NetApp
Go further, faster

Technical Report

# Backup, Restore, and Disaster Recovery Solutions for PostgreSQL Database on NetApp Storage Systems

Karthikeyan Nagalingam and Cindy Schneider, NetApp
April 2009 | TR-3763

TABLE OF CONTENTS

# 1 INTRODUCTION

As open source databases continue to grow and mature and with the continuing shift to control corporate spending, more and more corporations are turning to open source databases such as PostgreSQL and MySQL as viable alternatives for some applications instead of using commercial databases such as Oracle® or Microsoft® SQL Server®. NetApp currently has many customers successfully running PostgreSQL databases.

This document describes the operation of PostgreSQL on NetApp® storage systems. This report serves as a guide to get a PostgreSQL database up and running on NetApp. This guide assumes a basic knowledge of both NetApp and PostgreSQL and the underlying systems.

# 2 POSTGRESQL OVERVIEW

PostgreSQL system architecture is similar to that of most mainstream database systems and will use the same basic concepts when running on NetApp storage systems. To create a consistent Snapshot™ copy, it is necessary to quiesce the database using PostgreSQL specific SQL commands. These commands will put the database into hot backup. Two Snapshot copies will be saved for each full backup, one for the data volume and the other for the log volume. These Snapshot copies can then be sent to secondary storage or tape drives for archival.

PostgreSQL binary files can also be stored on a NetApp mountpoint for easy backup and recovery.

# 3 NETAPP RECOMMENDED SETTINGS

## 3.1 RECOMMENDED RAID

NetApp recommends using RAID-DP® for all data file and log volumes.

## 3.2 AGGREGATES AND FLEXVOL VOLUMES OR TRADITIONAL VOLUMES

With Data ONTAP® 7G, NetApp supports pooling a large number of disks into an aggregate and then building virtual volumes (FlexVol® volumes) on top of those disks. Pooling all disks into a single large aggregate and using FlexVol volumes for PostgreSQL data files and log files simplify administration of the storage, particularly for growing and reducing volume sizes without affecting performance.

## 3.3 VOLUME SIZE

While the maximum supported volume size on a NetApp system is 16TB, NetApp discourages customers from configuring individual volumes larger than 3TB for the following reasons:

- Reduced per volume backup time
- Individual grouping of Snapshot copies, qtrees, and so on
- Improved security and manageability through data separation
- Reduced risk from administrative mistakes, hardware failures, and so on

### 3.4 RECOMMENDED VOLUMES FOR DATABASE FILES AND LOG FILES

Based on common practices, the following layout is adequate for most scenarios. The recommendation is to have a single aggregate containing all the flexible volumes containing database components. Database binaries, transaction log files, and data files should reside on a separate FlexVol volume to improve performance and recovery.

**Table 1) Recommended volumes for database files and log files.**

| Files | Recommended Volumes | Comments |
|---|---|---|
| Database binaries | Dedicated FlexVol volume | During installation, binaries can be installed in an alternative location specified by running the configure script and supplying a command line option that includes:<br><br>`--prefix=PREFIX`<br><br>Install all files under the directory PREFIX instead of /usr/local/pgsql. The actual files will be installed into various subdirectories; no files will ever be installed directly into the PREFIX directory. |
| Transaction log files | Dedicated FlexVol volume | WAL and archive files. All data manipulation statements are logged here before changing the actual data. Each WAL file is 16MB in size, and when one fills up, another is created. These files are archived via the configuration parameter, **`archive_command`**, to a different location. It is recommended that these files are kept in a separate directory on the same volume as the WAL logs. |
| Data files | Dedicated FlexVol volume | Data files. All user data is written to this area. |

# 4 INSTALLATION

### 4.1 CREATING A NEW DATABASE CLUSTER

A database cluster is a collection of databases that are managed by a single server instance. The `initdb` command creates a new PostgreSQL database cluster.

1. Create and mount two NFS volumes on the storage system: one for logs (WAL and archive), and the other for data. The WAL logs are the write ahead logs. All data manipulation statements are logged here before changing the actual data. Each WAL file is 16MB in size, and when one fills up, another is created. These files are archived via the configuration parameter, `archive_command`, to a different location. NetApp recommends keeping these in a separate directory on the same volume as the WAL logs.

   **`initdb -D /mnt/filer1/pgsql -X ' /mnt/filer1/logs/pg_xlog'`**

2. After the data files are created, set the PGDATA environment variable to point to the location of the data by entering:

   **`export PGDATA=/mnt/filer1/pgsql`**

3. Start the database by entering:

   **`pg_ctl start`**

## 4.2 MIGRATING POSTGRESQL TO NETAPP STORAGE SYSTEMS

### MIGRATING POSTGRESQL VERSIONS 8.3.6 OR EARLIER

1. Create and mount two NFS volumes on the storage system: one for logs (WAL and archive), and the other for data. The WAL logs are the write ahead logs. All data manipulation statements are logged here before changing the actual data. Each WAL file is 16MB in size, and when one fills up, another is created. These files are archived via the configuration parameter, `archive_command`, to a different location. NetApp recommends keeping these in a separate directory on the same volume as the WAL logs.
2. Shut down the PostgreSQL database by entering:

   ```
   pg_ctl stop
   ```

3. Migrate the logs from their default location (`$PGDATA/pg_xlog`) to a volume on the storage system.
4. After successfully moving the log files, create a symbolic link from the original location of the logs to the new location by entering:

   ```
   mv $PGDATA/pg_xlog /mnt/filer1/logs/pg_xlog
   ```

   ```
   ln -s /mnt/filer1/logs/pg_xlog $PGDATA/pg_xlog
   ```

5. Move the database to the second mounted volume. With the database still shut down, move the data files to the new volume by entering:

   ```
   mv $PGDATA /mnt/filer1/pgsql
   ```

6. Check the configuration file (`$PGDATA/postgresql.conf`) for any necessary location changes as well (search for the old location in the file).
7. Make sure archive logging is turned on for PostgreSQL. To do this, first create the archive location and then set the "`archive_command`" parameter in the `postgresql.conf` file to the command you want to use to archive the logs.

   ```
   mkdir /mnt/filer1/logs/archive
   ```

   ```
   archive_command = 'cp -I "%p" /mnt/filer1/logs/archive/"%f"'
   ```

8. Once the data files are moved, change the PGDATA environment variable to point to the new location of the data by entering:

   ```
   export PGDATA=/mnt/filer1/pgsql
   ```

9. Restart the database by entering:

   ```
   pg_ctl start
   ```

### MIGRATING POSTGRESQL VERSION 8.3.6 AND LATER

Starting with release 8.3.6, an option was added to `initdb` to allow specifying the location of the `pg_xlog` directory. A soft link to the log file directory is no longer required (see www.postgresql.org/docs/8.3/interactive/app-initdb.html).

1. Create and mount two NFS volumes on the storage system: one for logs (WAL and archive), and the other for data. The WAL logs are the write ahead logs. All data manipulation statements are logged here before changing the actual data. Each WAL file is 16MB in size, and when one fills up, another is created. These files are archived via the configuration parameter, `archive_command`, to a different location. NetApp recommends keeping these in a separate directory on the same volume as the WAL logs.
2. Shut down the PostgreSQL database by entering:

   ```
   pg_ctl stop
   ```

3. Migrate the logs from their default location (`$PGDATA/pg_xlog`) to a volume on the storage system by entering:

```
mv $PGDATA/pg_xlog /mnt/filer1/logs/pg_xlog
```

4. Modify the `$PGDATA/postgresql.conf` file and set the `log_directory` to the correct mountpoint and directory by entering:

   ```
   log_directory = ' /mnt/filer1/logs/pg_xlog'
   ```

5. Move the database to the second mounted volume. With the database still shut down, move the data files to the new volume by entering:

   ```
   mv $PGDATA /mnt/filer1/pgsql
   ```

6. Check the configuration file (`$PGDATA/postgresql.conf`) for any necessary location changes as well (search for the old location in the file).

7. Make sure archive logging is turned on for PostgreSQL. To do this, first create the archive location and then set the "`archive_command`" parameter in the `postgresql.conf` file to the command you want to use to archive the logs:

   ```
   mkdir /mnt/filer1/logs/archive
   ```

   ```
   archive_command = 'cp –I "%p" /mnt/filer1/logs/archive/"%f"'
   ```

8. After the data files are moved, change the PGDATA environment variable to point to the new location of the data by entering:

   ```
   export PGDATA=/mnt/filer1/pgsql
   ```

9. Restart the database by entering:

   ```
   pg_ctl start
   ```

# 5 BACKUP AND RESTORE

## 5.1 BACKUP AND RESTORE USING NETAPP SNAPSHOT TECHNOLOGY

### POSTGRESQL DATABASE BACKUP USING SNAPSHOT TECHNOLOGY

1. To create Snapshot copies of the database, first instruct the database to place a mark in the current log files designating the backup. To do this, start the psql utility and execute the `pg_start_backup` SQL command. In the event the database needs to be restored from a Snapshot copy, it starts replaying the logs from this mark in the log file.

   ```
   # psql test
   ```

   ```
   test=# select pg_start_backup('label');
   ```

   Where label is the name you want to use to label this backup.

2. Create a Snapshot copy by entering:

   ```
   snap create pgdata pgdata.hot.1
   ```

   ```
   snap create pglog pglog.hot.1
   ```

3. Stop the database backup by entering:

   ```
   psql test
   ```

   ```
   test=# select pg_stop_backup();
   ```

4. Make sure you record and track the log file used at the time the Snapshot copy was created. The name of the log file will change to:

   *<big number>.<weird number>*.backup.

   Example: 000000010000000200000A3.00000020.backup

The file with the highest number and latest timestamp is the first log file necessary to do a restore from this Snapshot copy. You will need this log file and all others created after it to restore up to the last good transaction when restoring from the Snapshot copy you just created. Older log files will be archived to the archive directory by PostgreSQL using the `archive_command` command set in the `postgresql.conf` file (see section 4.2).

5. If you want to keep multiple Snapshot copies, make sure to create a Snapshot copy of the logs volume. To clone this database, a Snapshot copy of the logs is required.

#### RESTORING A POSTGRESQL DATABASE FROM A SNAPSHOT COPY

To successfully restore from a Snapshot copy:

1. Make sure the database is shut down by entering:

   ```
   pg_ctl stop
   ```

   a. If you are recovering from a crash, kill the postgres postmaster process and any of its children.

      ```
      ps –ef | grep postgres
      ```

   b. You may also have to remove semaphores and/or shared memory. The ipcclean utility can be used for this process. For information, see the *PostgreSQL Manual.*

2. Restore the Snapshot copy of the data.

   **Note**: Restore only the data, unless you want to overwrite your logs.

3. Make sure you have all the log files starting with the first one that was in use during creation of this Snapshot copy. If you are missing any logs, check the archive directory.

4. As hot Snapshot copies are created with the database running, there is a file called `postmaster.pid` located in `$PGDATA`. Make sure to remove this file by entering:

   ```
   rm $PGDATA/postmaster.pid
   ```

5. Once you have all the logs you want to roll forward in the `pg_xlog` directory, start the database and enter:

   ```
   pg_ctl start –l /usr/local/pgsql/logfile
   ```

## 5.2 BACKUP AND RESTORE USING POSTGRESQL

#### PG_DUMP

Pg_dump is a PostgreSQL client application for backing up the PostgreSQL database. Run this utility as the database superuser. Pg_Dump creates a text file with SQL commands, which recreates the database in the same state as it was at the time of dump.

```
pg_dump dbname > outfile
```

Options for pg_dump:

- -h host: Default is localhost or PGHOST
- -p port: Default is 5432 (can be modified during compilation) or PGPORT
- -U user: Default is database username or PGUSER
- -o oids: If the database relies on foreign key
- -Fc: Compress the data dumped into the file during backup

pg_dump does not block other operations while it is working The ability of pg_dump and psql to write to or read from pipes makes it possible to dump a database directly from one server to another, for example:

```
pg_dump -h host1 dbname | psql -h host2 dbname
```

`Pg_dumpall` backs up each database in a given cluster and also preserves cluster-wide data such as role and tablespace definitions. `Pg_dumpall` dumps all the databases with their role information and tablespaces.

```
pg_dumpall > outfile
```

The text files created by pg_dump are intended to be read in by the psql program. The general command form to restore a dump is:

```
psql dbname < infile
```

Make sure the database is already created before executing the psql command.

The resulting dump can be restored with psql:

```
psql -f infile postgres
```

Large databases can be backed up using gzip and tar:

```
pg_dump dbname | gzip > filename.gz
```

```
tar –zcvf backup.tar /postgresql_data/data
```

**Note**: Database should be down for usable backup.

Reload by entering:

```
gunzip -c filename.gz | psql dbname
```

A custom-format dump is not a script for psql, but instead must be restored with `pg_restore`, for example:

```
pg_restore -d dbname filename
```

#### CONTINUOUS ARCHIVING AND POINT-IN-TIME RECOVERY (PITR)

PostgreSQL maintains a WAL in the `pg_xlog` directory, which describes each change made to the database's datafiles. During recovery we can replay the log entries made since the last checkpoint.

1. Check section 4.2.2, which explains about the archive log enabling.
2. As a superuser, do as follows:
   a) Select `pg_start_backup('label');`.
   b) Perform backup using `pg_dump/pg_dumpall/gip/tar`.
   c) Select `pg_stop_backup();`.

   The archival procedure is completed when the WAL segment files used during the backup are archived. The file identified by the result of `pg_stop_backup` is the last segment that needs to be archived to complete the backup. Archival of these files happens automatically, as you have already configured `archive_command`. In many cases, this happens fairly quickly, but you are advised to monitor your archival system to make sure this has taken place so that you can be certain you have a complete backup.

# 6 DR SOLUTIONS: POSTGRESQL WITH NETAPP

The two architectures described in this section provide usage scenarios for small/medium business as well as large enterprises. Architecture 1 is probably best suited for small to medium-sized businesses, because the database, log, and bin volumes of production database servers are synchronized with secondary database servers as well as clients accessing the same volumes during production use. Architecture 2 is used when we want to separate client operations through SnapMirror®. This architecture is quite useful for database servers under heavy loads such as those used in large enterprises (example: the banking sector). The choice of architecture is based on the customers' needs.

## 6.1 ARCHITECTURE 1

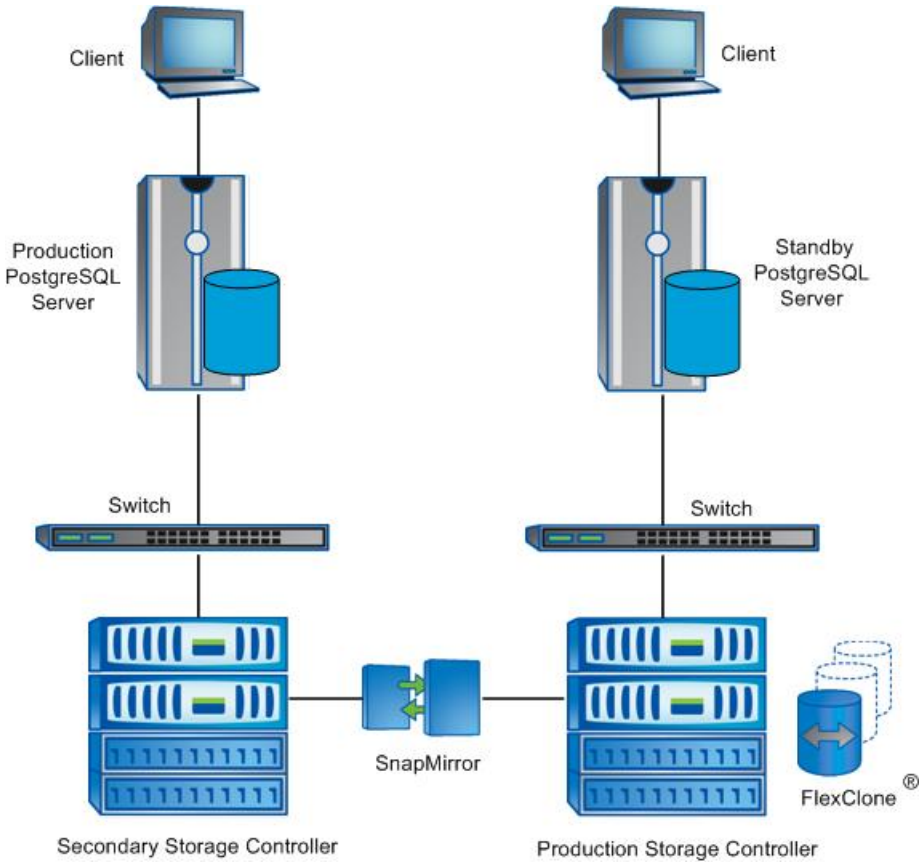


**Figure 1) System architecture 1.**

### SOLUTION OVERVIEW

The production PostgreSQL server serves clients, which access data using client programs such as PHP, Perl, and so on. The production PostgreSQL server accesses three volumes from the production storage controller for the PostgreSQL binary, data, and log files. The following procedure describes how to set up this architecture:

1. Create three volumes, each with same or 10% extra space of the production volumes, in a secondary storage controller with the same name as the production storage controller volume names.

2. Install PostgreSQL in the bin volume and change the "archivelog" and "log_directory" values in /postgresql_data/data/postgresql.conf for data and log volume mounted folders. Check the appendix for example configuration.
3. Check the Data Definition Language (DDL) statements, Data Manipulation Language (DML) statements, Data Control Language (DCL) statements, and Transaction Control (TCL) statement operations in PostgreSQL database.
4. Configure the synchronous SnapMirror relationship between the production and secondary storage controllers.
5. Check if the the SnapMirror status appears as "In sync".
6. During disaster recovery, disconnect the SnapMirror volumes from secondary storage controller.
7. Shutdown the PostgreSQL service in the standby server.
8. Map the SnapMirror destination volumes such as bin, data, and log volumes to the standby server.
9. Check the operating system details for PostgreSQL in the standby PostgreSQL server as same as production PostgreSQL server such as IP address, database user ID, and group; export the LD_LIBRARY_PATH in database user environment.
10. Start the PostgreSQL service, check the database operations, and check the tablespace and records.

    Now the standby server works like the production database server; the client continues to access the database through standby database server.

### SETUP

1. Production storage controller settings:
   a) For testing purposes we are using a Linux® operating system. The same configuration is applicable for the Solaris™ operating system.
   b) Create three volumes for database, binary, and log files.
   c) NFS protocol is used in our test setup. Export them as follows:

```
/vol/pgbin    -sec=none,rw,anon=0
/vol/pgsqldata -sec=none,rw,anon=0
/vol/pgsqllog  -sec=none,rw,anon=0
```

   d) Enable the SnapMirror options as follows:

```
btcppe5> options snapmirror
snapmirror.access       all
snapmirror.checkip.enable  off
snapmirror.delayed_acks.enable on
snapmirror.enable       on
snapmirror.log.enable     on
snapmirror.vbn_log_enable  off
```

2. Production PostgreSQL server settings:
   a) Linux operating system is used in our test setup.
   b) Mount the pgbin, pgsqldata, pgsqllog volumes in /postgresql_bin, /postgresql_data, /postgresql_log folders.
   c) Install PostgreSQL and the binary in the /postgresql_bin folder; database files in /postgresql_data.
   d) Configure the WAL and log files to /postgresql_log.
   e) In the /etc/init.d/postgresql-8.3 file, checks if the datafile and log file locations are correct.

f) Enable the archive log, as described in section 4.2.

g) Start the PostgreSQL service using either "/etc/init.d/postgresql-8.3 start" or "service postgresql start."

3. Secondary storage controller settings:

a) Create three volumes for database, binary, and log files, each with a little more space than production storage controller volumes for temporary or transient data.

b) For the testing purpose, we used NFS protocol. Export them like below:

```
/vol/pgbin   -sec=none,rw,anon=0
/vol/pgsqldata -sec=none,rw,anon=0
/vol/pgsqllog  -sec=none,rw,anon=0
```

c) Enable the SnapMirror options as follows:

```
btcppe6> options snapmirror
snapmirror.access      all
snapmirror.checkip.enable  off
snapmirror.delayed_acks.enable on
snapmirror.enable      on
snapmirror.log.enable    on
snapmirror.vbn_log_enable  off
```

d) Create the `snapmirror.conf` file with the following:

```
btcppe6> wrfile /etc/snapmirror.conf
btcppe5:pgsqldata btcppe6:pgsqldata - sync
btcppe5:pgsqllog btcppe6:pgsqllog - sync
btcppe5:pgbin btcppe6:pgbin - sync
```

e) Restrict the volumes by entering:

```
btcppe6> vol restrict pgbin
btcppe6> vol restrict pgsqldata
btcppe6> vol restrict pgsqllog
```

f) Initialize the volumes and sync to the production storage controller volumes by entering:

```
Snapmirror initialize –S btcppe5:pgbin pgbin
Snapmirror initialize –S btcppe5:pgsqldata pgsqldata
Snapmirror initialize –S btcppe5:pgsqllog pgsqllog
```

g) Check the `snapmirror status` output. Wait until the status changes to "`In-Sync.`"

```
btcppe6> snapmirror status
Snapmirror is on.
Source          Destination   State          Lag     Status
btcppe5:pgbin  btcppe6:pgbin Snapmirrored   16:36:42 In-Sync
btcppe5:pgsqldata btcppe6:pgsqldata Snapmirrored 16:36:41  In-Sync
btcppe5:pgsqllog btcppe6:pgsqllog Snapmirrored 16:36:46 In-Sync
```

4. Standby PostgreSQL server settings:

a) Create the local database user name "`postgres`" with same UID, GID, home location, password as production PostgreSQL server.

b) Stop the PostgreSQL service by entering either:

```
/etc/init.d/postgresql-8.3 stop
```

Or

```
service postgresql stop
```

c) Export the `LD_LIBRARY_PATH=/postgresql_bin/PostgreSQL/8.3/lib` to the "`postgres`" environment.

d) In case of a disaster, break the SnapMirror relationship in secondary storage controller:

```
snapmirror quiesce pgbin;
snapmirror break -f pgbin;
snapmirror quiesce pgsqldata;
snapmirror break -f pgsqldata;
snapmirror quiesce pgsqllog;
snapmirror break -f pgsqllog;"
```

e) Mount the `pgbin`, `pgsqldata`, `pgsqllog` volumes in `/postgresql_bin`, `/postgresql_data`, `/postgresql_log` folders.

f) Restart the PostgreSQL services by entering either:

```
/etc/init.d/postgresql-8.3 restart
```

Or

```
service postgresql restart
```

g) Check the database operations and check if the production databases are mirrored. Change the IP address to the production PostgreSQL server IP. Now the standby server is ready to serve the clients.

## 6.2 ARCHITECTURE 2

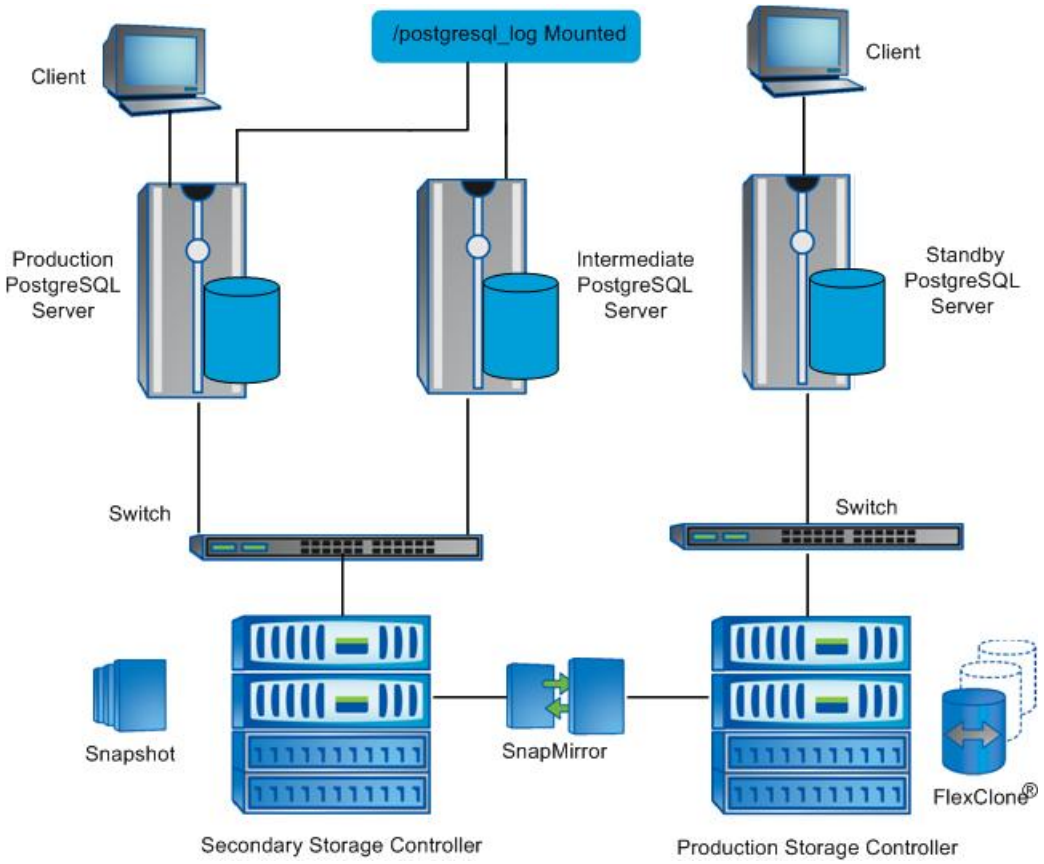

Figure 2) System architecture 2

### SOLUTION OVERVIEW

In this architecture, the settings for production storage controller and secondary storage controller are same as previous solution.

Using the full backup Snapshot copy of the data and bin volume, create the base database in intermediate database server. This is a one-time activity.

The log files volume is mounted in both production and intermediate PostgreSQL server.

Apply the logs needed to the intermediate database to sync the database with production PostgreSQL server.

The main advantage here is that the production server database, bin volumes are not used for the DR and backup solution. This is one of the recommended solutions for large enterprises.

# 7 FAQS

**Where is PostgreSQL used?**

PostgreSQL is used for Web applications, content management systems, and so on.

The basic deployment is based on free cluster products (such as Piranha), and they have been using MySQL and PostgreSQL for deployments, especially in government (adopt free software). Basically, they have servers using bond (to aggregate) interfaces to NetApp.

**What are the protocols supported by PostgreSQL?**

- FCP
- iSCSI
- NFS

**What are the applications used with PostgreSQL?**

- OLTP (small DB for Web environments, for specific applications)
- Content management
- RT (ticketing system), Bugzilla, media wiki
- Web applications

**What are the disaster recovery solutions for PostgreSQL using NetApp storage systems?**

- SnapMirror to remote data center with one-hour RPO. SnapVault® is also used for disaster recovery.
- Asynchronous disaster recovery
- For Linux DRBD solution is quite useful. DRBD refers to block devices designed as a building block to form high-availability (HA) clusters. This is done by mirroring a whole block device via an assigned network. DRBD can be understood as network-based RAID 1.
- Some tools such as Slony-I and Mammoth Replicator can also be used.

# APPENDIXES

# APPENDIX A: CONFIGURATION SCRIPTS

## A.1 POSTGRESQL CONFIGURATION

The basic configuration of `/postgresql_data/data/postgresql.conf` is as follows:

```
port = 5432                # (change requires restart)
max_connections = 100          # (change requires restart)
shared_buffers = 32MB          # min 128kB or max_connections*16kB
max_fsm_pages = 204800         # min max_fsm_relations*16, 6 bytes each
checkpoint_timeout = 5min        # range 30s-1h
archive_mode = on        # allows archiving to be done
                # (change requires restart)
archive_command = 'cp -i %p /postgresql_log/pg_xlog/archive/%f'     # command to
use to archive a logfile segment
archive_timeout = 300      # force a logfile segment switch after this
logging_collector = on     # Enable capturing of stderr and csvlog
log_directory = '/postgresql_log/pg_log'        # directory where log files are
written,
                    # can be absolute or relative to PGDATA
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log' # log file name pattern,
datestyle = 'iso, mdy'
lc_messages = 'en_US.UTF-8'           # locale for system error message
                    # strings
lc_monetary = 'en_US.UTF-8'           # locale for monetary formatting
lc_numeric = 'en_US.UTF-8'           # locale for number formatting
lc_time = 'en_US.UTF-8'             # locale for time formatting
default_text_search_config = 'pg_catalog.english'
```

## A.2 DAILYBACKUP SCRIPT: DAILYPGSQLBACKUP.PL

**DAILYPGSQLBACKUP.PL**

```
#!/usr/bin/perl
#print "Content-type: text/html\n\n";
#use strict;

######################################################################
# Please Modify the following parameter as per your environment      #
######################################################################
```

```
my $host        = 'localhost';

my $dbport       = '5432';

my $db         = 'test';

my $dbuser       = 'postgres';

my $dbuserpass     = 'btcppe1';

my $FilerName      = '10.73.68.105';

my $FilerUser      = 'root';

my $RSH_Access     = 'NO';     #YES for Passwordless access to filer, NO for
With password access to filer

my $FilerUserPass    = 'btcppe1';      #Provide the password, If the $RSH_Access
is "NO"

#####################################################################

# Parameter Modification Ends here               #

#####################################################################

my $SNAP        = 'snap';

my $CREATION      = 'create';

my $SNAPSHOT      = 'SShot'.`date +%d-%m-%y-%H-%M-%S`;

my $LABEL        = "$SNAPSHOT";

my $RSHPGM       = '/usr/bin/rsh';


use DBI;


    #make connection to database

    my $connectionInfo="DBI:Pg:database=$db";"host=$host";"port=$dbport";

    my $dbh = DBI->connect($connectionInfo,$dbuser,$dbuserpass) or die "Couldn't
connect to ----------- database: $DBI::errstr\n";


    #usage

    if ( scalar(@ARGV) < 1 ){

        die "Usage: \n\tperl dailypgsqlbackup.pl <vol-name> .....
\nExample:\n\tperl dailypgsqlbackup.pl pgsqldata pgbin pgsqllog\n\nMinimum One
volume name required\n\n";

    }


    #locking tables with read only mode

    my $query = "select pg_start_backup('$LABEL')";

    my $sth1234 = $dbh->prepare($query);

    $sth1234->execute() or die "Couldn't connect to database: $DBI::errstr\n";

    $sth1234->finish();
```

```
    chomp($SNAPSHOT);
    chomp($LABEL);

    $FilerUser = "$FilerUser:$FilerUserPass" if( $RSH_Access eq 'NO' );

    print "\n\n"."Start snap - Daily\n";
    my $tnum = scalar(@ARGV);
    while( $tnum >= 1 ){
        my $temp = --$tnum;
        system("date;sync;$RSHPGM -l $FilerUser $FilerName $SNAP $CREATION
$ARGV[$temp] $ARGV[$temp]$SNAPSHOT;date");
    }
    print ""."End snap - Daily\n";

    my $query = "select pg_stop_backup()";
    my $sth1234 = $dbh->prepare($query);
    $sth1234->execute() or die "Couldn't connect to database: $DBI::errstr\n";
    $sth1234->finish();

#disconnect from database
$dbh->disconnect;
```

HOURLYBACKUP SCRIPT - hourlypgsqlbackup.pl

#### HOURLYPGSQLBACKUP.PL

```
#!/usr/bin/perl

#print "Content-type: text/html\n\n";
#use strict;

######################################################################
# Please Modify the following parameter as per your environment      #
######################################################################
my $host        = 'localhost';
my $dbport       = '5432';
my $db         = 'test';
my $dbuser       = 'postgres';
my $dbuserpass     = 'btcppe1';
```

```
my $FilerName      = '10.73.68.105';

my $FilerUser      = 'root';

my $RSH_Access     = 'NO';     #YES for Passwordless access to filer, NO for
With password access to filer

my $FilerUserPass    = 'btcppe1';  #Provide the password, If the $RSH_Access is
"NO"

#####################################################################

# Parameter Modification Ends here               #

#####################################################################

my $SNAP        = 'snap';

my $CREATION      = 'create';

my $SNAPSHOT      = 'SShot'.`date +%d-%m-%y-%H-%M-%S`;

my $LABEL        = "$SNAPSHOT";

my $RENAME       = 'rename';

my $RSHPGM       = '/usr/bin/rsh';


use DBI;


    #make connection to database

    my $connectionInfo="DBI:Pg:database=$db";"host=$host";"port=$dbport";

    my $dbh = DBI->connect($connectionInfo,$dbuser,$dbuserpass) or die "Couldn't
connect to ----------- database: $DBI::errstr\n";


    #usage

    if ( scalar(@ARGV) < 1 ){

        die "Usage: \n\tperl hourlypgsqlbackup.pl <vol-name> .....
\nExample:\n\tperl hourlypgsqlbackup.pl pgsqldata pgbin pgsqllog\n\nMinimum One
volume name required\n\n";

    }


    #locking tables with read only mode

    my $query = "select pg_start_backup('$LABEL')";

    my $sth1234 = $dbh->prepare($query);

    $sth1234->execute() or die "Couldn't connect to database: $DBI::errstr\n";

    $sth1234->finish();


    chomp($SNAPSHOT);

    chomp($LABEL);
```

```
    #Change the FilerUser with password if the RSH_Access is NO
    $FilerUser = "$FilerUser:$FilerUserPass" if( $RSH_Access eq 'NO' );


    print "\n\n"."Delete the 24th Hour Snapshot\n";
    system("$RSHPGM -l $FilerUser $FilerName $SNAP delete $ARGV[0]
$ARGV[0]_recent24");


    #renaming the current hour snapshot to the next hour snapshot for taking the
current hour snapshot time consume for the rename process will not affect the
mysql
    print "\n\n"."Renaming the snapshot, Please wait...\n";
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent23 $ARGV[0]_recent24");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent22 $ARGV[0]_recent23");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent21 $ARGV[0]_recent22");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent20 $ARGV[0]_recent21");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent19 $ARGV[0]_recent20");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent18 $ARGV[0]_recent19");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent17 $ARGV[0]_recent18");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent16 $ARGV[0]_recent17");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent15 $ARGV[0]_recent16");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent14 $ARGV[0]_recent15");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent13 $ARGV[0]_recent14");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent12 $ARGV[0]_recent13");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent11 $ARGV[0]_recent12");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent10 $ARGV[0]_recent11");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent9 $ARGV[0]_recent10");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent8 $ARGV[0]_recent9");
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent7 $ARGV[0]_recent8");
```

```
    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent6 $ARGV[0]_recent7");

    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent5 $ARGV[0]_recent6");

    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent4 $ARGV[0]_recent5");

    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent3 $ARGV[0]_recent4");

    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent2 $ARGV[0]_recent3");

    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent1 $ARGV[0]_recent2");

    #system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent0 $ARGV[0]_recent1;date");

    system("$RSHPGM -l $FilerUser $FilerName $SNAP $RENAME $ARGV[0]
$ARGV[0]_recent0 $ARGV[0]_recent1");


    print "\n\n"."Start backup - Hourly\n";

    my $tnum = scalar(@ARGV);

    while( $tnum >= 1 ){

        my $temp = --$tnum;

        system("date;sync;$RSHPGM -l $FilerUser $FilerName $SNAP $CREATION
$ARGV[0] $ARGV[0]_recent0;date");

    }

    print ""."End backup - Hourly\n";


    my $query = "select pg_stop_backup()";

    my $sth1234 = $dbh->prepare($query);

    $sth1234->execute() or die "Couldn't connect to database: $DBI::errstr\n";

    $sth1234->finish();


#disconnect from database

$dbh->disconnect;
```

## A.3 INTERACTIVE BACKUP: SNAPFUNCTION.PM, SNAPPGSQL.PL

**SNAPFUNCTION.PM**

```
#!/usr/bin/perl

#print "Content-type: text/html\n\n";

#use strict;

package snapfunction;

$RSHPGM = "/usr/bin/rsh";
```

```
sub hostname
{
    print("\n" . "Enter the HostName [ Default: localhost ]\t: ");
    my $HostName = <STDIN>;
    chomp($HostName);
    $HostName = 'localhost' if ($HostName eq '');
    return ($HostName);
}

sub port
{
    print("\n" . "Enter the PortNum [ Default: 5432 ]\t\t: ");
    my $PortNum = <STDIN>;
    chomp($PortNum);
    $PortNum = '5432' if ($PortNum eq '');
    return ($PortNum);
}


sub dbname
{

  my ($HostName) = @_;
  my $DbName;
  while (1)
  {

    print("\n" . "Enter the DB name for \"$HostName\"\t\t\t: ");
    $DbName = <STDIN>;
    chomp($DbName);
    if ($DbName eq '')
    {
      print("\n\n" . "Please re-enter the DB Name for $HostName" . "\n");
      print("\n" . "PRESS <ENTER> TO CONTINUE : " . "");
      <STDIN>;
      next;
    }
    else
```

```
    {
      last;
    }
  }
  return ($DbName);
}

sub dbusername_pass {

    print("\n" . "Enter the Database User [ Default: postgres ]\t: ");
    my $DbUserName = <STDIN>;
    chomp($DbUserName);
    $DbUserName = 'postgres' if ($DbUserName eq '');

    my $DbUserPass = user_passwd($DbUserName);

    my $pass_string = '';
    for (my $i ; $i < length($DbUserPass) ; $i++)
    {
      $pass_string .= '*';
    }
    return ($DbUserName,$DbUserPass, $pass_string);
}

sub user_passwd
{

  my ($UserName) = @_;
  my $UserName_Passwd = '';
  while (1)
  {
    print("\n" . "Password for \"$UserName\" User \t\t\t: ");
    system "stty -echo";
    $UserName_Passwd = <STDIN>;
    system "stty echo";
    chomp($UserName_Passwd);

    if ($UserName_Passwd eq '')
```

```
    {
      print("\n" . "Please re-enter the Password for $UserName" . "\n");
      print("\n" . "PRESS <ENTER> TO CONTINUE : " . "");
      <STDIN>;
      next;
    }
    else
    {
      last;
    }
  }
  return ($UserName_Passwd);
}

sub Filername_user {

  my $HostName;

  while (1)
  {

    print("\n\n" . "Enter the FilerName/IP \t\t\t\t: ");
    $HostName = <STDIN>;
    chomp($HostName);
    if ($HostName eq '')
    {
      print("\n\n" . "Please re-enter the FilerName" . "\n");
      print("\n" . "PRESS <ENTER> TO CONTINUE : " . "");
      <STDIN>;
      next;
    }
    else
    {
      last;
    }
  }

  print ("\n"."Enter the Admin User in \"$HostName\" [ Default:root ] : "."");
```

```
  my $username=<STDIN>;
  chomp($username);
  $username='root' if($username eq '');

  return ($HostName, $username);
}


sub FilerRSH {

  my ($FilerName, $FilerUser) = @_;

  print("\n"."Do you have the RSH with \"$FilerName\" Without Password: [
Default: yes ]");
  my $confirm=<STDIN>;
  chown($confirm);
  $confirm='yes' if($confirm eq '');

  my $cmd = "$RSHPGM -l $FilerUser $FilerName date";
  my $output = `$cmd 2>&1`;
  $output = '' unless defined($output);
  my $status = $?;

  my ($pass_string,$UserPass) ='' ;

  if($status){
    print("\nRSH Access Without Password Not enabled\n");
    chomp($FilerUser);
    my $UserPass = user_passwd($FilerUser);

    for (my $i ; $i < length($UserPass) ; $i++)
    {
      $pass_string .= '*';
    }
  return("NO", $UserPass, $pass_string);
  }else{
    return("YES", $UserPass, $pass_string);
  }
```

```
}


1;
```

**SNAPPGSQL.PL**

```
#!/usr/bin/perl

#print "Content-type: text/html\n\n";

#use strict;


use DBI;

use snapfunction;


    #usage

    if ( scalar(@ARGV) < 1 ){

        die "Usage: \n\tsnapmysqldaily <vol-name> .....
\nExample:\n\tsnappgsqldaily pgsqldata pgbin pgsqllog\n\nMinimum One volume name
required\n\n";

    }


my $host = snapfunction::hostname();

my $port = snapfunction::port();

my $db = snapfunction::dbname($host);

my ($userid, $passwd, $pass_string) = snapfunction::dbusername_pass();

my $connectionInfo="DBI:Pg:database=$db";"host=$host";"port=$port";

my $SNAP="snap";

my $CREATION="create";

my $SNAPSHOT="SShot".`date +%d-%m-%y-%H-%M-%S`;

chomp($SNAPSHOT);

my $RSHPGM="/usr/bin/rsh";

my ($FilerName, $FilerUser) = snapfunction::Filername_user();

my ($Access, $UserPass, $pass_string) = snapfunction::FilerRSH($FilerName,
$FilerUser);

my $LABEL="$SNAPSHOT";


    #make connection to database

    my $dbh = DBI->connect($connectionInfo,$userid,$passwd) or die "Couldn't
connect to ----------- database: $DBI::errstr\n";
```

```
    #locking tables with read only mode
    my $query = "select pg_start_backup('$LABEL')";
    my $sth1234 = $dbh->prepare($query);
    $sth1234->execute() or die "Couldn't connect to database: $DBI::errstr\n";
    $sth1234->finish();

    $FilerUser = "$FilerUser:$UserPass" if($Access eq "NO" );

    #system("$RSHPGM -l root BTCPPE-FILER-5 snapmirror status");
    print "\n\n"."Start snap - Daily\n";
    my $tnum = scalar(@ARGV);
    while( $tnum >= 1 ){
        my $temp = --$tnum;
        system("date;sync;$RSHPGM -l $FilerUser $FilerName $SNAP $CREATION
$ARGV[$temp] $ARGV[$temp]$SNAPSHOT;date");
    }
    print "\n"."End snap - Daily\n";

    my $query = "select pg_stop_backup()";
    my $sth1234 = $dbh->prepare($query);
    $sth1234->execute() or die "Couldn't connect to database: $DBI::errstr\n";
    $sth1234->finish();

#disconnect from database
$dbh->disconnect;
```

## A.4 SNAPMIRROR: RESYNCHRONIZATION

During disaster recovery we use the standby server for production purposes. Suppose the production PostgreSQL server is repaired and ready for the production. We need to update the production storage controller with modified data from the secondary storage controller.

1. Shut down the PostgreSQL service in production PostgreSQL server by entering:

   **/etc/init.d/postgresql-8.3 stop**

2. Unmount the mounted partitions by entering:

   **umount /postgresql_data**
   **umount /postgresql_log**
   **umount /postgresql_bin**

3. From production storage controller, enter:

   **snapmirror resync -S btcppe6:pgsqldata pgsqldata**
   **snapmirror resync -S btcppe6:pgsqlbin pgbin**
   **snapmirror resync -S btcppe6:pgsqllog pgsqllog**

4. Break the SnapMirror relationship:

```
Snapmirror break pgsqldata
Snapmirror break pgbin
Snapmirror break pgsqllog
```

5. Mount them back by entering:

```
mount /postgresql_data
mount /postgresql_log
mount /postgresql_bin
```

6. Start the PostgreSQL service by entering:

```
/etc/init.d/postgresql-8.3 start
```

## APPENDIX B: ADDITIONAL RESOURCES

| Link | Description |
|---|---|
| www.postgresql.org/ | Main PostgreSQL Web site. |
| http://wiki.postgresql.org/ | This wiki contains user documentation, how-tos, and tips and tricks related to PostgreSQL. It also serves as a collaboration area for PostgreSQL contributors. |
| http://en.wikipedia.org/wiki/PostgreSQL | From Wikipedia. |