現代化資料分析
NetApp Solutions

NetApp
April 12, 2024

目錄

NetApp現代化資料分析解決方案

使用 NetApp 檔案物件雙重性和 AWS SageMaker 進行雲端資料管理

TR-4967 ：使用 NetApp 檔案物件雙重性和 AWS SageMaker 進行雲端資料管理

NetApp的Karthithkeyan Nagalingam

資料科學家和工程師通常需要存取 NFS 格式中儲存的資料、但直接從 AWS SageMaker 中的 S3 傳輸協定存取這些資料可能很困難、因為 AWS 只支援 S3 儲存區存取。不過、NetApp ONTAP 可為 NFS 和 S3 啟用雙傳輸協定存取、提供解決方案。有了這個解決方案、資料科學家和工程師就能透過 NetApp Cloud Volumes ONTAP 的 S3 儲存區、從 AWS SageMaker 筆記型電腦存取 NFS 資料。這種方法可讓您輕鬆從 NFS 和 S3 存取和共用相同的資料、而不需要額外的軟體。

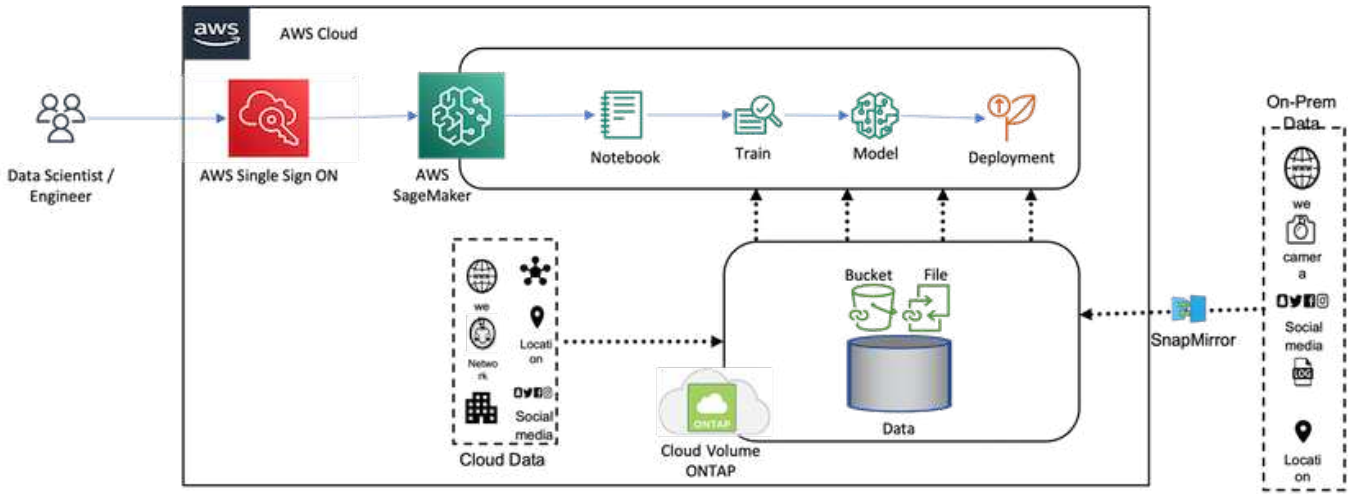
解決方案技術

本解決方案採用下列技術：

- * AWS SageMaker Notebook 。 * 為開發人員和資料科學家提供機器學習能力、以有效建立、訓練和部署高品質的 ML 模型。
- * NetApp BlueXP. * 可在內部部署、 AWS 、 Azure 和 Google Cloud 上探索、部署及操作儲存設備。它可提供資料保護、防範資料遺失、網路威脅和非計畫性中斷、並最佳化資料儲存和基礎架構。
- * NetApp Cloud Volumes ONTAP 。 * 在 AWS 、 Azure 和 Google Cloud 上提供企業級儲存磁碟區的 NFS 、 SMB/CIFS 、 iSCSI 和 S3 傳輸協定、讓使用者能夠更靈活地存取和管理雲端中的資料。

NetApp Cloud Volumes ONTAP 是從 BlueXP 建立來儲存 ML 資料。

下圖顯示解決方案的技術元件。



使用案例摘要

NFS 和 S3 的雙傳輸協定存取可能的使用案例、是在機器學習和資料科學領域。例如、資料科學家團隊可能正在使用 AWS SageMaker 進行機器學習專案、這需要存取 NFS 格式儲存的資料。不過、資料也可能需要透過 S3 儲存區來存取和共享、以便與其他團隊成員協同作業、或與其他使用 S3 的應用程式整合。

藉由使用 NetApp Cloud Volumes ONTAP 、團隊可以將資料儲存在單一位置、並可透過 NFS 和 S3 傳輸協定進行存取。資料科學家可直接從 AWS SageMaker 存取 NFS 格式的資料、而其他團隊成員或應用程式則可透過 S3 儲存區存取相同的資料。

這種方法可讓您輕鬆且有效率地存取和共用資料、而無需在不同的儲存解決方案之間進行額外的軟體或資料移轉。它也能讓團隊成員之間的工作流程和協同作業更有效率、進而更快、更有效地開發機器學習模式。

資料科學家和其他應用程式的資料雙重性

資料可在 NFS 中取得、並可從 AWS SageMaker 從 S3 存取。

技術需求

您需要 NetApp BlueXP 、 NetApp Cloud Volumes ONTAP 和 AWS SageMaker 筆記型電腦來處理資料雙重用途使用案例。

軟體需求

下表列出實作使用案例所需的軟體元件。

軟體	數量
藍圖	1.
NetApp Cloud Volumes ONTAP	1.
AWS SageMaker 筆記型電腦	1.

部署程序

部署資料雙重性解決方案涉及下列工作：

- BlueXP Connector
- NetApp Cloud Volumes ONTAP
- 用於機器學習的資料
- AWS SageMaker
- 通過 Jupyter 筆記型電腦驗證的機器學習

BlueXP 連接器

在此驗證中、我們使用 AWS 。也適用於 Azure 和 Google Cloud 。若要在 AWS 中建立 BlueXP Connector 、請完成下列步驟：

1. 我們使用的認證是以 BlueXP 中的 mcarl-Marketer-訂閱 為基礎。
2. 選擇適合您環境的區域（例如、 us-east-1 [N.）、然後選擇驗證方法（例如、承擔角色或 AWS 金鑰）。在

此驗證中、我們使用 AWS 金鑰。

3. 提供連接器的名稱並建立角色。
4. 根據您是否需要公有 IP 、提供 VPC 、子網路或金鑰組等網路詳細資料。
5. 提供安全性群組的詳細資料、例如從來源類型存取 HTTP 、 HTTPS 或 SSH 、例如 Anywhere 和 IP 範圍資訊。
6. 檢閱並建立 BlueXP Connector 。
7. 確認 BlueXP EC2 執行個體狀態在 AWS 主控台中執行、然後從 * 網路 * 索引標籤檢查 IP 位址。
8. 從 BlueXP 入口網站登入 Connector 使用者介面、或使用 IP 位址從瀏覽器存取。

NetApp Cloud Volumes ONTAP

若要在 BlueXP 中建立 Cloud Volumes ONTAP 執行個體、請完成下列步驟：

1. 建立新的工作環境、選取雲端供應商、然後選取 Cloud Volumes ONTAP 執行個體類型（例如單一 CVO 、 HA 或 Amazon FSxN for ONTAP ）。
2. 提供 Cloud Volumes ONTAP 叢集名稱和認證等詳細資料。在此驗證中、我們建立了一個名為的 Cloud Volumes ONTAP 執行個體 `svm_sagemaker_cvo_sn1`。
3. 選取 Cloud Volumes ONTAP 所需的服務。在此驗證中、我們選擇僅監控、因此我們停用了 * 資料感知與法規遵循 * 和 * 備份至雲端服務 * 。
4. 在 * 位置與連線 * 區段中、選取 AWS 區域、 VPC 、子網路、安全性群組、 SSH 驗證方法、 以及密碼或金鑰配對。
5. 選擇充電方式。我們使用 * Professional* 進行此驗證。
6. 您可以選擇預先設定的套件、例如 * POC 和小型工作負載 * 、 * 資料庫和應用程式資料生產工作負載 * 、 * 具成本效益的 DR* 或 * 最高效能的正式作業工作負載 * 。在此驗證中、我們選擇 * POC 和小型工作負載 * 。
7. 建立具有特定大小、允許的通訊協定和匯出選項的 Volume 。在此驗證中、我們建立了一個名為的 Volume `vol1`。
8. 選擇設定檔磁碟類型和分層原則。在此驗證中、我們停用了 * 儲存效率 * 和 * 通用 SSD ：動態效能 * 。
9. 最後、檢閱並建立 Cloud Volumes ONTAP 執行個體。然後等待 15 到 20 分鐘、讓 BlueXP 建立 Cloud Volumes ONTAP 工作環境。
10. 設定下列參數以啟用二元傳輸協定。ONTAP 9 支援二元傳輸協定（ NFS/S3 ）。12.1 及更新版本。
    a. 在此驗證中、我們建立了一個稱為的 SVM `svm_sagemaker_cvo_sn1` 和Volume `vol1`。
    b. 驗證 SVM 是否支援 NFS 和 S3 的傳輸協定。如果沒有、請修改 SVM 以支援它們。

```
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. 必要時建立並安裝 CA 憑證。
12. 建立服務資料原則。

```
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. 檢查 Aggregate 詳細資料。

```
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. 建立使用者和群組。

```
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::*>
```

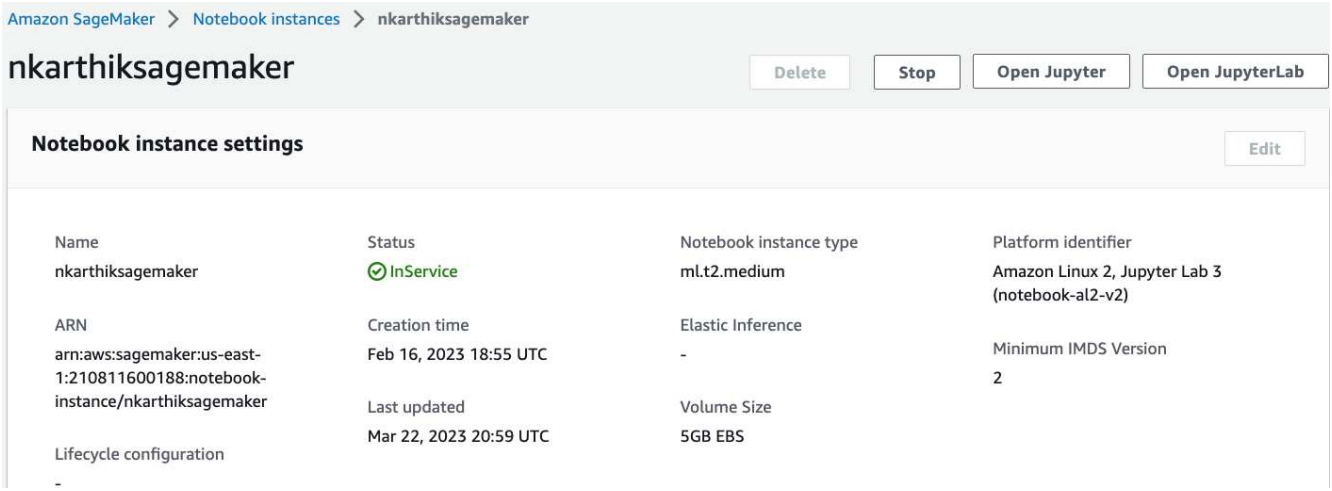
15. 在 NFS 磁碟區上建立貯體。

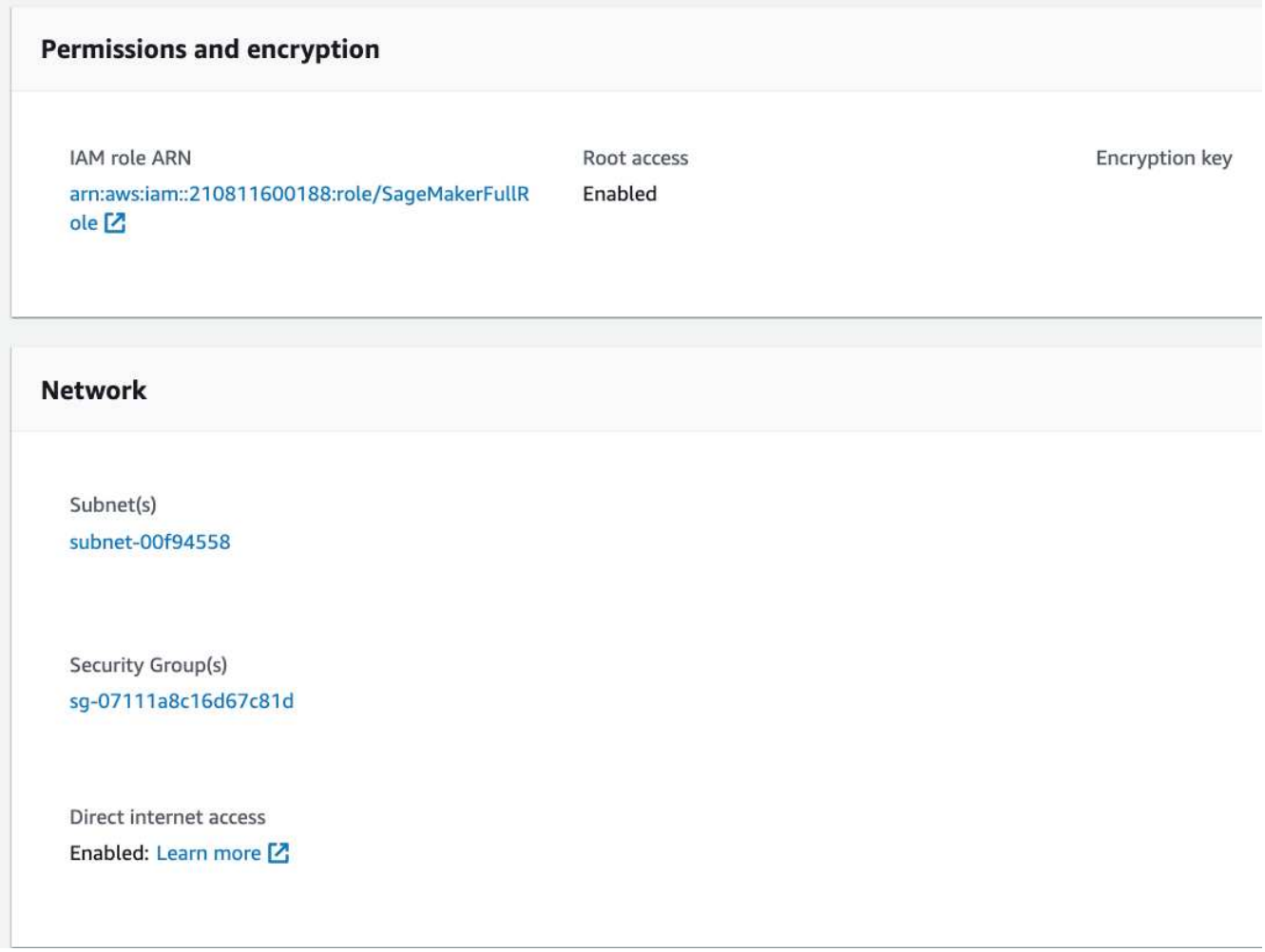
```
sagemaker_cvo_sn1::*> vserver object-store-server bucket create -bucket
ontapbucket1 -type nas -comment "" -vserver svm_sagemaker_cvo_sn1 -nas
-path /vol1
sagemaker_cvo_sn1::*> vserver object-store-server bucket show
Vserver     Bucket          Type     Volume            Size
Encryption Role    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

AWS SageMaker

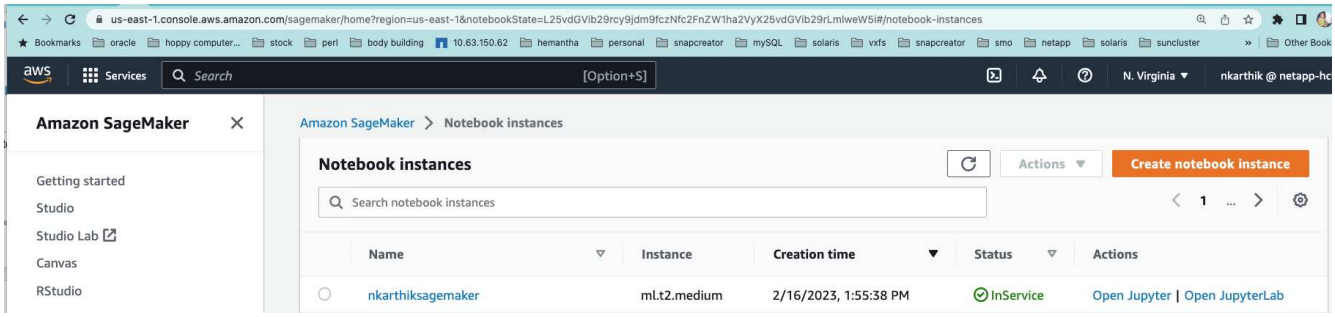
若要從 AWS SageMaker 建立 AWS 筆記型電腦、請完成下列步驟：

1. 請確定正在建立 Notebook 執行個體的使用者擁有 amzonSageMakerFullAccess IAM 原則、或是現有群組的一部分、該群組擁有 amzonSageMakerFullAccess 權限。在此驗證中、使用者是現有群組的一部分。
2. 提供下列資訊：
   - 筆記本執行個體名稱。
   - 執行個體類型。
   - 平台識別碼。
   - 選取具有 amaronSageMakerFullAccess 權限的 IAM 角色。
   - root 存取權–啟用。
   - 加密金鑰 - 選取「無自訂加密」。
   - 保留其餘的預設選項。
3. 在此驗證中、 SageMaker 執行個體詳細資料如下：

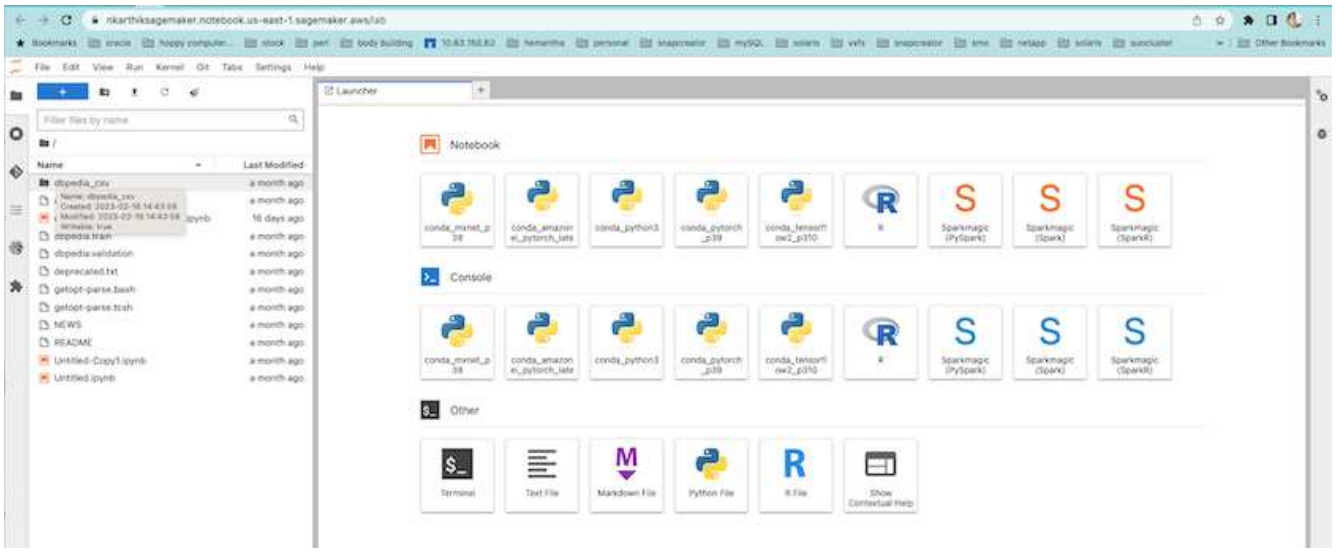




4. 啟動 AWS 筆記型電腦。



5. 開啟 Jupyter 實驗室。



6. 登入終端機並掛載 Cloud Volumes ONTAP Volume 。

```
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. 使用 AWS CLI 命令檢查在 Cloud Volumes ONTAP 磁碟區上建立的貯體。

```
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$
```

用於機器學習的資料

在這項驗證中、我們使用來自 DBexpedia 的資料集、這是一項來自群眾的社群努力、從各種 Wikimedia 專案所建立的資訊中擷取結構化內容。

1. 從 DBexpedia GitHub 位置下載資料並將其解壓縮。請使用上一節所使用的相同終端機。

```
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. 將資料複製到 Cloud Volumes ONTAP 位置、然後使用 AWS CLI 從 S3 儲存區檢查資料。

```
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. 執行基本驗證、確保 S3 儲存區的讀取 / 寫入功能正常運作。

```
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$
```

驗證 **Jupyter** 筆記型電腦的機器學習

下列驗證功能可透過以下 SageMaker BlazingText 範例、透過文字分類提供機器學習建置、訓練及部署模型：

1. 安裝 boto3 和 SageMaker 套件。

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

輸出：

```
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. 在下列步驟中、資料 (`dbpedia_csv`）從 S3 儲存區下載 `ontapbucket1` 至用於機器學習的 Jupyter Notebook 執行個體。

```
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. 下列程式碼會建立從整數索引到類別標籤的對應、以便在推斷期間擷取實際類別名稱。

```
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()
```

輸出會列出中的檔案和資料夾 `ontapbucket1` 做為 AWS SageMaker 機器學習驗證資料的貯體。

```
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. 開始資料預先處理階段、將訓練資料預先處理成空間分隔、可由 BlazingText 演算法和 nltk 程式庫使用的權證化文字格式、以使 DBPedia 資料集的輸入句子變成權證。下載 nltk tokenizer 和其他程式庫。。`transform_instance` 平行套用至每個資料執行個體使用 Python 多重處理模組。

```
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)
```

輸出：

```
[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. 將格式化和訓練資料集上傳至 S3 、讓 SageMaker 可以使用該資料集來執行訓練工作。然後使用 Python SDK 將兩個檔案上傳至貯體和前置碼位置。

```
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)
```

輸出：

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

6. 在 S3 上設定輸出位置、將模型成品載入其中、使成品能成為演算法訓練工作的輸出。建立 `sageMaker.estimator.Estimator` 物件以啟動訓練工作。

```
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))
```

輸出：

```
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. 定義 SageMaker `Estrimator` 使用資源組態和超參數、在 c4.4xlarge 執行個體上使用受監督模式、在 DBPedia 資料集上訓練文字分類。

```
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. 準備資料通道與演算法之間的交握。若要這麼做、請建立 `sagemaker.session.s3_input` 來自資料通道的物件、並將其保留在字典中、以供演算法使用。

```
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. 工作完成後、會出現「工作完成」訊息。您可以在設定為的 S3 儲存貯體中找到經過訓練的機型 `output_path` 在評估者中。

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

輸出：

```
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. 訓練完成後、請將經過訓練的模型部署為 Amazon SageMaker 即時代管端點、以做出預測。

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

輸出：

```
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. 根據預設，模型會傳回一個機率最高的預測值。以擷取頂端 `k` 預測、設定 `k` 在組態檔案中。

```
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. 在關閉筆記本之前刪除端點。

```
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
```

結論

根據這項驗證、資料科學家和工程師可透過 NetApp Cloud Volumes ONTAP 的 S3 儲存區、從 AWS SageMaker Jupyter 筆記型電腦存取 NFS 資料。這種方法可讓您輕鬆從 NFS 和 S3 存取和共用相同的資料、而不需要額外的軟體。

何處可找到其他資訊

若要深入瞭解本文所述資訊、請檢閱下列文件和 / 或網站：

- 使用 SageMaker BlazingText 進行文字分類

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

- 支援S3物件儲存的版本ONTAP

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

Apache Kafka工作負載搭配NetApp NFS儲存設備

TR-4947：Apache Kafka工作負載搭配NetApp NFS儲存設備-功能驗證與效能

Shantanu Chakole, Karthikeyan Nagalingam,以及NetApp公司Joe Scott

Kafka是一種分散式發佈訂閱訊息系統、具有強大的佇列、可接受大量的訊息資料。有了Kafka、應用程式就能以非常快速的方式將資料寫入和讀取至主題。由於其容錯能力與擴充性、因此Kafka經常被用在大資料空間、作為快速擷取和移動許多資料串流的可靠方法。使用案例包括串流處理、網站活動追蹤、指標收集與監控、記錄彙總、即時分析等。

雖然正常的Kafka NFS作業運作良好 "不太好重命名" 在NFS上執行的Kafka叢集調整大小或重新分割時、問題會使應用程式當機。這是一個重大問題、因為Kafka叢集必須調整大小或重新分割、才能進行負載平衡或維護。您可以找到其他詳細資料 "請按這裡"。

本文件說明下列主題：

- 這是個不太好重命名的問題、也是解決方案驗證的關鍵
- 降低CPU使用率以縮短I/O等待時間
- 更快的Kafka代理程式恢復時間
- 雲端和內部部署的效能

為什麼要將**NFS**儲存設備用於**Kafka**工作負載？

正式作業應用程式中的Kafka工作負載可在應用程式之間串流大量資料。此資料會保留並儲存在Kafka叢集中的Kafka Broker節點中。Kafka也以可用度和平行度而聞名、將主題分成分割區、然後在整個叢集內複寫這些分割區、即可達成此目標。這最終意味著流經Kafka叢集的大量資料通常會增加大小。NFS可在代理程式數量改變時、快速又輕鬆地重新平衡資料。在大型環境中、當代理商數量的變動非常耗時時、會在DAS之間重新平衡資料、而在大多數的Kafka環境中、代理商數量也會經常變動。

其他效益包括：

- 成熟度 NFS是一種成熟的傳輸協定、這表示實作、保護及使用它的大部分層面都已獲得充分的瞭解。
- 開放式 NFS是一種開放式傳輸協定、其持續開發作業記錄於網際網路規格中、以免費開放式網路傳輸協定的形式提供。
- 具成本效益的. NFS是一款低成本的網路檔案共用解決方案、因為它使用現有的網路基礎架構、所以很容易設定。
- 集中管理。 NFS的集中管理可減少個別使用者系統上新增軟體和磁碟空間的需求。
- 分散式 NFS可作為分散式檔案系統、減少對卸除式媒體儲存設備的需求。

為何選擇**NetApp**來處理**Kafka**工作負載？

NetApp NFS實作被視為傳輸協定的黃金標準、可用於無數的企業NAS環境。除了NetApp的可信度之外、它還提供下列優點：

- 可靠性與效率
- 擴充性與效能
- 高可用度（HA合作夥伴在NetApp ONTAP 供應叢集中）
- 資料保護
  - *災難恢復（NetApp SnapMirror）。*您的站台當機、或您想要從不同的站台開始、然後從您離開的地方繼續進行。
  - 儲存系統的管理能力（使用NetApp OnCommand 功能進行管理）。
  - *負載平衡。*叢集可讓您從位於不同節點上的資料LIF存取不同的磁碟區。
  - 不中斷營運。 lifs或Volume移動對NFS用戶端而言是透明的。

NetApp解決方案可解決**NFS**對**Kafka**工作負載的不合理重新命名問題

Kafka的建置假設基礎檔案系統符合POSIX標準：例如XFS或ext4。在應用程式仍在使用檔案時、Kafka資源重新平衡功能會移除檔案。符合POSIX標準的檔案系統可讓您繼續解除連結。不過、只有在所有檔案參考資料都消失之後、它才會移除檔案。如果基礎檔案系統是網路附加的、NFS用戶端會攔截取消連結的呼叫、並管理工作流程。由於在要取消連結的檔案上有擱置的開啟、NFS用戶端會將重新命名要求傳送至NFS伺服器、並在上次關閉未連結檔案時、對重新命名的檔案執行移除作業。這種行為通常稱為NFS ruted rame、由NFS用戶端協調。

任何使用NFSv3伺服器儲存設備的Kafka代理商、都會因為這種行為而發生問題。不過、NFSv4.x傳輸協定具備多項功能、可讓伺服器負責開啟且未連結的檔案、藉此解決此問題。支援此選用功能的NFS伺服器、會在檔案開啟時將擁有權功能傳達給NFS用戶端。NFS用戶端會在開啟擱置中時停止取消連結管理、並允許伺服器管理流程。雖然NFSv4規格提供實作準則、但直到現在為止、並沒有任何已知的NFS伺服器實作支援此選用功能。

NFS伺服器和NFS用戶端需要進行下列變更、才能解決這個「愚蠢」的重新命名問題：

- * NFS用戶端（Linux）的變更。*檔案開啟時、NFS伺服器會以旗標回應、表示能夠處理已開啟檔案的解除連結。NFS用戶端變更可讓NFS伺服器在旗標出現時處理取消連結。NetApp已針對這些變更更新開放原始碼Linux NFS用戶端。更新後的NFS用戶端現在一般可在RHEL8.7和RHEL9.1中使用。
- * NFS伺服器的變更。* NFS伺服器會持續追蹤開啟狀況。現在、伺服器會管理現有開放式檔案上的取消連結、以符合POSIX語義。關閉最後一個開啟的檔案時、NFS伺服器會啟動檔案的實際移除、進而避免執行不必要的重新命名程序。此功能已在最新版的《S59.12》中實作ONTAP ONTAP 。

上述NFS用戶端和伺服器的變更、讓Kafka能夠安全地享有網路附加NFS儲存設備的所有優點。

功能驗證-不太好用的重新命名修正程式

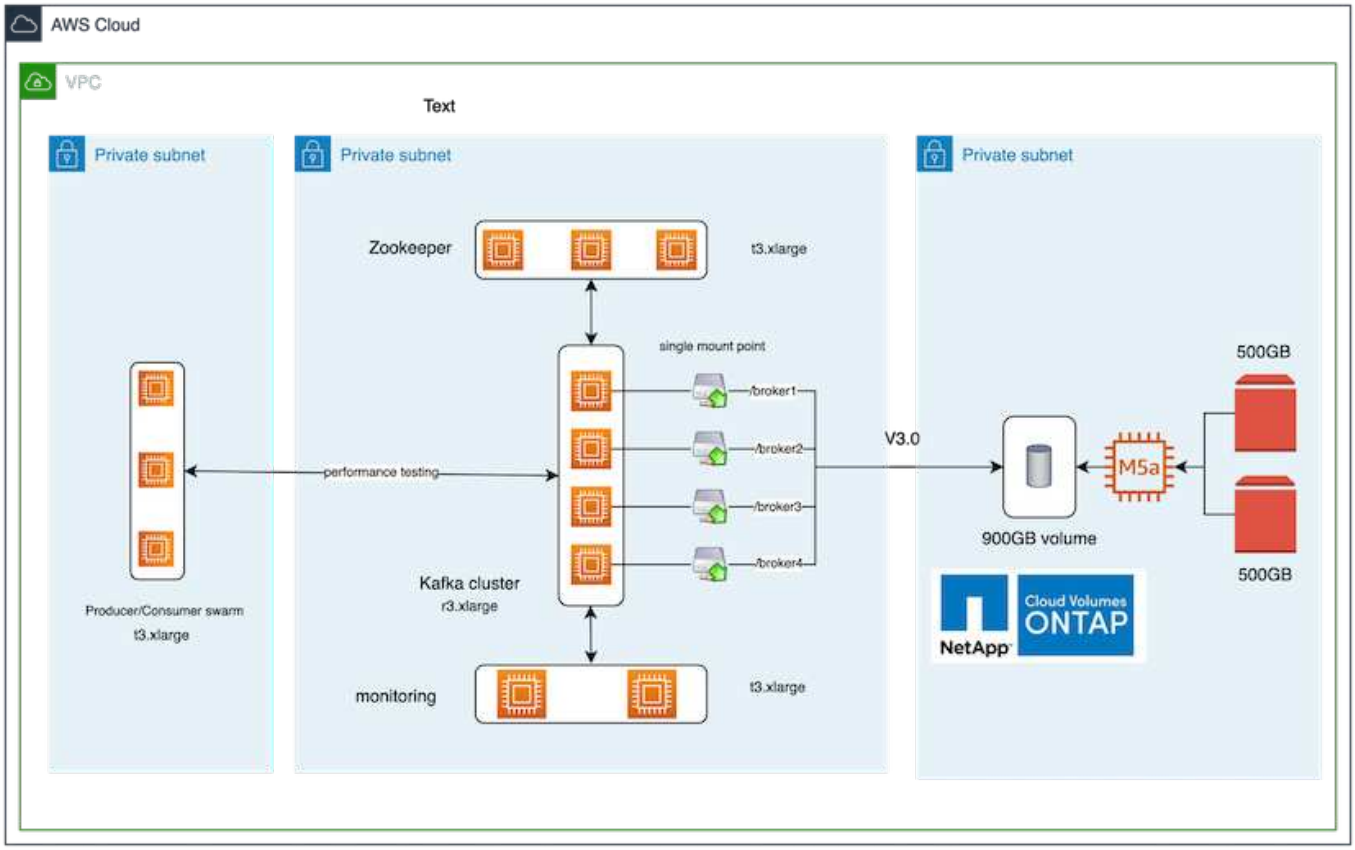
在功能驗證方面、我們發現裝有NFSv3儲存設備掛載的Kafka叢集無法執行像分割區重新分佈等Kafka作業、而裝在NFSv3上的另一個叢集搭配修正程式、則可執行相同的作業、而不會造成任何中斷。

驗證設定

設定會在AWS上執行。下表顯示驗證所使用的不同平台元件和環境組態。

平台元件	環境組態
ConFluent Platform 7.2.1版	• 3個zookeepers–T3.xlarge • 4個代理伺服器–R3.xlarge • 1 x Grafana–T3.xlarge • 1個控制中心–T3.xLarge • 3個製造商/消費者
所有節點上的作業系統	RHEL8.7或更新版本
NetApp Cloud Volumes ONTAP 執行個體	單節點執行個體–M5.2xLarge

下圖顯示此解決方案的架構組態。



架構流程

- *運算*我們使用四節點的Kafka叢集、其中三節點的zookeeper集合體執行於專屬伺服器上。
- *監控*我們使用兩個節點來搭配Prometheus-Grafana。
- 工作負載。*為了產生工作負載、我們使用了一個獨立的三節點叢集、可產生和使用此Kafka叢集。

- * Storage。*我們使用單節點NetApp Cloud Volumes ONTAP 支援執行個體、並將兩個500GB GP2 AWS-EBS磁碟區附加至執行個體。然後、這些磁碟區會透過LIF以單一NFSv4.1磁碟區的形式、公開給Kafka叢集。

所有伺服器都會選擇Kafka的預設屬性。對zookeeper swarm也是如此。

測試方法

1. 更新 `-is-preserve-unlink-enabled true` 至Kafka磁碟區、如下所示：

```
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. 建立了兩個類似的Kafka叢集、其差異如下：
   - *叢集1.*執行正式作業功能ONTAP 的後端NFS v4.1伺服器9.12.1版是由NetApp CVO執行個體代管。已在代理商上安裝RHEL 8.7/RHEL 9.1。
   - *叢集2.*後端NFS伺服器是手動建立的通用Linux NFSv3伺服器。
3. 兩個Kafka叢集都建立了示範主題。

   叢集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:
```

   叢集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. 資料已載入這兩個叢集的新建立主題。這是使用預設Kafka套件隨附的生產商perf-test工具組來完成：

```
./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. 已使用Telnet:對每個叢集的Broker 1執行健全狀況檢查。
   - 遠端登入 `172.30.0.160 9092`
   - 遠端登入 `172.30.0.198 9092`

     下一個快照會顯示兩個叢集上的代理程式健全狀況檢查是否成功：

```
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. 為了觸發故障情況、導致使用NFSv3儲存磁碟區的Kafka叢集當機、我們在兩個叢集上都開始重新指派磁碟分割。磁碟分割重新指派是使用執行 `kafka-reassign-partitions.sh`。詳細程序如下：
   a. 若要重新指派Kafka叢集中某個主題的分割區、我們會產生建議的重新指派組態Json（這是針對兩個叢集執行）。

```
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. 然後儲存產生的重新指派Json `/tmp/reassignment- file.json`。
   c. 實際的分割區重新指派程序是由下列命令觸發：

```
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. 重新指派完成幾分鐘後、代理商的另一項健全狀況檢查顯示、使用NFSv3儲存磁碟區的叢集發生錯誤的重新命名問題、並已當機、而使用NetApp ONTAP NFSv4.1儲存磁碟區的叢集1則可在不中斷營運的情況下繼續進行修正作業。

```
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
```

- 叢集1-Broler-1處於作用中狀態。
- Cluster2-Broker 1已死機。

8. 查看Kafka記錄目錄後、很明顯、使用NetApp ONTAP 支援NFSv3的叢集1儲存磁碟區搭配修復程式時、會有乾淨的分割區指派、而使用一般NFSv3儲存設備的叢集2則不會因為錯誤的重新命名問題而導致當機。下圖顯示叢集2的分割區重新平衡、導致NFSv3儲存設備發生不實的重新命名問題。

```
/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
```

下圖顯示使用NetApp NFSv4.1儲存設備重新平衡叢集1的乾淨分割區。

```
/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
```

為何選擇適用於**Kafka**工作負載的**NetApp NFS**？

既然有解決方法可解決使用Kafka進行NFS儲存時的不實重命名問題、您就能建立健全的部署、讓NetApp ONTAP 的支援資源能夠滿足您的Kafka工作負載。這不僅能大幅降低營運成本、還能為您的Kafka叢集帶來下列效益：

- *降低Kafka代理商的CPU使用率。*使用分離式NetApp ONTAP 解決方案、將磁碟I/O作業與代理商分開、進而減少其CPU佔用空間。
- *更快的代理程式還原時間。*由於在ONTAP Kafka代理節點之間共享分離式NetApp支援、因此新的運算執行個體可在短時間內取代不良的代理程式、而無需重建資料。
- *儲存效率。*由於應用程式的儲存層現在是透過NetApp ONTAP 資源中心進行配置、因此客戶可以利用ONTAP 隨附於此功能的所有儲存效率優勢、例如即時資料壓縮、重複資料刪除和資料壓縮。

這些效益已在測試案例中經過測試和驗證、我們將在本節中詳細討論。

降低**Kafka**代理程式的**CPU**使用率

我們發現、當我們在兩個sperate Kafka叢集上執行類似工作負載時、整體CPU使用率比DAS低、這些叢集在技術規格上完全相同、但儲存技術卻不同。不僅是卡夫卡叢集使用ONTAP 率較低的整體CPU使用率、而且CPU使用率的增加、顯示出比DAS型卡夫卡叢集更為溫和的漸層。

架構設定

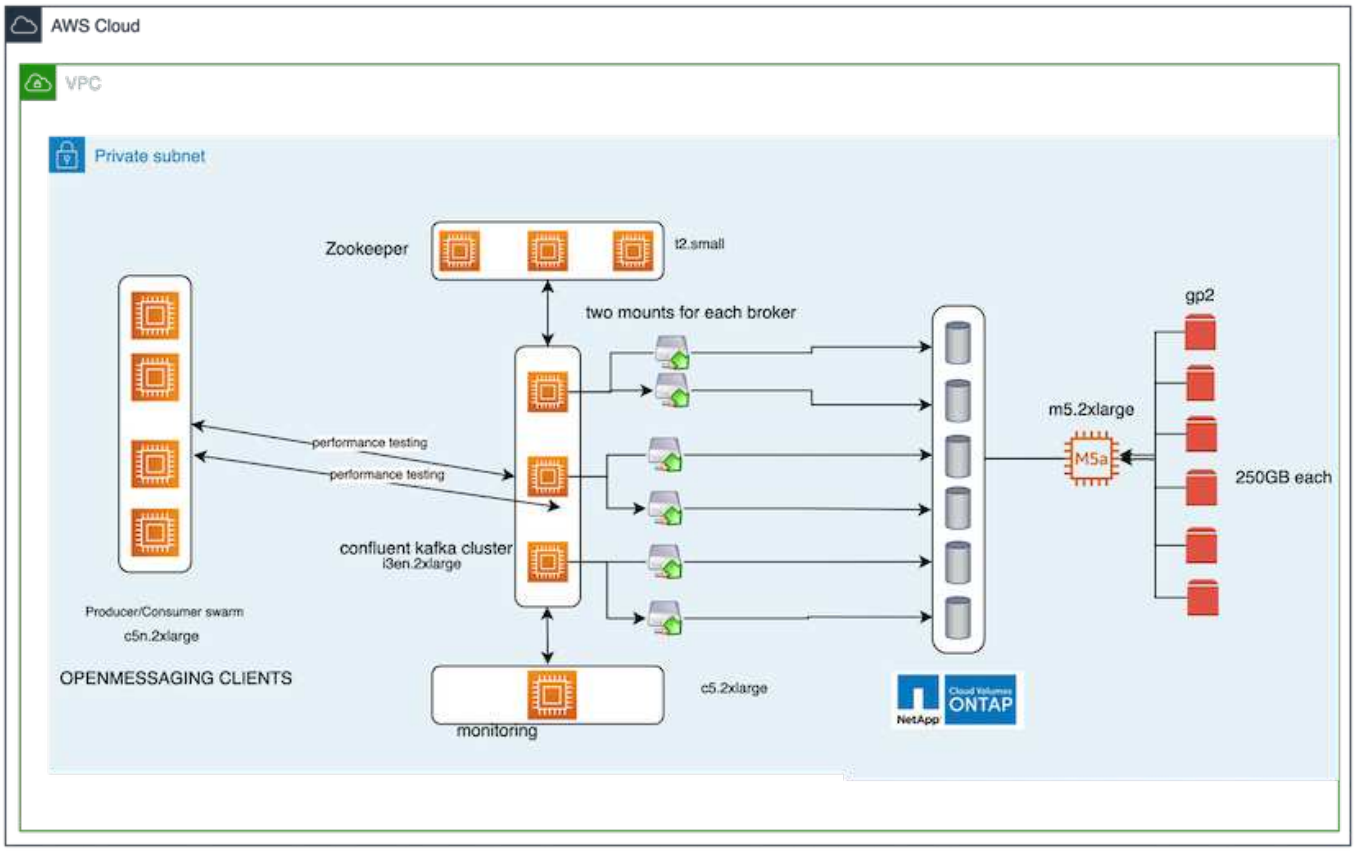
下表顯示用於示範降低CPU使用率的環境組態。

平台元件	環境組態
Kafka 3.2.3基準測試工具：OpenMessaging	• 3個zookeepers–T2.small • 3個代理伺服器–i3en.2xLarge • 1 x Grafana–c5n.2xLarge • 4個製造商/消費者- c5n.2xLarge

平台元件	環境組態
所有節點上的作業系統	RHEL 8.7或更新版本
NetApp Cloud Volumes ONTAP 執行個體	單節點執行個體–M5.2xLarg

基準測試工具

本測試案例中使用的基準測試工具為 "OpenMessaging" 架構。OpenMessaging不受廠商限制、不受語言限制；它提供金融、電子商務、物聯網和巨量資料等產業準則、有助於跨異質系統和平台開發訊息和串流應用程式。下圖說明OpenMessaging用戶端與Kafka叢集之間的互動。



- *運算*我們使用三節點的Kafka叢集、其中三節點的zookeeper集合體執行於專屬的伺服器上。每個代理商都有兩個NFSv4.1掛載點、透過專屬的LIF連接至NetApp CVO執行個體上的單一Volume。
- *監控*我們使用兩個節點來搭配Prometheus-Grafana。為了產生工作負載、我們有一個獨立的三節點叢集、可產生和使用此Kafka叢集。
- * Storage。*我們使用單節點NetApp Cloud Volumes ONTAP 支援執行個體、並在執行個體上安裝六個250GB GP2 AWS-EBS磁碟區。然後、這些磁碟區會透過專用的LIF、以六個NFSv4.1磁碟區的形式呈現給Kafka叢集。
- *組態。*此測試案例中的兩個可設定元素為Kafka Brokers和OpenMessaging工作負載。
  - * Broker config.*為Kafka經紀人選擇了下列規格。我們在所有測量項目中都使用3個複寫係數、如下所示。

```
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基準測試（OMB）工作負載組態。*提供下列規格。我們指定目標生產商比率、重點如下。

```
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
```

測試方法

1. 建立了兩個類似的叢集、每個叢集都有自己的基準測試叢集。
   - 叢集**1.** NFS型Kafka叢集。
   - 叢集**2.** DAS型Kafka叢集。
2. 使用OpenMessaging命令、會在每個叢集上觸發類似的工作負載。

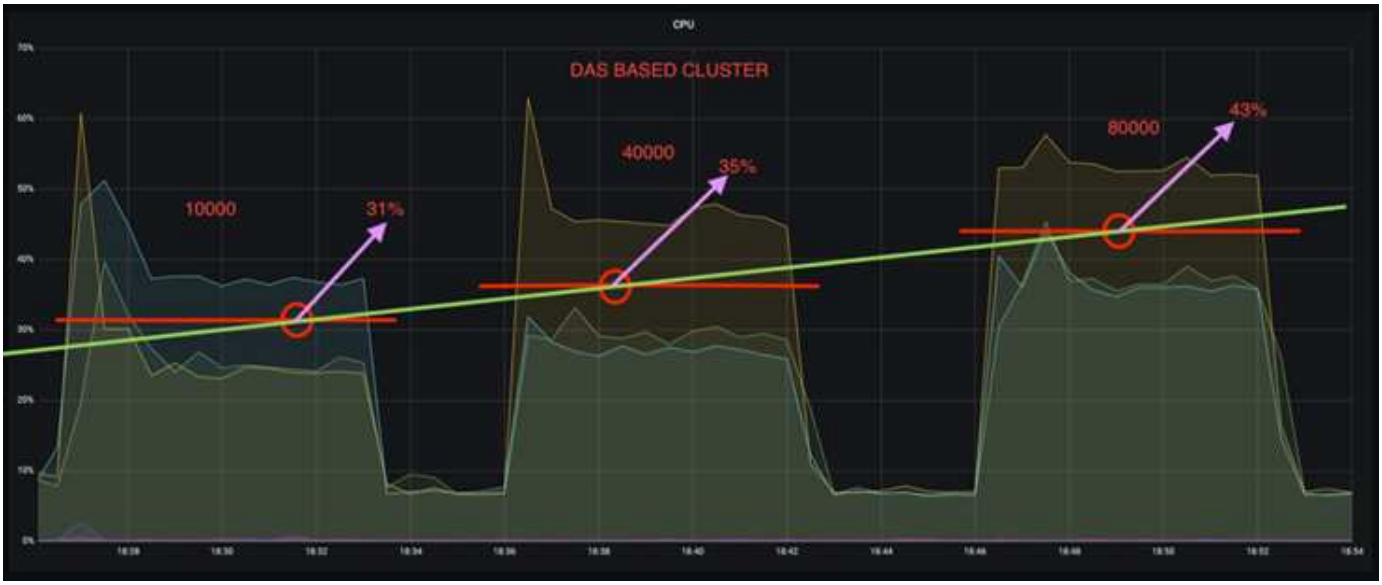
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

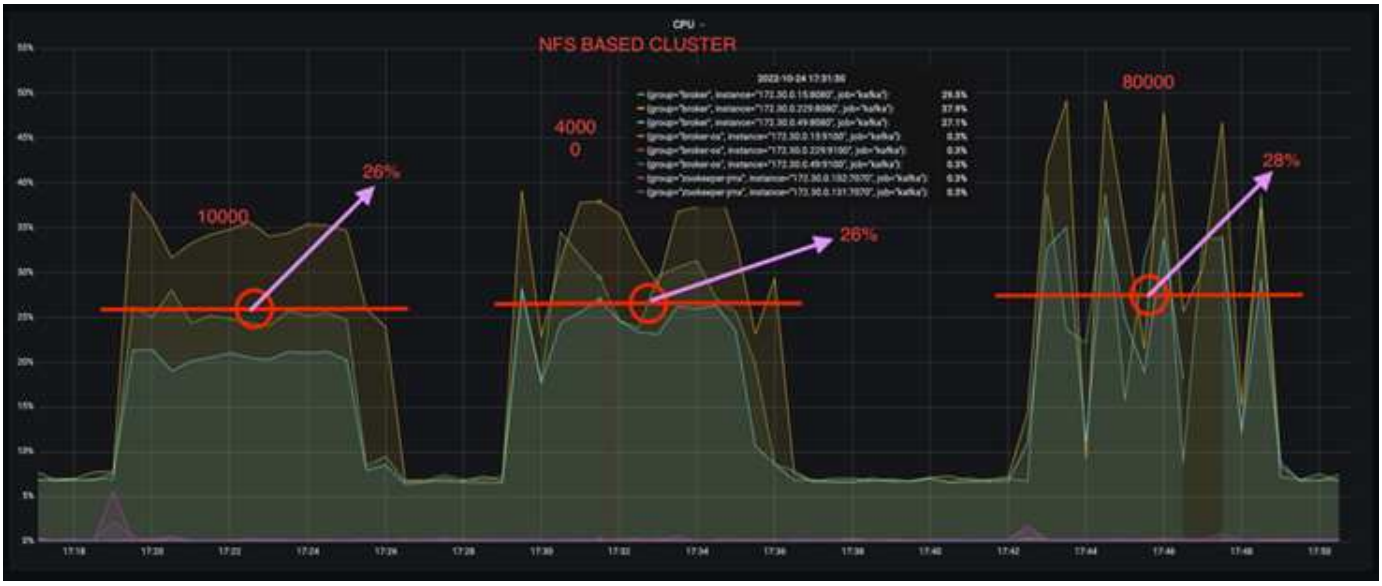
3. 產生率組態在四次迭代中增加、而CPU使用率則記錄在Grafana。產品出價設定為下列層級：
    - 10、000
    - 40、000
    - 80、000
    - 10、000

觀察

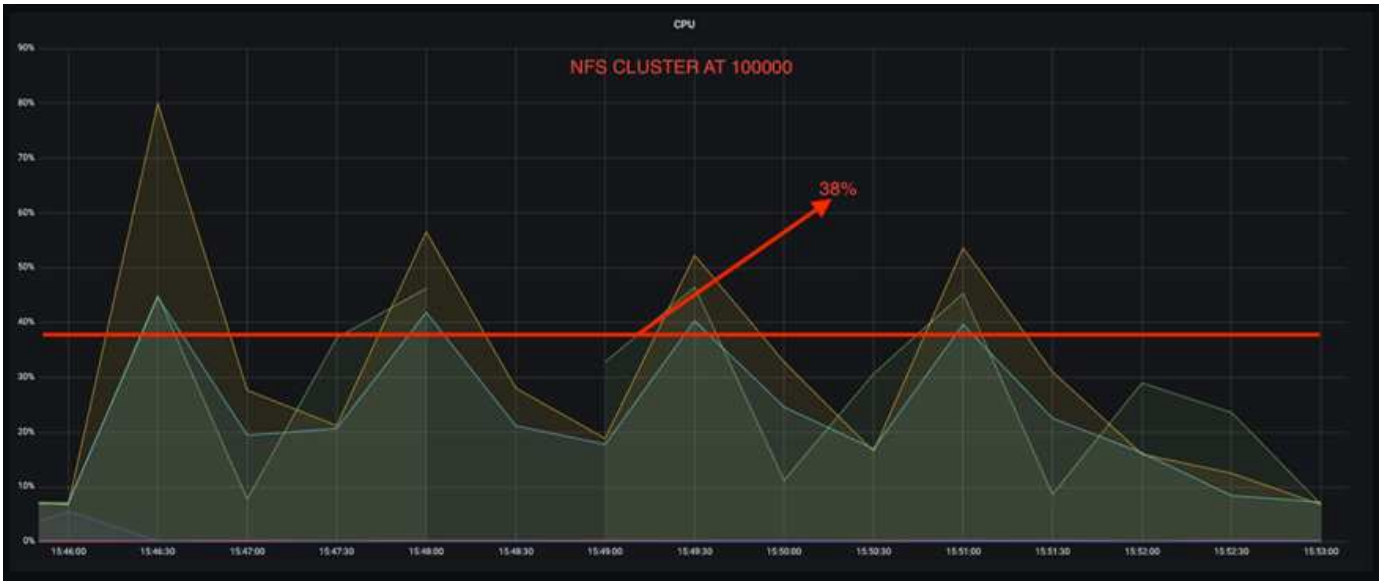
搭配Kafka使用NetApp NFS儲存設備有兩大主要效益：

- *您可以減少將近三分之一的CPU使用率。*相較於DAS SSD、NFS在類似工作負載下的整體CPU使用率較低；較低的產品使用率可節省5%、較高的產品使用率則可節省32%。
- 以較高的產生率、**CPU**使用率會減少三倍。*如預期、隨著生產率增加、CPU使用率會增加、因此CPU使用率會增加。不過、使用DAS的Kafka代理商的CPU使用率從較低的產品使用率的31 %增加到較高的產品使用率的70 %、增加39%。不過、有了NFS儲存後端、CPU使用率從26%增加到38%、增加12%。





此外、在100、000則訊息中、DAS顯示的CPU使用率比NFS叢集高。



更快速的代理程式還原

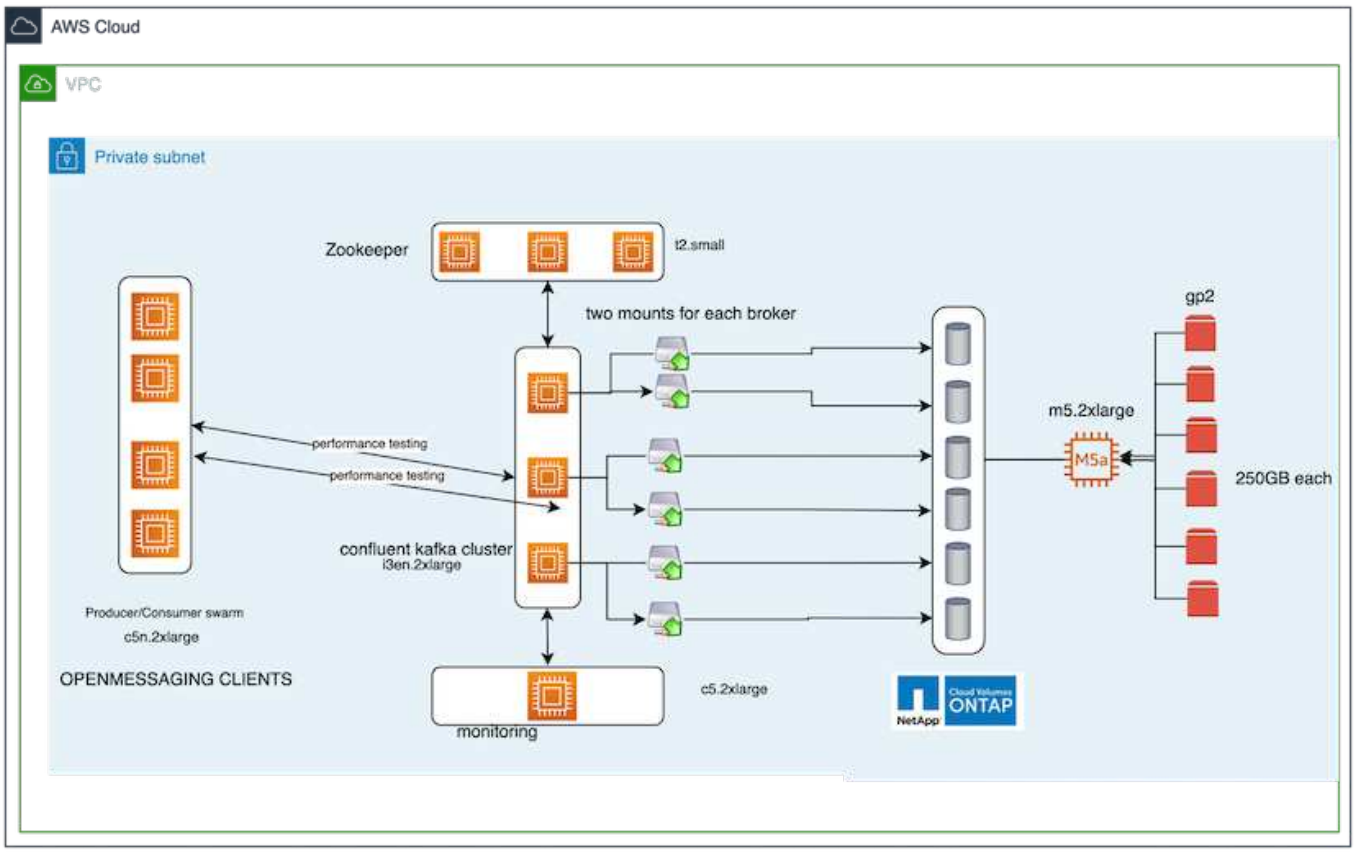
我們發現、Kafka代理商使用共享NetApp NFS儲存設備時、恢復速度更快。當Kafka叢集中的代理程式當機時、此代理程式可由具有相同代理程式ID的健全代理程式取代。在執行此測試案例時、我們發現在以DAS為基礎的Kafka叢集上、叢集會在新增的健全代理程式上重新建置資料、這相當耗時。在NetApp NFS型Kafka叢集的情況下、更換的代理程式會繼續從先前的記錄目錄讀取資料、並以更快的速度恢復。

架構設定

下表顯示使用NAS的Kafka叢集環境組態。

平台元件	環境組態
Kafka 3.2.3	• 3個zookeepers–T2.small • 3個代理伺服器–i3en.2xLarge • 1 x Grafana–c5n.2xLarge • 4個製造商/消費者- c5n.2xLarge • 1個備份Kafka節點–i3en.2xLarge
所有節點上的作業系統	RHEL8.7或更新版本
NetApp Cloud Volumes ONTAP 執行個體	單節點執行個體–M5.2xLarge

下圖說明NAS型Kafka叢集的架構。



- *運算。*三節點Kafka叢集、在專用伺服器上執行三節點zookeeper集合體。每個代理程式都有兩個NFS掛載點、可透過專屬LIF連接至NetApp CVO執行個體上的單一磁碟區。
- 監控 Prometheus-Grafana組合的兩個節點。為了產生工作負載、我們使用獨立的三節點叢集、可產生並使用此Kafka叢集。
- * Storage。*單節點NetApp Cloud Volumes ONTAP 效能實例、執行個體上安裝六個250GB GP2 AWS-EBS磁碟區。然後、這些磁碟區會透過專屬的LIF、以六個NFS磁碟區的形式呈現給Kafka叢集。
- * Broker組態。*此測試案例中的其中一個可設定元素是Kafka Broker。卡夫卡經紀公司選擇了下列規格。。`replica.lag.time.mx.ms` 設定為高值、因為這會決定從ISR清單中取出特定節點的速度。當您在不良和健全的節點之間切換時，您不希望將該代理ID排除在ISR清單之外。

```
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
```

測試方法

1. 建立了兩個類似的叢集：
   - 以EC2為基礎的匯合叢集。
   - NetApp NFS型的匯合叢集。
2. 建立一個待命的Kafka節點時、其組態與原始Kafka叢集的節點相同。
3. 在每個叢集上、都建立了範例主題、並在每個代理程式上填入約110 GB的資料。
   - *基於EC2的叢集。*會對應一個Kafka Broker資料目錄 `/mnt/data-2` （下圖為叢集1的Broler-1 [left終端機]）。
   - * NetApp NFS型叢集。*在NFS點上掛載Kafka Broker資料目錄 `/mnt/data` （下圖為叢集2的Broler-1 [右對講機]）。

4. 在每個叢集中、Brocher-1都會終止、以觸發失敗的Broker恢復程序。
5. 代理終止後、會將代理IP位址指派為次要IP給待命代理程式。這是必要的、因為Kafka叢集中的代理程式是由下列項目識別：
   - * IP位址。*指派方式是將故障的代理IP重新指派給待命代理程式。
   - * Broker ID。*這是在待命代理程式中設定的 `server.properties`。
6. 指派IP後、便會在待命代理程式上啟動Kafka服務。
7. 一段時間之後、伺服器記錄會被拉出、以檢查在叢集中的替換節點上建置資料所需的時間。

觀察

Kafka代理商的恢復速度快了將近九倍。相較於使用Kafka叢集中的DAS SSD、使用NetApp NFS共享儲存設備

時、恢復故障代理節點所花的時間大幅加快。對於1TB的主題資料、DAS型叢集的恢復時間為48分鐘、而NetApp NFS型Kafka叢集的恢復時間則不到5分鐘。

我們觀察到、以EC2為基礎的叢集花了10分鐘在新的代理節點上重建110GB的資料、而以NFS為基礎的叢集則在3分鐘內完成恢復。我們也在「In（記錄）」中發現、EC2的分割區使用者偏移值為0、而在NFS叢集上、使用者偏移值則是從先前的代理程式中取得。

```
[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型叢集

1. 備份節點於08：55：53、730開始。

```
2  [2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
3  [2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (or
4  [2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
5  [2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.3
6  [2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. 資料重建程序於09：05：24、860結束。處理110 GB的資料大約需要10分鐘。

```
[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型叢集

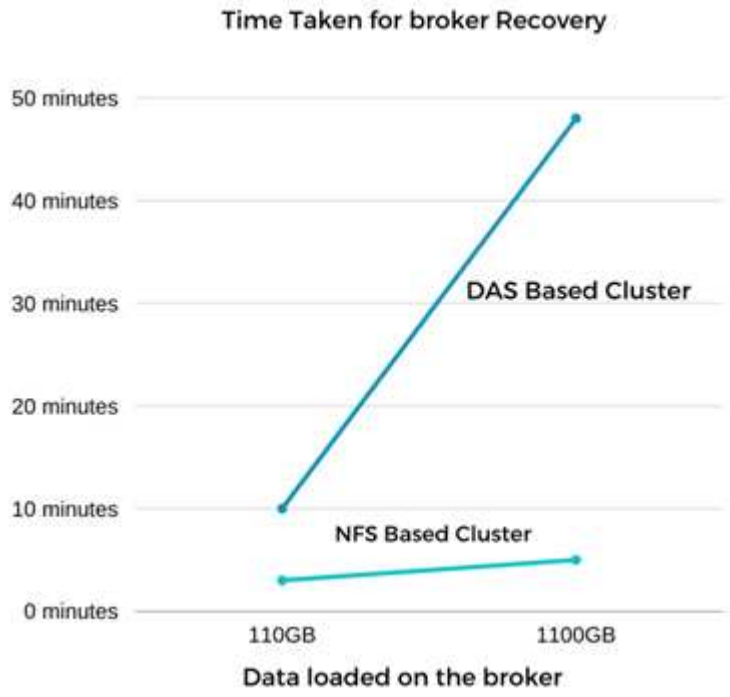
1. 備份節點於09：39：17、213開始。下方會強調顯示開始記錄項目。

```
2  [2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiati
3  [2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.
4  [2022-10-31 09:39:17,213] INFO starting (kafka.server.KafkaServer)
5  [2022-10-31 09:39:17,214] INFO Connecting to zookeeper on 172.30.0.22:2181,172.30.
6  [2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new s
7  [2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a
8  [2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in
9  [2022-10-31 09:39:17,244] INFO Client environment:java.version=11.0.17 (org.apache
```

2. 資料重建程序於09：42：29、115結束。處理110 GB的資料大約需要3分鐘。

```
[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)
```

針對包含約1TB資料的代理商重複測試、DAS約需48分鐘、NFS約需3分鐘。結果如下圖所示。



儲存效率

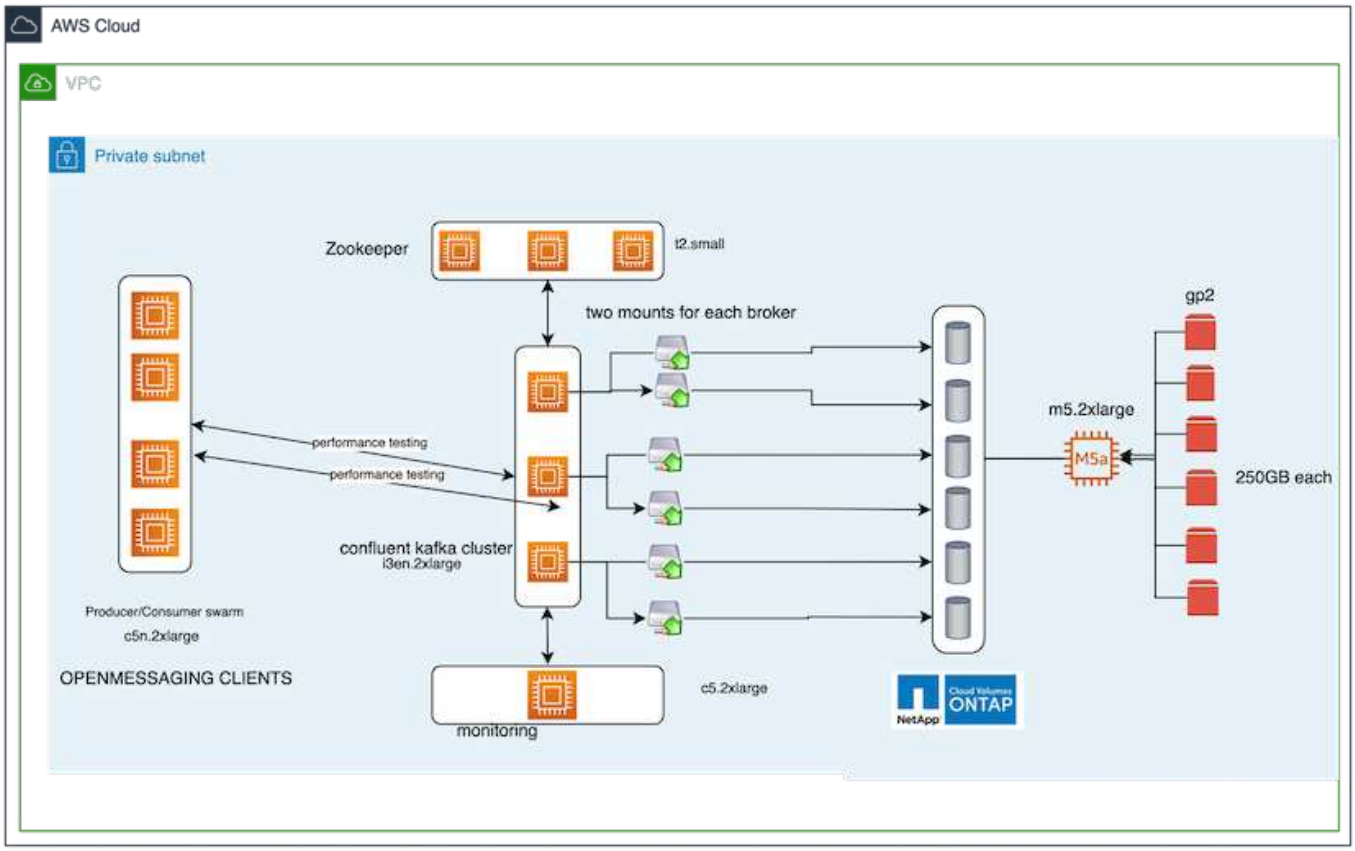
由於Kafka叢集的儲存層是透過NetApp ONTAP 供應、因此我們獲得ONTAP 了所有的NetApp儲存效率功能。測試結果是在安裝Cloud Volumes ONTAP 了NFS儲存設備的Kafka叢集上產生大量資料。我們可以看到ONTAP 、由於採用了一些功能、空間大幅縮減。

架構設定

下表顯示使用NAS的Kafka叢集環境組態。

平台元件	環境組態
Kafka 3.2.3	• 3個zookeepers–T2.small • 3個代理伺服器–i3en.2xLarge • 1 x Grafana–c5n.2xLarge • 4個製造商/消費者- c5n.2xLarge *
所有節點上的作業系統	RHEL8.7或更新版本
NetApp Cloud Volumes ONTAP 執行個體	單節點執行個體–M5.2xLarge

下圖說明NAS型Kafka叢集的架構。



- *運算*我們使用三節點的Kafka叢集、其中三節點的zookeeper集合體執行於專屬的伺服器上。每個代理商都有兩個NFS掛載點、可透過專屬LIF連接至NetApp CVO執行個體上的單一磁碟區。
- *監控*我們使用兩個節點來搭配Prometheus-Grafana。為了產生工作負載、我們使用了一個獨立的三節點叢集、可以產生和使用這個Kafka叢集。
- * Storage。*我們使用單節點NetApp Cloud Volumes ONTAP 支援執行個體、並在執行個體上安裝六個250GB GP2 AWS-EBS磁碟區。然後、這些磁碟區會透過專用的LIF、以六個NFS磁碟區的形式呈現給Kafka叢集。
- *組態。*此測試案例中的可設定元素是Kafka仲介。

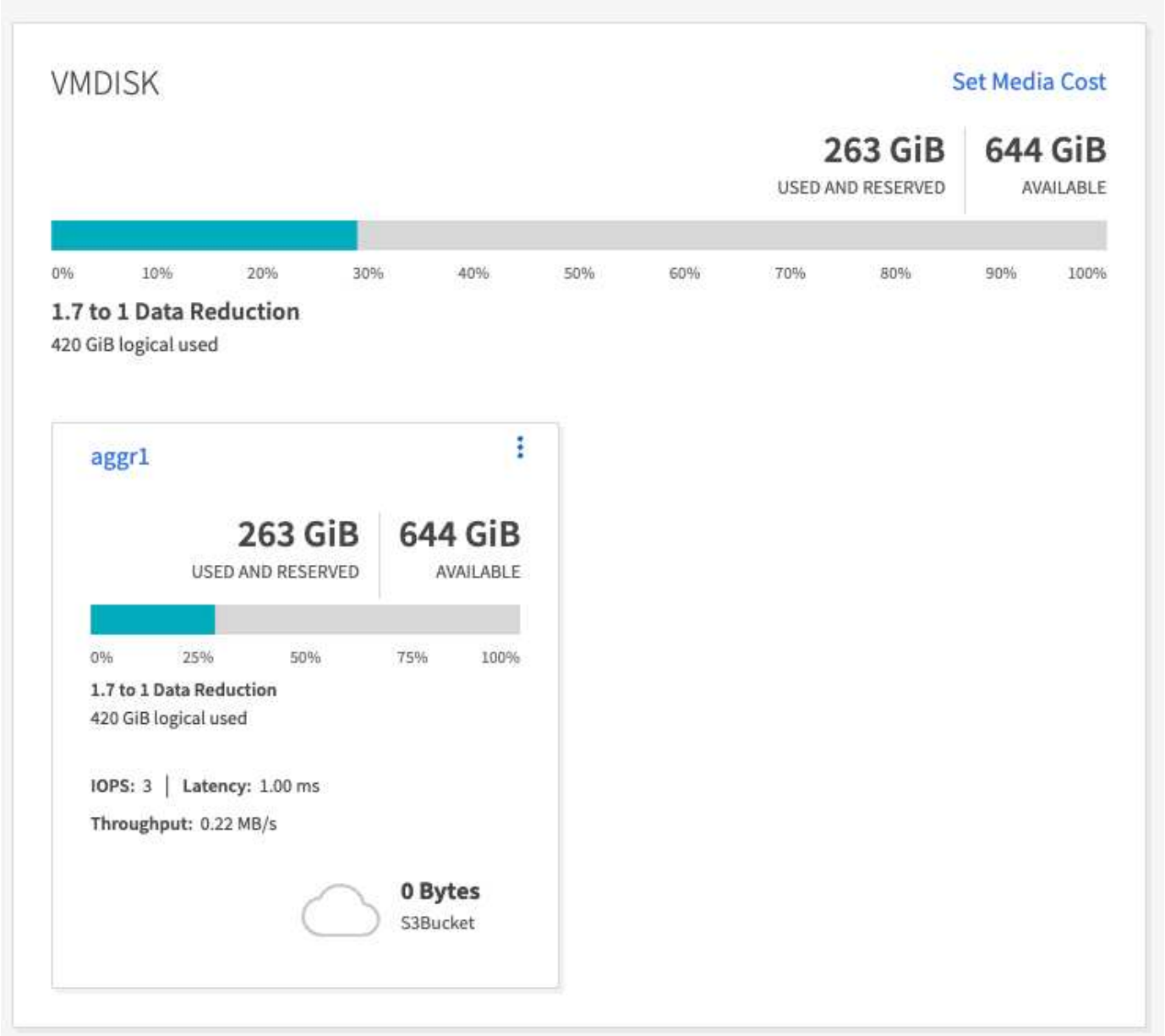
在生產商端點關閉壓縮功能、讓生產商產生高處理量。儲存效率則由運算層來處理。

測試方法

1. 卡夫卡叢集已配置上述規格。
2. 在叢集上、使用OpenMessaging基準測試工具產生約350 GB的資料。
3. 工作負載完成後、會使用ONTAP NetApp System Manager和CLI收集儲存效率統計資料。

觀察

對於使用OMB工具產生的資料、我們發現空間節約約33%、儲存效率比為1.70:1。如下圖所示、所產生資料所使用的邏輯空間為420.3GB、用於保存資料的實體空間為281.7GB。




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

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

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

AWS的效能總覽與驗證

安裝在NetApp NFS上的儲存層Kafka叢集、是以AWS雲端效能為基準。基準測試範例將於下列各節中說明。

採用**NetApp Cloud Volumes ONTAP** 技術的**AWS**雲端中的**Kafka**（高可用度配對與單一節點）

採用NetApp Cloud Volumes ONTAP 功能的Kafka叢集（HA配對）是AWS雲端效能的基準測試。以下各節將說明此基準測試。

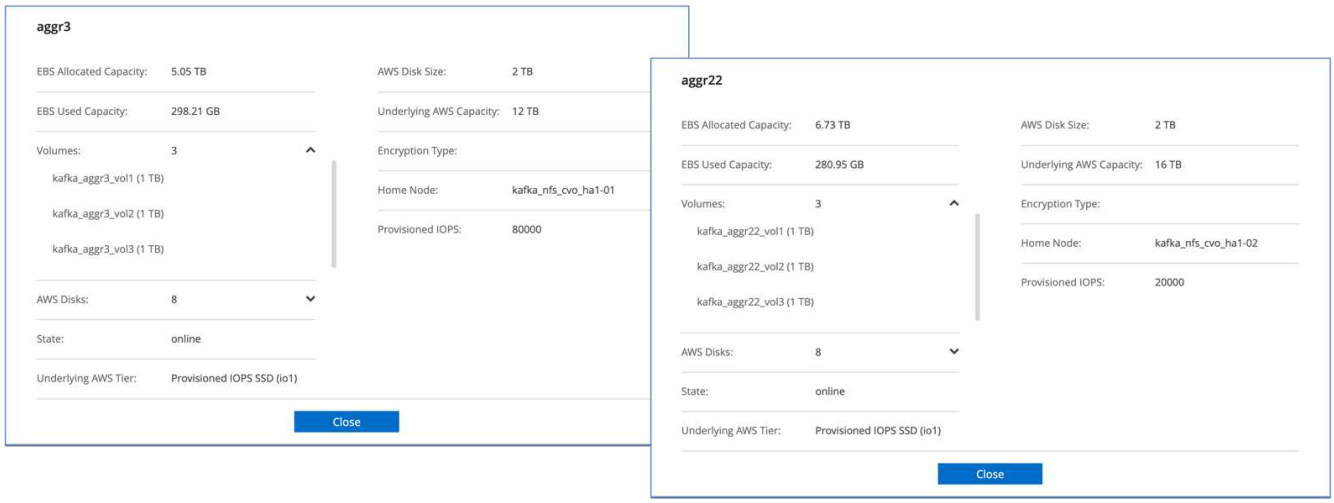
架構設定

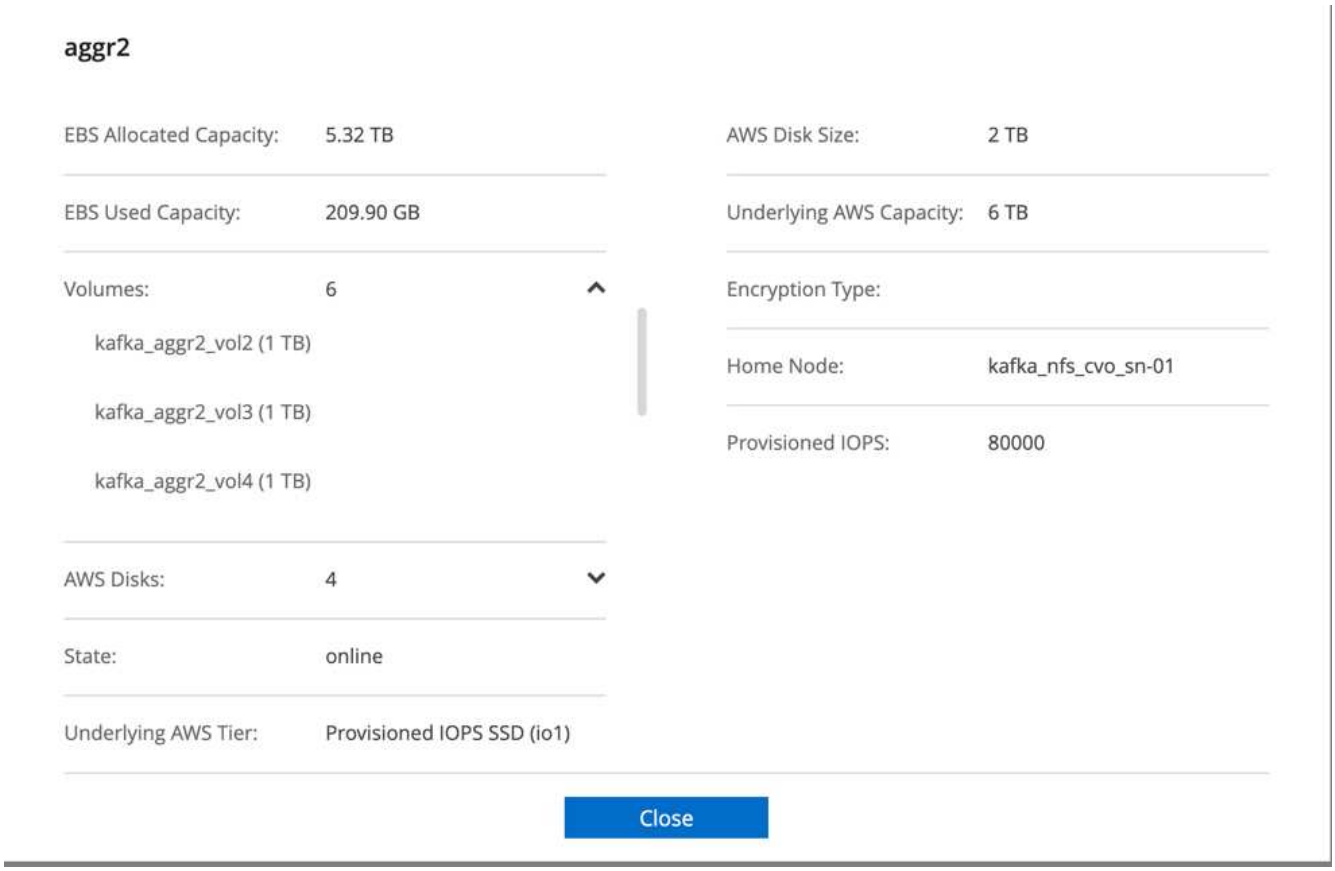
下表顯示使用NAS的Kafka叢集環境組態。

平台元件	環境組態
Kafka 3.2.3	• 3個zookeepers–T2.small • 3個代理伺服器–i3en.2xLarge • 1 x Grafana–c5n.2xLarge • 4個製造商/消費者- c5n.2xLarge *
所有節點上的作業系統	RHEL8.6.
NetApp Cloud Volumes ONTAP 執行個體	HA配對執行個體–m5dn.12xLargex 2節點單一節點執行個體- m5dn.12xLargex 1節點

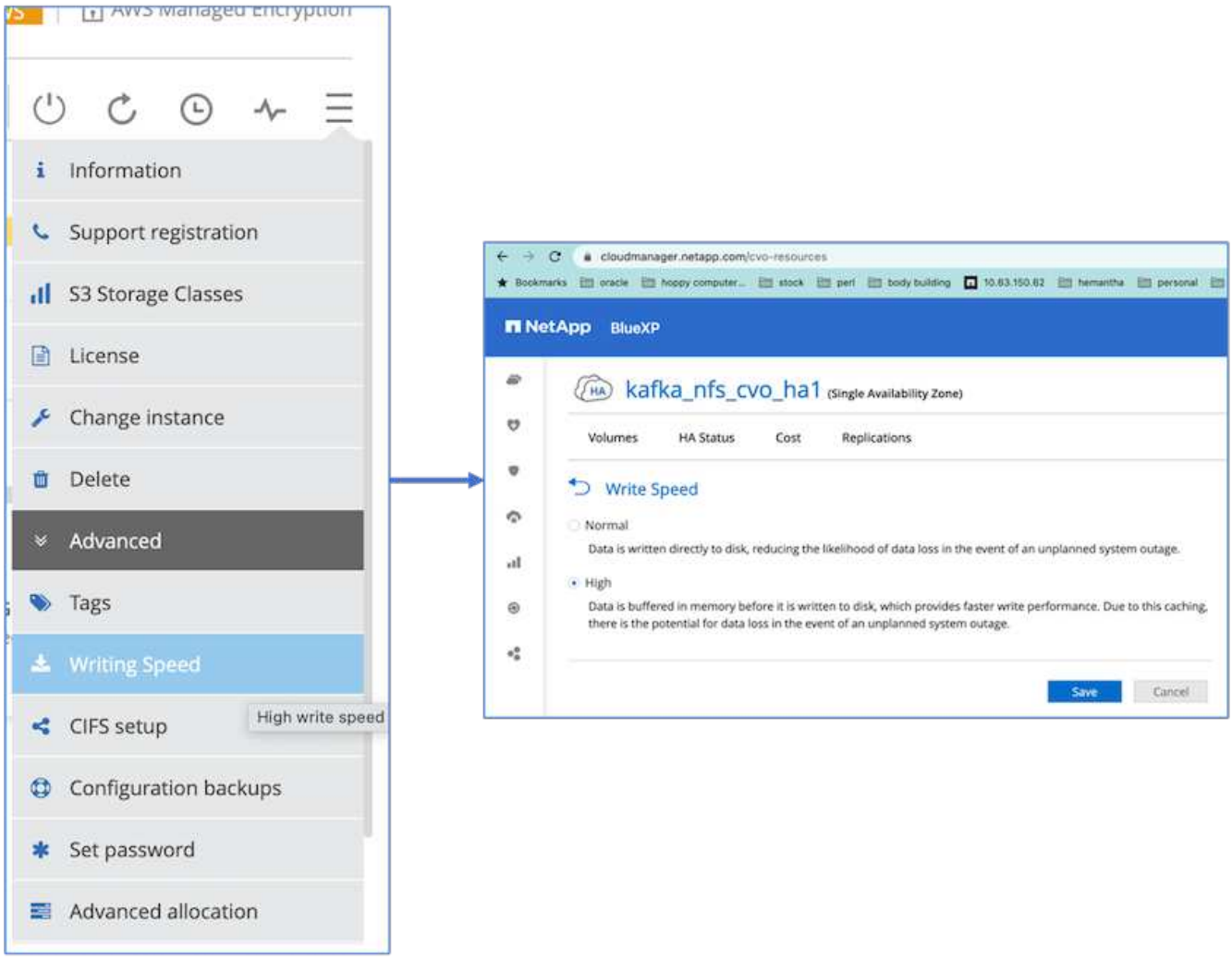
NetApp叢集**Volume ONTAP** 不全

1. 針對「樣片HA配對」、我們在每個儲存控制器的每個集合體上建立了兩個集合體、並有三個磁碟區Cloud Volumes ONTAP 。對於單Cloud Volumes ONTAP 一的實體節點、我們會在一個集合體中建立六個磁碟區。



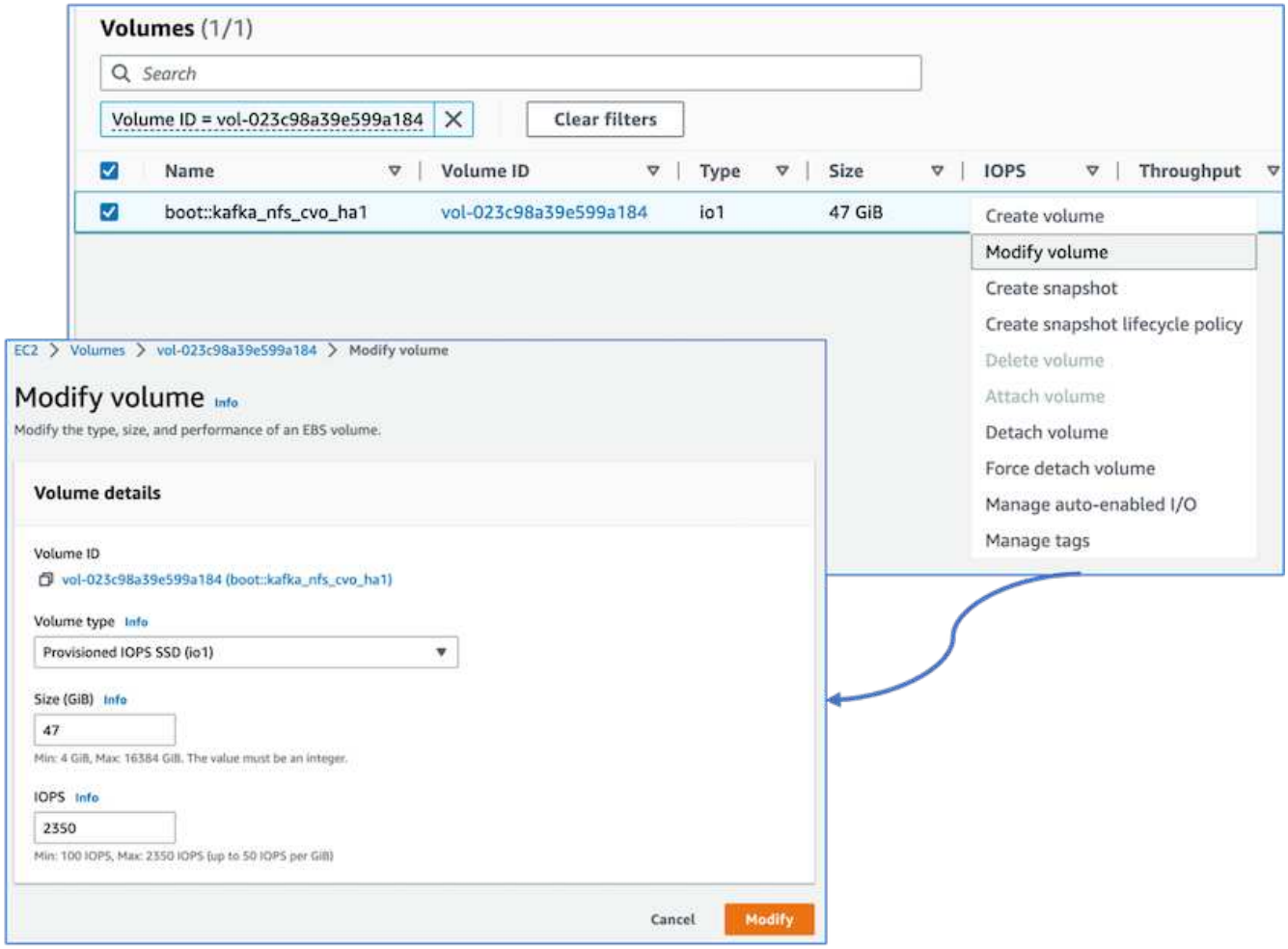


2. 為了提升網路效能、我們同時為HA配對和單一節點啟用高速網路功能。



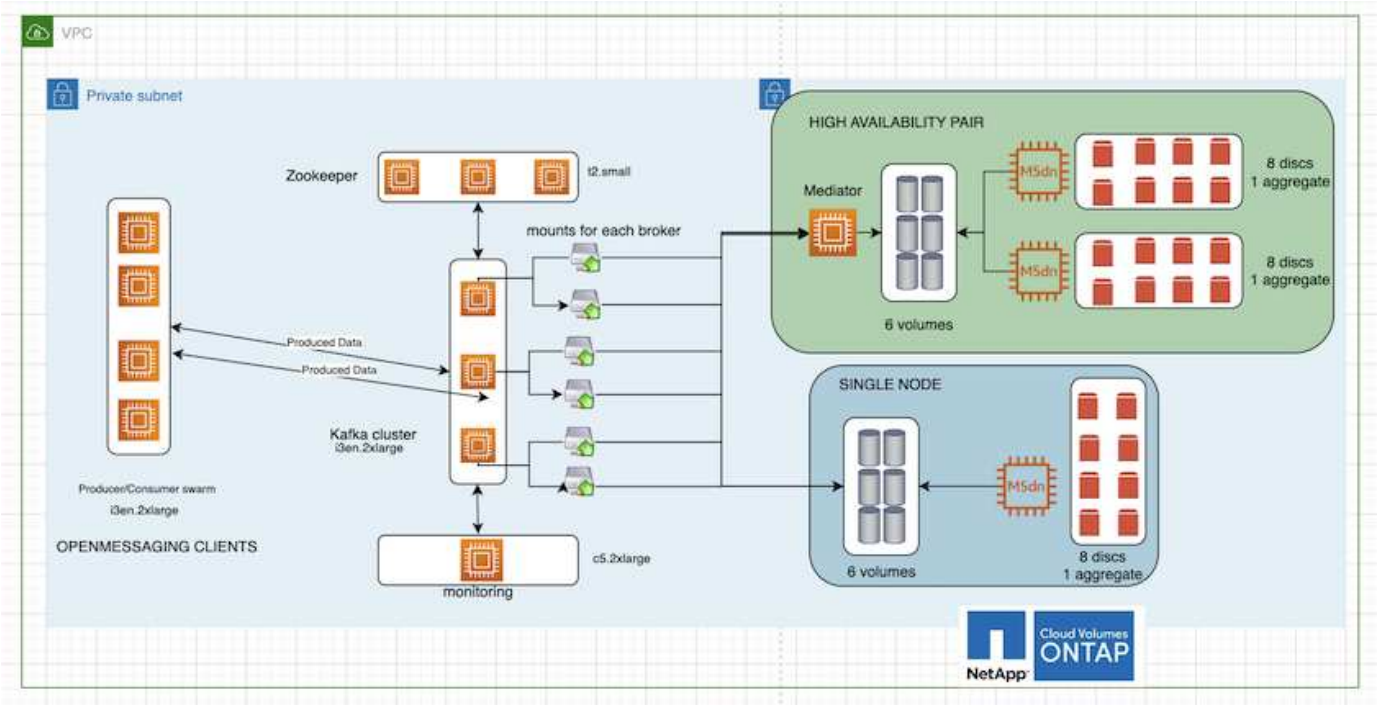
3. 我們注意到ONTAP 、由於不支援IOPS、Cloud Volumes ONTAP 所以我們將IOPS變更為2350、以供支援整個核心磁碟區使用。以47GB大小為基礎的根Volume磁碟Cloud Volumes ONTAP 。下列ONTAP 支援HA配對的支援功能使用下列支援功能、單一節點也適用相同的步驟。

```
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::*>
```



下圖說明NAS型Kafka叢集的架構。

- *運算*我們使用三節點的Kafka叢集、其中三節點的zookeeper集合體執行於專屬的伺服器上。每個代理商都有兩個NFS掛載點、Cloud Volumes ONTAP 透過專屬的LIF、連接到位於整個過程的單一Volume。
- *監控*我們使用兩個節點來搭配Prometheus-Grafana。為了產生工作負載、我們使用了一個獨立的三節點叢集、可以產生和使用這個Kafka叢集。
- * Storage。* Cloud Volumes ONTAP 我們使用HA配對的不二執行個體、在執行個體上掛載一個6TB GP3 AWS-EBS磁碟區。然後使用NFS掛載將磁碟區匯出至Kafka代理程式。



OpenMessage基準測試組態

1. 為了提升NFS效能、我們需要在NFS伺服器和NFS用戶端之間建立更多的網路連線、而NFS用戶端可以使用nconnect建立。執行下列命令、以nconnect選項在代理節點上掛載NFS磁碟區：

```
[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. 檢查Cloud Volumes ONTAP 內部的網路連線。下列ONTAP 支援從單Cloud Volumes ONTAP 一支援節點使用下列支援功能。同樣的步驟也適用於Cloud Volumes ONTAP 此功能。

```
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. 我們使用下列Kafka `server.properties` 所有的Kafka經紀人都能提供Cloud Volumes ONTAP。`log.dirs` 每個代理的屬性各不相同、其餘屬性則適用於代理程式。若為Broker1、則為 `log.dirs` 價值如下：

```
[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 ~]#
```

   - 若為Broker2 `log.dirs` 屬性值如下：

```
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
```

   - 若為Broker3 `log.dirs` 屬性值如下：

```
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. 對於單Cloud Volumes ONTAP 一的支援節點、卡夫卡（Kafka） `servers.properties` 與Cloud Volumes ONTAP 不包括在內的其他不相同 `log.dirs` 屬性。
   - 若為Broker1、則為 `log.dirs` 價值如下：

```
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
```

   - 若為Broker2 `log.dirs` 價值如下：

```
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
```

   - 若為Broker3 `log.dirs` 屬性值如下：

```
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中的工作負載會設定下列內容： `(/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` 可能因使用案例而異。在效能測試中、我們使用3K。

我們使用OMB的兩個不同驅動程式：同步或處理量、來產生Kafka叢集上的工作負載。

- 用於Sync驅動程式內容的yaml檔案如下 `(/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
```

- 用於處理量驅動程式內容的yaml檔案如下 `(/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
```

測試方法

1. 根據上述規格、我們使用Terraform和Ansible來配置Kafka叢集。Terraform是用來為Kafka叢集使用AWS執行個體來建置基礎架構、Ansible則是在這些執行個體上建置Kafka叢集。
2. 使用上述工作負載組態和Sync驅動程式觸發OMB工作負載。

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

3. 另一個工作負載是透過具有相同工作負載組態的處理量驅動程式觸發。

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

觀察

使用兩種不同類型的驅動程式來產生工作負載、以基準測試在NFS上執行的Kafka執行個體效能。驅動程式之間的差異在於記錄排清內容。

若為Cloud Volumes ONTAP 「解決方案」配對：

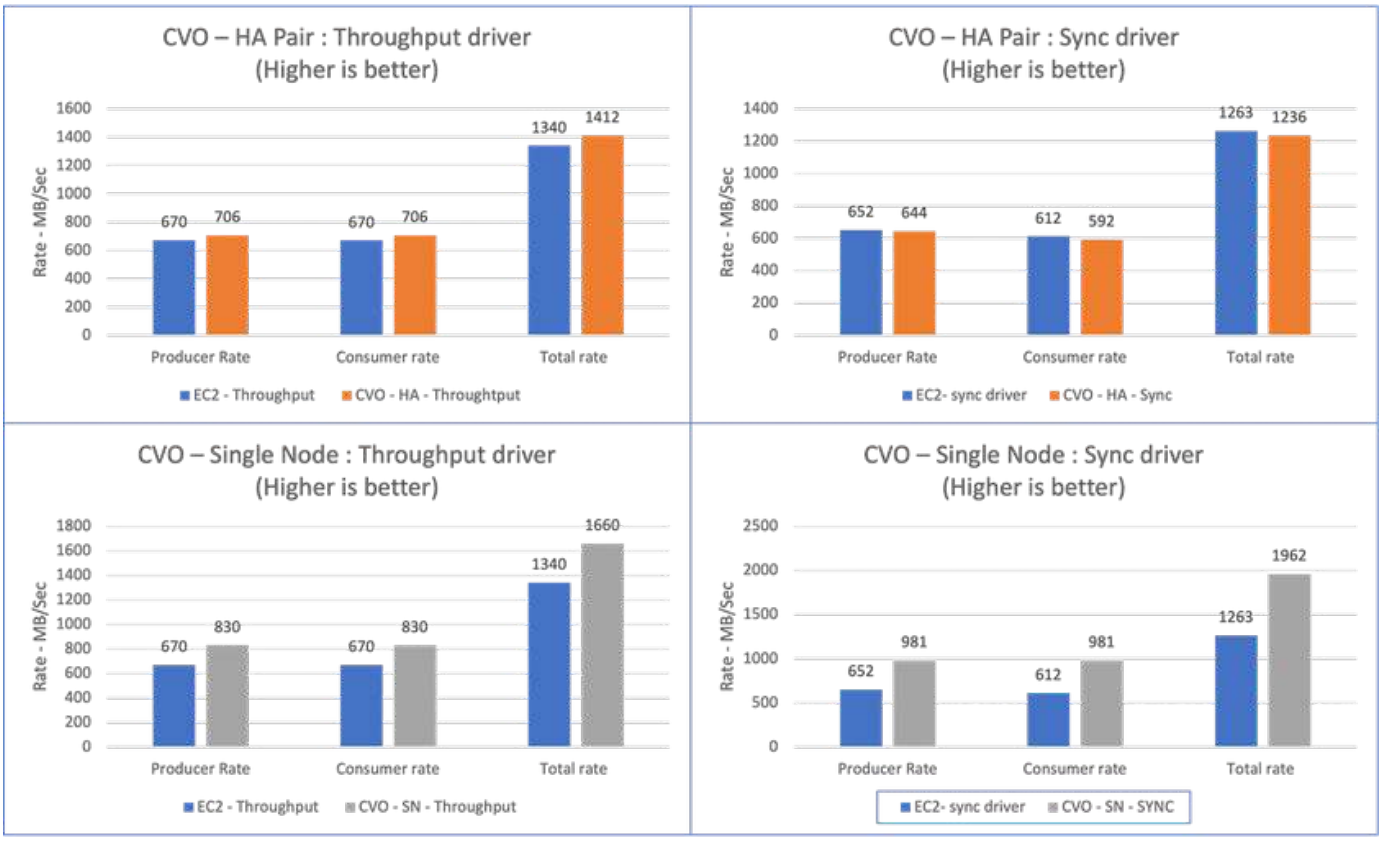
- Sync驅動程式持續產生的總處理量：約1236 Mbps。
- 為處理量驅動程式產生的總處理量：尖峰約1412 Mbps。

對於單Cloud Volumes ONTAP 一的節點：

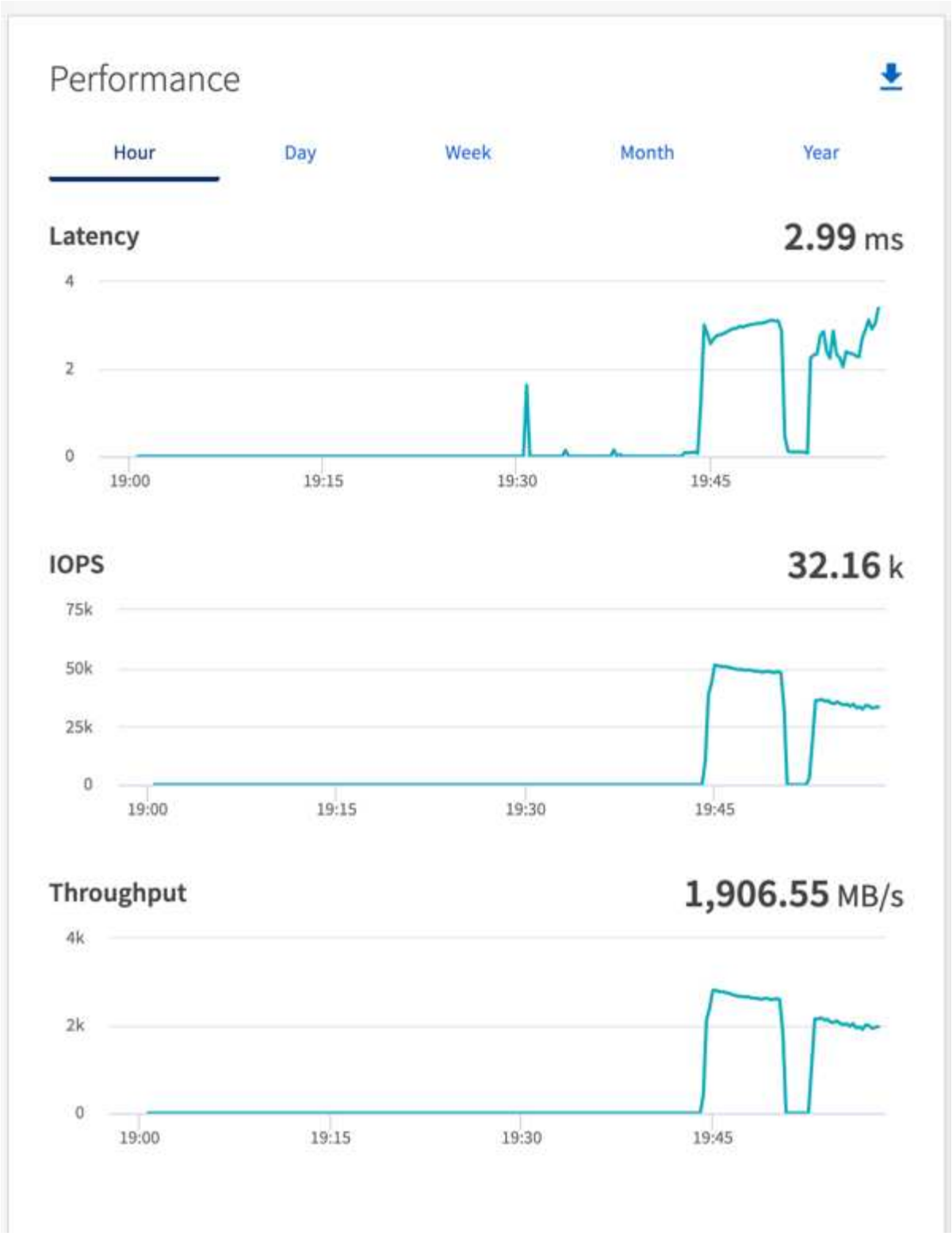
- Sync驅動程式持續產生的總處理量：約1962MBps。
- 處理量驅動程式產生的總處理量：尖峰約1660MBps

同步處理驅動程式可在記錄立即排入磁碟時產生一致的處理量、而處理量驅動程式則會在大量將記錄提交至磁碟時產生大量處理量。

這些處理量編號是針對指定的AWS組態所產生。為了達到更高的效能需求、可以進一步擴充和調整執行個體類型、以獲得更好的處理量。總處理量或總處理率是生產者和使用者速率的組合。



執行處理量或同步驅動程式基準測試時、請務必檢查儲存處理量。



AWS FSX for NetApp ONTAP 的效能概觀與驗證

NetApp NFS 上安裝儲存層的 Kafka 叢集、是 AWS FSX for NetApp ONTAP 效能的基準測試。基準測試範例將於下列各節中說明。

AWS FSX for NetApp ONTAP 中的 Apache Kafka

網路檔案系統（ NFS ）是廣泛使用的網路檔案系統、用於儲存大量資料。在大多數組織中、像 Apache Kafka 這樣的串流應用程式越來越多地產生資料。這些工作負載需要擴充性、低延遲、以及具備現代化儲存功能的健全資料擷取架構。為了實現即時分析並提供可據以行動的洞見、需要設計完善且效能優異的基礎架構。

Kafka by 的設計可搭配 POSIX 相容的檔案系統使用、並仰賴檔案系統來處理檔案作業、但在 NFSv3 檔案系統上儲存資料時、 Kafka Broker NFS 用戶端可以與本地檔案系統（例如 XFS 或 ext4 ）不同地解譯檔案作業。常見的例子是 NFS 愚蠢重新命名、導致 Kafka Brokers 在擴充叢集和重新分配分割區時失敗。為了因應這項挑戰、 NetApp 已更新開放原始碼 Linux NFS 用戶端、現在已在 RHEL8.7 、 RHEL9.1 中普遍提供變更、並從目前的適用於 NetApp ONTAP 版本 ONTAP 9.12.1 的 FSX 中獲得支援。

Amazon FSX for NetApp ONTAP 可在雲端中提供完全託管、可擴充且高效能的 NFS 檔案系統。適用於 NetApp ONTAP 的 FSX 上的 Kafka 資料可擴充以處理大量資料、並確保容錯能力。NFS 可為重要且敏感的資料集提供集中化的儲存管理與資料保護。

這些增強功能可讓 AWS 客戶在 AWS 運算服務上執行 Kafka 工作負載時、善用適用於 NetApp ONTAP 的 FSX 。這些效益包括：
* 降低 CPU 使用率、以縮短 I/O 等待時間
* 更快的 Kafka 代理程式恢復時間。
* 可靠性與效率。
* 擴充性與效能。
* 多可用性區域可用度。
* 資料保護。

AWS FSX for NetApp ONTAP 的效能概觀與驗證

安裝在NetApp NFS上的儲存層Kafka叢集、是以AWS雲端效能為基準。基準測試範例將於下列各節中說明。

AWS FSX for NetApp ONTAP 中的 Kafka

採用 AWS FSX for NetApp ONTAP 的 Kafka 叢集、在 AWS 雲端中以效能為基準。以下各節將說明此基準測試。

架構設定

下表顯示使用 AWS FSX for NetApp ONTAP 的 Kafka 叢集環境組態。

平台元件	環境組態
Kafka 3.2.3	• 3個zookeepers–T2.small • 3個代理伺服器–i3en.2xLarge • 1 x Grafana–c5n.2xLarge • 4個製造商/消費者- c5n.2xLarge *
所有節點上的作業系統	RHEL8.6.
AWS FSX for NetApp ONTAP	具有 4GB/ 秒處理量和 160000 IOPS 的多 AZ

NetApp FSX for NetApp ONTAP 設定

1. 在我們的初始測試中、我們為 NetApp ONTAP 檔案系統建立了 FSX 、容量為 2TB 、傳輸量為每秒 2GB 的 40000 IOPs 。

```
[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"}
```

   在我們的範例中、我們透過 AWS CLI 部署適用於 NetApp ONTAP 的 FSX 。您必須視需要在環境中進一步自訂命令。此外、也可透過 AWS 主控台部署和管理適用於 NetApp ONTAP 的 FSX 、以減少命令列輸入、提供更輕鬆、更精簡的部署體驗。

   FSX for NetApp ONTAP 的文件資料、我們測試區域（美國東部 -1 ）每秒 2 GB 處理量檔案系統的最高 IOPS 為 8 、 000 IOPS 。NetApp ONTAP 檔案系統的 FSX 總 IOPS 為 16 、 000 IOPS 、需要每秒 4 GB 的處理量部署、我們將在本文件稍後說明。

   如需適用於 NetApp ONTAP 效能規格的 FSX 詳細資訊、請隨時造訪 AWS FSX for NetApp ONTAP 文件： https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html 。

   您可以在這裡找到適用於 FSX 「 cree-file-system 」的詳細命令列語法： https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

   例如、您可以指定特定的 KMS 金鑰、而非指定 KMS 金鑰時所使用的預設 AWS FSX 主要金鑰。

2. 為 NetApp ONTAP 檔案系統建立 FSX 時、請等到 JSON 將檔案系統描述如下之後、「生命週期」狀態變更為「可用」：

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

3. 與 fsxadmin 使用者登入適用於 NetApp ONTAP SSH 的 FSX 、以驗證認證：
   Fsxadmin 是建立時適用於 NetApp ONTAP 檔案系統之 FSX 的預設管理帳戶。fsxadmin 的密碼是我們在步驟 1 中完成的第一次在 AWS 主控台或 AWS CLI 中建立檔案系統時所設定的密碼。

```
[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. 認證通過驗證後、請在適用於 NetApp ONTAP 檔案系統的 FSX 上建立儲存虛擬機器

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

儲存虛擬機器（ SVM ）是一部隔離式檔案伺服器、具有自己的管理認證和端點、可用於管理和存取適用於 NetApp ONTAP 磁碟區的 FSX 中的資料、並提供適用於 NetApp ONTAP 多租戶的 FSX 。

5. 設定主要儲存虛擬機器之後、請將 SSH 移至新建立的 NetApp ONTAP 檔案系統 FSX 、並使用以下範例命令在儲存虛擬機器中建立 Volume 、同樣地、我們也會建立 6 個磁碟區來進行此驗證。根據我們的驗證結果、保留預設成分（ 8 ）或以下成分、以提供更好的 Kafka 效能。

```
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. 我們的磁碟區需要額外的容量來進行測試。將磁碟區的大小延伸至 2TB 、並安裝在連接路徑上。

```
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
```

在適用於 NetApp ONTAP 的 FSX 中、磁碟區可以精簡配置。在我們的範例中、擴充磁碟區總容量超過檔案系統總容量、因此我們需要擴充檔案系統總容量、才能解除鎖定額外的資源配置磁碟區容量、我們將在下一步中示範。

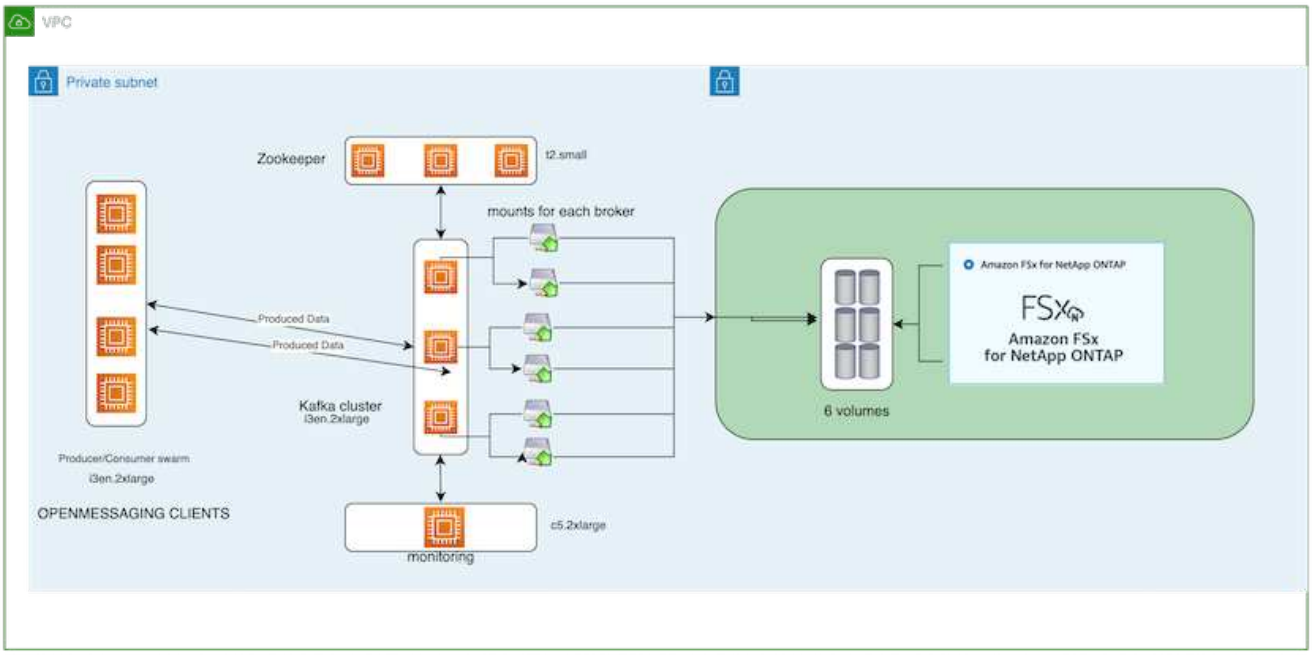
7. 接下來、為了提升效能與容量、我們將 NetApp ONTAP 處理量容量的 FSX 從每秒 2 GB 擴充至每秒 4 GB 、 IOPS 擴充至 160000 、容量則擴充至 5 TB

```
[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
```

您可以在這裡找到適用於 FSX 「 update-file-system 」的詳細命令列語法：
https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html

8. NetApp ONTAP 磁碟區的 FSX 會在 Kafka Brokers 中使用 nconnect 和預設選項進行掛載

下圖顯示我們的 FSX for NetApp ONTAP 卡夫卡叢集的最終架構：



- 運算：我們使用三節點的 Kafka 叢集、在專用伺服器上執行三節點的 zookeeper 群集。每個代理程式都

有六個 NFS 掛載點、可連接至 FSX for NetApp ONTAP 執行個體上的六個磁碟區。

- 監控。我們使用兩個節點作為 Prometheus-Grafana 組合。為了產生工作負載、我們使用了一個獨立的三節點叢集、可以產生和使用這個Kafka叢集。
- 儲存設備。我們使用適用於 NetApp ONTAP 的 FSX 搭配六個 2TB 磁碟區。然後使用 NFS 掛載將該 Volume 匯出至 Kafka 代理程式。 NetApp ONTAP 磁碟區的 FSX 會在 Kafka 代理程式中以 16 個 nconnect 工作階段和預設選項掛載。

OpenMessage 基準測試組態。

我們使用的組態與 NetApp Cloud Volumes ONTAP 相同、其詳細資料如下：
https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

測試方法

1. Kafka 叢集是根據上述規格、使用 terraform 和 Ansible 來進行佈建。Terraform 用於使用適用於 Kafka 叢集的 AWS 執行個體來建置基礎架構、 Ansible 則在其上建置 Kafka 叢集。
2. 使用上述工作負載組態和Sync驅動程式觸發OMB工作負載。

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

3. 另一個工作負載是透過具有相同工作負載組態的處理量驅動程式觸發。

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

觀察

使用兩種不同類型的驅動程式來產生工作負載、以基準測試在NFS上執行的Kafka執行個體效能。驅動程式之間的差異在於記錄排清內容。

對於 Kafka Replication factor 1 和適用於 NetApp ONTAP 的 FSX ：

- 同步驅動程式一致產生的總處理量：約 3218 Mbps 、尖峰效能約 3652 Mbps 。
- 輸送量驅動程式一致產生的總處理量：約 3679 Mbps 、尖峰效能約 3908 Mbps 。

適用於具有複寫係數 3 的 Kafka 和適用於 NetApp ONTAP 的 FSX ：

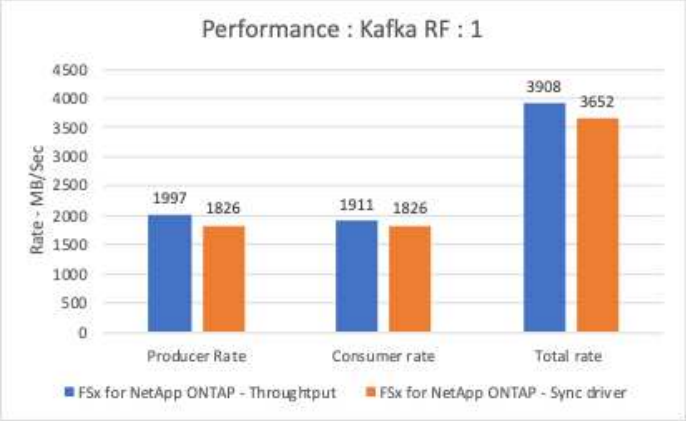
- 同步驅動程式一致產生的總處理量：約 1252 Mbps 、尖峰效能約 1382 Mbps 。
- 輸送量驅動程式一致產生的總處理量：約 1218 Mbps 、尖峰效能約 1328 Mbps 。

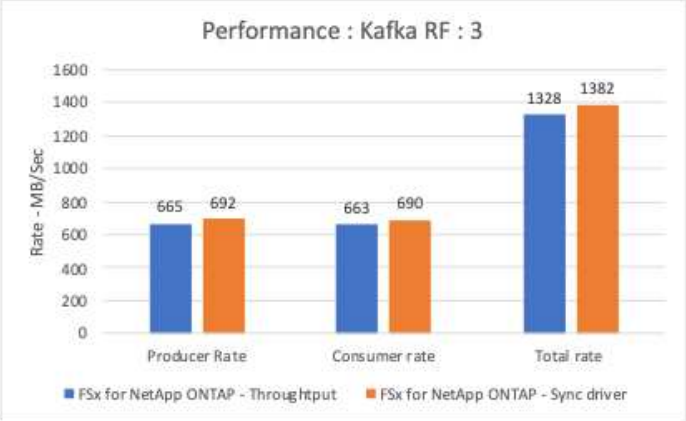
在 Kafka 複寫係數 3 中、 NetApp ONTAP 的 FSX 執行讀寫作業三次、在 Kafka 複寫係數 1 中、 NetApp ONTAP 的讀寫作業是在 FSX 上執行一次、因此在兩次驗證中、 我們能夠達到每秒 4GB 的最大處理量。

同步處理驅動程式可在記錄立即排入磁碟時產生一致的處理量、而處理量驅動程式則會在大量將記錄提交至磁碟

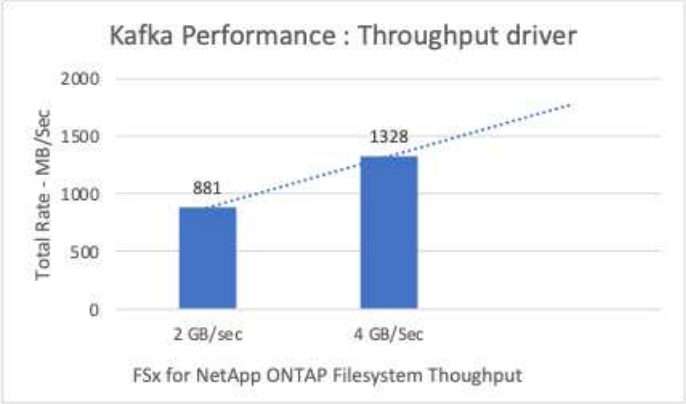
時產生大量處理量。

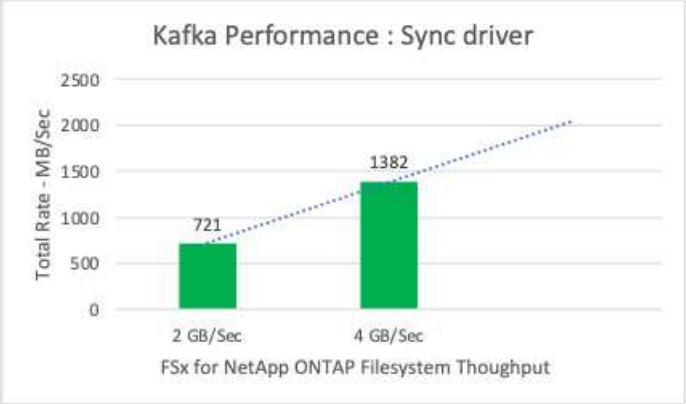
這些處理量編號是針對指定的AWS組態所產生。為了達到更高的效能需求、可以進一步擴充和調整執行個體類型、以獲得更好的處理量。總處理量或總處理率是生產者和使用者速率的組合。



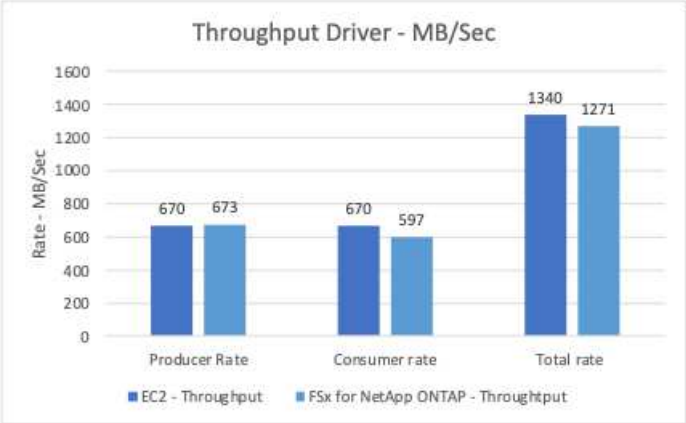


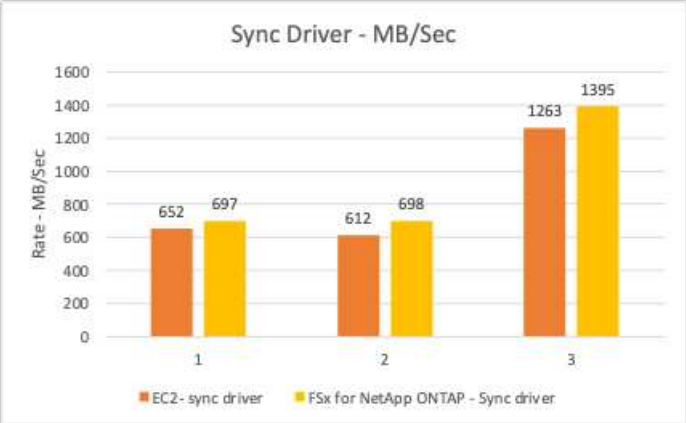
下表顯示 NetApp ONTAP 的 2GB/ 秒 FSX 和 Kafka 複寫係數 3 的 4GB/ 秒效能。複寫因素 3 會在 NetApp ONTAP 儲存設備的 FSX 上執行三次讀寫作業。處理量驅動程式的總速率是 881 MB/s 、在 NetApp ONTAP 檔案系統的 2GB/ 秒 FSX 上執行讀寫 Kafka 作業約 2.64 GB/ 秒、處理量驅動程式的總速率是 1328 MB/ 秒、可讀寫 Kafka 作業約 3.98 GB/ 秒。卡夫卡的效能是線性的、可根據 NetApp ONTAP 處理量的 FSX 進行擴充。





下表顯示 EC2 執行個體與 NetApp ONTAP 的 FSX 之間的效能（ Kafka 複寫係數： 3 ）

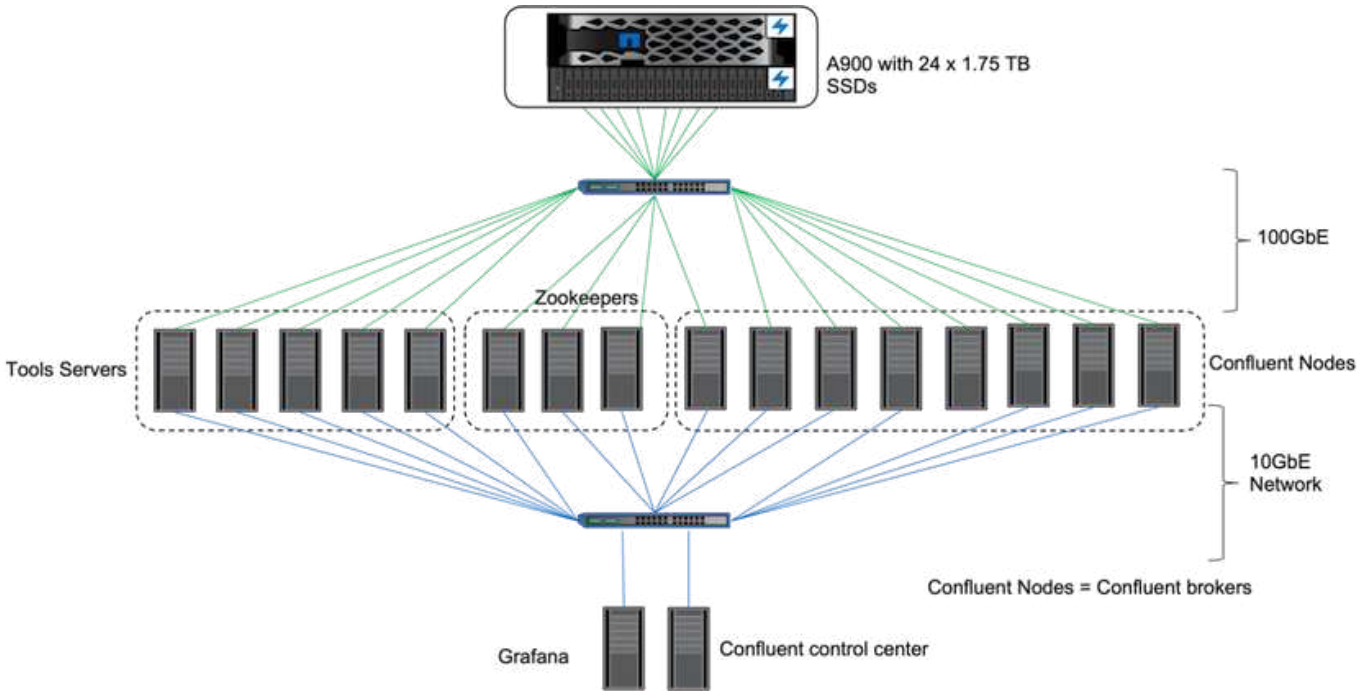




透過**AFF** 內部部署的**VMware**解決方案實現效能總覽與驗證

內部部署時、我們使用NetApp AFF VMware不支援ONTAP VMware的儲存控制器搭配VMware不支援套件（更新版本）來驗證Kafka叢集的效能與擴充性。我們使用的測試平台與ONTAP 先前採用的階層式儲存最佳實務做法相同、只要搭配使用即可AFF 。

我們使用Conflent Kafka 6.2.0來評估AFF 《The》。叢集具備八個代理節點和三個zookeeper節點。為了進行效能測試、我們使用了五個OMB工作節點。



儲存組態

我們使用NetApp FlexGroups執行個體為記錄目錄提供單一命名空間、簡化還原與組態。我們使用NFSv4.1和pNFS提供直接路徑存取記錄區段資料。

用戶端調校

每個用戶端都使用FlexGroup 下列命令來掛載這個執行個體。

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

此外、我們也增加了 `max_session_slots` 預設值 `64` 至 `180`。這與ONTAP 不符合的預設工作階段插槽限制。

Kafka代理商調校

為了將測試中系統的處理量最大化、我們大幅增加了特定關鍵執行緒集區的預設參數。我們建議針對大多數組態、遵循Conflent Kafka最佳實務做法。此調校可將出色的I/O與儲存設備的並行性最大化。這些參數可調整以符合代理商的運算資源和儲存屬性。

```
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
```

工作負載產生器測試方法

我們使用與雲端測試相同的OMB組態來測試處理量驅動程式和主題組態。

1. 使用可在一個叢集上配置的可執行個體AFF FlexGroup 。

```
---
- 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. 已在ONTAP SVM上啟用pNFS。

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

3. 此工作負載是透過處理量驅動程式觸發、其工作負載組態與Cloud Volumes ONTAP 執行動作時相同。請參閱「」一節[穩定狀態效能]」。工作負載使用3個複寫係數、表示NFS中保留了三個記錄區段複本。

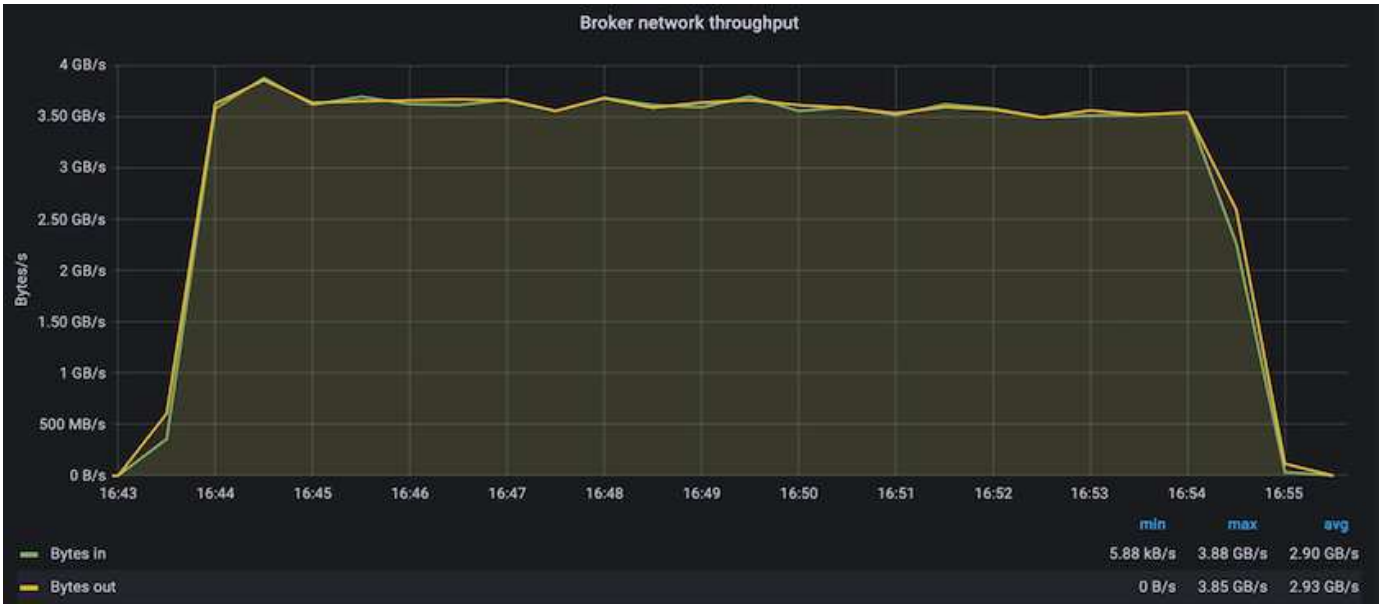
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最後、我們使用待處理項目來完成測量、以衡量消費者是否有能力掌握最新訊息。OMB會在測量開始時暫停使用者、藉此建構待處理項目。這會產生三個不同的階段：建立待處理項目（僅限生產商流量）、減少待處理項目（使用者負擔沉重的階段、消費者會在某個主題中趕上錯過的事件）、以及穩定狀態。請參閱「」一節探索儲存限制」以取得更多資訊。

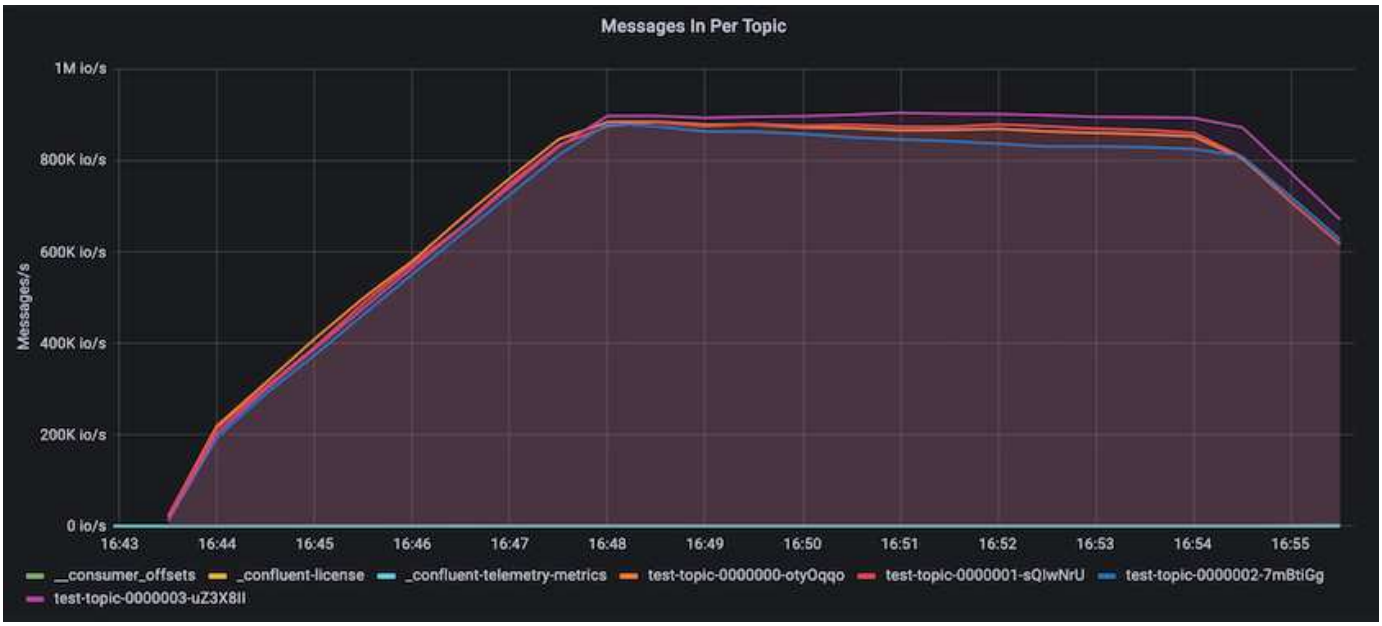
穩定狀態效能

我們使用AFF OpenMessaging基準測試評估了《支援不支援的資料」、以提供類似Cloud Volumes ONTAP 於AWS中的《支援的資料、支援的資料、以及AWS中的DAS。所有效能值均代表生產商與消費者層級的Kafka叢集處理量。

Conflent Kafka與AFF 《The》（《The》）的穩定狀態效能、使生產商與消費者的平均處理量均超過3.4GBps。這是卡夫卡叢集內超過340萬封訊息。透過將BrokerTopicMetrics的持續處理量以每秒位元組為單位視覺化、我們可以看到AFF 由VMware支援的卓越穩定狀態效能和流量。



這與每個主題所傳送訊息的檢視非常一致。下圖提供每個主題的詳細資料。在測試的組態中、我們在四個主題中、每個主題都看到將近900k個訊息。

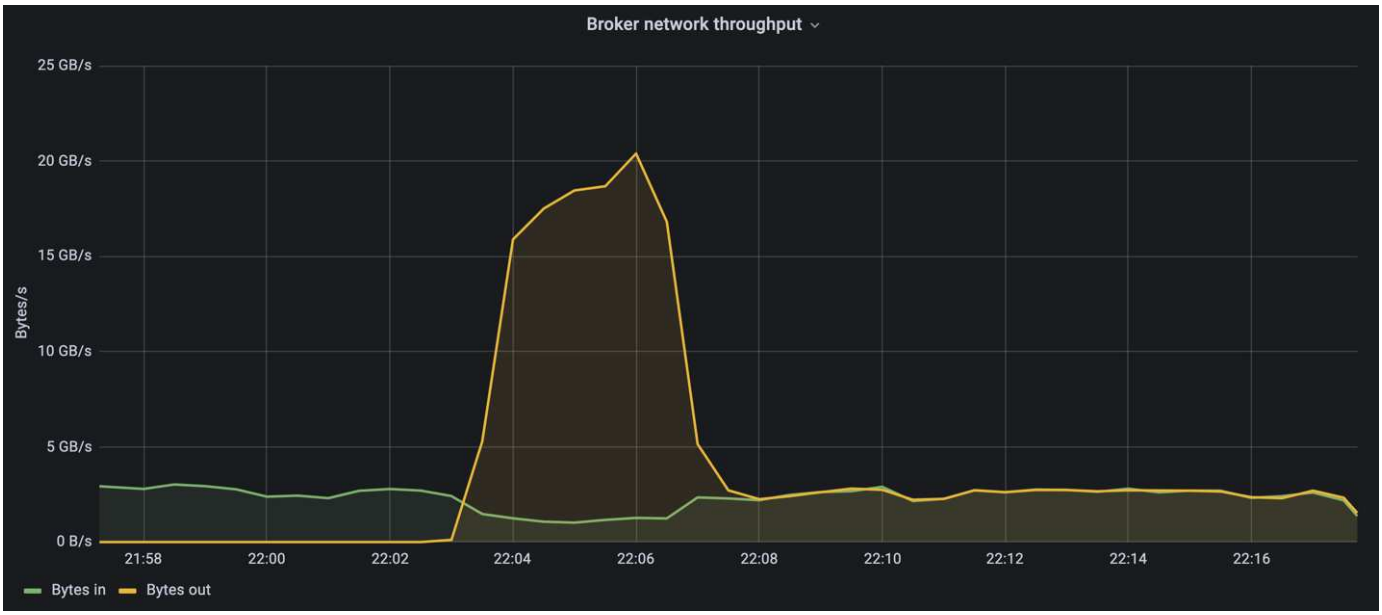


極致效能、探索儲存限制

針對這個解決方案、我們也使用待處理項目功能測試OMB AFF 。待處理項目功能會暫停訂閱消費者、而卡夫卡叢集中會建立大量的事件。在此階段中、只會發生產生者流量、產生提交至記錄的事件。這最能模擬批次處理或離線分析工作流程；在這些工作流程中、會啟動使用者訂閱、而且必須讀取已從代理快取中移出的歷史資料。

為了瞭解此組態中對使用者處理量的儲存限制、我們測量了純產生者階段、以瞭解A900可能會吸收多少寫入流量。請參閱下一節「[規模調整指南]」以瞭解如何運用這些資料。

在這項測量的純生產商部分期間、我們看到高尖峰處理量、使A900效能的極限推升（當其他代理商資源不飽和、無法為生產商和消費者流量提供服務時）。



我們將此測量的訊息大小增加至16k、以限制每個訊息的開銷、並將NFS掛載點的儲存處理量最大化。

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

Conflent Kafka叢集達到4.03GBps的尖峰生產量。

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

在OMB填入事件待處理項目之後、使用者流量便會重新啟動。在測量待處理項目耗盡時、我們觀察到所有主題的尖峰使用者處理量都超過20Gbps。儲存OMB記錄資料的NFS磁碟區總處理量接近30Gbps。

規模調整指南

Amazon Web Services提供 "規模調整指南" 適用於Kafka叢集規模調整與擴充。

此規模提供了一種實用的公式、可用來判斷Kafka叢集的儲存處理量需求：

對於複寫係數為r的tcluster叢集所產生的彙總處理量、Broker儲存設備所接收的處理量如下：

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

這點可以進一步簡化：

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

使用此公式可讓您針對ONTAP Kafka的熱階層需求、選擇適當的支援平台。

下表說明A900的預期生產商處理量、以及不同的複寫因素：

複寫因素	生產商處理量（**GPP**）
3（測量）	3.4.
2.	5.1
1.	10.2

結論

NetApp解決方案解決了這項不合理的重新命名問題、為先前與NFS不相容的工作負載、提供簡單、廉價且集中管理的儲存形式。

這項新模式可讓客戶建立更易於管理的Kafka叢集、以便更輕鬆地移轉和鏡射、以實現災難恢復和資料保護。我們也看到NFS提供更多優勢、例如降低CPU使用率、縮短恢復時間、大幅改善儲存效率、以及透過NetApp

ONTAP 功能提升效能。

何處可找到其他資訊

若要深入瞭解本文所述資訊、請檢閱下列文件和 / 或網站：

- 什麼是Apache Kafka？

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

- 什麼是「愚蠢的重新命名」？

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

- 串流應用程式讀取ONATP。

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

- Kafka的rug-重新命名問題。

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

- NetApp產品文件

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

- 什麼是NFS？

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

- 什麼是Kafka分割區重新指派？

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

- 什麼是OpenMessaging基準測試？

  "https://openmessaging.cloud/"

- 如何移轉Kafka代理商？

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

- 您如何監控Prometheus的Kafka代理商？

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

- Apache Kafka的託管平台

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

- 支援Apache Kafka

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

- Apache Kafka諮詢服務

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

Conflent Kafka搭配NetApp ONTAP 等儲存控制器

TR-4941：運用NetApp ONTAP 的不二儲存控制器、實現流暢的操作

Karthikeyan Nagalingam、Joe Scott、NetApp Rankesh Kumar、Conflent

為了讓Conflent Platform更具擴充性和彈性、IT必須能夠迅速擴充及平衡工作負載。階層式儲存可減少此營運負擔、讓儲存大量資料的工作更容易管理。

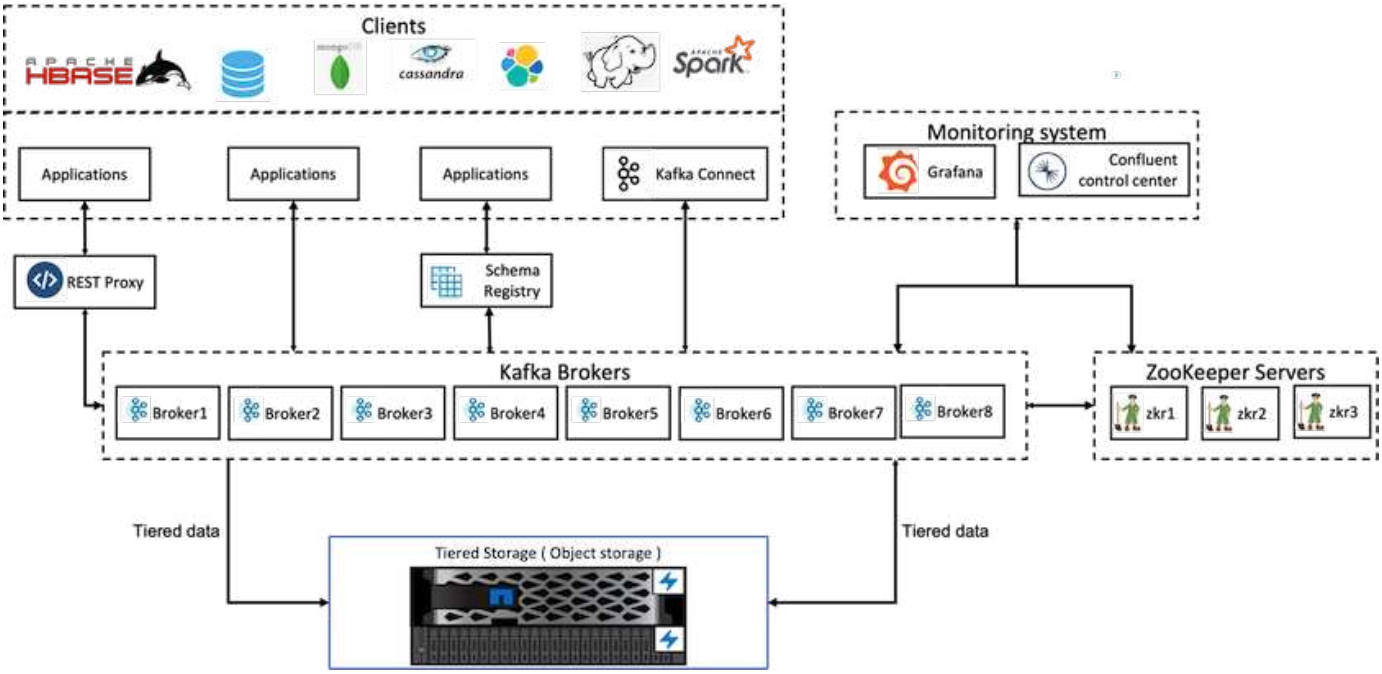
基本構想是將資料儲存設備與資料處理區分開、如此一來、每個儲存區的擴充都變得更加容易。

NetApp ONTAP 支援業界領先的創新技術、資料管理軟體可在資料存放的任何地方提供Conflent的諸多優勢。

本文件概述ONTAP 使用階層式儲存基準測試套件、在NetApp VMware上實現流暢平台的效能標竿。

解決方案

Confluent和NetApp AFF 的Arestora900儲存控制器採用ONTAP 了以資訊串流為設計基礎的分散式系統。兩者皆可橫向擴充、容錯、並可在負載下提供優異效能。透過資料減量技術、將資料佔用空間降至最低、以降低儲存成本的方式、在分散式資料串流和串流處理上相輔相成。藉由AFF 使用VMware的支援、您可以在不影響運算和資料儲存資源的情況下、將資料儲存控制器的效能發揮到極致。如此可簡化系統管理、並可獨立擴充資源。



解決方案架構詳細資料

本節涵蓋在採用NetApp ONTAP 的ConFluent Platform部署中、用於階層式儲存設備的效能驗證所使用的硬體與軟體。下表涵蓋解決方案架構和基礎元件。

平台元件	環境組態
ConFluent Platform 6.2版	• 3個zookeepers • 8個代理伺服器 • 5部工具伺服器 • 1 x Grafana • 1個控制中心
所有節點上的作業系統	Linux（Ubuntu 18.04）
適用於ONTAP 暖桶的NetApp解決方案	• 1 AFF 組供應高可用度（HA）的支援 • 4 x 24 x 800 SSD • S3傳輸協定 • 100GbE
15部Fujitsu PRIMERGY RX2540伺服器	• 2個CPU；總共16個實體核心 • Intel Xeon • 256GB實體記憶體 • 100GbE雙埠

技術總覽

本節說明本解決方案所使用的技術。

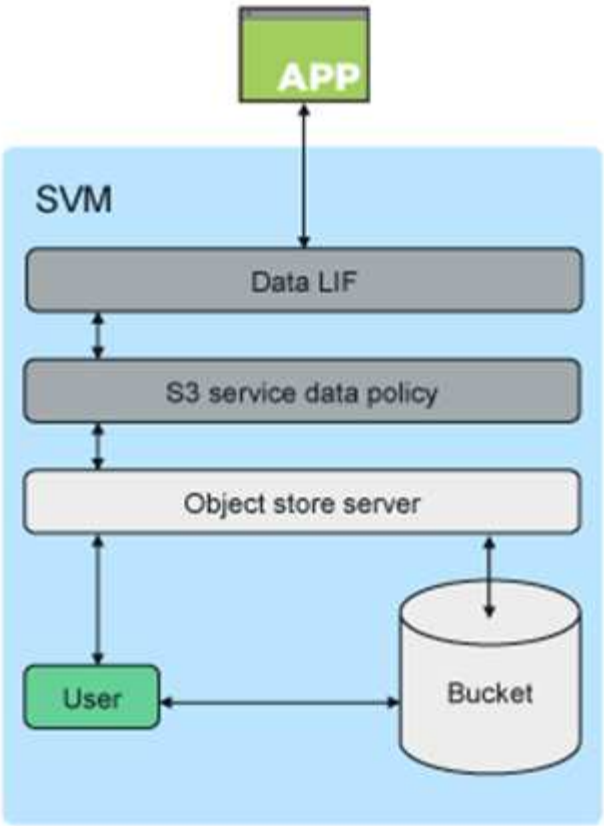
NetApp ONTAP 產品儲存控制器

NetApp ONTAP 產品是高效能的企業級儲存作業系統。

NetApp ONTAP 支援Amazon Simple Storage Service（S3）API。支援Amazon Web Services（AWS）S3 API動作的子集、並可在雲端供應商（AWS、Azure和GCP）和內部部署的ONTAP型系統中、將資料呈現為物件。ONTAP

NetApp StorageGRID 軟件是NetApp物件儲存解決方案的旗艦產品。支援功能：提供邊緣上的擷取和預先處理點、擴充以NetApp為基礎的資料架構來處理物件資料、並增加NetApp產品組合的價值、藉此輔助功能。ONTAP StorageGRID

S3儲存區的存取權是透過授權使用者和用戶端應用程式提供。下圖顯示存取S3儲存區的應用程式。



主要使用案例

支援S3 API的主要目的是提供ONTAP 物件存取功能。支援檔案（NFS和SMB）、區塊（FC和iSCSI）和物件（S3）的統一化解決方案。ONTAP

原生**S3**應用程式

越來越多的應用程式能夠運用ONTAP 支援功能、使用S3進行物件存取。雖然非常適合高容量歸檔工作負載、但對於原生S3應用程式的高效能需求卻迅速成長、包括：

- 分析
- 人工智慧
- 邊緣對核心擷取
- 機器學習

客戶現在可以使用熟悉的管理工具ONTAP （例如、《不穩定系統管理程式ONTAP 》）、快速配置高效能物件儲存設備、以利在VMware進行開發與營運、並ONTAP 充分發揮不穩定的儲存效率與安全性。

無縫端點**FabricPool**

從功能上說起ONTAP 、FabricPool 從功能上說起、支援將資料分層至ONTAP 功能區、以利ONTAP到ONTAP的分層。對於想要重新規劃FAS 現有的物件儲存端點的物件儲存基礎架構用途的客戶來說、這是絕佳的選擇。

支援以兩種方式分層至物件：FabricPool ONTAP

- *本機叢集分層。*非使用中資料會使用叢集生命體階層化至位於本機叢集上的儲存區。

- *遠端叢集分層。*非使用中資料會以類似傳統FabricPool 的速度階層、使用FabricPool 位於遠端叢集上的IC LIF、以及ONTAP 使用位於物件儲存區上的資料LIF、分層至遠端叢集上的儲存區。

如果您想要在現有叢集上使用S3功能、而不需要額外的硬體和管理、那麼支援該功能就很合適。ONTAP對於超過300TB的部署、NetApp StorageGRID 產品技術仍是NetApp物件儲存解決方案的旗艦產品。使用不含功能的不需要取得使用許可證、也不需要使用功能的不含功能的功能。FabricPool ONTAP StorageGRID

NetApp ONTAP 解決方案：適用於**Conflent**階層式儲存設備

每個資料中心都需要維持關鍵業務應用程式的執行、以及重要的資料可用且安全無虞。全新的NetApp AFF Wis-A900系統採用ONTAP 了一套功能強大的功能、採用了一套高恢復能力的設計。我們全新的閃電般快速NVMe儲存系統可避免關鍵任務作業中斷、將效能調校降至最低、並保護資料免受勒索軟體攻擊。

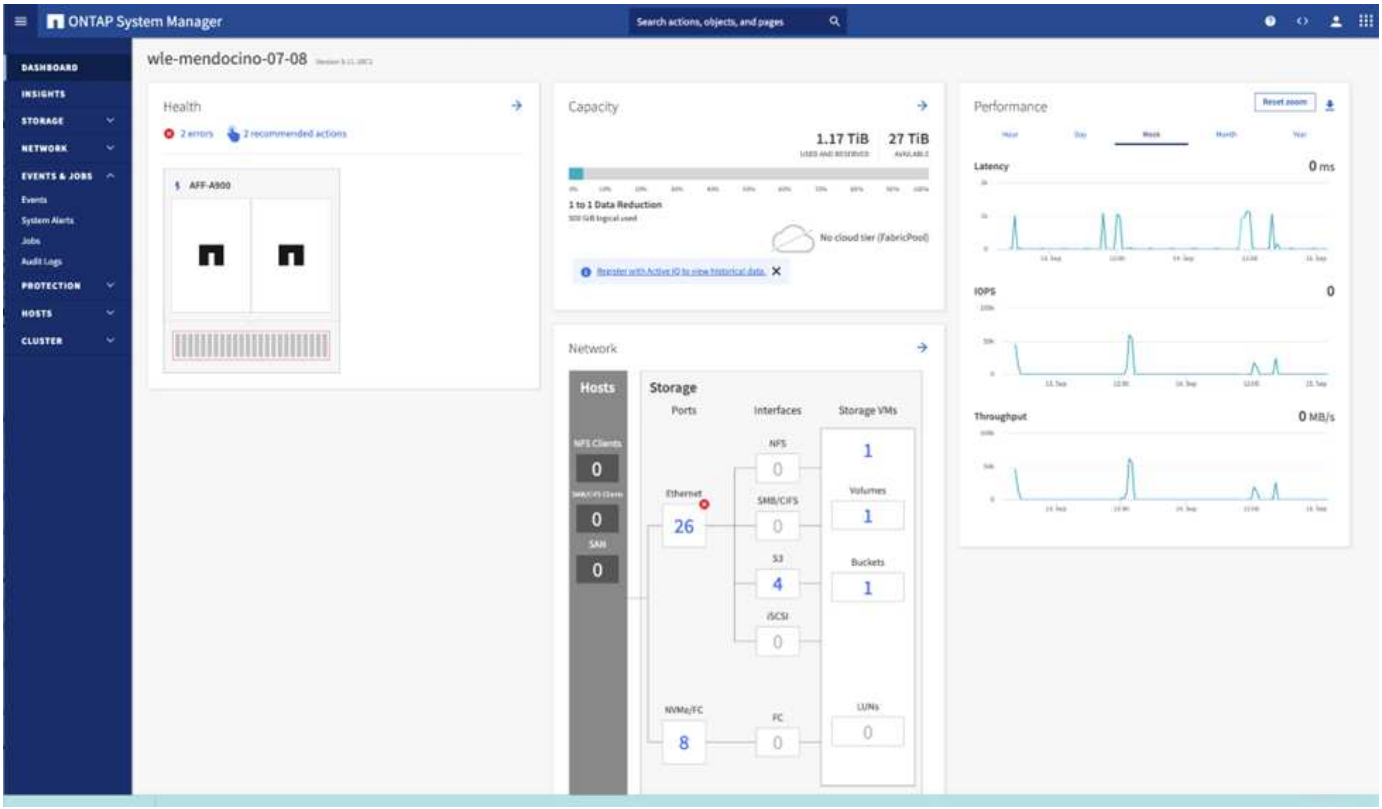
從初始部署到擴充Conflent叢集、您的環境都需要迅速因應不中斷業務關鍵應用程式的變更。利用企業資料管理、服務品質（QoS）和效能、您可以規劃並因應環境需求。ONTAP

結合使用NetApp ONTAP 的功能性和Conflent階層式儲存設備、可將ONTAP 效能提升為橫向擴充儲存目標、並可獨立擴充ConFluent的運算與儲存資源、進而簡化Apache Kafka叢集的管理。

支援豐富擴充儲存功能的支援基礎上、打造出一套功能完善的SS3伺服器。ONTAP ONTAP擴充ONTAP 您的叢集功能可透過擴充S3儲存區、將新增的節點延伸至ONTAP 該叢集、以無縫方式執行。

利用**NetApp System Manager**輕鬆管理**ONTAP**

支援瀏覽器的圖形介面、可讓您在單一窗口中、設定、管理及監控分散於全球各地的整個地區之靜態儲存控制器。ONTAP ONTAP



您可以使用ONTAP System Manager和ONTAP SURE CLI來設定及管理SS3。當您使用System Manager啟用S3並建立貯體時、ONTAP 支援以最佳實務方式進行簡化組態。如果您從CLI設定S3伺服器和儲存區、仍可視

需要使用System Manager來管理、反之亦然。

當您使用System Manager建立S3儲存區時ONTAP 、即可設定系統可用的最高預設效能服務層級。例如AFF 、在一個不完善的系統上、預設設定為「極致」。效能服務層級是預先定義的調適性QoS原則群組。您可以指定自訂QoS原則群組或無原則群組、而非其中一個預設服務層級。

預先定義的調適性QoS原則群組包括下列項目：

- *極致*適用於需要最低延遲和最高效能的應用程式。
- 效能。*適用於效能需求與延遲不佳的應用程式。
- *值。*用於處理量與容量比延遲更重要的應用程式。
- *自訂*指定自訂QoS原則或無QoS原則。

如果您選擇*用於分層*、則不會選擇效能服務層級、系統會嘗試選擇低成本媒體、並針對階層式資料提供最佳效能。

嘗試將此儲存庫配置到具有最適當磁碟的本機層、以符合所選的服務層級。ONTAP不過、如果您需要指定要包含在儲存區中的磁碟、請考慮從CLI設定S3物件儲存、方法是指定本機層（Aggregate）。如果您從CLI設定S3伺服器、仍可視需要使用System Manager來管理。

如果您想要指定要用於儲存區的集合體、只能使用CLI來執行此作業。

Confluent

Confluent Platform是一款全方位資料串流平台、可讓您輕鬆存取、儲存及管理資料、做為持續的即時串流。Conflent是由Apache Kafka原創原創者所打造、以企業級功能擴展Kafka的優勢、同時免除Kafka管理或監控的負擔。如今、超過80%的財星雜誌100大企業都採用資料串流技術、而且大部分企業都使用Conflent。

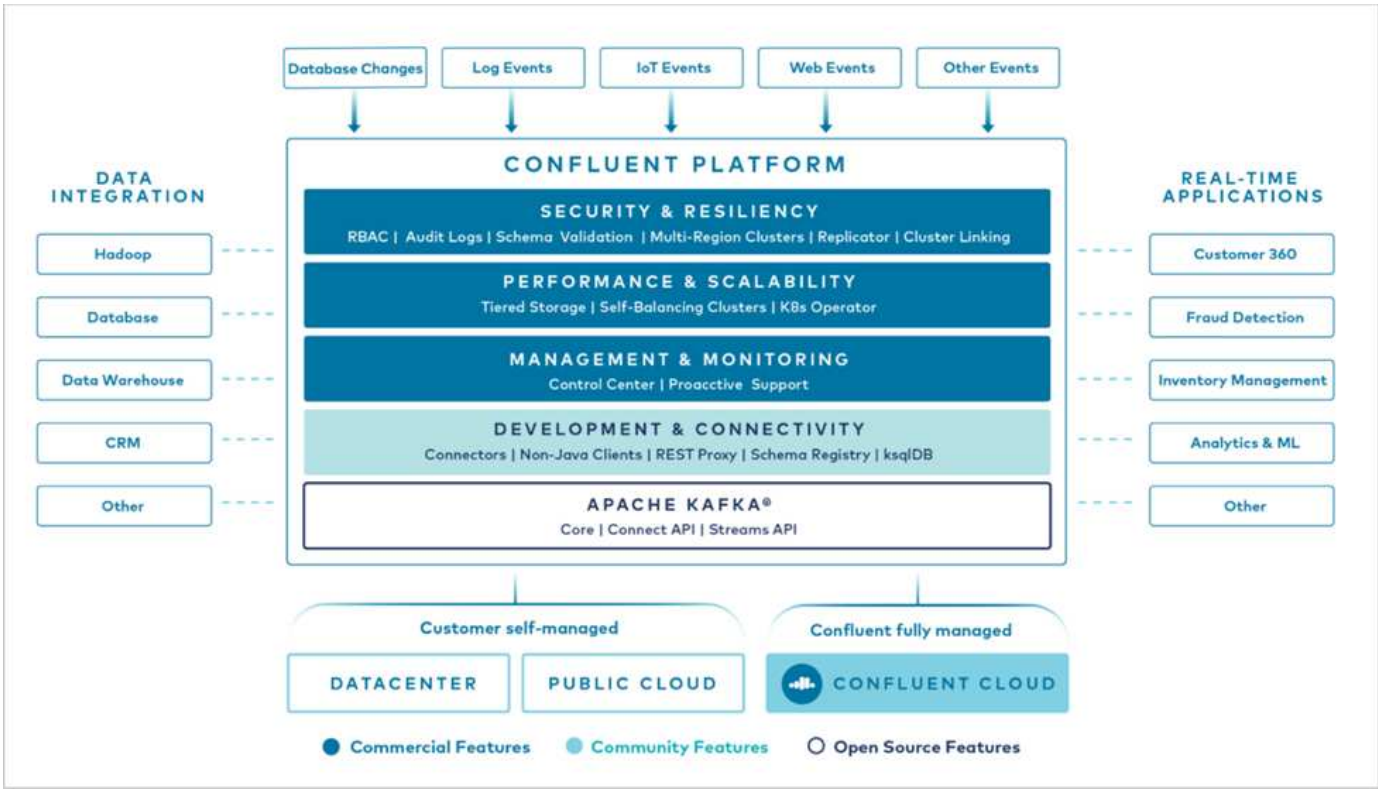
為何選擇**Conflent**？

藉由將歷史與即時資料整合至單一的集中式事實來源、Conflent可讓您輕鬆建置全新類別的現代化事件導向應用程式、取得通用資料管線、並以完整的擴充性、效能與可靠性、釋放強大的新使用案例。

什麼是**ConnFluent**的用途？

Conflent Platform可讓您專注於從資料中獲取商業價值、而非擔心基礎機制、例如資料如何在不同的系統之間傳輸或整合。具體而言、Conflent Platform可簡化資料來源與Kafka之間的連線、建置串流應用程式、以及保護、監控及管理Kafka基礎架構。現在、Conflent Platform適用於各種產業的使用案例、從金融服務、全通路零售和自主汽車、到詐欺偵測、微服務和物聯網等。

下圖顯示ConnFluent Platform的元件。



Confluent事件串流技術總覽

在Confluent Platform的核心是 "卡夫卡"是最受歡迎的開放原始碼分散式串流平台。卡夫卡的主要功能包括：

- 發佈及訂閱記錄串流。
- 以容錯的方式儲存記錄串流。
- 處理記錄串流。

隨裝即用的Confluent Platform也包括架構登錄、REST Proxy、總共100多個預先建置的Kafka連接器和ksqlDB。

Confluent平台企業功能總覽

- * Confluent Control Cent.*一種以UI為基礎的系統、用於管理及監控Kafka。它可讓您輕鬆管理Kafka Connect、以及建立、編輯及管理與其他系統的連線。
- * Kubernetes的Confluent。* Kubernetes的Connent是Kubernetes營運者。Kubernetes營運者提供特定平台應用程式的獨特功能和需求、藉此擴充Kubernetes的協調功能。對於Confluent Platform、這包括大幅簡化Kubernetes上的Kafka部署程序、以及自動化典型的基礎架構生命週期工作。
- * Kafka Connect Connectors。* Connectors使用Kafka Connect API將Kafka連線至其他系統、例如資料庫、金鑰價值儲存區、搜尋索引和檔案系統。Confluent Hub提供可下載的連接器、適用於最受歡迎的資料來源和接收器、包括這些連接器的完整測試和支援版本、以及Confluent Platform。如需詳細資料、請參閱 "請按這裡"。
- *自我平衡叢集。*提供自動負載平衡、故障偵測及自我修復功能。它也可視需要支援新增或取消委任代理人、無需手動調整。
- * Confluent叢集連結。*直接將叢集連線在一起、並透過連結橋接器將主題從一個叢集鏡射到另一個叢集。叢集連結可簡化多資料中心、多叢集及混合雲部署的設定。
- * Confluent自動資料平衡器。*監控叢集的代理程式數量、分割區大小、分割區數目、以及叢集內的領導者數

量。它可讓您將資料移轉至整個叢集、以建立平均工作負載、同時節流重新平衡流量、將對正式作業工作負載的影響降至最低、同時重新平衡。

- * Confluent replicator。*讓您在多個資料中心中維護多個Kafka叢集變得比以往更輕鬆。
- *分層儲存。*提供使用您最喜愛的雲端供應商儲存大量Kafka資料的選項、藉此降低營運負擔和成本。透過階層式儲存設備、您只能在需要更多運算資源時、將資料保存在具成本效益的物件儲存設備上、並擴充代理商。
- * Connent Jms用戶端。* Conflent Platform包含適用於Kafka的與Jms相容的用戶端。此Kafka用戶端實作了JMS 1.1標準API、使用Kafka Brokers做為後端。如果您使用的是使用Jms的舊應用程式、而且想要以Kafka取代現有的Jms訊息代理程式、這項功能就很實用。
- * Conflent MQtT Proxy。*提供一種從MQtT裝置和閘道直接發佈資料至Kafka的方法、而不需要中間的MQtT代理程式。
- * Confluent安全外掛程式。* Confluent安全外掛程式可用來新增各種Confluent Platform工具和產品的安全功能。目前有一個外掛程式可供Conflent REST Proxy使用、可協助驗證傳入要求、並將驗證的主體傳播至向Kafka的要求。這可讓Conflent REST Proxy用戶端利用Kafka代理程式的多租戶安全功能。

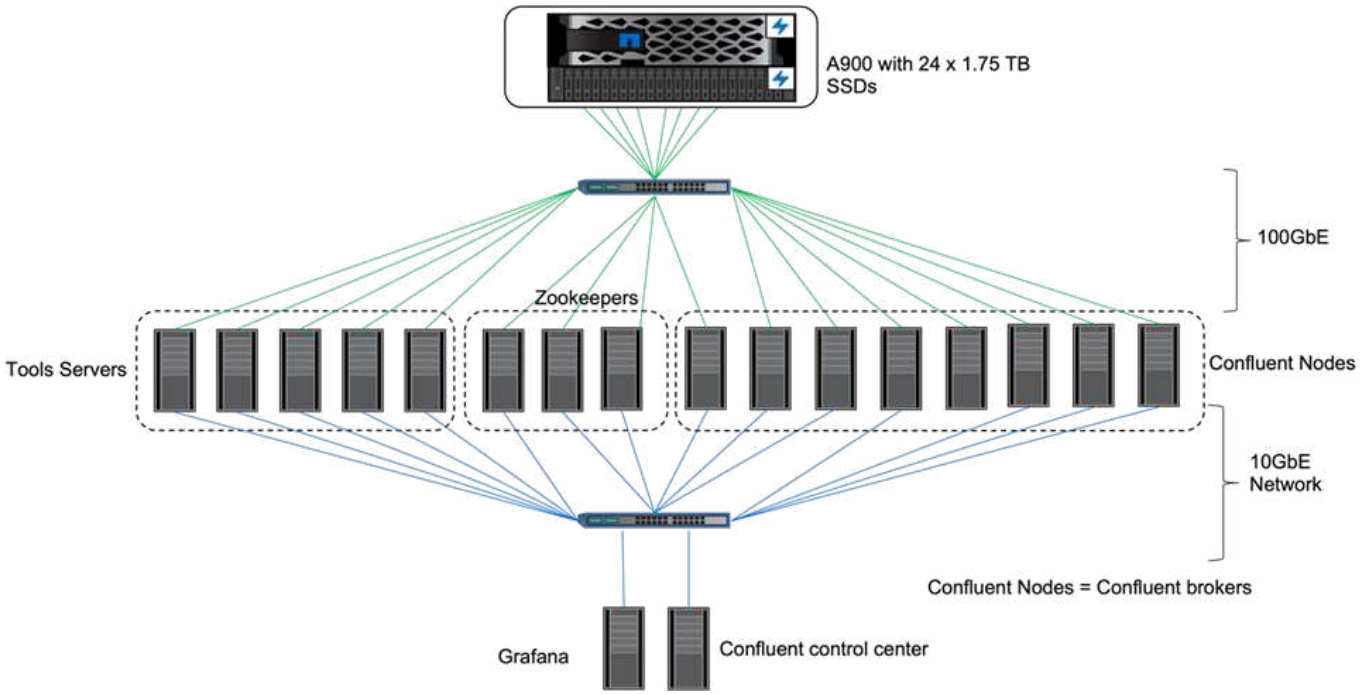
一致效能驗證

我們已使用Conflent Platform驗證NetApp ONTAP 的階層式儲存設備。NetApp與Conflent團隊共同進行這項驗證、並執行所需的測試案例。

簡易設定

在設定方面、我們使用三個zookeepers、五個代理商、以及五個測試伺服器、搭配256GB RAM和16個CPU。對於NetApp儲存設備、我們使用ONTAP 了一套AFF 用作Are4A900 HA配對的功能。儲存設備與代理商均透過100GbE連線進行連線。

下圖顯示用於階層式儲存驗證的組態網路拓撲。



這些工具伺服器可做為應用程式用戶端、用來傳送或接收來自ConFluent節點的事件。

一致的階層式儲存組態

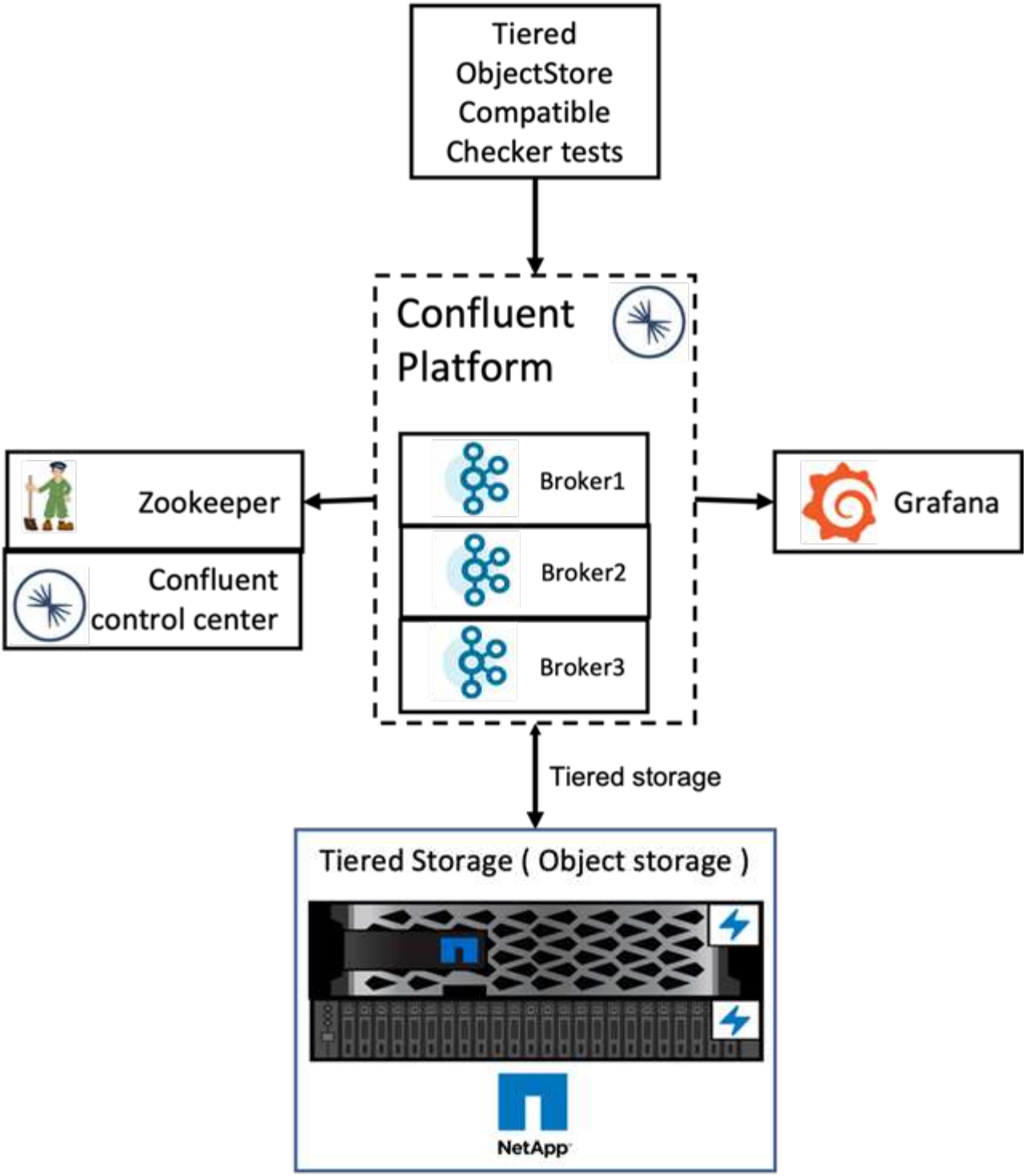
我們使用下列測試參數：

```
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
```

為了驗ONTAP 證、我們使用了搭配HTTP傳輸協定的功能、但HTTPS也能正常運作。存取金鑰和秘密金鑰會儲存在「confine.Tier .s3.cred.file.path」參數中提供的檔案名稱中。

NetApp儲存控制器ONTAP –功能

我們在ONTAP 不驗證的情況下、設定了單一HA配對組態。



驗證結果

我們完成下列五個驗證測試案例。前兩項是功能測試、其餘三項是效能測試。

物件存放區正確性測試

此測試會在使用API呼叫的階層式儲存區所使用的物件存放區上執行基本作業、例如Get、PUT和DELETE。

分層功能正確性測試

此測試會檢查物件儲存設備的端點對端點功能。它會建立主題、產生新建立主題的事件串流、等待代理程式將區段歸檔至物件儲存設備、使用事件串流、並驗證所使用的串流是否符合所產生的串流。我們已執行這項測試、且不需要進行物件存放區故障注入。我們在ONTAP 其中一個節點停止服務管理程式服務、並驗證端點對端點功能是否可與物件儲存搭配運作、藉此模擬節點故障。

階層擷取基準測試

此測試可驗證階層式物件儲存設備的讀取效能、並在基準測試產生的區段負載過重時、檢查範圍擷取讀取要求。在這個基準測試中、Connent開發了自訂用戶端、以滿足層級擷取要求。

產生耗用的工作負載產生器

此測試會透過區段歸檔、間接在物件存放區上產生寫入工作負載。讀取工作負載（區段讀取）是在使用者群組擷取區段時、從物件儲存區產生的。此工作負載是由TOCC指令碼產生。此測試會檢查平行執行緒中物件儲存設備的讀取和寫入效能。我們在進行分層功能正確性測試時、測試了是否有物件存放區故障注入。

保留工作負載產生器

這項測試檢查了在繁重的主題保留工作負載下、物件儲存設備的刪除效能。保留工作負載是使用TOCC指令碼產生、該指令碼會產生許多訊息、並與測試主題平行。測試主題是以積極的大小型和時間型保留設定來設定、導致事件串流持續從物件存放區中清除。然後將區段歸檔。這導致代理程式在物件儲存區中進行許多刪除、並收集物件儲存區刪除作業的效能。

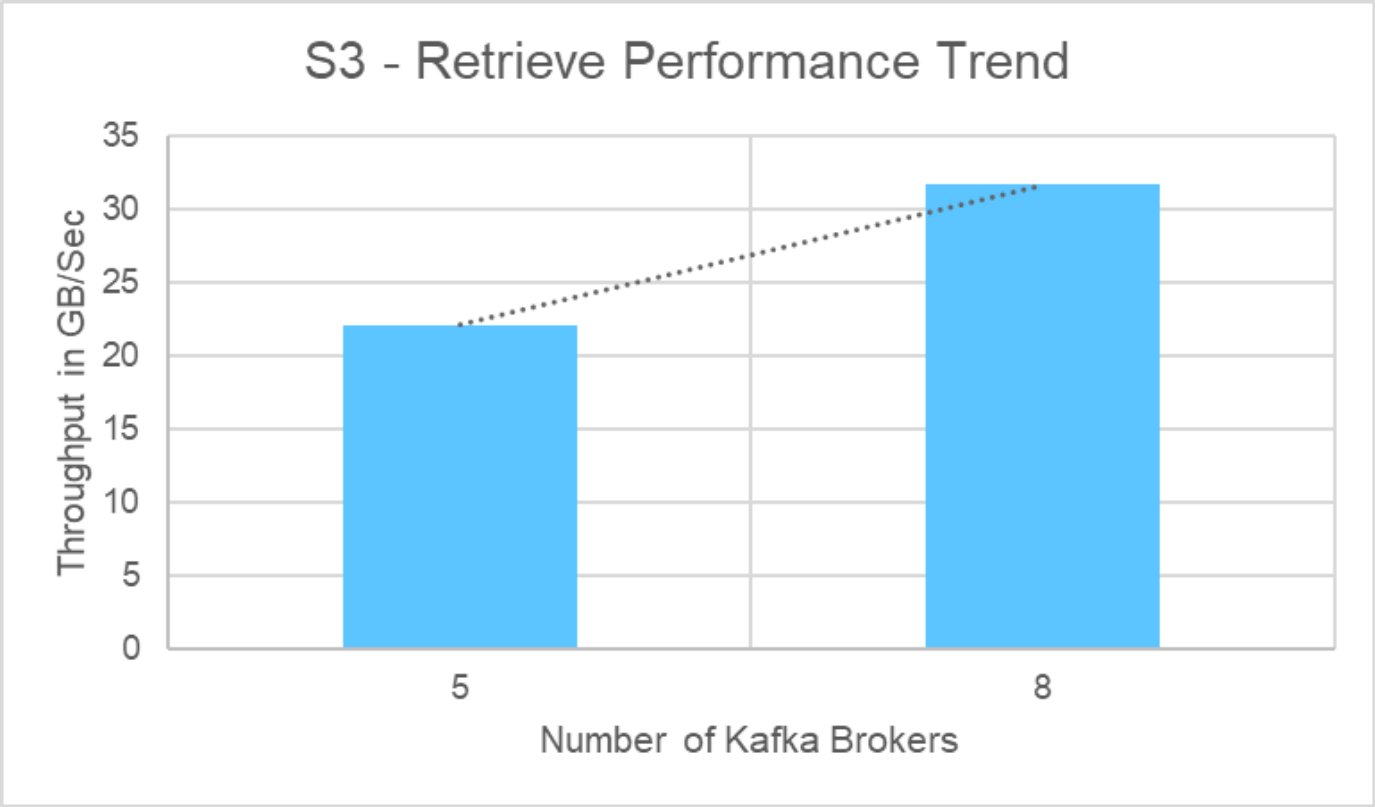
如需驗證詳細資料、請參閱 "Confluent" 網站。

使用生產消耗型工作負載產生器進行效能測試

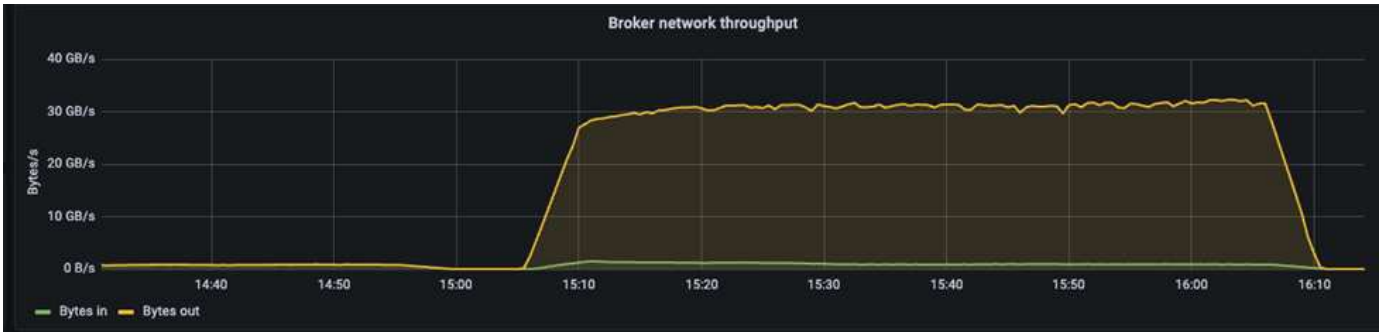
我們使用單AFF 一的NetApp儲存控制器、在生產消費型工作負載期間、使用五或八個代理節點執行階層式儲存測試。根據我們的測試、完成時間和效能結果會隨著代理節點數量而調整、直到AFF 資源使用率達到100 %為止。此功能需要至少一對HA配對才能完成。ONTAP

S3擷取作業的效能會根據ConFluent Broker節點的數量線性提升。在單一部署中、支援最多12個HA配對。ONTAP

下圖顯示結合了S3分層流量與五或八個代理節點。我們將AFF 單一HA配對效能發揮到極致。



下圖顯示卡夫卡的處理量約為31.74 GBps。



我們也在ONTAP 《perfstat》報告中看到類似的處理量。

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

效能最佳實務準則

本頁說明改善本解決方案效能的最佳實務做法。

- 若可行、請使用「Get size >=1MB（取得大小>=1MB）」ONTAP 。
- 在代理節點上的「erver.properties`」中增加「num.network.threads`」和「nm.io.thread」、可讓您將增加的分層活動推送到S3層。這些結果是「num.network.threads`」和「nm.io.thread」設為32。
- S3儲存區應鎖定每個成員集合體的八個成員。

- 推動S3流量的乙太網路連結在儲存設備和用戶端上應使用9k的MTU。

結論

這項驗證測試在運用NetApp ONTAP 知識儲存控制器的Conflent上達到31.74 GBps的分層處理量。

何處可找到其他資訊

若要深入瞭解本文所述資訊、請檢閱下列文件和 / 或網站：

- 什麼是Conflent？

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

- S3-sink參數詳細資料

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

- Apache Kafka

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

- S3的ONTAP 最佳實務做法

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

- S3物件儲存管理

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

- NetApp 產品文件

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

適用於**Apache Spark**的**NetApp**儲存解決方案

TR-4570：適用於**Apache Spark**的**NetApp**儲存解決方案：架構、使用案例及效能結果

Rick Huang、Karthithkeyan Nagalingam、NetApp

本文著重於Apache Spark架構、客戶使用案例、以及與Big Data分析和人工智慧（AI）相關的NetApp儲存產品組合。此外、它也會針對典型的Hadoop系統、使用業界標準AI、機器學習（ML）和深度學習（DL）工具、提供各種測試結果、讓您選擇適當的Spark解決方案。首先、您需要Spark架構、適當的元件、以及兩種部署模式（叢集與用戶端）。

本文件也提供客戶處理組態問題的使用案例、並討論NetApp儲存產品組合的總覽、其中與Big Data分析和AI、ML及DL with Spark有關。最後、我們將從特定於Spark的使用案例和NetApp Spark解決方案產品組合中取得測試結果。

客戶挑戰

本節著重於客戶在零售業、數位行銷、銀行、分離式製造、製程製造、 政府與專業服務。

無法預測的效能

傳統的Hadoop部署通常使用市售硬體。若要改善效能、您必須調整網路、作業系統、Hadoop叢集、Spark等生態系統元件和硬體。即使調校每個層級、也很難達到理想的效能等級、因為Hadoop是在非專為環境中的高效能所設計的市售硬體上執行。

媒體和節點故障

即使在正常情況下、市售硬體也容易故障。如果資料節點上的某個磁碟故障、Hadoop Master預設會將該節點視為不正常。然後、它會透過網路將特定資料從該節點複本複製到正常節點。此程序會減慢任何Hadoop工作的網路封包速度。然後叢集必須再次複製資料、並在不正常節點恢復正常狀態時移除過度複寫的資料。

Hadoop固定廠商

Hadoop經銷商擁有自己的Hadoop發佈版本、並將客戶鎖定在這些發佈版本上。不過、許多客戶需要記憶體內分析支援、而這並不會將客戶綁定到特定的Hadoop發佈。他們需要自由變更發佈、同時仍能隨之提供分析資料。

不支援一種以上的語言

除了MapReduce Java程式之外、客戶通常還需要支援多種語言才能執行工作。SQL和指令碼等選項可讓您更靈活地取得答案、更多組織和擷取資料的選項、以及更快將資料移至分析架構的方法。

難以使用

有一段時間、人們抱怨Hadoop難以使用。雖然Hadoop在每個新版本中都變得更簡單、更強大、但這項評論仍持續不變。Hadoop需要您瞭解Java和MapReduce程式設計模式、這是資料庫管理員和具有傳統指令碼技術集的人員所面臨的挑戰。

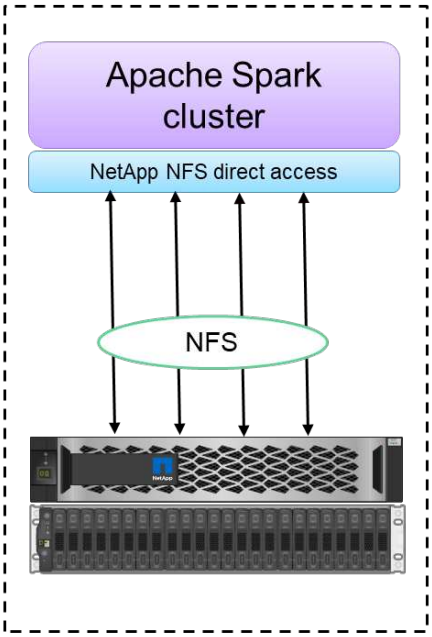
複雜的架構與工具

企業AI團隊面臨多項挑戰。即使擁有專家級的資料科學知識、不同部署生態系統和應用程式的工具和架構也可能無法簡單地相互翻譯。資料科學平台應能與以Spark為基礎的對應巨量資料平台無縫整合、輕鬆移動資料、可重複使用的模型、隨裝即用的程式碼、以及支援最佳實務做法的工具、以便進行原型製作、驗證、版本管理、共用、重複使用、 並將模型快速部署至正式作業環境。

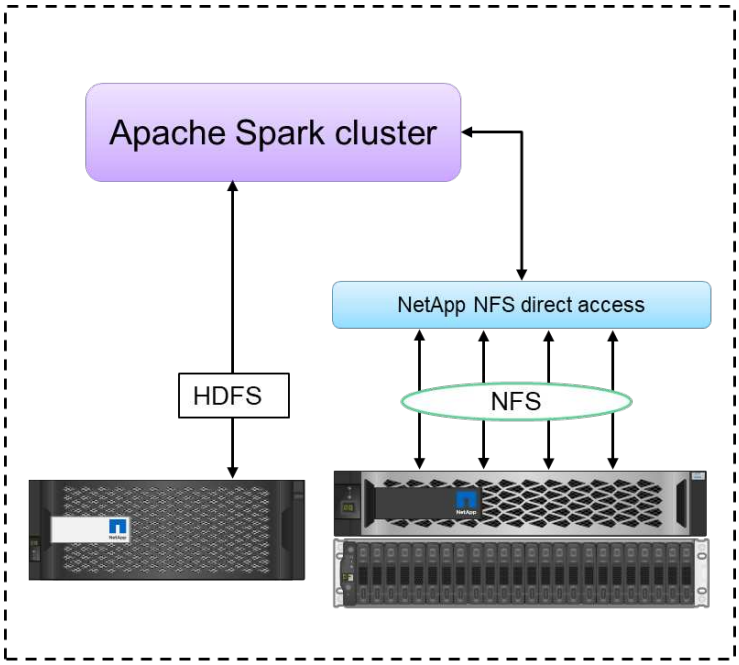
為何選擇**NetApp**？

NetApp可透過下列方式改善您的Spark體驗：

- NetApp NFS直接存取（如下圖所示）可讓客戶在現有或新的NFSv3或NFSv4資料上執行巨量資料分析工作、而無需移動或複製資料。它可防止多個資料複本、並免除將資料與來源同步的需求。
- 儲存效率更高、伺服器複寫更少。例如、NetApp E系列Hadoop解決方案需要兩個而非三個複本的資料、FAS 而《支援重複資料》解決方案則需要資料來源、但不需要複寫或複製資料。NetApp儲存解決方案也能減少伺服器對伺服器的流量。
- 在磁碟機和節點故障期間、執行更好的Hadoop工作和叢集行為。
- 更優異的資料擷取效能。



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

例如、在金融與醫療產業中、資料從一地搬移到另一地、必須符合法律義務、這並非易事。在此案例中、NetApp NFS直接存取會從原始位置分析財務與醫療資料。另一項主要優點是、使用NetApp NFS直接存取功能、使用原生Hadoop命令簡化Hadoop資料的保護、並透過NetApp豐富的資料管理產品組合來啟用資料保護工作流程。

NetApp NFS直接存取可為Hadoop / Spark叢集提供兩種部署選項：

- 根據預設、Hadoop或Spark叢集會使用Hadoop分散式檔案系統（HDFS）進行資料儲存、並使用預設的檔案系統。NetApp NFS直接存取可將預設的HDFS取代為預設的NFS儲存系統、以便直接分析NFS資料。
- 在另一個部署選項中、NetApp NFS直接存取可在單一Hadoop或Spark叢集中、將NFS設定為額外的儲存設備、以及HDFS。在這種情況下、客戶可以透過NFS匯出來共享資料、並從同一個叢集與HDFS資料一起存取。

使用NetApp NFS直接存取的主要效益包括：

- 從目前位置分析資料、避免將分析資料移至Hadoop基礎架構（例如HDFS）所需的時間與效能。
- 將複本數量從三個減少為一個。
- 讓使用者能夠分離運算與儲存設備、以獨立擴充。
- 運用ONTAP 豐富的資料管理功能來提供企業資料保護功能。
- Hortonworks資料平台認證。
- 實現混合式資料分析部署。
- 運用動態多執行緒功能、縮短備份時間。

請參閱 "TR-4657：NetApp混合雲資料解決方案：根據客戶使用案例而設計的Spark和Hadoop" 將Hadoop資料、備份及災難恢復從雲端備份至內部部署、讓DevTest能夠處理現有的Hadoop資料、資料保護及多雲端連線、並加速分析工作負載。

以下各節說明適用於Spark客戶的儲存功能。

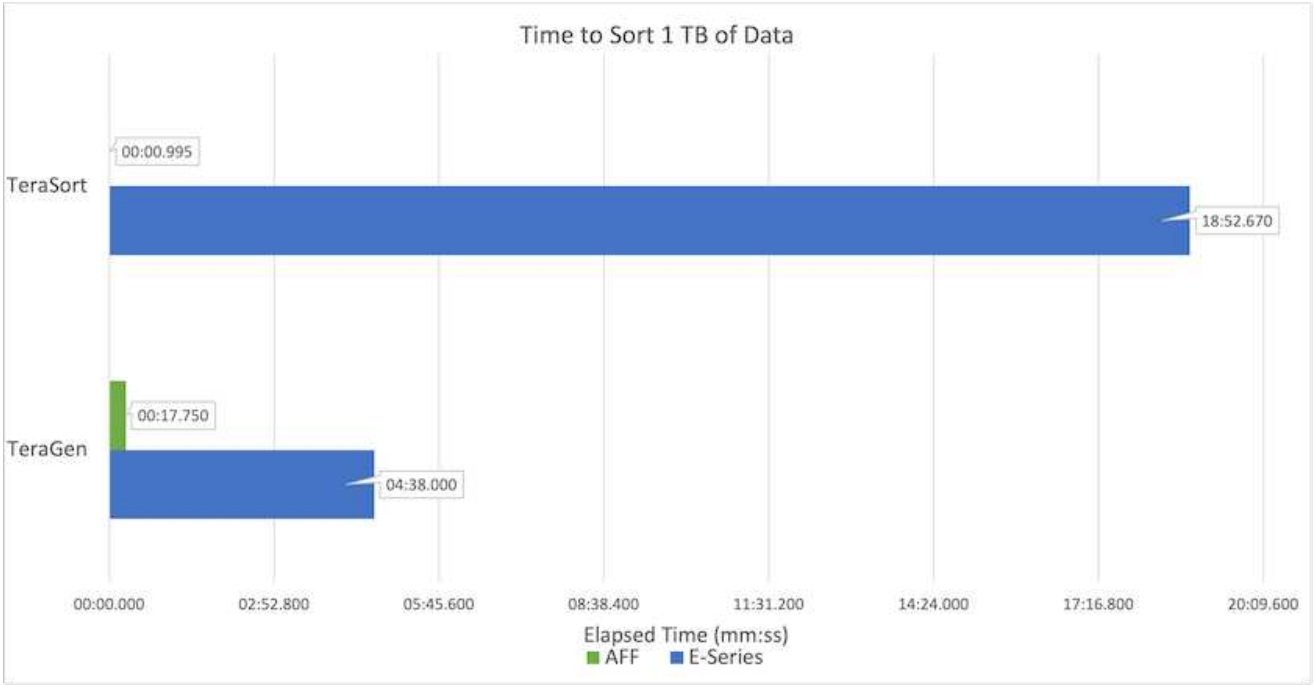
儲存分層

透過Hadoop儲存分層、您可以根據儲存原則、以不同的儲存類型來儲存檔案。儲存類型包括「Hot」、「Cold」、「warm」、「all_ssd」、「one _ssd」、 和"lazy_sent"。

<<<<<<<<部門主管我們在NetApp AFF 支援的儲存控制器和E系列儲存控制器上執行Hadoop儲存分層驗證、其中採用不同儲存原則的SSD和SAS磁碟機。配備AFF-A800的Spark叢集有四個運算工作者節點、而採用E系列的叢集則有八個節點。這主要是比較固態硬碟（SSD）與硬碟（HDD）的效能。

我們在NetApp AFF 支援的儲存控制器和E系列儲存控制器上執行Hadoop儲存分層驗證、其中的SSD和SAS磁碟機具有不同的儲存原則。配備AFF-A800的Spark叢集有四個運算工作者節點、而採用E系列的叢集則有八個節點。我們主要是為了比較固態硬碟與硬碟的效能。>>>a51c9ddf73ca69e1120ce05edc7b0b9607b96eae

下圖顯示適用於Hadoop SSD的NetApp解決方案效能。



- 基礎NL-SAS組態使用八個運算節點和96個NL-SAS磁碟機。此組態可在4分鐘38秒內產生1TB的資料。請參閱 "TR-3969適用於Hadoop的NetApp E系列解決方案" 以取得叢集與儲存組態的詳細資料。
- 使用TeraGen時、SSD組態產生的資料速度比NL-SAS組態快上1TB 15.66倍。此外、SSD組態使用一半的運算節點、一半的磁碟機（總共24個SSD磁碟機）。根據工作完成時間、速度幾乎是NL-SAS組態的兩倍。
- 使用TeraSort時、SSD組態的資料排序速度比NL-SAS組態快1TB（1138.36）。此外、SSD組態使用一半的運算節點、一半的磁碟機（總共24個SSD磁碟機）。因此、每個磁碟機的速度約比NL-SAS組態快三倍。<<<<<<<<標題
- 現在、我們正從旋轉式磁碟移轉至All Flash、以提升效能。運算節點的數量並不是瓶頸。有了NetApp的All Flash儲存設備、執行時間效能可大幅擴充。
- 有了NFS、資料在功能上等同於全部集合在一起、因此可根據您的工作負載減少運算節點的數量。在變更運算節點數量時、Apache Spark叢集使用者不需要手動重新平衡資料。

- 總而言之、從旋轉式磁碟移轉至All Flash可改善效能。運算節點的數量並不是瓶頸。有了NetApp All Flash儲存設備、執行時間效能可大幅擴充。
- 有了NFS、資料在功能上等同於全部集合在一起、因此可根據您的工作負載減少運算節點的數量。在變更運算節點數量時、Apache Spark叢集使用者不需要手動重新平衡資料。>>>a51c9ddf73ca69e1120ce05edc7b0b9607b96eae

效能擴充：橫向擴充

當您需要AFF 更多運算能力來自於支援各種解決方案的Hadoop叢集時、可以使用適當數量的儲存控制器來新增資料節點。NetApp建議從每個儲存控制器陣列的四個資料節點開始、並根據工作負載特性、將每個儲存控制器的資料節點數目增加至八個。

適用於就地分析的不只是指不需使用的資料。AFF FAS根據運算需求、您可以新增節點管理程式、而不中斷營運則可讓您在不需停機的情況下隨需新增儲存控制器。我們提供AFF 豐富的功能與功能、例如FAS NVMe媒體支援、保證效率、資料減量、QoS、預測分析、 雲端分層、複寫、雲端部署及安全性。為了協助客戶滿足其需求、NetApp提供檔案系統分析、配額及隨裝負載平衡等功能、無需額外的授權成本。NetApp在並行工作數量、延遲時間、作業簡化、以及每秒GB處理量方面的效能優於競爭對手。此外、NetApp Cloud Volumes ONTAP 的功能可在所有三家主要雲端供應商上執行。

效能擴充-垂直擴充

垂直擴充功能可讓您在AFF 需要額外儲存容量時、將磁碟機新增至效益管理系統、FAS 效益管理系統及E系列系統。利用功能、將儲存設備擴充至PB層級是兩大因素的組合：將不常用的資料分層、從區塊儲存設備物件儲存、以及堆疊不需額外運算的功能。Cloud Volumes ONTAP Cloud Volumes ONTAP

多種傳輸協定

NetApp系統支援大部分的Hadoop部署傳輸協定、包括SAS、iSCSI、FCP、InfiniBand、 和NFS。

營運與支援的解決方案

NetApp支援本文件中所述的Hadoop解決方案。這些解決方案也通過主要Hadoop經銷商的認證。如需相關資訊、請參閱 "MapR" 網站 "Hortonworks" 網站和Cloudera "認證" 和 "合作夥伴" 網站。

目標對象

分析與資料科學的世界涉及IT與企業的多個領域：

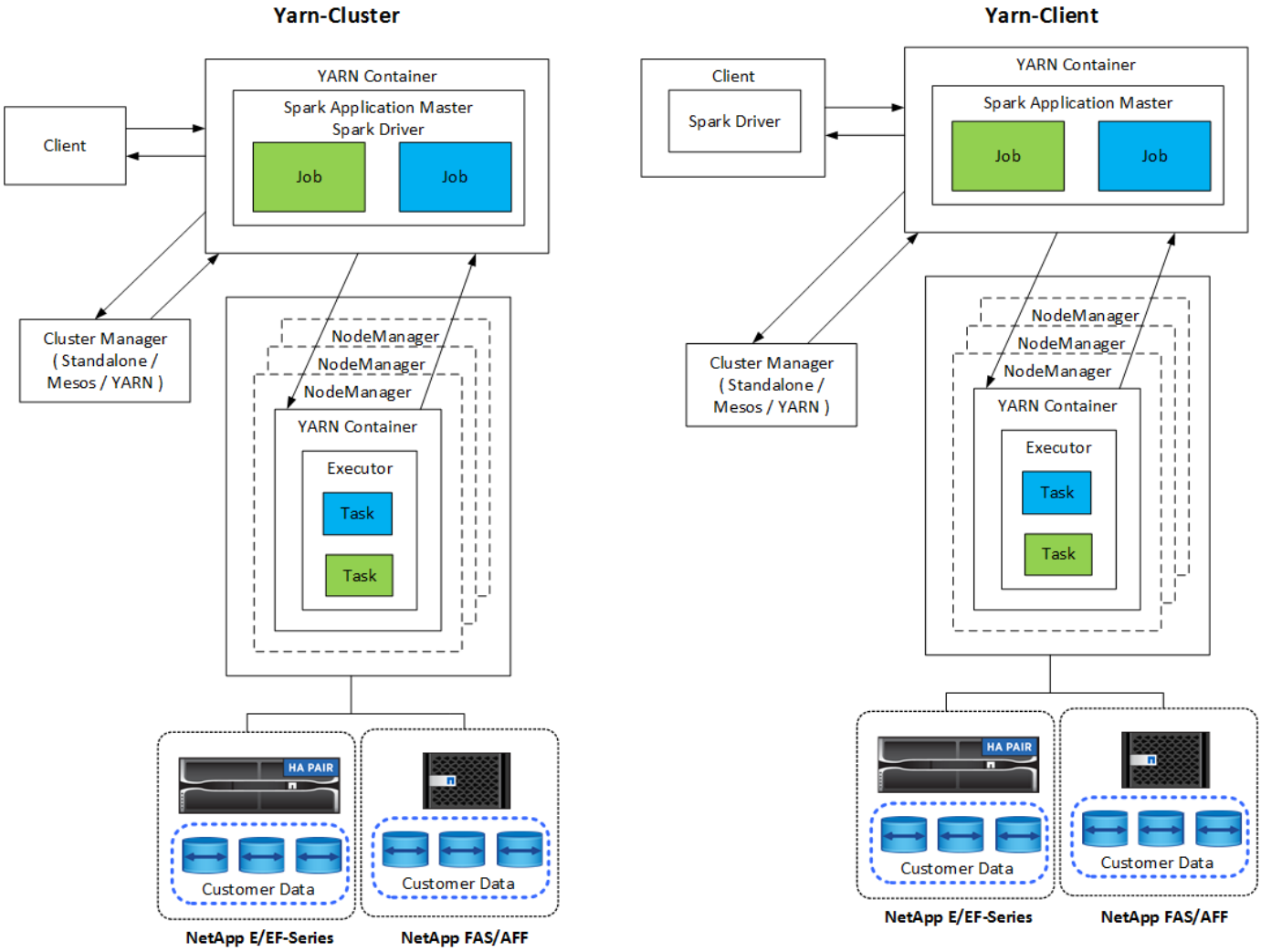
- 資料科學家需要靈活運用所選的工具和程式庫。
- 資料工程師需要知道資料的流通方式及存放位置。
- DevOps工程師需要工具、將新的AI和ML應用程式整合至其CI和CD管道。
- 雲端管理員和架構設計師必須能夠設定和管理混合雲資源。
- 企業使用者想要存取分析、AI、ML和DL應用程式。

在本技術報告中、我們將說明NetApp AFF 的功能、E系列、StorageGRID 支援、NFS直接存取、Apache Spark、 Horovod和Keras協助這些職務為企業帶來價值。

解決方案技術

Apache Spark是一種熱門的程式設計架構、可用來撰寫直接搭配Hadoop分散式檔案系統（HDFS）運作的Hadoop應用程式。Spark已準備就緒、支援串流資料處理、速度比MapReduce快。Spark可設定的記憶體內資料快取功能、可有效進行迭代、而Spark Shell則是互動式的、可用來學習及探索資料。有了Spark、您可以在Python、Scala或Java中建立應用程式。SPARK應用程式由一或多個工作組成、這些工作具有一或多項工作。

每個Spark應用程式都有Spark驅動程式。在線對用戶端模式中、驅動程式會在用戶端本機上執行。在線叢集模式中、驅動程式會在應用程式主機的叢集內執行。在叢集模式中、即使用戶端中斷連線、應用程式仍會繼續執行。



有三種叢集管理程式：

- *獨立式。*此管理程式是Spark的一部分、可讓您輕鬆設定叢集。
- * Apache Mesos.*這是一種通用叢集管理程式、可執行MapReduce及其他應用程式。
- *《哈多奧》*這是Hadoop 3的資源經理。

彈性分散式資料集（RDD）是Spark的主要元件。RDD會重新建立儲存在叢集記憶體中的資料遺失與遺失資料、並儲存來自檔案或以程式設計方式建立的初始資料。RDD是從檔案、記憶體中的資料或其他RDD所建立。Spark

程式設計可執行兩項作業：轉型與行動。轉換會根據現有的RDD建立新的RDD。動作會傳回RDD的值。

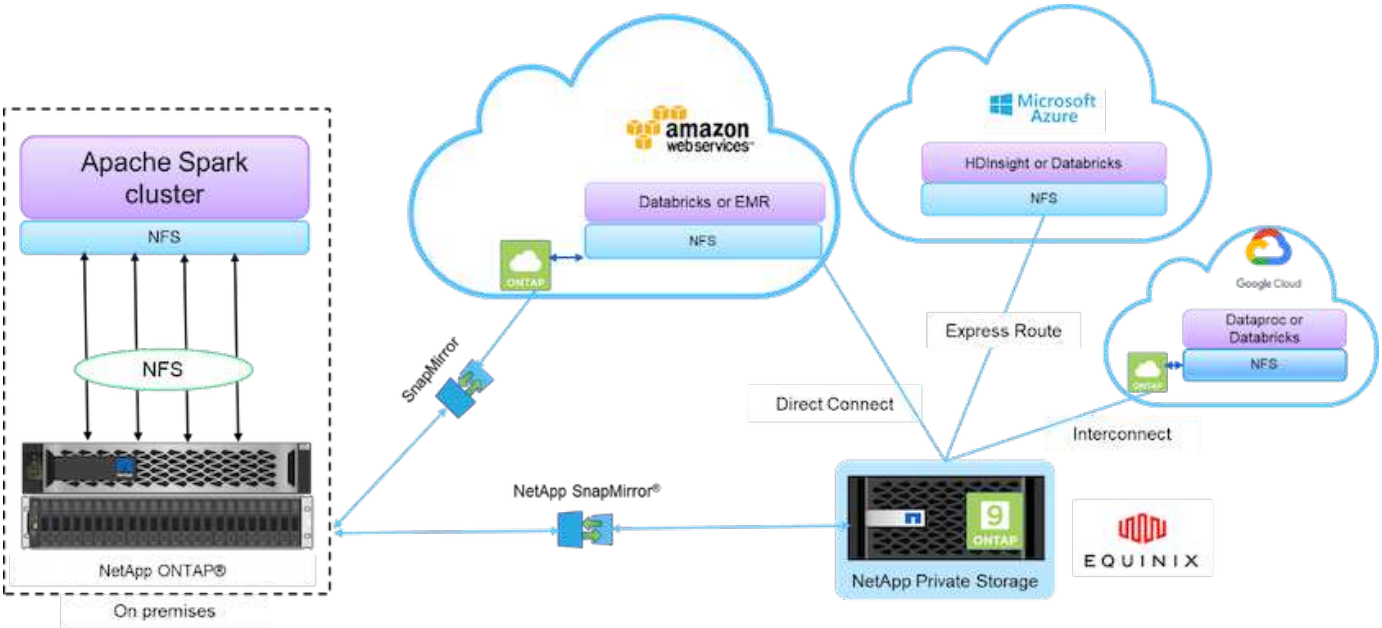
轉換與動作也適用於Spark資料集和DataFrames。資料集是分散式資料集合、提供RDD的優點（強式輸入、使用Lambda功能）、以及Spark SQL最佳化執行引擎的優點。資料集可從JVM物件建構、然後使用功能性轉換（地圖、扁平對應、篩選等）進行處理。DataFrame是一種資料集、組織成命名欄。它在概念上等同於關聯式資料庫中的資料表、或是R/Python中的資料框架。DataFrames可從各種來源建構、例如結構化資料檔案、Hive/HBase中的表格、內部部署或雲端中的外部資料庫、或現有的RDD。

Spark應用程式包括一或多個Spark工作。在執行者中執行工作、執行者則在紗櫃中執行工作。每個執行程式都會在單一容器中執行、而且執行程式會在應用程式的整個生命週期內存在。執行程式會在應用程式啟動後修正、而且實際不會調整已分配的容器大小。執行程式可在記憶體內的資料上同時執行工作。

NetApp Spark解決方案總覽

NetApp有三種儲存產品組合：FAS/AFF、E系列和Cloud Volumes ONTAP VMware。我們已驗證AFF 採用ONTAP NetApp技術的支援功能、以及採用NetApp技術的E系列、以利搭配Apache Spark的Hadoop解決方案。

採用NetApp技術的資料架構整合了資料管理服務與應用程式（建置區塊）、可用於資料存取、控制、保護及安全性、如下圖所示。

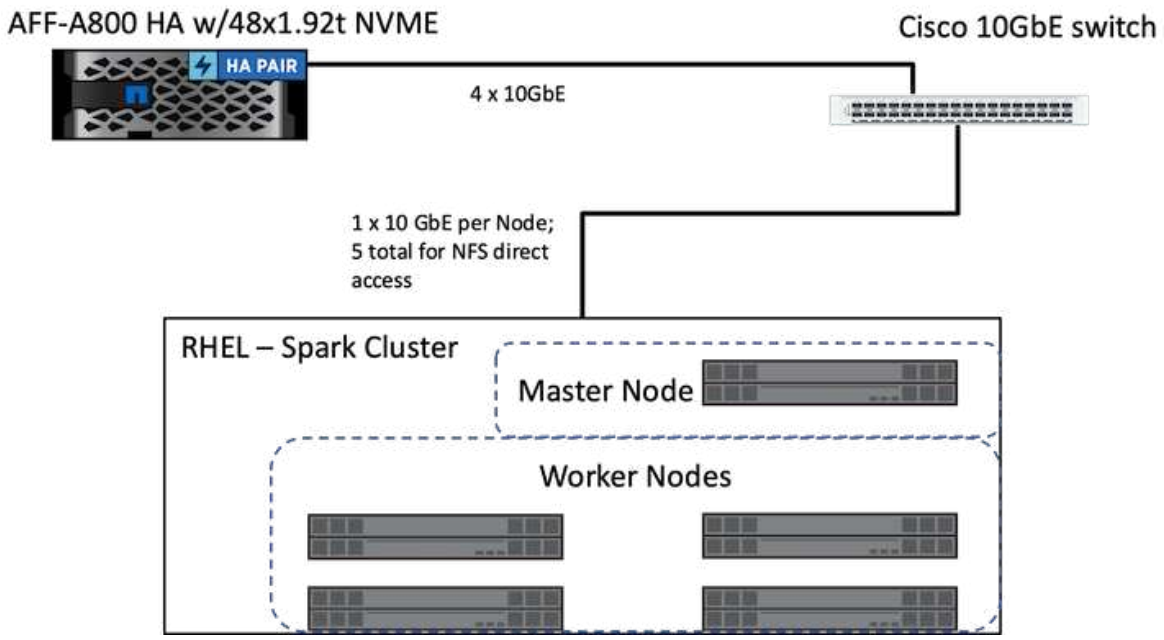


上圖中的建置區塊包括：

- * NetApp NFS直接存取。*提供最新的Hadoop和Spark叢集、可直接存取NetApp NFS磁碟區、無需額外的軟體或驅動程式需求。
- * NetApp Cloud Volumes ONTAP 功能與雲端Volume Services。*軟體定義的連線儲存設備、以ONTAP Microsoft Azure NetApp Files Azure雲端服務中Amazon Web Services（AWS）或Sf2（anf）執行的功能為基礎。
- * NetApp SnapMirror技術。*可在內部部署ONTAP 與支援內部部署的NetApp或NPS執行個體之間提供資料保護功能。
- *雲端服務供應商。*這些供應商包括AWS、Microsoft Azure、Google Cloud和IBM Cloud。

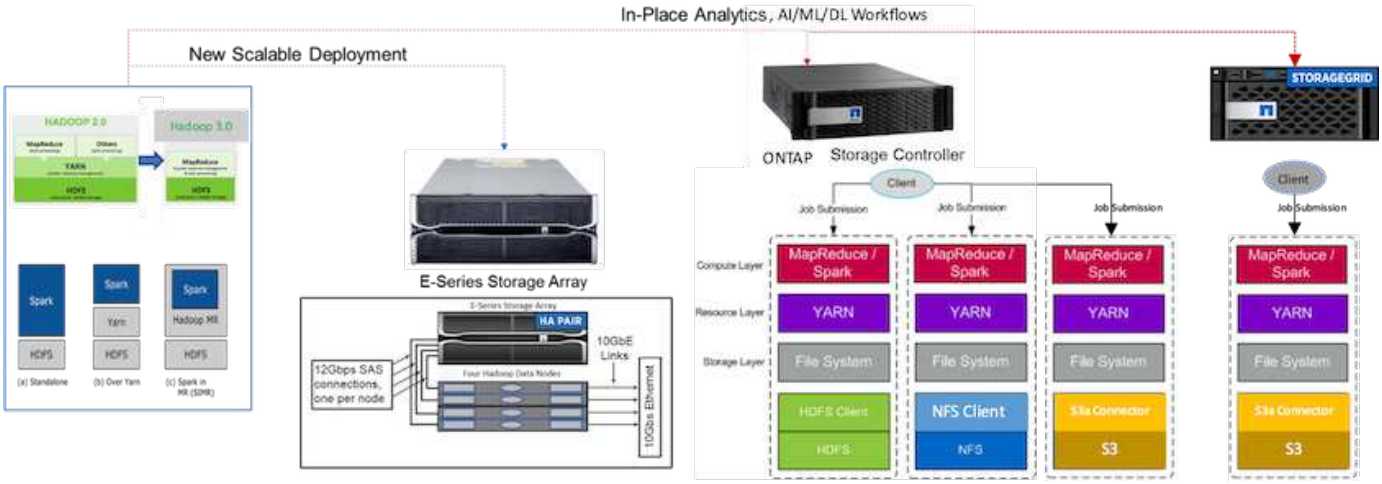
- * PaaS *雲端型分析服務、例如AWS中的Amazon Elastic MapReduce（EMR）和Databricks、以及Microsoft Azure HDInsight和Azure Databricks。

下圖說明採用NetApp儲存設備的Spark解決方案。

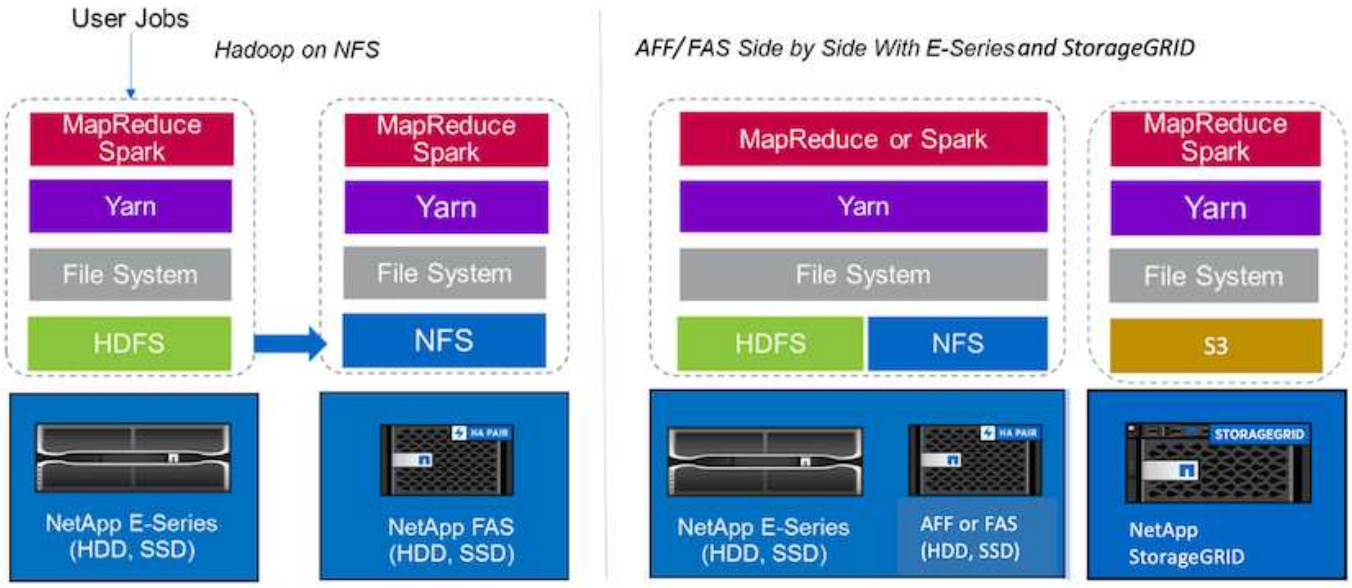


利用NetApp NFS直接存取傳輸協定、利用現有正式作業資料存取就地分析、以及AI、ML和DL工作流程。ONTAP可用於Hadoop節點的正式作業資料會匯出、以執行就地分析和AI、ML和DL工作。您可以透過NetApp NFS直接存取或不使用、存取Hadoop節點中的資料以進行處理。在Spark搭配獨立式或「線」叢集管理程式、您可以使用「」來設定NFS磁碟區<file:///<target_volume>`。我們已驗證三個使用案例、並使用不同的資料集。這些驗證的詳細資料請參閱「測試結果」一節。 （XEF）

下圖說明NetApp Apache Spark/Hadoop儲存設備定位。



我們找出E系列Spark解決方案的獨特功能、AFF/FAS ONTAP 電火花解決方案和StorageGRID VMware Spark解決方案、並執行詳細的驗證與測試。根據我們的觀察結果、NetApp建議採用E系列解決方案來進行全新的現場安裝和新的可擴充部署、以及AFF/FAS解決方案、以便使用現有NFS資料進行就地分析、AI、ML和DL工作負載、StorageGRID 並在需要物件儲存時、針對AI、ML和DL進行Sfor AI、以及現代化資料分析。



資料湖是以原生形式儲存大型資料集的儲存庫、可用於分析、AI、ML和DL工作。我們為E系列、AFF/FAS和StorageGRID VMware SG6060 Spark解決方案打造資料湖儲存庫。E系列系統可讓HDFS存取Hadoop Spark叢集、而現有的正式作業資料則是透過NFS直接存取傳輸協定存取Hadoop叢集。對於位於物件儲存設備中的資料集、NetApp StorageGRID 支援S3和S3a安全存取。

使用案例摘要

本頁說明可使用此解決方案的不同領域。

串流資料

Apache Spark可以處理串流資料、這些資料可用於串流擷取、轉換及負載（ETL）程序、資料豐富、觸發事件偵測、以及複雜的工作階段分析：

- *串流ETL.*資料會在推入資料存放區之前持續清除及彙總。Netflix使用Kafka和Spark串流來建置即時線上影片推薦和資料監控解決方案、每天都能從不同的資料來源處理數十億個事件。然而、用於批次處理的傳統ETL會以不同的方式處理。此資料會先讀取、然後轉換成資料庫格式、再寫入資料庫。
- 資料豐富。 Spark串流利用靜態資料豐富即時資料、實現更即時的資料分析。例如、線上廣告商可以根據客戶行為資訊、提供個人化的目標式廣告。
- 觸發事件偵測。 Spark串流可讓您偵測並快速回應可能指出潛在嚴重問題的異常行為。例如、金融機構使用觸發程序來偵測及停止詐欺交易、而醫院則使用觸發程序來偵測病患生命徵兆中偵測到的危險健康變化。
- 複雜的工作階段分析。 Spark串流會在登入網站或應用程式之後收集使用者活動等事件、然後加以分組和分析。例如、Netflix使用此功能來提供即時電影建議。

如需更多串流資料組態、Conflent Kafka驗證及效能測試、請參閱 "TR-4912：NetApp的Conflent Kafka階層式儲存設備最佳實務準則"。

機器學習

Spark整合式架構可協助您使用機器學習庫（MLLib）、在資料集上執行重複查詢。MLLib適用於叢集、分類及維度減量等常見的巨量資料功能、例如預測性情報、行銷目的的客戶區隔、以及情緒分析。MLLib用於網路安全性、可即時檢查資料封包、以利發現惡意活動。它可協助安全供應商瞭解新的威脅、並在即時保護客戶的同時、領先駭客。

深度學習

TensorFlow是業界廣受歡迎的深度學習架構。TensorFlow支援CPU或GPU叢集上的分散式訓練。這項分散式訓練可讓使用者在大量資料上執行、並有許多深度層級。

直到最近才剛開始、如果我們想將TensorFlow與Apache Spark搭配使用、我們需要在PySpark執行TensorFlow所需的所有ETL、然後將資料寫入中繼儲存設備。然後將該資料載入TensorFlow叢集、以進行實際的訓練程序。此工作流程需要使用者維護兩個不同的叢集、一個用於ETL、另一個用於TensorFlow的分散式訓練。執行及維護多個叢集通常很繁瑣且耗時。

早期Spark版本中的DataFrames和RDD不太適合深度學習、因為隨機存取有限。在Spark 3.0中搭配專案的打造、新增了對深度學習架構的原生支援。此方法允許在Spark叢集上進行非MapReduce型排程。

互動分析

Apache Spark的速度足以執行探索性查詢、無需使用除了Spark以外的開發語言進行取樣、包括SQL、R和Python。Spark使用視覺化工具來處理複雜的資料、並以互動方式視覺化。利用結構化串流來執行即時資料的互動查詢、透過網路分析功能、您可以針對網站訪客目前的工作階段執行互動式查詢。

推薦系統

多年來、建議系統已為我們的生活帶來巨大改變、因為企業和消費者已因應線上購物、線上娛樂和許多其他產業的劇烈變化。事實上、這些系統是AI在正式作業中最明顯的成功案例之一。在許多實際使用案例中、建議系統會與對話式AI或聊天機器人結合、與NLP後端互動、以取得相關資訊並產生實用的推斷。

如今、許多零售商正在採用較新的商業模式、例如在線上購買和在店內取貨、現場取貨、自助結帳、掃描與外出等等。這些模式在COVID-19大流行病期間變得非常顯著、因為它讓購物變得更安全、也更方便消費者使用。AI對於這些不斷成長的數位趨勢至關重要、這些趨勢受到消費者行為的影響、反之亦然。為了滿足消費者不斷成長的需求、提升客戶體驗、提升營運效率、並增加營收、NetApp協助企業客戶和企業使用機器學習和深度學習演算法、以更快更準確的建議系統。

提供建議的熱門技術有幾種、包括協同作業篩選、內容型系統、深度學習推薦模式（DLRM）和混合式技術。客戶先前使用PySpark來實作協同篩選、以建立建議系統。Spark MLLib採用替代最小平方數（ALS）來進行協同篩選、這是在DLRM興起之前、企業中非常受歡迎的演算法。

自然語言處理

人工智慧是人工智慧的分支機構、可透過自然語言處理（NLP）實現、協助電腦與人類溝通。NLP在每個產業垂直市場和許多使用案例中都十分普及、從智慧型助理和聊天機器人、到Google搜尋和預測性文字。根據A "Gartner" 預測到2022年、70%的人會每天與對話式AI平台互動。若要在人與機器之間進行高品質的對話、回應必須是快速、智慧且自然的。

客戶需要大量資料來處理及訓練NLP和自動語音辨識（ASR）模式。他們也需要在邊緣、核心和雲端之間移動資料、而且需要在毫秒內執行推斷、才能與人類建立自然的通訊。NetApp AI與Apache Spark是運算、儲存、資料處理、模型訓練、微調、 和部署。

情緒分析是NLP內部的研究領域、從文字中擷取正面、負面或中立的情緒。情緒分析有多種使用案例、從判斷支援中心員工在與來電者對話時的表現、到提供適當的自動聊天機器人回應。此外、它也可用來根據公司代表與每季營收拜訪對象之間的互動、預測公司的股票價格。此外、情緒分析可用來判斷客戶對於品牌所提供產品、服務或支援的看法。

我們使用 "Spark NLP" 程式庫來源 "John Snow Labs" 從Transformers（Bert）機型載入預先訓練的管線和雙向編碼器呈現、包括 "財務新聞意欲" 和 "FinBit"、大規模地執行令牌化、命名實體辨識、模型訓練、擬合和情緒分

析。Spark NLP是市面上唯一提供Bert、偉業、Elecra、XLNet、DisbilBit4等先進變壓器的開放原始碼NLP程式庫。 羅伯塔、DEBerta、XLM-羅伯塔、龍原、Elmo、 通用句子編碼器、Google T5、MarianMT和GPT2。此程式庫不僅可在Python和R中運作、也可在VM生態系統（Java、Scala和Kotlin）中大規模運作、只要將Apache Spark原生擴充即可。

主要AI、ML和DL使用案例和架構

AI、ML和DL的主要使用案例和方法可分為下列各節：

Spark NLP管道和TensorFlow分散式推斷

下列清單包含資料科學社群在不同開發層級採用的最受歡迎開放原始碼NLP程式庫：

- "自然語言工具套件（NLTK）"。所有NLP技術的完整工具套件。自2000年代初期起便一直維持這項功能。
- "文字Blob"。簡單易用的NLP工具Python API、建置於NLTK和Pattern之上。
- "史丹佛核心NLP"。由斯坦福NLP Group開發的Java NLP服務與套件。
- "Gensim"。「人類主題建模」從捷克數位數學庫專案的Python指令碼集合開始。
- "空間大"。端點對端點工業NLP工作流程、搭配Python和Cython、並可加速變壓器的GPU。
- "Fasttext"。免費、輕量化的開放原始碼NLP程式庫、適用於Facebook AI Research（Fair）實驗室所建立的詞彙學習文案與句子分類。

Spark NLP是單一的統一化解決方案、適用於所有NLP工作與需求、可針對實際的正式作業使用案例、提供可擴充、高效能及高準確度的NLP軟體。它運用移轉學習、並在研究領域和跨產業中實作最新的最先進演算法和模型。由於Spark對上述程式庫缺乏完整支援、因此Spark NLP建置於上方 "Spark ML" 善用Spark一般用途的記憶體內分散式資料處理引擎、做為企業級NLP程式庫、用於關鍵任務正式作業工作流程。其註釋工具運用規則型演算法、機器學習和TensorFlow來推動深度學習實作。這涵蓋常見的NLP工作、包括但不限於令牌化、模組化、產生、部分語音標記、具名實體辨識、 拼字檢查及情緒分析。

Transformers（Bert）的雙向編碼器表示法是NLP的一種以變壓器為基礎的機器學習技術。它推廣了預先訓練和微調的概念。Bert的變壓器架構源自機器翻譯、這種機器的長期相依性模型比經常性的神經網路（RNN）型語言模型更好。此外、它也引進了「遮罩語言模型（MLM）」工作、其中隨機遮罩了15%的所有權杖、模型會預測這些權杖、實現真正的雙向性。

由於該網域的專業語言和缺乏標記資料、因此財務情緒分析是一項挑戰。 "FinBit"是一種以訓練前的Bert為基礎的語言模式、已在網域上進行調整 "Reuters TRC2"、財務資料集、並以標記資料進行微調（ "金融市場銀行" ）。研究人員以財務術語從新聞文章中擷取4、500個句子。接著有16位具有財務背景的專家和碩士學生將句子標示為正面、中立和負面。我們建立了端點對端點的Spark工作流程、以分析2016至2020年前十大NASDAQ公司的營收客服記錄、使用FinBitt和其他兩個預先訓練的管道（.. "金融新聞的情緒分析"、 "說明DL文件"）來自Spark NLP。

適用於Spark NLP的基礎深度學習引擎是TensorFlow、這是一套端點對端點開放原始碼平台、可用於機器學習、可輕鬆建構模型、隨處進行強大的ML正式作業、以及進行強大的研究實驗。因此、在Spark「線叢集」模式下執行管線時、我們基本上是在叢集上執行分散式TensorFlow、並在單一主節點和多個工作節點之間執行資料和模型平行化、以及網路附加儲存設備。

Horovod分散式訓練

MapReduce相關效能的核心Hadoop驗證是透過TeraGen、TeraSort、TeraValidate和DFSIO（讀寫）來執行。TeraGen和TeraSort驗證結果會顯示在中 "TR-3969：NetApp Hadoop解決方案" 適用於E系列、並在「Storage Tiering」（xref）一節AFF 中介紹

根據客戶的要求、我們認為透過Spark進行的分散式訓練是各種使用案例中最重要的一項。在本文中、我們使用了 "Hovorod on Spark" 使用NetApp All Flash FAS （AFF VMware）儲存控制器Azure NetApp Files、VMware®及StorageGRID VMware®來驗證NetApp內部部署、雲端原生及混合雲解決方案的Spark效能。

Horovod on Spark套件提供了一款便利的包裝函式、可讓您輕鬆在Spark叢集中執行分散式訓練工作負載、實現緊密的模型設計迴圈、讓訓練和推斷資料所在的Spark執行資料處理、模型訓練和模型評估。

在Spark上執行Horovod的API有兩種：高階Estimator API和較低層級的Run API。儘管兩者都使用相同的基礎機制、在Spark執行程式上啟動Horovod、但Estimator API會將資料處理、模型訓練迴圈、模型檢查點、指標收集和分散式訓練擷取出來。我們使用Horovod Spark Estimators、TensorFlow和Keras來進行端點對端點資料準備、並根據進行分散式訓練工作流程 "Kaggle Rossmann商店銷售" 競爭。

指令碼「keras」（keras）-「SPARK」（火花）-「horovod_rossmann_redimator.py」（組態）可在一節中找到 "每個主要使用案例的Python指令碼。" 其中包含三個部分：

- 第一部分是針對Kaggle提供並由社群收集的一組CSV檔案、執行各種資料預先處理步驟。輸入資料會分成「驗證」子集和測試資料集的訓練集。
- 第二部分定義Keras Deep Neural Network（DNN）模型、其中包含對數型sigmoid啟動功能和Adam最佳化程式、並使用Spark的Horovod來執行模型的分散式訓練。
- 第三部分會使用最佳模式、對測試資料集執行預測、將驗證集整體平均絕對錯誤減至最低。然後建立輸出CSV檔案。

請參閱一節 "「機器學習」" 以取得各種執行時間比較結果。

使用**Keras**進行**CTR**預測的多工深度學習

隨著ML平台與應用程式的最新進展、現在許多人都開始大規模學習。點閱率（CTR）定義為每百個線上廣告曝光點閱率（以百分比表示）的平均點閱率。在各種產業垂直市場和使用案例中、包括數位行銷、零售、電子商務和服務供應商、廣泛採用此技術作為主要指標。請參閱我們的 "TR-4904：Azure中的分散式訓練-點擊率預測" 如需更多有關CTR應用程式和Kubernetes端點對端點雲端AI工作流程實作、分散式資料ETL、以及使用dask和CUDA ML進行模型訓練的詳細資訊。

在本技術報告中、我們使用的是不同的 "請按一下「記錄資料集」" （請參閱TR-4904）針對使用Keras建置Deep and Cross Network（DCN）機型的Spark工作流程的多位員工分散式深度學習、比較其記錄遺失錯誤功能與基礎Spark ML物流回歸模式的效能。新一代的新一代管理系統可有效擷取限定度的有效功能互動、學習高度非線性互動、不需手動進行特徵工程或詳盡搜尋、而且運算成本低廉。

Web等級建議系統的資料大多是分離式且明確的、導致功能探索的空間大而稀少。這將大多數大型系統限制為線性模型、例如物流回歸。然而、找出經常預測的功能、同時探索未被發現或罕見的交叉功能、是做出良好預測的關鍵。線性模型簡單易用、可解釋且易於擴充、但其表現能力有限。

另一方面、交叉功能在改善模型表現方面表現顯著。可惜的是、通常需要手動進行功能工程或詳盡搜尋、才能識別這些功能。一般而言、難以與看不到的功能互動。使用像是DCN這樣的跨神經網路、可透過自動明確套用功能交會、避免工作特定的功能工程。跨網路由多個層組成、其中最高程度的互動取決於層深度。每個層級都會根據現有層級產生較高順序的互動、並維持先前層級的互動。

深度神經網路（DNN）承諾能夠擷取功能之間非常複雜的互動。然而、相較於DCN、它需要的參數幾乎多出一個順序、無法明確形成交叉功能、也可能無法有效瞭解某些類型的功能互動。跨網路的記憶體效率極高、而且易於實作。共同訓練交叉和DNN元件、有效擷取預測性功能互動、並在Criteo CTR資料集上提供最先進的效能。

一種新的DCN模式從內嵌和堆疊層開始、接著是跨網路和平行的深度網路。接著是最後的組合層、將兩個網路的輸出結合在一起。您的輸入資料可以是具有稀少和密集功能的向量。在Spark、兩者兼具 "ML" 和 "mlLib" 程式

庫包含「parseVector」類型。因此、使用者必須區分兩者、並在呼叫各自的功能和方法時務必留意。在CTR預測等網路規模的建議系統中、輸入內容大多是明確的功能、例如「country=USA」（國家/地區=美國）。這類功能通常編碼為單一熱向量、例如「[0、1、0、…]」。單一熱編碼（OHT）搭配「parseVector」（parseVector）、在處理真實世界的資料集時非常實用、因為這些資料集的詞彙不斷變化且不斷成長。我們在中修改了範例 "DeepCTR" 為了處理大型詞彙、在我們的DCN嵌入與堆疊層中建立嵌入式向量。

。 "Criteo展示廣告資料集" 預測廣告點擊率。它有13個整數特徵和26個分類特徵、其中每個類別都有高基數。對於此資料集而言、由於輸入大小較大、記錄遺失的效能提升實際上相當可觀。對於大型使用者群而言、預測準確度略有改善、可能導致公司營收大幅增加。資料集包含7天內11GB的使用者記錄、相當於約4、100萬筆記錄。我們使用Spark的「datFrame.隨機 分割（）功能」來隨機分割訓練（80%）、交叉驗證（10%）及其餘10%的資料以供測試。

在TensorFlow上實作的是Keras。使用新一代會議管理系統進行模型訓練程序的主要部分有四個：

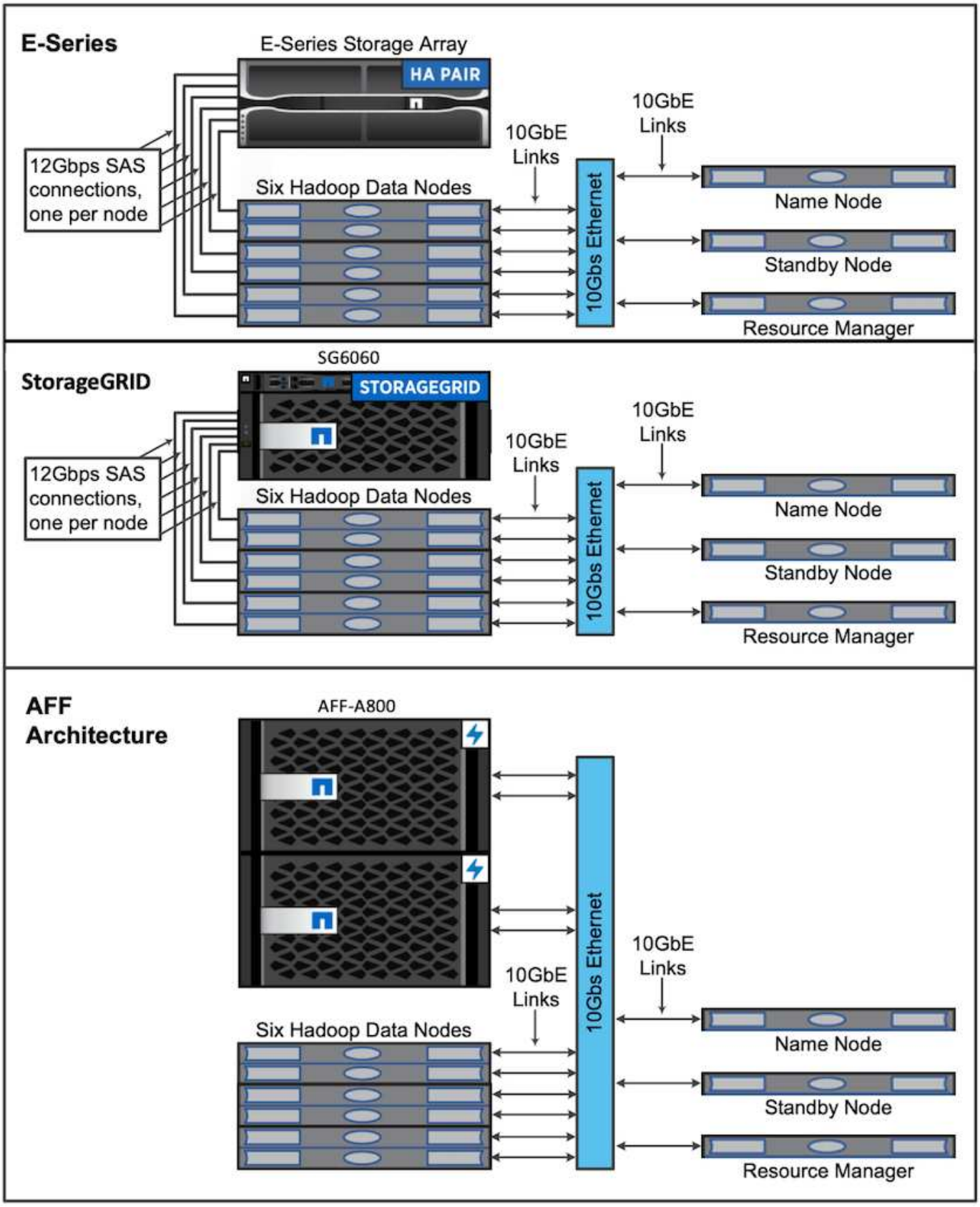
- *資料處理與內嵌。*實際價值的功能會透過套用記錄轉換來標準化。對於分類功能、我們將功能內嵌在尺寸6×（類別基數）1/4的密集向量中。串連所有的內嵌文字會產生維度向量1026。
- *最佳化。*我們使用Adam最佳化工具套用迷你批次隨機最佳化。批次大小設定為512。批次正規化已套用至深度網路、且漸層剪輯規範設定為100。
- *正規化。*我們使用早期停機、因為L2正規化或資料脫落並未生效。
- * Hyperparameters.*我們會根據在隱藏圖層數、隱藏圖層大小、初始學習率及跨圖層數上的網格搜尋結果來報告結果。隱藏的圖層數量介於2到5之間、隱藏的圖層大小介於32到1024之間。對於DCN、跨層的數量從1到6。初始學習率從0.0001調至0.001、增量為0.0001。所有實驗都會在訓練步驟150、000之前提早停止、之後就開始過度調整。

除了新增的新一代會議管理系統、我們也測試了其他熱門的深度學習模式、以利進行CTR預測、包括 "DeepFM"、 "深層FM"、 "自動整型"和 "新一代的"。

用於驗證的架構

在這項驗證中、我們使用四個工作節點和一個主節點、以及AFF-A800 HA配對。所有叢集成員都透過10GbE網路交換器連線。

針對本NetApp Spark解決方案驗證、我們使用三種不同的儲存控制器：E5760、E5724和AFF-A800。E系列儲存控制器連接至五個資料節點、並具有12Gbps SAS連線。透過10GbE連線至AFF Hadoop工作節點、可提供匯出的NFS磁碟區。Hadoop叢集成員是透過E系列AFF 、E-系列、E-、StorageGRID 及《Hadoop解決方案」中的10GbE連線進行連線。



測試結果

我們使用TeraGen基準測試工具中的TeraSort和TeraValidate指令碼、以E5760、E5724

和AFF-A800組態來測量Spark效能驗證。此外、我們也測試了三個主要使用案例：Spark NLP管道和TensorFlow分散式訓練、Horovod分散式訓練、以及使用Keras與DeepFM進行CTR預測的多工深度學習。

對於E系列和StorageGRID E驗 證、我們使用Hadoop複寫係數2。為了驗證、我們只使用一個資料來源。AFF

下表列出Spark效能驗證的硬體組態。

類型	**Hadoop**工作節點	磁碟機類型	每個節點的磁碟機數量	儲存控制器
SG6060	4.	SAS	12.	單一高可用度（HA）配對
E5760	4.	SAS	60	單一HA配對
E5724	4.	SAS	24	單一HA配對
AFF800	4.	SSD	6.	單一HA配對

下表列出軟體需求。

軟體	版本
RHEL	7.9
OpenJDK執行時間環境	1.8.0
OpenJDK 64位元伺服器VM	25、302
Git	2.24.1.
gcc/g++	11.2.1
火花	3.2.1
PySpark	3.1.2
SparkNLP	3.4.2
TensorFlow	2.9.0
Keras	2.9.0
霍羅沃德	0.24.3

財務情資分析

我們發表了 "TR-4910：客戶與NetApp AI溝通時的意見分析"、其中使用建立端點對端點對話AI管道 "NetApp DataOps工具套件"、不儲存及NVIDIA DGX系統。AFF此管線運用DataOps Toolkit執行批次音訊訊號處理、自動語音辨識（ASR）、傳輸學習和情緒分析、 "NVIDIA Riva SDK"和 "TAO架構"。我們將情緒分析使用案例擴大至金融服務業、建立SparkNLP工作流程、針對各種NLP工作（例如命名實體辨識）、載入三種Bert模式、並針對NASDAQ前10大公司的季度營收要求、獲得句子等級的感受。

下列指令碼「ention_sization_SPARK」。PY’使用FinBERT模型來處理HDFS的錄音記錄、並產生正面、中立和負面的感覺、如下表所示：

```
-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
```

下表列出2016年至2020年、NASDAQ前10大企業的獲利能力與句子層級意見分析。

情緒和百分比	所有**10**家公司	**AAPL**	**AMD**	**AMZN**	**CSCO**	**gOOgL**	**INTC**	**MSFT**	**NVDA**
正面評價	747	1567	743	2990	682	826	824	904	417
中性數	6967	6856	7596	5086	6650	5914	6099	5715	6189.
負面數	1787	253	213	84.	189.	97	282.5.	202.02	89
未分類的計數	196	0	0	76.	0	0	0	1.	0
（總計數）	73497.	8676.	8552	5536	7521	6837	7205.	6822	6695

就百分比而言、CEO和CFO所說的大部分句子都是事實、因此具有中立的感覺。在進行獲利拜訪時、分析師會提出一些可能會傳達正面或負面情緒的問題。值得進一步從數量上調查、也就是在同一天或隔天的交易中、負面或正面的情緒對股票價格有何影響。

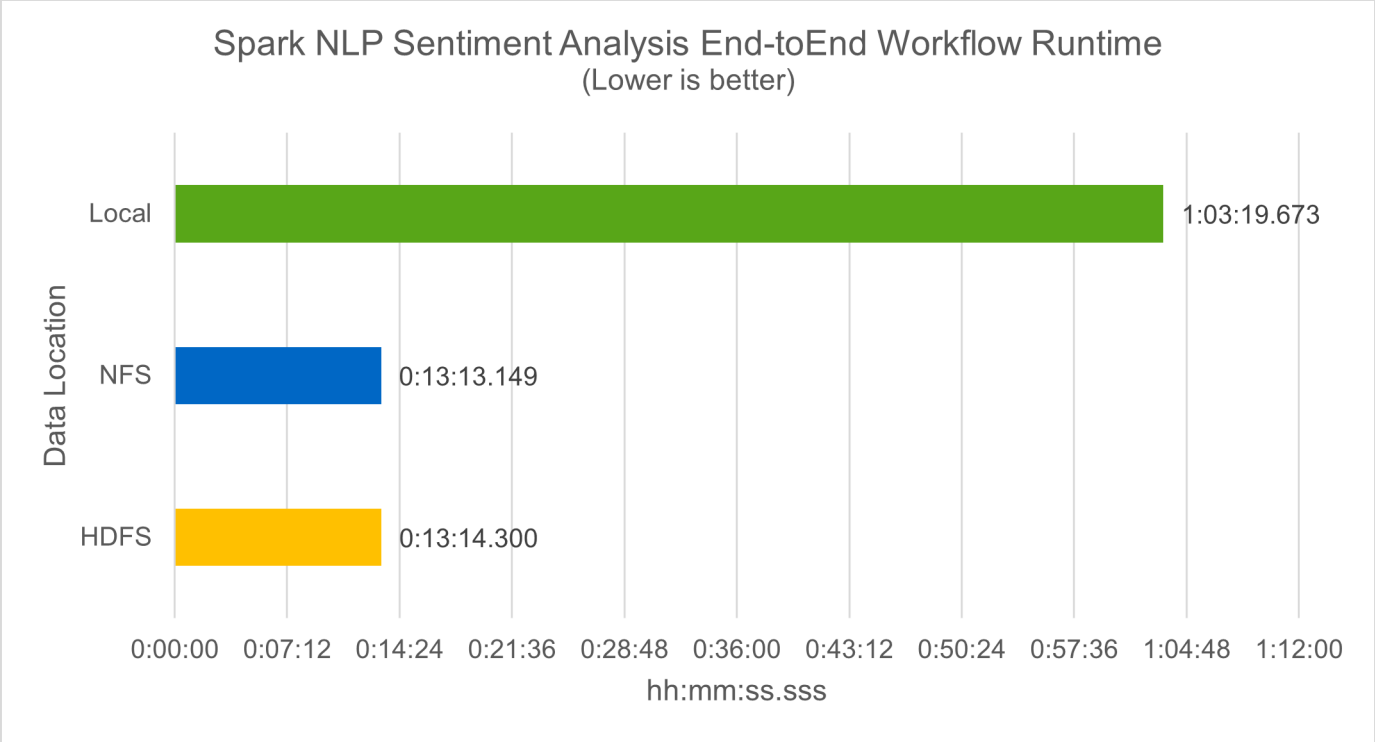
下表列出NASDAQ前10大公司的句子層級意見分析、以百分比表示。

情緒百分比	所有**10**家公司	**AAPL**	**AMD**	**AMZN**	**CSCO**	**gOOgL**	**INTC**	**MSFT**	**NVDA**
正面	10.13%	18.06%	8.69%	5.24%	9.07%	12.08%	11.44%	13.25%	6.23%
中立	87.17%	79.02%	88.82%	91.87%	88.42%	86.50%	84.65%	83.77%	92.44%
負面	2.43%	2.92%	2.49%	1.52%	2.51%	1.42%	3.91%	2.96%	1.33%
未分類	0.27%	0%	0%	1.37%	0%	0%	0%	0.01%	0%

在工作流程執行時間方面、我們發現HDFS的「本機」模式已大幅改善4.78倍、而NFS則進一步改善0.14%。

```
-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
```

如下圖所示、資料與模型平行處理可改善資料處理及分散式TensorFlow模型的推斷速度。NFS中的資料位置產生的執行時間稍微好一點、因為工作流程瓶頸是下載預先訓練的模型。如果我們增加記錄資料集的大小、NFS的優勢就更明顯。



以**Horovod**效能進行分散式訓練

下列命令使用單一「master」節點在Spark叢集中產生執行時間資訊和記錄檔、每個節點有160個執行器、各有一個核心。執行程式記憶體限制為5GB、以避免記憶體不足錯誤。請參閱一節 "「每個主要使用案例的Python指令碼」" 如需有關資料處理、模型訓練及模型準確度計算的詳細資訊、請參閱「keras」（keras）、「SPAR_horovod_rossmann_imer.py」（keras）。

```
(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
```

十個訓練期間的執行時間如下：

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

處理輸入資料、訓練DNN模型、計算準確度、以及產生TensorFlow檢查點和CSV檔案以供預測結果、所需時間超過43分鐘。我們將訓練時段的數量限制為10個、實際上通常設定為100個、以確保模型準確度令人滿意。訓練時間通常會隨著epochs的數量線性調整。

接下來、我們使用叢集中可用的四個工作節點、並在「線」模式中執行相同指令碼、並在HDFS中使用資料：

```
(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
```

結果的執行時間改善如下：

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

霍羅沃德在Spark的模式和資料平行化技術、讓我們看到5.29倍的執行時間加速比「線」與「本地」模式、並有十個訓練階段。下圖顯示了「HDFS」和「本地」的圖例。如果有可用的GPU、基礎TensorFlow DNN模型訓練

可進一步加速。我們計畫在未來的技術報告中進行此測試並發佈結果。

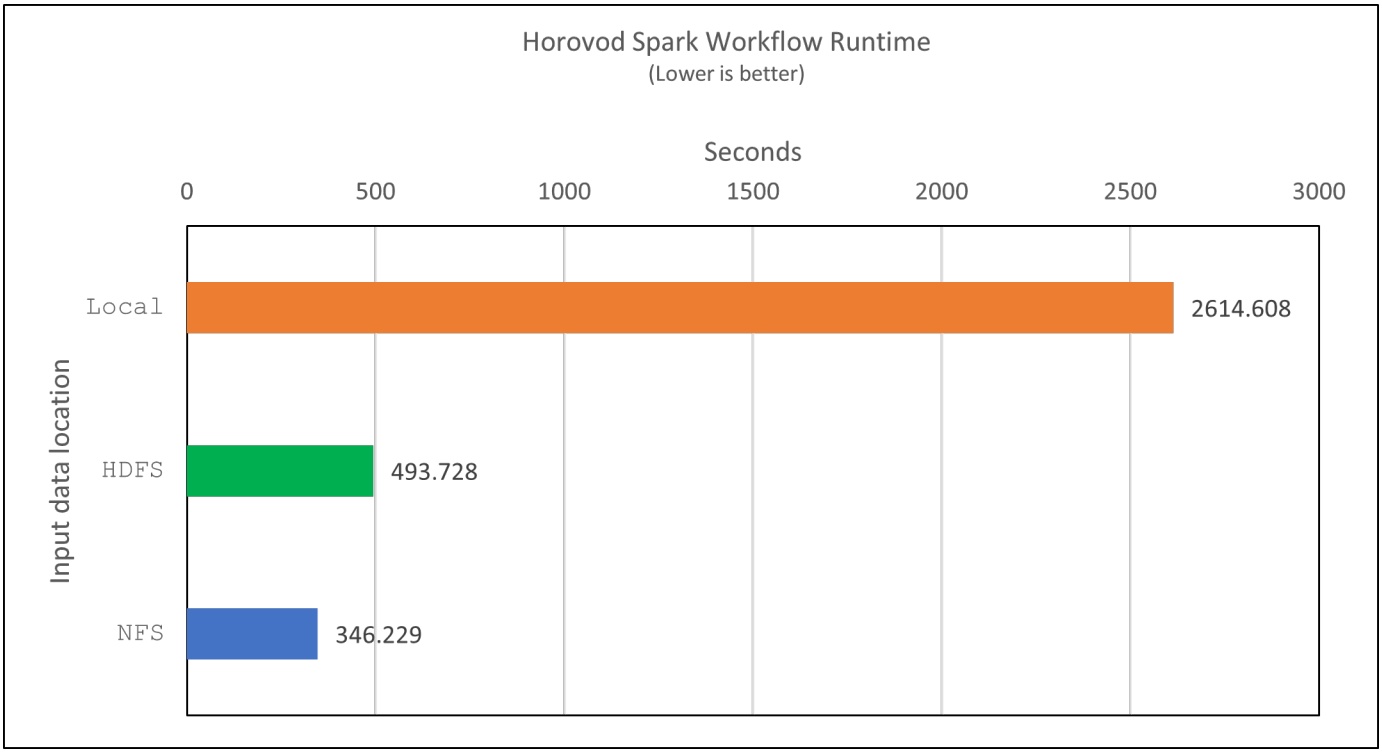
我們的下一項測試將執行時間與NFS中的輸入資料與HDFS進行比較。在Spark叢集中的五個節點（一位主節點、四位員工）上、安裝了位於Se A800上AFF 的NFS磁碟區。我們執行的命令與先前的測試類似、現在的「-data- dir」參數指向NFS掛載：

```
(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
```

產生的NFS執行時間如下：

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

另有1.43倍的加速比、如下圖所示。因此、透過將NetApp All Flash儲存設備連線至叢集、客戶可享有Horovod Spark工作流程的快速資料傳輸與發佈優勢、相較於在單一節點上執行、可獲得7.55倍的加速比。



深度學習模式、提供**CTR**預測效能

針對最大化CTR的推薦系統、您必須瞭解使用者行為背後的複雜功能互動、這些行為可以從低階到高階的數學計算得出。對於良好的深度學習模式而言、低階和高階功能互動同樣重要、而不需互相偏好。深度Factorization Machine（DeepFM）是一種面向機器的神經網路、結合了面向技術的機器、可在全新的神經網路架構中提供建議和深度學習功能。

雖然傳統的面向化機器會將配對功能互動視為潛在功能之間的內部產品、理論上也能擷取高階資訊、但實際上、機器學習工作者通常只會因為高運算和儲存複雜度而使用二階功能互動。深入的神經網路變種、例如Google "廣角安培；深層機型" 另一方面、將線性寬模型與深度模型結合、即可在混合式網路架構中學習精密的功能互動。

這種廣域與深層模型有兩種輸入、一種是基礎廣泛模型、另一種是深度模型、其後一部分仍需要專家特徵工程、因此技術較不適用於其他網域。與廣角和深層模型不同的是、DeepFM可有效訓練原始功能、無需任何特徵工程、因為其廣泛的部分和深層部分共用相同的輸入和內嵌向量。

我們首先使用本節中的「rrun _crite_criteo_wark.py」、將criteo「tr.txt」（11GB）檔案處理成一個CSV檔案、名稱為「ctr_tr.csv"、儲存在NFS掛載「/swarkdemo/tr-4570資料」中 "「每個主要使用案例的Python指令碼。」" 在此指令碼中、「Process輸入檔案」功能會執行數種字串方法來移除索引標籤、並將「、」插入為分隔符號、將「n」插入為新行。請注意、您只需處理一次原始的「train.txt」、就能將程式碼區塊顯示為註解。

針對下列不同DL機型的測試、我們使用「ctr_train.csv"做為輸入檔。在後續的測試執行中、輸入CSV檔案會讀入Spark DataFrame、其中架構包含「label」欄位、整數密集功能「'I1'、'I2」、「I3」、…、「I13」]、 以及「'c1'、'c2'、'c3」、…、'c26']等功能。下列「駐點提交」命令採用輸入CSV、將DeepFM模型分成20%進行交叉驗證、並在十個訓練期後挑選最佳模型、以計算測試集的預測準確度：

```
(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
```

請注意、由於資料檔案「ctr_tr.csv"超過11GB、因此您必須設定一個大於資料集大小的「shipt.driver.max.ResultSize'以避免錯誤。

```
 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()
```

在上述的「parkSession・builder」組態中、我們也啟用了 "Apache Arrow"、使用「d・toPandas（）」方法、將Spark DataFrame轉換成成Pandas DataFrame。

```
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.
```

隨機分割之後、訓練資料集中有超過36M列、測試集中有9M樣本：

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

由於本技術報告著重於不使用任何GPU的CPU測試、因此您必須使用適當的編譯器旗標來建置TensorFlow。此步驟可避免啟動任何GPU加速程式庫、並充分利用TensorFlow的進階向量擴充（AVX）和AVX2指令。這些功能是專為線性代數運算所設計、例如向量化的新增功能、饋送轉送內的矩陣複用、或是後傳DNN訓練。使用256位元浮點（FP）登錄的AVX2可搭配使用融合式多層新增（FMA）指令、是整型程式碼和資料類型的理想選擇、可產生高達2倍的加速。對於FP程式碼和資料類型、AVX2比AVX快8%。

```
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.
```

若要從來源建置TensorFlow、NetApp建議使用 "巴茲爾"。在我們的環境中、我們在Shell提示字元中執行下列命令、以安裝「dNF」、「dNF-plugins」和「Bazel」。

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

您必須在建置過程中啟用海灣合作委員會5或更新版本、才能使用C++17功能、這是由RHEL搭配軟體集合庫（SCL）提供的功能。下列命令會在RHEL 7.9叢集上安裝「devtoolset」和「gcc11.2.1」：

```
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
```

請注意、最後兩個命令會啟用「devtoolSet-11」、使用「/opt/r/devtoolSet-11/root/usr/in/gccs」（gcc11.2.1）。此外、請確定您的「git」版本大於1.8.3（RHEL 7.9隨附）。請參閱此 "文章" 將「git」更新為2.24.1。

我們假設您已複製最新的TensorFlow主要repo。然後使用「工作區」檔案建立「工作區」目錄、以使用AVX、AVX2和FMA從來源建置TensorFlow。執行「configure」檔案、並指定正確的Python二進位位置。 "CUDA" 因為我們沒有使用GPU、所以測試時停用。系統會根據您的設定產生「.bazelrc」檔案。此外、我們編輯檔案並設定「build -define = no_HDfs_support=fals'」以啟用HDFS支援。請參閱一節中的「.bazelrc」 "「每個主要使用案例的Python指令碼」" 以取得設定和旗標的完整清單。

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

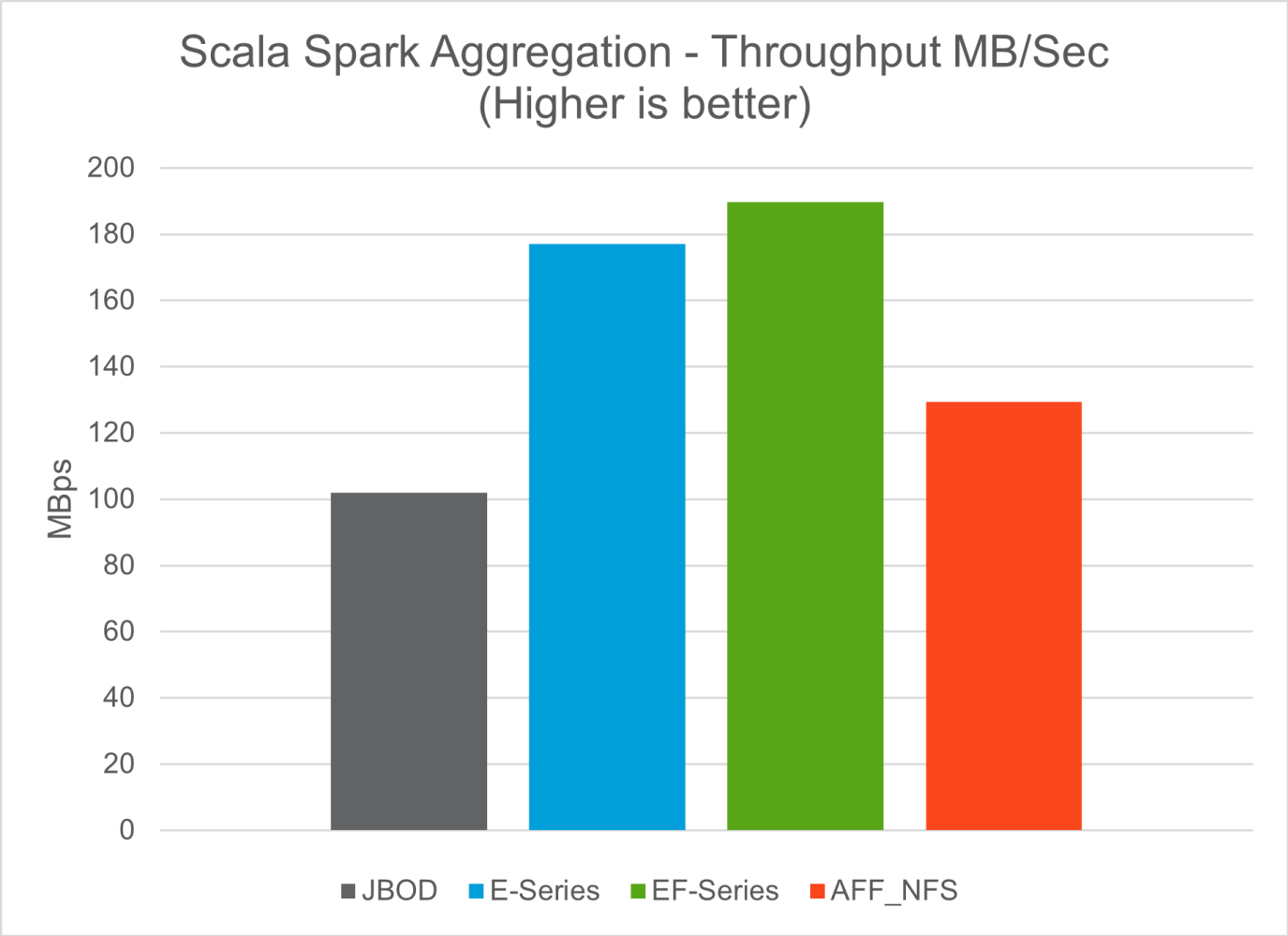
使用正確的旗標建置TensorFlow之後、請執行下列指令碼來處理Criteo顯示廣告資料集、訓練DeepFM模型、並從預測分數計算接收器作業特性曲線（ROC AUC）下的區域。

```
(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
```

經過十次訓練、我們在測試資料集上獲得AUC分數：

```
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
```

我們以類似先前使用案例的方式、比較Spark工作流程執行時間與位於不同位置的資料。下圖顯示Spark工作流程執行時間的深度學習CTR預測比較。

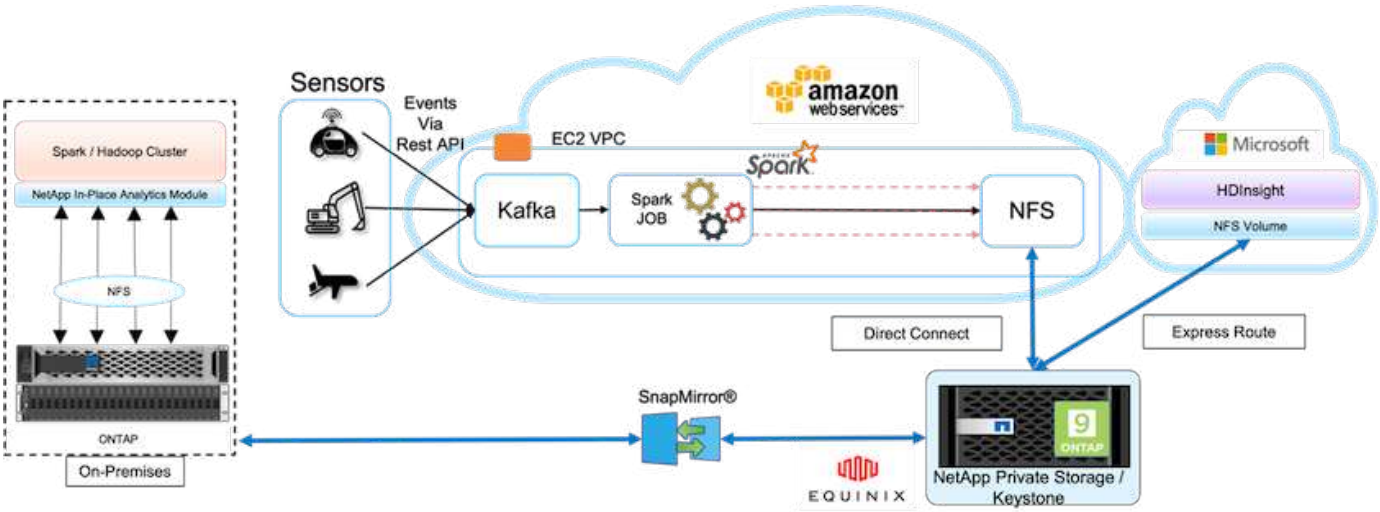


混合雲解決方案

現代化的企業資料中心是一種混合雲、可透過持續資料管理層面、以一致的營運模式、在內部部署和/或多個公有雲中、連接多個分散式基礎架構環境。若要充分發揮混合雲的效益、您必須能夠在內部部署和多雲端環境之間順暢地移動資料、而不需要進行任何資料轉換或應用程式重新構建。

客戶表示、他們開始混合雲的過程、可能是將二線儲存設備移至雲端以供使用、例如資料保護、或是將較不重要的業務工作負載（例如應用程式開發和DevOps）移至雲端。然後繼續處理更重要的工作負載。Web與內容代管、DevOps與應用程式開發、資料庫、分析及容器化應用程式是最受歡迎的混合雲工作負載之一。企業AI專案的複雜度、成本和風險、向來阻礙AI從實驗階段到正式作業階段的採用。

有了NetApp混合雲解決方案、客戶就能透過單一控制面板、在分散式環境中進行資料與工作流程管理、同時根據使用量最佳化總體擁有成本、從整合式安全性、資料治理和法規遵循工具中獲益。下圖是雲端服務合作夥伴的範例解決方案、其任務是為客戶的Big資料分析資料提供多雲端連線。



在此案例中、從不同來源收到的AWS IoT資料會儲存在NetApp私有儲存設備（NPS）的中央位置。NPS儲存設備連接到AWS和Azure中的Spark或Hadoop叢集、可讓在多個雲端上執行的巨量資料分析應用程式存取相同的資料。本使用案例的主要要求與挑戰包括：

- 客戶想要使用多個雲端在相同的資料上執行分析工作。
- 必須透過不同的感應器和集線器、從內部部署和雲端環境等不同來源接收資料。
- 解決方案必須有效率且具成本效益。
- 主要挑戰是建置具成本效益且高效率的解決方案、在不同的內部部署環境與雲端環境之間提供混合式分析服務。

我們的資料保護與多重雲端連線解決方案解決了跨多個大型擴充程式使用雲端分析應用程式的難題。如上圖所示、感應器的資料會透過Kafka串流並擷取至AWS Spark叢集。資料儲存在NPS中的NFS共用區、NPS位於Equinix資料中心內的雲端供應商外部。

由於NetApp NPS分別透過Direct Connect和Express Route連線連線至Amazon AWS和Microsoft Azure、因此客戶可以利用就地分析模組、從Amazon和AWS分析叢集存取資料。因此、因為內部部署和NPS儲存都執行ONTAP 了一套功能不全的軟體、 "SnapMirror" 可將NPS資料鏡射至內部部署叢集、在內部部署和多個雲端之間提供混合雲分析。

為獲得最佳效能、NetApp通常建議使用多個網路介面、直接連線或快速路由、從雲端執行個體存取資料。我們還有其他資料移動機解決方案、包括 "XCP" 和 "BlueXP 複製與同步" 協助客戶建置應用程式感知、安全且具成本效益的混合雲Spark叢集。

每個主要使用案例的**Python**指令碼

下列三個Python指令碼分別對應三個測試的主要使用案例。第一是"entitment_section_sparknlp.py"。

```
# 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}")
```

第二個指令碼是「keras」（keras）、「SPARK」（horovod_rossmann_imer.py）。

```
# 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()
```

第三個指令碼是「rrun _crite_criteo_site.py」。

```
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))
```

結論

在本文件中、我們將討論Apache Spark架構、客戶使用案例、以及NetApp儲存產品組合、這些內容涉及巨量資料、現代化分析、以及AI、ML和DL。在我們根據產業標準基準測試工具和客戶需求進行的效能驗證測試中、NetApp Spark解決方案展現出優於原生Hadoop系統的效能表現。本報告所述的客戶使用案例與效能結果組合、可協助您選擇適合部署的Spark解決方案。

何處可找到其他資訊

此TR使用下列參考資料：

- Apache Spark架構與元件

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

- Apache Spark使用案例

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

- Apache挑戰

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

- Spark NLP

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

- Bert

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

- 深入跨網的廣告點選預測

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

- FlexGroup

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

- 串流ETL

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

- 適用於Hadoop的NetApp E系列解決方案

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

- 客戶與NetApp AI溝通時的情緒分析

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

- NetApp現代化資料分析解決方案

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

- 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 複製與同步

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

- DataOps工具套件

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

Big Data分析資料到人工智慧

TR-4732：巨量資料分析資料與人工智慧

NetApp的Karthithkeyan Nagalingam

本文件說明如何將巨量資料分析資料和HPC資料移至AI。AI透過NFS匯出來處理NFS資料、而客戶通常會將AI資料放在大資料分析平台、例如HDFS、Blob或S3儲存設備、以及GPFS等HPC平台。本白皮書提供使用NetApp XCP和NIPAM將巨量資料分析資料和HPC資料移轉至AI的準則。我們也會討論將資料從Big Data和HPC移轉至AI的商業效益。

概念與元件

Big Data分析儲存設備

Big Data分析是HDFS的主要儲存供應商。客戶通常使用Hadoop相容的檔案系統（HCFS）、例如Windows Azure Blob Storage、MapR檔案系統（MapR-FS）和S3物件儲存設備。

通用平行檔案系統

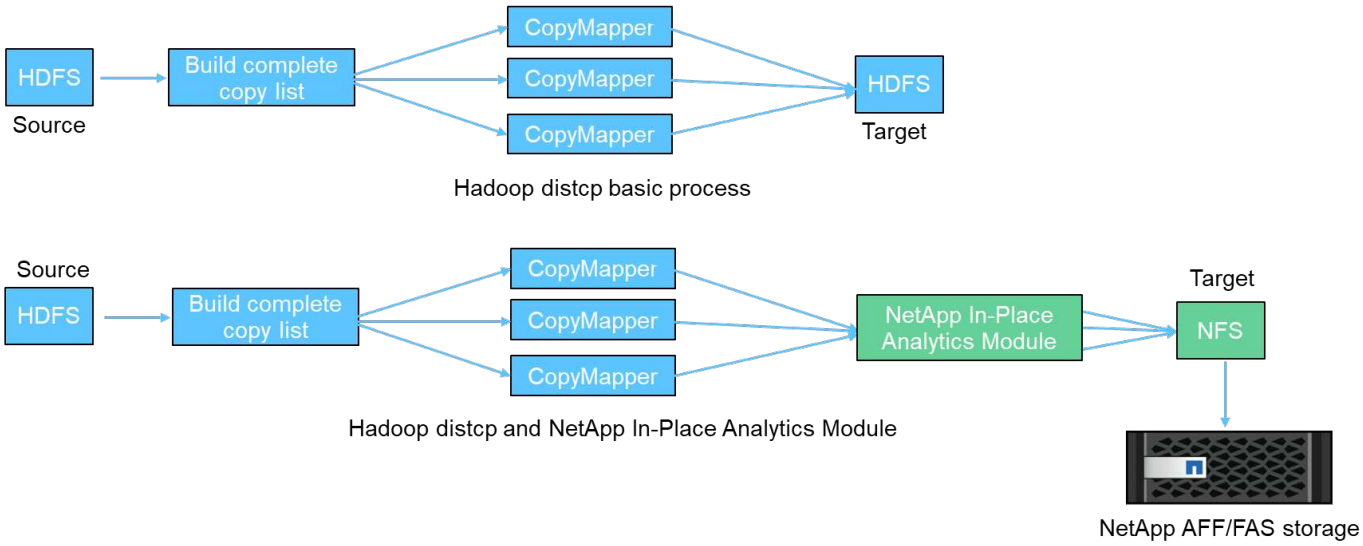
IBM的GPFS是企業級檔案系統、可取代HDFS。GPFS可讓應用程式靈活決定區塊大小和複寫配置、以提供良好的效能和效率。

NetApp就地分析模組

NetApp就地分析模組（NIPAM）是Hadoop叢集存取NFS資料的驅動程式。它有四個元件：連線集區、NFS輸入串流、檔案處理快取和NFS輸出串流。如需詳細資訊、請參閱 "TR-4382：NetApp就地分析模組。"

Hadoop分散式複本

Hadoop分散式複製（DistCp）是一種分散式複製工具、用於大型叢集間和叢集內的處理工作。此工具使用MapReduce進行資料發佈、錯誤處理及報告。它會展開檔案和目錄清單、並輸入它們來對應工作、以便從來源清單複製資料。下圖顯示HDFS和非HDFS的DistCp作業。



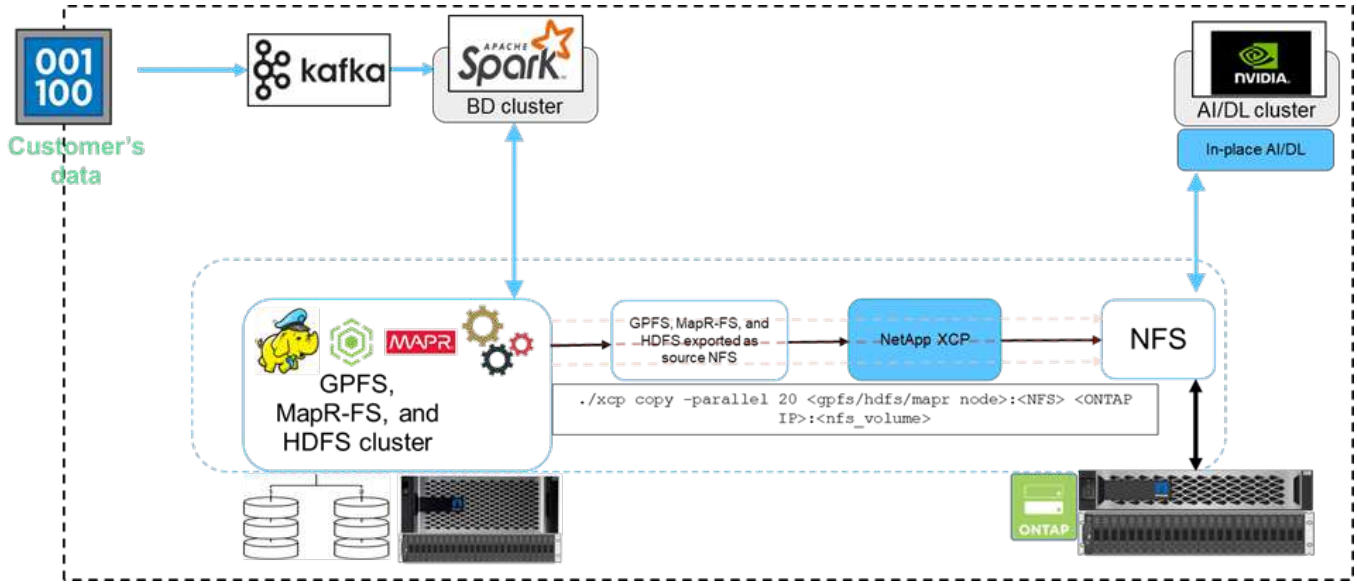
Hadoop DistCp可在兩個HDFS系統之間移動資料、而無需使用額外的驅動程式。NetApp為非HDFS系統提供驅動程式。對於NFS目的地、NIPAM提供驅動程式、可複製Hadoop DistCp在複製資料時用於與NFS目的地通訊的資料。

NetApp Cloud Volumes Service

NetApp Cloud Volumes Service 解決方案是雲端原生檔案服務、效能極佳。這項服務可快速將資源上下轉動、並使用NetApp功能來提高生產力並縮短員工停機時間、藉此協助客戶加速上市時間。此產品可減少整體資料中心佔用空間、減少使用原生公有雲儲存設備的成本、因此、此產品是災難恢復與雲端備份的理想替代方案。Cloud Volumes Service

NetApp XCP

NetApp XCP是用戶端軟體、可快速且可靠地進行NetApp與NetApp之間的資料移轉。此工具旨在將大量非結構化NAS資料從任何NAS系統複製到NetApp儲存控制器。XCP移轉工具使用多核心、多通道I/O串流引擎、可平行處理許多要求、例如資料移轉、檔案或目錄清單、以及空間報告。這是預設的NetApp資料移轉工具。您可以使用XCP將資料從Hadoop叢集和HPC複製到NetApp NFS儲存設備。下圖顯示使用XCP從Hadoop和HPC叢集傳輸到NetApp NFS磁碟區的資料。



NetApp BlueXP 複製與同步

NetApp BlueXP 複製與同步是混合式資料複寫軟體即服務、可在內部部署儲存設備和雲端儲存設備之間順暢且安全地傳輸及同步 NFS 、 S3 和 CIFS 資料。此軟體可用於資料移轉、歸檔、協同作業、分析等作業。傳輸資料後、 BlueXP 複製與同步會持續同步來源與目的地之間的資料。接下來、它會傳輸差異。它也能保護您自己網路、雲端或內部部署中的資料安全。本軟體以隨用隨付模式為基礎、提供具成本效益的解決方案、並提供監控與報告功能、方便您傳輸資料。

客戶挑戰

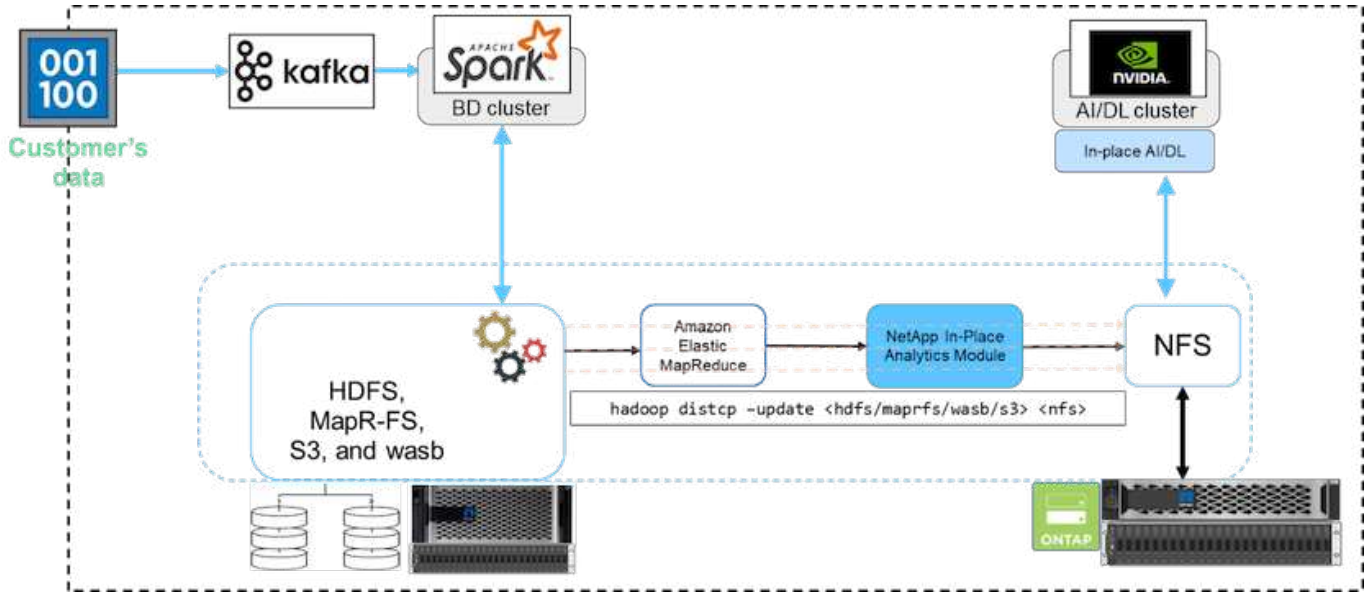
客戶在嘗試從Big Data分析中存取資料以進行AI作業時、可能會面臨下列挑戰：

- 客戶資料位於資料湖儲存庫中。資料湖可以包含不同類型的資料、例如結構化、非結構化、半結構化、記錄和機器對機器資料。所有這些資料類型都必須在AI系統中處理。
- AI與Hadoop檔案系統不相容。典型的AI架構無法直接存取HDFS和HCFS資料、而這些資料必須移至AI可理解的檔案系統（NFS）。
- 將資料湖資料搬移至AI通常需要專業的程序。資料湖中的資料量可能很大。客戶必須擁有高效率、高處理量且具成本效益的方法、才能將資料移入AI系統。
- 正在同步資料。如果客戶想要在Big Data平台和AI之間同步資料、有時候透過AI處理的資料可以搭配Big Data一起用於分析處理。

資料移動機解決方案

在巨量資料叢集中、資料會儲存在HDFS或HCFS中、例如MapR-FS、Windows Azure Storage Blob、S3或Google檔案系統。我們使用ONTAP 來自來源的「Hadoop distcp」命令、利用NIPAM的協助、使用HDFS、MapR-FS和S3作為將資料複製到NetApp支援NFS匯出的來源、進行測試。

下圖說明一般資料從執行HDFS儲存設備的Spark叢集移至NetApp ONTAP 支援NFS磁碟區、以便NVIDIA處理AI作業。



「Hadoop distcp」命令會使用MapReduce程式來複製資料。NIPAM與MapReduce搭配使用、可在複製資料時

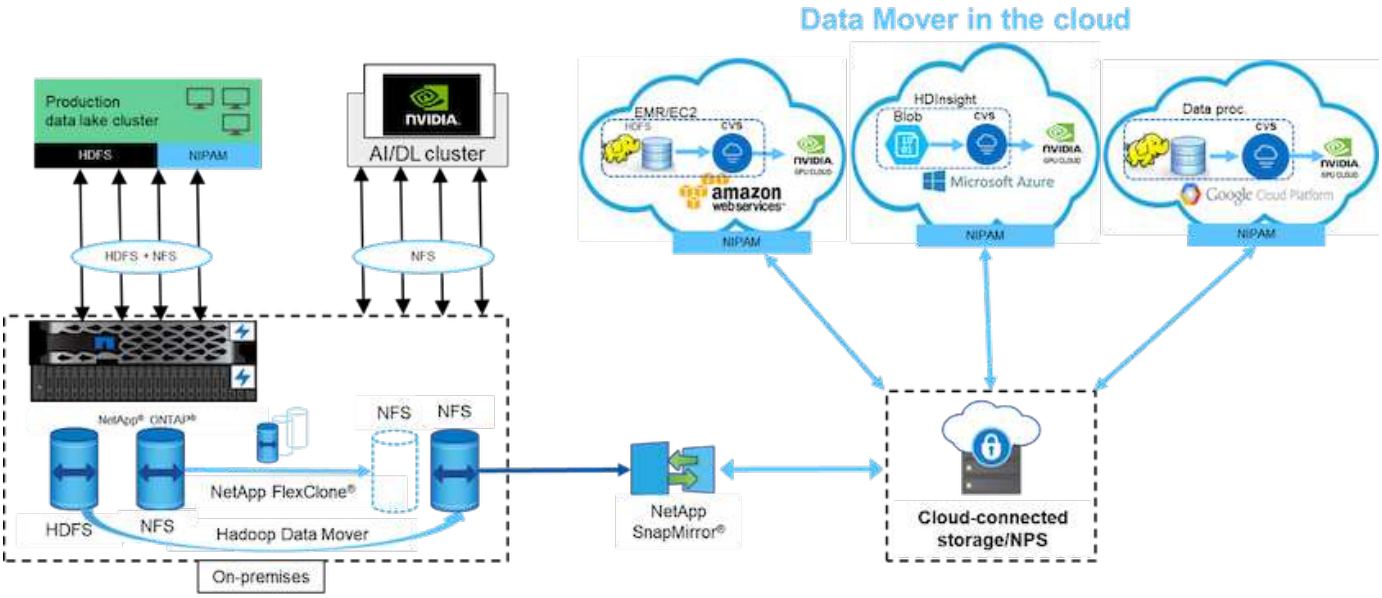
做為Hadoop叢集的驅動程式。NIPAM可在多個網路介面之間分散負載、以利單一匯出。當您將資料從HDFS或HCFS複製到NFS時、此程序會將資料分散到多個網路介面、藉此將網路處理量最大化。

> 不支援NIPAM、也不通過MapR認證。

AI的資料移動機解決方案

AI的資料移轉解決方案是根據客戶處理AI作業Hadoop資料的需求而打造。NetApp使用NIPAM將資料從HDFS移至NFS。在一種使用案例中、客戶需要將資料移至內部部署的NFS、而另一位客戶則需要將資料從Windows Azure Storage Blob移至Cloud Volumes Service Suse、才能處理來自雲端GPU雲端執行個體的資料。

下圖說明資料移動機解決方案的詳細資料。



建置資料移轉器解決方案需要執行下列步驟：

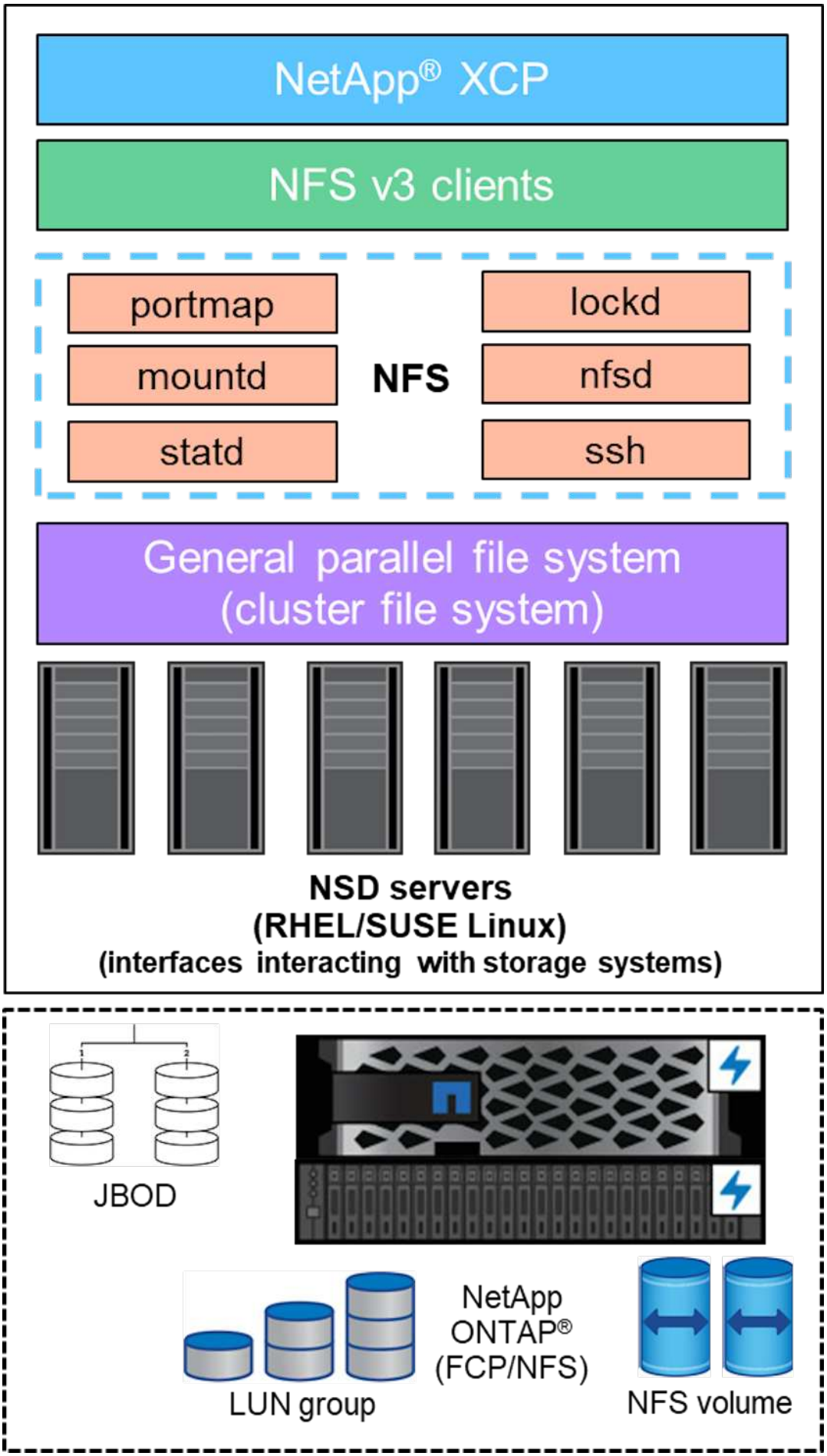
1. 支援HDFS的SAN、NAS則透過NIPAM將NFS磁碟區提供給正式作業資料湖叢集。ONTAP
2. 客戶的資料位於HDFS和NFS中。NFS資料可以是用於Big Data分析和AI作業的其他應用程式的正式作業資料。
3. NetApp FlexClone技術會建立正式作業NFS磁碟區的複本、並將其配置至內部部署的AI叢集。
4. 使用NIPAM和「Hadoop distcp」命令、將HDFS SAN LUN的資料複製到NFS磁碟區。NIPAM使用多個網路介面的頻寬來傳輸資料。此程序可縮短資料複製時間、以便傳輸更多資料。
5. 這兩個NFS磁碟區都會配置至AI叢集、以供AI作業。
6. 若要使用雲端中的GPU來處理內部部署NFS資料、NFS磁碟區會使用NetApp SnapMirror技術鏡射至NetApp私有儲存設備（NPS）、並掛載至GPU的雲端服務供應商。
7. 客戶想要處理來自雲端服務供應商之GPU中EC2/EMR、HDInsight或DataProc服務中的資料。Hadoop資料移轉器可利用NIPAM和「Hadoop distcp」命令、將資料從Hadoop服務移至Cloud Volumes Services。
8. 透過NFS傳輸協定將支援的資料配置給AI。透過AI處理的資料、除了可透過NIPAM、SnapMirror和NPS傳送NVIDIA叢集之外、也可傳送到內部部署位置進行巨量資料分析。Cloud Volumes Service

在此案例中、客戶在NAS系統的遠端位置擁有大量的檔案數資料、而在內部部署的NetApp儲存控制器上進行AI處理時、則需要這些資料。在此案例中、最好使用XCP移轉工具以更快的速度移轉資料。

混合使用案例客戶可使用 BlueXP 複製與同步功能、將內部部署資料從 NFS 、 CIFS 和 S3 資料移轉至雲端、反之亦然、可使用 GPU （例如 NVIDIA 叢集中的 GPU ）進行 AI 處理。BlueXP 複製與同步和 XCP 移轉工具都是用於將 NFS 資料移轉至 NetApp ONTAP NFS 。

GPFS移轉至NetApp ONTAP 的不整NFS

在此驗證中、我們使用四部伺服器做為網路共用磁碟（NSD）伺服器、為GPFS提供實體磁碟。GPFS是在NSD磁碟上建立、以匯出為NFS匯出、讓NFS用戶端可以存取這些磁碟、如下圖所示。我們使用XCP將資料從GPFS匯出的NFS複製到NetApp NFS磁碟區。


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

GPFS基本功能

GPFS使用下列節點類型：

- *管理節點。*指定選用欄位、其中包含管理命令用來在節點之間進行通訊的節點名稱。例如、管理節點「mastr-51.netapp.com`」可以將網路檢查傳遞給叢集中的所有其他節點。
- *仲裁節點。*決定節點是否包含在從中衍生仲裁的節點集區中。您至少需要一個節點作為仲裁節點。
- * Manager Nod.*表示節點是否為節點集區的一部分、可從中選取檔案系統管理程式和權杖管理程式。最好將多個節點定義為管理節點。您指定為管理程式的節點數量取決於工作負載和您擁有的GPFS伺服器授權數量。如果您執行的是大型平行工作、則可能需要比支援Web應用程式的四節點叢集更多的管理節點。
- * NSD伺服器。*準備每個實體磁碟以搭配GPFS使用的伺服器。
- *傳輸協定節點。*透過任何安全Shell（SSH）傳輸協定與NFS直接共用GPFS資料的節點。此節點需要GPFS伺服器授權。

GPFS、NFS和XCP的作業清單

本節提供建立GPFS、將GPFS匯出為NFS匯出、以及使用XCP傳輸資料的作業清單。

建立GPFS

若要建立GPFS、請完成下列步驟：

1. 在其中一部伺服器上、下載並安裝Linux版本的頻譜級資料存取。
2. 在所有節點上安裝必要的套件（例如主廚）、並在所有節點上停用增強安全性的Linux（SELinux）。
3. 設定安裝節點、並將管理節點和GPFS節點新增至叢集定義檔案。
4. 新增管理節點、仲裁節點、NSD伺服器和GPFS節點。
5. 新增GUI、admin和GPFS節點、並視需要新增額外的GUI伺服器。
6. 新增另一個GPFS節點、然後檢查所有節點的清單。
7. 在叢集定義檔中、指定要在所有GPFS節點上設定的叢集名稱、設定檔、遠端Shell二進位檔、遠端檔案複製二進位檔和連接埠範圍。
8. 檢視GPFS組態設定、並新增額外的管理節點。
9. 停用資料收集、並將資料套件上傳至IBM支援中心。
10. 啟用NTP並在安裝前預先檢查組態。
11. 設定、建立及檢查NSD磁碟。
12. 建立GPFS。
13. 掛載GPFS。
14. 驗證並提供GPFS所需的權限。
15. 執行「dd」命令來驗證GPFS的讀取和寫入。

將GPFS匯出至NFS

若要將GPFS匯出至NFS、請完成下列步驟：

1. 透過「/etc/exports」檔案將GPFS匯出為NFS。
2. 安裝所需的NFS伺服器套件。
3. 啟動NFS服務。
4. 列出GPFS中的檔案以驗證NFS用戶端。

設定**NFS**用戶端

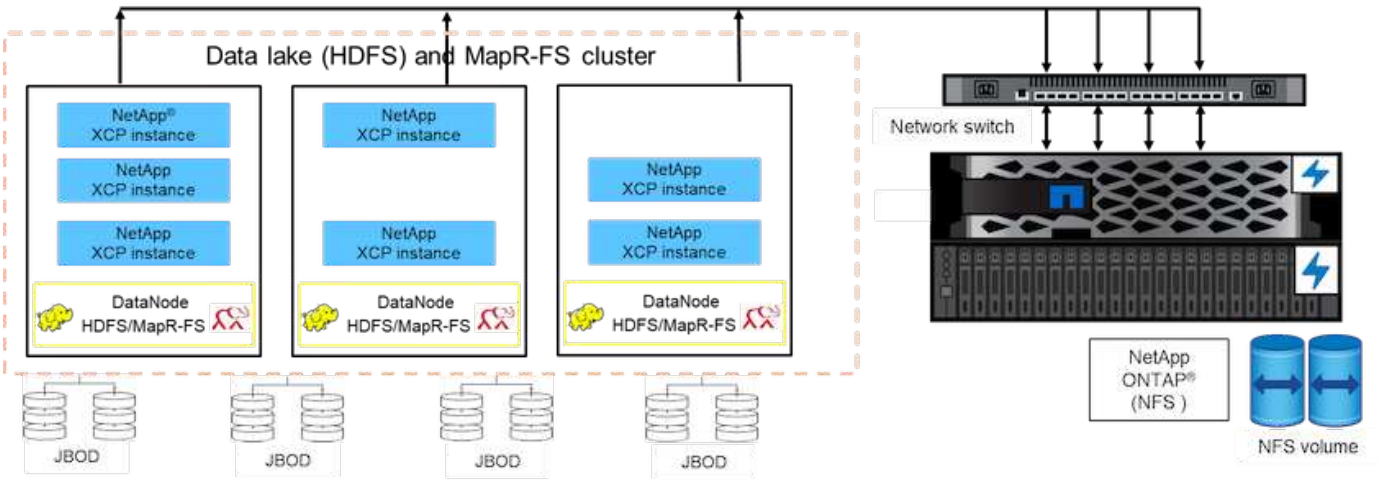
若要設定NFS用戶端、請完成下列步驟：

1. 透過「/etc/exports」檔案將GPFS匯出為NFS。
2. 啟動NFS用戶端服務。
3. 透過NFS用戶端上的NFS傳輸協定掛載GPFS。
4. 驗證NFS掛載資料夾中的GPFS檔案清單。
5. 使用XCP將資料從GPFS匯出的NFS移至NetApp NFS。
6. 驗證NFS用戶端上的GPFS檔案。

HDFS與MapR-FS移轉ONTAP 至不適用NFS

對於此解決方案、NetApp已驗證資料從資料湖（HDFS）和MapR叢集資料移轉至ONTAP不含NFS的情形。資料存放在MapR-FS和HDFS中。NetApp XCP推出一項新功能、可直接將資料從HDFS和MapR-FS等分散式檔案系統移轉至ONTAP 支援NFS。XCP使用非同步執行緒和HDFS C API呼叫來通訊和傳輸MapR-FS和HDFS的資料。

下圖顯示資料從資料湖（HDFS）和MapR-FS移轉到ONTAP SNFS的資料。有了這項新功能、您不需要將來源匯出為NFS共用區。



為何客戶會從**HDFS**和**MapR-FS**移轉至**NFS**？

大部分的Hadoop發佈（例如Cloudera和Hortonworks）都使用HDFS、而MapR發佈則使用自己的檔案系統（稱為MapR-FS）來儲存資料。HDFS與MapR-FS資料可為資料科學家提供寶貴的見解、並可用於機器學習（ML）和深度學習（DL）。HDFS和MapR-FS中的資料不會共用、這表示其他應用程式無法使用該資料。客戶正在尋找共享資料、尤其是在銀行部門、客戶的敏感資料會被多個應用程式使用。最新版本的Hadoop（3.x或更新版本

）支援NFS資料來源、無需其他協力廠商軟體即可存取。有了全新的NetApp XCP功能、資料可以直接從HDFS和MapR-FS移至NetApp NFS、以便存取多個應用程式

在Amazon Web Services（AWS）中進行測試、將資料從MapR-FS傳輸到NFS、以進行12個MapR節點和4個NFS伺服器的初始效能測試。

	數量	尺寸	**VCPU**	記憶體	儲存設備	網路
NFS 伺服器	4.	i3en.24xLarge	96	488GiB	8個7500 NVMe SSD	100
MapR節點	12.	I3en.12xllarge	48	384GiB	4個7500 NVMe SSD	50

根據初始測試結果、我們獲得20Gbps的處理量、而且能夠每天傳輸2PB的資料。

如需不將HDFS匯出至NFS的HDFS資料移轉詳細資訊、請參閱中的「部署步驟- NAS」一節 "TR-4863：TR-4863：NetApp XCP最佳實務準則：資料移轉、檔案移轉及分析"。

商業效益

將資料從Big Data分析移轉至AI可帶來下列效益：

- 能夠從不同的Hadoop檔案系統和GPFS擷取資料至統一化NFS儲存系統
- Hadoop整合式自動化資料傳輸方式
- 降低從Hadoop檔案系統移轉資料的程式庫開發成本
- 透過使用NIPAM、從單一資料來源將多個網路介面的總處理量集合起來、達到最高效能
- 排程及隨需傳輸資料的方法
- 使用ONTAP 效益資料管理軟體、儲存效率與企業管理功能、可用於統一化NFS資料
- 使用Hadoop資料傳輸方法、將資料搬移成本降至零

GPFS至NFS的詳細步驟

本節提供使用NetApp XCP設定GPFS並將資料移至NFS所需的詳細步驟。

設定GPFS

1. 在其中一部伺服器上下載並安裝適用於Linux的Spectrum Scale Data Access。

```
[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. 在所有節點上安裝必要的套件（包括主廚和核心標頭）。

```
[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. 在所有節點中停用SELinux。

```
[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. 設定安裝節點。

```
[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. 將管理節點和GPFS節點新增至叢集定義檔。

```
[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. 新增管理節點和GPFS節點。

```
[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. 新增仲裁節點和GPFS節點。

```
[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. 新增NSD伺服器和GPFS節點。

```
[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. 新增GUI、admin和GPFS節點。

```
[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. 新增另一個GUI伺服器。

```
[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. 新增另一個GPFS節點。

```
[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. 驗證並列出所有節點。

```
[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. 在叢集定義檔中指定叢集名稱。

```
[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. 指定設定檔。

```
[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. 指定GPFS要使用的遠端Shell二進位檔；使用「-r argument」。

```
[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. 指定GPFS要使用的遠端檔案複製二進位檔；使用「-rc引數」。

```
[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. 指定要在所有GPFS節點上設定的連接埠範圍；使用「-e argument」。

```
[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. 檢視GPFS組態設定。

```
[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. 新增管理節點。

```
[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. 停用資料收集、並將資料套件上傳至IBM支援中心。

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

21. 啟用NTP。

```
[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. 安裝前請先檢查組態。

```
[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. 設定NSD磁碟。

```
[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. 建立NSD磁碟。

```
[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. 檢查NSD磁碟狀態。

```
[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. 建立GPFS。

```
[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. 掛載GPFS。

```
[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. 檢查並提供GPFS所需的權限。

```
[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. 執行「dd」命令來檢查GPFS的讀取和寫入。

```
[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]#
```

將GPFS匯出至NFS

若要將GPFS匯出至NFS、請完成下列步驟：

1. 透過「/etc/exports」檔案將GPFS匯出為NFS。

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

2. 安裝所需的NFS伺服器套件。

```
[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. 啟動NFS服務。

```
[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. 在GPFS中列出檔案以驗證NFS用戶端。

```
[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]#
```

設定**NFS**用戶端

若要設定NFS用戶端、請完成下列步驟：

1. 在NFS用戶端中安裝套件。

```
[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. 啟動NFS用戶端服務。

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

3. 透過NFS用戶端上的NFS傳輸協定掛載GPFS。

```
[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. 驗證NFS掛載資料夾中的GPFS檔案清單。

```
[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. 使用XCP將資料從GPFS匯出的NFS移至NetApp NFS。

```
[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. 驗證NFS用戶端上的GPFS檔案。

```
[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]#
```

從MapR-FS移轉ONTAP 到Sfnfs

本節提供ONTAP 使用NetApp XCP將MapR-FS資料移至支援NFS所需的詳細步驟。

1. 為每個MapR節點配置三個LUN、並賦予所有MapR節點的LUN擁有權。
2. 在安裝期間、為MapR叢集磁碟選擇新增的LUN、以用於MapR-FS。
3. 根據安裝MapR叢集 "MapR 6.1文件"。
4. 使用MapReduce命令（例如「Hadoop Jar xxx」）檢查基本Hadoop作業。
5. 將客戶資料保留在MapR-FS中。例如、我們使用Teragen在MapR-FS中產生約1 TB的樣本資料。
6. 將MapR-FS設定為NFS匯出。
   a. 停用所有MapR節點上的nLockmanager服務。

```
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. 在所有MapR節點的「/opt/MapR/conf/exports」檔案中、從MapR-FS匯出特定資料夾。匯出子資料夾時、請勿以不同權限匯出父資料夾。

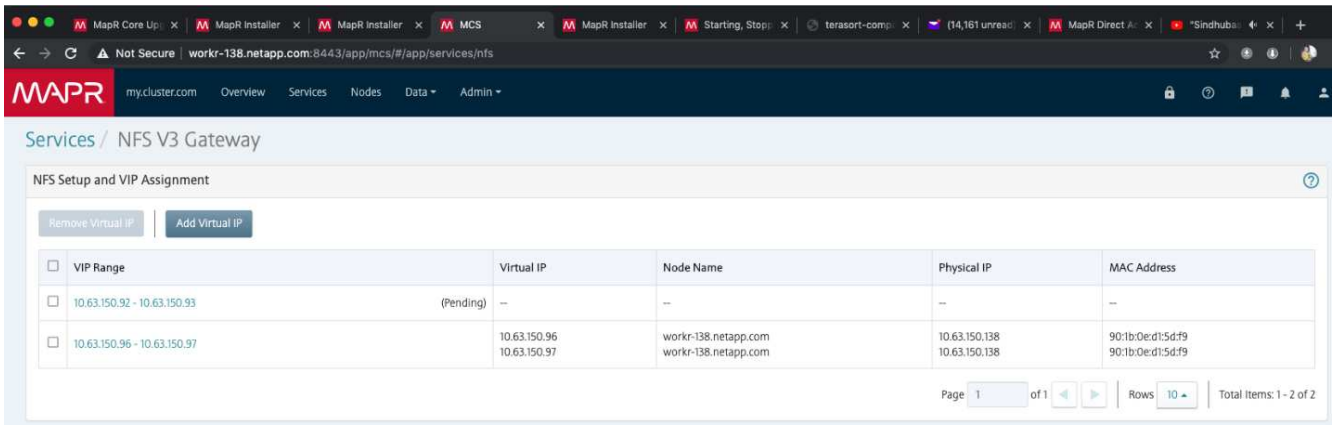
```
[mapr@workr-138 ~]$ cat /opt/mapr/conf/exports
# Sample Exports file
# for /mapr exports
# <Path> <exports_control>
#access_control -> order is specific to default
# list the hosts before specifying a default for all
#  a.b.c.d,1.2.3.4(ro) d.e.f.g(ro) (rw)
#  enforces ro for a.b.c.d & 1.2.3.4 and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

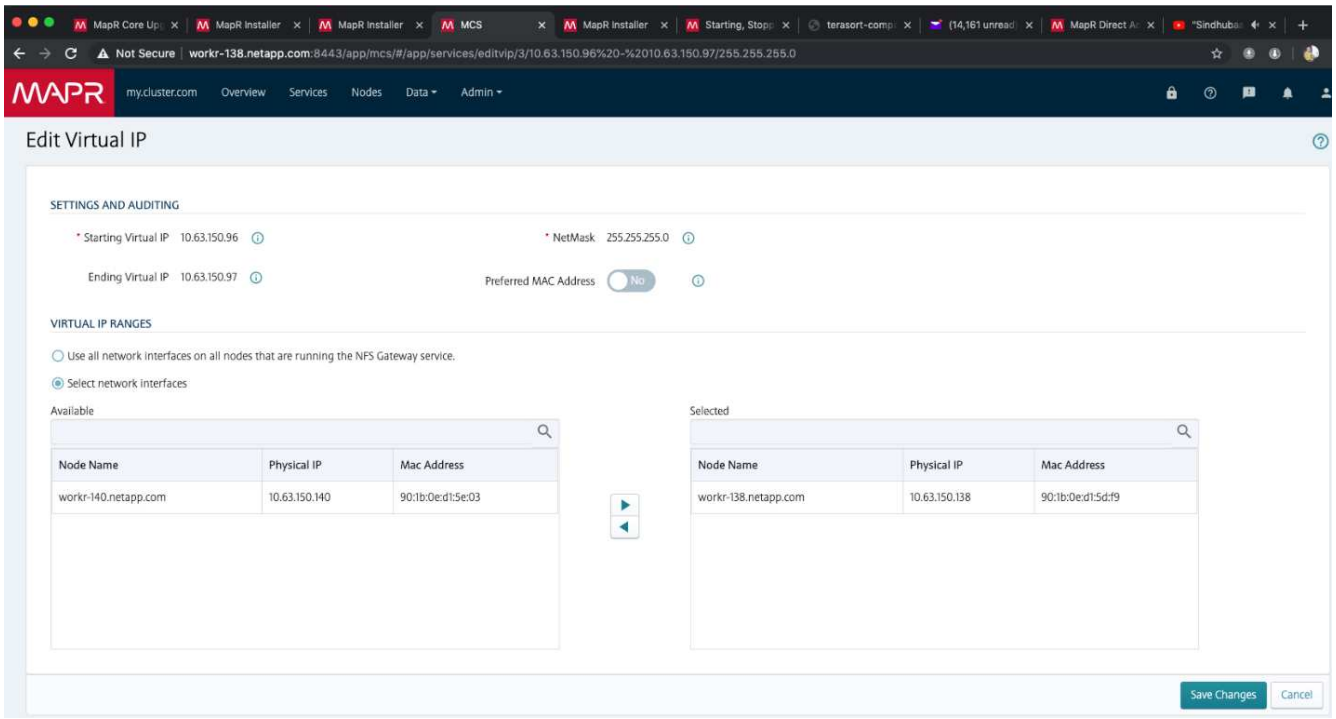
7. 重新整理MapR-FS NFS服務。

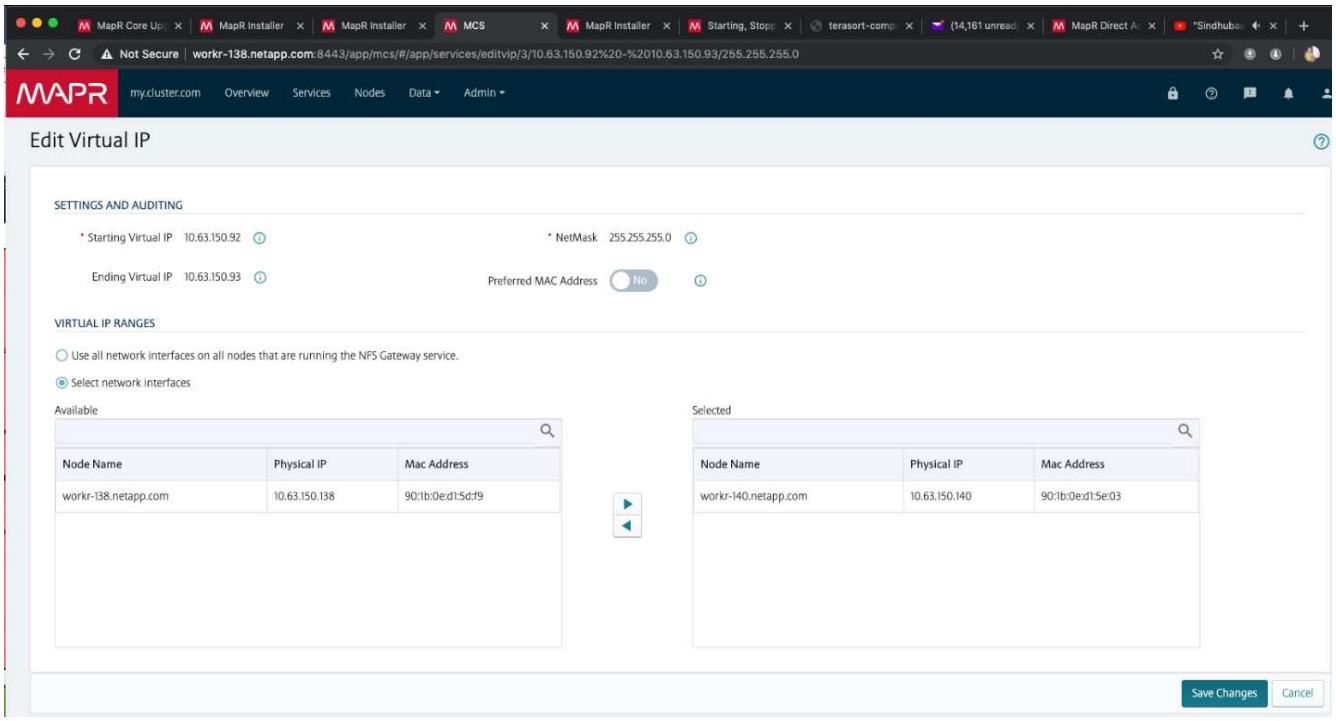
```
root@workr-138: tmp$ maprcli nfsmgmt refreshexports
ERROR (22) -  You do not have a ticket to communicate with
127.0.0.1:9998. Retry after obtaining a new ticket using maprlogin
root@workr-138: tmp$ su - mapr
[mapr@workr-138 ~]$ maprlogin password -cluster my.cluster.com
[Password for user 'mapr' 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. 將虛擬IP範圍指派給MapR叢集中的特定伺服器或一組伺服器。然後、MapR叢集會指派IP給特定伺服器、以供NFS資料存取。這些IP可實現高可用度、也就是說、如果某個伺服器或網路發生特定IP故障、IP範圍的下一個IP可用於NFS存取。

> 如果您想要從所有的MapR節點提供NFS存取、則可以將一組虛擬IP指派給每個伺服器、並使用每個MapR節點的資源進行NFS資料存取。







9. 檢查每個MapR節點上指派的虛擬IP、並將其用於NFS資料存取。

```
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. 使用指派的虛擬IP來掛載NFS匯出的MapR-FS、以檢查NFS作業。不過、使用NetApp XCP進行資料傳輸時、不需要執行此步驟。

```
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. 設定NetApp XCP、將資料從MapR-FS NFS閘道傳輸到ONTAP 靜態NFS。
    a. 設定XCP的目錄位置。

```
[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. 將授權檔案複製到「/opt /NetApp/xFiles/XCP /」。

```
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. 使用「XCP activate」命令啟動XCP。

d. 檢查NFS匯出的來源。

```
[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. 使用XCP從多個來源IP和多個目的地IP ONTAP （亦即多個IP）的多個MapR節點傳輸資料。

```
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. 檢查儲存控制器上的負載分配。

```
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
```

何處可找到其他資訊

若要深入瞭解本文所述資訊、請檢閱下列文件和 / 或網站：

- NetApp就地分析模組最佳實務做法

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

- NetApp FlexGroup 《NetApp》《最佳實務做法與實作指南》

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

- NetApp 產品文件

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

Conflent Kafka的最佳實務做法

TR-4912：NetApp的Conflent Kafka階層式儲存設備最佳實務準則

Karthikeyan Nagalingam、Joseph Kandatilparambil、NetApp Rankesh Kumar、Conflent

Apache Kafka是一個社群分散式事件串流平台、能夠每天處理數兆次的事件。最初構想為訊息佇列的Kafka是以分散式提交記錄的抽象化為基礎。自從LinkedIn於2011年建立並開放原始碼之後、Kafka便從訊息佇列發展成功能完善的活動串流平台。Conflent提供Apache Kafka的Conflent Platform發佈版本。Conflent Platform為卡夫卡（Kafka）提供額外的社群和商業功能、旨在提升營運者和開發人員大規模上線的串流體驗。

本文件說明在NetApp物件儲存產品上使用ConFluent分層儲存設備的最佳實務準則、提供下列內容：

- 運用NetApp物件儲存設備進行一致驗證–NetApp StorageGRID 產品特色
- 階層式儲存效能測試
- NetApp儲存系統上的Conflent最佳實務準則

為何要使用一致的分層儲存設備？

Connent已成為許多應用程式的預設即時串流平台、尤其是巨量資料、分析和串流工作負載。階層式儲存設備可讓使用者在ConFluent平台中、將運算與儲存區分開。它可讓儲存資料更具成本效益、讓您儲存幾乎無限量的資料、並可隨需擴充（或縮減）工作負載、讓資料和租戶重新平衡等管理工作變得更輕鬆。S3相容的儲存系統可利用所有這些功能、在同一個位置將所有事件的資料民主化、免除複雜資料工程的需求。如需瞭解為何應使用卡夫卡階層式儲存設備的詳細資訊、請查看 "本文作者：Conflent"。

為何選擇**NetApp StorageGRID** 解決方案來進行階層式儲存？

NetApp是領先業界的物件儲存平台。StorageGRID支援業界標準物件API（包括Amazon Simple Storage Service（S3）API）的軟體定義物件式儲存解決方案。StorageGRID可大規模儲存及管理非結構化資料、提供安全且持久的物件儲存。StorageGRID內容會放在適當的位置、適當的時間、以及適當的儲存層、以最佳化工作流程、並降低全球分散式多媒體的成本。

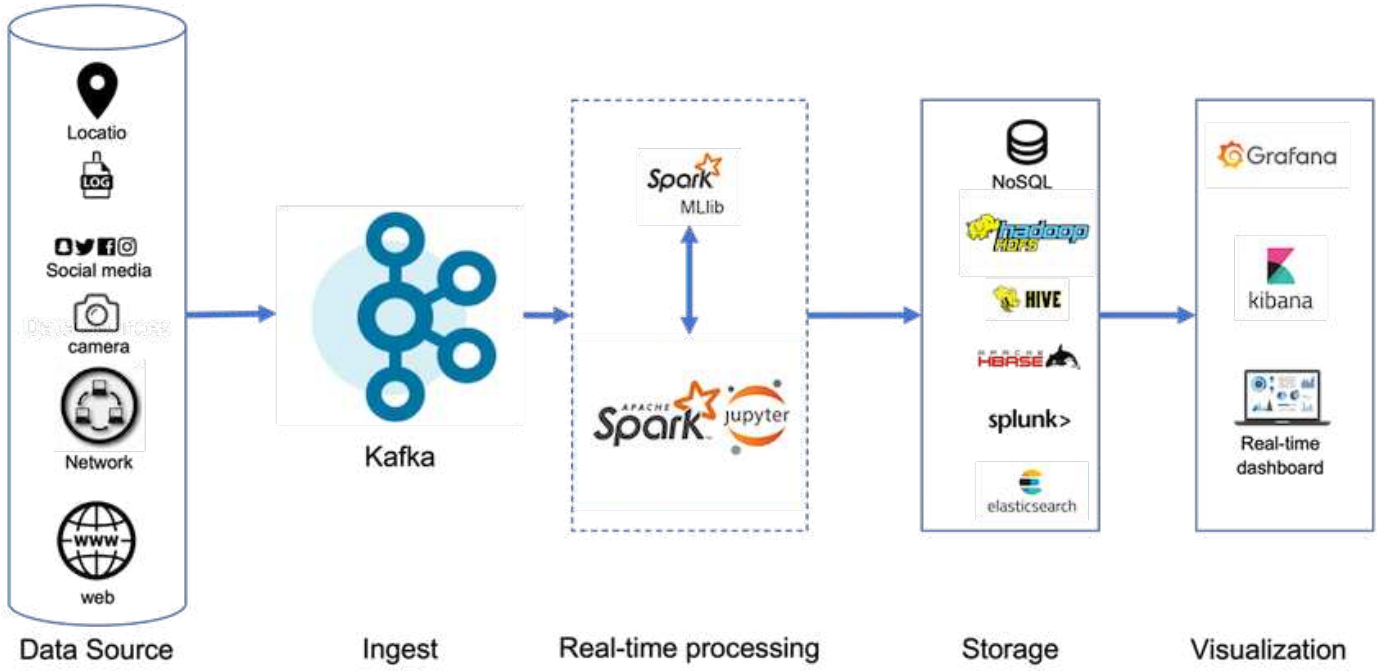
與眾不同之處在於StorageGRID 其資訊生命週期管理（ILM）原則引擎、可實現原則導向的資料生命週期管理。原則引擎可以使用中繼資料來管理整個生命週期內的資料儲存方式、以期一開始就最佳化效能、並自動最佳化資料存留期的成本與持久性。

實現一致的分層儲存

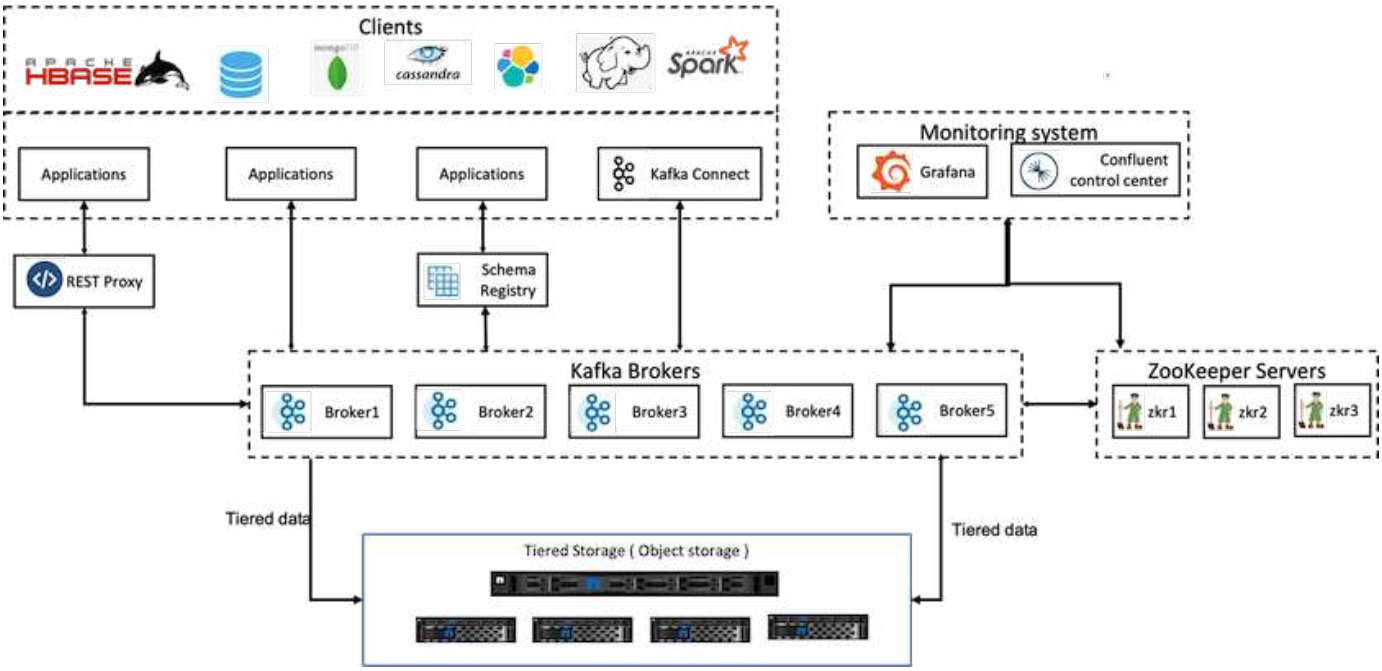
階層式儲存的基本概念是將資料儲存的工作與資料處理區分開。如此分離之後、資料儲存層和資料處理層就變得更加容易獨立擴充。

Conflent的階層式儲存解決方案必須面對兩個因素。首先、IT必須解決或避免常見的物件存放區一致性和可用度內容、例如清單作業不一致、偶爾物件無法使用。第二、IT必須正確處理階層式儲存設備與Kafka複寫與容錯模式之間的互動、包括殭屍領導廠商可能繼續分層調整偏移範圍。NetApp物件儲存設備提供一致的物件可用度和HA模式、讓舊舊儲存設備可用於層級偏移範圍。NetApp物件儲存設備提供一致的物件可用度、以及HA模式、可讓舊舊儲存設備用於層級偏移範圍。

透過階層式儲存設備、您可以使用高效能平台、在串流資料的尾端附近進行低延遲的讀取和寫入、也可以使用更便宜、可擴充的物件存放區（例如NetApp StorageGRID 流通量高的歷史讀取）。我們也提供適用於Spark with NetApp儲存控制器的技術解決方案、詳情請見此處。下圖顯示Kafka如何融入即時分析管道。



下圖說明NetApp StorageGRID 的物件儲存層如何融入ConFluent Kafka的物件儲存層。



解決方案架構詳細資料

本節涵蓋用於ConFluent驗證的硬體與軟體。此資訊適用於透過NetApp儲存設備進行的ConFluent Platform部署。下表涵蓋已測試的解決方案架構和基礎元件。

解決方案元件	詳細資料
Connent Kafka 6.2版	• 三位Zookkeepers • 五個代理伺服器 • 五種工具伺服器 • 單一Grafana • 單一控制中心
Linux（Ubuntu 18.04）	所有伺服器
適用於StorageGRID 階層式儲存的NetApp解決方案	• 軟件StorageGRID • 1個SG1000（負載平衡器） • 4個SGF6024 • 4 x 24 x 800 SSD • S3傳輸協定 • 4 x 100GbE（代理StorageGRID 程式與實例之間的網路連線）
15部Fujitsu PRIMERGY RX2540伺服器	每個配備：* 2個CPU、總共16個實體核心* Intel Xeon * 256GB實體記憶體* 100GbE雙埠

技術總覽

本節說明本解決方案所使用的技術。

NetApp StorageGRID

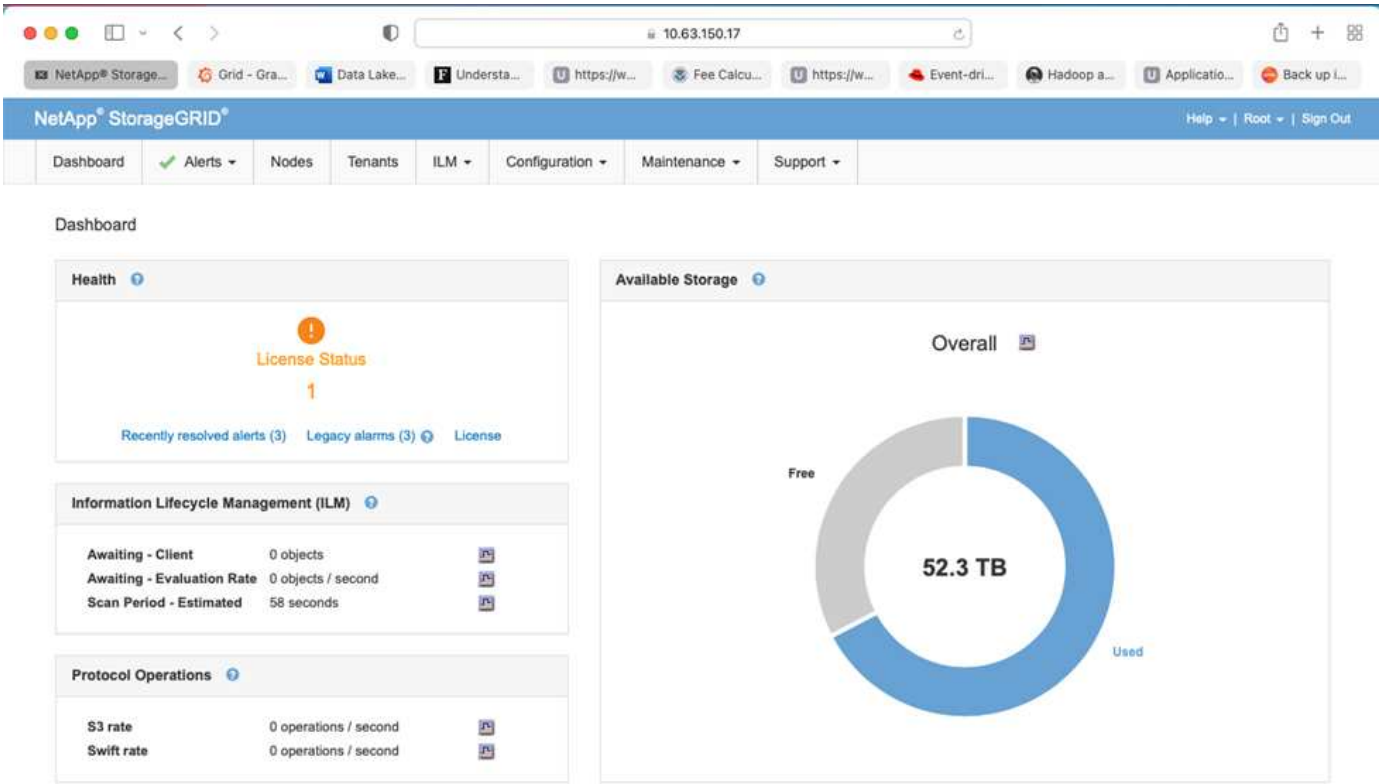
NetApp StorageGRID 產品是高效能且具成本效益的物件儲存平台。透過階層式儲存設備、儲存在本機儲存設備或代理程式SAN儲存設備上的Confluent Kafka上的大部分資料都會卸載到遠端物件存放區。此組態可減少重新平衡、擴充或縮減叢集或更換故障的代理程式所需的時間與成本、進而大幅改善營運。物件儲存在管理位於物件存放區層的資料方面扮演著重要角色、因此挑選適當的物件儲存非常重要。

利用分散式節點型網格架構、提供智慧型原則導向的全球資料管理功能。StorageGRID它透過無所不在的全域物件命名空間、加上精密的資料管理功能、簡化PB非結構化資料和數十億個物件的管理。單一通話物件存取功能可延伸至各個站台、並簡化高可用度架構、同時確保無論站台或基礎架構停機、都能持續存取物件。

多租戶功能可在同一個網格內安全地維護多個非結構化雲端和企業資料應用程式、進而提高NetApp StorageGRID 的ROI和使用案例。您可以利用中繼資料導向的物件生命週期原則來建立多個服務層級、以最佳化跨多個地理區的持久性、保護、效能和位置。使用者可以調整資料管理原則、並監控及套用流量限制、以便在瞬息萬變的IT環境中、隨著需求變化而不中斷地重新調整資料環境。

利用**Grid Manager**輕鬆管理

透過瀏覽器型的圖形介面StorageGRID 、您可以在StorageGRID 單一窗口中、設定、管理及監控分散於全球各地的整個系統。



您可以使用StorageGRID 「資訊網管理程式」介面執行下列工作：

- 管理分散在全球各地、PB規模的物件儲存庫、例如影像、視訊和記錄。
- 監控網格節點和服務、確保物件可用度。
- 使用資訊生命週期管理（ILM）規則、管理物件資料隨時間擺放的位置。這些規則可控制物件擷取後的資料處理方式、資料保護方式、資料儲存位置、以及資料儲存時間。
- 監控系統內的交易、效能和作業。

資訊生命週期管理原則

根據特定的效能和資料保護需求、支援靈活的資料管理原則、包括保留物件的複本複本、以及使用EC（銷毀編碼）配置（例如2+1和4+2）來儲存物件。StorageGRID隨著工作負載和需求隨時間變化、ILM原則也必須隨時間而改變、這是很常見的做法。修改ILM原則是一項核心功能、讓StorageGRID 客戶能夠快速輕鬆地因應瞬息萬變的環境。請檢查 "ILM原則" 和 "ILM規則" 安裝於此。StorageGRID

效能

透過新增更多儲存節點（例如VM、裸機或特定用途的應用裝置）來擴充效能StorageGRID "SG5712、SG5760、SG6060或SGF6024"。在我們的測試中、我們使用SGF6024應用裝置、以最小尺寸的三節點網格、超越Apache Kafka的關鍵效能要求。隨著客戶使用額外的代理商來擴充其Kafka叢集、他們可以新增更多儲存節點來提升效能和容量。

負載平衡器和端點組態

本功能提供Grid Manager UI（使用者介面）和REST API端點、StorageGRID 以檢視、設定及管理StorageGRID 您的不實系統、以及稽核記錄來追蹤系統活動。為了提供高可用度的S3端點給ConFluent Kafka階層式儲存設備、我們實作StorageGRID 了一套以服務形式在管理節點和閘道節點上執行的負載平衡器。此外、負載平衡器也會管理本機流量、並與GSLB（全域伺服器負載平衡）對話、以協助災難恢復。

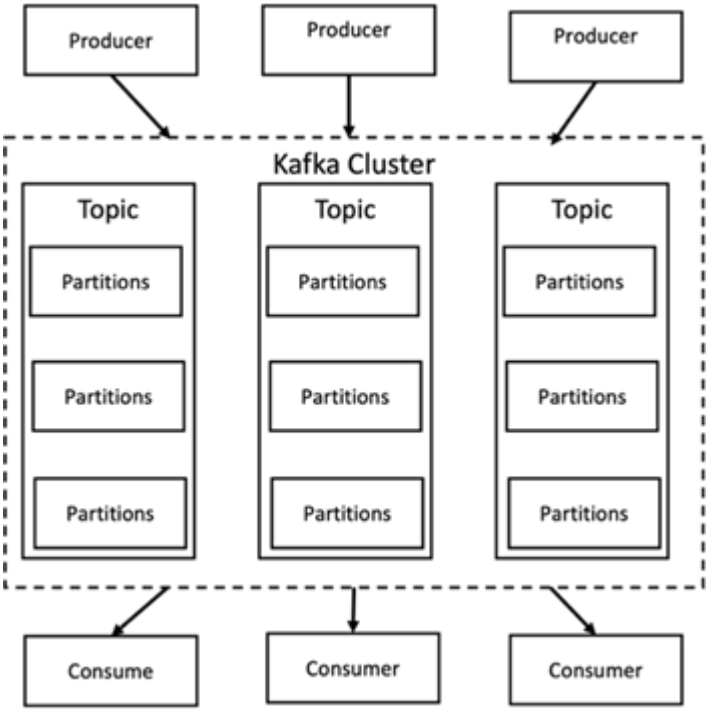
為了進一步強化端點組態、StorageGRID 支援內建於管理節點的流量分類原則、讓您監控工作負載流量、並將各種服務品質（QoS）限制套用至工作負載。流量分類原則會套用至StorageGRID 閘道節點和管理節點的「動態負載平衡器」服務上的端點。這些原則可協助流量調整和監控。

流量分類**StorageGRID**

包含內建QoS功能。StorageGRID流量分類原則可協助監控來自用戶端應用程式的不同類型S3流量。然後您可以建立並套用原則、根據輸入/輸出頻寬、讀取/寫入並行要求數或讀取/寫入要求率來限制此流量。

Apache Kafka

Apache Kafka是以Java和Scala寫入串流處理的軟體匯流排架構實作。它旨在提供統一化、高處理量、低延遲的平台、以處理即時資料饋送。卡夫卡可連線至外部系統、以便透過Kafka Connect匯出及匯入資料、並提供Kafka串流處理程式庫、即Java串流處理程式庫。Kafka使用以TCP為基礎的二進位傳輸協定、針對效率最佳化、並仰賴「訊息集」抽象化、將訊息自然地分組在一起、以降低網路往返的成本。如此一來、就能進行更大的連續磁碟作業、較大的網路封包和鄰近的記憶體區塊、進而讓Kafka將一串爆發性的隨機訊息寫入串流變成線性寫入。下圖說明Apache Kafka的基本資料流程。



Kafka儲存的關鍵價值訊息來自於任意數量的稱為「生產商」的程序。資料可分割成不同主題中的不同分割區。在磁碟分割內、訊息會依照偏移量（訊息在磁碟分割內的位置）嚴格排序、並與時間戳記一起索引及儲存。其他稱為「使用者」的程序也可以從分割區讀取訊息。對於串流處理、Kafka提供的STREAMS API可讓您寫入Java應用程式、以使用Kafka的資料、並將結果寫回Kafka。Apache Kafka也可搭配外部串流處理系統使用、例如Apache Apex、Apache Flink、Apache Spark、Apache Storm及Apache NiFI。

Kafka會在一個或多個伺服器（稱為代理程式）的叢集上執行、所有主題的分割區會分散到叢集節點。此外、分

割區也會複寫到多個代理程式。此架構可讓Kafka以容錯的方式提供大量訊息串流、並讓它取代一些傳統的訊息系統、例如Java Message Service（JMS）、進階訊息佇列傳輸協定（AMQP）等。自0.11.0.0發行以來、Kafka提供交易式寫入功能、使用STREAMS API提供一次完整的串流處理。

卡夫卡支援兩種主題：一般主題和精簡主題。一般主題可設定保留時間或空間限制。如果有記錄超過指定的保留時間、或磁碟分割超出空間限制、則Kafka可以刪除舊資料以釋放儲存空間。根據預設、主題的保留時間設定為7天、但也可以無限期儲存資料。對於精簡主題、記錄不會因時間或空間界限而過期。相反地、Kafka會將稍後的訊息視為舊訊息的更新、並保證不會刪除每個金鑰的最新訊息。使用者可以撰寫所謂的tombstone訊息、並針對特定金鑰使用null值、以完全刪除訊息。

卡夫卡有五大API：

- *監製API。*允許應用程式發佈記錄串流。
- *消費者API。*允許應用程式訂閱主題和處理記錄串流。
- *連接器API.*執行可重複使用的生產商與消費者API、這些API可將主題連結至現有的應用程式。
- *串流API.*此API會將輸入串流轉換成輸出、並產生結果。
- *管理API。*用於管理Kafka主題、代理人及其他Kafka物件。

消費者與製造商API以Kafka訊息傳輸協定為基礎、為Java中的Kafka消費者與製造商用戶端提供參考實作。基礎訊息傳輸協定是一種二進位傳輸協定、開發人員可用來以任何程式設計語言撰寫自己的消費者或生產用戶端。這可讓Kafka從Java虛擬機器（JVM）生態系統中解放。Apache Kafka wikis會維護可用的非Java用戶端清單。

Apache Kafka使用案例

Apache Kafka最受訊息、網站活動追蹤、指標、記錄彙總、串流處理、 事件來源及提交記錄。

- Kafka的處理量、內建分割區、複寫及容錯能力均有提升、因此是大型訊息處理應用程式的理想解決方案。
- Kafka可以在追蹤管道中重建使用者的活動（頁面檢視、搜尋）、做為一組即時發佈訂閱摘要。
- Kafka經常用於營運監控資料。這包括彙總分散式應用程式的統計資料、以產生集中化的作業資料饋送。
- 許多人使用Kafka來取代記錄彙總解決方案。記錄集合通常會從伺服器收集實體記錄檔、並將它們放在中央位置（例如檔案伺服器或HDFS）進行處理。Kafka會將檔案詳細資料擷取出來、並將記錄或事件資料當作訊息串流來提供更簡潔的抽象化。如此可降低延遲處理、更輕鬆支援多個資料來源和分散式資料使用。
- 卡夫卡的許多使用者會處理由多個階段組成的管線資料、其中原始輸入資料會從卡夫卡主題中消耗、然後彙總、豐富或以其他方式轉化為新主題、以供進一步消費或後續處理。例如、推薦新聞文章的處理管道可能會從RSS摘要串流文章內容、然後將其發佈至「文章」主題。進一步處理可能會將此內容正規化或重複資料刪除、並將已清除的文章內容發佈至新主題、最後的處理階段可能會嘗試將此內容推薦給使用者。這類處理管道會根據個別主題、建立即時資料流程的圖表。
- 事件來源是應用程式設計的一種樣式、其狀態變更會記錄為依時間順序排列的記錄順序。卡夫卡支援非常大的儲存記錄資料、因此對於以這種風格建置的應用程式來說、它是絕佳的後端。
- Kafka可以做為分散式系統的外部提交記錄。此記錄有助於在節點之間複寫資料、並可做為故障節點還原資料的重新同步機制。Kafka的記錄壓縮功能有助於支援此使用案例。

Confluent

Conflent Platform是企業級平台、具備進階功能、可協助加速應用程式開發與連線、透過串流處理實現轉型、大規模簡化企業營運、並符合嚴苛的架構要求。Conflent是由Apache Kafka原創原創者所打造、以企業級功能擴展Kafka的優勢、同時免除Kafka管理或監控的負擔。如今、超過80%的財星雜誌100大企業都採用資料串流技術、大部分企業都使用Conflent技術。

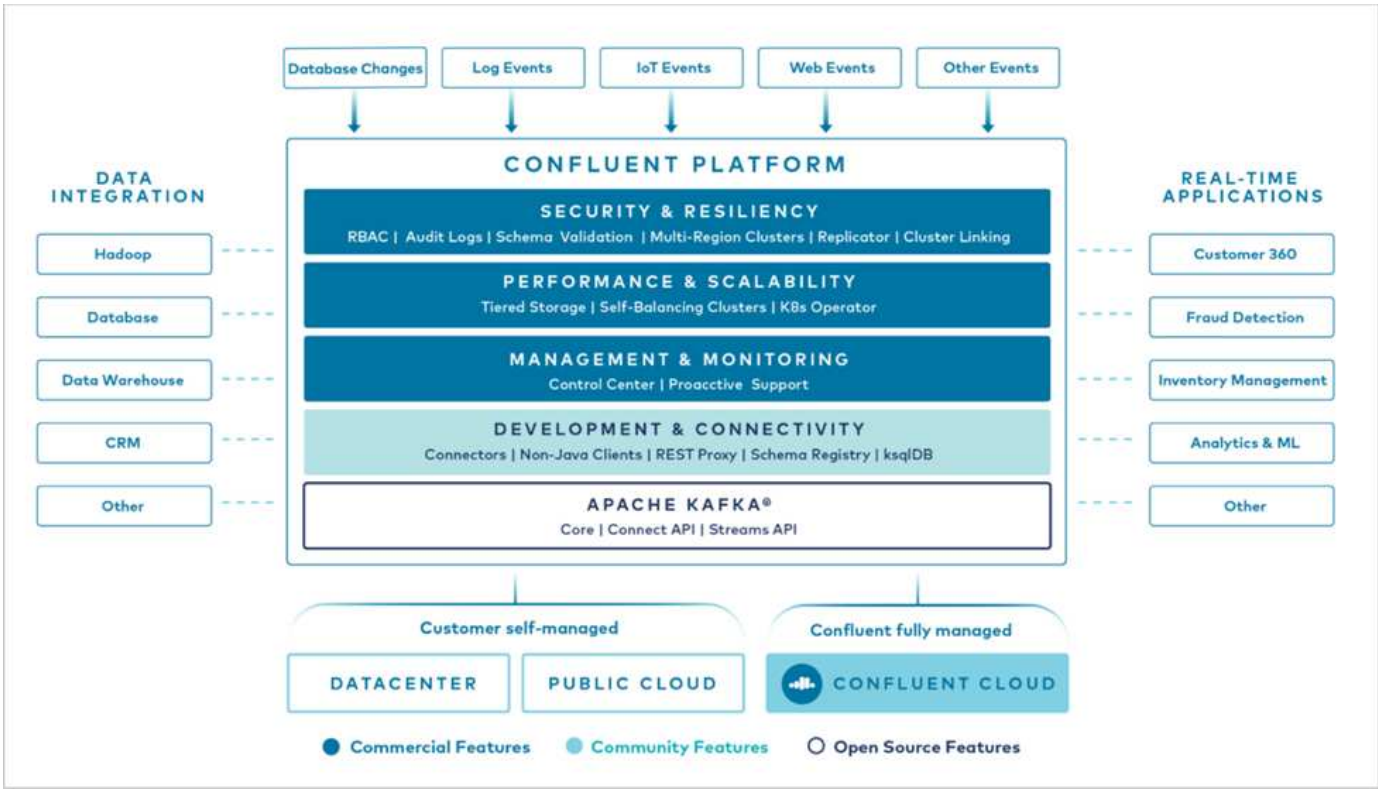
為何選擇**Conflent**？

藉由將歷史與即時資料整合至單一的集中式事實來源、Conflent可讓您輕鬆建置全新類別的現代化事件導向應用程式、取得通用資料管線、並以完整的擴充性、效能與可靠性、釋放強大的新使用案例。

什麼是**ConnFluent**的用途？

Conflent Platform可讓您專注於從資料中獲取商業價值、而非擔心基礎機制、例如資料如何在不同的系統之間傳輸或整合。具體而言、Conflent Platform可簡化資料來源與Kafka之間的連線、建置串流應用程式、以及保護、監控及管理Kafka基礎架構。如今、Conflent Platform可用於金融服務、全通路零售和自主汽車等多種產業的各種使用案例、以及詐欺偵測、 微服務和IoT。

下圖顯示ConnFluent Kafka平台元件。



Connent的事件串流技術總覽

在Conflent Platform的核心是 "Apache Kafka"是最受歡迎的開放原始碼分散式串流平台。卡夫卡的主要功能如下：

- 發佈及訂閱記錄串流。
- 以容錯的方式儲存記錄串流。
- 處理記錄串流。

隨裝即用的Conflent Platform也包括架構登錄、REST Proxy、總共100多個預先建置的Kafka連接器和ksqlDB。

Conflent平台的企業級功能總覽

- * Confluent Control Cent.*一種GUI型系統、用於管理及監控Kafka。它可讓您輕鬆管理Kafka Connect、以及建立、編輯及管理與其他系統的連線。

- * Kubernetes的Conflent。* Kubernetes的Connent是Kubernetes營運者。Kubernetes營運者提供特定平台應用程式的獨特功能和需求、藉此擴充Kubernetes的協調功能。對於Conflent Platform、這包括大幅簡化Kubernetes上的Kafka部署程序、以及自動化典型的基礎架構生命週期工作。
- *連接至Kafka的Confluent連接器*連接器使用Kafka Connect API將Kafka連接至其他系統、例如資料庫、金鑰值儲存區、搜尋索引和檔案系統。Conflent Hub提供可下載的連接器、適用於最受歡迎的資料來源和接收器、包括這些連接器的完整測試和支援版本、以及Conflent Platform。如需詳細資料、請參閱 "請按這裡"。
- *自我平衡叢集。*提供自動負載平衡、故障偵測及自我修復功能。它支援視需要新增或汰換代理商、無需手動調校。
- * Confluent叢集連結。*直接將叢集連線在一起、並透過連結橋接器將主題從一個叢集鏡射到另一個叢集。叢集連結可簡化多資料中心、多叢集及混合雲部署的設定。
- * Confluent自動資料平衡器。*監控叢集的代理程式數量、分割區大小、分割區數目、以及叢集內的領導者數量。它可讓您將資料移轉至整個叢集、以建立平均工作負載、同時節流重新平衡流量、將對正式作業工作負載的影響降至最低、同時重新平衡。
- * Confluent replicator。*讓您在多個資料中心中維護多個Kafka叢集變得比以往更輕鬆。
- *分層儲存。*提供使用您最喜愛的雲端供應商儲存大量Kafka資料的選項、藉此降低營運負擔和成本。透過階層式儲存設備、您只能在需要更多運算資源時、將資料保存在具成本效益的物件儲存設備上、並擴充代理商。
- * Connent Jms用戶端。* Conflent Platform包含適用於Kafka的與Jms相容的用戶端。此Kafka用戶端實作了JMS 1.1標準API、使用Kafka Brokers做為後端。如果您使用的是使用Jms的舊應用程式、而且想要以Kafka取代現有的Jms訊息代理程式、這項功能就很實用。
- * Conflent MQtT Proxy。*提供一種從MQtT裝置和閘道直接發佈資料至Kafka的方法、而不需要中間的MQtT代理程式。
- * Confluent安全外掛程式。* Confluent安全外掛程式可用來新增各種Confluent Platform工具和產品的安全功能。目前有一個外掛程式可供Conflent REST Proxy使用、可協助驗證傳入要求、並將驗證的主體傳播至向Kafka的要求。這可讓Conflent REST Proxy用戶端利用Kafka代理程式的多租戶安全功能。

一致驗證

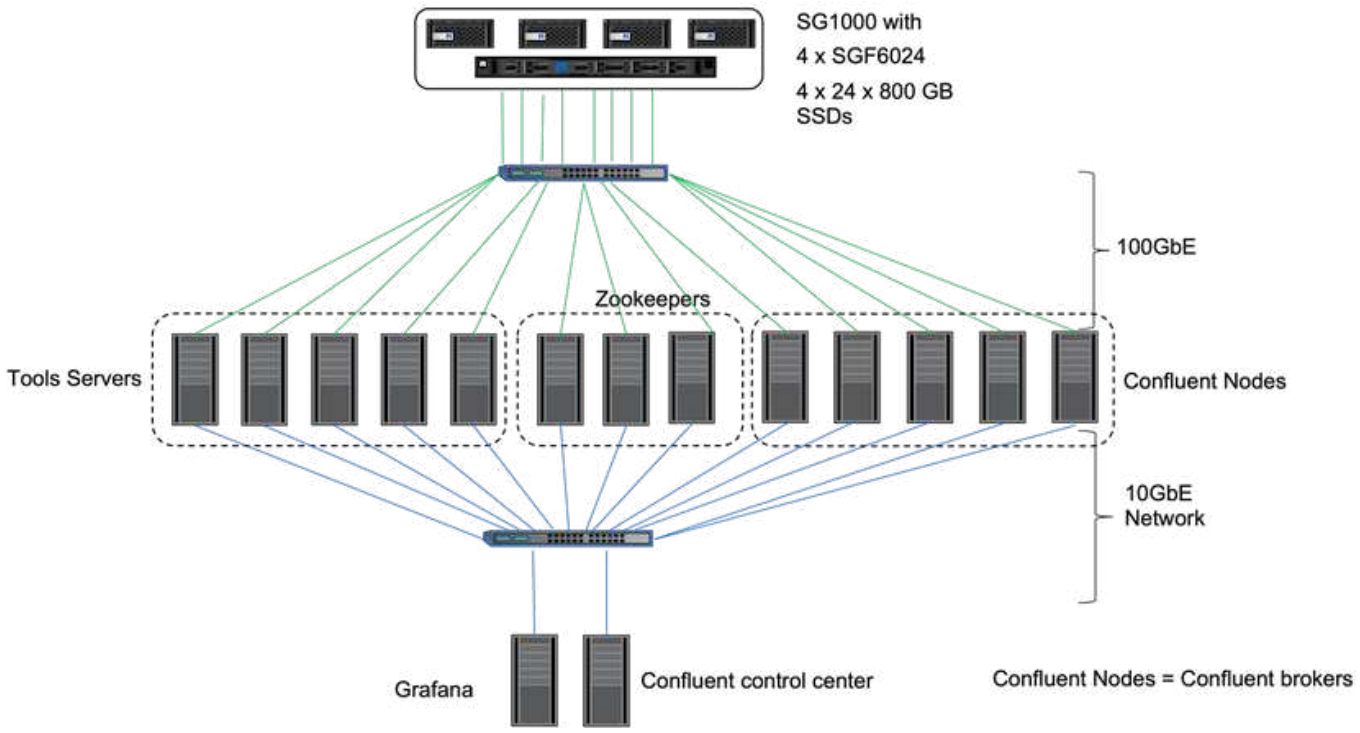
我們在NetApp StorageGRID 的《Consent Platform 6.2多層式儲存設備》中執行驗證。NetApp與Conflent團隊共同進行這項驗證、並執行驗證所需的測試案例。

ConFluent平台設定

我們使用下列設定進行驗證。

為了驗證、我們使用三個zookeepers、五個代理商、五個執行伺服器的測試指令碼、256GB RAM的命名工具伺服器、以及16個CPU。對於NetApp儲存設備、StorageGRID 我們使用的是搭載四個SGF6024s的Sg1000負載平衡器。儲存設備與代理商均透過100GbE連線進行連線。

下圖顯示用於ConFluent驗證的組態網路拓撲。



這些工具伺服器可做為將要求傳送至Confluent節點的應用程式用戶端。

一致的階層式儲存組態

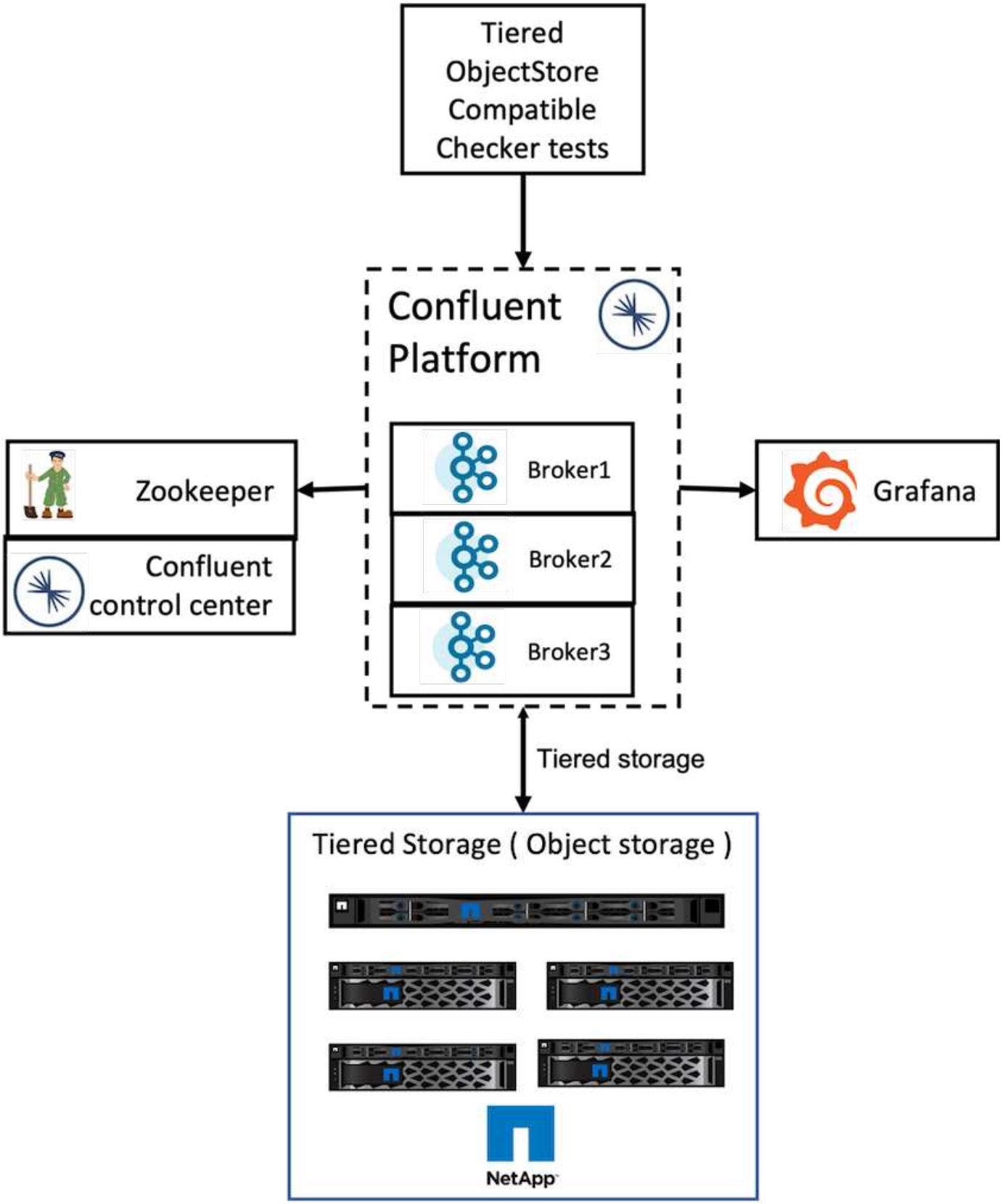
階層式儲存組態需要卡夫卡的下列參數：

```
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
```

為了驗StorageGRID 證、我們使用了搭配HTTP傳輸協定的功能、但HTTPS也能正常運作。存取金鑰和秘密金鑰會儲存在「confine.Tier .s3.cred.file.path」參數中提供的檔案名稱中。

NetApp物件儲存- StorageGRID 功能

我們將單一站台組態設定為StorageGRID 供驗證之用。



驗證測試

我們完成下列五個驗證測試案例。這些測試是在Trogdor架構上執行。前兩項是功能測試、其餘三項是效能測試。

物件存放區正確性測試

此測試可判斷物件存放區API上的所有基本作業（例如、Get/PUT / DELETE）是否能根據階層式儲存設備的需求順利運作。這是一項基本測試、每個物件存放區服務都應該在下列測試之前通過。這是一項堅定的測試、無論通過或失敗。

分層功能正確性測試

這項測試可判斷端點對端點階層式儲存功能是否與通過或失敗的權證測試搭配運作良好。測試會建立測試主題、預設設定為啟用分層、並大幅減少熱設定大小。它會產生事件串流至新建立的測試主題、等待代理程式將區段歸檔至物件存放區、然後使用事件串流、並驗證耗用的串流是否符合產生的串流。可設定產生給事件串流的訊息數量、讓使用者根據測試需求產生足夠大的工作負載。減少的熱集大小可確保使用者只能從物件存放區取得作用中區段以外的擷取、這有助於測試物件存放區的讀取正確性。我們已執行這項測試、且不需要進行物件存放區故障注入。我們在StorageGRID 其中一個節點停止服務管理程式服務、並驗證端點對端點功能是否可與物件儲存搭配運作、藉此模擬節點故障。

階層擷取基準測試

此測試可驗證階層式物件儲存設備的讀取效能、並在基準測試產生的區段負載過重時、檢查範圍擷取讀取要求。在這個基準測試中、Connent開發了自訂用戶端、以滿足層級擷取要求。

產生消耗的工作負載基準測試

此測試透過區段歸檔、間接在物件存放區上產生寫入工作負載。讀取工作負載（區段讀取）是在使用者群組擷取區段時、從物件儲存區產生的。此工作負載是由測試指令碼產生。此測試會檢查平行執行緒中物件儲存設備的讀取和寫入效能。我們在進行分層功能正確性測試時、測試了是否有物件存放區故障注入。

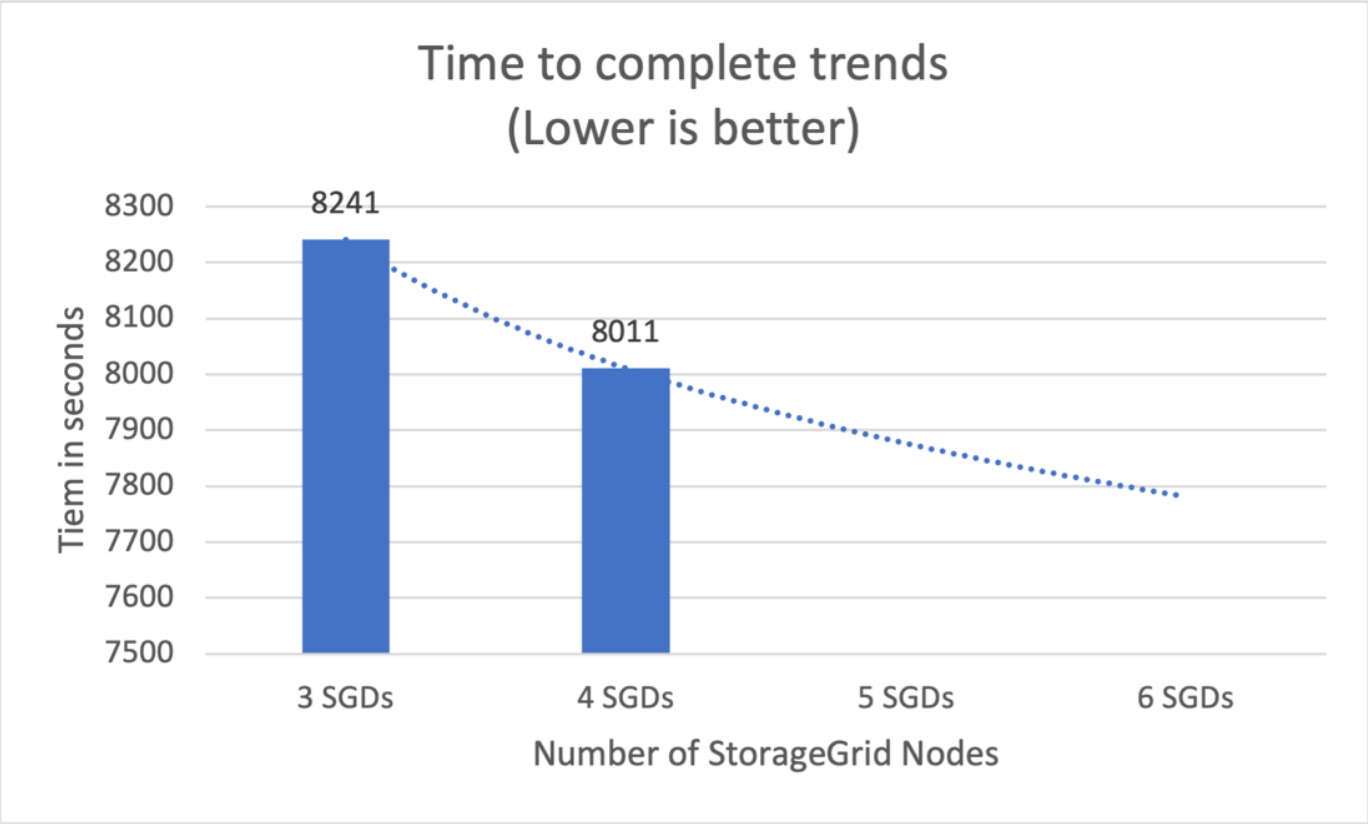
保留工作負載基準測試

此測試檢查了物件存放區在繁重主題保留工作負載下的刪除效能。保留工作負載是使用測試指令碼產生、此指令碼會產生許多訊息、並與測試主題平行。測試主題是以積極的大小型和時間型保留設定來設定、導致事件串流持續從物件存放區中清除。然後將區段歸檔。這導致代理程式在物件儲存區中刪除大量內容、並收集物件存放區刪除作業的效能。

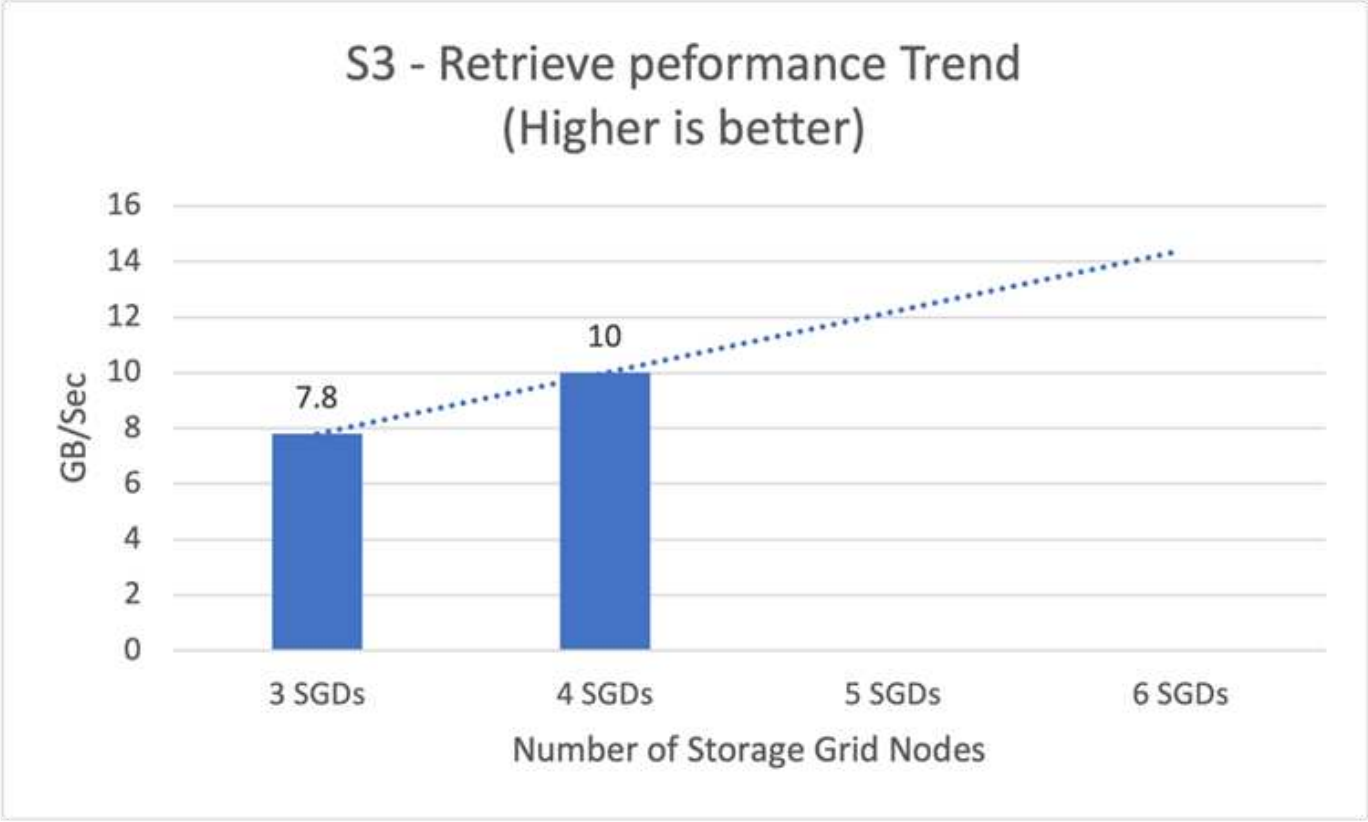
可擴充性的效能測試

我們利用NetApp StorageGRID 的支援功能、針對生產與消費型工作負載執行三到四個節點的階層式儲存測試。根據我們的測試、完成時間和效能結果與StorageGRID 各個節點的數量直接成比例。此功能需要至少三個節點。StorageGRID

- 當儲存節點數量增加時、完成產品和消費者作業的時間會線性縮短。



- S3擷取作業的效能會根據StorageGRID 各個節點的數量而線性提升。支援多達200個StorgeGRID節點。StorageGRID



Connent S3連接器

Amazon S3 Sink Connector會以Avro、Json或位元組格式、將Apache Kafka主題的資料匯出至S3物件。Amazon S3 Sink連接器會定期輪詢Kafka的資料、然後再將資料上傳至S3。分區工具用於將每個Kafka分區的資料分割成區塊。每一區塊的資料都會以S3物件表示。金鑰名稱會編碼主題、Kafka分割區、以及此資料區塊的起始偏移。

在此設定中、我們會示範如何使用Kafka S3接收器連接器、直接從Kafka讀取和寫入物件儲存區中的主題。在這項測試中、我們使用獨立的ConFluent叢集、但此設定適用於分散式叢集。

1. 從Conflent網站下載Conflent Kafka。
2. 將套件解壓縮至伺服器上的資料夾。
3. 匯出兩個變數。

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

4. 對於獨立的Conflent Kafka設定、叢集會在「/tmp」中建立一個暫存根資料夾。它也會建立Zookkeeper、Kafka、架構登錄、Connect、ksql-server、 和控制中心資料夾、並從「$Confluent_home'複製各自的組態檔案。請參閱下列範例：

```
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. 設定Zookkeeper。如果您使用預設參數、則不需要變更任何內容。

```
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:~#
```

在上述組態中、我們更新了「伺服器」。xxx屬性。根據預設、您需要三位Zooka主管才能選擇Kafka。

6. 我們在「/tmp/confine.406980/zookeeper / data」中建立一個具有唯一ID的MyID檔案：

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

我們使用最後數量的IP位址作為MyID檔案。我們使用卡夫卡、連線、控制中心、卡夫卡、卡夫卡、 ksql-server和schema-registry組態。

7. 啟動Kafka服務。

```
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#
```

每個組態都有一個記錄資料夾、有助於疑難排解問題。在某些情況下、服務需要更多時間才能啟動。請確定所有服務均已啟動且正在執行。

8. 使用「confhube-hub」安裝Kafka Connect。

```
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#
```

您也可以使用「conlume-hub install confluentinc / Kafka-connect：s3：10.0.3」來安裝特定版本。

9. 依預設、「confluentinc、Kafka-connect、S3」安裝於「/data / conflue/conlume-6.2.0/share/conflum-hub-sue-subs/confluentinc、Kafka-Connect、S3」中。
10. 使用全新的「confluentinc - Kafka-connect - S3」來更新外掛程式路徑。

```
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. 停止Conflent服務並重新啟動。

```
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. 在「/root/.AWS/IDes’檔案中設定存取ID和秘密金鑰。

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

13. 確認鏟斗可到達。

```
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. 設定S3和Bucket組態的S3接收器內容檔。

```
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. 將幾筆記錄匯入S3儲存區。

```
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. 裝入S3接收器連接器。

```
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. 檢查S3接收器狀態。

```
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. 檢查記錄、確定S3接收器已準備好接受主題。

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

19. 請查看卡夫卡的主題。

```
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. 檢查S3儲存區中的物件。

```
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. 若要驗證內容、請執行下列命令、將每個檔案從S3複製到您的本機檔案系統：

```
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. 若要列印記錄、請使用avro-tools-1.11.0.1.jar（可在 "Apache歸檔"）。

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

ConFluent自我平衡叢集

如果您之前曾管理過Kafka叢集、您可能會熟悉手動將分割區重新指派給不同的代理人所帶來的挑戰、以確保整個叢集的工作負載平衡。對於部署較大Kafka的組織而言、重新安排大量資料可能會令人望而生畏、繁瑣且風險高、尤其是當任務關鍵型應用程式建置於叢集之上時。然而、即使是卡夫卡最小的使用案例、這項程序仍耗時且容易發生人為錯誤。

在實驗室中、我們測試了Confluent自我平衡叢集功能、此功能可根據叢集拓撲變更或負載不均而自動重新平衡。當節點故障或擴充節點需要跨代理程式重新平衡資料時、「Confluent reBalance」測試有助於測量新增代理程式的時間。在傳統的Kafka組態中、隨著叢集成長、要重新平衡的資料量會隨之增加、但在階層式儲存設備中、重新平衡的資料量僅限於少量資料。根據我們的驗證結果、在傳統的Kafka架構中重新平衡階層式儲存設備需要數秒鐘或數分鐘、並隨著叢集成長線性成長。

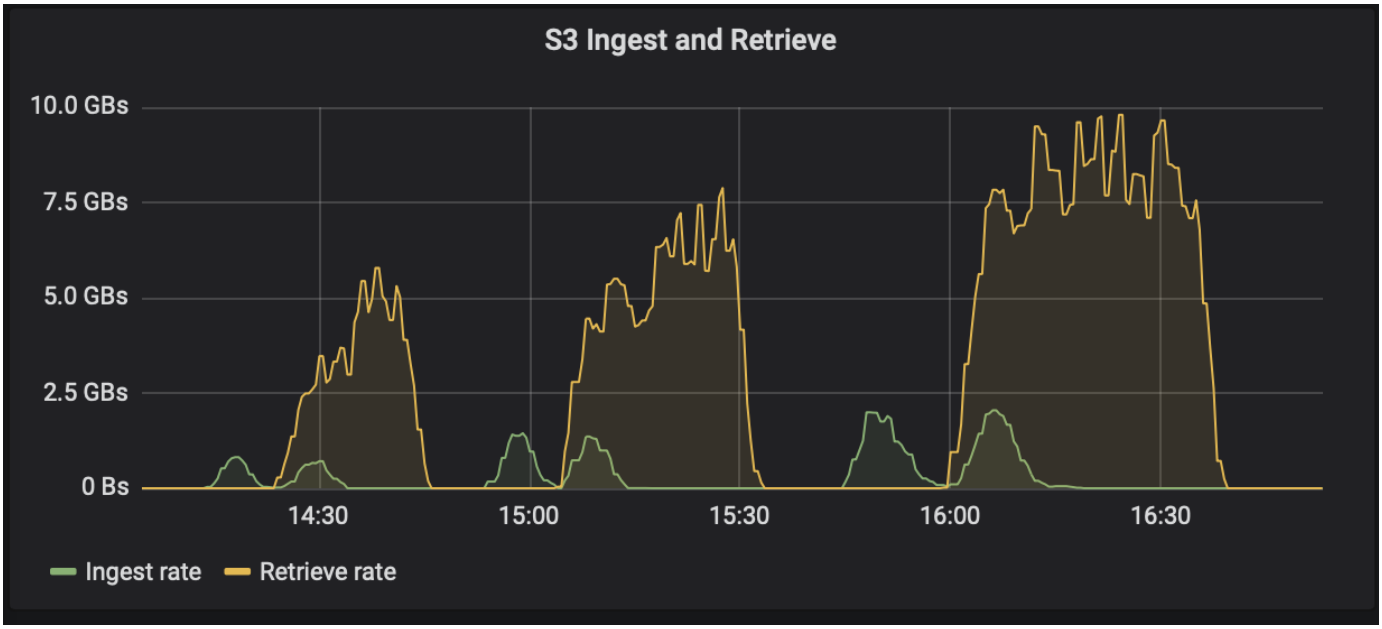
在自我平衡叢集中、磁碟分割重新平衡已完全自動化、可最佳化Kafka的處理量、加速代理程式擴充、並減輕執行大型叢集的作業負擔。在穩定狀態下、自我平衡叢集會監控所有代理程式的資料偏移、並持續重新分配分割區、以最佳化叢集效能。當垂直或向下擴充平台時、自我平衡叢集會自動辨識新的代理人是否存在、或移除舊的代理人、並觸發後續的分割區重新指派。這可讓您輕鬆新增及取消委任代理人、讓Kafka叢集更具彈性。這些效益不需要手動介入、複雜的數學運算、或是分割區重新指派通常會造成人為錯誤的風險。如此一來、資料重新平衡的完成時間就會大幅縮短、您可以專注於價值更高的事件串流專案、而不需要持續監督叢集。

最佳實務準則

本節將說明從本認證中學到的經驗教訓。

- 根據我們的驗證結果、S3物件儲存設備最適合Connent來保存資料。
- 我們可以使用高處理量SAN（特別是FC）來保留代理程式的熱資料或本機磁碟、因為在Conflent階層式儲存組態中、 代理資料目錄中的資料大小取決於資料移至物件儲存設備時的區段大小和保留時間。
- 當sege.bytes較高時、物件存放區可提供較佳的效能；我們測試了512MB。
- 在Kafka中、針對主題所產生的每筆記錄、其金鑰或值長度（以位元組為單位）是由「length、key、Value」（長度、金鑰、值）參數所控制。針對VMware、S3物件擷取和擷取效能提升至更高的價值。StorageGRID例如、512位元組提供5.8GBps擷取、而1024位元組提供7.5GBps S3擷取、以及提供接近10Gbps的2048位元組。

下圖顯示根據「length、key、Value」擷取和擷取S3物件。



- * Kafka調校。*若要提升階層式儲存設備的效能、您可以增加TierFetcherNumThreades和TierArchiverNum執行 緒。一般而言、您想要增加TierFetcherNum執行 緒、以符合實體CPU核心數量、並將TierArchiverNum執行 緒增加到CPU核心數量的一半。例如、在伺服器內容中、如果您的機器有八個實體核心、請將confucere.Tier .etcher。num.thread = 8和puceter.Tier。archiver.nm.thread = 4。
- *主題刪除的時間間隔。*刪除主題時、不會立即開始刪除物件儲存區中的記錄區段檔案。相反地、在刪除這些檔案之前、會有一個預設值為3小時的時間間隔。您可以修改組態confluent.tier.topic.delete.check.interval.ms來變更此時間間隔的值。如果您刪除主題或叢集、也可以手動刪除個別儲存區中的物件。
- *階層式儲存內部主題的ACL。*內部部署的建議最佳實務做法是讓ACL授權者能夠處理階層式儲存所使用的內部主題。設定ACL規則、僅限代理使用者存取此資料。如此可保護內部主題、防止未獲授權存取階層式儲存資料和中繼資料。

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

> 將使用者「<Kafka>'」取代為部署中的實際代理主體。

例如、命令「conflexe-層 狀態」會針對階層式儲存設備的內部主題設定ACL。目前只有一個與階層式儲存有關的內部主題。此範例會建立ACL、為內部主題的所有作業提供主要的Kafka權限。

規模調整

卡夫卡的規模調整可透過四種組態模式執行：簡單、精細、反轉和分割區。

簡單易用

簡易模式適用於首次使用Apache Kafka的使用者或早期狀態使用案例。在此模式下、您需要提供處理量MBps、讀取扇出、保留和資源使用率百分比（預設為60%）等需求。您也可以進入內部部署（裸機、VMware

、Kubernetes或OpenStack）或雲端等環境。根據這項資訊、Kafka叢集的規模可提供代理程式、zookeeper、Apache Kafka Connect工作人員、架構登錄、REST Proxy、ksqlDB及Confluent控制中心所需的伺服器數量。

對於階層式儲存設備、請考慮使用精細的組態模式來調整Kafka叢集的規模。精細模式適用於經驗豐富的Apache Kafka使用者或定義明確的使用案例。本節說明生產商、串流處理器及使用者的規模調整。

製造商

若要說明Apache Kafka的生產商（例如原生用戶端、REST Proxy或Kafka連接器）、請提供下列資訊：

- 名稱 Spark.
- *監製者類型。*應用程式或服務、Proxy（REST、MQTT、其他）、以及現有資料庫（RDBMS、NoSQL、其他）。您也可以選取「我不知道」。
- *平均處理量。*以每秒事件為單位（例如1、000、000）。
- *尖峰處理量。*每秒事件數（例如400萬）。
- 平均訊息大小（以位元組為單位）、未壓縮（最多1MB；例如1000）。
- *訊息格式。*選項包括Avro、Json、傳輸協定緩衝區、二進位、文字、 「我不知道」和其他。
- *複寫係數*選項為1、2、3（一致推薦）、4、5、 或6.
- *保留時間*一天（例如）。您想要將資料儲存在Apache Kafka多長時間？在無限時間內輸入-1及任何單位。此計算機假設保留時間為10年、以供無限期保留。
- 選取「啟用分層儲存以減少代理計數並允許無限儲存？」核取方塊。
- 啟用階層式儲存設備時、保留欄位會控制儲存在本機代理程式上的常用資料集。歸檔保留欄位可控制資料儲存在歸檔物件儲存區的時間長度。
- *歸檔儲存保留*一年（例如）。您希望將資料儲存在歸檔儲存設備中多久？在無限期間內輸入-1及任何單位。此計算機假設保留10年、以便無限期保留。
- 成長倍率 1（例如）。如果此參數的值是根據目前處理量而定、請將其設為1。若要根據額外成長來調整規模、請將此參數設為成長倍頻。
- 產生者執行個體數。 10（例如）。將執行多少個生產商執行個體？此輸入是將CPU負載併入規模計算所需的。空白值表示CPU負載未納入計算。

根據此範例輸入、規模調整對生產商有下列影響：

- 未壓縮位元組的平均處理量：1Gbps。未壓縮位元組的尖峰處理量：4Gbps。以壓縮位元組為單位的平均處理量：400Mbps。尖峰處理量（壓縮位元組）：1.6GBps。這是根據預設的60%壓縮率（您可以變更此值）。
  - 所需的總代理熱集儲存容量：31、104 TB、包括複寫、壓縮。所需的非代理歸檔儲存設備總數：378、432TB、已壓縮。使用 "https://fusion.netapp.com" 以利規模調整。StorageGRID

串流處理器必須說明其應用程式或服務、這些應用程式或服務會耗用Apache Kafka的資料、並將資料傳回Apache Kafka。在大多數情況下、這些都是以ksqlDB或Kafka串流建置而成。

- 名稱 Spark streamer。
- *處理時間。*此處理器處理單一訊息需要多久時間？
  - 1毫秒（簡單、無狀態轉換）[範例]、10毫秒（記憶體內有狀態作業）。
  - 100ms（狀態網路或磁碟作業）、1000ms（第三方REST通話）。

    - 我已對此參數進行基準測試、並確切瞭解所需時間。
- 輸出保留。 1天（範例）。串流處理器會將輸出傳回Apache Kafka。您希望此輸出資料在Apache Kafka中儲存多久？在無限期間內輸入-1及任何單位。
- 勾選「啟用分層儲存以減少Broker Count並允許無限儲存？」核取方塊。
- 歸檔儲存保留 1年（例如）。您希望將資料儲存在歸檔儲存設備中多久？在無限期間內輸入-1及任何單位。此計算機假設保留10年、以便無限期保留。
- 輸出傳遞百分比。 100（例如）。串流處理器會將輸出傳回Apache Kafka。傳入處理量的多少百分比會輸出回Apache Kafka？例如、如果傳入處理量為20Mbps、而此值為10、則輸出處理量將為2Mbps。
- 從哪些應用程式讀取？選取「Spark」，這是在生產廠商類型規模調整中使用的名稱。根據上述輸入、您可以預期調整大小對串流處理程序執行個體和主題分割區預估的影響如下：
- 此串流處理器應用程式需要下列數目的執行個體。傳入的主題也可能需要這麼多分割區。請聯絡Conflent以確認此參數。
    - 平均處理量為1、000、沒有成長倍頻
    - 4、000個尖峰處理量、無成長倍頻
    - 平均處理量1、000個、使用成長倍頻
    - 4、000個尖峰處理量、使用成長倍頻

消費者

請描述您的應用程式或服務、這些應用程式或服務會耗用Apache Kafka的資料、而不會產生回Apache Kafka；例如、原生用戶端或Kafka Connector。

- 名稱 Spark消費者。
- *處理時間。*此消費者處理單一訊息所需的時間？
    - 1毫秒（例如、記錄等簡單且無狀態的工作）
    - 10ms（快速寫入資料存放區）
    - 100毫秒（緩慢寫入資料存放區）
    - 1000毫秒（第三方REST通話）
    - 已知持續時間的其他基準程序。
- *使用者類型。*應用程式、Proxy或接收到現有的資料存放區（RDBMS、NoSQL、其他）。
- 從哪些應用程式讀取？將此參數與先前決定的監製者和串流規模連結起來。

根據上述輸入、您必須決定使用者執行個體和主題分割預估的規模。使用者應用程式需要下列執行個體數目。

- 平均處理量為2、000、沒有成長倍頻
- 8000個尖峰處理量、無成長倍頻
- 平均處理量為2、000、包括成長倍率
- 8000個尖峰處理量、包括成長倍頻

傳入的主題也可能需要這個數目的分割區。請聯絡Conflent以確認。

除了對生產商、串流處理器和消費者的要求之外、您還必須提供下列額外要求：

- *重建時間*例如4小時。如果Apache Kafka Broker主機故障、資料遺失、而且配置了新主機來更換故障主機、那麼這台新主機重建的速度必須有多快？如果值不明、請將此參數留白。
- *資源使用率目標（百分比）。*例如60。您希望主機在平均處理量期間的使用率為何？除非您使用Conflent自我平衡叢集、否則Conflent建議使用率為60%、在此情況下、使用率可能會較高。

描述您的環境

- 您的叢集將在哪些環境中執行？ Amazon Web Services、Microsoft Azure、Google雲端平台、內部部署裸機、內部部署VMware、 內部部署OpenStack或內部部署Kubernates？
- *主機詳細資料。*核心數：48（例如）、網路卡類型（10GbE、40GbE、16GbE、1GbE或其他類型）。
- *儲存磁碟區。*主機：12（例如）。每個主機支援多少個硬碟機或SSD？Conflent建議每個主機使用12個硬碟機。
- 儲存容量/磁碟區（單位：**GB**）。 1000（例如）。單一磁碟區可儲存多少GB儲存空間？Connent建議使用1TB磁碟。
- *儲存組態。*如何設定儲存磁碟區？Conflent建議使用RAID10、以充分發揮所有Conflent功能的優勢。JBOD、SAN、RAID 1、RAID 0、RAID 5、 也支援其他類型。
- 單一**Volume**處理量（**Mbps**）。 125（例如）。單一儲存磁碟區每秒讀取或寫入MB的速度有多快？Confluent建議使用標準硬碟、通常處理量為125MBps。
- 記憶體容量（**GB**）。 64（例如）。

確定環境變數之後、請選取「調整叢集大小」。根據上述範例參數、我們針對Connent Kafka決定了下列規模：

- * Apache Kafka.* Broker數量：22。您的叢集需要儲存設備。考慮啟用階層式儲存設備、以減少主機數量、並允許無限儲存。
- * Apache Zookeeper。*計數：5；Apache Kafka Connect工作人員：數：2；架構登錄：數：2；REST Proxy：數：2；ksqlDB：數：2；ConFluent Control Center：數：1。

不需考量使用案例、即可針對平台團隊使用反轉模式。使用分割模式來計算單一主題所需的分割區數量。請參閱 https://eventsizer.io 以反轉和分割模式為基礎進行規模調整。

結論

本文件提供將ConFluent階層式儲存設備與NetApp儲存設備搭配使用的最佳實務準則、包括驗證測試、階層式儲存效能結果、調校、ConFluent S3連接器、以及自我平衡功能。考慮到ILM原則、透過多項驗證效能測試的一致效能、以及業界標準的S3 API、NetApp StorageGRID 物件儲存設備是Confluent階層式儲存設備的最佳選擇。

何處可找到其他資訊

若要深入瞭解本文所述資訊、請檢閱下列文件和 / 或網站：

- 什麼是Apache Kafka

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

- NetApp 產品文件

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

- S3-sink參數詳細資料

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

- Apache Kafka

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

- 在Confluent平台上實現無限儲存

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

- 一致的分層儲存設備：最佳實務做法與規模調整

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

- 適用於ConFluent Platform的Amazon S3接收器連接器

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

- Kafka尺寸

  "https://eventsizer.io"

- 規模調整StorageGRID

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

- Kafka使用案例

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

- 在conf最 合體平台6.0中實現卡夫卡叢集的自我平衡

  "https://www.confluent.io/blog/self-balancing-kafka-clusters-in-confluent-platform-6-0/"

  "https://www.confluent.io/blog/confluent-platform-6-0-delivers-the-most-powerful-event-streaming-platform-to-date/"

NetApp混合雲資料解決方案：根據客戶使用案例、Spark和Hadoop

TR-4657：NetApp混合雲資料解決方案：根據客戶使用案例而設計的Spark和Hadoop

NetApp的Karthithkeyan Nagalingam和Sathish Thyagarajan

本文件說明混合雲資料解決方案、使用NetApp AFF 的解決方案、NetApp FAS 的解決方案、NetApp的NetApp連接儲存設備、以及適用於Spark和Hadoop的NetApp FlexClone技術。Cloud Volumes ONTAP這些解決方案架構可讓客戶針對其環境選擇適當的資料保護解決方案。NetApp根據與客戶及其商業使用案例的互動來設計這些解決方案。本文件提供下

列詳細資訊：

- 為何我們需要針對Spark和Hadoop環境及客戶挑戰提供資料保護。
- 採用NetApp願景及其建置區塊與服務的資料架構。
- 如何使用這些建置區塊來架構靈活的資料保護工作流程。
- 多種架構的優缺點是以實際客戶使用案例為基礎。每個使用案例均提供下列元件：
  - 客戶案例
  - 需求與挑戰
  - 解決方案
  - 解決方案摘要

為何需要**Hadoop**資料保護？

在Hadoop和Spark環境中、必須解決下列問題：

- *軟體或人為故障。*執行Hadoop資料作業時、軟體更新中發生人為錯誤、可能導致錯誤行為、導致工作產生非預期的結果。在這種情況下、我們需要保護資料、以避免失敗或不合理的結果。例如、由於流量訊號分析應用程式的軟體更新執行不良、因此這項新功能無法以純文字格式正確分析流量訊號資料。此軟體仍會分析Json及其他非文字檔案格式、進而產生即時流量控制分析系統、產生遺漏資料點的預測結果。這種情況可能會造成輸出故障、導致交通號誌發生意外。資料保護可提供快速回復至先前工作應用程式版本的功能、藉此解決此問題。
- *規模與規模。*由於資料來源與磁碟區數量不斷增加、分析資料的大小日增。社群媒體、行動應用程式、資料分析及雲端運算平台是目前巨量資料市場中資料的主要來源、而且資料的成長速度極快、因此資料必須受到保護、才能確保資料作業準確無誤。
- * Hadoop的原生資料保護功能。* Hadoop具有原生命令來保護資料、但此命令無法在備份期間提供一致的資料。它僅支援目錄層級備份。Hadoop建立的快照為唯讀、無法直接重複使用備份資料。

Hadoop和**Spark**客戶面臨的資料保護挑戰

Hadoop和Spark客戶的一項常見挑戰是縮短備份時間並提高備份可靠性、而不會在資料保護期間對正式作業叢集的效能造成負面影響。

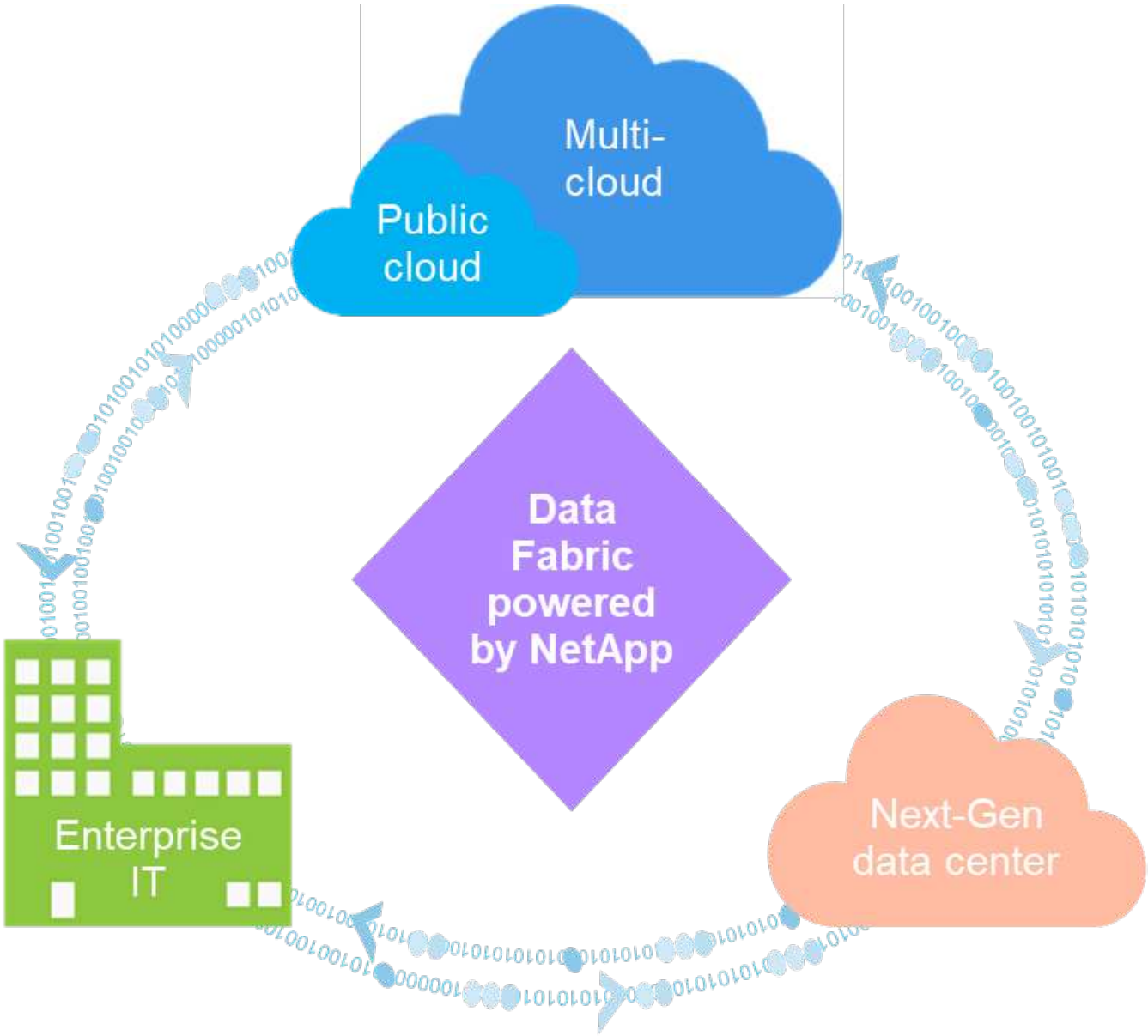
客戶也需要將恢復點目標（RPO）和恢復時間目標（RTO）停機時間降至最低、並控制內部部署和雲端型災難恢復站台、以達到最佳的營運持續性。這項控管措施通常來自於企業層級的管理工具。

Hadoop和Spark環境非常複雜、因為不僅資料量龐大且不斷成長、而且資料的送達速度也不斷提高。此案例讓您難以從來源資料快速建立有效率且最新的DevTest與QA環境。NetApp瞭解這些挑戰、並提供本白皮書所述的解決方案。

採用**NetApp**技術的**Data Fabric**、適用於**Big Data**架構

採用NetApp技術的資料架構可簡化並整合雲端與內部部署環境的資料管理、加速數位轉型。

採用NetApp技術的資料架構提供一致且整合的資料管理服務與應用程式（建置區塊）、可提供資料可見度與洞見、資料存取與控制、以及資料保護與安全性、如下圖所示。



備受肯定的**Data Fabric**客戶使用案例

採用NetApp技術的資料架構可為客戶提供下列九個獲證實的使用案例：

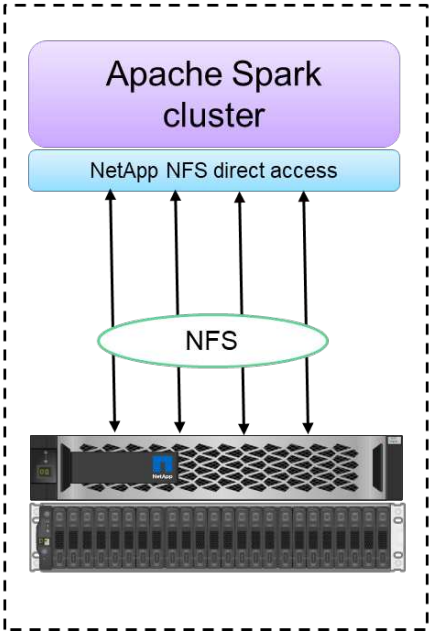
- 加速分析工作負載
- 加速DevOps轉型
- 建置雲端託管基礎架構
- 整合雲端資料服務
- 保護及保護資料安全
- 最佳化非結構化資料
- 提高資料中心效率
- 提供資料洞見與控管能力
- 簡化及自動化

本文件涵蓋九種使用案例中的兩種（連同其解決方案）：

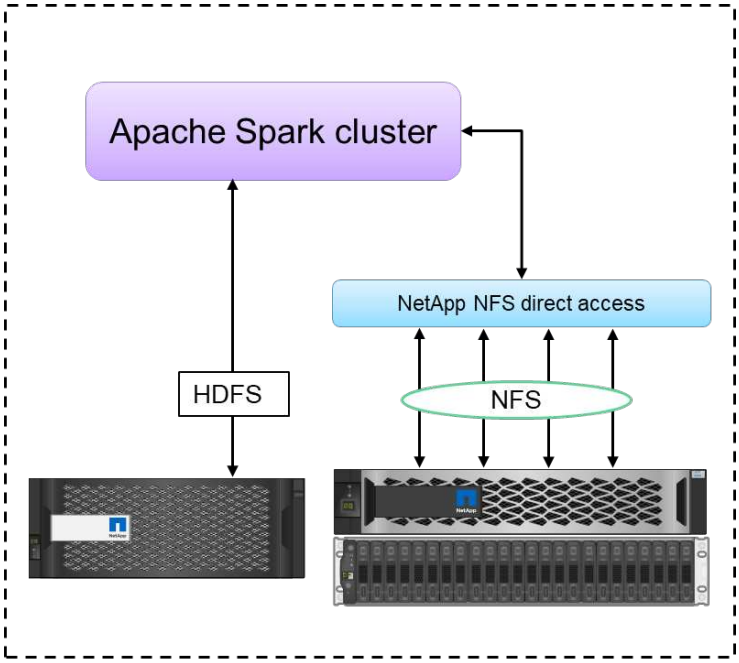
- 加速分析工作負載
- 保護及保護資料安全

NetApp NFS直接存取

NetApp NFS 可讓客戶在現有或新的 NFSv3 或 NFSv4 資料上執行巨量資料分析工作、而無需移動或複製資料。它可防止多個資料複本、並免除將資料與來源同步的需求。例如、在金融產業中、資料從一個地方搬移到另一個地方、必須符合法律義務、這不是一項容易的任務。在此案例中、NetApp NFS直接存取會從原始位置分析財務資料。另一項主要優點是使用NetApp NFS直接存取功能、使用原生Hadoop命令來簡化Hadoop資料的保護、並利用NetApp豐富的資料管理產品組合來啟用資料保護工作流程。



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

NetApp NFS直接存取可為Hadoop / Spark叢集提供兩種部署選項：

- 根據預設、Hadoop / Spark叢集會使用Hadoop分散式檔案系統（HDFS）進行資料儲存、並使用預設的檔案系統。NetApp NFS直接存取可將預設的HDFS取代為預設的NFS儲存系統、以便直接分析NFS資料。
- 在另一個部署選項中、NetApp NFS直接存取可在單一Hadoop / Spark叢集中、將NFS設定為額外的儲存設備、以及HDFS。在這種情況下、客戶可以透過NFS匯出來共享資料、並從同一個叢集與HDFS資料一起存取。

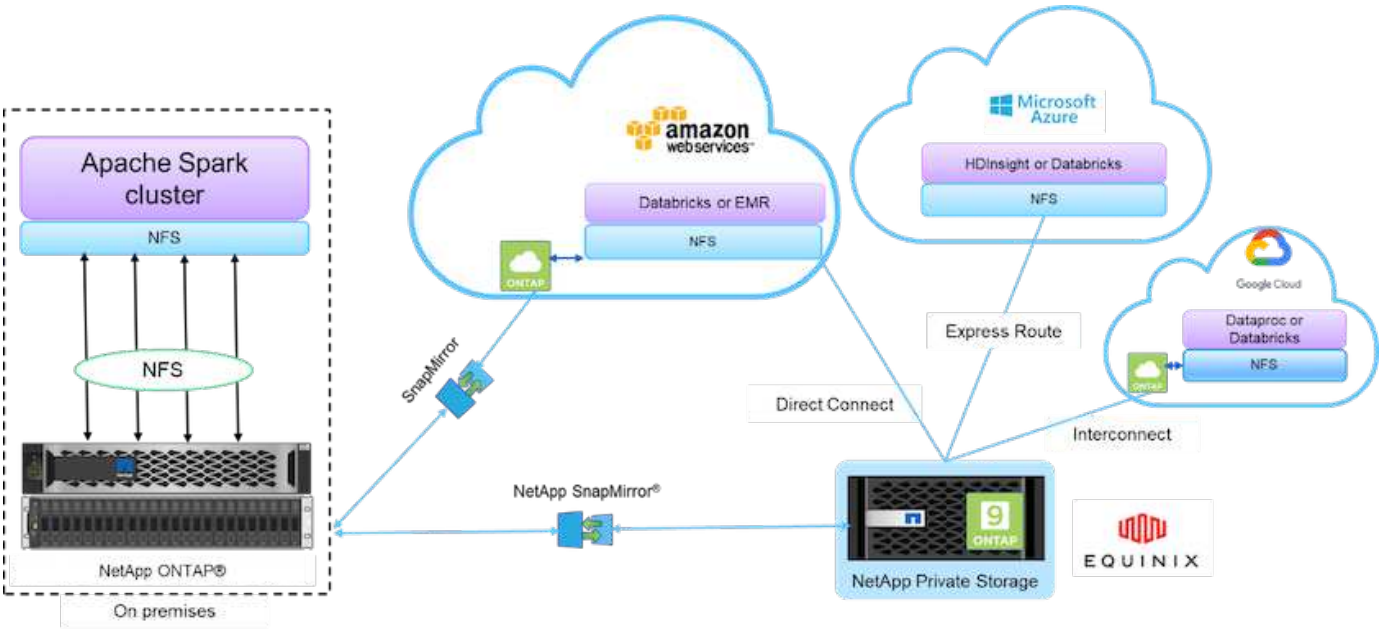
使用NetApp NFS直接存取的主要效益包括：

- 從目前位置分析資料、避免將分析資料移至Hadoop基礎架構（例如HDFS）所需的時間與效能。
- 將複本數量從三個減少為一個。
- 讓使用者能夠分離運算和儲存設備、以獨立擴充。
- 運用ONTAP 豐富的資料管理功能、提供企業資料保護功能。
- 通過Hortonworks資料平台認證。

- 實現混合式資料分析部署。
- 運用動態多執行緒功能、縮短備份時間。

巨量資料的建置區塊

採用NetApp技術的資料架構整合了資料管理服務與應用程式（建置區塊）、可用於資料存取、控制、保護及安全性、如下圖所示。



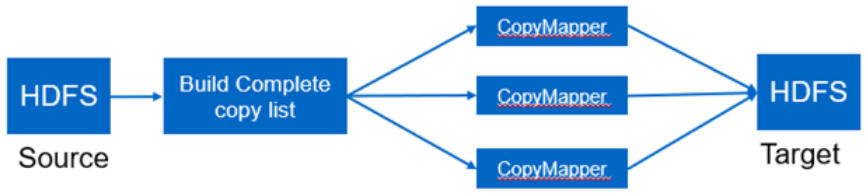
上圖中的建置區塊包括：

- * NetApp NFS直接存取。*提供最新的Hadoop和Spark叢集、可直接存取NetApp NFS磁碟區、無需額外的軟體或驅動程式需求。
- * NetApp Cloud Volumes ONTAP 功能與雲端Volume Services。*軟體定義的連線儲存設備、以ONTAP Microsoft Azure NetApp Files Azure雲端服務中Amazon Web Services（AWS）或Sf2（anf）執行的功能為基礎。
- * NetApp SnapMirror技術*。在內部部署ONTAP 與不支援的Cloud或NPS執行個體之間提供資料保護功能。
- *雲端服務供應商。*這些供應商包括AWS、Microsoft Azure、Google Cloud和IBM Cloud。
- * PaaS *雲端型分析服務、例如AWS中的Amazon Elastic MapReduce（EMR）和Databricks、以及Microsoft Azure HDInsight和Azure Databricks。

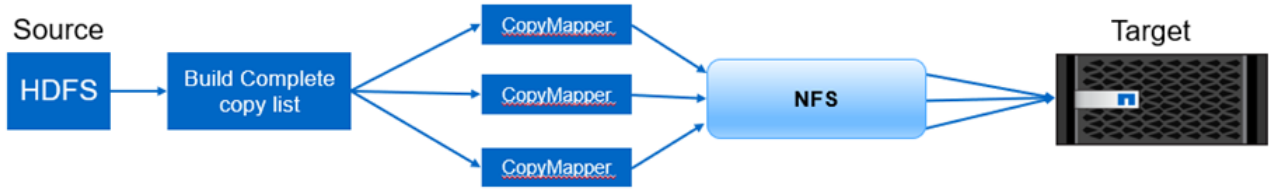
Hadoop資料保護與NetApp

Hadoop DistCp是一種原生工具、用於大型叢集間和叢集內複製。下圖所示的Hadoop DistCp基本程序、是使用Hadoop原生工具（例如MapReduce）將Hadoop資料從HDFS來源複製到對應目標的典型備份工作流程。

NetApp NFS直接存取可讓客戶將NFS設定為Hadoop DistCp工具的目標目的地、以便透過MapReduce將資料從HDFS來源複製到NFS共用。NetApp NFS直接存取可做為DistCp工具的NFS驅動程式。



Hadoop Distcp Basic Process



Hadoop Distcp and NetApp

Hadoop資料保護使用案例總覽

本節提供資料保護使用案例的詳細說明、這是本白皮書的重點。其餘各節將針對每個使用案例提供更多詳細資料、例如客戶問題（情境）、需求與挑戰、以及解決方案。

使用案例**1**：備份**Hadoop**資料

在這種使用案例中、 NetApp NFS Volume 協助大型金融機構將長備份時間從 24 小時以上縮短為幾小時以下。

使用案例**2**：從雲端到內部部署的備份與災難恢復

藉由使用NetApp技術的資料架構做為建置區塊、一家大型廣播公司能夠根據不同的資料傳輸模式（例如隨需、即時、 或根據Hadoop / Spark叢集負載。

使用案例**3**：在現有**Hadoop**資料上啟用**DevTest**

NetApp解決方案協助線上音樂經銷商在不同的分公司快速建置多個節省空間的Hadoop叢集、以使用排程原則來建立報告並執行每日DevTest工作。

使用案例**4**：資料保護與多雲端連線

一家大型服務供應商使用NetApp技術提供的資料架構、為不同雲端執行個體的客戶提供多重雲端分析功能。

使用案例**5**：加速分析工作負載

最大的金融服務與投資銀行之一使用NetApp網路附加儲存解決方案來縮短I/O等待時間、並加速其量化財務分析平台。

使用案例**1**：備份**Hadoop**資料

案例

在此案例中、客戶擁有大型內部部署Hadoop儲存庫、並想要備份以供災難恢復之用。然而、客戶目前的備份解決方案成本高昂、而且備份時間長達24小時以上。

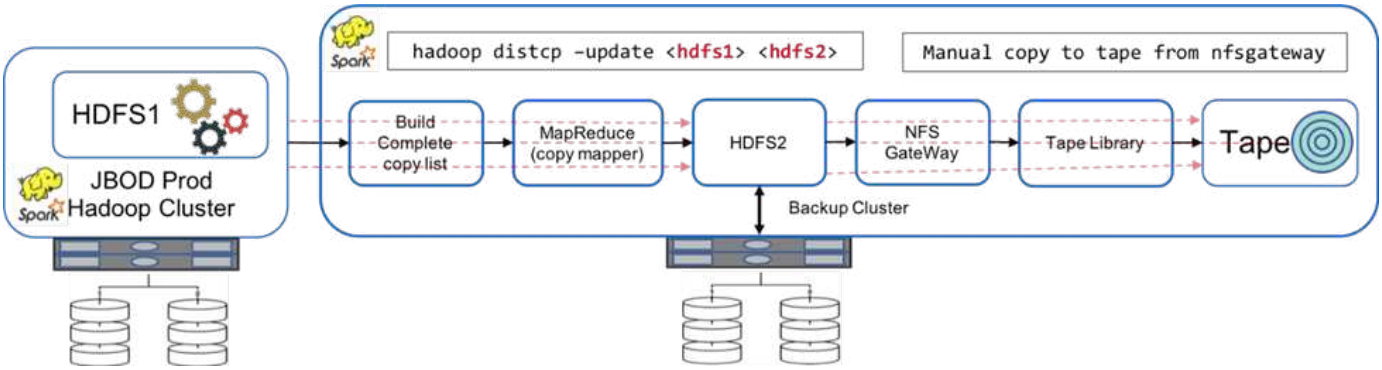
需求與挑戰

此使用案例的主要要求與挑戰包括：

- 軟體向下相容性：
  - 建議的替代備份解決方案應與目前在正式作業Hadoop叢集中使用的軟體版本相容。
- 為了符合承諾的SLA、建議的替代解決方案應達到極低的RPO和RTO。
- NetApp備份解決方案所建立的備份可用於本機建置於資料中心的Hadoop叢集、以及在遠端站台的災難恢復位置執行的Hadoop叢集。
- 建議的解決方案必須符合成本效益。
- 建議的解決方案必須在備份期間降低目前執行中的線上分析工作的效能影響。

客戶現有的備份解決方案 **x**

下圖顯示原始Hadoop原生備份解決方案。



正式作業資料會透過中繼備份叢集保護至磁帶：

- 執行「Hadoop distcp -update <hdfs1><hdfs2>'命令、將HDFS1資料複製到HDFS2。
- 備份叢集做為NFS閘道、資料會透過磁帶庫透過Linux「CP」命令手動複製到磁帶。

原始Hadoop原生備份解決方案的優點包括：

- 此解決方案以Hadoop原生命令為基礎、讓使用者不必學習新程序。
- 此解決方案採用業界標準的架構和硬體。

原始Hadoop原生備份解決方案的缺點包括：

- 備份時間過長超過24小時、使正式作業資料容易受到攻擊。
- 在備份期間、叢集效能大幅降低。
- 複製到磁帶是手動程序。

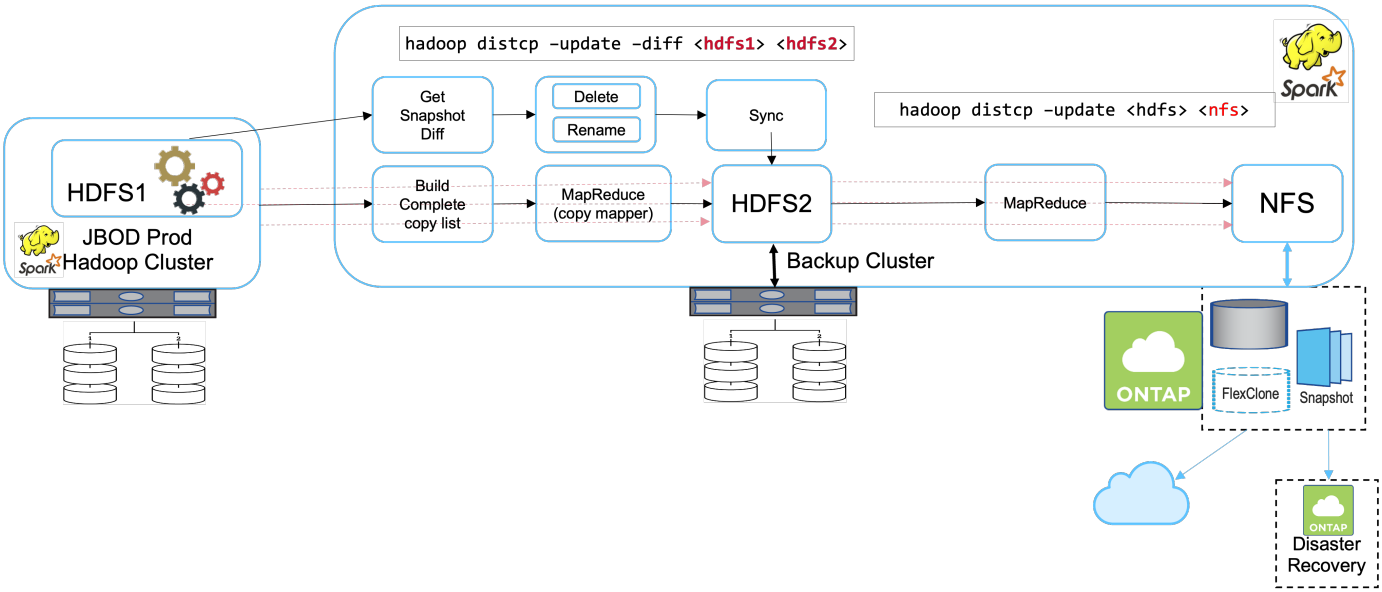
- 就所需的硬體和人工程序所需的人力時間而言、備份解決方案的成本相當昂貴。

備份解決方案

根據這些挑戰和要求、並考慮到現有的備份系統、建議三種可能的備份解決方案。以下各小節分別說明這三種不同的備份解決方案、分別標示為解決方案A至解決方案C

解決方案**A**

在解決方案 A 中、備份 Hadoop 叢集會將次要備份傳送至 NetApp NFS 儲存系統、免除磁帶需求、如下圖所示。



解決方案A的詳細工作包括：

- 正式作業Hadoop叢集的HDFS中有客戶的分析資料需要保護。
- 使用HDFS的備份Hadoop叢集可做為資料的中繼位置。在正式作業和備份Hadoop叢集中、只有一堆磁碟（JBOD）可為HDFS提供儲存設備。
- 執行「Hadoop distcp–update–diff <hdfs1><hdfs2>'命令、保護Hadoop正式作業資料、使其免受正式作業叢集HDFS對備份叢集HDFS的影響。

> Hadoop快照可用來保護資料、使其不受正式作業環境的影響、也不受Hadoop叢集的影響。

- NetApp ONTAP 功能區儲存控制器提供NFS匯出的Volume、可配置給備份Hadoop叢集。
- 執行 `Hadoop distcp` 命令運用 MapReduce 和多個地圖程式、可保護分析資料、使其不受從備份 Hadoop 叢集到 NFS 的影響。

  在NetApp儲存系統的NFS中儲存資料之後、NetApp Snapshot、SnapRestore 支援功能和FlexClone技術可視需要備份、還原及複製Hadoop資料。

> Hadoop資料可透過SnapMirror技術、保護至雲端及災難恢復位置。

解決方案A的優點包括：

- Hadoop正式作業資料受到保護、不受備份叢集的影響。
- HDFS資料是透過NFS保護、可保護雲端和災難恢復位置。
- 將備份作業卸載到備份叢集、藉此提升效能。
- 免除手動磁帶操作
- 可透過NetApp工具執行企業管理功能。
- 只需對現有環境進行最少的變更。
- 是具成本效益的解決方案。

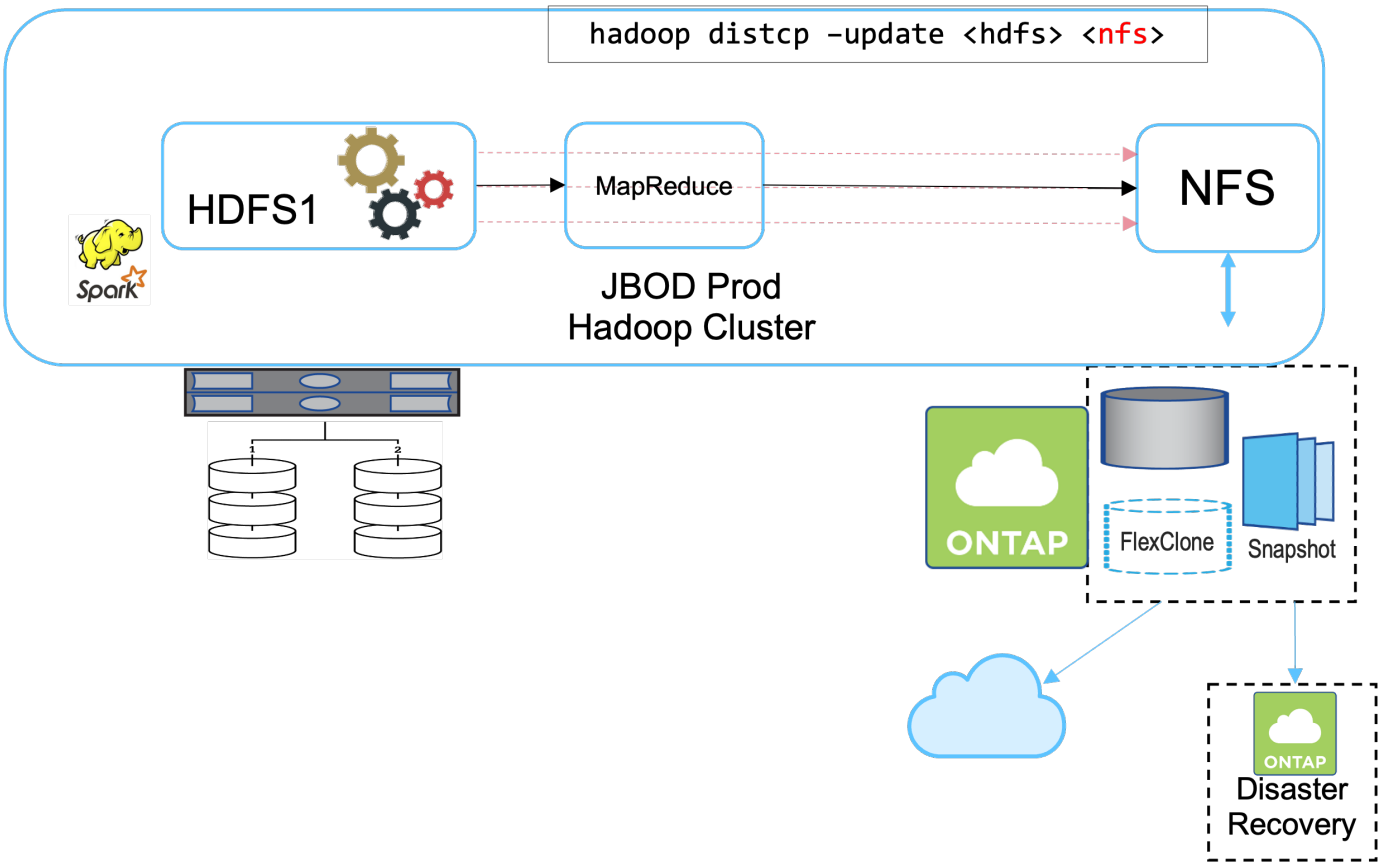
此解決方案的缺點是需要備份叢集和其他地圖來改善效能。

客戶最近部署了解決方案A、因為它的簡易性、成本和整體效能。

在本解決方案中、ONTAP 可以使用來自於支援的SAN磁碟來取代JBOD。此選項可將備份叢集儲存負載卸載ONTAP 至Ef2、但缺點是需要SAN光纖交換器。

解決方案B

解決方案 B 會將 NFS Volume 新增至正式作業的 Hadoop 叢集、這樣就不需要備份 Hadoop 叢集、如下圖所示。



解決方案B的詳細工作包括：

- NetApp ONTAP 功能區儲存控制器會將NFS匯出至正式作業Hadoop叢集。

Hadoop 原生 `hadoop distcp` 命令可保護從正式作業叢集 HDFS 到 NFS 的 Hadoop 資料。

- 在NetApp儲存系統的NFS中儲存資料之後、將SnapRestore 使用Snapshot、支援功能和FlexClone技術來備份、還原及複製Hadoop資料。

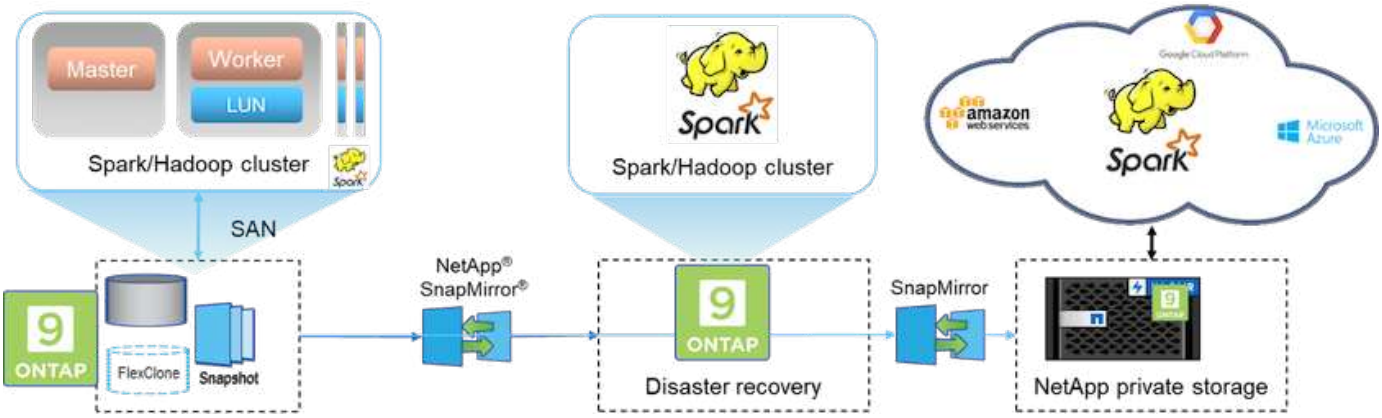
解決方案B的優點包括：

- 正式作業叢集已針對備份解決方案進行稍微修改、以簡化實作並降低額外的基礎架構成本。
- 不需要備份作業的備份叢集。
- HDFS正式作業資料在轉換為NFS資料時受到保護。
- 此解決方案可透過NetApp工具提供企業管理功能。

此解決方案的缺點在於它是在正式作業叢集中實作的、這可能會在正式作業叢集中新增額外的系統管理員工作。

解決方案C

在解決方案C中、NetApp SAN磁碟區會直接配置給HDFS儲存設備的Hadoop正式作業叢集、如下圖所示。



解決方案C的詳細步驟包括：

- NetApp ONTAP 支援SAN儲存設備是在正式作業的Hadoop叢集上配置、以供HDFS資料儲存使用。
- NetApp Snapshot與SnapMirror技術可用來備份來自正式作業Hadoop叢集的HDFS資料。
- 在Snapshot複本備份程序期間、Hadoop / Spark叢集的正式作業效能不會受到影響、因為備份是在儲存層。

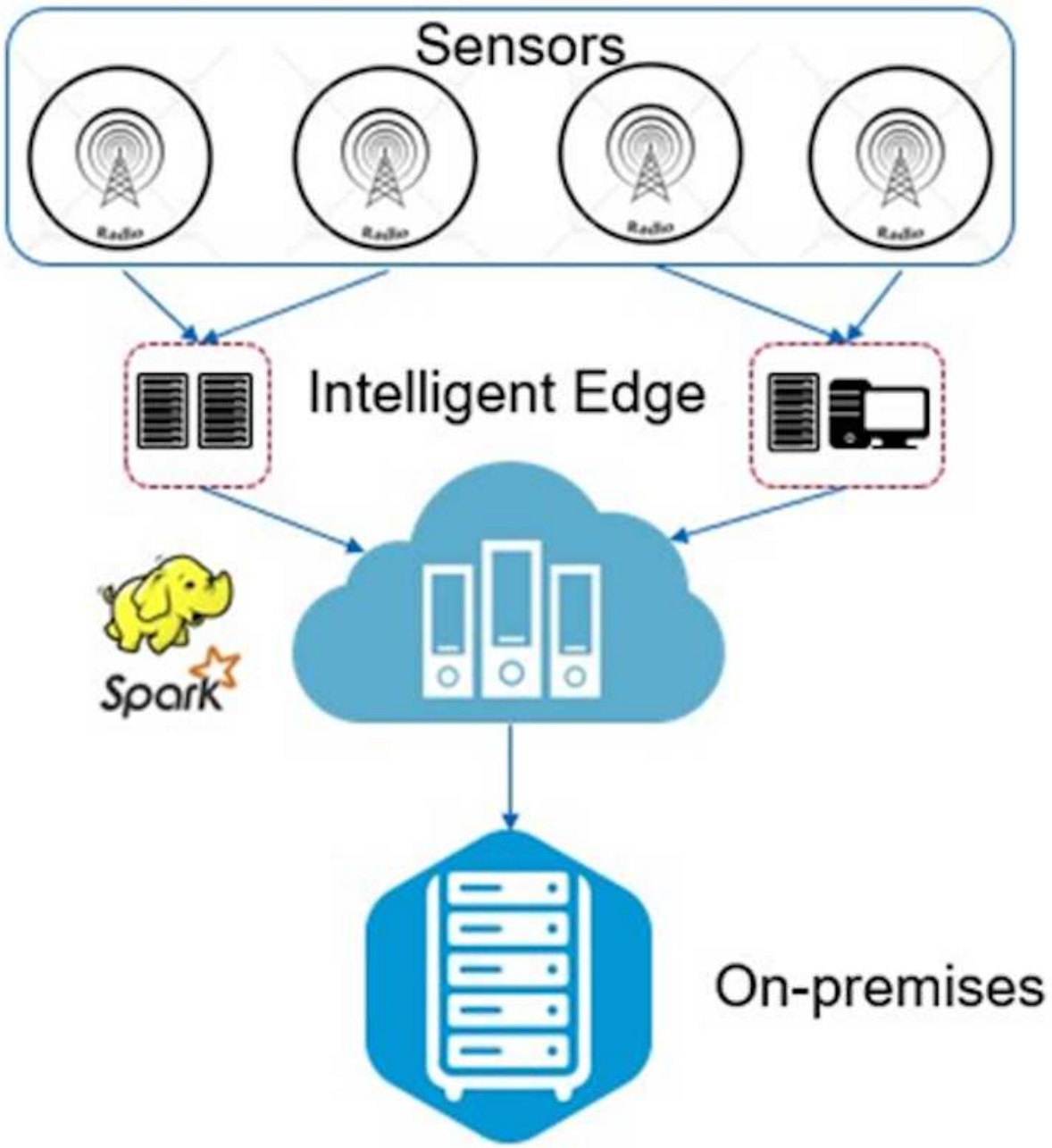
> Snapshot技術提供的備份、無論資料大小為何、都能在數秒內完成。

解決方案C的優點包括：

- 使用Snapshot技術可以建立節省空間的備份。
- 可透過NetApp工具執行企業管理功能。

使用案例2：從雲端到內部部署的備份與災難恢復

此使用案例是根據需要將雲端型分析資料備份到內部部署資料中心的廣播客戶、如下圖所示。



案例

在此案例中、IoT感應器資料會擷取至雲端、並使用AWS內的開放原始碼Apache Spark叢集進行分析。這項需求是將處理過的資料從雲端備份到內部部署。

需求與挑戰

此使用案例的主要要求與挑戰包括：

- 啟用資料保護不應對雲端上的正式作業Spark/Hadoop叢集造成任何效能影響。
- 雲端感測器資料必須以有效率且安全的方式移轉及保護至內部部署。
- 可在不同的情況下、例如隨需、即時和低叢集負載期間、靈活地將資料從雲端傳輸至內部部署。

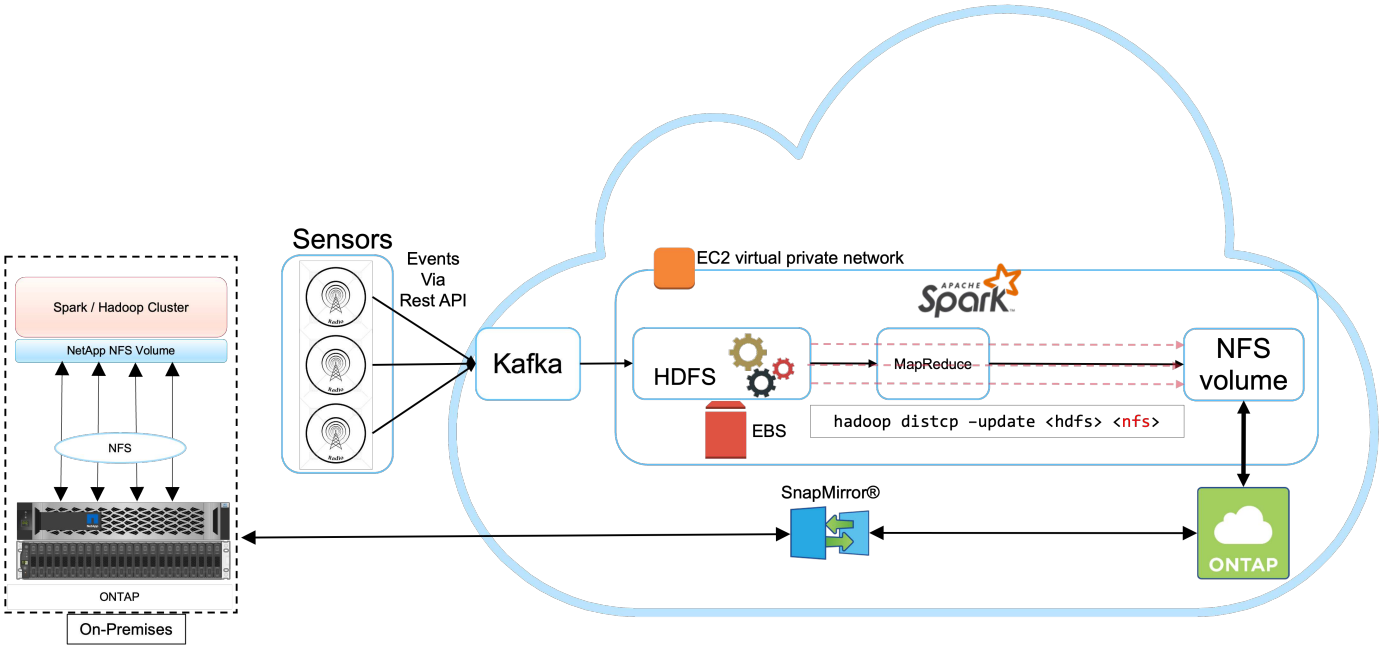
解決方案

客戶使用AWS Elastic Block Store（EBS）作為Spark叢集HDFS儲存設備、透過Kafka從遠端感應器接收及擷取資料。因此、HDFS儲存設備可做為備份資料的來源。

為了滿足這些需求、NetApp ONTAP 支援將NetApp支援雲端部署在AWS中、並建立NFS共用區、做為Spark/Hadoop叢集的備份目標。

建立 NFS 共用之後、將資料從 HDFS EBS 儲存設備複製到 ONTAP NFS 共用區。當資料位於ONTAP NFS中的SnapMirror Cloud之後、SnapMirror技術可用來將資料從雲端鏡射到內部部署儲存設備、並以安全且有效率的方式進行鏡射。

此影像顯示從雲端到內部部署解決方案的備份與災難恢復。



使用案例3：在現有Hadoop資料上啟用DevTest

在此使用案例中、客戶的需求是根據現有的Hadoop叢集、快速且有效率地建置新的Hadoop / Spark叢集、其中包含大量用於DevTest的分析資料、並在同一個資料中心和遠端位置進行報告。

案例

在此案例中、多個Spark / Hadoop叢集是以內部部署的大型Hadoop資料湖實作以及災難恢復位置所建置而成。

需求與挑戰

此使用案例的主要要求與挑戰包括：

- 針對DevTest、QA或任何其他需要存取相同正式作業資料的目的、建立多個Hadoop叢集。這方面的挑戰是、以極具空間效益的方式、即時複製大型Hadoop叢集多次。
- 將Hadoop資料同步至DevTest和報告團隊、以提高營運效率。

- 在正式作業和新叢集之間使用相同的認證資料來散佈Hadoop資料。
- 使用排程的原則來有效率地建立QA叢集、而不會影響正式作業叢集。

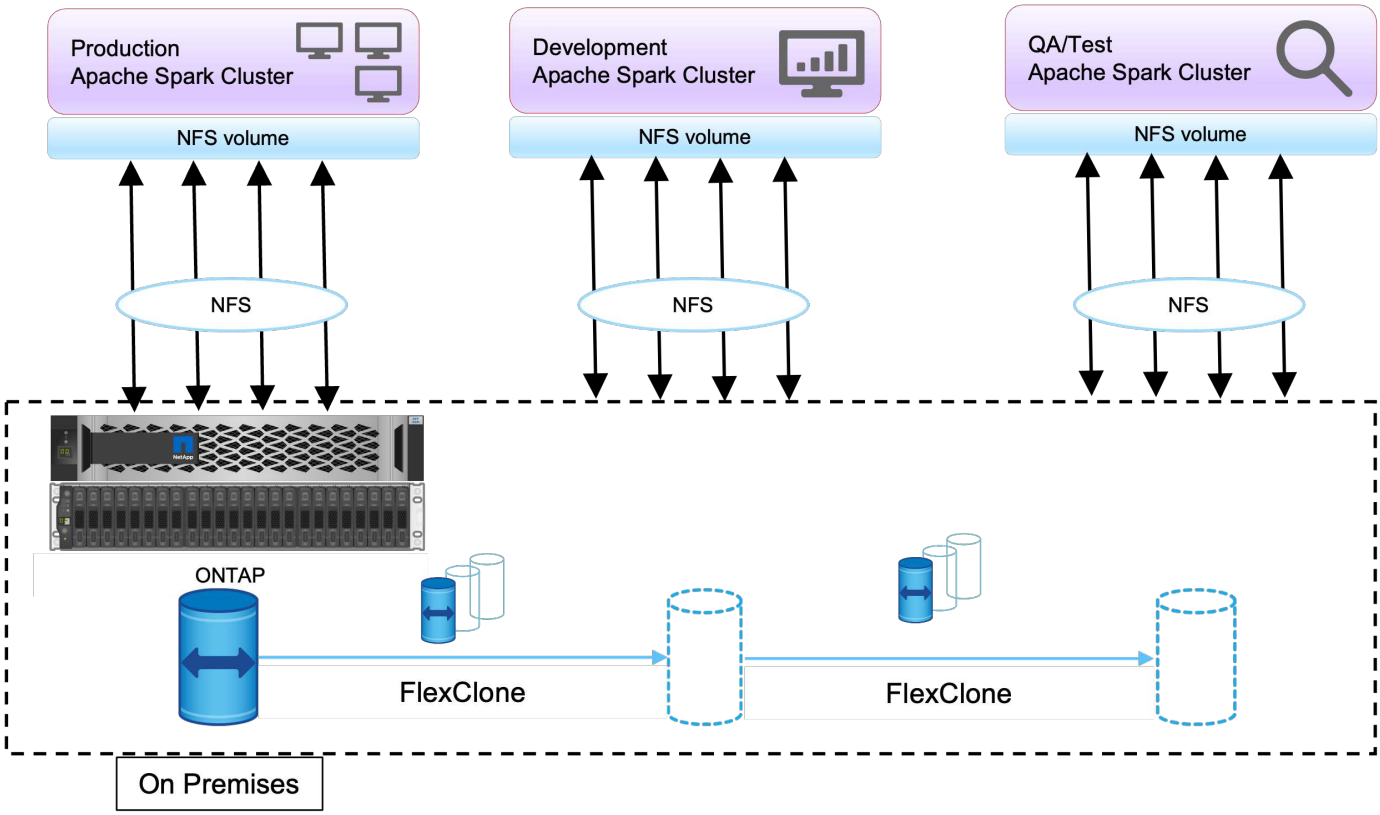
解決方案

FlexClone技術可用來滿足上述需求。FlexClone技術是Snapshot複本的讀寫複本。它會從父Snapshot複本資料讀取資料、只會消耗新增/修改區塊的額外空間。它既快速又節省空間。

首先、使用NetApp一致性群組建立現有叢集的Snapshot複本。

NetApp System Manager或儲存設備管理提示中的Snapshot複本。一致性群組Snapshot複本是應用程式一致的群組Snapshot複本、而FlexClone Volume則是根據一致性群組Snapshot複本建立。值得一提的是、FlexClone Volume繼承父Volume的NFS匯出原則。建立Snapshot複本之後、必須安裝新的Hadoop叢集以供DevTest和報告之用、如下圖所示。從新 Hadoop 叢集複製的 NFS Volume 可存取 NFS 資料。

此影像顯示DevTest的Hadoop叢集。



使用案例4：資料保護與多雲端連線

此使用案例與負責為客戶的Big Data分析資料提供多雲端連線的雲端服務合作夥伴有關。

案例

在此案例中、從不同來源收到的AWS IoT資料會儲存在NPS的中央位置。NPS儲存設備連接至AWS和Azure中的Spark/Hadoop叢集、可讓在多個雲端上執行的Big Data分析應用程式存取相同的資料。

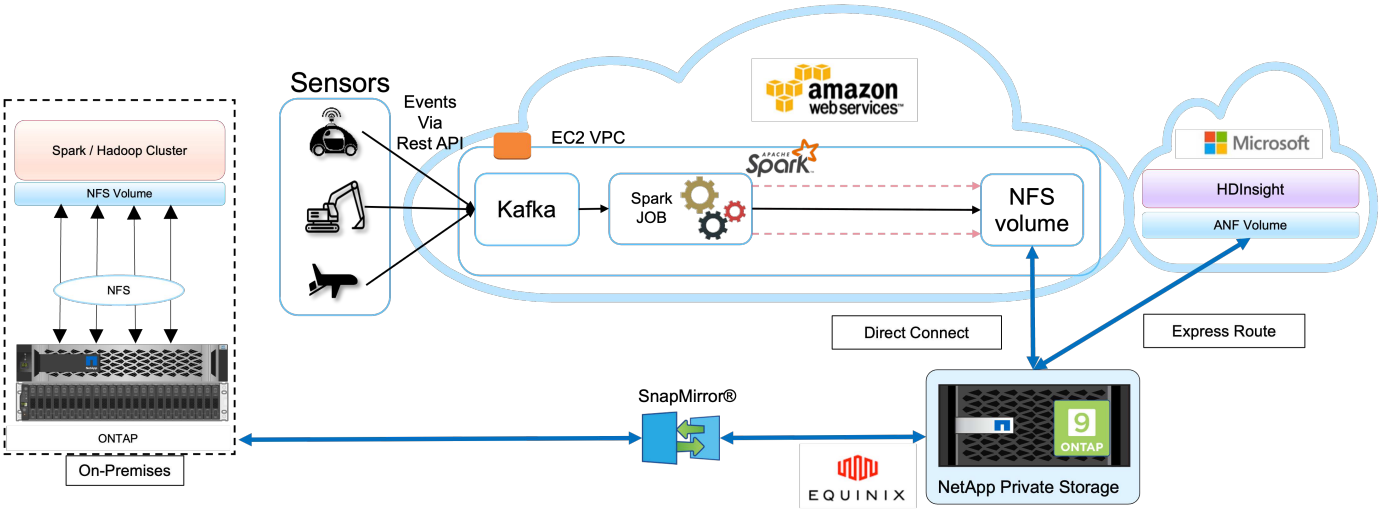
需求與挑戰

此使用案例的主要要求與挑戰包括：

- 客戶想要使用多個雲端在相同的資料上執行分析工作。
- 必須透過不同的感應器和集線器、從內部部署和雲端等不同來源接收資料。
- 解決方案必須有效率且具成本效益。
- 主要挑戰是建置具成本效益且有效率的解決方案、在內部部署和不同雲端之間提供混合式分析服務。

解決方案

此影像說明資料保護與多雲端連線解決方案。



如上圖所示、感應器的資料會透過Kafka串流並擷取至AWS Spark叢集。資料儲存在NPS中的NFS共用區、NPS位於Equinix資料中心內的雲端供應商外部。由於 NetApp NPS 分別透過直接連線和快速路由連線連線至 Amazon AWS 和 Microsoft Azure 、因此客戶可以從 Amazon 和 AWS 分析叢集存取 NFS 資料。這種方法可解決跨多個大型擴充系統進行雲端分析的問題。

因此ONTAP 、由於內部部署和NPS儲存設備都執行了一套功能完善的軟體、SnapMirror可以將NPS資料鏡射到內部部署叢集、在內部部署和多個雲端之間提供混合雲分析功能。

為了獲得最佳效能、NetApp通常建議使用多個網路介面和直接連線/快速路由、從雲端執行個體存取資料。

使用案例5：加速分析工作負載

在此案例中、大型金融服務與投資銀行的分析平台已採用NetApp NFS儲存解決方案進行現代化、以大幅改善分析資產管理與量化業務單位的投資風險與衍生工具。

案例

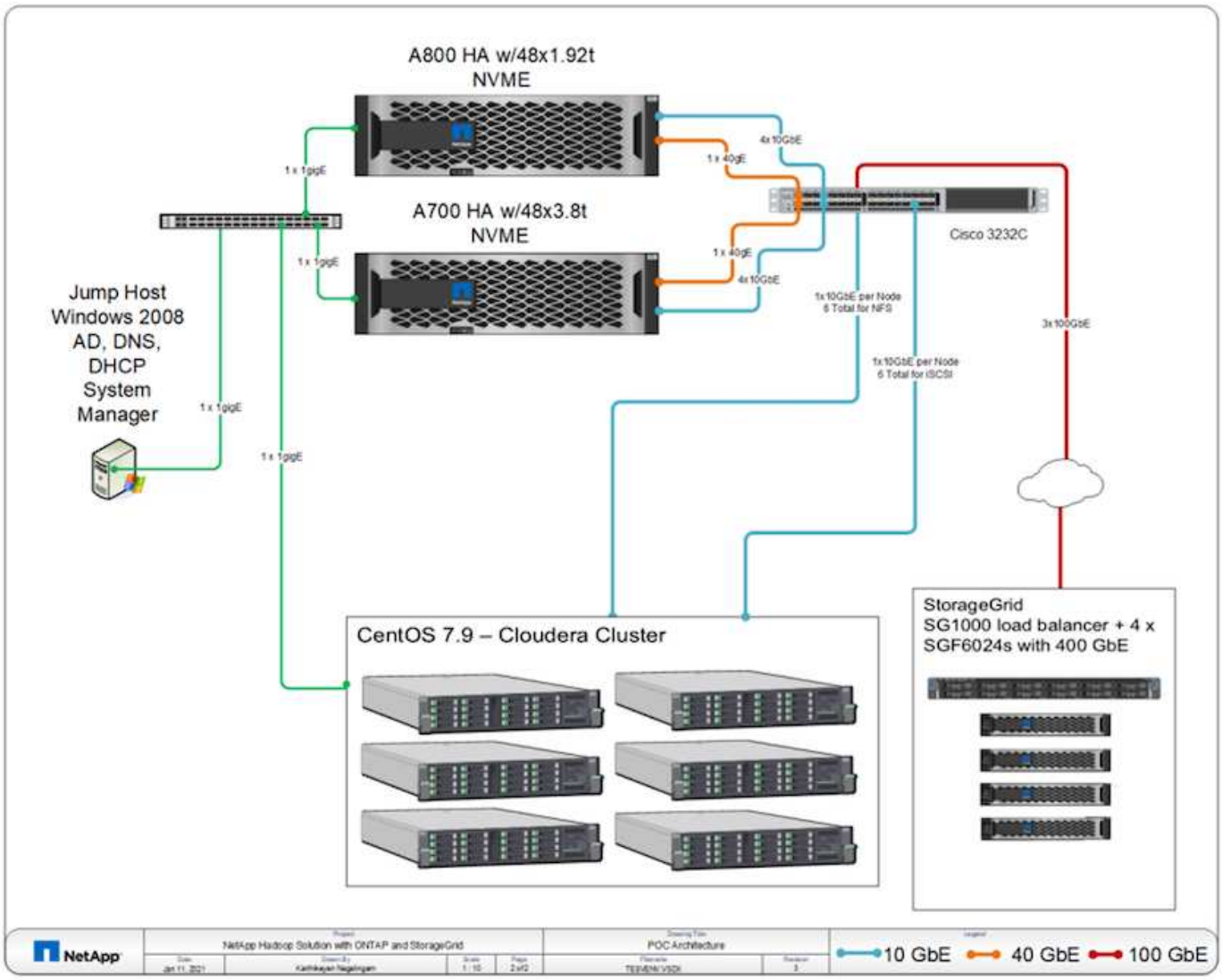
在客戶現有的環境中、用於分析平台的Hadoop基礎架構會利用Hadoop伺服器的內部儲存設備。由於JBOD環境的專屬性質、組織內的許多內部客戶無法利用Monte Carlo量化模型、這是一種仰賴即時資料重複取樣的模擬。對於量化資產管理業務單位而言、無法充分瞭解市場變動不確定性的影響。

需求與挑戰

該銀行的量化業務單位需要高效率的預測方法、才能達到準確且及時的預測。為達成此目標、團隊認為必須將基礎架構現代化、縮短現有I/O等待時間、並改善Hadoop和Spark等分析應用程式的效能、以有效模擬投資模式、衡量潛在獲利並分析風險。

解決方案

客戶現有的Spark解決方案擁有JBOD。NetApp的NetApp功能、NetApp功能區和MinIO閘道至NFS、可縮短銀行的量化財務團隊的I/O等待時間、以便針對評估潛在獲利與風險的投資模式、進行模擬與分析。ONTAP StorageGRID此影像顯示採用NetApp儲存設備的Spark解決方案。



如上圖所示、AFF 我們StorageGRID 部署了S還原A800、A700系統和支援Spark的六節點Hadoop叢集、以及適用於資料分析作業的線紗和Hive中繼資料服務、透過NFS和S3傳輸協定來存取硬地板檔案。

客戶舊環境中的直接附加儲存（DAS）解決方案、對於獨立擴充運算與儲存設備而說、有其缺點。有ONTAP 了適用於Spark的NetApp解決方案、該銀行的財務分析業務單位就能將儲存設備與運算分離、並視需要更有效地提供基礎架構資源。

藉由使用ONTAP 支援NFS的功能、運算伺服器CPU幾乎已完全用於Spark SQL工作、I/O等待時間縮短將近70%、因此能為Spark工作負載提供更好的運算能力和效能提升。之後、CPU使用率的提升也讓客戶能夠運

用GPU（例如GPUDirect）來進一步進行平台現代化。此外StorageGRID 、針對Spark工作負載提供低成本的儲存選項、MinIO閘道則透過S3傳輸協定提供對NFS資料的安全存取。對於雲端資料、NetApp建議Cloud Volumes ONTAP 使用「功能性」、Azure NetApp Files 「功能性」和「NetApp Cloud Volumes Service 功能性」。

結論

本節摘要說明NetApp為滿足各種Hadoop資料保護需求所提供的使用案例與解決方案。透過使用NetApp技術的資料架構、客戶可以：

- 運用NetApp豐富的資料管理功能、並與Hadoop原生工作流程整合、可靈活選擇適當的資料保護解決方案。
- 將Hadoop叢集備份時間縮短將近70%。
- 消除Hadoop叢集備份所造成的任何效能影響。
- 同時提供多雲端資料保護、並可從不同雲端供應商存取資料至單一分析資料來源。
- 使用FlexClone技術建立快速且節省空間的Hadoop叢集複本。

何處可找到其他資訊

若要深入瞭解本文所述資訊、請參閱下列文件和/或網站：

- NetApp Big Data分析解決方案

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

- NetApp儲存設備的Apache Spark工作負載

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

- 適用於Apache Spark的NetApp儲存解決方案

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

- 採用NetApp技術的Data Fabric上的Apache Hadoop

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

感謝

- Paul Burland、NetApp ANZ Victoria地區銷售代表
- NetApp業務開發經理Hosub Dermilian
- NetApp MPSG總監Lee Dorier
- NetApp ANZ Victoria District SE系統工程師David Thiessen

版本歷程記錄

版本	日期	文件版本歷程記錄
1.0版	2018年1月	初始版本

版本	日期	文件版本歷程記錄
2.0版	2021年10月	使用案例5更新：加速分析工作負載
3.0版	2023 年 11 月	移除 NIPAM 詳細資料

現代化資料分析：不同分析策略的不同解決方案

本白皮書說明NetApp現代化資料分析解決方案策略。其中包括業務成果、客戶挑戰、技術趨勢、競爭老舊架構、現代化工作流程、使用案例、產業、雲端、技術合作夥伴、資料移動者、NetApp Active IQ 功能區、NetApp DataOps Toolkit、Hadoop to Spark、採用NetApp Astra Control的軟體定義儲存設備、容器、企業資料管理、歸檔及分層、以達成AI與分析的目標、以及NetApp與客戶如何共同將資料架構現代化。

"現代化資料分析：不同分析策略的不同解決方案"

TR-4623：NetApp E系列E5700和Splunk Enterprise

NetApp的Mitch Blackbur燒

TR-4623說明NetApp E系列和Splunk設計的整合式架構。針對節點儲存平衡、可靠性、效能、儲存容量和密度進行最佳化、此設計採用Splunk叢集式索引節點模式、具有更高的擴充性和更低的TCO。將儲存設備與運算分離、可分別擴充、節省過度資源配置的成本。此外、本文也摘要說明從Splunk機器記錄事件模擬工具取得的效能測試結果。

"TR-4623：NetApp E系列E5700和Splunk Enterprise"

NVA-1157-Deploy：Apache Spark工作負載搭配NetApp儲存解決方案

NetApp的Karthithkeyan Nagalingam

NVA-1157-Deploy說明在NetApp NFS AFF 上驗證Apache Spark SQL的效能與功能。它會根據各種情境來審查組態、架構和效能測試、以及使用Spark搭配NetApp ONTAP 效益資料管理軟體的建議。本報告也涵蓋僅以一堆磁碟（JBOD）為基礎的測試結果、以及NetApp AFF 可靠性A800儲存控制器的測試結果。

"NVA-1157-Deploy：Apache Spark工作負載搭配NetApp儲存解決方案"