NETAPP TECHNICAL REPORT

An Automatic MetroCluster Site Failover Solution

Jorge Costa, Michael Kiernan, NetApp
Reviewed by Jim Lanson, NetApp
TR-3660

ABSTRACT

NetApp® MetroCluster provides a metropolitan-distance highly available synchronous mirroring solution with unique integrated rapid disaster recovery functionality. Normally the decision to bring services up on a disaster recovery site in the event of a real disaster is made manually. However, the demands of modern "twin-site" or colocation data center architectures mean that all services must keep running even in the event of loss of either of the two primary data centers. This paper defines a "tiebreaker" solution that enhances the current MetroCluster solution with arbitrated failover functionality, thus enabling a true twin-site high-availability multiprotocol storage solution.

Table of Contents

1 INTRODUCTION

1.1 INTENDED AUDIENCE

This document is intended for field personnel and administrators who are responsible for designing and deploying automatic MetroCluster high-availability (HA) and disaster recovery (DR) configurations.

1.2 SCOPE

This document covers the following MetroCluster configurations:

- Stretched or nonswitched MetroCluster (NetApp storage)
- Fabric or switched MetroCluster (NetApp storage)

1.3 REQUIREMENTS AND ASSUMPTIONS

For the methods and procedures described in this document to be useful, it is assumed that the reader has met the following prerequisites:

- At least basic NetApp administration skills and administrative access to the storage system via the command-line interface
- Full understanding of clustering as it applies to the NetApp storage controller environment
- At least basic understanding of the creation and implementation of Perl scripts

1.4 DISCLAIMER

The solution described in this document is an example of how to automate the site-to-site failover process; it is not the only way to accomplish this task. Therefore, NetApp does not support the scripts described and provided in this report; they are shown as examples only.

1.5 METROCLUSTER OVERVIEW

NetApp MetroCluster is a unique multiprotocol storage solution that combines synchronous data mirroring with high availability and rapid disaster recovery over campus and metropolitan distances. MetroCluster protects against a number of failure scenarios and outages, including:

- Hardware failure (controllers, disks, shelves, Fibre Channel switch infrastructure, fiber ring failure, and so on)
- Data center failure, human errors, and natural disasters
- Nondisruptive maintenance procedures to minimize planned and unplanned downtime

MetroCluster is a highly available solution by design, and automated recovery from component failures is a standard feature. However, the real value of MetroCluster is as an extremely cost-effective and efficient disaster recovery solution. During a total site loss event, services can be restored at the disaster recovery site with a single command in a matter of minutes—no complex failover scripts or procedures are required. This massively simplifies the process of deployment and maintenance of a true campus or metropolitan synchronous DR solution. Such solutions have historically involved the deployment of costly high-end frame arrays with heavily customized packages and scripts to bring volumes online and map them to hosts on the DR site in the event of a true site disaster. Lengthy consulting exercises and extensive testing are usually a prerequisite of these highly complex deployments. MetroCluster offers a far simpler and more effective and affordable solution for campus and metropolitan area disaster recovery, and it can typically be set up and deployed in just a few hours.

Failing over business-critical services to a disaster recovery site with any synchronous or asynchronous replication solution is usually a human decision—and for very good reason. Automated failover can result in a “split-brain” condition in which the two sites become independently active, although isolated and thus unaware of each other’s status, due to a network outage. At best, this can cause operational headaches; at worst it can result in corruption or pollution of both primary copies of the data, making resynchronization impossible. In that case, recovery from backup is probably the only option. Because the decision to fail over services to the remote DR site is manual, there is usually an extended period of downtime while the operational staff performs sanity checks and makes a go/no-go decision on performing the failover. Thus service availability is compromised in favor of data integrity. A full site loss is usually considered a rare and catastrophic event, so an extended outage may be deemed an acceptable business risk. Such an outage can be minimized with the standard NetApp MetroCluster solution because its single-command takeover massively simplifies the mechanics of cutting services over to the DR site.

1.6 THE TWIN-SITE DATA CENTER

Manual rather than automated failover is the norm for disaster recovery solutions in the event of primary site loss. However, there are some scenarios in which automated recovery from a site disaster is not only desirable, it is *required*. One such scenario is that of a ”twin-site” or colocation architecture—a single virtual data center traversing two geographically separated sites. The two physically separate data centers may even share FC SAN and network infrastructure with intentionally little or no differentiation between the two sites. Service-level demands for such deployments are such that even a catastrophic loss of one of the two data centers must not result in *any* service interruption. All solutions deployed in such an architecture must thus comply with a simple requirement: If either site drops, the other site must keep all the services running.

MetroCluster is perfectly suited to the demands of this kind of cross-site fault-tolerant solution. This paper describes a simple extension to the MetroCluster solution to remove the manual element of the disaster recovery decision, providing a true hands-off failover solution that complies with twin-site availability requirements. This solution is the “MetroCluster tiebreaker.”

1.7 HIGH AVAILABILITY OR DISASTER RECOVERY?

Although MetroCluster is often referred to as a “combined HA and DR solution,” the terms high availability and disaster recovery are not interchangeable.

High availability refers to the fault tolerance of the solution as a whole; that is, the ability of the solution to keep the services that it supports online and accessible by the end user.

Disaster recovery refers to a business continuity solution that can be implemented in the event of one or more catastrophic failures that cause loss of primary systems, data centers, or critical infrastructure.

Although the MetroCluster solution itself is an HA solution (it can tolerate failure of any individual component and keep services running), it cannot tolerate the total loss of a complete site and thus multiple components simultaneously. Complete site loss requires manual intervention and the invocation of the DR portion of the product, implementing a single-command failover to the DR site.

A twin-site solution blurs the lines even further. There is no primary site, and there is no disaster recovery site. Both sites perform both roles; each is the disaster recovery site for the other. Active-active data centers are very common, even for non-twin-site solutions. The differences between the twin-site solution and a normal active-active data center are the shared infrastructure and the availability requirement.

The true definition of DR in this context hinges on the definition of a disaster. If a disaster is the loss of a single data center, then this is an extremely fast DR solution. However, because the twin site is really a single logical data center, a disaster would actually mean complete loss of the whole twin site. Because the enhanced MetroCluster tiebreaker solution must tolerate single site loss and keep services running, it really provides a cross-site stretch HA solution.

2 TWIN-SITE DATA CENTER CHALLENGES

The demands of the twin-site architecture pose significant challenges to solution design. All services must have the ability to continue running uninterrupted in the event of the loss of either of the primary data centers. No compromises are offered to account for deficiencies in the underlying technology; the service level must be met, and the technology choices made must deliver. Thus from a storage perspective, whether the solution is based on SAN, iSAN, or NAS protocol is unimportant, as long as the availability requirement is met. Individually designed solutions are thus required in the following areas:

- HA Server clusters and SAN/iSAN protocols

- Clustered file system failover (for example, stretch Oracle® RAC or AFS deployments)
- NAS protocol failover (complementary or alternative to the block-based solutions

This paper describes all three areas and focuses on the third one.

2.1 HA SERVER CLUSTERS AND SAN/ISAN PROTOCOLS

Storage array level replication has historically not been suited to twin-site server clustered solutions such as HACMP and Veritas™ Cluster. The servers themselves provide a highly effective solution for maintaining service during total loss of a twin site: host-based mirroring. By mapping LUNs from both of the twin sites, and then mirroring between them by using the host-resident volume manager (such as LVM or VxVM), as shown in Figure 1, the server remains in control of the data volumes it requires to keep services running. Thus in the event of a total site loss, the server cluster software is capable of making its own failover decision. The mirrored storage resources are simply imported to the cluster partner and brought online, and services are restored. This all occurs with zero manual intervention.

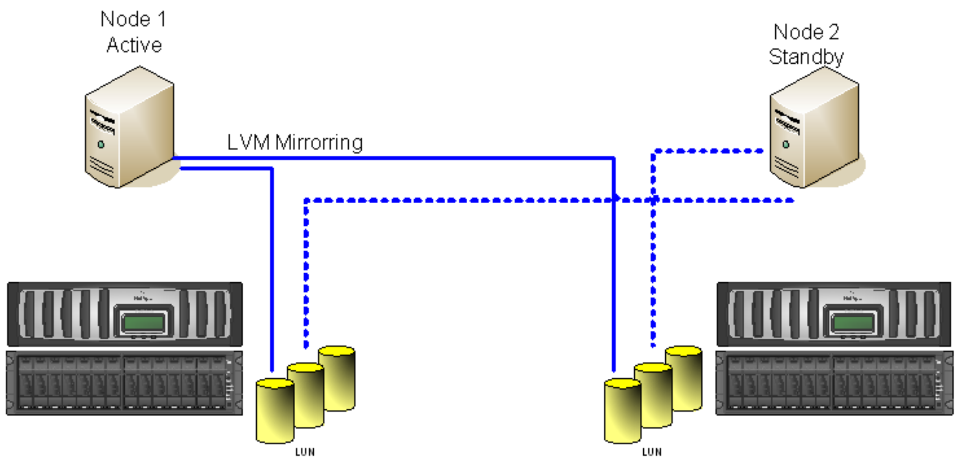

Figure 1) Twin-site data centers.

This works so well because of the small number of hosts involved (most commonly just two), and the hosts themselves are in control not only of the mirroring but also of the process of arbitrated failover to the partner site in the event of loss of the primary site. However, host-based server clustering solutions are also susceptible to the split-brain scenario described in section 3. Therefore server clustering solutions often have to provide their own tiebreaker type packages (residing in a third location) to solve the same problem.

The drawback of the host-based approach is that far more components need to be purchased, installed, maintained, and managed compared to a storage-level solution. Storage-level replication such as that provided by MetroCluster is a far simpler and more cost-effective method for providing synchronous off-site replication.

Although a similar level of uptime can be achieved by using MetroCluster with hands-off tiebreaker functionality, as described in this paper, host-based mirroring remains a highly effective, reliable, and widely deployed solution for the twin site when it comes to high-end SAN/iSAN and server clustering solutions.

2.2 CLUSTERED FILE SYSTEMS

Clustered file systems such as VxFS, OCFS, and GPFS are normally “host resident,” just like normal file systems, and so they rely on host-based volume mirroring of underlying block-based storage. The main difference with clustered file systems is that the file system does not usually reside on one site or the other, but traverses both locations as a single file system. This configuration allows active nodes on both locations to access the same file system, without incurring asymmetric round-trip times. This can be useful for certain application configurations such as stretched Oracle RAC. However, as with all twin-site solutions, a total site loss results in the need to bring mirrored volumes online, a decision that cannot be taken automatically. To automate the process, a third site using a steward process or package to initiate takeover is required. For a clustered file system, this often involves placing Fibre Channel equipment at the third site to supply quorum storage to govern the takeover process. Such configurations are usually highly complex and extremely difficult to maintain.

2.3 NAS PROTOCOL FAILOVER

NAS (NFS or CIFS) protocols require somewhat different considerations. Two types of NAS deployment should be considered:

- Tightly coupled; for example, an application or a database server or cluster with dedicated IP SAN running NFS
- Distributed; that is, tens, hundreds, or thousands of distributed clients

CIFS usually falls into the latter category (Windows® applications run almost exclusively on block protocols [iSCSI/FCP] these days). The CIFS protocol itself is inherently limited in its ability to provide a twin-site compliant solution because moving a CIFS connection from one twin site to the other always results in a service interruption.

NFS connections (provided that they are hard mounted) can be moved between sites with no problem, and thus are well suited to the twin-site architecture. In addition, unlike the block protocols, the NFS client or host itself cannot make a failover decision, even in the example of the tightly coupled deployment discussed earlier. The NFS client is generally not able to perform the mirroring, although in some cases the application itself could, by using application-level file multiplexing or mirroring (for example, Oracle with log file destinations and ASM). In general, when using NFS, the data can and must be replicated at the level of the storage array, which underlies the NFS server itself. This describes a NetApp MetroCluster.

Attempts have been made in the past to provide NFS clients with the intelligence to perform their own redirection to alternative locations (sources of data or servers) in the event of the loss of the primary system. The Solaris™ NFS client/automounter in particular includes a feature that allows a client to list alternative NFS servers for a single file system, as follows:

```
/nfsdata            -ro    controller1:/vol/nfsdata   controller2:/vol/nfsdata
```

In this example, a client that has mounted `/nfsdata` exchanges traffic with controller1. If controller1 goes down or is inaccessible by the client, the client switches its requests (after a timeout period) to controller2. Controller2 has a replicated or mirrored copy of the data, so it can continue to service the client's requests from the second site without interruption. The drawback is that this functionality only works for read-only file systems. A client cannot make its own decision to fail over to a read-write mirror because it may see that the data on the DR site is out of date or has rolled back in time from the copy it was just reading on the primary site.

Because NFS cannot perform its own replication, it must rely on storage-level replication, and another mechanism must be found to fail over the services to the twin site in the event of a disaster. This method is discussed in section 4, "The MetroCluster Tiebreaker Solution."

3 THE SPLIT-BRAIN PROBLEM

The very idea of a twin site is in itself somewhat flawed. In order to provide the specified service levels, a third site, or triple-site architecture, is required. This is because of the possibility of a "split brain" or partition occurring. If both sites and subsets of the sites lose contact with each other (loss of networks or fiber rings between the twin-site locations), none of the individual solutions deployed in the twin site can be relied upon to behave "correctly" on their own, and it's possible that data integrity may become compromised. Many host/server clustering software applications have relatively mature solutions to address this issue. However, the solution generally comes down to the following:

- A third site is required with full, reliable network connectivity to each of the primary sites.
- A daemon process on the third site arbitrates takeover. This process is often referred to as a "tiebreaker" or "steward." In some cases this is a mature, packaged product. In many cases it is a simple script.
- In some cases, Fibre Channel access may be required in order to provide a SCSI-reserved quorum volume to use as an alternative arbitration mechanism.

- Both primary nodes can be shut down, either via out-of-band communication or by shutting themselves down (usually by panicking), if they cannot be contacted by either the steward process or the partner node in the cluster.

Such solutions are rarely deployed due to the difficulty of locating and funding a suitable third site. It is also highly unlikely that the third site will have separate networks to each of the twin-site locations; it's far more likely that the connection to one twin site runs through the other. However, in the case of a true twin-site deployment, for at least the most critical services, tiebreaker solutions must be deployed, along with the infrastructure required to support them.

4 THE METROCLUSTER TIEBREAKER SOLUTION

4.1 SOLUTION OVERVIEW

The architecture of MetroCluster is in some ways similar to that of a host-based cluster, and therefore we have taken the same approach of using a third site to arbitrate takeover between the twin-site locations. Figure 2 shows the NetApp MetroCluster tiebreaker solution.

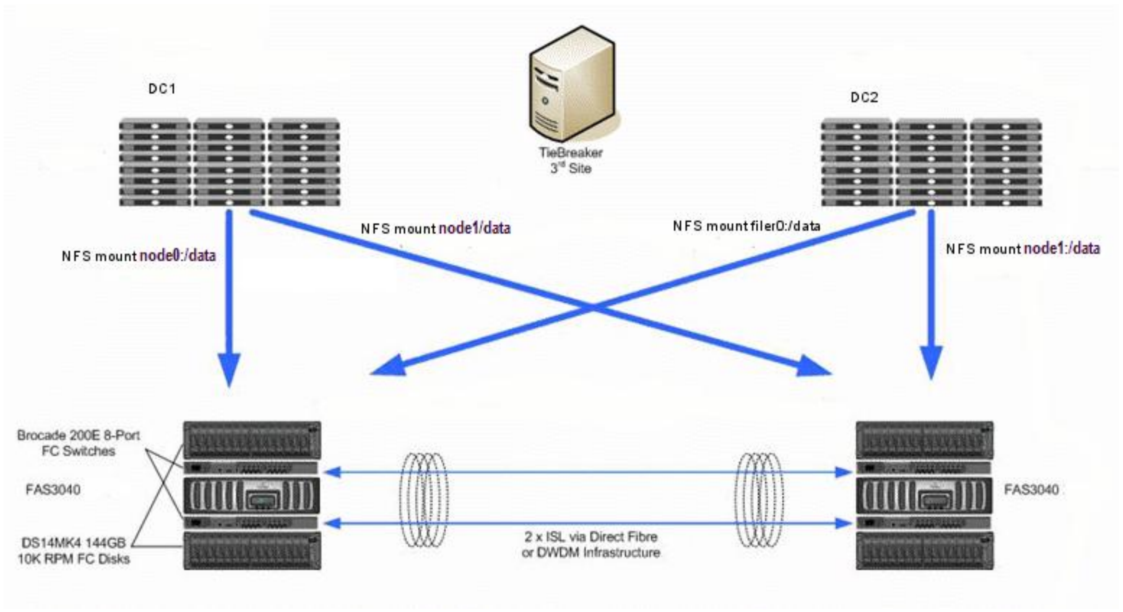

Figure 2) MetroCluster tiebreaker solution

The storage infrastructure is highly available. By placing two FAS controllers in a MetroCluster configuration, all single points of failure are eliminated. This solution uses the NetApp SyncMirror® functionality to synchronously mirror all data blocks between two sets of disks. These disks are located a significant distance from each other, thus securing the data in case of a site failure or disaster. This way, the solution combines both failover and disaster recovery functionality.

The tiebreaker server can be any Linux® host located in a third location with network connectivity to both sites. The tiebreaker script itself runs as a daemon process and communicates with both FAS controllers by using the Data ONTAP® Manage ONTAP® API over SSL. The process monitors both MetroCluster nodes continuously, and initiates takeover if neither the tiebreaker nor the remaining node is able to access the down site. The details of the takeover decision and processing are documented in the next section 4.3.

4.2 INFRASTRUCTURE REQUIREMENTS

THIRD LOCATION FOR THE TIEBREAKER SCRIPT

The tiebreaker script should be running on a location with network access to both primary storage locations or sites. This is required so that the tiebreaker can differentiate the loss of network connectivity between sites from the loss of a single site in order to be able to declare disaster and initiate automated recovery.

NETWORK REQUIREMENTS FOR THE THIRD LOCATION

The tiebreaker script is designed to handle loss of network connectivity to both sites by waiting until the network links are reestablished.

Multiple network communication links are recommended to allow the continuous monitoring of both sites.

FIREWALL CONSIDERATIONS

The tiebreaker requires access to the NetApp controllers on port 443. This port is used to monitor cluster status and to initiate takeover after a disaster.

For monitoring normal network availability, the tiebreaker needs to be able to ping the controller IP addresses and the default gateway IP addresses used by the controllers.

tiebreaker -> NetApp controllers: 443

tiebreaker -> NetApp controllers: ICMP

tiebreaker -> Sites default gateways: ICMP

UNIX SERVER WITH A RECENT PERL VERSION

The tiebreaker script is developed in Perl so that it can be easily ported between multiple UNIX platforms.

Because the script includes a NetApp controller user and password defined in the code, it must be compiled to secure the password during normal execution.

The tiebreaker is designed to be compiled during the initial installation, when network parameters change, or when the NetApp tiebreaker user password is modified.

MANAGE ONTAP SDK

The tiebreaker script relies on accessing the NetApp controllers by using the Manage ONTAP API. This requires a client-side library or SDK. (The example tiebreaker script uses the Perl library.) To get the most up-to-date copy of the Manage ONTAP Software Developer's Kit (SDK), go to the NetApp Advantage Developer Program page at http://www.netapp.com/solutions/dfms/advantage.html and create an account. You can then download the SDK from the Manage ONTAP Developer Portal at http://now.netapp.com/NOW/cgi-bin/manageontap.

DATA ONTAP 7.2.4

The tiebreaker script requires Data ONTAP 7.2.4 or later. Data ONTAP 7.2.4 is the first release to include the `cf.takeover.change_fsid` option, which is required for an automatic failover during a disaster.

4.3 FUNCTIONALITY

WHAT DOES THE TIEBREAKER MONITOR?

The tiebreaker software probes the health of the controllers and network connectivity of both sites by means of an ICMP connection to the default gateway defined on the NetApp controller and the controller itself.

Using the NetApp ONTAPI™ Perl libraries, the tiebreaker establishes an SSL connection to both controllers and continuously checks the status of the cluster and the availability of both nodes.

WHEN IS AUTOMATED RECOVERY INITIATED?

In case of a site failure resulting in the loss of both controller head and disk shelves, Data ONTAP on the remaining head detects the failure and reconfigures itself for a disaster situation. In this situation, Data ONTAP waits for manual intervention to initiate recovery.

During this period, the tiebreaker is constantly monitoring both sites and controllers. As soon as it detects a failure recognized by the surviving head as a disaster, it probes the failed site to check if the controller failed or if the site is no longer available. In either of these cases, the tiebreaker forces a takeover on the surviving site.

LOSS OF NETWORK CONNECTIVITY

The tiebreaker script monitors two possible situations: when a network outage occurs between site 1 and site 2; and when site 3 becomes isolated from all remaining sites.

In this situation, manual intervention is required because there are not any failures that require a takeover; the tiebreaker simply logs this condition to the log file.

5 INSTALLATION STEPS

5.1 REQUIRED PERL MODULES

DOWNLOAD AND INSTALL THE MANAGE ONTAP SDK

http://now.netapp.com/NOW/cgi-bin/sdklic.cgi/partner/manageontap/sdk/2.0P2/manage-ontap-sdk-2.0P2.zip

```
mkdir –p /opt/NetApp
cd /opt/NetApp
unzip manage-ontap-dsk-2.0P2.zip
```

DOWNLOAD AND INSTALL NET::PING

http://search.cpan.org/CPAN/authors/id/S/SM/SMPETERS/Net-Ping-2.33.tar.gz

```
gzip –d Net-Ping-2.33.tar.gz
tar xf Net-Ping-2.33.tar
perl Makefile.pl
make
make install
```

DOWNLOAD AND INSTALL EXPAT

http://ovh.dl.sourceforge.net/sourceforge/expat/expat-2.0.1.tar.gz

```
tar xvzf expat-2.0.1.tar.gz
cd expat-2.0.1
./configure
make
make install
```

DOWNLOAD AND INSTALL XML::PARSER

http://search.cpan.org/CPAN/authors/id/M/MS/MSERGEANT/XML-Parser-2.34.tar.gz

```
tar xvzf XML-Parser-2.34.tar.gz
cd XML-Parser-2.34
perl Makefile.PL
make
make install
```

5.2 INSTALL THE TIEBREAKER SCRIPT

For an example tiebreaker script, see Appendix B.

Create a directory for the `tiebreaker` script:

```
# mkdir –p /opt/NetApp/tiebreaker
# mkdir –p /opt/NetApp/tiebreaker/log
```

Copy the `tiebreaker` script from Appendix B to this directory:

```
# cp tiebreaker.pl /opt/NetApp/tiebreaker/
```

Edit `tiebreaker.pl` using VI or your favorite editor.

Variable `$site1_gw` is the default gateway IP address defined on the NetApp controller. Variables `$node1` and `$node2` are the IP addresses of the NetApp controllers, and `$password` is the password of `$user mc-cfod-steward`.

The user `mc-cfod-steward` is the user account used in the example to initiate DR. Choose a strong password, keeping in mind that this password may not be changed as often as the root password.

```
#- Enter Controllers Default GW site information
our $site1_gw = "10.10.11.1";
our $site2_gw = "10.10.12.1";


#- Enter MetroCluster Login Information
our $node1 = "10.10.11.230";
our $node2 = "10.10.12.231";
our $user = "mc-cfod-steward";
our $password = "netapp";  #
```

(Optional) After modifying this file, compile it using `perlcc`:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
# perlcc –B mc_tiebreaker.pl
# mv a.out mc_tiebreaker.exe
# chmod 700 mc_tiebreaker.exe
```

Remove the password entry from the Perl script.

5.3 ENABLE THE TIEBREAKER ON BOOT

Create an rc boot script `mc-tiebreaker` by using the example script from Appendix A and add the required entries so that the tiebreaker boots on startup.

```
# cp mc-tiebreaker.boot /etc/init.d/mc-tiebreaker
# chmod 755 /etc/init.d/mc-tiebreaker
# ln -s  /etc/init.d/mc-tiebreaker /etc/rc.d/rc2.d/S99mc-tiebreaker
```

5.4 CONFIGURE LOG ROTATION

The logs are kept in `/opt/NetApp/tiebreaker/log` with the name `mc_logfile.log`.

These logs can grow quite quickly and should be rotated by using a standard mechanism such as `logrotate`.

Use the following configuration file to rotate the log files on a weekly basis by using `logrotate`.

Create the following file: `/etc/logrotate.d/tiebreaker`.

```
/opt/NetApp/tiebreaker/log/mc_logfile.log {

    weekly

    rotate 5

    copytruncate

    compress

    notifempty

    missingok

}
```

5.5 CREATE THE TIEBREAKER USER

Create the user that the `tiebreaker` script uses to log in to the MetroCluster nodes and perform monitoring checks or initiate takeover. The account can be restricted to a limited set of activities by using the Data ONTAP RBAC rules.

Log in to the NetApp controllers and create the user `mc-cfod-steward` as defined in the `tiebreaker` script on both controllers:

```
node1*> useradmin role add mc-cfod-admin -a login-ssh,login-http-admin,api-cf*
-c "MetroCluster CFOD Tiebreaker"

node1*> useradmin group add mc-cfod-admin

node1*> useradmin group modify mc-cfod-admin -r mc-cfod-admin

node1*> useradmin user add mc-cfod-steward -g mc-cfod-admin -c "MetroCluster
CFOD Tiebreaker"

New password:

Retype new password:
```

Repeat these steps on the second controller.

5.6 CONFIGURE FSID PERSISTENCE

Log in to the NetApp controllers.

Change the option `cf.takeover.change_fsid` to off on both controllers:

```
node1*> options cf.takeover.change_fsid off

node2*> options cf.takeover.change_fsid off
```

5.7 START THE TIEBREAKER DAEMON

Log in to the UNIX host; as root, execute the following command:

```
# /etc/init.d/mc-tiebreaker start
```

5.8 STOPP THE TIEBREAKER DAEMON

Log in to the UNIX host; as root, execute the following command:

```
# /etc/init.d/mc-tiebreaker stop
```

5.9 ACCESS THE LOGFILE

Log in to the UNIX host; as root, execute the following command:

```
# tail -f /opt/NetApp/tiebreaker/log/mc_logfile.log
```

).

APPENDIX A: EXAMPLE TIEBREAKER BOOT SCRIPT

DISCLAIMER: This script is provided “as-is.” No formal support for this script is offered or may be inferred.

```
#! /bin/sh
 case "$1" in
    start)
        echo  "Start service NetApp tiebreaker"
        if [ $( ps -ef | grep -v grep |grep -c \opt/NetApp/tiebreaker/ mc_tiebreaker.exe) -
eq 0 ]
        then
            nohup /opt/NetApp/tiebreaker/mc_tiebreaker.exe >>
/opt/NetApp/tiebreaker/log/mc_logfile.log 2>&1 &
        else
            echo "NetApp tiebreaker already running"
        fi
        ;;
    stop)
    echo  "Stop service NetApp tiebreaker"
    if [ $(  ps -ef | grep -v grep | grep -c /opt/NetApp/tiebreaker/mc_tiebreaker.exe) -gt 0
]
        then
            killall mc_tiebreaker.exe
        else
            echo "NetApp tiebreaker not running"
    fi
    ;;
    restart)
        $0 stop  &&  $0 start
    ;;
    status)
    if [ $(  ps -ef | grep -v grep | grep -c /opt/NetApp/tiebreaker/mc_tiebreaker.exe) -gt 0
]
        then
            echo "NetApp tiebreaker is running"
        else
            echo "NetApp tiebreaker not running"
    fi
;;
*)
echo "Usage: $0 {start|stop|status|restart}"
exit 1
;;
 esac
```

APPENDIX B: EXAMPLE TIEBREAKER SCRIPT

DISCLAIMER: This script is provided “as-is.” No formal support for this script is offered or may be inferred.

```
#!/usr/bin/perl
#/**************************************************************************
#
# Copyright 2002-2007 Network Appliance, Inc. All rights
# reserved. Specifications subject to change without notice.
#
# NAME: mc_tiebreaker.pl
#
# PURPOSE: Provide cluster tiebreaker functionality for NetApp MetroCluster
# (RPO=0, RTO~=0 DR/HA twin-site solution)
#
my $VERSION = '2.0P2'; # Controls the SDK release.

#- Enter Controllers Default GW site information
our $site1_gw = "nnn.nnn.nnn.nnn";
our $site2_gw = "nnn.nnn.nnn.nnn";

#- Enter MetroCluster Login Information
our $node1 = "nnn.nnn.nnn.nnn";
our $node2 = "nnn.nnn.nnn.nnn";
our $user = "name";
our $password = "$7_sjxXlm";  #

#- Enter MetroCluster RLM Login Information

use strict;
use lib "/opt/NetApp/manage-ontap-sdk-2.0P2/lib/perl/NetApp/";
use NaServer;
use Pod::Usage;
use Net::Ping;

my %site1 = ();
my %site2 = ();
# site1{accessible} 'y' / 'n'
# site1{controller_accessible} 'y' / 'n'
# site1{controller_cluster_enabled} 'true' / 'false'
# site1{controller_cluster_IC} 'up' / 'down'
# site1{controller_cluster_partner} 'CONNECTED' / 'ERROR'
my $debug = 1;

sub check_sites_connectivity {
# first we check for network connectivity to the site
# best way is to ping the default gateway where the controllers are connected.

if ($debug) {print "DEBUG: BEGIN check_sites_connectivity \n" };

    # is site1 accessible ?
    our $p = Net::Ping->new("tcp",2);
    if ( $p->ping($site1_gw) )
    {
        $site1{accessible}="y";
        if ($debug) {print "DEBUG: site1{accessible}=y \n" };
    }
        else
    {
        $site1{accessible}="n";
        if ($debug) {print "DEBUG: site1{accessible}=n \n" };
    }
  $p->close();

    # is site2 accessible ?
    our $p = Net::Ping->new("tcp",2);
    if ( $p->ping($site2_gw) )
    {
        $site2{accessible}="y";
        if ( $debug ) { print "DEBUG: site2{accessible}=y \n" };
    }
    else
    {
        $site2{accessible}="n";
        if ( $debug ) { print "DEBUG: site2{accessible}=n \n" };
    }
if ($debug) {print "DEBUG: END check_sites_connectivity \n" };
}; #sub check_sites_connectivity

sub check_controllers_connectivity {
# check if we can get to the controllers
if ($debug) { print "DEBUG: BEGIN check_controllers_connectivity \n" };

# can we access controller1 ?
        # is controller1 accessible ?
        our $p = Net::Ping->new("tcp",2);
        if ( $p->ping($node1) )
        {
                $site1{controller_accessible}="y";
        if ( $debug ) { print "DEBUG: site1{controller_accessible}=y \n" };

        if ( $debug ) { print "DEBUG: checking cfstatus controller1 \n" };
        # define ontapi sessions to controller heads, s1 and s2
            our $s1 = NaServer->new($node1, 1, 1);
             $s1->set_admin_user($user, $password);

            our $output = $s1->invoke( "cf-status" );
                $site1{controller_cluster_enabled} = $output->child_get_string( "is-
enabled" );
                $site1{controller_cluster_IC} = $output->child_get_string( "is-
interconnect-up" );
                $site1{controller_cluster_partner} = $output->child_get_string( "state" );
        if ( $debug ) { print "DEBUG: site1{controller_cluster_enabled} =
$site1{controller_cluster_enabled}\n" };
        if ( $debug ) { print "DEBUG: site1{controller_cluster_IC} =
$site1{controller_cluster_IC}\n" };
        if ( $debug ) { print "DEBUG: site1{controller_cluster_partner} =
$site1{controller_cluster_partner}\n" };
        }
        else
        {
                $site1{controller_accessible}="n";
        if ( $debug ) { print "DEBUG: site1{controller_accessible}=n \n" };
        }
  $p->close();

# can we access controller2 ?
        # is controller2 accessible ?
        our $p = Net::Ping->new("tcp",2);
        if ( $p->ping($node2) )
        {
                $site2{controller_accessible}="y";
        if ( $debug ) { print "DEBUG: site2{controller_accessible}=y \n" };

                if ( $debug ) { print "DEBUG: checking cfstatus controller2 \n" };
                # define ontapi sessions to controller heads, s1 and s2
                our $s2 = NaServer->new($node2, 1, 1);
                 $s2->set_admin_user($user, $password);

                our $output = $s2->invoke( "cf-status" );
                $site2{controller_cluster_enabled} = $output->child_get_string( "is-
enabled" );
                $site2{controller_cluster_IC} = $output->child_get_string( "is-
interconnect-up" );
                $site2{controller_cluster_partner} = $output->child_get_string( "state" );
        if ( $debug ) { print "DEBUG: site2{controller_cluster_enabled} =
$site2{controller_cluster_enabled}\n" };
        if ( $debug ) { print "DEBUG: site2{controller_cluster_IC} =
$site2{controller_cluster_IC}\n" };
```

```
        if ( $debug ) { print "DEBUG: site2{controller_cluster_partner} =
$site2{controller_cluster_partner}\n" };
        }
        else
        {
                $site2{controller_accessible}="n";
        if ( $debug ) { print "DEBUG: site2{controller_accessible}=n \n" };
        }
  $p->close();
if ( $debug ) { print "DEBUG: END check_controllers_connectivity \n" };
}; #sub check_controllers_connectivity

#-- Parse Outputs above, and make a go/no-go decision on CFOD

while (1==1)
{
 my $current = `date`;
 print "Starting at: $current";

 check_sites_connectivity();
 check_controllers_connectivity();

 my $current = `date`;
 print "$current";

 if ( $debug ) { print "DEBUG: 1.0\n" };
 # 1.0 - check that both sites are available
 if ($site1{accessible} eq "n" && $site2{accessible} eq "n")
 {
    if ( $debug ) { print "DEBUG: 3.0\n" };
    print "ERROR: 3.0 Site 3 isolated from the network\n";
    #exit 30;
 }

 if ( $debug ) { print "DEBUG: 1.1\n" };
 # 1.1 - check if both sites are accessible but not talking between between them
 if ($site1{accessible} eq "y" && $site1{controller_accessible} eq 'y' &&
$site1{controller_cluster_partner} eq "ERROR" && $site1{controller_cluster_IC} eq 'false'
&& $site2{accessible} eq "y" && $site2{controller_accessible} eq 'y' &&
$site2{controller_cluster_partner} eq "ERROR" && $site2{controller_cluster_IC} eq 'false' )
 {
    # 3.1 Network partition between sites 1 and 2
    if ( $debug ) { print "DEBUG: 3.1 \n" };
     print "ERROR: 3.1 Network Partition between sites \n";
     #exit 31;
 }

if ( $debug ) { print "DEBUG: 1.2\n" };
 # 1.2 - Is site1 down ?
 if ($site1{controller_accessible} eq 'n' && $site2{accessible} eq "y" &&
$site2{controller_accessible} eq 'y' && $site2{controller_cluster_partner} eq "ERROR" &&
$site2{controller_cluster_IC} eq 'false')
 {
    # 4.0 Site 1 Down
    if ( $debug ) { print "DEBUG: 4.0 \n" };
    print "ERROR: 4.0 Site 1 is Down \n";
    print "ERROR: 4.0 site2: cf takeover -d \n";

    our $s2 = NaServer->new($node2, 1, 1);
        $s2->set_admin_user($user, $password);

        our $output = $s2->invoke( "cf-status" );
    our $output = $s2->invoke( "cf-force-takeover","disaster","true" );
    if ($output->results_errno != 0)
    {
            our $r = $output->results_reason();
            print "Failed: $r\n";
    }
    else
    {
            print "BANG\n";
    } #- endif
    #exit 40;
 }


 if ( $debug ) { print "DEBUG: 1.3\n" };
 # 1.3 - Is site2 down ?
 if ($site2{controller_accessible} eq 'n' && $site1{accessible} eq "y" &&
$site1{controller_accessible} eq 'y' && $site1{controller_cluster_partner} eq "ERROR" &&
$site1{controller_cluster_IC} eq 'false')
 {
    # 4.1 Site 2 Down
    if ( $debug ) { print "DEBUG: 4.1 \n" };
    print "ERROR: 4.1 Site 2 is Down \n";
    print "ERROR: 4.1 site1: cf takeover -d \n";

    our $s1 = NaServer->new($node1, 1, 1);

        $s1->set_admin_user($user, $password);

        our $output = $s1->invoke( "cf-status" );
    our $output = $s1->invoke( "cf-force-takeover","disaster","true" );
    if ($output->results_errno != 0)
    {
            our $r = $output->results_reason();
            print "Failed: $r\n";
    }
    else
    {
            print "BANG\n";
    } #- endif
    #exit 41;
 }

# 1.4 is cf status ok ?
 if ( $debug ) { print "DEBUG: 1.4 \n" };
 if ($site1{controller_cluster_partner} eq "CONNECTED"  &&
$site2{controller_cluster_partner} eq "CONNECTED" )
 {
    if ( $debug ) { print "DEBUG: 0.0 \n" };
    print "0 - Partner is up and running \n";
    #exit 0;
 }
 else
 {
    if ( $debug ) { print "DEBUG: 255 \n" };
    print "WARNING: 255 - Undefined condition \n";
    #exit 255;
 }

sleep 1;
} # end while loop
```