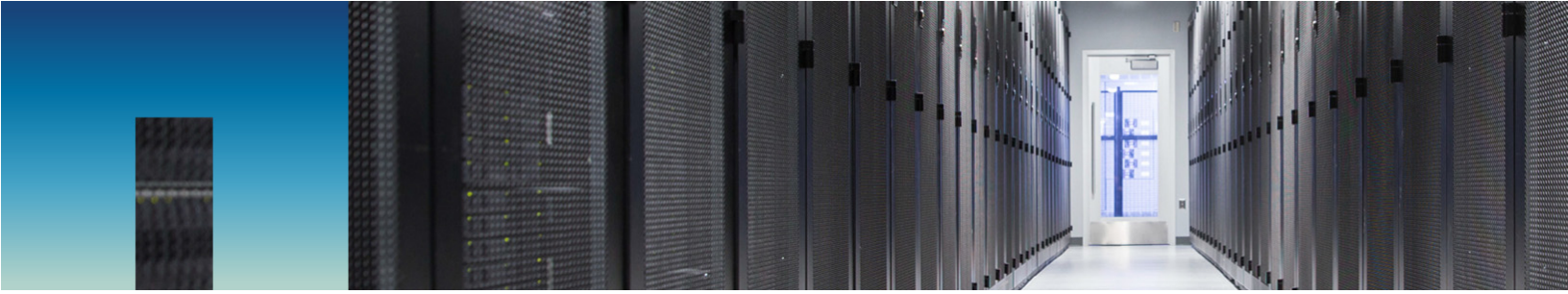


Technical Report

PostgreSQL Best Practices on NetApp SolidFire

Bobby Oommen, NetApp
November 2017 | TR-4610

Abstract

NetApp® SolidFire® storage offers a compelling advantage for a wide range of database application use cases. This document introduces SolidFire storage to the database administrator and provides important system design paradigms to consider when using SolidFire storage for a PostgreSQL database. From these design points, the reader can learn about the application profiles that are best suited for SolidFire storage and how to identify those types of applications.

NetApp®

TABLE OF CONTENTS

1 Introduction

NetApp SolidFire storage systems were born out of some of the of the largest cloud infrastructures in the world. They are designed to serve next-generation data center needs such as scaling with multitenancy, set-and-forget management, and guaranteed performance. SolidFire architecture provides you with greater predictability for your shared storage infrastructure. SolidFire storage optimizes solid-state drive (SSD) capacity to create high utilization and volume performance. The innovative architecture of SolidFire features can be leveraged across a number of different Oracle Database use cases to provide real benefits to administrators.

This guide discuses basic steps and best practices to set up an Oracle Database instance on SolidFire storage volumes. NetApp SolidFire storage systems have the following capabilities that support next-generation data center needs:

- Thin provisioning
- Compression and deduplication
- Quality of service (QoS)

These features reduce the amount of storage space that is required without affecting performance. You can use these features with various database use cases.

1.1 Thin Provisioning

SolidFire uses 4K granular thin provisioning that does not require any reserve space, increasing effective capacity by consuming less space. This feature increases efficiency and reduces overhead by using the smallest allocation possible and by maintaining alignment with the native 4KB allocation format that modern operating systems use.

Because SolidFire volumes do not use any reserve space, you can deploy a volume capacity for the estimated maximum size of the database. You can therefore purchase just enough physical hardware to support the actual space that is consumed by the database. As database space consumption approaches the physical limits of the installed cluster, you can dynamically add nodes to the cluster to increase its physical capacity. This process is completely transparent to the database and does not require downtime or reconfiguration of the operating system or the database.

Furthermore, SolidFire Helix® replication automatically redistributes existing data over the added nodes to provide optimal load balancing of both existing and new data. With this deployment paradigm, you can configure logical storage capacity once for the lifetime of the supported databases rather than using incremental updates that depend on the needs of the database.

1.2 Compression and Deduplication

Each SolidFire node includes a PCIe NVRAM card that serves as a write cache. When a host sends writes, they are divided into 4KB blocks that are immediately compressed, hashed, and stored in the NVRAM of the two storage nodes before an acknowledgement is returned. The resulting value serves as a block ID that determines the block placement and that is randomly distributed across all nodes to create an even load distribution.

The SolidFire deduplication block service identifies blocks that have previously been written based on the block ID. If a block already exists, metadata is updated accordingly, and the duplicate is discarded. The whole process is inline and global to the storage cluster.

The combination of inline compression and global deduplication has the following advantages:

- Reduced repetitive writes to media, prolonging drive life
- Increased system performance by minimizing system resources
- Evenly distributed capacity and performance loads across the system, eliminating hot spots

1.3 Quality of Service (QoS)

NetApp SolidFire storage arrays present performance and capacity as dynamic, independent pools. This feature enables administrators to set the performance requirements for all the databases or tenants that are hosted on the same cluster. The minimum, maximum, and burst control settings in the QoS policy guarantee the required performance and can be dynamically changed anytime. If the SolidFire hardware resources are pushed to their physical limits, more nodes can be added to the existing cluster. SolidFire Helix data distribution automatically redistributes data for optimal load balancing over all hardware resources. This process is transparent to upstream applications.

2 Application Use Cases

NetApp SolidFire can support a wide range of Oracle Database application use cases, including OLTP; decision support systems; database consolidation; data protection; disaster recovery; and development, testing, and quality assurance. This section shows how to identify when these applications are a good fit for SolidFire. It also reviews the innovative benefits of SolidFire to these applications.

2.1 Database Consolidation

NetApp SolidFire provides an optimal storage system for database consolidation. The per-volume QoS controls of SolidFire help individual databases get the I/O throughput that they need without being affected by other databases that run in parallel on the same storage system. With QoS and data reduction efficiencies, you can achieve higher database density within the shared storage infrastructure. The use case of deploying hundreds of individual databases is an excellent fit for SolidFire.

With SolidFire, you can use a single LUN to provision all the data files. Because the maximum size of a SolidFire LUN is 8TB, however, the size of the database must be under 7TB. This configuration can achieve the required performance levels without spreading data files across multiple volumes, controllers, and arrays. SolidFire QoS provides guaranteed performance for each database. You do not have to use complex logical volume management (LVM) configurations to consolidate multiple LUNs or volumes to meet the performance needs of your business.

2.2 Data Protection and Disaster Recovery

SolidFire Helix data protection is a distributed replication algorithm that spreads two redundant copies of data across all drives within the entire cluster. The shared-nothing architecture of SolidFire creates no single point of failure and can absorb multiple failures. The combined storage efficiency and QoS of SolidFire provide a compelling disaster recovery solution that enables the sharing of the same storage resources for disaster recovery and testing and development without performance penalties.

2.3 Development and Testing

Storage Snapshot® copies provide a point-in-time view of the contents of an active file system or storage volume. You can use these Snapshot copies for rapid recovery of corrupted datasets and to create space-efficient copies of datasets for development and testing use cases. The cloning process can be coupled with SolidFire QoS control so that database clones can coexist with the production copies without any performance effects on upstream applications.

The CopyVolume feature of NetApp SolidFire allows you to refresh an existing cloned copy of a database without performing any file system remount operations. In this use case, you frequently refresh a copy of the database by only taking changes from the production copy.

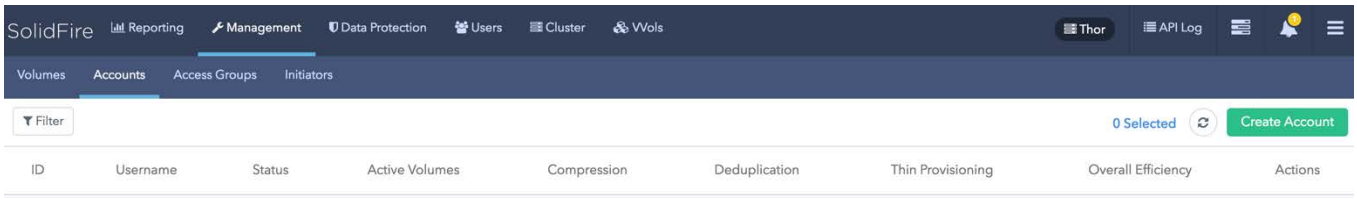
3 Storage Configuration

This section shows how to configure and provision SolidFire volumes for a PostgreSQL database. NetApp recommends that you have all the data files, configuration files, and transaction logs on the SolidFire storage array. NetApp supports presenting the storage in a 4K sector size (native mode) and in a 512-byte sector (512e).

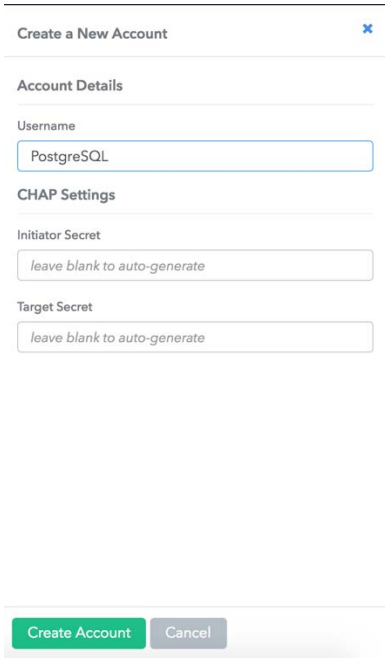
3.1 Create an Account

To create an account, complete the following steps:

1. Log in to the NetApp SolidFire Element® OS web UI.
2. Select Management → Accounts. The Account List window opens.



3. Click Create Accounts. The Create a New Account window opens.



4. Enter a user name.
5. In the CHAP Settings section, fill in the following fields:
   - Initiator secret for CHAP node session authentication
   - Target secret for CHAP node session authentication

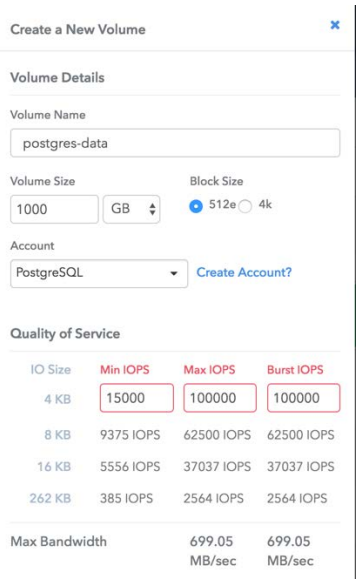
   Note: Leave the credentials field blank if you want passwords to be generated automatically.
6. Click Create Account.

   Note: If an account with the same name exists, you get an error message.

3.2 Create a Volume

To create a volume, complete the following steps:

1. Log in to the Element OS web UI.
2. Select Management → Volumes. The Volumes List window opens.
3. Click Create Volume. The Create a New Volume window opens.



4. Enter the volume name (1 to 64 characters in length). For example, enter the name `postgres-data`.
5. Enter the size of the volume.
6. Click the Account drop-down list and select the account that should have access to the volume. In this case, select PostgreSQL.
7. Set the quality of service values according to your requirements.

   Note: The sliders can be used to adjust the IOPS values, or you can click the number field to enter the desired IOPS values. For `postgres-data`, the following values were chosen for the QoS settings: maximum = 100,000, minimum = 15,000, and burst = 100,000.

8. Click Create Volume.
9. Repeat steps 1 through 7 to create the volume for the transaction log (in this case, '`postgres-log`').

3.3 Create Volume Access Groups

Volume access groups limit connectivity from designated host servers based on a unique identifier, whereas CHAP authentication uses secret keys for unidirectional or bidirectional authentication. In this document, initiator iSCSI qualified names (IQNs) are used to access the volumes.

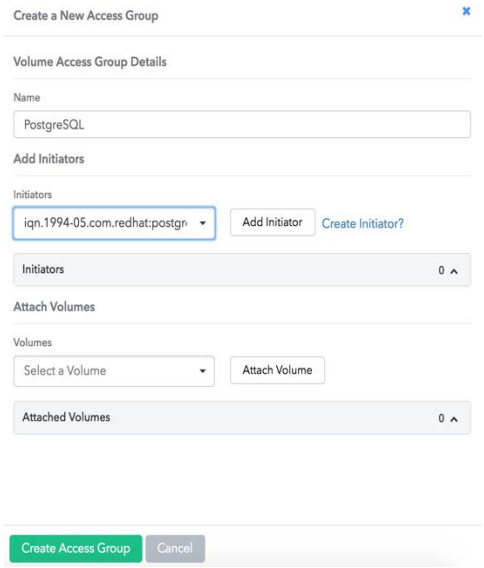
Volume access groups have the following system limits:

- They can have a maximum of 64 IQNs.
- An IQN can belong to only one access group.
- A single volume can belong to a maximum of four access groups.

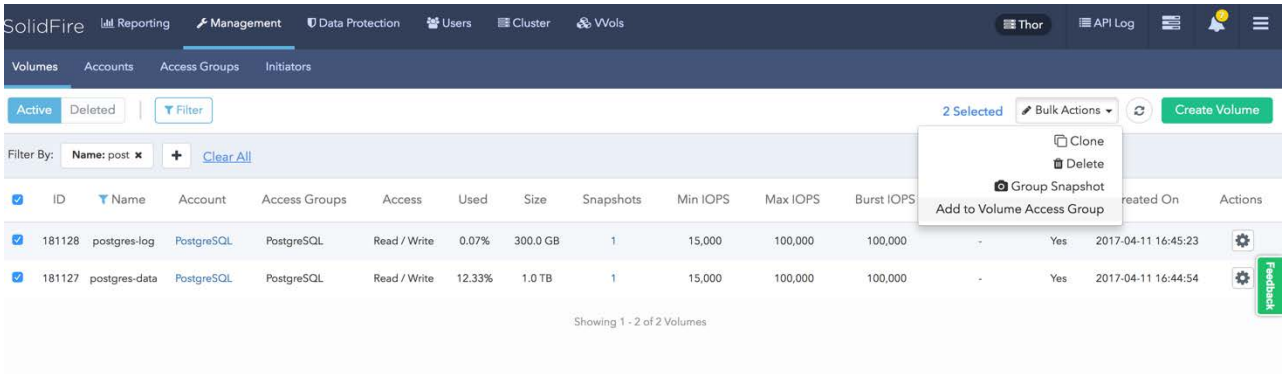
To create volume access groups, complete the following steps:

1. Log in to the Element OS web UI.

2. Select Management → Access Groups. The Access Group window opens.
3. Click Create Access Group. The Create a New Access Group window opens.



4. Enter a name for the volume access group.
5. Select the IQN from the initiator drop-down list or click Create Initiator.
6. Click Create Access Group.
7. Add the volumes to the access group by selecting Management → Volumes.
8. Select the checkbox to the left of each volume.
9. Near the Create Volume button, click the Bulk Actions drop-down list.



10. Click Add to Volume Access Group. The Add to Volume Access Group window opens.
11. Select the previously created volume access group from the drop-down list.
12. Click Add to join the selected volumes to the target group.

4 Operating System Configuration

Red Hat Enterprise Linux 7.x was used for this setup. Alternate distributions can be used, assuming that they have full compatibility with the database software.

4.1 Update Kernel Parameters

Update the kernel parameters for your host operating system to the following values:

```
fs.file-max=65535
kernel.sched_autogroup_enabled=0
kernel.sched_migration_cost_ns=5000000
kernel.shmall=2023636
kernel.shmmax=4144406528
vm.dirty_background_bytes=8388608
vm.dirty_bytes=67108864
vm.zone_reclaim_mode=0
```

4.2 Optimize Network Performance

Consider the following guidelines for optimal network performance:

- Enable jumbo frames for all host network interfaces.
- To isolate the data traffic, configure the interface that is used for the data traffic with a different subnet from the public network.

4.3 Optimize iSCSI Parameters

The Linux iSCSI initiator configuration works with NetApp SolidFire volumes in its default configuration. To maximize system throughput, increase the number of sessions per target (`nr_sessions`) from the default of 1 to 8.

1. Make the following changes to the iSCSI daemon in the `/etc/iscsi/iscsid.conf` file:

```
iscsid.startup = /etc/rc.d/init.d/iscsid force-start
node.startup = automatic
node.leading_login = No
node.session.timeo.replacement_timeout = 120
node.conn[0].timeo.login_timeout = 15
node.conn[0].timeo.logout_timeout = 15
node.conn[0].timeo.noop_out_interval = 5
node.conn[0].timeo.noop_out_timeout = 5
node.session.err_timeo.abort_timeout = 15
node.session.err_timeo.lu_reset_timeout = 30
node.session.err_timeo.tgt_reset_timeout = 30
node.session.initial_login_retry_max = 8
node.session.cmds_max = 128
node.session.queue_depth = 32
node.session.xmit_thread_priority = -20
node.session.iscsi.InitialR2T = No
node.session.iscsi.ImmediateData = Yes
node.session.iscsi.FirstBurstLength = 262144
node.session.iscsi.MaxBurstLength = 16776192
node.conn[0].iscsi.MaxRecvDataSegmentLength = 262144
node.conn[0].iscsi.MaxXmitDataSegmentLength = 0
discovery.sendtargets.iscsi.MaxRecvDataSegmentLength = 32768
node.conn[0].iscsi.HeaderDigest = None
node.session.iscsi.FastAbort = Yes
node.startup = automatic
node.session.nr_sessions = 8
```

2. Make discovery of iSCSI devices persistent over reboots.

```
chkconfig iscsid
```

3. To rescan the new storage volumes, run the following commands:

```
iscsiadm -m discovery -t sendtargets -p <SolidFire SVIP> --op update –n node.session.nr_sessions
–v 2
iscsiadm -m node -L all
```


4.4 Tune the I/O Scheduler

Run the following commands to tune the Linux operating system to take advantage of the performance characteristics of the SolidFire storage system (<devpath> is the device name).

```
echo 0 > /sys/<devpath>/queue/rotational
echo noop > /sys/<devpath>/queue/scheduler
echo 128 > /sys/<devpath>/queue/nr_requests
echo 2 > /sys/<devpath>/queue/rq_affinity
echo 0 > /sys/<devpath>/queue/add_random
```

4.5 Configure the Multipath Driver

Configure the Linux multipath driver (`multipathd`) by making the following changes to the `/etc/multipath.conf` file.

```
defaults {
        user_friendly_names yes
}

devices {
       device {
               vendor "SolidFir"
               product "SSD SAN"
              path_grouping_policy multibus
              path_checker tur
              hardware_handler  "0"
              failback immediate
              rr_weight uniform
              rr_min_io 10
              rr_min_io_rq 10
              features "0"
              no_path_retry 24
              prio const
              }
         }
```

Optionally, you can enable persistent mapping of `/dev/mapper` entries by associating the NetApp SolidFire storage system device's worldwide identifier (WWID) with a specific operating system alias. For this option, make the following additions to the `/etc/multipath.conf` file:

```
multipaths {

multipath {
wwid 36f47acc100000000707a646c000003b1
alias pgdata
}

multipath {
wwid 36f47acc100000000707a646c00000003
alias pglog
}
}
```

4.6 Enable Multipathing

To enable multipathing, run the following command:

```
systemctl enable multipathd.service
```

You can check the status of the multipath daemon with the following command:

```
systemctl list-unit-files|grep multipath
multipathd.service     enabled
```


4.7 Set Up Logical Volume Manager (LVM)

To set up an LVM to stripe the data across multiple SolidFire volume devices, complete the following steps:

1. Create a volume group using the multipath devices `mpatha` and `mpathb`.

```
vgcreate pgdatavg /dev/mapper/mpatha
vgcreate pdlogvg /dev/mapper/mpathb
```

2. Create a logical volume on this volume group.

```
lvcreate -l 100%FREE -n pgdatalv pgdatavg
lvcreate -l 100%FREE -n ploglv pglogvg
```

3. Create an ext4 file system.

```
mkfs.ext4 /dev/pgdatavg/pgdatalv
mkfs.ext4 /dev/pglogvg/ploglv
```

4. Create a database directory path for PostgreSQL and mount the file system.

```
mkdir -p /pgdata/
mkdir –p /pgdata/pg_xlog
mount -t ext4 -o nobarrier,discard,noatime /dev/pgdatavg/pgdatalv /pgdata/
mount -t ext4 -o nobarrier,discard,noatime /dev/pglogvg/ploglv /pgdata/pg_xlog
```

5 PostgreSQL Configuration

5.1 Tune PostgreSQL

PostgreSQL should be installed on a Linux distribution supported by NetApp SolidFire (RHEL 7.x is recommended). For the purposes of this guide, PostgreSQL version 9.5 was installed with the standard installation packages. See the PostgreSQL documentation for information about the latest stable release before you start the installation process. Also, a sample `postgresql.conf` file containing the tuning parameters used for this setup is provided in Appendix A. Some of the parameters might change, depending on the user environment.

5.2 Configure the Write-Ahead Log

PostgreSQL maintains a write-ahead log (WAL) in the `postgresql.conf` file that was enabled for this setup. Enabling the WAL allows you to make crash-consistent backups at the application level without suspending any I/O on the file system. After a successful recovery of the data file system, the log records can be used to replay the transactions.

6 Backup and Recovery Using Storage Snapshots

You can take point-in-time (PIT) snapshots of the volumes that are part of the database. These snapshots can be used to recover the database in the event of data corruption.

6.1 Create Snapshots (Backup)

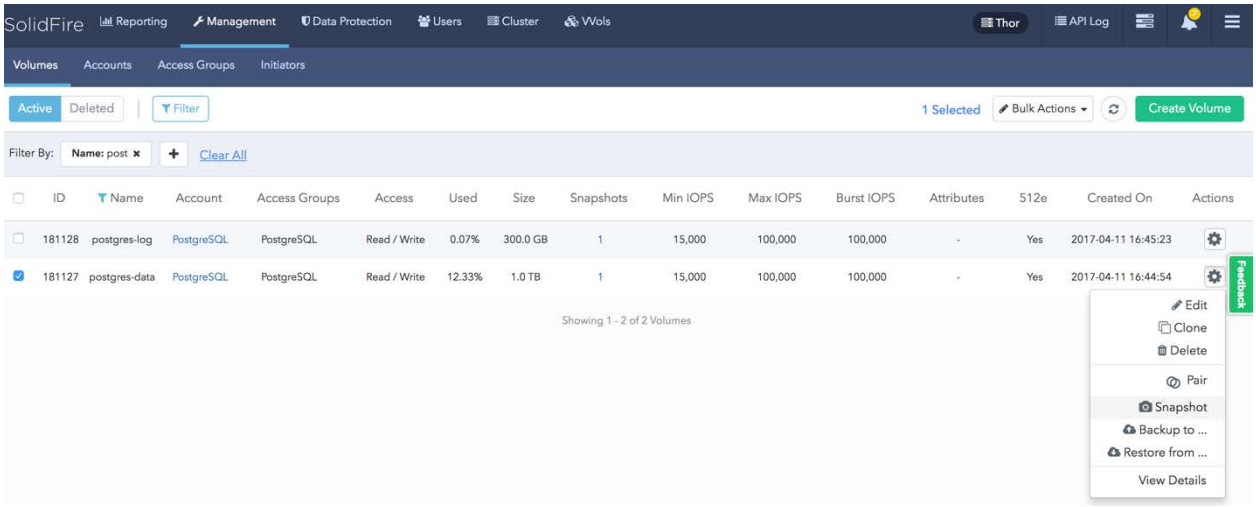
To create Snapshot copies of the database, a start backup is issued at the database to put a mark in the current log files. To do this, start the psql utility and execute the `pg_start_backup` SQL command followed by a snapshot on the storage array.

To create snapshots, complete the following steps:

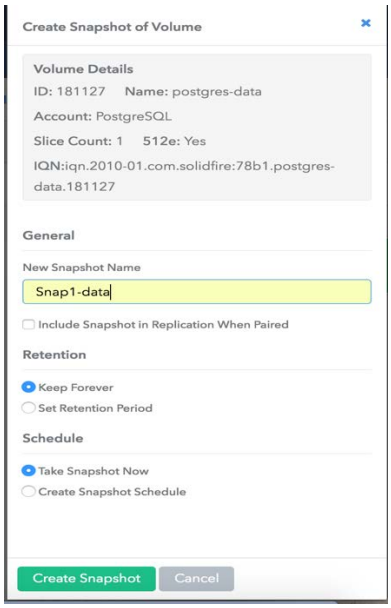
1. Log in to the PostgreSQL server as a postgres user and issue the following commands:


```
psql --u postgres <dbname>
select pg_start_backup('label'); ➔ where label is the name you would like to use to tag the backup
```

2. Log in to the SolidFire Element OS UI.
3. Select Management → Volumes. The Volumes List window opens.
4. Select the volumes that are part of the database, in this case, `postgres-data`.
5. Click the Actions button corresponding to the volume and select Snapshot.



6. In the Create Snapshot of Volume window, enter a name for the snapshot (`Snap1-data` in this example).



7. Set the desired retention time.
8. Select Create Snapshot. Users can schedule the snapshot using the option Create Snapshot Schedule.
9. Log back in to the PostgreSQL server as the postgres user and issue the following command to stop the backup:


```
psql --u postgres <dbname>
select pg_stop_backup;
```

10. In the SolidFire Element OS UI, repeat steps 3 through 8 for the volume `postgres-log`, which was created for transaction logs.

6.2 Restore Snapshots (Recovery)

You can perform database recovery by reverting the volumes on the SolidFire system with a snapshot. As soon as the database instance is started after a snapshot restore on the data volume, the database manager replays the logs from the position where the snapshot was taken. To recover the database, complete the following steps:

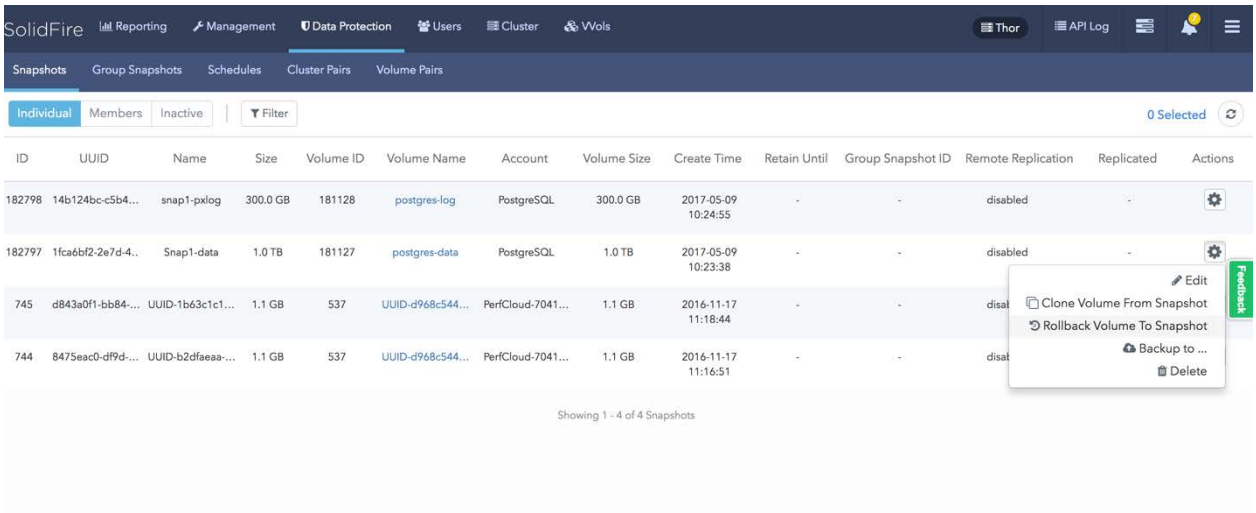
1. Log in to the PostgreSQL server as the root user and issue the following command:

```
systemctl stop postgresql-9.5.service
```

2. Unmount the file system that is part of the database (in this case, `/pgdata`).

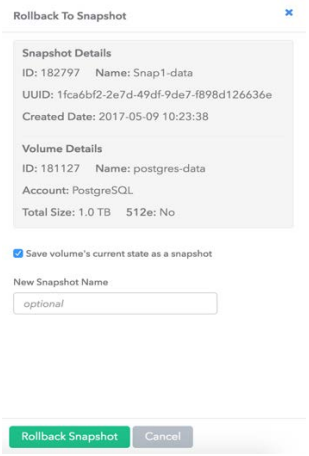
```
unmount /pgdata
```

3. Create a `recovery.conf` file that has the recovery parameters needed to recover the database. A sample `recovery.conf` file is provided in Appendix B.
4. Log in to the SolidFire Element OS UI.
5. Select Data Protection → Snapshots.
6. Click the Actions button for the `postgres-data` volume and select Rollback Volume to Snapshot.



7. The Rollback to Snapshot window opens. Click Rollback to Snapshot.

Note: You can optionally save the current state of the volume so that you can go back to the state before the rollback operation by selecting Save Volume's Current State as a Snapshot.



8. After the volume is successfully reverted using the snapshot, log in to the PostgreSQL instance.
9. Mount the file system back and start PostgreSQL.

```
mount /pgdata
systemctl start postgresql-9.5.service
```

10. PostgreSQL performs a recovery based on the parameters set in the `recovery.conf` file. A sample recovery output is provided in Appendix C.

Note: After a successful recovery is complete, the `recovery.conf` file is renamed to `recovery.done` to avoid another recovery during subsequent restart operations.

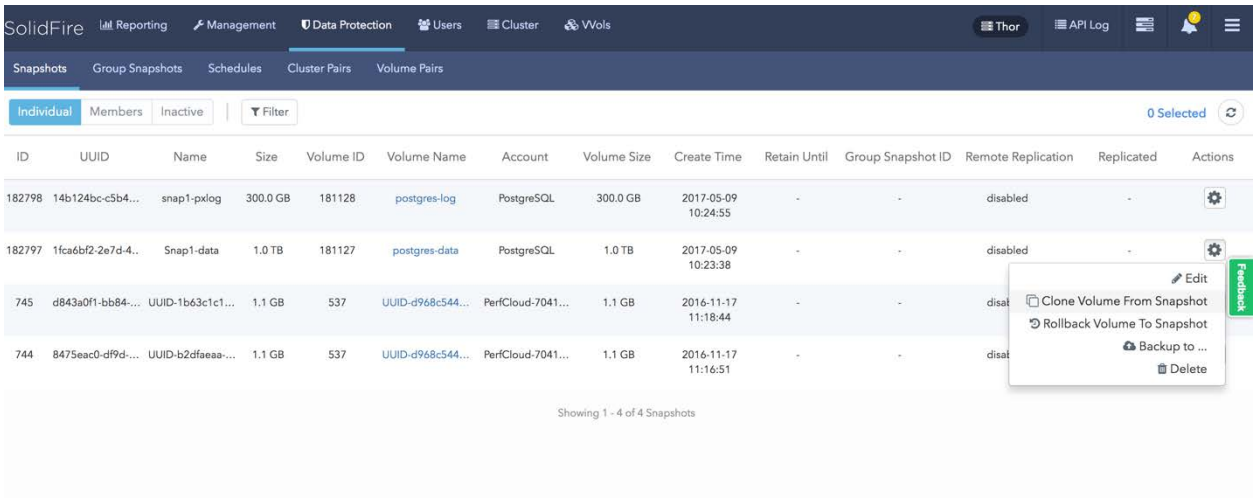
11. Check for any errors in the `pg_log` directory.

7 PostgreSQL Database Cloning

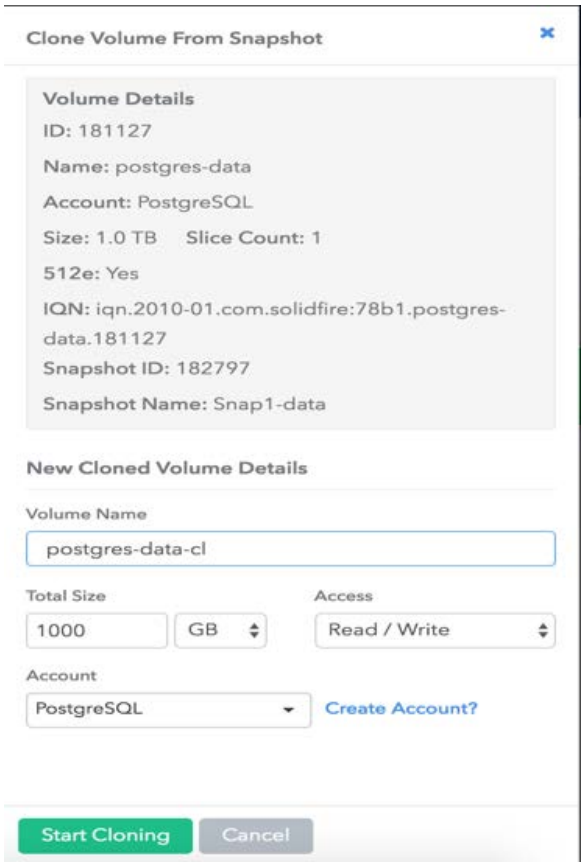
SolidFire volume cloning technology helps database and system administrators deliver a near-instantaneous, space-efficient, point-in-time copy of the production database. The SolidFire volume cloning process is completed very quickly, with virtually no performance effect on the production system. The cloned database is similar to the copy of a source PostgreSQL database instance, and it does not consume extra disk space at the time of creation. To create a copy of a source database to spin up a development or testing copy, complete the following steps. For this procedure, we assume that a server has already been provisioned to map the newly created cloned volumes.

7.1 Clone a Volume

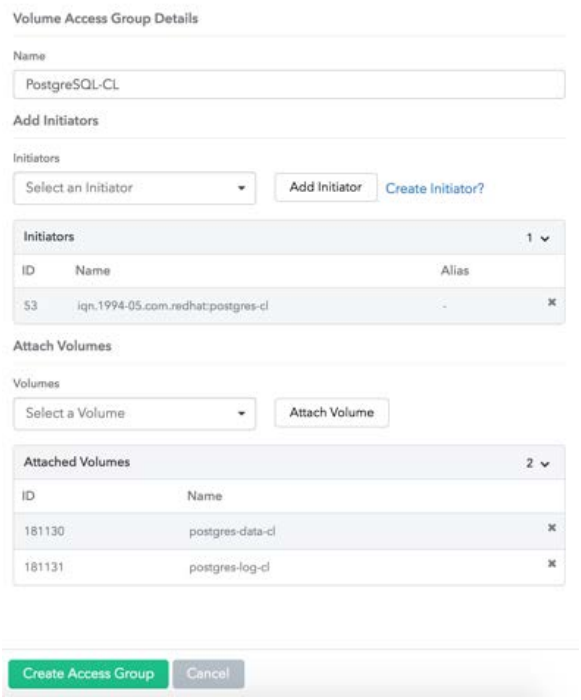
1. Log in to the SolidFire Element OS UI.
2. Select Data Protection → Snapshots. From Actions, select Clone Volume from Snapshot.



3. The Clone Volume from Snapshot windows opens.
4. Enter the volume name `postgres-data-cl`.
5. Click Start Cloning.



6. Repeat steps 2 through 5 for the volume `postgres-log` and name the clone target `postgres-log-cl`.
7. Create a new access group with the initiator from the newly provisioned host and add the cloned volumes to the access group.



8. Log in to the secondary or newly provisioned host.
9. Rescan the iSCSI devices to present the newly cloned volumes.

```
iscsiadm -m discovery -t sendtargets -p <SolidFire SVIP> --op update –n node.session.nr_sessions
–v 2 -> where “n” is the number of paths
iscsiadm -m node -L all
```

10. Create the mount point and mount the file system.

```
mkdir –p /pgdata
mkdir –p /pgdata/pg_xlog
mount -t ext4 -o nobarrier,discard,noatime /dev/pgdatavg/pgdatalv /pgdata/
mount -t ext4 -o nobarrier,discard,noatime /dev/pglogvg/pgloglv /pgdata/pg_xlog
```

11. Rename `recovery.done` to `recovery.conf` so that the database performs a clean recovery after the instance is started. After the instance is started, the file name is changed back to `recovery.done` by the PostgreSQL instance.

```
systemctl start postgresql-9.5.service
```

12. Check for any errors in the `pg_log` directory.

8 Conclusion

NetApp SolidFire provides an ideal storage platform for database applications that use all-flash media to provide performance, capacity, thin provisioning, per-volume QoS, inline deduplication, and compression. This architecture benefits system planners who are deploying and maintaining database applications that are similar to the use cases outlined in this document. You can also contact SolidFire directly at info@solidfire.com.

Appendix A: postgresql.conf File

```
# File Location
data_directory = '/pgdata'
```

```
# Connection
listen_addresses = '*'
port = 5432
max_connections = 1000
# Resource Usage (No WAL)
shared_buffers = 256MB
huge_pages = try
temp_buffers = 32MB
dynamic_shared_memory_type = posix
# - Kernel Resource Usage -
max_files_per_process = 3000
# - Asynchronous Behavior -
effective_io_concurrency = 128
max_worker_processes = 16
# WAL
wal_level = archive
fsync = on
synchronous_commit = on
wal_sync_method = fsync
full_page_writes = on
wal_writer_delay = 200ms
# - Checkpoints -
checkpoint_timeout = 5min
max_wal_size = 1GB
min_wal_size = 80MB
checkpoint_completion_target = 0.5
checkpoint_warning = 30s
# - Archiving -
archive_mode = on
archive_command = 'test ! -f /pgdata/pg_xlog/archive/%f && cp %p /pgdata/pg_xlog/archive/%f'
# Error Reporting
log_destination = 'stderr'
logging_collector = on
log_directory = 'pg_log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_size = 0
log_timezone = 'US/Eastern'
# - Locale and Formatting -
datestyle = 'iso, mdy'
timezone = 'US/Eastern'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
# Lock Management
deadlock_timeout = 2s
max_locks_per_transaction = 128
max_pred_locks_per_transaction = 128
```

Appendix B: A recovery.conf File

```
restore_command = 'cp /pgdata/pg_xlog/archive/%f %p'
recovery_target_timeline = 'latest'
#recovery_target_time = ''
#recovery_target_xid = ''
#recovery_target_inclusive = 'true'
```

Appendix C: Recovery Output from pg_log Directory

```
< 2017-05-11 09:42:30.776 EDT >LOG: database system is shut down
< 2017-05-11 09:45:01.553 EDT >LOG:  database system was shut down at 2017-05-11 09:42:30 EDT
cp: cannot stat ‘/pgdata/pg_xlog/archive/00000004.history’: No such file or directory
< 2017-05-11 09:45:01.559 EDT >LOG:  starting archive recovery
< 2017-05-11 09:45:01.563 EDT >LOG:  restored log file "00000003.history" from archive
cp: cannot stat ‘/pgdata/pg_xlog/archive/000000030000000000000003A’: No such file or directory
```

```
cp: cannot stat ‘/pgdata/pg_xlog/archive/000000020000000000000003A’: No such file or directory
cp: cannot stat ‘/pgdata/pg_xlog/archive/000000010000000000000003A’: No such file or directory
< 2017-05-11 09:45:01.589 EDT >LOG:  invalid primary checkpoint record
< 2017-05-11 09:45:01.595 EDT >LOG:  restored log file "00000003.history" from archive
< 2017-05-11 09:45:01.647 EDT >LOG:  restored log file "000000030000000000000039" from archive
< 2017-05-11 09:45:01.742 EDT >LOG:  using previous checkpoint record at 0/39000028
< 2017-05-11 09:45:01.751 EDT >LOG:  consistent recovery state reached at 0/39000098
< 2017-05-11 09:45:01.751 EDT >LOG:  redo starts at 0/39000098
cp: cannot stat ‘/pgdata/pg_xlog/archive/000000030000000000000003A’: No such file or directory
< 2017-05-11 09:45:01.757 EDT >LOG:  redo done at 0/39000098
< 2017-05-11 09:45:01.798 EDT >LOG:  restored log file "000000030000000000000039" from archive
cp: cannot stat ‘/pgdata/pg_xlog/archive/00000004.history’: No such file or directory
< 2017-05-11 09:45:01.902 EDT >LOG:  selected new timeline ID: 4
< 2017-05-11 09:45:01.909 EDT >LOG:  restored log file "00000003.history" from archive
< 2017-05-11 09:45:02.025 EDT >LOG:  archive recovery complete
< 2017-05-11 09:45:02.130 EDT >LOG:  MultiXact member wraparound protections are now enabled
< 2017-05-11 09:45:02.135 EDT >LOG:  autovacuum launcher started
< 2017-05-11 09:45:02.136 EDT >LOG:  database system is ready to accept connections
```

Where to Find Additional Information

To learn more about the information described in this document, refer to the following documents and/or websites:

- Configuring SolidFire on Linux for Element OS
  https://fieldportal.netapp.com/content/468085

Version History

Version	Date	Document Version History
Version 1.0	June 2017	Initial document creation
Version 1.1	November 2017	Changed the "References" section to "Where to Find Additional Information." Minor edits.