Red Hat
Training and Certification

# Student Workbook (ROLE)

Red Hat Enterprise Linux 7.1 RH436

## Red Hat High Availability Clustering

Edition 2

RH436-RHEL7.1-en-2-20200501

# Red Hat High Availability Clustering

# Red Hat Enterprise Linux 7.1 RH436
# Red Hat High Availability Clustering
# Edition 2 20200501

Authors: Wander Boessenkool, Chen Chang, Michael Jarrett, Rudolf Kastl
Editor: Steven Bonneville



Contributors: Rob Locke, Bowe Strickland, Scott McBrien, George Hacker, Forrest Taylor, Snehangshu Karmakar

# Document Conventions

**References**

"References" describe where to find external documentation relevant to a subject.

**Note**

"Notes" are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

**Important**

"Important" boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss, but may cause irritation and frustration.

**Warning**

"Warnings" should not be ignored. Ignoring warnings will most likely cause data loss.

# Introduction

## Red Hat High Availability Clustering

*Red Hat® High Availability Clustering* (RH436) provides intensive, hands-on experience with the Pacemaker component of the Red Hat Enterprise Linux High-Availability Add-On, and cluster storage components from the Resilient Storage Add-On, including Cluster Logical Volume Manager (CLVM), Red Hat Global File System 2 (GFS2), and Device-Mapper Multipath. Created for senior Linux® system administrators, this 4-day course has a strong emphasis on lab-based activities. At the end of the course, students will have learned to deploy and manage shared storage and server clusters that provide highly available network services to a mission-critical enterprise environment.

---

### Objectives

- Design and deploy a high availability cluster to provide active/passive failover services.
- Prepare students for the Red Hat High Availability Clustering Certificate of Expertise Exam (EX436).

### Audience

- Senior Linux system administrators responsible for maximizing resiliency though high availability clustering services and using fault tolerant shared storage technologies.
- An RHCSA/RHCE interested in earning a Red Hat Certification of Expertise or an Red Hat Certified Architect (RHCA).

### Prerequisites

- The prerequisites for this class are a Red Hat Certified Engineer (RHCE) certification or equivalent experience.

# Orientation to the Classroom Lab Environment

In this course, students will do most hands-on practice exercises and lab work with five computer systems, which will be referred to as **workstation**, **nodea**, **nodeb**, **nodec** and **noded**. These machines have the host names **workstation.clusterX.example.com**, **nodea.clusterX.example.com**, **nodeb.clusterX.example.com**, **nodec.clusterX.example.com**, and **noded.clusterX.example.com**, where the number *X* in the computers' host names will be a number that will vary from student to student. Both machines have a standard user account, *student*, with the password *student*. The *root* password on both systems is *redhat*.

Next to the **clusterX.example.com** network, three other networks are also in use: **private.example.com** for private cluster communications, and **storage1.example.com** and **storage2.example.com** for iSCSI and NFS storage traffic.

The systems used by each student use separate IPv4 subnets for their **clusterX.example.com** network. For a specific student, their IPv4 network is 172.25.*X*.0/24, where the number *X* matches the number in the host name of their **desktop** and **server** systems.

The other three networks in use are all private networks, traffic on these networks will not be routed outside of the students **foundationX.example.com** machine.

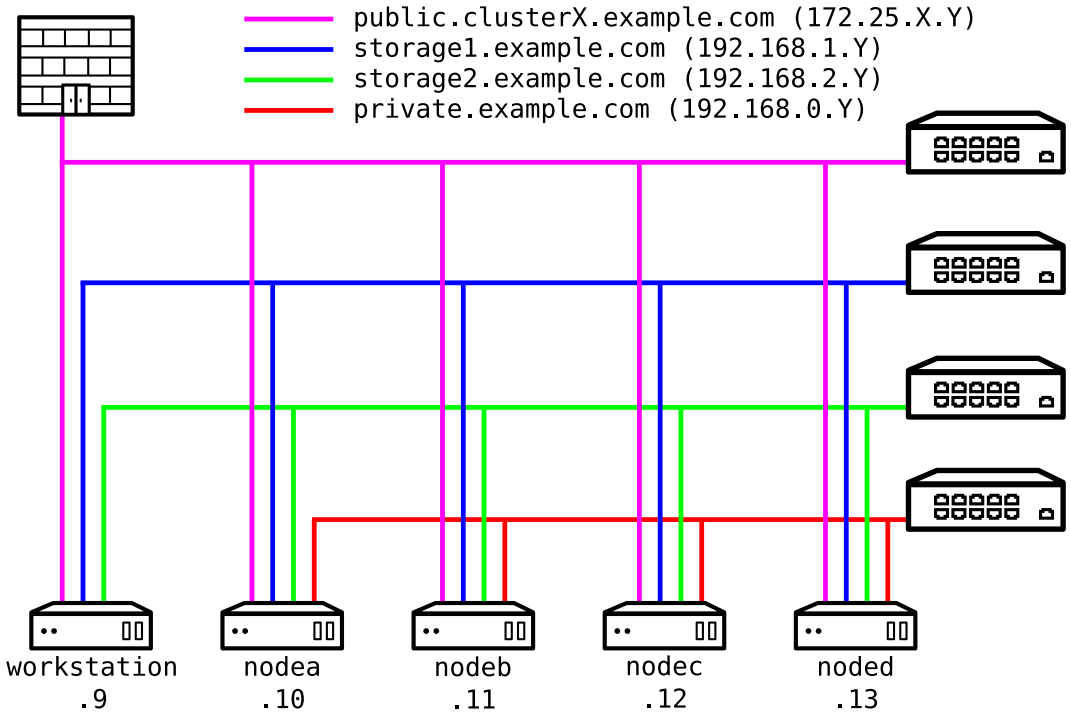



**Figure 0.1: The classroom layout**

In a Red Hat Online Learning classroom, students will be assigned remote computers which will be accessed through a web application hosted at rol.redhat.com. Students should log into this machine using the user credentials they provided when registering for the class.

## Controlling your stations

The top of the console describes the state of your machine.

### Machine States

| State | Description |
|---|---|
| none | Your machine has not yet been started. When started, your machine will boot into a newly initialized state (the desk will have been reset). |
| starting | Your machine is in the process of booting. |
| running | Your machine is running and available (or, when booting, soon will be.) |
| stopping | Your machine is in the process of shutting down. |
| stopped | Your machine is completely shut down. Upon starting, your machine will boot into the same state as when it was shut down (the disk will have been preserved). |
| impaired | A network connection to your machine cannot be made. Typically this state is reached when a student has corrupted networking or firewall rules. If the condition persists after a machine reset, or is intermittent, please open a support case. |

Depending on the state of your machine, a selection of the following actions will be available to you.

### Machine Actions

| Action | Description |
|---|---|
| Start Station | Start ("power on") the machine. |
| Stop Station | Stop ("power off") the machine, preserving the contents of its disk. |
| Reset Station | Stop ("power off") the machine, resetting the disk to its initial state. **Caution: Any work generated on the disk will be lost**. |
| Refresh | Refresh the page will re-probe the machine state. |
| Increase Timer | Adds 15 minutes to the timer for each click. |

## The station timer

Your Red Hat Online Learning enrollment entitles you to a certain amount of computer time. In order to help you conserve your time, the machines have an associated timer, which is initialized to 60 minutes when your machine is started.

The timer operates as a "dead man's switch," which decrements as your machine is running. If the timer is winding down to 0, you may choose to increase the timer.

# Internationalization

## Language support

Red Hat Enterprise Linux 7 officially supports 22 languages: English, Assamese, Bengali, Chinese (Simplified), Chinese (Traditional), French, German, Gujarati, Hindi, Italian, Japanese, Kannada, Korean, Malayalam, Marathi, Odia, Portuguese (Brazilian), Punjabi, Russian, Spanish, Tamil, and Telugu.

## Per-user language selection

Users may prefer to use a different language for their desktop environment than the system-wide default. They may also want to set their account to use a different keyboard layout or input method.

### Language settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the Region & Language application. Run the command **`gnome-control-center region`**, or from the top bar, select **(User)** → **Settings**. In the window that opens, select Region & Language. The user can click the **Language** box and select their preferred language from the list that appears. This will also update the **Formats** setting to the default for that language. The next time the user logs in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications, including **`gnome-terminal`**, started inside it. However, they do not apply to that account if accessed through an **`ssh`** login from a remote system or a local text console (such as **`tty2`**).

**Note**

A user can make their shell environment use the same **`LANG`** setting as their graphical environment, even when they log in through a text console or over **`ssh`**. One way to do this is to place code similar to the following in the user's **`~/.bashrc`** file. This example code will set the language used on a text login to match the one currently set for the user's GNOME desktop environment:

```
i=$(grep 'Language=' /var/lib/AccountService/users/${USER} \
  | sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanese, Korean, Chinese, or other languages with a non-Latin character set may not display properly on local text consoles.

Individual commands can be made to use another language by setting the **`LANG`** variable on the command line:

```
[user@host ~]$ LANG=fr_FR.utf8 date
jeu. avril 24 17:55:01 CDT 2014
```

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to check the current value of **LANG** and other related environment variables.

## Input method settings

GNOME 3 in Red Hat Enterprise Linux 7 automatically uses the IBus input method selection system, which makes it easy to change keyboard layouts and input methods quickly.

The Region & Language application can also be used to enable alternative input methods. In the Region & Language application's window, the **Input Sources** box shows what input methods are currently available. By default, **English (US)** may be the only available method. Highlight **English (US)** and click the **keyboard** icon to see the current keyboard layout.

To add another input method, click the **+** button at the bottom left of the **Input Sources** window. An **Add an Input Source** window will open. Select your language, and then your preferred input method or keyboard layout.

Once more than one input method is configured, the user can switch between them quickly by typing **Super**+**Space** (sometimes called **Windows**+**Space**). A *status indicator* will also appear in the GNOME top bar, which has two functions: It indicates which input method is active, and acts as a menu that can be used to switch between input methods or select advanced features of more complex input methods.

Some of the methods are marked with gears, which indicate that those methods have advanced configuration options and capabilities. For example, the Japanese **Japanese (Kana Kanji)** input method allows the user to pre-edit text in Latin and use **Down Arrow** and **Up Arrow** keys to select the correct characters to use.

US English speakers may find also this useful. For example, under **English (United States)** is the keyboard layout **English (international AltGr dead keys)**, which treats **AltGr** (or the right **Alt**) on a PC 104/105-key keyboard as a "secondary-shift" modifier key and dead key activation key for typing additional characters. There are also Dvorak and other alternative layouts available.

**Note**

Any Unicode character can be entered in the GNOME desktop environment if the user knows the character's Unicode code point, by typing **Ctrl**+**Shift**+**U**, followed by the code point. After **Ctrl**+**Shift**+**U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.

For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl**+**Shift**+**U**, then **03bb**, then **Enter**.

# System-wide default language settings

The system's default language is set to US English, using the UTF-8 encoding of Unicode as its character set (**en_US.utf8**), but this can be changed during or after installation.

From the command line, *root* can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it will display the current system-wide locale settings.

To set the system-wide language, run the command **`localectl set-locale LANG=`*`locale`***, where *locale* is the appropriate **`$LANG`** from the "Language Codes Reference" table in this chapter. The change will take effect for users on their next login, and is stored in **`/etc/locale.conf`**.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME, an administrative user can change this setting from Region & Language and clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the login screen will also adjust the system-wide default language setting stored in the **`/etc/locale.conf`** configuration file.

**Important**

Local text consoles such as **`tty2`** are more limited in the fonts that they can display than **`gnome-terminal`** and **`ssh`** sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a local text console. For this reason, it may make sense to use English or another language with a Latin character set for the system's text console.

Likewise, local text consoles are more limited in the input methods they support, and this is managed separately from the graphical desktop environment. The available global input settings can be configured through **`localectl`** for both local text virtual consoles and the X11 graphical environment. See the **`localectl`**(1), **`kbd`**(4), and **`vconsole.conf`**(5) man pages for more information.

# Language packs

When using non-English languages, you may want to install additional "language packs" to provide additional translations, dictionaries, and so forth. To view the list of available langpacks, run **`yum langavailable`**. To view the list of langpacks currently installed on the system, run **`yum langlist`**. To add an additional langpack to the system, run **`yum langinstall`** ***`code`***, where *code* is the code in square brackets after the language name in the output of **`yum langavailable`**.

**References**

**`locale`**(7), **`localectl`**(1), **`kbd`**(4), **`locale.conf`**(5), **`vconsole.conf`**(5), **`unicode`**(7), **`utf-8`**(7), and **`yum-langpacks`**(8) man pages

Conversions between the names of the graphical desktop environment's X11 layouts and their names in **`localectl`** can be found in the file **`/usr/share/X11/xkb/rules/base.lst`**.

# Language Codes Reference

### Language Codes

| Language | $LANG value |
|---|---|
| English (US) | en_US.utf8 |
| Assamese | as_IN.utf8 |

| Language | $LANG value |
|---|---|
| Bengali | bn_IN.utf8 |
| Chinese (Simplified) | zh_CN.utf8 |
| Chinese (Traditional) | zh_TW.utf8 |
| French | fr_FR.utf8 |
| German | de_DE.utf8 |
| Gujarati | gu_IN.utf8 |
| Hindi | hi_IN.utf8 |
| Italian | it_IT.utf8 |
| Japanese | ja_JP.utf8 |
| Kannada | kn_IN.utf8 |
| Korean | ko_KR.utf8 |
| Malayalam | ml_IN.utf8 |
| Marathi | mr_IN.utf8 |
| Odia | or_IN.utf8 |
| Portuguese (Brazilian) | pt_BR.utf8 |
| Punjabi | pa_IN.utf8 |
| Russian | ru_RU.utf8 |
| Spanish | es_ES.utf8 |
| Tamil | ta_IN.utf8 |
| Telugu | te_IN.utf8 |

Chapter 1

# Creating High-availability Clusters

**Goal**

Create a basic high-availability cluster.

**Objectives**

- Explain what a high-availability cluster is and when it should be used.
- Identify the components of a RHEL 7 high-availability cluster.
- Install and start a small high-availability cluster.

**Sections**

- High-availability Clustering (and Quiz)
- Architectural Overview (and Quiz)
- Configuring a Basic Cluster (and Practice)

**Lab**

- Creating High-availability Clusters

# High-availability Clustering

## Objectives

After completing this section, students should be able to explain what a high-availability cluster is and when it should be used.

## What is a cluster?

A *cluster* is a set of computers working together on a single task. Which task is performed, and how that task is performed, differs from cluster to cluster. There are two different kinds of clusters covered in this course.

- **High-availability** clusters: The goal of a high-availability cluster, also known as an *HA* cluster or *failover* cluster, is to keep running services as available as they can be. This is primarily achieved by having the nodes of the high-availability cluster monitor each other for failures, and migrating services to a node that is still considered "healthy" when a service or node fails. High-availability clusters can be grouped into two subsets:

  - Active-active high-availability clusters, where a service runs on multiple nodes, thus leading to shorter failover times.

  - Active-passive high-availability clusters, where a service only runs on one node at a time.

  High-availability clusters are often used to support mission-critical services in the enterprise. Examples of software that implement high-availability clustering are Pacemaker, and the Red Hat High Availability Add-On.

- **Storage** clusters: In a storage cluster, all members provide a single cluster file system that can be accessed by different server systems. The provided file system may be used to read and write data simultaneously. This is useful for providing high availability of application data, like web server content, without requiring multiple redundant copies of the same data. An example of a cluster file system is **GFS2**, which is provided by the Red Hat Resilient Storage Add-On.

**Note**

This course has a focus on high-availability clusters in an active-passive configuration (*cold failover*).

## What are the goals of a high-availability cluster?

The major goal of a high-availability cluster is to keep *services* as available as possible by eliminating bottlenecks and single points of failure. This is a different strategy than trying to keep the *uptime* for a single machine as high as possible. The uptime of the server running the service is not important for the consumers, but the service availability is. A high-availability cluster uses various concepts and techniques, which allow for service integrity and availability:

### Resources and resource groups

In clustering terminology, the basic unit of work is called a *resource*. A single IP address, filesystem or database would all be considered resources. Typically, relationships between these resources

are defined to create user-facing *services*. One of the most common ways to define these relationships is to combine a set of resources into a group. This specifies that all resources in the group need to run together on the same node and establishes a fixed (linear) start and stop order. For example, in order for a cluster to provide a web server service, the web server daemon, the data the server is supposed to share, and the IP address the daemon will listen on all need to be available on the same cluster node.

### Failover

High-availability clusters try to keep services available by migrating them to another *node* when the cluster notices that the node that was originally running the service is not responding; this is called a *failover*.

### Fencing

Fencing is a mechanism that ensures a malfunctioning cluster node can not cause corruption, so that its resources can be safely recovered elsewhere in the cluster. This is necessary because we cannot assume that an unreachable node is actually off. Fencing is often accomplished by powering the node off, since a dead node is clearly not able to do anything. In other cases, a combination of operations will be used to cut the node off from the network (to stop new work from arriving) or from storage (to stop the node from writing to shared storage).

### Shared storage

Most high-availability clusters will also need a form of *shared storage*, or storage that can be accessed from multiple nodes. Shared storage provides the same application data to multiple nodes in the cluster. The data may be accessed either sequentially or simultaneously by an application running on the cluster. A high-availability cluster needs to ensure data integrity on the shared storage. Data integrity is guaranteed by fencing.

### Quorum

Quorum describes a voting system that is required to maintain cluster integrity. Every cluster member has an assigned number of votes; by default, one vote. Depending on the cluster configuration, the cluster gains quorum when either half of the votes or more than half of the votes are present. Cluster members that fail to communicate with other cluster members and cannot send their votes are fenced by the majority of the cluster members that operate as expected. A cluster normally requires quorum to operate. If a cluster loses or cannot establish quorum, by default, no resources or resource groups are started up and running resource groups are stopped to ensure data integrity.

## When to use HA clustering?

When planning a high-availability cluster, there is one important question to answer: Will the availability of the service increase by putting it on an HA cluster?

To answer the question, it is important to know the capabilities of the service, and how the clients of the service can be configured:

- Depending on the solution, services such as **DNS** and **LDAP** with built-in failover or load balancing may not benefit from putting it on an HA cluster. For example, the **DNS** or **LDAP** services can use multiple servers with a master/slave or multimaster relation. The services can be configured for data replication between master and slave servers. Clients of **DNS** and **LDAP** can use multiple servers. There is less failover delay involved in a master/slave or multimaster configuration, so the availability of the service does not increase when putting them on an HA cluster. However, within an Openstack platform solution, it may be advantageous to put

resources such as RabbitMQ and Galera in an HA cluster. Further information concerning high availability in Red Hat Openstack Platform is outside the scope of this course.

- Services without built-in failover or load balancing can benefit from a high-availability cluster configuration. Examples include services like **`NFS`** and **`Samba`**.

Not every availability problem can be solved with high-availability clustering. Typically, problems that involve application crashes or network failures are not solved by a high-availability cluster:

- If a bug causes an application to crash when certain input is read, it will still crash if it is part of a high-availability cluster. In this case, the cluster will fail over the service to a different node, but if the same input is read again, the application will crash again.
- High-availability clusters also do not provide end-to-end redundancy. While the cluster itself may be fully operational, if a network error in the infrastructure causes the cluster to be unreachable, the clients will still not be able to reach the service, even though the service runs on a high-availability cluster. Therefore, it is important to think about the cluster's architecture and engineer it to avoid single points of failure throughout the deployment. There are tradeoffs here, and the cluster's architect will need to consider what level of risk they are prepared to tolerate with each component of the cluster.

**References**

Red Hat High Availability Add-On Overview

- Section 1: High Availability Add-On Overview

## Quiz

# Cluster Types

Match the following items to their counterparts in the table.

Compute

High-availability

Load balancing

Storage

| Use case | Cluster type |
|---|---|
| Keep services available, by using active/passive and/or active/active failover. | |
| Distribute network load for a single service across multiple nodes. Failed nodes can be ejected without affecting availability. | |
| Distribute larger calculations over multiple machines. | |
| Provide access to shared data for multiple clients, using multiple hosts. | |

## Solution

# Cluster Types

Match the following items to their counterparts in the table.

| Use case | Cluster type |
|---|---|
| Keep services available, by using active/passive and/or active/active failover. | High-availability |
| Distribute network load for a single service across multiple nodes. Failed nodes can be ejected without affecting availability. | Load balancing |
| Distribute larger calculations over multiple machines. | Compute |
| Provide access to shared data for multiple clients, using multiple hosts. | Storage |

# Architectural Overview

## Objectives

After completing this section, students should be able to identify the components of a Red Hat Enterprise Linux 7 high-availability (HA) cluster.

## Hardware

The following image shows a typical hardware configuration for a five-node HA cluster.

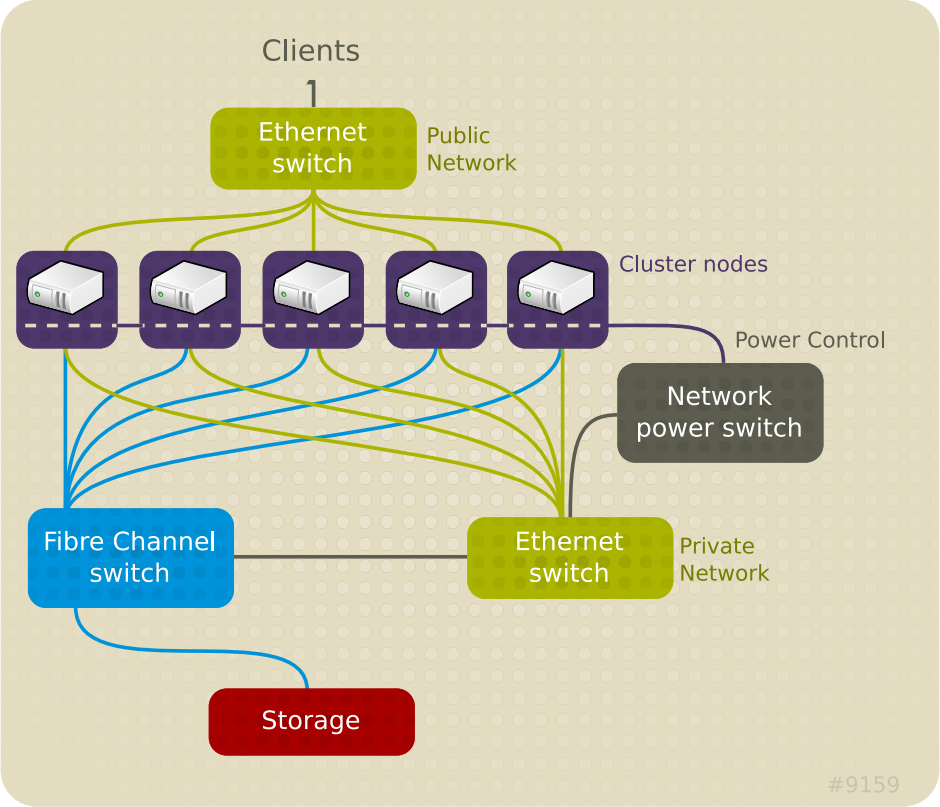



**Figure 1.1: A typical cluster infrastructure**

The different components in this infrastructure are as follows:

**Cluster Hardware**

| Hardware | Purpose |
|---|---|
| **Cluster Nodes** | These are the machines that will be running the cluster software and the services. |

| Hardware | Purpose |
|---|---|
| **Public Network** | This network is used for communication between the clients and the services running on the cluster. Services normally have a *floating* IP address, which means that the IP address can be assigned to whichever node is currently running the corresponding service. |
| **Private Network** | This network is used exclusively for cluster communications, and communication for critical cluster hardware such as networked power switches. |
| **Networked Power Switch** | A networked power switch can be used to remotely control power to the cluster nodes. This is one of the possible ways to implement *power fencing* as described later in this course. Remote management cards such as ILO or DRAC can also be used for this purpose. |
| **Fibre Channel Switch** | All nodes in the example are connected to the same shared storage. Although Fibre Channel is widely used for this, an alternative would be a separate Ethernet network with iSCSI or FCoE. |

In the image, only the components *above* the cluster nodes are publicly accessible. Everything below the cluster nodes is strictly *private*, and should not be reachable from the public network.

# Software

In order to provide cluster services with the Red Hat High Availability Add-on, multiple software components are required on the cluster nodes. An overview of the software and their functionality follows.

### Cluster Software

| Component | Purpose |
|---|---|
| **corosync** | This is the framework used by Pacemaker for handling communication between the cluster nodes. Corosync is also Pacemaker's source of membership and quorum data. |
| **pcs** | The *pcs* RPM package contains two cluster configuration tools:<br>1. The **pcs** command provides a command-line interface to create, configure, and control every aspect of a Pacemaker/Corosync cluster.<br>2. The **pcsd** service provides the cluster configuration synchronization and a web front end to create and configure a Pacemaker/Corosync cluster. |

| Component | Purpose |
|---|---|
| **pacemaker** | This is the component responsible for all cluster-related activities, such as monitoring cluster membership, managing the services and resources, and fencing cluster members. The *pacemaker* RPM package contains three important facilities:<br>1. Cluster Information Base (CIB): The Cluster Information Base contains configuration and status information about the cluster and the cluster resources in XML format. A cluster node in the cluster gets elected by Pacemaker to act as a designated coordinator (DC), and stores cluster and resource status and cluster configuration that gets synchronized to all other active cluster nodes.<br>2. Policy Engine (PEngine): The Policy Engine uses the contents of the cluster information base (CIB) to computer the ideal state of the cluster and how it should be reached.<br>3. Cluster Resource Management Daemon (CRMd): The Cluster Resource Management Daemon coordinates and sends the resource start, stop, and status query actions to the Local Resource Management Daemon (LRMd) that runs on every cluster node. The LRMd passes the actions received from the CRMd to the resource agents.<br>4. Shoot the Other Node in the Head (STONITH): STONITH is the facility responsible for processing fence requests and forwards the requested action to the fence device(s) configured in the CIB. |

# Requirements

Before deploying a high-availability cluster with the Red Hat High Availability Add-on, it is important to understand the requirements and supportability of the cluster configuration. Certain cluster configurations require an architecture review to get support from Red Hat. An architecture review process typically requires the transmission of relevant data about the cluster, such as the cluster configuration, network architecture, and fencing configuration to Red Hat Support. The support representative might request additional data if required. Red Hat Support will then decide if the cluster configuration can be supported by Red Hat.

There are important requirements and recommendations that should be considered before deploying a high-availability cluster based on the Red Hat High Availability Add-on.

## Number of nodes

Red Hat supports clusters with up to 16 nodes. Any cluster with eight or more nodes will need to be subjected to an architecture review to determine if the cluster setup is supported by Red Hat.

Clusters consisting of only two nodes are a special case. Although in most cases an architecture review is not mandatory, it is highly recommended to submit the architecture to Red Hat for a review before deploying a two-node cluster in production.

## Single site, multisite, and stretch clusters

Red Hat fully supports single-site clusters. This is a cluster setup where all cluster members are in the same physical location, connected by a local area network.

Multisite clusters consist of two clusters, one active and one for disaster recovery. Failover for multisite clusters must be managed manually. Multisite clusters are supported with the Red Hat Enterprise Linux 7 High Availability Add-on.

Stretch clusters, also known as geo clusters, are clusters stretched out over multiple physical locations. Stretch clusters must go through an architecture review process with Red Hat Support. It is possible that Red Hat Support will elect to not support the particular configuration, or impose restrictions on the level and type of support provided for the cluster. Typically, Red Hat will require a network latency of less than 2 ms for a stretched cluster to be deemed supportable. Stretch clusters require a quorum disk to be used when using four or more cluster nodes if a version prior to Red Hat Enterprise Linux 7 High Availability Add-on 7 is used.

## Fencing hardware

Fencing is a mechanism that ensures a malfunctioning cluster node can not cause corruption, so that its resources can be safely recovered elsewhere in the cluster. This can be done by power-cycling a node or disabling communication to the storage level. Fencing is required for all nodes in the cluster, either via power fencing, storage fencing, or a combination of both. Before deploying the high-availability infrastructure, ensure that supported hardware is used. If the cluster will use integrated fencing devices like ILO or DRAC, the systems acting as cluster nodes must power off immediately when a shutdown signal is received, instead of initiating a clean shutdown.

## Virtualization and clustering

The Red Hat High Availability Add-on supports virtual machines both as cluster resources and as cluster nodes.

When operating as cluster resources, the virtualization host is participating in a cluster and the virtual machine is a resource that can move between cluster nodes.

When operating as cluster nodes, the virtual machines running on a host are members of the cluster and run resources. Special fencing agents are available so these cluster nodes can fence each other, whether running on a RHEL 7 *libvirt*-based system, Red Hat Enterprise Virtualization, or other VM hypervisor hosts. In this case, the physical host is a single point of failure for all the virtual machine based cluster nodes running on that host. If the physical host crashes, it crashes all the cluster node VMs running on it.

## Networking

The **`corosync`** component requires unicast or multicast along with IGMP for the default network communication on the private network. In RHEL7, corosync/pacemaker clusters use unicast by default. For the public network, gratuitous ARP is used for the floating IP addresses. This must be supported by the network switch.

Any network ports used by the service(s) running on the cluster must be available on the public network. The following ports must be opened on the private network:

**Red Hat High Availability Add-on Network Ports**

| Network Port | Component |
|---|---|
| 5405/UDP | **`corosync`** |
| 2224/TCP | **`pcsd`** |
| 3121/TCP | **`pacemaker remote`** |

| Network Port | Component |
|---|---|
| 21064/TCP | **dlm** (used with GFS2 resources) |

In addition, port 5404/UDP may be used by **corosync** as a source for some cluster communication messages.

## SELinux support

The use of SELinux in **enforcing** mode is fully supported when using the **targeted** policy on the cluster nodes.

**References**

**What information is required for an Architecture Review of Red Hat Enterprise Linux High Availability and Resilient Storage?**
https://access.redhat.com/solutions/125153

**Support for Red Hat Enterprise Linux Cluster and High Availability Stretch Architectures**
https://access.redhat.com/knowledge/articles/27136

**Fence Device and Agent Information for Red Hat Enterprise Linux**
https://access.redhat.com/knowledge/articles/28603

## Quiz

# Cluster Architecture

Choose the correct answer to the following questions:

**1. What is the maximum number of cluster nodes supported with the Red Hat High Availability Add-On?**

a. 4
b. 8
c. 16
d. 32

**2. Which two types of fencing are supported? (Choose two.)**

a. No fencing
b. Power fencing
c. Storage (fabric) fencing
d. Manual fencing

**3. SELinux must be disabled on all cluster nodes.**

a. True
b. False

**4. An architecture review is always required when deploying a stretch cluster.**

a. True
b. False

**5. If in doubt that the proposed cluster architecture is supported, the best plan of action is to...**

a. Contact Red Hat for an architecture review.
b. Deploy it anyway.
c. Discontinue the cluster project.

## Solution

# Cluster Architecture

Choose the correct answer to the following questions:

**1. What is the maximum number of cluster nodes supported with the Red Hat High Availability Add-On?**

a. 4
b. 8
c. 16
d. 32

**2. Which two types of fencing are supported? (Choose two.)**

a. No fencing
b. Power fencing
c. Storage (fabric) fencing
d. Manual fencing

**3. SELinux must be disabled on all cluster nodes.**

a. True
b. False

**4. An architecture review is always required when deploying a stretch cluster.**

a. True
b. False

**5. If in doubt that the proposed cluster architecture is supported, the best plan of action is to...**

a. Contact Red Hat for an architecture review.
b. Deploy it anyway.
c. Discontinue the cluster project.

# Configuring a Basic Cluster

## Objectives

After completing this section, students should be able to install and start a small high-availability cluster.

## Installing the node software

The Red Hat High Availability Add-on requires installation of the required set of software packages, configuration of the firewall, and authentication of nodes.

**Important**

Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux 6 cluster nodes are not compatible. All nodes in a Pacemaker cluster must use the same major version of Red Hat Enterprise Linux; nodes running different major versions may not be mixed.

Red Hat Enterprise Linux 7 clusters use Corosync 2.x for communication, while Red Hat Enterprise Linux 6 Pacemaker clusters use Corosync 1.x.

### Installing required software on the node

The cluster configuration software is provided by the *pcs* package. The *pcs* package requires the *corosync* and *pacemaker* packages, which are automatically installed as dependencies if the installation is performed with **yum**. The fencing agents need to be installed on each of the cluster nodes. The *fence-agents-all* package pulls in all fencing agent packages that are available. Administrators can also choose to just install the *fence-agents-XXX* package, where *XXX* is the fencing agent they intend to use. The *pcs* and *fence-agents-all* packages need to be installed on all cluster nodes.

**Important**

The classroom requires the use of the *fence-agents-rht* package, a custom fencing agent for Red Hat classrooms. This package is not a part of the *fence-agents-all* package and will need to be installed explicitly.

```
[root@nodeY ~]# yum install pcs fence-agents-all
```

### Configuring a firewall for cluster communication

Cluster communication needs to be allowed by the firewall on all cluster nodes. The standard firewall service on a Red Hat Enterprise Linux 7 system is **firewalld**. The **firewalld** service comes with a standard service called **high-availability** to allow cluster communication. To activate the **high-availability** firewall configuration on each of the cluster nodes to allow cluster communication through the firewall, execute:

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

## Enabling pcsd cluster configuration of the node

The **pcsd** service provides the cluster configuration synchronization and the web front end for cluster configuration. The service is required on all cluster nodes. The **pcsd** service needs to be enabled and started on all cluster nodes with **systemctl**.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

The system user **hacluster** is used by **pcsd** for cluster communication and configuration. It is recommended to use the same password for the **hacluster** user among all nodes in the cluster. The password of the **hacluster** system user needs to be set on all cluster nodes.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
```

The **pcsd** service requires the cluster nodes to get authenticated with the user **hacluster** and the password of this user on one of the cluster nodes.

The cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com** are authenticated on the **nodea.private.example.com** system with the user **hacluster** and the corresponding password.

```
[root@nodea ~]# pcs cluster auth \
nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com
Username: hacluster
Password: password
nodea.private.example.com: Authorized
nodeb.private.example.com: Authorized
nodec.private.example.com: Authorized
```

For automation purposes, the **-u *<USERNAME>*** and **-p *<PASSWORD>*** can be used as well.

# Configuring basic cluster communication

After preparing the three nodes for the cluster setup, the cluster can be created with the **pcs cluster setup** command. A cluster name needs to be provided with **--name *cluster-name***, followed by the fully qualified domain names of the cluster nodes or their IP addresses. The optional parameter **--start** starts the cluster on all supplied cluster nodes.

```
[root@nodea ~]# pcs cluster setup --start --name mycluster \
nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com
```

By default, a cluster node that gets rebooted will not automatically join the cluster again. Automatic start of the cluster service can be enabled with the command **`pcs cluster enable`**. The option **`--all`** turns on automatic start of cluster services on every cluster member.

The following command allows all cluster nodes to start the cluster service and automatically join the cluster when executed on one of the cluster nodes.

```
[root@nodea ~]# pcs cluster enable --all
```

It is recommended to verify that the cluster is working as expected. The command **`pcs cluster status`** provides an overview of the current cluster status.

```
[root@nodea ~]# pcs cluster status
Cluster Status:
 Last updated: Mon Sep 15 05:41:18 2014
 Last change: Mon Sep 15 05:41:03 2014 via crmd on nodea.private.example.com
 Stack: corosync
 Current DC: nodea.private.example.com (1) - partition with quorum
 Version: 1.1.10-29.el7-368c726
 3 Nodes configured
 0 Resources configured

PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
```

# Configuring cluster node fencing

Fencing is a requirement for any high-availability cluster. Fencing prevents data corruption from an errant node. Fencing also isolates and restarts a cluster member if the node fails to join the cluster and the remaining cluster members are still quorate. Depending on the hardware used, it can fence a node by turning off the network connection to the shared storage or by power-cycling the node.

The first step to set up fencing is to set up the physical fencing device. There are different hardware devices that are capable of fencing cluster nodes, such as:

- Uninterruptible power supply (UPS)
- Power distribution unit (PDU)
- Blade power control devices
- Lights-out devices

**Note**

In the classroom a custom fencing agent is used: **`fence_rht`**. This fencing agent uses a special fencing server on **`classroom.example.com`**, and is inspired by the **`fence_rhevm`** fencing agent.

The fence devices need to be added to the cluster. For virtual machine fencing, each cluster node requires its own fence device. This gets accomplished with the **`pcs stonith create`** command. The command expects a set of parameter and value pairs required by the fence agent to be able

to fence the cluster node. To use the virtual machine fencing agent **fence_rht**, the parameters **port** and **pcmk_host_list** are required, as well as the parameter **ipaddr**. The **port** parameter expects the name of the virtual machine as a value. The parameter **pcmk_host_list** lists the corresponding host as it is known by the cluster:

```
[root@nodea ~]# pcs stonith create fence_device_name fence_rht
 port="node_private_fqdn" pcmk_host_list="node_private_fqdn"
 ipaddr="classroom.example.com"
```

The fence devices need to be added for the virtual machines **nodea**, **nodeb**, and **nodec** of the cluster from one of the cluster nodes:

```
[root@nodea ~]# pcs stonith create fence_nodea_virt fence_rht port="nodea"
 pcmk_host_list="nodea.private.example.com" ipaddr="classroom.example.com"
[root@nodea ~]# pcs stonith create fence_nodeb_virt fence_rht port="nodeb"
 pcmk_host_list="nodeb.private.example.com" ipaddr="classroom.example.com"
[root@nodea ~]# pcs stonith create fence_nodec_virt fence_rht port="nodec"
 pcmk_host_list="nodec.private.example.com" ipaddr="classroom.example.com"
```

The **pcs stonith show** command shows the status of the fence devices that are attached to the cluster. All **fence_rht** fence devices should show a status of **Started**.

```
[root@nodea ~]# pcs stonith show
 fence_nodea_rht (stonith:fence_rht): Started
 fence_nodeb_rht (stonith:fence_rht): Started
 fence_nodec_rht (stonith:fence_rht): Started
```

If the status of one or more fence devices is **Stopped**, there is most likely a problem in the communication between the fencing agent and the fencing server. Verify the settings of the fence device with **pcs stonith show *fence_device***. Settings can be updated with the command **pcs stonith update**.

**References**

**pcs**(8) man pages

*Clusters from Scratch* (Pacemaker 1.1 for Corosync 2.x and pcs), available in the *pacemaker-doc* package or at
http://clusterlabs.org/doc/

Additional information may be available in the document *Red Hat High Availability Add-On Administration* for Red Hat Enterprise Linux 7, and the document *Configuring the Red Hat High Availability Add-On with Pacemaker* for Red Hat Enterprise Linux 6, which can be found at
http://docs.redhat.com/

## Guided Exercise

# Configuring a Basic Cluster

In this lab, you will configure a basic three-node cluster

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to create a working cluster infrastructure with fencing.

- Reset your **nodea**, **nodeb**, **nodec**, and **workstation** machines.
- To make working on the various nodes easier, run the command **lab setupsshkeys setup** as **student** on your **workstation** machine. This will create an SSH key pair and install the public key on the **root** account of your various **node** machines. The script will also try to install the key on your **noded** machine, which may result in a failure for that machine if it is currently turned off.

```
[student@workstation ~]$ lab setupsshkeys setup
```

1. Prepare the virtual machines **nodea**, **nodeb**, and **nodec** to act as cluster nodes. Allow the cluster software to communicate through the firewall. Install, enable, and start the required software.

   1.1. Allow cluster communications to pass through the firewall on every cluster node.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

   1.2. Install the *pcs* and *fence-agents-rht* RPM packages on **nodea**, **nodeb**, and **nodec**.

```
[root@nodeY ~]# yum -y install pcs fence-agents-rht
```

   1.3. Enable and start the **pcsd** service on your **nodea**, **nodeb**, and **nodec** systems.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

   1.4. Change the password of the **hacluster** system user to **redhat** on your **nodea**, **nodeb**, and **nodec** machines.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
passwd: all authentication tokens updated successfully.
```

1.5. Authenticate the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com** on **nodea.private.example.com**.

```
[root@nodea ~]# pcs cluster auth nodea.private.example.com \
> nodeb.private.example.com \
> nodec.private.example.com
Username: hacluster
Password: redhat
nodea.private.example.com: Authorized
nodeb.private.example.com: Authorized
nodec.private.example.com: Authorized
```

**2.** Configure and start a three-node cluster named **clusterX**, using your **nodea**, **nodeb**, and **nodec** machines, using their host names on the **private.example.com** network.

2.1. Create and start the cluster **clusterX** with the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com** on **nodea.private.example.com**.

```
[root@nodea ~]# pcs cluster setup --start --name clusterX \
nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com
```

2.2. Enable automatic startup of the cluster on all configured cluster nodes.

```
[root@nodea ~]# pcs cluster enable --all
```

2.3. Verify the cluster is running and all configured cluster nodes have joined the cluster.

```
[root@nodea ~]# pcs status
Cluster name: clusterX
WARNING: no stonith devices and stonith-enabled is not false
Last updated: Mon Sep 15 05:41:18 2014
Last change: Mon Sep 15 05:41:03 2014 via crmd on nodea.private.example.com
Stack: corosync
Current DC: nodea.private.example.com (1) - partition with quorum
Version: 1.1.10-29.el7-368c726
3 Nodes configured
0 Resources configured


Online: [ nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com ]

Full list of resources:


PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
```

```
Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

3. Configure fencing for the virtual machines **nodea**, **nodeb**, and **nodec** that act as cluster nodes in the **clusterX** cluster. Use the fence_rht fencing agent, with **classroom.example.com** as the fencing device, and the host name of the machine to be fenced on the **private** network as the *plug*.

   3.1. Add the fence devices for the virtual machines **nodea**, **nodeb**, and **nodec** to the cluster on **nodea.private.example.com**.

```
[root@nodea ~]# pcs stonith create fence_nodea_rht \
> fence_rht port="nodea.private.example.com" \
> pcmk_host_list="nodea.private.example.com" \
> ipaddr="classroom.example.com"
```

```
[root@nodea ~]# pcs stonith create fence_nodeb_rht \
> fence_rht port="nodeb.private.example.com" \
> pcmk_host_list="nodeb.private.example.com" \
> ipaddr="classroom.example.com"
```

```
[root@nodea ~]# pcs stonith create fence_nodec_rht \
> fence_rht port="nodec.private.example.com" \
> pcmk_host_list="nodec.private.example.com" \
> ipaddr="classroom.example.com"
```

   3.2. Verify the fence devices have been added correctly to the **clusterX** cluster.

```
[root@nodea ~]# pcs stonith show
 fence_nodea_rht (stonith:fence_rht): Started
 fence_nodeb_rht (stonith:fence_rht): Started
 fence_nodec_rht (stonith:fence_rht): Started
```

   3.3. From your **nodea** machine, use the **pcs stonith fence** command to fence your **nodeb** machine. You can view the console of your **nodeb** machine to verify that a reboot is happening.

```
[root@nodea ~]# pcs stonith fence nodeb.private.example.com
```

## Lab

# Creating High Availability Clusters

### Performance Checklist

In this lab, you will configure a basic three-node cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to create a working three-node cluster named **`clusterX`**.

- Reset **nodea**, **nodeb**, and **nodec**.

Create a new cluster with the following requirements:

- The name of the new cluster is **`clusterX`**.
- The cluster consists of three nodes: **`nodea.private.example.com`**, **`nodeb.private.example.com`**, and **`nodec.private.example.com`**.
- The firewall allows cluster communication on all cluster nodes.
- Each of the cluster nodes automatically joins the cluster after reboot.
- The cluster nodes are configured to use the **`fence_rht`** classroom fencing agent. The fencing device is reachable at **`classroom.example.com`**, and the plug names are **`nodea.private.example.com`**, **`nodeb.private.example.com`**, and **`nodec.private.example.com`**.

1. Prepare the virtual machines **nodea**, **nodeb**, and **nodec** to act as cluster nodes. Allow the cluster software to communicate through the firewall. Install, enable, and start the required software.
2. Configure and start a three-node cluster named **`clusterX`**, using your **nodea**, **nodeb**, and **nodec** machines, using their host names on the **`private.example.com`** network.
3. Configure fencing for the virtual machines **nodea**, **nodeb**, and **nodec** that act as cluster nodes in the **`clusterX`** cluster. Use the fence_rht fencing agent, with **`classroom.example.com`** as the fencing device, and the host name of the machine to be fenced on the **`private`** network as the *plug*.

## Solution

# Creating High Availability Clusters

### Performance Checklist

In this lab, you will configure a basic three-node cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to create a working three-node cluster named **`clusterX`**.

- Reset **nodea**, **nodeb**, and **nodec**.

Create a new cluster with the following requirements:

- The name of the new cluster is **`clusterX`**.
- The cluster consists of three nodes: **`nodea.private.example.com`**, **`nodeb.private.example.com`**, and **`nodec.private.example.com`**.
- The firewall allows cluster communication on all cluster nodes.
- Each of the cluster nodes automatically joins the cluster after reboot.
- The cluster nodes are configured to use the **`fence_rht`** classroom fencing agent. The fencing device is reachable at **`classroom.example.com`**, and the plug names are **`nodea.private.example.com`**, **`nodeb.private.example.com`**, and **`nodec.private.example.com`**.

**1.** Prepare the virtual machines **nodea**, **nodeb**, and **nodec** to act as cluster nodes. Allow the cluster software to communicate through the firewall. Install, enable, and start the required software.

1.1. Allow cluster communications to pass through the firewall on every cluster node.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

1.2. Install the *pcs* and *fence-agents-rht* RPM packages on **`nodea.private.example.com`**, **`nodeb.private.example.com`**, and **`nodec.private.example.com`**.

```
[root@nodeY ~]# yum install -y pcs fence-agents-rht
```

1.3. Enable and start the **pcsd** service on your **nodea**, **nodeb**, and **nodec** systems.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

1.4. Change the password of the **hacluster** system user to **redhat** on your **nodea**, **nodeb**, and **nodec** machines.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
passwd: all authentication tokens updated successfully.
```

1.5. Authenticate the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com** on **nodea.private.example.com**.

```
[root@nodea ~]# pcs cluster auth nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com
Username: hacluster
Password: redhat
nodea.private.example.com: Authorized
nodeb.private.example.com: Authorized
nodec.private.example.com: Authorized
```

**2.** Configure and start a three-node cluster named **clusterX**, using your **nodea**, **nodeb**, and **nodec** machines, using their host names on the **private.example.com** network.

2.1. Create and start the cluster **clusterX** with the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com** on **nodea.private.example.com**.

```
[root@nodea ~]# pcs cluster setup --start --name clusterX \
nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com
```

2.2. Enable automatic startup of the cluster on all configured cluster nodes.

```
[root@nodea ~]# pcs cluster enable --all
```

2.3. Verify the cluster is running and all configured cluster nodes have joined the cluster.

```
[root@nodea ~]# pcs status
Cluster name: clusterX
WARNING: no stonith devices and stonith-enabled is not false
Last updated: Mon Sep 15 05:41:18 2014
Last change: Mon Sep 15 05:41:03 2014 via crmd on nodea.private.example.com
Stack: corosync
Current DC: nodea.private.example.com (1) - partition with quorum
Version: 1.1.10-29.el7-368c726
3 Nodes configured
0 Resources configured
```

```
Online: [ nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com ]

Full list of resources:


PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

3. Configure fencing for the virtual machines **nodea**, **nodeb**, and **nodec** that act as cluster nodes in the **clusterX** cluster. Use the fence_rht fencing agent, with **classroom.example.com** as the fencing device, and the host name of the machine to be fenced on the **private** network as the *plug*.

   3.1. Add the fence devices for the virtual machines **nodea**, **nodeb**, and **nodec** to the cluster on **nodea.private.example.com**.

```
[root@nodea ~]# pcs stonith create fence_nodea_rht \
fence_rht port="nodea.private.example.com" \
pcmk_host_list="nodea.private.example.com" \
ipaddr="classroom.example.com"
[root@nodeb ~]# pcs stonith create fence_nodeb_rht \
fence_rht port="nodeb.private.example.com" \
pcmk_host_list="nodeb.private.example.com" \
ipaddr="classroom.example.com"
[root@nodec ~]# pcs stonith create fence_nodec_rht \
fence_rht port="nodec.private.example.com" \
pcmk_host_list="nodec.private.example.com" \
ipaddr="classroom.example.com"
```

   3.2. Verify the fence devices have been added correctly to the **clusterX** cluster.

```
[root@nodea ~]# pcs stonith show
 fence_nodea_rht (stonith:fence_rht): Started
 fence_nodeb_rht (stonith:fence_rht): Started
 fence_nodec_rht (stonith:fence_rht): Started
```

   3.3. From your **nodea** machine, use the **pcs stonith fence** command to fence your **nodeb** machine. You can view the console of your **nodeb** machine to verify that a reboot is happening.

```
[root@nodea ~]# pcs stonith fence nodeb.private.example.com
```

# Summary

In this chapter, you learned:

- The difference between high-availability, compute, storage, and load balancing clusters.
- The required hard- and software for running a Red Hat high-avalability cluster.
- Support requirements for running a Red Hat high-availability cluster.
- How to configure a basic cluster using **pcs**.

Chapter 2

# Managing Cluster Nodes and Quorum

**Goal**

Manage cluster membership and quorum voting.

**Objectives**

- Change whether nodes are active members of a high-availability cluster on a temporary or permanent basis.
- Explain how quorum operates and protects a cluster from errors.
- Adjust the way quorum is determined to allow special configurations.

**Sections**

- Managing Cluster Membership (and Practice)
- Quorum Operation (and Practice)
- Managing Quorum Calculations (and Practice)

**Lab**

- Manage Cluster Nodes and Quorum

# Managing Cluster Membership

## Objectives

After completing this section, students should be able to change whether nodes are active members of a high-availability cluster on a temporary or permanent basis.

## Managing cluster membership

The Red Hat High Availability Add-on provides different ways for a system administrator to control membership of cluster nodes:

- Starting and stopping the cluster services control whether a cluster node is participating in the cluster at runtime.
- Enabling and disabling the cluster services control whether a cluster node automatically starts the cluster services and joins the cluster when it boots.
- Adding and removing a cluster node from the cluster permanently changes whether the node is a member of the cluster.
- The **standby** and **unstandby** modes control whether a cluster node is allowed to host resources in the cluster.

## Managing cluster services

The **systemd**-managed cluster services **corosync** and **pacemaker** need to be running on the cluster node to join the cluster the node is part of. On an authenticated cluster node that runs **pcsd**, the cluster services can be controlled with **pcs**.

### Starting and stopping cluster services

The **pcs cluster start** and **pcs cluster stop** commands start and stop the cluster services, respectively. The commands support starting or stopping cluster services on the local node, a remote node that is provided as a parameter, or on all nodes with the **--all** switch. To start the cluster services on the local node, execute:

```
[root@nodeY ~]# pcs cluster start
```

To stop cluster services on the remote node, **archer.example.com**, execute:

```
[root@nodeY ~]# pcs cluster stop archer.example.com
```

The **pcs cluster start --all** and **pcs cluster stop --all** commands start or stop all nodes that are in the same cluster as the local node. To stop the cluster services on all nodes in the same cluster as the local node, execute:

```
[root@nodeY ~]# pcs cluster stop --all
```

## Enable and disable cluster services

The **pcs cluster enable** and **pcs cluster disable** commands enable or disable the **systemd**-managed cluster services **corosync** and **pacemaker** from starting or stopping automatically when a cluster node boots up. A node automatically joins the cluster each time it is booted up if the cluster services are enabled on the node. Both commands control the services on the local node, a remote node that is provided as a parameter, or on all nodes in the same cluster as the local node with the **--all** switch. To enable automatic startup of the cluster services on the local node, execute:

```
[root@nodeY ~]# pcs cluster enable
```

To disable cluster services on the remote node, **lana.example.com**, execute:

```
[root@nodeY ~]# pcs cluster disable lana.example.com
```

The **pcs cluster enable --all** and **pcs cluster disable --all** commands enable and disable the cluster services on all nodes respectively that are in the same cluster as the local node. To disable the cluster services on all nodes in the same cluster as the local node, execute:

```
[root@nodeY ~]# pcs cluster disable --all
```

# Adding and removing a cluster node

The Red Hat High Availability Add-on allows for adding and removing cluster nodes on the fly. This allows for extending a cluster or replacing a cluster node without service downtime on the cluster. The **pcs** configuration utility allows a system administrator to add or remove cluster nodes. With **pcs cluster node add *node.fqdn***, a new node is added to the cluster, and with **pcs cluster node remove *node.fqdn***, a node is permanently removed.

## Adding a new node to the cluster

Adding a new node to an existing cluster requires multiple configuration steps:

1. The new node that will join the cluster has to be configured to fulfill the following requirements:
   - The firewall on the new cluster node is configured to allow cluster communication.
   - The *pcs* and *fence-agents-all* packages and their dependencies are installed. In the classroom environment, the package *fence-agents-rht* needs to be used in place of the *fence-agents-all* package.
   - The **pcsd** service is started and enabled.
   - The password of the **hacluster** user is changed to match the password of that user on the existing nodes.
   - The new cluster node was authenticated from all existing cluster members. This requires running **pcs cluster auth newnode.fqdn** from an existing cluster node. Unless the **--local** is used, the authentication token will be distributed across all cluster nodes automatically.
2. Once configured, the new cluster node can be added as a cluster member to the existing cluster and then authenticated with the previously existing cluster nodes.

- To add the prepared cluster node to the existing cluster, execute:

```
[root@oldnode1 ~]# pcs cluster node add newnode.fqdn
oldnode1.fqdn: Corosync updated
oldnode2.fqdn: Corosync updated
oldnode3.fqdn: Corosync updated
newnode.fqdn: Succeeded
```

- The existing nodes need to be authorized from the new node. The **pcs cluster auth** command authorizes all cluster members from the local node.

```
[root@newnode ~]# pcs cluster auth
Username: hacluster
Password: redhat
oldnode1.fqdn: Authorized
oldnode2.fqdn: Authorized
oldnode3.fqdn: Authorized
newnode.fqdn: Already authorized
```

After the node was successfully added and authorized, a system administrator may enable and start the cluster services on the new node. Fencing must be configured and operational for the new cluster node before any services can be safely migrated to it.

**Note**

Unless **pcs cluster start newnode.fqdn** and/or **pcs cluster enable newnode.fqdn** are used, the new node will not become an active member of the cluster.

## Removing a node from the cluster

A cluster node can be permanently removed from the cluster. This is useful, for example, if the cluster does not require the additional node for operation or if the hardware of the cluster node needs to be replaced by a new system.

Removing a cluster node from an existing cluster is a two-step process:

1. Remove the cluster node from the cluster.

```
[root@oldnode1 ~]# pcs cluster node remove newnode.fqdn
```

2. Adjust the fencing configuration by either removing the dedicated fence device or by reconfiguring the shared fence device to reflect that one node was removed.

```
[root@oldnode1 ~]# pcs stonith delete fence_deletednode
```

# Prohibiting a cluster node from hosting resources

There are times when an administrator needs to temporarily suspend resource hosting of a cluster node without disrupting regular cluster operation. This can happen, for example, when a critical security update needs to be applied for the hosted resource. The update can be applied after putting node by node into **standby** mode, resulting in a reduced downtime. A different use case

is to test resource migration. When a node is put in **standby** mode, it does not get any resources assigned. Resources that are currently running on the node are migrated to another node.

The **pcs cluster standby** command can put the local node in **standby** mode. The **pcs cluster standby** command can put a remote node that is provided as a parameter or all nodes with the **--all** option in **standby** mode. To put the remote cluster member, **archer.example.com**, into **standby** mode, run:

```
[root@nodeY ~]# pcs cluster standby archer.example.com
```

When using the **--all** switch, all nodes in the cluster are put in **standby** mode.

```
[root@nodeY ~]# pcs cluster standby --all
```

The *resource constraint* that has been applied by putting a node in **standby** can be removed with **pcs cluster unstandby**. Without additional options or parameters, the resource constraint is removed from the local node. A remote cluster member can be provided as a parameter, or the **--all** switch to allow the hosting of resources again on the remote node or all cluster nodes, respectively. Removing the resource constraint does not necessarily mean that a resource that was previously running on a node before it was put in **standby** mode migrates back. To remove the **standby** resource constraint from all nodes in the current cluster, run:

```
[root@nodeY ~]# pcs cluster unstandby --all
```

# Reviewing cluster status

For a system administrator, it is important to be able to retrieve the current status of the cluster, the cluster nodes, and the cluster resources.

A detailed overview of the cluster status, **corosync** status, configured resource groups, resources, and status of the cluster nodes is provided by **pcs status**.

The **pcs status** output can be limited with one of the following parameters:

**Cluster Status Switches**

| Switch | Purpose |
|---|---|
| **pcs status cluster** | Show only information related to cluster status. |
| **pcs status groups** | Show only the configured resource groups and their resources. |
| **pcs status resources** | Show only the status of the resource groups and their individual resources in the cluster. |
| **pcs status nodes** | Show only the status of the configured cluster nodes. |
| **pcs status corosync** | Show only the status of **corosync**. |
| **pcs status pcsd** | Show only the status of **pcsd** on all configured cluster nodes. |

The **pcs status** command is a powerful utility that enables a system administrator to determine the status of cluster node membership and display all information related to the cluster and cluster nodes:

```
[root@nodea ~]# pcs status
Cluster name: cluster
Last updated: Fri Sep 26 05:47:40 2014
Last change: Wed Sep 24 06:19:49 2014 via cibadmin on nodea.private.example.com
Stack: corosync
Current DC: nodeb.private.example.com (2) - partition with quorum
Version: 1.1.10-29.el7-368c726
4 Nodes configured
6 Resources configured


Node nodeb.private.example.com (2): standby
Online: [ nodea.private.example.com nodec.private.example.com ]
OFFLINE: [ noded.private.example.com ]

Full list of resources:

 fence_nodea (stonith:fence_rht): Started nodea.private.example.com
 fence_nodeb (stonith:fence_rht): Started nodeb.private.example.com
 fence_nodec (stonith:fence_rht): Started nodec.private.example.com
 fence_noded (stonith:fence_rht): Started noded.private.example.com
 Resource Group: web
     floatingip (ocf::heartbeat:IPaddr2):   Started nodea.private.example.com
     Website    (ocf::heartbeat:apache):    Started nodea.private.example.com


PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
  noded.private.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

In the previous example, the cluster consists of four cluster nodes with the following status:

- The cluster node **nodeb.private.example.com** is in **standby** mode.
- The cluster nodes **nodea.private.example.com** and **nodec.private.example.com** are fully operational and participating in the cluster, and therefore marked as **Online**.
- The cluster node **noded.private.example.com** is running because the status of **pcsd** is **Online**. The cluster node is marked as **OFFLINE** because the cluster services have either been stopped on this cluster node or failed to communicate with the quorate part of the cluster.

**References**

Red Hat High Availability Add-On Reference

- Section 3.2: Managing Cluster Nodes

**pcs**(8), **corosync**(8), and **pacemaker**(8) man pages

# Guided Exercise

# Manage Cluster Membership

In this lab, you will extend a three-node cluster with a fourth node.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **noded**. |

## Outcome(s)

You should be able to extend an existing cluster.

- You should have a working three-node cluster called **clusterX**. If you do *not* have a working three-node cluster, reset your **nodea**, **nodeb**, and **nodec** machines, then run the command **lab basiccluster setup** on your **workstation** machine.

```
[student@workstation ~]$ lab basiccluster setup
```

- Reset your **noded** machine.

1. Prepare the virtual machine **noded** to become a member of the **clusterX** cluster.

   1.1. Allow cluster communications to pass through the firewall on the new node.

```
[root@noded ~]# firewall-cmd --permanent --add-service=high-availability
[root@noded ~]# firewall-cmd --reload
```

   1.2. Install the *pcs* and *fence-agents-rht* packages on **noded.private.example.com**.

```
[root@noded ~]# yum install pcs fence-agents-rht
```

   1.3. Enable and start the **pcsd** service on the new node.

```
[root@noded ~]# systemctl enable pcsd
[root@noded ~]# systemctl start pcsd
```

   1.4. Change the password of the **hacluster** system user to **redhat** on **noded**.

```
[root@noded ~]# echo redhat | passwd --stdin hacluster
```

   1.5. Authenticate the cluster node **noded.private.example.com** on the existing cluster.

```
[root@nodea ~]# pcs cluster auth -u hacluster -p redhat \
> noded.private.example.com
```

2. Extend the existing cluster by adding **noded.private.example.com** to the cluster.

   2.1. Add **noded.private.example.com** to the existing cluster on **nodea**.

```
[root@nodea ~]# pcs cluster node add noded.private.example.com
nodea.private.example.com: Corosync updated
nodeb.private.example.com: Corosync updated
nodec.private.example.com: Corosync updated
noded.private.example.com: Succeeded
```

   2.2. Authenticate the cluster node **noded** with the **nodea**, **nodeb**, and **nodec** virtual machines.

```
[root@noded ~]# pcs cluster auth -u hacluster -p redhat
nodea.private.example.com: Authorized
nodeb.private.example.com: Authorized
nodec.private.example.com: Authorized
noded.private.example.com: Already authorized
```

3. Enable automatic startup of the cluster services on the **noded** system. Verify the **noded** machine automatically joins the cluster **clusterX** after system boot.

   3.1. Enable automatic startup of the cluster services on **noded** when the machine is rebooted.

```
[root@noded ~]# pcs cluster enable
```

   3.2. Start the cluster services on **noded**.

```
[root@noded ~]# pcs cluster start
Starting Cluster...
```

   3.3. Verify the cluster is running and all configured cluster nodes have joined the cluster. Verify on all cluster nodes that all nodes are showing up with the status **Online** to ensure the nodes are properly authorized.

```
[root@nodeY ~]# pcs status
Cluster name: clusterX
Last updated: Tue Sep 23 05:41:18 2014
Last change: Tue Sep 23 05:41:03 2014 via crmd on nodea.private.example.com
Stack: corosync
Current DC: nodea.private.example.com (1) - partition with quorum
Version: 1.1.10-29.el7-368c726
4 Nodes configured
3 Resources configured


Online: [ nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com noded.private.example.com ]

Full list of resources:
```

```
fence_nodea (stonith:fence_rht): Started nodea.private.example.com
fence_nodeb (stonith:fence_rht): Started nodeb.private.example.com
fence_nodec (stonith:fence_rht): Started nodec.private.example.com

PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
  noded.private.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

4. Configure fencing for **noded**. You will need to use the **fence_rht** fencing agent, with a plug name of **noded.private.example.com** and the **classroom.example.com** fencing device. This new fencing resource should be called **fence_noded**.

   4.1. Add the fence devices for **noded** to the cluster on **nodea.private.example.com**.

```
[root@nodea ~]# pcs stonith create fence_noded fence_rht \
> port="noded.private.example.com" \
> pcmk_host_list="noded.private.example.com" \
> ipaddr="classroom.example.com"
```

   4.2. Verify the fence devices have been added correctly to the **clusterX** cluster.

```
[root@noded ~]# pcs stonith show
 fence_nodea (stonith:fence_rht): Started
 fence_nodeb (stonith:fence_rht): Started
 fence_nodec (stonith:fence_rht): Started
 fence_noded (stonith:fence_rht): Started
```

5. Reboot **noded** and verify it automatically joins the cluster.

   5.1. Restart **noded**.

```
[root@noded ~]# reboot
```

   5.2. Verify that **noded** has automatically joined the cluster after it rebooted.

```
[root@nodea ~]# pcs status
Cluster name: clusterX
Last updated: Tue Sep 24 05:41:18 2014
Last change: Tue Sep 24 05:41:03 2014 via crmd on nodea.private.example.com
Stack: corosync
Current DC: nodea.private.example.com (1) - partition with quorum
Version: 1.1.10-29.el7-368c726
4 Nodes configured
4 Resources configured
```

```
Online: [ nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com noded.private.example.com ]

Full list of resources:

fence_nodea (stonith:fence_rht): Started nodea.private.example.com
fence_nodeb (stonith:fence_rht): Started nodeb.private.example.com
fence_nodec (stonith:fence_rht): Started nodec.private.example.com
fence_noded (stonith:fence_rht): Started nodeb.private.example.com

PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
  noded.private.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

# Quorum Operations

## Objectives

After completing this section, students should be able to explain how quorum operates and protects a cluster from errors.

## What is quorum?

In order for a cluster to work as expected, the nodes must be in agreement on certain facts, such as which machines are currently cluster members, where services are running, and which machines are using which resources.

The method in which this is implemented in the Red Hat High Availability Add-on is by the use of a majority voting scheme. Every cluster node casts one vote if it successfully joins the **`corosync`** network communication and is able to communicate with the other nodes that are already participating in the cluster.

The cluster is operational if more than half of all possible votes are successfully cast. The minimum number of votes needed to achieve more than half of the votes is called the *quorum*. If *quorum* is achieved, the cluster is considered *quorate*. A cluster loses quorum if half of the nodes or more cannot communicate with each other.

When a cluster gets started, all cluster nodes try to communicate with each other and aim to achieve quorum. As soon as a majority is formed, there is a quorate cluster. All other nodes that have not successfully joined the quorate cluster get fenced by a node that has quorum. If a node that is part of the quorate cluster is not able to communicate with the cluster anymore, the node gets fenced.

## Why is quorum calculation required?

Quorum is required in situations where some nodes in a cluster cannot communicate with certain other nodes. The following illustration shows a five-node cluster consisting of nodes **A**, **B**, **C**, **D**, and **E** with a service that uses an **`ext4`** file system on shared storage. This service is currently running on node **A**.

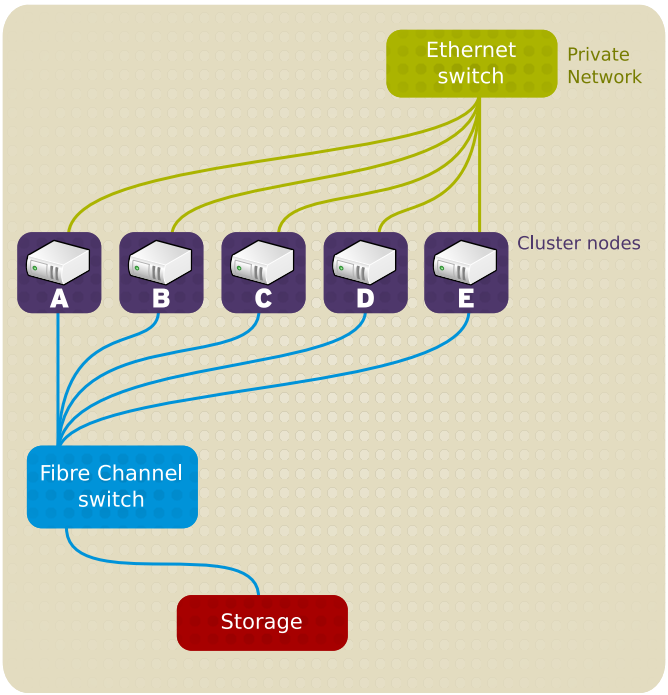



**Figure 2.1: A five-node cluster**

The nodes **D** and **E** get split off from the main private network and cannot communicate with the nodes **A**, **B**, and **C**. Without quorum, those two nodes will decide that **A**, **B**, and **C** have all failed and need to be fenced (remotely powered off) so that their resources can be recovered. In this way, quorum acts as an important gate prior to fencing.

Were the cluster to be without a fencing device, nodes **D** and **E** would immediately begin recovering resources, resulting in the same **`ext4`** file system being mounted in two places at once, which is not a healthy situation. This situation, where two halves of a cluster operate independently from each other, is referred to as a split-brain.

Split-brain is a particular concern in two-node clusters, since if either node fails, the other node does not consist of a majority of the nodes in the cluster. Special steps need to be taken to allow two-node cluster operation and still avoid split-brain."

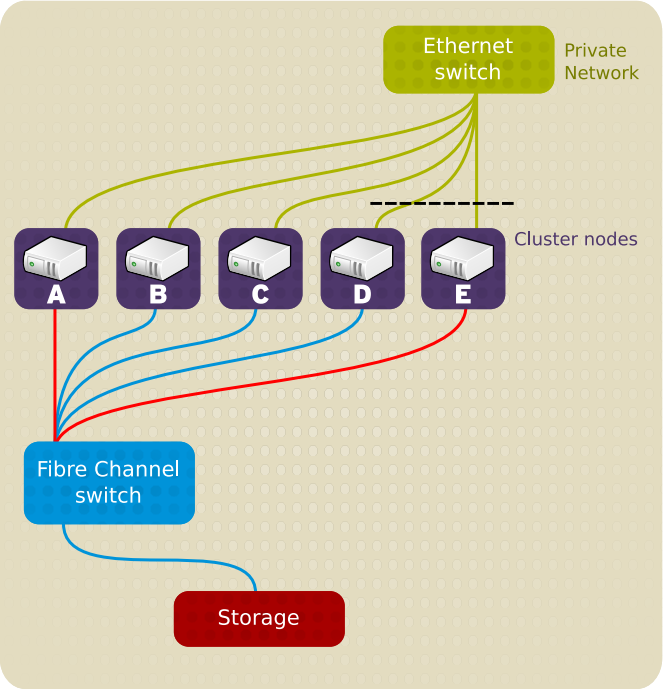



Figure 2.2: A split-brain five-node cluster

Assuming that all nodes in this example have one vote, this situation cannot arise, since node **D** and node **E** only have two votes together, which is not more than half of the total votes (five). This would result in node **D** and node **E** ceasing to function until at least one other vote was added. Nodes **A**, **B**, and **C**, on the other hand, remain active providing service, since they have three votes combined, which is more than half of the total votes.

# Calculating quorum

Quorum is calculated and managed in the Red Hat High Availability Add-on by the **`corosync`** component **`votequorum`**. The **`votequorum`** component uses two values to calculate if the cluster is quorate:

1. **Expected votes**: The number of votes expected if all cluster nodes are fully operational and communicating with each other.

2. **Total votes**: The number of votes currently present. This can be lower than the number of expected votes if some nodes are not up or not communicating with the cluster.

The number of votes required to achieve quorum is based on **`Expected Votes`**. The following formula shows how many votes are required for quorum. In this calculation, **`floor()`** means to always round down.

```
Quorum = floor(expected votes ÷ 2 + 1)
```

## Quorum calculation example

In the following example, a cluster with three nodes is assumed. A three-node cluster has an **`Expected Votes`** count of three.

```
Quorum = floor(expected votes ÷ 2 + 1)
Quorum = floor(3 ÷ 2 + 1)
Quorum = floor(1.5 + 1)
Quorum = floor(2.5)
Quorum = 2
```

In the three-node cluster, there needs to be a minimum of two nodes running to achieve quorum.

In the following example, a cluster with four nodes is assumed. A four-node cluster has an **`Expected Votes`** count of four.

```
Quorum = floor(expected votes ÷ 2 + 1)
Quorum = floor(4 ÷ 2 + 1)
Quorum = floor(2 + 1)
Quorum = floor(3)
Quorum = 3
```

In the four-node cluster, there needs to be a minimum of three nodes running to achieve quorum.

## Displaying quorum status

The Red Hat High Availability Add-on provides a comprehensive utility to display the current state of quorum in a cluster: **`corosync-quorumtool`**. The **`corosync-quorumtool`** provides an overview of quorum-related information, such as **`Total Votes`** and **`Expected Votes`**, and shows if the cluster is quorate at a glance.

```
[root@nodea ~]# corosync-quorumtool
Quorum information
------------------
Date:             Thu Sep 25 05:02:00 2014
Quorum provider:  corosync_votequorum
Nodes:            4          (1)
Node ID:          1          (2)
Ring ID:          4648
Quorate:          Yes        (3)

Votequorum information
----------------------
Expected votes:   4          (4)
Highest expected: 4          (5)
Total votes:      4          (6)
Quorum:           3          (7)
Flags:            Quorate    (8)

Membership information
----------------------
  Nodeid      Votes Name
       1          1 nodea.private.example.com (local)
       2          1 nodeb.private.example.com
       3          1 nodec.private.example.com
       4          1 noded.private.example.com
```

(1) The **`Nodes`** entry provides the node count of the nodes that belong to the cluster.

2. The **`Node ID`** entry shows the ID of the node on which **`corosync-quorumtool`** was executed.
3. The **`Quorate`** entry shows if the cluster is quorate.
4. The **`Expected votes`** entry shows the number of votes that are present if all configured cluster members are active in the cluster.
5. The **`Highest expected`** entry shows the largest count of expected votes seen in the cluster.
6. The **`Total votes`** entry shows the count of votes currently present in the cluster.
7. The **`Quorum`** entry shows how many votes need to be present at minimum so the cluster stays quorate.
8. The **`Flags`** entry shows any quorum-related properties that are currently set on the cluster. If the cluster is quorate, then the **`Quorate`** property will show. Additional special features will also display in this field when set, such as **`LastManStanding`** or **`WaitForAll`**.

**References**

Red Hat High Availability Add-On Overview

- Section 2.1: Quorum Overview

**`corosync-quorumtool`**(8) man page

## Guided Exercise

# Quorum Operations

In this lab, you will observe cluster quorum as nodes fail.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **noded** |

## Outcome(s)

You should be able to observe and describe cluster behavior as nodes get turned off.

- You should have a working three- or four-node cluster. If you do *not* currently have a operational cluster with fencing, reset all four of your **node*Y*** machines and run the command **`lab basiccluster setup`** on **`workstation`**.

```
[student@workstation ~]$ lab basiccluster setup
```

In this exercise, you will observe cluster behavior as nodes get turned off or cut off from networking.

1. On your **nodea**, open an instance of **`corosync-quorumtool -m`**, and leave it running for the remainder of this exercise. The **`-m`** flag tells **`corosync-quorumtool`** to listen for changes in cluster status and output them as they happen.

```
[root@nodea ~]# corosync-quorumtool -m
```

   Depending on if you have a three- or a four-node cluster, the output from **`corosync-quorumtool`** should show three or four active nodes, with a corresponding number of expected and total votes.

2. On your **nodec** system, disable cluster communications by temporarily closing the ports needed for **`corosync`**, then observe the results.

   2.1. On your **nodec** machine, remove the **`high-availability`** **`firewalld`** service from the running configuration (*not* from the permanent configuration).

```
[root@nodec ~]# firewall-cmd --remove-service=high-availability
```

   2.2. View the output from your running **`corosync-quorumtool -m`** command. The expected votes should have remained the same, but the number of total votes should have dropped by one.

   Additionally, your **`nodec.private.example.com`** machine should have dropped from the **`Membership information`** table.

   2.3. Wait for your **nodec** machine to become available again. The other cluster nodes should have rebooted it automatically.

2.4. The cluster should now show all nodes active again.

3. Stress-test your cluster by powering down two nodes, **nodeb** and **nodec**, then observe the results.

3.1. Power down both your **nodeb** and **nodec** systems.

```
[root@nodeb ~]# poweroff
```

```
[root@nodec ~]# poweroff
```

3.2. Return to the terminal where you are running **corosync-quorumtool -m**; the **Quorum** line should now read **Activity blocked**. Activity is now blocked because both three- and four-node cluster can only survive one node being down. To prevent resource corruption, the cluster has quit starting, stopping, or otherwise touching cluster resources.

**Note**

The nodes did not get fenced this time. A node that is powered down will not be automatically started by the other nodes. Only if a node is running, but not responding, will it be fenced.

4. Start your **nodeb** machine, then observe the results.

4.1. Start your **nodeb** machine.

4.2. **corosync-quorumtool** should now return to a quorate cluster, but with one missing node.

5. Start your **nodec** machine.

# Managing Quorum Calculations

## Objectives

After completing this section, students should be able to adjust the way quorum is determined to allow special configurations.

## Set quorum calculation options

The **votequorum** component allows a cluster administrator to build clusters with switches that alter the way quorum gets calculated. When building a new cluster with the **pcs cluster setup** command, the following switches may be added to change the behavior of quorum handling in the cluster:

**Cluster Setup Switches**

| Switch | Purpose |
|---|---|
| **--wait_for_all** | Wait for all cluster members to be online before starting to calculate quorum. This setting effectively prevents a fence race when the cluster is starting up. Without this setting enabled, all nodes that have not joined the cluster, or shut down cleanly, are automatically fenced as soon as quorum is achieved. |
| **--auto_tie_breaker** | Quorum can be achieved with only 50% of the cluster nodes up, instead of the default 50% + 1 votes. In case of a 50% vs. 50% split-brain situation, the side where the lowest **Node ID** takes part wins quorum. |

| Switch | Purpose |
|---|---|
| **--last_man_standing** | With this option enabled, the expected votes get recalculated every 10 seconds by default. This allows a system administrator to disconnect cluster nodes from the cluster one by one until there are only two nodes actively participating in the cluster. Be aware that disconnecting cluster nodes too fast can lead to loss of quorum even though there are two or more nodes operational.<br><br>The tunable **--last_man_standing_window** allows a system administrator to change the number of milliseconds until **expected votes** are recalculated when a node stops participating in the cluster. The default setting is 10000 milliseconds (10 seconds).<br><br>In conjunction with the **--auto_tie_breaker** option, **--last_man_standing** allows for downgrading the cluster to one node.<br><br>The use of **last_man_standing** also requires the use of **wait_for_all** because it allows multiple partitions (subsets of the cluster) to claim quorum at the same time.<br><br>The command **corosync-quorumtool -m** does *not* automatically update when the number of expected votes has changed. In these cases, it can be more useful to run **watch -n1 corosync-quorumtool** instead. |
| **--two_node** | This setting is for clusters that consist of only two nodes. It sets the expected votes to **1**, so the cluster is still operational if one of the two nodes fails. This is because a single node normally can not have quorum since we would otherwise need **floor(2 ÷ 2 + 1) = 2** votes. This option automatically enables **--wait_for_all**, which is automatically disabled if more than two nodes join the cluster. |

The switches listed previously can be combined when creating a new cluster as far as combining them is useful. To create a cluster with **--last_man_standing** and **--wait_for_all** enabled, execute:

```
[root@archer ~]# pcs cluster setup --start \
> --name testcluster --last_man_standing --wait_for_all \
> archer.example.com \
> lana.example.com \
> cyril.example.com
```

## Changing quorum options in an existing cluster

The cluster quorum calculation can be influenced on an existing cluster as well. For that, a maintenance window is recommended on a cluster in production so the cluster can be restarted for **corosync** to pick up the new configuration.

It is also possible to have **corosync** reload changes on the fly, using the **corosync-cfgtool -R** command, but quorum options are not reloaded. Another possibility is to manually restart

the **corosync.service** service on all nodes, one at a time. While the latter option does not require any downtime, the risks associated with having two halves of a cluster running on different configurations are greater than zero.

The configuration file that holds the quorum-related options of the cluster is **/etc/corosync/corosync.conf**. All quorum-related options are set in the **quorum** directive. The default quorum directive in **/etc/corosync/corosync.conf** without any special options set is as follows:

```
quorum {
provider: corosync_votequorum
}
```

The configuration options **wait_for_all**, **auto_tie_breaker**, **last_man_standing**, and **two_node** can be turned on by adding ***option*: 1** inside the quorum directive. The **last_man_standing_window: *milliseconds*** option may be added to tune the recalculation time of expected votes if **last_man_standing** is enabled.

Before making manual adjustments to the **/etc/corosync/corosync.conf** configuration file, it is advisable to schedule a maintenance window and shut down the cluster with **pcs cluster stop --all**. This avoids unplanned cluster downtime, for example, due to configuration files that cannot be parsed.

To enable the **last_man_standing**, **wait_for_all**, and **auto_tie_breaker** options, change the quorum directive in the **/etc/corosync/corosync.conf** file to include the options as follows:

```
quorum {
provider: corosync_votequorum
last_man_standing: 1
wait_for_all: 1
auto_tie_breaker: 1
auto_tie_breaker_node: lowest
}
```

**Note**

Even though **auto_tie_breaker_node: lowest** is the default, the current version of **corosync_votequorum** does not correctly configure **auto_tie_breaker** unless this option is present.

The content of the **corosync.conf** file must be the same on every node in the cluster. The command **pcs cluster sync** provides a mechanism to sync the **corosync.conf** file from the current node to all other cluster nodes in the same cluster. After synchronizing the **corosync** configuration file, the cluster can be started again with **pcs cluster start --all**.

Once the cluster is started with the new options enabled, the **corosync-quorumtool** command shows the new options listed in the **Flags** output:

```
[root@node ~]# corosync-quorumtool
...
Flags:            Quorate WaitForAll LastManStanding AutoTieBreaker
...
```

**References**

Red Hat High Availability Add-On Reference

- Section 3.1.5.: Configuring Quorum Options

**pcs**(1), **votequorum**(5), and **corosync-quorumtool**(8) man pages

## Guided Exercise

# Manage Quorum Calculations

In this lab, you will experiment with various quorum options on a four-node cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **noded** |

### Outcome(s)

You should be able to modify a cluster to use both the **last_man_standing** and **wait_for_all** quorum options.

- Reset all four of your **node*Y*** systems.
- From your **workstation** system, run the command **lab fournodecluster setup**.

```
[student@workstation ~]$ lab fournodecluster setup
```

- *Optional:* Run the command **lab setupsshkeys** on **workstation** to sync SSH keys from your **workstation** to all nodes.

```
[student@workstation ~]$ lab setupsshkeys setup
```

1. Reconfigure the cluster **cluster*X*** with **last_man_standing** and **wait_for_all** options enabled. Verify the new configuration is active. You are allowed to stop the running cluster to apply these changes.

   1.1. Stop the cluster services on all nodes.

```
[root@nodea ~]# pcs cluster stop --all
nodea.private.example.com: Stopping Cluster...
nodeb.private.example.com: Stopping Cluster...
nodec.private.example.com: Stopping Cluster...
noded.private.example.com: Stopping Cluster...
```

   1.2. Add the **last_man_standing** and **wait_for_all** options in the quorum section of the **/etc/corosync/corosync.conf** configuration file on **nodea**.

```
quorum {
provider: corosync_votequorum
last_man_standing: 1
wait_for_all: 1
}
```

   1.3. Synchronize the **corosync.conf** configuration file from the current node to all other nodes in the cluster.

```
[root@nodea ~]# pcs cluster sync
```

1.4. Start the cluster services on all cluster nodes.

```
[root@nodea ~]# pcs cluster start --all
nodea.private.example.com: Starting Cluster...
nodeb.private.example.com: Starting Cluster...
nodec.private.example.com: Starting Cluster...
noded.private.example.com: Starting Cluster...
```

**2.** Monitor quorum and degrade the cluster **clusterX** by powering down the two nodes **nodec** and **noded** while keeping quorum on the remaining cluster nodes.

2.1. Monitor quorum on **nodea** with **watch -n1 corosync-quorumtool** in a separate terminal window.

**Note**

**corosync-quorumtool -m** does not automatically update when the expected votes change.

```
[root@nodea ~]# watch -n1 corosync-quorumtool
```

2.2. Power down **nodec**.

```
[root@nodec ~]# poweroff
```

2.3. Verify that **Expected votes** has been reduced to **3** votes, 10 seconds after the machine **nodec** has been stopped.

```
...
Votequorum information
----------------------
Expected votes:   3
Highest expected: 3
Total votes:      3
Quorum:           2
Flags:            Quorate WaitForAll LastManStanding
...
```

2.4. Power down **noded**.

```
[root@noded ~]# poweroff
```

2.5. Verify that **Expected votes** has been reduced to **2** votes 10 seconds after the machine **noded** has been stopped because of the **last_man_standing** option. The cluster is still quorate even with only two active nodes.

```
...
Votequorum information
----------------------
Expected votes:   2
Highest expected: 2
Total votes:      2
Quorum:           2
Flags:            Quorate WaitForAll LastManStanding
...
```

3. Compare your observations with the **last_man_standing** option enabled with the observation of what happens if a cluster loses nodes without **last_man_standing** enabled.

   Without the **last_man_standing** option enabled, the cluster would have become inquorate when the second node went down.

4. Restart your **nodec** and **noded** machines.

## Lab

# Managing Cluster Nodes and Quorum

### Performance Checklist

In this lab, you will extend a three-node cluster to four nodes, then change the quorum options for the cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **noded** |

### Outcome(s)

You should be able to create a four-node cluster with the **auto_tie_breaker** option set.

- Reset all four of your **node*Y*** systems.
- From **workstation**, run the command **lab basiccluster setup**.

```
[student@workstation ~]$ lab basiccluster setup
```

- *Optional*: Run the command **lab setupsshkeys setup** on **workstation** to create and distribute a SSH key pair.

```
[student@workstation ~]$ lab setupsshkeys setup
```

You have been asked to take your existing three-node cluster, **cluster*X***, and to add the new node **noded.private.example.com** to it.

Fencing for this node should be handled by a fence device named **fence_noded**, and handled by the fencing server on **classroom.example.com**. The plug name for this node is **noded.private.example.com**.

After **noded** has joined the cluster, the cluster should be configured to use the **auto_tie_breaker** quorum option.

1. Prepare **noded** for cluster membership.
2. Add **noded** to the cluster, and create the fencing device. Make sure the new node will automatically start all cluster services at boot time.
3. Reconfigure cluster quorum options to include **auto_tie_breaker**.

## Solution

# Managing Cluster Nodes and Quorum

### Performance Checklist

In this lab, you will extend a three-node cluster to four nodes, then change the quorum options for the cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **noded** |

### Outcome(s)

You should be able to create a four-node cluster with the **auto_tie_breaker** option set.

- Reset all four of your **node*Y*** systems.
- From **workstation**, run the command **lab basiccluster setup**.

```
[student@workstation ~]$ lab basiccluster setup
```

- *Optional*: Run the command **lab setupsshkeys setup** on **workstation** to create and distribute a SSH key pair.

```
[student@workstation ~]$ lab setupsshkeys setup
```

You have been asked to take your existing three-node cluster, **cluster*X***, and to add the new node **noded.private.example.com** to it.

Fencing for this node should be handled by a fence device named **fence_noded**, and handled by the fencing server on **classroom.example.com**. The plug name for this node is **noded.private.example.com**.

After **noded** has joined the cluster, the cluster should be configured to use the **auto_tie_breaker** quorum option.

1. Prepare **noded** for cluster membership.

   1.1. Allow cluster communications to pass through the firewall on the new node.

```
[root@noded ~]# firewall-cmd --permanent --add-service=high-availability
[root@noded ~]# firewall-cmd --reload
```

   1.2. Install the *pcs* and *fence-agents-rht* packages on **noded.private.example.com**.

```
[root@noded ~]# yum install -y pcs fence-agents-rht
```

   1.3. Enable and start the **pcsd** service on the new node.

```
[root@noded ~]# systemctl enable pcsd
[root@noded ~]# systemctl start pcsd
```

1.4. Change the password of the **hacluster** system user to **redhat** on **noded**.

```
[root@noded ~]# echo redhat | passwd --stdin hacluster
```

1.5. Authenticate the cluster node **noded.private.example.com** on the existing cluster.

```
[root@nodea ~]# pcs cluster auth -u hacluster -p redhat noded.private.example.com
```

**2.** Add **noded** to the cluster, and create the fencing device. Make sure the new node will automatically start all cluster services at boot time.

2.1. Add **noded.private.example.com** to the existing cluster on **nodea**.

```
[root@nodea ~]# pcs cluster node add noded.private.example.com
nodea.private.example.com: Corosync updated
nodeb.private.example.com: Corosync updated
nodec.private.example.com: Corosync updated
noded.private.example.com: Succeeded
```

2.2. Authenticate the cluster node **noded** with the **nodea**, **nodeb**, and **nodec** virtual machines.

```
[root@noded ~]# pcs cluster auth -u hacluster -p redhat
nodea.private.example.com: Authorized
nodeb.private.example.com: Authorized
nodec.private.example.com: Authorized
noded.private.example.com: Already authorized
```

2.3. Add the fence devices for **noded** to the cluster on **nodea.private.example.com**.

```
[root@nodea ~]# pcs stonith create fence_noded fence_rht \
> port="noded.private.example.com" \
> pcmk_host_list="noded.private.example.com" \
> ipaddr="classroom.example.com"
```

2.4. Verify the fence devices have been added correctly to the **clusterX** cluster.

```
[root@nodea ~]# pcs stonith show
 fence_nodea (stonith:fence_rht): Started
 fence_nodeb (stonith:fence_rht): Started
 fence_nodec (stonith:fence_rht): Started
 fence_noded (stonith:fence_rht): Started
```

2.5. Enable automatic startup of the cluster services on **noded** when the machine is rebooted.

```
[root@noded ~]# pcs cluster enable
```

2.6. Start the cluster services on **noded**.

```
[root@noded ~]# pcs cluster start
Starting Cluster...
```

2.7. Verify the cluster is running and all configured cluster nodes have joined the cluster. Verify on all cluster nodes that all nodes are showing up with the status **Online** to ensure the nodes are properly authorized.

```
[root@nodeY ~]# pcs status
Cluster name: clusterX
Last updated: Tue Sep 23 05:41:18 2014
Last change: Tue Sep 23 05:41:03 2014 via crmd on nodea.private.example.com
Stack: corosync
Current DC: nodea.private.example.com (1) - partition with quorum
Version: 1.1.10-29.el7-368c726
4 Nodes configured
3 Resources configured


Online: [ nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com noded.private.example.com ]

Full list of resources:

fence_nodea (stonith:fence_rht): Started nodea.private.example.com
fence_nodeb (stonith:fence_rht): Started nodeb.private.example.com
fence_nodec (stonith:fence_rht): Started nodec.private.example.com
fence_noded (stonith:fence_rht): Started noded.private.example.com

PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
  noded.private.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

**3.** Reconfigure cluster quorum options to include **auto_tie_breaker**.

3.1. Stop the cluster services on all nodes.

```
[root@nodea ~]# pcs cluster stop --all
nodea.private.example.com: Stopping Cluster...
nodeb.private.example.com: Stopping Cluster...
nodec.private.example.com: Stopping Cluster...
noded.private.example.com: Stopping Cluster...
```

3.2. Add the **auto_tie_breaker** option in the quorum section of the **/etc/corosync/corosync.conf** configuration file on **nodea**.

```
quorum {
provider: corosync_votequorum
auto_tie_breaker: 1
auto_tie_breaker_node: lowest
}
```

3.3. Synchronize the **corosync.conf** configuration file from the current node to all other nodes in the cluster.

```
[root@nodea ~]# pcs cluster sync
```

3.4. Start the cluster services on all cluster nodes.

```
[root@nodea ~]# pcs cluster start --all
nodea.private.example.com: Starting Cluster...
nodeb.private.example.com: Starting Cluster...
nodec.private.example.com: Starting Cluster...
noded.private.example.com: Starting Cluster...
```

# Summary

In this chapter, you learned:

- **`pcs cluster node add`**
- **`pcs cluster node remove`**
- **`corosync-quorumtool`**
- Quorum options defined in **`/etc/corosync/corosync.conf`**.
- Modifying quorum options requires a sync of **`corosync.conf`**, and a restart of the **`corosync.service`** service.
- If downtime is allowed, stop all cluster nodes, update, then start all cluster nodes again to update quorum options.

Chapter 3

# Managing Fencing

**Goal**

Select and configure fencing devices to separate nodes from storage after a failure.

**Objectives**

- Explain how fencing is used to protect data during node failures.
- Prepare the hardware or software fencing device.
- Configure the cluster to use the correct fence devices when fencing a node.

**Sections**

- Protecting Data with Fencing (and Quiz)
- Setting Up Fencing Devices (and Practice)
- Configuring Cluster Fencing Agents (and Practice)

**Lab**

- Managing Fencing

# Protecting Data with Fencing

## Objectives

After completing this section, students should be able to explain how fencing is used to protect data during node failures.

## What is fencing?

The Red Hat High Availability Add-on uses fencing to ensure data integrity in the cluster. Fencing is often accomplished by powering the node off, since a dead node is clearly not able to do anything. In other cases, a combination of operations will be used to cut the node off from the network (to stop new work from arriving) or from storage (to stop the node from writing to shared storage). Fencing is a necessary step in service and resource recovery in a cluster. The Red Hat High Availability Add-on will not start resource and service recovery for a nonresponsive node until that node has been fenced.

### Cluster operation without fencing

Without fencing, data integrity on shared storage resources cannot be guaranteed. In a three-node cluster consisting of nodes **A**, **B**, and **C**, there are no fencing devices configured. Node **A** has an **`ext4`** file system mounted from shared storage, and is running a web server serving pages from that file system. If node **A** stops responding on the network, the following chain of events is triggered:

1. Node **B** mounts the file system from shared storage after performing a quick file system check.
2. Node **B** starts the web service.
3. Node **A** wakes up again and continues writing to the same **`ext4`** file system that is mounted on node **B** as well.
4. File system corruption ensues.

### Cluster operation with fencing

In order to stop node **A** from accessing the file system, and thus causing file system corruption, after node **B** has taken over the resource, it must be ensured that node **A** will no longer access this file system *before* another node attempts to mount the file system. This procedure is called fencing.

With fencing configured, the chain of events would be slightly different:

1. Node **B** and node **C** cut off node **A** from storage.
2. Node **B** mounts the file system from shared storage after performing a quick file system check.
3. Node **B** starts the web service.
4. Node **A** wakes up again and attempts to write to the mounted file system. This fails since node **A** can no longer access the shared storage resource.

*-or-*

Node **A** is rebooted and comes up cleanly, joining the cluster.

**Important**
In order for your cluster to be deemed supportable by Red Hat, fencing must be configured for all nodes.

# Fencing mechanism overview

There are two main methods of fencing: power fencing, also known as *Shoot The Other Node In The Head* (STONITH), and fabric fencing. Both fencing methods require a fence device, such as a power switch or the virtual fencing daemon and fencing agent software to enable communication between the cluster and the fencing device. The fencing agent communicates when a particular node should be fenced.

## Power fencing

Power fencing entails cutting off power to a server. This fencing method is called *STONITH*, short for *Shoot The Other Node In The Head*.

Two different kinds of power fencing devices exist:

- External fencing hardware that cuts off the power, such as a network-controlled power strip.
- Internal fencing hardware, such as ILO, DRAC, IPMI, or virtual machine fencing, that powers off the hardware of the node.

Power fencing can be configured to turn the target machine off and keep it off, or to turn it off and then on again. Turning a machine back on has the added benefit that the machine should come back up cleanly and rejoin the cluster if the cluster services have been enabled.

The following graphic shows an example of power fencing using a network-controlled power controller and two power supplies in a server.

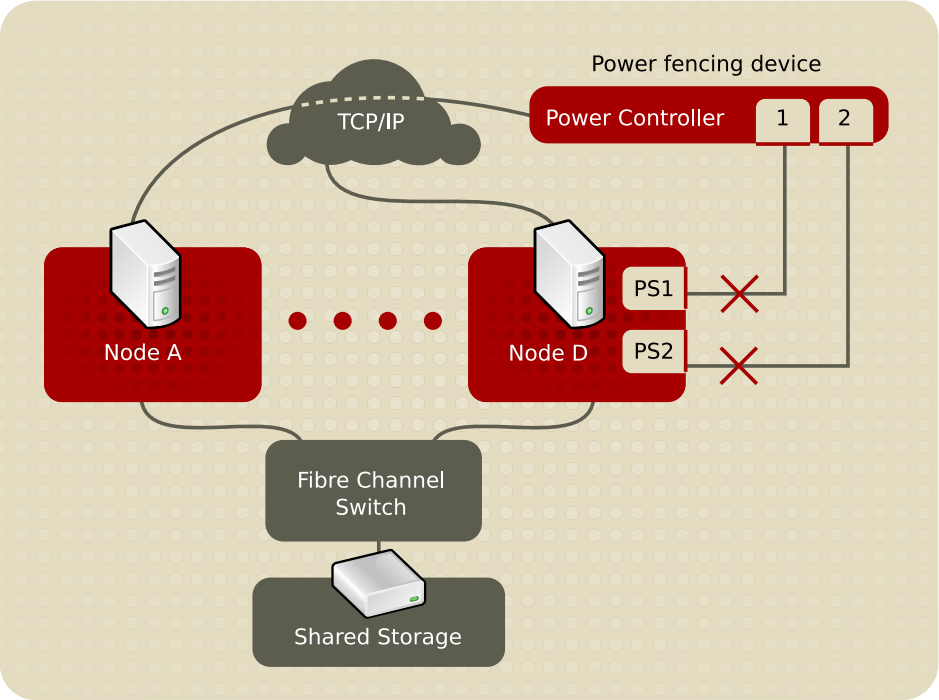



**Figure 3.1: Power fencing using two power supplies**

**Warning**

Power fencing with a machine that has more than one power supply needs to be configured in a way that all power supplies are turned off before power supplies are turned on again. Otherwise, the fenced machine is never turned off because it does not lose power.

This could cause the machine to be apparently fenced without *actually* being fenced, risking data loss.

## Fabric fencing

Fabric fencing (SCSI fencing) entails disconnecting a machine from storage at the storage level. This can be done by closing ports on a Fibre Channel switch, or by using SCSI reservations. If a machine is fenced only using fabric fencing and not in combination with power fencing, it is the administrator's responsibility to make sure that the machine will join the cluster again. This is usually done by rebooting or power-cycling the failed node.

The following graphic shows an example of fabric fencing using multipathed Fibre Channel storage.

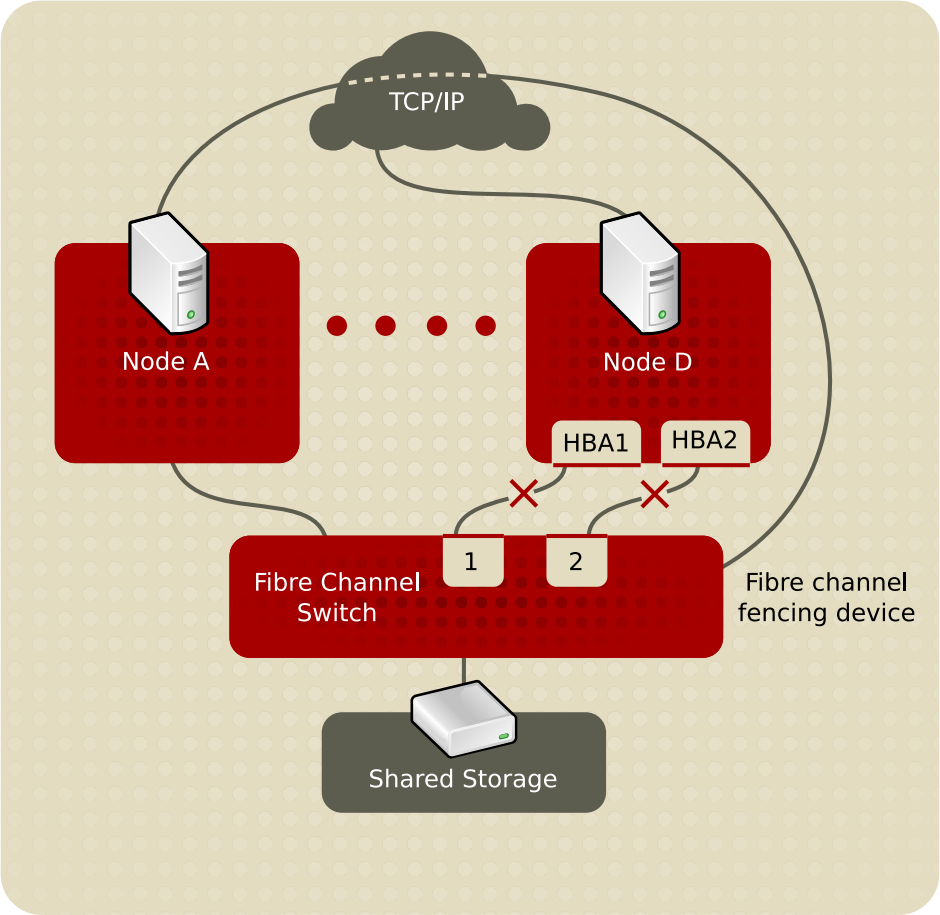



Figure 3.2: Fabric fencing using multipathed Fibre Channel storage and a fiber switch

## Combining fencing methods

Fencing methods can be combined. When a node needs to be fenced, one fence device can cut off Fibre Channel by blocking ports on a Fibre Channel switch, and an ILO card can then power cycle the offending machine. Multiple fencing methods can act as a backup for each other. For example, the cluster nodes are first fenced by power fencing, but if that fails, with fabric fencing.

**References**

Red Hat High Availability Add-On Overview

- Section 2.2: Fencing Overview

## Quiz

# Protecting Data with Fencing

Choose the correct answer to the following questions:

**1. The main goal of fencing is to...**

a. Power cycle a failed node.
b. Ensure a failed node can not cause corruption before recovering its resources.
c. Inform an administrator that a node has failed.
d. Migrate services to a different cluster node.

**2. Fencing is necessary to...**

a. Maintain data integrity.
b. Detect failed nodes.
c. Run a web server.

**3. Only one fencing method can be configured per node.**

a. True
b. False
c. It depends on the fencing agent.

**4. Power fencing will always attempt to reboot a failed node.**

a. True; this is the only way that power fencing can be configured.
b. False; power fencing can also be configured to only switch a machine off, not on again.

**5. Fencing methods can be combined to... (Choose two.)**

a. Run as one unit.
b. Run differently depending on the day of the week.
c. Run as primary/backup.
d. Run differently depending on the time of day.

## Solution

# Protecting Data with Fencing

Choose the correct answer to the following questions:

**1. The main goal of fencing is to...**

a. Power cycle a failed node.
b. Ensure a failed node can not cause corruption before recovering its resources.
c. Inform an administrator that a node has failed.
d. Migrate services to a different cluster node.

**2. Fencing is necessary to...**

a. Maintain data integrity.
b. Detect failed nodes.
c. Run a web server.

**3. Only one fencing method can be configured per node.**

a. True
b. False
c. It depends on the fencing agent.

**4. Power fencing will always attempt to reboot a failed node.**

a. True; this is the only way that power fencing can be configured.
b. False; power fencing can also be configured to only switch a machine off, not on again.

**5. Fencing methods can be combined to... (Choose two.)**

a. Run as one unit.
b. Run differently depending on the day of the week.
c. Run as primary/backup.
d. Run differently depending on the time of day.

# Setting Up Fencing Devices

## Objectives

After completing this section, students should be able to prepare hardware or software fencing devices.

## Overview of fence device configuration

Fencing is a requirement for every operational cluster. The first step when setting up fencing for the cluster is the setup of the hardware or software device that does the actual fencing.

The Red Hat Enterprise Linux High Availability Add-on provides a variety of fencing agents for use with different fence devices. The **`pcs stonith list`** provides a list of all installed fencing agents:

```
[root@nodeY ~]# pcs stonith list
fence_apc - Fence agent for APC over telnet/ssh
fence_apc_snmp - Fence agent for APC over SNMP
fence_bladecenter - Fence agent for IBM BladeCenter
...
```

Depending on the fence device and fencing agent in use, different parameters are required. The parameters are passed by the fencing agent to the fencing device. Communication between the fencing agent and the fencing device is established and fencing of a cluster node is successful only if the required set of parameters is passed. A man page is available in the system for every shipped fencing agent; the man page describes the parameters that can be passed to the fencing device. A list of possible and required parameters for a specific fencing agent can also be found by executing the command **`pcs stonith describe <FENCINGAGENT>`**.

```
[root@nodeY ~]# pcs stonith describe fence_rhevm
```

## Examples of fence device configuration

Different fencing devices require different hardware and software configurations. The hardware needs to be set up and configured. The software required to be configured and various configuration parameters need to be documented for later use with the fencing agent.

### APC network power switch fencing

One way to configure power fencing is to use an APC network power switch. The hardware setup includes power cabling of the cluster nodes with the APC network power switch. Fencing with an APC network power switch requires the fencing agent to log into the power switch to control the power outlet of a specific node. For setting up fencing with an APC fence device, it is important to document at least the following switch settings for later use with the fencing agent:

- IP address of the APC fence device.
- Username and password to access the APC fence device.

- If the device can be accessed by SSH or telnet.
- The plug ID(s) for each cluster node must be known.

### Management hardware fencing

Management hardware, such as ILO, DRAC, or IPMI hardware, can power down, power up, and power cycle systems. At a minimum, the following parameters need to be configured and known to the cluster administrator to use management cards as fencing devices:

- IP address of the management device.
- Username and password to access the management fence device.
- Which machines are handled by the management fence device.

### SCSI fencing

SCSI fencing does not require any physical hardware dedicated for fencing. The cluster administrator needs to know which device has to be blocked from cluster node access with SCSI reservation.

## Virtual machine fencing

The Red Hat High Availability Add-On ships with a number of different fencing agents for different hypervisors. With the exception of **`fence_virt`**, the fencing agent for libvirt, they require the same kind of parameters:

- IP or host name of the hypervisor.
- Username and password to access the hypervisor.
- The virtual machine name for each node.

## Libvirt fencing

Cluster nodes that are virtual machines running on a Red Hat Enterprise Linux host with KVM/libvirt require the software fencing device **`fence-virtd`** configured and running on the hypervisor. Virtual machine fencing in multicast mode works by sending a fencing request signed with a shared secret key to the libvirt fencing multicast group. This means that the actual node virtual machines can be run on different hypervisor machines, as long as all hypervisors have **`fence-virtd`** configured for the same multicast group, and using the same shared secret. To set up the **`fence-virtd`** software fence device on the hypervisor running the virtual machines, the following steps are required:

1. On the hypervisor, install the *fence-virtd*, *fence-virtd-libvirt*, and *fence-virtd-multicast* packages. These packages provide the virtual machine fencing daemon, libvirt integration, and multicast listener, respectively.

```
[root@hypervisor ~]# yum -y install fence-virtd fence-virtd-libvirt fence-virtd-
multicast
```

2. On the hypervisor, create a shared secret key called **`/etc/cluster/fence_xvm.key`**. The target directory **`/etc/cluster`** needs to be created manually.

```
[root@hypervisor ~]# mkdir -p /etc/cluster
[root@hypervisor ~]# dd if=/dev/urandom of=/etc/cluster/fence_xvm.key bs=1k
 count=4
```

3. On the hypervisor, configure the **fence_virtd** daemon. Defaults can be used for most options, but make sure to select the **libvirt** back end and the **multicast** listener.

```
[root@hypervisor ~]# fence_virtd -c
```

4. Enable and start the **fence_virtd** daemon on the hypervisor.

```
[root@hypervisor ~]# systemctl enable fence_virtd
[root@hypervisor ~]# systemctl start fence_virtd
```

5. Copy the shared secret key **/etc/cluster/fence_xvm.key** to all cluster nodes, keeping the name and the path the same as on the hypervisor.

## The classroom fencing device

Red Hat Training uses a custom fencing agent in the classroom, **fence_rht**. This fencing agent is designed to work in all Red Hat classroom types, using the same configuration on the fencing agent side. This fencing agent needs two parameters:

- The host name of the server running the fencing daemon (**classroom.example.com**).
- The host name of the node to be fenced as known by the cluster, for example, **noded.private.example.com**.

## Testing fence devices

Fencing is crucial for an operational cluster. It is mandatory for a cluster administrator to test the fencing setup thoroughly. The fencing device setup can be tested by calling fencing agents from the command line. All fencing agents reside in **/usr/sbin/fence_***. The fencing agents typically take a **-h** option to show all available options, or **pcs stonith describe *fence_agent*** can be used to investigate possible options. The options required to test fencing differ from agent to agent.

**Note**

When testing fencing with an APC power switch, it can be helpful to plug in a lamp for testing instead of the actual cluster node. This makes it visible if controlling the power works as expected.

**Important**

Calling a fencing agent directly for testing fencing will verify if the actual fencing device itself is working properly, but not if fencing is correctly configured in the cluster.

**References**

**fence_*** (8) man pages

## Guided Exercise

# Setting Up Fencing Devices

In this lab, you will experiment with manually executing a fencing agent.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb** |

## Outcome(s)

You should be able to manually execute a fencing agent and observe the results.

- Reset all of your **node*Y*** machines.
- On your **workstation** system, execute the command **lab fencing setup**.

```
[student@workstation ~]$ lab fencing setup
```

**1.** On your **nodea** machine, install both the *fence-agents-all* and *fence-agents-rht* packages, then update the man page database to reflect the newly installed man pages.

1.1.

```
[root@nodea ~]# yum -y install fence-agents-all fence-agents-rht
```

1.2.

```
[root@nodea ~]# mandb
```

**2.** Use the **pcs** command to list all the fencing agents available to the cluster, then use the **pcs** command to view the possible options for the **fence_rhevm** fencing agent.

2.1.

```
[root@nodea ~]# pcs stonith list
```

2.2.

```
[root@nodea ~]# pcs stonith describe fence_rhevm
```

**3.** View a list of all **fence_*** man pages, then view the man page for the **ipmilan** fencing agent.

3.1.

```
[root@nodea ~]# man -k fence_
```

3.2.

```
[root@nodea ~]# man fence_ipmilan
```

4. Use the **-h** command-line option to view the parameters available with the **fence_rht** fencing agent, then fence your **nodeb.private.example.com** machine and observe the results. The fencing daemon is available on **classroom.example.com**, and the plug name you will need is the fully qualified domain name for the node to be fenced on the **private.example.com** domain.

4.1.

```
[root@nodea ~]# fence_rht -h
```

4.2. In a separate terminal, start a **ping** to **nodeb**, or open a console window for your **nodeb** machine.

```
[student@workstation ~]$ ping nodeb
```

4.3. From your **nodea** machine, fence your **nodeb** machine.

```
[root@nodea ~]# fence_rht -a classroom.example.com -n nodeb.private.example.com
```

# Configuring Cluster Fencing Agents

## Objectives

After completing this section, students should be able to configure the cluster to use the correct fence devices when fencing a node.

## Creating a fence device

A system administrator can create a fence device in the cluster with the **pcs stonith create** command:

```
[root@nodeY ~]# pcs stonith create name fencing_agent parameters
```

The command requires additional arguments, such as the fencing agent and the fencing parameters required for the fencing agent. There are generic properties for all the fencing agents shipped with Red Hat Enterprise Linux High Availability Add-on:

**Generic Fence Agent Properties**

| Parameter | Purpose |
|---|---|
| **stonith-timeout** | This setting defines the time to wait for fencing to complete in seconds. The default value is 60 seconds and is defined in the **stonith-timeout** cluster property. Setting **stonith-timeout** for a fencing device overrides the cluster default setting. If a fencing action takes longer than this timeout, the cluster will consider the fence operation to have failed. |
| **priority** | When defining multiple fence devices for a single node, they need to be prioritized. The cluster tries the fence device first with the highest priority. If this setting is omitted, a priority of **0** is assumed. |
| **pcmk_host_map** | This parameter maps host names to fence device ports for fencing devices that require the mapping, such as devices using the **fence_apc** fencing agents. The list is a semicolon-separated list of ***hostname:port*** mappings, such as **"nodea.example.com:1;nodeb.example.com:2"**. |

| Parameter | Purpose |
|---|---|
| **pcmk_host_check** | This parameter defines how the cluster determines the machines that may be controlled from the fencing device. Possible values are:<br>• **dynamic-list**: The cluster queries the fencing device. This only works if the fencing device can return a list of ports, and the port names match the host names of the cluster nodes.<br>• **static-list**: The cluster uses a list of hosts provided by **pcmk_host_list**. Unless **pcmk_host_map** is also used, port names must match the host names of cluster nodes.<br>• **none**: The cluster assumes that every fencing device can fence every node in the cluster. Unless **pcmk_host_map** is also used, port names must match the host names of cluster nodes.<br>The default setting for **pcmk_host_check** is **dynamic-list**, but switches to **static-list** whenever a **pcmk_host_map** or **pcmk_host_list** option is used. |
| **pcmk_host_list** | This parameter provides a space-separated list of machines that may be controlled by the fencing device. It is required if **pcmk_host_check** is set to **static-list**. |

If none of the **pcmk_host_*** options are set, the cluster will default to querying the fence device for a list of ports. If a port name matches a host name of a cluster node, that port will be used to fence that device.

In addition, a fencing device can be created for a single machine by specifying explicit **port="*portname*"** and **pcmk_host_list="*hostname*"** options.

**Note**

Not all fencing devices support listing ports for use with **pcmk_host_check="dynamic-list"**. In those cases, the use of a **pcmk_host_map** or **port** option is always needed.

In addition to the generic fencing properties listed previously, there are fencing agent-specific properties. The command **pcs stonith describe *fence_agent*** shows all required and optional parameters that may be set for a particular fence device:

```
[root@nodeY ~]# pcs stonith describe fence_apc
Stonith options for: fence_apc
  ipaddr (required): IP Address or Host-name
  login (required): Login Name
  passwd: Login password or pass-phrase
...
```

In a cluster that uses virtual machine fencing with **fence-virtd**, the **fence_xvm** fencing agent can be configured as a fence device for a cluster node. To create the fence device **myfence**, using the **fence_xvm** fencing agent to fence the virtual machine **myvm** known in the cluster with the host name **myvm.example.com**, execute:

```
[root@nodeY ~]# pcs stonith create myfence fence_xvm port="myvm"
 pcmk_host_list="myvm.example.com"
```

## Storage-based fence devices

Storage-based fence devices cut off a fenced node from storage access. A storage-based fence device does not power cycle a fenced node. When configuring a storage-based fence agent, such as **fence_scsi**, as a cluster fence device, it is important to add the meta parameter **meta provides=unfencing** for the node to automatically get unfenced when the node is rebooted and the cluster services are starting to allow the node to rejoin the cluster.

```
[root@nodeY ~]# pcs stonith create myscsifence fence_scsi devices=/dev/sda meta
 provides=unfencing
```

## Display fencing devices

The **pcs stonith show** command allows a system administrator to view the list of configured fence devices in the cluster, the fencing agent that is used, and the current status of the fence device. Fence device status can be **Started** or **Stopped**. If the status of a fence device is **Started**, the device is operational; if it is **Stopped**, the fence device is not operational.

```
[root@nodeY ~]# pcs stonith show
 fence_nodea  (stonith:fence_rht): Started
 fence_nodeb  (stonith:fence_rht): Started
 fence_nodec  (stonith:fence_rht): Started
 fence_noded  (stonith:fence_rht): Started
```

If a cluster node is specified as a parameter or the **--full** option is added, the **pcs stonith show** command shows the configuration options of the specified cluster node or all cluster nodes, respectively.

```
[root@nodeY ~]# pcs stonith show fence_nodea
 Resource: fence_nodea (class=stonith type=fence_rht)
  Attributes: port=nodea pcmk_host_list=nodea.private.example.com
  Operations: monitor interval=60 (fence_nodea-monitor-interval-60s)
```

## Changing fencing devices

Fencing device options may be changed with the **pcs stonith update *fence_device_name*** command. This allows a system administrator to add a new fence device option or change an existing one.

For example, the fencing device **fence_nodeb** currently fences the virtual machine **nodea** instead of **nodeb**. This can be corrected by executing:

```
[root@nodeY ~]# pcs stonith update fence_nodeb port=nodeb
```

## Removing fencing devices

At some point, it might be necessary to remove a fencing device from the cluster. This might happen because the corresponding cluster node was removed from the cluster permanently, or a

different fencing mechanism is used to fence the node. The command **pcs stonith delete *fence_device_name*** allows a system administrator to remove a fencing device from the cluster. To remove the fencing device **fence_noded** from the cluster, execute:

```
[root@nodeY ~]# pcs stonith delete fence_noded
Attempting to stop: fence_noded...Stopped
Deleting Resource - fence_noded
```

## Testing fence configuration

There are two ways to check if a cluster fencing configuration is fully operational:

- By using the command **pcs stonith fence *hostname***. This will attempt to fence the requested node. If successful, the cluster can fence this node.
- By disabling the network on a node, either by unplugging the network cable(s), closing the cluster ports on the firewall, or disabling the entire network stack. The other nodes in the cluster should detect that the machine has failed, and fence it. This will test the cluster's ability to detect a failed node as well.

**References**

Red Hat High Availability Add-On Reference

- Section 4: Fencing: Configuring Stonith

**pcs**(8) and **fence_***(8) man pages

## Guided Exercise

# Configuring Cluster Fencing Agents

In this lab, you will update and modify cluster fencing agents.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **noded** |

## Outcome(s)

You should be able to create, remove, and modify cluster fencing resources.

- Reset all four of your **node*Y*** machines.
- From your **workstation** system, run the command **lab fournodecluster setup**.

```
[student@workstation ~]$ lab fournodecluster setup
```

**1.** Inspect the full fencing configuration on your cluster. This should show four separate fencing resources, one for each node. Test if this configuration is functional by fencing your **noded** machine using the **pcs** command.

1.1. Inspect the full fencing configuration on your cluster.

```
[root@nodea ~]# pcs stonith show --full
   Resource: fence_nodea (class=stonith type=fence_rht)
    Attributes: port=nodea.private.example.com
 pcmk_host_list=nodea.private.example.com ipaddr=classroom.example.com
    Operations: monitor_interval=60s (fence_nodea-monitor-interval-60s)
  ...
```

1.2. Verify operation by fencing **noded**.

```
[root@nodea ~]# pcs stonith fence noded.private.example.com
```

Wait for **noded** to rejoin the cluster before continuing.

**2.** While the four separate fencing resources are functional, they can be replaced with a single fencing resource, responsible for fencing all four nodes when needed.

Remove all four current fencing resources, then create a single new fencing resource called **fence_classroom**, using the **fence_rht** fencing agent with **pcmk_host_check** set to **static-list**. The address for the fencing daemon is **classroom.example.com**.

Test your configuration.

2.1. Remove all four existing fencing resources.

```
[root@nodea ~]# for I in fence_node{a..d}; do
> pcs stonith delete ${I}
> done
```

2.2. Create a new **fence_rht** fencing device called **fence_classroom** using **pcmk_host_check="static-list"**.

```
[root@nodea ~]# pcs stonith create fence_classroom fence_rht \
> ipaddr="classroom.example.com" \
> pcmk_host_check="static-list" \
> pcmk_host_list="nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com noded.private.example.com"
```

2.3. Verify operation by fencing **noded**.

```
[root@nodea ~]# pcs stonith fence noded.private.example.com
```

Wait for **noded** to rejoin the cluster before continuing.

**3.** While the configuration in the previous step was functional, it does require a lot of typing, with the danger of typing errors. Since all of the nodes in this cluster can be fenced by the same fencing device, the **pcmk_host_list** option can be removed if the **pcmk_host_check** option is set to none.

Modify your **fence_classroom** resource to use **pcmk_host_check="none"** and empty **pcmk_host_list**, then test.

3.1. Modify your **fence_classroom** resource.

```
[root@nodea ~]# pcs stonith update fence_classroom \
> pcmk_host_check="none" pcmk_host_list=""
```

3.2. Verify operation by fencing **noded**.

```
[root@nodea ~]# pcs stonith fence noded.private.example.com
```

## Lab

# Managing Fencing

### Performance Checklist

In this lab, you will configure fencing for a three-node cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to configure fencing.

- Reset all four of your **node*Y*** machines.
- From your **workstation** machine, run the command **lab fencing setup**.

```
[student@workstation ~]$ lab fencing setup
```

Before going on extended parenting leave, one of your co-workers was tasked with creating a three-node cluster named **cluster*X*** on the machines **nodea**, **nodeb**, and **nodec**.

While your colleague did create the cluster, fencing was left unconfigured. The following information was left for you:

- Fencing type: **fence_rht**
- Required fencing resource name: **fence_all**
- Fencing host: **classroom.example.com**
- All port/plug names on the fencing host correspond to host names on the **private.example.com** network.
- Only a single fencing resource should be created.

Implement fencing using these requirements, then test your configuration by interrupting cluster communications on your **nodeb** machine.

1. Create a fencing resource called **fence_all**, using the **fence_rht** fencing agent and the fencing daemon on **classroom.example.com**.
2. Test your configuration by interrupting cluster communications on **nodeb**.

## Solution

# Managing Fencing

### Performance Checklist

In this lab, you will configure fencing for a three-node cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to configure fencing.

- Reset all four of your **node*Y*** machines.
- From your **workstation** machine, run the command **lab fencing setup**.

```
[student@workstation ~]$ lab fencing setup
```

Before going on extended parenting leave, one of your co-workers was tasked with creating a three-node cluster named **cluster*X*** on the machines **nodea**, **nodeb**, and **nodec**.

While your colleague did create the cluster, fencing was left unconfigured. The following information was left for you:

- Fencing type: **fence_rht**
- Required fencing resource name: **fence_all**
- Fencing host: **classroom.example.com**
- All port/plug names on the fencing host correspond to host names on the **private.example.com** network.
- Only a single fencing resource should be created.

Implement fencing using these requirements, then test your configuration by interrupting cluster communications on your **nodeb** machine.

1. Create a fencing resource called **fence_all**, using the **fence_rht** fencing agent and the fencing daemon on **classroom.example.com**.

   1.1. From **nodea**, create the new fencing device.

```
[root@nodea ~]# pcs stonith create fence_all fence_rht \
> ipaddr="classroom.example.com" pcmk_host_check="none" pcmk_host_list=""
```

2. Test your configuration by interrupting cluster communications on **nodeb**.

2.1.

```
[root@nodeb ~]# firewall-cmd --remove-service=high-availability
```

2.2. Observe **nodeb** being fenced, using **corosync-quorumtool -m** on **nodea**.

```
[root@nodea ~]# corosync-quorumtool -m
```

# Summary

In this chapter, you learned:

- How fencing is necessary to prevent data and resource corruption.
- How to prepare hardware and software fencing devices.
- How to create fencing resources using **`pcs stonith create`**.
- How to remove fencing resources using **`pcs stonith delete`**.
- How to update fencing resources using **`pcs stonith update`**.
- How to use the various **`pcmk_host_*`** options when creating fencing devices.

Chapter 4

# Creating and Configuring Resources

**Goal**

Create basic resources and resource groups to provide highly available services.

**Objectives**

- Create and configure high-availability resources.
- Assemble resource groups and adjust the properties of their resources.
- Manually control and relocate resource groups in a running cluster.

**Sections**

- Creating and Configuring Resources (and Practice)
- Creating and Configuring Resource Groups (and Practice)
- Managing Resource Groups (and Practice)

**Lab**

- Creating and Configuring Resources

# Creating and Configuring Resources

## Objectives

After completing this section, students should be able to create and configure high-availability resources.

## Resources

Clustered services consist of one or more *resources*. A resource can be an IP address, a file system, or a service like **`httpd`**, among others. All the bits required to provide a service to consumers are a resource.

With the Red Hat High Availability Add-on, all resources are monitored independently from each other. In order to accomplish this, resources are controlled by resource agents. Pacemaker can manage different kinds of resource agents:

**Resource Agents**

| Agent | Purpose |
|---|---|
| **`LSB`** | Linux Standard Base-compatible init scripts residing in **`/etc/init.d/`**. |
| **`OCF`** | Open Cluster Framework-compatible scripts that are extended LSB init scripts that can process additional input parameters to provide additional control over the cluster resource. |
| **`Systemd`** | Systemd unit files, which are the standard for defining and managing services on Red Hat Enterprise Linux 7. |

For a cluster administrator, it is important to understand which system services are supported by the cluster software. Ideally, a service is directly supported by the cluster with an **`OCF`** script that provides a maximum amount of configurability when setting up the service in the cluster. For services that are not directly supported with an **`OCF`** script, there is still the possibility to use either a **`systemd`** unit file or a **`LSB`**-compliant init script.

## Commonly used resources

The following table has a list of some commonly used resources:

**Common Resources**

| Agent | Purpose |
|---|---|
| **`Filesystem`** | Used for mounting a file system. This can be a local file system, a file system on a iSCSI or Fibre Channel device, or a remote file system like a NFS export or an SMB share. |

| Agent | Purpose |
|---|---|
| **IPaddr2** | This resource is used to assign a floating IP address to a resource group. There is also a separate **IPaddr** resource that uses an older method of assigning a IP address. On Red Hat Enterprise Linux 7 **IPaddr** is a symbolic link to **IPaddr2**. |
| **apache** | This resource starts an apache **httpd** service. Unless otherwise configured this will use the configuration from **/etc/httpd**. |
| **mysql** | This resource controls a **mysql** database. Databases can be configured for standalone operation, a clone set with external replication, or as a full master/slave setup. |

# Resource groups

A service usually consists of more than one resource. As an example, a typical web service consists of an IP address resource, an Apache resource, and a shared file system resource for the **DocumentRoot** of the web server. All web service resources need to run on the same cluster node to provide a working service. A convenient way to tie those resources together is to add them to the same resource group. All services in the same resource group get started in the order in which they have been added to the resource group and stopped in the reverse order. When a cluster node fails, the cluster migrates the whole resource group to a different node and starts the resources on the new node.

# Configuring cluster resources

The Red Hat High Availability Add-on offers a wide range of support for cluster resources. Cluster resources are managed by scripts called **agents**. The agents are required to start, stop, and monitor the configured cluster resources.

## Display resource agents

It is important for a cluster administrator to get an overview of all available resource agents provided by the Red Hat High Availability Add-on. The command **pcs resource list** lists all resource agents known to **Pacemaker**.

```
[root@nodeY ~]# pcs resource list
ocf:heartbeat:CTDB - CTDB Resource Agent
ocf:heartbeat:Delay - Waits for a defined timespan
ocf:heartbeat:Dummy - Example stateless resource agent
ocf:heartbeat:Filesystem - Manages filesystem mounts
...
```

## Display resource agent parameters

Every resource agent offers a list of tunable parameters. The command **pcs resource describe *resource_name*** displays a description and a list of tunable parameters for a given cluster resource. To show information and parameters of the **Filesystem** resource, execute:

```
[root@nodeY ~]# pcs resource describe Filesystem
ocf:heartbeat:Filesystem - Manages filesystem mounts

...
```

```
Resource options:
  device (required): The name of block device for the filesystem, or -U, -L
 options for mount, or NFS mount specification.
  directory (required): The mount point for the filesystem.
  fstype (required): The type of filesystem to be mounted.
  options: Any extra options to be given as -o options to mount. For bind mounts,
 add "bind" here and set fstype to "none". We will do the right thing for options
 such as "bind,ro".
...
```

## Create cluster resources

When creating a cluster resource, it can be instantly added to a resource group with **pcs resource create *resource_name resource_provider resource parameters* --group *group_name***. If the group provided with the **--group** option does not exist, it is automatically created. If the resource group already exists, the resource is added as the last item to the existing group. The **--group** option may be omitted to create a standalone resource that is not part of any resource group. To create the **myfs** resource using the **Filesystem** resource agent that uses the **XFS**-formatted **/dev/sdb1** device mounted on **/var/www/html** as part of the **mygroup** resource group, execute:

```
[root@nodeY ~]# pcs resource create myfs Filesystem  device=/dev/sdb1 directory=/
var/www/html fstype=xfs --group mygroup
```

**Important**

When configuring a resource, it is important that all nodes that potentially could use the resource have access to that resource. For example, when configuring an **apache** resource, the *httpd* package must be installed on all cluster nodes.

## Display configured cluster resources

An overview of configured cluster resources and resource groups is viewable with the **pcs resource show** command, along with the resource status:

```
[root@nodeY ~]# pcs resource show
 Resource Group: mygroup
 myip    (ocf::heartbeat:IPaddr2):       Started
 myfs    (ocf::heartbeat:Filesystem):    Started
 myserv  (ocf::heartbeat:apache):        Started
```

The **pcs resource show** command, followed by a defined cluster resource as a parameter, shows details about the configured resource, such as the resource attributes and operation settings:

```
[root@nodeY ~]# pcs resource show myfs
 Resource: myfs (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/sdb1 directory=/var/www/html fstype=xfs
  Operations: start interval=0s timeout=60 (myfs-start-timeout-60)
              stop interval=0s timeout=60 (myfs-stop-timeout-60)
              monitor interval=20 timeout=40 (myfs-monitor-interval-20)
```

**Note**

The **pcs resource show** displays all resources and resource groups that have been configured for running on the cluster. The **pcs resource list** displays all available resource agents.

## Change resource parameter

At times, there might be a desire to fine-tune an already created cluster resource or to change options to refine the existing configuration. The Red Hat High Availability Add-on allows for changing cluster resource configurations with the **pcs resource update** command. To change the device of the cluster resource **myfs** to use **/dev/sda1** as the device, execute:

```
[root@nodeY ~]# pcs resource update myfs device=/dev/sda1
```

## Remove cluster resources

Cluster resources and resource group configurations may be removed from the cluster if they are not required any more. The **pcs resource delete *name*** allows for deleting single resources, as well as resource groups with all resources that are attached to it.

```
[root@nodeY ~]# pcs resource delete myfs
```

**References**

**Open Cluster Framework**
http://www.opencf.org

Red Hat High Availability Add-On Overview

- Section 3: Red Hat High Availability Add-On Resources

**pcs**(8) man page

## Guided Exercise

# Creating and Configuring Resources

In this lab, you will create a cluster resource.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to create a basic cluster resource.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, **nodec**, and **noded** systems.
- From your **workstation** system, run the command **lab resources setup**.

```
[student@workstation ~]$ lab resources setup
```

- *Optional*: To create and distribute SSH keys from your **workstation** system, you can run the command **lab setupsshkeys setup**.

```
[student@workstation ~]$ lab setupsshkeys setup
```

In this exercise, you will create a **Filesystem** resource, called **wwwfs**, for the NFS export **workstation.storage1.example.com:/exports/www** and configure it to be mounted *read-only* on **/mnt**.

After creating the resource, you will observe it in action, then see how the resource gets moved to a different node after fencing the node it is currently running on.

1. Create a new **Filesystem** resource to mount the NFS export **workstation.storage1.example.com:/exports/www** on **/mnt** read-only.

   The new resource should be called **wwwfs**.

   1.1. Inspect the possible options for the **Filesystem** resource.

```
[root@nodea ~]# pcs resource describe Filesystem
```

   1.2. Create the **wwwfs** resource using all relevant options.

```
[root@nodea ~]# pcs resource create wwwfs Filesystem \
> device=workstation.storage1.example.com:/exports/www \
> directory=/mnt \
> fstype=nfs \
> options=ro
```

1.3. Use **pcs status** to see the new resource being started.

```
[root@nodea ~]# pcs status
```

1.4. On the node that has started the **wwwfs** resource, inspect the contents of **/mnt**. You should see some files.

```
[root@nodeY ~]# ls -l /mnt
```

1.5. Disrupt cluster network communications on the node that is currently running the **wwwfs** resource.

```
[root@nodeY ~]# firewall-cmd --remove-service=high-availability
```

1.6. Use **watch -n1 pcs status** to observe the node previously running the **wwwfs** resource get fenced, and the resource being moved to another node.

# Creating and Configuring Resource Groups

## Objectives

After completing this section, students should be able to assemble resource groups and adjust the properties of their resources.

## Configuring resource groups

In most cases, having the cluster run individual resources is not an ideal situation. Mounting a file system with web content on one node, starting a **`httpd`** instance on a second, and assigning a floating IP address to a third node does not create a service that consumers can use.

Resources can be combined into *resource groups* that will force all resources in a group to start and stop at the same time, on the same node, creating clustered services for consumers.

### Creating a clustered Apache service

For a clustered Apache web server, a minimum of three different resources are required:

1. A (floating) IP address resource with an IP address from the public network to access the content served by the Apache web server.

2. A **`filesystem`** resource that provides the content to be served by the Apache web server. (In theory, it is also possible to have all web content on all nodes, but this requires a mechanism to sync the content.)

3. An **`apache`** resource controlling the **`httpd`** system service.

The first step toward a clustered web service is to define an IP address resource. In order to create the cluster resource called **`webip`**, using the resource agent **`IPaddr2`** with the floating IP address **`172.25.99.80/24`** as part of the **`myweb`** resource group, execute:

```
root@nodeY ~]# pcs resource create webip IPaddr2 ip=172.25.99.80 cidr_netmask=24
 --group myweb
```

**Note**

The Red Hat High Availability Add-on ships with two IP address resources: **`IPaddr`** and **`IPaddr2`**. In releases prior to Red Hat Enterprise Linux 7, **`IPaddr`** uses the *net-tools* infrastructure, such as the **`ifconfig`** command. In Red Hat Enterprise Linux 7, **`IPaddr`** is a symlink to **`IPaddr2`**, which uses the **`ip`** command to manage floating IP addresses.

The web server will also need access to shared storage hosting the content for the web server. In order to create a **`Filesystem`** resource for a NFS mount **`workstation.storage1.example.com:/exports/www`** on **`/var/www`**, execute:

```
[root@nodeY ~]# pcs resource create webfs Filesystem \
> device=workstation.storage1.example.com:/exports/www \
> directory=/var/www \
> fstype=nfs \
> options=ro \
> --group myweb
```

**Important**

With SELinux enabled on the cluster nodes, the cluster administrator must ensure that the content of any mounted file system can be used by the relevant daemons. For a block device, this can entail setting proper SELinux contexts; for NFS, this will mean setting the relevant SELinux Booleans, such as **`httpd_use_nfs=1`**.

The **`myweb`** resource group requires an Apache resource for sharing the document root content. For that, the *httpd* package needs to be installed and **`firewalld`** needs to allow access to the relevant ports on each cluster node the service can migrate to.

The cluster script can monitor the availability of the web server by periodically monitoring a status URL. For the monitoring to operate, the *wget* package must be installed on all cluster nodes that will serve the monitored Apache web server. The status URL must only be available from **`127.0.0.1`**. To create the required status URL, the following section must be present in the **`httpd`** configuration. Using a separate file in **`/etc/httpd/conf.d`** is recommended.

```
<Location /server-status>
SetHandler server-status
Require ip 127.0.0.1
Require all denied
</Location>
```

Other URLs can also be specified; for example, a main page or internal status page of the web application being hosted can be used to verify availability.

The resource called **`webserver`** using the **`apache`** resource agent can then be created and added to the **`myweb`** resource group with the **`httpd`** configuration file **`/etc/httpd/conf/httpd.conf`** and the status URL **`http://127.0.0.1/server-status`**, which adds monitoring to the resource.

```
[root@nodeY ~]# pcs resource create webserver apache \
> configfile="/etc/httpd/conf/httpd.conf" \
> statusurl="http://127.0.0.1/server-status" --group myweb
```

## Tune resource operations

Every cluster resource has three operations: **`start`**, **`stop`**, and **`monitor`**.

```
[root@nodeY ~]# pcs resource show webserver
Resource: webserver (class=ocf provider=heartbeat type=apache)
Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://127.0.0.1/
server-status
Operations: start interval=0s timeout=60 (apachefs-start-timeout-60)
            stop interval=0s timeout=60 (apachefs-stop-timeout-60)
            monitor interval=20 timeout=40 (apachefs-monitor-interval-20)
```

Operation parameters can be tuned by a system administrator when creating a resource by adding **op *operation***, followed by one or more operation options:

**Resource Operations**

| Parameter | Purpose |
|---|---|
| **interval=*value*** | Defines the time between the monitoring checks. The default value is taken from the resource agent. If the resource agent does not provide a default, the value is set to **60s**. |
| **timeout=*value*** | Defines the amount of time to wait before an operation is declared as failed if it did not complete until timeout is reached. |
| **on-fail=*action*** | The action to take if the operation failed:<br>• **ignore** ignores any failure of the operation.<br>• **block** stops performing any operations. This is the default action for a failed **stop** operation if the cluster does not have fencing configured.<br>• **stop** stops the resource from being active in the cluster. This is the default action for a failure of the **start** and **monitor** operations.<br>• **restart** stops and starts the resource.<br>• **fence** fences the node on which the resource failed. This is the default action for a failed **stop** operation for a cluster with fencing configured.<br>• **standby** moves all resources away from the cluster node the resource was running on. |

To create the **webserver** resource with a monitoring interval of **20** seconds with a timeout of **30** seconds, execute:

```
[root@nodeY ~]# pcs resource create webserver apache \
> configfile="/etc/httpd/conf/httpd.conf" \
> statusurl="http://127.0.0.1/server-status" --group myweb \
> op monitor interval=20s timeout=30s
```

Resource operations can be added and removed with the **pcs resource op add** and **pcs resource op remove** commands, respectively. To remove the existing monitoring operation from the **webserver** resource, execute:

```
[root@nodeY ~]# pcs resource op remove webserver monitor
```

For setting new parameters for the monitoring operation of the **webserver** resource with an monitoring interval of **10** seconds and a monitoring timeout of **15** seconds, resulting in a fenced node if the monitoring operation fails, execute:

```
[root@nodeY ~]# pcs resource op add webserver monitor interval=10s timeout=15s on-
fail=fence
```

**Important**

If a resource fails to start, the resource failcount will be set to **INFINITY**, which prevents the resource from starting on the node forever. Existing failcounts can be displayed with **pcs resource failcount show**. To troubleshoot the cause of failure, it is valuable to take a look at the output of **pcs resource debug-start *name***, which displays the error messages of the resource start attempt, and then repairs the issue. Once repaired, the failcount of the resource has to be reset for the resource to be allowed to start again on that node with: **pcs resource failcount reset *resourcename node***.

## Add and remove resources from a resource group

An existing resource can be added to a resource group. For that, the cluster administrator has to execute **pcs resource group add *groupname name***. If the resource is already a member of a group, it will be removed from the group and then added to the group specified on the command line. If the target resource group does not exist, it gets automatically created. To add the resource **myresource** to the **mygroup** resource group, execute:

```
[root@nodeY ~]# pcs resource group add mygroup myresource
```

A resource that is part of a resource group may be removed from the resource group with **pcs resource group remove *groupname name***. The resource then still exists in the cluster, but is not part of the resource group any more. If the last resource is removed from the resource group, the resource group is removed as well. To remove the **myresource** resource from the **mygroup** resource group, execute:

```
[root@nodeY ~]# pcs resource group remove mygroup myresource
```

# Resource ordering

In some cases there might be a dependency between resources added to a group. For example, a web server configured to listen on a specific IP address won't be able to start until that IP address is configured on a node. Normally resources will be started in the order that they are added to a group, and stopped in reverse order.

To influence the ordering, the options **--before *resourceid*** and **--after *resourceid*** can be added to **pcs resource create** and **pcs resource group add** to influence the ordering in which resources will be started.

**References**
**pcs**(8) man page

## Guided Exercise

# Creating and Configuring Resource Groups

In this lab, you will configure a resource group to serve web content.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, **nodec**, and **noded** |

## Outcome(s)

You should be able to create a basic resource group using multiple resources.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, **nodec**, and **noded** systems.
- From your **workstation** system, run the command **lab resources setup**.

```
[student@workstation ~]$ lab resources setup
```

- *Optional*: To create and distribute SSH keys from your **workstation** system, you can run the command **lab setupsshkeys setup**.

```
[student@workstation ~]$ lab setupsshkeys setup
```

In this exercise, you will create a resource group called **firstweb**, consisting of three resources:

- A **IPaddr2** resource called **firstwebip** for the IP address **172.25.*X*.80/24**.
- A **Filesystem** resource called **firstwebfs** that read-only mounts the NFS file system **workstation.storage1.example.com:/exports/www** on **/var/www**.
- An **apache** resource called **firstwebserver** that uses default settings for everything.

After creating your resource group, test it by accessing `http://172.25.X.80` from a web browser on **workstation**, then purposely fail the node the resource group is running on to test failover.

1. Prepare **nodea**, **nodeb**, and **nodec** to serve web content using **httpd** from NFS.

   1.1. On all three of your cluster nodes install the **httpd** package.

```
[root@nodeY ~]# yum -y install httpd
```

   1.2. On all three of your cluster nodes, set the required SELinux booleans to serve content from NFS.

```
[root@nodeY ~]# setsebool -P httpd_use_nfs 1
```

2. Create the three resources needed, making sure to put them in the **firstweb** resource group.

   2.1. Create the **IPaddr2** resource called **firstwebip** using the **172.25.*X*.80/24** IP address.

```
[root@nodea ~]# pcs resource create firstwebip IPaddr2 \
> ip=172.25.X.80 \
> cidr_netmask=24 \
> --group=firstweb
```

   2.2. Create the **firstwebfs Filesystem** resource to read-only mount **workstation.storage1.example.com:/exports/www** on **/var/www**.

```
[root@nodea ~]# pcs resource create firstwebfs Filesystem \
> device=workstation.storage1.example.com:/exports/www \
> directory=/var/www \
> fstype=nfs \
> options=ro \
> --group=firstweb
```

   2.3. Create the **firstwebserver apache** resource, using defaults for all settings.

```
[root@nodea ~]# pcs resource create firstwebserver apache --group=firstweb
```

3. On all three of your nodes allow **http** traffic through the firewall.

   3.1.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=http
[root@nodeY ~]# firewall-cmd --reload
```

4. Verify operation of your new resource group, including failover.

   4.1. Install *elinks* on **workstation**.

```
[student@workstation ~]$ sudo yum -y install elinks
```

   4.2. From your **workstation** system, use a web browser to access `http://172.25.X.80`.

```
[student@workstation ~]$ elinks -dump http://172.25.X.80
```

   4.3. Identify the node currently running the **firstweb** resource group.

```
[root@nodea ~]# pcs status
```

4.4. Impair cluster network communication on the node currently running the **firstweb** resource group, then use **pcs status** on another node to view the fencing and resource relocation process.

```
[root@nodeY ~]# firewall-cmd --remove-service=high-availability
```

4.5. Reload your web browsers to verify the **firstweb** group is still functional.

# Managing Resource Groups

## Objectives

After completing this section, students should be able to manually control and relocate resource groups in a running cluster.

## Managing services

The Red Hat High Availability Add-on allows a cluster administrator to control the resources and resource groups running in the cluster. Resources and resource groups may be started and stopped by the cluster administrator. Restrictions can be enforced to temporarily prohibit a resource or resource group from migrating to a specific cluster node, or to move a resource or resource group away from the node it is currently running on, and temporarily prohibit the resource from migrating back to the node it was moved from. The restrictions can be very helpful for starting maintenance on particular cluster nodes and minimizing service downtime by minimizing the amount of service migration by manual intervention with the cluster operation.

### Stop and start cluster resources

Cluster resources and resource groups can be stopped at any time to ensure they are not running on the cluster. This can be controlled by a cluster administrator with **`pcs resource disable`** ***`name`*** and **`pcs resource enable`** ***`name`*** for stopping and starting a resource, respectively. To stop all resources in the resource group **`someservice`**, execute:

```
[root@nodeY ~]# pcs resource disable someservice
```

Once stopped, a resource or resource group can be started again on the cluster. To start the resource group **`someservice`** on the cluster, execute:

```
[root@nodeY ~]# pcs resource enable someservice
```

### Move cluster resources

Resources and resource groups may be moved away from the cluster node where they are currently running with **`pcs resource move`** ***`name`***. Optionally, a target node can be added to the command to specify the node to which the resource or resource group should be moved. This feature is very helpful if a cluster node has a maintenance window, for example, to apply errata. To move the resource group **`someservice`** to **`nodec`**, execute:

```
[root@nodeY ~]# pcs resource move someservice nodec
```

### Prohibit a resource from migrating to a specific node

A cluster administrator can prevent a resource temporarily from migrating to a specific cluster node. The **`pcs resource ban`** ***`name`*** command prohibits the resource from running on the node where it currently runs. Optionally, a particular **`node`** can be added as a parameter on the

command line to restrict the resource from migrating to the specified node. To prevent the resource group **apacheweb** from running on **noded**, execute:

```
[root@nodeY ~]# pcs resource ban apacheweb noded.private.example.com
```

Both **pcs resource move** and **pcs resource ban** will create a temporary *constraint* rule on the cluster. Constraints are used, among other reasons, to influence which resources can run where. When a **move** command is used without a target, a **Disabled** rule with a score of **-INFINITY** will be created for the original node. When a **move** command is used with a target, an **Enabled** rule will be created (or moved) for the new target node, with a score of **INFINITY**.

## Display constraints

For a cluster administrator, it is important to understand the behavior of the configured services on the cluster. The command **pcs constraint list** allows an administrator to get an overview of the currently configured constraints in the cluster. In the following example, the resource group **myresourcegroup** has been banned from running on **nodec**.

```
[root@nodeY ~]# pcs constraint list
Location Constraints:
  Resource: myresourcegroup
    Disabled on: nodec.private.example.com (score:-INFINITY) (role: Started)
...
```

## Remove temporary resource restrictions

The temporary restrictions added by the **pcs resource ban** and **pcs resource move** commands can be removed for a particular resource with **pcs resource clear *name***. Optionally, a **node** can be added as a parameter on the command line to only remove restrictions for the given resource on a particular cluster node. To clear the **ban** restriction for the **ftpserv** resource group on **nodec**, execute:

```
[root@nodeY ~]# pcs resource clear ftpserv nodec.private.example.com
```

**References**

**pcs**(8) man page

## Guided Exercise

# Managing Resource Groups

In this lab, you will move a resource group between nodes in the cluster.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to move a running resource group.

- You should have a working three-node cluster with a resource group called **firstweb**. If *not*, reset your **workstation** and **node*Y*** machines, and run the command **lab resources solve** on your **workstation** machine.

```
[student@workstation ~]$ lab resources solve
```

1. Disable your **firstweb** resource group, then inspect the output of both **pcs status** and **pcs constraint list**.

   1.1. Disable your **firstweb** resource group.

```
[root@nodea ~]# pcs resource disable firstweb
```

   1.2. Inspect the output of **pcs status**.

```
[root@nodea ~]# pcs status
...
  Resource Group: firstweb
      firstwebip (ocf:heartbeat:IPaddr2):       Stopped
      firstwebfs (ocf:heartbeat:Filesystem):    Stopped
      firstwebserver     (ocf:heartbeat:apache):       Stopped
...
```

   1.3. Observe the output of **pcs constraint list**.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
Ordering Constraints:
Colocation Constraints:
```

2. From **nodea**, enable the **firstweb** resource group, then move it without a target. Inspect the resulting constraint rules.

   2.1. Enable the **firstweb** resource group from **nodea**.

```
[root@nodea ~]# pcs resource enable firstweb
```

2.2. Move the **firstweb** resource group to **nodeb**.

```
[root@nodea ~]# pcs resource move firstweb
```

2.3. Inspect the resulting constraint rules.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
  Resource: firstweb
    Disabled on: nodea.private.example.com (score:-INFINITY) (role: Started)
Ordering Constraints:
Colocation Constraints:
```

**3.** Ban the **firstweb** resource group from running on **nodeb** and **nodec**. What happens with the **firstweb** resource group?

3.1. Ban the **firstweb** resource group from running on **nodeb** and **nodec**.

```
[root@nodea ~]# pcs resource ban firstweb nodeb.private.example.com
[root@nodea ~]# pcs resource ban firstweb nodec.private.example.com
```

3.2. What happens to the **firstweb** resource group?

```
[root@nodea ~]# pcs status resources
  Resource Group: firstweb
      firstwebip (ocf:heartbeat:IPaddr2):       Stopped
      firstwebfs (ocf:heartbeat:Filesystem):    Stopped
      firstwebserver     (ocf:heartbeat:apache):       Stopped
```

```
[root@nodea ~]# pcs constraint list
Location Constraints:
  Resource: firstweb
    Disabled on: nodea.private.example.com (score:-INFINITY) (role: Started)
    Disabled on: nodeb.private.example.com (score:-INFINITY) (role: Started)
    Disabled on: nodec.private.example.com (score:-INFINITY) (role: Started)
Ordering Constraints:
Colocation Constraints:
```

**4.** Clear all constraints for the **firstweb** resource group.

```
[root@nodea ~]# pcs resource clear firstweb
```

## Lab

# Creating and Configuring Resources

### Performance Checklist

In this lab, you will create a three-node cluster with two resource groups.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to create multiple resource groups on a cluster.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, and **nodec** systems.
- From your **workstation** system, run the command **lab resources setup**.

```
[student@workstation ~]$ lab resources setup
```

For an upcoming project you have been requested to create two new resource groups on your **clusterX** cluster.

One resource group should be called **surprise**, and it should provide a web server that distributes the content found on the NFS share **workstation.storage1.example.com:/exports/www** on the IP address **172.25.X.80**. The correct mount point for the NFS share is **/var/www**.

The second resource group should be called **fear**, and it should use a **systemd:vsftpd** resource to distribute the content found on the NFS share **workstation.storage2.example.com:/exports/ftp** using the IP address **172.25.X.21**. The correct mount point for the NFS share is **/var/ftp**.

1. Create a new resource group called **surprise**, using all resources and requirements outlined.
2. Create a new resource group called **fear**, using all resources and requirements outlined.

## Solution

# Creating and Configuring Resources

### Performance Checklist

In this lab, you will create a three-node cluster with two resource groups.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to create multiple resource groups on a cluster.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, and **nodec** systems.
- From your **workstation** system, run the command **lab resources setup**.

```
[student@workstation ~]$ lab resources setup
```

For an upcoming project you have been requested to create two new resource groups on your **clusterX** cluster.

One resource group should be called **surprise**, and it should provide a web server that distributes the content found on the NFS share **workstation.storage1.example.com:/exports/www** on the IP address **172.25.X.80**. The correct mount point for the NFS share is **/var/www**.

The second resource group should be called **fear**, and it should use a **systemd:vsftpd** resource to distribute the content found on the NFS share **workstation.storage2.example.com:/exports/ftp** using the IP address **172.25.X.21**. The correct mount point for the NFS share is **/var/ftp**.

1. Create a new resource group called **surprise**, using all resources and requirements outlined.

   1.1. Install the **httpd** package on all three of your nodes.

```
[root@nodeY ~]# yum -y install httpd
```

   1.2. On all three of your nodes, configure SELinux so that **httpd** can serve files from NFS.

```
[root@nodeY ~]# setsebool -P httpd_use_nfs 1
```

   1.3. On all three of your nodes, open a port on the firewall to allow **http** traffic.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=http
[root@nodeY ~]# firewall-cmd --reload
```

1.4. Create an **IPaddr2** resource, in the **surprise** resource group, for **172.25.*X*.80/24**.

```
[root@nodea ~]# pcs resource create surpriseip IPaddr2 \
> ip=172.25.X.80 \
> cidr_netmask=24 \
> --group surprise
```

1.5. Create a **Filesystem** resource that read-only mounts the NFS share **workstation.storage1.example.com:/exports/www** on **/var/www**.

```
[root@nodea ~]# pcs resource create surprisefs Filesystem \
> device=workstation.storage1.example.com:/exports/www \
> directory=/var/www \
> fstype=nfs \
> options=ro \
> --group surprise
```

1.6. Create an **apache** resource in the **surprise** resource group.

```
[root@nodea ~]# pcs resource create surpriseserver apache --group surprise
```

**2.** Create a new resource group called **fear**, using all resources and requirements outlined.

2.1. Install the **vsftpd** package on all three of your nodes.

```
[root@nodeY ~]# yum -y install vsftpd
```

2.2. On all three of your nodes, configure SELinux so that **vsftpd** can serve files from NFS.

```
[root@nodeY ~]# setsebool -P ftpd_use_nfs 1
```

2.3. On all three of your nodes, open a port on the firewall to allow **ftp** traffic.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=ftp
[root@nodeY ~]# firewall-cmd --reload
```

2.4. Create an **IPaddr2** resource, in the **fear** resource group, for **172.25.*X*.21/24**.

```
[root@nodea ~]# pcs resource create fearip IPaddr2 \
> ip=172.25.X.21 \
> cidr_netmask=24 \
> --group fear
```

2.5. Create a **Filesystem** resource that read-only mounts the NFS share **workstation.storage2.example.com:/exports/ftp** on **/var/ftp**.

```
[root@nodea ~]# pcs resource create fearfs Filesystem \
> device=workstation.storage2.example.com:/exports/ftp \
> directory=/var/ftp \
> fstype=nfs \
> options=ro \
> --group fear
```

2.6. Create a **systemd:vsftpd** resource in the **fear** resource group.

```
[root@nodea ~]# pcs resource create fearserver systemd:vsftpd --group fear
```

# Summary

In this chapter, you learned:

- How to distinguish between the three main resource types (**LSB**, **OCF**, and **systemd**).
- How to list available resource agents (**pcs resource list**).
- How to view resource agent parameters (**pcs resource describe**).
- How to show configured resources (**pcs resource show**).
- How to configure resource groups.
- How to enable and disable resource groups (**pcs resource enable** and **pcs resource disable**).
- How to move resource groups between nodes (**pcs resource move**).
- How to stop a resource from running on a node (**pcs resource ban**).

Chapter 5

# Troubleshooting High-availability Clusters

**Goal**

Identify and troubleshoot cluster problems.

**Objectives**

- Inspect and configure cluster logging.
- Configure cluster event notifications.
- Debug resource failures.
- Debug cluster network issues.

**Sections**

- Configuring Cluster Logging (and Practice)
- Configuring Cluster Notifications (and Practice)
- Troubleshooting Resource Failures (and Practice)
- Troubleshooting Cluster Networking (and Practice)

**Lab**

- Troubleshooting High-availability Clusters

# Configuring Cluster Logging

## Objectives

After completing this section, students should be able to inspect and configure cluster logging.

## Corosync logging

When troubleshooting a cluster, one of the most useful resources at a system administrator's disposal is the system log. The **corosync** daemon will log to **syslog** by default, but logging can be controlled inside the **logging {}** block in **/etc/corosync/corosync.conf**.

### Logging to a file

To enable logging **corosync** messages to a file, the **to_file** and **logfile** directives can be added to the **logging {}** block.

```
logging {
 to_file: yes
 logfile: /var/log/corosync.log
}
```

### Logging to stderr

**corosync** can also send messages to **stderr**. From here, **systemd** can pick these messages up, and forward them to **journald**.

```
logging {
  to_stderr: yes
}
```

### Logging priorities

The priority at which messages will be written to logging can be controlled with **logfile_priority** and **syslog_priority** settings inside the logging block.

Both can be set to any of the following: **alert**, **crit**, **debug**, **emerg**, **err**, **info**, **notice**, or **warning**. The default level is **info**.

Debug-level logging for everything can also be forced by adding the line **debug: on** to the **logging {}** block in **/etc/corosync/corosync.conf**.

### Activating logging changes

After making changes to logging configuration for **corosync** on a node, distribute the changes to all nodes using the command **pcs cluster sync** from the node which has the updated configuration file. Then, on each of the nodes, run the command **pcs cluster reload corosync** to activate the logging changes.

Instead of reloading **corosync**, an administrator can also choose to restart the entire cluster using **pcs cluster stop --all** and **pcs cluster start --all**.

**Important**

Cluster log files have the ability to grow quite rapidly, especially when debug logging is turned on. Therefore, it is recommended to enable **logrotate** for any new log files. An example of a good **logrotate** configuration for cluster logs can be found in **/etc/logrotate.d/pacemaker**.

## Pacemaker logging

By default, **pacemaker** will log to the same location as **corosync**, but this can be changed by using the **PCMK_log*** options in **/etc/sysconfig/pacemaker**.

After modifying this configuration file, changes can be applied by restarting the **pacemaker.service**. To restart **pacemaker** without affecting cluster operations, a node can be put in **standby** mode before restarting **pacemaker.service**.

**Note**

**journalctl** can also be used to view and query the logs for **pacemaker** and **corosync**:

```
[root@nodeY ~]# journalctl -l -u pacemaker.service -u corosync.service
```

**References**

**corosync.conf**(5) man page

**/etc/sysconfig/pacemaker**

## Guided Exercise

# Configuring Cluster Logging

In this lab, you will modify the cluster logging behavior and read cluster logs.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec**. |

## Outcome(s)

You should be able to configure cluster logging.

- Reset your **workstation** system.
- Reset all four of your **node*Y*** systems.
- From your **workstation** system, run the command **lab resources solve**.

```
[student@workstation ~]$ lab resources solve
```

1. On your **nodea** machine, inspect the default log files for both **pacemaker** and **corosync**, then do the same but using **journalctl**.

   1.1. Inspect the default log file for **pacemaker**.

```
[root@nodea ~]# less /var/log/pacemaker.log
```

   1.2. Inspect the default log file for **corosync**.

```
[root@nodea ~]# less /var/log/messages
```

   1.3. Use the **journalctl** command to inspect the logs for both the **pacemaker.service** and **corosync.service** units.

```
[root@nodea ~]# journalctl -u pacemaker.service -u corosync.service
```

2. Change the configuration for both **corosync** and **pacemaker** to log into the file **/var/log/cluster.log** at debug level. Update the **logrotate** daemon to use the same log rotation settings for this file as for **/var/log/pacemaker.log**. Make sure to activate your changes.

   2.1. Change the configuration for **corosync** to log at **DEBUG** level into **/var/log/cluster.log**. In **/etc/corosync/corosync.conf**, change the **logging** block to the following:

```
logging {
  to_syslog: yes
  to_file: yes
  logfile: /var/log/cluster.log
  debug: on
}
```

2.2. Distribute your new **/etc/corosync/corosync.conf** to all nodes.

```
[root@nodea ~]# pcs cluster sync
```

2.3. Update the **pacemaker** configuration on **nodea**, to log at **DEBUG** level. It will default to using the same log file as **corosync**. On **nodea** add the following line to **/etc/sysconfig/pacemaker**, then copy that file to your other nodes.

```
PCMK_debug=yes
```

```
[root@nodea ~]# scp /etc/sysconfig/pacemaker nodeb:/etc/sysconfig
[root@nodea ~]# scp /etc/sysconfig/pacemaker nodec:/etc/sysconfig
```

2.4. On your **nodea** machine, copy the **logrotate** configuration for **pacemaker** to a new file called **/etc/logrotate.d/cluster**, then update the affected file names to **/var/log/cluster.log**. Distribute the new file to all nodes.

```
[root@nodea ~]# cp /etc/logrotate.d/pacemaker /etc/logrotate.d/cluster
[root@nodea ~]# sed -i 's/pacemaker/cluster/' /etc/logrotate.d/cluster
[root@nodea ~]# scp /etc/logrotate.d/cluster nodeb:/etc/logrotate.d/
[root@nodea ~]# scp /etc/logrotate.d/cluster nodec:/etc/logrotate.d/
```

2.5. Stop and start the cluster on all nodes to activate both the **corosync** and **pacemaker** changes.

```
[root@nodea ~]# pcs cluster stop --all
[root@nodea ~]# pcs cluster start --all
```

**3.** On your **nodea**, open the new log file **/var/log/cluster.log** in *follow* mode, and keep it open for the rest of this exercise.

```
[root@nodea ~]# tail -f /var/log/cluster.log
```

**4.** From your **nodeb** machine, attempt to move the **firstweb** resource group. Observe this move in the logs.

4.1. From your **nodeb** machine, move the **firstweb** resource group.

```
[root@nodeb ~]# pcs resource move firstweb
```

**5.** Disable **DEBUG** logging for both **corosync** and **pacemaker**.

5.1. From **nodea**, remove the **debug: on** line from the **logging** block in **/etc/corosync/corosync.conf**, then distribute your changes to the rest of the cluster.

```
[root@nodea ~]# pcs cluster sync
```

5.2. On all three of your nodes, remove the **PCMK_debug=yes** line from **/etc/sysconfig/pacemaker**.

5.3. Stop and start the cluster on all nodes to activate both the **corosync** and **pacemaker** changes.

```
[root@nodea ~]# pcs cluster stop --all
[root@nodea ~]# pcs cluster start --all
```

**6.** From your **nodeb** machine, attempt to move the **firstweb** resource group. Observe this move in the logs.

6.1. From your **nodeb** machine, move the **firstweb** resource group.

```
[root@nodeb ~]# pcs resource move firstweb
```

**7.** Intentionally break cluster networking on **nodec**, then observe the logs.

7.1. Break the cluster communications on **nodec** by closing off the cluster ports on the firewall.

```
[root@nodec ~]# firewall-cmd --remove-service=high-availability
```

# Configuring Cluster Notifications

## Objectives

After completing this section, students should be able to configure cluster event notifications.

## Configuring MailTo notifications

Most system administrators like to be notified when a resource migrates to a different node, since this is typically an indication of trouble with a node. One of the easiest way to get notifications of this sort is to add a **MailTo** resource to a resource group. Whenever the resource group is started or stopped, the **MailTo** resource will send an email to the configured address.

For example, to add a **MailTo** resource to the **importantgroup** resource group, sending emails to **admin@example.com** with a subject prefix of **importantgroup notification**, the following command can be used:

```
[root@nodeY ~]# pcs resource create importantgroup-mailto MailTo \
> email=admin@example.com \
> subject="importantgroup notification" \
> --group importantgroup
```

The **MailTo** resource agents sends messages using the **mailx** command. It is the responsibility of the system administrator to make sure that the *mailx* package is installed, and that outgoing SMTP connections are allowed from the cluster nodes.

Since **mailx** uses the local MTA for sending messages, the local MTA (**postfix**, for example) will have to be configured to use a *smarthost* where necessary.

## Cluster monitoring with ClusterMon

For more detailed monitoring, the Red Hat High Availability Add-on ships with the **crm_mon** utility, and a resource agent built around that utility: **ClusterMon**.

**crm_mon** can generate cluster status reports in HTML, XML, HTML ready for use as a CGI script, **nagios**-friendly formats, and more. It can be used in *one-shot* mode, where it generates one iteration of output and quits, or it can remain active, updating a status file on disk and optionally calling an external program whenever the status changes.

When using the **ClusterMon** resource agent, a number of important options are available:

| Option | Usage |
|---|---|
| **user** | The user to run the **crm_mon** utility. Defaults to **root**. |
| **update** | The interval in seconds between updates of the HTML file on disk. Defaults to **15** seconds. |
| **extra_options** | Any extra command-line options to pass to **crm_mon**. See **crm_mon**(8) for possibilities. |

| Option | Usage |
|---|---|
| **htmlfile** | The full path of the desired output file. Defaults to **/tmp/ClusterMon_*<RESOURCENAME>*.html**. |

One of the problems an administrator would face when creating a single **ClusterMon** resource is that it would only report the status of the cluster as seen by one node. A solution would be to create a single **ClusterMon** instance for each node, and use resource constraints to fix each instance to a specific node. Not an ideal situation, but an easier solution does exist.

## Resource clones

Whenever it is desirable to have a copy of a resource to run on each node, *resource clones* can be used. Whenever a resource is cloned, a copy of that resource will be started on every node in the cluster. To create a cloned resource, append the **--clone** option to the end of the **pcs resource create** command when creating a new resource.

To clone an existing resource, the command **pcs resource clone *<RESOURCE>*** command can be used. Existing clones can be removed with the **pcs resource unclone** command.

## Using an external notification agent with ClusterMon

Configuring email or SNMP notifications with **ClusterMon** requires an external program or script that performs the actual notification. These scripts can then be called using the **-E *<SCRIPT>*** option in the **extra_options** field for **ClusterMon**.

Additionally, the **-e *<RECIPIENT>*** option can also be used to specify an intended recipient. This options is passed on to the script defined with **-E**.

Parameters are passed into the external agent script using environment variables called **CRM_notify_***.

The following is an example script that can be used to send an email on *every* resource change. A script used in production will need to more aggressively filter which message to pass on, so as to not spam the notification mailbox.

```
#!/bin/bash
MAILTO=${CRM_notify_recipient:-root@localhost}
mutt -s "Cluster event from node ${CRM_notify_node}" ${MAILTO} << EOF

A cluster event was triggered by ${CRM_notify_rsc}
on ${CRM_notify_node}. This event was triggered
by a ${CRM_notify_task} event.

--
This mail has been generated automatically
EOF
```

With this script stored as **/usr/local/bin/myagent.sh**, a **ClusterMon** resource that uses this script can be configured as follows:

```
[root@nodeY ~]# pcs resource create myagent ClusterMon \
> extra_options="-E /usr/local/bin/myagent.sh \
> -e admin@example.com" \
> --clone
```

**References**

**ocf_heartbeat_MailTo**(7), **crm_mon**(8), and **ocf_pacemaker_ClusterMon**(7) man pages

Red Hat High Availability Add-On Reference Guide

- Event Notification With Monitoring Resources

## Guided Exercise

# Configuring Cluster Notifications

In this lab, you will configure notifications for various cluster actions.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to configure notifications for cluster events.

- Reset your **workstation** machine.
- Reset all four of your **node*Y*** machines.
- From your **workstation** system, run the command **lab monitor setup**.

```
[student@workstation ~]$ lab monitor setup
```

1. Configure the **firstweb** resource group to automatically send an email with the subject "CLUSTER-NOTIFICATION" to **student@workstation.cluster*X*.example.com** whenever a status change occurs on that resource group. This resource should be named **webmail**.

   **postfix** on your **workstation** machine has already been configured to accept mail from the outside.

   The simple act of updating this resource group should already trigger some messages. Use **mutt** on your **workstation** system to read these messages.

   1.1. From your **nodea** system, add a **MailTo** resource named **webmail** to the **firstweb** group with a subject of **CLUSTER-NOTIFICATION** and a recipient of **student@workstation.cluster*X*.example.com**.

```
[root@nodea ~]# pcs resource create webmail MailTo \
> email=student@workstation.clusterX.example.com \
> subject="CLUSTER-NOTIFICATION" \
> --group firstweb
```

   1.2. On your **workstation** system, open **mutt** as the user **student**, and view the generated message.

   Repeatedly pressing **q** will exit **mutt**.

```
[student@workstation ~]$ mutt
```

2. Add a (cloned) **ClusterMon** resource to your cluster, named **mailme**, that calls the external script **/usr/local/bin/crm_notify.sh** with a recipient of

**student@workstation.clusterX.example.com**. This custom script sends a simple notification of the status change to the recipient listed. In a production environment, a script that more aggressively filters messages might be a more prudent choice.

After adding the cloned resource, check the mail for **student** on your **workstation** machine to see if the resource works.

2.1. Add a (cloned) **ClusterMon** resource to your cluster, named **mailme**, that calls the external script **/usr/local/bin/crm_notify.sh** with a recipient of **student@workstation.clusterX.example.com**.

```
[root@nodea ~]# pcs resource create mailme ClusterMon \
> extra_options="-e student@workstation.clusterX.example.com \
> -E /usr/local/bin/crm_notify.sh" \
> --clone
```

2.2. Check the messages for **student** on your **workstation** machine.

```
[student@workstation ~]$ mutt
```

**3.** Manually move the **firstweb** resource group, and view the messages created by both the **MailTo** resource and the **ClusterMon** resource.

3.1. Manually move the **firstweb** resource group.

```
[root@nodea ~]# pcs resource move firstweb
```

3.2. View the resulting flurry of emails.

```
[student@workstation ~]$ mutt
```

**4.** Force your **nodec** machine to become fenced by interrupting cluster communications. View the resulting emails from your **mailme** resource.

4.1. Force your **nodec** machine to become fenced by interrupting cluster communications.

```
[root@nodec ~]# firewall-cmd --remove-service=high-availability
```

4.2. As **student** on **workstation**, read the resulting messages.

```
[student@workstation ~]$ mutt
```

# Troubleshooting Resource Failures

## Objectives

After completing this section, students should be able to debug resource failures.

## Resource failures

Resources can fail for multiple reasons. An administrator might have used incorrect settings when defining the resource, a configuration file might have an error in it, the system might be trying to start a resource that does not exist, or some other unforeseen issue might occur.

Whenever a resource fails, the cluster will increase the **`failcount`** for a resource. This count can be viewed with the command **`pcs resource failcount show <RESOURCE>`**.

Failure to start or stop will immediately set the failcount for a resource to **`INFINITY`**, forcing it to move to a different node. If fencing is enabled, a node that failed to stop a resource will also be fenced.

Resources can be configured to relocate to a different node after *N* amount of failures as well, by setting the option **`meta migration-threshold=N`** when creating or modifying the resource. By default, resources will not migrate unless their failcount reaches **`INFINITY`**.

## Troubleshooting resource failures

There are a number of steps an administrator can take when troubleshooting a resource failure:

- Inspect the log files for the affected resources.
- Inspect the cluster log files.
- Verify resource configuration.
- Verify configuration files.
- Attempt to manually start the resource with **`debug-start`**.

### Inspect log files

One of the first actions an administrator will likely take is inspecting the log files for both the cluster itself, and any log files the affected resource might generate. Be careful when reading these log files, as the actual error might be a single small warning, while the resulting failure cascade following the first error might generate far more logging.

### Verify resource configuration

Using the command **`pcs resource show --full`** or **`pcs resource show <RESOURCE>`**, inspect the cluster configuration for the failed resource. Small typos here, or missed options, might have a drastic effect.

## Verify configuration files

If the affected resource has its own configuration file, like an **apache** resource, it might also have its own configuration file validation tool, like **apacectl configtest**. Ensuring that the resource can start eliminates a whole slew of potential failures.

## Manually starting a resource with debug-start

When a resource has reached a failcount of **INFINITY** on all nodes, it is no longer possible to attempt to start that resource automatically. An administrator can still attempt to start the resource with the command **pcs resource debug-start *<RESOURCE>***. This will result in a short status output on both failure and success. Adding in the **--full** option as well will generate full debugging output, which can assist troubleshooting.

**Warning**

In some cases, when a resource agent fails to start, the **pcs resource debug-start** command might wait forever for the resource to start. In these cases, an administrator can use **Ctrl+C** to exit the **debug-start** command.

# Fixing resource failures

Updating a cluster resource definition will automatically reset the failcount for that resource, enabling its use on the cluster again.

When the performed fix was performed outside of the cluster configuration —for example, by updating a service configuration file— the failcount will remain, preventing the resource from being started. In those cases, an administrator can run the command **pcs resource failcount reset *<RESOURCE>*** to manually reset the failcount, enabling the resource.

**References**

Red Hat High Availability Add-On Reference Guide

- Moving Resources Due to Failure
- Enabling, Disabling, and Banning Cluster Resources

## Guided Exercise

# Troubleshooting Resource Failures

In this lab, you will troubleshoot a failed resource.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to troubleshoot a resource startup failure.

- Reset your **workstation** system.
- Reset all of your **node*Y*** machines.
- From your **workstation** machine, run the command **lab resourcefailure setup**.

```
[student@workstation ~]$ lab resourcefailure setup
```

One of your colleagues has created a new cluster named **cluster*X***, with a resource group called **firstweb** running on it. Almost everything is working, but your colleague cannot get the **firstweb** resource group to start. This resource group should provide regular **http** service on **http://172.25.*X*.80**, using port **80**, serving content from **/var/www/html**. **/var/www** should be mounted over NFS from **workstation.storage1.example.com:/exports/www**.

1. Wait for your cluster to stabilize. You can monitor the status of your cluster with **pcs status**. You can continue when the **firstweb** group has finished jumping between hosts.

2. Inspect the output of **pcs status**. Which resource is stopping the **firstweb** resource group from starting properly?

   2.1. According to **pcs status**, the **firstwebserver** resource failed to start on all three nodes.

3. Inspect the **failcount** for the **firstwebserver** resource, then perform a forced **debug-start** with extra verbosity on the **firstwebserver** resource. Perform this action on the node that is currently running the other resources in the **firstweb** group.

   Carefully analyze the resulting output, and investigate the problem.

   3.1. View the **failcount** for the **firstwebserver** resource.

```
[root@nodea ~]# pcs resource failcount show firstwebserver
```

   3.2. Perform a forced **debug-start** of the **firstwebserver** resource, with extra verbosity. Use the node that is currently hosting the other resources in the **firstweb** resource group.

```
[root@nodeY ~]# pcs resource debug-start firstwebserver --full
```

3.3. There are a couple of errors in the output from **debug-start**. The main one that gets reported is **Port number is invalid!**, the same message that appears in the output of **pcs status**.

A number of lines above this message, a second, seemingly less severe, error is reported:

```
GetParams:135: '[' '!'' -f /etc/httpd/conf/httpd.conf ']'
GetParams:136: return 5
```

This error seems to indicate that the specified configuration file cannot be found.

3.4. View the resource configuration for the **firstwebserver** resource.

```
[root@nodeY ~]# pcs resource show firstwebserver
 Resource: firstwebserver (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf
...
```

**4.** Fix the configuration problem, then inspect the **failcount** for the resource group.

4.1. Fix the configuration problem by removing the incorrect **configfile** attribute from the **firstwebserver** resource.

```
[root@nodeY ~]# pcs resource update firstwebserver configfile=
```

4.2. Inspect the failcount for the **firstwebserver** resource.

```
[root@nodeY ~]# pcs resource failcount show firstwebserver
```

Updating the resource has automatically reset the **failcount**.

4.3. Verify that the **firstweb** resource group has properly started, and is serving out content.

```
[root@nodeY ~]# pcs status
```

# Troubleshooting Cluster Networking

## Objectives

After completing this section, students should be able to debug cluster network issues.

## Identifying network issues

Faulty, or misconfigured, network connections can wreak havoc on a cluster. The following is a list of possible issues and fixes.

### Unicast-only networks

Some network switches actively block multicast traffic. If a cluster transport is set to **udp**, this enables multicast communications.

Possible fixes include switching the traffic to **udpu** (UDP Unicast), or enabling multicast on the network switches.

### Firewall issues

Incorrectly configured firewalls can make a machine unreachable to the other nodes. Make sure that all high-availability services can be reached, as well as the network ports on the public network for any clustered services that are offered to consumers.

### Split networks

If multiple cluster nodes are plugged into different switches, and the connection between those switches drops, the cluster will go into a split-brain mode, losing a number of nodes. The risk of these types of failures occurring can be reduced by using redundant networking and interconnects.

### Dropped packets

When a network link gets oversaturated, packets might be dropped, resulting in weird and intermittent cluster failures. These scenarios can be avoided by using separate networks for private cluster communications, public client access, and storage networks.

**References**

**`corosync.conf`**(5) man page

## Guided Exercise

# Troubleshooting Cluster Networking

In this lab, you will troubleshoot an issue with cluster networking.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to troubleshoot cluster networking issues.

- Reset your **workstation** system.
- Reset all of your **node*Y*** systems.
- From your **workstation** system, run the command **lab networkfailure setup**

```
[student@workstation ~]$ lab networkfailure setup
```

Overnight, your **cluster*X*** cluster fenced **nodec**, and that node has since not rejoined the cluster.

Since your three-node cluster is now running with only two nodes, any further node failures will take the cluster down.

Operations has asked you to investigate and fix the issue.

1. Start by gathering some information from **pcs status** and **corosync-quorumtool**.

   1.1.

```
[root@nodea ~]# pcs status
```

   1.2.

```
[root@nodea ~]# corosync-quorumtool
```

2. It appears that **nodec** is not reachable for both **corosync** and **pcsd**. From your **nodea**, machine check if the different network interfaces on **nodec** are still pingable.

   2.1.

```
[root@nodea ~]# for I in nodec.{clusterX,private,storage{1,2}}.example.com; do
> ping -c1 ${I}
> done
```

   2.2. All four interfaces are still responding to **ping** requests.

3. Log into your **nodec** system, then inspect how **nodec** views the cluster.

3.1.

```
[root@nodec ~]# corosync-quorumtool
```

3.2.

```
[root@nodec ~]# pcs status
```

3.3. Both the output from **corosync-quorumtool** and **pcs status** show that the cluster infrastructure services on **nodec** are active. This implies that cluster communications must be degraded in some way.

**4.** What could cause cluster communications to be dropped, while services like **ping** and **ssh** still work?

Investigate, and fix when applicable.

4.1. A firewall issue could be the root cause for cluster communications failure. View the firewall configuration on **nodec**.

```
[root@nodec ~]# firewall-cmd --list-all
  interfaces: eth0 eth1 eth2 eth3
  sources:
  services: dhcpv6-client http ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
```

4.2. The **high-availability** service is notably absent in the **firewall-cmd** output. Add this service back to the firewall configuration.

```
[root@nodec ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodec ~]# firewall-cmd --reload
```

4.3. Verify that the cluster is now fully operational again.

```
[root@nodec ~]# pcs status
```

## Lab

# Troubleshooting High-availability Clusters

### Performance Checklist

In this lab, you will troubleshoot a four-node cluster with two faulty resource groups.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, **nodec**, and **noded** |

### Outcome(s)

You should be able to troubleshoot a cluster.

- Reset your **workstation** system.
- Reset all of your **node*Y*** systems.
- From your **workstation** system, run the command **lab clustertrouble setup**

```
[student@workstation ~]$ lab clustertrouble setup
```

You have recently been asked to take over management of the **cluster*X*** cluster. This cluster should provide two services to consumers:

- A **http** service available on **http://172.25.*X*.80**, serving content from the NFS share **workstation.storage1.example.com:/exports/www**.
- An **ftp** service available on **ftp://172.25.*X*.21**, serving content from the NFS share **workstation.storage2.example.com:/exports/ftp**.

While both services are configured on the cluster, consumers cannot reach either one. Investigate and fix these issues.

1. Begin by investigating the current cluster, including all resources defined.
2. Investigate and fix the issue with the FTP service.
3. Investigate and fix the issue with the HTTP service.

## Solution

# Troubleshooting High-availability Clusters

### Performance Checklist

In this lab, you will troubleshoot a four-node cluster with two faulty resource groups.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, **nodec**, and **noded** |

### Outcome(s)

You should be able to troubleshoot a cluster.

- Reset your **workstation** system.
- Reset all of your **node*Y*** systems.
- From your **workstation** system, run the command **lab clustertrouble setup**

```
[student@workstation ~]$ lab clustertrouble setup
```

You have recently been asked to take over management of the **cluster*X*** cluster. This cluster should provide two services to consumers:

- A **http** service available on **http://172.25.*X*.80**, serving content from the NFS share **workstation.storage1.example.com:/exports/www**.
- An **ftp** service available on **ftp://172.25.*X*.21**, serving content from the NFS share **workstation.storage2.example.com:/exports/ftp**.

While both services are configured on the cluster, consumers cannot reach either one. Investigate and fix these issues.

1. Begin by investigating the current cluster, including all resources defined.

   1.1. Start with a **pcs status** command.

```
[root@nodea ~]# pcs status
```

   This gives us the following facts:

   - **cluster*X*** is a four-node cluster, consisting of **nodea.private.example.com**, **nodeb.private.example.com**, **nodec.private.example.com**, and **noded.private.example.com**.
   - There are two resource groups defined: **troubleweb** and **troubleftp**.

- According to the cluster, all resources in the **troubleftp** group are running fine.
- The cluster has trouble starting the **webserver** resource in the **troubleweb** group.

1.2. View the full configuration for each resource defined in both the **troubleweb** and **troubleftp** resource groups.

```
[root@nodea ~]# pcs resource show troubleweb
[root@nodea ~]# pcs resource show troubleftp
```

**2.** Investigate and fix the issue with the FTP service.

2.1. Start by attempting to use the FTP service on **ftp://172.25.*X*.21**.

On your **workstation** system, open a browser and navigate to `ftp://172.25.X.21`.

2.2. With the last step failed, verify that the **troubleftp** group is using the correct IP address, and that the IP address is reachable from **workstation**.

On your **nodea** machine, inspect the **ftpip** resource.

```
[root@nodea ~]# pcs resource
            show ftpip
```

From your **workstation** system, attempt to **ping** the floating IP address.

```
[student@workstation ~]$ ping 172.25.X.21
```

2.3. With the correct IP address being used, and reachable, and the **system:vsftpd** resource running without issues, a firewall problem is the most likely culprit. Inspect the current firewall configuration on your nodes.

```
[root@nodeY ~]# firewall-cmd --list-all
```

2.4. On all four of your nodes, permanently add the missing **ftp** service to your firewall.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=ftp
[root@nodeY ~]# firewall-cmd --reload
```

2.5. On your **workstation** system, open a browser and navigate to `ftp://172.25.X.21`. This should now show a working FTP service.

**3.** Investigate and fix the issue with the HTTP service.

3.1. Look up on which node the working parts of the **troubleweb** group are currently running.

```
[root@nodea ~]# pcs status
```

3.2. On the node that you found in the previous step, search the default **pacemaker** log location for any entries referencing **webserver** and **ERROR**.

```
[root@nodeY ~]# grep 'webserver.*ERROR' /var/log/pacemaker.log
apache(webserver)[11096]: 2015/06/18_13:53:38 ERROR: AH00526: Syntax error on line
 1 of /etc/httpd/conf.d/oops.conf: Invalid command 'Oops!', perhaps misspelled or
 defined by a module not included in the server configuration
apache(webserver)[11953]: 2015/06/18_13:54:18 ERROR: Port number  is invalid!
```

The second error might suggest a configuration error regarding a **httpd Listen** directive, but the first error message shows the actual problem.

3.3. On all four of your nodes, fix the **httpd** configuration by removing the offending configuration file.

```
[root@nodeY ~]# rm /etc/httpd/conf.d/oops.conf
```

3.4. Reset the **failcount** for the **webserver** resource on all nodes.

```
[root@nodea ~]# pcs resource failcount reset webserver
```

3.5. Verify that the **troubleweb** resource group is operating normally.

```
[root@nodea ~]# pcs resource
```

On your **workstation** system, open a web browser and navigate to `http://172.25.X.80`.

# Summary

In this chapter, you learned:

- **`corosync`** logging is configured in **`/etc/corosync/corosync.conf`**.
- **`pacemaker`** logging follows the **`corosync`** configuration, unless overridden in **`/etc/sysconfig/pacemaker`**.
- A **`MailTo`** resource can be added to a resource group to get notifications when the group migrates.
- **`ClusterMon`** resources can update a status file on disk, and call an external script on events.
- Cloned resources run on every available node.
- Failed resources can be debugged with **`pcs resource debug-start`**.

Chapter 6

# Controlling Complex Resource Groups

| | |
|---|---|
| **Goal** | Create complex resource groups using ordering and location constraints. |
| **Objectives** | • Demonstrate how resource groups can be used to control resource startup order by configuring an active/passive NFS server resource group.<br>• Set constraints manually for resource groups or individual resources to influence which nodes they run on. |
| **Sections** | • Managing Resource Startup Order (and Practice)<br>• Configuring Location Constraints (and Practice) |
| **Lab** | • Controlling Complex Resource Groups |

# Managing Resource Startup Order

## Objectives

After completing this section, students should be able to demonstrate how resource groups can be used to control resource startup order by configuring an active/passive NFS server resource group.

## What are constraints?

Constraints are restrictions that determine the order in which resources can be started and stopped, on which nodes they can run, or which other resources they can share the node with. Resource groups provide an easy implicit ordering constraint configuration that is sufficient for many use cases, as they provide a convenient shortcut to setting up ordering constraints. Resources that are part of the same resource group:

- Start in the defined sequence.
- Stop in the reverse order.
- Always run on the same cluster node.

## Configuring an active/passive NFS resource group

For providing a highly available **NFS** service, it is mandatory that all required resources run on the same cluster node. The cluster resources required for an active/passive **NFS** server must start services in a particular order. These requirements can be fulfilled by setting up the resources required for the **NFS** service to execute in the following order:

1. Start the **`Filesystem`** resource, which mounts the file system to export from shared storage.
2. Start the **`nfsserver`** resource, which controls the **NFS** system service.
3. Start one or more **`exportfs`** resource(s), responsible for exporting the **NFS** shared file systems.
4. Start the **`IPaddr2`** floating IP address resource.

By placing these resources in one resource group in the correct order, automatic constraints allow the service to function correctly.

Creating a highly available **NFS** export with the Red Hat High Availability Add-on requires the following procedure:

1. First, a shared file system resource, such as an iSCSI partition, needs to be created and formatted with either an **XFS** or an **EXT4** file system. The firewall must allow connections to an **NFSv4** server on all cluster nodes that will run the resource group that provides the **NFSv4** share.
2. The **`Filesystem`** resource must be started before and stopped after the **`nfsserver`** resource. This is important for stopping the resource group, because the file system cannot be unmounted if the **NFS** server is still running and there are **NFSv4** leases active. To create the **`Filesystem`** resource named **`nfsfs`**, with the **XFS**-formatted shared storage device **`/`**

**dev/sdb1** and the mount point directory **/myshare** as part of the **mynfs** resource group, execute:

```
[root@nodeY ~]# pcs resource create nfsfs Filesystem device=/dev/sdb1 directory=/
myshare fstype=xfs --group mynfs
```

3. The **NFS** server resource is started after the **Filesystem** resource. An **nfs_shared_infodir** on the shared storage is provided for the **NFS** server to maintain client data on the shared storage for failover recovery. To add the **nfsserver** resource to the resource group **mynfs**, with name **nfssvc** and the **nfs_shared_infodir** set to **/nfsshare/nfsinfo**, run:

```
[root@nodeY ~]# pcs resource create nfssvc nfsserver nfs_shared_infodir=/nfsshare/
nfsinfo --group mynfs
```

4. The **exportfs** resource agent allows for exporting one or more file systems as **NFSv4** shares to a specified host. The exports must be started after the **NFS** server. To create an **NFS** root export of the **/nfsshare** directory named **nfsfs1**, mountable by the **172.18.20.15/32** client with **rw,sync,no_root_squash** options as part of the **mynfs** resource group, run:

```
[root@nodeY ~]# pcs resource create nfsfs1 exportfs clientspec=172.18.20.15/32
 options=rw,sync,no_root_squash directory=/nfsshare fsid=0 --group mynfs
```

5. The **NFS** service requires an IP address on the public network for client access. For that, an IP address resource is required. The IP address resource must be started after all **exportfs** resources to ensure all **NFS** exports are available when a client successfully connects to the clustered **NFS** share. The **IPaddr2** resource named **nfsip** with IP **172.16.20.83/24** as part of the **mynfs** resource group is created by executing:

```
[root@nodeY ~]# pcs resource create nfsip IPaddr2 ip=172.16.20.83 cidr_netmask=24
 --group mynfs
```

The resulting highly available **NFS** service can be mounted as an **NFSv4** share on a client. In the event of a failover, the **NFSv3** client may pause up to 5 seconds and the **NFSv4** client may pause up to 90 seconds while the file locks are recovered.

**References**

**pcs**(8) man page

## Guided Exercise

# Managing Resource Startup Order

In this lab, you will configure a highly available **NFS** service.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

## Outcome(s)

You should be able to configure an active-passive highly available **NFS** service in an active-passive service that provides a share to the **workstation** system.

- Reset your **workstation** and all four **nodeY** systems.
- From your **workstation** system, run the command **lab nfs setup**.

**1.** Prepare a new **XFS**-formatted partition of 256 MiB on the iSCSI block device.

1.1. From **nodea**, create a 256 MiB primary partition on **/dev/sda**.

```
[root@nodea ~]# fdisk /dev/sda
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): <Enter>
First sector (270336-8388607, default 8192): <Enter>
Using default value 270336
Last sector, +sectors or +size{K,M,G} (270336-8388607, default 8388607): +256M
Partition 1 of type Linux and of size 256 MiB is set
Command (m for help): w
```

1.2. Make sure that all nodes see the new partition by running the **partprobe** command.

```
[root@nodeY ~]# partprobe
```

1.3. Format the **/dev/sda1** partition with the **XFS** file system.

```
[root@nodea ~]# mkfs -t xfs /dev/sda1
```

2. Create a highly available **NFS** service in an active-passive setup as resource group **nfs** and allow access to the **NFSv4** share through the firewall on all nodes that host the **nfs** resource group.

   2.1. Stop and disable the local nfs-lock service on *all* nodes. This will be controlled by pacemaker.

```
[root@nodeY ~]# systemctl stop nfs-lock;systemctl disable nfs-lock
```

   2.2. Create the **Filesystem** resource named **nfsshare** using the **XFS**-formatted device **/dev/sda1** and the mount point directory **/nfsshare** as part of the **nfs** resource group.

```
[root@nodea ~]# pcs resource create nfsshare Filesystem device=/dev/sda1
 directory=/nfsshare fstype=xfs --group nfs
```

   2.3. Create the **nfsserver** resource named **nfsd** as part of the **nfs** group. The **nfs_shared_infodir** is set to **/nfsshare/nfsinfo**.

```
[root@nodea ~]# pcs resource create nfsd nfsserver nfs_shared_infodir=/nfsshare/
nfsinfo --group nfs
```

   2.4. Create the **exportfs** resource named **nfsroot** to export the **/nfsshare** directory as an **NFSv4** share to the all hosts with the **NFS** options **rw,sync,no_root_squash** as part of the **nfs** resource group.

```
[root@nodea ~]# pcs resource create nfsroot exportfs clientspec="*"
 options=rw,sync,no_root_squash directory=/nfsshare fsid=0 --group nfs
```

   2.5. Create the **IPaddr2** resource named **nfsip** with IP **172.25.*X*.82/24** as part of the **nfs** resource group.

```
[root@nodea ~]# pcs resource create nfsip IPaddr2 ip=172.25.X.82 cidr_netmask=24
 --group nfs
```

   2.6. Allow access to the **nfs** server through the firewall on all cluster nodes if not already allowed.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=nfs
[root@nodeY ~]# firewall-cmd --reload
```

3. Validate the clustered **NFS** setup is accessible as an **NFSv4** share by mounting it on the newly created mount point **/mnt/nfsv4share** and creating a new empty file **/mnt/nfsv4share/myfile.txt**.

   3.1. Create the mount point **/mnt/nfsv4share** on **workstation**.

```
[root@workstation ~]# mkdir /mnt/nfsv4share
```

   3.2. Mount the clustered **NFSv4** share **172.25.*X*.82:/** as an **NFSv4** export on the **/mnt/nfsv4share** mount point on **workstation**.

```
[root@workstation ~]# mount 172.25.X.82:/ /mnt/nfsv4share
```

3.3. Create the empty file **/mnt/nfsv4share/myfile.txt** on **workstation**.

```
[root@workstation ~]# touch /mnt/nfsv4share/myfile.txt
```

**4.** Verify the **NFS** share is still available on **workstation** after a forced failover.

4.1. Determine the cluster node where the **nfs** resource group is running.

```
[root@nodea ~]# pcs status
...
Resource Group: nfs
 nfsshare        (ocf::heartbeat:Filesystem): Started nodea.private.clusterX
...
```

4.2. Pause the hosting of cluster resources on the cluster node where the **NFS** service is running. The node running the nfs group may vary. The following steps assume **nodea**.

```
[root@nodeb ~]# pcs cluster standby nodea.private.example.com
```

4.3. Verify the **NFSv4** mount is still accessible on **workstation** after waiting up to 90 seconds.

```
[root@workstation ~]# ls /mnt/nfsv4share
myfile.txt nfsinfo
```

**5.** Enable the node that was put into standby mode to host resources again. Unmount the **NFSv4** share on **workstation**.

5.1. Enable the node that was put into standby mode to host resources again.

```
[root@nodeb ~]# pcs cluster unstandby nodea.private.example.com
```

5.2. Unmount the **NFSv4** share on **workstation**.

```
[root@workstation ~]# umount /mnt/nfsv4share
```

# Configuring Location Constraints

## Objectives

After completing this section, students should be able to demonstrate how to set constraints manually for resource groups or individual resources to influence the order in which they start or the nodes on which they run.

## Managing constraints

*Constraints* are rules that place restrictions on the order in which resources or resource groups may be started, or the nodes on which they may run. Constraints are important for managing complex resource groups or sets of resource groups, which depend upon one another or which may interfere with each other.

There are three main types of constraints:

- *Order* constraints, which control the order in which resources or resource groups are started and stopped.
- *Location* constraints, which control the nodes on which resources or resource groups may run.
- *Colocation* constraints, which control whether two resources or resource groups may run on the same node.

Resource groups are the easiest way to set constraints on resources. All resources in a resource group implicitly have colocation constraints on each other that is, they must run on the same node. Resources that are members of a resource group also have order constraints on each other; they must start in the order in which they were added to the group, and they are stopped in the reverse of the startup order.

There are times when it is useful to configure explicit constraints on resources or resource groups manually. For example, it may be that two resource groups need to run on different cluster nodes, but start and stop their resources independently, or it may be that the cluster administrator wants a resource group to preferentially run on a particular cluster node if it is available. Explicit constraints can make these scenarios possible.

**Important**

Manual constraints should be set on resource groups as a whole and not on individual resources in the resource group, as the resource group may break or other unexpected effects may occur.

## Configuring order constraints

Order constraints may be the simplest to understand. Order constraints mandate the order in which services must start. This may be important if, for example, the resource group for a high-availability PostgreSQL database, its IP addresses, and other resources must be started before the resource group for some high-availability service that accesses the database on startup.

To set an order constraint between two resources or resource groups:

```
[root@nodeY ~]# pcs constraint order A then B
Adding A B (kind: Mandatory) (Options: first-action=start then-action=start)
[root@nodeY ~]#
```

The preceding command sets a mandatory order constraint between the two resources or resource groups *A* and *B*. This has the following effects on the operation of these resources or resource groups:

- If both resources are stopped, and *A* is started, then *B* is also allowed to start.
- If both resources are running, and *A* is disabled by pcs, then the cluster will stop *B* before stopping *A*.
- If both resources are running, and *A* is restarted, then the cluster will also restart *B*.
- If *A* is not running and the cluster cannot start it, *B* will be stopped. This can happen if the first resource is misconfigured or broken, for example.

# Viewing and removing constraints

## Viewing constraints

The **pcs constraint** command (or **pcs constraint list**) can be used to view the current constraints set for the cluster's resources. Using it with the **--full** option provides more detail, including the ID of the constraints.

```
[root@nodeY ~]# pcs constraint
Location Constraints:
Ordering Constraints:
  start testip then start webfarm
Colocation Constraints:
[root@nodeY ~]# pcs constraint --full
Location Constraints:
Ordering Constraints:
  start testip then start webfarm (Mandatory) (id:order-testip-webfarm-mandatory)
Colocation Constraints:
[root@nodeY ~]#
```

## Removing constraints

The **pcs constraint remove *id*** command can be used to delete a constraint. The *id* can be obtained from the **pcs constraint --full** command.

```
[root@nodeY ~]# pcs constraint remove order-testip-webfarm-mandatory
[root@nodeY ~]# pcs constraint
Location Constraints:
Ordering Constraints:
Colocation Constraints:
[root@nodeY ~]#
```

# Configuring location constraints

Location constraints are somewhat more complex than order constraints. A location constraint controls the node on which a resource or resource group will run. If no other influences or

constraints apply, the cluster software will normally try to spread resources evenly around the cluster. A location constraint on a resource will cause it to have a preferred node or nodes. The node selected is influenced by the constraint's *score*. The complexity arises in that the node chosen by the cluster also takes into effect the location preferences of resources with which the resource is collocated. If resource **A** must be collocated with resource **B**, and **B** can only run on node1, then it doesn't matter that **A** prefers node2.

## Score and score calculations

Score determines the node on which a resource will be started. The possible scores and their effects are:

- **INFINITY**: The resource must run here.
- Positive number or zero: The resource should run here.
- Negative number: The resource should not run here (will avoid this node).
- **-INFINITY**: The resource must not run here.

The resource will relocate to the node with the highest score that is available. If two nodes have the same score (for example, if two nodes have a score of **INFINITY**), then the cluster will start the resource on one of those two nodes. Generally, the cluster software will prefer a node that is not already running a resource.

If multiple constraints or scores apply, they are added together and the total score applies. If a score of **INFINITY** is added to a score of **-INFINITY**, the resulting score is **-INFINITY**. (That is, if a constraint is set so that a resource should always avoid a node, that constraint wins.)

If no node with a sufficiently high score can be found, the resource will not be started.

## Viewing current scores

The **crm_simulate -sL** command will display the scores (**-s**) currently allocated to resources, resource groups, and stonith devices on the live cluster (**-L**). Since multiple scores may apply, care needs to be taken when interpreting the output of the command.

```
[root@nodeY ~]# crm_simulate -sL
Current cluster status:
Online: [ nodea nodeb nodec ]
 fence_nodeb_virt       (stonith:fence_xvm):    Started nodea
 fence_nodea_virt       (stonith:fence_xvm):    Started nodec
 Resource Group: webfarm
     testip     (ocf::heartbeat:IPaddr2):       Started nodea
     apache     (ocf::heartbeat:IPaddr2):       Started nodea
 fence_nodec_virt       (stonith:fence_xvm):    Started nodea

Allocation scores:
native_color: fence_nodeb_virt allocation score on nodea: 0
native_color: fence_nodeb_virt allocation score on node2: 0
native_color: fence_nodeb_virt allocation score on nodec: 0
native_color: fence_nodea_virt allocation score on node1: 0
native_color: fence_nodea_virt allocation score on nodeb: 0
native_color: fence_nodea_virt allocation score on nodec: 0
group_color: webfarm allocation score on nodea: 100
group_color: webfarm allocation score on nodeb: 0
group_color: webfarm allocation score on nodec: 0
```

```
group_color: testip allocation score on nodea: 100
group_color: testip allocation score on nodeb: 0
group_color: testip allocation score on nodec: 0
group_color: apache allocation score on nodea: 0
group_color: apache allocation score on nodeb: 0
group_color: apache allocation score on nodec: 0
native_color: testip allocation score on nodea: 100
native_color: testip allocation score on nodeb: 0
native_color: testip allocation score on nodec: 0
native_color: apache allocation score on nodea: 0
native_color: apache allocation score on nodeb: -INFINITY
native_color: apache allocation score on nodec: -INFINITY
native_color: fence_nodec_virt allocation score on nodea: 0
native_color: fence_nodec_virt allocation score on nodeb: 0
native_color: fence_nodec_virt allocation score on node3: 0

Transition Summary:
[root@nodeY ~]#
```

In the preceding example, a location constraint was set such that the **webfarm** resource group prefers **nodea** with a score of 100. The cluster started its **testip** resource there. Once that was running, the resource group automatically set the score for its **apache** resource to **-INFINITY** on all other nodes in the cluster so that it had to also start on **nodea**. (All resources in a resource group must run on the same cluster node.)

**Note**

The **crm_simulate** command can also be used as a troubleshooting tool to predict how resources will relocate given a particular cluster configuration and sequence of events. How to use the tool in this manner is beyond the scope of this chapter.

## Setting location constraints

To set a mandatory location constraint:

```
[root@nodeY ~]# pcs constraint location id prefers node
```

The preceding command will set the resource or resource group *id* to have a score of **INFINITY** for running on *node*. If no other location constraints exist, this will force *id* to always run on *node* if it is up; otherwise, it will run on any other node in the cluster. The change will take effect immediately, relocating the resource if necessary.

A lower score can be set instead. If multiple nodes have different scores, the highest available will be used. This can be used to implement a priority ranking of preferred nodes for a resource or resource group.

For example, to set a score of 500 on a location constraint:

```
[root@nodeY ~]# pcs constraint location id prefers node=500
```

A resource or resource group can be told to avoid a particular node instead. The following command sets a location constraint with a score of **-INFINITY**, which will force the resource to run on another node if possible. (If no other nodes are available, the resource will refuse to start.)

```
[root@nodeY ~]# pcs constraint location id avoids node
```

**Important**

The **pcs resource move *id node*** and **pcs resource ban *id node*** commands set *temporary* location constraints. These constraints can be cleared with **pcs resource clear *id***, but that command has no effect on location constraints set with **pcs constraint**.

## Avoiding unnecessary resource relocation

Pacemaker assumes that resource relocation has no cost by default. In other words, if a higher-score node becomes available, Pacemaker will relocate the resource to that node. This can cause extra unplanned downtime for that resource, especially if it is expensive to relocate. (For example, the resource may take significant time to relocate.)

A default *resource stickiness* can be set that establishes a score for the node on which a resource is currently running. For example, assume the resource stickiness is set to 1000, and the resource's preferred node has a location constraint with a score of 500. On resource start, it will run on the preferred node. If the preferred node crashes, the resource will move to one of the other nodes and a score of 1000 will be set for it on that node. When the preferred node comes back, it only has a score of 500 and the resource will not automatically relocate to the preferred node. The cluster administrator will need to manually relocate the resource to the preferred node at a convenient time (perhaps during a planned outage window).

To set a default resource stickiness of 1000 for all resources:

```
[root@nodeY ~]# pcs resource defaults resource-stickiness=1000
```

To view current resource defaults:

```
[root@nodeY ~]# pcs resource defaults
```

To clear the resource-stickiness setting:

```
[root@nodeY ~]# pcs resource defaults resource-stickiness=
```

**Important**

Note that resource stickiness for a resource group is calculated based on how many resources are running in that group. If a resource group has five active resources and resource stickiness is 1000, then the resource group has an effective score of 5000.

# Configuring colocation constraints

Colocation constraints specify that two resources must (or must not) run on the same node. To set a colocation constraint to keep two resources or resource groups together:

```
[root@nodeY ~]# pcs constraint colocation add B with A
```

The preceding command will cause the resources or resource groups *B* and *A* to run on the same node. Note that this implies that Pacemaker will first figure out on which node **A** should live on, and only then see if **B** can live there as well. The starting of **A** and **B** occurs in parallel. If there isn't a location where both **A** and **B** can run, then **A** will get its preference and **B** will remain stopped. If there isn't an available location for **A**, then **B** will also have nowhere to run.

A colocation constraint with a score of **-INFINITY** can also be set to force the two resources or resource groups to never run on the same node:

```
[root@nodeY ~]# pcs constraint colocation add B with A -INFINITY
```

**References**

**pcs**(8) and **crm_simulate**(8) man pages

*Pacemaker Explained* (Pacemaker 1.1 for Corosync 2.x and pcs) Chapter 6 available in the *pacemaker-doc* package or at
http://clusterlabs.org/doc/

*Clusters from Scratch* (Pacemaker 1.1 for Corosync 2.x and pcs) Chapter 6 available in the *pacemaker-doc* package or at
http://clusterlabs.org/doc/

Additional information may be available in the *High Availability Add-On Reference* for Red Hat Enterprise Linux 7, which can be found at
http://docs.redhat.com/

## Guided Exercise

# Configuring Location Restraints

In this lab, you will set resource contraint on resource groups in the cluster.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

## Outcome(s)

You should be able to modify and manage **location constraints** on resource groups in the cluster.

- Reset your **workstation** and all **node*Y*** systems.
- From your **workstation** system, run the command **lab resources solve**. Once completed, run the command **lab nfs-build setup**.
- Ensure the resource group **firstweb** is running on **nodeb**. If the resource group is not running on **nodeb**, move it there and remove the **move** restriction.

```
[root@nodea ~]# pcs resource move firstweb nodeb.private.example.com
[root@nodea ~]# pcs resource clear firstweb
```

- Ensure the resource group **nfs** is running on **nodec**. If the resource group is not running on **nodec**, move it there and remove the **move** restriction.

```
[root@nodea ~]# pcs resource move nfs nodec.private.example.com
[root@nodea ~]# pcs resource clear nfs
```

1. Configure a location constraint for the resource group **firstweb** for the system **nodea** with a **prefers** score of **200**, then verify if the cluster moves **firstweb** from **nodeb** to **nodea**.

   1.1. Add a location constraint score of **200** for the **firstweb** resource group for **nodea.private.example.com**.

```
[root@nodea ~]# pcs constraint location firstweb prefers \
        > nodea.private.example.com=200
```

   1.2. Verify where the **firstweb** resource group is running in the cluster.

```
[root@nodea ~]# pcs status
...
 Resource Group: firstweb
     webfs     (ocf::heartbeat:Filesystem): Started nodea.private.example.com
     webip     (ocf::heartbeat:IPaddr2):    Started nodea.private.example.com
     webserver (ocf::heartbeat:apache):     Started nodea.private.example.com
...
```

1.3. The resource group **firstweb** was migrated to **nodea** because the **prefers** score of **200** is higher than the default resource stickiness of **0**.

**2.** Change the default **resource-stickiness** to **500**, then add a location **prefers** score of **499** to **nodeb** for the **firstweb** resource group. Verify where the **firstweb** resource group is running.

2.1. Change the default **resource-stickiness** to **500**.

```
[root@nodea ~]# pcs resource defaults resource-stickiness=500
```

2.2. Add a location **prefers** score of **499** to **nodeb** for the **firstweb** resource group.

```
[root@nodea ~]# pcs constraint location firstweb prefers \
> nodeb.private.example.com=499
```

2.3. Verify where the **firstweb** resource group is running in the cluster.

```
[root@nodea ~]# pcs status
...
 Resource Group: firstweb
     webfs     (ocf::heartbeat:Filesystem): Started nodea.private.example.com
     webip     (ocf::heartbeat:IPaddr2):    Started nodea.private.example.com
     webserver (ocf::heartbeat:apache):     Started nodea.private.example.com
...
```

2.4. The resource group **firstweb** was not moved to the **nodeb** system despite the fact that it has a higher score. The cluster operates like this because the score of **nodeb** is below the configured default **resource-stickiness** score.

**3.** Set a location constraint for the **nfs** resource group to avoid running on **nodec** with a score of **INFINITY**. Observe the location management of the **nfs** resource group in the cluster.

3.1

```
[root@nodea ~]# pcs constraint location nfs avoids nodec.private.example.com
```

3.2. Observe the location management of the **nfs** resource group in the cluster.

```
[root@nodea ~]# pcs status
...
 Resource Group: nfs
     nfsshare (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
...
```

3.3. The **nfs** resource group was migrated away from **nodec** because an **avoids** score of **-INFINITY** prohibits the resource from running on **nodec**. The cluster migrated the **nfs** resource group to **nodeb**.

## Lab

# Controlling Complex Resource Groups

### Performance Checklist

In this lab, you will configure a highly available **NFS** service.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

### Outcome(s)

You should be able to configure a highly available, active-passive, **NFSv4** service on the cluster to provide a share to the **workstation** system. The service can migrate between **nodec** and **nodeb**. The **nodec** system is the preferred node for the service.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, and **nodec** systems.
- From your **workstation** system, run the command **lab nfs setup**.

```
[student@workstation ~]$ lab nfs setup
```

Configure a highly available active-passive **NFS** share according to the following requirements:

- The **NFSv4** share uses the iSCSI-provided shared storage **/dev/sda1** with 256 MiB as the storage back end.
- The resource group providing the **NFSv4** share is named **hanfs**.
- The resource group providing the **NFSv4** share is accessible with the public IP address **172.25.*X*.83**.
- The **Filesystem** resource is named **nfsfs**.
- The mount point used by the **Filesystem** resource agent for **/dev/sda1** is **/hanfsmount**.
- The **NFS** server resource is named **nfssvc** and uses **/hanfsmount/nfsinfo** as a shared information directory.
- The **exportfs** agent resource is named **nfsexport**. The **/hanfsmount** directory is exported as an **NFSv4** share with the **NFS** options **rw,sync,no_root_squash** as **NFS** root.
- The **IPaddr2** resource is named **nfsip**.
- The **NFS** share can be mounted from the **workstation** system and is writable.
- The resource group **hanfs** can migrate between **nodec** and **nodeb**. The **nodec** system is the preferred node and the resource group will fall back to the **nodec** node whenever it is available. The resource group must not migrate to **nodea**.

1. Prepare a **XFS**-formatted partition **/dev/sda1** of 256 MiB on the iSCSI block device if it does not already exist.
2. Create a highly available **NFS** service according to the previously listed requirements in an active-passive setup as resource group **hanfs** and allow access to the **NFSv4** share through the firewall on all nodes that host the **hanfs** resource group.
3. Validate the clustered **NFS** setup is accessible as an **NFSv4** share by mounting it on the newly created mount point **/mnt/hanfsshare** and creating a new empty file **/mnt/hanfsshare/myfile.txt**.
4. Configure the resource group **hanfs** to fail over only between **nodec** and **nodeb**. The resource group **hanfs** always migrates back to cluster node **nodec** if it is available.

## Solution

# Controlling Complex Resource Groups

### Performance Checklist

In this lab, you will configure a highly available **NFS** service.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

### Outcome(s)

You should be able to configure a highly available, active-passive, **NFSv4** service on the cluster to provide a share to the **workstation** system. The service can migrate between **nodec** and **nodeb**. The **nodec** system is the preferred node for the service.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, and **nodec** systems.
- From your **workstation** system, run the command **lab nfs setup**.

```
[student@workstation ~]$ lab nfs setup
```

Configure a highly available active-passive **NFS** share according to the following requirements:

- The **NFSv4** share uses the iSCSI-provided shared storage **/dev/sda1** with 256 MiB as the storage back end.
- The resource group providing the **NFSv4** share is named **hanfs**.
- The resource group providing the **NFSv4** share is accessible with the public IP address **172.25.*X*.83**.
- The **Filesystem** resource is named **nfsfs**.
- The mount point used by the **Filesystem** resource agent for **/dev/sda1** is **/hanfsmount**.
- The **NFS** server resource is named **nfssvc** and uses **/hanfsmount/nfsinfo** as a shared information directory.
- The **exportfs** agent resource is named **nfsexport**. The **/hanfsmount** directory is exported as an **NFSv4** share with the **NFS** options **rw,sync,no_root_squash** as **NFS** root.
- The **IPaddr2** resource is named **nfsip**.
- The **NFS** share can be mounted from the **workstation** system and is writable.
- The resource group **hanfs** can migrate between **nodec** and **nodeb**. The **nodec** system is the preferred node and the resource group will fall back to the **nodec** node whenever it is available. The resource group must not migrate to **nodea**.

1. Prepare a **XFS**-formatted partition **/dev/sda1** of 256 MiB on the iSCSI block device if it does not already exist.

   1.1. From **nodea**, create a 256 MiB primary partition on **/dev/sda**.

```
[root@nodea ~]# fdisk /dev/sda
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): <Enter>
First sector (270336-8388607, default 8192): <Enter>
Using default value 270336
Last sector, +sectors or +size{K,M,G} (270336-8388607, default 8388607): +256M
Partition 2 of type Linux and of size 256 MiB is set

Command (m for help): w
```

   1.2. Make sure that all nodes see the new partition by running the **partprobe** command.

```
[root@nodeY ~]# partprobe
```

   1.3. Format the **/dev/sda1** partion with the **XFS** file system even if it already exists.

```
[root@nodea ~]# mkfs -t xfs /dev/sda1
```

2. Create a highly available **NFS** service according to the previously listed requirements in an active-passive setup as resource group **hanfs** and allow access to the **NFSv4** share through the firewall on all nodes that host the **hanfs** resource group.

   2.1. Stop and disable nfs-lock. This will be managed by pacemaker.

```
[root@nodeY ~]#systemctl stop nfs-lock;systemctl disable nfs-lock
```

   2.2. Create the **Filesystem** resource named **nfsfs** using the **XFS**-formatted device **/dev/sda1** and the mount point directory **/hanfsmount** as part of the **hanfs** resource group.

```
[root@nodea ~]# pcs resource create nfsfs Filesystem device=/dev/sda1 directory=/
hanfsmount fstype=xfs --group hanfs
```

   2.3. Create the **nfsserver** resource named **nfssvc** as part of the **hanfs** group. The **nfs_shared_infodir** is set to **/hanfsmount/nfsinfo**.

```
[root@nodea ~]# pcs resource create nfssvc nfsserver nfs_shared_infodir=/
hanfsmount/nfsinfo --group hanfs
```

2.4. Create the **exportfs** resource named **nfsexport** to export the **/hanfsmount** directory as an **NFSv4** share to the **workstation** host with the **NFS** options **rw,sync,no_root_squash** as part of the **hanfs** resource group.

```
[root@nodea ~]# pcs resource create nfsexport exportfs clientspec="*"
 options=rw,sync,no_root_squash directory=/hanfsmount fsid=0 --group hanfs
```

2.5. Create the **IPaddr2** resource named **nfsip** with IP **172.25.*X*.83/24** as part of the **hanfs** resource group.

```
[root@nodea ~]# pcs resource create nfsip IPaddr2 ip=172.25.X.83 cidr_netmask=24
 --group hanfs
```

2.6. Verify the status of the resource group with **pcs status**. If there are any errors, run the command **pcs resource cleanup**, then re-check the status of the resource group.

```
[root@nodea ~]# pcs status
```

```
[root@nodea ~]# pcs resource cleanup
```

2.7. Allow access to the **NFS** server through the firewall on all cluster nodes if not already allowed.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=nfs
[root@nodeY ~]# firewall-cmd --reload
```

3. Validate the clustered **NFS** setup is accessible as an **NFSv4** share by mounting it on the newly created mount point **/mnt/hanfsshare** and creating a new empty file **/mnt/hanfsshare/myfile.txt**.

3.1. Create the mount point **/mnt/hanfsshare** on **workstation**.

```
[root@workstation ~]# mkdir /mnt/hanfsshare
```

3.2. Mount the clustered **NFSv4** share **172.25.*X*.83:/** as an **NFSv4** export on the **/mnt/hanfsshare** mount point on **workstation**.

```
[root@workstation ~]# mount 172.25.X.83:/ /mnt/hanfshare
```

3.3. Create the empty file **/mnt/hanfsshare/myfile.txt** on **workstation**.

```
[root@workstation ~]# touch /mnt/hanfsshare/myfile.txt
```

4. Configure the resource group **hanfs** to fail over only between **nodec** and **nodeb**. The resource group **hanfs** always migrates back to cluster node **nodec** if it is available.

4.1. Add a location-constraint score of **INFINITY** for the **hanfs** resource group for **nodec.private.example.com**.

```
[root@nodea ~]# pcs constraint location hanfs prefers nodec.private.example.com
```

4.2. Add a location constraint to prohibit the **hanfs** resource group from migrating to **nodea**.

```
[root@nodea ~]# pcs constraint location hanfs avoids nodea.private.example.com
```

4.3. Display the location constraints configured in the cluster.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
...
  Resource: hanfs
    Enabled on: nodec.private.example.com (INFINITY)
    Disabled on: nodea.private.example.com (-INFINITY)
...
```

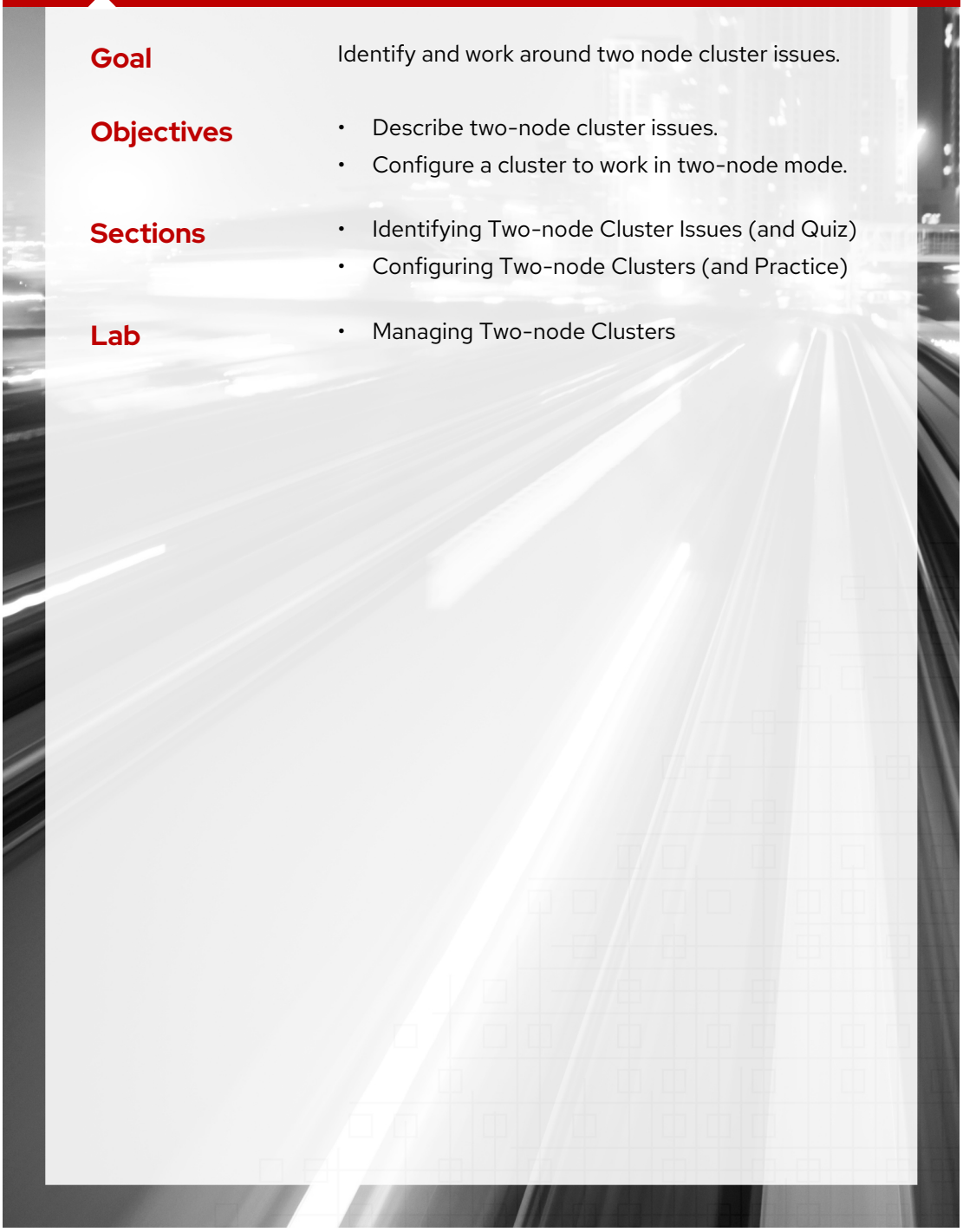
# Summary

In this chapter, you learned:

- What resource constraints are.
- How to view and remove constraints from a resource group.
- How to modify order and location constraints that are automatically set on resource group members.
- How to configure the resources required for a highly available **NFS** service.
- How to configure the required resource start order for a clustered service.

Chapter 7

# Managing Two Node Clusters



**Goal**

Identify and work around two node cluster issues.

**Objectives**

- Describe two-node cluster issues.
- Configure a cluster to work in two-node mode.

**Sections**

- Identifying Two-node Cluster Issues (and Quiz)
- Configuring Two-node Clusters (and Practice)

**Lab**

- Managing Two-node Clusters

# Identifying Two-node Cluster Issues

## Objectives

After completing this section, students should be able to describe two-node cluster issues.

## Two-node cluster issues

While two-node clusters may look good on paper, in practice there are a lot of extra failure scenarios that are not present with three or more node clusters.

If the choice for a three-node or larger cluster cannot be made, it is recommended to have Red Hat perform an architecture review of the intended two-node cluster.

### Possible issues

Two-node clusters have their own set of issues. The following table lists some of the most common ones, and workarounds to avoid these issues.

| Issue | Description and workaround |
|---|---|
| No room for node failure | In a default cluster setup, at least **`50% + 1`** of the nodes must be up for the cluster to be quorate. In a two-node cluster this would come down to two votes, meaning the entire cluster. By enabling the special **`two_node`** mode of **`votequorum`**, the number of expected votes will be brought down to one, as long as there are only two-nodes in the cluster. This allows one node to fail, with the other node remaining a quorate cluster all by itself. |
| Split-brain | A *split-brain* situation occurs when two halves of a cluster both think they are quorate, and start competing for shared resources.<br><br>If fencing is configured and tested, a split-brain scenario cannot occur, since both nodes will attempt to fence each other before recovering any resources. |

| Issue | Description and workaround |
|---|---|
| Fence death/fence racing | Since both nodes in a two-node cluster can maintain quorum by themselves, administrators can run into a phenomenon known as a *fence race*. A fence race is what happens when communication between the two-nodes is interrupted, but they can still fence each other, either because fabric-level fencing is used, or because the fencing devices are on a different network from the cluster communication.<br><br>If the fence device has serialized access, meaning that only one machine can talk to it a time, this causes no problems. One node will win the fence race, and the other node will be fenced (although if the communication problem is persistent, the node that was just fenced will fence the winning node when it comes back up, leading to a reboot-fence cycle).<br><br>In situations where there are multiple fencing devices (e.g., ILO cards), an undesirable outcome can occur: both nodes fencing each other at the same time. One way of combatting this is to set a delay on one of the fencing devices. |
| The cluster does not start until both nodes have started. | When the special **`two_node`** mode of **`corosync_votequorum`** is enabled, the **`wait_for_all`** mode will also be implicitly enabled, unless expressly disabled in the configuration file. The **`wait_for_all`** makes the cluster wait for all nodes to be up at the same time before starting any cluster resources.<br><br>If the cluster should be able to start with only one node, the **`wait_for_all`** option can be expressly disabled either at cluster creation time, or later in **`corosync.conf`**. |

**References**

**Can two-node RHEL 7 High Availability clusters using corosync, votequorum, and pacemaker be susceptible to fence loops?**
https://access.redhat.com/solutions/1294873

**Delaying Fencing in a Two Node Cluster to Prevent Fence Races or "Fence Death" Scenarios**
https://access.redhat.com/solutions/54829

**Why does my 2 node pacemaker cluster fail to obtain quorum when starting only one node?**
https://access.redhat.com/solutions/1488893

## Quiz

# Identifying Two-node Cluster Issues

Choose the correct answer to the following questions:

1. **Which of the following describes the effect of the `two_node` mode for `votequorum`?**
   a. **`two_node`** mode prevents fence races by delaying fencing for one node.
   b. **`two_node`** mode automatically reduces the number of expected votes to one for clusters with fewer than three nodes.
   c. **`two_node`** mode allows the cluster to start before all nodes have joined.
   d. **`two_node`** mode acts as a tie-breaker for split-brain situations by only allowing quorum for the half with the lowest node ID.

2. **Which of the following describes a fence race?**
   a. A fence race happens when power fencing completes before fabric fencing.
   b. A fence race happens when multiple nodes attempt to fence one other node.
   c. A fence race happens when two-nodes simultaneously attempt to fence each other.
   d. A fence race happens when fabric fencing completes before power fencing.

3. **What are the dangers of a fence race? Select all that apply.**
   a. Both nodes can turn each other off at the same time, disabling the entire cluster.
   b. A split-brain situation can occur.
   c. The "winner" of the fence race goes into **`m3d4l`** mode.
   d. Both nodes enter a reboot-then-fence cycle, disabling cluster operations.

## Solution

# Identifying Two-node Cluster Issues

Choose the correct answer to the following questions:

**1. Which of the following describes the effect of the `two_node` mode for `votequorum`?**

a. `two_node` mode prevents fence races by delaying fencing for one node.

b. **`two_node`** mode automatically reduces the number of expected votes to one for clusters with fewer than three nodes.

c. `two_node` mode allows the cluster to start before all nodes have joined.

d. `two_node` mode acts as a tie-breaker for split-brain situations by only allowing quorum for the half with the lowest node ID.

**2. Which of the following describes a fence race?**

a. A fence race happens when power fencing completes before fabric fencing.

b. A fence race happens when multiple nodes attempt to fence one other node.

c. A fence race happens when two-nodes simultaneously attempt to fence each other.

d. A fence race happens when fabric fencing completes before power fencing.

**3. What are the dangers of a fence race? Select all that apply.**

a. Both nodes can turn each other off at the same time, disabling the entire cluster.

b. A split-brain situation can occur.

c. The "winner" of the fence race goes into `m3d4l` mode.

d. Both nodes enter a reboot-then-fence cycle, disabling cluster operations.

# Configuring Two-node Clusters

## Objectives

After completing this section, students should be able to configure a cluster to work in two-node mode.

## Creating two-node clusters

When creating a new two-node cluster with **pcs**, the special **two_node** mode will automatically be enabled in **corosync.conf**. If extra nodes later join the cluster, this option will automatically be disabled again.

After creating a two-node cluster, the **quorum** block in **/etc/corosync/corosync.conf** will look like this:

```
quorum {
  provider: corosync_votequorum
  two_node: 1
}
```

### Disabling wait_for_all

Even though the **wait_for_all** option is not specified explicitly, it is turned on automatically by the **two_node** option.

To disable the **wait_for_all** option when creating a new cluster, the **--wait_for_all=0** option can be added, like this:

```
[root@nodea ~]# pcs cluster setup --start --enable \
> --name twonodecluster \
> --wait_for_all=0 \
> nodea.private.example.com \
> nodeb.private.example.com
```

To disable the **wait_for_all** option on an already existing cluster, the following procedure can be used:

1. Stop the running cluster.

```
[root@nodea ~]# pc cluster stop --all
```

2. Add a line **wait_for_all: 0** to the **quorum** block in **/etc/corosync/corosync.conf**:

```
quorum {
  provider: corosync_votequorum
  two_node: 1
  wait_for_all: 0
}
```

3. Sync **corosync.conf** to the other cluster node.

```
[root@nodea ~]# pcs cluster sync
```

4. Start the cluster on all nodes.

```
[root@nodea ~]# pc cluster start --all
```

# Configuring delayed fencing

To prevent fence races, one of the two-nodes should be configured for delayed fencing. This will help in keeping the cluster stable, but it will not prevent reboot-then-fence cycles if communications between the cluster nodes has been compromised because of an external influence like a broken switch.

With delayed fencing, one of the fencing devices is configured with a **delay=*N*** option, where ***N*** is a timeout in seconds. When the fence device is called, it will log the beginning of the fencing operation immediately, but then wait for ***N*** seconds before actually starting the fencing operation. This gives the node that would be fenced by this device time to fence the other node before fencing can begin, effectively saving the node for which the delayed fencing device is configured from being fenced, and making it "win" the fence race.

**Important**

Delayed fencing should only be used for a fence device targeting a single host. If a shared fence device is used, it should be split up into two separate fence devices, using the **pcmk_host_list** option.

## Creating a delayed fence device

The procedure for creating a fence device with delayed fencing is exactly the same as the procedure for creating a regular fence device, but with an added **delay=*N*** option. For example, to create a fence device called **fence_nodea_delayed**, targeting **nodea**, and with a delay of **10** seconds, the following command can be used:

```
[root@nodea ~]# pcs stonith create fence_nodea_delayed \
> fence_rht ipaddr="classroom.example.com" \
> port="nodea.private.example.com" \
> pcmk_host_list="nodea.private.example.com" \
> delay=10
```

If a fence device is created for the second node without a delay, this will prevent **nodea** from being fenced since it has a 10-second window to fence the second node. If the cluster communication network is still up, and **nodea** fails for another reason it will still be fenced, but with a 10-second delay.

## Updating an existing fence device for delayed fencing

An existing fence device can be updated for delayed fencing as well, using the **pcs stonith update** command. Ensure that the fence device is only used for one host, then add the **delay** option.

For example, to update the fence device **fence_mynodea** to use a five-second delay, the following command can be used:

```
[root@nodea ~]# pcs stonith update fence_mynodea delay=5
```

# Delayed fencing and fencing levels

Since fencing levels are attempted sequentially, and the fence devices inside a fencing level are executed in a serial fashion as well, only the first fence device in the first fencing level needs a delay.

While extra delays can be added, they will slow down the overall fencing process without providing additional protection against fence races. It can even increase the risk of a partially fenced machine, since it provides a bigger window of opportunity for the second node to put in some fencing actions.

**References**

**Delaying Fencing in a Two Node Cluster to Prevent Fence Races or "Fence Death" Scenarios**
https://access.redhat.com/solutions/54829

**Why does my 2 node pacemaker cluster fail to obtain quorum when starting only one node?**
https://access.redhat.com/solutions/1488893

## Guided Exercise

# Configuring a Two-node Cluster

In this lab, you will create a new two-node cluster.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, and **nodeb** |

## Outcome(s)

You should be able to configure a two-node cluster with delayed fencing.

- Reset your **workstation** system.
- Reset both your **nodea** and **nodeb** systems.

**1.** Prepare both your **nodea** and **nodeb** machines to become cluster nodes.

1.1. On both of your nodes, install the *fence-agents-all*, *fence-agents-rht*, and *pcs* packages.

```
[root@nodeY ~]# yum -y install pcs fence-agents-all fence-agents-rht
```

1.2. On both nodes, allow cluster traffic to pass through the firewall.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

1.3. On both nodes, enable and start the **pcsd** service.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

1.4. On both nodes, set the **hacluster** password to **redhat**.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
```

1.5. From your **nodea** machine, authenticate your **nodea.private.example.com** and **nodeb.private.example.com** machines for **pcs**.

```
[root@nodea ~]# pcs cluster auth -u hacluster -p redhat \
> nodea.private.example.com \
> nodeb.private.example.com
```

2. Create a new two-node cluster called **artoo** using your **nodea.private.example.com** and **nodeb.private.example.com** machines. Start and enable the cluster on both nodes.

   2.1. Create the cluster.

```
[root@nodea ~]# pcs cluster setup --start --enable \
> --name artoo \
> nodea.private.example.com \
> nodeb.private.example.com
```

   2.2. Verify the cluster is active.

```
[root@nodea ~]# pcs status
```

3. Add two fence devices to your cluster, one for each node, using the **fence_rht** fencing agent. The fencing daemon can be reached at **classroom.example.com**, and plug names should be the host names of your nodes on the **private.example.com** network.

   To prevent fence racing, a delay of five seconds should be configured so that your **nodea** machine will win the fence race in case of a split-brain scenario.

   3.1. Create a fence device for your **nodea** machine, using the **delay=5** option.

```
[root@nodea ~]# pcs stonith create fence_nodea \
> fence_rht ipaddr="classroom.example.com" \
> port="nodea.private.example.com" \
> pcmk_host_list="nodea.private.example.com" \
> delay=5
```

   3.2. Create a fence device for your **nodeb** machine.

```
[root@nodea ~]# pcs stonith create fence_nodeb \
> fence_rht ipaddr="classroom.example.com" \
> port="nodeb.private.example.com" \
> pcmk_host_list="nodeb.private.example.com"
```

## Lab

# Managing Two-node Clusters

### Performance Checklist

In this lab, you will configure a two-node cluster with delayed fencing.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, and **nodeb**. |

### Outcome(s)

You should be able to create a two-node cluster with delayed fencing.

- Reset both your **nodea** and **nodeb** machines.

You have been asked to create a new two-node cluster named **mewtwo** using your **nodea** and **nodeb** machines.

Fencing for this cluster should be handled using the fencing daemon running on **classroom.example.com** in combination with the **fence_rht** fencing agent. Plug names for this fence device correspond to host names on the **private.example.com** network. In order to prevent fence races, **nodeb** should have a 10-second delay when being fenced.

To speed up resource recovery when the cluster is started, the cluster should not wait for all nodes to join before beginning resource recovery by fencing.

1. Prepare both your **nodea** and **nodeb** machines to become cluster nodes.
2. Create a new two-node cluster called **mewtwo** using your **nodea.private.example.com** and **nodeb.private.example.com** machines. Start and enable the cluster on both nodes. Ensure that the cluster will not wait for the cluster to be completely formed when starting before initiating fencing actions.
3. Add two fence devices to your cluster, one for each node, using the **fence_rht** fencing agent. The fencing daemon can be reached at **classroom.example.com**, and plug names should be the host names of your nodes on the **private.example.com** network. There should be a 10-second delay before fencing **nodeb**.

## Solution

# Managing Two-node Clusters

### Performance Checklist

In this lab, you will configure a two-node cluster with delayed fencing.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, and **nodeb**. |

### Outcome(s)

You should be able to create a two-node cluster with delayed fencing.

- Reset both your **nodea** and **nodeb** machines.

You have been asked to create a new two-node cluster named **mewtwo** using your **nodea** and **nodeb** machines.

Fencing for this cluster should be handled using the fencing daemon running on **classroom.example.com** in combination with the **fence_rht** fencing agent. Plug names for this fence device correspond to host names on the **private.example.com** network. In order to prevent fence races, **nodeb** should have a 10-second delay when being fenced.

To speed up resource recovery when the cluster is started, the cluster should not wait for all nodes to join before beginning resource recovery by fencing.

1. Prepare both your **nodea** and **nodeb** machines to become cluster nodes.

   1.1. On both of your nodes, install the *fence-agents-all*, *fence-agents-rht*, and *pcs* packages.

```
[root@nodeY ~]# yum -y install pcs fence-agents-all fence-agents-rht
```

   1.2. On both nodes, allow cluster traffic to pass through the firewall.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

   1.3. On both nodes, enable and start the **pcsd** service.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

   1.4. On both nodes, set the **hacluster** password to **redhat**.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
```

1.5. From your **nodea** machine, authenticate your **nodea.private.example.com** and **nodeb.private.example.com** machines for **pcs**.

```
[root@nodea ~]# pcs cluster auth -u hacluster -p redhat \
> nodea.private.example.com \
> nodeb.private.example.com
```

2. Create a new two-node cluster called **mewtwo** using your **nodea.private.example.com** and **nodeb.private.example.com** machines. Start and enable the cluster on both nodes. Ensure that the cluster will not wait for the cluster to be completely formed when starting before initiating fencing actions.

2.1. Create the cluster.

```
[root@nodea ~]# pcs cluster setup --start --enable \
> --name mewtwo \
> --wait_for_all=0 \
> nodea.private.example.com \
> nodeb.private.example.com
```

2.2. Verify the cluster is active.

```
[root@nodea ~]# pcs status
```

3. Add two fence devices to your cluster, one for each node, using the **fence_rht** fencing agent. The fencing daemon can be reached at **classroom.example.com**, and plug names should be the host names of your nodes on the **private.example.com** network. There should be a 10-second delay before fencing **nodeb**.

3.1. Create a fence device for your **nodea** machine.

```
[root@nodea ~]# pcs stonith create fence_nodea \
> fence_rht ipaddr="classroom.example.com" \
> port="nodea.private.example.com" \
> pcmk_host_list="nodea.private.example.com"
```

3.2. Create a fence device for your **nodeb** machine, using a **10** second delay.

```
[root@nodea ~]# pcs stonith create fence_nodeb \
> fence_rht ipaddr="classroom.example.com" \
> port="nodeb.private.example.com" \
> pcmk_host_list="nodeb.private.example.com" \
> delay=10
```

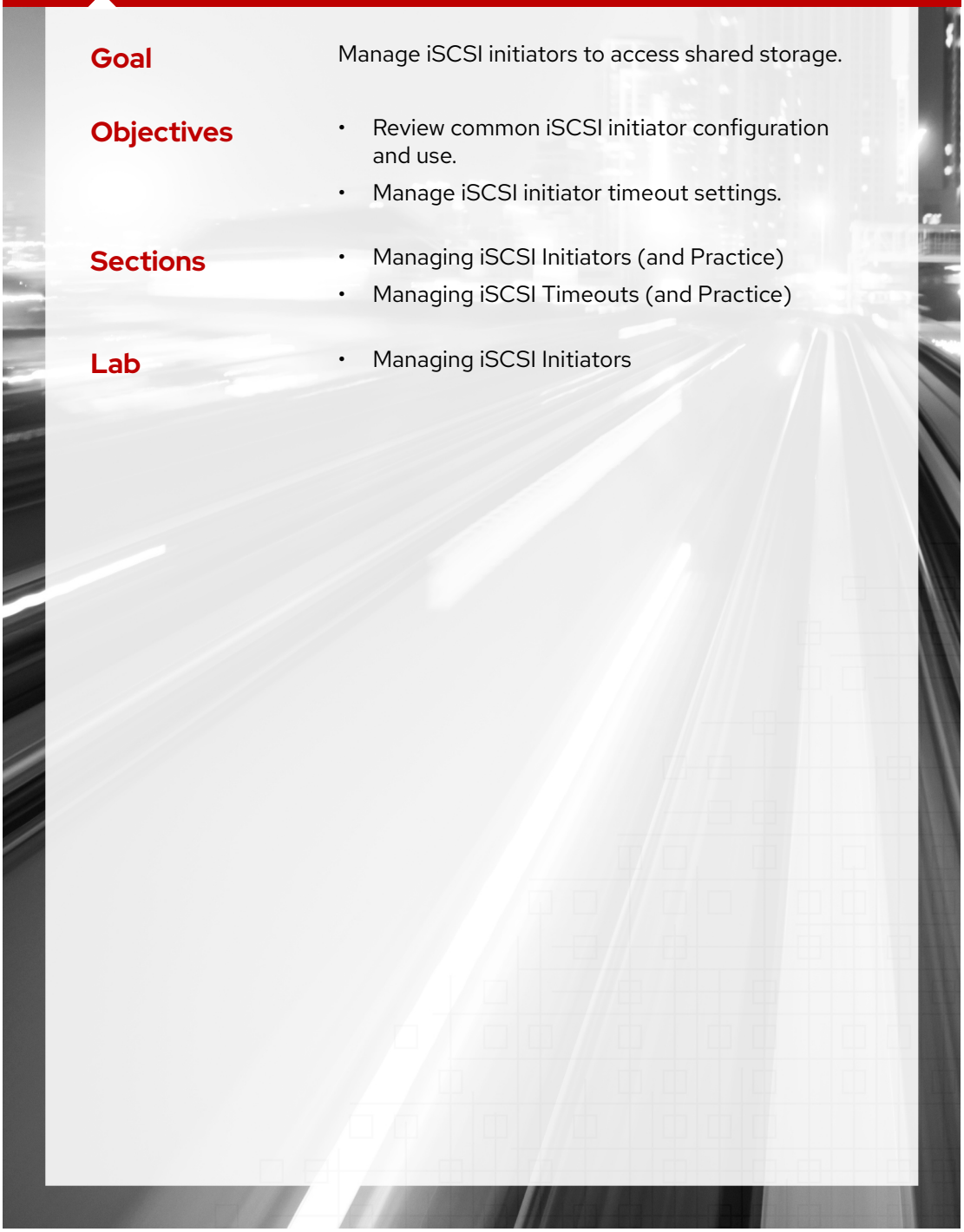
# Summary

In this chapter, you learned:

- Two-node clusters are susceptible to fence loops and fence races.
- The **`two_node: 1`** option in **`corosync.conf`** reduces the number of expected votes for a two-node cluster down to one.
- The **`two_node: 1`** option in **`corosync.conf`** automatically enables the **`wait_for_all: 1`** option.
- **`wait_for_all`** can be explicitly disabled if desired.
- Delayed fencing can stop a fence race.
- Delayed fencing is configured by adding a **`delay=N`** option to a fencing device.

Chapter 8

# Managing iSCSI Initiators



| | |
|---|---|
| **Goal** | Manage iSCSI initiators to access shared storage. |
| **Objectives** | • Review common iSCSI initiator configuration and use.<br>• Manage iSCSI initiator timeout settings. |
| **Sections** | • Managing iSCSI Initiators (and Practice)<br>• Managing iSCSI Timeouts (and Practice) |
| **Lab** | • Managing iSCSI Initiators |

# Managing iSCSI Initiators

## Objectives

After completing this section, students should be able to:

- Prepare a Red Hat Enterprise Linux client to access an iSCSI target service.
- Access and log into a remote iSCSI portal, target, and LUN.
- Practice iSCSI persistence through the use and deletion of persistent node records.

## iSCSI initiator introduction

In Red Hat Enterprise Linux an iSCSI initiator is typically implemented in software, and functions similar to a hardware iSCSI *host bus adapter* (HBA) to access targets from a remote storage server. Using a software-based iSCSI initiator requires connecting to an existing Ethernet network of sufficient bandwidth to carry the expected storage traffic.

iSCSI can also be implemented using a hardware initiator that includes the required protocols in a dedicated host bus adapter. iSCSI HBAs and TCP offload engines (TOE), which include the TCP network stack on an Ethernet NIC, move the processing of iSCSI or TCP overhead and Ethernet interrupts to hardware, easing the load on system CPUs.

Configuring an iSCSI client initiator requires installing the *iscsi-initiator-utils* package, which includes the **iscsi** and **iscsid** services and the **/etc/iscsi/iscsid.conf** and **/etc/iscsi/initiatorname.iscsi** configuration files.

As an iSCSI node, the client requires a unique *iSCSI Qualified Name* (IQN). The default **/etc/iscsi/initiatorname.iscsi** file contains a generated IQN using Red Hat's domain. Administrators typically reset the IQN to their own domain and an appropriate client system string.

The **/etc/iscsi/iscsid.conf** file contains default settings for node records created during new target discovery. Settings include iSCSI timeouts, retry parameters, and authentication usernames and passwords. Changing this file requires restarting the **iscsi** service.

```
[root@desktopX ~]# systemctl restart iscsi
```

To be able to discover targets, install the *iscsi-initiator-utils* package, then enable and start the **iscsi** service. Targets must be discovered before device connection and use. The discovery process stores target node information and settings in the **/var/lib/iscsi/nodes** directory, using defaults from **/etc/iscsi/iscsid.conf**. Since the same target can exist on multiple portals, node records are stored for each portal. Perform discovery with the following command:

```
[root@desktopX ~]# iscsiadm -m discovery -t sendtargets -p target_server[:port]
172.25.X.11:3260,1 iqn.2014-06.com.example:serverX
```

In discovery mode, the **sendtargets** request returns only targets with access configured for this initiator. The port number can be omitted when the target server is configured on default port 3260. Upon discovery, a node record is written to **/var/lib/iscsi/nodes** and used for subsequent logins. To use the listed target, log in using the following command:

```
[root@desktopX ~]# iscsiadm -m node -T iqn.2014-06.com.example:serverX [-
p target_server[:port]] -l
```

Specifying the portal is optional. If the target exists on multiple portals (e.g., in a multipathed, redundant server configuration), performing a login without specifying a portal will connect to every portal node that accepts this target name.

Once discovered, obtain information about targets with the **iscsiadm** command. Use the option **-P *N*** to set the command detail level, with **0** specifying the least verbose output.

- **iscsiadm -m discovery [-P 0|1]**: Show information about discovered targets.
- **iscsiadm -m node [-P 0|1]**: Show information about known targets.
- **iscsiadm -m session [-P 0|1|2|3]**: Show information about active sessions.

To discontinue using a target, use **iscsiadm** to log out temporarily. By design, node records remain after logout and are used to automatically log into targets upon system reboot or **iscsi** service restart. Log out of a target using the following command (notice the similarity to login):

```
[root@desktopX ~]# iscsiadm -m node -T iqn.2012-04.com.example:example [-
p target_server[:port]] -u
```

If a portal is not specified, the target logs out of all relevant portals. To log into the target again, repeating discovery is not necessary since the node records already exist. Permanently logging out of a target requires deleting the node records so that manual or automatic login cannot reoccur without performing another discovery. Not specifying a portal removes the target node records for all relevant portals. Delete the node record permanently by using the following command (notice again the command similarity):

```
[root@desktopX ~]# iscsiadm -m node -T iqn.2012-04.com.example:example [-
p target_server[:port]] -o delete
```

**References**

**iscsiadm**(8), **iscsid**(8) man pages

**Open-iSCSI**

http://www.open-iscsi.org

## Guided Exercise

# Accessing iSCSI Storage

In this lab, you will discover targets on an iSCSI target portal, then practice manual and automatic login and logout of an iSCSI target.

| Resources | |
|---|---|
| **Files** | /etc/iscsi/initiatorname.iscsi<br>/var/lib/iscsi/nodes/* |
| **Machines** | **workstation**<br>**nodea** |

## Outcomes

You should be able to discover and connect to an access-controlled remote iSCSI target and LUN.

Log into **nodea** and switch user to **root**.

Run **lab iscsi setup** on **workstation**.

The target *iqn.2015-06.com.example:cluster* has been configured on **workstation** and is available through both portal addresses 192.168.1.9 and 192.168.2.9. Configure the iSCSI initiator on **nodea** to connect to the target.

1. Prepare the client system to become an iSCSI initiator node by installing the initiator utilities, setting the unique iSCSI client name, and starting the iSCSI client service.

   1.1. Install the *iscsi-initiator-utils* RPM, if not already installed.

```
[root@nodea ~]# yum install -y iscsi-initiator-utils
```

   1.2. Create a unique IQN for the client initiator by modifying the **InitiatorName** setting in **/etc/iscsi/initiatorname.iscsi**. Use the client system name as the optional string after the colon.

```
[root@nodea ~]# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2015-06.com.example:nodea
```

   1.3. Enable and start the **iscsi** client service.

```
[root@nodea ~]# systemctl enable iscsi; systemctl start iscsi
```

2. Discover and log into the configured target from the iSCSI target server.

   2.1. Discover the configured iSCSI target(s) provided by the iSCSI target server portal.

```
[root@nodea ~]# iscsiadm -m discovery -t st -p 192.168.1.9
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster
```

2.2. Log into the presented iSCSI target.

```
[root@nodea ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p 192.168.1.9
 -l
```

2.3. Identify the newly available block device created by the iSCSI target login.

```
[root@nodea ~]# lsblk
[root@nodea ~]# tail /var/log/messages
```

**3.** Browse the connection information about the target portal, connection, and parameters used by the connected device. Locate the node record.

3.1. List the targets recognized by the **`iscsi`** service.

```
[root@nodea ~]# iscsiadm -m session -P 3
iSCSI Transport Class version 2.0-870
version 6.2.0.873-28
Target: iqn.2015-06.com.example:cluster (non-flash)
 Current Portal: 192.168.1.9:3260,1
 Persistent Portal: 192.168.1.9:3260,1
  **********
  Interface:
  **********
  Iface Name: default
  Iface Transport: tcp
  Iface Initiatorname: iqn.2015-06.com.example:nodea
  Iface IPaddress: 192.168.1.10
  Iface HWaddress: <empty>
  Iface Netdev: <empty>
  SID: 2
  iSCSI Connection State: LOGGED IN
  iSCSI Session State: LOGGED_IN
  Internal iscsid Session State: NO CHANGE
  *********
  Timeouts:
  *********
  Recovery Timeout: 120
  Target Reset Timeout: 30
  LUN Reset Timeout: 30
  Abort Timeout: 15
  *****
  CHAP:
  *****
  username: <empty>
  password: ********
  username_in: <empty>
  password_in: ********
  ************************
```

```
  Negotiated iSCSI params:
  ************************
  HeaderDigest: None
  DataDigest: None
  MaxRecvDataSegmentLength: 262144
  MaxXmitDataSegmentLength: 262144
  FirstBurstLength: 65536
  MaxBurstLength: 262144
  ImmediateData: Yes
  InitialR2T: Yes
  MaxOutstandingR2T: 1
  ************************
  Attached SCSI devices:
  ************************
  Host Number: 3 State: running
  scsi3 Channel 00 Id 0 Lun: 0
   Attached scsi disk sda  State: running
```

3.2. Change directory to the location of the iSCSI node records for the remainder of this exercise. Locate the persistent node record for the new iSCSI target.

```
[root@nodea ~]# cd /var/lib/iscsi/nodes
[root@nodea nodes]$ ls -lR
```

3.3. View the connection parameters defaults in an iSCSI node record.

```
[root@nodea nodes]# less iqn.2015-06.com.example:cluster/192.168.1.9,3260,1/
default
```

**4.** Connect to and disconnect from the target, using both manual commands and methods that utilize the node record(s) to connect automatically.

4.1. Disconnect the iSCSI block device by logging out of the iSCSI target. Confirm that the corresponding iSCSI block device has disappeared.

**Important**

In this exercise, the block device was not formatted or otherwise used. If the device is in use, unmount it properly before continuing.

```
[root@nodea nodes]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p
 192.168.1.9 -u
[root@nodea nodes]$ lsblk
```

4.2. Confirm that the node record for the disconnected iSCSI target still exists.

```
[root@nodea nodes]# ls -lR
```

4.3. Restart the **iscsi** client service. Confirm that the iSCSI block device returns. Since the **iscsi** client performs logins for all node records found, you should see an additional iSCSI block device returned.

```
[root@nodea nodes]# systemctl restart iscsi
[root@nodea nodes]# lsblk
```

4.4. Disconnect the iSCSI block devices again by logging out of the iSCSI targets. Additionally, delete the node records using the proper command. Confirm that the iSCSI block devices disappear.

```
[root@nodea nodes]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p
 192.168.1.9 -u
[root@nodea nodes]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p
 192.168.1.9 -o delete
[root@nodea nodes]# lsblk
```

4.5. Confirm that the node records for the disconnected iSCSI targets no longer exists.

```
[root@nodea nodes]# ls -lR
```

4.6. Restart the **iscsi** client service. Confirm that the iSCSI block devices does not return since there are no node records to trigger the target login.

```
[root@nodea nodes]# systemctl restart iscsi
[root@nodea nodes]# lsblk
```

**5.** Rediscover the target and confirm that the discovery step alone creates the node record(s).

5.1. Rediscover the configured iSCSI targets provided by a specific iSCSI target server portal, but do *not* log into the target.

```
[root@nodea nodes]# iscsiadm -m discovery -t st -p 192.168.1.9
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster
```

5.2. Confirm that the node records for the iSCSI targets are created by discovery, even before login has occurred.

```
[root@nodea nodes]# ls -R
```

5.3. Clean up by deleting the node records again. Confirm that the node records no longer exists. When finished, return to your home directory.

```
[root@nodea nodes]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p
 192.168.1.9 -o delete
[root@nodea nodes]# ls -lR
[root@nodea nodes]# cd ~
```

# Managing iSCSI Timeouts

## Objectives

After completing this section, students should be able to:

- Tune NOP-Out interval and timeout settings for improved network monitoring.
- Manage recovery timeout setting on iSCSI initiators.

## NOP-Out interval and timeout

Communication between iSCSI initiators and targets are sent in iSCSI protocol data units (PDUs). Request PDUs are sent by initiators, while response PDUs are sent by targets.

The **Opcode** field in the iSCSI PDU header indicates the iSCSI PDU type. There are two categories of opcodes: initiator opcodes and target opcodes. Initiator opcodes are used by initiators in request PDUs to targets, while target opcodes are used by targets in response PDUs to initiators.

One of the initiator opcodes is *NOP-Out*. Its target opcode counterpart is the *NOP-In*. The NOP-Out opcode can be sent by initiators to targets as a "ping" request to verify that a connection or session is active and operational and that its components are operational. Upon receiving a NOP-Out PDU from an initiator, a target responds with a NOP-In PDU. If a NOP-In response is not received by the initiator within a certain amount of time, the running commands will be failed and requeued for a later retry.

By default, in RHEL 7, initiators send out NOP-Out requests every five seconds. Once sent, NOP-Out requests will time out in five seconds. Depending on the iSCSI implementation or the performance and utilization of the SAN, it may be desirable to tune these timeout settings.

In **/etc/iscsi/iscsid.conf**, NOP-Out request interval can be specified with the following entry:

```
node.conn[0].timeo.noop_out_interval = [interval value]
```

NOP-Out request timeout can be customized by editing the following entry in the same file.

```
node.conn[0].timeo.noop_out_timeout = [timeout value]
```

Once changes to **/etc/iscsi/iscsid.conf** are made, the **iscsi** service must be restarted for the changes to take effect.

```
[root@nodey ~]# systemctl restart iscsi
```

NOP-Out interval and timeout setting changes will only be applied to future target interactions. iSCSI targets that are already discovered or logged into will still retain the old settings. To effect the new settings on these targets, use the **update** operator to the **iscsiadm** command.

```
iscsiadm -m node -T TARGET_NAME -p PORTAL_ADDRESS -o update -n
 node.conn[0].timeo.noop_out_interval -v INTERVAL_VALUE.
```

```
iscsiadm -m node -T TARGET_NAME -p PORTAL_ADDRESS -o update -n
 node.conn[0].timeo.noop_out_timeout -v TIMEOUT_VALUE.
```

## Session replacement timeout

Once NOP-Out timeout has expired, running commands will be failed immediately. However, if the SCSI Error Handler is active, running commands are not failed immediately. Instead, when NOP-Out request timeout is reached, the iSCSI layer will wait for the disrupted session or connection to re-establish before failing the running commands. The wait time is controlled by the **replacement_timeout** setting.

In RHEL 7, **replacement_timeout** is set by default to 120 seconds. It controls how long the iSCSI layer will wait before failing pending SCSI commands up to a higher level like multipath or to an application if multipath is not being used. When multipath is being used, it is desirable to lower the **replacement_timeout** value. By lowering **replacement_timeout** from 15 to 20 seconds, during error conditions, pending I/O commands will be promptly directed to a new physical path while the iSCSI layer works on session recovery.

Changes made to the **replacement_timeout** parameter will not affect iSCSI targets which are already discovered or logged into. To apply a new **replacement_timeout** setting on these targets, use the **update** operator to the **iscsiadm** command.

```
iscsiadm -m node -T TARGET_NAME -p PORTAL_ADDRESS -o update -n
 node.session.timeo.replacement_timeout -v TIMEOUT_VALUE.
```

To customize the **replacement_timeout** value for future discovered targets, edit the following entry in **/etc/iscsi/iscsid.conf** and then restarting the **iscsi** service.

```
node.session.timeo.replacement_timeout = [timeout value]
```

**Important**

Optimal value for **replacement_timeout** is dependent on multiple factors such as network performance, target specifications, and system workload. Therefore, all changes to **replacement_timeout** should be thoroughly tested before being applied to critical systems.

**References**

**RFC 3720 - Internet Small Computer Systems Interface (iSCSI)**
http://www.ietf.org/rfc/rfc3720.txt

**iscsiadm**(8) man pages

## Guided Exercise

# Modifying iSCSI Timeouts

In this lab, you will modify and observe the effects of modifying iSCSI initiator timeout settings.

| Resources | |
|---|---|
| **Files** | **/mnt/iscsi**<br>**/mnt/iscsi/file1**<br>**/mnt/iscsi/file2** |
| **Machines** | **workstation**<br>**nodea** |

## Outcome(s)

You should be able to modify iSCSI initiator timeout configuration so that iSCSI errors are quickly reported.

Reset **nodea** and then log into **nodea** and switch user to **root**.

Reset **workstation**, then log into **workstation** and run **lab iscsi setup** and **lab iscsi-timeout setup**.

The **nodea** host has been logged into the target *iqn.2015-06.com.example:cluster* on **workstation**, which is available through both portal addresses, 192.168.1.9 and 192.168.2.9. Create and mount an XFS file system on the target and then mount it at **/mnt/iscsi**. Observe the impact that lowering the **replacement_timeout** parameter has on I/O operations on this iSCSI device when network connectivity is disrupted.

1. Create a partition on the **/dev/sda** device and mount it as an XFS file system at **/mnt/iscsi**.

   1.1. Create a primary partition on **/dev/sda** of type **83**.

```
[root@nodea ~]# fdisk /dev/sda
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): Enter
First sector (8192-204799, default 8192): Enter
Last sector, +sectors or +size{K,M,G} (8192-204799, default 204799): Enter
```

```
Using default value 204799
Partition 1 of type Linux and of size 96 MiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

1.2. Run the **partprobe** command to update the kernel's view of the partition table on all devices.

```
[root@nodea ~]# partprobe
```

1.3. Create an XFS file system on **/dev/sda1**.

```
[root@nodea ~]# mkfs.xfs /dev/sda1
```

1.4. Mount **/dev/sda1** at **/mnt/iscsi**.

```
[root@nodea ~]# mkdir /mnt/iscsi; mount -t xfs -o rw,_netdev /dev/sda1 /mnt/iscsi
```

**2.** Disable network access to the iSCSI target server portals.

```
[root@nodea ~]# nmcli device disconnect eth2; nmcli device disconnect eth3
Device 'eth2' successfully disconnected.
Device 'eth3' successfully disconnected.
```

**3.** Attempt to create the file **/mnt/iscsi/file1** and observe the time it takes before an error is reported.

```
[root@nodea ~]# time touch /mnt/iscsi/file1
touch: cannot touch ‘/mnt/iscsi/file1’: No space left on device

real 2m1.016s
user 0m0.000s
sys 0m0.005s
```

**4.** Re-enable network access to the iSCSI target server portal.

```
[root@nodea ~]# nmcli device connect eth2; nmcli device connect eth3
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/4)
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/5)
```

**5.** Change the **replacement_timeout** setting to five seconds.

5.1. Update the value of the **node.session.timeo.replacement_timeout** parameter for the **iqn.2015-06.com.example:cluster** target on both portals.

```
[root@nodea ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p \
> 192.168.1.9 -o update -n node.session.timeo.replacement_timeout -v 5
[root@nodea ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p \
> 192.168.2.9 -o update -n node.session.timeo.replacement_timeout -v 5
```

5.2. Unmount **/dev/sda1** from **/mnt/iscsi**.

```
[root@nodea ~]# umount /mnt/iscsi
```

5.3. Log out and log into the portals again for the timeout change to take effect.

```
[root@nodea ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p \
> 192.168.1.9 -u
Logging out of session [sid: 1, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260]
Logout of [sid: 1, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] successful.
[root@nodea ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p \
> 192.168.2.9 -u
Logging out of session [sid: 2, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260]
Logout of [sid: 2, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] successful.
[root@nodea ~]#iscsiadm -m node -T iqn.2015-06.com.example:cluster -p \
> 192.168.1.9 -l
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] successful.
[root@nodea ~]#iscsiadm -m node -T iqn.2015-06.com.example:cluster -p \
> 192.168.2.9 -l
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] successful.
```

5.4. Remount **/dev/sda1** at **/mnt/iscsi**.

```
[root@nodea ~]# mount -t xfs -o rw,_netdev /dev/sda1 /mnt/iscsi
```

**6.** Disable network access to the iSCSI target server portal.

```
[root@nodea ~]# ifdown eth2; ifdown eth3
Device 'eth2' successfully disconnected.
Device 'eth3' successfully disconnected.
```

**7.** Attempt to create the file **/mnt/iscsi/file2** and observe the time it takes before an error is reported. What effect did the timeout setting change have?

```
[root@nodea ~]# time touch /mnt/iscsi/file2
touch: cannot touch '/mnt/iscsi/file2': No space left on device

real 0m13.288s
user 0m0.003s
sys 0m0.063s
```

8. Re-enable network access to the iSCSI target server portal.

```
[root@nodea ~]# ifup eth2; ifup eth3
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/4)
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/5)
```

## Lab

# Managing iSCSI Initiators

### Performance Checklist

In this lab, you will manage timeout settings that will be used for iSCSI initiators sessions to iSCSI targets.

| Resources | |
|---|---|
| **Files** | /etc/iscsi/iscsid.conf |
| **Machines** | **workstation**<br>**nodea** |

### Outcome(s)

You should be able to configure timeout settings for iSCSI initiators and discover and connect to remote iSCSI targets.

Reset **nodea** and then log into **nodea** and switch user to **root**.

Reset **workstation** and then log into **workstation** and run **lab iscsi setup**

The target *iqn.2015-06.com.example:cluster* has been configured on **workstation** and is available through both portal addresses, *192.168.1.9* and *192.168.2.9*. Configure the iSCSI initiator on **nodea** with an IQN of **iqn.2015-06.com.example:nodea** and change the **replacement_timeout** value on **nodea** from the default to 5 seconds.

Once the configuration is complete, log in the initiator, **nodea**, to the target on **workstation**,then execute **lab iscsi-timeout grade** as **root** on **workstation** to verify your work.

1. Prepare the client system to become an iSCSI initiator node.
2. Configure the iSCSI initiator so that the **replacement_timeout** setting is **5** seconds.
3. Discover and log into the configured target from the iSCSI target server.
4. Verify that the configured recovery timeout setting is used by the current session.

## Solution

# Managing iSCSI Initiators

## Performance Checklist

In this lab, you will manage timeout settings that will be used for iSCSI initiators sessions to iSCSI targets.

| Resources | |
|---|---|
| **Files** | /etc/iscsi/iscsid.conf |
| **Machines** | **workstation**<br>**nodea** |

## Outcome(s)

You should be able to configure timeout settings for iSCSI initiators and discover and connect to remote iSCSI targets.

Reset **nodea** and then log into **nodea** and switch user to **root**.

Reset **workstation** and then log into **workstation** and run **lab iscsi setup**

The target *iqn.2015-06.com.example:cluster* has been configured on **workstation** and is available through both portal addresses, *192.168.1.9* and *192.168.2.9*. Configure the iSCSI initiator on **nodea** with an IQN of **iqn.2015-06.com.example:nodea** and change the **replacement_timeout** value on **nodea** from the default to 5 seconds.

Once the configuration is complete, log in the initiator, **nodea**, to the target on **workstation**,then execute **lab iscsi-timeout grade** as **root** on **workstation** to verify your work.

1. Prepare the client system to become an iSCSI initiator node.

   1.1. Install the *iscsi-initiator-utils* RPM, if not already installed.

```
[root@nodea ~]# yum install -y iscsi-initiator-utils
```

   1.2. Create a unique IQN for the client initiator by modifying the **InitiatorName** setting in **/etc/iscsi/initiatorname.iscsi**. Use the client system name as the optional string after the colon.

```
[root@nodea ~]# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2015-06.com.example:nodea
```

   1.3. Enable and start the **iscsi** client service.

```
[root@nodea ~]# systemctl enable iscsi; systemctl start iscsi
```

2. Configure the iSCSI initiator so that the **replacement_timeout** setting is **5** seconds.

   2.1. Edit the **node.session.timeo.replacement_timeout** setting in **/etc/iscsi/iscsid.conf** so it resembles the following.

```
node.session.timeo.replacement_timeout = 5
```

   2.2. Restart the **iscsi** client service for the timeout setting to take effect.

```
[root@nodea ~]# systemctl restart iscsi
```

3. Discover and log into the configured target from the iSCSI target server.

   3.1. Discover the configured iSCSI target(s) provided by the iSCSI target server portal.

```
[root@nodea ~]# iscsiadm -m discovery -t st -p 192.168.1.9
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster
```

   3.2. Log into the presented iSCSI target.

```
[root@nodea ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p 192.168.1.9
 -l
```

   3.3. Identify the newly available block device created by the iSCSI target login.

```
[root@nodea ~]# lsblk
```

4. Verify that the configured recovery timeout setting is used by the current session.

   4.1. List the targets recognized by the **iscsi** service.

```
[root@nodea ~]# iscsiadm -m session -P 3
iSCSI Transport Class version 2.0-870
version 6.2.0.873-28
Target: iqn.2015-06.com.example:cluster (non-flash)
 Current Portal: 192.168.1.9:3260,1
 Persistent Portal: 192.168.1.9:3260,1
  **********
  Interface:
  **********
  Iface Name: default
  Iface Transport: tcp
  Iface Initiatorname: iqn.2015-06.com.example:nodea
  Iface IPaddress: 192.168.1.10
  Iface HWaddress: <empty>
  Iface Netdev: <empty>
  SID: 2
  iSCSI Connection State: LOGGED IN
  iSCSI Session State: LOGGED_IN
  Internal iscsid Session State: NO CHANGE
  *********
  Timeouts:
```

```
*********
Recovery Timeout: 5
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 15
*****
CHAP:
*****
username: <empty>
password: ********
username_in: <empty>
password_in: ********
************************
Negotiated iSCSI params:
************************
HeaderDigest: None
DataDigest: None
MaxRecvDataSegmentLength: 262144
MaxXmitDataSegmentLength: 262144
FirstBurstLength: 65536
MaxBurstLength: 262144
ImmediateData: Yes
InitialR2T: Yes
MaxOutstandingR2T: 1
************************
Attached SCSI devices:
************************
Host Number: 3 State: running
scsi3 Channel 00 Id 0 Lun: 0
 Attached scsi disk sda  State: running
```

# Summary

In this chapter, you learned:

- How to configure an iSCSI initiator to connect to remote iSCSI targets.
- The **`iscsiadm`** can be used to manage iSCSI target discovery, logins, and logouts, as well as session properties.
- NOP-Out interval and timeout can be tuned to adjust iSCSI response times to network issues.
- The **`recovery_timeout`** setting should be decreased when multipath is used.

Chapter 9

# Accessing Storage Devices Redundantly

**Goal**

Configure redundant storage access.

**Objectives**

- Describe multipathing and multipathing terminology.
- Configuring redundant storage access with dm-multipath.
- Testing multipath configurations.

**Sections**

- Multipathing Concepts (and Quiz)
- Configuring Redundant Storage Access (and Practice)
- Testing Redundant Storage (and Practice)

**Lab**

- Accessing Storage Devices Redundantly

# Multipathing Concepts

## Objectives

After completing this section, students should be able to describe multipathing and multipathing terminology.

## What is multipathing?

Multipathing allows the combination of multiple physical connections between a server and a storage array into one virtual device. This can be done to provide a more resilient connection to your storage (a path going down will not hamper connectivity), or to aggregate storage bandwidth for improved performance.

As an example, the server in the following diagram has two HBAs, each connected to a separate Fibre Channel switch, which in turn are connected to separate controllers on the storage array.

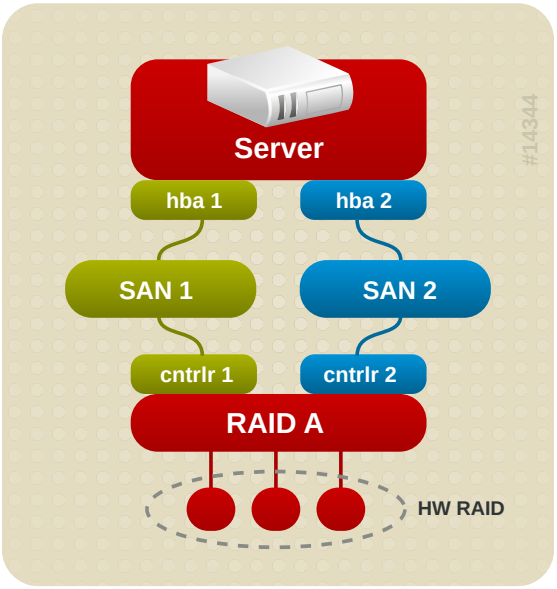



**Figure 9.1: An example multipath setup**

Red Hat Enterprise Linux 7 supports multipathing using the **`dm-multipath`** subsystem. This uses the kernel *device mapper* system to generate virtual devices, managed by the multipathd daemon and the **`multipath`** command-line tool.

The necessary binaries, daemons, and kernel modules are available in the *device-mapper-multipath* package.

Once *device-mapper-multipath* is installed, configured, and started, device nodes for multipathed devices will be created under two different locations.

For administrative purposes, multipath devices are created under **`/dev/mapper`**. They can be named **`mpathN[pM]`** if *user-friendly* names are chosen, or they can be named after the *World Wide ID* (WWID) of the storage device. An administrator can also set custom names

for multipathed devices. These custom names are established using the **alias** option in the **multipaths** section of the multipath configuration file.

Multipath devices are also created under **/dev** in the form of **/dev/dm-*N*** to match those created under **/dev/mapper**. These devices are strictly for the system's internal use and should therefore never be used directly for administrative purposes.

**Important**

Multipathing provides protection against a storage access path going down. If the storage itself becomes unavailable, access to the storage will be lost.

To create a multipath device, different paths will be combined into groups, based on settings in the **/etc/multipath.conf** configuration file. Typically only one group will be active at a time, but a group can consist of multiple paths. When a group fails, the multipath daemon will switch storage traffic to a different group.

**References**

Red Hat Enterprise Linux 7 DM Multipath Guide

- Section 1.2: Overview of DM-Multipath

## Quiz

# Multipathing Concepts

Choose the correct answer to the following questions:

1. **Which of the following can be benefits of implementing multipathing storage? (Choose two.)**
   a. Redundancy in storage access
   b. Increased storage performance
   c. Storage redundancy
   d. RAID

2. **What package provides the multipathing capabilities for Red Hat Enterprise Linux 7?**
   a. *dm-multipath*
   b. *dm-multipathd*
   c. *device-mapper-multipath*
   d. *multipathd*

3. **When the `user_friendly_names` configuration option is set to `yes` in the multipath configuration file, in which directory are device nodes for multipathed storage created?**
   a. **`/dev/`**
   b. **`/dev/multipath/`**
   c. **`/dev/mapper/`**
   d. **`/dev/dm-multipath/`**

## Solution

# Multipathing Concepts

Choose the correct answer to the following questions:

1. **Which of the following can be benefits of implementing multipathing storage? (Choose two.)**
   a. Redundancy in storage access
   b. Increased storage performance
   c. Storage redundancy
   d. RAID

2. **What package provides the multipathing capabilities for Red Hat Enterprise Linux 7?**
   a. *dm-multipath*
   b. *dm-multipathd*
   c. *device-mapper-multipath*
   d. *multipathd*

3. **When the `user_friendly_names` configuration option is set to yes in the multipath configuration file, in which directory are device nodes for multipathed storage created?**
   a. `/dev/`
   b. `/dev/multipath/`
   c. **`/dev/mapper/`**
   d. `/dev/dm-multipath/`

# Configuring Redundant Storage Access

## Objectives

After completing this section, students should be able to configure a multipath device for redundant storage access.

## Configuring multipathing

To configure multipathing, first make sure that the *device-mapper-multipath* package is installed.

```
[root@nodeY ~]# yum -y install device-mapper-multipath
```

Once the *device-mapper-multipath* package is installed, a configuration file must be created for the multipath daemon, **`/etc/multipath.conf`**. The easiest way to create this file is to use the **`mpathconf`** utility.

If there already is a file called **`/etc/multipath.conf`**, the **`mpathconf`** command will edit that file. If no such file exists, **`mpathconf`** will copy the default configuration from **`/usr/share/doc/device-mapper-multipath-*/multipath.conf`**. If that file does not exist, **`mpathconf`** will create a new configuration file from scratch.

To create a default configuration, and then start and enable the **`multipathd`** daemon, use the following command:

```
[root@nodeY ~]# mpathconf --enable --with_multipathd y --with_chkconfig y
```

**Note**

In the default configuration file created by **`mpathconf`**, *user-friendly names* are enabled with the **`user_friendly_names`** option. User friendly names will result in multipathed devices being named **`mpathN`**. While this can be useful if there is only one multipathed device, it can become confusing when there are multiple multipathed devices. To disable user-friendly names, use the **`--user_friendly_names n`** option to **`mpathconf`**. This will result in multipathed devices being named after their WWIDs.

If fine-tuning multipath configuration is desired before starting the **`multipathd`** daemon, use the **`mpathconf`** command with just the **`--enable`** option:

```
[root@nodeY ~]# mpathconf --enable
```

After editing the configuration file, enable and start the **`multipathd`** daemon as normal with the **`systemctl`** command.

## The multipath.conf configuration file

The **`multipath.conf`** configuration file consists of five sections:

| multipath.conf sections | |
|---|---|
| **blacklist {}** | This sections defines which devices should be excluded from the multipath topology discovery. |
| **blacklist_exceptions {}** | This section defines which devices should be included in the multipath topology discovery, despite being listed in the **blacklist** section. |
| **defaults {}** | This section defines the default settings to be used for all multipaths, unless explicitly overridden in the **devices {}** or **multipaths {}** section. |
| **devices {}** | This section contains overrides for the **defaults {}** section for specific types of devices, unless overridden from the **multipaths {}** section. Devices are identified based on their **vendor**, **product**, and **revision** keywords (regular expressions matching information from **sysfs**). |
| **multipaths {}** | This section contains settings for specific multipaths. This section overrides what is defined in the **defaults {}** and **devices {}** section. Multipaths are identified based on their WWIDs (obtained using the **getuid_callout** function). |

An easy way to remember overrides is: **multipaths** > **devices** > **defaults**.

# Blacklisting

Devices can be blacklisted in the configuration file using the **blacklist {}** section of **multipath.conf**. If blacklisting using wildcards, individual devices can be exempted from the blacklist using the **blacklist_exceptions {}** section. Devices can be blacklisted using either their device node or their WWID. The following example shows an example of both:

```
blacklist {
    devnode "^cciss"
    wwid 1234567890abcde
}
```

**Note**

To determine the WWID of a disk device, use the **scsi_id** utility.

```
[root@nodeY ~] /usr/lib/udev/scsi_id -g -u /dev/sdN
360014053bd9ea2a35914e39a556051cf
```

# Defaults

Defaults for all multipaths can be set in the **defaults {}** section of **multipath.conf**. The complete list of all built-in defaults can be found in the file **/usr/share/doc/device-mapper-multipath-*/multipath.conf.defaults**.

Some of the most interesting settings are:

- **path_selector**: The algorithm that determines which path inside a priority group to use for the next I/O. The default of **"service-time 0"**, will send the next I/O request to the path that has the shortest estimated service time. The alternatives are **"round-robin 0"**, which distributes I/O over all paths in the group. The number of request to be sent using one path before switching to the next is determined by the **rr_min_io_rq** setting, and **"queue-length 0"**, which will send the next I/O request to the path with the shortest queue of outstanding requests, and

- **path_grouping_policy**: This setting defines how multiple paths are combined into priority groups. In the default of **failover**, every path will be put into a separate group. On the other hand, with the **multibus** policy, all possible paths are aggregated into a single group. Before configuring a multipath device to use the **multibus** policy, make sure that the storage controller supports active-active connections.

- **path_checker**: This setting determines how the **multipathd** daemon will determine if a path is healthy. Other than the hardware independent options of **directio** and **readsector0**, there are a number of hardware independent checkers. Although the default for this option is **directio**, it is typically overridden in one of the default devices specified in the **devices {}** section.

- **features**: This option specifies the multipath features to enable. Syntax is the form of *num list*, where *num* represents the number of features being enabled and *list* represents list of features being enabled. The two available features are **queue_if_no_path** and **no_partitions**.

- **user_friendly_names**: This setting determines whether multipaths without a defined alias will be named **mpath*N*** (when set to yes), or if they will be named after their WWID.

**Warning**

If the **queue_if_no_path** feature is enabled with the setting, **features "1 queue_if_no_path"**, and paths fail, processes issuing I/O will hang until the paths are restored. This behavior is undesirable in cluster implementations since one node stuck blocking on I/O to a failed storage device can block the rest of the cluster from accessing the storage resource. To avoid this situation, specify a value of **fail** for the **no_path_retry** parameter. Doing so will fail I/O immediately back up to higher layers rather than blocking on I/O indefinitely until paths are recovered.

**Important**

The commented **defaults {}** section in the **multipath.conf** produced by **mpathconf** does *not* reflect the actual built-in defaults of the multipath daemon.

## The devices {} section

In the **devices {}** section, the defaults for specific devices can be overridden. Inside the **devices {}** section, there are individual **device {}** subsections detailing settings for specific devices. Most common storage hardware already have their own section defined in the built-in defaults for the multipath daemon. If a hardware is not (yet) listed, a section for the hardware can be manually added. Following is an example definition for a nonexistent piece of storage hardware. The device itself is selected using a combination of **vendor**, **product**, and **revision**.

```
devices {
    device {
        vendor "MegaHyperSuperStorage"
```

```
        product "BAS"
        revision "513/B"
        features "1 queue_if_no_path"
        path_grouping_policy multibus
        path_checker tur
    }
}
```

In the preceding examples, the **features** line indicates that new I/O requests will be accepted and queued even when no paths are currently available.

## The multipaths {} section

In the **multipaths {}** section, overrides can be defined for specific multipaths. This can be used to set different **path_grouping** policies for a specific multipath. One of the other common uses of the **multipaths {}** section is to define an alias for a multipath. When an alias is set, the name for the device node in **/dev/mapper/** for this multipath will be based on the alias, making it easier to distinguish between different multipaths.

As an example, the following configuration will set an alias of **clusterstorage** for the multipath with a WWID of **"1234567890abcdef"**, as well as a path_selector of **queue-length**.

```
multipaths {
    multipath {
        wwid "1234567890abcdef"
        alias "clusterstorage"
        path_selector "queue-length 0"
    }
}
```

## Adding partitions

To add a partition on a multipathed device, use the following steps:

1. Create the partition on the multipathed device using a partition editor, e.g., **fdisk /dev/mapper/mpath0**.

2. Run the **partprobe** command to update the kernel's view of the partition table on all devices (including the devices collated into a multipath).

3. Run the command, **kpartx -a**, on the multipath device to create device mapper devices for the newly created partition(s).

## Removing a multipath

After removing all paths for a multipath, remove the multipathed device by running the command **multipath -f <*device*>**.

If the **multipathd** daemon has been stopped and there are still device nodes for multipathed devices, flush all multipathed devices by running **multipath -F**. This can be useful when testing out different configurations and see remnants of old configurations lingering around.

**References**

Red Hat Enterprise Linux 7 DM Multipath Guide

- Section 3: Setting Up DM-Multipath
- Section 4: The DM-Multipath Configuration File

**`multipath.conf`**(5) manual page

## Guided Exercise

# Configuring Redundant Storage Access

In this lab, you will configure redundant access to iSCSI storage through a multipath device.

| Resources | |
|---|---|
| **Files** | **/etc/multipath.conf** |
| **Machines** | **workstation**<br>**nodea**<br>**nodeb**<br>**nodec** |

## Outcome(s)

You should be able to configure a multipath device with an alias for redundant access to iSCSI storage.

- Reset **nodea**, **nodeb**, and **nodec**.
- Reset **workstation**.
- Log into **workstation** and run **lab multipath-config setup**.

```
[student@workstation ~]$ lab multipath-config setup
```

- *Optional*: To create and distribute SSH keys from your **workstation** system, you can run the command **lab setupsshkeys setup**.

```
[student@workstation ~]$ lab setupsshkeys setup
```

1. Install the *device-mapper-multipath* package on **nodea**, **nodeb**, and **nodec**.

   1.1. Install the package on each node in the cluster.

```
[student@workstation ~]$ for x in {a..c}; do ssh root@node${x} 'yum install -y
 device-mapper-multipath'; done
```

2. Enable multipath configuration on **nodea**.

```
[root@nodea ~]# mpathconf --enable
```

3. Modify the multipath configuration on **nodea** so that local disks are ignored. Edit the **blacklist** section of the **/etc/multipath.conf** file so that it contains the following lines.

```
blacklist {
        devnode "^vd[a-z]"
}
```

4. On **nodea**, configure the iSCSI storage for multipathing with an alias of **ClusterStorage** and utilizing **failover** policy.

   4.1. Determine the WWID of the iSCSI storage.

```
[root@nodea ~]# /usr/lib/udev/scsi_id -g -u /dev/sda
36001405b7118eb213ff4bc792f144a04
```

   4.2. Configure multipathing for the iSCSI storage by adding the following entries to **/etc/multipath.conf** on **nodea**.

```
multipaths {
        multipath {
                wwid                    WWID
                alias                   ClusterStorage
                path_grouping_policy    failover
        }
}
```

5. Copy the **/etc/multipath.conf** configuration file from **nodea** to the remaining nodes.

```
[root@workstation ~]# rsync -avz -e ssh nodea:/etc/multipath.conf .
receiving incremental file list
multipath.conf

sent 30 bytes  received 1224 bytes  836.00 bytes/sec
total size is 2708  speedup is 2.16
```

```
[root@workstation ~]# for x in {b..c}; do rsync -avz -e ssh multipath.conf node
${x}:/etc/multipath.conf; done
sending incremental file list

sent 39 bytes  received 12 bytes  102.00 bytes/sec
total size is 2708  speedup is 53.10
sending incremental file list

sent 39 bytes  received 12 bytes  102.00 bytes/sec
total size is 2708  speedup is 53.10
```

6. Enable and start the **multipathd** service on each node in the cluster.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'systemctl enable multipathd; systemctl start multipathd; systemctl status
 multipathd'; echo; done
```

7. Verify the multipath configuration on each node in the cluster.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'multipath -ll'; echo; done
===== nodea =====
ClusterStorage (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sdb 8:16 active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sda 8:0  active ready running

===== nodeb =====
ClusterStorage (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sdb 8:16 active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sda 8:0  active ready running

===== nodec =====
ClusterStorage (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running
```

8. On **nodea**, create a new partition on the iSCSI storage.

```
[root@nodea ~]# fdisk /dev/mapper/ClusterStorage
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x6c5f3cb5.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (8192-8388607, default 8192): Enter
Using default value 8192
Last sector, +sectors or +size{K,M,G} (8192-8388607, default 8388607): Enter
```

```
Using default value 8388607
Partition 1 of type Linux and of size 4 GiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 22: Invalid argument.
The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8)
Syncing disks.
```

9. Run the **partprobe** command on each node to update the kernel's view of the partition table on all devices.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} 'partprobe'; echo; done
```

10. Verify that each node has created a device mapper device for the new aprtition.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'ls -la /dev/mapper/'; echo; done
===== nodea =====
total 0
drwxr-xr-x.  2 root root     100 Jun 29 11:45 .
drwxr-xr-x. 19 root root    2820 Jun 29 11:45 ..
lrwxrwxrwx.  1 root root       7 Jun 29 11:44 ClusterStorage -> ../dm-0
lrwxrwxrwx.  1 root root       7 Jun 29 11:44 ClusterStorage1 -> ../dm-1
crw-------.  1 root root 10, 236 Jun 29 11:29 control

===== nodeb =====
total 0
drwxr-xr-x.  2 root root      80 Jun 29 11:43 .
drwxr-xr-x. 19 root root    2800 Jun 29 11:41 ..
lrwxrwxrwx.  1 root root       7 Jun 29 11:43 ClusterStorage -> ../dm-0
lrwxrwxrwx.  1 root root       7 Jun 29 11:43 ClusterStorage1 -> ../dm-1
crw-------.  1 root root 10, 236 Jun 29 11:29 control

===== nodec =====
total 0
drwxr-xr-x.  2 root root      80 Jun 29 11:43 .
```

```
drwxr-xr-x. 19 root root    2800 Jun 29 11:41 ..
lrwxrwxrwx.  1 root root       7 Jun 29 11:43 ClusterStorage -> ../dm-0
lrwxrwxrwx.  1 root root       7 Jun 29 11:43 ClusterStorage1 -> ../dm-1
crw-------.  1 root root 10, 236 Jun 29 11:29 control
```

If you do not see the **ClusterStorage1** devices, create them with the following commands:

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'kpartx -a -v /dev/mapper/ClusterStorage'; echo; done
===== nodea =====
add map ClusterStorage1 (252:1): 0 8380416 linear /dev/mapper/ClusterStorage 8192

===== nodeb =====
add map ClusterStorage1 (252:1): 0 8380416 linear /dev/mapper/ClusterStorage 8192

===== nodec =====
add map ClusterStorage1 (252:1): 0 8380416 linear /dev/mapper/ClusterStorage 8192
```

# Testing Redundant Storage

## Objectives

After completing this section, students should be able to test multipath configuration.

## Monitoring

The **multipath** command can be used to monitor the status of multipaths. When used with one **-l** option, it will show a quick overview of multipath topologies. If the **-l** option is specified twice (**-ll**), it will also perform a check on all paths to see if it is active. If everything is fine, a path will be reported as **active ready**.

The output of **multipath -ll** provides information on each discovered multipath device. This information is comprised of three sections. The sections provide information on the multipath device, its path group(s), and the path(s) which comprise each path group. The following is an example of the information provided for a multipath device.

```
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running
```

The first section of the output provides information on the multipath device. Alias, wwid, device name, vendor, and product information are provided in the first line. The second line displays the size, enabled features, hardware handlers, and write permission settings for the multipath device.

```
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
```

For each multipath device, the **multipath -ll** command also provides information on each of its path groups. The scheduling policy, priority, and status of each path group is displayed for each path group. This is followed by a listing of the path(s) that makes up the path group.

For each path, the device node name, along with the device major and minor information, is provided. Path status is also reported and is useful for assessing the health of each path.

```
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
```

A path which is up and ready for I/O operations will be reported with a status of **ready**.

```
| `- 2:0:0:0 sda 8:0  active ready running
```

On the other hand, a path which is down will be reported with a status of **faulty**.

```
`- 3:0:0:0 sdb 8:16 failed faulty offline
```

## Identifying path grouping policy

While the path grouping policy configured for a multipath device is not explicitly stated in the output of **`multipath -ll`**, it is indicated by the path grouping displayed in the output. For example, a multipath device configured with a path grouping policy of **`failover`** will have only one path in each group. As shown in the following example, multiple path groups are displayed, with each path group containing a single path.

```
        multipath {
                wwid                    360014053bd9ea2a35914e39a556051cf
                path_grouping_policy    failover
        }
```

```
[root@nodea ~]# multipath -ll
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running
```

On the other hand, when configured with a path grouping policy of **`multibus`**, a multipath device will group all paths into one priority group. As shown in the following example, with this configuration, only a single path group is displayed and all paths in the multipath device are members of this single group.

```
        multipath {
                wwid                    360014053bd9ea2a35914e39a556051cf
                path_grouping_policy    multibus
        }
```

```
[root@nodea ~]# multipath -ll
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=1 status=active
  |- 2:0:0:0 sda 8:0  active ready running
  `- 3:0:0:0 sdb 8:16 active ready running
```

## Observing path failover

The **`multipath -ll`** command can also be useful in assessing the failover activities of multipath devices configured with the **`failover`** path grouping policy. The failover policy implements an active-passive multipath configuration. Therefore, at any given time, only one path group will be in **`active`** status while the remaining path groups are waiting in **`enabled`** status, as shown in the following example.

```
[root@nodea ~]# multipath -ll
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running
```

The following example illustrates the change in the output of the **`multipath -ll`** command when the path failure occurs on a passive path group. While the status for the path in the passive path group has changed, the status of the active path group and its corresponding path remains unchanged and unimpaired.

```
[root@nodea ~]# multipath -ll
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready  running
`-+- policy='service-time 0' prio=0 status=enabled
  `- 3:0:0:0 sdb 8:16 failed faulty offline
```

The following example illustrates the change in the output of the **`multipath -ll`** command when the path failure occurs on a path in the active path group. The status for the previously active path has changed to **`failed faulty offline`**. Consequently, the status of the corresponding path group has also changed, from **`active`** to **`enabled`**. Due to the failover configuration, the previous passive path has transitioned to a status of **`active ready running`**. The corresponding path group's status has also changed from **`enabled`** to **`active`**.

```
[root@nodea ~]# multipath -ll
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=0 status=enabled
| `- 2:0:0:0 sda 8:0  failed faulty offline
`-+- policy='service-time 0' prio=1 status=active
  `- 3:0:0:0 sdb 8:16 active ready running
```

**Important**

Once a failed path recovers, the current active path will remain active. This is the case even if the failed path was the previously active path. Path group failback does not occur without manual intervention. This behavior is controlled in **`/etc/multipath.conf`** by the **`failback`** option, which defaults to **`manual`**.

**References**

Red Hat Enterprise Linux 7 DM Multipath Guide

- Section 5.9: Multipath Command Options

**`multipath`**(8) manual page

## Guided Exercise

# Testing Redundant Storage

In this lab, you will test the redundancy of iSCSI storage through a multipath device.

| Resources | |
|---|---|
| **Files** | **/mnt** |
| **Machines** | **nodea** |

## Outcome(s)

You should be able to test and observe the redundancy of a multipath device configured with the failover path grouping policy.

- The cluster node, **nodea**, should be configured with redundant access to iSCSI storage per the previous exercise.
- Log into the **nodea** as the **root** user.

**1.** On **nodea**, create an XFS file system on the partition previously created on the **/dev/mapper/ClusterStorage** multipath device and mount the file system at **/mnt**.

1.1. Create the XFS file system on **/dev/mapper/ClusterStorage1**.

```
[root@nodea ~]# mkfs.xfs /dev/mapper/ClusterStorage1
meta-data=/dev/mapper/ClusterStorage1 isize=256    agcount=8, agsize=130944 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=1047552, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

1.2. Mount the file system.

```
[root@nodea ~]# mount /dev/mapper/ClusterStorage1 /mnt
```

**2.** Determine the active path in use by the multipath device.

```
[root@nodea ~]# multipath -ll
ClusterStorage (36001405acc49ded72944f3ab14e1e525) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 3:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 2:0:0:0 sdb 8:16 active ready running
```

3. Determine the network providing connectivity for the active path to the iSCSI storage.

```
[root@nodea ~]# iscsiadm -m session -P 3 | egrep 'Current Portal|scsi disk'
 Current Portal: 192.168.1.9:3260,1
   Attached scsi disk sdb  State: running
 Current Portal: 192.168.2.9:3260,1
   Attached scsi disk sda  State: running
```

4. Disable the network interface for the active path in use by the multipath device and observe the failover of the path to the iSCSI storage.

   4.1. Monitor the status of the multipath device.

```
[root@nodea ~]# watch -n 1 -d "multipath -ll"
Every 1.0s: multipath -ll                              Jun 19 09:48:36 2015

ClusterStorage (36001405acc49ded72944f3ab14e1e525) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=enabled
| `- 3:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=active
  `- 2:0:0:0 sdb 8:16 active ready running
```

   4.2. In another terminal window, disable the network interface.

```
[root@nodea ~]# ifdown eth2
```

   4.3. Observe the failover to the redundant path.

```
[root@nodea ~]# watch -n 1 -d "multipath -ll"
Every 1.0s: multipath -ll                              Jun 19 09:50:36 2015

ClusterStorage (36001405acc49ded72944f3ab14e1e525) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 3:0:0:0 sda 8:0  active ready  running
`-+- policy='service-time 0' prio=0 status=enabled
  `- 2:0:0:0 sdb 8:16 failed faulty offline
```

5. Re-enable the network interface to restore redundancy to the multipath device.

   5.1. Enable the network interface.

```
[root@nodea ~]# ifup eth2
```

5.2. Observe the restoration of the redundancy to the iSCSI storage.

```
[root@nodea ~]# watch -n 1 -d "multipath -ll"
Every 1.0s: multipath -ll                                Jun 19 09:52:36 2015

ClusterStorage (36001405acc49ded72944f3ab14e1e525) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 3:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 2:0:0:0 sdb 8:16 active ready running
```

## Lab

# Accessing Storage Devices Redundantly

### Performance Checklist

In this lab, you will configure an identically named multipath device on cluster nodes to provide redundant access to iSCSI storage.

| Resources | |
|---|---|
| **Files** | **/etc/multipath.conf**<br>**/etc/multipath/bindings**<br>**/etc/fstab** |
| **Machines** | **workstation**<br>**nodea**<br>**nodeb**<br>**nodec** |

### Outcome(s)

You should be able to create an identically named multipath device on cluster nodes and configure it for redundant access to iSCSI storage.

- Reset **nodea**, **nodeb**, and **nodec**.
- Reset **workstation**. Log into **workstation** and run **lab multipath-iscsi setup**.
- *Optional*: To create and distribute SSH keys from your **workstation** system, you can run the command **lab setupsshkeys setup**.

```
[student@workstation ~]$ lab setupsshkeys setup
```

The hosts **nodea**, **nodeb**, and **nodec** have been configured to form a three-node cluster. The **workstation** system provides a iSCSI target, **iqn.2015-06.com.example:cluster**, which is available through both portal address **192.168.1.9** and **192.168.2.9**. All three cluster nodes have dual network connectivity to the target's two portal addresses.

Configure **nodea**, **nodeb**, and **nodec** as iSCSI initiator nodes with the client names **iqn.2015-06.com.example:nodea**, **iqn.2015-06.com.example:nodeb**, and **iqn.2015-06.com.example:nodec**, respectively. On each node, use the two network paths to configure an identically named multipath device, **/dev/mapper/mpath*n***, which provides redundant access to the iSCSI target on **workstation** using the **failover** policy. To ensure timely failover in the case of a path failure, configure the iSCSI initiators with a **replacement_timeout** of **5** seconds.

Using the multipath device configured on each node, create a new partition on the iSCSI storage and format it with the XFS file system. Configure the file system to mount at **/mnt** upon boot.

1. Configure each cluster node as a iSCSI initiator node by installing the initiator utilities, setting the unique iSCSI client name, and starting the iSCSI client service.

2. On each cluster node, discover and log into the configured target on the iSCSI target server.
3. Install the **`device-mapper-multipath`** package and enable multipath configuration on each cluster node.
4. Configure multipathing on **`nodea`** to ignore the local disk and to create a **`/dev/mpath`*`n`*** multipath device which provides redundant access to the iSCSI storage using the **`failover`** policy.
5. Configure **`nodeb`** and **`nodec`** so that a **`/dev/mpath`*`n`*** multipath device is named and configured identical to the multipath device on **`nodea`**.
6. Create a new partition on the multipath device. Format it with an XFS file system and configure it to mount at **`/mnt`** upon boot.

## Solution

# Accessing Storage Devices Redundantly

### Performance Checklist

In this lab, you will configure an identically named multipath device on cluster nodes to provide redundant access to iSCSI storage.

| Resources | |
|---|---|
| **Files** | **/etc/multipath.conf**<br>**/etc/multipath/bindings**<br>**/etc/fstab** |
| **Machines** | **workstation**<br>**nodea**<br>**nodeb**<br>**nodec** |

### Outcome(s)

You should be able to create an identically named multipath device on cluster nodes and configure it for redundant access to iSCSI storage.

- Reset **nodea**, **nodeb**, and **nodec**.
- Reset **workstation**. Log into **workstation** and run **lab multipath-iscsi setup**.
- *Optional*: To create and distribute SSH keys from your **workstation** system, you can run the command **lab setupsshkeys setup**.

```
[student@workstation ~]$ lab setupsshkeys setup
```

The hosts **nodea**, **nodeb**, and **nodec** have been configured to form a three-node cluster. The **workstation** system provides a iSCSI target, **iqn.2015-06.com.example:cluster**, which is available through both portal address **192.168.1.9** and **192.168.2.9**. All three cluster nodes have dual network connectivity to the target's two portal addresses.

Configure **nodea**, **nodeb**, and **nodec** as iSCSI initiator nodes with the client names **iqn.2015-06.com.example:nodea**, **iqn.2015-06.com.example:nodeb**, and **iqn.2015-06.com.example:nodec**, respectively. On each node, use the two network paths to configure an identically named multipath device, **/dev/mapper/mpath*n***, which provides redundant access to the iSCSI target on **workstation** using the **failover** policy. To ensure timely failover in the case of a path failure, configure the iSCSI initiators with a **replacement_timeout** of **5** seconds.

Using the multipath device configured on each node, create a new partition on the iSCSI storage and format it with the XFS file system. Configure the file system to mount at **/mnt** upon boot.

1. Configure each cluster node as a iSCSI initiator node by installing the initiator utilities, setting the unique iSCSI client name, and starting the iSCSI client service.

1.1. Install the *iscsi-initiator-utils* RPM on each node.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} 'yum install -y iscsi-
initiator-utils'; done
```

1.2. Create a unique iSCSI qualified name for each client initiator by modifying the **InitiatorName** setting in **/etc/iscsi/initiatorname.iscsi**. Use the client system name as the optional string after the colon.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} "echo
 'InitiatorName=iqn.2015-06.com.example:node${x}' > /etc/iscsi/
initiatorname.iscsi"; done
```

1.3. Configure the **replacement_timeout** setting on each iSCSI client.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x}
 "sed -i 's/^node.session.timeo.replacement_timeout.*=.*$/
node.session.timeo.replacement_timeout = 5/' /etc/iscsi/iscsid.conf"; done
```

1.4. Enable and start the **iscsi** client service.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} 'systemctl enable iscsi;
 systemctl start iscsi'; done
```

**2.** On each cluster node, discover and log into the configured target on the iSCSI target server.

2.1. Discover the configured iSCSI targets provided by the iSCSI target server portal.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'iscsiadm -m discovery -t st -p 192.168.1.9'; echo; done
===== nodea =====
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster

===== nodeb =====
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster

===== nodec =====
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster
```

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'iscsiadm -m discovery -t st -p 192.168.2.9'; echo; done
===== nodea =====
192.168.2.9:3260,1 iqn.2015-06.com.example:cluster

===== nodeb =====
192.168.2.9:3260,1 iqn.2015-06.com.example:cluster

===== nodec =====
192.168.2.9:3260,1 iqn.2015-06.com.example:cluster
```

2.2. Log into the presented iSCSI target.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'iscsiadm -m node -T iqn.2015-06.com.example:cluster -p 192.168.1.9 -l';
 echo; done
===== nodea =====
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] successful.

===== nodeb =====
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] successful.

===== nodec =====
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.1.9,3260] successful.
```

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'iscsiadm -m node -T iqn.2015-06.com.example:cluster -p 192.168.2.9 -l';
 echo; done
===== nodea =====
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] successful.

===== nodeb =====
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] successful.

===== nodec =====
Logging in to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] (multiple)
Login to [iface: default, target: iqn.2015-06.com.example:cluster, portal:
 192.168.2.9,3260] successful.
```

2.3. Identify the newly available block device created by the iSCSI target login.

```
[root@workstation ~]#
for x in {a..c}; do echo "===== node${x} ====="; ssh node${x} 'lsblk'; echo; done

===== nodea =====
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda      8:0    0   4G  0 disk
sdb      8:16   0   4G  0 disk
vda    253:0    0  10G  0 disk
```

```
└─vda1 253:1    0  10G  0 part /

===== nodeb =====
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda      8:0    0   4G  0 disk
sdb      8:16   0   4G  0 disk
vda    253:0    0  10G  0 disk
└─vda1 253:1    0  10G  0 part /

===== nodec =====
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda      8:0    0   4G  0 disk
sdb      8:16   0   4G  0 disk
vda    253:0    0  10G  0 disk
└─vda1 253:1    0  10G  0 part /
```

3. Install the **device-mapper-multipath** package and enable multipath configuration on each cluster node.

   3.1. Install the package on each cluster node.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'yum install -y device-mapper-multipath'; echo; done
```

   3.2. Enable multipath configuration on each cluster node.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} 'mpathconf --enable'; done
```

4. Configure multipathing on **nodea** to ignore the local disk and to create a **/dev/mpath*n*** multipath device which provides redundant access to the iSCSI storage using the **failover** policy.

   4.1. Log into **nodea** and modify its multipath configuration so that local disks are ignored by configuring the following **blacklist** statement in **/etc/multipath.conf**.

```
blacklist {
        devnode "^vd[a-z]"
}
```

   4.2. Determine the WWID of the iSCSI storage.

```
[root@nodea ~]# /usr/lib/udev/scsi_id -g -u /dev/sda
360014053bd9ea2a35914e39a556051cf
```

   4.3. Configure multipathing for the iSCSI storage by adding the following entries to **/etc/multipath.conf** on **nodea**.

```
multipaths {
        multipath {
                wwid                    WWID
                path_grouping_policy    failover
        }
}
```

4.4. Enable and start the **multipathd** service on **nodea**.

```
[root@nodea ~]# systemctl enable multipathd; systemctl start multipathd; systemctl
 status multipathd
```

**5.** Configure **nodeb** and **nodec** so that a **/dev/mpath*n*** multipath device is named and configured identical to the multipath device on **nodea**.

5.1. Copy **/etc/multipath.conf** from **nodea** to **nodeb** and **nodec**.

```
[root@workstation ~]# rsync -avz -e ssh nodea:/etc/multipath.conf .
receiving incremental file list
multipath.conf

sent 30 bytes  received 1204 bytes  2468.00 bytes/sec
total size is 2646  speedup is 2.14
```

```
[root@workstation ~]# for x in {b..c}; do echo "===== node${x} ====="; rsync -avz
 -e ssh multipath.conf node${x}:/etc/multipath.conf; echo; done
===== nodeb =====
sending incremental file list

sent 39 bytes  received 12 bytes  102.00 bytes/sec
total size is 2646  speedup is 51.88

===== nodec =====
sending incremental file list

sent 39 bytes  received 12 bytes  102.00 bytes/sec
total size is 2646  speedup is 51.88
```

5.2. Copy **/etc/multipath/bindings** from **nodea** to **nodeb** and **nodec**.

```
[root@workstation ~]# rsync -avz -e ssh nodea:/etc/multipath/bindings .
receiving incremental file list
bindings

sent 30 bytes  received 257 bytes  574.00 bytes/sec
total size is 241  speedup is 0.84
```

```
[root@workstation ~]# for x in {b..c}; do echo "===== node${x} ====="; rsync -avz
 -e ssh bindings node${x}:/etc/multipath/bindings; echo; done
===== nodeb =====
sending incremental file list
bindings

sent 252 bytes  received 31 bytes  566.00 bytes/sec
total size is 241  speedup is 0.85

===== nodec =====
sending incremental file list
bindings

sent 252 bytes  received 31 bytes  566.00 bytes/sec
total size is 241  speedup is 0.85
```

5.3. Enable and start the **multipathd** service on **nodeb** and **nodec**.

```
[root@workstation ~]# for x in {b..c}; do echo "===== node${x} ====="; ssh node
${x} 'systemctl enable multipathd; systemctl start multipathd; systemctl status
 multipathd'; echo; done
```

5.4. Verify the multipath configuration on each node in the cluster.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'multipath -ll'; echo; done
===== nodea =====
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running

===== nodeb =====
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running

===== nodec =====
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor
size=4.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 2:0:0:0 sda 8:0  active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 3:0:0:0 sdb 8:16 active ready running
```

6. Create a new partition on the multipath device. Format it with an XFS file system and configure it to mount at **/mnt** upon boot.

6.1. On **nodea**, create a new partition on the iSCSI storage.

```
[root@nodea ~]# fdisk /dev/mapper/mpatha
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0xe3e8c2c2.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (8192-8388607, default 8192): Enter
Using default value 8192
Last sector, +sectors or +size{K,M,G} (8192-8388607, default 8388607): Enter
Using default value 8388607
Partition 1 of type Linux and of size 4 GiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 22: Invalid argument.
The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8)
Syncing disks.
```

6.2. Run the **partprobe** command on each node to update the kernel's view of the partition table on all devices.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} 'partprobe'; echo; done
```

6.3. Create the device mapper device for the new partition on each node in the cluster.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'ls -la /dev/mapper/'; echo; done
===== nodea =====
total 0
drwxr-xr-x.  2 root root      80 Jul  1 11:20 .
drwxr-xr-x. 19 root root    2800 Jul  1 11:20 ..
crw-------.  1 root root 10, 236 Jul  1 10:06 control
lrwxrwxrwx.  1 root root       7 Jul  1 11:47 mpatha -> ../dm-0

===== nodeb =====
total 0
drwxr-xr-x.  2 root root      80 Jul  1 11:43 .
drwxr-xr-x. 19 root root    2800 Jul  1 11:43 ..
```

```
crw-------.  1 root root 10, 236 Jul  1 10:06 control
lrwxrwxrwx.  1 root root       7 Jul  1 11:43 mpatha -> ../dm-0

===== nodec =====
total 0
drwxr-xr-x.  2 root root      80 Jul  1 11:43 .
drwxr-xr-x. 19 root root    2800 Jul  1 11:43 ..
crw-------.  1 root root 10, 236 Jul  1 10:06 control
lrwxrwxrwx.  1 root root       7 Jul  1 11:43 mpatha -> ../dm-0
```

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'kpartx -a -v /dev/mapper/mpatha'; echo; done
===== nodea =====
add map mpatha1 (252:1): 0 8380416 linear /dev/mapper/mpatha 8192

===== nodeb =====
add map mpatha1 (252:1): 0 8380416 linear /dev/mapper/mpatha 8192

===== nodec =====
add map mpatha1 (252:1): 0 8380416 linear /dev/mapper/mpatha 8192
```

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'ls -la /dev/mapper/'; echo; done
===== nodea =====
total 0
drwxr-xr-x.  2 root root     100 Jul  1 11:55 .
drwxr-xr-x. 19 root root    2820 Jul  1 11:55 ..
crw-------.  1 root root 10, 236 Jul  1 10:06 control
lrwxrwxrwx.  1 root root       7 Jul  1 11:47 mpatha -> ../dm-0
lrwxrwxrwx.  1 root root       7 Jul  1 11:55 mpatha1 -> ../dm-1

===== nodeb =====
total 0
drwxr-xr-x.  2 root root     100 Jul  1 11:55 .
drwxr-xr-x. 19 root root    2820 Jul  1 11:55 ..
crw-------.  1 root root 10, 236 Jul  1 10:06 control
lrwxrwxrwx.  1 root root       7 Jul  1 11:43 mpatha -> ../dm-0
lrwxrwxrwx.  1 root root       7 Jul  1 11:55 mpatha1 -> ../dm-1

===== nodec =====
total 0
drwxr-xr-x.  2 root root     100 Jul  1 11:55 .
drwxr-xr-x. 19 root root    2820 Jul  1 11:55 ..
crw-------.  1 root root 10, 236 Jul  1 10:06 control
lrwxrwxrwx.  1 root root       7 Jul  1 11:43 mpatha -> ../dm-0
lrwxrwxrwx.  1 root root       7 Jul  1 11:55 mpatha1 -> ../dm-1
```

6.4. On **nodea**, create an XFS file system on the new partition.

```
[root@nodea ~]# mkfs.xfs /dev/mapper/mpatha1
meta-data=/dev/mapper/mpatha1    isize=256    agcount=8, agsize=130944 blks
         =                       sectsz=512   attr=2, projid32bit=1
```

```
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=1047552, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

6.5. Configure the file system to be mounted to **/mnt** upon boot on each node.

```
[root@workstation ~]# for x in {a..c}; do ssh node${x} "echo '/dev/mapper/
mpatha1 /mnt xfs _netdev 0 0' >> /etc/fstab"; done
```

6.6. Mount the file system on each node.

```
[root@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh node
${x} 'mount /mnt; mount | grep -w /mnt'; echo; done
===== nodea =====
/dev/mapper/mpatha1 on /mnt type xfs
 (rw,relatime,seclabel,attr2,inode64,noquota,_netdev)

===== nodeb =====
/dev/mapper/mpatha1 on /mnt type xfs
 (rw,relatime,seclabel,attr2,inode64,noquota,_netdev)

===== nodec =====
/dev/mapper/mpatha1 on /mnt type xfs
 (rw,relatime,seclabel,attr2,inode64,noquota,_netdev)
```

# Summary

In this chapter, you learned:

- Multipathing, which requires installation of the *device-mapper-multipath* package, can be used to provide redundant access to storage.
- With user-friendly names enabled, multipath devices are created as **`/dev/mapper/mpathN`** devices.
- Multipath devices default to a active-passive configuration using the **`failover`** path grouping policy.
- Active-active multipath configuration can be implemented with the **`multibus`** path grouping policy.
- The **`multipath`** command can be used to monitor the status of multipath devices, their path groups, and the paths which comprise each path group.

Chapter 10

# Configuring Logical Volumes For Cluster File Systems

**Goal**

Manage clustered LVM.

**Objectives**

- Review LVM concepts and tools.
- Manage and deploy HA-LVM.
- Manage and deploy clustered LVM.

**Sections**

- Reviewing LVMs (and Practice)
- Managing High Availability Logical Volumes (and Practice)
- Managing Clustered Logical Volumes (and Practice)

**Lab**

- Configuring Logical Volumes for Cluster File Systems

# Reviewing LVMs

## Objectives

After completing this section, students should be able to revert inadvertent changes to LVM volume groups.

## LVM review

Logical Volume Management (LVM) provides administrators with a powerful storage virtualization framework. One or more databearers (physical volumes) get aggregated into a storage pool (volume group), from which volumes can get carved to act as block devices (logical volumes). LVM also allows for the striping and/or mirroring of data between physical volumes. The following graphic shows the basic steps of creating a volume group and logical volumes.

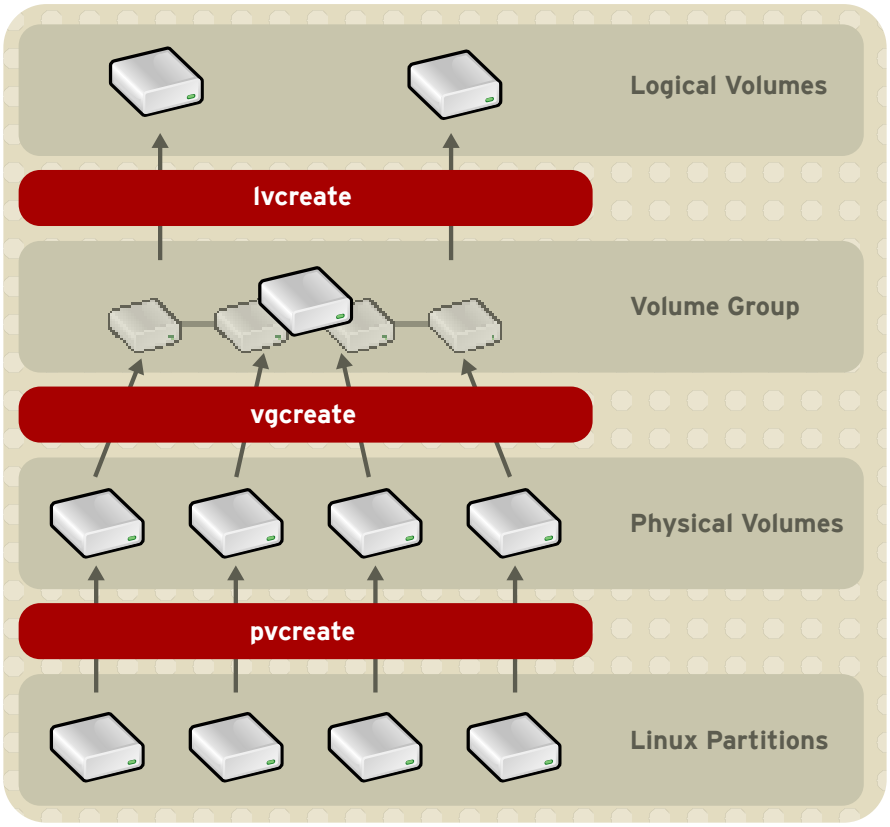



**Figure 10.1: An overview of the LVM architecture**

Just like dm-multipath, LVM uses the kernel *Device Mapper* subsystem to create its devices. Logical volumes are created in **`/dev/`** as **`dm-*`** device nodes, but with symlinks in both **`/dev/mapper`** and **`/dev/<vgname>/`** using more administrator-friendly (and persistent) names.

## Configuration files

The behavior of LVM can be configured in **`/etc/lvm/lvm.conf`**. This file has settings for controlling locking behavior, which devices should be scanned for LVM signatures, and which name

should be displayed for physical volumes that are available on multiple device nodes, as well as many other behaviors. The following table lists some of the more common options:

| /etc/lvm/lvm.conf Options | |
|---|---|
| **dir** | Which directory to scan for physical volume device nodes. |
| **obtain_device_list_from_udev** | If udev should be used to obtain a list of block devices eligible for scanning. |
| **preferred_names** | A list of regular expressions detailing which device names should have preference when displaying devices. If multiple device names (nodes) are found for the same PV, the LVM tools will give preference to those that appear earlier in this list. |
| **filter** | A list of regular expressions prefixed with **a** for *Add* or **r** for *Remove*. This list determines which device nodes will be scanned for the presence of a PV signature. The default is **[ "a/.*/" ]**, which adds every single device. This option can be used to remove devices that should never be scanned. |
| **backup** | This setting determines if a text-based backup of volume group metadata should be stored after every change. This backup can be used to restore a volume group if the on-disk metadata gets corrupted. |
| **backup_dir** | This setting specifies where the backup of volume group metadata should be stored. |
| **archive** | This setting determines if old volume group configurations/layouts should be archived so they can later be used to revert changes. |
| **archive_dir** | This setting specifies where archives of old configurations should be stored. |

# Reverting LVM changes

Sometimes, administrators may perform erroneous changes to a volume group, e.g., shrinking a logical volume before shrinking the file system on that logical volume. In this example, an attempt may be made to extend the logical volume to the previous size, but there is no guarantee that the same blocks on disk will be used as before and the file system could therefore possibly be corrupted.

To offer recovery options in this type of situation, LVM can be configured to keep archived copies of volume group metadata. If the **archive** option is set to **1** in **/etc/lvm/lvm.conf**, the LVM tools will create an archived copy of the current volume group metadata *before* making any changes on disk. In a default configuration, these archives can be found in **/etc/lvm/archive**. To view a list of all archived metadata for a volume group, examine the files in **/etc/lvm/**

**archive**, paying special attention to the **description** lines. Alternatively, the **vgcfgrestore --list <*vgname*>** command can be used.

```
[root@desktopX ~]# vgcfgrestore --list vg_example
File:   /etc/lvm/archive/vg_example_00000-943759797.vg
VG name:      vg_example
Description:  Created *before* executing 'vgcreate vg_example /dev/sda5'
Backup Time:  Mon Mar 19 07:01:47 2012


File:   /etc/lvm/archive/vg_example_00001-1528176681.vg
VG name:      vg_example
Description:  Created *before* executing 'lvcreate -L 1G -n lv_example vg_example'
Backup Time:  Mon Mar 19 07:02:00 2012


File:   /etc/lvm/archive/vg_example_00002-1484695080.vg
VG name:      vg_example
Description:  Created *before* executing 'lvresize -L -256M /dev/vg_example/
lv_example'
Backup Time:  Mon Mar 19 07:02:34 2012


File:   /etc/lvm/backup/vg_example
VG name:      vg_example
Description:  Created *after* executing 'lvresize -L -256M /dev/vg_example/
lv_example'
Backup Time:  Mon Mar 19 07:02:34 2012
```

The preceding example shows the volume group **vg_example** being created, then a logical volume **lv_example** is created in the volume group, and finally the logical volume is reduced by **256** MiB. It is worth noting that the last output block shows the backup of the current volume group metadata in **/etc/lvm/backup**, created *after* the logical volume was shrunk.

To undo a change, make sure that the logical volume is not currently in use (unmount any file systems that might have been created on the logical volume), and then use the **vgcfgrestore** command again, but this time with the **-f <*archive_file*>** option.

```
[root@desktopX ~]# vgcfgrestore -f /etc/lvm/archive/vg_example_00001-1528176681.vg
 vg_example
Restored volume group vg_example
```

In some cases, it might be necessary to deactivate, then reactivate the logical volume to make sure that all changes are committed in memory, as well.

```
[root@desktopX ~]# lvchange -an /dev/vg_example/lv_example
[root@desktopX ~]# lvchange -ay /dev/vg_example/lv_example
```

**References**

Red Hat Enterprise Linux 7 Logical Volume Manager Administration Guide

- Section 2: LVM Components
- Section 4.3.13: Backing Up Volume Group Metadata
- Section 4.5: Controlling LVM Device Scans with Filters
- Section 6.4: Recovering Physical Volume Metadata

**`lvm.conf`**(5) and **`vgcfgrestore`**(8) manual pages

## Guided Exercise

# Reverting LVM Changes

In this lab, you will revert changes made to a logical volume.

| Resources | |
|---|---|
| **Machines** | **nodea** and **workstation** |

## Outcome(s)

You should be able to recover from erroneous changes made to an LVM volume group.

- Reset your **workstation** system and all four of your **nodeY** systems.
- From your **workstation** system, run the command **lab lvm-revert setup**.
- Log into **nodea** as the **root** user.

1. On **nodea**, you will find a volume group, **vgsrv**. Create a 2 GiB logical volume called **resizeme** on this volume group.

```
[root@nodea ~]# vgs
  VG    #PV #LV #SN Attr   VSize VFree
  vgsrv   1   0   0 wz--n- 4.00g 4.00g
```

```
[root@nodea ~]# lvcreate -n resizeme -L2G vgsrv
  Logical volume "resizeme" created.
```

2. Create a new **XFS** file system on your **resizeme** logical volume.

```
[root@nodea ~]# mkfs -t xfs /dev/vgsrv/resizeme
```

3. Mount your new file system on **/mnt** and create some test files on it.

```
[root@nodea ~]# mount /dev/vgsrv/resizeme /mnt
[root@nodea ~]# touch /mnt/file{0..9}
```

4. Unmount your new file system.

```
[root@nodea ~]# umount /mnt
```

5. Resize the logical volume **/dev/vgsrv/resizeme** to 1 GiB; accidentally *forget* to resize the file system first.

```
[root@nodea ~]# lvresize -L1G /dev/vgsrv/resizeme
WARNING: Reducing active logical volume to 1.00 GiBRNING: Reducing active logical
 volume to 1.00 GiB
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce resizeme? [y/n]: y
  Size of logical volume vgsrv/resizeme changed from 2.00 GiB (512 extents) to
 1.00 GiB (256 extents).
  Logical volume resizeme successfully resized
```

6. Attempt to mount **/dev/vgsrv/resizeme** on **/mnt**.

```
[root@nodea ~]# mount /dev/vgsrv/resizeme /mnt
mount: /dev/mapper/vgsrv-resizeme: can't read superblock
```

7. Examine the archive files for the **vgsrv** volume group. Locate the one that has a description of **Created *before* executing 'lvresize -L1G /dev/vgsrv/resizeme'**.

```
[root@nodea ~]# vgcfgrestore -l vgsrv

  File:  /etc/lvm/archive/vgsrv_00000-1480297499.vg
  Couldn't find device with uuid acqVsw-O9jn-dPIT-6eZi-G7Zh-jD6C-g1oHXd.
  VG name:     vgsrv
  Description: Created *before* executing 'vgcreate vgsrv /dev/sda'
  Backup Time: Wed Jul 15 12:32:53 2015


  File:  /etc/lvm/archive/vgsrv_00001-1917636328.vg
  VG name:     vgsrv
  Description: Created *before* executing 'lvcreate -n resizeme -L2G vgsrv'
  Backup Time: Wed Jul 15 12:51:15 2015


  File:  /etc/lvm/archive/vgsrv_00002-565820768.vg
  VG name:     vgsrv
  Description: Created *before* executing 'lvresize -L1G /dev/vgsrv/resizeme'
  Backup Time: Wed Jul 15 12:57:29 2015


  File:  /etc/lvm/backup/vgsrv
  VG name:     vgsrv
  Description: Created *after* executing 'lvresize -L1G /dev/vgsrv/resizeme'
  Backup Time: Wed Jul 15 12:57:29 2015
```

8. Using the file you just found, revert the resize operation on **/dev/vgsrv/resizeme**.

```
[root@nodea ~]# vgcfgrestore -f /etc/lvm/archive/vgsrv_00002-565820768.vg vgsrv
  Restored volume group vgsrv
```

9. To make sure that everything is updated with regard to our logical volume size, deactivate, then reactivate the **/dev/vgsrv/resizeme** logical volume.

```
[root@nodea ~]# lvchange -an /dev/vgsrv/resizeme
[root@nodea ~]# lvchange -ay /dev/vgsrv/resizeme
```

10. Attempt to mount **/dev/vgsrv/resizeme** on **/mnt**. This time, it should work.

```
[root@nodea ~]# mount /dev/vgsrv/resizeme /mnt
```

11. **Important Cleanup:** Unmount your file system and remove the **resizeme** logical volume.

```
[root@nodea ~]# umount /mnt
[root@nodea ~]# lvremove /dev/vgsrv/resizeme
```

# Managing High Availability Logical Volumes

## Objectives

After completing this section, students should be able to configure HA-LVM.

## HA-LVM

There are two ways to use LVM on shared storage in a cluster, Clustered LVM (described in the next section) and HA-LVM. With Clustered LVM, all volume groups and logical volumes on shared storage are available to all cluster nodes all of the time. With HA-LVM, a volume group and its logical volumes can only be accessed by one node at a time.

Clustered LVM is a good choice when working with a shared file system, like **GFS2**. On the other hand, HA-LVM is a good choice when working with a more traditional file system like **ext4** or **XFS**, and restricted access to just one node at a time is desired to prevent file system and/or data corruption.

## Creating an HA-LVM volume

To create an HA-LVM logical volume, create it and its associated physical volume and volume group as would normally be performed with standard LVM commands.

```
[root@nodeY ~]# pvcreate /dev/sdd1
[root@nodeY ~]# vgcreate shared_vg /dev/sdd1
[root@nodeY ~]# lvcreate -L 1G -n ha_lv shared_vg
[root@nodeY ~]# mkfs -t xfs /dev/shared_vg/ha_lv
```

## Configuring HA-LVM

There are two ways to configure HA-LVM. One method uses LVM *tags*, and the other, newer method uses the clustered LVM daemon, **clvmd**.

The use of **clvmd** in conjunction with the **LVM** resource agent used by Pacemaker is not supported for active/passive LVM configurations. Therefore, in a cluster managed with Pacemaker, HA-LVM must use the tagging method. Since this course uses Pacemaker clusters, it will use the tagging method in its examples and exercises of HA-LVM configuration.

LVM performs file locking to prevent conflicting LVM commands from running concurrently, either on a single machine or across a cluster. To implement HA-LVM, it is necessary to ensure that LVM is configured to use local file-based locking.

This is the default locking type for LVM and is specified by setting the **locking_type** parameter to **1** in **/etc/lvm/lvm.conf**.

```
locking_type = 1
```

In order to ensure that HA-LVM volume groups are only activated on a single node in a cluster at any given time, volume group activation must be completely in the control of the cluster.

Activation of an HA-LVM volume group performed outside of the cluster's control can result in corruption of the volume group's metadata.

To prevent this, the list of volume groups that are not part of a cluster's HA-LVM configuration must be explicitly defined on each cluster node. Populate this list into the **volume_list** parameter in the **/etc/lvm/lvm.conf** file on each node in the cluster.

```
volume_list = [ "vg_root", "vg_home" ]
```

**Important**

Even if there are no volume groups that are outside of the cluster's control, the **volume_list** parameter must still be defined. In this case, define the parameter with an empty list per the following example.

```
volume_list = []
```

Only volume groups defined in the **volume_list** parameter will be activated by LVM. By excluding cluster HA-LVM volume groups from this list, their activation is effectively placed in the sole control of the cluster.

To ensure that cluster-controlled volume groups are not inadvertently activated by the **initramfs** boot image, it is also necessary to rebuild the image on each cluster node. Each node should be rebooted once the image rebuild executed on it is completed.

```
[root@nodeY ~]# dracut -H -f /boot/initramfs-$(uname -r).img $(uname -r); reboot
```

After all cluster nodes have been rebooted, verify that cluster services started successfully on each cluster. With the changes to LVM configuration activated, the HA-LVM volume group will be in the cluster's control. The HA-LVM volume group and its logical volume will be visible to all the cluster nodes, but the logical volumes will be in an inactive state.

```
[root@nodeY ~]# lvs
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  shared_vg ha_lv     -wi------- 1.00g
```

When the cluster activates the volume group on the single active node, the volume group's logical volume will be set to active on that node but will remain in an inactive state on the remaining nodes in the cluster. This is demonstrated by the following example where **nodea** is the active note in the cluster.

```
# for x in {a..c}; do echo "===== node${x} ====="; ssh root@node${x} 'lvs'; echo;
 done
===== nodea =====
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi-a----- 1.00g


===== nodeb =====
```

```
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi------- 1.00g


===== nodec =====
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi------- 1.00g
```

# Adding an HA-LVM resource to a cluster

Once an HA-LVM volume group is configured, it can be added to a cluster as an LVM resource. The cluster LVM resource agent is responsible for activating, deactivating, and locking the volume group and its logical volumes.

To add an HA-LVM resource, use the **`pcs resource create`** command in conjunction with the **`LVM`** resource agent.

```
[root@nodeY ~]# pcs resource create halvm LVM volgrpname=clustervg exclusive=true
```

**Important**

An **`LVM`** resource is *not* a file system resource. Before a file system on an HA-LVM volume in a **`LVM`** resource can be utilized, a separate **`Filesystem`** resource needs to be created. When adding these resources to a resource group, the ordering needs to allow for the **`LVM`** resource to be started before and stopped after the **`Filesystem`** resource.

**References**

**What is a Highly Available LVM (HA-LVM) configuration and how do I implement it?**

https://access.redhat.com/knowledge/solutions/3067

Red Hat Enterprise Linux 7 High Availability Add-On Administration

- Section 2.3: Exclusive Activation of a Volume Group in a Cluster

## Guided Exercise

# Using HA-LVM

In this lab, you will create a HA-LVM volume group and logical volume, and then configure a cluster service to use its logical volume.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

## Outcome(s)

You should be able to configure a cluster file system resource to use a highly available logical volume for its file system.

- Reset your **workstation** machine.
- Reset all of your **node*Y*** machines.
- From your **workstation** system, run the command **lab lvm setup**.

```
[student@workstation ~]$ lab lvm setup
```

1. On **nodea**, create an LVM physical volume on the multipath device, **/dev/mapper/mpatha**, which provides redundant access to the iSCSI storage shared by the cluster nodes.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
  Physical volume "/dev/mapper/mpatha" successfully created
```

2. On **nodea**, create an LVM volume group, **clustervg**, composed of the newly created LVM physical volume.

```
[root@nodea ~]# vgcreate clustervg /dev/mapper/mpatha
  Volume group "clustervg" successfully created
```

3. On **nodea**, create a 1 GiB LVM logical volume, **clusterlv**, in the **clustervg** volume group.

```
[root@nodea ~]# lvcreate -L 1G -n clusterlv clustervg
  Logical volume "clusterlv" created.
```

4. On **nodea**, create an XFS file system on the newly created **clusterlv** logical volume.

```
[root@nodea ~]# mkfs -t xfs /dev/clustervg/clusterlv
meta-data=/dev/clustervg/clusterlv isize=256    agcount=8, agsize=32768 blks
         =                       sectsz=512   attr=2, projid32bit=1
```

```
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=262144, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

5. On all three of the nodes, modify LVM configuration so that only the cluster is capable of activating the **clustervg** volume group.

   5.1. Ensure that LVM is configured for local file-based locking with the following setting in **/etc/lvm/lvm.conf**.

```
locking_type = 1
```

   5.2. Define a list of the local volume groups that are not shared by the cluster nodes with the following setting in **/etc/lvm/lvm.conf**. The nodes in our example have no volume groups other than **clustervg**, so the list is empty.

```
volume_list = []
```

6. On each of the three nodes, remake the initramfs for the current kernel version and then the node.

```
[root@nodeY ~]# dracut -H -f /boot/initramfs-$(uname -r).img $(uname -r); reboot
```

7. Once all the nodes have rebooted, verify that each node is online and has started cluster services successfully.

```
[root@nodeY ~]# pcs status
```

8. On **nodea**, create a **LVM** resource, **halvm**, as a member of the resource group, **halvmfs**.

```
[root@nodea ~]# pcs resource create halvm LVM volgrpname=clustervg exclusive=true
 --group halvmfs
```

9. On **nodea**, create a **Filesystem** resource, **xfsfs**, as a member of the resource group, **halvmfs**. This resource should mount the file system on the **clusterlv** logical volume at the **/mnt/** mount point.

```
[root@nodea ~]# pcs resource create xfsfs Filesystem device="/dev/clustervg/
clusterlv" directory="/mnt" fstype="xfs" --group halvmfs
```

10. Verify that the **halvmfs** resource group is working properly on **nodea**.

    10.1. Verify that the resource group started successfully on **nodea**.

```
[root@nodea ~]# pcs status
...
 Resource Group: halvmfs
     halvm (ocf::heartbeat:LVM): Started nodea.private.example.com
     xfsfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com
...
```

10.2. Verify that the **clusterlv** logical volume is active and properly mounted on **nodea**.

```
[root@nodea ~]# lvs
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi-ao---- 1.00g
```

```
[root@nodeY ~]# mount | grep clusterlv
/dev/mapper/clustervg-clusterlv on /mnt type xfs
 (rw,relatime,seclabel,attr2,inode64,noquota)
```

**11.** Test the HA-LVM implementation by failing the **halvmfs** resource group over to another node and validating that the logical volume is active and mounted exclusively on the new node.

```
[root@nodea ~]# pcs cluster standby nodea.private.example.com
[root@nodea ~]# pcs status
...
 Resource Group: halvmfs
     halvm (ocf::heartbeat:LVM): Started nodeb.private.example.com
     xfsfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
...
```

```
[root@nodea ~]# pcs cluster unstandby nodea.private.example.com
```

```
[student@workstation ~]# for x in {a..c}; do echo "===== node${x} ====="; ssh
 root@node${x} 'lvs; echo; mount | grep /mnt'; echo; echo; done
===== nodea =====
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi------- 1.00g




===== nodeb =====
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi-ao---- 1.00g
```

```
/dev/mapper/clustervg-clusterlv on /mnt type xfs
 (rw,relatime,seclabel,attr2,inode64,noquota)


===== nodec =====
  LV        VG        Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
 Convert
  clusterlv clustervg -wi------- 1.00g
```

# Managing Clustered Logical Volumes

## Objectives

After completing this section, students should be able to configure clustered LVM.

## Clustered LVM

Clustered LVM allows the use of regular LVM volume groups and logical volumes on shared storage. In a cluster configured with clustered LVM, a volume group and its logical volumes are accessible to all cluster nodes at the same time. With clustered LVM, administrators can use the management benefits of LVM in conjunction with a shared file system like **GFS2**, for scenarios such as making virtual machine images inside logical volumes available to all cluster nodes.

The active/active configuration of logical volumes in a cluster using clustered LVM is accomplished by using a daemon called clvmd to propagate metadata changes to all cluster nodes. The **`clvmd`** daemon manages clustered volume groups and communicates their metadata changes made on one cluster node to all the remaining nodes in the cluster.

Without the **`clvmd`** daemon, LVM metadata changes made on one cluster node would be unknown to other cluster nodes. Since these metadata define which storage addresses are available for data and file system information, metadata changes not propagated to all cluster nodes can lead to corruption of LVM metadata as well as data residing on LVM physical volumes.

In order to prevent multiple nodes from changing LVM metadata simultaneously, clustered LVM uses *Distributed Lock Manager (DLM)* for lock management. The **`clvmd`** daemon and the **`DLM`** lock manager must be installed prior to configuring clustered LVM. They can be obtained by installing the *lvm2-cluster* and *dlm* packages, respectively. Both packages are available from the *Resilient Storage* repository.

**Warning**

While **`clvmd`** makes it possible for a volume group to be activated concurrently on multiple nodes in a cluster, the file systems contained by the volume group can only be mounted on multiple nodes at the same time if they are cluster-aware file systems, such as **GFS2**. Non-clustered file systems such as ext2, ext3, ext4, and XFS cannot be mounted concurrently on multiple nodes in a cluster, even when the nodes are configured to run **`clvmd`**.

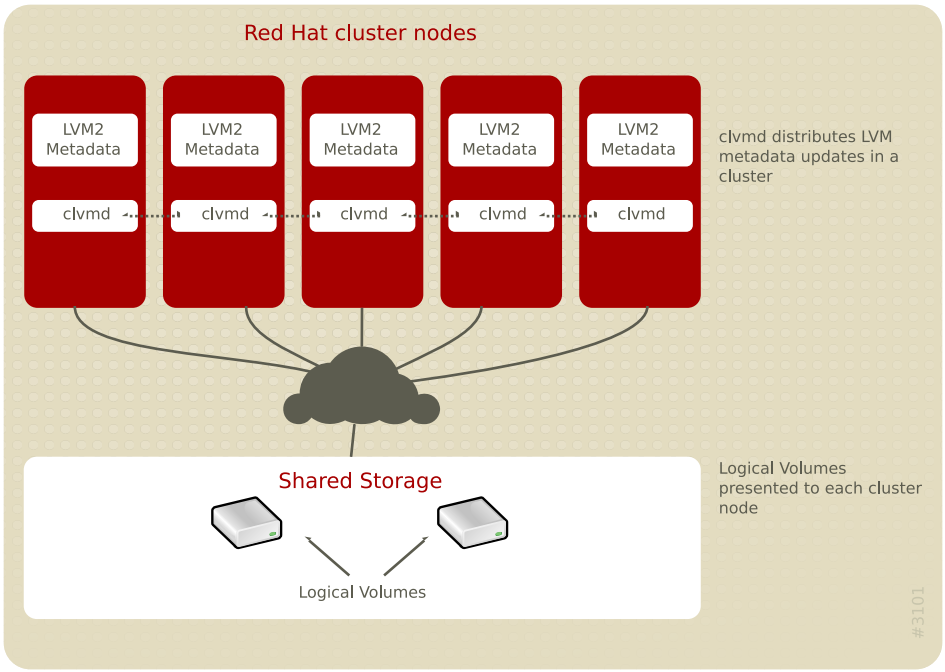



Figure 10.2: An overview of the clustered LVM architecture

# Configuring clustered LVM

To configure clustered LVM, all cluster nodes must be configured to use LVM's built-in clustered locking. This can be done by manually editing **/etc/lvm/lvm.conf** on each node and setting the **locking_type** option to **3**.

```
locking_type = 3
```

To improve performance and automatic activation of volume groups and logical volumes by **udev**, LVM makes use of a metadata cache. By default, LVM manages its metadata centrally using a daemon, *lvmetad*. While the use of **lvmetad** is supported when LVM is configured for local file-based locking (**locking_type = 1**), it is not currently supported for use across cluster nodes. Therefore, when LVM is configured for clustered locking, **lvmetad** must also be disabled on each node with the following setting in **/etc/lvm/lvm.conf**.

```
use_lvmetad = 0
```

When implementing clustered LVM, the preceding LVM configuration changes can be performed manually by editing the **/etc/lvm/lvm.conf** configuration file. Alternatively, both configuration changes can be effected by running the following command.

```
[root@nodeY ~]# lvmconf --enable-cluster
```

After the use of the **lvmetad** metadata cache is disabled, the **lvmetad** service can be disabled on the cluster as well.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
```

Once LVM has been configured for clustered locking, create DLM and clvmd resources in the cluster using the **controld** and **clvm** resource agents, respectively. Since both resources need to run on every node in the cluster, it is necessary to create them as cloned resources.

```
[root@nodeY ~]# pcs resource create mydlm controld op monitor interval=30s on-
fail=fence clone interleave=true ordered=true
```

```
[root@nodeY ~]# pcs resource create myclvmd clvm op monitor interval=30s on-
fail=fence clone interleave=true ordered=true
```

The clvmd resource depends on the DLM resource. An order constraint can be used to enforce the startup of the DLM resource prior to the clvmd resource. In addition, a colocation constraint should be specified so that both resources run together on the same node.

```
[root@nodeY ~]# pcs constraint order start mydlm-clone then myclvmd-clone
```

```
[root@nodeY ~]# pcs constraint colocation add myclvmd-clone with mydlm-clone
```

## Adding logical volumes with CLVM

After LVM clustered locking, DLM, and clvmd cluster resources have been configured, logical volumes can be created with CLVM using standard LVM commands. With CLVM configured, any changes made to a clustered volume group will be propagated to all cluster nodes. Volume groups created will automatically be marked as clustered, which indicates that they are shared with other nodes in the cluster.

**Important**

When making changes to a clustered volume group, **clvmd** requires that the cluster is quorate and that *all* cluster nodes are currently up.

**Important**

LVM snapshots can *not* be used with either clustered LVM or HA-LVM.

**References**

Red Hat Enterprise Linux 7 Logical Volume Manager Administration Guide

- Section 1.4: The Clustered Logical Volume Manager (CLVM)
- Section 3.1: Creating LVM Volumes in a Cluster

**What is the recommended LVM configuration when multiple Red Hat Enterprise Linux cluster nodes are accessing the same shared storage?**
https://access.redhat.com/knowledge/solutions/16664

**How can I use clvmd to manage shared LVM volumes with Pacemaker in a RHEL 6 or 7 High Availability cluster?**
https://access.redhat.com/solutions/530223

**How can I make my highly available resources dependent upon clone resources in RHEL 7 with pacemaker?**
https://access.redhat.com/solutions/885823

**How do I start and manage clvmd and cmirrord in a RHEL 7 pacemaker cluster?**
https://access.redhat.com/solutions/738423

**Do I need an LVM resource managed by the cluster resource manager if I'm using clustered LVM (clvmd) in a RHEL 5, 6, or 7 Resilient Storage cluster?**
https://access.redhat.com/solutions/1306223

**Can I use clvmd to manage shared LVM volumes with Pacemaker in a RHEL 6 or 7 High Availability cluster?**
https://access.redhat.com/solutions/530223

## Guided Exercise

# Configuring Clustered LVM

In this lab, you will create a logical volume in a cluster using the Clustered Logical Volume Manager (CLVM).

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

### Outcome(s)

You should be able to configure a cluster resource to use a clustered LVM logical volume.

- Reset your **workstation** machine.
- Reset all of your **nodeY** machines.
- From your **workstation** system, run the command **lab lvm setup**.

```
[student@workstation ~]$ lab lvm setup
```

1. On each of the three nodes, install the necessary software for CLVM.

   1.1. Install the **dlm** package to implement lock management on the cluster.

```
[root@nodeY ~]# yum install -y dlm
```

   1.2. Install the **lvm2-cluster** package to obtain the clustering extensions to LVM.

```
[root@nodeY ~]# yum install -y lvm2-cluster
```

2. On each of the three nodes, modify LVM configuration so that cluster locking is enabled.

```
[root@nodeY ~]# lvmconf --enable-cluster
```

3. On each of the three nodes, stop the **lvmetad** service to match the modified LVM configuration.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
Warning: Stopping lvm2-lvmetad, but it can still be activated by:
  lvm2-lvmetad.socket
```

4. On **nodea**, create a **dlm** resource using the **controld** resource agent. Create it as a cloned resource so it runs on every cluster node.

```
[root@nodea ~]# pcs resource create dlm controld op monitor interval=30s on-
fail=fence clone interleave=true ordered=true
```

5. On **nodea**, create a **clvmd** resource using the **clvm** resource agent. Create it as a cloned resource so it runs on every cluster node.

```
[root@nodea ~]# pcs resource create clvmd clvm op monitor interval=30s on-
fail=fence clone interleave=true ordered=true
```

6. On **nodea** configure the **dlm** and **clvmd** resources so that **dlm** starts prior to **clvmd** and that both resources can only run on the same node.

```
[root@nodea ~]# pcs constraint order start dlm-clone then clvmd-clone
Adding dlm-clone clvmd-clone (kind: Mandatory) (Options: first-action=start then-
action=start)
```

```
[root@nodea ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

7. On **nodea**, create a 1 GiB logical volume on the multipath device, **/dev/mapper/mpatha**, which provides redundant access to the iSCSI storage shared by the cluster nodes.

   7.1. On **nodea**, create an LVM physical volume on **/dev/mapper/mpatha**.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
      Physical volume "/dev/mapper/mpatha" successfully created
```

   7.2. On **nodea**, create an LVM volume group, **myvg**, composed of the newly created LVM physical volume.

```
[root@nodea ~]# vgcreate myvg /dev/mapper/mpatha
      Volume group "myvg" successfully created
```

   7.3. On **nodea**, create a 1 GiB LVM logical volume, **mylv**, in the **myvg** volume group.

```
[root@nodea ~]# lvcreate -L 1G -n mylv myvg
      Logical volume "mylv" created.
```

## Lab

# Configuring Logical Volumes for Cluster File Systems

### Performance Checklist

In this lab, you will configure a clustered logical volume using the Clustered Logical Volume Manager (CLVM).

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

### Outcome(s)

You should be able to configure a cluster file system resource to use a clustered logical volume.

- Reset your **workstation** machine.
- Reset all of your **node*Y*** machines.
- From your **workstation** system, run the command **lab lvm setup**.

```
[student@workstation ~]$ lab lvm setup
```

You have received a request to create a 1 GiB LVM logical volume for share use across your cluster. This logical volume will be used in an active/active configuration in the cluster. Create the logical volume with the name **clusterlv** in a volume group called **clustervg**. Once created, add the logical volume as a new resource in your cluster called **clusteredlvm**.

1. On each of the three nodes, install the necessary software for Clustered LVM.
2. On each of the three nodes, modify LVM configuration so that cluster locking is enabled.
3. On each of the three nodes, stop the **lvmetad** service to match the modified LVM configuration.
4. On **nodea**, create a **dlm** resource using the **controld** resource agent. Create it as a cloned resource so it runs on every cluster node.
5. On **nodea**, create a **clvmd** resource using the **clvm** resource agent. Create it as a cloned resource so it runs on every cluster node.
6. On **nodea** configure the **dlm** and **clvmd** resources so that **dlm** starts prior to **clvmd** and that both resources can only run on the same node.
7. On **nodea**, create a 1 GiB logical volume on the multipath device, **/dev/mapper/mpatha**, which provides redundant access to the iSCSI storage shared by the cluster nodes.

## Solution

# Configuring Logical Volumes for Cluster File Systems

## Performance Checklist

In this lab, you will configure a clustered logical volume using the Clustered Logical Volume Manager (CLVM).

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, and **workstation** |

## Outcome(s)

You should be able to configure a cluster file system resource to use a clustered logical volume.

- Reset your **workstation** machine.
- Reset all of your **node*Y*** machines.
- From your **workstation** system, run the command **lab lvm setup**.

```
[student@workstation ~]$ lab lvm setup
```

You have received a request to create a 1 GiB LVM logical volume for share use across your cluster. This logical volume will be used in an active/active configuration in the cluster. Create the logical volume with the name **clusterlv** in a volume group called **clustervg**. Once created, add the logical volume as a new resource in your cluster called **clusteredlvm**.

1. On each of the three nodes, install the necessary software for Clustered LVM.

   1.1. Install the **dlm** package to implement lock management on the cluster.

```
[root@nodeY ~]# yum install -y dlm
```

   1.2. Install the **lvm2-cluster** package to obtain the clustering extensions to LVM.

```
[root@nodeY ~]# yum install -y lvm2-cluster
```

2. On each of the three nodes, modify LVM configuration so that cluster locking is enabled.

```
[root@nodeY ~]# lvmconf --enable-cluster
```

3. On each of the three nodes, stop the **lvmetad** service to match the modified LVM configuration.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
Warning: Stopping lvm2-lvmetad, but it can still be activated by:
  lvm2-lvmetad.socket
```

4. On **nodea**, create a **dlm** resource using the **controld** resource agent. Create it as a cloned resource so it runs on every cluster node.

```
[root@nodea ~]# pcs resource create dlm controld op monitor interval=30s on-
fail=fence clone interleave=true ordered=true
```

5. On **nodea**, create a **clvmd** resource using the **clvm** resource agent. Create it as a cloned resource so it runs on every cluster node.

```
[root@nodea ~]# pcs resource create clvmd clvm op monitor interval=30s on-
fail=fence clone interleave=true ordered=true
```

6. On **nodea** configure the **dlm** and **clvmd** resources so that **dlm** starts prior to **clvmd** and that both resources can only run on the same node.

```
[root@nodea ~]# pcs constraint order start dlm-clone then clvmd-clone
Adding dlm-clone clvmd-clone (kind: Mandatory) (Options: first-action=start then-
action=start)
```

```
[root@nodea ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

7. On **nodea**, create a 1 GiB logical volume on the multipath device, **/dev/mapper/mpatha**, which provides redundant access to the iSCSI storage shared by the cluster nodes.

   7.1. On **nodea**, create an LVM physical volume on **/dev/mapper/mpatha**.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
      Physical volume "/dev/mapper/mpatha" successfully created
```

   7.2. On **nodea**, create an LVM volume group, **clustervg**, composed of the newly created LVM physical volume.

```
[root@nodea ~]# vgcreate clustervg /dev/mapper/mpatha
      Volume group "clustervg" successfully created
```

   7.3. On **nodea**, create a 1 GiB LVM logical volume, **clusterlv**, in the **clustervg** volume group.

```
[root@nodea ~]# lvcreate -L 1G -n clusterlv clustervg
      Logical volume "clusterlv" created.
```

# Summary

In this chapter, you learned:

- With the LVM **`archive`** option enabled, changes made inadvertently to volume groups can be reverted with the **`vgcfgrestore`** command.
- LVM can be used in a cluster in an active/passive configuration using HA-LVM.
- In HA-LVM cluster implementations, LVM must be configured so that control of volume groups used in the cluster is in the sole control of the cluster.
- LVM can be used in a cluster in an active/active configuration using CLVM.
- Clustered LVM requires the installation of the *lvm2-cluster* and *lvm* packages.
- LVM needs to be configured for clustered locking and **`lvmetad`** disablement for CLVM implementations.

Chapter 11

# Providing Storage with the GFS2 Cluster File System

**Goal**

Configure and manage a GFS2 file system as a cluster resource.

**Objectives**

- Create and configure a GFS2 file system.
- Manage GFS2 mount options and journals.
- Configure a GFS2 file system as a cluster resource.

**Sections**

- GFS2 Concepts (and Quiz)
- Creating a GFS2 Formatted Cluster File System (and Practice)
- Managing a GFS2 File System (and Practice)
- Managing a GFS2 Resource in the Cluster (and Practice)

**Lab**

- Providing Storage with the GFS2 Cluster File System

# GFS2 Concepts

## Objectives

After completing this section, students should be able to explain the concepts and features of GFS2.

## Global File System 2 (GFS2)

*Global File System 2* (GFS2) is a cluster file system interfacing directly with the kernel VFS layer. This means that the same file system can be mounted and used by multiple cluster nodes simultaneously, while still providing a full regular file system, including features such as support for POSIX ACLs, extended attributes, and quotas.

To accomplish this, every node accessing a GFS2 file system uses the cluster infrastructure provided by Corosync and Pacemaker to provide services such as fencing and locking. Each cluster node mounting a GFS2 file system will use a separate journal. If a node fails, one of the other nodes in the cluster will replay the journal for the failed node after the failed node has been fenced. To prevent race conditions between two nodes when accessing the file system, GFS2 uses the *Distributed Lock Manager* (DLM) to coordinate locks on files and directories.

**Important**

Red Hat does not support the use of GFS2 as a single-node file system, or cluster deployments of more than 16 nodes.

If a high-performance, scalable, server file system is needed for use by a single node, Red Hat recommends the use of the default XFS file system, or as an alternative, ext4.

When running in a pure 64-bit environment, a GFS2 file system can theoretically scale up to 8 EiB. However, the maximum GFS2 file system size currently supported by Red Hat is 100 TiB.

For most deployments, having multiple smaller file systems makes more sense than a single large file system. On a larger file system, running a **`fsck.gfs2`** command will take longer, and use more memory. A full restore of a file system from backup will also take longer.

**Important**

Red Hat currently only supports GFS2 file systems that have been created on a CLVM cluster logical volume.https://www.redhat.com/apps/store/add-ons/?tindex=tcontent4

Red Hat offers the GFS2 file system and CLVM as part of the *Resilient Storage Add-on* for Red Hat Enterprise Linux Server. The Resilient Storage Add-on includes the High-Availability Add-on as part of the subscription.

## SELinux and GFS2

GFS2 supports extended attributes (xattrs) and can store file labels used by Security Enhanced Linux (SELinux) just like XFS and ext4. However, using SELinux with GFS2 is complicated by the fact that updates to SELinux file labels on a GFS2 file system are currently not cluster coherent. This means that if one node changes the SELinux context of a file on a GFS2 file system, other cluster nodes that have that file system mounted may continue using the old context on that file indefinitely. This is somewhat tricky to resolve, and the issue is currently being tracked at bugzilla.redhat.com as bug #437984.

Due to this issue, it may be beneficial to not write SELinux labels to individual files on a GFS2 file system. Due to the way in which GFS2 stores file xattrs, updating those labels may result in a performance penalty specific to GFS2. This is likely to be most noticeable with workloads that involve many small files. One approach that may help to work around this while still using SELinux is to take advantage of the **`mount`** option **`context=`** to set the context of all files on that GFS2 file system to a particular value when mounted. This will avoid xattr lookups and writes. Alternatively, other steps might be taken to ensure that file labels are not changed on the file system while it is mounted by multiple nodes.

If SELinux labels are not changed on files, then SELinux in enforcing mode will otherwise function normally with GFS2 file systems, and Red Hat does test GFS2 file systems and the entire cluster stack with SELinux enforcing on.

**Note**

At the time of writing, the official *Red Hat Global File System 2 Guide* at https://access.redhat.com/documentation/, in section 2.5.4 "SELinux: Avoid SELinux on GFS2", seems to state that SELinux *must* be turned off on GFS2 file systems.

It is the authors' understanding based on discussions with Red Hat GFS2 engineers that this statement is overly strong, and SELinux may be used in enforcing mode with an understanding of the discussed details.

In any event, it is recommend that customers using GFS2 complete an architectural review as discussed in the next note to discuss a planned cluster configuration with Red Hat Support, no matter what operating mode of SELinux will be used.

**Important**

Red Hat recommends that customers using GFS2 in a cluster deployment contact Red Hat for an architectural review to ensure that the planned cluster configuration is supportable. GFS2 is fully supported, but it is not suitable for all environments and workloads. It is best to consult with Red Hat about an architecture and intended use of the file system before proceeding.

For more information, see the following Red Hat Knowledgebase articles mentioned in the References section that follows:

- *Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Recommended Practices*
- *What information is required for an Architecture Review of Red Hat Enterprise Linux High Availability and/or Resilient Storage cluster?*

**References**

**gfs2**(5), **mount**(8), and **attr**(5) man pages

**Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Recommended Practices**
https://access.redhat.com/articles/40051

**What information is required for an Architecture Review of Red Hat Enterprise Linux High Availability and/or Resilient Storage cluster?**
https://access.redhat.com/solutions/125153

**What subscriptions are required for RHEL High Availability and/or Resilient Storage clusters?**
https://access.redhat.com/solutions/55242

**On which architectures are the High Availability and Resilient Storage Add-Ons supported?**
https://access.redhat.com/solutions/20876

**Add-Ons for Red Hat Enterprise Linux Server**
https://www.redhat.com/apps/store/add-ons/?tindex=tcontent4

**Bug 437984 - RFE: GFS/GFS2 w/SELinux xattr issue**
https://bugzilla.redhat.com/show_bug.cgi?id=437984

Additional information on GFS2 may be available in the *Red Hat Global File System 2 Guide* for Red Hat Enterprise Linux 7, which can be found at
https://access.redhat.com/documentation/

## Quiz

# GFS2 Concepts

Choose the correct answer to the following questions:

1. **How many nodes can mount and access a GFS2 file system simultaneously, assuming enough journals have been created on the file system?**
   a. 4
   b. 8
   c. 16
   d. 32

2. **You have a five-node cluster with a GFS2 file system. No more than three nodes will ever mount the GFS2 simultaneously. How many journals do you need to configure in the GFS2 file system at a minimum?**
   a. 3
   b. 4
   c. 5
   d. 6

3. **Which of the following features is not currently supported with GFS2?**
   a. Extended Attributes
   b. Using GFS2 on a block device that is not a CLVM logical volume
   c. POSIX ACLs
   d. Symbolic Links

4. **For a Red Hat Enterprise Linux 7-based cluster, the maximum GFS2 file system size currently supported by Red Hat is:**
   a. 16 PiB
   b. 100 TiB
   c. 8 EiB

5. **Which of the following Add-ons provides support for GFS2?**
   a. High Availability Add-on
   b. Resilient Storage Add-on
   c. Scalable File System Add-on

## Solution

# GFS2 Concepts

Choose the correct answer to the following questions:

1. **How many nodes can mount and access a GFS2 file system simultaneously, assuming enough journals have been created on the file system?**
   a. 4
   b. 8
   c. 16
   d. 32

2. **You have a five-node cluster with a GFS2 file system. No more than three nodes will ever mount the GFS2 simultaneously. How many journals do you need to configure in the GFS2 file system at a minimum?**
   a. 3
   b. 4
   c. 5
   d. 6

3. **Which of the following features is not currently supported with GFS2?**
   a. Extended Attributes
   b. Using GFS2 on a block device that is not a CLVM logical volume
   c. POSIX ACLs
   d. Symbolic Links

4. **For a Red Hat Enterprise Linux 7-based cluster, the maximum GFS2 file system size currently supported by Red Hat is:**
   a. 16 PiB
   b. 100 TiB
   c. 8 EiB

5. **Which of the following Add-ons provides support for GFS2?**
   a. High Availability Add-on
   b. Resilient Storage Add-on
   c. Scalable File System Add-on

# Creating a GFS2 Formatted Cluster File System

## Objectives

After completing this section, students should be able to configure a GFS2 file system.

## Global File System 2 (GFS2) preparations

Before creating a GFS2 file system:

- Make sure that the *gfs2-utils* package is installed on all of the cluster nodes.
- Make sure that there is a clustered logical volume, accessible from all cluster nodes, on which to create the GFS2 file system.
- Make sure that the clocks between all cluster nodes are synchronized (preferably with ntp or chronyd).

Furthermore, a few pieces of information are needed when creating a GFS2 file system:

- The name of the cluster that will use the file system.
- The number of nodes that will be accessing the file system simultaneously, including any that might be added in the future.
- The name to use for the new file system.
- The device node for the clustered logical volume that will be used to store the file system.

## Creating a GFS2 file system

Once all the prerequisites are in place, use the **`mkfs.gfs2`** command from any one of the nodes to create a GFS2 file system. The most common options to **`mkfs.gfs2`** are listed in the following table.

| **`mkfs.gfs2` options** | |
|---|---|
| **`-t <lock_table_name>`** | The name of the locking table (not used with **`lock_nolock`**). For **`lock_dlm`** this should be *<clustername>*:*<fs_name>*. Only nodes that are a member of *<clustername>* will be allowed to mount this file system. *<fs_name>* should be a unique name to distinguish this file system between one and sixteen characters in length. |
| **`-j <number_of_journals>`** | The number of journals to create initially (more journals can be added later). Each node accessing a file system simultaneously will need one journal. This option will default to one journal if omitted. |
| **`-J <journal_size>`** | The size of the journals to be created in MiB. Journals will need to be at least 8 MiB. The journal size will default to 128 MiB if no size is given. |

For example, to create a GFS2 file system called **examplegfs2**, belonging to the **examplecluster** cluster, with three 64 MiB journals, on the **/dev/clusteredvg/lv_gfs** clustered logical volume, use the following command:

```
[root@nodea ~]# mkfs.gfs2 -t examplecluster:examplegfs2 -j 3 -J 64 /dev/
clusteredvg/lv_gfs
```

**Note**

When running the **mkfs.gfs2** command to create a GFS2 file system, the size of the journals may be specified. If a size is not specified, it will default to 128 MB, which should be optimal for most applications.

It is generally recommended to use the default journal size of 128 MB. If the file system is very small (for example, 5GB), having a 128 MB journal might be impractical. If you have a larger file system and can afford the space, using 256 MB journals might improve performance.

For more information, see the GFS2 documentation at https://access.redhat.com/documentation/.

## Mounting a GFS2 file system

Before mounting a **GFS2** file system, the file system must exist, the volume where the file system exists must be activated, and the supporting clustering and locking systems must be started.

For testing purposes, the **GFS2** file system can be mounted in the same way as any other typical Linux file system. For normal production operation, the **GFS2** file system should be mounted by configuring it as a cluster resource.

**Important**

GFS2 file systems that have been mounted manually rather than automatically through Pacemaker will not be known to the system when file systems are unmounted at system shutdown. As a result, the GFS2 script will not unmount the GFS2 file system. After the GFS2 shutdown script is run, the standard shutdown process kills off all remaining user processes, including the cluster infrastructure, and tries to unmount the file system. This unmount will fail without the cluster infrastructure and the system will hang.

To prevent the system from hanging when the GFS2 file systems are unmounted, do one of the following:

- Always use Pacemaker to manage the GFS2 file system as a cluster resource.
- If a GFS2 file system has been mounted manually with the **mount** command, be sure to unmount the file system manually with the **umount** command before rebooting or shutting down the system.

If the file system hangs while it is being unmounted during system shutdown under these circumstances, perform a hardware reboot. It is unlikely that any data will be lost since the file system is synced earlier in the shutdown process.

The basics of mounting a GFS2 file system are identical to that of any other regular file system:

```
[root@nodea ~]# mount [-t gfs2] [-o <mount_options>] <blockdevice> <mountpoint>
```

To use POSIX ACLs (with **getfacl** and **setfacl**) on a GFS2 file system, it will need to be mounted with the **acl** mount option.

A line for a GFS2 file system in **/etc/fstab** might look something like this:

```
/dev/clustererdvg/lv_gfs2   /mountpoint   gfs2  acl 0 0
```

**Warning**

It is *critical* that the last column of any entry in **/etc/fstab** (the **fs_passno** column) is set to **0** so that **fsck.gfs2** is not run on boot.

It is possible that the GFS2 file system is already mounted on another cluster node at boot. Running an **fsck** on a GFS2 file system that is currently mounted (even on another node) can lead to serious damage to the file system and data loss.

For further discussion, see the **fsck.gfs2**(8) man page.

In general, it is not a recommended practice to add GFS2 file systems to **/etc/fstab**. They should, by preference, be mounted at boot as a cluster resource by Pacemaker. If mounted for a testing purpose outside Pacemaker, it is better to manually mount the GFS2 file system and then manually unmount it from all nodes when no longer needed, prior to shutdown of the cluster nodes.

**References**

**mkfs.gfs2**(8), **mount**(8), **getfacl**(1), **setfacl**(1), **fstab**(5), and **fsck.gfs2**(8) man pages

Additional information on GFS2 may be available in the "Getting Started" chapter of the *Red Hat Global File System 2 Guide* for Red Hat Enterprise Linux 7, which can be found at
https://access.redhat.com/documentation/

## Guided Exercise

# Creating a GFS2 Formatted Cluster File System

In this lab, you will create a 2 GiB GFS2 file system with two journals and mount it on **nodea** and **nodeb** simultaneously.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to configure a GFS2 file system that mounts simultaneously on **nodea** and **nodeb**.

- Reset your **nodea**, **nodeb**, **nodec**, and **workstation** systems.
- From **workstation**, run the command **lab dlm-clvm setup**.

1. Prepare your **nodea**, **nodeb**, and **nodec** machines for DLM and **clvmd**.

1.1. On **nodea**, **nodeb**, and **nodec**, install the *gfs2-utils* and *lvm2-cluster* packages.

```
[root@nodeY ~]# yum -y install gfs2-utils lvm2-cluster
```

1.2. Configure **no-quorum-policy=freeze** when GFS2 is in use.

```
[root@nodea ~]# pcs property set no-quorum-policy=freeze
```

1.3. Configure a DLM resource. This is a required dependency for **clvmd** and GFS2.

```
[root@nodea ~]# pcs resource create dlm ocf:pacemaker:controld op monitor
 interval=30s on-fail=fence clone interleave=true ordered=true
```

1.4. On **nodea**, **nodeb**, and **nodec**, set the **locking_type** parameter in the **/etc/lvm/lvm.conf** file to **3** (enable cluster-wide locking for LVM).

```
[root@nodeY ~]# lvmconf --enable-cluster
```

1.5. On each of the three nodes, stop the **lvmetad** service to match the modified LVM configuration.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
  Warning: Stopping lvm2-lvmetad, but it can still be activated by:
     lvm2-lvmetad.socket
```

1.6. Configure **clvmd** as a cluster resource.

```
[root@nodea ~]# pcs resource create clvmd ocf:heartbeat:clvm op monitor
 interval=30s on-fail=fence clone interleave=true ordered=true
```

1.7. Configure **clvmd** and DLM dependency and start up order. **clvmd** must start after DLM and must run on the same node as DLM.

```
[root@nodea ~]# pcs constraint order start dlm-clone then clvmd-clone
[root@nodea ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

**2.** Create a CLVM logical volume for your GFS2 file system.

2.1. Create a LVM physical volume on **/dev/mapper/mpatha**.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
  Physical volume "/dev/mapper/mpatha" successfully created
```

2.2. Create a clustered volume group with the name **cluster_vg**.

```
[root@nodea ~]# vgcreate -Ay -cy cluster_vg /dev/mapper/mpatha
      Volume group "clustervg" successfully created
```

2.3. Create a 2 GiB logical volume as part of **cluster_vg** with the name **gfsdata**.

```
[root@nodea ~]# lvcreate -L 2G -n gfsdata cluster_vg
      Logical volume "gfsdata" created.
```

**3.** Create a GFS2 file system on your new CLVM logical volume. Test this file system to verify that it can be mounted on both **nodea** and **nodeb**.

3.1. On your new logical volume, create a GFS2 file system named **gfsdata** with two journals.

```
[root@nodea ~]# mkfs.gfs2 -t clusterX:gfsdata -j2 /dev/mapper/cluster_vg-gfsdata
```

3.2. On **nodea**, **nodeb**, and **nodec**, create the mount point **/gfsdata**.

```
[root@nodeY ~]# mkdir /gfsdata
```

3.3. On both **nodea** and **nodeb**, mount the GFS2 file system.

```
[root@nodea ~]# mount -t gfs2 /dev/mapper/cluster_vg-gfsdata /gfsdata
```

3.4. Test your new file system by writing a file to it from **nodea**, then reading that file back from **nodeb**.

```
[root@nodea ~]# echo "Sharing is caring" >> /gfsdata/shared_file
```

```
[root@nodeb ~]# cat /gfsdata/shared_file
Sharing is caring
```

# Managing a GFS2 File System

## Objectives

After completing this section, students should be able to extend a GFS2 file system and add additional journals.

## Adding journals to a GFS2 file system

When trying to mount a GFS2 file system from more nodes than the file system has journals, the error **`Too many nodes mounting file system, no free journals`** will occur. To find out how many journals a GFS2 file system has, use the following command:

```
[root@nodea ~]# gfs2_edit -p jindex /dev/mapper/cluster_vg-gfsdata |grep journal
 3/3 [fc7745eb] 1/20 (0x1/0x14): File journal0
 4/4 [8b70757d] 2/32859 (0x2/0x805b): File journal1
```

**Note**

The **`gfs2_tool`** command from earlier versions of the product is not supported in Red Hat Enterprise Linux 7.

To add more journals, use the **`gfs2_jadd`** command. This command takes two options, **`-j <new_journals>`** and **`-J <journalsize_in_MiB>`**, and one argument: a block device containing a GFS2 file system or the mount point of a mounted GFS2 file system.

If no journal size is specified, the default size of 128 MiB will be used. Journals are created by converting free file system space into a journal, so make sure that the file system has enough free space to accommodate the new journals you wish to create.

In the next example, two 64 MiB journals will be added to the GFS2 file system mounted at **`/gfs`**:

```
[root@nodea ~]# gfs2_jadd -j 2 -J 64 /gfs
Filesystem:       /gfs
Old Journals:     3
New Journals:     5
```

## Growing a GFS2 file system

A GFS2 file system can be grown to take advantage of any extra space that might have been added to the clustered logical volume on which the file system lives. Growing a GFS2 file system can be done online, although it is recommended to perform a test run first. To grow a GFS2 file system, use the command **`gfs2_grow <mountpoint|device_node>`**. Adding the **`-T`** option will enable test mode. Test mode does everything a normal grow operation would, except actually committing changes to disk.

As an example, assume that there is a GFS2 file system on the clustered logical volume **/dev/cluster_vg/cluster_lv**, mounted on **/gfs**. The logical volume will first be grown by 10 GiB, and then grow the GFS2 file system to use the newly available space.

```
[root@nodea ~]# lvextend -L +10G /dev/cluster_vg/cluster_lv
[root@nodea ~]# gfs2_grow -T /gfs
(Test mode--File system will not be changed)
...
The file system grew by 10 GB.
gfs2_grow complete.
[root@nodea ~]# gfs2_grow /gfs
```

## Repairing a GFS2 file system

Under normal circumstances, a GFS2 file system should not require a manual repair. When a node fails, the journal for that node will be replaced by another node after the failed node has been fenced, thus ensuring file system integrity.

If for some reason the system is left with a dirty or corrupted GFS2 file system, a normal **fsck** command can be run on it. The **fsck** command will recognize the file system as being a GFS2 file system, and call out to the **fsck.gfs2** command to do the actual checking. The **fsck.gfs2** command can be run directly as well.

**Important**

The **fsck.gfs2** command must be run only on a file system that is unmounted from all nodes. When the file system is being managed as a Pacemaker cluster resource, disable the file system resource, which unmounts the file system. After running the **fsck.gfs2** command, enable the file system resource again. The timeout value specified with the **--wait** option of the **pcs resource disable** command indicates a value in seconds. To ensure that the **fsck.gfs2** command does not run on a GFS2 file system at boot time, set the **run_fsck** parameter of the options argument when creating the GFS2 file system resource in a cluster. Specifying **run_fsck=no** will indicate that the node should not run the **fsck** command on resource activation.

**Note**

Note that the **fsck.gfs2** command differs from some earlier releases of **gfs2_fsck** in the following ways: Pressing **Ctrl**+**C** while running the **fsck.gfs2** command interrupts processing and displays a prompt asking whether to abort the command, skip the rest of the current pass, or continue processing. Increase the level of verbosity by using the **-v** option. Adding a second **-v** option increases the verbosity level again. Decrease the level of verbosity by using the **-q** option. Adding a second **-q** option decreases the level again. The **-n** option opens a file system as read-only, and answers no to any queries automatically. The option provides a way of trying the command to reveal errors without actually allowing the **fsck.gfs2** command to take effect. Refer to the **fsck.gfs2** man page for additional information about other command options.

# Making superblock changes

If moving a GFS2 file system to a different cluster is desired, or renaming the cluster is needed, the locking table name in your GFS2 superblock will need to be updated.

**Warning**
Before making any changes to a GFS2 superblock, make sure that the file system is not currently mounted on *any* node.

To request information from a GFS2 superblock, use the command **`tunegfs2 -l <devicenode>`**. The command **`tunegfs2`** can be used to modify the locking table and locking protocol as well.

To change one of these options use the following command:

```
[root@nodea ~]# tunegfs2 -o locktable=<filesystem name> <device>
```

For example, to change the cluster name for a GFS2 file system called **`examplegfs`** that lives on the clustered logical volume **`/dev/cluster_vg/cluster_lv`** to **`newcluster`**, call this command like this:

```
[root@nodea ~]#  tunegfs2 -o locktable=cluster0:newcluster /dev/cluster_vg/
cluster_lv;tunegfs2 -l /dev/cluster_vg/cluster_lv
tunegfs2 (Jan 16 2015 10:53:28)
File system volume name: cluster0:newcluster
File system UUID: c8fd0a3c-1ad8-bc36-fca0-29eb1c7f2c28
File system magic number: 0x1161970
Block size: 4096
Block shift: 12
Root inode: 99316
Master inode: 32854
Lock protocol: lock_dlm
Lock table: cluster0:newcluster
```

**References**
**`fsck.gfs2`**(8), **`gfs2_edit`**(8), **`gfs2_grow`**(8), **`gfs2_jadd`**(8), and **`lvextend`**(8) man pages

Red Hat Global File System 2 Guide

- Section 4.2: Growing a Filesystem
- Section 4.7: Adding Journals to a Filesystem
- Section 4.11: Repairing a File System

## Guided Exercise

# Managing a GFS2 File System

In this lab, you will extend an existing GFS2 file system with more space and additional journals.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

A GFS2 file system with four journals and can be mounted on all cluster nodes at the same time.

- Ensure you still have a working three-node cluster, with clustered LVM and a 2 GiB GFS2 file system with two journals mounted on **/gfsdata** on both **nodea** and **nodeb**.
- In this exercise, you will first add two journals to your GFS2 file system, then extend the logical volume that houses the file system, as well as the file system itself, by 512 MiB.

1. On **nodec**, attempt to manually mount the GFS2 file system from **/dev/mapper/cluster_vg/gfsdata** on **/gfsdata**. Did this work? If not, why not?

```
[root@nodec ~]# mount /dev/mapper/cluster_vg-gfsdata /gfsdata

Too many nodes mounting file system, no free journals
```

2. From **nodea**, examine the number and size of the journals on your GFS2 file system, as well as the free space on your GFS2 file system.

```
[root@nodea ~]# gfs2_edit -p jindex /dev/mapper/cluster_vg-gfsdata |grep journal
3/3 [fc7745eb] 1/20 (0x1/0x14): File journal0
4/4 [8b70757d] 2/32859 (0x2/0x805b): File journal1

[root@nodea ~]# df -h /gfsdata
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/cluster_vg-gfsdata  2.0G  259M  1.8G  13% /gfsdata
```

3. To be ready for any future expansion of your cluster, and to accommodate **nodec**, add two journals to your GFS2 file system.

```
[root@nodea ~]# gfs2_jadd -j2 /gfsdata
Filesystem: /gfsdata
Old Journals: 2
New Journals: 4
```

4. From **nodea**, examine the journals and the free space on your GFS2 file system.

```
[root@nodea ~]# gfs2_edit -p jindex /dev/mapper/cluster_vg-gfsdata |grep journal
3/3 [fc7745eb] 1/20 (0x1/0x14): File journal0
4/4 [8b70757d] 2/32859 (0x2/0x805b): File journal1
5/5 [127924c7] 5/66486 (0x5/0x103b6): File journal2
6/6 [657e1451] 9/99583 (0x9/0x184ff): File journal3

[root@nodea ~]# df -h /gfsdata
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/cluster_vg-gfsdata  2.0G  518M  1.5G  26% /gfsdata
```

5. Extend the **/dev/mapper/cluster_vg-gfsdata** logical volume by 512 MiB.

```
[root@nodea ~]# lvextend -L +512M /dev/mapper/cluster_vg-gfsdata
Size of logical volume cluster_vg/gfsdata changed from 2.00 GiB (512 extents) to
 2.50 GiB (640 extents). Logical volume gfsdata successfully resized
```

6. Grow your GFS2 file system into the newly available space.

```
[root@nodea ~]# gfs2_grow /gfsdata
FS: Mount point:             /gfsdata
FS: Device:                  /dev/mapper/cluster_vg-gfsdata
FS: Size:                    524288 (0x80000)
FS: New resource group size: 131072 (0x20000)
DEV: Length:                 655360 (0xa0000)
The file system will grow by 512MB.
gfs2_grow complete.
```

7. Examine the free space on your GFS2 file system.

```
[root@nodea ~]# df -h /gfsdata
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/cluster_vg-gfsdata  2.5G  518M  2.0G  21% /gfsdata
```

8. Mount the GFS2 file system on **nodec**.

```
[root@nodec ~]# mount -t gfs2 /dev/mapper/cluster_vg-gfsdata /gfsdata
```

# Managing a GFS2 Resource in the Cluster

## Objectives

After completing this section, students should be able to use a GFS2 file system as a cluster resource.

## GFS2 cluster resources

Global File System 2 (GFS2) file systems should not be mounted automatically at boot through configuration of **`/etc/fstab`**. Instead, a Pacemaker resource group should be set up that automatically mounts the GFS2 resources on appropriate nodes when the supporting cluster infrastructure starts up.

Using a Pacemaker cluster file system resource to manage GFS2 file systems allows Pacemaker to manage and control the GFS2 file system resource in the same way as other resources. It also ensures that the file system is mounted and unmounted correctly and cleanly.

This section will outline the steps required to set up a cluster that includes a GFS2 file system.

## Review: Creating a GFS2 file system

The following procedure reviews the steps necessary to create a GFS2 file system on a CLVM logical volume and to properly prepare the cluster for the Pacemaker cluster resource that will manage it.

1. Set the global Pacemaker parameter **`no-quorum-policy`** to **`freeze`**.

```
[root@nodeY ~]# pcs property set no-quorum-policy=freeze
```

**Important**

By default, the property **`no-quorum-policy`** is set to **`stop`**, which will immediately stop all resources in the cluster if quorum is lost. This is normally the best option. However, if the property is set to **`stop`**, the mounted GFS2 file systems and applications using them cannot use the cluster infrastructure to correctly stop.

In this case, any attempts to stop these resources without quorum will fail, which will ultimately result in the entire cluster being fenced every time quorum is lost.

By setting **`no-quorum-policy=freeze`**, when quorum is lost the cluster nodes will do nothing until quorum is regained.

2. Set up a DLM resource. This is a required dependency for **`clvmd`** and GFS2 to manage cluster locking.

```
[root@nodeY ~]# pcs resource create dlm ocf:pacemaker:controld op monitor \
> interval=30s on-fail=fence clone interleave=true ordered=true
```

3. Execute the following command in each node of the cluster to enable clustered locking. This command sets the **locking_type** parameter in the **/etc/lvm/lvm.conf** file to **3**.

```
[root@nodeY ~]# /sbin/lvmconf --enable-cluster
```

4. Set up **clvmd** as a cluster resource.

```
[root@nodeY ~]# pcs resource create clvmd ocf:heartbeat:clvm op monitor \
> interval=30s on-fail=fence clone interleave=true ordered=true
```

5. Set up **clvmd** and DLM dependency and startup order. **clvmd** must start after DLM and must run on the same node as DLM.

```
[root@nodeY ~]# pcs constraint order start dlm-clone then clvmd-clone
[root@nodeY ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

6. Create the clustered LV and format the volume with a GFS2 file system. Ensure that enough journals are created for each of the nodes in the cluster.

```
[root@nodeY ~]# pvcreate /dev/sdb
[root@nodeY ~]# vgcreate -Ay -cy cluster_vg /dev/vdb
[root@nodeY ~]# lvcreate -L5G -n cluster_lv cluster_vg
[root@nodeY ~]# mkfs.gfs2 -j2 -p lock_dlm -t rhel7-demo:gfs2-demo \
> /dev/cluster_vg/cluster_lv
```

## Configuring a GFS2 Pacemaker cluster resource

Once the supporting cluster infrastructure and a CLVM logical volume formatted with a GFS2 file system is available, the Pacemaker cluster file system resource can be configured.

The **Filesystem** resource is used to configure GFS2 resources. Detailed information about the parameters this resource takes is available by running the command **pcs resource describe Filesystem**. Key information needed will be the CLVM **device** that is formatted with the file system, and the **directory** that is the planned mount point for the file system.

Whenever it is desirable to have a copy of a resource to run on each node, resource **clones** can be used. Whenever a resource is cloned, a copy of that resource will be started on every node in the cluster. Cloning the **GFS2** file system resource enables the resource to run on every node in the cluster and mount the file system, provided a journal was created for every node.

Mount options can be specified as part of the resource configuration with **options=*options***.

**Note**

Two mount options that can have a significant effect on the performance of GFS2 are **noatime** and **relatime**.

If **atime** updates are enabled, as they are by default on GFS2, every time a file is read, its time stamp recording when the file was last accessed needs to be updated. The file's access timestamp is used by a limited number of applications. These **atime** updates can require a significant amount of write and file-locking traffic, which can degrade GFS2 performance.

The **relatime** (relative atime) Linux mount option specifies that limited updates to the access time of files are allowed. The **atime** is only updated if the previous **atime** update is older than the **mtime** (modification time) or **ctime** (last change time) of the file.

1. Create the Pacemaker GFS2 cluster resource. This cluster resource creation command also specifies the **noatime** mount option to improve performance.

```
[root@nodeY ~]# pcs resource create clusterfs Filesystem \
> device="/dev/cluster_vg/cluster_lv" directory="/var/mountpoint" \
> fstype="gfs2" "options=noatime" op monitor interval=10s on-fail=fence \
> clone interleave=true
```

2. Create resource constraints to set GFS2 and **clvmd** dependency and startup order. GFS2 must start after **clvmd** and must run on the same node as **clvmd**.

```
[root@nodeY ~]# pcs constraint order start clvmd-clone then clusterfs-clone
[root@nodeY ~]# pcs constraint colocation add clusterfs-clone with clvmd-clone
```

3. Verify that the GFS2 file system is mounted as expected.

```
[root@nodeY ~]# mount | grep /mnt/gfs2-demo
/dev/mapper/cluster_vg-cluster_lv on /mnt/gfs2-demo type gfs2
(rw,noatime,seclabel)
```

**References**

**pcs**(8) man page

## Guided Exercise

# Managing a GFS2 Resource in the Cluster

In this lab, you will create a GFS2 cluster resource that contains three journals and mounts on **nodea**, **nodeb**, and **nodec** simultaneously.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to configure a GFS2 cluster resource that mounts simultaneously on **nodea**, **nodeb**, and **nodec** and is managed by Pacemaker.

- Reset your **nodea**, **nodeb**, **nodec**, and **workstation** systems.
- From **workstation**, run the command **lab gfs2 setup**.

**1.** Prepare your **nodea**, **nodeb**, and **nodec** machines for DLM and **clvmd**.

1.1. On **nodea**, **nodeb**, and **nodec**, install the *gfs2-utils* and *lvm2-cluster* packages.

```
[root@nodeY ~]# yum -y install gfs2-utils lvm2-cluster
```

1.2. Configure **no-quorum-policy=freeze** when GFS2 is in use.

```
[root@nodea ~]#  pcs property set no-quorum-policy=freeze
```

1.3. Configure a DLM resource. This is a required dependency for **clvmd** and GFS2.

```
[root@nodea ~]# pcs resource create dlm ocf:pacemaker:controld op monitor \
> interval=30s on-fail=fence clone interleave=true ordered=true
```

1.4. On **nodea**, **nodeb**, and **nodec**, set the **locking_type** parameter in the **etc/lvm/lvm.conf** file to **3**.

```
[root@nodeY ~]# lvmconf --enable-cluster
```

1.5. On each of the three nodes, stop the **lvmetad** service to match the modified LVM configuration.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
Warning: Stopping lvm2-lvmetad, but it can still be activated by:
                        lvm2-lvmetad.socket
```

1.6. Configure **clvmd** as a cluster resource.

```
[root@nodea ~]# pcs resource create clvmd ocf:heartbeat:clvm op monitor \
> interval=30s on-fail=fence clone interleave=true ordered=true
```

1.7. Configure **clvmd** and DLM dependency and startup order. **clvmd** must start after DLM and must run on the same node as DLM.

```
[root@nodea ~]#  pcs constraint order start dlm-clone then clvmd-clone
[root@nodea ~]#  pcs constraint colocation add clvmd-clone with dlm-clone
```

**2.** Create a CLVM logical volume and format it with a GFS2 file system for use by three nodes.

2.1. Create a LVM physical volume on **/dev/mapper/mpatha**.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
```

2.2. Create a clustered volume group with the name **socks**.

```
[root@nodea ~]# vgcreate -Ay -cy socks /dev/mapper/mpatha
```

2.3. Create a 1 GiB logical volume as part of **socks** with the name **shoes**.

```
[root@nodea ~]# lvcreate -L 1G -n shoes socks
```

2.4. On the new logical volume, create a GFS2 file system named **rhel7-gfs2** with three journals. Configure this file system so that it can be used with the cluster.

```
[root@nodea ~]# mkfs.gfs2 -t clusterX:rhel7-gfs2 -j3 /dev/mapper/socks-shoes
```

**3.** Create a Pacemaker cluster resource to manage the GFS2 file system and automatically mount it on all three nodes on cluster startup.

3.1. Configure a **Filesystem** resource named **clusterfs** that mounts the GFS2 file system on all nodes on **/mnt/pants**.

```
[root@nodea ~]#  pcs resource create clusterfs Filesystem \
> device="/dev/socks/shoes" directory="/mnt/pants" \
> fstype="gfs2" "options=noatime" op monitor interval=10s \
> on-fail=fence clone interleave=true
```

3.2. Set up GFS2 and **clvmd** dependency and startup order. GFS2 must start after **clvmd** and must run on the same node as **clvmd**

```
[root@nodea ~]# pcs constraint order start clvmd-clone then clusterfs-clone
[root@nodea ~]# pcs constraint colocation add clusterfs-clone with clvmd-clone
```

3.3. Check the cluster for the GFS2 file system mounted on **/mnt/pants** on all cluster nodes.

```
[root@nodeY ~]# mount | grep /mnt/pants
/dev/mapper/cluster_vg-cluster_lv on /mnt/pants type gfs2
(rw,noatime,seclabel)
```

## Lab

# Providing Storage with the GFS2 Cluster File System

### Performance Checklist

In this lab, you will create resources which provide a clustered file system to all nodes in a cluster.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to configure a highly available clustered GFS2 file system to be mounted simultaneously on all cluster nodes.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, and **nodec** systems.
- From your **workstation** system, run the command **lab gfs2 setup**.

```
[student@workstation ~]$ lab gfs2 setup
```

For an upcoming project, you have been requested to create a file system resource on your **clusterX** cluster that can be simultaneously accessed by all nodes in the cluster.

The cluster nodes have redundant access to iSCSI storage through a multipath device, **/dev/mapper/mpatha**. For ease of future administration, you will create a LVM volume group and a 1 GiB logical volume on the multipath device. The volume group and logical volume will be called **base** and **ball**, respectively.

Enable the volume group for use on all cluster nodes in an active/active configuration using clvmd and DLM. Create a cloned **controld** resource called *dlm* to implement the DLM lock manager on all nodes. Create a cloned **clvm** resource called *clvmd* to run the clustered logical volume manager on all nodes. The **dlm** resource needs to start before the **clvmd** resource and both resources need to run together on each node.

To prevent data corruption, you will format the logical volume with a cluster-aware GFS2 file system named **gfsdata**. This file system will be named **gfsdata** and will use the default journal size of *64 MiB*.

The clustered file system should have the necessary journals to support the three cluster nodes and will be made accessible to all the nodes at mount point **/mnt/stadium** at cluster startup using a cloned **Filesystem** resource called **clusterfs**. The GFS2 file system should mount with the **noatime** option.

1. Prepare the three nodes for DLM and clvmd.

2. Create the clustered volume group and logical volume. Format the logical volume with a GFS2 file system.
3. Create a Pacemaker cluster resource to manage your GFS2 file system and automatically mount it on all three nodes on cluster startup.

## Solution

# Providing Storage with the GFS2 Cluster File System

### Performance Checklist

In this lab, you will create resources which provide a clustered file system to all nodes in a cluster.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

### Outcome(s)

You should be able to configure a highly available clustered GFS2 file system to be mounted simultaneously on all cluster nodes.

- Reset your **workstation** system.
- Reset your **nodea**, **nodeb**, and **nodec** systems.
- From your **workstation** system, run the command **lab gfs2 setup**.

```
[student@workstation ~]$ lab gfs2 setup
```

For an upcoming project, you have been requested to create a file system resource on your **clusterX** cluster that can be simultaneously accessed by all nodes in the cluster.

The cluster nodes have redundant access to iSCSI storage through a multipath device, **/dev/mapper/mpatha**. For ease of future administration, you will create a LVM volume group and a 1 GiB logical volume on the multipath device. The volume group and logical volume will be called **base** and **ball**, respectively.

Enable the volume group for use on all cluster nodes in an active/active configuration using clvmd and DLM. Create a cloned **controld** resource called *dlm* to implement the DLM lock manager on all nodes. Create a cloned **clvm** resource called *clvmd* to run the clustered logical volume manager on all nodes. The **dlm** resource needs to start before the **clvmd** resource and both resources need to run together on each node.

To prevent data corruption, you will format the logical volume with a cluster-aware GFS2 file system named **gfsdata**. This file system will be named **gfsdata** and will use the default journal size of *64 MiB*.

The clustered file system should have the necessary journals to support the three cluster nodes and will be made accessible to all the nodes at mount point **/mnt/stadium** at cluster startup using a cloned **Filesystem** resource called **clusterfs**. The GFS2 file system should mount with the **noatime** option.

1. Prepare the three nodes for DLM and clvmd.

1.1. Install the **gfs2-utils** and **lvm2-cluster** packages on **nodea**, **nodeb**, and **nodec**.

```
[root@nodeY ~]# yum -y install gfs2-utils lvm2-cluster
```

1.2. On all three of your nodes, set the LVM **locking_type** to **3** (enable cluster locking).

```
[root@nodeY ~]# lvmconf --enable-cluster
```

1.3. On each of the three nodes, stop the **lvmetad** service to match the modified LVM configuration.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
    Warning: Stopping lvm2-lvmetad, but it can still be activated by:
                        lvm2-lvmetad.socket
```

1.4. Set the global Pacemaker parameter **no_quorum_policy** to **freeze**.

```
[root@nodea ~]# pcs property set no-quorum-policy=freeze
```

1.5. Create the **controld** resource. This is a required dependency for clvmd and GFS2.

```
[root@nodea ~]# pcs resource create dlm ocf:pacemaker:controld op monitor
 interval=30s on-fail=fence clone interleave=true ordered=true
```

1.6. Create the **clvm** resource.

```
[root@nodea ~]# pcs resource create clvmd ocf:heartbeat:clvm op monitor
 interval=30s on-fail=fence clone interleave=true ordered=true
```

1.7. Create resource constraints to control clvmd and DLM startup order and enforce them to run on the same node.

```
[root@nodea ~]# pcs constraint order start dlm-clone then clvmd-clone
[root@nodea ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

**2.** Create the clustered volume group and logical volume. Format the logical volume with a GFS2 file system.

2.1. Create an LVM physical volume on **/dev/mapper/mpatha**.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
```

2.2. Create a clustered volume group named **base**.

```
[root@nodea ~]# vgcreate -Ay -cy base /dev/mapper/mpatha
```

2.3. Create a 1 GiB logical volume named **ball**.

```
[root@nodea ~]# lvcreate -L 1G -n ball base
```

2.4. Format the logical volume with a GFS2 file system.

```
[root@nodea ~]# mkfs.gfs2 -j3 -p lock_dlm -t clusterX:gfsdata /dev/base/ball
```

**3.** Create a Pacemaker cluster resource to manage your GFS2 file system and automatically mount it on all three nodes on cluster startup.

3.1. Configure the **`Filesystem`** cloned resource.

```
[root@nodea ~]# pcs resource create clusterfs Filesystem device="/dev/base/ball"
 directory="/mnt/stadium" fstype="gfs2" options="noatime" op monitor interval=10s
 on-fail=fence clone interleave=true
```

3.2. Create constraints to mandate the startup ordering of the **`clusterfs`** cloned resource in relationship to the **`clvmd`** cloned resource, as well as to dictate that the two resources need to run on the same nodes.

```
[root@nodea ~]# pcs constraint order start clvmd-clone then clusterfs-clone
[root@nodea ~]# pcs constraint colocation add clusterfs-clone with clvmd-clone
```

3.3. Verify that the GFS2 file system is mounted at **`/mnt/stadium`** on each cluster node.

```
[root@nodeY ~]# mount | grep /mnt/stadium
/dev/mapper/base-ball on /mnt/stadium type gfs2 (rw,noatime,seclabel)
```

# Summary

In this chapter, you learned:

- How **DLM** is used for locking with a **GFS2** file system.
- Not implementing an NTP server can cause **GFS2** performance issues.
- Although journaling can be enabled on files and directories, it only improves performance for file servers.
- The **GFS2** quota facility does not allow for a soft limit grace period.

Chapter 12

# Eliminating Single Points of Failure

**Goal**

Eliminate single points of failure to increase service availability.

**Objectives**

- Describe the need for redundant hardware.
- Prepare for network failures.
- Configure multilevel fencing.

**Sections**

- Planning for Failures (and Quiz)
- Configure Network Redundancy for Cluster Communication (and Practice)
- Configuring Multiple Fence-device Levels (and Practice)

**Lab**

- Eliminating Single Points of Failure

# Planning for Failures

## Objectives

After completing this section, students should be able to describe the need for redundant hardware.

## Expecting components to fail

Sooner or later, every single piece of hardware will fail. For some hardware, this might be a very long time, but another piece might fail within two weeks of installing it.

Furthermore, every single (complex) piece of software will have bugs. Some of these bugs might be unnoticeable, while others might corrupt an entire database.

One of the major tasks for a system administrator is acknowledge that these failures will occur, and plan accordingly. When the failing piece of hardware is a simple desktop machine, the correct approach is most likely to simply have the help desk replace the failed machine, while for a mission-critical server a more proactive approach is needed. When a machine failsi, the service running on that machine should not fail.

## What is a single point of failure?

A *single point of failure* (SPoF) is any part of a complex setup that, when it fails, can and will take down an entire environment. Image a metal chain made up of individual links; when one link breaks, the entire chain fails.

Single points of failure can be found in a lot of places. In a car, having the distribution belt fail will grind the car to a halt; in a home electrical system, a single blown fuse can cut off multiple rooms from electricity.

## Common single points of failure in a cluster

A typical high-availability cluster can have many possible single points of failure as well. The following are by no means exhaustive lists, but they do contain the most common offenders.

**Hardware single points of failure**

| SPoF | Impact and solution |
|---|---|
| Power supply | When a machine loses power it turns off. A standard solution is to use two redundant power supplies, attached to two different UPS systems, attached to different mains providers, or at least attached to different electrical groups. A larger datacenter will also have backup generators to provide emergency power in case of a longer blackout. |
| Local storage | When a hard disk, or SSD, dies, it can take down an entire machine. A common solution is to run multiple drives in a fault-tolerant RAID setup, such as RAID 1, RAID 5, or RAID 6. |

| SPoF | Impact and solution |
|---|---|
| Network interfaces | When a NIC fails, a machine is effectively cut off from the network that NIC was connected to. A solution is to use two or more NICs in a *bonded* configuration. |
| Network switches | When a switch fails, it can take out more than one machine, cutting them off from the network. A solution is to used bonded NICs, connected to different switches. |
| Fencing hardware | If only a single fencing device is used for a host, and that fencing device fails—for example, an ILO card is no longer responding—the cluster can no longer remove a failed machine from the cluster. This will halt service relocation, disabling the service that was running on the failed node that could not be fenced. This can be solved by using multiple fencing devices, assigned to multiple fencing levels. |

**Software single points of failure**

| SPoF | Impact and solution |
|---|---|
| Cluster communications | A Red Hat high-availability cluster relies on continuous communications between all nodes. If that communication gets interrupted, nodes will be considered as failed. Communications can be made more resilient by using two networks for communication instead of one. Furthermore, bonded interfaces can be used for the different networks for further resilience. |
| Shared storage connection | When a node gets cut off from shared storage, it can probably no longer effectively provide services to consumers. For Fibre Channel storage, multiple HBAs can be used, combined with multipathing, to add resilience to the storage connection. For iSCSI, multiple (bonded) network interfaces can be used to log into the same target over different networks, combined with multipathing. The actual shared storage back end should also use some form of redundant storage, such as RAID 15. |
| Software fencing configuration | Having multiple fencing devices per node becomes moot when they are all configured to be used at the same time. If one fails, the cluster will still consider fencing to have failed. To prevent this, fencing devices can be assigned to *fence levels*. If a fence device in the first level fails, the cluster will try the second level, etc. |

**References**
Red Hat High Availability Add-On Administration Guide

## Quiz

# Planning for Failures

Choose the correct answer to the following questions, based on the following diagram. Assume that all lines are single network or power cables, attached to a single NIC or PDU.

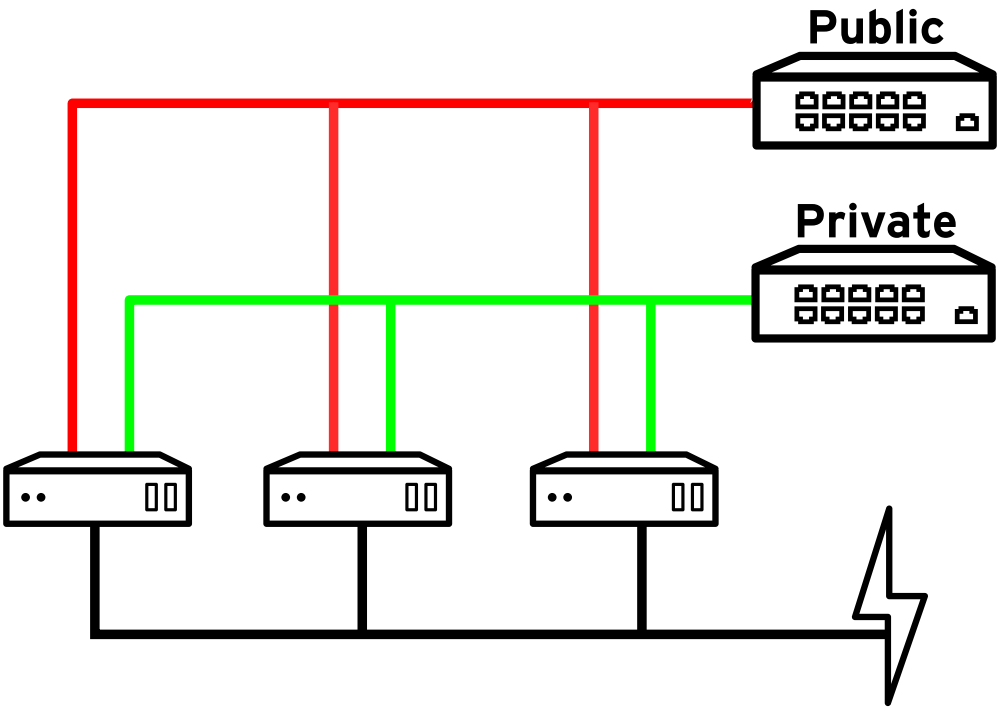



Figure 12.1: A three-node cluster

**1. Given the three-node cluster design in the image, what single point(s) of failure can you identify? Choose all that apply.**

a. All "Private" network connections run through the same switch.
b. All "Public" network connections run through the same switch.
c. All power comes out of the same connection.
d. The cluster does not use shared storage.

**2. What steps could be taken to improve the network redundancy in this cluster? Choose all that apply.**

a. Multiple switches should be used.
b. Shared storage should be added.
c. Multiple NICs should be bonded together.
d. Multiple power sources should be used.

**3. Assuming that all nodes have two redundant power supplies, and that the "bolt" icon is a UPS with network-controlled outlets, how could this be improved specifically for fencing? Choose all that apply.**

a. Plug both power supplies on both nodes into the UPS.

b. Configure a redundant network connection to the UPS, routable from the "Private" network.

c. Add a second UPS and wire one power supply per server to each of the UPSs.

d. Run a daily UPS test at 10:00 a.m.

## Solution

# Planning for Failures

Choose the correct answer to the following questions, based on the following diagram. Assume that all lines are single network or power cables, attached to a single NIC or PDU.

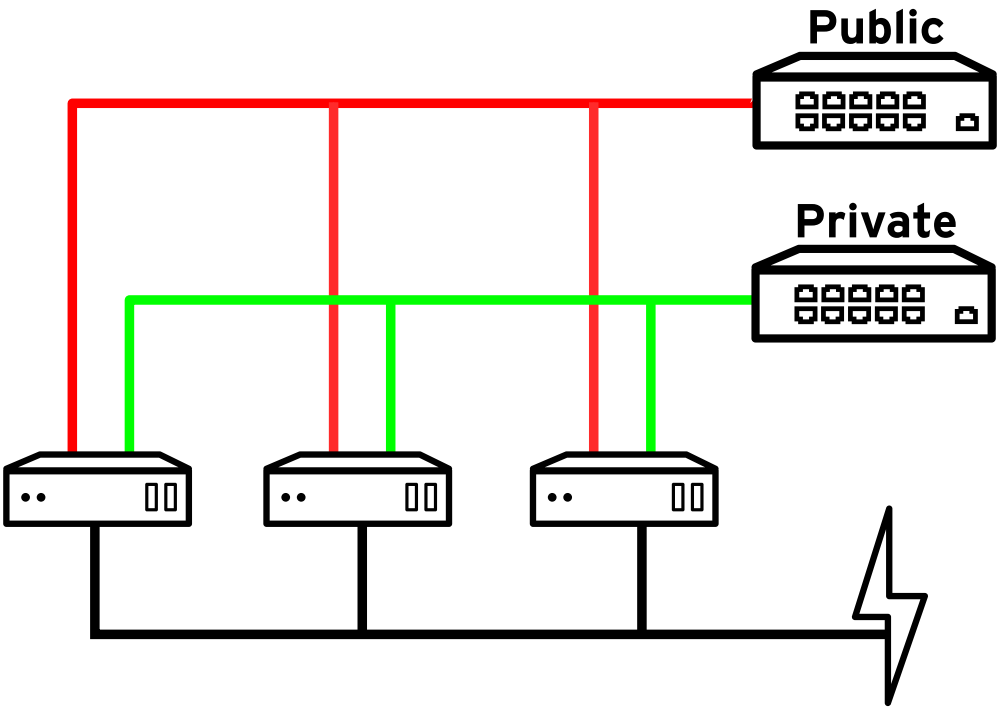



Figure Error.1: A three-node cluster

1. **Given the three-node cluster design in the image, what single point(s) of failure can you identify? Choose all that apply.**
   a. All "Private" network connections run through the same switch.
   b. All "Public" network connections run through the same switch.
   c. All power comes out of the same connection.
   d. The cluster does not use shared storage.

2. **What steps could be taken to improve the network redundancy in this cluster? Choose all that apply.**
   a. Multiple switches should be used.
   b. Shared storage should be added.
   c. Multiple NICs should be bonded together.
   d. Multiple power sources should be used.

**3. Assuming that all nodes have two redundant power supplies, and that the "bolt" icon is a UPS with network-controlled outlets, how could this be improved specifically for fencing? Choose all that apply.**

a. Plug both power supplies on both nodes into the UPS.

b. Configure a redundant network connection to the UPS, routable from the "Private" network.

c. Add a second UPS and wire one power supply per server to each of the UPSs.

d. Run a daily UPS test at 10:00 a.m.

# Configuring Network Redundancy for Cluster Communication

## Objectives

After completing this section, students should be able to prepare for network failures.

## What is Redundant Ring Protocol

*Redundant Ring Protocol* (RRP) is a communication protocol where **`corosync`** can use two separate networks to establish connections between cluster nodes. When one network fails, communication can continue over the second network. RRP is not a replacement for network channel bonding; instead, bonded interfaces can be used to connect to the two rings for further redundancy.

Where bonded interfaces protect against hardware failures (NIC, network cable, switch), RRP adds protection against larger infrastructure collapses, such as a BGP storm taking out an entire network.

## Redundant Ring Protocol support limitations

In Red Hat Enterprise Linux 7 there are a number of support limitations when using RRP:

- RRP supports **`multicast`** and **`udpu`** transports. **`broadcast`** is not supported.
- RRP is only supported on Ethernet networks. *IP over InfiniBand* (IPoIB) is not supported.
- RRP only supports a maximum of two rings, the **`clusternode`** name and the **`altname`**.
- RRP is supported on bonded network interfaces, but *teaming* using **`teamd`** on Red Hat Enterprise Linux 7 is not supported. Supported bonding modes on Red Hat Enterprise Linux 7 are **`0`** (balance-rr), **`1`** (active-backup), and **`2`** (balance-xor).
- Services using DLM are not supported. This includes clvmd, cmirror, GFS2, and rgmanager.

## Redundant ring configuration options

RRP has one main configuration option, **`rrp_mode`**. **`rrp_mode`** can be set to one of the following three settings:

- **`none`**: No RRP is used. This is the default when using only one ring, and cannot be used with two rings.
- **`active`**: Active replication offers slightly lower latencies, combined with a slower total throughput.
- **`passive`**: Passive replication may double the total speed of the totem protocol, at the cost of increased latency.

### Creating a new cluster with RRP

When creating a new cluster, RRP can be activated by specifying two host names per node, separated by a comma: the **`nodename`**, and the **`altname`**. When a cluster is created in this way, RRP will be activated automatically. A mode can be chose by adding the **`--rrpmode active|passive`** option.

```
[root@nodeY ~]# pcs cluster setup --start --enable \
> --name rrpcluster \
> node1fqdn,node1alt \
> node2fqdn,node2alt \
> node3fqdn,node3alt \
> --rrpmode active
```

## Updating an existing cluster to use RRP

Updating an existing cluster to use RRP will require the entire cluster to be stopped. Use the following procedure to update an existing cluster:

1. Stop the entire cluster.

```
[root@nodeY ~]# pcs cluster stop --all
```

2. Update **/etc/corosync/corosync.conf** on one node with the following:
   - In the **totem** section, add a line for **rrp_mode: active** or **rrp_mode: passive**.
   - For each **node** block inside the **nodelist** block, add a line for **ring1_addr: *altnamefqdn***.
3. Sync the updated **corosync.conf** to all other cluster nodes from the node where it was updated.

```
[root@nodeY ~]# pcs cluster sync
```

4. Start the cluster.

```
[root@nodeY ~]# pcs cluster start --all
```

**Important**

When using **multicast** and not **udpu**, instead of updating the **nodelist** block, a second **interface** block should be added to the **totem** block, with **ringnumber: 1**, and a full set of **bindnetaddr**, **mcastaddr**, **mcastport**, and **ttl** options.

**References**

Red Hat High Availability Add-On Reference Guide

- Configuring Redundant Ring Protocol (RRP)

**How can I configure a redundant heartbeat network for a RHEL 5, 6, or 7 High Availability cluster?**
https://access.redhat.com/solutions/61832

**What are the supported network bonding modes for Red Hat High Availability clusters?**
https://access.redhat.com/solutions/27604

## Guided Exercise

# Configuring Network Redundancy for Cluster Communication

In this lab, you will configure a new three-node cluster using Redundant Ring Protocol.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to create a cluster using Redundant Ring Protocol.

- Reset all four of your **node*Y*** machines.

**1.** Prepare your **nodea**, **nodeb**, and **nodec** systems to act as cluster nodes.

1.1. On all three of your nodes, install the necessary software, including the custom *fence-agents-rht* fencing agent.

```
[root@nodeY ~]# yum -y install pcs fence-agents-all fence-agents-rht
```

1.2. On all three of your nodes, start and enable the pcs service.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

1.3. On all three of your nodes, open all relevant firewall ports for cluster communications.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

1.4. On all three of your nodes, set the password for the **hacluster** user to **redhat**.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
```

**2.** From your **nodea** system, authenticate all nodes, then create a new three-node cluster named **clusterX** using your node host names on the **private.example.com** network for **ring0**, and the host names on the **storage1.example.com** network as **ring1**. The cluster should use **passive** mode for the Redundant Ring Protocol.

2.1. From your **nodea** system, authenticate all nodes.

```
[root@nodea ~]# pcs cluster auth -u hacluster -p redhat \
> node{a..c}.private.example.com
```

2.2. Create a new three-node cluster named **clusterX** using your node host names on the **private.example.com** network for **ring0**, and the host names on the **storage1.example.com** network as **ring1**. The cluster should use **passive** mode for the Redundant Ring Protocol.

```
[root@nodea ~]# pcs cluster setup --start --enable \
> --name clusterX \
> nodea.private.example.com,nodea.storage1.example.com \
> nodeb.private.example.com,nodeb.storage1.example.com \
> nodec.private.example.com,nodec.storage1.example.com \
> --rrpmode passive
```

**3.** Set up fencing for your cluster. You can use the **fence_rht** fencing agent, with **classroom.example.com** as the fencing daemon. The fencing device plug names for your hosts are their host names on the **private.example.com** network.

3.1. Create a new fencing device called **fence_all**.

```
[root@nodea ~]# pcs stonith create fence_all fence_rht \
> ipaddr="classroom.example.com" \
> pcmk_host_check="none" \
> pcmk_host_list=""
```

**4.** Test your configuration by first bringing down the **System eth1** and **System eth2** network connections on your **nodec** machine in order, pausing between the two interfaces to observe that the cluster is still fully operational (apart from **pcsd**).

4.1. Bring down the **System eth1** network connection on your **nodec** machine.

```
[root@nodec ~]# nmcli con down "System eth1"
```

4.2. Observe that the cluster still considers **nodec** as **Online**.

```
[root@nodea ~]# corosync-quorumtool
```

4.3. Bring down the **System eth2** network connection on your **nodec** machine.

```
[root@nodec ~]# nmcli con down "System eth2"
```

4.4. Observe how **nodec** is now fenced, since it can no longer communicate on either of the two redundant rings.

# Configuring Multiple Fencing-device Levels

## Objectives

After completing this section, students should be able to configure multilevel fencing.

## Configuring storage fencing

Next to configuring fencing to power-cycle a node (or to simply turn it off), fencing can also be configured to shut a faulty node off from shared storage. This is called *storage fencing* or *SCSI fencing*.

SCSI fencing is implemented in the Red Hat Enterprise Linux High Availability Add-on using the **`fence_scsi`** fencing agent. When **`fence_scsi`** is used, whenever a cluster node starts it will create a *SCSI reservation* for itself using a key. Once one or more keys are added to a SCSI device, it will only allow communication from clients using one of the configured keys.

When the node needs to be fenced from the storage, one of the other nodes will remove the reservation for the key from the failed node, resulting in the failed node being cut off from storage.

Most SCSI devices (including Fibre Channel devices) support SCSI reservations, including the Linux iSCSI target that ships with Red Hat Enterprise Linux 7.

### Creating a fence_scsi fence device

Creating a **`fence_scsi`** fencing device is almost the same as creating any other fencing device, but a number of options must be used:

| Option | Explanation |
|---|---|
| **`devices=`*`devicenodefullpath`*** | A comma-separated list of devices to be fenced. Since shared storage devices might end up with different device names on different nodes, the use of the symbolic links under **`/dev/disk/by-id/`** is highly recommended. |
| **`pcmk_monitor_action=metadata`** | This option tells the **`stonithd`** daemon to request the default **`fence_scsi`** metadata action, instead of the **`monitor`** action, when attempting to monitor the fence device. This is necessary to work around an open bug in the **`fence_scsi`** fencing agent. |

| Option | Explanation |
|---|---|
| **pcmk_host_list="*FQDN...*"** | This options lists for which nodes the fencing device can be used. It should typically include all nodes that can utilize the shared storage device. |
| **pcmk_reboot_action="off"** | The default operation for **stonithd** is **reboot**: turning something off, then on again. In the case of storage fencing this is suboptimal, since the node being fenced will remain powered on. |
| **meta provides="unfencing"** | When a node is fenced using a power cycle, the act of rejoining the cluster will automatically mark the host as being *unfenced*. With storage fencing, this will not happen, and a separate option is needed to tell the cluster that when the node rejoins the cluster, it should be unfenced. |

# Configuring redundant fencing

When multiple fence devices are available for a single node, it can be useful to configure them in a layered method; for example, always try to fence using a networked power strip, but if that fails, and only if that fails, fence the node using the built-in management card.

This type of redundant fencing can be implemented using *fencing levels*. Fencing levels are numbered sequentially, starting at **1**. When a node needs to be fenced, **stonithd** will first attempt to use *all* fencing devices in level **1**, in the order in which they were added to the level. If any of the fencing devices fails, processing is stopped, and **stonithd** will continue trying the fence devices configured in level **2**, etc.

## Creating a fence level

A fencing level can be created with the command **pcs stonith level add <*level*> <*node*> <*devices*>**.

For example, to create two fencing levels for **nodea**, level **1** using the fence device **fence_nodea_ilo**, and level **2** using the fence device **fence_nodea_apc**, the following commands can be used.

```
[root@nodea ~]# pcs stonith level add 1 nodea fence_nodea_ilo
[root@nodea ~]# pcs stonith level add 2 nodea fence_nodea_apc
```

## Managing fence levels

Fence level configurations can be viewed using both the **pcs status** and **pcs stonith level** commands. To remove a fence level, or a single node or device from a fence level, the **pcs stonith level remove** command can be used (with appropriate options).

**pcs stonith level clear** can be used to clear all fence levels, or to remove a node or fence device from all fence levels.

# Configuring fencing for more than one power supply

A common setup that needs careful configuration is power-fencing a node that has redundant power supplies. Somewhere during the fencing process, power must be removed from both power supplies in order to fence the node.

Simply creating a fence device without extra options and adding both of them to a fence level will not work, since the default action for power fencing is **reboot**. This would result in the following order:

1. Reboot power supply A
2. Reboot power supply B

At no point in the previous example has power been completely lost to the machine. This can be seen by changing the **reboot** action into separate **off** and **on** actions:

1. Power supply A off
2. Power supply A on
3. Power supply B off
4. Power supply B on

What is needed in order to ensure that the machine will be completely rebooted is the following:

1. Power supply A off
2. Power supply B off
3. Power supply A on
4. Power supply B on

In order to configure this using **pcs**, four separate fence devices will need to be created:

```
[root@nodea ~]# pcs stonith create node1_port1_off fence_apc port=1
 pcmk_reboot_action="off"
[root@nodea ~]# pcs stonith create node1_port2_off fence_apc port=2
 pcmk_reboot_action="off"
[root@nodea ~]# pcs stonith create node1_port1_on fence_apc port=1
 pcmk_reboot_action="on"
[root@nodea ~]# pcs stonith create node1_port2_on fence_apc port=2
 pcmk_reboot_action="on"
```

With these four explicit **off** and **on** actions, a fencing level can be created for this node:

```
[root@nodea ~]# pcs stonith level add 1 nodea
 nodea_port1_off,nodea_port2_off,nodea_port1_on,nodea_port2_on
```

**Important**

The order in which the fence devices are added to the level is the order in which they will be executed. Somewhere during the sequence of fence devices, power should be lost completely in order to fence a machine. If all individual fencing actions complete successfully, the cluster will assume that the node has been properly fenced, even if power was never lost completely.

## Adding more devices

If a device ever needs to be added to a fencing level after it has been created, for example, when an extra power supply is added, the entire level will need to be deleted and re-created. This is because ordering between devices at creation time is preserved, but cannot be guaranteed for devices added afterwards.

**References**

Red Hat High Availability Add-On Reference Guide

- Fencing: Configuring STONITH

**fence_scsi stonith device fails to start or function in a RHEL 7 Update 1 High Availability cluster and logs show "Failed: nodename or key is required"**
https://access.redhat.com/solutions/1421063

**How can I configure fencing for redundant power supplies in a RHEL 6 or 7 High Availability cluster with pacemaker?**
https://access.redhat.com/solutions/1173123

## Guided Exercise

# Configuring Multiple Fencing-device Levels

In this lab, you will configure multilevel fencing.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec** |

## Outcome(s)

You should be able to configure multilevel fencing.

- Reset your **workstation** machine.
- Reset all four of your **node*Y*** machines.
- On your **workstation** machine, run the command **lab multi-level-fencing setup**.

Your nodes have all been configured with two redundant paths into an iSCSI storage device. These paths have been combined into a multipath device.

1. Create a new fence device, called **scsi-killer**, that can be used to separate any of your nodes from shared storage, using the **/dev/disk/by-id/dm-uuid-mpath-*** name of your multipath device.

   1.1. Discover the **/dev/disk/by-id/dm-uuid-mpath-*** name of your multipathed shared storage.

```
[root@nodea ~]# ls /dev/disk/by-id/dm-uuid-mpath-*
```

   1.2.

```
[root@nodea ~]# pcs stonith create scsi-killer fence_scsi \
> devices=/dev/disk/by-id/dm-uuid-mpath-FOUNDNAME \
> pcmk_monitor_action=metadata \
> pcmk_host_list="nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com" \
> pcmk_reboot_action="off" \
> meta provides="unfencing"
```

2. For each of your three nodes, create two fencing levels: a first level using the **fence_rht** fencing device, and a second level using the **fence_scsi** fencing device.

   2.1. For each node, create a level **1** fencing level with the relevant **fence_rht** fencing device in it.

```
[root@nodeY ~]# pcs stonith level add 1 nodeY.private.example.com fence_nodeY
```

2.2. For each node, create a level **2** fencing level with the **scsi-killer** fencing device in it.

```
[root@nodeY ~]# pcs stonith level add 2 nodeY.private.example.com scsi-killer
```

**3.** Intentionally break the configuration for **fence_nodec** by changing the **ipaddr** setting to **broke.example.com**, then try to fence **nodec** and observe what happens.

3.1. Intentionally break the configuration for **fence_nodec**.

```
[root@nodea ~]# pcs stonith update fence_nodec ipaddr=broken.example.com
```

3.2. Manually fence **nodec**.

```
[root@nodea ~]# pcs stonith fence nodec.private.example.com
```

3.3. Observe what happens. While **nodec** is not automatically rebooted, fencing still succeeds. When running **multipath -ll**, the links sometimes show as **faulty** before iSCSI attempts to recover. In reality, not a single write from **nodec** will ever make it to the storage device.

## Lab

# Eliminating Single Points of Failure

### Performance Checklist

In this lab, you will update a three-node cluster to use Redundant Ring Protocol and combined fencing.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec**. |

### Outcome(s)

You should be able to update an existing cluster to use multilevel combined fencing, and Redundant Ring Protocol.

- Reset your **workstation** system.
- Reset all four of your **node*Y*** systems.
- From your **workstation** system, run **lab fencerrp setup**.

```
[student@workstation ~]$ lab fencerrp setup
```

One of your former colleagues had started setup on a new cluster, **cluster*X***, when he left your company. He has already configured fencing using **fence_rht** devices, and iSCSI storage using the **storage2.example.com** network, but had not finished the configuration on combined fencing and Redundant Ring Protocol.

Your job, as part of the cleanup operation, is to add a second ring using the **storage1.example.com** network, and to put the Redundant Ring Protocol in **active** mode.

You will also need to configure **fence_scsi** fencing, and create a level **1** fence level for each node, using both the old **fence_rht** fencing device and the newly created **fence_scsi** fence device.

1. Update the current cluster configuration to use **active** Redundant Ring Protocol, using the **storage1.example.com** network for the second ring.
2. Configure a **fence_scsi** device for the logged-in iSCSI device, then create a level **1** fence level for each node, using both the old **fence_rht** fence device and the new **fence_scsi** device.

## Solution

# Eliminating Single Points of Failure

## Performance Checklist

In this lab, you will update a three-node cluster to use Redundant Ring Protocol and combined fencing.

| Resources | |
|---|---|
| **Machines** | **workstation**, **nodea**, **nodeb**, and **nodec**. |

## Outcome(s)

You should be able to update an existing cluster to use multilevel combined fencing, and Redundant Ring Protocol.

- Reset your **workstation** system.
- Reset all four of your **node*Y*** systems.
- From your **workstation** system, run **lab fencerrp setup**.

```
[student@workstation ~]$ lab fencerrp setup
```

One of your former colleagues had started setup on a new cluster, **cluster*X***, when he left your company. He has already configured fencing using **fence_rht** devices, and iSCSI storage using the **storage2.example.com** network, but had not finished the configuration on combined fencing and Redundant Ring Protocol.

Your job, as part of the cleanup operation, is to add a second ring using the **storage1.example.com** network, and to put the Redundant Ring Protocol in **active** mode.

You will also need to configure **fence_scsi** fencing, and create a level **1** fence level for each node, using both the old **fence_rht** fencing device and the newly created **fence_scsi** fence device.

1. Update the current cluster configuration to use **active** Redundant Ring Protocol, using the **storage1.example.com** network for the second ring.

   1.1. On **nodea**, update **/etc/corosync/corosync.conf** to include the following:

   - A line reading **rrp_mode: active** in the **totem** block.
   - A **ring1_addr: node*Y*.storage1.example.com** line in each **node** block.

   1.2. Sync the new configuration to all nodes.

```
[root@nodea ~]# pcs cluster sync
```

   1.3. Stop, then start, all cluster nodes.

```
[root@nodea ~]# pcs cluster stop --all
```

```
[root@nodea ~]# pcs cluster start --all
```

2. Configure a **fence_scsi** device for the logged-in iSCSI device, then create a level **1** fence level for each node, using both the old **fence_rht** fence device and the new **fence_scsi** device.

   2.1. Find the **/dev/disk/by-path/** of the iSCSI device.

```
[root@nodea ~]# ls -l /dev/disk/by-path/ip-192*
```

   2.2. Create the new fencing device.

```
[root@nodea ~]# pcs stonith create scsi-killer fence_scsi \
> devices=/dev/disk/by-path/FOUNDNAME \
> pcmk_monitor_action=metadata \
> pcmk_host_list="nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com" \
> pcmk_reboot_action="off" \
> meta provides="unfencing"
```

   2.3. For each node, create a level **1** fence level with both fence devices.

```
[root@nodeY ~]# pcs stonith level add 1 nodeY.private.example.com fence_nodeY
[root@nodeY ~]# pcs stonith level add 1 nodeY.private.example.com scsi-killer
```

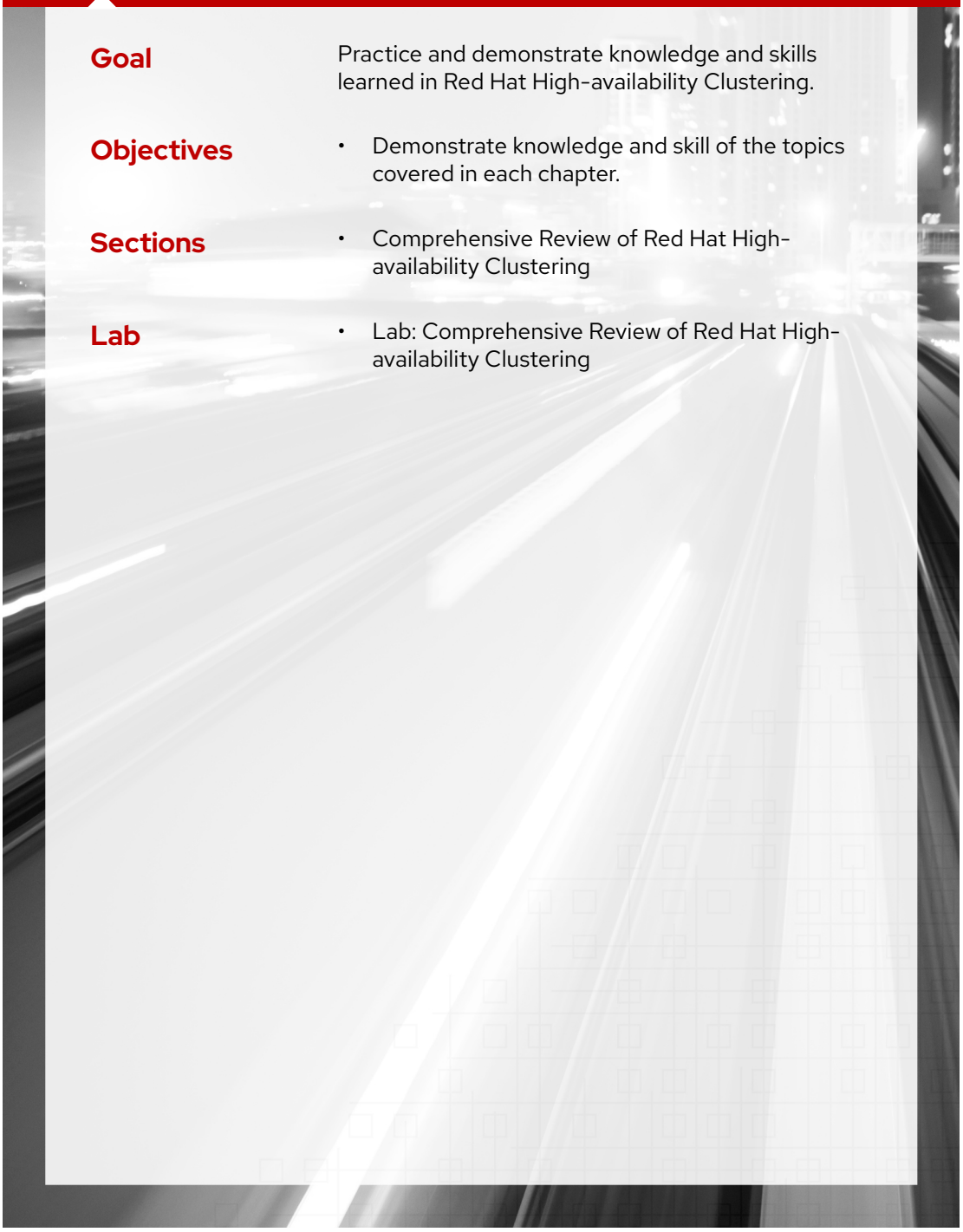
# Summary

In this chapter, you learned:

- Any piece of software or hardware that takes down more than itself when it fails can be considered a *Single Point of Failure* (SPoF).
- All hardware and software will fail at some point.
- *Redundant Ring Protocol* (RRP) can be configured during cluster creation time by specifying an **`altname`** on a second network for each node.
- A second ring can also be manually added to **`/etc/corosync/corosync.conf`** by adding a **`ring1_addr`** line for each node.
- **`fence_scsi`** can be used as a fence device to separate a node from shared storage by using SCSI reservations.
- Multiple fence levels can be configured, with multiple fence devices in each level.
- If one or more fence devices in a level fail, that level will be considered failed. The next level will then be attempted.
- Fence devices inside a level are executed in the order in which they were added at *creation* time. This can be useful for fencing machines with more than one power supply.

Chapter 13

# Lab: Comprehensive Review of Red Hat High-availability Clustering



| | |
|---|---|
| **Goal** | Practice and demonstrate knowledge and skills learned in Red Hat High-availability Clustering. |
| **Objectives** | • Demonstrate knowledge and skill of the topics covered in each chapter. |
| **Sections** | • Comprehensive Review of Red Hat High-availability Clustering |
| **Lab** | • Lab: Comprehensive Review of Red Hat High-availability Clustering |

# Comprehensive Review of Red Hat High-availability Clustering

## Objectives

After completing this section, students should be able to demonstrate knowledge and skill of the topics covered in each chapter.

## Reviewing Red Hat enterprise high-availability clustering

Before beginning the comprehensive review for this course, students should be comfortable with the topics covered in each chapter.

Students can refer to earlier sections in the textbook for extra study.

### Chapter 1, *Creating High-availability Clusters*

Create a basic high-availability cluster.

- Explain what a high-availability cluster is and when it should be used.
- Identify the components of a RHEL 7 high-availability cluster.
- Install and start a small high-availability cluster.

### Chapter 2, *Managing Cluster Nodes and Quorum*

Manage cluster membership and quorum voting.

- Change whether nodes are active members of a high-availability cluster on a temporary or permanent basis.
- Explain how quorum operates and protects a cluster from errors.
- Adjust the way quorum is determined to allow special configurations.

### Chapter 3, *Managing Fencing*

Select and configure fencing devices to separate nodes from storage after a failure.

- Explain how fencing is used to protect data during node failures.
- Prepare the hardware or software fencing device.
- Configure the cluster to use the correct fence devices when fencing a node.

### Chapter 4, *Creating and Configuring Resources*

Create basic resources and resource groups to provide highly available services.

- Create and configure high-availability resources.
- Assemble resource groups and adjust the properties of their resources.

- Manually control and relocate resource groups in a running cluster.

## Chapter 5, *Troubleshooting High-availability Clusters*

Identify and troubleshoot cluster problems.

- Inspect and configure cluster logging.
- Configure cluster event notifications.
- Debug resource failures.
- Debug cluster network issues.

## Chapter 6, *Controlling Complex Resource Groups*

Create complex resource groups using ordering and location constraints.

- Demonstrate how resource groups can be used to control resource startup order by configuring an active/passive NFS server resource group.
- Set constraints manually for resource groups or individual resources to influence which nodes they run on.

## Chapter 7, *Managing Two Node Clusters*

Identify and work around two node cluster issues.

- Describe two-node cluster issues.
- Configure a cluster to work in two-node mode.

## Chapter 8, *Managing iSCSI Initiators*

Manage iSCSI initiators to access shared storage.

- Review common iSCSI initiator configuration and use.
- Manage iSCSI initiator timeout settings.

## Chapter 9, *Accessing Storage Devices Redundantly*

Configure redundant storage access.

- Describe multipathing and multipathing terminology.
- Configuring redundant storage access with dm-multipath.
- Testing multipath configurations.

## Chapter 12, *Eliminating Single Points of Failure*

Eliminate single points of failure to increase service availability.

- Describe the need for redundant hardware.
- Prepare for network failures.
- Configure multilevel fencing.

## Lab

# Comprehensive Review of Red Hat High-availability Clustering

In this lab, you will configure a new four node cluster with monitoring, clustered LVM, and GFS2.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, **noded**, and **workstation** |

## Outcome(s)

You should be able to create a highly available, active-passive four-node cluster named **clusterX** with fencing, logging, event notification, services, and clustered file system configured.

- Reset **nodea**, **nodeb**, **nodec**. **noded**, and **workstation**.
- Log into **workstation** and run the command **lab hacluster setup**.

Create a new cluster with the following requirements. After completion, run the command **lab hacluster grade** on **workstation** to verify your work.

- The name of the new cluster is **clusterX**.
- The cluster consists of four nodes: **nodea.private.example.com**, **nodeb.private.example.com**, **nodec.private.example.com**. and **noded.private.example.com**.
- The firewall allows cluster communication on all cluster nodes.
- The cluster is configure to use the **auto_tie_breaker** quorum option and to log messages to **/var/log/cluster.log**.
- Each of the cluster nodes automatically joins the cluster after reboot.
- Each cluster node, **nodeY**, is configured with a fence device, **fence_nodeY**, using the **fence_rht** classroom fencing agent. Each fencing device is reachable at **classroom.example.com**, and has its node's FQDN on the **private.example.com** domain, **nodeY.private.example.com**, as the plug name.
- The target *iqn.2015-06.com.example:cluster* has been configured on **workstation** and is available to all nodes through both portal addresses, *192.168.1.9* and *192.168.2.9*. The iSCSI initiator on each node is configured with an IQN of **iqn.2015-06.com.example:nodeY** and also has a **replacement_timeout** setting of *5* seconds. Upon boot, each node logs into the target on **workstation** at both portal addresses.
- Each cluster nodes has a multipath device, **/dev/mapper/mpatha**, that provides redundant access to the iSCSI target, *iqn.2015-06.com.example:cluster*, at portal addresses, *192.168.1.9* and *192.168.2.9*.

- The cluster nodes are configured with a logical volume, *clusterlv*, in a volume group, *clustervg* which is set up as an active/active configuration in the cluster using CLVM. A cloned **controld** resource called *dlm* implements the DLM lock manager on each node. A cloned **clvm** resource called *clvmd* offers the clustered logical volume manager on each node. The *dlm* resource starts before the *clvmd* resource and both resources run together on each node.

- Upon cluster startup, a **Filesystem** resource called **clusterfs** makes a GFS2 filesystem located on the **/dev/clustervg/clusterlv** logical volume available to all nodes at mount point **/var/log/httpd**. The file system is mounted with the **noatime** option and its contents have the proper SELinux context for the **/var/log/httpd** directory.

- The cluster is configured with resource group, **haweb** which provides a webserver that distributes read-only content from the NFS share **workstation.storage1.example.com:/exports/www** on the IP address **172.25.X.80** which resides on a class C (/24) subnet. The resource group consists of an **IPAddr2** resource called *webip*, a **Filesystem** resource called *webfs*, and an **apache** resource called *webserver*. The *webip* resource is started first, followed by the *webfs* resource, and then the *webserver* resource.

- The **haweb** resource group is also configured with a **MailTo** resource called **webmail** which automatically sends an email with the subject "CLUSTER-NOTIFICATION" to **student@workstation.clusterX.example.com** whenever a status change occurs on the resource group. This resource will be started after all other resources in the resource group has started..

- The **haweb** resource group is configured so that **noded** is the preferred node.

- The **haweb** resource group starts after the *clusterfs* cloned resource.

## Solution

# Comprehensive Review of Red Hat High-availability Clustering

In this lab, you will configure a new four node cluster with monitoring, clustered LVM, and GFS2.

| Resources | |
|---|---|
| **Machines** | **nodea**, **nodeb**, **nodec**, **noded**, and **workstation** |

## Outcome(s)

You should be able to create a highly available, active-passive four-node cluster named **clusterX** with fencing, logging, event notification, services, and clustered file system configured.

- Reset **nodea**, **nodeb**, **nodec**. **noded**, and **workstation**.
- Log into **workstation** and run the command **lab hacluster setup**.

Create a new cluster with the following requirements. After completion, run the command **lab hacluster grade** on **workstation** to verify your work.

- The name of the new cluster is **clusterX**.
- The cluster consists of four nodes: **nodea.private.example.com**, **nodeb.private.example.com**, **nodec.private.example.com**. and **noded.private.example.com**.
- The firewall allows cluster communication on all cluster nodes.
- The cluster is configure to use the **auto_tie_breaker** quorum option and to log messages to **/var/log/cluster.log**.
- Each of the cluster nodes automatically joins the cluster after reboot.
- Each cluster node, **nodeY**, is configured with a fence device, **fence_nodeY**, using the **fence_rht** classroom fencing agent. Each fencing device is reachable at **classroom.example.com**, and has its node's FQDN on the **private.example.com** domain, **nodeY.private.example.com**, as the plug name.
- The target *iqn.2015-06.com.example:cluster* has been configured on **workstation** and is available to all nodes through both portal addresses, *192.168.1.9* and *192.168.2.9*. The iSCSI initiator on each node is configured with an IQN of **iqn.2015-06.com.example:nodeY** and also has a **replacement_timeout** setting of *5* seconds. Upon boot, each node logs into the target on **workstation** at both portal addresses.
- Each cluster nodes has a multipath device, **/dev/mapper/mpatha**, that provides redundant access to the iSCSI target, *iqn.2015-06.com.example:cluster*, at portal addresses, *192.168.1.9* and *192.168.2.9*.

- The cluster nodes are configured with a logical volume, *clusterlv*, in a volume group, *clustervg* which is set up as an active/active configuration in the cluster using CLVM. A cloned **controld** resource called *dlm* implements the DLM lock manager on each node. A cloned **clvm** resource called *clvmd* offers the clustered logical volume manager on each node. The *dlm* resource starts before the *clvmd* resource and both resources run together on each node.

- Upon cluster startup, a **Filesystem** resource called **clusterfs** makes a GFS2 filesystem located on the **/dev/clustervg/clusterlv** logical volume available to all nodes at mount point **/var/log/httpd**. The file system is mounted with the **noatime** option and its contents have the proper SELinux context for the **/var/log/httpd** directory.

- The cluster is configured with resource group, **haweb** which provides a webserver that distributes read-only content from the NFS share **workstation.storage1.example.com:/exports/www** on the IP address **172.25.X.80** which resides on a class C (/24) subnet. The resource group consists of an **IPAddr2** resource called *webip*, a **Filesystem** resource called *webfs*, and an **apache** resource called *webserver*. The *webip* resource is started first, followed by the *webfs* resource, and then the *webserver* resource.

- The **haweb** resource group is also configured with a **MailTo** resource called **webmail** which automatically sends an email with the subject "CLUSTER-NOTIFICATION" to **student@workstation.clusterX.example.com** whenever a status change occurs on the resource group. This resource will be started after all other resources in the resource group has started..

- The **haweb** resource group is configured so that **noded** is the preferred node.

- The **haweb** resource group starts after the *clusterfs* cloned resource.

1. Prepare the virtual machines **nodea**, **nodeb**, **nodec**, and **noded** to act as cluster nodes. Allow the cluster software to communicate through the firewall. Install, enable, and start the required software.

   Fencing for each node should be handled by a fence device named **fence_nodeY**, and handled by the fencing server on **classroom.example.com**. The plug name for each node is **nodeY.private.example.com**.

   Configure the cluster to use the **auto_tie_breaker** quorum option and to log messages to **/var/log/cluster.log**..

   1.1. Allow cluster communications to pass through the firewall on every cluster node.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=high-availability
[root@nodeY ~]# firewall-cmd --reload
```

   1.2. Install the *pcs* and *fence-agents-rht* RPM packages on **nodea.private.example.com**, **nodeb.private.example.com**, **nodec.private.example.com**. and **noded.private.example.com**.

```
[root@nodeY ~]# yum install -y pcs fence-agents-rht
```

   1.3. Enable and start the **pcsd** service on your **nodea**, **nodeb**, **nodec**, and **noded** systems.

```
[root@nodeY ~]# systemctl enable pcsd
[root@nodeY ~]# systemctl start pcsd
```

1.4. Change the password of the **hacluster** system user to **redhat** on your **nodea**, **nodeb**, **nodec**, and **noded** machines.

```
[root@nodeY ~]# echo redhat | passwd --stdin hacluster
passwd: all authentication tokens updated successfully.
```

1.5. Authenticate the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, **nodec.private.example.com**, and **noded.private.example.com** on **nodea.private.example.com**.

```
[root@nodea ~]# pcs cluster auth nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com \
noded.private.example.com
Username: hacluster
Password: redhat
nodea.private.example.com: Authorized
nodeb.private.example.com: Authorized
nodec.private.example.com: Authorized
noded.private.example.com: Authorized
```

**2.** Configure a four-node cluster named **clusterX**, using your **nodea**, **nodeb**, **nodec**, and **noded** machines. Use the nodes' host names on the **private.example.com** network. Configure the cluster to use the **auto_tie_breaker** quorum option and to log to **/var/log/cluster.log** and then start the cluster.

2.1. Create the cluster **clusterX** with the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com** on **nodea.private.example.com**.

```
[root@nodea ~]# pcs cluster setup --name clusterX \
nodea.private.example.com \
nodeb.private.example.com \
nodec.private.example.com \
noded.private.example.com
```

2.2. Add the **auto_tie_breaker** option in the quorum section of the **/etc/corosync/corosync.conf** configuration file on **nodea**. Do not forget to also add the **auto_tie_breaker_node: lowest** option as well.

```
quorum {
provider: corosync_votequorum
auto_tie_breaker: 1
auto_tie_breaker_node: lowest
}
```

2.3. Change the configuration for **corosync** to log cluster messages to **/var/log/cluster.log**. In **/etc/corosync/corosync.conf**, change the **logging** block to the following:

```
logging {
  to_syslog: yes
  to_file: yes
  logfile: /var/log/cluster.log
}
```

2.4. Synchronize the **corosync.conf** configuration file from the current node to all other nodes in the cluster.

```
[root@nodea ~]# pcs cluster sync
```

2.5. Enable automatic startup of the cluster on all configured cluster nodes.

```
[root@nodea ~]# pcs cluster enable --all
```

2.6. Start the cluster.

```
[root@nodea ~]# pcs cluster start --all
```

2.7. Verify the cluster is running and all configured cluster nodes have joined the cluster.

```
[root@nodea ~]# pcs status
Cluster name: clusterX
WARNING: no stonith devices and stonith-enabled is not false
Last updated: Mon Sep 15 05:41:18 2014
Last change: Mon Sep 15 05:41:03 2014 via crmd on nodea.private.example.com
Stack: corosync
Current DC: nodea.private.example.com (1) - partition with quorum
Version: 1.1.10-29.el7-368c726
4 Nodes configured
0 Resources configured


Online: [ nodea.private.example.com nodeb.private.example.com
 nodec.private.example.com noded.private.example.com]

Full list of resources:


PCSD Status:
  nodea.private.example.com: Online
  nodeb.private.example.com: Online
  nodec.private.example.com: Online
  noded.private.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

3. Configure fencing for the virtual machines **nodea**, **nodeb**, **nodec** and **noded** that act as cluster nodes in the **clusterX** cluster. Use the fence_rht fencing agent, with **classroom.example.com** as the fencing device, and the host name of the machine to be fenced on the **private** network as the *plug*.

```
[root@nodea ~]# pcs stonith create fence_nodea \
fence_rht port="nodea.private.example.com" \
pcmk_host_list="nodea.private.example.com" \
ipaddr="classroom.example.com"
[root@nodeb ~]# pcs stonith create fence_nodeb \
fence_rht port="nodeb.private.example.com" \
pcmk_host_list="nodeb.private.example.com" \
ipaddr="classroom.example.com"
[root@nodec ~]# pcs stonith create fence_nodec \
fence_rht port="nodec.private.example.com" \
pcmk_host_list="nodec.private.example.com" \
ipaddr="classroom.example.com"
[root@noded ~]# pcs stonith create fence_nodec \
fence_rht port="noded.private.example.com" \
pcmk_host_list="noded.private.example.com" \
ipaddr="classroom.example.com"
```

4. Prepare each node to be an iSCSI initiator node.

   4.1. Install the *iscsi-initiator-utils* RPM.

```
[root@nodeY ~]# yum install -y iscsi-initiator-utils
```

   4.2. Create a unique IQN for the client initiator by modifying the **InitiatorName** setting in **/etc/iscsi/initiatorname.iscsi**. Use the client system name as the optional string after the colon.

```
[root@nodeY ~]# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2015-06.com.example:nodeY
```

   4.3. Enable and start the **iscsi** client service.

```
[root@nodeY ~]# systemctl enable iscsi; systemctl start iscsi
```

5. Configure the iSCSI initiator so that the **replacement_timeout** setting is **5** seconds.

   5.1. Edit the **node.session.timeo.replacement_timeout** setting in **/etc/iscsi/iscsid.conf** so it resembles the following.

```
node.session.timeo.replacement_timeout = 5
```

   5.2. Restart the **iscsi** client service for the timeout setting to take effect.

```
[root@nodeY ~]# systemctl restart iscsi
```

6. Discover and log into the configured target from the iSCSI target server.

6.1. Discover the configured iSCSI target(s) provided by the iSCSI target server portals.

```
[root@nodeY ~]# iscsiadm -m discovery -t st -p 192.168.1.9
192.168.1.9:3260,1 iqn.2015-06.com.example:cluster
```

```
[root@nodeY ~]# iscsiadm -m discovery -t st -p 192.168.2.9
192.168.2.9:3260,1 iqn.2015-06.com.example:cluster
```

6.2. Log into the presented iSCSI target.

```
[root@nodeY ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p 192.168.1.9
 -l
```

```
[root@nodeY ~]# iscsiadm -m node -T iqn.2015-06.com.example:cluster -p 192.168.2.9
 -l
```

**7.** Install the **device-mapper-multipath** package and enable multipath configuration on each cluster node.

7.1. Install the package on each cluster node.

```
[root@nodeY ~]# yum install -y device-mapper-multipath
```

7.2. Enable multipath configuration on each cluster node.

```
[root@nodeY ~]# mpathconf --enable
```

**8.** Configure multipathing on each node to ignore the local disk.

8.1. Modify each node's multipath configuration so that local disks are ignored by configuring the following **blacklist** statement in **/etc/multipath.conf**.

```
blacklist {
        devnode "^vd[a-z]"
}
```

8.2. Enable and start the **multipathd** service on each node.

```
[root@nodeY ~]# systemctl enable multipathd; systemctl start multipathd
```

**9.** Prepare the nodes for DLM and clvmd.

9.1. Install the **dlm** and **lvm2-cluster** packages on each node.

```
[root@nodeY ~]# yum -y install dlm lvm2-cluster
```

9.2. On all nodes, set the LVM **locking_type** to **3** (enable cluster locking).

```
[root@nodeY ~]# lvmconf --enable-cluster
```

9.3. On each node, stop the **lvmetad** service to match the modified LVM configuration.

```
[root@nodeY ~]# systemctl stop lvm2-lvmetad
    Warning: Stopping lvm2-lvmetad, but it can still be activated by:
                        lvm2-lvmetad.socket
```

9.4. Set the global Pacemaker parameter **no_quorum_policy** to **freeze**.

```
[root@nodea ~]# pcs property set no-quorum-policy=freeze
```

9.5. Create the **controld** resource. This is a required dependency for clvmd and GFS2.

```
[root@nodea ~]# pcs resource create dlm ocf:pacemaker:controld op monitor
 interval=30s on-fail=fence clone interleave=true ordered=true
```

9.6. Create the **clvm** resource.

```
[root@nodea ~]# pcs resource create clvmd ocf:heartbeat:clvm op monitor
 interval=30s on-fail=fence clone interleave=true ordered=true
```

9.7. Create resource constraints to control clvmd and DLM start up order and enforce them to run on the same node.

```
[root@nodea ~]# pcs constraint order start dlm-clone then clvmd-clone
[root@nodea ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

**10.** Create the clustered volume group and logical volume.

10.1. Create an LVM physical volume on **/dev/mapper/mpatha**.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
```

10.2. Create a clustered volume group named **clustervg**.

```
[root@nodea ~]# vgcreate -Ay -cy clustervg /dev/mapper/mpatha
```

10.3. Create a 1 GiB logical volume named **clusterlv**.

```
[root@nodea ~]# lvcreate -L 1G -n clusterlv clustervg
```

**11.** Format the logical volume with a GFS2 file system.

11.1. On each node, install the software needed to manage GFS2 file systems.

```
[root@nodeY ~]# yum -y install gfs2-utils
```

11.2. Format the logical volume with a GFS2 file system.

```
[root@nodea ~]# mkfs.gfs2 -j4 -p lock_dlm -t clusterX:weblog /dev/clustervg/
clusterlv
```

**12.** Create a Pacemaker cluster resource to manage your GFS2 file system and automatically mount it on all four nodes on cluster startup.

12.1. Configure the **Filesystem** cloned resource.

```
[root@nodea ~]# pcs resource create clusterfs Filesystem device="/dev/clustervg/
clusterlv" directory="/var/log/httpd" fstype="gfs2" options="noatime" op monitor
 interval=10s on-fail=fence clone interleave=true
```

12.2. Create constraints to mandate the startup ordering of the **clusterfs** cloned resource in relationship to the **clvmd** cloned resource, as well as to dictate that the two resources need to run on the same nodes.

```
[root@nodea ~]# pcs constraint order start clvmd-clone then clusterfs-clone
[root@nodea ~]# pcs constraint colocation add clusterfs-clone with clvmd-clone
```

**13.** Create a new resource group called **haweb**, using all resources and requirements outlined.

13.1. Install the **httpd** package on all four of your nodes.

```
[root@nodeY ~]# yum -y install httpd
```

13.2. Ensure that the contents of the mounted filesystem has the proper SELinux context for the **/var/log/httpd** directory.

```
[root@nodeY ~]# restorecon -R -v /var/log/httpd
restorecon reset /var/log/httpd context system_u:object_r:unlabeled_t:s0-
>system_u:object_r:httpd_log_t:s0
```

13.3. On all four of your nodes, configure SELinux so that **httpd** can serve files from NFS.

```
[root@nodeY ~]# setsebool -P httpd_use_nfs 1
```

13.4. On all four of your nodes, open a port on the firewall to allow **http** traffic.

```
[root@nodeY ~]# firewall-cmd --permanent --add-service=http
[root@nodeY ~]# firewall-cmd --reload
```

13.5. Create an **IPaddr2** resource, in the **haweb** resource group, for **172.25.*X*.80/24**.

```
[root@nodea ~]# pcs resource create webip IPaddr2 \
> ip=172.25.X.80 \
> cidr_netmask=24 \
> --group haweb
```

13.6. Create a **Filesystem** resource that read-only mounts the NFS share **workstation.storage1.example.com:/exports/www** on **/var/www**.

```
[root@nodea ~]# pcs resource create webfs Filesystem \
> device=workstation.storage1.example.com:/exports/www \
> directory=/var/www \
> fstype=nfs \
> options=ro \
> --group haweb
```

13.7. Create an **apache** resource in the **haweb** resource group.

```
[root@nodea ~]# pcs resource create webserver apache --group haweb
```

**14.** Configure the **haweb** resource group to start after the *clusterfs* resource.

```
[root@nodea ~]# pcs constraint order start clusterfs-clone then haweb
```

**15.** Configure the **haweb** resource group to automatically send an email with the subject "CLUSTER-NOTIFICATION" to **student@workstation.cluster*X*.example.com** whenever a status change occurs on that resource group. This resource should be named **webmail**.

```
[root@nodea ~]# pcs resource create webmail MailTo \
> email=student@workstation.cluster X.example.com \
> subject="CLUSTER-NOTIFICATION" \
> --group haweb
```

**16.** Add a location-constraint score of **INFINITY** for the **haweb** resource group for **noded.private.example.com**.

```
[root@nodea ~]# pcs constraint location haweb prefers noded.private.example.com
```

**17.** Run the grading script to verify your cluster.

```
[student@workstation ~]# lab hacluster grade
```