# 최신 데이터 분석

## NetApp Solutions

NetApp
April 20, 2024

# 목차

# NetApp의 최신 데이터 분석 솔루션

## NetApp File-Object 이중화 및 AWS SageMaker를 통한 클라우드 데이터 관리

### TR-4967: NetApp 파일 오브젝트 이중화 및 AWS SageMaker를 통한 클라우드 데이터 관리

NetApp의 Karthikeyan Nagalingam입니다

데이터 과학자와 엔지니어는 NFS 형식으로 저장된 데이터에 액세스해야 하는 경우가 많지만, AWS는 S3 버킷 액세스만 지원하므로 AWS SageMaker의 S3 프로토콜에서 이 데이터에 직접 액세스하는 것은 어려울 수 있습니다. 하지만 NetApp ONTAP는 NFS 및 S3에 대한 이중 프로토콜 액세스를 지원하여 솔루션을 제공합니다. 이 솔루션을 통해 데이터 과학자와 엔지니어는 NetApp Cloud Volumes ONTAP의 S3 버킷을 통해 AWS SageMaker 노트북에서 NFS 데이터에 액세스할 수 있습니다. 이 접근 방식을 사용하면 추가 소프트웨어 없이 NFS 및 S3에서 동일한 데이터에 쉽게 액세스하고 공유할 수 있습니다.

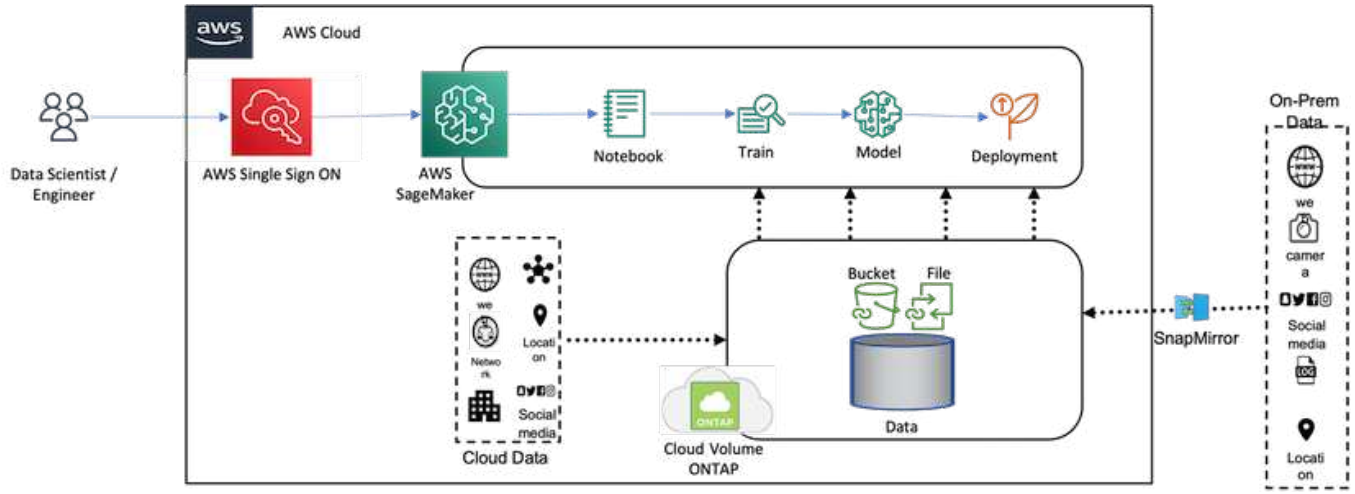
### 솔루션 기술

이 솔루션은 다음 기술을 사용합니다.

- * AWS SageMaker 노트북 * 은 개발자와 데이터 과학자가 고품질 ML 모델을 효율적으로 제작, 교육 및 배포할 수 있는 머신 러닝 기능을 제공합니다.
- * NetApp BlueXP. * 를 사용하면 AWS, Azure, Google Cloud뿐만 아니라 온프레미스에서 스토리지를 검색, 구축, 운영할 수 있습니다. 데이터 손실, 사이버 위협, 계획되지 않은 운영 중단에 대비한 데이터 보호 기능을 제공하고 데이터 스토리지 및 인프라를 최적화합니다.
- * NetApp Cloud Volumes ONTAP. * 는 AWS, Azure 및 Google Cloud에서 NFS, SMB/CIFS, iSCSI, S3 프로토콜을 지원하는 엔터프라이즈급 스토리지 볼륨을 제공하여 사용자가 클라우드에서 데이터에 액세스하고 관리할 수 있는 유연성을 제공합니다.

BlueXP에서 NetApp Cloud Volumes ONTAP을 생성하여 ML 데이터를 저장합니다.

다음 그림은 솔루션의 기술 구성요소를 보여 줍니다.




사용 사례 요약

NFS 및 S3의 이중 프로토콜 액세스 사용 사례는 머신 러닝 및 데이터 과학 분야입니다. 예를 들어, 데이터 과학자 팀이 AWS SageMaker를 사용하여 머신 러닝 프로젝트에 참여하고 있을 수 있으며 이때 NFS 형식으로 저장된 데이터에 액세스해야 합니다. 하지만 다른 팀 구성원과 협업하거나 S3를 사용하는 다른 애플리케이션과 통합하기 위해 S3 버킷을 통해 데이터에 액세스하고 공유해야 할 수도 있습니다.

이 팀은 NetApp Cloud Volumes ONTAP를 활용하여 데이터를 단일 위치에 저장하고 NFS 및 S3 프로토콜 모두에서 액세스할 수 있습니다. 데이터 과학자는 AWS SageMaker에서 직접 NFS 형식의 데이터에 액세스할 수 있는 반면, 다른 팀원이나 애플리케이션은 S3 버킷을 통해 동일한 데이터에 액세스할 수 있습니다.

따라서 서로 다른 스토리지 솔루션 간에 추가 소프트웨어 또는 데이터 마이그레이션을 수행할 필요 없이 데이터를 쉽고 효율적으로 액세스 및 공유할 수 있습니다. 또한 팀 구성원 간의 보다 능률적인 워크플로 및 협업을 통해 머신 러닝 모델을 보다 빠르고 효과적으로 개발할 수 있습니다.

## 데이터 과학자 및 기타 애플리케이션을 위한 데이터 중복

## 데이터는 NFS에서 사용 가능하며 AWS SageMaker에서 S3에서 액세스할 수 있습니다.

기술 요구 사항

데이터 중복 사용 사례를 위해서는 NetApp BlueXP, NetApp Cloud Volumes ONTAP 및 AWS SageMaker 노트북 이 필요합니다.

소프트웨어 요구 사항

다음 표에는 사용 사례를 구현하는 데 필요한 소프트웨어 구성요소가 나와 있습니다.

| 소프트웨어 | 수량 |
|---|---|
| BlueXP | 1 |
| NetApp Cloud Volumes ONTAP를 참조하십시오 | 1 |
| AWS SageMaker 노트북 | 1 |

구현 절차

데이터 이중화 솔루션을 구축하려면 다음 작업이 필요합니다.

- BlueXP 커넥터
- NetApp Cloud Volumes ONTAP를 참조하십시오
- 머신 러닝을 위한 데이터
- AWS SageMaker를 참조하십시오
- Jupyter Notebooks에서 검증된 머신 러닝

#### BlueXP 커넥터

이 검증에서 AWS를 사용했습니다. Azure 및 Google Cloud에도 적용됩니다. AWS에서 BlueXP 커넥터를 생성하려면 다음 단계를 수행하십시오.

1. BlueXP의 mcel-marketplace-subscription을 기반으로 자격 증명을 사용했습니다.
2. 환경에 적합한 지역을 선택하십시오(예: us-east-1[N Virginia])를 선택하고 인증 방법(예: 역할 또는 AWS 키 가정)을 선택합니다. 이 검증에서는 AWS 키를 사용합니다.
3. 커넥터 이름을 제공하고 역할을 생성합니다.
4. 공용 IP가 필요한지 여부에 따라 VPC, 서브넷 또는 키 쌍 등의 네트워크 세부 정보를 제공합니다.
5. HTTP, HTTPS 또는 소스 유형의 SSH 액세스(예: Anywhere 및 IP 범위 정보)와 같은 보안 그룹에 대한 세부 정보를 제공합니다.
6. BlueXP 커넥터를 검토하고 생성합니다.
7. BlueXP EC2 인스턴스 상태가 AWS 콘솔에서 실행 중인지 확인하고 * Networking * 탭에서 IP 주소를 확인합니다.
8. BlueXP 포털에서 커넥터 사용자 인터페이스에 로그인하거나 브라우저에서 IP 주소를 사용하여 액세스할 수 있습니다.

#### NetApp Cloud Volumes ONTAP를 참조하십시오

BlueXP에서 Cloud Volumes ONTAP 인스턴스를 만들려면 다음 단계를 수행하십시오.

1. 새 작업 환경을 생성하고 클라우드 공급자를 선택한 다음 Cloud Volumes ONTAP 인스턴스 유형(예: ONTAP의 경우 단일 CVO, HA 또는 Amazon FSxN)을 선택합니다.
2. Cloud Volumes ONTAP 클러스터 이름 및 자격 증명과 같은 세부 정보를 제공합니다. 이 검증에서 라는 Cloud Volumes ONTAP 인스턴스를 만들었습니다 `svm_sagemaker_cvo_sn1`.
3. Cloud Volumes ONTAP에 필요한 서비스를 선택합니다. 이 검증에서는 모니터링만 하기로 선택했으므로 * 데이터 감지 및 규정 준수 * 와 * 클라우드 서비스로 백업 * 을 비활성화했습니다.
4. Location & Connectivity * 섹션에서 AWS 지역, VPC, 서브넷, 보안 그룹, SSH 인증 방법을 선택합니다. 암호 또는 키 쌍 중 하나를 입력합니다.
5. 충전 방법을 선택합니다. 이 검증에는 * Professional * 을 사용했습니다.
6. POC 및 소규모 워크로드 *, * 데이터베이스 및 애플리케이션 데이터 운영 워크로드 *, * 비용 효율적인 DR * 또는 * 고성능 운영 워크로드 * 와 같은 사전 구성된 패키지를 선택할 수 있습니다. 이 검증에서 우리는 * POC 및 소규모 워크로드 * 를 선택합니다.
7. 특정 크기, 허용되는 프로토콜 및 내보내기 옵션으로 볼륨을 생성합니다. 이 검증에서 라는 볼륨을 생성했습니다

`vol1`.

8. 프로파일 디스크 유형 및 계층화 정책을 선택합니다. 이 검증에서는 * 스토리지 효율성 * 및 * 범용 SSD – 동적 성능 * 을 비활성화했습니다.
9. 마지막으로 Cloud Volumes ONTAP 인스턴스를 검토하고 만듭니다. 그런 다음 BlueXP가 Cloud Volumes ONTAP 작업 환경을 만들 때까지 15-20분 정도 기다립니다.
10. duality 프로토콜을 사용하도록 다음 매개 변수를 구성합니다. 이중화 프로토콜(NFS/S3)은 ONTAP 9에서 지원됩니다. 12.1 이상
    a. 이 검증에서 라는 SVM을 생성했습니다 `svm_sagemaker_cvo_sn1` 볼륨을 높이십시오 `vol1`.
    b. SVM이 NFS 및 S3에 대한 프로토콜 지원을 가지고 있는지 확인합니다. 그렇지 않은 경우 SVM을 수정하여 지원을 받을 수 있습니다.

```
sagemaker_cvo_sn1::> vserver show -vserver svm_sagemaker_cvo_sn1
                                    Vserver: svm_sagemaker_cvo_sn1
                               Vserver Type: data
                            Vserver Subtype: default
                               Vserver UUID: 911065dd-a8bc-11ed-bc24-
e1c0f00ad86b
                                Root Volume:
svm_sagemaker_cvo_sn1_root
                                  Aggregate: aggr1
                                 NIS Domain: -
                  Root Volume Security Style: unix
                                LDAP Client: -
               Default Volume Language Code: C.UTF-8
                            Snapshot Policy: default
                              Data Services: data-cifs, data-
flexcache,
                                             data-iscsi, data-nfs,
                                             data-nvme-tcp
                                    Comment:
                               Quota Policy: default
                List of Aggregates Assigned: aggr1
 Limit on Maximum Number of Volumes allowed: unlimited
                        Vserver Admin State: running
                  Vserver Operational State: running
   Vserver Operational State Stopped Reason: -
                          Allowed Protocols: nfs, cifs, fcp, iscsi,
ndmp, s3
                       Disallowed Protocols: nvme
            Is Vserver with Infinite Volume: false
                           QoS Policy Group: -
                        Caching Policy Name: -
                                Config Lock: false
                               IPspace Name: Default
                         Foreground Process: -
                    Logical Space Reporting: true
                  Logical Space Enforcement: false
Default Anti_ransomware State of the Vserver's Volumes: disabled
            Enable Analytics on New Volumes: false
    Enable Activity Tracking on New Volumes: false

sagemaker_cvo_sn1::>
```

11. 필요한 경우 CA 인증서를 만들어 설치합니다.
12. 서비스 데이터 정책을 생성합니다.

```
sagemaker_cvo_sn1::*> network interface service-policy create -vserver
svm_sagemaker_cvo_sn1 -policy sagemaker_s3_nfs_policy -services data-
core,data-s3-server,data-nfs,data-flexcache
sagemaker_cvo_sn1::*> network interface create -vserver
svm_sagemaker_cvo_sn1 -lif svm_sagemaker_cvo_sn1_s3_lif -service-policy
sagemaker_s3_nfs_policy -home-node sagemaker_cvo_sn1-01 -address
172.30.10.41 -netmask 255.255.255.192

Warning: The configured failover-group has no valid failover targets for
the LIF's failover-policy. To view the failover targets for a LIF, use
         the "network interface show -failover" command.

sagemaker_cvo_sn1::*>
sagemaker_cvo_sn1::*> network interface show
Logical    Status     Network            Current       Current Is
Vserver     Interface  Admin/Oper Address/Mask       Node          Port
Home
----------- ---------- ---------- ------------------ -------------
------- ----
sagemaker_cvo_sn1
            cluster-mgmt up/up    172.30.10.40/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            intercluster up/up    172.30.10.48/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            sagemaker_cvo_sn1-01_mgmt1
                         up/up    172.30.10.58/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
svm_sagemaker_cvo_sn1
            svm_sagemaker_cvo_sn1_data_lif
                         up/up    172.30.10.23/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            svm_sagemaker_cvo_sn1_mgmt_lif
                         up/up    172.30.10.32/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            svm_sagemaker_cvo_sn1_s3_lif
                         up/up    172.30.10.41/26    sagemaker_cvo_sn1-
```

```
01
                                                                    e0a
true
6 entries were displayed.

sagemaker_cvo_sn1::*>
sagemaker_cvo_sn1::*> vserver object-store-server create -vserver
svm_sagemaker_cvo_sn1  -is-http-enabled true -object-store-server
svm_sagemaker_cvo_s3_sn1 -is-https-enabled false
sagemaker_cvo_sn1::*> vserver object-store-server show

Vserver: svm_sagemaker_cvo_sn1

           Object Store Server Name: svm_sagemaker_cvo_s3_sn1
               Administrative State: up
                        HTTP Enabled: true
             Listener Port For HTTP: 80
                       HTTPS Enabled: false
     Secure Listener Port For HTTPS: 443
  Certificate for HTTPS Connections: -
                 Default UNIX User: pcuser
               Default Windows User: -
                            Comment:

sagemaker_cvo_sn1::*>
```

13. 집계 세부 정보를 확인합니다.

```
sagemaker_cvo_sn1::*> aggr show


Aggregate     Size Available Used% State   #Vols  Nodes            RAID
Status
--------- -------- --------- ----- ------- ------ ----------------
------------
aggr0_sagemaker_cvo_sn1_01
           124.0GB   50.88GB   59% online       1 sagemaker_cvo_
raid0,
                                                   sn1-01
normal
aggr1      907.1GB   904.9GB    0% online       2 sagemaker_cvo_
raid0,
                                                   sn1-01
normal
2 entries were displayed.

sagemaker_cvo_sn1::*>
```

14. 사용자 및 그룹을 생성합니다.

```
sagemaker_cvo_sn1::*> vserver object-store-server user create -vserver
svm_sagemaker_cvo_sn1 -user s3user

sagemaker_cvo_sn1::*> vserver object-store-server user show
Vserver     User            ID        Access Key          Secret Key
----------- --------------- --------- -------------------
-------------------
svm_sagemaker_cvo_sn1
            root            0         -                   -
   Comment: Root User
svm_sagemaker_cvo_sn1
            s3user          1         0ZNAX21JW5Q8AP80CQ2E

PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr
2 entries were displayed.

sagemaker_cvo_sn1::*>


sagemaker_cvo_sn1::*> vserver object-store-server group create -name
s3group -users s3user -comment ""

sagemaker_cvo_sn1::*>
sagemaker_cvo_sn1::*> vserver object-store-server group delete -gid 1
-vserver svm_sagemaker_cvo_sn1

sagemaker_cvo_sn1::*> vserver object-store-server group create -name
s3group -users s3user -comment "" -policies FullAccess

sagemaker_cvo_sn1::*>
```

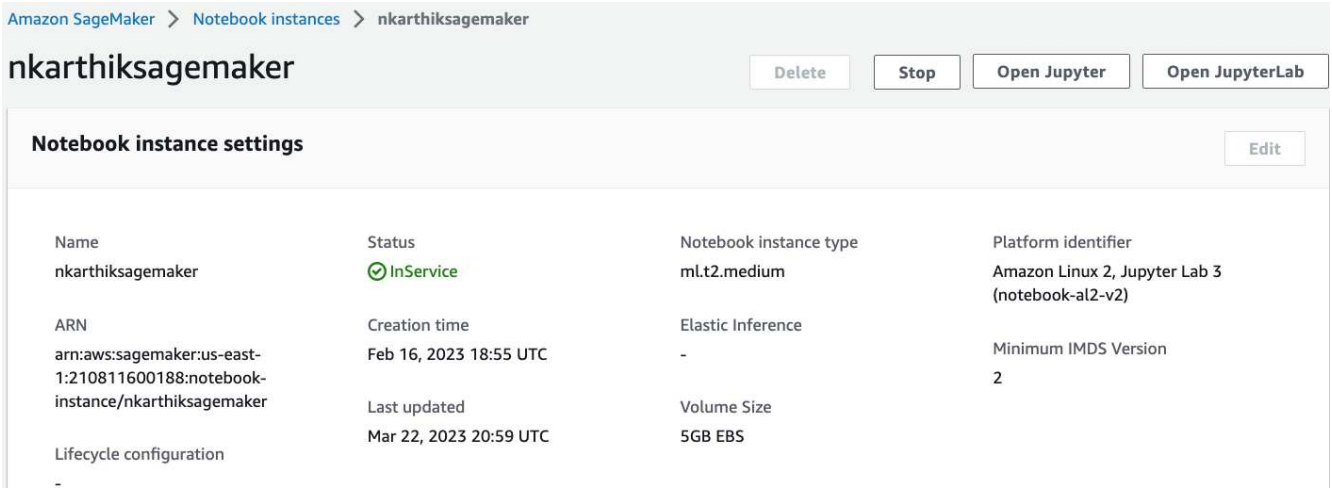
15. NFS 볼륨에 버킷을 생성합니다.

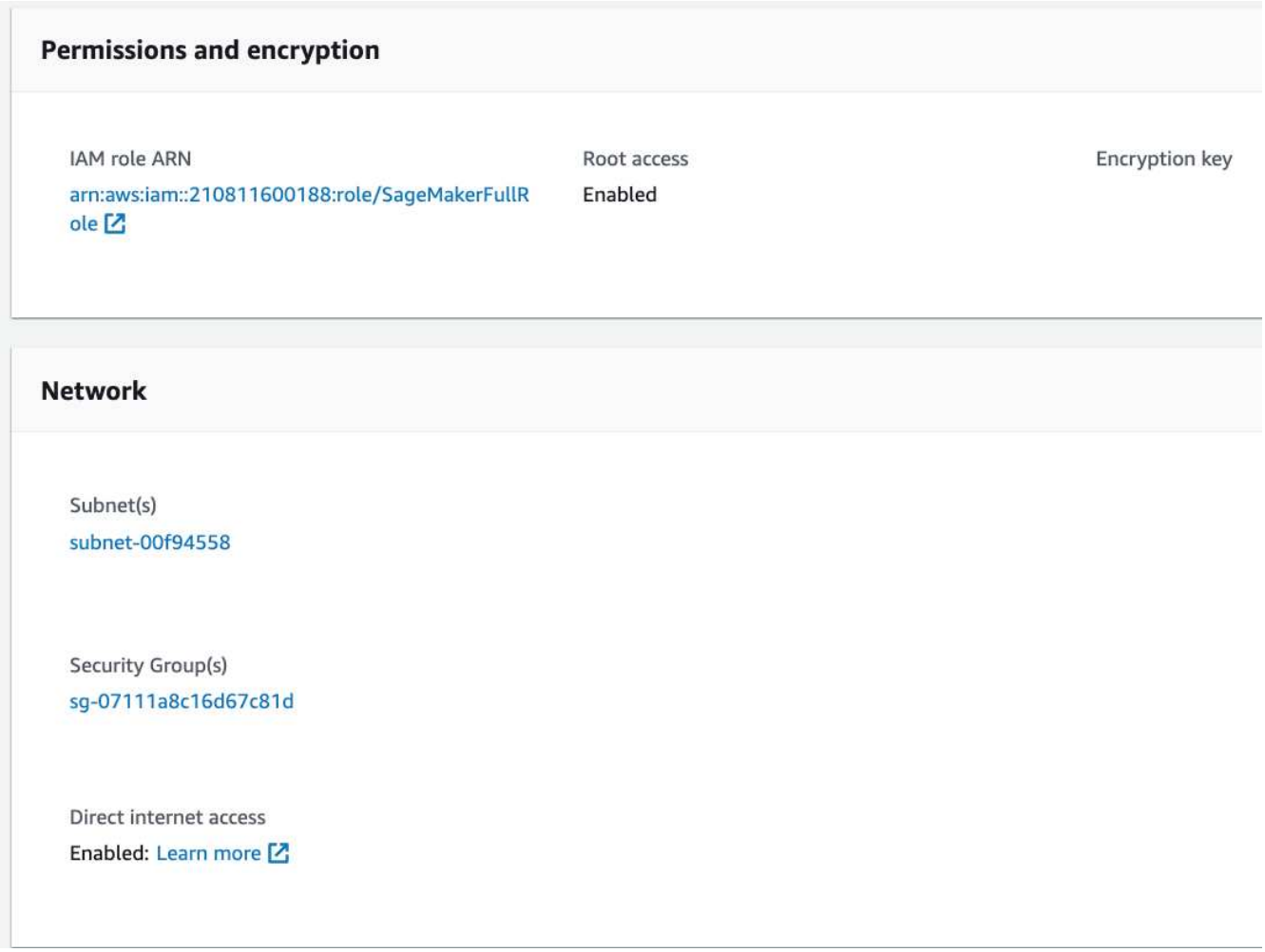
```
sagemaker_cvo_sn1::*> vserver object-store-server bucket create -bucket
ontapbucket1 -type nas -comment "" -vserver svm_sagemaker_cvo_sn1 -nas
-path /vol1
sagemaker_cvo_sn1::*> vserver object-store-server bucket show
Vserver     Bucket          Type     Volume            Size
Encryption Role       NAS Path
----------- --------------- -------- ----------------- ----------
---------- ---------- ----------
svm_sagemaker_cvo_sn1
            ontapbucket1    nas      vol1              -          false
-          /vol1
sagemaker_cvo_sn1::*>
```

**AWS SageMaker**를 참조하십시오

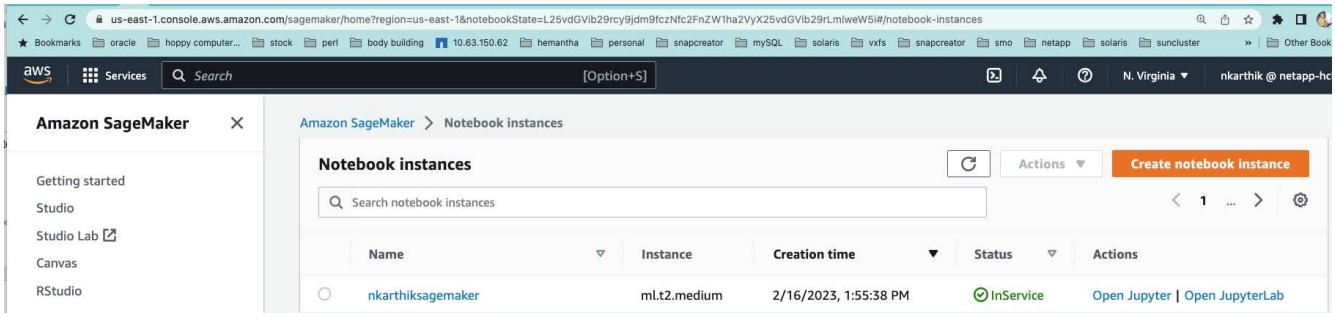
AWS SageMaker에서 AWS 노트북을 만들려면 다음 단계를 수행하십시오.

1. 노트북 인스턴스를 만드는 사용자에게 AmazonSageMakerFullAccess IAM 정책이 있거나 AmazonSageMakerFullAccess 권한이 있는 기존 그룹에 속하는지 확인하십시오. 이 검증에서 사용자는 기존 그룹의 일부입니다.
2. 다음 정보를 제공합니다.
   - 전자 필기장 인스턴스 이름입니다.
   - 인스턴스 유형.
   - 플랫폼 식별자입니다.
   - AmazonSageMakerFullAccess 권한이 있는 IAM 역할을 선택합니다.
   - 루트 액세스 – 설정.
   - 암호화 키 - 사용자 정의 암호화 없음을 선택합니다.
   - 나머지 기본 옵션을 유지합니다.
3. 이 검증에서 SageMaker 인스턴스 세부 정보는 다음과 같습니다.

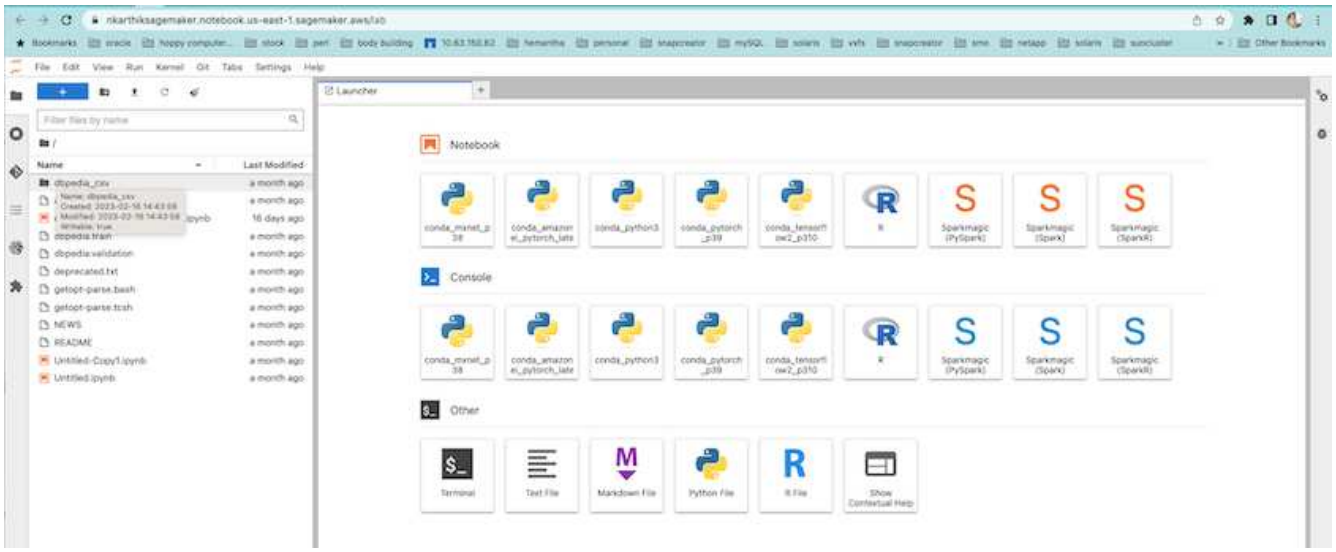





4. AWS 노트북을 시작합니다.




5. Jupyter 연구실을 엽니다.




6. 터미널에 로그인하여 Cloud Volumes ONTAP 볼륨을 마운트합니다.

```
sh-4.2$ sudo mkdir /vol1; sudo mount -t nfs 172.30.10.41:/vol1 /vol1
sh-4.2$ df -h
Filesystem          Size  Used Avail Use% Mounted on
devtmpfs            2.0G     0  2.0G   0% /dev
tmpfs               2.0G     0  2.0G   0% /dev/shm
tmpfs               2.0G  624K  2.0G   1% /run
tmpfs               2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/xvda1          140G  114G   27G  82% /
/dev/xvdf           4.8G   72K  4.6G   1% /home/ec2-user/SageMaker
tmpfs               393M     0  393M   0% /run/user/1001
tmpfs               393M     0  393M   0% /run/user/1002
tmpfs               393M     0  393M   0% /run/user/1000
172.30.10.41:/vol1  973M  189M  785M  20% /vol1
sh-4.2$
```

7. AWS CLI 명령을 사용하여 Cloud Volumes ONTAP 볼륨에 생성된 버킷을 확인합니다.

```
sh-4.2$ aws configure --profile netapp
AWS Access Key ID [None]: 0ZNAX21JW5Q8AP80CQ2E
AWS Secret Access Key [None]: PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr
Default region name [None]: us-east-1
Default output format [None]:
sh-4.2$

sh-4.2$ aws s3 ls --profile netapp --endpoint-url
2023-02-10 17:59:48 ontapbucket1

sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/


2023-02-10 18:46:44       4747 1
2023-02-10 18:48:32         96 setup.cfg

sh-4.2$
```

머신 러닝을 위한 데이터

이 검증에서 우리는 다양한 Wikimedia 프로젝트에서 생성된 정보에서 구조화된 컨텐츠를 추출하기 위해 군중의 노력을 기울인 DBpedia의 데이터 세트를 사용했습니다.

1. DBpedia GitHub 위치에서 데이터를 다운로드하고 압축을 풉니다. 이전 단원에서 사용한 것과 동일한 터미널을 사용합니다.

```
sh-4.2$ wget
--2023-02-14 23:12:11--
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:  [following]
--2023-02-14 23:12:11--
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 68431223 (65M) [application/octet-stream]
Saving to: ‘dbpedia_csv.tar.gz’

100%[=====================================================================
=====================================================================
===================>] 68,431,223  56.2MB/s   in 1.2s

2023-02-14 23:12:13 (56.2 MB/s) - ‘dbpedia_csv.tar.gz’ saved
[68431223/68431223]

sh-4.2$ tar -zxvf dbpedia_csv.tar.gz
dbpedia_csv/
dbpedia_csv/test.csv
dbpedia_csv/classes.txt
dbpedia_csv/train.csv
dbpedia_csv/readme.txt
sh-4.2$
```

2. 데이터를 Cloud Volumes ONTAP 위치로 복사하고 AWS CLI를 사용하여 S3 버킷에서 확인합니다.

```
sh-4.2$ df -h
Filesystem          Size  Used Avail Use% Mounted on
devtmpfs            2.0G     0  2.0G   0% /dev
tmpfs               2.0G     0  2.0G   0% /dev/shm
tmpfs               2.0G  628K  2.0G   1% /run
tmpfs               2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/xvda1          140G  114G   27G  82% /
/dev/xvdf           4.8G   52K  4.6G   1% /home/ec2-user/SageMaker
tmpfs               393M     0  393M   0% /run/user/1002
tmpfs               393M     0  393M   0% /run/user/1001
tmpfs               393M     0  393M   0% /run/user/1000
172.30.10.41:/vol1  973M  384K  973M   1% /vol1
sh-4.2$ pwd
/home/ec2-user
sh-4.2$ cp -ra dbpedia_csv /vol1
sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/
                           PRE dbpedia_csv/
2023-02-10 18:46:44       4747 1
2023-02-10 18:48:32         96 setup.cfg
sh-4.2$
```

3. 기본 검증을 수행하여 읽기/쓰기 기능이 S3 버킷에서 작동하는지 확인합니다.

```
sh-4.2$ aws s3 cp  --profile netapp --endpoint-url  /usr/share/doc/util-
linux-2.30.2 s3://ontapbucket1/ --recursive
upload: ../../../usr/share/doc/util-linux-2.30.2/deprecated.txt to
s3://ontapbucket1/deprecated.txt
upload: ../../../usr/share/doc/util-linux-2.30.2/getopt-parse.bash to
s3://ontapbucket1/getopt-parse.bash
upload: ../../../usr/share/doc/util-linux-2.30.2/README to
s3://ontapbucket1/README
upload: ../../../usr/share/doc/util-linux-2.30.2/getopt-parse.tcsh to
s3://ontapbucket1/getopt-parse.tcsh
upload: ../../../usr/share/doc/util-linux-2.30.2/AUTHORS to
s3://ontapbucket1/AUTHORS
upload: ../../../usr/share/doc/util-linux-2.30.2/NEWS to
s3://ontapbucket1/NEWS
sh-4.2$ aws s3 ls --profile netapp --endpoint-url
s3://ontapbucket1/s3://ontapbucket1/

An error occurred (InternalError) when calling the ListObjectsV2
operation: We encountered an internal error. Please try again.
sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/
                           PRE dbpedia_csv/
```

```
2023-02-16 19:19:27      26774 AUTHORS
2023-02-16 19:19:27      72727 NEWS
2023-02-16 19:19:27       4493 README
2023-02-16 19:19:27       2825 deprecated.txt
2023-02-16 19:19:27       1590 getopt-parse.bash
2023-02-16 19:19:27       2245 getopt-parse.tcsh
sh-4.2$ ls -ltr /vol1
total 132
drwxrwxr-x 2 ec2-user ec2-user  4096 Mar 29  2015 dbpedia_csv
-rw-r--r-- 1 nobody   nobody    2245 Apr 10 17:37 getopt-parse.tcsh
-rw-r--r-- 1 nobody   nobody    2825 Apr 10 17:37 deprecated.txt
-rw-r--r-- 1 nobody   nobody    4493 Apr 10 17:37 README
-rw-r--r-- 1 nobody   nobody    1590 Apr 10 17:37 getopt-parse.bash
-rw-r--r-- 1 nobody   nobody   26774 Apr 10 17:37 AUTHORS
-rw-r--r-- 1 nobody   nobody   72727 Apr 10 17:37 NEWS
sh-4.2$ ls -ltr /vol1/dbpedia_csv/
total 192104
-rw------- 1 ec2-user ec2-user 174148970 Mar 28  2015 train.csv
-rw------- 1 ec2-user ec2-user  21775285 Mar 28  2015 test.csv
-rw------- 1 ec2-user ec2-user       146 Mar 28  2015 classes.txt
-rw-rw-r-- 1 ec2-user ec2-user      1758 Mar 29  2015 readme.txt
sh-4.2$ chmod -R 777 /vol1/dbpedia_csv
sh-4.2$ ls -ltr /vol1/dbpedia_csv/
total 192104
-rwxrwxrwx 1 ec2-user ec2-user 174148970 Mar 28  2015 train.csv
-rwxrwxrwx 1 ec2-user ec2-user  21775285 Mar 28  2015 test.csv
-rwxrwxrwx 1 ec2-user ec2-user       146 Mar 28  2015 classes.txt
-rwxrwxrwx 1 ec2-user ec2-user      1758 Mar 29  2015 readme.txt
sh-4.2$ aws s3 cp --profile netapp --endpoint-url http://172.30.2.248/
s3://ontapbucket1/ /tmp --recursive
download: s3://ontapbucket1/AUTHORS to ../../tmp/AUTHORS
download: s3://ontapbucket1/README to ../../tmp/README
download: s3://ontapbucket1/NEWS to ../../tmp/NEWS
download: s3://ontapbucket1/dbpedia_csv/classes.txt to
../../tmp/dbpedia_csv/classes.txt
download: s3://ontapbucket1/dbpedia_csv/readme.txt to
../../tmp/dbpedia_csv/readme.txt
download: s3://ontapbucket1/deprecated.txt to ../../tmp/deprecated.txt
download: s3://ontapbucket1/getopt-parse.bash to ../../tmp/getopt-
parse.bash
download: s3://ontapbucket1/getopt-parse.tcsh to ../../tmp/getopt-
parse.tcsh
download: s3://ontapbucket1/dbpedia_csv/test.csv to
../../tmp/dbpedia_csv/test.csv
download: s3://ontapbucket1/dbpedia_csv/train.csv to
../../tmp/dbpedia_csv/train.csv
```

```
sh-4.2$
sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/
                           PRE dbpedia_csv/
2023-02-16 19:19:27      26774 AUTHORS
2023-02-16 19:19:27      72727 NEWS
2023-02-16 19:19:27       4493 README
2023-02-16 19:19:27       2825 deprecated.txt
2023-02-16 19:19:27       1590 getopt-parse.bash
2023-02-16 19:19:27       2245 getopt-parse.tcsh
sh-4.2$
```

**Jupyter Notebooks**에서 머신 러닝을 검증합니다

다음 검증에서는 아래의 SageMaker BlazingText 예제를 사용하여 텍스트 분류를 통해 머신 러닝 빌드, 교육 및 배포 모델을 제공합니다.

1. boto3 및 SageMaker 패키지를 설치합니다.

```
In [1]:  pip install --upgrade boto3 sagemaker
```

출력:

```
Looking in indexes: https://pypi.org/simple,
https://pip.repos.neuron.amazo naws.com
Requirement already satisfied: boto3 in /home/ec2-
user/anaconda3/envs/pytho n3/lib/python3.10/site-packages (1.26.44)
Collecting boto3
  Downloading boto3-1.26.72-py3-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
132.7/132.7 kB 14.6 MB/s eta 0: 00:00
Requirement already satisfied: sagemaker in /home/ec2-
user/anaconda3/envs/p ython3/lib/python3.10/site-packages (2.127.0)
Collecting sagemaker
  Downloading sagemaker-2.132.0.tar.gz (668 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
668.0/668.0 kB 12.3 MB/s eta 0:
00:0000:01
  Preparing metadata (setup.py) ... done
Collecting botocore<1.30.0,>=1.29.72
  Downloading botocore-1.29.72-py3-none-any.whl (10.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
10.4/10.4 MB 44.3 MB/s eta 0: 00:0000:010:01
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in /home/ec2-
user/a naconda3/envs/python3/lib/python3.10/site-packages (from boto3)
```

```
(0.6.0)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /home/ec2-
user/ana conda3/envs/python3/lib/python3.10/site-packages (from boto3)
(0.10.0)
Requirement already satisfied: attrs<23,>=20.3.0 in /home/ec2-
user/anaconda
3/envs/python3/lib/python3.10/site-packages (from sagemaker) (22.1.0)
Requirement already satisfied: google-pasta in /home/ec2-
user/anaconda3/env s/python3/lib/python3.10/site-packages (from
sagemaker) (0.2.0)
Requirement already satisfied: numpy<2.0,>=1.9.0 in /home/ec2-
user/anaconda
3/envs/python3/lib/python3.10/site-packages (from sagemaker) (1.22.4)
Requirement already satisfied: protobuf<4.0,>=3.1 in /home/ec2-
user/anacond a3/envs/python3/lib/python3.10/site-packages (from
sagemaker) (3.20.3)
Requirement already satisfied: protobuf3-to-dict<1.0,>=0.1.5 in
/home/ec2-u ser/anaconda3/envs/python3/lib/python3.10/site-packages
(from sagemaker)
(0.1.5)
Requirement already satisfied: smdebug_rulesconfig==1.0.1 in /home/ec2-
use r/anaconda3/envs/python3/lib/python3.10/site-packages (from
sagemaker) (1.
0.1) Requirement already satisfied: importlib-metadata<5.0,>=1.4.0 in
/home/ec2user/anaconda3/envs/python3/lib/python3.10/site-packages (from
sagemaker)
(4.13.0)
Requirement already satisfied: packaging>=20.0 in /home/ec2-
user/anaconda3/ envs/python3/lib/python3.10/site-packages (from
sagemaker) (21.3)
Requirement already satisfied: pandas in /home/ec2-
user/anaconda3/envs/pyth on3/lib/python3.10/site-packages (from
sagemaker) (1.5.1)
Requirement already satisfied: pathos in /home/ec2-
user/anaconda3/envs/pyth on3/lib/python3.10/site-packages (from
sagemaker) (0.3.0)
Requirement already satisfied: schema in /home/ec2-
user/anaconda3/envs/pyth on3/lib/python3.10/site-packages (from
sagemaker) (0.7.5) Requirement already satisfied: python-
dateutil<3.0.0,>=2.1 in /home/ec2-use
r/anaconda3/envs/python3/lib/python3.10/site-packages (from
botocore<1.30.
0,>=1.29.72->boto3) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-
user/anac onda3/envs/python3/lib/python3.10/site-packages (from
botocore<1.30.0,>=1.2
```

```
9.72->boto3) (1.26.8) Requirement already satisfied: zipp>=0.5 in
/home/ec2-user/anaconda3/envs/p ython3/lib/python3.10/site-packages
(from importlib-metadata<5.0,>=1.4.0->s agemaker) (3.10.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/ec2-
user/a naconda3/envs/python3/lib/python3.10/site-packages (from
packaging>=20.0->s agemaker) (3.0.9)
Requirement already satisfied: six in /home/ec2-
user/anaconda3/envs/python
3/lib/python3.10/site-packages (from protobuf3-to-dict<1.0,>=0.1.5-
>sagemak er) (1.16.0)
Requirement already satisfied: pytz>=2020.1 in /home/ec2-
user/anaconda3/env s/python3/lib/python3.10/site-packages (from pandas-
>sagemaker) (2022.5)
Requirement already satisfied: ppft>=1.7.6.6 in /home/ec2-
user/anaconda3/en vs/python3/lib/python3.10/site-packages (from pathos-
>sagemaker) (1.7.6.6) Requirement already satisfied:
multiprocess>=0.70.14 in /home/ec2-user/anac
onda3/envs/python3/lib/python3.10/site-packages (from pathos->sagemaker)
(0.70.14)
Requirement already satisfied: dill>=0.3.6 in /home/ec2-
user/anaconda3/env s/python3/lib/python3.10/site-packages (from pathos-
>sagemaker) (0.3.6)
Requirement already satisfied: pox>=0.3.2 in /home/ec2-
user/anaconda3/envs/ python3/lib/python3.10/site-packages (from pathos-
>sagemaker) (0.3.2) Requirement already satisfied: contextlib2>=0.5.5 in
/home/ec2-user/anacond a3/envs/python3/lib/python3.10/site-packages
(from schema->sagemaker) (21.
6.0) Building wheels for collected packages: sagemaker
  Building wheel for sagemaker (setup.py) ... done
  Created wheel for sagemaker: filename=sagemaker-2.132.0-py2.py3-none-
any. whl size=905449
sha256=f6100a5dc95627f2e2a49824e38f0481459a27805ee19b5a06ec
83db0252fd41
  Stored in directory: /home/ec2-
user/.cache/pip/wheels/60/41/b6/482e7ab096
520df034fbf2dddd244a1d7ba0681b27ef45aa61
Successfully built sagemaker
Installing collected packages: botocore, boto3, sagemaker
  Attempting uninstall: botocore     Found existing installation:
botocore 1.24.19
    Uninstalling botocore-1.24.19:       Successfully uninstalled
botocore-1.24.19
  Attempting uninstall: boto3     Found existing installation: boto3
1.26.44
    Uninstalling boto3-1.26.44:
      Successfully uninstalled boto3-1.26.44
```

```
  Attempting uninstall: sagemaker     Found existing installation:
sagemaker 2.127.0
    Uninstalling sagemaker-2.127.0:
      Successfully uninstalled sagemaker-2.127.0
ERROR: pip's dependency resolver does not currently take into account
all t he packages that are installed. This behaviour is the source of
the followi ng dependency conflicts.
awscli 1.27.44 requires botocore==1.29.44, but you have botocore 1.29.72
wh ich is incompatible.
aiobotocore 2.0.1 requires botocore<1.22.9,>=1.22.8, but you have
botocore 1.29.72 which is incompatible. Successfully installed boto3-
1.26.72 botocore-1.29.72 sagemaker-2.132.0 Note: you may need to restart
the kernel to use updated packages.
```

2. 다음 단계에서는 데이터를 입력합니다 (`dbpedia_csv`)가 S3 버킷에서 다운로드됩니다 `ontapbucket1` 기계 학습에 사용되는 Jupyter Notebook 인스턴스에.

```
In [2]: import sagemaker
In [3]: from sagemaker import get_execution_role
In [4]:
import json
import boto3
sess = sagemaker.Session()
role = get_execution_role()
print(role)
bucket = "ontapbucket1"
print(bucket)
sess.s3_client = boto3.client('s3',region_name='',aws_access_key_id =
'0ZNAX21JW5Q8AP80CQ2E',  aws_secret_access_key =
'PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr',
                              use_ssl = False, endpoint_url =
'http://172.30.10.41',

config=boto3.session.Config(signature_version='s3v4',
s3={'addressing_style':'path'}) )
sess.s3_resource = boto3.resource('s3',region_name='',aws_access_key_id
= '0ZNAX21JW5Q8AP80CQ2E', aws_secret_access_key =
'PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr',
                              use_ssl = False, endpoint_url =
'http://172.30.10.41',

config=boto3.session.Config(signature_version='s3v4',
s3={'addressing_style':'path'}) )
prefix = "blazingtext/supervised"
import os
my_bucket = sess.s3_resource.Bucket(bucket)
my_bucket = sess.s3_resource.Bucket(bucket)
#os.mkdir('dbpedia_csv')
for s3_object in my_bucket.objects.all():
    filename = s3_object.key
#    print(filename)
#    print(s3_object.key)
    my_bucket.download_file(s3_object.key, filename)
```

3. 다음 코드에서는 추론 중에 실제 클래스 이름을 검색하는 데 사용되는 클래스 레이블에 대한 정수 인덱스의 매핑을 만듭니다.

```
index_to_label = {}
with open("dbpedia_csv/classes.txt") as f:
    for i,label in enumerate(f.readlines()):
        index_to_label[str(i + 1)] = label.strip()
```

출력에는 의 파일과 폴더가 나열됩니다 `ontapbucket1` AWS SageMaker 머신 러닝 검증을 위한 데이터로 사용되는 버킷

```
arn:aws:iam::210811600188:role/SageMakerFullRole ontapbucket1
AUTHORS
AUTHORS
NEWS
NEWS
README README
dbpedia_csv/classes.txt dbpedia_csv/classes.txt dbpedia_csv/readme.txt
dbpedia_csv/readme.txt dbpedia_csv/test.csv dbpedia_csv/test.csv
dbpedia_csv/train.csv dbpedia_csv/train.csv deprecated.txt
deprecated.txt getopt-parse.bash getopt-parse.bash getopt-parse.tcsh
getopt-parse.tcsh
In [5]: ls
AUTHORS       deprecated.txt     getopt-parse.tcsh  NEWS
Untitled.ipynb dbpedia_csv/  getopt-parse.bash  lost+found/
README
In [6]: ls -l dbpedia_csv
total 191344
-rw-rw-r-- 1 ec2-user ec2-user       146 Feb 16 19:43 classes.txt
-rw-rw-r-- 1 ec2-user ec2-user      1758 Feb 16 19:43 readme.txt
-rw-rw-r-- 1 ec2-user ec2-user  21775285 Feb 16 19:43 test.csv
-rw-rw-r-- 1 ec2-user ec2-user 174148970 Feb 16 19:43 train.csv
```

4. 데이터 전처리 단계를 시작하여 BlazingText 알고리즘과 nltk 라이브러리에서 사용할 수 있는 공간 분리 토큰화된 텍스트 형식으로 교육 데이터를 사전 처리하여 DBPedia 데이터 세트의 입력 문장을 토큰화합니다. nltk 토큰화 및 기타 라이브러리를 다운로드합니다. 를 클릭합니다 `transform_instance` 병렬로 각 데이터 인스턴스에 적용되는 Python 다중 처리 모듈을 사용합니다.

```
ln [7]: from random import shuffle
import multiprocessing
from multiprocessing import Pool
import csv
import nltk
nltk.download("punkt")
def transform_instance(row):
    cur_row = []
    label ="__label__" + index_to_label [row[0]] # Prefix the index-ed
label with __label__
    cur_row.append (label)
    cur_row.extend(nltk.word_tokenize(row[1].lower ()))
    cur_row.extend(nltk.word_tokenize(row[2].lower ()))
    return cur_row
def preprocess(input_file, output_file, keep=1):
```

```
    all_rows = []
    with open(input_file,"r") as csvinfile:
        csv_reader = csv.reader(csvinfile, delimiter=",")
        for row in csv_reader:
            all_rows.append(row)
    shuffle(all_rows)
    all_rows = all_rows[: int(keep * len(all_rows))]
    pool = Pool(processes=multiprocessing.cpu_count())
    transformed_rows = pool.map(transform_instance, all_rows)
    pool.close()
    pool. join()
    with open(output_file, "w") as csvoutfile:
        csv_writer = csv.writer (csvoutfile, delimiter=" ",
lineterminator="\n")
        csv_writer.writerows (transformed_rows)

# Preparing the training dataset
# since preprocessing the whole dataset might take a couple of minutes,
# we keep 20% of the training dataset for this demo.
# Set keep to 1 if you want to use the complete dataset
preprocess("dbpedia_csv/train.csv","dbpedia.train", keep=0.2)
# Preparing the validation dataset
preprocess("dbpedia_csv/test.csv","dbpedia.validation")
sess = sagemaker.Session()
role = get_execution_role()
print (role) # This is the role that sageMaker would use to leverage Aws
resources (S3,  Cloudwatch) on your behalf
bucket = sess.default_bucket() # Replace with your own bucket name if
needed
print("default Bucket::: ")
print(bucket)
```

출력:

```
[nltk_data] Downloading package punkt to /home/ec2-user/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
arn:aws:iam::210811600188:role/SageMakerFullRole default Bucket:::
sagemaker-us-east-1-210811600188
```

5. SageMaker에서 교육 작업을 실행하는 데 사용할 수 있도록 형식 지정된 교육 데이터 세트를 S3에 업로드합니다. 그런 다음 Python SDK를 사용하여 버킷과 접두사 위치에 두 개의 파일을 업로드합니다.

```
ln [8]: %%time
train_channel = prefix + "/train"
validation_channel = prefix + "/validation"
sess.upload_data(path="dbpedia.train", bucket=bucket,
key_prefix=train_channel)
sess.upload_data(path="dbpedia.validation", bucket=bucket,
key_prefix=validation_channel)
s3_train_data = "s3://{}/{}".format(bucket, train_channel)
s3_validation_data = "s3://{}/{}".format(bucket, validation_channel)
```

출력:

```
CPU times: user 546 ms, sys: 163 ms, total: 709 ms
Wall time: 1.32 s
```

6. 모델 아티팩트가 로드되는 S3에서 출력 위치를 설정하여 아티팩트가 알고리즘 교육 작업의 출력이 될 수 있도록 합니다. 을 생성합니다 `sageMaker.estimator.Estimator` 교육 작업을 시작할 것을 반대합니다.

```
In [9]: s3_output_location = "s3://{}/{}/output".format(bucket, prefix)
In [10]: region_name = boto3.Session().region_name
In [11]: container =
sagemaker.amazon.amazon_estimator.get_image_uri(region_name,
"blazingtext","latest")
print("Using SageMaker BlazingText container: {} ({})".format(container,
region_name))
```

출력:

```
The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
Defaulting to the only supported framework/algorithm version: 1.
Ignoring f ramework/algorithm version: latest.
Using SageMaker BlazingText container: 811284229777.dkr.ecr.us-east-
1.amazo naws.com/blazingtext:1 (us-east-1)
```

7. SageMaker를 정의합니다 `Estrimator` c4.4x4 대형 인스턴스의 중계 모드를 사용하여 DBPedia 데이터세트에서 텍스트 분류를 훈련하기 위한 리소스 구성 및 하이퍼 매개변수.

```
In [12]: bt_model = sagemaker.estimator.Estimator(
container,
role,
instance_count=1,
instance_type="ml.c4.4xlarge",
volume_size=30,
max_run=360000,
input_mode="File",
output_path=s3_output_location,
hyperparameters={
        "mode": "supervised",
        "epochs": 1,
        "min_count": 2,
        "learning_rate": 0.05,
        "vector_dim": 10,
        "early_stopping": True,
        "patience": 4,
        "min_epochs": 5,
        "word_ngrams": 2,
 },
     )
```

8. 데이터 채널과 알고리즘 간의 핸드셰이크를 준비합니다. 이렇게 하려면 를 만듭니다 `sagemaker.session.s3_input` 데이터 채널의 객체를 알고리즘에 사용하기 위한 사전 내에 보관합니다.

```
ln [13]: train_data = sagemaker.inputs.TrainingInput(
    s3_train_data,
    distribution="FullyReplicated",
    content_type="text/plain",
    s3_data_type="S3Prefix",
)
validation_data = sagemaker.inputs.TrainingInput(
    s3_validation_data,
    distribution="FullyReplicated",
    content_type="text/plain",
    s3_data_type="S3Prefix",
)
data_channels = {"train": train_data, "validation": validation_data}
```

9. 작업이 완료되면 작업 완료 메시지가 나타납니다. 훈련된 모델은 로 설정된 S3 버킷에서 찾을 수 있습니다 `output_path` 계산기로.

```
ln [14]: bt_model.fit(inputs=data_channels, logs=True)
```

출력:

```
INFO:sagemaker:Creating training-job with name: blazingtext-2023-02-16-
20-3
7-30-748
2023-02-16 20:37:30 Starting - Starting the training job......
2023-02-16 20:38:09 Starting - Preparing the instances for
training......
2023-02-16 20:39:24 Downloading - Downloading input data
2023-02-16 20:39:24 Training - Training image download completed.
Training in progress... Arguments: train
[02/16/2023 20:39:41 WARNING 140279908747072] Loggers have already been
set up. [02/16/2023 20:39:41 WARNING 140279908747072] Loggers have
already been set up.
[02/16/2023 20:39:41 INFO 140279908747072] nvidia-smi took:
0.0251793861389
16016 secs to identify 0 gpus
[02/16/2023 20:39:41 INFO 140279908747072] Running single machine CPU
Blazi ngText training using supervised mode.
Number of CPU sockets found in instance is  1
[02/16/2023 20:39:41 INFO 140279908747072] Processing
/opt/ml/input/data/tr ain/dbpedia.train . File size: 35.0693244934082 MB
[02/16/2023 20:39:41 INFO 140279908747072] Processing
/opt/ml/input/data/va lidation/dbpedia.validation . File size:
21.887572288513184 MB
Read 6M words
Number of words:  149301
Loading validation data from
/opt/ml/input/data/validation/dbpedia.validati on
Loaded validation data.
-------------- End of epoch: 1 ##### Alpha: 0.0000  Progress: 100.00%
Million Words/sec: 10.39 ##### Training finished.
Average throughput in Million words/sec: 10.39
Total training time in seconds: 0.60
#train_accuracy: 0.7223
Number of train examples: 112000
#validation_accuracy: 0.7205
Number of validation examples: 70000
2023-02-16 20:39:55 Uploading - Uploading generated training model
2023-02-16 20:40:11 Completed - Training job completed
Training seconds: 68
Billable seconds: 68
```

10. 교육이 완료되면 아마존 SageMaker 실시간 호스팅 엔드포인트로 교육 받은 모델을 배포하여 예측을 수행합니다.

```
In [15]: from sagemaker.serializers import JSONSerializer
 text_classifier = bt_model.deploy(
     initial_instance_count=1, instance_type="ml.m4.xlarge",
serializer=JSONS
)
```

출력:

```
INFO:sagemaker:Creating model with name: blazingtext-2023-02-16-20-41-
33-10
0
INFO:sagemaker:Creating endpoint-config with name blazingtext-2023-02-
16-20
-41-33-100
INFO:sagemaker:Creating endpoint with name blazingtext-2023-02-16-20-41-
33-
100
-------!
```

```
In [16]: sentences = [
    "Convair was an american aircraft manufacturing company which later
expanded into rockets and spacecraft.",
       "Berwick secondary college is situated in the outer melbourne
metropolitan suburb of berwick .",
]
# using the same nltk tokenizer that we used during data preparation for
training
tokenized_sentences = [" ".join(nltk.word_tokenize(sent)) for sent in
sentences]
payload = {"instances": tokenized_sentences} response =
text_classifier.predict(payload)
predictions = json.loads(response)
print(json.dumps(predictions, indent=2))
```

```
[
  {
    "label": [
      "__label__Artist"
    ],
    "prob": [
      0.4090951681137085
    ]
  },
  {
    "label": [
      "__label__EducationalInstitution"
    ],
    "prob": [
      0.49466073513031006
    ]
  }
]
```

11. 기본적으로 모델은 가장 높은 확률로 하나의 예측을 반환합니다. 를 눌러 맨 위를 검색합니다 `k` 예측, 설정 `k` 를 구성 파일에 저장합니다.

```
In [17]: payload = {"instances": tokenized_sentences, "configuration":
{"k": 2}}
 response = text_classifier.predict(payload)

 predictions = json.loads(response)
 print(json.dumps(predictions, indent=2))
```

```
[
  {
    "label": [
      "__label__Artist",
      "__label__MeanOfTransportation"
    ],
    "prob": [
      0.4090951681137085,
      0.26930734515190125
    ]
  },
  {
    "label": [
      "__label__EducationalInstitution",
      "__label__Building"
    ],
    "prob": [
      0.49466073513031006,
      0.15817692875862122
    ]
  }
]
```

12. 전자 필기장을 닫기 전에 끝점을 삭제합니다.

```
In [18]: sess.delete_endpoint(text_classifier.endpoint)
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed
in s agemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
INFO:sagemaker:Deleting endpoint with name: blazingtext-2023-02-16-20-
41-33
-100
```

## 결론

이 검증 결과에 따라 데이터 과학자와 엔지니어는 NetApp Cloud Volumes ONTAP의 S3 버킷을 통해 AWS SageMaker Jupyter Notebooks에서 NFS 데이터에 액세스할 수 있습니다. 이 접근 방식을 사용하면 추가 소프트웨어 없이 NFS 및 S3에서 동일한 데이터에 쉽게 액세스하고 공유할 수 있습니다.

### 추가 정보를 찾을 수 있는 위치

이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹 사이트를 검토하십시오.

- SageMaker BlazingText를 사용한 텍스트 분류

  "https://sagemaker-examples.readthedocs.io/en/latest/introduction_to_amazon_algorithms/blazingtext_text_classification_dbpedia/blazingtext_text_classification_dbpedia.html"

- S3 오브젝트 스토리지를 위한 ONTAP 버전 지원

  "https://docs.netapp.com/us-en/ontap/s3-config/ontap-version-support-s3-concept.html"

# NetApp NFS 스토리지를 사용하는 Apache Kafka 워크로드

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

Shantanu Chakole, Karthikeyan Nagalingam 및 NetApp의 Joe Scott입니다

Kafka는 많은 양의 메시지 데이터를 수용할 수 있는 강력한 큐가 있는 분산 게시 구독 메시징 시스템입니다. Kafka를 사용하면 응용 프로그램에서 매우 빠른 방법으로 주제에 데이터를 쓰고 읽을 수 있습니다. 내결함성 및 확장성 덕분에 Kafka는 많은 데이터 스트림을 매우 빠르게 수집하고 이동하는 안정적인 방법으로 빅 데이터 공간에서 자주 사용됩니다. 사용 사례에는 스트림 처리, 웹 사이트 활동 추적, 메트릭 수집 및 모니터링, 로그 집계, 실시간 분석 등이 포함됩니다.

NFS에서 Kafka의 정상 작동이 잘 되긴 하지만 "이름 바꾸기 바보 같은 소리" NFS에서 실행 중인 Kafka 클러스터의 크기 조정 또는 재분할 중에 문제가 발생하면 응용 프로그램이 중단됩니다. 로드 밸런싱 또는 유지 관리를 위해 Kafka 클러스터의 크기를 조정하거나 재분할해야 하기 때문에 이는 중요한 문제입니다. 추가 세부 정보를 찾을 수 있습니다 "여기".

이 문서에서는 다음 주제에 대해 설명합니다.

- 이름 바꾸기 문제 및 솔루션 유효성 검사가 어리석은 문제입니다
- CPU 활용률을 줄여 I/O 대기 시간을 줄입니다
- Kafka 브로커 복구 시간 단축
- 클라우드 및 사내 성능

**Kafka 워크로드를 위해 NFS 스토리지를 사용하는 이유는 무엇입니까?**

운영 애플리케이션의 Kafka 워크로드는 애플리케이션 간에 방대한 양의 데이터를 스트리밍할 수 있습니다. 이 데이터는 Kafka 클러스터의 Kafka 브로커 노드에 저장되어 저장됩니다. Kafka는 또한 가용성 및 병렬 처리도 잘 알려져 있으며, 이를 위해 여러 주제를 파티션으로 분할한 다음 클러스터 전체에 파티션을 복제합니다. 이는 결국 Kafka 클러스터를 통해 흐르는 엄청난 양의 데이터가 일반적으로 크기에 배가된다는 것을 의미합니다. 브로커들의 수가 매우 빠르고 쉽게 변경됨에 따라 NFS에서 데이터의 균형을 재조정합니다. 대규모 환경에서는 브로커 변경 횟수가 매우 많은 경우 DAS 전반에서 데이터 균형을 재조정하는 데 많은 시간이 걸리며, 대부분의 Kafka 환경에서는 브로커 수가 자주 변경됩니다.

그 밖의 이점은 다음과 같습니다.

- 성숙도. * NFS는 완성도 높은 프로토콜입니다. 즉, NFS를 구현, 보안 및 사용하는 대부분의 측면을 충분히 이해할 수 있습니다.

- * Open. * NFS는 개방형 프로토콜이며, 지속적인 개발은 인터넷 사양에 무료 개방형 네트워크 프로토콜로 문서화되어 있습니다.
- 비용 효율성 * NFS는 기존 네트워크 인프라를 사용하기 때문에 설정하기 쉬운 네트워크 파일 공유를 위한 저렴한 솔루션입니다.
- * 중앙 집중식 관리. * NFS의 중앙 집중식 관리로 개별 사용자 시스템에 소프트웨어 및 디스크 공간을 추가할 필요가 없습니다.
- * Distributed. * NFS는 분산 파일 시스템으로 사용할 수 있으므로 이동식 미디어 저장 장치의 필요성을 줄여줍니다.

**Kafka** 워크로드를 위해 **NetApp**을 선택해야 하는 이유

NetApp NFS 구현은 프로토콜의 골드 표준으로 간주되며 수많은 엔터프라이즈 NAS 환경에서 사용됩니다. 또한 NetApp의 신용도와 더불어 다음과 같은 이점도 제공합니다.

- 안정성 및 효율성
- 확장성 및 성능
- 고가용성(NetApp ONTAP 클러스터의 HA 파트너)
- 데이터 보호
  - * 재해 복구(NetApp SnapMirror). * 사이트가 다운되거나 다른 사이트에서 바로 시작하여 중단된 부분부터 계속 진행하려고 합니다.
  - 스토리지 시스템의 관리 효율성(NetApp OnCommand를 사용한 관리 및 관리).
  - * 로드 밸런싱. * 이 클러스터를 사용하면 다른 노드에 호스팅된 데이터 LIF에서 서로 다른 볼륨에 액세스할 수 있습니다.
  - * 무중단 운영 * LIF 또는 볼륨 이동은 NFS 클라이언트에 영향을 미치지 않습니다.

## **NFS**에서 **Kafka** 워크로드로의 이름 바꾸기 문제를 해결하기 위한 **NetApp** 솔루션

Kafka는 기본 파일 시스템이 POSIX를 준수한다고 가정합니다(예: XFS 또는 ext4). Kafka 리소스 재조정은 애플리케이션이 파일을 계속 사용하는 동안 파일을 제거합니다. POSIX 호환 파일 시스템을 사용하면 연결을 끊을 수 있습니다. 그러나 파일에 대한 모든 참조가 제거된 후에만 파일을 제거합니다. 기본 파일 시스템이 네트워크에 연결되어 있는 경우 NFS 클라이언트는 링크 해제 통화를 차단하고 워크플로우를 관리합니다. 연결 해제 중인 파일에 대해 보류 중인 열기 때문에 NFS 클라이언트는 이름 바꾸기 요청을 NFS 서버로 보내고 연결되지 않은 파일의 마지막 닫기에서는 이름이 변경된 파일에 대해 제거 작업을 실행합니다. 이 동작은 일반적으로 NFS의 이름이 바보 같은 이름 바꾸기라고 하며 NFS 클라이언트가 조율합니다.

NFSv3 서버의 스토리지를 사용하는 Kafka 브로커는 이 동작으로 인해 문제가 발생합니다. 그러나 NFSv4.x 프로토콜에는 열려 있는 연결되지 않은 파일에 대해 서버가 책임을 지도록 하여 이 문제를 해결할 수 있는 기능이 있습니다. 이 선택적 기능을 지원하는 NFS 서버는 파일을 열 때 NFS 클라이언트에 소유권 기능을 전달합니다. 그런 다음 보류 중인 가 열려 있고 서버가 흐름을 관리할 수 있으면 NFS 클라이언트가 연결 해제 관리를 중단합니다. NFSv4 사양에서는 구축 지침을 제공하지만 지금까지 이 선택적 기능을 지원하는 알려진 NFS 서버 구현은 없었습니다.

NFS 서버와 NFS 클라이언트가 이름 바꾸기 문제를 해결하려면 다음 변경 사항이 필요합니다.

- * NFS 클라이언트 변경 사항(Linux). * 파일을 열 때 NFS 서버가 플래그를 응답하여 열린 파일의 연결 해제 처리 기능을 나타냅니다. NFS 클라이언트 측 변경 사항을 사용하면 NFS 서버가 해당 플래그가 있을 때 연결 해제를

처리할 수 있습니다. NetApp은 이러한 변경 사항으로 오픈 소스 Linux NFS 클라이언트를 업데이트했습니다. 업데이트된 NFS 클라이언트는 RHEL8.7 및 RHEL9.1에서 일반적으로 사용할 수 있습니다.

- * NFS 서버에 대한 변경. * NFS 서버가 열린 내용을 추적합니다. 이제 기존 열린 파일의 연결 해제는 POSIX 의미론과 일치하도록 서버에서 관리됩니다. 마지막 열기가 닫히면 NFS 서버가 파일의 실제 제거를 시작하여 이름이 바보스러운 이름 바꾸기 프로세스를 방지합니다. ONTAP NFS 서버는 최신 버전인 ONTAP 9.12.1에서 이 기능을 구현했습니다.

Kafka는 NFS 클라이언트 및 서버의 변경 사항을 통해 네트워크에 연결된 NFS 스토리지의 모든 이점을 안전하게 누릴 수 있습니다.

## 기능 검증 - 이름 바꾸기 수정 바보

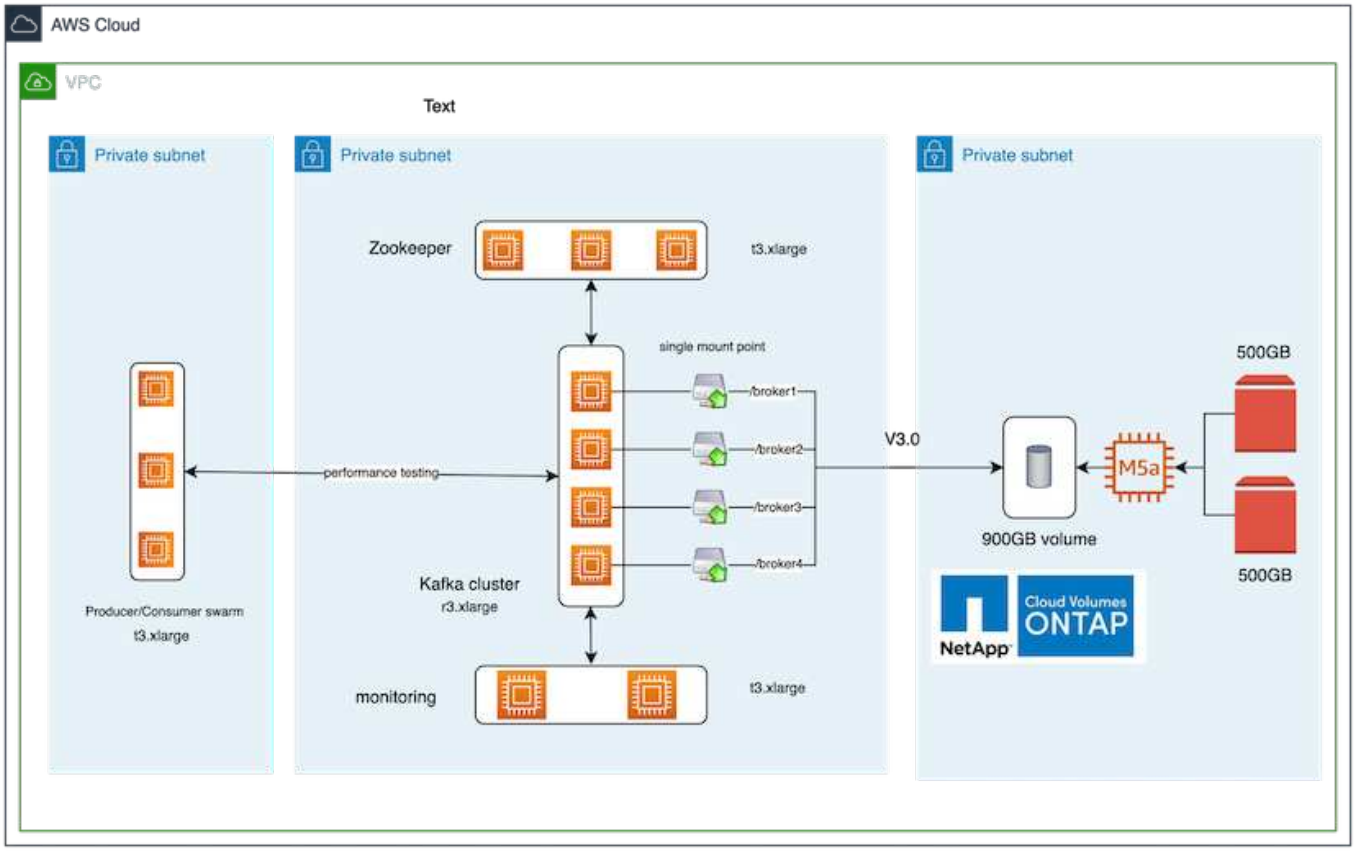
기능 검증을 위해 NFSv3 마운트를 사용하는 Kafka 클러스터가 파티션 재배포와 같은 Kafka 작업을 수행하지 않는 반면, 수정 사항이 있는 NFSv4에 마운트된 또 다른 클러스터는 중단 없이 동일한 작업을 수행할 수 있음을 보여주었습니다.

### 검증 설정

이 설정은 AWS에서 실행됩니다. 다음 표에는 검증에 사용된 다양한 플랫폼 구성 요소와 환경 구성이 나와 있습니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Confluent 플랫폼 버전 7.2.1 | • 3 x zookeers – T3.xLarge<br>• 브로커 서버 4대 – R3.xLarge<br>• Grafana 1개 – T3.xLarge<br>• 제어 센터 1개 – T3.xLarge<br>• 프로듀서/소비자 3대 |
| 모든 노드의 운영 체제 | RHEL8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

다음 그림에서는 이 솔루션의 아키텍처 구성을 보여 줍니다.




아키텍처 흐름

- * Compute. * 4노드 Kafka 클러스터와 3노드 zookeeper 앙상블이 전용 서버에서 실행되고 있었습니다.
- * 모니터링. * Prometheus-Grafana 조합에 두 개의 노드를 사용했습니다.
- * 워크로드. * 워크로드를 생성하는 데 별도의 3노드 클러스터를 사용하여 Kafka 클러스터에 생성하고 사용합니다.
- * 스토리지. * 이 인스턴스에는 500GB GP2 AWS-EBS 볼륨 2개가 연결된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스가 사용되었습니다. 그런 다음 LIF를 통해 이러한 볼륨을 단일 NFSv4.1 볼륨으로 Kafka 클러스터에 노출했습니다.

Kafka의 기본 속성은 모든 서버에 대해 선택되었습니다. 지퍼지기 백팔도 마찬가지였습니다.

테스트 방법

1. 업데이트 `-is-preserve-unlink-enabled true` 다음과 같이 Kafka 볼륨으로 이동합니다.

```
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 생성되었으며 다음과 같은 차이점이 있습니다.

- * 클러스터 1. * 운영 지원 ONTAP 버전 9.12.1을 실행하는 백엔드 NFS v4.1 서버는 NetApp CVO 인스턴스에서 호스팅되었습니다. 브로커에 RHEL 8.7/RHEL 9.1이 설치되었습니다.
- * 클러스터 2. * 백엔드 NFS 서버는 수동으로 생성된 일반 Linux NFSv3 서버였습니다.

3. 두 Kafka 클러스터 모두에서 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 클러스터에 대해 새로 생성된 이러한 항목에 데이터가 로드되었습니다. 이 작업은 기본 Kafka 패키지에 포함된 프로듀서-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 클러스터에 대해 broker-1에 대한 상태 점검을 수행했습니다.

   - 텔넷 `172.30.0.160 9092`
   - 텔넷 `172.30.0.198 9092`

     두 클러스터 모두에서 브로커의 상태 점검이 성공적으로 완료된 경우 다음 스크린샷에 나와 있습니다.

```
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 중단되는 장애 조건을 트리거하기 위해 두 클러스터에서 파티션 재할당 프로세스를 시작했습니다. 파티션 재할당이 을(를) 사용하여 수행되었습니다 `kafka-reassign-partitions.sh`. 자세한 프로세스는 다음과 같습니다.

   a. Kafka 클러스터의 주제에 대한 파티션을 재할당하기 위해 제안된 재할당 구성 JSON을 생성했습니다(두 클러스터에 대해 수행됨).

   ```
   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 JSON이 에 저장되었습니다 `/tmp/reassignment- file.json`.

   c. 실제 파티션 재할당 프로세스는 다음 명령을 통해 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 완료되고 몇 분 후에 브로커에 대한 또 다른 상태 점검 결과, NFSv3 스토리지 볼륨을 사용하는 클러스터가 이름이 바보 같은 문제로 실행되고 충돌이 발생했던 반면, NetApp ONTAP NFSv4.1 스토리지 볼륨을 사용하는 클러스터 1은 수정 작업을 중단 없이 계속했습니다.

```
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-1이 활성 상태입니다.
- Cluster2-broker-1이 작동하지 않습니다.

8. Kafka 로그 디렉토리를 확인했을 때 수정하여 NetApp ONTAP NFSv4.1 스토리지 볼륨을 사용하는 클러스터 1에서 파티션을 클린 할당했다는 것이 분명했지만, 일반 NFSv3 스토리지를 사용하는 클러스터 2에서는 이름 변경 문제가 우스꽝스러운 이유로 발생하지 않아 충돌이 발생하지 않았습니다. 다음 그림에서는 클러스터 2의 파티션 재조정을 보여 주며, NFSv3 스토리지에서 이름 바꾸기 문제가 우스꽝스러운 것으로 나타났습니다.

```
/demo/broker_demo_1/__a_demo_topic-1.b31a8dd60fd443b283ffda2ecca9c2b9-delete:
total 40
drwxr-xr-x.   2 nobody nobody  4096 Sep 19 10:37 .
drwxr-xr-x. 246 nobody nobody 32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody     5 Sep 19 10:22 .nfs0000000025f9008400000045
-rw-r--r--.   1 nobody nobody     0 Sep 19 10:25 .nfs0000000025f91d6800000048

/demo/broker_demo_1/__a_demo_topic-2:
total 832592
drwxr-xr-x.   2 nobody nobody      4096 Sep 19 10:26 .
drwxr-xr-x. 246 nobody nobody     32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody         5 Sep 19 10:22 .nfs0000000025f91d5500000046
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:25 .nfs0000000025f91fce00000047
-rw-r--r--.   1 nobody nobody  10485760 Sep 19 10:24 00000000000000000000.index
-rw-r--r--.   1 nobody nobody 848113134 Sep 19 10:24 00000000000000000000.log
-rw-r--r--.   1 nobody nobody  10485756 Sep 19 10:24 00000000000000000000.timeindex
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:16 leader-epoch-checkpoint
-rw-r--r--.   1 nobody nobody        43 Sep 19 10:16 partition.metadata
```

다음 그림은 NetApp NFSv4.1 스토리지를 사용하여 클러스터 1의 파티션을 완전히 재조정하는 방법을 보여줍니다.

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

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

## **Kafka** 워크로드를 위해 **NetApp NFS**를 선택해야 하는 이유

이제 Kafka를 사용하는 NFS 스토리지에 이름 바꾸기 관련 우스꽝스러운 문제를 해결하는 솔루션이 존재하므로 Kafka 워크로드를 위해 NetApp ONTAP 스토리지를 활용하는 강력한 구축을 구현할 수 있습니다. 이는 운영 오버헤드를 크게 줄일 뿐만 아니라 Kafka 클러스터에 다음과 같은 이점을 제공합니다.

- * Kafka 브로커의 CPU 사용률 감소 * disaggregated NetApp ONTAP 스토리지를 사용하면 디스크 I/O 작업이 브로커에서 분리되므로 CPU 설치 공간이 감소합니다.
- * 더 빠른 브로커 복구 시간. * 더 빠른 브로커 복구 시간 * 은 Kafka 브로커 노드 간에 분할된 NetApp ONTAP 스토리지가 공유되므로 데이터 재구축이 없는 기존 Kafka 구축에 비해 훨씬 빠른 시간 내에 모든 지점에서 새로운 컴퓨팅 인스턴스가 불량 브로커를 대체할 수 있습니다.
- * 스토리지 효율성. * 이제 애플리케이션의 스토리지 계층이 NetApp ONTAP를 통해 프로비저닝됨에 따라 고객은 인라인 데이터 압축, 중복제거, 컴팩션과 같은 ONTAP의 스토리지 효율성이 제공하는 모든 이점을 활용할 수 있습니다.

이 이점에 대해서는 이 섹션에서 자세히 설명하는 테스트 사례에서 테스트 및 검증을 거쳤습니다.

### **Kafka** 브로커의 **CPU** 사용률 감소

두 개의 정자로 된 Kafka 클러스터에서 유사한 워크로드를 실행했을 때 전체 CPU 사용률이 DAS보다 낮다는 사실을 발견했습니다. 이러한 워크로드는 기술 사양에서 동일하지만 스토리지 기술에서는 달랐습니다. Kafka 클러스터가 ONTAP 스토리지를 사용하는 경우 전체 CPU 사용률이 낮아질 뿐만 아니라 CPU 활용률이 DAS 기반 Kafka 클러스터보다 더 부드럽게 증가함을 보여 주었습니다.

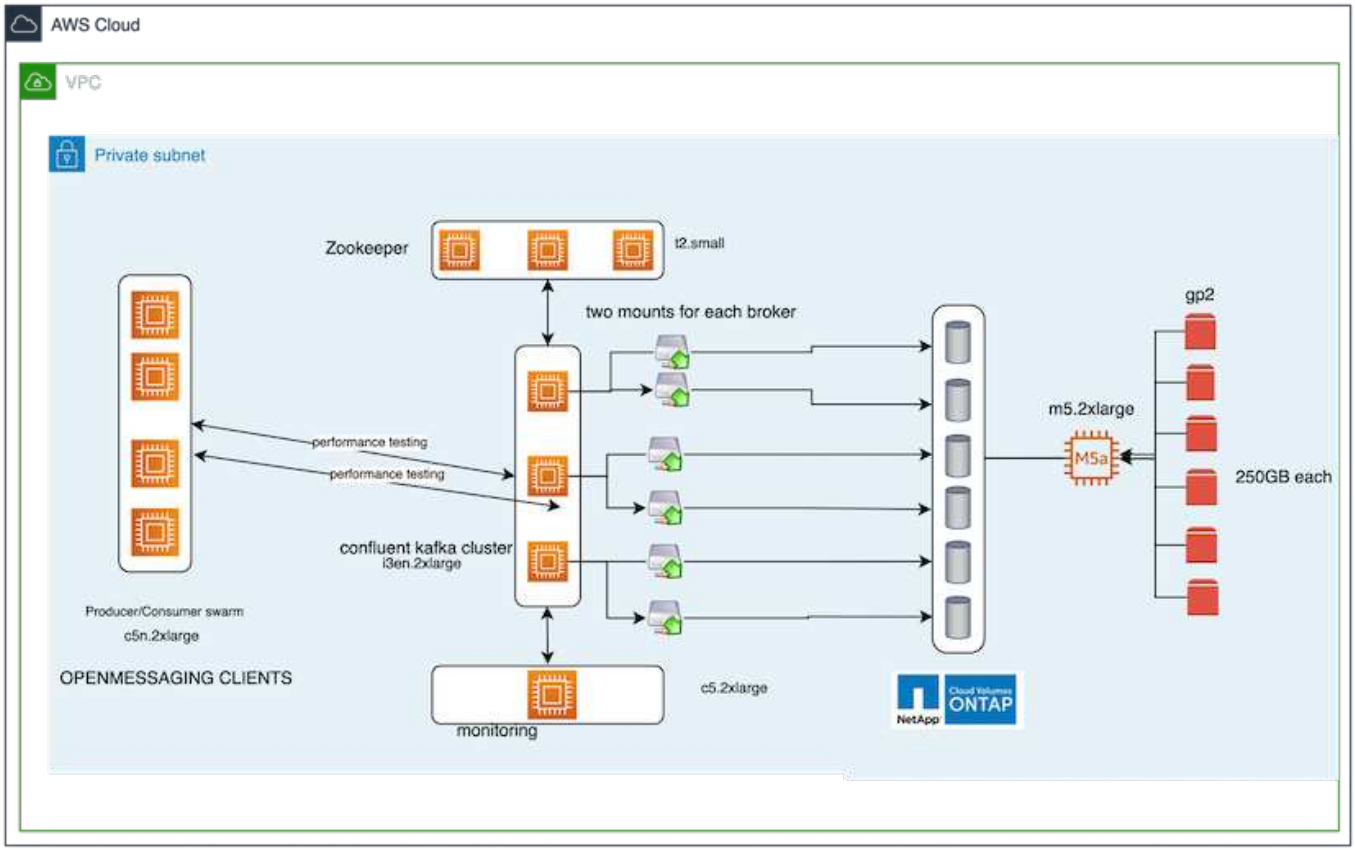
#### 아키텍처 설정

다음 표에는 CPU 사용률 감소를 보여 주는 데 사용되는 환경 구성이 나와 있습니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Kafka 3.2.3 벤치마킹 도구: OpenMessaging | • 3 x zookeers – T2.small<br>• 브로커 서버 3대 – i3en.2xLarge<br>• Grafana 1개 – c5n.2xLarge<br>• 프로듀서/소비자 4대 — c5n.2xLarge |
| 모든 노드의 운영 체제 | RHEL 8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

#### 벤치마킹 툴

이 테스트 사례에 사용된 벤치마킹 툴은 입니다 "OpenMessaging" 프레임워크: OpenMessaging은 공급업체에 구애받지 않고 언어와 독립적입니다. 금융, 전자 상거래, IoT 및 빅 데이터에 대한 업계 지침을 제공하며 이기종 시스템 및 플랫폼에서 메시징 및 스트리밍 애플리케이션을 개발하는 데 도움이 됩니다. 다음 그림에서는 OpenMessaging 클라이언트와 Kafka 클러스터의 상호 작용을 보여 줍니다.




- * Compute. * 3노드 Kafka 클러스터와 전용 서버에서 실행되는 3노드 zookeeper 앙상블이 사용되었습니다. 각 브로커는 전용 LIF를 통해 NetApp CVO 인스턴스의 단일 볼륨에 대한 NFSv4.1 마운트 지점을 2개 가지고 있었습니다.
- * 모니터링. * Prometheus-Grafana 조합에 두 개의 노드를 사용했습니다. 워크로드를 생성하기 위해 Kafka 클러스터에 대해 생성하고 사용할 수 있는 별도의 3노드 클러스터가 있습니다.
- * 스토리지. * 이 인스턴스에 마운트된 250GB GP2 AWS-EBS 볼륨 6개가 포함된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스를 사용했습니다. 그런 다음 전용 LIF를 통해 이러한 볼륨을 6개의 NFSv4.1 볼륨으로 Kafka 클러스터에 노출했습니다.
- * 구성. * 이 테스트 사례에서 구성 가능한 두 가지 요소는 Kafka 브로커와 OpenMessaging 워크로드였습니다.
  - * 브로커 구성 * Kafka 브로커에 대해 다음 사양이 선택되었습니다. 아래에 강조 표시된 것처럼 모든 측정에 3의 복제 계수를 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
```

- * OMB(OpenMessaging 벤치마크) 워크로드 구성 * 다음 사양이 제공됩니다. 아래에 강조 표시된 목표 생산자 비율을 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 방법

1. 서로 유사한 클러스터 두 개가 생성되었으며, 각 클러스터마다 자체 벤치마킹 클러스터 군들이 있습니다.
    - * 클러스터 1. * NFS 기반 Kafka 클러스터
    - * 클러스터 2. * DAS 기반 Kafka 클러스터
2. OpenMessaging 명령을 사용하여 각 클러스터에서 유사한 워크로드가 트리거되었습니다.

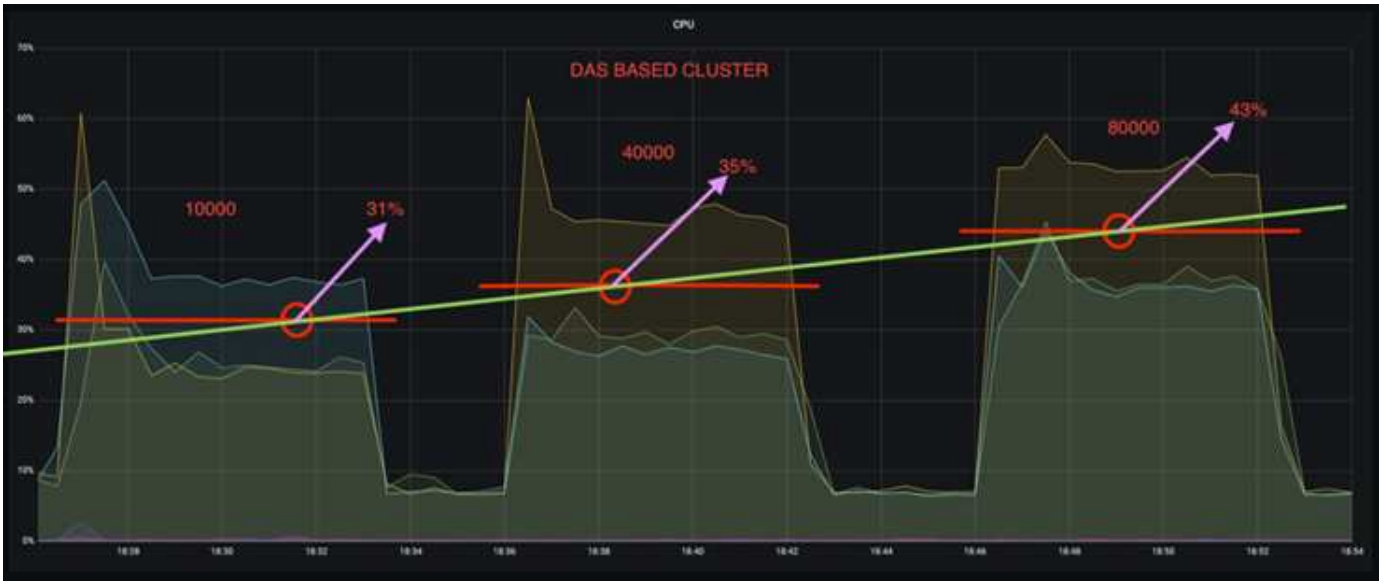
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

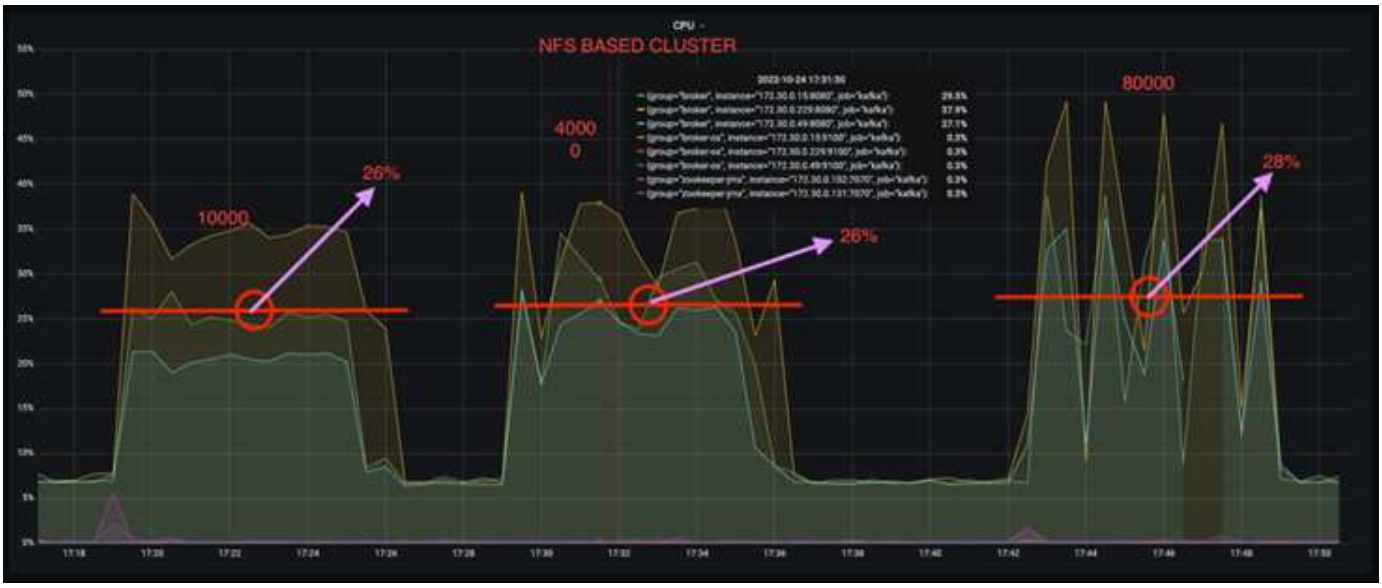
3. 생산율 구성이 4번 반복되어 Grafana로 기록되었으며, CPU 활용률이 기록되었습니다. 생산률이 다음 수준으로 설정되었습니다.
    - 10,000
    - 40,000개
    - 80,000
    - 100,000

관찰

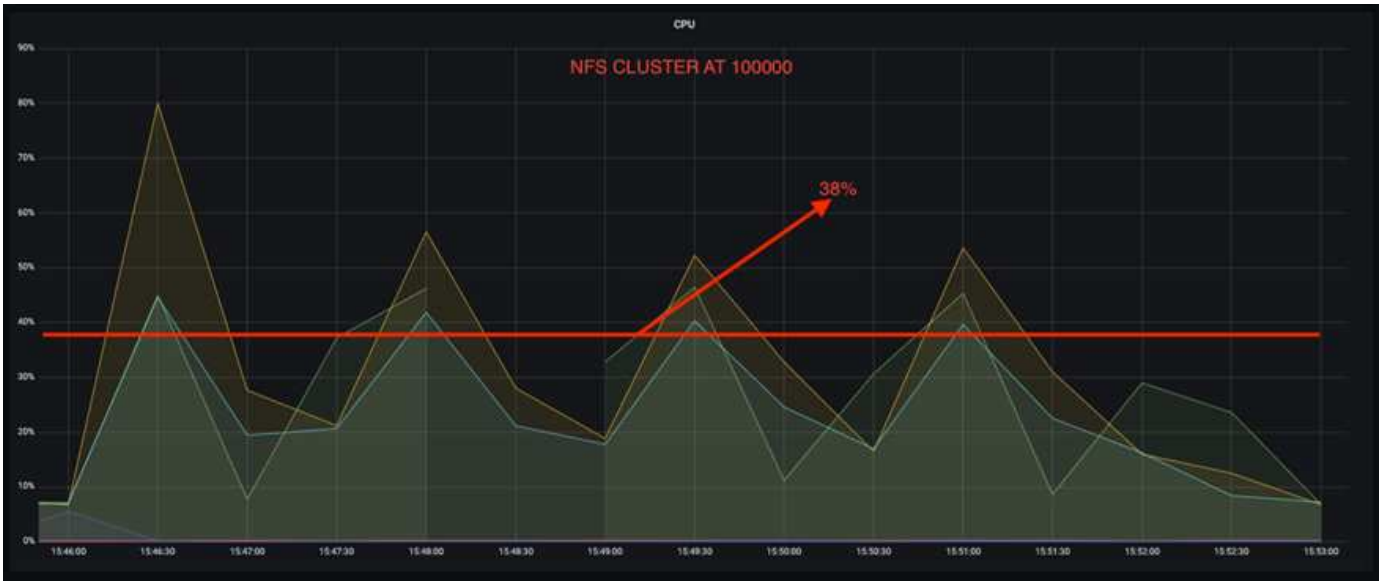
NetApp NFS 스토리지와 Kafka를 함께 사용하면 다음과 같은 두 가지 주요 이점을 얻을 수 있습니다.

- * CPU 사용률을 거의 1/3까지 줄일 수 있습니다. * 유사한 워크로드에서의 전체 CPU 사용은 NFS의 경우 DAS SSD에 비해 더 낮았습니다. 낮은 생산률의 경우 5%에서 더 높은 생산률의 경우 32%까지 절약되었습니다.
- * 더 높은 생산률에서 CPU 사용률 편차가 3배 감소했습니다. * 예상대로 생산률이 증가할수록 CPU 활용률이 증가하기 때문에 상승률이 나타났습니다. 그러나 DAS를 사용하는 Kafka 브로커의 CPU 활용률은 낮은 생산률의 31%에서 높은 생산률의 70%로 39% 증가했습니다. 하지만 NFS 스토리지 백엔드를 사용할 경우 CPU 활용률이 26%에서 38%로 12% 증가했습니다.






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

브로커 복구 시간 단축

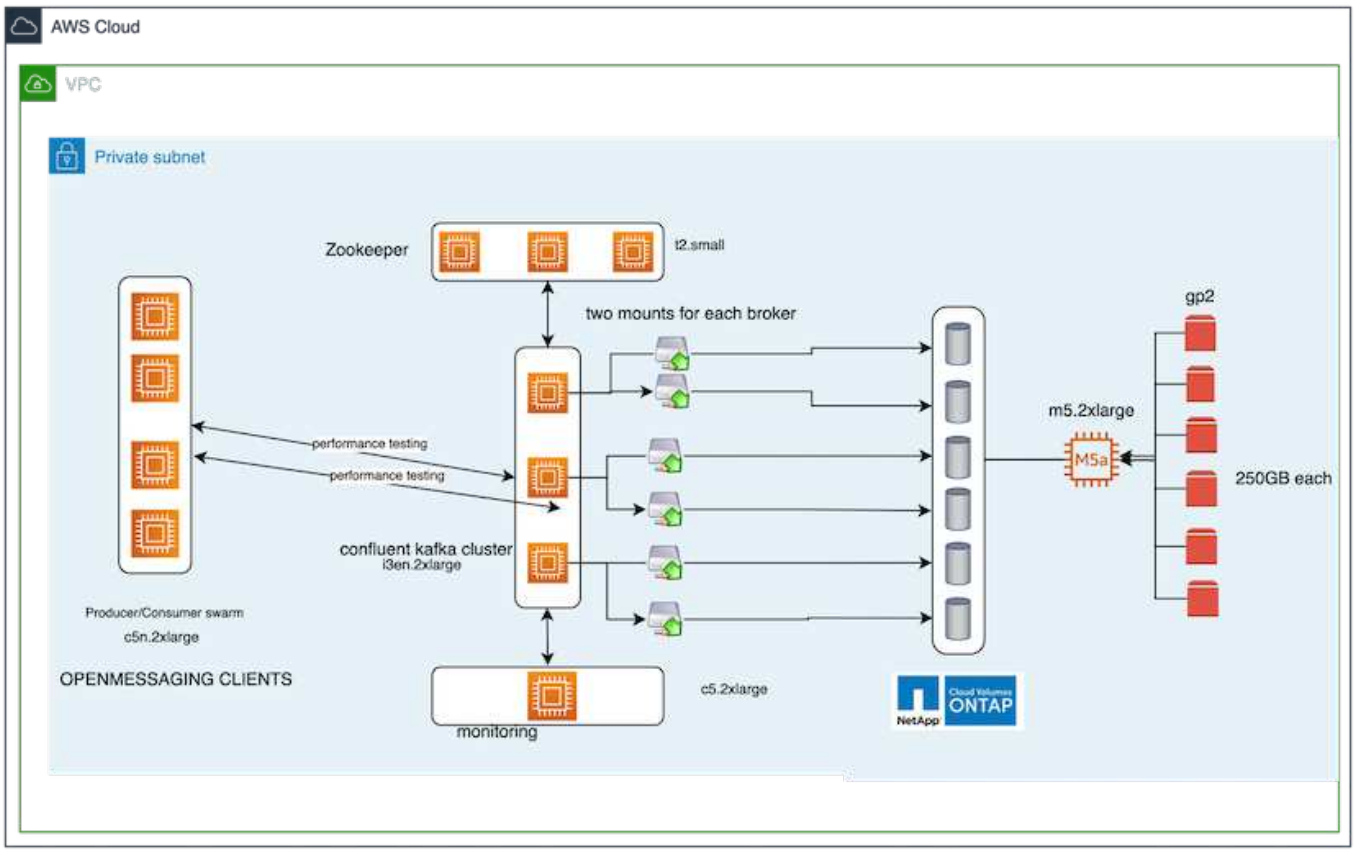
Kafka 브로커가 공유 NetApp NFS 스토리지를 사용할 때 더 빨리 복구한다는 사실을 알게 되었습니다. Kafka 클러스터에서 브로커가 충돌하는 경우 이 브로커는 동일한 브로커 ID를 가진 정상 브로커로 교체될 수 있습니다. 이 테스트 사례를 수행한 결과, DAS 기반 Kafka 클러스터의 경우 클러스터가 새로 추가된 정상적인 브로커로 데이터를 재구축하므로 시간이 오래 걸립니다. NetApp NFS 기반 Kafka 클러스터의 경우 대체 브로커가 이전 로그 디렉토리에서 데이터를 계속 읽고 복구 속도를 훨씬 높일 수 있습니다.

아키텍처 설정

다음 표에는 NAS를 사용하는 Kafka 클러스터의 환경 구성이 나와 있습니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Kafka 3.2.3 | • 3 x zookeers – T2.small<br>• 브로커 서버 3대 – i3en.2xLarge<br>• Grafana 1개 – c5n.2xLarge<br>• 생산자/소비자 4대 — c5n.2xLarge<br>• 백업 Kafka 노드 1개 – i3en.2xLarge |
| 모든 노드의 운영 체제 | RHEL8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

다음 그림에서는 NAS 기반 Kafka 클러스터의 아키텍처를 보여 줍니다.



- * Compute. * 전용 서버에서 3노드 zookeeper 앙상블이 실행되는 3노드 Kafka 클러스터. 각 브로커는 전용 LIF를 통해 NetApp CVO 인스턴스의 단일 볼륨에 대한 NFS 마운트 지점을 2개 가집니다.
- * 모니터링. * Prometheus-Grafana 조합에 대한 두 개의 노드. 워크로드를 생성하는 경우 이 Kafka 클러스터를 생성하고 사용할 수 있는 별도의 3노드 클러스터를 사용합니다.
- * 스토리지. * 250GB GP2 AWS-EBS 볼륨 6개가 인스턴스에 마운트된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스. 그런 다음 전용 LIF를 통해 Kafka 클러스터에 6개의 NFS 볼륨으로 노출됩니다.
- * 브로커 구성. * 이 테스트 사례에서 구성 가능한 한 가지 요소는 Kafka 브로커입니다. Kafka 브로커에 대해 다음과 같은 사양이 선택되었습니다. 를 클릭합니다 `replica.lag.time.mx.ms` ISR 목록에서 특정 노드를 얼마나 빨리 제외할지 결정하므로 값이 높게 설정됩니다. 불량 노드와 정상 노드 간에 전환할 때 ISR 목록에서 브로커 ID를 제외하지 않도록 합니다.

```
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 방법

1. 두 개의 유사한 클러스터가 생성되었습니다.
   - EC2 기반 confluent 클러스터
   - NetApp NFS 기반 confluent 클러스터
2. 하나의 대기 Kafka 노드가 원래 Kafka 클러스터의 노드와 동일한 구성으로 생성되었습니다.
3. 각 클러스터마다 샘플 주제가 생성되었으며 각 브로커에 약 110GB의 데이터가 채워졌습니다.
   - * EC2 기반 클러스터 * Kafka 브로커 데이터 디렉토리가 에 매핑되어 있습니다 `/mnt/data-2` (다음 그림에서 cluster1 의 Broker-1 [왼쪽 터미널]).
   - * NetApp NFS 기반 클러스터 * Kafka 브로커 데이터 디렉토리가 NFS 지점에 마운트됩니다 `/mnt/data` (다음 그림에서 클러스터2의 브로커-1[오른쪽 터미널])


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

5. 브로커가 종료된 후 브로커 IP 주소가 스탠바이 브로커의 보조 IP로 할당되었습니다. Kafka 클러스터의 브로커가 다음과 같이 식별되기 때문에 이 작업이 필요했습니다.
    - * IP 주소. * 장애가 발생한 브로커 IP를 대기 브로커에 재할당하여 지정합니다.
    - * 브로커 ID. * 이것은 대기 브로커에서 구성되었습니다 `server.properties`.
6. IP 할당 시 Kafka 서비스는 대기 브로커에서 시작되었습니다.
7. 잠시 후 서버 로그를 가져와 클러스터의 교체 노드에 데이터를 구축하는 데 걸린 시간을 확인합니다.

관찰

Kafka 브로커 복구는 거의 9배 빨라졌습니다. NetApp NFS 공유 스토리지를 사용할 경우 실패한 브로커 노드를 복구하는 데 걸리는 시간이 Kafka 클러스터에서 DAS SSD를 사용하는 경우에 비해 훨씬 빠른 것으로 확인되었습니다. 1TB의 항목 데이터에서 DAS 기반 클러스터의 복구 시간은 NetApp-NFS 기반 Kafka 클러스터의 복구 시간은 5분 미만이었습니다.

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

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

2. 데이터 리빌딩 프로세스는 09:05:24,860으로 끝났습니다. 110GB의 데이터를 처리하는 데 약 10분이 소요됩니다.

```
[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** 기반 클러스터

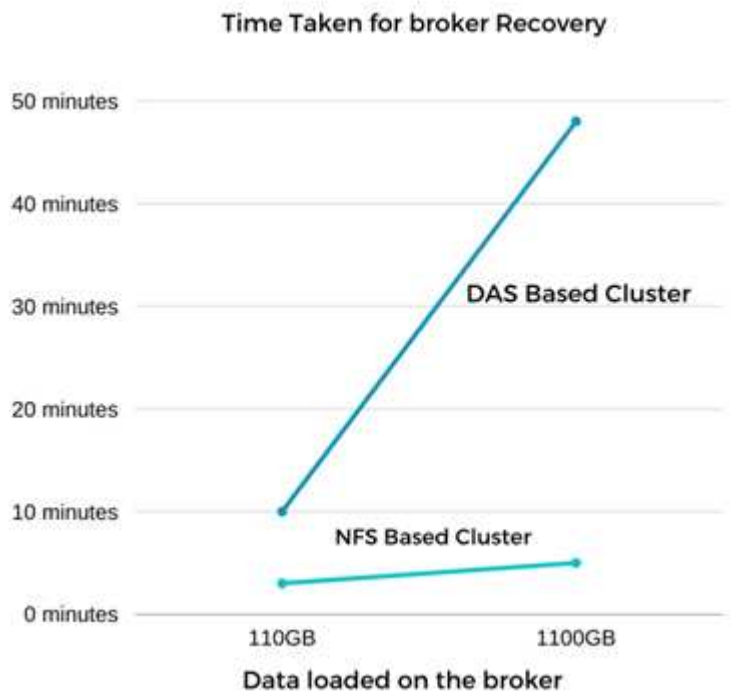
1. 백업 노드는 09:39:17,213에 시작되었습니다. 시작 로그 항목이 아래에 강조 표시됩니다.

```
[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 리빌드 프로세스는 09:42:29,115로 종료되었습니다. 110GB의 데이터를 처리하는 데 약 3분이 소요됩니다.

```
[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)
```

이 테스트는 약 1TB 데이터를 포함하는 브로커에 대해 반복되었으며 DAS의 경우 약 48분, NFS의 경우 약 3분이 소요됩니다. 결과는 다음 그래프에 나와 있습니다.




스토리지 효율성

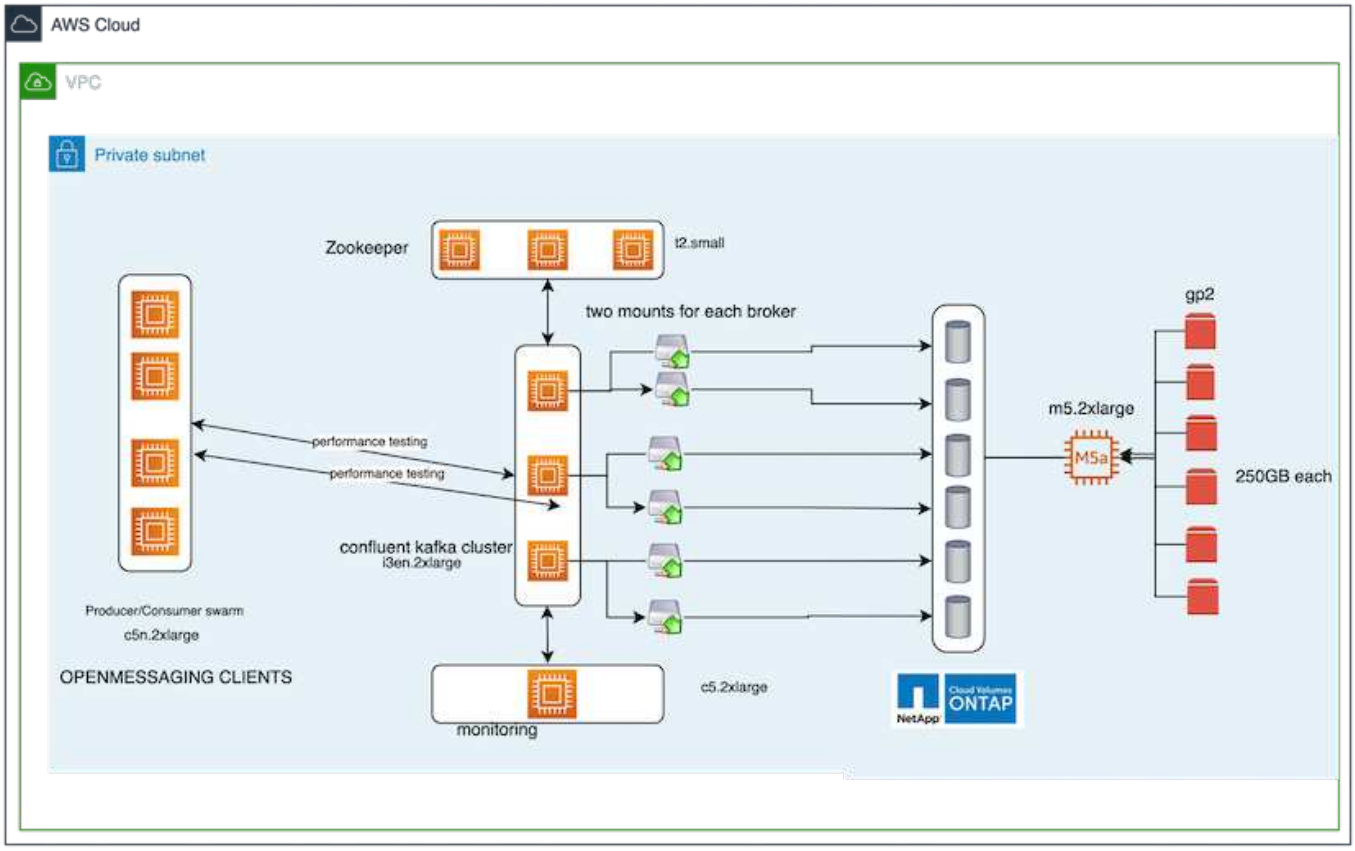
Kafka 클러스터의 스토리지 계층은 NetApp ONTAP를 통해 프로비저닝되었기 때문에 ONTAP의 모든 스토리지 효율성 기능을 얻었습니다. 이 테스트는 Cloud Volumes ONTAP에서 프로비저닝된 NFS 스토리지가 있는 Kafka 클러스터에 상당한 양의 데이터를 생성하여 테스트했습니다. ONTAP 기능 덕분에 공간을 상당히 줄일 수 있다는 것을 알 수 있었습니다.

아키텍처 설정

다음 표에는 NAS를 사용하는 Kafka 클러스터의 환경 구성이 나와 있습니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Kafka 3.2.3 | • 3 x zookeers – T2.small<br>• 브로커 서버 3대 – i3en.2xLarge<br>• Grafana 1개 – c5n.2xLarge<br>• 생산자/소비자 4대 — c5n.2xLarge * |
| 모든 노드의 운영 체제 | RHEL8.7 이상 |
| NetApp Cloud Volumes ONTAP 인스턴스 | 단일 노드 인스턴스 – M5.2xLarge |

다음 그림에서는 NAS 기반 Kafka 클러스터의 아키텍처를 보여 줍니다.




- * Compute. * 3노드 Kafka 클러스터와 전용 서버에서 실행되는 3노드 zookeeper 앙상블이 사용되었습니다. 각 브로커는 전용 LIF를 통해 NetApp CVO 인스턴스의 단일 볼륨에 대한 NFS 마운트 지점을 2개 가지고 있었습니다.
- * 모니터링. * Prometheus-Grafana 조합에 두 개의 노드를 사용했습니다. 워크로드를 생성하는데 이 Kafka 클러스터를 생성하고 사용할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- * 스토리지. * 이 인스턴스에 마운트된 250GB GP2 AWS-EBS 볼륨 6개가 포함된 단일 노드 NetApp Cloud Volumes ONTAP 인스턴스를 사용했습니다. 그런 다음 전용 LIF를 통해 이 볼륨을 6개의 NFS 볼륨으로 Kafka 클러스터에 노출했습니다.
- * 구성. * 이 테스트 사례에서 구성 가능한 요소는 Kafka 브로커였습니다.

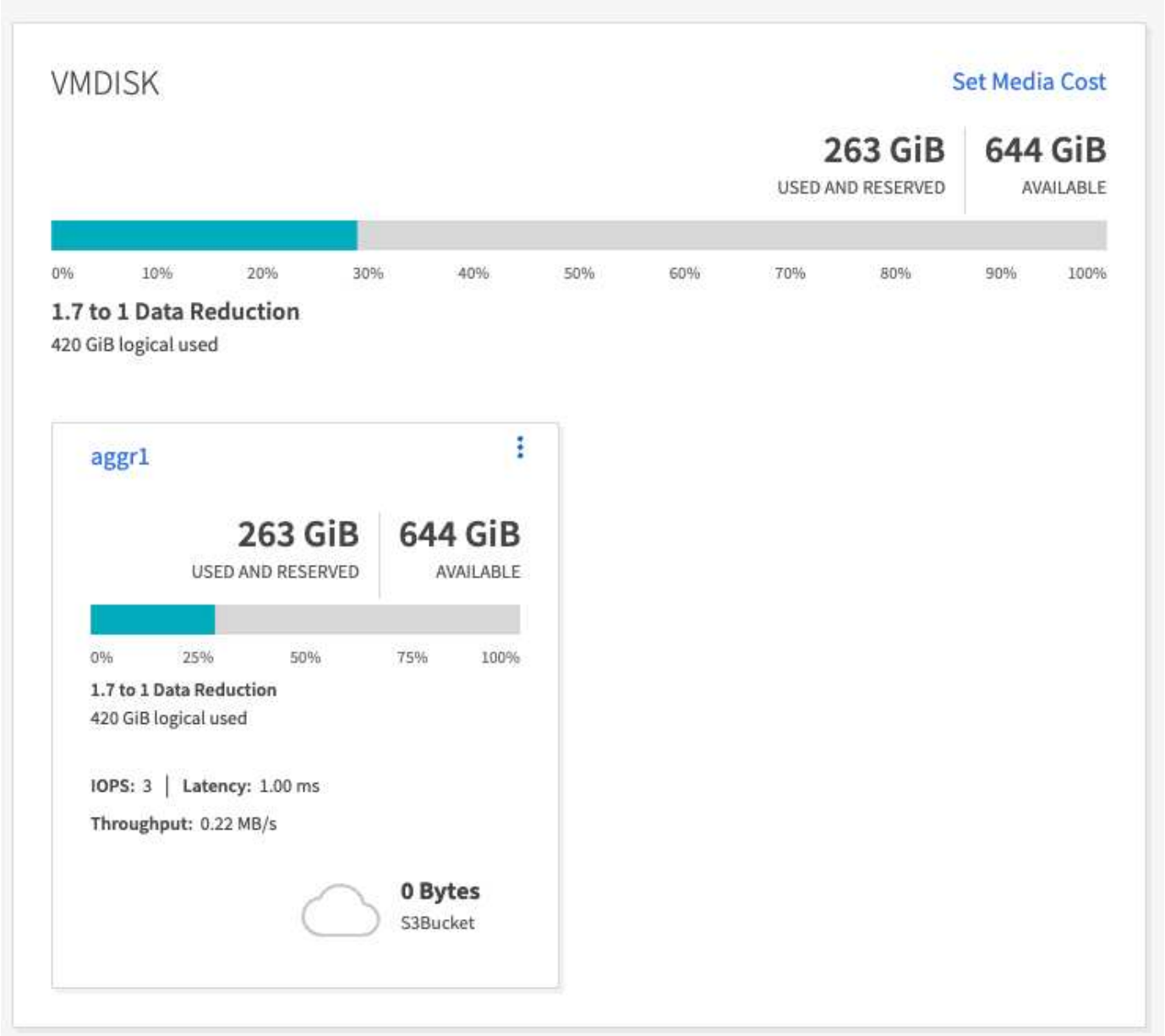
생산자 측의 압축 기능이 꺼지므로 생산자가 높은 처리량을 생성할 수 있습니다. 대신 컴퓨팅 계층에서 스토리지 효율성을 처리했습니다.

테스트 방법

1. Kafka 클러스터는 위에서 언급한 사양을 바탕으로 프로비저닝되었습니다.
2. 클러스터에서는 OpenMessaging 벤치마킹 도구를 사용하여 약 350GB의 데이터가 생성되었습니다.
3. 워크로드가 완료된 후 ONTAP System Manager 및 CLI를 사용하여 스토리지 효율성 통계가 수집됩니다.

관찰

OMB 툴을 사용하여 생성한 데이터의 경우, 스토리지 효율성 비율이 1.70:1인 33%까지 절약되었습니다. 다음 그림에서 볼 수 있듯이, 생성된 데이터에 사용된 논리적 공간은 420.3GB였으며 데이터를 저장하는 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
```

## AWS의 성능 개요 및 검증

NetApp NFS에 스토리지 계층이 마운트된 Kafka 클러스터가 AWS 클라우드의 성능을 벤치마킹했습니다. 벤치마킹 예는 다음 섹션에 설명되어 있습니다.

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

NetApp Cloud Volumes ONTAP(HA 쌍)를 사용한 Kafka 클러스터는 AWS 클라우드의 성능을 벤치마킹했습니다. 이 벤치마킹은 다음 섹션에서 설명합니다.

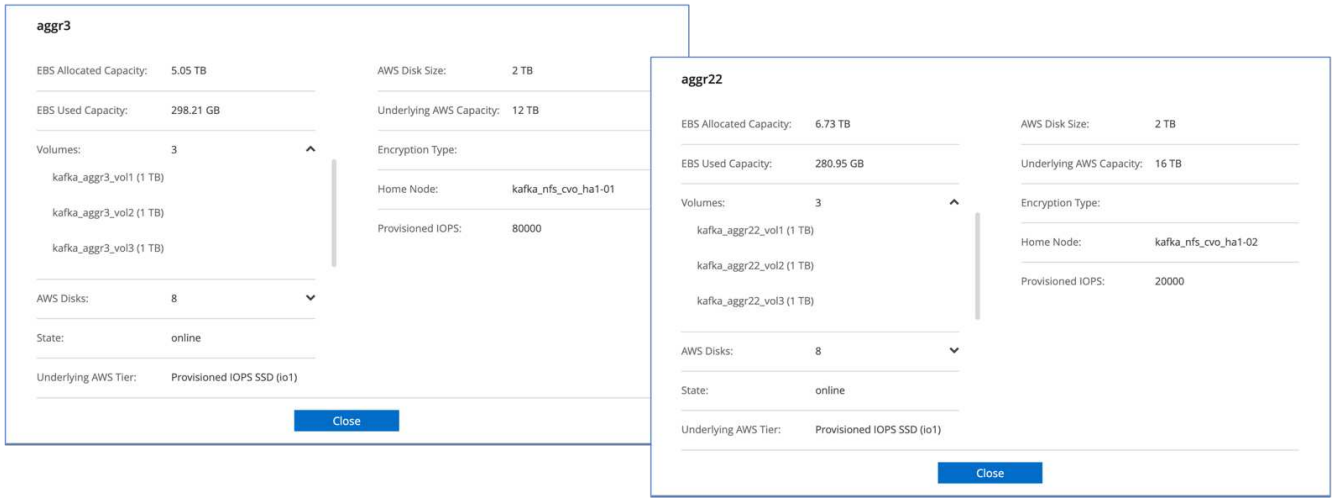
#### 아키텍처 설정

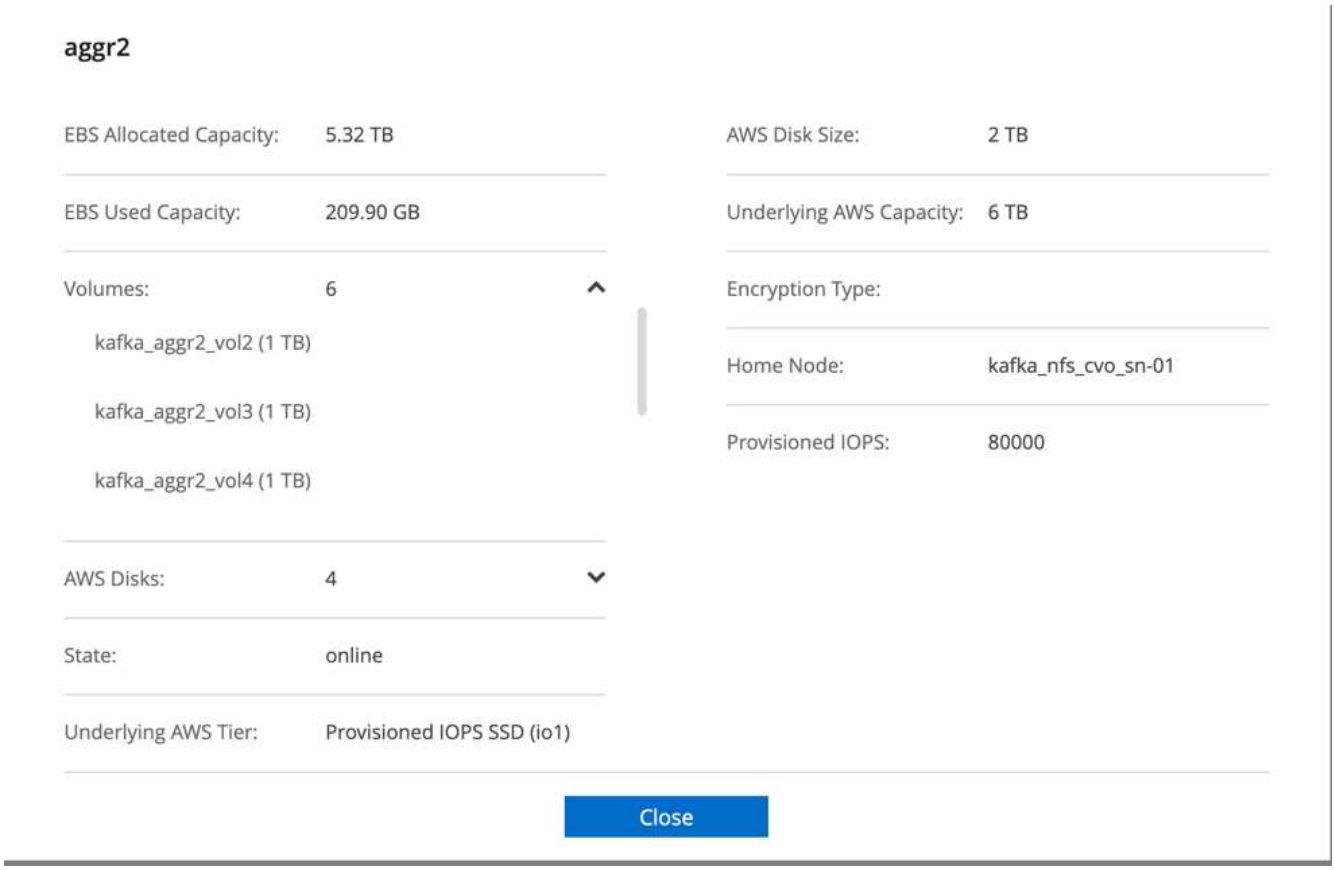
다음 표에는 NAS를 사용하는 Kafka 클러스터의 환경 구성이 나와 있습니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Kafka 3.2.3 | • 3 x zookeers – T2.small<br>• 브로커 서버 3대 – i3en.2xLarge<br>• Grafana 1개 – c5n.2xLarge<br>• 생산자/소비자 4대 — c5n.2xLarge * |
| 모든 노드의 운영 체제 | RHEL8.6 |
| NetApp Cloud Volumes ONTAP 인스턴스 | HA 쌍 인스턴스 – m5dn.12xLargge x 2노드 단일 노드 인스턴스 - m5dn.12xLargge x 1개 노드 |

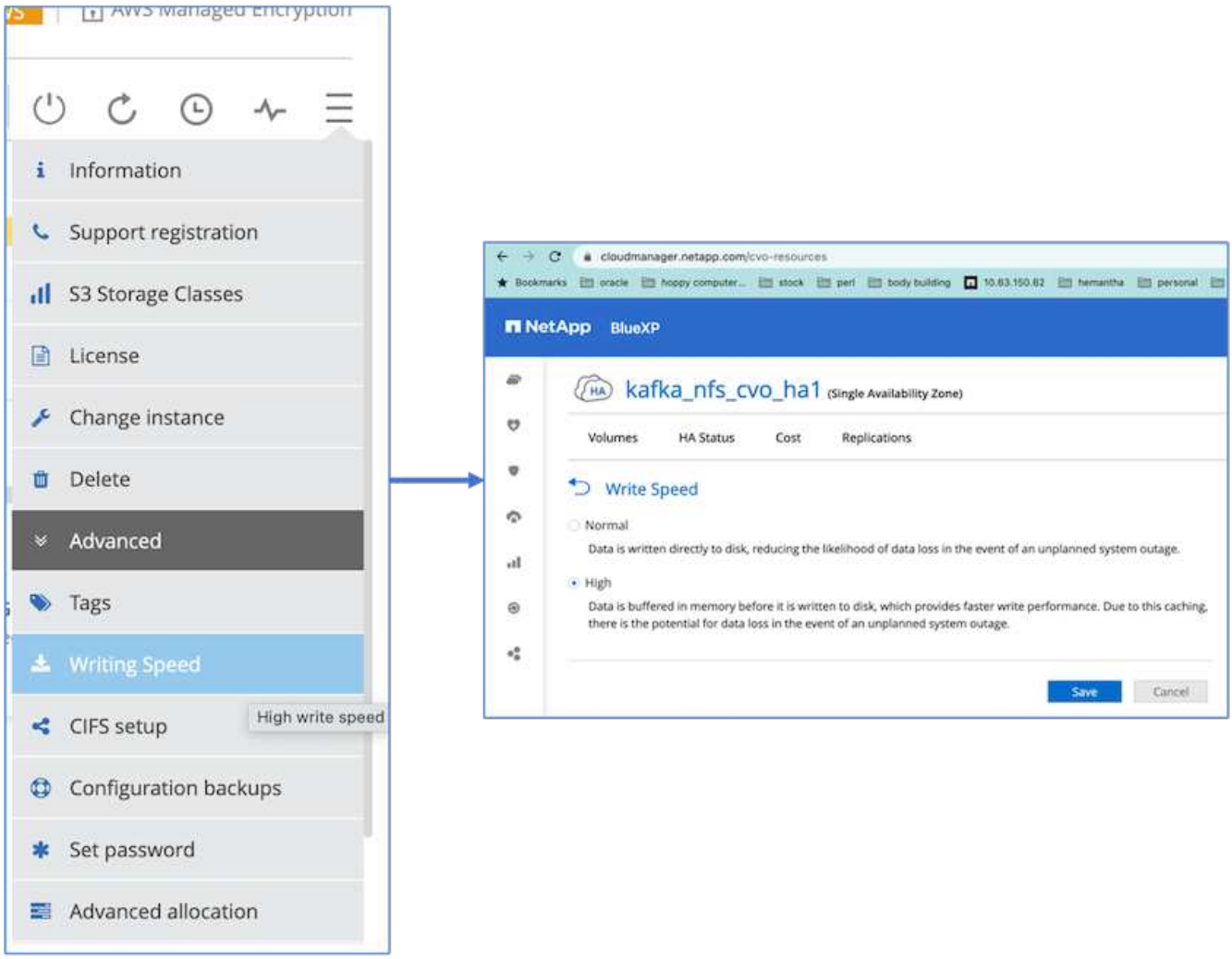
#### NetApp 클러스터 볼륨 ONTAP 설정

1. Cloud Volumes ONTAP HA 쌍의 경우, 각 스토리지 컨트롤러에서 각 애그리게이트의 볼륨 3개가 포함된 애그리게이트 2개를 생성했습니다. 단일 Cloud Volumes ONTAP 노드의 경우 Aggregate에 6개의 볼륨을 생성합니다.



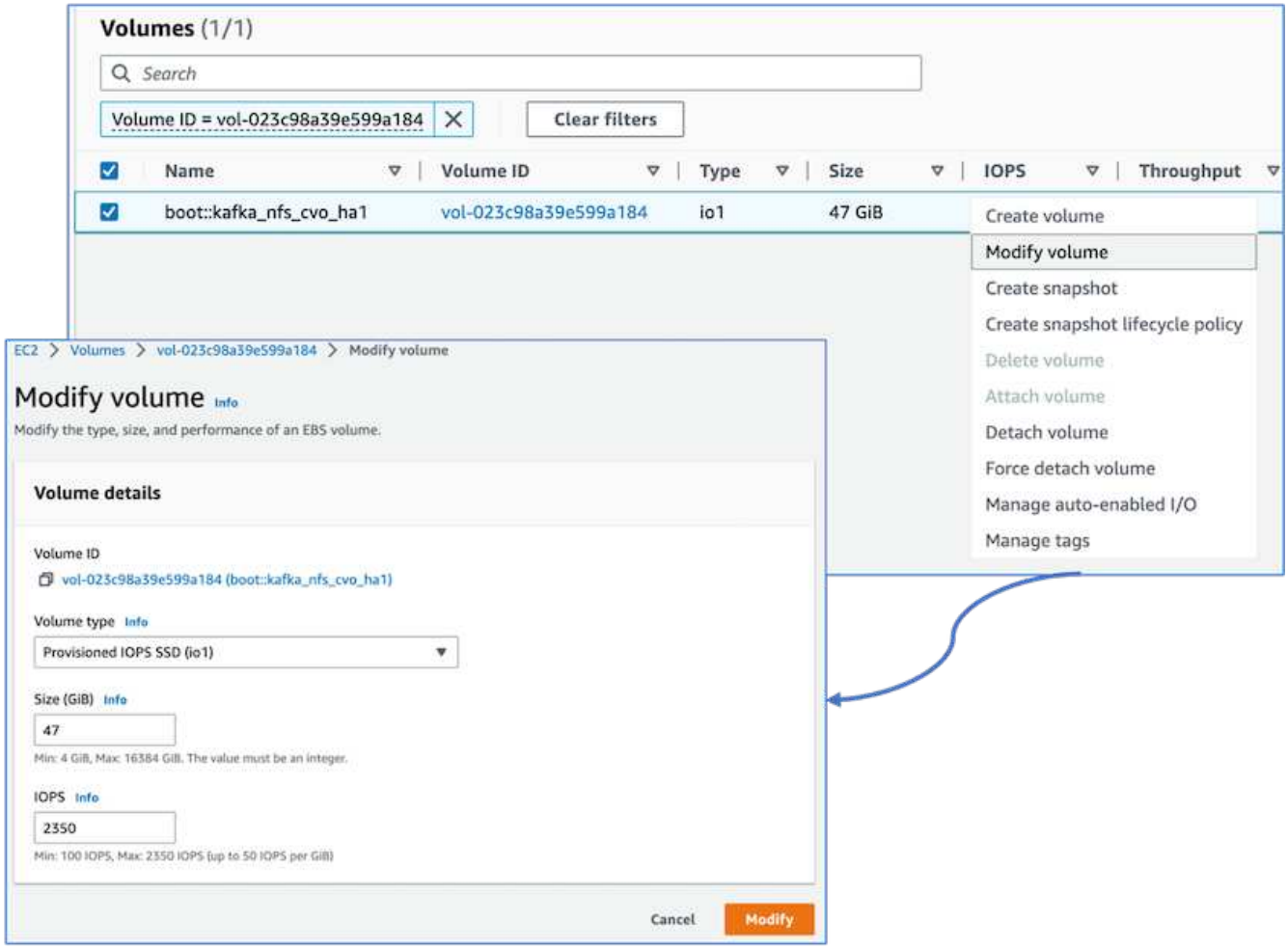




2. 네트워크 성능을 향상시키기 위해 HA 쌍과 단일 노드 모두에 고속 네트워킹을 활성화했습니다.



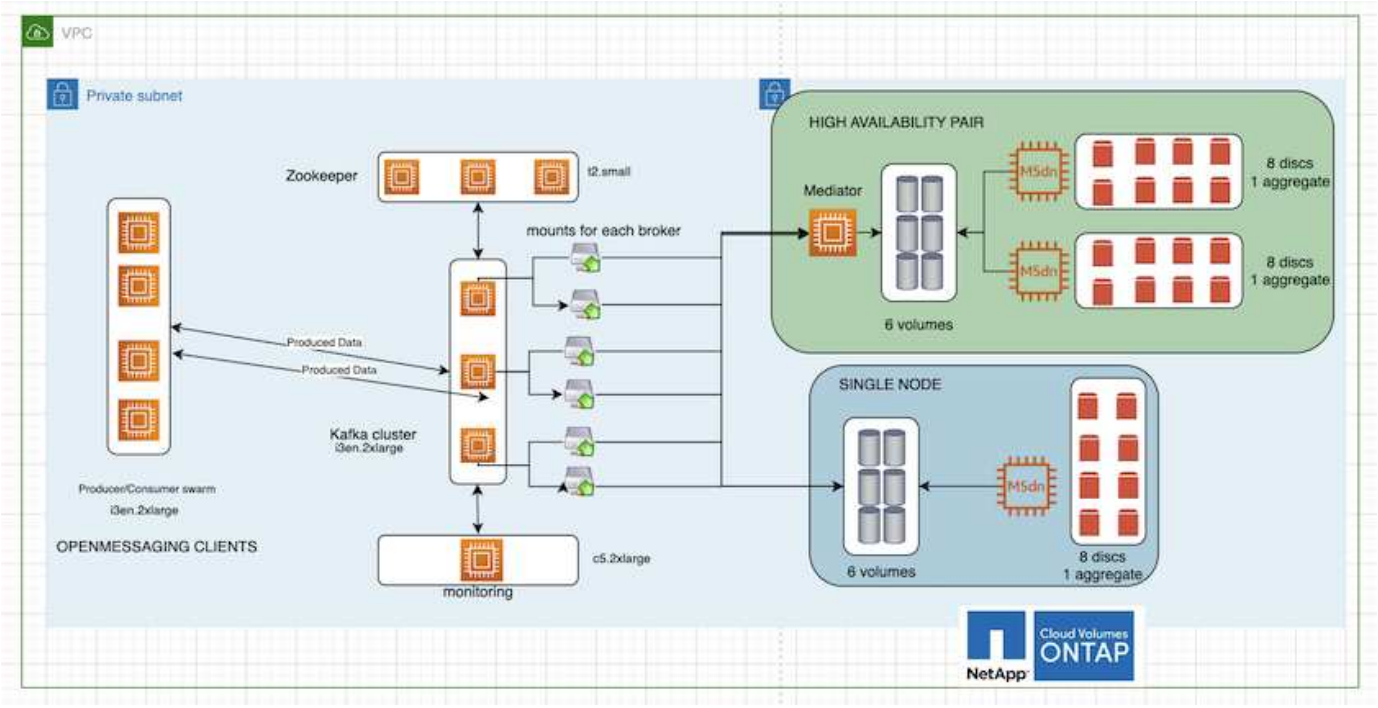

3. ONTAP NVRAM의 IOPS가 더 많으므로 Cloud Volumes ONTAP 루트 볼륨의 IOPS를 2350으로 변경했습니다. Cloud Volumes ONTAP의 루트 볼륨 디스크 크기는 47GB였습니다. 다음 ONTAP 명령은 HA 쌍용이며 단일 노드에도 동일한 단계를 적용할 수 있습니다.

```
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 그림에서는 NAS 기반 Kafka 클러스터의 아키텍처를 보여 줍니다.

- * Compute. * 3노드 Kafka 클러스터와 전용 서버에서 실행되는 3노드 zookeeper 앙상블이 사용되었습니다. 각 브로커에는 전용 LIF를 통해 Cloud Volumes ONTAP 인스턴스의 단일 볼륨에 대한 NFS 마운트 지점이 2개 있었습니다.
- * 모니터링. * Prometheus-Grafana 조합에 두 개의 노드를 사용했습니다. 워크로드를 생성하는데 이 Kafka 클러스터를 생성하고 사용할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- * 스토리지. * 인스턴스에 하나의 6TB GP3 AWS-EBS 볼륨이 마운트된 HA 쌍 Cloud Volumes ONTAP 인스턴스를 사용했습니다. 그런 다음 NFS 마운트를 사용하여 Kafka 브로커로 볼륨을 내보냅니다.




**Openmessage** 벤치마킹 구성

1. NFS 성능을 개선하려면 nconnect를 사용하여 생성할 수 있는 NFS 서버와 NFS 클라이언트 간에 더 많은 네트워크 연결이 필요합니다. 다음 명령을 실행하여 브로커 노드에서 nconnect 옵션으로 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 ONTAP에서 네트워크 연결을 확인합니다. 다음 ONTAP 명령은 단일 Cloud Volumes ONTAP 노드에서 사용됩니다. Cloud Volumes ONTAP HA 쌍에도 동일한 단계가 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. 다음 Kafka를 사용합니다 `server.properties` 모든 Kafka 브로커는 Cloud Volumes ONTAP HA 페어를 지원합니다. 를 클릭합니다 `log.dirs` 각 브로커에 따라 속성이 다르며 나머지 속성은 브로커에 공통입니다. broker1의 경우, 를 참조하십시오 `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 ~]#
```

   - broker2의 경우, 를 참조하십시오 `log.dirs` 속성 값은 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
```

   - broker3의 경우, 를 참조하십시오 `log.dirs` 속성 값은 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 Kafka입니다 `servers.properties` 은(는) 를 제외하고 Cloud Volumes ONTAP HA 쌍과 동일합니다 `log.dirs` 속성.
   - broker1의 경우, 를 참조하십시오 `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
```

   - broker2의 경우, 를 참조하십시오 `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
```

   - broker3의 경우, 를 참조하십시오 `log.dirs` 속성 값은 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 워크로드는 다음과 같은 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
```

를 클릭합니다 `messageSize` 사용 사례마다 다를 수 있습니다. 성능 테스트에서는 3K를 사용했습니다.

OMB에서 서로 다른 두 드라이버, 즉 Sync 또는 Throughput을 사용하여 Kafka 클러스터에서 워크로드를 생성했습니다.

- Sync 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
```

- 처리량 운전자 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 방법

1. Kafka 클러스터는 Terraform 및 Ansible을 사용하여 위에서 설명한 사양에 따라 프로비저닝되었습니다. Terraform은 Kafka 클러스터용 AWS 인스턴스를 사용하여 인프라를 구축하는 데 사용되며, Ansible은 Kafka 클러스터를 기반으로 합니다.
2. 위에 설명된 워크로드 구성과 동기화 드라이버로 OMB 워크로드가 트리거되었습니다.

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

3. 동일한 워크로드 구성의 처리량 드라이버에서 또 다른 워크로드가 트리거되었습니다.

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

관찰

NFS에서 실행되는 Kafka 인스턴스의 성능을 벤치마크하는 워크로드를 생성하는 데 두 가지 유형의 드라이버가 사용되었습니다. 드라이버의 차이점은 로그 플러시 속성입니다.

Cloud Volumes ONTAP HA 쌍:

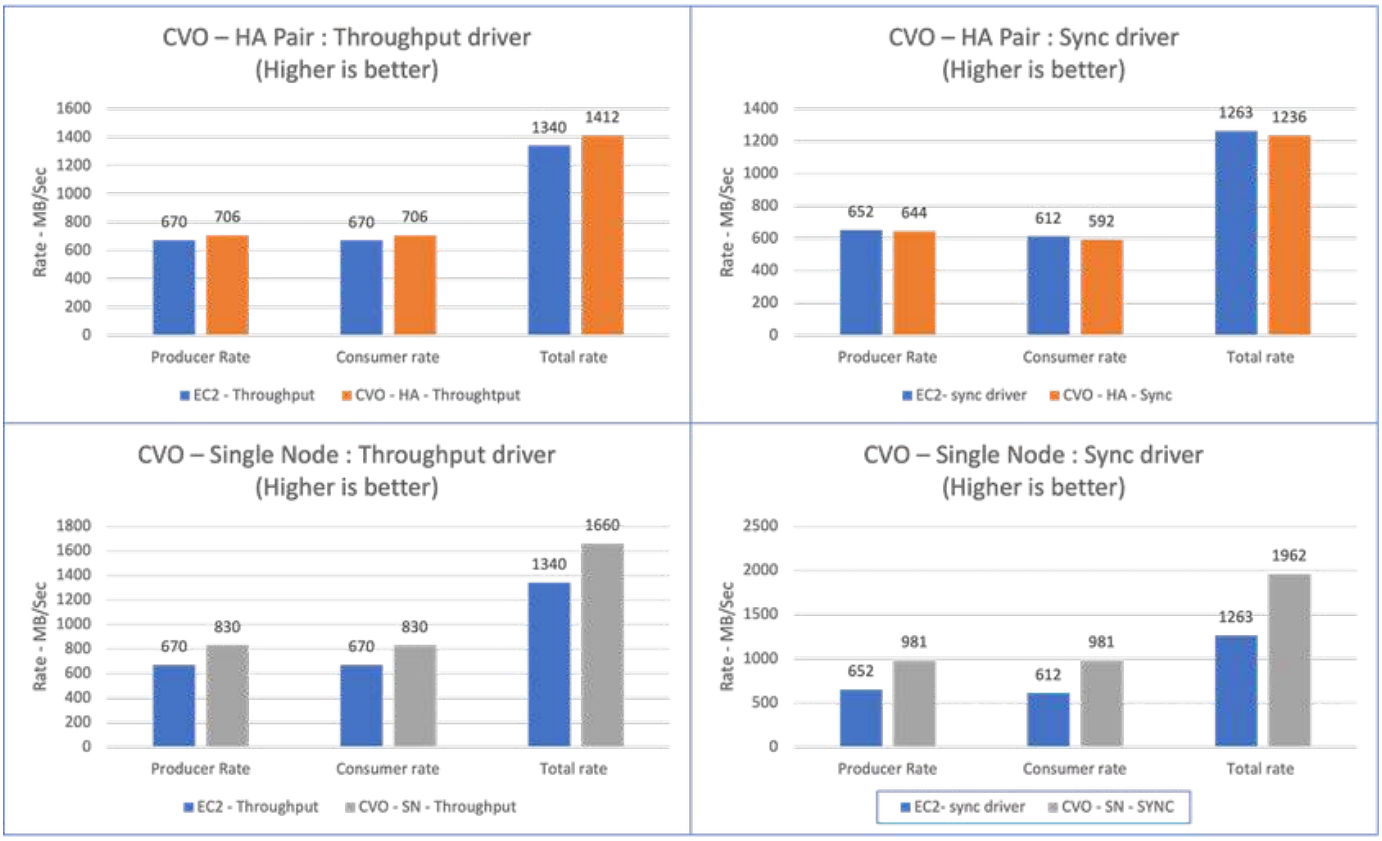
- Sync 드라이버에서 일관되게 생성된 총 처리량: ~1236Mbps.
- 처리량 드라이버에 대해 생성된 총 처리량: 최대 1412Mbps.

단일 Cloud Volumes ONTAP 노드의 경우:

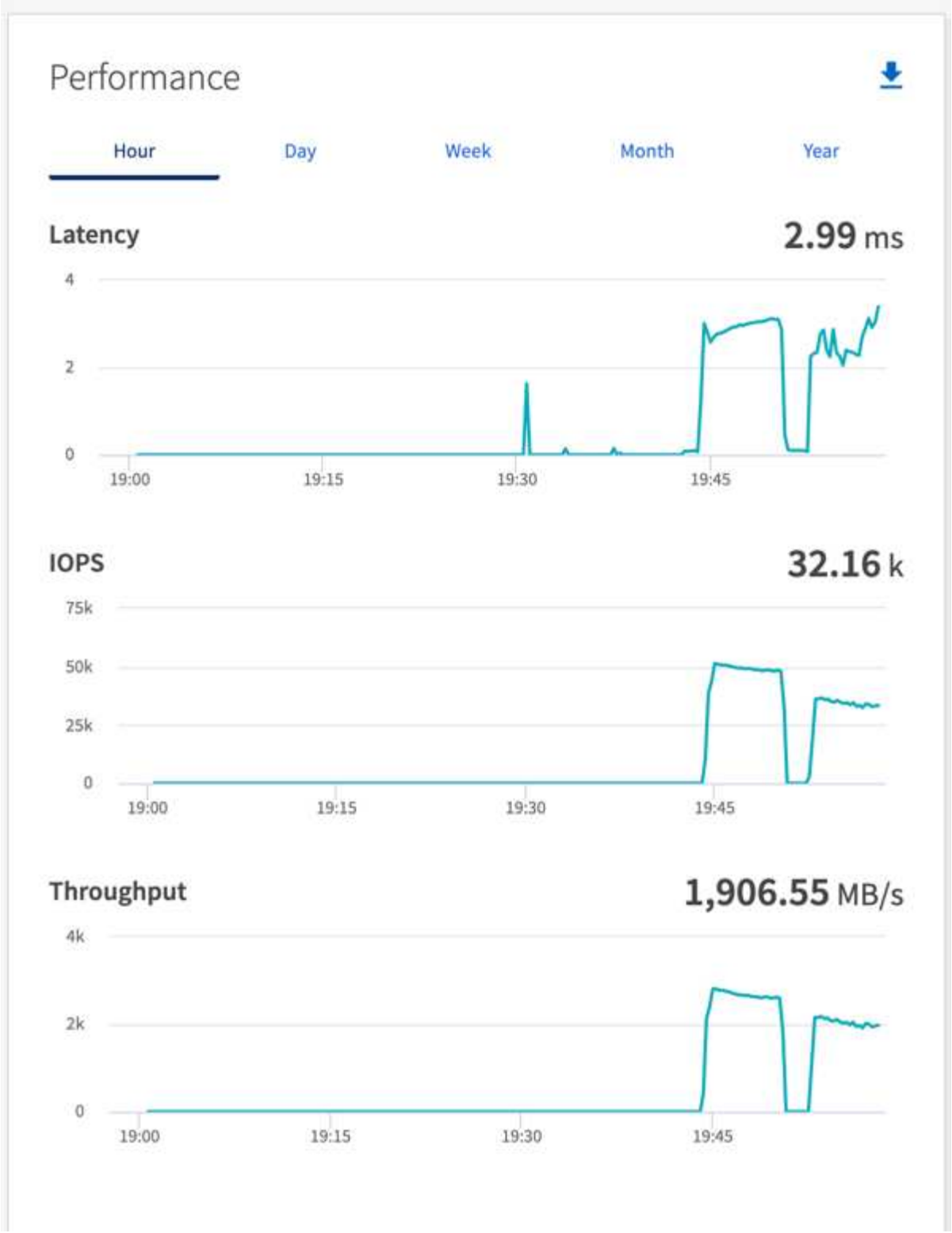
- Sync 드라이버에서 일관되게 생성된 총 처리량: ~1962MBps
- 처리량 드라이버에서 생성된 총 처리량: 최대 1660MBps

Sync 드라이버는 로그가 디스크에 즉시 플러시될 때 일관된 처리량을 생성할 수 있는 반면, 처리량 드라이버는 로그가 대량으로 디스크에 커밋될 때 처리량 버스트를 생성합니다.

이러한 처리량 수치는 지정된 AWS 구성에 대해 생성됩니다. 더 높은 성능 요구 사항을 위해 더 나은 처리량 수치를 위해 인스턴스 유형을 확장하고 조정할 수 있습니다. 총 처리량 또는 총 속도는 생산자와 소비자 속도의 조합입니다.




처리량 또는 동기화 드라이버 벤치마킹 수행 시 스토리지 처리량을 확인하십시오.




**NetApp ONTAP용 AWS FSx**의 성능 개요 및 검증

NetApp NFS에 스토리지 계층이 마운트된 Kafka 클러스터가 NetApp ONTAP용 AWS FSx의 성능을 벤치마킹했습니다. 벤치마킹 예는 다음 섹션에 설명되어 있습니다.

### NetApp ONTAP용 AWS FSx의 Apache Kafka

NFS(Network File System)는 대량의 데이터를 저장하는 데 널리 사용되는 네트워크 파일 시스템입니다. 대부분의 조직에서 데이터는 Apache Kafka와 같은 스트리밍 애플리케이션에 의해 점점 더 많이 생성되고 있습니다. 이러한 워크로드는 최신 스토리지 기능을 갖춘 확장성, 짧은 지연 시간, 강력한 데이터 수집 아키텍처를 필요로 합니다. 실시간 분석을 지원하고 실행 가능한 통찰력을 제공하기 위해 우수한 설계 및 고성능 인프라가 필요합니다.

Kafka는 POSIX 호환 파일 시스템과 호환되며 파일 시스템에 의존하여 파일 작업을 처리하지만 NFSv3 파일 시스템에 데이터를 저장할 때 Kafka 브로커 NFS 클라이언트는 XFS 또는 ext4 같은 로컬 파일 시스템과 다르게 파일 작업을 해석할 수 있습니다. 일반적인 예로는 클러스터를 확장하고 파티션을 다시 할당할 때 Kafka 브로커가 실패하는 NFS의 이름이 있습니다. 이러한 문제를 해결하기 위해 NetApp은 RHEL8.7, RHEL9.1에서 일반적으로 사용할 수 있는 변경 사항으로 오픈 소스 Linux NFS 클라이언트를 업데이트했으며 현재 NetApp ONTAP용 FSx 릴리즈 ONTAP 9.12.1에서 지원합니다.

NetApp ONTAP용 Amazon FSx는 클라우드에서 완벽하게 관리되고 확장 가능하며 뛰어난 성능의 NFS 파일 시스템을 제공합니다. NetApp ONTAP용 FSx의 Kafka 데이터는 대량의 데이터를 처리하고 내결함성을 보장하기 위해 확장할 수 있습니다. NFS는 중요하거나 민감한 데이터 세트에 대해 중앙 집중식 스토리지 관리 및 데이터 보호를 제공합니다.

이러한 향상된 기능을 통해 AWS 고객은 AWS 컴퓨팅 서비스에서 Kafka 워크로드를 실행할 때 NetApp ONTAP용 FSx를 활용할 수 있습니다. 다음과 같은 이점을 제공합니다.
* CPU 활용률을 줄여 I/O 대기 시간을 줄입니다
* 더 빠른 Kafka 브로커 복구 시간.
* 안정성 및 효율성.
* 확장성 및 성능.
* 다중 가용성 영역 가용성.
* 데이터 보호.

#### NetApp ONTAP용 AWS FSx의 성능 개요 및 검증

NetApp NFS에 스토리지 계층이 마운트된 Kafka 클러스터가 AWS 클라우드의 성능을 벤치마킹했습니다. 벤치마킹 예는 다음 섹션에 설명되어 있습니다.

### NetApp ONTAP용 AWS FSx의 Kafka

NetApp ONTAP용 AWS FSx가 있는 Kafka 클러스터는 AWS 클라우드에서 성능을 벤치마킹했습니다. 이 벤치마킹은 다음 섹션에서 설명합니다.

아키텍처 설정

다음 표에는 NetApp ONTAP용 AWS FSx를 사용하는 Kafka 클러스터의 환경 구성이 나와 있습니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Kafka 3.2.3 | • 3 x zookeers – T2.small<br>• 브로커 서버 3대 – i3en.2xLarge<br>• Grafana 1개 – c5n.2xLarge<br>• 생산자/소비자 4대 — c5n.2xLarge * |
| 모든 노드의 운영 체제 | RHEL8.6 |
| NetApp ONTAP용 AWS FSx | 4GB/sec의 처리량과 160000 IOPS의 Multi-AZ |

**NetApp ONTAP 설정용 NetApp FSx**

1. 초기 테스트를 위해 2TB 용량과 40000 IOPS로 2GB/sec 처리량의 NetApp ONTAP 파일 시스템에 대한 FSx를 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"}
```

   이 예에서는 AWS CLI를 통해 NetApp ONTAP용 FSx를 구축하고 있습니다. 필요에 따라 사용자 환경에서 명령을 추가로 사용자 지정해야 합니다. NetApp ONTAP용 FSX는 AWS 콘솔을 통해 추가로 구축 및 관리할 수 있어 명령줄 입력 작업을 줄이고 더욱 쉽고 능률적인 배포 환경을 제공합니다.

   FSx for NetApp ONTAP에 대한 설명서이며, 테스트 영역(US-East-1)의 2GB/sec 처리 파일 시스템에 대해 달성 가능한 최대 IOPS는 80,000 IOPS입니다. NetApp ONTAP 파일 시스템용 FSx의 총 최대 IOPS는 160,000 IOPS이며, 이를 위해서는 4GB/sec의 처리량 구축이 필요하며, 이 문서의 후반부에 이 내용이 설명되어 있습니다.

   NetApp ONTAP 성능 사양에 대한 FSx에 대한 자세한 내용은 여기 에서 AWS FSx for NetApp ONTAP 설명서를 참조하십시오. https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html .

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

   예를 들어 KMS 키를 지정하지 않을 때 사용되는 기본 AWS FSx 마스터 키가 아니라 특정 KMS 키를 지정할 수 있습니다.

2. NetApp ONTAP 파일 시스템에 대한 FSx를 생성하는 동안 다음과 같이 파일 시스템을 설명한 후 JSON 반환에서 "lifeCycle" 상태가 "Available"으로 변경될 때까지 기다립니다.

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

3. fsxadmin 사용자로 NetApp ONTAP SSH용 FSX에 로그인하여 자격 증명을 검증합니다. Fsxadmin은 생성 시 NetApp ONTAP 파일 시스템용 FSx의 기본 관리자 계정입니다. fsxadmin의 암호는 1단계에서 완료한 대로 AWS 콘솔 또는 AWS CLI에서 파일 시스템을 처음 생성할 때 구성된 암호입니다.

```
[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 확인되면 NetApp ONTAP 파일 시스템용 FSx에 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(스토리지 가상 시스템)은 NetApp ONTAP 볼륨용 FSx의 데이터를 관리 및 액세스하기 위한 자체 관리 자격 증명과 엔드포인트를 갖춘 격리된 파일 서버이며 NetApp ONTAP 멀티 테넌시를 위한 FSx를 제공합니다.

5. 기본 스토리지 가상 머신을 구성한 후에는 새로 생성된 NetApp ONTAP 파일 시스템용 FSx에 SSH를 사용하여 스토리지 가상 머신에 볼륨을 생성하고, 이와 유사하게 이 검증을 위해 6개의 볼륨을 생성합니다. 당사의 검증을 기반으로 Kafka에 더 나은 성능을 제공할 수 있는 기본 구성 요소(8) 이하를 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 위해 볼륨에 추가 용량이 필요합니다. 볼륨의 크기를 2TB로 확장하고 접합 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
```

NetApp ONTAP용 FSx에서는 볼륨을 씬 프로비저닝할 수 있습니다. 이 예에서는 총 확장 볼륨 용량이 총 파일 시스템 용량을 초과하므로 다음 단계에서 보여 드릴 추가 프로비저닝 볼륨 용량을 잠금 해제하려면 전체 파일 시스템 용량을 확장해야 합니다.

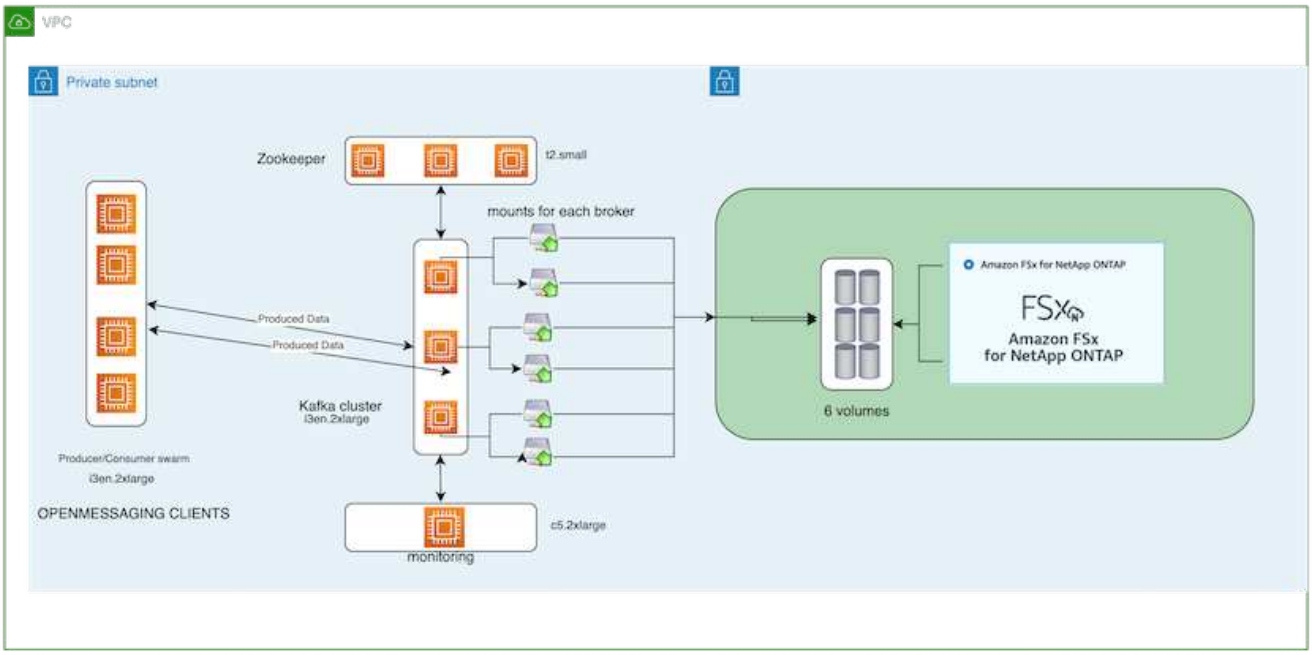
7. 다음으로 성능 및 용량을 늘리기 위해 NetApp ONTAP용 FSx 처리량 용량을 2GB/sec에서 4GB/sec로, IOPS는 160000으로, 용량은 5TB로 확장합니다

```
[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 찾을 수 있습니다. https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html

8. NetApp ONTAP 볼륨용 FSx는 Kafka 브로커에서 nconnect 및 기본 opions로 마운트됩니다

다음 그림은 NetApp ONTAP 기반 Kafka 클러스터용 FSx의 최종 아키텍처를 보여줍니다.




- 컴퓨팅. 3노드 Kafka 클러스터를 전용 서버에서 실행되는 3노드 zookeeper 앙상블과 함께 사용했습니다. 각

브로커는 NetApp ONTAP 인스턴스의 FSx에 있는 6개의 볼륨에 6개의 NFS 마운트 지점을 가지고 있었습니다.

- 모니터링. 두 개의 노드를 사용하여 Prometheus-Grafana 조합을 사용했습니다. 워크로드를 생성하는데 이 Kafka 클러스터를 생성하고 사용할 수 있는 별도의 3노드 클러스터를 사용했습니다.
- 스토리지. 2TB 볼륨 6개가 마운트된 NetApp ONTAP용 FSx를 사용했습니다. 그런 다음 NFS 마운트를 사용하여 Kafka 브로커로 볼륨을 내보냅니다. NetApp ONTAP 볼륨의 FSx는 Kafka 브로커에서 16개의 nconnect 세션과 기본 옵션으로 마운트됩니다.

**Openmessage 벤치마킹 구성.**

NetApp Cloud Volumes ONTAP에 사용된 것과 동일한 구성을 사용했으며 세부 정보는 다음과 같습니다. https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

테스트 방법

1. Kafka 클러스터는 Terraform 및 Ansible을 사용하여 위에서 설명한 사양에 따라 프로비저닝되었습니다. Terraform은 Kafka 클러스터용 AWS 인스턴스를 사용하여 인프라를 구축하는 데 사용되며, Ansible은 Kafka 클러스터를 기반으로 합니다.
2. 위에 설명된 워크로드 구성과 동기화 드라이버로 OMB 워크로드가 트리거되었습니다.

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

3. 동일한 워크로드 구성의 처리량 드라이버에서 또 다른 워크로드가 트리거되었습니다.

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

관찰

NFS에서 실행되는 Kafka 인스턴스의 성능을 벤치마크하는 워크로드를 생성하는 데 두 가지 유형의 드라이버가 사용되었습니다. 드라이버의 차이점은 로그 플러시 속성입니다.

Kafka 복제 계수 1 및 NetApp ONTAP용 FSx의 경우:

- Sync 드라이버에서 일관되게 생성된 총 처리량: ~3218Mbps 및 최대 성능: ~3652Mbps.
- 처리량 드라이버에서 일관되게 생성된 총 처리량: ~3679Mbps 및 최대 성능: ~3908Mbps.

복제 계수가 3인 Kafka 및 NetApp ONTAP용 FSx의 경우:

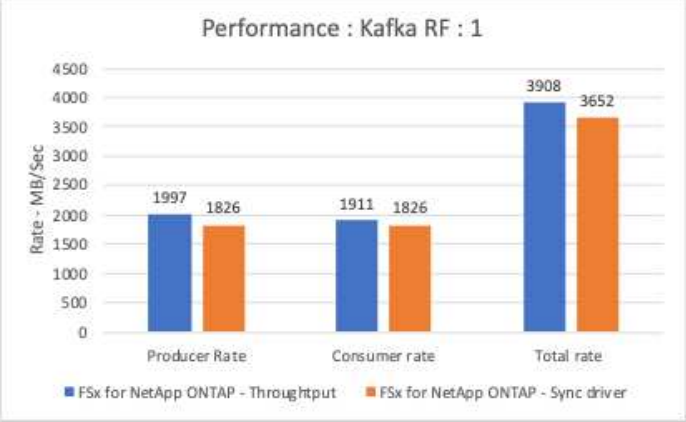
- Sync 드라이버에서 일관되게 생성된 총 처리량: ~1252Mbps 및 최대 성능: ~1382Mbps.
- 처리량 드라이버에서 일관되게 생성된 총 처리량: ~1218Mbps 및 최대 성능: ~1328Mbps.

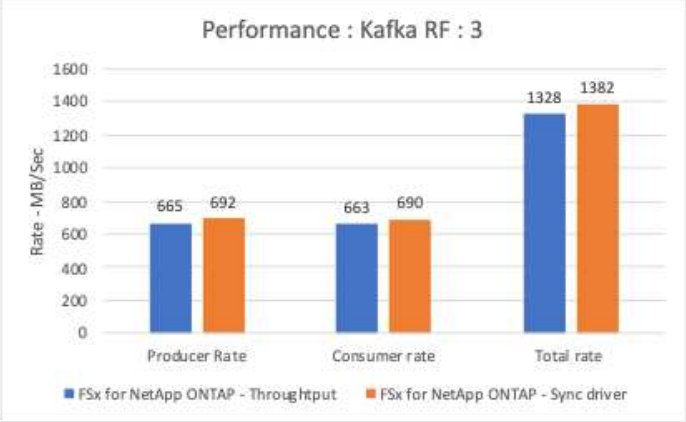
Kafka 복제 계수 3에서 읽기 및 쓰기 작업은 FSx for NetApp ONTAP에서 세 번, Kafka 복제 계수 1에서 NetApp ONTAP용 FSx에서 읽기 및 쓰기 작업이 한 번 수행되므로 두 검증 모두에서 최대 4GB/sec 처리량에 도달할 수

있습니다.

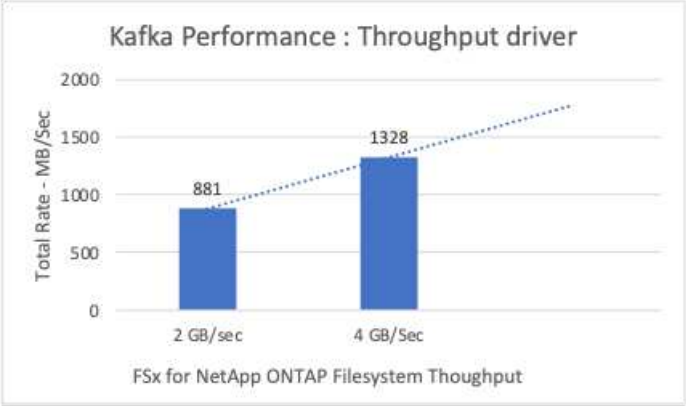
Sync 드라이버는 로그가 디스크에 즉시 플러시될 때 일관된 처리량을 생성할 수 있는 반면, 처리량 드라이버는 로그가 대량으로 디스크에 커밋될 때 처리량 버스트를 생성합니다.

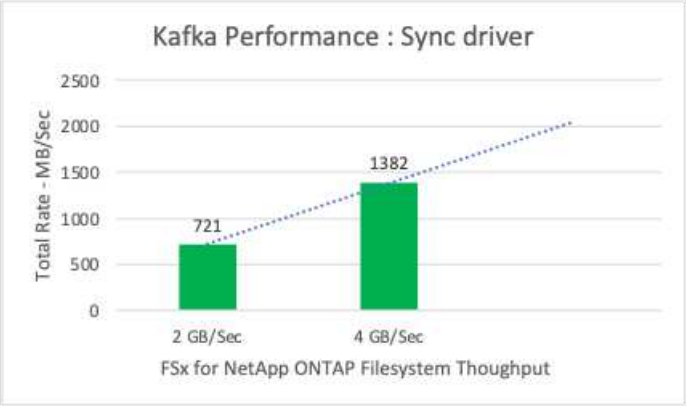
이러한 처리량 수치는 지정된 AWS 구성에 대해 생성됩니다. 더 높은 성능 요구 사항을 위해 더 나은 처리량 수치를 위해 인스턴스 유형을 확장하고 조정할 수 있습니다. 총 처리량 또는 총 속도는 생산자와 소비자 속도의 조합입니다.



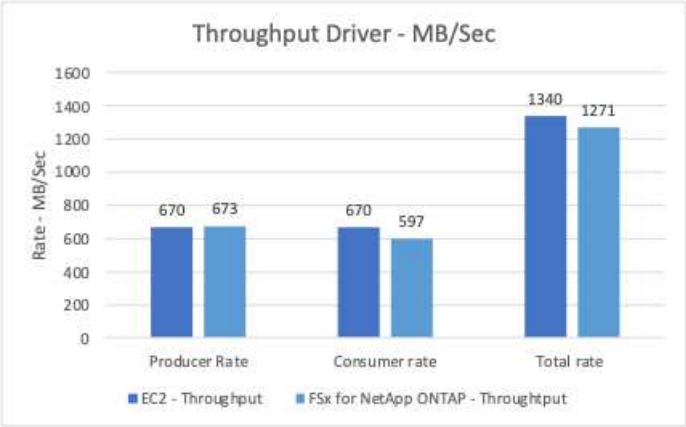



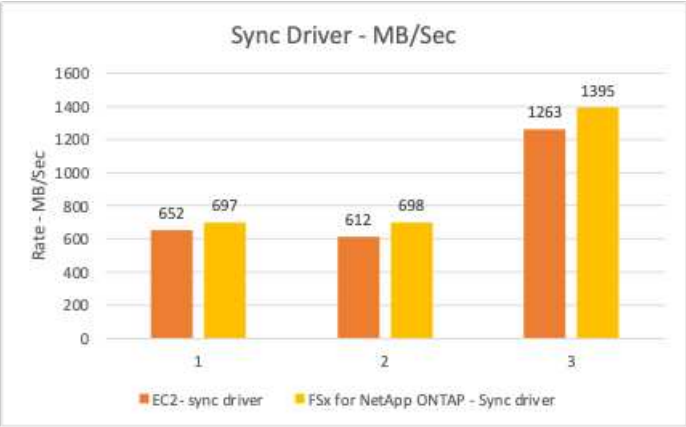

아래 차트는 NetApp ONTAP의 경우 2GB/sec FSx, Kafka 복제 계수 3의 경우 4GB/sec 성능을 보여 줍니다. 복제 계수 3은 NetApp ONTAP 스토리지용 FSx에서 읽기 및 쓰기 작업을 세 번 수행합니다. 처리량 드라이버의 총 속도는 881MB/sec이며, Kafka는 NetApp ONTAP 파일 시스템용 2GB/sec FSx에서 약 2.64GB/sec의 읽기 및 쓰기 작업을 수행하고 처리량 드라이버의 총 속도는 1328MB/sec이며, Kafka 작업은 약 3.98GB/sec입니다. 또한 Kafka 성능은 NetApp ONTAP 처리량을 위한 FSx를 기반으로 하는 선형적이고 확장성이 뛰어납니다.




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

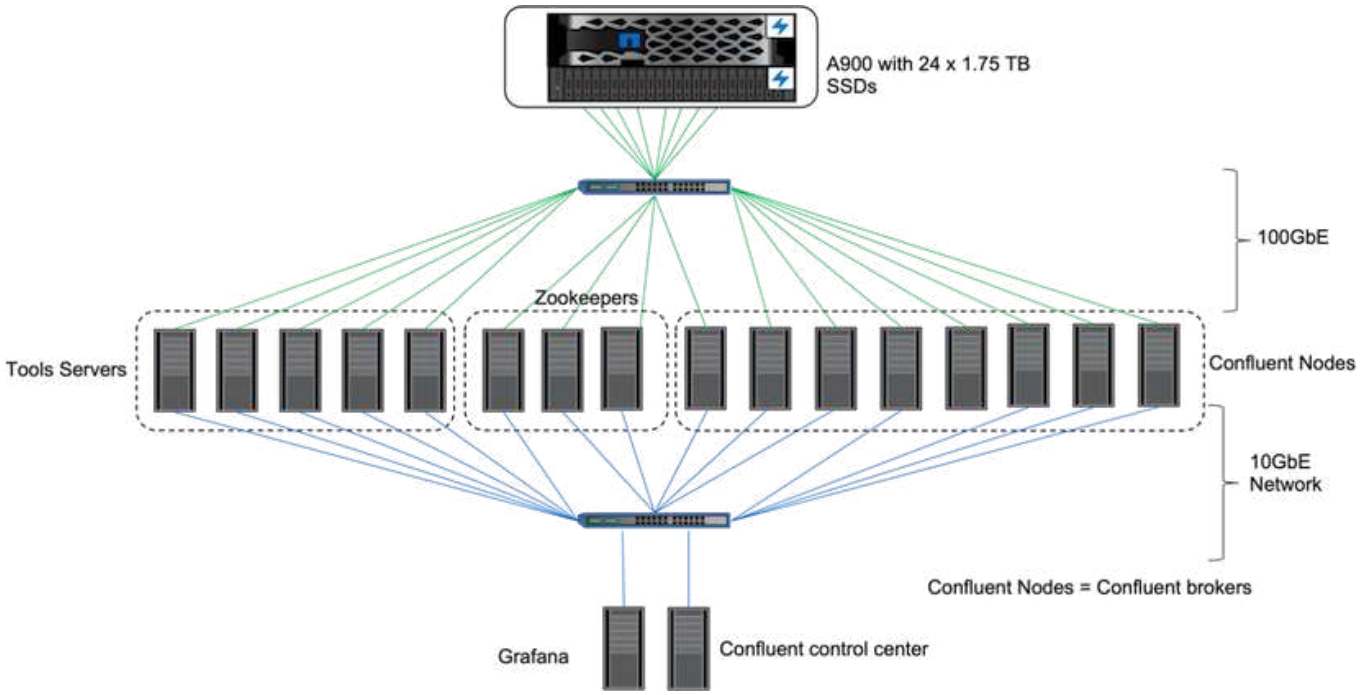



## **AFF A900** 온프레미스 성능 개요 및 검증

사내에서 NetApp AFF A900 스토리지 컨트롤러와 ONTAP 9.12.1 RC1을 사용하여 Kafka 클러스터의 성능 및 확장성을 검증했습니다. ONTAP 및 AFF의 이전 계층형 스토리지 모범 사례와 동일한 테스트 베드를 사용했습니다.

AFF A900을 평가하기 위해 Confluent Kafka 6.2.0을 사용했습니다. 클러스터에는 8개의 브로커 노드와 3개의 Zookeeper 노드가 있습니다. 성능 테스트를 위해 OMB 작업자 노드 5개를 사용했습니다.




스토리지 구성

NetApp FlexGroups 인스턴스를 사용하여 로그 디렉토리에 단일 네임스페이스를 제공하여 복구 및 구성을 간소화했습니다. NFSv4.1 및 pNFS를 사용하여 로그 세그먼트 데이터에 직접 액세스할 수 있습니다.

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, 을 늘렸습니다 `max_session_slots` 를 선택합니다 `64` 를 선택합니다 `180`. 이는 ONTAP의 기본 세션 슬롯 제한과 일치합니다.

**Kafka** 브로커 튜닝

테스트 중인 시스템의 처리량을 극대화하기 위해 특정 주요 스레드 풀에 대한 기본 매개 변수를 크게 늘렸습니다. 대부분의 구성에 대해 Confluent Kafka 모범 사례를 따르는 것이 좋습니다. 이 튜닝은 스토리지에 대한 미해결 입출력의 동시성을 극대화하는 데 사용되었습니다. 이러한 매개 변수는 브로커의 컴퓨팅 리소스 및 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 방법

처리량 드라이버 및 주제 구성에 대한 클라우드 테스트와 동일한 OMB 구성을 사용했습니다.

1. AFF 클러스터에서 Ansible을 사용하여 FlexGroup 인스턴스를 프로비저닝했습니다.

```
---
- 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 pNFS를 사용할 수 있게 되었습니다.

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

3. 워크로드는 Cloud Volumes ONTAP와 동일한 워크로드 구성을 사용하는 처리량 드라이버에서 트리거되었습니다. 자세한 내용은 " 단원을 참조하십시오안정적인 성능"아래에 있습니다. 워크로드는 복제 계수 3을 사용했습니다. 즉 NFS에서 로그 세그먼트의 복제본 3개가 유지되었습니다.

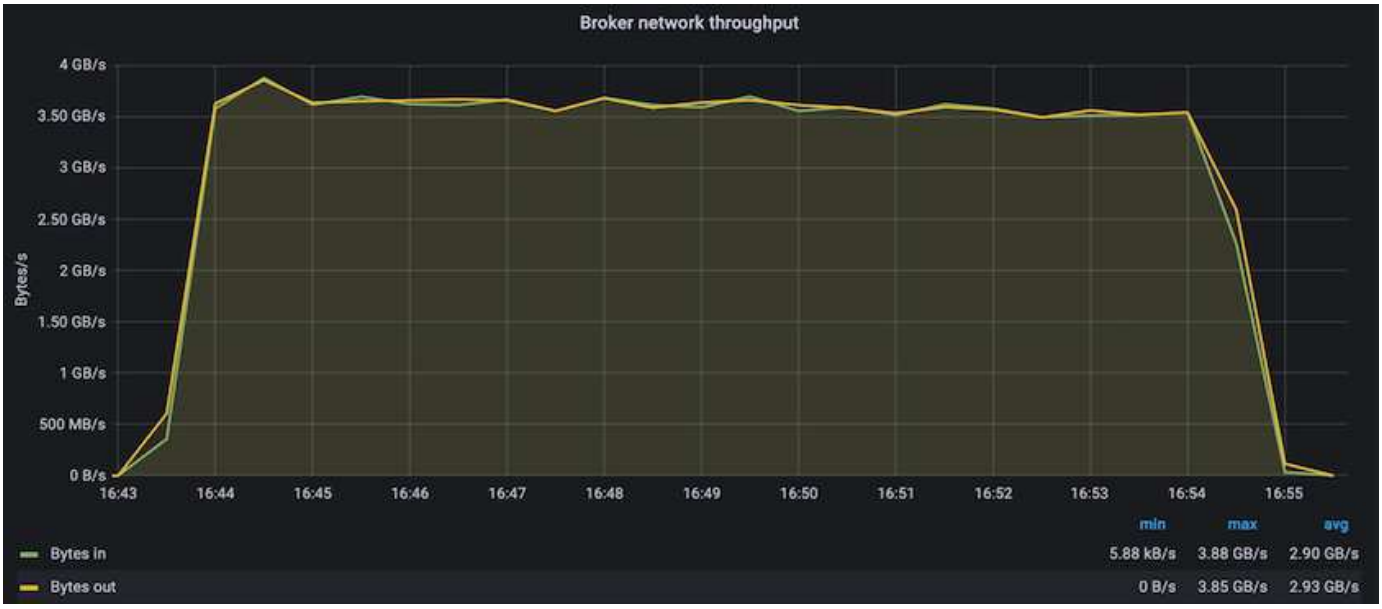
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 마지막으로, 백로그를 사용하여 소비자가 최신 메시지를 따라잡을 수 있는 능력을 측정하는 측정을 완료했습니다. OMB는 측정을 시작하는 동안 소비자를 일시 중지하여 백로그를 생성합니다. 이 경우 백로그 생성(생산자 전용 트래픽), 백로그 드레이닝(소비자가 주제에서 놓친 이벤트를 포착하는 소비자 중심 단계), 안정적 상태 등 세 가지 단계가 발생합니다. 자세한 내용은 " 단원을 참조하십시오스토리지 제한값 탐색"를 참조하십시오.

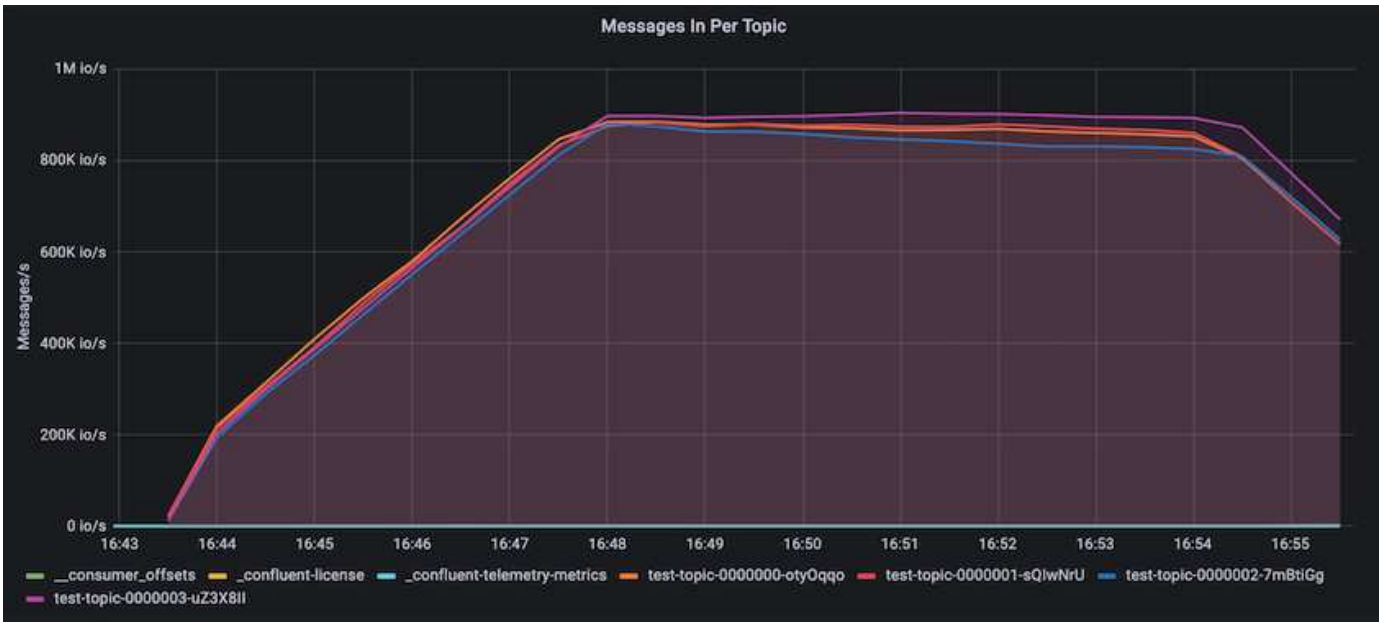
안정적인 성능

NetApp은 OpenMessaging 벤치마크를 사용하여 AFF A900을 평가하여 AWS의 Cloud Volumes ONTAP 및 AWS의 DAS와 유사한 비교를 제공했습니다. 모든 성능 값은 생산자 및 소비자 수준에서 Kafka 클러스터 처리량을 나타냅니다.

Confluent Kafka 및 AFF A900의 안정적인 상태 성능은 생산자와 소비자 모두의 평균 3.4GBps 처리량을 달성했습니다. 이는 Kafka 클러스터에서 340만 개 이상의 메시지입니다. BrokerTopicMetrics의 지속적인 처리량을 초당 바이트 단위로 시각화함으로써 AFF A900이 지원하는 뛰어난 안정적인 상태 성능과 트래픽을 확인할 수 있습니다.




이것은 주제별로 전달되는 메시지의 보기에 잘 부합됩니다. 다음 그래프는 주제별 분석을 제공합니다. 테스트를 거친 구성에서 4개 주제에 대해 주제당 약 900,000개의 메시지가 나타났습니다.

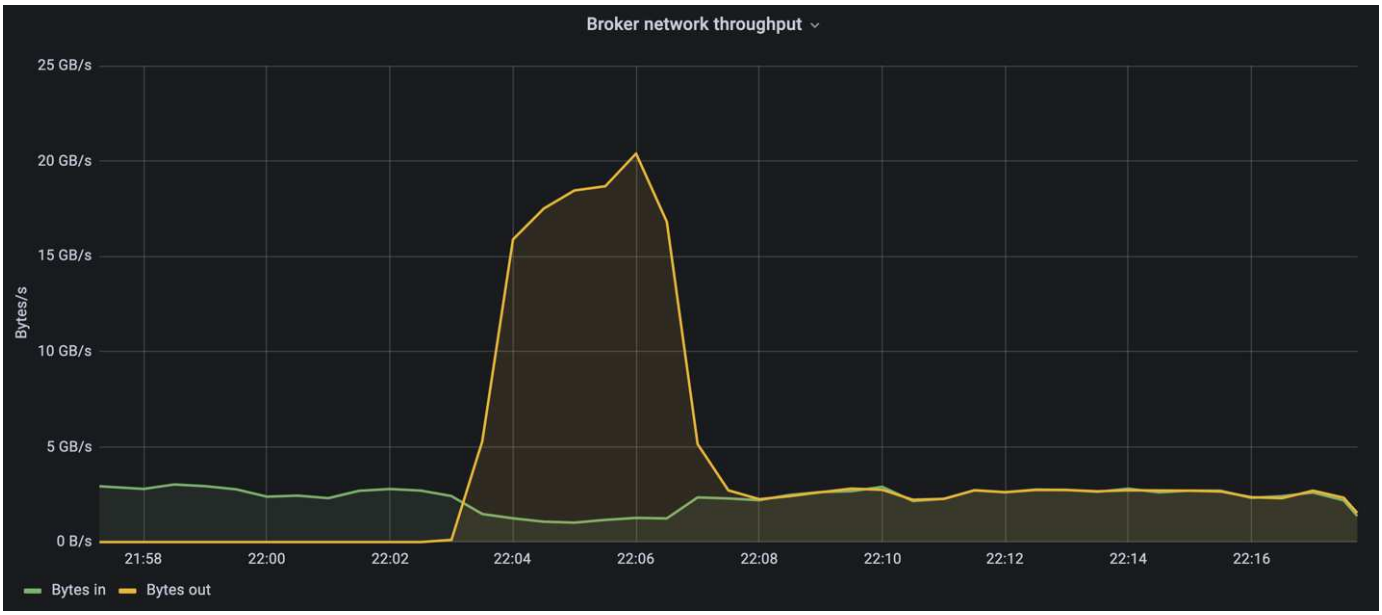



성능을 극대화 및 스토리지 제한 탐색

AFF의 경우 백로그 기능을 사용하여 OMB를 테스트했습니다. 백로그 기능은 Kafka 클러스터에 이벤트 백로그가 구축된 동안 소비자 가입을 일시 중지합니다. 이 단계에서는 프로듀서 트래픽만 발생하므로 로그에 커밋되는 이벤트가 생성됩니다. 이렇게 하면 일괄 처리 또는 오프라인 분석 워크플로를 가장 긴밀하게 에뮬레이트할 수 있습니다. 이러한 워크플로에서는 소비자 구독이 시작되고 브로커 캐시에서 이미 제거된 내역 데이터를 읽어야 합니다.

이 구성에서 소비자 처리량에 대한 저장소 제한을 이해하기 위해 A900이 흡수할 수 있는 쓰기 트래픽의 양을 파악하기 위해 생산자 전용 단계를 측정했습니다. 다음 섹션 " 을 참조하십시오사이징 지침"이 데이터를 활용하는 방법을 이해합니다.

생산자 전용 측정 과정에서 A900 성능의 한계를 넘어선 높은 피크 처리량을 볼 수 있었습니다(다른 브로커 리소스가 생산자 및 소비자 트래픽을 지원하는 데 포화 상태가 되지 않았을 때).




이 측정을 위해 메시지당 오버헤드를 제한하고 NFS 마운트 지점에 대한 스토리지 처리량을 최대화하기 위해 메시지 크기를 16K로 늘렸습니다.

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

Confluent Kafka 클러스터는 4.03GBps의 피크 프로듀서 처리량을 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 eventbacklog를 채운 후 소비자 트래픽이 다시 시작되었습니다. 백로그 드레이닝으로 측정하는 동안 모든 항목에서 20Gbps 이상의 최고 소비자 처리량을 관찰했습니다. OMB 로그 데이터를 저장하는 NFS 볼륨에 대한 총 처리량이 최대 30Gbps에 근접했습니다.

사이징 지침

Amazon Web Services에서 제공합니다 "사이징 가이드" Kafka 클러스터의 사이징 및 확장에 사용됩니다.

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

복제 계수가 r인 tcluster 클러스터로 생성되는 총 처리량의 경우 브로커 스토리지에서 수신한 처리량은 다음과 같습니다.

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

이 작업은 더욱 단순화될 수 있습니다.

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

이 공식을 사용하여 Kafka 핫 티어 요구에 적합한 ONTAP 플랫폼을 선택할 수 있습니다.

다음 표에는 여러 복제 요소를 사용하는 A900의 예상 생산자 처리량이 설명되어 있습니다.

| 복제 계수 | 생산자 처리량**(GPPS)** |
|---|---|
| 3(측정) | 3.4 |
| 2 | 5.1 |
| 1 | 10.2 |

## 결론

### 이름 바꾸기 비용이 저렴하다는 어리석은 문제를 위한 NetApp 솔루션은 이전에 NFS와 호환되지 않았던 워크로드를 위해 중앙에서 관리하는 단순한 형태의 스토리지를 제공합니다.

이 새로운 패러다임은 고객이 재해 복구 및 데이터 보호를 위해 마이그레이션 및 미러링이 용이한 보다 관리가 용이한

Kafka 클러스터를 생성할 수 있도록 해 줍니다.
또한 NFS는 CPU 활용률 감소, 복구 시간 단축, 스토리지 효율성 대폭 향상, NetApp ONTAP를 통한 성능 향상과 같은 추가 이점을 제공합니다.

## 추가 정보를 찾을 수 있는 위치

### 이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹 사이트를 검토하십시오.

- 아파치 카프카란?

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

- 이름이 바보 같습니까?

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

- ONATP는 스트리밍 애플리케이션에 대해 읽혀집니다.

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

- Kafka의 이름 변경 문제.

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

- NetApp 제품 설명서

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

- NFS란 무엇입니까?

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

- Kafka 파티션 재할당이란 무엇입니까?

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

- OpenMessaging 벤치마크란 무엇입니까?

  "https://openmessaging.cloud/"

- Kafka 브로커를 어떻게 마이그레이션합니까?

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

- Prometheus로 Kafka 브로커를 어떻게 모니터링하십니까?

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

- Apache Kafka용 관리 플랫폼

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

- 아파치 카프카 지원

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

- Apache Kafka용 컨설팅 서비스

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

# NetApp ONTAP 스토리지 컨트롤러를 사용하는 Confluent Kafka

## TR-4941: NetApp ONTAP 스토리지 컨트롤러에 대해 잘 알고 있습니다

Karthikeyan Nagalingam, Joe Scott, NetApp Rankesh Kumar, Confluent

Confluent Platform의 확장성과 탄력성을 크게 확보하려면 IT에서 워크로드를 매우 빠르게 확장하고 균형을 맞출 수 있어야 합니다. 계층형 스토리지를 사용하면 운영 부담을 줄여 대량의 데이터를 유창한 관리 방식으로 저장할 수 있습니다.

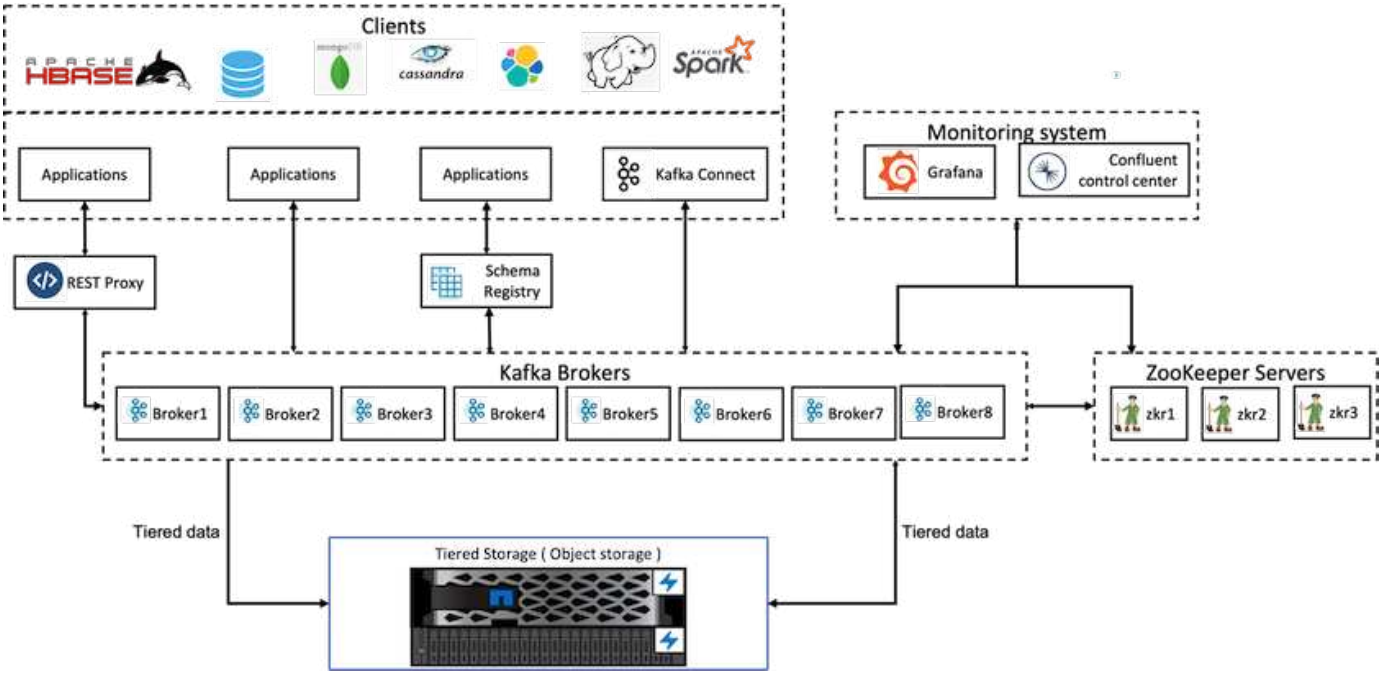
기본 개념은 데이터 스토리지와 데이터 처리를 분리하여 각각을 독립적으로 쉽게 확장할 수 있도록 하는 것입니다.

업계 최고의 혁신 기술을 갖춘 NetApp ONTAP 데이터 관리 소프트웨어는 데이터가 어디에 있든 다양한 이점을 유창하게 제공합니다.

이 문서에서는 계층형 스토리지 벤치마킹 키트를 사용하여 NetApp ONTAP에 대해 Confluent 플랫폼의 성능 벤치마크를 설명합니다.

### 해결 방법

ONTAP 기반 Confluent 및 NetApp AFF A900 스토리지 컨트롤러는 데이터 스트림을 위해 설계된 분산 시스템입니다. 두 가지 모두 수평적으로 확장 가능하고 내결함성이 있으며 부하 상태에서 뛰어난 성능을 제공합니다. 데이터 공간을 최소화하는 데이터 축소 기술로 분산된 데이터 스트리밍 및 스트림 처리에서 서로 보완하여 스토리지 비용을 절감합니다. AFF A900 스토리지 컨트롤러는 뛰어난 성능을 제공하는 동시에 컴퓨팅과 데이터 스토리지 리소스를 분리할 수 있도록 합니다. 따라서 시스템 관리가 단순화되고 리소스를 독립적으로 확장할 수 있습니다.




솔루션 아키텍처 세부 정보

이 섹션에서는 계층형 스토리지를 위한 NetApp ONTAP를 사용하여 Confluent Platform 배포에서 성능을 검증하는 데 사용되는 하드웨어 및 소프트웨어에 대해 설명합니다. 다음 표에서는 솔루션 아키텍처와 기본 구성 요소에 대해 설명합니다.

| 플랫폼 구성 요소 | 환경 구성 |
|---|---|
| Confluent 플랫폼 버전 6.2 | • 주키프 3개<br>• 브로커 서버 8대<br>• 도구 서버 5대<br>• Grafana 1개<br>• 제어 센터 1개 |
| 모든 노드의 운영 체제 | Linux(Ubuntu 18.04) |
| 웜 버킷용 NetApp ONTAP | • AFF A900 고가용성(HA) 쌍 1개<br>• 24 x 800 SSD 4개<br>• S3 프로토콜<br>• 100GbE |
| 15 Fujitsu Primergy RX2540 서버 | • CPU 2개, 총 물리적 코어 16개<br>• 인텔 제온<br>• 256GB 물리적 메모리<br>• 100GbE 이중 포트 |

## 기술 개요

이 섹션에서는 이 솔루션에 사용된 기술에 대해 설명합니다.

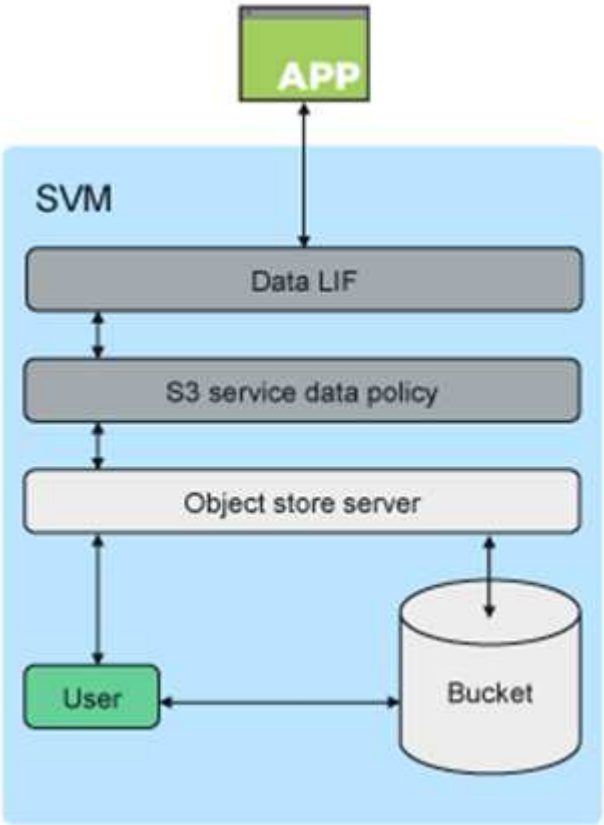
### **NetApp ONTAP** 스토리지 컨트롤러

NetApp ONTAP는 고성능 엔터프라이즈급 스토리지 운영 체제입니다.

NetApp ONTAP 9.8은 Amazon S3(Simple Storage Service) API를 지원합니다. ONTAP은 AWS(Amazon Web Services) S3 API 작업의 서브셋을 지원하며 클라우드 공급자(AWS, Azure 및 GCP) 및 온프레미스 전반에서 데이터를 ONTAP 기반 시스템에서 오브젝트로 나타낼 수 있습니다.

NetApp StorageGRID 소프트웨어는 오브젝트 스토리지를 위한 NetApp의 대표적인 솔루션입니다. ONTAP은 오브젝트 데이터를 위해 NetApp에서 제공하는 Data Fabric을 확장하고 NetApp 제품 포트폴리오의 가치를 높여 에지에서 수집 및 사전 처리 지점을 제공함으로써 StorageGRID를 보완합니다.

S3 버킷에 대한 액세스는 인증된 사용자 및 클라이언트 애플리케이션을 통해 제공됩니다. 다음 다이어그램에서는 S3 버킷에 액세스하는 애플리케이션을 보여 줍니다.




### 주요 사용 사례

S3 API를 지원하는 주된 목적은 ONTAP에서 오브젝트 액세스를 제공하는 것입니다. ONTAP 유니파이드 스토리지 아키텍처는 이제 파일(NFS 및 SMB), 블록(FC 및 iSCSI), 오브젝트(S3)를 지원합니다.

#### 네이티브 **S3** 애플리케이션

점점 더 많은 애플리케이션에서 S3를 사용하여 오브젝트 액세스에 ONTAP 지원을 활용할 수 있습니다. 고용량

아카이브 워크로드에 적합하지만 네이티브 S3 애플리케이션에서 고성능을 필요로 하는 것은 다음과 같습니다.

- 분석
- 인공 지능
- 엣지-코어 수집
- 머신 러닝

고객은 이제 ONTAP System Manager와 같은 익숙한 관리 효율 툴을 사용하여 ONTAP의 개발 및 운영에 필요한 고성능 오브젝트 스토리지를 신속하게 프로비저닝할 수 있습니다. 또한 ONTAP 스토리지 효율성 및 보안을 활용할 수 있습니다.

**FabricPool** 엔드포인트

ONTAP 9.8부터 FabricPool는 ONTAP의 버킷 계층화를 지원하므로 ONTAP에서 ONTAP로 계층화할 수 있습니다. 이 옵션은 기존 FAS 인프라를 오브젝트 저장소 엔드포인트로 용도 변경하고자 하는 고객에게 적합합니다.

FabricPool는 다음 두 가지 방법으로 ONTAP에 대한 계층화를 지원합니다.

- * 로컬 클러스터 계층화. * 비활성 데이터는 클러스터 LIF를 사용하여 로컬 클러스터에 있는 버킷으로 계층화합니다.
- * 원격 클러스터 계층화. * 비활성 데이터는 FabricPool 클라이언트의 IC LIF와 ONTAP 오브젝트 저장소의 데이터 LIF를 사용하여 기존 FabricPool 클라우드 계층과 비슷한 방식으로 원격 클러스터에 있는 버킷으로 계층화합니다.

ONTAP S3는 추가 하드웨어 및 관리 없이 기존 클러스터에서 S3 기능을 원하는 경우에 적합합니다. 300TB 이상의 배포를 위해 NetApp StorageGRID 소프트웨어는 계속해서 오브젝트 스토리지를 위한 주력 NetApp 솔루션으로 부상하고 있습니다. ONTAP 또는 StorageGRID를 클라우드 계층으로 사용할 때는 FabricPool 라이센스가 필요하지 않습니다.

#### **Confluent** 계층형 스토리지를 위한 **NetApp ONTAP**

모든 데이터 센터는 비즈니스 크리티컬 애플리케이션을 계속 실행하고 중요한 데이터를 안전하게 사용할 수 있어야 합니다. 새로운 NetApp AFF A900 시스템은 ONTAP 엔터프라이즈 에디션 소프트웨어와 복원력이 뛰어난 설계를 기반으로 합니다. NetApp의 새로운 초고속 NVMe 스토리지 시스템은 미션 크리티컬 운영의 중단을 제거하고, 성능 조정을 최소화하고, 랜섬웨어 공격으로부터 데이터를 보호합니다.

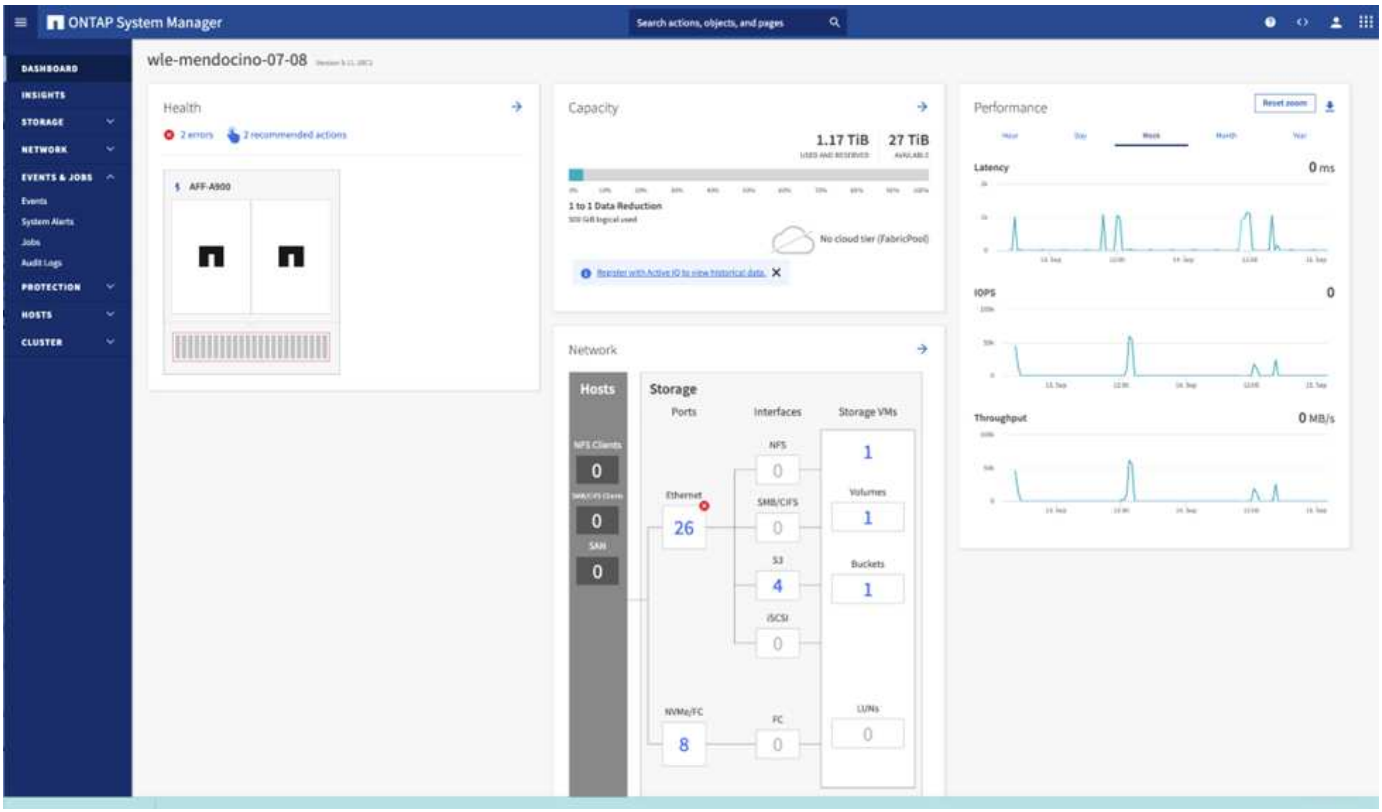
초기 구축부터 Confluent 클러스터의 확장까지, 귀사의 환경에서는 비즈니스 크리티컬 애플리케이션의 중단 없는 변화에 빠르게 적응해야 합니다. ONTAP 엔터프라이즈 데이터 관리, 서비스 품질(QoS) 및 성능 을 통해 고객 환경을 계획 및 조정할 수 있습니다.

NetApp ONTAP와 Confluent Tiered Storage를 함께 사용하면 ONTAP를 스케일아웃 스토리지 대상으로 활용하여 Apache Kafka 클러스터의 관리를 단순화하고 컴퓨팅 및 스토리지 리소스를 Confluent로 독립적으로 확장할 수 있습니다.

ONTAP S3 서버는 ONTAP의 성숙한 스케일아웃 스토리지 기능을 기반으로 구축되었습니다. ONTAP 클러스터는 S3 버킷을 확장하여 ONTAP 클러스터에 새로 추가된 노드를 사용함으로써 원활하게 확장할 수 있습니다.

#### **ONTAP** 시스템 관리자를 사용하여 간단하게 관리할 수 있습니다

ONTAP System Manager는 브라우저 기반의 그래픽 인터페이스로, 단일 창에서 전 세계에 분산된 위치에서 ONTAP 스토리지 컨트롤러를 구성, 관리, 모니터링할 수 있습니다.




System Manager 및 ONTAP CLI를 사용하여 ONTAP S3를 구성 및 관리할 수 있습니다. System Manager를 사용하여 S3를 활성화하고 버킷을 생성할 때 ONTAP은 단순한 구성에 대한 모범 사례 기본값을 제공합니다. CLI에서 S3 서버 및 버킷을 구성할 경우에도 원하는 경우 System Manager로 관리하거나 그 반대로 구성할 수 있습니다.

System Manager를 사용하여 S3 버킷을 생성하는 경우, ONTAP은 시스템에서 가장 가용성이 높은 기본 성능 서비스 수준을 구성합니다. 예를 들어 AFF 시스템에서 기본 설정은 Extreme입니다. 성능 서비스 수준은 사전 정의된 적응형 QoS 정책 그룹입니다. 기본 서비스 수준 대신 사용자 지정 QoS 정책 그룹 또는 정책 그룹을 지정할 수 있습니다.

사전 정의된 적응형 QoS 정책 그룹은 다음과 같습니다.

- * Extreme. * 가장 짧은 지연 시간과 최고의 성능이 필요한 애플리케이션에 사용됩니다.
- 성능. * 적절한 성능 요구사항과 지연 시간이 필요한 애플리케이션에 사용됩니다.
- * 값. * 처리량과 용량이 지연 시간보다 더 중요한 애플리케이션에 사용됩니다.
- * 사용자 정의. * 사용자 정의 QoS 정책을 지정하거나 QoS 정책을 지정하지 않습니다.

계층화에 * 사용을 선택하면 성능 서비스 수준이 선택되지 않으며 시스템은 계층형 데이터에 대해 최적의 성능을 갖춘 저비용 미디어를 선택합니다.

ONTAP는 가장 적합한 디스크가 있는 로컬 계층에서 이 버킷을 프로비저닝하려고 시도하여 선택한 서비스 수준을 충족시킵니다. 그러나 버킷에 포함할 디스크를 지정해야 하는 경우 로컬 계층(애그리게이트)을 지정하여 CLI에서 S3 오브젝트 스토리지를 구성하는 것이 좋습니다. CLI에서 S3 서버를 구성할 경우에도 원할 경우 System Manager로 관리할 수 있습니다.

버킷에 사용할 애그리게이트를 지정할 수 있는 기능은 CLI를 통해서만 지정할 수 있습니다.

유창하게

Confluent Platform은 지속적인 실시간 스트림으로 데이터에 쉽게 액세스, 저장 및 관리할 수 있는 포괄적인 데이터 스트리밍 플랫폼입니다. Apache Kafka를 처음 개발한 Confluent는 Kafka 관리 또는 모니터링의 부담을 덜면서 엔터프라이즈급 기능을 통해 Kafka의 이점을 확장해 줍니다. 현재 Fortune 100대 기업 중 80% 이상이 데이터 스트리밍 기술을 사용하고 있으며 대부분 Confluent를 사용하고 있습니다.

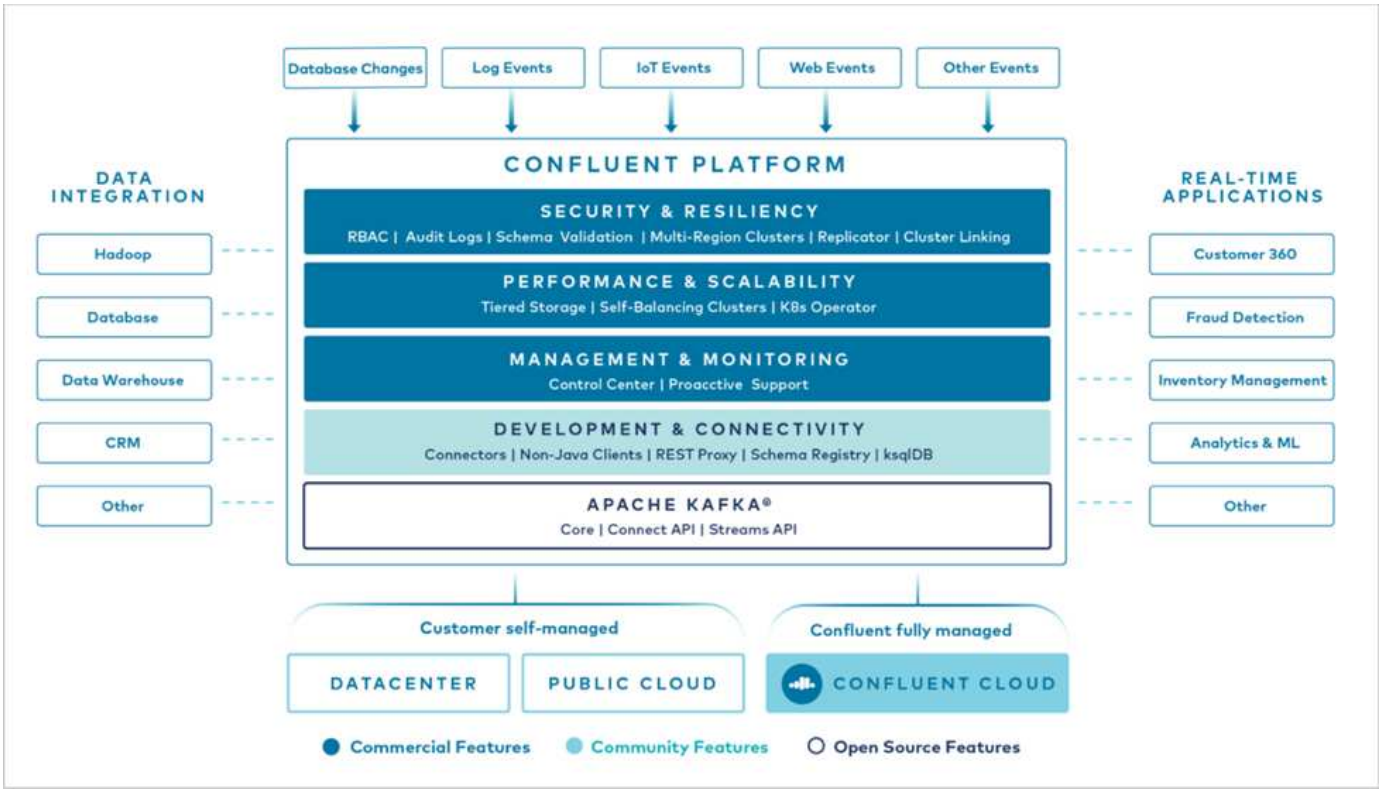
#### 왜 **Confluent**인가**?**

Confluorent는 기록 데이터와 실시간 데이터를 단일 중앙 데이터 소스에 통합하여 완전히 새로운 범주의 최신 이벤트 기반 애플리케이션을 쉽게 구축하고, 범용 데이터 파이프라인을 구축하며, 완전한 확장성, 성능, 안정성으로 강력한 새 사용 사례를 활용할 수 있도록 지원합니다.

#### **Contfluent**는 어떤 용도로 사용됩니까**?**

Confluent Platform을 사용하면 데이터가 다른 시스템 간에 어떻게 전송 또는 통합되는지 등의 기본 메커니즘을 걱정하지 않고 데이터에서 비즈니스 가치를 창출하는 방법에 집중할 수 있습니다. 특히 Confluent Platform은 데이터 소스를 Kafka에 연결하고 스트리밍 애플리케이션을 구축하며 Kafka 인프라의 보안, 모니터링 및 관리를 간소화합니다. 현재 Confluent Platform은 금융 서비스, 옴니채널 소매, 자율 자동차, 사기 탐지, 마이크로서비스, IoT 등 다양한 산업 전반의 다양한 사용 사례에 사용됩니다.

다음 그림에서는 Confluent 플랫폼의 구성 요소를 보여 줍니다.




#### **Confluent** 이벤트 스트리밍 기술 개요

Confluent Platform의 핵심은 입니다 "카프카"가장 인기 있는 오픈 소스 분산 스트리밍 플랫폼입니다. Kafka의 주요 기능은 다음과 같습니다.

- 레코드 스트림을 게시하고 구독합니다.

- 내결함성이 있는 방식으로 레코드 스트림을 저장합니다.
- 레코드 스트림을 처리합니다.

즉시 사용할 수 있는 Confluorent Platform에는 스키마 레지스트리, REST 프록시, 총 100개 이상의 사전 구축된 Kafka 커넥터 및 ksqlDB도 포함되어 있습니다.

**Confluent** 플랫폼 엔터프라이즈 기능 개요

- * Confluent Control Center. * Kafka 관리 및 모니터링을 위한 UI 기반 시스템. Kafka Connect를 쉽게 관리하고 다른 시스템에 대한 연결을 생성, 편집 및 관리할 수 있습니다.
- Kubernetes를 위한 * Contfluent. * Kubernetes를 위한 Confluent는 Kubernetes 운영자입니다. Kubernetes 운영자는 특정 플랫폼 애플리케이션에 대한 고유한 기능과 요구 사항을 제공하여 Kubernetes의 오케스트레이션 기능을 확장합니다. Confluent Platform의 경우, Kubernetes에서 Kafka의 구축 프로세스를 크게 간소화하고 일반적인 인프라 라이프사이클 작업을 자동화할 수 있습니다.
- * Kafka 연결 커넥터 * 커넥터는 Kafka Connect API를 사용하여 Kafka를 데이터베이스, 키 값 저장소, 검색 인덱스 및 파일 시스템 등의 다른 시스템에 연결합니다. Confluorent Hub에는 가장 널리 사용되는 데이터 소스 및 싱크에 대한 다운로드 가능한 커넥터가 있습니다. 여기에는 Confluorent Platform이 포함된 이러한 커넥터의 전체 테스트 및 지원 버전이 포함됩니다. 자세한 내용은 을 참조하십시오 "여기".
- * 자체 밸런싱 클러스터 * 는 자동화된 로드 밸런싱, 장애 감지 및 자동 복구를 제공합니다. 또한 수동 조정 없이 필요에 따라 브로커를 추가하거나 해체할 수 있도록 지원합니다.
- * 연결 클러스터. * 직접 클러스터를 연결하고 링크 브리지를 통해 클러스터 간에 주제를 미러링합니다. 클러스터 링크를 사용하면 멀티 데이터 센터, 멀티 클러스터, 하이브리드 클라우드 구축을 간편하게 설정할 수 있습니다.
- * Confluorent auto data balancer. * 브로커 수, 파티션 크기, 파티션 수 및 클러스터 내의 리더 수에 대한 클러스터를 모니터링합니다. 균형 조정을 통해 트래픽을 재조정함으로써 운영 워크로드에 미치는 영향을 최소화하면서 클러스터 전체에서 짝수 워크로드를 생성할 수 있습니다.
- * Confluent Replicator. * 여러 데이터 센터에서 여러 Kafka 클러스터를 훨씬 쉽게 유지 관리할 수 있습니다.
- * 계층형 스토리지. * 즐겨 사용하는 클라우드 공급자를 사용하여 대량의 Kafka 데이터를 저장할 수 있는 옵션을 제공하므로 운영 부담과 비용이 줄어듭니다. 계층형 스토리지를 사용하면 비용 효율적인 오브젝트 스토리지에 데이터를 보관하고 더 많은 컴퓨팅 리소스가 필요할 때만 브로커를 확장할 수 있습니다.
- * Confluent JMS 클라이언트. * Confluent Platform에는 Kafka용 JMS 호환 클라이언트가 포함되어 있습니다. 이 Kafka 클라이언트는 Kafka 브로커를 백엔드로 사용하여 JMS 1.1 표준 API를 구현합니다. JMS를 사용하는 레거시 애플리케이션이 있고 기존 JMS 메시지 브로커를 Kafka로 교체하려는 경우 유용합니다.
- * Confluent MQTT proxy. * 중간에 MQTT 브로커가 없어도 MQTT 장치 및 게이트웨이에서 Kafka에 직접 데이터를 게시할 수 있는 방법을 제공합니다.
- * Confluent 보안 플러그인 * Confluent 보안 플러그인은 다양한 Confluent 플랫폼 도구 및 제품에 보안 기능을 추가하는 데 사용됩니다. 현재 Confluent REST 프록시에 사용할 수 있는 플러그인이 있어 수신 요청을 인증하고 인증된 보안 주체를 Kafka에 요청에 전파할 수 있습니다. 이렇게 하면 Confluent REST 프록시 클라이언트가 Kafka 브로커의 멀티테넌트 보안 기능을 활용할 수 있습니다.

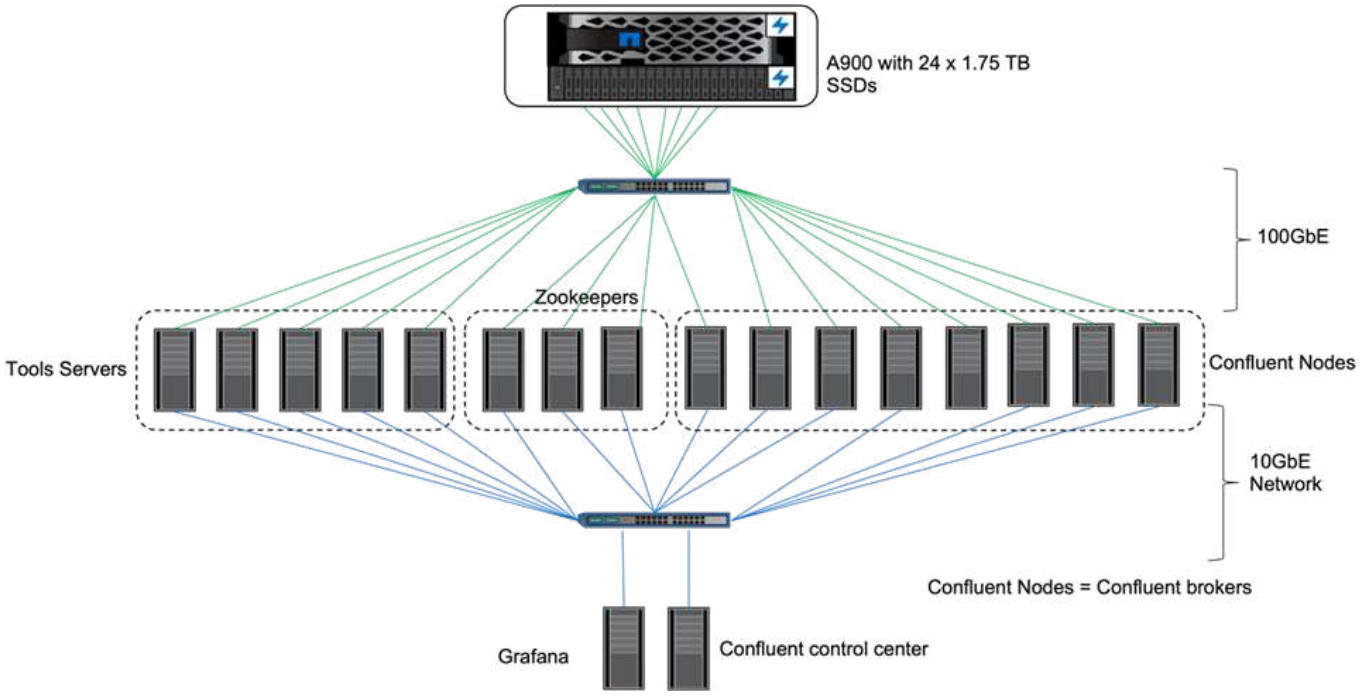
## **Confluent** 성능 검증

NetApp ONTAP의 계층형 스토리지를 위한 Confluent Platform을 사용하여 검증을 수행했습니다. NetApp과 Confluent 팀은 이 검증을 함께 수행하여 IT에 필요한 테스트 사례를 실행했습니다.

**Confluent** 설정

설치를 위해 256GB RAM과 16개의 CPU를 갖춘 3대의 zookeepers, 5개의 브로커, 5대의 테스트 서버를 사용했습니다. NetApp 스토리지의 경우 ONTAP와 AFF A900 HA 쌍을 사용했습니다. 스토리지와 브로커는 100GbE 연결을 통해 연결되었습니다.

다음 그림에서는 계층형 스토리지 검증에 사용된 구성의 네트워크 토폴로지를 보여 줍니다.




도구 서버는 Confluent 노드로 이벤트를 보내거나 받는 응용 프로그램 클라이언트로 작동합니다.

**Confluent** 계층형 스토리지 구성

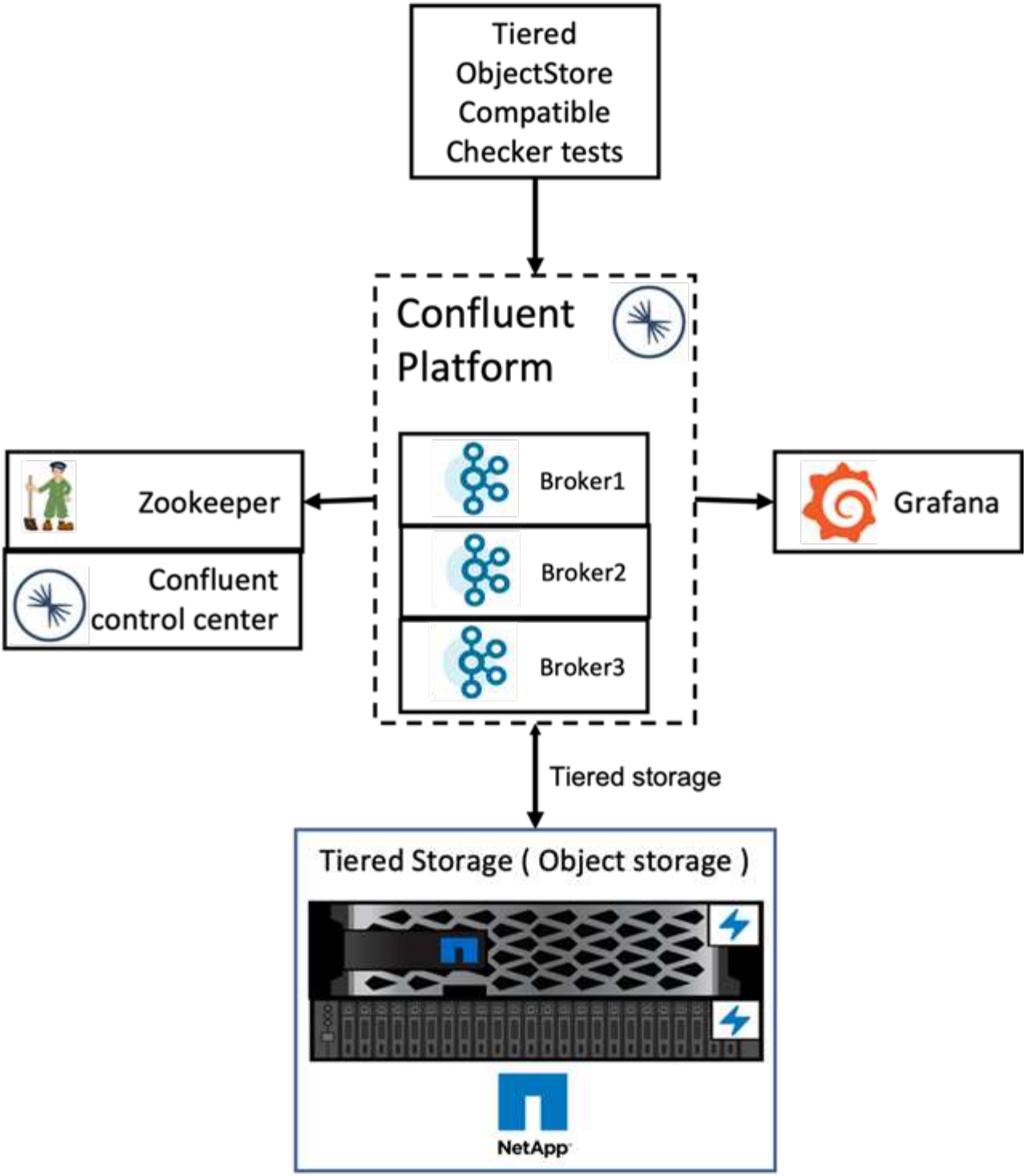
다음과 같은 테스트 매개 변수를 사용했습니다.

```
confluent.tier.fetcher.num.threads=80
confluent.tier.archiver.num.threads=80
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkabucket1-1
confluent.tier.s3.region=us-east-1
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://wle-mendocino-07-08/
confluent.tier.s3.force.path.style.access=true
bootstrap.server=192.168.150.172:9092,192.168.150.120:9092,192.168.150.164
:9092,192.168.150.198:9092,192.168.150.109:9092,192.168.150.165:9092,192.1
68.150.119:9092,192.168.150.133:9092
debug=true
jmx.port=7203
num.partitions=80
num.records=200000000
#object PUT size - 512MB and fetch 100MB - netapp
segment.bytes=536870912
max.partition.fetch.bytes=1048576000
#GET size is max.partition.fetch.bytes/num.partitions
length.key.value=2048
trogdor.agent.nodes=node0,node1,node2,node3,node4
trogdor.coordinator.hostname.port=192.168.150.155:8889
num.producers=20
num.head.consumers=20
num.tail.consumers=1
test.binary.task.max.heap.size=32G
test.binary.task.timeout.sec=3600
producer.timeout.sec=3600
consumer.timeout.sec=3600
```

검증을 위해 HTTP 프로토콜과 함께 ONTAP를 사용했지만 HTTPS도 작동했습니다. 액세스 키와 비밀 키는 confluent.tier.s3.cred.file.path 매개 변수에 제공된 파일 이름에 저장됩니다.

**NetApp 스토리지 컨트롤러 – ONTAP**

검증을 위해 ONTAP에서 단일 HA 쌍 구성을 구성했습니다.




확인 결과

검증을 위해 다음 5가지 테스트 사례를 완료했습니다. 첫 번째 두 테스트는 기능 테스트이고 나머지 세 가지는 성능 테스트입니다.

#### 객체 저장소 정확도 테스트

이 테스트는 API 호출을 사용하여 계층형 스토리지에 사용되는 객체 저장소에서 GET, PUT 및 DELETE 등의 기본 작업을 수행합니다.

#### 계층화 기능 정확도 테스트

이 테스트에서는 오브젝트 스토리지의 엔드 투 엔드 기능을 검사합니다. 토픽을 생성하고, 새로 생성된 토픽에 대한 이벤트 스트림을 생성하고, 브로커가 세그먼트를 오브젝트 스토리지에 아카이브하고, 이벤트 스트림을 소비하며, 소비된 스트림이 생성된 스트림과 일치하는지 확인합니다. 객체 저장소 결함 주입 여부와 관계없이 이 테스트를 수행했습니다. ONTAP의 노드 중 하나에서 서비스 관리자 서비스를 중지하고 엔드 투 엔드 기능이 오브젝트 스토리지에서 작동하는지 확인하여 노드 장애를 시뮬레이션했습니다.

#### 계층 가져오기 벤치마크

이 테스트에서는 계층형 오브젝트 스토리지의 읽기 성능을 검증하고 벤치마크에 의해 생성된 세그먼트에서 과부하된 읽기 요청 범위를 검사했습니다. 이 벤치마크에서 Confluent는 계층 가져오기 요청을 처리하기 위해 사용자 지정 클라이언트를 개발했습니다.

#### 생산 - 워크로드 생성기 사용

이 테스트에서는 세그먼트 아카이브를 통해 오브젝트 저장소에서 쓰기 워크로드를 간접적으로 생성합니다. 소비자 그룹이 세그먼트를 가져올 때 객체 스토리지에서 읽기 워크로드(세그먼트 읽기)가 생성되었습니다. 이 워크로드는 TOCC 스크립트에 의해 생성되었습니다. 이 테스트에서는 오브젝트 저장소에서 병렬 스레드의 읽기 및 쓰기 성능을 확인했습니다. 계층화 기능 정확도 테스트에서 보았듯이, 객체 저장소 결함 주입을 사용하여 테스트했으며 포함하지 않았습니다.

#### 보존 워크로드 생성기

이 테스트에서는 무거운 주제 보존 워크로드에서 오브젝트 스토리지의 삭제 성능을 검사했습니다. 보존 워크로드는 테스트 주제와 동시에 많은 메시지를 생성하는 TOCC 스크립트를 사용하여 생성되었습니다. 테스트 주제는 공격적인 크기 기반 및 시간 기반 보존 설정으로 구성되었으며, 이로 인해 이벤트 스트림이 개체 저장소에서 지속적으로 제거됩니다. 그런 다음 세그먼트가 아카이브되었습니다. 이로 인해 오브젝트 저장소 삭제 작업의 브로커링 및 컬렉션에 의해 오브젝트 스토리지가 많이 삭제되었습니다.

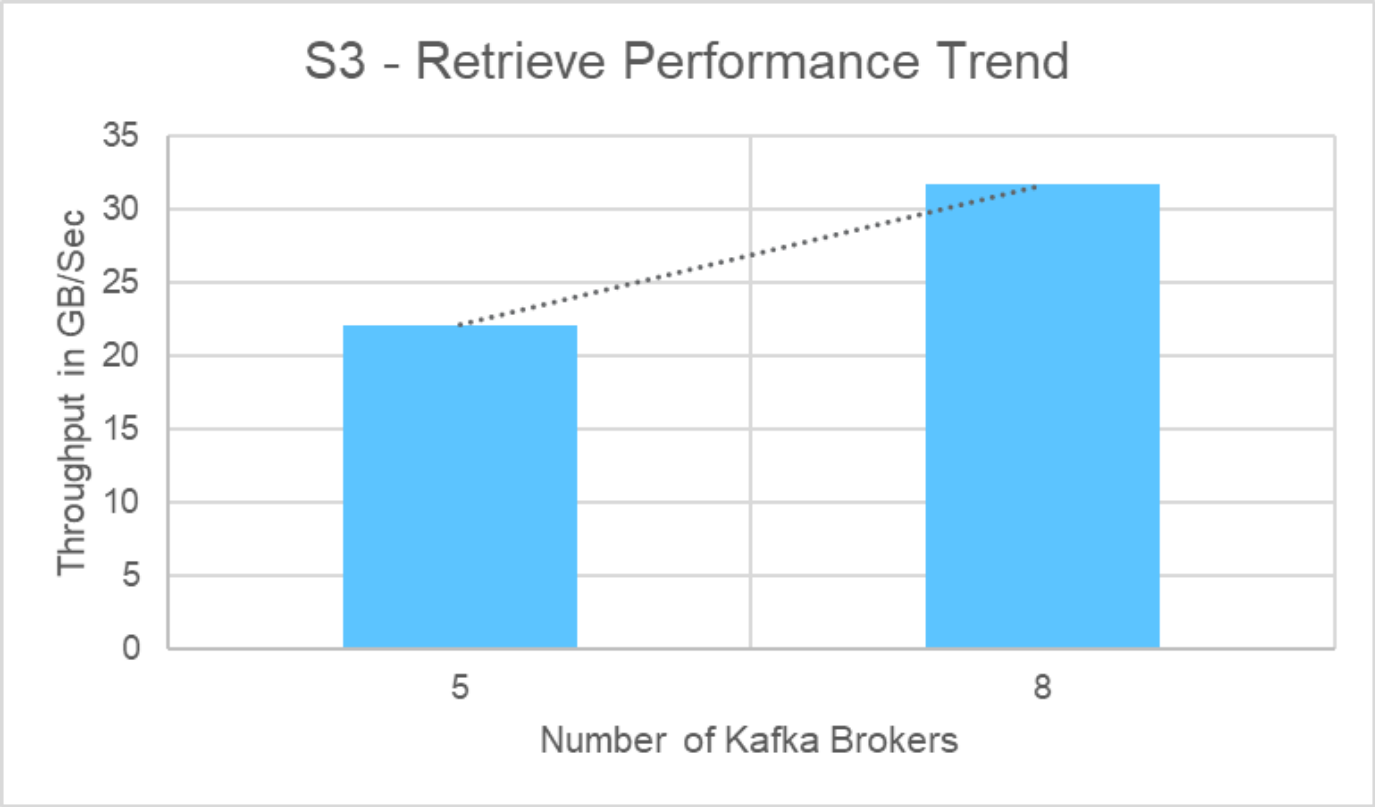
검증 세부 정보는 를 참조하십시오 "유창하게" 웹 사이트.

## 농작물 사용 워크로드 생성기를 사용한 성능 테스트

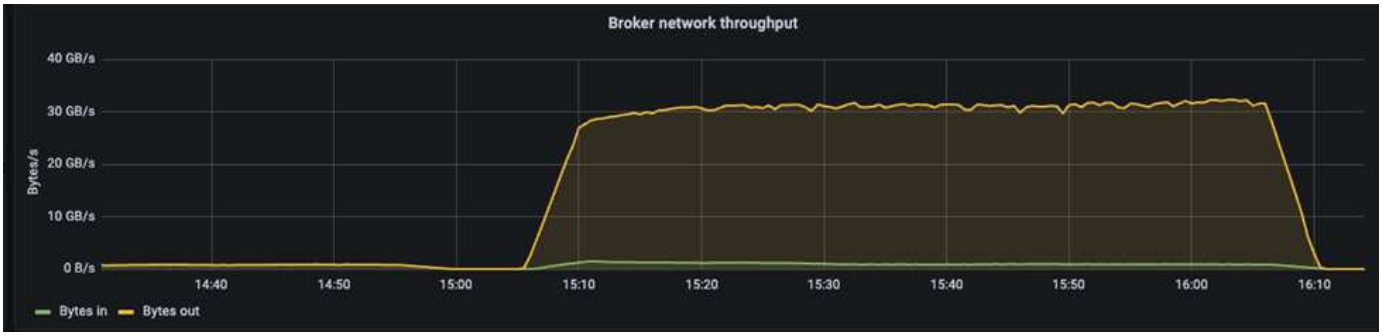
AFF A900 HA 쌍 NetApp 스토리지 컨트롤러 1개를 사용하여 농작물 사용 워크로드 중 5개 또는 8개의 브로커 노드로 계층형 스토리지 테스트를 수행했습니다. 테스트에 따르면 AFF A900 리소스 활용률이 100%에 이를 때까지 브로커 노드의 수에 따라 완료 시간과 성능 결과가 확장되었습니다. ONTAP 스토리지 컨트롤러를 설치하려면 최소 하나의 HA 쌍이 필요했습니다.

S3 검색 작업에 대한 성능은 Confluent 브로커 노드의 수에 따라 선형으로 향상되었습니다. ONTAP 스토리지 컨트롤러는 단일 배포에서 최대 12개의 HA 쌍을 지원합니다.

다음 그래프에는 브로커 노드 5개 또는 8개가 결합된 S3 계층화 트래픽이 나와 있습니다. AFF A900 단일 HA Pair 성능을 극대화했습니다.




다음 그래프는 약 31.74GBps의 Kafka 처리량을 보여 줍니다.




또한 ONTAP 스토리지 컨트롤러 'perfstat' 보고서에서 유사한 처리량도 관찰되었습니다.

```
object_store_server:wle-mendocino-07-08:get_data:34080805907b/ s
object_store_server:wle-mendocino-07-08:put_data:484236974b/ s
```

## 성능 모범 사례 지침

### 이 페이지에서는 이 솔루션의 성능을 개선하기 위한 모범 사례를 설명합니다.

- ONTAP의 경우 가능하면 GET SIZE >= 1MB 를 사용합니다.
- 브로커 노드의 erver.properties` 에서 num.network.threads` 및 num.io.threads를 증가시키면 계층화 작업이 S3 계층으로 증가하게 됩니다. 이러한 결과는 num.network.threads` 및 num.io.threads가 32로 설정된 것입니다.
- S3 버킷은 구성원 애그리게이트당 8개의 구성요소를 목표로 해야 합니다.

- S3 트래픽을 구동하는 이더넷 링크는 스토리지와 클라이언트 모두에서 가능하면 MTU 9k 를 사용해야 합니다.

### 결론

이 검증 테스트는 NetApp ONTAP 스토리지 컨트롤러에 유창한 confluent에서 31.74GBps의 계층화 처리량에 도달했습니다.

#### 추가 정보를 찾을 수 있는 위치

이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹 사이트를 검토하십시오.

- Confluent란 무엇입니까?

  "https://www.confluent.io/apache-kafka-vs-confluent/"

- S3 싱크 매개 변수 세부 정보

  "https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options"

- 아파치 카프카

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

- S3 in ONTAP 모범 사례

  https://www.netapp.com/pdf.html?item=/media/17219-tr4814.pdf

- S3 오브젝트 스토리지 관리

  "https://docs.netapp.com/us-en/ontap/s3-config/s3-support-concept.html"

- NetApp 제품 설명서

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

# Apache Spark용 NetApp 스토리지 솔루션

## TR-4570: Apache Spark용 NetApp 스토리지 솔루션: 아키텍처, 사용 사례 및 성능 결과

Rick Huang, Karthikeyan Nagalingam, NetApp

이 문서에서는 Apache Spark 아키텍처, 고객 사용 사례 및 빅데이터 분석 및 인공 지능(AI)과 관련된 NetApp 스토리지 포트폴리오에 대해 중점적으로 소개합니다. 또한 적절한 Spark 솔루션을 선택할 수 있도록 업계 표준 AI, 머신 러닝(ML) 및 딥 러닝(DL) 툴을 일반적인 Hadoop 시스템과 비교하여 다양한 테스트 결과를 제공합니다. 시작하려면 Spark 아키텍처, 적절한 구성 요소 및 두 가지 배포 모드(클러스터 및 클라이언트)가 필요합니다.

또한 이 문서는 구성 문제를 해결하기 위한 고객 사용 사례를 제공하며 빅데이터 분석 및 AI, ML, Spark를 지원하는 DL과 관련된 NetApp 스토리지 포트폴리오의 개요를 설명합니다. 그런 다음 Spark 관련 사용 사례 및 NetApp Spark

솔루션 포트폴리오의 테스트 결과를 마무리합니다.

## 고객의 당면 과제

이 섹션에서는 소매, 디지털 마케팅, 은행, 이산 제조, 프로세스 제조 등 데이터 성장 산업에서 빅데이터 분석 및 AI/ML/DL과 관련된 고객 과제에 대해 정부 및 전문 서비스.

### 성능을 예측할 수 없습니다

기존 Hadoop 구축에는 일반적으로 일반 하드웨어가 사용됩니다. 성능을 향상시키려면 네트워크, 운영 체제, Hadoop 클러스터, Spark와 같은 에코시스템 구성 요소 및 하드웨어를 조정해야 합니다. 각 계층을 튜닝하더라도 Hadoop은 사용자 환경에서 고성능을 발휘하도록 설계되지 않은 상용 하드웨어에서 실행되기 때문에 원하는 성능 수준을 달성하는 것이 어려울 수 있습니다.

### 미디어 및 노드 장애

정상적인 조건에서 상용 하드웨어는 장애가 발생하기 쉽습니다. 데이터 노드의 한 디스크에 장애가 발생하면 기본적으로 Hadoop 마스터는 해당 노드가 정상 상태가 아닌 것으로 간주합니다. 그런 다음 네트워크를 통해 해당 노드의 특정 데이터를 복제본에서 정상 노드로 복제합니다. 이 프로세스는 모든 Hadoop 작업에 대한 네트워크 패킷의 속도를 늦춥니다. 그런 다음, 정상적인 상태가 아닌 노드가 정상 상태가 될 때 클러스터가 데이터를 다시 복제하고 초과 복제된 데이터를 제거해야 합니다.

### **Hadoop** 공급업체에 종속

Hadoop 총판은 자체 버전 관리 기능을 통해 Hadoop을 직접 배포하며, 고객은 이 버전을 통해 해당 배포판에 종속됩니다. 그러나 많은 고객들은 특정 Hadoop 배포와 고객을 연계하지 않는 인메모리 분석에 대한 지원을 필요로 합니다. 따라서 고객이 원하는 대로 배포를 변경하고 분석을 실행할 수 있어야 합니다.

### 둘 이상의 언어를 지원하지 않습니다

고객은 업무를 실행하기 위해 MapReduce Java 프로그램 외에 여러 언어에 대한 지원을 필요로 하는 경우가 많습니다. SQL 및 스크립트와 같은 옵션을 사용하면 답변을 보다 유연하게 얻고, 데이터를 구성 및 검색하는 더 많은 옵션을 사용할 수 있으며, 데이터를 분석 프레임워크로 신속하게 이동할 수 있습니다.

### 사용의 어려움

Hadoop을 사용하기 어렵다고 불만을 토로하는 사람들이 있습니다. 새로운 버전이 출시될 때마다 Hadoop이 더욱 단순해지고 강력해지기는 했지만 이러한 비판은 지속되었습니다. Hadoop을 사용하려면 Java 및 MapReduce 프로그래밍 패턴을 이해해야 합니다. 이는 데이터베이스 관리자와 기존 스크립팅 기술을 사용하는 사람들에게 어려운 과제입니다.

### 복잡한 프레임워크 및 도구

엔터프라이즈 AI 팀은 여러 가지 과제에 직면합니다. 전문 데이터 과학 지식을 갖추고 있더라도 다양한 구축 에코시스템과 애플리케이션을 위한 툴과 프레임워크에서 다른 에코시스템으로 변환되지 않을 수 있습니다. 데이터 과학 플랫폼은 Spark 기반의 해당 빅 데이터 플랫폼과 원활하게 통합되어야 하며, 간편한 데이터 이동, 재사용 가능 모델, 즉시 사용 가능한 코드, 프로토타입 제작, 검증, 버전 관리, 공유, 재사용, 운영 환경에 모델을 빠르게 구축할 수 있습니다.

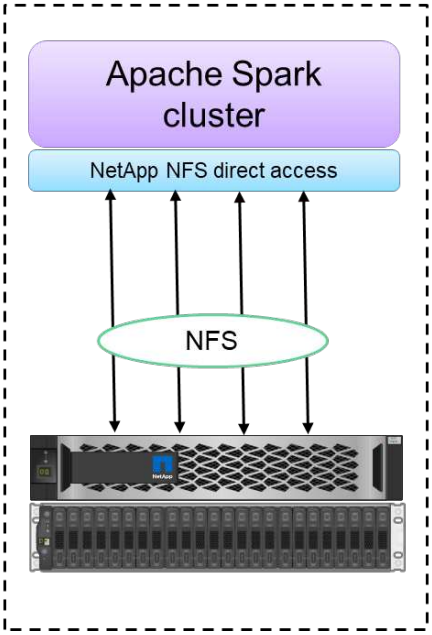
## **NetApp**을 선택해야 하는 이유

NetApp은 다음과 같은 방법으로 Spark 경험을 개선할 수 있습니다.

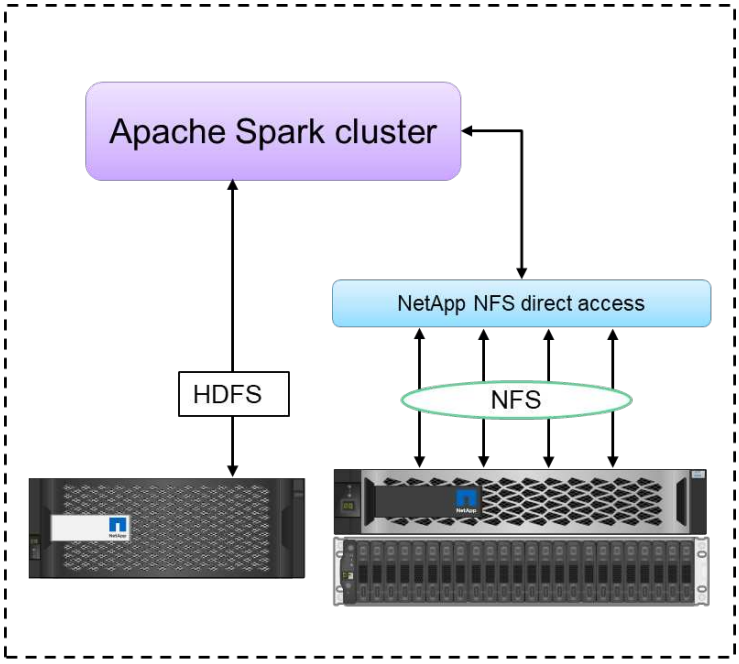
- NetApp NFS 직접 액세스(아래 그림에 표시)를 사용하면 데이터를 이동하거나 복사하지 않고도 기존 또는 새로운

NFSv3 또는 NFSv4 데이터에 대해 빅데이터 분석 작업을 실행할 수 있습니다. 여러 데이터 복제본을 방지하고 소스와 데이터를 동기화할 필요가 없습니다.

- 스토리지 효율성 향상 및 서버 복제 감소 예를 들어, NetApp E-Series Hadoop 솔루션은 3개의 데이터 복제본이 아닌 2개의 복제본을 필요로 하며 FAS Hadoop 솔루션은 데이터 소스만 필요로 하지만 데이터의 복제 또는 복제본은 필요로 하지 않습니다. NetApp 스토리지 솔루션은 또한 서버 간 트래픽을 줄여 줍니다.
- 드라이브 및 노드 장애 시 Hadoop 작업 및 클러스터 동작이 향상됩니다.
- 더 나은 데이터 수집 성능:




Configuration 1: NFS as primary storage




Configuration 2: HDFS and NFS in single Spark cluster

예를 들어, 금융 및 의료 부문에서 데이터를 한 위치에서 다른 위치로 이동하는 것은 쉬운 일이 아닌 법적 의무를 준수해야 합니다. 이 시나리오에서는 NetApp NFS 직접 액세스가 원래의 위치에서 재무 및 의료 데이터를 분석합니다. 또 다른 주요 이점은 NetApp NFS 직접 액세스를 통해 기본 Hadoop 명령을 사용하여 Hadoop 데이터를 간편하게 보호하고 NetApp의 강력한 데이터 관리 포트폴리오를 통해 데이터 보호 워크플로우를 구축할 수 있다는 것입니다.

NetApp NFS 직접 액세스는 Hadoop/Spark 클러스터에 대한 두 가지 유형의 구축 옵션을 제공합니다.

- 기본적으로 Hadoop 또는 Spark 클러스터는 데이터 스토리지와 기본 파일 시스템에 HDFS(Hadoop Distributed File System)를 사용합니다. NetApp NFS 직접 액세스는 기본 HDFS를 NFS 스토리지로 대체하여 NFS 데이터에 대한 직접 분석을 지원합니다.
- 다른 구축 옵션에서 NetApp NFS 직접 액세스는 NFS를 단일 Hadoop 또는 Spark 클러스터의 HDFS와 함께 추가 스토리지로 구성할 수 있도록 지원합니다. 이 경우 고객은 NFS 내보내기를 통해 데이터를 공유하고 HDFS 데이터와 함께 동일한 클러스터에서 데이터를 액세스할 수 있습니다.

NetApp NFS 직접 액세스를 사용할 때의 주요 이점은 다음과 같습니다.

- 현재 위치에서 데이터를 분석하면 분석 데이터를 HDFS와 같은 Hadoop 인프라스트럭처로 이동하는 데 시간과 성능 소모가 큰 작업이 발생하지 않습니다.
- 복제본 수를 3개부터 1개로 축소
- 사용자가 컴퓨팅과 스토리지를 분리하여 독립적으로 확장할 수 있도록 지원

- ONTAP의 강력한 데이터 관리 기능을 활용하여 엔터프라이즈 데이터 보호 제공
- Hortonworks 데이터 플랫폼을 사용한 인증
- 하이브리드 데이터 분석을 구축할 수 있습니다.
- 동적 다중 스레드 기능을 활용하여 백업 시간 단축

을 참조하십시오 "TR-4657: NetApp 하이브리드 클라우드 데이터 솔루션 - 고객 사용 사례를 기반으로 Spark 및 Hadoop" Hadoop 데이터, 클라우드에서 사내로 백업 및 재해 복구, 기존 Hadoop 데이터에 대한 DevTest, 데이터 보호 및 멀티 클라우드 연결, 분석 워크로드 가속화 등을 지원합니다.

다음 섹션에서는 Spark 고객에게 중요한 스토리지 기능에 대해 설명합니다.

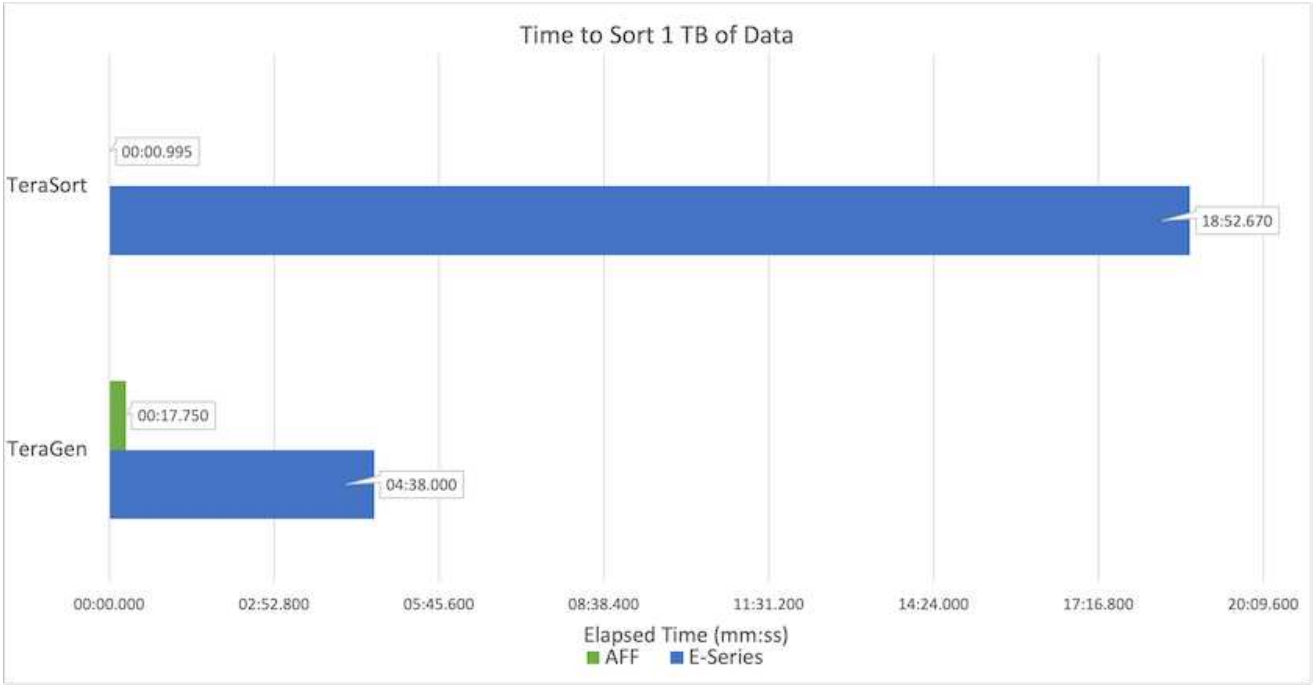
#### 스토리지 계층화

Hadoop 스토리지 계층화를 사용하면 스토리지 정책에 따라 다양한 스토리지 유형의 파일을 저장할 수 있습니다. 스토리지 유형으로는 핫, 콜드, 웜, 올 SSD, 원 SSD 등이 있습니다. 그리고 게으른 유영이 있습니다.

<<<<<<< head는 NetApp AFF 스토리지 컨트롤러에서 Hadoop 스토리지 계층화를 검증하고 SSD 및 SAS 드라이브를 사용하는 E-Series 스토리지 컨트롤러를 다양한 스토리지 정책으로 수행했습니다. AFF-A800의 Spark 클러스터에는 4개의 컴퓨팅 작업자 노드가 있는 반면 E-Series를 사용하는 클러스터는 8개의 노드를 가지고 있습니다. 주로 SSD(Solid-State Drive)와 HDD(하드 드라이브 디스크)의 성능을 비교합니다.

NetApp AFF 스토리지 컨트롤러에서 Hadoop 스토리지 계층화를 검증하고 SSD 및 SAS 드라이브를 사용하는 E-Series 스토리지 컨트롤러를 다양한 스토리지 정책으로 수행했습니다. AFF-A800의 Spark 클러스터에는 4개의 컴퓨팅 작업자 노드가 있는 반면 E-Series를 사용하는 클러스터는 8개의 노드를 가지고 있습니다. 주로 SSD와 하드 드라이브 디스크의 성능을 비교하기 위해 이 작업을 수행하였습니다. >>>>>> a51c9ddf73ca69e1120ce05edc7b0b9607b96eae

다음 그림은 Hadoop SSD를 위한 NetApp 솔루션의 성능을 보여줍니다.




- 기본 NL-SAS 구성에는 8개의 컴퓨팅 노드와 96개의 NL-SAS 드라이브가 사용되었습니다. 이 구성에서는 4분 38초 내에 1TB의 데이터가 생성되었습니다. 을 참조하십시오 "TR-3969 Hadoop용 NetApp E-Series 솔루션" 클러스터 및 스토리지 구성에 대한 자세한 내용은 를 참조하십시오.
- TeraGen을 사용하면 SSD 구성에서 NL-SAS 구성보다 1TB의 데이터가 15.66x 더 빠르게 생성됩니다. 또한 SSD 구성에서는 컴퓨팅 노드 수의 절반과 디스크 드라이브 수의 절반을 사용했습니다(총 24개의 SSD 드라이브). 작업 완료 시간을 기준으로 할 때 NL-SAS 구성의 속도는 약 2배였습니다.
- TeraSort를 사용하면 SSD 구성에서 NL-SAS 구성에 비해 1TB의 데이터를 1138.36배 더 빠르게 정렬할 수 있습니다. 또한 SSD 구성에서는 컴퓨팅 노드 수의 절반과 디스크 드라이브 수의 절반을 사용했습니다(총 24개의 SSD 드라이브). 따라서 드라이브당 NL-SAS 구성보다 약 3배 빠른 속도를 제공합니다. <<<<<<< 머리
- 여기서 주목할 점은 회전식 디스크에서 All-Flash로 전환하여 성능을 향상할 수 있다는 것입니다. 컴퓨팅 노드의 수는 병목 현상이 아니었습니다. NetApp의 All-Flash 스토리지를 사용하면 런타임 성능이 원활하게 확장됩니다.
- NFS를 사용하면 데이터가 모두 함께 풀링되는 것과 기능적으로 동일하므로 워크로드에 따라 컴퓨팅 노드의 수를 줄일 수 있습니다. Apache Spark 클러스터 사용자는 컴퓨팅 노드의 수를 변경할 때 데이터를 수동으로 재조정할 필요가 없습니다.

- 즉, 회전식 디스크에서 All-Flash로 전환하면 성능이 향상됩니다. 컴퓨팅 노드의 수는 병목 현상이 아니었습니다. NetApp All-Flash 스토리지를 사용하면 런타임 성능이 원활하게 확장됩니다.
- NFS를 사용하면 데이터가 모두 함께 풀링되는 것과 기능적으로 동일하므로 워크로드에 따라 컴퓨팅 노드의 수를

줄일 수 있습니다. Apache Spark 클러스터 사용자는 컴퓨팅 노드 수를 변경할 때 데이터를 수동으로 재조정할 필요가 없습니다. >>>>>> a51c9ddf73ca69e1120ce05edc7b0b9607b96eae

#### 성능 확장 - 스케일아웃

AFF 솔루션에서 Hadoop 클러스터의 컴퓨팅 성능이 더 많이 필요한 경우 적절한 수의 스토리지 컨트롤러를 사용하여 데이터 노드를 추가할 수 있습니다. 스토리지 컨트롤러 어레이당 4개의 데이터 노드로 시작하고 워크로드 특성에 따라 스토리지 컨트롤러당 8개의 데이터 노드로 숫자를 늘리는 것이 좋습니다.

AFF와 FAS는 데이터 이동 없는 분석에 적합합니다. 계산 요구사항에 따라 노드 관리자를 추가할 수 있으며 무중단 운영을 통해 다운타임 없이 스토리지 컨트롤러를 온디맨드 방식으로 추가할 수 있습니다. NetApp은 AFF NVMe 미디어 지원, 효율성 보장, 데이터 축소, QoS, 예측 분석, FAS 클라우드 계층화, 복제, 클라우드 구축, 보안 고객의 요구사항을 충족할 수 있도록 NetApp에서는 추가 라이센스 비용 없이 파일 시스템 분석, 할당량, 온박스 로드 밸런싱과 같은 기능을 제공합니다. NetApp은 동시 작업 수, 낮은 지연 시간, 단순한 운영, 경쟁업체보다 더 높은 초당 처리 성능 등의 이점을 제공합니다. 또한 NetApp Cloud Volumes ONTAP은 3가지 주요 클라우드 공급자 모두에서 실행됩니다.

#### 성능 확장 - 스케일업

스케일업 기능을 사용하면 스토리지 용량이 더 필요할 때 AFF, FAS, E-Series 시스템에 디스크 드라이브를 추가할 수 있습니다. Cloud Volumes ONTAP를 사용하면 자주 사용되지 않는 데이터를 블록 스토리지의 오브젝트 스토리지로 계층화하고, 추가 컴퓨팅 없이 Cloud Volumes ONTAP 라이센스를 스태킹하는 두 가지 요소의 조합으로 스토리지를 PB 수준으로 확장할 수 있습니다.

#### 다중 프로토콜

NetApp 시스템은 SAS, iSCSI, FCP, InfiniBand를 비롯한 대부분의 Hadoop 구현 프로토콜을 및 NFS 로 이동합니다.

#### 운영 및 지원 솔루션

이 문서에 설명된 Hadoop 솔루션은 NetApp에서 지원됩니다. 또한 이러한 솔루션은 주요 Hadoop 총판에서도 인증되었습니다. 자세한 내용은 를 참조하십시오 "MapR" 사이트 "Hortonworks의" 사이트 및 Cloudera "인증" 및 "파트너" 있습니다.

## 대상

### 분석 및 데이터 과학의 세계는 IT와 비즈니스의 여러 분야를 아우릅니다.

- 데이터 과학자는 자신이 선택한 도구와 라이브러리를 사용할 수 있는 유연성이 필요합니다.
- 데이터 엔지니어는 데이터 흐름과 데이터 위치를 알아야 합니다.
- DevOps 엔지니어는 새로운 AI 및 ML 애플리케이션을 CI 및 CD 파이프라인에 통합하는 툴을 필요로 합니다.
- 클라우드 관리자와 설계자는 하이브리드 클라우드 리소스를 설정하고 관리할 수 있어야 합니다.
- 비즈니스 사용자는 분석, AI, ML 및 DL 애플리케이션에 액세스해야 합니다.

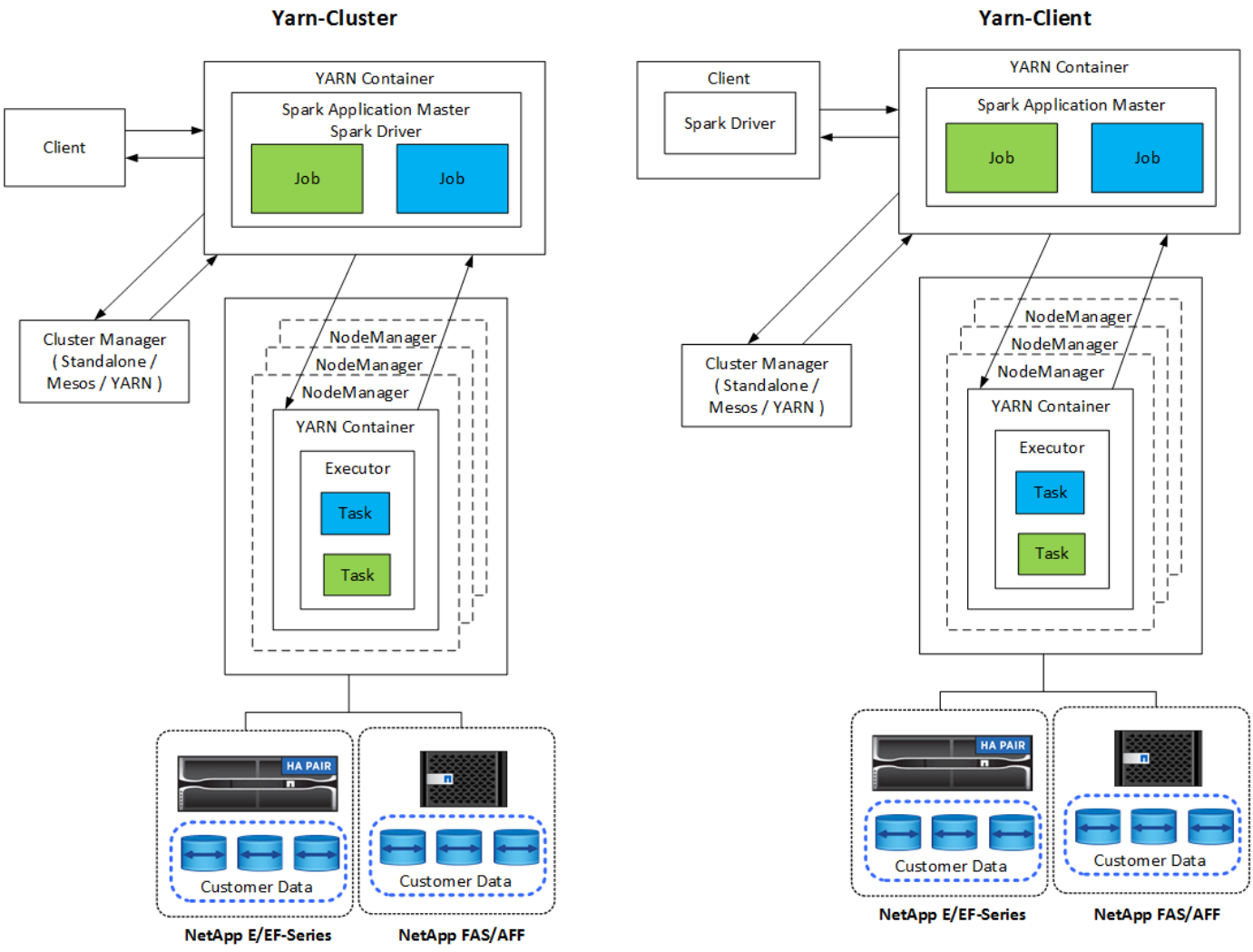
이 기술 보고서에서는 NetApp AFF, E-Series, StorageGRID, NFS 직접 액세스, Apache Spark가 어떤 식으로 Horovod와 Keras는 이러한 각 역할이 비즈니스에 가치를 제공하는 데 도움을 줍니다.

## 솔루션 기술

### Apache Spark는 HDFS(Hadoop Distributed File System)와 직접 연동되는 Hadoop 애플리케이션을 작성하기 위한 인기 있는 프로그래밍 프레임워크입니다. Spark는 프로덕션을

지원하며 스트리밍 데이터의 처리를 지원하며 MapReduce보다 빠릅니다. Spark는 효율적인 반복을 위해 메모리 내 데이터 캐싱을 구성할 수 있으며 Spark 셸은 데이터를 학습하고 탐색하는 대화형 기능을 제공합니다. Spark를 사용하면 Python, Scala 또는 Java로 애플리케이션을 만들 수 있습니다. SPARK 응용 프로그램은 하나 이상의 작업이 있는 하나 이상의 작업으로 구성됩니다.

모든 Spark 응용 프로그램에는 Spark 드라이버가 있습니다. YARN-Client 모드에서 드라이버는 클라이언트에서 로컬로 실행됩니다. YARN-Cluster 모드에서는 드라이버가 애플리케이션 마스터의 클러스터에서 실행됩니다. 클러스터 모드에서는 클라이언트의 연결이 끊기더라도 애플리케이션이 계속 실행됩니다.




세 가지 클러스터 관리자가 있습니다.

- * 독립 실행형. * 이 매니저는 Spark의 일부이며, 클러스터를 쉽게 설정할 수 있습니다.
- * Apache Mesos. * MapReduce 및 기타 애플리케이션도 실행하는 일반 클러스터 관리자입니다.
- * Hadoop YARN. * Hadoop 3의 리소스 관리자입니다.

탄력적인 분산 데이터 세트(RDD)는 Spark의 주요 구성 요소입니다. RDD는 클러스터의 메모리에 저장된 데이터에서 손실되거나 누락된 데이터를 재생성하고 파일에서 나오거나 프로그래밍 방식으로 생성된 초기 데이터를 저장합니다. RDD는 파일, 메모리에 있는 데이터 또는 다른 RDD에서 생성됩니다. SPARK 프로그래밍은 변환과 동작이라는 두 가지 작업을 수행합니다. 변환은 기존 RDD를 기반으로 새 RDD를 생성합니다. 작업은 RDD에서 값을 반환합니다.

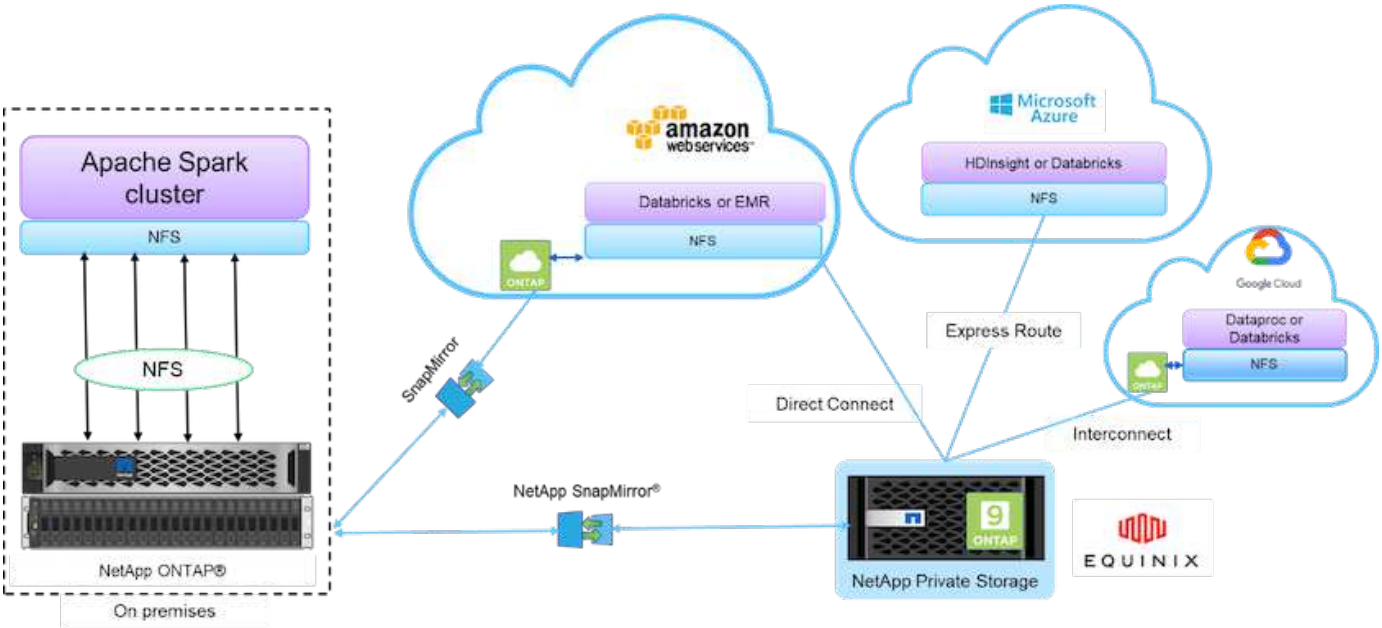
변환 및 작업은 Spark 데이터 집합 및 DataFrames에도 적용됩니다. 데이터 세트는 분산 데이터 모음으로서 Spark SQL의 최적화된 실행 엔진의 이점과 함께 RDD(강력한 타이핑, 람다 함수 사용)의 이점을 제공합니다. 데이터세트는 JVM 객체에서 생성한 다음 기능 변환(맵, 평면 맵, 필터 등)을 사용하여 조작할 수 있습니다. DataFrame 은 명명된 열로 구성된 데이터 집합입니다. 개념적으로 관계형 데이터베이스의 테이블 또는 R/Python의 데이터 프레임에 해당합니다. DataFrames는 정형 데이터 파일, Hive/HBase의 테이블, 사내 또는 클라우드의 외부 데이터베이스 또는 기존 RDD와 같은 다양한 소스에서 구성할 수 있습니다.

스파크 응용 프로그램에는 하나 이상의 스파크 작업이 포함됩니다. 작업은 실행기에서 작업을 실행하고 실행자는 YARN 컨테이너에서 실행됩니다. 각 실행자는 단일 컨테이너에서 실행되며 실행자는 응용 프로그램 수명 내내 존재합니다. 응용 프로그램이 시작된 후 실행자가 수정되고 YARN은 이미 할당된 컨테이너의 크기를 조정하지 않습니다. 집행자는 인메모리 데이터에 대해 작업을 동시에 실행할 수 있습니다.

## NetApp Spark 솔루션 개요

NetApp은 FAS/AFF, E-Series, Cloud Volumes ONTAP의 3가지 스토리지 포트폴리오를 보유하고 있습니다. Apache Spark를 지원하는 Hadoop 솔루션을 위한 ONTAP 스토리지 시스템을 통해 AFF 및 E-Series를 검증했습니다.

NetApp 기반의 Data Fabric은 아래 그림과 같이 데이터 액세스, 제어, 보호 및 보안을 위한 데이터 관리 서비스와 애플리케이션(구성 요소)을 통합합니다.

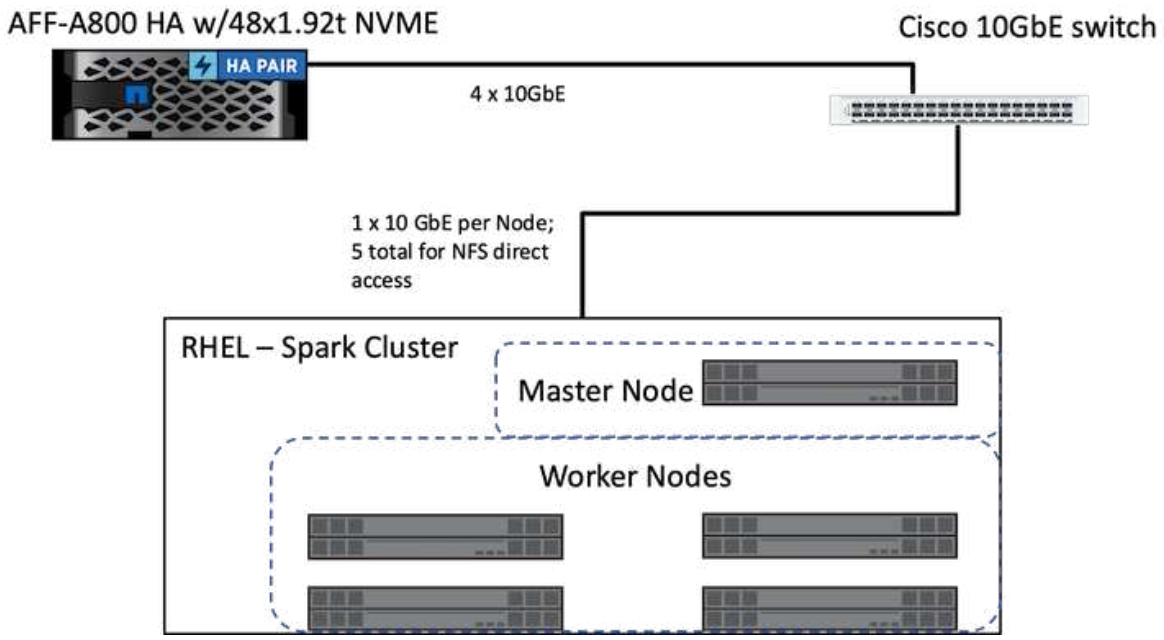



위 그림의 구성 요소는 다음과 같습니다.

- * NetApp NFS 직접 액세스 * 는 추가 소프트웨어 또는 드라이버 요구사항 없이 최신 Hadoop 및 Spark 클러스터를 NetApp NFS 볼륨에 직접 액세스할 수 있도록 지원합니다.
- * NetApp Cloud Volumes ONTAP 및 클라우드 볼륨 서비스. * Microsoft Azure 클라우드 서비스의 AWS(Amazon Web Services) 또는 ANF(Azure NetApp Files)에서 실행되는 ONTAP를 기반으로 하는 소프트웨어 정의 연결형 스토리지.
- * NetApp SnapMirror 기술. * 사내 및 ONTAP 클라우드 또는 NPS 인스턴스 간에 데이터 보호 기능을 제공합니다.
- * 클라우드 서비스 공급자 * 이러한 공급자에는 AWS, Microsoft Azure, Google Cloud 및 IBM Cloud가 포함됩니다.

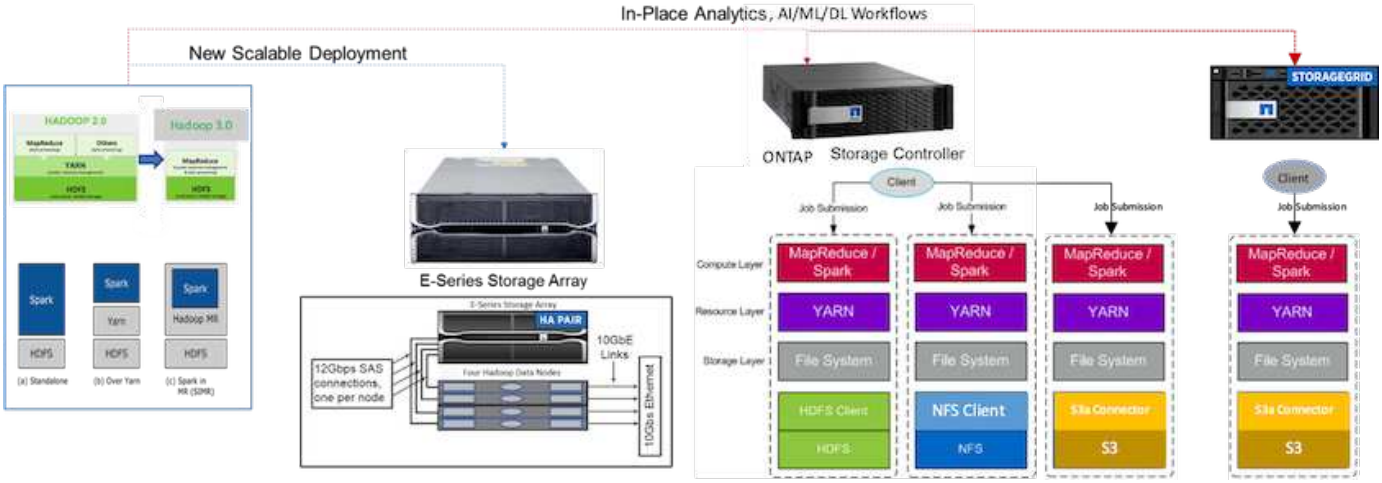
- * PaaS. * AWS의 EMR(Amazon Elastic MapReduce) 및 Databricks와 같은 클라우드 기반 분석 서비스와 Microsoft Azure HDInsight 및 Azure Databricks

다음 그림은 NetApp 스토리지를 포함한 Spark 솔루션을 보여 줍니다.

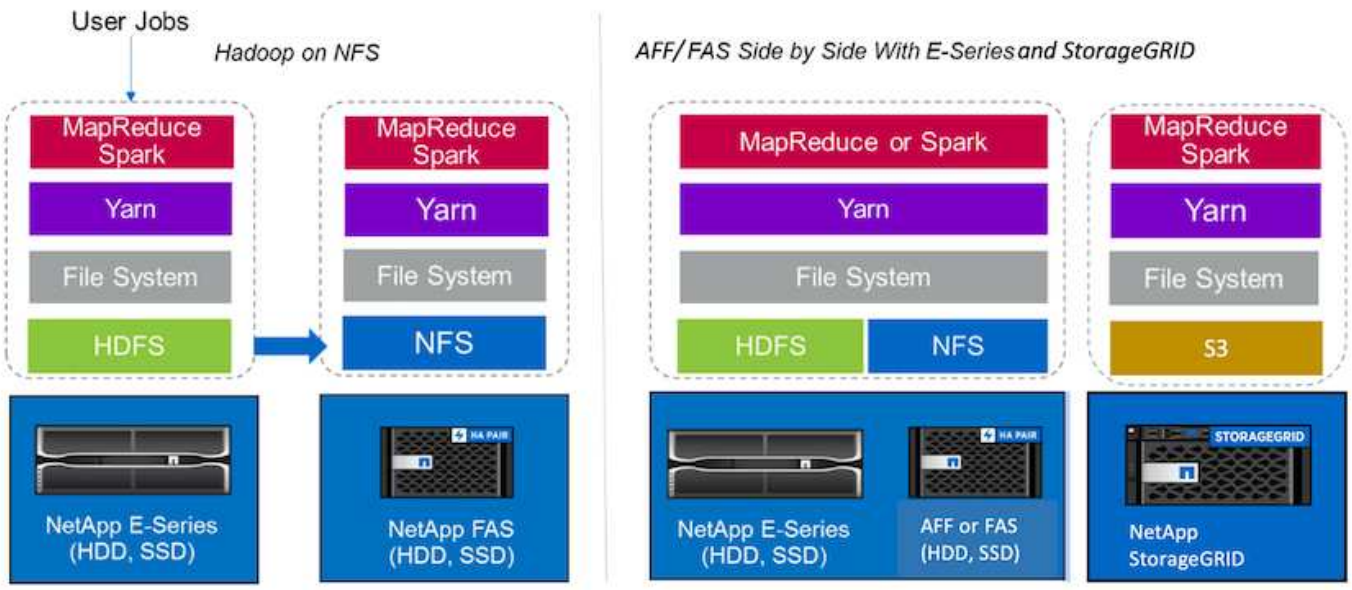



ONTAP Spark 솔루션은 NetApp NFS 직접 액세스 프로토콜을 사용하여 기존 운영 데이터에 대한 액세스를 통해 데이터 이동 없이 분석 및 AI, ML 및 DL 워크플로우를 지원합니다. Hadoop 노드에 사용 가능한 운영 데이터를 내보내 데이터 이동 없이 분석 및 AI, ML, DL 작업을 수행합니다. Hadoop 노드에서 처리하는 데이터에 NetApp NFS 직접 액세스 또는 사용하지 않고 액세스할 수 있습니다. 독립 실행형 또는 'YARN' 클러스터 관리자를 사용하여 Spark에서 ' 을 사용하여 NFS 볼륨을 구성할 수 있습니다<file:///<target_volume>`. 서로 다른 데이터 세트를 사용하여 세 가지 사용 사례를 검증했습니다. 이러한 검증에 대한 자세한 내용은 "테스트 결과" 섹션에 나와 있습니다. (Xref)

다음 그림은 NetApp Apache Spark/Hadoop 스토리지 포지셔닝을 보여줍니다.




E-Series Spark 솔루션, AFF/FAS ONTAP Spark 솔루션 및 StorageGRID Spark 솔루션의 고유한 기능을 식별하고 자세한 검증 및 테스트를 수행했습니다. 관찰 결과에 따라, NetApp은 E-Series 솔루션을 그린필드 설치 및 새로운 확장 가능한 구축에 사용하고, 기존 NFS 데이터를 사용하는 데이터 이동 없는 분석, AI, ML, DL 워크로드, 오브젝트 스토리지가 필요할 경우 StorageGRID for AI, ML, DL 및 최신 데이터 분석을 지원하는 AFF/FAS 솔루션을 권장합니다.




데이터 레이크는 분석, AI, ML 및 DL 작업에 사용할 수 있는 기본 형식의 대규모 데이터 세트를 위한 스토리지 리포지토리입니다. E-Series, AFF/FAS, StorageGRID SG6060 Spark 솔루션을 위한 데이터 레이크 저장소를 구축했습니다. E-Series 시스템은 HDFS에 Hadoop Spark 클러스터에 액세스할 수 있는 반면, 기존 운영 데이터는 NFS 직접 액세스 프로토콜을 통해 Hadoop 클러스터에 액세스할 수 있습니다. 오브젝트 스토리지에 상주하는 데이터 세트의 경우 NetApp StorageGRID를 통해 S3 및 S3a의 안전한 액세스를 제공합니다.

## 사용 사례 요약

### 이 페이지에서는 이 솔루션을 사용할 수 있는 다양한 영역에 대해 설명합니다.

#### 스트리밍 데이터

Apache Spark는 스트리밍 추출, 변환 및 로드(ETL) 프로세스, 데이터 보강, 이벤트 감지 트리거 및 복잡한 세션 분석에 사용되는 스트리밍 데이터를 처리할 수 있습니다.

- * Streaming ETL. * 데이터가 데이터 저장소로 푸시되기 전에 지속적으로 정리 및 집계됩니다. Netflix는 Kafka 및 Spark 스트리밍을 사용하여 다양한 데이터 소스에서 매일 수십억 개의 이벤트를 처리할 수 있는 실시간 온라인 영화 추천 및 데이터 모니터링 솔루션을 구축합니다. 그러나 일괄 처리를 위한 기존 ETL은 다르게 처리됩니다. 이 데이터를 먼저 읽은 다음 데이터베이스에 쓰기 전에 데이터베이스 형식으로 변환됩니다.
- * 데이터 보강. * Spark 스트리밍은 라이브 데이터를 정적 데이터로 보강하여 보다 실시간 데이터 분석을 가능하게 합니다. 예를 들어, 온라인 광고주는 고객 행동에 대한 정보를 바탕으로 맞춤화된 맞춤형 광고를 제공할 수 있습니다.
- * 이벤트 감지 트리거 * 스파크 스트리밍을 사용하면 잠재적으로 심각한 문제를 나타낼 수 있는 비정상적인 동작을 신속하게 감지하고 대응할 수 있습니다. 예를 들어, 금융 기관은 사기 거래를 탐지 및 중지하는 트리거를 사용하며 병원에서는 환자의 바이탈 사인에 감지된 위험한 건강 변화를 감지하기 위해 트리거를 사용합니다.
- * 복잡한 세션 분석. * Spark 스트리밍은 웹 사이트나 응용 프로그램에 로그인한 후 사용자 활동과 같은 이벤트를 수집하여 그룹화하고 분석합니다. 예를 들어 Netflix는 이 기능을 사용하여 실시간 영화 권장 사항을 제공합니다.

스트리밍 데이터 구성, Confluent Kafka 검증 및 성능 테스트에 대한 자세한 내용은 을 참조하십시오 "TR-4912: NetApp을 통해 Confluent Kafka 계층형 스토리지를 위한 모범 사례 지침".

### 머신 러닝

Spark 통합 프레임워크는 MLlib(Machine Learning Library)를 사용하여 데이터 세트에서 반복되는 쿼리를 실행할 수 있도록 도와줍니다. MLlib는 예측 인텔리전스, 마케팅 목적으로 고객 세분화, 감정 분석과 같은 일반적인 빅 데이터 기능의 클러스터링, 분류 및 차원 감소 등의 영역에서 사용됩니다. MLlib는 네트워크 보안에 사용되어 데이터 패킷의 실시간 검사를 수행하여 악의적인 활동의 징후를 파악합니다. 이 솔루션은 보안 제공업체가 새로운 위협에 대해 학습하고 해커보다 앞서나가는 동시에 클라이언트를 실시간으로 보호할 수 있도록 도와줍니다.

### 딥 러닝

TensorFlow는 업계 전반에 걸쳐 사용되는 인기 있는 딥 러닝 프레임워크입니다. TensorFlow는 CPU 또는 GPU 클러스터에 대한 분산 교육을 지원합니다. 이렇게 분산된 교육을 통해 사용자는 많은 양의 데이터에서도 딥 레이어가 많이 포함된 상태로 데이터를 실행할 수 있습니다.

얼마 전까지만 해도 TensorFlow에 Apache Spark를 사용하려면 PySpark에서 TensorFlow에 필요한 ETL을 수행한 다음 중간 스토리지에 데이터를 써야 했습니다. 그런 다음 실제 훈련 프로세스를 위해 TensorFlow 클러스터에 데이터를 로드합니다. 이 워크플로우에서는 사용자가 ETL용 클러스터와 TensorFlow의 분산 교육용으로 각각 하나씩, 두 개의 서로 다른 클러스터를 유지해야 했습니다. 여러 클러스터를 실행하고 유지하는 일은 일반적으로 지루하고 시간이 오래 걸립니다.

이전 Spark 버전의 DataFrames 및 RDD는 랜덤 액세스가 제한되었기 때문에 딥 러닝에 적합하지 않았습니다. 프로젝트 수소가 포함된 Spark 3.0에서는 딥 러닝 프레임워크에 대한 기본 지원이 추가됩니다. 이 접근 방식을 사용하면 Spark 클러스터에서 MapReduce를 기반으로 하지 않는 일정을 수행할 수 있습니다.

### 대화형 분석

Apache Spark는 SQL, R, Python 등 Spark 이외의 개발 언어를 사용하여 샘플링하지 않고 탐색 쿼리를 수행할 수 있을 만큼 빠릅니다. Spark는 시각화 도구를 사용하여 복잡한 데이터를 처리하고 대화형으로 시각화합니다. Spark with 구조화된 스트리밍은 웹 분석의 라이브 데이터에 대한 대화형 쿼리를 수행하여 웹 방문자의 현재 세션에 대해 대화형 쿼리를 실행할 수 있도록 합니다.

### 추천 시스템

기업과 소비자가 온라인 쇼핑, 온라인 엔터테인먼트 및 기타 다양한 산업의 급격한 변화에 대응함에 따라, 지난 몇 년 동안 추천 시스템은 우리의 삶에 엄청난 변화를 가져왔습니다. 실제로 이러한 시스템은 생산 과정에서 AI의 가장 확실한 성공 사례 중 하나입니다. 많은 실제 사용 사례에서 추천 시스템은 NLP 백엔드와 상호 작용 AI 또는 챗봇과 결합되어 관련 정보를 얻고 유용한 추론을 생성합니다.

오늘날, 많은 소매업체들은 온라인 구매 및 매장에서 픽업, 큐브사이드 픽업, 셀프 체크아웃, 스캔 및 이동 등 새로운 비즈니스 모델을 채택하고 있습니다. COVID-19가 범세계적으로 확산되고 있는 가운데, 이러한 모델은 더욱 안전하고 편리한 쇼핑을 통해 뚜렷하게 부각되고 있습니다. AI는 소비자 행동의 영향을 받는 디지털 트렌드에 있어 매우 중요합니다. 고객의 증가하는 요구사항을 충족하고, 고객 경험을 보강하고, 운영 효율성을 개선하고, 수익을 증대하기 위해 NetApp은 엔터프라이즈 고객과 기업이 머신 러닝 및 딥 러닝 알고리즘을 사용하여 더 빠르고 정확한 추천 시스템을 설계할 수 있도록 지원합니다.

협업 필터링, 콘텐츠 기반 시스템, 딥 러닝 추천 모델(DLRM), 하이브리드 기술 등 권장 사항을 제공하는 데 사용되는 몇 가지 기술이 있습니다. 고객은 이전에 PySpark를 사용하여 권장 시스템 생성을 위한 협업 필터링을 구현했습니다. Spark MLlib는 DLRM이 등장하기 전에 엔터프라이즈 사이에서 매우 널리 사용되는 알고리즘인 협업 필터링을 위해 교류 최소 사각형(ALS)을 구현합니다.

### 자연어 처리

자연어 처리(NLP)로 가능해진 대화형 AI는 컴퓨터가 인간과 통신할 수 있도록 지원하는 AI의 지점입니다. NLP는

스마트 비서 및 챗봇에서 Google 검색 및 예측 텍스트에 이르기까지 모든 업계 수직 및 다양한 사용 사례에서 널리 사용되고 있습니다. 에 따르면 "Gartner" 2022년까지 70%의 사용자가 매일 대화형 AI 플랫폼과 상호 작용할 것으로 예측 인간과 기계 사이의 고품질 대화를 위해서는 신속하고 지능적이며 자연스러운 대화가 이루어져야 합니다.

고객은 NLP 및 ASR(자동 음성 인식) 모델을 처리하고 교육하기 위해 대량의 데이터가 필요합니다. 또한 에지, 코어, 클라우드 전반에서 데이터를 이동해야 하며, 인류와 자연적 통신을 위해 수 밀리초 내에 추론을 수행할 수 있는 기능이 필요합니다. NetApp AI 및 Apache Spark는 컴퓨팅, 스토리지, 데이터 처리, 모델 교육, 미세 조정, 있습니다.

정서 분석은 NLP에서 텍스트에서 긍정적, 부정적 또는 중립적 감정을 추출하는 연구 분야입니다. 고객 의견 분석에는 지원 센터 직원의 성과 파악부터 적절한 자동 챗봇 응답 제공에 이르기까지 다양한 활용 사례가 있습니다. 또한 분기별 수익 통화 시 기업 담당자와 대상 간의 상호 작용을 기반으로 회사의 주가를 예측하기도 했습니다. 또한, 감정 분석을 사용하여 브랜드가 제공하는 제품, 서비스 또는 지원에 대한 고객의 관점을 결정할 수 있습니다.

을 사용했습니다 "스파크 NLP" 라이브러리 시작 "John Snow Labs를 참조하십시오" 를 포함하여 Transformers(BERT) 모델의 사전 교육 파이프라인 및 양방향 인코더 표현을 로드합니다 "금융 뉴스 정서" 및 "핀베르트"즉, 토큰화, 명명된 엔터티 인식, 모델 교육, 피팅 및 정서 분석을 대규모로 수행합니다. Spark NLP는 BERT, Albert, Electra, XLNet, DistillBERT 등의 최첨단 변압기(transformer)를 제공하는 유일한 오픈 소스 NLP 라이브러리입니다. Roberta, DeBERTa, XLM-Roberta, Longrofer, Elmo, Universal 문장 인코더, Google T5, MarianMT 및 GPT2. 이 라이브러리는 Python 및 R 뿐만 아니라 Apache Spark를 기본적으로 확장하여 JVM 에코시스템(Java, Scala, Kotlin)에서도 사용할 수 있습니다.

## 주요 **AI, ML** 및 **DL** 사용 사례 및 아키텍처

주요 AI, ML 및 DL 사용 사례 및 방법론을 다음 절로 나눌 수 있습니다.

### **NLP** 파이프라인 및 **TensorFlow** 분산 추론을 스파크합니다

다음 목록에는 데이터 과학 커뮤니티가 다양한 개발 수준에서 채택한 가장 널리 사용되는 오픈 소스 NLP 라이브러리가 포함되어 있습니다.

- "자연어 도구 키트(NLTK)". 모든 NLP 기술에 대한 완벽한 도구 키트. 2000년대 초반부터 유지되었습니다.
- "텍스트 부분". NLTK 및 패턴 위에 구축된 사용하기 쉬운 NLP 도구 Python API.
- "스탠포드 코어 NLP". 스탠포드 NLP Group이 개발한 Java의 NLP 서비스 및 패키지.
- "젠심". 인간을 위한 주제 모델링은 체코 디지털 수학 라이브러리 프로젝트를 위한 Python 스크립트의 모음으로 시작되었습니다.
- "스파이". 트랜스포머용 GPU 가속 기능을 갖춘 Python 및 Cython을 사용한 엔드 투 엔드 산업용 NLP 워크플로.
- "빠른 텍스트". Facebook의 AI 연구(Fair) 연구소에서 만든 단어 인식 및 문장 분류용 무료 경량 오픈 소스 NLP 라이브러리.

SPARK NLP는 모든 NLP 작업 및 요구 사항을 충족하는 단일 통합 솔루션으로, 실제 생산 사용 사례에 맞게 확장 가능하고 성능이 뛰어나며 정확도가 높은 NLP 기반 소프트웨어를 지원합니다. 이 제품은 전송 학습을 활용하고 연구 및 산업 전반에 걸쳐 최신 최신 알고리즘 및 모델을 구현합니다. Spark는 위의 라이브러리에 대한 완벽한 지원이 부족하기 때문에 Spark NLP를 기반으로 구축되었습니다 "스파크 ML" Spark의 범용 인메모리 분산 데이터 처리 엔진을 미션 크리티컬 프로덕션 워크플로우를 위한 엔터프라이즈급 NLP 라이브러리로 활용하십시오. 주석 달기 는 규칙 기반 알고리즘, 머신 러닝 및 TensorFlow를 활용하여 딥 러닝 구현을 지원합니다. 여기에는 토큰화, 레몬화, 스테밍, 부분 음성 태깅, 명명된 엔터티 인식 등을 포함한 일반적인 NLP 작업이 포함됩니다. 맞춤법 검사 및 정서 분석.

Transformers(BERT)의 양방향 인코더 표현은 NLP를 위한 변압기 기반 기계 학습 기술입니다. 사전 교육과 미세 조정 개념을 대중화했습니다. BERT의 변압기 아키텍처는 재발성 신경망(RNN) 기반 언어 모델보다 장기적인 종속성을 더 잘 모델링하는 기계 번역에서 비롯되었습니다. 또한 MLM(Masked Language Modeling) 작업도 도입되었으며, 이

작업에서는 모든 토큰의 임의 15%가 마스킹되고 모델이 이를 예측하여 진정한 방향성을 가능하게 합니다.

전문 언어와 해당 영역에 레이블이 지정된 데이터가 없기 때문에 재무 감정의 분석은 어렵습니다. "핀베르트"사전 훈련된 BERT에 기반한 언어 모델은 도메인 적용에 적합합니다 "로이터 TRC2", 재정 문제, 라벨이 붙은 데이터( "금융 서비스 은행")를 참조하십시오. 연구원들은 재무 용어로 뉴스 기사에서 4개, 500개 문장을 추출했습니다. 그때 16명의 전문가들과 재정 배경을 가진 대학생들은 그 문장을 긍정적이고 중립적이며 부정적으로 표시했다. 2016년부터 2020년까지 FinBERT와 사전 교육 받은 두 개의 다른 파이프라인을 사용하여 상위 10개 NASDAQ 회사 수익 통화 녹취록의 정서를 분석하기 위해 Spark 워크플로를 구축했습니다 "금융 뉴스를 위한 감정 분석", "문서 DL을 설명합니다 ") Spark NLP에서.

Spark NLP를 위한 기본 딥 러닝 엔진은 손쉬운 모델 구축, 어디서나 강력한 ML 생산, 연구를 위한 강력한 실험을 지원하는 기계 학습용 엔드 투 엔드 오픈 소스 플랫폼인 TensorFlow입니다. 따라서 Spark 'YARN 클러스터' 모드에서 파이프라인을 실행할 때 기본적으로 하나의 마스터 노드와 여러 작업자 노드 및 클러스터에 마운트된 네트워크 연결 스토리지에서 데이터와 모델 병렬화를 통해 분산된 TensorFlow를 실행하고 있었습니다.

**Horovod**의 분산 훈련

MapReduce 관련 성능을 위한 핵심 Hadoop 검증은 TeraGen, TeraSort, TeraValidate 및 DFSIO(읽기 및 쓰기)를 통해 수행됩니다. TeraGen 및 TeraSort 검증 결과는 에 나와 있습니다 "TR-3969: Hadoop용 NetApp 솔루션" E-Series의 경우 및 AFF의 경우 "스토리지 계층화"(xref) 섹션

고객의 요청에 따라 Spark와 함께 분산된 교육을 다양한 사용 사례 중 가장 중요한 것으로 간주합니다. 이 문서에서는 을 사용했습니다 "Spark의 Hoborod" AFF(All Flash FAS) 스토리지 컨트롤러, Azure NetApp Files 및 StorageGRID를 사용하여 NetApp 사내, 클라우드 네이티브 및 하이브리드 클라우드 솔루션을 통해 Spark 성능을 검증합니다.

Horovod on Spark 패키지는 Spark 클러스터에서 분산된 훈련 워크로드를 간단하게 실행할 수 있도록 Horovod를 둘러싸는 편리한 래퍼를 제공합니다. 따라서 훈련 및 추론 데이터가 상주하는 Spark에서 데이터 처리, 모델 훈련 및 모델 평가를 모두 수행하는 엄격한 모델 설계 루프를 사용할 수 있습니다.

Spark에서 Horovod를 실행하기 위한 두 가지 API(고급 추정기 API 및 하위 수준의 Run API)가 있습니다. 둘 다 동일한 기본 메커니즘을 사용하여 Spark 실행기에서 Horovod를 실행하지만 Estimator API는 데이터 처리, 모델 훈련 루프, 모델 체크포인트, 메트릭 수집 및 분산 교육을 추상화합니다. Horovod Spark Estimators, TensorFlow 및 Keras를 사용하여 에 기반한 엔드 투 엔드 데이터 준비 및 분산 교육 워크플로를 사용했습니다 "Kaggle Rossmann 매장 판매" 경쟁업체.

'keras_spark_horovod_rossmann_estimator.py' 스크립트는 섹션에서 찾을 수 있습니다 "주요 활용 사례별로 Python 스크립트" 여기에는 다음 세 가지 부분이 포함됩니다.

- 첫 번째 부분은 Kaggle이 제공하고 커뮤니티에서 수집한 초기 CSV 파일 세트에 대해 다양한 데이터 전처리 단계를 수행합니다. 입력 데이터는 '검증' 하위 세트와 테스트 데이터 세트가 있는 교육 세트로 구분됩니다.
- 두 번째 부분은 로그 시그마리드 활성화 기능과 ADAM 옵티마이저가 있는 Keras Deep Neural Network(DNN) 모델을 정의하고 Spark On을 사용하여 모델 분산 훈련을 수행합니다.
- 세 번째 부분은 검증 세트의 전체 평균 절대 오류를 최소화하는 최상의 모델을 사용하여 테스트 데이터 세트에 대한 예측을 수행합니다. 그런 다음 출력 CSV 파일을 만듭니다.

섹션을 참조하십시오 ""머신 러닝"" 다양한 런타임 비교 결과.

**CTR** 예측에 **Keras**를 사용한 다중 작업자 딥 러닝

최근 ML 플랫폼 및 애플리케이션의 발전으로 이제 대규모 학습에 많은 관심이 집중되고 있습니다. 클릭 비율(CTR)은 온라인 광고 노출 100회 당 평균 클릭 수(백분율로 표시)로 정의됩니다. 디지털 마케팅, 소매, 전자 상거래 및 서비스 공급자를 포함한 다양한 산업 및 사용 사례에서 핵심 메트릭으로 널리 채택되고 있습니다. 자세한 내용은 를

참조하십시오 "TR-4904: Azure에서 제공되는 분산 교육 - 클릭 비율 예측" CTR 및 Kubernetes를 통한 엔드 투 엔드 클라우드 AI 워크플로우 구현, 분산된 데이터 ETL, Dask 및 CUDA ML을 사용한 모델 교육에 대한 자세한 내용을 확인하십시오.

이 기술 보고서에서는 의 변형을 사용했습니다 "Criteo Terabyte Click Logs 데이터 세트" (TR-4904 참조) Keras를 사용하여 여러 작업자가 딥 및 크로스 네트워크(DCN) 모델로 Spark 워크플로를 구축하여 로그 손실 오류 기능과 기준 Spark ML 회귀 모델의 성능을 비교한 딥 러닝(Deep and Cross Network) 모델에 대해 알아보십시오. DCN은 경계가 지정된 수준의 효과적인 기능 상호 작용을 효율적으로 캡처하고, 고도의 비선형 상호 작용을 학습하며, 수동 기능 엔지니어링 또는 철저한 검색이 필요하지 않으며, 계산 비용이 낮습니다.

웹 스케일 추천 시스템을 위한 데이터는 대부분 불연속적이고 범주적이며, 이는 기능 탐색이 어려운 크고 부족한 기능 공간으로 이어집니다. 이는 대부분의 대규모 시스템을 물류 회귀와 같은 선형 모델로 제한합니다. 그러나 자주 예측 가능한 기능을 식별하고 보이지 않거나 희귀한 크로스 기능을 탐색하는 것은 좋은 예측을 위한 열쇠입니다. 선형 모델은 단순하고, 해석 가능하며, 확장이 쉽지만 표현 능력이 제한적입니다.

반면 크로스 기능은 모델의 표현 능력을 향상시키는 데 중요한 것으로 나타났습니다. 하지만 이러한 기능을 식별하려면 수동 기능 엔지니어링이나 철저한 검색이 필요한 경우가 많습니다. 보이지 않는 피처 상호작용에 일반화하기는 어려운 경우가 많습니다. DCN과 같은 교차 신경망(cross neural network)을 사용하면 자동 방식으로 기능적으로 명시적으로 교차하여 작업별 기능별 엔지니어링을 피할 수 있습니다. 크로스 네트워크는 다중 계층으로 구성되어 있으며, 이 경우 가장 높은 수준의 상호 작용이 레이어 깊이에 따라 결정될 수 있습니다. 각 계층은 기존 레이어를 기반으로 보다 높은 수준의 상호 작용을 생성하고 이전 레이어의 상호 작용을 유지합니다.

DNN(Deep Neural Network)은 여러 기능에서 매우 복잡한 상호 작용을 포착할 수 있는 가능성을 가지고 있습니다. 그러나 DCN과 비교할 때 거의 많은 매개변수가 필요하며, 상호 기능을 명시적으로 구성할 수 없으며, 일부 유형의 기능 상호 작용을 효율적으로 학습하지 못할 수 있습니다. 크로스 네트워크는 메모리 효율적이고 구현하기 쉽습니다. 상호 및 DNN 구성 요소를 공동으로 교육하여 예측 가능한 기능 상호 작용을 효율적으로 캡처하고 Critio CTR 데이터 세트에 대한 최첨단 성능을 제공합니다.

DCN 모델은 포매 및 스태킹 계층, 크로스 네트워크 및 딥 네트워크를 병렬로 사용하여 시작합니다. 두 네트워크의 출력을 결합하는 최종 조합 계층이 이어집니다. 입력 데이터는 희소 및 고밀도 피처를 가진 벡터가 될 수 있습니다. Spark에서는 두 가지 모두 가능합니다 "ML" 및 "mllib를 참조하십시오" 라이브러리에는 'parseVector' 형식이 포함됩니다. 따라서 사용자는 두 가지 기능을 구분할 수 있어야 하며 각 함수와 메서드를 호출할 때 유의해야 합니다. CTR 예측과 같은 웹 스케일 추천 시스템에서 입력은 주로 범주적인 기능(예: ``country=USA')입니다. 이러한 기능은 `[0,1,0,…]''와 같은 하나의 인기 벡터로 인코딩되는 경우가 많습니다. 'ParseVector'를 사용한 단일 핫 인코딩(OHE)은 끊임없이 변화하고 성장하는 어휘를 사용하여 실제 데이터세트를 처리할 때 유용합니다. 에서 예제를 수정했습니다 "DeepCTR" 대용량 어휘를 처리하기 위해 DCN의 포매 및 스태킹 계층에 포매 벡터를 만듭니다.

를 클릭합니다 "Criteo Display Ads 데이터 세트" 광고 클릭률을 예측합니다. 13개의 정수 기능과 각 범주에 높은 카디널리티가 있는 26개의 범주 기능이 있습니다. 이 데이터 세트의 경우 입력 크기가 커서 로그손실이 0.001로 향상되는 것이 실질적으로 중요합니다. 대규모 사용자 기반의 예측 정확도가 약간 개선되는 경우 회사의 수익이 크게 증가할 수 있습니다. 데이터 세트에는 7일 동안 11GB의 사용자 로그가 포함되어 있으며, 이는 약 4100만 개의 레코드에 해당합니다. Spark dataFrame.RandomSplit() 기능을 사용하여 교육(80%), 교차 검증(10%) 및 나머지 10%의 테스트 데이터를 무작위로 분할했습니다.

DCN은 Keras를 통해 TensorFlow에 구현되었습니다. DCN을 통해 모델 교육 프로세스를 구현하는 주요 구성 요소는 다음과 같습니다.

- * 데이터 처리 및 포함. * 실제 가치가 있는 기능은 로그 변환을 적용하여 정규화됩니다. 카테고리 피처의 경우, 치수 6 × (카테고리 카디널리티) 1/4의 고밀도 벡터에 피처를 포함시킵니다. 모든 임베디드 디딩을 연결하면 치수 1026의 벡터가 됩니다.
- 최적화. * ADAM 옵티마이저를 사용하여 미니 배치 확률적 최적화를 적용했습니다. 배치 크기가 512로 설정되었습니다. 배치 정규화가 딥 네트워크에 적용되고 그레이디언트 클립 표준은 100으로 설정되었습니다.
- "규칙화." 우리는 L2 규칙화 또는 중도탈락이 효과가 없는 것으로 확인됨에 따라 조기 정지에 사용했습니다.

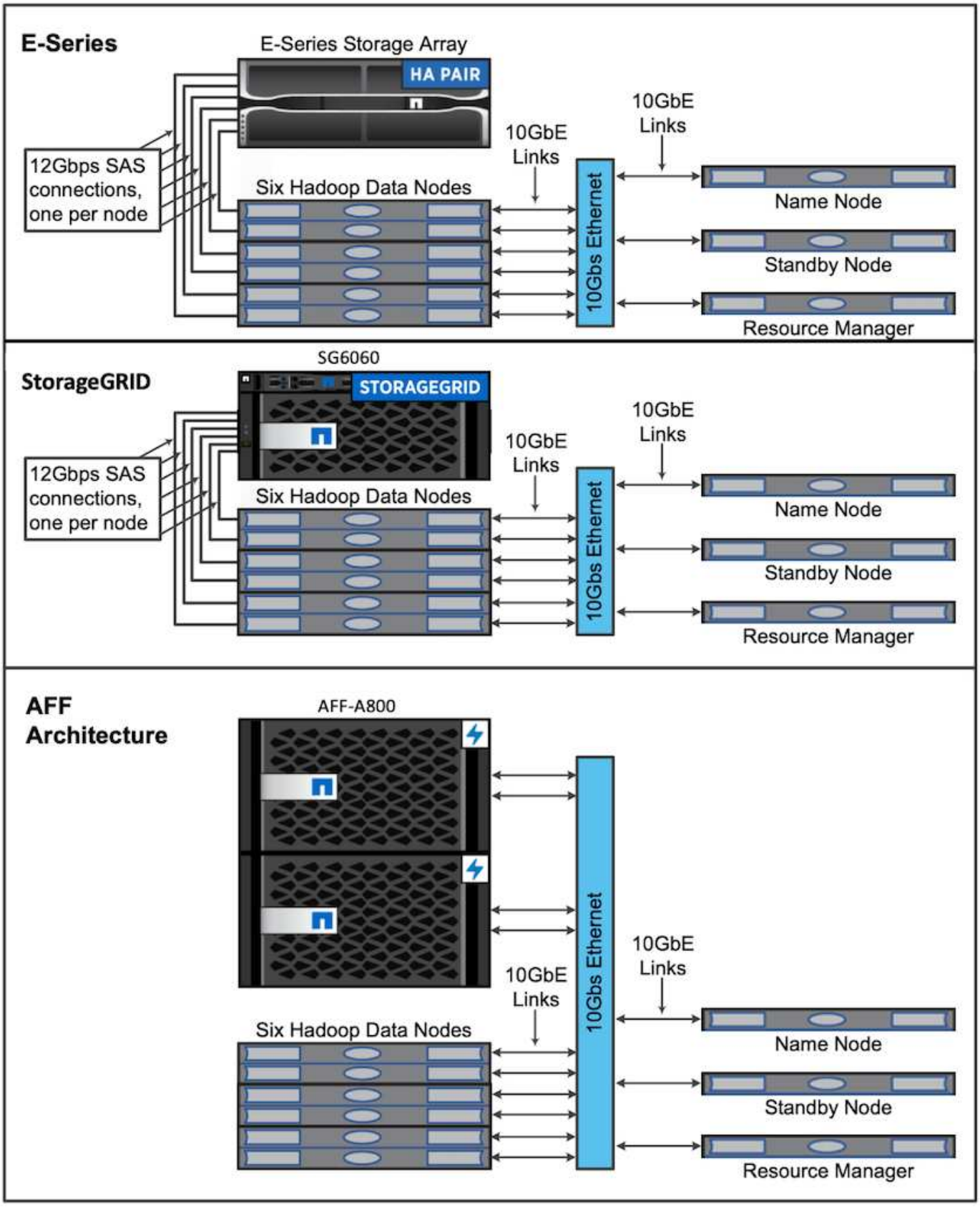
- * Hyperparameters. * 숨겨진 계층 수, 숨겨진 계층 크기, 초기 학습 속도 및 교차 계층 수에 대한 그리드 검색을 기반으로 결과를 보고합니다. 숨겨진 레이어의 수는 2-5개까지이며 숨겨진 레이어 크기는 32-1024입니다. DCN의 경우 교차 계층 수는 1에서 6까지 있었습니다. 초기 학습 속도는 0.0001에서 0.0001로 조정되었으며 0.0001씩 증가하였습니다. 모든 실험은 훈련 단계 150,000에 일찍 멈추어 적용되었고, 그 이후에는 과다 피팅이 발생하기 시작했습니다.

DCN 외에도 CTR 예측을 위해 기타 인기 있는 딥 러닝 모델도 테스트했습니다(예 "DeepFM", "xDeepFM", "자동 내부", 및 "DCN v2".

검증에 사용된 아키텍처

이 검증을 위해 4개의 작업자 노드와 AFF-A800 HA 쌍의 1개의 마스터 노드를 사용했습니다. 모든 클러스터 구성원이 10GbE 네트워크 스위치를 통해 연결되었습니다.

이러한 NetApp Spark 솔루션 검증을 위해 E5760, E5724, AFF-A800의 3가지 스토리지 컨트롤러를 사용했습니다. E-Series 스토리지 컨트롤러는 12Gbps SAS로 연결된 5개의 데이터 노드에 연결되었습니다. AFF HA 쌍 스토리지 컨트롤러는 10GbE 연결을 통해 Hadoop 작업자 노드에 내보낸 NFS 볼륨을 제공합니다. Hadoop 클러스터 멤버는 E-Series, AFF 및 StorageGRID 하둡 솔루션에서 10GbE 연결을 통해 연결했습니다.




테스트 결과

TeraGen 벤치마킹 툴에서 TeraSort 및 TeraValidate 스크립트를 사용하여 E5760, E5724 및

AFF-A800 구성을 사용하여 Spark 성능 검증을 측정했습니다. 또한, Spark NLP 파이프라인 및 TensorFlow 배포 교육, Horovod 분산 교육, DeepFM을 통한 CTR 예측에 Keras를 사용한 다중 작업자 딥 러닝 등 3가지 주요 사용 사례를 테스트했습니다.

E-Series와 StorageGRID 검증 모두에 Hadoop 복제 계수 2를 사용했습니다. AFF 검증에서는 하나의 데이터 소스만 사용했습니다.

다음 표에는 Spark 성능 검증을 위한 하드웨어 구성이 나와 있습니다.

| 유형 | **Hadoop** 작업자 노드 | 드라이브 유형입니다 | 노드당 드라이브 수 | 스토리지 컨트롤러 |
|---|---|---|---|---|
| SG6060 | 4 | SAS를 참조하십시오 | 12 | 단일 고가용성(HA) 쌍 |
| E5760에서 | 4 | SAS를 참조하십시오 | 60 | 단일 HA 쌍 |
| E5724)를 참조하십시오 | 4 | SAS를 참조하십시오 | 24 | 단일 HA 쌍 |
| AFF800 | 4 | SSD를 지원합니다 | 6 | 단일 HA 쌍 |

다음 표에는 소프트웨어 요구 사항이 나와 있습니다.

| 소프트웨어 | 버전 |
|---|---|
| RHEL을 참조하십시오 | 7.9 |
| OpenJDK 런타임 환경 | 1.8.0 |
| OpenJDK 64비트 서버 VM | 25.302 |
| 기트 | 2.24.1 |
| GCC/G++ | 11.2.1 |
| 스파크 | 3.2.1 |
| 피스파크 | 3.1.2 |
| 스파르크LP | 3.4.2 |
| TensorFlow를 참조하십시오 | 2.9.0 |
| Keras | 2.9.0 |
| 호로브 | 0.24.3 |

재무 정서 분석

발표했습니다 "TR-4910: NetApp AI를 통한 고객 커뮤니케이션의 감정 분석"즉, 포괄적인 대화형 AI 파이프라인은 를 사용하여 구축되었습니다 "NetApp DataOps 툴킷", AFF 스토리지 및 NVIDIA DGX 시스템. 파이프라인은 DataOps 툴킷을 활용하여 배치 오디오 신호 처리, 자동 음성 인식(ASR), 전송 학습 및 정서 분석을 수행합니다. "NVIDIA Riva SDK를 참조하십시오", 및 "Tao 프레임워크". 정서 분석 활용 사례를 금융 서비스 산업으로 확대하면서 스파르크LP 워크플로를 구축하고, 명명된 엔터티 인식 등의 다양한 NLP 작업을 위한 BERT 모델을 3개 로드했으며, NASDAQ 상위 10개 기업의 분기별 수익 통화에 대해 문장 수준의 감정을 얻었습니다.

다음 스크립트의 entiment_analysis_spark. py는 다음 표와 같이 FinBERT 모델을 사용하여 HDFS에서 전사체를 처리하고 양의, 중성적이고, 부정적인 정서 수를 산출합니다.

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
hdfs:///data1/Transcripts/
> ./sentiment_analysis_hdfs.log 2>&1
real13m14.300s
user557m11.319s
sys4m47.676s
```

다음 표에는 2016년부터 2020년까지 NASDAQ 상위 10대 기업에 대한 순익금 및 문장 수준의 정서 분석이 나와 있습니다.

| 정서 수와 백분율 | **10**개 회사 모두 | **AAPL** | **AMD** | **AMZN** | **CSCO** | **GOOGL** | 아이**NTC** | **MSFT** | **NVDA** |
|---|---|---|---|---|---|---|---|---|---|
| 양수입니다 | 7447 | 1567 | 743 | 290 | 682를 참조하십시오 | 826 | 824 | 904 | 417 |
| 중립 카운트 | 64067 | 6856)을 참조하십시오 | 7596 | 5086 | 6650 | 5914 | 6099 | 5715)를 참조하십시오 | 6189 |
| 음의 카운트 | 1787 | 253 | 213 | 84 | 189 | 97 | 282 | 202 | 89 |
| 분류되지 않은 카운트 | 196 | 0 | 0 | 76 | 0 | 0 | 0 | 1 | 0 |
| (총 카운트) | 73497 | 8676)을 참조하십시오 | 8552 | 5536 | 7521 | 6837 | 7205 | 6822 | 6695)를 참조하십시오 |

백분율의 경우 CEO와 CFO가 말하는 대부분의 문장이 사실에 근거해 중립적인 감정을 가지고 있습니다. 수익 통화 중에 분석가들은 긍정적이거나 부정적인 감정을 전달할 수 있는 질문을 합니다. 이에 따라 내당일 또는 다음 날 주가가 음성이나 양성으로 어떻게 영향을 미치는지를 정량적으로 조사할 필요가 있다.

다음 표에는 NASDAQ 상위 10대 기업에 대한 문장 수준의 감정 분석이 백분율로 나와 있습니다.

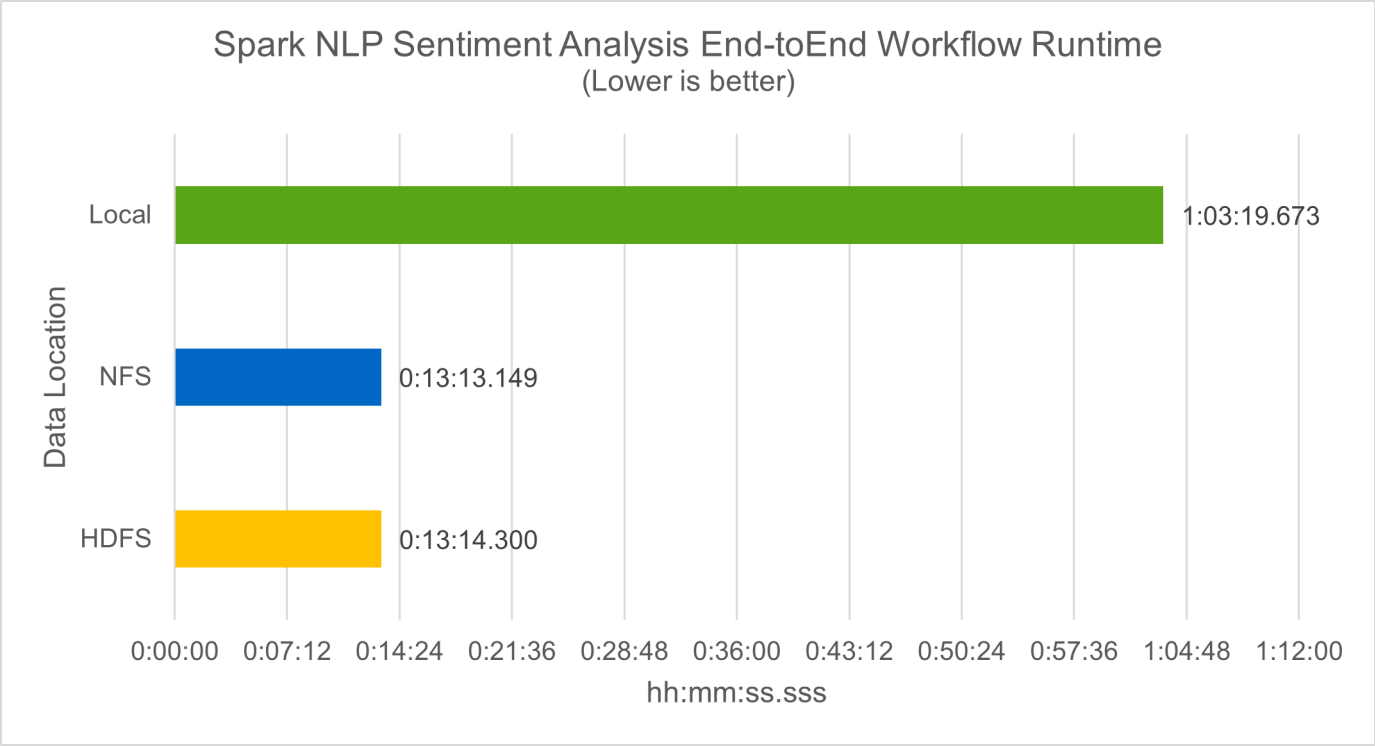
| 정서 백분율 | **10**개 회사 모두 | **AAPL** | **AMD** | **AMZN** | **CSCO** | **GOOGL** | 아이**NTC** | **MSFT** | **NVDA** |
|---|---|---|---|---|---|---|---|---|---|
| 긍정적 | 10.13% | 18.06% | 8.69% | 5.24% | 9.07% | 12.08% | 11.44% | 13.25% | 6.23% |
| 중립 | 87.17% | 79.02% | 88.82% | 91.87% | 88.42% | 86.50% | 84.65% | 83.77% | 92.44% |

| 정서 백분율 | **10**개 회사 모두 | **AAPL** | **AMD** | **AMZN** | **CSCO** | **GOOGL** | 아이**NTC** | **MSFT** | **NVDA** |
|---|---|---|---|---|---|---|---|---|---|
| 네거티브 | 2.43% | 2.92% | 2.49% | 1.52% | 2.51% | 1.42% | 3.91% | 2.96% | 1.33% |
| 분류되지 않았습니다 | 0.27% | 0% | 0% | 1.37% | 0% | 0% | 0% | 0.01% | 0% |

워크플로우 런타임 측면에서 HDFS의 로컬 모드에서 분산 환경으로 4.78배, NFS를 활용하여 0.14% 향상되었습니다.

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
file:///sparkdemo/sparknlp/Transcripts/
> ./sentiment_analysis_nfs.log 2>&1
real13m13.149s
user537m50.148s
sys4m46.173s
```

다음 그림에서 볼 수 있듯이 데이터 및 모델 병렬 처리를 통해 데이터 처리 및 분산 TensorFlow 모델 추론 속도가 향상되었습니다. NFS의 데이터 위치는 워크플로우 병목 현상이 사전 교육 모델을 다운로드하는 것이기 때문에 런타임 성능이 약간 향상되었습니다. 전사체 데이터 세트 크기를 늘릴 경우 NFS의 장점은 더욱 명확합니다.




**Horovod** 성과를 통한 분산 훈련

다음 명령을 실행하면 코어 1개가 포함된 160개의 실행자가 있는 단일 마스터 노드를 사용하여 Spark 클러스터에서 런타임 정보와 로그 파일이 생성되었습니다. 메모리 부족 오류가 발생하지 않도록 executor 메모리가 5GB로 제한되었습니다. 섹션을 참조하십시오 ""주요 활용 사례별로 Python 스크립트"" 'keras_spark_horovod_rossmann_estimator.py'의 데이터 처리, 모델 훈련 및 모델 정확도 계산에 대한 자세한 내용

```
(base) [root@n138 horovod]# time spark-submit
--master local
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkusecase/horovod
--local-submission-csv /tmp/submission_0.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_local. log 2>&1
```

10번의 교육 Epoch를 사용한 결과 런타임은 다음과 같습니다.

```
real43m34.608s
user12m22.057s
sys2m30.127s
```

입력 데이터를 처리하고, DNN 모델을 교육하고, 정확도를 계산하고, 예측 결과를 위한 TensorFlow 체크포인트와 CSV

파일을 생성하는 데 43분 이상이 걸렸습니다. 교육 Epoch의 수를 10개로 제한했습니다. 실제로 만족스러운 모델 정확도를 보장하기 위해 대개 100으로 설정되어 있습니다. 일반적으로 교육 시간은 Epoch 수에 비례하여 확장됩니다.

다음으로 클러스터에서 사용할 수 있는 4개의 작업자 노드를 사용하고 HDFS에서 데이터와 함께 'YARN' 모드로 동일한 스크립트를 실행했습니다.

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir hdfs:///user/hdfs/tr-4570/experiments/horovod
--local-submission-csv /tmp/submission_1.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_yarn.log 2>&1
```

결과 런타임은 다음과 같이 개선되었습니다.

```
real8m13.728s
user7m48.421s
sys1m26.063s
```

Spark의 Horovod 모델과 데이터 병렬화를 통해 10번의 교육 Epoch로 "원사"와 "로컬" 모드의 실행 속도가 5.29배 빨라졌습니다. 이 그림은 다음 그림과 같이 전설적인 HDFS와 Local로 표시됩니다. 기본 TensorFlow DNN 모델 교육은 GPU를 사용하여 더 가속화될 수 있습니다. NetApp은 이 테스트를 수행하고 결과를 향후 기술 보고서에 게시할 계획입니다.

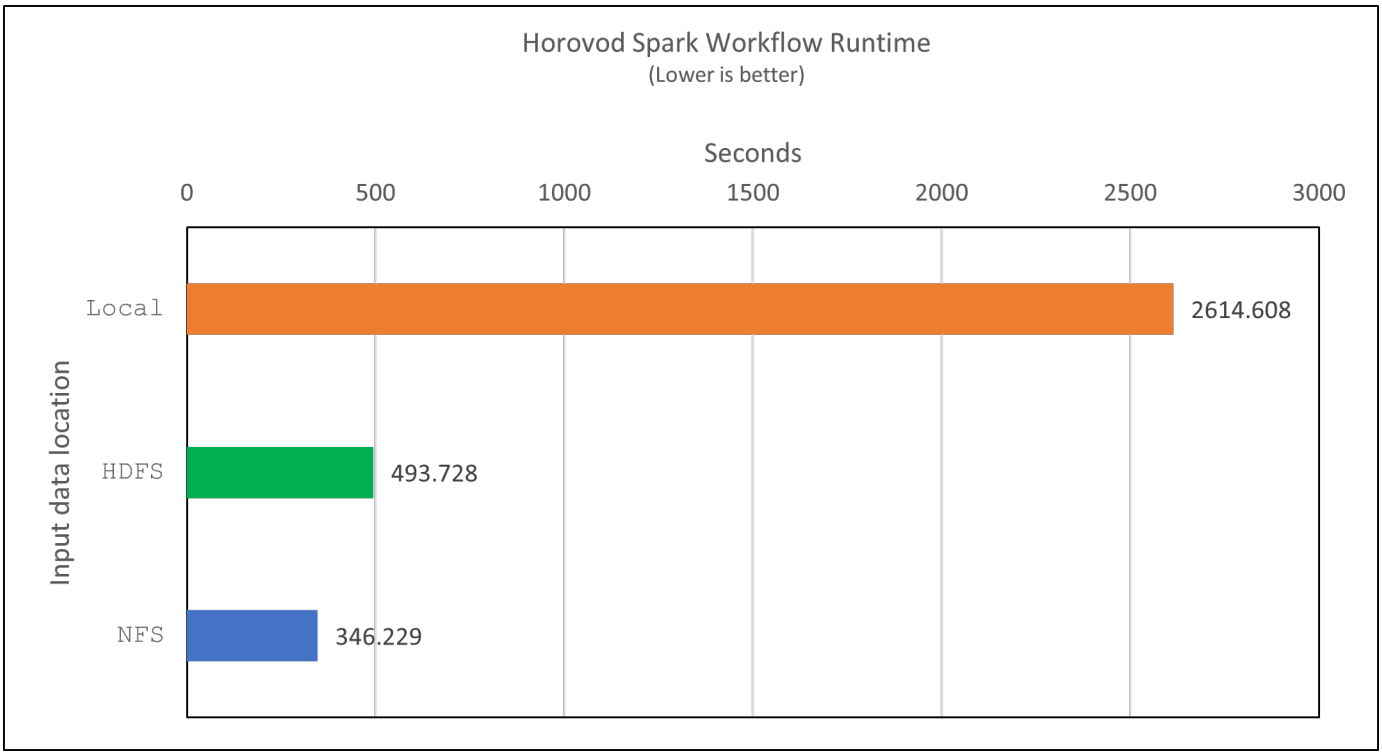
다음 테스트에서는 NFS에 상주하는 입력 데이터와 HDFS를 비교하여 실행 시간을 비교했습니다. AFF A800의 NFS 볼륨은 Spark 클러스터의 5개 노드(마스터 1개, 작업자 4명)에 걸쳐 '/Spkdemo/horovod'에 마운트되었습니다. NFS 마운트를 가리키는 '--data-dir' 매개 변수를 사용하여 이전 테스트와 비슷한 명령을 실행했습니다.

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkdemo/horovod
--local-submission-csv /tmp/submission_2.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_nfs.log 2>&1
```

NFS의 결과 런타임은 다음과 같습니다.

```
real 5m46.229s
user 5m35.693s
sys  1m5.615s
```

다음 그림과 같이 속도가 1.43배 더 향상되었습니다. 따라서 고객은 클러스터에 연결된 NetApp All-Flash 스토리지를 통해 Horovod Spark 워크플로우에서 빠른 데이터 전송 및 배포의 이점을 누리고 단일 노드에서 실행되는 것에 비해 7.55배 더 빠른 속도를 달성할 수 있습니다.




**CTR** 예측 성능을 위한 딥 러닝 모델

CTR을 최대화하도록 설계된 추천 시스템의 경우 낮은 순서에서 높은 순서로 수학적으로 계산할 수 있는 사용자 행동 뒤에 정교한 기능 상호 작용을 학습해야 합니다. 낮은 순서의 기능과 높은 순서의 기능 상호 작용은 둘 중 하나를 편향하지 않고 우수한 딥 러닝 모델에 똑같이 중요합니다. 인수 기계 기반 신경 네트워크인 DeepFM(Deep Factorization Machine)은 새로운 신경망 아키텍처에서 기능 학습을 위한 권장 사항과 딥 러닝을 위한 인수 기계(factorization Machine)를 결합합니다.

기존의 공장 인수 기계는 쌍 단위 기능 상호 작용을 기능 간의 잠재적 벡터의 내부 제품으로 모델링하고 이론적으로 높은 순서 정보를 캡처할 수 있지만, 실제로 머신 러닝 실무자는 높은 계산 및 스토리지 복잡성으로 인해 일반적으로 2차 기능 상호 작용만 사용합니다. Google과 같은 딥 뉴럴 네트워크 변형 "와이드 앰프, 딥 모델" 반면, 선형 와이드 모델과 딥 모델을 결합하여 하이브리드 네트워크 구조에서 정교한 기능 상호 작용을 학습합니다.

이 Wide & Deep Model에는 기본 와이드 모델과 딥 모델에 대한 두 가지 입력이 있으며, 그 중 후자는 여전히 전문적인 피처 엔지니어링을 필요로 하므로 다른 영역에 대해 일반화할 수 없는 기술을 렌더링합니다. 광각 및 딥 모델과 달리, DeepFM은 넓은 부분과 깊은 부분이 동일한 입력 및 포함 벡터를 공유하기 때문에 기능 엔지니어링 없이 RAW 기능으로 효율적으로 교육을 받을 수 있습니다.

먼저 Criteo '기차.txt'(11GB) 파일을 NFS 마운트에 저장된 CTR_트레인.csv라는 CSV 파일로 처리했습니다. /spclassification_criteo_spark.py라는 섹션을 사용하여 /spkdemo/TR-4570-data로 처리했습니다 ""각 주요 활용 사례에 대한 Python 스크립트"" 이 스크립트에서 함수 "process_input_file"은 여러 문자열 메소드를 수행하여 탭을

제거하고 구분 기호로 ",", 줄 바꿈으로 "\n""를 삽입합니다. 코드 블록이 주석으로 표시되도록 원래 기차 .txt만 처리하면 됩니다.

다음 DL 모델 테스트를 위해 입력 파일로 CTR_트레인.csv를 사용했습니다. 후속 테스트 실행에서 입력 CSV 파일은 "'레이블", 정수 밀도 기능 "['I1', 'I2', 'i3',..., 'I13']" 필드가 포함된 스키마가 있는 Spark DataFrame으로 읽혀졌습니다. 그리고 스파스 피처 "['C1','C2','C3',...,'C26']". 다음 'park-submit' 명령은 입력 CSV에서 수행하고 교차 검증을 위해 20% 분할로 DeepFM 모델을 교육하고 10번의 교육 Epoch 후에 최적의 모델을 선택하여 테스트 세트의 예측 정확도를 계산합니다.

```
(base) [root@n138 ~]# time spark-submit --master yarn --executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py --data
-dir file:///sparkdemo/tr-4570-data >
/tmp/run_classification_criteo_spark_local.log 2>&1
```

데이터 파일 'CTR_트레인.csv'가 11GB를 초과하므로 오류를 방지하려면 데이터 세트 크기보다 충분한 spark.driver.maxResultSize를 설정해야 합니다.

```
 spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
```

위 'parkSession.builder' 설정에서도 활성화했다 "Apache 화살표"이는 Df. toPandas() 메소드를 사용하여 Spark DataFrame을 Pandas DataFrame으로 변환합니다.

```
22/06/17 15:56:21 INFO scheduler.DAGScheduler: Job 2 finished: toPandas at
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py:96, took
627.126487 s
Obtained Spark DF and transformed to Pandas DF using Arrow.
```

무작위 분할 후, 교육 데이터 세트에 36M 행이 넘고 테스트 세트에 9M 샘플이 있습니다.

```
Training dataset size =  36672493
Testing dataset size =  9168124
```

이 기술 보고서는 GPU를 사용하지 않고 CPU 테스트에 집중되므로 적절한 컴파일러 플래그를 사용하여 TensorFlow를 구축하는 것이 중요합니다. 이 단계는 GPU 가속 라이브러리를 호출하지 않고 TensorFlow의 AVX(Advanced Vector Extensions) 및 AVX2 명령을 최대한 활용합니다. 이러한 기능은 벡터화된 추가, 피드 포워드 내부의 행렬 다중화 또는 역전파 DNN 교육과 같은 선형 대수 계산에 맞게 설계되었습니다. 256비트 부동 소수점(FP) 레지스터를 사용하는 AVX2에서 사용할 수 있는 FMA(Fused Multiply Add) 명령은 정수 코드 및 데이터 형식에 적합하며 최대 2배의 속도를 제공합니다. FP 코드 및 데이터 유형의 경우 AVX2는 AVX에 비해 8% 빠른 속도를 제공합니다.

```
2022-06-18 07:19:20.101478: I
tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
```

소스에서 TensorFlow를 빌드하려면 사용을 권장합니다 "Bazel". 우리는 현재 환경에 대해 셸 프롬프트에서 다음 명령을 실행하여 df, df-plugins, Bazel을 설치합니다.

```
yum install dnf
dnf install 'dnf-command(copr)'
dnf copr enable vbatts/bazel
dnf install bazel5
```

빌드 프로세스 중에 C++ 17 기능을 사용하려면 GCC 5 이상을 활성화해야 합니다. 이 기능은 RHEL에서 SCL(Software Collections Library)과 함께 제공합니다. 다음 명령은 RHEL 7.9 클러스터에 devt툴셋 및 GCC 11.2.1을 설치합니다.

```
subscription-manager repos --enable rhel-server-rhscl-7-rpms
yum install devtoolset-11-toolchain
yum install devtoolset-11-gcc-c++
yum update
scl enable devtoolset-11 bash
. /opt/rh/devtoolset-11/enable
```

마지막 두 명령은 /opt/rh/dev툴셋 -11/root/usr/bin/gcc(GCC 11.2.1)를 사용하는 dev툴셋 -11을 활성화합니다. 또한 git 버전이 1.8.3 이상인지 확인하십시오(RHEL 7.9와 함께 제공됨). 이를 참조하십시오 "기사" git를 2.24.1로 업데이트하는 경우.

최신 TensorFlow 마스터 저장소 를 이미 복제했다고 가정합니다. 그런 다음 "작업 공간" 파일을 사용하여 "작업 공간" 디렉토리를 만들어 AVX, AVX2 및 FMA를 사용하여 소스에서 TensorFlow를 구축합니다. '설정' 파일을 실행하고 올바른 Python 바이너리 위치를 지정합니다. "CUDA" GPU를 사용하지 않았기 때문에 테스트에 사용할 수 없습니다. 설정에 따라 '.bazelrc' 파일이 생성됩니다. 또한 파일을 편집하고 "build—define=no_hdfs_support=false"를 설정하여

HDFS 지원을 활성화했습니다. 절의 '.bazelrc'를 참조하십시오 ""주요 활용 사례별로 Python 스크립트,"" 설정 및 플래그의 전체 목록을 표시합니다.

```
./configure
bazel build -c opt --copt=-mavx --copt=-mavx2 --copt=-mfma --copt=
-mfpmath=both -k //tensorflow/tools/pip_package:build_pip_package
```

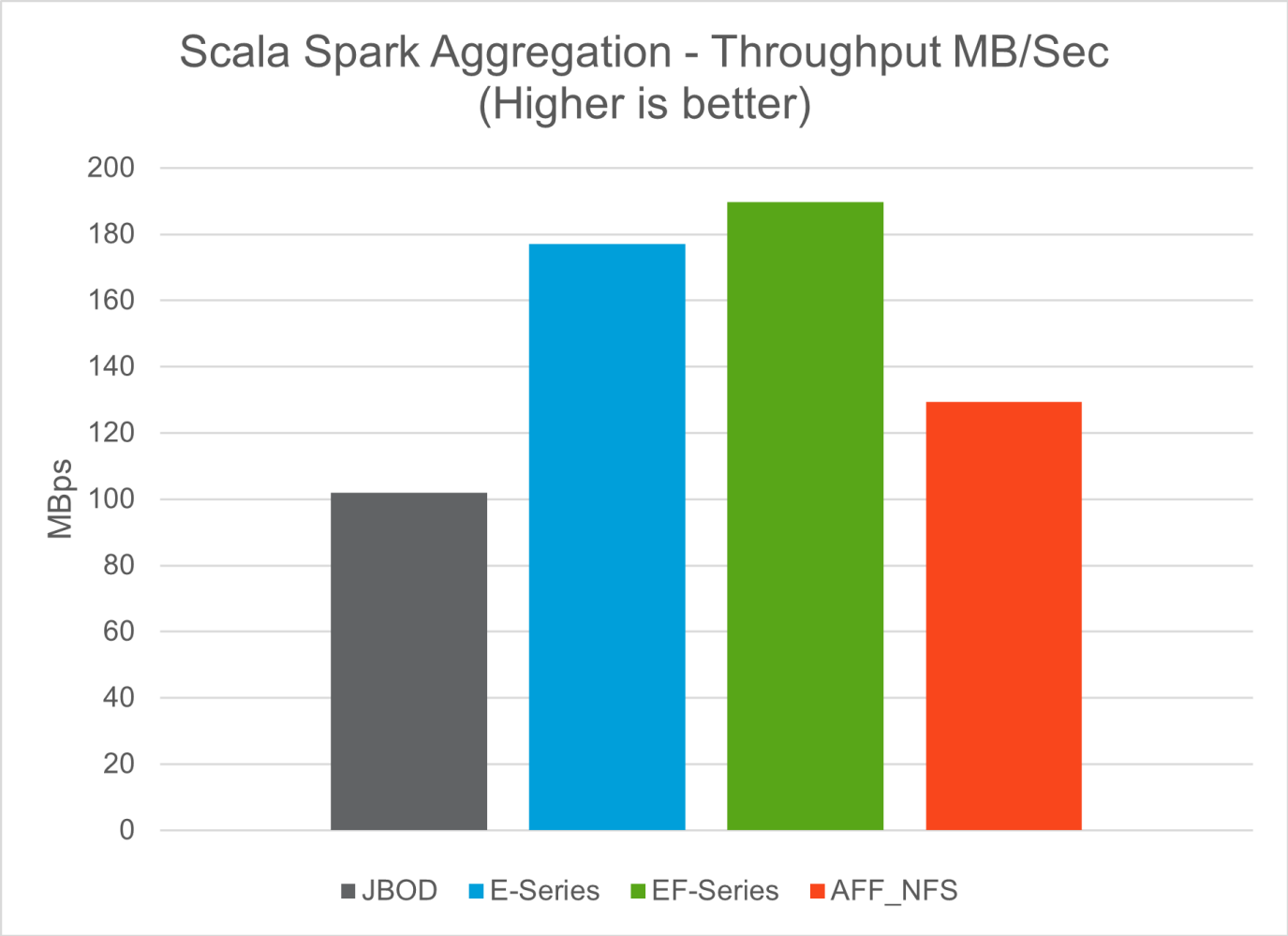
올바른 플래그를 사용하여 TensorFlow를 빌드한 후 다음 스크립트를 실행하여 Critio Display Ads 데이터 세트를 처리하고 DeepFM 모델을 교육하고 예측 점수의 Receiver Operating Characteristic Curve(ROC AUC) 아래에 있는 영역을 계산합니다.

```
(base) [root@n138 examples]# ~/anaconda3/bin/spark-submit
--master yarn
--executor-memory 15g
--executor-cores 1
--num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py
--data-dir file:///sparkdemo/tr-4570-data
> . /run_classification_criteo_spark_nfs.log 2>&1
```

10번의 교육 Epoch 후에 테스트 데이터 세트에 AUC 점수를 얻었습니다.

```
Epoch 1/10
125/125 - 7s - loss: 0.4976 - binary_crossentropy: 0.4974 - val_loss:
0.4629 - val_binary_crossentropy: 0.4624
Epoch 2/10
125/125 - 1s - loss: 0.3281 - binary_crossentropy: 0.3271 - val_loss:
0.5146 - val_binary_crossentropy: 0.5130
Epoch 3/10
125/125 - 1s - loss: 0.1948 - binary_crossentropy: 0.1928 - val_loss:
0.6166 - val_binary_crossentropy: 0.6144
Epoch 4/10
125/125 - 1s - loss: 0.1408 - binary_crossentropy: 0.1383 - val_loss:
0.7261 - val_binary_crossentropy: 0.7235
Epoch 5/10
125/125 - 1s - loss: 0.1129 - binary_crossentropy: 0.1102 - val_loss:
0.7961 - val_binary_crossentropy: 0.7934
Epoch 6/10
125/125 - 1s - loss: 0.0949 - binary_crossentropy: 0.0921 - val_loss:
0.9502 - val_binary_crossentropy: 0.9474
Epoch 7/10
125/125 - 1s - loss: 0.0778 - binary_crossentropy: 0.0750 - val_loss:
1.1329 - val_binary_crossentropy: 1.1301
Epoch 8/10
125/125 - 1s - loss: 0.0651 - binary_crossentropy: 0.0622 - val_loss:
1.3794 - val_binary_crossentropy: 1.3766
Epoch 9/10
125/125 - 1s - loss: 0.0555 - binary_crossentropy: 0.0527 - val_loss:
1.6115 - val_binary_crossentropy: 1.6087
Epoch 10/10
125/125 - 1s - loss: 0.0470 - binary_crossentropy: 0.0442 - val_loss:
1.6768 - val_binary_crossentropy: 1.6740
test AUC 0.6337
```

이전 사용 사례와 비슷한 방식으로 Spark 워크플로우 런타임을 다른 위치에 있는 데이터와 비교했습니다. 다음 그림은 Spark 워크플로 런타임에 대한 딥 러닝 CTR 예측을 비교한 것입니다.

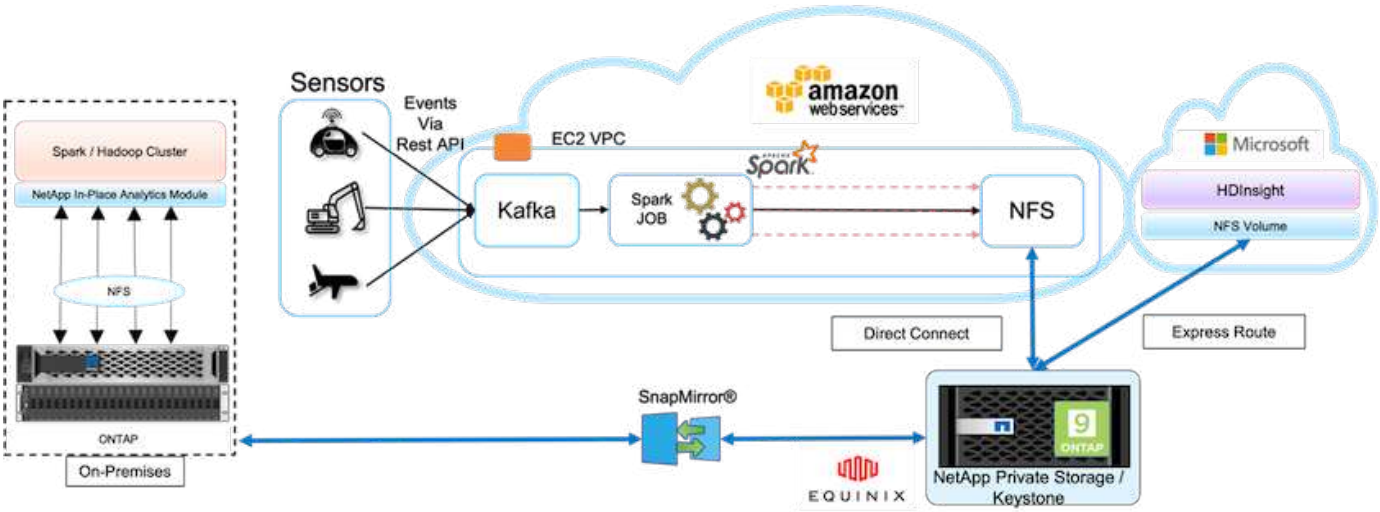



## 하이브리드 클라우드 솔루션

최신 엔터프라이즈 데이터 센터는 지속적인 데이터 관리 계층을 통해 여러 분산 인프라 환경을 연결하여 온프레미스 및/또는 멀티 퍼블릭 클라우드의 일관된 운영 모델을 유지하는 하이브리드 클라우드입니다. 하이브리드 클라우드를 최대한 활용하려면 데이터 변환 또는 애플리케이션 리팩토링 없이 사내 환경과 멀티 클라우드 환경 간에 데이터를 원활하게 이동할 수 있어야 합니다.

고객은 2차 스토리지를 클라우드로 이동하여 데이터 보호와 같은 사용 사례를 활용하거나 애플리케이션 개발, DevOps 등과 같은 비즈니스 크리티컬 워크로드를 클라우드로 이전함으로써 하이브리드 클라우드 여정을 시작한다고 했습니다. 그런 다음 더 중요한 워크로드로 이동합니다. 웹 및 콘텐츠 호스팅, DevOps 및 애플리케이션 개발, 데이터베이스, 분석, 컨테이너 앱 등이 가장 인기 있는 하이브리드 클라우드 워크로드 중 하나입니다. 엔터프라이즈 AI 프로젝트의 복잡성, 비용, 위험은 기존에 AI 채택의 실험 단계에서 운영 단계로 넘어왔던 것입니다.

NetApp 하이브리드 클라우드 솔루션을 사용하는 고객은 통합 보안, 데이터 거버넌스, 규정 준수 툴을 활용하여 분산 환경에서 데이터 및 워크플로우를 단일 제어판에서 관리할 수 있으며, 사용에 따른 총 소유 비용을 최적화할 수 있습니다. 다음 그림은 고객의 빅데이터 분석 데이터를 위한 멀티 클라우드 연결을 제공하는 클라우드 서비스 파트너의 예입니다.




이 시나리오에서는 여러 소스에서 AWS로 수신된 IoT 데이터가 NPS(NetApp 프라이빗 스토리지)의 중앙 위치에 저장됩니다. NPS 스토리지는 AWS와 Azure에 위치한 Spark 또는 Hadoop 클러스터에 연결되어 여러 클라우드에서 실행되는 빅데이터 분석 애플리케이션이 동일한 데이터에 액세스할 수 있도록 지원합니다. 이 활용 사례의 주요 요구사항과 과제는 다음과 같습니다.

- 고객은 여러 클라우드를 사용하여 동일한 데이터에 대한 분석 작업을 실행하려고 합니다.
- 다양한 센서와 허브를 통해 사내 환경, 클라우드 환경과 같은 다양한 소스로부터 데이터를 받아야 합니다.
- 솔루션은 효율적이고 비용 효율적이어야 합니다.
- 주된 과제는 서로 다른 사내 및 클라우드 환경 간에 하이브리드 분석 서비스를 제공하는 비용 효율적이고 효율적인 솔루션을 구축하는 것입니다.

NetApp의 데이터 보호 및 멀티 클라우드 연결 솔루션은 여러 하이퍼스케일러에 클라우드 분석 애플리케이션이 있다는 문제를 해결합니다. 위 그림과 같이 센서의 데이터가 스트리밍되어 Kafka를 통해 AWS Spark 클러스터로 수집됩니다. 이 데이터는 Equinix 데이터 센터 내의 클라우드 공급자 외부에 있는 NPS에 있는 NFS 공유에 저장됩니다.

NetApp NPS는 각각 Direct Connect와 Express Route 연결을 통해 Amazon AWS 및 Microsoft Azure에 연결되므로, 고객은 데이터 이동 없는 분석 모듈을 활용하여 Amazon 및 AWS 분석 클러스터 모두에서 데이터에 액세스할 수 있습니다. 따라서 사내 스토리지와 NPS 스토리지 모두 ONTAP 소프트웨어를 실행하기 때문에 "SnapMirror를 참조하십시오" NPS 데이터를 사내 클러스터에 미러링하여 사내 및 여러 클라우드 전반에 하이브리드 클라우드 분석 기능을 제공할 수 있습니다.

최적의 성능을 위해 일반적으로 여러 네트워크 인터페이스 및 직접 연결 또는 빠른 경로를 사용하여 클라우드 인스턴스의 데이터에 액세스하는 것이 좋습니다. NetApp은 다음을 비롯한 다른 Data Mover 솔루션을 보유하고 있습니다 "xCP" 및 "BlueXP 복사 및 동기화" 고객이 애플리케이션 인식, 보안 및 비용 효율적인 하이브리드 클라우드 스파크 클러스터를 구축할 수 있도록 지원합니다.

## 주요 활용 사례별로 **Python** 스크립트

다음 세 가지 Python 스크립트는 테스트를 거친 세 가지 주요 사용 사례에 해당합니다. 첫 번째는 'sentiment_analysis_sparklp.py'입니다.

```
# TR-4570 Refresh NLP testing by Rick Huang
from sys import argv
import os
```

```
import sparknlp
import pyspark.sql.functions as F
from sparknlp import Finisher
from pyspark.ml import Pipeline
from sparknlp.base import *
from sparknlp.annotator import *
from sparknlp.pretrained import PretrainedPipeline
from sparknlp import Finisher
# Start Spark Session with Spark NLP
spark = sparknlp.start()
print("Spark NLP version:")
print(sparknlp.version())
print("Apache Spark version:")
print(spark.version)
spark = sparknlp.SparkSession.builder \
     .master("yarn") \
     .appName("test_hdfs_read_write") \
     .config("spark.executor.cores", "1") \
     .config("spark.jars.packages", "com.johnsnowlabs.nlp:spark-
nlp_2.12:3.4.3")\
     .config('spark.executor.memory', '5gb') \
     .config('spark.executor.memoryOverhead','1000')\
     .config('spark.driver.memoryOverhead','1000')\
     .config("spark.sql.shuffle.partitions", "480")\
     .getOrCreate()
sc = spark.sparkContext
from pyspark.sql import SQLContext
sql = SQLContext(sc)
sqlContext = SQLContext(sc)
# Download pre-trained pipelines & sequence classifier
explain_pipeline_model = PretrainedPipeline('explain_document_dl',
lang='en').model#pipeline_sa =
PretrainedPipeline("classifierdl_bertwiki_finance_sentiment_pipeline",
lang="en")
# pipeline_finbert =
BertForSequenceClassification.loadSavedModel('/sparkusecase/bert_sequence_
classifier_finbert_en_3', spark)
sequenceClassifier = BertForSequenceClassification \
        .pretrained('bert_sequence_classifier_finbert', 'en') \
        .setInputCols(['token', 'document']) \
        .setOutputCol('class') \
        .setCaseSensitive(True) \
        .setMaxSentenceLength(512)
def process_sentence_df(data):
    # Pre-process: begin
    print("1. Begin DataFrame pre-processing...\n")
```

```
    print(f"\n\t2. Attaching DocumentAssembler Transformer to the
pipeline")
    documentAssembler = DocumentAssembler() \
        .setInputCol("text") \
        .setOutputCol("document") \
        .setCleanupMode("inplace_full")
        #.setCleanupMode("shrink", "inplace_full")
    doc_df = documentAssembler.transform(data)
    doc_df.printSchema()
    doc_df.show(truncate=50)
    # Pre-process: get rid of  blank lines
    clean_df = doc_df.withColumn("tmp", F.explode("document")) \
        .select("tmp.result").where("tmp.end !=
-1").withColumnRenamed("result", "text").dropna()
    print("[OK!] DataFrame after initial cleanup:\n")
    clean_df.printSchema()
    clean_df.show(truncate=80)
    # for FinBERT
    tokenizer = Tokenizer() \
        .setInputCols(['document']) \
        .setOutputCol('token')
    print(f"\n\t3. Attaching Tokenizer Annotator to the pipeline")
    pipeline_finbert = Pipeline(stages=[
        documentAssembler,
        tokenizer,
        sequenceClassifier
        ])
    # Use Finisher() & construct PySpark ML pipeline
    finisher = Finisher().setInputCols(["token", "lemma", "pos",
"entities"])
    print(f"\n\t4. Attaching Finisher Transformer to the pipeline")
    pipeline_ex = Pipeline() \
        .setStages([
           explain_pipeline_model,
           finisher
           ])
    print("\n\t\t\t ---- Pipeline Built Successfully ----")
    # Loading pipelines to annotate
    #result_ex_df = pipeline_ex.transform(clean_df)
    ex_model = pipeline_ex.fit(clean_df)
    annotations_finished_ex_df = ex_model.transform(clean_df)
    # result_sa_df = pipeline_sa.transform(clean_df)
    result_finbert_df = pipeline_finbert.fit(clean_df).transform(clean_df)
    print("\n\t\t\t ----Document Explain, Sentiment Analysis & FinBERT
Pipeline Fitted Successfully ----")
    # Check the result entities
```

```
    print("[OK!] Simple explain ML pipeline result:\n")
    annotations_finished_ex_df.printSchema()
    annotations_finished_ex_df.select('text',
'finished_entities').show(truncate=False)
    # Check the result sentiment from FinBERT
    print("[OK!] Sentiment Analysis FinBERT pipeline result:\n")
    result_finbert_df.printSchema()
    result_finbert_df.select('text', 'class.result').show(80, False)
    sentiment_stats(result_finbert_df)
    return
def sentiment_stats(finbert_df):
    result_df = finbert_df.select('text', 'class.result')
    sa_df = result_df.select('result')
    sa_df.groupBy('result').count().show()
    # total_lines = result_clean_df.count()
    # num_neutral = result_clean_df.where(result_clean_df.result ==
['neutral']).count()
    # num_positive = result_clean_df.where(result_clean_df.result ==
['positive']).count()
    # num_negative = result_clean_df.where(result_clean_df.result ==
['negative']).count()
    # print(f"\nRatio of neutral sentiment = {num_neutral/total_lines}")
    # print(f"Ratio of positive sentiment = {num_positive / total_lines}")
    # print(f"Ratio of negative sentiment = {num_negative /
total_lines}\n")
    return
def process_input_file(file_name):
    # Turn input file to Spark DataFrame
    print("START processing input file...")
    data_df = spark.read.text(file_name)
    data_df.show()
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    output_df.printSchema()
    return output_dfdef process_local_dir(directory):
    filelist = []
    for subdir, dirs, files in os.walk(directory):
        for filename in files:
            filepath = subdir + os.sep + filename
            print("[OK!] Will process the following files:")
            if filepath.endswith(".txt"):
                print(filepath)
                filelist.append(filepath)
    return filelist
def process_local_dir_or_file(dir_or_file):
    numfiles = 0
```

```
    if os.path.isfile(dir_or_file):
        input_df = process_input_file(dir_or_file)
        print("Obtained input_df.")
        process_sentence_df(input_df)
        print("Processed input_df")
        numfiles += 1
    else:
        filelist = process_local_dir(dir_or_file)
        for file in filelist:
            input_df = process_input_file(file)
            process_sentence_df(input_df)
            numfiles += 1
    return numfiles
def process_hdfs_dir(dir_name):
    # Turn input files to Spark DataFrame
    print("START processing input HDFS directory...")
    data_df = spark.read.option("recursiveFileLookup",
"true").text(dir_name)
    data_df.show()
    print("[DEBUG] total lines in data_df = ", data_df.count())
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    print("[DEBUG] output_df looks like: \n")
    output_df.show(40, False)
    print("[DEBUG] HDFS dir resulting data_df schema: \n")
    output_df.printSchema()
    process_sentence_df(output_df)
    print("Processed HDFS directory: ", dir_name)
    returnif __name__ == '__main__':
    try:
        if len(argv) == 2:
            print("Start processing input...\n")
    except:
        print("[ERROR] Please enter input text file or path to
process!\n")
        exit(1)
    # This is for local file, not hdfs:
    numfiles = process_local_dir_or_file(str(argv[1]))
    # For HDFS single file & directory:
    input_df = process_input_file(str(argv[1]))
    print("Obtained input_df.")
    process_sentence_df(input_df)
    print("Processed input_df")
    numfiles += 1
    # For HDFS directory of subdirectories of files:
    input_parse_list = str(argv[1]).split('/')
```

```
    print(input_parse_list)
    if input_parse_list[-2:-1] == ['Transcripts']:
        print("Start processing HDFS directory: ", str(argv[1]))
        process_hdfs_dir(str(argv[1]))
    print(f"[OK!] All done. Number of files processed = {numfiles}")
```

두 번째 스크립트는 keras_spark_horovod_rossmann_estimator.py입니다.

```
# Copyright 2022 NetApp, Inc.
# Authored by Rick Huang
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
==========================================================================
====
# The below code was modified from: https://www.kaggle.com/c/rossmann-
store-sales
import argparse
import datetime
import os
import sys
from distutils.version import LooseVersion
import pyspark.sql.types as T
import pyspark.sql.functions as F
from pyspark import SparkConf, Row
from pyspark.sql import SparkSession
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.layers import Input, Embedding, Concatenate, Dense,
Flatten, Reshape, BatchNormalization, Dropout
import horovod.spark.keras as hvd
from horovod.spark.common.backend import SparkBackend
from horovod.spark.common.store import Store
from horovod.tensorflow.keras.callbacks import BestModelCheckpoint
parser = argparse.ArgumentParser(description='Horovod Keras Spark Rossmann
```

```
Estimator Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--master',
                    help='spark cluster to use for training. If set to
None, uses current default cluster. Cluster'
                         'should be set up to provide a Spark task per
multiple CPU cores, or per GPU, e.g. by'
                         'supplying `-c <NUM_GPUS>` in Spark Standalone
mode')
parser.add_argument('--num-proc', type=int,
                    help='number of worker processes for training,
default: `spark.default.parallelism`')
parser.add_argument('--learning_rate', type=float, default=0.0001,
                    help='initial learning rate')
parser.add_argument('--batch-size', type=int, default=100,
                    help='batch size')
parser.add_argument('--epochs', type=int, default=100,
                    help='number of epochs to train')
parser.add_argument('--sample-rate', type=float,
                    help='desired sampling rate. Useful to set to low
number (e.g. 0.01) to make sure that '
                         'end-to-end process works')
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
parser.add_argument('--local-submission-csv', default='submission.csv',
                    help='output submission predictions CSV')
parser.add_argument('--local-checkpoint-file', default='checkpoint',
                    help='model checkpoint')
parser.add_argument('--work-dir', default='/tmp',
                    help='temporary working directory to write
intermediate files (prefix with hdfs:// to use HDFS)')
if __name__ == '__main__':
    args = parser.parse_args()
    # ================ #
    # DATA PREPARATION #
    # ================ #
    print('================')
    print('Data preparation')
    print('================')
    # Create Spark session for data preparation.
    conf = SparkConf() \
        .setAppName('Keras Spark Rossmann Estimator Example') \
        .set('spark.sql.shuffle.partitions', '480') \
        .set("spark.executor.cores", "1") \
```

```
        .set('spark.executor.memory', '5gb') \
        .set('spark.executor.memoryOverhead','1000')\
        .set('spark.driver.memoryOverhead','1000')
    if args.master:
        conf.setMaster(args.master)
    elif args.num_proc:
        conf.setMaster('local[{}]'.format(args.num_proc))
    spark = SparkSession.builder.config(conf=conf).getOrCreate()
    train_csv = spark.read.csv('%s/train.csv' % args.data_dir,
header=True)
    test_csv = spark.read.csv('%s/test.csv' % args.data_dir, header=True)
    store_csv = spark.read.csv('%s/store.csv' % args.data_dir,
header=True)
    store_states_csv = spark.read.csv('%s/store_states.csv' %
args.data_dir, header=True)
    state_names_csv = spark.read.csv('%s/state_names.csv' % args.data_dir,
header=True)
    google_trend_csv = spark.read.csv('%s/googletrend.csv' %
args.data_dir, header=True)
    weather_csv = spark.read.csv('%s/weather.csv' % args.data_dir,
header=True)
    def expand_date(df):
        df = df.withColumn('Date', df.Date.cast(T.DateType()))
        return df \
            .withColumn('Year', F.year(df.Date)) \
            .withColumn('Month', F.month(df.Date)) \
            .withColumn('Week', F.weekofyear(df.Date)) \
            .withColumn('Day', F.dayofmonth(df.Date))
    def prepare_google_trend():
        # Extract week start date and state.
        google_trend_all = google_trend_csv \
            .withColumn('Date', F.regexp_extract(google_trend_csv.week,
'(.*?) -', 1)) \
            .withColumn('State', F.regexp_extract(google_trend_csv.file,
'Rossmann_DE_(.*)', 1))
        # Map state NI -> HB,NI to align with other data sources.
        google_trend_all = google_trend_all \
            .withColumn('State', F.when(google_trend_all.State == 'NI',
'HB,NI').otherwise(google_trend_all.State))
        # Expand dates.
        return expand_date(google_trend_all)
    def add_elapsed(df, cols):
        def add_elapsed_column(col, asc):
            def fn(rows):
                last_store, last_date = None, None
                for r in rows:
```

```
                    if last_store != r.Store:
                        last_store = r.Store
                        last_date = r.Date
                    if r[col]:
                        last_date = r.Date
                    fields = r.asDict().copy()
                    fields[('After' if asc else 'Before') + col] = (r.Date
- last_date).days
                    yield Row(**fields)
            return fn
        df = df.repartition(df.Store)
        for asc in [False, True]:
            sort_col = df.Date.asc() if asc else df.Date.desc()
            rdd = df.sortWithinPartitions(df.Store.asc(), sort_col).rdd
            for col in cols:
                rdd = rdd.mapPartitions(add_elapsed_column(col, asc))
            df = rdd.toDF()
        return df
    def prepare_df(df):
        num_rows = df.count()
        # Expand dates.
        df = expand_date(df)
        df = df \
            .withColumn('Open', df.Open != '0') \
            .withColumn('Promo', df.Promo != '0') \
            .withColumn('StateHoliday', df.StateHoliday != '0') \
            .withColumn('SchoolHoliday', df.SchoolHoliday != '0')
        # Merge in store information.
        store = store_csv.join(store_states_csv, 'Store')
        df = df.join(store, 'Store')
        # Merge in Google Trend information.
        google_trend_all = prepare_google_trend()
        df = df.join(google_trend_all, ['State', 'Year',
'Week']).select(df['*'], google_trend_all.trend)
        # Merge in Google Trend for whole Germany.
        google_trend_de = google_trend_all[google_trend_all.file ==
'Rossmann_DE'].withColumnRenamed('trend', 'trend_de')
        df = df.join(google_trend_de, ['Year', 'Week']).select(df['*'],
google_trend_de.trend_de)
        # Merge in weather.
        weather = weather_csv.join(state_names_csv, weather_csv.file ==
state_names_csv.StateName)
        df = df.join(weather, ['State', 'Date'])
        # Fix null values.
        df = df \
            .withColumn('CompetitionOpenSinceYear',
```

```
F.coalesce(df.CompetitionOpenSinceYear, F.lit(1900))) \
            .withColumn('CompetitionOpenSinceMonth',
F.coalesce(df.CompetitionOpenSinceMonth, F.lit(1))) \
            .withColumn('Promo2SinceYear', F.coalesce(df.Promo2SinceYear,
F.lit(1900))) \
            .withColumn('Promo2SinceWeek', F.coalesce(df.Promo2SinceWeek,
F.lit(1)))
        # Days & months competition was open, cap to 2 years.
        df = df.withColumn('CompetitionOpenSince',
                           F.to_date(F.format_string('%s-%s-15',
df.CompetitionOpenSinceYear,

df.CompetitionOpenSinceMonth)))
        df = df.withColumn('CompetitionDaysOpen',
                           F.when(df.CompetitionOpenSinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(360 *
2), F.datediff(df.Date, df.CompetitionOpenSince))))
                           .otherwise(0))
        df = df.withColumn('CompetitionMonthsOpen',
(df.CompetitionDaysOpen / 30).cast(T.IntegerType()))
        # Days & weeks of promotion, cap to 25 weeks.
        df = df.withColumn('Promo2Since',
                           F.expr('date_add(format_string("%s-01-01",
Promo2SinceYear), (cast(Promo2SinceWeek as int) - 1) * 7)'))
        df = df.withColumn('Promo2Days',
                           F.when(df.Promo2SinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(25 *
7), F.datediff(df.Date, df.Promo2Since))))
                           .otherwise(0))
        df = df.withColumn('Promo2Weeks', (df.Promo2Days /
7).cast(T.IntegerType()))
        # Check that we did not lose any rows through inner joins.
        assert num_rows == df.count(), 'lost rows in joins'
        return df
    def build_vocabulary(df, cols):
        vocab = {}
        for col in cols:
            values = [r[0] for r in df.select(col).distinct().collect()]
            col_type = type([x for x in values if x is not None][0])
            default_value = col_type()
            vocab[col] = sorted(values, key=lambda x: x or default_value)
        return vocab
    def cast_columns(df, cols):
        for col in cols:
            df = df.withColumn(col,
F.coalesce(df[col].cast(T.FloatType()), F.lit(0.0)))
```

```
        return df
    def lookup_columns(df, vocab):
        def lookup(mapping):
            def fn(v):
                return mapping.index(v)
            return F.udf(fn, returnType=T.IntegerType())
        for col, mapping in vocab.items():
            df = df.withColumn(col, lookup(mapping)(df[col]))
        return df
    if args.sample_rate:
        train_csv = train_csv.sample(withReplacement=False,
fraction=args.sample_rate)
        test_csv = test_csv.sample(withReplacement=False,
fraction=args.sample_rate)
    # Prepare data frames from CSV files.
    train_df = prepare_df(train_csv).cache()
    test_df = prepare_df(test_csv).cache()
    # Add elapsed times from holidays & promos, the data spanning training
& test datasets.
    elapsed_cols = ['Promo', 'StateHoliday', 'SchoolHoliday']
    elapsed = add_elapsed(train_df.select('Date', 'Store', *elapsed_cols)
                          .unionAll(test_df.select('Date', 'Store',
*elapsed_cols)),
                          elapsed_cols)
    # Join with elapsed times.
    train_df = train_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(train_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    test_df = test_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(test_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    # Filter out zero sales.
    train_df = train_df.filter(train_df.Sales > 0)
    print('===================')
    print('Prepared data frame')
    print('===================')
    train_df.show()
    categorical_cols = [
        'Store', 'State', 'DayOfWeek', 'Year', 'Month', 'Day', 'Week',
'CompetitionMonthsOpen', 'Promo2Weeks', 'StoreType',
        'Assortment', 'PromoInterval', 'CompetitionOpenSinceYear',
'Promo2SinceYear', 'Events', 'Promo',
        'StateHoliday', 'SchoolHoliday'
    ]
```

```
    continuous_cols = [
        'CompetitionDistance', 'Max_TemperatureC', 'Mean_TemperatureC',
'Min_TemperatureC', 'Max_Humidity',
        'Mean_Humidity', 'Min_Humidity', 'Max_Wind_SpeedKm_h',
'Mean_Wind_SpeedKm_h', 'CloudCover', 'trend', 'trend_de',
        'BeforePromo', 'AfterPromo', 'AfterStateHoliday',
'BeforeStateHoliday', 'BeforeSchoolHoliday', 'AfterSchoolHoliday'
    ]
    all_cols = categorical_cols + continuous_cols
    # Select features.
    train_df = train_df.select(*(all_cols + ['Sales', 'Date'])).cache()
    test_df = test_df.select(*(all_cols + ['Id', 'Date'])).cache()
    # Build vocabulary of categorical columns.
    vocab = build_vocabulary(train_df.select(*categorical_cols)

.unionAll(test_df.select(*categorical_cols)).cache(),
                              categorical_cols)
    # Cast continuous columns to float & lookup categorical columns.
    train_df = cast_columns(train_df, continuous_cols + ['Sales'])
    train_df = lookup_columns(train_df, vocab)
    test_df = cast_columns(test_df, continuous_cols)
    test_df = lookup_columns(test_df, vocab)
    # Split into training & validation.
    # Test set is in 2015, use the same period in 2014 from the training
set as a validation set.
    test_min_date = test_df.agg(F.min(test_df.Date)).collect()[0][0]
    test_max_date = test_df.agg(F.max(test_df.Date)).collect()[0][0]
    one_year = datetime.timedelta(365)
    train_df = train_df.withColumn('Validation',
                                   (train_df.Date > test_min_date -
one_year) & (train_df.Date <= test_max_date - one_year))
    # Determine max Sales number.
    max_sales = train_df.agg(F.max(train_df.Sales)).collect()[0][0]
    # Convert Sales to log domain
    train_df = train_df.withColumn('Sales', F.log(train_df.Sales))
    print('===================================')
    print('Data frame with transformed columns')
    print('===================================')
    train_df.show()
    print('================')
    print('Data frame sizes')
    print('================')
    train_rows = train_df.filter(~train_df.Validation).count()
    val_rows = train_df.filter(train_df.Validation).count()
    test_rows = test_df.count()
    print('Training: %d' % train_rows)
```

```
    print('Validation: %d' % val_rows)
    print('Test: %d' % test_rows)
    # ============== #
    # MODEL TRAINING #
    # ============== #
    print('==============')
    print('Model training')
    print('==============')
    def exp_rmspe(y_true, y_pred):
        """Competition evaluation metric, expects logarithic inputs."""
        pct = tf.square((tf.exp(y_true) - tf.exp(y_pred)) /
tf.exp(y_true))
        # Compute mean excluding stores with zero denominator.
        x = tf.reduce_sum(tf.where(y_true > 0.001, pct,
tf.zeros_like(pct)))
        y = tf.reduce_sum(tf.where(y_true > 0.001, tf.ones_like(pct),
tf.zeros_like(pct)))
        return tf.sqrt(x / y)
    def act_sigmoid_scaled(x):
        """Sigmoid scaled to logarithm of maximum sales scaled by 20%."""
        return tf.nn.sigmoid(x) * tf.math.log(max_sales) * 1.2
    CUSTOM_OBJECTS = {'exp_rmspe': exp_rmspe,
                      'act_sigmoid_scaled': act_sigmoid_scaled}
    # Disable GPUs when building the model to prevent memory leaks
    if LooseVersion(tf.__version__) >= LooseVersion('2.0.0'):
        # See https://github.com/tensorflow/tensorflow/issues/33168
        os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
    else:

K.set_session(tf.Session(config=tf.ConfigProto(device_count={'GPU': 0})))
    # Build the model.
    inputs = {col: Input(shape=(1,), name=col) for col in all_cols}
    embeddings = [Embedding(len(vocab[col]), 10, input_length=1,
name='emb_' + col)(inputs[col])
                  for col in categorical_cols]
    continuous_bn = Concatenate()([Reshape((1, 1), name='reshape_' +
col)(inputs[col])
                                   for col in continuous_cols])
    continuous_bn = BatchNormalization()(continuous_bn)
    x = Concatenate()(embeddings + [continuous_bn])
    x = Flatten()(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
```

```
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(500, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation=act_sigmoid_scaled)(x)
    model = tf.keras.Model([inputs[f] for f in all_cols], output)
    model.summary()
    opt = tf.keras.optimizers.Adam(lr=args.learning_rate, epsilon=1e-3)
    # Checkpoint callback to specify options for the returned Keras model
    ckpt_callback = BestModelCheckpoint(monitor='val_loss', mode='auto',
save_freq='epoch')
    # Horovod: run training.
    store = Store.create(args.work_dir)
    backend = SparkBackend(num_proc=args.num_proc,
                           stdout=sys.stdout, stderr=sys.stderr,
                           prefix_output_with_timestamp=True)
    keras_estimator = hvd.KerasEstimator(backend=backend,
                                         store=store,
                                         model=model,
                                         optimizer=opt,
                                         loss='mae',
                                         metrics=[exp_rmspe],
                                         custom_objects=CUSTOM_OBJECTS,
                                         feature_cols=all_cols,
                                         label_cols=['Sales'],
                                         validation='Validation',
                                         batch_size=args.batch_size,
                                         epochs=args.epochs,
                                         verbose=2,

checkpoint_callback=ckpt_callback)
    keras_model =
keras_estimator.fit(train_df).setOutputCols(['Sales_output'])
    history = keras_model.getHistory()
    best_val_rmspe = min(history['val_exp_rmspe'])
    print('Best RMSPE: %f' % best_val_rmspe)
    # Save the trained model.
    keras_model.save(args.local_checkpoint_file)
    print('Written checkpoint to %s' % args.local_checkpoint_file)
    # ================ #
    # FINAL PREDICTION #
    # ================ #
    print('================')
    print('Final prediction')
    print('================')
    pred_df=keras_model.transform(test_df)
```

```
    pred_df.printSchema()
    pred_df.show(5)
    # Convert from log domain to real Sales numbers
    pred_df=pred_df.withColumn('Sales_pred', F.exp(pred_df.Sales_output))
    submission_df = pred_df.select(pred_df.Id.cast(T.IntegerType()),
pred_df.Sales_pred).toPandas()
    submission_df.sort_values(by=['Id']).to_csv(args.local_submission_csv,
index=False)
    print('Saved predictions to %s' % args.local_submission_csv)
    spark.stop()
```

세 번째 스크립트는 run_classification_criteo_spark.py입니다.

```
import tempfile, string, random, os, uuid
import argparse, datetime, sys, shutil
import csv
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from pyspark import SparkContext
from pyspark.sql import SparkSession, SQLContext, Row, DataFrame
from pyspark.mllib import linalg as mllib_linalg
from pyspark.mllib.linalg import SparseVector as mllibSparseVector
from pyspark.mllib.linalg import VectorUDT as mllibVectorUDT
from pyspark.mllib.linalg import Vector as mllibVector, Vectors as
mllibVectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.ml import linalg as ml_linalg
from pyspark.ml.linalg import VectorUDT as mlVectorUDT
from pyspark.ml.linalg import SparseVector as mlSparseVector
from pyspark.ml.linalg import Vector as mlVector, Vectors as mlVectors
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import OneHotEncoder
from math import log
from math import exp  # exp(-t) = e^-t
from operator import add
from pyspark.sql.functions import udf, split, lit
from pyspark.sql.functions import size, sum as sqlsum
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql.types import ArrayType, StructType, StructField,
LongType, StringType, IntegerType, FloatType
from pyspark.sql.functions import explode, col, log, when
from collections import defaultdict
```

```
import pandas as pd
import pyspark.pandas as ps
from sklearn.metrics import log_loss, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from deepctr.models import DeepFM
from deepctr.feature_column import SparseFeat, DenseFeat,
get_feature_names
spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
# spark.conf.set("spark.sql.execution.arrow.enabled", "true") # deprecated
print("Apache Spark version:")
print(spark.version)
sc = spark.sparkContext
sqlContext = SQLContext(sc)
parser = argparse.ArgumentParser(description='Spark DCN CTR Prediction
Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
def process_input_file(file_name, sparse_feat, dense_feat):
    # Need this preprocessing to turn Criteo raw file into CSV:
    print("START processing input file...")
    # only convert the file ONCE
    # sample = open(file_name)
    # sample = '\n'.join([str(x.replace('\n', '').replace('\t', ',')) for
x in sample])
    # # Add header in data file and save as CSV
    # header = ','.join(str(x) for x in (['label'] + dense_feat +
sparse_feat))
    # with open('/sparkdemo/tr-4570-data/ctr_train.csv', mode='w',
encoding="utf-8") as f:
    #     f.write(header + '\n' + sample)
```

```
    #     f.close()
    # print("Raw training file processed and saved as CSV: ", f.name)
    raw_df = sqlContext.read.option("header", True).csv(file_name)
    raw_df.show(5, False)
    raw_df.printSchema()
    # convert columns I1 to I13 from string to integers
    conv_df = raw_df.select(col('label').cast("double"),
                            *(col(i).cast("float").alias(i) for i in
raw_df.columns if i in dense_feat),
                            *(col(c) for c in raw_df.columns if c in
sparse_feat))
    print("Schema of raw_df with integer columns type changed:")
    conv_df.printSchema()
    # result_pdf = conv_df.select("*").toPandas()
    tmp_df = conv_df.na.fill(0, dense_feat)
    result_df = tmp_df.na.fill('-1', sparse_feat)
    result_df.show()
    return result_df
if __name__ == "__main__":
    args = parser.parse_args()
    # Pandas read CSV
    # data = pd.read_csv('%s/criteo_sample.txt' % args.data_dir)
    # print("Obtained Pandas df.")
    dense_features = ['I' + str(i) for i in range(1, 14)]
    sparse_features = ['C' + str(i) for i in range(1, 27)]
    # Spark read CSV
    # process_input_file('%s/train.txt' % args.data_dir, sparse_features,
dense_features) # run only ONCE
    spark_df = process_input_file('%s/data.txt' % args.data_dir,
sparse_features, dense_features) # sample data
    # spark_df = process_input_file('%s/ctr_train.csv' % args.data_dir,
sparse_features, dense_features)
    print("Obtained Spark df and filled in missing features.")
    data = spark_df
    # Pandas
    #data[sparse_features] = data[sparse_features].fillna('-1', )
    #data[dense_features] = data[dense_features].fillna(0, )
    target = ['label']
    label_npa = data.select("label").toPandas().to_numpy()
    print("label numPy array has length = ", len(label_npa)) # 45,840,617
w/ 11GB dataset
    label_npa.ravel()
    label_npa.reshape(len(label_npa), )
    # 1.Label Encoding for sparse features,and do simple Transformation
for dense features
    print("Before LabelEncoder():")
```

```
    data.printSchema()  # label: float (nullable = true)
    for feat in sparse_features:
        lbe = LabelEncoder()
        tmp_pdf = data.select(feat).toPandas().to_numpy()
        tmp_ndarray = lbe.fit_transform(tmp_pdf)
        print("After LabelEncoder(), tmp_ndarray[0] =", tmp_ndarray[0])
        # print("Data tmp PDF after lbe transformation, the output ndarray
has length = ", len(tmp_ndarray)) # 45,840,617 for 11GB dataset
        tmp_ndarray.ravel()
        tmp_ndarray.reshape(len(tmp_ndarray), )
        out_ndarray = np.column_stack([label_npa, tmp_ndarray])
        pdf = pd.DataFrame(out_ndarray, columns=['label', feat])
        s_df = spark.createDataFrame(pdf)
        s_df.printSchema() # label: double (nullable = true)
        print("Before joining data df with s_df, s_df example rows:")
        s_df.show(1, False)
        data = data.drop(feat).join(s_df, 'label').drop('label')
        print("After LabelEncoder(), data df example rows:")
        data.show(1, False)
        print("Finished processing sparse_features: ", feat)
    print("Data DF after label encoding: ")
    data.show()
    data.printSchema()
    mms = MinMaxScaler(feature_range=(0, 1))
    # data[dense_features] = mms.fit_transform(data[dense_features]) # for
Pandas df
    tmp_pdf = data.select(dense_features).toPandas().to_numpy()
    tmp_ndarray = mms.fit_transform(tmp_pdf)
    tmp_ndarray.ravel()
    tmp_ndarray.reshape(len(tmp_ndarray), len(tmp_ndarray[0]))
    out_ndarray = np.column_stack([label_npa, tmp_ndarray])
    pdf = pd.DataFrame(out_ndarray, columns=['label'] + dense_features)
    s_df = spark.createDataFrame(pdf)
    s_df.printSchema()
    data.drop(*dense_features).join(s_df, 'label').drop('label')
    print("Finished processing dense_features: ", dense_features)
    print("Data DF after MinMaxScaler: ")
    data.show()

    # 2.count #unique features for each sparse field,and record dense
feature field name
    fixlen_feature_columns = [SparseFeat(feat,
vocabulary_size=data.select(feat).distinct().count() + 1, embedding_dim=4)
                              for i, feat in enumerate(sparse_features)] +
\
                             [DenseFeat(feat, 1, ) for feat in
```

```
dense_features]
    dnn_feature_columns = fixlen_feature_columns
    linear_feature_columns = fixlen_feature_columns
    feature_names = get_feature_names(linear_feature_columns +
dnn_feature_columns)
    # 3.generate input data for model
    # train, test = train_test_split(data.toPandas(), test_size=0.2,
random_state=2020) # Pandas; might hang for 11GB data
    train, test = data.randomSplit(weights=[0.8, 0.2], seed=200)
    print("Training dataset size = ", train.count())
    print("Testing dataset size = ", test.count())
    # Pandas:
    # train_model_input = {name: train[name] for name in feature_names}
    # test_model_input = {name: test[name] for name in feature_names}
    # Spark DF:
    train_model_input = {}
    test_model_input = {}
    for name in feature_names:
        if name.startswith('I'):
            tr_pdf = train.select(name).toPandas()
            train_model_input[name] = pd.to_numeric(tr_pdf[name])
            ts_pdf = test.select(name).toPandas()
            test_model_input[name] = pd.to_numeric(ts_pdf[name])
    # 4.Define Model,train,predict and evaluate
    model = DeepFM(linear_feature_columns, dnn_feature_columns,
task='binary')
    model.compile("adam", "binary_crossentropy",
                  metrics=['binary_crossentropy'], )
    lb_pdf = train.select(target).toPandas()
    history = model.fit(train_model_input,
pd.to_numeric(lb_pdf['label']).values,
                        batch_size=256, epochs=10, verbose=2,
validation_split=0.2, )
    pred_ans = model.predict(test_model_input, batch_size=256)
    print("test LogLoss",
round(log_loss(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
    print("test AUC",
round(roc_auc_score(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
```

## 결론

이 문서에서는 Apache Spark 아키텍처, 고객 사용 사례 및 빅데이터, 최신 분석, AI, ML 및 DL과 관련된 NetApp 스토리지 포트폴리오에 대해 설명합니다. 업계 표준 벤치마킹 툴 및 고객

요구사항을 기반으로 한 성능 검증 테스트에서 NetApp Spark 솔루션은 네이티브 Hadoop 시스템에 비해 뛰어난 성능을 입증했습니다. 이 보고서에 제공된 고객 사용 사례 및 성능 결과를 조합하여 배포할 적절한 Spark 솔루션을 선택할 수 있습니다.

## 추가 정보를 찾을 수 있는 위치

이 TR에 사용된 참조 자료는 다음과 같습니다.

- Apache Spark 아키텍처 및 구성 요소

  "http://spark.apache.org/docs/latest/cluster-overview.html"

- Apache Spark 사용 사례

  "https://www.qubole.com/blog/big-data/apache-spark-use-cases/"

- Apache 당면 과제

  "http://www.infoworld.com/article/2897287/big-data/5-reasons-to-turn-to-spark-for-big-data-analytics.html"

- 스파크 NLP

  "https://www.johnsnowlabs.com/spark-nlp/"

- 베르

  "https://arxiv.org/abs/1810.04805"

- Deep and Cross Network for Ad Predictions를 클릭합니다

  "https://arxiv.org/abs/1708.05123"

- FlexGroup

  "http://www.netapp.com/us/media/tr-4557.pdf"

- 스트리밍 ETL

  "https://www.infoq.com/articles/apache-spark-streaming"

- Hadoop용 NetApp E-Series 솔루션

  "https://www.netapp.com/media/16420-tr-3969.pdf"

- NetApp AI와 고객 커뮤니케이션을 통한 감정 분석

  "https://docs.netapp.com/us-en/netapp-solutions/pdfs/sidebar/Sentiment_analysis_with_NetApp_AI.pdf"

- NetApp의 최신 데이터 분석 솔루션

  "https://docs.netapp.com/us-en/netapp-solutions/data-analytics/index.html"

- SnapMirror를 참조하십시오

"https://docs.netapp.com/us-en/ontap/data-protection/snapmirror-replication-concept.html"

- xCP

  https://mysupport.netapp.com/documentation/docweb/index.html?productID=63942&language=en-US

- BlueXP 복사 및 동기화

  "https://cloud.netapp.com/cloud-sync-service"

- DataOps 툴킷

  "https://github.com/NetApp/netapp-dataops-toolkit"

# 인공 지능에 빅 데이터 분석 데이터 활용

## **TR-4732:** 인공 지능에 대한 빅 데이터 분석 데이터

NetApp의 Karthikeyan Nagalingam입니다

이 문서에서는 빅데이터 분석 데이터와 HPC 데이터를 AI로 이동하는 방법에 대해 설명합니다. AI는 NFS 엑스포트를 통해 NFS 데이터를 처리하는 한편, 고객은 HDFS, Blob 또는 S3 스토리지와 같은 빅데이터 분석 플랫폼과 GPFS와 같은 HPC 플랫폼에 AI 데이터를 저장할 수 있습니다. 이 문서에서는 NetApp XCP 및 NIPAM을 사용하여 빅데이터 분석 데이터와 HPC 데이터를 AI로 이동하는 방법에 대해 설명합니다. 또한 빅데이터 및 HPC에서 AI로 데이터를 이동하면 얻을 수 있는 비즈니스 이점에 대해서도 설명합니다.

### 개념 및 구성 요소

#### 빅데이터 분석 스토리지

빅데이터 분석은 HDFS를 위한 주요 스토리지 제공업체입니다. 고객은 종종 Windows Azure Blob Storage, MapR-FS(MapR-FS) 및 S3 오브젝트 스토리지와 같은 HCFS(Hadoop 호환 파일 시스템)를 사용합니다.

#### 일반 병렬 파일 시스템입니다

IBM의 GPFS는 HDFS에 대한 대안을 제공하는 엔터프라이즈 파일 시스템입니다. GPFS는 애플리케이션에 블록 크기 및 복제 레이아웃을 결정할 수 있는 유연성을 제공하여 우수한 성능과 효율성을 제공합니다.

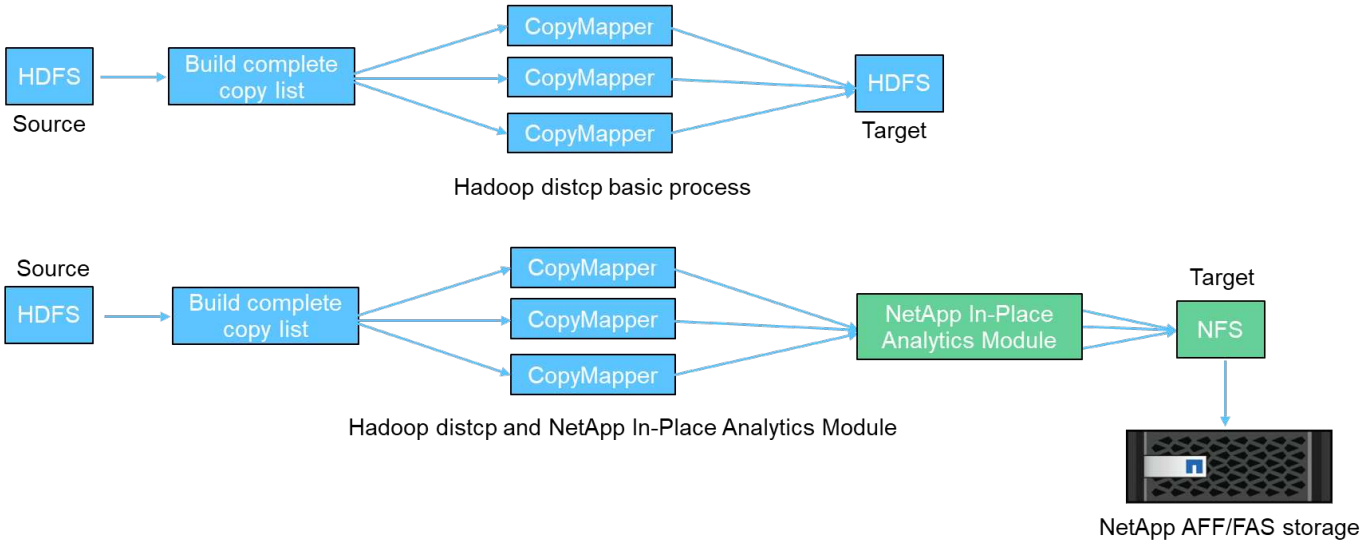
#### **NetApp** 데이터 이동 없는 분석 모듈

NetApp NIPAM(In-Place Analytics Module)은 NFS 데이터에 액세스하기 위한 Hadoop 클러스터의 역할을 합니다. 이 스트림에는 연결 풀, NFS InputStream, 파일 핸들 캐시 및 NFS OutputStream의 네 가지 구성 요소가 있습니다. 자세한 내용은 을 참조하십시오 "TR-4382: NetApp 데이터 이동 없는 분석 모듈."

#### **Hadoop** 분산 복제본

Hadoop Distributed Copy(DistCp)는 대규모 클러스터 간 및 클러스터 내 대처 작업에 사용되는 분산 복사 툴입니다. 이 툴은 데이터 배포, 오류 처리 및 보고를 위해 MapReduce를 사용합니다. 파일 및 디렉토리 목록을 확장하고 소스 목록에서 데이터를 복사하기 위한 작업을 매핑하기 위해 파일 및 디렉토리 입력을 수행합니다. 아래 이미지는 HDFS 및

비 HDFS의 DistCp 작업을 보여 줍니다.



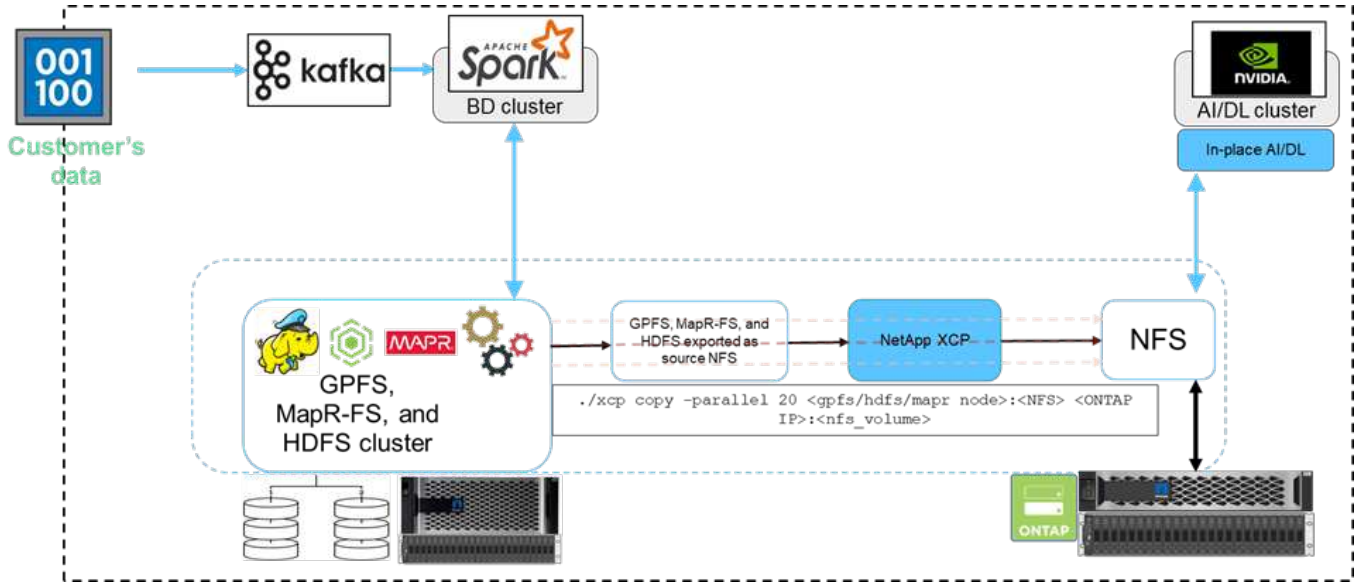

Hadoop DistCp는 추가 드라이버를 사용하지 않고 두 HDFS 시스템 간에 데이터를 이동합니다. NetApp은 비 HDFS 시스템용 드라이버를 제공합니다. NFS 대상의 경우 NIPAM은 데이터를 복사할 때 Hadoop DistCp가 NFS 대상과 통신하는 데 사용하는 데이터를 복사하는 드라이버를 제공합니다.

### NetApp Cloud Volumes Service를 참조하십시오

NetApp Cloud Volumes Service는 최고의 성능을 제공하는 클라우드 네이티브 파일 서비스입니다. 고객은 이 서비스를 통해 신속하게 리소스를 가동하고 속도를 낮추며 NetApp 기능을 사용하여 생산성을 높이고 직원 다운타임을 줄임으로써 출시 기간을 단축할 수 있습니다. Cloud Volumes Service는 전체 데이터 센터 설치 공간을 줄이고 기본 퍼블릭 클라우드 스토리지를 덜 사용하기 때문에 재해 복구 및 클라우드 백업을 위한 최적의 대안이 됩니다.

### NetApp XCP

NetApp XCP는 NetApp 제품 간 및 NetApp 간 데이터 마이그레이션을 빠르고 안정적으로 수행할 수 있는 클라이언트 소프트웨어입니다. 이 툴은 대량의 비정형 NAS 데이터를 NAS 시스템에서 NetApp 스토리지 컨트롤러로 복사하도록 설계되었습니다. xCP 마이그레이션 툴은 데이터 마이그레이션, 파일 또는 디렉토리 목록, 공간 보고 등 여러 요청을 병렬로 처리할 수 있는 멀티코어 다중 채널 I/O 스트리밍 엔진을 사용합니다. 기본 NetApp 데이터 마이그레이션 툴입니다. XCP를 사용하여 Hadoop 클러스터 및 HPC에서 NetApp NFS 스토리지로 데이터를 복사할 수 있습니다. 아래 다이어그램에서는 XCP를 사용하여 Hadoop 및 HPC 클러스터에서 NetApp NFS 볼륨으로 데이터를 전송하는 방법을 보여 줍니다.




**NetApp BlueXP** 복사 및 동기화

NetApp BlueXP Copy and Sync는 사내 스토리지와 클라우드 스토리지 간에 NFS, S3 및 CIFS 데이터를 원활하고 안전하게 전송 및 동기화하는 하이브리드 데이터 복제 서비스형 소프트웨어입니다. 이 소프트웨어는 데이터 마이그레이션, 아카이빙, 협업, 분석 등에 사용됩니다. 데이터가 전송되면 BlueXP Copy 및 Sync는 소스와 타겟 간에 데이터를 지속적으로 동기화합니다. 그런 다음 델타를 전송합니다. 또한 자체 네트워크, 클라우드 또는 사내 내의 데이터를 보호합니다. 이 소프트웨어는 비용 효율적인 솔루션을 제공하고 데이터 전송을 위한 모니터링 및 보고 기능을 제공하는 용량제 모델을 기반으로 합니다.

## 고객의 당면 과제

AI 운영을 위한 빅데이터 분석에서 데이터에 액세스하려고 할 때 고객이 다음과 같은 과제에 직면할 수 있습니다.

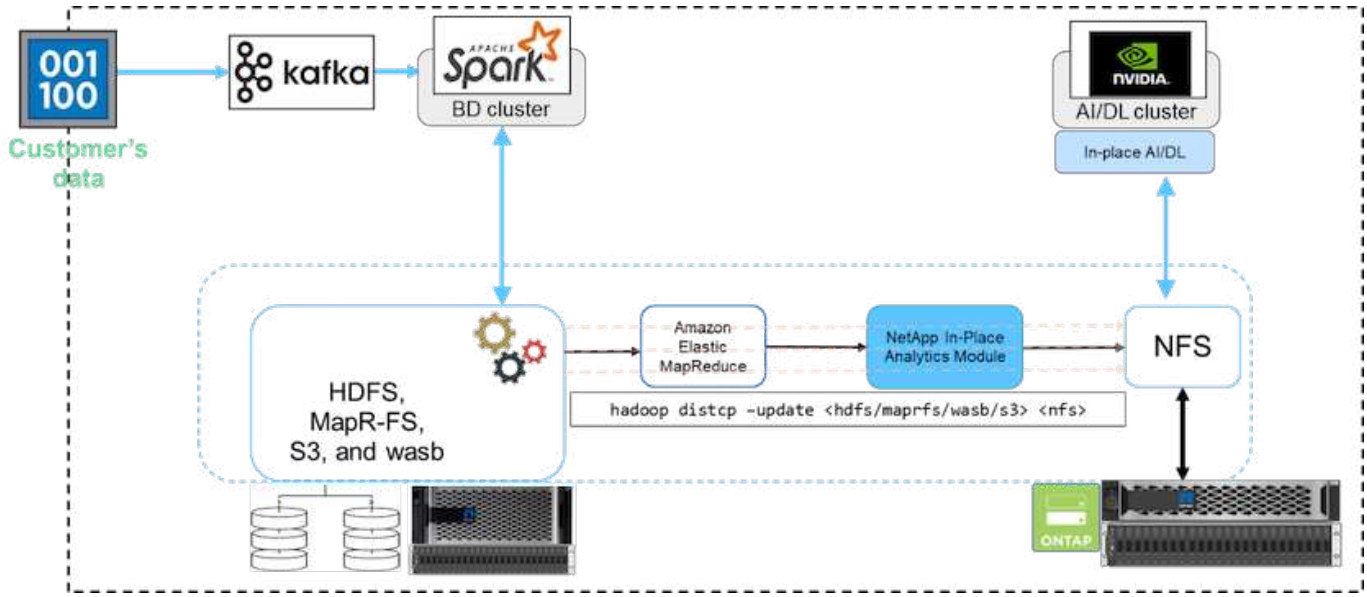
- 고객 데이터가 데이터 레이크 저장소에 있습니다. 데이터 레이크는 정형, 비정형, 반정형, 로그, 머신 간 데이터 등 다양한 유형의 데이터를 포함할 수 있습니다. 이러한 모든 데이터 유형은 AI 시스템에서 처리해야 합니다.
- AI는 Hadoop 파일 시스템과 호환되지 않습니다. 일반적인 AI 아키텍처는 HDFS 및 HCFS 데이터에 직접 액세스할 수 없으며, 이 데이터는 AI에 대해 이해할 수 있는 파일 시스템(NFS)으로 이동되어야 합니다.
- 데이터 레이크 데이터를 AI로 이동하는 작업에는 일반적으로 특수 프로세스가 필요합니다. 데이터 레이크의 데이터 양은 매우 클 수 있습니다. 고객은 AI 시스템으로 데이터를 이동할 수 있는 효율적이고 높은 처리량과 비용 효율적인 방법을 가지고 있어야 합니다.
- 데이터를 동기화하는 중입니다. 고객이 빅데이터 플랫폼과 AI 간의 데이터를 동기화하려는 경우 AI를 통해 처리된 데이터를 분석 처리에 빅데이터와 함께 사용할 수 있습니다.

## Data Mover 솔루션

빅데이터 클러스터에서 데이터는 MapR-FS, Windows Azure Storage Blob, S3 또는 Google 파일 시스템과 같은 HDFS 또는 HCFS에 저장됩니다. 우리는 소스에서 'Hadoop distcp' 명령을 사용하여 NIPAM의 도움을 받아 HDFS, MapR-FS 및 S3를 NetApp ONTAP NFS로 데이터를 복사하는 소스로 사용하여 테스트를 수행했습니다.

다음 다이어그램에서는 HDFS 스토리지를 사용하여 실행되는 Spark 클러스터에서 NetApp ONTAP NFS 볼륨으로

이동하는 일반적인 데이터 이동을 보여 줍니다. 이렇게 하면 NVIDIA에서 AI 작업을 처리할 수 있습니다.



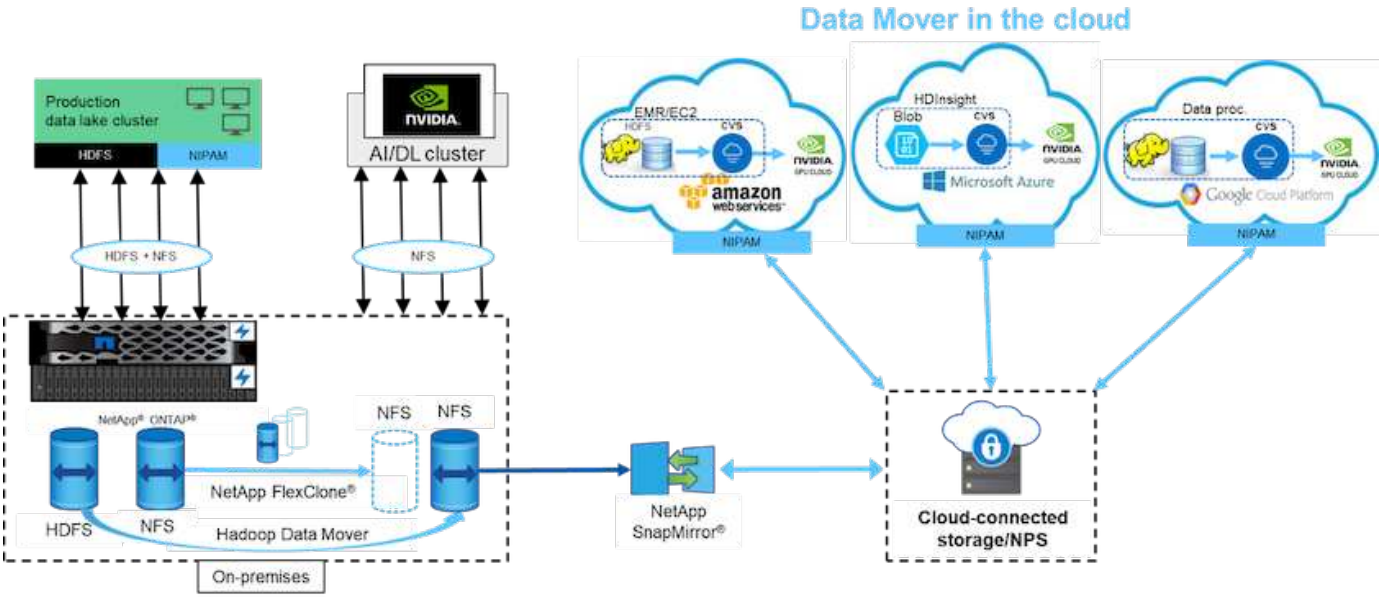

Hadoop distcp 명령은 MapReduce 프로그램을 사용하여 데이터를 복사합니다. NIPAM은 데이터를 복사할 때 MapReduce와 함께 Hadoop 클러스터의 드라이버 역할을 합니다. NIPAM은 단일 내보내기를 위해 여러 네트워크 인터페이스에 로드를 분산할 수 있습니다. 이 프로세스는 HDFS 또는 HCFS에서 NFS로 데이터를 복사할 때 여러 네트워크 인터페이스에 데이터를 분산하여 네트워크 처리량을 극대화합니다.

NIPAM은 MapR에서 지원 또는 인증되지 않았습니다.

## AI용 Data Mover 솔루션

AI용 Data Mover 솔루션은 AI 작업에서 Hadoop 데이터를 처리해야 하는 고객의 요구사항을 기반으로 합니다. NetApp은 NIPAM을 사용하여 HDFS에서 NFS로 데이터를 이동합니다. 한 사용 사례에서 고객은 사내 NFS로 데이터를 이동해야 했고, 또 다른 고객은 클라우드의 GPU 클라우드 인스턴스에서 데이터를 처리하기 위해 Windows Azure Storage Blob에서 Cloud Volumes Service로 데이터를 이동해야 했습니다.

다음 다이어그램에서는 Data Mover 솔루션 세부 정보를 보여 줍니다.




Data Mover 솔루션을 구축하려면 다음 단계가 필요합니다.

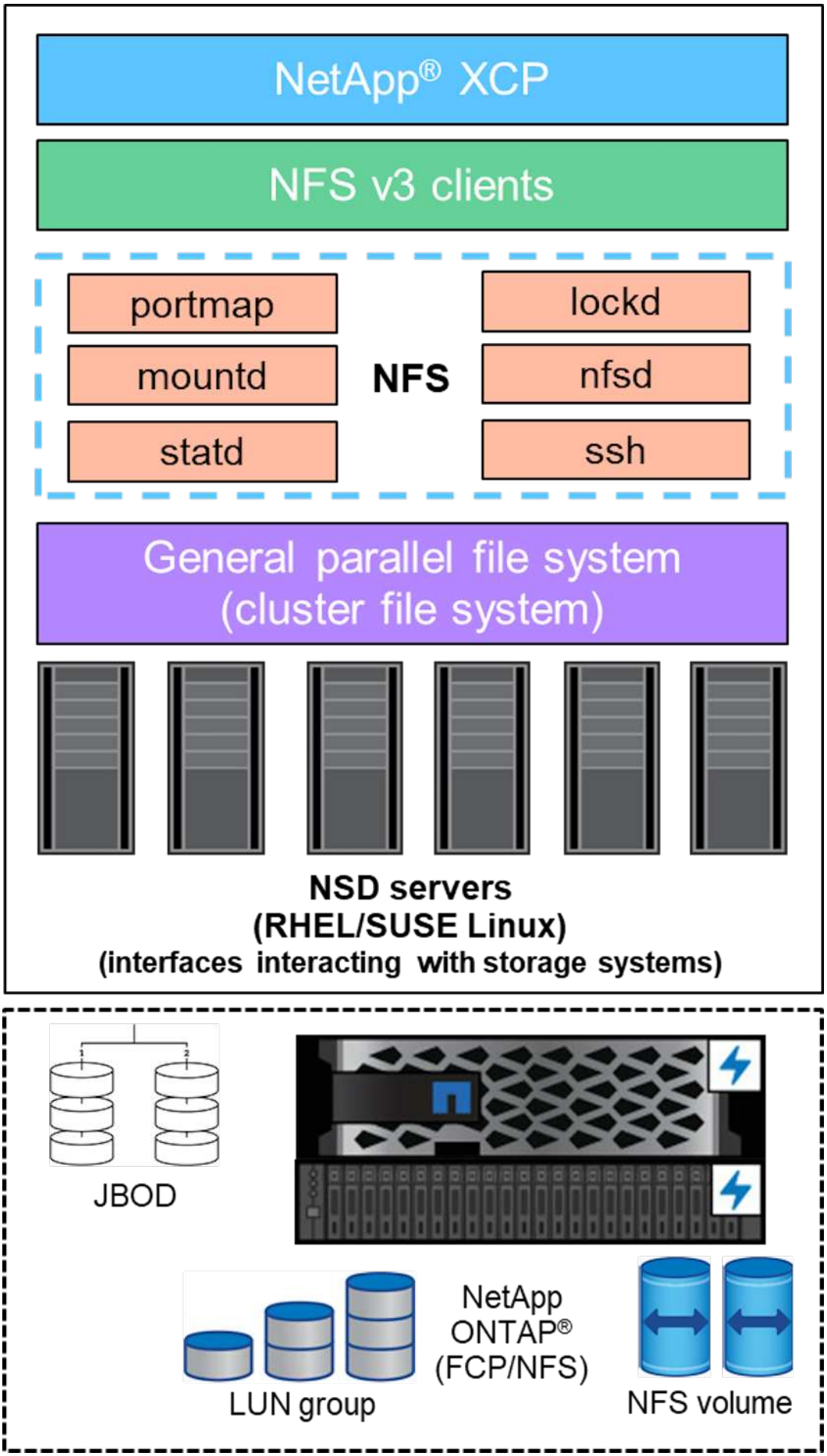
1. ONTAP SAN은 HDFS를 제공하고, NAS는 NIPAM을 통해 NFS 볼륨을 운영 데이터 레이크 클러스터에 제공합니다.
2. 고객의 데이터는 HDFS 및 NFS에 있습니다. NFS 데이터는 빅데이터 분석 및 AI 운영에 사용되는 다른 애플리케이션의 운영 데이터일 수 있습니다.
3. NetApp FlexClone 기술은 운영 NFS 볼륨의 클론을 생성하여 사내 AI 클러스터에 프로비저닝합니다.
4. HDFS SAN LUN의 데이터는 NIPAM 및 "Hadoop distcp" 명령을 사용하여 NFS 볼륨으로 복제됩니다. NIPAM은 여러 네트워크 인터페이스의 대역폭을 사용하여 데이터를 전송합니다. 이 프로세스는 더 많은 데이터를 전송할 수 있도록 데이터 복사 시간을 줄입니다.
5. 두 NFS 볼륨 모두 AI 운영을 위해 AI 클러스터에 프로비저닝됩니다.
6. 클라우드의 GPU로 사내 NFS 데이터를 처리하기 위해 NFS 볼륨은 NetApp SnapMirror 기술을 사용해 NPS(NetApp Private Storage)로 미러링되고 GPU를 위한 클라우드 서비스 공급자에 마운트됩니다.
7. 고객은 클라우드 서비스 공급자의 GPU에서 EC2/EMR, HDInsight 또는 DataProc 서비스의 데이터를 처리하려고 합니다. Hadoop Data Mover는 데이터를 Hadoop 서비스에서 NIPAM 및 'Hadoop distcp' 명령을 사용하여 Cloud Volumes Services로 이동합니다.
8. Cloud Volumes Service 데이터는 NFS 프로토콜을 통해 AI에 프로비저닝됩니다.AI를 통해 처리되는 데이터는 NIPAM, SnapMirror 및 NPS를 통해 NVIDIA 클러스터와 함께 사내 위치에서 빅데이터 분석을 위한 데이터가 전송될 수 있습니다.

이 시나리오에서는 고객이 사내의 NetApp 스토리지 컨트롤러에서 AI 처리를 위해 필요한 원격 위치의 NAS 시스템에 대용량 파일 개수 데이터가 있습니다. 이 시나리오에서는 XCP 마이그레이션 도구를 사용하여 데이터를 더 빠른 속도로 마이그레이션하는 것이 좋습니다.

하이브리드 사용 사례인 고객은 BlueXP Copy 및 Sync를 사용하여 NVIDIA 클러스터에 있는 GPU와 같은 GPU를 사용하여 NFS, CIFS, S3 데이터의 온프레미스 데이터를 클라우드로 마이그레이션하고 AI 처리를 위해 그 반대로 마이그레이션할 수 있습니다. BlueXP Copy 및 Sync와 XCP 마이그레이션 툴은 모두 NetApp ONTAP NFS로의 NFS 데이터 마이그레이션에 사용됩니다.

### GPFS를 NetApp ONTAP NFS로

이 검증에서 4대의 서버를 NSD(Network Shared Disk) 서버로 사용하여 GPFS에 물리적 디스크를 제공했습니다. GPFS는 NSD 디스크 위에 생성되어 NFS 내보내기로 내보내므로 NFS 클라이언트가 아래 그림과 같이 액세스할 수 있습니다. XCP를 사용하여 GPFS로 내보낸 NFS의 데이터를 NetApp NFS 볼륨으로 복사했습니다.


NetApp® XCP
NFS v3 clients
portmap
mountd
statd
NFS
lockd
nfsd
ssh
General parallel file system
(cluster file system)
NSD servers
(RHEL/SUSE Linux)
(interfaces interacting with storage systems)
JBOD
LUN group
NetApp
ONTAP®
(FCP/NFS)
NFS volume

### GPFS 기본

GPFS에 사용되는 노드 유형은 다음과 같습니다.

- * Admin node. * 관리 명령에서 노드 간 통신에 사용하는 노드 이름을 포함하는 선택적 필드를 지정합니다. 예를 들어, 관리자 노드 mastr-51.netapp.com` 가 클러스터의 다른 모든 노드에 네트워크 검사를 전달할 수 있습니다.
- * 쿼럼 노드. * 쿼럼이 파생되는 노드 풀에 노드가 포함되어 있는지 여부를 확인합니다. 쿼럼 노드로 적어도 하나의 노드가 필요합니다.
- * Manager Node. * 노드가 파일 시스템 관리자 및 토큰 관리자를 선택할 수 있는 노드 풀의 일부인지 여부를 나타냅니다. 둘 이상의 노드를 관리자 노드로 정의하는 것이 좋습니다. 관리자로 지정하는 노드 수는 워크로드와 GPFS 서버 라이센스 수에 따라 다릅니다. 대규모 병렬 작업을 실행 중인 경우 웹 애플리케이션을 지원하는 4노드 클러스터보다 더 많은 관리자 노드가 필요할 수 있습니다.
- * NSD 서버. * GPFS와 함께 사용할 각 물리적 디스크를 준비하는 서버입니다.
- * 프로토콜 노드. * NFS를 통해 모든 SSH(Secure Shell) 프로토콜을 통해 GPFS 데이터를 직접 공유하는 노드입니다. 이 노드에는 GPFS 서버 라이센스가 필요합니다.

### GPFS, NFS 및 XCP의 운영 목록입니다

이 섹션은 GPFS를 만들고 GPFS를 NFS 내보내기로 내보낸 후 XCP를 사용하여 데이터를 전송하는 작업 목록을 제공합니다.

#### GPFS 생성

GPFS를 만들려면 다음 단계를 완료하십시오.

1. 서버 중 하나에서 Linux 버전에 대한 스펙트럼 스케일 데이터 액세스를 다운로드하고 설치합니다.
2. 모든 노드에 필수 구성 요소 패키지(예: Chef)를 설치하고 모든 노드에서 SELinux(Security-Enhanced Linux)를 비활성화합니다.
3. 설치 노드를 설정하고 클러스터 정의 파일에 관리 노드 및 GPFS 노드를 추가합니다.
4. 관리자 노드, 쿼럼 노드, NSD 서버 및 GPFS 노드를 추가합니다.
5. GUI, 관리 및 GPFS 노드를 추가하고, 필요한 경우 추가 GUI 서버를 추가합니다.
6. 다른 GPFS 노드를 추가하고 모든 노드 목록을 확인하십시오.
7. 클러스터 정의 파일의 모든 GPFS 노드에 설정할 클러스터 이름, 프로필, 원격 셸 바이너리, 원격 파일 복사본 바이너리 및 포트 범위를 지정합니다.
8. GPFS 구성 설정을 보고 추가 관리 노드를 추가합니다.
9. 데이터 수집을 비활성화하고 데이터 패키지를 IBM 지원 센터에 업로드합니다.
10. NTP를 활성화하고 설치 전에 구성을 미리 확인합니다.
11. NSD 디스크를 구성, 생성 및 확인합니다.
12. GPFS를 생성합니다.
13. GPFS를 마운트합니다.
14. GPFS에 필요한 권한을 확인하고 제공하십시오.
15. ddd 명령을 실행하여 GPFS 읽기 및 쓰기를 확인합니다.

#### GPFS를 NFS로 내보냅니다

GPFS를 NFS로 내보내려면 다음 단계를 완료하십시오.

1. '/etc/exports' 파일을 통해 GPFS를 NFS로 내보냅니다.
2. 필요한 NFS 서버 패키지를 설치합니다.
3. NFS 서비스를 시작합니다.
4. GPFS에 파일을 나열하여 NFS 클라이언트를 검증합니다.

#### NFS 클라이언트를 구성합니다

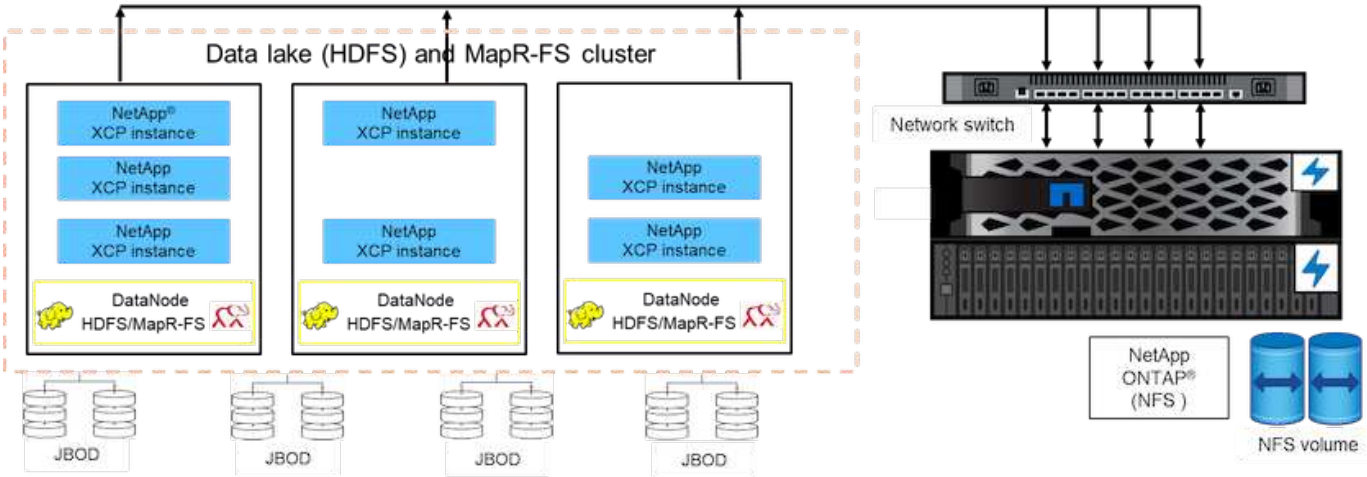
NFS 클라이언트를 구성하려면 다음 단계를 수행하십시오.

1. '/etc/exports' 파일을 통해 GPFS를 NFS로 내보냅니다.
2. NFS 클라이언트 서비스를 시작합니다.
3. NFS 클라이언트의 NFS 프로토콜을 통해 GPFS를 마운트합니다.
4. NFS 마운트 폴더에서 GPFS 파일 목록을 검증합니다.
5. XCP를 사용하여 GPFS에서 내보낸 NFS에서 NetApp NFS로 데이터를 이동하십시오.
6. NFS 클라이언트에서 GPFS 파일을 검증합니다.

## HDFS 및 MapR-FS를 ONTAP NFS로 설정합니다

이 솔루션을 위해 NetApp은 데이터 레이크(HDFS) 및 MapR 클러스터 데이터에서 ONTAP NFS로 데이터 마이그레이션을 검증했습니다. MapR-FS 및 HDFS에 데이터가 상주했습니다. NetApp XCP는 HDFS 및 MapR-FS와 같은 분산 파일 시스템에서 ONTAP NFS로 데이터를 직접 마이그레이션하는 새로운 기능을 소개했습니다. xCP는 비동기 스레드 및 HDFS C API 호출을 사용하여 MapR-FS 및 HDFS에서 데이터를 통신 및 전송합니다.

아래 그림은 데이터 레이크(HDFS) 및 MapR-FS에서 ONTAP NFS로 데이터 마이그레이션을 보여 줍니다. 이 새로운 기능을 사용하면 소스를 NFS 공유로 내보낼 필요가 없습니다.



고객이 **HDFS** 및 **MapR-FS**에서 **NFS**로 이동하는 이유는 무엇입니까**?**

Cloudera 및 Hortonworks와 같은 Hadoop 배포 대부분은 HDFS 및 MapR 배포판과 같은 자체 파일 시스템 MapR-FS를 사용하여 데이터를 저장합니다. HDFS 및 MapR-FS 데이터는 머신 러닝(ML) 및 딥 러닝(DL)에 활용할 수 있는 데이터 과학자에게 중요한 통찰력을 제공합니다. HDFS 및 MapR-FS의 데이터는 공유되지 않으며 다른 애플리케이션에서 사용할 수 없습니다. 고객은 공유 데이터, 특히 고객의 중요한 데이터가 여러 애플리케이션에서 사용되는 은행 부문에서 데이터를 찾고 있습니다. 최신 버전의 Hadoop(3.x 이상)은 NFS 데이터 소스를 지원하며, 타사 소프트웨어를 추가하지 않고도 이 소스에 액세스할 수 있습니다. 새로운 NetApp XCP 기능을 사용하면 데이터를 HDFS 및 MapR-FS에서 NetApp NFS로 직접 이동하여 여러 애플리케이션에 액세스할 수 있습니다

12개의 MapR 노드 및 4개의 NFS 서버를 통해 초기 성능 테스트를 위해 AWS(Amazon Web Services)에서 MapR-FS의 데이터를 NFS로 전송하기 위한 테스트가 완료되었습니다.

| | 수량 | 크기 | **vCPU** | 메모리 | 스토리지 | 네트워크 |
|---|---|---|---|---|---|---|
| NFS 서버 | 4 | i3en.24xLarge | 96 | 488GiB | 8x 7500 NVMe SSD | 100 |
| MapR 노드 | 12 | I3en.12xLarge | 48 | 384GiB | 4x 7500 NVMe SSD | 50 |

초기 테스트에 따르면 20Gbps의 처리량을 확보했으며 일일 2PB의 데이터를 전송할 수 있었습니다.

HDFS를 NFS로 내보내지 않고 HDFS 데이터 마이그레이션에 대한 자세한 내용은 의 "배포 단계 - NAS" 섹션을 참조하십시오 "TR-4863: NetApp XCP-Data Mover, 파일 마이그레이션 및 분석에 대한 모범 사례 지침 - 4863".

## 비즈니스 이점

데이터를 빅데이터 분석에서 AI로 이동하면 다음과 같은 이점이 있습니다.

- 서로 다른 Hadoop 파일 시스템과 GPFS에서 데이터를 유니파이드 NFS 스토리지 시스템으로 추출하는 기능
- 데이터 전송을 위한 Hadoop 통합 및 자동화 방식
- Hadoop 파일 시스템에서 데이터를 이동하는 데 필요한 라이브러리 개발 비용 절감
- NIPAM을 사용하여 단일 데이터 소스에서 여러 네트워크 인터페이스의 총 처리량을 통해 최대 성능을 발휘합니다
- 데이터 전송을 위한 예약 방식 및 온디맨드 방식
- ONTAP 데이터 관리 소프트웨어를 사용하여 유니파이드 NFS 데이터에 대한 스토리지 효율성 및 엔터프라이즈 관리 기능을 제공합니다
- 데이터 전송을 위한 Hadoop 방식을 사용하면 데이터 이동 비용이 전혀 들지 않습니다

## **GPFS**에서 **NFS**로 **-** 자세한 단계

이 섹션은 GPFS를 구성하고 NetApp XCP를 사용하여 NFS로 데이터를 이동하는 데 필요한 세부 단계를 제공합니다.

### **GPFS** 구성

1. 서버 중 하나에서 Linux용 Spectrum Scale 데이터 액세스를 다운로드하고 설치합니다.

```
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# ls
Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-install
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# chmod +x Spectrum_Scale_Data_Access-5.0.3.1-x86_64-
Linux-install
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# ./Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install --manifest
manifest
…
<contents removes to save page space>
…
```

2. 모든 노드에 필수 구성 요소 패키지(셰프 및 커널 헤더 포함)를 설치합니다.

```
[root@mastr-51 5.0.3.1]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; rpm -ivh  /gpfs_install/chef* "; done
mastr-51.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
package chef-13.6.4-1.el7.x86_64 is already installed
mastr-53.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-136.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-138.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
```

```
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-140.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
[root@mastr-51 5.0.3.1]#
[root@mastr-51 installer]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; yumdownloader kernel-headers-3.10.0-
862.3.2.el7.x86_64 ; rpm -Uvh --oldpackage kernel-headers-3.10.0-
862.3.2.el7.x86_64.rpm"; done
mastr-51.netapp.com
Loaded plugins: priorities, product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-957.21.2.el7
########################################
mastr-53.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
########################################
workr-136.netapp.com
Loaded plugins: product-id, subscription-manager
Repository ambari-2.7.3.0 is listed more than once in the configuration
Preparing...
########################################
```

```
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
##########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
##########################################
workr-138.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
##########################################
package kernel-headers-3.10.0-862.3.2.el7.x86_64 is already installed
workr-140.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
##########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
##########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
##########################################
[root@mastr-51 installer]#
```

3. 모든 노드에서 SELinux를 해제합니다.

```
[root@mastr-51 5.0.3.1]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; sudo setenforce 0"; done
mastr-51.netapp.com
setenforce: SELinux is disabled
mastr-53.netapp.com
setenforce: SELinux is disabled
workr-136.netapp.com
setenforce: SELinux is disabled
workr-138.netapp.com
setenforce: SELinux is disabled
workr-140.netapp.com
setenforce: SELinux is disabled
[root@mastr-51 5.0.3.1]#
```

4. 설치 노드를 설정합니다.

```
[root@mastr-51 installer]# ./spectrumscale setup -s 10.63.150.51
[ INFO  ] Installing prerequisites for install node
[ INFO  ] Existing Chef installation detected. Ensure the PATH is
configured so that chef-client and knife commands can be run.
[ INFO  ] Your control node has been configured to use the IP
10.63.150.51 to communicate with other nodes.
[ INFO  ] Port 8889 will be used for chef communication.
[ INFO  ] Port 10080 will be used for package distribution.
[ INFO  ] Install Toolkit setup type is set to Spectrum Scale (default).
If an ESS is in the cluster, run this command to set ESS mode:
./spectrumscale setup -s server_ip -st ess
[ INFO  ] SUCCESS
[ INFO  ] Tip : Designate protocol, nsd and admin nodes in your
environment to use during install:./spectrumscale -v node add <node> -p
-a -n
[root@mastr-51 installer]#
```

5. 클러스터 정의 파일에 관리 노드 및 GPFS 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-51 -a
[ INFO  ] Adding node mastr-51.netapp.com as a GPFS node.
[ INFO  ] Setting mastr-51.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

6. 관리자 노드 및 GPFS 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-53 -m
[ INFO  ] Adding node mastr-53.netapp.com as a GPFS node.
[ INFO  ] Adding node mastr-53.netapp.com as a manager node.
[root@mastr-51 installer]#
```

7. 쿼럼 노드와 GPFS 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -q
[ INFO  ] Adding node workr-136.netapp.com as a GPFS node.
[ INFO  ] Adding node workr-136.netapp.com as a quorum node.
[root@mastr-51 installer]#
```

8. NSD 서버 및 GPFS 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-138 -n
[ INFO  ] Adding node workr-138.netapp.com as a GPFS node.
[ INFO  ] Adding node workr-138.netapp.com as an NSD server.
[ INFO  ] Configuration updated.
[ INFO  ] Tip :If all node designations are complete, add NSDs to your
cluster definition and define required filessytems:./spectrumscale nsd
add <device> -p <primary node> -s <secondary node> -fs <file system>
[root@mastr-51 installer]#
```

9. GUI, 관리 및 GPFS 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -g
[ INFO  ] Setting workr-136.netapp.com as a GUI server.
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -a
[ INFO  ] Setting workr-136.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

10. 다른 GUI 서버를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-53 -g
[ INFO  ] Setting mastr-53.netapp.com as a GUI server.
[root@mastr-51 installer]#
```

11. 다른 GPFS 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-140
[ INFO  ] Adding node workr-140.netapp.com as a GPFS node.
[root@mastr-51 installer]#
```

12. 모든 노드를 확인하고 나열합니다.

```
[root@mastr-51 installer]# ./spectrumscale node list
[ INFO  ] List of nodes in current configuration:
[ INFO  ] [Installer Node]
[ INFO  ] 10.63.150.51
[ INFO  ]
[ INFO  ] [Cluster Details]
[ INFO  ] No cluster name configured
[ INFO  ] Setup Type: Spectrum Scale
[ INFO  ]
[ INFO  ] [Extended Features]
[ INFO  ] File Audit logging     : Disabled
[ INFO  ] Watch folder           : Disabled
[ INFO  ] Management GUI         : Enabled
[ INFO  ] Performance Monitoring : Disabled
[ INFO  ] Callhome               : Enabled
[ INFO  ]
[ INFO  ] GPFS                 Admin  Quorum  Manager   NSD   Protocol
GUI   Callhome   OS   Arch
[ INFO  ] Node                  Node   Node     Node   Server   Node
Server  Server
[ INFO  ] mastr-51.netapp.com    X
rhel7  x86_64
[ INFO  ] mastr-53.netapp.com                    X
X             rhel7  x86_64
[ INFO  ] workr-136.netapp.com   X       X
X             rhel7  x86_64
[ INFO  ] workr-138.netapp.com                          X
rhel7  x86_64
[ INFO  ] workr-140.netapp.com
rhel7  x86_64
[ INFO  ]
[ INFO  ] [Export IP address]
[ INFO  ] No export IP addresses configured
[root@mastr-51 installer]#
```

13. 클러스터 정의 파일에 클러스터 이름을 지정합니다.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -c mastr-
51.netapp.com
[ INFO  ] Setting GPFS cluster name to mastr-51.netapp.com
[root@mastr-51 installer]#
```

14. 프로파일을 지정합니다.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -p default
[ INFO  ] Setting GPFS profile to default
[root@mastr-51 installer]#
Profiles options: default [gpfsProtocolDefaults], random I/O
[gpfsProtocolsRandomIO], sequential I/O [gpfsProtocolDefaults], random
I/O [gpfsProtocolRandomIO]
```

15. GPFS에서 사용할 원격 셸 바이너리를 지정하고 '-r 인수'를 사용합니다.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -r /usr/bin/ssh
[ INFO  ] Setting Remote shell command to /usr/bin/ssh
[root@mastr-51 installer]#
```

16. GPFS에서 사용할 원격 파일 복사 바이너리를 지정하고 '-rc 인수'를 사용하십시오.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -rc /usr/bin/scp
[ INFO  ] Setting Remote file copy command to /usr/bin/scp
[root@mastr-51 installer]#
```

17. 모든 GPFS 노드에 설정할 포트 범위를 지정하고, '-e argument'를 사용한다.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -e 60000-65000
[ INFO  ] Setting GPFS Daemon communication port range to 60000-65000
[root@mastr-51 installer]#
```

18. GPFS 구성 설정을 봅니다.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs --list
[ INFO  ] Current settings are as follows:
[ INFO  ] GPFS cluster name is mastr-51.netapp.com.
[ INFO  ] GPFS profile is default.
[ INFO  ] Remote shell command is /usr/bin/ssh.
[ INFO  ] Remote file copy command is /usr/bin/scp.
[ INFO  ] GPFS Daemon communication port range is 60000-65000.
[root@mastr-51 installer]#
```

19. 관리 노드를 추가합니다.

```
[root@mastr-51 installer]# ./spectrumscale node add 10.63.150.53 -a
[ INFO  ] Setting mastr-53.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

20. 데이터 수집을 비활성화하고 데이터 패키지를 IBM 지원 센터에 업로드합니다.

```
[root@mastr-51 installer]# ./spectrumscale callhome disable
[ INFO  ] Disabling the callhome.
[ INFO  ] Configuration updated.
[root@mastr-51 installer]#
```

21. NTP를 활성화합니다.

```
[root@mastr-51 installer]# ./spectrumscale config ntp -e on
[root@mastr-51 installer]# ./spectrumscale config ntp -l
[ INFO  ] Current settings are as follows:
[ WARN  ] No value for Upstream NTP Servers(comma separated IP's with NO
space between multiple IPs) in clusterdefinition file.
[root@mastr-51 installer]# ./spectrumscale config ntp -s 10.63.150.51
[ WARN  ] The NTP package must already be installed and full
bidirectional access to the UDP port 123 must be allowed.
[ WARN  ] If NTP is already running on any of your nodes, NTP setup will
be skipped. To stop NTP run 'service ntpd stop'.
[ WARN  ] NTP is already on
[ INFO  ] Setting Upstream NTP Servers(comma separated IP's with NO
space between multiple IPs) to 10.63.150.51
[root@mastr-51 installer]# ./spectrumscale config ntp -e on
[ WARN  ] NTP is already on
[root@mastr-51 installer]# ./spectrumscale config ntp -l
[ INFO  ] Current settings are as follows:
[ INFO  ] Upstream NTP Servers(comma separated IP's with NO space
between multiple IPs) is 10.63.150.51.
[root@mastr-51 installer]#

[root@mastr-51 installer]# service ntpd start
Redirecting to /bin/systemctl start ntpd.service
[root@mastr-51 installer]# service ntpd status
Redirecting to /bin/systemctl status ntpd.service
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor
```

```
preset: disabled)
   Active: active (running) since Tue 2019-09-10 14:20:34 UTC; 1s ago
  Process: 2964 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS
(code=exited, status=0/SUCCESS)
 Main PID: 2965 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─2965 /usr/sbin/ntpd -u ntp:ntp -g

Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: ntp_io: estimated max
descriptors: 1024, initial socket boundary: 16
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen and drop on 0
v4wildcard 0.0.0.0 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen and drop on 1
v6wildcard :: UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 2 lo
127.0.0.1 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 3
enp4s0f0 10.63.150.51 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 4 lo
::1 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 5
enp4s0f0 fe80::219:99ff:feef:99fa UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listening on routing
socket on fd #22 for interface updates
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: 0.0.0.0 c016 06 restart
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: 0.0.0.0 c012 02 freq_set
kernel 11.890 PPM
[root@mastr-51 installer]#
```

22. 설치하기 전에 구성을 미리 확인합니다.

```
[root@mastr-51 installer]# ./spectrumscale install -pr
[ INFO  ] Logging to file: /usr/lpp/mmfs/5.0.3.1/installer/logs/INSTALL-
PRECHECK-10-09-2019_14:51:43.log
[ INFO  ] Validating configuration
[ INFO  ] Performing Chef (deploy tool) checks.
[ WARN  ] NTP is already running on: mastr-51.netapp.com. The install
toolkit will no longer setup NTP.
[ INFO  ] Node(s): ['workr-138.netapp.com'] were defined as NSD node(s)
but the toolkit has not been told about any NSDs served by these node(s)
nor has the toolkit been told to create new NSDs on these node(s). The
install will continue and these nodes will be assigned server licenses.
If NSDs are desired, either add them to the toolkit with
<./spectrumscale nsd add> followed by a <./spectrumscale install> or add
them manually afterwards using mmcrnsd.
[ INFO  ] Install toolkit will not configure file audit logging as it
has been disabled.
[ INFO  ] Install toolkit will not configure watch folder as it has been
disabled.
[ INFO  ] Checking for knife bootstrap configuration...
[ INFO  ] Performing GPFS checks.
[ INFO  ] Running environment checks
[ INFO  ] Skipping license validation as no existing GPFS cluster
detected.
[ INFO  ] Checking pre-requisites for portability layer.
[ INFO  ] GPFS precheck OK
[ INFO  ] Performing Performance Monitoring checks.
[ INFO  ] Running environment checks for Performance Monitoring
[ INFO  ] Performing GUI checks.
[ INFO  ] Performing FILE AUDIT LOGGING checks.
[ INFO  ] Running environment checks for file  Audit logging
[ INFO  ] Network check from admin node workr-136.netapp.com to all
other nodes in the cluster passed
[ INFO  ] Network check from admin node mastr-51.netapp.com to all other
nodes in the cluster passed
[ INFO  ] Network check from admin node mastr-53.netapp.com to all other
nodes in the cluster passed
[ INFO  ] The install toolkit will not configure call home as it is
disabled. To enable call home, use the following CLI command:
./spectrumscale callhome enable
[ INFO  ] Pre-check successful for install.
[ INFO  ] Tip : ./spectrumscale install
[root@mastr-51 installer]#
```

23. NSD 디스크를 구성합니다.

```
[root@mastr-51 cluster-test]# cat disk.1st
%nsd: device=/dev/sdf
nsd=nsd1
servers=workr-136
usage=dataAndMetadata
failureGroup=1

%nsd: device=/dev/sdf
nsd=nsd2
servers=workr-138
usage=dataAndMetadata
failureGroup=1
```

24. NSD 디스크를 생성합니다.

```
[root@mastr-51 cluster-test]# mmcrnsd -F disk.1st -v no
mmcrnsd: Processing disk sdf
mmcrnsd: Processing disk sdf
mmcrnsd: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
[root@mastr-51 cluster-test]#
```

25. NSD 디스크 상태를 확인합니다.

```
[root@mastr-51 cluster-test]# mmlsnsd

 File system   Disk name    NSD servers
---------------------------------------------------------------------------
---
 (free disk)   nsd1         workr-136.netapp.com
 (free disk)   nsd2         workr-138.netapp.com

[root@mastr-51 cluster-test]#
```

26. GPFS를 생성합니다.

```
[root@mastr-51 cluster-test]# mmcrfs gpfs1 -F disk.1st -B 1M -T /gpfs1

The following disks of gpfs1 will be formatted on node workr-
136.netapp.com:
    nsd1: size 3814912 MB
    nsd2: size 3814912 MB
Formatting file system ...
Disks up to size 33.12 TB can be added to storage pool system.
Creating Inode File
Creating Allocation Maps
Creating Log Files
Clearing Inode Allocation Map
Clearing Block Allocation Map
Formatting Allocation Map for storage pool system
Completed creation of file system /dev/gpfs1.
mmcrfs: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
[root@mastr-51 cluster-test]#
```

27. GPFS를 마운트합니다.

```
[root@mastr-51 cluster-test]# mmmount all -a
Tue Oct  8 18:05:34 UTC 2019: mmmount: Mounting file systems ...
[root@mastr-51 cluster-test]#
```

28. GPFS에 필요한 권한을 확인하고 제공하십시오.

```
[root@mastr-51 cluster-test]# mmlsdisk gpfs1
disk         driver   sector     failure holds    holds
storage
name         type       size       group metadata data  status
availability pool
------------ -------- ------ ----------- -------- ----- -------------
------------ ------------
nsd1         nsd         512           1 Yes      Yes   ready         up
system
nsd2         nsd         512           1 Yes      Yes   ready         up
system
[root@mastr-51 cluster-test]#

[root@mastr-51 cluster-test]# for i in 51 53 136 138  ; do ssh
10.63.150.$i "hostname; chmod 777 /gpfs1" ; done;
mastr-51.netapp.com
mastr-53.netapp.com
workr-136.netapp.com
workr-138.netapp.com
[root@mastr-51 cluster-test]#
```

29. ddd 명령을 실행하여 GPFS 읽기 및 쓰기를 확인합니다.

```
[root@mastr-51 cluster-test]# dd if=/dev/zero of=/gpfs1/testfile
bs=1024M count=5
5+0 records in
5+0 records out
5368709120 bytes (5.4 GB) copied, 8.3981 s, 639 MB/s
[root@mastr-51 cluster-test]# for i in 51 53 136 138  ; do ssh
10.63.150.$i "hostname; ls -ltrh /gpfs1" ; done;
mastr-51.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
mastr-53.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
workr-136.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
workr-138.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
[root@mastr-51 cluster-test]#
```

**GPFS를 NFS로 내보냅니다**

GPFS를 NFS로 내보내려면 다음 단계를 완료하십시오.

1. '/etc/exports' 파일을 통해 GPFS를 NFS로 내보냅니다.

```
[root@mastr-51 gpfs1]# cat /etc/exports
/gpfs1        *(rw,fsid=745)
[root@mastr-51 gpfs1]
```

2. 필요한 NFS 서버 패키지를 설치합니다.

```
[root@mastr-51 ~]# yum install rpcbind
Loaded plugins: priorities, product-id, search-disabled-repos,
subscription-manager
Resolving Dependencies
--> Running transaction check
---> Package rpcbind.x86_64 0:0.2.0-47.el7 will be updated
---> Package rpcbind.x86_64 0:0.2.0-48.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

=================================================================================
=================================================================================
=================================================================================
======================
 Package                                                   Arch
Version                                                        Repository
Size
=================================================================================
=================================================================================
=================================================================================
======================
Updating:
 rpcbind                                                   x86_64
0.2.0-48.el7                                                   rhel-7-
server-rpms                                                    60 k

Transaction Summary
=================================================================================
=================================================================================
=================================================================================
======================
Upgrade  1 Package
```

```
Total download size: 60 k
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
rpcbind-0.2.0-48.el7.x86_64.rpm
|  60 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating   : rpcbind-0.2.0-48.el7.x86_64
1/2
  Cleanup    : rpcbind-0.2.0-47.el7.x86_64
2/2
  Verifying  : rpcbind-0.2.0-48.el7.x86_64
1/2
  Verifying  : rpcbind-0.2.0-47.el7.x86_64
2/2

Updated:
  rpcbind.x86_64 0:0.2.0-48.el7

Complete!
[root@mastr-51 ~]#
```

3. NFS 서비스를 시작합니다.

```
[root@mastr-51 ~]# service nfs status
Redirecting to /bin/systemctl status nfs.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: inactive (dead)
[root@mastr-51 ~]# service rpcbind start
Redirecting to /bin/systemctl start rpcbind.service
[root@mastr-51 ~]# service nfs start
Redirecting to /bin/systemctl start nfs.service
[root@mastr-51 ~]# service nfs status
Redirecting to /bin/systemctl status nfs.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: active (exited) since Wed 2019-11-06 16:34:50 UTC; 2s ago
  Process: 24402 ExecStartPost=/bin/sh -c if systemctl -q is-active
gssproxy; then systemctl reload gssproxy ; fi (code=exited,
status=0/SUCCESS)
  Process: 24383 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS (code=exited,
status=0/SUCCESS)
  Process: 24379 ExecStartPre=/usr/sbin/exportfs -r (code=exited,
status=0/SUCCESS)
 Main PID: 24383 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/nfs-server.service

Nov 06 16:34:50 mastr-51.netapp.com systemd[1]: Starting NFS server and
services...
Nov 06 16:34:50 mastr-51.netapp.com systemd[1]: Started NFS server and
services.
[root@mastr-51 ~]#
```

4. GPFS에 파일을 나열하여 NFS 클라이언트를 검증합니다.

```
[root@mastr-51 gpfs1]# df -Th
Filesystem                                  Type      Size  Used Avail
Use% Mounted on
/dev/mapper/rhel_stlrx300s6--22--irmc-root xfs        94G   55G   39G
59% /
devtmpfs                                    devtmpfs   32G     0   32G
0% /dev
tmpfs                                       tmpfs      32G     0   32G
0% /dev/shm
tmpfs                                       tmpfs      32G  3.3G   29G
11% /run
tmpfs                                       tmpfs      32G     0   32G
0% /sys/fs/cgroup
/dev/sda7                                   xfs       9.4G  210M  9.1G
3% /boot
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/10065
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/10068
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/10069
10.63.150.213:/nc_volume3                   nfs4      380G  8.0M  380G
1% /mnt
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/0
gpfs1                                       gpfs      7.3T  9.1G  7.3T
1% /gpfs1
[root@mastr-51 gpfs1]#
[root@mastr-51 ~]# cd /gpfs1
[root@mastr-51 gpfs1]# ls
catalog  ces  gpfs-ces  ha  testfile
[root@mastr-51 gpfs1]#
[root@mastr-51 ~]# cd /gpfs1
[root@mastr-51 gpfs1]# ls
ces  gpfs-ces  ha  testfile
[root@mastr-51 gpfs1]# ls -ltrha
total 5.1G
dr-xr-xr-x   2 root root 8.0K Jan  1  1970 .snapshots
-rw-r--r--   1 root root 5.0G Oct  8 18:10 testfile
dr-xr-xr-x. 30 root root 4.0K Oct  8 18:19 ..
drwxr-xr-x   2 root root 4.0K Nov  5 20:02 gpfs-ces
drwxr-xr-x   2 root root 4.0K Nov  5 20:04 ha
drwxrwxrwx   5 root root 256K Nov  5 20:04 .
drwxr-xr-x   4 root root 4.0K Nov  5 20:35 ces
[root@mastr-51 gpfs1]#
```

**NFS** 클라이언트를 구성합니다

NFS 클라이언트를 구성하려면 다음 단계를 수행하십시오.

1. NFS 클라이언트에 패키지를 설치합니다.

```
[root@hdp2 ~]# yum install nfs-utils rpcbind
Loaded plugins: product-id, search-disabled-repos, subscription-manager
HDP-2.6-GPL-repo-4
| 2.9 kB  00:00:00
HDP-2.6-repo-4
| 2.9 kB  00:00:00
HDP-3.0-GPL-repo-2
| 2.9 kB  00:00:00
HDP-3.0-repo-2
| 2.9 kB  00:00:00
HDP-3.0-repo-3
| 2.9 kB  00:00:00
HDP-3.1-repo-1
| 2.9 kB  00:00:00
HDP-3.1-repo-51
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-1
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-2
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-3
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-4
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-51
| 2.9 kB  00:00:00
ambari-2.7.3.0
| 2.9 kB  00:00:00
epel/x86_64/metalink
|  13 kB  00:00:00
epel
| 5.3 kB  00:00:00
mysql-connectors-community
| 2.5 kB  00:00:00
mysql-tools-community
| 2.5 kB  00:00:00
mysql56-community
| 2.5 kB  00:00:00
rhel-7-server-optional-rpms
| 3.2 kB  00:00:00
```

```
rhel-7-server-rpms
| 3.5 kB  00:00:00
(1/10): mysql-connectors-community/x86_64/primary_db
|  49 kB  00:00:00
(2/10): mysql-tools-community/x86_64/primary_db
|  66 kB  00:00:00
(3/10): epel/x86_64/group_gz
|  90 kB  00:00:00
(4/10): mysql56-community/x86_64/primary_db
| 241 kB  00:00:00
(5/10): rhel-7-server-optional-rpms/7Server/x86_64/updateinfo
| 2.5 MB  00:00:00
(6/10): rhel-7-server-rpms/7Server/x86_64/updateinfo
| 3.4 MB  00:00:00
(7/10): rhel-7-server-optional-rpms/7Server/x86_64/primary_db
| 8.3 MB  00:00:00
(8/10): rhel-7-server-rpms/7Server/x86_64/primary_db
|  62 MB  00:00:01
(9/10): epel/x86_64/primary_db
| 6.9 MB  00:00:08
(10/10): epel/x86_64/updateinfo
| 1.0 MB  00:00:13
Resolving Dependencies
--> Running transaction check
---> Package nfs-utils.x86_64 1:1.3.0-0.61.el7 will be updated
---> Package nfs-utils.x86_64 1:1.3.0-0.65.el7 will be an update
---> Package rpcbind.x86_64 0:0.2.0-47.el7 will be updated
---> Package rpcbind.x86_64 0:0.2.0-48.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

=========================================================================
===============================================
 Package                 Arch                 Version
Repository                         Size
=========================================================================
===============================================
Updating:
 nfs-utils               x86_64               1:1.3.0-0.65.el7
rhel-7-server-rpms               412 k
 rpcbind                 x86_64               0.2.0-48.el7
rhel-7-server-rpms                60 k

Transaction Summary
=========================================================================
```

```
================================================
Upgrade  2 Packages

Total download size: 472 k
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
(1/2): rpcbind-0.2.0-48.el7.x86_64.rpm
|  60 kB  00:00:00
(2/2): nfs-utils-1.3.0-0.65.el7.x86_64.rpm
| 412 kB  00:00:00
--------------------------------------------------------------------------
-----------------------------------------------
Total
1.2 MB/s | 472 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating   : rpcbind-0.2.0-48.el7.x86_64
1/4
service rpcbind start

  Updating   : 1:nfs-utils-1.3.0-0.65.el7.x86_64
2/4
  Cleanup    : 1:nfs-utils-1.3.0-0.61.el7.x86_64
3/4
  Cleanup    : rpcbind-0.2.0-47.el7.x86_64
4/4
  Verifying  : 1:nfs-utils-1.3.0-0.65.el7.x86_64
1/4
  Verifying  : rpcbind-0.2.0-48.el7.x86_64
2/4
  Verifying  : rpcbind-0.2.0-47.el7.x86_64
3/4
  Verifying  : 1:nfs-utils-1.3.0-0.61.el7.x86_64
4/4

Updated:
  nfs-utils.x86_64 1:1.3.0-0.65.el7
rpcbind.x86_64 0:0.2.0-48.el7

Complete!
[root@hdp2 ~]#
```

2. NFS 클라이언트 서비스를 시작합니다.

```
[root@hdp2 ~]# service rpcbind start
Redirecting to /bin/systemctl start rpcbind.service
 [root@hdp2 ~]#
```

3. NFS 클라이언트의 NFS 프로토콜을 통해 GPFS를 마운트합니다.

```
[root@hdp2 ~]# mkdir /gpfstest
[root@hdp2 ~]# mount 10.63.150.51:/gpfs1 /gpfstest
[root@hdp2 ~]# df -h
Filesystem                              Size  Used Avail Use% Mounted on
/dev/mapper/rhel_stlrx300s6--22-root  1.1T  113G  981G  11% /
devtmpfs                                126G     0  126G   0% /dev
tmpfs                                   126G   16K  126G   1% /dev/shm
tmpfs                                   126G  510M  126G   1% /run
tmpfs                                   126G     0  126G   0%
/sys/fs/cgroup
/dev/sdd2                               197M  191M  6.6M  97% /boot
tmpfs                                    26G     0   26G   0% /run/user/0
10.63.150.213:/nc_volume2                95G  5.4G   90G   6% /mnt
10.63.150.51:/gpfs1                     7.3T  9.1G  7.3T   1% /gpfstest
[root@hdp2 ~]#
```

4. NFS 마운트 폴더에서 GPFS 파일 목록을 검증합니다.

```
[root@hdp2 ~]# cd /gpfstest/
[root@hdp2 gpfstest]# ls
ces  gpfs-ces  ha  testfile
[root@hdp2 gpfstest]# ls -l
total 5242882
drwxr-xr-x 4 root root       4096 Nov  5 15:35 ces
drwxr-xr-x 2 root root       4096 Nov  5 15:02 gpfs-ces
drwxr-xr-x 2 root root       4096 Nov  5 15:04 ha
-rw-r--r-- 1 root root 5368709120 Oct  8 14:10 testfile
[root@hdp2 gpfstest]#
```

5. XCP를 사용하여 GPFS로 내보낸 NFS에서 NetApp NFS로 데이터를 이동하십시오.

```
[root@hdp2 linux]# ./xcp copy -parallel 20 10.63.150.51:/gpfs1
10.63.150.213:/nc_volume2/
XCP 1.4-17914d6; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Tue Nov  5 12:39:36 2019

xcp: WARNING: your license will expire in less than one week! You can
renew your license at https://xcp.netapp.com
xcp: open or create catalog 'xcp': Creating new catalog in
'10.63.150.51:/gpfs1/catalog'
xcp: WARNING: No index name has been specified, creating one with name:
autoname_copy_2019-11-11_12.14.07.805223
xcp: mount '10.63.150.51:/gpfs1': WARNING: This NFS server only supports
1-second timestamp granularity. This may cause sync to fail because
changes will often be undetectable.
 34 scanned, 32 copied, 32 indexed, 1 giant, 301 MiB in (59.5 MiB/s),
784 KiB out (155 KiB/s), 6s
 34 scanned, 32 copied, 32 indexed, 1 giant, 725 MiB in (84.6 MiB/s),
1.77 MiB out (206 KiB/s), 11s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.17 GiB in (94.2 MiB/s),
2.90 MiB out (229 KiB/s), 16s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.56 GiB in (79.8 MiB/s),
3.85 MiB out (194 KiB/s), 21s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.95 GiB in (78.4 MiB/s),
4.80 MiB out (191 KiB/s), 26s
 34 scanned, 32 copied, 32 indexed, 1 giant, 2.35 GiB in (80.4 MiB/s),
5.77 MiB out (196 KiB/s), 31s
 34 scanned, 32 copied, 32 indexed, 1 giant, 2.79 GiB in (89.6 MiB/s),
6.84 MiB out (218 KiB/s), 36s
 34 scanned, 32 copied, 32 indexed, 1 giant, 3.16 GiB in (75.3 MiB/s),
7.73 MiB out (183 KiB/s), 41s
 34 scanned, 32 copied, 32 indexed, 1 giant, 3.53 GiB in (75.4 MiB/s),
8.64 MiB out (183 KiB/s), 46s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.00 GiB in (94.4 MiB/s),
9.77 MiB out (230 KiB/s), 51s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.46 GiB in (94.3 MiB/s),
10.9 MiB out (229 KiB/s), 56s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.86 GiB in (80.2 MiB/s),
11.9 MiB out (195 KiB/s), 1m1s
Sending statistics...
34 scanned, 33 copied, 34 indexed, 1 giant, 5.01 GiB in (81.8 MiB/s),
12.3 MiB out (201 KiB/s), 1m2s.
[root@hdp2 linux]#
```

6. NFS 클라이언트에서 GPFS 파일을 검증합니다.

```
[root@hdp2 mnt]# df -Th
Filesystem                          Type      Size  Used Avail Use%
Mounted on
/dev/mapper/rhel_stlrx300s6--22-root xfs       1.1T  113G  981G  11% /
devtmpfs                            devtmpfs  126G     0  126G   0%
/dev
tmpfs                               tmpfs     126G   16K  126G   1%
/dev/shm
tmpfs                               tmpfs     126G  518M  126G   1%
/run
tmpfs                               tmpfs     126G     0  126G   0%
/sys/fs/cgroup
/dev/sdd2                           xfs       197M  191M  6.6M  97%
/boot
tmpfs                               tmpfs      26G     0   26G   0%
/run/user/0
10.63.150.213:/nc_volume2           nfs4       95G  5.4G   90G   6%
/mnt
10.63.150.51:/gpfs1                 nfs4      7.3T  9.1G  7.3T   1%
/gpfstest
[root@hdp2 mnt]#
[root@hdp2 mnt]# ls -ltrha
total 128K
dr-xr-xr-x   2 root        root               4.0K Dec 31  1969
.snapshots
drwxrwxrwx   2 root        root               4.0K Feb 14  2018 data
drwxrwxrwx   3 root        root               4.0K Feb 14  2018
wcresult
drwxrwxrwx   3 root        root               4.0K Feb 14  2018
wcresult1
drwxrwxrwx   2 root        root               4.0K Feb 14  2018
wcresult2
drwxrwxrwx   2 root        root               4.0K Feb 16  2018
wcresult3
-rw-r--r--   1 root        root               2.8K Feb 20  2018
README demo
drwxrwxrwx   3 root        root               4.0K Jun 28 13:38 scantg
drwxrwxrwx   3 root        root               4.0K Jun 28 13:39
scancopyFromLocal
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f3
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 README
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f9
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f6
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f5
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:30 f4
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:30 f8
```

```
-rw-r--r--   1 hdfs        hadoop              1.2K Jul  3 19:30 f2
-rw-r--r--   1 hdfs        hadoop              1.2K Jul  3 19:30 f7
drwxrwxrwx   2 root        root                4.0K Jul  9 11:14 test
drwxrwxrwx   3 root        root                4.0K Jul 10 16:35
warehouse
drwxr-xr-x   3       10061 tester1             4.0K Jul 15 14:40 sdd1
drwxrwxrwx   3 testeruser1 hadoopkerberosgroup 4.0K Aug 20 17:00
kermkdir
-rw-r--r--   1 testeruser1 hadoopkerberosgroup    0 Aug 21 14:20 newfile
drwxrwxrwx   2 testeruser1 hadoopkerberosgroup 4.0K Aug 22 10:13
teragen1copy_3
drwxrwxrwx   2 testeruser1 hadoopkerberosgroup 4.0K Aug 22 10:33
teragen2copy_1
-rw-rwxr--   1 root        hdfs                1.2K Sep 19 16:38 R1
drwx------   3 root        root                4.0K Sep 20 17:28 user
-rw-r--r--   1 root        root                5.0G Oct  8 14:10
testfile
drwxr-xr-x   2 root        root                4.0K Nov  5 15:02 gpfs-
ces
drwxr-xr-x   2 root        root                4.0K Nov  5 15:04 ha
drwxr-xr-x   4 root        root                4.0K Nov  5 15:35 ces
dr-xr-xr-x. 26 root        root                4.0K Nov  6 11:40 ..
drwxrwxrwx  21 root        root                4.0K Nov 11 12:14 .
drwxrwxrwx   7 nobody      nobody              4.0K Nov 11 12:14 catalog
[root@hdp2 mnt]#
```

## MapR-FS에서 ONTAP NFS로

이 섹션에서는 NetApp XCP를 사용하여 MapR-FS 데이터를 ONTAP NFS로 이동하는 데 필요한 자세한 단계를 제공합니다.

1. 각 MapR 노드에 대해 LUN 3개를 프로비저닝하고 모든 MapR 노드에 대한 LUN 소유권을 제공합니다.
2. 설치 중에 MapR-FS에 사용되는 MapR 클러스터 디스크에 새로 추가된 LUN을 선택합니다.
3. 에 따라 MapR 클러스터를 설치합니다 "MapR 6.1 문서".
4. Hadoop Jar xxx와 같은 MapReduce 명령을 사용하여 기본 Hadoop 작업을 확인합니다.
5. 고객 데이터를 MapR-FS에 유지 예를 들어, Teragen을 사용하여 MapR-FS에서 약 테라바이트의 샘플 데이터를 생성했습니다.
6. MapR-FS를 NFS 내보내기로 구성합니다.
   a. 모든 MapR 노드에서 nlockmgr 서비스를 사용하지 않도록 설정합니다.

```
root@workr-138: ~$ rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100003    4   tcp   2049  nfs
    100227    3   tcp   2049  nfs_acl
    100003    4   udp   2049  nfs
    100227    3   udp   2049  nfs_acl
    100021    3   udp  55270  nlockmgr
    100021    4   udp  55270  nlockmgr
    100021    3   tcp  35025  nlockmgr
    100021    4   tcp  35025  nlockmgr
    100003    3   tcp   2049  nfs
    100005    3   tcp   2049  mountd
    100005    1   tcp   2049  mountd
    100005    3   udp   2049  mountd
    100005    1   udp   2049  mountd
root@workr-138: ~$

root@workr-138: ~$ rpcinfo -d 100021 3
root@workr-138: ~$ rpcinfo -d 100021 4
```

b. '/opt/mMapR/conf/exports' 파일의 모든 MapR 노드에 MapR-FS에서 특정 폴더를 내보냅니다. 하위 폴더를 내보낼 때 다른 권한이 있는 상위 폴더를 내보내지 마십시오.

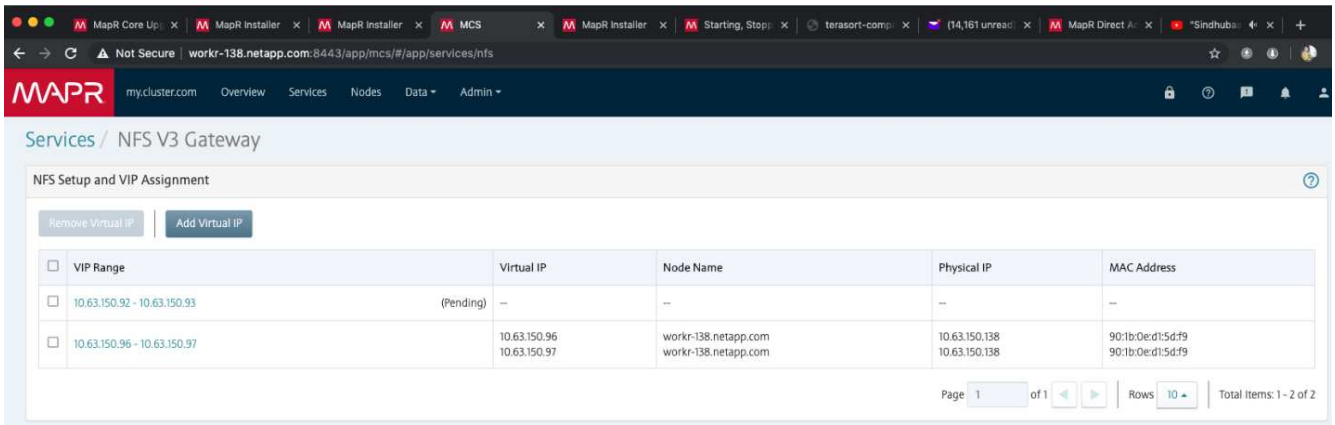
```
[mapr@workr-138 ~]$ cat /opt/mapr/conf/exports
# Sample Exports file
# for /mapr exports
# <Path> <exports_control>
#access_control -> order is specific to default
# list the hosts before specifying a default for all
#  a.b.c.d,1.2.3.4(ro) d.e.f.g(ro) (rw)
#  enforces ro for a.b.c.d & 1.2.3.4 and everybody else is rw
# special path to export clusters in mapr-clusters.conf. To disable
exporting,
# comment it out. to restrict access use the exports_control
#
#/mapr (rw)
#karthik
/mapr/my.cluster.com/tmp/testnfs /maprnfs3 (rw)
#to export only certain clusters, comment out the /mapr & uncomment.
#/mapr/clustername (rw)
#to export /mapr only to certain hosts (using exports_control)
#/mapr a.b.c.d(rw),e.f.g.h(ro)
# export /mapr/cluster1 rw to a.b.c.d & ro to e.f.g.h (denied for
others)
#/mapr/cluster1 a.b.c.d(rw),e.f.g.h(ro)
# export /mapr/cluster2 only to e.f.g.h (denied for others)
#/mapr/cluster2 e.f.g.h(rw)
# export /mapr/cluster3 rw to e.f.g.h & ro to others
#/mapr/cluster2 e.f.g.h(rw) (ro)
#to export a certain cluster, volume or a subdirectory as an alias,
#comment out  /mapr & uncomment
#/mapr/clustername         /alias1 (rw)
#/mapr/clustername/vol     /alias2 (rw)
#/mapr/clustername/vol/dir /alias3 (rw)
#only the alias will be visible/exposed to the nfs client not the
mapr path, host options as before
[mapr@workr-138 ~]$
```

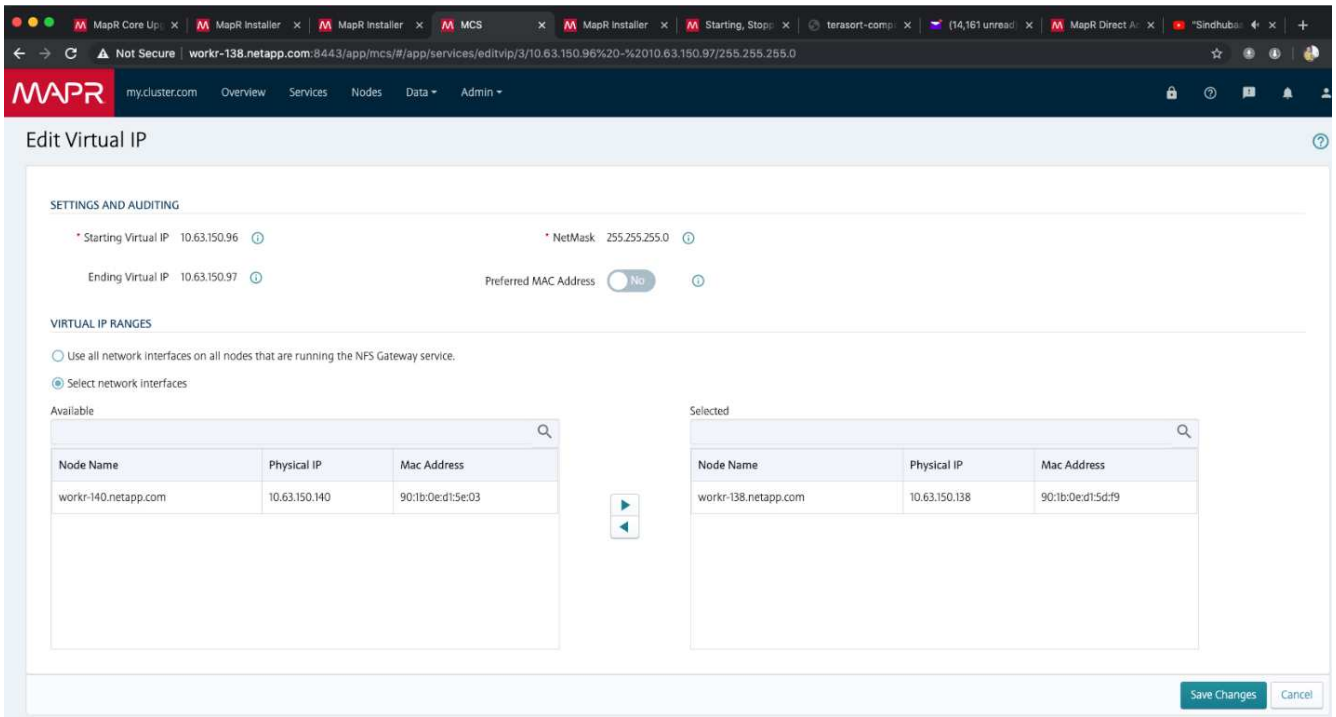
7. MapR-FS NFS 서비스를 새로 고칩니다.

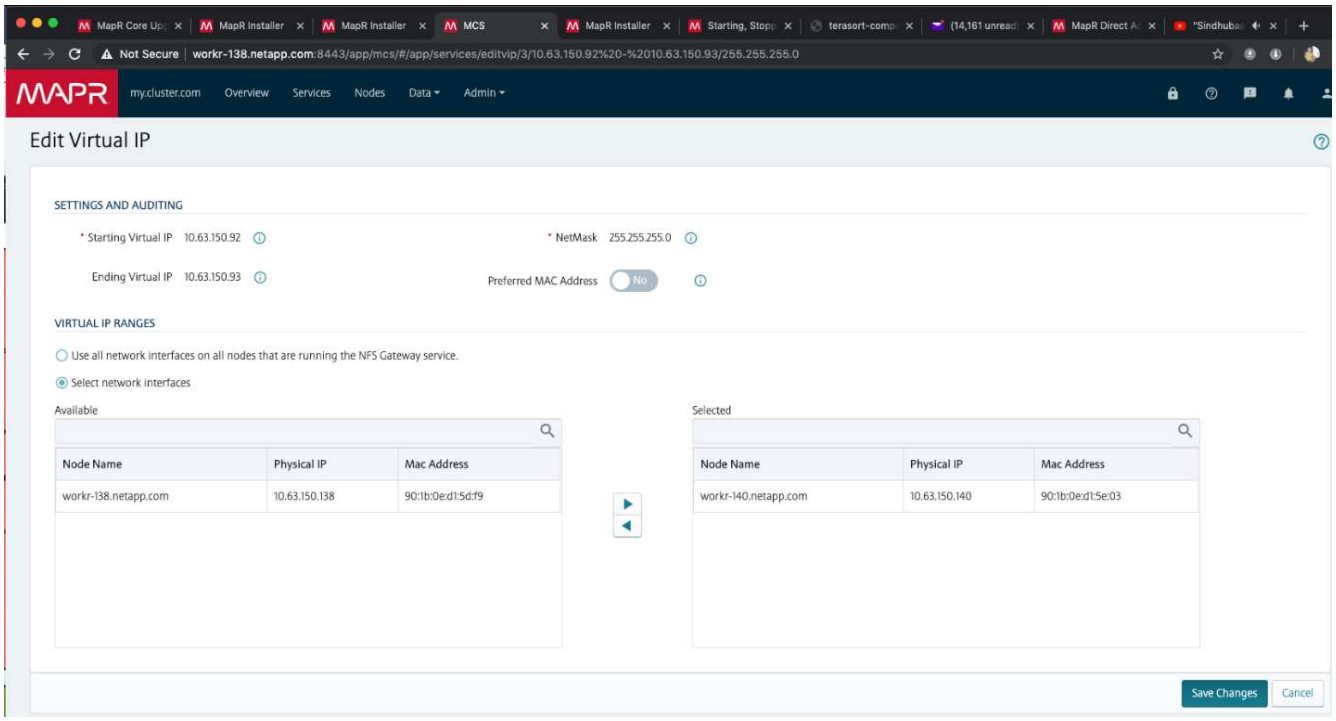
```
root@workr-138: tmp$ maprcli nfsmgmt refreshexports
ERROR (22) -  You do not have a ticket to communicate with
127.0.0.1:9998. Retry after obtaining a new ticket using maprlogin
root@workr-138: tmp$ su - mapr
[mapr@workr-138 ~]$ maprlogin password -cluster my.cluster.com
[Password for user 'mapr' at cluster 'my.cluster.com': ]
MapR credentials of user 'mapr' for cluster 'my.cluster.com' are written
to '/tmp/maprticket_5000'
[mapr@workr-138 ~]$ maprcli nfsmgmt refreshexports
```

8. MapR 클러스터의 특정 서버 또는 서버 세트에 가상 IP 범위를 할당합니다. 그런 다음 MapR 클러스터는 NFS 데이터 액세스를 위해 특정 서버에 IP를 할당합니다. IP를 통해 고가용성을 구현할 수 있습니다. 즉, 특정 IP를 사용하는 서버 또는 네트워크에 장애가 발생할 경우 IP 범위의 다음 IP를 NFS 액세스에 사용할 수 있습니다.

> 모든 MapR 노드에서 NFS 액세스를 제공하려면 각 서버에 가상 IP 세트를 할당하고 NFS 데이터 액세스를 위해 각 MapR 노드의 리소스를 사용할 수 있습니다.






9. 각 MapR 노드에 할당된 가상 IP를 확인하고 이를 NFS 데이터 액세스에 사용하십시오.

```
root@workr-138: ~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: ens3f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5d:f9 brd ff:ff:ff:ff:ff:ff
    inet 10.63.150.138/24 brd 10.63.150.255 scope global noprefixroute
ens3f0
       valid_lft forever preferred_lft forever
    inet 10.63.150.96/24 scope global secondary ens3f0:~m0
       valid_lft forever preferred_lft forever
    inet 10.63.150.97/24 scope global secondary ens3f0:~m1
       valid_lft forever preferred_lft forever
    inet6 fe80::921b:eff:fed1:5df9/64 scope link
       valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:af:b4 brd ff:ff:ff:ff:ff:ff
4: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5d:fa brd ff:ff:ff:ff:ff:ff
5: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state
DOWN group default qlen 1000
    link/ether 90:1b:0e:d1:af:b5 brd ff:ff:ff:ff:ff:ff
[root@workr-138: ~$
[root@workr-140 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5e:03 brd ff:ff:ff:ff:ff:ff
    inet 10.63.150.140/24 brd 10.63.150.255 scope global noprefixroute
ens3f0
       valid_lft forever preferred_lft forever
    inet 10.63.150.92/24 scope global secondary ens3f0:~m0
       valid_lft forever preferred_lft forever
    inet6 fe80::921b:eff:fed1:5e03/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:af:9a brd ff:ff:ff:ff:ff:ff
4: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
```

```
    link/ether 90:1b:0e:d1:5e:04 brd ff:ff:ff:ff:ff:ff
5: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state
DOWN group default qlen 1000
    link/ether 90:1b:0e:d1:af:9b brd ff:ff:ff:ff:ff:ff
[root@workr-140 ~]#
```

10. NFS 작업을 확인하기 위해 할당된 가상 IP를 사용하여 NFS 내보내기 MapR-FS를 마운트합니다. 하지만 NetApp XCP를 사용하여 데이터를 전송하는 경우 이 단계가 필요하지 않습니다.

```
root@workr-138: tmp$ mount -v -t nfs 10.63.150.92:/maprnfs3
/tmp/testmount/
mount.nfs: timeout set for Thu Dec  5 15:31:32 2019
mount.nfs: trying text-based options
'vers=4.1,addr=10.63.150.92,clientaddr=10.63.150.138'
mount.nfs: mount(2): Protocol not supported
mount.nfs: trying text-based options
'vers=4.0,addr=10.63.150.92,clientaddr=10.63.150.138'
mount.nfs: mount(2): Protocol not supported
mount.nfs: trying text-based options 'addr=10.63.150.92'
mount.nfs: prog 100003, trying vers=3, prot=6
mount.nfs: trying 10.63.150.92 prog 100003 vers 3 prot TCP port 2049
mount.nfs: prog 100005, trying vers=3, prot=17
mount.nfs: trying 10.63.150.92 prog 100005 vers 3 prot UDP port 2049
mount.nfs: portmap query retrying: RPC: Timed out
mount.nfs: prog 100005, trying vers=3, prot=6
mount.nfs: trying 10.63.150.92 prog 100005 vers 3 prot TCP port 2049
root@workr-138: tmp$ df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/sda7                84G   48G   37G  57% /
devtmpfs                126G     0  126G   0% /dev
tmpfs                   126G     0  126G   0% /dev/shm
tmpfs                   126G   19M  126G   1% /run
tmpfs                   126G     0  126G   0% /sys/fs/cgroup
/dev/sdd1               3.7T  201G  3.5T   6% /mnt/sdd1
/dev/sda6               946M  220M  726M  24% /boot
tmpfs                    26G     0   26G   0% /run/user/5000
gpfs1                   7.3T  9.1G  7.3T   1% /gpfs1
tmpfs                    26G     0   26G   0% /run/user/0
localhost:/mapr         100G     0  100G   0% /mapr
10.63.150.92:/maprnfs3   53T  8.4G   53T   1% /tmp/testmount
root@workr-138: tmp$
```

11. MapR-FS NFS 게이트웨이에서 ONTAP NFS로 데이터를 전송하도록 NetApp XCP를 구성합니다.
    a. xCP에 대한 카탈로그 위치를 구성합니다.

```
[root@hdp2 linux]# cat /opt/NetApp/xFiles/xcp/xcp.ini
# Sample xcp config
[xcp]
#catalog =  10.63.150.51:/gpfs1
catalog =  10.63.150.213:/nc_volume1
```

b. 라이센스 파일을 '/opt/netapp/xFiles/xCP/'에 복사합니다.

```
root@workr-138: src$ cd /opt/NetApp/xFiles/xcp/
root@workr-138: xcp$ ls -ltrha
total 252K
drwxr-xr-x 3 root   root     16 Apr  4  2019 ..
-rw-r--r-- 1 root   root    105 Dec  5 19:04 xcp.ini
drwxr-xr-x 2 root   root     59 Dec  5 19:04 .
-rw-r--r-- 1 faiz89 faiz89  336 Dec  6 21:12 license
-rw-r--r-- 1 root   root    192 Dec  6 21:13 host
-rw-r--r-- 1 root   root   236K Dec 17 14:12 xcp.log
root@workr-138: xcp$
```

c. xCP activate 명령을 사용하여 xCP를 활성화합니다.

d. NFS 내보내기에 대한 소스를 확인합니다.

```
[root@hdp2 linux]# ./xcp show 10.63.150.92
XCP 1.4-17914d6; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
getting pmap dump from 10.63.150.92 port 111...
getting export list from 10.63.150.92...
sending 1 mount and 4 nfs requests to 10.63.150.92...
== RPC Services ==
'10.63.150.92': TCP rpc services: MNT v1/3, NFS v3/4, NFSACL v3, NLM
v1/3/4, PMAP v2/3/4, STATUS v1
'10.63.150.92': UDP rpc services: MNT v1/3, NFS v4, NFSACL v3, NLM
v1/3/4, PMAP v2/3/4, STATUS v1
== NFS Exports ==
 Mounts  Errors  Server
      1       0  10.63.150.92
     Space    Files      Space    Files
      Free     Free       Used     Used Export
  52.3 TiB    53.7B   8.36 GiB    53.7B 10.63.150.92:/maprnfs3
== Attributes of NFS Exports ==
drwxr-xr-x --- root root 2 2 10m51s 10.63.150.92:/maprnfs3
1.77 KiB in (8.68 KiB/s), 3.16 KiB out (15.5 KiB/s), 0s.
[root@hdp2 linux]#
```

e. 여러 소스 IP 및 여러 대상 IP(ONTAP LIF)에서 여러 MapR 노드에서 XCP를 사용하여 데이터를 전송합니다.

```
root@workr-138: linux$ ./xcp_yatin copy --parallel 20
10.63.150.96,10.63.150.97:/maprnfs3/tg4
10.63.150.85,10.63.150.86:/datapipeline_dataset/tg4_dest
XCP 1.6-dev; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
xcp: WARNING: No index name has been specified, creating one with
name: autoname_copy_2019-12-06_21.14.38.652652
xcp: mount '10.63.150.96,10.63.150.97:/maprnfs3/tg4': WARNING: This
NFS server only supports 1-second timestamp granularity. This may
cause sync to fail because changes will often be undetectable.
 130 scanned, 128 giants, 3.59 GiB in (723 MiB/s), 3.60 GiB out (724
MiB/s), 5s
 130 scanned, 128 giants, 8.01 GiB in (889 MiB/s), 8.02 GiB out (890
MiB/s), 11s
 130 scanned, 128 giants, 12.6 GiB in (933 MiB/s), 12.6 GiB out (934
MiB/s), 16s
 130 scanned, 128 giants, 16.7 GiB in (830 MiB/s), 16.7 GiB out (831
MiB/s), 21s
 130 scanned, 128 giants, 21.1 GiB in (907 MiB/s), 21.1 GiB out (908
MiB/s), 26s
```

```
 130 scanned, 128 giants, 25.5 GiB in (893 MiB/s), 25.5 GiB out (894
MiB/s), 31s
 130 scanned, 128 giants, 29.6 GiB in (842 MiB/s), 29.6 GiB out (843
MiB/s), 36s
….
[root@workr-140 linux]# ./xcp_yatin copy  --parallel 20
10.63.150.92:/maprnfs3/tg4_2
10.63.150.85,10.63.150.86:/datapipeline_dataset/tg4_2_dest
XCP 1.6-dev; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
xcp: WARNING: No index name has been specified, creating one with
name: autoname_copy_2019-12-06_21.14.24.637773
xcp: mount '10.63.150.92:/maprnfs3/tg4_2': WARNING: This NFS server
only supports 1-second timestamp granularity. This may cause sync to
fail because changes will often be undetectable.
 130 scanned, 128 giants, 4.39 GiB in (896 MiB/s), 4.39 GiB out (897
MiB/s), 5s
 130 scanned, 128 giants, 9.94 GiB in (1.10 GiB/s), 9.96 GiB out
(1.10 GiB/s), 10s
 130 scanned, 128 giants, 15.4 GiB in (1.09 GiB/s), 15.4 GiB out
(1.09 GiB/s), 15s
 130 scanned, 128 giants, 20.1 GiB in (953 MiB/s), 20.1 GiB out (954
MiB/s), 20s
 130 scanned, 128 giants, 24.6 GiB in (928 MiB/s), 24.7 GiB out (929
MiB/s), 25s
 130 scanned, 128 giants, 29.0 GiB in (877 MiB/s), 29.0 GiB out (878
MiB/s), 31s
 130 scanned, 128 giants, 33.2 GiB in (852 MiB/s), 33.2 GiB out (853
MiB/s), 36s
 130 scanned, 128 giants, 37.8 GiB in (941 MiB/s), 37.8 GiB out (942
MiB/s), 41s
 130 scanned, 128 giants, 42.0 GiB in (860 MiB/s), 42.0 GiB out (861
MiB/s), 46s
 130 scanned, 128 giants, 46.1 GiB in (852 MiB/s), 46.2 GiB out (853
MiB/s), 51s
 130 scanned, 128 giants, 50.1 GiB in (816 MiB/s), 50.2 GiB out (817
MiB/s), 56s
 130 scanned, 128 giants, 54.1 GiB in (819 MiB/s), 54.2 GiB out (820
MiB/s), 1m1s
 130 scanned, 128 giants, 58.5 GiB in (897 MiB/s), 58.6 GiB out (898
MiB/s), 1m6s
 130 scanned, 128 giants, 62.9 GiB in (900 MiB/s), 63.0 GiB out (901
MiB/s), 1m11s
 130 scanned, 128 giants, 67.2 GiB in (876 MiB/s), 67.2 GiB out (877
MiB/s), 1m16s
```

f. 스토리지 컨트롤러의 로드 분산을 확인합니다.

```
Hadoop-AFF8080::*> statistics show-periodic -interval 2 -iterations 0
-summary true -object nic_common -counter rx_bytes|tx_bytes -node
Hadoop-AFF8080-01 -instance e3b
Hadoop-AFF8080: nic_common.e3b: 12/6/2019 15:55:04
 rx_bytes tx_bytes
 -------- --------
    879MB   4.67MB
    856MB   4.46MB
    973MB   5.66MB
    986MB   5.88MB
    945MB   5.30MB
    920MB   4.92MB
    894MB   4.76MB
    902MB   4.79MB
    886MB   4.68MB
    892MB   4.78MB
    908MB   4.96MB
    905MB   4.85MB
    899MB   4.83MB
Hadoop-AFF8080::*> statistics show-periodic -interval 2 -iterations 0
-summary true -object nic_common -counter rx_bytes|tx_bytes -node
Hadoop-AFF8080-01 -instance e9b
Hadoop-AFF8080: nic_common.e9b: 12/6/2019 15:55:07
 rx_bytes tx_bytes
 -------- --------
    950MB   4.93MB
    991MB   5.84MB
    959MB   5.63MB
    914MB   5.06MB
    903MB   4.81MB
    899MB   4.73MB
    892MB   4.71MB
    890MB   4.72MB
    905MB   4.86MB
    902MB   4.90MB
```

## 추가 정보를 찾을 수 있는 위치

이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹 사이트를 검토하십시오.

- NetApp 데이터 이동 없는 분석 모듈 모범 사례

  "https://www.netapp.com/us/media/tr-4382.pdf"

- NetApp FlexGroup 볼륨 모범 사례 및 구현 가이드 를 참조하십시오

  "https://www.netapp.com/us/media/tr-4571.pdf"

- NetApp 제품 설명서

  https://www.netapp.com/us/documentation/index.aspx

# Confluent Kafka 모범 사례

## TR-4912: NetApp을 통해 Confluent Kafka 계층형 스토리지를 위한 모범 사례 지침

Karthkeyan Nagalingam, Joseph Kandatillarambil, NetApp Rankesh Kumar, Confluent

Apache Kafka는 매일 수조 건의 이벤트를 처리할 수 있는 커뮤니티 분산 이벤트 스트리밍 플랫폼입니다. 처음에 메시징 큐로 구상된 Kafka는 분산 커밋 로그의 추상화를 기반으로 합니다. Kafka는 2011년 LinkedIn에서 제작 및 오픈 소스를 통해 메시지 큐에서 완전한 이벤트 스트리밍 플랫폼으로 발전했습니다. Confluent는 Apache Kafka를 Confluent Platform과 함께 배포할 수 있도록 합니다. Confluent Platform은 Kafka를 보완하는 추가적인 커뮤니티 및 상용 기능을 제공하여 대규모 생산 환경에서 운영자와 개발자 모두의 스트리밍 경험을 개선하도록 설계되었습니다.

이 문서에서는 다음 내용을 제공하여 NetApp의 오브젝트 스토리지 제품에서 Confluent Tiered Storage를 사용하기 위한 모범 사례 지침을 설명합니다.

- NetApp 오브젝트 스토리지를 위한 Confluent 검증 – NetApp StorageGRID
- 계층형 스토리지 성능 테스트
- NetApp 스토리지 시스템에 대한 Confluent의 모범 사례 지침

#### ContFluent Tiered Storage를 선택해야 하는 이유

Confluent는 많은 애플리케이션, 특히 빅데이터, 분석 및 스트리밍 워크로드를 위한 기본 실시간 스트리밍 플랫폼이 되었습니다. 계층적 스토리지를 사용하면 사용자가 Confluent 플랫폼의 스토리지에서 컴퓨팅을 분리할 수 있습니다. Data Fabric을 활용하면 데이터를 보다 경제적으로 저장하고, 사실상 무한대의 데이터를 저장하고, 필요 시 워크로드를 스케일업(또는 스케일다운) 할 수 있으며, 데이터 및 테넌트 재조정과 같은 관리 작업을 더 쉽게 수행할 수 있습니다. S3 호환 스토리지 시스템은 이러한 모든 기능을 활용하여 모든 이벤트의 데이터를 한 곳에서 대중화할 수 있으며, 복잡한 데이터 엔지니어링이 필요하지 않습니다. Kafka에 계층형 스토리지를 사용해야 하는 이유에 대한 자세한 내용은 을 참조하십시오 "이 기사는 Confluent에 의해 작성되었습니다".

#### NetApp StorageGRID를 계층형 스토리지로 선택해야 하는 이유

StorageGRID은 NetApp에서 제공하는 업계 최고의 오브젝트 스토리지 플랫폼입니다. StorageGRID은 Amazon S3(Simple Storage Service) API를 비롯한 업계 표준 오브젝트 API를 지원하는 소프트웨어 정의 오브젝트 기반 스토리지 솔루션입니다. StorageGRID는 비정형 데이터를 대규모로 저장 및 관리하여 안전하고 내구성 있는 오브젝트 스토리지를 제공합니다. 콘텐츠가 적절한 위치, 적합한 시간 및 적합한 스토리지 계층에 배치되어 전 세계적으로 분산된 다양한 미디어의 워크플로우를 최적화하고 비용을 줄입니다.

StorageGRID의 가장 큰 차별화 요소는 정책 기반 데이터 라이프사이클 관리를 지원하는 ILM(정보 라이프사이클 관리) 정책 엔진입니다. 정책 엔진은 메타데이터를 사용해 수명 주기 동안 데이터가 저장되는 방식을 관리하여 처음에 성능을

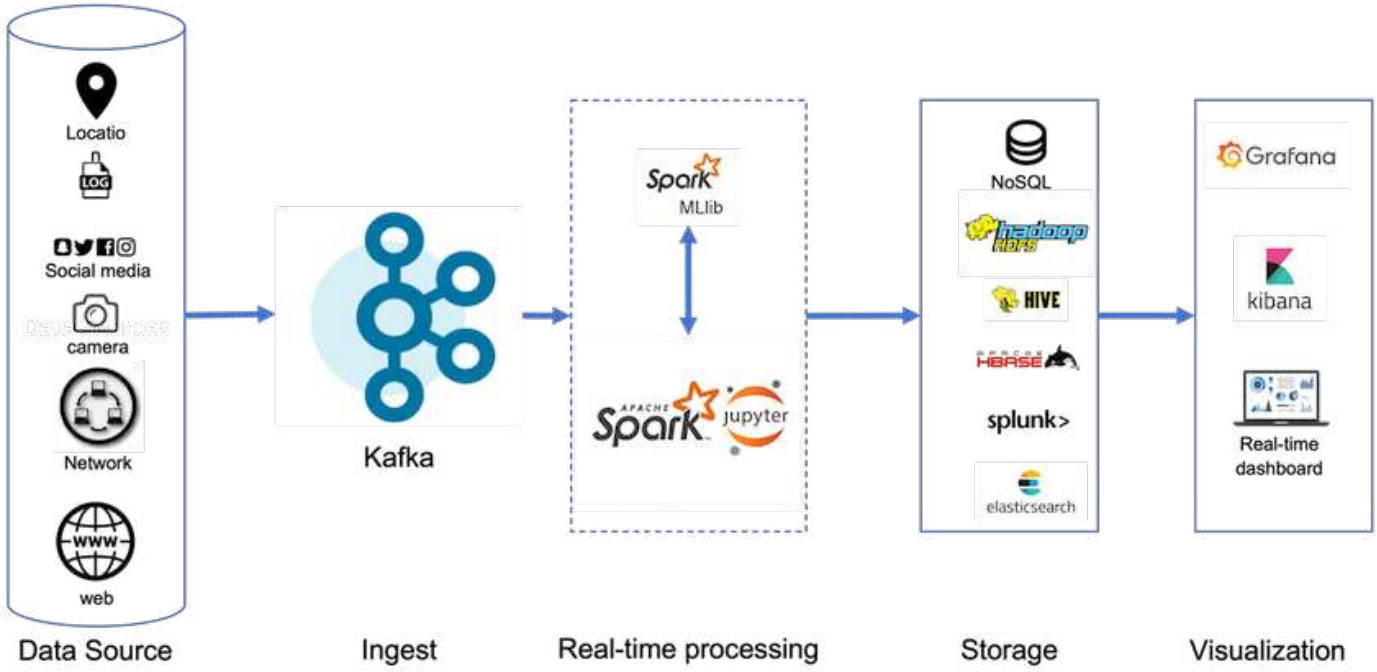
최적화하고 데이터 사용 기간에 따라 비용 및 내구성을 자동으로 최적화할 수 있습니다.

**ContFluent Tiered Storage** 활성화

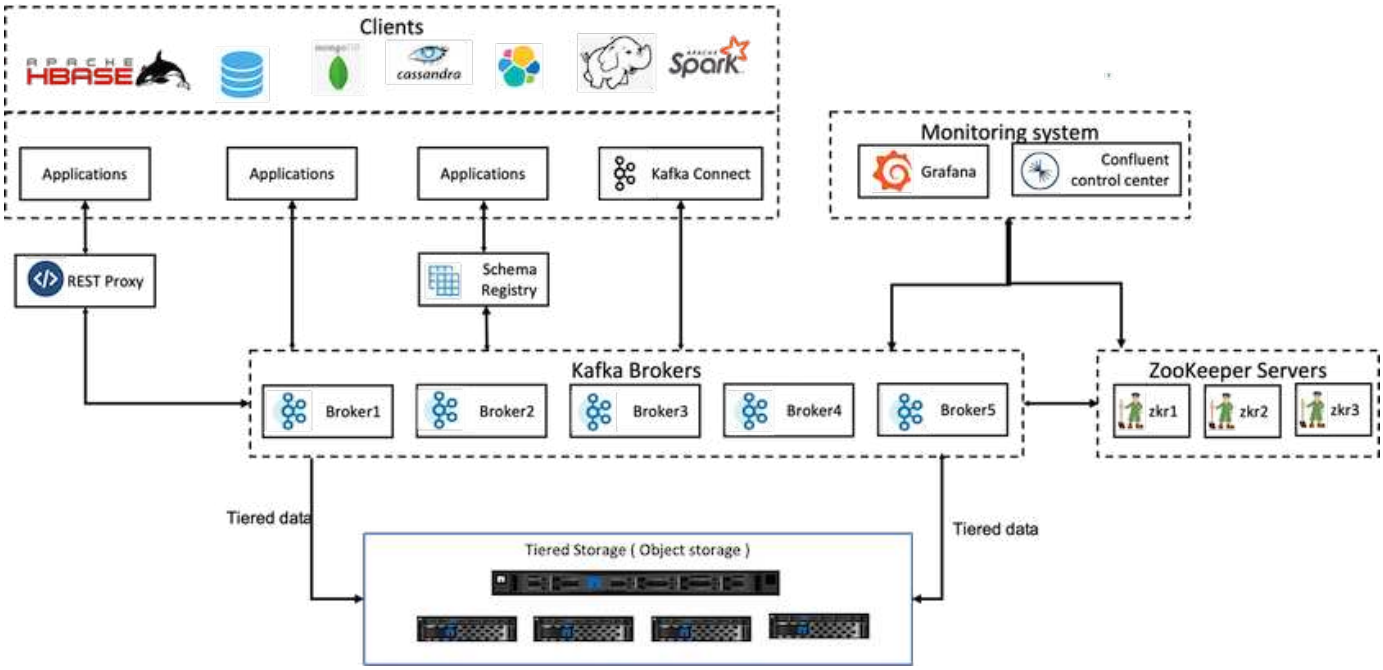
계층형 스토리지의 기본 개념은 데이터 스토리지와 데이터 처리 작업을 분리하는 것입니다. 이와 같은 분리 덕분에 데이터 스토리지 계층과 데이터 처리 계층이 독립적으로 확장하는 것이 훨씬 쉬워졌습니다.

Confluent를 위한 계층형 스토리지 솔루션은 두 가지 요소를 고려해야 합니다. 먼저, 목록 작업의 일관성 오류, 간헐적인 개체 가용성 손실 등과 같은 일반적인 개체 저장소의 일관성 및 가용성 속성을 문제를 해결 또는 방지해야 합니다. 두 번째로, 이 제품은 계층형 스토리지와 Kafka의 복제 및 내결함성 모델 간의 상호 작용을 올바르게 처리해야 하며, 이러한 상호 작용에는 좀비 리더가 계속해서 오프셋 범위를 계층화할 수 있는 가능성이 포함됩니다. NetApp 오브젝트 스토리지는 일관된 오브젝트 가용성과 HA 모델을 모두 제공하므로 피로한 스토리지를 계층 오프셋 범위에 사용할 수 있습니다. NetApp 오브젝트 스토리지는 일관된 오브젝트 가용성과 HA 모델을 제공하여 지친 스토리지를 계층 오프셋 범위에 사용할 수 있도록 합니다.

계층형 스토리지를 사용하면 스트리밍 데이터의 지연 시간이 짧은 읽기 및 쓰기 작업에 고성능 플랫폼을 사용할 수 있으며, 처리량이 높은 내역 읽기를 위해 NetApp StorageGRID와 같은 경제적이고 확장 가능한 오브젝트 저장소를 사용할 수도 있습니다. NetApp은 Spark with NetApp 스토리지 컨트롤러 및 세부 정보를 위한 기술 솔루션도 갖추고 있습니다. 다음 그림은 Kafka가 실시간 분석 파이프라인에 어떻게 부합하는지를 보여줍니다.




다음 그림은 NetApp StorageGRID가 Confluent Kafka의 오브젝트 스토리지 계층으로 적합한 방식을 보여줍니다.




## 솔루션 아키텍처 세부 정보

이 섹션에서는 Confluent 검증에 사용된 하드웨어 및 소프트웨어에 대해 다룹니다. 이 정보는 NetApp 스토리지를 사용하는 Confluent Platform 구축에 적용할 수 있습니다. 다음 표에서는 테스트를 거친 솔루션 아키텍처 및 기본 구성 요소에 대해 설명합니다.

| 솔루션 구성 요소 | 세부 정보 |
|---|---|
| Confluent Kafka 버전 6.2 | • 세 명의 주키퍼입니다<br>• 브로커 서버 5대<br>• 5개의 도구 서버<br>• 하나의 Grafana<br>• 제어 센터 1개 |
| Linux(Ubuntu 18.04) | 모든 서버 |
| 계층형 스토리지를 위한 NetApp StorageGRID | • StorageGRID 소프트웨어<br>• SG1000(로드 밸런서) 1개<br>• SGF6024 4개<br>• 24 x 800 SSD 4개<br>• S3 프로토콜<br>• 4 x 100GbE(브로커와 StorageGRID 인스턴스 간 네트워크 연결) |
| 15 Fujitsu Primergy RX2540 서버 | 각 장착 사양: * CPU 2개, 물리적 코어 16개 * Intel Xeon * 256GB 물리적 메모리 * 100GbE 듀얼 포트 |

## 기술 개요

이 섹션에서는 이 솔루션에 사용된 기술에 대해 설명합니다.

### NetApp StorageGRID를 참조하십시오

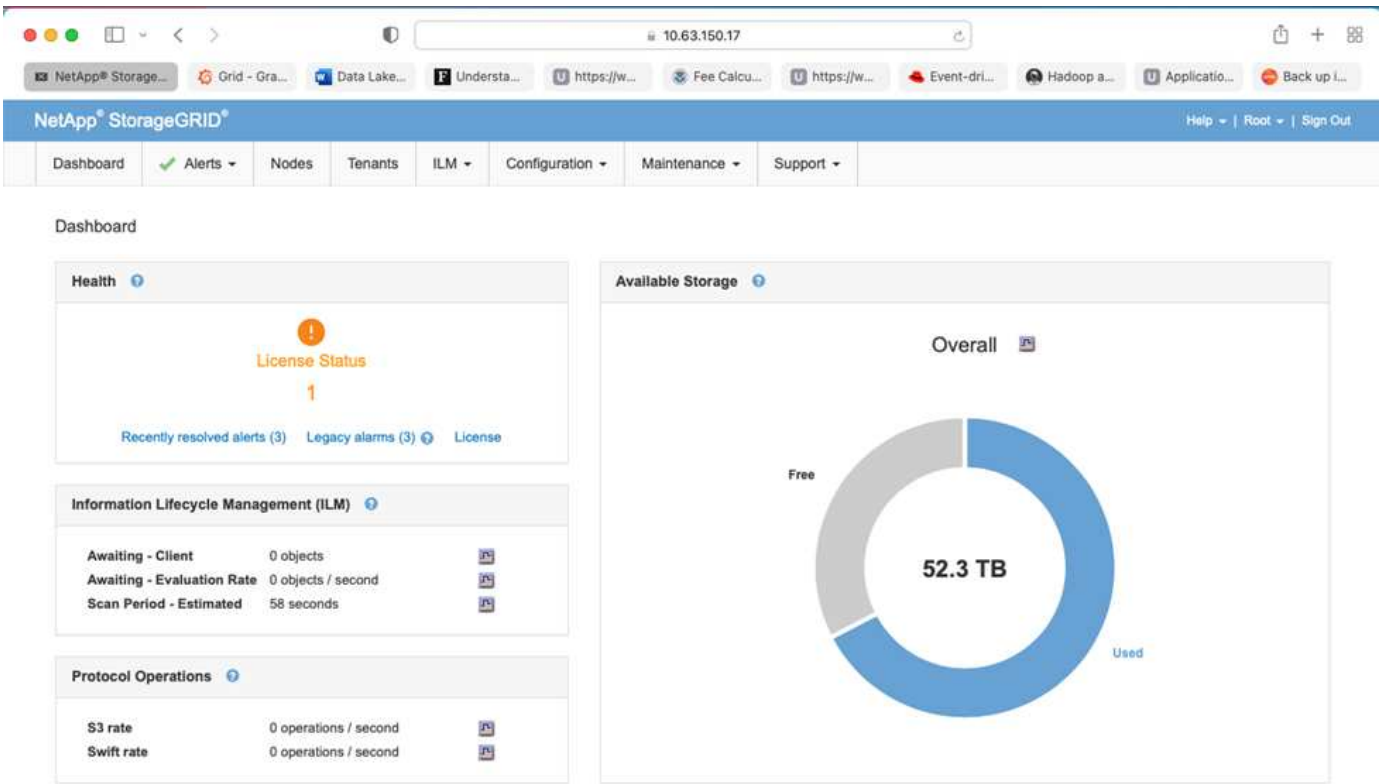
NetApp StorageGRID는 비용 효율적인 고성능 오브젝트 스토리지 플랫폼입니다. 계층형 스토리지를 사용하면 중개인의 로컬 스토리지 또는 SAN 스토리지에 저장된 Confluent Kafka의 데이터 대부분이 원격 오브젝트 저장소로 오프로드됩니다. 이 구성은 클러스터 재조정, 확장 또는 축소 또는 실패한 브로커 교체에 드는 시간과 비용을 줄여 운영 효율성을 크게 개선합니다. 오브젝트 스토리지는 오브젝트 저장소 계층에 있는 데이터를 관리하는 데 중요한 역할을 합니다. 따라서 적합한 오브젝트 스토리지를 선택하는 것이 중요합니다.

StorageGRID는 분산된 노드 기반 그리드 아키텍처를 사용하여 정책 중심의 지능형 글로벌 데이터 관리를 제공합니다. 정교한 데이터 관리 기능과 결합된 유비쿼터스 글로벌 오브젝트 네임스페이스를 통해 페타바이트 단위의 비정형 데이터와 수십억 개의 오브젝트 관리를 간소화합니다. 단일 호출 개체 액세스는 사이트 간에 확장되고 고가용성 아키텍처를 단순화하는 동시에 사이트 또는 인프라 중단과 관계없이 지속적인 개체 액세스를 보장합니다.

멀티 테넌시를 사용하면 동일한 그리드 내에서 여러 비정형 클라우드 및 엔터프라이즈 데이터 애플리케이션을 안전하게 서비스할 수 있으므로 NetApp StorageGRID의 ROI 및 사용 사례가 증가합니다. 여러 지역에서 내구성, 보호, 성능, 인접성을 최적화하여 메타데이터 기반 오브젝트 라이프사이클 정책을 통해 여러 서비스 레벨을 생성할 수 있습니다. 사용자는 데이터 관리 정책을 조정하고 트래픽 제한을 모니터링 및 적용하여 끊임없이 변화하는 IT 환경에서 요구 사항이 변경됨에 따라 데이터 환경을 중단 없이 다시 조정할 수 있습니다.

#### Grid Manager를 통한 간편한 관리

StorageGRID 그리드 관리자는 브라우저 기반의 그래픽 인터페이스로, 단일 창에서 전세계에 분산된 위치에 걸쳐 StorageGRID 시스템을 구성, 관리 및 모니터링할 수 있습니다.




StorageGRID 그리드 관리자 인터페이스를 사용하여 다음 작업을 수행할 수 있습니다.

- 이미지, 비디오, 레코드 등 전 세계에 분산된 페타바이트 규모의 오브젝트 저장소를 관리합니다.
- 그리드 노드 및 서비스를 모니터링하여 개체 가용성을 보장합니다.
- ILM(정보 수명 주기 관리) 규칙을 사용하여 시간이 지남에 따라 오브젝트 데이터의 배치를 관리합니다. 이러한 규칙은 수집된 개체의 데이터, 데이터가 손실되지 않도록 보호하는 방법, 오브젝트 데이터가 저장되는 위치 및 기간에 대해 적용됩니다.
- 시스템 내의 트랜잭션, 성능 및 운영을 모니터링합니다.

#### 정보 수명 주기 관리 정책

StorageGRID는 오브젝트의 복제본 보존 및 특정 성능 및 데이터 보호 요구사항에 따라 오브젝트를 저장할 수 있도록 2+1 및 4+2(특히 다른)와 같은 EC(삭제 코딩) 스키마를 사용하는 등의 유연한 데이터 관리 정책을 제공합니다. 시간에 따라 워크로드와 요구사항이 달라지날수록 ILM 정책도 시간에 따라 바뀌어야 합니다. ILM 정책을 수정하는 것은 핵심 기능이므로 StorageGRID 고객은 끊임없이 변화하는 환경에 빠르고 쉽게 적응할 수 있습니다. 를 확인하십시오 "ILM 정책" 및 "ILM 규칙" StorageGRID에서 설정.

#### 성능

StorageGRID는 VM, 베어 메탈 또는 특수 제작된 어플라이언스 등 스토리지 노드를 추가하여 성능을 확장합니다 "SG5712, SG5760, SG6060 또는 SGF6024". 이 테스트에서는 SGF6024 어플라이언스를 사용하는 최소 크기의 3노드 그리드로 Apache Kafka의 주요 성능 요구사항을 초과했습니다. 고객이 추가 브로커로 Kafka 클러스터를 확장함에 따라 스토리지 노드를 추가하여 성능과 용량을 확장할 수 있습니다.

#### 부하 분산 장치 및 엔드포인트 구성

StorageGRID의 관리 노드는 StorageGRID 시스템을 보고, 구성하고, 관리할 수 있는 그리드 관리자 UI(사용자 인터페이스) 및 REST API 엔드포인트와 시스템 작업을 추적할 수 있는 감사 로그를 제공합니다. Confluent Kafka 계층형 스토리지에 가용성이 높은 S3 엔드포인트를 제공하기 위해 관리 노드와 게이트웨이 노드에서 서비스로 실행되는 StorageGRID 로드 밸런서를 구현했습니다. 또한 로드 밸런서는 로컬 트래픽을 관리하고 GSLB(Global Server Load Balancing)에 연결하여 재해 복구를 지원합니다.

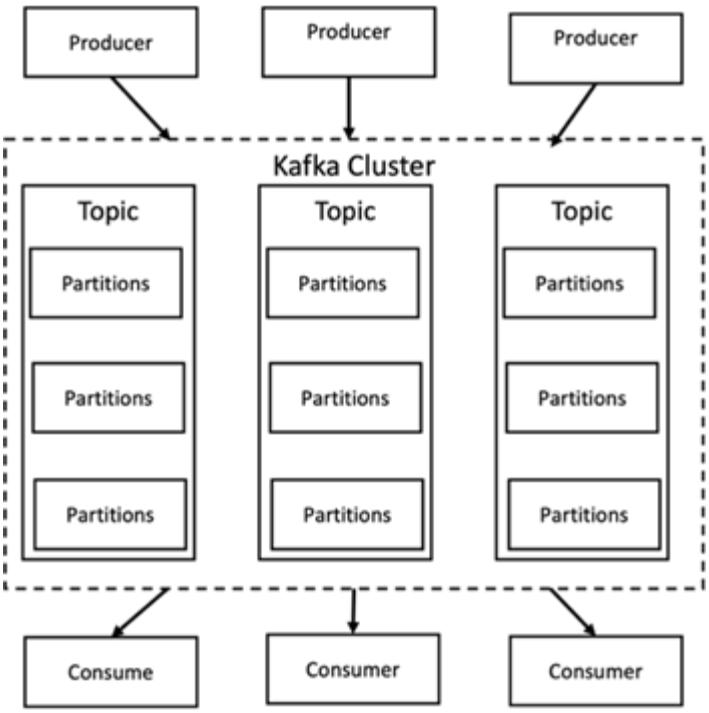
엔드포인트 구성을 더욱 개선하기 위해 StorageGRID는 관리 노드에 내장된 트래픽 분류 정책을 제공하고, 워크로드 트래픽을 모니터링하고, 워크로드에 다양한 QoS(서비스 품질) 제한을 적용할 수 있도록 지원합니다. 트래픽 분류 정책은 게이트웨이 노드 및 관리 노드에 대한 StorageGRID 부하 분산 서비스의 끝점에 적용됩니다. 이러한 정책은 트래픽 셰이핑 및 모니터링을 지원할 수 있습니다.

#### **StorageGRID**의 트래픽 분류

StorageGRID에는 QoS 기능이 내장되어 있습니다. 트래픽 분류 정책은 클라이언트 애플리케이션에서 들어오는 다양한 유형의 S3 트래픽을 모니터링하는 데 도움이 될 수 있습니다. 그런 다음 정책을 생성하여 적용하여 In/Out 대역폭, 읽기/쓰기 동시 요청 수 또는 읽기/쓰기 요청 속도에 따라 이 트래픽에 제한을 적용할 수 있습니다.

### 아파치 카프카

Apache Kafka는 Java 및 Scala로 작성된 스트림 처리를 사용하는 소프트웨어 버스의 프레임워크입니다. 이 제품은 실시간 데이터 피드 처리를 위한 높은 처리량의 짧은 대기 시간을 갖춘 통합 플랫폼을 제공하기 위해 마련되었습니다. Kafka는 데이터 내보내기 및 가져오기를 위해 Kafka Connect를 통해 외부 시스템에 연결할 수 있으며 Java 스트림 처리 라이브러리인 Kafka 스트림을 제공합니다. Kafka는 효율성을 위해 최적화된 바이너리 TCP 기반 프로토콜을 사용하며, 네트워크를 통한 왕복 작업의 오버헤드를 줄이기 위해 자연스럽게 메시지를 그룹화하는 "메시지 세트" 추상화에 의존합니다. 이렇게 하면 순차 디스크 작업, 더 큰 네트워크 패킷 및 연속 메모리 블록이 증가하므로 Kafka는 랜덤 메시지 쓰기의 폭주 스트림을 선형 쓰기로 전환할 수 있습니다. 다음 그림은 Apache Kafka의 기본 데이터 흐름을 보여 줍니다.




Kafka는 생산자라는 임의의 수의 프로세스에서 가져온 키 값 메시지를 저장합니다. 데이터는 여러 주제 내의 서로 다른 파티션으로 분할될 수 있습니다. 파티션 내에서 메시지는 해당 오프셋(파티션 내의 메시지 위치)에 의해 엄격하게 정렬되고 타임 스탬프와 함께 인덱싱되고 저장됩니다. 소비자라고 하는 다른 프로세스는 파티션에서 메시지를 읽을 수 있습니다. 스트림 처리를 위해 Kafka는 Kafka의 데이터를 사용하는 Java 애플리케이션을 작성하고 결과를 Kafka에 다시 쓸 수 있는 스트림 API를 제공합니다. Apache Kafka는 Apache Apex, Apache Flink, Apache Spark, Apache Storm, Apache nifi 등의 외부 스트림 처리 시스템과도 작동합니다.

Kafka는 하나 이상의 서버(브로커)로 구성된 클러스터에서 실행되며 모든 항목의 파티션이 클러스터 노드에 분산됩니다. 또한 파티션이 여러 브로커에 복제됩니다. 이 아키텍처를 통해 Kafka는 내결함성이 있는 방식으로 대규모 메시지 스트림을 전달할 수 있으며 JMS(Java Message Service), AMQP(Advanced Message Queuing Protocol) 등의 기존 메시징 시스템을 대체할 수 있습니다. Kafka는 0.11.0.0 릴리스 이후 트랜잭션 쓰기를 제공하여 스트림 API를 사용하여 정확히 한 번의 스트림 처리를 제공합니다.

Kafka는 일반과 컴팩션이라는 두 가지 유형의 주제를 지원합니다. 정규 주제는 보존 시간 또는 공간 바인딩으로 구성할 수 있습니다. 지정된 보존 시간보다 오래된 레코드가 있거나 파티션에 대해 바인딩된 공간이 초과된 경우 Kafka는 사용 가능한 저장소 공간을 확보하기 위해 이전 데이터를 삭제할 수 있습니다. 기본적으로 항목은 보존 기간이 7일로 구성되지만 데이터를 무기한 저장할 수도 있습니다. 압축된 항목의 경우 시간 또는 공간 범위에 따라 레코드가 만료되지 않습니다. 대신 Kafka는 나중에 받은 메시지를 동일한 키를 사용하는 이전 메시지의 업데이트로 취급하며 키당 최신 메시지를 삭제하지 않도록 보장합니다. 사용자는 특정 키에 대해 null 값을 갖는 소위 tombstone 메시지를 작성하여 메시지를 완전히 삭제할 수 있습니다.

Kafka에는 다음과 같은 5가지 주요 API가 있습니다.

- * Producer API. * 응용 프로그램에서 레코드 스트림을 게시할 수 있도록 허용합니다.
- * 소비자 API. * 응용 프로그램에서 항목 및 레코드 스트림 프로세스를 구독할 수 있도록 허용합니다.
- * Connector API. * 항목을 기존 응용 프로그램에 연결할 수 있는 재사용 가능한 프로듀서 및 소비자 API를 실행합니다.
- * Streams API. * 이 API는 입력 스트림을 출력으로 변환하고 결과를 생성합니다.
- * Admin API. * Kafka 주제, 브로커 및 기타 Kafka 객체를 관리하는 데 사용됩니다.

소비자 및 생산자 API는 Kafka 메시징 프로토콜을 기반으로 하며 Java의 Kafka 소비자 및 생산자 클라이언트에 대한 참조 구현을 제공합니다. 기본 메시징 프로토콜은 개발자가 프로그래밍 언어로 소비자 또는 생산자 클라이언트를 작성하는 데 사용할 수 있는 이진 프로토콜입니다. 이렇게 하면 JVM(Java Virtual Machine) 에코시스템에서 Kafka가 잠금 해제됩니다. 사용 가능한 비 Java 클라이언트 목록은 Apache Kafka wiki에서 유지 관리됩니다.

#### **Apache Kafka** 사용 사례

Apache Kafka는 메시징, 웹 사이트 활동 추적, 메트릭, 로그 집계, 스트림 처리, 이벤트 소싱 및 로깅 커밋

- Kafka는 향상된 처리량, 내장 파티셔닝, 복제 및 내결함성 기능을 제공하므로 대규모 메시지 처리 애플리케이션에 적합한 솔루션입니다.
- Kafka는 실시간 게시 구독 피드 집합으로 추적 파이프라인에서 사용자의 활동(페이지 보기, 검색)을 재구축할 수 있습니다.
- Kafka는 운영 모니터링 데이터에 자주 사용됩니다. 이를 위해서는 분산된 애플리케이션에서 통계를 집계하여 운영 데이터의 중앙 집중식 피드를 생성하는 작업이 필요합니다.
- 많은 사람들이 Kafka를 로그 집계 솔루션의 대안으로 사용합니다. 로그 집계는 일반적으로 서버에서 물리적 로그 파일을 수집하여 처리를 위해 중앙 위치(예: 파일 서버 또는 HDFS)에 배치합니다. Kafka는 파일 세부 정보를 추상화하고 로그 또는 이벤트 데이터를 메시지 스트림으로 추상화합니다. 따라서 대기 시간이 짧아지며 여러 데이터 소스 및 분산된 데이터 사용을 더욱 쉽게 지원할 수 있습니다.
- Kafka의 많은 사용자는 여러 스테이지로 구성된 처리 파이프라인에서 원시 입력 데이터가 Kafka 주제에서 소비된 후 추가 소비 또는 후속 처리를 위해 새로운 주제로 집계, 강화 또는 기타 방식으로 변환되는 데이터를 처리합니다. 예를 들어 뉴스 기사를 추천하기 위한 처리 파이프라인은 RSS 피드에서 기사 콘텐츠를 크롤링하여 "기사" 항목에 게시할 수 있습니다. 추가 처리에서는 이 콘텐츠를 정규화하거나 중복 제거하고 정리된 문서 콘텐츠를 새 주제에 게시하며 최종 처리 단계에서 사용자에게 이 콘텐츠를 추천하려고 할 수 있습니다. 이러한 처리 파이프라인은 개별 주제를 기반으로 실시간 데이터 플로우의 그래프를 작성합니다.
- 이벤트 수그리기는 상태 변경이 시간 순서 기록 시퀀스로 기록되는 응용 프로그램 디자인의 스타일입니다. Kafka는 매우 큰 저장 로그 데이터를 지원하므로 이 스타일로 구축된 애플리케이션에 대한 탁월한 백엔드로 활용할 수 있습니다.
- Kafka는 분산 시스템에 대한 일종의 외부 커밋 로그 역할을 할 수 있습니다. 이 로그는 노드 간 데이터를 복제하고 장애가 발생한 노드가 데이터를 복원할 수 있도록 재동기화 메커니즘 역할을 합니다. Kafka의 로그 컴팩션 기능은 이 활용 사례를 지원하는 데 도움이 됩니다.

유창하게

Confluent Platform은 Kafka를 완성하는 엔터프라이즈급 플랫폼으로, 애플리케이션 개발 및 연결 속도를 높이고, 스트림 처리를 통해 혁신을 지원하고, 규모에 따라 엔터프라이즈 운영을 간소화하고, 엄격한 아키텍처 요구 사항을 충족하도록 설계된 고급 기능을 제공합니다. Apache Kafka를 처음 개발한 Confluent는 Kafka 관리 또는 모니터링의 부담을 덜면서 엔터프라이즈급 기능을 통해 Kafka의 이점을 확장해 줍니다. 현재 Fortune 100대 기업 중 80% 이상이 데이터 스트리밍 기술을 사용하고 있으며 대부분 Confluent를 사용하고 있습니다.

#### 왜 **Confluent**인가**?**

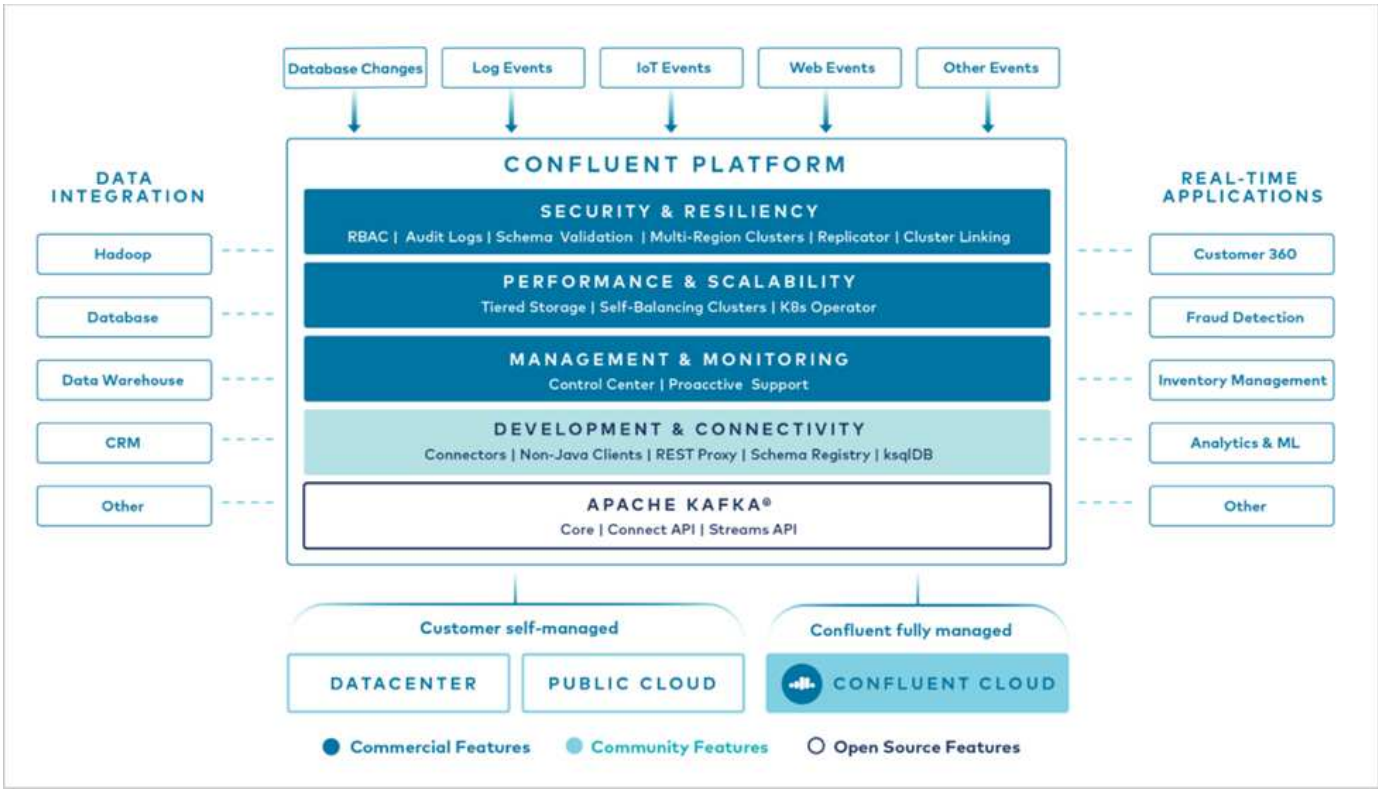
Confluorent는 기록 데이터와 실시간 데이터를 단일 중앙 데이터 소스에 통합하여 완전히 새로운 범주의 최신 이벤트 기반 애플리케이션을 쉽게 구축하고, 범용 데이터 파이프라인을 구축하며, 완전한 확장성, 성능, 안정성으로 강력한 새 사용 사례를 활용할 수 있도록 지원합니다.

#### **Contfluent**는 어떤 용도로 사용됩니까**?**

Confluent Platform을 사용하면 데이터가 다른 시스템 간에 어떻게 전송 또는 통합되는지 등의 기본 메커니즘을 걱정하지 않고 데이터에서 비즈니스 가치를 창출하는 방법에 집중할 수 있습니다. 특히 Confluent Platform은 데이터

소스를 Kafka에 연결하고 스트리밍 애플리케이션을 구축하며 Kafka 인프라의 보안, 모니터링 및 관리를 간소화합니다. 현재 Confluent Platform은 금융 서비스, 옴니채널 소매, 자율 자동차, 사기 탐지 등 다양한 산업 전반의 다양한 사용 사례에 사용됩니다. 마이크로서비스, IoT

다음 그림에서는 Confluent Kafka 플랫폼 구성 요소를 보여 줍니다.




**Confluent**의 이벤트 스트리밍 기술 개요

Confluent Platform의 핵심은 입니다 "아파치 카프카"가장 널리 사용되는 오픈 소스 분산 스트리밍 플랫폼입니다. Kafka의 주요 기능은 다음과 같습니다.

- 레코드 스트림을 게시하고 구독합니다.
- 내결함성이 있는 방식으로 레코드 스트림을 저장합니다.
- 레코드 스트림을 처리합니다.

즉시 사용할 수 있는 Confluorent Platform에는 스키마 레지스트리, REST 프록시, 총 100개 이상의 사전 구축된 Kafka 커넥터 및 ksqlDB도 포함되어 있습니다.

**Confluent** 플랫폼의 엔터프라이즈 기능 개요

- * Confluent Control Center. * Kafka 관리 및 모니터링을 위한 GUI 기반 시스템. Kafka Connect를 쉽게 관리하고 다른 시스템에 대한 연결을 생성, 편집 및 관리할 수 있습니다.
- Kubernetes를 위한 * Contfluent. * Kubernetes를 위한 Confluent는 Kubernetes 운영자입니다. Kubernetes 운영자는 특정 플랫폼 애플리케이션에 대한 고유한 기능과 요구 사항을 제공하여 Kubernetes의 오케스트레이션 기능을 확장합니다. Confluent Platform의 경우, Kubernetes에서 Kafka의 구축 프로세스를 크게 간소화하고 일반적인 인프라 라이프사이클 작업을 자동화할 수 있습니다.
- * Kafka * 커넥터에 대한 Confluent 커넥터 Kafka Connect API를 사용하여 Kafka를 데이터베이스, 키 값 저장소, 검색 인덱스 및 파일 시스템과 같은 다른 시스템에 연결합니다. Confluorent Hub에는 가장 널리 사용되는 데이터

소스 및 싱크에 대한 다운로드 가능한 커넥터가 있습니다. 여기에는 Confluorent Platform이 포함된 이러한 커넥터의 전체 테스트 및 지원 버전이 포함됩니다. 자세한 내용은 을 참조하십시오 "여기".

- * 자체 밸런싱 클러스터 * 는 자동화된 로드 밸런싱, 장애 감지 및 자동 복구를 제공합니다. 필요에 따라 브로커를 추가하거나 해체할 수 있도록 지원하며 수동 튜닝이 필요하지 않습니다.
- * 연결 클러스터. * 직접 클러스터를 연결하고 링크 브리지를 통해 클러스터 간에 주제를 미러링합니다. 클러스터 링크를 사용하면 멀티 데이터 센터, 멀티 클러스터, 하이브리드 클라우드 구축을 간편하게 설정할 수 있습니다.
- * Confluorent auto data balancer. * 브로커 수, 파티션 크기, 파티션 수 및 클러스터 내의 리더 수에 대한 클러스터를 모니터링합니다. 균형 조정을 통해 트래픽을 재조정함으로써 운영 워크로드에 미치는 영향을 최소화하면서 클러스터 전체에서 짝수 워크로드를 생성할 수 있습니다.
- * Confluent Replicator. * 여러 데이터 센터에서 여러 Kafka 클러스터를 훨씬 쉽게 유지 관리할 수 있습니다.
- * 계층형 스토리지. * 즐겨 사용하는 클라우드 공급자를 사용하여 대량의 Kafka 데이터를 저장할 수 있는 옵션을 제공하므로 운영 부담과 비용이 줄어듭니다. 계층형 스토리지를 사용하면 비용 효율적인 오브젝트 스토리지에 데이터를 보관하고 더 많은 컴퓨팅 리소스가 필요할 때만 브로커를 확장할 수 있습니다.
- * Confluent JMS 클라이언트. * Confluent Platform에는 Kafka용 JMS 호환 클라이언트가 포함되어 있습니다. 이 Kafka 클라이언트는 Kafka 브로커를 백엔드로 사용하여 JMS 1.1 표준 API를 구현합니다. JMS를 사용하는 레거시 애플리케이션이 있고 기존 JMS 메시지 브로커를 Kafka로 교체하려는 경우 유용합니다.
- * Confluent MQTT proxy. * 중간에 MQTT 브로커가 없어도 MQTT 장치 및 게이트웨이에서 Kafka에 직접 데이터를 게시할 수 있는 방법을 제공합니다.
- * Confluent 보안 플러그인 * Confluent 보안 플러그인은 다양한 Confluent 플랫폼 도구 및 제품에 보안 기능을 추가하는 데 사용됩니다. 현재 Confluent REST 프록시에 사용할 수 있는 플러그인이 있어 수신 요청을 인증하고 인증된 보안 주체를 Kafka에 요청에 전파할 수 있습니다. 이렇게 하면 Confluent REST 프록시 클라이언트가 Kafka 브로커의 멀티테넌트 보안 기능을 활용할 수 있습니다.

## Confluent 검증

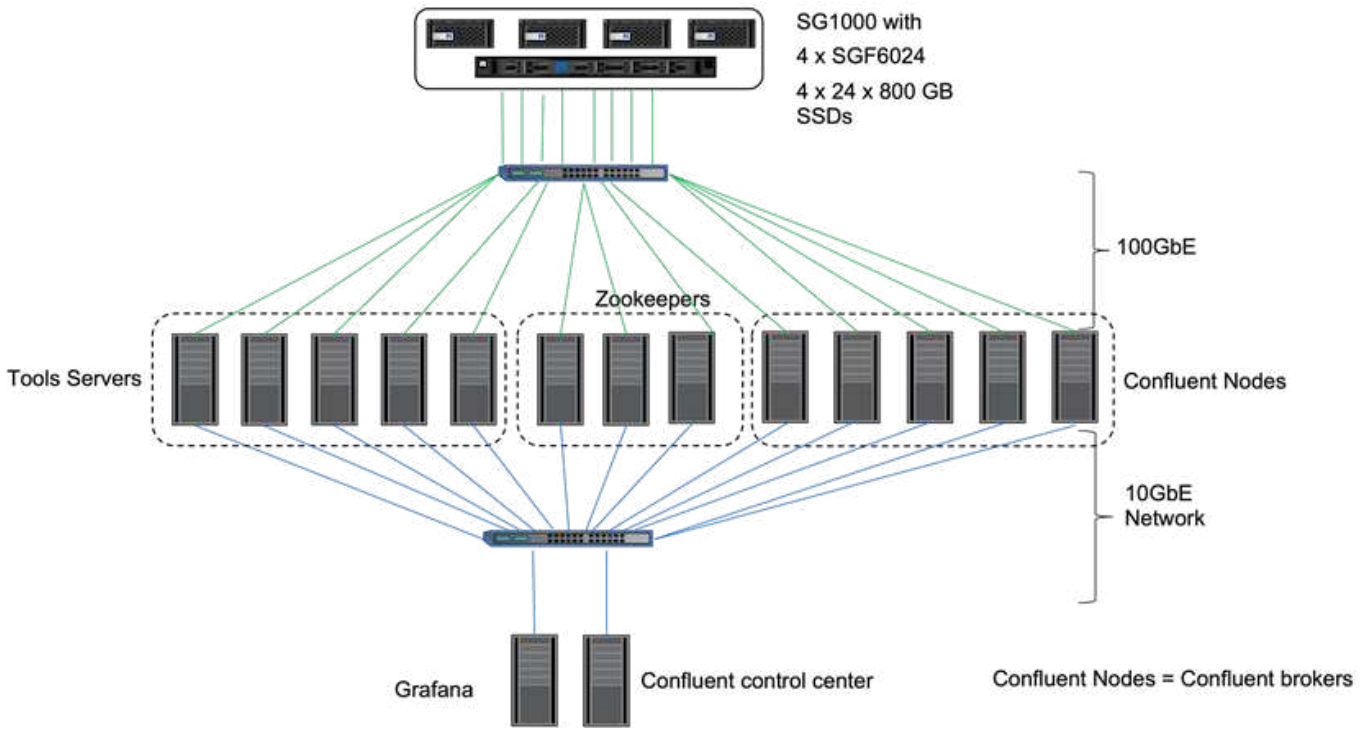
NetApp StorageGRID에서 Confluent Platform 6.2 계층형 스토리지를 사용하여 검증을 수행했습니다. NetApp과 Confluent 팀은 이 검증을 함께 수행하여 검증에 필요한 테스트 사례를 실행했습니다.

### Confluent 플랫폼 설정

검증을 위해 다음 설정을 사용했습니다.

검증을 위해 3대의 zookeers, 5개의 브로커, 5개의 테스트 스크립트 실행 서버, 256GB RAM이 장착된 명명된 도구 서버, 16개의 CPU를 사용했습니다. NetApp 스토리지의 경우 SGF6024s 4개로 SG1000 로드 밸런서와 함께 StorageGRID를 사용했습니다. 스토리지와 브로커는 100GbE 연결을 통해 연결되었습니다.

다음 그림은 Confluent 확인에 사용되는 구성의 네트워크 토폴로지를 보여줍니다.




도구 서버는 Confluent 노드에 요청을 보내는 애플리케이션 클라이언트의 역할을 합니다.

**Confluent** 계층형 스토리지 구성

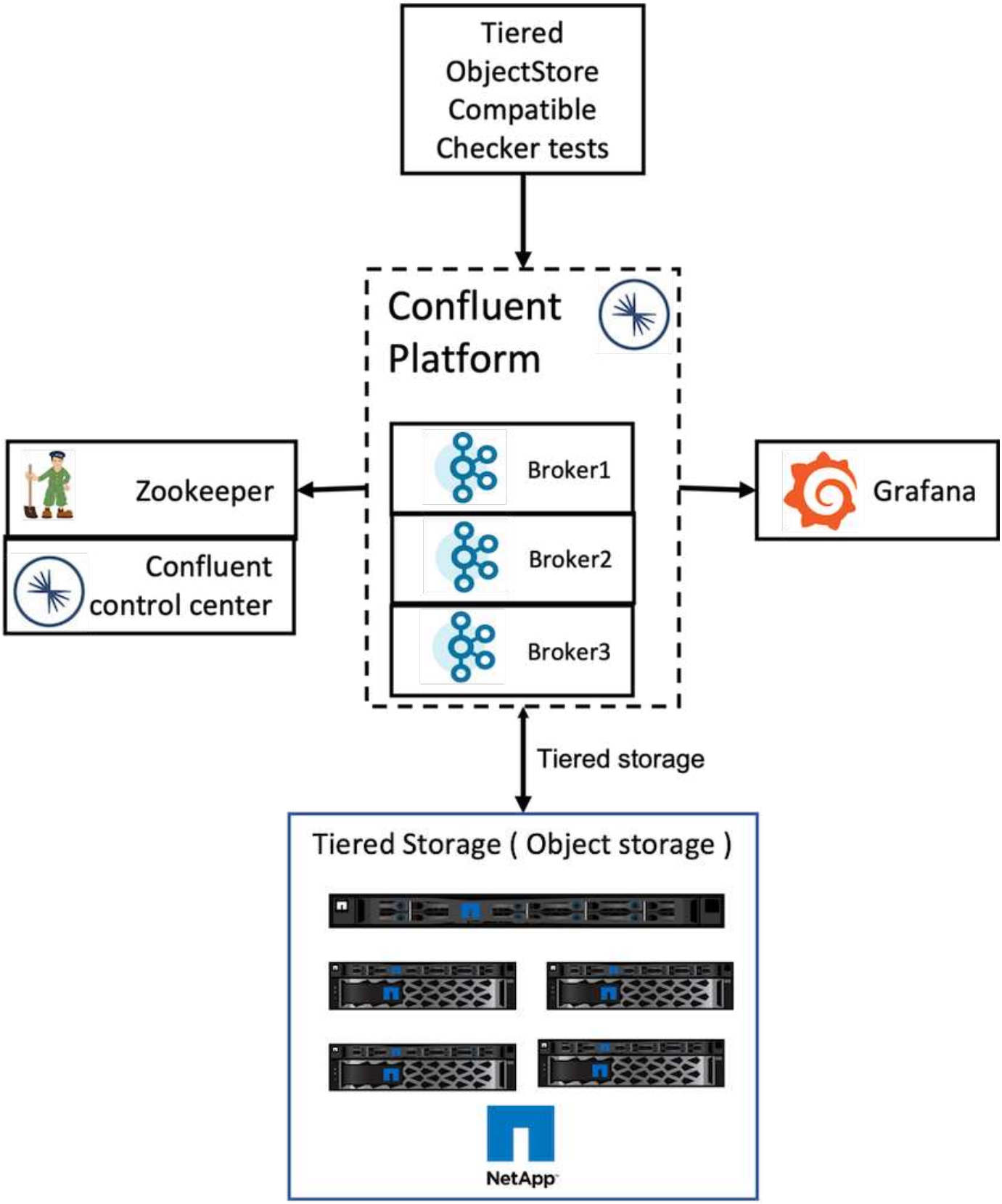
계층화된 스토리지 구성에는 Kafka에서 다음 매개 변수가 필요합니다.

```
Confluent.tier.archiver.num.threads=16
confluent.tier.fetcher.num.threads=32
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkasgdbucket1-2
confluent.tier.s3.region=us-west-2
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://kafkasgd.rtpppe.netapp.com:
10444/
confluent.tier.s3.force.path.style.access=true
```

검증을 위해 StorageGRID를 HTTP 프로토콜과 함께 사용했지만 HTTPS도 작동합니다. 액세스 키와 비밀 키는 confluent.tier.s3.cred.file.path 매개 변수에 제공된 파일 이름에 저장됩니다.

**NetApp** 오브젝트 스토리지 **- StorageGRID**

StorageGRID에서 정화를 위해 단일 사이트 구성을 구성했습니다.




검증 테스트

검증을 위해 다음 5가지 테스트 사례를 완료했습니다. 이러한 검사는 Trogor 프레임워크에서 실행됩니다. 첫 번째 두 테스트는 기능 테스트이고 나머지 세 가지는 성능 테스트입니다.

#### 객체 저장소 정확도 테스트

이 테스트에서는 오브젝트 저장소 API의 모든 기본 작업(예: get/put/delete)이 계층적 스토리지의 요구에 따라 잘 작동하는지 여부를 확인합니다. 모든 오브젝트 저장소 서비스가 다음 테스트보다 먼저 통과해야 하는 기본 테스트입니다. 통과 또는 실패한 단정적인 테스트입니다.

#### 계층화 기능 정확도 테스트

이 테스트에서는 종단 간 계층형 스토리지 기능이 통과 또는 실패한 적극적 테스트와 잘 작동하는지 여부를 확인합니다. 이 테스트에서는 기본적으로 계층화를 사용하도록 구성되고 핫 세트 크기가 크게 줄어든 테스트 항목을 생성합니다. 새로 만든 테스트 항목으로 이벤트 스트림을 생성하고 브로커가 세그먼트를 개체 저장소에 아카이빙할 때까지 대기한 다음 이벤트 스트림을 사용하여 소비된 스트림이 생성된 스트림과 일치하는지 확인합니다. 이벤트 스트림에 생성되는 메시지 수를 구성할 수 있으므로 테스트 요구에 따라 충분한 크기의 워크로드를 생성할 수 있습니다. 줄어든 핫세트 크기는 소비자가 활성 세그먼트 밖에 있는 페치를 개체 저장소에서만 제공되도록 합니다. 이렇게 하면 읽기에 대한 개체 저장소의 정확성을 테스트하는 데 도움이 됩니다. 객체 저장소 결함 주입 여부와 관계없이 이 테스트를 수행했습니다. StorageGRID의 노드 중 하나에서 서비스 관리자 서비스를 중지하고 엔드 투 엔드 기능이 오브젝트 스토리지에서 작동하는지 확인하여 노드 장애를 시뮬레이션했습니다.

#### 계층 가져오기 벤치마크

이 테스트에서는 계층형 오브젝트 스토리지의 읽기 성능을 검증하고 벤치마크에 의해 생성된 세그먼트에서 과부하된 읽기 요청 범위를 검사했습니다. 이 벤치마크에서 Confluent는 계층 가져오기 요청을 처리하기 위해 사용자 지정 클라이언트를 개발했습니다.

#### 생산 - 워크로드 벤치마크 소비

이 테스트에서는 세그먼트 아카이브를 통해 오브젝트 저장소에서 쓰기 워크로드를 간접적으로 생성했습니다. 소비자 그룹이 세그먼트를 가져올 때 객체 스토리지에서 읽기 워크로드(세그먼트 읽기)가 생성되었습니다. 이 워크로드는 테스트 스크립트에 의해 생성되었습니다. 이 테스트에서는 오브젝트 저장소에서 병렬 스레드의 읽기 및 쓰기 성능을 확인했습니다. 계층화 기능 정확도 테스트에서 보았듯이, 객체 저장소 결함 주입을 사용하여 테스트했으며 포함하지 않았습니다.

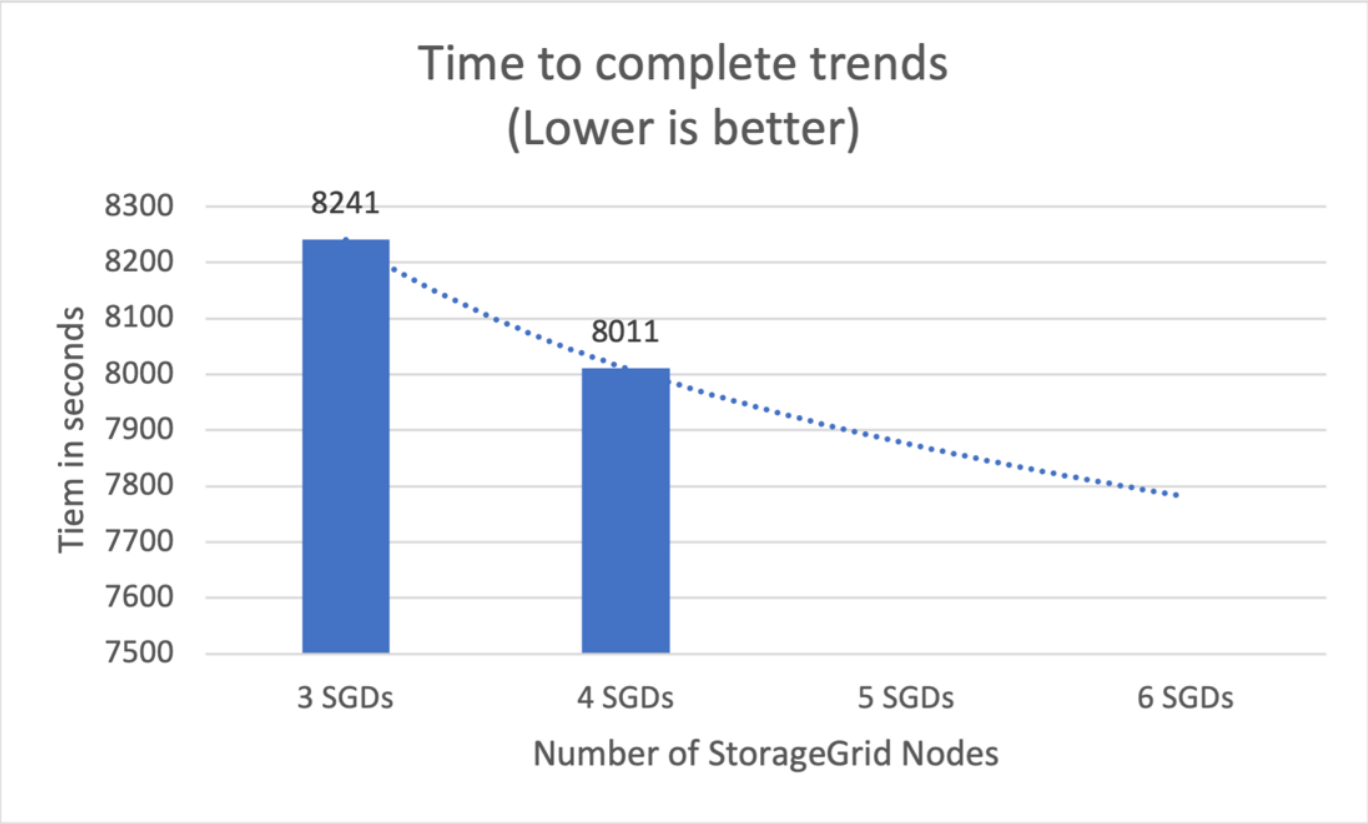
#### 보존 워크로드 벤치마크

이 테스트에서는 무거운 주제 보존 워크로드 하에서 오브젝트 저장소의 삭제 성능을 검사했습니다. 보존 워크로드는 테스트 주제와 동시에 많은 메시지를 생성하는 테스트 스크립트를 사용하여 생성되었습니다. 테스트 주제는 공격적인 크기 기반 및 시간 기반 보존 설정으로 구성되었으며, 이로 인해 이벤트 스트림이 개체 저장소에서 지속적으로 제거됩니다. 그런 다음 세그먼트가 아카이브되었습니다. 이로 인해 오브젝트 저장소 삭제 작업의 수행 중개 및 컬렉션에 의해 오브젝트 저장소에서 많은 삭제가 수행되었습니다.

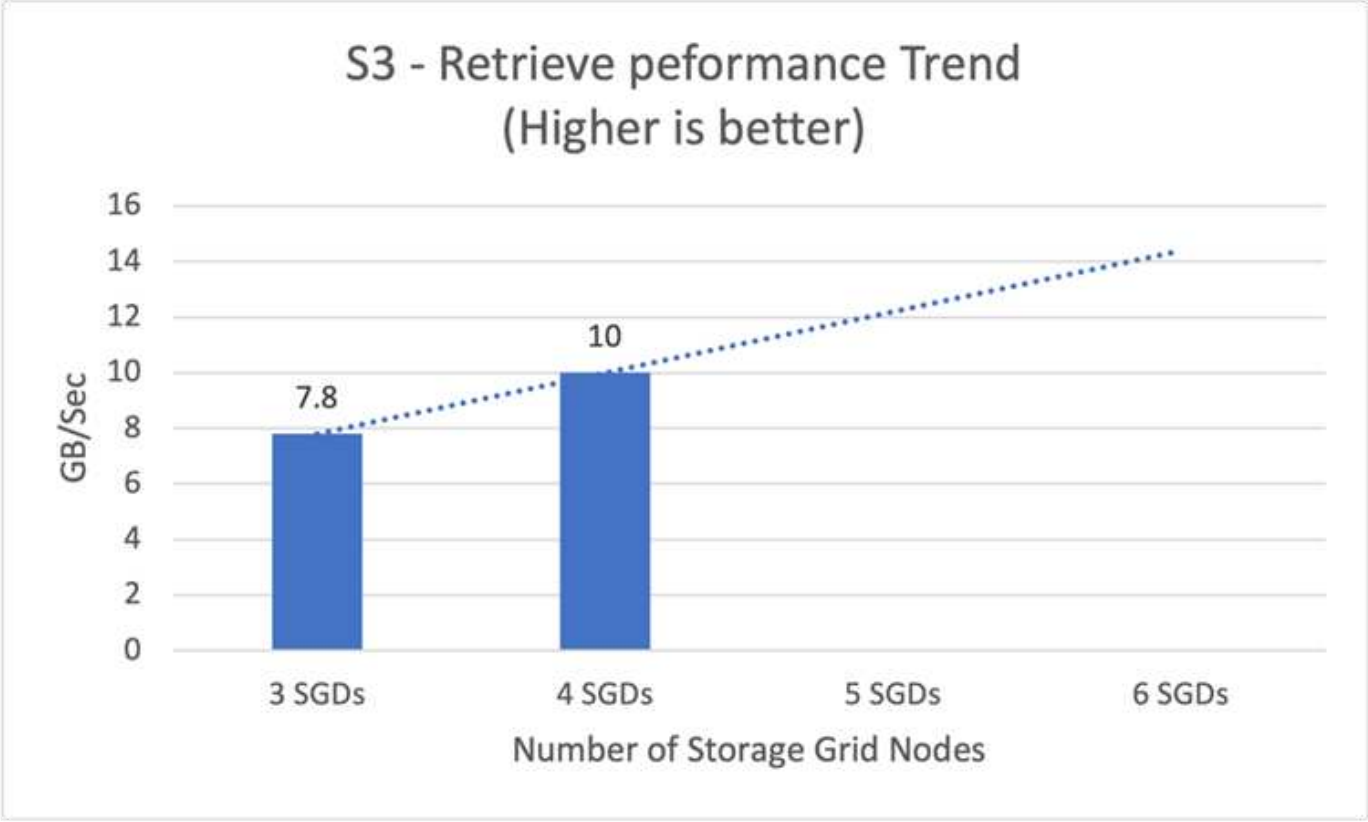
## 확장성을 갖춘 성능 테스트

NetApp StorageGRID 설정을 통해 생산자 및 소비자 워크로드를 위한 3~4개의 노드로 계층형 스토리지 테스트를 수행했습니다. 이 테스트에 따르면 완료 시간과 성능 결과는 StorageGRID 노드 수에 정비례합니다. StorageGRID를 설치하려면 최소 3개의 노드가 필요합니다.

- 스토리지 노드의 수가 증가했을 때 생산 및 소비자 작업을 완료하는 데 걸리는 시간이 선형적으로 감소했습니다.




- S3 검색 작업의 성능은 StorageGRID 노드 수에 따라 선형으로 증가합니다. StorageGRID는 최대 200개의 StorgeGRID 노드를 지원합니다.



## Confluent S3 커넥터

Amazon S3 싱크 커넥터는 Apache Kafka 항목의 데이터를 Avro, JSON 또는 바이트 형식의 S3 오브젝트로 내보냅니다. Amazon S3 싱크 커넥터는 주기적으로 Kafka의 데이터를 폴링하여 S3로 업로드합니다. 분할자는 모든 Kafka 파티션의 데이터를 청크로 분할하는 데 사용됩니다. 각 데이터 청크는 S3 오브젝트로 표시됩니다. 키 이름은 주제, Kafka 파티션 및 이 데이터 청크의 시작 오프셋을 인코딩합니다.

이 설정에서는 Kafka S3 싱크 커넥터를 사용하여 바로 Kafka에서 오브젝트 스토리지의 항목을 읽고 쓰는 방법을 보여줍니다. 이 테스트에서는 독립 실행형 Confluent 클러스터를 사용했지만 이 설정은 분산 클러스터에 적용됩니다.

1. Confluent 웹 사이트에서 Confluent Kafka를 다운로드하십시오.
2. 패키지를 서버의 폴더에 압축을 풉니다.
3. 두 변수를 내보냅니다.

```
Export CONFLUENT_HOME=/data/confluent/confluent-6.2.0
export PATH=$PATH:/data/confluent/confluent-6.2.0/bin
```

4. 독립 실행형 Confluorent Kafka 설정의 경우 클러스터는 "/tmp"에 임시 루트 폴더를 생성합니다. 또한 ZooKeeper, Kafka, 스키마 레지스트리, 연결, ksql-server를 생성합니다. 제어 센터 폴더를 만들고 해당 구성 파일을 '$CONFLUENT_HOME'에서 복사합니다. 다음 예를 참조하십시오.

```
root@stlrx2540m1-108:~# ls -ltr /tmp/confluent.406980/
total 28
drwxr-xr-x 4 root root 4096 Oct 29 19:01 zookeeper
drwxr-xr-x 4 root root 4096 Oct 29 19:37 kafka
drwxr-xr-x 4 root root 4096 Oct 29 19:40 schema-registry
drwxr-xr-x 4 root root 4096 Oct 29 19:45 kafka-rest
drwxr-xr-x 4 root root 4096 Oct 29 19:47 connect
drwxr-xr-x 4 root root 4096 Oct 29 19:48 ksql-server
drwxr-xr-x 4 root root 4096 Oct 29 19:53 control-center
root@stlrx2540m1-108:~#
```

5. ZooKeeper를 구성합니다. 기본 매개 변수를 사용하는 경우 아무것도 변경할 필요가 없습니다.

```
root@stlrx2540m1-108:~# cat
/tmp/confluent.406980/zookeeper/zookeeper.properties  | grep -iv ^#
dataDir=/tmp/confluent.406980/zookeeper/data
clientPort=2181
maxClientCnxns=0
admin.enableServer=false
tickTime=2000
initLimit=5
syncLimit=2
server.179=controlcenter:2888:3888
root@stlrx2540m1-108:~#
```

위 구성에서 서버를 업데이트했습니다. XXX'재산. 기본적으로 Kafka 리더 선택을 위해서는 Zookeepers가 세 개 필요합니다.

6. 고유한 ID로 '/tmp/confluent.406980/zookeeper/data'에 myid 파일을 만들었습니다.

```
root@stlrx2540m1-108:~# cat /tmp/confluent.406980/zookeeper/data/myid
179
root@stlrx2540m1-108:~#
```

myid 파일에 대한 마지막 IP 주소 수를 사용했습니다. Kafka, CONNECT, CONTROL-CENTER, Kafka, Kafka-Rest, ksql-server 및 schema-registry 구성.

7. Kafka 서비스를 시작합니다.

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  start
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
ZooKeeper is [UP]
Kafka is [UP]
Schema Registry is [UP]
Kafka REST is [UP]
Connect is [UP]
ksqlDB Server is [UP]
Control Center is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

각 구성에 대한 로그 폴더가 있어 문제를 해결하는 데 도움이 됩니다. 경우에 따라 서비스를 시작하는 데 더 많은 시간이 걸릴 수 있습니다. 모든 서비스가 실행 중인지 확인합니다.

8. 'confluent-hub'을 이용하여 Kafka CONNECT를 설치한다.

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# ./confluent-
hub install confluentinc/kafka-connect-s3:latest
The component can be installed in any of the following Confluent
Platform installations:
  1. /data/confluent/confluent-6.2.0 (based on $CONFLUENT_HOME)
  2. /data/confluent/confluent-6.2.0 (where this tool is installed)
Choose one of these to continue the installation (1-2): 1
Do you want to install this into /data/confluent/confluent-
6.2.0/share/confluent-hub-components? (yN) y

Component's license:
Confluent Community License
http://www.confluent.io/confluent-community-license
I agree to the software license agreement (yN) y
Downloading component Kafka Connect S3 10.0.3, provided by Confluent,
Inc. from Confluent Hub and installing into /data/confluent/confluent-
6.2.0/share/confluent-hub-components
Do you want to uninstall existing version 10.0.3? (yN) y
Detected Worker's configs:
  1. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  2. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  3. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-distributed.properties
  4. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-standalone.properties
  5. Based on CONFLUENT_CURRENT:
/tmp/confluent.406980/connect/connect.properties
  6. Used by Connect process with PID 15904:
/tmp/confluent.406980/connect/connect.properties
Do you want to update all detected configs? (yN) y
Adding installation directory to plugin path in the following files:
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
standalone.properties
  /tmp/confluent.406980/connect/connect.properties
  /tmp/confluent.406980/connect/connect.properties
```

```
Completed
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

confluent-hub install confluentinc/Kafka-connect-S3:10.0.3'을 사용하여 특정 버전을 설치할 수도 있습니다.

9. 기본적으로 '/data/confluent/confluent-6.2.0/share/confluent-hub-components/confluententinc-kafka-connect-s3'에 confluentinc-kafka-connect-s3이 설치됩니다.
10. 새로운 'confluentinc-kafka-connect-s3'으로 플러그인 경로를 업데이트합니다.

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/etc/kafka/connect-distributed.properties | grep plugin.path
#
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/co
nnectors,
plugin.path=/usr/share/java,/data/zookeeper/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3
root@stlrx2540m1-108:~#
```

11. Confluent 서비스를 중지하고 다시 시작합니다.

```
confluent local services  stop
confluent local services  start
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  status
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
Connect is [UP]
Control Center is [UP]
Kafka is [UP]
Kafka REST is [UP]
ksqlDB Server is [UP]
Schema Registry is [UP]
ZooKeeper is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

12. '/root/.aws/credentials' 파일에서 액세스 ID와 비밀 키를 설정한다.

```
root@stlrx2540m1-108:~# cat /root/.aws/credentials
[default]
aws_access_key_id = xxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxx
root@stlrx2540m1-108:~#
```

13. 버킷에 도달할 수 있는지 확인합니다.

```
root@stlrx2540m4-01:~# aws s3 –endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls kafkasgdbucket1-2
2021-10-29 21:04:18       1388 1
2021-10-29 21:04:20       1388 2
2021-10-29 21:04:22       1388 3
root@stlrx2540m4-01:~#
```

14. S3 및 버킷 구성에 대해 S3-싱크 속성 파일을 구성합니다.

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-
s3/etc/quickstart-s3.properties | grep -v ^#
name=s3-sink
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=1
topics=s3_testtopic
s3.region=us-west-2
s3.bucket.name=kafkasgdbucket1-2
store.url=http://kafkasgd.rtpppe.netapp.com:10444/
s3.part.size=5242880
flush.size=3
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.avro.AvroFormat
partitioner.class=io.confluent.connect.storage.partitioner.DefaultPartit
ioner
schema.compatibility=NONE
root@stlrx2540m1-108:~#
```

15. S3 버킷으로 몇 개의 레코드를 가져옵니다.

```
kafka-avro-console-producer --broker-list localhost:9092 --topic
s3_topic \
--property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1",
"type":"string"}]}'
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
{"f1": "value4"}
{"f1": "value5"}
{"f1": "value6"}
{"f1": "value7"}
{"f1": "value8"}
{"f1": "value9"}
```

16. S3 싱크 커넥터를 로드합니다.

```
root@stlrx2540m1-108:~# confluent local services connect connector load
s3-sink  --config /data/confluent/confluent-6.2.0/share/confluent-hub-
components/confluentinc-kafka-connect-s3/etc/quickstart-s3.properties
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "flush.size": "3",
    "format.class": "io.confluent.connect.s3.format.avro.AvroFormat",
    "partitioner.class":
"io.confluent.connect.storage.partitioner.DefaultPartitioner",
    "s3.bucket.name": "kafkasgdbucket1-2",
    "s3.part.size": "5242880",
    "s3.region": "us-west-2",
    "schema.compatibility": "NONE",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "store.url": "http://kafkasgd.rtpppe.netapp.com:10444/",
    "tasks.max": "1",
    "topics": "s3_testtopic",
    "name": "s3-sink"
  },
  "tasks": [],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

17. S3 싱크 상태를 확인합니다.

```
root@stlrx2540m1-108:~# confluent local services connect connector
status s3-sink
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "10.63.150.185:8083"
  },
  "tasks": [
    {
      "id": 0,
      "state": "RUNNING",
      "worker_id": "10.63.150.185:8083"
    }
  ],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

18. 로그를 확인하여 S3 싱크가 항목을 수락할 준비가 되었는지 확인합니다.

```
root@stlrx2540m1-108:~# confluent local services connect log
```

19. Kafka의 주제를 확인하십시오.

```
kafka-topics --list --bootstrap-server localhost:9092
…
connect-configs
connect-offsets
connect-statuses
default_ksql_processing_log
s3_testtopic
s3_topic
s3_topic_new
root@stlrx2540m1-108:~#
```

20. S3 버킷의 오브젝트를 확인합니다.

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls --recursive kafkasgdbucket1-
2/topics/
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000003.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000006.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000009.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000012.avro
2021-10-29 21:24:09        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000015.avro
root@stlrx2540m1-108:~#
```

21. 내용을 확인하려면 다음 명령을 실행하여 각 파일을 S3에서 로컬 파일 시스템으로 복사합니다.

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 cp s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
tes.avro
download: s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro to
./tes.avro
root@stlrx2540m1-108:~#
```

22. 레코드를 인쇄하려면 avro-tools-1.11.0.1.jar (에서 사용 가능)를 사용합니다 "아파치 아카이브")를 클릭합니다.

```
root@stlrx2540m1-108:~# java -jar /usr/src/avro-tools-1.11.0.1.jar
tojson tes.avro
21/10/30 00:20:24 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
{"f1":"value1"}
{"f1":"value2"}
{"f1":"value3"}
root@stlrx2540m1-108:~#
```

## Confluent Self-Balancing 클러스터

이전에 Kafka 클러스터를 관리했다면 수동으로 파티션을 다른 브로커에 재할당하여 클러스터 전체에서 워크로드가 균형 있게 조정되도록 하는 데 따르는 문제에 익숙할 것입니다. Kafka가 대규모로 구축된 조직의 경우 대량의 데이터를 개각하는 일은 까다롭고 번거로우며 위험할 수 있습니다. 특히 클러스터 위에 미션 크리티컬 애플리케이션을 구축하는 경우에는 더욱 그렇습니다. 그러나 가장 작은 Kafka 사용 사례에서도 이 프로세스는 시간이 많이 걸리며 오류가 발생하기 쉽습니다.

이 실습에서는 Confluent Self-Balancing 클러스터 기능을 테스트하여 클러스터 토폴로지 변경 또는 불균등한 로드를 기반으로 재조정을 자동화했습니다. Confluent rebalance 테스트는 노드 장애 또는 확장 노드에서 브로커 간의 데이터 재조정이 필요한 경우 새 브로커를 추가하는 시간을 측정하는 데 도움이 됩니다. 기존 Kafka 구성에서는 클러스터의 증가에 따라 재조정할 데이터의 양이 증가하지만 계층형 스토리지에서는 소량의 데이터로 제한됩니다. NetApp의 검증을 바탕으로, 계층화된 스토리지에서 재조정은 기존 Kafka 아키텍처에서 몇 초 또는 몇 분 만에 이루어지게 되며, 클러스터의 확장에 따라 선형으로 증가합니다.

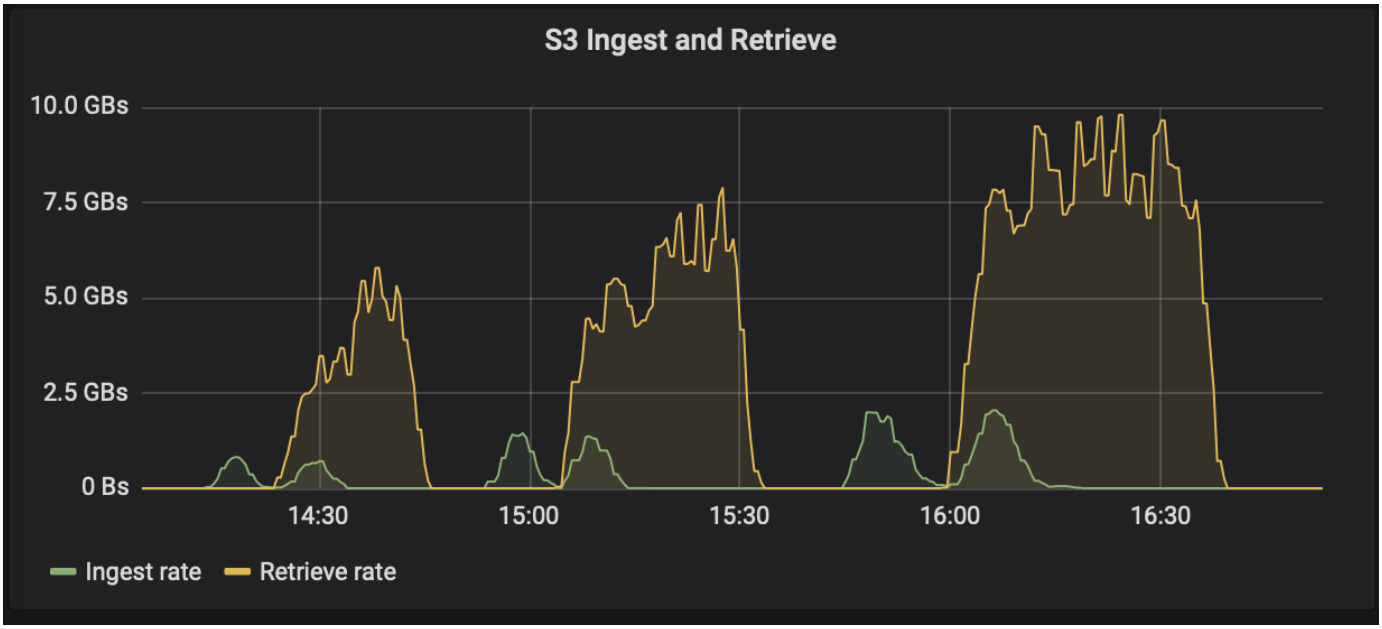
자체 밸런싱 클러스터에서는 파티션 재조정이 완전히 자동화되어 Kafka의 처리량을 최적화하고 브로커 확장을 가속화하며 대규모 클러스터를 실행하는 데 따른 운영 부담을 줄일 수 있습니다. 안정적인 상태에서 자체 균형 조정 클러스터는 브로커 전체의 데이터 불균형을 모니터링하고 지속적으로 파티션을 재할당하여 클러스터 성능을 최적화합니다. 플랫폼을 확장하거나 축소할 때 자체 균형 조정 클러스터는 새로운 브로커의 존재 또는 이전 브로커의 제거를 자동으로 인식하고 후속 파티션 재할당을 트리거합니다. 이를 통해 브로커를 쉽게 추가하고 해체할 수 있으므로 Kafka 클러스터의 유연성이 근본적으로 높아집니다. 이러한 이점은 수동 개입, 복잡한 수학 또는 재할당을 분할하는 데 수반되는 인적 오류의 위험 없이 제공됩니다. 따라서 데이터 재조정이 훨씬 더 빠르게 완료되고, 클러스터를 지속적으로 감독해야 하는 대신 더 가치 있는 이벤트 스트리밍 프로젝트에 집중할 수 있습니다.

## 모범 사례 지침

이 섹션에서는 이 인증에서 얻은 교훈을 설명합니다.

- NetApp의 검증을 기반으로 S3 오브젝트 스토리지는 데이터를 유창하게 유지하는 데 가장 적합합니다.
- 처리량이 높은 SAN(특히 FC)을 사용하여 브로커의 핫 데이터 또는 로컬 디스크를 유지할 수 있습니다. 왜냐하면, ContFluent 계층형 스토리지 구성에서 브로커 데이터 디렉토리에 있는 데이터의 크기는 데이터가 오브젝트 스토리지로 이동되는 세그먼트 크기 및 보존 시간을 기준으로 합니다.
- 오브젝트 저장소는 세그먼트일 때 더 나은 성능을 제공합니다. 바이트 수는 더 높고 512MB를 테스트했습니다.
- Kafka에서는 해당 주제에 대해 생성된 각 레코드에 대한 키 또는 값(바이트)의 길이가 length.key.value 매개 변수에 의해 제어됩니다. StorageGRID의 경우 S3 오브젝트 수집 및 검색 성능이 더 높은 값으로 향상되었습니다. 예를 들어, 512바이트는 5.8GBps 검색, 1024바이트는 7.5GBps S3 검색, 2048바이트는 10Gbps 가까이 제공

다음 그림은 length.key.value를 기준으로 S3 오브젝트 수집 및 조회 결과를 나타낸 것이다.




- * Kafka 튜닝. * 계층형 스토리지의 성능을 향상시키려면 TierFetcherNumThreads 및 TierArchivernumThreads를 늘릴 수 있습니다. 일반적으로 TierFetchernumThreads를 물리적 CPU 코어 수와 일치하도록 늘리고 TierArchivernumThreads를 CPU 코어 수의 절반으로 늘리고자 합니다. 예를 들어, 서버 속성에서 8개의 물리적 코어가 있는 컴퓨터를 사용하는 경우 confluent.tier.fetcher.num.threads=8 및 confluent.tier.Archiver.num.threads=4를 설정합니다.
- * 항목 삭제에 대한 시간 간격 * 주제가 삭제되면 객체 저장소에서 로그 세그먼트 파일 삭제가 즉시 시작되지 않습니다. 대신 이 파일을 삭제하기 전에 기본값이 3시간인 시간 간격이 있습니다. confluent.tier.topic.delete.check.interval.ms 구성을 수정하여 이 간격의 값을 변경할 수 있습니다. 주제 또는 클러스터를 삭제하는 경우 해당 버킷의 오브젝트도 수동으로 삭제할 수 있습니다.
- * 계층형 스토리지 내부 항목에 대한 ACL. * 온-프레미스 배포의 권장 모범 사례는 계층형 스토리지에 사용되는 내부 항목에 대해 ACL 허가자를 활성화하는 것입니다. 이 데이터에 대한 액세스를 브로커 사용자에게만 제한하려면 ACL 규칙을 설정합니다. 이렇게 하면 내부 항목이 안전하게 보호되며 계층형 스토리지 데이터 및 메타데이터에 대한 무단 액세스가 방지됩니다.

```
kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-
configs.conf \
--add --allow-principal User:<kafka> --operation All --topic "_confluent-
tier-state"
```

ⓘ 사용자 '<Kafka>'를 배포의 실제 브로커 교장으로 바꿉니다.

예를 들어, "confluent-tier-state" 명령은 계층형 스토리지의 내부 항목에 대한 ACL을 설정합니다. 현재 계층형 스토리지와 관련된 내부 주제는 하나뿐입니다. 이 예제에서는 내부 항목의 모든 작업에 대해 Kafka의 주요 권한을 제공하는 ACL을 만듭니다.

## 사이징

Kafka 사이징은 단순, 세분화, 역방향 및 파티션의 4가지 구성 모드로 수행할 수 있습니다.

단순함

단순 모드는 최초 Apache Kafka 사용자 또는 초기 상태 사용 사례에 적합합니다. 이 모드에서는 처리량 MBps, 읽기 팬아웃, 보존 및 리소스 사용률(기본값 60%)과 같은 요구 사항을 제공합니다. 또한 온프레미스(베어 메탈, VMware, Kubernetes 또는 OpenStack) 또는 클라우드와 같은 환경에 진입할 수 있습니다. 이 정보를 바탕으로 Kafka 클러스터의 크기를 조정하면 브로커, zookeeper, Apache Kafka 연결 작업자, 스키마 레지스트리, REST 프록시, ksqlDB 및 Confluorent 제어 센터에 필요한 서버 수가 제공됩니다.

계층형 스토리지의 경우 Kafka 클러스터의 크기를 조정할 수 있는 세분화된 구성 모드를 고려해 보십시오. 세밀한 모드는 숙련된 Apache Kafka 사용자나 잘 정의된 사용 사례에 적합합니다. 이 섹션에서는 생산자, 스트림 프로세서 및 소비자를 위한 크기를 조정하는 방법에 대해 설명합니다.

프로듀서

Apache Kafka의 생산자(예: 네이티브 클라이언트, REST 프록시 또는 Kafka 커넥터)를 설명하기 위해 다음 정보를 제공합니다.

- * 이름. * Spark.
- * Producer type. * 응용 프로그램 또는 서비스, 프록시(RDBMS, MQTT, 기타) 및 기존 데이터베이스(RDBMS, NoSQL, 기타). "모름"을 선택할 수도 있습니다.
- * 초당 이벤트 수 평균 처리량 * (예: 1,000,000).
- * 최대 처리량 * 초당 이벤트 수(예: 4,000,000).
- * 평균 메시지 크기. * (바이트), 압축되지 않음(예: 최대 1MB, 1000).
- * 메시지 형식. * 옵션은 Avro, JSON, 프로토콜 버퍼, 바이너리, 텍스트, “잘 모르겠군요.” 및 기타.
- * 복제 계수. * 옵션은 1, 2, 3(Confluent recommendation), 4, 5, 또는 6.
- * 보존 시간. * 1일(예:) 데이터를 Apache Kafka에 얼마나 오랫동안 저장하기를 원하십니까? 무한 시간 동안 임의의 단위로 -1을 입력합니다. 이 계산기는 보존 기간이 10년으로 무한 경우를 가정합니다.
- "Enable Tiered Storage to Decrease Broker Count and Allow for Infinite Storage?" 확인란을 선택합니다.
- 계층형 스토리지를 사용하는 경우 보존 필드는 브로커에 로컬로 저장된 데이터의 핫 세트를 제어합니다. 아카이브 보존 필드는 아카이브 오브젝트 스토리지에 데이터가 저장되는 기간을 제어합니다.
- * 아카이브 스토리지 보존. * 1년(예:) 아카이브 스토리지에 데이터를 얼마나 오랫동안 저장하려는 경우 무한 기간 동안 임의의 단위로 -1을 입력합니다. 이 계산기는 무한 보존을 위해 10년을 유지하는 것으로 가정합니다.
- * Growth Multiplier. * 1(예:) 이 매개 변수의 값이 현재 처리량을 기반으로 하는 경우 1로 설정합니다. 추가 성장에 따라 크기를 조정하려면 이 매개 변수를 성장 배수로 설정합니다.
- * 생산자 인스턴스 수. * 10(예:) 얼마나 많은 프로듀서 인스턴스가 실행됩니까? 이 입력은 CPU 부하를 사이징 계산에 통합하기 위해 필요합니다. 빈 값은 CPU 로드가 계산에 포함되지 않았음을 나타냅니다.

이 예제 입력에 따라 크기를 조정하면 생산자에 다음과 같은 영향을 줍니다.

- 압축되지 않은 바이트의 평균 처리량: 1Gbps. 압축되지 않은 바이트 단위의 최대 처리량: 4Gbps. 압축된 바이트의 평균 처리량: 400Mbps 압축된 바이트 단위의 최대 처리량: 1.6GBps. 이 값은 기본 60% 압축률을 기반으로 합니다(이 값은 변경할 수 있음).
    - 필요한 총 온브로커 핫 세트 스토리지 수: 31,104TB(복제 포함), 압축. 총 비브로커 아카이브 스토리지 필요: 378,432TB, 압축. 사용 "https://fusion.netapp.com" StorageGRID 사이징:

스트림 프로세서는 아파치 Kafka로부터 데이터를 소비하고 아파치 Kafka로 다시 생산하는 애플리케이션 또는 서비스를

설명해야 합니다. 대부분의 경우 ksqlDB 또는 Kafka 스트림에 빌드됩니다.

- * 이름. * Spark streamer.
- * 처리 시간. * 이 프로세서는 단일 메시지를 처리하는 데 얼마나 걸립니까?
  - 1ms(단순한 상태 비저장 변환) [예], 10ms(Stateful 인메모리 작업).
  - 100ms(stateful 네트워크 또는 디스크 작동), 1000ms(타사 REST 호출)
  - 이 매개 변수를 벤치마킹하고 시간이 얼마나 걸리는지 정확히 알고 있습니다.
- * 출력 보존. * 1일(예) 스트림 프로세서는 Apache Kafka로 출력을 다시 생성합니다. 이 출력 데이터를 Apache Kafka에 얼마나 오랫동안 저장하기를 원하십니까? 무한 기간 동안 임의의 단위로 -1을 입력합니다.
- "계층 스토리지를 사용하여 브로커 수를 줄이고 무한 확장 스토리지를 허용하시겠습니까?" 확인란을 선택합니다.
- * 아카이브 스토리지 보존. * 1년(예:) 아카이브 스토리지에 데이터를 얼마나 오랫동안 저장하려는 경우 무한 기간 동안 임의의 단위로 -1을 입력합니다. 이 계산기는 무한 보존을 위해 10년을 유지하는 것으로 가정합니다.
- * 출력 통과 백분율. * 100(예:) 스트림 프로세서는 Apache Kafka로 출력을 다시 생성합니다. Apache Kafka로 다시 출력되는 인바운드 처리량의 비율은 얼마입니까? 예를 들어, 인바운드 처리량이 20Mbps이고 이 값이 10이면 출력 처리량은 2Mbps입니다.
- 어떤 응용 프로그램에서 이 정보를 읽습니까? 프로듀서 유형 기반 사이징에 사용되는 이름인 “Spark”를 선택합니다. 위의 입력을 기반으로 스트림 프로세스 인스턴스 및 주제 파티션 예상에 대한 사이징의 영향을 예상할 수 있습니다.
- 이 스트림 프로세서 응용 프로그램에는 다음과 같은 수의 인스턴스가 필요합니다. 들어오는 항목에는 이러한 많은 파티션이 필요할 수 있습니다. 이 매개 변수를 확인하려면 Confluent에 문의하십시오.
  - 성장 승수가 없는 평균 처리량에 대해 1,000개
  - 최대 처리량에 대해 4,000(성장 승수 없음)
  - 성장 승수가 포함된 평균 처리량의 경우 1,000개
  - 최대 처리량에 대해 4,000(성장 승수 포함)

소비자

Apache Kafka의 데이터를 사용하고 Apache Kafka로 다시 생성하지 않는 애플리케이션 또는 서비스(예: 네이티브 클라이언트 또는 Kafka Connector)에 대해 설명하십시오.

- * 이름. * Spark 소비자.
- * 처리 시간. * 이 소비자는 단일 메시지를 처리하는 데 얼마나 걸립니까?
  - 1ms(예: 로깅 같은 간단하고 상태 비저장 작업)
  - 10ms(데이터 저장소에 대한 빠른 쓰기)
  - 100ms(데이터 저장소에 대한 느린 쓰기)
  - 1000ms(타사 휴면 통화)
  - 알려진 기간의 다른 벤치마크 프로세스
- * 소비자 유형. * 기존 데이터 저장소(RDBMS, NoSQL, 기타)에 대한 애플리케이션, 프록시 또는 싱크
- 어떤 응용 프로그램에서 이 정보를 읽습니까? 이 매개 변수를 이전에 결정된 생산자 및 스트림 사이징에 연결합니다.

위의 입력 내용에 따라 소비자 인스턴스 및 주제 파티션 추정치에 대한 사이징을 결정해야 합니다. 소비자 응용 프로그램에는 다음과 같은 수의 인스턴스가 필요합니다.

- 평균 처리량에 대해 2,000개, 성장 승수 없음
- 최대 처리량에 대해 8,000개, 성장 승수 없음
- 성장 승수를 포함한 평균 처리량에 대해 2,000개
- 성장 승수를 포함한 최대 처리량에 대해 8,000개

들어오는 주제에는 이 수의 파티션도 필요할 것입니다. 확인하려면 Confluent에 문의하십시오.

생산자, 스트림 프로세서 및 소비자에 대한 요구 사항 외에도 다음과 같은 추가 요구 사항을 제공해야 합니다.

- * 재생성 시간. * 예: 4시간. Apache Kafka 브로커 호스트에 장애가 발생하고 데이터가 손실되며 장애가 발생한 호스트를 대체하기 위해 새 호스트를 프로비저닝하는 경우 이 새 호스트 재구축 속도는 얼마나 빨라야 합니까? 값을 알 수 없는 경우 이 매개 변수를 비워 둡니다.
- * 리소스 활용률 목표(백분율) * 예: 60. 평균 처리량 중에 호스트를 얼마나 활용하기를 원하십니까? Confluent는 Confluent 셀프 밸런싱 클러스터를 사용하고 있지 않는 한 60%의 사용률을 권장합니다. 이 경우 활용률이 더 높을 수 있습니다.

환경에 대해 설명하십시오

- * 클러스터가 어떤 환경에서 실행됩니까? * Amazon Web Services, Microsoft Azure, Google 클라우드 플랫폼, 베어 메탈 온프레미스, VMware 온프레미스, 사내에 OpenStack 또는 온프레미스에 Kubernates가 있습니까?
- * 호스트 세부 정보. * 코어 수: 48(예:), 네트워크 카드 유형(10GbE, 40GbE, 16GbE, 1GbE 또는 다른 유형).
- * 스토리지 볼륨. * 호스트: 12(예:) 호스트당 지원되는 하드 드라이브 또는 SSD 수는 몇 개입니까? Confluent는 호스트당 12개의 하드 드라이브를 권장합니다.
- * 스토리지 용량/볼륨(GB). * 1000(예:) 단일 볼륨에서 몇 기가바이트의 스토리지를 저장할 수 있습니까? Confluent에서는 1TB 디스크를 권장합니다.
- * 스토리지 구성. * 스토리지 볼륨은 어떻게 구성됩니까? Confluent는 Raid10에서 모든 Confluent 기능을 이용할 것을 권장합니다. JBOD, SAN, RAID 1, RAID 0, RAID 5, 및 기타 유형도 지원됩니다.
- * 단일 볼륨 처리량(Mbps). * 125(예:) 단일 스토리지 볼륨이 초당 메가바이트 단위로 읽거나 쓸 수 있는 속도는 얼마나 됩니까? Confluent는 일반적으로 125MBps 처리량의 표준 하드 드라이브를 권장합니다.
- * 메모리 용량(GB). * 64(예:)

환경 변수를 결정한 후 클러스터 크기를 선택합니다. 위에 표시된 예시 매개 변수를 토대로 Confluent Kafka에 대한 다음 사이징을 결정했습니다.

- * 아파치 Kafka. * 브로커 수: 22. 클러스터가 스토리지에 바인딩되어 있습니다. 호스트 수를 줄이고 무한 스토리지를 허용하도록 계층형 스토리지를 설정하는 것이 좋습니다.
- * Apache ZooKeeper. * Count:5; Apache Kafka Connect 작업자: Count:2; Schema Registry: Count:2; REST Proxy: Count:2; ksqlDB:Count:2; Confluorent Control Center: Count:1.

사용 사례를 염두에 두고 플랫폼 팀에 리버스 모드를 사용합니다. 파티션 모드를 사용하여 단일 항목에 필요한 파티션 수를 계산합니다. 을 참조하십시오 https://eventsizer.io 역 및 파티션 모드에 따른 크기 조정.

## 결론

이 문서에서는 검증 테스트, 계층형 스토리지 성능 결과, 튜닝, Confluent S3 커넥터 및 셀프 밸런싱 기능을 포함하여 NetApp 스토리지와 함께 Contuent Tiered Storage를 사용하기 위한

모범 사례 지침을 제공합니다. ILM 정책, 검증을 위한 다양한 성능 테스트 및 산업 표준 S3 API를 통한 유창한 성능을 고려할 때 NetApp StorageGRID 오브젝트 스토리지는 유창한 계층형 스토리지를 위한 최적의 선택입니다.

추가 정보를 찾을 수 있는 위치

이 문서에 설명된 정보에 대해 자세히 알아보려면 다음 문서 및/또는 웹 사이트를 검토하십시오.

- 아파치 카프카란

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

- NetApp 제품 설명서

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

- S3 싱크 매개 변수 세부 정보

  "https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options"

- 아파치 카프카

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

- Confluent 플랫폼의 무한 스토리지

  "https://www.confluent.io/blog/infinite-kafka-storage-in-confluent-platform/"

- ContFluent Tiered Storage - 모범 사례 및 사이징

  "https://docs.confluent.io/platform/current/kafka/tiered-storage.html#best-practices-and-recommendations"

- Confluent 플랫폼용 Amazon S3 싱크 커넥터

  "https://docs.confluent.io/kafka-connect-s3-sink/current/overview.html"

- Kafka 사이징

  "https://eventsizer.io"

- StorageGRID 사이징

  "https://fusion.netapp.com/"

- Kafka 활용 사례

  "https://kafka.apache.org/uses"

- confluent platform 6.0의 자체 균형 조정 Kafka 클러스터

  "https://www.confluent.io/blog/self-balancing-kafka-clusters-in-confluent-platform-6-0/"

  "https://www.confluent.io/blog/confluent-platform-6-0-delivers-the-most-powerful-event-streaming-platform-

to-date/"

# NetApp 하이브리드 클라우드 데이터 솔루션 - 고객 사용 사례를 기반으로 Spark 및 Hadoop

## TR-4657: NetApp 하이브리드 클라우드 데이터 솔루션 - 고객 사용 사례를 기반으로 Spark 및 Hadoop

NetApp의 Karthikeyan Nagalingam 및 Sathish Thyagarajan

본 문서는 NetApp AFF 및 FAS 스토리지 시스템, NetApp Cloud Volumes ONTAP, NetApp 연결형 스토리지, Spark 및 Hadoop용 NetApp FlexClone 기술을 사용하는 하이브리드 클라우드 데이터 솔루션에 대해 설명합니다. 이러한 솔루션 아키텍처를 통해 고객은 자신의 환경에 적합한 데이터 보호 솔루션을 선택할 수 있습니다. NetApp은 고객과의 상호 작용 및 비즈니스 사용 사례를 기반으로 이러한 솔루션을 설계했습니다. 이 문서에서는 다음과 같은 자세한 정보를 제공합니다.

- Spark 및 Hadoop 환경 및 고객의 당면 과제를 해결하기 위해 데이터 보호가 필요한 이유
- NetApp 비전 및 구성 요소와 서비스를 기반으로 하는 Data Fabric
- 이러한 구성 요소를 사용하여 유연한 데이터 보호 워크플로우를 구축하는 방법
- 실제 고객 사용 사례를 기반으로 한 여러 아키텍처의 장단점을 설명합니다. 각 활용 사례는 다음과 같은 구성 요소를 제공합니다.
  - 고객 시나리오
  - 요구사항 및 당면 과제
  - 해결하세요
  - 솔루션 요약

**Hadoop** 데이터 보호를 선택해야 하는 이유

Hadoop 및 Spark 환경에서는 다음과 같은 문제를 해결해야 합니다.

- * 소프트웨어 또는 사용자 오류. * Hadoop 데이터 작업을 수행하는 동안 소프트웨어 업데이트에 사람의 실수가 발생할 수 있으며, 이로 인해 작업에 예상치 못한 결과가 발생할 수 있습니다. 이 경우 오류나 부당한 결과를 방지하려면 데이터를 보호해야 합니다. 예를 들어, 소프트웨어 업데이트가 제대로 실행되지 않아 트래픽 신호 데이터를 일반 텍스트 형식으로 제대로 분석하지 못하는 새로운 기능이 추가되었습니다. 이 소프트웨어는 JSON 및 기타 비 텍스트 파일 형식을 분석하여 실시간 트래픽 제어 분석 시스템을 통해 데이터 포인트가 누락된 예측 결과를 생성합니다. 이 상황은 출력 결함을 초래하여 교통 신호에서 사고를 일으킬 수 있습니다. 데이터 보호는 이전 작업 중인 응용 프로그램 버전으로 빠르게 롤백할 수 있는 기능을 제공하여 이 문제를 해결할 수 있습니다.
- * 규모와 확장성 * 데이터 소스와 볼륨의 수가 계속 증가함에 따라 분석 데이터의 크기가 매일 증가하고 있습니다. 소셜 미디어, 모바일 앱, 데이터 분석 및 클라우드 컴퓨팅 플랫폼은 현재 빅데이터 시장의 주요 데이터 소스로서 빠르게 증가하고 있으며, 따라서 정확한 데이터 운영을 위해 데이터를 보호해야 합니다.
- * Hadoop의 기본 데이터 보호. * Hadoop에는 데이터를 보호하는 기본 명령이 있지만 이 명령은 백업 중에 데이터의 일관성을 제공하지 않습니다. 디렉토리 레벨 백업만 지원합니다. Hadoop에서 생성된 스냅샷은 읽기 전용이며 백업 데이터를 직접 재사용하는 데 사용할 수 없습니다.

**Hadoop** 및 **Spark** 고객의 데이터 보호 당면 과제

Hadoop 및 Spark 고객의 일반적인 과제는 데이터 보호 중에 운영 클러스터의 성능에 부정적인 영향을 주지 않고 백업 시간을 단축하고 백업 안정성을 높이는 것입니다.

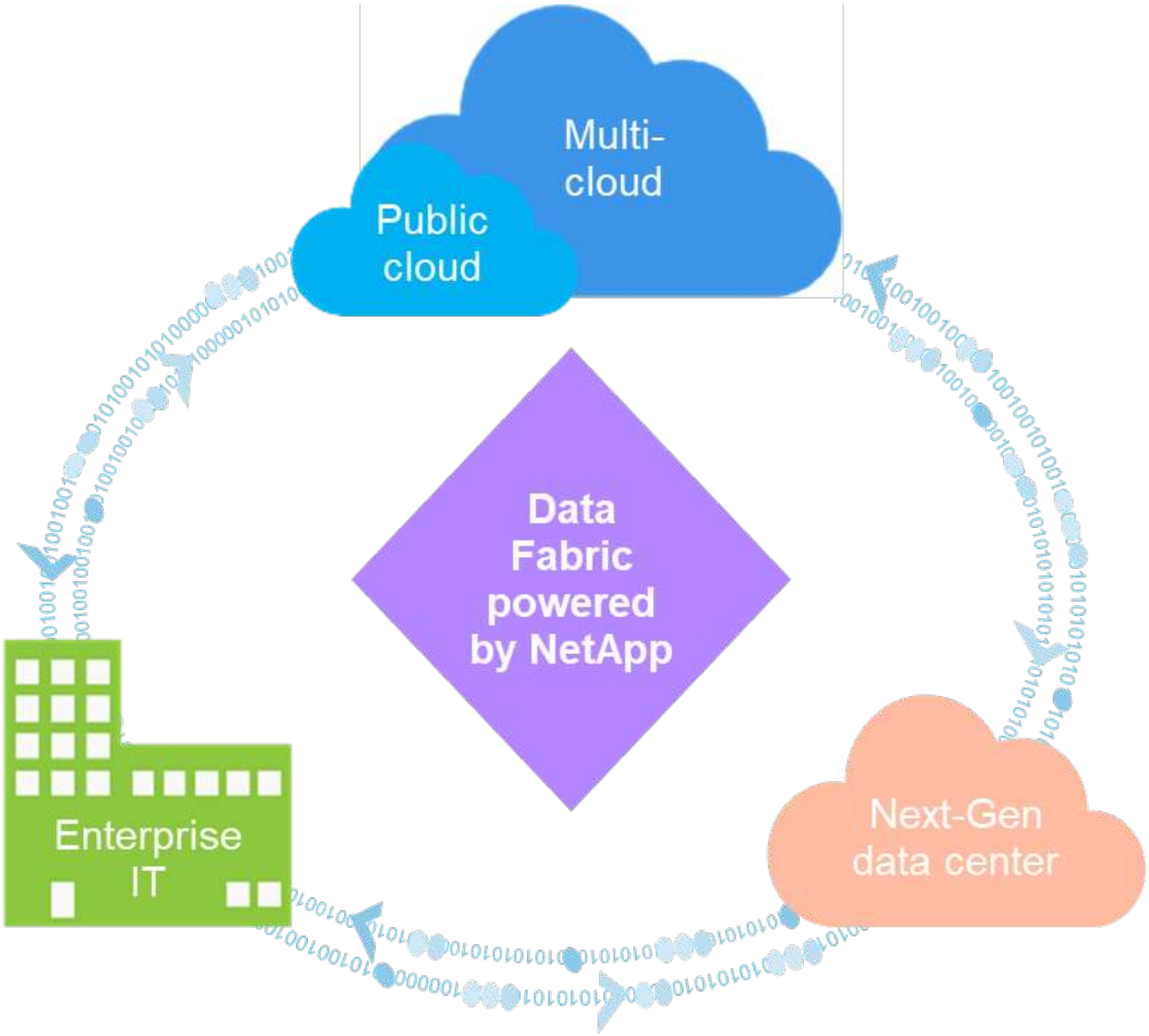
또한 고객은 RPO(복구 시점 목표) 및 RTO(복구 시간 목표) 다운타임을 최소화하고 사내 및 클라우드 기반 재해 복구 사이트를 제어하여 비즈니스 연속성을 최적화해야 합니다. 이 제어 기능은 일반적으로 엔터프라이즈급 관리 툴을 사용하는 데서 비롯됩니다.

Hadoop과 Spark 환경은 데이터 볼륨이 엄청나며 증가하기 때문에 복잡합니다. 하지만 데이터가 도착하는 속도는 점점 증가하고 있습니다. 이 시나리오를 통해 소스 데이터에서 효율적인 최신 DevTest 및 QA 환경을 빠르게 생성하기가 어렵습니다. NetApp은 이러한 과제를 인식하고 이 백서에 제공된 솔루션을 제공합니다.

## **NetApp**에서 제공하는 빅데이터 아키텍처 관련 **Data Fabric**

### NetApp이 제공하는 Data Fabric은 클라우드 및 사내 환경에서 데이터 관리를 단순화하고 통합하여 디지털 전환을 가속합니다.

NetApp이 제공하는 Data Fabric은 데이터 가시성과 통찰력, 데이터 액세스 및 제어, 데이터 보호 및 보안을 위한 일관성 있는 통합 데이터 관리 서비스 및 애플리케이션(구성 요소)을 제공합니다(아래 그림 참조).




검증된 **Data Fabric** 고객 사용 사례

NetApp이 제공하는 Data Fabric은 고객에게 다음과 같은 9가지 검증된 사용 사례를 제공합니다.

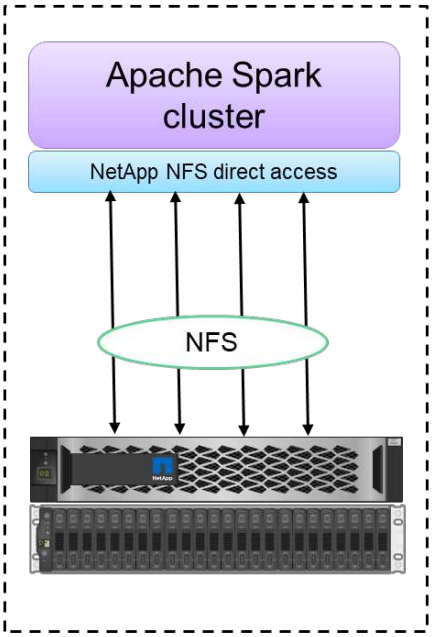
- 분석 워크로드 가속화
- DevOps 혁신 가속
- 클라우드 호스팅 인프라 구축
- 클라우드 데이터 서비스 통합
- 데이터 보호 및 보안
- 비정형데이터의 최적화
- 데이터 센터의 효율성 확보
- 데이터 통찰력 및 제어 제공
- 단순화 및 자동화

본 문서는 9가지 사용 사례 중 2가지(솔루션 포함)를 다룹니다.

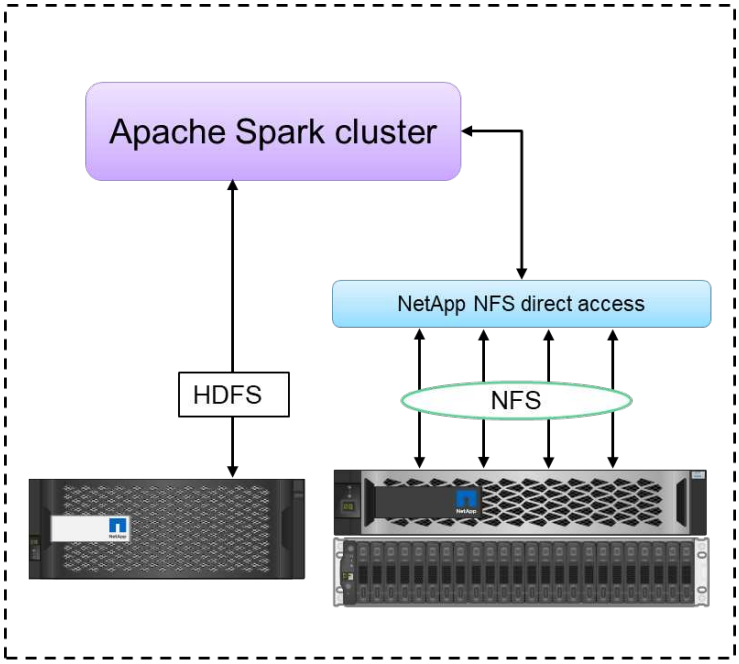
- 분석 워크로드 가속화
- 데이터 보호 및 보안

**NetApp NFS** 직접 액세스

NetApp NFS를 사용하면 고객이 데이터를 이동하거나 복사하지 않고도 기존 또는 새로운 NFSv3 또는 NFSv4 데이터에 대해 빅데이터 분석 작업을 실행할 수 있습니다. 여러 데이터 복제본을 방지하고 소스와 데이터를 동기화할 필요가 없습니다. 예를 들어 금융 부문에서 데이터를 한 위치에서 다른 위치로 이동하는 것은 쉬운 일이 아닌 법적 의무를 준수해야 합니다. 이 시나리오에서는 NetApp NFS 직접 액세스가 원래의 위치에서 재무 데이터를 분석합니다. 또 다른 주요 이점은 NetApp NFS 직접 액세스를 사용하여 기본 Hadoop 명령을 사용하여 Hadoop 데이터를 간편하게 보호하고 NetApp의 강력한 데이터 관리 포트폴리오를 활용하여 데이터 보호 워크플로우를 활성화할 수 있다는 것입니다.




Configuration 1: NFS as primary storage




Configuration 2: HDFS and NFS in single Spark cluster

NetApp NFS 직접 액세스에서는 Hadoop/Spark 클러스터에 다음과 같은 두 가지 유형의 구축 옵션을 제공합니다.

- 기본적으로 Hadoop/Spark 클러스터는 데이터 스토리지와 기본 파일 시스템에 HDFS(Hadoop Distributed File System)를 사용합니다. NetApp NFS 직접 액세스는 기본 HDFS를 NFS 스토리지로 대체하여 NFS 데이터에 대한 직접 분석 작업을 지원합니다.
- 다른 구축 옵션에서 NetApp NFS 직접 액세스는 NFS를 단일 Hadoop/Spark 클러스터의 HDFS와 함께 추가 스토리지로 구성할 수 있도록 지원합니다. 이 경우 고객은 NFS 내보내기를 통해 데이터를 공유하고 HDFS 데이터와 함께 동일한 클러스터에서 데이터를 액세스할 수 있습니다.

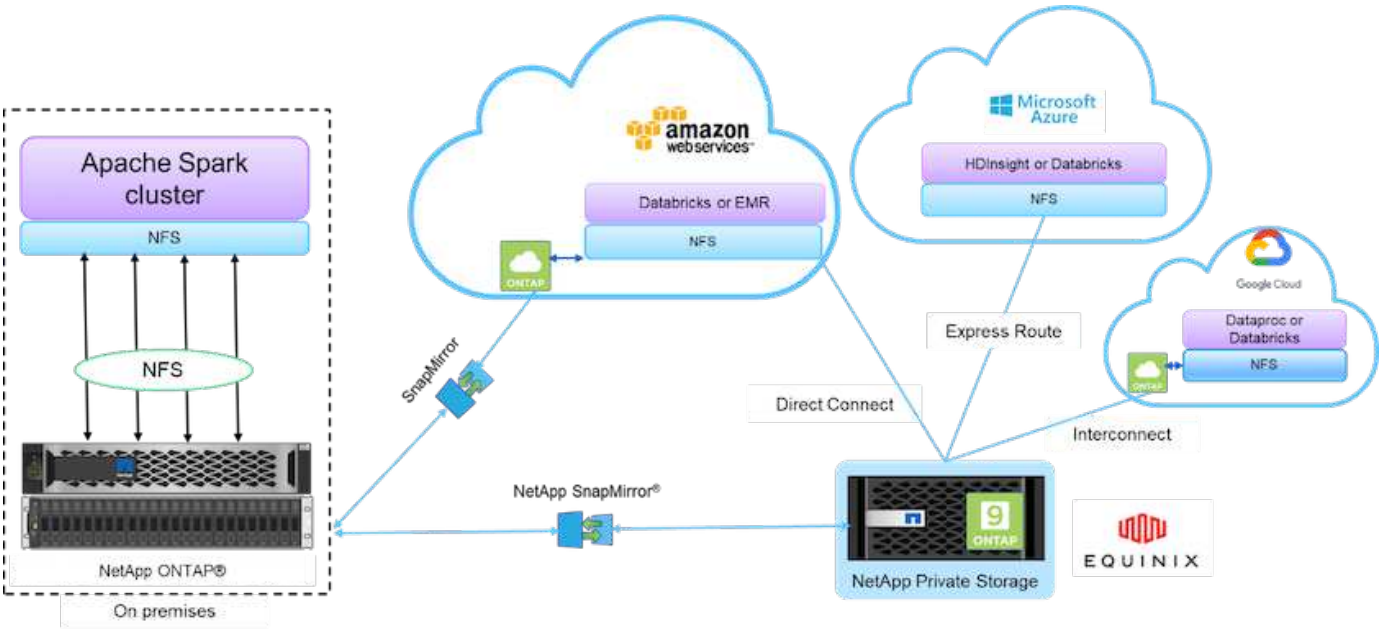
NetApp NFS 직접 액세스를 사용할 때의 주요 이점은 다음과 같습니다.

- 현재 위치의 데이터를 분석하여 분석 데이터를 HDFS와 같은 Hadoop 인프라스트럭처로 이동하는 데 시간과 성능이 많이 소모되는 작업을 방지합니다.
- 복제본 수를 3개부터 1개로 줄입니다.

- 사용자가 컴퓨팅 및 스토리지를 분리하여 독립적으로 확장할 수 있습니다.
- ONTAP의 강력한 데이터 관리 기능을 활용하여 엔터프라이즈 데이터 보호 기능을 제공합니다.
- Hortonworks 데이터 플랫폼에서 인증되었습니다.
- 하이브리드 데이터 분석을 구축할 수 있습니다.
- 동적 다중 스레드 기능을 활용하여 백업 시간을 단축합니다.

빅 데이터를 위한 구성 요소

NetApp 기반의 Data Fabric은 아래 그림과 같이 데이터 액세스, 제어, 보호 및 보안을 위한 데이터 관리 서비스와 애플리케이션(구성 요소)을 통합합니다.




위 그림의 구성 요소는 다음과 같습니다.

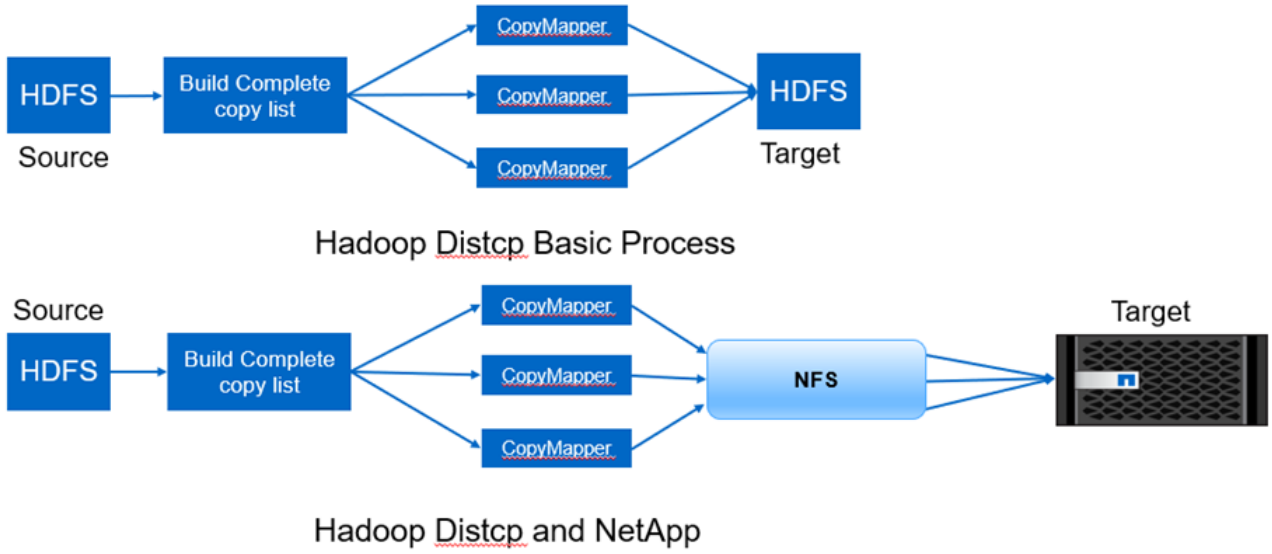
- * NetApp NFS 직접 액세스 * 는 추가 소프트웨어 또는 드라이버 요구사항 없이 최신 Hadoop 및 Spark 클러스터를 NetApp NFS 볼륨에 직접 액세스할 수 있도록 지원합니다.
- * NetApp Cloud Volumes ONTAP 및 클라우드 볼륨 서비스. * Microsoft Azure 클라우드 서비스의 AWS(Amazon Web Services) 또는 ANF(Azure NetApp Files)에서 실행되는 ONTAP를 기반으로 하는 소프트웨어 정의 연결형 스토리지.
- NetApp SnapMirror 기술 *. 사내 및 ONTAP 클라우드 또는 NPS 인스턴스 간에 데이터 보호 기능을 제공합니다.
- * 클라우드 서비스 공급자 * 이러한 공급자에는 AWS, Microsoft Azure, Google Cloud 및 IBM Cloud가 포함됩니다.
- * PaaS. * AWS의 EMR(Amazon Elastic MapReduce) 및 Databricks와 같은 클라우드 기반 분석 서비스와 Microsoft Azure HDInsight 및 Azure Databricks

## Hadoop 데이터 보호 및 NetApp

Hadoop DistCp는 대규모 클러스터 간 및 클러스터 간 복제에 사용되는 기본 툴입니다. 아래 그림에 표시된 Hadoop DistCp 기본 프로세스는 MapReduce와 같은 Hadoop 기본 툴을 사용하여 HDFS 소스에서 해당 타겟으로 Hadoop 데이터를 복제하는 일반적인 백업

워크플로우입니다.

NetApp NFS 직접 액세스를 통해 고객은 NFS를 Hadoop DistCp 툴의 타겟 타겟으로 설정하여 MapReduce를 통해 HDFS 소스에서 NFS 공유로 데이터를 복사할 수 있습니다. NetApp NFS 직접 액세스는 DistCp 툴을 위한 NFS 드라이버 역할을 합니다.




## Hadoop 데이터 보호 활용 사례 개요

이 섹션에서는 이 백서의 초점을 구성하는 데이터 보호 활용 사례에 대해 자세히 설명합니다. 나머지 섹션에서는 고객 문제(시나리오), 요구 사항 및 당면 과제, 솔루션 등 각 사용 사례에 대한 자세한 정보를 제공합니다.

### 사용 사례 1: Hadoop 데이터 백업

이 사용 사례에서 NetApp NFS 볼륨을 통해 대형 금융 기관의 긴 백업 시간을 24시간 이상에서 몇 시간 이하로 줄일 수 있습니다.

### 사용 사례 2: 클라우드에서 사내로 백업 및 재해 복구

NetApp에서 제공하는 Data Fabric을 구성 요소로 사용함으로써 대규모 브로드캐스트 회사는 주문형, 즉각적인 데이터 전송 같은 다양한 데이터 전송 모드에 따라 클라우드 데이터를 사내 데이터 센터에 백업하는 요구사항을 충족시킬 수 있었습니다. 또는 Hadoop/Spark 클러스터 로드를 기반으로 합니다.

### 사용 사례 3: 기존 Hadoop 데이터에 대해 DevTest 활성화

NetApp 솔루션을 통해 온라인 음악 유통업체는 다양한 지점에서 공간 효율적인 다중 Hadoop 클러스터를 신속하게 구축하여 보고서를 생성하고 예약된 정책을 사용하여 일일 DevTest 작업을 실행할 수 있었습니다.

### 사용 사례 4: 데이터 보호 및 멀티 클라우드 연결

한 대형 서비스 공급자는 NetApp이 제공하는 Data Fabric을 사용하여 다양한 클라우드 인스턴스에서 고객에게 멀티 클라우드 분석을 제공합니다.

사용 사례 **5:** 분석 워크로드 가속화

가장 큰 금융 서비스 및 투자 은행 중 하나는 NetApp 네트워크 연결 스토리지 솔루션을 사용하여 I/O 대기 시간을 줄이고 정량 재무 분석 플랫폼을 가속화했습니다.

## 사용 사례 **1: Hadoop** 데이터 백업

시나리오

이 시나리오에서는 고객이 대규모 사내 Hadoop 저장소를 보유하고 있으며 재해 복구를 위해 백업하려고 합니다. 그러나 고객의 현재 백업 솔루션은 비용이 많이 들고 24시간 이상의 긴 백업 윈도우에 어려움을 겪고 있습니다.

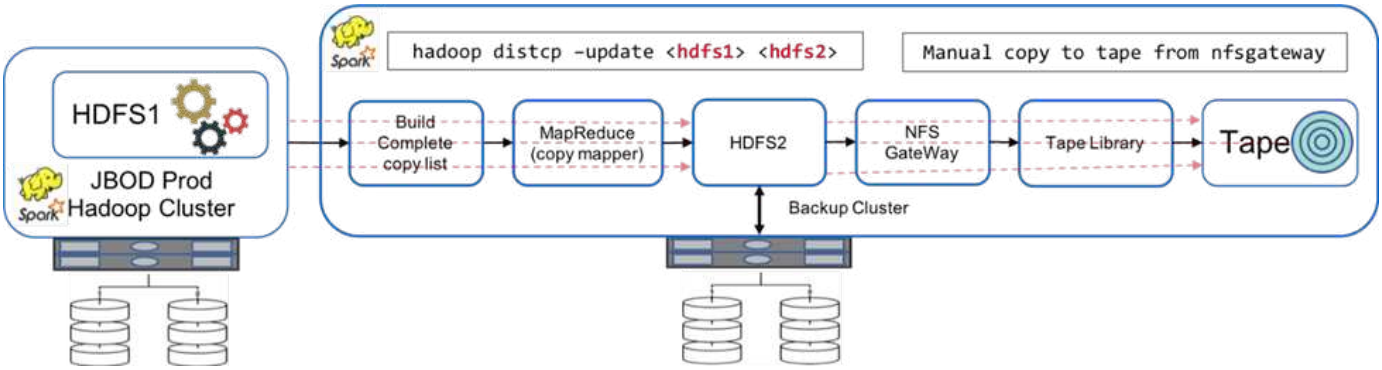
요구사항 및 당면 과제

이 사용 사례의 주요 요구사항과 과제는 다음과 같습니다.

- 소프트웨어 하위 호환성:
  - 제안된 대체 백업 솔루션은 운영 Hadoop 클러스터에 사용되는 현재 실행 중인 소프트웨어 버전과 호환되어야 합니다.
- 확정된 SLA를 충족하기 위해 제안된 대체 솔루션은 매우 낮은 RPO 및 RTO를 달성해야 합니다.
- NetApp 백업 솔루션을 통해 생성된 백업은 데이터 센터에서 로컬로 구축된 Hadoop 클러스터뿐만 아니라 원격 사이트의 재해 복구 위치에서 실행되는 Hadoop 클러스터에서도 사용할 수 있습니다.
- 제안된 솔루션은 비용 효율적이어야 합니다.
- 제안 솔루션은 백업 시간 동안 현재 실행 중인 운영 중인 분석 작업의 성능 영향을 줄여야 합니다.

고객의 기존 백업 솔루션 **x**

아래 그림은 원래 Hadoop 네이티브 백업 솔루션을 보여 줍니다.




운영 데이터는 중간 백업 클러스터를 통해 테이프로 보호됩니다.

- HDFS1 데이터는 'Hadoop distcp-update<hdfs1><hdfs2>' 명령을 실행하여 HDFS2에 복사됩니다.
- 백업 클러스터는 NFS 게이트웨이 역할을 하며 테이프 라이브러리를 통해 Linux 'CP' 명령을 통해 테이프에 데이터를 수동으로 복사합니다.

원래 Hadoop 네이티브 백업 솔루션의 이점은 다음과 같습니다.

- 이 솔루션은 Hadoop 기본 명령을 기반으로 하므로 사용자가 새로운 절차를 배울 필요가 없습니다.

- 이 솔루션은 업계 표준 아키텍처와 하드웨어를 활용합니다.

원래 Hadoop 네이티브 백업 솔루션의 단점은 다음과 같습니다.

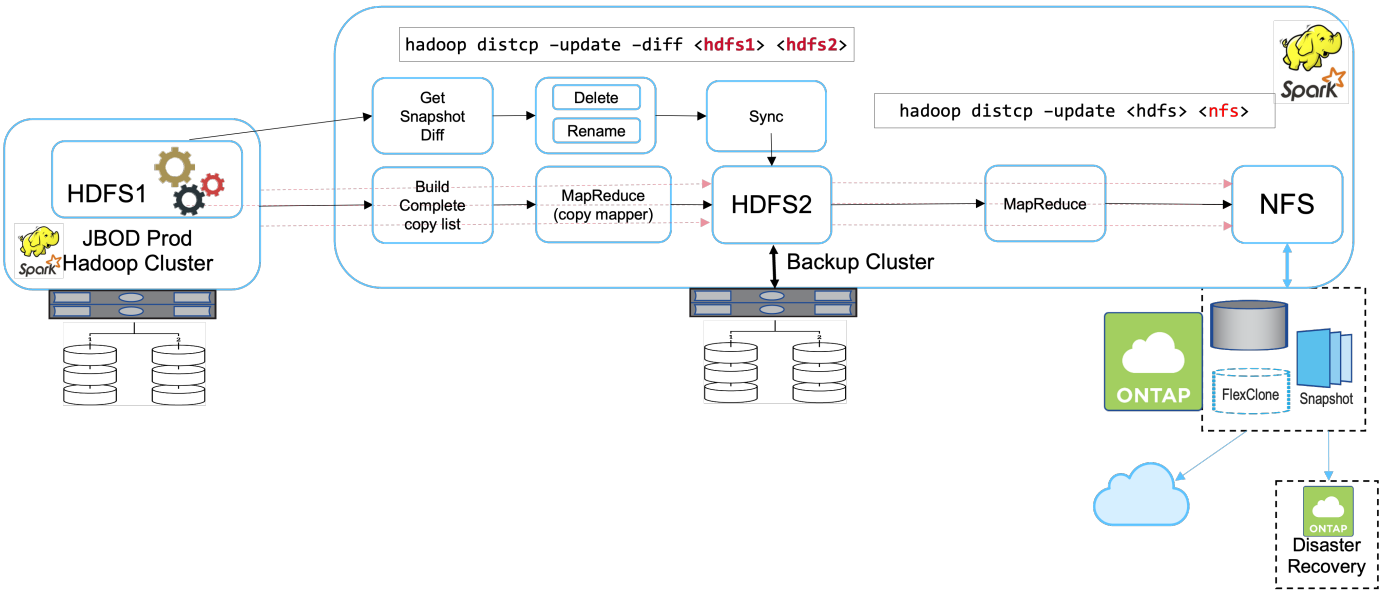
- 긴 백업 시간이 24시간을 초과하므로 운영 데이터가 취약해집니다.
- 백업 시간 동안 클러스터 성능이 크게 저하되었습니다.
- 테이프에 복사하는 것은 수동 프로세스입니다.
- 백업 솔루션은 필요한 하드웨어 및 수동 프로세스에 필요한 인력 시간의 측면에서 비용이 많이 듭니다.

## 백업 솔루션

이러한 당면 과제와 요구 사항을 바탕으로 기존 백업 시스템을 고려하여 세 가지 가능한 백업 솔루션을 제안하였습니다. 다음 하위 섹션에서는 이러한 세 가지 백업 솔루션 각각에 대해 설명합니다. 솔루션 A에서 솔루션 C까지의 레이블이 지정되어 있습니다

### 솔루션 A

솔루션 A에서 백업 Hadoop 클러스터는 보조 백업을 NetApp NFS 스토리지 시스템으로 전송하므로 아래 그림과 같이 테이프 요구 사항이 해소됩니다.




솔루션 A에 대한 자세한 작업은 다음과 같습니다.

- 운영 Hadoop 클러스터에는 보호가 필요한 HDFS에 고객의 분석 데이터가 있습니다.
- HDFS가 포함된 백업 Hadoop 클러스터는 데이터의 중간 위치로 작동합니다. JBOD(Just a Bunch of Disks)는 운영 및 백업 Hadoop 클러스터 모두에서 HDFS용 스토리지를 제공합니다.
- Hadoop distcp –update –diff <hdfs1><hdfs2>' 명령을 실행하여 운영 클러스터 HDFS에서 백업 클러스터 HDFS로 Hadoop 운영 데이터를 보호합니다.

> Hadoop 스냅샷은 운영 환경에서 백업 Hadoop 클러스터로 데이터를 보호하는 데 사용됩니다.

- NetApp ONTAP 스토리지 컨트롤러는 NFS로 내보낸 볼륨을 제공하며 백업 Hadoop 클러스터에 프로비저닝됩니다.

- 를 실행합니다 `Hadoop distcp` MapReduce 및 여러 매퍼를 활용하여 분석 데이터는 백업 Hadoop 클러스터에서 NFS로 보호됩니다.

  NetApp 스토리지 시스템의 NFS에 데이터가 저장된 후 NetApp Snapshot, SnapRestore 및 FlexClone 기술을 사용하여 필요에 따라 Hadoop 데이터를 백업, 복원, 복제할 수 있습니다.

> SnapMirror 기술을 사용하여 Hadoop 데이터를 클라우드뿐 아니라 재해 복구 위치에도 보호할 수 있습니다.

솔루션 A의 이점은 다음과 같습니다.

- Hadoop 운영 데이터는 백업 클러스터로부터 보호됩니다.
- HDFS 데이터는 NFS를 통해 보호되므로 클라우드 및 재해 복구 위치에 대한 보호가 가능합니다.
- 백업 작업을 백업 클러스터로 오프로드하여 성능을 향상시킵니다.
- 수동 테이프 작업이 필요 없습니다
- NetApp 툴을 통해 엔터프라이즈 관리 기능을 지원합니다.
- 기존 환경을 최소한으로 변경해야 합니다.
- 비용 효율적인 솔루션입니다.

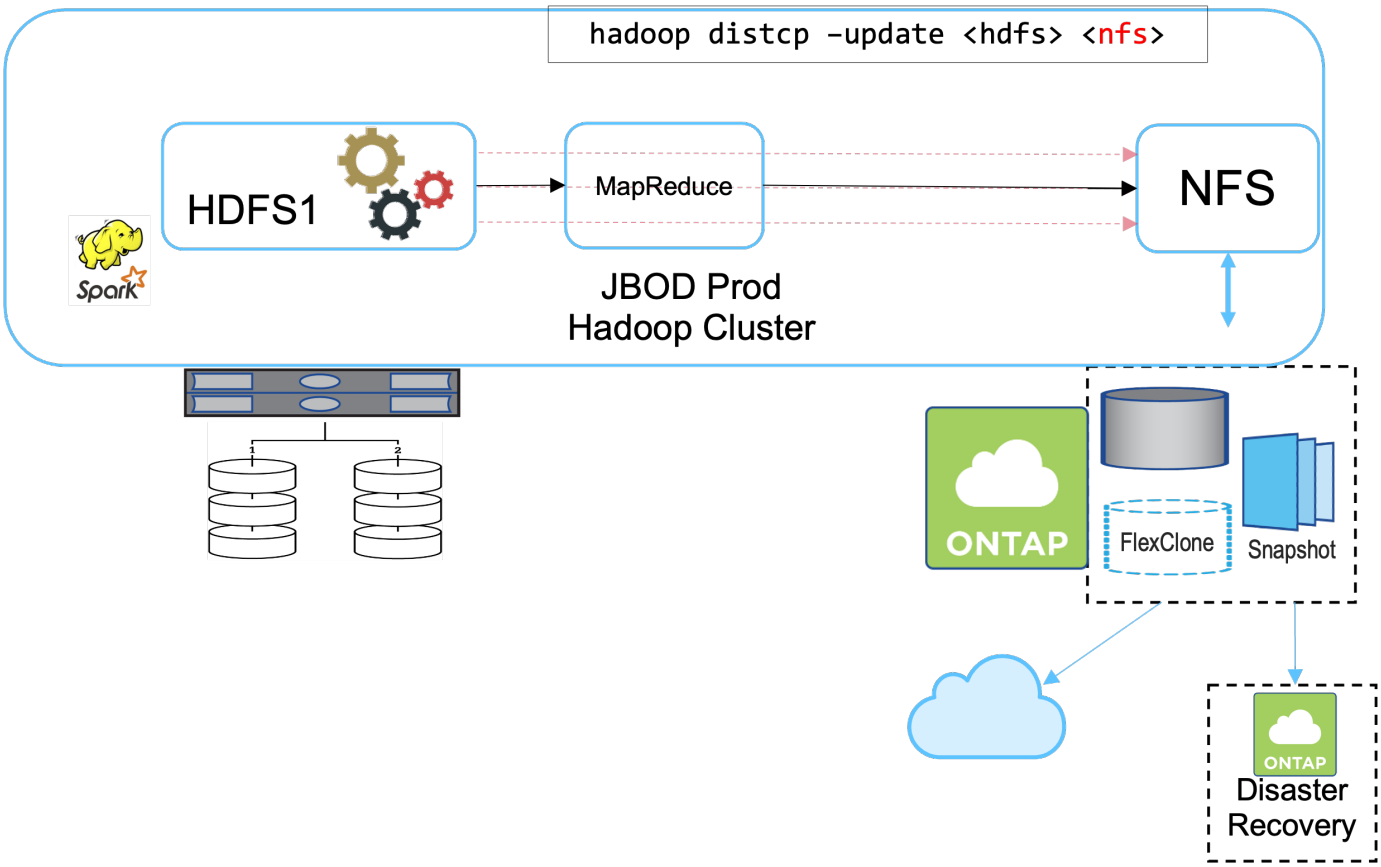
이 솔루션의 단점은 성능을 향상시키기 위해 백업 클러스터와 추가 매퍼가 필요하다는 것입니다.

이 고객은 단순성, 비용, 전반적인 성능 때문에 최근에 솔루션 A를 배포했습니다.

이 솔루션에서는 JBOD 대신 ONTAP의 SAN 디스크를 사용할 수 있습니다. 이 옵션은 백업 클러스터 스토리지 로드를 ONTAP로 오프로드하지만, 단점은 SAN 패브릭 스위치가 필요하다는 점입니다.

#### 해결 방법 B

솔루션 B는 아래 그림과 같이 NFS 볼륨을 운영 Hadoop 클러스터에 추가하므로 백업 Hadoop 클러스터가 필요하지 않습니다.




솔루션 B에 대한 자세한 작업은 다음과 같습니다.

- NetApp ONTAP 스토리지 컨트롤러는 운영 Hadoop 클러스터에 NFS 내보내기를 프로비저닝합니다.

  Hadoop의 기본 구성 `hadoop distcp` 명령은 운영 클러스터 HDFS에서 NFS로 Hadoop 데이터를 보호합니다.

- NetApp 스토리지 시스템의 NFS에 데이터가 저장된 후에는 Snapshot, SnapRestore 및 FlexClone 기술을 사용하여 필요에 따라 Hadoop 데이터를 백업, 복원, 복제할 수 있습니다.

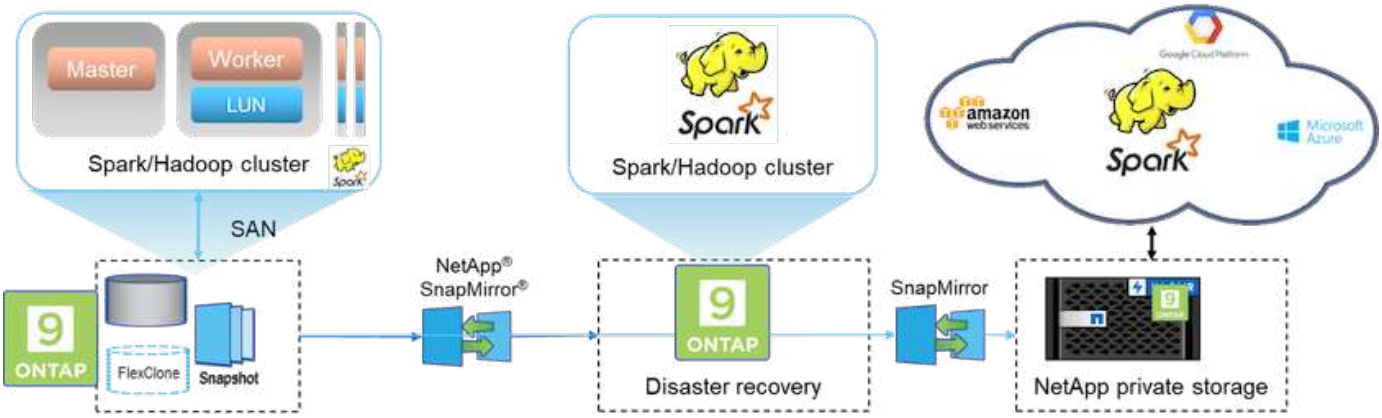
솔루션 B의 이점은 다음과 같습니다.

- 운영 클러스터는 백업 솔루션에 맞게 약간 수정되어 구축이 간소화되고 추가 인프라스트럭처 비용이 절감됩니다.
- 백업 작업을 위한 백업 클러스터는 필요하지 않습니다.
- HDFS 운영 데이터는 NFS 데이터 변환 시 보호됩니다.
- 이 솔루션을 사용하면 NetApp 툴을 통해 엔터프라이즈 관리 기능을 수행할 수 있습니다.

이 솔루션의 단점은 프로덕션 클러스터에 구현되어 운영 클러스터에 추가 관리자 작업을 추가할 수 있다는 것입니다.

#### 솔루션 C

솔루션 C에서는 아래 그림과 같이 NetApp SAN 볼륨을 HDFS 스토리지용 Hadoop 운영 클러스터에 직접 프로비저닝합니다.




솔루션 C에 대한 자세한 단계는 다음과 같습니다.

- NetApp ONTAP SAN 스토리지는 HDFS 데이터 스토리지를 위한 운영 Hadoop 클러스터에서 프로비저닝됩니다.
- NetApp Snapshot 및 SnapMirror 기술은 운영 Hadoop 클러스터의 HDFS 데이터를 백업하는 데 사용됩니다.
- 백업이 스토리지 계층에 있기 때문에 스냅샷 복사본 백업 프로세스 중에 Hadoop/Spark 클러스터의 운영에 미치는 성능 영향은 없습니다.

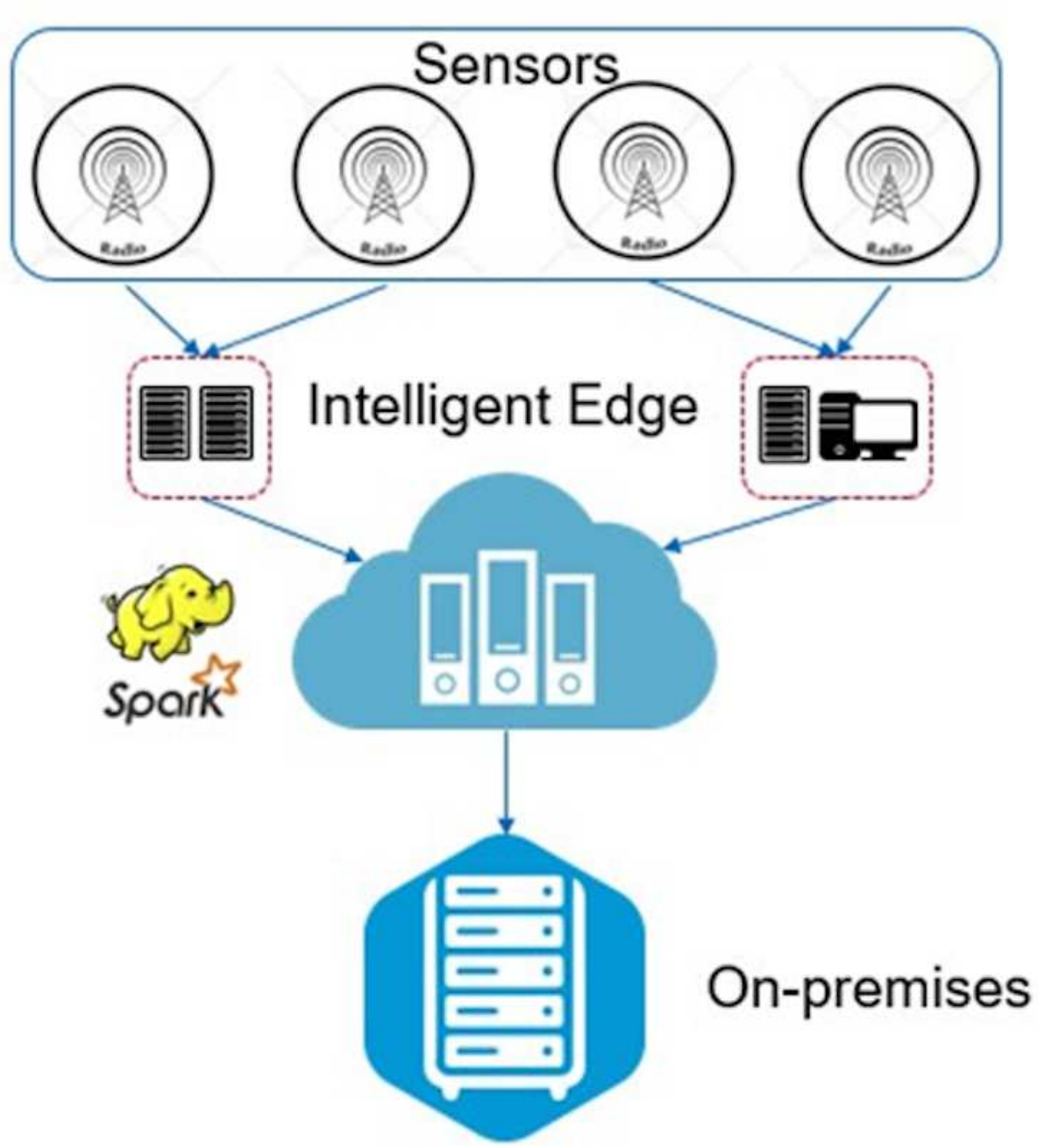
> 스냅샷 기술은 데이터 크기에 관계없이 몇 초 내에 백업을 완료합니다.

솔루션 C의 이점은 다음과 같습니다.

- 스냅샷 기술을 사용하여 공간 효율적인 백업을 생성할 수 있습니다.
- NetApp 툴을 통해 엔터프라이즈 관리 기능을 지원합니다.

## 사용 사례 2: 클라우드에서 사내로 백업 및 재해 복구

이 사용 사례는 아래 그림과 같이 클라우드 기반 분석 데이터를 사내 데이터 센터에 백업해야 하는 브로드캐스트 고객을 기반으로 합니다.




시나리오

이 시나리오에서는 IoT 센서 데이터가 클라우드로 수집되고 AWS 내의 오픈 소스 Apache Spark 클러스터를 사용하여 분석됩니다. 따라서 클라우드에서 처리된 데이터를 사내로 백업해야 합니다.

요구사항 및 당면 과제

이 사용 사례의 주요 요구사항과 과제는 다음과 같습니다.

- 데이터 보호를 사용하도록 설정하면 클라우드의 운영 Spark/Hadoop 클러스터에 성능 영향이 미치지 않습니다.
- 클라우드 센서 데이터를 효율적이고 안전한 방식으로 이동하고 사내로 보호해야 합니다.
- 온디맨드, 즉각적인, 낮은 클러스터 로드 시간 등 다양한 조건에서 데이터를 클라우드로 유연하게 전송할 수 있습니다.

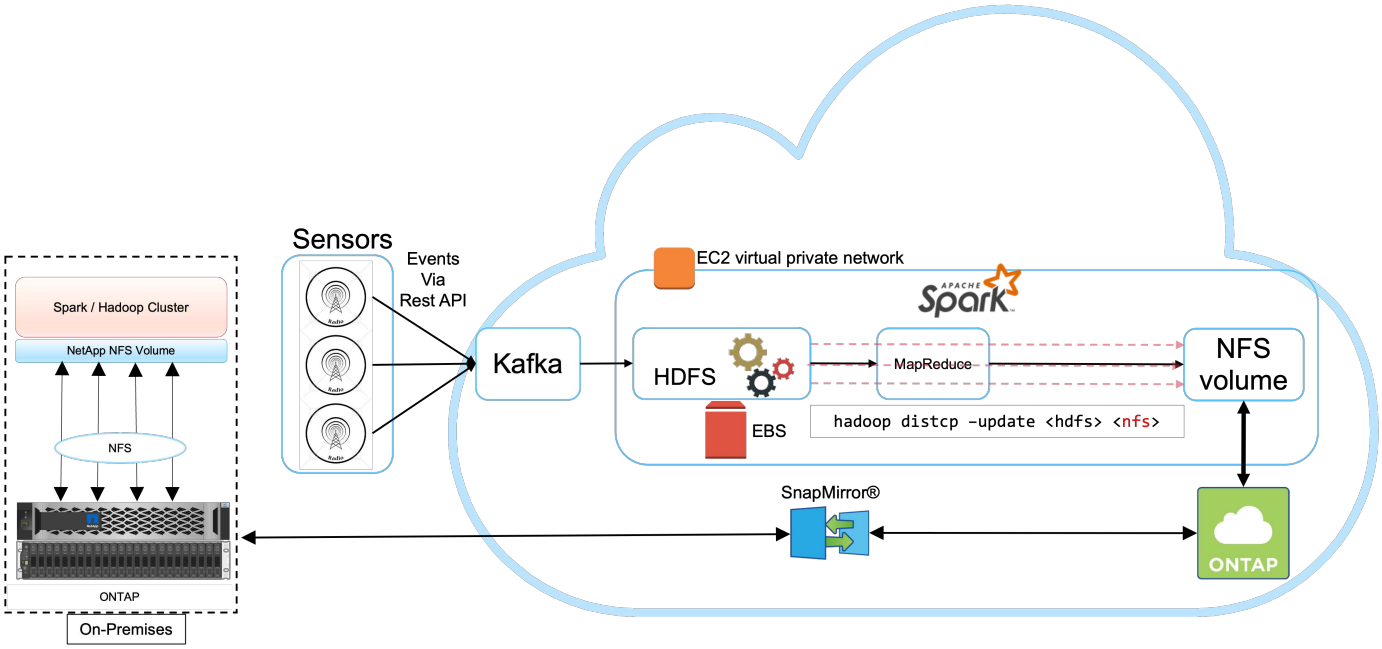
해결 방법

이 고객은 Spark 클러스터 HDFS 스토리지에 AWS EBS(Elastic Block Store)를 사용하여 Kafka를 통해 원격 센서에서 데이터를 수신 및 수집했습니다. 따라서 HDFS 스토리지는 백업 데이터의 소스 역할을 합니다.

이러한 요구사항을 충족하기 위해 NetApp ONTAP Cloud를 AWS에 구축하고 NFS 공유를 생성하여 Spark/Hadoop 클러스터의 백업 타겟 역할을 수행합니다.

NFS 공유가 생성된 후 HDFS EBS 스토리지의 데이터를 ONTAP NFS 공유로 복사합니다. ONTAP 클라우드의 NFS에 데이터가 상주한 후에는 SnapMirror 기술을 사용하여 필요에 따라 클라우드에서 사내 스토리지로 데이터를 안전하고 효율적으로 미러링할 수 있습니다.

이 이미지는 클라우드에서 사내 솔루션으로 백업 및 재해 복구를 보여줍니다.




## 사용 사례 3: 기존 Hadoop 데이터에 대해 DevTest 활성화

이 사용 사례에서 고객은 DevTest와 보고 목적으로 동일한 데이터 센터와 원격 위치에서 대량의 분석 데이터를 포함하는 기존 Hadoop 클러스터를 기반으로 새로운 Hadoop/Spark 클러스터를 신속하고 효율적으로 구축해야 합니다.

시나리오

이 시나리오에서는 사내 및 재해 복구 위치에 대규모 Hadoop 데이터 레이크를 구축하고 Spark/Hadoop 클러스터를 여러 개 구축했습니다.

요구사항 및 당면 과제

이 사용 사례의 주요 요구사항과 과제는 다음과 같습니다.

- DevTest, QA 또는 동일한 운영 데이터에 액세스해야 하는 기타 목적으로 여러 Hadoop 클러스터를 생성합니다. 여기서 문제는 매우 큰 Hadoop 클러스터를 공간 효율적인 방식으로 여러 번 즉시 클론 복제해야 한다는 것입니다.
- 운영 효율성을 위해 Hadoop 데이터를 DevTest 및 보고 팀에 동기화합니다.

- 운영 클러스터와 새 클러스터 간에 동일한 자격 증명을 사용하여 Hadoop 데이터를 배포합니다.
- 예약된 정책을 사용하여 운영 클러스터에 영향을 주지 않고 QA 클러스터를 효율적으로 생성합니다.

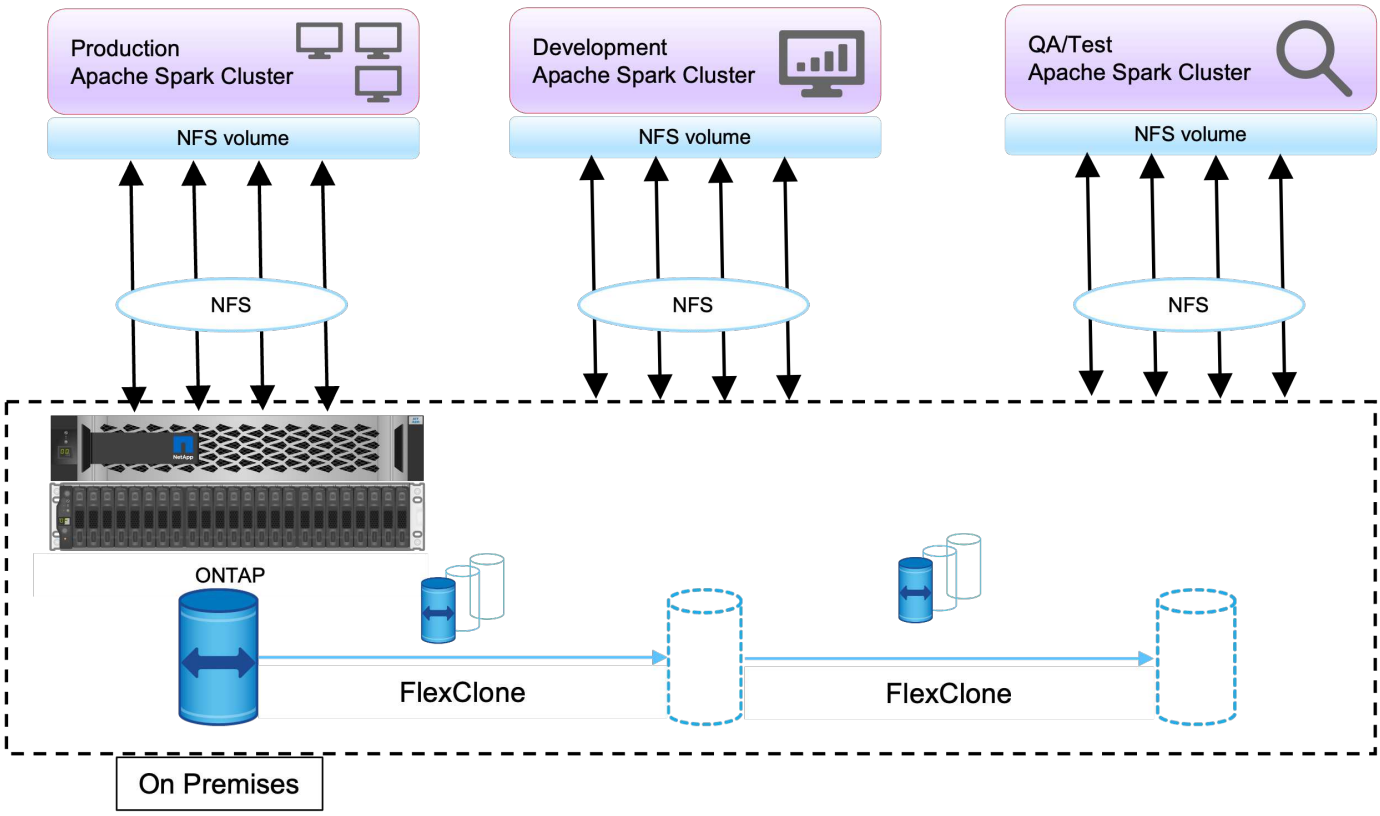
해결 방법

FlexClone 기술은 방금 설명한 요구 사항에 응답하는 데 사용됩니다. FlexClone 기술은 스냅샷 복사본의 읽기/쓰기 복사본입니다. 이 기능은 상위 스냅샷 복사본 데이터에서 데이터를 읽고 새 블록/수정된 블록에 대해 추가 공간만 사용합니다. 빠르고 공간 효율적입니다.

먼저, NetApp 일관성 그룹을 사용하여 기존 클러스터의 스냅샷 복사본을 생성했습니다.

NetApp System Manager 또는 스토리지 관리 프롬프트 내의 Snapshot 복사본 일관성 그룹 스냅샷 복사본은 애플리케이션 정합성이 보장된 그룹 스냅샷 복사본이며, FlexClone 볼륨은 일관성 그룹 스냅샷 복사본을 기반으로 생성됩니다. FlexClone 볼륨이 상위 볼륨의 NFS 엑스포트 정책을 상속한다는 점을 언급하는 것이 좋습니다. 스냅샷 복사본이 생성된 후에는 아래 그림과 같이 DevTest 및 보고를 위해 새 Hadoop 클러스터를 설치해야 합니다. 새 Hadoop 클러스터의 클론 복제된 NFS 볼륨은 NFS 데이터에 액세스합니다.

이 이미지는 DevTest용 Hadoop 클러스터를 보여 줍니다.




## 사용 사례 4: 데이터 보호 및 멀티 클라우드 연결

이 사용 사례는 고객의 빅데이터 분석 데이터를 위한 멀티 클라우드 연결을 제공하는 클라우드 서비스 파트너와 관련이 있습니다.

시나리오

이 시나리오에서는 여러 소스에서 AWS로 수신된 IoT 데이터가 NPS의 중앙 위치에 저장됩니다. NPS 스토리지는

AWS 및 Azure에 위치한 Spark/Hadoop 클러스터에 연결되므로, 여러 클라우드에서 실행되는 빅데이터 분석 애플리케이션이 동일한 데이터에 액세스할 수 있습니다.

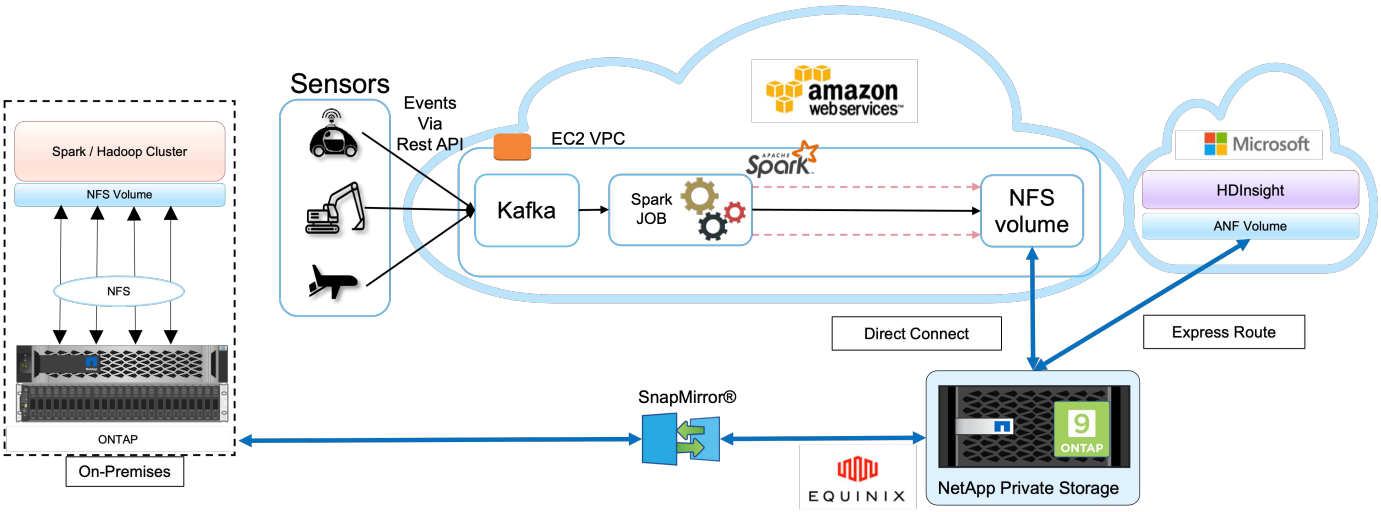
요구사항 및 당면 과제

이 사용 사례의 주요 요구사항과 과제는 다음과 같습니다.

- 고객은 여러 클라우드를 사용하여 동일한 데이터에 대한 분석 작업을 실행하려고 합니다.
- 다양한 센서와 허브를 통해 사내, 클라우드와 같은 다양한 소스로부터 데이터를 받아야 합니다.
- 솔루션은 효율적이고 비용 효율적이어야 합니다.
- 주요 과제는 사내 및 다른 클라우드 간에 하이브리드 분석 서비스를 제공하는 비용 효율적이고 효율적인 솔루션을 구축하는 것입니다.

해결 방법

이 이미지는 데이터 보호 및 멀티 클라우드 연결 솔루션을 보여줍니다.




위 그림과 같이 센서의 데이터가 스트리밍되어 Kafka를 통해 AWS Spark 클러스터로 수집됩니다. 이 데이터는 Equinix 데이터 센터 내의 클라우드 공급자 외부에 있는 NPS에 있는 NFS 공유에 저장됩니다. NetApp NPS는 Direct Connect 및 Express Route 연결을 통해 Amazon AWS 및 Microsoft Azure에 연결되므로 고객은 Amazon 및 AWS 분석 클러스터 모두에서 NFS 데이터에 액세스할 수 있습니다. 이 접근 방식은 여러 하이퍼 스케일러의 클라우드 분석 문제를 해결합니다.

따라서 사내 스토리지와 NPS 스토리지 모두 ONTAP 소프트웨어를 실행하므로 SnapMirror는 사내 클러스터에 NPS 데이터를 미러링하여 사내 및 여러 클라우드에 하이브리드 클라우드 분석을 제공할 수 있습니다.

최적의 성능을 위해 일반적으로 여러 네트워크 인터페이스 및 직접 연결/빠른 경로를 사용하여 클라우드 인스턴스에서 데이터에 액세스할 것을 권장합니다.

## 사용 사례 5: 분석 워크로드 가속화

이 시나리오에서는 NetApp NFS 스토리지 솔루션을 사용하여 대형 금융 서비스 및 투자 은행의 분석 플랫폼을 현대화함으로써 자산 관리 및 정량적 사업부의 투자 위험 및 파생물 분석에 상당한 개선을 이루었습니다.

시나리오

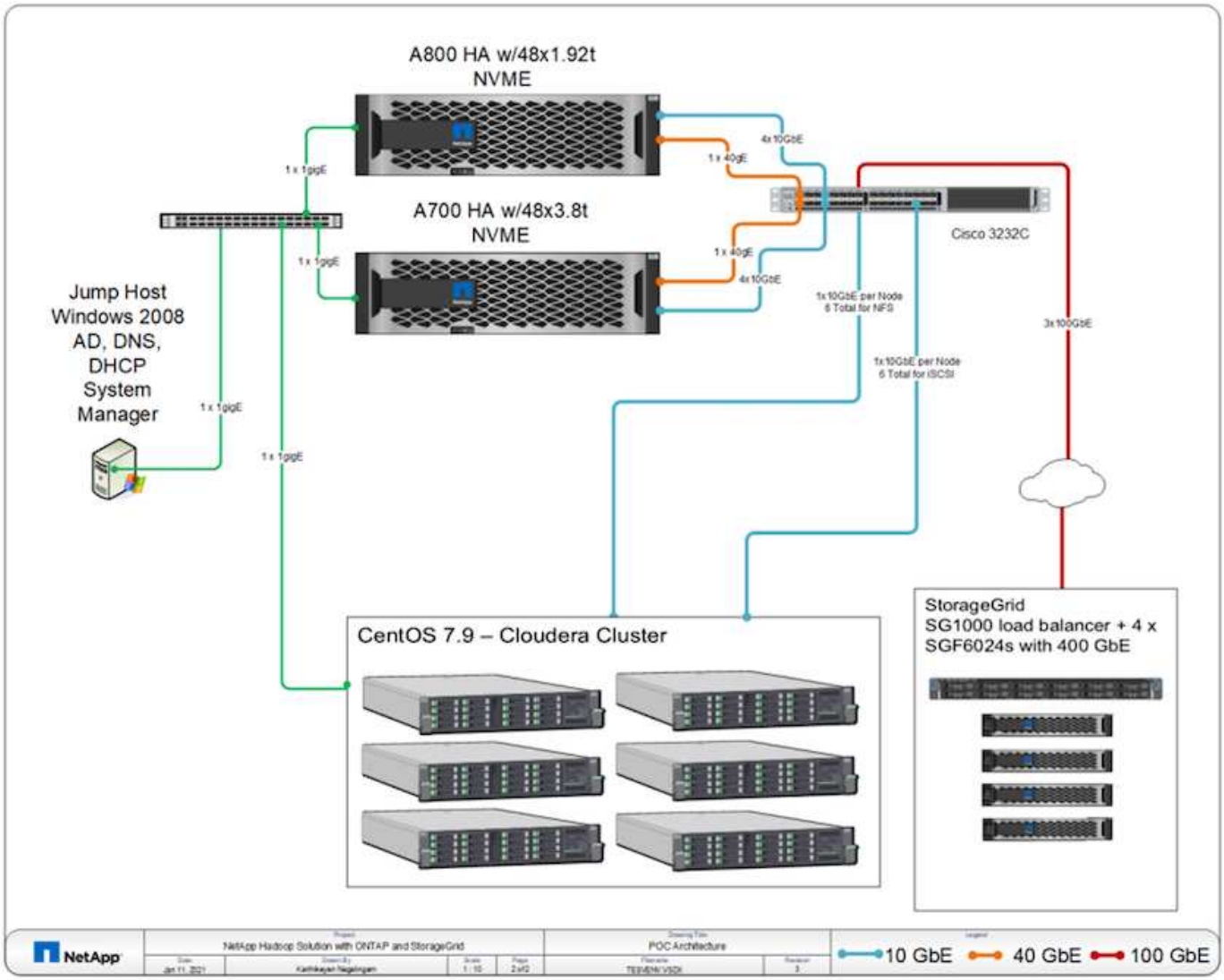
고객의 기존 환경에서 분석 플랫폼에 사용되는 Hadoop 인프라는 Hadoop 서버의 내부 스토리지를 활용했습니다. JBOD 환경의 독점 특성으로 인해 조직 내의 많은 내부 고객은 실시간 데이터의 반복 샘플에 의존하는 시뮬레이션인 Monte Carlo 정량 모델을 활용할 수 없었습니다. 시장 이동이 불확실성의 영향을 이해하는 데 있어 최적화되지 않은 능력은 정량적 자산 관리 사업부에 비호의적으로 작용했습니다.

요구사항 및 당면 과제

이 은행의 정량적 사업부는 정확하고 시기 적절한 예측을 얻기 위한 효율적인 예측 방법을 원했습니다. 이를 위해 IT 팀에서는 인프라를 현대화하고, 기존 I/O 대기 시간을 줄이며, Hadoop, Spark와 같은 분석 애플리케이션의 성능을 개선하여 투자 모델을 효율적으로 시뮬레이션하고, 잠재적인 이익을 측정하고, 위험을 분석해야 한다는 사실을 깨달았습니다.

해결 방법

고객은 기존 Spark 솔루션에 대한 JBOD를 가지고 있었습니다. 그런 다음 NetApp ONTAP, NetApp StorageGRID 및 MinIO Gateway to NFS를 활용하여 잠재적인 이익과 위험을 평가하는 투자 모델에 대한 시뮬레이션 및 분석을 실행하는 금융 그룹의 정량적 대기 시간을 줄였습니다. 이 이미지는 NetApp 스토리지의 Spark 솔루션을 보여줍니다.




위 그림과 같이 StorageGRID A800, A700 시스템 및 AFF는 Spark가 있는 6노드 Hadoop 클러스터 및 데이터 분석

작업을 위한 YARN 및 Hive 메타데이터 서비스의 NFS 및 S3 프로토콜을 통해 쪽모이 세공 파일에 액세스하기 위해 구축되었습니다.

고객의 기존 환경에서 DAS(직접 연결 스토리지) 솔루션을 사용할 경우 컴퓨팅과 스토리지를 독립적으로 확장한다는 단점이 있었습니다. Spark용 NetApp ONTAP 솔루션을 통해 은행의 재무 분석 사업부에서 스토리지를 컴퓨팅에서 분리하여 필요에 따라 인프라 리소스를 보다 효율적으로 제공할 수 있게 되었습니다.

NFS와 함께 ONTAP를 사용함으로써 컴퓨팅 서버 CPU는 Spark SQL 작업에 거의 전적으로 활용되었고 I/O 대기 시간이 약 70% 감소되어 스파크 워크로드에 더 나은 컴퓨팅 성능과 성능 향상을 제공했습니다. 또한 CPU 활용률을 높임으로써 고객이 GPUDirect와 같은 GPU를 활용하여 플랫폼을 더욱 현대화할 수 있도록 했습니다. 또한 StorageGRID는 스파크 워크로드를 위한 저렴한 스토리지 옵션을 제공하며 MinIO 게이트웨이는 S3 프로토콜을 통해 NFS 데이터에 대한 안전한 액세스를 제공합니다. 클라우드의 데이터에 대해 NetApp은 Cloud Volumes ONTAP, Azure NetApp Files 및 NetApp Cloud Volumes Service를 권장합니다.

## 결론

### 이 섹션에서는 다양한 Hadoop 데이터 보호 요구사항을 충족하기 위해 NetApp에서 제공하는 사용 사례 및 솔루션을 요약합니다. 고객은 NetApp이 제공하는 Data Fabric을 사용하여 다음을 수행할 수 있습니다.

- NetApp의 풍부한 데이터 관리 기능과 Hadoop 기본 워크플로우와의 통합을 활용하여 적합한 데이터 보호 솔루션을 유연하게 선택할 수 있습니다.
- Hadoop 클러스터의 백업 윈도우 시간을 약 70% 단축
- Hadoop 클러스터 백업으로 인한 성능 영향을 없앱니다.
- 여러 클라우드 공급자로부터 데이터 보호 및 데이터 액세스를 동시에 단일 분석 데이터 소스에 제공
- FlexClone ® 기술을 사용하여 빠르고 공간 효율적인 Hadoop 클러스터 복사본을 생성합니다.

추가 정보를 찾을 수 있는 위치

이 문서에 설명된 정보에 대한 자세한 내용은 다음 문서 및/또는 웹 사이트를 참조하십시오.

- NetApp 빅데이터 분석 솔루션

  "https://www.netapp.com/us/solutions/applications/big-data-analytics/index.aspx"

- NetApp 스토리지를 사용하는 Apache Spark 워크로드

  https://www.netapp.com/pdf.html?item=/media/26877-nva-1157-deploy.pdf

- Apache Spark용 NetApp 스토리지 솔루션

  "https://www.netapp.com/media/16864-tr-4570.pdf"

- NetApp에서 지원하는 Data Fabric 기반 Apache Hadoop

  "https://www.netapp.com/media/16877-tr-4529.pdf"

감사의 말

- Paul Burland, 영업 담당자, ANZ Victoria District Sales, NetApp
- Hoseb Dermanilian, NetApp 비즈니스 개발 매니저
- Lee Dorrier, NetApp MPSG 이사
- David Thiessen, 시스템 엔지니어, ANZ Victoria District SE, NetApp

버전 기록

| 버전 | 날짜 | 문서 버전 기록 |
|---|---|---|
| 버전 1.0 | 2018년 1월 | 최초 릴리스 |
| 버전 2.0 | 2021년 10월 | 사용 사례 5로 업데이트: 분석 워크로드 가속화 |
| 버전 3.0 | 2023년 11월 | NIPAM 세부 정보가 제거되었습니다 |

# 최신 데이터 분석 - 다양한 분석 전략을 위한 다양한 솔루션

이 백서에서는 NetApp의 최신 데이터 분석 솔루션 전략을 설명합니다. 비즈니스 성과, 고객 과제, 기술 동향, 경쟁 레거시 아키텍처, 최신 워크플로우, 사용 사례, 산업, 클라우드, 기술 파트너, Data Mover, NetApp Active IQ, NetApp DataOps 툴킷, Hadoop~Spark, NetApp Astra Control, 컨테이너, 엔터프라이즈 데이터 관리, 아카이빙, 계층화를 지원하는 소프트웨어 정의 스토리지, AI 및 분석 목표를 달성하는 방법, NetApp과 고객이 함께 데이터 아키텍처를 현대화하는 방법을 알아봅니다.

"최신 데이터 분석 - 다양한 분석 전략을 위한 다양한 솔루션"

# TR-4623: NetApp E-Series E5700 및 Splunk Enterprise

Mitch Blackburn, NetApp

TR-4623에서는 NetApp E-Series 및 Splunk 설계의 통합 아키텍처에 대해 설명합니다. 노드 스토리지의 균형, 안정성, 성능, 스토리지 용량, 밀도 및 이 설계에서는 Splunk clustered 인덱스 노드 모델을 채택하므로 확장성이 높고 TCO가 낮습니다. 스토리지를 컴퓨팅에서 분리하여 개별적으로 확장할 수 있으므로 오버프로비저닝 비용을 절감할 수 있습니다. 또한, 이 문서에는 Splunk 머신 로그 이벤트 시뮬레이션 툴에서 얻은 성능 테스트 결과가 요약되어 있습니다.

"TR-4623: NetApp E-Series E5700 및 Splunk Enterprise"

# NVA-1157 - 배포: NetApp 스토리지 솔루션을 사용한 Apache Spark 워크로드

NetApp의 Karthikeyan Nagalingam입니다

NVA-1157-deploy는 NetApp NFS AFF 스토리지 시스템에서 Apache Spark SQL의 성능 및

기능 검증을 설명합니다. 다양한 시나리오를 기반으로 구성, 아키텍처, 성능 테스트를 검토하고 Spark with NetApp ONTAP 데이터 관리 소프트웨어 사용을 위한 권장사항을 살펴보십시오. 또한 JBOD(Just a Bunch of Disks) 와 NetApp AFF A800 스토리지 컨트롤러를 기반으로 한 테스트 결과에 대해서도 다룹니다.

"NVA-1157 - 배포: NetApp 스토리지 솔루션을 사용한 Apache Spark 워크로드"