





# Red Hat High Availability Clustering



# **Red Hat Enterprise Linux 8.3 RH436**

## **Red Hat High Availability Clustering**

### **Edition 1 20210610**

### **Publication date 20210610**

Authors: Herve Quatremain, Mauricio Santacruz, Bernardo Gargallo  
Course Architects: Steven Bonneville, Philip Sweany  
DevOps Engineer: Artur Glogowski  
Editor: Julian Cable, David O'Brien

Copyright © 2021 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are  
Copyright © 2021 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to [training@redhat.com](mailto:training@redhat.com) or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: David Sacco, Roberto Velazquez, Sajith Eyamkuzhy, Victor Costea, Greg Hosler, Hemant Chauhan, Samik Sanyal

<b>Document Conventions</b>	<b>ix</b>
	ix
<b>Introduction</b>	<b>xi</b>
Red Hat High Availability Clustering .....	xi
Orientation to the Classroom Environment .....	xii
Performing Lab Exercises .....	xxii
<b>1. Creating High Availability Clusters</b>	<b>1</b>
Selecting a High Availability Cluster Configuration .....	2
Quiz: Selecting High Availability Clustering .....	6
Describing the Architecture of High Availability Clustering .....	8
Quiz: Describing the Architecture of High Availability Clustering .....	13
Configuring a Basic High Availability Cluster .....	15
Guided Exercise: Configuring a Basic Cluster .....	19
Lab: Creating High Availability Clusters .....	24
Summary .....	32
<b>2. Managing Cluster Nodes and Quorum</b>	<b>33</b>
Managing Cluster Membership .....	34
Guided Exercise: Managing Cluster Membership .....	39
Describing and Observing Quorum Operation .....	44
Guided Exercise: Describing and Observing Quorum Operation .....	48
Managing Quorum Calculations .....	53
Guided Exercise: Managing Quorum Calculations .....	56
Lab: Managing Cluster Nodes and Quorum .....	60
Summary .....	67
<b>3. Isolating Malfunctioning Cluster Nodes</b>	<b>69</b>
Protecting Data with Fencing .....	70
Quiz: Protecting Data with Fencing .....	74
Setting Up Fencing Devices .....	76
Guided Exercise: Setting Up Fencing Devices .....	80
Configuring Cluster Fencing Agents .....	83
Guided Exercise: Configuring Cluster Fencing Agents .....	87
Lab: Isolating Malfunctioning Cluster Nodes .....	91
Summary .....	96
<b>4. Creating and Configuring Resources</b>	<b>97</b>
Creating and Configuring a Cluster Resource .....	98
Guided Exercise: Creating and Configuring a Cluster Resource .....	102
Creating and Configuring Resource Groups .....	105
Guided Exercise: Creating and Configuring Resource Groups .....	110
Managing Resource Groups .....	114
Guided Exercise: Managing Resource Groups .....	116
Configuring Constraints .....	120
Guided Exercise: Configuring Constraints .....	126
Lab: Creating and Configuring Resources .....	130
Summary .....	136
<b>5. Troubleshooting High Availability Clusters</b>	<b>137</b>
Configuring Cluster Logging .....	138
Guided Exercise: Configuring Cluster Logging .....	142
Configuring Cluster Notifications .....	146
Guided Exercise: Configuring Cluster Notifications .....	150
Troubleshooting Resource Failures .....	156
Guided Exercise: Troubleshooting Resource Failures .....	159
Troubleshooting Cluster Networking .....	162

Guided Exercise: Troubleshooting Cluster Networking .....	164
Lab: Troubleshooting High Availability Clusters .....	168
Summary .....	176
<b>6. Automating Cluster and Resource Deployment</b>	<b>177</b>
Automating Cluster Deployment .....	178
Guided Exercise: Automating Cluster Deployment .....	182
Automating Resource and Resource Group Deployment .....	192
Guided Exercise: Automating Resource and Resource Group Deployment .....	195
Lab: Automating Cluster and Resource Deployment .....	201
Summary .....	211
<b>7. Managing Two-node Clusters</b>	<b>213</b>
Planning Two-node Cluster Deployments .....	214
Quiz: Planning Two-node Cluster Deployments .....	216
Configuring a Two-node Cluster .....	218
Guided Exercise: Configuring a Two-node Cluster .....	223
Configuring and Managing a Quorum Device .....	229
Guided Exercise: Configuring and Managing a Quorum Device .....	233
Lab: Managing Two-node Clusters .....	238
Summary .....	246
<b>8. Accessing iSCSI Storage</b>	<b>247</b>
Describing iSCSI Concepts .....	248
Quiz: Describing iSCSI Concepts .....	250
Accessing iSCSI Storage .....	252
Guided Exercise: Accessing iSCSI Storage .....	256
Lab: Accessing iSCSI Storage .....	260
Summary .....	265
<b>9. Accessing Storage Devices Resiliently</b>	<b>267</b>
Describing Device-Mapper Multipath .....	268
Quiz: Describing Device-Mapper Multipath .....	271
Configuring Resilient Storage Access .....	273
Guided Exercise: Configuring Resilient Storage Access .....	279
Testing Resilient Storage Access .....	283
Guided Exercise: Testing Resilient Storage Access .....	287
Lab: Accessing Storage Devices Resiliently .....	291
Summary .....	299
<b>10. Configuring LVM in Clusters</b>	<b>301</b>
Reviewing LVM Concepts .....	302
Quiz: Reviewing LVM Concepts .....	308
Managing High Availability LVM .....	310
Guided Exercise: Managing High Availability LVM .....	313
Managing LVM Shared Volume Groups .....	318
Guided Exercise: Managing LVM Shared Volume Groups .....	323
Lab: Configuring LVM in Clusters .....	329
Summary .....	338
<b>11. Providing Storage with the GFS2 Cluster File System</b>	<b>339</b>
Describing GFS2 Concepts .....	340
Quiz: Describing GFS2 Concepts .....	342
Creating a GFS2 Cluster File System .....	344
Guided Exercise: Creating a GFS2 Cluster File System .....	348
Managing a GFS2 Cluster File System .....	352
Guided Exercise: Managing a GFS2 Cluster File System .....	355
Lab: Providing Storage with the GFS2 Cluster File System .....	358

Summary .....	363
<b>12. Eliminating Single Points of Failure</b>	<b>365</b>
Configuring Network Redundancy for Cluster Communication .....	366
Guided Exercise: Configuring Network Redundancy for Cluster Communication .....	370
Configuring Multiple Fencing Device Levels .....	374
Guided Exercise: Configuring Multiple Fencing Device Levels .....	377
Lab: Eliminating Single Points of Failure .....	381
Summary .....	387
<b>13. Comprehensive Review</b>	<b>389</b>
Comprehensive Review .....	390
Lab: Configuring a High Availability Cluster .....	393
Lab: Configuring Cluster Resources on a Two-node Cluster .....	401
Lab: Deploying a GFS2 File System .....	411



# Document Conventions

---

This section describes various conventions and practices used throughout all Red Hat Training courses.

## Admonitions

Red Hat Training courses use the following admonitions:



### References

These describe where to find external documentation relevant to a subject.



### Note

These 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 something that makes your life easier.



### Important

These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.



### Warning

These should not be ignored. Ignoring these admonitions will most likely cause data loss.

## Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.



# 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 Logical Volume Manager (LVM) Shared Volume Groups, 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.

---

### Course Objectives

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

### Audience

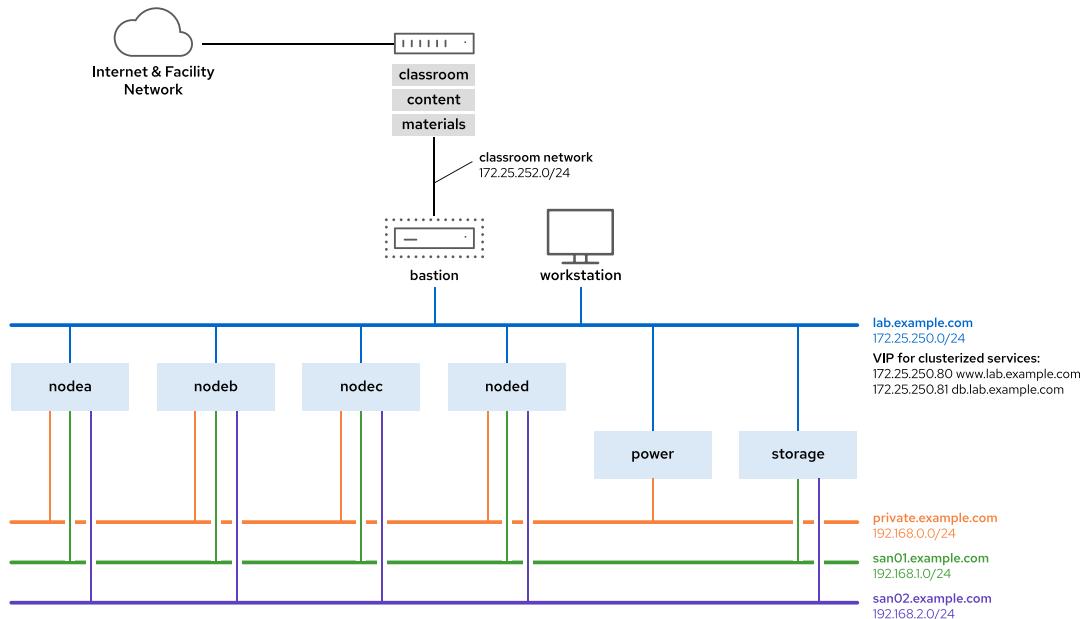
- A senior Linux system administrator responsible for maximizing resiliency through high availability clustering services and using fault-tolerant shared storage technologies.
- An RHCSA/RHCE interested in earning a Red Hat Certification of Expertise or a Red Hat Certified Architect (RHCA).

### Prerequisites

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

# Orientation to the Classroom Environment

## Classroom Environment



**Figure 0.1: Classroom environment**

In this course, the main computer system that is used for hands-on learning activities is **workstation**. The **workstation** virtual machine (VM) is the only one with a graphical desktop. You should always log in directly to **workstation** first.

From **workstation**, use SSH for command-line access to all other VMs. The **workstation** machine has a standard user account, **student** with the password **student**. The **student** user account can become the **root** user if it is necessary. No exercise in this course requires that you log in directly as **root**, but if you must, then the password is **redhat**.

You can also use four other machines for these activities: **nodea**, **nodeb**, **nodec**, and **noded**. For these machines, the password for the **student** user is **student**, with the same privileges as in the **workstation** machine.

As you can see in *Figure 0.1*, all VMs share the **lab.example.com** DNS domain (172.25.250.0/24). Next to the **lab.example.com** network, three other networks are also in use: **private.example.com** (192.168.0.0/24) for private cluster communications, and **san01.example.com** (192.168.1.0/24) and **san02.example.com** (192.168.2.0/24) for iSCSI and NFS storage traffic.

In the classroom environment, only the **private.example.com** network is used for private cluster communications. The private cluster communication network is critical in the cluster infrastructure, because if this network fails, then the whole cluster fails. For this reason, Red Hat recommends to improve the cluster resiliency by using network redundancy in production environments. Network redundancy in the cluster is covered later in this course.

## Introduction

The system named **bastion** must always be running. The **bastion** system acts as a router between the network that connects your lab machines and the classroom network. If **bastion** is down, other lab machines might not function properly or might even hang during boot.

Several systems in the classroom provide supporting services. Two servers, **content.example.com** and **materials.example.com**, are sources for software and lab materials that are used in hands-on activities. An iSCSI and NFS storage server **storage.lab.example.com** is also provided. Information on how to use these servers is provided in the instructions for those activities.

The Classroom Machines table provides information for the different machines that are used in the classroom environment, including their role and IP addresses.

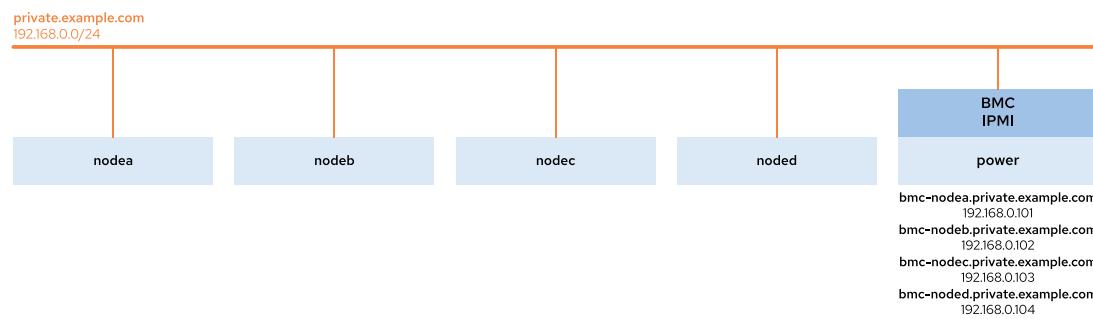
### Classroom Machines

Machine name	IP addresses	Role
workstation.lab.example.com	172.25.250.9	Graphical workstation for system administration.
classroom.example.com	172.25.252.254	Router to link the Classroom Network to the Internet.
bastion.lab.example.com	172.25.250.254 172.25.252.1	Router to link VMs to central servers.
power	172.25.250.100 192.168.0.100	Machine to simulate the fencing devices through BMC or chassis.
nodea.lab.example.com	172.25.250.10 192.168.0.10 192.168.1.10 192.168.2.10	Cluster node A.
nodeb.lab.example.com	172.25.250.11 192.168.0.11 192.168.1.11 192.168.2.11	Cluster node B.
nodec.lab.example.com	172.25.250.12 192.168.0.12 192.168.1.12 192.168.2.12	Cluster node C.
noded.lab.example.com	172.25.250.13 192.168.0.13 192.168.1.13 192.168.2.13	Cluster node D. In some exercises, it is used as the quorum device.
storage.lab.example.com	172.25.250.15 192.168.1.15 192.168.2.15	iSCSI and NFS storage server.

## Fencing Environment

Fencing is one important part of a high availability cluster. It is used to restrict the access of an unresponsive node to the cluster resources. Fencing is explained in detail later in this course, but it is introduced here from a network perspective. Two different fencing methods are used during this course:

### Fencing through simulated BMC



**Figure 0.2: Simulated BMC environment**

Because virtual machines have no embedded Baseboard Management Controller (BMC) device for power management, the BMC function is simulated by using the **power** machine. The simulated BMC mechanism performs monitoring and management tasks remotely on the cluster nodes.

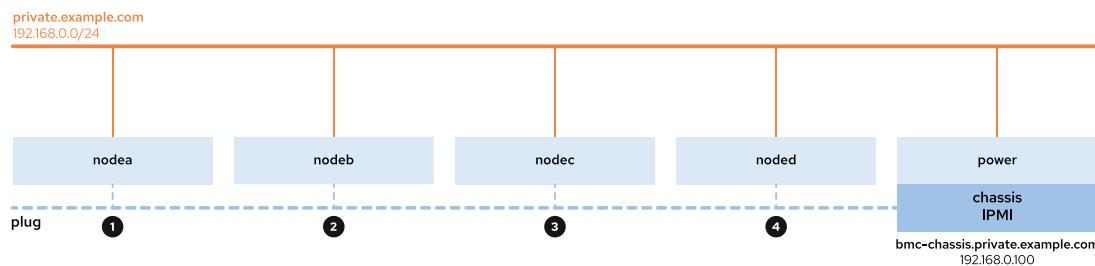
The IP addresses of the BMC devices (192.168.0.101, 192.168.0.102, 192.168.0.103, and 192.168.0.104, for nodea, nodeb, nodec, and noded, respectively) are hosted on **power**.

The BMC IP addresses and node names are assigned when the classroom environment is built. The `openstackbmc` service runs on the **power** machine with one process for each power-managed cluster node. This service responds to the Intelligent Platform Management Interface (IPMI) requests on behalf of the corresponding node.

```
[root@power ~]# systemctl status openstackbmc
openstackbmc.service - OpenStack BMC using fakeipmi
   Loaded: loaded (/etc/systemd/system/openstackbmc.service; enabled; vendor
   preset: disabled)
     Active: active (running) since Tue 2021-05-18 03:01:24 EDT; 2h 21m ago
       Process: 912 ExecStart=/usr/local/bin/openstackbmc-wrap.bash (code=exited,
      Status: 0/SUCCESS)
      Tasks: 13 (limit: 5036)
     Memory: 171.8M
      CGroup: /system.slice/openstackbmc.service
              |-1285 python /usr/local/bin/openstackbmc.py --project-name=ole-
fb6154cc-2848-48b8-b81b-ac78d341a3cb --vm-name=noded
              |-1288 python /usr/local/bin/openstackbmc.py --project-name=ole-
fb6154cc-2848-48b8-b81b-ac78d341a3cb --vm-name=nodec
              |-1291 python /usr/local/bin/openstackbmc.py --project-name=ole-
fb6154cc-2848-48b8-b81b-ac78d341a3cb --vm-name=nodeb
              |-1294 python /usr/local/bin/openstackbmc.py --project-name=ole-
fb6154cc-2848-48b8-b81b-ac78d341a3cb --vm-name=nodea
...
...output omitted...
```

For all the BMC devices, the login is admin and the password is password.

### Fencing through simulated chassis



**Figure 0.3: Simulated chassis**

The second fencing method that is used in this course simulates a *management chassis* (such as ibmblade, hpblade, or bladecenter). With this method, only the chassis IP address is needed to request fencing.

This fencing method uses the `fence_rh436` custom script and the `pcmk_host_map` parameter to assign a *plug* number to each cluster node. A request that is sent to the chassis IP (192.168.0.100) includes the node to fence, specified by the *plug* number. The `fence_rh436` script converts the request to an IPMI call to the *Fencing through simulated BMC* method.

```
[root@nodeX ~]# cat /usr/sbin/fence_rh436
...output omitted...
ip_name_mapping = {
    "nodea": socket.gethostbyname("bmc-nodea"),
    "nodeb": socket.gethostbyname("bmc-nodeb"),
    "nodec": socket.gethostbyname("bmc-nodec"),
    "noded": socket.gethostbyname("bmc-noded"),
}
...output omitted...
```

In this classroom, the power machine, to simulate a chassis, performs the request, such as powering off the virtual machine that is associated with the node to fence.

## Managing the High Availability Cluster in the Classroom

Procedures for managing high availability clusters in the classrooms are different from production environments. Typically, high availability clusters require resilient, low-latency communication between all nodes. Nodes can be taken offline or rebooted for maintenance, with no functional loss. In production, the full environment is rarely, if ever, shut down completely.

**Note**

Always refer to current Red Hat High Availability Add-On documentation for the supported start and stop procedures for production environments. Classroom procedures that are discussed here might include shortcuts or exclude recommended procedures that are acceptable only for this custom classroom environment.

For more information about performing cluster maintenance, see *Chapter 29. Performing cluster maintenance in the Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_cluster-maintenance-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_cluster-maintenance-configuring-and-managing-high-availability-clusters)

You can also refer to Knowledgebase: "Recommended Practices for Applying Software Updates to a RHEL High Availability or Resilient Storage Cluster" [<https://access.redhat.com/articles/2059253>]

The major difference in the classroom environment is that it uses virtual machines that are either deployed online or on a single physical system, while production environments typically use bare metal systems. Most training locations, both online and physical, automatically shut down student systems after timed use or after class hours.

Although automated shutdowns are not graceful, current high availability clusters are resilient enough that classrooms restart and operate without difficulty.

## Shutting Down the Classroom Environment

A high availability cluster is an infrastructure that is designed for long-term operation. However, you might need to reboot nodes in the cluster. If you need to reboot all the nodes, then it is advisable to reboot each node individually one after another, because rebooting all nodes at the same time can cause service downtime.

Recommended classroom practice is to gracefully stop your high availability cluster when it is finished for an extended time. To stop the cluster services, execute the following command:

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

If you forget to shut down manually, then your course environment might time out and be shut down automatically. If you encounter major issues with restarting your environment, then discard and reprovision your student environment by using the correct procedure for the delivery environment.

## Resetting Your Classroom Environment

*Resetting your classroom environment* is the procedure to set some or all of your classroom nodes back to their beginning state when the course was first created. By resetting, you can clean your virtual machines and start the exercises over again. It is also a simple method for clearing a classroom issue that is blocking your progress and is not easily solved.

This classroom has some specific constraints when you want to reset part or all of your environment. In most Red Hat Training courses, individual systems can be reset separately as needed. However, in this course, resetting only a single cluster node results in that node losing

## Introduction

necessary information and failing to communicate with other nodes as part of the cluster after resetting it.

Thus, the correct procedure is to remove first the node from your cluster and then to reset it. Removing a cluster node from an existing cluster is a two-step process.

First, remove the node from the cluster:

```
[root@oldnode ~]# pcs cluster remove newnode.example.com  
...output omitted...
```

Then, 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@oldnode ~]# pcs stonith delete fence_deletednode  
...output omitted...
```

After resetting the node, you must configure that node again as part of your cluster. Thus, you must install all the required packages again on the node, authorize it in the cluster, open firewall ports, start the pcscd service, and configure it as part of the cluster.

The commands for resetting individual nodes are discussed in the upcoming section named "Controlling Your Systems".

## The Nodes of the High Availability Cluster

Some classroom VMs are not modified during exercises, and never need to be reset unless you are solving a technical problem. For example, the `workstation` machine would need to be reset only if it became unstable or out of communication, and could be reset by itself. This table lists the machines that are never intended to be reset and those machines that can be reset if necessary:

### Which machines are normally reset or not reset?

Typically do not need to be reset	If required, can be reset
<ul style="list-style-type: none"><li>• bastion</li><li>• classroom</li><li>• power</li></ul>	<ul style="list-style-type: none"><li>• nodea</li><li>• nodeb</li><li>• nodec</li><li>• noded</li><li>• storage</li></ul>

In the online environment, you can reset the selected machine by clicking ACTION → Reset.

**Note**

If you reset the power machine, then the fencing resources might fail and stop when it times out. In that case, you must reset the fail count for that resource, to enable its use on the cluster again. For this purpose, execute the following command:

```
[root@node ~]# pcs resource cleanup my_resource
```

You can also reset the classroom environment by re-creating the original course build. Re-creating the course is quick, typically taking only a few minutes, and results in a clean, working environment.

**Warning**

For the sake of time, re-creating the course is useful to avoid troubleshooting in a classroom environment. However, in a production environment, probably you cannot start your cluster from the outset. Thus, you must troubleshoot your cluster so that it is ready again.

In the online environment, click the **DELETE** button. Wait, and then click the **CREATE** button.

## Controlling Your Systems

You are assigned remote computers in a Red Hat Online Learning classroom. Self-paced courses are accessed through a web application hosted at [rol.redhat.com](http://rol.redhat.com) [<http://rol.redhat.com>]. If your course is an instructor-led virtual training, you will be provided with your course location URL. Log in to this site using your Red Hat Customer Portal user credentials.

## Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through web page interface controls. The state of each classroom virtual machine is displayed on the **Lab Environment** tab.

The screenshot shows a user interface for managing a lab environment. At the top, there are tabs for 'Table of Contents', 'Course', 'Lab Environment', and icons for a star and a question mark. Below this is a section titled 'Lab Controls' with instructions: 'Click CREATE to build all of the virtual machines needed for the classroom lab environment. This may take several minutes to complete. Once created the environment can then be stopped and restarted to pause your experience.' It also states: 'If you DELETE your lab, you will remove all of the virtual machines in your classroom and lose all of your progress.' Below these instructions are three buttons: 'DELETE' (red), 'STOP' (teal), and an information icon (blue). The main area displays a table of virtual machines:

Machine Name	Status	Action	Console
bastion	active	ACTION -	OPEN CONSOLE
classroom	active	ACTION -	OPEN CONSOLE
nodea	active	ACTION -	OPEN CONSOLE
nodeb	active	ACTION -	OPEN CONSOLE
nodec	active	ACTION -	OPEN CONSOLE
noded	active	ACTION -	OPEN CONSOLE
power	active	ACTION -	OPEN CONSOLE
storage	active	ACTION -	OPEN CONSOLE
utility	active	ACTION -	OPEN CONSOLE
workstation	active	ACTION -	OPEN CONSOLE

Figure 0.4: An example course Lab Environment management page

### Machine States

Virtual Machine State	Description
building	The virtual machine is being created.
active	The virtual machine is running and available. If just started, it may still be starting services.
stopped	The virtual machine is completely shut down. Upon starting, the virtual machine boots into the same state as it was before it was shut down. The disk state is preserved.

### Classroom Actions

Button or Action	Description
CREATE	Create the ROLE classroom. Creates and starts all of the virtual machines needed for this classroom.
CREATING	The ROLE classroom virtual machines are being created. Creates and starts all of the virtual machines needed for this classroom. Creation can take several minutes to complete.

Button or Action	Description
DELETE	Delete the ROLE classroom. Destroys all virtual machines in the classroom. <b>All work saved on that system's disks is lost.</b>
START	Start all virtual machines in the classroom.
STARTING	All virtual machines in the classroom are starting.
STOP	Stop all virtual machines in the classroom.

### Machine Actions

Button or Action	Description
OPEN CONSOLE	Connect to the system console of the virtual machine in a new browser tab. You can log in directly to the virtual machine and run commands, when required. Normally, log in to the <b>workstation</b> virtual machine only, and from there, use ssh to connect to the other virtual machines.
ACTION → Start	Start (power on) the virtual machine.
ACTION → Shutdown	Gracefully shut down the virtual machine, preserving disk contents.
ACTION → Power Off	Forcefully shut down the virtual machine, while still preserving disk contents. This is equivalent to removing the power from a physical machine.
ACTION → Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. <b>All work saved on that system's disks is lost.</b>

At the start of an exercise, if instructed to reset a single virtual machine node, click **ACTION → Reset** for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION → Reset** on every virtual machine in the list.

If you want to return the classroom environment to its original state at the start of the course, you can click **DELETE** to remove the entire classroom environment. After the lab has been deleted, you can click **CREATE** to provision a new set of classroom systems.



#### Warning

The **DELETE** operation cannot be undone. All work you have completed in the classroom environment will be lost.

## The Auto-stop and Auto-destroy Timers

The Red Hat Online Learning enrollment entitles you to a set allotment of computer time. To help conserve your allotted time, the ROLE classroom uses timers, which shut down or delete the classroom when the appropriate timer expires.

To adjust the timers, locate the two + buttons at the bottom of the course management page. Click the auto-stop + button to add another hour to the auto-stop timer. Click the auto-destroy +

button to add another day to the auto-destroy timer. There is a maximum for auto-stop at 11 hours, and a maximum auto-destroy at 14 days. Be careful to keep the timers set while you are working, so as to not have your environment unexpectedly shut down. Be careful not to set the timers unnecessarily high, which could waste your subscription time allotment.

# Performing Lab Exercises

Run the `lab` command from `workstation` to prepare your environment before each hands-on exercise, and again to clean up after an exercise. Each hands-on exercise has a unique name within a course.

There are two types of exercises. The first type, a *guided exercise*, is a practice exercise that follows a course narrative. If a narrative is followed by a quiz, it usually indicates that the topic did not have an achievable practice exercise. The second type, an *end-of-chapter lab*, is a gradable exercise to help to verify your learning. When a course includes a comprehensive review, the review exercises are structured as gradable labs.

The syntax for running an exercise script is as follows:

```
[student@workstation ~]$ lab action exercise
```

The `action` is a choice of `start`, `grade`, or `finish`. All exercises support `start` and `finish`. Only end-of-chapter labs and comprehensive review labs support `grade`.

## **start**

A script's start logic verifies the required resources to begin an exercise. It might include configuring settings, creating resources, checking prerequisite services, and verifying necessary outcomes from previous exercises. With exercise start logic, you can perform any exercise at any time, even if you did not perform prerequisite exercises.

## **grade**

End-of-chapter labs help to verify what you learned, after practicing with earlier guided exercises. The grade action directs the `lab` command to display a list of grading criteria, with a PASS or FAIL status for each. If the grade action shows a FAIL status, then it also shows information so you can detect which step is failing. Moreover, it logs the errors as it is explained in the next section. To achieve a PASS status for all criteria, fix the failures and rerun the grade action.

## **finish**

A script's finish logic deletes exercise resources that are no longer necessary. With cleanup logic, you can repeatedly perform an exercise, and it helps course performance by ensuring that unneeded objects release their resources.

To list the available exercises, use tab completion in the `lab` command with an action:

```
[student@workstation ~]$ lab start Tab Tab
auto-ha           gfs-manage          resources-group
auto-res          gfs-review          resources-manage
auto-review       iscsi-initiator    resources-review
cluster-create   iscsi-review        spof-fencing
```

## Troubleshooting Lab Scripts

All the exercise scripts are stored on the `workstation` machine in the folder `/home/student/.venv/labs/lib/python3.6/site-packages/rh436/ansible`. When you run

## Introduction

the `lab` command with a valid exercise and action, it creates one log file in `/tmp/log/labs`, plus the directory if it does not exist. The file, named `exercise`, captures error messages that are normally displayed on your terminal.

```
[student@workstation ~]$ ls -l /tmp/log/labs/
-rw-rw-r-- 1 student student 7520 Apr 30 05:34 cluster-review
```

## Interpreting the Exercise Log Files

Exercise scripts send output to the log file when the scripts fail. Thus, the exercise log is useful for troubleshooting. Because the exercise scripts are based on Ansible Playbooks, system administrators should recognize common error message entries.

Although exercise scripts are always run from `workstation`, they perform tasks on other systems in the course environment. Many course environments, including OpenStack and OpenShift, use a command-line interface (CLI) that is invoked from `workstation` to communicate with server systems by using API calls. Because script actions typically distribute tasks to multiple systems, additional troubleshooting is necessary to determine where a failed task occurred. Log in to those other systems and use Linux diagnostic skills to read local system log files and determine the root cause of the lab script failure.



## 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 Red Hat Enterprise Linux high availability cluster.
- Install and start a small high availability cluster.

### Sections

- Selecting High Availability Clustering (and Quiz)
- Describing the Architecture of High Availability Clustering (and Quiz)
- Configuring a Basic Cluster (and Guided Exercise)

### Lab

- Creating High Availability Clusters

# Selecting a High Availability Cluster Configuration

## Objectives

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

## Introducing High Availability Cluster Configurations

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.

The goal of a *high availability* cluster, also known as an *HA* cluster, is to keep running services as available as possible by eliminating bottlenecks and single points of failure. 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. This is a different strategy from 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.

High availability cluster configurations can be grouped into two subsets:

- **Active-active** high availability clusters, where a service runs on multiple nodes, thus leading to shorter failover times. The main goal of these type of clusters is to perform load balancing, and thus they can scale to many instances handling higher loads. However, they require load balancer devices to perform this load balancing. Active-active HA clusters can be as small as only two nodes. If one node fails, then the cluster redirects all the work load to the rest of the nodes. When the failing node recovers, then the cluster distributes again the work load between the available nodes.

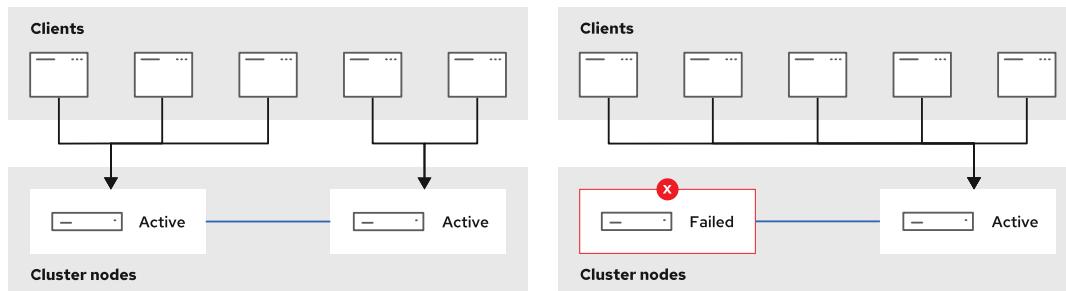


Figure 1.1: Active-active High Availability Cluster Configuration

- **Active-passive** high availability clusters, where a service only runs on one node at a time. When one node fails, then and only then the cluster starts the service on another one to replace it. This configuration requires *fencing* to restrict the access of an unresponsive node to the cluster resources, avoiding corruption on the cluster. Fencing is covered more in detail later in this course.

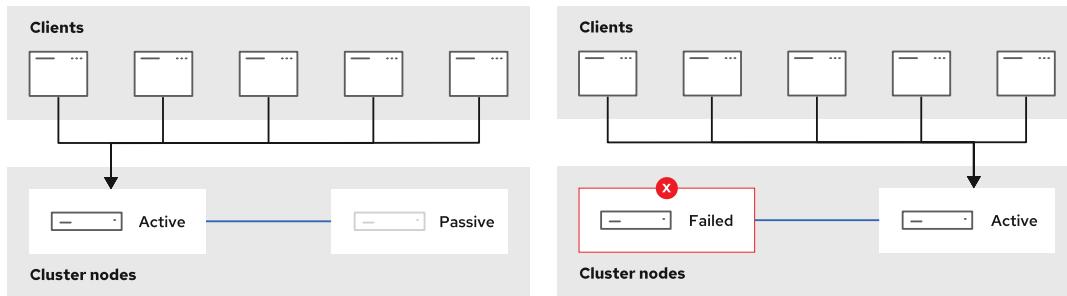


Figure 1.2: Active-passive High Availability Cluster Configuration



### Note

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

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.

System administrators should decide what is the cluster configuration that best fits their requirements. For example, many common applications already have service redundancy built-in because they have specific requirements that are best handled by being built-in. Examples include LDAP with primary/secondary, or databases with multiprimary partitioning and scaling. Thus, these applications are not appropriated for using the Red Hat High Availability Add-On to provide service redundancy. However, you can still use these services as resources for setting up resource groups for other services.

## When to Use the High Availability Add-On for 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 might not benefit from putting it on an HA cluster. For example, the DNS or LDAP services can use multiple servers with a primary/secondary or multiprimary relation. The services can be configured for data replication between primary and secondary servers. Clients of DNS and LDAP can use multiple servers. There is less failover delay involved in a primary/secondary or multiprimary configuration, and so the availability of the service does not increase when putting them on an HA cluster. However, within an OpenStack platform solution, it might 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 such as 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 fail again.
- High availability clusters do not provide end-to-end redundancy. Although the cluster itself might be fully operational, if a network error in the infrastructure causes the cluster to be unreachable, the clients will not be able to reach the service, even though the service runs on a high availability cluster. Therefore, it is important to consider the cluster's architecture and design it to avoid single points of failure throughout the deployment. There are trade-offs here, and the cluster architect needs to consider what level of risk they are prepared to tolerate with each component of the cluster.

## Components of a High Availability Cluster

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, file system, or database are all considered to be 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, you need to set up a web server daemon, the data that the server must share, and the IP address assigned to the service. All these resources need to be available on the same cluster node, and thus you should combine them into a resource group.

### Describing 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*.

### Describing Fencing

Fencing is a mechanism that ensures a malfunctioning cluster node cannot cause corruption on the cluster, because the node could still have access to the cluster resources. Fencing also allows the cluster to safely recover its resources on another node. This is necessary because you cannot assume that an unreachable node is actually off. Fencing is often accomplished by powering the node off, because a dead node is clearly not able to do anything. In other cases, a combination of operations is used to cut the node off from the network (to stop the cluster allocating resources on that node) or from storage (to stop the node from writing to shared storage).

### Shared Storage

Most high availability clusters 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 can 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. Fencing improves data integrity.

## Describing 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 at least 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, then by default no resources or resource groups are started and running resource groups are stopped to ensure data integrity.



### References

For more information, refer to the *High Availability Add-On Overview* chapter in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_overview-of-high-availability-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_overview-of-high-availability-configuring-and-managing-high-availability-clusters)

## ► Quiz

# Selecting High Availability Clustering

Match the items below 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

# Selecting High Availability Clustering

Match the items below 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

# Describing the Architecture of High Availability Clustering

## Objectives

After completing this section, you should be able to identify the components of a Red Hat Enterprise Linux high availability cluster.

## Describing the Hardware Configuration of an HA Cluster

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

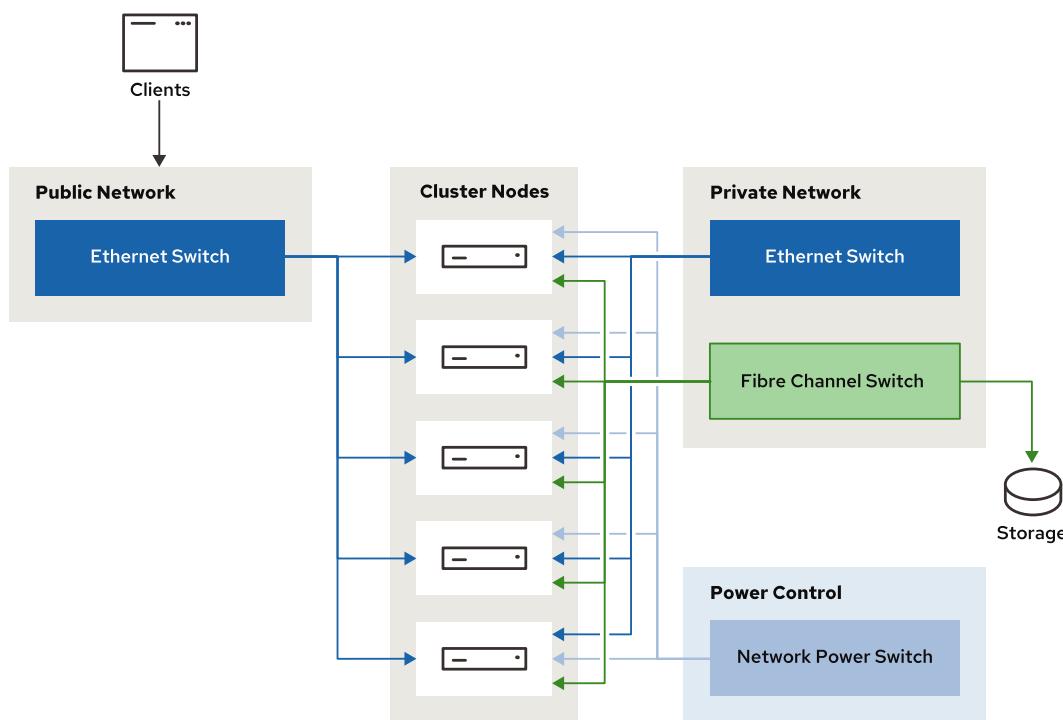


Figure 1.3: A Typical Cluster Infrastructure

The different components in this infrastructure are as follows:

### Cluster Nodes

These are the machines that run the cluster software and the services.

### Public Network

The clients use this network for communicating with the services running on the cluster. Services normally have a *floating* IP address that the cluster assigns to whichever node is currently running the corresponding service.

### Private Network

The cluster uses this network for its inter-node communication.

### **Networked Power Switch**

It is necessary to remotely control power to the cluster nodes through, for example, a networked power switch. This is one of the possible ways to implement *power fencing* as described later in this course. Use remote management cards such as Integrated Lights-out (iLO) or Dell Remote Access Card (DRAC) for this purpose.

### **Fibre Channel Switch**

In the figure showing a typical cluster infrastructure, the same shared storage connects to all nodes at the same time. It is usual to use a Fibre Channel for this purpose. An alternative method would be a separate Ethernet network with iSCSI or FCoE.

In the figure, only the components on the left side of the cluster nodes are publicly accessible. Everything on the right side of the cluster nodes is strictly *private*, and should not be reachable from the public network.

## **Describing the Software Configuration of an HA Cluster**

Cluster nodes require multiple software components to provide cluster services with the Red Hat High Availability Add-On. An overview of these components and their functionality follows:

### **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.

### **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:

- Cluster Information Base (CIB): The CIB 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. The scheduler (`pacemaker -schedulerd`) uses the contents of the CIB to compute the ideal state of the cluster and how to reach it.
- 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.
- 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 devices configured in the CIB.

### **pcs**

The pcs RPM package contains two cluster configuration tools:

- The `pcs` command provides a command-line interface to create, configure, and control every aspect of a Pacemaker/Corosync cluster.
- The `pcsd` service provides the cluster configuration synchronization and a web front end to create and configure a Pacemaker/Corosync cluster.

## HA Cluster 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. Red Hat can assist you to evaluate the performance of an existing cluster or to build a new cluster. The process 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 then decides if it supports the cluster configuration.

System administrators should consider some important requirements and recommendations 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 32 nodes for Red Hat Enterprise Linux 8.1 and later. In the case of using RHEL 8.0 or earlier versions, or the Resilient Storage Add-On, then the supported number of nodes is up to 16 nodes.

Clusters consisting of only one or two nodes are special cases.

RHEL 8.2 and later versions support single-node clusters. However, fencing is not available and thus single-node clusters do not support file systems that require it, such as DLM and GFS2.

RHEL 8.1 and earlier versions only support clusters with two or more nodes. Red Hat supports two-node clusters in most cases but recommends submitting the cluster design and consulting Red Hat Support 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. Multisite clusters do require special consideration in their design. Red Hat Enterprise Linux 8 High Availability Add-On supports multisite clusters.

Stretch clusters, also known as geo clusters, are clusters stretched out over multiple physical locations. Red Hat does not treat these clusters as a special class of deployment with separate rules. Thus, standard policies, requirements, and limitations that apply to RHEL high availability clusters bind them. However, stretch clusters do have inherent complexities, and Red Hat Support highly recommends submitting the cluster design before deploying them.

### Fencing Hardware

Fencing is a mechanism that ensures that a malfunctioning cluster node cannot cause corruption. Thus, it is possible to safely recover its resources 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 you use supported hardware. If the cluster uses integrated fencing devices like iLO or DRAC, then systems acting as cluster nodes must power off immediately when a shutdown signal is received, instead of initiating a clean shutdown.

## Virtualized and Cloud Environments

Red Hat supports using virtual machines as cluster members on the most popular virtual environments and cloud providers.

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 8 libvirt-based system, Red Hat 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, then it crashes all the cluster node VMs running on it.

## Networking

In RHEL 8, the `corosync` component uses the unicast transport protocol `kronosnet` for the default network communication on the private network. For the public network, gratuitous ARP is used for the floating IP addresses. The network switch must support this.

On the public network, it is necessary to open the network ports required by the services running on the cluster. For correct operation, you should also open the following ports on the private network:

- `pcsd` port 2224/TCP
- `corosync` ports 5404–5412/UDP

Optional cluster components require additional ports. For example, the GFS2 file system uses port 21064/TCP, the quorum device uses port 5403/TCP, and the Booth ticket manager uses port 9929/TCP and UDP. Some of these components are described in other chapters.



### Note

Booth ticket manager is a distributed service which facilitates support of multisite clusters.

For more information about configuring multisite clusters with Pacemaker using Booth ticket manager, see the documentation at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-multisite-cluster-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-multisite-cluster-configuring-and-managing-high-availability-clusters)

## SELinux Support

Red Hat Enterprise Linux High Availability Add-On supports using SELinux in `enforcing` mode when using the `targeted` policy on the cluster nodes.

## Planning for Failures

All hardware eventually fails. Hardware life cycle can range from weeks to years. Furthermore, almost every single (complex) piece of software has bugs. Some might be unnoticeable, and others might corrupt an entire database. One of the major tasks for a system administrator is to acknowledge that these failures occur, and plan accordingly.

When the failing piece of hardware is a simple desktop machine, the correct approach is most likely to replace the failed machine, although for a mission-critical server a more proactive approach is needed. When a machine fails, the service running on that machine should not fail.

## Chapter 1 | Creating High Availability Clusters

A single point of failure (SPOF) is any part of a complex setup that, when it fails, can take down an entire environment. A typical high availability cluster can have many possible single points of failure. The following are by no means exhaustive lists, but they do contain the most common offenders.

### Hardware Single Points of Failure

- Power supply
- Local storage
- Network interfaces
- Network switches
- Fencing software

### Software Single Points of Failure

- Cluster communications
- Shared storage connection
- Software fencing configuration



#### References

**Knowledgebase: "How can Red Hat assist me in assessing the design of my RHEL High Availability or Resilient Storage cluster?"**

<https://access.redhat.com/articles/2359891>

**Knowledgebase: "Support Policies for RHEL High Availability Clusters"**

<https://access.redhat.com/articles/2912891>

## ► Quiz

# Describing the Architecture of High Availability Clustering

Choose the correct answers to the following questions:

- ▶ 1. **What is the maximum number of cluster nodes supported by Red Hat High Availability Add-On in RHEL 8.1 or later, if the cluster does not use the Resilient Storage Add-On?**
  - a. 4
  - b. 8
  - c. 16
  - d. 32
  
- ▶ 2. **Which two types of fencing are supported by the Red Hat High Availability Add-On? (Choose two.)**
  - a. Compute 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. **Which of the following plans of action is the best if you have doubts about the proposed cluster architecture for your environment?**
  - a. Contact Red Hat Support for guidance.
  - b. Deploy it anyway.
  - c. Discontinue the cluster project.

## ► Solution

---

# Describing the Architecture of High Availability Clustering

Choose the correct answers to the following questions:

- ▶ 1. **What is the maximum number of cluster nodes supported by Red Hat High Availability Add-On in RHEL 8.1 or later, if the cluster does not use the Resilient Storage Add-On?**
  - a. 4
  - b. 8
  - c. 16
  - d. 32
  
- ▶ 2. **Which two types of fencing are supported by the Red Hat High Availability Add-On? (Choose two.)**
  - a. Compute 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. **Which of the following plans of action is the best if you have doubts about the proposed cluster architecture for your environment?**
  - a. Contact Red Hat Support for guidance.
  - b. Deploy it anyway.
  - c. Discontinue the cluster project.

# Configuring a Basic High Availability Cluster

## Objectives

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

## Installing the Node Software

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



### Important

Red Hat Enterprise Linux 8 and Red Hat Enterprise Linux 7 cluster nodes are not compatible. All nodes in a Pacemaker cluster must use the same major version of Red Hat Enterprise Linux.

Red Hat Enterprise Linux 8 clusters use Corosync 3.x for communication, though Red Hat Enterprise Linux 7 Pacemaker clusters use Corosync 2.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 only 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 environment includes a Baseboard Management Controller (BMC) that you can use to power off, power on, or restart the machine using IPMI over LAN commands. To use the BMC on the cluster, the `fence-agents-ipmilan` package needs to be installed on all the cluster nodes.

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

## Configuring a Firewall for Cluster Communication

You need to allow cluster communications through the firewall on all cluster nodes. The standard firewall service on a Red Hat Enterprise Linux 8 system is `firewalld`. The `firewalld` daemon ships with a standard service called `high-availability` to allow cluster communication. To activate the `high-availability` firewall service on each of the cluster nodes to allow cluster communication through the firewall, execute:

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

## Enabling Pacemaker and Corosync on the Nodes

The pcsd service provides the cluster configuration synchronization and the web front end for cluster configuration. The service is required on all cluster nodes. Use the `systemctl` command to start and enable the `pcsd` service on all cluster nodes.

```
[root@node ~]# systemctl enable --now pcsd
```

`pcsd` uses the system user `hacluster` for cluster communication and configuration. Red Hat recommends that you use the same password for the `hacluster` user on all nodes in the cluster. You need to set the password of the `hacluster` system user on all cluster nodes. The following example sets the `hacluster` user password to `redhat`.

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

You need to authenticate the cluster nodes in the `pcsd` service with the user `hacluster` and the password that you set up for this user. You only need to run the `pcs host auth` command on one node to authenticate all of the nodes in the cluster.

The cluster nodes `node1.example.com`, `node2.example.com`, and `node3.example.com` are authenticated on the `node1.example.com` system with the user `hacluster` and the corresponding password.

```
[root@node ~]# pcs host auth node1.example.com \  
> node2.example.com \  
> node3.example.com  
Username: hacluster  
Password: redhat  
node1.example.com: Authorized  
node2.example.com: Authorized  
node3.example.com: Authorized
```

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

## Configuring Basic Cluster Communication

After preparing the three nodes for the cluster setup, the `pcs cluster setup` command creates the cluster. This command takes the cluster name and fully qualified domain names or IP addresses of the cluster nodes as arguments. The optional `--start` parameter starts the cluster on all supplied cluster nodes.

```
[root@node ~]# pcs cluster setup mycluster --start \  
> node1.example.com \  
> node2.example.com \  
> node3.example.com
```

By default, a cluster node that gets rebooted does not automatically rejoin the cluster. You can use the `pcs cluster enable` command to enable automatic start of the cluster service. The `--all` option enables 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@node ~]# pcs cluster enable --all
```

Red Hat recommends that you verify that the cluster is working as expected. The `pcs cluster status` command provides an overview of the current cluster status.

```
[root@node ~]# pcs cluster status
Cluster Status:
  Cluster Summary:
    * Stack: corosync
    * Current DC: node2.example.com (version 2.0.4-6.el8-2decea3ae) - partition
      with quorum
    * Last updated: Fri Mar  5 12:23:08 2021
    * Last change: Fri Mar  5 12:22:57 2021 by root via cibadmin on
      node1.example.com
    * 3 nodes configured
    * 0 resource instances configured
  Node List:
    * Online: [ node1.example.com node2.example.com node3.example.com ]

PCSD Status:
  node1.example.com: Online
  node3.example.com: Online
  node2.example.com: Online
```

## Configuring Cluster Node Fencing

Fencing is a requirement for any high availability cluster. It prevents data corruption from an errant node, and also isolates and restarts a cluster member if the node fails to join the cluster and the remaining cluster members still form a quorum. Depending on the hardware used, the cluster 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. Different hardware devices are capable of fencing cluster nodes, such as:

- Uninterruptible power supplies (UPS)
- Power distribution units (PDU)
- Blade power control devices
- Lights-out devices

The fence devices need to be added to the cluster. For virtual machine fencing, each cluster node requires its own fence device. This is 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 fencing agent `fence_ipmilan`, the parameters `pcmk_host_list`, `username`, `password`, and `ip` are required. The `pcmk_host_list` parameter lists the corresponding host as it is known by the cluster. The `ip` parameter expects the IP address or host name of the fencing device.

```
[root@node ~]# pcs stonith create fence_device_name fence_ipmilan \
> pcmk_host_list=node_private_fqdn \
> ip=node_IP_BMC \
> username=username \
> password=password
```

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

```
[root@node ~]# pcs stonith status
* fence_nodea (stonith:fence_ipmilan): Started node1.example.com
* fence_nodeb (stonith:fence_ipmilan): Started node2.example.com
* fence_nodec (stonith:fence_ipmilan): Started node3.example.com
```

If the status of one or more fence devices is Stopped then 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 config fence_device`. You can update the settings with the `pcs stonith update` command.



## References

`pcs(8)` man page

For more information, refer to *Chapter 2. Getting started with Pacemaker* in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_getting-started-with-pacemaker-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_getting-started-with-pacemaker-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Configuring a Basic Cluster

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

## Outcomes

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

## Before You Begin

As the student user on the workstation machine, use the `lab start cluster-create` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start cluster-create
```

## Instructions

- 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. Connect to nodea and become the root user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. On workstation, in a separate terminal, connect to nodeb and become the root user.

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

- 1.3. In a third terminal, connect to nodec and become the root user.

```
[student@workstation ~]$ ssh nodec  
...output omitted...  
[student@nodec ~]$ sudo -i  
[sudo] password for student: student  
[root@nodec ~]#
```

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

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

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

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

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

```
[root@nodea ~]# yum install pcs fence-agents-ipmilan
```

```
[root@nodeb ~]# yum install pcs fence-agents-ipmilan
```

```
[root@nodec ~]# yum install pcs fence-agents-ipmilan
```

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

```
[root@nodea ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

```
[root@nodeb ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

```
[root@nodec ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

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

```
[root@nodea ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

```
[root@nodeb ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

```
[root@nodec ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

- 1.8. From the nodea machine, authenticate the cluster nodes nodea.private.example.com, nodeb.private.example.com, and nodec.private.example.com. You only have to run that command once, from the nodea machine.

```
[root@nodea ~]# pcs host 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 cluster1, using your nodea, nodeb, and nodec machines, using their host names on the private.example.com network.
- 2.1. Create and start the cluster 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 cluster1 --start \
> nodea.private.example.com \
> nodeb.private.example.com \
> nodec.private.example.com
No addresses specified for host 'nodea.private.example.com', using
'nodea.private.example.com'
No addresses specified for host 'nodeb.private.example.com', using
'nodeb.private.example.com'
```

**Chapter 1 |** Creating High Availability Clusters

```
No addresses specified for host 'nodec.private.example.com', using  
'nodec.private.example.com'  
...output omitted...  
Cluster has been successfully set up.  
Starting cluster on hosts: '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  
nodea.private.example.com: Cluster Enabled  
nodeb.private.example.com: Cluster Enabled  
nodec.private.example.com: Cluster Enabled
```

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

```
[root@nodea ~]# pcs status  
Cluster name: cluster1  
  
WARNINGS:  
No stonith devices and stonith-enabled is not false  
  
Cluster Summary:  
* Stack: corosync  
* Current DC: nodeb.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -  
partition with quorum  
* Last updated: Tue Jan 5 16:28:06 2021  
* Last change: Tue Jan 5 16:26:46 2021 by hacluster via crmd on  
nodeb.private.example.com  
* 3 nodes configured  
* 0 resource instances configured  
  
Node List:  
* Online: [ nodea.private.example.com nodeb.private.example.com  
nodec.private.example.com ]  
  
Full List of Resources:  
* No resources  
  
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 cluster. Use the fence\_ipmi\_lan fencing agent, which sends IPMI over LAN commands to perform fencing.

- 3.1. Add the fence devices for the virtual machines nodea, nodeb, and nodec to the cluster on nodea.private.example.com. For your convenience, you can copy and paste the following commands from the /root/cluster-create/resource.txt file.

```
[root@nodea ~]# pcs stonith create fence_nodea fence_ipmilan \
> pcmk_host_list=nodea.private.example.com \
> ip=192.168.0.101 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_nodeb fence_ipmilan \
> pcmk_host_list=nodeb.private.example.com \
> ip=192.168.0.102 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_nodec fence_ipmilan \
> pcmk_host_list=nodec.private.example.com \
> ip=192.168.0.103 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
```

- 3.2. Verify that the fence devices have been added correctly to the cluster.

```
[root@nodea ~]# pcs stonith status
* fence_nodea  (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb  (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodec  (stonith:fence_ipmilan): Started nodec.private.example.com
```

- 3.3. From your nodea machine, use the `pcs stonith fence` command to fence your nodeb machine. The command may take up to three minutes to complete. You can view the console of your nodeb machine to verify that it has been fenced and is rebooting.

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

## Finish

On the workstation machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish cluster-create
```

This concludes the section.

## ► Lab

# Creating High Availability Clusters

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

## Outcomes

You should be able to:

- Install the high availability software.
- Configure Pacemaker and add member nodes for a new cluster.
- Configure the cluster with the fence devices for each of its member nodes.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start cluster-review
```

## Instructions

Create a new cluster with the following requirements:

- The name of the new cluster is `cluster1`.
- 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_ipmi_lan` fencing agent, which sends IPMI over LAN commands to perform fencing. The following table provides the connection details to access those BMC devices.

### BMC Device Parameters

Machine	BMC IP address	Fence name	Login	Password
nodea	192.168.0.101	fence_nodea	admin	password
nodeb	192.168.0.102	fence_nodeb	admin	password
nodec	192.168.0.103	fence_nodec	admin	password

On your machines, the password for the student user is `student`, and the root password is `redhat`.

1. Prepare the nodea, nodeb, and nodec machines to act as cluster nodes. Allow the cluster software to communicate through the firewall. Install, enable, and start the required software. The password of the hacluster user should be redhat.
2. Configure and start a three-node cluster named cluster1, consisting of nodea, nodeb, and nodec. Use their fully qualified domain names in the private.example.com subdomain. Remember to authenticate the nodes for pcs. Cluster must be enabled so that the nodes join the cluster on boot.
3. Configure fencing for the virtual machines nodea, nodeb, and nodec that act as cluster nodes in the cluster. Use the fence\_ipmi\_lan fencing agent, which sends IPMI-over-LAN commands to perform fencing. Because in the classroom environment the BMC device may take a long time to reply, add the power\_timeout=180 option when you create the fence resource. The lanplus option should be set to 1. See the BMC Device Parameters table in the Instructions section for more information. To test your configuration, fence nodeb machine and check that it rebooted. You might have to wait a few minutes for the nodeb machine to rejoin the cluster.

## Evaluation

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade cluster-review
```

## Finish

As the student user on the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish cluster-review
```

This concludes the section.

## ► Solution

# Creating High Availability Clusters

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

### Outcomes

You should be able to:

- Install the high availability software.
- Configure Pacemaker and add member nodes for a new cluster.
- Configure the cluster with the fence devices for each of its member nodes.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start cluster-review
```

### Instructions

Create a new cluster with the following requirements:

- The name of the new cluster is `cluster1`.
- 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_ipmi_lan` fencing agent, which sends IPMI over LAN commands to perform fencing. The following table provides the connection details to access those BMC devices.

#### BMC Device Parameters

Machine	BMC IP address	Fence name	Login	Password
nodea	192.168.0.101	fence_nodea	admin	password
nodeb	192.168.0.102	fence_nodeb	admin	password
nodec	192.168.0.103	fence_nodec	admin	password

On your machines, the password for the student user is `student`, and the root password is `redhat`.

1. Prepare the nodea, nodeb, and nodec machines to act as cluster nodes. Allow the cluster software to communicate through the firewall. Install, enable, and start the required software. The password of the hacluster user should be redhat.

- 1.1. Connect to nodea, nodeb, and nodec in three different terminals and become the root user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

```
[student@workstation ~]$ ssh nodec  
...output omitted...  
[student@nodec ~]$ sudo -i  
[sudo] password for student: student  
[root@nodec ~]#
```

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

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

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

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

- 1.3. Install the pcs and fence-agents-ipmilan RPM packages on nodea, nodeb, and nodec.

```
[root@nodea ~]# yum install pcs fence-agents-ipmilan
```

```
[root@nodeb ~]# yum install pcs fence-agents-ipmilan
```

```
[root@nodec ~]# yum install pcs fence-agents-ipmilan
```

- 1.4. Enable and start the pcsd service on your nodea, nodeb, and nodec systems.

```
[root@nodea ~]# systemctl enable --now pcsd
```

```
[root@nodeb ~]# systemctl enable --now pcsd
```

```
[root@nodec ~]# systemctl enable --now pcsd
```

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

```
[root@nodea ~]# echo redhat | passwd --stdin hacluster  
Changing password for user hacluster.  
passwd: all authentication tokens updated successfully.
```

```
[root@nodeb ~]# echo redhat | passwd --stdin hacluster  
Changing password for user hacluster.  
passwd: all authentication tokens updated successfully.
```

```
[root@nodec ~]# echo redhat | passwd --stdin hacluster  
Changing password for user hacluster.  
passwd: all authentication tokens updated successfully.
```

2. Configure and start a three-node cluster named **cluster1**, consisting of nodea, nodeb, and nodec. Use their fully qualified domain names in the **private.example.com** subdomain. Remember to authenticate the nodes for pcs. Cluster must be enabled so that the nodes join the cluster on boot.

- 2.1. On nodea, as **root** user, authenticate the cluster nodes **nodea.private.example.com**, **nodeb.private.example.com**, and **nodec.private.example.com**.

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

- 2.2. Create and start the cluster **cluster1**, consisting of 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 cluster1 --start \
> nodea.private.example.com \
> nodeb.private.example.com \
> nodec.private.example.com
No addresses specified for host 'nodea.private.example.com', using
'nodea.private.example.com'
No addresses specified for host 'nodeb.private.example.com', using
'nodeb.private.example.com'
No addresses specified for host 'nodec.private.example.com', using
'nodec.private.example.com'
...output omitted...
Cluster has been successfully set up.
Starting cluster on hosts: 'nodea.private.example.com',
'nodeb.private.example.com', 'nodec.private.example.com'...
```

### 2.3. Enable automatic startup of the cluster on all configured cluster nodes.

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

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

```
[root@nodea ~]# pcs status
Cluster name: cluster1

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync
  * Current DC: nodea.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
  * Last updated: Mon Feb 22 08:40:15 2021
  * Last change: Mon Feb 22 08:39:59 2021 by hacluster via crmd on
nodea.private.example.com
  * 3 nodes configured
  * 0 resource instances configured

Node List:
  * Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

Full List of Resources:
  * No resources

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

3. Configure fencing for the virtual machines nodea, nodeb, and nodec that act as cluster nodes in the cluster. Use the `fence_ipmilan` fencing agent, which sends IPMI-over-LAN commands to perform fencing. Because in the classroom environment the BMC device may take a long time to reply, add the `power_timeout=180` option when you create the fence resource. The `lanplus` option should be set to 1. See the BMC Device Parameters table in the Instructions section for more information. To test your configuration, fence nodeb machine and check that it is rebooted. You might have to wait a few minutes for the nodeb machine to rejoin the cluster.
- 3.1. Add the fence devices for the nodea, nodeb, and nodec machines to the cluster on `nodea.private.example.com`.

```
[root@nodea ~]# pcs stonith create fence_nodea fence_ipmilan \
> pcmk_host_list=nodea.private.example.com \
> ip=192.168.0.101 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_nodeb fence_ipmilan \
> pcmk_host_list=nodeb.private.example.com \
> ip=192.168.0.102 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_nodec fence_ipmilan \
> pcmk_host_list=nodec.private.example.com \
> ip=192.168.0.103 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
```

- 3.2. Verify that the fence devices have been added correctly to the cluster.

```
[root@nodea ~]# pcs stonith status
* fence_nodea  (stonith:fence_ipmilan):  Started nodea.private.example.com
* fence_nodeb  (stonith:fence_ipmilan):  Started nodeb.private.example.com
* fence_nodec  (stonith:fence_ipmilan):  Started nodec.private.example.com
```

- 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 it has been fenced and is rebooting.

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

## Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade cluster-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish cluster-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- The difference between the types of high-availability clusters.
- The required hardware and software for running a Red Hat high availability 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 node membership in the cluster and describe how it impacts cluster operation.

### Objectives

- Add and remove nodes from an existing high availability cluster.
- Explain how quorum operates and how it protects a high availability cluster from errors.
- Adjust the way quorum is calculated and describe when this might be necessary.

### Sections

- Managing Cluster Membership (and Guided Exercise)
- Describing and Observing Quorum Operation (and Guided Exercise)
- Managing Quorum Calculations (and Guided Exercise)

### Lab

- Managing Cluster Nodes and Quorum

# Managing Cluster Membership

## Objectives

After completing this section, you should be able to add and remove nodes from an existing high availability cluster.

## 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 controls whether a cluster node is participating in the cluster at runtime.
- Enabling and disabling the cluster services controls 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` must 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@node ~]# pcs cluster start
```

To stop cluster services on the remote node, `node1.example.com`, execute:

```
[root@node ~]# pcs cluster stop node1.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@node ~]# 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@node ~]# pcs cluster enable
```

To disable cluster services on the remote node, `node2.example.com`, execute:

```
[root@node ~]# pcs cluster disable node2.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@node ~]# 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. The classroom environment includes a Baseboard Management Controller (BMC) that you can use to power off, power on, or restart the machine by using IPMI over LAN commands, the use of the `fence-agents-ipmilan` package is needed.
  - 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 host auth node.fqdn` from an existing cluster node.
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. This step should be

done on a node that is a current node withing the cluster, and that node should be in good standing.

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

```
[root@node1 ~]# pcs cluster node add node4.example.com
No addresses specified for host 'node4.example.com', using 'node4.example.com'
Sending 'corosync authkey', 'pacemaker authkey' to 'node4.example.com'
node4.example.com: successful distribution of the file 'corosync authkey'
node4.example.com: successful distribution of the file 'pacemaker authkey'
Sending updated corosync.conf to nodes...
node1.example.com: Succeeded
node4.example.com: Succeeded
node2.example.com: Succeeded
node3.example.com: Succeeded
node1.example.com: Corosync configuration reloaded
```

After the node is successfully added and authorized, a system administrator can 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 node4.example.com` and/or `pcs cluster enable node4.example.com` are used, the new node does 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. This command must be run from an existing node in good standing withing the cluster, and not run from the node to be removed.

```
[root@node1 ~]# pcs cluster node remove node4.example.com
```

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@node1 ~]# 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.

## Chapter 2 | Managing Cluster Nodes and Quorum

The `pcs node standby` command can put the local node in `standby` mode. The `pcs node 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, `node1.example.com`, into `standby` mode, run:

```
[root@node ~]# pcs node standby node1.example.com
```

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 can 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@node ~]# pcs node 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
<code>pcs status cluster</code>	Show only information related to cluster status.
<code>pcs status resources</code>	Show only the status of the resource groups and their individual resources in the cluster.
<code>pcs status nodes</code>	Show only the status of the configured cluster nodes.
<code>pcs status corosync</code>	Show only the status of <code>corosync</code> .
<code>pcs status pcsd</code>	Show only the status of <code>pcsd</code> 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 displays all information related to the cluster and cluster nodes:

```
[root@node1 ~]# pcs status
Cluster name: cluster1
Cluster Summary:
  * Stack: corosync
  * Current DC: node3.example.com (version 2.0.4-6.el8-2deceaa3ae) - partition
    with quorum
  * Last updated: Tue Mar  9 20:47:51 2021
```

**Chapter 2 |** Managing Cluster Nodes and Quorum

```
* Last change: Tue Mar 9 20:46:30 2021 by root via cibadmin on
node4.example.com
* 4 nodes configured
* 7 resource instances configured

Node List:
* Node node3.example.com: standby ①
* Online: [ node1.example.com node2.example.com ] ②
* OFFLINE: [ node4.example.com ] ③

Full List of Resources:
* fence_node1 (stonith:fence_ipmilan): Started node1.example.com
* fence_node2 (stonith:fence_ipmilan): Started node2.example.com
* fence_node3 (stonith:fence_ipmilan): Started node1.example.com
* fence_node4 (stonith:fence_ipmilan): Started node1.example.com
* Resource Group: web:
  * firstwebip (ocf::heartbeat:IPAddr2): Started node2.example.com
  * firstwebfs (ocf::heartbeat:Filesystem): Started node2.example.com
  * firstwebserver (ocf::heartbeat:apache): Started node2.example.com

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:

- ① The cluster node `node3.example.com` is in **standby** mode.
- ② The cluster nodes `node1.example.com` and `node2.example.com` are fully operational and participating in the cluster, and therefore marked as **Online**.
- ③ The cluster node `node4.private.example.com` 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

For more information, refer to the *Managing cluster nodes* chapter in the *Configuring and managing high availability clusters* at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_clusternode-management-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_clusternode-management-configuring-and-managing-high-availability-clusters)

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

## ► Guided Exercise

# Managing Cluster Membership

In this exercise, you will add a fourth node to an existing three-node cluster.

## Outcomes

You should be able to extend an existing cluster.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start quorum-extend
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines.

## Instructions

- 1. Prepare the machine noded to become a cluster member by configuring the firewall to allow communication, installing and starting the required software, setting the hacluster user password to match the password on the other cluster members, and authenticating the node to the cluster.

- 1.1. Connect to noded and become the `root` user.

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

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

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

- 1.3. Install the `pcs` and `fence-agents-ipmilan` packages on `noded.private.example.com`.

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

- 1.4. Enable and start the `pcsd` service on the new node.

```
[root@noded ~]# systemctl enable --now pcsd
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/
lib/systemd/system/pcsd.service.
```

- 1.5. To communicate among the nodes, change the hacluster system user's password to redhat on noded to match other nodes.

```
[root@noded ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

- 1.6. In a separate terminal, connect to nodea, become the root user, and then authenticate the cluster node noded.private.example.com on the existing cluster.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]# pcs host auth -u hacluster -p redhat noded.private.example.com
noded.private.example.com: Authorized
```

- ▶ 2. From the nodea machine, extend the existing cluster by adding noded.private.example.com to the cluster.

```
[root@nodea ~]# pcs cluster node add noded.private.example.com
No addresses specified for host 'noded.private.example.com', using
'noded.private.example.com'
Disabling sbd...
noded.private.example.com: sbd disabled
Sending 'corosync authkey', 'pacemaker authkey' to 'noded.private.example.com'
noded.private.example.com: successful distribution of the file 'corosync authkey'
noded.private.example.com: successful distribution of the file 'pacemaker authkey'
Sending updated corosync.conf to nodes...
nodea.private.example.com: Succeeded
noded.private.example.com: Succeeded
nodeb.private.example.com: Succeeded
nodec.private.example.com: Succeeded
nodea.private.example.com: Corosync configuration reloaded
```

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

- 3.1. Return to the terminal connected to the noded machine and then enable automatic startup of the cluster services on noded when the machine is rebooted.

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

- 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@noded ~]# pcs status  
Cluster name: cluster1  
Cluster Summary:  
  * Stack: corosync  
  * Current DC: nodec.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -  
    partition with quorum  
    * Last updated: Thu Jan  7 14:54:20 2021  
    * Last change: Thu Jan  7 14:54:14 2021 by hacluster via crmd on  
      nodec.private.example.com  
    * 4 nodes configured  
    * 3 resource instances configured  
  
Node List:  
  * 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_ipmilan):  Started nodea.private.example.com  
  * fence_nodeb  (stonith:fence_ipmilan):  Started nodeb.private.example.com  
  * fence_nodec  (stonith:fence_ipmilan):  Started nodec.private.example.com  
  
Daemon Status:  
  corosync: active/enabled  
  pacemaker: active/enabled  
  pcsd: active/enabled
```

- 4. Configure fencing for noded. Use the **fence\_ipmilan** fencing agent. The noded BMC device IP address is **192.168.0.104**, the login is **admin**, and the password is **password**. Because in the classroom environment the BMC device might take a long time to reply, increase the timeout to 180 seconds. This new fencing resource should be called **fence\_noded**.
- 4.1. Add the fence device for noded. You can run the following command from any cluster node. For your convenience, you can copy and paste the following command from the **/root/quorum-extend/resource.txt** file.

```
[root@noded ~]# pcs stonith create fence_noded fence_ipmilan \
> pcmk_host_list=noded.private.example.com \
> ip=192.168.0.104 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@noded ~]#
```

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

```
[root@noded ~]# pcs stonith status
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
* fence_noded (stonith:fence_ipmilan): Started noded.private.example.com
```

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

5.1. Restart noded.

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

5.2. Return to the terminal connected to the nodea machine, and then verify that noded has automatically joined the cluster after it rebooted.

```
[root@nodea ~]# pcs status
Cluster name: cluster1
Cluster Summary:
  * Stack: corosync
  * Current DC: nodec.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
  * Last updated: Thu Jan  7 15:04:43 2021
  * Last change: Thu Jan  7 15:02:11 2021 by root via cibadmin on
noded.private.example.com
  * 4 nodes configured
  * 4 resource instances configured

Node List:
  * 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_ipmilan): Started nodea.private.example.com
  * fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
  * fence_noded (stonith:fence_ipmilan): Started noded.private.example.com

Daemon Status:
```

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

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish quorum-extend
```

This concludes the section.

# Describing and Observing Quorum Operation

## Objectives

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

## What Is Quorum?

For a cluster to work as expected, the nodes must agree on certain facts, such as which machines are currently cluster members, where services are running, and which machines are using which resources.

The High Availability Add-On implements the method by the use of a majority voting scheme. Every cluster node casts one vote if it successfully joins the corosync network communication and can 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, then 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 1, 2, 3, 4, and 5 with a service that uses an ext4 file system on shared storage. This service is currently running on node 1.

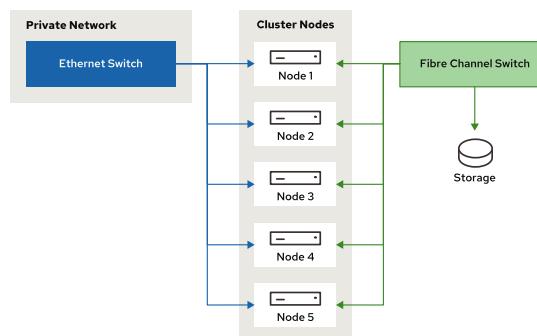


Figure 2.1: A Five-node Cluster

The nodes 4 and 5 get split off from the main private network and cannot communicate with the nodes 1, 2, and 3. Without quorum, those two nodes will decide that 1, 2, and 3 have all failed.

and must 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 4 and 5 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, because if either node fails, then the other node does not consist of a majority of the nodes in the cluster.

Assuming that all nodes in this example have one vote, this situation cannot arise, because node 4 and node 5 only have two votes together, which is not more than half of the total votes (five). This would result in node 4 and node 5 ceasing to function until at least one other vote was added. Nodes 1, 2, and 3, on the other hand, remain active providing service, because they have three votes combined, which is more than half of the total votes.

## Calculating Quorum

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

### Expected votes

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

### 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 High Availability Add-On provides a comprehensive utility to display the current state of quorum in a cluster: `pcs quorum status`. The `pcs quorum status` 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@node ~]# pcs quorum status
Quorum information
-----
Date: Tue Mar 16 15:30:56 2021
Quorum provider: corosync_votequorum
Nodes: 4          ①
Node ID: 1        ②
Ring ID: 1.2a
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   Qdevice Name
    1       1       NR node1.private.example.com (local)
    2       1       NR node2.private.example.com
    3       1       NR node3.private.example.com
    4       1       NR node4.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 `pcs quorum status` 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 value of expected votes that `corosync` could see at the last transition and is always expected to be the same as `Expected votes`.
- ⑥ The `Total votes` entry shows the count of votes currently present in the cluster.

- ⑦ The **Quorum** entry shows how many votes must 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

`corosync-quorumtool(8)` man page

For more information, refer to the *Cluster quorum* chapter in the *Configuring and managing high availability clusters* at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-cluster-quorum-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-cluster-quorum-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Describing and Observing Quorum Operation

In this exercise, you will observe how a cluster's quorum status and calculations change as cluster members leave the cluster for various reasons and then rejoin it.

## Outcomes

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

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start quorum-operation
```

This command deploys a four-node cluster on the nodea, nodeb, nodec, and noded machines.

## Instructions

- ▶ 1. On your nodea machine, monitor the output of the `pcs quorum status` command by using the `watch` command. Leave it running.
  - 1.1. Connect to nodea and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Run the `pcs quorum status` command by using the `watch` command.

```
[root@nodea ~]# watch pcs quorum status  
Every 2.0s: pcs quorum status    nodea.lab.example.com: Wed Jan  6 19:51:22 2021
```

```
Quorum information  
-----  
Date:           Wed Jan  6 19:51:22 2021  
Quorum provider: corosync_votequorum  
Nodes:        4  
Node ID:       1  
Ring ID:       1.d  
Quorate:     Yes
```

```
Votequorum information
```

```

-----
Expected votes: 4
Highest expected: 4
Total votes: 4
Quorum: 3
Flags: Quorate

Membership information
-----

```

Nodeid	Votes	Qdevice Name
1	1	NR nodea.private.example.com (local)
2	1	NR nodeb.private.example.com
3	1	NR nodec.private.example.com
4	1	NR noded.private.example.com

The preceding output shows the four active nodes, with a corresponding number of expected and total votes. The cluster is in the quorate state.

- ▶ 2. On your nodec machine, disable cluster communications by temporarily closing the ports that the corosync process uses. Observe the results.

- 2.1. In a separate terminal, connect to the nodec machine and become the root user.

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo -i
[sudo] password for student: student
[root@nodec ~]#
```

- 2.2. Add temporary firewalld rules to block the corosync traffic both inbound and outbound. To do so, run the /root/quorum-ops/firewall-block.sh script that the exercise preparation command has deployed for you.

```
[root@nodec ~]# /root/quorum-ops/firewall-block.sh
success
success
```

Because the other nodes cannot reach the nodec machine anymore, the cluster fences it. As a consequence, you should lose the SSH connection to the nodec machine.

- 2.3. Return to the terminal where the pcs quorum status command is running and then review the output.

```

Quorum information
-----
Date: Wed Jan 6 20:08:56 2021
Quorum provider: corosync_votequorum
Nodes: 3
Node ID: 1
Ring ID: 1.11
Quorate: Yes

Votequorum information

```

```
-----
Expected votes: 4
Highest expected: 4
Total votes: 3
Quorum: 3
Flags: Quorate

Membership information
-----

```

Nodeid	Votes	Qdevice Name
1	1	NR nodea.private.example.com (local)
2	1	NR nodeb.private.example.com
4	1	NR noded.private.example.com

Notice that the number of expected votes does not change, but the number of total votes has dropped to 3. Additionally, the command does not list the `nodec.private.example.com` machine anymore in the **Membership information** table.

- 2.4. Wait for your `noded` machine to become available. The cluster should show all nodes active again.
- ▶ 3. Test your cluster further by powering down two nodes, `nodeb` and `noded`. Observe the results.
  - 3.1. Power down both your `nodeb` and `noded` machines.

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

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



### Note

The cluster does not try to fence the nodes you just powered off, because you shut them down gracefully. The cluster only fences unexpectedly unresponsive nodes.

- 3.2. Return to the terminal where the `pcs quorum status` command is running. Notice the error message in the output.

```
Error: Unable to get quorum status:
```

The command does not work anymore because the cluster has lost quorum. However, you can still use the `corosync-quorумtool` command to display the cluster's quorum status. To do so, press `Ctrl+C` to exit from the `watch` command and then run the `corosync-quorūmtool` command.

```
[root@nodea ~]# corosync-quorūmtool
Quorum information
-----
Date: Wed Jan 6 20:31:06 2021
Quorum provider: corosync_votequorum
Nodes: 2
Node ID: 1
```

**Chapter 2 |** Managing Cluster Nodes and Quorum

```
Ring ID:          1.22
Quorate:        No

Votequorum information
-----
Expected votes: 4
Highest expected: 4
Total votes:    2
Quorum:        3 Activity blocked
Flags:

Membership information
-----
  Nodeid   Votes   Qdevice Name
    1       1       NR nodea.private.example.com (local)
    3       1       NR nodec.private.example.com
```

Notice that the Quorum line now reads **Activity blocked**. The cluster blocks the activity because the four-node cluster needs three votes present to have quorum, and it only has two, one from each of the remaining nodes. To prevent resource corruption, the cluster has quit starting, stopping, or otherwise touching cluster resources.

- 4. Start your nodeb machine and then observe the results.
- 4.1. Use the appropriate method for your classroom environment to start your nodeb machine.
  - 4.2. Run the `pcs quorum status` command and then notice that the cluster returns to a quorate state, but with one missing node. You might have to wait a few minutes for the nodeb machine to rejoin the cluster.

```
[root@nodea ~]# pcs quorum status
Quorum information
-----
Date:          Wed Jan  6 20:51:08 2021
Quorum provider: corosync_votequorum
Nodes:        3
Node ID:      1
Ring ID:      1.27
Quorate:    Yes

Votequorum information
-----
Expected votes: 4
Highest expected: 4
Total votes:    3
Quorum:        3
Flags:         Quorate

Membership information
-----
  Nodeid   Votes   Qdevice Name
```

1	1	NR nodea.private.example.com (local)
2	1	NR nodeb.private.example.com
4	1	NR noded.private.example.com

- 5. Start your nodec machine.

## Finish

On the workstation machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish quorum-operation
```

This concludes the section.

# Managing Quorum Calculations

## Objectives

After completing this section, you should be able to adjust the way quorum is calculated and describe when this might be necessary.

## Set Quorum Calculation Options

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

### Cluster Setup Options

Quorum Option	Description
<code>wait_for_all</code>	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.
<code>auto_tie_breaker</code>	Quorum can be achieved with only 50% of the cluster nodes up, instead of the default 50% + 1 votes. In case of a 50% versus 50% split-brain situation, the side where the lowest node id takes part wins quorum.
<code>last_man_standing</code>	<p>When you set <code>last_man_standing=1</code>, 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.</p> <p>The tunable <code>last_man_standing_window</code> 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).</p> <p>In conjunction with the <code>auto_tie_breaker</code> option, <code>last_man_standing</code> allows for downgrading the cluster to one node.</p> <p>The use of <code>last_man_standing</code> also requires the use of <code>wait_for_all</code> to prevent multiple partitions (subsets of the cluster) to claim quorum at the same time.</p>

## Chapter 2 | Managing Cluster Nodes and Quorum

The quorum options 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=1` and `wait_for_all=1` enabled, execute:

```
[root@node1 ~]# pcs cluster setup mycluster \
> node1.example.com node2.example.com node3.example.com \
> quorum last_man_standing=1 wait_for_all=1 \
> --start
```

## 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.

Changing the general quorum options requires that the cluster be stopped, you can stop your cluster with `pcs cluster stop --all`, and modify the quorum options with the `pcs quorum update` command.

The following example enables the `last_man_standing`, `wait_for_all`, and `auto_tie_breaker` options:

```
[root@node1 ~]# pcs quorum update \
> auto_tie_breaker=1 \
> last_man_standing=1 \
> wait_for_all=1
Checking corosync is not running on nodes...
node3.example.com: corosync is not running
node4.example.com: corosync is not running
node2.example.com: corosync is not running
node1.example.com: corosync is not running
Sending updated corosync.conf to nodes...
node1.example.com: Succeeded
node2.example.com: Succeeded
node3.example.com: Succeeded
node4.example.com: Succeeded
```

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.

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



### Note

The content of the `corosync.conf` file must be the same on every node in the cluster. The `pcs quorum update` command updates the file locally and then automatically synchronizes it with all the nodes, even when the cluster is stopped. If you edit the file manually, which Red Hat discourages, then use the `pcs cluster sync` command to replicate the file on all the nodes. 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 `pcs quorum status` command shows the new options listed in the **Flags** output:

```
[root@node1 ~]# pcs quorum status
Quorum information
-----
Date:           Thu Mar 18 19:02:38 2021
Quorum provider: corosync_votequorum
Nodes:          4
Node ID:        1
Ring ID:        1.70
Quorate:       Yes

Votequorum information
-----
Expected votes: 4
Highest expected: 4
Total votes:    4
Quorum:         3
Flags:          Quorate WaitForAll LastManStanding AutoTieBreaker

Membership information
-----
Nodeid   Votes   Qdevice Name
      1       1     NR node1.example.com (local)
      2       1     NR node2.example.com
      3       1     NR node3.example.com
      4       1     NR node4.example.com
```



### References

`votequorum(5)` man page

For more information, refer to the *Cluster quorum* chapter in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-cluster-quorum-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-cluster-quorum-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Managing Quorum Calculations

In this exercise, you will modify the configuration of a four-node cluster so that it maintains quorum with a smaller number of active nodes if some missing nodes are gracefully shut down.

## Outcomes

You should be able to:

- Modify a cluster quorum configuration.
- Enable the `last_man_standing` and `wait_for_all` options so that an administrator can gracefully disconnect nodes from the cluster, one by one, and keep a functional cluster at the same time.

## Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start quorum-modify
```

This command deploys a four-node cluster on the `nodea`, `nodeb`, `nodec`, and `noded` machines.

## Instructions

- 1. Reconfigure your cluster to enable the `last_man_standing` and `wait_for_all` options and then verify that the new configuration is active.

The `last_man_standing` option recalculates the quorum and its expected votes every ten seconds under certain circumstances. For the cluster to work correctly, whenever you enable the `last_man_standing` option you must also enable the `wait_for_all` option, which allows the cluster to gain quorum for the first time only after all nodes have been up simultaneously at least once.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. Check the cluster configuration options.

```
[root@nodea ~]# pcs quorum config
Options:
```

**Chapter 2 |** Managing Cluster Nodes and Quorum

Notice that no standard options have been modified.

- 1.3. Stop the cluster services on all nodes because the cluster only allows modification of the quorum configuration when the services are stopped.

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

- 1.4. Enable the `last_man_standing` and `wait_for_all` options.

```
[root@nodea ~]# pcs quorum update last_man_standing=1 wait_for_all=1
Checking corosync is not running on nodes...
nodeb.private.example.com: corosync is not running
noded.private.example.com: corosync is not running
nodec.private.example.com: corosync is not running
nodea.private.example.com: corosync is not running
Sending updated corosync.conf to nodes...
nodeb.private.example.com: Succeeded
nodea.private.example.com: Succeeded
nodec.private.example.com: Succeeded
noded.private.example.com: Succeeded
```

- 1.5. Start the cluster services on all cluster nodes.

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

- 1.6. Verify the new configuration.

```
[root@nodea ~]# pcs quorum config
Options:
  last_man_standing: 1
  wait_for_all: 1
```

Notice that the `last_man_standing` and `wait_for_all` options are set to 1 (enabled).

- 2. Monitor quorum and then power down the `nodec` machine to degrade the cluster.

- 2.1. Monitor quorum by using the `watch pcs quorum status` command.

```
[root@nodea ~]# watch pcs quorum status
Every 2.0s: pcs quorum status    nodea.lab.example.com: Thu Jan  7 03:54:25 2021

Quorum information
-----
Date:           Thu Jan  7 03:54:25 2021
Quorum provider: corosync_votequorum
Nodes:          4
Node ID:        1
Ring ID:        1.16
Quorate:       Yes

Votequorum information
-----
Expected votes: 4
Highest expected: 4
Total votes:     4
Quorum:         3
Flags:          Quorate WaitForAll LastManStanding
...output omitted...
```

Leave the command running.

- 2.2. In a separate terminal, connect to the `nodec` machine and then power it down.

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo poweroff
[sudo] password for student: student
```

- 2.3. Return to the terminal where the `pcs quorum status` command is running. Ten seconds after you stop the `nodec` machine, the cluster reduces the `Expected votes` value from 4 to 3. Notice that the cluster recalculates the quorum value, which is now set to 2.

The cluster behaves that way because the node is gracefully leaving the cluster. You can control how long the cluster waits before recalculating the quorum with the `last_man_standing_window` option, which is 10000 milliseconds (10 seconds) by default.

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

Keep the `watch` command running.

- 3. Power down an additional machine, `noded`, and then confirm that the cluster is still in a quorate state.

- 3.1. Connect to the noded machine and then power it down.

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo poweroff
[sudo] password for student: student
```

- 3.2. Return to the terminal where the `pcs quorum status` command is running. Ten seconds after you stop the noded machine, the cluster reduces the `Expected votes` value from 3 to 2.

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

Without the `last_man_standing` option enabled, the cluster would have become inquorate when the second node went down. Remember that by default, a four-node cluster needs to have three nodes operating to have quorum. Normally, you need at least two nodes operating for the cluster to stay in a quorate state, so the `Quorum` value is not reduced to 1. By default, the cluster does not allow running with a single node and therefore never sets the `Quorum` value to 1. There is a configuration that allows you to create a two-node cluster that will continue to operate with only one remaining node. This has some special considerations and is discussed elsewhere in this course.

Press `Ctrl+C` to exit from the `watch` command.

- 4. Use the appropriate method for your classroom environment to restart the `nodec` and `noded` machines.

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish quorum-modify
```

This concludes the section.

## ▶ Lab

# Managing Cluster Nodes and Quorum

In this lab, you will extend a three-node cluster to four nodes, and make changes to the quorum calculations for that cluster.

## Outcomes

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

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start quorum-review
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines.

## Instructions

1. Prepare your `noded` machine to become a cluster node. To do so, allow the cluster software to communicate through the firewall, install the cluster packages, set the password of the `hacluster` user to `redhat`, and then enable and start the required service. In addition, authenticate your `noded.private.example.com` machine for `pcs` to the cluster already running on the `nodea`, `nodeb`, and `nodec` machines.  
On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.
2. Add the `noded.private.example.com` machine as a new node to the existing cluster running on the `nodea`, `nodeb`, and `nodec` machines. Make sure that when the new node starts, it joins the cluster automatically. Configure fencing for the `noded` machine. To do so, install and use the `fence_ipmilan` fencing agent. The `noded` BMC device IP address is `192.168.0.104`, the login is `admin`, and the password is `password`. The name of the fence resource must be `fence_noded`. Because in the classroom environment the BMC device might take a long time to reply, add the `power_timeout=180` option when you create the fence resource. The `Lanplus` option should be set to 1.
3. Reconfigure cluster quorum options to include `auto_tie_breaker`.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade quorum-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish quorum-review
```

This concludes the section.

## ► Solution

# Managing Cluster Nodes and Quorum

In this lab, you will extend a three-node cluster to four nodes, and make changes to the quorum calculations for that cluster.

### Outcomes

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

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start quorum-review
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines.

### Instructions

1. Prepare your `noded` machine to become a cluster node. To do so, allow the cluster software to communicate through the firewall, install the cluster packages, set the password of the `hacluster` user to `redhat`, and then enable and start the required service. In addition, authenticate your `noded.private.example.com` machine for `pcs` to the cluster already running on the `nodea`, `nodeb`, and `nodec` machines.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

- 1.1. Connect to `noded` and become the `root` user.

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

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

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

- 1.3. Install the `pcs` package on `noded.private.example.com`.

```
[root@noded ~]# yum install pcs
```

- 1.4. Enable and start the pcsd service on the new node.

```
[root@noded ~]# systemctl enable --now pcsd
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/
lib/systemd/system/pcsd.service.
```

- 1.5. Change the password of the hacluster system user to redhat on noded.

```
[root@noded ~]# echo redhat | passwd --stdin hacluster
Changing password for user hacluster.
passwd: all authentication tokens updated successfully.
```

- 1.6. In a separate terminal, connect to nodea and become the root user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.7. Authenticate the cluster node noded.private.example.com on the existing cluster.  
Do this in the terminal connected to nodea.

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

2. Add the noded.private.example.com machine as a new node to the existing cluster running on the nodea, nodeb, and nodec machines. Make sure that when the new node starts, it joins the cluster automatically. Configure fencing for the noded machine. To do so, install and use the fence\_ipmilan fencing agent. The noded BMC device IP address is 192.168.0.104, the login is admin, and the password is password. The name of the fence resource must be fence\_noded. Because in the classroom environment the BMC device might take a long time to reply, add the power\_timeout=180 option when you create the fence resource. The lanplus option should be set to 1.

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

```
[root@nodea ~]# pcs cluster node add noded.private.example.com
No addresses specified for host 'noded.private.example.com', using
'noded.private.example.com'
Disabling sbd...
noded.private.example.com: sbd disabled
Sending 'corosync authkey', 'pacemaker authkey' to 'noded.private.example.com'
noded.private.example.com: successful distribution of the file 'corosync authkey'
noded.private.example.com: successful distribution of the file 'pacemaker authkey'
Sending updated corosync.conf to nodes...
nodec.private.example.com: Succeeded
noded.private.example.com: Succeeded
nodea.private.example.com: Succeeded
nodeb.private.example.com: Succeeded
nodea.private.example.com: Corosync configuration reloaded
```

**Chapter 2 |** Managing Cluster Nodes and Quorum

- 2.2. From your terminal connected to the noded machine, enable automatic startup of the cluster services on noded when the machine is rebooted.

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

- 2.3. Start the cluster services on noded.

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

- 2.4. Install the `fence-agents-ipmilan` package on `noded.private.example.com`.

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

- 2.5. On nodea, as root user, configure fencing for noded. Use the `fence_ipmilan` fencing agent. The noded BMC device IP address is `192.168.0.104`, the login is `admin`, and the password is `password`. Because in the classroom environment the BMC device might take a long time to reply, increase the timeout to 180 seconds. The `lanplus` option should be set to 1. This new fencing resource should be called `fence_noded`.

```
[root@nodea ~]# pcs stonith create fence_noded fence_ipmilan \
> pcmk_host_list=noded.private.example.com \
> ip=192.168.0.104 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
```

- 2.6. Verify that all fence devices have been correctly added to the cluster.

```
[root@nodea ~]# pcs stonith status
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
* fence_noded (stonith:fence_ipmilan): Started noded.private.example.com
```

- 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@nodea ~]# pcs status
Cluster name: cluster1
Cluster Summary:
* Stack: corosync
* Current DC: nodea.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
* Last updated: Mon Feb 22 17:56:16 2021
* Last change: Mon Feb 22 17:52:02 2021 by root via cibadmin on
nodea.private.example.com
* 4 nodes configured
* 4 resource instances configured
```

```
Node List:  
* 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_ipmilan): Started nodea.private.example.com  
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com  
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com  
* fence_noded (stonith:fence_ipmilan): Started noded.private.example.com  
  
Daemon Status:  
corosync: active/enabled  
pacemaker: active/enabled  
pcsd: active/enabled
```

### 3. Reconfigure cluster quorum options to include auto\_tie\_breaker.

#### 3.1. On nodea, as root user, stop the cluster services on all nodes.

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

#### 3.2. Add the auto\_tie\_breaker option to the cluster on nodea.

```
[root@nodea ~]# pcs quorum update auto_tie_breaker=1  
Checking corosync is not running on nodes...  
noded.private.example.com: corosync is not running  
nodeb.private.example.com: corosync is not running  
nodea.private.example.com: corosync is not running  
nodec.private.example.com: corosync is not running  
Sending updated corosync.conf to nodes...  
nodea.private.example.com: Succeeded  
nodec.private.example.com: Succeeded  
nodeb.private.example.com: Succeeded  
noded.private.example.com: Succeeded
```

#### 3.3. Start the cluster services on all cluster nodes.

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

#### 3.4. Verify the quorum configuration has been modified.

```
[root@nodea ~]# pcs quorum config  
Options:  
    auto_tie_breaker: 1
```

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade quorum-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish quorum-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- Add and remove nodes to and from the cluster.
- Review cluster quorum by means of `corosync-quorумtool`.
- Modify quorum options defined in `/etc/corosync/corosync.conf`.
- Synchronize quorum options in all the clusters by using `pcs cluster sync`.



## Chapter 3

# Isolating Malfunctioning Cluster Nodes

### Goal

Isolate unresponsive cluster nodes to protect data and recover services and resources after a failure.

### Objectives

- Describe what fencing is and how it is used to protect data during cluster node failures.
- Identify supported fencing devices and test them.
- Configure the cluster to use the correct fencing device when fencing a cluster node.

### Sections

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

### Lab

- Isolating Malfunctioning Cluster Nodes

# Protecting Data with Fencing

---

## Objectives

After completing this section, you should be able to describe what fencing is and how it is used to protect data during cluster node failures.

## What Is Fencing?

When a node in the cluster fails, this can lead to a loss of data integrity in the cluster because the node could still have access to the cluster resources. For this reason, it is necessary to have an external method to restrict the access of the unresponsive node to the cluster resources. This method is called fencing. The Red Hat Enterprise Linux High Availability Add-On uses fencing to ensure data integrity in the cluster.

Powering the node off often accomplishes fencing because a dead node is not able to do anything. In other cases, fencing uses a combination of operations 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 Enterprise Linux High Availability Add-On does not start resource and service recovery for an unresponsive node until the cluster has fenced that node. For fencing to be correctly configured in your cluster, every node in the cluster should be able to fence every other node.

## Cluster Operation Without Fencing

Without fencing, it is not possible to guarantee data integrity on shared storage resources. For example, a three-node cluster consisting of nodes 1, 2, and 3 has no fencing devices configured. Node 1 has an ext4 file system mounted from shared storage, and it is running a web server serving pages from that file system. If node 1 stops responding on the network, the following chain of events is triggered:

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

## Cluster Operation with Fencing

You must ensure that the unresponsive node no longer has access to the file system *before* another node attempts to mount it. This procedure is called fencing. Fencing stops node 1 from accessing the file system before node 2 has taken over the resource, preventing file-system corruption.

With fencing configured, the chain of events for the previous three-node cluster example would be slightly different:

1. Node 2 and node 3 cut off node 1 from storage.

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

-or-

Node 1 is rebooted and comes up cleanly, joining the cluster.



### Important

Fencing must be configured for all cluster nodes for your cluster to be deemed supportable by Red Hat.

## Fencing Mechanism Overview

There are two main fencing methods: power fencing and storage 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. When the cluster needs to fence a node, it delegates the operation to the fence agent. In turn, the fence agent contacts the fence device to perform the action.

### Describing Power Fencing

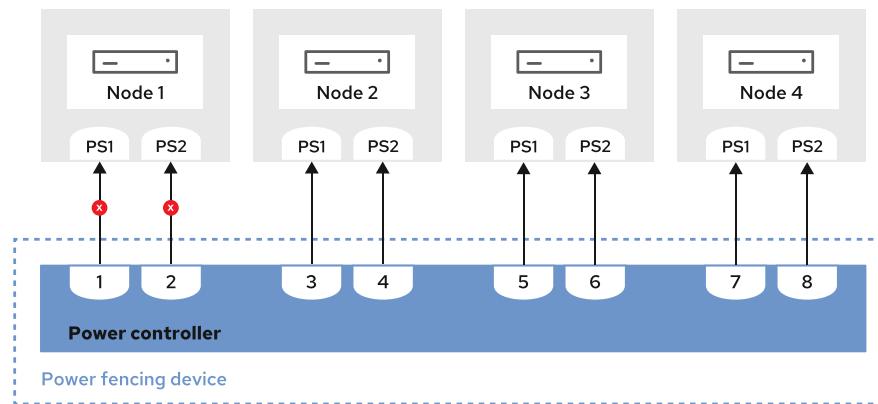
Power fencing entails cutting off power to a server. This fencing method is called STONITH, short for Shoot The Other Node In The Head.

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.

Configuration of the power fencing can turn the target machine off and keep it off, or turn it off and then on again. Turning a machine back on has the added benefit that it should come back up cleanly and rejoin the cluster if the cluster services are 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**

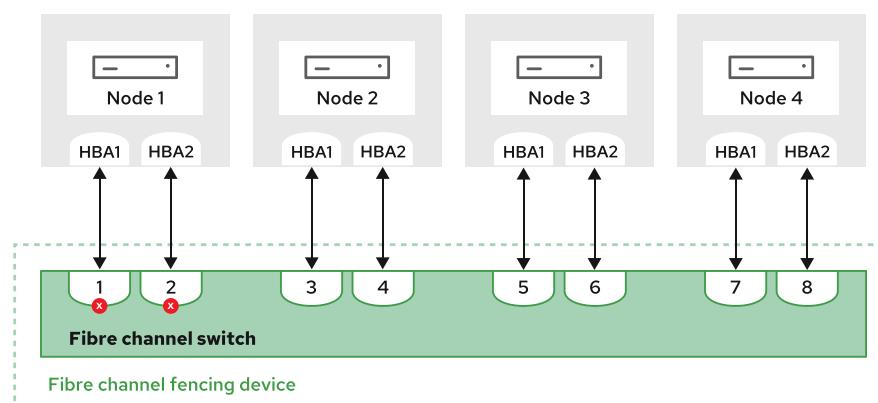
When using power fencing, if the machine has more than one power supply, then all power supplies should turn off before being turned on again. Otherwise, the fenced machine never turns off because it does not lose power.

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

## Describing Storage Fencing

Storage 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 storage fencing without combining it with power fencing, it is the administrator's responsibility to make sure that it joins the cluster again. This is usually done by rebooting or power-cycling the failed node.

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

**Figure 3.2: Storage Fencing Using Multipathed Fibre Channel Storage and a Fiber Switch**

## Combining Fencing Methods

It is possible to combine fencing methods. 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, and if that fails, with storage fencing.



### References

For more information, refer to the *High Availability Add-On Overview* chapter in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_overview-of-high-availability-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_overview-of-high-availability-configuring-and-managing-high-availability-clusters)

**Knowledgebase: "Support Policies for RHEL High Availability Clusters - General Requirements for Fencing/STONITH"**

<https://access.redhat.com/articles/2881341>

## ► Quiz

# Protecting Data with Fencing

Choose the correct answers 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. Ensure 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

► 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:**

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

## ► Solution

# Protecting Data with Fencing

Choose the correct answers 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. Ensure 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

► **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:**

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

# Setting Up Fencing Devices

## Objectives

After completing this section, you should be able to identify supported fencing devices and test them.

## Explaining Fence Device Configuration

Fencing is a requirement for every operational cluster. When setting up fencing for the cluster, the first step is to setup the hardware or software device that performs the actual fencing.

The Red Hat Enterprise Linux High Availability Add-On provides a variety of fencing agents to use with different fence devices. You can install the most used fencing agents supported by Red Hat with the package *fence-agents-all*. The `pcs stonith list` provides a list with the name and description of all the installed fencing agents:

```
[root@node ~]# pcs stonith list
fence_amt_ws - Fence agent for AMT (WS)
fence_apc - Fence agent for APC over telnet/ssh
fence_apc_snmp - Fence agent for APC, Tripplite PDU over SNMP
fence_bla decenter - Fence agent for IBM BladeCenter
...output omitted...
```

Depending on the fence device and fencing agent in use, the required parameters are different. Fencing of a cluster node is successful only if there is communication between the fencing agent and the fencing device. Moreover, the fencing agent must pass the required set of parameters to the fencing device.

A man page is available in the system for every shipped fencing agent, which describes the fencing agent parameters. It is also possible to list the parameters for a specific fencing agent by executing the command `pcs stonith describe fence_agent`.

```
[root@node ~]# pcs stonith describe fence_rhev m
fence_rhev m - Fence agent for RHEV-M REST API

fence_rhev m is an I/O Fencing agent which can be used with RHEV-M REST API to
fence virtual machines.

Stonith options:
  ip (required): IP address or hostname of fencing device
  iport: TCP/UDP port to use for connection with device
...output omitted...
```

In the preceding example, the description shows that the fence agent `fence_rhev m` is appropriate to fence virtual machines by using the Red Hat Virtualization Manager REST API. This fence agent always requires the `ip` parameter, though `iport` is an optional parameter.

## Describing Examples of Fence Device Configuration

Distinct fencing devices require different hardware and software configurations. The hardware needs always to be set up and configured. For the software configuration, it is necessary to document various configuration parameters. The fencing agent uses these documented parameters. The following are some examples of different fencing devices, and their hardware and software configurations.

### Defining 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 (`fence_apc`):

- IP address of the APC fence device.
- User name and password to access the APC fence device.
- Network protocol used to access the device (SSH or telnet).
- The plug number, UUID or identification for each cluster node.

### Defining Management Hardware Fencing

Management hardware, such as iLO, DRAC, or IPMI hardware, can power down, power up, and power cycle systems. At a minimum, system administrators should configure and know the following parameters to use management cards as fencing devices:

- IP address of the management device.
- User name and password to access the management fence device.
- The machines handled by the management fence device.

Red Hat provides different fencing agents for iLO, DRAC, or IPMI hardware, as for example `fence_ilo`, `fence_drac5`, or `fence_ipmilan`. Red Hat provides different fencing agents for iLO, DRAC, or IPMI hardware than for example, `fence_ilo`, `fence_drac5`, or `fence_ipmilan`.

### Defining 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. Red Hat provides the `fence_scsi` fence agent for this purpose.

### Defining Virtual Machine Fencing

The Red Hat Enterprise Linux High Availability Add-On ships with several fencing agents for different hypervisors, for example `fence_rhevm` or `fence_vmware_rest`. They accept the same kind of parameters:

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

## Defining Libvirt Fencing

Cluster nodes that are virtual machines running on a Red Hat Enterprise Linux host with KVM/libvirt, require the daemon `fence-virtd` to be 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 run on different hypervisor machines, as long as all hypervisors have `fence-virt` configured for the same multicast group and use the same shared secret.

## Defining Cloud Instances Fencing

The Red Hat Enterprise Linux High Availability Add-On supports fencing agents for the major cloud providers. For example, `fence-agents-aliyun`, `fence-agents-aws`, or `fence-agents-azure-arm`, are the supported fencing agents for Alibaba Cloud Instances, Amazon Web Services EC2 Instances, or Microsoft Azure Virtual Machines, respectively. Although Red Hat supports these packages, they are not part of the `fence-agents-all` package, and you must install them individually. Because every cloud provider requires different parameters, reviewing them is beyond the scope of this chapter.



### Note

For more information about supported fencing agents for the major cloud providers, see the section *Cluster Platforms and Architectures* in Knowledgebase: "Support Policies for RHEL High Availability Clusters" [<https://access.redhat.com/articles/2912891>]

## Testing Fence Devices

Fencing is crucial for an operational cluster. It is mandatory for a cluster administrator to test the fencing setup thoroughly. It is possible to test the fencing device setup 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. You can also use `pcs stonith describe fence_agent` to investigate possible options. The options required to test fencing differ from agent to agent.

For example, to show all available options for the `fence_ipmilan` fencing agent, you can use:

```
[root@node ~]# fence_ipmilan -h
Usage:
      fence_ipmilan [options]
Options:
      -a, --ip=[ip]                  IP address or hostname of fencing device
      -l, --username=[name]          Login name
      -p, --password=[password]     Login password or passphrase
      -P, --lanplus                 Use Lanplus to improve security of connection
      -A, --auth=[auth]              IPMI Lan Auth type (md5|password|none)
...output omitted...
```

Then, for a fencing device in the `192.168.100.101` IP address you can use `fence_ipmilan` command to fence the node connected to that fence device:

```
[root@node ~]# fence_ipmilan --ip=192.168.100.101 \
> --username=admin --password=password
Success: Rebooted
```

**Note**

When testing fencing with an APC power switch, it is helpful to plug in a lamp for testing instead of the actual cluster node. This makes it visible to see 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. However, it does not verify if fencing configuration in the cluster is correct.

**References**

fence\_\*(8) man pages

For more information, refer to *Chapter 9. Configuring fencing in a Red Hat High Availability cluster* in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-fencing-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-fencing-configuring-and-managing-high-availability-clusters)

**Knowledgebase: "How to configure fence agent 'fence\_xvm' in RHEL cluster"**

<https://access.redhat.com/solutions/917833>

## ► Guided Exercise

# Setting Up Fencing Devices

In this exercise, you will test IPMI-based fencing devices that are integrated with your cluster nodes, by manually running a fencing agent to fence and unfence specific nodes.

## Outcomes

You should be able to manually run a fencing agent to trigger a fence device, and observe the effects of that operation on the node attached to that fence device.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start fencing-verify
```

## Instructions

Each machine in the classroom environment includes a Baseboard Management Controller (BMC) that you can use to power off, power on, or restart the machine by using IPMI over LAN commands. In this course, this is the method that you use as the fencing mechanism for your cluster.

The following table provides the connection details to access those BMC devices.

### BMC Device Parameters

Machine	BMC IP address	User name	Password
nodea	192.168.0.101	admin	password
nodeb	192.168.0.102	admin	password
nodec	192.168.0.103	admin	password
noded	192.168.0.104	admin	password

In this exercise, you verify that the IPMI over LAN fencing method works as expected. Checking that the fencing method you choose works reliably is a mandatory step before deploying a production cluster.

- 1. On your `nodea` machine, install the `fence-agents-all` package. The package installs all the fence agents that Red Hat supports. Update the man page database to reflect the newly installed man pages.
  - 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Install the `fence-agents-all` package.

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

Because in this exercise you only use the IPMI over LAN fencing agent, you could directly install the `fence-agents-ipmilan` package instead.

- 1.3. Run the `mandb` command to refresh the man pages.

```
[root@nodea ~]# mandb  
...output omitted...
```

- 2. List all the `fence_*` man pages, view the man page for the IPMI over LAN fencing agent, and then use the `fence_ipmilan --help` command to list the available parameters.

- 2.1. List all the `fence_*` man pages.

```
[root@nodea ~]# man -k fence_  
...output omitted...  
fence_ipmilan (8)      - Fence agent for IPMI  
...output omitted...
```

- 2.2. View the man page for the IPMI over LAN fencing agent.

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

- 2.3. As an alternative to the man page, you can use the `--help` option of the `fence_ipmilan` command to view the available parameters.

```
[root@nodea ~]# fence_ipmilan --help  
...output omitted...
```

- 3. Monitor the `nodeb` machine and then, from the `nodea` machine, initiate the fencing of `nodeb`. Observe the results to confirm that fencing is working as expected.

- 3.1. In a separate terminal, run the `ping` command to monitor the connection to the `nodeb` machine.

```
[student@workstation ~]$ ping nodeb
PING nodeb.lab.example.com (172.25.250.11) 56(84) bytes of data.
64 bytes from nodeb.lab.example.com (172.25.250.11): icmp_seq=1 ttl=64 time=1.05
ms
64 bytes from nodeb.lab.example.com (172.25.250.11): icmp_seq=2 ttl=64 time=0.297
ms
64 bytes from nodeb.lab.example.com (172.25.250.11): icmp_seq=3 ttl=64 time=0.319
ms
```

Leave the command running.

- 3.2. From your nodea machine, use the `fence_ipmilan` command to fence the nodeb machine. Refer to the table at the beginning of the exercise for the nodeb BMC device parameters. Because in the classroom environment the BMC device can take a long time to reply, increase the timeout to 180 seconds by using the `--power-timeout` option.

```
[root@nodea ~]# fence_ipmilan --lanplus --ip=192.168.0.102 \
> --username=admin --password=password --power-timeout=180
Success: Rebooted
```

In the output of the `ping` command, notice that the `nodeb` machine stops responding. The `nodeb` machine takes a few minutes to restart. Wait for the machine to respond again to the `ping` command. When done, press `Ctrl+C` to exit the `ping` command.

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish fencing-verify
```

This concludes the section.

# Configuring Cluster Fencing Agents

## Objectives

After completing this section, you should be able to configure the cluster to use the correct fencing device when fencing a cluster node.

## Creating a Fence Device

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

```
[root@node ~]# pcs stonith create name fencing_agent [fencing_parameters]
```

The command requires additional arguments:

- `name`: The name for the STONITH fence device.
- `fencing_agent`: The fencing agent used by the fence device.
- `fencing_parameters`: Parameters required for the fencing agent.

As stated in the previous section, you must select the appropriate fencing agent for your fence device. The `pcs stonith list` provides a list with the name and description of all the installed fencing agents.



### Note

The package `fence-agents-all` provides the most used fencing agents supported by Red Hat. However, it does not include all the supported fencing agents, and you should install some of them manually. For more information about the supported fencing agents, see Knowledgebase: "Support Policies for RHEL High Availability Clusters" [<https://access.redhat.com/articles/2912891>]

There are generic properties for all the fencing agents shipped with Red Hat Enterprise Linux High Availability Add-On:

#### pcmk\_reboot\_timeout

This setting defines the time to wait for fencing to complete in seconds. The default value is 60 seconds, but you can define a device-specific timeout with the `pcmk_reboot_timeout` property. If a fencing action takes longer than this timeout, then the cluster considers that the fence operation has failed. If you want to set this value at the cluster level, you can do it through `pcs property set stonith-timeout=180s`.

#### pcmk\_host\_list

This parameter provides a space-separated list of nodes controlled by the fencing device. It is required if `pcmk_host_check` is set to `static-list`.

#### pcmk\_host\_map

You usually use this parameter with fence agents that you can use to fence multiple nodes. For example, with the `fence_rhev` agent, you create only one STONITH resource to fence

all the cluster nodes. You list the nodes in `pcmk_host_map` and associate each one with its Red Hat Virtualization VM name. The list is a semicolon-separated list of `nodename:port` mappings, such as "node1.example.com:1;node2.example.com:2".

#### `pcmk_host_check`

This parameter defines how the cluster determines the nodes that may be controlled from the fencing device. Possible values are:

- `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 names of the cluster nodes.
- `static-list`: The cluster uses a list of nodes provided by `pcmk_host_list`, or a list of `nodename:port` mappings provided by `pcmk_host_map`.
- `none`: The cluster assumes that every fencing device can fence every node.

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.

The cluster defaults to querying the fence device for a list of ports if you do not set any of the `pcmk_host_*` options. If a port name matches the name of a cluster node, then the cluster uses that port to fence that device.

You can also create a fencing device for a single node by setting the `pcmk_host_list` parameter to the name of the node to fence.



#### Note

Not all fencing devices support listing ports for use with `pcmk_host_check="dynamic-list"`. In those cases, you must use the option `pcmk_host_map`.

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 you can set for a particular fence device:

```
[root@node ~]# pcs stonith describe fence_apc
Stonith options for: fence_apc
  ip (required): IP address or hostname of fencing device
  username (required): Login name
  password: Login password or passphrase
...output omitted...
```



#### Warning

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`. This parameter allows the node to automatically get unfenced before it reboots. It also starts the cluster services in the node. Then, the node can rejoin the cluster.

To create a fencing device using the `fence_apc` fencing agent, you can use:

```
[root@node ~]# pcs stonith create fence_node1 fence_apc ip=10.168.0.101 \
> username=admin password=s3cr3t pcmk_host_list="node1.example.com"
[root@node ~]#
```

## Displaying Fencing Devices

The `pcs stonith status` 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, then the device is operational; if it is Stopped, then the fence device is not operational.

```
[root@node ~]# pcs stonith status
* fence_node1 (stonith:fence_apc): Started node1.example.com
* fence_node2 (stonith:fence_apc): Started node2.example.com
* fence_node3 (stonith:fence_apc): Started node3.example.com
```

The `pcs stonith config` command shows the configuration options of all the STONITH resources. You can also specify the name of the STONITH resource as a parameter to show the configuration options only for that resource.

```
[root@node ~]# pcs stonith config fence_node1
Resource: fence_node1 (class=stonith type=fence_apc)
  Attributes: ip=10.168.0.101 password=s3cr3t pcmk_host_list=node1.example.com
    username=admin
  Operations: monitor interval=60s (fence_node1-monitor-interval-60s)
```

## Changing Fencing Devices

You can change fencing device options with the `pcs stonith update fence_device_name` command. This allows you to add a new fence device option or change an existing one.

For example, the fencing device `fence_node2` wrongly fences `node1` instead of `node2`. You can correct it by executing:

```
[root@node ~]# pcs stonith update fence_node2 pcmk_host_list=node2.example.com
```

## Removing Fencing Devices

At some point, it might be necessary to remove a fencing device from the cluster. This might happen because you want to remove the corresponding cluster node permanently, or you want to use a different fencing mechanism to fence the node. The command `pcs stonith delete fence_device_name` allows you to remove a fencing device from the cluster. For example, to remove the fencing device `fence_node4` from the cluster, execute:

```
[root@node ~]# pcs stonith delete fence_node4
Attempting to stop: fence_node4...Stopped
Deleting Resource - fence_node4
```

## Testing Fence Configuration

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

- By using the command `pcs stonith fence nodename`. This attempts to fence the requested node. You should run the command from other node in the cluster, not from the node you want to fence. If the command runs successfully, then the cluster can fence this node, and fencing on that node works correctly.
- 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 node has failed, and fence it. This tests the cluster's ability to detect a failed node as well.



## References

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

For more information, refer to *Chapter 9. Configuring fencing in a Red Hat High Availability cluster* in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-fencing-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-fencing-configuring-and-managing-high-availability-clusters)

**Knowledgebase: "What format should I use to specify node mappings to stonith devices in `pcmk_host_list` and `pcmk_host_map` in a RHEL 6, 7, or 8 High Availability cluster?"**

<https://access.redhat.com/solutions/2619961>

## ► Guided Exercise

# Configuring Cluster Fencing Agents

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

## Outcomes

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

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start fencing-inspect
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines. It also configures the fencing resources for those machines.

## Instructions

In the first part of the exercise, you inspect and then test the existing fencing configuration. In the second part of the exercise, you remove all that fencing configuration and then configure a new STONITH resource that uses a custom fencing agent.

- 1. Inspect the fencing configuration on your cluster. This should show three separate fencing resources, one for each node. Confirm that this configuration is functional by fencing your nodeb machine using the `pcs` command.

- 1.1. Connect to nodea and become the root user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. Inspect the fencing configuration on your cluster.

```
[root@nodea ~]# pcs stonith config
Resource: fence_nodea (class=stonith type=fence_ipmilan)
  Attributes: ip=192.168.0.101 lanplus=1 password=password
  pcmk_host_list=nodea.private.example.com power_timeout=180 username=admin
  Operations: monitor interval=60s (fence_nodea-monitor-interval-60s)
Resource: fence_nodeb (class=stonith type=fence_ipmilan)
  Attributes: ip=192.168.0.102 lanplus=1 password=password
  pcmk_host_list=nodeb.private.example.com power_timeout=180 username=admin
  Operations: monitor interval=60s (fence_nodeb-monitor-interval-60s)
```

**Chapter 3 |** Isolating Malfunctioning Cluster Nodes

```
Resource: fence_nodec (class=stonith type=fence_ipmilan)
  Attributes: ip=192.168.0.103 lanplus=1 password=password
  pcmk_host_list=nodec.private.example.com power_timeout=180 username=admin
  Operations: monitor interval=60s (fence_nodec-monitor-interval-60s)
```

- 1.3. Verify operation by fencing the nodeb machine. The command might take up to three minutes to complete.

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

- 1.4. In a separate terminal, log in to the nodeb machine and use the `pcs status` command to confirm that the nodeb machine has rejoined the cluster. You might have to wait a few minutes for the nodeb machine to restart.

```
[student@workstation ~]$ ssh nodeb
...output omitted...
[student@nodeb ~]$ sudo -i
[sudo] password for student: student
[root@nodeb ~]# pcs status
Cluster name: cluster1
...output omitted...
Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]
...output omitted...
```

- ▶ 2. Your cluster is already configured to use `fence_ipmilan` as a fencing agent. The classroom also provides a secondary fence device that you can configure by using a custom fencing agent called `fence_rh436`. This agent communicates with the virtual classroom chassis to control the classroom machines. The chassis assigns a slot number, or plug, to each machine according to the following table.

**Classroom Machines Plug Numbers**

Machine	Plug number
nodea.private.example.com	1
nodeb.private.example.com	2
nodec.private.example.com	3
noded.private.example.com	4

The IP address of the chassis is 192.168.0.100. The user name is `admin` and the password is `password`.

**Note**

Red Hat does not support custom fencing scripts. The `fence_rh436` custom agent is for demonstration purposes only.

Remove the three current STONITH resources, install the *fence-agents-rh436* package on the three machines, and then create a single new STONITH resource called *fence\_custom*, by using the *fence\_rh436* fencing agent. Set the timeout to 180 seconds.

- 2.1. Remove the three existing STONITH resources.

```
[root@nodea ~]# pcs stonith delete fence_nodea
Attempting to stop: fence_nodea... Stopped
[root@nodea ~]# pcs stonith delete fence_nodeb
Attempting to stop: fence_nodeb... Stopped
[root@nodea ~]# pcs stonith delete fence_nodec
Attempting to stop: fence_nodec... Stopped
```

- 2.2. Install the *fence-agents-rh436* package on the nodea, nodeb, and nodec machines. The preparation script has copied the RPM file in the */root/fencing-inspect/* directory.

```
[root@nodea ~]# yum install /root/fencing-inspect/fence-agents-rh436-*.rpm
```

```
[root@nodeb ~]# yum install /root/fencing-inspect/fence-agents-rh436-*.rpm
```

```
[root@nodec ~]# yum install /root/fencing-inspect/fence-agents-rh436-*.rpm
```

- 2.3. Configure a fence device by creating a new STONITH resource called *fence\_custom* that uses the *fence\_rh436* fencing agent. Use the *pcmk\_host\_map* parameter to associate each cluster node to its plug number. Set the timeout to 180 seconds by using the *power\_timeout* parameter. For your convenience, you can copy and paste the following command from the */root/fencing-inspect/stonith.txt* file.

The last line in the following command is very long. It has been split so that it displays correctly in the guide. If you type the command, do not add the line break or a space.

```
[root@nodea ~]# pcs stonith create fence_custom fence_rh436 \
> ip=192.168.0.100 username=admin password=password power_timeout=180 \
> pcmk_host_map="nodea.private.example.com:1;nodeb.private.example.com:2;
nodec.private.example.com:3"
```

- 2.4. Verify operation by fencing the nodeb machine. The command might take up to three minutes to complete.

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

- 2.5. Log in to the nodeb machine and use the *pcs status* command to confirm that the nodeb machine has rejoined the cluster. You might have to wait a few minutes for the nodeb machine to restart.

```
[student@workstation ~]$ ssh nodeb
...output omitted...
[student@nodeb ~]$ sudo -i
[sudo] password for student: student
[root@nodeb ~]# pcs status
Cluster name: cluster1
...output omitted...
Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]
...output omitted...
```

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish fencing-inspect
```

This concludes the section.

## ▶ Lab

# Isolating Malfunctioning Cluster Nodes

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

## Outcomes

You should be able to configure fencing.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start fencing-review
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines. It also prepares an Ansible Playbook on the `workstation` machine to help you deploy the custom fence agent on the cluster nodes.

## Instructions

1. Create a fencing resource called `fence_all` on nodea, nodeb and nodec machines, by using the `fence_rh436` custom fencing agent. The `fence_rh436` fencing agent is provided in the `fence-agents-rh436` package. To install that agent on the nodea, nodeb, and nodec machines, run the `/home/student/labs/fencing-review/playbook.yml` Ansible Playbook.

This agent communicates with the virtual classroom chassis to control the classroom machines. The chassis assigns a slot number, or plug, to each machine according to the following table.

### Classroom Machines Plug Numbers

Machine	Plug number
nodea.private.example.com	1
nodeb.private.example.com	2
nodec.private.example.com	3

The IP address of the chassis is `192.168.0.100`. The chassis user name is `admin` and the password is `password`.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

2. Manually fence `nodeb` to test your work.

## Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade fencing-review
```

## Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish fencing-review
```

This concludes the section.

## ► Solution

# Isolating Malfunctioning Cluster Nodes

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

### Outcomes

You should be able to configure fencing.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start fencing-review
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines. It also prepares an Ansible Playbook on the `workstation` machine to help you deploy the custom fence agent on the cluster nodes.

### Instructions

1. Create a fencing resource called `fence_all` on nodea, nodeb and nodec machines, by using the `fence_rh436` custom fencing agent. The `fence_rh436` fencing agent is provided in the `fence-agents-rh436` package. To install that agent on the nodea, nodeb, and nodec machines, run the `/home/student/labs/fencing-review/playbook.yml` Ansible Playbook.

This agent communicates with the virtual classroom chassis to control the classroom machines. The chassis assigns a slot number, or plug, to each machine according to the following table.

#### Classroom Machines Plug Numbers

Machine	Plug number
nodea.private.example.com	1
nodeb.private.example.com	2
nodec.private.example.com	3

The IP address of the chassis is `192.168.0.100`. The chassis user name is `admin` and the password is `password`.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

- 1.1. Install the `fence-agents-rh436` package on the nodea, nodeb, and nodec machines. To do so, run the `playbook.yml` Ansible Playbook in the `/home/student/labs/fencing-review/` directory. As the `student` user on the `workstation` machine, change to the `/home/student/labs/fencing-review/` directory.

**Chapter 3 |** Isolating Malfunctioning Cluster Nodes

```
[student@workstation ~]$ cd /home/student/labs/fencing-review  
[student@workstation fencing-review]$
```

- 1.2. Run the playbook.

```
[student@workstation fencing-review]$ ansible-playbook playbook.yml  
...output omitted...
```

- 1.3. Connect to nodea and become the root user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.4. Configure a fence device by creating a new STONITH resource called fence\_all that uses the fence\_rh436 fencing agent. Use the pcmk\_host\_map parameter to associate each cluster node to its plug number described in the preceding table. Set the timeout to 180 seconds using the power\_timeout parameter.

The last line in the following command is very long. It has been split so that it displays correctly in the guide. If you type the command, do not add the line break or a space.

```
[root@nodea ~]# pcs stonith create fence_all fence_rh436 \  
> ip=192.168.0.100 username=admin password=password power_timeout=180 \  
> pcmk_host_map="nodea.private.example.com:1;nodeb.private.example.com:2;  
nodec.private.example.com:3"  
[root@nodea ~]#
```

- 1.5. Verify the fencing configuration on your cluster has been created successfully.

```
[root@nodea ~]# pcs stonith config  
Resource: fence_all (class=stonith type=fence_rh436)  
  Attributes: ip=192.168.0.100 password=password  
    pcmk_host_map=nodea.private.example.com:1;nodeb.private.example.com:2;  
    nodec.private.example.com:3 power_timeout=180 username=admin  
Operations: monitor interval=60s (fence_all-monitor-interval-60s)
```

2. Manually fence nodeb to test your work.

- 2.1. Verify operation by manually fencing the nodeb machine.

You can view the console of your nodeb machine to verify that it has been fenced and is rebooting.

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

- 2.2. Observe nodeb being fenced, by using pcs quorum status on nodea.

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

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade fencing-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish fencing-review
```

This concludes the section.

# 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 by using `pcs stonith create`.
- How to remove fencing resources by using `pcs stonith delete`.
- How to update fencing resources by 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 a single high availability cluster resource.
- Assemble a resource group from multiple related resources.
- Manually control and relocate resource groups in a high availability cluster.
- Set constraints that control on which cluster nodes resources and resource groups can run.

### Sections

- Creating and Configuring a Cluster Resource (and Guided Exercise)
- Creating and Configuring Resource Groups (and Guided Exercise)
- Managing Resource Groups (and Guided Exercise)
- Configuring Constraints (and Guided Exercise)

### Lab

- Creating and Configuring Resources

# Creating and Configuring a Cluster Resource

---

## Objectives

After completing this section, you should be able to create and configure a single high availability cluster resource.

## Describing 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 required components to provide a service to consumers are a resource.

With the High Availability Add-On, all resources are monitored independently from each other. In order to accomplish this monitoring task, resource agents control resources. Pacemaker can manage different kinds of resource agents:

### LSB

Linux Standard Base-compatible init scripts that reside 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 and located beneath `/usr/lib/ocf/resource.d/provider`. Most of the agents that are shipped by Red Hat use heartbeat as the provider.

### Systemd

Systemd unit files, which are the standard for defining and managing services on Red Hat Enterprise Linux 8.

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

## Identifying Commonly Used Resources

The following list includes some commonly used resources:

### Filesystem

Used for mounting a file system. It can be a local file system, a file system on an iSCSI or Fibre Channel device, or a remote file system such as an NFS export or an SMB share.

### IPAddr2

`IPAddr2` assigns a floating IP address to a resource group. Also, a separate `IPAddr` resource uses an older method of assigning a IP address. On Red Hat Enterprise Linux 8, `IPAddr` is a symbolic link to `IPAddr2`.

### apache

This resource starts an Apache `httpd` service. Unless otherwise configured, it uses the configuration from `/etc/httpd`.

### mysql

This resource controls a MySQL database. You can configure databases for stand-alone operation, a clone set with external replication, or as a full primary/secondary setup.

## Describing Resource Groups

A service usually consists of more than one resource. For example, a typical web service consists of an IP address resource, an Apache resource, and a shared file system resource for the DocumentRoot folder 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 that they are added to the resource group and get 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. Resource group is described later in this chapter.

The command `pcs status resources` lists all resources. The following example shows three resources:

```
[root@node ~]# pcs status resources
  * myfirstip (ocf::heartbeat:IPaddr2): Started node1.example.com
  * myfirstfs (ocf::heartbeat:Filesystem): Started node1.example.com
  * myfirstweb (ocf::heartbeat:apache): Started node1.example.com
```

## Describing Cluster Resources

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

## Listing Resource Agents

It is important for a cluster administrator to get an overview of all available resource agents that the High Availability Add-On provides. The command `pcs resource list` lists all resource agents that are known to Pacemaker.

```
[root@node ~]# pcs resource list
ocf:heartbeat:apache - Manages an Apache Web server instance
ocf:heartbeat:aws-vpc-move-ip - Move IP within a VPC of the AWS EC2
ocf:heartbeat:aws-vpc-route53 - Update Route53 VPC record for AWS EC2
ocf:heartbeat:awseip - Amazon AWS Elastic IP Address Resource Agent
ocf:heartbeat:awsvip - Amazon AWS Secondary Private IP Address Resource Agent
ocf:heartbeat:azure-events - Microsoft Azure Scheduled Events monitoring agent
ocf:heartbeat:azure-lb - Answers Azure Load Balancer health probe requests
ocf:heartbeat:conntrackd - This resource agent manages conntrackd
ocf:heartbeat:CTDB - CTDB Resource Agent
...output omitted...
```

## Listing 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 the following command:

```
[root@node ~]# pcs resource describe Filesystem
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
ocf:heartbeat:Filesystem - Manages filesystem mounts

...output omitted...

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".

...output omitted...
```

## Creating Cluster Resources

By executing the `pcs resource create resource_name resource_provider resource_parameters --group=group_name` command, you can instantly add a cluster resource to a resource group when creating it. If the group that is 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 can be omitted to create a stand-alone resource that is not part of any resource group. To create the `myfs` resource by 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 the following command:

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



### Important

When configuring a resource, all nodes that potentially could use the resource must have access to that resource; otherwise, it will fail to start and cause an issue when it tries to relocate. For example, before you configure an `apache` resource, ensure that the `httpd` package is installed on all cluster nodes.

## Listing Configured Cluster Resources

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

```
[root@node ~]# pcs resource status
* Resource Group: mygroup:
  * myip (ocf::heartbeat:IPAddr2): Started node1.example.com
  * myfs (ocf::heartbeat:Filesystem): Started node1.example.com
  * myserv (ocf::heartbeat:apache): Started node1.example.com
```

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

## Modifying Resource Parameter

At times, you might want to fine-tune an already created cluster resource or to change options to refine the existing configuration. The High Availability Add-On allows for changing cluster resource configurations with the `pcs resource update` command. For example, to change the device of the cluster resource `myfs` to use `/dev/sda1` as the device, execute the following command:

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

## Deleting Cluster Resources

You can remove cluster resources and resource group configurations from the cluster if they are not required any more. The `pcs resource delete resource_name` command allows for deleting single resources, as well as resource groups with all resources that are attached to it.

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



### References

[pcs\(8\) man page](#)

For more information, refer to the *Configuring cluster resources* chapter in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-cluster-resources-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-cluster-resources-configuring-and-managing-high-availability-clusters)

### OCF Resource Agents

<https://github.com/ClusterLabs/resource-agents>

### The OCF Resource Agent Developer's Guide

<https://github.com/ClusterLabs/resource-agents/blob/master/doc/dev-guides/ra-dev-guide.asc>

## ► Guided Exercise

# Creating and Configuring a Cluster Resource

In this exercise, you will create a single cluster resource that mounts an NFS file system on a cluster node.

### Outcomes

You should be able to create a basic cluster resource.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start resources-create
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines. It also shares the `/srv/www` directory on the storage machine using NFS.

## Instructions

This exercise is a walk-through in which you set up a single cluster resource that runs on one active node at a time. It guides you to create a `Filesystem` resource named `wwwfs`, which mounts the NFS export `storage.san01.example.com:/srv/www` read-only on the node's `/mnt` directory.

It does not provide a useful network service to any clients. It only makes the NFS export available to one of your nodes. In subsequent exercises, you create a more complex configuration that combines the NFS export with other resources in order to provide a more useful highly available service.

When the resource is operating, in this exercise you fence the node on which the resource is running, and you observe how the cluster restarts the resource on a different node.

- ▶ 1. Create a `Filesystem` resource, named `wwwfs`, to mount the NFS export `storage.san01.example.com:/srv/www` read-only on `/mnt`.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. Inspect the possible options for the `Filesystem` resource.

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

- 1.3. Create the `wwwfs` resource by using all relevant options. For your convenience, you can copy and paste the following command from the `/root/resources-create/resource.txt` file.

```
[root@nodea ~]# pcs resource create wwwfs Filesystem \  
> device=storage.san01.example.com:/srv/www directory=/mnt \  
> fstype=nfs options=ro  
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 1.4. Use the `pcs status` command to see the new resource.

```
[root@nodea ~]# pcs status  
...output omitted...  
Full List of Resources:  
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com  
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com  
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com  
* wwwfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com  
...output omitted...
```

The cluster started the resource on the `nodea` machine. On your system, it might be running on a different machine.

- 1.5. On the node where the `wwwfs` resource is started, inspect the contents of the `/mnt` directory. You should see some files.

```
[root@nodea ~]# ls /mnt  
html test.txt
```

- ▶ 2. Disrupt the cluster network communications on the node that is currently running the `wwwfs` resource. Observe how the cluster moves the resource to a different node.
- 2.1. Add temporary `firewalld` rules to block the `corosync` traffic both inbound and outbound. Use the `/root/resources-create/firewall-block.sh` script that the exercise preparation command deployed for you.

```
[root@nodea ~]# /root/resources-create/firewall-block.sh  
success  
success
```

- 2.2. On a different node, run the `pcs status` command.

```
[root@nodeb ~]# pcs status  
...output omitted...  
Node List:  
* Online: [ nodeb.private.example.com nodec.private.example.com ]  
* OFFLINE: [ nodea.private.example.com ]
```

**Full List of Resources:**

```
* fence_nodea (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodec.private.example.com
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
* wwwfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
...output omitted...
```

Observe that the cluster fences the node that previously ran the **wwwfs** resource. That node is now offline. Notice that the cluster moves the **wwwfs** resource to another node.

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish resources-create
```

This concludes the section.

# Creating and Configuring Resource Groups

## Objectives

After completing this section, you should be able to assemble a resource group from multiple related 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 an `httpd` instance on a second, and assigning a floating IP address to a third node does not create a cluster resource that consumers can use.

Resource groups join different cluster resources, which forces all resources in a group to start and stop in a specific order, on the same node, creating clustered services for consumers.

## Creating a Clustered Apache Service

For a clustered Apache HTTP 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 to be 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 it requires a mechanism to sync the content.)
3. An `apache` resource that controls 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 named `webip`, by using the resource agent `IPAddr2` with the floating IP address `172.25.99.80/24` as part of the `myweb` resource group, execute the following commands:

```
[root@node ~]# pcs resource create webip IPAddr2 \
> ip=172.25.99.80 \
> cidr_netmask=24 \
> --group=myweb
```

If the target resource group does not exist, then it gets automatically created.

The web server also needs access to shared storage to host the content for the web server. In order to create a `Filesystem` resource for an NFS mount `storage.example.com:/exports/www` on `/var/www`, execute the following commands:

```
[root@node ~]# pcs resource create webfs Filesystem \
> device=storage.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, it can entail setting proper SELinux contexts; for NFS, it means 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. To do so, the `httpd` package must be installed and `firewalld` must allow access to the relevant ports on each cluster node that 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` or `curl` packages must be installed on all cluster nodes that serve the monitored Apache web server. The status URL must be available only from `127.0.0.1`. To create the required status URL, the following section must be present in the `/etc/httpd/conf/httpd.conf` file:

```
<Location /server-status>
  SetHandler server-status
  Require local
</Location>
```

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

The resource named `webserver` that uses 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@node ~]# pcs resource create webserver apache \
> configfile="/etc/httpd/conf/httpd.conf" \
> statusurl="http://127.0.0.1/server-status" \
> --group=myweb
```

The previous example used the default values. If `configfile` is unspecified, then it takes the default file `/etc/httpd/conf/httpd.conf` for Red Hat Enterprise Linux. If `statusurl` is unspecified, then it is inferred from the Apache configuration file.

## Tune Resource Operations

To ensure that resources remain healthy, add monitoring operation to the cluster resources. Those operation parameters can be tuned by a system administrator when creating a resource by adding `op operation`, followed by one or more operation options:

**name**`name=value`

It is the action to perform. The values are `start`, `stop`, and `monitor`.

**interval**`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, then the value is set to 60s.

**timeout**`timeout=value`

Defines the amount of time to wait before an operation is declared as failed if it did not complete before timeout is reached.

**on-fail**`on-fail=action`

The action to take if the operation failed:

- `ignore`: Ignores any failure of the operation.
- `block`: Stops performing any operations. This is the default action for a failed `stop` operation if fencing is not configured on the cluster.
- `stop`: Stops the resource from being active in the cluster. This is the default action for a failure of the `start` and `monitor` operations.
- `restart`: Stops and starts the resource.
- `fence`: Fences the node on which the resource failed. This is the default action for a failed `stop` operation for a cluster where fencing is configured.
- `standby`: Moves all resources away from the cluster node that the resource was running on.

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

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

Use the `pcs resource config resource` command to display the options that are configured for that resource:

```
[root@node ~]# pcs resource config 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: monitor interval=20s timeout=30s (webserver-monitor-interval-20s)
               start interval=0s timeout=40s (webserver-start-interval-0s)
               stop interval=0s timeout=60s (webserver-stop-interval-0s)
```

## Chapter 4 | Creating and Configuring Resources

Resource operations can be added with the `pcs resource op add` command or removed with the `pcs resource op remove` command. To remove the existing monitoring operation from the `webserver` resource, execute the following command:

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

You can always set new parameters to an existing resource using the `pcs resource op add` command. For example, if you want to add to the existing `webserver` resource a monitoring interval of 10 seconds, a monitoring timeout of 15 seconds, and the option to fence the node if the monitoring operation fails, execute the following commands:

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



### Important

If a resource fails to start, then the resource failcount is 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 look at the output of `pcs resource debug-start resourcename`, which displays the error messages of the resource start attempt, and then repairs the issue. When repaired, the failcount of the resource must be reset for the resource to be allowed to start again on that node with: `pcs resource cleanup resourcename`.

## Adding Resources to a Resource Group

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

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

## Removing Resources from a Resource Group

A resource that is part of a resource group might be removed from the resource group with `pcs resource group remove groupname resourcename`. The resource then still exists in the cluster but is not part of the resource group anymore. If the last resource is removed from the resource group, then the resource group is removed as well. To remove the `myresource` resource from the `mygroup` resource group, execute the following command:

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

## Resource Ordering

In some cases, a dependency might exist between resources added to a group. For example, a web server that is configured to listen on a specific IP address cannot start until the node configures

the IP address. Usually, resources start in the order that they are added to a group, and are stopped in reverse order.

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



## References

`pcs(8)` man page

For more information, refer to the *Configuring cluster resources* chapter in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-cluster-resources-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-cluster-resources-configuring-and-managing-high-availability-clusters)

For more information, refer to the *Resource monitoring operations* chapter in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_resource-monitoring-operations-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_resource-monitoring-operations-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Creating and Configuring Resource Groups

In this exercise, you will configure a highly available web server by configuring a resource group consisting of multiple resources.

### Outcomes

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

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start resources-group
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines and shares the `/srv/www` directory on the `storage` machine by using NFS. It also prepares an Ansible Playbook on the `workstation` machine to help you to deploy the Apache HTTP Server on the cluster nodes.

### Instructions

This exercise walks you through the configuration of a resource group that implements a highly available web server. The resource group consists of three resources:

- An `IPAddr2` resource named `firstwebip` for the resource group's `172.25.250.80/24` IP address.
- A `Filesystem` resource named `firstwebfs`, which mounts the NFS file system `storage.san01.example.com:/srv/www` read-only on `/var/www`.
- An `apache` resource named `firstwebserver`, which manages the service for the Apache HTTP Server.

After creating the resource group, you should be able to access the web server at `http://172.25.250.80`.

To confirm that the resource group is providing a highly available service, when it is running you will disrupt networking on the resource group's current node.

This causes the cluster to fence it. You then confirm that the service restarts on one of the remaining nodes in the cluster.

- 1. To prepare the `nodea`, `nodeb`, and `nodec` machines to serve web content, review and then run the Ansible Playbook that the exercise preparation command deployed on the `workstation` machine, under the `/home/student/labs/resources-group/` directory.

1. As the `student` user on the `workstation` machine, change to the `/home/student/labs/resources-group` project directory.

```
[student@workstation ~]$ cd /home/student/labs/resources-group  
[student@workstation resources-group]$
```

- 1.2. Look at the content of the `playbook.yml` playbook. You do not have to modify anything in that file.

```
[student@workstation resources-group]$ cat playbook.yml  
---  
- name: Ensure the Apache HTTP Server is installed  
  hosts: nodes  
  become: yes  
  gather_facts: no  
  
  tasks:  
    - name: Ensuring the httpd package is installed ①  
      yum:  
        name: httpd  
        state: present  
  
    - name: Ensuring the required ports are open ②  
      firewalld:  
        service: http  
        permanent: yes  
        state: enabled  
        immediate: yes  
  
    - name: Ensuring SELinux allows Apache HTTP Server to access NFS shares ③  
      seboolean:  
        name: httpd_use_nfs  
        state: yes  
        persistent: yes  
...
```

- ① The first task installs the `httpd` package on the cluster nodes.
- ② The second task configures the firewall to accept HTTP traffic.
- ③ The last task sets the `httpd_use_nfs` SELinux Boolean to `true` so that the Apache HTTP Server can access its document root from the NFS share.

- 1.3. Run the playbook.

```
[student@workstation resources-group]$ ansible-playbook playbook.yml
```

- 2. Create the three cluster resources, and be sure to put them in the `firstweb` resource group. For your convenience, you can copy and paste the commands to create the resources from the `/root/resources-group/resources.txt` file on the `nodea` machine.

- 2.1. Connect to `nodea` and become the `root` user.

```
[student@workstation resources-group]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 2.2. Create the `IPAddr2` resource named `firstwebip` by using the `172.25.250.80/24` address.

```
[root@nodea ~]# pcs resource create firstwebip IPAddr2 \  
> ip=172.25.250.80 \  
> cidr_netmask=24 \  
> --group=firstweb  
Assumed agent name 'ocf:heartbeat:IPAddr2' (deduced from 'IPAddr2')
```

- 2.3. Create the `Filesystem` resource named `firstwebfs` to mount `storage.san01.example.com:/srv/www` read-only on `/var/www`.

```
[root@nodea ~]# pcs resource create firstwebfs Filesystem \  
> device=storage.san01.example.com:/srv/www \  
> directory=/var/www \  
> fstype=nfs \  
> options=ro \  
> --group=firstweb  
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 2.4. Create the `apache` resource named `firstwebserver`.

```
[root@nodea ~]# pcs resource create firstwebserver apache --group=firstweb  
Assumed agent name 'ocf:heartbeat:apache' (deduced from 'apache')
```

- ▶ 3. Confirm that the web server is working as expected and then purposely fail the node where the resource group is running to verify failover.

- 3.1. In a separate terminal on the `workstation` machine, use the `curl` command to access `http://172.25.250.80`. The web server that is running in the cluster should return a sample page.

```
[student@workstation ~]$ curl http://172.25.250.80  
Hello, world!
```

- 3.2. Use the `pcs status` command to identify the node that is currently running the `firstweb` resource group.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip    (ocf::heartbeat:IPAddr2): Started nodea.private.example.com
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache): Started nodea.private.example.com
...output omitted...
```

The preceding output shows that the resource group is running on the nodea machine. On your system, the group might be running on a different node.

- 3.3. Disrupt the cluster network communications on the node that is currently running the resource group. To do so, use the /root/resources-group/firewall-block.sh script that the exercise preparation command deployed for you.

```
[root@nodea ~]# /root/resources-group/firewall-block.sh
success
success
```

- 3.4. On another node, use the pcs status command to monitor the fencing and the resource relocation process. Failover can take several minutes to complete.

```
[root@nodeb ~]# watch pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip    (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
  * firstwebserver (ocf::heartbeat:apache): Started nodeb.private.example.com
...output omitted...
```

- 3.5. On the workstation machine, run the curl command again to confirm that the firstweb group is still functional.

```
[student@workstation ~]$ curl http://172.25.250.80
Hello, world!
```

## Finish

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish resources-group
```

This concludes the section.

# Managing Resource Groups

## Objectives

After completing this section, you should be able to manually control and relocate resource groups in a high availability cluster.

## Managing Services

The cluster administrator can use the High Availability Add-On to control the resources and resource groups that run in the cluster. As a cluster administrator, you start and stop cluster resources and resource groups as needed. Restrictions enforce prohibiting a cluster resource or resource group temporarily from migrating to a specific cluster node, moving them away from the node they are running on, and temporarily banning the resource from migrating back to the node. Restrictions help for starting maintenance on particular cluster nodes and minimize service downtime by reducing the amount of service migration by manual intervention with the cluster operation.

### Stopping and Starting Cluster Resources

As a cluster administrator, you can stop cluster resources and resource groups at any time to ensure that they are not running on the cluster. Run the `pcs resource disable resourcename` command to stop a running resource entirely and prevent the cluster from starting it again.

```
[root@node ~]# pcs resource disable myresource
```

Use the `pcs resource enable resourcename` command to allow the cluster to start the cluster resource or resource group.

```
[root@node ~]# pcs resource enable myresource
```

### Moving Cluster Resources

Cluster resources and resource groups might be moved away from the cluster node where they are currently running. Optionally, specify a target node to indicate on which node to run the cluster resource or resource group that you are moving. If a cluster node has a maintenance window, for example, to apply errata, then use this feature. Run the `pcs resource move resourcename` command to move the resource group `myresource` to `node2`.

```
[root@node ~]# pcs resource move myresource node2.example.com
```

The `pcs resource move` command adds a constraint to the resource to prevent it from running on the node where it currently runs. How to list and delete constraints is described later in this chapter. The `move` command without a target creates a `Disable` rule for the original node. The `move` command with a target creates for the original node a `Enable` rule.

## Controlling Resource Migration

As a cluster administrator, you can temporarily prevent the migration of a resource to a specific cluster node. The `pcs resource ban resourcename` command prohibits the resource from running on the node where it currently runs. Optionally, you can add a particular node as a parameter on the command line to restrict the cluster resource from migrating to the specified node. To prevent the resource group `myresource` from running on `node2.example.com`, execute the following command:

```
[root@node ~]# pcs resource ban myresource node2.example.com
```

Both `pcs resource move` and `pcs resource ban` create a temporary constraint rule on the cluster. Constraints are used, among other reasons, to influence which resources can run where. How to list and delete constraints is described later in this chapter.

## Listing Constraints

For a cluster administrator, it is important to understand the behavior of the configured services on the cluster. The `pcs constraint list` command allows an administrator to get an overview of the currently configured constraints in the cluster. In the following example, the `myresourcegroup` resource group is banned from running on `node2.example.com`.

```
[root@node ~]# pcs constraint list
Location Constraints:
Resource: myresourcegroup
  Disabled on:
    Node: node2.example.com (score:-INFINITY) (role:Started)
Ordering Constraints:
Colocation Constraints:
Ticket Constraints:
```

## Deleting Temporary Resource Restrictions

As a cluster administrator, you can remove the temporary restrictions for a resource with the `pcs resource clear resourcename` command. You can also add as a parameter a particular cluster node to remove restrictions for the given resource on that specific cluster node. To clear the ban restriction for the `myresource` resource group on `node2`, execute the following command:

```
[root@node ~]# pcs resource clear myresource node2.example.com
```



### References

`pcs(8)` man page

For more information, refer to the *Managing cluster resources*, *Displaying resource constraints*, and *Performing cluster maintenance* chapters in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index)

## ► Guided Exercise

# Managing Resource Groups

In this exercise, you will move a resource group between nodes in a cluster.

## Outcomes

You should be able to move a running resource group.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start resources-manage
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines and exports the `/srv/www` directory on the storage machine to the cluster by using NFS. It also configures a resource group named `firstweb` that has three resources, `firstwebip`, `firstwebfs`, and `firstwebserver`, to run a highly available Apache HTTP Server.

## Instructions

- 1. Disable the `firstweb` resource group and then inspect the output of both `pcs status` and `pcs constraint list`.
- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Disable the `firstweb` resource group.

```
[root@nodea ~]# pcs resource disable firstweb  
[root@nodea ~]#
```

- 1.3. Inspect the output of `pcs status`. Notice that the `firstweb` resource group is stopped.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip    (ocf::heartbeat:IPAddr2):     Stopped (disabled)
  * firstwebfs   (ocf::heartbeat:Filesystem): Stopped (disabled)
  * firstwebserver (ocf::heartbeat:apache):   Stopped (disabled)
...output omitted...
```

- 1.4. Observe the output of `pcs constraint list`. No constraint rules are set.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
Ordering Constraints:
Colocation Constraints:
Ticket Constraints:
```

- 2. From nodea, enable the `firstweb` resource group, and 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
[root@nodea ~]#
```

- 2.2. Use the `pcs status` command to identify the node that currently runs the `firstweb` resource group.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip    (ocf::heartbeat:IPAddr2):     Started nodea.private.example.com
  * firstwebfs   (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache):   Started nodea.private.example.com
...output omitted...
```

The preceding output shows that the resource group is running on the `nodea` machine. On your system, the resource group might be running on a different node.

- 2.3. Move the `firstweb` resource group to another node.

```
[root@nodea ~]# pcs resource move firstweb
Warning: Creating location constraint 'cli-ban-firstweb-on-
nodea.private.example.com' with a score of -INFINITY for resource firstweb on
nodea.private.example.com.
This will prevent firstweb from running on nodea.private.example.com until the
constraint is removed
This will be the case even if nodea.private.example.com is the last node in the
cluster
```

- 2.4. Use the `pcs status` command to identify the node that currently runs the `firstweb` resource group.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip      (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com
  * firstwebfs      (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
  * firstwebserver  (ocf::heartbeat:apache):     Started nodeb.private.example.com
...output omitted...
```

Notice that the resource group is running on the nodeb machine now. On your system, the resource group might be running on a different node.

- 2.5. Inspect the resulting constraint rules and notice that the `firstweb` resource group is disabled to run on `nodea`.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
Resource: firstweb
  Disabled on:
    Node: nodea.private.example.com (score:-INFINITY) (role:Started)
Ordering Constraints:
Colocation Constraints:
Ticket Constraints:
```

- 3. Clear all constraints for the `firstweb` resource group and confirm that the constraint is removed.

- 3.1 Clear all constraints for the `firstweb` resource group.

```
[root@nodea ~]# pcs resource clear firstweb
Removing constraint: cli-ban-firstweb-on-nodea.private.example.com
```

- 3.1. Use the `pcs constraint list` command and confirm that the constraint is removed.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
Ordering Constraints:
Colocation Constraints:
Ticket Constraints:
```

- 3.2. Use the `pcs status` command to identify the node that currently runs the `firstweb` resource group.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip      (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com
  * firstwebfs      (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
  * firstwebserver  (ocf::heartbeat:apache):     Started nodeb.private.example.com
...output omitted...
```

Notice that the resource group is still running on the `nodeb` machine. The resource group is not affected by removing the constraints. On your system, the resource group might be running on a different node.

## Finish

On the `workstation` machine, use the `lab` command to conclude this exercise and undo your work. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish resources-manage
```

This concludes the section.

# Configuring Constraints

## Objectives

After completing this section, you should be able to set constraints that control on which cluster nodes resources and resource groups can run.

## Managing constraints

The High Availability Add-On allows three types of constraints:

### Order Constraints

Order constraints control the order in which resources or resource groups are started and stopped.

### Location Constraints

Location constraints control the nodes on which resources or resource groups may run.

### Co-location Constraints

Co-location constraints dictate where resources can be placed in a cluster relative to other resources in the cluster.

Resource groups are the easiest way to set constraints on resources. All resources in a resource group implicitly have co-location 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 must stop in the reverse of the startup order.

At times, it is helpful to configure explicit constraints on resources or resource groups manually. For example, two resource groups need to run on different cluster nodes but start and stop their resources independently. Another example is that the cluster administrator wants a resource group to run on a particular cluster node if 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 might break or other unexpected effects might occur.

## Configuring Order Constraints

Order constraints mandate the order in which services must run. This might 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 cluster resource that accesses the database on startup.

To set an order constraint, use the `pcs constraint order` command:

```
[root@node ~]# pcs constraint order A then B  
Adding A B (kind: Mandatory) (Options: first-action=start then-action=start)
```

The preceding command sets a mandatory order constraint between the two resources or resource groups *A* and *B*. It 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 stops *B* before stopping *A*.
- If both resources are running, and *A* is restarted, then the cluster also restarts *B*.
- If *A* is not running and the cluster cannot start it, *B* stops. It can happen if the first resource is misconfigured or broken, for example.

## Configuring Location Constraints

A location constraint controls the node on which a resource or resource group runs. If no other influences or constraints apply, then the cluster software generally spreads resources evenly around the cluster. A location constraint on a resource causes it to have a preferred node or nodes. The selected node is influenced by the constraint's score. The cluster's chosen node also considers the location preferences of resources that the resource is colocated with. If resource *A* must be colocated with resource *B*, and *B* can only run on *node1*, then it does not matter that *A* prefers *node2*.

## Describing Score and Score Calculations

Score determines the resource's decision to prefer or avoid a node. The possible scores and their effects are as follows:

- **INFINITY**: The resource must run here.
- Positive number or zero: The resource should run here.
- Negative number: The resource should not run here (cluster avoids this node).
- **-INFINITY**: The resource must not run here.

The resource relocates to the node with the highest available score. If two nodes have the same score (for example, if two nodes have a score of **INFINITY**), then the cluster starts the resource on one of those two nodes. Generally, the cluster software prefers 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 high enough score can be found, then the resource remain stopped.

## Viewing Current Scores

The `crm_simulate -sL` command displays the scores (-s) that are currently allocated to resources, resource groups, and STONITH devices on the live cluster (-L). The `crm_simulate` command generates a workload by using allocation methods, formerly known as color, which

are called for a particular resource depending on the resource's type. A regular resource calls only `native_allocate`, while a cloned resource calls both `native_allocate`, for the regular resource that is being cloned, and `clone_allocate`, for its nature as a clone. Similarly, a grouped resource calls both `native_allocate` and `group_allocate`. Because this example focused on node placement by using only the location preference constraint, the resource allocation types are not relevant here.

```
[root@node1 ~]# crm_simulate -sL

Current cluster status:
Online: [ node1.example.com node2.example.com ]

fence_node1 (stonith:fence_ipmilan): Started node1.example.com
fence_node2 (stonith:fence_ipmilan): Started node2.example.com
Resource Group: myweb
    webip (ocf::heartbeat:IPAddr2): Started node1.example.com
    webserver (ocf::heartbeat:apache): Started node1.example.com

Allocation scores:
pcmk_native_allocate: fence_node1 allocation score on node1.example.com: 0
pcmk_native_allocate: fence_node1 allocation score on node2.example.com: 0
pcmk_native_allocate: fence_node2 allocation score on node1.example.com: 0
pcmk_native_allocate: fence_node2 allocation score on node2.example.com: 0
pcmk_group_allocate: myweb allocation score on node1.example.com: 100
pcmk_group_allocate: myweb allocation score on node2.example.com: 0
pcmk_group_allocate: webip allocation score on node1.example.com: 100
pcmk_group_allocate: webip allocation score on node2.example.com: 0
pcmk_group_allocate: webserver allocation score on node1.example.com: 0
pcmk_group_allocate: webserver allocation score on node2.example.com: 0
pcmk_native_allocate: webip allocation score on node1.example.com: 100
pcmk_native_allocate: webip allocation score on node2.example.com: 0
pcmk_native_allocate: webserver allocation score on node1.example.com: 0
pcmk_native_allocate: webserver allocation score on node2.example.com: 0
pcmk_native_allocate: webserver allocation score on node2.example.com: -INFINITY

Transition Summary:
```

In the preceding example, a location constraint was set such that the `myweb` resource group prefers `node1` with a score of 100. The cluster started its `webip` resource there. When that resource was running, the resource group automatically set the score for its `webserver` resource to `-INFINITY` on all other nodes in the cluster so that resource also had to start on `node1`. (All resources in a resource group must run on the same cluster node.)

**Note**

The `crm_simulate` command is also a troubleshooting tool to predict how resources 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

Use the `pcs constraint location` command to set a mandatory location constraint:

```
[root@node ~]# pcs constraint location A prefers node
```

The preceding command sets the resource or resource group *A* to have a score of **INFINITY** for running on *node*. If no other location constraints exist, it forces *A* to always run on *node* if it is up; otherwise, it runs on any other node in the cluster. The change takes effect immediately, relocating the resource if necessary.

If multiple nodes have different scores, then the cluster resource or resource group runs on the node with the highest score. Use location constraints 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@node ~]# pcs constraint location A prefers node=500
```

A cluster resource or resource group avoids a particular node when the score is negative. The following command sets a location constraint with a score of **-INFINITY**, which forces the resource to run on another node if possible. (If no other nodes are available, the resource cannot start.)

```
[root@node ~]# pcs constraint location A avoids node
```

The `pcs resource move resource_id [destination_node]` and `pcs resource ban resource_id [node]` commands set temporary location constraints. Use the `pcs resource clear id` command to delete these constraints; nevertheless, that command does not affect location constraints that are 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, then Pacemaker relocates the resource to that node. It can cause extra unplanned downtime for that resource, especially if it is expensive to relocate. (For example, the resource might take significant time to relocate.)

A default resource stickiness establishes a score for the node on which a resource is currently running. For example, assume that 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 runs on the preferred node. If the preferred node crashes, then the resource moves to one of the other nodes, and that node gets a new score of 1000. When the preferred node comes back, it has a score of only 500, and the resource does not automatically relocate to the preferred node. The cluster administrator must 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@node ~]# pcs resource defaults update resource-stickiness=1000
```

To view current resource defaults:

```
[root@node ~]# pcs resource defaults
```

To clear the resource-stickiness setting:

```
[root@node ~]# pcs resource defaults update 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@node ~]# pcs constraint colocation add B with A
```

The preceding command causes the resources or resource groups *B* and *A* to run on the same node. It implies that Pacemaker determines which node *A* should live on and only then determine whether *B* can also live there. The starting of *A* and *B* occurs in parallel. If no location exists where both *A* and *B* can run, then *A* gets its preference and *B* remains stopped. If no available location exists for *A*, then *B* also has nowhere to run.

A colocation constraint with a score of **-INFINITY** forces the two resources or resource groups to never run on the same node:

```
[root@node ~]# pcs constraint colocation add B with A -INFINITY
```

## Viewing and Removing Constraints

You can always check the current constraints in your cluster and remove them if it is necessary.

### Viewing Constraints

Use the `pcs constraint list` command to view the current constraints that are set for the cluster's resources. Using it with the `--full` option provides more detail including the ID of the constraints.

```
[root@node ~]# pcs constraint list --full
Location Constraints:
  Resource: myweb
    Enabled on:
      Node: node.example.com (score:100) (id:location-myweb-node.example.com-100)
Ordering Constraints:
  Colocation Constraints:
  Ticket Constraints:
```

### Removing Constraints

The `pcs constraint delete id` command can be used to delete a constraint. The *id* can be obtained from the `pcs constraint --full` command.

```
[root@node ~]# pcs constraint delete location-myweb-node.example.com-100
```



## References

pcs(8) and crm\_simulate(8) man pages

For more information, refer to the *Determining which nodes a resource can run on*, *Determining the order in which cluster resources are run*, *Colocating cluster resources*, and *Displaying resource constraints* chapters in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index)

## ► Guided Exercise

# Configuring Constraints

In this exercise, you will set location constraints for a resource group in the cluster.

### Outcomes

You should be able to modify and manage location constraints on resource groups in the cluster.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start resources-constraints
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines and configures two resource groups.

### Instructions

- 1. To prepare for the exercise, ensure that the `nfs` resource group is running on the `nodec.private.example.com` machine and that the `firstweb` resource group is running on the `nodeb.private.example.com` machine.
  - 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Use the `pcs status` command to retrieve the location of the two resource groups.

```
[root@nodea ~]# pcs status  
...output omitted...  
* Resource Group: nfs:  
  * nfsshare      (ocf::heartbeat:Filesystem): Started nodea.private.example.com  
  * nfssd        (ocf::heartbeat:nfsserver): Started nodea.private.example.com  
  * nfsroot      (ocf::heartbeat:exportfs): Started nodea.private.example.com  
  * nfsip        (ocf::heartbeat:IPAddr2): Started nodea.private.example.com  
* Resource Group: firstweb:  
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodeb.private.example.com  
  * firstwebserver (ocf::heartbeat:apache):   Started nodeb.private.example.com  
  * firstwebip    (ocf::heartbeat:IPAddr2):   Started nodeb.private.example.com  
...output omitted...
```

- 1.3. If the `nfs` resource group is not already running on the `nodec.private.example.com` machine, then move it to that node, and then remove the restriction.

```
[root@nodea ~]# pcs resource move nfs nodec.private.example.com  
[root@nodea ~]# pcs resource clear nfs  
Removing constraint: cli-prefer-nfs
```

- 1.4. If the `firstweb` resource group is not already running on the `nodeb.private.example.com` machine, then move it to that node, and then remove the restriction.

```
[root@nodea ~]# pcs resource move firstweb nodeb.private.example.com  
[root@nodea ~]# pcs resource clear firstweb  
Removing constraint: cli-prefer-firstweb
```

- 1.5. Use the `pcs status` command to confirm that the resource groups are now running on the expected machines.

```
[root@nodea ~]# pcs status  
...output omitted...  
* Resource Group: nfs:  
  * nfsshare (ocf::heartbeat:Filesystem): Started nodec.private.example.com  
  * nfsd (ocf::heartbeat:nfsserver): Started nodec.private.example.com  
  * nfsroot (ocf::heartbeat:exportfs): Started nodec.private.example.com  
  * nfsip (ocf::heartbeat:IPAddr2): Started nodec.private.example.com  
* Resource Group: firstweb:  
  * firstwebfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com  
  * firstwebserver (ocf::heartbeat:apache): Started nodeb.private.example.com  
  * firstwebip (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com  
...output omitted...
```

- 2. Configure a location constraint for the `firstweb` resource group to run preferably on the `nodea` machine. Set the score to 200. Confirm that the cluster moves the `firstweb` resource group from the `nodeb` machine to the `nodea` machine.

- 2.1. Add a location constraint score of 200 for the `firstweb` resource group for the `nodea.private.example.com` machine.

```
[root@nodea ~]# pcs constraint location firstweb prefers \  
> nodea.private.example.com=200  
[root@nodea ~]#
```

- 2.2. Confirm that the `firstweb` resource group is now running on the `nodea` machine.

```
[root@nodea ~]# pcs status  
...output omitted...  
* Resource Group: nfs:  
  * nfsshare (ocf::heartbeat:Filesystem): Started nodec.private.example.com  
  * nfsd (ocf::heartbeat:nfsserver): Started nodec.private.example.com  
  * nfsroot (ocf::heartbeat:exportfs): Started nodec.private.example.com  
  * nfsip (ocf::heartbeat:IPAddr2): Started nodec.private.example.com
```

```
* Resource Group: firstweb:
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache):      Started nodea.private.example.com
  * firstwebip     (ocf::heartbeat:IPAddr2):      Started nodea.private.example.com
...output omitted...
```

The cluster migrates the `firstweb` resource group to the `nodea` machine because the score of 200 is higher than the default resource stickiness, which is set to 0 by default.

- 3. Change the `resource-stickiness` parameter to 500, and then add a location score of 499 for the `nodeb` machine for the `firstweb` resource group. Confirm that the `firstweb` resource group does not move.

- 3.1. Change the default `resource-stickiness` to 500.

```
[root@nodea ~]# pcs resource defaults update resource-stickiness=500
Warning: Defaults do not apply to resources which override them with their own
defined values
```

- 3.2. Add a location constraint score of 499 for the `firstweb` resource group for the `nodeb.private.example.com` machine.

```
[root@nodea ~]# pcs constraint location firstweb prefers \
> nodeb.private.example.com=499
[root@nodea ~]#
```

- 3.3. Confirm that the `firstweb` resource group is still running on the `nodea` machine.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: nfs:
  * nfsshare   (ocf::heartbeat:Filesystem): Started nodec.private.example.com
  * nfsd       (ocf::heartbeat:nfsserver):   Started nodec.private.example.com
  * nfsroot    (ocf::heartbeat:exportfs):    Started nodec.private.example.com
  * nfssip     (ocf::heartbeat:IPAddr2):     Started nodec.private.example.com
* Resource Group: firstweb:
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache):      Started nodea.private.example.com
  * firstwebip     (ocf::heartbeat:IPAddr2):      Started nodea.private.example.com
...output omitted...
```

The cluster does not move the `firstweb` resource group to the `nodeb` machine despite its higher score. The cluster operates in this way because the score of the `nodeb` machine is below the `resource-stickiness` score that you configured.

- 4. Set a location constraint for the `nfs` resource group to avoid running on the `nodec` machine with a score of `INFINITY`. Observe the location management of the `nfs` resource group in the cluster.

- 4.1. Set a location constraint for the `nfs` resource group.

```
[root@nodea ~]# pcs constraint location nfs avoids nodec.private.example.com  
[root@nodea ~]#
```

You do not have to specify the score in the preceding command because INFINITY is the default value.

4.2. Observe the location management of the nfs resource group in the cluster.

```
[root@nodea ~]# pcs status  
...output omitted...  
* Resource Group: nfs:  
  * nfsshare (ocf::heartbeat:Filesystem): Started nodea.private.example.com  
  * nfsd (ocf::heartbeat:nfsserver): Started nodea.private.example.com  
  * nfsroot (ocf::heartbeat:exportfs): Started nodea.private.example.com  
  * nfsip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com  
* Resource Group: firstweb:  
  * firstwebfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com  
  * firstwebserver (ocf::heartbeat:apache): Started nodea.private.example.com  
  * firstwebip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com  
...output omitted...
```

In the preceding output, the nfs resource group is running on the nodea machine. On your system, it might be running on the nodeb machine instead.

The cluster migrates the nfs resource group away from the nodec machine because the score prohibits the resource from running on that machine.

## Finish

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish resources-constraints
```

This concludes the section.

## ▶ Lab

# Creating and Configuring Resources

In this lab, you will configure resources and resource groups and set constraints controlling on which nodes a resource group may run.

## Outcomes

You should be able to create multiple resource groups on a cluster.

## Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start resources-review
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines. It creates a resource group named `nfs`, and it also prepares an Ansible Playbook on the `workstation` machine to help you to deploy the Apache HTTP Server on the cluster nodes.

## Instructions

1. Create a resource group named `firstweb` that implements a highly available web server. You can copy and paste the commands to create the resources from the `/root/resources-review/resources-http.txt` file on the `nodea` machine. The resource group consists of three resources:
  - An `IPaddr2` resource named `firstwebip` for the resource group's `172.25.250.80/24` IP address.
  - A `Filesystem` resource named `firstwebfs`, which mounts the NFS file system `storage.san01.example.com:/srv/www` read-only on `/var/www`.
  - An `apache` resource named `firstwebserver`, which manages the service for the Apache HTTP Server.The last resource must install and configure the Apache HTTP Server. To do so, as the student user on the `workstation` machine, run the `/home/student/labs/resources-review/playbook.yml` Ansible Playbook.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

2. Now that two resource groups `firstweb` and `nfs` are running in your cluster, force the resource groups to run on the same node.

## Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade resources-review
```

## Finish

On the **workstation** machine, use the **lab** command to conclude this exercise and undo your work. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish resources-review
```

This concludes the section.

## ► Solution

# Creating and Configuring Resources

In this lab, you will configure resources and resource groups and set constraints controlling on which nodes a resource group may run.

### Outcomes

You should be able to create multiple resource groups on a cluster.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start resources-review
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines. It creates a resource group named `nfs`, and it also prepares an Ansible Playbook on the `workstation` machine to help you to deploy the Apache HTTP Server on the cluster nodes.

### Instructions

1. Create a resource group named `firstweb` that implements a highly available web server. You can copy and paste the commands to create the resources from the `/root/resources-review/resources-http.txt` file on the `nodea` machine. The resource group consists of three resources:
  - An `IPAddr2` resource named `firstwebip` for the resource group's `172.25.250.80/24` IP address.
  - A `Filesystem` resource named `firstwebfs`, which mounts the NFS file system `storage.san01.example.com:/srv/www` read-only on `/var/www`.
  - An `apache` resource named `firstwebserver`, which manages the service for the Apache HTTP Server.

The last resource must install and configure the Apache HTTP Server. To do so, as the student user on the `workstation` machine, run the `/home/student/labs/resources-review/playbook.yml` Ansible Playbook.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

- 1.1. As the `student` user on the `workstation` machine, change to the `/home/student/labs/resources-review/` project directory.

```
[student@workstation ~]$ cd /home/student/labs/resources-review  
[student@workstation resources-review]$
```

- 1.2. Run the Ansible Playbook to prepare the cluster nodes.

```
[student@workstation resources-review]$ ansible-playbook playbook.yml  
...output omitted...
```

- 1.3. Connect to nodea and become the root user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.4. On nodea, as root, create an IPAddr2 resource named firstwebip, in the firstweb resource group, for the IP address and netmask 172.25.250.80/24.

```
[root@nodea ~]# pcs resource create firstwebip IPAddr2 \  
> ip=172.25.250.80 \  
> cidr_netmask=24 \  
> --group=firstweb
```

- 1.5. Create a Filesystem resource that mounts the NFS export workstation.storage1.example.com:/exports/www in read-only mode on the mount point /var/www.

```
[root@nodea ~]# pcs resource create firstwebfs Filesystem \  
> device=storage.san01.example.com:/srv/www \  
> directory=/var/www \  
> fstype=nfs \  
> options=ro \  
> --group=firstweb
```

- 1.6. Create an apache resource named firstwebserver in the firstweb resource group.

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

2. Now that two resource groups firstweb and nfs are running in your cluster, force the resource groups to run on the same node.

- 2.1. Use the pcs status command to determine the node that is currently running the firstweb and the nfs resource groups.

```
[root@nodea ~]# pcs status  
...output omitted...  
* Resource Group: nfs:  
  * nfsshare (ocf::heartbeat:Filesystem): Started nodea.private.example.com  
  * nfssd (ocf::heartbeat:nfsserver): Started nodea.private.example.com  
  * nfsroot (ocf::heartbeat:exportfs): Started nodea.private.example.com  
  * nfsip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com  
* Resource Group: firstweb:
```

## Chapter 4 | Creating and Configuring Resources

```
* firstwebip (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com
* firstwebfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
* firstwebserver (ocf::heartbeat:apache): Started nodeb.private.example.com
...output omitted...
```

Notice that the resource groups are running on different nodes.

- 2.2. Create a colocation constraint to ensure that the `firstweb` resource group runs on the same node as the `nfs` resource group.

```
[root@nodea ~]# pcs constraint colocation add firstweb with nfs
[root@nodea ~]#
```

- 2.3. List the constraint rules. The `firstweb with nfs` colocation constraint that you just created is now present.

```
[root@nodea ~]# pcs constraint list
Location Constraints:
Ordering Constraints:
Colocation Constraints:
  firstweb with nfs (score:INFINITY)
Ticket Constraints:
```

The resource groups are now running on the same node.

- 2.4. Use the `pcs status` command to determine the node that is currently running the `firstweb` and the `nfs` resource groups.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: nfs:
  * nfsshare (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * nfssd (ocf::heartbeat:nfsserver): Started nodea.private.example.com
  * nfsroot (ocf::heartbeat:exportfs): Started nodea.private.example.com
  * nfsip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com
* Resource Group: firstweb:
  * firstwebip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com
  * firstwebfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache): Started nodea.private.example.com
...output omitted...
```

## Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade resources-review
```

## Finish

On the `workstation` machine, use the `lab` command to conclude this exercise and undo your work. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish resources-review
```

This concludes the section.

# Summary

---

In this chapter, you learned how to:

- Distinguish between the three main resource types (LSB, OCF, and `systemd`).
- List available resource agents (`pcs resource list`).
- View resource agent parameters (`pcs resource describe`).
- Show configured resources (`pcs resource show`).
- Configure resource groups.
- Enable and disable resource groups (`pcs resource enable` and `pcs resource disable`).
- Move resource groups between nodes (`pcs resource move`).
- Stop a resource from running on a node (`pcs resource ban`).
- Modify order and location constraints that are automatically set on resource group members.
- Configure the required resources for a highly available NFS service.
- Configure the required resource start order for a clustered service.

## Chapter 5

# Troubleshooting High Availability Clusters

### Goal

Identify, diagnose, and fix cluster problems.

### Objectives

- Inspect and configure log files that the high availability cluster keeps to help diagnose problems.
- Configure the cluster software to send notifications when important events occur.
- Perform common troubleshooting steps to diagnose a failed resource.
- Diagnose problems that relate to cluster network communication.

### Sections

- Configuring Cluster Logging (and Guided Exercise)
- Configuring Cluster Notifications (and Guided Exercise)
- Troubleshooting Resource Failures (and Guided Exercise)
- Troubleshooting Cluster Networking (and Guided Exercise)

### Lab

- Troubleshooting High Availability Clusters

# Configuring Cluster Logging

## Objectives

After completing this section, you should be able to inspect and configure log files that the high availability cluster keeps to help diagnose problems.

## Corosync Logging

When troubleshooting a cluster, one of the most useful resources is the system log. The corosync daemon logs to `/var/log/cluster/corosync.log` and to `syslog`. Consequently, you can also access the log messages with the `journalctl` command and the `/var/log/messages` file. System administrators can control logging by modifying the `logging {}` block in the corosync configuration file located in `/etc/corosync/corosync.conf`.

## Logging Options

### Logging to a File

To enable logging corosync messages to a file, you can add the `to_logfile` and `logfile` directives to the `logging {}` block.

```
logging {  
    to_logfile: yes  
    logfile: /var/log/cluster/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

You can control the logging verbosity and priority at which corosync writes messages by using the `syslog_priority` and `logfile_priority` settings inside the `logging {}` block.

You can set both parameters to any of the following: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, or `debug` (ordered from the highest priority to the lowest priority). When selecting one priority, corosync also includes messages with higher priority in the log file. The default level is `info`.

You can also force debug-level logging for everything by adding the line `debug: on` to the `logging {}` block in `/etc/corosync/corosync.conf`.

### Activating Logging Changes

After making changes to the logging configuration for corosync on a node, you have to distribute the changes to all nodes and restart the cluster. To distribute the changes, you can

use the `pcs cluster sync` command from the node that has the updated configuration file. Then, on each node, run the `pcs cluster reload corosync` command to activate the logging changes. To restart the entire cluster you should use `pcs cluster stop --all` and `pcs cluster start --all`.

## Reviewing Corosync Log Files

You can review the log file to inspect the different messages generated by corosync. This file helps you to verify that you have correctly set up the cluster. More importantly, it also helps you to find any errors.

For example, the corosync log file shows the following information in a three-node cluster configuration when one node is down:

```
[root@node ~]# cat /var/log/cluster/corosync.log
...output omitted...
Mar 29 05:51:57 [31496] node.example.com corosync info      [KNET ] link: host: 1
link: 0 is down ①
Mar 29 05:51:57 [31496] node.example.com corosync info      [KNET ] host: host: 1
(pассив) best link: 0 (pri: 1)
Mar 29 05:51:57 [31496] node.example.com corosync warning [KNET ] host: host: 1
has no active links
Mar 29 05:51:58 [31496] node.example.com corosync notice   [TOTEM ] Token has not
been received in 1237 ms ②
Mar 29 05:51:58 [31496] node.example.com corosync notice   [TOTEM ] A processor
failed, forming new configuration.
Mar 29 05:52:00 [31496] node.example.com corosync notice   [TOTEM ] A new
membership (2.d) was formed. Members left: 1
Mar 29 05:52:00 [31496] node.example.com corosync notice   [TOTEM ] Failed to
receive the leave message. failed: 1
Mar 29 05:52:00 [31496] node.example.com corosync warning [CPG   ] downlist
left_list: 1 received ③
Mar 29 05:52:00 [31496] node.example.com corosync warning [CPG   ] downlist
left_list: 1 received
Mar 29 05:52:00 [31496] node.example.com corosync notice   [QUORUM] Members[2]: 2
3 ④
Mar 29 05:52:00 [31496] node.example.com corosync notice   [MAIN  ] Completed
service synchronization, ready to provide service. ⑤
```

- ① kronosnet reports that node number 1 is down and has no active links.
- ② The totem protocol does not receive information from the node.
- ③ The CPG service sends messages reporting that one node is down.
- ④ The cluster retains quorum with nodes 2 and 3.
- ⑤ The cluster is ready to provide service operating with nodes 2 and 3.

## Pacemaker Logging

By default, pacemaker logs to `/var/log/pacemaker/pacemaker.log` and, only for notice and above messages, to `syslog`. Edit the `/etc/sysconfig/pacemaker` configuration file to change the logging options.

## Logging Options

### Logging to a File

You can change the file that pacemaker uses for logging by using the `PCMK_logfile` option in `/etc/sysconfig/pacemaker`.

### Logging to Syslog

As previously stated, by default pacemaker logs to syslog. However, you can change the specified syslog facility or disable syslog logging by means of the `PCMK_logfacility` option. You can also control the message priority at which pacemaker writes to syslog with the `PCMK_logpriority` setting in the configuration file. The default level is `info`.

System administrators can force debug-level logging for everything by changing the `PCMK_debug=no` parameter to `yes` in `/etc/sysconfig/pacemaker`.

### Activating Logging Changes

After modifying this configuration file, restart the `pacemaker.service` to apply the changes. To restart pacemaker without affecting cluster operations, you can put a node in `standby` mode before restarting the `pacemaker` service. To apply the changes to the cluster, you can also restart the entire cluster using `pcs cluster stop --all` and `pcs cluster start --all`.

## Reviewing Pacemaker Log Files

You can review the pacemaker log file to inspect all the messages generated by the cluster. The `/var/log/pacemaker/pacemaker.log` file includes a lot of cluster-related information that is important for product developers, the Red Hat support team, and advanced cluster administrators. On the other hand, pacemaker only sends important messages to syslog, which you can consult using `journalctl -u pacemaker` or the `/var/log/messages` log file. Thus, the best procedure to review the pacemaker log file is filtering the different results using the `grep` command.

The following is by no means an exhaustive list, but it contains some of the most important functions that you can use for filtering the log file:

#### `crm_update_peer_state_iter`

Shows information about the state of the cluster nodes. Use that pattern with the `grep` command to determine when nodes lose connection to the cluster.

#### `crm_get_peer`

Shows the node ID of each node. Use that pattern to retrieve a node's ID from its name.

#### `pcmk_cpg_membership`

Shows information about cluster membership. Use that pattern to monitor the nodes' subsystems joining or leaving the cluster.

#### `pcmk_quorum_notification`

Shows the quorum status. Use that pattern to inspect when the cluster acquires, retains, or loses quorum.

#### `te_fence_node`

Shows information about fence requests. Use that pattern to determine when nodes get fenced.

#### `tengine_stonith_notify`

Shows information about the status of fencing operations. Use that pattern to determine if the fencing agents are successful.

**determine\_online\_status\_fencing**

Shows the status of the fence devices. Provides similar information as the `pcs stonith status` command.

**Note**

You can also use `journalctl` to view and query the logs for pacemaker and corosync:

```
[root@node ~]# journalctl -l -u pacemaker.service -u corosync.service
```

**References**

`corosync.conf(5)` man page

`/etc/corosync/corosync.conf.example` configuration file

`/etc/sysconfig/pacemaker` configuration file

**Knowledgebase: "How do I configure logging for corosync in Red Hat Enterprise Linux (RHEL) 8 High Availability clusters?"**

<https://access.redhat.com/solutions/5017511>

## ► Guided Exercise

# Configuring Cluster Logging

In this exercise, you will examine the pacemaker log file to identify various cluster events, and configure logging to record additional debug-level messages.

### Outcomes

You should be able to identify key events in the pacemaker log and configure it to record debugging messages.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start troubleshooting-logs
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines and configures a highly available Apache HTTP Server.

### Instructions

- 1. On nodea, identify the node currently running the `firstweb` resource group. Use the `/root/firewall-script/firewall-block.sh` script to cause that node to fail. To generate an event that will be recorded by the cluster logs, cause the node that is running the `firstweb` resource group to fail. Then, determine which node is the current DC (Designated Controller).

- 1.1. Connect to nodea and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. Use the `pcs status` command to identify the node currently running the `firstweb` resource group.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip    (ocf::heartbeat:IPAddr2): Started nodea.private.example.com
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache):     Started nodea.private.example.com
...output omitted...
```

The preceding output shows that the resource group is running on the nodea node. On your system, the group might be running on a different node.

13. Cause the cluster to fence the node running the `firstweb` resource group by disrupting network communication on that node. Use the `/root/firewall-script/firewall-block.sh` script that has been provided for you by the exercise preparation command.

```
[root@nodea ~]# /root/firewall-script/firewall-block.sh
success
success
```

14. On another node, use the `pcs status` command to monitor the fencing and the resource relocation process and identify the current DC node.

```
[root@nodeb ~]# watch pcs status
...output omitted...
Cluster Summary:
  * Stack: corosync
  * Current DC: nodeb.private.example.com (version 2.0.4-6.el8-2decea3ae) -
partition with quorum
...output omitted...
  * Resource Group: firstweb:
    * firstwebip      (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com
    * firstwebfs     (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
    * firstwebserver (ocf::heartbeat:apache):   Started nodeb.private.example.com
...output omitted...
```

The preceding output shows that the current DC is the `nodeb` node. It might be a different node on your system.

- ▶ 2. Use the logs on the current DC to determine when `nodea` lost communication with the cluster. Determine if the cluster has fenced the node.
- 2.1. On the current DC, use `grep` on the `/var/log/pacemaker/pacemaker.log` log file to find any entries containing the string `crm_update_peer_state_iter`. This string shows when a cluster member changes its state. Use those entries to determine when contact with the node that had been running the `firstweb` resource group was lost.

```
[root@nodeb ~]# grep crm_update_peer_state_iter /var/log/pacemaker/pacemaker.log
...output omitted...
Jan 25 20:30:56 nodeb.lab.example.com pacemaker-based [151995]
(crm_update_peer_state_iter) notice: Node nodea.private.example.com state is
now lost | nodeid=1 previous=member source=crm_update_peer_proc
Jan 25 20:30:56 nodeb.lab.example.com pacemaker-attrd [151998]
(crm_update_peer_state_iter) notice: Node nodea.private.example.com state is
now lost | nodeid=1 previous=member source=crm_update_peer_proc
Jan 25 20:30:56 nodeb.lab.example.com pacemaker-fenced [151996]
(crm_update_peer_state_iter) notice: Node nodea.private.example.com state is
now lost | nodeid=1 previous=member source=crm_update_peer_proc
Jan 25 20:30:56 nodeb.lab.example.com pacemaker-controld [152000]
(crm_update_peer_state_iter) notice: Node nodea.private.example.com state is
now lost | nodeid=1 previous=member source=crm_reap_unseen_nodes
...output omitted...
```

- 2.2. On the current DC, use grep on the /var/log/pacemaker/pacemaker.log log file to find any entries containing the string te\_fence\_node. Use those entries to identify any attempts by the cluster to fence nodes.

```
[root@nodeb ~]# grep te_fence_node /var/log/pacemaker/pacemaker.log
Jan 25 20:30:57 nodeb.lab.example.com pacemaker-controld [152000] (te_fence_node)
  notice: Requesting fencing (reboot) of node nodea.private.example.com |
action=1 timeout=180000
```

- 2.3. On the current DC, use grep on the /var/log/pacemaker/pacemaker.log log file to find any entries containing the string tengine\_stonith\_notify. Use those entries to determine if the node that the firstweb resource group had been running on was successfully fenced.

```
[root@nodeb ~]# grep tengine_stonith_notify /var/log/pacemaker/pacemaker.log
Jan 25 20:32:11 nodeb.lab.example.com pacemaker-controld [152000]
(tengine_stonith_notify)  notice: Peer nodea.private.example.com was terminated
(reboot) by nodec.private.example.com on behalf of pacemaker-controld.152000: OK
| initiator=nodeb.private.example.com ref=d0b23257-62bb-48d5-ae82-d9528b933f70
```

- ▶ 3. Change the pacemaker configuration to log additional debugging messages. Review the Ansible Playbook named playbook.yml in the /home/student/labs/troubleshooting-logs directory that modifies the /etc/sysconfig/pacemaker file in the cluster nodes, and then run that playbook.
- 3.1. Look at the content of the playbook.yml playbook. The playbook ensures that the /etc/sysconfig/pacemaker file has the line PCMK\_debug=yes present on all cluster nodes and restarts the cluster. You do not have to modify anything in that file.

```
[student@workstation ~]$ cd /home/student/labs/troubleshooting-logs
[student@workstation troubleshooting-logs]$ cat playbook.yml
---
- name: Ensure pacemaker configuration to log at DEBUG level
  hosts: nodes
  become: yes
  gather_facts: no

  tasks:
    - name: Ensuring pacemaker configuration is configured
      lineinfile:
        path: /etc/sysconfig/pacemaker
        regex: "^\# PCMK_debug=no"
        line: "PCMK_debug=yes"
        state: present
        backup: yes

    - name: Ensuring pacemaker configuration is loaded
      command:
        cmd: "{{ item }}"
      with_items:
        - pcs cluster stop --all
```

```
- pcs cluster start --all  
run_once: yes  
...
```

- 3.2. Run the Ansible Playbook to make sure that the /etc/sysconfig/pacemaker file is properly modified on your cluster nodes.

```
[student@workstation troubleshooting-logs]$ ansible-playbook playbook.yml
```

- ▶ 4. Validate the new logging configuration by opening the /var/log/pacemaker/pacemaker.log log file in follow mode. Notice the new debug: entries.

```
[root@nodea ~]# tail -f /var/log/pacemaker/pacemaker.log  
...output omitted...  
Jan 25 21:06:00 nodea.lab.example.com pacemaker-controld [3343]  
(cluster_connect_quorum)  debug: Configuring Pacemaker to obtain quorum from  
Corosync  
...output omitted...
```

## Finish

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish troubleshooting-logs
```

This concludes the section.

# Configuring Cluster Notifications

## Objectives

After completing this section, you should be able to configure the cluster software to send notifications when important events occur.

## Cluster Monitoring with Pacemaker

Every event in the cluster can be monitored. An event can be a node starting, a resource failing, or a change in the cluster configuration. Monitoring the cluster is important because it provides information about the cluster status. Thus, system administrators should monitor the cluster so that they receive notifications for some events. For example, you can configure the cluster to send notifications when a node fails. This way, you can be aware of any problem in the cluster and respond accordingly.

For detailed monitoring, the Red Hat Enterprise Linux High Availability Add-On ships with the `pcs alert` utility. This utility uses external programs, named *alert agents*, which the cluster calls for sending alerts.

When calling alert agents, the cluster uses environment variables to provide information to them.



### Note

For more information regarding the environment variables that you can use for your alert agents, see the *Environment Variables Passed to Alert Agents* table in [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#tb-alert-environmentvariables-HAAR](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#tb-alert-environmentvariables-HAAR)

Script templates for alert agents are available in the `/usr/share/pacemaker/alerts` directory. System administrators can directly use these templates or modify them as needed. You can also create custom alert agents.

## Declaring Cluster Alerts

Use the `pcs alert create` command to create a cluster alert:

```
[root@node ~]# pcs alert create path=path [id=alert-id] \
> [options option=value ...]
```

The `path` parameter provides the path to the alert agent script. The options are agent-specific configuration values that are passed to the alert agent script as environment variables. If you do not specify a value for `id`, then `pcs` uses `alert-X`, where `X` is a number.

For example, to create a sample cluster alert named `myalert` that runs a script in the `/usr/share/pacemaker/alerts/sample_alert.sh` file, use the following command:

```
[root@node ~]# pcs alert create \
> path=/usr/share/pacemaker/alerts/sample_alert.sh id=myalert
```

**Note**

When using an alert agent, ensure that you installed it on each node in the cluster. In the preceding example, every node in the cluster must have the `sample_alert.sh` script in the same location (`/usr/share/pacemaker/alerts/`).

You can use the `pcs alert update` command to modify the options of a cluster alert.

```
[root@node ~]# pcs alert update alert-id [options [option=value] ...]
```

## Displaying Cluster Alerts

You can display the active cluster alerts by using the `pcs alert show` or `pcs alert config` commands. Both commands show the same information.

```
[root@node ~]# pcs alert show
Alerts:
Alert: myalert (path=/usr/share/pacemaker/alerts/sample_alert.sh)
```

## Configuring Cluster Alerts

When setting up a cluster alert, you must specify the recipients for that cluster alert. The recipients you specify depend on the type of alert agent. For example, an email agent can send alerts to one or more email addresses. An agent that writes events to a file can write to one or more files as recipients.

To add a recipient to your cluster alert, use the `pcs alert recipient add` command:

```
[root@node ~]# pcs alert recipient add alert-id \
> value=recipient-value [id=recipient-id] [options [option=value] ...]
```

For example, if the previous `sample_alert.sh` script sends an email, then configure the recipient to be the email address. To add an email as a recipient for the previously created `myalert` cluster alert, you can use the following command:

```
[root@node ~]# pcs alert recipient add myalert value=email_one@example.com
[root@node ~]# pcs alert show
Alerts:
Alert: myalert (path=/usr/share/pacemaker/alerts/sample_alert.sh)
Recipients:
Recipient: myalert-recipient (value=email_one@example.com)
```

To add another email to the same cluster alert, create another recipient for the same agent:

```
[root@node ~]# pcs alert recipient add myalert value=email_two@example.com
[root@node ~]# pcs alert show
Alerts:
  Alert: myalert (path=/usr/share/pacemaker/alerts/sample_alert.sh)
    Recipients:
      Recipient: myalert-recipient (value=email_one@example.com)
      Recipient: myalert-recipient-1 (value=email_two@example.com)
```

You can modify the recipient for a cluster alert by using the `pcs alert recipient update` command. To remove a recipient, use either the `pcs alert recipient delete` or `pcs alert recipient remove` command. Both commands have the same behavior.

## Removing Cluster Alerts

When it is necessary to remove a cluster alert, use either the `pcs alert remove` or `pcs alert delete` commands. Both commands have the same behavior.

For example, to remove the `myalert` cluster alert, run the following command:

```
[root@node ~]# pcs alert remove myalert
```

## Configuring MailTo Notifications

System administrators typically find notifications to be useful, because they indicate some type of cluster change. For example, a notification when a resource migrates to a different node might indicate trouble with a node. Adding a `MailTo` resource to a resource group is a simple way to receive such notifications. The `MailTo` resource sends an email to the recipient address whenever the resource group starts or stops.



### Note

Verify that all alert agent dependencies are present and working on every cluster node. For example, an email alert agent requires a mail transport agent, such as `postfix` or `sendmail`, as described later in this section.

In this example, you add a `MailTo` resource to the `importantgroup` resource group. The resource sends emails to `admin@example.com` with a subject prefix of `importantgroup notification` followed by the action, the timestamp, and the node name.

```
[root@node ~]# pcs resource create importantgroup-mailto MailTo \
> email=admin@example.com \
> subject="importantgroup notification" \
> --group=importantgroup
```

## Mail-sending Configuration on Cluster Nodes

The `MailTo` resource agent and other resource agents that use mail alerts require a mail transport agent (MTA) and a mail-sending client to be installed and configured on every cluster node. This example uses `postfix` because it is simple to configure, and `mailx` because only command-line mail sending is required without other user interface features. The `postfix` service must be enabled to restart after a reboot. No additional configuration is required for this send-only requirement.

```
[root@node ~]# yum install postfix mailx  
[root@node ~]# systemctl enable --now postfix
```

## Mail-receiving Configuration on the User's Workstation

Typically, your user computer is already configured for receiving mail. If not already configured in this class environment, then your `workstation` system might need mail components to be installed.

To configure to receive SMTP mail on a RHEL system, install and configure the `postfix` MTA. Open the SMTP firewall ports by using the existing `firewalld` `smtp` service configuration. Configure `postfix` to listen for incoming mail on all network interfaces. Start and enable the `postfix` service to ensure that the service restarts after a reboot.

```
[root@host ~]# yum install postfix  
[root@host ~]# firewall-cmd --permanent --add-service=smtp  
[root@host ~]# firewall-cmd --reload  
[root@host ~]# postconf -e 'inet_interfaces = all'  
[root@host ~]# systemctl enable --now postfix
```

To configure a practical mail reader, install `mutt`. Mutt prompts to create a mail folder the first time you run it. To create the folder in advance, use `mkdir` as the mail user, not as root.

```
[user@host ~]$ sudo yum install mutt  
[user@host ~]$ mkdir -m 0700 ~/Mail
```

You are now ready to receive mail alerts from your cluster nodes on your host system.



### Important

To ensure that alerts are received, test MailTo resources before deploying your cluster in a production environment. You can test the MailTo alerts in a resource group by, for example, moving resources on that resource group by using the `pcs resource move` command.



### References

`mailx(1)`, `mutt(1)`, `pcs(8)`, and `postfix(1)` man pages

For more information, refer to the *Triggering scripts for cluster events* chapter in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-pacemaker-alert-agents\\_configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-pacemaker-alert-agents_configuring-and-managing-high-availability-clusters)

**Knowledgebase: "Can we configure pacemaker cluster to send Email notification when a resource goes in stopped state ?"**

<https://access.redhat.com/solutions/4210311>

## ► Guided Exercise

# Configuring Cluster Notifications

In this exercise, you will configure email notifications for cluster events.

## Outcomes

You should be able to configure notifications for cluster events.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start troubleshooting-notification
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines, and configures a highly available Apache HTTP Server.

## Instructions

- 1. Ensure that the `mutt` and `postfix` packages are installed and configured on `workstation` to receive mail from the cluster nodes. Verify that the `inet_interfaces = all` parameter is set using the `postconf` command. Verify that the firewall accepts `smtp` communication. Ensure that the `postfix` service is enabled and started. For your convenience, the `lab start` command places the `prepare-workstation.yml` playbook in the `/home/student/labs/troubleshooting-notification` directory.
- 1.1. Display the content of the `prepare-workstation.yml` playbook. No modification is necessary.

```
[student@workstation ~]$ cd /home/student/labs/troubleshooting-notification/  
[student@workstation troubleshooting-notification]$ cat prepare-workstation.yml  
---  
- name: Ensure the postfix and mutt are installed  
  hosts: localhost ①  
  connection: local ②  
  become: yes  
  gather_facts: no  
  
  tasks:  
    - name: Ensure packages in workstation ③  
      yum:  
        state: present  
        name:  
          - postfix  
          - mutt  
  
    - name: Ensure Mail directory for student user ④  
      file:
```

```
path: /home/student/Mail
state: directory
owner: student
group: student
mode: '0700'

- name: Ensure right parameters for postfix 5
  command: postconf -e 'inet_interfaces = all'

- name: Ensuring the required ports are open 6
  firewalld:
    service: smtp
    permanent: yes
    state: enabled
    immediate: yes

- name: Ensuring the postfix service is started and enabled 7
  service:
    name: postfix
    state: started
    enabled: yes
...
...
```

- 1** The play targets the localhost.
- 2** Runs the entire playbook locally.
- 3** The first task installs the *postfix* and *mutt* packages.
- 4** The second task creates */home/student/Mail* directory
- 5** The third task ensures the *inet\_interfaces = all* parameter.
- 6** The fourth task prepares the firewall.
- 7** The last task enables and starts the *postfix* service.

- 1.2. Run the *prepare-workstation.yml* Ansible playbook, defining the variable *ansible\_become\_password=student* by using *--extra-vars* Ansible argument.

```
[student@workstation troubleshooting-notification]$ ansible-playbook --extra-vars
  ansible_become_password=student prepare-workstation.yml
...output omitted...
```

- ▶ **2.** Ensure that the *mailx* and *postfix* packages are installed in the cluster nodes to send mail. Ensure that the *postfix* service is enabled and started. For your convenience, the *lab start* command places the *prepare-nodes.yml* Ansible playbook in the */home/student/labs/troubleshooting-notification* directory.
- 2.1. Display the content of the *prepare-nodes.yml* playbook. No modification is necessary.

```
[student@workstation ~]$ cd /home/student/labs/troubleshooting-notification/
[student@workstation troubleshooting-notification]$ cat prepare-nodes.yml
---
- name: Ensure the postfix and mailx are installed
  hosts: nodes ①
  become: yes
  gather_facts: no

  tasks:
    - name: Installing packages in nodes ②
      yum:
        state: present
        name:
          - postfix
          - mailx

    - name: Ensuring the postfix service is started and enabled ③
      service:
        name: postfix
        state: started
        enabled: yes
...

```

- ① The play targets the `nodes` group, which groups the three cluster nodes. Therefore, the following tasks run on all three nodes.
- ② The first task installs the `postfix` and `mailx` packages.
- ③ The last task enables and starts the `postfix` service.

## 2.2. Run the `prepare-nodes.yml` Ansible playbook.

```
[student@workstation troubleshooting-notification]$ ansible-playbook
  prepare-nodes.yml
  ...output omitted...
```

- ▶ 3. Add a MailTo resource named `webmail` to the `firstweb` resource group. It should automatically send an email with the subject `CLUSTER-NOTIFICATION` to `student@workstation.lab.example.com` when it is triggered. Use `mutt` on your `workstation` machine to read these messages.

### 3.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 3.2. From `nodea`, add a MailTo resource named `webmail` to the `firstweb` group with a subject of `CLUSTER-NOTIFICATION` and a recipient of `student@workstation.lab.example.com`. Updating this resource group should

trigger some messages. You can copy and paste the following command from the /root/troubleshooting-notification/resource.txt file.

```
[root@nodea ~]# pcs resource create webmail MailTo \
> email=student@workstation.lab.example.com \
> subject="CLUSTER-NOTIFICATION" \
> --group=firstweb
```

- 3.3. On your workstation machine, open **mutt** as the student user, and view the generated message. The message has a subject that begins with CLUSTER-NOTIFICATION, and its contents alert you that the cluster has relocated, stopped, or started the resource group.

```
[student@workstation ~]$ mutt
```

Press q several times to exit **mutt**.

- 4. Install an alert agent and then use it to configure an alert to send cluster events as email messages. Review the Ansible Playbook in the /home/student/labs/troubleshooting-notification directory, which deploys the agent script to the cluster nodes. Run the playbook.

Check the mail for student on your workstation machine to see if the resource works. Use the values in the following table:

<b>Setting</b>	<b>Value</b>
Alert name	mailme
email	student@workstation.lab.example.com
Sender email	donotreply@example.com
Sample alert agent path	/usr/share/pacemaker/alerts/alert_smtp.sh.sample
Alert agent path	/var/lib/pacemaker/alert_smtp.sh

- 4.1. As the student user on the workstation machine, change to the /home/student/labs/troubleshooting-notification/ project directory.

```
[student@workstation ~]$ cd /home/student/labs/troubleshooting-notification
[student@workstation troubleshooting-notification]$
```

- 4.2. Review the **playbook.yml** playbook. This playbook ensures that the /var/lib/pacemaker/alert\_smtp.sh agent script has been installed on all cluster nodes. You do not have to modify anything in that file.

```
[student@workstation troubleshooting-notification]$ cat playbook.yml
---
- name: Ensure the agent script is installed
  hosts: nodes
  become: yes
  gather_facts: no
```

```
tasks:  
- name: Copying agent script file to the nodes  
copy:  
  src: /usr/share/pacemaker/alerts/alert_smtp.sh.sample  
  dest: /var/lib/pacemaker/alert_smtp.sh  
  owner: hacluster  
  group: haclient  
  mode: 0755  
  remote_src: yes  
...
```

- 4.3. Run the Ansible Playbook to make sure that the `/var/lib/pacemaker/alert_smtp.sh` alert script is properly installed on your cluster nodes.

```
[student@workstation troubleshooting-notification]$ ansible-playbook playbook.yml
```

- 4.4. Create an alert agent in your cluster, using the settings in the table above.

```
[root@nodea ~]# pcs alert create \  
> id=mailme \  
> path=/var/lib/pacemaker/alert_smtp.sh \  
> options email_sender=donotreply@example.com
```

- 4.5. Add an email recipient to your alert.

```
[root@nodea ~]# pcs alert \  
> recipient add mailme \  
> value=student@workstation.lab.example.com
```

- 4.6. Verify that the `mailme` alert agent has been created.

```
[root@nodea ~]# pcs alert  
Alerts:  
  Alert: mailme (path=/var/lib/pacemaker/alert_smtp.sh)  
    Options: email_sender=donotreply@example.com  
  Recipients:  
    Recipient: mailme-recipient (value=student@workstation.lab.example.com)
```

- 5. Generate some cluster notifications by manually moving the `firstweb` resource group, and by interrupting network communications to `nodec` so that it is fenced by the cluster. View the messages created by both the `MailTo` resource and the alert agent.

- 5.1. Manually move the `firstweb` resource group.

```
[root@nodea ~]# pcs resource move firstweb
```

- 5.2. On `nodec`, interrupt cluster communications to force the cluster to fence the node.

```
[root@nodec ~]# /root/firewall-script/firewall-block.sh
```

- 5.3. On nodea, use the `pcs status` command to confirm that the nodec machine has rejoined the cluster. You might have to wait a few minutes for the nodec machine to restart.

```
[root@nodea ~]# pcs status
Cluster name: cluster1
...output omitted...
Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]
...output omitted...
```

- 5.4. As student on workstation, read the resulting messages. Look at the email messages with a subject that starts with a time stamp and include text such as `cluster1: Resource operation`. Those emails alert you about changes to the cluster and cluster resources.

```
[student@workstation ~]$ mutt
```

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish troubleshooting-notification
```

This concludes the section.

# Troubleshooting Resource Failures

## Objectives

After completing this section, you should be able to perform common troubleshooting steps to diagnose a failed resource.

## Resource Failures

Resources can fail for multiple reasons. Incorrect settings for a resource, errors in configuration files, trying to start a resource that does not exist, and other unforeseen errors can all cause failures.

Whenever a resource fails, the cluster increments the value of `failcount` for the resource. You can use the `pcs resource failcount show <RESOURCE>` command to view this count.

Failure to start a resource sets its fail count to `INFINITY`, forcing the resource to move to a different node. Failure to stop a resource also sets its fail count to `INFINITY`, and fences the node that failed so the cluster can start the resource on a different node.

You can configure resources to relocate to a different node after  $N$  number of failures by setting the `meta migration-threshold=N` option when creating or modifying the resource.

For example, to create an Apache resource that relocates to a different node after five failures, run the following command:

```
[root@node ~]# pcs resource create my_resource ocf:heartbeat:apache \
> meta migration-threshold=5
```

By default, resources do not migrate unless their fail count reaches `INFINITY`.

## Troubleshooting Resource Failures

Steps to take when troubleshooting a resource failure include:

- Inspect the log files for the affected resources.
- Inspect the cluster log files.
- Verify the resource configuration.
- Verify the configuration files.
- Attempt to manually start the resource with `debug-start`.

## Inspect Log Files

You should first inspect the cluster log files and any log files that the affected resource might generate. Be careful when reading these log files, because the actual error might be a single small warning, whereas the resulting failure cascade following the first error might generate far more logging.

## Verify Resource Configuration

Using the `pcs resource config` or `pcs resource config <RESOURCE>` commands to inspect the cluster configuration for the failed resource. Small mistakes here, or missed options, might have a drastic effect.

## Verify Configuration Files

If the affected resource has its own configuration file, such as an apache resource, then it might also have its own configuration file validation tool, such as `apachectl configtest`. Ensuring that the resource can start eliminates most of the potential failures.

## Manually Starting a Resource with Debug-start

When a resource reaches a fail count of `INFINITY` on all nodes, it is no longer possible to start that resource automatically. You can still attempt to start the resource with the `pcs resource debug-start <RESOURCE>` command. This results in a short status output on both failure and success. You can include the `--full` option to generate full debugging output, which can assist troubleshooting.



### Important

In some cases, when a resource agent fails to start, the `pcs resource debug-start` command might not complete successfully. In these cases, you can use `Ctrl+C` to exit the `debug-start` command.

## Cluster Maintenance

Several tools are available to perform maintenance on your cluster, such as the `pcs` command. Those tools are helpful when you need to perform troubleshooting operations or update the software on the different nodes. You have different options available to move resources from one node to another, start or stop services, or put nodes into standby mode.

### Putting Nodes in Standby Mode

Occasionally, you might need a node to move any running resources to other nodes in the cluster, and prevent it from running new resources. You can do this by putting the node in standby mode, using the `pcs node standby node` command. This command is useful when updating packages on the different nodes. Thus, you can put a node on standby, update its packages, and then remove the standby mode using the `pcs node unstandby node` command.

### Putting a Cluster in Maintenance Mode

When the entire cluster changes in maintenance mode, the cluster resources continue running, and the cluster will not attempt to start, stop or recover services because the resources are unmanaged. The `pcs property set maintenance-mode=true` command puts your cluster in maintenance mode.

To remove your cluster from maintenance mode, run the `pcs property set maintenance-mode=false` command. After you remove the maintenance mode, the cluster performs some basic tests to ensure that the resources can run correctly.

## Fixing Resource Failures

Updating a cluster resource definition automatically resets the fail count for that resource, enabling its use on the cluster again.

When you perform the fix outside of the cluster configuration, for example by updating a service configuration file, the fail count remains, preventing the resource from being started. In those cases, you can run the `pcs resource cleanup <RESOURCE>` or `pcs resource refresh <RESOURCE>` command to manually reset the fail count, enabling the resource. The difference between `cleanup` and `refresh` is that the former makes the cluster clean failed operations, on the other hand, the latter removes the complete operation history. You can also reset the fail count for all the resources at the same time by adding the `--full` option at the end.

For example, to reset the fail count for the resource `my_resource` in a three-node cluster configuration, use the following command:

```
[root@node1 ~]# pcs resource failcount show my_resource
Failcounts for resource 'my_resource'
  node1.example.com: INFINITY
[root@node1 ~]# pcs resource cleanup my_resource
Cleaned up my_resource on node1.example.com
Cleaned up my_resource on node2.example.com
Cleaned up my_resource on node3.example.com
Waiting for 3 replies from the controller..... OK
[root@node1 ~]# pcs resource failcount show my_resource
No failcounts for resource 'my_resource'
```



### References

`pcs(8)` man page

For more information, refer to the *Managing cluster resources* chapter in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_managing-cluster-resources-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_managing-cluster-resources-configuring-and-managing-high-availability-clusters)

For more information, refer to the *Performing cluster maintenance* chapter in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_cluster-maintenance-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_cluster-maintenance-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Troubleshooting Resource Failures

In this exercise, you will troubleshoot a failed resource.

## Outcomes

You should be able to troubleshoot a resource startup failure.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start troubleshooting-resource
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines and configures a highly available Apache HTTP Server with a failure.

## Instructions

- 1. On your nodea machine, use `pcs status` command to identify which resource failed. Notice that the cluster stabilizes when failed resource stops migrating from node to node.
- 1.1. Connect to nodea and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Inspect the output of `pcs status` to identify which resource failed.

```
[root@nodea ~]# pcs status  
...output omitted...  
* Resource Group: firstweb:  
  * firstwebip (ocf::heartbeat:IPAddr2): Started nodec.private.example.com  
  * firstwebfs (ocf::heartbeat:Filesystem): Started nodec.private.example.com  
  * firstwebserver (ocf::heartbeat:apache): Stopped  
  
Failed Resource Actions:  
  * firstwebserver_start_0 on nodec.private.example.com 'not installed' (5):  
    call=32, status='complete', exitreason='environment is invalid, resource  
    considered stopped', last-rc-change='2021-02-01 17:03:41 -05:00', queued=1ms,  
    exec=32ms
```

```
* firstwebserver_start_0 on nodeb.private.example.com 'not installed' (5):
call=32, status='complete', exitreason='environment is invalid, resource
considered stopped', last-rc-change='2021-02-01 17:03:41 -05:00', queued=0ms,
exec=29ms
* firstwebserver_start_0 on nodea.private.example.com 'not installed' (5):
call=32, status='complete', exitreason='environment is invalid, resource
considered stopped', last-rc-change='2021-02-01 17:03:40 -05:00', queued=0ms,
exec=28ms
...output omitted...
```

- ▶ 2. Review the fail count for the `firstwebserver` resource, and 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.

#### 2.1. View the fail count for the `firstwebserver` resource.

```
[root@nodec ~]# pcs resource failcount show firstwebserver
Failcounts for resource 'firstwebserver'
nodea.private.example.com: INFINITY
nodeb.private.example.com: INFINITY
nodec.private.example.com: INFINITY
```

- 2.2. Perform a forced `debug-start` of the `firstwebserver` resource, with full verbosity. Look for any messages that might help troubleshoot the problem. Use the node that is currently hosting the other resources in the `firstweb` resource group.

```
[root@nodec ~]# pcs resource debug-start firstwebserver --full
...output omitted...
> stderr: + 18:20:15: GetParams:136: ConfigFile=/etc/httpd/conf/httpd.conf
> stderr: + 18:20:15: GetParams:137: '[' '!' -f /etc/httpd/conf/httpd.conf ']'
> stderr: + 18:20:15: GetParams:138: return 5
...output omitted...
> stderr: + 18:20:15: __ha_log:248: echo 'apache(firstwebserver)[157294]: Feb'
02 18:20:15 'ERROR: Configuration file /etc/httpd/conf/httpd.conf not found!'
> stderr: + 18:20:15: apache_validate_all:618: return 5
...output omitted...
> stderr: ocf-exit-reason:environment is invalid, resource considered stopped
...output omitted...
> stderr: + 18:20:15: __ha_log:248: echo 'apache(firstwebserver)[157294]: Feb'
02 18:20:15 'ERROR: environment is invalid, resource considered stopped'
> stderr: + 18:20:15: handle_invalid_env:108: exit 5
```

In the preceding output, the resource group is running on the `nodec` machine. On your system, the resource group might be running on a different node. Those errors seem to indicate that the resource cannot find the specified configuration file.

- 2.3. View the resource configuration for the `firstwebserver` resource. Notice the `configfile` attribute; small mistakes might have a drastic effect. In this case, there is an extra "t" in the path.

```
[root@nodec ~]# pcs resource config firstwebserver
Resource: firstwebserver (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf
...output omitted...
```

- 3. Fix the configuration problem and then review the fail count for the resource group.
- 3.1. Fix the configuration problem by removing the incorrect `configfile` attribute from the `firstwebserver` resource. By removing the `configfile` option value, you instruct the resource to use the default value, which is `/etc/httpd/conf/httpd.conf`.

```
[root@nodec ~]# pcs resource update firstwebserver configfile=
[root@nodec ~]#
```

- 3.2. Review the fail count for the `firstwebserver` resource.

```
[root@nodec ~]# pcs resource failcount show firstwebserver
No failcounts for resource 'firstwebserver'
```

Updating the resource has automatically reset the fail count.

- 3.3. Verify that the `firstweb` resource group has properly started and is serving content.

```
[root@nodec ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebip (ocf::heartbeat:IPAddr2): Started nodec.private.example.com
  * firstwebfs (ocf::heartbeat:Filesystem): Started nodec.private.example.com
  * firstwebserver (ocf::heartbeat:apache): Started nodec.private.example.com
...output omitted...
[root@nodec ~]# curl 172.25.250.80
Hello, world!
```

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish troubleshooting-resource
```

This concludes the section.

# Troubleshooting Cluster Networking

## Objectives

After completing this section, you should be able to diagnose problems that relate to cluster network communication.

## Identifying Network Issues

Faulty or misconfigured network connections can wreak havoc on a cluster. The following is by no means an exhaustive list, but it contains some of the possible issues and fixes.

## Identifying Firewall Issues

Incorrectly configured firewalls can make a machine unreachable by the other nodes. Ensure that all nodes can reach the high availability services. You should also open the network ports on the public network for any offered cluster services.

## Identifying Split Networks

If you plug multiple cluster nodes into different switches, and the connection between those switches drops, then the cluster goes into a split-brain mode, losing a number of nodes. You can reduce the risk of these types of failures by using redundant networking and interconnects.

## Identifying Dropped Packets

When a network link becomes saturated, it might drop some packets, resulting in intermittent cluster failures. You can avoid these scenarios by using separate networks for private cluster communications, public client access, and storage networks.

## Identifying Latency on Cluster Network

The time that it takes to transfer information from one node to another, called latency, can also lead to cluster failures if it is too high. Red Hat recommends using networks that provide round-trip latencies lower than 2 ms. You should avoid networks that provide high-latency links, with round-trip latencies higher than 300 ms, because they produce instability in the cluster.

You can test the round-trip latency between two nodes in the cluster using the `ping` command. For example, to obtain the round-trip latency between `node1` and `node2`, run the following command:

```
[user@node1 ~]$ ping -c10 -q node2.example.com
PING node2.example.com (172.10.10.11) 56(84) bytes of data.

--- node2.example.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 241ms
rtt min/avg/max/mdev = 0.201/0.497/1.207/0.349 ms
```

This example shows that when transmitting ten packets between the two nodes, the round-trip latency is around 0.5 ms on average.



## References

`firewall-cmd(1)` man page

For more information, refer to the *Creating a high availability cluster with multiple links* chapter in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#proc\\_configure-multiple-ip-cluster-creating-high-availability-cluster](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#proc_configure-multiple-ip-cluster-creating-high-availability-cluster)

For more information, refer to the *Enabling ports for the High Availability Add-On* chapter in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#proc\\_enabling-ports-for-high-availability-creating-high-availability-cluster](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#proc_enabling-ports-for-high-availability-creating-high-availability-cluster)

**Knowledgebase: "Support Policies for RHEL High Availability Clusters - Cluster Interconnect Network Interfaces"**

<https://access.redhat.com/articles/3068841>

**Knowledgebase: "Support Policies for RHEL High Availability Clusters - Cluster Interconnect Network Latency"**

<https://access.redhat.com/articles/2823721>

## ► Guided Exercise

# Troubleshooting Cluster Networking

In this exercise, you will troubleshoot an issue with cluster networking.

### Outcomes

You should be able to troubleshoot cluster networking issues.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start troubleshooting-issue
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines, with `nodec` deployed with a failure.

### Instructions

At the start of this exercise, your three-node cluster is running with only two nodes. Because `nodec` has not joined the cluster, any further node failures will take the entire cluster offline. Investigate and fix the issue.

- 1. Use the `pcs status` and `pcs quorum status` commands to gather initial information.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. On `nodea`, run the `pcs status` command. Notice that `nodec` is offline and has not joined the cluster.

```
[root@nodea ~]# pcs status
...output omitted...
Node List:
  * Online: [ nodea.private.example.com nodeb.private.example.com ]
  * OFFLINE: [ nodec.private.example.com ]
...output omitted...
```

- 1.3. On `nodea`, run the `pcs quorum status` command. Notice that the cluster expects three votes, but only two are present. At the end of the output, notice that `nodec` is missing from the list.

```
[root@nodea ~]# pcs quorum status
...output omitted...
Votequorum information
-----
Expected votes: 3
Highest expected: 3
Total votes: 2
Quorum: 2
Flags: Quorate

Membership information
-----
  Nodeid   Votes   Qdevice Name
    1        1       NR nodea.private.example.com (local)
    2        1       NR nodeb.private.example.com
```

corosync and pcsd on the nodes that have quorum cannot reach nodec.

- 1.4. On nodea, determine if you can still ping the different network interfaces on nodec.

```
[root@nodea ~]# for I in nodec.{lab,private,san{01,02}}.example.com; do
> ping -c1 ${I}
> done
PING nodec.lab.example.com (172.25.250.12) 56(84) bytes of data.
64 bytes from nodec.lab.example.com (172.25.250.12): icmp_seq=1 ttl=64 time=0.869
ms

--- nodec.lab.example.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.869/0.869/0.869/0.000 ms
PING nodec.private.example.com (192.168.0.12) 56(84) bytes of data.
64 bytes from nodec.private.example.com (192.168.0.12): icmp_seq=1 ttl=64
time=0.558 ms

--- nodec.private.example.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.558/0.558/0.558/0.000 ms
PING nodec.san01.example.com (192.168.1.12) 56(84) bytes of data.
64 bytes from nodec.san01.example.com (192.168.1.12): icmp_seq=1 ttl=64 time=0.453
ms

--- nodec.san01.example.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.453/0.453/0.453/0.000 ms
PING nodec.san02.example.com (192.168.2.12) 56(84) bytes of data.
64 bytes from nodec.san02.example.com (192.168.2.12): icmp_seq=1 ttl=64 time=0.434
ms

--- nodec.san02.example.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.434/0.434/0.434/0.000 ms
```

Notice all four interfaces are still responding to ping requests.

- 2. From the workstation machine as the **student** user, log in to **nodec**, become the **root** user, and inspect how **nodec** views the cluster.

2.1. Connect to **nodec** and become the **root** user.

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo -i
[sudo] password for student: student
[root@nodec ~]#
```

2.2. Run the **pcs quorum status** command.

```
[root@nodec ~]# pcs quorum status
Error: Unable to get quorum status:
```

2.3. Run the **pcs status** command.

```
[root@nodec ~]# pcs status
...output omitted...
Node List:
* Online: [ nodec.private.example.com ]
* OFFLINE: [ nodea.private.example.com nodeb.private.example.com ]
...output omitted...
```

Notice that the cluster infrastructure services on **nodec** are active. This implies that cluster communications must be degraded in some way.

- 3. Based on the previous step, **nodec** cannot communicate with the rest of the cluster, even though commands like **ping** and **ssh** still work. A firewall problem affecting only the cluster services could be the cause for the communication issue. Investigate the firewall on **nodec**, and if it has any problems, fix them.

3.1. View the firewall configuration on **nodec**.

```
[root@nodec ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0 eth1 eth2 eth3
  sources:
  services: cockpit dhcpcv6-client high-availability http ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
    rule family="ipv4" port port="5405" protocol="udp" drop
```

**pacemaker** and **corosync** use UDP ports 5404 and 5405 to determine cluster membership.

- 3.2. View all rules that have been added to the firewall chains and tables on nodec.

```
[root@nodec ~]# firewall-cmd --direct --get-all-rules  
ipv4 filter OUTPUT 2 -p udp --dport=5405 -j DROP
```

corosync uses UDP port 5405.

- 3.3. Remove the rich rule in the firewall to allow communication with the cluster.

```
[root@nodec ~]# firewall-cmd \  
> --remove-rich-rule='rule family="ipv4" port port="5405" protocol="udp" drop'  
success
```

- 3.4. Restore communication with the cluster by removing the firewall rule added directly to the firewall.

```
[root@nodec ~]# firewall-cmd --direct --remove-rule \  
> ipv4 filter OUTPUT 2 -p udp --dport=5405 -j DROP  
success
```

- 3.5. Verify that the cluster is now fully operational again.

```
[root@nodec ~]# pcs status  
...output omitted...  
Node List:  
* Online: [ nodea.private.example.com nodeb.private.example.com  
nodec.private.example.com ]  
...output omitted...
```

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish troubleshooting-issue
```

This concludes the section.

## ▶ Lab

# Troubleshooting High Availability Clusters

In this lab, you will troubleshoot a faulty four-node cluster and its two resource groups.

## Outcomes

You should be able to troubleshoot a cluster.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start troubleshooting-review
```

This command deploys a four-node cluster on the nodea, nodeb, nodec, and noded machines, and configures two resource groups to provide highly available web and MariaDB database services, both with a failure. It also prepares an Ansible Playbook on the workstation machine to help you fix the issues.

## Instructions

This cluster should provide two services to consumers:

- An `http` service available on `http://172.25.250.80` (`www.lab.example.com`), serving content from the NFS share `storage.san01.example.com:/srv/www`.
- A MariaDB database available on `172.25.250.81` (`db.lab.example.com`), storing content on an XFS file system provided by iSCSI from `storage.san01.example.com` mounted on `/var/lib/mysql`.

Although both services are configured on the cluster, consumers cannot reach either one. Investigate and fix these issues.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Investigate the current cluster, including all defined resources.
2. Investigate and fix the issue with the MariaDB service. You can check the connection with the MariaDB service on `db.lab.example.com` with the `mysql -h db.lab.example.com -u root -p` command. The password for the `root` user is `redhat`. To fix any issues in the firewall, you can connect to the nodes and use the `firewall-cmd` command or, as an alternative, use the Ansible Playbook located in `/home/student/labs/troubleshooting-review/fix_troubledb.yml`. Confirm that you can send commands to the database with the `show databases;` command. It should list the three databases on the server: `information_schema`, `mysql`, and `performance_schema`.
3. Investigate and fix the issue with the HTTP service. To fix any issues with the HTTP configuration file, you can connect to the nodes and manually remove the offending

configuration file or, as an alternative, use the Ansible Playbook located in `/home/student/labs/troubleshooting-review/fix_troubleweb.yml`.

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade troubleshooting-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish troubleshooting-review
```

This concludes the section.

## ► Solution

# Troubleshooting High Availability Clusters

In this lab, you will troubleshoot a faulty four-node cluster and its two resource groups.

### Outcomes

You should be able to troubleshoot a cluster.

### Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start troubleshooting-review
```

This command deploys a four-node cluster on the `nodea`, `nodeb`, `nodec`, and `noded` machines, and configures two resource groups to provide highly available web and MariaDB database services, both with a failure. It also prepares an Ansible Playbook on the `workstation` machine to help you fix the issues.

### Instructions

This cluster should provide two services to consumers:

- An `http` service available on `http://172.25.250.80` (`www.lab.example.com`), serving content from the NFS share `storage.san01.example.com:/srv/www`.
- A MariaDB database available on `172.25.250.81` (`db.lab.example.com`), storing content on an XFS file system provided by iSCSI from `storage.san01.example.com` mounted on `/var/lib/mysql`.

Although both services are configured on the cluster, consumers cannot reach either one. Investigate and fix these issues.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Investigate the current cluster, including all defined resources.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. Run the `pcs status` command.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: troubledb:
  * mariadbfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * mariadbvip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com
  * mariadbserver (ocf::heartbeat:mysql): Started nodea.private.example.com
* Resource Group: troubleweb:
  * webfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
  * webserver (ocf::heartbeat:apache): Stopped
  * webip (ocf::heartbeat:IPAddr2): Stopped

Failed Resource Actions:
  * webserver_start_0 on noded.private.example.com 'error' (1): call=46,
    status='Timed Out', exitreason='', last-rc-change='2021-03-01 22:57:51 -05:00',
    queued=0ms, exec=40002ms
  * webserver_start_0 on nodea.private.example.com 'error' (1): call=52,
    status='Timed Out', exitreason='', last-rc-change='2021-03-01 22:58:32 -05:00',
    queued=0ms, exec=40001ms
  * webserver_start_0 on nodeb.private.example.com 'error' (1): call=42,
    status='Timed Out', exitreason='', last-rc-change='2021-03-01 22:56:30 -05:00',
    queued=0ms, exec=40002ms
  * webserver_start_0 on nodec.private.example.com 'error' (1): call=46,
    status='Timed Out', exitreason='', last-rc-change='2021-03-01 22:57:11 -05:00',
    queued=0ms, exec=40002ms
...output omitted...
```

Notice that all resources in the **troubledb** resource group are running as expected. Also notice that the cluster has trouble starting the **webserver** resource in the **troubleweb** resource group. If the status for the **webserver** resource is **Starting**, then wait a few seconds until it changes to the **Stopped** status.

- 1.3. View the full configuration for each resource defined in the **troubleweb** and **troubledb** resource groups.

```
[root@nodea ~]# pcs resource config
Group: troubledb
Resource: mariadbfs (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/sda directory=/var/lib/mysql fstype=xfs
  Operations: monitor interval=20s timeout=40s (mariadbfs-monitor-interval-20s)
               start interval=0s timeout=60s (mariadbfs-start-interval-0s)
               stop interval=0s timeout=60s (mariadbfs-stop-interval-0s)
Resource: mariadbvip (class=ocf provider=heartbeat type=IPAddr2)
  Attributes: cidr_netmask=24 ip=172.25.250.81
  Operations: monitor interval=10s timeout=20s (mariadbvip-monitor-interval-10s)
               start interval=0s timeout=20s (mariadbvip-start-interval-0s)
               stop interval=0s timeout=20s (mariadbvip-stop-interval-0s)
Resource: mariadbserver (class=ocf provider=heartbeat type=mysql)
  Attributes: additional_parameters="--bind-address=0.0.0.0 binary=/usr/bin/
mysqld_safe config=/etc/my.cnf datadir=/var/lib/mysql pid=/var/lib/mysql/mysql.pid
socket=/var/lib/mysql/mysql.sock"
  Operations: demote interval=0s timeout=120s (mariadbserver-demote-interval-0s)
               monitor interval=20s timeout=30s (mariadbserver-monitor-
interval-20s)
```

**Chapter 5 |** Troubleshooting High Availability Clusters

```

    notify interval=0s timeout=90s (mariadbserver-notify-interval-0s)
    promote interval=0s timeout=120s (mariadbserver-promote-
interval-0s)
        start interval=0s timeout=60s (mariadbserver-start-interval-0s)
        stop interval=0s timeout=60s (mariadbserver-stop-interval-0s)

Group: troubleweb
Resource: webfs (class=ocf provider=heartbeat type=Filesystem)
    Attributes: device=storage.san01.example.com:/srv/www directory=/var/www
    fstype=nfs options=ro
    Operations: monitor interval=20s timeout=40s (webfs-monitor-interval-20s)
        start interval=0s timeout=60s (webfs-start-interval-0s)
        stop interval=0s timeout=60s (webfs-stop-interval-0s)

Resource: webserver (class=ocf provider=heartbeat type=apache)
    Operations: monitor interval=10s timeout=20s (webserver-monitor-interval-10s)
        start interval=0s timeout=40s (webserver-start-interval-0s)
        stop interval=0s timeout=60s (webserver-stop-interval-0s)

Resource: webip (class=ocf provider=heartbeat type=IPAddr2)
    Attributes: cidr_netmask=24 ip=172.25.250.80
    Operations: monitor interval=10s timeout=20s (webip-monitor-interval-10s)
        start interval=0s timeout=20s (webip-start-interval-0s)
        stop interval=0s timeout=20s (webip-stop-interval-0s)

```

2. Investigate and fix the issue with the MariaDB service. You can check the connection with the MariaDB service on db.lab.example.com with the `mysql -h db.lab.example.com -u root -p` command. The password for the `root` user is `redhat`. To fix any issues in the firewall, you can connect to the nodes and use the `firewall-cmd` command or, as an alternative, use the Ansible Playbook located in `/home/student/labs/troubleshooting-review/fix_troubledb.yml`. Confirm that you can send commands to the database with the `show databases;` command. It should list the three databases on the server: `information_schema`, `mysql`, and `performance_schema`.
  - 2.1. On `workstation`, as the `student` user, use the `mysql` command to attempt to connect the MariaDB service on `db.lab.example.com`, authenticating to the database as user `root` with the password `redhat`.

```
[student@workstation ~]$ mysql -h db.lab.example.com -u root -p
Enter password: redhat
ERROR 2002 (HY000): Can't connect to MySQL server on 'db.lab.example.com' (115)
```

- 2.2. Because the preceding step failed, verify that the `troubledb` group is using the correct IP address, and that the IP address is reachable from `workstation`.  
On your `nodea` machine, inspect the `mariadbvip` resource.

```
[root@nodea ~]# pcs resource config mariadbvip
Resource: mariadbvip (class=ocf provider=heartbeat type=IPAddr2)
  Attributes: cidr_netmask=24 ip=172.25.250.81
...output omitted...
```

From your `workstation` machine, as the `student` user, attempt to ping the floating IP address.

```
[student@workstation ~]$ ping -c1 172.25.250.81
PING 172.25.250.81 (172.25.250.81) 56(84) bytes of data.
64 bytes from 172.25.250.81: icmp_seq=1 ttl=64 time=1.55 ms

--- 172.25.250.81 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.554/1.554/1.554/0.000 ms
```

- 2.3. The `troubledb` resource group is running without issues and the correct IP address is reachable. Verify that the firewall allows traffic to the service's port. Inspect the current firewall configuration on your nodes.

```
[root@nodea ~]# firewall-cmd --list-all
```

Repeat this process on all cluster nodes. The `mysql` service is not permitted by the firewall.

- 2.4. On all four of your nodes, add the `mysql` service to the set of services permitted by your firewall. As the `student` user on the `workstation` machine, change to the `/home/student/labs/troubleshooting-review/` project directory, and look at the content of the `fix_troubledb.yml` playbook. It configures the firewall on the cluster nodes to permit MariaDB traffic. Run the Ansible Playbook.

```
[student@workstation ~]$ cd /home/student/labs/troubleshooting-review/
[student@workstation troubleshooting-review]$ ansible-playbook fix_troubledb.yml
...output omitted...
```

- 2.5. On your `workstation` machine as `student`, connect to the MariaDB service on `db.lab.example.com` by using the `mysql` command, authenticating to the database as user `root` with the password `redhat`. This should now show a working MariaDB service.

```
[student@workstation ~]$ mysql -h db.lab.example.com -u root -p
Enter password: redhat
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.3.17-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

- 2.6. Confirm that you can send commands to the database. At the MariaDB prompt, run the `show databases;` command to list the databases on the server.

```
MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
```

```
| mysql          |
| performance_schema |
+-----+
3 rows in set (0.002 sec)
```

3. Investigate and fix the issue with the HTTP service. To fix any issues with the HTTP configuration file, you can connect to the nodes and manually remove the offending configuration file or, as an alternative, use the Ansible Playbook located in /home/student/labs/troubleshooting-review/fix\_troubleweb.yml.
  - 3.1. Determine which node is currently running the functioning parts of the troubleweb resource group.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: troubleweb:
  * webfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
  * webserver (ocf::heartbeat:apache): Stopped
  * webip (ocf::heartbeat:IPAddr2): Stopped
...output omitted...
```

- 3.2. On the node that you found in the previous step, as the root user, search the default pacemaker log location for any entries referencing webserver and ERROR.

```
[root@nodeb ~]# grep 'webserver.*ERROR' /var/log/pacemaker/pacemaker.log
Mar 01 22:56:30 apache(webserver)[223560]:    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
```

The error suggest a syntax error on the server configuration.

- 3.3. On all four of your nodes, fix the httpd configuration by removing the offending configuration file. As the student user on the workstation machine, change to the /home/student/labs/troubleshooting-review/ project directory, and then run the Ansible Playbook called fix\_troubleweb.yml to remove the file on the cluster nodes.

```
[student@workstation ~]$ cd /home/student/labs/troubleshooting-review/
[student@workstation troubleshooting-review]$ ansible-playbook fix_troubleweb.yml
...output omitted...
```

- 3.4. Reset the fail count for the webserver resource on all nodes.

```
[root@nodeb ~]# pcs resource cleanup webserver
Cleaned up webfs on noded.private.example.com
Cleaned up webfs on nodec.private.example.com
Cleaned up webfs on nodeb.private.example.com
Cleaned up webfs on nodea.private.example.com
Cleaned up webserver on noded.private.example.com
Cleaned up webserver on nodec.private.example.com
Cleaned up webserver on nodeb.private.example.com
Cleaned up webserver on nodea.private.example.com
Cleaned up webip on noded.private.example.com
```

**Chapter 5 |** Troubleshooting High Availability Clusters

```
Cleaned up webip on nodec.private.example.com
Cleaned up webip on nodeb.private.example.com
Cleaned up webip on nodea.private.example.com
Waiting for 4 replies from the controller.... OK
```

- 3.5. Verify that the **troubleweb** resource group is operating normally.

```
[root@nodeb ~]# pcs resource
  * Resource Group: troubledb:
    * mariadbfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com
    * mariadbvip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com
    * mariadbserver (ocf::heartbeat:mysql): Started nodea.private.example.com
  * Resource Group: troubleweb:
    * webfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
    * webserver (ocf::heartbeat:apache): Started nodeb.private.example.com
    * webip (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com
```

- 3.6. On your workstation machine, as **student** user, use the **curl** command to navigate to **www.lab.example.com**.

```
[student@workstation ~]$ curl www.lab.example.com
Hello HTTP World!
```

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade troubleshooting-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish troubleshooting-review
```

This concludes the section.

# 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

# Automating Cluster and Resource Deployment

### Goal

Deploy a new high availability cluster and cluster resources by using Ansible automation.

### Objectives

- Use Ansible to deploy a high availability cluster.
- Use Ansible to deploy and configure resources and resource groups in a high availability cluster.

### Sections

- Automating Cluster Deployment (and Guided Exercise)
- Automating Resource and Resource Group Deployment (and Guided Exercise)

### Lab

Automating Cluster and Resource Deployment

# Automating Cluster Deployment

## Objectives

After completing this section, you should be able to use Ansible to deploy a high availability cluster.

## Preparing Cluster Nodes

Preparing the cluster nodes using Ansible follows the standard process:

- Use the Ansible `yum` module to install the `pcs` and the fencing agent packages.
- Use the Ansible `firewalld` module to configure the firewall rules.
- Use the Ansible `user` module to set a password for the `hacluster` user account.
- Use the Ansible `service` module to enable and start the `pcsd` service.

This works much like other services you deploy with Ansible. Therefore, this section does not focus on how to write such an Ansible Playbook. Instead, it investigates how to use Ansible to create a cluster by using the `pcs` command.



### Note

You will have an opportunity to look at Ansible Playbooks that prepare the cluster nodes in the Guided Exercise for this section.

To target all the cluster nodes, group them in the inventory. The following example shows an inventory file with four cluster nodes in the `hanodes` group.

```
[hanodes]
node1.example.com
node2.example.com
node3.example.com
node4.example.com
```

## Using Cluster Commands with Ansible

Red Hat does not yet provide Ansible modules nor roles to manage High Availability clusters. However, there is an Ansible role for managing high-availability clusters under development. For more information about this role, see the entry in the References at the end of this section.

In the meantime, you can use the `command` Ansible module to wrap `pcs` commands in Ansible tasks. When doing so, take into account the following constraints:

- You run the `pcs` command only on one node.
- The `pcs` command often works asynchronously. It triggers the operation in the background and then returns with a success code, even though the operation might fail later.

- For idempotency, Ansible expects its modules to indicate whether the task has changed the system state or not. With the `command` module, you can use the `changed_when` Ansible keyword to report that state.

## Running Cluster Commands

You use the `pcs` command from only one node when you manage your cluster. In an Ansible Playbook, you do the same.

If your play targets all your cluster nodes through the `hosts` keyword, then add the `run_once` directive to all the tasks that use the `pcs` command. The following playbook example shows a task that runs the `pcs` command to set a cluster property.

```
---
- name: Ensure the cluster is configured
  hosts: hanodes ①
  become: yes

  tasks:
    - name: Ensuring the fence agent is installed on all the nodes
      yum:
        name: fence-agents-rhev
        state: present

    - name: Ensuring the global STONITH timeout is set
      command:
        cmd: pcs property set stonith-timeout=120s
      run_once: ②
    ...
  
```

- ➊ The play targets the `hanodes` group, which includes all the cluster nodes.
- ➋ The `run_once` directive instructs Ansible to select one host to run the task.

An alternative is to create separate plays in your playbook. The following playbook example uses two plays, the first one targets all the cluster nodes and the second one only runs on the `node1.example.com` machine.

```
---
- name: Ensure the fence agent is installed
  hosts: hanodes
  become: yes

  tasks:
    - name: Ensuring the fence agent is installed on all the nodes
      yum:
        name: fence-agents-apc
        state: present

- name: Ensure the cluster is configured
  hosts: node1.example.com
  become: yes

  tasks:
```

```
- name: Ensuring the global STONITH timeout is set
  command:
    cmd: pcs property set stonith-timeout=120s
  ...
```

## Handling Command Failure

By default, the status of an Ansible task that uses the `command` module depends on the command's return code. If the command returns 0, then the task is successful. Otherwise the task fails.

The `pcs` command complies with that rule. It returns 0 when it succeeds and another value when it fails. However, for some operations, the `pcs` command triggers a subcommand in the background and returns without waiting for the result. For example, the cluster executes the `pcs resource create` command in the background. The `pcs` command returns immediately with a success code (0).

Consequently, when you use the `pcs` command in Ansible, the associated task might report a status that does not reflect the subcommands actual status. For some operations, the `pcs` command accepts the `--wait` option. With that option, it waits for the completion of the subcommand to reports a status. The `--wait` option takes a number of seconds to wait for the completion of the subcommand.

The following task creates a STONITH resource and waits up to 40 seconds for the cluster to create the resource and start it. If the cluster correctly starts the resource, then the `pcs` command and the Ansible task succeed. If starting the resource fails or if the wait time expires, then the command and the task fail.

```
- name: Ensuring the fence resource exists
  command:
    cmd: >
      pcs stonith create fence_apcA_node1 fence_apc
      ip=10.12.0.7 username=apc password=s3cr3t pcmk_host_map=node1.example.com:2
      --wait=40
```

## Managing the Change State

With the Ansible `command` module, you can use the `changed_when` keyword to report the task's change state.

The following task records the `pcs` command's output and then only reports a change state if the command output contains the "Cluster Enabled" string. If that string is not in the output, then that means that the cluster is already enabled on the nodes. The task reports a "not changed" state.

```
- name: Ensuring the cluster is enabled
  command:
    cmd: pcs cluster enable --all
  register: enable_cluster
  changed_when: "'Cluster Enabled' in enable_cluster['stdout']"
```

In some other cases, you can create a first task that determines the state of a system's item and then a conditional second task that runs when the item has not the expected state. The following example illustrates that construct.

```
- name: Checking if the global STONITH timeout is set
  command:
    cmd: pcs property show stonith-timeout
  register: stonith ①
  changed_when: false ②

- name: Ensuring the global STONITH timeout is set
  command:
    cmd: pcs property set stonith-timeout=120s
  when: "'stonith-timeout: 120s' not in stonith['stdout']" ③
```

- ① The first task retrieves the value of the `stonith-timeout` cluster parameter and registers it in the `stonith` variable.
- ② That first task does not change anything; therefore the `changed_when` keyword is set to `false`.
- ③ The second task sets the `stonith-timeout` parameter only when the first task's output does not indicate that the parameter already has the correct value.



## References

[ansible-doc\(1\) man page](#)

**yum - Manages packages with the yum package manager – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/modules/yum\\_module.html](https://docs.ansible.com/ansible/latest/modules/yum_module.html)

**firewalld - Manage arbitrary ports/services with firewalld – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/modules/firewalld\\_module.html](https://docs.ansible.com/ansible/latest/modules/firewalld_module.html)

**user - Manage user accounts – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/modules/user\\_module.html](https://docs.ansible.com/ansible/latest/modules/user_module.html)

**service - Manage services – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/modules/service\\_module.html](https://docs.ansible.com/ansible/latest/modules/service_module.html)

**command - Execute commands on targets – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/modules/command\\_module.html](https://docs.ansible.com/ansible/latest/modules/command_module.html)

**Ansible role for managing High Availability Clustering**

[https://github.com/linux-system-roles/ha\\_cluster](https://github.com/linux-system-roles/ha_cluster)

## ► Guided Exercise

# Automating Cluster Deployment

In this exercise, you will use Ansible to deploy a high availability cluster.

### Outcomes

You should be able to write Ansible Playbooks that deploy a high availability cluster and configure fencing agents for cluster nodes.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start auto-ha
```

This command prepares an Ansible project in `/home/student/labs/auto-ha/` on `workstation`.

### Instructions

In this exercise, you use Ansible to deploy a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines.

#### ► 1. Get familiar with the Ansible project and its current state.

- 1.1. As the student user on `workstation`, change to the `/home/student/labs/auto-ha/` project directory.

```
[student@workstation ~]$ cd /home/student/labs/auto-ha  
[student@workstation auto-ha]$
```

- 1.2. List the files that the `lab` command has prepared for you.

```
[student@workstation auto-ha]$ tree  
. . .  
└── 01-preparing.yml  
└── 02-deploying.yml  
└── 03-stonith.yml  
└── ansible.cfg  
└── create_ipmi.yml  
└── inventory  
└── passwords.yml  
└── solutions  
    └── 02-deploying.yml  
  
1 directory, 8 files
```

The project provides three playbooks to deploy the cluster, `01-preparing.yml`, `02-deploying.yml`, and `03-stonith.yml`. In the following steps, you review, complete, and then run those playbooks.

For your convenience, the `solutions/` directory provides a completed version of the `02-deploying.yml` playbook. The other two playbooks have been provided for you in complete form.

1.3. Review the inventory file.

```
[nodes]
nodea
nodeb
nodec
```

The file groups the three cluster machines under the `nodes` group.

► 2. Review the node preparation playbook and then run it.

2.1. Display the content of the `01-preparing.yml` playbook. You do not have to modify anything in this file.

```
[student@workstation auto-ha]$ cat 01-preparing.yml
---
- name: Preparing the nodes for Red Hat High Availability cluster
  hosts: nodes ①
  become: yes
  gather_facts: no
  vars_files:
    - passwords.yml ②

  tasks:
    - name: Ensuring the cluster packages are present ③
      yum:
        state: present
        name:
          - pcs
          - fence-agents-ipmilan

    - name: Ensuring the required ports are open ④
      firewalld:
        service: high-availability
        permanent: yes
        state: enabled
        immediate: yes

    - name: Ensuring the password for hacluster is configured ⑤
      user:
        name: hacluster
        password: "{{ ha_password | password_hash('sha512') }}"

    - name: Ensuring the pcsd service is started and enabled ⑥
      service:
        name: pcsd
```

```
state: started  
enabled: yes  
...
```

- ➊ The play targets the `nodes` group, which groups the three cluster nodes. Therefore, the following tasks run on all three nodes.
  - ➋ The `passwords.yml` file defines variables for passwords, such as the password for the `hacluster` user or the password to access the IPMI over LAN fencing device. The `passwords.yml` file has been encrypted by using Ansible Vault with `redhat` as its Vault password.
  - ➌ The first task installs the cluster packages. This includes the `fence-agents-ipmilan` package, which provides the IPMI over LAN fencing agent.
  - ➍ The second task prepares the firewall.
  - ➎ The next task sets the password for the `hacluster` system account. The `ha_password` variable is defined in the `passwords.yml` file.
  - ➏ The last task enables and starts the `pcsd` service.
- 2.2. Use the `ansible-vault` command to view the content of the `passwords.yml` file. The Vault password is `redhat`.

```
[student@workstation auto-ha]$ ansible-vault view passwords.yml  
Vault password: redhat  
---  
ha_password: tbe6W3hz  
ipmi_login: admin  
ipmi_password: password  
...
```

- 2.3. Run the playbook. Use the `--ask-vault-pass` option to specify the Vault password. The password is `redhat`.

```
[student@workstation auto-ha]$ ansible-playbook --ask-vault-pass 01-preparing.yml  
Vault password: redhat  
...output omitted...
```

- 3. Review and complete the `02-deploying.yml` Ansible Playbook. That playbook uses the `pcs` command on one cluster node to create, enable, and then start the cluster.

- 3.1. Open the `02-deploying.yml` file and review the play parameters. You do not have to modify anything at this point.

```
---  
- name: Deploying a Red Hat High Availability cluster  
  hosts: nodea ①  
  become: yes  
  gather_facts: no  
  vars_files:  
    - passwords.yml ②  
  vars:
```

```
ha_cluster_name: cluster1
ha_nodes: ③
  - nodea.private.example.com
  - nodeb.private.example.com
  - nodec.private.example.com
...output omitted...
```

- ① The play targets one of the cluster nodes. Remember that the `pcs` command used in the play tasks needs to run only on one cluster node.
- ② The `passwords.yml` file defines the `ha_password` variable, which contains the password for the `hacluster` system user. A play task uses that variable.
- ③ The `ha_nodes` variable defines the names of the cluster nodes.

Do not close the file yet.

- 3.2. Complete the first task. That task uses the `pcs host auth` command to authenticate the cluster nodes. Use the `-u` and `-p` options to provide the user name and the password so that the command does not ask for them. The `ha_nodes` variable contains a YAML list of nodes. Convert it to a space-delimited list of nodes for the command by using the `join(' ')` Jinja2 filter.

```
...output omitted...
tasks:
  - name: Ensuring the cluster nodes are authenticated
    command:
      cmd: "pcs host auth -u hacluster -p {{ ha_password }}
            {{ ha_nodes | join(' ') }}"
    register: auth_cluster
    changed_when: "'Authorized' in auth_cluster['stdout']"
...output omitted...
```

Do not close the file yet.

- 3.3. Complete the third task. This task uses the `pcs cluster setup` command to create the cluster. The `ha_cluster_name` variable provides the name of the cluster.

```
...output omitted...
  - name: Ensuring the cluster exists
    command:
      cmd: "pcs cluster setup {{ ha_cluster_name }} {{ ha_nodes | join(' ') }}"
    register: create_cluster
    changed_when: "'successfully set up' in create_cluster['stdout']"
    when: not cluster_config['stat']['exists']
...output omitted...
```

Do not close the file yet.

- 3.4. Complete the next task. The task enables the cluster processes on all the nodes.

```
...output omitted...
- name: Ensuring the cluster is enabled
  command:
    cmd: pcs cluster enable --all
  register: enable_cluster
  changed_when: "'Cluster Enabled' in enable_cluster['stdout']"
...output omitted...
```

Do not close the file yet.

- 3.5. Complete the last task. The task starts the cluster processes on all the nodes. Set the `--request-timeout` and `--wait` parameters so that the `pcs` command only returns when it has successfully started the cluster. A value of 180 seconds should give enough time for that start process to complete.

```
...output omitted...
- name: Ensuring the cluster is started
  command:
    cmd: pcs cluster start --all --request-timeout=180 --wait=180
  register: start_cluster
  changed_when: "'Starting Cluster' in start_cluster['stdout']"
...
```

The resulting file should display as follows:

```
---
- name: Deploying a Red Hat High Availability cluster
  hosts: nodea
  become: yes
  gather_facts: no
  vars_files:
    - passwords.yml
  vars:
    ha_cluster_name: cluster1
    ha_nodes:
      - nodea.private.example.com
      - nodeb.private.example.com
      - nodec.private.example.com

  tasks:
    - name: Ensuring the cluster nodes are authenticated
      command:
        cmd: "pcs host auth -u hacluster -p {{ ha_password }}"
        "{{ ha_nodes | join(' ') }}"
      register: auth_cluster
      changed_when: "'Authorized' in auth_cluster['stdout']"

    - name: Checking the cluster configuration
      stat:
        path: /etc/corosync/corosync.conf
      register: cluster_config

    - name: Ensuring the cluster exists
```

```
command:  
  cmd: "pcs cluster setup {{ ha_cluster_name }} {{ ha_nodes | join(' ') }}"  
register: create_cluster  
changed_when: "'successfully set up' in create_cluster['stdout']"  
when: not cluster_config['stat']['exists']  
  
- name: Ensuring the cluster is enabled  
  command:  
    cmd: pcs cluster enable --all  
  register: enable_cluster  
  changed_when: "'Cluster Enabled' in enable_cluster['stdout']"  
  
- name: Ensuring the cluster is started  
  command:  
    cmd: pcs cluster start --all --request-timeout=180 --wait=180  
  register: start_cluster  
  changed_when: "'Starting Cluster' in start_cluster['stdout']"  
...  
...
```

When done, save and then close the file.

► 4. Verify the syntax of the `02-deploying.yml` playbook and then run it.

- 4.1. Verify the syntax of the Ansible Playbook. Because the `passwords.yml` file is encrypted, use the `--ask-vault-pass` option to specify the Vault password. Use `redhat` for the password.

```
[student@workstation auto-ha]$ ansible-playbook --ask-vault-pass \  
> --syntax-check 02-deploying.yml  
Vault password: redhat  
  
playbook: 02-deploying.yml
```

If the command reports errors, then fix them before running the playbook. For your convenience, you can compare your work with the already completed `/home/student/labs/auto-ha/solutions/02-deploying.yml` file.

- 4.2. Run the playbook. Use the `--ask-vault-pass` option to specify the Vault password. The password is `redhat`.

```
[student@workstation auto-ha]$ ansible-playbook --ask-vault-pass 02-deploying.yml  
Vault password: redhat  
...output omitted...
```

If there are errors, then make any necessary edits and then execute the playbook again.

► 5. Review the playbook that creates the STONITH resources and then run it.

- 5.1. Display the content of the `03-stonith.yml` playbook. You do not have to modify anything in this file.

```
[student@workstation auto-ha]$ cat 03-stonith.yml  
---  
- name: Configuring fencing
```

```
hosts: nodea 1
become: yes
gather_facts: no
vars_files:
  - passwords.yml 2
vars:
  stonith_timeout: 180 3

tasks:
  - name: Checking if the global STONITH timeout is set 4
    command:
      cmd: pcs property show stonith-timeout
    register: stonith
    changed_when: false

  - name: Ensuring the global STONITH timeout is set 5
    command:
      cmd: "pcs property set stonith-timeout={{ stonith_timeout }}s"
      when: "('stonith-timeout: ' + stonith_timeout|string + 's') not in stonith['stdout']"

  - name: Ensuring the STONITH resources exist
    include_tasks: create_ipmi.yml 6
    loop:
      - id: fence_nodea
        node: nodea.private.example.com
        ip: 192.168.0.101
        # ipmi_login and ipmi_password are defined in the Vault protected
        # passwords.yml file.
        login: "{{ ipmi_login }}"
        password: "{{ ipmi_password }}"
      - id: fence_nodeb
        node: nodeb.private.example.com
        ip: 192.168.0.102
        login: "{{ ipmi_login }}"
        password: "{{ ipmi_password }}"
      - id: fence_nodec
        node: nodec.private.example.com
        ip: 192.168.0.103
        login: "{{ ipmi_login }}"
        password: "{{ ipmi_password }}"
    ...
  
```

- 1** The play targets one node because the `pcs` command must only run on one cluster node.
- 2** The `passwords.yml` file defines the `ipmi_login` and `ipmi_password` variables. Those variables provide the connection parameters to access the IPMI over LAN device.
- 3** The `stonith_timeout` parameter specifies how long the STONITH resource must wait for the fencing device to reply. Because in the classroom environment

the BMC device might take a long time to reply, the playbook sets that timeout to 180 seconds.

- ④ The task retrieves the value of the `stonith-timeout` cluster parameter.
- ⑤ The task updates the `stonith-timeout` cluster parameter only if it does not already have the correct value.
- ⑥ The task creates one STONITH resource per node. To do so, it calls the `create_ipmi.yml` task file in a loop.

## 5.2. Display the content of the `create_ipmi.yml` task file.

```
[student@workstation auto-ha]$ cat create_ipmi.yml
---
# Task file that creates a fence resource using the fence_ipmilan fencing agent.
#
# Expected variables:
#   stonith_timeout
#   item
#
# The item variable must be a dictionary with the following entries:
#   id: name of the fence resource to create or update
#   node: name of the cluster node
#   ip: IP address of the IPMI over LAN device
#   login: User name for accessing the IPMI over LAN device
#   password: Associated password

- name: Checking if the STONITH resource exists ①
  command:
    cmd: "pcs stonith config {{ item['id'] }}"
  register: result
  failed_when: false
  changed_when: false

- name: Ensuring the fence resource exists ②
  command:
    cmd: "pcs stonith create {{ item['id'] }} fence_ipmilan
          pcmk_host_list={{ item['node'] }}
          ip={{ item['ip'] }}
          username={{ item['login'] }}
          password={{ item['password'] }}
          lanplus=1
          power_timeout={{ stonith_timeout }}"
  when: result['rc'] != 0

- name: Ensuring the fence resource is updated ③
  command:
    cmd: "pcs stonith update {{ item['id'] }}
          pcmk_host_list={{ item['node'] }}
          ip={{ item['ip'] }}
          username={{ item['login'] }}
          password={{ item['password'] }}
          lanplus=1"
```

```
power_timeout={{ stonith_timeout }}"
when: result['rc'] == 0
...
```

- ➊ The task determines whether the STONITH resource already exists.
  - ➋ If the resource does not exist, then that task creates it.
  - ➌ If the resource exists, then the task updates its parameters.
- 5.3. Run the playbook. Use the --ask-vault-pass option to specify the Vault password. The password is redhat.

```
[student@workstation auto-ha]$ ansible-playbook --ask-vault-pass 03-stonith.yml
Vault password: redhat
...output omitted...
```

▶ 6. Confirm that you have correctly deployed the cluster.

- 6.1. Connect to nodea and become the root user.

```
[student@workstation auto-ha]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 6.2. Run the pcs status command to confirm that the cluster is working as expected.

```
[root@nodea ~]# pcs status
Cluster name: cluster1
Cluster Summary:
  * Stack: corosync
  * Current DC: nodec.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
  * Last updated: Wed Feb 17 03:34:15 2021
  * Last change: Wed Feb 17 03:32:04 2021 by root via cibadmin on
nodea.private.example.com
  * 3 nodes configured
  * 3 resource instances configured

Node List:
  * Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

Full List of Resources:
  * fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
  * fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com

Daemon Status:
```

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

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish auto-ha
```

This concludes the section.

# Automating Resource and Resource Group Deployment

## Objectives

After completing this section, you should be able to use Ansible to deploy and configure resources and resource groups in a high availability cluster.

## Creating Resources and Resource Groups Using Ansible

To create resources and resource groups, use the Ansible command module to run `pcs resource create` commands.

Because the `pcs` command needs to run only on one cluster node, use the `run_once` keyword with the Ansible task or create a play that only targets one cluster node.

The `pcs resource create` command accepts the `--wait` option to instruct the command to wait for the completion of the operation before returning. Use that option to make sure that the task reports the correct completion status.

Make sure that the resource you create does not already exist. The `pcs resource create` command fails otherwise. To do so, you can list the existing resources by using the `pcs resource config` command in a task and then inspect that list before creating your resources.

The following play example retrieves the existing resources and then creates the `mynginx` and `myip` resources in the `myweb` resource group if they do not already exist.

```
---
- name: Create the cluster resources to manage Nginx
  hosts: node1.example.com
  become: yes

  tasks:
    - name: Collecting the existing resources
      command:
        cmd: pcs resource config
      changed_when: false
      register: resources 1

    - name: Ensuring the mynginx resource exists
      command:
        cmd: pcs resource create mynginx service:nginx --group=myweb --wait=40
      when: "' mynginx ' not in resources['stdout']" 2

    - name: Ensuring the myip resource exists
      command:
        cmd: >
          pcs resource create myip IPAddr2
```

```
ip=10.42.0.80 cidr_netmask=24 --group=myweb --wait=40
when: "' myip ' not in resources['stdout']'" ③
...
```

- ① The first task gathers the resource details in the `resources` variable.
- ② The second task only runs when the resource name, `mynginx`, does not show in the standard output of the `pcs resource config` command. The `resources['stdout']` entry contains the standard output of that command. The following example shows that output when the two resources already exist.

```
Group: myweb
Resource: mynginx (class=service type=nginx)
Operations: monitor interval=60 timeout=100 (mynginx-monitor-interval-60)
            start interval=0s timeout=100 (mynginx-start-interval-0s)
            stop interval=0s timeout=100 (mynginx-stop-interval-0s)
Resource: myip (class=ocf provider=heartbeat type=IPAddr2)
Attributes: cidr_netmask=24 ip=10.42.0.80
Operations: monitor interval=10s timeout=20s (myip-monitor-interval-10s)
            start interval=0s timeout=20s (myip-start-interval-0s)
            stop interval=0s timeout=20s (myip-stop-interval-0s)
```

- ③ The third task only runs when the resource name, `myip`, does not show in the standard output of the `pcs resource config` command.



### Note

The preceding playbook creates the resources if they do not already exist. It does not update existing resources with new parameters.

Consequently, the playbook works well when creating the initial resources or when adding new resources to your cluster. It does not work when you want to change some parameters in existing resources.

## Deploying Applications Using Ansible

In addition to setting up the resources and resource group in the cluster software itself, you can also use Ansible to ensure the cluster nodes have all the prerequisites they need to run the application.

For example, when deploying a resource group to run Apache HTTP Server, the playbook should also prepare all the nodes to run that web service. This preparation includes installing the `httpd` package, setting up consistently the configuration files in the `/etc/httpd/` directory across all nodes, adding the `http` service to the firewall rules, and ensuring that the nodes can all see the correct data to serve.



## References

### **command - Execute commands on targets – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/modules/command\\_module.html](https://docs.ansible.com/ansible/latest/modules/command_module.html)

### **Conditionals – Ansible Documentation**

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html)

## ► Guided Exercise

# Automating Resource and Resource Group Deployment

In this exercise, you will use Ansible to deploy a resource group in a high availability cluster.

### Outcomes

You should be able to develop Ansible Playbooks for deploying resources and resource groups in a high availability cluster.

### Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start auto-res
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines and shares the `/srv/www` directory on the `storage` machine by using NFS. It also prepares an Ansible project in the `/home/student/labs/auto-res/` directory on `workstation`.

### Instructions

You have been provided with an Ansible project that contains several playbooks, some of which are incomplete. You review, complete, and then run the playbooks to deploy a resource group and its resources on your cluster.

► 1. Get familiar with the Ansible project and its current state.

- 1.1. As the `student` user on `workstation`, change to the `/home/student/labs/auto-res/` project directory.

```
[student@workstation ~]$ cd /home/student/labs/auto-res  
[student@workstation auto-res]$
```

- 1.2. List the files that the `lab` command has prepared for you.

```
[student@workstation auto-res]$ tree  
. .  
└── 01-deploy-httpd.yml  
└── 02-create-resources.yml  
└── 03-smoke-test.yml  
└── ansible.cfg  
└── inventory  
└── resourcegroup.yml  
└── solutions
```

```
└── 02-create-resources.yml
```

1 directory, 7 files

The project provides three playbooks to deploy the resources, `01-deploy-httpd.yml`, `02-create-resources.yml`, and `resourcegroup.yml`. In the following steps, you review, complete, and then run those playbooks.

For your convenience, the `solutions/` directory provides a completed version of the `02-create-resources.yml` playbook. The other two playbooks have been provided for you in complete form.

- 1.3. Review the `inventory` file.

```
[student@workstation auto-res]$ cat inventory
[nodes]
nodea
nodeb
nodec
```

This inventory organizes the three cluster machines into an Ansible group named `nodes`.

- 2. Display the content of the `01-deploy-httpd.yml` playbook. You do not have to modify anything in this file.

```
[student@workstation auto-res]$ cat 01-deploy-httpd.yml
---
- name: Deploying Apache HTTP Server on the nodes
  hosts: nodes ①
  become: yes
  gather_facts: no

  tasks:
    - name: Ensuring the httpd package is installed ②
      yum:
        name: httpd
        state: present

    - name: Ensuring the required ports are open ③
      firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: yes

    - name: Ensuring SELinux allows Apache HTTP Server to access NFS shares ④
      seboolean:
        name: httpd_use_nfs
        state: yes
        persistent: yes
  ...
```

**①** The play targets the `nodes` group, which groups the three cluster nodes. Therefore, the following tasks run on all three nodes.

- ❷ The first task installs the `httpd` package.
  - ❸ The second task prepares the firewall to allow HTTP communication.
  - ❹ The third task prepares SELinux to allow the Apache HTTP Server to access NFS shares.
- ▶ 3. Review and complete the `02-create-resources.yml` Ansible Playbook. This playbook uses the `pcs` command on one cluster node to create the resources.
- 3.1. Display the content of the `02-create-resources.yml` playbook.

```
[student@workstation auto-res]$ cat 02-create-resources.yml
---
- name: Creating the cluster resources to manage Apache HTTP Server
  hosts: nodes[0] ❶
  become: yes
  gather_facts: no

  tasks:
    - name: Collecting the existing resources ❷
      command:
        cmd: pcs resource config
      changed_when: false
      register: resources

...output omitted...
```

- ❶ The play targets the first machine in the `nodes` group. Remember that the `pcs` command used in the play tasks needs to run only on one cluster node.
  - ❷ The first task displays a list of all configured resources and the parameters configured for those resources.
- 3.2. Complete the second task. Use the `pcs` command to create a `Filesystem` resource called `firstwebfs` that mounts the `storage.san01.example.com:/srv/www` share read-only on `/var/www`. The web content is already deployed in that share. Make sure to put the resource in the `firstweb` resource group.

```
...output omitted...
- name: Ensuring the firstwebfs resource exists
  command:
    cmd: >
      pcs resource create firstwebfs Filesystem
      device=storage.san01.example.com:/srv/www directory=/var/www
      fstype=nfs options=ro --group=firstweb --wait=60
    when: "' firstwebfs ' not in resources['stdout']"
...output omitted...
```

Do not close the file yet.

- 3.3. Complete the third task. Create an `apache` resource called `firstwebserver`. Make sure to put the resource in the `firstweb` resource group.

```
...output omitted...
- name: Ensuring the firstwebserver resource exists
  command:
    cmd: pcs resource create firstwebserver apache --group=firstweb --wait=60
    when: "' firstwebserver ' not in resources['stdout']"
...output omitted...
```

Do not close the file yet.

- 3.4. Complete the last task. Create an IPAddr2 resource called `firstwebip` that uses the `172.25.250.80/24` IP address. Make sure to put the resource in the `firstweb` resource group.

```
...output omitted...
- name: Ensuring the firstwebip resource exists
  command:
    cmd: >
      pcs resource create firstwebip IPAddr2
      ip=172.25.250.80 cidr_netmask=24 --group=firstweb --wait=60
    when: "' firstwebip ' not in resources['stdout']"
...output omitted...
```

The resulting file should display as follows:

```
---
- name: Creating the cluster resources to manage Apache HTTP Server
  hosts: nodes[0]
  become: yes
  gather_facts: no

  tasks:
    - name: Collecting the existing resources
      command:
        cmd: pcs resource config
      changed_when: false
      register: resources

    - name: Ensuring the firstwebfs resource exists
      command:
        cmd: >
          pcs resource create firstwebfs Filesystem
          device=storage.san01.example.com:/srv/www directory=/var/www
          fstype=nfs options=ro --group=firstweb --wait=60
      when: "' firstwebfs ' not in resources['stdout']"

    - name: Ensuring the firstwebserver resource exists
      command:
        cmd: pcs resource create firstwebserver apache --group=firstweb --wait=60
      when: "' firstwebserver ' not in resources['stdout']"

    - name: Ensuring the firstwebip resource exists
      command:
        cmd: >
```

```
pcs resource create firstwebip IPAddr2
  ip=172.25.250.80 cidr_netmask=24 --group=firstweb --wait=60
when: "' firstwebip ' not in resources['stdout']"
...
```

When done, save and then close the file.

- 3.5. Verify the syntax of the `02-create-resources.yml` playbook.

```
[student@workstation auto-res]$ ansible-playbook \
> --syntax-check 02-create-resources.yml

playbook: 02-create-resources.yml
```

- 4. Review the content of the `resourcegroup.yml` playbook and then run it.

- 4.1. Display the content of the `resourcegroup.yml` playbook. You do not have to modify anything in this file.

```
[student@workstation auto-res]$ cat resourcegroup.yml
---
- name: Include a play to deploy Apache HTTP Server on the nodes
  import_playbook: 01-deploy-httpd.yml ①

- name: Include a play to deploy the cluster resources to manage Apache HTTP
  Server
  import_playbook: 02-create-resources.yml ②
...
```

- ① The task imports then executes the `01-deploy-httpd.yml` playbook.
- ② The task imports then executes the `02-create-resources.yml` playbook.

- 4.2. Run the playbook.

```
[student@workstation auto-res]$ ansible-playbook resourcegroup.yml
...output omitted...
```

- 5. Use the `pcs` command to confirm that you have correctly deployed the resource group. Use the `curl` command to test the web server.

- 5.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 5.2. Run the `pcs status` command to confirm that the resource group has been created as expected.

```
[root@nodea ~]# pcs status
...output omitted...
* Resource Group: firstweb:
  * firstwebfs    (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * firstwebserver (ocf::heartbeat:apache):      Started nodea.private.example.com
  * firstwebip     (ocf::heartbeat:IPAddr2):      Started nodea.private.example.com
...output omitted...
```

- 5.3. On workstation, use the `curl` command to access `http://172.25.250.80`.  
The web server running in the cluster returns the sample page.

```
[student@workstation ~]$ curl http://172.25.250.80
Hello, world!
```

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish auto-res
```

This concludes the section.

## ▶ Lab

# Automating Cluster and Resource Deployment

In this lab, you will use Ansible to deploy a high availability cluster and a resource group in that cluster.

## Outcomes

You should be able to write Ansible Playbooks to create high availability clusters and resources.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start auto-review
```

This command prepares an Ansible project in `/home/student/labs/auto-review/` on `workstation`.

## Instructions

In this exercise, you write an Ansible Playbook that deploys a three-node cluster on the `nodeb`, `nodec`, and `noded` machines.

During development, you can run your playbook to verify your work. To clean up the machines between playbook executions, run the `reset.yml` playbook. The exercise preparation command has deployed that playbook for you in the `/home/student/labs/auto-review/` directory on `workstation`.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Install Apache HTTP Server on the `nodeb`, `nodec`, and `noded` machines and deploy some sample web content. To do so, run the `web_server.yml` playbook from the `/home/student/labs/auto-review/` directory on `workstation`. You do not have to modify anything in this file.
2. Complete the first play in the `cluster.yml` playbook. The exercise preparation command has deployed the playbook for you. If you need some help, then example tasks are available under the `task_examples/` directory. You can copy and then paste those tasks into your `cluster.yml` playbook. You have to adjust some of the tasks.  
The `passwords.yml` Vault file contains passwords and sensitive information. The Vault password is `redhat`.
3. Complete the second play of the `cluster.yml` playbook. That second play must deploy a three-node cluster using the `nodeb.private.example.com`, `nodec.private.example.com`, and `noded.private.example.com` machines. For help,

## Chapter 6 | Automating Cluster and Resource Deployment

use the example tasks available under the `task_examples/` directory. When done, run the playbook to verify your work. The password for the `passwords.yml` Vault file is `redhat`.

When the deployment is successful, the web content is available through the `172.25.250.80` IP address:

```
[student@workstation ~]$ curl 172.25.250.80  
Hello, world!
```

If there are errors in your playbook, then make any necessary edits, run the `reset.yml` playbook to clean up the machines, and then execute the `web_server.yml` and `cluster.yml` playbooks again.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade auto-review
```

## Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish auto-review
```

This concludes the section.

## ► Solution

# Automating Cluster and Resource Deployment

In this lab, you will use Ansible to deploy a high availability cluster and a resource group in that cluster.

### Outcomes

You should be able to write Ansible Playbooks to create high availability clusters and resources.

### Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start auto-review
```

This command prepares an Ansible project in `/home/student/labs/auto-review/` on `workstation`.

### Instructions

In this exercise, you write an Ansible Playbook that deploys a three-node cluster on the `nodeb`, `nodec`, and `noded` machines.

During development, you can run your playbook to verify your work. To clean up the machines between playbook executions, run the `reset.yml` playbook. The exercise preparation command has deployed that playbook for you in the `/home/student/labs/auto-review/` directory on `workstation`.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Install Apache HTTP Server on the `nodeb`, `nodec`, and `noded` machines and deploy some sample web content. To do so, run the `web_server.yml` playbook from the `/home/student/labs/auto-review/` directory on `workstation`. You do not have to modify anything in this file.
  - 1.1. As the `student` user on `workstation`, change to the `/home/student/labs/auto-review/` project directory.

```
[student@workstation ~]$ cd /home/student/labs/auto-review  
[student@workstation auto-review]$
```

- 1.2. Review the `inventory` file.

```
[cluster_nodes]
nodeb
nodec
noded
```

The file groups the three cluster machines under the `cluster_nodes` group.

- 1.3. Display the content of the `web_server.yml` playbook. You do not have to modify anything in this file.

```
[student@workstation auto-review]$ cat web_server.yml
---
- name: Ensure the Apache HTTP Server is installed
  hosts: cluster_nodes
  become: yes
  gather_facts: no

  tasks:
    - name: Ensuring the httpd package is installed
      yum:
        name: httpd
        state: present

    - name: Ensuring the required port is open
      firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: yes

    - name: Ensuring the initial web content is deployed
      copy:
        content: "Hello, world!\n"
        dest: /var/www/html/index.html
        mode: 0644
        group: root
        owner: root
        setype: httpd_sys_content_t
...

```

- 1.4. Run the playbook.

```
[student@workstation auto-review]$ ansible-playbook web_server.yml
...output omitted...
```

2. Complete the first play in the `cluster.yml` playbook. The exercise preparation command has deployed the playbook for you. If you need some help, then example tasks are available under the `task_examples/` directory. You can copy and then paste those tasks into your `cluster.yml` playbook. You have to adjust some of the tasks.

The `passwords.yml` Vault file contains passwords and sensitive information. The Vault password is `redhat`.

- 2.1. Edit the `cluster.yml` file and review the first play.

```
---
- name: Preparing the nodes for Red Hat High Availability cluster
  hosts: cluster_nodes
  become: yes
  gather_facts: no
  vars_files:
    - passwords.yml

  tasks:
    # Install the packages
    # Open the firewall ports
    # Set the password of the hacluster user to {{ ha_password }}
    # Enable and start the pcsd service
...output omitted...
```

- 2.2. Use the `ansible-vault` command to view the content of the `passwords.yml` file. The Vault password is `redhat`.

```
[student@workstation auto-review]$ ansible-vault view passwords.yml
Vault password: redhat
---
ha_password: redhat
ipmi_login: admin
ipmi_password: password
...
```

- 2.3. List the task examples under the `task_examples/` directory.

```
[student@workstation auto-review]$ ls task_examples/
cluster_auth.yml      firewall.yml  stonith_resources.yml  web_ip.yml
cluster_enable_start.yml packages.yml  stonith_timeout.yml  web_server.yml
cluster_setup.yml      services.yml  user_password.yml
```

To complete the first play, you can copy and then paste the content of the `packages.yml`, `firewall.yml`, `user_password.yml`, and `services.yml` task files. You must adjust the `user_password.yml` example to use the `ha_password` variable instead of the example 'FIXME' value:

```
password: "{{ ha_password | password_hash('sha512') }}"
```

- 2.4. Edit the `cluster.yml` file and complete the first play. The resulting file should look like this:

```
---
- name: Preparing the nodes for Red Hat High Availability cluster
  hosts: cluster_nodes
  become: yes
  gather_facts: no
  vars_files:
    - passwords.yml

  tasks:
```

```
- name: Ensuring the cluster packages are present
  yum:
    state: present
    name:
      - pcs
      - fence-agents-ipmilan

- name: Ensuring the required ports are open
  firewalld:
    service: high-availability
    permanent: yes
    state: enabled
    immediate: yes

- name: Ensuring the password for hacluster is configured
  user:
    name: hacluster
    password: "{{ ha_password | password_hash('sha512') }}"

- name: Ensuring the pcscd service is started and enabled
  service:
    name: pcscd
    state: started
    enabled: yes
...output omitted...
```

3. Complete the second play of the `cluster.yml` playbook. That second play must deploy a three-node cluster using the `nodeb.private.example.com`, `nodec.private.example.com`, and `noded.private.example.com` machines. For help, use the example tasks available under the `task_examples/` directory. When done, run the playbook to verify your work. The password for the `passwords.yml` Vault file is `redhat`.

When the deployment is successful, the web content is available through the `172.25.250.80` IP address:

```
[student@workstation ~]$ curl 172.25.250.80
Hello, world!
```

If there are errors in your playbook, then make any necessary edits, run the `reset.yml` playbook to clean up the machines, and then execute the `web_server.yml` and `cluster.yml` playbooks again.

- 3.1. Edit the `cluster.yml` file and review the second play.

```
...output omitted...
- name: Deploying a Red Hat High Availability cluster
  hosts: nodeb
  become: yes
  gather_facts: no
  vars_files:
    - passwords.yml
  vars:
    ha_cluster_name: cluster2
    ha_nodes:
      - nodeb.private.example.com
```

```

    - nodec.private.example.com
    - noded.private.example.com
stonith_timeout: 180
web_ip: 172.25.250.80
web_mask: 24

tasks:
  # Authenticate the cluster nodes
  # Create the cluster
  # Enable the cluster
  # Start the cluster
  # Set the global STONITH timeout
  # Create the resources for the fencing devices
  # Create the resource to manage the Apache HTTP Server (resource group: web)
  # Create the resource for the {{ web_ip }} IP address (resource group: web)
...

```

3.2. Review the example tasks under the `task_examples/` directory.

```
[student@workstation auto-review]$ ls task_examples/
cluster_auth.yml      firewall.yml  stonith_resources.yml  web_ip.yml
cluster_enable_start.yml packages.yml  stonith_timeout.yml  web_server.yml
cluster_setup.yml      services.yml   user_password.yml
```

To complete the play, you can use the `cluster_auth.yml`, `cluster_setup.yml`, `cluster_enable_start.yml`, `stonith_timeout.yml`, `stonith_resources.yml`, `web_server.yml`, and `web_ip.yml` task files.

You must adapt the `cluster_auth.yml`, `cluster_setup.yml`, and `web_ip.yml` examples to the requirements. The other task files do not need any change.

3.3. Edit the `cluster.yml` file and complete the second play. Your second play should look like this:

```

...output omitted...
- name: Deploying a Red Hat High Availability cluster
  hosts: nodeb
  become: yes
  gather_facts: no
  vars_files:
    - passwords.yml
  vars:
    ha_cluster_name: cluster2
    ha_nodes:
      - nodeb.private.example.com
      - nodec.private.example.com
      - noded.private.example.com
    stonith_timeout: 180
    web_ip: 172.25.250.80
    web_mask: 24

  tasks:
    - name: Ensuring the cluster nodes are authenticated
      command:
        cmd: "pcs host auth -u hacluster -p {{ ha_password }}"

```

```
    {{ ha_nodes | join(' ') }}"
register: auth_cluster
changed_when: "'Authorized' in auth_cluster['stdout']"

- name: Checking the cluster configuration
  stat:
    path: /etc/corosync/corosync.conf
  register: cluster_config

- name: Ensuring the cluster exists
  command:
    cmd: "pcs cluster setup {{ ha_cluster_name }} {{ ha_nodes | join(' ') }}"
  register: create_cluster
  changed_when: "'successfully set up' in create_cluster['stdout']"
  when: not cluster_config['stat']['exists']

- name: Ensuring the cluster is enabled
  command:
    cmd: pcs cluster enable --all
  register: enable_cluster
  changed_when: "'Cluster Enabled' in enable_cluster['stdout']"

- name: Ensuring the cluster is started
  command:
    cmd: pcs cluster start --all --request-timeout=180 --wait=180
  register: start_cluster
  changed_when: "'Starting Cluster' in start_cluster['stdout']"

- name: Checking if the global STONITH timeout is set
  command:
    cmd: pcs property show stonith-timeout
  register: stonith
  changed_when: false

- name: Ensuring the global STONITH timeout is set
  command:
    cmd: "pcs property set stonith-timeout={{ stonith_timeout }}s"
  when: "('{stonith-timeout: ' + stonith_timeout|string + 's}')"
        not in stonith['stdout']"

- name: Ensuring the STONITH resources exist
  include_tasks: create_ipmi.yml
  loop:
    - id: fence_nodeb
      node: nodeb.private.example.com
      ip: 192.168.0.102
      # ipmi_login and ipmi_password are defined in the Vault protected
      # passwords.yml file.
      login: "{{ ipmi_login }}"
      password: "{{ ipmi_password }}"
    - id: fence_nodec
      node: nodec.private.example.com
      ip: 192.168.0.103
      login: "{{ ipmi_login }}"
      password: "{{ ipmi_password }}"
```

```
- id: fence_noded
  node: noded.private.example.com
  ip: 192.168.0.104
  login: "{{ ipmi_login }}"
  password: "{{ ipmi_password }}"

- name: Collecting the existing resources
  command:
    cmd: pcs resource config
  changed_when: false
  register: resources

- name: Ensuring the server resource exists
  command:
    cmd: pcs resource create server apache --group=web --wait=60
  when: "'server' not in resources['stdout']"

- name: Ensuring the ip resource exists
  command:
    cmd: "pcs resource create ip IPAddr2 ip={{ web_ip }}
          cidr_netmask={{ web_mask }} --group=web --wait=60"
  when: "'ip' not in resources['stdout']"
...

```

The exercise preparation command has deployed an already completed playbook in the /home/student/labs/auto-review/.solutions/cluster.yml file. You can compare your work with that file.

- 3.4. Verify that you do not have any syntax errors in your Ansible Playbook. Because the passwords.yml file is encrypted, use the --ask-vault-pass option to specify the Vault password. Use redhat for the password.

```
[student@workstation auto-review]$ ansible-playbook --ask-vault-pass \
> --syntax-check cluster.yml
Vault password: redhat

playbook: cluster.yml
```

If the command reports errors, then fix them before running the playbook. You can compare your work with the already completed /home/student/labs/auto-review/.solutions/cluster.yml file.

- 3.5. Run the playbook. Use the --ask-vault-pass option to specify the Vault password. The password is redhat.

```
[student@workstation auto-review]$ ansible-playbook --ask-vault-pass cluster.yml
Vault password: redhat
...output omitted...
```

If there are errors, then make any necessary edits, run the reset.yml playbook to clean up the machines, and then execute the web\_server.yml and cluster.yml playbooks again.

- 3.6. Confirm that you can access the web content through the 172.25.250.80 IP address.

```
[student@workstation auto-review]$ curl 172.25.250.80
Hello, world!
```

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade auto-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish auto-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- Use the `yum`, `firewalld`, `user`, and `service` Ansible modules to prepare the cluster nodes.
- Wrap the `pcs` command in Ansible tasks by using the `command` module.
- Add the `--wait` option to `pcs` commands so that the subcommand status is reported.
- Use the `run_once` Ansible directive to run the `pcs` command only on one cluster node.



## Chapter 7

# Managing Two-node Clusters

### Goal

Operate two-node clusters, identifying and avoiding issues that are specific to a two-node cluster configuration.

### Objectives

- Describe the special considerations to follow when configuring a two-node cluster.
- Configure a two-node high availability cluster.
- Configure a high availability cluster with a quorum device to help resolve quorum ties.

### Sections

- Planning Two-node Cluster Deployments (and Quiz)
- Configuring a Two-node Cluster (and Guided Exercise)
- Configuring and Managing a Quorum Device (and Guided Exercise)

### Lab

- Managing Two-node Clusters

# Planning Two-node Cluster Deployments

## Objectives

After completing this section, you should be able to describe the special considerations to follow when configuring a two-node cluster.

## Handling Quorum in Two-node Clusters

There are many reasons why you might want to run a cluster consisting of only two nodes rather than one with three or more nodes. However, two-node clusters require some special configuration considerations to ensure correct operation. Some of these are specific to two-node clusters, and others apply in general to clusters that have an even number of nodes.

## Avoiding a Quorum Tie in Even-node Clusters

Clusters with an even number of nodes, including two-node clusters, are susceptible to a scenario in which a *split-brain* situation can occur. This condition usually occurs when there is an issue with the cluster's private network that splits the cluster exactly in half. The nodes in each half are still working correctly, but they cannot communicate with each other. This is more likely to happen in a two-node cluster, because each node is exactly half of the cluster. When this occurs, each half of the cluster might consider that they are quorate and both try to start cluster resources.

To avoid this condition, Red Hat recommends that you use a *quorum device*. A quorum device acts as a third node and therefore allows standard quorum rules to apply. Ideally, one half of the cluster should fence the other half and take over all resources, and a quorum device helps mediate this process. Another section later in this chapter describes quorum devices in more detail.

## Avoiding Issues Specific to Two-node Clusters

The following list presents some of the most common issues with two-node clusters.

### 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. This would come down to two votes in a two-node cluster, meaning the two nodes must always be up for the cluster to remain quorate.

When you deploy a two-node cluster, Pacemaker automatically sets the `two_node` flag of `votequorum`. In that special mode, the cluster artificially sets the quorum to 1. This mode allows one node to fail, with the other node remaining a quorate cluster all by itself.

### Fence races

In the case of a network failure between the two nodes, both nodes maintain quorum by themselves. Under that condition, administrators can run into a phenomenon known as a *fence race*.

A fence race occurs when the two nodes cannot communicate. However, they can still fence each other, either because you use fabric fencing or because the fencing devices are on a different network from the cluster communication. The two nodes initiate fencing simultaneously, resulting in both nodes going down and, therefore, in the cluster's complete failure.

To prevent fence races, you can set a delay on one of the fencing devices. Another option available with Red Hat Enterprise Linux 8.3 and later is to prioritize the node running most of the resources. With that configuration, the cluster first fences the node with the fewest number of resources.

### Fence loops

When the node that the cluster has fenced restarts, it tries to rejoin the cluster. If the communication issue between the two nodes persists, then the node forms a cluster all by itself and then fences the other node before taking over the resources. That second node restarts, constitutes a cluster by itself, and then, in turn, fences the initial node.

To prevent that infinite fence loop, the cluster automatically sets the `wait_for_all` flag when it sets the `two_node` flag of `votequorum`. The `wait_for_all` flag forces the cluster to wait for all nodes to be up at the same time before starting any cluster activity. With that flag, the node that the cluster has fenced does not start its cluster activity if it cannot reach the other node and therefore does not initiate fencing.

If you want to start the cluster with only one node because the other node is in maintenance, for example, then use the `pcs quorum unblock` command.

Details on how to configure the remediations for these issues are covered in subsequent sections of this chapter.



### References

`votequorum(5)` man page

**Knowledgebase: "Delaying Fencing in a Two Node Cluster to Prevent Fence Races or "Fence Death" Scenarios"**

<https://access.redhat.com/solutions/54829>

## ► Quiz

---

# Planning Two-node Cluster Deployments

Choose the correct answers to the following questions:

► 1. **What is the purpose of the two\_node mode for votequorum?**

- a. two\_node mode prevents fence races by delaying fencing for one node.
- b. two\_node mode artificially sets the quorum to 1 so that a two-node cluster can operate with only one node.
- 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. **In which situation can a fence race occur?**

- a. A fence race occurs when power fencing completes before fabric fencing.
- b. A fence race occurs when multiple nodes attempt to fence the same node at the same time.
- c. A fence race occurs when two nodes simultaneously attempt to fence each other.
- d. A fence race occurs when fabric fencing completes before power fencing.

► 3. **What is a possible consequence of a fence race?**

- 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 kernel of the "loser" node can crash.
- d. Both nodes enter a reboot-then-fence cycle, disabling cluster operations.

## ► Solution

# Planning Two-node Cluster Deployments

Choose the correct answers to the following questions:

► 1. **What is the purpose of the two\_node mode for votequorum?**

- a. two\_node mode prevents fence races by delaying fencing for one node.
- b. two\_node mode artificially sets the quorum to 1 so that a two-node cluster can operate with only one node.
- 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. **In which situation can a fence race occur?**

- a. A fence race occurs when power fencing completes before fabric fencing.
- b. A fence race occurs when multiple nodes attempt to fence the same node at the same time.
- c. A fence race occurs when two nodes simultaneously attempt to fence each other.
- d. A fence race occurs when fabric fencing completes before power fencing.

► 3. **What is a possible consequence of a fence race?**

- 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 kernel of the "loser" node can crash.
- d. Both nodes enter a reboot-then-fence cycle, disabling cluster operations.

# Configuring a Two-node Cluster

## Objectives

After completing this section, you should be able to configure a two-node high availability cluster.

## Creating Two-node Clusters

When you create a two-node cluster with the `pcs cluster setup` command, the cluster automatically enables the special two-node mode. If you add extra nodes later, then the cluster automatically disables the mode.

Run the `pcs quorum status` command to confirm that the cluster is operating in two-node mode.

```
[root@node1 ~]# pcs quorum status
...output omitted...
Votequorum information
-----
Expected votes: 2
Highest expected: 2
Total votes: 2
Quorum: 1
Flags: 2Node Quorate WaitForAll
...output omitted...
```

Notice from the preceding output that the cluster also enables the `WaitForAll` flag when working in two-node mode.



### Note

The quorum section of the `/etc/corosync/corosync.conf` file also shows the `two_node` flag.

```
quorum {
    provider: corosync_votequorum
    two_node: 1
}
```

Notice, however, that the section does not report the `WaitForAll` flag even though it is on by default in two-node mode.

## Disabling the "Wait for All" Feature

When you start your two-node cluster, the cluster begins operating only when the two nodes have successfully established an initial cluster communication. The `wait_for_all` option in `votequorum` controls this behavior.

## Chapter 7 | Managing Two-node Clusters

If you disable that option, then the cluster can start as soon as one node is up; it does not have to wait for the second node to join. Remember, however, that this option prevents fence loops. By disabling it, the cluster might become vulnerable to those fence loops when the cluster communication fails.

To disable the `wait_for_all` option when creating a new cluster, use the `quorum wait_for_all=0` option with the `pcs cluster setup` command:

```
[root@node1 ~]# pcs cluster setup mycluster node1.example.com node2.example.com \
> quorum wait_for_all=0
```

To disable the `wait_for_all` option on an existing cluster, use the following procedure:

- Stop the running cluster.

```
[root@node1 ~]# pcs cluster stop --all
```

- Run the `pcs quorum update` command to disable the `wait_for_all` option.

```
[root@node1 ~]# pcs quorum update wait_for_all=0
```

- Start the cluster on all nodes.

```
[root@node1 ~]# pcs cluster start --all
```

## Configuring Delayed Fencing

To avoid having both nodes fence each other simultaneously in the event of a network failure, you should configure delayed fencing. There are two ways to do this, but one of them requires that you are running Red Hat Enterprise Linux 8.3 or later.

- You can configure the cluster to delay fencing one of your nodes so that it always wins a fence race with the other node.
- If you are running RHEL 8.3 or later, you can assign a priority to each of your cluster resources, and the cluster will fence the node running the resources with the lowest total priority first.

## Configuring Delayed Fencing in Red Hat Enterprise Linux 8.2 and Earlier

To implement this approach, configure one of the two nodes for delayed fencing. Create one of the fence devices with the `pcmk_delay_base=Ns` option, where  $N$  is a time-out in seconds.

When the cluster calls that fence device, it logs the beginning of the fencing operation immediately but then waits for  $N$  seconds before actually starting the fencing operation. At the same time, the second node also initiates fencing. Because there is no time-out on that second device, the fencing operation starts with no delay, and the device fences the first node immediately. The second node wins the fence race.

In other words, the node for which you specify a delay on its device always wins the fence race. The node without a delay set always loses, and the fencing operation restarts it.

## Chapter 7 | Managing Two-node Clusters

The delay you set should be long enough for the other device to complete the fencing operation. Some fence devices might take a long time to operate. If the delay is too short, then it does not prevent fence races.

### Creating a Delayed Fence Device

To create a fence device with delayed fencing, add the `pcmk_delay_base=Ns` option to the command you use to create the STONITH resource.

For example, to create a fence device named `fence_node1_delayed`, targeting the `node1` machine, and with a delay of 30 seconds, use the following command:

```
[root@node1 ~]# pcs stonith create fence_node1_delayed fence_ipmilan \
> ip=10.12.0.8 username=admin password=s3cr3t lanplus=1 \
> pcmk_host_list=node1.example.com \
> pcmk_delay_base=30s
```

Create the fence device for the second node without the `pcmk_delay_base` option. With this configuration, the `node1` machine always wins the fence race because the cluster causes the other node wait for 30 seconds before allowing it to fence `node1`, giving `node1` an opportunity to fence `node2` first.

That configuration does not prevent the cluster from fencing the `node1` machine. If the cluster communication network is up, and the `node1` machine fails for another reason, then the cluster can still fence the `node1` machine, but with a 30-second delay.

### Updating an Existing Fence Device for Delayed Fencing

You can update an existing fence device for delayed fencing by using the `pcs stonith update` command.

For example, to update the `fence_mynode1` fence device to use a 25-second delay, use the following command:

```
[root@node1 ~]# pcs stonith update fence_mynode1 pcmk_delay_base=25s
```

### Configuring Delayed Fencing for Shared Fence Devices

A shared fence device is a device that can fence multiple nodes. Instead of having one STONITH resource per node, you create a single STONITH resource that targets all nodes.

Setting the `pcmk_delay_base` parameter with this kind of device cannot prevent fence races because that same delay would apply for both nodes.

For shared fence devices, use the `pcmk_delay_max=Ns` option instead. With that option, the cluster adds a random delay, between 0 and  $N$  seconds, before initiating a fence operation. If the two nodes request fencing simultaneously, then they get a different delay, and therefore, one of the nodes wins the fence race.

The following example creates a shared fence device for an APC by Schneider Electric (formerly American Power Conversion Corporation) network power switch. The cluster uses the same STONITH resource for fencing both the `node1` and `node2` machines but adds a random delay from 0 - 40 seconds.

```
[root@node1 ~]# pcs stonith create fence_apcA_node1 fence_apc \
> ip=10.12.0.7 username=apc password=pAssw0rD \
> pcmk_host_map="node1.example.com:2;node2.example.com:7" \
> pcmk_delay_max=40s
```

## Configuring Delayed Fencing in Red Hat Enterprise Linux 8.3 and Later

You can still use the method from the preceding paragraphs with RHEL 8.3 and later. In addition, a new method is available that does not require any configuration changes to your fencing devices. With this new method, in a split-brain situation the cluster selects the node to fence first based on the total priority value assigned to the resources running on each node.

To configure this, you must first assign a priority to each resource running on your cluster. The cluster usually uses each resource's priority to organize the resources on the cluster nodes, and to sacrifice resources with the lowest priority when not all resources can be active. The new fencing method also uses the resources' priorities to decide which node to fence.

### Setting Resource Priorities

You can assign a priority value when you create a resource. The following example creates a floating IP resource with a priority of 100. Higher numbers mean higher priorities.

```
[root@node1 ~]# pcs resource create httpIP IPAddr2 ip=10.42.0.80 cidr_netmask=24 \
> meta priority=100
```

You can also set or change the priority of existing resources with the `pcs resource meta` command.

```
[root@node1 ~]# pcs resource meta httpIP priority=50
```

If you do not want to assign a priority to every resource explicitly, then you can define a default priority. The following example sets a default priority of 1. That priority applies to all the resources that do not have an explicit priority set.

```
[root@node1 ~]# pcs resource defaults update priority=1
```

### Enabling Fencing Based on Priorities

After each resource has been assigned a priority value, activate the delayed fencing method based on the resources' priorities by setting the `priority-fencing-delay` cluster property.

```
[root@node1 ~]# pcs property set priority-fencing-delay=40s
```

With that configuration, and if all the resources have the same priority, then in case of a split-brain condition, the node with the fewest resources waits 40 seconds before initiating fencing. The other node does not wait and wins the fence race.



## References

votequorum(5) man page

**Knowledgebase: "Delaying Fencing in a Two Node Cluster to Prevent Fence Races or "Fence Death" Scenarios"**

<https://access.redhat.com/solutions/54829>

**Knowledgebase: "How do I delay fencing to prevent fence races when using a shared stonith device in a two-node cluster?"**

<https://access.redhat.com/solutions/3565071>

For more information, refer to the *Configuring resource meta options* section in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#proc\\_configuring-resource-meta-options-configuring-cluster-resources](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#proc_configuring-resource-meta-options-configuring-cluster-resources)

For more information, refer to the *Pacemaker cluster properties* chapter in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_controlling-cluster-behavior-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_controlling-cluster-behavior-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Configuring a Two-node Cluster

In this exercise, you will create a two-node cluster.

## Outcomes

You should be able to configure a two-node cluster with delayed fencing.

## Before You Begin

As the student user on the workstation machine, use the `lab start twonode-conf` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start twonode-conf
```

## Instructions

- 1. Prepare both your `nodea` and `nodeb` machines to become cluster nodes. To do so, allow the cluster software to communicate through the firewall, install the cluster packages, set the password of the `hacluster` user to `redhat`, and then enable and start the `pcsd` service.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. In a separate terminal, connect to `nodeb` and become the `root` user.

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

- 1.3. On both nodes, allow cluster communications to pass through the firewall.

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

**Chapter 7 |** Managing Two-node Clusters

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

- 1.4. On both nodes, install the `pcs` and `fence-agents-ipmilan` packages.

```
[root@nodea ~]# yum install pcs fence-agents-ipmilan
```

```
[root@nodeb ~]# yum install pcs fence-agents-ipmilan
```

- 1.5. On both nodes, change the password of the `hacluster` system user to `redhat`.

```
[root@nodea ~]# passwd hacluster  
Changing password for user hacluster.  
New password: redhat  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password: redhat  
passwd: all authentication tokens updated successfully.
```

```
[root@nodeb ~]# passwd hacluster  
Changing password for user hacluster.  
New password: redhat  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password: redhat  
passwd: all authentication tokens updated successfully.
```

- 1.6. On both nodes, enable and start the `pcsd` service.

```
[root@nodea ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

```
[root@nodeb ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

- ▶ 2. Create a two-node cluster called `twonodes` that uses your `nodea.private.example.com` and `nodeb.private.example.com` machines. Start and enable the cluster on both nodes, confirm that the cluster is active, and then verify that the cluster has set the `2Node` and `WaitForAll` flags.
- 2.1. From your `nodea` machine, authenticate your `nodea.private.example.com` and `nodeb.private.example.com` machines.

**Chapter 7 |** Managing Two-node Clusters

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

2.2. Create the cluster.

```
[root@nodea ~]# pcs cluster setup twonodes \
> nodea.private.example.com nodeb.private.example.com
...output omitted...
Cluster has been successfully set up.
```

2.3. Start the cluster and enable automatic activation of the services.

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

2.4. Confirm that the cluster is active. You might have to run the `pcs status` command several times because it takes up to a minute for the nodes to be on line.

```
[root@nodea ~]# pcs status
...output omitted...
Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com ]
...output omitted...
```

2.5. Use the `pcs quorum status` command to verify that the cluster has automatically set the `2Node` and `WaitForAll` flags.

```
[root@nodea ~]# pcs quorum status
...output omitted...
Votequorum information
-----
Expected votes: 2
Highest expected: 2
Total votes: 2
Quorum: 1
Flags: 2Node Quorate WaitForAll
...output omitted...
```

Notice that the cluster sets the quorum to 1.

- 3. Set the cluster STONITH time-out to 180 seconds to allow enough time for the fence operations to complete. To prevent fence races, add a fencing delay of 190 seconds and set the default priority for resources to 1.

## Chapter 7 | Managing Two-node Clusters

You can copy and paste the following commands from the /root/twonode-conf/commands.txt file.

- 3.1. Set the cluster STONITH time-out to 180 seconds.

```
[root@nodea ~]# pcs property set stonith-timeout=180s  
[root@nodea ~]#
```

- 3.2. Set the priority-fencing-delay cluster property to 190 seconds.

```
[root@nodea ~]# pcs property set priority-fencing-delay=190s  
[root@nodea ~]#
```

- 3.3. Confirm that the stonith-timeout and priority-fencing-delay properties are set.

```
[root@nodea ~]# pcs property  
Cluster Properties:  
cluster-infrastructure: corosync  
cluster-name: twonodes  
dc-version: 2.0.4-6.el8-2deceaa3ae  
have-watchdog: false  
priority-fencing-delay: 190s  
stonith-timeout: 180s
```

- 3.4. Set the resource default priority to 1.

```
[root@nodea ~]# pcs resource defaults update priority=1  
Warning: Defaults do not apply to resources which override them with their own  
defined values
```

- 3.5. Confirm that the resource default priority is 1.

```
[root@nodea ~]# pcs resource defaults  
Meta Attrs: rsc_defaults-meta_attributes  
priority=1
```

- ▶ 4. Add two fence devices, one for each node, by using the fence\_ipmilan fencing agent. The IP addresses of the Baseboard Management Controller (BMC) devices are 192.168.0.101 for the nodea machine and 192.168.0.102 for the nodeb machine. Use admin for the user name and password for the password for both devices. Allow a time-out of 180 seconds for the BMC devices to respond.

You can copy and paste the following commands from the /root/twonode-conf/commands.txt file.

- 4.1. Create the fence resource for the nodea machine.

```
[root@nodea ~]# pcs stonith create fence_nodea fence_ipmilan \  
> pcmk_host_list=nodea.private.example.com ip=192.168.0.101 username=admin \  
> password=password lanplus=1 power_timeout=180  
[root@nodea ~]#
```

## 4.2. Create the fence resource for the nodeb machine.

```
[root@nodea ~]# pcs stonith create fence_nodeb fence_ipmilan \
> pcmk_host_list=nodeb.private.example.com ip=192.168.0.102 username=admin \
> password=password lanplus=1 power_timeout=180
[root@nodea ~]#
```

- 5. To test your configuration, create a test resource and then disable cluster communications. Observe the cluster behavior.

5.1. Create a IPAddr2 resource called webip that uses the 172.25.250.80/24 address.

```
[root@nodea ~]# pcs resource create webip IPAddr2 ip=172.25.250.80 cidr_netmask=24
Assumed agent name 'ocf:heartbeat:IPAddr2' (deduced from 'IPAddr2')
```

5.2. Use the pcs status command to identify the node currently running the webip resource.

```
[root@nodea ~]# pcs status
...output omitted...
Full List of Resources:
* fence_nodea (stonith:fence_ipmilan):     Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan):     Started nodeb.private.example.com
* webip          (ocf::heartbeat:IPAddr2):   Started nodea.private.example.com
...output omitted...
```

The preceding output shows that the resource is running on the nodea machine. On your system, the resource might be running on the nodeb machine.

5.3. Disrupt the cluster network communications. To do so, on the node where the resource is **not** running, use the /root/twonode-conf/firewall-block.sh script that the exercise preparation command has deployed for you.

```
[root@nodeb ~]# /root/twonode-conf/firewall-block.sh
success
success
```

5.4. On the node where the resource is running, use the pcs status command to monitor the cluster. Because the webip resource is running on that node, the node has the highest resource priority and initiates fencing of the other node without delay. Simultaneously, the other node, which is not running any resources, has to wait 190 seconds before starting fencing. As a consequence, it loses the race, and it is restarted.

```
[root@nodea pacemaker]# pcs status
...output omitted...
Node List:
* Node nodeb.private.example.com: UNCLEAN (offline)
* Online: [ nodea.private.example.com ]

Full List of Resources:
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
```

**Chapter 7 |** Managing Two-node Clusters

```
* fence_nodeb (stonith:fence_ipmilan): Starting [ nodeb.private.example.com
nodea.private.example.com ]
* webip          (ocf::heartbeat:IPAddr2): Started nodea.private.example.com

Pending Fencing Actions:
* reboot of nodeb.private.example.com pending: client=pacemaker-controld.1170,
origin=nodea.private.example.com
...output omitted...
```

In the preceding output, notice that the nodeb machine is unresponsive. The webip resource is still active and a fence action is in progress.

## Finish

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish twonode-conf
```

This concludes the section.

# Configuring and Managing a Quorum Device

## Objectives

After completing this section, you should be able to configure a high availability cluster with a quorum device to help resolve quorum ties.

## Improving Quorum Operation with a Quorum Device

You can use a quorum device to act as an arbitrator to help resolve certain quorum issues for your cluster. Two-node clusters in particular, and clusters with an even number of nodes in general, should use a quorum device to help determine which nodes should survive in a split-brain situation. Clusters with an odd number of nodes do not have those issues.

By adding a quorum device, your two-node cluster operates like a three-node cluster. Using a quorum device means you do not need to make the configuration changes that two-node clusters usually require.

A quorum device has two parts:

- The QNet daemon runs on a separate machine, which must not be a cluster node. The daemon acts as a third-party arbitrator for your two-node cluster.
- The QDevice daemon runs on all cluster nodes. It communicates with the QNet daemon and adds a vote to the quorum when that communication is successful.

The QNet daemon should not run on the cluster network but preferably on a distinct physical network. This way, the nodes can still access the QNet daemon under a split-brain condition due to the cluster network being down.

You do not have to deploy one QNet daemon per cluster. Multiple clusters can share the same QNet daemon.



### Warning

The quorum device is a critical device in your cluster, because it acts as a quorum arbitrator. Thus, you must ensure that the cluster nodes can always access the QNet daemon to avoid issues in the quorum calculation which can lead to cluster failure.

## Describing Quorum Device Algorithms

Quorum devices support two different algorithms. Those algorithms specify how the QNet daemon provides its vote to the cluster nodes in a split-brain scenario.

### `ffsplit`

With the `fifty/fifty split (ffsplit)` algorithm, the QNet daemon provides one vote to the partition with more nodes.

If the number of nodes is the same in both partitions, then the QNet daemon gives its vote to the partition hosting the node with the lowest node ID. You can retrieve your node IDs by using the `pcs quorum status` command or the `/etc/corosync/corosync.conf` file.

### lms

With the `Last man standing` (`lms`) algorithm, the cluster can survive more node failures.

If only one node continues communicating with the QNet daemon, then that node gets the extra vote. This mechanism allows the cluster to stay quorate when only one node remains active. Under a split-brain condition, the algorithm behaves like the `ffsplit` algorithm.

For two-node clusters, the two algorithms are very similar. For those clusters, the QNet daemon gives its vote to the node with the lowest node ID in a split-brain scenario.

The QDevice daemon accepts additional parameters for more advanced configurations. Refer to the `corosync-qdevice(8)` man page for more details.

## Deploying a Quorum Device

Deploying a quorum device for a two-node cluster requires preparing the QNet daemon on a new system and then declaring the quorum device to the cluster.

### Preparing the Quorum Device System

You must deploy the QNet daemon on a system that is not a cluster node. On that system, install the `pcs` and the `corosync-qnetd` packages. The `pcs` package provides the tools to authenticate the machine with the cluster nodes and configure the quorum device.

```
[root@host ~]# yum install pcs corosync-qnetd
```

You prepare the `pcsd` daemon as you would on regular cluster nodes:

```
[root@host ~]# passwd hacluster
[root@host ~]# systemctl enable --now pcasd
[root@host ~]# firewall-cmd --permanent --add-service=high-availability
[root@host ~]# firewall-cmd --reload
```

Use the `pcs` command to configure and start the QNet daemon. The `model` option indicates the arbitrator module to use. Only the `net` arbitrator is available. The `--enable` option configures the QNet daemon to start when the system starts. The `--start` option starts the daemon immediately.

```
[root@host ~]# pcs qdevice setup model net --enable --start
```

### Adding the Quorum Device to the Cluster

On all the cluster nodes, install the `corosync-qdevice` package, which provides the QDevice daemon.

```
[root@node ~]# yum install corosync-qdevice
```

From one node, authenticate the `pcsd` daemon running on the quorum device machine.

```
[root@node1 ~]# pcs host auth host.example.net
```

Use the `pcs` command from one of the cluster node to declare the quorum device.

```
[root@node1 ~]# pcs quorum device add model net host=host.example.net \
> algorithm=ffssplit
```

## Inspecting the Quorum Device Status

From a cluster node, the `pcs quorum status` command reports the quorum details and the quorum device status.

```
[root@node1 ~]# pcs quorum status
Quorum information
-----
Date:           Tue May 25 18:12:18 2021
Quorum provider: corosync_votequorum
Nodes:          2
Node ID:        1
Ring ID:        1.9
Quorate:       Yes

Votequorum information
-----
Expected votes: 3
Highest expected: 3
Total votes:    3
Quorum:         2
Flags:          Quorate Qdevice

Membership information
-----
  Nodeid   Votes   Qdevice Name
    1        1     A,V,NMW node1.example.com (local)
    2        1     A,V,NMW node2.example.com
    0        1           Qdevice
```

In the preceding output, the membership information section reports the quorum device status for each node. The command displays the status as a comma-separated string:

- The first value indicates whether the communication with the quorum device is active (A) or not (NA).
- The second value is only relevant under a split-brain condition. It shows which partition gets the QNet daemon vote. A V value indicates that the corresponding node gets the vote. An NV value indicates the node that does not get the vote.
- The third value, NMW (Not Master Wins) or MW (Master Wins), is only relevant for a specific advanced configuration, which is beyond the scope of this course.

On the machine running the QNet daemon, use the `pcs qdevice status net` command to list the clusters and the nodes that use the quorum device.

```
[root@host ~]# pcs qdevice status net
QNetd address:      *:5403
TLS:               Supported (client certificate required)
Connected clients: 2
Connected clusters: 1
```

```
Cluster "mycluster":  
    Algorithm:  Fifty-Fifty split  
    Tie-breaker: Node with lowest node ID  
    Node ID 1:  
        Client address:  ::ffff:10.25.0.10:38210  
        Configured node list:  1, 2  
        Membership node list:  1, 2  
        Vote:      ACK (ACK)  
    Node ID 2:  
        Client address:  ::ffff:10.25.0.11:58048  
        Configured node list:  1, 2  
        Membership node list:  1, 2  
        Vote:      No change (ACK)
```



## References

[corosync-qdevice\(8\) man page](#)

**Knowledgebase: "Does Red Hat support highly available or redundant corosync-qnetd servers?"**

<https://access.redhat.com/solutions/5828321>

For more information, refer to the *Quorum devices* section in the *Configuring and managing high availability clusters* Guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-quorum-devices-configuring-cluster-quorum](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-quorum-devices-configuring-cluster-quorum)

## ► Guided Exercise

# Configuring and Managing a Quorum Device

In this exercise, you will deploy a quorum device on a two-node cluster to better manage split-brain situations.

## Outcomes

You should be able to deploy a quorum device on a new machine, and then configure a two-node cluster to use that quorum device.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start twonode-qdevice
```

This command deploys a two-node cluster on the `nodea` and `nodeb` machines.

## Instructions

- 1. From the `nodea` machine, confirm that the cluster operates in a two-node configuration.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 1.2. Use the `pcs quorum status` command to confirm that the `2Node` flag is set. Notice that the expected votes is 2 and that the quorum is 1.

```
[root@nodea ~]# pcs quorum status
Quorum information
-----
Date: Thu Feb 4 08:09:02 2021
Quorum provider: corosync_votequorum
Nodes: 2
Node ID: 1
Ring ID: 1.9
Quorate: Yes

Votequorum information
-----
Expected votes: 2
```

```
Highest expected: 2
Total votes:      2
Quorum:        1
Flags:           2Node Quorate WaitForAll

Membership information
-----
  Nodeid   Votes   Qdevice Name
    1       1       NR nodea.private.example.com (local)
    2       1       NR nodeb.private.example.com
```

- 2. Prepare the quorum device on the noded machine. In this exercise, the noded machine is not part of the cluster, which is composed of the nodea and nodeb machines. It is an independent machine that you use to host the quorum device.

- 2.1. In a separate terminal, connect to the noded machine and become the **root** user.

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

- 2.2. Install the **pcs** and **corosync-qnetd** packages, change the password of the **hacluster** system user to **redhat**, and then enable and start the **pcsd** service.

```
[root@noded ~]# yum install pcs corosync-qnetd
[root@noded ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
[root@noded ~]# systemctl enable --now pcasd
Created symlink /etc/systemd/system/multi-user.target.wants/pcasd.service → /usr/
lib/systemd/system/pcasd.service.
```

- 2.3. Configure, start, and enable the quorum device.

```
[root@noded ~]# pcs qdevice setup model net --enable --start
Quorum device 'net' initialized
quorum device enabled
Starting quorum device...
quorum device started
```

- 2.4. Allow cluster communications to pass through the firewall.

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

- 2.5. Confirm that the quorum device is ready. Notice that no cluster is using it yet, and that no cluster nodes are connected.

```
[root@noded ~]# pcs qdevice status net
QNetd address: *:5403
TLS: Supported (client certificate required)
Connected clients: 0
Connected clusters: 0
```

- 3. Install the `corosync-qdevice` package on the cluster nodes, `nodea` and `nodeb`. That package provides the cluster component that connects to the `corosync-qnet` daemon running on the `noded` machine.

```
[root@nodea ~]# yum install corosync-qdevice
```

```
[root@nodeb ~]# yum install corosync-qdevice
```

- 4. From the `nodea` machine, configure the cluster to use the quorum device. Red Hat recommends that the cluster nodes not use the cluster network to access the quorum device. In this exercise, the nodes use the `private.example.com` (192.168.0.0/24) network for cluster communications. To access the quorum device, use instead the `lab.example.com` (172.25.250.0/24) network.

- 4.1. Authenticate the `noded.lab.example.com` machine.

```
[root@nodea ~]# pcs host auth noded.lab.example.com
Username: hacluster
Password: redhat
noded.lab.example.com: Authorized
```

- 4.2. Declare the quorum device at `noded.lab.example.com`. Set the `algorithm` parameter to `ffsplit`. You can copy and paste the following command from the `/root/twonode-qdevice/commands.txt` file.

```
[root@nodea ~]# pcs quorum device add model net host=noded.lab.example.com \
> algorithm=ffsplit
Setting up qdevice certificates on nodes...
nodea.private.example.com: Succeeded
nodeb.private.example.com: Succeeded
Enabling corosync-qdevice...
nodea.private.example.com: corosync-qdevice enabled
nodeb.private.example.com: corosync-qdevice enabled
Sending updated corosync.conf to nodes...
nodea.private.example.com: Succeeded
nodeb.private.example.com: Succeeded
nodea.private.example.com: Corosync configuration reloaded
Starting corosync-qdevice...
nodea.private.example.com: corosync-qdevice started
nodeb.private.example.com: corosync-qdevice started
```

- 4.3. Use the `pcs quorum config` command to verify your configuration.

```
[root@nodea ~]# pcs quorum config
Options:
Device:
  votes: 1
  Model: net
    algorithm: ffsplit
  host: noded.lab.example.com
```

#### 4.4. Inspect the quorum status.

```
[root@nodea ~]# pcs quorum status
Quorum information
-----
Date:           Thu Feb  4 08:22:58 2021
Quorum provider: corosync_votequorum
Nodes:          2
Node ID:        1
Ring ID:        1.9
Quorate:       Yes

Votequorum information
-----
Expected votes: 3
Highest expected: 3
Total votes:      3
Quorum:         2
Flags:           Quorate Qdevice

Membership information
-----
  Nodeid   Votes   Qdevice Name
    1        1     A,V,NMW nodea.private.example.com (local)
    2        1     A,V,NMW nodeb.private.example.com
    0        1     Qdevice
```

Notice that the new configuration removes the 2Node flag and instead adds the Qdevice flag. The expected votes is 3, the quorum is 2, and the command shows a new member of type Qdevice. The cluster now behaves like a three-node cluster.

#### ► 5. From the noded machine, inspect the status of the quorum device.

```
[root@noded ~]# pcs qdevice status net
QNetd address:          *:5403
TLS:                   Supported (client certificate required)
Connected clients:    2
Connected clusters:   1
Cluster "twonodes":
  Algorithm:            Fifty-Fifty split
  Tie-breaker:          Node with lowest node ID
  Node ID 2:
    Client address:     ::ffff:172.25.250.11:53338
    Configured node list: 1, 2
    Membership node list: 1, 2
```

```
Vote:          ACK (ACK)
Node ID 1:
Client address:   ::ffff:172.25.250.10:54036
Configured node list: 1, 2
Membership node list: 1, 2
Vote:          No change (ACK)
```

Notice that the cluster and its two nodes are connected to the quorum device.

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish twonode-qdevice
```

This concludes the section.

## ▶ Lab

# Managing Two-node Clusters

In this lab, you will configure a two-node cluster that has a quorum device.

## Outcomes

You should be able to:

- Create a two-node cluster.
- Create and prepare a quorum device.
- Add a quorum device to a two-node cluster.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start twonode-review
```

## Instructions

1. Prepare both your `nodec` and `noded` machines to become cluster nodes. To do so, allow the cluster software to communicate through the firewall, install the cluster packages, set the password of the `hacluster` user to `redhat`, and then enable and start the required service. In addition, install the `fence-agents-ipmilan` package on both nodes.  
On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.
2. Create a two-node cluster called `cluster2` that uses the `nodec.private.example.com` and `noded.private.example.com` nodes. Start and enable the cluster on both nodes.
3. Set the cluster STONITH time-out to 180 seconds to allow enough time for the fence operations to complete. Add two fence devices, one for each node, by using the `fence_ipmilan` fencing agent. To add those two devices, run the `/root/twonode-review/stonith.sh` script on one node. The exercise preparation command has deployed that script to simplify your work.
4. Prepare a quorum device on the `nodea` machine. To do so, first allow the cluster software to communicate through the firewall, and then install the cluster packages, set the password of the `hacluster` user to `redhat`, and then enable and start the required service as well as the quorum device.
5. Install the `corosync` quorum device package on the `nodec` and `noded` cluster nodes. When done, configure your cluster to use the quorum device running on the `nodea` machine. To access that quorum device, use the `san01.example.com` (192.168.1.0/24) network. The DNS name of the `nodea` machine on that network is `nodea.san01.example.com` (192.168.1.10). Use the `ffsplit` algorithm when declaring the quorum device.

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade twonode-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish twonode-review
```

This concludes the section.

## ► Solution

# Managing Two-node Clusters

In this lab, you will configure a two-node cluster that has a quorum device.

### Outcomes

You should be able to:

- Create a two-node cluster.
- Create and prepare a quorum device.
- Add a quorum device to a two-node cluster.

### Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start twonode-review
```

### Instructions

1. Prepare both your `nodec` and `noded` machines to become cluster nodes. To do so, allow the cluster software to communicate through the firewall, install the cluster packages, set the password of the `hacluster` user to `redhat`, and then enable and start the required service. In addition, install the `fence-agents-ipmilan` package on both nodes.

On your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

- 1.1. In two separate terminals, connect to the `nodec` and `noded` machines, and then become the `root` user.

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo -i
[sudo] password for student: student
[root@nodec ~]#
```

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

- 1.2. On both nodes, allow cluster communications to pass through the firewall.

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

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

- 1.3. On both nodes, install the `pcs` and `fence-agents-ipmilan` packages.

```
[root@nodec ~]# yum install pcs fence-agents-ipmilan
```

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

- 1.4. On both nodes, change the password of the `hacluster` system user to `redhat`.

```
[root@nodec ~]# passwd hacluster  
Changing password for user hacluster.  
New password: redhat  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password: redhat  
passwd: all authentication tokens updated successfully.
```

```
[root@noded ~]# passwd hacluster  
Changing password for user hacluster.  
New password: redhat  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password: redhat  
passwd: all authentication tokens updated successfully.
```

- 1.5. On both nodes, enable and start the `pcsd` service.

```
[root@nodec ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

```
[root@noded ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/  
lib/systemd/system/pcsd.service.
```

2. Create a two-node cluster called `cluster2` that uses the `nodec.private.example.com` and `noded.private.example.com` nodes. Start and enable the cluster on both nodes.
  - 2.1. From your `nodec` machine, authenticate your `nodec.private.example.com` and `noded.private.example.com` machines for `pcs`.

```
[root@nodec ~]# pcs host auth nodec.private.example.com noded.private.example.com
Username: hacluster
Password: redhat
nodec.private.example.com: Authorized
noded.private.example.com: Authorized
```

2.2. Create the cluster.

```
[root@nodec ~]# pcs cluster setup cluster2 \
> nodec.private.example.com noded.private.example.com
...output omitted...
Cluster has been successfully set up.
```

2.3. Start the cluster and enable automatic activation of the services.

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

2.4. Confirm that the cluster is active. You might have to run the `pcs status` command several times because it takes up to a minute for the nodes to be on line.

```
[root@nodec ~]# pcs status
...output omitted...
Node List:
* Online: [ nodec.private.example.com noded.private.example.com ]
...output omitted...
```

3. Set the cluster STONITH time-out to 180 seconds to allow enough time for the fence operations to complete. Add two fence devices, one for each node, by using the `fence_ipmilib` fencing agent. To add those two devices, run the `/root/twonode-review/stonith.sh` script on one node. The exercise preparation command has deployed that script to simplify your work.

3.1. From the `nodec` machine, set the cluster STONITH time-out to 180 seconds.

```
[root@nodec ~]# pcs property set stonith-timeout=180s
[root@nodec ~]#
```

3.2. Run the `/root/twonode-review/stonith.sh` script on the `nodec` machine.

```
[root@nodec ~]# /root/twonode-review/stonith.sh
[root@nodec ~]#
```

3.3. Use the `pcs status` command to verify the two fence devices.

```
[root@nodec ~]# pcs status
...output omitted...
Full List of Resources:
  * fence_nodec (stonith:fence_ipmilan):      Started nodec.private.example.com
  * fence_noded (stonith:fence_ipmilan):      Started noded.private.example.com
...output omitted...
```

4. Prepare a quorum device on the nodea machine. To do so, first allow the cluster software to communicate through the firewall, and then install the cluster packages, set the password of the hacluster user to redhat, and then enable and start the required service as well as the quorum device.

- 4.1. In a separate terminal, connect to the nodea machine and become the root user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 4.2. Allow cluster communications to pass through the firewall.

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

- 4.3. Install the pcs and corosync-qnetd packages.

```
[root@nodea ~]# yum install pcs corosync-qnetd
```

- 4.4. Change the password of the hacluster system user to redhat.

```
[root@nodea ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

- 4.5. Enable and start the pcasd service.

```
[root@nodea ~]# systemctl enable --now pcasd
Created symlink /etc/systemd/system/multi-user.target.wants/pcasd.service → /usr/
lib/systemd/system/pcasd.service.
```

- 4.6. Configure, start, and enable the quorum device.

```
[root@nodea ~]# pcs qdevice setup model net --enable --start
Quorum device 'net' initialized
quorum device enabled
Starting quorum device...
quorum device started
```

4.7. Confirm that the quorum device is ready.

```
[root@nodea ~]# pcs qdevice status net
QNetd address: *:5403
TLS: Supported (client certificate required)
Connected clients: 0
Connected clusters: 0
```

5. Install the corosync quorum device package on the nodec and noded cluster nodes. When done, configure your cluster to use the quorum device running on the nodea machine. To access that quorum device, use the san01.example.com (192.168.1.0/24) network. The DNS name of the nodea machine on that network is nodea.san01.example.com (192.168.1.10). Use the ffsplit algorithm when declaring the quorum device.

5.1. Install the corosync-qdevice package on the cluster nodes.

```
[root@nodec ~]# yum install corosync-qdevice
```

```
[root@noded ~]# yum install corosync-qdevice
```

- 5.2. From the nodec machine, authenticate the nodea.san01.example.com machine for pcs.

```
[root@nodec ~]# pcs host auth nodea.san01.example.com
Username: hacluster
Password: redhat
nodea.san01.example.com: Authorized
```

- 5.3. Add the quorum device at nodea.san01.example.com. Set the algorithm parameter to ffsplit.

```
[root@nodec ~]# pcs quorum device add model net host=nodea.san01.example.com \
> algorithm=ffsplit
...output omitted...
```

- 5.4. Use the pcs quorum config command to verify your configuration.

```
[root@nodec ~]# pcs quorum config
Options:
Device:
  votes: 1
  Model: net
    algorithm: ffsplit
  host: nodea.san01.example.com
```

## 5.5. Inspect the quorum status.

```
[root@nodec ~]# pcs quorum status
Quorum information
-----
Date:           Thu Feb 25 04:32:03 2021
Quorum provider: corosync_votequorum
Nodes:          2
Node ID:        1
Ring ID:        1.9
Quorate:       Yes

Votequorum information
-----
Expected votes: 3
Highest expected: 3
Total votes:      3
Quorum:          2
Flags:           Quorate Qdevice

Membership information
-----
  Nodeid   Votes   Qdevice Name
    1         1     A,V,NMW nodec.private.example.com (local)
    2         1     A,V,NMW noded.private.example.com
    0         1           Qdevice
```

## Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade twonode-review
```

## Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish twonode-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- Two-node clusters are susceptible to fence loops, fence races, and split-brain situations.
- The `wait_for_all` flag with the two-node mode avoids fence loops.
- The two-node mode sets the quorum to 1 so that a two-node cluster can operate with only one node and automatically enables the `wait_for_all` option.
- Delayed fencing prevents fence races.
- There are two ways to configure delayed fencing: by configuring one node's STONITH device or, (on RHEL 8.3 or later), by using resource priorities.
- A quorum device acts as a tie breaker for split-brain situations.
- You should configure a quorum device for any cluster with an even number of nodes.

## Chapter 8

# Accessing iSCSI Storage

### Goal

Configure iSCSI initiators on your servers to access block-based storage devices that network storage arrays or Ceph storage clusters provide.

### Objectives

- Explain what iSCSI is, define key iSCSI terms, and describe how it is useful for cluster storage.
- Configure an iSCSI initiator to access a network-based block device, format a new iSCSI device with a file system, configure it for use at boot, and safely discontinue the use of an existing iSCSI block device.

### Sections

- Describing iSCSI Concepts (and Quiz)
- Accessing iSCSI Storage (and Guided Exercise)

### Lab

- Accessing iSCSI Storage

# Describing iSCSI Concepts

## Objectives

After completing this section, you should be able to explain what iSCSI is, define key iSCSI terms, and describe how it is useful for cluster storage.

## Describing iSCSI

iSCSI is a TCP/IP based protocol for sending SCSI commands over IP based networks. It allows you to connect block-based storage devices that act like normal hard drives or solid-state drives from a server to clients over the network. As a *Storage Area Network (SAN)* protocol, iSCSI helps you consolidate your storage devices in your local or remote data centers.

Because performance can degrade due to bandwidth congestion on shared networks, you should implement iSCSI on cabling that is independent of standard LAN traffic. Typically, iSCSI is paired with dedicated 10 Gigabit Ethernet or better networking to maximize performance.

SAN traffic is typically unencrypted to maximize performance. The cabling from the physical server to the storage is normally enclosed in the data center, and is ideally not directly connected to LAN networks. For WAN security, iSCSI administrators may use *IPsec*, a protocol suite for securing IP network traffic.

In the context of high availability clustering, administrators often use iSCSI to provide access to a shared storage from all cluster nodes. The application that the cluster manages can then store its data on that shared storage. Because the application now has access to its data from any node, the cluster can quickly restart it on a different node without having to copy, replicate, or move the data locally to the new node.

## Defining iSCSI Key Terms

The iSCSI protocol works in a client/server configuration. Client systems configure *initiator* software to send SCSI commands to remote server storage *targets*. On client systems, iSCSI targets appear as local SCSI disks, just like devices connected with SCSI cabling or to a Fibre Channel switched fabric.

Some terms you will hear when working with iSCSI include:

### Initiator

An iSCSI client, typically available as software. You can also buy hardware initiators in the form of iSCSI *Host Bus Adapters (HBAs)*. Initiators must have unique names (see IQN).

### Target

An iSCSI storage resource on an iSCSI server. Targets must have unique names (see IQN). Each target provides one or more block devices, or *logical units*. In most cases a target provides exactly one device. A single server can provide multiple targets.

### IQN (iSCSI Qualified Name)

A unique worldwide name used to identify both initiators and targets. IQNs have the following format:

```
iqn.YYYY-MM.com.reversed.domain:name_string
```

**YYYY-MM**

The first year and full month that you had control over your DNS domain name. (For example, 2020-06 for June 2020.) That date helps ensure that even if the organization that controls the domain name changes, the IQN you used is still unique.

**com.reversed.domain**

Your domain name reversed. For example, `www.example.com` becomes `com.example.www`.

**name\_string**

A unique string in your naming space (`YYYY-MM.com.reversed.domain`) that identifies a particular target you manage. This is sometimes omitted if the reversed domain name is a host name specific to a single server with exactly one target.

**Portal**

Each target has one or more portals, which are IP address and port pairs that initiators may use to reach the target.

**LUN (Logical Unit Number)**

A LUN represents a block device provided by the target. Each target provides one or more LUNs. (Therefore, a single target can provide multiple storage devices.)

**ACL (Access Control List)**

An access restriction that uses the initiator's IQN to validate its access permissions.

**TPG (Target Portal Group)**

A TPG is a complete configuration for a target, including portals, LUNs, and ACLs. Almost all targets use one TPG, but advanced configurations might occasionally define multiple TPGs.

**References**

For more information, refer to the *Getting started with iSCSI* chapter in the *Managing Storage Devices Guide* at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/managing\\_storage\\_devices/index#getting-started-with-iscsi\\_managing-storage-devices](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/managing_storage_devices/index#getting-started-with-iscsi_managing-storage-devices)

## ► Quiz

# Describing iSCSI Concepts

Match the items below to their counterparts in the table.

- IQN
- LUN
- initiator
- portal
- target

Description	Term
A unique name that identifies an individual iSCSI target or initiator.	
A storage resource provided by an iSCSI server.	
A number used to specify an individually addressable block device provided by a storage resource on an iSCSI server.	
An iSCSI client implemented in either software or hardware.	
An IP address on an initiator or an IP address and a TCP port on a target that may be used for an iSCSI connection.	

## ► Solution

# Describing iSCSI Concepts

Match the items below to their counterparts in the table.

Description	Term
A unique name that identifies an individual iSCSI target or initiator.	IQN
A storage resource provided by an iSCSI server.	target
A number used to specify an individually addressable block device provided by a storage resource on an iSCSI server.	LUN
An iSCSI client implemented in either software or hardware.	initiator
An IP address on an initiator or an IP address and a TCP port on a target that may be used for an iSCSI connection.	portal

# Accessing iSCSI Storage

## Objectives

After completing this section, you should be able to configure an iSCSI initiator to access a network-based block device, format a new iSCSI device with a file system, configure it for use at boot, and be able to safely discontinue the use of an existing iSCSI block device.

## Configuring iSCSI Initiators

In Red Hat Enterprise Linux, an iSCSI initiator is typically implemented in software. Using a software-based iSCSI initiator requires connecting to an existing Ethernet network of sufficient bandwidth to carry the storage traffic.



### Note

As an alternative to the software implementation covered in this section, you can use hardware initiators that include the required protocols in a dedicated *host bus adapter (HBA)*. These can offload Ethernet, TCP, and iSCSI processing to hardware, which reduce the load on other system resources.

## Preparing the System

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.

```
[root@host ~]# yum install iscsi-initiator-utils
```

As an iSCSI initiator, the client requires its own unique iSCSI Qualified Name (IQN). During the installation of *iscsi-initiator-utils*, the package generates a unique IQN that uses Red Hat's DNS domain and then stores that IQN into the */etc/iscsi/initiatorname.iscsi* file. Administrators typically change the IQN in that file to their DNS domain in addition to an appropriate client name string.

The */etc/iscsi/iscsid.conf* file contains default settings for targets that you connect. These settings include iSCSI timeouts, retry parameters, and user names and passwords for authentication.

The package installation automatically configures the *iscsi* and *iscsid* services so that the initiator automatically reconnects to any already discovered targets when the system boots. Whenever you modify the initiator's configuration files, restart the *iscsid* service.

## Connecting iSCSI Targets

Before you can connect to and use a remote device, you first must discover the target. The discovery process stores target information and settings in the */var/lib/iscsi/nodes* directory, by using defaults from */etc/iscsi/iscsid.conf*.

You perform the discovery of remote targets with the following command:

```
[root@host ~]# iscsiadm --mode discovery --type st --portal portal_ip[:port]
192.168.0.10:3260,1 iqn.2021-04.com.example:disk1
192.168.0.10:3260,1 iqn.2021-04.com.example:disk2
192.168.0.10:3260,1 iqn.2021-04.com.example:target1
```

The `portal_ip` parameter is the IP address of the target portal. If you do not specify the `port` parameter, then it uses the default port which is 3260. The command returns the IQNs of the available targets.

To use one of the listed targets, log in to it by using the following command:

```
[root@host ~]# iscsiadm --mode node --targetname iqn.2021-04.com.example:disk1 \
> --portal portal_ip[:port] --login
```

The Red Hat Customer Portal Labs provide the iSCSI helper tool at <https://access.redhat.com/labs/iscsihelper/>. You can generate a script to configure your iSCSI target or initiator by using that online tool.



### Note

If discovery is successful, but the initiator experiences an issue logging in to the discovered target, then the problem is likely related to access control or authentication.

You grant access to iSCSI targets on the iSCSI server by specifying the initiators' IQNs that should be allowed access. On the client, you configure the initiator IQN in `/etc/iscsi/initiatorname.iscsi`. A mismatch between the IQN on the client and the one on the server can result in a target login failure because the wrong ACL might be used.

If the issue is due to an incorrect initiator IQN on the client, fix the IQN in the client's `/etc/iscsi/initiatorname.iscsi` file, and then restart the `iscsid` service in order for the change to take effect.

At that point, the system detects a new SCSI block device as if it is a locally attached hard drive. You can identify that new device with the `iscsiadm --mode session --print 3` command, which displays information about the current iSCSI login session at print level 3.

```
[root@host ~]# iscsiadm --mode session --print 3
iSCSI Transport Class version 2.0-870
version 6.2.0.877-0
Target: iqn.2021-04.com.example:disk1 (non-flash)
Current Portal: 192.168.0.10:3260,1
...output omitted...
Attached scsi disk sdc State: running
```

As an alternative, you can look at the output of the `dmesg`, `tail /var/log/messages`, or `ls -l /dev/disk/by-path/*iscsi*` commands.

That login process is persistent across reboots. Therefore, the block device is automatically available after boot.

## Formatting iSCSI Devices

If the discovered block device already has partitions, file systems, or LVM volumes, then you can access that data with the usual commands, such as `mount`. You can inspect the device with the `lsblk --fs` command to discover such structures.

```
[root@host ~]# lsblk --fs
NAME      FSTYPE LABEL UUID                                     MOUNTPOINT
sda
├─sda1    xfs     f7614c41-2835-4125-bb13-50772dc2f30c  /boot
├─sda2    swap    78a5f1eb-a593-4db3-b15a-f49042e182a2  [SWAP]
└─sda3    xfs     ac2790c4-cfbb-409e-88a9-fe8c963fc00   /
sdb      LVM2_member  4ALe1y-oEXe-XLer-PjYW-n5ud-Ryko-XZZ2PM
├─dbdata-log ext4   bd566c1f-01a1-404d-8f01-9495004f327c  /data/db/log
└─dbdata-data ext4   5640b4ae-5868-4bf5-8715-51fc18d8656b  /data/db/data
sdc
```

In the preceding output, the `/dev/sda` device has three partitions, the `/dev/sdb` device is a physical volume for LVM, and the `/dev/sdc` device has no data.



### Warning

Do not allow multiple initiators to mount the same file system from the same target at the same time.

Unlike local block devices, several remote initiators can discover iSCSI block devices through the network. Local file systems, such as ext4 or XFS, do not support concurrent mounting from multiple systems. Doing so can result in file system corruption or inconsistencies when attempting to read data.

If you need to allow simultaneous access to an iSCSI-based block device from multiple systems, use a clustered file system such as GFS2 that can permit this access safely.

If the disk is empty, you can format it, create partitions, or use it as an LVM physical volume.

When persistently mounting a file system on an iSCSI target in `/etc/fstab`, make sure to adhere to the following recommendations:

- Use the `lsblk` command with the `--fs` option to determine the file system UUID and then mount the file system by using that UUID. Do not use the device name (`/dev/sd*`) because it can change from boot to boot. The name of the device depends on the order in which iSCSI devices respond over the network. If you use the device name in `/etc/fstab`, and that name changes after a reboot, then the system might mount the device under the wrong mount point.
- Use the `_netdev` mount option in `/etc/fstab`. Because iSCSI relies on the network to access the remote device, this option ensures that the system does not attempt to mount the file system until the network and the initiator are up.

## Disconnecting from Targets

To discontinue use of an iSCSI target, use the following steps:

- Make sure none of the devices provided by the target are in use. For example, unmount the file system.

**Chapter 8 |** Accessing iSCSI Storage

- Remove all persistent references to the target from places such as /etc/fstab.
- Log off from the iSCSI target.

```
[root@host ~]# iscsiadm --mode node --targetname iqn.2021-04.com.example:disk1 \
> --portal 192.168.0.10:3260 --logout
```

- Delete the local record of the iSCSI target so that the initiator does not automatically log in to the target during boot.

```
[root@host ~]# iscsiadm --mode node --targetname iqn.2021-04.com.example:disk1 \
> --portal 192.168.0.10:3260 --op delete
```



## References

iscsiadm(8) man page

/usr/share/doc/iscsi-initiator-utils/README

**Knowledgebase: "What are the advantages and disadvantages to using partitioning on LUNs, either directly or with LVM in between?"**

<https://access.redhat.com/solutions/163853>

For more information, refer to the *Creating an iSCSI initiator* section in the *Managing Storage Devices Guide* at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/managing\\_storage\\_devices/index#creating-an-iscsi-initiator\\_adding-an-iscsi-target](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/managing_storage_devices/index#creating-an-iscsi-initiator_adding-an-iscsi-target)

For more information, refer to the *Getting started with XFS* and *Mounting file systems* sections in the *Managing File Systems Guide* at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/managing\\_file\\_systems/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/managing_file_systems/index)

The Red Hat iSCSI Helper Tool

<https://access.redhat.com/labsinfo/iscsihelper/>

## ► Guided Exercise

# Accessing iSCSI Storage

In this exercise, you will configure an iSCSI initiator to access a network-based block device, use it, and then safely discontinue using it.

## Outcomes

You should be able to:

- Configure the iSCSI initiator on a client system to access a target.
- Format and use the discovered block device.
- Cleanly disconnect an initiator from a target.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start iscsi-initiator
```

This command configures an iSCSI target on the storage machine. You will connect to that target by using an initiator on nodea.

## Instructions

- 1. Connect to nodea and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 2. Install the `iscsi-initiator-utils` package.

```
[root@nodea ~]# yum install iscsi-initiator-utils
```

- 3. Set the IQN for the client initiator to `iqn.2021-04.com.example:nodea`.

- 3.1. Edit the `/etc/iscsi/initiatorname.iscsi` file and update the `InitiatorName` parameter. For your convenience, you can copy and paste the IQN from the `/root/iscsi-initiator/iscsi.txt` file.

```
InitiatorName=iqn.2021-04.com.example:nodea
```

**Chapter 8 |** Accessing iSCSI Storage

- 3.2. Restart the `iscsid` service to take the new IQN into account. If the service is not yet running, then the `systemctl restart iscsid` command starts it.

```
[root@nodea ~]# systemctl restart iscsid
```

- 4. Discover and log in to the configured target from storage.

- 4.1. Discover the iSCSI target from storage (192.168.1.15).

```
[root@nodea ~]# iscsiadm --mode discovery --type st --portal 192.168.1.15
192.168.1.15:3260,1 iqn.2021-04.com.example:storage
```

Notice the target IQN in the preceding output.

- 4.2. Log in to the iSCSI target.

```
[root@nodea ~]# iscsiadm --mode node \
> --targetname iqn.2021-04.com.example:storage --portal 192.168.1.15 --login
Logging in to [iface: default, target: iqn.2021-04.com.example:storage, portal:
192.168.1.15,3260]
Login to [iface: default, target: iqn.2021-04.com.example:storage, portal:
192.168.1.15,3260] successful.
```

If the command fails, then confirm that you correctly set the initiator IQN in the `/etc/iscsi/initiatorname.iscsi` file to `iqn.2021-04.com.example:nodea`. If not, then fix the issue in the file, restart the `iscsid` service, and try the preceding command again.

- 4.3. Identify the newly available block device.

```
[root@nodea ~]# iscsiadm --mode session --print 3
iSCSI Transport Class version 2.0-870
version 6.2.0.877-0
Target: iqn.2021-04.com.example:storage (non-flash)
Current Portal: 192.168.1.15:3260,1
...output omitted...
Attached scsi disk sda State: running
```

- 5. Confirm that the new block device has no partition and no file system. Format the device as XFS and then configure it to persistently mount on `/data` at boot.

- 5.1. Use the `lsblk` command with the `--fs` option to list the partitions and file systems on the block devices.

```
[root@nodea ~]# lsblk --fs
NAME   FSTYPE LABEL UUID                                     MOUNTPOINT
sda
vda
└─vda1
└─vda2 vfat            F537-0F4F                         /boot/efi
└─vda3 xfs             root   fe1e8b67-e41b-44b8-bcfe-e0ec966784ac  /
```

Notice that the `sda` block device has no partition and no file system.

**Chapter 8 |** Accessing iSCSI Storage

- 5.2. Use the `mkfs` command to apply an XFS file system to `/dev/sda`.

```
[root@nodea ~]# mkfs -t xfs /dev/sda  
...output omitted...
```

Notice that in this exercise, you do not create a partition on the device. Instead, you use the whole disk for the file system. Red Hat supports both methods.

- 5.3. Create the `/data` mount point.

```
[root@nodea ~]# mkdir /data
```

- 5.4. Use the `lsblk` command with the `--fs` option to discover the UUID of the `/dev/sda` file system.

```
[root@nodea ~]# lsblk --fs /dev/sda  
NAME FSTYPE LABEL UUID MOUNTPOINT  
sda xfs f691e715-ccaa-40a2-8156-921e3df59f7f
```

The UUID in the preceding output might be different on your system.

- 5.5. Edit the `/etc/fstab` file and add a line for the new file system. Remember to specify the `_netdev` option.

```
UUID=f691e715-ccaa-40a2-8156-921e3df59f7f /data xfs _netdev 0 0
```

- 5.6. Mount the file system and then verify that the mount is successful.

```
[root@nodea ~]# mount /data  
[root@nodea ~]# df /data  
Filesystem 1K-blocks Used Available Use% Mounted on  
/dev/sda 10475520 106088 10369432 2% /data
```

- 6. Cleanup your work. To do so, unmount the file system, log out of the iSCSI target, and then delete all records of the target.

- 6.1. Unmount the `/data` file system.

```
[root@nodea ~]# umount /data
```

- 6.2. Edit the `/etc/fstab` file and remove the `/data` entry.

- 6.3. Log off from the iSCSI target.

```
[root@nodea ~]# iscsiadm --mode node \  
> --targetname iqn.2021-04.com.example:storage --portal 192.168.1.15 --logout  
Logging out of session [sid: 1, target: iqn.2021-04.com.example:storage, portal:  
192.168.1.15,3260]  
Logout of [sid: 1, target: iqn.2021-04.com.example:storage, portal:  
192.168.1.15,3260] successful.
```

- 6.4. Delete the local record so that the initiator does not automatically log in to the target during boot.

```
[root@nodea ~]# iscsiadadm --mode node \
> --targetname iqn.2021-04.com.example:storage --portal 192.168.1.15 --op delete
```

## Finish

On the workstation machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish iscsi-initiator
```

This concludes the section.

## ► Lab

# Accessing iSCSI Storage

In this lab, you will configure an iSCSI initiator to access an iSCSI-based block device, format it with a file system, and mount it.

## Outcomes

You should be able to:

- Configure an iSCSI initiator and access a target.
- Format and mount a shared block device.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start iscsi-review
```

This command configures an iSCSI target on the storage machine.

## Instructions

- Configure an iSCSI initiator on the noded machine to access the target that the storage machine provides according to the following table.

### iSCSI Initiator Configuration

Parameter	Value
Initiator IQN	iqn.2021-04.com.example:noded
Target IQN	iqn.2021-04.com.example:store1
Portal address	192.168.2.15, port 3260

For your convenience, you can copy and paste those parameters from the `/root/iscsi-review/iscsi.txt` file on the noded machine.

The password for the `student` user on the noded machine is `student`, and the `root` password is `redhat`.

- On the noded machine, create an `ext4` file system on the discovered device, and then mount it under the `/iscsidisk` mount point. Make sure that the file system automatically mounts when the system starts.

## Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade iscsi-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish iscsi-review
```

This concludes the section.

## ► Solution

# Accessing iSCSI Storage

In this lab, you will configure an iSCSI initiator to access an iSCSI-based block device, format it with a file system, and mount it.

## Outcomes

You should be able to:

- Configure an iSCSI initiator and access a target.
- Format and mount a shared block device.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start iscsi-review
```

This command configures an iSCSI target on the storage machine.

## Instructions

- Configure an iSCSI initiator on the noded machine to access the target that the storage machine provides according to the following table.

### iSCSI Initiator Configuration

Parameter	Value
Initiator IQN	iqn.2021-04.com.example:noded
Target IQN	iqn.2021-04.com.example:store1
Portal address	192.168.2.15, port 3260

For your convenience, you can copy and paste those parameters from the `/root/iscsi-review/iscsi.txt` file on the noded machine.

The password for the student user on the noded machine is `student`, and the root password is `redhat`.

- Connect to noded and become the root user.

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

**Chapter 8 |** Accessing iSCSI Storage

- 1.2. Install the *iscsi-initiator-utils* package.

```
[root@noded ~]# yum install iscsi-initiator-utils
```

- 1.3. Edit the */etc/iscsi/initiatorname.iscsi* file and set the IQN for the client initiator to *iqn.2021-04.com.example:noded*.

```
InitiatorName=iqn.2021-04.com.example:noded
```

- 1.4. Restart the *iscsid* service to take the new IQN into account.

```
[root@noded ~]# systemctl restart iscsid
```

- 1.5. Discover the iSCSI target from the storage machine (192.168.2.15).

```
[root@noded ~]# iscsidadm --mode discovery --type st --portal 192.168.2.15  
192.168.2.15:3260,1 iqn.2021-04.com.example:store1
```

- 1.6. Log in to the iSCSI target.

```
[root@noded ~]# iscsidadm --mode node \  
> --targetname iqn.2021-04.com.example:store1 --portal 192.168.2.15 --login  
Logging in to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.2.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.2.15,3260] successful.
```

If the command fails, then confirm that you correctly set the initiator IQN in the */etc/iscsi/initiatorname.iscsi* file to *iqn.2021-04.com.example:noded*. If not, then fix the issue in the file, restart the *iscsid* service, and try the preceding command again.

2. On the noded machine, create an *ext4* file system on the discovered device, and then mount it under the */iscsidisk* mount point. Make sure that the file system automatically mounts when the system starts.

- 2.1. Identify the newly available block device.

```
[root@noded ~]# iscsidadm --mode session --print 3  
...output omitted...  
Attached scsi disk sda State: running
```

- 2.2. Use the *mkfs.ext4* command to apply an *ext4* file system to */dev/sda*.

```
[root@noded ~]# mkfs.ext4 /dev/sda  
...output omitted...
```

- 2.3. Create the */iscsidisk* mount point.

```
[root@noded ~]# mkdir /iscsidisk  
[root@noded ~]#
```

**Chapter 8 |** Accessing iSCSI Storage

- 2.4. Use the `lsblk` command with the `--fs` option to discover the UUID of the `/dev/sda` file system.

```
[root@noded ~]# lsblk --fs /dev/sda
NAME FSTYPE LABEL UUID                                     MOUNTPOINT
sda   ext4      e5e67c89-a97e-42cf-ba02-13519751cfb1
```

The UUID in the preceding output might be different on your system.

- 2.5. Edit the `/etc/fstab` file and add a line for the new file system. Remember to specify the `_netdev` option.

```
UUID=e5e67c89-a97e-42cf-ba02-13519751cfb1  /iscsidisk  ext4  _netdev  0  0
```

- 2.6. Mount the file system and then verify that the mount is successful.

```
[root@noded ~]# mount /iscsidisk
[root@noded ~]# df /iscsidisk
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/sda        10255636  36888   9678076   1% /iscsidisk
```

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade iscsi-review
```

## Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish iscsi-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- Configuring an iSCSI client initiator requires installing the `iscsi-initiator-utils` package.
- The initiator IQN is defined in the `/etc/iscsi/initiatorname.iscsi` file.
- The `iscsid` service must be restarted whenever you modify the `/etc/iscsi/initiatorname.iscsi` file.
- The `iscsiadm` command can be used to manage iSCSI target discovery, logins, and logouts.



## Chapter 9

# Accessing Storage Devices Resiliently

### Goal

Configure resilient access to storage devices that have multiple access paths.

### Objectives

- Describe storage multipathing and its terminology, and explain when it should be used.
  - Configure resilient access to a storage device by using a multipath device.
  - Monitor, troubleshoot, and test multipath devices.
- 
- Describing Device-Mapper Multipath (and Quiz)
  - Configuring Resilient Storage Access (and Guided Exercise)
  - Testing Resilient Storage Access (and Guided Exercise)

### Sections

- Accessing Storage Devices Resiliently

### Lab

# Describing Device-Mapper Multipath

## Objectives

After completing this section, you should be able to describe storage multipathing and its terminology, and explain when it should be used.

## Describing Multipathing

A system might be able to access the same storage device through multiple different communication paths, whether those are using Fibre Channel, SAS, iSCSI, or some other technology. *Multipathing* allows you to configure a virtual device that can use any of these communication paths to access your storage.

If one path fails, then the system automatically switches to use one of the other paths instead. Alternatively, you might use multipathing to aggregate multiple storage paths as a group to provide more bandwidth and improve storage performance.

For example, the server in the following diagram has two Fibre Channel *Host Bus Adapters* (*HBAs*), each connected to separate Fibre Channel switches, which in turn are connected to separate controllers on the storage array.

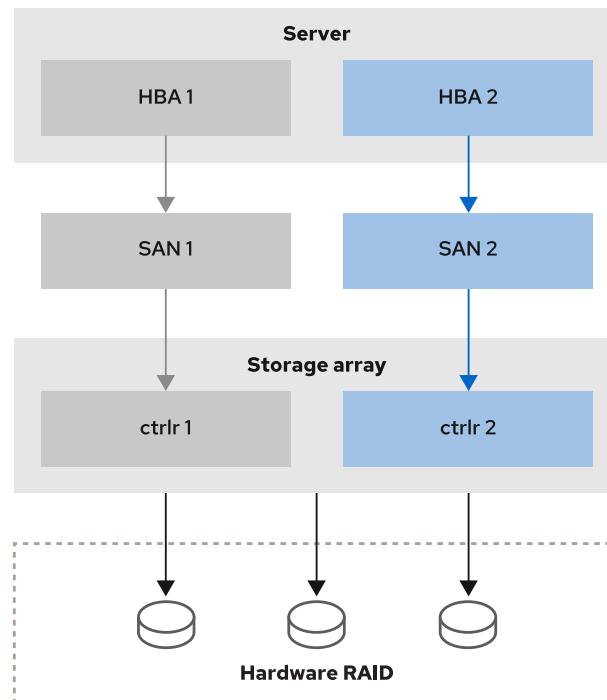


Figure 9.1: An Example Multipath Setup



### Important

Multipathing provides protection against a storage access path going down. If the storage itself becomes unavailable, even if the multiple paths are still working, then the system loses access to the storage.

Red Hat Enterprise Linux uses the `dm-multipath` subsystem to implement multipath support. The `dm_multipath` kernel module generates virtual devices. The `multipathd` daemon manages those devices and monitors the paths. You can use the `multipath` command-line tool to retrieve the status of the multipath devices.

The `device-mapper-multipath` RPM package provides the necessary binaries and daemons.

The system automatically detects which paths go to the same storage devices and combines them into groups based on settings in the `/etc/multipath.conf` file. Typically, only one group is active at a time. When a group fails, the multipath daemon switches storage traffic to a different group.

A group can consist of multiple paths. With that configuration, the system spreads the I/O traffic between the paths inside the group.

## Accessing Multipath Devices by Name

The kernel assigns a *World Wide Identifier (WWID)* to every multipath device. WWIDs are globally unique and unchanging. By default, the system sets the name of a multipath device to its WWID. The system also creates a device file for each WWID in the `/dev/mapper/` directory.

However, if you set the `user_friendly_names` option to `yes` in the `/etc/multipath.conf` file, then the system assigns a simple, persistent, and unique alias for the multipath device in the form `/dev/mapper/mpathN`, where `N` is a letter.

The system stores these mappings of WWIDs to device names in the `/etc/multipath/bindings` file. To ensure that the names are the same on all cluster nodes, deploy the same `/etc/multipath/bindings` file on all of them.

If the devices have partitions, then the system also creates device files for them. For example, the `/dev/mapper/mpathap1` device file represents the first partition of the `/dev/mapper/mpatha` device.



### Note

You can also set custom names for multipath devices. To set custom names, use the `alias` option in the `multipaths` section of the `/etc/multipath.conf` configuration file.



### Warning

The system also creates devices with names like `/dev/dm-N`. These devices are strictly for the system's internal use. Do not use them directly to access your storage.



## References

`multipath.conf(5)` man page

For more information, refer to the *Overview of Device Mapper Multipathing* and *Multipath devices* chapter in the *Configuring device mapper multipath* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_device\\_mapper\\_multipath/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_device_mapper_multipath/index)

## ► Quiz

# Describing Device-Mapper Multipath

Choose the correct answers to the following questions:

- ▶ 1. Which two 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. Which package provides the multipathing capabilities for Red Hat Enterprise Linux 8?
  - 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

# Describing Device-Mapper Multipath

Choose the correct answers to the following questions:

- ▶ 1. Which two 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. Which package provides the multipathing capabilities for Red Hat Enterprise Linux 8?
  - 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 Resilient Storage Access

## Objectives

After completing this section, you should be able to configure resilient access to a storage device by using a multipath device.

## Configuring Multipathing

To configure multipathing, first install the `device-mapper-multipath` RPM package.

```
[root@host ~]# yum install device-mapper-multipath
```

The package does not deploy the `/etc/multipath.conf` configuration file. Without that file, the `multipathd` service cannot start.

The Red Hat Customer Portal Labs provide the multipath helper tool at <https://access.redhat.com/labs/multipathhelper/>. You can generate your configuration file by using that online tool.

The other option is to use the `mpathconf` utility available on your system. If the `/etc/multipath.conf` file already exists, then the `mpathconf` command edits that file. Otherwise, the command creates the file.

To create a default configuration and then start the `multipathd` daemon, use the following command:

```
[root@host ~]# mpathconf --enable --with_multipathd y
```

The `mpathconf` command accepts additional options to set some of the most common configuration parameters. For example, the `--user_friendly_names` option sets the `user_friendly_names` parameter in the configuration file.



### Note

By default, the `mpathconf` command sets the `user_friendly_names` parameter to yes. With that configuration, the system creates multipath device files with names of the form `/dev/mapper/mpathN`.

Although this naming scheme can be useful if there is only one multipath device, it can become confusing with multiple devices. To disable those names, run the `mpathconf` command with the `--user_friendly_names n` option. With that configuration, the multipath devices have names based on their WWIDs.

If you want to tune the `/etc/multipath.conf` configuration file before starting the `multipathd` daemon, run the `mpathconf --enable` command without the `--with_multipathd y` option:

```
[root@host ~]# mpathconf --enable
```

In this case, after editing the configuration file, start the `multipathd` daemon with the `systemctl` command.

## Editing the Configuration File

The `/etc/multipath.conf` file organizes the configuration into sections:

### `blacklist {}`

During the discovery process, the system adds all the devices it can find to the multipath topology. The `blacklist` section lists the devices that the system should exclude from the topology.

### `blacklist_exceptions {}`

The `blacklist_exceptions` section defines which devices the system should include in the multipath topology, despite being listed in the `blacklist` sections.

### `defaults {}`

The `default` section defines the default settings for all the multipath devices unless explicitly overridden in the `devices` or `multipaths` section.

### `devices {}`

The `devices` section contains overrides for the `defaults` section for specific types of storage controllers. The system identifies the devices from their `vendor`, `product`, and `revision` keywords. Those keywords are regular expressions matching information from `sysfs`.

### `multipaths {}`

The `multipaths` section contains settings for specific multipath devices. This section overrides the parameters from the `defaults` and `devices` sections. The system identifies the multipath devices from their WWIDs.

To retrieve a parameter for a specific multipath device, the system looks for a matching device in the `multipaths` section first and then in the `devices` section. If the system cannot find a match in those sections, then it uses the parameter from the `defaults` section.

## Excluding Devices

Use an exclude list to prevent multipath from managing some devices, usually local disks or storage controllers with no redundant paths. Declare those devices in the `blacklist` section of the `/etc/multipath.conf` configuration file.



### Note

By default, the `mpathconf` command sets the `find_multipaths` parameter to yes in the `defaults` section. With that parameter, multipath automatically excludes all the devices that only have one path. This configuration prevents multipath from managing local disks or storage devices with no redundant paths. In that situation, you usually leave the `blacklist` section empty.

You can exclude a device based on its device file by using the `devnode` parameter or based on its WWID by using the `wwid` parameter. Both parameters accept a regular expression. The following example illustrates both approaches:

```
blacklist {
    devnode "^cciss"
    wwid 3600140562aeac25dc4c4eb5842574c7a
}
```

The system uses the device's serial number for the WWID. The system queries the udev database to get that information. That database maintains a list of all the system's devices with their attributes. You can use the `udevadm info` command to consult that database.

The following example runs the command with the device file as an argument to retrieve the WWID from the `ID_SERIAL` attribute.

```
[root@host ~]# udevadm info /dev/sda
...output omitted...
E: ID_SCSI_INQUIRY=1
E: ID_SCSI_SERIAL=62aeac25-dc4c-4eb5-8425-74c7a42b2ded
E: ID_SERIAL=3600140562aeac25dc4c4eb5842574c7a
E: ID_SERIAL_SHORT=600140562aeac25dc4c4eb5842574c7a
E: ID_TARGET_PORT=0
E: ID_TYPE=disk
...output omitted...
```

An alternative configuration consists of excluding all the devices by using the `blacklist` section and then allowing individual devices by using the `blacklist_exceptions` section. In that configuration, the `blacklist_exceptions` section acts as an inclusive list.

## Setting Default Parameters

Define default parameter values in the `defaults` section of the `/etc/multipath.conf` configuration file.

Use the `multipath -t` command to display the current configuration, including all the parameters not explicitly set in the configuration file. The `multipath.conf(5)` man page gives the default value of each parameter.

The following list describes some of the most useful settings.

- `path_selector`: This setting sets the algorithm that multipath uses for selecting the path inside a group to use for I/O. The default value is "service-time 0", which sends the next I/O request to the path that has the shortest estimated service time. The "round-robin 0" algorithm distributes I/O over all the paths in the group. The "queue-length 0" algorithm sends the next I/O request to the path with the shortest queue of outstanding requests.
- `path_grouping_policy`: This setting defines how the system combines the paths into groups. The default value is `failover`, which instructs multipath to put each path in a separate group. With that configuration, the system uses only one path for I/O and switches to the next path in case of failure. With the `multibus` policy, the system aggregates all possible paths into a single group. Multipath then distributes the I/O requests between the paths according to the `path_selector` algorithm. You must ensure that the storage controller supports active-active connections before setting the parameter to `multibus`.
- `path_checker`: This setting configures how the `multipathd` daemon verifies that a path is healthy. The default value is `tur`, which instructs the `multipathd` daemon to use Test Unit Ready (TUR) SCSI commands to retrieve the storage controller's status. Some path checkers only work with specific hardware, such as the `emc_clariion` checker for EMC VNX Storage

or the `hp_sw` checker for HPE MSA Storage. Because the path checker often depends on the storage device, you usually set the `path_checker` parameter in the `devices` section of the configuration file.

- `no_path_retry`: This setting controls multipath behavior when all the paths have failed. The default value is `fail`, which instructs multipath to immediately report the I/O error to higher layers. This parameter also accepts a number as the value to indicate the number of retries before reporting the error. With a value of `queue`, multipath queues all I/O requests until a path becomes active again. Under that condition, some processes might hang forever in an uninterruptible sleep state.
- `user_friendly_names`: This setting determines whether multipath devices get a name from their WWID (when set to `no`) or a name in the form `mpathN` (when set to `yes`).

## Configuring Multipathing for a Specific Storage Controller

For specific storage hardware, you can use the `devices` section to override the default parameters. In the `devices` section, you add individual `device` subsections to provide specific hardware settings.

Most common storage hardware already has their section defined in the built-in defaults. You can list the configuration of those storage controllers by using the `multipath -t` command. The following example lists some of the known devices.

```
[root@host ~]# multipath -t
...output omitted...
devices {
    device {
        vendor "NVME"
        product ".*"
        uid_attribute "ID_WWN"
        path_checker "none"
        retain_attached_hw_handler "no"
    }
    device {
        vendor "APPLE"
        product "Xserve RAID"
        path_grouping_policy "multibus"
    }
    device {
        vendor "3PARdata"
        product "VV"
        path_grouping_policy "group_by_prio"
        hardware_handler "1 alua"
        prio "alua"
        fallback "immediate"
        no_path_retry 18
        fast_io_fail_tmo 10
        dev_loss_tmo "infinity"
        vpd_vendor hp3par
    }
...output omitted...
```

## Chapter 9 | Accessing Storage Devices Resiliently

If your hardware is not listed, then you can add a section for it. The following is an example definition for a nonexistent piece of storage hardware. Select the device itself with the vendor, product, and revision parameters.

```
devices {
    device {
        vendor "MegaHyperSuperStorage"
        product "BAS"
        revision "513/B"
        no_path_retry 20
        path_grouping_policy multibus
        path_checker tur
    }
}
```

From a connected storage device, you can retrieve the values for vendor, product, and revision by using the `multipathd show paths` format "%d %s" command. You can also get that information from the `/sys/block/sda/device/vendor`, `/sys/block/sda/device/model`, and `/sys/block/sda/device/rev` files (for the sda device).

## Configuring a Specific Path

For specific multipath devices, you can use the `multipaths` section to override the parameters from the `defaults` and `devices` sections. One common use of the `multipaths` section is to define an alias for a multipath device. When you define an alias, the system uses that alias as the name of the device file in the `/dev/mapper/` directory, making it easier to distinguish between different multipath devices.

For example, the following configuration sets an alias of `clusterstorage` for the multipath device with a WWID of `3600140562aeac25dc4c4eb5842574c7a`. With that configuration, multipath creates the `/dev/mapper/clusterstorage` device file.

```
multipaths {
    multipath {
        wwid 3600140562aeac25dc4c4eb5842574c7a
        alias clusterstorage
    }
}
```

## Adding Partitions

To add a partition on a multipath device, use the following steps:

- Create the partition on the multipath device by using a partition editor, such as `parted /dev/mapper/mpatha`.
- Run the `udevadm settle` command. This command waits for the system to detect the new partition and to create the associated device file under the `/dev/mapper/` directory. It only returns when it is done.

## Removing a Multipath Device

After removing all paths for a multipath device, run the `multipath -f <multipath_device>` command to clear the multipath device.

If the `multipathd` daemon is not running and there are still device files for multipath devices, then flush all the multipath devices by running the `multipath -F` command. This command is useful when you are testing different configurations, and you want to remove remnants of old configurations.



## References

`mpathconf(8)`, `multipath.conf(5)`, and `udevadm(8)` man pages

For more information, refer to the *Setting up DM Multipath* and *Modifying the DM-Multipath configuration file* chapters in the *Configuring device mapper multipath* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_device\\_mapper\\_multipath/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_device_mapper_multipath/index)

The Red Hat Multipath Helper Tool

<https://access.redhat.com/labs/multipathhelper/>

## ► Guided Exercise

# Configuring Resilient Storage Access

In this exercise, you will configure resilient access to iSCSI-based storage by combining devices that represent two independent paths to the same storage into one multipath device.

## Outcomes

You should be able to configure a multipath device with an alias for redundant access to iSCSI storage.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start multipath-conf
```

This command configures the iSCSI initiator on `nodea`, `nodeb`, and `nodec` to access the target on the storage machine through the two storage networks, `san01.example.com` and `san02.example.com`. As a result, the storage is accessible on the nodes through the `/dev/sda` and `/dev/sdb` device nodes.

## Instructions

- 1. Install the `device-mapper-multipath` package on `nodea`, `nodeb`, and `nodec`.

- 1.1. In three separate terminals, connect to the machines and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

```
[student@workstation ~]$ ssh nodeb
...output omitted...
[student@nodeb ~]$ sudo -i
[sudo] password for student: student
[root@nodeb ~]#
```

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo -i
[sudo] password for student: student
[root@nodec ~]#
```

- 1.2. Install the *device-mapper-multipath* package on the three machines.

```
[root@nodea ~]# yum install device-mapper-multipath
```

```
[root@nodeb ~]# yum install device-mapper-multipath
```

```
[root@nodec ~]# yum install device-mapper-multipath
```

- 2. On the nodea machine, use the *mpathconf* command to create the multipath configuration file.

```
[root@nodea ~]# mpathconf --enable
[root@nodea ~]#
```

- 3. On nodea, retrieve the WWID of the iSCSI storage and then edit the configuration file to assign the *ClusterStorage* alias to the multipath device. This way, you will be able to use that alias to refer to the device instead of the more complex WWID.

- 3.1. Use the *udevadm info* command to determine the WWID of the iSCSI storage.

```
[root@nodea ~]# udevadm info /dev/sda | grep ID_SERIAL=
E: ID_SERIAL=36001405b7118eb213ff4bc792f144a04
```

On your system, the WWID is probably different.

- 3.2. Edit the */etc/multipath.conf* configuration file and then add the *multipaths* section at the end. In the following example, replace the WWID with the one you retrieved in the preceding step.

```
multipaths {
    multipath {
        wwid           36001405b7118eb213ff4bc792f144a04
        alias          ClusterStorage
    }
}
```

- 4. Copy the */etc/multipath.conf* configuration file from nodea to nodeb and nodec.

- 4.1. Use the *scp* command to copy the file to nodeb. The *root* password is *redhat*.

```
[root@nodea ~]# scp /etc/multipath.conf nodeb:/etc
The authenticity of host 'nodeb (172.25.250.11)' can't be established.
ECDSA key fingerprint is SHA256:WL9YdEHqYZwY45yU+mS9pU7bnS08wsNvxGZ1LegLU/Y.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'nodeb,172.25.250.11' (ECDSA) to the list of known
hosts.
root@nodeb's password: redhat
multipath.conf                                         100%   596   901.1KB/s   00:00
```

- 4.2. Copy the file to nodec.

```
[root@nodea ~]# scp /etc/multipath.conf nodec:/etc
The authenticity of host 'nodec (172.25.250.12)' can't be established.
ECDSA key fingerprint is SHA256:WL9YdEHqYZwY45yU+mS9pU7bnS08wsNxvGZ1LegLU/Y.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'nodec,172.25.250.12' (ECDSA) to the list of known
hosts.
root@nodec's password: redhat
multipath.conf                                              100%   596    924.5KB/s   00:00
```

- 5. Enable and start the `multipathd` service on the nodea, nodeb, and nodec machines.

```
[root@nodea ~]# systemctl enable --now multipathd
[root@nodea ~]#
```

```
[root@nodeb ~]# systemctl enable --now multipathd
[root@nodeb ~]#
```

```
[root@nodec ~]# systemctl enable --now multipathd
[root@nodec ~]#
```

- 6. Confirm that the multipath device is available on the three machines.

- 6.1. Use the `multipath` command to verify the configuration.

```
[root@nodea ~]# multipath -ll
ClusterStorage (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=active
|   - 2:0:0:0 sda 8:0  active ready running
`--- policy='service-time 0' prio=50 status=enabled
   - 3:0:0:0 sdb 8:16 active ready running
```

```
[root@nodeb ~]# multipath -ll
ClusterStorage (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=active
|   - 2:0:0:0 sda 8:0  active ready running
`--- policy='service-time 0' prio=50 status=enabled
   - 3:0:0:0 sdb 8:16 active ready running
```

```
[root@nodec ~]# multipath -ll
ClusterStorage (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=active
|   - 2:0:0:0 sda 8:0  active ready running
`--- policy='service-time 0' prio=50 status=enabled
   - 3:0:0:0 sdb 8:16 active ready running
```

6.2. Confirm that the new device node is available under the /dev/mapper/ directory.

```
[root@nodea ~]# ls /dev/mapper/ClusterStorage  
/dev/mapper/ClusterStorage
```

```
[root@nodeb ~]# ls /dev/mapper/ClusterStorage  
/dev/mapper/ClusterStorage
```

```
[root@nodec ~]# ls /dev/mapper/ClusterStorage  
/dev/mapper/ClusterStorage
```

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish multipath-conf
```

This concludes the section.

# Testing Resilient Storage Access

## Objectives

After completing this section, you should be able to monitor, troubleshoot, and test multipath devices.

## Monitoring the Multipath Devices

Use the `multipath` command to monitor the status of the multipath devices. With the `-l` option, the command shows a quick overview of the multipath topology. With the `-ll` option, the command immediately controls all the paths and then reports their status.

The output of the `multipath -ll` command provides information on each multipath device. For each device, the command lists the path groups and the paths in each group. The following example lists two multipath devices.

```
[root@node ~]# multipath -ll
mpathb (3624a937081dd276770004f93b8f9ce10) dm-1 PURE      ,FlashArray
size=2.0T features='0' hwhandler='1' alua' wp=rw
|--- policy='queue-length 0' prio=50 status=active
|   | 4:0:0:1 sdc 8:32  active ready running
|   | 5:0:0:1 sde 8:64  active ready running
|   | 6:0:0:1 sdg 8:96  active ready running
|   \ 7:0:0:1 sdi 8:128 active ready running
\--- policy='queue-length 0' prio=10 status=enabled
  | 8:0:0:1 sdd 8:48  active ready running
  | 9:0:0:1 sdf 8:80  active ready running
  | 10:0:0:1 sdh 8:112 active ready running
  \ 11:0:0:1 sdj 8:144 active ready running
mpatha (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-ORG,clusterstor
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=active
|   ` 2:0:0:0 sda 8:0  active ready running
`--- policy='service-time 0' prio=50 status=enabled
  ` 3:0:0:0 sdb 8:16 active ready running
```

The first two lines provide information on the multipath device:

```
mpathb (3624a937081dd276770004f93b8f9ce10) dm-1 PURE      ,FlashArray
size=2.0T features='0' hwhandler='1' alua' wp=rw
...output omitted...
```

The first line displays the user-friendly name (`mpathb`), the device's WWID, and the vendor and product name. The second line gives the device size, the write permission (`wp`), and other details.

The output shows the details of each path group:

```
...output omitted...
|-+ policy='queue-length 0' prio=50 status=active
...output omitted...
`-- policy='queue-length 0' prio=10 status=enabled
...output omitted...
```

In this example, the `mpathb` device has two path groups. Only one path group is active at a time, and multipath switches to the inactive path group when the active group fails. The `status` attribute indicates which path group is active. The second path group has a status of `enabled`, which indicates that the path group is ready if the first path group fails. The `policy` attribute gives the algorithm that multipath uses to distributes the I/O requests between the group's paths. That attribute corresponds to the `path_selector` parameter in the `/etc/multipath.conf` file.

For each path group, the command lists the paths:

```
...output omitted...
| |- 4:0:0:1 sdc 8:32 active ready running
| |- 5:0:0:1 sde 8:64 active ready running
| |- 6:0:0:1 sdg 8:96 active ready running
| `-- 7:0:0:1 sdi 8:128 active ready running
...output omitted...
```

multipath distributes the I/O requests between all the paths in the active group. The status information helps assess the health of the path. A path that is up and ready for I/O operations has a status of `ready`. A path that is down has a status of `faulty`:

```
| |- 4:0:0:1 sdc 8:32 failed faulty offline
```

The two fields should report the same information. The first field reports the status of the path device from the kernel point of view. The second field reports the status from the multipath path checker. The `/etc/multipath.conf` file defines that path checker by using the `path_checker` parameter.

The output shows that the system has a second multipath device. That device's name is `mpatha`. The device has two path groups, and each group has one path:

```
...output omitted...
mpatha (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-ORG,clusterstor
size=10G features='0' hwhandler='1' alua' wp=rw
|-+ policy='service-time 0' prio=50 status=active
| `-- 2:0:0:0 sda 8:0 active ready running
`-- policy='service-time 0' prio=50 status=enabled
  -- 3:0:0:0 sdb 8:16 active ready running
```

## Identifying the Path Grouping Policy

The `multipath -ll` command does not explicitly state the path grouping policy of a multipath device. However, you can deduce that information by inspecting the command output.

A multipath device with a path grouping policy of `failover` has only one path in each group. The following output shows an example of this policy. Each path group contains a single path.

```
[root@node ~]# multipath -ll
mpatha (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-ORG,clusterstor
size=10G features='0' hwhandler='1' alua' wp=rw
`-- policy='service-time 0' prio=50 status=active
  |- 2:0:0:0 sda 8:0 active ready running
  `-- policy='service-time 0' prio=50 status=enabled
    '- 3:0:0:0 sdb 8:16 active ready running
```

On the other hand, a multipath device with a path grouping policy of **multibus** puts all the paths into one group. The following example shows a single path group. All the paths are members of this single group.

```
[root@node ~]# multipath -ll
mpatha (360014053bd9ea2a35914e39a556051cf) dm-0 LIO-ORG ,clusterstor2
size=4.0G features='0' hwhandler='1' alua' 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
```

## Monitoring Failover

The **multipath -ll** command helps assess the failover activities of multipath devices. Remember that at any given time, only one path group is in the **active** state. The remaining path groups are waiting with the **enabled** status, as shown in the following example.

```
[root@node ~]# multipath -ll
mpatha (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-ORG,clusterstor
size=10G features='0' hwhandler='1' alua' wp=rw
`-- policy='service-time 0' prio=50 status=active
  |- 2:0:0:0 sda 8:0 active ready running
  `-- policy='service-time 0' prio=50 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 a path failure occurs on a passive path group. Although the status for the path in the passive path group changes, the status of the active path group and its corresponding path remains unchanged and unimpaired.

```
[root@node ~]# multipath -ll
mpatha (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-ORG,clusterstor
size=10G features='0' hwhandler='1' alua' wp=rw
`-- policy='service-time 0' prio=50 status=active
  |- 2:0:0:0 sda 8:0 active ready running
  `-- policy='service-time 0' prio=50 status=enabled
    '- 3:0:0:0 sdb 8:16 failed faulty offline
```

The following example illustrates the failure of a path in the active path group. The status for the previously active path changes to **failed faulty offline**. Consequently, the corresponding path group's status also changes, from **active** to **enabled**. The passive path group becomes **active**.

```
[root@node ~]# multipath -ll
mpatha (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-0RG,clusterstor
size=10G features='0' hwhandler='1' alua' wp=rw
|-- policy='service-time 0' prio=50 status=enabled
| `-- 2:0:0:0 sda 8:0 failed faulty offline
`-- policy='service-time 0' prio=50 status=active
   `-- 3:0:0:0 sdb 8:16 active ready running
```



### Note

Even though the failed path recovers, the current active path remains active. You can control this behavior with the `fallback` parameter in the `/etc/multipath.conf` file. The default value of the `fallback` parameter is `manual`.



### References

`multipath(8)` man page

For more information, refer to the *The multipath command* section in the *Configuring device mapper multipath* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_device\\_mapper\\_multipath/index#assembly\\_multipath-command-managing-mpath](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_device_mapper_multipath/index#assembly_multipath-command-managing-mpath)

## ► Guided Exercise

# Testing Resilient Storage Access

In this exercise, you will confirm that you can access iSCSI-based storage through a multipath device that has two paths to the storage, even when one path is not available.

## Outcomes

You should be able to test and observe the redundancy of a multipath device configured with the failover path grouping policy.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start multipath-test
```

This command configures a block device named `/dev/mapper/mpatha` on nodea that provides multiple paths to a shared iSCSI storage device for redundancy.

## Instructions

- 1. On nodea, create an XFS file system on the `/dev/mapper/mpatha` device and then mount it at `/mnt`.
- 1.1. Connect to nodea and become the root user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Create an XFS file system on the `/dev/mapper/mpatha` device.

```
[root@nodea ~]# mkfs -t xfs /dev/mapper/mpatha  
...output omitted...
```

- 1.3. Mount the file system.

```
[root@nodea ~]# mount /dev/mapper/mpatha /mnt  
[root@nodea ~]#
```

- 2. Determine the active path in use by the multipath device.

```
[root@nodea ~]# multipath -ll
mpatha (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|-- policy='service-time 0' prio=50 status=active
| `-- 2:0:0:0 sda 8:0 active ready running
`-- policy='service-time 0' prio=50 status=enabled
`- 3:0:0:0 sdb 8:16 active ready running
```

In the preceding output, the active path is using the /dev/sda device.

- ▶ 3. Determine the network device that provides connectivity for the active path to the iSCSI storage.
  - 3.1. Use the `iscsiadm` command to retrieve the IP address of the remote portal that the active path is using.

```
[root@nodea ~]# iscsiadm -m session -P 3 | egrep 'Current Portal|scsi disk'
Current Portal: 192.168.1.15:3260,1
    Attached scsi disk sda      State: running
Current Portal: 192.168.2.15:3260,1
    Attached scsi disk sdb      State: running
```

In the preceding output, the /dev/sda device is accessing the target at 192.168.1.15.

- 3.2. Use the `ip route` command to retrieve the name of the network device that the system is using to access the remote portal.

```
[root@nodea ~]# ip route
...output omitted...
172.25.250.0/24 dev eth0 proto kernel scope link src 172.25.250.10 metric 100
192.168.0.0/24 dev eth1 proto kernel scope link src 192.168.0.10 metric 105
192.168.1.0/24 dev eth2 proto kernel scope link src 192.168.1.10 metric 102
192.168.2.0/24 dev eth3 proto kernel scope link src 192.168.2.10 metric 103
```

- ▶ 4. Disable the network interface for the active path and then observe the failover of the path to the iSCSI storage. Confirm that you can still access the file system contents.
  - 4.1. Use the `watch` command to monitor the status of the multipath device.

```
[root@nodea ~]# watch -n 1 -d "multipath -ll"
Every 1.0s: multipath -ll      nodea.lab.example.com: Mon Jan  4 07:07:37 2021

mpatha (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|-- policy='service-time 0' prio=50 status=active
| `-- 2:0:0:0 sda 8:0 active ready running
`-- policy='service-time 0' prio=50 status=enabled
`- 3:0:0:0 sdb 8:16 active ready running
```

Leave the command running.

- 4.2. Open a new terminal, log in to nodea, become the `root` user, and then disable the network interface.

```
[root@nodea ~]# nmcli device disconnect eth2
Device 'eth2' successfully disconnected.
```

- 4.3. In the first terminal, observe the failover to the redundant path.

```
Every 1.0s: multipath -ll      nodea.lab.example.com: Mon Jan  4 07:10:39 2021

mpatha (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=0 status=enabled
|   `-- 2:0:0:0 sda 8:0  failed faulty running
`--- policy='service-time 0' prio=50 status=active
    '- 3:0:0:0 sdb 8:16 active ready running
```

- 4.4. Create a file in /mnt to confirm that you can still access the file system.

```
[root@nodea ~]# echo Hello World > /mnt/test1.txt
[root@nodea ~]#
```

- 5. Re-enable the network interface to restore redundancy to the multipath device. Confirm that you can access the file system contents.

- 5.1. Enable the network interface.

```
[root@nodea ~]# nmcli device connect eth2
Device 'eth2' successfully activated with '1b5fb073-2a7f-4bc7-8f4d-354dabe8a447'.
```

- 5.2. Observe the restoration of the redundancy to the iSCSI storage. Notice that the current path group stays active. The multipath device does not fall back to the previously active path group. When done, press **Ctrl+C** to exit the **watch** command.

```
Every 1.0s: multipath -ll      nodea.lab.example.com: Mon Jan  4 07:14:49 2021

mpatha (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=enabled
|   `-- 2:0:0:0 sda 8:0  active ready running
`--- policy='service-time 0' prio=50 status=active
    '- 3:0:0:0 sdb 8:16 active ready running
```

- 5.3. Create a file in /mnt to confirm that you can still access the file system.

```
[root@nodea ~]# echo Hello World > /mnt/test2.txt
[root@nodea ~]#
```

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish multipath-test
```

This concludes the section.

## ▶ Lab

# Accessing Storage Devices Resiliently

In this lab, you will configure a multipath device to the same shared storage on multiple nodes of a high availability cluster, ensuring that the multipath device has the same name on all nodes.

## Outcomes

You should be able to create an identically named multipath device on several machines and configure it for redundant access to iSCSI storage.

## Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start multipath-review
```

This command configures an iSCSI target on the **storage** machine.

## Instructions

- Configure an iSCSI initiator on the **nodec** and **noded** machines to access the target that the **storage** machine provides. Discover and then log in to the target twice, once through each of the two network paths. The following table provides the configuration details.

### iSCSI Initiator Configuration

Parameter	Value
Initiator IQN on <b>nodec</b>	<code>iqn.2021-04.com.example:nodec</code>
Initiator IQN on <b>noded</b>	<code>iqn.2021-04.com.example:noded</code>
Target IQN	<code>iqn.2021-04.com.example:store1</code>
Portal address on the first network	<code>192.168.1.15, port 3260</code>
Portal address on the second network	<code>192.168.2.15, port 3260</code>

You can copy and paste those parameters from the `/root/multipath-review/iscsi.txt` file on the **nodec** and **noded** machines.

The password for the **student** user is **student**, and the **root** password is **redhat**.

- On both the **nodec** and **noded** machines, use the two network paths to configure a multipath device that provides redundant access to the iSCSI target on the **storage** machine. Assign the `disk1` alias to the multipath device so that both machines can use the storage through the `/dev/mapper/disk1` device node.

3. Create an `ext4` file system on the multipath device. To test your work, mount the file system on the `nodec` machine, create a file, and then unmount the file system. Repeat the process on the `noded` machine. Remember never to mount the file system on both nodes at the same time.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade multipath-review
```

## Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish multipath-review
```

This concludes the section.

## ► Solution

# Accessing Storage Devices Resiliently

In this lab, you will configure a multipath device to the same shared storage on multiple nodes of a high availability cluster, ensuring that the multipath device has the same name on all nodes.

## Outcomes

You should be able to create an identically named multipath device on several machines and configure it for redundant access to iSCSI storage.

## Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start multipath-review
```

This command configures an iSCSI target on the storage machine.

## Instructions

- Configure an iSCSI initiator on the **nodec** and **noded** machines to access the target that the **storage** machine provides. Discover and then log in to the target twice, once through each of the two network paths. The following table provides the configuration details.

### iSCSI Initiator Configuration

Parameter	Value
Initiator IQN on <b>nodec</b>	<code>iqn.2021-04.com.example:nodec</code>
Initiator IQN on <b>noded</b>	<code>iqn.2021-04.com.example:noded</code>
Target IQN	<code>iqn.2021-04.com.example:store1</code>
Portal address on the first network	<code>192.168.1.15, port 3260</code>
Portal address on the second network	<code>192.168.2.15, port 3260</code>

You can copy and paste those parameters from the `/root/multipath-review/iscsi.txt` file on the **nodec** and **noded** machines.

The password for the **student** user is **student**, and the **root** password is **redhat**.

- In two separate terminals, connect to the **nodec** and **noded** machines, and then become the **root** user.

```
[student@workstation ~]$ ssh nodec  
...output omitted...  
[student@nodec ~]$ sudo -i  
[sudo] password for student: student  
[root@nodec ~]#
```

```
[student@workstation ~]$ ssh noded  
...output omitted...  
[student@noded ~]$ sudo -i  
[sudo] password for student: student  
[root@noded ~]#
```

- 1.2. On both machines, install the *iscsi-initiator-utils* package.

```
[root@nodec ~]# yum install iscsi-initiator-utils
```

```
[root@noded ~]# yum install iscsi-initiator-utils
```

- 1.3. On both machines, edit the */etc/iscsi/initiatorname.iscsi* file and set the IQN for the client initiator, *iqn.2021-04.com.example:nodec* for the *nodec* machine and *iqn.2021-04.com.example:noded* for the *noded* machine.

```
InitiatorName=iqn.2021-04.com.example:nodec
```

```
InitiatorName=iqn.2021-04.com.example:noded
```

- 1.4. On both machines, restart the *iscsid* service to take the new IQN into account.

```
[root@nodec ~]# systemctl restart iscsid
```

```
[root@noded ~]# systemctl restart iscsid
```

- 1.5. On both machines, discover and then log in to the iSCSI target through the first network. The portal address on that network is *192.168.1.15*.

```
[root@nodec ~]# iscsiadm -m discovery -t st -p 192.168.1.15  
192.168.1.15:3260,1 iqn.2021-04.com.example:store1  
[root@nodec ~]# iscsiadm -m node -T iqn.2021-04.com.example:store1 \  
> -p 192.168.1.15 -l  
Logging in to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.1.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.1.15,3260] successful.
```

```
[root@noded ~]# iscsiadadm -m discovery -t st -p 192.168.1.15  
192.168.1.15:3260,1 iqn.2021-04.com.example:store1  
[root@noded ~]# iscsiadadm -m node -T iqn.2021-04.com.example:store1 \  
> -p 192.168.1.15 -l  
Logging in to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.1.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.1.15,3260] successful.
```

If the command fails, confirm that you correctly set the initiator IQN in the /etc/iscsi/initiatorname.iscsi file. If not, then fix the issue in the file, restart the iscsid service, and try the preceding commands again.

- 1.6. On both machines, discover and then log in to the iSCSI target through the second network. The portal address on that network is 192.168.2.15.

```
[root@nodec ~]# iscsiadadm -m discovery -t st -p 192.168.2.15  
192.168.2.15:3260,1 iqn.2021-04.com.example:store1  
[root@nodec ~]# iscsiadadm -m node -T iqn.2021-04.com.example:store1 \  
> -p 192.168.2.15 -l  
Logging in to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.2.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.2.15,3260] successful.
```

```
[root@noded ~]# iscsiadadm -m discovery -t st -p 192.168.2.15  
192.168.2.15:3260,1 iqn.2021-04.com.example:store1  
[root@noded ~]# iscsiadadm -m node -T iqn.2021-04.com.example:store1 \  
> -p 192.168.2.15 -l  
Logging in to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.2.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store1, portal:  
192.168.2.15,3260] successful.
```

2. On both the nodec and noded machines, use the two network paths to configure a multipath device that provides redundant access to the iSCSI target on the storage machine. Assign the disk1 alias to the multipath device so that both machines can use the storage through the /dev/mapper/disk1 device node.

- 2.1. Install the *device-mapper-multipath* package on the two machines.

```
[root@nodec ~]# yum install device-mapper-multipath
```

```
[root@noded ~]# yum install device-mapper-multipath
```

- 2.2. On the nodec machine, use the mpathconf command to create the multipath configuration file. You create and configure that file on the nodec machine and then copy it to the noded machine in a following step.

```
[root@nodec ~]# mpathconf --enable  
[root@nodec ~]#
```

**Chapter 9 |** Accessing Storage Devices Resiliently

- 2.3. On the nodec machine, identify one of the newly available block devices.

```
[root@nodec ~]# iscsidadm -m session -P 3  
...output omitted...  
Attached scsi disk sdb           State: running
```

- 2.4. Retrieve the WWID of the iSCSI storage.

```
[root@nodec ~]# udevadm info /dev/sdb | grep ID_SERIAL=  
E: ID_SERIAL=36001405b7118eb213ff4bc792f144a04
```

On your system, the WWID is probably different.

- 2.5. Edit the `/etc/multipath.conf` configuration file and then add the `multipaths` section at the end. In the following example, replace the WWID with the one you retrieved in the preceding step.

```
multipaths {  
    multipath {  
        wwid            36001405b7118eb213ff4bc792f144a04  
        alias          disk1  
    }  
}
```

- 2.6. Copy the `/etc/multipath.conf` configuration file from the nodec to the noded machine.

```
[root@nodec ~]# scp /etc/multipath.conf noded:/etc  
The authenticity of host 'noded (172.25.250.13)' can't be established.  
ECDSA key fingerprint is SHA256:WL9YdEHqYZwY45yU+mS9pU7bnS08wsNxvGZ1LegLU/Y.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added 'noded,172.25.250.13' (ECDSA) to the list of known  
hosts.  
root@noded's password: redhat  
multipath.conf                                         100%   596   901.1KB/s   00:00
```

- 2.7. Enable and start the `multipathd` service on both machines.

```
[root@nodec ~]# systemctl enable --now multipathd  
[root@nodec ~]#
```

```
[root@noded ~]# systemctl enable --now multipathd  
[root@noded ~]#
```

- 2.8. On both machines, use the `multipath` command to verify the configuration.

```
[root@nodec ~]# multipath -ll
disk1 (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=active
| `-- 2:0:0:0 sda 8:0 active ready running
`--- policy='service-time 0' prio=50 status=enabled
   `-- 3:0:0:0 sdb 8:16 active ready running
```

```
[root@noded ~]# multipath -ll
disk1 (36001405b7118eb213ff4bc792f144a04) dm-0 LIO-ORG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|--- policy='service-time 0' prio=50 status=active
| `-- 2:0:0:0 sda 8:0 active ready running
`--- policy='service-time 0' prio=50 status=enabled
   `-- 3:0:0:0 sdb 8:16 active ready running
```

- 2.9. Confirm that the new device node is available under the /dev/mapper/ directory.

```
[root@nodec ~]# ls /dev/mapper/disk1
/dev/mapper/disk1
```

```
[root@noded ~]# ls /dev/mapper/disk1
/dev/mapper/disk1
```

3. Create an ext4 file system on the multipath device. To test your work, mount the file system on the nodec machine, create a file, and then unmount the file system. Repeat the process on the noded machine. Remember never to mount the file system on both nodes at the same time.

- 3.1. On the nodec machine, use the mkfs.ext4 command to apply an ext4 file system to /dev/mapper/disk1.

```
[root@nodec ~]# mkfs.ext4 /dev/mapper/disk1
...output omitted...
```

- 3.2. Temporarily mount the file system under the /mnt mount point, create a test file, and then unmount the file system.

```
[root@nodec ~]# mount /dev/mapper/disk1 /mnt
[root@nodec ~]# echo Hello World > /mnt/test.txt
[root@nodec ~]# umount /mnt
[root@nodec ~]#
```

- 3.3. On the noded machine, mount the file system on the /mnt mount point, and then confirm that you can access the test file you have created in the preceding step. Unmount the file system when done.

```
[root@noded ~]# mount /dev/mapper/disk1 /mnt
[root@noded ~]# cat /mnt/test.txt
Hello World
[root@noded ~]# umount /mnt
[root@noded ~]#
```

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade multipath-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish multipath-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- Multipathing, which requires installation of the `device-mapper-multipath` package, can be used to provide resilient access to storage.
- With user-friendly names enabled, multipath devices are created as `/dev/mapper/mpathN` devices.
- Multipath devices default to an 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 LVM in Clusters

### Goal

Select the correct Logical Volume Management (LVM) configuration for use in your cluster, and configure and manage it.

### Objectives

- Describe the basic logical volume management concepts.
- Configure a highly available logical volume as a resource that one node uses at a time in a high availability cluster.
- Create a logical volume from a shared volume group that all nodes can use at the same time in a high availability cluster.

### Sections

- Reviewing LVM Concepts (and Quiz)
- Managing High Availability LVM (and Guided Exercise)
- Managing LVM Shared Volume Groups (and Guided Exercise)

### Lab

- Configuring LVM in Clusters

# Reviewing LVM Concepts

---

## Objectives

After completing this section, you should be able to describe the basic logical volume management concepts.

## Defining Logical Volume Management

*Logical volume management (LVM)* makes it easier to manage disk space. For example, when a file system on a logical volume needs more space, you extend the logical volume using the free space in its volume group and then resize the file system. If a disk starts to fail then you register a replacement disk as a physical volume with the volume group and migrate the logical volume's extents to the new disk.

## Defining LVM Key Terms

Some terms you will hear when working with LVM include:

### Physical devices

Physical devices are the storage devices used to save data stored in a logical volume. These are block devices and could be disk partitions, whole disks, RAID arrays, or SAN disks. A device must be initialized as an LVM physical volume in order to be used with LVM. The entire device is used as a physical volume.

### Physical volumes (PVs)

Physical volumes are the underlying physical storage used with LVM. You must initialize a device as a physical volume before using it in an LVM system. LVM tools segment physical volumes into *physical extents (PEs)*, which are small chunks of data that act as the smallest storage block on a physical volume.

### Volume groups (VGs)

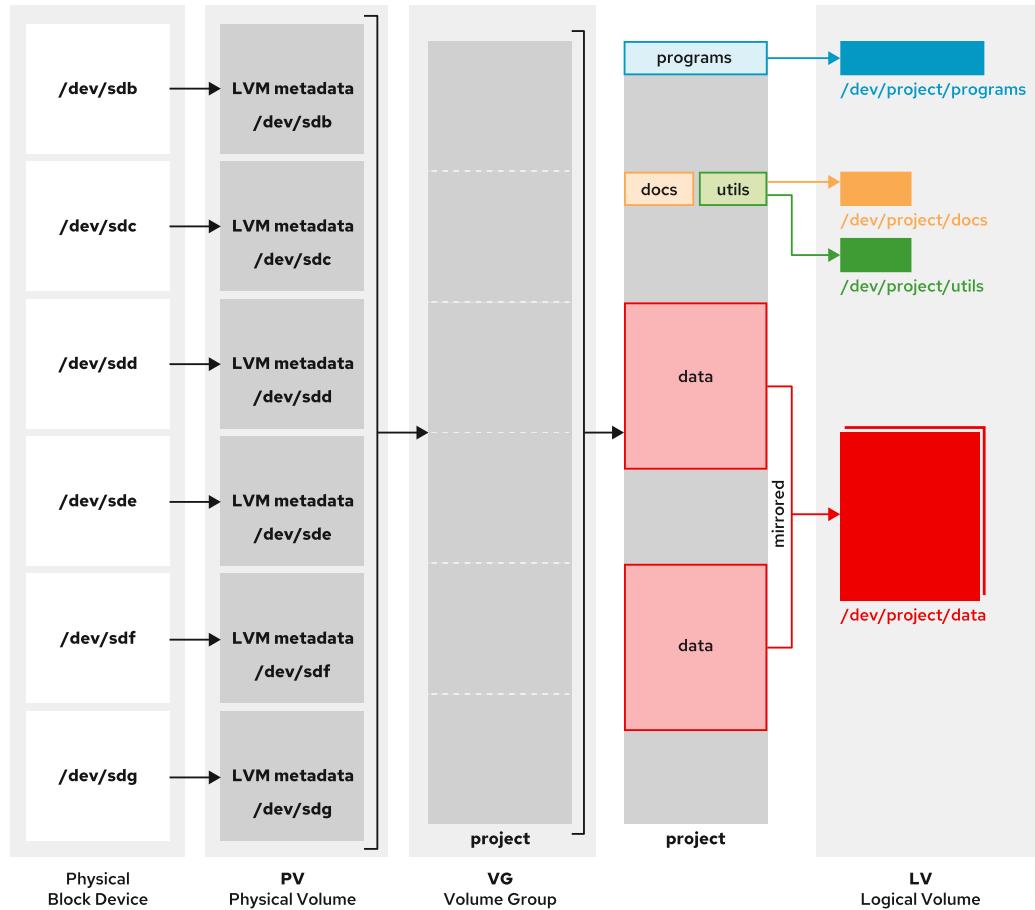
Volume groups are storage pools made up of one or more physical volumes. This is the functional equivalent of a whole disk in basic storage. A PV can only be allocated to a single VG. A VG can consist of unused space and any number of logical volumes.

### Logical volumes (LVs)

Logical volumes are created from free physical extents in a volume group and provide the "storage" device used by applications, users, and the operating system. LVs are a collection of *logical extents (LEs)*, which map to physical extents, the smallest storage chunk of a PV. By default, each LE maps to one PE. Setting specific LV options changes this mapping; for example, mirroring causes each LE to map to two PEs.

## Implementing LVM Storage

Creating LVM storage requires several steps. The first step is to determine which physical devices to use. Those devices are initialized as physical volumes so that they are recognized as belonging to LVM. The physical volumes are then combined into a volume group. This creates a pool of disk space out of which logical volumes can be allocated. Logical volumes created from the available space in a volume group can be formatted with a file system, activated as swap space, and mounted or activated persistently.



**Figure 10.1: Logical Volume Management Components**

LVM provides a comprehensive set of command-line tools for implementing and managing LVM storage.

## Preparing the Physical Device

Before proceeding with LVM, select the block device that you will use to create the physical volume. You can choose an entire disk or only a partition on a disk.



### Important

The following examples use the `sdb` and `sdc` devices to illustrate LVM commands. In practice, you need to use the correct devices for the disk and disk partitions used by the system. Use the `lsblk`, `blkid`, or `cat /proc/partitions` commands to identify the devices on your system.

## Creating a Physical Volume

Use the `pvcreate` command to label the physical device (or the partition) as a physical volume. The `pvcreate` command divides the physical volume into physical extents (PEs) of a fixed size; 4 MiB blocks for example. You can label multiple devices at the same time by using space-delimited device names as arguments to the `pvcreate` command.

```
[root@host ~]# pvcreate /dev/sdb /dev/sdc
```

The preceding command labels the devices `/dev/sdb` and `/dev/sdc` as PVs, ready for allocation into a volume group.

## Creating a Volume Group

Use the `vgcreate` command to collect one or more physical volumes into a volume group. A volume group is the functional equivalent of a hard disk; you create logical volumes from the pool of free physical extents in the volume group.

The `vgcreate` command consists of a volume group name followed by one or more physical volumes to allocate to this volume group.

```
[root@host ~]# vgcreate vg01 /dev/sdb /dev/sdc
```

The preceding command creates a VG called `vg01` that is the combined size, in PE units, of the two PVs `/dev/sdb` and `/dev/sdc`.

## Creating a Logical Volume

Use the `lvcreate` command to create a new logical volume from the available physical extents in a volume group. At a minimum, the `lvcreate` command requires the `-n` option to set the LV name, either the `-L` option to set the LV size in bytes or the `-l` option to set the LV size in extents, and the name of the volume group hosting this logical volume.

```
[root@host ~]# lvcreate -n lv01 -L 700M vg01
```

The preceding command creates an LV called `lv01`, 700 MiB in size, in the VG `vg01`. The command fails if the volume group does not have a sufficient number of free physical extents for the requested size. Note also that the size is rounded to a factor of the physical extent size if the size cannot match exactly.

You can specify the size by using the `-L` option, which expects sizes in bytes, mebibytes (binary megabytes, 1048576 bytes), gibibytes (binary gigabytes), or similar. Alternatively, you can use the `-l` option, which expects sizes specified as a number of physical extents.

For example, with the `-l 700` option, the `lvcreate` command sets the size of the logical volume to 700 extents. The total number of bytes depends on the size of the physical extent block on the underlying physical volume.

## Adding a File System

Use the `mkfs` command to create a file system on the new logical volume. The following command creates an XFS file system.

```
[root@host ~]# mkfs -t xfs /dev/vg01/lv01
```



### Important

Different tools display the logical volume name using either the traditional name, `/dev/vgname/lvname`, or the kernel device mapper name, `/dev/mapper/vgname-lvname`.

In the preceding example, you could use the `/dev/mapper/vg01-lv01` device node as an alternative.

## Reviewing LVM Status Information

LVM provides a set of command-line tools to inspect your configuration.

### Retrieving Physical Volumes Details

Use the `pvdisplay` command to display information about physical volumes. To list information about all physical volumes, use the command without arguments. To list information about a specific physical volume, pass that device name to the command.

```
[root@host ~]# pvdisplay /dev/sdb
--- Physical volume ---
PV Name      /dev/sdb
VG Name      vg01
PV Size      10.00 GiB / not usable 4.00 MiB
Allocatable   yes
PE Size      4.00 MiB
Total PE     2559
Free PE      2384
Allocated PE 175
PV UUID      JWzDpn-LG3e-n2oi-9EtD-VT2H-PMem-1ZXwP1
```

- ➊ PV Name maps to the device name.
- ➋ VG Name shows the volume group where the PV is allocated.
- ➌ PV Size shows the physical size of the PV, including any unusable space.
- ➍ PE Size is the physical extent size, which is the smallest size that can be allocated to a logical volume. It is also the multiplying factor when calculating the size of any value reported in PE units, such as Free PE or Allocated PE. For example, 175 PEs x 4 MiB (the PE Size) equals 700 MiB of allocated space. A logical volume size is rounded to a factor of PE units. LVM sets the PE size automatically, although it is possible to specify it.
- ➎ Free PE shows how many PE units are available for allocation to new logical volumes. As an alternative, the `pvs` command provides a more compact output:

```
[root@host ~]# pvs /dev/sdb
PV          VG      Fmt  Attr PSize    PFree
/dev/sdb    vg01   lvm2  a--  <10.00g  9.31g
```

## Listing Volume Groups

Use the `vgdisplay` command to display information about volume groups. To list information about all volume groups, use the command without arguments. To list information about a specific volume group, pass that VG name to the command.

```
[root@host ~]# vgdisplay vg01
--- Volume group ---
VG Name          vg01      ①
System ID
Format           lvm2
Metadata Areas   2
Metadata Sequence No 2
VG Access        read/write
VG Status        resizable
MAX LV
Cur LV
Open LV
Max PV
Cur PV
Act PV
VG Size          <10.00 GiB    ②
PE Size          4.00 MiB
Total PE         2559      ③
Alloc PE / Size  175 / 700.00 MiB
Free  PE / Size  2384 / 9.31 MiB ④
VG UUID          3snNw3-CF71-CcYG-Llk1-p6EY-rHEv-xfUSez
```

- ① VG Name is the name of the volume group.
- ② VG Size is the total size of the storage pool available for logical volume allocation.
- ③ Total PE is the total size expressed in number of physical extents (PEs).
- ④ Free PE / Size shows how much space is free in the VG for allocating to new LVs or to extend existing logical volumes.

As an alternative, the `vgs` command provides a more compact output:

```
[root@host ~]# vgs vg01
VG #PV #LV #SN Attr   VSize   VFree
vg01   2   1   0 wza--n- <10.00g 9.31g
```

## Listing Logical Volumes

Use the `lvdisplay` command to display information about logical volumes. If you run the command without arguments, then it displays information about all logical volumes. If you provide an LV device name as an argument, then the command displays information about that specific device.

```
[root@host ~]# lvdisplay /dev/vg01/lv01
--- Logical volume ---
LV Path          /dev/vg01/lv01      ①
LV Name          lv01
```

VG Name	vg01	❷
LV UUID	5IyRea-W8Zw-xLHk-3h2a-IuVN-YaeZ-i3IRrN	
LV Write Access	read/write	
LV Creation host, time	host.lab.example.com, 2019-03-28 17:17:47 -0400	
LV Status	available	
# open	1	
LV Size	700 MiB	❸
Current LE	175	❹
Segments	1	
Allocation	inherit	
Read ahead sectors	auto	
- current set to	8192	
Block device	253:0	

- ❶ LV Path shows the device name of the logical volume. Some tools report the device name as `/dev/mapper/vgname-lvname`; both represent the same LV.
- ❷ VG Name shows the volume group that the LV is allocated from.
- ❸ LV Size shows the total size of the LV. Use file-system tools to determine the free space and used space for storage of data.
- ❹ Current LE shows the number of logical extents used by this LV. An LE usually maps to a physical extent in the VG, and therefore the physical volume.

As an alternative, the `lvs` command provides a more compact output:

```
[root@host ~]# lvs /dev/vg01/lv01
  LV   VG Attr      LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  lv01 vg01 -wi-a---- 700.00m
```



## References

`lvm(8)`, `pvcreate(8)`, `vgcreate(8)`, `lvcreate(8)`, `pvdisplay(8)`, `pvs(8)`, `vgdisplay(8)`, `vgs(8)`, `lvdisplay(8)`, `lvs(8)`, and `lvm.conf(5)` man pages

**Knowledgebase: "What are the advantages and disadvantages to using partitioning on LUNs, either directly or with LVM in between?"**

<https://access.redhat.com/solutions/163853>

## ► Quiz

# Reviewing LVM Concepts

Choose the correct answers to the following questions:

- ▶ 1. **What is the correct order of the LVM commands for creating a volume group and a logical volume on a new block device?**
  - a. pvcreate, lvcreate, and then vgcreate
  - b. pvcreate, vgcreate, and then lvcreate
  - c. lvcreate, vgcreate, and then pvcreate
  - d. vgcreate, pvcreate, and then lvcreate
  
- ▶ 2. **Which two of the following device nodes are valid when referring to the dbdata logical volume in the vg01 volume group? (Choose two.)**
  - a. /dev/vg01/dbdata
  - b. /dev/dbdata/vg01
  - c. /dev/vg01-dbdata
  - d. /dev/mapper/vg01-dbdata
  - e. /dev/mapper/dbdata-vg01
  
- ▶ 3. **On your system, the /dev/sdb device hosts the vg01 volume group. What is the valid command to create a 5 GiB logical volume named dbdata in that volume group?**
  - a. lvcreate -L 5G dbdata vg01
  - b. lvcreate -l 5G -n dbdata vg01
  - c. lvcreate -L 5G -n dbdata /dev/sdb
  - d. lvcreate -L 5G -n dbdata vg01
  - e. lvcreate -L 5G dbdata /dev/sdb
  
- ▶ 4. **Which two commands list the logical volumes on your system? (Choose two.)**
  - a. lvs
  - b. lvmconfig
  - c. lvdisplay
  - d. vgs
  - e. lvmdump

## ► Solution

# Reviewing LVM Concepts

Choose the correct answers to the following questions:

- ▶ 1. **What is the correct order of the LVM commands for creating a volume group and a logical volume on a new block device?**
  - a. pvcreate, lvcreate, and then vgcreate
  - b. pvcreate, vgcreate, and then lvcreate
  - c. lvcreate, vgcreate, and then pvcreate
  - d. vgcreate, pvcreate, and then lvcreate
  
- ▶ 2. **Which two of the following device nodes are valid when referring to the dbdata logical volume in the vg01 volume group? (Choose two.)**
  - a. /dev/vg01/dbdata
  - b. /dev/dbdata/vg01
  - c. /dev/vg01-dbdata
  - d. /dev/mapper/vg01-dbdata
  - e. /dev/mapper/dbdata-vg01
  
- ▶ 3. **On your system, the /dev/sdb device hosts the vg01 volume group. What is the valid command to create a 5 GiB logical volume named dbdata in that volume group?**
  - a. lvcreate -L 5G dbdata vg01
  - b. lvcreate -l 5G -n dbdata vg01
  - c. lvcreate -L 5G -n dbdata /dev/sdb
  - d. lvcreate -L 5G -n dbdata vg01
  - e. lvcreate -L 5G dbdata /dev/sdb
  
- ▶ 4. **Which two commands list the logical volumes on your system? (Choose two.)**
  - a. lvs
  - b. lvmconfig
  - c. lvdisplay
  - d. vgs
  - e. lvmdump

# Managing High Availability LVM

## Objectives

After completing this section, you should be able to configure a highly available logical volume as a resource that one node uses at a time in a high availability cluster.

## Introducing the LVM Configuration File

Use the `/etc/lvm/lvm.conf` file to configure the behavior of LVM. This file has settings for controlling locking behavior, which devices should be scanned for LVM signatures, as well as many other behaviors. In most situations, you do not have to modify that file.

## Comparing High Availability LVM with LVM Shared Volume Groups

There are two ways you can use LVM with shared storage in a cluster: *High Availability LVM (HA-LVM)* or *LVM Shared Volume Groups*.

With HA-LVM, one node at a time accesses a volume group and its logical volumes. HA-LVM is a good choice for active/passive applications that work with a traditional file system, such as ext4 or XFS. To prevent data corruption, only one node at a time can access the volume group and can mount the file system.

With LVM shared volume groups, the volume group and its logical volumes are available to all cluster nodes at all times. You use LVM shared volume groups when working with a shared file system, such as GFS2. Because all the nodes can mount and use the GFS2 file system simultaneously, you choose LVM shared volume groups for applications that require access to the shared data from multiple nodes at the same time. Configuring LVM shared volume groups is a complex process, which is described in the next section.

## Creating a High Availability LVM Volume

In preparation for HA-LVM, you first configure LVM on all your cluster nodes. After that initial step, you create physical volumes, volume groups, and logical volumes as you would normally, using the standard LVM commands.

## Preparing the Systems for HA-LVM

To prevent the systems from activating a volume group on multiple nodes, HA-LVM stores a system ID in the volume group metadata. With that configuration, only the node with a matching system ID can activate the volume group and its logical volumes.

LVM supports different types of system IDs. The `uname` type, which uses the system host name as the system ID, is easy to configure. To configure `uname` support, on every cluster nodes, edit the `/etc/lvm/lvm.conf` file and set the `system_id_source` parameter to `uname`.

```
...output omitted...
# Configuration option global/system_id_source.
# The method LVM uses to set the local system ID.
```

**Chapter 10 |** Configuring LVM in Clusters

```
# Volume Groups can also be given a system ID (by vgcreate, vgchange,
# or vgimport.) A VG on shared storage devices is accessible only to
# the host with a matching system ID. See 'man lvmsystemid' for
# information on limitations and correct usage.
#
# Accepted values:
#   none
#       The host has no system ID.
#   lvmlocal
#       Obtain the system ID from the system_id setting in the 'local'
#       section of an lvm configuration file, e.g. lvmlocal.conf.
#   uname
#       Set the system ID from the hostname (uname) of the system.
#       System IDs beginning localhost are not permitted.
#   machineid
#       Use the contents of the machine-id file to set the system ID.
#       Some systems create this file at installation time.
#       See 'man machine-id' and global/etc.
#   file
#       Use the contents of another file (system_id_file) to set the
#       system ID.
#
system_id_source = "uname"

...output omitted...
```

You can verify your configuration with the `lvm systemid` command. The command returns the system host name.

```
[root@node ~]# lvm systemid
system ID: node.example.com
```

## Creating a Logical Volume

After that initial configuration, use the LVM commands as usual on one of the cluster nodes. The following example creates a logical volume and an XFS file system.

```
[root@node ~]# pvcreate /dev/sdb
[root@node ~]# vgcreate shared_vg /dev/sdb
[root@node ~]# lvcreate -L 1G -n ha_lv shared_vg
[root@node ~]# mkfs -t xfs /dev/shared_vg/ha_lv
```

Use the `vgdisplay` command to display the system ID of the volume group.

```
[root@node ~]# vgdisplay shared_vg
--- Volume group ---
VG Name          shared_vg
System ID      node.example.com
Format           lvm2
Metadata Areas   1
...output omitted...
```

## Adding a High Availability LVM Resource to a Cluster

The Red Hat High Availability cluster comes with the LVM-activate resource agent. That agent manages an HA-LVM volume group and is responsible for activating and deactivating the volume group and its logical volumes.

For the cluster to manage your volume group, use the `pcs resource create` command in conjunction with the LVM-activate resource agent.

```
[root@node ~]# pcs resource create halvm LVM-activate \
> vgname=shared_vg vg_access_mode=system_id --group=hafs
```

Because the LVM-activate resource does not mount the file system on top of LVM, you usually need to create an additional Filesystem resource to perform that operation. Group the two resources in a resource group, and create the LVM-activate resource first to ensure that the LVM-activate resource starts before and stops after the Filesystem resource.

```
[root@node ~]# pcs resource create xfsfs Filesystem \
> device=/dev/shared_vg/ha_lv directory=/data fstype=xfs --group=hafs
```



### References

[lvm系統\(7\) man page](#)

For more information, refer to the *LVM logical volumes in a Red Hat high availability cluster* section in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#con\\_HA-lvm-shared-volumes-overview-of-high-availability](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#con_HA-lvm-shared-volumes-overview-of-high-availability)

## ► Guided Exercise

# Managing High Availability LVM

In this exercise, you will configure a shared storage device to provide your cluster with an HA-LVM logical volume cluster resource, formatted with an XFS file system, that can be used by one cluster node at a time.

## Outcomes

You should be able to configure a cluster file-system resource to use a highly available logical volume for its file system.

## Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start lvm-ha
```

This command deploys a three-node cluster on the **nodea**, **nodeb**, and **nodec** machines. It configures a block device named **/dev/mapper/mpatha**, which provides multiple paths to a shared iSCSI storage device for redundancy. The command also prepares an Ansible Playbook on the **workstation** machine to help you configure the LVM system ID source on the cluster nodes.

## Instructions

- 1. Prepare the **nodea**, **nodeb**, and **nodec** machines for High availability LVM (HA-LVM). To do so, configure LVM to use the machine host name for identifying the owner of volume groups.

You have two ways to perform this step. You can edit the **/etc/lvm/lvm.conf** file on each machine, locate the **system\_id\_source** parameter, and then replace its value with **uname**:

```
system_id_source = "uname"
```

Alternatively, you can use the Ansible Playbook that the exercise preparation command has deployed on the **workstation** machine under the **/home/student/labs/lvm-ha/** directory. The following steps describe the second method.

A backup of the original **/etc/lvm/lvm.conf** file is available at **/root/lvm-ha/lvm.conf** in case you make a mistake.

- 1.1. As the **student** user on the **workstation** machine, change to the **/home/student/labs/lvm-ha/** project directory.

```
[student@workstation ~]$ cd /home/student/labs/lvm-ha  
[student@workstation lvm-ha]$
```

- 1.2. Display the content of the `playbook.yml` playbook. You do not have to modify anything in that file.

```
[student@workstation lvm-ha]$ cat playbook.yml
---
- name: Ensure LVM uses the host name for identifying the owner of VGs
  hosts: nodes
  become: yes
  gather_facts: yes

  tasks:
    - name: Ensuring system_id_source is configured ①
      lineinfile:
        path: /etc/lvm/lvm.conf
        regex: "\tsystem_id_source =.*"
        line: "\tsystem_id_source = \"uname\""
        state: present
        backup: yes

    - name: Confirming LVM uses the host name as the VG system ID ②
      command:
        cmd: lvm systemid
        register: result
        changed_when: no
        failed_when: "'system ID: ' + ansible_nodename not in result['stdout']"
...

```

- ① The first task uses the `lineinfile` module to set the `system_id_source` parameter in the `/etc/lvm/lvm.conf` file. The module creates the `/etc/lvm/lvm.conf.lvm.conf.*.YYYY-MM-DD@HH:MM:SS~` backup file.
- ② The second task runs the `lvm systemid` command to verify that LVM takes the new parameter into account.

- 1.3. Run the playbook.

```
[student@workstation lvm-ha]$ ansible-playbook playbook.yml
```

- ▶ 2. On the `nodea` machine, create an LVM physical volume on the `/dev/mapper/mpatha` multipath device, which provides redundant access to the shared iSCSI storage.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
Physical volume "/dev/mapper/mpatha" successfully created.
```

- ▶ 3. On the `nodea` machine, create an LVM volume group named `clustervg`, that uses the LVM physical volume from the preceding step.

```
[root@nodea ~]# vgcreate clustervg /dev/mapper/mpatha
Volume group "clustervg" successfully created with system ID
nodea.lab.example.com
```

In the preceding output, notice that LVM uses the machine host name as the system ID for the volume group. With that configuration, only the nodea machine can interact with the volume group.

- ▶ 4. On the nodea machine, create a 1 GiB LVM logical volume named `clusterlv` in the `clustervg` volume group.

```
[root@nodea ~]# lvcreate -L 1G -n clusterlv clustervg
Logical volume "clusterlv" created.
```



### Note

Previous exercises might have left a file system signature on the logical volume. If the `lvcreate` command asks you whether to wipe the device, then answer yes.

```
[root@nodea ~]# lvcreate -L 1G -n clusterlv clustervg
WARNING: xfs signature detected on /dev/clustervg/clusterlv at offset 0. Wipe
it? [y/n]: y
Wiping xfs signature on /dev/clustervg/clusterlv.
Logical volume "clusterlv" created.
```

- ▶ 5. On the nodea machine, create an XFS file system on the `clusterlv` logical volume.

```
[root@nodea ~]# mkfs -t xfs /dev/clustervg/clusterlv
meta-data=/dev/clustervg/clusterlv isize=512    agcount=4, agsize=65536 blks
          =                      sectsz=512  attr=2, projid32bit=1
          =                      crc=1     finobt=1, sparse=1, rmapbt=0
          =                      reflink=1
data     =                      bsize=4096   blocks=262144, imaxpct=25
          =                      sunit=0    swidth=0 blks
naming   =version 2           bsize=4096   ascii-ci=0, ftype=1
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. From the nodea machine, create an LVM-activate cluster resource named `halvm` as a member of the `halvms` resource group. When the resource starts, it activates the `clustervg` volume group on the node where it is running. Because the `halvms` resource group does not exist, the cluster automatically creates it. You can copy and paste the following command from the `/root/lvm-ha/resources.txt` file.

```
[root@nodea ~]# pcs resource create halvm LVM-activate vgname=clustervg \
> vg_access_mode=system_id --group=halvms
Assumed agent name 'ocf:heartbeat:LVM-activate' (deduced from 'LVM-activate')
```

- 7. From the nodea machine, create a `Filesystem` resource named `xfsfs`, as a member of the `halvmfs` resource group. The 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
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 8. Verify that the `halvmfs` resource group is working properly.

- 8.1. Use the `pcs status` command to confirm that the resource group is successfully running on the nodea machine.

```
[root@nodea ~]# pcs status
...output omitted...
Full List of Resources:
  * fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
  * fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
  * Resource Group: halvmfs:
    * halvm (ocf::heartbeat:LVM-activate): Started nodea.private.example.com
    * xfsfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com
...output omitted...
```

- 8.2. Verify that the `clusterlv` logical volume is active and that its XFS file system is mounted under `/mnt`.

```
[root@nodea ~]# lvs
  LV      VG      Attr      LSize Pool Origin Data%  Meta%  Move Log
  clusterlv clustervg -wi-ao---- 1.00g
[root@nodea ~]# mount | grep clusterlv
/dev/mapper/clustervg-clusterlv on /mnt type xfs
  (rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,noquota)
```

- 9. To test the HA-LVM implementation, move the `halvmfs` resource group to another node, and then confirm that the logical volume is active on the new node.

- 9.1. Use the `pcs resource move` command to relocate the `halvmfs` resource group.

```
[root@nodea ~]# pcs resource move halvmfs
Warning: Creating location constraint 'cli-ban-halvmfs-on-
nodea.private.example.com' with a score of -INFINITY for resource halvmfs on
nodea.private.example.com.
This will prevent halvmfs from running on nodea.private.example.com until the
constraint is removed
This will be the case even if nodea.private.example.com is the last node in the
cluster
```

- 9.2. Identify the node that is now running the `halvmfs` resource group.

```
[root@nodea ~]# pcs status
...output omitted...
Full List of Resources:
  * fence_nodea  (stonith:fence_ipmilan):    Started nodea.private.example.com
  * fence_nodeb  (stonith:fence_ipmilan):    Started nodeb.private.example.com
  * fence_nodec  (stonith:fence_ipmilan):    Started nodec.private.example.com
  * Resource Group: halvmsfs:
    * halvm   (ocf::heartbeat:LVM-activate):  Started nodeb.private.example.com
    * xfsfs   (ocf::heartbeat:Filesystem):      Started nodeb.private.example.com
...output omitted...
```

The preceding output shows that the group is running on the nodeb machine. On your system, the group might be running on nodec instead.

- 9.3. Confirm that the LVM logical volume is only visible on the node running the resource group and that the cluster has only mounted the XFS file system on that node.

```
[root@nodea ~]# lvs
[root@nodea ~]# mount | grep /mnt
[root@nodea ~]#
```

```
[root@nodeb ~]# lvs
  LV      VG      Attr      LSize Pool Origin Data%  Meta%  Move Log
  clusterlv clustervg -wi-ao---- 1.00g
[root@nodeb ~]# mount | grep /mnt
/dev/mapper/clustervg-clusterlv on /mnt type xfs
  (rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,noquota)
```

```
[root@nodec ~]# lvs
[root@nodec ~]# mount | grep /mnt
[root@nodec ~]#
```

- 9.4. For cleanup purposes, you can remove the location constraint that the cluster created when you moved the resource group.

```
[root@nodea ~]# pcs resource clear halvmsfs
Removing constraint: cli-ban-halvmsfs-on-nodea.private.example.com
```

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish lvm-ha
```

This concludes the section.

# Managing LVM Shared Volume Groups

## Objectives

After completing this section, you should be able to create a logical volume from a shared volume group that all nodes can use at the same time in a high availability cluster.

## Describing LVM Shared Volume Groups

LVM shared volume groups allow the use of regular LVM volume groups and logical volumes on shared storage. In contrast to HA-LVM, where the volume group is only accessible from one cluster node at a time, an LVM shared volume group and its logical volumes are usable from all cluster nodes at the same time. In conjunction with the GFS2 file system, LVM shared volume groups give an application concurrent read/write access to its data from all the cluster nodes.



### Warning

Do not use a nonclustered file system such as ext4 or XFS with LVM shared volume groups. Doing so can result in file system corruption or inconsistencies when attempting to read data.

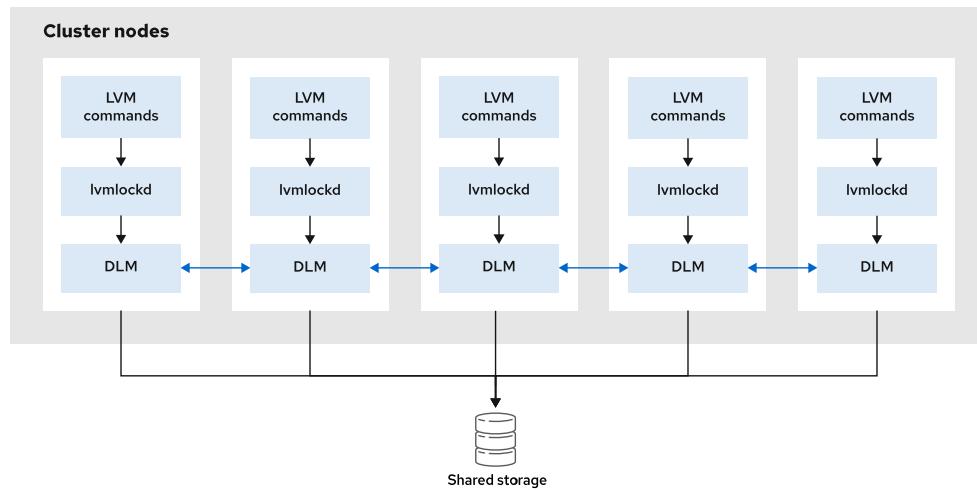
Instead, use a cluster-aware file system, such as GFS2. With GFS2, you can concurrently mount and access the file system on multiple nodes.

## Identifying LVM Shared Volume Groups Components

When using LVM shared volume groups, the LVM commands interact with a local LVM Locking Daemon (`lvmlockd`) to coordinate metadata changes with all cluster nodes. An `lvmlockd` process must run on each cluster node. `lvmlockd` is part of the Resilient Storage Add-On and the `lvm2-lockd` RPM package provides the software.

For the `lvmlockd` processes running on the different nodes to synchronize their operations, `lvmlockd` relies on the *Distributed Lock Manager (DLM)*. DLM is composed of the `dlm` kernel module and the `dlm_controld` process, which pilots the module. In turn, the `dlm_controld` process relies on the cluster infrastructure for node memberships. The `dlm` RPM package provides DLM.

The following graphic shows the interactions between all those LVM shared volume group components.



**Figure 10.2: An overview of the LVM shared volume groups architecture**

After the DLM and `lvmlockd` infrastructure is in place, you use the traditional `pvccreate`, `vgcreate`, and `lvcreate` commands to manage LVM shared volume groups.

## Preparing for LVM Shared Volume Groups

As a first step, and because DLM relies on a cluster infrastructure, ensure that you have deployed such an infrastructure, including working fencing devices.

Install the `dlm` and `lvm2-lockd` RPM packages on each node.

```
[root@node ~]# yum install dlm lvm2-lockd
```

The next step is to start the `dlm_controld` and `lvmlockd` processes on all the nodes. Because you have a cluster infrastructure in place, use it to manage those processes. Red Hat High Availability cluster also comes with the `ocf:pacemaker:controld` and `ocf:heartbeat:lvmlockd` resource agents to control the two processes. In a resource group, create the two resources in the correct order so that DLM starts before `lvmlockd`. The following example creates the `dlm` and `lvmlockd` resources in the `lock_group` resource group. The `on-fail` parameter set to `fence` instructs the cluster to fence the node if one of the processes fails instead of trying to restart it.

```
[root@node ~]# pcs resource create dlm ocf:pacemaker:controld \
>   op monitor interval=30s on-fail=fence --group=lock_group
[root@node ~]# pcs resource create lvmlockd ocf:heartbeat:lvmlockd \
>   op monitor interval=30s on-fail=fence --group=lock_group
```

Because the processes must run on all the cluster nodes, clone the resource group. Set the `interleave` parameter to `true` so that the `lvmlockd` process can start as soon as the `dlm_controld` process has started on the local node. Without this parameter, the `lvmlockd` resource waits for all the `dlm_controld` processes to start on all the nodes.

```
[root@node ~]# pcs resource clone lock_group interleave=true
```

Use the `pcs status --full` command to view all information about the cluster and cluster resources. The following output shows you that the `lock_group` resource group runs on the three nodes.

```
[root@node ~]# pcs status --full
...output omitted...
* Clone Set: lock_group-clone [lock_group]:
  * Resource Group: lock_group:0:
    * dlm (ocf::pacemaker:controld): Started node1.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started node1.example.com
  * Resource Group: lock_group:1:
    * dlm (ocf::pacemaker:controld): Started node2.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started node2.example.com
  * Resource Group: lock_group:2:
    * dlm (ocf::pacemaker:controld): Started node3.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started node3.example.com
...output omitted...
```



### Note

In contrast with HA-LVM, do not change the `system_id_source` parameter in the `/etc/lvm/lvm.conf` file. For LVM Shared Volume Groups, leave the value set to none, which is the default.

## Adding Logical Volumes to LVM Shared Volume Groups

When the LVM shared volume group infrastructure is in place, you can create volume groups and logical volumes using standard LVM commands. You only need to run those commands on one node: `lvmlockd` replicates the configuration on the other nodes.



### Note

When making changes to an LVM shared volume group, `lvmlockd` requires all the nodes up and working correctly.

## Creating a Logical Volume

The following example creates a physical volume on the shared `/dev/sdb` device and then creates the `lvmsharedvg` volume group with that PV. Add the `--shared` option so that the `vgcreate` command creates an LVM shared volume group instead of a regular local volume group.

```
[root@node ~]# pvcreate /dev/sdb
[root@node ~]# vgcreate --shared lvmsharedvg /dev/sdb
```

On all the nodes, start the volume group. The following example shows how to use the `vgchange` command with the `--lock-start` option. This command prepares the VG for using locks. Repeat the command on all the cluster nodes.

```
[root@node ~]# vgchange --lock-start lvmsharedvg
```

To create a logical volume, use the `lvcreate` command with the `--activate sy` option. This option allows multiple nodes to activate and use the LV simultaneously. You run the command only on one node.

```
[root@node ~]# lvcreate --activate sy -L500G -n lv1 lvmsharedvg
```

A shared logical volume like this is often used for a cluster file system such as GFS2. How to format the shared logical volume with a GFS2 file system and use it is covered later in this course.

## Adding an LVM Shared Volume Group Resource to a Cluster

To use a shared logical volume on a cluster node, you must first activate it on that node. Red Hat High Availability cluster provides the LVM-activate resource agent to perform that activation on all the cluster nodes. The following example creates the `sharedlv1` resource to activate the `lvmsharedvg/lv1` logical volume and adds the resource to a new `LVMshared_group` resource group. The example clones the resource group so that the activation occurs on all the cluster nodes.

```
[root@node ~]# pcs resource create sharedlv1 LVM-activate \
> vgnname=lvmsharedvg lvname=lv1 activation_mode=shared vg_access_mode=lvmlockd \
> --group=LVMshared_group
[root@node ~]# pcs resource clone LVMshared_group interleave=true
```

For now, the resource group contains only one resource. You usually format the logical volume with a shared file system such as GFS2 and then add a second resource to the resource group that mounts the file system on all the nodes. How to mount a GFS2 file system and create that resource is covered later in this course.

After creating that last resource group, the cluster hosts two cloned resource groups: the resource group that manages the DLM and `lvmlockd` processes and the resource group that activates the shared logical volume.

Because activating shared logical volumes relies on the DLM and `lvmlockd` processes, ensure that the resource group that manages those processes starts before the resource group that activates the shared logical volumes. To do so, add an order and a colocation constraints between the two groups.

```
[root@node ~]# pcs constraint order start \
> lock_group-clone then LVMshared_group-clone
[root@node ~]# pcs constraint colocation add \
> LVMshared_group-clone with lock_group-clone
```



## References

`lvmlockd(8)` and `dlm_controld(8)` man pages

**Knowledgebase: "Support Policies for RHEL Resilient Storage - dlm General Policies"**

<https://access.redhat.com/articles/3068921>

For more information, refer to the *Configuring a GFS2 file system in a cluster* chapter in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#proc\\_configuring-gfs2-in-a-cluster.adoc-configuring-gfs2-cluster](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#proc_configuring-gfs2-in-a-cluster.adoc-configuring-gfs2-cluster)

## ► Guided Exercise

# Managing LVM Shared Volume Groups

In this exercise, you will create a logical volume in a cluster using a shared volume group.

## Outcomes

You should be able to create cluster resources to configure a logical volume on a shared volume group.

## Before You Begin

As the student user on the **workstation** machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start lvm-shared
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines. It also configures a block device named `/dev/mapper/mpatha`, which provides multiple paths to a shared iSCSI storage device for redundancy. The command also prepares an Ansible Playbook on the **workstation** machine to help you install the required LVM packages on the cluster nodes.

## Instructions

- 1. Install the packages needed to set up an LVM shared volume group (VG) on the `nodea`, `nodeb`, and `nodec` machines.

You have two ways to perform that step. You can use the `yum` command on each machine to install the `lvm2-lockd` and `dlm` packages.

Alternatively, you can use the Ansible Playbook that the exercise preparation command deployed on the **workstation** machine under the `/home/student/labs/lvm-shared/` directory. The following steps describe the second method.

- 1.1. As the student user on the **workstation** machine, change to the `/home/student/labs/lvm-shared/` project directory.

```
[student@workstation ~]$ cd /home/student/labs/lvm-shared  
[student@workstation lvm-shared]$
```

- 1.2. Display the content of the `playbook.yml` playbook. You do not have to modify anything in that file.

```
[student@workstation lvm-shared]$ cat playbook.yml  
---  
- name: Ensure the Cluster LVM packages are installed  
  hosts: nodes  
  become: yes  
  gather_facts: no
```

```
tasks:
  - name: Ensuring the packages are installed
    yum:
      name:
        - lvm2-lockd
        - dlm
      state: present
...
...
```

- 1.3. Run the playbook.

```
[student@workstation lvm-shared]$ ansible-playbook playbook.yml
```

- ▶ 2. Create a cluster resource named `dlm` to manage the Distributed Lock Manager (DLM) control daemon. Create another resource named `lvmlockd` to manage the `lvmlockd` daemon on the nodes. Add both resources to the `locking` resource group. You can copy and paste the following commands from the `/root/lvm-shared/resources.txt` file.

- 2.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 2.2. Use the `pcs resource create` command to create a resource named `dlm` of type `ocf:pacemaker:controld`. Add the resource to the `locking` resource group.

```
[root@nodea ~]# pcs resource create dlm ocf:pacemaker:controld \
> op monitor interval=30s on-fail=fence --group=locking
[root@nodea ~]#
```

- 2.3. Create a resource named `lvmlockd` of type `ocf:heartbeat:lvmlockd`. Add the resource to the `locking` resource group.

```
[root@nodea ~]# pcs resource create lvmlockd ocf:heartbeat:lvmlockd \
> op monitor interval=30s on-fail=fence --group=locking
[root@nodea ~]#
```

- ▶ 3. Clone the `locking` resource group so that the cluster starts the group's resources on all cluster nodes.

- 3.1. Use the `pcs resource clone` command to clone the `locking` resource group. Set the `interleave` clone option to `true`.

```
[root@nodea ~]# pcs resource clone locking interleave=true
[root@nodea ~]#
```

- 3.2. Use the `pcs status --full` command to verify your work.

```
[root@nodea ~]# pcs status --full
...output omitted...
Full List of Resources:
  * fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
  * fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
  * Clone Set: locking-clone [locking]:
    * Resource Group: locking:0:
      * dlm (ocf::pacemaker:controld): Started nodea.private.example.com
      * lvmlockd (ocf::heartbeat:lvmlockd): Started nodea.private.example.com
    * Resource Group: locking:1:
      * dlm (ocf::pacemaker:controld): Started nodeb.private.example.com
      * lvmlockd (ocf::heartbeat:lvmlockd): Started nodeb.private.example.com
    * Resource Group: locking:2:
      * dlm (ocf::pacemaker:controld): Started nodec.private.example.com
      * lvmlockd (ocf::heartbeat:lvmlockd): Started nodec.private.example.com
...output omitted...
```

Notice that the cluster starts the resources on all three nodes.

- ▶ 4. On the nodea machine, create an LVM physical volume on the /dev/mapper/mpatha multipath device and then create a shared volume group named sharedvg that uses that physical volume. When done, register the volume group with the LVM lock space on the other two nodes.
  - 4.1. On the nodea machine, create an LVM physical volume on the /dev/mapper/mpatha multipath device, which provides redundant access to the shared iSCSI storage.

```
[root@nodea ~]# pvcreate /dev/mapper/mpatha
Physical volume "/dev/mapper/mpatha" successfully created.
```

- 4.2. On the nodea machine, create a shared LVM volume group, named sharedvg, that uses the LVM physical volume from the preceding step.

```
[root@nodea ~]# vgcreate --shared sharedvg /dev/mapper/mpatha
Volume group "sharedvg" successfully created
VG sharedvg starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

- 4.3. In a separate terminal, log in to the nodeb machine and then register the volume group with the LVM lock space.

```
[student@workstation ~]$ ssh nodeb
...output omitted...
[student@nodeb ~]$ sudo -i
[sudo] password for student: student
[root@nodeb ~]# vgchange --lock-start sharedvg
VG sharedvg starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

- 4.4. In another separate terminal, log in to the `nodec` machine and then register the volume group with the LVM lock space.

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo -i
[sudo] password for student: student
[root@nodec ~]# vgchange --lock-start sharedvg
  VG sharedvg starting dlm lockspace
  Starting locking. Waiting until locks are ready...
```

- 5. On the `nodea` machine, create a 5 GiB LVM logical volume named `sharedlv1` in the `sharedvg` volume group. Activate the logical volume with a shared lock so that all three nodes can use it concurrently.

```
[root@nodea ~]# lvcreate --activate sy -L5G -n sharedlv1 sharedvg
Logical volume "sharedlv1" created.
```

- 6. Create a cluster resource named `sharedlvm1` to manage the shared volume group and its logical volume. Add the resource to the `LVMshared` resource group and then clone the resource group.

- 6.1. Use the `pcs resource create` command to create the `sharedlvm1` resource of type `LVM-activate`. Add the resource to the `LVMshared` resource group.

```
[root@nodea ~]# pcs resource create sharedlvm1 LVM-activate vgname=sharedvg \
> lvname=sharedlv1 activation_mode=shared vg_access_mode=lvmlockd \
> --group=LVMshared
Assumed agent name 'ocf:heartbeat:LVM-activate' (deduced from 'LVM-activate')
```

- 6.2. Clone the `LVMshared` resource group so that the cluster starts its resources on all nodes. Set the `interleave` clone option to `true`.

```
[root@nodea ~]# pcs resource clone LVMshared interleave=true
[root@nodea ~]#
```

- 6.3. Use the `pcs status --full` command to verify your work.

```
[root@nodea ~]# pcs status --full
...output omitted...
* Clone Set: locking-clone [locking]:
  * Resource Group: locking:0:
    * dlm (ocf::pacemaker:controld): Started nodea.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodea.private.example.com
  * Resource Group: locking:1:
    * dlm (ocf::pacemaker:controld): Started nodeb.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodeb.private.example.com
  * Resource Group: locking:2:
    * dlm (ocf::pacemaker:controld): Started nodec.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodec.private.example.com
* Clone Set: LVMshared-clone [LVMshared]:
  * Resource Group: LVMshared:0:
```

```
* sharedlvm1 (ocf::heartbeat:LVM-activate): Started nodea.private.example.com
* Resource Group: LVMshared:1:
  * sharedlvm1 (ocf::heartbeat:LVM-activate): Started nodeb.private.example.com
* Resource Group: LVMshared:2:
  * sharedlvm1 (ocf::heartbeat:LVM-activate): Started nodec.private.example.com
...output omitted...
```

- 7. Add a resource constraint so that the **locking** resource group starts before the **LVMshared** resource group. Add a second constraint so that the cluster collocates the two resource groups.
- 7.1. Use the **pcs constraint order** command to ensure that the **locking** resource group starts before the **LVMshared** resource group.

```
[root@nodea ~]# pcs constraint order start locking-clone then LVMshared-clone
Adding locking-clone LVMshared-clone (kind: Mandatory) (Options: first-
action=start then-action=start)
```

- 7.2. Use the **pcs constraint colocation** command to colocate the two groups.

```
[root@nodea ~]# pcs constraint colocation add LVMshared-clone with locking-clone
[root@nodea ~]#
```

- 7.3. List the constraints to verify your work.

```
[root@nodea ~]# pcs constraint
Location Constraints:
Ordering Constraints:
  start locking-clone then start LVMshared-clone (kind:Mandatory)
Colocation Constraints:
  LVMshared-clone with locking-clone (score:INFINITY)
Ticket Constraints:
```

- 8. Confirm that the **sharedlv1** logical volume is active on all three nodes.

```
[root@nodea ~]# lvs
  LV      VG      Attr      LSize Pool Origin Data%  Meta%  Move Log
  sharedlv1 sharedvg -wi-a---- 5.00g
```

```
[root@nodeb ~]# lvs
  LV      VG      Attr      LSize Pool Origin Data%  Meta%  Move Log
  sharedlv1 sharedvg -wi-a---- 5.00g
```

```
[root@nodec ~]# lvs
  LV      VG      Attr      LSize Pool Origin Data%  Meta%  Move Log
  sharedlv1 sharedvg -wi-a---- 5.00g
```

A shared logical volume like this is often used for a cluster file system such as GFS2. How to format the shared logical volume with a GFS2 file system and use it is covered later in this course.

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish lvm-shared
```

This concludes the section.

## ▶ Lab

# Configuring LVM in Clusters

In this lab, you will create a highly available logical volume and a logical volume that uses a shared volume group in your cluster.

## Outcomes

You should be able to:

- Configure a highly available logical volume and create cluster resources to manage it.
- Configure a shared volume group and the associated cluster resources.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start lvm-review
```

This command deploys a two-node cluster on the `nodea` and `nodeb` machines. It also configures a block device named `/dev/mapper/disk1`, which provides multiple paths to a shared iSCSI storage device for redundancy. You use those resources in the first part of the exercise.

The command also deploys a second two-node cluster on the `nodec` and `noded` machines. It also configures a block device named `/dev/mapper/disk2`, which provides multiple paths to a shared iSCSI storage device for redundancy. You use those resources in the second part of the exercise.

## Instructions

- Prepare the `nodea` and `nodeb` machines for High Availability LVM (HA-LVM). To do so, configure LVM to use the machine host name for identifying the owner of volume groups. On those machines, the password for the `student` user is `student` and the `root` password is `redhat`.
- The exercise preparation script created the `/dev/mapper/disk1` multipath device on the `nodea` and `nodeb` machines. That device is a shared iSCSI storage device that is hosted on the `storage` machine.  
Create an LVM physical volume on the `/dev/mapper/disk1` device, create a volume group called `data` using that physical volume, and then create a 2 GiB logical volume named `db` in the `data` volume group.  
Create an ext4 file system on the `db` logical volume.
- Configure the cluster to activate the `data` volume group and mount the file system under the `/var/lib/db` mount point. To do so, create a cluster resource named `lv` that activates the `data` volume group. Create a second resource named `ext4fs` that mounts the file system. The two resources must belong to the `halvm` resource group.

**Chapter 10 |** Configuring LVM in Clusters

4. In the second part of this exercise, you set up an LVM shared volume group (VG) on the two-node cluster running on the `nodec` and `noded` machines. On those machines, the password for the `student` user is `student` and the `root` password is `redhat`.

Prepare the cluster to manage an LVM shared volume group. To do so, install the required packages and then create the following two cluster resources:

- The `dlm` resource manages the Distributed Lock Manager (DLM) control daemon.
- The `lvmlockd` resource manages the `lvmlockd` daemon on the nodes.

The resources must belong to the `locking` cloned resource group.

5. The exercise preparation script created the `/dev/mapper/disk2` multipath device on the `nodec` and `noded` machines. That device is a shared iSCSI storage device that is hosted on the `storage` machine.

Create an LVM physical volume on the `/dev/mapper/disk2` device, and then create a shared volume group named `webdata` using that physical volume. Register the volume group with the LVM lock space on both nodes.

Create a 3 GiB LVM logical volume named `logs` in the `webdata` volume group. Activate the logical volume with a shared lock so that the two nodes can use it concurrently.

6. Create a cluster resource named `lv1` to manage the shared volume group and its logical volume. Add the resource to the `sharedVG` resource group and then clone the resource group.
7. Add a resource constraint so that the `locking` resource group starts before the `sharedVG` resource group. Add a second constraint so that the cluster collocates the two resource groups.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade lvm-review
```

## Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish lvm-review
```

This concludes the section.

## ► Solution

# Configuring LVM in Clusters

In this lab, you will create a highly available logical volume and a logical volume that uses a shared volume group in your cluster.

### Outcomes

You should be able to:

- Configure a highly available logical volume and create cluster resources to manage it.
- Configure a shared volume group and the associated cluster resources.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start lvm-review
```

This command deploys a two-node cluster on the `nodea` and `nodeb` machines. It also configures a block device named `/dev/mapper/disk1`, which provides multiple paths to a shared iSCSI storage device for redundancy. You use those resources in the first part of the exercise.

The command also deploys a second two-node cluster on the `nodec` and `noded` machines. It also configures a block device named `/dev/mapper/disk2`, which provides multiple paths to a shared iSCSI storage device for redundancy. You use those resources in the second part of the exercise.

### Instructions

- Prepare the `nodea` and `nodeb` machines for High Availability LVM (HA-LVM). To do so, configure LVM to use the machine host name for identifying the owner of volume groups. On those machines, the password for the `student` user is `student` and the `root` password is `redhat`.
  - In two separate terminals, connect to the `nodea` and `nodeb` machines, and then become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

- 1.2. On both machines, edit the /etc/lvm/lvm.conf file, locate the system\_id\_source parameter, and then replace its value with uname.

```
system_id_source = "uname"
```

- 1.3. On both machines, confirm that LVM now uses the host name of the machine as the volume group system ID.

```
[root@nodea ~]# lvm systemid  
system ID: nodea.lab.example.com
```

```
[root@nodeb ~]# lvm systemid  
system ID: nodeb.lab.example.com
```

2. The exercise preparation script created the /dev/mapper/disk1 multipath device on the nodea and nodeb machines. That device is a shared iSCSI storage device that is hosted on the storage machine.

Create an LVM physical volume on the /dev/mapper/disk1 device, create a volume group called data using that physical volume, and then create a 2 GiB logical volume named db in the data volume group.

Create an ext4 file system on the db logical volume.

- 2.1. Because both the nodea and nodeb machines access the same shared device, run the following commands only on one machine. On the nodea machine for example, create the physical volume, the volume group, and then the logical volume.

```
[root@nodea ~]# pvcreate /dev/mapper/disk1  
Physical volume "/dev/mapper/disk1" successfully created.  
[root@nodea ~]# vgcreate data /dev/mapper/disk1  
Volume group "data" successfully created with system ID nodea.lab.example.com  
[root@nodea ~]# lvcreate -L 2G -n db data  
Logical volume "db" created.
```

- 2.2. Use the lvs command to verify your work.

```
[root@nodea ~]# lvs  
LV   VG   Attr      LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert  
db   data -wi-a---- 2.00g
```

- 2.3. Create an ext4 file system on the logical volume.

```
[root@nodea ~]# mkfs.ext4 /dev/data/db  
...output omitted...
```

3. Configure the cluster to activate the **data** volume group and mount the file system under the `/var/lib/db` mount point. To do so, create a cluster resource named `lv` that activates the **data** volume group. Create a second resource named `ext4fs` that mounts the file system. The two resources must belong to the `halvm` resource group.
- 3.1. From the `nodea` machine, create an LVM-activate cluster resource named `lv` as a member of the `halvm` resource group.

```
[root@nodea ~]# pcs resource create lv LVM-activate vgname=data \
> vg_access_mode=system_id --group=halvm
Assumed agent name 'ocf:heartbeat:LVM-activate' (deduced from 'LVM-activate')
```

- 3.2. Create a `Filesystem` resource named `ext4fs` as a member of the `halvm` resource group. The resource should mount the file system on the `db` logical volume under the `/var/lib/db` mount point.

```
[root@nodea ~]# pcs resource create ext4fs Filesystem \
> device=/dev/data/db directory=/var/lib/db fstype=ext4 --group=halvm
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 3.3. Use the `pcs status` command to confirm that the resource group is successfully running.

```
[root@nodea ~]# pcs status
...output omitted...
Full List of Resources:
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
* Resource Group: halvm:
  * lv      (ocf::heartbeat:LVM-activate): Started nodea.private.example.com
  * ext4fs   (ocf::heartbeat:Filesystem):   Started nodea.private.example.com
...output omitted...
```

On your system, the resource group might be running on the `nodeb` machine. In that case, run the following step on the `nodeb` machine.

- 3.4. Verify that the `db` logical volume is active and that its `ext4` file system is mounted under `/var/lib/db`.

```
[root@nodea ~]# lvs
  LV   VG   Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  db    data -wi-ao---- 2.00g
[root@nodea ~]# mount | grep db
/dev/mapper/data-db on /var/lib/db type ext4 (rw,relatime,seclabel)
```

4. In the second part of this exercise, you set up an LVM shared volume group (VG) on the two-node cluster running on the `nodec` and `noded` machines. On those machines, the password for the `student` user is `student` and the `root` password is `redhat`.

Prepare the cluster to manage an LVM shared volume group. To do so, install the required packages and then create the following two cluster resources:

- The `dlm` resource manages the Distributed Lock Manager (DLM) control daemon.
- The `lvmlockd` resource manages the `lvmlockd` daemon on the nodes.

**Chapter 10 |** Configuring LVM in Clusters

The resources must belong to the **locking** cloned resource group.

- 4.1. In two separate terminals, connect to the **nodec** and **noded** machines and become the **root** user.

```
[student@workstation ~]$ ssh nodec  
...output omitted...  
[student@nodec ~]$ sudo -i  
[sudo] password for student: student  
[root@nodec ~]#
```

```
[student@workstation ~]$ ssh noded  
...output omitted...  
[student@noded ~]$ sudo -i  
[sudo] password for student: student  
[root@noded ~]#
```

- 4.2. On both machines, install the **dlm** and **lvm2-lockd** packages.

```
[root@nodec ~]# yum install dlm lvm2-lockd
```

```
[root@noded ~]# yum install dlm lvm2-lockd
```

- 4.3. From one machine, create a cluster resource named **dlm** to manage the Distributed Lock Manager (DLM) control daemon. Add the resource to the **locking** resource group.

```
[root@nodec ~]# pcs resource create dlm ocf:pacemaker:controld \  
> op monitor interval=30s on-fail=fence --group=locking  
[root@nodec ~]#
```

- 4.4. Create a cluster resource named **lvmlockd** to manage the **lvmlockd** daemon on the nodes. Add the resource to the **locking** resource group.

```
[root@nodec ~]# pcs resource create lvmlockd ocf:heartbeat:lvmlockd \  
> op monitor interval=30s on-fail=fence --group=locking  
[root@nodec ~]#
```

- 4.5. Clone the **locking** resource group so that the cluster starts the group's resources on all cluster nodes. Set the **interleave** clone option to **true**.

```
[root@nodec ~]# pcs resource clone locking interleave=true  
[root@nodec ~]#
```

- 4.6. Use the **pcs status --full** command to verify your work.

```
[root@nodec ~]# pcs status --full  
...output omitted...  
Full List of Resources:
```

```
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
* fence_noded (stonith:fence_ipmilan): Started noded.private.example.com
* Clone Set: locking-clone [locking]:
  * Resource Group: locking:0:
    * dlm (ocf::pacemaker:controld): Started nodec.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodec.private.example.com
  * Resource Group: locking:1:
    * dlm (ocf::pacemaker:controld): Started noded.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started noded.private.example.com
...output omitted...
```

Notice that the cluster starts the resources on the two nodes.

- The exercise preparation script created the /dev/mapper/disk2 multipath device on the nodec and noded machines. That device is a shared iSCSI storage device that is hosted on the storage machine.

Create an LVM physical volume on the /dev/mapper/disk2 device, and then create a shared volume group named webdata using that physical volume. Register the volume group with the LVM lock space on both nodes.

Create a 3 GiB LVM logical volume named logs in the webdata volume group. Activate the logical volume with a shared lock so that the two nodes can use it concurrently.

- On the nodec machine, create an LVM physical volume on the /dev/mapper/disk2 device.

```
[root@nodec ~]# pvcreate /dev/mapper/disk2
Physical volume "/dev/mapper/disk2" successfully created.
```

- Create a shared LVM volume group named webdata that uses the LVM physical volume from the preceding step.

```
[root@nodec ~]# vgcreate --shared webdata /dev/mapper/disk2
Volume group "webdata" successfully created
VG webdata starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

- Register the volume group with the LVM lock space on the noded machine.

```
[root@noded ~]# vgchange --lock-start webdata
VG webdata starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

- On the nodec machine, create the 3 GiB logs logical volume with a shared lock in the webdata volume group.

```
[root@nodec ~]# lvcreate --activate sy -L3G -n logs webdata
Logical volume "logs" created.
```

- Create a cluster resource named lv1 to manage the shared volume group and its logical volume. Add the resource to the sharedVG resource group and then clone the resource group.

**Chapter 10 |** Configuring LVM in Clusters

- 6.1. On the nodec machine, use the `pcs resource create` command to create the `lv1` resource of type `LVM-activate`. Add the resource to the `sharedVG` resource group.

```
[root@nodec ~]# pcs resource create lv1 LVM-activate vgname=webdata \
> lvname=logs activation_mode=shared vg_access_mode=lvmlockd --group=sharedVG
Assumed agent name 'ocf:heartbeat:LVM-activate' (deduced from 'LVM-activate')
```

- 6.2. Clone the `sharedVG` resource group so that the cluster starts its resources on all nodes. Set the `interleave` clone option to `true`.

```
[root@nodec ~]# pcs resource clone sharedVG interleave=true
[root@nodec ~]#
```

- 6.3. Use the `pcs status --full` command to verify your work.

```
[root@nodec ~]# pcs status --full
...output omitted...
* Clone Set: locking-clone [locking]:
  * Resource Group: locking:0:
    * dlm (ocf::pacemaker:controld): Started nodec.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodec.private.example.com
  * Resource Group: locking:1:
    * dlm (ocf::pacemaker:controld): Started noded.private.example.com
    * lvmlockd (ocf::heartbeat:lvmlockd): Started noded.private.example.com
* Clone Set: sharedVG-clone [sharedVG]:
  * Resource Group: sharedVG:0:
    * lv1 (ocf::heartbeat:LVM-activate): Started nodec.private.example.com
  * Resource Group: sharedVG:1:
    * lv1 (ocf::heartbeat:LVM-activate): Started noded.private.example.com
...output omitted...
```

7. Add a resource constraint so that the `locking` resource group starts before the `sharedVG` resource group. Add a second constraint so that the cluster collocates the two resource groups.

- 7.1. On the nodec machine, use the `pcs constraint order` command to ensure that the `locking` resource group starts before the `sharedVG` resource group.

```
[root@nodec ~]# pcs constraint order start locking-clone then sharedVG-clone
Adding locking-clone sharedVG-clone (kind: Mandatory) (Options: first-action=start
then-action=start)
```

- 7.2. Use the `pcs constraint colocation` command to colocate the two groups.

```
[root@nodec ~]# pcs constraint colocation add sharedVG-clone with locking-clone
[root@nodec ~]#
```

- 7.3. List the constraints to verify your work.

```
[root@nodec ~]# pcs constraint
Location Constraints:
Ordering Constraints:
  start locking-clone then start sharedVG-clone (kind:Mandatory)
Colocation Constraints:
  sharedVG-clone with locking-clone (score:INFINITY)
Ticket Constraints:
```

7.4. Confirm that the logs logical volume is active on the two nodes.

```
[root@nodec ~]# lvs
  LV   VG      Attr       LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
  logs webdata -wi-a---- 3.00g
```

```
[root@nodec ~]# lvs
  LV   VG      Attr       LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
  logs webdata -wi-a---- 3.00g
```

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade lvm-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish lvm-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- LVM allows you to create flexible storage by allocating space on multiple storage devices.
- Physical volumes, volume groups, and logical volumes are created with the `pvcreate`, `vgcreate`, and `lvcreate` commands.
- With HA-LVM, only one node at a time can access the volume group.
- To prepare for HA-LVM, enable VG system ID by setting the `system_id_source` parameter to `uname` in the `/etc/lvm/lvm.conf` configuration file. For LVM shared volume groups, leave this parameter set to `none`, which is the default value.
- With LVM shared volume groups, the logical volumes are usable from all cluster nodes at the same time.
- LVM shared volume groups require the DLM and `lvmlockd` services to run on all the nodes.
- The `LVM-activate` resource agent manages both HA-LVM and LVM shared volume groups. Set the `vg_access_mode` parameter to `system_id` for HA-LVM and to `lvmlockd` for LVM shared volume groups.

## Chapter 11

# Providing Storage with the GFS2 Cluster File System

### Goal

Provide tightly coupled shared storage that multiple nodes can access simultaneously, using the GFS2 cluster file system.

### Objectives

- Describe key concepts of the GFS2 file system and when to use it.
- Create a GFS2 file system in a high availability cluster.
- Extend an existing GFS2 file system and create multiple journals.

### Sections

- Describing GFS2 Concepts (and Quiz)
- Creating a GFS2 Cluster File System (and Guided Exercise)
- Managing a GFS2 Cluster File System (and Guided Exercise)

### Lab

- Providing Storage with the GFS2 Cluster File System

# Describing GFS2 Concepts

## Objectives

After completing this section, you should be able to describe key concepts of the GFS2 file system and when to use it.

## Describing Global File System 2 (GFS2)

*Global File System 2 (GFS2)* is a cluster file system that you can mount simultaneously in read/write mode on multiple cluster nodes. It provides similar features as regular file systems, such as POSIX ACLs, extended attributes, quotas, symbolic links, and SELinux support. You usually reserve GFS2 for specialized cluster-aware applications or active/active applications that require access to their data from several nodes at the same time.

To coordinate those concurrent accesses, GFS2 relies on a Red Hat High Availability cluster infrastructure to monitor and fence the nodes and on the Distributed Lock Manager (DLM) to control locks on files and directories.

Each cluster node mounting a GFS2 file system uses a separate journal. When a node fails, the cluster fences that node, and then one of the other nodes replays the journal for the failed node. You allocate those journals when you create the file system, but you can also add additional journals to an existing file system by using the `gfs2_jadd` command.



### Important

Red Hat does not support the use of GFS2 as a single-node file system or on cluster deployments of more than 16 nodes. Also, Red Hat only supports GFS2 when you create the file system on top of an LVM shared logical volume.

Do not use GFS2 when you only need to mount the file system on one node at a time. File systems such as XFS are easier to setup and manage and usually provide better performance.

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 that Red Hat supports is 100 TiB.

For most deployments, having multiple smaller GFS2 file systems makes more sense than a single large file system. On large file systems, the `fsck.gfs2` command takes a long time to complete and consumes a lot of memory. A full restore of such a file system from a backup also takes a long time.

Red Hat provides the GFS2 file system as part of the Resilient Storage Add-On. Because GFS2 relies on a cluster infrastructure, you also need the High Availability Add-On.



### 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. For more information, see the Red Hat Knowledgebase articles mentioned in the References section that follows.



### References

**Knowledgebase: "What are some examples of GFS & GFS2 workloads that should be avoided?"**

<https://access.redhat.com/solutions/41223>

**Knowledgebase: "How can Red Hat assist me in assessing the design of my RHEL High Availability or Resilient Storage cluster?"**

<https://access.redhat.com/articles/2359891>

For more information, refer to the *Planning a GFS2 file system deployment* chapter in the *Configuring GFS2 file systems* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_gfs2\\_file\\_systems/index#assembly\\_planning-gfs2-deployment-configuring-gfs2-file-systems](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_gfs2_file_systems/index#assembly_planning-gfs2-deployment-configuring-gfs2-file-systems)

## ► Quiz

# Describing GFS2 Concepts

Choose the correct answers to the following questions:

- ▶ 1. **On systems running the x86\_64 processor architecture, 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. **Which three of the following GFS2 features are supported by Red Hat? (Choose three.)**
  - a. Extended Attributes
  - b. Using GFS2 on a standalone system
  - c. POSIX ACLs
  - d. Symbolic Links
  
- ▶ 3. **What is the maximum GFS2 file system size currently supported by Red Hat?**
  - a. 16 PiB
  - b. 100 TiB
  - c. 8 EiB
  
- ▶ 4. **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

# Describing GFS2 Concepts

Choose the correct answers to the following questions:

- ▶ 1. **On systems running the x86\_64 processor architecture, 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. **Which three of the following GFS2 features are supported by Red Hat? (Choose three.)**
  - a. Extended Attributes
  - b. Using GFS2 on a standalone system
  - c. POSIX ACLs
  - d. Symbolic Links
  
- ▶ 3. **What is the maximum GFS2 file system size currently supported by Red Hat?**
  - a. 16 PiB
  - b. 100 TiB
  - c. 8 EiB
  
- ▶ 4. **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 Cluster File System

## Objectives

After completing this section, you should be able to create a GFS2 file system in a high availability cluster.

## Preparing for Global File System 2 (GFS2)

As a preliminary step, review the GFS2 requirements and prepare your cluster for GFS2 file systems.

### Verifying the Requirements

Ensure that you meet the following requirements before deploying GFS2.

#### **Red Hat High Availability cluster infrastructure**

GFS2 relies on a cluster infrastructure with working fence devices.

#### **LVM shared volume group with a shared logical volume**

Red Hat supports GFS2 when you deploy the file system on top of a logical volume in an LVM shared volume group. You must activate the logical volume on all the cluster nodes.

#### **Time protocol**

GFS2 requires that you synchronize the cluster nodes' system clock. You can deploy chrony on your systems for synchronizing with Network Time Protocol (NTP) servers.

## Gathering GFS2 Deployment Details

Collect the following pieces of information before creating a GFS2 file system.

#### **Number of cluster nodes**

The number of cluster nodes that simultaneously mount the GFS2 file system determines the number of journals you have to create. If you plan to add more nodes to the cluster in the future, then you can already prepare additional journals.

#### **Name of the cluster**

You use the cluster name when you create a GFS2 file system. Only members of this cluster can mount the file system.

#### **File system name**

Each GFS2 file system has a name that uniquely identifies it in the cluster. The name you choose when you create the GFS2 file system is limited to 16 characters.

#### **LVM shared logical volume name**

You use the logical volume device node to format and mount the GFS2 file system.

## Preparing the Cluster for GFS2 File Systems

Before creating your first GFS2 file system, prepare the cluster:

- Set the global cluster parameter `no_quorum_policy` to `freeze`. This parameter controls the cluster behavior when the quorum is lost. The default value is `stop`, which instructs the

**Chapter 11 |** Providing Storage with the GFS2 Cluster File System

cluster to stop all the resources. However, to operate, GFS2 requires the cluster to be quorate. Consequently, any attempts to stop a GFS2 file system resource fail without the quorum, ultimately resulting in a complete cluster failure.

By setting the `no-quorum-policy` cluster option to `freeze`, the cluster does nothing until it regains quorum.

```
[root@node ~]# pcs property set no-quorum-policy=freeze
```

- Install the `gfs2-utils` RPM package on all the cluster nodes.

```
[root@node ~]# yum install gfs2-utils
```

## Creating a GFS2 File System

Once all the prerequisites are in place, use the `mkfs.gfs2` command from any node to create a GFS2 file system. The following table lists the most common options to `mkfs.gfs2`.

### `mkfs.gfs2` Options

<code>-t &lt;lock_table_name&gt;</code>	The name of the locking table. This name has two parts separated by a colon: <code>&lt;clustername&gt;:&lt;fs_name&gt;</code> . Only nodes that are a member of the <code>&lt;clustername&gt;</code> cluster can mount the file system. The file system name, <code>&lt;fs_name&gt;</code> , is a unique name that identifies the file system. Limit the name you choose to 16 characters.
<code>-j &lt;number_of_journals&gt;</code>	The number of journals to create initially. You can add more journals later. Each node accessing a file system simultaneously needs one journal. If you omit the option, then it defaults to one journal.
<code>-J &lt;journal_size&gt;</code>	The size of each journal in MiB. The minimum size is 8 MiB and the maximum size is 1 GiB. By default, the <code>mkfs.gfs2</code> command computes a value based on the file system size. For better performances, Red Hat recommends a size of 128 MiB or more.

The following example shows how to create a GFS2 file system called `examplegfs2`, belonging to the `examplecluster` cluster, with three 128 MiB journals, on the `/dev/lvmsharedvg/lv1` LVM shared logical volume.

```
[root@node ~]# mkfs.gfs2 -t examplecluster:examplegfs2 \
> -j 3 -J 128 /dev/lvmsharedvg/lv1
```

## Mounting a GFS2 File System

For testing purposes, you can use the `mount` command to temporarily mount a GFS2 file system in the same way as any other typical file system. For production, you must create a cluster resource that mounts the GFS2 file system.



### Important

If you manually mount a GFS2 file system by using the `mount` command, then always unmount the file system before shutting down or restarting the system. Otherwise, the system might hang during shutdown. For the same reason, never add a GFS2 file system to the `/etc/fstab` file.

## Mounting a GFS2 File System Using a Cluster Resource

Using cluster file system resources for GFS2 file systems allows Pacemaker to manage and control those resources like any other cluster resource. It also ensures that the file systems are mounted and unmounted correctly and cleanly.

Because GFS2 relies on an LVM shared logical volume, which in turn depends on `lvmlockd` and DLM, ensure that you already have created the resources to manage those services. A preceding chapter describes that process in detail. As a reminder, the following example creates the `lock_group` resource group to manage the `dlm_controld` and `lvmlockd` processes. The `LVMshared_group` resource group activates the LVM shared logical volume on all the cluster nodes. The resource constraints guarantee that DLM and `lvmlockd` start before activating the LVM shared logical volume.

```
[root@node ~]# pcs resource create dlm ocf:pacemaker:controld \
> op monitor interval=30s on-fail=fence --group=lock_group
[root@node ~]# pcs resource create lvmlockd ocf:heartbeat:lvmlockd \
> op monitor interval=30s on-fail=fence --group=lock_group
[root@node ~]# pcs resource clone lock_group interleave=true
[root@node ~]# pcs resource create sharedlv1 LVM-activate \
> vgname=lvmsharedvg lvname=lv1 activation_mode=shared vg_access_mode=lvmlockd \
> --group=LVMshared_group
[root@node ~]# pcs resource clone LVMshared_group interleave=true
[root@node ~]# pcs constraint order start \
> lock_group-clone then LVMshared_group-clone
[root@node ~]# pcs constraint colocation add \
> LVMshared_group-clone with lock_group-clone
```

With those resources in place, you can create the file system resource. The following command creates a file system resource that mounts the `/dev/lvmsharedvg/lv1` GFS2 file system on the `/data` mount point. The command adds the resource to the `LVMshared_group` resource group. Because you cloned the `LVMshared_group` resource group, the cluster mounts and manages the file system on all the cluster nodes.

```
[root@node ~]# pcs resource create clusterfs Filesystem \
> device=/dev/lvmsharedvg/lv1 directory=/data fstype=gfs2 \
> on-fail=fence --group=LVMshared_group
```



## References

`mkfs.gfs2(8)` and `mount(8)` man pages

### Knowledgebase: "What are the recommended best practices for `mkfs.gfs2`?"

<https://access.redhat.com/solutions/2883291>

For more information, refer to the *GFS2 file systems* chapter in the *Configuring GFS2 file systems* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_gfs2\\_file\\_systems/index#assembly\\_creating-mounting-gfs2-configuring-gfs2-file-systems](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_gfs2_file_systems/index#assembly_creating-mounting-gfs2-configuring-gfs2-file-systems)

For more information, refer to the *GFS2 file systems in a cluster* chapter in the *Configuring and managing high availability clusters* guide at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#assembly\\_configuring-gfs2-in-a-cluster-configuring-and-managing-high-availability-clusters](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#assembly_configuring-gfs2-in-a-cluster-configuring-and-managing-high-availability-clusters)

## ► Guided Exercise

# Creating a GFS2 Cluster File System

In this exercise, you will create a 5 GiB GFS2 file system with three journals and mount it on nodea, nodeb, and nodec simultaneously.

## Outcomes

You should be able to configure a GFS2 file system that is mounted simultaneously on all nodes in a high availability cluster.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start gfs-create
```

This command deploys a three-node cluster on the nodea, nodeb, and nodec machines. It configures a block device named `/dev/mapper/mpatha` that provides multiple paths to a shared iSCSI storage device for redundancy. The command also creates a shared LVM volume group and logical volume on that block device, along with associated cluster resources.

## Instructions

- 1. Check that the GFS2 package is installed on the nodea, nodeb, and nodec machines. This package is installed by default in all the cluster nodes, but this step installs it in case it is not present.

- 1.1. Confirm that the GFS2 package is installed on the three nodes.

```
[root@nodea ~]# yum install gfs2-utils  
...output omitted...  
Complete!
```

```
[root@nodeb ~]# yum install gfs2-utils  
...output omitted...  
Complete!
```

```
[root@nodec ~]# yum install gfs2-utils  
...output omitted...  
Complete!
```

- 2. From the nodea machine, create a GFS2 file system on the shared logical volume that the exercise preparation command has created.

- 2.1. Connect to nodea and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

- 2.2. Use the `lvs` command to confirm that lab preparation command has created the `sharedvg` shared LVM volume group and the `sharedlv1` logical volume.

```
[root@nodea ~]# lvs
  LV      VG      Attr       LSize Pool Origin Data%  Meta%  Move Log
sharedlv1 sharedvg -wi-a----- 5.00g
```

- 2.3. Use the `pcs status` command to retrieve the name of the cluster. You need that information to create the GFS2 file system.

```
[root@nodea ~]# pcs status
Cluster name: cluster1
Cluster Summary:
  * Stack: corosync
  * Current DC: nodec.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
  * Last updated: Thu Jan 28 02:47:01 2021
  * Last change: Thu Jan 28 02:36:49 2021 by root via cibadmin on
nodea.private.example.com
  * 3 nodes configured
  * 12 resource instances configured

Node List:
  * Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

Full List of Resources:
  * fence_nodea  (stonith:fence_ipmilan):    Started nodea.private.example.com
  * fence_nodeb  (stonith:fence_ipmilan):    Started nodeb.private.example.com
  * fence_nodec  (stonith:fence_ipmilan):    Started nodec.private.example.com
  * Clone Set: locking-clone [locking]:
    * Started: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]
  * Clone Set: LVMshared-clone [LVMshared]:
    * Started: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]
...output omitted...
```

The cluster name is `cluster1`.

- 2.4. Format the `/dev/sharedvg/sharedlv1` device with a GFS2 file system. Set the number of journals to 4, one per node, and set the lock space name to `gfsdata`.

```
[root@nodea ~]# mkfs.gfs2 -j 4 -t cluster1:gfsdata /dev/sharedvg/sharedlv1
/dev/sharedvg/sharedlv1 is a symbolic link to /dev/dm-1
This will destroy any data on /dev/dm-1
```

```
Are you sure you want to proceed? [y/n] y
Discarding device contents (may take a while on large devices): Done
Adding journals: Done
Building resource groups: Done
Creating quota file: Done
Writing superblock and syncing: Done
Device: /dev/sharedvg/sharedlv1
Block size: 4096
Device size: 5.00 GB (1310720 blocks)
Filesystem size: 5.00 GB (1310718 blocks)
Journals: 4
Journal size: 16MB
Resource groups: 24
Locking protocol: "lock_dlm"
Lock table: "cluster1:gfsdata"
UUID: 088db7bf-6fd7-4448-8b99-a74bf1ac28b3
```

- 3. Create the cluster resource to automatically mount the GFS2 file system on all the nodes under the /srv/data mount point.

### 3.1. Set the no-quorum-policy cluster option to freeze.



#### Note

By default, the cluster sets the `no-quorum-policy` property to `stop`, which immediately stops all resources in the cluster when quorum is lost.

However, to work correctly, GFS2 relies on the cluster infrastructure. For example, GFS2 uses the Distributed Lock Manager (DLM), which the cluster manages as a cloned resource. Without quorum, any attempts to stop GFS2 fail, ultimately resulting in a complete cluster failure.

By setting the `no-quorum-policy` cluster option to `freeze`, the cluster does nothing until it regains quorum.

```
[root@nodea ~]# pcs property set no-quorum-policy=freeze
[root@nodea ~]#
```

- 3.2. Create a file system resource named `clusterfs` that mounts the GFS2 file system under the /srv/data mount point. Add that resource to the `LVMshared` resource group, which also activates the shared volume group and logical volume. For your convenience, you can copy and paste the following command from the `/root/gfs-create/resources.txt` file.

```
[root@nodea ~]# pcs resource create clusterfs Filesystem \
> device=/dev/sharedvg/sharedlv1 directory=/srv/data fstype=gfs2 \
> op monitor interval=10s on-fail=fence --group=LVMshared
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 3.3. Use the `pcs status --full` command to confirm that the cluster starts the `clusterfs` resource on the three nodes.

```
[root@nodea ~]# pcs status --full
...output omitted...
* Clone Set: LVMshared-clone [LVMshared]:
  * Resource Group: VMshared:0:
    * sharedlvm1 (ocf::heartbeat:LVM-activate): Started nodec.private.example.com
    * clusterfs (ocf::heartbeat:Filesystem):   Started nodec.private.example.com
  * Resource Group: LVMshared:1:
    * sharedlvm1 (ocf::heartbeat:LVM-activate): Started nodea.private.example.com
    * clusterfs (ocf::heartbeat:Filesystem):   Started nodea.private.example.com
  * Resource Group: LVMshared:2:
    * sharedlvm1 (ocf::heartbeat:LVM-activate): Started nodeb.private.example.com
    * clusterfs (ocf::heartbeat:Filesystem):   Started nodeb.private.example.com
...output omitted...
```

3.4. Confirm that the cluster mounts the GFS2 file system on the three nodes.

```
[root@nodea ~]# mount | grep gfs2
/dev/mapper/sharedvg-sharedlv1 on /srv/data type gfs2 (rw,noatime,seclabel)
```

```
[root@nodeb ~]# mount | grep gfs2
/dev/mapper/sharedvg-sharedlv1 on /srv/data type gfs2 (rw,noatime,seclabel)
```

```
[root@nodec ~]# mount | grep gfs2
/dev/mapper/sharedvg-sharedlv1 on /srv/data type gfs2 (rw,noatime,seclabel)
```

3.5. To test your new file system, write a file on the nodea machine and then read it from the nodeb machine.

```
[root@nodea ~]# echo Hello world > /srv/data/test.txt
[root@nodea ~]#
```

```
[root@nodeb ~]# cat /srv/data/test.txt
Hello world
```

## Finish

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish gfs-create
```

This concludes the section.

# Managing a GFS2 Cluster File System

## Objectives

After completing this section, you should be able to extend an existing GFS2 file system and create multiple journals.

## Adding Journals to a GFS2 File System

You cannot mount a GFS2 file system on more nodes than the file system has journals. To find out how many journals a GFS2 file system has, use the `gfs2_edit -p journals` command. The following example shows that the file system has two journals.

```
[root@node ~]# gfs2_edit -p journals /dev/lvmsharedvg/lv1
Block #Journal Status:          of 1310720 (0x140000)
----- Journal List -----
journal0: 0x12    256MB clean.
journal1: 0x223a  256MB clean.
-----
```

To add more journals, use the `gfs2_jadd` command. You specify the number of additional journals by using the `-j` option. The `-J` option indicates the size of the new journals in MiB. The default value for that option is 128 (128 MiB). You can provide the file system by its device name or by its mount point.

The `gfs2_jadd` command only works with mounted file systems. Because the command creates journals by converting free space into a journal, ensure that the file system has enough free space to accommodate the new journals.

The following example creates three 256 MiB additional journals. You run the command on only one node.

```
[root@node ~]# gfs2_jadd -j 3 -J 256 /dev/lvmsharedvg/lv1
Filesystem: /dev/lvmsharedvg/lv1
Old journals: 2
New journals: 5
```

## Growing a GFS2 File System

You can grow a GFS2 file system to take advantage of any extra space that you might have added to its LVM shared logical volume. Red Hat recommends that you back up the file system before proceeding with the growing operation.



### Note

You cannot shrink a GFS2 file system.

## Chapter 11 | Providing Storage with the GFS2 Cluster File System

Use the `gfs2_grow` command to grow a GFS2 file system. Indicate the file system by its device name or by its mount point. You can run the command even when the file system is mounted.

Adding the `-T` option enables the test mode. In that mode, the command does everything a normal grow operation would, except actually committing changes to disk.

The following example adds 10 GiB to the logical volume, runs the `gfs2_grow` command in test mode, and then grows the GFS2 file system to use the newly available space.

```
[root@node ~]# lvextend -L +10G /dev/lvmsharedvg/lv1
[root@node ~]# gfs2_grow -T /dev/lvmsharedvg/lv1
(Test mode - file system will not be changed)
...output omitted...
The file system will grow by 10 GB.
gfs2_grow complete.
[root@node ~]# gfs2_grow /dev/lvmsharedvg/lv1
```

## Repairing a GFS2 File System

Under normal circumstances, a GFS2 file system should not require a manual repair. To ensure file system integrity when a node fails, the cluster fences the node, and then another node replays its journal.

In some conditions, when the storage device fails, for example, the system might leave the GFS2 file system in a dirty or corrupted state. In that situation, use the `fsck` command to repair the file system. The `fsck` command recognizes GFS2 file systems and calls the `fsck.gfs2` command to do the actual checking. You can directly run the `fsck.gfs2` command as well.



### Warning

Before running the `fsck.gfs2` command, unmount the file system from all the cluster nodes.

When the cluster manages the file system as a cluster resource, disable the resource to unmount the file system from all the nodes. The following example disables the `clusterfs` resource. The `--wait` option instructs the command to wait for the operation to complete before returning. In this example, the command waits for a maximum of 180 seconds and then returns an error if that timer expires.

```
[root@node ~]# pcs resource disable --wait=180 clusterfs
```

After the `fsck.gfs2` command has completed, enable the resource again.

```
[root@node ~]# pcs resource enable clusterfs
```

## Making Super Block Changes

If you move a GFS2 file system to a different cluster, or if you rename your cluster, then you must update the locking table name in your GFS2 super block.

**Warning**

Before making any changes to a GFS2 super block, unmount the file system from all the cluster nodes.

Use the `tunegfs2 -l` command to display a GFS2 super block.

```
[root@node ~]# tunegfs2 -l /dev/lvmsharedvg/lv1
...output omitted...
Lock protocol: lock_dlm
Lock table: examplecluster:examplegfs2
```

Change the locking table name by using the `tunegfs2 -o locktable=<new_name>` command. The following example changes the cluster name to `newcluster`.

```
[root@node ~]# tunegfs2 -o locktable=newcluster:examplegfs2 /dev/lvmsharedvg/lv1
[root@node ~]# tunegfs2 -l /dev/lvmsharedvg/lv1
...output omitted...
Lock protocol: lock_dlm
Lock table: newcluster:examplegfs2
```

**References**

`fsck.gfs2(8)`, `gfs2_edit(8)`, `gfs2_grow(8)`, `gfs2_jadd(8)`, and `tunegfs2(8)` man pages

**Knowledgebase: "Do I need to make changes to my GFS/GFS2 file systems when renaming the cluster in RHEL?"**

<https://access.redhat.com/solutions/18430>

For more information, refer to the *GFS2 file systems* and *GFS2 file system repair* chapters in the *Configuring GFS2 file systems* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_gfs2\\_file\\_systems/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_gfs2_file_systems/index)

## ► Guided Exercise

# Managing a GFS2 Cluster File System

In this exercise, you will extend an existing GFS2 file system with more space and additional journals.

### Outcomes

A GFS2 file system with four journals that can be mounted on all cluster nodes at the same time.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start gfs-manage
```

This command deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines. It configures a block device named `/dev/mapper/mpatha` that provides multiple paths to a shared iSCSI storage device for redundancy. The command also creates a shared LVM volume group and logical volume on that block device, along with a 5 GiB GFS2 file system and the associated cluster resources. The cluster mounts the GFS2 file system under the `/srv/data` mount point on all three nodes.

### Instructions

For a future project, you plan to extend your existing three-node cluster with an extra node. In this exercise, to prepare for that operation, you configure an additional journal to the GFS2 file system so that the cluster can mount the file system on four nodes.

In the second part of the exercise, you increase the GFS2 file system's size to 7 GiB.

- 1. From the `nodea` machine, examine the number of journals on your GFS2 file system.

- 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Retrieve the name of the device path and the mount point of the GFS2 file system. You use that information in the following steps.

```
[root@nodea ~]# mount | grep gfs2  
/dev/mapper/sharedvg-sharedlv1 on /srv/data type gfs2 (rw,noatime,seclabel)
```

- 1.3. Retrieve the number of journals of the GFS2 file system.

```
[root@nodea ~]# gfs2_edit -p journals /dev/mapper/sharedvg-sharedlv1
Block #Journal Status:          of 1310720 (0x140000)
----- Journal List -----
journal0: 0x12    32MB clean.
journal1: 0x2027  32MB clean.
journal2: 0x403c  32MB clean.
```

The preceding output shows that the GFS2 file system has only three journals. As a consequence, you need to add an extra journal if you plan to mount that file system on a fourth node.

- 2. To be ready for the expansion of your cluster, add an extra journal to your GFS2 file system. Set the size of the new journal to 32 MiB. The following `gfs2_jadd` command can take up to five minutes to complete. Do not interrupt it.

```
[root@nodea ~]# gfs2_jadd -j 1 -J 32 /dev/mapper/sharedvg-sharedlv1
Filesystem: /dev/mapper/sharedvg-sharedlv1
Old journals: 3
New journals: 4
```

- 3. In this second part of the exercise, you extend the size of the GFS2 file system to 7 GiB. From the `nodea` machine, examine the size of your GFS2 file system.

```
[root@nodea ~]# df -h /srv/data
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/sharedvg-sharedlv1  5.0G  133M  4.9G  3% /srv/data
```

The size of the file system is 5 GiB. You need to extend its size by 2 GiB to get a 7 GiB file system.

- 4. Extend the `/dev/mapper/sharedvg-sharedlv1` logical volume by 2 GiB.

```
[root@nodea ~]# lvextend -L +2G /dev/mapper/sharedvg-sharedlv1
WARNING: extending LV with a shared lock, other hosts may require LV refresh.
Size of logical volume sharedvg/sharedlv1 changed from 5.00 GiB (1280 extents)
to 7.00 GiB (1792 extents).
Logical volume sharedvg/sharedlv1 successfully resized.
Refreshing LV /dev//sharedvg/sharedlv1 on other hosts...
```

- 5. Grow your GFS2 file system into the newly available space.

```
[root@nodea ~]# gfs2_grow /srv/data
FS: Mount point:          /srv/data
FS: Device:               /dev/mapper/sharedvg-sharedlv1
FS: Size:                 1310717 (0x13ffffd)
DEV: Length:              1835008 (0x1c0000)
The file system will grow by 2048MB.
gfs2_grow complete.
```

- 6. Examine the new size of your GFS2 file system.

```
[root@nodea ~]# df -h /srv/data
Filesystem           Size  Used Avail Use% Mounted on
/dev/mapper/sharedvg-sharedlv1  7.0G  133M  6.9G  2% /srv/data
```

## Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish gfs-manage
```

This concludes the section.

## ► Lab

# Providing Storage with the GFS2 Cluster File System

In this lab, you will create resources that provide a GFS2 clustered file system to all nodes in a high availability cluster.

## Outcomes

You should be able to configure a highly available clustered GFS2 file system to be mounted simultaneously on all cluster nodes.

## Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start gfs-review
```

This command deploys a two-node cluster on the `nodec` and `noded` machines. It also creates a shared LVM volume group and a logical volume, along with associated cluster resources.

## Instructions

1. Install the required package for GFS2 on the `nodec` and `noded` machines and then create a GFS2 file system on the `/dev/webdata/logs` shared logical volume that the exercise preparation command has created. Because you plan to extend the cluster to four nodes, make sure to create enough journals to accommodate for that future cluster size. Each journal should have a size of 128 MiB. For the lock table parameter, use the current cluster name and `webfs` for the lock space name.  
On your machines, the password for the `student` user is `student` and the `root` password is `redhat`.
2. Set the `no-quorum-policy` cluster option to `freeze` and then create a cluster resource to automatically mount the `/dev/webdata/logs` GFS2 file system under the `/var/log/web` mount point on the two nodes. Set the name of the resource to `gfs2fs` and add it to the `sharedVG` resource group.

## Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade gfs-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish gfs-review
```

This concludes the section.

## ► Solution

# Providing Storage with the GFS2 Cluster File System

In this lab, you will create resources that provide a GFS2 clustered file system to all nodes in a high availability cluster.

## Outcomes

You should be able to configure a highly available clustered GFS2 file system to be mounted simultaneously on all cluster nodes.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start gfs-review
```

This command deploys a two-node cluster on the `nodec` and `noded` machines. It also creates a shared LVM volume group and a logical volume, along with associated cluster resources.

## Instructions

1. Install the required package for GFS2 on the `nodec` and `noded` machines and then create a GFS2 file system on the `/dev/webdata/logs` shared logical volume that the exercise preparation command has created. Because you plan to extend the cluster to four nodes, make sure to create enough journals to accommodate for that future cluster size. Each journal should have a size of 128 MiB. For the lock table parameter, use the current cluster name and `webfs` for the lock space name.

On your machines, the password for the `student` user is `student` and the `root` password is `redhat`.

- 1.1. In two separate terminals, connect to the `nodec` and `noded` machines, and then become the `root` user.

```
[student@workstation ~]$ ssh nodec
...output omitted...
[student@nodec ~]$ sudo -i
[sudo] password for student: student
[root@nodec ~]#
```

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

- 1.2. On both machines, install the `gfs2-utils` package.

```
[root@nodec ~]# yum install gfs2-utils
```

```
[root@noded ~]# yum install gfs2-utils
```

- 1.3. From the nodec machine, retrieve the cluster name. You need that information to create the GFS2 file system.

```
[root@nodec ~]# pcs status  
Cluster name: cluster2  
...output omitted...
```

- 1.4. On the nodec machine, format the `/dev/webdata/logs` device with a GFS2 file system. Set the number of journals to 4, the journal size to 128 MiB, and the lock table name to `cluster2:webfs`.

```
[root@nodec ~]# mkfs.gfs2 -j 4 -J 128 -t cluster2:webfs /dev/webdata/logs  
/dev/webdata/logs is a symbolic link to /dev/dm-1  
This will destroy any data on /dev/dm-1  
Are you sure you want to proceed? [y/n] y  
Discarding device contents (may take a while on large devices): Done  
Adding journals: Done  
...output omitted...
```

2. Set the `no-quorum-policy` cluster option to `freeze` and then create a cluster resource to automatically mount the `/dev/webdata/logs` GFS2 file system under the `/var/log/web` mount point on the two nodes. Set the name of the resource to `gfs2fs` and add it to the `sharedVG` resource group.

- 2.1. From the nodec machine, set the `no-quorum-policy` cluster option to `freeze`.

```
[root@nodec ~]# pcs property set no-quorum-policy=freeze  
[root@nodec ~]#
```

- 2.2. Create a file system resource named `gfs2fs` that mounts the `/dev/webdata/logs` GFS2 file system under the `/var/log/web` mount point. Add that resource to the `sharedVG` resource group, which also activates the shared volume group and logical volume.

```
[root@nodec ~]# pcs resource create gfs2fs Filesystem \  
> device=/dev/webdata/logs directory=/var/log/web fstype=gfs2 \  
> options=noatime op monitor interval=10s on-fail=fence --group=sharedVG  
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 2.3. Use the `pcs status --full` command to confirm that the cluster starts the `gfs2fs` resource on the two nodes.

```
[root@nodec ~]# pcs status --full  
...output omitted...  
* Clone Set: sharedVG-clone [sharedVG]:
```

```
* Resource Group: sharedVG:0:  
  * lv1      (ocf::heartbeat:LVM-activate): Started noded.private.example.com  
  * gfs2fs  (ocf::heartbeat:Filesystem):    Started noded.private.example.com  
* Resource Group: sharedVG:1:  
  * lv1      (ocf::heartbeat:LVM-activate): Started nodec.private.example.com  
  * gfs2fs  (ocf::heartbeat:Filesystem):    Started nodec.private.example.com  
...output omitted...
```

2.4. Confirm that the cluster mounts the GFS2 file system on the two nodes.

```
[root@nodec ~]# mount | grep gfs2  
/dev/mapper/webdata-logs on /var/log/web type gfs2 (rw,noatime,seclabel)
```

```
[root@noded ~]# mount | grep gfs2  
/dev/mapper/webdata-logs on /var/log/web type gfs2 (rw,noatime,seclabel)
```

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade gfs-review
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish gfs-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- GFS2 relies on a Red Hat High Availability cluster infrastructure and requires an LVM shared logical volume.
- The `mkfs.gfs2` command creates GFS2 file systems. The `-t cluster-name:fs-name` option provides the cluster and the file system name. The `-j N` option indicates the number of journals to create.
- The `gfs2_jadd` command adds journals to an existing GFS2 file system.
- The `gfs2_grow` command grows a GFS2 file system.



## Chapter 12

# Eliminating Single Points of Failure

### Goal

Identify and eliminate single points of failure in your cluster to decrease risk and increase average service availability.

### Objectives

- Configure a high availability cluster to use multiple network links for cluster communication.
- Configure multilevel fencing.

### Sections

- Configuring Network Redundancy for Cluster Communication (and Guided Exercise)
- Configuring Multiple Fencing Device Levels (and Guided Exercise)

### Lab

- Eliminating Single Points of Failure

# Configuring Network Redundancy for Cluster Communication

## Objectives

After completing this section, you should be able to configure a high availability cluster to use multiple network links for cluster communication.

## 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 take down an entire environment. Imagine a metal chain that is made up of individual links; when one link breaks, then the entire chain fails. A typical high availability cluster can have many possible single points of failure as well.

### Hardware Single Points of Failure

The following list, although not exhaustive, does contain the most common hardware offenders.

#### Power supply

When a machine loses power, it turns off. A standard solution is to use two redundant power supplies, which are attached to two UPS systems, to different mains providers, or at least to different electrical groups. A larger data center also has backup generators to provide emergency power in the event of a longer blackout.

#### Local storage

When a hard disk, or SSD, dies, then 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.

#### Network interfaces

When a network interface card (NIC) fails, then the machine that uses that NIC loses connection to the network. A solution is to use two or more NICs in a bonded configuration.

#### Network switches

When a switch fails, then it can take out more than one machine, cutting them off from the network. A solution is to use bonded NICs that are connected to different switches.

#### Fencing hardware

If only a single fencing device is used for a host and that fencing device fails, then the cluster can no longer remove a failed machine from the cluster. This fencing failure halts service relocation, disabling the service that was running on the failed node that could not be fenced. You can solve this issue by using multiple fencing devices that are assigned to multiple fencing levels.

### Software Single Points of Failure

The following list, although not exhaustive, does contain the most common software offenders.

#### Cluster communications

A Red Hat high availability cluster relies on continuous communications between all nodes. If that communication gets interrupted, then the cluster considers the nodes as failed. You can make communications more resilient by using two networks for communication instead of one. Furthermore, you can use bonded interfaces for the different networks for further resilience.

### Shared storage connection

When a node gets cut off from shared storage, then it can probably no longer effectively provide services to consumers. For Fibre Channel storage, you can use multiple HBAs, combined with multipathing, to add resilience to the storage connection. For iSCSI, you can use multiple (bonded) network interfaces to log in to the same target over different networks, combined with multipathing. The 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 for use at the same time. If one device fails, then the cluster still considers fencing to have failed. To prevent this fencing failure, you can assign fencing devices to fence levels. If a fence device in the first level fails, then the cluster tries the second level.

## Improving Cluster Resiliency with Network Redundancy

corosync relies on `kronosnet` (or `knet`) for its network communication layer. `knet` provides a resilient and secure network abstraction layer for cluster communication.

With `knet`, corosync can use up to eight separate networks to establish connections between the nodes. When one network fails, communication can continue over the remaining networks.

`knet` requires each network link to be on a different subnet. If you want to use several network interfaces on the same subnet, then configure teaming instead.

Red Hat does not support directly connecting two machines by using a crossover Ethernet cable. Red Hat only supports one network connection when you use DLM. That limitation also applies to `lvmlockd`, LVM shared volumes groups, and GFS2, because they all rely on DLM.



#### Note

The `pcs` command and the associated `pcsd` daemon that you use to administer your cluster do not use `knet`. They communicate through the network associated with the node names that you specify when creating the cluster with the `pcs host auth` and the `pcs cluster setup` commands.

If that network fails, then the `pcs` command no longer works, even though the cluster still functions correctly because of multiple `knet` links. Use teaming if you want to protect the network that the `pcs` command is using.

## Creating a New Cluster with Multiple Links

When you create a new cluster with the `pcs cluster setup` command, you can configure multiple links by providing each machine's IP addresses on the link's subnets.

The following example creates a four-node cluster with three communication links on the `192.168.0.0/24`, `10.4.0.0/16`, and `172.16.0.0/16` subnets. Use the `addr` option after each node name to provide the machines' IP addresses on each subnet.

```
[root@node1 ~]# pcs cluster setup mycluster \
> node1.example.com addr=192.168.0.1 addr=10.4.0.101 addr=172.16.0.1 \
> node2.example.com addr=192.168.0.2 addr=10.4.0.102 addr=172.16.0.2 \
> node3.example.com addr=192.168.0.3 addr=10.4.0.103 addr=172.16.0.3 \
> node4.example.com addr=192.168.0.4 addr=10.4.0.104 addr=172.16.0.4
```

## Adding Links to an Existing Cluster

Use the `pcs cluster link add` command to add a link to an existing cluster. You must provide the IP address of each node on the new link's subnet. The following example adds a link that uses the 192.168.100.0/24 subnet to a four-node cluster.

```
[root@node1 ~]# pcs cluster link add \
> node1.example.com=192.168.100.211 \
> node2.example.com=192.168.100.212 \
> node3.example.com=192.168.100.213 \
> node4.example.com=192.168.100.214
```

## Reviewing the Link Configuration

The cluster stores the links' configuration in the `/etc/corosync/corosync.conf` file. The following example shows that each cluster node has a section that displays those links.

```
...output omitted...
node {
    ring0_addr: 192.168.0.1
    ring1_addr: 10.4.0.101
    ring2_addr: 172.16.0.1
    ring3_addr: 192.168.100.211
    name: node1.example.com
    nodeid: 1
}

node {
    ring0_addr: 192.168.0.2
    ring1_addr: 10.4.0.102
    ring2_addr: 172.16.0.2
    ring3_addr: 192.168.100.212
    name: node2.example.com
    nodeid: 2
}

node {
    ring0_addr: 192.168.0.3
    ring1_addr: 10.4.0.103
    ring2_addr: 172.16.0.3
    ring3_addr: 192.168.100.213
    name: node3.example.com
    nodeid: 3
}

node {
    ring0_addr: 192.168.0.4
```

```
ring1_addr: 10.4.0.104
ring2_addr: 172.16.0.4
ring3_addr: 192.168.100.214
name: node4.example.com
nodeid: 4
}
...output omitted...
```

The cluster identifies each link by a unique ring number, from 0 to 3 in the preceding example. To remove a link, use its ring number as an argument to the `pcs cluster link remove` command. The following example removes ring number 3, which uses the 192.168.100.0/24 subnet.

```
[root@node1 ~]# pcs cluster link remove 3
```

You also use that ring number when you troubleshoot issues in the `/var/log/cluster/corosync.log` log file.

The following line in the log file shows that a link is down on a node. From the preceding `/etc/corosync/corosync.conf` file, you can deduce that on the `node2.example.com` machine (node ID 2), the connection to the 10.4.0.0/16 subnet (ring ID 1) is down.

```
Mar 11 02:35:19 ... corosync info [KNET ] link: host: 2 link: 1 is down
```



## References

`corosync.conf(5)` man page

### Knowledgebase: "Support Policies for RHEL High Availability Clusters - Cluster Interconnect Network Interfaces"

<https://access.redhat.com/articles/3068841>

For more information, refer to *Creating a high availability cluster with multiple links* section in the *Configuring and managing high availability clusters* guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#proc\\_configure-multiple-ip-cluster-creating-high-availability-cluster](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#proc_configure-multiple-ip-cluster-creating-high-availability-cluster)

## ► Guided Exercise

# Configuring Network Redundancy for Cluster Communication

In this exercise, you will configure a new three-node cluster using multiple network links.

### Outcomes

You should be able to create a cluster that uses multiple links to improve cluster network resilience.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start spof-network
```

This command creates Ansible Playbooks on the `workstation` machine that help you deploy a three-node cluster.

### Instructions

- ▶ 1. To prepare the `nodea`, `nodeb`, and `nodec` machines for clustering, review and then run the `prep.yml` Ansible Playbook that has been deployed in the `/home/student/labs/spof-network/` directory. This Ansible Playbook installs the cluster packages on the nodes, configures the firewall, sets the `hacluster` password, starts and enables the `pcsd` service on the nodes, and authenticates the nodes in the cluster.
  - 1.1. As the student user on the `workstation` machine, change to the `/home/student/labs/spof-network/` project directory.

```
[student@workstation ~]$ cd /home/student/labs/spof-network  
[student@workstation spof-network]$
```

- 1.2. Run the playbook.

```
[student@workstation spof-network]$ ansible-playbook prep.yml
```

- ▶ 2. From your `nodea` machine, create a new three-node cluster named `cluster1` that uses your `nodea.private.example.com`, `nodeb.private.example.com`, and `nodeb.private.example.com` machines. For each node, configure two network links, one that uses the `192.168.0.0/24` network, and another that uses the `192.168.2.0/24` network. The three machines have network connections to those networks.
  - 2.1. In a separate terminal, connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 2.2. From the /etc/hosts file, retrieve the machines' IP addresses on the 192.168.0.0/24 and 192.168.2.0/24 networks.

```
[root@nodea ~]# grep -e 192.168.0 -e 192.168.2 /etc/hosts  
192.168.0.10 nodea.private.example.com  
192.168.0.11 nodeb.private.example.com  
192.168.0.12 nodec.private.example.com  
192.168.0.13 noded.private.example.com  
192.168.2.10 nodea.san02.example.com  
192.168.2.11 nodeb.san02.example.com  
192.168.2.12 nodec.san02.example.com  
192.168.2.13 noded.san02.example.com  
...output omitted...
```

- 2.3. Create the three-node cluster with redundant communication channels. You can copy and paste the following command from the /root/spof-network/cluster.txt file.

```
[root@nodea ~]# pcs cluster setup cluster1 \  
> nodea.private.example.com addr=192.168.0.10 addr=192.168.2.10 \  
> nodeb.private.example.com addr=192.168.0.11 addr=192.168.2.11 \  
> nodec.private.example.com addr=192.168.0.12 addr=192.168.2.12  
...output omitted...  
Cluster has been successfully set up.
```

- 2.4. Start the cluster and enable automatic activation of the services.

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

- 3. Set up fencing for your cluster. To do so, review and then run the `stonith.yml` Ansible Playbook in the /home/student/labs/spof-network/ directory on the workstation machine. This Ansible Playbook creates and configures the STONITH resources for fencing.

- 3.1. Run the playbook.

```
[student@workstation spof-network]$ ansible-playbook stonith.yml
```

**Chapter 12 |** Eliminating Single Points of Failure

- 4. On the nodea machine, confirm that the cluster is operational and then test your configuration by disconnecting the network interface on the 192.168.2.0/24 network. Verify that the cluster is still fully operational.
- 4.1. On the nodea machine, use the `pcs status` command to confirm that the cluster is functional.

```
[root@nodea ~]# pcs status
...output omitted...
Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

Full List of Resources:
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
...output omitted...
```

- 4.2. Use the `ip addr` command to retrieve the name of the interface connected to the 192.168.2.0/24 network.

```
[root@nodea ~]# ip addr
...output omitted...
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8942 qdisc fq_codel state UP group
default qlen 1000
    link/ether 2c:c2:60:7e:dd:03 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.10/24 brd 192.168.2.255 scope global noprefixroute eth3
        valid_lft forever preferred_lft forever
    inet6 fe80::747c:8b4b:8c0c:addd/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
...output omitted...
```

- 4.3. Use the `nmcli` command to bring down the eth3 network interface.

```
[root@nodea ~]# nmcli dev disconnect eth3
Device 'eth3' successfully disconnected.
```

- 4.4. Run the `pcs status` command again to confirm that the cluster is still operational. Notice that the nodea machine is still online.

```
[root@nodea ~]# pcs status
...output omitted...
Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

Full List of Resources:
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
...output omitted...
```

- 4.5. Search the log file of the corosync process for messages related to link issues.

```
[root@nodea ~]# grep down /var/log/cluster/corosync.log
...output omitted...
Jan 14 06:25:11 ... corosync info [KNET ] link: host: 3 link: 1 is down
Jan 14 06:25:11 ... corosync info [KNET ] link: host: 2 link: 1 is down
```

The preceding output shows that the nodea machine cannot reach hosts 2 and 3 through link 1.

- 4.6. Review the /etc/corosync/corosync.conf configuration file to identify the hosts and the link from the preceding step.

```
[root@nodea ~]# cat /etc/corosync/corosync.conf
...output omitted...
nodelist {
    node {
        ring0_addr: 192.168.0.10
        ring1_addr: 192.168.2.10
        name: nodea.private.example.com
        nodeid: 1
    }

    node {
        ring0_addr: 192.168.0.11
        ring1_addr: 192.168.2.11
        name: nodeb.private.example.com
        nodeid: 2
    }

    node {
        ring0_addr: 192.168.0.12
        ring1_addr: 192.168.2.12
        name: nodec.private.example.com
        nodeid: 3
    }
}
...output omitted...
```

The preceding output shows that link 1 is connected to the 192.168.2.0/24 network. Host 2 is the nodeb machine and host 3 is the nodec machine.

## Finish

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish spof-network
```

This concludes the section.

# Configuring Multiple Fencing Device Levels

## Objectives

After completing this section, you should be able to configure multilevel fencing.

## Configuring Multilevel Fencing

When multiple fence devices are available for a single node, you can configure them in a layered topology. For example, in an environment where IPMI over LAN and APC by Schneider Electric (formerly American Power Conversion Corporation) network power switch devices are available, the cluster can first try the `fence_ipmilan` fencing agent. If that fails, then the cluster can use the `fence_apc` fencing agent, which accesses the APC power switch to turn off the node's power.

You can implement this type of redundant fencing by organizing the STONITH resources in *fencing levels*. The cluster identifies each level by a sequential number, starting at 1. When the cluster fences a node, it first attempts to use all the fencing devices at level 1, in the order you added them to the level. If they all succeed, then the cluster considers that the node is fenced and stops there. However, if any of the fencing devices fails, then the cluster stops processing that level and continues with the fence devices at level 2, and so on.

## Creating Fence Levels

Before adding fence levels, create all the STONITH resources for the fence devices you plan to use. When done, create fence levels using the `pcs stonith level add <level> <node> <devices>` command.

The following example creates two fence levels for the `node1` machine. The first level uses the `fence_ipmi_node1` STONITH resource. The second level uses the `fence_apc_node1` STONITH resource.

```
[root@node ~]# pcs stonith level add 1 node1.example.com fence_ipmi_node1
[root@node ~]# pcs stonith level add 2 node1.example.com fence_apc_node1
```

## Managing Fence Levels

Use the `pcs stonith level` command to view the topology.

```
[root@node ~]# pcs stonith level
Target: node1.example.com
Level 1 - fence_ipmi_node1
Level 2 - fence_apc_node1
```

To remove a level for a cluster node, use the `pcs stonith level remove <level> <node>` command.

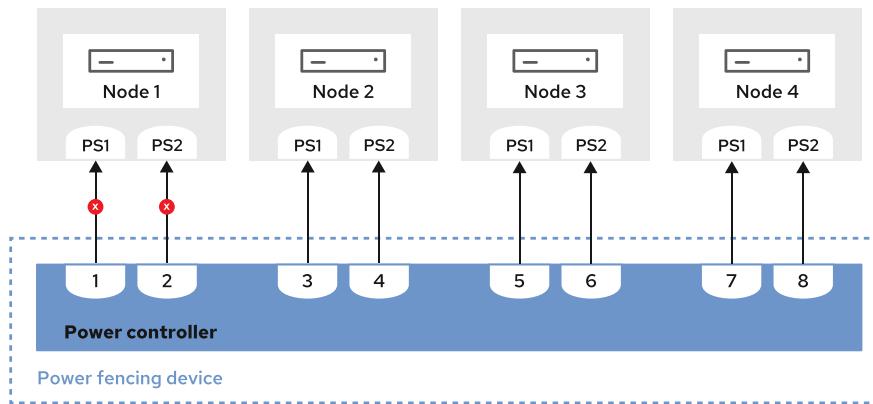
```
[root@node ~]# pcs stonith level remove 2 node1.example.com
```

To remove all the levels for a cluster node, use the `pcs stonith level clear <node>` command. If you do not provide a node name, then the command removes all the levels for all the nodes.

```
[root@node ~]# pcs stonith level clear node1.example.com
```

## Configuring Fencing for Redundant Power Supplies

A typical setup that needs careful configuration is power fencing a node that has redundant power supplies. During the fencing process, the cluster must remove power from both power supplies to effectively fence the node.



**Figure 12.1: Power Fencing Using Two Power Supplies**

The following fencing scenario is incorrect because at no point has the machine completely lost power:

1. Turn off power supply A
2. Turn on power supply A
3. Turn off power supply B
4. Turn on power supply B

To properly fence a node with redundant power supplies, create the STONITH resources, one per power supply, and then add them at the same fence level. To add multiple devices at a level, the `pcs stonith level add` command takes a comma-separated list of STONITH resources.

The following example creates two STONITH resources for fencing the `node1` machine. That machine has two power supplies connected to two APC power switches. The STONITH resources use the `fence_apc` agent to address those two power switches. Finally, the last command creates fence level 1 with the two resources.

```
[root@node ~]# pcs stonith create fence_apcA_node1 fence_apc \
> ip=10.12.0.7 username=apc password=s3cr3t pcmk_host_map="node1.example.com:2"
[root@node ~]# pcs stonith create fence_apcB_node1 fence_apc \
> ip=10.12.0.8 username=apc password=pAssw0rD pcmk_host_map="node1.example.com:1"
[root@node ~]# pcs stonith level add 1 node1.example.com \
> fence_apcA_node1,fence_apcB_node1
```

**Note**

When you add STONITH resources at the same level, the cluster first performs the "off" action for all the devices and only then executes the "on" action for all the devices. This behavior guarantees that at a point, all the node's power sources are turned off, effectively fencing the node.

In Red Hat Enterprise Linux 7.1 and earlier, the cluster does not work that way and requires a more complex configuration. See Knowledgebase: "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>] for more information.

## Adding More Devices

The `pcs stonith level` command does not provide an easy way to add a device to an existing fence level. To add a device, delete the entire level and then recreate it.

**References**

For more information, refer to the *Configuring fencing levels* section in the *Configuring and managing high availability clusters* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/8/html-single/configuring\\_and\\_managing\\_high\\_availability\\_clusters/index#proc\\_configuring-fencing-levels-configuring-fencing](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_and_managing_high_availability_clusters/index#proc_configuring-fencing-levels-configuring-fencing)

## ► Guided Exercise

# Configuring Multiple Fencing Device Levels

In this exercise, you will configure a cluster to use multiple levels of fence devices.

### Outcomes

You should be able to configure multilevel fencing in order to provide backup fencing methods if the primary fence device fails.

### Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start spof-fencing
```

This command deploys a three-node cluster on the **nodea**, **nodeb**, and **nodec** machines.

### Instructions

- 1. Inspect the fencing configuration on your cluster. This should show three separate fencing resources, one for each node.
- 1.1. Connect to **nodea** and become the **root** user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

- 1.2. Inspect the fencing configuration on your cluster.

```
[root@nodea ~]# pcs stonith config  
Resource: fence_nodea (class=stonith type=fence_ipmilan)  
  Attributes: ip=192.168.0.101 lanplus=1 password=password  
  pcmk_host_list=nodea.private.example.com power_timeout=180 username=admin  
  Operations: monitor interval=60s (fence_nodea-monitor-interval-60s)  
Resource: fence_nodeb (class=stonith type=fence_ipmilan)  
  Attributes: ip=192.168.0.102 lanplus=1 password=password  
  pcmk_host_list=nodeb.private.example.com power_timeout=180 username=admin  
  Operations: monitor interval=60s (fence_nodeb-monitor-interval-60s)  
Resource: fence_nodec (class=stonith type=fence_ipmilan)  
  Attributes: ip=192.168.0.103 lanplus=1 password=password  
  pcmk_host_list=nodec.private.example.com power_timeout=180 username=admin  
  Operations: monitor interval=60s (fence_nodec-monitor-interval-60s)
```

- 2. Your cluster is already configured to use `fence_ipmilan` as a fencing agent. The classroom also provides a secondary fence device that you can configure by using a custom fencing agent called `fence_rh436`. Create a new STONITH resource called `fence_classroom`, by using that fencing agent.

The `fence_rh436` fencing agent communicates with the virtual classroom chassis to control the classroom machines. The chassis assigns a slot number, or plug, to each machine according to the following table.

#### Classroom Machines Plug Numbers

Machine	Plug number
nodea.private.example.com	1
nodeb.private.example.com	2
nodec.private.example.com	3

The IP address of the chassis is 192.168.0.100. The user name is `admin` and the password is `password`.

- 2.1. Configure a fence device by creating a new STONITH resource called `fence_classroom` that uses the `fence_rh436` fencing agent. Use the `pcmk_host_map` parameter to associate each cluster node with its plug number. Set the time-out to 180 seconds by using the `power_timeout` parameter. You can copy and paste the following command from the `/root/spof-fencing/resources.txt` file.

The last line in the following command is very long. It has been split so that it displays correctly in the guide. If you type the command, do not add the line break nor any spaces.

```
[root@nodea ~]# pcs stonith create fence_classroom fence_rh436 \
> ip=192.168.0.100 username=admin password=password power_timeout=180 \
> pcmk_host_map="nodea.private.example.com:1;
nodeb.private.example.com:2;nodec.private.example.com:3"
[root@nodea ~]#
```

- 2.2. Use the `pcs status` command to confirm that the cluster has started the new STONITH resource.

```
[root@nodea ~]# pcs status
...output omitted...
Full List of Resources:
  * fence_nodea  (stonith:fence_ipmilan):  Started nodea.private.example.com
  * fence_nodeb  (stonith:fence_ipmilan):  Started nodeb.private.example.com
  * fence_nodec  (stonith:fence_ipmilan):  Started nodec.private.example.com
  * fence_classroom  (stonith:fence_rh436): Started nodea.private.example.com
...output omitted...
```

The preceding output shows that the resource is running on the `nodea` machine. On your system, the resource might be running on a different machine.

- ▶ 3. For each of your three nodes, create two fencing levels: a first level that uses the `fence_nodeY` fencing device, and a second level that uses the `fence_classroom` fencing device.
  - 3.1. Create the first fencing level for the `nodea` machine with the `fence_nodea` fencing device.

```
[root@nodea ~]# pcs stonith level add 1 nodea.private.example.com fence_nodea
[root@nodea ~]#
```

- 3.2. Create the second fencing level for the `nodea` machine with the `fence_classroom` fencing device.

```
[root@nodea ~]# pcs stonith level add 2 nodea.private.example.com fence_classroom
```

With that configuration, if the cluster needs to fence the `nodea` machine, then it first tries the `fence_nodea` fencing device. If the fencing fails, then the cluster uses the `fence_classroom` fencing device.

- 3.3. Repeat the process to create the fencing levels for the `nodeb` machine.

```
[root@nodea ~]# pcs stonith level add 1 nodeb.private.example.com fence_nodeb
[root@nodea ~]# pcs stonith level add 2 nodeb.private.example.com fence_classroom
[root@nodea ~]#
```

- 3.4. Repeat the process to create the fencing levels for the `nodec` machine.

```
[root@nodea ~]# pcs stonith level add 1 nodec.private.example.com fence_nodec
[root@nodea ~]# pcs stonith level add 2 nodec.private.example.com fence_classroom
[root@nodea ~]#
```

- 3.5. Use the `pcs stonith level` command to verify your work.

```
[root@nodea ~]# pcs stonith level
Target: nodea.private.example.com
  Level 1 - fence_nodea
  Level 2 - fence_classroom
Target: nodeb.private.example.com
  Level 1 - fence_nodeb
  Level 2 - fence_classroom
Target: nodec.private.example.com
  Level 1 - fence_nodec
  Level 2 - fence_classroom
```

- ▶ 4. Intentionally break the configuration of the `fence_nodec` fence device by changing the `ip` setting to `broke.example.com`. When done, try to fence the `nodec` machine and then confirm that the cluster successfully fenced the machine.

- 4.1. Intentionally break the configuration of the `fence_nodec` fence device.

```
[root@nodea ~]# pcs stonith update fence_nodec ip=broken.example.com
[root@nodea ~]#
```

- 4.2. Use the `pcs status` command to retrieve the status of the `fence_nodec` resource.

```
[root@nodea ~]# pcs status
...output omitted...
Full List of Resources:
  * fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
  * fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec (stonith:fence_ipmilan): Stopped
  * fence_classroom (stonith:fence_rh436): Started nodec.private.example.com

Failed Resource Actions:
  * fence_nodec_start_0 on nodeb.private.example.com 'error' (1): call=20,
    status='complete', exitreason='', last-rc-change='2021-02-05 00:35:58 -05:00',
    queued=0ms, exec=1155ms
  * fence_nodec_start_0 on nodea.private.example.com 'error' (1): call=24,
    status='complete', exitreason='', last-rc-change='2021-02-05 00:35:57 -05:00',
    queued=0ms, exec=1158ms
  * fence_nodec_start_0 on nodec.private.example.com 'error' (1): call=22,
    status='complete', exitreason='', last-rc-change='2021-02-05 00:35:56 -05:00',
    queued=0ms, exec=1157ms
...output omitted...
```

Notice that the `fence_nodec` resource is not available anymore. The cluster stops the resource because it cannot access the fencing device from any node.

- 4.3. Manually fence the `nodec` machine. The following command might take a few minutes to complete. Wait for the `nodec` machine to restart.

```
[root@nodea ~]# pcs stonith fence nodec.private.example.com
Node: nodec.private.example.com fenced
```

The cluster reports that it successfully fenced `nodec`. Because the fence device at the first level, `fence_nodec`, does not work, the cluster proceeds with the device at the second level, `fence_classroom`, which succeeds.

## Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish spof-fencing
```

This concludes the section.

## ▶ Lab

# Eliminating Single Points of Failure

In this lab, you will configure a high availability cluster to use multiple network links for cluster communication and multilevel fencing.

## Outcomes

You should be able to update an existing cluster to use multiple network links and multilevel combined fencing.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start spof-review
```

This command deploys a three-node cluster on the nodeb, nodec, and noded machines.

## Instructions

1. Add a second network link to the three-node cluster running on the nodeb, nodec, and noded machines. Use the 192.168.1.0/24 network for that link. The three machines have a network connection to this network according to the following table.

### Node IP Addresses on the 192.168.1.0/24 Network

Cluster node	IP address
nodeb.private.example.com	192.168.1.11
nodec.private.example.com	192.168.1.12
noded.private.example.com	192.168.1.13

On your machines, the password for the student user is `student` and the root password is `redhat`.

2. The cluster is already configured to use `fence_ipmilan` as a fencing agent. The classroom also provides a secondary fence device that you can configure by using a custom fencing agent named `fence_rh436`.

Use the `fence_rh436` fencing agent to create an additional STONITH resource named `fence_classroom`. To do so, run the `/root/spof-review/stonith.sh` script from one

**Chapter 12 |** Eliminating Single Points of Failure

cluster node. The exercise preparation program deployed that script and installed the *fence-agents-rh436* package for you.

When done, configure fencing levels so that when the cluster fences a node, it uses the *fence\_rh436* agent first, and then the *fence\_ipmilan* agent if that first agent fails. In other words, the cluster must use the STONITH resources in the following order:

**STONITH Resources Call Order**

Cluster node	STONITH resources order
nodeb.private.example.com	<i>fence_classroom</i> then <i>fence_nodeb</i>
nodec.private.example.com	<i>fence_classroom</i> then <i>fence_nodec</i>
noded.private.example.com	<i>fence_classroom</i> then <i>fence_noded</i>

**Evaluation**

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade spof-review
```

**Finish**

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish spof-review
```

This concludes the section.

## ► Solution

# Eliminating Single Points of Failure

In this lab, you will configure a high availability cluster to use multiple network links for cluster communication and multilevel fencing.

## Outcomes

You should be able to update an existing cluster to use multiple network links and multilevel combined fencing.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start spof-review
```

This command deploys a three-node cluster on the nodeb, nodec, and noded machines.

## Instructions

1. Add a second network link to the three-node cluster running on the nodeb, nodec, and noded machines. Use the 192.168.1.0/24 network for that link. The three machines have a network connection to this network according to the following table.

### Node IP Addresses on the 192.168.1.0/24 Network

Cluster node	IP address
nodeb.private.example.com	192.168.1.11
nodec.private.example.com	192.168.1.12
noded.private.example.com	192.168.1.13

On your machines, the password for the student user is `student` and the root password is `redhat`.

- 1.1. Connect to one of the cluster nodes and become the root user.

```
[student@workstation ~]$ ssh nodeb
...output omitted...
[student@nodeb ~]$ sudo -i
[sudo] password for student: student
[root@nodeb ~]#
```

- 1.2. Use the `pcs cluster link add` command to add a new network link to the cluster.

```
[root@nodeb ~]# pcs cluster link add \
> nodeb.private.example.com=192.168.1.11 \
> nodec.private.example.com=192.168.1.12 \
> noded.private.example.com=192.168.1.13
Sending updated corosync.conf to nodes...
nodec.private.example.com: Succeeded
noded.private.example.com: Succeeded
nodeb.private.example.com: Succeeded
nodeb.private.example.com: Corosync configuration reloaded
```

- 1.3. Review the /etc/corosync/corosync.conf file to verify your work.

```
[root@nodeb ~]# cat /etc/corosync/corosync.conf
...output omitted...
nodelist {
    node {
        ring0_addr: nodeb.private.example.com
        name: nodeb.private.example.com
        nodeid: 1
        ring1_addr: 192.168.1.11
    }

    node {
        ring0_addr: nodec.private.example.com
        name: nodec.private.example.com
        nodeid: 2
        ring1_addr: 192.168.1.12
    }

    node {
        ring0_addr: noded.private.example.com
        name: noded.private.example.com
        nodeid: 3
        ring1_addr: 192.168.1.13
    }
}
...output omitted...
```

2. The cluster is already configured to use `fence_ipmilan` as a fencing agent. The classroom also provides a secondary fence device that you can configure by using a custom fencing agent named `fence_rh436`.

Use the `fence_rh436` fencing agent to create an additional STONITH resource named `fence_classroom`. To do so, run the `/root/spof-review/stonith.sh` script from one cluster node. The exercise preparation program deployed that script and installed the `fence-agents-rh436` package for you.

When done, configure fencing levels so that when the cluster fences a node, it uses the `fence_rh436` agent first, and then the `fence_ipmilan` agent if that first agent fails. In other words, the cluster must use the STONITH resources in the following order:

**STONITH Resources Call Order**

<b>Cluster node</b>	<b>STONITH resources order</b>
nodeb.private.example.com	fence_classroom then fence_nodeb
nodec.private.example.com	fence_classroom then fence_nodec
noded.private.example.com	fence_classroom then fence_noded

- 2.1. Run the `/root/spof-review/stonith.sh` script to create the new STONITH resource.

```
[root@nodeb ~]# /root/spof-review/stonith.sh
[root@nodeb ~]#
```

- 2.2. Use the `pcs status` command to list the STONITH resources in the cluster.

```
[root@nodeb ~]# pcs status
...output omitted...
Full List of Resources:
  * fence_nodeb  (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec  (stonith:fence_ipmilan): Started nodec.private.example.com
  * fence_noded   (stonith:fence_ipmilan): Started noded.private.example.com
  * fence_classroom (stonith:fence_rh436): Started nodeb.private.example.com
...output omitted...
```

- 2.3. Create the first fencing level for the nodeb machine by using the `fence_classroom` STONITH resource.

```
[root@nodeb ~]# pcs stonith level add 1 nodeb.private.example.com fence_classroom
[root@nodeb ~]#
```

- 2.4. Create the second fencing level for the nodeb machine by using the `fence_nodeb` STONITH resource.

```
[root@nodeb ~]# pcs stonith level add 2 nodeb.private.example.com fence_nodeb
[root@nodeb ~]#
```

- 2.5. Repeat the process to create the fencing levels for the nodec machine.

```
[root@nodeb ~]# pcs stonith level add 1 nodec.private.example.com fence_classroom
[root@nodeb ~]# pcs stonith level add 2 nodec.private.example.com fence_nodec
[root@nodeb ~]#
```

- 2.6. Repeat the process to create the fencing levels for the noded machine.

```
[root@nodeb ~]# pcs stonith level add 1 noded.private.example.com fence_classroom
[root@nodeb ~]# pcs stonith level add 2 noded.private.example.com fence_noded
[root@nodeb ~]#
```

**Chapter 12 |** Eliminating Single Points of Failure

2.7. Use the `pcs stonith level` command to verify your work.

```
[root@nodeb ~]# pcs stonith level
Target: nodeb.private.example.com
  Level 1 - fence_classroom
  Level 2 - fence_nodeb
Target: nodec.private.example.com
  Level 1 - fence_classroom
  Level 2 - fence_nodec
Target: noded.private.example.com
  Level 1 - fence_classroom
  Level 2 - fence_noded
```

## Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade spof-review
```

## Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish spof-review
```

This concludes the section.

# Summary

---

In this chapter, you learned:

- Multiple network links can be configured during cluster creation by specifying the additional node addresses.
- Additional links can also be added to an existing cluster using the `pcs cluster link add` command.
- Multiple fence levels can be configured, with multiple fence devices in each level.
- If one fence device in a level fails, then that level is considered as failed. In a multi-level configuration, the device in the next level is then attempted.
- Fence devices inside a level are executed in the order in which they were added at creation time.



## Chapter 13

# Comprehensive Review

### Goal

Review tasks from *Red Hat High Availability Clustering*

### Objectives

- Review tasks from *Red Hat High Availability Clustering*

### Sections

- Comprehensive Review

### Lab

- Lab: Configuring a High Availability Cluster
- Lab: Configuring Cluster Resources on a Two-node Cluster
- Lab: Deploying a GFS2 File System

# Comprehensive Review

---

## Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills learned in *Red Hat High Availability Clustering*.

## Reviewing Red Hat High Availability Clustering

Before beginning the comprehensive review for this course, you should be comfortable with the topics covered in each chapter.

You 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 Red Hat Enterprise Linux high availability cluster.
- Install and start a small high availability cluster.

### **Chapter 2, Managing Cluster Nodes and Quorum**

Manage node membership in the cluster and describe how it impacts cluster operation.

- Add and remove nodes from an existing high availability cluster.
- Explain how quorum operates and how it protects a high availability cluster from errors.
- Adjust the way quorum is calculated and describe when this might be necessary.

### **Chapter 3, Isolating Malfunctioning Cluster Nodes**

Isolate unresponsive cluster nodes to protect data and recover services and resources after a failure.

- Describe what fencing is and how it is used to protect data during cluster node failures.
- Identify supported fencing devices and test them.
- Configure the cluster to use the correct fencing device when fencing a cluster node.

### **Chapter 4, Creating and Configuring Resources**

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

- Create and configure a single high availability cluster resource.
- Assemble a resource group from multiple related resources.
- Manually control and relocate resource groups in a high availability cluster.

**Chapter 13 |** Comprehensive Review

- Set constraints that control on which cluster nodes resources and resource groups can run.

## **Chapter 5, Troubleshooting High Availability Clusters**

Identify, diagnose, and fix cluster problems.

- Inspect and configure log files that the high availability cluster keeps to help diagnose problems.
- Configure the cluster software to send notifications when important events occur.
- Perform common troubleshooting steps to diagnose a failed resource.
- Diagnose problems that relate to cluster network communication.

## **Chapter 6, Automating Cluster and Resource Deployment**

Deploy a new high availability cluster and cluster resources by using Ansible automation.

- Use Ansible to deploy a high availability cluster.
- Use Ansible to deploy and configure resources and resource groups in a high availability cluster.

## **Chapter 7, Managing Two-node Clusters**

Operate two-node clusters, identifying and avoiding issues that are specific to a two-node cluster configuration.

- Describe the special considerations to follow when configuring a two-node cluster.
- Configure a two-node high availability cluster.
- Configure a high availability cluster with a quorum device to help resolve quorum ties.

## **Chapter 8, Accessing iSCSI Storage**

Configure iSCSI initiators on your servers to access block-based storage devices that network storage arrays or Ceph storage clusters provide.

- Explain what iSCSI is, define key iSCSI terms, and describe how it is useful for cluster storage.
- Configure an iSCSI initiator to access a network-based block device, format a new iSCSI device with a file system, configure it for use at boot, and safely discontinue the use of an existing iSCSI block device.

## **Chapter 9, Accessing Storage Devices Resiliently**

Configure resilient access to storage devices that have multiple access paths.

- Describe storage multipathing and its terminology, and explain when it should be used.
- Configure resilient access to a storage device by using a multipath device.
- Monitor, troubleshoot, and test multipath devices.

## **Chapter 10, Configuring LVM in Clusters**

Select the correct Logical Volume Management (LVM) configuration for use in your cluster, and configure and manage it.

- Describe the basic logical volume management concepts.

**Chapter 13 |** Comprehensive Review

- Configure a highly available logical volume as a resource that one node uses at a time in a high availability cluster.
- Create a logical volume from a shared volume group that all nodes can use at the same time in a high availability cluster.

## **Chapter 11, Providing Storage with the GFS2 Cluster File System**

Provide tightly coupled shared storage that multiple nodes can access simultaneously, using the GFS2 cluster file system.

- Describe key concepts of the GFS2 file system and when to use it.
- Create a GFS2 file system in a high availability cluster.
- Extend an existing GFS2 file system and create multiple journals.

## **Chapter 12, Eliminating Single Points of Failure**

Identify and eliminate single points of failure in your cluster to decrease risk and increase average service availability.

- Configure a high availability cluster to use multiple network links for cluster communication.
- Configure multilevel fencing.

## ▶ Lab

# Configuring a High Availability Cluster

In this review, you will prepare a four-node cluster.

## Outcomes

You should be able to create a highly available, active-passive four-node cluster named `prod1` with fencing.

## Before You Begin

If you did not reset your `workstation`, `storage`, and `nodeX` machines at the end of the last chapter, save any work that you want to keep from earlier exercises on those machines, and reset them now.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start compreview-hacluster
```

## Instructions

Create a cluster with the following requirements. On all your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Deploy a cluster named `prod1` on the `nodea`, `nodeb`, `nodec`, and `noded` machines. Use the following names for the cluster nodes: `nodea.private.example.com`, `nodeb.private.example.com`, `nodec.private.example.com`, and `noded.private.example.com`. Make sure that the cluster automatically starts when the nodes restart. Set the password of the `hacluster` user to `redhat`. Use the following information to create redundant network links:
  - `nodea.private.example.com` with IPs `192.168.0.10` and `192.168.2.10`.
  - `nodeb.private.example.com` with IPs `192.168.0.11` and `192.168.2.11`.
  - `nodec.private.example.com` with IPs `192.168.0.12` and `192.168.2.12`.
  - `noded.private.example.com` with IPs `192.168.0.13` and `192.168.2.13`.
2. Enable the `wait for all` and the `last man standing` options in the cluster. Make sure to start the cluster.
3. Configure the cluster to use the IPMI (Intelligent Platform Management Interface) over LAN fencing agent. Because in the classroom environment the BMC devices might take a long time to reply, add the `power_timeout=180` option when you create the fence resources. To access those devices, use `admin` for the user name with `password` for the password. Use the following IP address and name of each device:
  - `nodea.private.example.com` with IP `192.168.0.101`, and name `fence_nodea`.
  - `nodeb.private.example.com` with IP `192.168.0.102`, and name `fence_nodeb`.

- nodec.private.example.com with IP 192.168.0.103, and name fence\_nodec.
- noded.private.example.com with IP 192.168.0.104, and name fence\_noded.

## Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade compreview-hacluster
```

## Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-hacluster
```

This concludes the section.

## ► Solution

# Configuring a High Availability Cluster

In this review, you will prepare a four-node cluster.

## Outcomes

You should be able to create a highly available, active-passive four-node cluster named `prod1` with fencing.

## Before You Begin

If you did not reset your `workstation`, `storage`, and `nodeX` machines at the end of the last chapter, save any work that you want to keep from earlier exercises on those machines, and reset them now.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start compreview-hacluster
```

## Instructions

Create a cluster with the following requirements. On all your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Deploy a cluster named `prod1` on the `nodea`, `nodeb`, `nodec`, and `noded` machines. Use the following names for the cluster nodes: `nodea.private.example.com`, `nodeb.private.example.com`, `nodec.private.example.com`, and `noded.private.example.com`. Make sure that the cluster automatically starts when the nodes restart. Set the password of the `hacluster` user to `redhat`. Use the following information to create redundant network links:
  - `nodea.private.example.com` with IPs `192.168.0.10` and `192.168.2.10`.
  - `nodeb.private.example.com` with IPs `192.168.0.11` and `192.168.2.11`.
  - `nodec.private.example.com` with IPs `192.168.0.12` and `192.168.2.12`.
  - `noded.private.example.com` with IPs `192.168.0.13` and `192.168.2.13`.
- 1.1. In separate terminals, connect to the nodes and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

```
[student@workstation ~]$ ssh nodec  
...output omitted...  
[student@nodec ~]$ sudo -i  
[sudo] password for student: student  
[root@nodec ~]#
```

```
[student@workstation ~]$ ssh noded  
...output omitted...  
[student@noded ~]$ sudo -i  
[sudo] password for student: student  
[root@noded ~]#
```

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

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

Repeat this process on all cluster nodes.

- 1.3. Install the **pcs** and **fence-agents-ipmilan** RPM packages on the nodea, nodeb, nodec, and noded machines.

```
[root@nodea ~]# yum install pcs fence-agents-ipmilan
```

Repeat this process on all cluster nodes.

- 1.4. Enable and start the **pcsd** service on the nodea, nodeb, nodec, and noded machines.

```
[root@nodea ~]# systemctl enable --now pc sd  
Created symlink /etc/systemd/system/multi-user.target.wants/pc sd.service → /usr/  
lib/systemd/system/pc sd.service.
```

Repeat this process on all cluster nodes.

- 1.5. Change the password of the hacluster system user to **redhat** on the nodea, nodeb, nodec, and noded machines.

```
[root@nodea ~]# echo redhat | passwd --stdin hacluster  
Changing password for user hacluster.  
passwd: all authentication tokens updated successfully.
```

Repeat this process on all cluster nodes.

- 1.6. From the nodea machine, authenticate the cluster nodes nodea.private.example.com, nodeb.private.example.com, nodec.private.example.com, and noded.private.example.com.

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

- 1.7. Create the cluster prod1 with the cluster nodes nodea.private.example.com, nodeb.private.example.com, nodec.private.example.com, and noded.private.example.com.

```
[root@nodea ~]# pcs cluster setup prod1 \
> nodea.private.example.com addr=192.168.0.10 addr=192.168.2.10 \
> nodeb.private.example.com addr=192.168.0.11 addr=192.168.2.11 \
> nodec.private.example.com addr=192.168.0.12 addr=192.168.2.12 \
> noded.private.example.com addr=192.168.0.13 addr=192.168.2.13
...output omitted...
Cluster has been successfully set up.
```

- 1.8. Enable automatic startup of the cluster on all configured cluster nodes.

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

2. Enable the `wait_for_all` and the `last_man_standing` options in the cluster. Make sure to start the cluster.

- 2.1. Enable the `last_man_standing` and `wait_for_all` options.

```
[root@nodea ~]# pcs quorum update wait_for_all=1 last_man_standing=1
Checking corosync is not running on nodes...
noded.private.example.com: corosync is not running
nodec.private.example.com: corosync is not running
nodea.private.example.com: corosync is not running
nodeb.private.example.com: corosync is not running
Sending updated corosync.conf to nodes...
noded.private.example.com: Succeeded
nodea.private.example.com: Succeeded
nodec.private.example.com: Succeeded
noded.private.example.com: Succeeded
```

- 2.2. Start the cluster services on all cluster nodes.

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

2.3. Use the `pcs status` command to confirm that the cluster is functional.

```
[root@nodea ~]# pcs status
Cluster name: prod1

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
* Stack: corosync
* Current DC: nodea.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
* Last updated: Wed Apr 21 17:32:25 2021
* Last change: Wed Apr 21 17:24:28 2021 by hacluster via crmd on
nodeb.private.example.com
* 4 nodes configured
* 0 resource instances configured

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

Full List of Resources:
* No resources

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

3. Configure the cluster to use the IPMI (Intelligent Platform Management Interface) over LAN fencing agent. Because in the classroom environment the BMC devices might take a long time to reply, add the `power_timeout=180` option when you create the fence resources. To access those devices, use `admin` for the user name with `password` for the password. Use the following IP address and name of each device:
  - `nodea.private.example.com` with IP `192.168.0.101`, and name `fence_nodea`.
  - `nodeb.private.example.com` with IP `192.168.0.102`, and name `fence_nodeb`.
  - `nodec.private.example.com` with IP `192.168.0.103`, and name `fence_nodec`.
  - `noded.private.example.com` with IP `192.168.0.104`, and name `fence_noded`.
- 3.1. Add the fence devices for the virtual machines `nodea`, `nodeb`, `nodec`, and `noded` to the cluster on `nodea.private.example.com`.

```
[root@nodea ~]# pcs stonith create fence_nodea fence_ipmilan \
> pcmk_host_list=nodea.private.example.com \
> ip=192.168.0.101 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_nodeb fence_ipmilan \
> pcmk_host_list=nodeb.private.example.com \
> ip=192.168.0.102 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_nodec fence_ipmilan \
> pcmk_host_list=nodec.private.example.com \
> ip=192.168.0.103 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
[root@nodea ~]# pcs stonith create fence_noded fence_ipmilan \
> pcmk_host_list=noded.private.example.com \
> ip=192.168.0.104 \
> username=admin \
> password=password \
> lanplus=1 \
> power_timeout=180
```

### 3.2. Verify that the fence devices were added correctly to the cluster.

```
[root@nodea ~]# pcs status
Cluster name: prod1
Cluster Summary:
  * Stack: corosync
  * Current DC: nodea.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
    partition with quorum
    * Last updated: Wed Apr 21 17:41:22 2021
    * Last change: Wed Apr 21 17:38:28 2021 by root via cibadmin on
      nodea.private.example.com
    * 4 nodes configured
    * 4 resource instances configured

Node List:
  * 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_ipmilan): Started nodea.private.example.com
  * fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
  * fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
  * fence_noded (stonith:fence_ipmilan): Started noded.private.example.com
```

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

## Evaluation

As the student user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade compreview-hacluster
```

## Finish

As the student user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-hacluster
```

This concludes the section.

## ▶ Lab

# Configuring Cluster Resources on a Two-node Cluster

In this review, you will create a two-node cluster with a quorum device and deploy a secure web server on that cluster.

## Outcomes

You should be able to:

- Deploy a two-node cluster with a quorum device.
- Use Ansible Playbooks to deploy applications on cluster nodes.
- Write playbooks to create cluster resources.
- Configure resource colocations.

## Before You Begin

If you did not reset your `workstation`, `storage`, and `nodeX` machines at the end of the last chapter, save any work that you want to keep from earlier exercises on those machines, and reset them now.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start compreview-twinode
```

This command prepares an Ansible project in the `/home/student/labs/cr2/` directory on `workstation`.

## Instructions

Perform the following tasks to complete this comprehensive review exercise. On all your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Deploy a cluster named `webfarm` on the `nodea` and `nodeb` machines. Use the `private.example.com` network for cluster communications. On that network, `nodea` is defined as `nodea.private.example.com` and `nodeb` as `nodeb.private.example.com`. Set the password of the `hacluster` user to `redhat`. Start the cluster and make sure that it starts when the nodes start.
2. Prepare a quorum device on the `noded` machine. Set the password of the `hacluster` user to `redhat`.
3. Configure your cluster to use the new quorum device. To access that quorum device on the `noded` machine, use the `san01.example.com` network. On that network, `noded` is defined as `noded.san01.example.com`. Use the `ffsplit` algorithm when declaring the quorum device.

4. As the **student** user on the **workstation** machine, run the `/home/student/labs/cr2/fencing.yml` Ansible playbook. That playbook creates the fencing resources for your cluster. You do not have to change anything in that file.
5. As the **student** user on the **workstation** machine, run the `apache.yml` and `haproxy.yml` playbooks in the `/home/student/labs/cr2/` directory. Those playbooks deploy an Apache HTTP Server and an HAProxy service on all nodes. You do not have to change anything in those playbooks.
6. On the **workstation** machine, in the `/home/student/labs/cr2/` directory, complete the `cluster_apache.yml` playbook. The playbook creates a resource group named `web` that manages the Apache HTTP Server and its floating IP address. Run the playbook.
7. By using the `pcs` command on one of the cluster nodes, create a resource group named `termination`. That resource group must include the following resources:
  - A resource named `proxy` to manage the HAProxy systemd service.
  - A resource named `frontIP` to manage the `172.25.250.80/24` floating IP address.
8. Configure a constraint so that the `termination` and the `web` resource groups never run on the same node.
9. On your **workstation** machine, as **student** user, use the `curl` command to navigate to `www.lab.example.com`.

## Evaluation

As the **student** user on the **workstation** machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade compreview-twinode
```

## Finish

As the **student** user on the **workstation** machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-twinode
```

This concludes the section.

## ► Solution

# Configuring Cluster Resources on a Two-node Cluster

In this review, you will create a two-node cluster with a quorum device and deploy a secure web server on that cluster.

## Outcomes

You should be able to:

- Deploy a two-node cluster with a quorum device.
- Use Ansible Playbooks to deploy applications on cluster nodes.
- Write playbooks to create cluster resources.
- Configure resource colocations.

## Before You Begin

If you did not reset your `workstation`, `storage`, and `nodeX` machines at the end of the last chapter, save any work that you want to keep from earlier exercises on those machines, and reset them now.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start compreview-twinode
```

This command prepares an Ansible project in the `/home/student/labs/cr2/` directory on `workstation`.

## Instructions

Perform the following tasks to complete this comprehensive review exercise. On all your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Deploy a cluster named `webfarm` on the `nodea` and `nodeb` machines. Use the `private.example.com` network for cluster communications. On that network, `nodea` is defined as `nodea.private.example.com` and `nodeb` as `nodeb.private.example.com`. Set the password of the `hacluster` user to `redhat`. Start the cluster and make sure that it starts when the nodes start.
  - 1.1. Connect to `nodea` and become the `root` user.

```
[student@workstation ~]$ ssh nodea
...output omitted...
[student@nodea ~]$ sudo -i
[sudo] password for student: student
[root@nodea ~]#
```

**Chapter 13 |** Comprehensive Review

- 1.2. In a separate terminal, connect to nodeb and become the **root** user.

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

- 1.3. On both nodes, allow cluster communications to pass through the firewall.

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

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

- 1.4. On both nodes, install the **pcs** package.

```
[root@nodea ~]# yum install pcs
```

```
[root@nodeb ~]# yum install pcs
```

- 1.5. On both nodes, change the password of the **hacluster** system user to **redhat**.

```
[root@nodea ~]# passwd hacluster  
Changing password for user hacluster.  
New password: redhat  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password: redhat  
passwd: all authentication tokens updated successfully.
```

```
[root@nodeb ~]# passwd hacluster  
Changing password for user hacluster.  
New password: redhat  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password: redhat  
passwd: all authentication tokens updated successfully.
```

- 1.6. On both nodes, enable and start the **pcsd** service.

```
[root@nodea ~]# systemctl enable --now pcasd  
Created symlink /etc/systemd/system/multi-user.target.wants/pcasd.service → /usr/  
lib/systemd/system/pcasd.service.
```

```
[root@nodeb ~]# systemctl enable --now pcsd
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service → /usr/
lib/systemd/system/pcsd.service.
```

- 1.7. From your nodea machine, authenticate your nodea.private.example.com and nodeb.private.example.com machines for pcs.

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

- 1.8. Create the cluster.

```
[root@nodea ~]# pcs cluster setup webfarm \
> nodea.private.example.com nodeb.private.example.com
...output omitted...
Cluster has been successfully set up.
```

- 1.9. Start the cluster and enable automatic activation of the services.

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

- 1.10. Confirm that the cluster is active. You might have to run the `pcs status` command several times because it takes up to a minute for the nodes to be online.

```
[root@nodea ~]# pcs status
...output omitted...
Node List:
 * Online: [ nodea.private.example.com nodeb.private.example.com ]
...output omitted...
```

2. Prepare a quorum device on the noded machine. Set the password of the hacluster user to redhat.

- 2.1. In a separate terminal, connect to the noded machine and become the root user.

```
[student@workstation ~]$ ssh noded
...output omitted...
[student@noded ~]$ sudo -i
[sudo] password for student: student
[root@noded ~]#
```

- 2.2. Install the `pcs` and `corosync-qnetd` packages.

```
[root@noded ~]# yum install pcs corosync-qnetd
```

- 2.3. Change the password of the hacluster system user to redhat.

```
[root@noded ~]# passwd hacluster
Changing password for user hacluster.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

- 2.4. Enable and start the pcscd service.

```
[root@noded ~]# systemctl enable --now pcscd
Created symlink /etc/systemd/system/multi-user.target.wants/pcscd.service → /usr/
lib/systemd/system/pcscd.service.
```

- 2.5. Configure, start, and enable the quorum device.

```
[root@noded ~]# pcs qdevice setup model net --enable --start
Quorum device 'net' initialized
quorum device enabled
Starting quorum device...
quorum device started
```

- 2.6. Allow cluster communications to pass through the firewall.

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

- 2.7. Confirm that the quorum device is ready. Notice that no cluster is using it yet, and that no cluster nodes are connected.

```
[root@noded ~]# pcs qdevice status net
QNtd address: *:5403
TLS: Supported (client certificate required)
Connected clients: 0
Connected clusters: 0
```

3. Configure your cluster to use the new quorum device. To access that quorum device on the noded machine, use the san01.example.com network. On that network, noded is defined as noded.san01.example.com. Use the ffsplit algorithm when declaring the quorum device.

- 3.1. Install the corosync-qdevice package on the cluster nodes, nodea and nodeb.

```
[root@nodea ~]# yum install corosync-qdevice
```

```
[root@nodeb ~]# yum install corosync-qdevice
```

- 3.2. From the nodea machine, authenticate the noded.san01.example.com machine for pcs.

```
[root@nodea ~]# pcs host auth noded.san01.example.com  
Username: hacluster  
Password: redhat  
noded.san01.example.com: Authorized
```

- 3.3. Declare the quorum device at noded.san01.example.com. Set the algorithm parameter to ffsplit.

```
[root@nodea ~]# pcs quorum device add model net host=noded.san01.example.com \  
> algorithm=ffsplit  
...output omitted...
```

- 3.4. Use the pcs quorum config command to verify your configuration.

```
[root@nodea ~]# pcs quorum config  
Options:  
Device:  
  votes: 1  
  Model: net  
    algorithm: ffsplit  
    host: noded.san01.example.com
```

4. As the student user on the workstation machine, run the /home/student/labs/cr2/fencing.yml Ansible playbook. That playbook creates the fencing resources for your cluster. You do not have to change anything in that file.

- 4.1. As the student user on workstation, change to the /home/student/labs/cr2/ project directory.

```
[student@workstation ~]$ cd /home/student/labs/cr2  
[student@workstation cr2]$
```

- 4.2. Run the fencing.yml playbook.

```
[student@workstation cr2]$ ansible-playbook fencing.yml
```

5. As the student user on the workstation machine, run the apache.yml and haproxy.yml playbooks in the /home/student/labs/cr2/ directory. Those playbooks deploy an Apache HTTP Server and an HAProxy service on all nodes. You do not have to change anything in those playbooks.

- 5.1. Run the apache.yml playbook.

```
[student@workstation cr2]$ ansible-playbook apache.yml
```

- 5.2. Run the haproxy.yml playbook.

```
[student@workstation cr2]$ ansible-playbook haproxy.yml
```

6. On the workstation machine, in the /home/student/labs/cr2/ directory, complete the cluster\_apache.yml playbook. The playbook creates a resource group named web that manages the Apache HTTP Server and its floating IP address. Run the playbook.
- 6.1. Edit the cluster\_apache.yml playbook. That playbook uses the pcs command on one cluster node to create the apache and the ip resources, and the web resource group.

The resulting file should be displayed as follows:

```
---
- name: Create the resource group to manage Apache HTTP Server
  hosts: nodea
  become: yes
  gather_facts: no

  tasks:
    - name: Collecting the existing resources
      command:
        cmd: pcs resource config
      changed_when: false
      register: resources

    # Create a resource named "server" as follows:
    # * The resource uses the "apache" resource agent
    # * The resource belongs to the resource group named "web"
    - name: Ensuring the server resource exists
      command:
        cmd: pcs resource create server apache --group=web --wait=60
      when: "'server' not in resources['stdout']"

    # Create a resource named "ip" as follows:
    # * The resource uses the "IPAddr2" resource agent
    # * The address to manage is 172.25.250.88/24
    # * The resource belongs to the resource group named "web"
    - name: Ensuring the ip resource exists
      command:
        cmd: >
          pcs resource create ip IPAddr2 ip=172.25.250.88 cidr_netmask=24
          --group=web --wait=60
      when: "'ip' not in resources['stdout']"
...

```

- 6.2. Run the cluster\_apache.yml playbook.

```
[student@workstation cr2]$ ansible-playbook cluster_apache.yml
```

7. By using the pcs command on one of the cluster nodes, create a resource group named termination. That resource group must include the following resources:
- A resource named proxy to manage the HAProxy systemd service.

**Chapter 13 |** Comprehensive Review

- A resource named `frontIP` to manage the `172.25.250.80/24` floating IP address.

71. Create the proxy resource that uses the `haproxy` systemd service as the resource agent.

```
[root@nodea ~]# pcs resource create proxy systemd:haproxy --group=termination  
[root@nodea ~]#
```

72. Create the `frontIP` resource that uses the `172.25.250.80/24` address.

```
[root@nodea ~]# pcs resource create frontIP IPAddr2 \  
> ip=172.25.250.80 cidr_netmask=24 --group=termination  
Assumed agent name 'ocf:heartbeat:IPAddr2' (deduced from 'IPAddr2')
```

8. Configure a constraint so that the `termination` and the `web` resource groups never run on the same node.

8.1. From the `nodea` machine, add the colocation constraint by using the `pcs` command.

```
[root@nodea ~]# pcs constraint colocation add termination with web -INFINITY  
[root@nodea ~]#
```

8.2. Confirm that the resource groups are not running on the same node.

```
[root@nodea ~]# pcs status  
...output omitted...  
Full List of Resources:  
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com  
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com  
* Resource Group: web:  
  * server (ocf::heartbeat:apache): Started nodea.private.example.com  
  * ip (ocf::heartbeat:IPAddr2): Started nodea.private.example.com  
* Resource Group: termination:  
  * proxy (systemd:haproxy): Started nodeb.private.example.com  
  * frontIP (ocf::heartbeat:IPAddr2): Started nodeb.private.example.com  
...output omitted...
```

9. On your `workstation` machine, as `student` user, use the `curl` command to navigate to `www.lab.example.com`.

9.1. From the `workstation` machine, use the `curl` command to test your configuration.

```
[student@workstation ~]$ curl https://www.lab.example.com  
Hello, world!
```

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade compreview-twinode
```

## Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-twinode
```

This concludes the section.

## ▶ Lab

# Deploying a GFS2 File System

In this review, you will deploy a GFS2 file system on an existing cluster.

## Outcomes

You should be able to:

- Configure an iSCSI initiator.
- Configure multipath on all the nodes.
- Create an LVM shared volume group and shared logical volume.
- Create a cluster resource group to activate the volume group and its logical volume on all the nodes.
- Create a GFS2 file system to mount the file system on the three nodes.

## Before You Begin

If you did not reset your `workstation`, `storage`, and `nodeX` machines at the end of the last chapter, save any work that you want to keep from earlier exercises on those machines, and reset them now.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start compreview-gfs2
```

This command ensures that the systems are up and reachable, and configures an iSCSI target on the storage machine named `iqn.2021-04.com.example:store-prod`. It also deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines with fencing resources for those machines.

## Instructions

Perform the following tasks to complete this comprehensive review exercise. On all your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Configure an iSCSI initiator on the `nodea`, `nodeb`, and `nodec` machines. For the initiator IQN, choose `iqn.2021-04.com.example:<short_hostname>`, for example `iqn.2021-04.com.example:nodea` on the `nodea` machine. On the three cluster nodes, access the iSCSI target on the storage machine through the two storage networks. You can reach the storage machine on those networks at `192.168.1.15` and `192.168.2.15`.
2. On the three cluster nodes, configure multipath so that you can access the shared storage through the `/dev/mapper/diska` device node.
3. Prepare your three nodes for LVM shared volume groups. To do so, install the required packages and then create the `dlm` and `lvmlockd` resources in the cloned `locking` resource group.

**Chapter 13 |** Comprehensive Review

4. Create an LVM shared volume group named `vg1` that uses the `/dev/mapper/diska` volume. Create a 4 GiB LVM shared logical volume named `lv1` in the `vg1` VG.
5. Create the `sharedlv` resource in a `gfsres` resource group to activate the volume group and its logical volume on all the nodes. Clone the `gfsres` resource group to start the resources on all cluster nodes. Ensure that the `gfsres` resource group starts after the `locking` resource group.
6. Create a GFS2 file system on the logical volume, set 3 journals with a size of 128 MiB, and set the lock space name to `gfsdata`.
7. Create a cluster resource named `clusterfs` to mount the GFS2 file system under `/srv/data` on all the cluster nodes. Add the resource to the `gfsres` resource group.

## Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade compreview-gfs2
```

## Finish

As the student user on the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-gfs2
```

This concludes the section.

## ► Solution

# Deploying a GFS2 File System

In this review, you will deploy a GFS2 file system on an existing cluster.

## Outcomes

You should be able to:

- Configure an iSCSI initiator.
- Configure multipath on all the nodes.
- Create an LVM shared volume group and shared logical volume.
- Create a cluster resource group to activate the volume group and its logical volume on all the nodes.
- Create a GFS2 file system to mount the file system on the three nodes.

## Before You Begin

If you did not reset your `workstation`, `storage`, and `nodeX` machines at the end of the last chapter, save any work that you want to keep from earlier exercises on those machines, and reset them now.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start compreview-gfs2
```

This command ensures that the systems are up and reachable, and configures an iSCSI target on the storage machine named `iqn.2021-04.com.example:store-prod`. It also deploys a three-node cluster on the `nodea`, `nodeb`, and `nodec` machines with fencing resources for those machines.

## Instructions

Perform the following tasks to complete this comprehensive review exercise. On all your machines, the password for the `student` user is `student`, and the `root` password is `redhat`.

1. Configure an iSCSI initiator on the `nodea`, `nodeb`, and `nodec` machines. For the initiator IQN, choose `iqn.2021-04.com.example:<short_hostname>`, for example `iqn.2021-04.com.example:nodea` on the `nodea` machine. On the three cluster nodes, access the iSCSI target on the storage machine through the two storage networks. You can reach the storage machine on those networks at `192.168.1.15` and `192.168.2.15`.
  - 1.1. In separate terminals, connect to the nodes and become the `root` user.

```
[student@workstation ~]$ ssh nodea  
...output omitted...  
[student@nodea ~]$ sudo -i  
[sudo] password for student: student  
[root@nodea ~]#
```

```
[student@workstation ~]$ ssh nodeb  
...output omitted...  
[student@nodeb ~]$ sudo -i  
[sudo] password for student: student  
[root@nodeb ~]#
```

```
[student@workstation ~]$ ssh nodec  
...output omitted...  
[student@nodec ~]$ sudo -i  
[sudo] password for student: student  
[root@nodec ~]#
```

- 1.2. Install the `iscsi-initiator-utils` package on every cluster node.

```
[root@nodea ~]# yum install iscsi-initiator-utils  
...output omitted...  
Complete!
```

Repeat this process on all cluster nodes.

- 1.3. On every cluster node, edit the `/etc/iscsi/initiatorname.iscsi` file and set the IQN for the client initiator. Use `iqn.2021-04.com.example:<short_hostname>`, for example `iqn.2021-04.com.example:nodea` on the nodea machine.

```
InitiatorName=iqn.2021-04.com.example:nodea
```

```
InitiatorName=iqn.2021-04.com.example:nodeb
```

```
InitiatorName=iqn.2021-04.com.example:nodec
```

- 1.4. Restart the `iscsid` service to take the new IQN into account on every cluster node.

```
[root@nodea ~]# systemctl restart iscsid
```

Repeat this process on all cluster nodes.

- 1.5. Discover and then log in to the iSCSI target through the first network on every cluster node. The portal address on that network is `192.168.1.15`.

```
[root@nodea ~]# iscsiadadm -m discovery -t st -p 192.168.1.15  
192.168.1.15:3260,1 iqn.2021-04.com.example:store-prod  
[root@nodea ~]# iscsiadadm -m node -T iqn.2021-04.com.example:store-prod \  
> -p 192.168.1.15 -l  
Logging in to [iface: default, target: iqn.2021-04.com.example:store-prod, portal:  
192.168.1.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store-prod, portal:  
192.168.1.15,3260] successful.
```

Repeat this process on all cluster nodes.

- 1.6. Now discover and then log in to the iSCSI target through the second network on every cluster node. The portal address on that network is 192.168.2.15.

```
[root@nodea ~]# iscsiadadm -m discovery -t st -p 192.168.2.15  
192.168.2.15:3260,1 iqn.2021-04.com.example:store-prod  
[root@nodea ~]# iscsiadadm -m node -T iqn.2021-04.com.example:store-prod \  
> -p 192.168.2.15 -l  
Logging in to [iface: default, target: iqn.2021-04.com.example:store-prod, portal:  
192.168.2.15,3260]  
Login to [iface: default, target: iqn.2021-04.com.example:store-prod, portal:  
192.168.2.15,3260] successful.
```

Repeat this process on all cluster nodes.

2. On the three cluster nodes, configure multipath so that you can access the shared storage through the /dev/mapper/diska device node.

- 2.1. Install the `device-mapper-multipath` package on all the machines.

```
[root@nodea ~]# yum install device-mapper-multipath  
...output omitted...  
Complete!
```

Repeat this process on all cluster nodes.

- 2.2. On the nodea machine, use the `mpathconf` command to create the multipath configuration file. You create and configure that file on the nodea machine and then copy it to the nodeb and the nodec machines in a following step.

```
[root@nodea ~]# mpathconf --enable
```

- 2.3. On the nodea machine, identify one of the newly available block devices.

```
[root@nodea ~]# iscsiadadm -m session -P 3  
...output omitted...  
Attached scsi disk sdb           State: running
```

- 2.4. Retrieve the WWID of the iSCSI storage.

```
[root@nodea ~]# udevadm info /dev/sdb | grep ID_SERIAL=  
E: ID_SERIAL=3600140562aeac25dc4c4eb5842574c7a
```

**Chapter 13 |** Comprehensive Review

On your system, the WWID is probably different.

- 2.5. Edit the `/etc/multipath.conf` configuration file and then add the `multipaths` section at the end. In the following example, replace the WWID by the one that you retrieved in the preceding step.

```
multipaths {
    multipath {
        wwid           3600140562aeac25dc4c4eb5842574c7a
        alias          diskA
    }
}
```

- 2.6. Copy the `/etc/multipath.conf` configuration file from the nodea machine to the nodeb and nodec machines. For your convenience, you can run the `playbook.yml` Ansible playbook on the `/home/student/labs/cr3/` directory. As the student user on the workstation machine, change to the `/home/student/labs/cr3/` project directory and run the playbook.

```
[student@workstation ~]$ cd /home/student/labs/cr3/
[student@workstation cr3]$
```

```
[student@workstation cr3]$ ansible-playbook playbook.yml
...output omitted...
```

- 2.7. Enable and start the `multipathd` service on every cluster node.

```
[root@nodea ~]# systemctl enable --now multipathd
```

Repeat this process on all cluster nodes.

- 2.8. Use the `multipath` command to verify the configuration on every cluster node.

```
[root@nodea ~]# multipath -ll
diskA (3600140562aeac25dc4c4eb5842574c7a) dm-0 LIO-0RG,storage.disk1
size=10G features='0' hwhandler='1' alua' wp=rw
|-- policy='service-time 0' prio=50 status=active
| `-- 2:0:0:0 sda 8:0 active ready running
`-- policy='service-time 0' prio=50 status=enabled
   `-- 3:0:0:0 sdb 8:16 active ready running
```

Confirm the same output on all other cluster nodes.

- 2.9. Ensure that the new device node is available under the `/dev/mapper/` directory.

```
[root@nodea ~]# ls /dev/mapper/diska
/dev/mapper/diska
```

Confirm the same output on all other cluster nodes.

3. Prepare your three nodes for LVM shared volume groups. To do so, install the required packages and then create the `dlm` and `lvmlockd` resources in the cloned `locking` resource group.

- 3.1. Check that the `dlm` and `lvm2-lockd` packages are installed on all cluster nodes. These packages are installed by default in all the cluster nodes, but this step installs them in case they are not present.

```
[root@nodea ~]# yum install dlm lvm2-lockd  
...output omitted...  
Complete!
```

Repeat this process on all cluster nodes.

- 3.2. On `nodea` as `root`, use the `pcs resource create` command to create a resource named `dlm` of type `ocf:pacemaker:controld`. Add the resource to the `locking` resource group.

```
[root@nodea ~]# pcs resource create dlm ocf:pacemaker:controld \  
> op monitor interval=30s on-fail=fence --group=locking
```

- 3.3. Create a resource named `lvmlockd` of type `ocf:heartbeat:lvmlockd`. Add the resource to the `locking` resource group.

```
[root@nodea ~]# pcs resource create lvmlockd ocf:heartbeat:lvmlockd \  
> op monitor interval=30s on-fail=fence --group=locking  
[root@nodea ~]#
```

- 3.4. Use the `pcs resource clone` command to clone the `locking` resource group. Set the `interleave` clone option to `true`.

```
[root@nodea ~]# pcs resource clone locking interleave=true
```

- 3.5. Use the `pcs status --full` command to verify your work.

```
[root@nodea ~]# pcs status --full  
...output omitted...  
Full List of Resources:  
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com  
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com  
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com  
* Clone Set: locking-clone [locking]:  
  * Resource Group: locking:0:  
    * dlm (ocf::pacemaker:controld): Started nodec.private.example.com  
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodec.private.example.com  
  * Resource Group: locking:1:  
    * dlm (ocf::pacemaker:controld): Started nodea.private.example.com  
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodea.private.example.com  
  * Resource Group: locking:2:  
    * dlm (ocf::pacemaker:controld): Started nodeb.private.example.com  
    * lvmlockd (ocf::heartbeat:lvmlockd): Started nodeb.private.example.com  
...output omitted...
```

Notice that the cluster starts the resources on all three nodes.

4. Create an LVM shared volume group named `vg1` that uses the `/dev/mapper/diska` volume. Create a 4 GiB LVM shared logical volume named `lv1` in the `vg1` VG.

**Chapter 13 |** Comprehensive Review

- 4.1. On the nodea machine, create an LVM physical volume on the /dev/mapper/diska multipath device, which provides redundant access to the shared iSCSI storage.

```
[root@nodea ~]# pvcreate /dev/mapper/diska
Physical volume "/dev/mapper/diska" successfully created.
```

- 4.2. On the nodea machine, create a shared LVM volume group, named vg1, that uses the LVM physical volume from the preceding step.

```
[root@nodea ~]# vgcreate --shared vg1 /dev/mapper/diska
Volume group "vg1" successfully created
VG vg1 starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

- 4.3. Register the volume group with the LVM lock space on the other cluster nodes.

```
[root@nodeb ~]# vgchange --lock-start vg1
VG vg1 starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

```
[root@nodec ~]# vgchange --lock-start vg1
VG vg1 starting dlm lockspace
Starting locking. Waiting until locks are ready...
```

- 4.4. Create a 4 GiB LVM logical volume, named lv1, in the vg1 volume group and activate the logical volume with a shared lock so that all three nodes can use it concurrently.

```
[root@nodea ~]# lvcreate --activate sy -L4G -n lv1 vg1
Logical volume "lv1" created.
```

5. Create the `sharedlv` resource in a `gfsres` resource group to activate the volume group and its logical volume on all the nodes. Clone the `gfsres` resource group to start the resources on all cluster nodes. Ensure that the `gfsres` resource group starts after the `locking` resource group.

- 5.1. Use the `pcs resource create` command to create the `sharedlv` resource of type `LVM-activate`. Add the resource to the `gfsres` resource group.

```
[root@nodea ~]# pcs resource create sharedlv LVM-activate vgname=vg1 \
> lvname=lv1 activation_mode=shared vg_access_mode=lvmlockd \
> --group=gfsres
Assumed agent name 'ocf:heartbeat:LVM-activate' (deduced from 'LVM-activate')
```

- 5.2. Clone the `gfsres` resource group so that the cluster starts its resources on all nodes. Set the `interleave` clone option to `true`.

```
[root@nodea ~]# pcs resource clone gfsres interleave=true
```

- 5.3. Use the `pcs status --full` command to verify your work.

```
[root@nodea ~]# pcs status --full
...output omitted...
* Clone Set: gfsres-clone [gfsres]:
  * Resource Group: gfsres:0:
    * sharedlv (ocf::heartbeat:LVM-activate): Started nodec.private.example.com
  * Resource Group: gfsres:1:
    * sharedlv (ocf::heartbeat:LVM-activate): Started nodea.private.example.com
  * Resource Group: gfsres:2:
    * sharedlv (ocf::heartbeat:LVM-activate): Started nodeb.private.example.com
...output omitted...
```

- 5.4. Use the `pcs constraint order` command to ensure that the `locking` resource group starts before the `gfsres` resource group.

```
[root@nodea ~]# pcs constraint order start locking-clone then gfsres-clone
Adding locking-clone gfsres-clone (kind: Mandatory) (Options: first-action=start
then-action=start)
```

- 5.5. Use the `pcs constraint colocation` command to colocate the two groups.

```
[root@nodea ~]# pcs constraint colocation add gfsres-clone with locking-clone
```

- 5.6. Confirm that the `lv1` logical volume is active on every cluster node.

```
[root@nodea ~]# lvs
  LV   VG Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  lv1  vg1 -wi-a---- 4.00g
```

Confirm the same output on all other cluster nodes.

6. Create a GFS2 file system on the logical volume, set 3 journals with a size of 128 MiB, and set the lock space name to `gfsdata`.

- 6.1. Check that the `gfs2-utils` package is installed on all the machines. This package is installed by default in all the cluster nodes, but this step installs it in case it is not present.

```
[root@nodea ~]# yum install gfs2-utils
...output omitted...
Complete!
```

Repeat this process on all cluster nodes.

- 6.2. Use the `pcs status` command to retrieve the name of the cluster. You need that information to create the GFS2 file system.

```
[root@nodea ~]# pcs status
Cluster name: cluster1
Cluster Summary:
  * Stack: corosync
  * Current DC: nodec.private.example.com (version 2.0.4-6.el8-2deceaa3ae) -
partition with quorum
```

**Chapter 13 |** Comprehensive Review

```
* Last updated: Fri Apr 23 20:06:35 2021
* Last change: Fri Apr 23 19:25:32 2021 by root via cibadmin on
nodea.private.example.com
* 3 nodes configured
* 12 resource instances configured

Node List:
* Online: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

Full List of Resources:
* fence_nodea (stonith:fence_ipmilan): Started nodea.private.example.com
* fence_nodeb (stonith:fence_ipmilan): Started nodeb.private.example.com
* fence_nodec (stonith:fence_ipmilan): Started nodec.private.example.com
* Clone Set: locking-clone [locking]:
  * Started: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]
* Clone Set: gfsres-clone [gfsres]:
  * Started: [ nodea.private.example.com nodeb.private.example.com
nodec.private.example.com ]

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

- 6.3. Format the /dev/vg1/lv1 device with a GFS2 file system. Set three journals with a size of 128 MiB, and set the lock space name to gfsdata.

```
[root@nodea ~]# mkfs.gfs2 -j 3 -J 128 -t cluster1:gfsdata /dev/vg1/lv1
/dev/vg1/lv1 is a symbolic link to /dev/dm-1
This will destroy any data on /dev/dm-1
Are you sure you want to proceed? [y/n] y
Discarding device contents (may take a while on large devices): Done
Adding journals: Done
Building resource groups: Done
Creating quota file: Done
Writing superblock and syncing: Done
Device:          /dev/vg1/lv1
Block size:      4096
Device size:     4.00 GB (1048576 blocks)
Filesystem size: 4.00 GB (1048575 blocks)
Journals:        3
Journal size:    128MB
Resource groups: 18
Locking protocol: "lock_dlm"
Lock table:      "cluster1:gfsdata"
UUID:            1280455d-d93e-4179-914d-f4d6fe16331d
```

- 6.4. Set the no-quorum-policy cluster option to freeze.

```
[root@nodea ~]# pcs property set no-quorum-policy=freeze
```

**Chapter 13 |** Comprehensive Review

7. Create a cluster resource named `clusterfs` to mount the GFS2 file system under `/srv/data` on all the cluster nodes. Add the resource to the `gfsres` resource group.
  - 7.1. Create a file system resource named `clusterfs` that mounts the GFS2 file system under the `/srv/data` mount point. Add that resource to the `gfsres` resource group, which also activates the shared volume group and logical volume.

```
[root@nodea ~]# pcs resource create clusterfs Filesystem \
> device=/dev/vg1/lv1 directory=/srv/data fstype=gfs2 \
> op monitor interval=10s on-fail=fence --group=gfsres
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
```

- 7.2. Use the `pcs status --full` command to confirm that the cluster starts the `clusterfs` resource on the three nodes.

```
[root@nodea ~]# pcs status --full
...output omitted...
* Clone Set: gfsres-clone [gfsres]:
  * Resource Group: gfsres:0:
    * sharedlv (ocf::heartbeat:LVM-activate): Started nodec.private.example.com
    * clusterfs (ocf::heartbeat:Filesystem): Started nodec.private.example.com
  * Resource Group: gfsres:1:
    * sharedlv (ocf::heartbeat:LVM-activate): Started nodea.private.example.com
    * clusterfs (ocf::heartbeat:Filesystem): Started nodea.private.example.com
  * Resource Group: gfsres:2:
    * sharedlv (ocf::heartbeat:LVM-activate): Started nodeb.private.example.com
    * clusterfs (ocf::heartbeat:Filesystem): Started nodeb.private.example.com
...output omitted...
```

- 7.3. Confirm that the cluster mounts the GFS2 file system on the three nodes.

```
[root@nodea ~]# mount | grep gfs2
/dev/mapper/vg1-lv1 on /srv/data type gfs2 (rw,relatime,seclabel)
```

```
[root@nodeb ~]# mount | grep gfs2
/dev/mapper/vg1-lv1 on /srv/data type gfs2 (rw,relatime,seclabel)
```

```
[root@nodec ~]# mount | grep gfs2
/dev/mapper/vg1-lv1 on /srv/data type gfs2 (rw,relatime,seclabel)
```

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade compreview-gfs2
```

## Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-gfs2
```

This concludes the section.