Technical Report

IBM DB2 and NetApp FlexClone: Business Use Cases

Bobby Oommen, Jawahar Lal, NetApp
March 2011 | TR-3460

EXECUTIVE SUMMARY

The ability to create copies of a DB2 database quickly and easily can be extremely valuable. NetApp® FlexClone® technology that is available on NetApp FAS or IBM N series storage systems with NetApp Data ONTAP® 7.0 or higher can be used to create an instantaneous point-in-time copy of a production database by creating a space-efficient, exact copy of the storage system volumes that are used to hold the database's data and transaction logs. The cloned database can be put in use for development, to test FixPack upgrades or migration, and to analyze any other problems. This technical paper describes the cloning process and business use cases in which FlexClone can be extremely useful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Database cloning is a process by which you can create an exact copy of a DB2 database, either by physically copying the data or by performing what is known as a redirected restore. Database cloning is performed frequently by database administrators to provide near-production data for various business needs, such as application development, QA testing, and report generation. Traditional methods of cloning a database pose various challenges, including system downtime and degraded system performance during the cloning process. Additionally, a large amount of storage space is required to store each clone. Furthermore, the maintenance overhead can be enormous if each cloned database requires a frequent data refresh.

In this era of high system availability, organizations can't afford extended downtime and degraded performance for their production systems. Therefore, the ability to create a space and time efficient usable database clone quickly and with virtually no effect on the production system is extremely important.

NetApp FlexClone is an advanced and proven cloning technology that helps database and system administrators deliver a near-instantaneous, space-efficient, point-in-time copy of the production database. The database cloning process is completed in a few seconds, with virtually no performance effect on the production system. The clone database is similar to the production database in all aspects, and, unlike the traditional database clone, it consumes no extra disk space at the time of creation. NetApp SnapMirror is a data replication solution that not only can provide disaster recovery protection for your business-critical data, but also enables your DR site for other business activities so you can turn your disaster recovery solution into a business accelerator.

1.1 PURPOSE AND SCOPE

This document describes the process used to create a clone of a DB2 database using FlexClone technology. It also covers creating a database clone on a remote or disaster recovery site that has replicated data by using Data ONTAP SnapMirror technology. Specifically, this document covers the following topics:

- Creating a DB2 database on a NetApp FAS or IBM N series storage system
- Creating a clone of a DB2 database on the same storage system
- Creating a clone of a DB2 database on a remote (disaster recovery) site
- DB2 use cases of FlexClone

1.2 GENERAL ASSUMPTIONS

To get the maximum advantage of the procedures and steps described in this document, we assume that you are familiar with the following:

- Commands and operations of Data ONTAP and the NetApp FAS or IBM N series storage system
- Administration and operations of a DB2 instance and database as well as DB2 utilities such as `db2inidb` and `db2relocatedb`
- UNIX® system administration commands
- File and block access protocols such as Fibre Channel and iSCSI

The assumption is that the NetApp FAS or IBM N series storage systems used are loaded with Data ONTAP 7.0 or later and are licensed for NFS, FCP, iSCSI, FlexClone, and SnapMirror. Additionally, license keys for SnapMirror sync are required for both the source and destination storage systems if synchronous SnapMirror is used.

Another assumption is that the UNIX hosts used to access the production database and the cloned database have the following software and utilities installed:

- DB2 ESE V9.7.
- For a SAN environment, a supported host bus adapter (HBA), SanSurfer utility, and host attach kit are installed and configured. Host attach kits can be downloaded from the NOW™ (NetApp on the Web) site.

1.3 OVERVIEW OF THE FLEXCLONE TECHNOLOGY

FlexClone is a powerful new feature introduced in Data ONTAP 7G that adds a new level of agility and efficiency to storage operations by allowing an individual to create an instant clone of a flexible volume (NetApp FlexVol® volume). A FlexClone volume is a writable point-in-time image of a FlexVol volume or another FlexClone volume. With FlexClone, it takes only a few seconds to create a clone of a FlexVol volume, and such a volume can be created without interrupting access to the parent volume the clone is based on. The clone volume uses space very efficiently, allowing both the original FlexVol volume and the FlexClone volume to share common data, storing only the data that changes between the original volume and the clone. This provides a huge potential saving in storage space, resources, and cost. In addition, a FlexClone volume has all the features and capabilities of a regular FlexVol volume, including the ability to be grown or shrunk and the ability to be the source of another FlexClone volume. A database clone can be created simply by creating a clone of the FlexVol volumes that are used for the database's data and transaction logs.

1.4 OVERVIEW OF THE SNAPMIRROR TECHNOLOGY

Data ONTAP SnapMirror® technology provides an efficient way to transfer data from one NetApp FAS or IBM N series storage system to another. The storage system from which data is transferred is referred to as the SnapMirror source, and the storage system to which data is transferred is referred to as the SnapMirror destination. A SnapMirror source and its corresponding destination can reside on the same storage system or on two separate storage systems that are miles apart (provided that both storage systems are able to communicate with each other over a network). Data changes made at the SnapMirror source are continuously replicated at the SnapMirror destination to keep the data at both locations synchronized.

Data ONTAP provides both asynchronous and synchronous SnapMirror functionality. With asynchronous SnapMirror, each write is acknowledged as soon as it is written to nonvolatile RAM (NVRAM) at the source, even if the destination has not yet received and/or processed the request. Normally, data changes at the SnapMirror source are copied to the destination periodically, based on the schedule defined in the `/etc/snapmirror.conf` file that resides on the destination storage system.

With synchronous SnapMirror, each time a transaction attempts to write data to disk, the data is sent to both the SnapMirror source and the SnapMirror destination in parallel. Not until both storage systems have committed the write to NVRAM does the system acknowledge that the transaction is complete. In other words, the application that initiated the write must wait until it receives the acknowledgement from both the source and destination storage systems before it can continue.

A detailed discussion of SnapMirror is beyond the scope of this paper. For further information on SnapMirror technology, refer to TR-3446: SnapMirror Async Overview and Best Practices Guide.

1.5 NETAPP SNAP CREATOR FRAMEWORK

The NetApp Snap Creator framework is a backup and restore integration framework for integrating applications (specifically non-SnapManager) with NetApp technologies. Snap Creator is designed to be platform and OS independent without any OS or application hooks. Snap Creator is similar to a generic SnapManager product that handles the application consistency while creating a Snapshot backup on the

underlying NetApp storage system. Snap Creator has plug-ins on top of this generic SnapManager layer. A plug-in tells Snap Creator how to handle quiesce and unquiesce for a given application or database. Section 6 covers the Snap Creator use cases.

1.6 PROVIDING CONSISTENCY FOR THE DB2 DATABASE (SUSPENDED I/O)

To obtain a consistent Snapshot® copy for an online recoverable DB2 database, all write operations must be suspended before invoking the Data ONTAP **`snapshot`** command. DB2 provides support for temporarily suspending writes to the database by executing the **`set write suspend`** command. This functionality allows an individual to temporarily suspend all writes to the database and its transaction log files before creating a Snapshot copy. These transactions proceed normally when write operations on the database are resumed.

To suspend write operations to a database, connect to the database and execute the following command:

```
set write suspend for database
```

After Snapshot copies of the FlexVol volumes are created using the **`snapshot`** command, resume write operations to a database by executing the following command on the database server:

```
db2 set write resume for database
```

1.7 THE DB2INIDB COMMAND

This command is used primarily to initialize a Snapshot copy of a database. In the context of creating a clone of a DB2 database from a Snapshot copy, its purpose is to initialize the clone database as a Snapshot image. The syntax for the **`db2inidb`** command is as follows:

```
db2inidb [DatabaseAlias] as [snapshot|standby|mirror] <relocate using
[ConfigFile]>
```

Where

- `DatabaseAlias` identifies the alias assigned to the snapshot (secondary) database that is initialized.
- `ConfigFile` identifies a configuration file that contains information on how the database files contained in the Snapshot copy of the database are relocated when the Snapshot image is initialized.

Note: Parameters shown in angle brackets (< >) are optional. Parameters or options shown in square brackets ([ ]) are required and must be provided. A comma followed by ellipses (…) indicates that the preceding parameter can be repeated multiple times.

The **`db2inidb`** command can also be used to change the database name, the database instance name, and the tablespace header information based on the user-provided configuration file. The configuration file must adhere to the following format:

```
DB_NAME=oldName,newName
DB_PATH=oldPath,newPath
INSTANCE=oldInst,newInst
NODENUM=nodeNumber
LOG_DIR=oldDirPath,newDirPath
CONT_PATH=oldContPath1,newContPath1
CONT_PATH=oldContPath2,newContPath2
-----
STORAGE_PATH= oldStoragePath1,newStoragePath1
STORAGE_PATH= oldStoragePath2,newStoragePath2
```

The sample configuration file looks similar to the following:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

If the database has automatic storage enabled, you must specify changes to the location of the database storage paths (using the `STORAGE_PATH` parameter), not the tablespace containers (using the `CONT_PATH` parameter).

For example, to initialize a DB2 database named `mydbcl`, execute the following command at the database server:

```
db2inidb mydbcl as snapshot
```

As a second example, to initialize a clone DB2 database named `mydbcl`, and to change the database name, the database instance name, and the tablespace header information using the configuration file named `config.txt`, execute the following command at the database server:

```
db2inidb mydbcl as snapshot relocate using config.txt
```

Note: `db2inidb` should never run on the production or the parent database. Instead, it must only be executed against the clone database.

1.8 THE DB2RELOCATEDB COMMAND

The **`db2relocatedb`** command allows a database administrator (DBA) to change the location of one or more tablespace containers or an entire database, without having to perform a backup and a redirected restore operation. It also provides a way to rename a database and/or change the instance to which a database belongs, per specifications in a configuration file that is provided by the user. When executed, this command makes the necessary changes to the DB2 instance and the appropriate database support files. From the clone database perspective, it is used to rename the clone database, change the DB2 instance with which the clone is associated, and change the tablespace container metadata for the tablespace containers that are associated with the clone.

To rename a clone database and update the metadata for its tablespace containers, execute the following command on the database server:

```
db2relocatedb -f [ConfigFile]
```

Where

- `ConfigFile` identifies the name of a configuration file that contains information that is needed to alter the DB2-specific metadata stored in files associated with a database.

The configuration file used must follow the same format specified in section 1.4.

For example, to relocate a DB2 database using a configuration file named `config.txt`, execute the following command on the database server:

```
db2relocatedb -f config.txt
```

1.9 DATABASE USE CASES FOR USING FLEXCLONE TECHNOLOGY

There are abundant business needs that require production or near-production data. These needs can be satisfied by creating a clone of the production database. The added overhead and the complexities associated with the traditional method of creating a clone database pose immense challenges for

database and storage administrators. However, FlexClone technology has made the database cloning process so simple and easy that a clone can be created in a couple of seconds without creating any system overhead. The database clone copies can be created on the primary storage where the production database resides. However, as a best practice, NetApp recommends creating the cloned copies on the secondary or the DR storage. The following key business requirements can benefit greatly from a clone database created by using FlexClone technology:

- **Writable disaster recovery destination.** Generally, a disaster recovery site is equipped with systems that have almost the same capacity and power as the production systems and is not accessible until disaster strikes. The capacity and power of the disaster recovery site systems can be used for various purposes such as reporting, data mining, testing, development, and testing by making the replicated data accessible without affecting the data replication process.
- **Reporting environment.** Business reports need read-only access to the production database. Normally, reporting environments have near-production data and are refreshed based on a frequency level; for example, every night. By cloning a disaster recovery site, you can deliver a highly efficient and low-cost reporting environment.
- **Database software and application upgrade test.** Application upgrades and database software patches can be tested on a cloned database before they are implemented into production.
- **Data mining.** Data-mining software and operations can be implemented with great flexibility because both reads and writes are allowed on a cloned database.
- **Data warehouse/data mart.** In a typical data warehouse architecture, data from operational datastores (ODSs) are copied to staging areas. The staging area is used for performing cleansing and transformation, and as the data source for the data loads into the data warehouse database. By creating a clone of the replicated data at the disaster recovery site or at a production database, you can provide a staging area that can be refreshed quickly and can be used for cleansing and transformation of the data.
- **Standby database.** Immediately resume the read-write workload on discovering corruption in the production dataset by mounting the clone instead. You must use the database feature DB2 write-suspend mode to transparently prepare the database volumes for cloning by delaying write activity to the database. This is necessary because databases need to maintain a point of consistency.
- **Application testing.** Enterprise production data is growing exponentially, and application testing is not performed on near-production data because of the size of the data and maintenance overhead. Therefore, some real-life system problems go undetected during the testing phase of the system. By providing a cloned database, you can deliver near-production data for application testing and avoid production issues.
- **Online database backup.** The clone database is a frozen image of the database file system at the time of the clone creation. If necessary, the primary database can be restored from the Snapshot copy created for the clone, or the applications can point directly to the clone database.

2 REQUIREMENTS AND ASSUMPTIONS

2.1 ENVIRONMENTAL ASSUMPTIONS

This technical report covers cloning of a DB2 Enterprise Server Edition V9.7 database running under UNIX with database storage on a NetApp FAS or IBM N series storage system. The sample scripts and steps in this technical report assume the following:

- The name of the storage system containing the production database is `srcstore`.
- The name of the storage system used as the SnapMirror destination is `dststore`.
- The database host system used to access the production database is `hostsrc`.
- The database host system used to access the clone database is `hostdst`.

- The name of the source database is `mydb`.
- The name of the cloned database is `mydbcl`.
- The name of the aggregate on the storage systems is `dbaggr`.
- The name of the FlexVol volume used to store the production database table data is `dbdata`.
- The name of the FlexVol volume used to store the production database transaction logs is `dblogs`.
- The name of the clone volume that is created from the FlexVol volume named `dbdata` is `dbdata_cl`.
- The name of the clone volume that is created from the FlexVol volume named `dblogs` is `dblogs_cl`.
- The mountpoints used for the production database are `/mnt/dbdata` and `/mnt/dblogs`.
- The mountpoints used to mount the volumes used for the clone database are `/mnt/dbdata_cl` and `/mnt/dblogs_cl`.

The scripts contained in this document may require significant modifications to run under your version of UNIX.

2.2 SECURITY AND ACCESS ISSUES

Make sure that each FlexVol volume used for the DB2 database's data and transaction logs has its security style set to UNIX. Update the security style by executing the following command on the storage system:

```
qtree security [FlexVolPath] unix
```

Where

- `FlexVolPath` identifies the flexible volume path on the storage system that is used for the database.

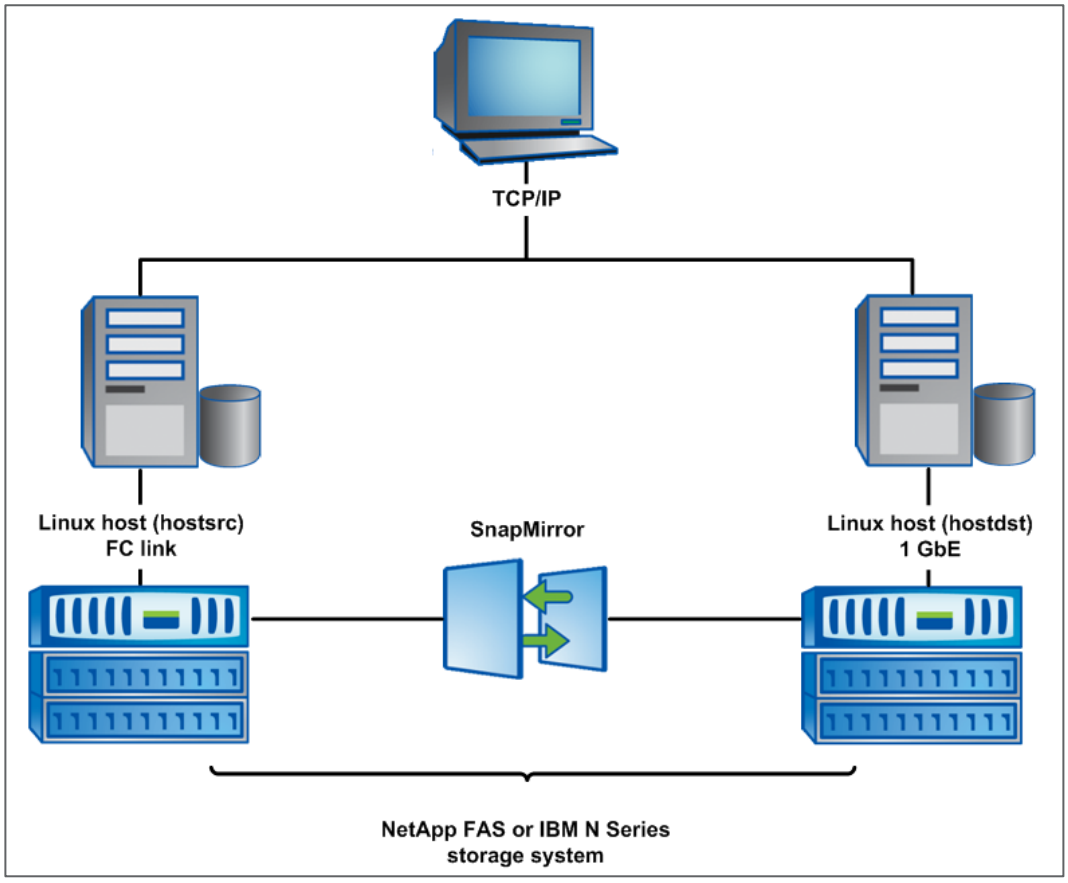
For example, to update the security style of a FlexVol volume named `dbdata`, execute the following command on the storage system:

```
qtree security /vol/db2data unix
```

2.3 NETWORK AND STORAGE INFRASTRUCTURE

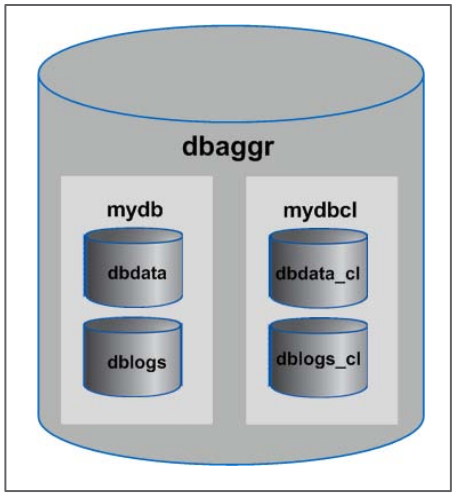
Figure 1 illustrates a simple basic architecture used to create a clone of a DB2 database in the UNIX, NetApp FAS, or IBM N series storage system environments.

Figure 1) DB2 database cloning using NetApp FlexClone technology.



A clone volume of a FlexVol volume, also known as a FlexClone volume, resides in the same aggregate as the parent FlexVol volume. Figure 2 illustrates that both the parent volumes and the clone volumes reside in the aggregate named `dbaggr`.

Figure 2) Both the source and the clone volumes reside in the same aggregate.



The FlexClone feature, combined with SnapMirror, enables database and storage administrators to create a clone on another aggregate or on another storage system. Figure 3 illustrates the use of

SnapMirror to replicate the data from the source volume to the destination volume, creating a clone on the destination storage system.

Figure 3) Clone database volumes on the second NetApp FAS or IBM N series storage system.

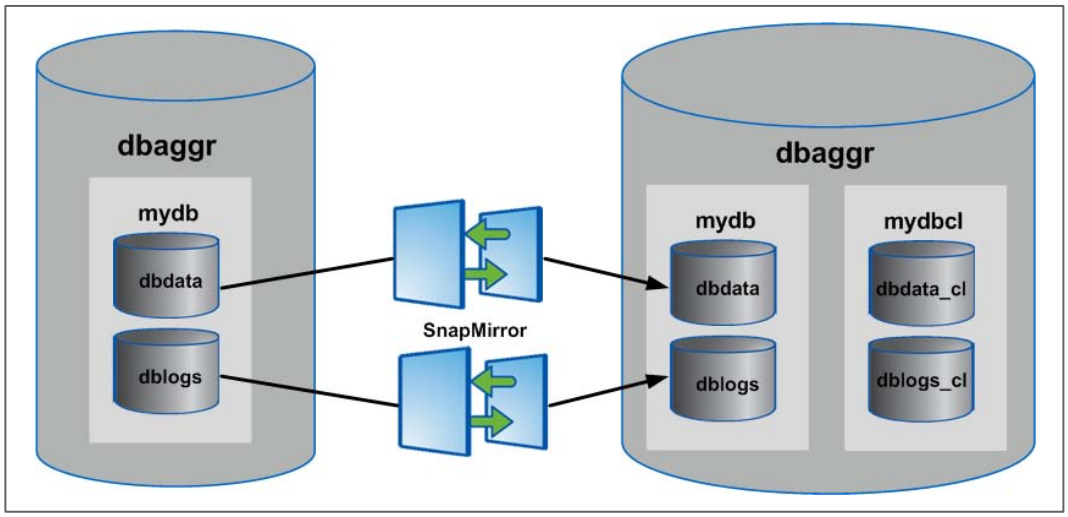
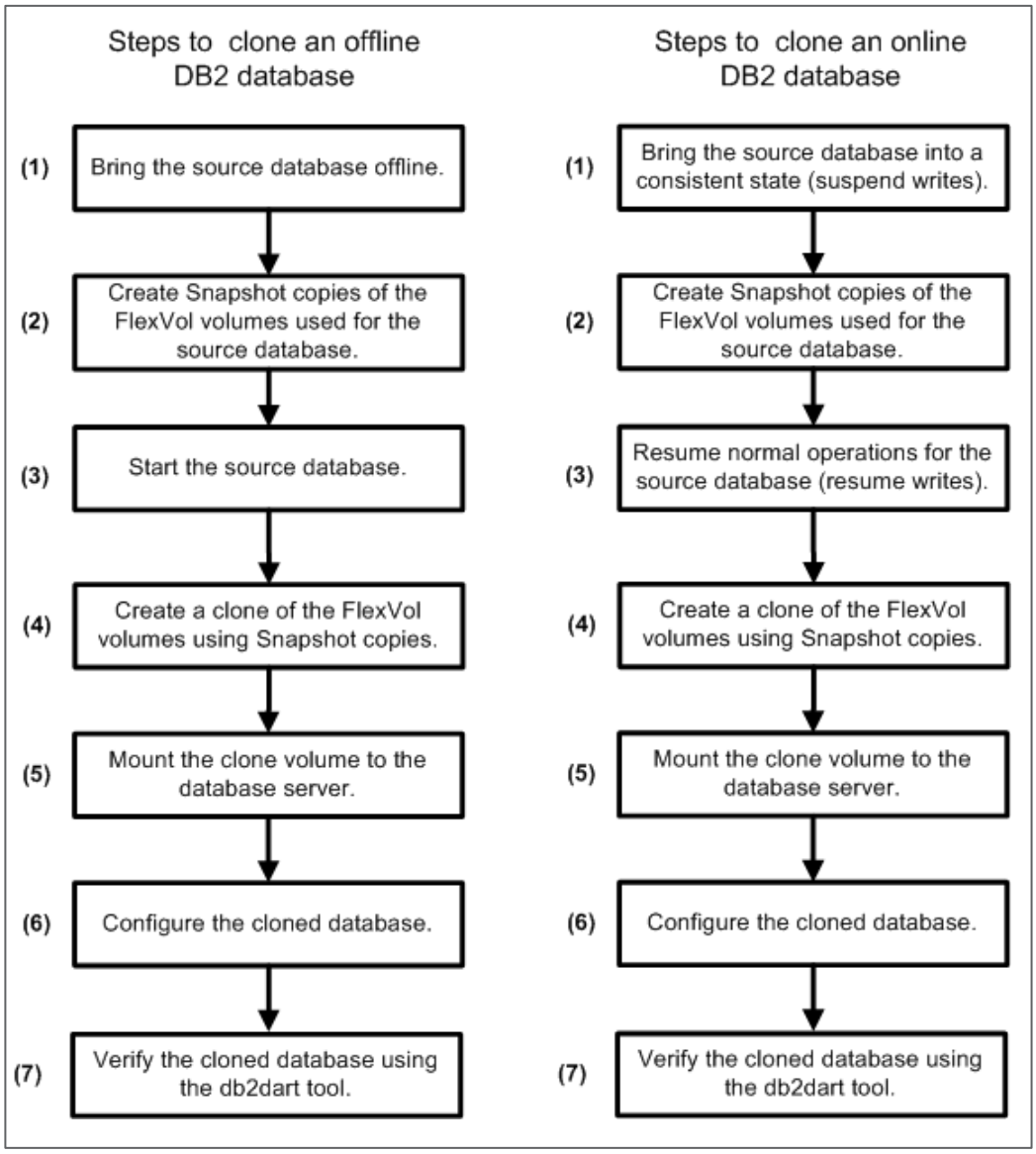


Figure 4 illustrates the steps necessary to create a clone of a database in a NetApp or IBM N series storage system environment.

Figure 4) Steps necessary to clone a BD2 database in a NetApp FAS or IBM N series storage system environment.



3 CREATING A DB2 DATABASE ON A NETAPP OR IBM N SERIES STORAGE SYSTEM

To create a DB2 database on a NetApp or IBM N series storage system, you must perform a few configuration steps on the database server and the NetApp or IBM N series storage system based on best practices. To set up the database with NetApp or IBM N series as the underlying storage system, refer to TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices. While configuring the database server and the storage system, create the following storage containers and objects on the storage system used and follow these naming conventions that conform to the scripts provided in the Appendix:

- An aggregate named `dbaggr1`
- Flexible volumes named `dbdata` and `dblogs` within aggregate `dbaggr`

For SAN environments, perform the following additional steps:

- Create a LUN named `/vol/dbdata/data` within FlexVol volume `dbdata`.
- Create a LUN named `/vol/dblogs/logs` within FlexVol volume `dblogs`.
- Create an igroup named `dbhost1_fcp_igp` for the database server `dbhost1`.
- On the storage system named `dbstore1`, create mappings for the LUNs named `/vol/dbdata/data` and `/vol/dbdata/logs` to the igroup named `dbhost1_fcp_igp`, using ID 0 and 1, respectively.

After you complete the initial steps, as mentioned in TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices, the production database is ready to create database objects.

4 CLONING A DB2 DATABASE IN THE NAS ENVIRONMENT

This section covers all the mechanics used in cloning a database. The Snap Creator framework fully automates the steps described in this section. Alternatively, you can use these steps to create custom automation scripts.

4.1 SELECTING A DATABASE SERVER TO ACCESS THE CLONED DATABASE

Select the database server that will be used to access the cloned database. You have the following two choices:

- The database server that is used to access the production (parent) database
- A database server other than the production database server

DATABASE SERVER USED TO ACCESS PRODUCTION (PARENT) DATABASE

In this case, you can use an existing DB2 instance or create a new one using the same DB2 V9.7 code as the production instance. To create a new DB2 instance, complete the following steps:

1. Switch the authority to the user root and create a user named `db2instc` on the database server by executing the following command:

```
useradd –c "DB2 clone db instance owner" -u 710 –g db2adm -G db2adm db2instc
-p db2instc
```

The new user owns the DB2 instance used to access the cloned database.

2. Create a new DB2 instance using the same DB2 code as the production database instance by executing the following command on the database server:

```
[DB2InstallationPath]/instance/db2icrt –u [FencedUser] [InstanceName]
```

Where

- `DB2InstallationPath` identifies the directory where the DB2 V9.7 code is installed.
- `FencedUser` identifies the ID of the user under which fenced user-defined functions and fenced stored procedures run.
- `InstanceName` identifies the name assigned to the new instance.

For example, if DB2 V9.7 is installed in the `/opt/IBM/db2/V9.7` directory, execute the following command on the database server to create a DB2 instance named `db2instc`:

```
/opt/IBM/db2/V9.7/instance/db2icrt –u db2instc db2instc
```

3. Check the list of instances and the DB2 code used by executing the following command on the database server:

```
/opt/IBM/db2/V9.7/instance/db2ilist –a
```

The output from this command should look similar to the following:

```
db2inst1   32   /opt/IBM/db2/V9.7
db2instc   32   /opt/IBM/db2/V9.7
```

DATABASE SERVER OTHER THAN PRODUCTION DATABASE SERVER

In this case, you must install the same DB2 code as the production database server has and create a database instance as described in the first option in this section. The new instance name can be the same as the production DB2 instance if no other instance has the same name on this server.

4.2 CREATING A DATABASE CLONE IN A NAS ENVIRONMENT

You can create a database clone while your database is online. If you want to clone an offline database by deactivating the database, then skip step 1, because it does the write suspend on the database.

1. Bring the database into a consistent state (suspend writes).

In this scenario, the source database is online. Therefore, to provide application consistency, temporarily suspend the database write operations by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to tablespace containers and log files that are associated with the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly; therefore, the database does not need to stay in write suspend mode for more than a few seconds.

2. Create Snapshot copies of the database FlexVol volumes.

Next, create a Snapshot copy of each FlexVol volume that is used for the production database. A Snapshot copy of a FlexVol volume can be created by executing the following command on the storage system:

```
snap create –V [VolName] [SnapName]
```

Where

- `VolName` identifies the name assigned to the FlexClone volume that is created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap.1` for a FlexVol volume named `dbdata`, execute the following command on the storage system:

```
snap create –V dbdata dbdata_cl_snp01
```

NetApp has the following two recommendations: Develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purpose.

3. Resume normal database operations (resume writes).

After the Snapshot copies are created, resume write operations to the database by executing the following command on the database server:

```
db2 set write resume for database
```

4. Clone the FlexVol volumes.

Next, create a clone of each FlexVol volume using the Snapshot copies created in step 2. Create a clone volume by executing the following command on the storage system:

```
vol clone create [CloneVol] –s [volume|file|none] –b [ParentVol] <ParentSnap>
```

Where

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume snapshot that is used as the source for the clone volume.

For example, to create a clone volume of a FlexVol volume named `dblogs` using the Snapshot copy named `dbdata_cl_snp01`, execute the following command on the NetApp storage system:

```
vol clone create dbdata_cl –s none –b dbdata dbdata_cl_snp01
```

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

Note: A Snapshot copy is not required to create a clone of a FlexVol volume. If you do not explicitly create a Snapshot copy and specify it when executing the **`vol clone`** command, a Snapshot copy is implicitly created and used for the clone volume. A Snapshot copy created implicitly has a system-assigned name. NetApp recommends explicitly creating a Snapshot copy and assigning a meaningful name to the copy before creating a clone FlexVol volume.

5. Create NFS export entries for the cloned volumes.

To mount a clone volume to the database server, create an export entry for it in the `/etc/exports` file that resides on the storage system. Create the export entry by executing the following command on the storage system:

```
exportfs –p rw=[HostName],root=[HostName] [PathName]
```

Where

- `HostName` identifies the database server name that uses these clone volumes as storage.
- `PathName` identifies the clone volume path.

For example, to create an export entry for a clone volume named `dbdata_cl` and to allow root access to it from a database server named `dbhost1`, execute the following command on the storage system:

```
exportfs -p rw=dbhost1,root=dbhost1 /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that will be used for the database in the test environment.

6. Mount the cloned volumes.

To access the clone database, mount the clone volume to a database server. First, create a mountpoint for each clone volume and append a mount entry to the `/etc/fstab` file. The mount entry should specify the mount options, and it should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

Where

- `StorageSystemName` identifies the name assigned to the storage system used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.

- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on a storage system named `srcstore`, append the following entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where

- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has the mount entry specified in the `/etc/fstab` file, execute the following command on the second database server, named `hostdst`:

```
mount /mnt/dbdata_cl
```

The database servers that we used had Linux® operating systems. For a database server with another operating system, refer to TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown –R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mountpoint named `/mnt/dbdata_cl`, execute the following command on the second database server:

```
chown –R db2inst1:db2adm /mnt/dbdata_cl
```

7. Configure the cloned database.

We will use the clone volumes created in step 4 and mounted in step 6 of this section as the storage containers for the cloned database. Skip parts (a) and (b) of this step if the following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
- The mountpoints that are used to mount the clone volumes have the same name as the mountpoints that are used to mount the volumes of the production database.

   a. By default when a database is created, the tablespace containers for the default tablespaces (`syscatspace, tempspace1, and userspace1`) reside in the following directory:

      `[DBDir]/[InstanceName]/NODE000n`

      Where

- `DBDir` identifies the name assigned to the directory/device the database is created on.
- `InstanceName` identifies the name assigned to the DB2 instance the database belongs to.

For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace containers reside in the following directory:

```
/mnt/dbdata/db2inst1/NODE0000
```

The DB2 instance name must be unique for the database server. Therefore, create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, change the default tablespace container's path name by executing the following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n
[DBDir]/[NewInstanceName]/NODE000n
```

Where

- `DBDir` identifies the name assigned to the directory/device the database is created on.
- `OldInstanceName` identifies the name assigned to the DB2 instance the production database belongs to.
- `NewInstanceName` identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named `db2instc`, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

b. Change the database name and tablespace containers' header information using the **`db2relocatedb`** command as specified in section 1.5. To make it simple, NetApp recommends creating a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look something like this:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs,/mnt/dblogs_cl
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is `mydb` and its logs are on `/mnt/dblogs` and its data is on `/mnt/dbdata`. The clone database must be renamed to `mydbcl`; it has its data on `/mnt/dbdata_cl` and its logs on `/mnt/dblogs_cl`. The source database instance name is `db2inst1` and the clone database instance name is `db2instc`.

Save the configuration file as `/home/db2inst1/dbrelocate.cfg`, and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

c. Start the database manager instance that you created for the migration test environment by executing the following command on the database server:

```
db2start
```

The production database was online during the Snapshot creation process. Therefore, use the `db2inidb` utility to initialize the clone database as a snapshot by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

Where

- `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to initialize a clone database named `mydbcl`, execute the following command on the database server:

```
db2inidb mydbcl as snapshot
```

d. If the production database is running in archive logging mode, update the configuration parameter named Primary Archive Logging Method (`LOGARCHMETH1`) for the cloned database. Update this parameter by executing the following command on the database server:

```
db2 update db cfg for [DatabaseName] using LOGARCHMETH1
DISK:[ArchiveDir]
```

Where

- `DatabaseName` identifies the name assigned to the database whose logging method changes.
- `ArchiveDir` identifies the directory (location) where the archived transaction log files are stored.

For example, to update the primary archive log parameter and place the archive log files in a directory named `/mnt/dbarch_cl/`, execute the following command on the database server:

```
db2 update db cfg for mydbcl using LOGARCHMETH1 DISK:/mnt/dbarch_cl
```

8. Verify the database.

After performing the previous steps, check the entire database for architectural correctness by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

Where

- `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to test the database named `mydbcl`, execute the following command on the database server:

```
db2dart mydbcl /db
```

The `db2dart` utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the `<$HOME>/sqllib/db2dump/DART0000/>` directory. A summary is listed at the end of the report. Read the summary to find any errors.

If the source and the clone database need to be accessed from the same database server, complete the additional steps described in the Configure a UNIX Host to Access Both the Clone and the Source Databases in a NAS Environment appendix.

4.3 CLONING A DATABASE IN THE DR ENVIRONMENT FOR NAS

The FlexClone feature combined with SnapMirror makes it possible to clone a DB2 database on a remote NetApp FAS or IBM N series storage system over the network or on another aggregate of the same NetApp FAS or IBM N series storage system. This section describes the steps necessary to clone a database on the second storage system at a remote location.

1. Configure SnapMirror.

To configure SnapMirror, complete the following steps on the SnapMirror source and the SnapMirror destination NetApp FAS or IBM N series storage system:

a. Add licenses for SnapMirror and SnapMirror sync on the source and destination NetApp FAS or IBM N series storage system by executing the following command:

```
license add  [licenseCode]
```

Where

– `licenseCode` identifies the license key for the product or feature that is provided to you by the NetApp or IBM sales representative.

For example, to enable a SnapMirror license key, execute the following command on the storage system:

```
license add  1234ABCDE
```

b. Enable the SnapMirror feature by executing the following command on the SnapMirror source and destination storage systems:

```
snapmirror on
```

or

```
options snapmirror.enable on
```

c. On the SnapMirror source storage system, create a `snapmirror.allow` file that resides in the `/vol/vol0/etc` directory and append an entry for the SnapMirror destination storage system. The entries in the `snapmirror.allow` file should look similar to the following:

```
[DestinationStorageSystemName]
```

Where

– `DestinationStorageSystemName` identifies the name assigned to the SnapMirror destination storage system.

For example, to append an entry for the storage system named `dststore`, add the following entry to the `snapmirror.allow` file on the SnapMirror source storage system:

```
dststore
```

2. Initialize SnapMirror.

As mentioned previously, the initialization process transfers data, including all Snapshot copies, from the source volume to the destination volume for the first time; thereafter, only the changed blocks are transferred. To initialize the SnapMirror relationship, you must restrict the destination volume by executing the following command on the destination storage system:

```
vol restrict [VolumeName]
```

Where

- `VolumeName` identifies the name assigned to the volume that is being used as the SnapMirror destination.

For example, to restrict a SnapMirror destination volume named `dbdata`, execute the following command on the destination storage system:

```
vol restrict dbdata
```

After restricting the destination FlexVol volumes, initialize the SnapMirror relationship for each volume that is used for the database by executing the following command on the destination storage system:

```
snapmirror initialize -S
[SourceStorageSystem]:[VolumeName][DestinationStorageSystem]:[VolumeName]
```

Where

- `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.
- `VolumeName` identifies the name assigned to the volume that is the SnapMirror source/destination.
- `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

For example, to initialize the SnapMirror relationship that is specified in the configuration file for the volume named `dbdata`, execute the following command on the destination storage system:

```
snapmirror initialize -S srcstore:dbdata dststore:dbdata
```

The **`SnapMirror initialize`** command creates a Snapshot copy of the source volume and transfers data from the source to the destination volume. After the baseline data transfer is finished, the destination volume data is available in read-only mode.

The progress of SnapMirror can be checked by executing the following command on the storage system:

```
snapmirror status
```

The output from this command should look similar to the following:

```
snapmirror status
snapmirror is on.
Source           Destination      State          Lag      Status
srcstore:dbdata  dststore:dbdata  SnapMirrored   00:07:33 Idle
srcstore:dblogs  dststore:dblogs  Uninitialized  - Transferring(116 MB done)
```

3. Bring the source database into a consistent state (suspend writes).

The source database is online. Therefore, to prevent partial page writes while database cloning is in progress, temporarily suspend the write operations to the database by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to tablespace containers and log files that are associated with the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The cloning process is completed very quickly, so the database does not need to stay in write suspend mode for more than a few seconds.

Note: Skip the previous step that does the write suspend on the database if you would like to keep the database offline.

4. Create Snapshot copies of the FlexVol volumes.

Create a Snapshot copy of each FlexVol volume that is used for the source database by executing the following command on the source storage system:

```
snap create –V [VolName] [SnapName]
```

Where

- `VolName` identifies the name assigned to the FlexClone volume that is created.

- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_cl_snap01` for a FlexVol volume named `dbdata`, execute the following command on the storage system:

```
snap create -V dbdata dbdata_cl_snp01
```

NetApp recommends developing a naming convention and assigning a meaningful name to the Snapshot copies that are created for cloning purposes.

Note: Important—the Snapshot copies are created on the SnapMirror source storage system.

5. Update the SnapMirror destination.

Now update the SnapMirror destination volumes manually by executing the following command on the destination storage system:

```
snapmirror update < –S [SourceStorageSystem]:[VolumeName]>
[DestinationStorageSystem]:[VolumeName]
```

Where

- `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.
- `VolumeName` identifies the name assigned to the volume that is being used as the SnapMirror destination.
- `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

For example, to update SnapMirror for a volume named `dbdata`, execute the following command on the SnapMirror destination storage system named `dststore`:

```
snapmirror update –S srcstore:dbdata dststore:dbdata
```

Update the SnapMirror relationship for each volume that is used for the database.

When the **`update`** command is executed for an asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes available on the destination volume immediately; therefore, a manual update is not required.

Note: Snapshot copies are also created automatically for SnapMirror update purposes. NetApp does not recommend using these automatically created Snapshot copies to create a clone volume.

6. Resume normal database operations (resume writes).

After the Snapshot copies are created, resume write operations to the database by executing the following command on the database server:

```
db2 set write resume for database
```

7. Create clone volumes using Snapshot copies.

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the source storage system and replicated to the destination. A clone volume can be created from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] –s [volume|file|none] –b [ParentVol] <ParentSnap>
```

Where

- `CloneVol` identifies the name of the FlexClone volume that is being created.

- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named `dbdata_cl` from the Snapshot copy named `dbdata_cl_snp01` of the volume named `dbdata`, execute the following command on the destination storage system:

```
vol clone create dbdata_cl –s none –b dbdata dbdata_cl_snp01
```

Note: Important—the Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship fails.

8. Create the NFS export entries for the cloned volumes.

To mount a clone volume to the database server, create an export entry for it in the `/etc/exports` file that resides on the NetApp FAS or IBM N series storage system. Create the export entry by executing the following command on the NetApp FAS or IBM N series storage system:

```
exportfs –p rw=[HostName],root=[HostName] [PathName]
```

Where

- `HostName` identifies the name assigned to the database server.
- `PathName` identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named `dbdata_cl` and to allow the root access from the database server named `hostdst`, execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume used for the clone database.

9. Mount the cloned volumes.

To access the clone database, mount the clone volumes to a database server. First, create a mountpoint for each clone volume and append a mount entry to the `/etc/fstab` file. The mount entry should specify the mount options and should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

Where

- `StorageSystemName` identifies the name assigned to the storage system used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.
- `MountPoint` identifies the name assigned to the mount location used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on a storage system named `srcstore`, append the following entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where

- `MountPoint` identifies the name assigned to the mount location used to mount the flexible volume on the database server.

For example, to mount a clone volume that has a mount entry specified in the `/etc/fstab` file, execute the following command on the second database server named `hostdst`:

```
mount /mnt/dbdata_cl
```

The database servers that we used had Linux operating systems. For a database server with another operating system, refer to TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Change ownership by executing the following command on the database server:

```
chown –R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mountpoint named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

```
chown –R db2inst1:db2adm /mnt/dbdata_cl
```

10. Configure the cloned database.

The clone volumes created in step (6) and mounted in step (8) of this section are used as the storage containers for the cloned database. Skip parts (a) and (b) of this step if the following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
- The mountpoints used to mount the clone volumes have the same name as the mountpoints used to mount volumes of the production database.

  a. By default, when a database is created, the tablespace containers for the default tablespaces (`syscatspace`, `tempspace1`, and `userspace1`) reside in the following directory:

  ```
  [DBDir]/[InstanceName]/NODE000n
  ```

  Where

  – `DBDir` identifies the name assigned to the directory/device the database is created on.

  – `InstanceName` identifies the name assigned to the DB2 instance the database belongs to.

  For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace container resides in the following directory:

  ```
  /mnt/dbdata/db2inst1/NODE0000
  ```

  The DB2 instance name must be unique for a database server. Therefore, create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, change the default tablespace container's path name by executing the following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n
```

Where

- `DBDir` identifies the name assigned to the directory/device the database is created on.
- `OldInstanceName` identifies the name assigned to the DB2 instance the production database belongs to.
- `NewInstanceName` identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named `db2instc`, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

b. Next, change the database name and tablespace containers' header information using the **`db2relocatedb`** command as specified in section 1.5. To simplify this process, NetApp recommends creating a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look something like this:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs,/mnt/dblogs_cl
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is `mydb`, its logs are located on `/mnt/dblogs`, and its data is located on `/mnt/dbdata.` The new name for the clone database is `mydbcl`, it has its data on `/mnt/dbdata_cl`, and its logs are on `/mnt/dblogs_cl`. The source database instance name is `db2inst1` and the clone database instance name is `db2instc`.

Save the configuration file as `/home/db2inst1/dbrelocate.cfg` and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

c. Start the database manager instance by executing the following command on the database server:

```
db2start
```

If you are not required to run the **`db2relocatedb`** command, catalog the database manually by executing the following command:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on
[FileSystem]"
```

Where

- `DatabaseName` identifies the name assigned to the database that is being cataloged.
- `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.
- `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named `mydb` that resides on the file system named `/mnt/dbdata`, execute the following command on the database server:

```
db2 “catalog database mydb as mydb on /mnt/dbdata”
```

After completing these steps, you should be able to access both the source and the cloned databases from the same database server.

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing the **db2start** command.

The production database was online when the Snapshot copies were created. Therefore, use the db2inidb utility to initialize the clone database as a Snapshot copy by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

Where

- DatabaseName identifies the name of the clone database that is used for the test environment.

For example, to initialize a clone database named mydbcl, execute the following command on the database server:

```
db2inidb mydbcl as snapshot
```

d. If the source database is running in archive logging mode, then update the configuration parameter named Primary Archive Logging Method (logarchmeth1) for the cloned database. Update this parameter by executing the following command on the database server:

```
db2 update db cfg for [DatabaseName] using LOGARCHMETH1 DISK:[ArchiveDir]
```

Where

- DatabaseName identifies the name assigned to the database whose logging method changes.
- ArchiveDir identifies the directory (location) where archived transaction log files are stored.

For example, to update the primary archive log parameter and place the archive log files in a directory named /mnt/dbarch_cl/, execute the following command on the database server:

```
db2 update db cfg for mydbcl using LOGARCHMETH1 DISK:/mnt/dbarch_cl
```

11. Verify the cloned database.

After performing these steps, check the entire database for architectural correctness by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

Where

- DatabaseName identifies the name of the clone database used for the test environment.

For example, to test the database named mydbcl, execute the following command on the database server:

```
db2dart mydbcl /db
```

The db2dart utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <$HOME>/sqllib/db2dump/DART0000/> directory. A summary is listed at the end of the report. Read the summary to find any errors.

If the source and the clone database need to be accessed from the same database server, complete the additional steps described in the Configure a UNIX Host to Access Both the Clone and the Source Databases in a NAS Environment appendix.

5 CLONING A DB2 DATABASE IN THE SAN ENVIRONMENT

5.1 CREATING A DATABASE CLONE IN A SAN ENVIRONMENT

To create a clone on an online DB2 database on the same storage system in a SAN environment, complete the following steps.

1. Bring the source database into a consistent state (suspend writes).

In this scenario, the source database is online. Therefore, to prevent partial page writes while the database cloning is in progress, temporarily suspend the write operations to the database by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly, so the database does not need to stay in write suspend mode for more than a few seconds.

Note: Skip the previous step, which does the write suspend on the database, if you want to keep the database offline.

2. Create Snapshot copies of the FlexVol volumes.

The logical unit numbers (LUNs) reside within a FlexVol volume. Therefore, create a Snapshot copy for each FlexVol volume that consists of LUNs used for the production database. Create a Snapshot copy of a FlexVol volume by executing the following command on the storage system:

```
snap create –V [VolName] [SnapName]
```

Where

- `VolName` identifies the name assigned to the FlexClone volume that is created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap01` for a FlexVol volume named `dbdata`, execute the following command on the storage system:

```
snap create –V dbdata dbdata_cl_snp01
```

NetApp recommends developing a naming convention and assigning a meaningful name to the Snapshot copies created for cloning purposes.

3. Resume normal database operation (resume writes).

After the Snapshot copies are created, resume the write operations to the database by executing the following command on the database server:

```
db2 set write resume for database
```

4. Clone the FlexVol volumes using the Snapshot copies.

Next, create a clone of each FlexVol volume by using the Snapshot copies created in step 2. Create a clone volume by executing the following command on the storage system:

```
vol clone create [CloneVol] –s [volume|file|none] –b [ParentVol] <ParentSnap>
```

Where

- `CloneVol` identifies the name of the FlexClone volume that is being created.

- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone of a FlexVol volume named `dblogs` by using its existing Snapshot copy named `dbdata_cl_snp01`, execute the following command on the storage system:

```
vol clone create dbdata_cl –s none –b dbdata dbdata_cl_snp01
```

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

5. Create new mapping for the LUNs.

As previously mentioned, a clone volume is a writable Snapshot copy. At the time of the clone creation, it has data that is exactly similar to that of the source volume. The LUNs on the clone volume have the same mapping as the LUNs on the source volume, and they are set offline. To create new mapping for LUNs on the FlexClone volumes, complete the following steps:

a. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

```
lun unmap [LunPath]  [iGroupName]
```

Where

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

For example, to remove old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

```
lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
```

b. Create a new mapping for the LUNs on the clone volume by using a new ID and/or by mapping them to a new igroup. Create a LUN mapping by executing the following command on the storage system:

```
lun unmap [LunPath]  [iGroupName] [LunId]
```

Where

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.
- `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

```
lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
```

c. After remapping, each LUN that is on the clone volume and that is used for the clone database needs to be brought online by executing the following command on the storage system:

```
lun online [LunPath]
```

Where

- `LunPath` identifies the name assigned to the new clone volume.

For example, to bring a LUN named `/vol/dbdata_cl/data` online, execute the following command on the storage system:

```
lun online /vol/dbdata_cl/data
```

6. Mount the LUNs that reside on the FlexClone volumes.

   To mount the LUNs that reside on the FlexClone volumes, complete the following steps:

   a. Refresh the HBA driver on the database server. For example, to refresh a QLogic FC HBA on a Linux database host, execute the following commands on the database server:

```
modprobe –r qla2300
modprobe –v qla2300
```

      For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

   b. Obtain the LUN device names by executing the following command on the database server used to access the clone database:

```
sanlun lun show
```

Note: If your database host is running the AIX operating system, you must perform a few extra steps before you can mount the LUNs on the cloned volume. The extra steps necessary to mount LUNs for the cloned database are described in the Configure Cloned LUNs for an AIX Host in SAN Environment appendix.

   c. Mount the LUN devices by executing the following command on the database server:

   `mount [DeviceName] [MountPoint]`

   Where

   – `DeviceName` identifies the name assigned to the new clone volume.
   – `MountPoint` identifies the name assigned to the mountpoint that is used to mount the LUN device.

   For example, to mount a LUN device that is identified by the name `/dev/sdb` by the database server, execute the following command on the database server:

```
mount /dev/sdb /mnt/dbdata
```

   d. To operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Ownership can be changed by executing the following command on the database server:

   `chown –R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]`

   Where

   – `InstanceOwner` identifies the name assigned to the user who owns the database instance.
   – `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
   – `FileSystem` identifies the name of the file system whose ownership is changed.

   For example, to change ownership of the file system mounted on the mountpoint named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

```
chown –R db2inst1:db2adm /mnt/dbdata_cl
```

7. Configure the cloned database.

Use the clone volumes created in step 6 and mounted in step 8 of this section as the storage containers for the cloned database. Skip steps (a) and (b) if the following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
- The mountpoints that are used to mount the clone volumes have the same name as the mountpoints that are used to mount the volumes of the production database.

a. By default, when the database is created, the tablespace containers for the default tablespaces (`syscatspace`, `tempspace1`, and `userspace1`) reside in the following directory:

`[DBDir]/[InstanceName]/NODE000n`

Where

- `DBDir` identifies the name assigned to the directory/device the database is created on.
- `InstanceName` identifies the name assigned to the DB2 instance the database belongs to.

For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace container resides in the following directory:

`/mnt/dbdata/db2inst1/NODE0000`

The DB2 instance name must be unique for a database server. Therefore, create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, change the default tablespace container's path name by executing the following command on the database server:

`mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n`

Where

- `DBDir` identifies the name assigned to the directory/device the database is created on.
- `OldInstanceName` identifies the name assigned to the DB2 instance the production database belongs to.
- `NewInstanceName` identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named `db2instc`, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

b. Next, change the database name and tablespace containers' header information using the **`db2relocatedb`** command as specified in section 1.5. Create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to the following:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs,/mnt/dblogs_cl
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is `mydb`, its logs are on `/mnt/dblogs`, and its data is on `/mnt/dbdata`. The new name for the clone database is `mydbcl`, its data is on `/mnt/dbdata_cl`, and its logs are on `/mnt/dblogs_cl`. The source database instance name is `db2inst1` and the clone database instance name is `db2instc`.

Save the configuration file as `/home/db2inst1/dbrelocate.cfg` and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

c. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing the **`db2start`** command.

The production database was online when the Snapshot copies were created. Therefore, use the `db2inidb` utility to initialize the clone database as a Snapshot copy by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

Where

- `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to initialize a clone database named `mydbcl`, execute the following command on the database server:

```
db2inidb mydbcl as snapshot
```

d. If the source database is running in the archive logging mode, update the configuration parameter named Primary Archive Logging Method (`logarchmeth1`) for the cloned database. Update this parameter by executing the following command on the database server:

```
db2 update db cfg for [DatabaseName] using LOGARCHMETH1 DISK:[ArchiveDir]
```

Where

- `DatabaseName` identifies the name assigned to the database whose logging method changes.
- `ArchiveDir` identifies the directory (location) where archived transaction log files are stored.

For example, to update the primary archive log parameter and to place the archive log files in a directory named `/mnt/dbarch_cl/`, execute the following command on the database server:

```
db2 update db cfg for mydbcl using LOGARCHMETH1 DISK:/mnt/dbarch_cl
```

After you perform these steps, the clone database is ready for use. You can connect to the database and perform data verification.

If the source and the clone database need to be accessed from the same database host, complete the additional steps described in Configure a UNIX Host to Access Both the Clone and the Source Databases in a SAN Environment appendix.

In this scenario, the LUNs on the clone volume are accessed using FCP, but they can also be accessed by using the iSCSI access protocol.

5.2 CLONING A DATABASE IN A DR ENVIRONMENT FOR SAN

1. Set up and initialize SnapMirror.

The necessary steps to set up and initialize a SnapMirror relationship are described in parts 1 and 2 of section 5.3. After completing these steps, you should have working SnapMirror relationships for the volumes that are used for the cloned database.

In the SnapMirror relationship, the destination volumes are read only. Therefore, to create a clone of the destination volume, you need a Snapshot copy that is created on the SnapMirror source and that is replicated to the destination volume. Obtain this Snapshot copy by completing the following steps.

2. Bring the database into a consistent state (suspend writes).

In this scenario, the source database is online. Therefore, to prevent partial page writes while the database Snapshot copy process is in progress, temporarily suspend the write operations to the database by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly, so the database does not need to stay in write-suspend mode for more than a few seconds

Note: Skip this step, which does the write suspend on the database, if you want to keep the database offline.

3. Create Snapshot copies of the volumes that are used for the database.

The LUNs reside within a FlexVol volume. Therefore, you must create a Snapshot copy for each FlexVol volume that consists of LUNs that are used for the production database. Create a Snapshot copy of a FlexVol volume by executing the following command on the storage system:

```
snap create –V [VolName] [SnapName]
```

Where

- `VolName` identifies the name assigned to the FlexClone volume that is created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap01` for a FlexVol volume named `dbdata`, execute the following command on the storage system:

```
snap create –V dbdata dbdata_cl_snp01
```

NetApp recommends developing a naming convention and assigning a meaningful name to the Snapshot copies that are created for cloning purposes.

4. Resume normal operations for the database (resume writes).

After the Snapshot copies are created, resume write operations to the database by executing the following command on the database server:

```
db2 set write resume for database
```

5. Update the SnapMirror destination volumes.

Update the SnapMirror destination volumes manually by executing the following command on the destination storage system:

```
snapmirror update < –S [SourceStorageSystem]:[VolumeName]>
[DestinationStorageSystem]:[VolumeName]
```

Where

- `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.
- `VolumeName` identifies the name assigned to the volume that is being used as the SnapMirror destination.

- `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

For example, to update SnapMirror for a volume named `dbdata`, execute the following command on the SnapMirror destination storage system named `dststore`:

```
snapmirror update –S srcstore:dbdata dststore:dbdata
```

Update the SnapMirror relationship for each volume that is used for the database.

For asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including the Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes visible on the destination volume immediately; therefore, a manual update is not required.

Note: Important—the Snapshot copies are also created automatically for SnapMirror update purposes. NetApp does not recommend using these automatically created Snapshot copies to create a clone volume.

6. Create clone volumes using Snapshot copies.

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the SnapMirror source storage system and replicated to the destination. Create a clone volume from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] –s [volume|file|none] –b [ParentVol] <ParentSnap>
```

Where

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named `dbdata_cl` from the Snapshot copy named `dbdata_cl_snp01` of the volume named `dbdata`, execute the following command on the destination storage system:

```
vol clone create dbdata_cl –s none –b dbdata dbdata_cl_snp01
```

Note: Important—the Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship fails.

7. Create new mapping for LUNs that reside on the clone volumes.

When a clone volume is created, by default the LUNs residing on it have the same mapping as the source FlexVol volume LUNs, and they are placed offline. View the LUN status by executing the following command on the storage system:

```
lun show
```

The output from the **`lun show`** command should look similar to the following:

```
lun show
/vol/dbdata/data    15g (16106127360)  (r/o, online, mapped)
/vol/dbdata_cl/data 15g (16106127360)  (r/w, offline, mapped)
```

List the LUN mappings by executing the following command on the storage system:

```
lun show –m
```

The output from the **`lun show –m command`** should look similar to the following:

```
lun show –m
LUN path                  Mapped to          LUN ID  Protocol
----------------------------------------------------------------
/vol/dbdata/data            host_src_fcp_igp      0     FCP
/vol/dbdata_cl/data         host_src_fcp_igp      0     FCP
```

The output shows that the LUNs on the clone volume have the same mapping as the parent.

The FlexClone volume LUNs can be accessed from the same database server that is used to access the source database, or from a completely different server. The scenarios described in this paper were produced using a second database server to access the clone database.

To access the clone database from a second database server, complete the following steps:

a. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

`lun unmap [LunPath]  [iGroupName]`

Where

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

For example, to remove the old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

```
lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
```

b. Create a new mapping for the LUNs on the clone volume by using a new ID and/or by mapping them to a new igroup. Create a LUN mapping by executing the following command on the storage system:

`lun unmap [LunPath]  [iGroupName] [LunId]`

Where

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.
- `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

```
lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
```

c. After remapping, each LUN that is on the clone volume and that is used for the clone database needs to be brought online by executing the following command on the storage system:

`lun online [LunPath]`

Where

- `LunPath` identifies the name assigned to the new clone volume.

For example, to bring a LUN named `/vol/dbdata_cl/data` online, execute the following command on the storage system:

```
lun online /vol/dbdata_cl/data
```

8. Mount the LUN devices.

To mount the LUNs that reside on the FlexClone volumes, complete the following steps:

a. Refresh the HBA driver on the database server. For example, to refresh a QLogic FC HBA on a Linux database host, execute the following commands on the database server:

```
modprobe –r qla2300
modprobe –v qla2300
```

For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

b. Obtain the device names for the LUNs by executing the following command on the database server used to access the clone database:

```
sanlun lun show
```

Note: If your database host is running an AIX operating system, you must perform a few extra steps before you can mount the LUNs on the cloned volume. The extra steps necessary to mount LUNs for the cloned database are described in the Configure Cloned LUNs for an AIX Host in SAN Environment.

c. Mount the LUN devices by executing the following command on the database server:

`mount [DeviceName] [MountPoint]`

Where

- `DeviceName` identifies the name assigned to the new clone volume.
- `MountPoint` identifies the name assigned to the mount location that is used to mount the LUN device.

For example, to mount a LUN device named `/dev/sdb` to a mount location named `/mnt/dbdata_cl`, execute the following command on the database server:

```
mount /dev/sdb /mnt/dbdata_cl
```

d. To operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Ownership can be changed by executing the following command on the database server:

`chown –R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]`

Where

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mountpoint named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

```
chown –R db2inst1:db2adm /mnt/dbdata_cl
```

9. Configure the cloned database.

The clone volumes created in steps 6 and 8 of this section are used as the storage containers for the cloned database. Skip parts (a) and (b) of this step if the following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is the same as the source database instance.
- The mountpoints that are used to mount the clone volumes have the same name as the mountpoints that are used to mount the volumes of the production database.

  a. By default, when the database is created, the tablespace containers for the default tablespaces (`syscatspace`, `tempspace1`, and `userspace1`) reside in the following directory:

  `[DBDir]/[InstanceName]/NODE000n`

  Where

  – `DBDir` identifies the name assigned to the directory/device the database is created on.

  – `InstanceName` identifies the name assigned to the DB2 instance the database belongs to.

  For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace container resides in the following directory:

  `/mnt/dbdata/db2inst1/NODE0000`

  The DB2 instance name must be unique for a database server. Therefore, you must create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, change the default tablespace container's path name by executing the following command on the database server:

  `mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n`

  Where

  – `DBDir` identifies the name assigned to the directory/device the database is created on.

  – `OldInstanceName` identifies the name assigned to the DB2 instance the production database belongs to.

  – `NewInstanceName` identifies the name assigned to the DB2 instance the clone database belongs to.

  For example, to access the clone database from the instance named `db2instc`, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

  b. Next, change the database name and tablespace containers' header information using the **`db2relocatedb`** command as specified in section 1.5. Create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file is as follows:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs,/mnt/dblogs_cl
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

  In this configuration file, the source database name is `mydb`, its logs are on `/mnt/dblogs`, and its data is on `/mnt/dbdata`. Rename the clone database to `mydbcl`; it has its data on `/mnt/dbdata_cl` and its logs on `/mnt/dblogs_cl`. The source database instance name is `db2inst1` and the clone database instance name is `db2instc`. Save the configuration file as `/home/db2inst1/dbrelocate.cfg` and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

c. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing the **db2start** command. The production database was online when the Snapshot copies were created. Therefore, use the **db2inidb** utility to initialize the clone database as a snapshot by executing the following command on the database server:

```
db2inidb [DatabaseName] as snapshot
```

Where

- `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to initialize a clone database named `mydbcl`, execute the following command on the database server:

```
db2inidb mydbcl as snapshot
```

d. If the source database is running in the archive logging mode, update the configuration parameter named Primary Archive Logging Method (`logarchmeth1`) for the cloned database. Update this parameter by executing the following command on the database server:

```
db2 update db cfg for [DBName] using LOGARCHMETH1 DISK:[ArchiveDir]
```

Where

- `DBName` identifies the name assigned to the database whose logging method changed.
- `ArchiveDir` identifies the directory (location) where the archived transaction log files are stored.

For example, to update the primary archive log parameter and to place the archive log files in a directory named `/mnt/dbarch_cl/`, execute the following command on the database server:

```
db2 update db cfg for mydbcl using LOGARCHMETH1 DISK:/mnt/dbarch_cl
```

After completing these steps, the cloned database is ready for use. If the source and the clone database need to be accessed from the same database host, then you must complete the additional steps described in the Configure a UNIX Host to Access Both the Clone and the Source Databases in a SAN Environment appendix. In this scenario, the LUNs on the clone volume are accessed using FCP, but they can be accessed using the iSCSI access protocol as well.

6 CLONING A DB2 DATABASE USING NETAPP SNAP CREATOR

The purpose of NetApp Snap Creator is to create a central framework that provides seamless integration with the DB2 application and NetApp storage Snapshot technology. Normally, using NetApp Snapshot technology to create Snapshot copies, using SnapRestore®, or creating a FlexClone volume requires a hard-coded custom script that interfaces with the application and the NetApp storage. These custom scripts are written every day, over and over again, and are normally not reused, so the purpose of Snap Creator is to save time and to provide the most reliable solution possible. The application piece is usually unique. Snap Creator does not contain code to specifically handle application consistency. Instead, it has a framework in which you can integrate the DB2 module. This module handles the application consistency in Snap Creator. Snap Creator communicates with NetApp storage and performs various tasks, which include policy-based Snapshot management and integration with other NetApp products. The installation and configuration of Snap Creator is out of the scope of this document. Refer to TR-3841: Snap Creator 3.2 Installation and Administration Guide and TR-3114: IBM DB2 for UNIX: Backup and Recovery Using NetApp Technologies for a detailed discussion of Snap Creator.

6.1 CONFIGURING SNAP CREATOR TO CLONE A DB2 DATABASE

A Snap Creator configuration file must be created to back up, recover, or clone the DB2 database. The Snap Creator configuration file has different sections, and the Configure Snap Creator to Clone a DB2 Database appendix shows the minimum required fields for cloning a DB2 database.

6.2 CREATING A CLONE OF THE DB2 DATABASE USING SNAP CREATOR

To create a clone of the DB2 database using Snap Creator, complete the following steps:

1. Clone a DB2 database using an existing Snapshot copy.

In this case, we are using an existing snapshot created by the Snap Creator backup operation. This situation occurs when you are using a Snapshot copy during a nightly backup that was taken with the DB2 write-suspend operation and you don't want to take another Snapshot copy with the write-suspend option again. After the Snap Creator configuration file is created and configured to clone the DB2 database, execute Snap Creator as follows:

```
snapcreator --profile <profilename> --config <config name> --action
<SnapCreator action> --policy <defined policy> --verbose --user_defined
<snapshot name>
```

Where

- `profile` identifies the profile name of your database. NetApp recommends using `database` as the profile name.
- `config` identifies the name of the specific configuration file for your database. Snap Creator allows you to maintain more than one configuration file for your database. So you could create multiple configuration files for your database based on your needs.
- `action` identifies the Snap Creator action. We are cloning a database; therefore, the action would be `clone_vol`.
- `policy` identifies the policy defined in the Snap Creator configuration file that we use daily for this example.
- `verbose` displays information on the console while Snap Creator gets executed. You can skip this option and the output still gets logged in the Snap Creator logs.
- `user_defined` identifies the snapshot name that got created by a previous Snap Creator backup operation.

For example, to create a clone of the database with the name of `mydb` and the configuration file `mydb_clone` and by using an existing snapshot, execute Snap Creator as follows:

```
./snapcreator --profile mydb --config mydb_clone --action clone_vol --policy
daily --verbose --user_defined 20110110130535
```

2. Clone a DB2 database using a new Snapshot copy.

In this scenario, Snap Creator creates a new Snapshot copy before the actual cloning process. The Snapshot copies are created by keeping the DB2 database in a write-suspend mode, which is done by the DB2 module in Snap Creator. After the Snapshot creation is complete, the database is put back into a normal mode by the DB2 module. To clone the DB2 database using a new Snapshot copy, execute Snap Creator as follows:

```
snapcreator --profile <profilename> --config <config name> --action
<SnapCreator action> --policy <defined policy> --verbose
```

Where

- `profile` identifies the profile name of your database. NetApp recommends using `database` as the profile name.
- `config` identifies the name of the specific configuration file for your database. Snap Creator allows you to maintain more than one configuration file for your database. So you can create multiple configuration files for your database based on your needs.
- `action` identifies the Snap Creator action. We are cloning a database; therefore, the action would be `clone_vol`.
- `policy` identifies the policy defined in the Snap Creator configuration file that we use daily for this example.
- `verbose` displays information on the console while Snap Creator gets executed. You can skip this option and the output still gets logged in the Snap Creator logs.

For example, to create a clone of the database name `mydb,` with the configuration file `mydb_clone`, and with using an existing Snapshot copy, execute Snap Creator as follows:

```
./snapcreator --profile mydb --config mydb_clone --action clone_vol --policy
daily --verbose
```

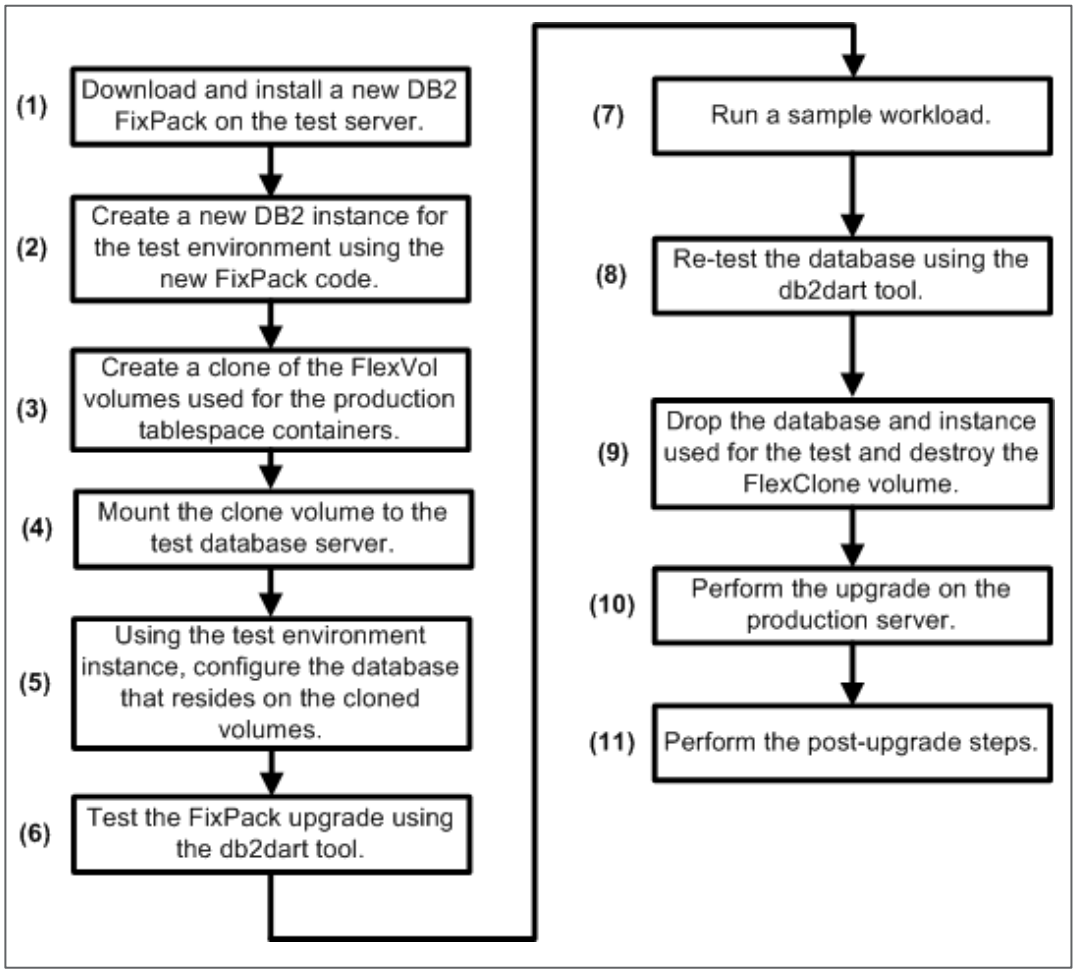
After Snap Creator creates the clone of your database, you can complete the rest of the steps, which include mounting the new clone volumes on the database server, configuring the database, and verifying the database. These steps can be automated through the Snap Creator POST command steps by using a custom script.

7 USE CASES

FlexClone is a very powerful feature of Data ONTAP and has many use cases as described in section 1.6. This section covers only a few of the most common use cases.

7.1 CLONING A DATABASE TO TEST A NEW DB2 FIXPACK

Figure 5) DB2 FixPack test process using NetApp or IBM N series FlexClone.



To test a DB2 FixPack upgrade, create a test environment that is similar to a production environment and that contains production-like data. FlexClone technology allows you to create a copy of the production database and mount it on a test server that has the latest DB2 FixPack installed. Depending on your environment, you can create a DB2 database clone as described in section 5 or section 6.

Use the cloned production database to test your sample workload. After the test is successful, you can destroy the cloned volumes. Figure 5 lists the steps involved in creating an environment to test the DB2 FixPack upgrade.

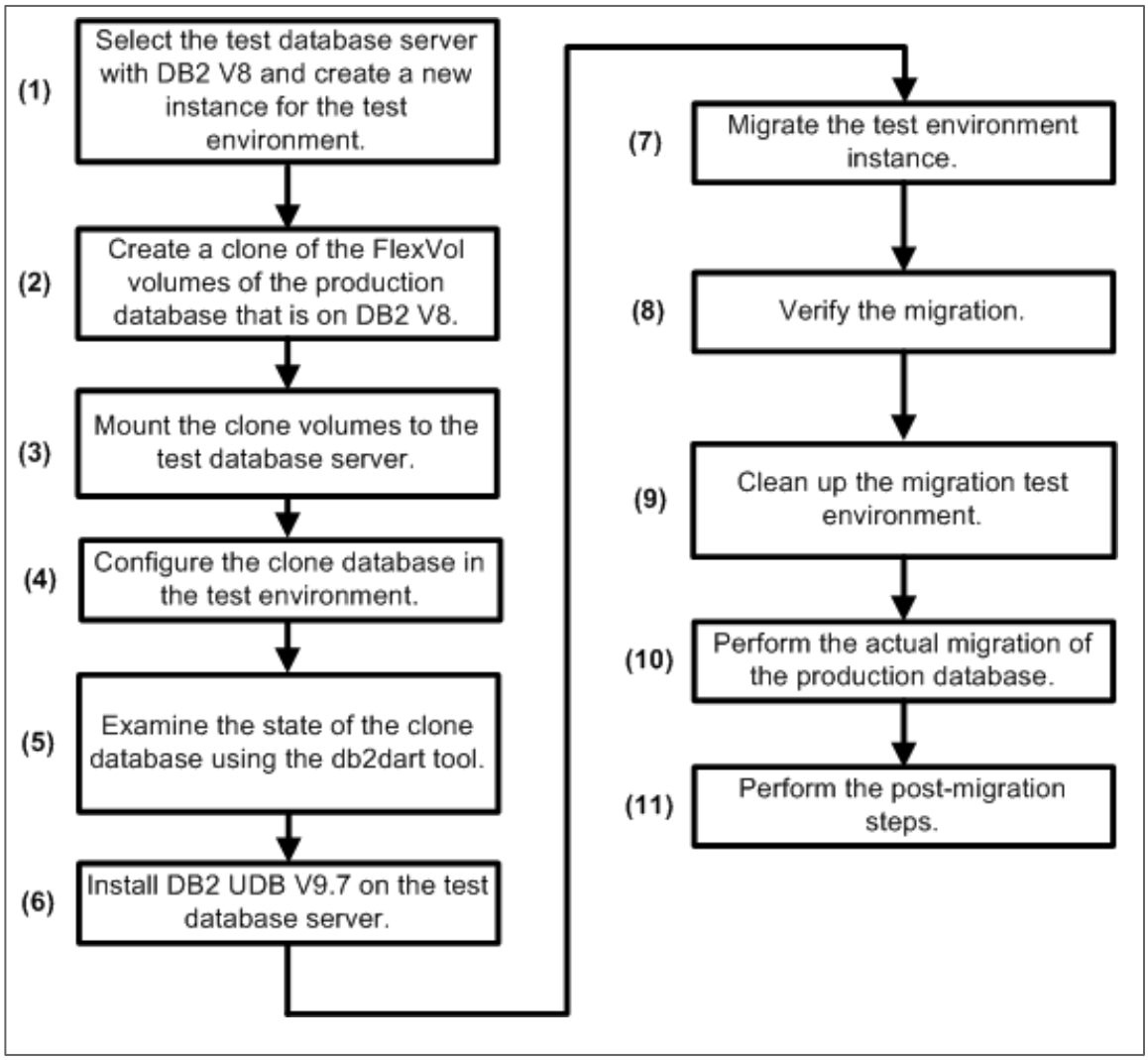
7.2 CLONING A DATABASE TO TEST MIGRATION FROM A PRIOR DB2 RELEASE

The database migration process must be conducted in a test environment before it is performed in a production environment. The migration of DB2 components is out of the scope of this document. For details on the migration of these components, refer to the IBM DB2 Migration Guide.

To test a database migration to DB2 9.7 from a previous version of DB2 8 (for this test, we used a DB2 database server with DB2 V8 installed), create a test environment using production or near-production data. All migration issues must be addressed before starting the actual migration of the production database. Creating a test environment using the traditional method of backup/restore is very time

consuming and requires additional storage space. Alternatively, by using FlexClone technology, you can create the clone of the production database, which is a DB2 V8, and mount it in a test environment, which is on DB2 V9.7, for migration testing. Figure 6 shows the steps that are involved in creating a test environment for the migration by using NetApp FlexClone technology.

Figure 6) DB2 V9.7 migration test process using NetApp or IBM N series FlexClone.



7.3 CLONING AN HADR STANDBY DATABASE TO CREATE A READ/WRITE COPY

The DB2 High Availability Disaster Recovery (HADR) feature provides immense data protection from both partial and full site failures. In case of failure, the standby database can be made available to clients across the network until the primary database is repaired and returned to service. Recovery can include recovery from corruption, natural disaster at the source site, accidental deletion, and sabotage. In the event of a disaster, all application traffic is rerouted to the standby database at the disaster recovery site for as long as necessary to recover the primary site. After the primary database is repaired, it can rejoin as a standby and perform catch-up with the new primary database. The role of the new standby database can be switched back to the primary database by a takeover operation.

FlexClone technology can be used in conjunction with the DB2 HADR to create a read/write copy of the standby database when the standby database is hosted on a NetApp underlying storage system. The

HADR relationship between the primary and the standby database remains intact, and the clone of the standby database is still available for read and writes. To clone an HADR standby database, complete the following steps.

1. Make sure that the standby database is in a consistent state.

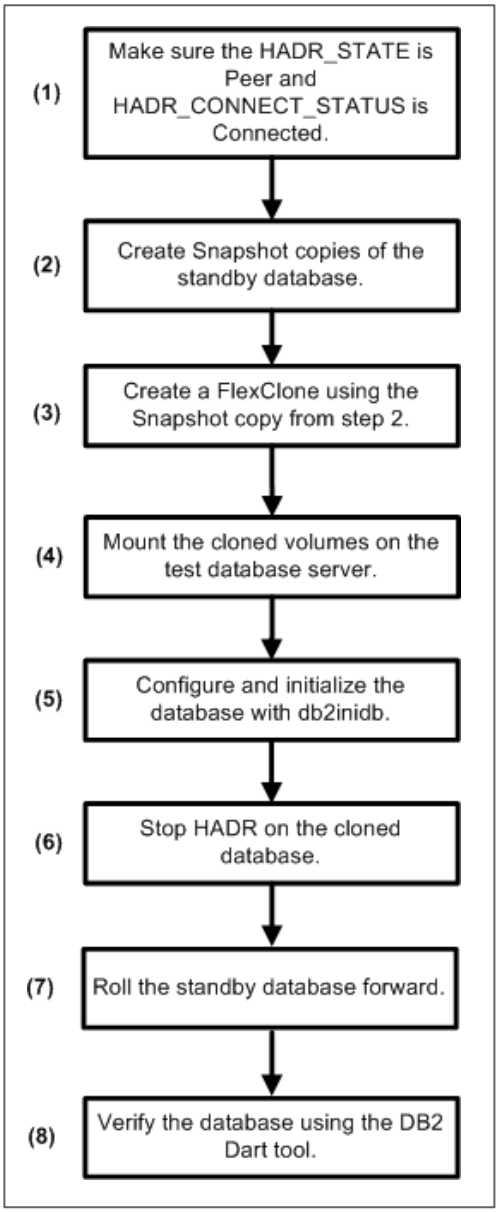
The true state of the HADR pair is maintained on the primary system. Therefore, the only way to make sure that the primary and standby are in sync is to check the status of the pair from the primary system. Execute the following command on the primary system:

```
snap_get_hadr mydb
```

Make sure that the HADR_STATE is set to PEER and the HADR_CONNECT_STATUS is set to CONNECTED.

```
DBNAME   HADR_ROLE HADR_STATE     HADR_SYNCMODE HADR_CONNECT_STATUS
-------- --------- -------------- ------------- ------------------
mydb     PRIMARY   PEER           SYNC          CONNECTED
```

Figure 7) Creating a copy of the HADR standby database using NetApp for IBM N series FlexClone.



Confirm that the HADR pair is in sync, and then suspend the transmission of the log files to standby to make sure that no physical I/O is in progress while the Snapshot copy is being created. Figure 7 lists the steps involved in creating a test environment with an HADR standby database as the source and using NetApp FlexClone technology.

Both of the database servers that we used for our HADR testing had an AIX 6.1 operating system. For information on mounting a FlexVol volume or FlexClone volume on a database server with a different operating system, refer to TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices.

If a customer has a 16TB DB2 database and wants to create a copy of the production database and refresh it every day for reporting purposes, using the traditional storage would require 16TB of storage. However, by using the NetApp storage system, only 320GB of storage is required. This is computed based on the assumption of a 2% data change between the production and the clone every day, which is

highly unlikely. Therefore, the total saving in terms of disk shelves is about four DS14 300GB disk shelves.

Table 1 shows the storage saved by using NetApp FlexClone to clone a 16TB database.

Table 1) Storage savings for cloning a 16TB database.

	Traditional storage	NetApp FlexClone
Usable storage required	16TB	320GB

8 CONCLUSIONS

NetApp storage technology offers the DB2 database administrator a compelling advantage and an elegant solution in terms of database cloning. The use of the FlexVol and FlexClone features, combined with SnapMirror, enables database cloning at a remote storage system. The clone database can be used to take load off the production database, in the reporting environment, for data mining, for production environment troubleshooting, to refresh the test and development environment, to work closely with live production data, and for disaster recovery. Database cloning using NetApp technology is very easy and simple, sparing database administrators from working late hours and weekends just to replicate data for various IT operations. The storage space is used very efficiently with Snapshot technology, and this translates into huge monetary savings.

9 CAVEATS

The information presented in this report was tested only with a specific hardware and software configuration. NetApp tested DB2 V9.7 for Linux. Hosts used for these tests were running RHEL 5.0. NetApp has not tested this configuration with other combinations of hardware and UNIX flavors. Your environment configuration and setup may have significant differences that will alter the procedures necessary to accomplish the objectives outlined in this paper. If any of the procedures described in this paper do not work in your environment, contact the authors for assistance.

10 APPENDIXES

10.1 CONFIGURE A UNIX HOST TO ACCESS BOTH THE CLONE AND THE SOURCE DATABASES IN A NAS ENVIRONMENT

To access the clone database and the source database from the same database server, complete the following steps:

1. Create a mountpoint for each clone volume by executing the following command on the database server:

```
mkdir –p [MountPoint]
```

Where

- `MountPoint` identifies the name assigned to the mount location on the database server.

For example, to create a mountpoint named `/mnt/db2data_cl`, execute the following command on the database server:

```
mkdir –p /mnt/dbdata_cl
```

2. Define the mount options in the appropriate file for your operating system. For example, for Linux, define the mount options in the `/etc/fstab` file. The mount should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

Where

- `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.
- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on a NetApp FAS or IBM N series storage system named `srcstore`, append the following mount entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

The mount entry should specify the NetApp or IBM recommended mount options. For details on the mount options for other operating systems refer to TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices.

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where

- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has mount entry specified in the `/etc/fstab` file, execute the following command on the second database server named `hostdst`:

```
mount /mnt/dbdata_cl
```

The database servers we used had a Linux operating system. For information on mounting a FlexVol volume or FlexClone volume on a database server with a different operating system, refer to TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices.

3. To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown –R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mountpoint named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

```
chown –R db2inst1:db2adm /mnt/dbdata_cl
```

4. The clone database has the old database name and old tablespace container information. Rename the clone database and update its tablespace header information to represent the new database and new tablespace containers. The `db2relocatedb` utility allows you to rename a database and to update the tablespace header information. First, create a configuration file specifying both the new and old database names and tablespace containers. The configuration file should look similar to the following:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1
NODENUM=0
LOG_DIR=/mnt/dblogs,/mnt/dblogs_cl
STORAGE_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

Save the file as `dbrelocate.cfg` and grant execute permission on it. If the database was offline during the Snapshot copy process, then modify the database information by executing the **`db2relocatedb`** command:

```
db2relocatedb -f [ConfigFile]
```

Where

- `ConfigFile` identifies the name of a configuration file that contains information that is needed to alter the DB2-specific metadata stored in files associated with a database.

For example, to update the tablespace container information and to rename the clone database using the sample configuration file named `/home/db2inst1/dbrelocate.cfg`, execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

Note: When the **`db2inidb`** command is executed with the `relocate using` option, it internally calls the `db2relocatedb` tool and relocates a database as defined in the db2relocate.cfg file. As previously mentioned, the `db2inidb` utility is used only to initialize a clone database that was created from an online database.

5. Check whether the database is cataloged correctly by executing the following command on the database server that is used to access the clone database:

```
db2 list db directory
```

The output from this command should look similar to the following:

```
System Database Directory
 Number of entries in the directory = 1

Database 1 entry:
 Database alias                       = MYPRD_CL
 Database name                        = MYPRD_CL
 Local database directory             = /mnt/dbdata_cl
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
```

6. Upon execution of the **db2relocatedb** command, the source database is automatically uncataloged. Recatalog the source database by executing the following command on the database server:

`db2 “catalog database [DatabaseName] as [DatabaseAlias] on [FileSystem]”`

Where

- `DatabaseName` identifies the name assigned to the database that is being cataloged.
- `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.
- `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named `mydb` that resides on a file system named `/mnt/dbdata`, execute the following command on the database server:

```
db2 “catalog database mydb as mydb on /mnt/dbdata”
```

After completing these steps, you should be able to access both the source and the cloned databases from the same database server.

10.2 CONFIGURE A UNIX HOST TO ACCESS BOTH THE CLONE AND THE SOURCE DATABASES IN A SAN ENVIRONMENT

To access the clone database from the same database server, complete the following steps:

1. List the LUN mappings by executing the following command from the NetApp FAS or IBM N series storage system:

```
lun show –m
```

The output from the **lun show –m** command should look similar to the following:

```
lun show –m
LUN path                Mapped to          LUN ID  Protocol
----------------------------------------------------------
/vol/dbdata/data          host_src_fcp_igp      0     FCP
```

The output shows that the LUNs on the clone volume have the same mapping as the parent.

2. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

```
lun unmap [LunPath] [iGroupName]
```

Where

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

For example, to remove an old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

```
lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
```

3. Create a new mapping for the LUNs on the FlexClone volume by using a new ID and/or by mapping them to a new igroup. Create a LUN mapping by executing the following command on the storage system:

```
lun unmap [LunPath] [iGroupName] [LunId]
```

Where

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.
- `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

```
lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
```

4. After remapping, each LUN that is on the clone volume and that is used for the clone database must be brought online by executing the following command on the storage system:

```
lun online [LunPath]
```

Where

- LunPath identifies the name assigned to the new clone volume.

For example, to bring a LUN named /`vol/dbdata_cl/data` online, execute the following command on the storage system:

```
lun online /vol/dbdata_cl/data
```

5. Create a mountpoint for each clone volume by executing the following command on the database server:

```
mkdir –p [MountPoint]
```

Where

- `MountPoint` identifies the name assigned to the mount location on the database server.

For example, to create a mountpoint named `/mnt/db2data_cl`, execute the following command on the database server:

```
mkdir –p /mnt/dbdata_cl
```

6. Refresh the HBA driver on the database server. For example, to refresh a QLogic FC HBA on a Linux database host, execute the following commands on the database server:

```
modprobe –r qla2300
modprobe –v qla2300
```

For any other operating system and HBA, refer to the OS reference manual and the HBA installation guide.

7. Obtain the LUN device names by executing the following command on the database server used to access the clone database:

```
sanlun lun show
```

Note: If your database host is running an AIX operating system, you must perform a few extra steps before you can mount the LUNs on the cloned volume. The extra steps necessary to mount LUNs for the cloned database are described in the Configure Cloned LUNs for an AIX Host in a SAN Environment appendix.

8. Mount the LUN devices by executing the following command on the database server:

```
mount [DeviceName] [MountPoint]
```

Where

- `DeviceName` identifies the name assigned to the new clone volume.
- `MountPoint` identifies the name assigned to the mountpoint that is used to mount the LUN device.

For example, to mount a LUN device that is identified by `/dev/sdb` on the database server, execute the following command on the database server:

```
mount /dev/sdb /mnt/dbdata
```

9. To operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown –R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the file system whose ownership is being changed.

For example, to change ownership of the file system mounted on the mountpoint named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

```
chown –R db2inst1:db2adm /mnt/dbdata_cl
```

10. The cloned database has the old database name and old tablespace container information. Rename the cloned database and update its tablespace header information to represent the new database and new tablespace containers. The `db2relocatedb` utility allows you to rename a database and to update the tablespace header information. First, create a configuration file specifying both the new and old database names and tablespace containers. The configuration file should look similar to the following:

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1
NODENUM=0
```

```
LOG_DIR=/mnt/dblogs,/mnt/dblogs_cl
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

Save the file as `dbrelocate.cfg`, grant execute permission if necessary, and execute the **`db2relocatedb`** command as follows:

```
db2relocatedb -f [ConfigFile]
```

Where

- `ConfigFile` identifies the name assigned to the user-created configuration file that is used to relocate the database.

For example, to update the tablespace containers' information and to rename the clone database using the sample configuration file named `/home/db2inst1/dbrelocate.cfg`, execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

11. Check whether the database is cataloged correctly by executing the following command on the database server:

```
db2 list db directory
```

The output from this command should look similar to the following:

```
db2 list db directory
 System Database Directory
 Number of entries in the directory = 1

Database 1 entry:
 Database alias                       = MYPRD_CL
 Database name                        = MYPRD_CL
 Local database directory             = /mnt/dbdata_cl
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
 Alternate server hostname            =
```

12. Upon execution of the **`db2relocatedb`** command, the source database is automatically uncataloged. Recatalog the source DB2 database. Now you can access both the source and the cloned database from the same database server.

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

After completing these steps, you should be able to access both the source and the cloned databases from the same database server.

10.3 CONFIGURE CLONED LUNS FOR AN AIX HOST IN A SAN ENVIRONMENT

When a volume containing LUNs for a DB2 database is cloned, the volume group information is preserved, and the LUNs on the cloned volume have the same volume group as the LUNs on the source volume. To mount the LUN device on an AIX database server, remove the volume group information from the LUNs and recreate the volume group. (In Cluster Failover mode, the `SANPATH` layer takes care of this cleanup work.) The following sections describe the required steps to clean cloned volume group information.

1. Clean volume group information in a stand-alone database server environment.

a. Refresh the HBA driver using the following command on the database server:

```
cfgmgr
```

b. To create a volume group for the LUNs, you must know the device name assigned to each LUN. Check the device names assigned to the LUNs on the cloned volume by executing the following command on the database server:

```
sanlun lun show
```

The output from this command looks similar to the following:

```
# sanlun lun show
                            device                                    lun
   filer:lun-pathname         filename adapter protocol lun size
state
crusher01:/vol/db2vol/dbdata    hdisk3   fcs0    fcp      3g (3221225472)
GOOD
crusher01:/vol/db2vol/dblogs    hdisk4   fcs0    fcp      3g (3221225472)
GOOD
crusher01:/vol/db2vol_cl/dbdata hdisk5   fcs0    fcp      3g (3221225472)
GOOD
crusher01:/vol/db2vol_cl/dblogs hdisk6   fcs0    fcp      3g (3221225472)
GOOD
```

In the previous output, the LUNs `/vol/db2vol_cl/dbdata` and `/vol/db2v0l_cl/dblogs` are assigned device file name `hdisk5` and `hdisk6`.

c. Next, check the physical volumes group information by executing the following command on the database server:

```
lspv
```

Output from the **`lspv`** command should look similar to the following:

```
# lspv
hdisk0          0005299a4a89243c                    rootvg          active
hdisk1          0005299a8aa192f5                    vg1             active
hdisk3          0005299a45c6fe35                    db2data         active
hdisk4          0005299a45c72af1                    db2logs         active
hdisk5          0005299a45c6fe35                    db2data         active
hdisk6          0005299a45c72af1                    db2logs         active
```

The previous example shows that the `hdisk5` device has the same volume group as the `hdisk3`, and the `hdisk6` has the same volume group as the `hdisk4`. To mount the devices that correspond to the LUNs on the cloned volumes, remove the volume group information by executing the following command on the database server:

```
chdev -l [DeviceName] -a pv=clear
```

Where

– DeviceName identifies the file name assigned to the LUN device on the database server.

For example, to remove the volume group information for a device named `hdisk5`, execute the following command on the database server:

```
chdev –l hdisk5 –a pv=clear
```

d. Next, check the physical volumes again by executing the following command on the database server:

```
lsvp
```

Output from the `lspv` command should look similar to the following:

```
# lspv
hdisk3          0005299a45c6fe35                    db2data         active
hdisk4          0005299a45c72af1                    db2logs         active
hdisk5          none                                None
hdisk6          none                                None
```

e. After clearing the volume group information, create new physical volumes by executing the following command on the database server:

```
recreatevg -y [VolumeGroup] [DeviceName]
```

Where

- `VolumeGroup` identifies the name assigned to the physical group.
- `DeviceName` identifies the file name assigned to the LUN device on the database server.

For example, to recreate a volume group named `db2datacl` on a device named `hdisk5`, execute the following command:

```
 recreatevg -y db2datacl hdisk5
```

You can check if the volume is created successfully by executing the following command on the database server:

```
lspv
```

The output from the **`lspv`** command should look similar to the following:

```
# lspv
hdisk3          0005299a45c6fe35                    db2data         active
hdisk4          0005299a45c72af1                    db2logs         active
hdisk5          0005299a45e0ea9d                    db2datacl       active
```

f. You must modify the `/etc/filesystems stanza` manually to define the unique mountpoints for the cloned LUNs.

After completing these steps, the LUNs on the cloned volume can be mounted to a mountpoint on the database server.

2. Clean volume group information in a cluster failover (CF) environment.

The multipathing or SANpath layer takes care of cleaning the volume group information, but you must recreate the volume group for the LUNs on the cloned volume. Complete the following steps to configure LUNs on the cloned volume.

a. Refresh the HBA driver using the following command on the database server:

```
cfgmgr
```

b. After refreshing the HBA, execute following command on the database server to check the primary device name assigned to LUNs on the cloned volume:

```
sanlun lun show –p
```

The output from this command looks similar to the following:

```
# sanlun lun show -p
fas270c-rtp02:/vol/db2vol_cl/dbdata.lun07 (LUN 2)  spd5 = hdisk15
```

```
    30g (32212254720)    lun state: GOOD
Filer_CF_State: Cluster Enabled  Multipath_Policy: A/PG-C
Multipath-provider: Dot Hill SANpath version  4.0.5m
-------- --------- ------------ ---- ------- ------- --------
host     filer                      primary partner SANpath
path     path            device host filer   filer   failover
state    type          filename HBA  port    port    priority
-------- --------- ------------ ---- ------- ------- --------
up       secondary       hdisk16 fcs0         0c      failover
up       primary         hdisk15 fcs0 0c              primary
up       secondary       hdisk17 fcs1         0c      failover
up       primary         hdisk18 fcs1 0c              primary
```

Note: In the previous output, the device `hdisk15` (`spd5 = hdisk15`) has been identified on the primary access path for the LUN `/vol/db2vol_cl/dbdata`. Therefore, create a volume group (VG) for `hdisk15` to access the LUN `/vol/db2vol_cl/dbdata`.

c. Check the physical volume group information for the LUNs by executing the following command on the database server:

```
lspv
```

The following output shows that the device that corresponds to the LUNs on the cloned volume does not have volume group information.

```
# lspv
hdisk0          0005299a4a89243c                    rootvg       active
hdisk1          0005299a8aa192f5                    vg1          active
hdisk15                    none                     none
hdisk16                    none                     none
hdisk17                    none                     none
hdisk18                    none                     none
```

Note: Sometimes because of the corruption of Object Data Manager (ODM) entries, you may see that the devices corresponding to the LUNs on the clone volume have volume group (VG) information. In that case, remove the VG information using the following command:

```
chdev -l [DeviceName] -a pv=clear
```

Where

- `DeviceName` identifies the file name assigned to the LUN device on the database server.

d. Next, create new physical volumes for each LUN. Use the device that is identified as a primary device by the SANpath. Create a volume group by executing the following command on the database server:

```
recreatevg -y [VolumeGroup] [DeviceName]
```

Where

- `VolumeGroup` identifies the name assigned to the physical group.
- `DeviceName` identifies the file name assigned to the LUN device on the database server.

For example, to recreate a volume group named `db2datacl` on a device named `hdisk15`, execute the following command:

```
recreatevg -y db2datacl hdisk15
```

You can check if the volume is created successfully by executing the following command on the database server:

```
lspv
```

The output from the **lspv** command should look similar to the following:

```
# lspv
hdisk0          0005299a4a89243c                    rootvg          active
hdisk1          0005299a8aa192f5                    vg1             active
hdisk15         0005299a45e0ea9d                    db2datacl       active
```

e. Modify the /etc/filesystems file manually to define the unique mountpoints for the cloned LUNs.

After completing these steps, the LUNs on the cloned volume can be mounted to a mountpoint on the database server.

10.4 CONFIGURE SNAP CREATOR TO CLONE A DB2 DATABASE

BASIC CONFIGURATION

- SNAME=MYDB. This field defines the prefix for your Snapshot copy. The naming convention should be unique because Snapshot copies on the NetApp storage system are deleted according to the naming convention and retention policy used. As a best practice, NetApp recommends configuring the SNAME field as a database name, which, in this case, is MYPRODDB.
- SNAP_TIMESTAMP_ONLY=Y. When this field is set to Y, Snapshot copies end with a DB2 timestamp (YYYYMMDDHHMMSS).
- SNAP_TIME=%USER_DEFINED. When this field is configured, you can use an existing snapshot created by Snap Creator during the backup operation. If you would like to use this setting, you must configure NTAP_SNAPSHOT_DISABLE to Y, which disables the Snapshot creation on the NetApp storage system during a FlexClone operation.
- NTAP_SNAPSHOT_DISABLE=Y. This setting tells Snap Creator not to take a snapshot.
- VOLUMES=dbdata,dblogs. These volumes are used by the database MYDB.
- NTAP_USERS=srcstore:db2inst1/db2inst1.To create a Snapshot copy or a SnapRestore operation on the NetApp storage system, create a user on the NetApp storage system with the necessary privileges. Refer to TR-3841: SnapCreator 3.2 Installation and Administration Guide for directions on how to set up the user on the storage system.
- TRANSPORT=HTTP. Snap Creator supports both HTTP and HTTPS protocols for API communications.

Note: HTTPS requires openssl-devel rpm and, at this point in time, has only been tested with Linux and AIX.

- PORT=80. The port that Snap Creator uses to communicate with the NetApp storage controller(s) is normally 80 or 443. Because we are using HTTP for the TRANSPORT type, the port is configured as 80.

NETAPP OPTIONS

- NTAP_SNAPSHOT_RETENTIONS=daily:7,weekly:4,monthly:1. This setting determines the number of NetApp Snapshot copies to retain for a given policy. Based on the setting, Snap Creator keeps at least seven daily backups, four weekly backups, and one monthly backup preserved on the NetApp storage system.
- NTAP_SNAPSHOT_RETENTION_AGE=7. This setting, which is in days, defines a retention age for the Snapshot copies.

- NTAP_CONSISTENCY_GROUP_SNAPSHOT=Y. This setting uses the Data ONTAP consistency group feature during the Snapshot operation and makes sure that all of the storage volumes of the database have the same consistency point across the volumes and storage controllers. This setting is important when a database is spanned across storage volumes and storage controllers.

Note: Enabling this option requires NTAP_CONSISTENCY_GROUP_TIMEOUT.

- NTAP_CONSISTENCY_GROUP_TIMEOUT=medium. This setting controls how long the Snapshot creation process on the NetApp storage system waits for I/O fencing between volumes to finish.

ADDITIONAL MODULES

APP_NAME=db2. This is the section that calls the DB2 module of Snap Creator to make sure that the Snapshot copies are consistent with the database.

DB2 OPTIONS (UNIX ONLY)

- DATABASES=mydb:db2inst1. This is the list of databases and the user name (db1:user1;db2:user2). NetApp recommends using the database instance ID as the user.
- DB2_CMD=db2. This is the path to the DB2 command, which is what we use to interact with the database.

CLONING OPTIONS

- NTAP_NUM_VOL_CLONES=0. Snap Creator uses this setting to decide how many clone copies of a volume to keep on the storage system. For example, to refresh your clone every day, maintain this setting so your clone copy of the database gets refreshed when you execute Snap Creator with the clone as the action.
- NTAP_CLONE_IGROUP_MAP=srcstore:dbdata/igroup1. This option executes LUN mapping right after a clone process. Therefore, manual steps are not required. This option is not valid for NFS.
- NTAP_NFS_EXPORT_HOST=hostname|ip. The hostname or IP address of the server in which the cloned volume should be exported. After this is configured, the cloned volumes are exported to the host and are ready for the NFS mounts.
- NTAP_NFS_EXPORT_ACCESS=root|read-write|read-only. This setting controls access permission to cloned volumes.
- NTAP_NFS_EXPORT_PERSISTENT= true|false. This setting allows for export permissions of cloned vol to be saved in the /etc/exports file on the storage controller so that the changes are permanent.

11 REFERENCES

- TR-3114: IBM DB2 for UNIX: Backup and Recovery Using NetApp Technologies
- TR-3272: IBM DB2 on NetApp Storage: Deployment and Best Practices
- TR-3446: SnapMirror Async Overview and Best Practices Guide
- TR-3841: SnapCreator 3.2 Installation and Administration Guide

www.netapp.com