Configuring a SecureSpan Disaster Recovery System

Introduction

Disaster Recovery (DR) is a critical component of a highly available environment. A typical SecureSpan Cluster is configured in a single data center. In the event that the data center is incapacitated by a disaster (e.g. earthquake, flood, terrorist attack, etc) there needs to be a mechanism whereby the functionality of the SecureSpan is brought back online as quickly as possible.

There are several possible solutions to a DR configuration, ranging from keeping a cold spare with recent backups available that can be manually implemented as required to a remotely located system that is configured to take over the responsibilities of the primary SecureSpan cluster within a minimum time frame.

This paper outlines a procedure for configuring a single node in a remote location that can serve as a Disaster Recovery (DR) system. The idea is to create a database node that replicates the database from the secondary node of the cluster (i.e. chain replication) and is thus continually up to date with respect to the data store. The DR node is configured to be in a disabled state so that it will not write to the database and cause a collision. When the DR node is activated several manual steps will be required to be run.

Note: This tutorial applies to version 5.2 of the SecureSpan Gateway.

Assumptions

The following assumptions are made:

- An operating SecureSpan Gateway cluster has already been configured with two database nodes
- All systems have been mapped in the /etc/hosts files if not configured in the DNS
- All ancillary systems also have a redundant configuration in the DR environment for the SecureSpan Gateway to access with the same mappings as the live cluster, including application servers, LDAP, JMS, JDBC, SNMP, SMTP, etc.

Configuration

Throughout these instructions the two nodes in the cluster will be referred to as ssg1 and ssg2, with ssg1 being the database node hosting the primary database. The DR node being configured will be referred to as ssg-dr.

Preparing the secondary DB node on ssg2

Log on to ssg2 as root and run */opt/SecureSpan/Appliance/bin/add_slave_user.sh* to add a new ACL to the MySQL database for the DR system to replicate:

```
[root@ssg2 ~]# /opt/SecureSpan/Appliance/bin/add_slave_user.sh -v

Gathering information for SLAVE user
Enter hostname or IP for the SLAVE: ssg-dr.l7tech.com
Enter replication user: [repluser] repluser
Enter replication password: [replpass] replpass
Enter MySQL root user: [root] root
Enter MySQL root password: [] 7layer

Checking configuration of running MySQL...
MySQL appears to be properly configured with server_id=2
Do you want to continue? [Y] Y
Granting slave permissions to repluser@ssg-dr.l7tech.com
Done.
[root@ssg2 ~]#
```

Configuring the DR node replication

1. Log on to ssg-dr as ssgconfig then access the privileged shell using option **3) Use a privileged shell (root)** to log in as root.
2. Edit /etc/my.cnf and add a line to set server-id=3:

```
.
.
.
# Uncommment log-bin and log-slave-update if a clustered
# db server
#log-bin=/var/lib/mysql/ssgbin-log
#log-slave-update
# uncomment the next item on 1st db master server
#server-id=1
# uncomment the next item on 2nd db master servers
#server-id=2
# This is a DR node
server-id=3
relay-log = /var/lib/mysql/ssgrelay-bin
relay-log-index = /var/lib/mysql/ssgrelay-bin.index
relay-log-info-file = /var/lib/mysql/ssgrelay-bin.info
.
```

```
.
.
```

3. Restart the mysqld service:

```
[root@ssg-dr ~]# service mysqld restart
Stopping MySQL:                                            [  OK  ]
Starting MySQL:                                            [  OK  ]
[root@ssg-dr ~]#
```

4. Use scp to copy the create_DR_slave.sh script (see Appendix 1 for listing) to the ssgconfig user's home directory. Set ownership to root:root and permissions to 700 then move to /usr/local/bin:

```
[root@ssg-dr ~]# chown root:root ~ssgconfig/create_DR_slave.sh
[root@ssg-dr ~]# chmod 700 ~ssgconfig/create_DR_slave.sh
[root@ssg-dr ~]# mv ~ssgconfig/create_DR_slave.sh /usr/local/bin
[root@ssg-dr ~]#
```

5. Run the *create_DR_slave.sh* script to configure the slave database:

```
[root@ssg-dr ~]# /usr/local/bin/create_DR_slave.sh -v
Enter hostname or IP for the secondary DB node in the cluster: ssg2.l7tech.com
Enter replication user: [repluser] repluser
Enter replication password: [replpass] replpass
Enter monitor user: [monitor] monitor
Enter monitor password: [monitorpass] monitorpass
Enter MySQL root user: [root] root
Enter MySQL root password: [] 7layer
Do you want to clone a database from ssg2.l7tech.com (yes or no)? [yes] yes
Enter name of database to clone: [ssg] ssg
--> MASTER = ssg2.l7tech.com
--> DBUSER = repluser
--> DBPWD = replpass
--> ROOT = root
--> ROOT_PWD = 7layer
--> CLONE_DB = yes
--> DB = ssg
--> Stopping slave
--> File = ssgbin-log.000002
--> Position = 62095412
--> Changing MASTER settings
--> Creating database: ssg
--> Copying database from ssg2.l7tech.com
--> Starting slave
--> Confirming slave startup
--> Slave_IO_Running = Yes
--> Slave_SQL_Running = Yes
Slave successfully created
--> Getting password for user gateway
--> Granting access for 'gateway'@'localhost'
--> Granting access for 'gateway'@'%'
--> Granting access for 'gateway'@'localhost.localdomain'
--> Granting monitor rights for monitor@localhost
[root@ssg-dr ~]#
```

Note: If the database is very large the clone command may time out. If this happens access the root shell on ssg2 and manually dump the database with:

```
# mysqldump --master-data=1 -r ssgdump.sql ssg
```

then copy ssgsump.sql to ssg-dr and load it using:

```
# mysqladmin stop-slave
# mysqladmin create ssg
# mysql ssg < ssgdump.sql
# mysqladmin start-slave
# mysql -e "SHOW SLAVE STATUS\G" | grep Running
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
#
```

6. Exit the root shell then select option **2) Display SecureSpan Gateway configuration menu** followed by option **3) Configure the SecureSpan Gateway** to configure the DR gateway to use the localhost database but remain in a disabled state:

```
[root@ssg-dr ~]# exit
Welcome to the SecureSpan Gateway

This user account allows you to configure the appliance
What would you like to do?

 1) Configure networking
 2) Display SecureSpan Gateway configuration menu
 3) Use a privileged shell (root)
 4) Change the Master Passphrase
 5) Display Remote Management configuration menu
 7) Display Enterprise Service Manager configuration menu
 8) Display Patch Management menu
 R) Reboot the SSG appliance (apply the new configuration)
 X) Exit (no reboot)

Please make a selection: 2

This menu allows you to configure the SecureSpan Gateway application
What would you like to do?

 1) Upgrade the SecureSpan Gateway database
 2) Create a new SecureSpan Gateway database
 3) Configure the SecureSpan Gateway
 4) Change the SecureSpan Gateway cluster passphrase
 5) Delete the SecureSpan Gateway
 6) Display the current SecureSpan Gateway configuration
 7) Manage SecureSpan Gateway status
 X) Exit menu

Please make a selection: 3

----------------------------------------------------------------------
Configure SecureSpan Gateway
----------------------------------------------------------------------
At any time type "quit" to quit.


-----------------------
Set Up the SSG Database
-----------------------
At any time type "quit" to quit.
Press "<" to go to the previous step.

This step lets you create or set up a connection to the SSG database.

Enter the database hostname.

Database Host [localhost]: localhost

Enter the database port.

Database Port [3306]: 3306
```

```
Enter the database name.

Database Name [ssg]: ssg

Enter the database user.

Database Username [gateway]: gateway

Enter the database password.

Database Password: 7layer
Confirm Database Password: 7layer

--------------------------------
Set Up the SSG Failover Database
--------------------------------
At any time type "quit" to quit.
Press "<" to go to the previous step.

This step lets you create or set up a connection to the SSG failover
database.

This step is optional, enter "Yes" to continue or "No" to skip.

Configure Database Failover Connection? [No]: No

----------------------
Set Up the SSG Cluster
----------------------
At any time type "quit" to quit.
Press "<" to go to the previous step.

This step lets you set up the SSG cluster.

Enter the cluster passphrase (6-128 characters).

Cluster Passphrase: 7layer
Confirm Cluster Passphrase: 7layer

-------------------
Set Up the SSG Node
-------------------
At any time type "quit" to quit.
Press "<" to go to the previous step.

This step lets you set up the SSG node.

Enable or disable the node.

Enabled [Yes]: No
---------------------
Configuration Summary
---------------------
Press < to go to the previous step, type "quit" to quit.

The following configuration will be applied:

  Database Connection
    Database Host = localhost
    Database Port = 3306
    Database Name = ssg
    Database Username = gateway

  Node Configuration
    Enabled = No

Press [Enter] to continue.
Please wait while the configuration is applied ...
```

```
---------------------
Configuration Results
---------------------

The configuration was successfully applied.

Press [Enter] to continue
```

Monitoring the Replication State

1. Use scp to copy the monitor_replication.sh script (see Appendix 2 for listing) to the ssgconfig user's home directory.
2. Log on to ssg-dr as ssgconfig then access the privileged shell using option **3) Use a privileged shell (root)**. Set ownership of monitor_replication.sh to root:root and permissions to 700 then move it to /usr/local/bin:

```
[root@ssg-dr ~]# chown root:root ~ssgconfig/monitor_replication.sh
[root@ssg-dr ~]# chmod 700 ~ssgconfig/monitor_replication.sh
[root@ssg-dr ~]# mv ~ssgconfig/monitor_replication.sh /usr/local/bin
[root@ssg-dr ~]#
```

3. Edit /usr/local/bin/monitor_replication.sh and set the configurable settings, including:
   - MONUSER (must be same as was used during create_DR_slave.sh)
   - MONPWD (must be same as was used during create_DR_slave.sh)
   - NOTIFY (default is probably OK)
   - NOTIFY_SMTP
     - Configure exim per notes in script
     - NOTIFY_TO (required)
     - NOTIFY_CC (optional)
     - NOTIFY_BCC (optional)
     - MTA_FLAG (see notes for value. Default is probably correct)
   - NOTIFY_SNMP
     - SNMP_HOST (required)
     - COMMUNITY (required)
     - OID_ERROR (default is probably OK)
     - OID_INFO (default is probably OK)
   - VERBOSE (set to yes for testing, no for production)

Note: If using SNMP for notification be sure that net-snmp-utils is installed. See notes in script.

4. Test monitor_replication.sh:

```
[root@ssg-dr ~]# monitor_replication.sh
Master_Host = ssg2.l7tech.com
Slave_IO_Running = Yes
Slave_SQL_Running = Yes
Master_Log_File = ssgbin-log.000005

Slave is functioning properly.
```

```
[root@ssg-dr ~]# mysqladmin stop-slave
Slave stopped
[root@ssg-dr ~]# monitor_replication.sh
Master_Host = ssg2.l7tech.com
Slave_IO_Running = No
Slave_SQL_Running = No
Master_Log_File = ssgbin-log.000005

WARNING: Slave is not functioning properly
 --- This *should* have been sent by SMTP or SNMP ---
Sending notification by SNMP trap
[root@ssg-dr ~]# mysqladmin start-slave
Slave started
[root@ssg-dr ~]# monitor_replication.sh
Master_Host = ssg2.l7tech.com
Slave_IO_Running = Yes
Slave_SQL_Running = Yes
Master_Log_File = ssgbin-log.000005

Slave is functioning properly.

[root@ssg-dr ~]#
```

5. Confirm that notification was properly received for the fail test. If everything looks correct edit */usr/local/bin/monitor_replication.sh* and set **VERBOSE='no'** then add */usr/local/bin/monitor_replication.sh* to crontab. See notes in the script for more information:

```
[root@ssg-dr ~]# crontab -e
# Monitor DR database replication
0 * * * * /usr/local/bin/monitor_replication.sh
```

Activation

Activating the failover node involves logging in to the DR node, stopping replication then activating the node. Once the node is activated traffic can be routed to it.

Note: It is very important that the slave on DR be disabled in the event that the database on ssg2 inadvertently comes back up.

1. Log on to ssg-dr as ssgconfig then access the privileged shell using option **3) Use a privileged shell (root)**.
2. Run `mysqladmin stop-slave` to stop the database from slaving the secondary node:

```
# mysqladmin stop-slave
Slave stopped
# mysql -e "SHOW SLAVE STATUS\G" | grep Running
           Slave_IO_Running: No
          Slave_SQL_Running: No
#
```

3. Optional: Run crontab -e to comment out the replication monitoring:

```
# Monitor DR database replication
#0 * * * * /usr/local/bin/monitor_replication.sh
```

4. Type 'exit' to log out of the root shell and return to the ssgconfig menu.

5. From the main ssgconfig menu select option **2) Display SecureSpan Gateway Configuration menu.**
6. Select option **3) Configure the SecureSpan Gateway.**
7. Select option **4) Node Configuration.**
8. Enter **Yes** to enable the node.
9. Enter **S) Save and exit** to save the new setting followed by the **[Enter]** key to continue until you are back at the SecureSpan Gateway application menu:

```
----------------------------------------------------------------------
Configure SecureSpan Gateway
----------------------------------------------------------------------
At any time type "quit" to quit.

Select option to configure:
1) Database Connection
2) Configure Database Failover Connection
3) Cluster Configuration
4) Node Configuration

S) Save and exit.
X) Exit without saving.

Select: 4
Enable or disable the node.

Enabled [No]: Yes

Select option to configure:
1) Database Connection
2) Configure Database Failover Connection
3) Cluster Configuration
4) Node Configuration

S) Save and exit.
X) Exit without saving.

Select: S
---------------------
Configuration Summary
---------------------
Press < to go to the previous step, type "quit" to quit.

The following configuration will be applied:

  Database Connection
    Database Host = localhost
    Database Port = 3306
    Database Name = ssg
    Database Username = gateway

  Node Configuration
    Enabled = Yes

Press [Enter] to continue.
Please wait while the configuration is applied ...
---------------------
Configuration Results
---------------------

The configuration was successfully applied.

Press [Enter] to continue
```

10. Wait about a minute for the SecureSpan Gateway to start then select option **7) Manage SecureSpan Gateway status** to check state of the SecureSpan startup. If it is not

running exit back to the menu and wait, then select option 7 again. Once it is **RUNNING** you can proceed to the next step..

```
This menu allows you to configure the SecureSpan Gateway application
What would you like to do?

 1) Upgrade the SecureSpan Gateway database
 2) Create a new SecureSpan Gateway database
 3) Configure the SecureSpan Gateway
 4) Change the SecureSpan Gateway cluster passphrase
 5) Delete the SecureSpan Gateway
 6) Display the current SecureSpan Gateway configuration
 7) Manage SecureSpan Gateway status
 X) Exit menu

Please make a selection: 7

----------------------------------------------------------------------
SecureSpan Gateway Status
----------------------------------------------------------------------

Configuration:
    Node Status = RUNNING
    Node Status Timestamp = 2010-02-16 09:47:49
    Node Status Since = 2010-02-16 09:45:00

Press [Enter] to continue.
```

11. Exit from the ssgconfig menu and confirm that the SecureSpan Manager can connect to the DR node directly.
12. Configure the network routing infrastructure to route messages to the DR system.

Restoring the Cluster

The following steps for restoring the cluster after the primary site is recovered assumes that a recent back up of the cluster before the disaster happened is available.

1. Restore the cluster to its last known good state, including database replication between the database nodes.
2. Schedule a maintenance window for failback to the cluster. *Do not continue these steps until the maintenance window arrives!*
3. Log in to each cluster node as ssgconfig, select **Option 3) Use a privileged shell (root)** to access the root user then stop the ssg service and confirm the listening ports:

```
[root@ssg1 ~]# service ssg stop
Shutting down Gateway Services:                            [  OK  ]
[root@ssg1 ~]# netstat -tnl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address              Foreign Address            State
tcp        0      0 0.0.0.0:3306               0.0.0.0:*                  LISTEN
tcp        0      0 0.0.0.0:22                 0.0.0.0:*                  LISTEN
[root@ssg1 ~]#
.
.
.
[root@ssg2 ~]# service ssg stop
Shutting down Gateway Services:                            [  OK  ]
[root@ssg2 ~]# netstat -tnl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address              Foreign Address            State
tcp        0      0 0.0.0.0:3306               0.0.0.0:*                  LISTEN
tcp        0      0 0.0.0.0:22                  0.0.0.0:*                   LISTEN
[root@ssg2 ~]#
```

4. Confirm that database replication is functioning on each node. If it is not take steps to ensure replication is properly functioning:

```
[root@ssg1 ~]# mysql -e "SHOW SLAVE STATUS\G" | grep Running
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
[root@ssg1 ~]#
.
.
.
[root@ssg2 ~]# mysql -e "SHOW SLAVE STATUS\G" | grep Running
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
[root@ssg2 ~]#
```

5. Log in to the DR node as ssgconfig, select **Option 3) Use a privileged shell (root)** then dump the database and copy to primary DB node (ssg1) using scp:

```
[root@ssg-dr ~]# mysqldump -r ssg-dr.sql ssg
[root@ssg-dr ~]# scp ssg-dr.sql ssgconfig@ssg1.l7tech.com:
The authenticity of host 'ssg1.l7tech.com (10.7.50.201)' can't be established.
RSA key fingerprint is 7a:43:f1:ed:a3:23:e1:70:ea:dc:3b:f7:c0:b8:c7:b2.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ssg1.l7tech.com,10.7.50.201' (RSA) to the list of known hosts.
ssgconfig@ssg1.l7tech.com's password: <password>
ssg-dr.sql                                                         100% 1052KB   1.0MB/s   00:00
[root@ssg-dr ~]#
```

6. On ssg1 load the ssg-dr.sql database dump into the ssg database:

```
[root@ssg1 ~]# mysql ssg < ~ssgconfig/ssg-dr.sql
[root@ssg1 ~]#
```

7. Start the SSG service on both cluster nodes:

```
[root@ssg1 ~]# service ssg start
Starting Gateway Services:                                  [  OK  ]
[root@ssg1 ~]#
.
.
.
[root@ssg2 ~]# service ssg start
Starting Gateway Services:                                  [  OK  ]
[root@ssg2 ~]#
```

8. Confirm that replication monitoring is working on both nodes and that the SecureSpan Manager can connect. Once verified redirect traffic to the cluster.
9. Disable the SSG on the DR node through the ssgconfig menu and reconfigure the DR node to replicate from ssg2.

Discussion

This method of configuring a Disaster Recovery node has the advantage of being fully up to date and ready to go live with minimal effort. The impact on the production cluster nodes is limited to replication reading from the secondary database node, which should be minimal.

Another option would be to retrieve a periodic backup image from the primary node using wget and running it through ssgrestore.sh on an automated basis. This has the disadvantage of possibly being out of date by hours. Retrieving a full backup image would have the additional disadvantage of being a larger performance impact on the primary database node. There are also implications of OS level setting (IP addresses, etc) that may need to be addressed.

It is important to consider the issue of load capacity when using a single DR node. If the normal operating level of traffic is greater than a single node can handle then either some form of traffic shaping will be required in the DR networking infrastructure or the DR system will need to be configured as a cluster itself. A DR cluster would need to be limited to a single database node for initial takeover, with both processing nodes in a disabled state until activated. If there is a chance that the DR cluster will be running for an extended period of time it would be possible to configure the DR cluster with a replicated database. If a DR cluster is required contact Layer 7 Pro Services for help with configuration.

Appendix 1: Listing of create_DR_slave.sh

```
#!/bin/sh
#
# Script to automatically configure a Disaster Recovery slave system
#
# Call with -v for verbose output
#
# Note: You *MUST* configure MASTER, DBUSER, DBPWD and DB below
#
# Note: You *MUST* configure SELECT, LOCK TABLES and RELOAD privilege on the master:
#
# GRANT SELECT, LOCK TABLES, RELOAD ON *.* TO 'DBUSER'@'SLAVE' IDENTIFIED BY 'DBPWD';
#
# Where:
#   SLAVE is the IP address or host name of the slave system where this script is installed
#   DBUSER is the database user configured below
#   DBPWD is the database password configured below
#
# Jay MacDonald - v1 - 20100216 - modified create_slave.sh

################################################# Start configurable settings

# Set the defaults up
DBUSER="repluser"
DBPWD="replpass"
MONUSER="monitor"
MONPWD="monitorpass"
ROOT="root"
ROOT_PWD="7layer"

CLONE_DB="yes"
DB="ssg"

################################################# End configurable settings

clean_up() {
        rm -f /tmp/mb_*.$$
        exit $1
}

verbose() {
        if test $VERBOSE == yes  ; then
```

```
                echo "--> $1"
        fi
}

mysql_fail() {
        cat <<-EOM

                ==> $1

                Message: `cat /tmp/mb_error.$$`

                Refer to http://dev.mysql.com/doc/refman/4.1/en/error-messages-client.html
                for more information.

        EOM
        clean_up 1
}

evaluate_result() {
        eval `echo "$1" | \
                sed 's/^  *//' | \
                sed 's/: \(.*\)/="\1"/' | \
                grep -v '=""$' | \
                grep '='`
}

set_grants() {
        echo "--> Granting access for $1"
        CMD="GRANT USAGE ON *.* TO $1 IDENTIFIED BY PASSWORD '$PASSWORD'"
        RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

        if test $? -ne 0; then
                mysql_fail "Error granting USAGE for $1"
        fi

        CMD="GRANT ALL PRIVILEGES ON ssg.* TO $1"
        RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

        if test $? -ne 0; then
                mysql_fail "Error granting PRIVILEGES for $1"
        fi
}

# Check if -v set
if test "$1" == "-v" ; then
        VERBOSE="yes"
else
        VERBOSE="no"
        echo "--> For verbose output run with -v"
        echo ""
fi

################
# Get the settings

echo -n "Enter hostname or IP for the secondary DB node in the cluster: "
read -e MASTER

if test -z $MASTER ; then
        echo "You must enter a value for the secondary DB node in the cluster"
        clean_up 1
fi

echo -n "Enter replication user: [$DBUSER] "
read -e
if test $REPLY ; then DBUSER=$REPLY ; fi

echo -n "Enter replication password: [$DBPWD] "
```

```
read -e
if test $REPLY ; then DBPWD=$REPLY ; fi

echo -n "Enter monitor user: [$MONUSER] "
read -e
if test $REPLY ; then MONUSER=$REPLY ; fi

echo -n "Enter monitor password: [$MONPWD] "
read -e
if test $REPLY ; then MONPWD=$REPLY ; fi

echo -n "Enter MySQL root user: [$ROOT] "
read -e
if test $REPLY ; then ROOT=$REPLY ; fi

echo -n "Enter MySQL root password: [$ROOT_PWD] "
read -e
if test $REPLY ; then ROOT_PWD=$REPLY ; fi

echo -n "Do you want to clone a database from $MASTER (yes or no)? [yes] "
read -e
if test $REPLY ; then CLONE_DB=$REPLY ; fi

if test "$CLONE_DB" != "yes" -a "$CLONE_DB" != "no" ; then
        echo "Must answer 'yes' or 'no' (type whole word)"
        clean_up 1
fi

if test "$CLONE_DB" == "yes" ; then
        echo -n "Enter name of database to clone: [$DB] "
        read -e
        if test $REPLY ; then DB=$REPLY ; fi
fi

verbose "MASTER = $MASTER"
verbose "DBUSER = $DBUSER"
verbose "DBPWD = $DBPWD"
verbose "MONUSER = $MONUSER"
verbose "MONPWD = $MONPWD"
verbose "ROOT = $ROOT"
verbose "ROOT_PWD = $ROOT_PWD"
verbose "CLONE_DB = $CLONE_DB"
verbose "DB = $DB"

# Massage the user and password args for mysql on slave.
if test $ROOT_PWD ; then ROOT_PWD="-p$ROOT_PWD" ; fi
if test $ROOT ; then ROOT="-u$ROOT" ; fi

# Set commands for slave and master mysql calls
SLAVE_MYSQL="/usr/bin/mysql $ROOT $ROOT_PWD"
MASTER_MYSQL="/usr/bin/mysql -h$MASTER -u$DBUSER -p$DBPWD"

# Initialize values
File=""
Position=0
Slave_IO_Running="No"
Slave_SQL_Running="No"

################
# Set the master configuration

# Stop the slave
verbose "Stopping slave"
CMD="STOP SLAVE\G"
RESULT=`$SLAVE_MYSQL -e "$CMD\G" 2>/tmp/mb_error.$$`

if test $? -ne 0; then
```

```
        mysql_fail "Error stopping slave"
fi

# Query the MASTER STATUS from the master database
CMD="SHOW MASTER STATUS\G"
RESULT=`$MASTER_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

if test $? -ne 0 ; then
        mysql_fail "Error getting master status"
fi

# Get the useful information from SHOW MASTER STATUS\G and convert to variables
evaluate_result "$RESULT"

verbose "File = $File"
verbose "Position = $Position"

# Change the MASTER settings in slave
verbose "Changing MASTER settings"
CMD="CHANGE MASTER TO MASTER_HOST='$MASTER',
        MASTER_USER='$DBUSER',
        MASTER_PASSWORD='$DBPWD',
        MASTER_PORT=3307,
        MASTER_CONNECT_RETRY=100,
        MASTER_LOG_FILE='$File',
        MASTER_LOG_POS=$Position;"

RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

if test $? -ne 0; then
        mysql_fail "Error changing master settings"
fi

################
# Clone the database if requested

if test "$CLONE_DB" == "yes" ; then

        ################
        # Check if the database already exists and drop if so
        CMD="SHOW DATABASES"
        RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$ | grep "^${DB}$"`

        if test $RESULT ; then
                # Database exists, so drop it
                # This is a drastic procedure. Confirm before doing anything...
                echo ""
                echo "W A R N I N G"
                echo "  About to drop the $DB database on localhost"
                echo "  and copy from $MASTER"
                echo ""
                echo -n "Are you sure you want to do this? [N] "
                read -e

                if test -z $REPLY ; then
                        REPLY="N"
                fi

                if test $REPLY != "y" -a $REPLY != "Y" -a $REPLY != "yes" ; then
                        clean_up 0
                fi


                ################
                # Drop the database

                verbose "Dropping database"
                CMD="DROP DATABASE $DB"
                RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

                if test $? -ne 0 ; then
                        mysql_fail "Error dropping database"
                fi
        fi

        ################
        # Create the database

        verbose "Creating database: $DB"
        CMD="CREATE DATABASE $DB"
        RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

        if test $? -ne 0 ; then
                mysql_fail "Error creating database"
        fi

        ################
        # pipe in the database

        verbose "Copying database from $MASTER"
        mysqldump -h $MASTER -u $DBUSER -p$DBPWD --master-data=1 $DB | $SLAVE_MYSQL $DB
fi

################
# Start the slave

verbose "Starting slave"
CMD="START SLAVE\G"
RESULT=`$SLAVE_MYSQL -e "$CMD\G" 2>/tmp/mb_error.$$`

if test $? -ne 0; then
        mysql_fail "Error starting slave"
fi

# Give the slave a chance to start
sleep 1

################
# Confirm the slave settings on slave
verbose "Confirming slave startup"
CMD="SHOW SLAVE STATUS\G"
RESULT=`$SLAVE_MYSQL -e "$CMD\G" 2>/tmp/mb_error.$$`

if test $? -ne 0; then
        mysql_fail "Error querying slave status"
fi

evaluate_result "$RESULT"

verbose "Slave_IO_Running = $Slave_IO_Running"
verbose "Slave_SQL_Running = $Slave_SQL_Running"

if test $Slave_IO_Running == "Yes" -a $Slave_SQL_Running == "Yes" ; then
        echo "Slave successfully created"
else        cat <<-EOM

                ==> Error: Slave not started

                Confirm your settings in $0

                Result of SLAVE STATUS:

                $RESULT

        EOM
        clean_up 1
fi

# Assume password is the same for all gateway users
verbose "Getting password for user gateway"
CMD="SELECT Password FROM mysql.user WHERE User='gateway' AND Host='%'"
PASSWORD=`$MASTER_MYSQL -e "$CMD" | tail -1`

set_grants "'gateway'@'localhost'"
set_grants "'gateway'@'%'"
set_grants "'gateway'@'localhost.localdomain'"

echo "--> Granting monitor rights for $MONUSER@localhost"
CMD="GRANT REPLICATION CLIENT ON *.* TO $MONUSER@localhost IDENTIFIED BY '$MONPWD'"
RESULT=`$SLAVE_MYSQL -e "$CMD" 2>/tmp/mb_error.$$`

if test $? -ne 0; then
        mysql_fail "Error granting REPLICATION CLIENT for $MONUSER@localhost"
fi
```

Appendix 2: Listing of monitor_replication.sh

```
#!/bin/bash
#
# Script to monitor replication on a DR node
# Periodically run this on the DR node from crontab.
#
# Note: You *MUST* configure NOTIFY_TO below
#
# Note: You *MUST* configure REPLICATION CLIENT privilege:
#
# GRANT REPLICATION CLIENT ON *.* TO 'MONUSER'@'localhost' IDENTIFIED BY 'MONPWD';
#
# That should have been done by create_DR_slave.sh. Make sure MONUSER and MONPWD
# are the same that were set when create_DR_slave.sh was run.
#
# Note: An MTA daemon should not be running on a SecureSpan Gateway. To enable
# sending of notification by SMTP you must configure a smart relay host in the
# MTA configuration. Appliances running on RHEL 4 (SSG v5.0 and previous) have
# sendmail installed. Appliances running on RHEL 5 (SSG v5.1 and later) have
# exim installed. The following sections outline steps to take for each
# version.
#
# Using SNMP Trap for Alerts
# ~~~~~~~~~~~~~~~~~~~~~~~~~~
#
# To send snmp trap alerts the net-snmp-utils rpm package must be installed.
# This is not a default in the SecureSpan appliances (as of v5.2). Use scp to
# copy the rpm file to the ssgconfig user's home directory then run:
#   # rpm -Uvh ~ssgconfig/net-snmp-utils-<version info>.rpm
#
# There are two possible traps that may be sent: ERROR and INFO. The OID for
# ERROR is 1.3.6.1.4.1.17304.7.3.50 and indicates that action should be taken.
# The OID for INFO is 1.3.6.1.4.1.17304.7.3.51. No action needs to be taken
# for INFO. Layer 7 has reserved the 1.3.6.1.4.1.17304.7.3.* MIB space. To
# set the OID to something other than .50 or .51 change the values of
# OID_ERROR and OID_INFO below.
#
# Configuring Exim
# ~~~~~~~~~~~~~~~~
#
# To configure exim to use a smarthost edit /etc/exim/exim.conf and find the
# line with "begin routers". Add the following lines immediately after "begin
```

```
# routers" line. Change "mailhost.example.com" to your smarthost:
#
# # Configure a smart host to route everything but local domain
# smarthost:
# driver = manualroute
# domains = !+local_domains
# transport = remote_smtp
# route_data = mailhost.example.com
#
# Make sure the MTA_FLAGS below is set to "-t"
#
# Configuring Sendmail
# ~~~~~~~~~~~~~~~~~~~~
#
# Configure the smarthost in sendmail by setting the DS configuration in the
# /etc/mail.sendmail.cf file and set the MTA_FLAG value below to "-Am -t"
#
# Configuring crontab
# ~~~~~~~~~~~~~~~~~~~
#
# Run the command 'crontab -e' as root and add the following line (note: it
# opens the file in vi). See the crontab man page for details:
#
# to check every hour on the hour:
#    0 * * * * /usr/local/bin/monitor_replication.sh
#
# to check every day at 04:15
#    15 4 * * * /usr/local/bin/monitor_replication.sh
#
# Jay MacDonald - v1 - 20100303 - modified manage_binlogs.sh

################################################# Configurable settings

# SLAVE is either IP address or host name
MONUSER="monitor"
MONPWD="monitorpass"

# NOTIFY sends email on error conditions only. If you want notifications on
# all runs set NOTIFY_PURGE as well
NOTIFY="yes"

# Set the notification methods
NOTIFY_SMTP="yes"
NOTIFY_SNMP="no"

if test "$NOTIFY_SMTP" == "yes" ; then
        # Configure email alert settings. Must set NOTIFY_TO.
        # NOTIFY_CC and NOTIFY_BCC are optional.
        NOTIFY_TO="SET ME"
        NOTIFY_CC=""
        NOTIFY_BCC=""

        # For exim set MTA_FLAG to "-t". For sendmail set it to "-Am -t"
        MTA_FLAG="-t"

        if test "$NOTIFY_TO" == "SET ME" ; then
                echo "Error: NOTIFY_TO is not configured"
                exit 1
        fi
fi

if test "$NOTIFY_SNMP" == "yes" ; then
        # Configure SNMP Trap alert settings.
        # Must set SNMP_HOST and COMMUNITY
        SNMP_HOST="SET ME"
        COMMUNITY="SET ME"

        # Change these if you want to use different values
```

```
        OID_ERROR="54"
        OID_INFO="55"

        UPTIME=`/bin/cat /proc/uptime | /bin/cut -f 1 -d '.'`

        if test "$SNMP_HOST" == "SET ME" ; then
                echo "Error: SNMP_HOST is not configured"
                exit 1
        fi

        if test "$COMMUNITY" == "SET ME" ; then
                echo "Error: COMMUNITY is not configured"
                exit 1
        fi

        if test ! -x /usr/bin/snmptrap ; then
                echo "Error: Can't find /usr/bin/snmptrap. Make sure the net-snmp-utils rpm"
                echo "has been installed."
                exit 1
        fi
fi

# Set VERBOSE to yes when debugging. Under normal operations it should
# probably be set to "no"
VERBOSE="yes"

############################################################################
################################################## End configurable settings
############################################################################

notify () {
        if test "$NOTIFY" != "yes" ; then
                return 1;
        fi

        if test "$NOTIFY_SMTP" == "yes" ; then
                echo "From: ${USER}@${HOSTNAME}" > /tmp/mb_notify.$$
                echo "To: $NOTIFY_TO" >> /tmp/mb_notify.$$
                if test -n $NOTIFY_CC ; then
                        echo "Cc: $NOTIFY_CC" >> /tmp/mb_notify.$$
                fi
                if test -n $NOTIFY_BCC ; then
                        echo "Bcc: $NOTIFY_BCC" >> /tmp/mb_notify.$$
                fi
                echo "Subject: $SUBJECT" >> /tmp/mb_notify.$$
                echo "" >> /tmp/mb_notify.$$
                cat /tmp/mb_message.$$ >> /tmp/mb_notify.$$

                if test "$VERBOSE" == "yes"  ; then
                        echo "Sending notification by email"
                fi

                cat /tmp/mb_notify.$$ | /usr/sbin/sendmail $MTA_FLAG
        fi

        if test "$NOTIFY_SNMP" == "yes" ; then
                if test "$VERBOSE" == "yes"  ; then
                        echo "Sending notification by SNMP trap"
                fi

                if test -z $1 ; then
                        OID=$OID_ERROR
                else
                        OID=$1
                fi

                # Send the trap. Note: this is a v2c style message.
                /usr/bin/snmptrap -v 2c -c $COMMUNITY $SNMP_HOST $UPTIME \
                1.3.6.1.4.1.17304.7.3.${OID} \
                SNMPv2-SMI::enterprises.17304.7.3.0 s "${SUBJECT}"
        fi
}

# Get the timestamp
DATE=`date`

# Query the SLAVE STATUS from the slave database
RESULT=`mysql -u$MONUSER -p$MONPWD -e "SHOW SLAVE STATUS\G" 2>/tmp/mb_error.$$`

# Check to make sure we got the connection, notify and fail if not
if test $? -ne 0; then
        cat > /tmp/mb_message.$$ <<-EOM

                ==> Error querying database

                Timestamp: $DATE
                Message: `cat /tmp/mb_error.$$`

                Refer to http://dev.mysql.com/doc/refman/4.1/en/error-messages-client.html
                for more information.

        EOM
        if test $VERBOSE == yes ; then
                cat /tmp/mb_message.$$
        fi

        SUBJECT="WARNING: Error querying replication status"
        notify $OID_ERROR

        rm -f /tmp/mb_*.$$
        exit 1
fi

# Get the useful information from SHOW SLAVE STATUS\G and convert to variables
eval `echo "$RESULT" | \
        sed 's/^  *//' | \
        sed 's/: \(.*\)/="\1"/' | \
        grep -v '=""$' | \
        grep '='`

if test "$VERBOSE" == "yes"  ; then
        echo "Master_Host = $Master_Host"
        echo "Slave_IO_Running = $Slave_IO_Running"
        echo "Slave_SQL_Running = $Slave_SQL_Running"
        echo "Master_Log_File = $Master_Log_File"
        echo ""
fi

# Check the status of the slave and send notification if in failed state
if test "$Slave_IO_Running" == "Yes" -a "$Slave_SQL_Running" == "Yes" ; then
        if test "$VERBOSE" == "yes"  ; then
                echo "Slave is functioning properly."
                echo ""
        fi
        rm -f /tmp/mb_*.$$
        exit 0
else
        if test "$VERBOSE" == "yes" ; then
                echo "WARNING: Slave is not functioning properly"
                if test "$NOTIFY" == "yes" ; then
                        echo " --- This *should* have been sent by SMTP or SNMP ---"
                else
                        echo " --- No one was notified! ---"
                fi
        fi
        # Create message and send

        SUBJECT="WARNING: DR slave on $HOSTNAME is not functioning properly"

        cat > /tmp/mb_message.$$ <<-EOM
                WARNING: The database slave at $HOSTNAME is not functioning properly
                Timestamp: $DATE

                Here are the results of SHOW SLAVE STATUS\G:

                $RESULT
        EOM

        notify $OID_ERROR
        rm -f /tmp/mb_*.$$
        exit 1
fi
```

Version History

Date/Ver	Edited By	Comments
20100303	Jay MacDonald	Initial Version for SecureSpan v5.2