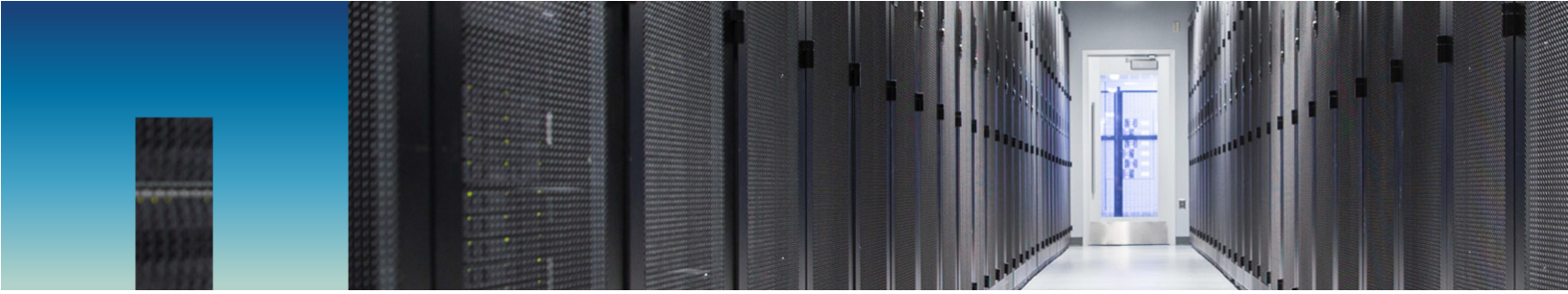


Technical Report

MySQL Best Practices on NetApp SolidFire

Bobby Oommen, NetApp
November 2017 | TR-4605

Abstract

MySQL is widely used for many online applications, from global social networking sites and massive e-commerce systems to SMB hosting systems containing thousands of database instances. Managing the sprawl of commodity hardware that is associated with large-scale MySQL deployments can be a massive operational undertaking. Supporting various service and performance levels creates a complex hardware environment in which one size might not fit all. This document describes using the capabilities of NetApp® SolidFire® storage to solve business problems associated with streamlining MySQL operations.

NetApp®

TABLE OF CONTENTS

1 Introduction 4

1.1 Thin Provisioning 4

1.2 Compression and Deduplication 4

1.3 Quality of Service 5

2 Application Use Cases 5

2.1 Database Consolidation 5

2.2 Data Protection and Disaster Recovery 5

2.3 Development and Testing 5

3 Storage Configuration 5

3.1 Create an Account 6

3.2 Create a Volume 6

3.3 Create Volume Access Groups 7

4 Operating System Configuration 9

4.1 Add Additional Packages 9

4.2 Update Kernel Parameters 9

4.3 Optimize Network Performance 10

4.4 Optimize iSCSI Parameters 10

4.5 Tune the I/O Scheduler 10

4.6 Configure the Multipath Driver 11

4.7 Enable Multipathing 11

4.8 Set Up a Logical Volume Manager (LVM) 11

5 MySQL Configuration 12

5.1 Tune MySQL Instance 12

6 Backup and Recovery by Using Storage Snapshots 12

6.1 Creating Snapshots (Backup) 12

6.2 Restore Snapshots (Recovery) 13

7 MySQL Database Cloning 14

7.1 Clone a Volume 15

8 Conclusion 16

Appendix A: Master my.cnf 17
Appendix B: Slave my.cnf 17
Appendix C: Slave Status 18
Where to Find Additional Information 19
Version History 19

1 Introduction

NetApp SolidFire storage systems were born out of some of the largest cloud infrastructures in the world. They are designed to serve next-generation data center needs, including scaling with multitenancy, set-and-forget management, and guaranteed performance. Adopting the 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.

NetApp SolidFire storage systems have the following features to 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, which increases effective capacity by consuming less space. This feature increases efficiency and reduces overhead by using the smallest allocation possible while 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 create 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 to creating an even distribution of load.

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

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 QoS guarantee required performance levels 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 database application use cases. This section shows how to identify when application use cases are an ideal fit for SolidFire and reviews the innovative benefits of SolidFire to these applications.

2.1 Database Consolidation

SolidFire provides an optimal storage system for database consolidation. The per-volume QoS controls of SolidFire help individual databases get the I/O throughput 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 by having hundreds of MySQL databases on a storage cluster.

In addition, a slave copy of the MySQL database can be provisioned on the same storage array as the master. Database administrators have full control of each storage volume on which the database resides and can perform all maintenance operations, including setting the QoS for each database copy. Administrators can use REST APIs to achieve full automation and make storage management simpler.

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 cluster. The shared-nothing architecture from 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 snapshots provide a point-in-time view of the contents of an active file system or storage volume. You can use these snapshots 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 controls so that database clones can coexist with production copies without any performance effect on the upstream applications. The SolidFire cloning process can be used to create a slave copy of the database by using an application-consistent snapshot (see the section “MySQL Database Cloning”).

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 can frequently refresh a copy of the database by only taking changes from the production copy.

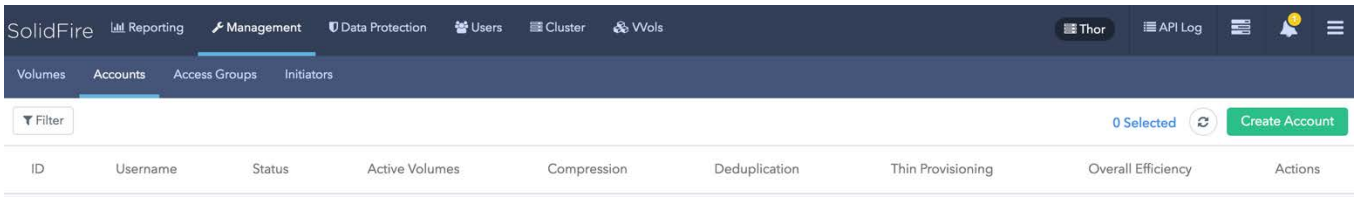
3 Storage Configuration

This section shows how to configure SolidFire volumes to support a MySQL database application. NetApp recommends that you have all the MySQL database components on the SolidFire storage array.

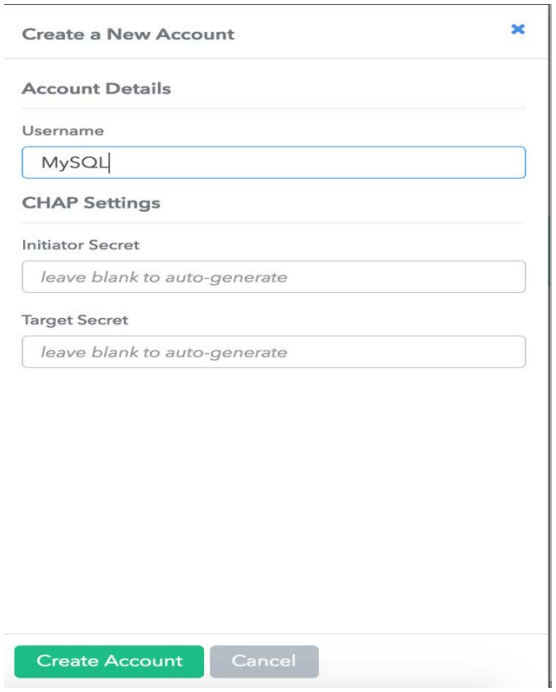
NetApp supports presenting the storage in a 4K sector size (native mode) and in a traditional 512-byte sector size (512e). Tests conducted in NetApp labs have demonstrated that there is no performance effect for choosing emulation mode as long as there is no partition misalignment at the host level.

3.1 Create an Account

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



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



4. Enter a user name.
5. In the CHAP Settings section, enter the following information:
   a. Initiator secret for CHAP node session authentication
   b. Target secret for CHAP node session authentication

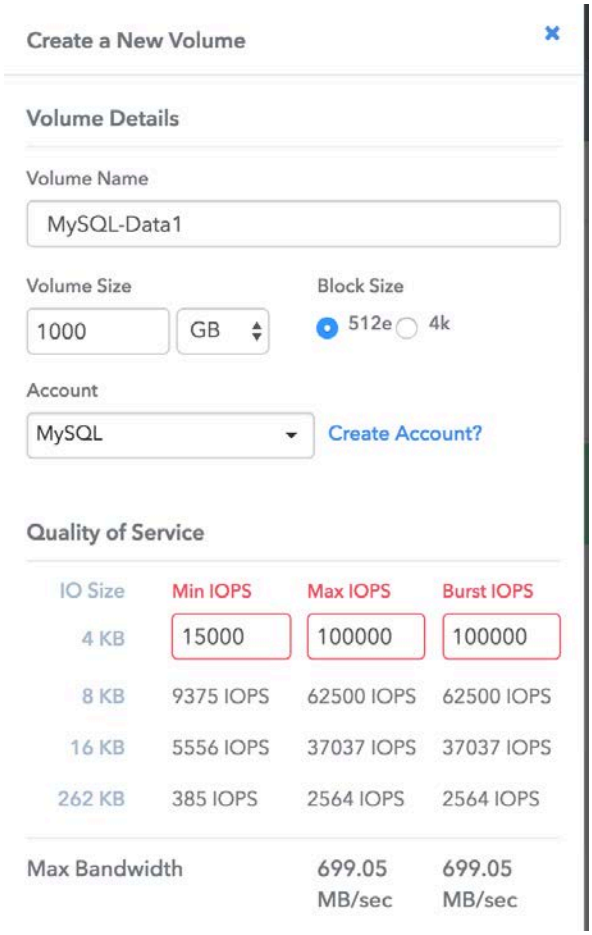
   Note: Leave the credentials field blank if you want the passwords to be generated automatically.
6. Click Create Account.

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

3.2 Create a Volume

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 `MySQL-Data1`.
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 MySQL.
7. Set the Quality of Service Settings values according to your requirements.

   Note: The sliders may be used to adjust the IOPS values, or click the number field to enter the desired IOPS values. For MySQL-Data1, use the following values: maximum = 100,000, minimum = 15,000, and burst = 100,000.
8. Click Create Volume.
9. Repeat steps 1 through 7 for all volumes that are part of the MySQL database.

3.3 Create Volume Access Groups

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

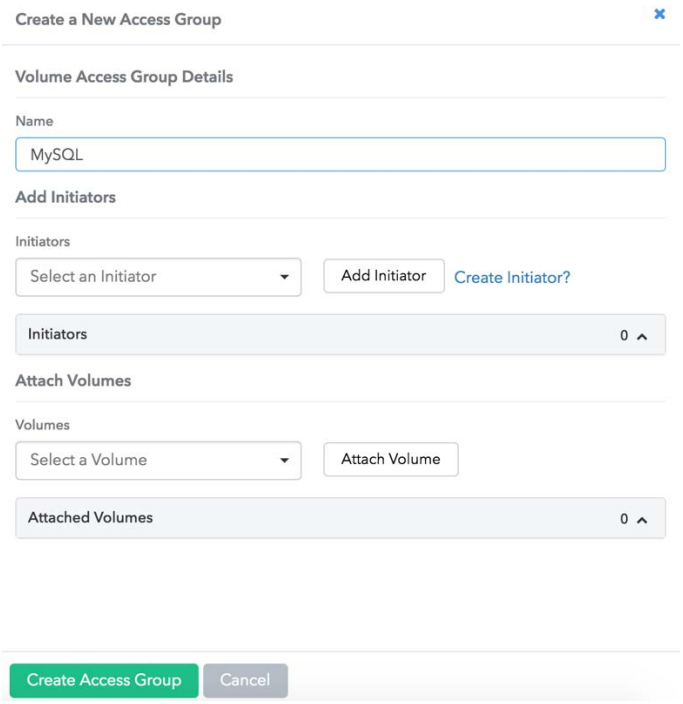
Volume access groups have the following system limits:

- They can have a maximum of 64 IQNs.
- An IQN can belong to only one access group.

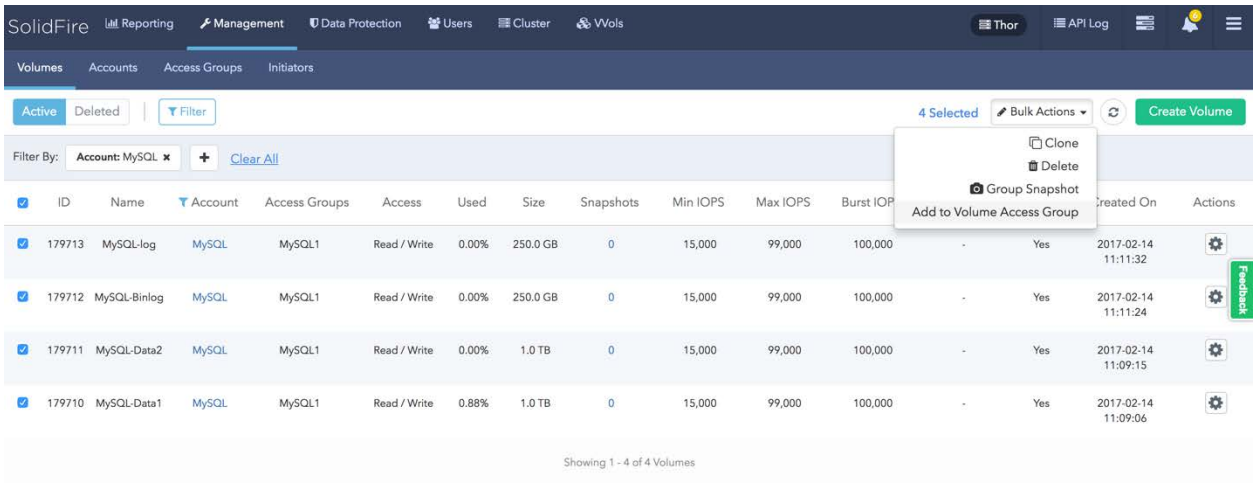
- A single volume can belong to a maximum of four access groups.

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, select the Bulk Actions drop-down list.
10. Select Add to Volume Access Group.



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

The MySQL database volumes are now listed as part of the selected volume access group and are ready to be mapped to the host operating system.

Note: For this configuration, two SolidFire volumes were chosen for the MySQL database, and an LVM configuration was used to stripe data across both volumes. For a web-scale deployment that involves hundreds of databases, NetApp recommends that you configure one volume for individual databases and that you control performance through the QoS settings.

4 Operating System Configuration

The guidelines in this document apply to Red Hat Enterprise Linux 7.x distributions of MySQL Enterprise edition. Alternate distributions can be used, assuming that they have full compatibility with MySQL database software.

4.1 Add Additional Packages

After installing the base operating system, you might need to update the operating system to meet MySQL installation requirements. Refer to the latest MySQL documentation to meet these requirements, depending on the release.

4.2 Update Kernel Parameters

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

```
vm.dirty_ratio = 15
vm.dirty_background_ratio = 5
vm.swappiness = 0
net.core.somaxconn = 4096
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_keepalive_time = 120
net.ipv4.tcp_max_syn_backlog = 4096
```

4.3 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 MySQL data traffic with a different subnet from the public network.

4.4 Optimize iSCSI Parameters

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

1. Make the following changes for 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
```

4.5 Tune the I/O Scheduler

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
```

4.6 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 ID (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 mysql-disk1
}

multipath {
wwid 36f47acc100000000707a646c00000003
alias mysql-disk2
}
}
```

4.7 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
```

4.8 Set Up a Logical Volume Manager (LVM)

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

1. Element OS software can achieve up to 700MB for single volume throughput for large block I/O. For this setup, two SolidFire volumes (`mpatha` and `mpathb`) were used for the data directory to stripe the data across the SolidFire devices. To create a volume group using the multipath devices `mpatha` and `mpathb`, run the following command:

```
vgcreate mysqldatavg /dev/mapper/mpatha /dev/mapper/mpathb
```


2. Create a logical volume on this volume group.

```
lvcreate -l 100%FREE -n mysqldatalv mysqldatavg
```

3. Create an ext4 file system.

```
mkfs.ext4 /dev/mysqldatavg/mysqldatalv
```

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

```
mkdir -p /var/lib/mysql
mount -t ext4 -o nobarrier,discard,noatime /dev/mysqldatavg/mysqldatalv /var/lib/mysql
```

Before you install MySQL, you should set appropriate permissions so that the MySQL user can access the devices on the SolidFire system. You can set permissions on multipath devices by creating a `udev` rule file that allows appropriate access to the devices.

5 MySQL Configuration

5.1 Tune MySQL Instance

The instance tunings parameters were applied in the `my.cnf` file. These are guidelines and might change depending upon customer requirements. A sample `my.cnf` file is presented in Appendix A.

6 Backup and Recovery by Using Storage Snapshots

You can take point-in-time snapshots on the SolidFire array for the volumes that are part of the MySQL user and system databases. The application-consistent snapshots can be used to perform database recovery if data corruption or media failure occurs. SolidFire supports having multiple volumes for the MySQL database and confirms that all volumes that are part of the database have the same consistency point during the group snapshot.

6.1 Creating Snapshots (Backup)

1. Log in to the MySQL server instance.
2. To check the BIN log and position, run the following command:

```
SHOW MASTER STATUS;
```

Note: This step is used in step 18 of the section “Clone a Volume” to initiate slave replication.

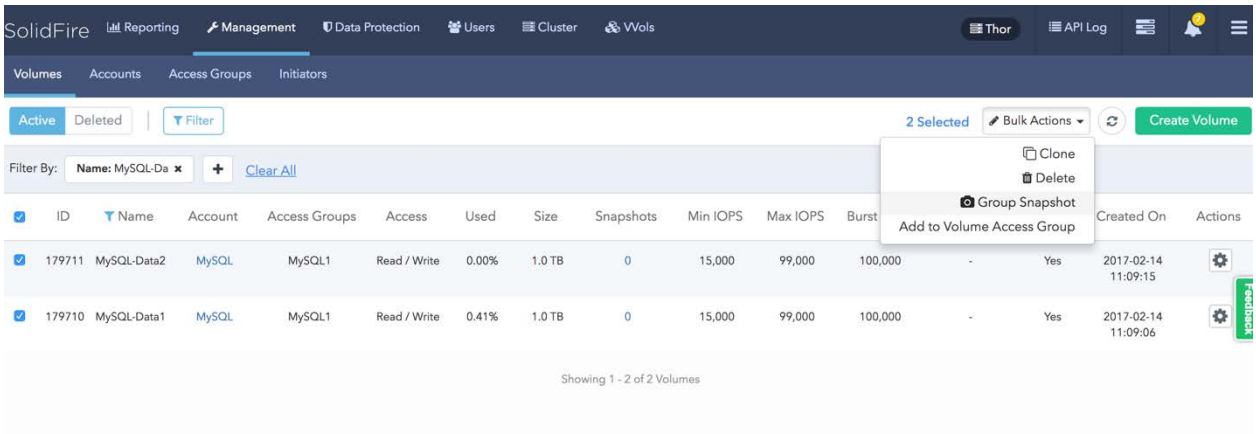
3. To put the MySQL database into a consistent state to take an application-consistent storage snapshot, run the following commands. These commands make sure that the database is in a consistent state before a snapshot is taken at the SolidFire storage array. Step 2 provides the binary location entered in this step.

```
FLUSH TABLES WITH READ LOCK;
PURGE BINARY LOGS TO LOCATION ‘XXXX’;
```

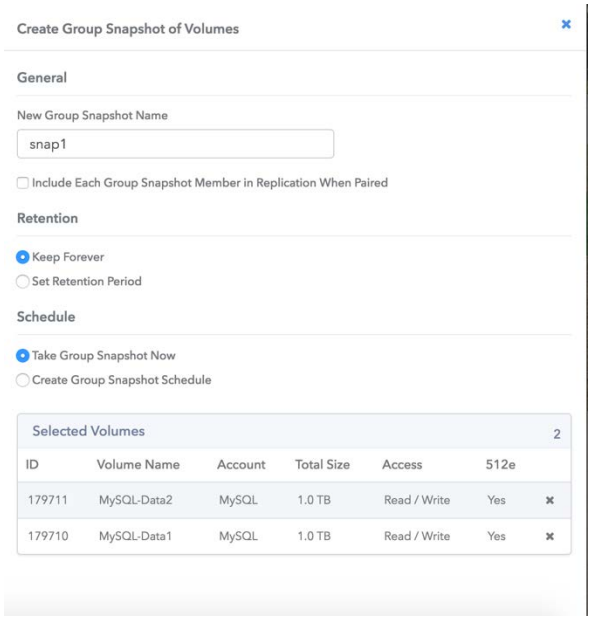
4. Run the command to flush the file system buffers. This step flushes all changes to storage.

```
sync
```

5. Log in to the Element OS web UI.
6. Select Management → Volumes. The Volumes List window opens.
7. Select the volumes that are part of the database.
8. Select Bulk Actions → Group Snapshot.



9. In the Create Group Snapshot of Volumes window, enter a name for the snapshot (`Snap1` in this case).
10. Set the desired retention time.
11. Select Take Group Snapshot Now or Create Group Snapshot Schedule.
12. Scroll to the bottom and click Create Group Snapshot.



13. Unlock the tables by connecting to the MySQL server instance.

```
UNLOCK TABLES;
```

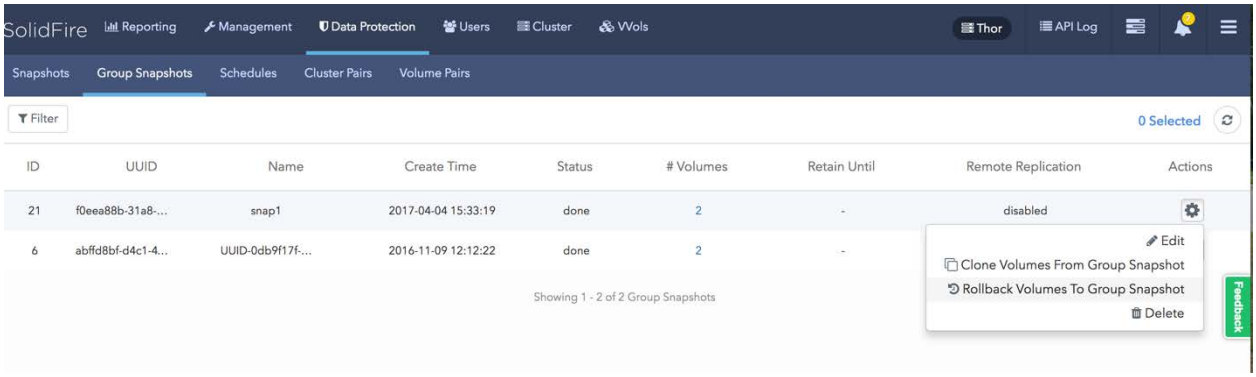
6.2 Restore Snapshots (Recovery)

You can perform database recovery by reverting the volumes on the SolidFire system with a snapshot. To recover the database, complete the following steps:

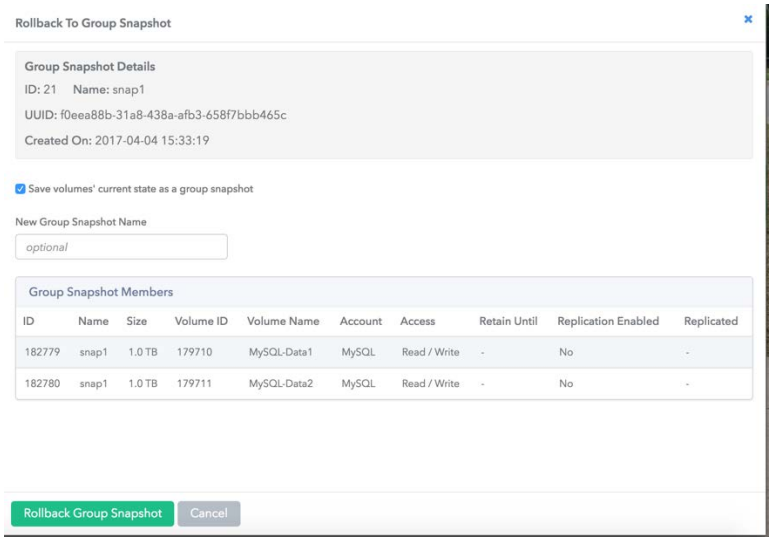
1. Log in to the server on which MySQL is running, stop the instance, and then unmount the file system that is part of the database.

```
systemctl stop mysqld
unmount /var/lib/mysql
```

2. Log in to the SolidFire Element OS UI.
3. Select Data Protection→Group Snapshots. The Group Snapshot list opens.



4. Select the application snapshot that was taken in section 6.
   a. From the Actions button, select Rollback Volumes to Group Snapshot.
   b. The Rollback to Group Snapshot window opens.



5. Click Rollback Group Snapshot.
6. After the volume is reverted successfully, log in to the MySQL instance.
7. Mount the file systems and start the MySQL instance.

```
mount /var/lib/mysql
systemctl start mysqld
```

8. Check for any errors in the logs.

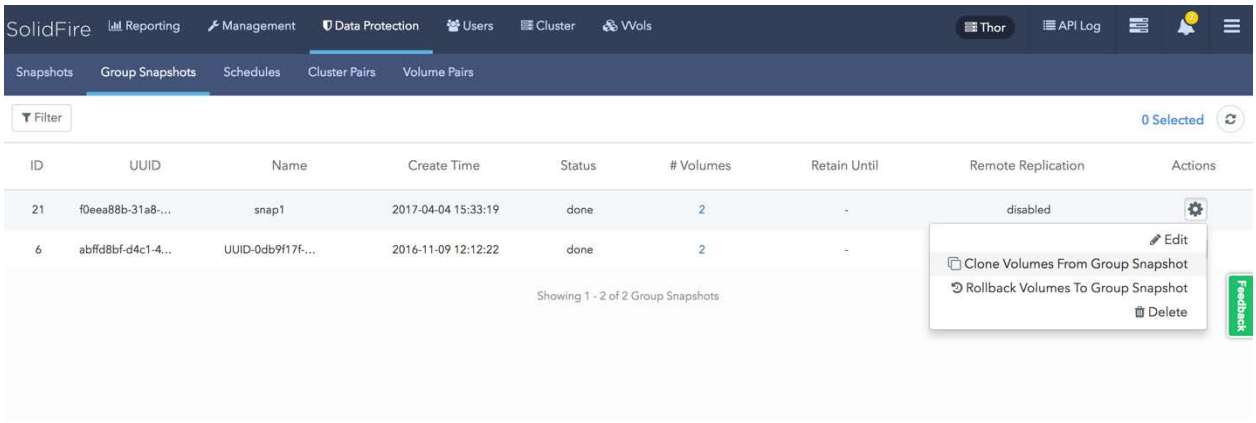
7 MySQL Database Cloning

NetApp SolidFire volume cloning technology helps database and system administrators deliver a near-instantaneous, space-efficient, point-in-time copy of the MySQL master instance to create any number of slave copies. 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 master and does not

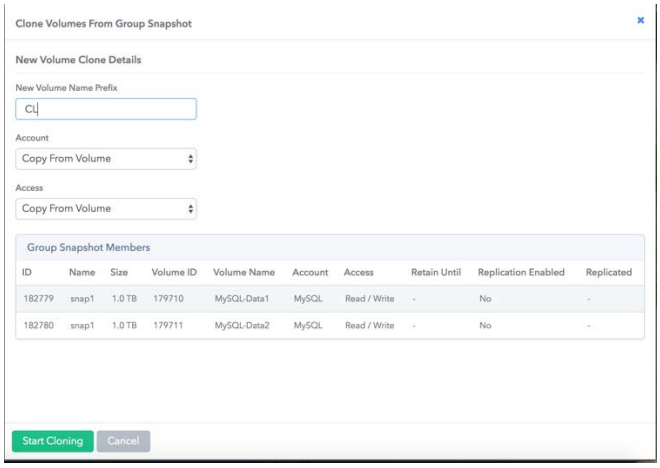
consume extra disk space at the time of creation. Users can connect to the newly created slave copies after they are activated and work completely independently of the master copy.

7.1 Clone a Volume

1. Log in to the SolidFire Element OS UI.
2. Select Data Protection→Group Snapshots and then select the snapshot that was taken in the section "Creating Snapshots (Backup)."



3. From the Actions button, select Clone Volumes from Group Snapshot.
4. The Clone Volumes from Group Snapshot window opens.
5. Enter a prefix (CL for this example).
6. Give the account and access information, depending on what access you need for the clone volumes.



7. Click Start Cloning.
8. After the clone process is complete, add the newly created volumes to the volume access group of the slave MySQL host server.
9. Log in to the slave server.
10. Rescan the iSCSI devices to present the newly cloned volume.
11. Mount the file system.

```
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
mount -t xfs -o nobarrier,discard,noatime /dev/mysqlvg/mysqllv /var/lib/mysql
```

12. Edit the `/etc/my.cnf` file to change the `server-id` from master to slave. In this case, the server ID was changed to 2 from 1. A sample slave `my.cnf` file is provided in Appendix B.

```
Server-id = 2
```

13. Remove the `auto.cnf` file that has the server UUID referencing to the original master server. This file is under the data directory (for example, `/var/lib/mysql/auto.cnf`).
14. Start the MySQL instance.

```
systemctl start mysql
```

15. To verify that the server ID is different on the slave server, run the following command in the MySQL shell.

```
show variables like 'server_id';
```

16. Create a replication user on the master server and grant the privilege.

```
CREATE USER repl@localhost;
GRANT REPLICATION SLAVE ON *.* TO repl@slavehost IDENTIFIED BY 'password';
```

17. Check the log file and log position on the master provided in step 18.

```
SHOW MASTER STATUS;
```

18. To initiate the replication between the master and the slave, log in to the MySQL shell and run the following commands. The bin log information is captured from section 6.1, step 4.

```
 CHANGE MASTER TO
    -> MASTER_HOST = 'master-ip', → IP address of the master server
    -> MASTER_USER = 'repl', → Replication user
    -> MASTER_PASSWORD = 'password',→ Replication password
    -> MASTER_LOG_FILE = 'bin-log-file-name', → Bin log file name from
    -> MASTER_LOG_POS = bin-log-position-number; → Bin log position during the snapshot

START SLAVE;
```

19. To check the status of the slave, run the following command. A sample output is given in Appendix A.

```
SHOW SLAVE STATUS;
```

8 Conclusion

SolidFire provides an optimal storage system for database applications that use all-flash media, improving performance and providing significant storage savings. This architecture benefits system planners deploying and maintaining MySQL databases, as can be seen in the use cases outlined in this document. For additional information, you can contact SolidFire directly at info@solidfire.com.

Appendix A: Master my.cnf

```
[mysqld]
server-id      = 1
# INNODB CONFIG
innodb_buffer_pool_size = 4G
innodb_log_buffer_size = 64M
innodb_log_file_size = 1G
innodb_file_per_table = 1
innodb_flush_method = O_DIRECT
innodb_write_io_threads = 64
innodb_read_io_threads = 64
innodb_lock_wait_timeout = 5
innodb_adaptive_flushing = 1
innodb_io_capacity = 4000
innodb_io_capacity_max = 8000
innodb_thread_concurrency = 0

character-set-server=utf8
wait_timeout = 86400
user           = mysql
pid-file       = /var/run/mysqld/mysqld.pid
socket         = /var/run/mysqld/mysqld.sock
port           = 3306
basedir        = /usr
datadir        = /var/lib/mysql
tmpdir         = /tmp
skip-external-locking
max_allowed_packet     = 1024M
thread_stack           = 192K
thread_cache_size       = 1000
max_connections        = 2048
query_cache_limit      = 1M
query_cache_size        = 16M
general_log_file        = /var/log/mysql.log
general_log             = 1
long_query_time = 2
log-bin=mysql-bin
expire_logs_days       = 5
max_binlog_size         = 1G
server_id              = 1
binlog_do_db           = include_database_name
binlog_ignore_db       = include_database_name
binlog_format          = STATEMENT

[mysqldump]
quick
quote-names
max_allowed_packet     = 16M
```

Appendix B: Slave my.cnf

```
[mysqld]

server-id      = 2
# INNODB CONFIG
innodb_buffer_pool_size = 4G
innodb_log_buffer_size = 64M
innodb_log_file_size = 1G
innodb_file_per_table = 1
innodb_flush_method = O_DIRECT
innodb_write_io_threads = 64
innodb_read_io_threads = 64
innodb_lock_wait_timeout = 5
innodb_adaptive_flushing = 1
innodb_io_capacity = 4000
innodb_io_capacity_max = 8000
innodb_thread_concurrency = 0
```

```
character-set-server=utf8
wait_timeout = 86400
user           = mysql
pid-file       = /var/run/mysqld/mysqld.pid
socket         = /var/run/mysqld/mysqld.sock
port           = 3306
basedir        = /usr
datadir        = /var/lib/mysql
tmpdir         = /tmp
skip-external-locking
#key_buffer            = 16M
max_allowed_packet     = 1024M
thread_stack           = 192K
thread_cache_size       = 1000
max_connections        = 2048
query_cache_limit      = 1M
query_cache_size        = 16M
general_log_file        = /var/log/mysql.log
general_log             = 1
long_query_time = 2
log-bin=mysql-bin
expire_logs_days       = 5
max_binlog_size         = 1G
server_id              = 2
binlog_do_db           = include_database_name
binlog_ignore_db       = include_database_name
binlog_format          = STATEMENT

[mysqldump]
quick
quote-names
max_allowed_packet     = 16M
```

Appendix C: Slave Status

```
show slave status
    -> \G
*************************** 1. row ***************************
               Slave_IO_State: Queueing master event to the relay log
                  Master_Host: 172.27.152.48
                  Master_User: repl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000003
          Read_Master_Log_Pos: 2956
               Relay_Log_File: mysql-cl-relay-bin.000002
                Relay_Log_Pos: 320
        Relay_Master_Log_File: mysql-bin.000003
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 2956
              Relay_Log_Space: 530
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
```

```
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 10496bb5-f2c3-11e6-8f91-005056bb5fd7
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
```

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 2014	Initial document creation
Version 2.0	April 2017	Refresh for Element OS 9.x
Version 2.1	November 2017	Changed the "References" section to "Where to Find Additional Information." Minor edits.