最新のデータ分析
NetApp Solutions

NetApp
September 10, 2024

目次

ネットアップの最新データ分析ソリューション

NetApp File-Object DualityとAWS SageMakerを使用したクラウドデータ管理

TR-4967：『Cloud Data Management with NetApp File-Object Duality and AWS SageMaker』

ネットアップ Karthikeyan Nagalingam

データサイエンティストやエンジニアは、NFS形式で保存されたデータにアクセスする必要があることがよくありますが、AWSはS3バケットへのアクセスのみをサポートしているため、AWS SageMakerのS3プロトコルからこのデータに直接アクセスすることは困難です。ただし、NetApp ONTAP は、NFSとS3へのデュアルプロトコルアクセスを可能にすることで解決策 を提供します。この解決策 を使用すると、データサイエンティストやエンジニアは、NetApp Cloud Volumes ONTAP のS3バケットを介して、AWS SageMakerノートブックのNFSデータにアクセスできます。このアプローチでは、追加のソフトウェアを必要とせずに、NFSとS3の両方から同じデータに簡単にアクセスして共有できます。

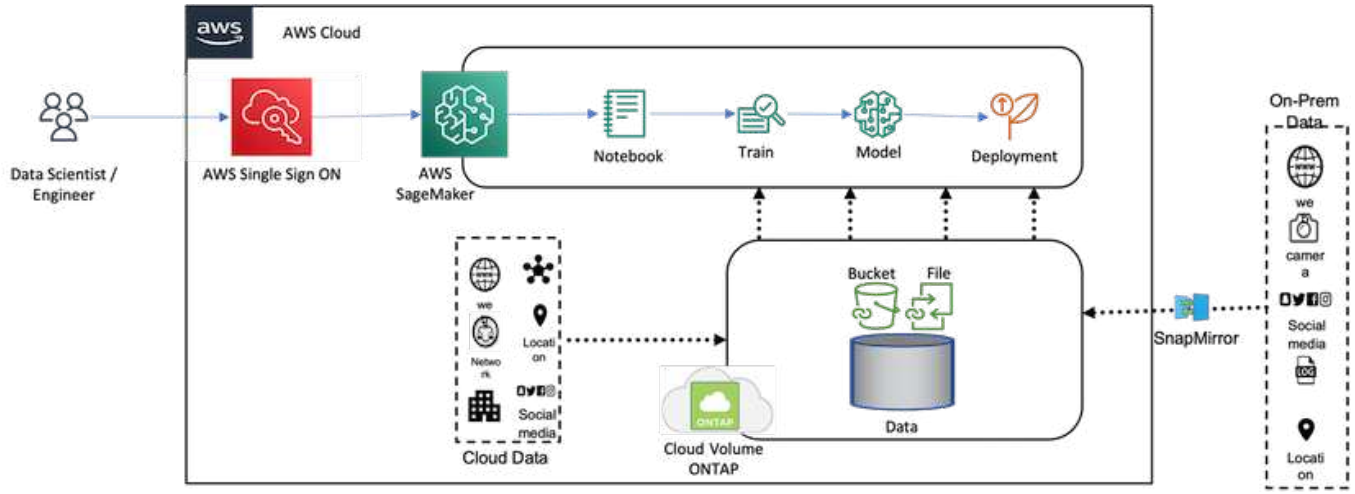
解決策テクノロジ

この解決策 では、次のテクノロジを使用します。

- * AWS SageMaker Notebook。*は、高品質のMLモデルを効率的に作成、トレーニング、導入するための機械学習機能を開発者やデータサイエンティストに提供します。
- * NetApp BlueXP *。オンプレミスとAWS、Azure、Google Cloudのストレージの検出、導入、運用が可能です。データ損失、サイバー脅威、計画外停止からデータを保護し、データストレージとインフラを最適化します。
- * NetApp Cloud Volumes ONTAP 。* AWS、Azure、Google CloudでNFS、SMB / CIFS、iSCSI、S3プロトコルを使用したエンタープライズクラスのストレージボリュームを提供し、クラウド内のデータに柔軟にアクセスして管理できるようにします。

ネットアップのCloud Volumes ONTAP は、MLデータを格納するためにBlueXPから開発されました。

次の図に、解決策の技術コンポーネントを示します。



ユースケースの概要

NFSとS3のデュアルプロトコルアクセスのユースケースとしては、機械学習とデータサイエンスの分野が考えられます。たとえば、データサイエンティストのチームがAWS SageMakerを使用した機械学習プロジェクトに取り組んでいるとします。このプロジェクトでは、NFS形式で保存されたデータにアクセスする必要があります。ただし、他のチームメンバーとコラボレーションしたり、S3を使用する他のアプリケーションと統合したりするために、データにS3バケット経由でアクセスして共有しなければならない場合もあります。

NetApp Cloud Volumes ONTAP を利用することで、チームはデータを単一の場所に保存し、NFSプロトコルとS3プロトコルの両方でデータにアクセスできるようになります。データサイエンティストはAWS SageMakerからNFS形式のデータに直接アクセスできますが、他のチームメンバーやアプリケーションはS3バケットを介して同じデータにアクセスできます。

このアプローチにより、ソフトウェアを追加したり、異なるストレージソリューション間でデータを移行したりすることなく、簡単かつ効率的にデータにアクセスして共有できます。また、ワークフローとチームメンバー間のコラボレーションがより合理化され、機械学習モデルの開発がより迅速かつ効果的になります。

データサイエンティストやその他のアプリケーション向けのデータの二重性

データはNFSで利用でき、AWS SageMakerからS3からアクセスできます。

テクノロジ要件

データ二重性のユースケースには、NetApp BlueXP、NetApp Cloud Volumes ONTAP 、AWS SageMakerノートブックが必要です。

ソフトウェア要件

次の表に、ユースケースの実装に必要なソフトウェアコンポーネントを示します。

ソフトウェア	数量
BlueXP	1.
NetApp Cloud Volumes ONTAP の略	1.
AWS SageMaker Notebook	1.

導入手順

data-duality解決策 を導入するには、次のタスクを実行します。

- BlueXPコネクタ
- NetApp Cloud Volumes ONTAP の略
- 機械学習のためのデータ
- AWS SageMaker
- Jupyter Notebooksによる検証済みの機械学習

BlueXPコネクタ

この検証ではAWSを使用しました。AzureやGoogle Cloudにも対応しています。AWSでBlueXPコネクタを作成するには、次の手順を実行します。

1. BlueXPでは、mcarl-marketplace-subscriptionに基づくクレデンシャルを使用しました。
2. 環境に適したリージョンを選択します（例：us-east-1 [N.[Virginia]）をクリックし、認証方法を選択します（例：[Assume Role]や[AWS Keys]）。この検証では、AWSキーを使用します。
3. コネクタの名前を指定し、ロールを作成します。
4. パブリックIPが必要かどうかに応じて、VPC、サブネット、キーペアなどのネットワークの詳細を指定します。
5. セキュリティグループの詳細（任意の場所やIP範囲の情報など、ソースタイプからのHTTP、HTTPS、SSHアクセスなど）を指定します。
6. BlueXPコネクタを確認して作成します。
7. AWSコンソールでBlueXP EC2インスタンスの状態がrunningであることを確認し、*[ネットワーク]*タブでIPアドレスを確認します。
8. BlueXPポータルからコネクタのユーザインターフェイスにログインするか、ブラウザからIPアドレスを使用してアクセスできます。

NetApp Cloud Volumes ONTAP の略

BlueXPでCloud Volumes ONTAP インスタンスを作成するには、次の手順を実行します。

1. 新しい作業環境を作成し、クラウドプロバイダを選択して、Cloud Volumes ONTAP インスタンスのタイプ（single-CVO、HA、Amazon FSxN for ONTAP など）を選択します。
2. Cloud Volumes ONTAP クラスタ名やクレデンシャルなどの詳細を入力します。この検証では、というCloud Volumes ONTAP インスタンスを作成しました `svm_sagemaker_cvo_sn1`。
3. Cloud Volumes ONTAP に必要なサービスを選択します。この検証では監視のみを選択したため、* Data Sense & Compliance と Backup to Cloud Services *を無効にしました。
4. [Location & Connectivity]*セクションで、AWSリージョン、VPC、サブネット、セキュリティグループ、SSH認証方式を選択します。 パスワードまたはキーペアのいずれかです。
5. 充電方法を選択します。この検証には* Professional *を使用しました。
6. POCとSmall Workloads 、**Database and Application Data Production Workloads *、Cost Effective DR *、Highest Performance Production Workloads** *などの構成済みパッケージを選択できます。この検証では、 POCと小規模ワークロード*を選択しました。

7. 特定のサイズ、許可するプロトコル、およびエクスポートオプションを指定してボリュームを作成します。この検証では、というボリュームを作成しました `vol1`。
8. プロファイルのディスクタイプと階層化ポリシーを選択します。この検証では、* Storage Efficiency *と*汎用SSD–動的パフォーマンス*を無効にしました。
9. 最後に、Cloud Volumes ONTAP インスタンスを確認して作成します。BlueXPでCloud Volumes ONTAP 作業環境が作成されるまで15~20分待ちます。
10. 二重プロトコルをイネーブルにするには、次のパラメータを設定します。デュアルプロトコル（NFS / S3）はONTAP 9からサポートされています。12.1以降。
    a. この検証では、というSVMを作成しました `svm_sagemaker_cvo_sn1` およびvolumeです `vol1`。
    b. SVMのプロトコルでNFSとS3がサポートされていることを確認します。サポートされていない場合は、サポートするようにSVMを変更します。

```
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. 必要に応じてCA証明書を作成してインストールします。
12. サービスデータポリシーを作成します。

```
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. アグリゲートの詳細を確認します。

```
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. ユーザとグループを作成します。

```
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::*>
```

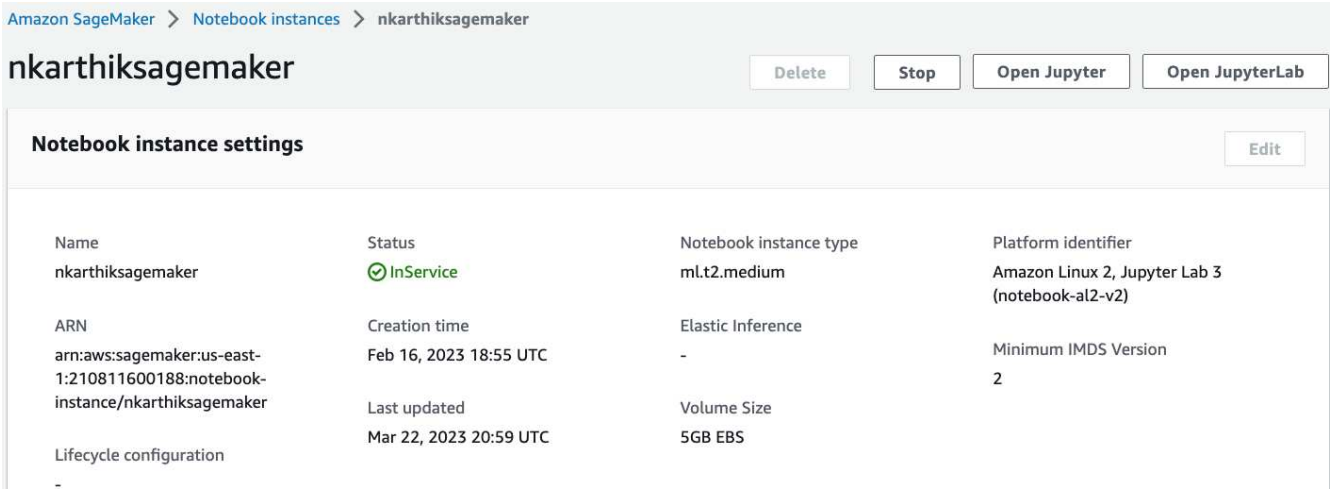
15. NFSボリューム上にバケットを作成します。

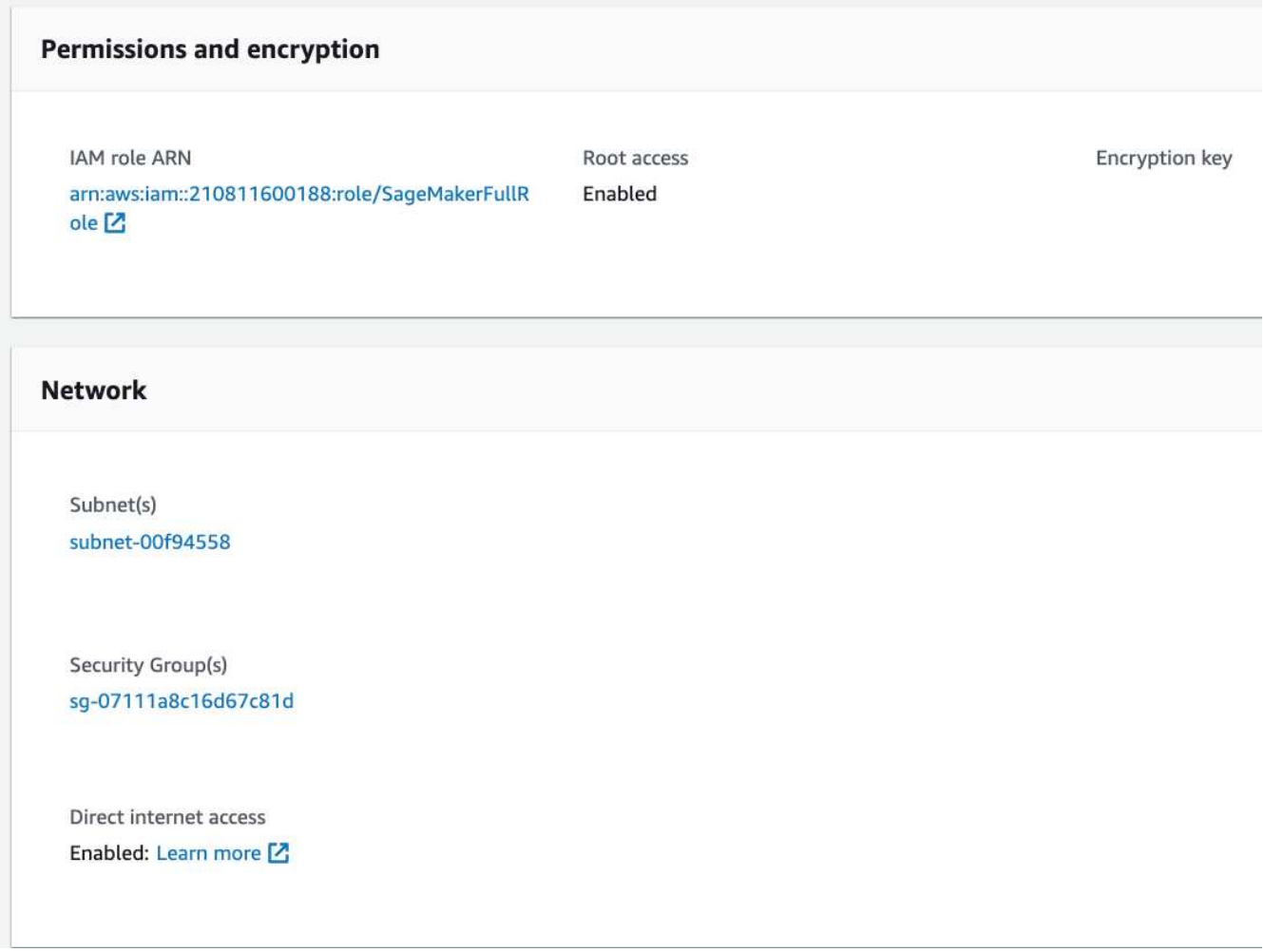
```
sagemaker_cvo_sn1::*> vserver object-store-server bucket create -bucket
ontapbucket1 -type nas -comment "" -vserver svm_sagemaker_cvo_sn1 -nas
-path /vol1
sagemaker_cvo_sn1::*> vserver object-store-server bucket show
Vserver     Bucket          Type     Volume            Size
Encryption Role    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

AWS SageMaker

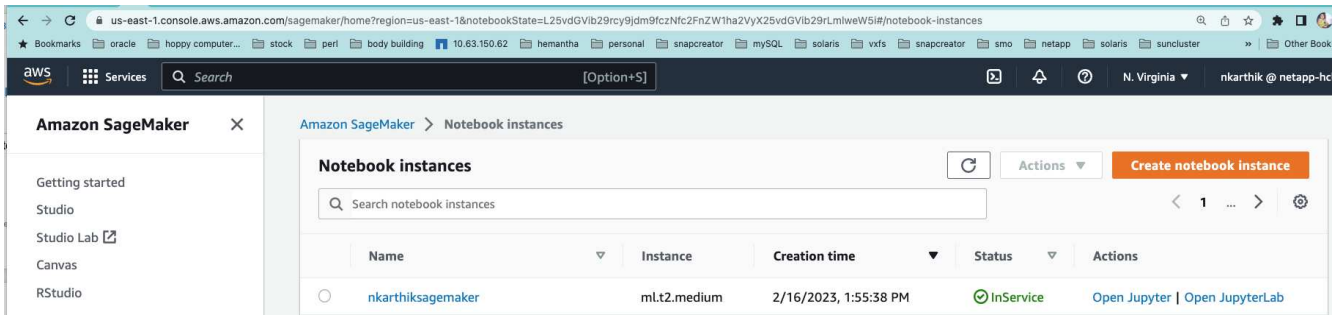
AWS SageMakerからAWS Notebookを作成するには、次の手順を実行します。

1. NotebookインスタンスをAmazonSageMakerFullAccess IAMポリシーを持っているか、またはAmazonSageMakerFullAccess権限を持つ既存のグループに属していることを確認します。この検証では、ユーザは既存のグループに属しています。
2. 次の情報を入力します。
   - ノートブックインスタンス名。
   - インスタンスタイプ。
   - プラットフォームID。
   - AmazonSageMakerFullAccess権限を持つIAMロールを選択します。
   - ルートアクセス–イネーブル。
   - Encryption key -カスタム暗号化なしを選択します。
   - 残りのデフォルトオプションはそのままにします。
3. この検証では、SageMakerインスタンスの詳細は次のとおりです。

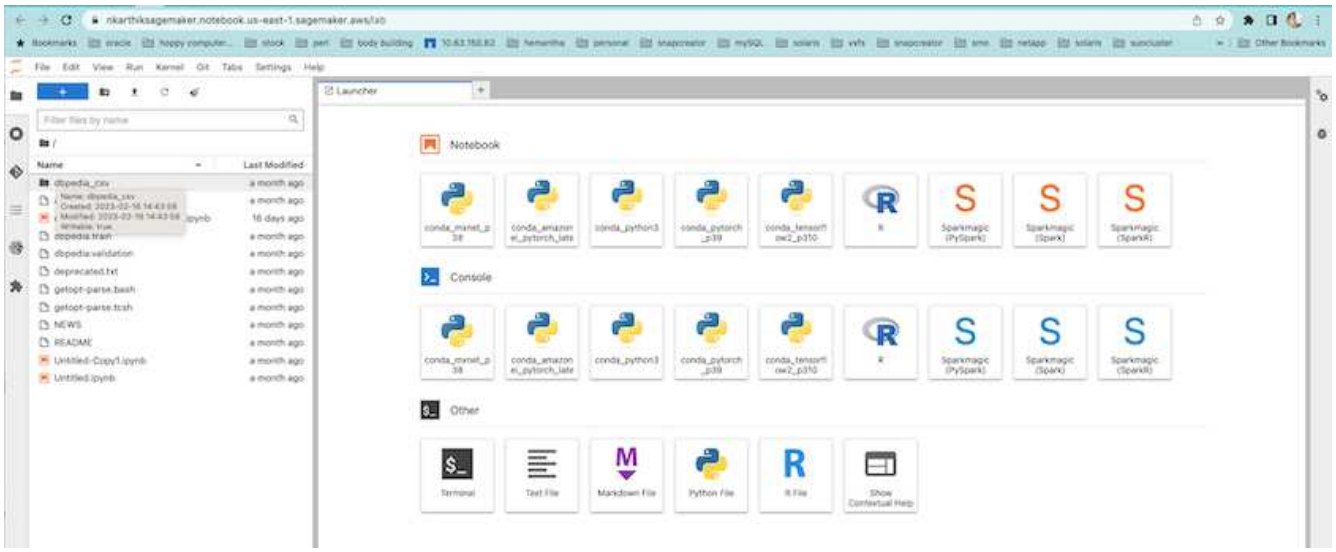




4. AWS Notebookを起動します。



5. Jupyterラボを開きます。



6. 端末にログインし、Cloud Volumes ONTAP ボリュームをマウントします。

```
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コマンドを使用して、Cloud Volumes ONTAP ボリュームに作成されたバケットを確認します。

```
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$
```

機械学習のためのデータ

この検証では、クラウドソーシングされたコミュニティの取り組みであるDBpediaのデータセットを使用して、さまざまなウィキメディアプロジェクトで作成された情報から構造化コンテンツを抽出しました。

1. DBpedia GitHubの場所からデータをダウンロードして抽出します。前のセクションで使用したのと同じターミナルを使用します。

```
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. AWS CLIを使用して、データをCloud Volumes ONTAP の場所にコピーし、S3バケットから確認します。

```
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. 基本的な検証を実行して、S3バケットで読み取り/書き込み機能が動作することを確認

```
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の機械学習を検証します

次の検証では、以下のSageMaker BlazingTextの例を使用して、テキスト分類によってモデルの機械学習のビルド、トレーニング、およびデプロイを行います。

1. boto3パッケージとSageMakerパッケージをインストールします。

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

出力：

```
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. 次の手順では、データを使用します (`dbpedia_csv`）はs3バケットからダウンロードされます `ontapbucket1` 機械学習で使用されるJupyter Notebookインスタンスにコピーします。

```
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. 次のコードは、整数インデックスからクラスラベルへのマッピングを作成します。このマッピングは、推論時に実際のクラス名を取得するために使用されます。

```
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()
```

出力には、内のファイルとフォルダが一覧表示されます `ontapbucket1` AWS SageMaker機械学習検証のデータとして使用されるバケット。

```
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. データ前処理フェーズを開始して、トレーニングデータをスペース区切りのトークン化されたテキスト形式に前処理します。この形式は、BlazingTextアルゴリズムとnltkライブラリによって使用され、DBPediaデータセットから入力文をトークン化します。nltkトークナイザーおよびその他のライブラリをダウンロードします。。 `transform_instance` 並列で各データインスタンスに適用するには、Pythonマルチプロセッシングモジュールを使用します。

```
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)
```

出力：

```
[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. SageMakerでトレーニングジョブを実行するために使用できるように、フォーマット済みデータセットとトレーニングデータセットをS3にアップロードします。次に、Python SDKを使用して、バケットとプレフィックスの場所に2つのファイルをアップロードします。

```
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)
```

出力：

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

6. アーティファクトがアルゴリズムのトレーニングジョブの出力になるように、モデルアーティファクトがロードされるS3に出力場所を設定します。を作成します `sageMaker.estimator.Estimator` トレーニングジョブを起動するオブジェクト。

```
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))
```

出力：

```
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. SageMakerを定義します `Estrimator` リソース構成とハイパーパラメータを使用して、c4.4xlargeインスタンスの監視モードを使用してDBPediaデータセットでテキスト分類をトレーニングします。

```
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. データチャネルとアルゴリズム間のハンドシェイクを準備します。これを行うには、を作成します `sagemaker.session.s3_input` データチャネルからオブジェクトを取得し、アルゴリズムが使用するためにディクショナリに保持します。

```
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. ジョブが完了すると、[Job Complete]メッセージが表示されます。トレーニング済みモデルは、としてセットアップされたS3バケットにあります `output_path` 推定量の中で。

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

出力：

```
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. トレーニングが完了したら、トレーニング済みモデルをAmazon SageMakerリアルタイムホストエンドポイントとしてデプロイして予測を行います。

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

出力：

```
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. デフォルトでは、モデルは最も高い確率で1つの予測を返します。上部を取得します `k` 予測、設定 `k` を設定ファイルに保存します。

```
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. ノートブックを閉じる前にエンドポイントを削除してください。

```
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
```

まとめ

この検証に基づいて、データサイエンティストとエンジニアは、AWS SageMaker Jupyter Notebookから、NetApp Cloud Volumes ONTAP のS3バケットを介してNFSデータにアクセスできます。このアプローチでは、追加のソフトウェアを必要とせずに、NFSとS3の両方から同じデータに簡単にアクセスして共有できます。

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- SageMaker BlazingTextを使用したテキスト分類

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

- S3 オブジェクトストレージでの ONTAP バージョンのサポート

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

NetApp NFSストレージを使用したApache Kafkaワークロード

TR-4947：『Apache Kafka workload with NetApp NFS storage - Functional Validation and performance』

ネットアップShantanu Chakole、Karthikeyan Nagalingam、Joe Scott

Kafkaは、大量のメッセージデータを受け入れることができる堅牢なキューを備えた分散型パブリッシュサブスクライブメッセージングシステムです。Kafkaを使用すると、アプリケーションは非常に高速な方法でトピックへのデータの書き込みと読み取りを行うことができます。フォールトトレランスと拡張性を備えているため、多くのデータストリームを迅速に取り込み、移動するための信頼性の高い方法として、ビッグデータ分野でよく使用されています。ユースケースには、ストリーム処理、Webサイトアクティビティの追跡、指標の収集と監視、ログの集約、リアルタイム分析などがあります。

NFSでの通常のKafka操作は問題なく機能しますが "変な名前だな" NFSで実行されているKafkaクラスタのサイズ変更中または再パーティション中に、問題 がアプリケーションをクラッシュさせます。これは、負荷分散やメンテナンスのためにKafkaクラスタのサイズを変更または再パーティションする必要があるため、重要な問題 です。詳細については、こちらを参照してください "こちらをご覧ください"。

このドキュメントでは、次の項目について説明します。

- silly-renameの問題と解決策 の検証
- CPU使用率を削減してI/O待機時間を短縮します
- Kafkaブローカーのリカバリ時間の短縮
- クラウドとオンプレミスでのパフォーマンス

Kafkaワークロードに**NFS**ストレージを使用する理由

本番アプリケーションのKafkaワークロードでは、アプリケーション間で大量のデータをストリーミングすることができます。このデータは、Kafkaクラスタ内のKafkaブローカーノードに保持され、格納されます。Kafkaは可用性と並列処理でも知られています。Kafkaは、トピックをパーティションに分割し、それらのパーティションをクラスタ全体に複製することで実現します。これは、最終的には、Kafkaクラスタを通過する膨大な量のデータのサイズが一般的に倍増されることを意味します。NFSを使用すると、ブローカーの数が非常に迅速かつ簡単に変更されるため、データのリバランシングが可能になります。大規模な環境では、ブローカーの数が変更されたときにDAS間でデータをリバランシングするのは非常に時間がかかり、ほとんどのKafka環境ではブローカーの数が頻繁に変更されます。

その他にも、次のようなメリットがあります。

- 成熟度。 NFSは成熟したプロトコルであり、実装、セキュリティ保護、使用のほとんどの側面がよく理解されています。
- オープン。 NFSはオープンプロトコルであり、その継続的な開発はインターネット仕様で無料でオープンなネットワークプロトコルとして文書化されています。
- コスト効率に優れています。 NFSは、ネットワークファイル共有用の低コストの解決策 であり、既存のネットワークインフラストラクチャを使用しているため、セットアップが簡単です。
- 一元管理 NFSの一元管理により'個のユーザー・システムでソフトウェアやディスク・スペースを追加する必要がなくなります
- 分散。 NFSを分散ファイルシステムとして使用できるため、リムーバブルメディアストレージデバイスの必要性が軽減されます。

ネットアップが**Kafka**ワークロードに最適な理由

ネットアップのNFS実装はプロトコルのゴールドスタンダードとみなされ、無数のエンタープライズNAS環境で使用されています。ネットアップの信頼性に加えて、次のようなメリットもあります。

- 信頼性と効率性
- 拡張性とパフォーマンス
- ハイアベイラビリティ（NetApp ONTAP クラスタ内のHAパートナー）
- データ保護
  - *ディザスタリカバリ（NetApp SnapMirror）。*サイトがダウンしたか、別のサイトから始めて中断したところから作業を続行したい。
  - ストレージシステムの管理性（NetApp OnCommand を使用した管理）。
  - *ロードバランシング。*クラスタでは、異なるノードでホストされているデータLIFから異なるボリュームにアクセスできます。
  - ノンストップオペレーション。 LIFやボリュームの移動はNFSクライアントに対して透過的です。

NetApp解決策 for silly rename問題 for NFS to Kafka workloads.

Kafkaは、基盤となるファイルシステムがPOSIXに準拠していることを前提に構築されています。たとえば、XFSやext4です。Kafkaリソースのリバランシングは、アプリケーションがまだファイルを使用している間にファイルを削除します。POSIX準拠のファイルシステムでは、リンク解除を続行できます。ただし、ファイルへのすべての参照が失われた後にのみ、ファイルが削除されます。基盤となるファイルシステムがネットワークに接続されている場合、NFSクライアントはunlink呼び出しを代行受信してワークフローを管理します。リンク解除されているファイルでは保留中のオープンがあるため、NFSクライアントはNFSサーバに名前変更要求を送信し、リンク解除されたファイルの最後のクローズ時に、名前変更されたファイルに対して削除操作を実行します。この動作は、一般にNFSのsilly renameと呼ばれ、NFSクライアントによってオーケストレーションされます。

NFSv3サーバのストレージを使用するKafkaブローカーでは、この動作が原因で問題が発生します。ただ

し、NFSv4.xプロトコルには、リンクされていない開いたファイルをサーバが処理できるようにすることで、この問題 に対処する機能があります。このオプション機能をサポートするNFSサーバは、ファイルを開いたときに所有権機能をNFSクライアントに通知します。その後、オープン保留中の状態があるとNFSクライアントはリンク解除の管理を中止し、サーバがフローを管理できるようにします。NFSv4の仕様には実装に関するガイドラインが規定されていますが、これまでこのオプション機能をサポートする既知のNFSサーバの実装はありませんでした。

NFSサーバとNFSクライアントで、silly rename問題 に対処するには、次の変更が必要です。

- * NFSクライアント（Linux）への変更。*ファイルを開くときに、NFSサーバは、開いているファイルのリンクを解除する機能を示すフラグを返します。NFSクライアント側の変更により、フラグが指定された状態でNFSサーバがリンク解除を処理できるようになります。ネットアップでは、これらの変更でオープンソースのLinux NFSクライアントを更新しました。更新されたNFSクライアントがRHEL8.7およびRHEL9.1で一般提供されるようになりました。
- * NFSサーバーへの変更。* NFSサーバーはオープンを追跡します。既存の開いているファイルのリンク解除は、POSIXセマンティクスと一致するようにサーバーによって管理されるようになりました。最後に開いていたファイルが閉じると、NFSサーバによって実際のファイルの削除が開始されるため、名前の変更が不要になります。この機能は、ONTAP NFSサーバの最新リリースであるONTAP 9.12.1で実装されています。

NFSクライアントとNFSサーバに対する上記の変更により、Kafkaはネットワーク接続型NFSストレージのすべてのメリットを安全に享受できます。

機能検証**- silly rename fix**

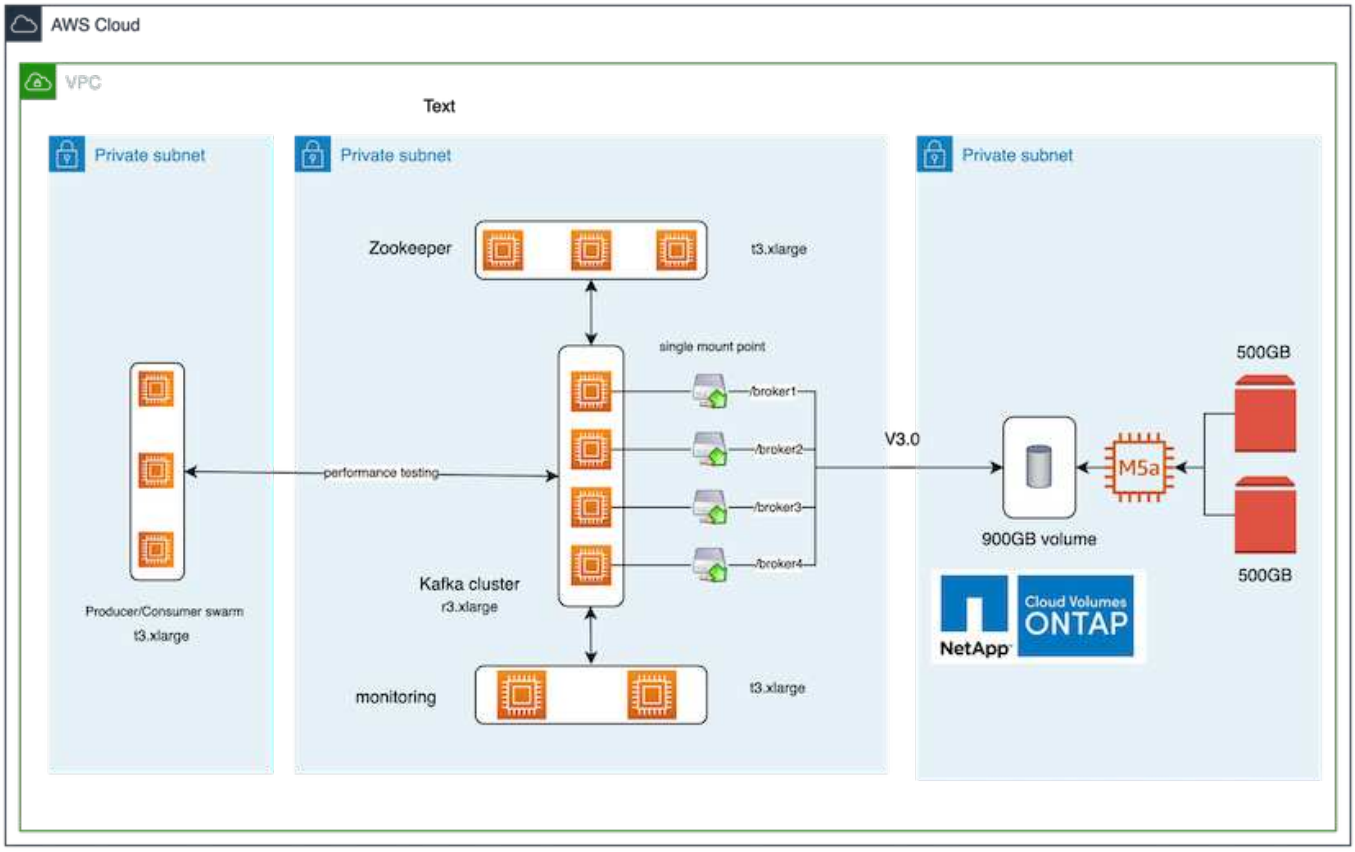
機能検証の結果、ストレージ用にNFSv3マウントを使用したKafkaクラスタではパーティションの再配置などのKafka処理を実行できませんが、修正後にNFSv4にマウントされた別のクラスタでは、システムを停止することなく同じ処理を実行できることがわかりました。

検証のセットアップ

セットアップはAWSで実行されます。次の表に、この検証で使用したプラットフォームコンポーネントと環境構成を示します。

プラットフォームコンポーネント	環境の構成
Confluent Platformバージョン7.2.1	• 3 x動物飼育係–T3.xlarge • ブローカーサーバ×4–r3.xlarge • Grafana–T3.xlarge×1 • 1 xコントロールセンター–T3.xlarge • 3 xプロデューサー/コンシューマー
すべてのノード上のオペレーティングシステム	RHEL8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス–M5.2xLarge

次の図は、この解決策 のアーキテクチャ構成を示しています。



アーキテクチャの流れ

- コンピューティング。 4ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。
- 監視。 Prometheus-Grafanaの組み合わせには2つのノードを使用しました。
- *ワークロード。*ワークロードの生成には、このKafkaクラスタとの間でやり取り可能な3ノードクラスタを使用しました。
- *ストレージ。*シングルノードのNetApp Cloud Volumes ONTAP インスタンスを使用し、2つの500GB gp2 AWS-EBSボリュームをインスタンスに接続しました。これらのボリュームは、LIFを介して単一のNFSv4.1ボリュームとしてKafkaクラスタに公開されました。

Kafkaのデフォルトプロパティは、すべてのサーバーに対して選択されています。動物園の群れについても同じことが行われました。

テストの方法論

1. 更新 `-is-preserve-unlink-enabled true` 次のようにKafkaボリュームに追加します。

```
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. 似たような2つのKafkaクラスタが作成されましたが、次の違いがあります。
   - *クラスタ1。*本番環境対応のONTAP バージョン9.12.1を実行しているバックエンドNFS v4.1サーバは、NetApp CVOインスタンスによってホストされていました。ブローカーにRHEL 8.7 / RHEL 9.1をインストールしました。
   - *クラスタ2。*バックエンドNFSサーバは、手動で作成した汎用Linux NFSv3サーバです。
3. 両方のKafkaクラスタでデモトピックを作成しました。

   クラスタ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:
```

   クラスタ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. 両方のクラスタについて、新しく作成されたこれらのトピックにデータがロードされました。これには、デフォルトのKafkaパッケージに含まれているproducer-perf-testツールキットを使用しました。

```
./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. telnetを使用して、各クラスタのbroker-1の健全性チェックを実行しました。
   - Telnet `172.30.0.160 9092`
   - Telnet `172.30.0.198 9092`

     次のスクリーンショットには、両方のクラスタのブローカーの健全性チェックが成功したことが示されています。

```
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ストレージボリュームを使用するKafkaクラスタがクラッシュする障害状態をトリガーするために、両方のクラスタでパーティションの再割り当てプロセスを開始しました。パーティションの再割り当てはを使用して実行されました `kafka-reassign-partitions.sh`。詳細なプロセスは次のとおりです。
   a. Kafkaクラスタでトピックのパーティションを再割り当てするために、提案された再割り当て構成JSONを生成しました（これは両方のクラスタで実行しました）。

      ```
      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. 生成された再割り当てJSONがに保存されました `/tmp/reassignment- file.json`。
   c. 実際のパーティション再割り当てプロセスは、次のコマンドによって開始されました。

      ```
      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. 再割り当てが完了してから数分後、ブローカーで別の健全性チェックを実行したところ、NFSv3ストレージボリュームを使用するクラスタで誤った名前の問題 が発生してクラッシュしたことがわかりました。一方、クラスタ1では、NetApp ONTAP NFSv4.1ストレージボリュームを使用しています。この問題は修正され、システムが停止することなく継続的に処理

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

Connection closed by foreign host.
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.198 9092
Trying 172.30.0.198...
telnet: connect to address 172.30.0.198: Connection refused
telnet: Unable to connect to remote host
```

- cluster1-Broker-1がアクティブです。
- cluster2-broker-1は停止しています。

8. Kafkaログディレクトリを確認したところ、修正済みのNetApp ONTAP NFSv4.1ストレージボリュームを使用しているクラスタ1ではパーティションがクリーンに割り当てられているのに対し、汎用のNFSv3ストレージを使用しているクラスタ2では異常な名前変更の問題が原因でクラッシュが発生したことがわかりました。次の図は、クラスタ2のパーティションのリバランシングを示しています。これにより、NFSv3ストレージで問題 名が誤って変更されました。

```
/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
```

次の図は、NetApp NFSv4.1ストレージを使用したクラスタ1のパーティションのクリーンなリバランシングを示しています。

```
/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
```

ネットアップの**NFS**が**Kafka**ワークロードに最適な理由

Kafkaを使用するNFSストレージの問題 名を変更するための解決策 が追加されたので、KafkaワークロードにNetApp ONTAP ストレージを活用する堅牢な環境を構築できます。これにより、運用オーバーヘッドが大幅に削減されるだけでなく、Kafkaクラスタに次のメリットがもたらされます。

- * KafkaブローカーのCPU利用率を削減*分離されたネットアップONTAP ストレージを使用すると、ディスクI/O処理がブローカーから分離されるため、CPUフットプリントが削減されます。
- *ブローカーのリカバリ時間が短縮されます。*分離型のNetApp ONTAP ストレージはKafkaブローカーのノード間で共有されるため、データを再構築することなく、従来のKafka環境と比較して、任意の時点で新しいコンピューティングインスタンスを使用して不良ブローカーに置き換えることができます。
- * Storage Efficiency。*アプリケーションのストレージレイヤがNetApp ONTAP でプロビジョニングされるようになったため、インラインデータ圧縮、重複排除、コンパクションなど、ONTAP に備わっているStorage Efficiencyのメリットをすべて活用できます。

これらの利点は、このセクションで詳しく説明するテストケースでテストおよび検証されました。

Kafkaブローカーの**CPU**使用率の低下

技術仕様は同一だがストレージ技術が異なる2つの精子Kafkaクラスタで同様のワークロードを実行した場合、全体的なCPU利用率がDASに比べて低いことがわかりました。KafkaクラスタがONTAP ストレージを使用している場合、全体的なCPU利用率は低くなるだけでなく、CPU利用率の増加はDASベースのKafkaクラスタよりも緩やかな勾配を示しています。

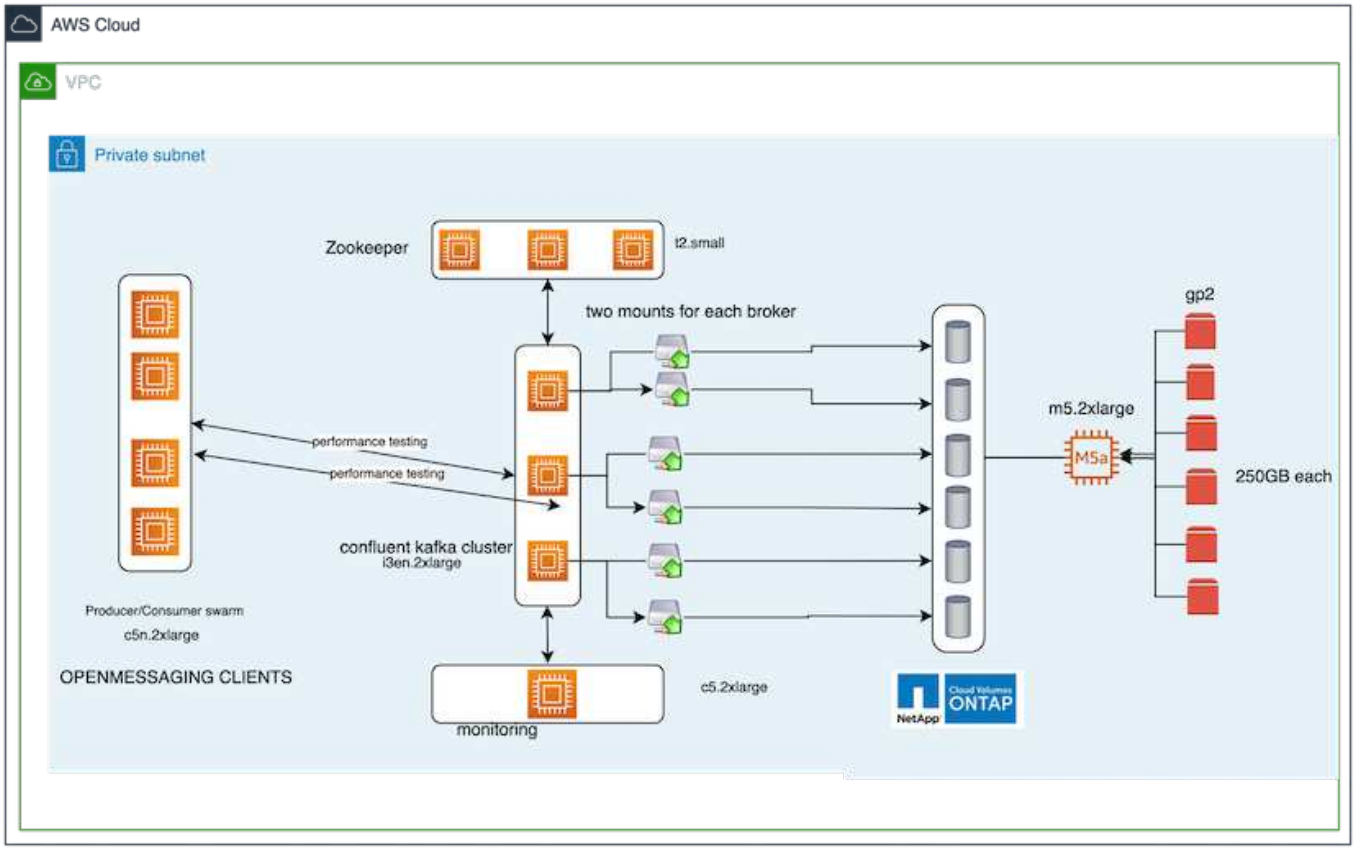
アーキテクチャのセットアップ

次の表に、CPU利用率の低下を実証するために使用する環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3ベンチマークツール: OpenMessaging	• 3 x動物飼育係–T2.small • ブローカーサーバ×3–i3en.2xlarge • Grafana–c5n.2xlarge×1 • 4 xプロデューサー/コンシューマー-- c5n.2xlarge
すべてのノード上のオペレーティングシステム	RHEL 8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス–M5.2xLarge

ベンチマークツール

このテストケースで使用されているベンチマークツールはです "OpenMessagingの略" フレームワーク：OpenMessagingは、ベンダーに依存せず、言語に依存しません。金融、eコマース、IoT、ビッグデータに関する業界ガイドラインを提供し、異種システムやプラットフォーム間でメッセージングやストリーミングアプリケーションを開発するのに役立ちます。次の図は、OpenMessagingクライアントとKafkaクラスタの相互作用を示しています。



- コンピューティング。 3ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。各ブローカーには、専用のLIFを介してNetApp CVOインスタンス上の単一のボリュームへのNFSv4.1マウントポイントが2つありました。
- 監視。 Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成については、このKafkaクラスタとの間でデータを生成したり消費したりできる3ノードクラスタを別途用意しています。

- *ストレージ。*シングルノードのNetApp Cloud Volumes ONTAP インスタンスを使用し、250GB gp2 AWS-EBSボリュームを6個マウントしました。その後、これらのボリュームは、専用のLIFを介して6つのNFSv4.1ボリュームとしてKafkaクラスタに公開されました。
- *設定。*このテストケースで設定可能な2つの要素は、KafkaブローカーとOpenMessagingワークロードでした。
  - ブローカー設定 Kafkaブローカーには以下の仕様が選択されています。以下に示すように、すべての測定値にレプリケーション係数3を使用しました。

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

- * OpenMessagingベンチマーク（OMB）のワークロード構成。*次の仕様が提供されました。以下で強調されている目標生産者率を指定しました。

```
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
```

テストの方法論

1. 2つの類似したクラスタが作成され、それぞれに独自のベンチマーククラスタ群があります。
   - クラスタ**1**。 NFSベースのKafkaクラスタ。

- クラスタ**2**。 DASベースのKafkaクラスタ。

2. OpenMessagingコマンドを使用すると、各クラスタで同様のワークロードがトリガーされます。

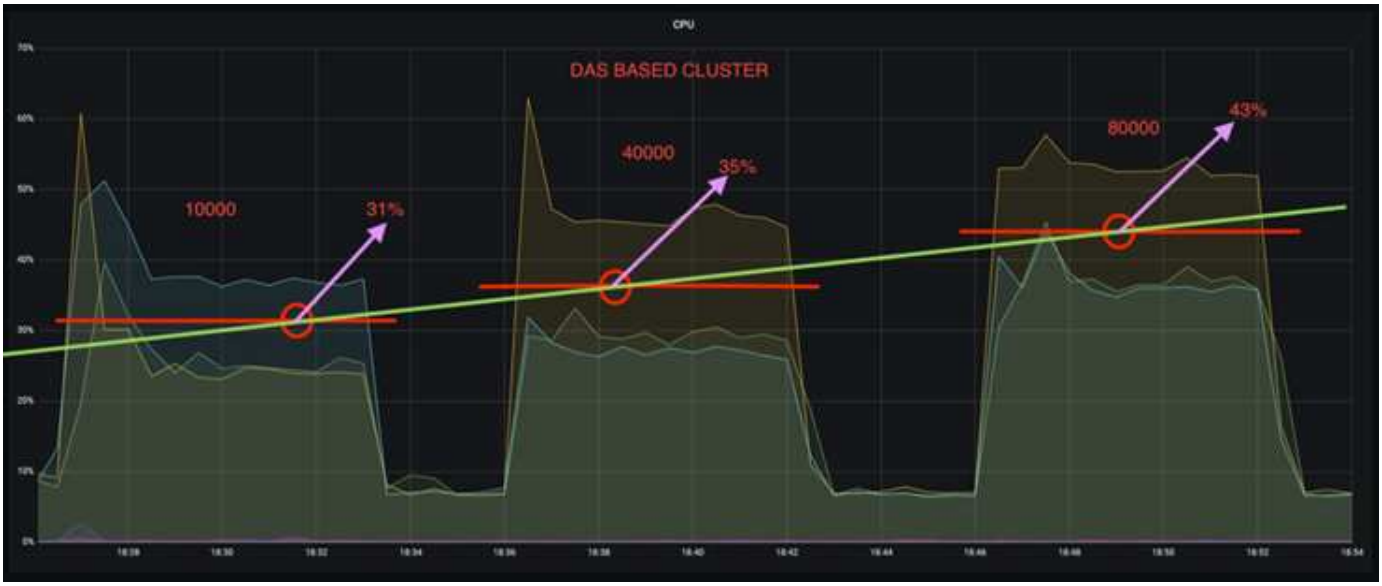
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

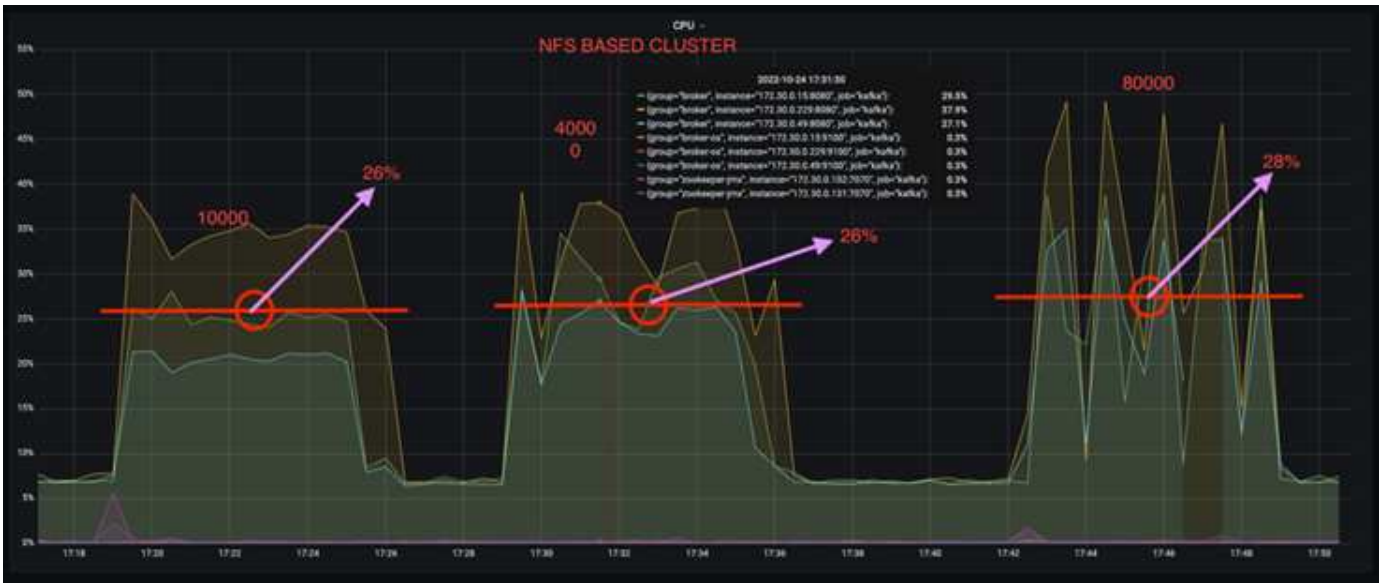
3. プロデュースレートの設定は4回の繰り返しで増加し、CPU使用率はGrafanaで記録されました。生産率は次のレベルに設定されました。
   - 1万だ
   - 40,000
   - 8万だ
   - 10万だ

観察

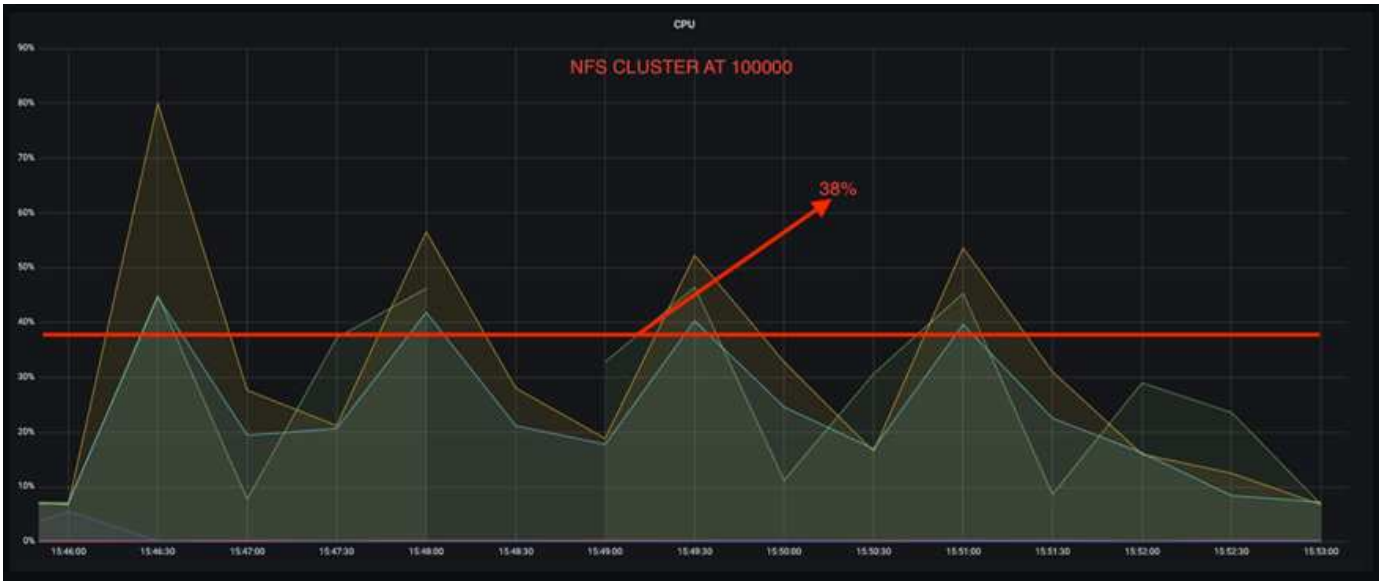
KafkaでNetApp NFSストレージを使用する主なメリットは2つあります。

- * CPU使用率をほぼ3分の1に削減できます。*同様のワークロードでの全体的なCPU使用率は、DAS SSDと比較してNFSでは低く、削減率は低い場合は5%、高い場合は32%です。
- ※プロデュース率が高い場合、CPU使用率のドリフトが3倍に減少※プロデュース率の上昇に伴い、CPU使用率の上昇は予想通り上昇しました。しかし、DASを使用するKafkaブローカーのCPU使用率は、低い生産率では31%から高い生産率では70%に上昇し、39%の増加となりました。一方、NFSストレージバックエンドでは、CPU利用率が26%から38%に上昇し、12%も上昇しました。





また、メッセージ数が100、000の場合、DASのCPU利用率はNFSクラスタよりも高くなります。



ブローカーの迅速なリカバリ

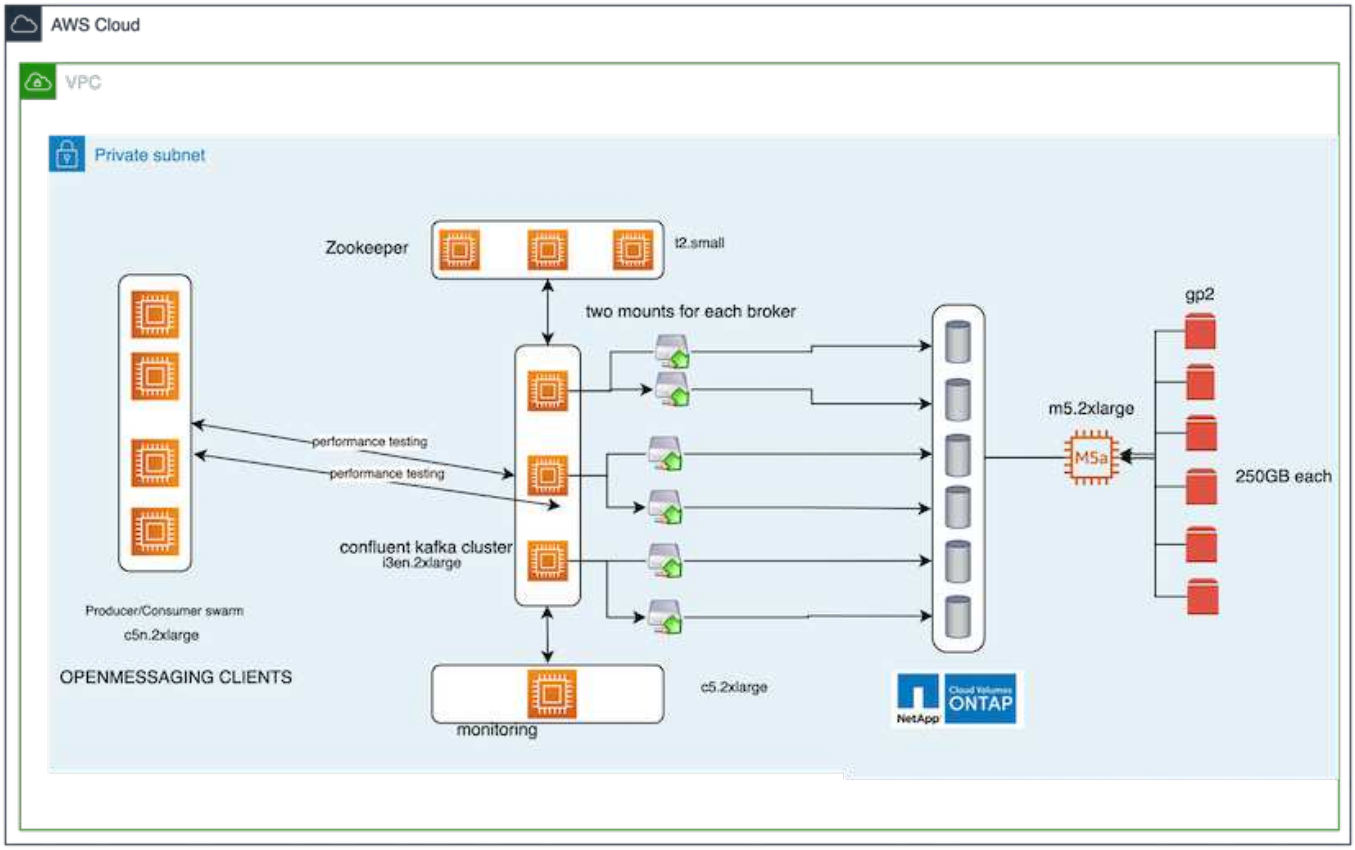
ネットアップの共有NFSストレージを使用すると、Kafkaブローカーのリカバリ時間が短縮されることがわかりました。Kafkaクラスタでブローカーがクラッシュした場合、このブローカーは同じブローカーIDを持つ正常なブローカーに置き換えることができます。このテストケースを実行したところ、DASベースのKafkaクラスタでは、新しく追加された正常なブローカーにデータが再構築されるため、時間がかかることがわかりました。NetApp NFSベースのKafkaクラスタの場合、交換後のブローカーは以前のログディレクトリから引き続きデータを読み取り、はるかに高速にリカバリします。

アーキテクチャのセットアップ

次の表に、NASを使用するKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	• 3 x動物飼育係–T2.small • ブローカーサーバ×3–i3en.2xlarge • Grafana–c5n.2xlarge×1 • 4 x producer/consumer — c5n.2xlarge • 1 xバックアップKafkaノード–i3en.2xlarge
すべてのノード上のオペレーティングシステム	RHEL8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス–M5.2xLarge

次の図は、NASベースのKafkaクラスタのアーキテクチャを示しています。



- コンピューティング。 3ノードのZookeeperアンサンブルを専用サーバー上で実行する3ノードのKafkaクラスタ。各ブローカーには、専用のLIFを介してNetApp CVOインスタンス上の単一のボリュームへのNFSマウントポイントが2つあります。
- 監視。 PrometheusとGrafanaの組み合わせでは2ノード。ワークロードの生成には、このKafkaクラスタを生成して使用できる3ノードクラスタを別 々 に使用します。
- *ストレージ。*シングルノードのNetApp Cloud Volumes ONTAP インスタンス。250GB gp2 AWS-EBSボリュームが6個マウントされています。これらのボリュームは、専用のLIFを介して6つのNFSボリュームとしてKafkaクラスタに提供されます。
- *ブローカーの設定*このテストケースで設定可能な要素の1つはKafkaブローカーです。Kafkaブローカーのために以下の仕様が選択されました。。 `replica.lag.time.mx.ms` は、特定のノードがISRリストから削除される速度を決定するため、高い値に設定されます。不良ノードと正常ノードを切り替える場合、そのブローカーIDがISRリストから除外されないようにします。

```
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
```

テストの方法論

1. 同様の2つのクラスタが作成されました。
   - EC2ベースのコンフルエントクラスタ。
   - NetApp NFSベースのコンフルエントクラスタ。
2. 1つのスタンバイKafkaノードが、元のKafkaクラスタのノードと同じ構成で作成されました。
3. 各クラスタでサンプルトピックを作成し、各ブローカーに約110GBのデータが読み込まれました。
   - * EC2ベースのクラスタ。* Kafkaブローカーのデータディレクトリがにマッピングされています `/mnt/data-2` （次の図では、cluster1のBroker-1（左側のターミナル））。
   - * NetApp NFSベースのクラスタ。* KafkaブローカーのデータディレクトリがNFSポイントにマウントされている `/mnt/data` （次の図では、cluster2のBroker-1（右側の端末））。

4. 各クラスタで、Broker-1が終了し、ブローカーのリカバリプロセスが失敗しました。
5. ブローカーが終了した後、ブローカーのIPアドレスがセカンダリIPとしてスタンバイブローカーに割り当てられました。これは、Kafkaクラスタ内のブローカーが次のように識別されるために必要でした。
    - * IPアドレス。*障害が発生したブローカーのIPをスタンバイブローカーに再割り当てすることによって割り当てられます。
    - *ブローカーID。*これはスタンバイブローカーで設定されました `server.properties`。
6. IP割り当て時に、スタンバイブローカーでKafkaサービスが開始されました。
7. しばらくすると、サーバログがプルされ、クラスタ内の交換用ノードでデータを構築するのにかかった時間が確認されました。

観察

Kafkaブローカーの回復はほぼ9倍速くなりました。NetApp NFS共有ストレージを使用すると、KafkaクラスタでDAS SSDを使用する場合と比較して、障害が発生したブローカーノードのリカバリにかかる時間が大幅に短縮されることがわかりました。1TBのトピックデータの場合、DASベースのクラスタのリカバリ時間は48分でしたが、NetApp-NFSベースのKafkaクラスタのリカバリ時間は5分未満でした。

EC2ベースのクラスタで110GBのデータを新しいブローカーノードにリビルドするのに10分かかったのに対し、NFSベースのクラスタでは3分でリカバリが完了しました。また、ログでは、EC2のパーティションのコンシューマオフセットが0であり、NFSクラスタではコンシューマオフセットが前のブローカーから取得されていることがわかりました。

```
[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ベースのクラスタ

1. バックアップノードは08：55：53、730に開始されました。

```
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. データの再構築プロセスは09:05:24,860に終了しました。110GBのデータの処理には約10分かかります。

```
[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ベースのクラスタ

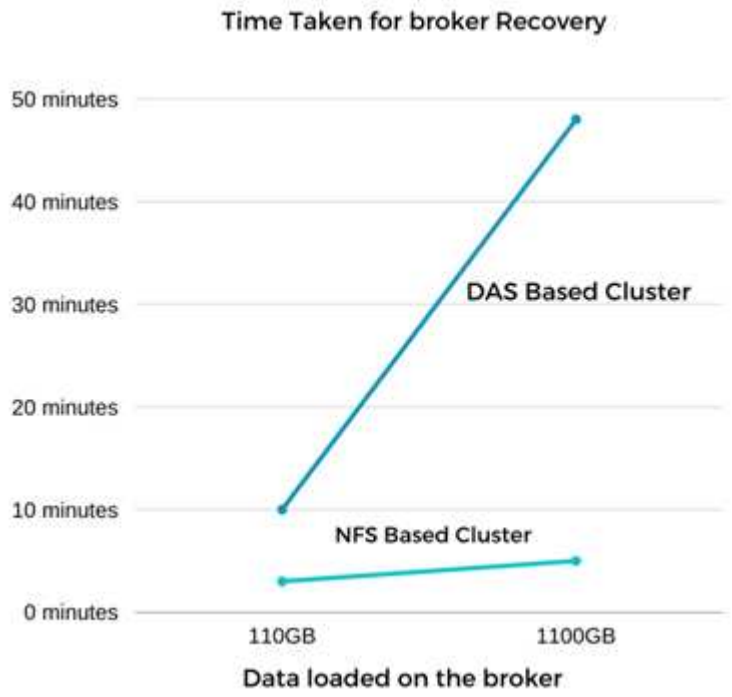
1. バックアップノードは09：39：17、213に開始されました。開始ログエントリは以下のように強調表示されます。

```
[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. データの再構築プロセスは09:42:29,115に終了しました。110GBのデータの処理には約3分かかります。

```
[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)
```

このテストを、約1TBのデータを含むブローカーに対して繰り返しました。DASでは約48分、NFSでは約3分かかりました。結果を次のグラフに示します。



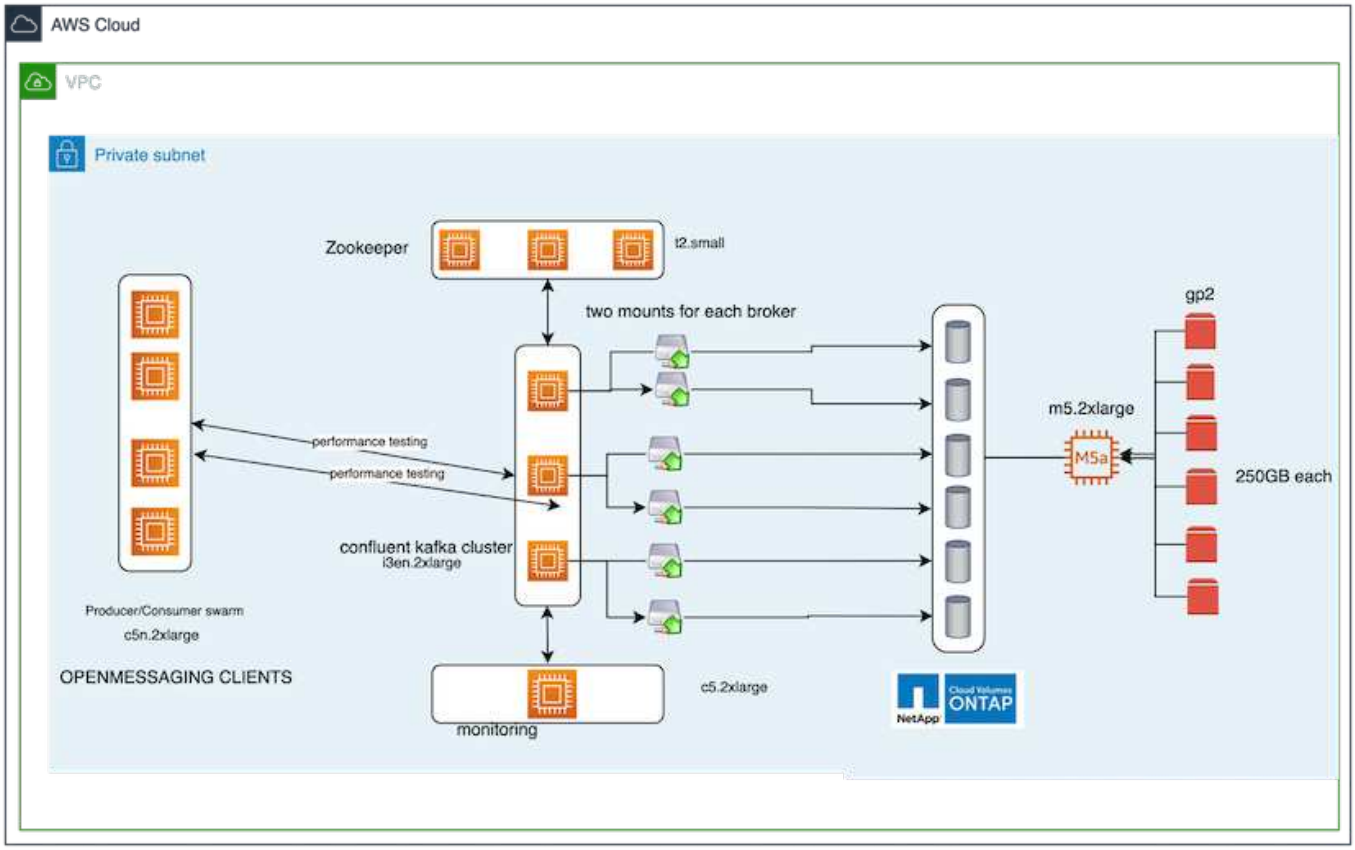
ストレージ効率

KafkaクラスタのストレージレイヤはNetApp ONTAP を介してプロビジョニングされていたため、ONTAP のすべてのStorage Efficiency機能を利用できました。このテストでは、Cloud Volumes ONTAP でNFSストレージをプロビジョニングしたKafkaクラスタで大量のデータを生成しました。ONTAP 機能により、スペースが大幅に削減されたことがわかりました。

次の表に、NASを使用するKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	• 3 x動物飼育係–T2.small • ブローカーサーバ×3–i3en.2xlarge • Grafana–c5n.2xlarge×1 • 4 x producer/consumer — c5n.2xlarge *
すべてのノード上のオペレーティングシステム	RHEL8.7以降
NetApp Cloud Volumes ONTAP インスタンス	シングルノードインスタンス–M5.2xLarge

次の図は、NASベースのKafkaクラスタのアーキテクチャを示しています。



- コンピューティング。 3ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。各ブローカーには、専用のLIFを介してNetApp CVOインスタンス上の単一のボリュームへのNFSマウントポイントが2つありました。
- 監視。 Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成には、独立した3ノードクラスタを使用し、このKafkaクラスタを生成して使用しました。
- *ストレージ。*シングルノードのNetApp Cloud Volumes ONTAP インスタンスを使用し、250GB gp2 AWS-EBSボリュームを6個マウントしました。その後、これらのボリュームは、専用のLIFを介して6つのNFSボリュームとしてKafkaクラスタに公開されました。

- *構成*このテストケースの構成要素はKafkaブローカーです。

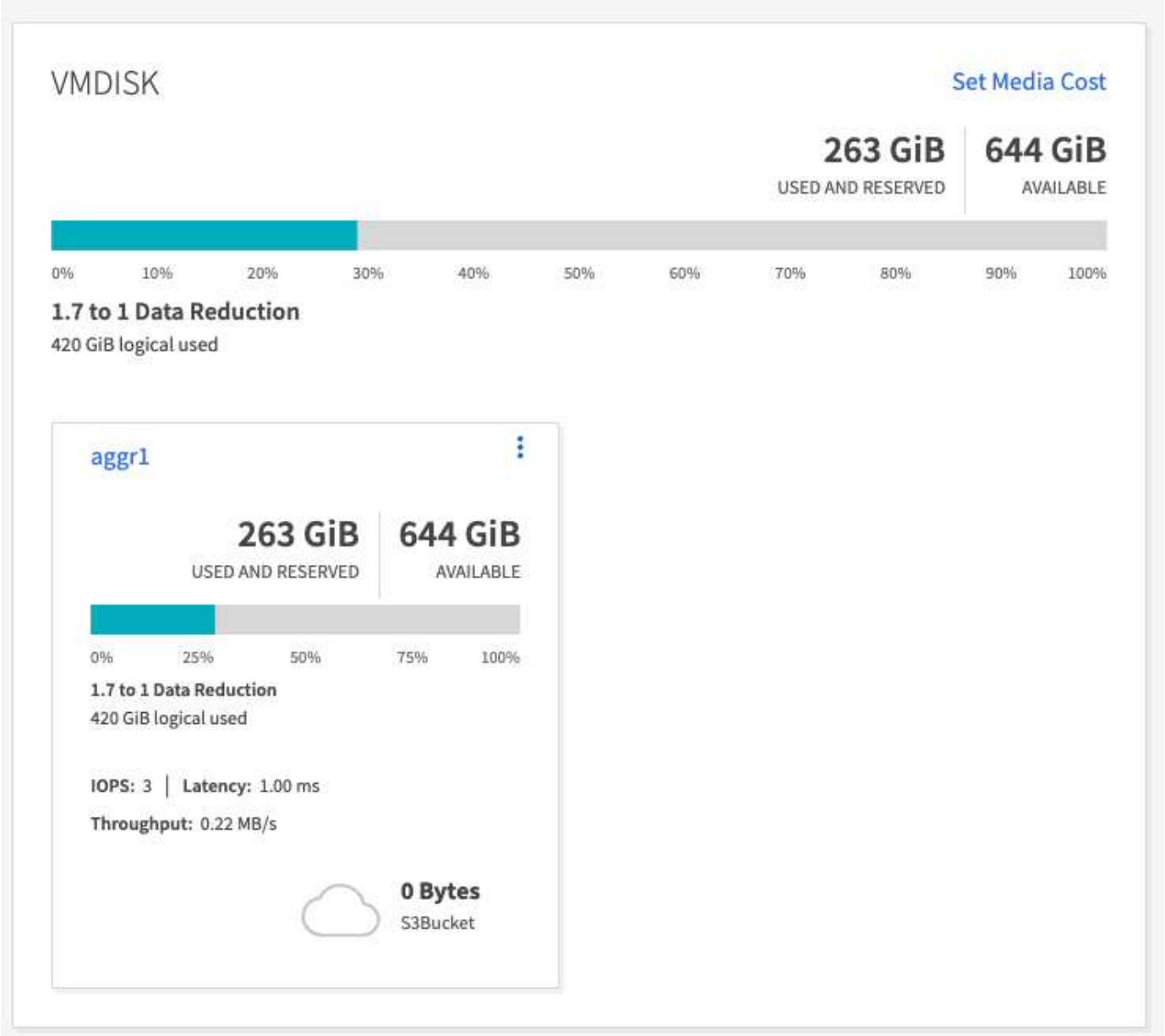
プロデューサー側で圧縮がオフになっているため、プロデューサーは高いスループットを生成できます。Storage Efficiencyは、代わりにコンピューティングレイヤで処理されました。

テストの方法論

1. 上記の仕様でKafkaクラスタがプロビジョニングされました。
2. クラスタでは、OpenMessaging Benchmarkingツールを使用して約350GBのデータが生成されました。
3. ワークロードの完了後、ONTAP System ManagerとCLIを使用してStorage Efficiencyの統計を収集しました。

観察

OMBツールを使用して生成したデータでは、ストレージ容量削減比率が1.70：1で約33%削減されました。次の図に示すように、生成されたデータに使用された論理スペースは420.3GB、データの保持に使用された物理スペースは281.7GBです。




```
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
```

AWSでのパフォーマンスの概要と検証

ストレージレイヤをNetApp NFS上にマウントしたKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークテストを実施しました。ベンチマークの例については、次のセクションで説明します。

AWSクラウドのKafkaとNetApp Cloud Volumes ONTAP （ハイアベイラビリティペアとシングルノード）

NetApp Cloud Volumes ONTAP （HAペア）を使用したKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークを実施しました。このベンチマークについては、以降のセクションで説明します。

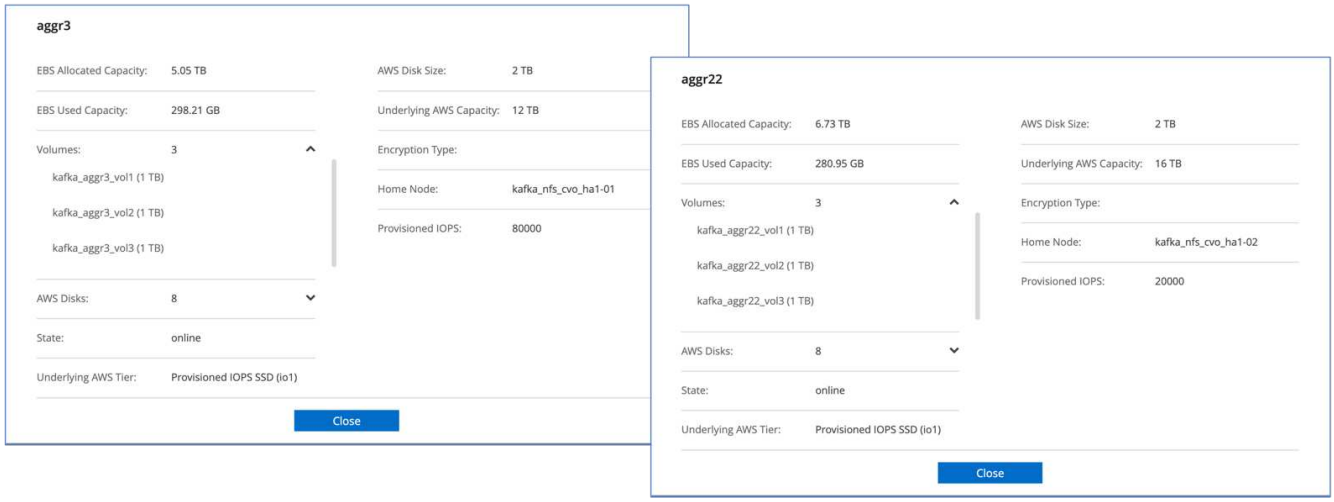
アーキテクチャのセットアップ

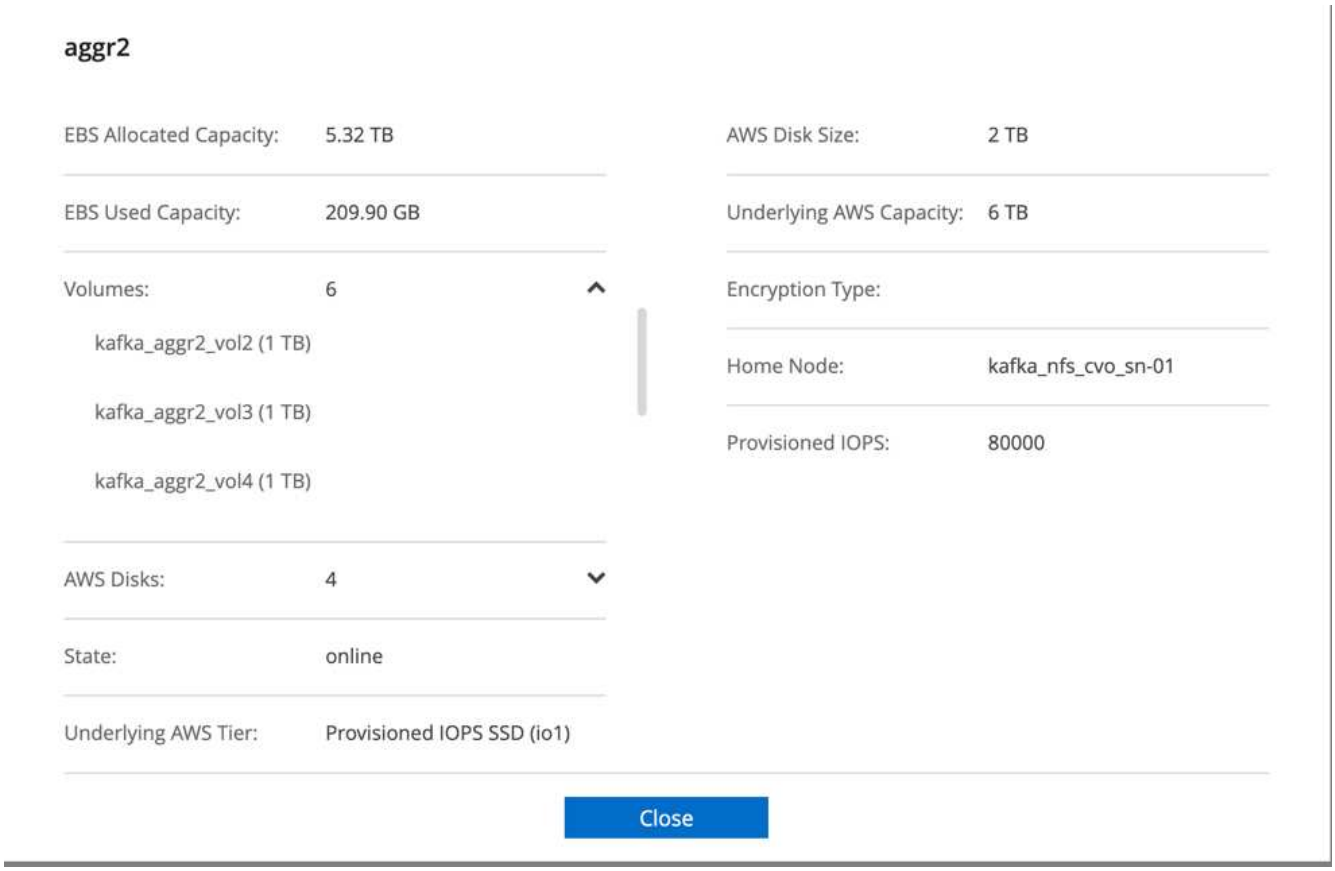
次の表に、NASを使用するKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	• 3 x動物飼育係–T2.small • ブローカーサーバ×3–i3en.2xlarge • Grafana–c5n.2xlarge×1 • 4 x producer/consumer — c5n.2xlarge *
すべてのノード上のオペレーティングシステム	RHEL8.6
NetApp Cloud Volumes ONTAP インスタンス	HAペアインスタンス–m5dn.12xLarge x 2ノードシングルノードインスタンス- m5dn.12xLarge x 1ノード

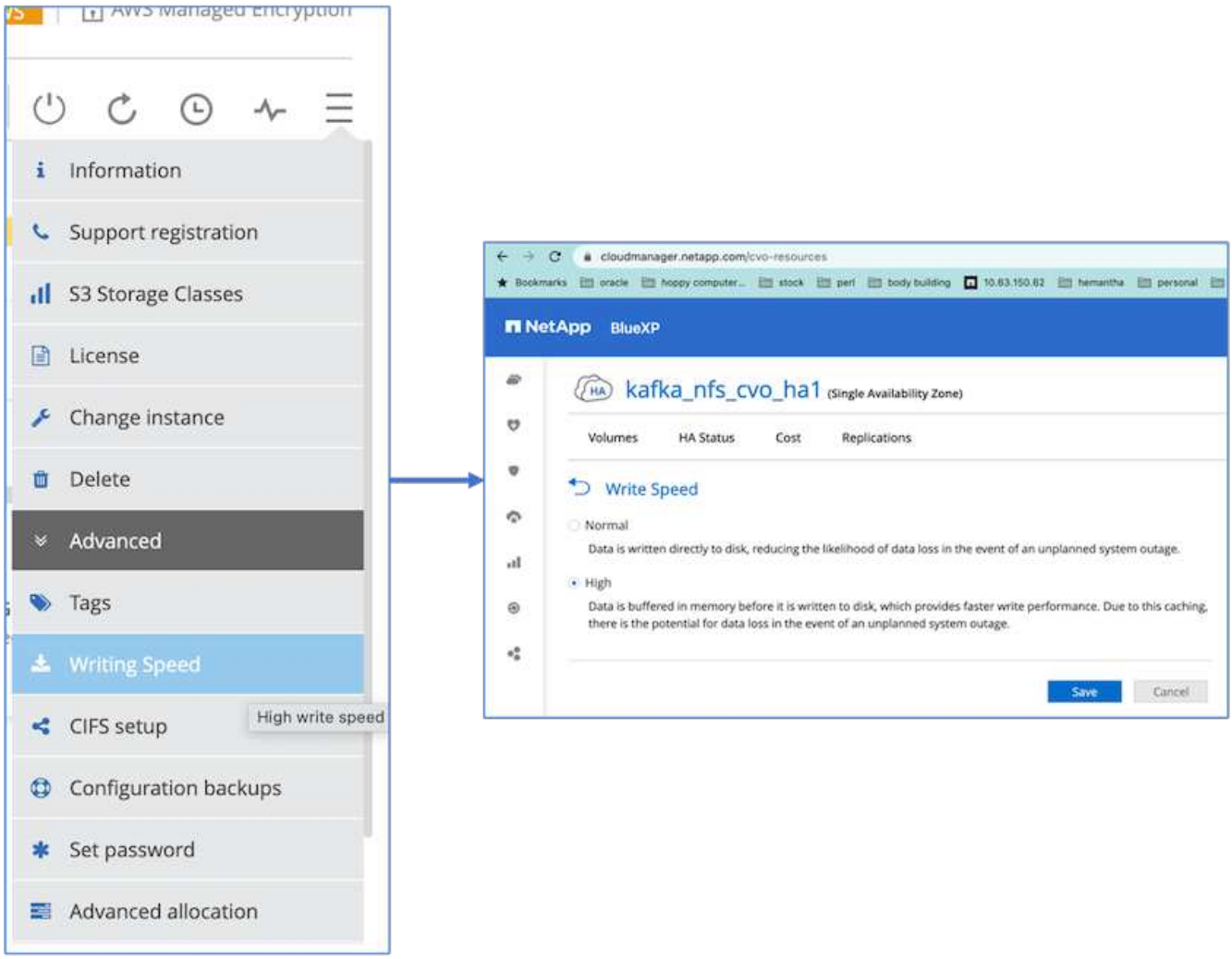
ネットアップクラスタボリュームのONTAP セットアップ

1. Cloud Volumes ONTAP HAペアについては、各ストレージコントローラの各アグリゲートに3つのボリュームを含む2つのアグリゲートを作成しました。単一のCloud Volumes ONTAP ノードの場合は、アグリゲートに6つのボリュームを作成します。



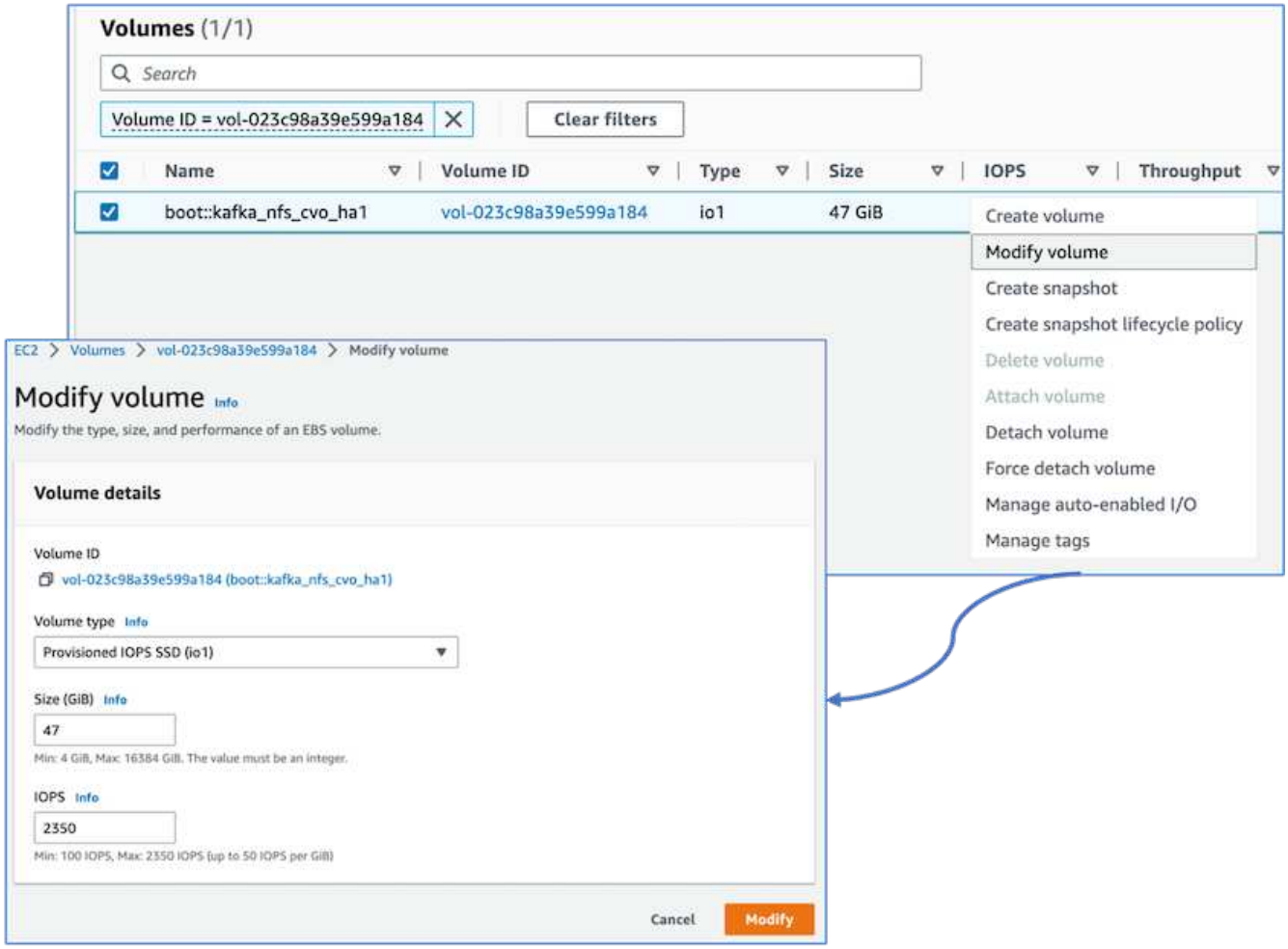


2. ネットワークパフォーマンスを高めるために、HAペアとシングルノードの両方で高速ネットワークを有効にしました。



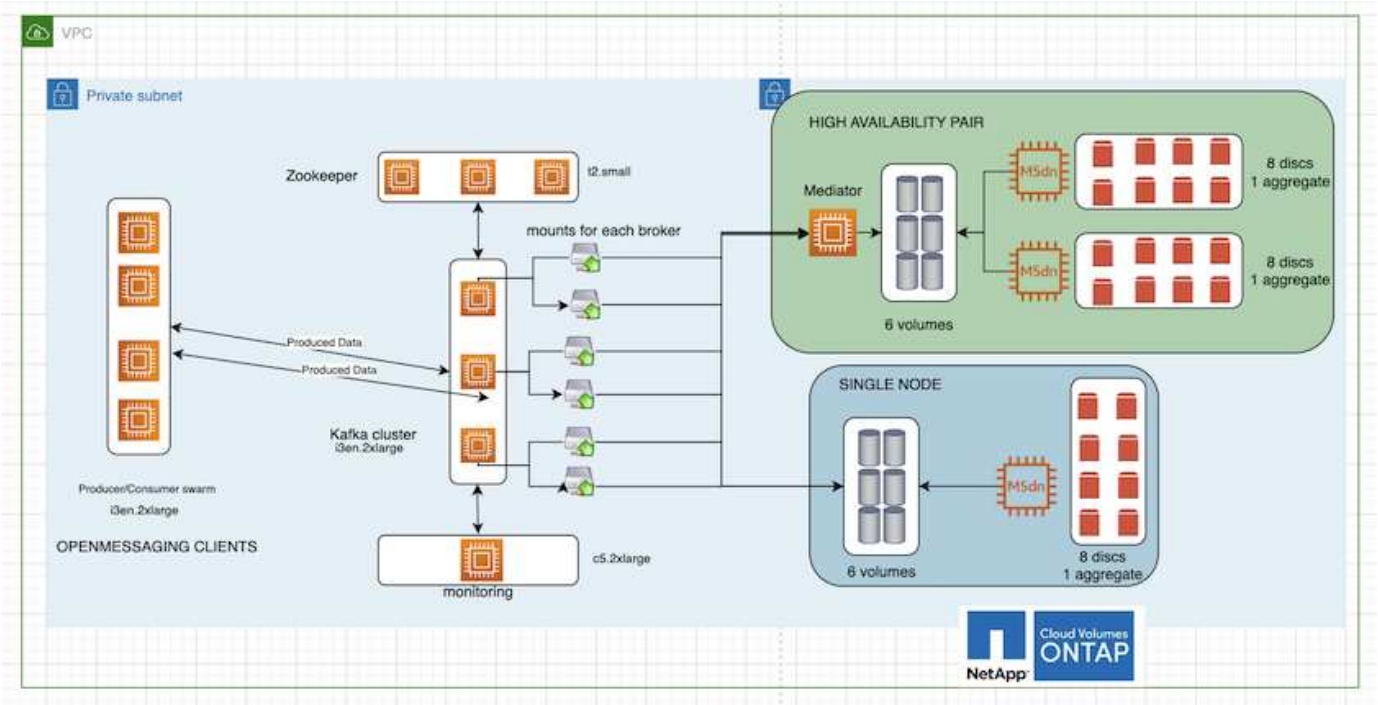
3. ONTAP NVRAMのIOPSが多いことに気付いたので、Cloud Volumes ONTAP ルートボリュームのIOPSを2350に変更しました。Cloud Volumes ONTAP のルートボリュームディスクのサイズは47GBでした。次のONTAP コマンドはHAペア用で、同じ手順をシングルノードにも適用します。

```
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::*>
```



次の図は、NASベースのKafkaクラスタのアーキテクチャを示しています。

- コンピューティング。 3ノードのKafkaクラスタを使用し、3ノードのZookeeperアンサンブルを専用サーバ上で実行しました。各ブローカーには、専用のLIFを介してCloud Volumes ONTAP インスタンス上の単一のボリュームへのNFSマウントポイントが2つあります。
- 監視。 Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成には、独立した3ノードクラスタを使用し、このKafkaクラスタを生成して使用しました。
- ストレージ。 HAペアCloud Volumes ONTAP インスタンスを使用し、インスタンスに6TB gp3 AWS-EBSボリュームを1つマウントしました。その後、ボリュームはNFSマウントを使用してKafkaブローカーにエクスポートされました。



OpenMessageベンチマーク設定

1. NFSのパフォーマンスを向上させるには、NFSサーバとNFSクライアントの間のネットワーク接続を増やす必要があります。この接続は、nconnectを使用して作成できます。次のコマンドを実行して、nconnectオプションを使用して、ブローカーノードにNFSボリュームをマウントします。

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

2. Cloud Volumes ONTAP でネットワーク接続を確認します。次のONTAP コマンドは、単一のCloud Volumes ONTAP ノードから使用します。同じ手順をCloud Volumes ONTAP HAペアにも適用できます。

```
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. 以下のKafkaを使用します `server.properties` Cloud Volumes ONTAP HAペアのすべてのKafkaブローカー。 `log.dirs` プロパティはブローカーごとに異なり、残りのプロパティはブローカーに共通です。broker1の場合は `log.dirs` 値は次のとおりです。

```
[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 ~]#
```

- 仲介業者2の場合は、 `log.dirs` プロパティ値は次のとおりです。

```
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
```

- 仲介業者3の場合は、 `log.dirs` プロパティ値は次のとおりです。

```
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. 単一のCloud Volumes ONTAP ノードの場合は、Kafka `servers.properties` は、を除き、Cloud Volumes ONTAP HAペアと同じです `log.dirs` プロパティ。
   - broker1の場合は `log.dirs` 値は次のとおりです。

     ```
     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
     ```

   - 仲介業者2の場合は、 `log.dirs` 値は次のとおりです。

     ```
     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
     ```

   - 仲介業者3の場合は、 `log.dirs` プロパティ値は次のとおりです。

     ```
     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. OMB内のワークロードには、次のプロパティが設定されます。 `(/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
```

。 `messageSize` はユースケースごとに異なる場合があります。パフォーマンステストでは、3Kを使用しました。

OMBのSyncまたはThroughputという2つのドライバを使用して、Kafkaクラスタでワークロードを生成しました。

- 同期ドライバのプロパティに使用されるYAMLファイルは次のとおりです (`/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
```

- スループットドライバのプロパティに使用されるYAMLファイルは次のとおりです (`/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
```

テストの方法論

1. Kafkaクラスタは、前述の仕様に従ってTerraformとAnsibleを使用してプロビジョニングされました。Terraformを使用して、Kafkaクラスタ用のAWSインスタンスを使用してインフラを構築し、Ansibleを使用してKafkaクラスタを構築します。
2. 上記のワークロード構成とSyncドライバでOMBワークロードがトリガーされました。

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

3. 同じワークロード構成でスループットドライバを使用して別のワークロードがトリガーされました。

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

観察

NFSで実行されるKafkaインスタンスのパフォーマンスをベンチマークするために、2種類のドライバを使用してワークロードを生成しました。ドライバの違いは、log flushプロパティです。

Cloud Volumes ONTAP HAペアの場合：

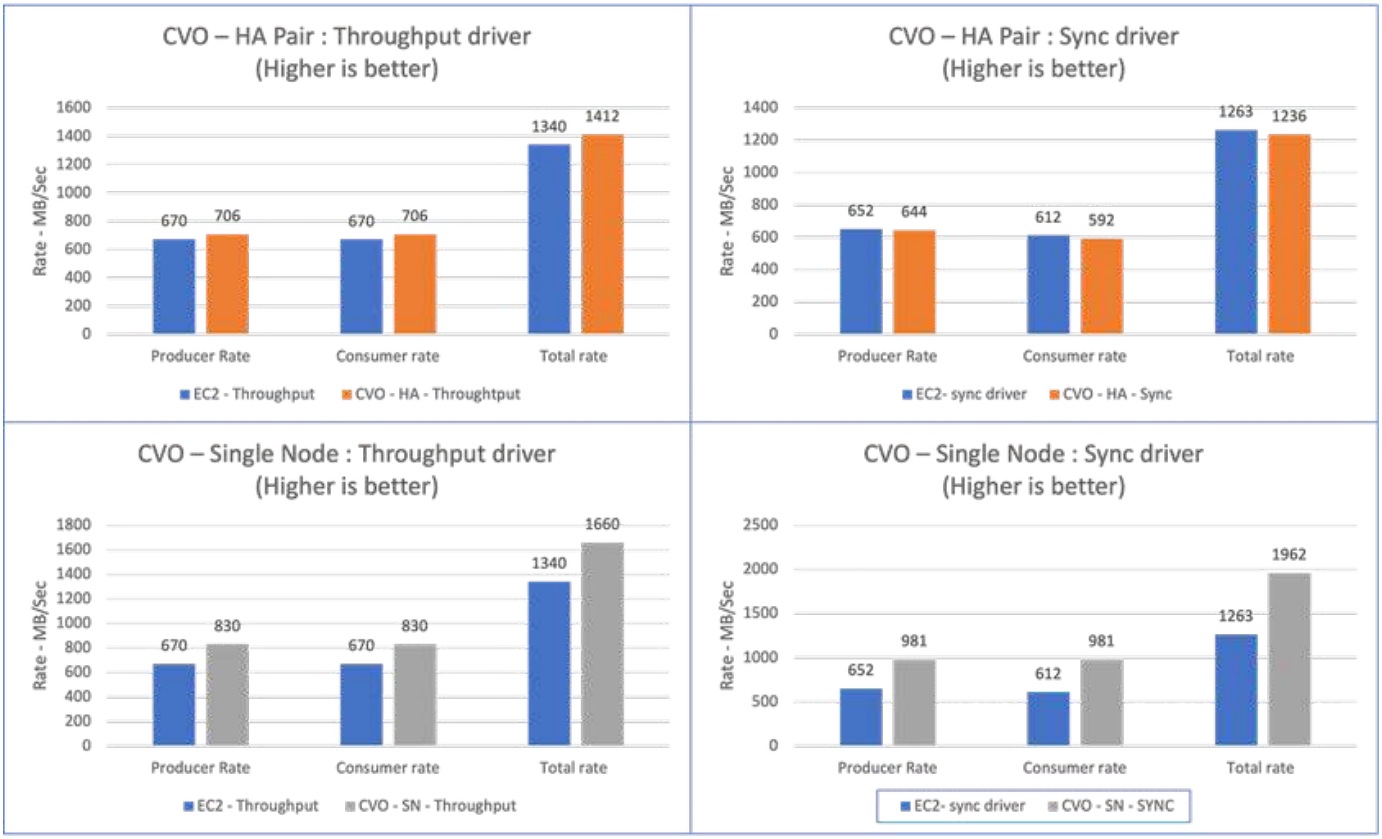
- Syncドライバによって一貫して生成される合計スループット：最大1236 Mbps
- スループットドライバに対して生成された合計スループット：ピーク時最大1412 Mbps

単一のCloud Volumes ONTAP ノードの場合：

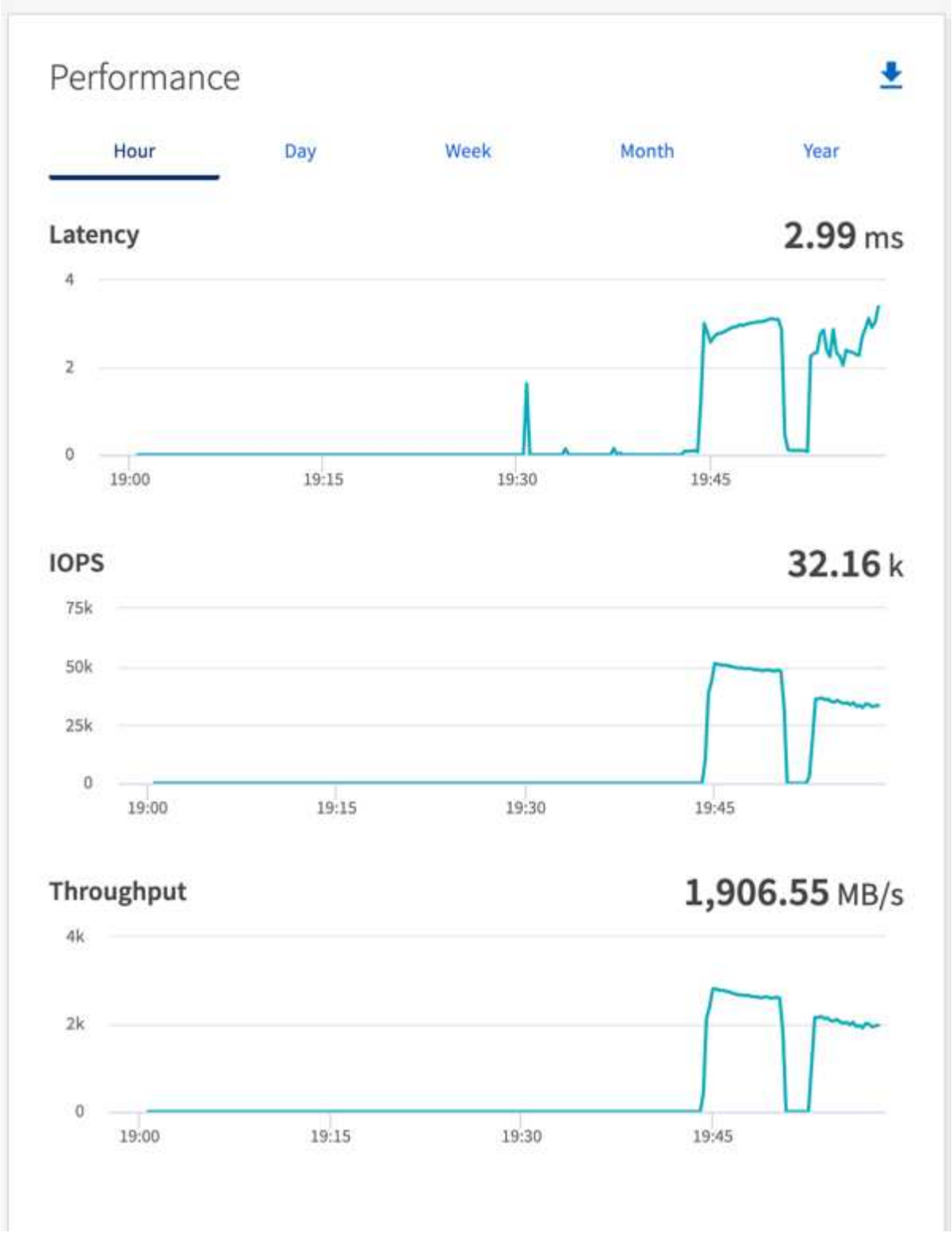
- Syncドライバで一貫して生成される合計スループット：約1962MBps
- スループットドライバによって生成される合計スループット：最大1660MBps

Syncドライバはログが即座にディスクにフラッシュされるときに一貫したスループットを生成できますが、Throughputドライバはログがディスクに一括コミットされるときにスループットのバーストを生成します。

これらのスループット値は、指定されたAWS構成に対して生成されます。より高いパフォーマンス要件に対応するには、インスタンスタイプをスケールアップしてさらに調整し、スループットを向上させることができます。総スループットまたは総レートは、生産者と消費者の両方のレートの組み合わせです。



スループットまたは同期ドライバのベンチマークを実行するときは、必ずストレージスループットを確認してください。



AWS FSx for NetApp ONTAPでのパフォーマンスの概要と検証

ストレージレイヤをNetApp NFS上にマウントしたKafkaクラスタは、AWS FSx for NetApp ONTAPでパフォーマンスのベンチマークを実施しました。ベンチマークの例については、次のセクションで説明します。

AWS FSx for NetApp ONTAPでのApache Kafkaの使用

Network File System（NFS；ネットワークファイルシステム）は、大量のデータを格納するために広く使用されているネットワークファイルシステムです。ほとんどの組織では、データがApache Kafkaなどのストリーミングアプリケーションによって生成されるようになっています。このようなワークロードには、拡張性、低レイテンシ、最新のストレージ機能を備えた堅牢なデータ取り込みアーキテクチャが必要です。リアルタイム分析を実現し、実用的な分析情報を提供するには、適切に設計されたパフォーマンスの高いインフラが必要です。

Kafkaは、設計上POSIX準拠のファイルシステムと連携し、ファイル操作をファイルシステムに依存して処理しますが、NFSv3ファイルシステムにデータを格納する場合、KafkaブローカーのNFSクライアントは、ファイル操作をXFSやext4などのローカルファイルシステムとは異なる方法で解釈できます。一般的な例としては、NFSのsilly renameがあり、クラスタの拡張時やパーティションの再割り当て時にKafkaブローカーでエラーが発生していました。この課題に対処するため、NetAppはオープンソースのLinux NFSクライアントを更新し、RHEL8.7とRHEL9.1で一般提供されるようになりました。また、最新のFSx for NetApp ONTAPリリースであるONTAP 9.12.1からサポートされるようになりました。

Amazon FSx for NetApp ONTAPは、拡張性とパフォーマンスに優れたフルマネージドのNFSファイルシステムをクラウドで提供します。FSx for NetApp ONTAPのKafkaデータは、大量のデータを処理し、フォールトトレランスを確保するように拡張できます。NFSは、重要で機密性の高いデータセットに対して一元的なストレージ管理とデータ保護を提供します。

これらの機能強化により、AWSのお客様はAWSコンピューティングサービスでKafkaワークロードを実行する際にFSx for NetApp ONTAPを活用できるようになります。次のようなメリットがあります。
* CPU使用率を削減して、I/O待機時間を短縮します
* Kafkaブローカーのリカバリ時間の短縮。
*信頼性と効率性。
*拡張性とパフォーマンス。
*マルチアベイラビリティゾーンの可用性。
*データ保護：

AWS FSx for NetApp ONTAPでのパフォーマンスの概要と検証

ストレージレイヤをNetApp NFS上にマウントしたKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークテストを実施しました。ベンチマークの例については、次のセクションで説明します。

AWS FSx for NetApp ONTAPのKafka

AWS FSx for NetApp ONTAPを使用したKafkaクラスタは、AWSクラウドでのパフォーマンスについてベンチマークを実施しました。このベンチマークについては、以降のセクションで説明します。

アーキテクチャのセットアップ

次の表に、AWS FSx for NetApp ONTAPを使用したKafkaクラスタの環境構成を示します。

プラットフォームコンポーネント	環境の構成
Kafka 3.2.3.	• 3 x動物飼育係–T2.small • ブローカーサーバ×3–i3en.2xlarge • Grafana–c5n.2xlarge×1 • 4 x producer/consumer — c5n.2xlarge *

プラットフォームコンポーネント	環境の構成
すべてのノード上のオペレーティングシステム	RHEL8.6
AWS FSx for NetApp ONTAP	マルチアストラゼネカ、4GB/秒のスループット、160000 IOPS

NetApp FSx for NetApp ONTAPのセットアップ

1. 最初のテストでは、2TBの容量と40000のIOPSで2GB/秒のスループットを実現するFSx for NetApp ONTAPファイルシステムを作成しました。

```
[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"}
```

   この例では、AWS CLIを使用してFSx for NetApp ONTAPを導入しています。必要に応じて、環境内でコマンドをさらにカスタマイズする必要があります。FSx for NetApp ONTAPはAWSコンソールからさらに導入、管理できるため、コマンドラインの入力が少なく、導入がより簡単かつ合理的になります。

   ドキュメントFSx for NetApp ONTAPでは、テストリージョン（US-East-1）の2GB/秒スループットファイルシステムで達成可能な最大IOPSは8万IOPSです。FSx for NetApp ONTAPファイルシステムの合計最大IOPSは16万IOPSですが、そのためには4GB/秒のスループット導入が必要です。このことについては、このドキュメントの後半で説明します。

   FSx for NetApp ONTAPのパフォーマンス仕様の詳細については、AWS FSx for NetApp ONTAPのドキュメントを参照してください。 https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html 。

   FSx「create-file-system」のコマンドライン構文の詳細については、以下を参照してください。 https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

   たとえば、KMSキーが指定されていない場合に使用されるデフォルトのAWS FSxマスターキーとは異なり、特定のKMSキーを指定できます。

2. FSx for NetApp ONTAPファイルシステムを作成するときは、ファイルシステムを次のように定義したら、JSONで「lifecycle」ステータスが「available」に変わるまで待ちます。

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

3. fsxadminユーザを使用してFSx for NetApp ONTAP SSHにログインし、クレデンシャルを検証します。Fsxadminは、作成時にFSx for NetApp ONTAPファイルシステムのデフォルトの管理者アカウントです。fsxadminのパスワードは、手順1で完了したように、AWSコンソールまたはAWS CLIでファイルシステムを最初に作成したときに設定したパスワードです。

```
[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. クレデンシャルの検証が完了したら、FSx for NetApp ONTAPファイルシステムにSVMを作成

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

Storage Virtual Machine（SVM）は分離されたファイルサーバで、FSx for NetApp ONTAPボリューム内のデータを管理およびアクセスするための独自の管理クレデンシャルとエンドポイントを備え、FSx for NetApp ONTAPマルチテナンシーを提供します。

5. プライマリSVMの設定が完了したら、新しく作成したFSx for NetApp ONTAPファイルシステムにSSHで接続し、以下のサンプルコマンドを使用してSVMにボリュームを作成します。同様に、この検証用に6つのボリュームを作成します。Kafkaのパフォーマンスが向上するように、デフォルトのコンスティチュエント（8個）以下のコンスティチュエントを使用してください。

```
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. テスト用にボリュームに追加の容量が必要になります。ボリュームのサイズを2TBに拡張し、ジャンクションパスにマウントします。

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

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

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

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN4 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN4" size set to 2.10t.

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

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

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

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

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

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

```
-path /kafkafsxN3

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

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

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

FSx for NetApp ONTAPでは、ボリュームをシンプロビジョニングできます。この例では、拡張されたボリュームの合計容量がファイルシステムの合計容量を超えているため、プロビジョニングされた追加のボリューム容量のロックを解除するには、ファイルシステムの合計容量を拡張する必要があります。これについては、次の手順で説明します。

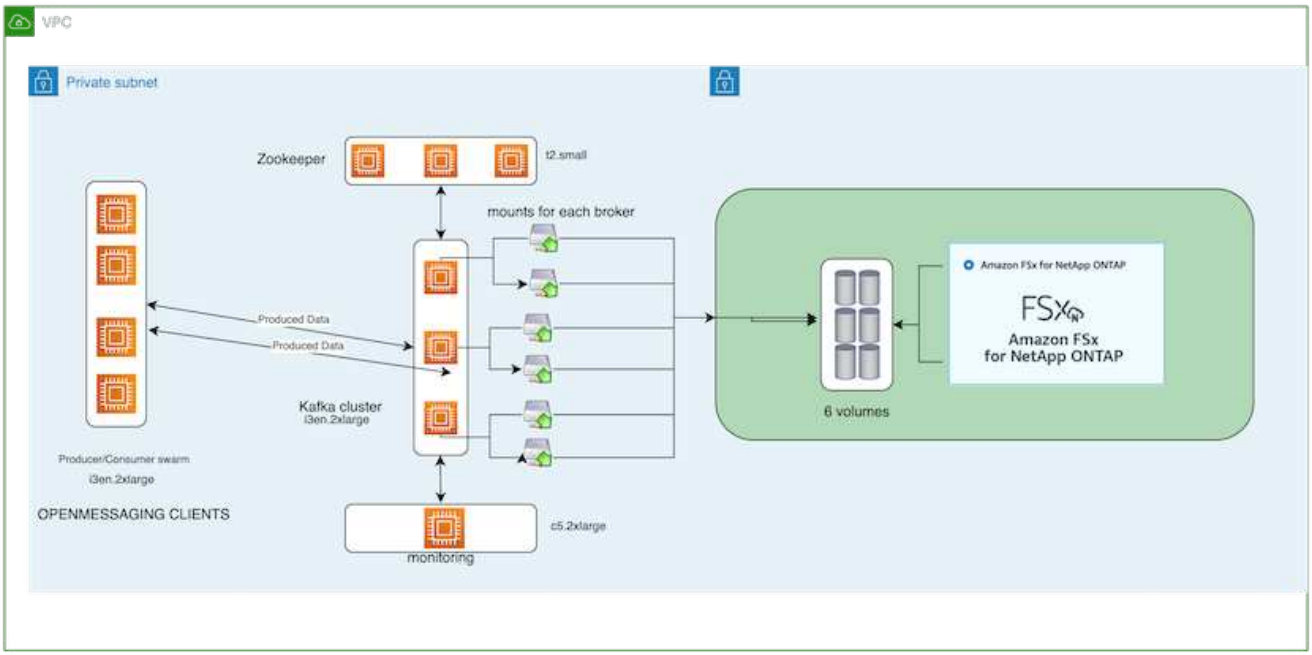
7. 次に、パフォーマンスと容量を強化するために、FSx for NetApp ONTAPのスループット容量を2GB/秒から4GB/秒、IOPSから160000、容量を5TBに拡張しました。

```
[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」の詳細なコマンドライン構文は、次のとおりです。
https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html

8. FSx for NetApp ONTAPボリュームは、nconnectおよびデフォルトのオプションを使用してKafkaブローカーにマウントされます。

次の図は、FSx for NetApp ONTAPベースのKafkaクラスタの最終的なアーキテクチャを示しています。



- コンピューティング：3ノードのKafkaクラスタを使用し、専用サーバで3ノードのZookeeperアンサンブルを実行しました。各ブローカーには、FSx for NetApp ONTAPインスタンス上の6つのボリュームに対するNFSマウントポイントが6つありました。
- 監視：Prometheus-Grafanaの組み合わせには2つのノードを使用しました。ワークロードの生成には、独立した3ノードクラスタを使用し、このKafkaクラスタを生成して使用しました。
- ストレージ：FSx for NetApp ONTAPを使用し、2TBのボリュームを6個マウントしました。その後、NFSマウントを使用してボリュームをKafkaブローカーにエクスポートしました。FSx for NetApp ONTAPボリュームは、16のnconnectセッションとKafkaブローカーのデフォルトオプションでマウントされます。

OpenMessageベンチマーク設定。

NetApp Cloud Volumes ONTAPと同じ構成を使用しました。詳細はこちらをご覧ください。
リンク：kafka-nfs-performance-overview-and-validation-in-aws.html#architecture-setup

テストの方法論

1. Kafkaクラスタは、前述の仕様に従ってterraformとAnsibleを使用してプロビジョニングされました。Terraformを使用して、Kafkaクラスタ用のAWSインスタンスを使用してインフラを構築し、Ansibleを使用してKafkaクラスタを構築します。
2. 上記のワークロード構成とSyncドライバでOMBワークロードがトリガーされました。

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

3. 同じワークロード構成でスループットドライバを使用して別のワークロードがトリガーされました。

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

観察

NFSで実行されるKafkaインスタンスのパフォーマンスをベンチマークするために、2種類のドライバを使用してワークロードを生成しました。ドライバの違いは、log flushプロパティです。

Kafkaレプリケーションファクタ1とFSx for NetApp ONTAPの場合：

- Syncドライバで一貫して生成された総スループット：最大3218 Mbps、最大パフォーマンス（最大3652 Mbps）
- スループットドライバによって一貫して生成された総スループット：最大3679 Mbps、最大3908 Mbpsのパフォーマンス。

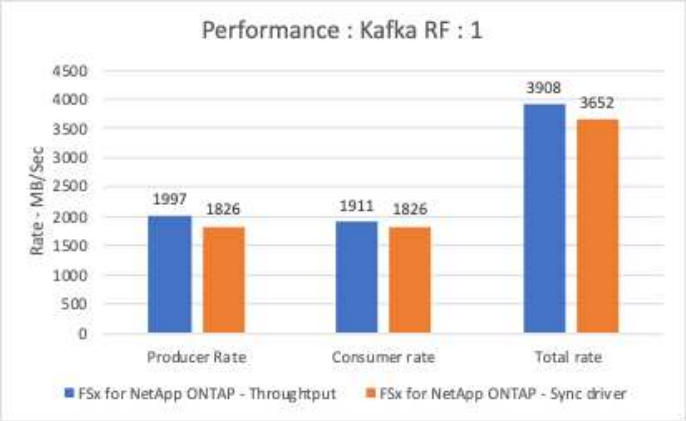
レプリケーションファクタ3のKafkaとFSx for NetApp ONTAPの場合：

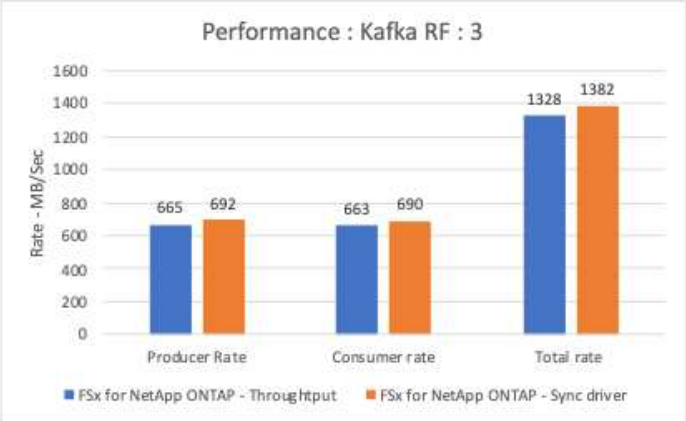
- Syncドライバで一貫して生成される合計スループット：最大1252 Mbps、最大1382 Mbps。
- スループットドライバによって一貫して生成される総スループット：最大1218 Mbps、最大パフォーマンス（最大1328 Mbps）

Kafkaレプリケーションファクタ3では、FSx for NetApp ONTAPで読み取りと書き込みの処理が3回行われました。Kafkaレプリケーションファクタ1では、読み取りと書き込みの処理がFSx for NetApp ONTAPで1回行われたため、どちらの検証でも、 4GB/秒の最大スループットに到達できました。

Syncドライバはログが即座にディスクにフラッシュされるときに一貫したスループットを生成できますが、Throughputドライバはログがディスクに一括コミットされるときにスループットのバーストを生成します。

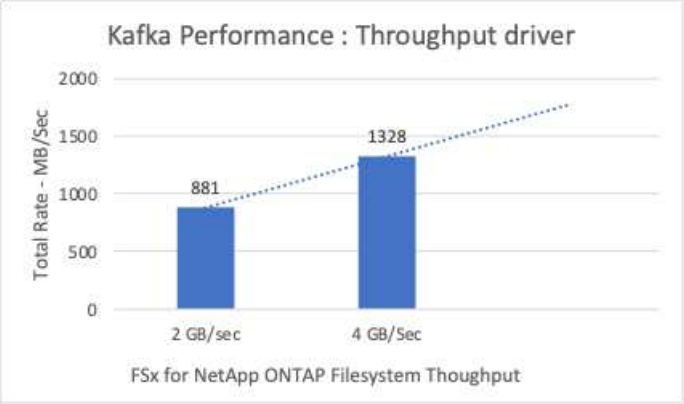
これらのスループット値は、指定されたAWS構成に対して生成されます。より高いパフォーマンス要件に対応するには、インスタンスタイプをスケールアップしてさらに調整し、スループットを向上させることができます。総スループットまたは総レートは、生産者と消費者の両方のレートの組み合わせです。

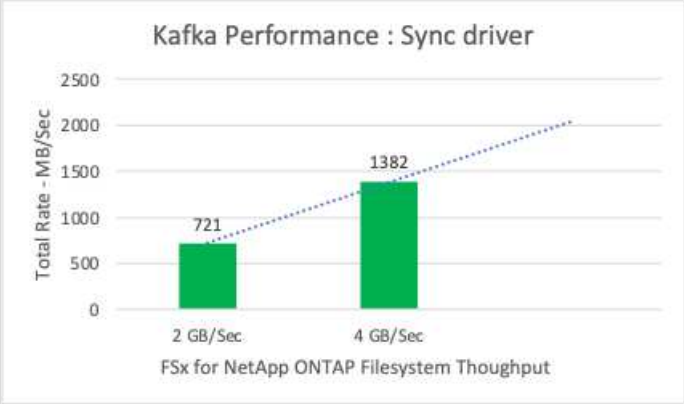




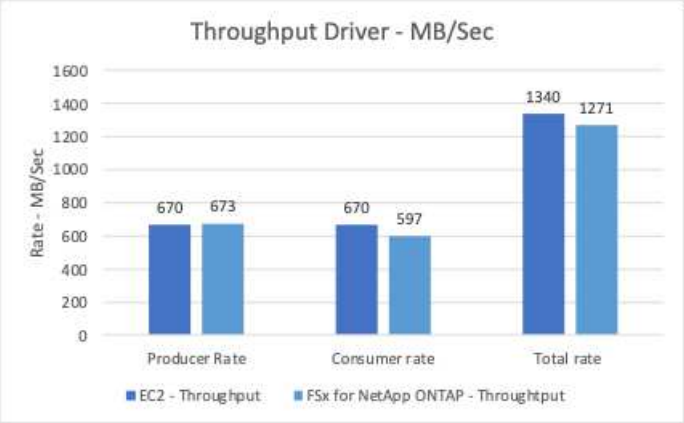
以下の表は、Kafkaレプリケーションファクタ3で2GB/秒のFSx for NetApp ONTAPと4GB/秒のパフォーマンスを示しています。レプリケーションファクタ3は、FSx for NetApp ONTAPストレージで読み取りと書き込みの処理を3回行います。スループットドライバの合計レートは881 MB/秒で、2GB/秒のFSx for NetApp ONTAPファイルシステムではKafkaの読み取りと書き込みを行います。スループットドライバの合計レートは1328

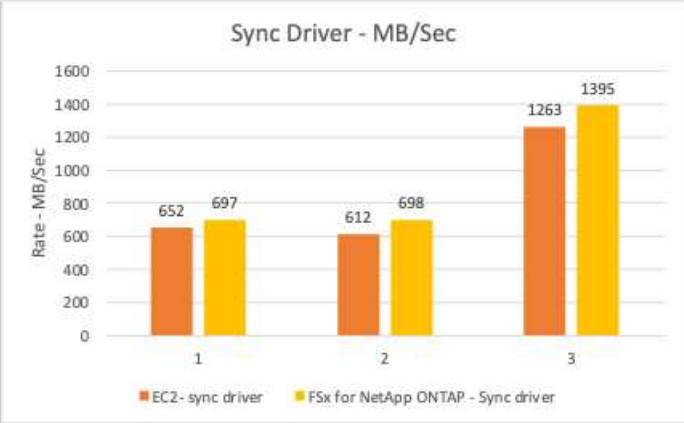
MB/秒で、Kafkaの読み取りと書き込みを行います。Kafkaのパフォーマンスは、FSx for NetApp ONTAPのスループットに基づいてリニアで拡張性に優れています。





以下のグラフは、EC2インスタンスとFSx for NetApp ONTAPのパフォーマンスを示しています（Kafkaレプリケーション係数：3）。

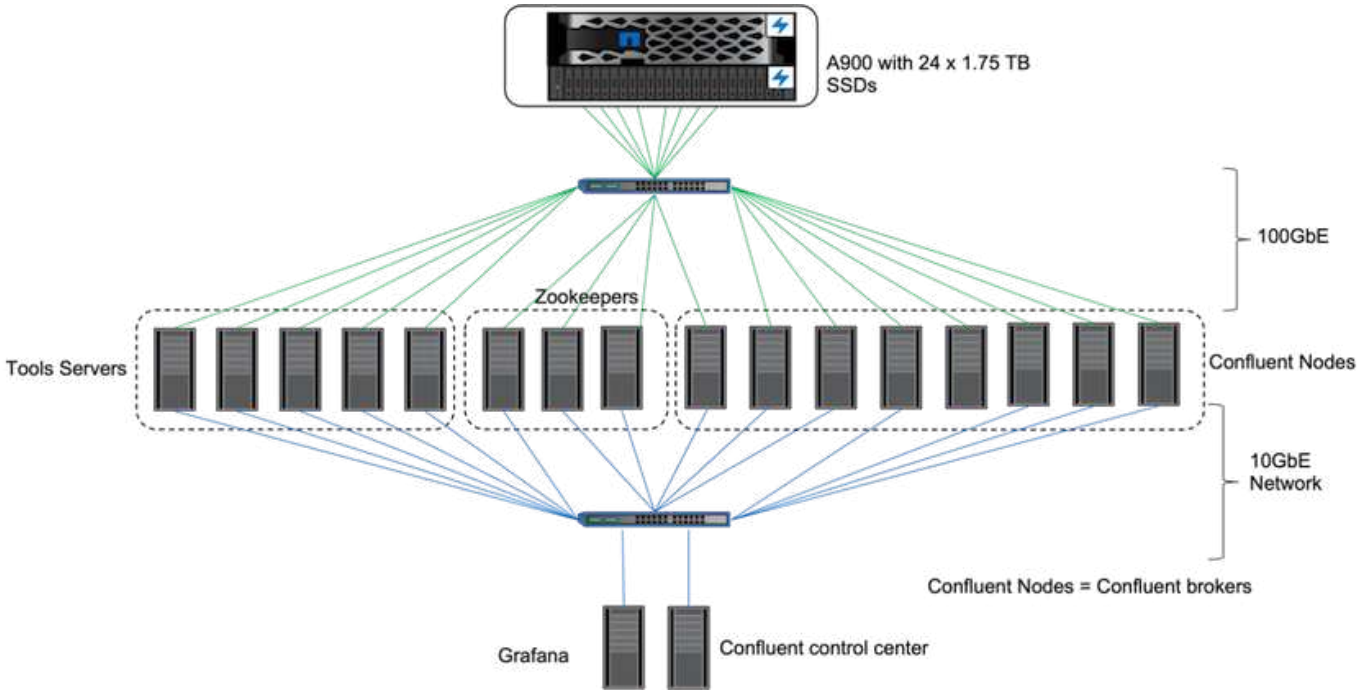




オンプレミスの**AFF A900**によるパフォーマンスの概要と検証

オンプレミスでは、NetApp AFF A900ストレージコントローラとONTAP 9.12.1RC1を使用して、Kafkaクラスタのパフォーマンスと拡張性を検証しました。ONTAP およびAFFでは、以前の階層型ストレージのベストプラクティスと同じテストベッドを使用しました。

AFF A900の評価にはConfluent Kafka 6.2.0を使用しました。このクラスタには、8つのブローカーノードと3つのzookeeperノードがあります。パフォーマンステストでは、5つのOMBワーカーノードを使用しました。



ストレージ構成

ネットアップのFlexGroupインスタンスを使用してログディレクトリに単一のネームスペースを提供し、リカバリと設定を簡易化しました。NFSv4.1とpNFSを使用して、ログセグメントデータへの直接アクセスを提供しました。

クライアントの調整

各クライアントは、次のコマンドを使用してFlexGroup インスタンスをマウントしました。

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

さらに、を増やしました `max_session_slots` デフォルトから変更します `64` 終了： `180`。これは、ONTAP のデフォルトのセッションスロット制限と一致します。

Kafkaブローカーチューニング

テスト対象のシステムのスループットを最大化するために、特定のキースレッドプールのデフォルトパラメータを大幅に増やしました。ほとんどの構成では、Kafkaのベストプラクティスに準拠することを推奨します。この調整は、ストレージに対する未処理のI/Oの同時処理率を最大化するために使用されました。これらのパラメータは、ブローカーのコンピューティングリソースとストレージ属性に合わせて調整できます。

```
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
```

ワークロードジェネレータのテスト方法

スループットドライバとトピック構成には、クラウドテストと同じOMB構成を使用しました。

1. AFF クラスタでAnsibleを使用してFlexGroup インスタンスをプロビジョニングしました。

```
---
- 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 SVMでpNFSが有効になりました。

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

3. Cloud Volumes ONTAP と同じワークロード構成でを使用してスループットドライバでワークロードがトリガーされました。「」を参照してください[安定した状態でのパフォーマンス]」を参照してください。このワークロードではレプリケーションファクタ3を使用しました。つまり、NFSでログセグメントのコピーが3つ保持されていました。

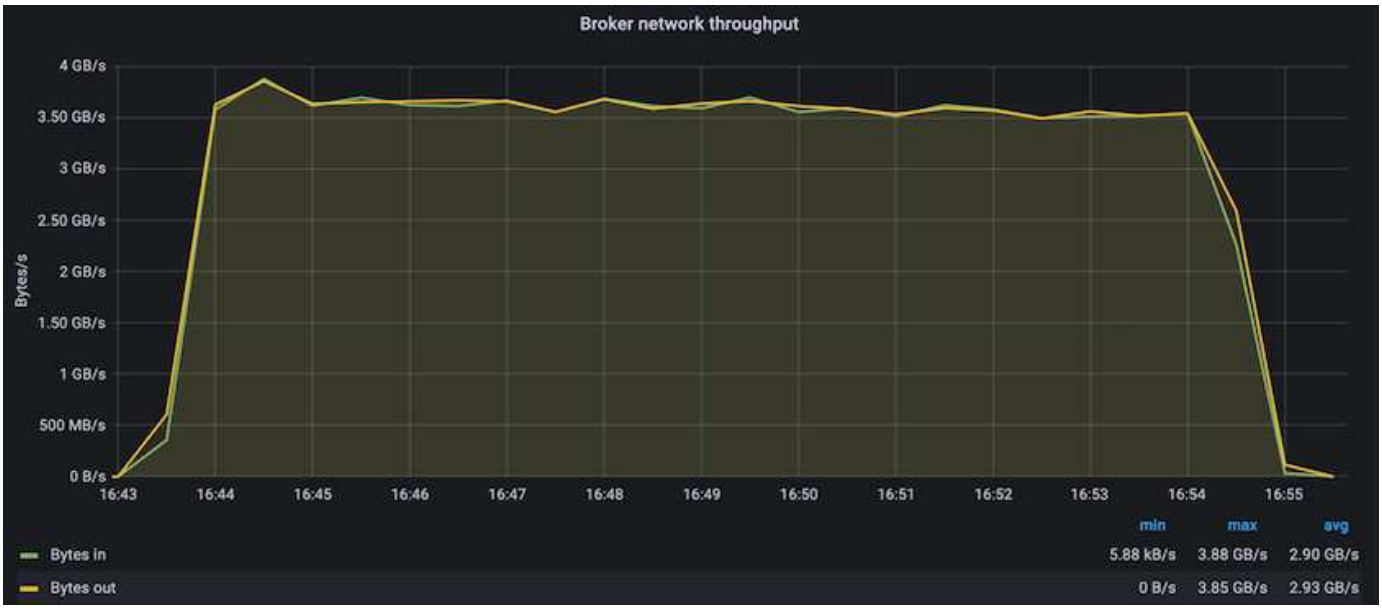
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最後に、バックログを使用して測定を完了し、消費者が最新のメッセージに追いつく能力を測定しました。OMBは、測定の開始時にコンシューマを一時停止することでバックログを作成します。これにより、バックログの作成（生産者のみのトラフィック）、バックログの排出（消費者がトピックで見逃したイベントに追いつくための消費者の多いフェーズ）、および定常状態の3つの異なるフェーズが生成されます。「」を参照してください[卓越したパフォーマンスを発揮し、ストレージの限界を探る]」を参照してください。

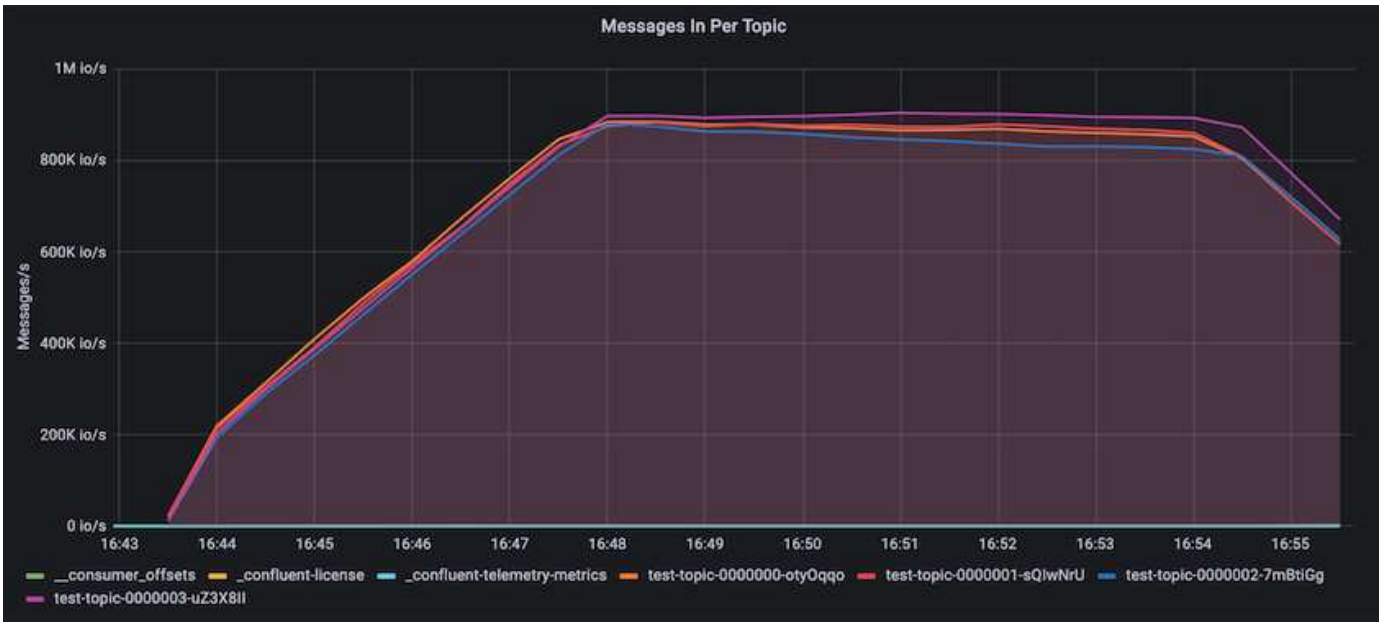
安定した状態でのパフォーマンス

AFF A900は、OpenMessagingベンチマークを使用して評価し、AWSのCloud Volumes ONTAP とAWSのDASと同様の比較を行いました。すべてのパフォーマンス値は、プロデューサーレベルとコンシューマレベルでのKafkaクラスタのスループットを表します。

Confluent KafkaとAFF A900を使用した定常状態のパフォーマンスは、生産者と消費者の両方で平均3.4GBps以上のスループットを達成しました。これは、Kafkaクラスタ全体で340万件を超えるメッセージです。BrokerTopicMetricsの持続スループット（バイト/秒）を視覚化することで、AFF A900がサポートする優れた安定状態のパフォーマンスとトラフィックを確認できます。



これは、トピックごとに配信されるメッセージのビューとよく一致しています。次のグラフは、トピックごとの内訳を示しています。テストした構成では、4つのトピックにわたってトピックごとに約9万件のメッセージが表示されました。

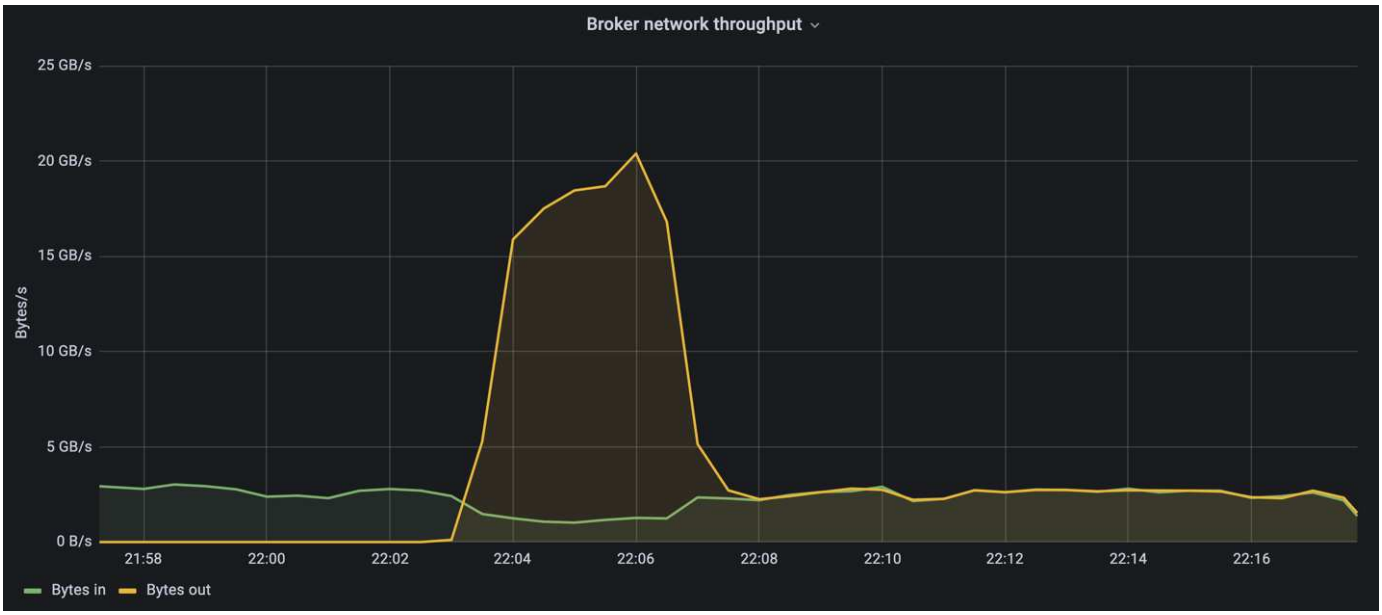


卓越したパフォーマンスを発揮し、ストレージの限界を探る

AFF では、バックログ機能を使用してOMBでテストしました。バックログ機能は、Kafkaクラスタでイベントのバックログが作成されている間、コンシューマサブスクリプションを一時停止します。このフェーズでは、プロデューサトラフィックのみが発生し、ログにコミットされたイベントが生成されます。これは、バッチ処理またはオフライン分析ワークフローを最も厳密にエミュレートします。これらのワークフローでは、コンシューマーサブスクリプションが開始され、ブローカーキャッシュからすでに削除されている履歴データを読み取る必要があります。

この構成で利用者のスループットに関するストレージの制限を把握するために、Producer-Onlyフェーズを測定して、A900がどの程度の書き込みトラフィックを吸収できるかを調べました。次のセクションを参照してください[サイジングガイダンス]」を参照してください。

この測定のプロデューサーのみの部分では、ピークスループットが高く、A900のパフォーマンスの限界を押し上げていることがわかりました（他のブローカーリソースがプロデューサーおよびコンシューマートラフィックに対応していない場合）。



この測定では、メッセージあたりのオーバーヘッドを制限し、NFSマウントポイントに対するストレージスループットを最大化するために、メッセージサイズを16kに増やしました。

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

Confluent Kafkaクラスターは、生産者のピークスループット4.03GBpsを達成しました。

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

OMBによるイベントバックログの入力が完了すると、コンシューマトラフィックが再開されました。バックログドレーンを使用した測定では、すべてのトピックで消費者のピークスループットが20Gbpsを超えることが確認されました。OMBログデータを格納するNFSボリュームの合計スループットは約30Gbpsに達しました。

サイジングガイダンス

Amazon Web Servicesは、を提供します "サイジングガイド" Kafkaクラスタのサイジングと拡張に使用できます。

このサイジングは、Kafkaクラスタのストレージスループット要件を決定するのに便利な計算式を提供します。

tclusterのクラスタ内で生成される集約スループット（レプリケーション係数r）の場合、ブローカーストレージが受け取るスループットは次のとおりです。

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

これはさらに単純化することができます。

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

この式を使用すると、Kafkaホットティアのニーズに適したONTAP プラットフォームを選択できます。

次の表に、A900の予測される生産者スループットと、さまざまなレプリケーション要因を示します。

レプリケーションファクタ	生産者スループット（**GPPS**）
3（実測値）	3.4.
2.	5.1
1.	10.2

まとめ

NetApp解決策 for the silly rename problemは、これまでNFSと互換性がなかったワークロード向けに、シンプルで安価な一元管理されたストレージを提供します。

この新しいパラダイムにより、ディザスタリカバリやデータ保護を目的とした移行やミラーリングが容易な、管理しやすいKafkaクラスタを作成できます。
また、NFSは、CPU利用率の削減とリカバリ時間の短縮、ストレージ効率の大幅な向上、NetApp ONTAP によるパフォーマンスの向上など、追加のメリットをもたらすこともわかりました。

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- Apache Kafkaとは

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

- 愚かな名前変更とは何ですか?

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

- ONATPはストリーミングアプリケーション用に読み取られます。

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

- Silly -問題 の名前をKafkaに変更します。

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

- ネットアップの製品マニュアル

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

- NFSとは

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

- Kafkaパーティションの再割り当てとは何ですか。

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

- OpenMessagingベンチマークとは

  "https://openmessaging.cloud/"

- Kafkaブローカーはどのように移行しますか？

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

- PrometheusでKafkaブローカーをどのように監視していますか？

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

- Apache Kafka向けマネージドプラットフォーム

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

- Apache Kafkaのサポート

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

- Apache Kafkaのコンサルティングサービス

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

NetApp ONTAP ストレージコントローラとKafkaが競合します

TR-4941：ネットアップのONTAP ストレージコントローラに精通していること

Karthikeyan Nagalingam、Joe Scott、NetApp Rankesh Kumar、Conluent

Conluent Platformの拡張性と柔軟性を高めるには、ワークロードを非常に迅速に拡張し、バランスを調整できる必要があります。階層型ストレージを使用すると、このような運用上の負担が軽減されるため、大量のデータを管理しやすくなります。

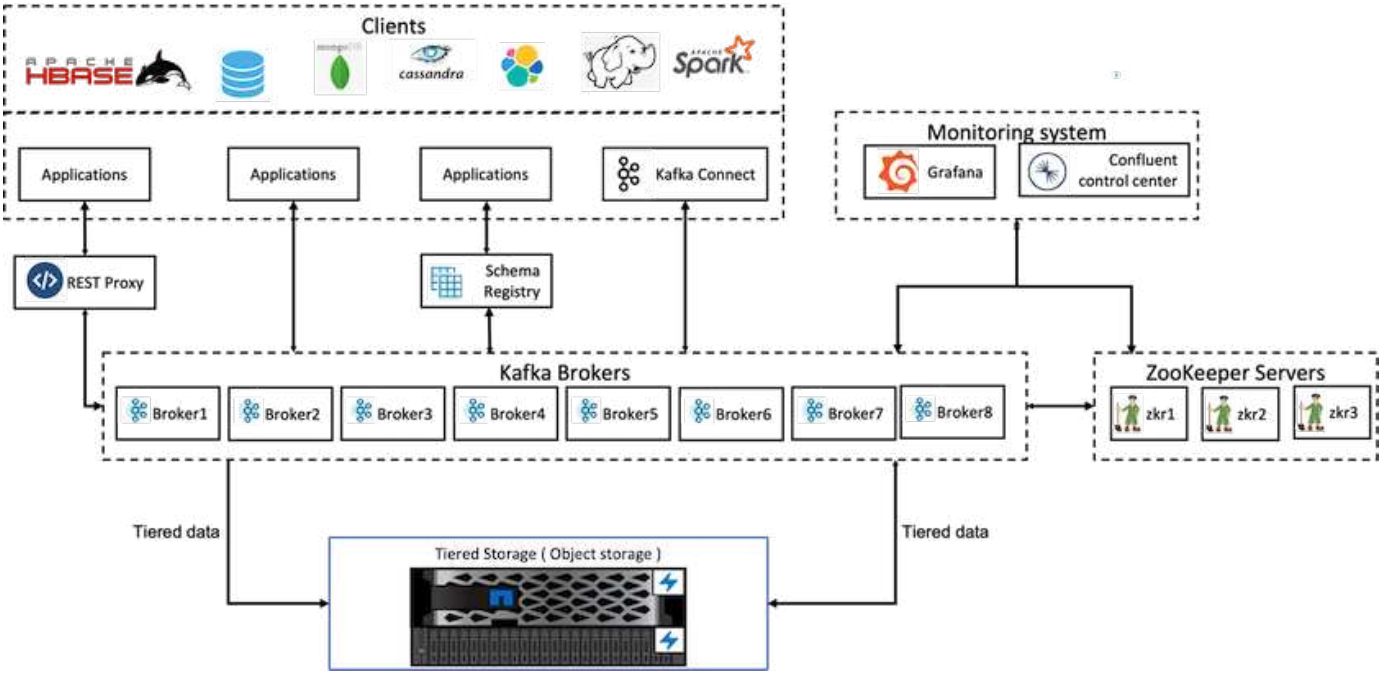
基本的には、データストレージとデータ処理を分離することで、各ストレージを個別に簡単に拡張できるようにすることが重要です。

業界をリードする革新的なテクノロジを搭載したNetApp ONTAP データ管理ソフトウェアは、データの保存場所にかかわらず、多様なメリットを提供します。

本ドキュメントでは、ティアストレージベンチマークキットを使用した、NetApp ONTAP 上の流暢なプラットフォームのパフォーマンスベンチマークについて説明します。

解決策

ONTAP を搭載したNetApp AFF A900ストレージコントローラは、データストリーム向けに設計された分散システムです。どちらも水平方向に拡張可能で、耐障害性に優れており、負荷時に優れたパフォーマンスを提供します。データ量を最小限に抑えるデータ削減テクノロジにより、分散型データストリーミングとストリーム処理を相互に補完し、ストレージコストを削減します。AFF A900ストレージコントローラは優れたパフォーマンスを提供すると同時に、コンピューティングリソースとデータストレージリソースを分離できます。これにより、システム管理が簡素化され、リソースを個別に拡張できます。



解決策アーキテクチャの詳細

このセクションでは、階層型ストレージ用のNetApp ONTAP を使用した、Conluent Platform環境でのパフォーマンス検証に使用するハードウェアとソフトウェアについて説明します。次の表に、解決策 のアーキテクチャと基本コンポーネントを示します。

プラットフォームコンポーネント	環境の構成
矛盾するプラットフォームバージョン6.2	• ゾーニングキーパー×3 • ブローカーサーバ×8 • 5台のツールサーバ • Grafana×1 • 1 xコントロールセンター
すべてのノード上のオペレーティングシステム	Linux （ Ubuntu 18.04 ）
NetApp ONTAP ：ウォームバケット用	• AFF A900ハイアベイラビリティ（HA）ペア×1 • 24 本、 800 本の SSD × 4 • S3 プロトコル • 100GbE
Fujitsu Primergy RX2540 サーバ × 15	• CPU×2、合計16個の物理コア • インテルXeon • 256GBの物理メモリ • 100GbEデュアルポート

テクノロジの概要

このセクションでは、この解決策で使用されるテクノロジーについて説明します。

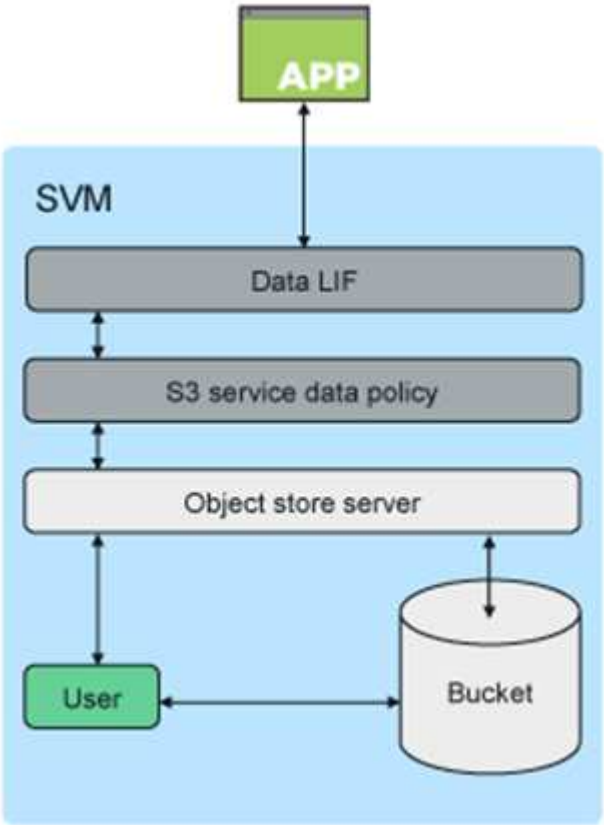
NetApp ONTAP ストレージコントローラ

NetApp ONTAP は、ハイパフォーマンスなエンタープライズクラスのストレージオペレーティングシステムです。

NetApp ONTAP 9.8では、Amazon Simple Storage Service（S3）APIがサポートされるようになりました。ONTAP は、Amazon Web Services（AWS）S3 APIアクションのサブセットをサポートしており、クラウドプロバイダ（AWS、Azure、GCP）とオンプレミスの両方のONTAPベースシステムでデータをオブジェクトとして表すことができます。

ネットアップのStorageGRID ソフトウェアは、オブジェクトストレージ向けの主力製品であるネットアップの解決策 です。ONTAP は、エッジ上での取り込み/前処理ポイントを提供することでStorageGRID を補完します。ネットアップが提供するオブジェクトデータ向けデータファブリックを拡張し、ネットアップ製品ポートフォリオの価値を高めます。

S3バケットへのアクセスは、許可されたユーザアプリケーションとクライアントアプリケーションを介して提供されます。次の図は、S3バケットにアクセスするアプリケーションを示しています。



主なユースケース

S3 APIをサポートする主な目的は、ONTAP でオブジェクトへのアクセスを提供することです。ONTAP ユニファイドストレージアーキテクチャで、ファイル（NFSおよびSMB）、ブロック（FCおよびiSCSI）、オブジェクト（S3）がサポートされるようになりました。

ネイティブ**S3**アプリケーション

S3を使用したオブジェクトアクセスにONTAP のサポートを利用できるアプリケーションが増えています。大容量のアーカイブワークロードには適していますが、ネイティブS3アプリケーションではハイパフォーマンスが急速に拡大しており、次のようなニーズがあります。

- 分析
- 人工知能
- エッジからコアへの取り込み
- 機械学習

ONTAP System Managerなど、使い慣れた管理ツールを使用して、ONTAP の開発や運用に使用できる高性能オブジェクトストレージを迅速にプロビジョニングできるようになりました。そのため、ONTAP のStorage Efficiency機能とセキュリティを活用できます。

FabricPool エンドポイント

ONTAP 9.8以降では、FabricPool でONTAP のバケットへの階層化がサポートされるため、ONTAP間で階層化できます。これは、既存のFAS インフラをオブジェクトストアのエンドポイントとして転用する場合に最適なオプションです。

FabricPool では、次の2つの方法でONTAP への階層化がサポートさ

- *ローカルクラスタ階層化。*非アクティブなデータは、クラスタLIFを使用してローカルクラスタにあるバケットに階層化されます。
- *リモートクラスタの階層化。*非アクティブなデータは、FabricPool クライアントのIC LIFとONTAP オブジェクトストアのデータLIFを使用して、従来のFabricPool クラウド階層と同様に、リモートクラスタにあるバケットに階層化されます。

ONTAP S3 は、ハードウェアや管理の追加なしで既存のクラスタの S3 機能を利用する場合に適しています。300TBを超える導入については、NetApp StorageGRID ソフトウェアを引き続き、オブジェクトストレージ向けの主要なネットアップ解決策 としてご利用いただけます。ONTAP またはStorageGRID をクラウド階層として使用する場合は、FabricPool ライセンスは必要ありません。

格納域の優れた階層化ストレージを実現する**NetApp ONTAP**

すべてのデータセンターで、ビジネスクリティカルなアプリケーションの稼働を維持し、重要なデータの可用性とセキュリティを確保する必要があります。新しいNetApp AFF A900システムは、ONTAP Enterprise Editionソフトウェアを搭載し、耐障害性に優れた設計を採用しています。ネットアップの新しい高速NVMeストレージシステムは、ミッションクリティカルな運用の中断をなくし、パフォーマンスの調整を最小限に抑え、ランサムウェア攻撃からデータを保護します。

初期導入から流暢なクラスタの拡張に至るまで、ビジネスクリティカルなアプリケーションに影響を伴わない変化に迅速に適応する必要があります。ONTAP のエンタープライズデータ管理、Quality of Service（QoS；サービス品質）、およびパフォーマンスにより、お客様の環境を計画して適合させることができます。

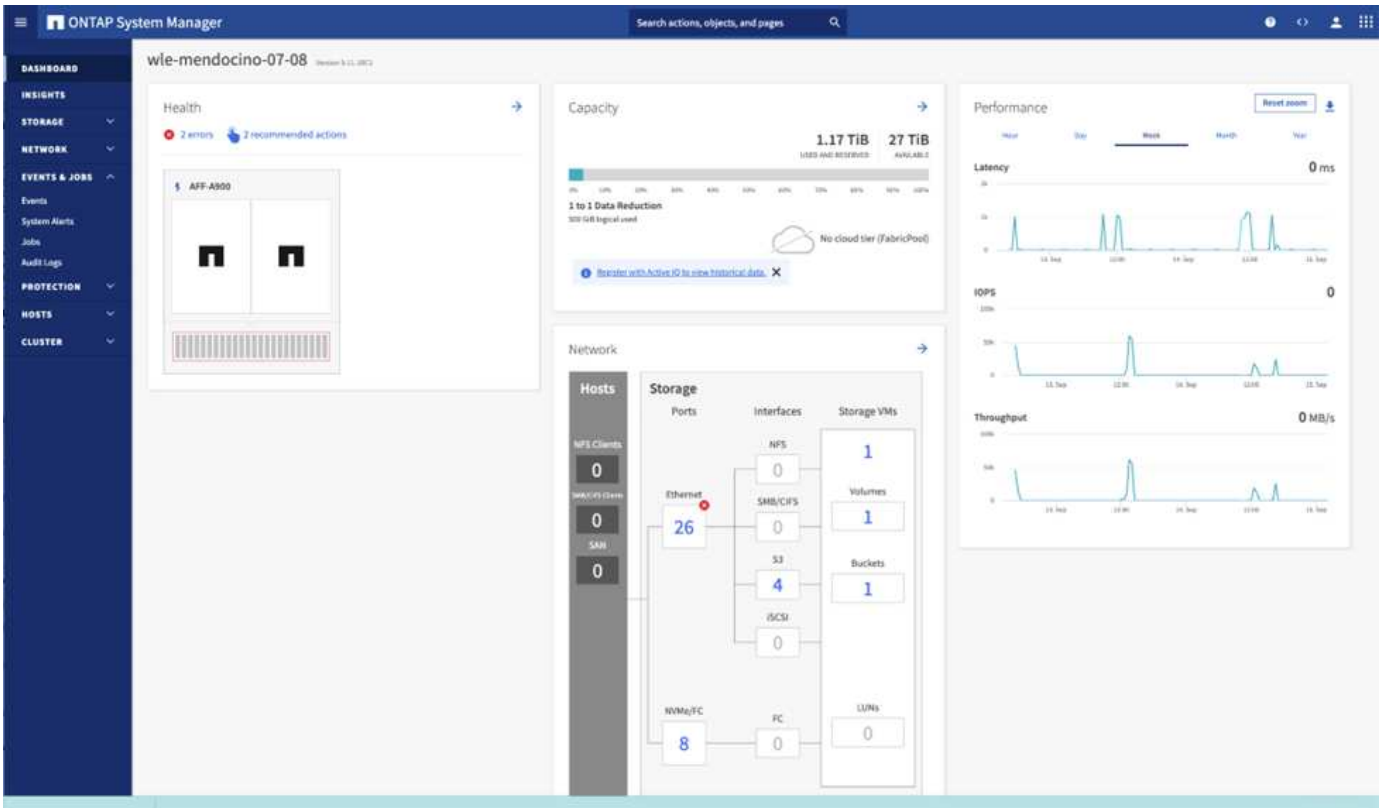
ネットアップのONTAP とConluent Tiered Storageを併用すると、スケールアウトストレージターゲットとしてONTAP を利用することでApache Kafkaクラスタの管理が簡素化され、流暢なコンピューティングリソースとストレージリソースを個別に拡張できます。

ONTAP S3サーバは、成熟したONTAP のスケールアウトストレージ機能を基盤としています。ONTAP クラ

スタをシームレスONTAP に拡張するには、新しく追加したノードを使用するようにS3バケットを拡張します。

ONTAP システムマネージャを使用してシンプルな管理を実現

ONTAP System Managerは、ブラウザベースのグラフィカルインターフェイスで、ONTAP ストレージコントローラの設定、管理、監視を、単一のコンソールから世界中に分散して実行できます。



ONTAP S3は、System ManagerおよびONTAP CLIを使用して設定および管理できます。System Managerを使用してS3を有効にしてバケットを作成する場合、ONTAP では、シンプルな設定を実現するためのデフォルトのベストプラクティスが提供されます。CLIからS3サーバとバケットを設定した場合は、必要に応じてSystem Managerでそれらを管理することも、その逆も可能です。

System Manager を使用して S3 バケットを作成すると、 ONTAP によって、システムで最も使用可能なパフォーマンスサービスレベルがデフォルトで設定されます。たとえば、AFF システムの場合、デフォルト設定は「最高レベル」です。パフォーマンスサービスレベルは、事前定義されたアダプティブQoSポリシーグループです。カスタムの QoS ポリシーグループを指定する場合は、デフォルトのサービスレベルのいずれかを指定する代わりに、ポリシーグループを指定しなくてもかまいません。

事前定義されたアダプティブQoSポリシーグループには次のものがあります。

- 最高レベル。低レイテンシと最高レベルのパフォーマンスを必要とするアプリケーションに使用されます。
- *パフォーマンス。*適度なパフォーマンスとレイテンシが求められるアプリケーションに使用します。
- *値。スループットと容量がレイテンシよりも重要なアプリケーションに使用。
- *カスタム。*カスタムのQoSポリシーを指定するか、QoSポリシーなしで指定します。

[階層化に使用する *] を選択した場合、パフォーマンスサービスレベルは選択されず、階層化データに最適な

パフォーマンスを備えた低コストのメディアを選択しようとします。

ONTAP は、選択したサービスレベルを満たす最も適切なディスクを含むローカル階層でこのバケットをプロビジョニングしようとします。ただし、バケットに含めるディスクを指定する必要がある場合は、 CLI でローカル階層（アグリゲート）を指定して S3 オブジェクトストレージを設定することを検討してください。CLI から S3 サーバを設定した場合も、必要に応じて System Manager で管理できます。

バケットに使用するアグリゲートを指定できるようにするには、 CLI を使用する必要があります。

矛盾する

Conflicent Platform は、データへのアクセス、保存、管理を継続的なリアルタイムストリームとして簡単に行うことができる、フルスケールのデータストリーミングプラットフォームです。ConFluent では、 Apache Kafka を作成した元のクリエイターが開発したサービスを利用して、 Kafka のメリットをエンタープライズクラスの機能で拡張しながら、 Kafka の管理や監視の負担を軽減することができます。現在、Fortune 100企業の80%以上がデータストリーミングテクノロジを採用しており、大部分が活用されています。

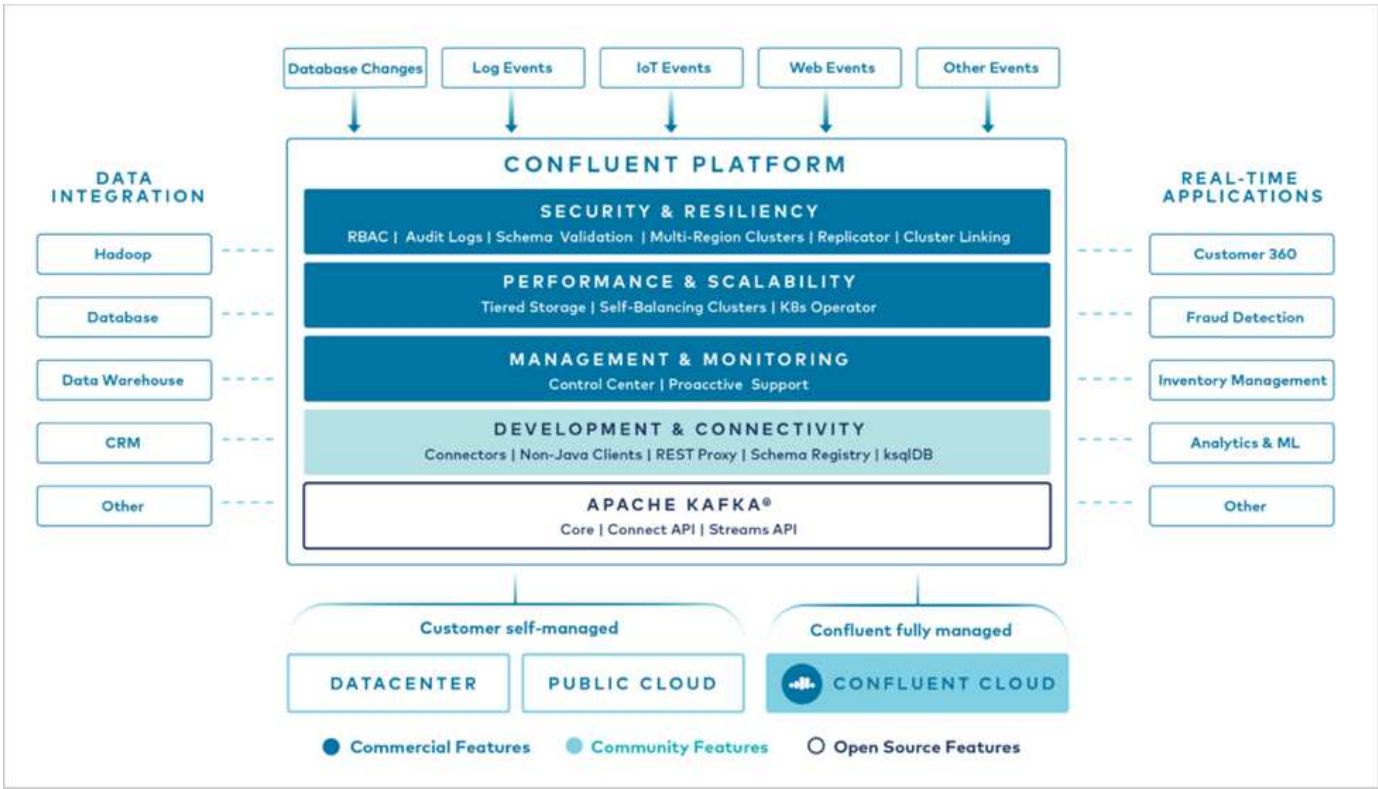
流暢な理由

履歴データとリアルタイムデータを一元化された単一の情報源に統合することで、 Conluent は、まったく新しいカテゴリの最新のイベント駆動型アプリケーションを簡単に構築し、ユニバーサルデータパイプラインを取得し、拡張性、パフォーマンス、信頼性を備えた強力な新しいユースケースを開放します。

流暢なものは何のために使用されるか。

Conflicent Platform を使用すると、データが異なるシステム間でどのように転送または統合されるかなど、基本的なメカニズムを気にすることなく、データからビジネス価値を引き出す方法に集中できます。具体的には、 Con裕福 なプラットフォームによって、 Kafka へのデータソースの接続やストリーミングアプリケーションの構築、 Kafka インフラの保護、監視、管理が簡易化されます。現在、Conluent Platformは、金融サービス、オムニチャネル小売、自律走行車から不正検出、マイクロサービス、IoTまで、さまざまな業界で幅広く使用されています。

次の図は、流暢なプラットフォームのコンポーネントを示しています。



流暢なイベントストリーミング技術の概要

流暢なプラットフォームの中核はです "カフカ"最も人気の高いオープンソース分散ストリーミングプラットフォームです。Kafkaの主な機能は次のとおりです。

- レコードのストリームをパブリッシュしてサブスクライブします。
- レコードのストリームをフォールトトレラントな方法で保存します。
- レコードのストリームを処理します。

Conluent Platform には Schema Registry 、 REST Proxy 、合計 100 以上の Kafka コネクタ、および ksqlDB も含まれています。

流暢なプラットフォームのエンタープライズ機能の概要

- * ConFluent Control Center * Kafkaの管理と監視を行うためのUIベースのシステムです。Kafka Connect の管理や、他のシステムとの接続の作成、編集、管理を簡単に行うことができます。
- * Kubernetes には流暢な言葉があります。 * Kubernetes の流暢な言葉は Kubernetes のオペレータです。Kubernetes の運用担当者は、特定のプラットフォームアプリケーションに固有の機能と要件を提供することで、 Kubernetes のオーケストレーション機能を拡張します。Con裕福 なプラットフォームの場合は、 Kubernetes での Kafka の導入プロセスを大幅に簡易化し、一般的なインフラのライフサイクルタスクを自動化します。
- * Kafka Connectコネクタ。*コネクタはKafka Connect APIを使用して、Kafkaと他のシステム（データベース、キーバリューストア、検索インデックス、ファイルシステムなど）との接続に使用します。Confluent Hub には、一般的なデータソースおよびシンク用のダウンロード可能なコネクタがあります。これには、 Conluent Platform でこれらのコネクタの完全なテストとサポートされたバージョンが含まれます。詳細については、を参照してください "こちらをご覧ください"。
- * セルフバランシングクラスタ。 * 自動ロードバランシング、障害検出、自己修復機能を提供します。ま

た、必要に応じてブローカーの追加や運用停止も可能で、手動での調整は不要です。

- * クラスタを直接接続し、リンクブリッジを介して 1 つのクラスタから別のクラスタにトピックをミラーリングします。クラスタリンクにより、マルチデータセンター、マルチクラスタ、ハイブリッドクラウドの導入を簡易化できます。
- *流暢な自動データバランサ。*ブローカーの数、パーティションのサイズ、パーティションの数、およびクラスタ内のリーダーの数について、クラスタを監視します。これにより、データを移動してクラスタ全体で均等なワークロードを作成しながら、トラフィックのリバランシングを調整して、リバランシング中の本番ワークロードへの影響を最小限に抑えることができます。
- * 流暢なリプリケータ * により、複数のデータセンターで複数の Kafka クラスターを容易に保守できます。
- * 階層化ストレージ。 * 任意のクラウドプロバイダを使用して大量の Kafka データを保存するオプションを提供し、運用上の負担とコストを削減します。階層型ストレージでは、コスト効率に優れたオブジェクトストレージにデータを格納し、ブローカーを拡張するために、必要なコンピューティングリソースが増えた場合のみデータを利用できます。
- * Conluent JMS Client. * Conluent Platform には Kafka 用の JMS 対応クライアントが含まれています。Kafka クライアントは、 Kafka ブローカーをバックエンドとして使用して、 JMS 1.1 標準 API を実装しています。これは 'JMS を使用するレガシーアプリケーションがあり ' 既存の JMS メッセージブローカを Kafka に置き換える場合に便利です
- * Coneluent MQTT プロキシ * を使用すると、 MQTT デバイスやゲートウェイから Kafka に直接データを公開できます。 MQTT ブローカーは必要ありません。
- * 流暢なセキュリティプラグイン。 * 流暢なセキュリティプラグインは、各種の流暢なプラットフォームツールや製品にセキュリティ機能を追加するために使用されます。現在、 Conluent REST プロキシ用のプラグインが用意されており、受信要求の認証に役立ち、認証されたプリンシパルを要求に Kafka に伝播できます。これにより、 Con裕福 な REST プロキシクライアントでは、 Kafka ブローカーのマルチテナントセキュリティ機能を利用できます。

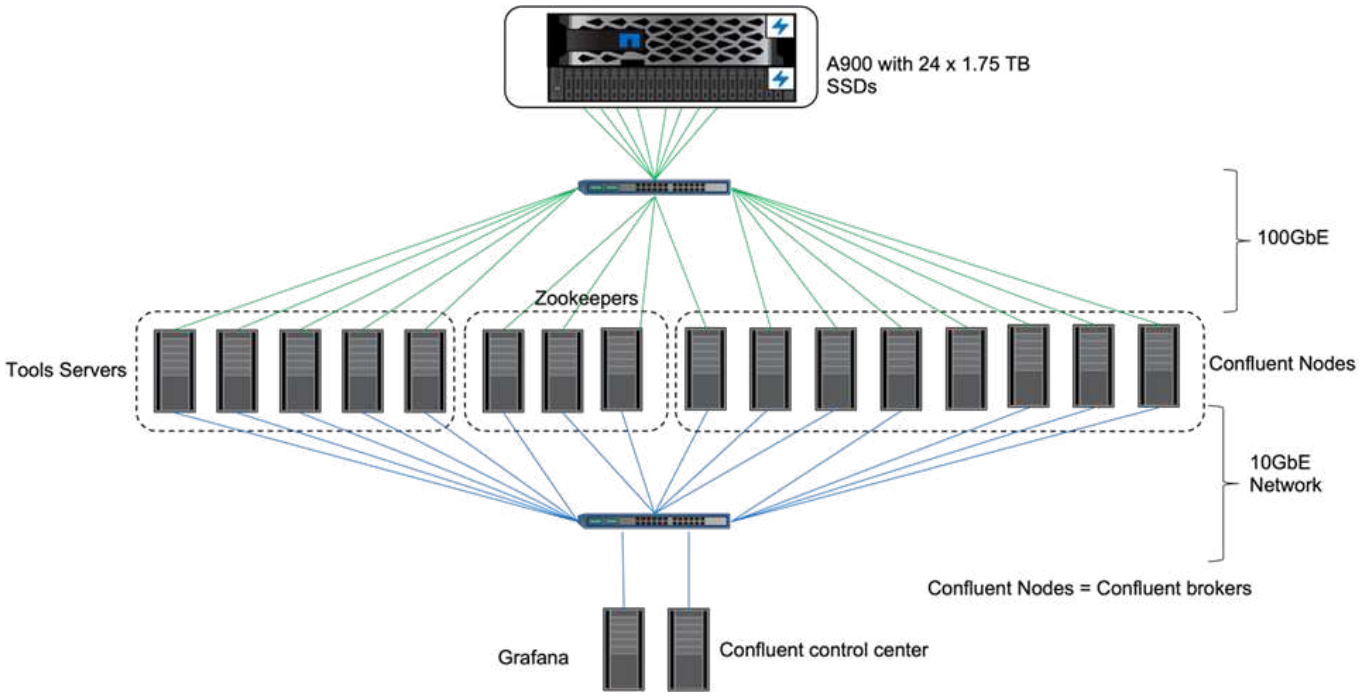
競合する性能検証

この検証は、NetApp ONTAP 上の階層型ストレージについて、流暢なプラットフォームで実施しました。ネットアップと流暢なチームがこの検証に協力し、必要なテストケースを実施しました。

矛盾する設定です

セットアップには、3台の動物園、5台のブローカー、256GBのRAMと16個のCPUを搭載した5台のテストサーバを使用しました。ネットアップストレージには、AFF A900 HAペアでONTAP を使用しました。ストレージとブローカーは、100GbE接続経由で接続されています。

次の図に、階層型ストレージの検証に使用する構成のネットワークトポロジを示します。



ツールサーバは、Conluentノードとの間でイベントを送受信するアプリケーションクライアントとして機能します。

競合する階層型ストレージ構成

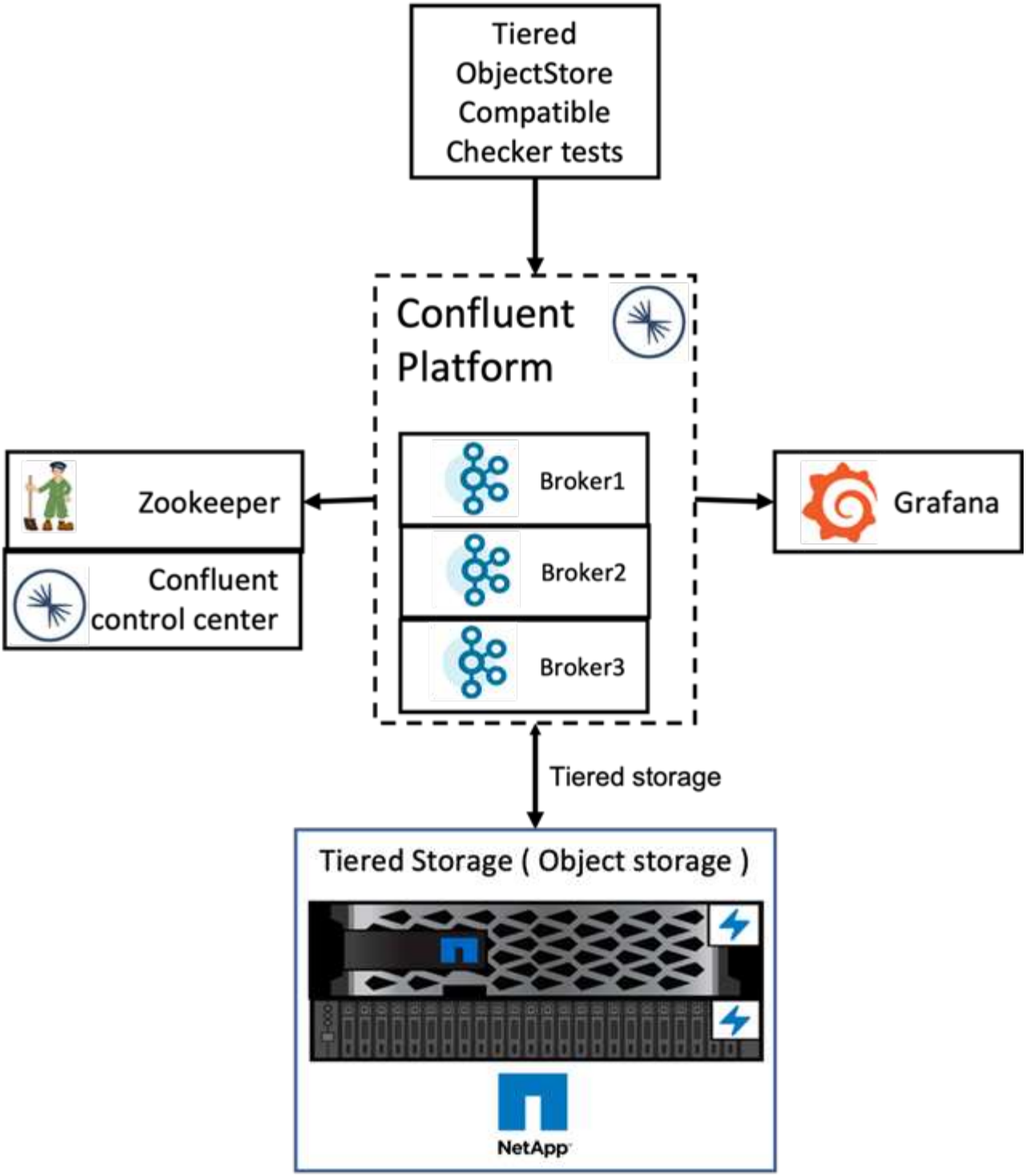
テストには次のパラメータを使用しました。

```
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
```

検証にはHTTPプロトコルにONTAP を使用しましたが、HTTPSも使用しました。アクセスキーとシークレットキーは、「 confliclus.tir.s3.cred.file.path 」パラメータで指定したファイル名に格納されます。

ネットアップストレージコントローラ**–ONTAP**

ONTAP では、検証用に単一のHAペア構成を設定しました。



検証結果

以下の 5 つの検証ケースを完了しました。最初の 2 つは機能テストで、残りの 3 つはパフォーマンステストです。

オブジェクトストアの正確性テスト

このテストでは、API呼び出しを使用して階層化ストレージに使用されるオブジェクトストアで、GET、PUT、DELETEなどの基本的な処理を実行します。

階層化機能の正確性テスト

このテストでは、オブジェクトストレージのエンドツーエンド機能をチェックします。トピックを作成し、新たに作成されたトピックにイベントストリームを生成し、ブローカーがオブジェクトストレージにセグメントをアーカイブするのを待機し、イベントストリームを消費して、消費されたストリームが生成されたストリームと一致するかどうかを検証します。このテストは、オブジェクトストアフォールト挿入の有無にかかわらず実施しました。ONTAP のいずれかのノードでサービスマネージャサービスを停止し、エンドツーエンド機能がオブジェクトストレージで機能することを検証することで、ノード障害をシミュレートしました。

ティアフェッチベンチマーク

このテストでは、階層型オブジェクトストレージの読み取りパフォーマンスを検証し、ベンチマークによって生成されたセグメントからの負荷が大きい範囲での読み取り要求のフェッチをチェックしました。このベンチマークでは、 Conluent 社は階層フェッチ要求に対応するカスタムクライアントを開発しました。

ワークロードジェネレータを消費

このテストでは、セグメントをアーカイブすることにより、オブジェクトストアへの書き込みワークロードを間接的に生成します。コンシューマグループがセグメントを取得すると、読み取りワークロード（セグメント読み取り）がオブジェクトストレージから生成されました。このワークロードはTOCCスクリプトによって生成されました。このテストでは、並列スレッドでのオブジェクトストレージの読み取りと書き込みのパフォーマンスをチェックしました。階層化機能の正確性テストと同様に、オブジェクトストアフォールト挿入を使用したテストと使用しなかったテストを実施しました。

保持ワークロードジェネレータ

このテストでは、トピックの保持ワークロードが多い場合のオブジェクトストレージの削除パフォーマンスを確認しました。保存ワークロードは、テストトピックと並行して多数のメッセージを生成するTOCCスクリプトを使用して生成されました。テストトピックでは、サイズベースおよび時間ベースの強力な保持設定を使用してイベントストリームをオブジェクトストアから継続的にパージするように設定しました。その後、セグメントがアーカイブされました。その結果、ブローカーによるオブジェクトストレージの削除や、オブジェクトストアの削除処理のパフォーマンス収集が行われ、多くの削除が発生していました。

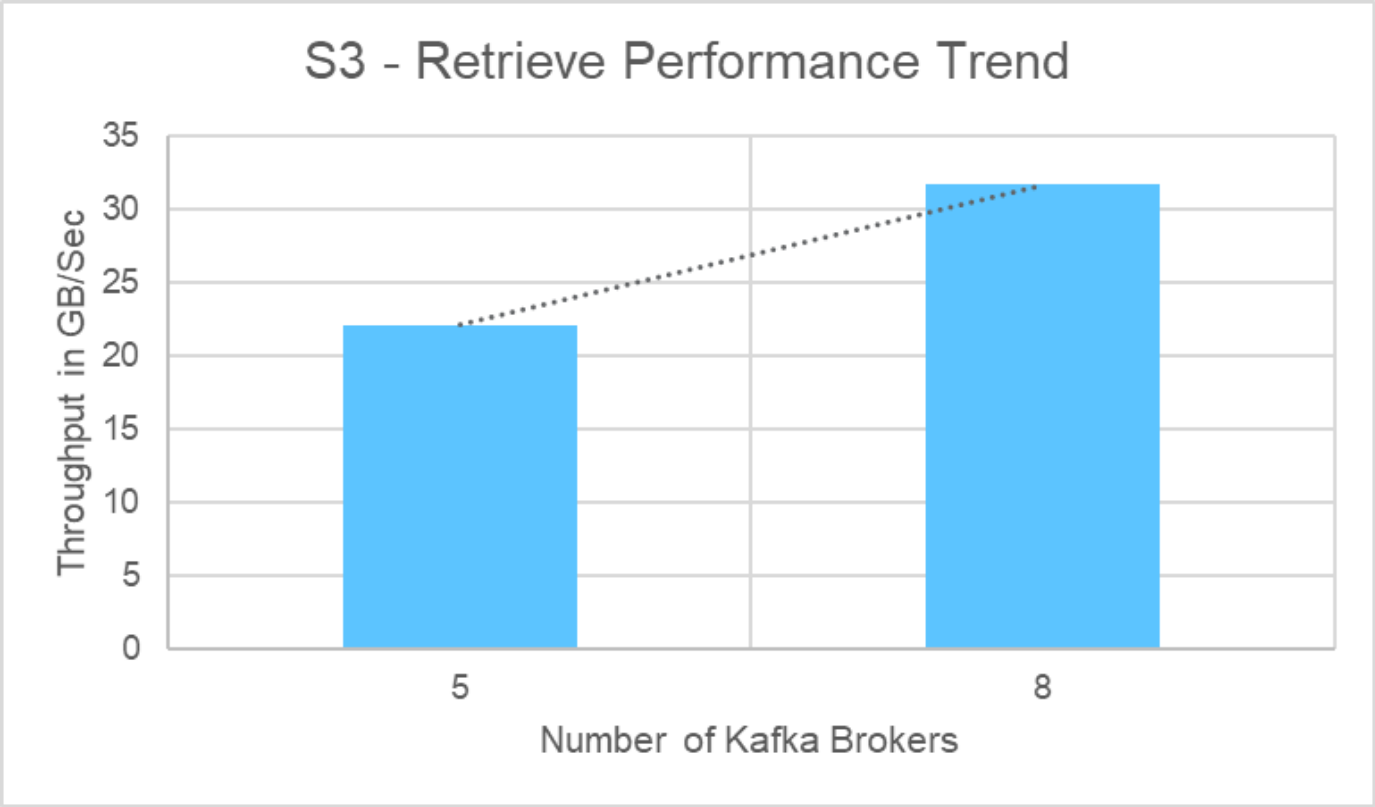
検証の詳細については、を参照してください "矛盾する" Webサイト。

ワークロードジェネレータを使用したパフォーマンステスト

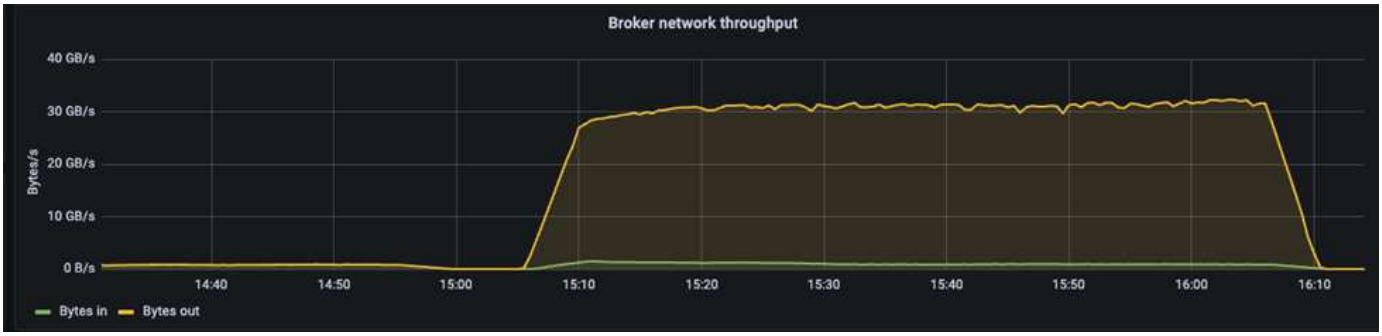
NetAppストレージ・コントローラ1台のAFF A900を使用して’ワークロードの生成中に5ノードまたは8ノードのブローカ・ノードを使用して階層型ストレージのテストを実施しました今回のテストでは、AFF A900のリソース使用率が100%に達するまで、ブローカーノードの数に合わせて完了までの時間とパフォーマンスの結果が拡張されました。ONTAP ストレージコントローラには、少なくとも1つのHAペアが必要です。

S3読み出し処理のパフォーマンスは、ブローカーノードの数に基づいてリニアに向上しました。ONTAP ストレージコントローラを使用すると、1つの環境で最大12個のHAペアをサポートできます。

次のグラフは、S3の階層化トラフィックと5ノードまたは8ノードのブローカーを組み合わせたものです。AFF A900の単一HAペアのパフォーマンスを最大化しました。



次のグラフは、Kafkaのスループットを約31.74GBpsで示しています。



また、ONTAP ストレージコントローラの「perfstat」レポートでも同様のスループットが確認されました。

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

パフォーマンスのベストプラクティスのガイドライン

このページでは、この解決策 のパフォーマンスを向上させるためのベストプラクティスについて説明します。

- ONTAP の場合は、可能なかぎりGETサイズ>= 1MBを使用します。
- ブローカ・ノードでserver.properties`にあるnum.network.threads`と’num.io.threadsを増やすと’階層化アクティビティの増加をS3階層にプッシュできますこれらの結果は’num.network.threads`と’num.io.threads'32に設定されています

- S3バケットは、メンバーアグリゲートごとに8つのコンスティチュエントをターゲットとする
- S3トラフィックを伝送するイーサネットリンクでは、ストレージとクライアントの両方で可能な場合、MTU 9、000を使用する必要があります。

まとめ

この検証テストは、NetApp ONTAP ストレージコントローラを使用した流暢なティアのスループットが31.74GBpsに達しました。

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- 流暢とは

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

- S3 シンクパラメータの詳細

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

- Apache Kafka です

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

- ONTAP におけるS3のベストプラクティスです

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

- S3オブジェクトストレージの管理

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

- ネットアップの製品マニュアル

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

ネットアップの **Apache Spark** 向けストレージソリューション

TR-4570：『NetApp Storage Solutions for Apache Spark：Architecture、Use Cases、Performance Results』

ネットアップKarthikeyan Nagalingam、Rick Huang氏

本ドキュメントでは、Apache Sparkのアーキテクチャ、お客様のユースケース、ビッグデータ分析と人工知能（AI）に関連するネットアップストレージポートフォリオについて説明します。また、一般的なHadoopシステムに対して業界標準のAI、機械学習（ML

）、ディープラーニング（DL）ツールを使用してさまざまなテスト結果を提示することで、適切なSpark解決策 を選択できます。まず、Sparkアーキテクチャ、適切なコンポーネント、2つの配置モード（クラスタとクライアント）が必要です。

このドキュメントでは、構成の問題に対処するためのユースケースも紹介しています。また、Sparkでのビッグデータ分析、AI、ML、DLに関連するネットアップのストレージポートフォリオの概要についても説明しています。その後、Spark固有のユースケースとNetApp Sparkの解決策 ポートフォリオから得られたテスト結果をご紹介します。

お客様の課題

このセクションでは、小売、デジタルマーケティング、銀行業務、ディスクリート製造、プロセス製造などのデータ成長業界におけるビッグデータ分析とAI / ML / DLに関するお客様の課題に焦点を当てます。 政府機関やプロフェッショナルサービスも利用できます。

予測不可能なパフォーマンス

従来型のHadoop導入では、一般にコモディティハードウェアを使用します。パフォーマンスを向上させるには、ネットワーク、オペレーティングシステム、Hadoopクラスタ、Sparkなどのエコシステムコンポーネント、ハードウェアをチューニングする必要があります。それぞれのレイヤを調整しても、Hadoopは環境内のハイパフォーマンスを想定して設計されていないコモディティハードウェア上で実行されるため、必要なパフォーマンスレベルを達成することは困難です。

メディアおよびノードの障害

通常の状態でも、コモディティハードウェアは障害が発生しやすくなります。データノードの1つのディスクに障害が発生すると、Hadoopマスターはデフォルトでそのノードを正常な状態ではないとみなします。次に、レプリカから正常なノードに、そのノードからネットワーク経由で特定のデータをコピーします。このプロセスにより、Hadoopジョブのネットワークパケット速度が低下します。正常な状態に戻ったら、クラスタでデータを再度コピーしてレプリケートデータを削除する必要があります。

Hadoopベンダーロックイン

Hadoopディストリビュータは独自のバージョン管理機能を備えたHadoopディストリビューションを所有しており、これによってお客様はこれらのディストリビューションにロックされます。ただし、多くのお客様が、特定のHadoopディストリビューションにお客様を結び付けることのないインメモリ分析をサポートしている必要があります。ディストリビューションを自由に変更し、分析機能を活用する必要があります。

複数の言語をサポートしていない

MapReduce Javaプログラムに加えて、ジョブを実行するために複数の言語のサポートが必要になることがよくあります。SQLやスクリプトなどのオプションを使用すると、回答を取得する柔軟性が高まり、データを整理して取得するためのオプションが増え、データを分析フレームワークにすばやく移動できます。

使用の難しさ

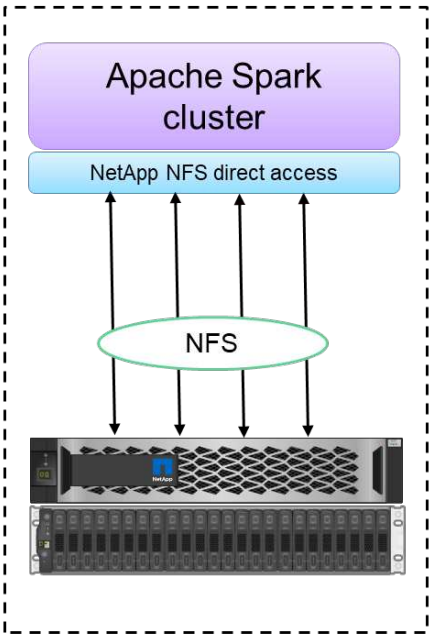
しばらくの間、Hadoopが使いにくいとユーザからクレームを受けています。Hadoopは新しいバージョンが登場するたびにシンプルかつ強力になりましたが、今もこの批評家は存続しています。Hadoopを使用するには、JavaおよびMapReduceのプログラミングパターンを理解する必要があります。これは、データベース管理者や、従来のスクリプトスキルセットを持つユーザにとっての課題です。

企業のAIチームは、さまざまな課題に直面します。導入エコシステムやアプリケーションごとに専門的なデータサイエンスの知識があるとしても、ツールやフレームワークは単に相互に変換されるわけではありません。データサイエンスプラットフォームは、Spark上に構築された対応するビッグデータプラットフォームとシームレスに統合する必要があります。データ移動が容易で、再利用可能なモデル、すぐに使えるコード、プロトタイプ作成、検証、バージョン管理、共有、再利用といったベストプラクティスに対応するツールを備えています。 また、モデルを迅速に本番環境に導入できます。

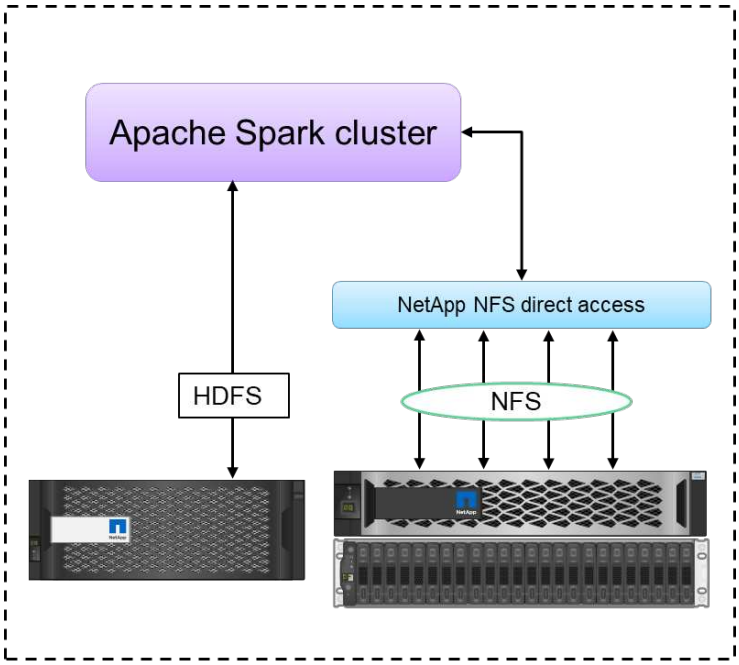
ネットアップを選ぶ理由

ネットアップでは、次の方法でSpark体験を向上させています。

- NetApp NFSから直接アクセスする（次の図を参照）ことで、データを移動したりコピーしたりすることなく、既存または新規のNFSv3 / NFSv4データに対してビッグデータ分析ジョブを実行できます。データの複数のコピーが作成されるため、ソースとデータを同期する必要がありません。
- ストレージ効率が向上し、サーバのレプリケーションが不要になります。たとえば、NetApp EシリーズHadoop解決策 にはデータのレプリカが3つではなく2つ必要であり、FAS Hadoop解決策 にはデータソースが必要ですが、データのレプリケーションやコピーは必要ありません。ネットアップのストレージソリューションは、サーバ間のトラフィックも削減します。
- ドライブおよびノードの障害時のHadoopジョブとクラスタの動作が向上します。
- データ取り込みのパフォーマンスが向上します。



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

たとえば、金融機関や医療機関では、ある場所から別の場所へデータを移動する際に法的義務を果たす必要がありますが、これは容易な作業ではありません。このシナリオでは、NetApp NFSから直接アクセスして、財務データと医療データを元の場所から分析します。もう1つの主なメリットは、NetApp NFS直接アクセスを使用することで、ネイティブのHadoopコマンドを使用してHadoopデータを簡単に保護できるようになることと、ネットアップの充実したデータ管理ポートフォリオでデータ保護ワークフローを実現できることです。

NetApp NFS直接アクセスでは、HadoopクラスタとSparkクラスタに対して次の2種類の導入オプションを提

供しています。

- デフォルトでは、HadoopクラスタまたはSparkクラスタは、データストレージとデフォルトのファイルシステムにHadoop Distributed File System（HDFS；Hadoop分散ファイルシステム）を使用しています。NetApp NFSから直接アクセスできるため、デフォルトのHDFSをデフォルトのファイルシステムとしてNFSストレージに置き換えることができ、NFSデータを直接分析できます。
- もう1つの導入オプションでは、NetApp NFS直接アクセスを使用して、1つのHadoopクラスタまたはSparkクラスタ内でHDFSを追加のストレージとして構成することもできます。この場合、 NFS エクスポートを介してデータを共有し、 HDFS データと同じクラスタからデータにアクセスできます。

NetApp NFS直接アクセスを使用する主な利点は次のとおりです。

- 現在の場所からデータを分析することで、分析データをHDFSなどのHadoopインフラに移動するための時間とパフォーマンスのかかる作業を回避できます。
- レプリカの数を3分の1から1に削減。
- ユーザを分離してコンピューティングとストレージを分離し、個別に拡張
- ONTAP の充実したデータ管理機能を活用して、エンタープライズデータを保護
- Hortonworksデータプラットフォームの認定。
- ハイブリッドデータ分析の導入を実現
- 動的なマルチスレッド機能を活用してバックアップ時間を短縮

を参照してください "TR-4657 ：ネットアップのハイブリッドクラウドデータソリューション - Spark と Hadoop はお客様のユースケースに基づいています" Hadoopデータのバックアップ、クラウドからオンプレミスへのバックアップ、ディザスタリカバリ、既存のHadoopデータに対するDevTestの有効化、データ保護とマルチクラウド接続の実現、分析ワークロードの高速化を実現します。

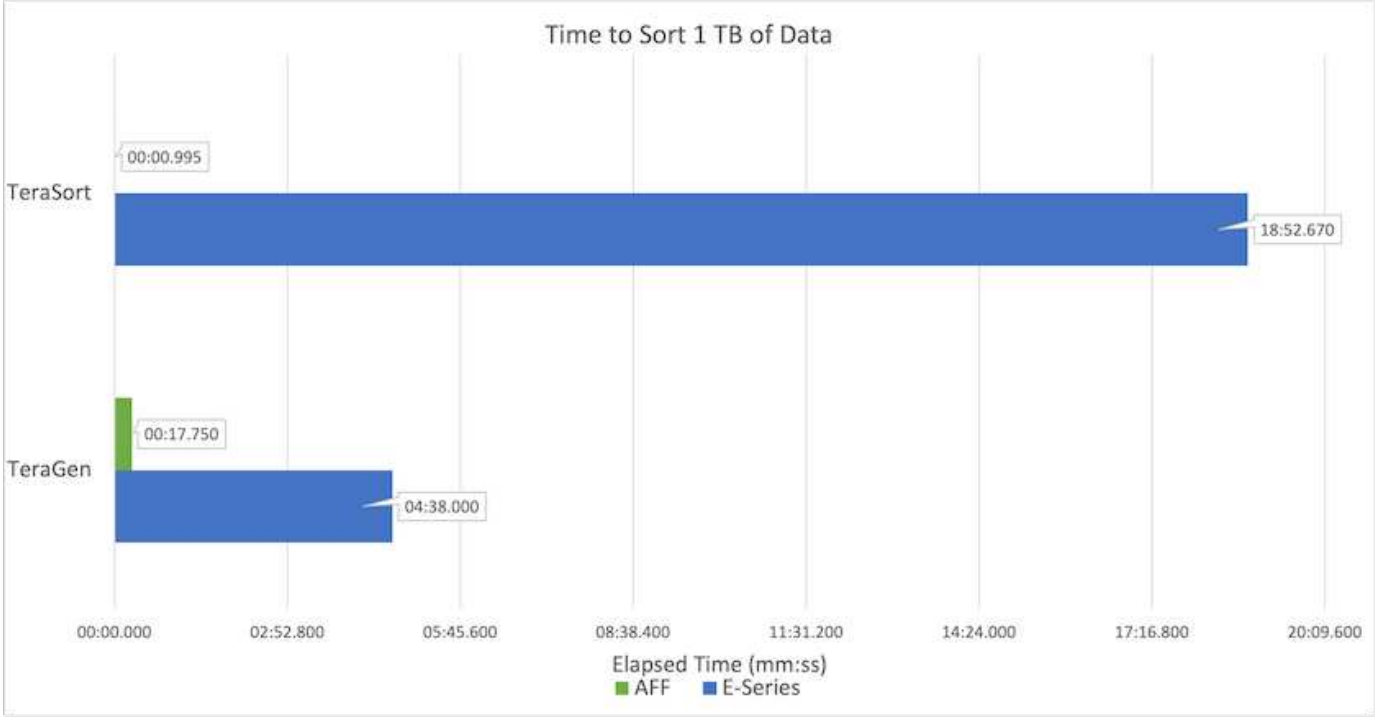
次のセクションでは、Sparkのお客様にとって重要なストレージ機能について説明します。

ストレージ階層化

Hadoopストレージ階層化を使用すると、ストレージポリシーに従ってさまざまなタイプのファイルを格納できます。ストレージ・タイプには' hot "cold "warm "all_sssd "one _sssd 'が含まれます 「lazy_persist`」。

ネットアップのAFF ストレージコントローラと、ストレージポリシーが異なるSSDおよびSASドライブを搭載したEシリーズストレージコントローラでHadoopストレージの階層化を検証しました。AFF-A800のSparkクラスタには4つのコンピュートワーカーノードがあり、Eシリーズのクラスタには8つのノードがあります。主な用途は、ソリッドステートドライブ（SSD）とハードドライブディスク（HDD）のパフォーマンスを比較することです。

次の図に、ネットアップのHadoop SSD向けソリューションのパフォーマンスを示します。



- ベースラインNL-SAS構成では、8つのコンピューティングノードと96本のNL-SASドライブを使用しました。この構成では、1TBのデータが4分38秒で生成され、 を参照してください "TR-3969『NetApp Eシリーズ解決策 for Hadoop』" クラスタとストレージ構成の詳細については、を参照してください。
- TeraGenを使用した場合、SSD構成ではNL-SAS構成よりも1TBのデータが15.66x高速で生成されました。さらに、SSD構成では、コンピューティングノードの半分とディスクドライブの半分（合計24本のSSDドライブ）が使用されていました。ジョブの完了時間に基づき、NL-SAS構成の約2倍の速さで処理されました。
- SSD構成は、TeraSortを使用してNL-SAS構成の1TBのデータを1138.36倍高速にソートしました。さらに、SSD構成では、コンピューティングノードの半分とディスクドライブの半分（合計24本のSSDドライブ）が使用されていました。そのため、ドライブあたりの速度は、NL-SAS構成の約3倍です。
- ここで重要なのは、回転式ディスクからオールフラッシュに移行することでパフォーマンスを向上させることです。コンピューティングノードの数がボトルネックになっていません。ネットアップのオールフラッシュストレージなら、ランタイムのパフォーマンスを大幅に向上できます。
- NFSでは、データがすべてプールされる機能と同等で、ワークロードに応じてコンピューティングノードの数を減らすことができました。コンピューティングノードの数を変更した場合、Apache Sparkクラスタユーザは手動でデータをリバランシングする必要がありません。

パフォーマンスの拡張-スケールアウト

AFF 解決策 のHadoopクラスタの処理能力を強化する必要がある場合は、適切な数のストレージコントローラを使用してデータノードを追加できます。ワークロードの特性に応じて、ストレージコントローラアレイごとにデータノードを4つから始めて、ストレージコントローラごとにデータノードを8つに増やすことを推奨します。

AFF とFAS はインプレース分析に最適です。コンピューティング要件に基づいて、ノードマネージャを追加できます。また、ノンストップオペレーション機能により、ダウンタイムなしでストレージコントローラをオンデマンドで追加できます。AFF とFAS を備えた豊富な機能を備えており、NVMeメディアのサポート、効率性の保証、データ削減、QoS、予測分析、 クラウドの階層化、レプリケーション、クラウドの導入、セキュリティお客様が要件を満たせるように、ネットアップでは、追加のライセンスコストなしでファイルシステム分析、クォータ、オンボックスロードバランシングなどの機能を提供しています。ネットアップは、同時ジ

ョブ数やレイテンシの低減、処理の簡易化、1秒あたりのスループットの向上といった、競合他社よりも優れたパフォーマンスを提供しています。さらに、ネットアップのCloud Volumes ONTAP は、主要な3つのクラウドプロバイダすべてで動作します。

パフォーマンスの拡張-スケールアップ

ストレージ容量を追加する必要がある場合は、スケールアップ機能を使用して、AFF 、FAS 、およびEシリーズシステムにディスクドライブを追加できます。Cloud Volumes ONTAP を使用してストレージをPBレベルに拡張するには、使用頻度の低いデータをブロックストレージからオブジェクトストレージに階層化し、追加のコンピューティングなしでCloud Volumes ONTAP ライセンスをスタックするという2つの要素を組み合わせます。

複数のプロトコル

ネットアップシステムは、SAS、iSCSI、FCP、InfiniBandなど、Hadoop導入のほとんどのプロトコルをサポートしています。 およびNFSが必要です。

運用およびサポートされるソリューション

本ドキュメントに記載されているHadoopソリューションは、ネットアップによってサポートされています。これらのソリューションは、主要なHadoopディストリビュータでも認定されています。詳細については、を参照してください "MapR" サイト、 "Hortonworks" サイト、Cloudera "認定資格" および "パートナー" サイト：

対象読者

分析とデータサイエンスの世界は、ITとビジネスのさまざまな分野に影響を与えています。

- データサイエンティストは、選択したツールとライブラリを柔軟に使用できる必要があります。
- データエンジニアは、データの流れと配置場所を把握する必要があります。
- DevOpsエンジニアは、新しいAIアプリケーションやMLアプリケーションをCIパイプラインやCDパイプラインに統合するためのツールを必要としています。
- クラウド管理者とアーキテクトは、ハイブリッドクラウドリソースを設定し、管理できる必要があります。
- ビジネスユーザは、分析、AI、ML、DLアプリケーションにアクセスしたいと考えています。

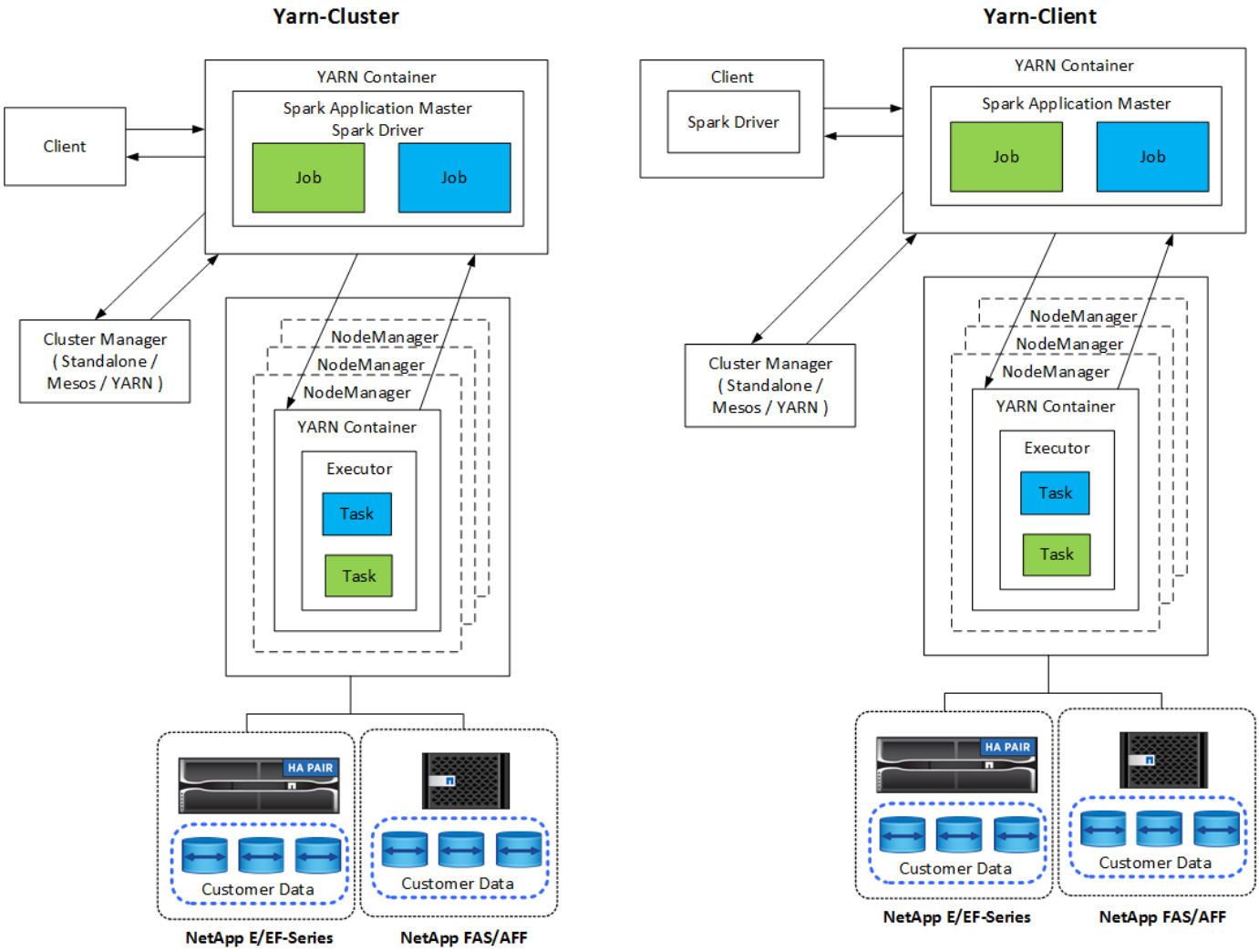
このテクニカルレポートでは、NetApp AFF 、Eシリーズ、StorageGRID 、NFSへの直接アクセス、Apache Spark HorovodとKerasは、これらの役割のそれぞれがビジネスに価値をもたらすのを支援します。

解決策テクノロジ

Apache Sparkは、Hadoop Distributed File System（HDFS）と直接連携するHadoopアプリケーションを作成するための一般的なプログラミングフレームワークです。Sparkは本番環境に対応しており、ストリーミングデータの処理をサポートしています。MapReduceよりも高速です。Sparkには、効率的なイテレーションのためのメモリ内データキャッシングが設定可能であり、Sparkシェルはデータの学習と探索のためにインタラクティブです。Sparkでは、Python、Scala、Javaでアプリケーションを作成できます。Sparkアプリケーションは、1つ以上のタスクを持つ1つ以上のジョブで構成され

ています。

すべてのSparkアプリケーションにはSparkドライバがあります。yarn -クライアントモードでは、ドライバはクライアント上でローカルに実行されます。Yarn -クラスタモードでは、ドライバはアプリケーションマスター上のクラスタで実行されます。クラスタモードでは、クライアントが切断してもアプリケーションは引き続き実行されます。



クラスタマネージャは3種類あります。

- *スタンドアロン。*このマネージャーはSparkの一部であり、クラスタのセットアップが簡単です。
- * Apache Mesos.* MapReduceなどのアプリケーションも実行される一般的なクラスタマネージャです。
- * Hadoop糸。*これはHadoop 3のリソースマネージャーです。

レジリエントな分散データセット（RDD）はSparkの主要コンポーネントです。RDDは、クラスタ内のメモリに格納されているデータから、失われたデータや欠落しているデータを再作成し、ファイルから取得した初期データまたはプログラムによって作成された初期データを格納します。RDDは、ファイル、メモリ内のデータ、または別のRDDから作成されます。Sparkプログラミングは、変換とアクションという2つの操作を実行します。既存のRDDに基づいて新しいRDDが作成されます。アクションはRDDから値を返します。

変換とアクションはSparkのデータセットとDataFramesにも適用されます。データセットとは、分散されたデータの集合であり、Spark SQLの最適化された実行エンジンの利点とともに、RDDの利点（強力なタイピン

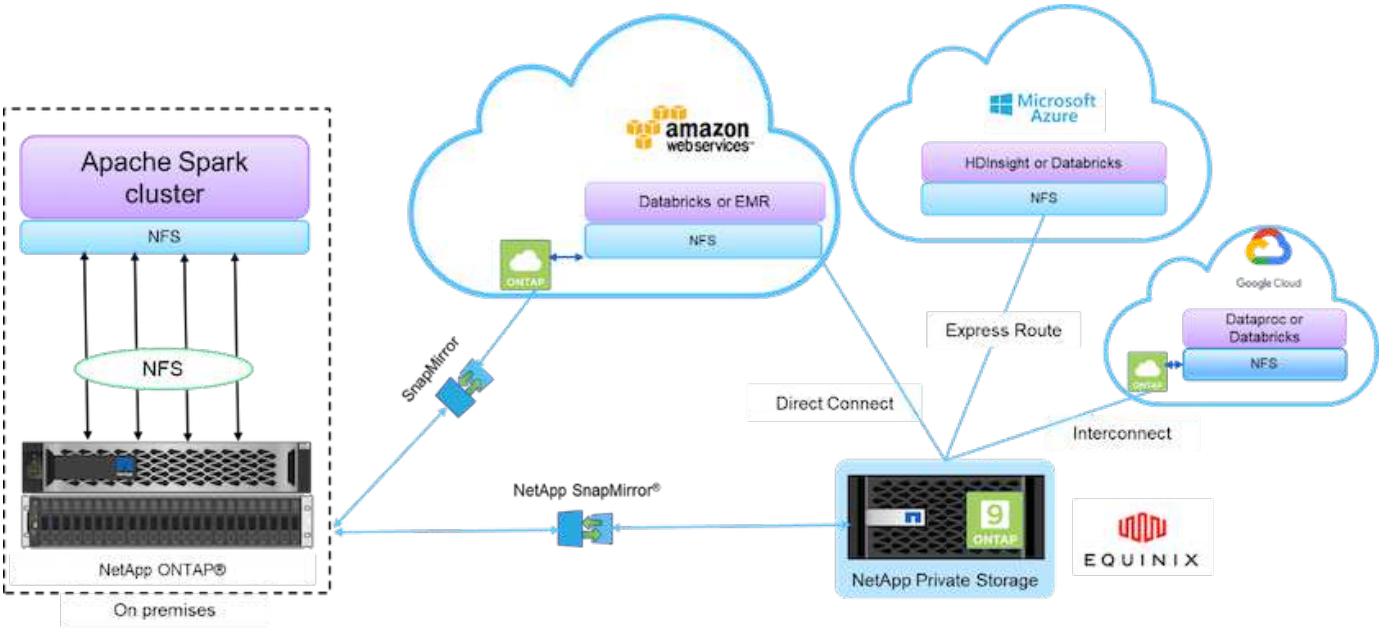
グ、ラムダ関数の使用）を提供します。データセットはJVMオブジェクトから作成し、機能変換（マップ、フラットマップ、フィルタなど）を使用して操作できます。DataFrameは、名前付き列に編成されたデータセットです。概念的には、リレーショナルデータベースのテーブルまたはR/Pythonのデータフレームに相当します。データフレームは、構造化データファイル、HiveまたはHBaseのテーブル、オンプレミスまたはクラウドの外部データベース、既存のRDDなど、さまざまなソースから構築できます。

Sparkアプリケーションには1つ以上のSparkジョブが含まれています。ジョブは実行者でタスクを実行し、実行者はYarnコンテナで実行します。各実行者は1つのコンテナで実行され、実行者はアプリケーションのライフサイクルを通して存在します。実行者はアプリケーションの起動後に修正され、yarnは割り当て済みのコンテナのサイズを変更しません。実行者は、メモリ内のデータに対してタスクを同時に実行できます。

NetApp Sparkソリューションの概要

ネットアップには、FAS / AFF、Eシリーズ、Cloud Volumes ONTAP の3つのストレージポートフォリオがあります。AFF とONTAP を搭載したEシリーズは、Apache Spark搭載のHadoopソリューション向けに検証済みです。

ネットアップのデータファブリックは、以下の図に示すように、データアクセス、制御、保護、セキュリティのためのデータ管理サービスとアプリケーション（ビルディングブロック）を統合しています。

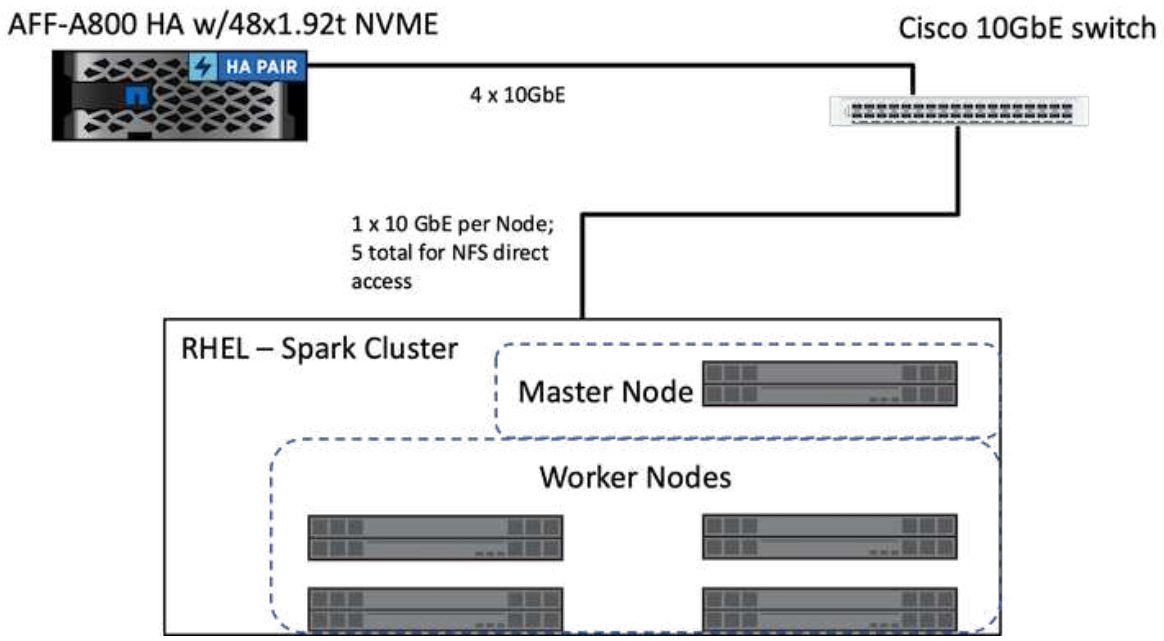


上の図の構成要素は次のとおりです。

- * NetApp NFS 直接アクセス。 * 最新の Hadoop クラスタと Spark クラスタを、ソフトウェアやドライバの追加の必要なしに NetApp NFS ボリュームに直接アクセスできます。
- * ネットアップの Cloud Volumes ONTAP とクラウドボリュームサービス。 * ソフトウェア定義型の接続ストレージ。 Amazon Web Services （ AWS ）で実行されている ONTAP または Microsoft Azure クラウドサービスで実行されている Azure NetApp Files （ ANF ）に基づいています。
- * NetApp SnapMirrorテクノロジ。*オンプレミスとONTAP クラウドまたはNPSインスタンス間のデータ保護機能を提供します。
- * クラウド・サービス・プロバイダー。 * これらのプロバイダーには、 AWS 、 Microsoft Azure 、 Google Cloud 、 IBM Cloud が含まれます。

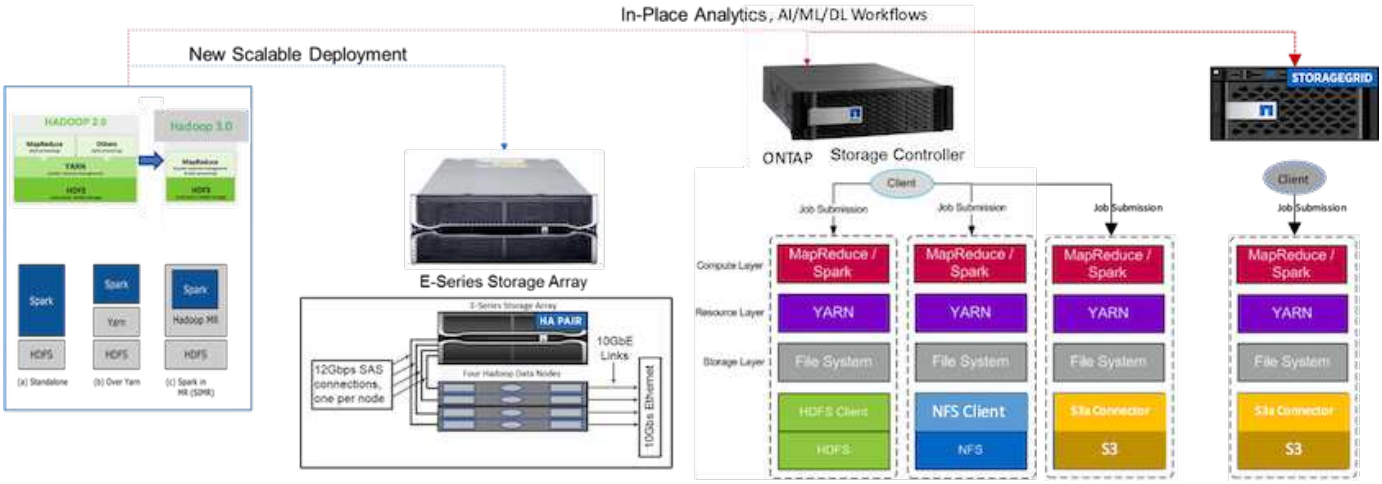
- * PaaS * AWS の Amazon Elastic MapReduce （ EMR ）や Databricks 、 Microsoft Azure HDInsight 、 Azure Databricks などのクラウドベースの分析サービスを利用できます。

次の図は、Sparkの解決策 とネットアップストレージを示しています。



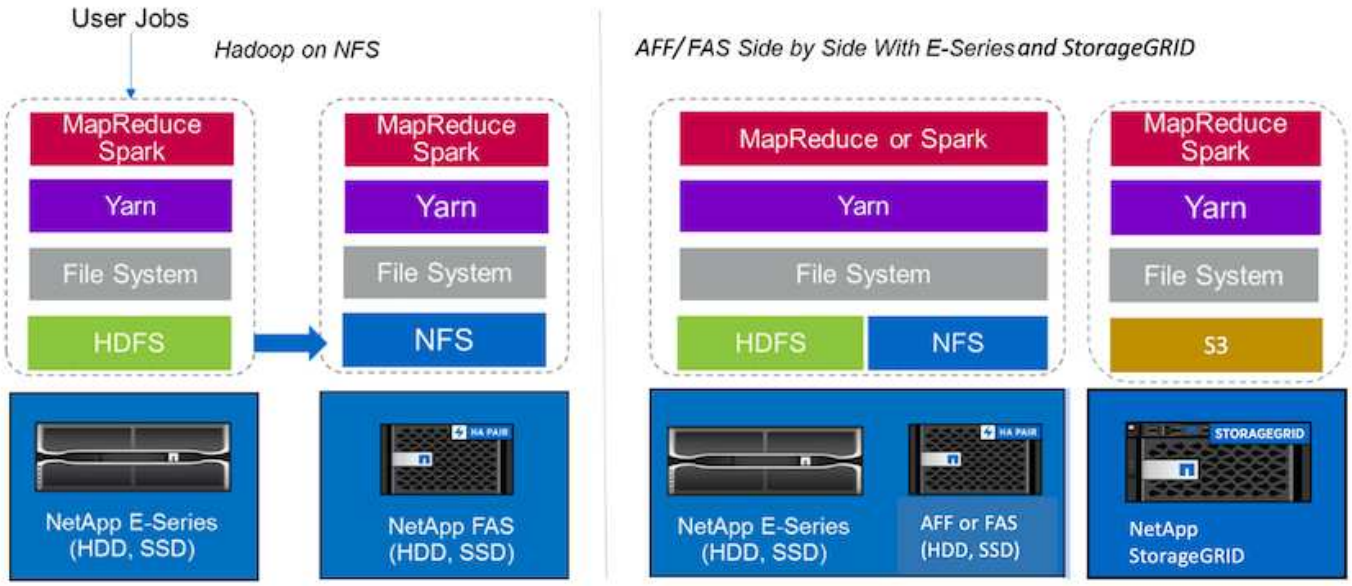
ONTAP Spark解決策 は、既存の本番データへのアクセスを使用して、インプレース分析、AI、ML、DLのワークフローに、ネットアップNFSダイレクトアクセスプロトコルを使用しています。Hadoopノードで使用可能な本番データは、インプレース分析ジョブ、AIジョブ、MLジョブ、DLジョブを実行するためにエクスポートされます。データにアクセスしてHadoopノード内で処理することができ、NetApp NFSに直接アクセスするかどうかは関係ありません。Sparkでは、スタンドアロンのクラスタマネージャまたは「yarn」クラスタマネージャを使用して、「」を使用してNFSボリュームを構成できます<file:///<target_volume>`。3つのユースケースに異なるデータセットを使用して検証しました。これらの検証の詳細については、「テスト結果」セクションを参照してください。 （XRef）

次の図は、NetApp Apache Spark / Hadoopストレージの位置付けを示しています。



また、EシリーズSparkの解決策 、AFF / FAS ONTAP Spark解決策 、StorageGRID Spark解決策 の独自の機能を特定し、詳細な検証とテストを実施しました。ネットアップでは、今回の調査結果に基づいStorageGRIDて、新規導入時と拡張性に優れた新規導入時にEシリーズ解決策 を使用し、既存のNFSデータを使用したインプレース分析、AI、ML、DL、DLのワークロードにはAFF / FAS解決策 を、オブジェクトストレージが必要な

場合には最新のデータ分析に使用することを推奨しています。



データレイクは、分析、AI、ML、DLの各ジョブに使用できる、ネイティブ形式の大規模データセット用のストレージリポジトリです。Eシリーズ、AFF / FAS、StorageGRID SG6060 Sparkソリューション用のデータレイクリポジトリを構築しました。Eシリーズシステムでは、Hadoop SparkクラスタへのHDFSアクセスが提供されますが、既存の本番環境のデータには、NFSの直接アクセスプロトコルを通じてHadoopクラスタへアクセスされます。オブジェクトストレージに配置されるデータセットに対しては、NetApp StorageGRID によってS3とS3aのセキュアなアクセスが提供されます。

ユースケースの概要

このページでは、この解決策 を使用できるさまざまな領域について説明します。

ストリーミングデータ

Apache Sparkはストリーミングデータを処理できます。ストリーミングデータは、抽出、変換、読み込み（ETL）プロセス、データのエンリッチ化、イベント検出のトリガー、複雑なセッション分析に使用されます。

- *ストリーミングETL。*データは継続的に消去され、データストアにプッシュされる前に集約されます。Netflixでは、KafkaストリーミングとSparkストリーミングを使用して、リアルタイムのオンラインムービーの推奨事項とデータ監視解決策 を構築しています。このソリューションでは、さまざまなデータソースから1日数十億件のイベントを処理できます。ただし、従来のETLではバッチ処理の処理方法が異なります。このデータは最初に読み取られ、次にデータベースに書き込まれる前にデータベース形式に変換されます。
- データのエンリッチ化。 Sparkストリーミングは、静的データを活用してライブデータを強化し、よりリアルタイムのデータ分析を可能にします。たとえば、オンライン広告主は、顧客行動に関する情報に基づいてパーソナライズされたターゲット広告を配信できます。
- イベント検出のトリガ。 Sparkストリーミングを使用すると、重大な問題を示す可能性のある異常な動作をすばやく検出して対応することができます。たとえば、金融機関はトリガーを使用して不正取引を検出および停止し、病院はトリガーを使用して患者のバイタルサインで検出された危険な健康状態変化を検出します。
- 複雑なセッション分析。 Sparkストリーミングは、Webサイトやアプリケーションにログインした後で、

ユーザーのアクティビティなどのイベントを収集し、それらをグループ化して分析します。たとえば、Netflixでは、この機能を使用してリアルタイムの動画推奨を提供しています。

より多くのストリーミングデータ構成、ConFluent Kafkaの検証、およびパフォーマンステストについては、を参照してください "TR-4912 ：『 Best Practices guidelines for ConFluent Kafka Tiered Storage with NetApp 』"。

機械学習

Sparkの統合フレームワークを使用すると、機械学習ライブラリ（MLlib）を使用してデータセットに対して繰り返しクエリを実行できます。MLlibは、予測インテリジェンス、マーケティング目的での顧客セグメンテーション、感情分析など、一般的なビッグデータ機能のためのクラスタリング、分類、次元縮小などの領域で使用されます。MLlibは、ネットワークセキュリティで、悪意のあるアクティビティの兆候を示すデータパケットをリアルタイムで検査するために使用されます。セキュリティプロバイダは、新しい脅威について学び、ハッカーの一歩先を行きながら、クライアントをリアルタイムで保護することができます。

ディープラーニング

TensorFlowは、業界全体で使用されている一般的なディープラーニングフレームワークです。TensorFlowは、CPUまたはGPUクラスタでの分散トレーニングをサポートしています。このトレーニングは分散されているため、ユーザはディープレイヤを大量に含む大容量のデータを使用してトレーニングを実行できます。

最近まで、Apache SparkとTensorFlowを使用したい場合は、PySparkでTensorFlowに必要なすべてのETLを実行し、中間ストレージにデータを書き込む必要がありました。そのデータは、実際のトレーニングプロセス用にTensorFlowクラスタにロードされます。このワークフローでは、ETL用とTensorFlowの分散トレーニング用の2つの異なるクラスタを管理する必要がありました。複数のクラスタを実行して保守する作業は、一般に煩雑で時間もかかりました。

以前のSparkバージョンのデータフレームとRDDは、ランダムアクセスが制限されていたため、ディープラーニングには適していませんでした。Spark 3.0では、プロジェクト水素を使用してディープラーニングフレームワークのネイティブサポートが追加されました。このアプローチにより、Sparkクラスタで非MapReduceベースのスケジューリングが可能になります。

対話式解析

Apache Sparkは、SQL、R、Pythonを含むSpark以外の開発言語でサンプリングしなくても、探索クエリを実行するのに十分な速さです。Sparkは視覚化ツールを使用して複雑なデータを処理し、対話式に視覚化します。Spark with structured streamingは、Web分析のライブデータに対してインタラクティブなクエリを実行します。これにより、Web訪問者の現在のセッションに対してインタラクティブなクエリを実行できます。

推薦システム

推薦システムは長年にわたり、オンラインショッピング、オンラインエンターテインメント、その他多くの業界の劇的な変化に企業や消費者が対応してきたため、私たちの生活に大きな変化をもたらしてきました。実際、これらのシステムは、AIの本番運用における成功事例として最も顕著に表れています。多くの実用的なユースケースでは、推薦システムを会話型AIやチャットボットと組み合わせてNLPバックエンドに接続することで、関連情報を取得し、有益な推論を作成しています。

現在、多くの小売業者は、オンラインでの購入や店舗での集荷、縁石側での集荷、セルフチェックアウト、スキャンと外出など、新しいビジネスモデルを採用しています。これらのモデルは、COVID-19のパンデミックでショッピングの安全性と消費者の利便性を高めることで、注目を引くようになっています。このようなデジタルトレンドの拡大には、AIが欠かせません。こうしたトレンドは消費者の行動に左右され、その逆も同様です。お客様のニーズの高まりに応え、カスタマーエクスペリエンスの強化、運用効率の向上、収益の拡大を実

現するために、ネットアップは、機械学習とディープラーニングのアルゴリズムを使用して、より迅速で正確な推奨システムを設計できるよう、大企業のお客様とビジネスを支援しています。

コラボレーションフィルタリング、コンテンツベースのシステム、ディープラーニングレコメンダモデル（DLRM）、ハイブリッド手法など、推奨事項を提供するために使用される一般的な手法がいくつかあります。以前は、PySparkを使って、推奨システムを作成するためのコラボレーションフィルタリングを実装していました。Spark MLlibは、コラボレーションフィルタリングのために交互に最小二乗（ALS）を実装しています。これは、DLRMが増加する前に、企業間で非常に人気のあるアルゴリズムです。

自然言語処理

会話型AIは、自然言語処理（NLP）によって実現され、コンピュータが人間と通信するのを支援するAIのブランチです。NLPは、スマートアシスタントやチャットボット、Google検索、予測テキストなど、業界のあらゆる業種や多くのユースケースで広く使用されています。に従って "Gartner社" 予測：2022年までに、人の70%が日常的に会話型AIプラットフォームとやり取りしています。人間と機械の間で質の高い会話を行うには、応答が迅速かつインテリジェントで、自然な音声である必要があります。

お客様は、NLPモデルとAutomatic Speech Recognition（ASR）モデルを処理してトレーニングするために大量のデータを必要としています。また、エッジ、コア、クラウドにわたってデータを移動する必要もあり、推論を数ミリ秒で実行して人間との自然な通信を確立する機能も必要です。NetApp AIとApache Sparkは、コンピューティング、ストレージ、データ処理、モデルトレーニング、微調整、 そして展開。

感情分析とは、NLP内で、肯定的、否定的、または中立的な感情がテキストから抽出される研究領域のことです。感情分析には、サポートセンターの従業員のパフォーマンスを発信者との会話から適切な自動チャットボット応答まで、さまざまなユースケースがあります。また、四半期ごとの収益呼において、企業の代表者と対象者の間のやり取りに基づいて会社の株価を予測するためにも使用されています。さらに、感情分析を使用して、ブランドが提供する製品、サービス、サポートに関するお客様の見解を判断できます。

使用しました "SparkのNLPです" ライブラリ元 "ジョンスノーラボ" を含むTransformers（BERT）モデルから事前にトレーニングされたパイプラインと双方向エンコーダリプレゼンテーションをロードするため "財務ニュースのセンチメント" および "FinBert"トークン化、Named Entity Recognition、モデルトレーニング、フィッティング、センチメント分析を大規模に実施しています。Spark NLPは、BERT、Albert、Electra、XLNet、DistilBERTなどの最先端のトランスを提供する唯一のオープンソースNLPライブラリです。 Roberta、DeberTa、XLM-Roberta、Longform, Elmo ユニバーサルセンテンスエンコーダー、Google T5、MarianMT、およびGPT2。このライブラリはPythonとRだけでなく、Apache Sparkをネイティブに拡張することで、JVMエコシステム（Java、Scala、Kotlin）でも大規模に動作します。

AI、ML、DLの主なユースケースとアーキテクチャ

主なAI、ML、DLのユースケースと手法は、以下のセクションに分類できます。

SparkのNLPパイプラインとTensorFlow分散推論です

次のリストには、データサイエンスコミュニティで採用されている最も一般的なオープンソースのNLPライブラリが、さまざまな開発レベルで含まれています。

- "Natural Language Toolkit（NLTK）"。すべてのNLP手法に対応する完全なツールキットです。2000年代初頭から維持されています。
- "TextBLOB"。NLTKとPatternの上に構築された使いやすいNLPツールPython API。
- "Stanford Core NLP"。Stanford NLP Groupが開発したJavaのNLPサービスとパッケージ。
- "Gensim氏"。人間のトピックモデリングは、チェコデジタル数学ライブラリプロジェクトのPythonスクリ

プトの集合として開始されました。

- "スパレーシー"。PythonとCythonを使用した、トランスフォーマ用GPUアクセラレーションを備えたエンドツーエンドの産業用NLPワークフロー。
- "Fasttextの場合"。FacebookのAI Research（Fair）ラボで作成された単語埋め込みや文分類の学習用の無料の軽量オープンソースNLPライブラリです。

Spark NLPは、あらゆるNLPタスクと要件に対応する単一のユニファイド解決策 です。拡張性が高く、パフォーマンスが高く、精度の高いNLPベースのソフトウェアを、実稼働環境で使用できます。また、転移学習を活用し、研究やさまざまな業界における最新のアルゴリズムとモデルを実装しています。Sparkは上記のライブラリを完全にサポートしていないため、Spark NLPはの上に構築されました "Spark ML" Sparkの汎用インメモリ分散データ処理エンジンを、ミッションクリティカルな本番ワークフロー向けのエンタープライズクラスのNLPライブラリとして活用しよう。アノテータは、ルールベースのアルゴリズム、機械学習、TensorFlowを利用してディープラーニングの実装を強化しています。トークン化、レマタイ化、語幹化、部分読み上げタギング、名前付きエンティティー認識など、一般的なNLPタスクを取り上げますが、これらに限定されません。スペルチェックと感情分析。

トランスフォーマー（BERT）の双方向エンコーダリプレゼンテーションは、NLPのトランスベースの機械学習技術です。事前トレーニングと微調整の概念を普及させました。BERTの変圧器アーキテクチャは機械翻訳から生まれたもので、回帰型ニューラルネットワーク（RNN）ベースの言語モデルよりも長期的な依存関係をモデル化します。また、マスク言語モデリング（MLM）タスクも導入されました。このタスクでは、すべてのトークンのランダムな15%がマスクされ、モデルによって予測され、真の双方向性が実現されます。

金融感情の分析は、専門的な言語と、その分野のラベル付けされたデータが不足しているために困難になっています。 "FinBert"事前に訓練されたBERTに基づく言語モデルであるBERTは、ドメインに適合しています "ロイターTRRC2"、金融コーパス、およびラベル付けされたデータと微調整された("金融PhraseBankの"）を参照してください。研究者たちは、財務用語を使ってニュース記事から4、500件の文章を抽出した。次に、16人の専門家と修士の学生が、財務上の背景にポジティブ、ニュートラル、ネガティブのラベルを付けています。2016年から2020年までの間に、FinBERTと他の2つの事前トレーニングパイプライン（）を使用してNASDAQ企業収益の売上高記録の感情を分析するために、Sparkのエンドツーエンドのワークフローを構築しました "財務ニュースの業況分析"、 "ドキュメントDLについて説明する"）をSparkのNLPから取得します。

Spark NLPの基礎となるディープラーニングエンジンは、機械学習向けのエンドツーエンドのオープンソースプラットフォームであるTensorFlowです。モデル構築が容易で、どこでも堅牢なML生産を実現し、研究のための強力な実験を可能にします。このため、Sparkの「yarn cluster」モードでパイプラインを実行する場合、基本的には分散TensorFlowを実行し、1つのマスターノードと複数のワーカーノード、およびクラスタにマウントされたネットワーク接続型ストレージにわたって、データとモデルの並列化を行いました。

Horovodの分散トレーニング

MapReduce関連のパフォーマンスの中核となるHadoop検証は、TeraGen、TeraSort、TeraValidate、およびDFSIO（読み取りおよび書き込み）を使用して実行されます。に、TeraGenおよびTeraSortの検証結果を示します "TR-3969：『NetApp Solutions for Hadoop』" Eシリーズおよび「ストレージ階層化」（xref）for AFFのセクションに記載されています。

お客様からの要望に基づいて、Sparkを使用したトレーニングの配布は、さまざまなユースケースで最も重要なトレーニングの1つと考えています。このドキュメントでは、を使用しました "Hovorod on Spark（SparkでのHovorod" ネットアップのオンプレミス、クラウドネイティブ、ハイブリッドクラウドソリューションでSparkのパフォーマンスを検証するには、AFF ストレージコントローラ、Azure NetApp Files 、FAS StorageGRID をご利用ください。

Horovod on Sparkパッケージは、Horovodの便利なラッパーを提供します。このラッパーはSparkクラスタで分散されたトレーニングワークロードを簡単に実行できるようにするものです。厳密なモデル設計ループでは、トレーニングデータと推論データが存在するSparkで、データ処理、モデルトレーニング、モデル評価が

すべて行われます。

SparkでHorovodを実行するためのAPIには、高レベルのエスティメータAPIと低レベルの実行APIの2つがあります。どちらも、Sparkの実行者に対してHorovodを起動するために同じ基盤メカニズムを使用していますが、Estimator APIはデータ処理、モデルトレーニングループ、モデルチェックポイント、メトリック収集、および分散トレーニングを抽象化します。Horovod Spark Estimators、TensorFlow、Kerasを使用して、に基づくエンドツーエンドのデータ準備と分散トレーニングワークフローを実施しました "Kagle Rossmann Store Sales" 競合他社

スクリプト「kers_spark_horovod_Rossmann _ estimator.py」は、のセクションにあります "主なユースケースごとにPythonスクリプトを使用できます。" 次の3つの部分で構成されます

- 最初の部分では、Kaggleが提供し、コミュニティが収集した最初のCSVファイルのセットを介して、さまざまなデータ前処理ステップを実行します。入力データは'Validation'サブセットとテストデータセットで構成されるトレーニングセットに分かれています
- 2番目の部分では、対数シグスモイド活性化関数とAdamオプティマイザを使用してKeras Deep Neural Network（DNN）モデルを定義し、Sparkに対してHorovodを使用してモデルの分散トレーニングを実行します。
- 3番目の部分では、検証セット全体の平均絶対エラーを最小化する最適なモデルを使用して、テストデータセットの予測を実行します。次に、出力CSVファイルが作成されます。

を参照してください "「機械学習」" を参照してください。

CTR予測に**Keras**を使用したマルチワーカーディープラーニング

ML プラットフォームとアプリケーションの最近の進歩により、現在は大規模な学習が注目されています。クリックスルー率（ CTR ）は、オンライン広告インプレッション数 100 件あたりの平均クリックスルー数（パーセンテージ）と定義されています。デジタルマーケティング、小売、 E コマース、サービスプロバイダなど、さまざまな業界やユースケースで重要な指標として広く採用されています。を参照してください "TR-4904 ：『 Distributed Training in Azure - Click Through Rate Prediction 』" Kubernetes、分散データETL、DaskおよびCUDA MLを使用したモデルトレーニングなど、CTRのアプリケーションとエンドツーエンドのクラウドAIワークフロー実装の詳細を確認できます。

このテクニカルレポートでは、別のを使用しました "Criteo Terabyteのログデータセットをクリックします"（TR-4904を参照）。Kerasを使用した複数のワーカーによる分散型ディープラーニングで、Deep NetworkモデルとCross Network（DCN）モデルを使用したSparkワークフローを構築し、ログ損失エラー機能のパフォーマンスをベースラインSparkのSpark ML Logistic Regression(ログ記録的回帰)モデルと比較します。DCNは、制限された角度の効率的な機能の相互作用を効率的にキャプチャし、高度な非線形相互作用を学習し、手動によるフィーチャーエンジニアリングや完全な検索を必要とせず、計算コストも低くなります。

Webスケールのレコメンダシステムのデータはほとんどが個別に分類されるため、フィーチャーの探索には困難な大規模でスパースな機能スペースが必要になります。これは、大規模なシステムのほとんどをロジスティック回帰などの線形モデルに限定しています。しかし、予測可能な機能を頻繁に特定すると同時に、見過ごしていない機能やまれなクロス機能を調べることが、予測を適切に行うための鍵となります。線形モデルは単純で、解析可能で、簡単にスケール変更できますが、表現力は限られています。

一方、クロス機能は、モデルの表現力を向上させる上で重要であることが示されています。残念なことに、このような機能を特定するには、手動での機能開発や完全な検索が必要になることがよくあります目に見えない機能の相互作用を一般化することは、しばしば困難です。DCNのような十字型ニューラルネットワークを使用すると、自動で機能交差を明示的に適用することで、タスク固有の機能エンジニアリングを回避できます。クロスネットワークは複数のレイヤで構成されており、レイヤの深さによって高度な相互作用がプロバンスされます。各レイヤは、既存のレイヤに基づいて上位の相互作用を生成し、以前のレイヤからの相互作用を保持します。

Deep Neural Network（DNN；ディープニューラルネットワーク）は、さまざまな機能で非常に複雑なインタラクションをキャプチャすることを約束します。ただし、DCNと比較して、必要なパラメータの数は非常に多く、クロス機能を明示的に形成できず、一部のタイプの機能の相互作用を効率的に学習できない場合があります。クロスネットワークはメモリ効率が高く、実装も簡単です。クロスコンポーネントとDNNコンポーネントを共同でトレーニングし、予測機能のインタラクションを効率的に取り込み、Crito CTRデータセットで最先端のパフォーマンスを提供します。

DCNモデルは、埋め込みレイヤーとスタッキングレイヤーから始まり、クロスネットワークとディープネットワークが並行して使用されます。次に、2つのネットワークからの出力を組み合わせた最終的な組み合わせレイヤを示します。入力データは、スパースフィーチャーとデンスフィーチャーを持つベクトルにすることができます。Sparkでは、その両方です "ml" および "ミリリブ" ライブラリには「SparseVector」タイプが含まれます。したがって、ユーザーがそれぞれの機能やメソッドを呼び出す際には、2つの機能を区別し、注意することが重要です。CTR予測などのWebスケールの推薦システムでは、入力は主に「country = USA」などの分類的な機能です。このような機能は、多くの場合、1つのホットベクトルとしてエンコードされます。たとえば、「[0,1,0,..]」のようになります。「SparseVector」を使用したワン・ホット・エンコーディング（OHE）は、絶えず変化する語彙や拡大する語彙を持つ実世界のデータセットを扱う場合に便利です。で例を変更しました "Deepctr" 大きなボキャブラリを処理するために、DCNの埋め込みレイヤーとスタッキングレイヤーに埋め込みベクトルを作成します。

。 "Critoディスプレイ広告のデータセット" 広告のクリックスルーレートを予測します。13の整数型の機能と、各カテゴリの基数が多い26の分類的な機能があります。このデータセットでは、入力サイズが大きいため、ログロスの0.001が実質的に大きく改善されています。大規模なユーザベースの予測精度がわずかに向上すると、企業の収益が大きく増加する可能性があります。データセットには7日間の11GBのユーザログが格納されており、これは約4100万レコードに相当します。Sparkのdataframe .randomSplit()関数を使用して、トレーニング用のデータ(80%)、クロス検証(10%)、およびテスト用の残りの10%をランダムに分割しました。

DCNは、Kerasを使用したTensorFlowに実装されました。DCNを使用したモデルトレーニングプロセスの実装には、次の4つの主要コンポーネントがあります。

- *データ処理と埋め込み。*ログトランスフォームを適用することで、リアルタイム機能が正規化されます。カテゴリフィーチャーの場合、寸法6×（カテゴリの基数）1/4の密度の高いベクトルにフィーチャーを埋め込みます。すべての埋め込み結果を次元1026のベクトルに連結します。
- 最適化 Adam Optimizerを使用してミニバッチ確率的最適化を適用しました。バッチサイズは512に設定されています。ディープネットワークにバッチ正規化が適用され、グラジエントクリップの基準が100に設定されました。
- *均一化。*私達はL2の均一化かドロップアウトが有効であることが見つけられなかったので早い停止を使用した。
- * Hyperparameters*。非表示レイヤー数、非表示レイヤーサイズ、初期学習レート、およびクロスレイヤー数に基づくグリッド検索に基づく結果を報告します。非表示レイヤーの数は2〜5で、非表示レイヤーのサイズは32〜1024です。DCNの場合、クロスレイヤの数は1〜6です。初期学習レートは0.0001から0.001に調整され、0.0001単位で増加しました。すべての実験は訓練ステップ150,000で早期停止を適用し、それを超えて過剰なフィッティングが発生し始めました。

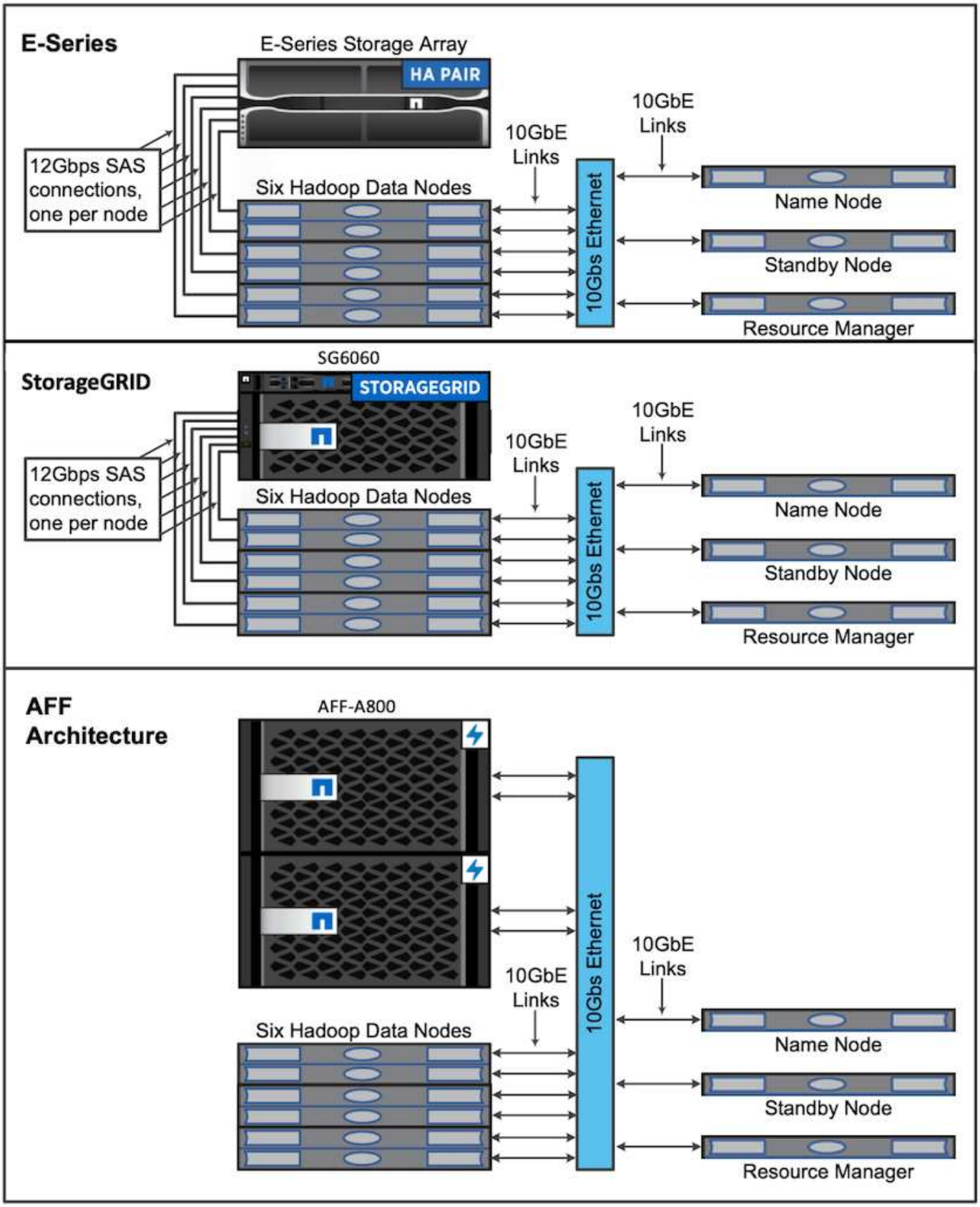
DCNに加えて、CTRの予測に使用される他の一般的なディープラーニングモデルもテストしました "DeepFM"、 "xDeepFM"、 "自動内部（AutoInt"および "DCN v2"。

検証に使用するアーキテクチャ

この検証では、4つのワーカーノードと1つのマスターノードにAFF-A800 HAペアを使用しました。すべてのクラスタ・メンバーを、10GbEネットワーク・スイッチを介して接続しました。

今回のNetApp Sparkの解決策 検証では、E5760、E5724、AFF-A800の3種類のストレージコントローラを使

用しました。Eシリーズストレージコントローラは、12Gbps SAS接続の5つのデータノードに接続されました。AFF のHAペアストレージコントローラは、エクスポートされたNFSボリュームを10GbEでHadoopワーカーノードに接続することで提供します。Hadoopクラスタのメンバーは、Eシリーズ、AFF 、およびStorageGRID のHadoopソリューションで10GbE接続を介して接続されます。



テスト結果

TeraGenベンチマークツールでTeraSortおよびTeraValidateスクリプトを使用して、E5760、E5724、およびAFF-A800構成でのSparkのパフォーマンス検証を測定しました。さらに、SparkのNLPパイプラインとTensorFlow分散トレーニング、Horovod分散トレーニング、Kerasを使用したディープFMでのCTR予測を利用した複数ワーカーのディープラーニングという、3つの主なユースケースをテストしました。

EシリーズとStorageGRID の検証には、Hadoopレプリケーションファクタ2を使用しました。AFF の検証に使用したデータソースは1つだけです。

次の表に、Sparkのパフォーマンス検証のハードウェア構成を示します。

を入力します	**Hadoop**ワーカーノード	ドライブタイプ	ノードあたりのドライブ数	ストレージコントローラ
SG6060 の設計	4.	（ SAS ）。	12.	単一のハイアベイラビリティ（HA）ペア
E5760	4.	（ SAS ）。	60	単一のHAペア
E5724	4.	（ SAS ）。	24	単一のHAペア
AFF800が必要です	4.	SSD の場合	6.	単一のHAペア

次の表に、ソフトウェア要件を示します。

ソフトウェア	バージョン
RHEL	7.9
OpenJDKランタイム環境	1.8.0
OpenJDK 64ビットのServer VM	25.302
Git	2.24.1
GCC/G++	11.2.1.
火花	3.2.1
PySpark	3.1.2
SparkNLPです	3.4.2
TensorFlowです	2.9.0
クラス	2.9.0
ホロボド	0.24.3

財務心理分析

公開しました "TR-4910 ：『 NetApp AI と顧客コミュニケーションを組み合わせた感情分析』"を使用して、エンドツーエンドの会話型AIパイプラインを構築しました "NetApp DataOps ツールキット"、AFF ストレージ、NVIDIA DGXシステムパイプラインは、DataOpsツールキットを活用して、バッチオーディオ信号処理、Automatic Speech Recognition（ASR）、転送学習、感情分析を実行します。 "NVIDIA Riva SDKを参照

してください"および "Taoフレームワーク"。金融サービス業界へのセンチメント分析のユースケースの拡大、SparkNLPワークフローの構築、特定事業体の認識など、さまざまなNLPタスクに対する3つのBERTモデルのロード、NASDAQ Top 10 Companiesの四半期収益コールに関するセンテンスレベルのセンチメントの取得。

次のスクリプト「センチメント_アナリシス_スパーク」。PY'はFinBERTモデルを使用してHDFS内のトランスクリプトを処理し、次の表に示すようにポジティブでニュートラルでネガティブな感情カウントを生成します。

```
-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
```

次の表に、2016年から2020年までのNASDAQトップ10企業の収益/コール、センテンスレベルの感情分析を示します。

センチメントの数値と割合	**10**社すべて	**AAPL**	**AMD**	**AMZN**	**CSCO**	**GOOGL**	**INTC**	マイクロソフト	**NVDA**
正の数	7447	1567年	743	290	682	826	824	904	417
ニュートラルカウント	64067	6856	7596	5086	6650	5914	6099	5715	6189
負の数	1787年になります	253	213	84	189	97	282	202.	89
分類なしのカウント	196	0	0	76	0	0	0	1.	0
（合計数）	73497	8676	8552.	5536	7521	6837	7205.	6822	6695

割合の点では、CEOとCFOが話している文のほとんどは事実上であり、中立的な感情を持っています。決算発表時に、アナリストは肯定的または否定的な感情を伝える可能性のある質問をします。また、マイナスやプラスの心理が株価に与える影響についても、取引日の同日または翌日に定量的に調査することもできます。

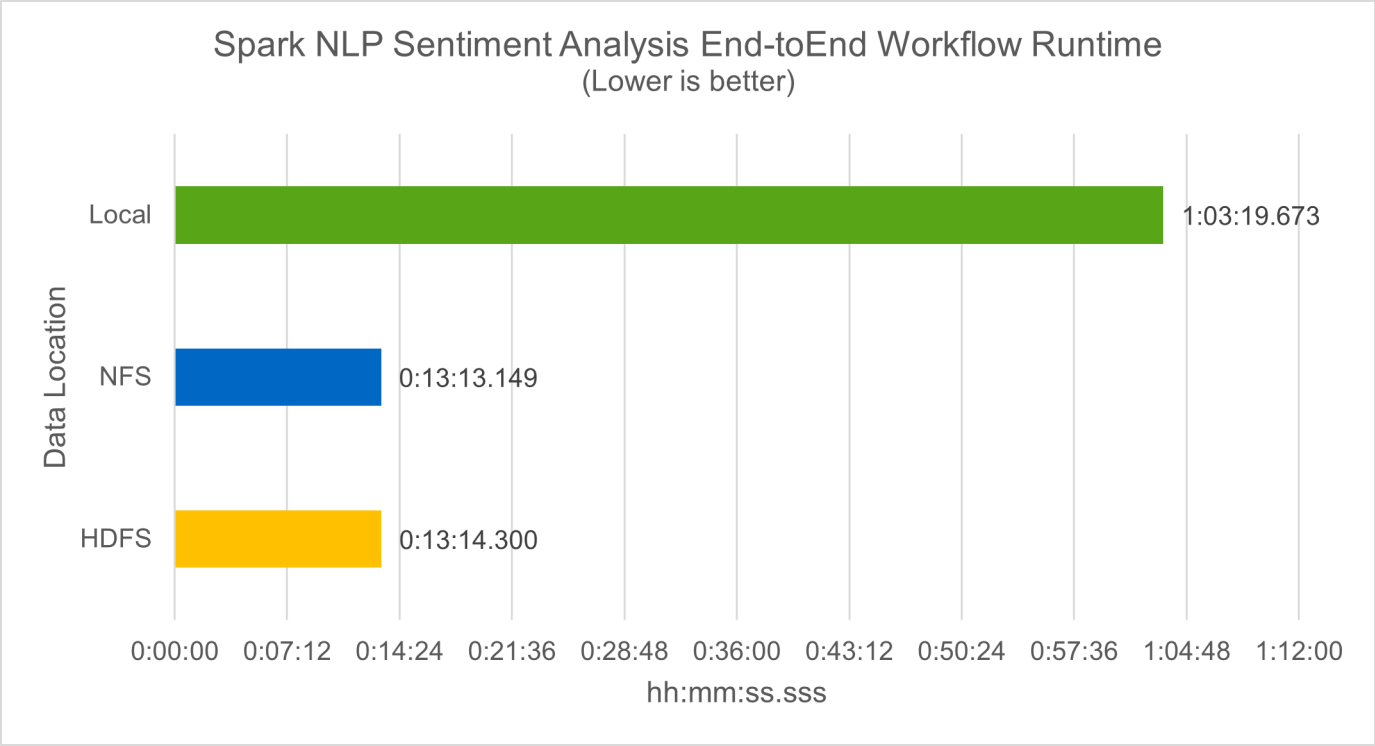
次の表に、NASDAQトップ10企業の文章レベルの感情分析をパーセントで示します。

センチメントの割合	**10**社すべて	**AAPL**	**AMD**	**AMZN**	**CSCO**	**GOOGL**	**INTC**	マイクロソフト	**NVDA**
肯定的	10.13%	18.06%	8.69%	5.24%	9.07%	12.08%	11.44%	13.25%	6.23%
ニュートラル	87.17%	79.02%	88.82%	91.87%	88.42%	86.50%	84.65%	83.77%	92.44%
負	2.43%	2.92%	2.49%	1.52%	2.51 %	1.42 %	3.91%	2.96 %	1.33%
分類なし	0.27%	0%	0%	1.37 %	0%	0%	0%	0.01%	0%

ワークフローの実行時間に関しては'local'モードからHDFSの分散環境に至るまで4.78倍の大幅な改善が見られましたまた'NFSを活用することで'さらに0.14%の向上が見られました

```
-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
```

次の図に示すように、データとモデルの並列処理によって、データ処理と分散TensorFlowモデルの推論速度が向上しています。NFSのデータの場所では、トレーニング済みのモデルがワークフローのボトルネックになっているため、ランタイムが若干向上しました。Transcriptデータセットのサイズを増やすと、NFSの方が明らかになります。



Horovodのパフォーマンスを使用した分散トレーニング

次のコマンドでは、1つのコアを持つ160個の実行者を持つ単一の「マスター」ノードを使用して、Sparkクラスタ内にランタイム情報とログファイルを生成しました。実行者メモリはメモリ不足エラーを回避するために5GBに制限されていました。を参照してください "「主要な各ユースケース用のPythonスクリプト」" データ処理、モデル・トレーニング、およびモデル精度計算の詳細については、「kers_spark_horovod_Rossmann _dimator.py」を参照してください。

```
(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
```

トレーニング期間が10回の場合の結果、次のようになりました。

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

入力データの処理、DNNモデルのトレーニング、精度の計算、TensorFlowチェックポイントと予測結果のCSVファイルの作成に43分以上かかりました。トレーニング期間を10に制限しました。実際には100に設定されていることが多く、モデルの精度が十分であることを確認しています。トレーニング時間は通常、エポックの数に比例して拡大します。

次に、クラスタで使用可能な4つのワーカーノードを使用して、HDFS内のデータで「yarn」モードで同じスクリプトを実行しました。

```
(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
```

結果として得られる実行時間は次のように改善されました。

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

HorovodのモデルとSparkのデータの並列化により、「yarn」と「local」モードを比較したランタイムが5.29x短縮され、トレーニング期間が10時間に短縮されました。次の図に、凡例に「hdfs」と「Local」を示します。基盤となるTensorFlow DNNモデルのトレーニングを、GPUがあればさらに高速化できます。このテストを実施し、今後のテクニカルレポートに結果を公開する予定です。

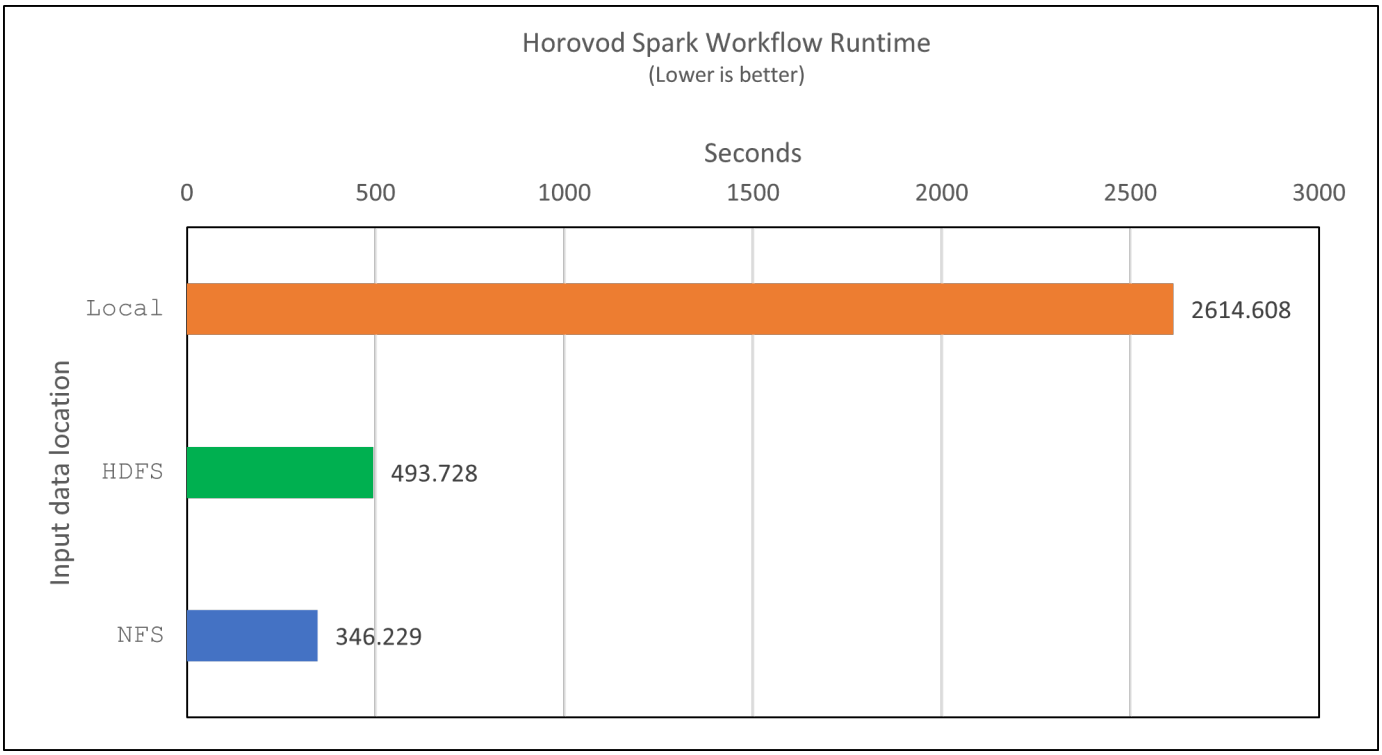
次のテストでは、NFSとHDFSの入力データをランタイムで比較しました。AFF A800のNFSボリュームは、Sparkクラスタ内の5つのノード（マスター1つ、ワーカー4つ）にまたがって「/sparkdemo/horovod」にマウントされました。前のテストと同様のコマンドを実行しましたが'--data-dir'パラメータは現在NFSマウントを指しています

```
(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
```

NFSを使用した場合の実行時間は次のようになりました。

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

次の図に示すように、1.43倍の速度がさらに向上しました。このため、ネットアップのオールフラッシュストレージをクラスタに接続することで、Horovod Sparkワークフローの高速データ転送と配信というメリットを享受し、1つのノードで実行する場合と比べて7.55x高速化を達成できます。



CTR予測パフォーマンスのディープラーニングモデル

クリック率を最大化するように設計されたレコメンダシステムでは、低い順に数学的に計算される可能性のある、ユーザーの行動の背後にある高度な機能の相互作用を学習する必要があります。低次と高次の両方の機能

の相互作用は、どちらか一方をバイアスすることなく、ディープラーニングモデルにとっても同様に重要です。新しいニューラルネットワークアーキテクチャでの機能学習に向けて、界面活性化機械ベースのニューラルネットワークであるDeep Factorization Machine（DeepFM）は、界面活性化装置を組み合わせた推奨製品です。

従来の三角分解機械は、機能間の潜伏ベクトルの内側としてのペアワイズ機能の相互作用をモデル化しますが、理論的には高次情報をキャプチャすることができます。実際には、機械学習の実践では、一般的に、計算と保管の複雑さが高いため、二次フィーチャーの相互作用しか使用しませGoogleなどのディープニューラルネットワークのバリエーション "ワイド・モデルとディープ・モデル" 一方、リニアワイドモデルとディープモデルを組み合わせて、ハイブリッドネットワーク構造で高度な機能の相互作用を学習します。

このワイド・ディープ・モデルには2つの入力があります。1つは基本的なワイド・モデル用で、もう1つはディープのためです。後者の部分では、エキスパートフィーチャー・エンジニアリングが必要です。このため、この手法は他のドメインには一般的にできません。ワイド・ディープ・モデルとは異なり、DeepFMはフィーチャー・エンジニアリングなしでRAW機能を使用して効率的にトレーニングできます。ワイド・パートとディープ・パートは同じ入力と埋め込みベクトルを共有するためです。

私たちはまず'セクションのrun_classification_Crito_spark.pyを使用して'ctr_trine.csv'という名前のCSVファイルにCrito'trine.csv'をNFSマウント'/sparkdemo/tr-4570-data'に格納しました "「主なユースケースごとにPythonスクリプトを用意しています。」" このスクリプト内で'関数process_input_file'は'タブを削除し'区切り文字として"を'改行として"を挿入するための複数の文字列メソッドを実行しますコードブロックがコメントとして表示されるように、元の「train .txt」を1回だけ処理する必要があることに注意してください。

以下の異なるDLモデルのテストでは、「ctr_train .csv」を入力ファイルとして使用しました。その後のテスト実行では、入力CSVファイルがSpark DataFrameに読み込まれ、スキーマに「label」のフィールド、整数の高密度フィーチャー「I1」、「I2」、「I3」、…、「I13」が含まれています。 また'希薄な機能は"C1'、'C2'、'C3'、…、'C26']`です次の「spark-smSubmit」コマンドは、入力CSVで実行し、クロス検証用に20%のスプリットを備えたDeepFMモデルをトレーニングし、10回のトレーニング期間後に最適なモデルを選択して、テストセットの予測精度を計算します。

```
(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
```

データファイル「ctr_train .csv」は11 GBを超えるため、エラーを回避するには、データセットサイズよりも十分な「spark.driver.maxResultSize」を設定する必要があります。

```
 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()
```

上記のSparkSession.Builder'構成でも有効になっています "Apache Arrowの"は、SparkのDataFrameを「df.toPandas ()」メソッドを使用してPandas DataFrameに変換します。

```
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.
```

ランダムにスプリットした後、トレーニングデータセットに3、6M行以上、テストセットに9、000サンプル以上が存在します。

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

このテクニカルレポートでは、GPUを使用せずにCPUテストに焦点を当てているため、適切なコンパイラフラグを使用してTensorFlowを構築することが重要です。これにより、GPUアクセラレーションライブラリの呼び出しを回避し、TensorFlowのAdvanced Vector Extensions（AVX）およびAVX2命令を最大限に活用できます。これらの機能は、ベクトル化された加算、フィードフォワード内の行列乗算、またはバック伝播DNNトレーニングなどの線形代数計算用に設計されています。256ビット浮動小数点(FP)レジスタを使用したAVX2で使用可能なFMA (fMultiply Add)命令は、整数コードとデータ型に最適で、最大2倍の速度を実現します。FPコードとデータ型の場合、AVX2はAVXと比較して8%の高速化を実現します。

```
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.
```

ソースからTensorFlowを構築する場合は、を使用することを推奨します "バザー"。今回の環境では、シェルプロンプトで以下のコマンドを実行して、「リタイア」、「リタイヤ」、「バザール」をインストールしました。

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

ビルドプロセス中にC++ 17の機能を使用するには、GCC 5以降を有効にする必要があります。この機能は、RHELがソフトウェアコレクションライブラリ(SCL)とともに提供します。次のコマンドは'devtoolset'とGCC 11.2.1をRHEL 7.9クラスタにインストールします

```
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
```

最後の2つのコマンドは'devtoolsets-11'を有効にしますこれには'/opt/r/devtoolset11-root//usr/bin/gcc`(GCC 11.2.1)が使用されますまた'gitのバージョンが1.8.3よりも大きいことを確認してください(RHEL 7.9に付属しています)これを参照してください "記事" 「git」を2.24.1に更新します。

最新のTensorFlowマスターリポジトリもすでにクローニング済みであるとします。次に'workspace'ファイルを使用して'workspace'ディレクトリを作成し'AVX、AVX2、FMAを使用してソースからTensorFlowを構築しますconfigureファイルを実行し、正しいPythonバイナリの場所を指定します。 "CUDA （ CUDA" はGPUを使用していないため、テストでは無効になっています。設定に応じて'`. bazelrc'ファイルが生成されますさらに、ファイルを編集し、HDFSのサポートを有効にするために「build—define=no_hdfs_support=false」を設定しました。セクションの「. bazelrc」を参照してください "「主なユースケースごとにPythonスクリプトを用意しています。」" 設定とフラグの完全なリストについては、を参照してください。

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

適切なフラグを使用してTensorFlowを構築したら、次のスクリプトを実行してCritoディスプレイ広告データセットを処理し、DeepFMモデルをトレーニングし、予測スコアからReceiver Operating Characteristic Curve（ROC AUC）の下の領域を計算します。

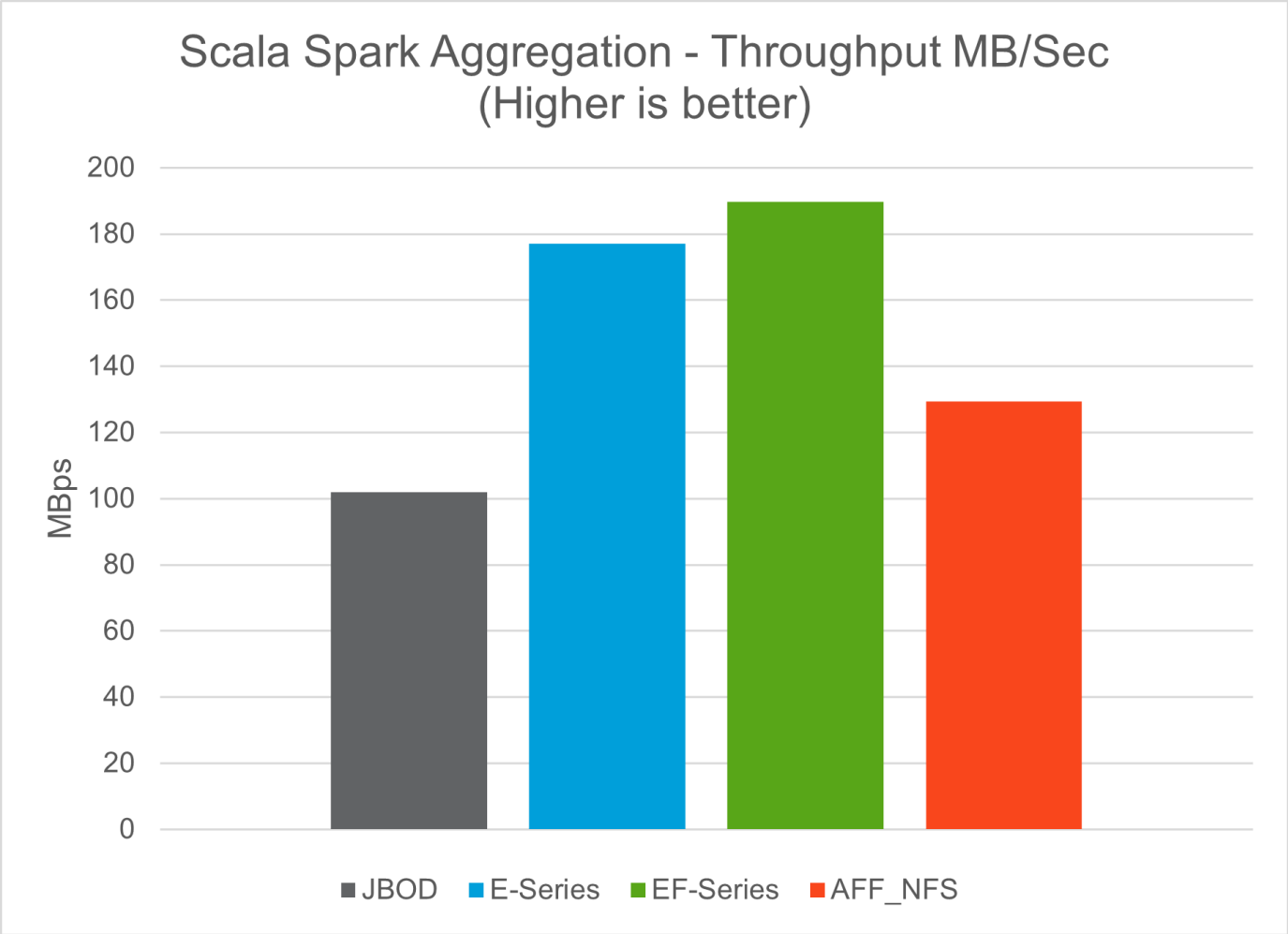
```
(base) [root@n138 examples]# ~/anaconda3/bin/spark-submit
--master yarn
--executor-memory 15g
--executor-cores 1
--num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py
--data-dir file:///sparkdemo/tr-4570-data
> . /run_classification_criteo_spark_nfs.log 2>&1
```

トレーニング期間が10回終了したら、テストデータセットのAUCスコアを取得しました。

```
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
```

以前のユースケースと同様に、Sparkワークフローランタイムを異なる場所にあるデータと比較しました。次

の図は、Sparkワークフローランタイムのディープラーニングの CTR予測の比較を示しています。

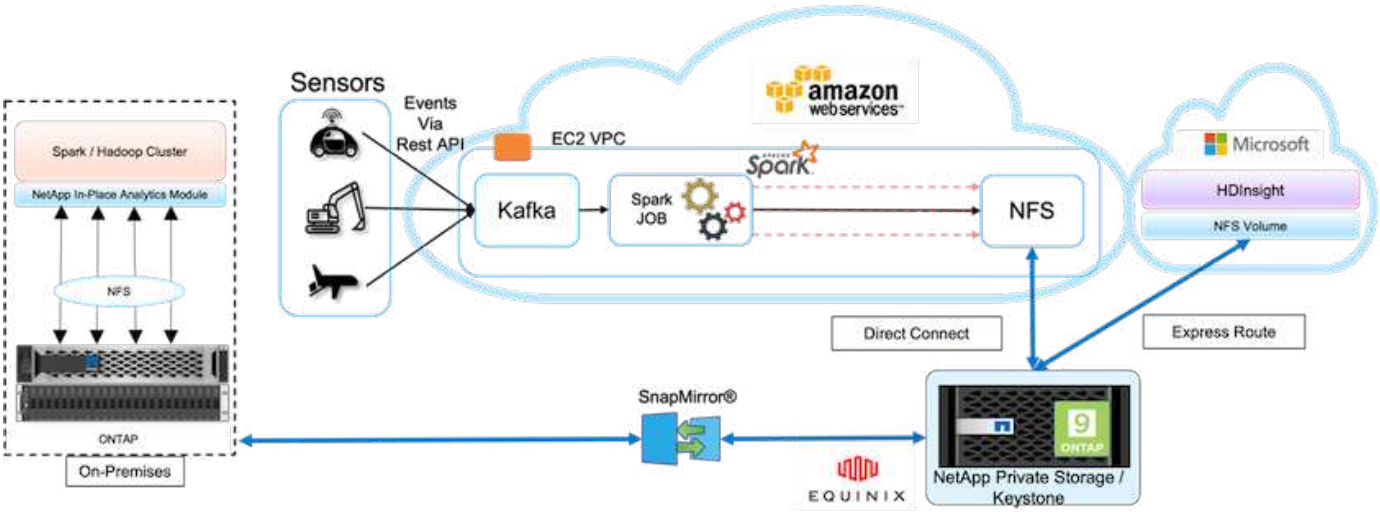


Hybrid Cloud解決策 の略

最新のエンタープライズデータセンターとは、複数の分散インフラ環境を、オンプレミスや複数のパブリッククラウドと一貫した運用モデルで継続的なデータ管理プレーンを通じて接続するハイブリッドクラウドです。ハイブリッドクラウドを最大限に活用するには、データの変換やアプリケーションのリファクタリングを行わなくても、オンプレミス環境とマルチクラウド環境間でデータをシームレスに移動できる必要があります。

データ保護などのユースケースでセカンダリストレージをクラウドに移行するか、アプリケーション開発やDevOpsなどのビジネスクリティカルなワークロードをクラウドに移行することで、ハイブリッドクラウドへの移行を開始するとお客様から指摘を受けています。そして、より重要なワークロードに移行します。Webおよびコンテンツホスティング、DevOpsやアプリケーション開発、データベース、分析、コンテナ化されたアプリケーションは、最も一般的なハイブリッドクラウドワークロードです。エンタープライズAIプロジェクトの複雑さ、コスト、リスクは、これまでAIの導入を試験段階から本番運用に妨げていました。

ネットアップのハイブリッドクラウド解決策 なら、セキュリティ、データガバナンス、コンプライアンスの統合ツールを活用できます。また、分散環境全体でデータとワークフローを一元管理できる単一のコントロールパネルを使用して、総所有コストを消費量に応じて最適化できます。次の図は、お客様のビッグデータ分析データにマルチクラウド接続を提供するという課題を抱えるクラウドサービスパートナーの解決策 の例です。



このシナリオでは、AWSでさまざまなソースから受け取ったIoTデータが、NetApp Private Storage（NPS）内の1箇所に保存されます。NPSストレージは、AWSとAzure上にあるSparkクラスタやHadoopクラスタに接続されているため、複数のクラウドで同じデータにアクセスするビッグデータ分析アプリケーションを実行できます。このユースケースの主な要件と課題は次のとおりです。

- お客様は、複数のクラウドを使用して、同じデータに対して分析ジョブを実行したいと考えています。
- オンプレミス環境やクラウド環境などのさまざまなソースから、さまざまなセンサーやハブを介してデータを受信する必要があります。
- 解決策 は、効率性とコスト効率に優れている必要があります。
- 主な課題は、コスト効率と効率に優れた解決策 を構築し、オンプレミス環境とクラウド環境の間でハイブリッド分析サービスを提供することです。

ネットアップのデータ保護機能とマルチクラウド接続解決策 は、複数のハイパースケーラにわたるクラウド分析アプリケーションの課題を解決します。上の図に示すように、センサーからのデータはストリーミングされ、 Kafka を介して AWS Spark クラスタに取り込まれます。データは NPS 内の NFS 共有に格納されます。NPS は、 Equinix データセンター内のクラウドプロバイダの外部にあります。

NetApp NPSは、それぞれDirect Connect接続とExpress Route接続を通じてAmazon AWSとMicrosoft Azureに接続されているため、インプレース分析モジュールを利用して、AmazonとAWS両方の分析クラスタからデータにアクセスできます。そのため、オンプレミスストレージとNPSストレージの両方でONTAP ソフトウェアが実行され、 "SnapMirror" NPSデータをオンプレミスクラスタにミラーリングし、オンプレミスと複数のクラウドにわたってハイブリッドクラウド分析を実現できます。

パフォーマンスを最大限に高めるために、通常は複数のネットワークインターフェイスと直接接続またはエクスプレスルートを使用してクラウドインスタンスからデータにアクセスすることを推奨します。ネットアップには、ほかにも、などのData Moverソリューションがあります "XCP" および "BlueXPのコピーと同期" お客様がアプリケーション対応でセキュア、コスト効率に優れたハイブリッドクラウドSparkクラスタを構築できるよう支援します。

主なユースケースごとに**Python**スクリプトを使用できます

以下の3つのPythonスクリプトは、テストした3つの主なユースケースに対応しています。1つ目は’ecimation_analysis _sparknlp.pyです

```
# 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}")
```

2番目のスクリプトは’kers_spark_horovod_Rossmann _ estimator.pyです

```
# 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()
```

3番目のスクリプトは'run_classification_Crito_spark.pyです

```
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))
```

まとめ

本ドキュメントでは、Apache Sparkのアーキテクチャ、お客様のユースケース、ネットアップストレージポートフォリオについて、ビッグデータ、最新の分析、AI、ML、DLに関連する情報をご紹介します。業界標準のベンチマークツールとお客様からの要望に基づくネットアップのパフォーマンス検証テストでは、NetApp Sparkソリューションは、ネイティブのHadoopシステムと比較して優れたパフォーマンスを実証しました。このレポートで紹介したお客様のユースケースとパフォーマンス結果を組み合わせることで、導入環境に適したSpark解決策 を選択できます。

追加情報の参照先

このTRでは次の資料を参照しています。

- Apache Sparkのアーキテクチャとコンポーネント

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

- Apache Sparkのユースケース

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

- Apacheの課題

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

- SparkのNLPです

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

- BERT

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

- 広告クリック予測のためのディープおよびクロスネットワーク

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

- FlexGroup

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

- ストリーミングETL

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

- NetApp EシリーズHadoop向けソリューション

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

- ネットアップの最新データ分析ソリューション

"データ分析ソリューション"

- SnapMirror

  "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のコピーと同期

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

- DataOpsツールキット

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

ビッグデータ分析データを人工知能に

TR-4732 ：『 Big data analytics data to 人工知能』

ネットアップ Karthikeyan Nagalingam

このドキュメントでは、ビッグデータ分析データと HPC データを AI に移行する方法について説明します。AI は、 NFS エクスポートを介して NFS データを処理しますが、お客様の AI データは、 HDFS 、 Blob 、 S3 ストレージなどのビッグデータ分析プラットフォームや、 GPFS などの HPC プラットフォームに格納されていることがよくあります。このホワイトペーパーでは、 NetApp XCP と NIPAM を使用して、ビッグデータ分析データと HPC データを AI に移行するためのガイドラインを説明します。また、ビッグデータや HPC から AI にデータを移行することで得られるビジネス上のメリットについても説明します。

概念とコンポーネント

ビッグデータ分析ストレージ

ビッグデータ分析は、 HDFS の主要なストレージプロバイダです。お客様は多くの場合、 Windows Azure Blob Storage 、 MapR File System （ MapR - FS ）、 S3 オブジェクトストレージなどの Hadoop 対応ファイルシステム（ HCFS ）を使用しています。

一般的な並列ファイルシステム

IBM の GPFS は、 HDFS の代わりとなるエンタープライズファイルシステムです。GPF は、アプリケーションがブロックサイズとレプリケーションレイアウトを決定できる柔軟性を備えており、優れたパフォーマンスと効率を実現します。

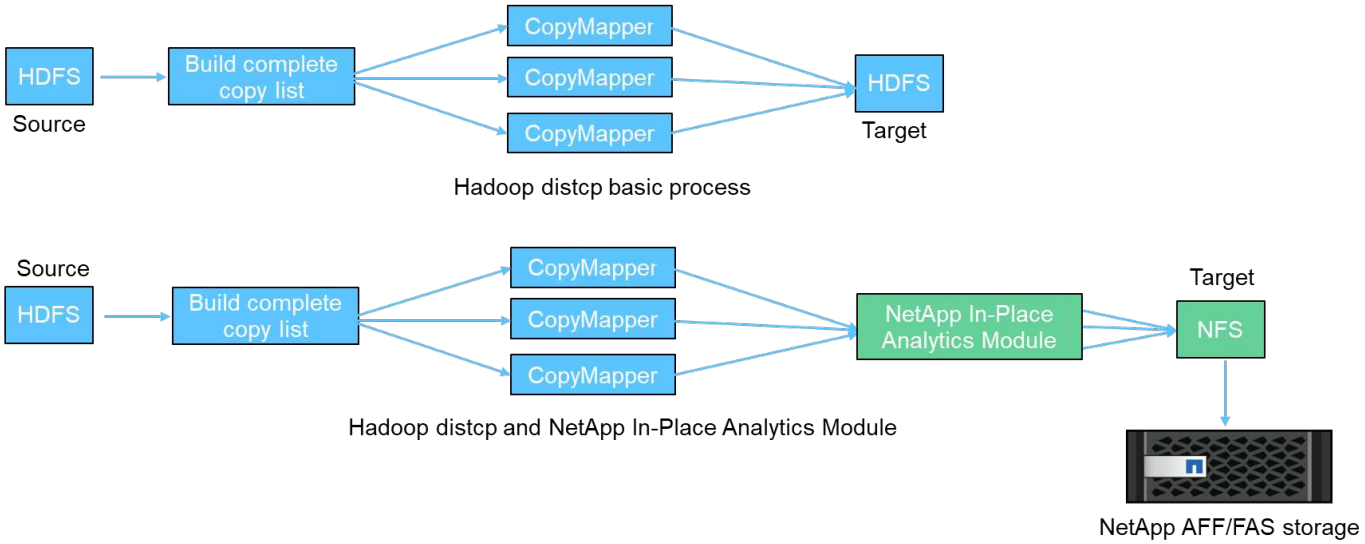
NetApp In-Place Analytics Module の略

NetApp In-Place Analytics Module （ NIPAM ）は、 NFS データにアクセスする Hadoop クラスタのドライバ

として機能します。接続プール、 NFS InputStream 、ファイル・ハンドル・キャッシュ、 NFS OutputStream の 4 つのコンポーネントで構成されています。詳細については、を参照してください "TR-4382 ：『 NetApp In-Place Analytics Module 』"

Hadoop 分散コピー

Hadoop Distributed Copy （ DistCp ）は、クラスタ間およびクラスタ内の大規模なコピー作業に使用される分散コピーツールです。このツールは、データの配信、エラー処理、およびレポートに MapReduce を使用します。ファイルとディレクトリのリストが展開され、タスクをマッピングしてソースリストからデータをコピーするように入力されます。次の図は、 HDFS と HDFS 以外の間の DistCp 処理を示しています。



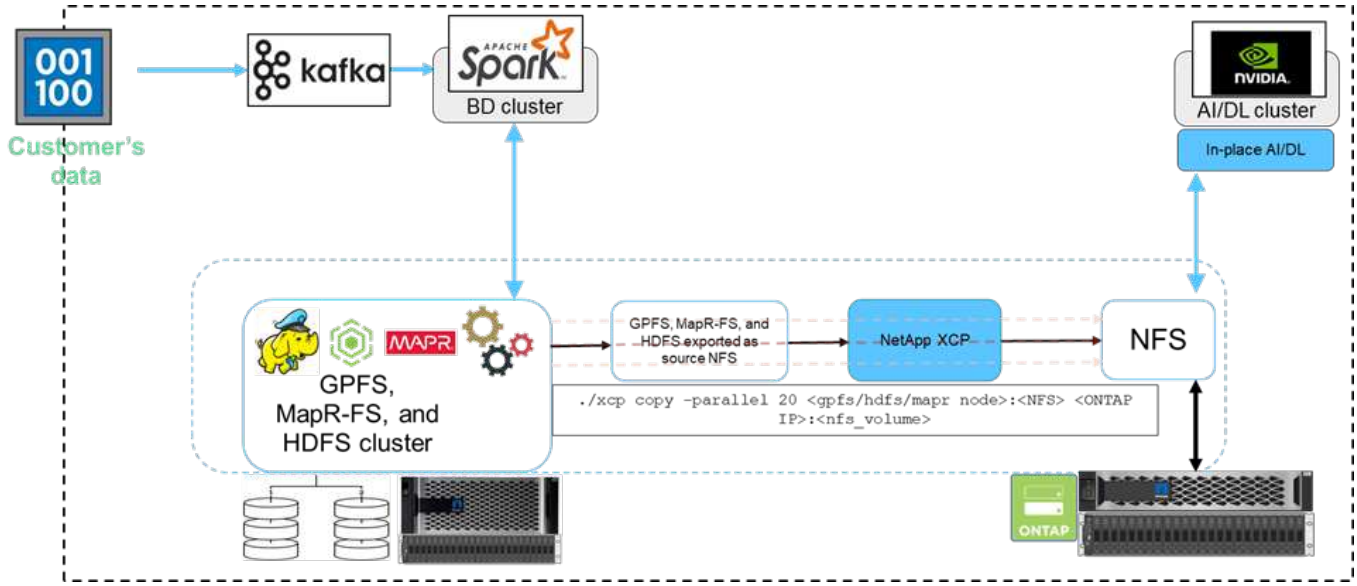
Hadoop DistCp は、追加のドライバを使用せずに 2 つの HDFS システム間でデータを移動します。ネットアップは HDFS 以外のシステム向けのドライバを提供しています。NFS デスティネーションの場合、 NIPAM は、データのコピー時に Hadoop DistCp が NFS デスティネーションとの通信に使用するデータをコピーするためのドライバを提供します。

NetApp Cloud Volumes Service の略

NetApp Cloud Volumes Service は、卓越したパフォーマンスを発揮するクラウドネイティブのファイルサービスです。このサービスを利用すると、お客様はリソースのスピンアップとスピンダウンを迅速に行い、ネットアップの機能を使用して生産性を高め、スタッフのダウンタイムを短縮し、市場投入までの期間を短縮できます。Cloud Volumes Service は、データセンター全体の設置面積を削減し、ネイティブのパブリッククラウドストレージを消費しないため、ディザスタリカバリやクラウドへのバックアップに最適な選択肢です。

NetApp XCP

NetApp XCP は、高速で信頼性の高いネットアップおよびネットアップからネットアップへのデータ移行を可能にするクライアントソフトウェアです。このツールは、あらゆる NAS システムからネットアップストレージコントローラに大量の非構造化 NAS データをコピーするために設計されています。XCP Migration Tool では、マルチコアのマルチチャネル I/O ストリーミングエンジンが使用されており、データの移行、ファイルやディレクトリの一覧表示、スペースレポートなど、多数の要求を並行して処理できます。これは、デフォルトのネットアップデータ移行ツールです。XCP を使用して、 Hadoop クラスタと HPC から NetApp NFS ストレージにデータをコピーできます。次の図は、 Hadoop クラスタと HPC クラスタから XCP を使用した NetApp NFS ボリュームへのデータ転送を示しています。



NetApp BlueXPのコピーと同期

NetApp BlueXPのコピーと同期は、ハイブリッドデータレプリケーションのソフトウェアサービスで、オンプレミスストレージとクラウドストレージ間でNFS、S3、CIFSのデータをシームレスかつセキュアに転送、同期します。このソフトウェアは、データの移行、アーカイブ、コラボレーション、分析などに使用されます。データの転送が完了すると、BlueXPのCopy and Syncはソースとデスティネーションの間でデータを継続的に同期します。次に、デルタを転送します。また、自社ネットワーク内、クラウド内、オンプレミス内でもデータを保護できます。このソフトウェアは従量課金制モデルをベースとしており、対費用効果の高い解決策を提供し、データ転送の監視とレポートの機能を提供します。

お客様の課題

AI 運用のためにビッグデータ分析からデータにアクセスしようとすると、お客様は次のような課題に直面することがあります。

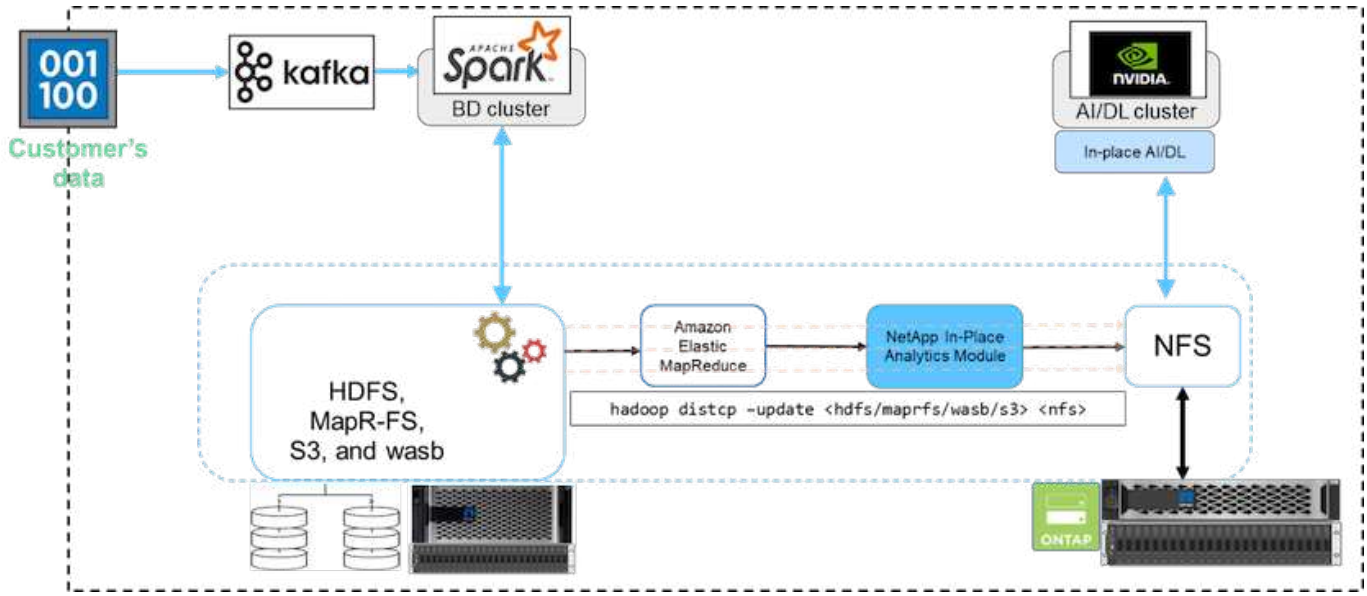
- お客様のデータは、データレイクリポジトリに格納されています。データレイクには、構造化データ、非構造化データ、半構造化データ、ログ、マシン間データなど、さまざまなタイプのデータを格納できます。これらのデータタイプはすべて AI システムで処理する必要があります。
- AI は Hadoop ファイルシステムには対応していません。一般的な AI アーキテクチャでは、 HDFS データと HCFS データに直接アクセスできないため、 AI に対応したファイルシステム（ NFS ）に移行する必要があります。
- データレイクのデータを AI に移行するには、通常、特別なプロセスが必要です。データレイク内のデータ量は非常に多くなる可能性があります。効率性、スループット、コスト効率に優れた方法でデータを AI システムに移動する必要がある。
- データを同期中です。お客様がビッグデータプラットフォームと AI の間でデータを同期したい場合は、ビッグデータを使用して処理されたデータを分析処理に使用できることがあります。

Data Mover の解決策

ビッグデータクラスタでは、 MapR -FS 、 Windows Azure Storage Blob 、 S3 、 Google ファイルシステムなどの HDFS または HCFS にデータが格納されます。ソース側で「 hadoop distcp 」コマンドを使用し、データを NIPAM の助けを得て NetApp

ONTAP NFS エクスポートにコピーするソースとして、 HDFS 、 MapR FS 、および S3 を使用してテストを実施しました。

次の図は、 HDFS ストレージで稼働している Spark クラスタから、 NVIDIA が AI 処理を行えるようにするための NetApp ONTAP NFS ボリュームへの、一般的なデータ移動を示しています。



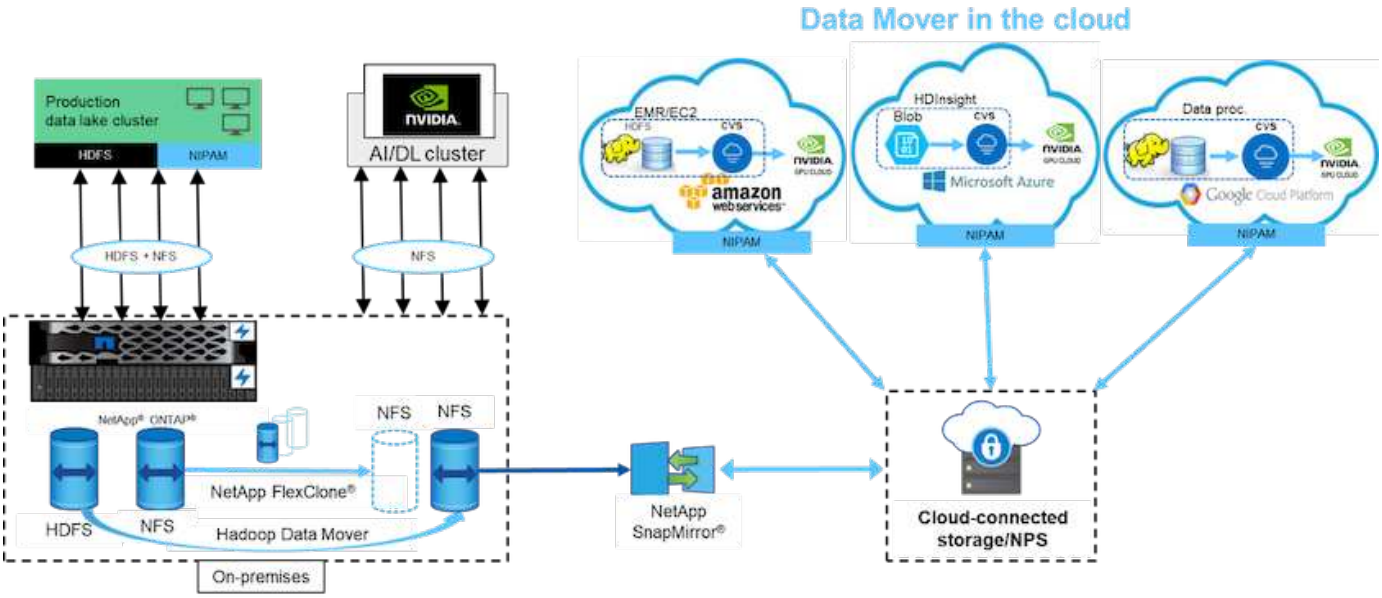
「 hadoop distcp 」コマンドは、 MapReduce プログラムを使用してデータをコピーします。NIPAM は MapReduce と連携して、データをコピーする際の Hadoop クラスタのドライバとして機能します。NIPAM では、 1 つのエクスポートのために複数のネットワークインターフェイスに負荷を分散できます。このプロセスにより、 HDFS または HCFS から NFS にデータをコピーするときに、複数のネットワークインターフェイスにデータを分散させることにより、ネットワークスループットが最大になります。

NIPAM は MapR でサポートまたは認定されていません。

AI 向け Data Mover 解決策

Data Mover 解決策 for AI は、お客様のニーズに基づいて AI 運用の Hadoop データを処理します。ネットアップは、 NIPAM を使用して HDFS から NFS にデータを移動します。あるユースケースでは、クラウド内の GPU クラウドインスタンスからデータを処理するために、データをオンプレミスの NFS に移動し、別のお客様が Windows Azure ストレージ Blob から Cloud Volumes Service に移動する必要がありました。

次の図は、 Data Mover の解決策の詳細を示しています。



Data Mover 解決策を構築するには、次の手順が必要です。

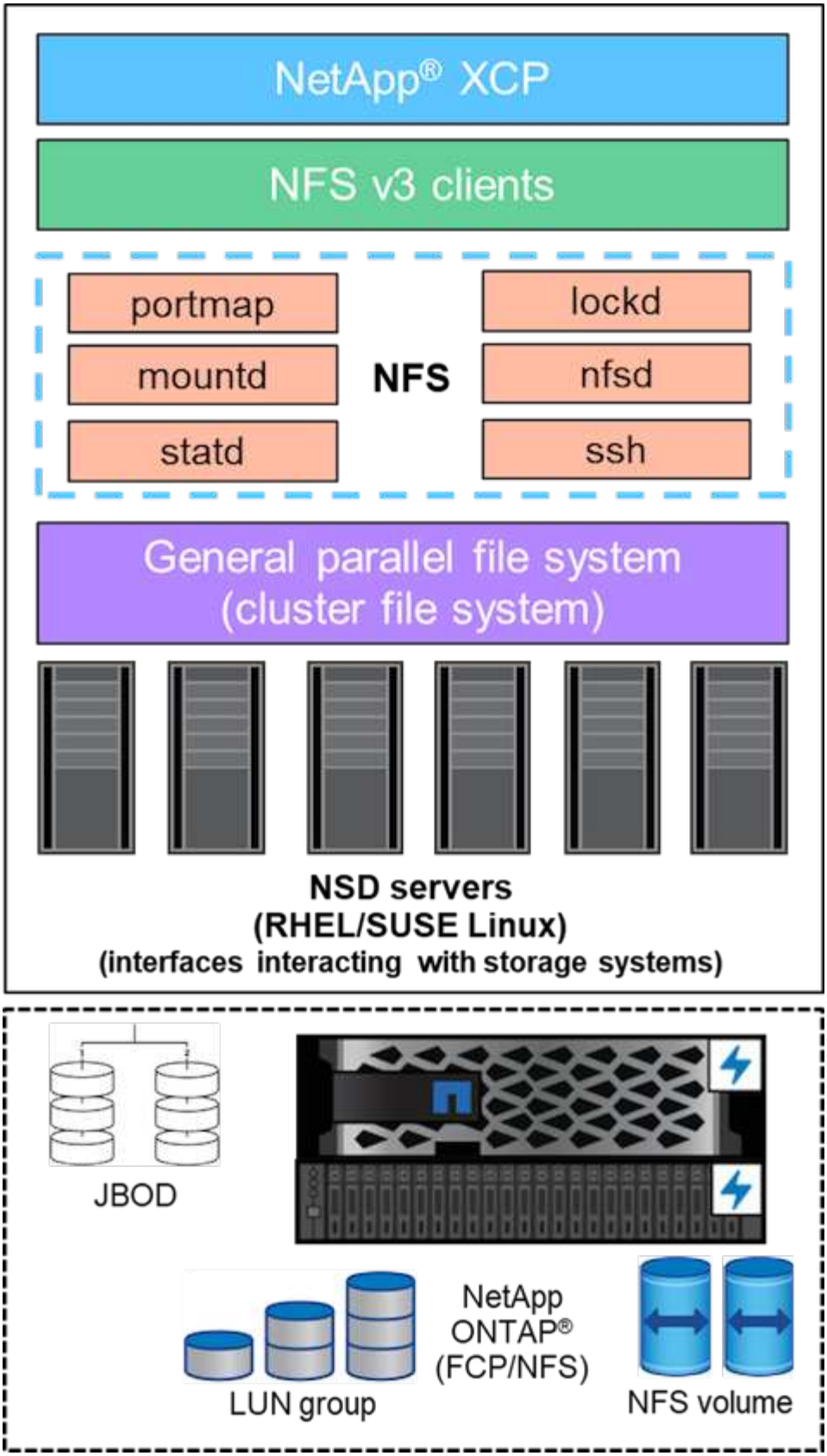
1. ONTAP SAN は HDFS を提供し、 NAS は NIPAM 経由で本番データレイククラスタに NFS ボリュームを提供します。
2. お客様のデータは HDFS と NFS にあります。NFS データは、ビッグデータ分析や AI 処理に使用される他のアプリケーションの本番データにすることができます。
3. NetApp FlexClone テクノロジは、本番用 NFS ボリュームのクローンを作成し、オンプレミスの AI クラスタにプロビジョニングします。
4. HDFS SAN LUN からのデータは、 NIPAM および「 hadoop distcp 」コマンドによって NFS ボリュームにコピーされます。NIPAM は、複数のネットワークインターフェイスの帯域幅を使用してデータを転送します。この処理によってデータコピー時間が短縮され、より多くのデータを転送できるようになります。
5. どちらの NFS ボリュームも、 AI 処理用に AI クラスタにプロビジョニングされます。
6. オンプレミスの NFS データを処理するために、クラウド内の GPU を使用する場合は、 NFS ボリュームが NetApp SnapMirror テクノロジを使用して NetApp Private Storage （ NPS ）にミラーリングされ、 GPU のクラウドサービスプロバイダにマウントされます。
7. お客様は、 GPU のデータをクラウドサービスプロバイダから EC2 / EMR 、 HDInsight 、または DataProc サービスで処理したいと考えています。Hadoop データムーバーは、 NIPAM および「 hadoop distcp 」コマンドを使用して、 Hadoop サービスから Cloud Volume サービスにデータを移動します。
8. Cloud Volumes Service データは、 NFS プロトコルを使用して AI にプロビジョニングされます。 AI で処理されたデータは、オンプレミスの場所に送信し、 NIPAM 、 SnapMirror 、 NPS を通じて、 NVIDIA クラスタに加えてビッグデータ分析にも使用できます。

このシナリオでは、 NAS システム内のファイル数が多く、オンプレミスのネットアップストレージコントローラで AI 処理を実行するために必要なリモートサイトにデータが配置されています。このシナリオでは、 XCP Migration Tool を使用してデータをより高速に移行することをお勧めします。

ハイブリッドユースケースでは、BlueXPのCopy and Syncを使用してオンプレミスのデータをNFS、CIFS、S3のデータからクラウドに移行し、NVIDIAクラスタ内のGPUなどを使用してAI処理を行うことができます。BlueXPのコピーと同期とXCP移行ツールは、NetApp ONTAP NFSへのNFSデータ移行に使用します。

NetApp ONTAP NFS への GPF

この検証では、 4 台のサーバを Network Shared Disk （ NSD; ネットワーク共有ディスク）サーバとして使用して、 GPFS 用の物理ディスクを提供しました。GPF は、次の図に示すように、 NFS エクスポートとしてエクスポートするために NSD ディスクの上に作成されます。XCP を使用して、 GPFS でエクスポートされた NFS から NetApp NFS ボリュームにデータをコピーしました。



GPF の要点

GPFS では、次のノードタイプが使用されます。

- * 管理ノード。 * ノード間の通信に管理コマンドが使用するノード名を含むオプションのフィールドを指定します。たとえば ' 管理ノード mastr-51.netapp.com` は ' クラスタ内の他のすべてのノードにネットワーク・チェックを渡すことができます
- * クォーラムノード。クォーラムの取得元のノードのプールにノードが含まれているかどうかを判断します。少なくとも 1 つのノードがクォーラムノードとして必要です。
- * マネージャノード。 * ノードがノードプールの一部であるかどうかを示します。このプールからファイルシステムマネージャとトークンマネージャを選択できます。複数のノードをマネージャーノードとして定義することをお勧めします。マネージャとして指定するノードの数は、ワークロードと所有する GPFS サーバライセンスの数によって異なります。大規模な並列ジョブを実行する場合は、 Web アプリケーションをサポートする 4 ノードクラスタよりも多くのマネージャノードが必要になることがあります。
- *NSD サーバ。 *GPFS で使用する各物理ディスクを準備するサーバ。
- * プロトコルノード。 *Secure Shell （ SSH; セキュアシェル）プロトコルを介して、 NFS と GPFS データを直接共有するノード。このノードには GPFS サーバーライセンスが必要です。

GPFS 、 **NFS** 、および **XCP** の操作のリスト

このセクションでは、 GPFS を作成し、 NFS エクスポートとして GPFS をエクスポートし、 XCP を使用してデータを転送する操作について説明します。

GPFS を作成します

GPFS を作成するには、次の手順を実行します。

1. Linux 版のスペクトルスケールデータアクセスをいずれかのサーバにダウンロードしてインストールします。
2. すべてのノードに前提条件パッケージ（シェフなど）をインストールし、すべてのノードで Security-Enhanced Linux （ SELinux ）を無効にする。
3. インストールノードをセットアップし、管理ノードと GPFS ノードをクラスタ定義ファイルに追加します。
4. マネージャーノード、クォーラムノード、 NSD サーバー、および GPFS ノードを追加します。
5. GUI 、管理ノード、 GPFS ノードを追加し、必要に応じて GUI サーバを追加します。
6. 別の GPFS ノードを追加し、すべてのノードのリストを確認します。
7. クラスタ定義ファイル内のすべての GPFS ノードで設定するクラスタ名、プロファイル、リモートシェルバイナリ、リモートファイルコピーバイナリ、およびポート範囲を指定します。
8. GPFS 構成設定を表示し、追加の管理ノードを追加します。
9. データ収集を無効にし、 IBM サポートセンターにデータパッケージをアップロードします。
10. インストールの前に、 NTP を有効にし、構成を事前確認します。
11. NSD ディスクを構成、作成、およびチェックする。
12. GPFS を作成します。
13. GPFS をマウントします。
14. GPFS に必要な権限を確認して提供します。
15. 「 dd 」コマンドを実行して、 GPFS の読み書きを確認します。

GPFS を NFS にエクスポートする

GPFS を NFS にエクスポートするには、次の手順を実行します。

1. GPFS を /etc/exports ファイルを介して NFS としてエクスポートします
2. 必要な NFS サーバパッケージをインストールします。
3. NFS サービスを開始します。
4. NFS クライアントを検証するために、 GPFS 内のファイルをリストします。

NFS クライアントを設定します

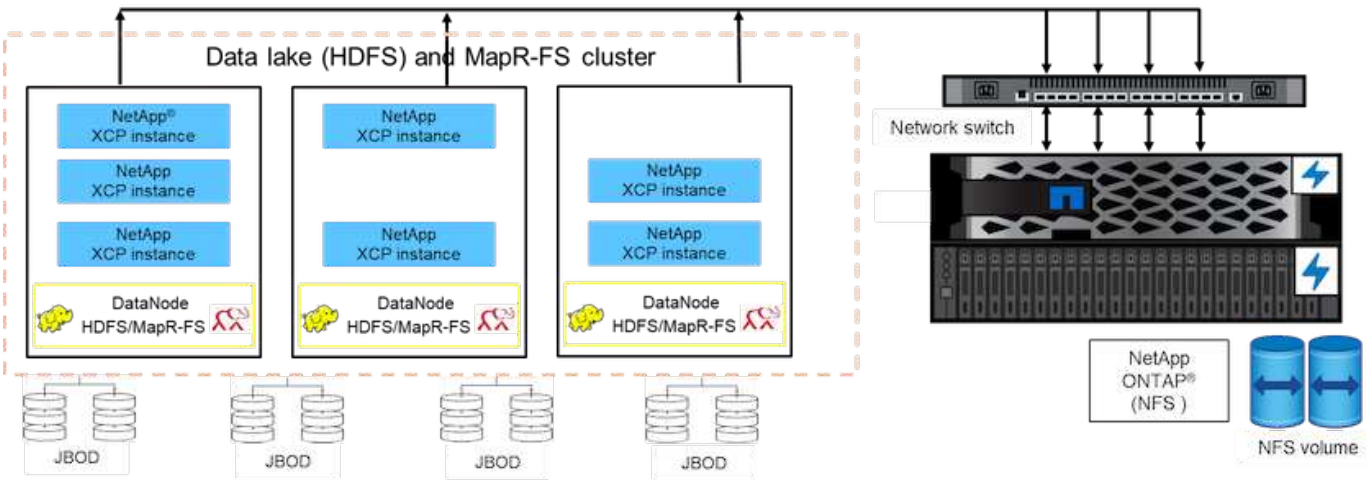
NFS クライアントを設定するには、次の手順を実行します。

1. GPFS を /etc/exports ファイルを使用して NFS としてエクスポートします。
2. NFS クライアントサービスを開始します。
3. NFS クライアントで NFS プロトコルを使用して GPFS をマウントします。
4. NFS Mounted フォルダ内の GPFS ファイルのリストを検証します。
5. XCP を使用して、 GPFS エクスポート NFS から NetApp NFS にデータを移動します。
6. NFS クライアントで GPFS ファイルを検証します。

HDFS と MapR - FS を ONTAP NFS に接続

この解決策では、ネットアップがデータレイク（ HDFS ）と MapR クラスタデータから ONTAP NFS へのデータ移行を検証しました。データは MapR - FS と HDFS に存在します。NetApp XCP は、 HDFS や MapR FS などの分散ファイルシステムから ONTAP NFS にデータを直接移行する新しい機能を導入しました。XCP は、非同期スレッドと HDFS C API 呼び出しを使用して、 MapR FS と HDFS の間でデータを通信および転送します。

次の図は、データレイク（ HDFS ）と MapR FS から ONTAP NFS へのデータ移行を示しています。この新機能により、ソースを NFS 共有としてエクスポートする必要がなくなります。



お客様が **HDFS** と **MapR FS** から **NFS** に移行するのはなぜですか。

Cloudera や Hortonworks などの Hadoop ディストリビューションのほとんどは HDFS ディストリビューションと MapR ディストリビューションを使用してデータを格納しています。HDFS と MapR - FS データは、機械学習（ ML ）とディープラーニング（ DL ）に活用できるデータサイエンティストにとって価値ある分析情報を提供します。HDFS と MapR FS のデータは共有されないため、他のアプリケーションでは使用できません。顧客は、共有データ、特に顧客の機密データが複数のアプリケーションで使用される銀行業界を探しています。最新バージョンの Hadoop （ 3.x 以降）は NFS データソースをサポートしており、サードパーティ製ソフトウェアを追加することなくアクセスできます。新しい NetApp XCP 機能を使用すると、データを HDFS および MapR FS から NetApp NFS に直接移動して、複数のアプリケーションへのアクセスを提供できます

テストは Amazon Web Services （ AWS ）で実施され、 MapR ノード 12 台と NFS サーバ 4 台を使用した初期パフォーマンステストで MapR FS から NFS にデータを転送しました。

	数量	サイズ	**vCPU**	メモリ	ストレージ	ネットワーク
NFS サーバ	4.	i3en.24xlarge のサイズ	96	488GiB	8 台の 7 、 500 NVMe SSD	100
MapR ノード	12.	I3en. 12xlarge	48	384GiB	7 、 500 NVMe SSD × 4	50

初期テストでは 20Gbps のスループットを達成し、 2PB のデータを 1 日あたり転送可能でした。

HDFS を NFS にエクスポートせずに HDFS データを移行する方法の詳細については、の「 Deployment Steps - NAS 」を参照してください "TR-4863 ：『 Best Practice Guidelines for NetApp XCP - Data Mover 、 File Migration 、 and Analytics 』"。

ビジネス上のメリット

ビッグデータ分析から AI にデータを移動することには、次のようなメリットがあります。

- 異なる Hadoop ファイルシステムと GPFS から Unified NFS ストレージシステムにデータを抽出する機能
- Hadoop 統合によりデータ転送を自動化します
- Hadoop ファイルシステムからデータを移動するためのライブラリ開発コストを削減することです
- NIPAM を使用して、 1 つのデータソースからの複数のネットワークインターフェイスの集約スループットによる最大パフォーマンス
- データを転送するための、スケジュールされた方法とオンデマンドの方法
- ONTAP データ管理ソフトウェアを使用した、ユニファイド NFS データ用のストレージ効率化およびエンタープライズ管理機能
- データ転送用の Hadoop メソッドにより、データ移動のコストがゼロになります

NFS に対する **GPF -** 詳細な手順

このセクションでは、 NetApp XCP を使用して GPFS を設定し、 NFS にデータを移動

するために必要な詳細な手順について説明します。

GPFS を設定します

1. いずれかのサーバに Linux 用 Spectrum Scale Data Access をダウンロードしてインストールします。

```
[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. すべてのノードに前提条件パッケージ（シェフとカーネルヘッダーを含む）をインストールします。

```
[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. すべてのノードで SELinux を無効にする。

```
[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. インストールノードをセットアップします。

```
[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. 管理ノードと GPFS ノードをクラスタ定義ファイルに追加します。

```
[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. マネージャノードと GPFS ノードを追加します。

```
[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. クォーラムノードと GPFS ノードを追加します。

```
[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 サーバと GPFS ノードを追加します。

```
[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 、管理ノード、および GPFS ノードを追加します。

```
[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. 別の GUI サーバを追加します。

```
[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. 別の GPFS ノードを追加します。

```
[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. すべてのノードを検証およびリストします。

```
[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. クラスタ定義ファイルでクラスタ名を指定します。

```
[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. プロファイルを指定します。

```
[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. GPFS で使用するリモートシェルバイナリを指定します。引数には -r を使用します。

```
[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. GPFS で使用するリモートファイルコピーバイナリを指定します。「 -rc 引数」を使用します。

```
[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. すべての GPFS ノードに設定するポート範囲を指定します。「 -e 引数」を使用します。

```
[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 構成設定を表示します。

```
[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. 管理ノードを追加

```
[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. データ収集を無効にし、 IBM サポートセンターにデータパッケージをアップロードします。

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

21. NTP を有効にします。

```
[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. インストール前に設定を事前確認します。

```
[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 ディスクを設定します。

```
[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 ディスクを作成します。

```
[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 ディスクのステータスを確認します。

```
[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. GPFS を作成します。

```
[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. GPFS をマウントします。

```
[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. GPFS に必要な権限を確認して付与します。

```
[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. 「 dd 」コマンドを実行して、 GPFS の読み取りと書き込みを確認します。

```
[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]#
```

GPFS を NFS にエクスポートする

GPFS を NFS にエクスポートするには、次の手順を実行します。

1. GPFS を /etc/exports ファイルを使用して NFS としてエクスポートします。

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

2. 必要な NFS サーバパッケージをインストールします。

```
[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 サービスを開始します。

```
[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. NFS クライアントを検証するために、 GPFS 内のファイルをリストします。

```
[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 クライアントを設定します

NFS クライアントを設定するには、次の手順を実行します。

1. NFS クライアントにパッケージをインストールします。

```
[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 クライアントサービスを開始します。

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

3. NFS クライアントで NFS プロトコルを使用して GPFS をマウントします。

```
[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 マウントフォルダ内の GPFS ファイルのリストを確認します。

```
[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. XCP を使用して、 GPFS でエクスポートされた NFS から NetApp NFS にデータを移動します。

```
[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 クライアントで GPFS ファイルを検証します。

```
[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 - FS から ONTAP NFS へ

このセクションでは、 NetApp XCP を使用して MapR FS データを ONTAP NFS に移動するために必要な手順について説明します。

1. MapR ノードごとに 3 つの LUN をプロビジョニングし、すべての MapR ノードの LUN の所有権を付与します。
2. インストール時に、 MapR FS に使用されている MapR クラスタディスク用に新しく追加された LUN を選択します。
3. に従って MapR クラスタをインストールします "MapR 6.1 のドキュメント"。
4. 「 hadoop jar xxx 」などの MapReduce コマンドを使用して、基本的な Hadoop 動作をチェックします。
5. MapR - FS で顧客データを保持します。たとえば、 Teragen を使用して MapR -FS に約 1 テラバイトのサンプルデータを生成しました。
6. MapR - FS を NFS エクスポートとして設定します。
   a. すべての MapR ノードで nlockmgr サービスを無効にします。

```
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. MapR から特定のフォルダをエクスポートします。これは、「 /opt/MapR /conf/exports 」ファイルのすべての MapR ノードにあります。サブフォルダをエクスポートするときは、異なる権限を持つ親フォルダをエクスポートしないでください。

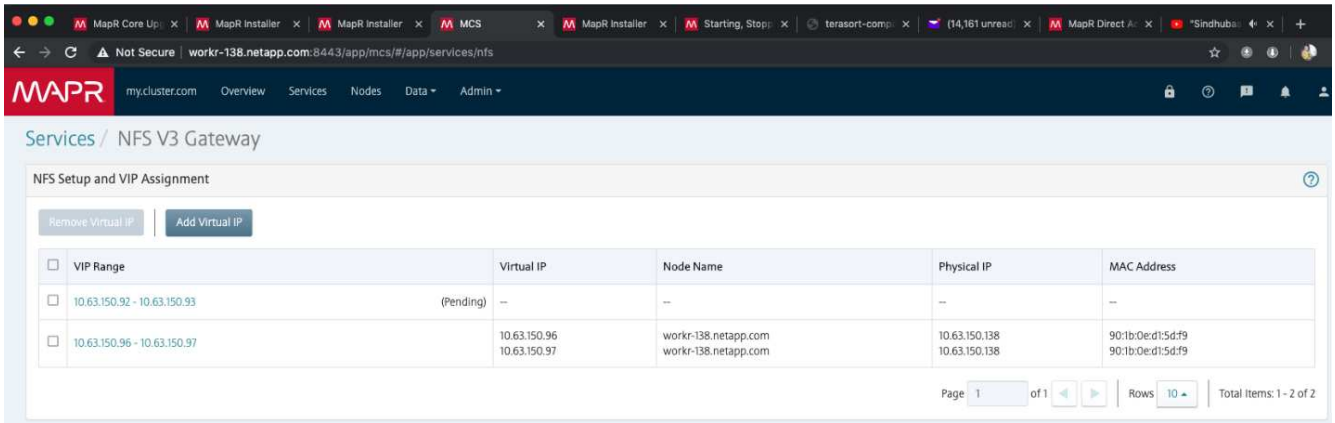
```
[mapr@workr-138 ~]$ cat /opt/mapr/conf/exports
# Sample Exports file
# for /mapr exports
# <Path> <exports_control>
#access_control -> order is specific to default
# list the hosts before specifying a default for all
#  a.b.c.d,1.2.3.4(ro) d.e.f.g(ro) (rw)
#  enforces ro for a.b.c.d & 1.2.3.4 and 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 ~]$
```

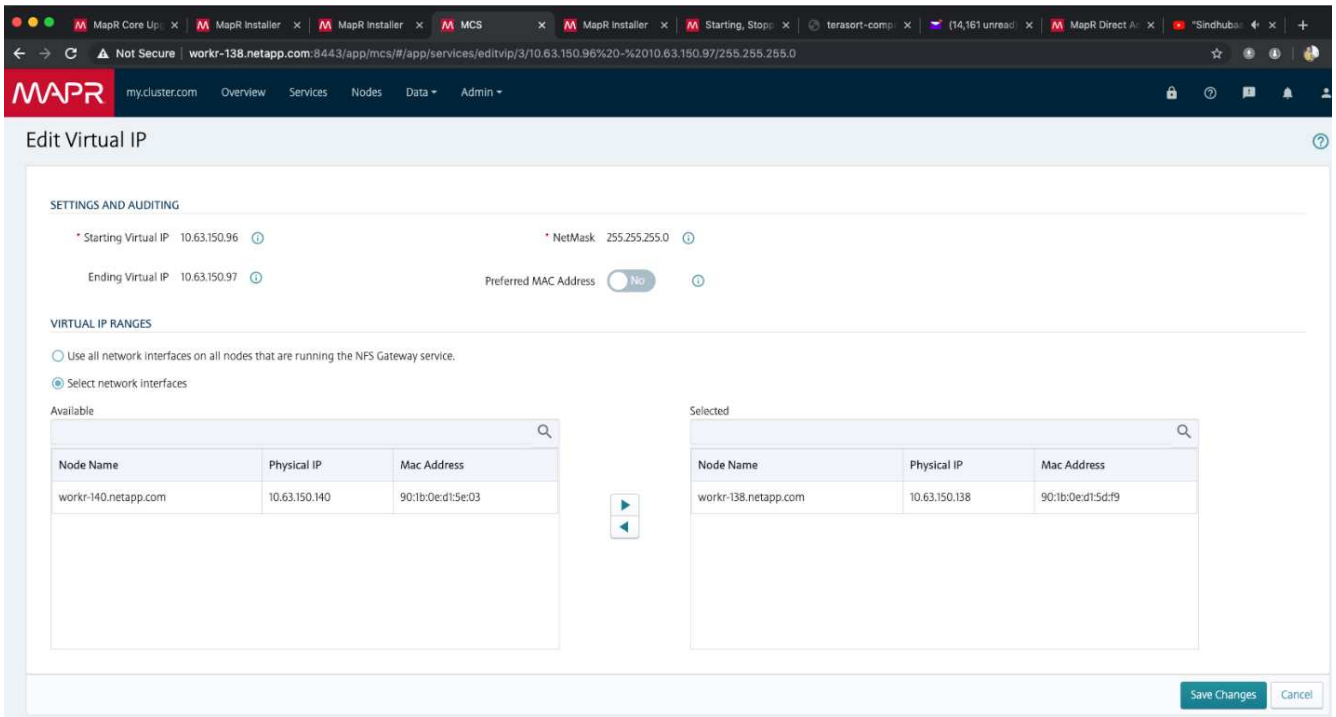
7. MapR - FS NFS サービスを更新します。

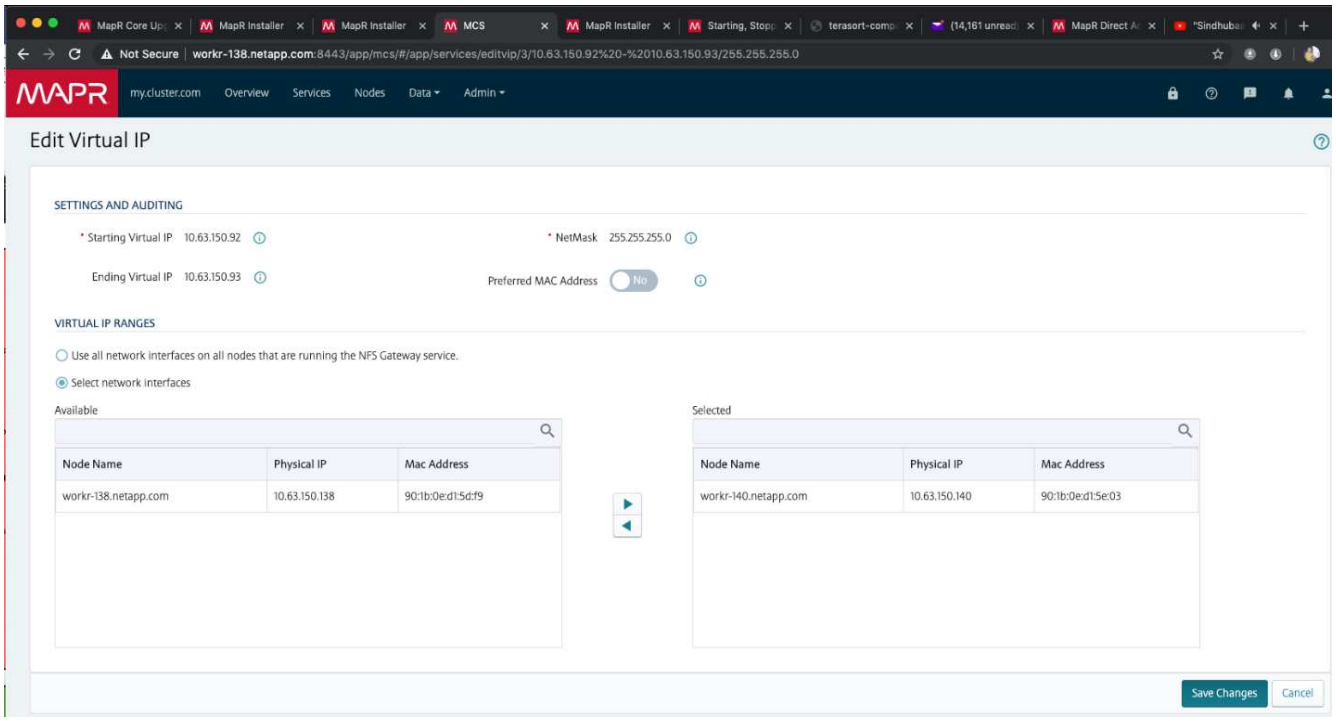
```
root@workr-138: tmp$ maprcli nfsmgmt refreshexports
ERROR (22) -  You do not have a ticket to communicate with
127.0.0.1:9998. Retry after obtaining a new ticket using maprlogin
root@workr-138: tmp$ su - mapr
[mapr@workr-138 ~]$ maprlogin password -cluster my.cluster.com
[Password for user 'mapr' 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. 仮想 IP 範囲を MapR クラスタ内の特定のサーバまたはサーバセットに割り当てます。次に、 MapR クラスタが NFS データアクセス用の特定のサーバに IP を割り当てます。IP を使用することで高可用性が実現します。つまり、特定の IP を持つサーバまたはネットワークで障害が発生した場合に、 IP 範囲内の次の IP を NFS アクセスに使用できます。

> すべての MapR ノードからの NFS アクセスを提供する場合は、各サーバに一連の仮想 IP を割り当て、各 MapR ノードのリソースを NFS データアクセスに使用できます。







9. 各 MapR ノードに割り当てられている仮想 IP を確認し、 NFS データアクセスに使用します。

```
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 をエクスポートした MapR FS をマウントするには、 NFS の動作を確認するために割り当てられた仮想 IP を使用します。ただし、 NetApp XCP を使用したデータ転送では、この手順は必要ありません。

```
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 ゲートウェイから ONTAP NFS にデータを転送するように NetApp XCP を設定します。
    a. XCP のカタログの場所を設定します。

```
[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. ライセンスファイルを「 /opt/NetApp/xFiles/XCP 」にコピーします。

```
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 コマンドを使用して XCP をアクティブにします。

d. ソースで NFS エクスポートを確認します。

```
[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. 複数のソース IP と複数のデスティネーション IP （ ONTAP LIF ）から、複数の MapR ノードから XCP を使用してデータを転送します。

```
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. ストレージコントローラ上の負荷分散を確認します。

```
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
```

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- NetApp In-Place Analytics Module Best Practices 』を参照してください

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

- 『 NetApp FlexGroup Volume Best Practices and Implementation Guide 』にある、ボリュームへの移行に関するセクション

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

- ネットアップの製品マニュアル

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

ConFluent Kafka のベストプラクティスをご確認ください

TR-4912 ：『 Best Practices guidelines for ConFluent Kafka Tiered Storage with NetApp 』

Karthikeyan Nagalingam 、 Joseph Kandatilparamil 、 NetApp Rankesh Kumar 、 Conluent

Apache Kafka は、 1 日に数兆ものイベントを処理できる、コミュニティで分散されたイベントストリーミングプラットフォームです。Kafka は、当初はメッセージングキューとして構築され、分散コミットログの抽象化に基づいています。Kafka は、 2011 年に LinkedIn で作成されオープンソースとなって以来、メッセージキューから本格的なイベントストリーミングプラットフォームへと進化してきました。Confluent Platform で Apache Kafka を配布します。Coneluent Platform は、 Kafka を補完するための追加のコミュニティ機能と商用機能を備えています。これらの機能は、本番環境の運用者と開発者の両方のストリーミングエクスペリエンスを大規模に向上させるように設計されています。

本ドキュメントでは、次のコンテンツを提供することで、ネットアップのオブジェクトストレージ製品での上位階層型ストレージの使用に関するベストプラクティスのガイドラインについて説明します。

- ネットアップオブジェクトストレージとの競合検証– NetApp StorageGRID
- 階層型ストレージのパフォーマンステスト
- ネットアップのストレージシステムを使用する場合のベストプラクティスのガイドラインを参照してください

階層型ストレージが優れている理由

競合製品は、多くのアプリケーション、特にビッグデータ、分析、ストリーミングワークロードに対応するデフォルトのリアルタイムストリーミングプラットフォームになっています。階層型ストレージを使用すると、ユーザは Conluent プラットフォームのストレージからコンピューティングを分離できます。データをより対費用効果の高い方法で保存し、ほぼ無制限のデータ量を保存し、オンデマンドでワークロードを増減できます。また、データやテナントのリバランシングなどの管理タスクが容易になります。S3 互換のストレージシステムでは、これらすべての機能を活用して、すべてのイベントのデータを 1 箇所で民主化できるため、複雑なデータエンジニアリングは不要です。Kafka に階層化ストレージを使用すべき理由については、を参照してください "この記事は流暢なものです"。

階層化ストレージに **NetApp StorageGRID** を使用する理由

StorageGRID は、業界をリードするネットアップのオブジェクトストレージプラットフォームです。StorageGRID は、ソフトウェアで定義されるオブジェクトベースのストレージ解決策で、 Amazon Simple Storage Service （ S3 ） API などの業界標準のオブジェクト API をサポートします。StorageGRID は、大規模な非構造化データを格納および管理し、セキュアでデータ保持性に優れたオブジェクトストレージを実現します。コンテンツは適切なタイミングで適切な場所の適切なストレージ階層に配置されるため、グローバルに分散されるリッチメディアのワークフローを最適化し、コストを削減できます。

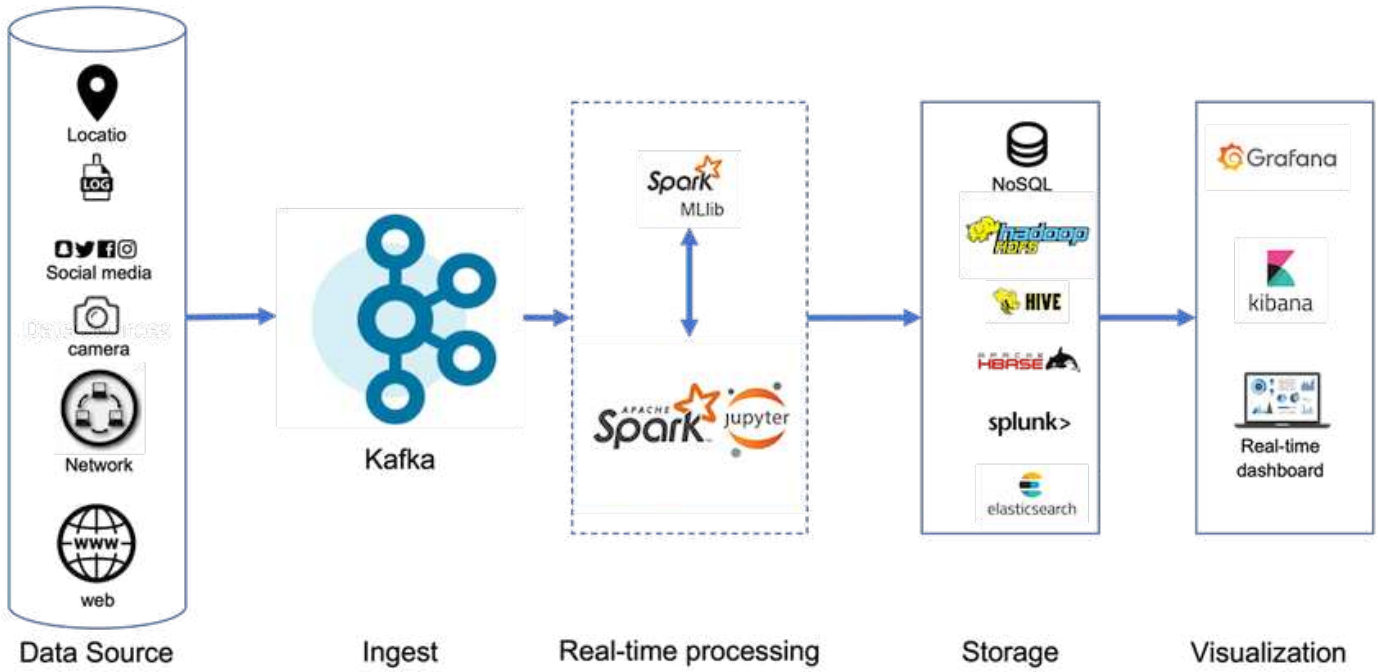
StorageGRID の最大の差別化要因は、ポリシーベースのデータライフサイクル管理を可能にする情報ライフサイクル管理（ ILM ）ポリシーエンジンです。ポリシーエンジンでは、メタデータを使用して、データの有効期間全体の格納方法を管理できます。これにより、最初はパフォーマンスを最適化し、データの経過に応じてコストと保持性を自動的に最適化できます。

階層型ストレージを有効にします

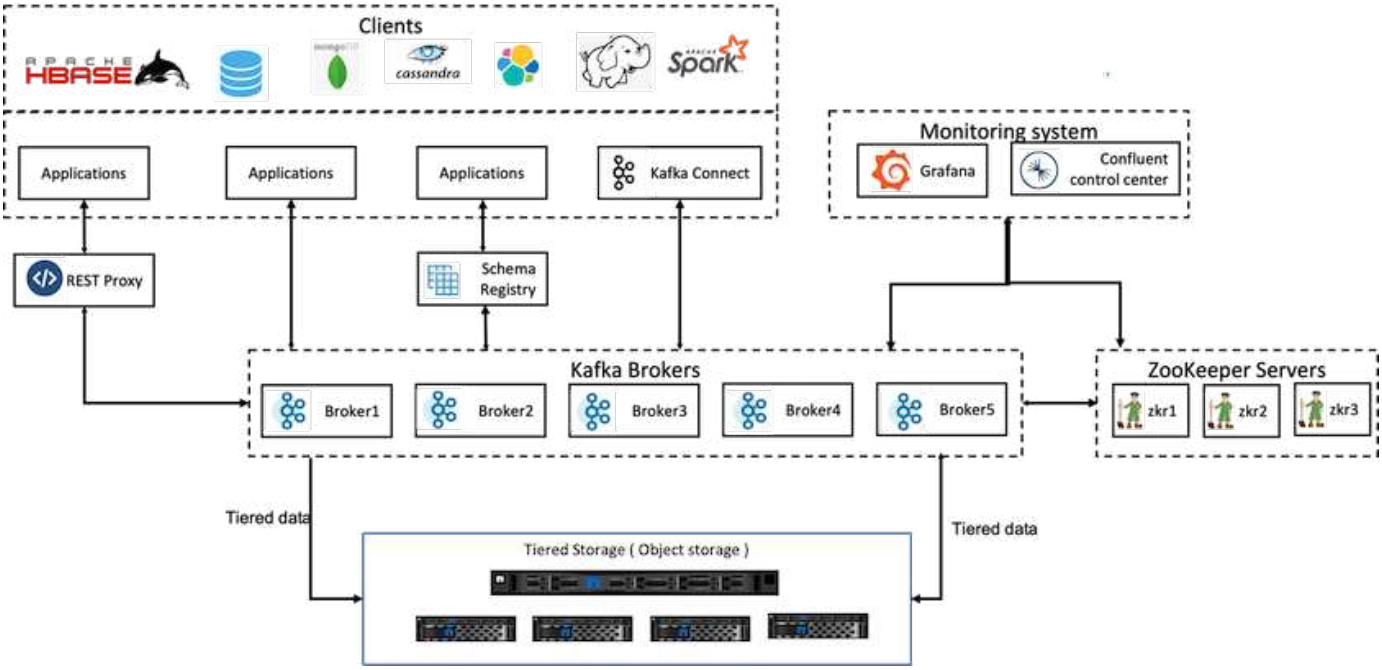
階層型ストレージの基本的な目的は、データストレージのタスクをデータ処理から分離することです。この分離によって、データストレージ階層とデータ処理階層を別々に拡張しやすくなります。

流暢な層ストレージの解決策は、 2 つの要因で競合する必要があります。まず、リスト操作の不整合やオブジェクトを使用できないことがあるなど、一般的なオブジェクトストアの整合性と可用性のプロパティを回避または回避する必要があります。次に、階層型ストレージと Kafka のレプリケーションとフォールトトレランスモデルの間の相互作用を正しく処理する必要があります。これには、ゾンビのリーダーが引き続きオフセット範囲を階層化する可能性も含まれます。ネットアップのオブジェクトストレージは、整合性のあるオブジェクトを使用できるようにするとともに、 HA モデルによって、オフセット範囲を階層化するために使用できるストレージにします。ネットアップのオブジェクトストレージを使用すると、オブジェクトの可用性に一貫性があり、オフセット範囲を階層化するために使用される階層化ストレージには HA モデルが採用されています。

階層型ストレージでは、ストリーミングデータの末尾付近での低レイテンシの読み取りや書き込みにハイパフォーマンスプラットフォームを使用できます。また、 NetApp StorageGRID などの低コストで拡張性に優れたオブジェクトストレージを使用して、高スループットの履歴読み取りを実行することもできます。また、ネットアップストレージコントローラを搭載した Spark に関するテクニカル解決策もご用意しています。詳細はこちらをご覧ください。Kafka がリアルタイムの分析パイプラインにどのように適しているかを次の図に示します。



次の図は、 NetApp StorageGRID が ConFluent Kafka のオブジェクトストレージ層にどのように適合するかを示しています。



解決策アーキテクチャの詳細

このセクションでは、流暢な検証に使用するハードウェアおよびソフトウェアについて説明します。この情報は、ネットアップストレージを使用した流暢なプラットフォームの導入に該当します。次の表に、テストに使用した解決策アーキテクチャと基本コンポーネントを示します。

解決策コンポーネント	詳細
Kafka バージョン 6.2 と競合します	• ご主人の 3 人 • 5 台のブローカーサーバ • 5 台のツールサーバ • Grafana × 1 • 1 つのコントロールセンター
Linux （ Ubuntu 18.04 ）	すべてのサーバ
階層化ストレージ向けの NetApp StorageGRID	• StorageGRID ソフトウェア • SG1000 × 1 （ロードバランサ） • SGF6024 × 4 • 24 本、 800 本の SSD × 4 • S3 プロトコル • 100GbE × 4 （ブローカーと StorageGRID インスタンス間のネットワーク接続）
Fujitsu Primergy RX2540 サーバ × 15	各モデルには、 CPU × 2 、物理コア × 16 、 Intel Xeon × 256GB 物理メモリ × 100GbE デュアルポートが搭載されています

テクノロジの概要

このセクションでは、この解決策で使用されるテクノロジーについて説明します。

NetApp StorageGRID

NetApp StorageGRID は、ハイパフォーマンスで対費用効果の高いオブジェクトストレージプラットフォームです。階層型ストレージを使用することで、ローカルストレージやブローカーの SAN ストレージに格納されている ConFluent Kafka にあるデータのほとんどが、リモートのオブジェクトストアにオフロードされます。この構成では、クラスタのリバランシング、拡張、縮小、障害が発生したブローカーの交換にかかる時間とコストを削減することで、運用が大幅に改善されます。オブジェクトストレージは、オブジェクトストア階層にあるデータの管理に重要な役割を果たします。そのため、適切なオブジェクトストレージを選択することが重要です。

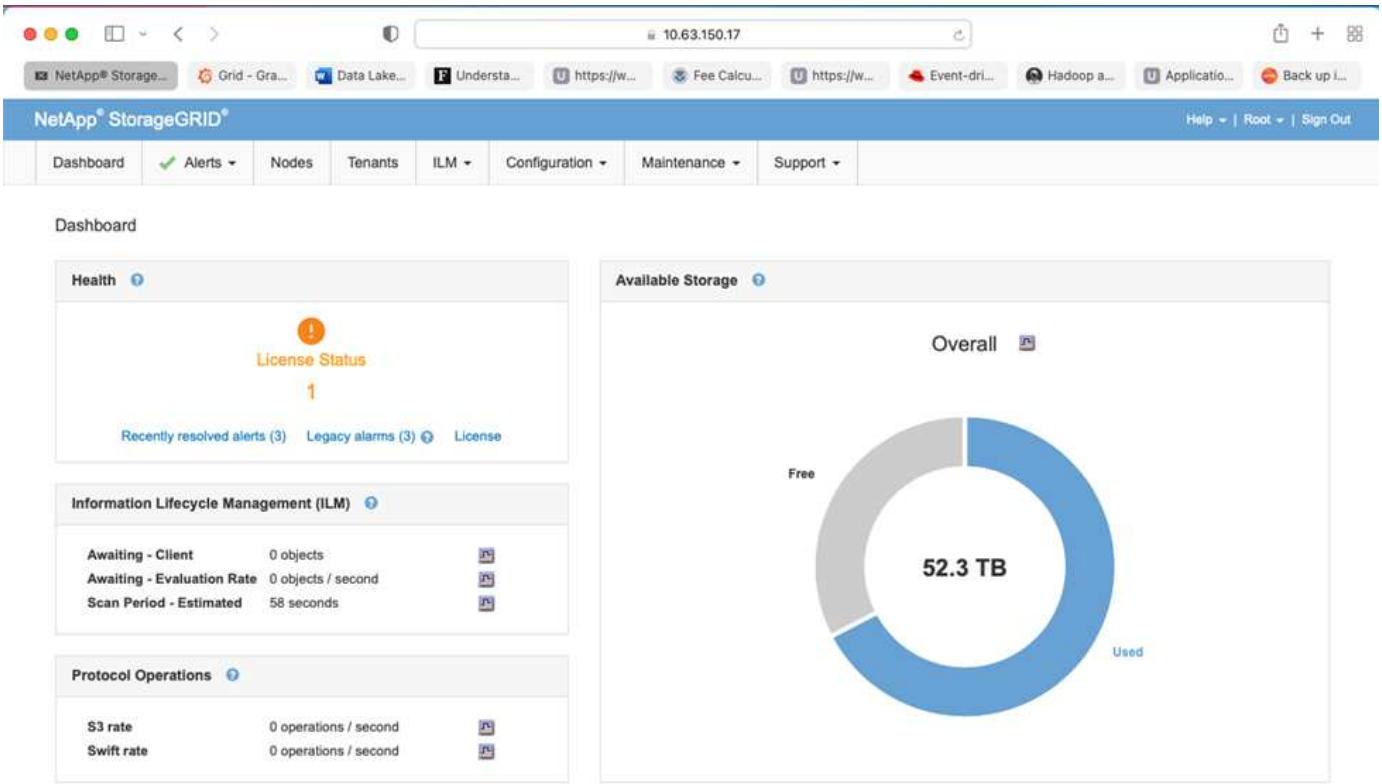
StorageGRID は、ノードベースの分散グリッドアーキテクチャを使用して、インテリジェントでポリシーベースのグローバルデータ管理を実現します。数ペタバイトの非構造化データと数十億のオブジェクトを、ユビキタスなグローバルオブジェクトネームスペースと高度なデータ管理機能を組み合わせることでシンプルに管理できます。単一コールのオブジェクトアクセスは、サイト間を拡張し、高可用性アーキテクチャを簡素化しながら、サイトやインフラストラクチャの停止に関係なく、オブジェクトへの継続的なアクセスを保証します。

マルチテナンシーを使用すると、複数の非構造化クラウドアプリケーションやエンタープライズデータアプリケーションを同じグリッド内で安全に処理できるため、 NetApp StorageGRID の ROI とユースケースが向上します。メタデータベースのオブジェクトライフサイクルポリシーを使用して複数のサービスレベルを作成し、複数の地域にわたるデータの保持、保護、パフォーマンス、ローカリティを最適化できます。ユーザは、データ管理ポリシーを調整し、トラフィック制限を監視および適用して、絶えず変化する IT 環境で要件が変

化した場合に備えて、システムを停止することなくデータランドスケープに再調整できます。

Grid Manager で管理を簡易化

StorageGRID Grid Manager はブラウザベースのグラフィカルインターフェイスで、世界中に分散された複数のサイトにまたがる StorageGRID システムの設定、管理、監視を、 1 つの画面で実行できます。



StorageGRID グリッドマネージャインターフェイスでは、次のタスクを実行できます。

- イメージ、ビデオ、レコードなどのオブジェクトを収めた、グローバルに分散されたペタバイト規模のリポジトリを管理します。
- グリッドノードとサービスを監視してオブジェクトの可用性を確保します。
- Information Lifecycle Management （ ILM ；情報ライフサイクル管理）ルールを使用してオブジェクトデータの配置を継続的に管理します。これらのルールによって、取り込まれたオブジェクトのデータの処理、損失から保護する方法、格納場所と保管期間が決まります。
- システム内のトランザクション、パフォーマンス、処理を監視します。

情報ライフサイクル管理ポリシー

StorageGRID には、オブジェクトのレプリカコピーを保持し、特定のパフォーマンスおよびデータ保護要件に応じて 2+1 や 4+2 などの EC （イレイジャーコーディング）スキームを使用してオブジェクトを格納するなどの、柔軟なデータ管理ポリシーが用意されています。ワークロードと要件が時間の経過とともに変化する場合、 ILM ポリシーも時間の経過とともに変化する必要があることがよくあります。ILM ポリシーの変更は中核的な機能であり、絶えず変化する環境に StorageGRID のお客様がすばやく簡単に適応できるようにします。

パフォーマンス

StorageGRID は、ストレージノードを追加することでパフォーマンスを拡張します。ストレージノードには、 VM 、ベアメタル、またはなどの専用アプライアンスを指定できます "SG5712 、 SG5760 、 SG6060 、 SGF6024"。今回のテストでは、 SGF6024 アプライアンスを使用した最小サイズの 3 ノードグリッドで、 Apache Kafka の主要なパフォーマンス要件を超えました。Kafka クラスタを追加のブローカーとともに拡張すれば、ストレージノードを追加してパフォーマンスと容量を高めることができます。

ロードバランサとエンドポイントの設定

StorageGRID の管理ノードは、 StorageGRID システムを表示、設定、管理するための Grid Manager UI （ユーザインターフェイス）エンドポイントと REST API エンドポイント、およびシステムアクティビティを追跡するための監査ログを提供します。そこで、 Conluent Kafka の階層化ストレージに可用性の高い S3 エンドポイントを提供するために、 StorageGRID ロードバランサを実装しました。このロードバランサは、管理ノードとゲートウェイノードでサービスとして実行されます。また、ロードバランサはローカルトラフィックを管理し、ディザスタリカバリに役立つ GSLB （グローバルサーバロードバランシング）と通信します。

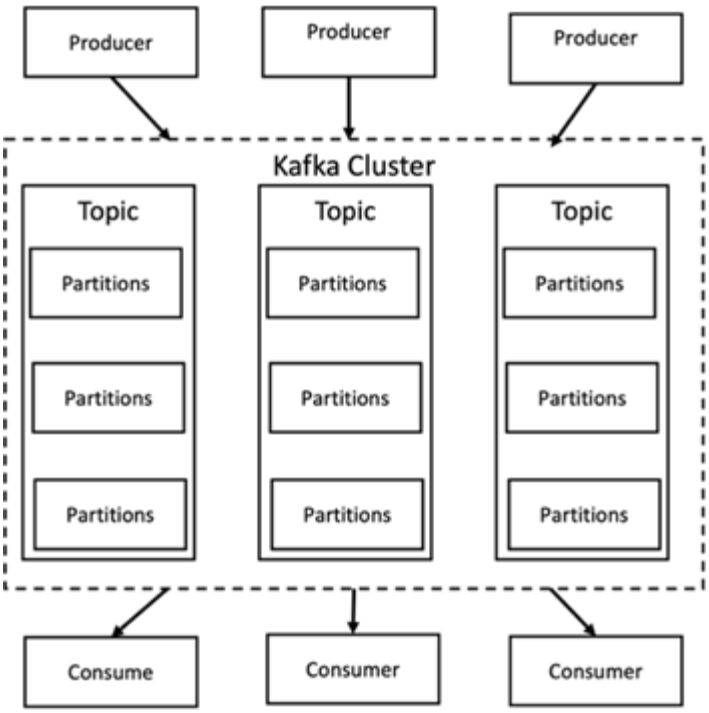
エンドポイントの設定をさらに強化するために、 StorageGRID は管理ノードに組み込まれたトラフィック分類ポリシーを提供し、ワークロードトラフィックを監視し、さまざまな Quality of Service （ QoS ；サービス品質）制限をワークロードに適用できます。トラフィック分類ポリシーは、ゲートウェイノードおよび管理ノードの StorageGRID ロードバランササービス上のエンドポイントに適用されます。これらのポリシーは、トラフィックシェーピングおよびモニタリングに役立ちます。

StorageGRID でのトラフィック分類

StorageGRID には QoS 機能が組み込まれています。トラフィック分類ポリシーを使用すると、クライアントアプリケーションからのさまざまなタイプの S3 トラフィックを監視できます。次に、ポリシーを作成して適用し、イン / アウト帯域幅、読み取り / 書き込み同時要求の数、または読み取り / 書き込み要求の速度に基づいて、このトラフィックに制限を設けることができます。

Apache Kafka です

Apache Kafka は、 Java と Scala で書かれたストリーム処理を使用したソフトウェアバスのフレームワーク実装です。リアルタイムデータフィードを処理するための、スループットが高く、低レイテンシの統合プラットフォームを提供することを目的としています。Kafka は外部システムに接続して Kafka Connect からデータをエクスポートし、インポートすることができます。また、 Java のストリーム処理ライブラリである Kafka ストリームが提供されます。Kafka では、効率性を重視して最適化されたバイナリの TCP ベースのプロトコルを使用しています。また、ネットワーク往復のオーバーヘッドを軽減するために、メッセージを自然にまとめてグループ化する「メッセージセット」抽象化に依存しています。これにより、より大規模なシーケンシャルディスク処理や、大容量のネットワークパケット、連続するメモリブロックが実現し、 Kafka では、バースト性の高いランダムメッセージ書き込みをリニア書き込みに変換することができます。次の図は、 Apache Kafka の基本的なデータフローを示しています。



Kafka には、 Producer と呼ばれる任意の数のプロセスから生成されるキーと値のメッセージが格納されます。データは、異なるトピック内の異なるパーティションにパーティショニングできます。パーティション内では、メッセージはオフセット（パーティション内のメッセージの位置）によって厳密に順序付けされ、インデックスが作成され、タイムスタンプとともに格納されます。コンシューマと呼ばれる他のプロセスは、パーティションからメッセージを読み取ることができます。Kafka はストリーム処理用の API を提供しており、Kafka からデータを利用する Java アプリケーションを作成して Kafka に結果を書き込むことができます。Apache Kafka は、 Apache Apex 、 Apache Flink 、 Apache Spark 、 Apache Storm 、 Apache NiFi などの外部ストリーム処理システムとも連携します。

Kafka は 1 つ以上のサーバで構成されたクラスタ（ブローカー）上で実行され、すべてのトピックのパーティションがクラスタノード全体に分散されます。さらに、パーティションは複数のブローカーにレプリケートされます。Kafka はこのアーキテクチャにより、フォールトトレラントな方法で大量のメッセージストリームを配信でき、 Java Message Service （ JMS ）や Advanced Message Queuing Protocol （ AMQP ）などの従来のメッセージングシステムの一部を置き換えることができます。0.11.0.0 リリース以降、 Kafka はトランザクション書き込みを提供しており、これは Streams API を使用して一度のストリーム処理を提供します。

Kafka では、 Regular とコンパクションの 2 種類のトピックをサポートしています。通常のトピックでは、保持期限またはスペースバインドを設定できます。指定した保持期限よりも古いレコードがある場合や、パーティションのスペースバインドを超過している場合、 Kafka では古いデータを削除してストレージスペースを解放することができます。デフォルトでは、トピックの保持期間は 7 日間に設定されていますが、データを無期限に保存することもできます。コンパクションの対象となるトピックについては、レコードの有効期限は時刻やスペースの上限に基づいて切れません。Kafka では、以降のメッセージを同じキーを持つ古いメッセージの更新として扱い、キーごとに最新のメッセージを削除しないことを保証しています。ユーザは、特定のキーのヌル値を持つ、いわゆる tombstone メッセージを書き込むことによって、メッセージを完全に削除できます。

Kafka には 5 つの主要な API があります。

- * Producer API. * は、アプリケーションがレコードのストリームをパブリッシュすることを許可します。
- *Consumer API. * は、アプリケーションがトピックを購読し、レコードのストリームを処理することを許可します。

- * Connector API. * は、トピックを既存のアプリケーションにリンクできる再利用可能なプロデューサおよびコンシューマ API を実行します。
- *Streams API. * この API は入力ストリームを出力に変換し、結果を生成します。
- * 管理者 API 。 Kafka のトピック、ブローカー、その他の Kafka のオブジェクトを管理するのに使用されます。

Kafka メッセージングプロトコルをベースに構築されたコンシューマ向け API とプロデューサー用 API は、Java で Kafka コンシューマクライアントとプロデューサークライアント向けのリファレンス実装を提供します。基本的なメッセージングプロトコルは、開発者が任意のプログラミング言語で独自のコンシューマクライアントまたはプロデューサクライアントを作成するために使用できるバイナリプロトコルです。これにより、Java Virtual Machine （ JVM ； Java 仮想マシン）エコシステムの Kafka のロックが解除されます。使用可能な Java 以外のクライアントの一覧は、 Apache Kafka wiki で管理されています。

Apache Kafka のユースケース

Apache Kafka は、メッセージング、 Web サイトのアクティビティ追跡、指標、ログ集約、ストリーム処理に最もよく使用されています。 イベントのソーシングとロギングのコミット

- Kafka はスループットの向上、組み込みのパーティショニング、レプリケーション、およびフォールトトレランスを実現しており、大規模なメッセージ処理アプリケーションに適した解決策となっています。
- Kafka では、リアルタイムのパブリッシュサブスクライブフィードのセットとして、追跡パイプラインでユーザのアクティビティ（ページビュー、検索）を再構築できます。
- Kafka は、多くの場合、運用監視データに使用されます。これには、分散アプリケーションからの統計情報を集約して、運用データの一元化フィードを作成する作業が含まれます。
- 多くの人が、ログアグリゲーション解決策の代わりに Kafka を使用しています。ログアグリゲーションは、一般にサーバから物理ログファイルを収集して処理のために一元的な場所（ファイルサーバや HDFS など）に配置します。Kafka は、ファイルの詳細を抽象化し、ログやイベントデータをメッセージのストリームとしてより明確に抽象化します。これにより、低レイテンシの処理が可能になり、複数のデータソースと分散データ消費のサポートが容易になります。
- Kafka のユーザの多くは、複数のステージで構成されるパイプラインでデータを処理しています。 Kafka のトピックから生の入力データが消費され、さらに消費やフォローアップ処理のために、集約、エンリッチ化、または新しいトピックへと変換されます。たとえば、ニュース記事を推薦するための処理パイプラインでは、 RSS フィードから記事のコンテンツをクロールし、それを「記事」トピックに公開することができます。さらに処理を行うと、このコンテンツをノーマライズまたは重複排除し、クレンジングされた記事コンテンツを新しいトピックにパブリッシュすることができます。また、最終的な処理段階では、このコンテンツをユーザーに推奨しようとする場合があります。このような処理パイプラインでは、個々のトピックに基づいてリアルタイムのデータフローのグラフが作成されます。
- イベントソースとは、状態の変化を時系列のレコードとしてログに記録するアプリケーション設計のスタイルです。Kafka は、非常に大容量の格納ログデータをサポートしているため、この形式のアプリケーションのバックエンドとして最適です。
- Kafka は分散システム用の一種の外部コミットログとして機能します。ログはノード間でデータをレプリケートするのに役立ち、障害が発生したノードがデータをリストアする際の再同期メカニズムとして機能します。Kafka のログコンパクション機能は、このユースケースに対応しています。

矛盾する

Conflicent Platform は、 Kafka を完成させるエンタープライズ対応プラットフォームです。高度な機能を備えており、アプリケーションの開発と接続を高速化し、ストリーム処理による変換を可能にし、大規模なエンタープライズ運用を簡易化し、厳しいアーキテクチャ要件に対応します。ConFluent では、 Apache Kafka を作

成した元のクリエイターが開発したサービスを利用して、 Kafka のメリットをエンタープライズクラスの機能で拡張しながら、 Kafka の管理や監視の負担を軽減することができます。現在、 Fortune 100 企業の 80% 以上がデータストリーミングテクノロジを採用しており、そのほとんどが Conluent 社を使用しています。

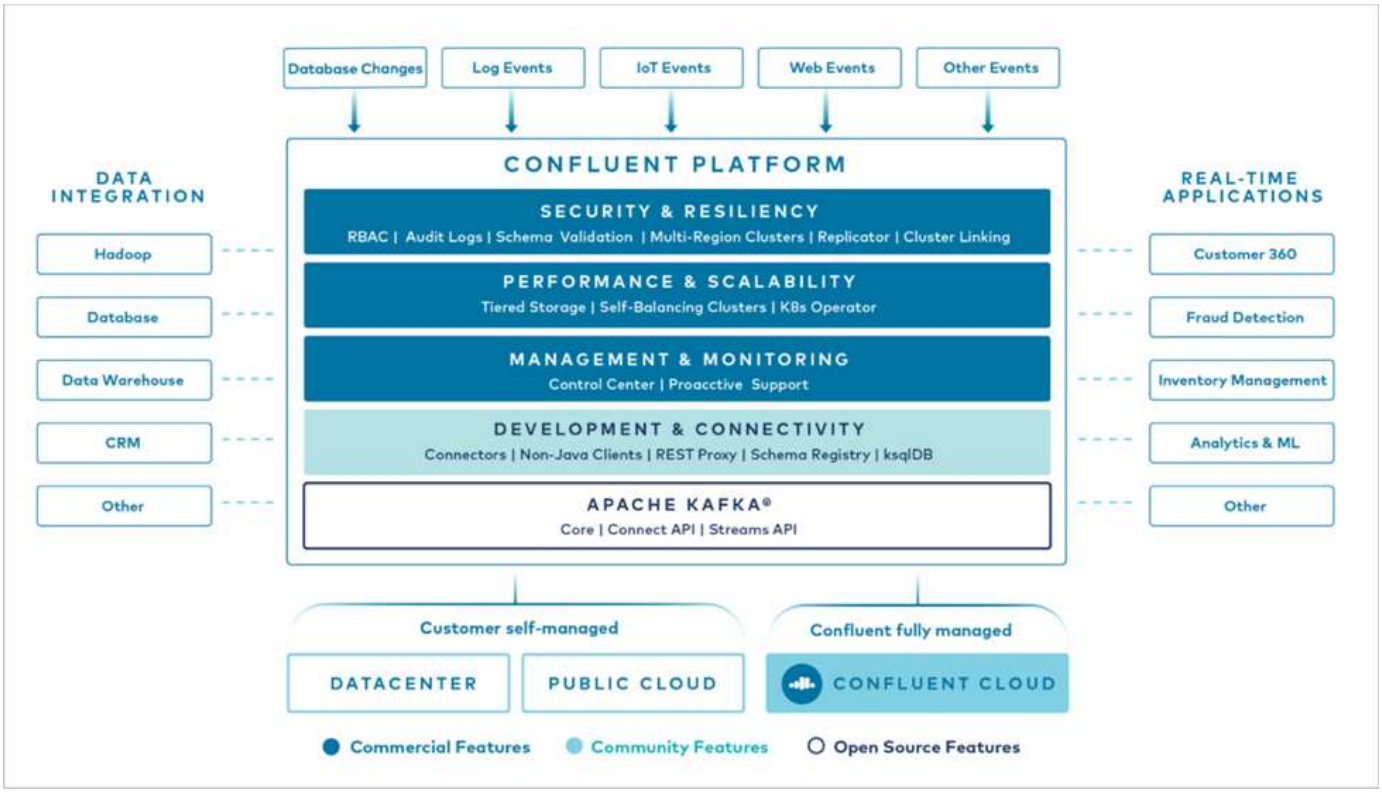
流暢な理由

履歴データとリアルタイムデータを一元化された単一の情報源に統合することで、 Conluent は、まったく新しいカテゴリの最新のイベント駆動型アプリケーションを簡単に構築し、ユニバーサルデータパイプラインを取得し、拡張性、パフォーマンス、信頼性を備えた強力な新しいユースケースを開放します。

流暢なものは何のために使用されるか。

Conflicent Platform を使用すると、データが異なるシステム間でどのように転送または統合されるかなど、基本的なメカニズムを気にすることなく、データからビジネス価値を引き出す方法に集中できます。具体的には、 Con裕福 なプラットフォームによって、 Kafka へのデータソースの接続やストリーミングアプリケーションの構築、 Kafka インフラの保護、監視、管理が簡易化されます。現在、 Conluent Platform は、金融サービス、オムニチャネル小売、自律走行車など、さまざまな業界のさまざまなユースケースに使用されています。 マイクロサービス、 IoT 。

以下の図は、 ConFluent Kafka Platform のコンポーネントを示しています。



流暢なイベントストリーミング技術の概要

流暢なプラットフォームの中核はです "Apache Kafka です"最も人気の高いオープンソースの分散ストリーミングプラットフォームです。Kafka の主な機能は次のとおりです。

- レコードのストリームをパブリッシュしてサブスクライブします。
- レコードのストリームをフォールトトレラントな方法で保存します。

- レコードのストリームを処理します。

Conluent Platform には Schema Registry 、 REST Proxy 、合計 100 以上の Kafka コネクタ、および ksqlDB も含まれています。

流暢なプラットフォームのエンタープライズ機能の概要

- * Conluent Control Center * Kafka を管理および監視するための GUI ベースのシステム。Kafka Connect の管理や、他のシステムとの接続の作成、編集、管理を簡単に行うことができます。
- * Kubernetes には流暢な言葉があります。 * Kubernetes の流暢な言葉は Kubernetes のオペレータです。Kubernetes の運用担当者は、特定のプラットフォームアプリケーションに固有の機能と要件を提供することで、 Kubernetes のオーケストレーション機能を拡張します。Con裕福 なプラットフォームの場合は、 Kubernetes での Kafka の導入プロセスを大幅に簡易化し、一般的なインフラのライフサイクルタスクを自動化します。
- * Kafka コネクタは、 Kafka Connect API を使用して、 Kafka をデータベース、キーバリューストア、検索インデックス、ファイルシステムなどの他のシステムに接続します。Confluent Hub には、一般的なデータソースおよびシンク用のダウンロード可能なコネクタがあります。これには、 Conluent Platform でこれらのコネクタの完全なテストとサポートされたバージョンが含まれます。詳細については、を参照してください "こちらをご覧ください"。
- * セルフバランシングクラスタ。 * 自動ロードバランシング、障害検出、自己修復機能を提供します。必要に応じてブローカーの追加や運用停止をサポートし、手動での調整は不要です。
- * クラスタを直接接続し、リンクブリッジを介して 1 つのクラスタから別のクラスタにトピックをミラーリングします。クラスタリンクにより、マルチデータセンター、マルチクラスタ、ハイブリッドクラウドの導入を簡易化できます。
- * 流暢な自動データバランサ。 * ブローカーの数、パーティションのサイズ、パーティションの数、およびクラスタ内のリーダーの数について、クラスタを監視します。これにより、データを移動してクラスタ全体で均等なワークロードを作成しながら、トラフィックのリバランシングを調整して、リバランシング中の本番ワークロードへの影響を最小限に抑えることができます。
- * 流暢なリプリケータ * により、複数のデータセンターで複数の Kafka クラスターを容易に保守できます。
- * 階層化ストレージ。 * 任意のクラウドプロバイダを使用して大量の Kafka データを保存するオプションを提供し、運用上の負担とコストを削減します。階層型ストレージでは、コスト効率に優れたオブジェクトストレージにデータを格納し、ブローカーを拡張するために、必要なコンピューティングリソースが増えた場合のみデータを利用できます。
- * Conluent JMS Client. * Conluent Platform には Kafka 用の JMS 対応クライアントが含まれています。Kafka クライアントは、 Kafka ブローカーをバックエンドとして使用して、 JMS 1.1 標準 API を実装しています。これは 'JMS を使用するレガシーアプリケーションがあり ' 既存の JMS メッセージブローカを Kafka に置き換える場合に便利です
- * Coneluent MQTT プロキシ * を使用すると、 MQTT デバイスやゲートウェイから Kafka に直接データを公開できます。 MQTT ブローカーは必要ありません。
- * 流暢なセキュリティプラグイン。 * 流暢なセキュリティプラグインは、各種の流暢なプラットフォームツールや製品にセキュリティ機能を追加するために使用されます。現在、 Conluent REST プロキシ用のプラグインが用意されており、受信要求の認証に役立ち、認証されたプリンシパルを要求に Kafka に伝播できます。これにより、 Con裕福 な REST プロキシクライアントでは、 Kafka ブローカーのマルチテナントセキュリティ機能を利用できます。

矛盾する検証

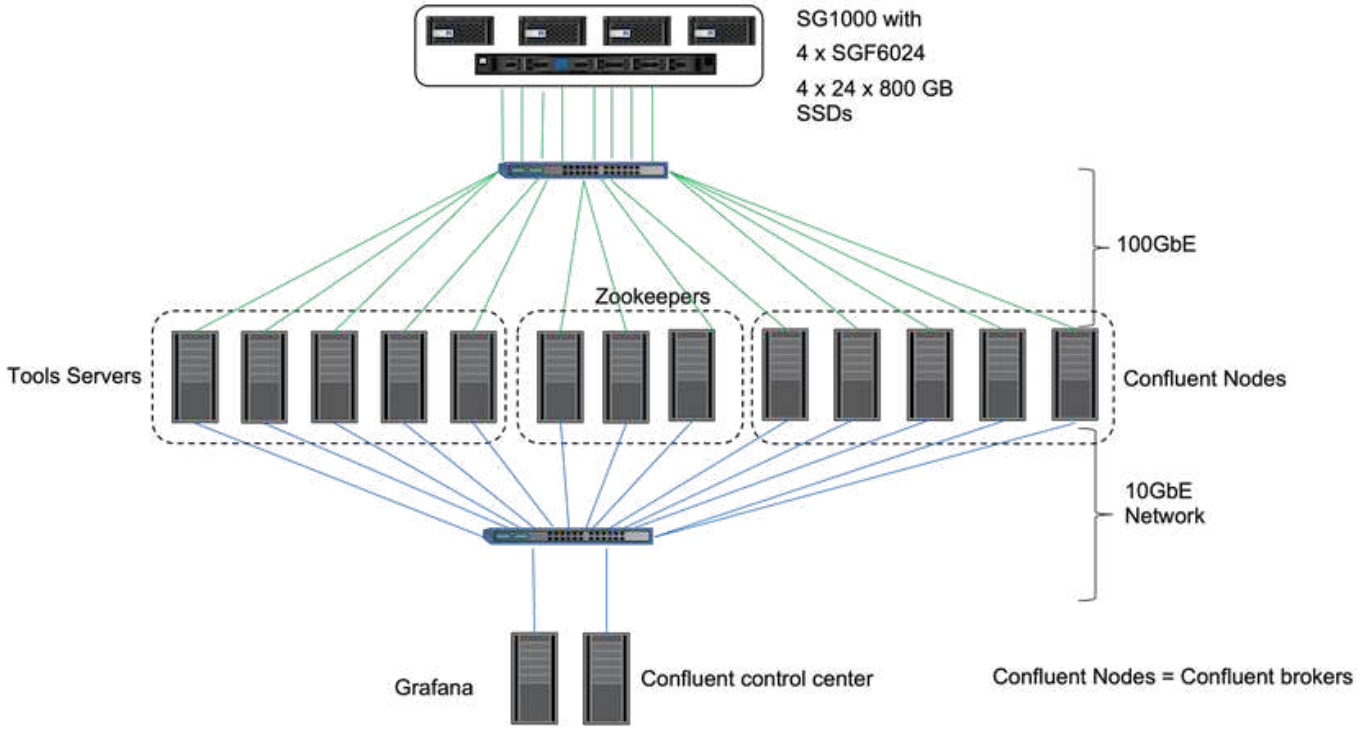
NetApp StorageGRID で、 Conluent Platform 6.2 の階層型ストレージを使用して検証を実施しました。ネットアップと流暢なチームがこの検証に協力し、検証に必要なテストケースを実施しました。

競合するプラットフォームの設定

検証には次のセットアップを使用しました。

検証には、 3 台の zookeepers 、 5 台のブローカー、 5 台のテストスクリプトを実行するサーバ、 256GB の RAM を搭載した tools サーバ、 16 個の CPU を使用しました。ネットアップストレージの場合は、 4 つの SGF6024 を搭載した SG1000 ロードバランサで StorageGRID を使用しました。ストレージとブローカーは、 100GbE 接続経由で接続されています。

次の図に、流暢な検証に使用される設定のネットワークトポロジを示します。



ツールサーバは、要求をノードに送信するアプリケーションクライアントとして機能します。

競合する階層型ストレージ構成

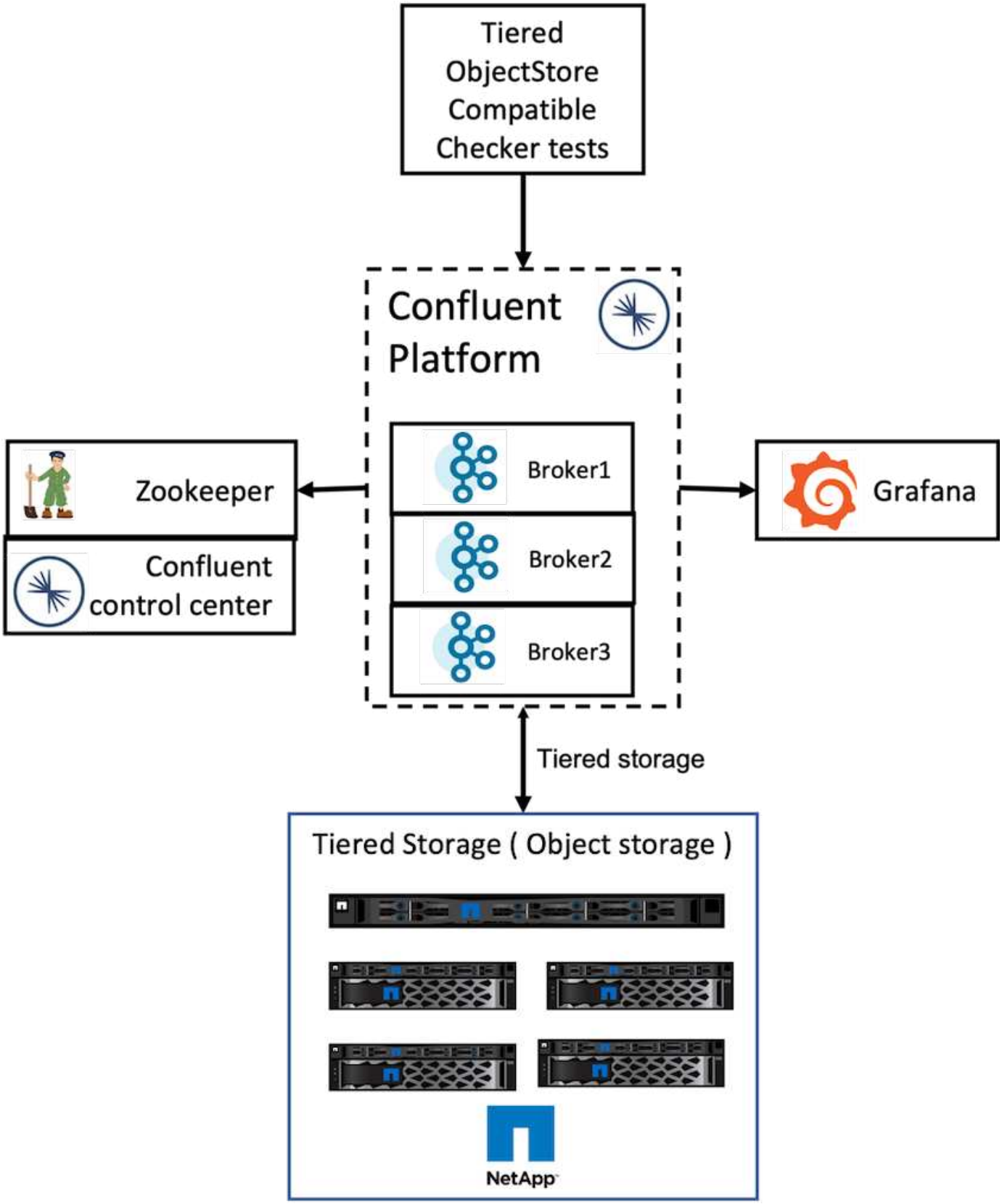
階層化ストレージの構成には、 Kafka に次のパラメータが必要です。

```
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
```

検証 StorageGRID には HTTP プロトコルを使用しましたが、 HTTPS も使用できます。アクセスキーとシークレットキーは、「 confliclus.tir.s3.cred.file.path 」パラメータで指定したファイル名に格納されます。

ネットアップオブジェクトストレージ **- StorageGRID**

単一サイト構成を StorageGRID で検証用に設定しました。



検証テスト

以下の 5 つの検証ケースを完了しました。これらのテストは、 Trogdor フレームワークで実行されます。最初の 2 つは機能テストで、残りの 3 つはパフォーマンステストです。

オブジェクトストアの正確性テスト

このテストでは、階層化ストレージのニーズに応じて、オブジェクトストア API のすべての基本的な処理（GET / PUT / DELETE など）が適切に機能するかどうかを確認します。これは、すべてのオブジェクトストアサービスが次のテストよりも先に実施されることを想定した基本的なテストです。合格または不合格の自己主張的なテストです。

階層化機能の正確性テスト

このテストでは ' エンド・ツー・エンドの階層型ストレージ機能が ' 合格または不合格のアサート型テストで適切に機能するかどうかを判断しますテストでは、デフォルトで階層化が有効になっており、ホットセットサイズが大幅に縮小されたテストトピックが作成されます。新しく作成されたテストトピックへのイベントストリームが生成され、ブローカーがセグメントをオブジェクトストアにアーカイブするのを待機し、イベントストリームを消費して、消費されたストリームが生成されたストリームと一致することを検証します。イベントストリームに生成されるメッセージの数は設定可能で、テストのニーズに応じてユーザが十分な大きさのワークロードを生成できます。ホットセットのサイズを小さくすることで、消費者がアクティブなセグメントの外部でフェッチしたファイルはオブジェクトストアからのみ提供されます。これにより、オブジェクトストアの読み取りの正確性をテストできます。このテストは、オブジェクトストアフォールト挿入の有無にかかわらず実施しました。StorageGRID のいずれかのノードでサービスマネージャサービスを停止し、エンドツーエンド機能がオブジェクトストレージで機能することを検証することで、ノード障害をシミュレートしました。

ティアフェッチベンチマーク

このテストでは、階層型オブジェクトストレージの読み取りパフォーマンスを検証し、ベンチマークによって生成されたセグメントからの負荷が大きい範囲での読み取り要求のフェッチをチェックしました。このベンチマークでは、 Conluent 社は階層フェッチ要求に対応するカスタムクライアントを開発しました。

ワークロードベンチマークを消費

このテストでは、セグメントをアーカイブすることにより、オブジェクトストアへの書き込みワークロードを間接的に生成しました。コンシューマグループがセグメントを取得すると、読み取りワークロード（セグメント読み取り）がオブジェクトストレージから生成されました。このワークロードはテストスクリプトで生成されました。このテストでは、並列スレッドでのオブジェクトストレージの読み取りと書き込みのパフォーマンスをチェックしました。階層化機能の正確性テストと同様に、オブジェクトストアフォールト挿入を使用したテストと使用しなかったテストを実施しました。

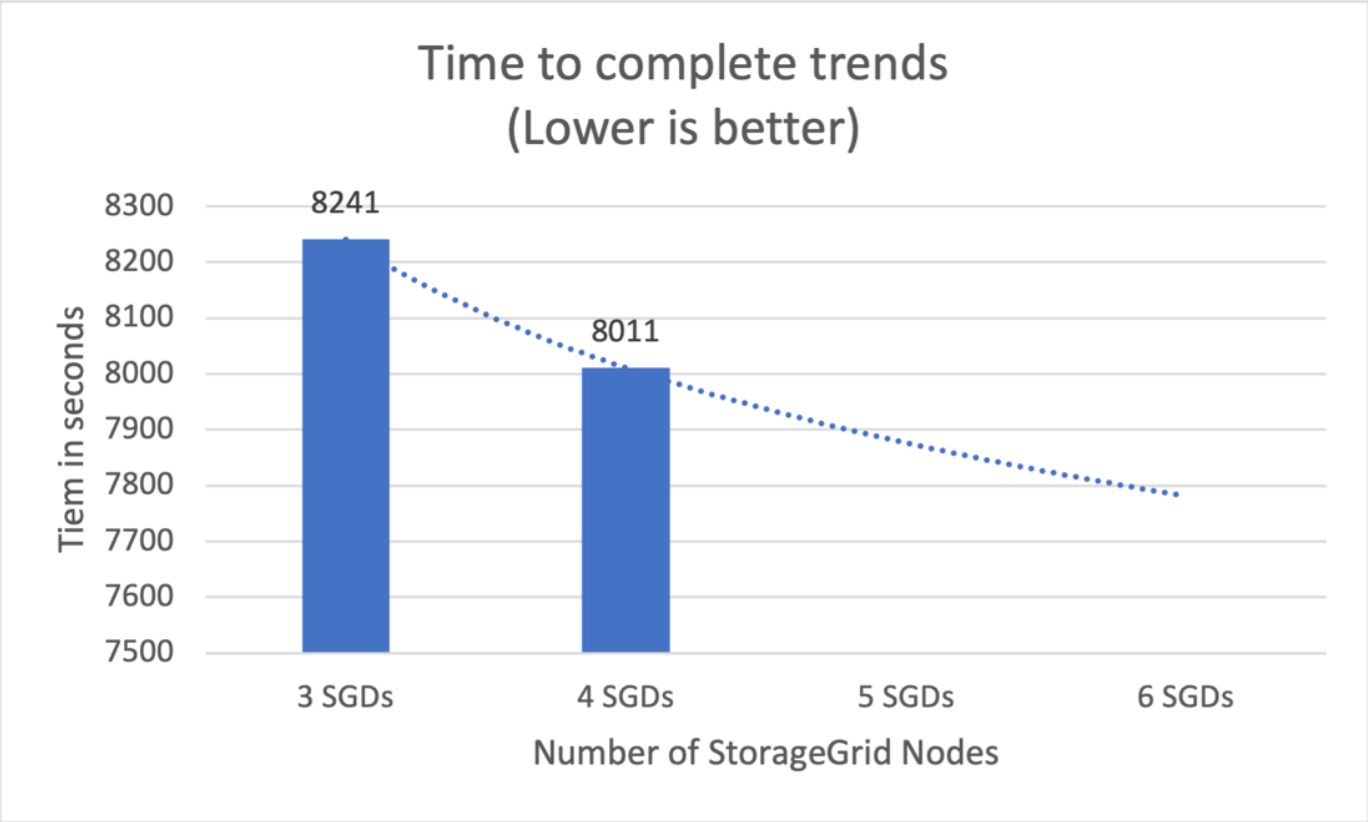
保存ワークロードベンチマーク

このテストでは、トピックの保持ワークロードが多い場合のオブジェクトストアの削除パフォーマンスを確認しました。保持ワークロードは、テストトピックと並行して多数のメッセージを生成するテストスクリプトを使用して生成されました。テストトピックでは、サイズベースおよび時間ベースの強力な保持設定を使用してイベントストリームをオブジェクトストアから継続的にパージするように設定しました。その後、セグメントがアーカイブされました。その結果、ブローカーによるオブジェクトストレージの削除や、オブジェクトストアの削除処理のパフォーマンス収集が行われ、大量の削除が発生していました。

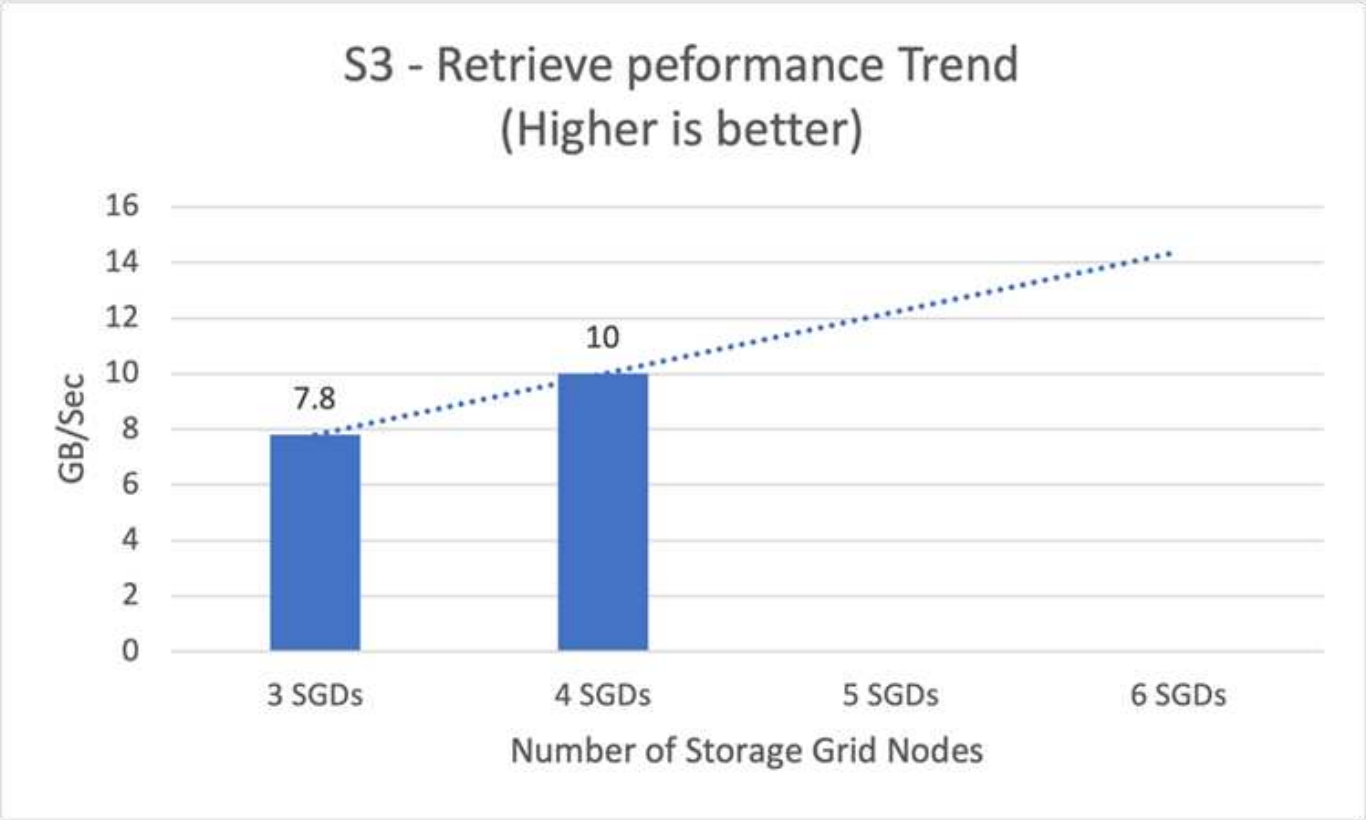
拡張性を備えたパフォーマンステスト

このティアストレージのテストでは、プロデューサーワークロードとコンシューマーワークロード向けに、 NetApp StorageGRID セットアップを使用して、 3 ノードから 4 ノードのノードを使用しました。テストによると、完了までの時間とパフォーマンス結果は StorageGRID ノードの数に直接比例しました。StorageGRID セットアップには、少なくとも 3 つのノードが必要でした。

- ストレージノードの数が増えると、農産物と消費者の処理を完了するまでの時間が直線的に短くなりました。



- s3 読み出し処理のパフォーマンスは、 StorageGRID ノードの数に基づいてリニアに向上します。StorageGRID は、最大 200 個の StorgeGRID ノードをサポートします。



矛盾点 3 コネクタ

Amazon S3 Sink Connector は、 Apache Kafka のトピックから Avro 、 JSON 、またはバイト形式の S3 オブジェクトにデータをエクスポートします。Amazon S3 シンクコネクタは、 Kafka から定期的にデータをポーリングし、 S3 にアップロードします。ユーザは、各 Kafka パーティションのデータをチャンクに分割するために使用します。データのチャンクはそれぞれ S3 オブジェクトとして表されます。キー名は、トピック、Kafka パーティション、およびこのデータチャンクの開始オフセットをエンコードします。

このセットアップでは、 Kafka s3 sink Connector を使用して、 Kafka のオブジェクトストレージのトピックの読み取りと書き込みを直接実行する方法を紹介します。このテストでは、スタンドアロンの流暢なクラスタを使用しましたが、このセットアップは分散クラスタに適用できます。

1. Conluent Kafka の Web サイトからダウンロードできます。
2. パッケージをサーバー上のフォルダに展開します。
3. 2 つの変数をエクスポートします。

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

4. スタンドアロンの ConFluent Kafka セットアップの場合、クラスタは「 /tmp 」に一時的なルートフォルダを作成します。Zookeeper 、 Kafka 、スキーマレジストリ、 connect 、 ksql-server 、 とコントロールセンターのフォルダを作成し、それぞれの構成ファイルを「 $confliclus_home 」からコピーします。次の

例を参照してください。

```
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. Zookeeper を構成します。デフォルトのパラメータを使用する場合は、何も変更する必要はありません。

```
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:~#
```

上記の設定では、サーバを更新しました。xxx ’プロパティ。デフォルトでは、 Kafka リーダーの選択に 3 名の Zookeepers が必要です。

6. myid ファイルを tmp/conflicluent .406980/zookeeper /data に一意の ID で作成しました。

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

myid ファイルの最後の数の IP アドレスを使用しました。Kafka 、 connect 、 control-ccenter、 Kafka 、 Kafkakarest 、 ksql-server 、およびスキーマレジストリ設定。

7. Kafka サービスを開始します。

```
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#
```

構成ごとにログフォルダがあり、問題のトラブルシューティングに役立ちます。場合によっては、サービスの開始に時間がかかることがあります。すべてのサービスが稼働中であることを確認します。

8. 「 confliclue-hub 」を使用して Kafka connect をインストールします。

```
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#
```

また、「 conflicluent -hub install conflicentinc / Kafka-connect-s3 ： 10.0.3` を使用して、特定のバージョンをインストールすることもできます。

9. デフォルトでは、「 confluentinc - Kafka-connect-s3 」は「 /data/luent confin/conflicluent - 6.2.0/conflicluent -huber-components/conflicentinc - Kafka-connect-s3 」にインストールされています。
10. 新しい「 confluentinc - Kafka -connect-s3` でプラグインパスを更新します。

```
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. 流暢なサービスを停止し、再起動します。

```
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. アクセス ID とシークレットキーを「 /root/.AWS/credentials 」ファイルに設定します。

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

13. バケットに到達できることを確認します。

```
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 およびバケット設定用の s3-sink プロパティファイルを設定します。

```
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 バケットに数件のレコードをインポートします。

```
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 シンクコネクタを取り付けます。

```
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-sink のステータスを確認します。

```
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. ログをチェックして、 s3-sink のトピックを受け入れる準備ができていることを確認します。

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

19. Kafka のトピックを確認してください。

```
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 バケット内のオブジェクトを確認します。

```
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. 内容を確認するには、次のコマンドを実行して、 S3 からローカルファイルシステムに各ファイルをコピーします。

```
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. レコードを印刷するには、 avro-tools-1.11.0.1.jar を使用します（『』で入手できます） "Apache アーカイブ"）。

```
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:~#
```

競合する自己バランシングクラスタ

Kafka クラスタを以前に管理していたことがある場合、パーティションを別のブローカーに手動で再割り当てすることで、ワークロードがクラスタ全体に分散されるようにするという問題に慣れている方がよいでしょう。Kafka を大規模に導入している組織では、大容量のデータを変更するのが難しく、面倒でリスクが伴う可能性があります。特に、ミッションクリティカルなアプリケーションをクラスタの上に構築する場合、リスクが高くなります。しかし、 Kafka を使用するケースが最小であっても、処理に時間がかかり、人為的ミスが発生しやすくなります。

このラボでは、流暢な自己分散クラスタ機能をテストし、クラスタトポロジの変更や不均衡な負荷に基づいてリバランシングを自動化しました。一括リバランシングテストは、ノード障害や拡張ノードでブローカー間のデータのリバランシングが必要になった場合に、新しいブローカーを追加する時間を測定するのに役立ちます。従来の Kafka 構成では、クラスタの拡張に合わせてデータの再分散量が増える一方で、階層型ストレージでは少量のデータしかリバランシングされません。この検証に基づき、階層型ストレージのリバランシングには数秒から数分かかりますが、従来の Kafka アーキテクチャでは、クラスタの拡張に伴ってリニアに拡張されます。

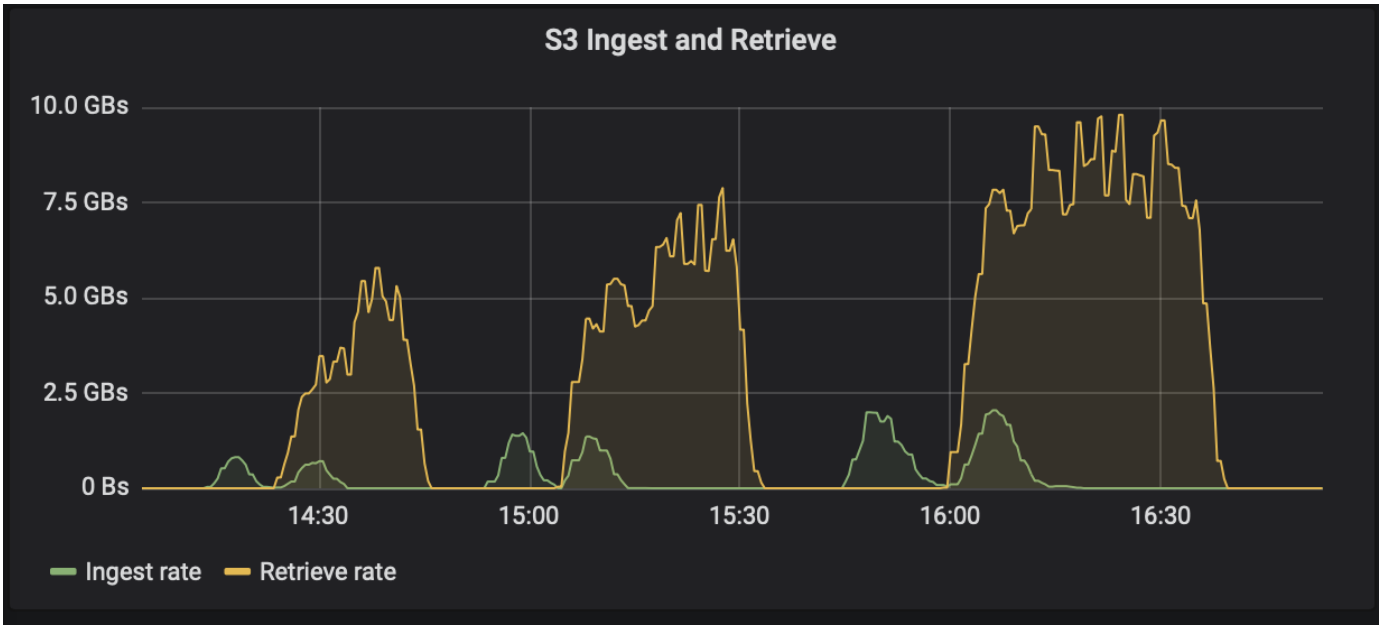
セルフバランシングクラスタでは、パーティションの再調整が完全に自動化され、 Kafka のスループットが最適化され、ブローカーの拡張が高速化され、大規模なクラスタを実行する際の運用上の負担が軽減されます。安定した状態では、自己バランシングクラスタがブローカー間のデータのスキューを監視し、パーティションを継続的に再割り当てしてクラスタのパフォーマンスを最適化します。プラットフォームをスケールアップまたはスケールダウンすると、新しいブローカーが存在することを自己分散クラスタが自動的に認識したり、古いブローカーが削除されたことを確認したり、後続のパーティションの再割り当てをトリガーしたりします。これにより、ブローカーの追加や運用停止が容易になり、 Kafka クラスタの柔軟性が根本的に向上します。これらの利点は、手動操作、複雑な計算、またはパーティションの再割り当てが一般的に発生する人的ミスのリスクを必要としないことです。その結果、データのリバランシングが完了するまでの時間が大幅に短縮され、クラスタを常時監視するのではなく、価値の高いイベントストリーミングプロジェクトに集中できます。

ベストプラクティスのガイドライン

このセクションでは、この認定資格から得られた教訓について説明します。

- 検証に基づくと、 S3 オブジェクトストレージにはデータを格納するのがベストプラクティスです。
- 高スループットの SAN （特に FC ）を使用して、ブローカーのホットデータやローカルディスクを保持できます。これは、階層型ストレージの構成が流暢であるためです。 ブローカーデータディレクトリに格納されているデータのサイズは、オブジェクトストレージにデータが移動される際のセグメントサイズと保持時間に基づいています。
- オブジェクトストレージは、セグメントの方がパフォーマンスに優れています。バイト数が多い場合は 512 MB をテストしました。
- Kafka では、トピックに対して生成される各レコードのキーまたは値の長さ（バイト単位）は、「 length.key.value 」パラメータによって制御されます。StorageGRID では、 S3 オブジェクトの取り込みと読み出しのパフォーマンスがより高い値に引き上げられました。たとえば、 512 バイトが 5.8GBps の読み出しを提供し、 1024 バイトが 7.5GBps の s3 読み出しを提供し、 2048 バイトが 10Gbps 近く提供します。

次の図に、「 length.key.value 」に基づいた S3 オブジェクトの取り込みと読み出しを示します。



- * Kafka のチューニング。 * 階層型ストレージのパフォーマンスを向上させるには、TierFetcherNumThreads と TierArchiverNumThreads を増やします。一般的なガイドラインとして、TierFetcherNumThreads を増やして物理 CPU コア数に合わせ、 TierArchiverNumThreads を CPU コア数の半分に増やします。たとえば、サーバープロパティで、 8 つの物理コアを持つマシンがある場合、conflicent.tier.fetcher.threads=8 と、 conflicent.tier.Archiver .num.threads=4 を設定します。
- * トピックが削除されるまでの時間です。 * トピックが削除されると、オブジェクトストレージ内のログセグメントファイルの削除はすぐには開始されません。デフォルト値の 3 時間を指定した時間間隔が設定されているため、この時間が経過するとファイルが削除されます。この間隔の値を変更するには、設定 confluent.tier.topic.delete.check.interval.ms を変更します。トピックまたはクラスタを削除する場合は、それぞれのバケット内のオブジェクトを手動で削除することもできます。
- * 階層型ストレージの内部トピックに関する ACL 。 * オンプレミス環境に推奨されるベストプラクティスは、階層型ストレージに使用される内部トピックについて、 ACL 承認者を有効にすることです。このデータへのアクセスをブローカーユーザのみに制限するには、 ACL ルールを設定してください。これにより、内部トピックが保護され、階層化されたストレージデータおよびメタデータへの不正アクセスを防止できます。

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

ユーザ「 <Kafka> 」を、導入環境の実際のブローカプリンシパルに置き換えます。

たとえば ' コマンド confliclus-tier-state は ' 階層型ストレージの内部トピックに ACL を設定します現在、階層化ストレージに関連する内部トピックは 1 つだけです。この例では、内部トピックのすべての処理にプリンシパル Kafka 権限を提供する ACL を作成しています。

サイジング

Kafka サイジングは、シンプル、きめ細かい、リバース、およびパーティションの 4 つ

の構成モードで実行できます。

シンプル

シンプルモードは、 Apache Kafka を初めて使用するユーザや初期状態のユースケースに適しています。このモードでは、スループット MBps 、読み取りファンアウト、保持、リソース利用率（デフォルトは 60% ）などの要件を指定します。オンプレミス（ベアメタル、 VMware 、 Kubernetes 、 OpenStack ）やクラウドなどの環境にも移行できます。Kafka クラスタのサイジングは、この情報に基づいて、ブローカーに必要なサーバ数、 Zookeeper 、 Apache Kafka Connect Workers 、スキーマレジストリ、 REST プロキシ、 ksqlDB 、および Conluent コントロールセンターを提供します。

階層型ストレージの場合、 Kafka クラスタのサイジングのためのきめ細かい構成モードを検討してください。Granular モードは、経験豊富な Apache Kafka ユーザや明確に定義されたユースケースに適しています。このセクションでは、プロデューサ、ストリームプロセッサ、およびコンシューマのサイジングについて説明します。

プロデューサー

Apache Kafka の開発者（ネイティブクライアント、 REST プロキシ、 Kafka コネクタなど）については、次の情報を参照してください。

- * 名前。 * Spark 。
- * 開発者の種類。 * アプリケーションまたはサービス、プロキシ (REST 、 MQTT 、その他) 、および既存のデータベース (RDBMS, NoSQL 、その他) 。「わからない」を選択することもできます。
- * 平均スループット。 * 1 秒あたりのイベント数（ 1 、 000 、 000 など）。
- * 最大スループット。 * 1 秒あたりのイベント数（ 4 、 000 、 000 など）。
- * 平均メッセージサイズ。 * バイト単位、非圧縮（最大 1MB 、 1000 など）。
- * メッセージ形式。 * Avro 、 JSON 、プロトコルバッファ、バイナリ、テキスト、 「わからない」など。
- * 複製係数 * オプションは 1 、 2 、 3 （流暢な推奨）、 4 、 5 です。 または 6.
- * 保持時間。 * 1 日（例：）。Apache Kafka にデータを保存しておく期間を教えてください。無期限には、任意の単位で -1 を入力します。無制限の保持期間を 10 年と想定しています。
- [ブローカー数を減らし、 Infinite Storage を許可するための階層化ストレージを有効にする] のチェックボックスをオンにします。
- 階層型ストレージが有効になっている場合は ' リテンションフィールドによって ' ブローカにローカルに保存されるデータのホットセットが制御されますアーカイブ保持フィールドは、アーカイブオブジェクトストレージにデータを格納する期間を制御します。
- * アーカイブ・ストレージの保存期間 * 1 年（例：）データをアーカイブストレージに保存する期間無期限の任意の単位を指定して -1 を入力します。無制限の保持期間を 10 年と想定しています。
- * 成長乗数 * 1 （例：）。このパラメータの値が現在のスループットに基づく場合は、 1 に設定します。追加の増加に基づいてサイズを決定するには、このパラメータに増加率を設定します。
- * プロデューサーインスタンスの数。 * 10 （例：）実行されるプロデューサーインスタンスの数はいくつですか？この値は、 CPU 負荷をサイジング計算に含めるために必要です。空白の値は、 CPU 負荷が計算に組み込まれていないことを示します。

この入力例に基づいて、サイジングはプロデューサーに次のように影響します。

- 非圧縮バイト単位の平均スループット： 1Gbps圧縮されていないバイト数のピークスループット： 4Gbps圧縮されたバイト数の平均スループット： 400MBps 。圧縮バイトの最大スループット： 1.6GBps 。デフォルトの 60% の圧縮率に基づいています（この値は変更できます）。
  - 必要なホットセットストレージの合計数 :31 、 104TB (レプリケーションを含む) 、圧縮。必要なオフブローカーアーカイブストレージの合計： 378 、 432TB 、圧縮使用 "https://fusion.netapp.com" StorageGRID のサイジングの場合：

Stream Processors は、 Apache Kafka のデータを使用し、 Apache Kafka に生成するアプリケーションまたはサービスを記述する必要があります。ほとんどの場合、これらは ksqlDB ストリームまたは Kafka ストリームで構築されます。

- * 名前。 * Spark streamer 。
- * 処理時間。 * このプロセッサーは 1 つのメッセージを処理するのにどれくらいかかりますか？
  - 1 ミリ秒（シンプルでステートレスな変換）（例）、 10 ミリ秒（ステートフルなメモリ内動作）
  - 100ms （ステートフルネットワークまたはディスク処理）、 1000ms （サードパーティ製 REST コール）
  - このパラメータをベンチマークして、その所要時間を正確に把握しました。
- * 出力保持 * 1 日（例）ストリームプロセッサは Apache Kafka に出力を返します。この出力データを Apache Kafka にどれくらいの期間保存しますか？無期限の任意の単位を指定して -1 を入力します。
- ［ ブローカー数を減らし、 Infinite Storage を許可するには、 ［ 階層型ストレージを有効にする ］ チェックボックスをオンにします。
- * アーカイブ・ストレージの保存期間 * 1 年（例：）データをアーカイブストレージに保存する期間無期限の任意の単位を指定して -1 を入力します。無制限の保持期間を 10 年と想定しています。
- * 出力パススルー率。 * 100 （例：）ストリームプロセッサは Apache Kafka に出力を返します。Apache Kafka に出力されるインバウンドスループットの割合を教えてください。たとえば、インバウンドスループットが 20Mbps で、この値が 10 の場合、出力スループットは 2Mbps になります。
- この機能はどのアプリケーションから読み取られますか。「 Spark 」を選択すると、販売担当者タイプに基づいたサイジングで使用された名前になります。上記の入力に基づいて、ストリームプロセッサインスタンスとトピックパーティションの推定にサイジングを行うと、次のような効果が期待できます。
- このストリームプロセッサアプリケーションには、次の数のインスタンスが必要です。受信するトピックでは、多くのパーティションが必要になる場合があります。このパラメータを確認するには、 [流暢] に連絡してください
  - 平均スループットで 1 、 000 、増加率なし
  - 4 、 000 ：増加率のないピークスループット
  - 1 、 000 ：平均スループットと増加率
  - 4 、 000 ：最大スループットで増加率

消費者

Apache Kafka のデータを利用していて、 Apache Kafka にデータを生成していないアプリケーションやサービスについて説明してください。たとえば、ネイティブのクライアントや Kafka Connector などです。

- * 名前 . * Spark consumer 。
- * 処理時間。 * この消費者は、 1 つのメッセージを処理するのにどれくらいの時間がかかりますか。

    - 1 ミリ秒（シンプルでステートレスなロギングなどのタスク）
    - 10 ミリ秒（データストアへの高速書き込み）
    - 100 ミリ秒（データストアへの書き込み速度が遅い）
    - 1000 ミリ秒（サードパーティの REST コール）
    - その他のベンチマークされたプロセスの中には、既知の期間があります。
- * 消費者タイプ。 * アプリケーション、プロキシ、シンクを既存のデータストア（ RDBMS 、 NoSQL など）に。
- この機能はどのアプリケーションから読み取られますか。このパラメータは、以前に決定したプロデューサーおよびストリームのサイジングを使用して接続します。

上記の入力に基づいて、コンシューマインスタンスとトピックパーティションの推定サイズを決定する必要があります。コンシューマアプリケーションには、次の数のインスタンスが必要です。

- 平均スループットで 2 、 000 、増加率はゼロ
- 8 、 000 ：ピークスループットで、増加率はゼロです
- 平均スループットで 2 、 000 、増加率も含まれます
- 8 、 000 ：最大スループット。増大の乗数も含まれます

受信トピックでは、この数のパーティションも必要になる場合があります。確認のため、流暢な連絡をします。

生産者、ストリームプロセッサ、および消費者の要件に加えて、次の追加要件を提供する必要があります。

- * 再構築時間。 * 例： 4 時間。Apache Kafka ブローカーホストで障害が発生すると、そのデータは失われ、障害が発生したホストと交換するために新しいホストをプロビジョニングすると、この新しいホストをどのくらいの速さで再構築する必要がありますか？値が不明な場合は、このパラメータを空白のままにします。
- * リソース使用率目標（パーセンテージ）。 * 例： 60 。平均スループット中にホストをどの程度使用しますか？Conluent の自己バランシングクラスタを使用していない場合は、 60% の使用率を推奨します。この場合、使用率が高くなります。

環境の説明

- * どの環境でクラスターを実行しますか？ * Amazon Web Services 、 Microsoft Azure 、 Google クラウドプラットフォーム、オンプレミスのベアメタル、 VMware オンプレミス、 OpenStack をオンプレミスで運用するのか、 Kubernates をオンプレミスで運用するのか
- * ホストの詳細。 * コアの数： 48 （例：）、ネットワークカードのタイプ（ 10GbE 、 40GbE 、 16GbE 、 1GbE 、またはその他のタイプ）。
- * ストレージボリューム。 * ホスト： 12 （例：）。ホストあたりのハードドライブまたは SSD の数はいくつですか。競合するホストごとに 12 台のハードドライブを推奨します。
- * ストレージ容量 / ボリューム（ GB 単位）。 * 1000 （例：）。1 つのボリュームストアでギガバイト単位のストレージ容量はどれくらいですか？競合する場合は 1TB のディスクを使用します。
- * ストレージ構成 * ストレージ・ボリュームの構成方法競合製品は、 Conluent のすべての機能を活用するために RAID10 を推奨しています。JBOD 、 SAN 、 RAID 1 、 RAID 0 、 RAID 5 、 その他のタイプもサポートされています。

- * 単一ボリュームのスループット（ Mbps ）。 * 125 （例：）1 つのストレージボリュームで 1 秒あたりのメガバイト数で読み取りまたは書き込みを行うことができる速度はどれくらいですか。競合するハードディスクドライブは、通常 125 Mbps のスループットを持つ標準ハードディスクドライブをお勧めします。
- * メモリ容量（ GB ）。 * 64 （例）。

環境変数を決定したら、 Size my Cluster （マイクラスタのサイズ）を選択します。前述の例に基づいて、Con裕福 な Kafka のサイジングを決定しました。

- * Apache Kafka * Broker count ： 22 。クラスタはストレージバウンドです。階層型ストレージを有効にして、ホスト数を減らし、ストレージを無制限にすることを検討してください。
- * Apache ZooKeeper. * Count ： 5 ； Apache Kafka Connect Workers ： Count ： 2 ； Schema Registry ： Count ： 2 ； REST Proxy ： Count ： 2 ； ksqlDB ： Count ： 2 ； Conluent Control Center ： Count ： 1 。

ユースケースを考慮せずに、プラットフォームチームにリバースモードを使用する。パーティションモードを使用して、 1 つのトピックに必要なパーティションの数を計算します。を参照してください https://eventsizer.io リバースモードとパーティションモードに基づいたサイジングの場合

まとめ

このドキュメントでは、検証テスト、階層型ストレージのパフォーマンス結果、調整、S3 コネクタの堪能、セルフバランシング機能など、ネットアップストレージでの Conluent Tiered Storage の使用に関するベストプラクティスを紹介しています。ILM ポリシー、検証のための複数のパフォーマンステストと業界標準の S3 API を使用した流暢なパフォーマンスを考慮した場合、 NetApp StorageGRID オブジェクトストレージは流暢な階層化ストレージに最適な選択肢です。

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- Apache Kafka とは何ですか

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

- ネットアップの製品マニュアル

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

- S3 シンクパラメータの詳細

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

- Apache Kafka です

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

- Conluent Platform の無限のストレージ

"https://www.confluent.io/blog/infinite-kafka-storage-in-confluent-platform/"

- 競合する階層型ストレージ - ベストプラクティスとサイジング

  "https://docs.confluent.io/platform/current/kafka/tiered-storage.html#best-practices-and-recommendations"

- Conluent Platform 用の Amazon S3 シンクコネクタ

  "https://docs.confluent.io/kafka-connect-s3-sink/current/overview.html"

- Kafka のサイジング

  "https://eventsizer.io"

- StorageGRID のサイジング

  "https://fusion.netapp.com/"

- Kafka のユースケース

  "https://kafka.apache.org/uses"

- 統合されたプラットフォーム 6.0 の自律分散 Kafka クラスター

  "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/"

ネットアップのハイブリッドクラウドデータソリューション - Spark と Hadoop はお客様のユースケースに基づいています

TR-4657 ：ネットアップのハイブリッドクラウドデータソリューション - Spark と Hadoop はお客様のユースケースに基づいています

ネットアップ、 Karthikeyan Nagalingam と Sathish Thyagarajan

本ドキュメントでは、 NetApp AFF および FAS ストレージシステム、 NetApp Cloud Volumes ONTAP 、ネットアップ接続ストレージ、 Spark および Hadoop 向けの NetApp FlexClone テクノロジを使用したハイブリッドクラウドデータソリューションについて説明します。これらの解決策アーキテクチャを使用することで、お客様の環境に適したデータ保護解決策を選択できます。ネットアップは、お客様とのやり取りと、お客様のビジネスユースケースに基づいてこれらのソリューションを設計しました。このドキュメントでは、次の詳細情報を提供します。

- Spark 環境や Hadoop 環境、お客様の課題に対応するデータ保護が必要な理由
- ネットアップのビジョンと、そのビルディングブロックとサービスを基盤とするデータファブリック。
- これらのビルディングブロックを使用して、柔軟なデータ保護ワークフローを構築する方法

- 実際のお客様のユースケースに基づく、複数のアーキテクチャの長所と短所各ユースケースには、次のコンポーネントがあります。
  - お客様のシナリオ
  - 要件と課題
  - 解決策
  - ソリューションの概要

Hadoop のデータ保護を選ぶ理由

Hadoop 環境と Spark 環境では、次の点に注意する必要があります。

- * ソフトウェアや人為的なエラー。 * Hadoop データの処理中にソフトウェアを更新したときに人的エラーが発生すると、業務によって原因が予期せぬ結果を招く可能性がある動作不良になることがあります。このような場合は、障害や妥当でない結果が生じないように、データを保護する必要があります。たとえば、ソフトウェアアップデートの実行が不十分でトラフィック信号分析アプリケーションが実行されたため、トラフィック信号データをプレーンテキスト形式で適切に分析できない新機能があります。ソフトウェアは JSON やその他の非テキストファイル形式を分析して、リアルタイムトラフィック制御分析システムを生成し、データポイントが不足している予測結果を生成します。このような状況では、原因が出力不良の可能性があり、交通信号で事故につながるおそれがあります。データ保護機能を使用すると、以前の作業中のアプリケーションバージョンにすばやくロールバックできるため、この問題に対応できます。
- * サイズと拡張性。 * 分析データのサイズは日々増え続けています。その理由は、データソースとボリュームの数が増え続けることにあります。現在のビッグデータ市場では、ソーシャルメディア、モバイルアプリ、データ分析、クラウドコンピューティングの各プラットフォームがデータの主要なソースとなっており、データは急速に増加しています。そのため、データを保護して、正確なデータ運用を確保する必要があります。
- * Hadoop のネイティブデータ保護。 * Hadoop には、データを保護するためのネイティブコマンドがありますが、このコマンドはバックアップ中のデータの整合性を提供しません。ディレクトリレベルのバックアップのみをサポートします。Hadoop によって作成された Snapshot は読み取り専用であり、バックアップデータを直接再利用することはできません。

Hadoop や **Spark** のお客様にとって、データ保護の課題が発生しています

Hadoop と Spark のお客様にとってよくある課題は、データ保護の際に本番クラスタのパフォーマンスに悪影響を与えることなく、バックアップ時間を短縮し、バックアップの信頼性を向上させることです。

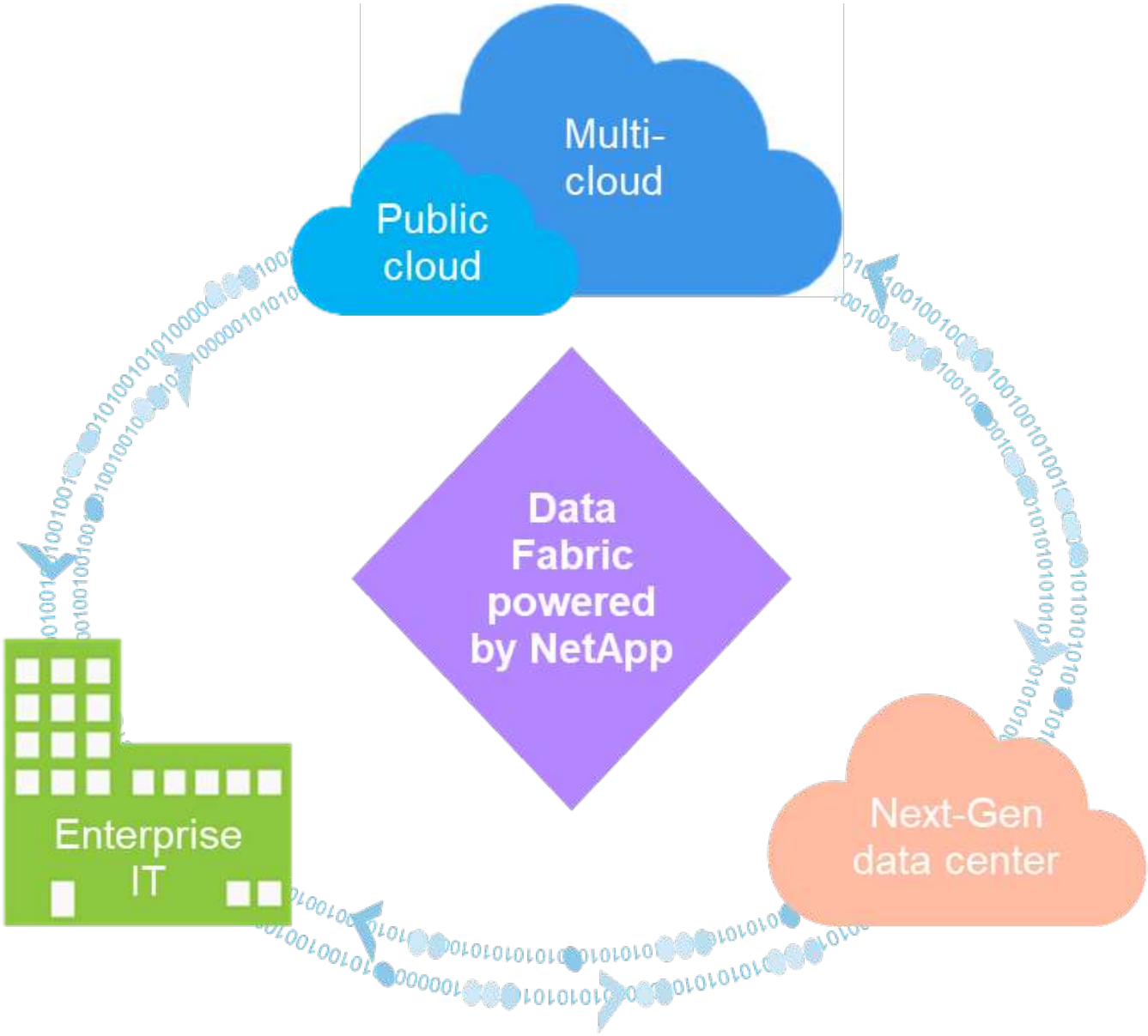
また、 RPO （目標復旧時点）と RTO （目標復旧時間）のダウンタイムを最小限に抑え、オンプレミスとクラウドベースのディザスタリカバリサイトを制御して、ビジネス継続性を最適化する必要もあります。この制御は、通常、エンタープライズレベルの管理ツールを使用して行われます。

データ量が膨大で増え続けているだけでなく、データの到着率も増加しているため、 Hadoop 環境と Spark 環境は複雑化しています。このようなシナリオでは、ソースデータから効率的で最新の DevTest 環境と QA 環境を迅速に構築することは困難です。ネットアップはこれらの課題を認識し、本ホワイトペーパーで紹介しているソリューションを提供しています。

ネットアップのデータファブリックを基盤としたビッグデータアーキテクチャ

ネットアップのデータファブリックは、クラウド環境とオンプレミス環境全体でデータ管理を簡易化、統合することで、デジタル変革を加速します。

ネットアップのデータファブリックは、一貫した統合的データ管理サービスとアプリケーション（ビルディングブロック）を提供し、データの可視性と分析、データのアクセスと制御、データの保護とセキュリティを実現します。以下の図を参照してください。



実績のあるデータファブリックのユースケース

ネットアップのデータファブリックは、以下の 9 つのユースケースをお客様に提供します。

- 分析ワークロードを高速化
- DevOps 変革を加速
- クラウドとホスティングのインフラ構築
- クラウドデータサービスを統合
- データの保護とセキュリティ
- 非構造化データを最適化

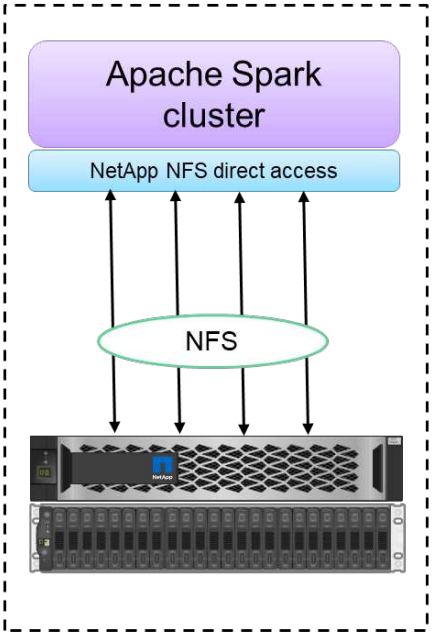
- データセンターの効率化
- データの分析と管理を実現
- 簡易化と自動化

このドキュメントでは、 9 つのユースケースのうち 2 つを取り上げ、それぞれのソリューションを紹介します。

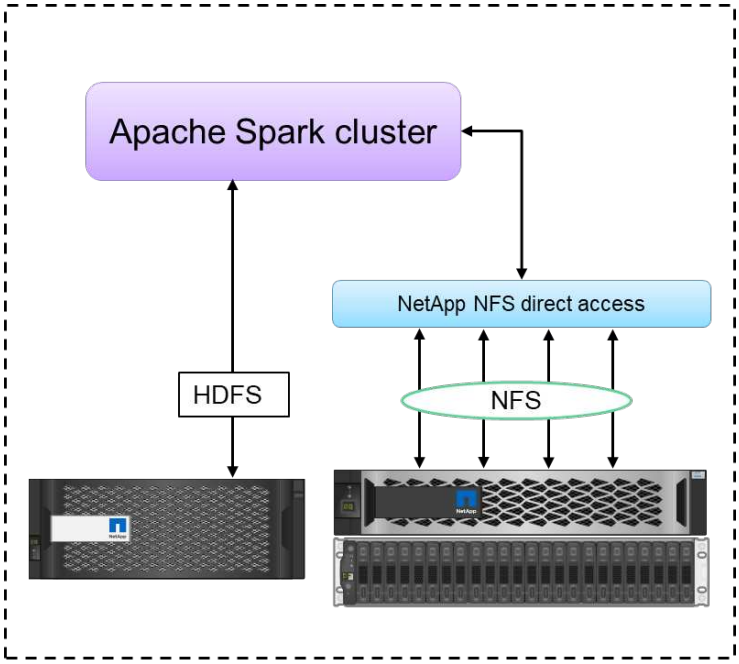
- 分析ワークロードを高速化
- データの保護とセキュリティ

NetApp NFS への直接アクセス

NetApp NFSを使用すると、既存または新規のNFSv3 / NFSv4データに対して、データを移動またはコピーすることなくビッグデータ分析ジョブを実行できます。データの複数のコピーが作成されるため、ソースとデータを同期する必要がありません。たとえば、金融機関では、ある場所から別の場所へデータを移動する際に法的義務を果たす必要がありますが、これは容易な作業ではありません。このシナリオでは、 NetApp NFS の直接アクセスによって、元の場所から財務データが分析されます。もう 1 つの主な利点は、 NetApp NFS 直接アクセスを使用すると、ネイティブの Hadoop コマンドを使用して Hadoop データを保護しやすくなることと、ネットアップの充実したデータ管理ポートフォリオを活用してデータ保護ワークフローを実現できることです。



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

NetApp NFS 直接アクセスでは、 Hadoop クラスタと Spark クラスタに対して次の 2 種類の導入オプションを提供しています。

- デフォルトでは、 Hadoop / Spark クラスタは、データストレージとデフォルトのファイルシステムに Hadoop Distributed File System （ HDFS ； Hadoop 分散ファイルシステム）を使用しています。NetApp NFS の直接アクセスを使用すると、デフォルトの HDFS をデフォルトのファイルシステムとして NFS ストレージに置き換えることができるため、 NFS データに対する直接分析処理が可能になります。
- もう 1 つの導入オプションでは、 NetApp NFS 直接アクセスを使用して、 1 つの Hadoop / Spark クラスタ内に HDFS を追加のストレージとして構成することもできます。この場合、 NFS エクスポートを介し

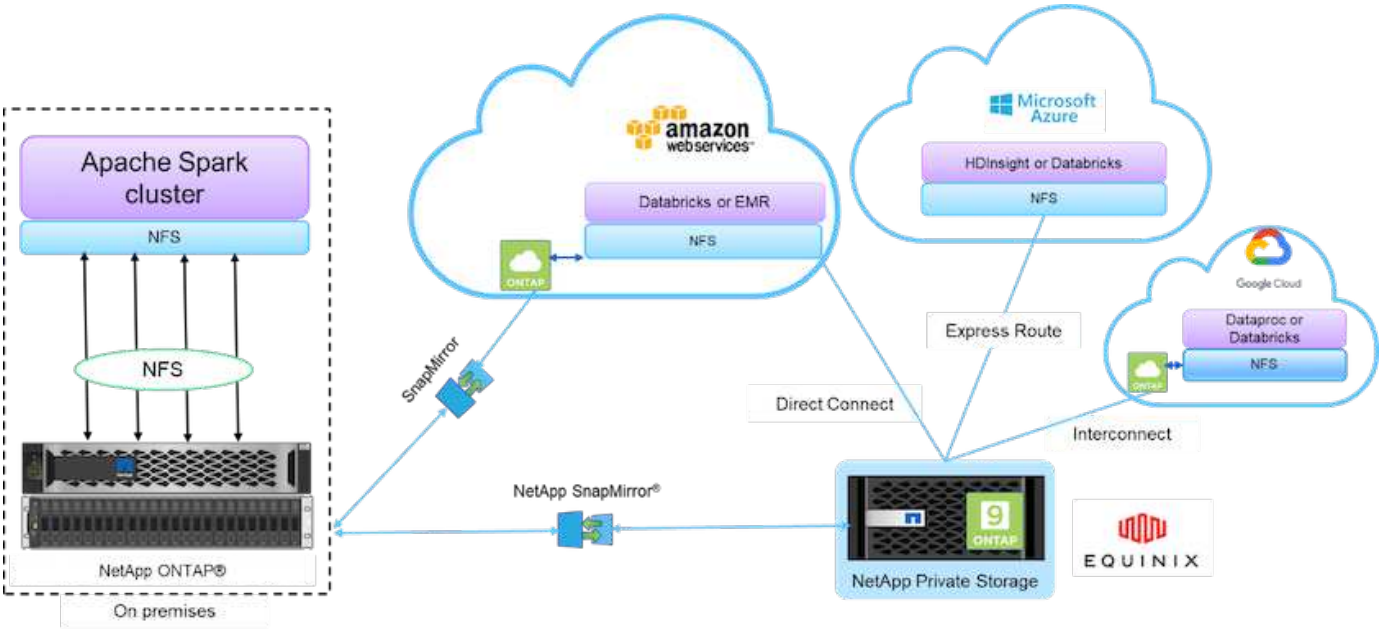
てデータを共有し、 HDFS データと同じクラスタからデータにアクセスできます。

NetApp NFS 直接アクセスを使用する主な利点は次のとおりです。

- 現在の場所からデータを分析するため、分析データを HDFS などの Hadoop インフラに移動する時間とパフォーマンスのかかるタスクは発生しません。
- レプリカの数を 3 つから 1 つに減らします。
- ユーザはコンピューティングとストレージを切り離して個別に拡張できます。
- ONTAP の豊富なデータ管理機能を活用して、エンタープライズデータを保護します。
- Hortonworks データプラットフォームで認定されています。
- ハイブリッドデータ分析環境を実現
- 動的なマルチスレッド機能を活用して、バックアップ時間を短縮します。

ビッグデータ向けのビルディングブロック

ネットアップのデータファブリックは、以下の図に示すように、データアクセス、制御、保護、セキュリティのためのデータ管理サービスとアプリケーション（ビルディングブロック）を統合しています。



上の図の構成要素は次のとおりです。

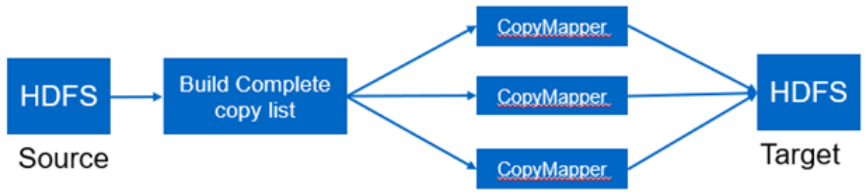
- * NetApp NFS 直接アクセス。 * 最新の Hadoop クラスタと Spark クラスタを、ソフトウェアやドライバの追加の必要なしに NetApp NFS ボリュームに直接アクセスできます。
- * ネットアップの Cloud Volumes ONTAP とクラウドボリュームサービス。 * ソフトウェア定義型の接続ストレージ。 Amazon Web Services （ AWS ）で実行されている ONTAP または Microsoft Azure クラウドサービスで実行されている Azure NetApp Files （ ANF ）に基づいています。
- * NetApp SnapMirror テクノロジ * 。オンプレミスと ONTAP クラウドインスタンスまたは NPS インスタンス間でデータ保護機能を提供します。
- * クラウド・サービス・プロバイダー。 * これらのプロバイダーには、 AWS 、 Microsoft Azure 、 Google Cloud 、 IBM Cloud が含まれます。

- * PaaS * AWS の Amazon Elastic MapReduce （ EMR ）や Databricks 、 Microsoft Azure HDInsight 、 Azure Databricks などのクラウドベースの分析サービスを利用できます。

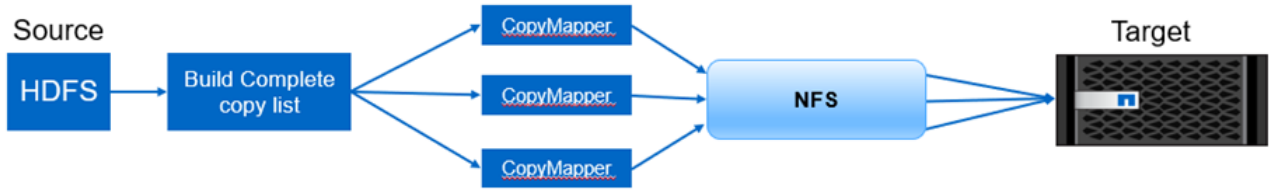
Hadoop データ保護機能とネットアップ

Hadoop ディストリビュータは、大規模なクラスタ間コピーとクラスタ内コピーに使用されるネイティブツールです。次の図に示す Hadoop ディストリビュータの基本的なプロセスは、 MapReduce などの Hadoop ネイティブツールを使用した一般的なバックアップワークフローで、 HDFS ソースから対応するターゲットに Hadoop データをコピーします。

NetApp NFS の直接アクセスを使用すると、 Hadoop DistCp ツールのターゲットデスティネーションとして NFS を設定し、 HDFS ソースから MapReduce 経由で NFS 共有にデータをコピーできます。NetApp NFS への直接アクセスは、 DistCp ツールの NFS ドライバとして機能します。



Hadoop Distcp Basic Process



Hadoop Distcp and NetApp

Hadoop のデータ保護のユースケースの概要

このセクションでは、データ保護のユースケースの概要を概要で説明します。これが、本ドキュメントで重要となるのです。以降のセクションでは、それぞれのユースケースについて、お客様の問題（シナリオ）、要件と課題、ソリューションなどの詳細を説明します。

使用事例 1 ： Hadoop データのバックアップ

このユースケースでは、NetApp NFSボリュームを使用することで、大規模な金融機関では、長いバックアップウィンドウ時間を24時間以上からわずか数時間に短縮することができました。

ユースケース 2 ：クラウドからオンプレミスへのバックアップとディザスタリカバリ

大規模な放送会社では、ネットアップのデータファブリックをビルディングブロックとして使用することで、オンデマンド、瞬時、データ転送などのさまざまなデータ転送モードに応じて、クラウドデータをオンプレミスのデータセンターにバックアップするという要件を満たすことができました。 または、 Hadoop / Spark の

クラスタの負荷に基づいて計算されました。

ユースケース **3** ：既存の **Hadoop** データに対して **DevTest** を有効化

ネットアップのソリューションは、オンラインの音楽配信企業が、スペース効率に優れた複数の Hadoop クラスタをさまざまなブランチオフィスに迅速に構築し、レポートを作成したり、定期的なポリシーを使用して日々の DevTest タスクを実行したりできるよう支援しました

ユースケース **4** ：データ保護とマルチクラウド接続

ある大手サービスプロバイダは、ネットアップのデータファブリックを使用して、さまざまなクラウドインスタンスからお客様にマルチクラウド分析を提供していました。

ユースケース **5** ：分析ワークロードを高速化

最大規模の金融サービスおよび投資銀行の 1 つは、ネットアップのネットワーク接続型ストレージ解決策を使用して、 I/O 待ち時間を短縮し、定量的な金融分析プラットフォームを高速化しました。

使用事例 **1** ： **Hadoop** データのバックアップ

このシナリオでは、大規模なオンプレミスの Hadoop リポジトリがあり、ディザスタリカバリのためにバックアップを作成したいと考えています。しかし、お客様の現在のバックアップ解決策はコストが高く、 24 時間以上のバックアップウィンドウに悩まされています。

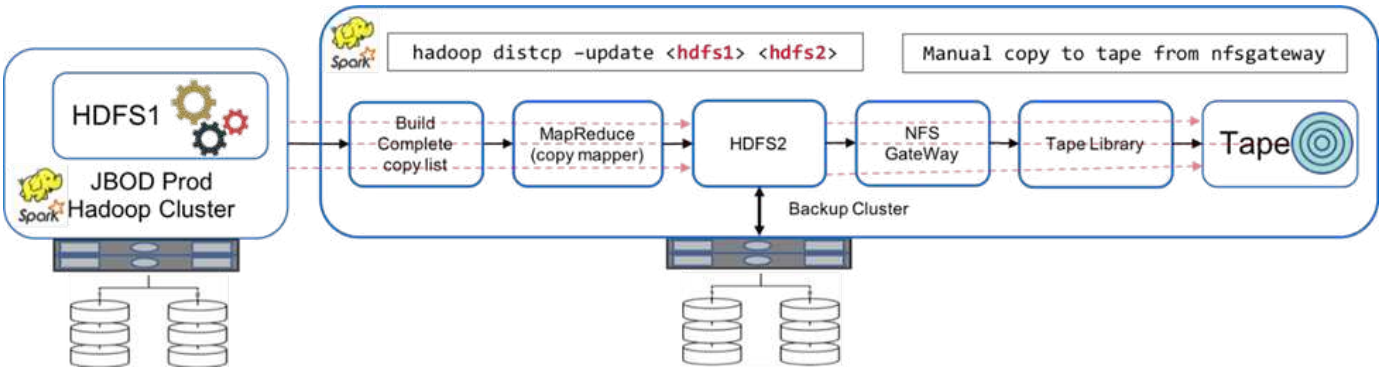
要件と課題

このユースケースの主な要件と課題は次のとおりです。

- ソフトウェアの下位互換性：
  - 提案する代替バックアップ解決策は、本番用 Hadoop クラスタで現在実行しているソフトウェアバージョンと互換性があることが必要です。
- コミットされた SLA を満たすためには、代替の解決策で非常に低い RPO と RTO を達成することを推奨します。
- ネットアップのバックアップ解決策で作成したバックアップは、データセンターのローカルに構築された Hadoop クラスタ、およびリモートサイトのディザスタリカバリロケーションで実行されている Hadoop クラスタで使用できます。
- 提案する解決策は対費用効果が高いものでなければなりません。
- 提案する解決策は、バックアップ処理中に実行中の本番環境の分析ジョブに与えるパフォーマンスへの影響を軽減する必要があります。

お客様の既存のバックアップソリューション**x**

次の図は、元の Hadoop ネイティブのバックアップ解決策を示しています。



本番環境のデータは、中間バックアップクラスタを通じてテープに保護されます。

- hadoop distcp-update <hdfs1 ><hdfs2>` コマンドを実行することにより、 HDFS1 データが HDFS2 にコピーされます。
- バックアップ・クラスタは NFS ゲートウェイとして機能し ' テープ・ライブラリを介して Linux’cp' コマンドを使用してデータを手動でテープにコピーします

元の Hadoop ネイティブバックアップ解決策には次のようなメリットがあります。

- 解決策は Hadoop ネイティブのコマンドをベースにしているため、新しい手順を習得する必要がなくなります。
- 解決策は、業界標準のアーキテクチャとハードウェアを活用しています。

元の Hadoop ネイティブバックアップ解決策には、次のような欠点があります。

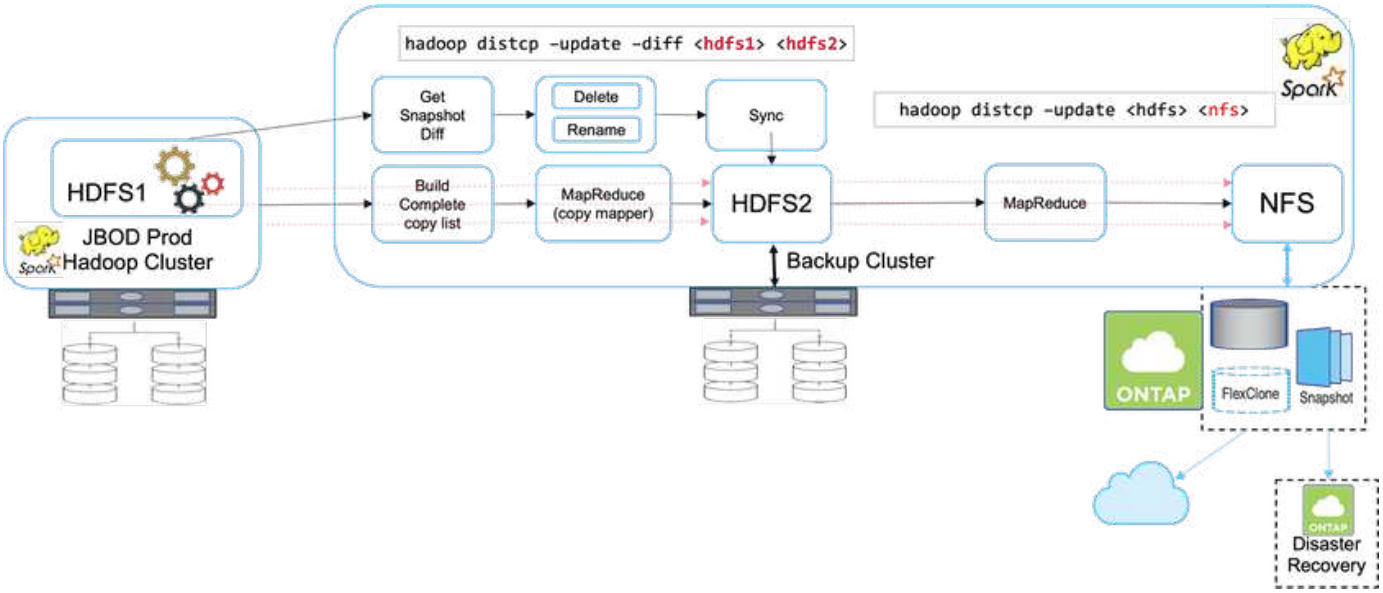
- バックアップ時間が長いと 24 時間を超えるため、本番環境のデータが脆弱になります。
- バックアップ時間中にクラスタのパフォーマンスが大幅に低下します。
- テープへのコピーは手動で行います。
- バックアップ解決策は、必要なハードウェアと、手動プロセスに必要な人的時間の点でコストが高くなります。

バックアップソリューション

これらの課題と要件に基づいて、既存のバックアップシステムを検討し、 3 つのバックアップソリューションを提案しました。以降のサブセクションでは、解決策 A ～ 解決策 C というラベルの付いた 3 種類のバックアップソリューションについて説明します

解決策 A の略

解決策Aでは、次の図に示すように、バックアップHadoopクラスタからNetApp NFSストレージシステムにセカンダリバックアップが送信されるため、テープは必要ありません。



解決策 A の詳細なタスクは次のとおりです。

- 本番環境の Hadoop クラスタには、保護が必要な HDFS 内のお客様の分析データがあります。
- HDFS を使用するバックアップ Hadoop クラスタは、データの中間的な場所として機能します。Just a Bunch of Disks （ JBOD ）は、本番環境の Hadoop クラスタとバックアップの Hadoop クラスタの両方で HDFS にストレージを提供する。
- 「 hadoop distcp – update – diff<hdfs1 ><hdfs2>` コマンド」を実行することで、 Hadoop 本番クラスタの HDFS からバックアップクラスタの HDFS へと Hadoop 本番データを保護します。

> Hadoop スナップショットは、本番環境からバックアップ Hadoop クラスタへデータを保護するために使用されます。

- NetApp ONTAP ストレージコントローラは、バックアップ Hadoop クラスタにプロビジョニングされる NFS エクスポートボリュームを提供します。
- を実行します `Hadoop distcp` コマンドMapReduceと複数のマッパーを活用して、分析データをバックアップHadoopクラスタからNFSに保護します。

  ネットアップストレージシステム上の NFS にデータを格納したあと、必要に応じて、ネットアップの Snapshot 、 SnapRestore 、および FlexClone テクノロジを使用して Hadoop データをバックアップ、リストア、および複製します。

> Hadoop データは、 SnapMirror テクノロジを使用してクラウドやディザスタリカバリロケーションに保護できます。

解決策 A には、次のような利点があります。

- Hadoop の本番データはバックアップクラスタから保護されます。
- HDFS データは NFS を通じて保護されるため、クラウドやディザスタリカバリの場所を保護できます。
- バックアップ処理をバックアップクラスタにオフロードすることでパフォーマンスを向上します。
- 手動でのテープ操作が不要になります

- ネットアップのツールを使用してエンタープライズ管理機能を利用できます。
- 既存の環境への変更は最小限で済みます。
- 対費用効果の高い解決策です。

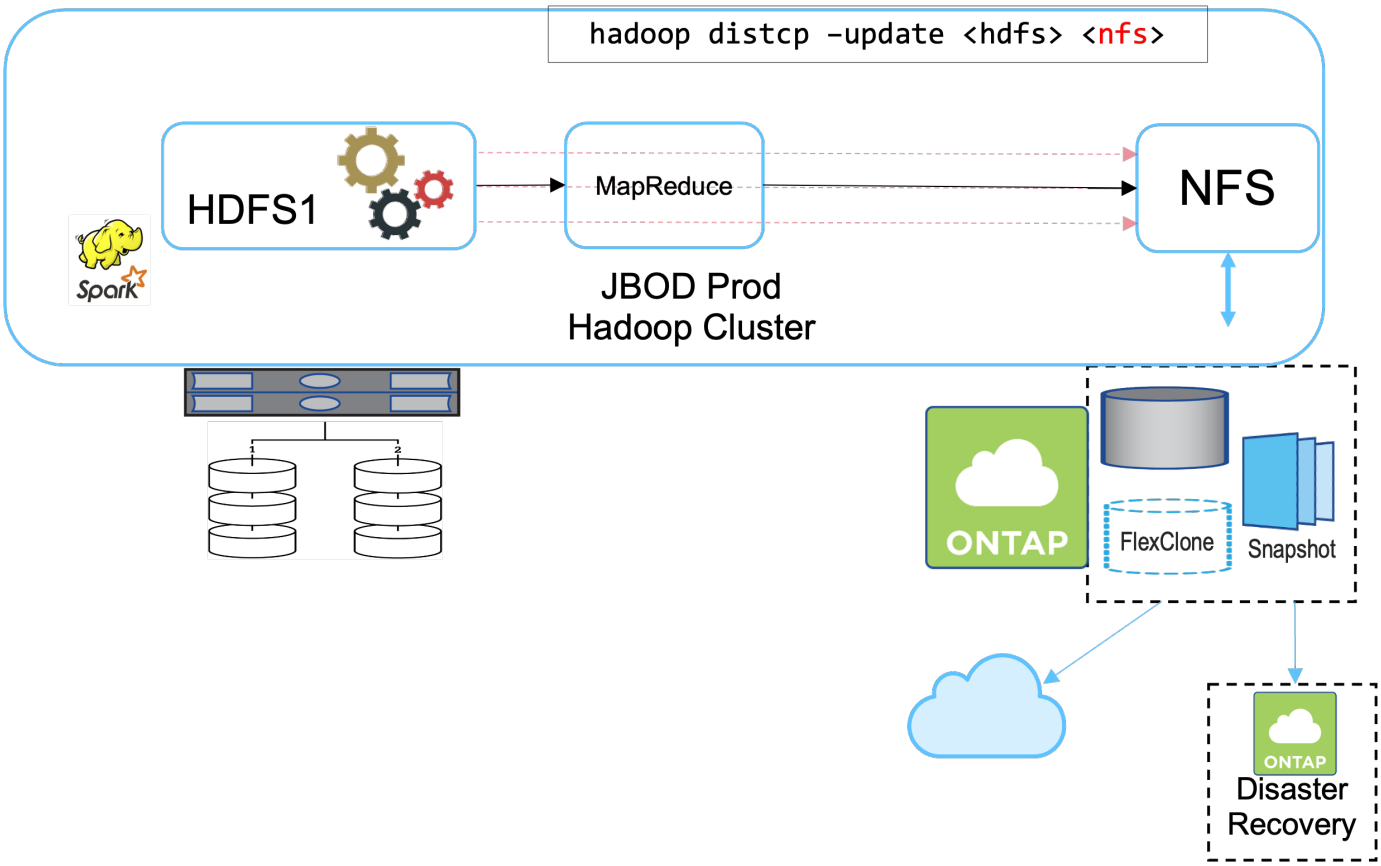
この解決策の欠点は、パフォーマンスを向上させるためにバックアップクラスタと追加のマッパが必要であることです。

お客様は最近、解決策 A を導入しました。シンプルさ、コスト、全体的なパフォーマンスが理由です。

この解決策では、 JBOD の代わりに ONTAP の SAN ディスクを使用できます。このオプションを選択すると、バックアップクラスタのストレージ負荷が ONTAP にオフロードされますが、問題となるのは SAN ファブリックスイッチが必要な場合です。

解決策 B

解決策BはNFSボリュームを本番用Hadoopクラスタに追加するため、次の図に示すように、バックアップHadoopクラスタは必要ありません。



解決策 B の詳細なタスクは次のとおりです。

- NetApp ONTAP ストレージコントローラは、本番用 Hadoop クラスタに対して NFS エクスポートをプロビジョニングします。

  Hadoopネイティブ `hadoop distcp` コマンドは、Hadoopデータを本番用クラスタのHDFSからNFSに保護します。

- ネットアップストレージシステム上の NFS にデータを格納したあと、 Snapshot 、 SnapRestore 、およ

び FlexClone テクノロジを使用して、必要に応じて Hadoop データをバックアップ、リストア、および複製します。

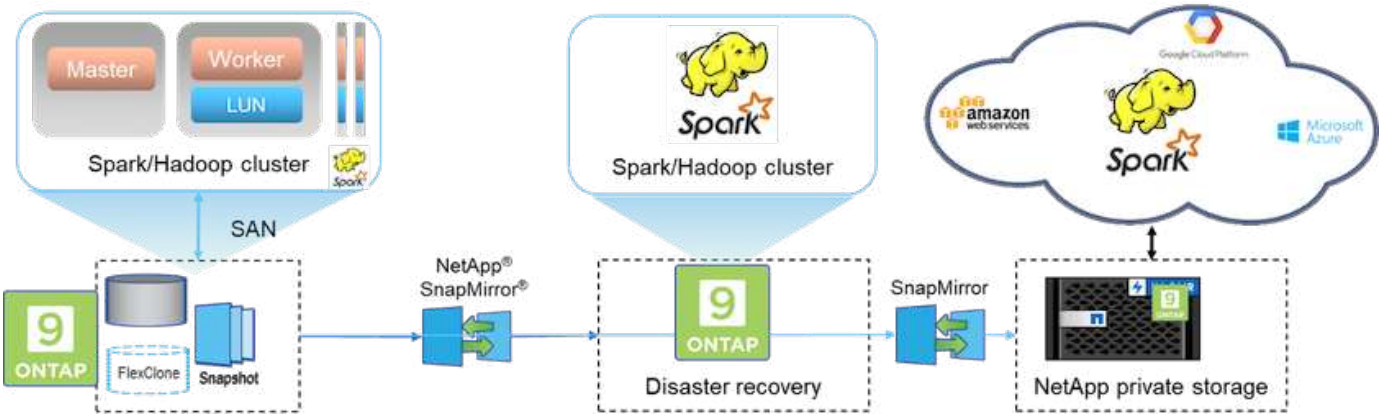
解決策 B には次のような利点があります。

- 本番環境クラスタは、バックアップ解決策用に若干変更されるため、実装が簡単になり、インフラコストを削減できます。
- バックアップ処理のためのバックアップクラスタは必要ありません。
- HDFS の本番環境のデータは、 NFS データへの変換によって保護されます。
- 解決策では、ネットアップのツールを使用してエンタープライズ管理機能を実行できます。

この解決策の欠点は、本番クラスタに実装されており、本番クラスタに管理者タスクを追加できることです。

解決策 C

解決策 C では、次の図に示すように、 NetApp SAN ボリュームが HDFS ストレージの Hadoop 本番クラスタに直接プロビジョニングされます。



解決策 C の詳細な手順は次のとおりです。

- NetApp ONTAP SAN ストレージは、 HDFS データストレージの本番用 Hadoop クラスタでプロビジョニングされます。
- NetApp Snapshot テクノロジと SnapMirror テクノロジを使用して、本番用 Hadoop クラスタの HDFS データをバックアップします。
- バックアップはストレージレイヤにあるため、 Snapshot コピーのバックアッププロセス中は Hadoop / Spark クラスタの本番環境でパフォーマンスが低下することはありません。

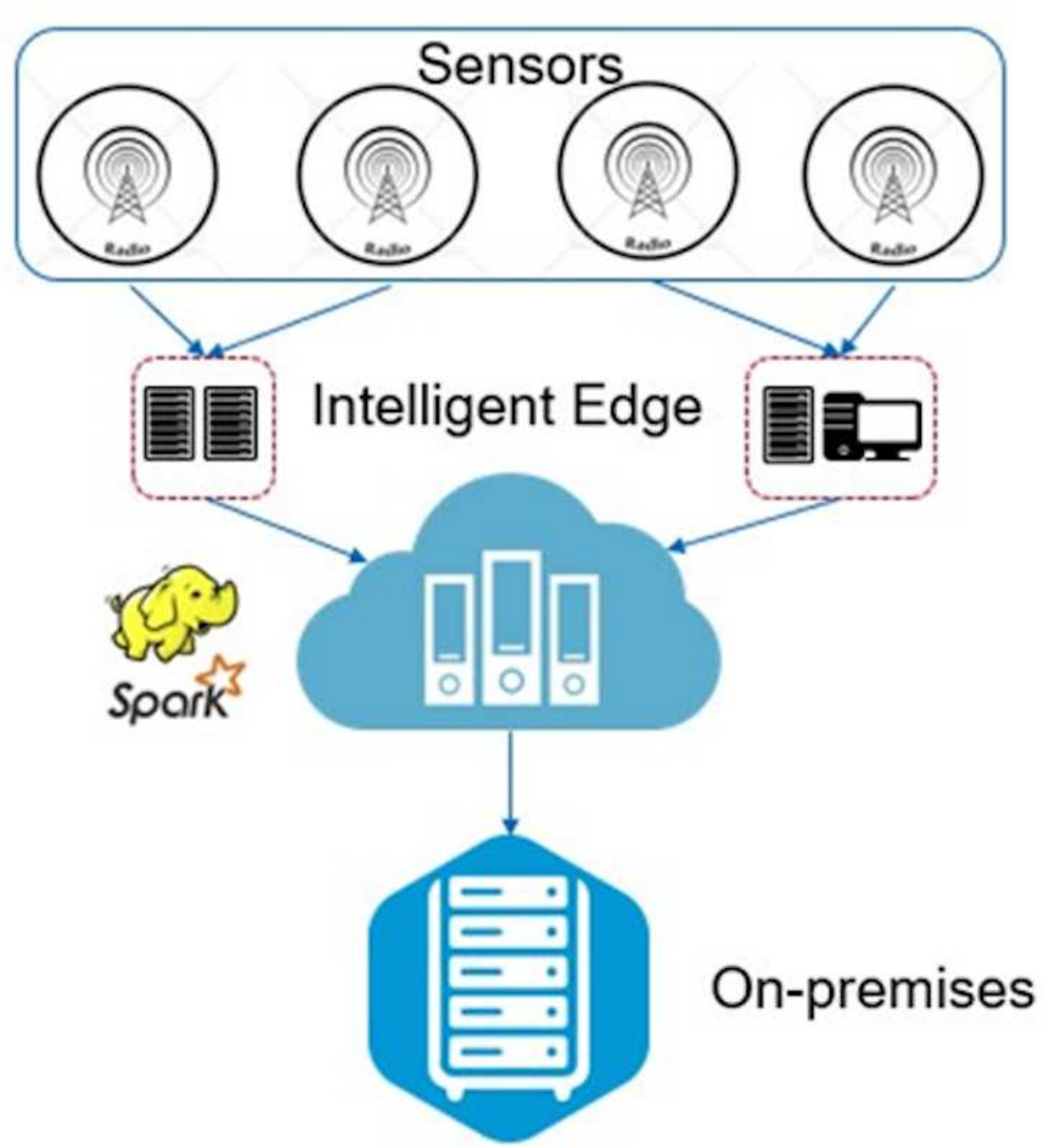
> Snapshot テクノロジを使用すると、データのサイズに関係なく数秒で完了するバックアップを作成できます。

解決策 C には次のような利点があります。

- スペース効率に優れたバックアップは、 Snapshot テクノロジを使用して作成できます。
- ネットアップのツールを使用してエンタープライズ管理機能を利用できます。

ユースケース **2** ：クラウドからオンプレミスへのバックアップとディザスタリカバリ

このユースケースは、放送局のお客様がクラウドベースの分析データをオンプレミスのデータセンターにバックアップする必要がある場合を基準にしています。以下の図を参照してください。



シナリオ（ **Scenario** ）

このシナリオでは、 IoT センサーのデータがクラウドに取り込まれ、 AWS 内のオープンソースの Apache Spark クラスタを使用して分析されます。処理されたデータをクラウドからオンプレミスにバックアップすることが要件です。

要件と課題

このユースケースの主な要件と課題は次のとおりです。

- データ保護を有効原因にしても、本番環境の Spark / Hadoop クラスタのパフォーマンスへの影響は一切ありません。
- 効率的かつ安全な方法で、クラウドセンサーデータをオンプレミスに移動して保護する必要があります。
- オンデマンド、瞬時、クラスタの低負荷時など、さまざまな条件下でクラウドからオンプレミスにデータを柔軟に転送できます。

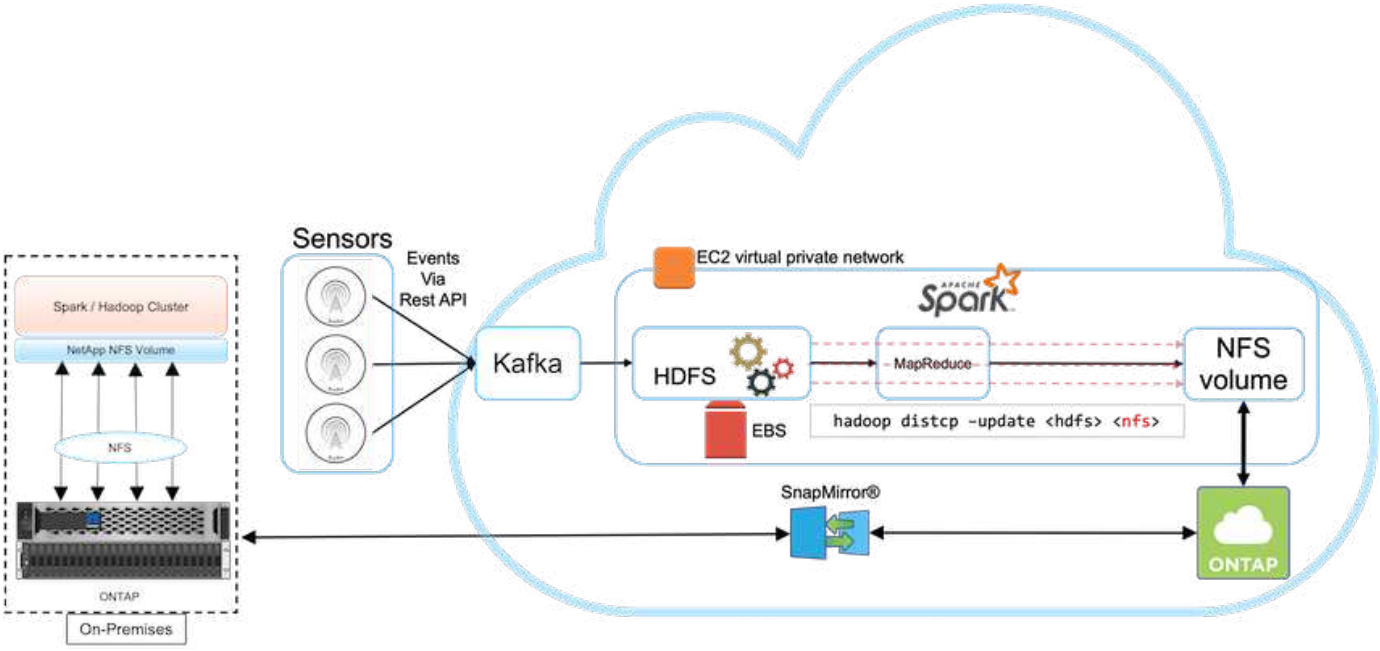
解決策

お客様は、 Spark クラスタの HDFS ストレージとして AWS Elastic Block Store （ EBS ）を使用して、 Kafka 経由でリモートセンサーからデータを受け取り、取り込むことになりました。そのため、 HDFS ストレージはバックアップデータのソースとして機能します。

これらの要件を満たすために、 NetApp ONTAP Cloud が AWS に導入され、 Spark / Hadoop クラスタのバックアップターゲットとして機能する NFS 共有が作成されます。

NFS共有が作成されたら、HDFS EBSストレージのデータをONTAP NFS共有にコピーします。データがONTAP クラウド上の NFS に配置されると、 SnapMirror テクノロジを使用して、必要に応じてクラウドからオンプレミスのストレージにデータを安全かつ効率的にミラーリングできます。

この図は、クラウドからオンプレミスの解決策へのバックアップとディザスタリカバリを示しています。



ユースケース 3 ：既存の Hadoop データに対して DevTest を有効化

このユースケースでは、 DevTest と Reporting の目的で同じデータセンターとリモートサイトに大量の分析データを格納した既存の Hadoop クラスタをベースに、新しい Hadoop / Spark クラスタを迅速かつ効率的に構築することがお客様の要件となります。

シナリオ（ **Scenario** ）

このシナリオでは、大規模な Hadoop データレイク実装をオンプレミスとディザスタリカバリサイトで使用して、複数の Spark / Hadoop クラスタを構築しています。

要件と課題

このユースケースの主な要件と課題は次のとおりです。

- DevTest 、 QA 用など、同じ本番環境のデータへのアクセスを必要とする用途に複数の Hadoop クラスタを作成この課題は、非常に大規模な Hadoop クラスタを、スペース効率に優れた方法で何度も瞬時にクローニングすることです。
- 運用効率を高めるために、 Hadoop データを DevTest チームとレポートチームに同期します。
- 業務用クラスタと新規クラスタに同じクレデンシャルを使用して Hadoop データを分散します。
- スケジュールされたポリシーを使用して、本番クラスタに影響を与えずに効率的に QA クラスタを作成

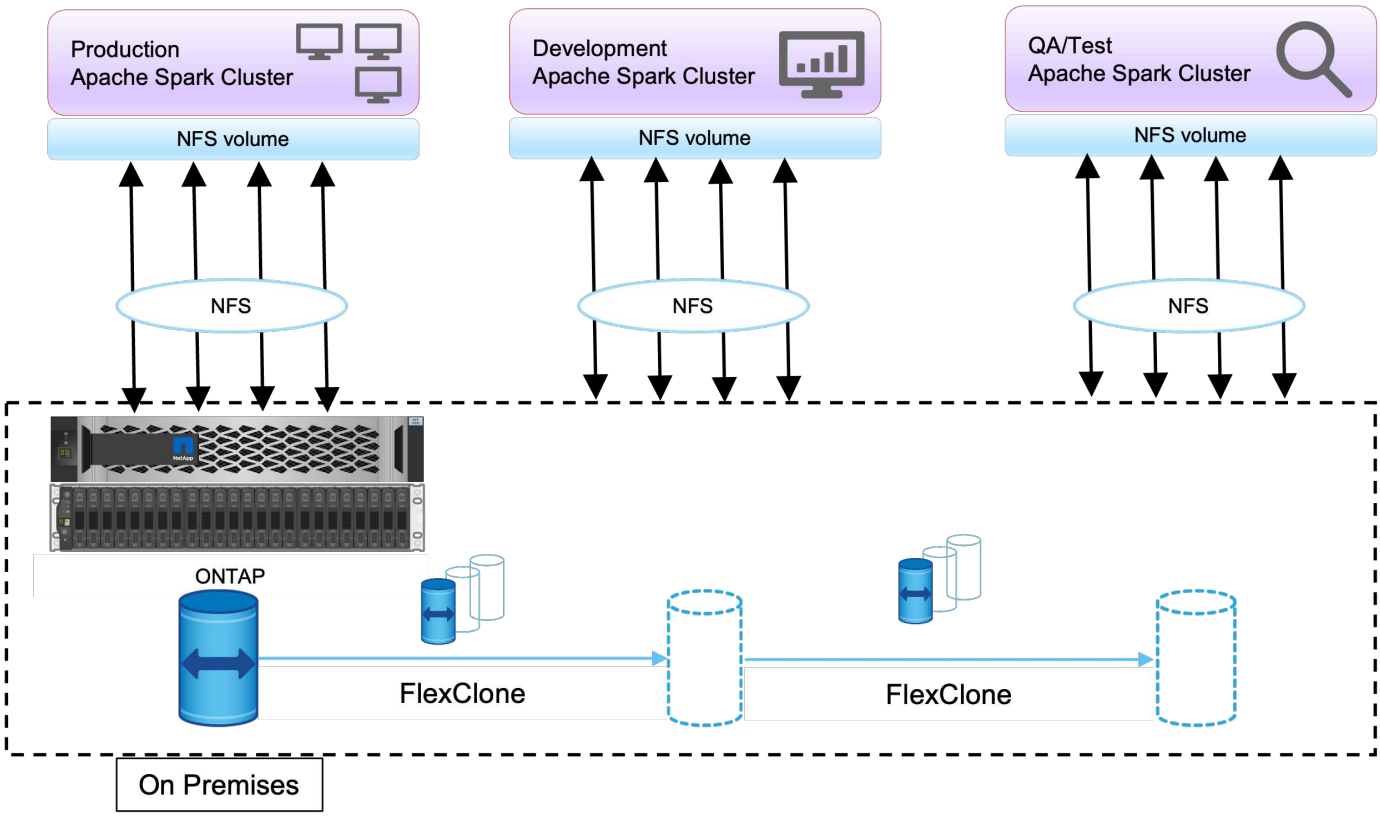
解決策

FlexClone テクノロジは、直前に説明した要件を回答に適用するために使用されます。FlexClone テクノロジは、 Snapshot コピーの読み取り / 書き込みコピーです。親 Snapshot コピーのデータを読み取り、新規または変更されたブロック用に追加のスペースのみを消費します。高速でスペース効率に優れています。

まず、ネットアップの整合グループを使用して既存のクラスタの Snapshot コピーを作成し、

NetApp System Manager またはストレージ管理プロンプト内の Snapshot コピー。整合グループ Snapshot コピーはアプリケーションと整合性のあるグループ Snapshot コピーであり、整合グループ Snapshot コピーに基づいて FlexClone ボリュームが作成されます。FlexClone ボリュームは親ボリュームの NFS エクスポートポリシーを継承することに留意する必要があります。Snapshot コピーの作成後、次の図に示すように、 DevTest および Reporting 用に新しい Hadoop クラスタをインストールする必要があります。新しいHadoopクラスタのクローンNFSボリュームがNFSデータにアクセスします。

この図は、 DevTest 用の Hadoop クラスタを示しています。



ユースケース 4 ：データ保護とマルチクラウド接続

このユースケースは、お客様のビッグデータ分析データにマルチクラウド接続を提供するという課題を抱えるクラウドサービスパートナーに適しています。

シナリオ（ **Scenario** ）

このシナリオでは、さまざまなソースから AWS で受信した IoT データが NPS の中央の場所に保存されます。NPS ストレージは、 AWS と Azure 上にある Spark / Hadoop クラスタに接続されています。これにより、同じデータにアクセスする複数のクラウドで実行されるビッグデータ分析アプリケーションを実現できます。

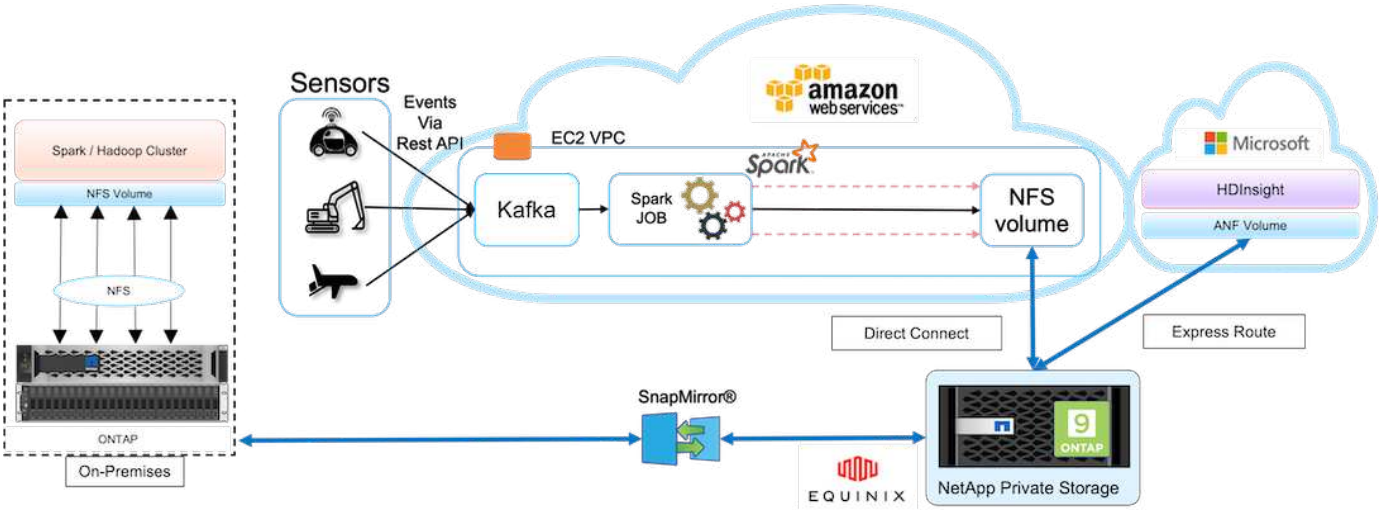
要件と課題

このユースケースの主な要件と課題は次のとおりです。

- お客様は、複数のクラウドを使用して、同じデータに対して分析ジョブを実行したいと考えています。
- オンプレミスやクラウドなどのさまざまなソースから、さまざまなセンサーやハブを介してデータを受信する必要があります。
- 解決策は、効率性とコスト効率に優れている必要があります。
- 主な課題は、オンプレミスと異なるクラウドの間でハイブリッド分析サービスを提供する、対費用効果の高い効率的な解決策を構築することです。

解決策

この図は、データ保護とマルチクラウド接続解決策を示しています。



上の図に示すように、センサーからのデータはストリーミングされ、 Kafka を介して AWS Spark クラスタに取り込まれます。データは NPS 内の NFS 共有に格納されます。 NPS は、 Equinix データセンター内のクラウドプロバイダの外部にあります。NetApp NPSはDirect Connect接続とExpress Route接続を介してAmazon AWSとMicrosoft Azureに接続されるため、お客様はAmazonとAWSの両方の分析クラスタからNFSデータにアクセスできます。このアプローチは、複数のハイパースケーラにわたるクラウド分析の実現を解決します。

そのため、オンプレミスと NPS ストレージはどちらも ONTAP ソフトウェアを実行するため、 SnapMirror を使用して NPS データをオンプレミスクラスタにミラーリングし、オンプレミスと複数のクラウドにわたるハイブリッドクラウド分析を実現できます。

パフォーマンスを最大限に高めるために、通常は複数のネットワークインターフェイスと直接接続 / エクスプレスルートを使用してクラウドインスタンスからデータにアクセスすることを推奨します。

ユースケース **5** ：分析ワークロードを高速化

このシナリオでは、 NetApp NFS ストレージ解決策を使用して大規模な金融サービスおよび投資銀行の分析プラットフォームを最新化し、資産管理および定量的ビジネスユニットの投資リスクおよび派生物の分析を大幅に改善しました。

シナリオ（ **Scenario** ）

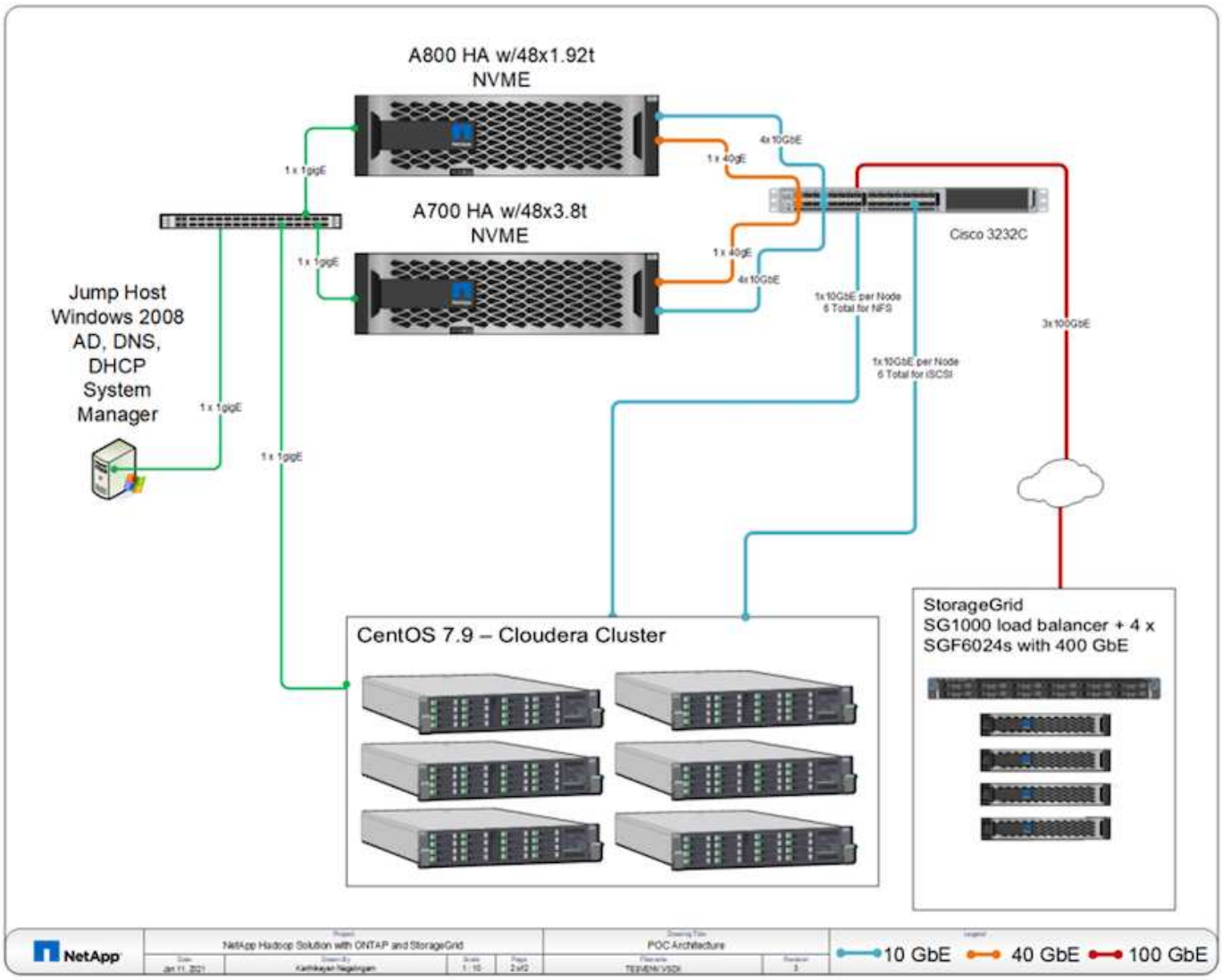
お客様の既存の環境では、分析プラットフォームに使用される Hadoop インフラストラクチャは、 Hadoop サーバの内部ストレージを活用しています。JBOD 環境の専有特性により、組織内の多くの社内顧客は、リアルタイムデータの繰り返しサンプルに依存するシミュレーションであるモンテカルロ定量モデルを利用できませんでした。市場動向の不確実性の影響を理解するのに最適な能力は、量的資産管理事業部門にとって好ましくないものとなっていました。

要件と課題

銀行の定量事業部門は、正確でタイムリーな予測を実現するための効率的な予測方法を求めていました。そのためには、インフラを刷新し、既存の I/O 待機時間を短縮し、 Hadoop や Spark などの分析アプリケーションのパフォーマンスを向上させて、投資モデルを効率的にシミュレートし、潜在的な利益を測定し、リスクを分析する必要性を認識しました。

解決策

お客様は、既存の Spark 解決策の JBOD を使用していました。その後、 NetApp ONTAP 、 NetApp StorageGRID 、 MinIO Gateway to NFS を活用して、銀行の定量的財務グループの I/O 待機時間を短縮し、潜在的な利益とリスクを評価する投資モデルのシミュレーションと分析を実行しました。この図は、 Spark の解決策とネットアップストレージを示しています。



上の図に示すように、 Spark 搭載の 6 ノード Hadoop クラスタで NFS プロトコルと S3 プロトコルを使用して寄木細工のファイルにアクセスするために AFF A800 、 A700 システム、 StorageGRID を導入し、データ分析処理用に糸と Hive のメタデータサービスを用意しました。

お客様の古い環境にある DAS （直接接続型ストレージ）解決策には、コンピューティングとストレージを個別に拡張するという欠点がありました。NetApp ONTAP 解決策 for Spark を使用することで、銀行の財務分析事業部門はストレージをコンピューティングから切り離し、必要に応じてインフラリソースをより効率的に提供することができました。

NFS で ONTAP を使用することで、 Spark の SQL ジョブにはコンピュートサーバの CPU がほぼフルに活用され、 I/O 待機時間が 70% 近く削減されました。その結果、 Spark のワークロードの処理能力とパフォーマンスが向上しました。また、 CPU 利用率の向上により、お客様は GPUDirect などの GPU を活用してプラットフォームをさらに最新化できるようになりました。さらに、 StorageGRID は Spark のワークロードに低コストのストレージオプションを提供し、 MinIO Gateway は S3 プロトコル経由で NFS データへの安全なアク

セスを提供します。クラウド内のデータには、 Cloud Volumes ONTAP 、 Azure NetApp Files 、 NetApp Cloud Volumes Service を推奨します。

まとめ

このセクションでは、ネットアップが提供するさまざまな Hadoop データ保護要件を満たすユースケースとソリューションの概要を説明します。ネットアップのデータファブリックを使用することで、お客様は次のことが可能になります。

- ネットアップの充実したデータ管理機能と Hadoop ネイティブワークフローとの統合により、適切なデータ保護ソリューションを柔軟に選択できます。
- Hadoop クラスタのバックアップ時間を約 70% 短縮します。
- Hadoop クラスタのバックアップによるパフォーマンスへの影響を排除します。
- 異なるクラウドプロバイダからのマルチクラウドデータ保護とデータアクセスを、単一の分析データソースに同時に提供できます。
- FlexClone テクノロジを使用すると、スペース効率に優れた高速な Hadoop クラスタコピーを作成できます。

追加情報の参照先

このドキュメントに記載されている情報の詳細については、以下のドキュメントや Web サイトを参照してください。

- ネットアップのビッグデータ分析ソリューション

  "https://www.netapp.com/us/solutions/applications/big-data-analytics/index.aspx"

- ネットアップストレージを使用した Apache Spark ワークロード

  https://www.netapp.com/pdf.html?item=/media/26877-nva-1157-deploy.pdf

- ネットアップの Apache Spark 向けストレージソリューション

  "https://www.netapp.com/media/16864-tr-4570.pdf"

- ネットアップが有効にしたデータファブリック上の Apache Hadoop

  "https://www.netapp.com/media/16877-tr-4529.pdf"

謝辞

- ネットアップ、 ANZ 地域セールス担当、 Paul Burland 氏
- ネットアップ、ビジネス開発マネージャー、 Hoseb Dermanilian 氏
- ネットアップ、 MPSG ディレクター、 Lee Dorrior 氏
- ネットアップ、 ANZ ビクトリア地区 SE 、システムエンジニア David Thiessen 氏

バージョン履歴

バージョン	日付	ドキュメントのバージョン履歴
バージョン 1.0 以降	2018 年 1 月	初版リリース
バージョン 2.0 以降	2021年10月	ユースケース 5 ：分析ワークロードの高速化を更新
バージョン 3.0 以降	2023年11月	削除されたNIPAMの詳細

最新のデータ分析-さまざまな分析戦略に対応するさまざまなソリューション

このホワイトペーパーでは、ネットアップの最新のデータ分析解決策 戦略について説明します。また、ビジネス成果、お客様の課題、テクノロジトレンド、競合他社のレガシーアーキテクチャ、最新のワークフロー、 ユースケース、業界、クラウド、テクノロジパートナー、Data Mover、 NetApp Active IQデジタルアドバイザ（デジタルアドバイザとも呼ばれる）、NetApp DataOpsツールキット、HadoopからSpark、NetApp Astra Controlを使用したSoftware-Defined Storage、コンテナ、エンタープライズデータの管理、アーカイブ、階層化により、AIと分析の目標を達成し、NetAppとお客様が連携してデータアーキテクチャを最新化する方法を実現します。

"最新のデータ分析-さまざまな分析戦略に対応するさまざまなソリューション"

TR-4869：『NetApp StorageGRID with Splunk SmartStore』

カーティケヤンナガリンガム、ボビー・オムメン、ジョセフ・カンダティルバビル

Splunk Enterpriseは、市場をリードするSecurity Information and Event Management（SIEM）解決策 であり、セキュリティ、IT、DevOpsの各チームの成果を生み出します。データ量は急激に増大し続けており、この膨大なリソースを活用できる大企業にとって大きな販売機会となります。Splunk Enterpriseは、今後もさまざまなユースケースに採用されていきます。ユースケースの増加に伴い、Splunk Enterpriseが取り込んで処理するデータ量も増えています。Splunk Enterpriseの従来型アーキテクチャは、優れたデータアクセスと可用性を提供する分散型スケールアウト設計です。しかし、このアーキテクチャを使用している企業は、急速に拡大するデータ量に対応するための拡張に伴うコストの増大に直面しています。

Splunk SmartStoreとNetApp StorageGRID は、コンピューティングとストレージが分離された新しい導入モデルを提供することで、この課題を解決します。また、Splunkエンタープライズ環境に比類のない拡張性と柔軟性をもたらす解決策 を導入して、単一サイトから複数サイトに拡張できます。コンピューティングとストレージを個別に拡張し、コストを削減しながら、コスト効率に優れたクラウドベースのS3オブジェクトストレージにインテリジェントな階層化を追加することで、コストを削減できます。

解決策 を使用すると、検索パフォーマンスを維持しながらローカルストレージ内のデータ量を最適化できるため、コンピューティングとストレージをオンデマンドで拡張できます。SmartStoreは、データアクセスパターンを自動的に評価して、リアルタイム分析用にアクセス可能なデータと、低コストのS3オブジェクトスト

レージに格納するデータを決定します。

このテクニカルレポートでは、Splunk SmartStore解決策 にネットアップが提供するメリットを紹介するとともに、環境内のSplunk SmartStoreの設計とサイジングを行うためのフレームワークを紹介します。その結果、シンプルで拡張性と耐障害性に優れた解決策 が実現し、説得力のあるTCOが実現します。StorageGRIDは、拡張性と対費用効果に優れたS3プロトコル/ APIベースのオブジェクトストレージであり、リモートストレージとも呼ばれます。そのため、Splunk解決策 を低コストで拡張しながら、耐障害性を高めることができます。

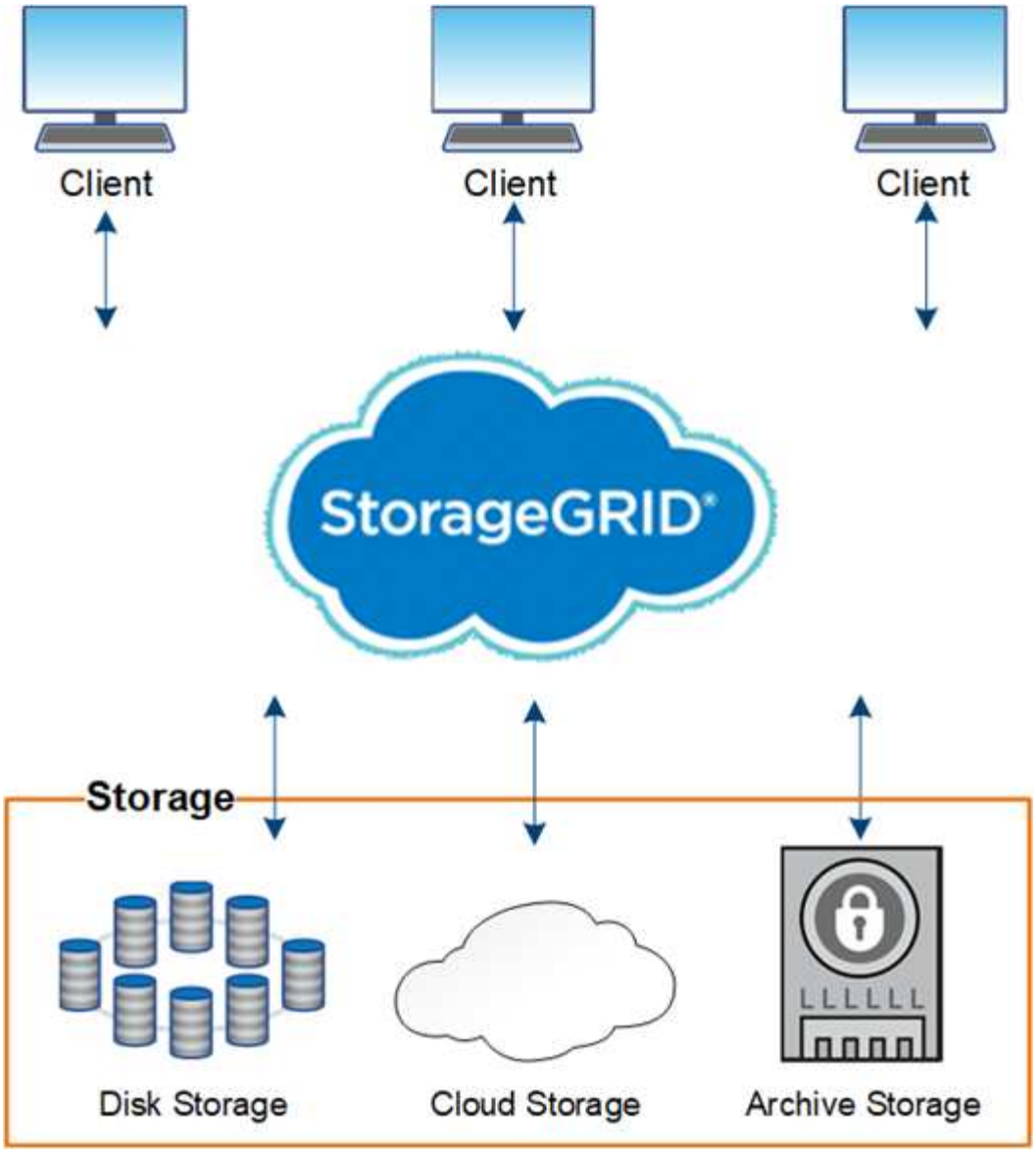
> Splunk SmartStoreは、オブジェクトストレージをリモートストアまたはリモートストレージ階層と呼んでいます。

NetApp StorageGRID について

NetApp StorageGRID は、大規模アーカイブ、メディアリポジトリ、Webデータストア向けの、ソフトウェア定義型のオブジェクトストレージ解決策 です。ネットアップは、StorageGRID を導入したことで、業界をリードするイノベーションソリューションとデータ管理ソリューションの提供に関して20年に及ぶ経験を活かし、オンプレミス環境とパブリッククラウド環境、プライベートクラウド環境、ハイブリッドクラウド環境の両方で、情報の価値を最大限に管理、最大限に引き出しています。

StorageGRID は、大規模な非構造化データを長期間保管するためのセキュアなストレージを提供します。メタデータベースの統合ライフサイクル管理ポリシーによって、データのライフサイクルを通して最適な保存先が選択されます。コンテンツは適切な場所、適切なタイミングで、適切なストレージ階層に配置されるため、コストを削減できます。単一のネームスペースを使用すると、StorageGRID ストレージの地理的な場所に関係なく、単一の呼び出しでデータにアクセスできます。データセンターとクラウドインフラの間に複数のStorageGRID インスタンスを導入、管理できます。

StorageGRID システムは、グローバルに分散された冗長で種類の異なる複数のノードで構成され、既存および次世代のクライアントアプリケーションと統合することができます。



IDC MarketScape：Worldwide Object-Based Storage 2019 Vendor Assessmentで、ネットアップを最新レポートのリーダーに位置付けました。StorageGRID は、最も要件の厳しい業界で20年近くの本番環境の導入実績があり、非構造化データのリーダーとして認められています。

StorageGRID を使用すると、次のことが可能になります。

- 複数のStorageGRID インスタンスを導入して、データセンターとクラウドの間のあらゆる場所から、数百ペタバイトまで簡単に拡張できる単一のネームスペースを通じてデータにアクセスします。
- 複数のインフラにわたって導入と一元管理を柔軟に行うことができます。
- 階層型イレイジャーコーディング（EC）を活用し、99.999%を超えるデータ保持性で、卓越したデータ保持性を実現します。
- Amazon S3 GlacierとAzure Blobに検証済みの統合により、ハイブリッドマルチクラウド機能をさらに有効化
- 独自のAPIやベンダーロックインを使用せずに、改ざん防止機能を備えたデータ保持によって、法規制義務を満たし、コンプライアンスを促進します。

StorageGRID を使用して、最も複雑な非構造化データ管理の問題を解決する方法の詳細については、を参照

してください "NetApp StorageGRID のホームページ"。

Splunk Enterpriseについて

Splunk Enterpriseは、データを活用するためのプラットフォームです。ログファイル、Webサイト、デバイス、センサー、アプリケーションなど、さまざまなソースで生成されたデータがSplunkのインデクサーに送信されて解析されるため、豊富な分析情報をデータから取得できます。データ漏えいの特定、お客様や製品のトレンドの指摘、インフラの最適化の機会の発見、さまざまなユースケースにわたる実用的な分析情報の提供などが含まれます。

Splunk SmartStoreについて

Splunk SmartStoreを使用すると、Splunkアーキテクチャのメリットがさらに広がり、コスト効率の高い方法で拡張できます。コンピューティングリソースとストレージリソースを分離することで、I/O用に最適化されたインデクサノードには、データのサブセットだけをキャッシュとして格納できるため、ストレージの必要性を大幅に削減できます。コンピューティングリソースとストレージリソースのどちらか1つしか必要ない場合は追加しなくても、コストを大幅に削減できます。対費用効果が高く、拡張性に優れたS3ベースのオブジェクトストレージを使用できます。これにより、環境がさらに簡易化され、コストが削減され、より大規模なデータセットを保持できます。

Splunk SmartStoreは、次のような企業に大きな価値をもたらします。

- コストが最適化されたS3オブジェクトストレージにウォームデータを移動することで、ストレージコストを削減する
- ストレージとコンピューティングを分離してシームレスに拡張
- 耐障害性に優れたクラウドネイティブストレージを活用して、ビジネス継続性を簡易化

TR-4623：『NetApp E-Series E5700 and Splunk Enterprise』

Mitch Blackburn、ネットアップ

TR-4623に、NetApp EシリーズとSplunkの設計を統合したアーキテクチャを示します。ノードストレージのバランス、信頼性、パフォーマンス、ストレージ容量、密度を最適化この設計では、Splunkクラスタ化インデックスのノードモデルを採用し、拡張性を高めてTCOを削減しています。ストレージをコンピューティングから切り離すことで、それぞれを個別に拡張できるため、オーバープロビジョニングを行うコストを削減できます。また、Splunkのマシンログイベントシミュレーションツールから取得したパフォーマンステスト結果をまとめたものです。

"TR-4623：『NetApp E-Series E5700 and Splunk Enterprise』"

NVA-1157 -導入：ネットアップストレージ解決策 を使用したApache Sparkワークロード

ネットアップ Karthikeyan Nagalingam

NVA-1157-deployでは、NetApp NFS AFF ストレージシステムでのApache Spark SQLのパフォーマンスと機能性検証について説明しています。さまざまなシナリオに基づいた構成、アーキテクチャ、パフォーマンステストのほか、SparkとNetApp ONTAP データ管理ソフトウェアの使用に関する推奨事項を確認します。また、NetApp AFF A800ストレージコントローラに比べて、Just a Bunch of Disks（JBOD）に基づくテスト結果についても説明します。

"NVA-1157 -導入：ネットアップストレージ解決策 を使用したApache Sparkワークロード"