Technical Report

NetApp SANtricity Cloud Connector 4.0

Robert Stankey and Brook Wilson, NetApp
January 2020 | TR-4658

In partnership with

aws

Abstract

NetApp® SANtricity® Cloud Connector 4.0 is a host-based Linux application built specifically for engagement with your data fabric powered by NetApp through NetApp E-Series storage systems. NetApp has partnered with Amazon Web Services (AWS) to allow full-block and file-based backup and recovery of E-Series volumes to Amazon Simple Storage Service (Amazon S3) and NetApp StorageGRID®. Available for installation on Red Hat and SUSE Linux systems, the SANtricity Cloud Connector is a packaged solution (.bin file). After it is installed, you can configure the SANtricity Cloud Connector to perform backup and restore jobs for E-Series volumes to your existing Amazon S3 or StorageGRID account. All jobs performed through the SANtricity Cloud Connector use RESTful APIs. This report provides the technical details for each feature, general limitations, and specific restrictions that must be considered to successfully use the features.

NetApp®

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

The NetApp SANtricity Cloud Connector is a host-based Linux application that allows you to perform full-block and file-based backup and recovery of NetApp E-Series volumes to Amazon S3 and NetApp StorageGRID. Available for installation on Red Hat and SUSE Linux systems, the SANtricity Cloud Connector is a packaged solution (.bin file). After it is installed, you can configure the SANtricity Cloud Connector to perform backup and restore jobs for E-Series volumes to your existing Amazon S3 or StorageGRID account. All jobs performed through the SANtricity Cloud Connector use RESTful APIs.

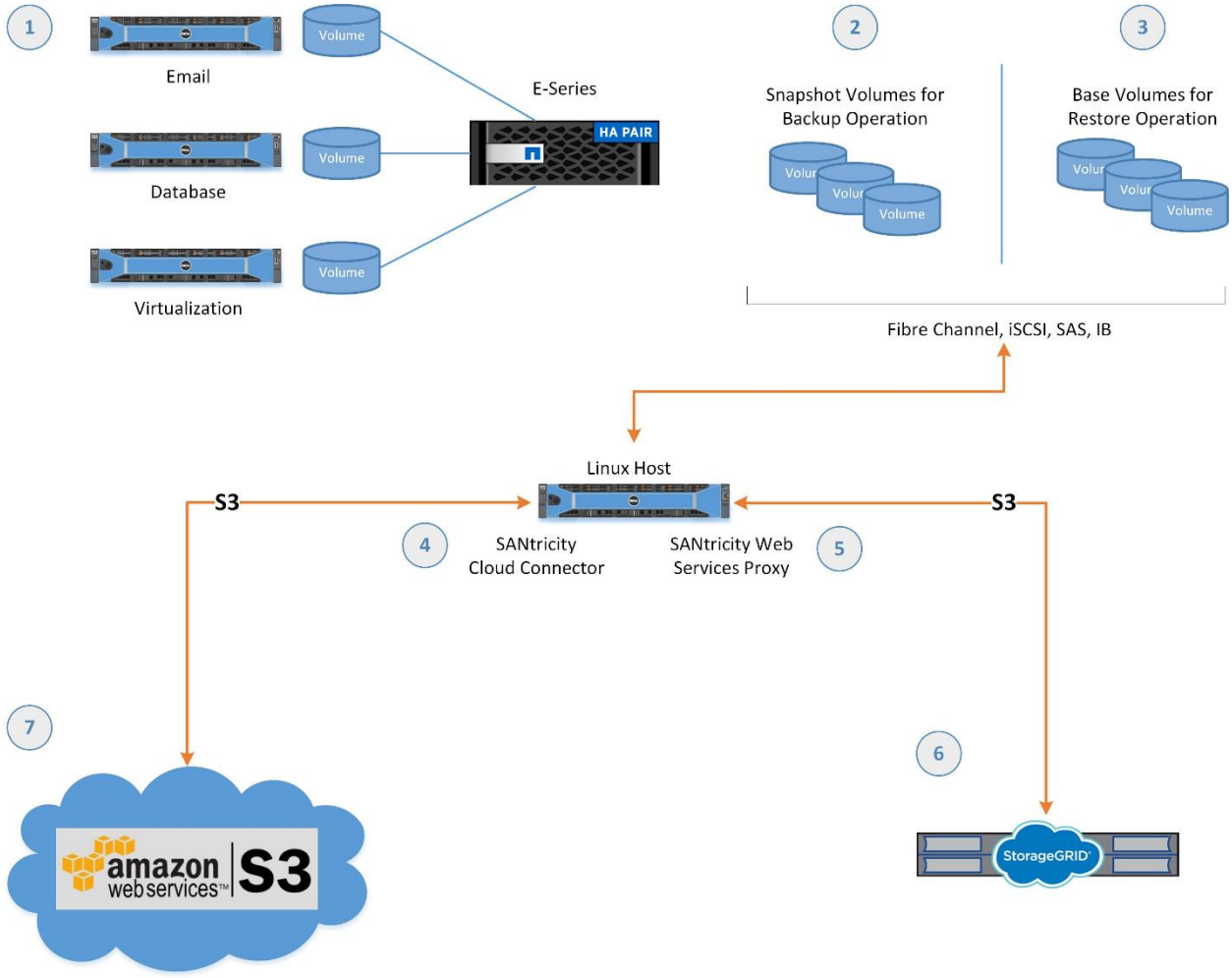
2 Overview

2.1 Terminology

- **SANtricity Cloud Connector host.** The Linux host where SANtricity Cloud Connector is installed.
- **Volume or base volume.** An E-Series volume that is typically mapped to a host system through LUN mapping.
- **Backup volume or backup base volume.** The volume or base volume selected for a backup.
- **Backup target.** The destination where the backed-up data is stored. This is either an Amazon S3 bucket or NFS mount point, based on the intended destination.
- **Restore source.** The Amazon S3 bucket or NFS mount point where data is restored from.
- **Restore volume or restore base volume.** The volume or base volume selected to receive the restored data.

2.2 Workflow Diagram

Figure 1) Solution workflow diagram.



1. Customer production environment. Direct-attached storage (DAS) or SAN environment with E-Series storage arrays.
2. NetApp Snapshot™ volumes for backup operation. Before you create a Snapshot volume of the base volumes, you must manually quiesce any application by using the base volumes for data consistency. Snapshot volumes (copy-on-write) are then created from the base volumes from the E-series array. The Snapshot volumes are manually mapped to the SANtricity Cloud Connector Server through host interconnect.
3. Base volumes for restore operation. During a restore operation, base volumes are mapped to the SANtricity Cloud Connector server through host interconnect. The restore data files are read from restore source to reconstruct the restore volume.
4. SANtricity Cloud Connector. A host-based Linux application that allows you to perform backup and restore of E-Series volumes. All backup and restore jobs performed use RESTful API calls to the application. Backup operation consists of reading data from the Snapshot volumes and copying the data to one of several destinations:
5. AWS (through HTTPS and S3 protocols).
6. StorageGRID (through HTTPS and Amazon Simple Storage Service protocols).

7. The restore operation consists of reading the data from the restore source and reconstructing the data to the restore volumes.
8. SANtricity Web Services Proxy. Used by SANtricity Cloud Connector to communicate with the E-Series array.
9. StorageGRID solution.
10. Bucket for AWS cloud account.

3 System Requirements

3.1 Compatible Storage Arrays

For a complete and up-to-date listing of all compatible storage arrays and firmware for the SANtricity Cloud Connector, see the NetApp Interoperability Matrix Tool.

3.2 Compatible Operating Systems

The SANtricity Cloud Connector 4.0 application is compatible with and supported on the following operating systems shown in Table 1.

Table 1) Operating system requirements.

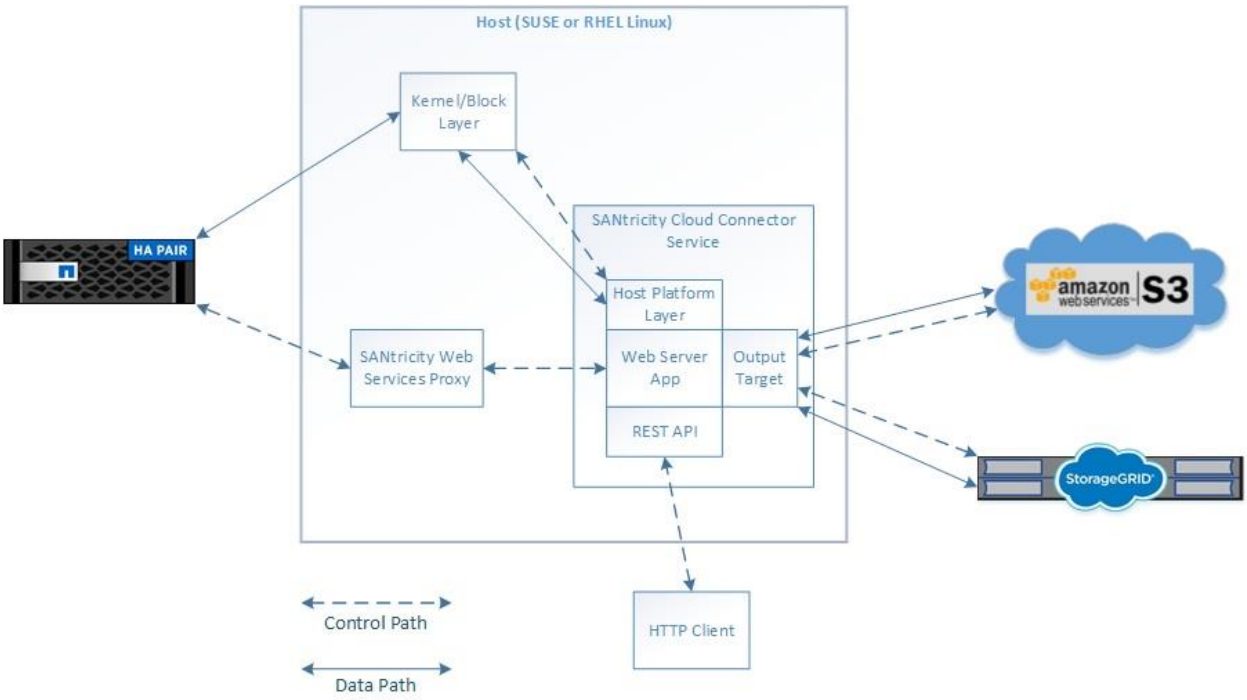
Operating System	Version	Architecture
Red Hat Enterprise Linux	7.x	64-bit
SUSE Linux Enterprise Server	12.x	64-bit

4 Technical Overview

4.1 Solution Block Diagram

Figure 2 provides an overview of SANtricity Cloud Connector.

Figure 2) Solution block diagram



SANtricity Cloud Connector

The SANtricity Cloud Connector web server application runs in the background as a Linux service. It is responsible for coordinating all backup- and restore-related activities. It is divided into four main components:

- **REST API**. This component provides structured output to HTTP requests using standard HTTP verbs and URL structure. It is primarily used by the SANtricity Cloud Connector GUI.
- **Host platform layer.** The backup and restore of user data between E-Series storage systems and backup target passes through the SANtricity Cloud Connector host and SANtricity Cloud Connector application. This component handles the operating system-specific responsibilities for discovery of E-Series volumes in both image and file-based backups.
- **Output target.** This component handles the target-specific requirements associated with the management of user data including data transfer, deletion, inventory, and so on.
- **Web server application.** This component handles the bulk of responsibilities within SANtricity Cloud Connector:
  - Manages the jobs that perform a backup of E-Series volumes to the backup target. The volume data read by the host platform layer and transferred to the web server application which performs deduplication, compression, and encryption of the data. Finally, the data is handed over to the output target component for transfer.
  - Manages the jobs that perform a restore of E-Series volumes from the backup target. The volume data is read by the output target and handed over to the web server application which performs decompression and decryption of the data. Finally, the data is handed over to the host system layer where it is written to the E-Series volume.

SANtricity Web Services Proxy (Installed Separately)

The SANtricity Web Services Proxy web server application runs in the background as a Linux service and provides configuration access to NetApp E-Series storage systems through standard HTTPS mechanisms.

4.2 Data Chunks and Security

Data from an image-based or file-based backup are stored in a series of data chunks that are 64MB. For security, each data chunk uses a separate encryption key to encrypt the data. The key is an SHA256 hash consisting of a combination of a user supplied passphrase and the SHA256 hash of the user data. A random initialization vector is generated and used to feed a CBC cipher that encrypts the hash of the original block and the compressed block. The initialization vector is saved with the encrypted block.

The data chunk is stored as an object in Amazon S3 or StorageGRID with the object's hash as the object key.

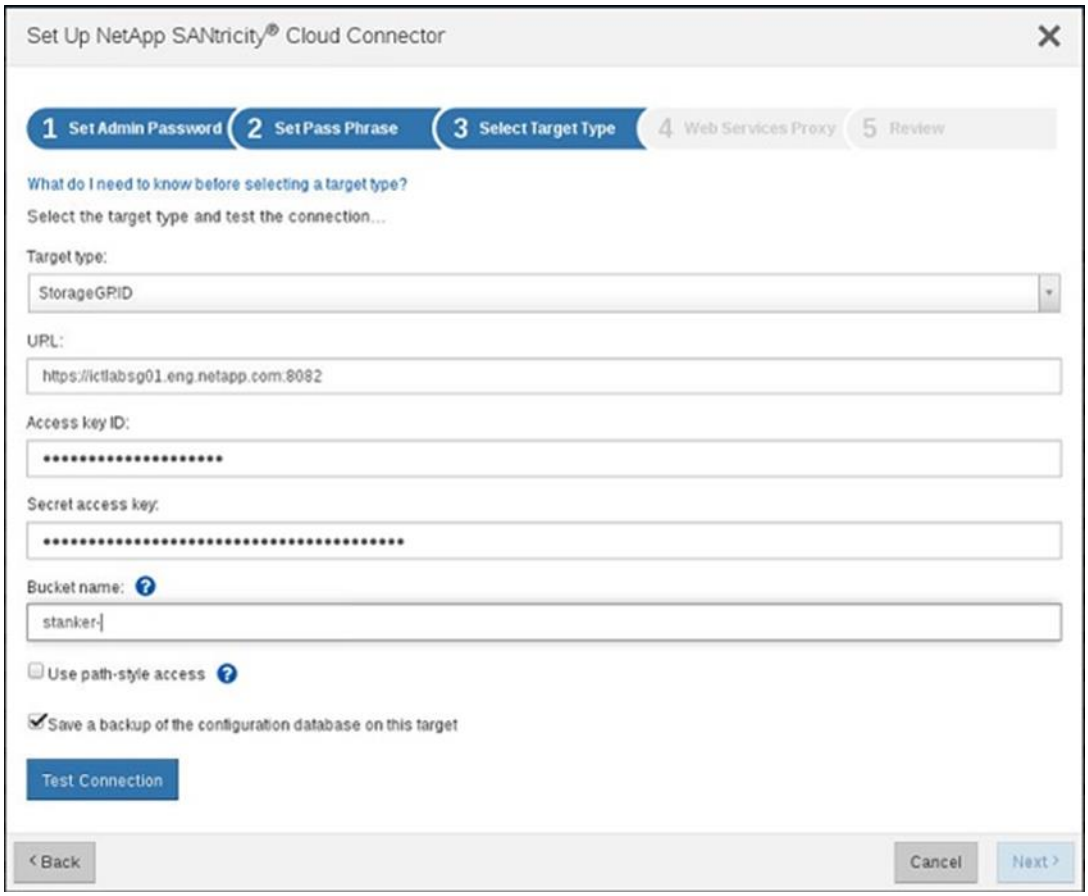
4.3 Configuration Database

A SANtricity Cloud Connector configuration contains the following metadata that is used to manage the backup and restore operations:

- Backup sets
- Base and incremental backups
- Endpoint information
- Status of backup and restore jobs
- Password and pass phrase
- And more

This information is kept in a configuration database that resides on the SANtricity Cloud Connector Host. Because there is no other way to restore data from the backup target if the database is lost, make sure that you enable the database backup. To enable the database backups, you select the target type when you initially set up SANtricity Cloud Connector.

Figure 3) Enabling configuration database backups.



When database backups are enabled, SANtricity Cloud Connector automatically backs up the database to the target location after each base or incremental backup request completes. Database backups are only intended to be used as a disaster recovery mechanism. For example, if the current database is corrupt, if the SANtricity Cloud Connector host has crashed and is unrecoverable, or if you decide to move the installation to another host.

You can restore from a database backup through one of two ways:

- The first way is during initial setup of SANtricity Cloud Connector. When you select the database backup option, SANtricity Cloud Connector establishes a connection to the target and automatically determines whether a database backup exists. If so, you are asked whether you want to restore from the backup.

  Note: This method overwrites any database currently on the SANtricity Cloud Connector host and is irreversible.

- The second way to use the SANtricity Cloud Connector API to call `POST:/database/restore`.

  Note: This method overwrites any database currently on the SANtricity Cloud Connector host and is irreversible.

The database backup process creates a temporary point-in-time (PiT) copy of the database, which might consist of multiple files. The PiT copy typically takes a few seconds to perform, when access to the database is blocked. After the copy completes, a background process compresses and encrypts the database files and then sends them to the backup target. The same encryption process described for image and file-based backups is used for the database files, except the chunks are not 64MB. If the database backup process is successful, any previous PiT backups are removed. For Amazon S3 and

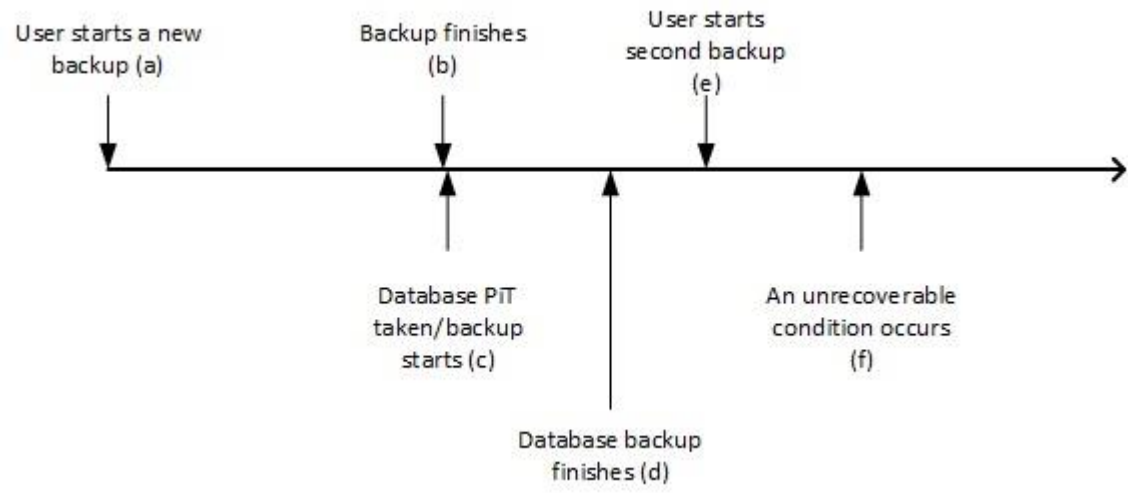
StorageGRID, the object keys associated with any database files have “database/” appended to them, which mimics a folder hierarchy found in traditional file systems.

The database restore process retrieves a copy of the database files from the backup target and then temporarily blocks access to the database while the retrieved files are uncompressed, decrypted, and written over database files that might exist on the SANtricity Cloud Connector host. The database is typically blocked for a few seconds.

Database Restore Considerations

Consider the following diagram:

Figure 4) Example backup timeline—database restore.



In this example, a PiT copy of the database is created and backed up (c and d) after the user completes a backup of their data (a and b). The user starts a second backup, which doesn’t complete due to an unrecoverable condition on the host (for example, the host’s hard drive crashes). The user can install SANtricity Cloud Connector on another host, walk through the initial setup, and then recover their database from the backup target. However, any backup information regarding the second backup is not available.

SANtricity Cloud Connector tracks each data chunk (associated with user data) that is sent to the backup target in the configuration database, so deleting a base or incremental backup automatically deletes the data chunks from the backup target. In the scenario described, a recovered database does not have information about the data chunks sent to the backup target by the second backup (e). Therefore, you can have orphaned data chunks. Currently, SANtricity Cloud Connector does not provide a feature to automatically remove these data chunks but the following workaround is suggested.

Note: An example Python script showing how these steps are used is provided in the Appendix under `‘getDataChunkList.py’`.

1. Log in to SANtricity Cloud Connector API to obtain an authorization token, which is then used with subsequent API requests. Call `POST:/login` with the correct password and an authorization token is returned.
2. Use the SANtricity Cloud Connector API to call `GET:/endpoint/configuration`, which returns a list of all configured endpoints. At the end of the list is an item named `defaultBackupTargetEndpointConfiguration` followed by unique identifier. This unique identifier is used in the step3.

   For example:

```
defaultBackupTargetEndpointConfiguration: "833348a9-e731-43ca-a2fc-be2c02d1bd4a"
```

3. Use the SANtricity Cloud Connector API to call `POST:/database/store/chunk-inventory/{<unique_identifier>}`.

   For example:

```
POST:/database/store/chunk-inventory/{833348a9-e731-43ca-a2fc-be2c02d1bd4a}
```

4. This action returns a list of data chunks that the service knows about. Data chunks related to the configuration database are not included.
5. Obtain the current list of objects or files at the backup target. For Amazon S3 and StorageGRID, the target location is a bucket.
6. Compare the two lists from steps 2 and 3. Any chunk that exists in the list from step 3 but not in the list from step 2 can be considered candidates for removal.

4.4 Snapshot Feature

SANtricity Cloud Connector uses the Snapshot feature provided by E-Series storage systems for performing image or file-based backups. This section provides a quick summary of the Snapshot feature.

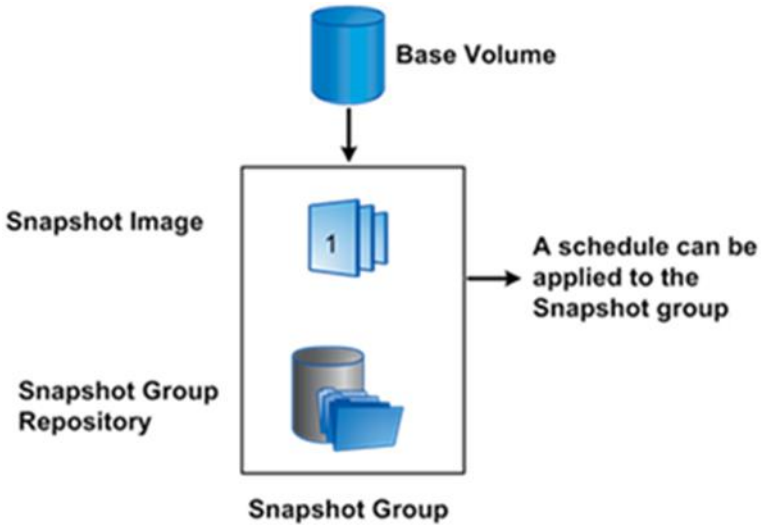
A Snapshot image is a PiT logical copy of a user volume (also known as a base volume) that ensures its contents do not change while the backup is in progress.

To maintain the PiT image of the base volume, Snapshot images use a repository to keep the original data blocks for those blocks that change in the base volume after the Snapshot image is taken. The repository is automatically created when the Snapshot group is created, and is, by default, 40% of the size of the base volume. Snapshot images are logical copies of the base volume, so they do not take any capacity from the storage system other than the capacity that the repository requires.

Snapshot Image

Figure 5 shows how Snapshot images are created from a base volume at specific PiT.

Figure 5) Snapshot image and Snapshot group.



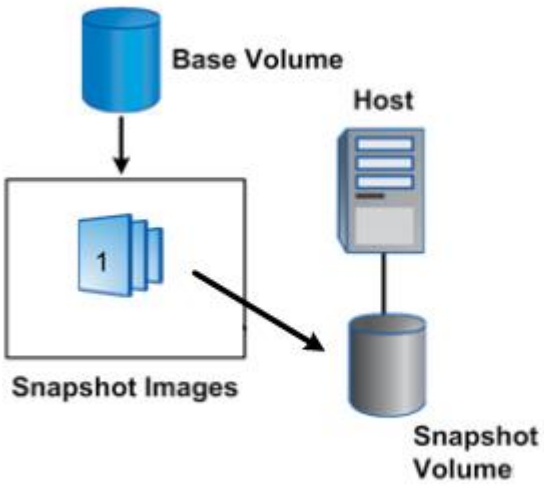
Snapshot images are collected into a Snapshot group that is associated only with a unique base volume. When the first Snapshot image of the base volume is created, if a Snapshot group does not yet exist, SANtricity Cloud Connector automatically also creates a Snapshot group in which to place that Snapshot image. Each Snapshot group includes a Snapshot group repository in which to save Snapshot images. Although multiple Snapshot images are possible, SANtricity Cloud Connector only creates one per Snapshot group.

The base volume that is associated with a Snapshot group can reside in a disk pool or in a volume group. If the base volume resides in a volume group, the repository members for any associated Snapshot group can reside in a disk pool or in a volume group. If, however, the base volume resides in a disk pool, all repository members for any associated Snapshot group must reside in the same disk pool as the base volume.

Snapshot Volume

A Snapshot volume is a logical entity that is structured on top of a Snapshot image, as illustrated in Figure 6.

Figure 6) Snapshot volume.

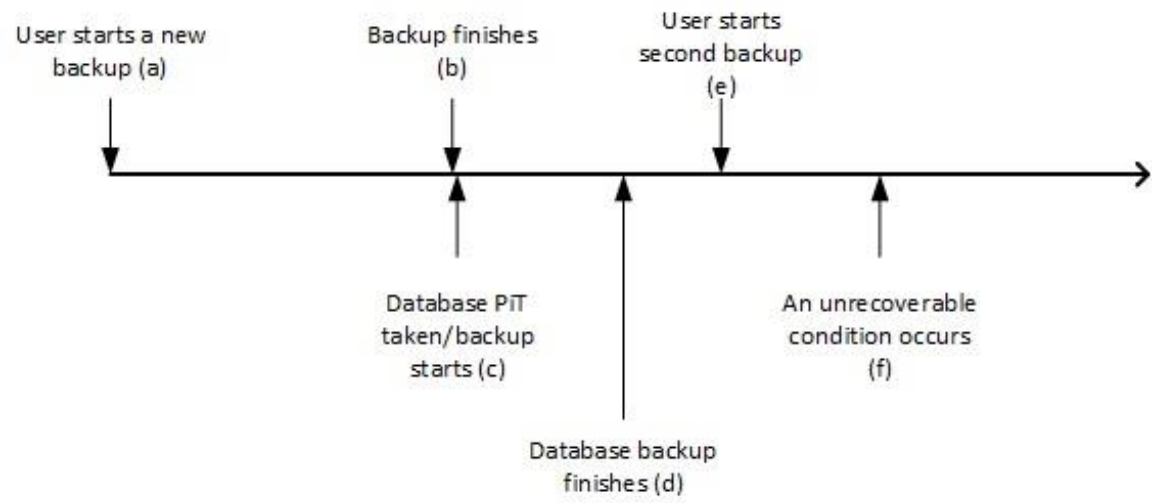


Snapshot images are not directly read/write accessible to hosts. To map a Snapshot image to a host, a Snapshot volume is created from the Snapshot image. The Snapshot volume has the same characteristics as the base volume (such as RAID level and I/O characteristics). However, the Snapshot volume is viewed as a separate standard volume by the storage system; therefore, it can be mapped to a host, read from, or written to. SANtricity Cloud Connector automatically creates a Snapshot volume when an image or file-based backup takes place. A Snapshot volume is designated as read-only so no associated repository is required.

Snapshot Considerations

Consider the following diagram:

Figure 7) Example backup timeline – Snapshot volume.



In this example, a PiT copy of the database is created and backed up (c & d) after the user has completed a backup of their data (a and b). The user starts a second backup, which doesn't complete due to an unrecoverable condition on the host (ex. the host's hard drive crashes). Currently, SANtricity Cloud Connector does not provide a feature to automatically remove the Snapshot image and volume created for the second backup when this condition happens so the following workaround is suggested. An example Python script showing how these steps are used is provided in the Appendix under `'delSnapshotVolume.py'`.

1. Log in to SANtricity Cloud Connector API to obtain an authorization token, which is then used with subsequent API requests. Call `POST:/login` with the correct password and an authorization token is returned.
2. Use the SANtricity Cloud Connector API to call `GET:/storage/eligible-volumes` which returns a list of all volumes eligible for use in a backup. Each volume in the list returns information similar to the following example:


```
{
 "capacity": "2147483648",
 "storageSystemWwn": "600A098000AF3E1000000000590A8AE3",
 "volumeName": "amyv-3",
 "snapshotVolumes": [
  {
   "pitTimeStamp": "Tue Apr 09 15:09:36 CDT 2019",
   "label": "SCC_SV_2019-04-09_150937",
   "wwn": "600A098000AF3E1000004AB35CAD6D4F",
   "id": "35000000600A098000AF3E1000004AB35CAD6D4F",
   "mappedLocally": false,
   "reportedCapacity": "2147483648"
  },
 ],
 "volumeId": "02000000600A098000AF3E10000049635CAC7E93",
 "storageSystemName": "icta-SH-148078",
 "mappedLocally": false,
 "volumeWwn": "600A098000AF3E10000049635CAC7E93"
}
```


This example shows a volume named "`amyv-3`" that belongs to storage array "`icta-SH-148078`" and has a Snapshot copy created for it named "`SCC_SV_2019-04-09_150937`". Snapshot volumes are created by SANtricity Cloud Connector are named `SCC_SV_<year>-<month>-<day>_<hour><minute><second>` for easy identification.

Note: The `storageSystemWwn` and Snapshot volume `wwn` fields (`600A098000AF3E1000000000590A8AE3` and `600A098000AF3E1000004AB35CAD6D4F.`)

3. Use the SANtricity Cloud Connector API to call `POST:/storage/storage-systems/{system-wwn}/volumes/{volume-wwn}`, which cleans up the Snapshot copies.

   For example:

```
POST:/storage/storage-
systems/600A098000AF3E1000000000590A8AE3/volumes/600A098000AF3E1000004AB35CAD6D4F
```

5 Restrictions

5.1 Backups

When you delete a completed backup, the data chunks associated with the backup are not removed from the backup location.

SANtricity Cloud Connector does not currently remove the data chunks associated with the backup.

Note: Follow the workaround described in section “Database Restore Considerations."

5.2 Restore Operations

When you perform a restore operation, files with the same path and name that exist on the restore volume is overwritten.

For example, when you perform a backup, you have a file named `'/home/myUsername/directory1/myFile'`. When you restore this file to the restore volume, it overwrites anything that exists at `'/home/myUsername/directory1/myFile'`.

6 Troubleshooting

6.1 Backups

When you create a file-based backup, you are asked to select the file system to backup; however, the dialog shows no file systems.

SANtricity Cloud Connector requires the volume containing the file system to be recognized and mounted to the SANtricity Cloud Connector Host. Several reasons why the volume might not be recognized include:

- You selected a volume that does not have the file system you want to back up.

  Solution: Select the correct volume with the file system you want to back up and re-attempt the backup.

- Timing delays in the host discovery process. SANtricity Cloud Connector calls the `rescan-scsi-bus.sh` script to perform discovery of volumes. After this script completes, control is returned to SANtricity Cloud Connector which immediately proceeds to discover what file systems are available. On some systems, a slight delay might be needed between the time the rescan script completes and discovery of file systems occurs.

  Solution: Add a delay to the end of the `scc_rescan.sh` script, located in the SANtricity Cloud Connector installation directory (by default, /`opt/netapp/santricity_cloud_connector4`).

  For example:

```
            # .... script code removed here for brevity ...
            if grep -rnw /etc/*release -e "Red Hat"; then
```

```
        echo "Executing $REDHAT_RESCAN_SCRIPT"
        eval $REDHAT_RESCAN_SCRIPT
    else
        echo "Executing $DEFAULT_RESCAN_SCRIPT"
        eval $DEFAULT_RESCAN_SCRIPT
    fi

    # wait a few seconds before returning back to Cloud Connector
    sleep 3
```

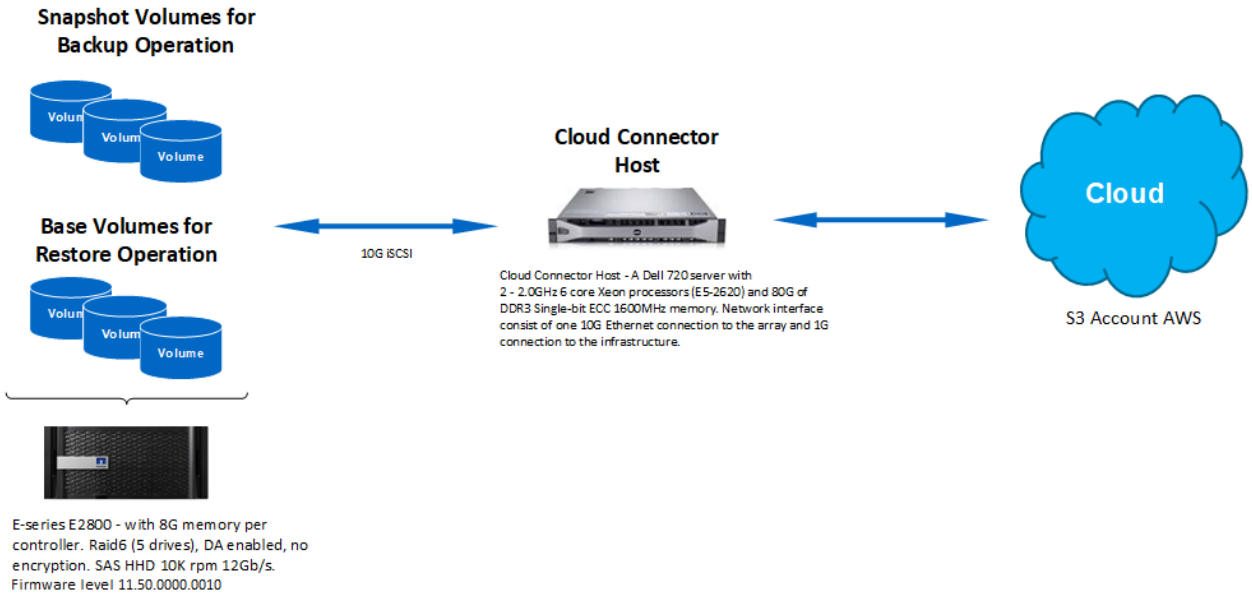
7 Benchmark Results

7.1 Overview

The purpose of benchmarking SANtricity Cloud Connector 4 is to show the performance numbers between the E-series storage system (ESS), Cloud Connector Host (CCH), and AWS when performing backup and restore operations from the CCH. The duration of a backup operation to AWS and restore operation back to an ESS are recorded as well.

Figure 2 illustrates a configuration that is used to demonstrate SANtricity Cloud Connector performance between a NetApp E-Series storage system and AWS.

Figure 8) SANtricity Cloud Connector performance demonstration.



The following components were included in this test system:

- Dell 720 server with 2-2.0GHz 6 core Xeon processors and 80GB of DDR3 single-bit ECC 1600MHz memory.
- An E-Series E2800 using 8GB of memory per controller was chosen as the storage array.
- The drives used in the storage system were SAS HDD 10K RPM 12Gbps, and the firmware was 11.50.0000.0010.
- Three volumes were created from an 11-driver DDP configuration with data assurance (DA) enabled and no encryption. The sizes chosen for the three volumes were 50GB, 500GB, and 1TB.

- The interconnect between the E-Series E2800 and the SANtricity Cloud Connector host was 10Gbps iSCSI.
- The internal network infrastructure between the SANtricity Cloud Connector Host and firewall to the external internet was 1GB.

  We used Cloud Harmony's speed test for AWS to help gauge the latency of the network between the SANtricity Cloud Connector host and AWS.

7.2 Image-Based Backup and Restore Results

The benchmarks presented in Table 2 represent an image-based backup and restore scenario. In these tests, the whole volume's capacity is written with random data. This situation represents a worst-case backup scenario because it restricts SANtricity Cloud Connector's ability to deduplicate data prior to backup.

Table 2) Image-based backup and restore results.

Host to Target Description	Backup			Restore		
SANtricity Cloud Connector host to and from storage (10Gb iSCSI)	50GB	500GB	1TB	50GB	500GB	1TB
Rx Mibps	72.1	76.2	79.9	0.03	0.03	0.01
Tx Mibps	0.17	0.17	0.10	30.8	28	6.7
Total throughput	72.3	76.4	80	30.8	28	6.7
SANtricity Cloud Connector host to and from AWS (avg. latency ~ 46ms)						
Rx Mibps	0.97	1	0.1	29.2	27.3	6.6
Tx Mibps	57.2	59.7	76.9	0.1	0.09	0.01
Total throughput	58.2	60.7	77	29.3	27.4	6.6
Total transfer time (sec.)	840	7440	14,580	1,804	19,140	166,320
SANtricity Cloud Connector host[1] utilization						
CPU % average	28.9	21.2	26.9	3	6.3	1.5
Memory % average	7.4	9.7	9.5	7.6	10.2	9.9
NetApp E-Series E2800[2] utilization						
IOPS average	100	150	150	20	20	5
CPU % average	35	35	35	35	35	35

[1] SANtricity Cloud Connector host - A Dell 720 server with two 2.0GHz 6-core Xeon processors (E5-2620) and 80GB of DDR3 single-bit ECC 1600MHz memory. The network interface consists of one 10G Ethernet connection to the array and a 1Gb connection to the infrastructure.

[2] NetApp E-Series E2800 with 8GB memory per controller. DDP (11 drives), DA enabled, and no encryption. SAS HHD 10K RPM 12Gbps. Firmware level 11.50.0000.0010.

7.3 File-Based Backup and Restore Results

These benchmarks represent a file-based backup and restore scenario. In these tests, we formatted the volumes with the ext4 file system, and we wrote directories and files with random data.

Table 3) File-based backup and restore results.

Host to Target Description	Backup			Restore		
SANtricity Cloud Connector host to and from storage (10Gb iSCSI)	50GB	500GB	1TB	50GB	500GB	1TB
Rx Mibps	73.1	72.4	92.9	30.1	30.3	6.3
Tx Mibps	0.1	0.1	0.2	0.2	0.03	6.3
Total throughput	73.2	72.5	93.1	30.3	30.3	12.7
SANtricity Cloud Connector host to and from AWS (avg. latency ~ 46ms)						
Rx Mibps	0.8	0.6	0.1	30	30.1	6.3
Tx Mibps	39.8	38.2	45.5	0.1	0.1	0.01
Total throughput	40.6	38.9	45.6	30.1	30.2	6.3
Total transfer time (sec.)	1,360	13,920	22,020	2,050	21,060	160,020
SANtricity Cloud Connector host[1] utilization						
CPU % average	13.1	13.3	19.4	8	7	1.4
Memory % average	10.1	10	9.1	10	10	8.6
NetApp E-Series E2800[2] utilization						
IOPS average	100	100	135	30	30	10
CPU % average	35	35	35	35	35	35

[1] SANtricity Cloud Connector host - A Dell 720 server with two 2.0GHz 6-core Xeon processors (E5-2620) and 80GB of DDR3 single-bit ECC 1600MHz memory. The network interface consists of one 10G Ethernet connection to the array and a 1Gb connection to the infrastructure.

[2] NetApp E-Series E2800 with 8GB memory per controller. DDP (11 drives), DA-enabled, and no encryption. SAS HHD 10K RPM 12Gbps. Firmware level 11.50.0000.0010.

7.4 Conclusions from The Benchmark Test

- Backup and restore operations are consistent between the volume size.
- The effect on the SANtricity Cloud Connector host and the E-Series storage array is minimal.
- The backup operation uses multithreaded processes to help with performance. Unfortunately, the restore operation is restricted to a single thread.

- These results show that the image-based backup and restore performs better than the file-based backup and restore. However, keep in mind that the table is showing worst-case scenarios. Also keep in mind that an image-based restore must restore the full contents of the volume, whereas a file-based restore provides the option to only restore a subset of the data.
- Connection bandwidth can be thought of in two ways:
  - **The bandwidth of the network connection that you have to the backup target itself.** A typical business might have 100Mb or 1GbE connections for its internal network. However, connecting to an external backup target (such as Amazon S3) over consumer-grade internet service provides only a fraction of the internal network speed. Faster internet speeds might not be possible, depending on cost and customer location. This situation affects both backup and restore operations.
  - **Bandwidth limits imposed by the backup target service.** Services such as Amazon S3 can offer different levels of service based on connection speeds. This situation affects both backup and restore operations.
- Array-level performance. The SANtricity Cloud Connector application relies on the underlying NetApp E-Series and EF-Series array firmware to perform the following tasks:
  - The initial Snapshot copy of the base volume
  - The discovery of the SANtricity Cloud Connector host interface and multipath driver
  - The designation of base Snapshot copy volumes (LUNs) to the SANtricity Cloud Connector host

These operations might take place while the storage array is serving other workloads, so the storage array Snapshot feature performance overhead and time to complete a Snapshot copy can vary. The SANtricity Cloud Connector application does not change the performance profile associated with normal SANtricity Snapshot feature functionality. As a result, deciding when to initiate Snapshot copies includes pausing all write I/O to the target volume and some consideration for other workloads the array is simultaneously serving. For example, to minimize the performance overhead associated with initial Snapshot copies, consider taking Snapshot copies during the storage array's lowest overall I/O processing window.

Appendix A: Sample Scripts

getDataChunkList.py

This example script shows how to retrieve a list of data chunks that SANtricity Cloud Connector is tracking for backup and restore purposes.

```
#!/usr/bin/python
# This example script shows how to use the Cloud Connector API to obtain a
# list of data chunks it is currently managing.  The script does not make an attempt
# to account for all possible exception conditions!  Use this code at your own risk!
import requests
import json
import time

# URL of Cloud Connector service.  Replace with your actual URL.
srvAddress = "http://localhost:8080/v2"

# Password information.  Replace the default with your actual password.
password = "password"


# Almost all Cloud Connector APIs require an authorization token, which is obtained via login.
def login(passwd):
        headers = {
                'Content-Type': 'application/json',
                'Accept': 'application/json'
        }
```



```
        payload = {'password': passwd}

        print("Logging into Cloud Connector service")
        url = srvAddress + "/login"
        req = requests.post(url, headers=headers, data=json.dumps(payload))
        if req.status_code != 200:
                exMsg = "Login failed (status {0}): {1}".format(req.status_code, req.content)
                raise Exception(exMsg)

        resp = req.json()
        return resp['token']


def getEndpointConfiguration(token):

        tokenStr = 'Bearer ' + token
        headers = {
                'Content-Type': 'application/json',
                'Accept': 'application/json',
                'Authorization': tokenStr
        }

        print("Get the default backup target endpoint Id")
        url = srvAddress + "/endpoint/configuration"
        req = requests.get(url, headers=headers)
        if req.status_code != 200:
                exMsg = "GetEndpointConfiguration failed (status {0}):
{1}".format(req.status_code, req.content)
                raise Exception(exMsg)

        resp = req.json()
        return resp['defaultBackupTargetEndpointConfiguration']


def getDbChunkInventory(token,endptCfgId):

        tokenStr = 'Bearer ' + token
        headers = {
                'Content-Type': 'application/json',
                'Accept': 'application/json',
                'Authorization': tokenStr
        }

        print("Get the chunk inventory for endpoint Id "+ endptCfgId)
        url = srvAddress + "/database/store/chunk-inventory/{" + endptCfgId + "}"
        req = requests.get(url, headers=headers)
        if req.status_code != 200:
                exMsg = "GetDbChunkInventory failed (status {0}): {1}".format(req.status_code,
req.content)
                raise Exception(exMsg)

        resp = req.json()
        return resp



try:
        authToken = login(password)
        endptCfg = getEndpointConfiguration(authToken)
        chunkList = getDbChunkInventory(authToken,endptCfg)

        print chunkList['dataChunkList']['dataChunks']

except Exception as ex:
        print ex
```


delSnapshotVolume.py

This example script shows how to remove Snapshot copies created by SANtricity Cloud Connector for backup operations but were not cleaned up properly.

```
#!/usr/bin/python
# This example script shows how to use the Cloud Connector API to obtain a
# list of data chunks it is currently managing.  The script does not make an attempt
# to account for all possible exception conditions!  Use this code at your own risk!
import requests
import json
import time

# URL of Cloud Connector service.  Replace with your actual URL.
srvAddress = "http://localhost:8080/v2"

# Password information.  Replace the default with your actual password.
password = "password"

# Volume name of the base volume
baseVolName = "MyBaseVolume"


# Almost all Cloud Connector APIs require an authorization token, which is obtained via login.
def login(passwd):
        headers = {
                'Content-Type': 'application/json',
                'Accept': 'application/json'
        }

        payload = {'password': passwd}

        print("Logging into Cloud Connector service")
        url = srvAddress + "/login"
        req = requests.post(url, headers=headers, data=json.dumps(payload))
        if req.status_code != 200:
                exMsg = "Login failed (status {0}): {1}".format(req.status_code, req.content)
                raise Exception(exMsg)

        resp = req.json()
        return resp['token']


def getEligibleVolumes(token):

    tokenStr = 'Bearer ' + token
    headers = {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
        'Authorization': tokenStr
    }

    print("Obtain list of eligible volumes")
    url = srvAddress + "/storage/eligible-volumes"
    req = requests.get(url, headers=headers)
    if req.status_code != 200:
                exMsg = "GetEligibleVolumes failed (status {0}): {1}".format(req.status_code,
req.content)
                raise Exception(exMsg)

    resp = req.json()
    return resp


def unmapAndDeleteSnapshot(token, systemWwn, snapWwn):

        payload = {
                "volumeOperation": "unmap",
                "snapshot": "true"
        }
```

```
        print("Cleaning up snapshot " + snapWwn)
        url = srvAddress + "/storage/storage-systems/" + systemWwn + "/volumes/" + snapWwn
        print(url)

        postPayloadRequestWithAuthorization(token, url, payload)


def postPayloadRequestWithAuthorization(token, url, payload):
        tokenStr = 'Bearer ' + token
        headers = {
                'Content-Type': 'application/json',
                'Accept': 'application/json',
                'Authorization': tokenStr
        }

        req = requests.post(url, headers=headers, data=json.dumps(payload))
        if req.status_code != 200:
                exMsg = "PostRequest failed (status {0}): {1}".format(req.status_code,
req.content)
                raise Exception(exMsg)

        resp = req.json()
        print json.dumps(resp, indent=1)


def delSCCSnapshotsByBaseVolName(token, volName):

        print("Determine what snapshot volumes exist for the base volume named '" + volName +
"'")
        for bv in vols['volumes']:
                if bv['volumeName'] == baseVolName:
                        if len(bv['snapshotVolumes']) == 0:
                                exMsg = "No snapshots exist for base volume."
                                raise Exception(exMsg)

                        systemWwn = bv['storageSystemWwn']
                        print("Base volume found.  Array '" + systemWwn + "'")
                        for sv in bv['snapshotVolumes']:
                                if sv['label'].startswith('SCC_SV_'):
                                        snapWwn = sv['wwn']
                                        unmapAndDeleteSnapshot(token, systemWwn, snapWwn)



try:
        authToken = login(password)
        vols = getEligibleVolumes(authToken)
        delSCCSnapshotsByBaseVolName(authToken,baseVolName)

except Exception as ex:
        print ex
```

Where to Find Additional Information

To learn more about the information that is described in this document, review the following documents and/or websites:

- E-Series and SANtricity Documentation Center
  https://docs.netapp.com/ess-11/index.jsp
- E-Series and SANtricity Documentation Resources page
  https://www.netapp.com/us/documentation/eseries-santricity.aspx
- SANtricity Cloud Connector
  http://mysupport.netapp.com/NOW/download/software/eseries_cloudconnector/

- SANtricity Web Services Proxy
  http://mysupport.netapp.com/NOW/download/software/eseries_webservices/

Version History

Version	Date	Document Version History
Version 1.0	September 2017	Initial release.
Version 2.0	May 2018	Revised for version 3.1.
Version 3.0	May 2019	Revised for NetApp SANtricity® Cloud Connector version 4.0.
Version 4.0	August 2019	Updated benchmarking results based on tests performed with firmware level 11.50.
Version 4.1	January 2020	Removed Elliptic Curve Cryptography (ECC) abbreviation breakout from the Table 2 and 3 notes.

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

TR-4658-0120