## Mobile DLP agent using cluster solutions

**Overview**

The Forcepoint TRITON AP-DATA mobile agent is a solution that lets you secure content (such as email messages, calendar events, or tasks) synced to a mobile device. This document describes 2 cluster solution techniques you can use to improve the stability and performance on machines using the mobile DLP agent.

- 2-node high-availability solution
- 4-node high-availability and load-distribution solution

When the mobile DLP agent is configured for high-availability, it has the capacity to operate seamlessly and continuously, especially in the event of a system outage (such as a hardware or software failure). This enables mobile devices to be continuously protected against data leak without experiencing a system downtime, or causing any disruption to the user.

This document describes how you can set up a mobile agent for high-availability (HA) by using a 2-node proxy cluster, or for load-distribution by using a 4-node proxy cluster. Neither method requires you to install any additional hardware.

## High level architecture and basic flow

### How it works

The mobile agent acts as a reverse proxy agent, and is located at the front of an Exchange server. As a reverse proxy agent, all ActiveSync traffic is routed through the agent, allowing it to monitor and potentially manipulate requests and responses to the mobile device.
Typically, the mobile agent is deployed on a single node reverse proxy. In this deployment scenario, the agent has no fault-tolerance capabilities; therefore may become a single point of failure in the case of hardware or software failure. A single-node agent may also lack processing power to meet a corporate's ActiveSync demand. To guard against these problems, the mobile agent uses a cluster solution that provides high-availability and/or load-distribution.

**Cluster solutions**

A HA cluster solution consists of an active and passive 2-node model that is based on RedHat's cluster suite. RedHat’s cluster suite contains a collection of software that creates high-availability cluster. It is an integral part of the mobile agent operating system (CentOS), and offers a wider set of capabilities than those described in this document (e.g. asymmetric nodes topology, hardware dependent, mechanisms, etc.).

The RedHat cluster suite is installed on top of a regular mobile agent installation, and is completely orthogonal to the Data Security Manager system. Users, however, are free to use the additional capabilities offered by RedHat based to their actual needs. Refer to the [RedHat Cluster Administration document](#) for further details.

A 4-node HA/LD solution consists of two (or more) 2-node HA clusters with a round robin DNS setup to share the load traversing between these two clusters. The more HA clusters you deploy, the larger the load distribution.
Cluster environment

High-availability cluster (2-node)

The following diagram illustrates the components of a 2-node cluster deployment, and shows the logical channels between them.

Channel description:
0 – This channel denotes the interface required between each mobile agent node and organization LAN that essentially include the Data Security Manager server and the corporate Exchange. Note that this abstract channel might be implemented using two NICs per node, depends on network topology. There are NO special requirements on IP/Net addresses beside those required for a non-cluster deployment (routing, etc.).
1 – This channel denotes the per-node interface that serves for mobile devices access. These interfaces are usually connected to some “edge” device (firewall, NAT, etc.). Each interface has its own unique IP address, but the two addresses MUST be part of the same subnet or VLAN that must have room for the “floating” IP that the cluster is actually defining on the active node (which is also the one actually accessed by mobile devices).

2 – This channel serves the internal-cluster communication required in order to keep monitoring the health of each node. This channel MUST enable IGMP and multicast packets, and thus the two logical interfaces must share the same subnet or VLAN.

In a 2-node HA cluster solution, the traffic received from mobile devices is automatically routed to the active node which handles it as it would for a single node deployment. Once the cluster, using its heartbeat services implemented on top of logical channel 2, detects that the current active node is broken, it immediately disables everything it can in the current active node and defines a floating IP address in the passive node. This essentially switches roles between the two nodes providing the high-availability capability. Once the other node is revived, it is ready to serve as the passive node in this topology.

The cluster continuously maintains an active node where it defines the floating IP address on logical channel 1. This floating IP address is the one resolved by mobile devices, and is a cluster entity that does not appear in the Data Security Manager user interface. Under normal conditions, both agents are running and ready to serve requests and the only difference between these two nodes is the floating IP definition.

---

**Note**

The system regards each of the two mobile agent nodes as active and independent from each other; that is, the Data Security Manager has no cluster “notion”.

---

**Load-distribution cluster (4-node)**

The 4-node high-availability/load-distribution solution is an extension of the 2-node HA cluster solution. To extend to a 4-node HA/LD cluster solution, you need to deploy a second 2-node HA cluster (using the same deployment guidelines for both clusters), and configure the relevant DNS entry.

The process is simple: install 2 (or more) separate clusters, give each cluster a dedicated floating IP address (to be resolved by the same DNS hostname entry), and make sure the DNS service is configured to resolve this DNS entry according to a cyclic mechanism.
The following diagram shows the HA/LD topology and illustrates the components of a 4-node load-distribution deployment.

Each HA cluster here is simply a replica of the 2-node HA cluster (as described in the *High-availability cluster (2-node)* section).

Each of the 2-node HA cluster should be configured with an appropriate dedicated floating IP address. The method required to use these two floating IP addresses in a round robin fashion by the DNS system depends on the customer's DNS provider, and is beyond the scope of this document.

The DNS entry resolved by mobile devices for this site returns the two floating IP addresses owned by the two HA clusters in a cyclic-varied order (i.e. the first querying mobile device receives the list in a different order than the second device, and so forth). Most clients use the first IP address in the list. Since the DNS is configured in a way that produces cyclic ordering for a sequence of query requests, the effective resolving by different devices complies with a round robin schema.

Once a mobile device picks up an IP address to use, it generates an ActiveSync flow which is quite identical to the flow described above. One thing to take note of is that the mobile device will be served by one of the two clusters based on the resolved floating IP address.
You can also extend the cluster as much as required by adding additional 2-node HA clusters and floating IP addresses to this setup.

**Tip**
As a best practice, it is recommended that you use multiple mobile agent boxes in a load-balancing environment where there are multiple simultaneous users (rather than having a single agent connected to multiple Policy Engine servers).

**Note**
The system regards each of the mobile agent nodes (two for a 2-node cluster, or four for a 4-node cluster) as active and independent from each other; that is, there is no cluster notion.

### High-availability cluster deployment

The instructions provided in this section highlight the basic procedures required for a mobile agent deployment. It also includes some cluster-derived advice. For more information on mobile agent deployment, see the TRITON AP-DATA Installation Guide.

### General deployment

The following procedure uses a single-NIC deployment where all logical network interfaces are served by a single physical interface. If preferred, you can change to a multi-NIC setup. Changing to a multi-NIC setup is quite simple once you know how to set up a single-NIC deployment.

To deploy the mobile agent, you must log on as user root and follow these procedures:

1. Install the mobile agent image on 2 physical nodes. It is recommended that you use identical or similar hardware for these 2 nodes.
2. Make sure the network device(s) connecting the 2 nodes allows IGMP and multicast packets. This is essential for the cluster-internal heartbeat traffic (logical channel 2 above).
3. On each node, log-on as user root and then run the wizard script:
   a. Define the mobile mode, accept the license, and define the users’ passwords.
   b. Define a unique address for each node’s eth0. The 2 addresses must be included in the same subnet (required by logical channels 1 and 2 above).
   c. Use this single NIC as the default gateway.
   d. Define eth0 as the “Management NIC” (logical channel 0 above).
   e. Define a unique hostname per node.
Define a Subject field for the default certificate that the mobile agent can generate. This field should be identical in the two nodes in order for the cluster to identify itself as a single abstract object to mobile devices.

Define a DNS server for each node. Typically, the same DNS should serve both nodes.

Define the date. It is recommended that the dates are identical or similar for both nodes – ntpd could serve for that.

Register both nodes to the same Data Security Manager.

4. Log on to the Data Security Manager user interface and then select Settings > Deployment > System Modules. Configure both nodes using identical parameters. Two important points to consider here:

- The “Mobile Devices Connection” IP address should be configured to “ALL IP Address”.
- In case a custom certificate is required, we can assume that the same certificate should be deployed in both nodes.

5. Deploy the settings.

You have successfully configured the two mobile agent nodes in TRITON AP-DATA.

Cluster deployment

Setting up a cluster involves the following steps:

1. Configuring the node agent
2. Installing supplemental packages on the node
3. Configuring the cluster
4. Replacing instances in the cluster (on every node)
5. Replicating changes across the cluster
6. Enabling the cluster

Configuring the node agent

1. CCSD require an IPV6 binding, hence we should enable it on each mobile agent node:
   a. Stop network services.
      
      service network stop
   
   b. Using a vi or similar editor, add 2 lines to the /etc/sysconfig/network file:
      
      NETWORKING_IPV6=yes
   
   c. Add 2 lines to the /etc/sysconfig/network-scripts/ifcft-eth0 file:
      
      IPV6INIT=yes
   
   d. In /etc/sysctl.conf file, change the following parameter value to 0:
      
      net.ipv6.conf.all.disable_ipv6 = 0
   
   e. Restart network services.
      
      service network restart
2. To allow client computers to communicate with a computer that runs luci (the Conga user interface server), and to allow a computer that runs luci to communicate with ricci in the cluster nodes, you must enable the IP ports assigned to luci and ricci. Add the following **iptables** rules (in addition to the default mobile agent firewall hardening rules):

   ```
   iptables -I INPUT -p udp --state NEW -m multiport --dports 5404,5405 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 11111 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 14567 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 16851 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 21064 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 50006 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 50008 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 50009 -j ACCEPT
   iptables -I INPUT -p udp --state NEW -m multiport --dports 50007 -j ACCEPT
   iptables -I INPUT -p tcp --state NEW -m multiport --dports 8084 -j ACCEPT
   Then run the following command:
   ```
   ```
   service iptables save ; service iptables reload
   ```
   **Note:** This setup might be further secured by having additional source or destination restriction firewall rules.

3. Add the following lines to the `/etc/hosts` file on each mobile agent node:

   ```
   ...
   IP.addr.of.node1 node1
   IP.addr.of.node2 node2
   And delete the localhost alias:
   ```
   127.0.0.1 node1
   node1.localdomain localhost.localdomain
   localhost
   The IP address of each node is the one defined for eth0. The names will be used by the cluster manager facilities, as will be described later in this doc.
   **Note:** The names used here for clustering purposes (node1 and node2) may or may not be identical to those defined as the hostname while running the installation wizard. In case the same name should be used in both instances, and for both purposes, its first appearance in the file - resolving the loopback address (127.0.0.1) - should be removed. Be aware though that the loopback entry will be generated again whenever the user changes the hostname using the wizard tool.
4. Restore the original redhat-release file:
   
   ```
   cp /etc/redhat-release /etc/redhat-release.wbsns
   echo "CentOS release 5.9 (Final)" > /etc/redhat-release
   ```

**Installing supplemental packages on the node**

_**Conga**_ is an integrated set of software components that provides centralized configuration and management of Red Hat clusters and storage. _**Conga**_ provides the following major features:

- One Web interface for managing cluster and storage
- Automated Deployment of Cluster Data and Supporting Packages
- Easy Integration with Existing Clusters
- No Need to Re-Authenticate
- Integration of Cluster Status and Logs
- Fine-Grained Control over User Permissions

The primary components in _**Conga**_ are _**luci**_ and _**ricci**_, which are separately installable.

_**ricci**_ is an agent that runs on each computer (either a cluster member or a standalone computer) managed by _**Conga**_.

_**luci**_ is a server that runs on one computer and communicates with multiple clusters and computers via _**ricci**_.

1. To administer Red Hat Clusters with _**Conga**_, install and run _**luci**_ and _**ricci**_ as follows:
   a. At each node to be administered by _**Conga**_, install the _**ricci**_ agent.
      ```
      yum install ricci
      service ricci start
      ```
   b. Select a computer to host _**luci**_ and install the _**luci**_ software on that computer.
      ```
      yum install luci
      luci_admin init
      service luci restart
      ```

2. Install the supplemental CentOS packages on each mobile agent node using one of the following methods:
   a. If the mobile agent nodes have access to the Web, then use the _**yum**_ installation command:
      ```
      yum -y install package.name
      ```
      This command installs the packages considering all dependencies.
      **Note:** The _**yum**_ utility is a software management system that lets you install, update, or remove software from your system.
   b. If the mobile agent nodes do not have access to the Web, then the required packages should be downloaded manually to the nodes’ disk and then installed using the _**rpm**_ command. The set of packages is described in
Appendix 1 - Complete set of required packages, page 22. Installation command:

```
cd /downloaded/packages/folder
rpm -ivh package1.rpm package2.rpm...
```

Configuring the cluster

1. At a Web browser, place the URL of the luci server into the URL address box and click Go (or the equivalent). The URL syntax for the luci server is https://luci_server_hostname:8084. The first time you access luci, two SSL certificate dialog boxes are displayed. Upon acknowledging the dialog boxes, your Web browser displays the luci login page.
2. Create a new cluster (mdlp-cs in this sample), and add two nodes to the cluster using the same names used in the modified /etc/hosts file described above (node1 and node2 in this sample):
3. For both nodes, define manual fence (mdlpFencing in this sample):

- For every other node a previously defined fence should be selected from the drop list.
4. Create failover domain for the two nodes. Make it prioritized and restricted (*mdlp* in this sample):

![Failover Domain Configuration](image)

5. Create the following resources:
   - IP resource – this is a floating IP to be used for client’s connection that is always available on the active node (10.0.34.118 in this sample):

![Resource Configuration](image)
- Health script used to raise a flag when one of active’s node main services is down (script name in this sample: mdlp_rhcs, always pointing: /opt/websense/rproxy/cs/mdlp_rhcs):

![Image of CentOS Cluster and Storage Systems]

6. Create a service using the 2 resources defined in the previous step. Define the service name (mdlp in this sample):
   a. Select the following options: **Run Exclusive**, **Autostart**, and **Relocate** for the Recovery policy.
   b. Select the failover domain (as defined in step 4).
c. Add a resource to this service:
d. Enable the service:

Replacing instances in the cluster (on every node)

1. Replace the default manual fencing executable on the node running the cluster manager tool:
   
   ```
   ln -s /bin/true /sbin/fence_mdlp
   ```

   a. Open `/etc/cluster/cluster.conf` and search for the following line
   
   ```
   <fencedevice agent="fence_manual" name="MdlpFencing"/>
   ```

   b. Replace the following text: `fence_manual` to `fence_mdlp`. 
Note that this dummy fencing for mobile agent seems quite sufficient (as no real harm can occur in case the non-responding node is up while the previously passive node becomes active). However, one can use any custom mechanism by simply deploying a custom executable (i.e. install an executable in the /sbin folder and update the cluster.conf file accordingly) or by using one of the built-in fencing devices mechanisms provided by the cluster suite (this depends on the user’s physical and hardware conditions).

c. You may need to restart the cluster after modifying manually the cluster.conf file (see the next section).

Verifying the cluster

![CentOS Cluster and Storage Systems]

The green color indicates that the cluster is configured and ready to run.

Active/passive switch events

The cluster uses several signs in order to keep tracking its own state. These signs are eventually used to tell the cluster when something in the active node is broken. When this happens, then the cluster needs a passive node to take over.

The signs used by this cluster include:

1. Floating IP address becomes unavailable (including when the active node is rebooted).
2. Services are down (httpd_filter, httpd_broker, PolicyEngine and RProxyWD) as implemented by the /opt/websense/rproxy/cs/mdlp_rhcs script described above.
3. Machine stops delivering heartbeat traffic (i.e. the machine stops working suddenly).
When the broken, previously active, node is fully operational again, it will automatically rejoin the cluster. Depending on why the node went down, the cluster manager decides whether it will want to keep it passive or make it active again.
Cluster monitoring and maintenance

The cluster management interface Conga, also provides cluster monitoring capability that basically indicates which node is currently active.
Additionally, *Show recent log activity for this node* provides another method to track cluster activities.
Appendix 1 - Complete set of required packages

This is the entire set of dependent packages that are silently installed as part of the deployment.

**package: ricci.i386 0.12.2-81.el5.centos**
- dependency: libz.so.1
  - provider: zlib.i386 1.2.3-7.el5
- dependency: libgcc_s.so.1
  - provider: libgcc.i386 4.1.2-55.el5
- dependency: libstdc++.so.6
  - provider: libstdc++.i386 4.1.2-55.el5
- dependency: libm.so.6
  - dependency: libc.so.6(GLIBC_2.1.3)
  - dependency: libc.so.6(GLIBC_2.4)
  - dependency: libdl.so.2
  - dependency: libc.so.6(GLIBC_2.1)
  - dependency: libc.so.6(GLIBC_2.3.4)
  - dependency: libpthread.so.0(GLIBC_2.1)
  - dependency: libpthread.so.0(GLIBC_2.0)
  - dependency: libc.so.6(GLIBC_2.0)
  - dependency: libc.so.6(GLIBC_2.1.2)
  - provider: glibc.i686 2.5-123
  - provider: glibc.i386 2.5-123
  - provider: glibc.i386 2.5-123.el5_11.3
  - provider: glibc.i686 2.5-123.el5_11.1
  - provider: glibc.i686 2.5-123.el5_11.3
  - provider: glibc.i386 2.5-123.el5_11.1
  - dependency: libssl.so.6
  - dependency: libcrypto.so.6
    - provider: openssl.i386 0.9.8e-27.el5_10.4
    - provider: openssl.i686 0.9.8e-27.el5_10.4
    - provider: openssl.i386 0.9.8e-34.el5_11
    - provider: openssl.i686 0.9.8e-36.el5_11
    - provider: openssl.i686 0.9.8e-31.el5_11
    - provider: openssl.i386 0.9.8e-33.el5_11
    - provider: openssl.i686 0.9.8e-36.0.1.el5_11
    - provider: openssl.i386 0.9.8e-36.0.1.el5_11
    - provider: openssl.i686 0.9.8e-34.el5_11
    - provider: openssl.i386 0.9.8e-36.el5_11
    - provider: openssl.i686 0.9.8e-33.el5_11
    - provider: openssl.i386 0.9.8e-31.el5_11
Mobile DLP agent using cluster solutions

provider: openssl.i386 0.9.8e-32.el5_11
provider: openssl.i686 0.9.8e-32.el5_11
dependency: pam
provider: pam.i386 0.99.6.2-12.el5
provider: pam.i386 0.99.6.2-14.el5_11
dependency: libxml2.so.2
provider: libxml2.i386 2.6.26-2.1.21.el5_9.3
provider: libxml2.i386 2.6.26-2.1.25.el5_11
dependency: /bin/sh
provider: bash.i386 3.2-32.el5_9.1
provider: bash.i386 3.2-33.el5.1
provider: bash.i386 3.2-33.el5_11.4
dependency: oddjob
provider: oddjob.i386 0.27-12.el5
dependency: initscripts
provider: initscripts.i386 8.45.45-1.el5.centos
dependency: libsasl2.so.2
provider: cyrus-sasl-lib.i386 2.1.22-7.el5_8.1
dependency: parted
provider: parted.i386 1.8.1-30.el5
dependency: chkconfig
provider: chkconfig.i386 1.3.30.2-2.el5
dependency: grep
provider: grep.i386 2.5.1-55.el5
dependency: dbus
provider: dbus.i386 1.1.2-21.el5
dependency: libdbus-1.so.3
provider: dbus-libs.i386 1.1.2-21.el5
dependency: libgce_c.so.1(GCC_3.0)
provider: libgce_i386 4.1.2-55.el5
dependency: shadow-utils
provider: shadow-utils.i386 2:4.0.17-23.el5
dependency: libmagic.so.1
provider: file.i386 4.17-28
dependency: sed
provider: sed.i386 4.1.5-8.el5
dependency: libgce_c.so.1(GLIBC_2.0)
provider: libgce_i386 4.1.2-55.el5
dependency: libstdc++-so.6(CXXABI_1.3)
provider: libstdc++.i386 4.1.2-55.el5
dependency: cyrus-sasl >= 2.1
provider: cyrus-sasl.i386 2.1.22-7.el5_8.1
dependency: libstdc++-so.6(GLIBCXX_3.4)
provider: libstdc++.i386 4.1.2-55.el5
dependency: modcluster >= 0.12.0
package: luci.i386 0.12.2-81.el5.centos

dependency: libgcc_s.so.1
  provider: libgcc.i386 4.1.2-55.el5
  dependency: libstdc++.so.6(CXXABI_1.3.1)
  provider: libstdc++.i386 4.1.2-55.el5
dependency: util-linux
  provider: util-linux.i386 2.13-0.59.el5_8

package: luci.i386 0.12.2-81.el5.centos

dependency: libgcc.s.so.1
  provider: libgcc.i386 4.1.2-55.el5
  dependency: libstdc++.so.6
  provider: libstdc++.i386 4.1.2-55.el5
dependency: libc.so.6(GLIBC_2.1.3)
  dependency: libm.so.6
  dependency: libc.so.6(GLIBC_2.4)
  dependency: libc.so.6(GLIBC_2.1)
  dependency: libc.so.6(GLIBC_2.3.4)
  dependency: libpthread.so.0(GLIBC_2.1)
  dependency: libc.so.6
  dependency: libc.so.6(GLIBC_2.0)
  dependency: libc.so.6(GLIBC_2.1.2)
  dependency: libc.so.6(GLIBC_2.3)
  dependency: libpthread.so.0(GLIBC_2.0)
  provider: glibc.i686 2.5-123
  provider: glibc.i386 2.5-123
  provider: glibc.i386 2.5-123.el5_11.3
  provider: glibc.i686 2.5-123.el5_11.1
  provider: glibc.i686 2.5-123.el5_11.3
  provider: glibc.i386 2.5-123.el5_11.1
  provider: libssl.so.6
  dependency: openssl
  provider: openssl.i386 0.9.8e-27.el5_10.4
  provider: openssl.i686 0.9.8e-27.el5_10.4
  provider: openssl.i386 0.9.8e-34.el5_11
  provider: openssl.i686 0.9.8e-36.el5_11
  provider: openssl.i686 0.9.8e-31.el5_11
  provider: openssl.i386 0.9.8e-33.el5_11
  provider: openssl.i686 0.9.8e-36.0.1.el5_11
  provider: openssl.i386 0.9.8e-36.0.1.el5_11
  provider: openssl.i686 0.9.8e-34.el5_11
  provider: openssl.i386 0.9.8e-36.el5_11
  provider: openssl.i686 0.9.8e-33.el5_11
  provider: openssl.i386 0.9.8e-31.el5_11
  provider: openssl.i386 0.9.8e-32.el5_11
  provider: openssl.i686 0.9.8e-32.el5_11
dependency: /usr/bin/python
  provider: python.i386 2.4.3-56.el5
dependency: libxml2.so.2
  provider: libxml2.i386 2.6.26-2.1.21.el5_9.3
  provider: libxml2.i386 2.6.26-2.1.25.el5_11
dependency: /bin/sh
  provider: bash.i386 3.2-32.el5_9.1
  provider: bash.i386 3.2-33.el5.1
  provider: bash.i386 3.2-33.el5_11.4
dependency: initscripts
  provider: initscripts.i386 8.45.45-1.el5.centos
dependency: /usr/bin/env
  provider: coreutils.i386 5.97-34.el5_8.1
dependency: chkconfig
  provider: chkconfig.i386 1.3.30.2-2.el5
dependency: stunnel
  provider: stunnel.i386 4.15-2.el5.1
  provider: stunnel.i386 4.15-2.el5.2
dependency: grep
  provider: grep.i386 2.5.1-55.el5
dependency: python-imaging
  provider: python-imaging.i386 1.1.5-7.el5
dependency: mailcap
  provider: mailcap.noarch 2.1.23-1.fc6
dependency: python >= 2.4.1
  provider: python.i386 2.4.3-56.el5
dependency: libgcc_s.so.1(GCC_3.0)
  provider: libgcc.i386 4.1.2-55.el5
dependency: shadow-utils
  provider: shadow-utils.i386 2:4.0.17-23.el5
dependency: sed
  provider: sed.i386 4.1.5-8.el5
dependency: libstdc++.so.6(CXXABI_1.3)
  provider: libstdc++.i386 4.1.2-55.el5
dependency: bc
  provider: bc.i386 1.06-21
dependency: libstdc++.so.6(GLIBCXX_3.4)
  provider: libstdc++.i386 4.1.2-55.el5
dependency: libstdc++.so.6(CXXABI_1.3.1)
  provider: libstdc++.i386 4.1.2-55.el5
dependency: util-linux
  provider: util-linux.i386 2.13-0.59.el5_8
Appendix 2 - Sample cluster configuration file

Below is a sample cluster configuration file. This file can serve as a template. By modifying some values and storing it in the proper `/etc/cluster/cluster.conf` path, users can skip the cluster manager GUI utility steps described earlier in this document.

```xml
<?xml version="1.0"?>
<cluster config_version="11" name="mdlp-cluster">
    <fence daemon post_fail_delay="0" post_join_delay="3"/>
    <clusternode name="node1" nodeid="1" votes="1">
        <fence>
            <method name="1">
                <device name="MdlPfencing" nodename="node1"/>
            </method>
        </fence>
    </clusternode>
    <clusternode name="node2" nodeid="2" votes="1">
        <fence>
            <method name="1">
                <device name="MdlPfencing" nodename="node2"/>
            </method>
        </fence>
    </clusternode>
    <fencedevice>
        <max_expected_votes="1" two_node="1"/>
        <fencedevice agent="fence_manual" name="MdlPfencing"/>
    </fencedevice>
    <failoverdomains>
        <failoverdomain name="mdlp" ordered="1" restricted="1">
            <failoverdomainnode name="node1" priority="1"/>
        </failoverdomain>
        <failoverdomain name="mdlp" ordered="1" restricted="1">
            <failoverdomainnode name="node2" priority="1"/>
        </failoverdomain>
    </failoverdomains>
    <resources>
        <ip address="10.0.32.201" monitor_link="1"/>
        <script file="/opt/webcache/sproxy/c6/mdlp_shcs" name="mdlp_shcs"/>
    </resources>
    <service autostart="1" domain="mdlp" exclusive="1" name="mdlp">
        <ip ref="10.0.32.201"/>
        <script ref="mdlp_shcs"/>
        <recovery="relocate"/>
    </service>
</cluster>
```

Appendix 3 - Troubleshooting

**Error while creating a new cluster**

Check all the steps in **configuring the node agent** section. Make sure the **ricci** service is running on every node.

**Error while configuring the cluster**

Make sure the **rgmanager** service is running on every node.
Additionally, the cluster system dumps log records to /var/log/messages file, providing another method for troubleshooting the cluster.

### Debugging and Testing Services and Resource Ordering

You can debug and test services and resource ordering with the `rg_test` utility. `rg_test` is a command-line utility that is run from a shell or a terminal (it is not available in Conga or system-config-cluster.).

The following table summarizes the actions and syntax for the `rg_test` utility:

<table>
<thead>
<tr>
<th>Action</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| Display the resource rules that `rg_test` understands. Test a configuration (and /usr/share/cluster) for errors or redundant resource agents. | `rg_test rules`  
`rg_test test /etc/cluster/cluster.conf` |
| Display the start and stop ordering of a service. | Display start order:  
`rg_test noop /etc/cluster/cluster.conf start servicename`  
Display stop order:  
`rg_test noop /etc/cluster/cluster.conf stop servicename` |
| Explicitly start or stop a service. **Important** Only do this on one node, and always disable the service in rgmanager first. | Start a service:  
`rg_test test /etc/cluster/cluster.conf start servicename`  
Stop a service:  
`rg_test test /etc/cluster/cluster.conf stop servicename` |
| Calculate and display the resource tree delta between two cluster.conf files. | `rg_test delta cluster.conf file 1 cluster.conf file 2`  
For example:  
`rg_test delta /etc/cluster/cluster.conf.bak /etc/cluster/cluster.conf` |