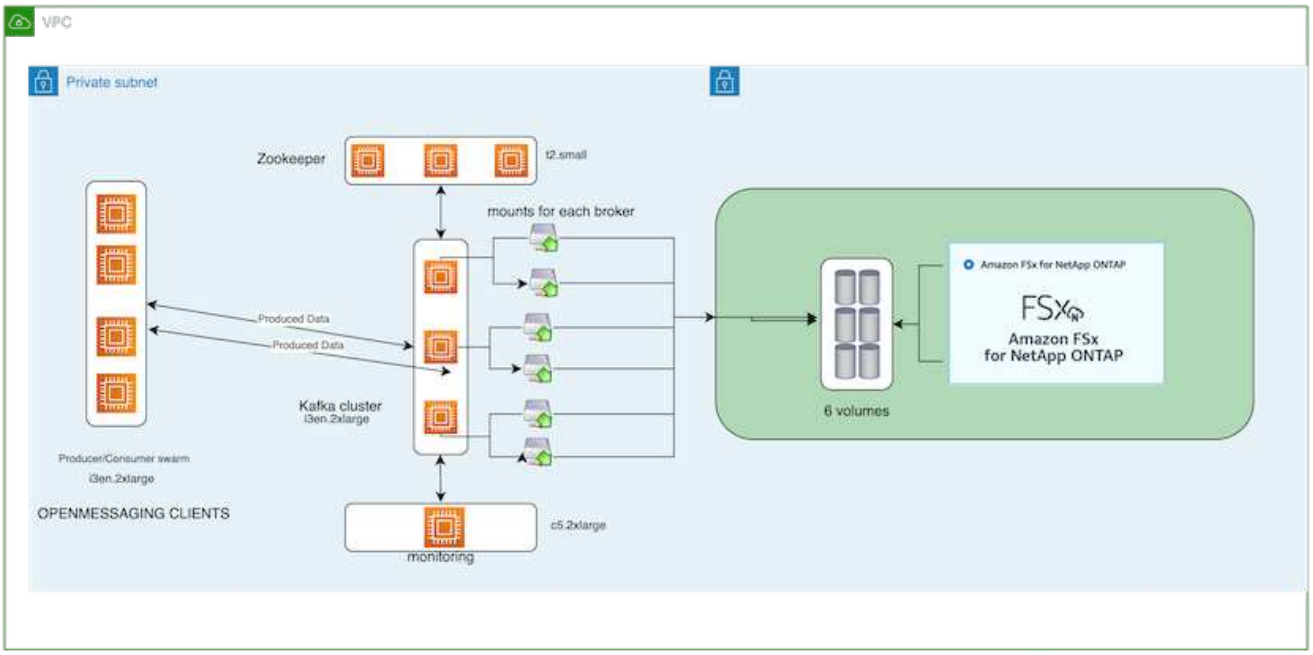
7. 다음으로, 추가적인 성능 및 용량을 위해 FSx ONTAP 처리량 용량을 2GB/초에서 4GB/초로, IOPS를 160000으로, 용량을 5TB로 확장합니다.

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

FSx "update-file-system"에 대한 자세한 명령줄 구문은 여기에서 확인할 수 있습니다.https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. FSx ONTAP 볼륨은 Kafka 브로커의 nconnect 및 기본 옵션으로 마운트됩니다.

다음 그림은 FSx ONTAP 기반 Kafka 클러스터의 최종 아키텍처를 보여줍니다.




◦ 계산하다. 우리는 전용 서버에서 실행되는 3노드 Zookeeper 앙상블과 함께 3노드 Kafka 클러스터를

사용했습니다. 각 브로커는 FSx ONTAP 인스턴스의 6개 볼륨에 대해 6개의 NFS 마운트 포인트를 갖고 있었습니다.

- 모니터링. 우리는 Prometheus-Grafana 조합을 위해 두 개의 노드를 사용했습니다. 워크로드를 생성하기 위해 이 Kafka 클러스터에서 워크로드를 생성하고 소비할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- 저장. 우리는 6개의 2TB 볼륨이 마운트된 FSx ONTAP 사용했습니다. 그런 다음 볼륨은 NFS 마운트를 통해 Kafka 브로커로 내보내졌습니다. FSx ONTAP 볼륨은 16개의 nconnect 세션과 Kafka 브로커의 기본 옵션으로 마운트됩니다.

**OpenMessage** 벤치마킹 구성.

NetApp Cloud Volumes ONTAP 에 사용된 것과 동일한 구성을 사용했으며 자세한 내용은 여기에 있습니다. 링크:kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

테스트 방법론

1. 위에 설명된 사양에 따라 Terraform과 Ansible을 사용하여 Kafka 클러스터를 프로비저닝했습니다. Terraform은 Kafka 클러스터용 AWS 인스턴스를 사용하여 인프라를 구축하는 데 사용되고, Ansible은 해당 인스턴스에 Kafka 클러스터를 구축합니다.
2. 위에 설명된 워크로드 구성과 Sync 드라이버를 사용하여 OMB 워크로드가 트리거되었습니다.

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

3. 동일한 작업 부하 구성을 사용하는 Throughput 드라이버로 또 다른 작업 부하가 트리거되었습니다.

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

관찰

NFS에서 실행되는 Kafka 인스턴스의 성능을 벤치마킹하기 위한 워크로드를 생성하기 위해 두 가지 유형의 드라이버가 사용되었습니다. 드라이버 간의 차이점은 로그 플러시 속성입니다.

Kafka 복제 요소 1 및 FSx ONTAP 의 경우:

- Sync 드라이버에서 지속적으로 생성된 총 처리량은 ~3218MBps이고 최대 성능은 ~3652MBps입니다.
- Throughput 드라이버에서 지속적으로 생성된 총 처리량은 ~3679MBps이고 최대 성능은 ~3908MBps입니다.

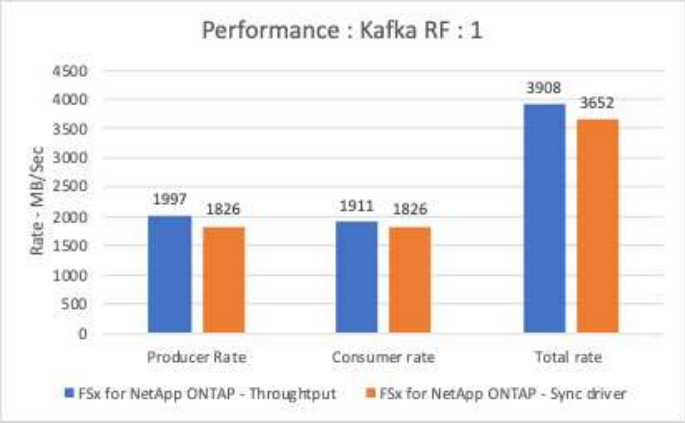
복제 계수 3과 FSx ONTAP 있는 Kafka의 경우:

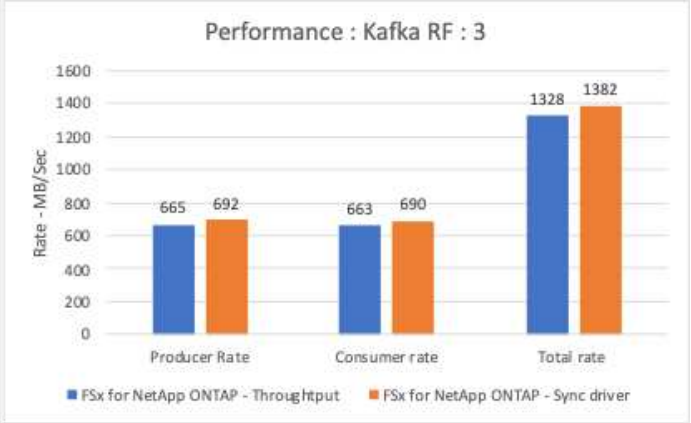
- Sync 드라이버에서 지속적으로 생성된 총 처리량은 ~1252MBps이고 최대 성능은 ~1382MBps입니다.
- Throughput 드라이버에서 지속적으로 생성된 총 처리량은 ~1218MBps이고 최대 성능은 ~1328MBps입니다.

Kafka 복제 요소 3에서는 읽기 및 쓰기 작업이 FSx ONTAP 에서 3번 발생했고, Kafka 복제 요소 1에서는 읽기 및 쓰기 작업이 FSx ONTAP 에서 한 번 발생했으므로 두 가지 검증에서 모두 최대 처리량인 4GB/초에 도달할 수 있었습니다.

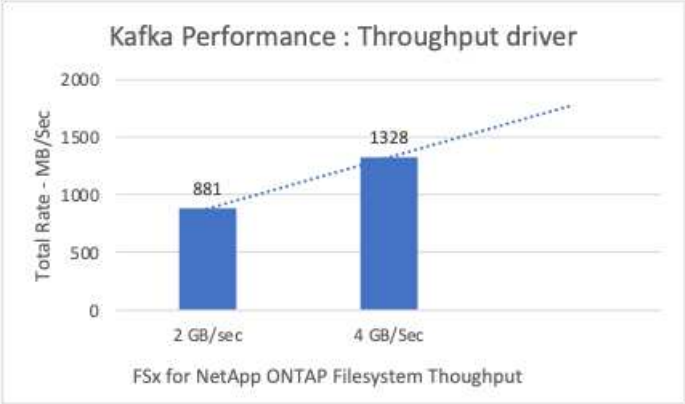
Sync 드라이버는 로그가 디스크에 즉시 플러시되므로 일관된 처리량을 생성할 수 있는 반면, Throughput 드라이버는 로그가 대량으로 디스크에 커밋되므로 처리량이 급증합니다.

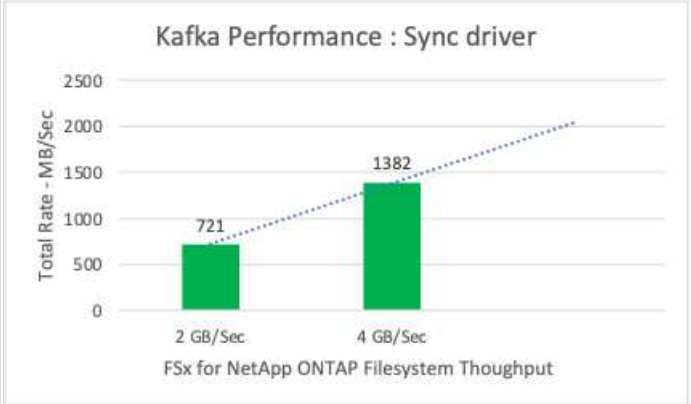
이러한 처리량 수치는 주어진 AWS 구성에 대해 생성됩니다. 더 높은 성능이 필요한 경우 인스턴스 유형을 확장하고 조정하여 처리량을 더욱 높일 수 있습니다. 총 처리량 또는 총 속도는 생산자 속도와 소비자 속도를 합친 것입니다.



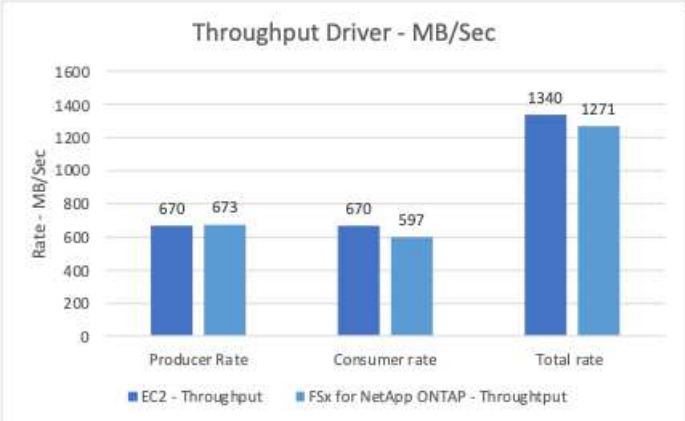



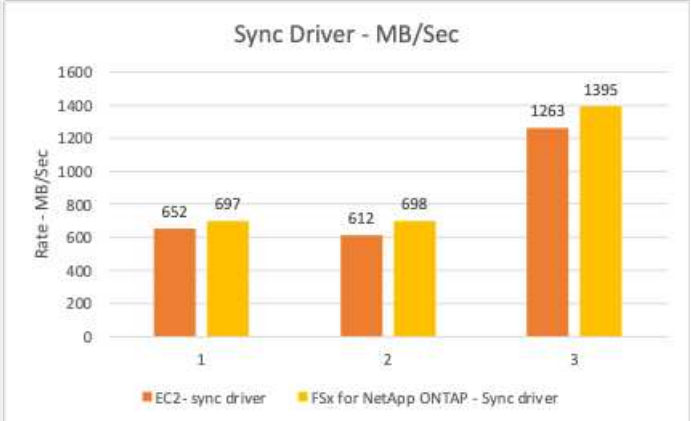

아래 차트는 Kafka 복제 요소 3에 대한 2GB/초 FSx ONTAP 및 4GB/초 성능을 보여줍니다. 복제 요소 3은 FSx ONTAP 스토리지에서 읽기 및 쓰기 작업을 세 번 수행합니다. 처리량 드라이버의 총 속도는 881MB/초이고, 2GB/초 FSx ONTAP 파일 시스템에서 Kafka 작업을 읽고 쓰는 데 약 2.64GB/초가 소요됩니다. 처리량 드라이버의 총 속도는 1328MB/초이고, Kafka 작업을 읽고 쓰는 데 약 3.98GB/초가 소요됩니다. Kafka 성능은 FSx ONTAP 처리량에 따라 선형적이고 확장 가능합니다.




아래 차트는 EC2 인스턴스와 FSx ONTAP (Kafka 복제 계수: 3) 간의 성능을 보여줍니다.

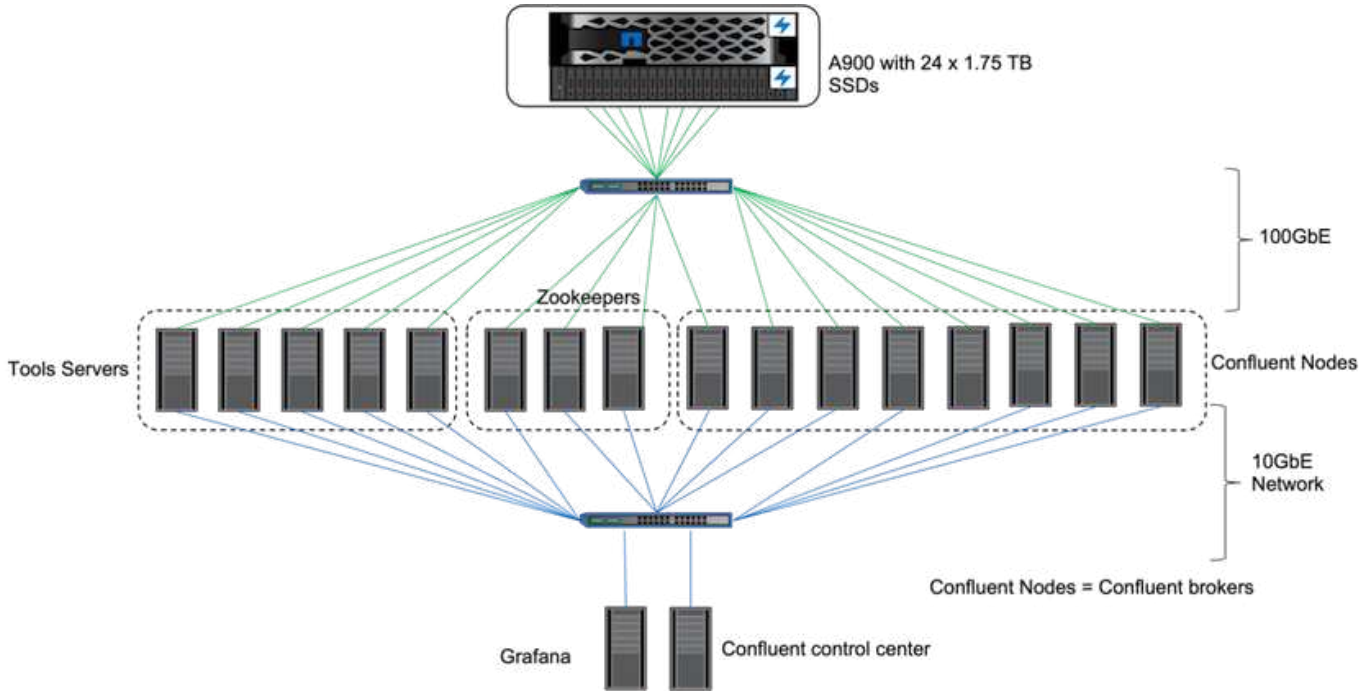


# 온프레미스 **AFF A900** 통한 성능 개요 및 검증

온프레미스에서는 ONTAP 9.12.1RC1과 NetApp AFF A900 스토리지 컨트롤러를 사용하여 Kafka 클러스터의 성능과 확장성을 검증했습니다. 우리는 ONTAP 과 AFF 활용한 이전 계층형 스토리지 모범 사례와 동일한 테스트베드를 사용했습니다.

우리는 Confluent Kafka 6.2.0을 사용하여 AFF A900 평가했습니다. 클러스터는 8개의 브로커 노드와 3개의 주키퍼 노드로 구성됩니다. 성능 테스트를 위해 OMB 작업자 노드 5개를 사용했습니다.




## 스토리지 구성

NetApp FlexGroups 인스턴스를 사용하여 로그 디렉토리에 대한 단일 네임스페이스를 제공하고, 이를 통해 복구 및 구성을 간소화했습니다. 로그 세그먼트 데이터에 대한 직접 경로 액세스를 제공하기 위해 NFSv4.1과 pNFS를 사용했습니다.

## 클라이언트 튜닝

각 클라이언트는 다음 명령을 사용하여 FlexGroup 인스턴스를 마운트했습니다.

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

또한, 우리는 증가했습니다 `max_session_slots` 기본값에서 `64` 에게 `180` . 이는 ONTAP 의 기본 세션 슬롯 제한과 일치합니다.

## 카프카 브로커 튜닝

테스트 중인 시스템의 처리량을 극대화하기 위해 특정 주요 스레드 풀에 대한 기본 매개변수를 크게 늘렸습니다.

대부분의 구성에 대해 Confluent Kafka 모범 사례를 따르는 것이 좋습니다. 이 튜닝은 스토리지에 대한 뛰어난 I/O의 동시성을 극대화하는 데 사용되었습니다. 이러한 매개변수는 브로커의 컴퓨팅 리소스와 스토리지 속성에 맞게 조정할 수 있습니다.

```
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
```

## 워크로드 생성기 테스트 방법론

처리량 드라이버와 주제 구성을 위해 클라우드 테스트와 동일한 OMB 구성을 사용했습니다.

1. FlexGroup 인스턴스는 AFF 클러스터에서 Ansible을 사용하여 프로비저닝되었습니다.

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

2. ONTAP SVM에서 pNFS가 활성화되었습니다.

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

3. 작업 부하는 Cloud Volumes ONTAP 과 동일한 작업 부하 구성을 사용하여 처리량 드라이버로 트리거되었습니다. "섹션을 참조하세요.정상 상태 성능 " 아래에. 작업 부하에는 3의 복제 요소가 사용되었는데, 이는 로그 세그먼트의 3개 사본이 NFS에 유지된다는 것을 의미합니다.

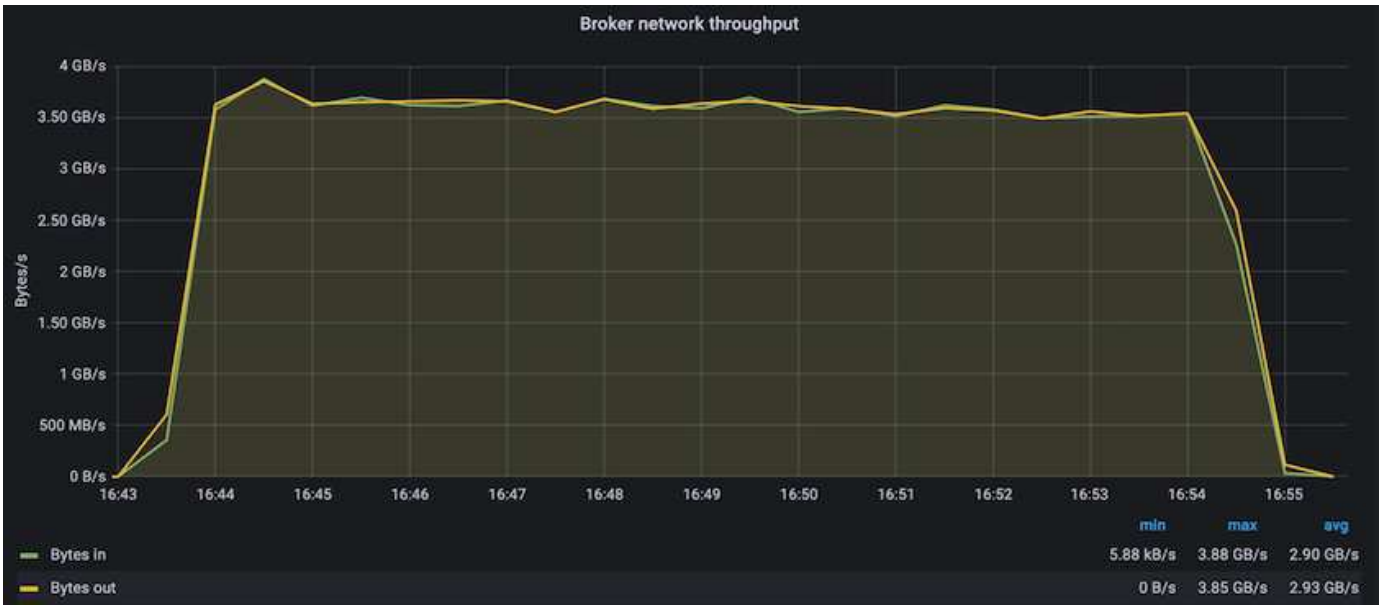
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 마지막으로, 우리는 소비자들이 최신 메시지를 따라잡을 수 있는 능력을 측정하기 위해 백로그를 활용한 측정을 완료했습니다. OMB는 측정 시작 시 소비자의 활동을 일시 정지시켜 백로그를 구성합니다. 이는 세 가지 뚜렷한 단계를 생성합니다. 백로그 생성(생산자 전용 트래픽), 백로그 드레이닝(소비자가 토픽에서 놓친 이벤트를 따라잡는 소비자 중심 단계), 정상 상태입니다. "섹션을 참조하세요.극한의 성능과 저장 한계 탐색 " 자세한 내용은.

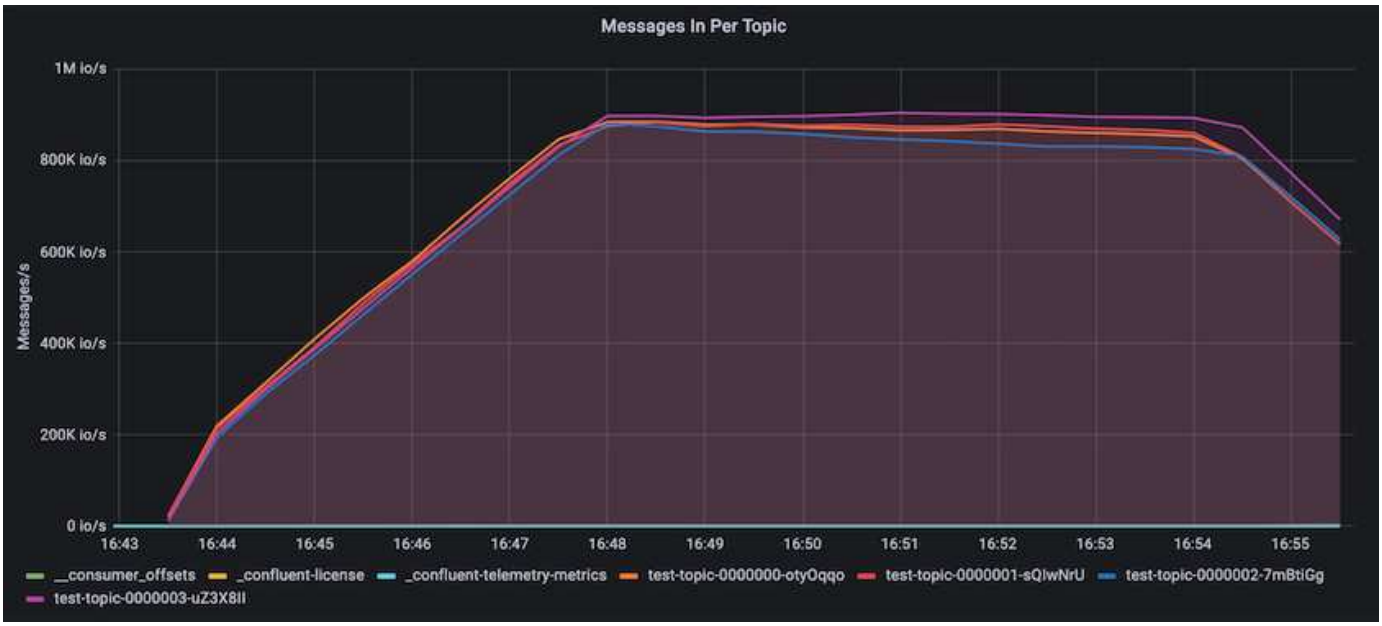
정상 상태 성능

AWS의 Cloud Volumes ONTAP 과 AWS의 DAS에 대한 비교와 유사한 비교를 제공하기 위해 OpenMessaging 벤치마크를 사용하여 AFF A900 평가했습니다. 모든 성능 값은 프로듀서 및 소비자 수준에서의 카프카 클러스터 처리량을 나타냅니다.

Confluent Kafka와 AFF A900 사용한 정상 상태 성능은 제작자와 소비자 모두에서 평균 3.4GBps 이상의 처리량을 달성했습니다. 이는 Kafka 클러스터 전체에 340만 개가 넘는 메시지입니다. BrokerTopicMetrics의 초당 바이트 단위의 지속적인 처리량을 시각화하면 AFF A900 이 지원하는 뛰어난 정상 상태 성능과 트래픽을 확인할 수 있습니다.




이는 주제별로 전달된 메시지의 관점과 잘 일치합니다. 다음 그래프는 주제별 분석을 제공합니다. 테스트한 구성에서는 4개 주제에 걸쳐 주제당 약 90만 개의 메시지를 확인했습니다.

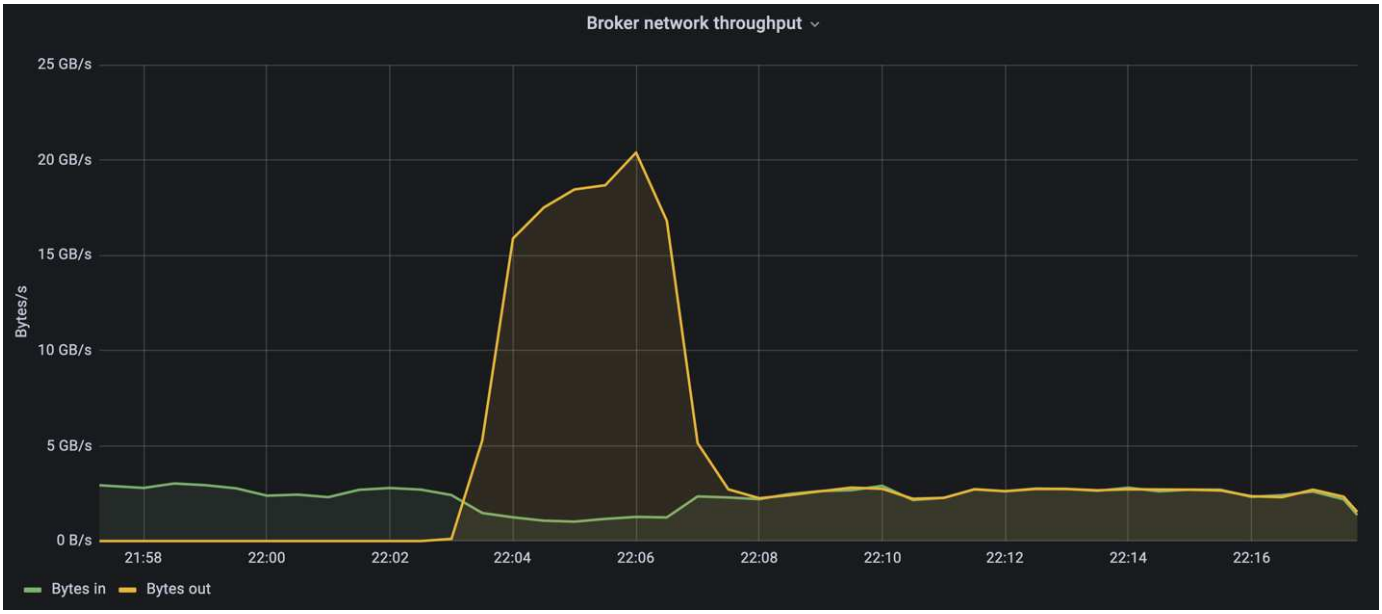



## 극한의 성능과 저장 한계 탐색

AFF 의 경우, 백로그 기능을 사용하여 OMB에서도 테스트했습니다. 백로그 기능은 Kafka 클러스터에 이벤트 백로그가 축적되는 동안 소비자 구독을 일시 중지합니다. 이 단계에서는 프로듀서 트래픽만 발생하며, 이를 통해 로그에 커밋되는 이벤트가 생성됩니다. 이는 일괄 처리나 오프라인 분석 워크플로를 가장 밀접하게 에뮬레이션합니다. 이러한 워크플로에서 소비자 구독이 시작되고 브로커 캐시에서 이미 제거된 과거 데이터를 읽어야 합니다.

이 구성에서 소비자 처리량에 대한 저장 제한을 파악하기 위해 생산자 전용 단계를 측정하여 A900이 얼마나 많은 쓰기 트래픽을 흡수할 수 있는지 파악했습니다. 다음 섹션을 참조하세요.사이즈 가이드 "이 데이터를 활용하는 방법을 이해합니다.

이 측정의 생산자 전용 부분에서 우리는 A900 성능의 한계를 뛰어넘는 높은 피크 처리량을 확인했습니다(생산자 및 소비자 트래픽을 제공하는 다른 브로커 리소스가 포화 상태가 아니었을 때).



이 측정에서는 메시지 당 오버헤드를 제한하고 NFS 마운트 지점에 대한 저장 처리량을 극대화하기 위해 메시지 크기를 16k로 늘렸습니다.

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

Confluent Kafka 클러스터는 최대 프로듀서 처리량 4.03GBps를 달성했습니다.

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

OMB가 이벤트 백로그 채우기를 완료한 후 소비자 트래픽이 다시 시작되었습니다. 백로그 드레이닝을 통한 측정 동안 모든 주제에서 최대 소비자 처리량이 20GBps가 넘는 것을 관찰했습니다. OMB 로그 데이터를 저장하는 NFS 볼륨에 대한 결합 처리량은 ~30GBps에 달했습니다.

## 사이즈 가이드

Amazon Web Services는 다음을 제공합니다. "사이즈 가이드" 카프카 클러스터 크기 조정 및 확장을 위한 것입니다.

이 크기 조정은 Kafka 클러스터의 스토리지 처리량 요구 사항을 결정하는 데 유용한 공식을 제공합니다.

r의 복제 인자를 사용하여 tcluster의 클러스터에 생성된 집계된 처리량의 경우, 브로커 저장소에서 수신한 처리량은 다음과 같습니다.

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

이것은 더욱 단순화될 수 있습니다.

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

이 공식을 사용하면 Kafka 핫 티어 요구 사항에 맞는 적절한 ONTAP 플랫폼을 선택할 수 있습니다.

다음 표는 다양한 복제 요소를 적용한 A900의 예상 생산자 처리량을 설명합니다.

| 복제 인자 | 생산자 처리량**(GPps)** |
|---|---|
| 3(측정됨) | 3.4 |
| 2 | 5.1 |
| 1 | 10.2 |

# 결론

어리석은 이름 변경 문제에 대한 NetApp 솔루션은 이전에는 NFS와 호환되지 않았던 워크로드에 대해 간단하고 저렴하며 중앙에서 관리되는 형태의 스토리지를 제공합니다.

이 새로운 패러다임을 통해 고객은 재해 복구 및 데이터 보호 목적으로 마이그레이션 및 미러링하기 쉬운 Kafka 클러스터를 보다 쉽게 만들 수 있습니다. 또한 NFS는 CPU 사용률 감소, 복구 시간 단축, 스토리지 효율성의 획기적 개선, NetApp ONTAP 통한 성능 향상 등의 추가 이점을 제공한다는 것을 확인했습니다.

# 추가 정보를 찾을 수 있는 곳

이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹사이트를 검토하세요.

- 아파치 카프카란 무엇인가요?

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

- 어리석은 이름 바꾸기란 무엇입니까?

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

- ONATP는 스트리밍 애플리케이션에 사용됩니다.

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

- NetApp 제품 설명서

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

- NFS란 무엇인가요?

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

- 카프카 파티션 재할당이란 무엇인가요?

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

- OpenMessaging 벤치마크란 무엇인가요?

  "https://openmessaging.cloud/"

- 카프카 브로커를 어떻게 마이그레이션하나요?

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

- Prometheus를 사용하여 Kafka 브로커를 어떻게 모니터링합니까?

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

- Apache Kafka를 위한 관리형 플랫폼

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

- Apache Kafka 지원

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

- Apache Kafka 컨설팅 서비스

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