Clusters from Scratch

Release 2.1.9

the Pacemaker project contributors

Nov 04, 2024

CONTENTS

Step-by-Step Instructions for Building Your First High-Availability Cluster

CHAPTER

ONE

ABSTRACT

This document provides a step-by-step guide to building a simple high-availability cluster using Pacemaker.

The example cluster will use:

- AlmaLinux 9 as the host operating system
- Corosync to provide messaging and membership services
- Pacemaker 2 as the cluster resource manager
- DRBD as a cost-effective alternative to shared storage
- GFS2 as the cluster filesystem (in active/active mode)

Given the graphical nature of the install process, a number of screenshots are included. However, the guide is primarily composed of commands, the reasons for executing them, and their expected outputs.

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Introduction

2.1.1 The Scope of This Document

Computer clusters can be used to provide highly available services or resources. The redundancy of multiple machines is used to guard against failures of many types.

This document will walk through the installation and setup of simple clusters using the AlmaLinux distribution, version 9.

The clusters described here will use Pacemaker and Corosync to provide resource management and messaging. Required packages and modifications to their configuration files are described along with the use of the `pcs` command line tool for generating the XML used for cluster control.

Pacemaker is a central component and provides the resource management required in these systems. This management includes detecting and recovering from the failure of various nodes, resources, and services under its control.

When more in-depth information is required, and for real-world usage, please refer to the Pacemaker Explained manual.

2.1.2 What Is Pacemaker?

Pacemaker is a high-availability *cluster resource manager* – software that runs on a set of hosts (a *cluster* of *nodes*) in order to preserve integrity and minimize downtime of desired services (*resources*).[1] It is maintained by the ClusterLabs community.

Pacemaker's key features include:

- Detection of and recovery from node- and service-level failures
- Ability to ensure data integrity by fencing faulty nodes
- Support for one or more nodes per cluster
- Support for multiple resource interface standards (anything that can be scripted can be clustered)
- Support (but no requirement) for shared storage
- Support for practically any redundancy configuration (active/passive, N+1, etc.)
- Automatically replicated configuration that can be updated from any node

[1] *Cluster* is sometimes used in other contexts to refer to hosts grouped together for other purposes, such as high-performance computing (HPC), but Pacemaker is not intended for those purposes.

- Ability to specify cluster-wide relationships between services, such as ordering, colocation, and anti-colocation
- Support for advanced service types, such as *clones* (services that need to be active on multiple nodes), *promotable clones* (clones that can run in one of two roles), and containerized services
- Unified, scriptable cluster management tools

Note: Fencing

Fencing, also known as *STONITH* (an acronym for Shoot The Other Node In The Head), is the ability to ensure that it is not possible for a node to be running a service. This is accomplished via *fence devices* such as intelligent power switches that cut power to the target, or intelligent network switches that cut the target's access to the local network.

Pacemaker represents fence devices as a special class of resource.

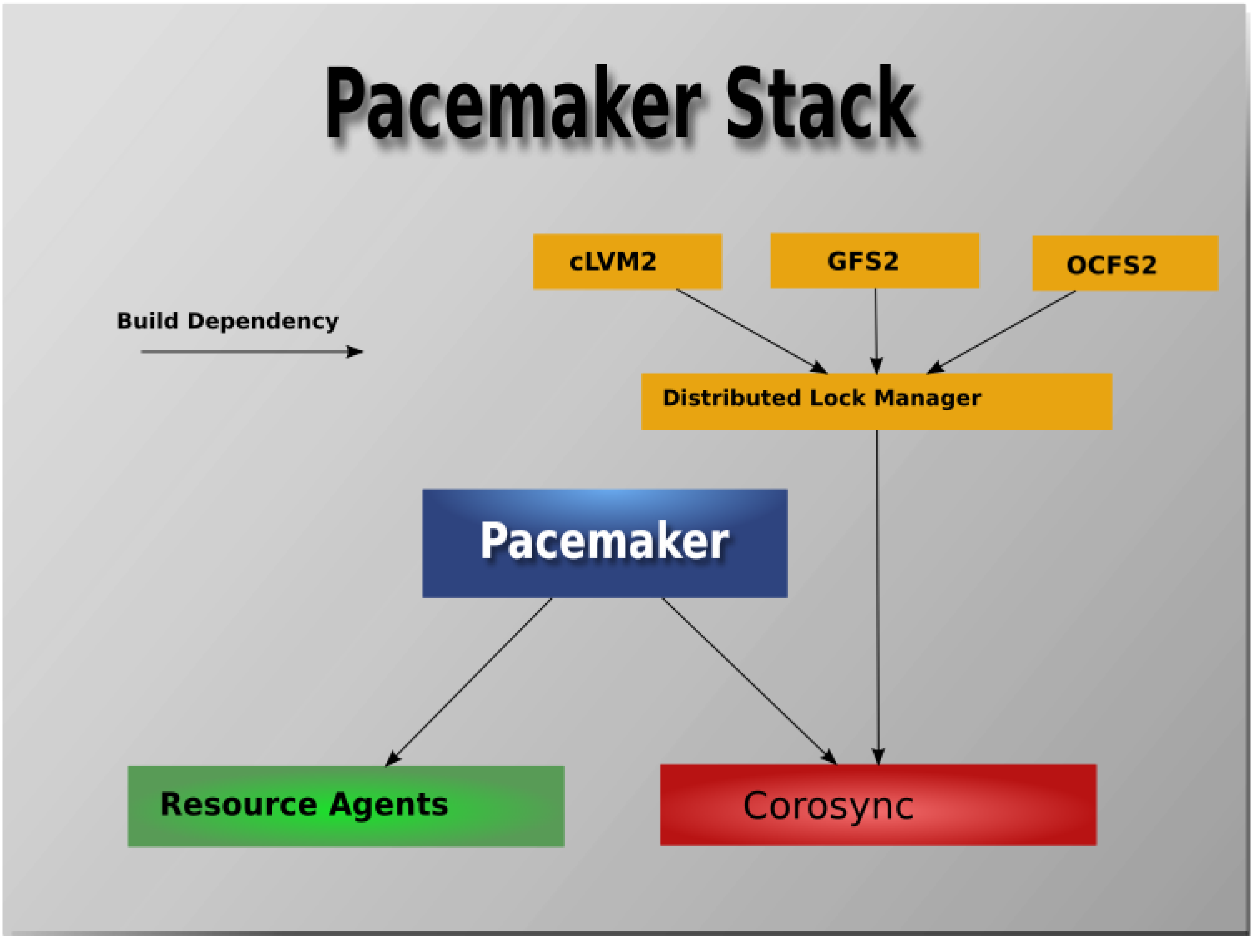
A cluster cannot safely recover from certain failure conditions, such as an unresponsive node, without fencing.

Cluster Architecture

At a high level, a cluster can be viewed as having these parts (which together are often referred to as the *cluster stack*):

- **Resources:** These are the reason for the cluster's being – the services that need to be kept highly available.
- **Resource agents:** These are scripts or operating system components that start, stop, and monitor resources, given a set of resource parameters. These provide a uniform interface between Pacemaker and the managed services.
- **Fence agents:** These are scripts that execute node fencing actions, given a target and fence device parameters.
- **Cluster membership layer:** This component provides reliable messaging, membership, and quorum information about the cluster. Currently, Pacemaker supports Corosync as this layer.
- **Cluster resource manager:** Pacemaker provides the brain that processes and reacts to events that occur in the cluster. These events may include nodes joining or leaving the cluster; resource events caused by failures, maintenance, or scheduled activities; and other administrative actions. To achieve the desired availability, Pacemaker may start and stop resources and fence nodes.
- **Cluster tools:** These provide an interface for users to interact with the cluster. Various command-line and graphical (GUI) interfaces are available.

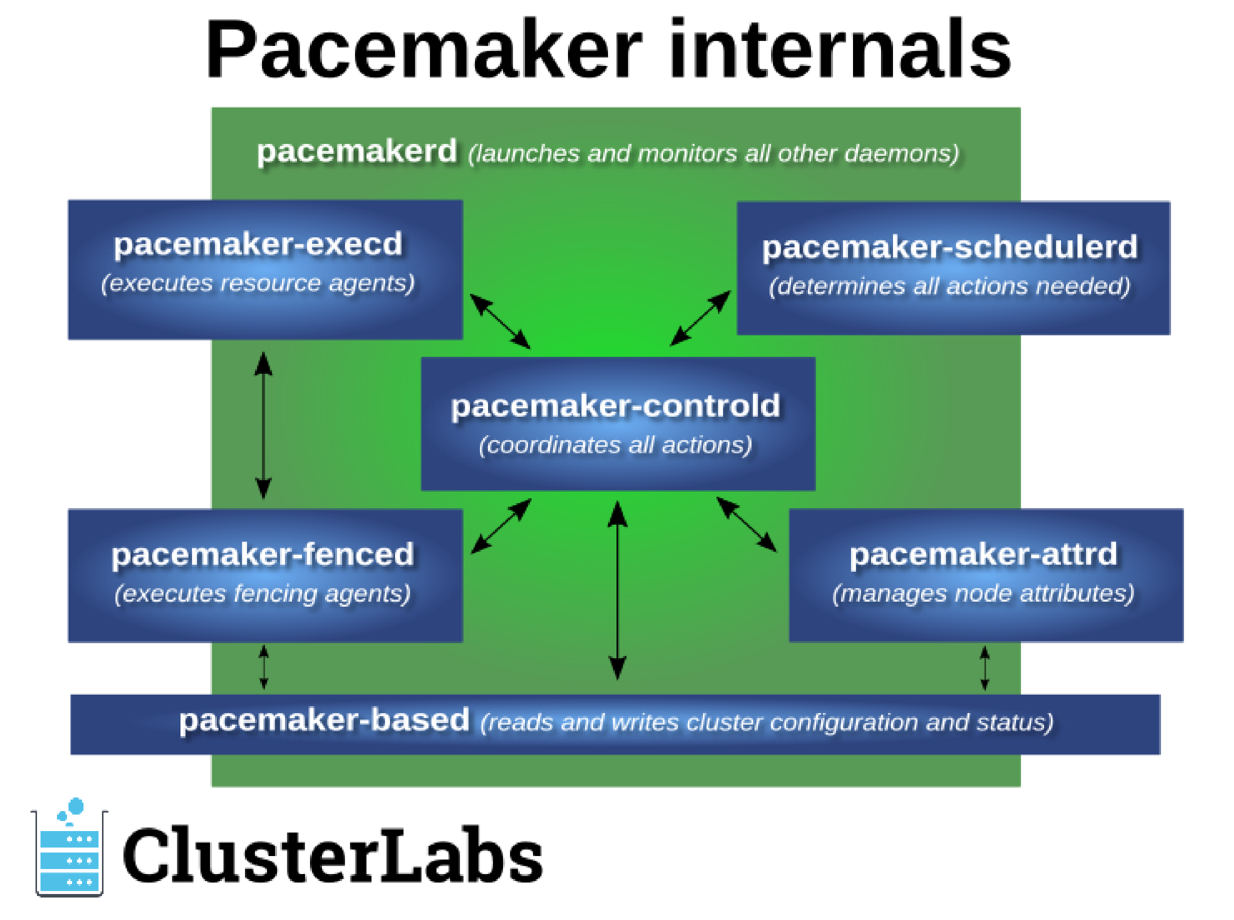
Most managed services are not, themselves, cluster-aware. However, many popular open-source cluster filesystems make use of a common *Distributed Lock Manager* (DLM), which makes direct use of Corosync for its messaging and membership capabilities and Pacemaker for the ability to fence nodes.



Pacemaker Architecture

Pacemaker itself is composed of multiple daemons that work together:

- `pacemakerd`
- `pacemaker-attrd`
- `pacemaker-based`
- `pacemaker-controld`
- `pacemaker-execd`
- `pacemaker-fenced`
- `pacemaker-schedulerd`



Pacemaker's main process (`pacemakerd`) spawns all the other daemons, and respawns them if they unexpectedly exit.

The *Cluster Information Base* (CIB) is an XML representation of the cluster's configuration and the state of all nodes and resources. The *CIB manager* (`pacemaker-based`) keeps the CIB synchronized across the cluster, and handles requests to modify it.

The *attribute manager* (`pacemaker-attrd`) maintains a database of attributes for all nodes, keeps it synchronized across the cluster, and handles requests to modify them. These attributes are usually recorded in the CIB.

Given a snapshot of the CIB as input, the *scheduler* (`pacemaker-schedulerd`) determines what actions are necessary to achieve the desired state of the cluster.

The *local executor* (`pacemaker-execd`) handles requests to execute resource agents on the local cluster node, and returns the result.

The *fencer* (`pacemaker-fenced`) handles requests to fence nodes. Given a target node, the fencer decides which cluster node(s) should execute which fencing device(s), and calls the necessary fencing agents (either directly, or via requests to the fencer peers on other nodes), and returns the result.

The *controller* (`pacemaker-controld`) is Pacemaker's coordinator, maintaining a consistent view of the cluster membership and orchestrating all the other components.

Pacemaker centralizes cluster decision-making by electing one of the controller instances as the *Designated Controller* (*DC*). Should the elected DC process (or the node it is on) fail, a new one is quickly established. The DC responds to cluster events by taking a current snapshot of the CIB, feeding it to the scheduler, then asking the executors (either directly on the local node, or via requests to controller peers on other nodes) and the fencer to execute any necessary actions.

Note: Old daemon names

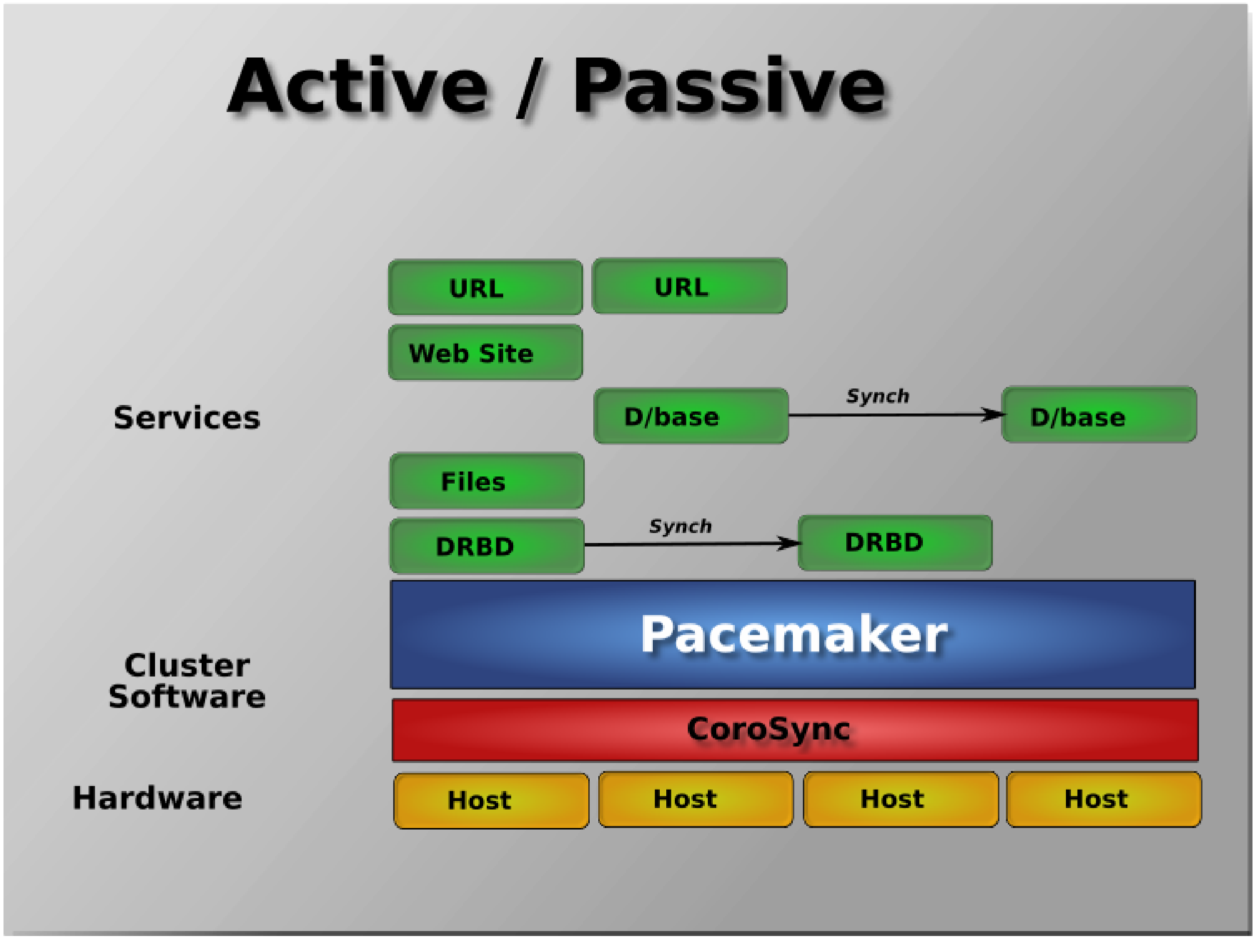
The Pacemaker daemons were renamed in version 2.0. You may still find references to the old names, especially in documentation targeted to version 1.1.

Old name	New name
`attrd`	`pacemaker-attrd`
`cib`	`pacemaker-based`
`crmd`	`pacemaker-controld`
`lrmd`	`pacemaker-execd`
`stonithd`	`pacemaker-fenced`
`pacemaker_remoted`	`pacemaker-remoted`

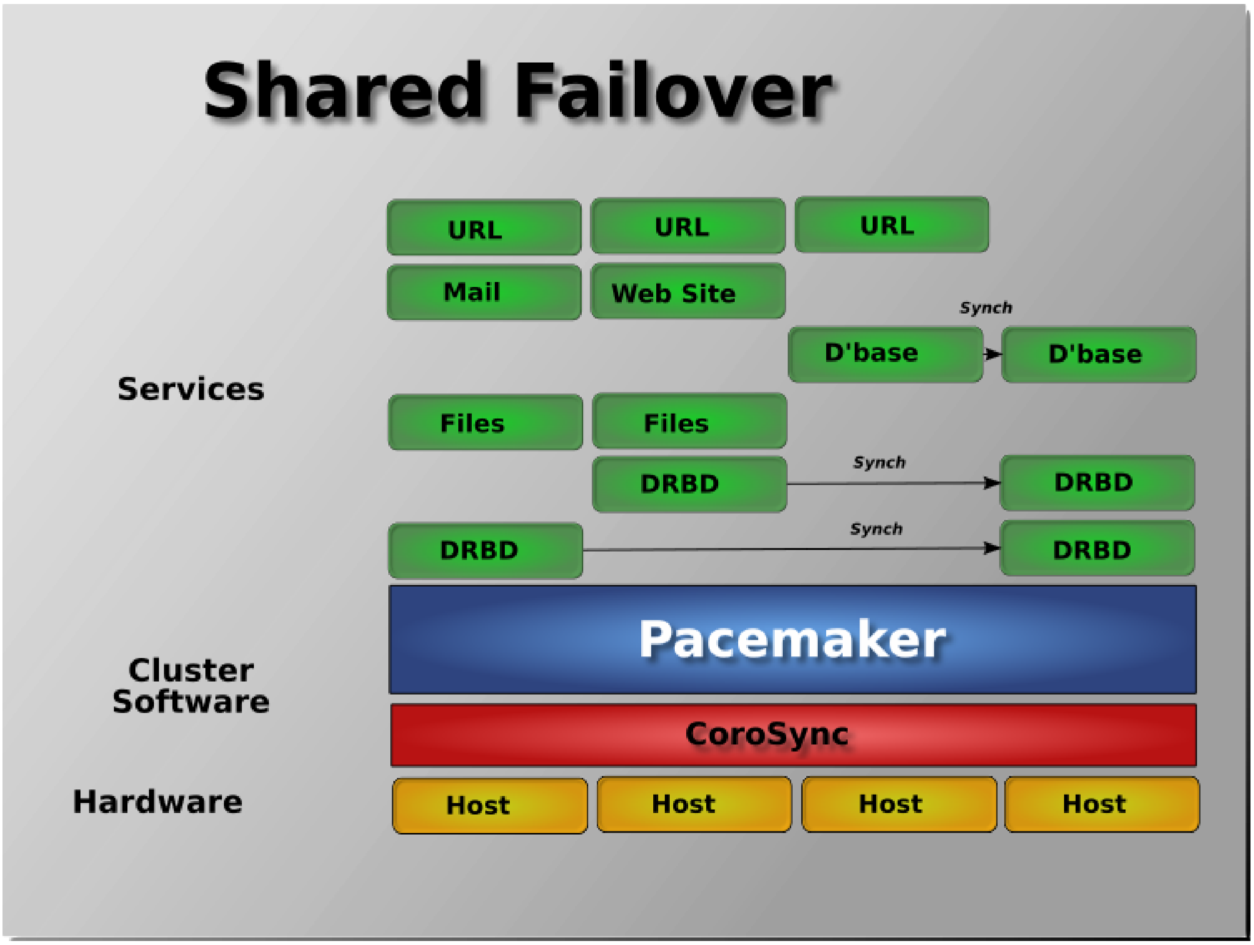
Node Redundancy Designs

Pacemaker supports practically any node redundancy configuration including *Active/Active*, *Active/Passive*, *N+1*, *N+M*, *N-to-1*, and *N-to-N*.

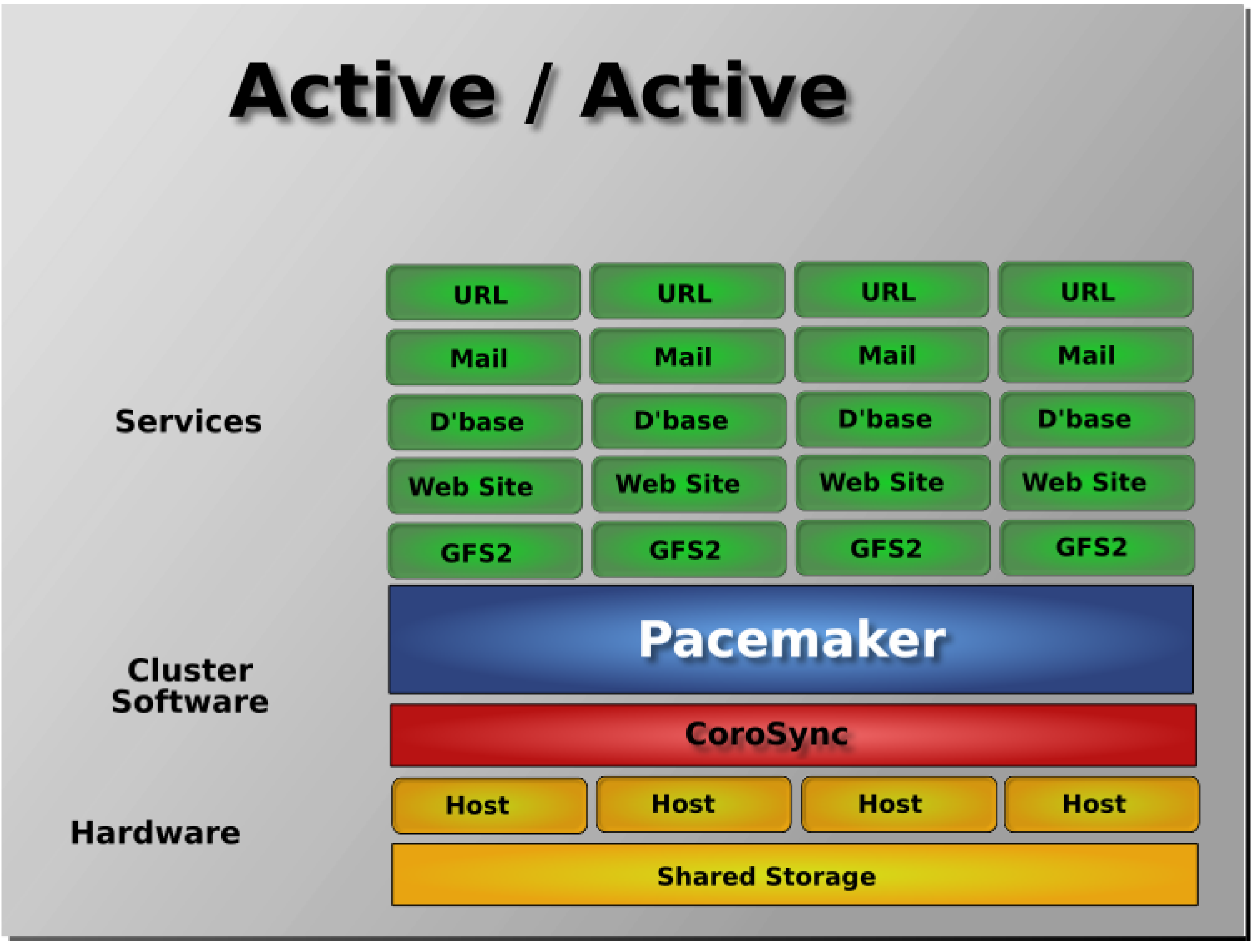
Active/passive clusters with two (or more) nodes using Pacemaker and DRBD are a cost-effective high-availability solution for many situations. One of the nodes provides the desired services, and if it fails, the other node takes over.



Pacemaker also supports multiple nodes in a shared-failover design, reducing hardware costs by allowing several active/passive clusters to be combined and share a common backup node.



When shared storage is available, every node can potentially be used for failover. Pacemaker can even run multiple copies of services to spread out the workload. This is sometimes called N-to-N redundancy.



2.2 Installation

2.2.1 Install AlmaLinux 9

Boot the Install Image

Download the latest AlmaLinux 9 DVD ISO by navigating to the AlmaLinux mirrors list, selecting the latest 9.x version for your machine's architecture, selecting a download mirror that's close to you, and finally selecting the latest .iso file that has "dvd" in its name. Use the image to boot a virtual machine, or burn it to a DVD or USB drive and boot a physical server from that.

After starting the installation, select your language and keyboard layout at the welcome screen.

Installation Options

At this point, you get a chance to tweak the default installation options.

Click on the **SOFTWARE SELECTION** section (try saying that 10 times quickly). The default environment, **Server with GUI**, does have add-ons with much of the software we need, but we will change the environment to a **Minimal Install** here, so that we can see exactly what software is required later, and press **Done**.

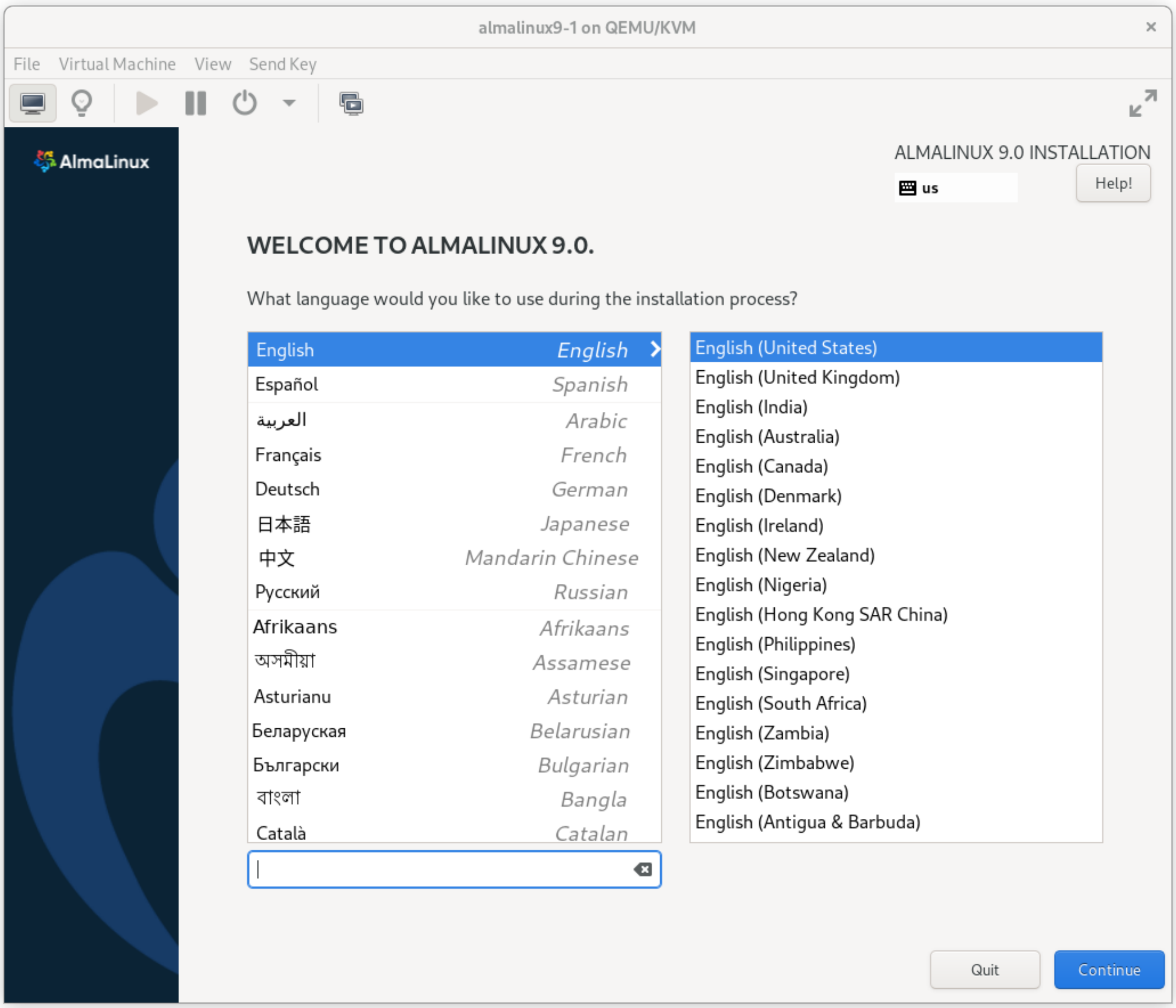


Fig. 1: AlmaLinux 9 Installation Welcome Screen

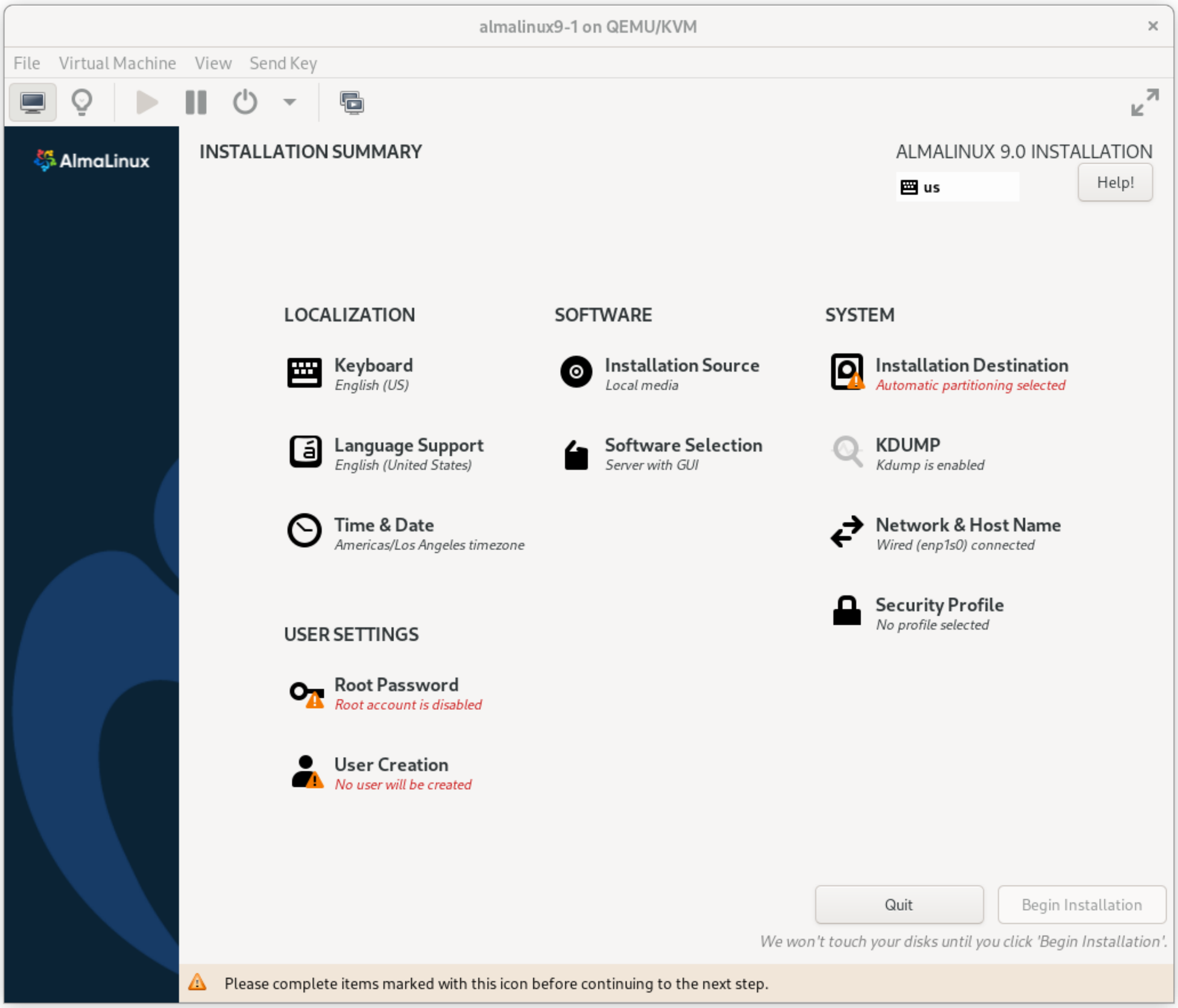


Fig. 2: AlmaLinux 9 Installation Summary Screen

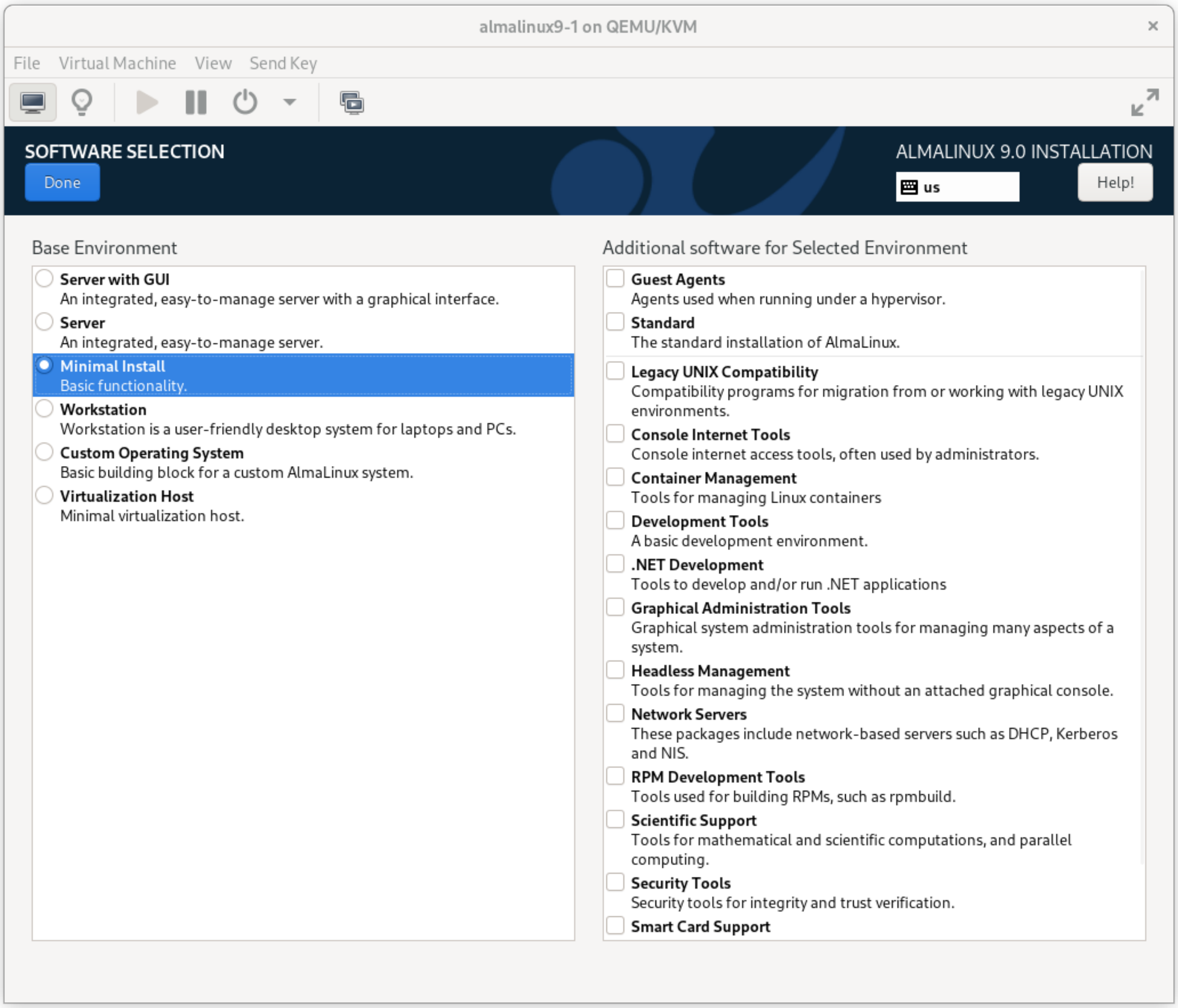


Fig. 3: AlmaLinux 9 Software Selection Screen

Configure Network

In the **NETWORK & HOST NAME** section:

- Edit **Host Name:** as desired. For this example, we will enter `pcmk-1.localdomain` and then press **Apply**.
- Select your network device, press **Configure...**, select the **IPv4 Settings** tab, and select **Manual** from the **Method** dropdown menu. Then assign the machine a fixed IP address with an appropriate netmask, gateway, and DNS server. For this example, we'll use `192.168.122.101` for the address, `24` for the netmask, and `192.168.122.1` for the gateway and DNS server.
- Press **Save**.
- Flip the switch to turn your network device on (if it is not on already), and press **Done**.

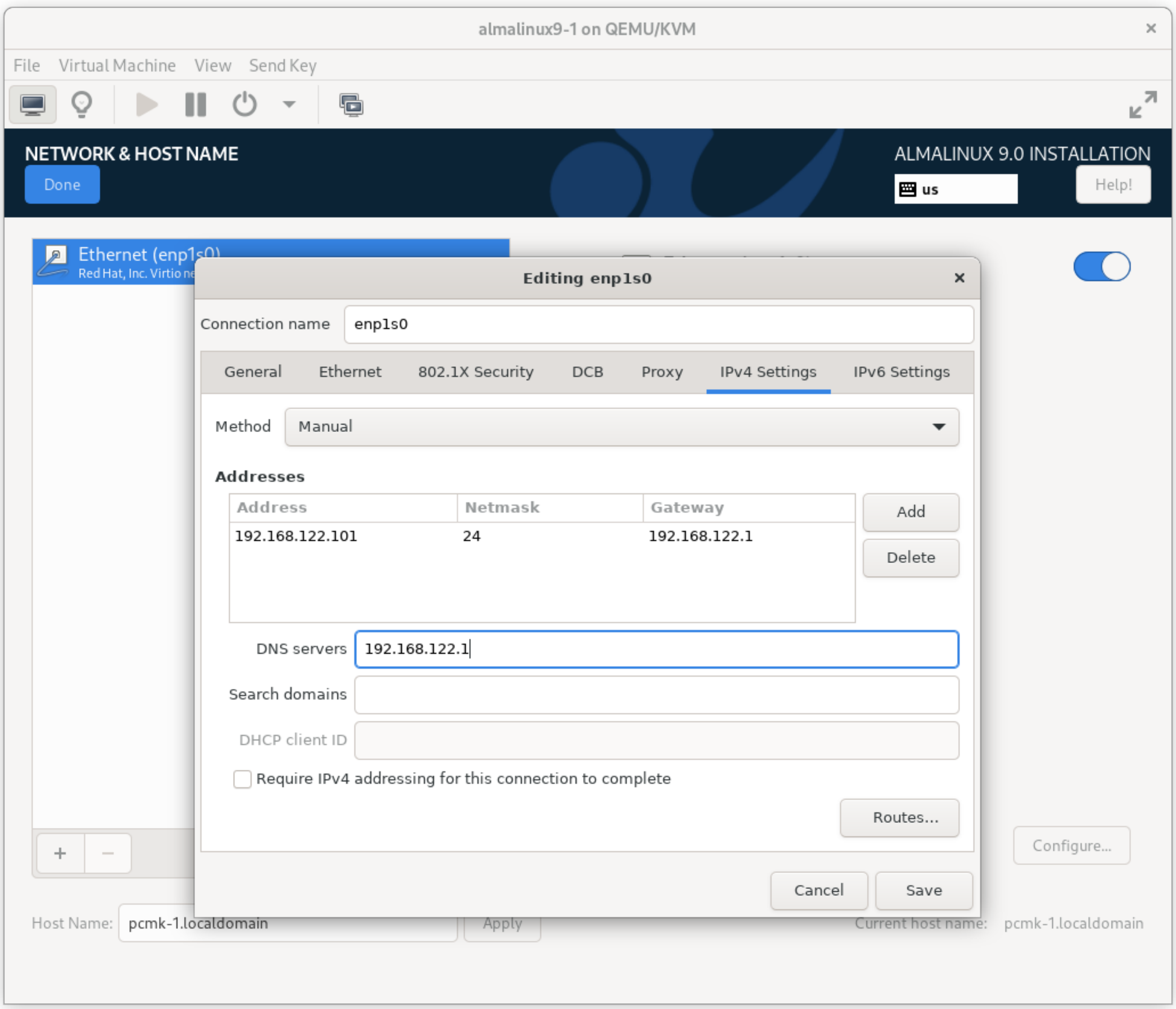


Fig. 4: AlmaLinux 9 Network Interface Screen

Important: Do not accept the default network settings. Cluster machines should never obtain an IP address via DHCP, because DHCP's periodic address renewal will interfere with Corosync.

Configure Disk

By default, the installer's automatic partitioning will use LVM (which allows us to dynamically change the amount of space allocated to a given partition). However, it allocates all free space to the / (a.k.a. **root**) partition, which cannot be reduced in size later (dynamic increases are fine).

In order to follow the DRBD and GFS2 portions of this guide, we need to reserve space on each machine for a replicated volume.

Enter the **INSTALLATION DESTINATION** section and select the disk where you want to install the OS. Then under **Storage Configuration**, select **Custom** and press **Done**.

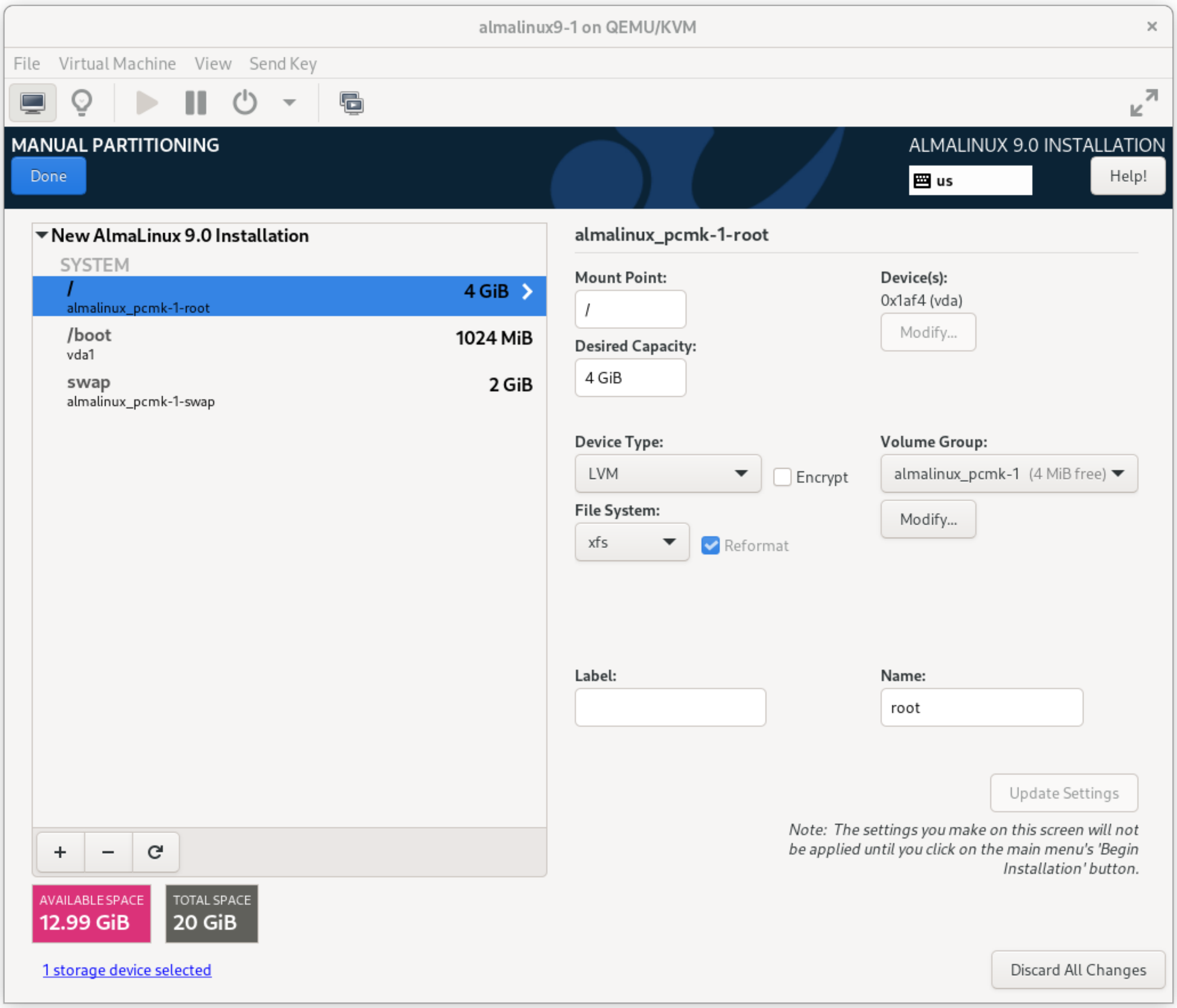


Fig. 5: AlmaLinux 9 Installation Destination Screen

On the **MANUAL PARTITIONING** screen that comes next, click the option to create mountpoints automatically. Select the / mountpoint and reduce the **Desired Capacity** down to 4 GiB or so. (The installer will not allow you to proceed if the / filesystem is too small to install all required packages.)

Then select **Modify…** next to the volume group name. In the **CONFIGURE VOLUME GROUP** dialog box that appears, change the **Size policy** to **As large as possible**, to make the reclaimed space available inside the LVM volume group. We'll add the additional volume later.

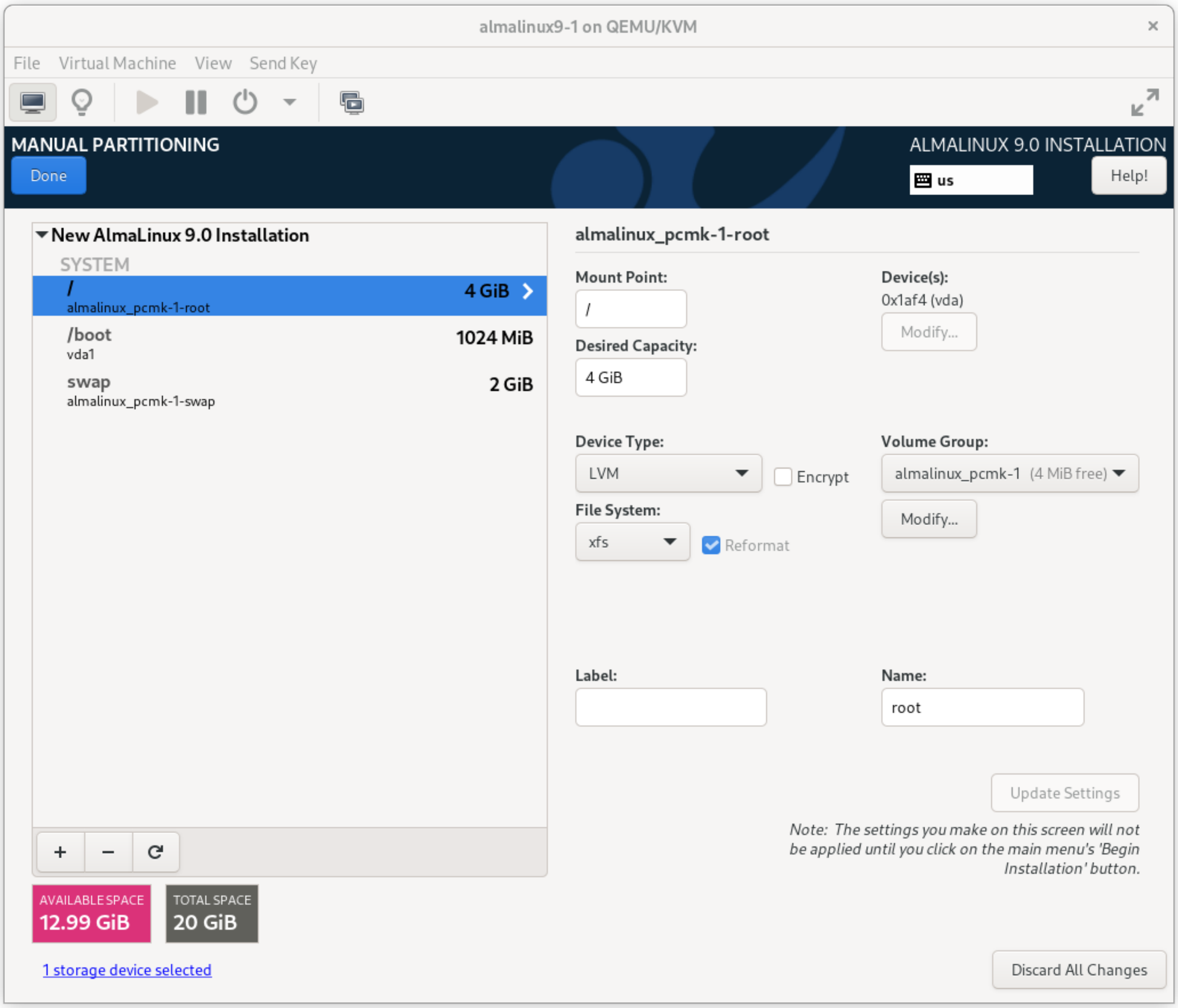


Fig. 6: AlmaLinux 9 Manual Partitioning Screen

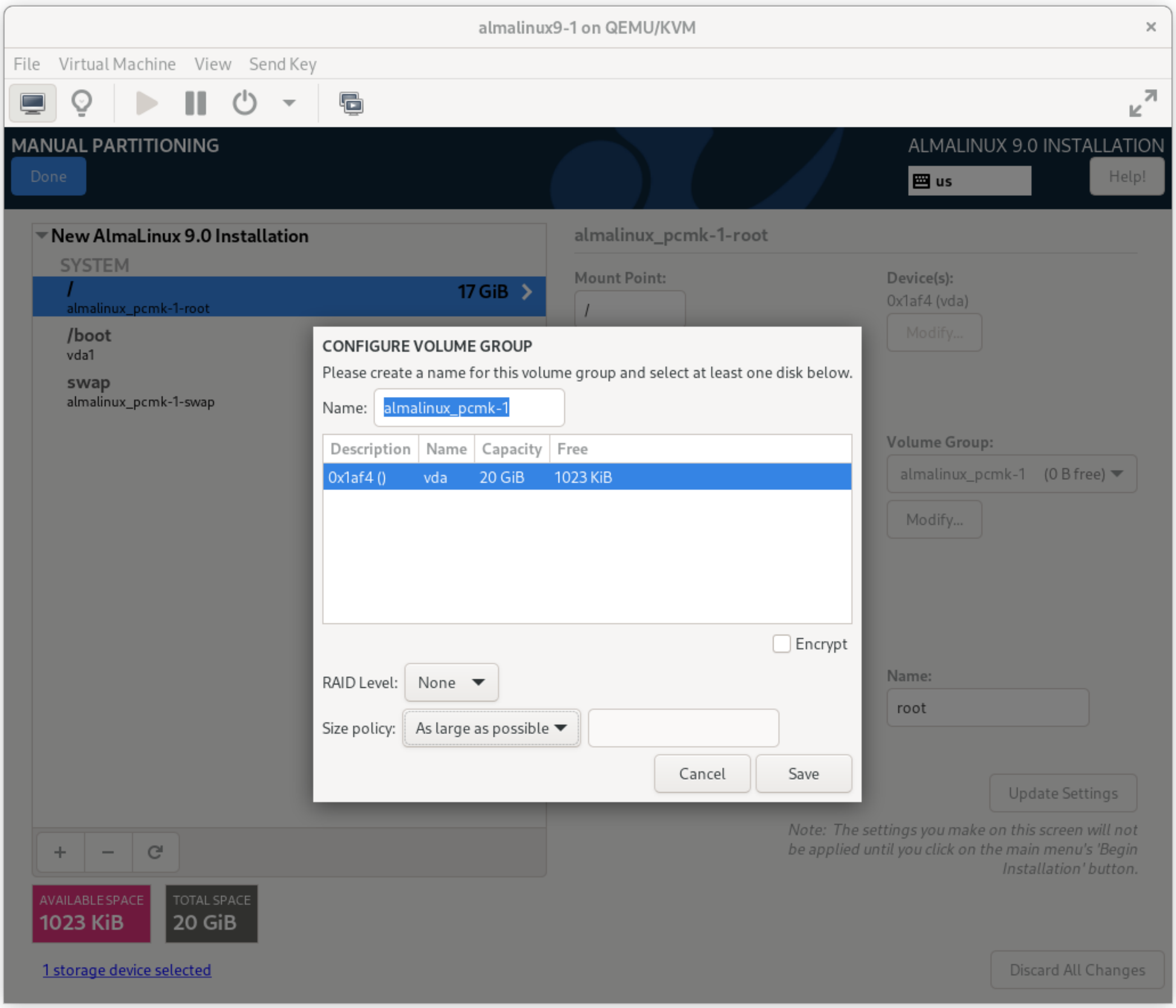


Fig. 7: AlmaLinux 9 Configure Volume Group Dialog

Press **Done**. Finally, in the **SUMMARY OF CHANGES** dialog box, press **Accept Changes**.

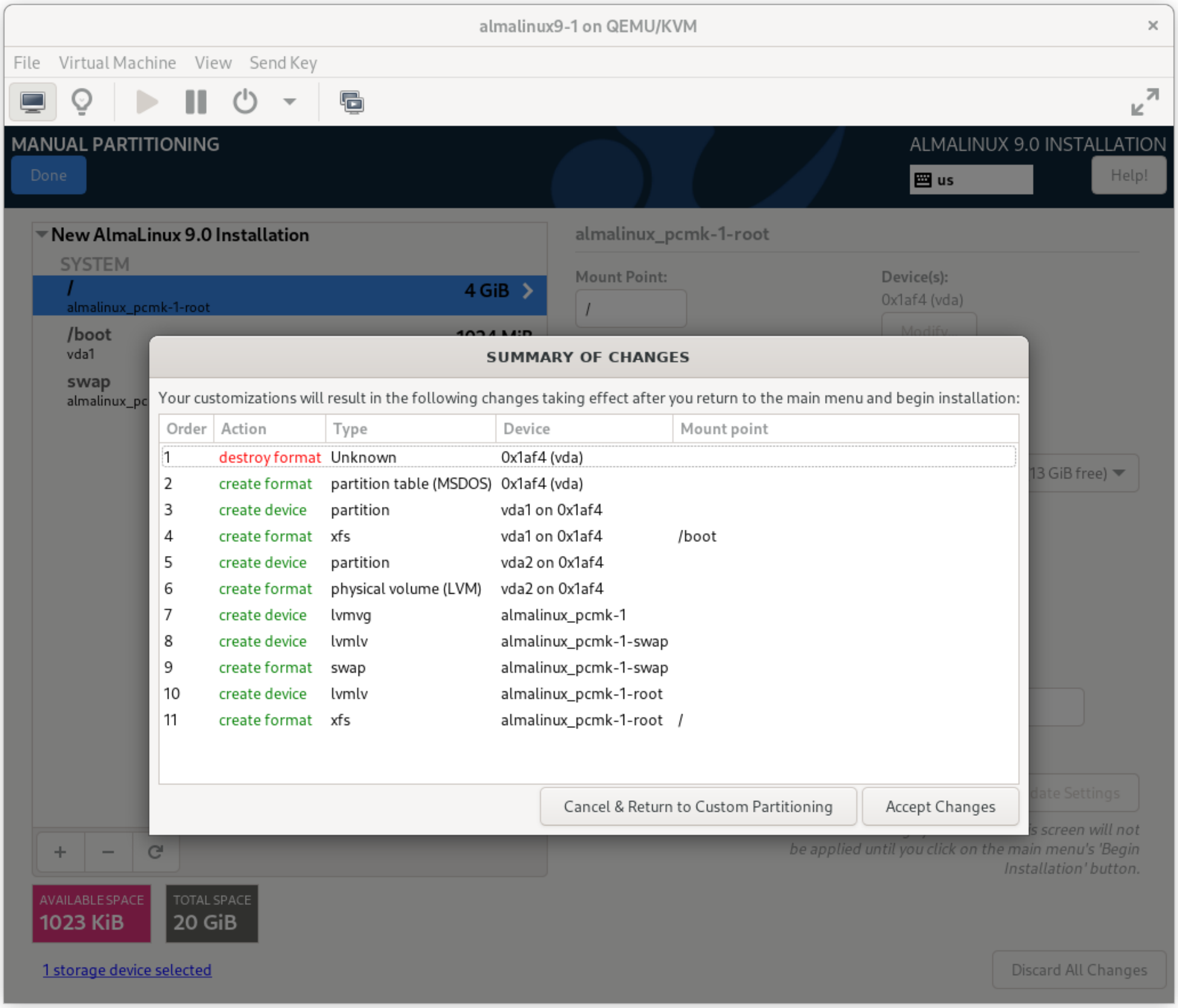


Fig. 8: AlmaLinux 9 Summary of Changes Dialog

Configure Time Synchronization

It is highly recommended to enable NTP on your cluster nodes. Doing so ensures all nodes agree on the current time and makes reading log files significantly easier.

AlmaLinux will enable NTP automatically. If you want to change any time-related settings (such as time zone or NTP server), you can do this in the **TIME & DATE** section. In this example, we configure the time zone as UTC (Coordinated Universal Time).

Root Password

In order to continue to the next step, a **Root Password** must be set. Be sure to check the box marked **Allow root SSH login with password**.

Press **Done**. (Depending on the password you chose, you may need to do so twice.)

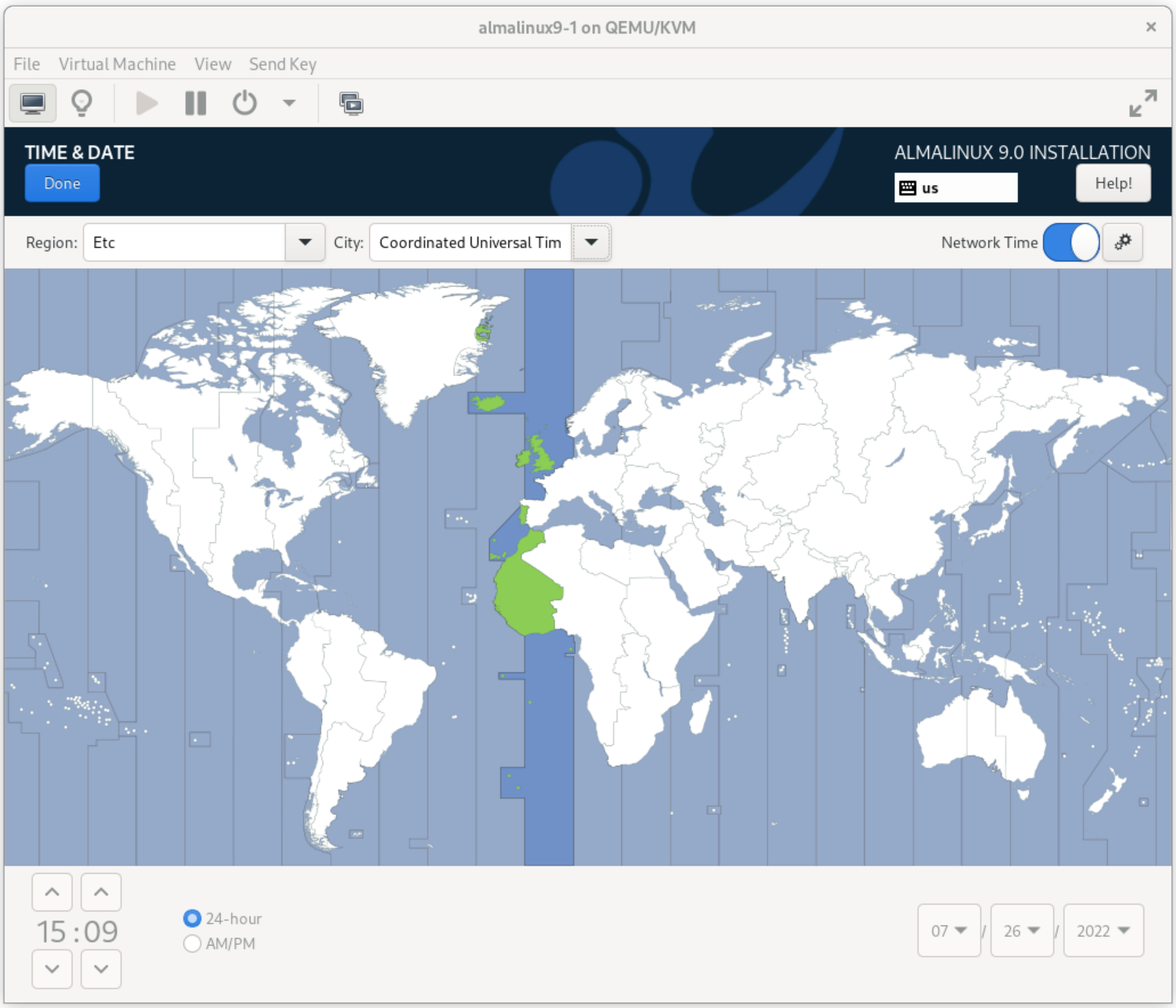


Fig. 9: AlmaLinux 9 Time & Date Screen

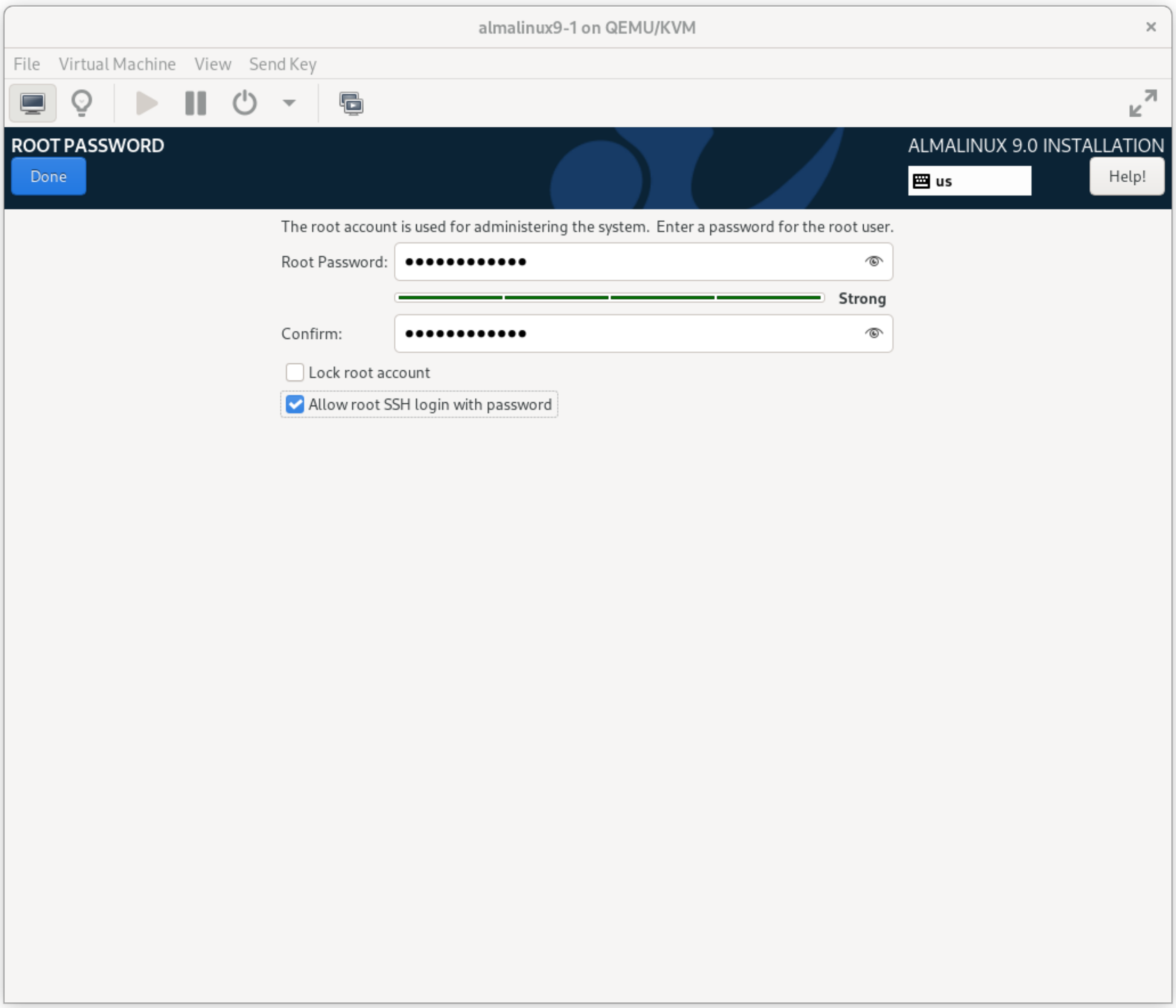


Fig. 10: AlmaLinux 9 Root Password Screen

Finish Install

Select **Begin Installation**. Once it completes, **Reboot System** as instructed. After the node reboots, you'll see a login prompt on the console. Login using `root` and the password you created earlier.

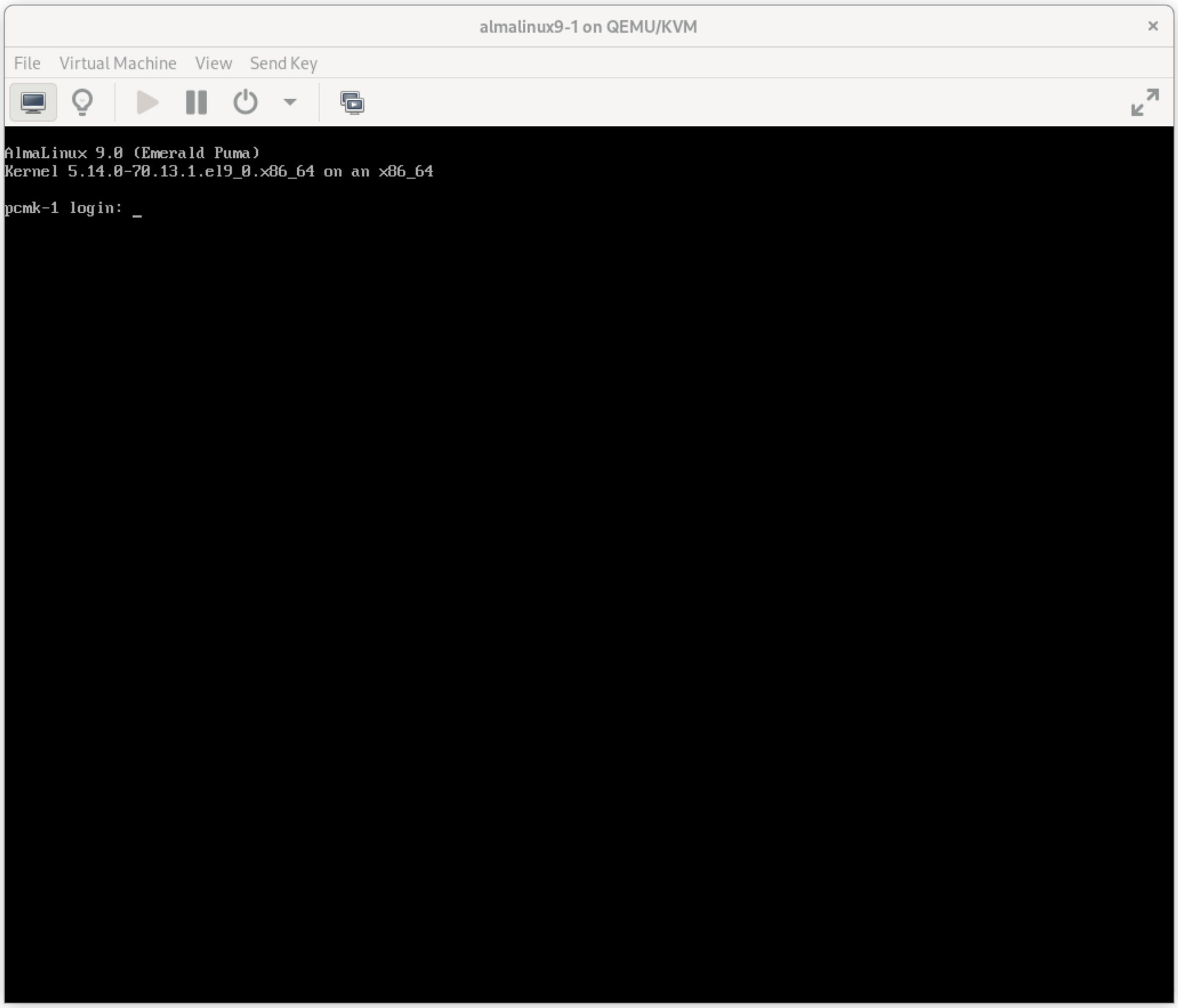


Fig. 11: AlmaLinux 9 Console Prompt

Note: From here on, we're going to be working exclusively from the terminal.

2.2.2 Configure the OS

Verify Networking

Ensure that the machine has the static IP address you configured earlier.

```
[root@pcmk-1 ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen␣
↪1000
    link/ether 52:54:00:32:cf:a9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.101/24 brd 192.168.122.255 scope global noprefixroute enp1s0
       valid_lft forever preferred_lft forever
    inet6 fe80::c3e1:3ba:959:fa96/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Note: If you ever need to change the node's IP address from the command line, follow these instructions, replacing `${conn}` with the name of your network connection. You can find the list of all network connection names by running `nmcli con show`; you can get details for each connection by running `nmcli con show ${conn}`.

```
[root@pcmk-1 ~]# nmcli con mod ${conn} ipv4.addresses "${new_address}"
[root@pcmk-1 ~]# nmcli con up ${conn}
```

Next, ensure that the routes are as expected:

```
[root@pcmk-1 ~]# ip route
default via 192.168.122.1 dev enp1s0 proto static metric 100
192.168.122.0/24 dev enp1s0 proto kernel scope link src 192.168.122.101 metric 100
```

If there is no line beginning with `default via`, then use `nmcli` to add a gateway:

```
[root@pcmk-1 ~]# nmcli con mod ${conn} ipv4.gateway "${new_gateway_addr}"
[root@pcmk-1 ~]# nmcli con up ${conn}
```

Now, check for connectivity to the outside world. Start small by testing whether we can reach the gateway we configured.

```
[root@pcmk-1 ~]# ping -c 1 192.168.122.1
PING 192.168.122.1 (192.168.122.1) 56(84) bytes of data.
64 bytes from 192.168.122.1: icmp_seq=1 ttl=64 time=0.492 ms

--- 192.168.122.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.492/0.492/0.492/0.000 ms
```

Now try something external; choose a location you know should be available.

```
[root@pcmk-1 ~]# ping -c 1 www.clusterlabs.org
PING mx1.clusterlabs.org (95.217.104.78) 56(84) bytes of data.
64 bytes from mx1.clusterlabs.org (95.217.104.78): icmp_seq=1 ttl=54 time=134 ms

--- mx1.clusterlabs.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 133.987/133.987/133.987/0.000 ms
```

Login Remotely

The console isn't a very friendly place to work from, so we will now switch to accessing the machine remotely via SSH where we can use copy and paste, etc.

From another host, check whether we can see the new host at all:

```
[gchin@gchin ~]$ ping -c 1 192.168.122.101
PING 192.168.122.101 (192.168.122.101) 56(84) bytes of data.
64 bytes from 192.168.122.101: icmp_seq=1 ttl=64 time=0.344 ms

--- 192.168.122.101 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.344/0.344/0.344/0.000 ms
```

Next, login as `root` via SSH.

```
[gchin@gchin ~]$ ssh root@192.168.122.101
The authenticity of host '192.168.122.101 (192.168.122.101)' can't be established.
ECDSA key fingerprint is SHA256:NBvcRrPDLIt39Rf0Tz4/f2Rd/FA5wUiDOd9bZ9QWWjo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.122.101' (ECDSA) to the list of known hosts.
root@192.168.122.101's password:
Last login: Tue Jan 10 20:46:30 2021
[root@pcmk-1 ~]#
```

Apply Updates

Apply any package updates released since your installation image was created:

```
[root@pcmk-1 ~]# dnf update -y
```

Use Short Node Names

During installation, we filled in the machine's fully qualified domain name (FQDN), which can be rather long when it appears in cluster logs and status output. See for yourself how the machine identifies itself:

```
[root@pcmk-1 ~]# uname -n
pcmk-1.localdomain
```

We can use the `hostnamectl` tool to strip off the domain name:

```
[root@pcmk-1 ~]# hostnamectl set-hostname $(uname -n | sed s/\\..*//)
```

Now, check that the machine is using the correct name:

```
[root@pcmk-1 ~]# uname -n
pcmk-1
```

You may want to reboot to ensure all updates take effect.

2.2.3 Repeat for Second Node

Repeat the installation steps so far, so that you have two nodes ready to have the cluster software installed.

For the purposes of this document, the additional node is called `pcmk-2` with address `192.168.122.102`.

2.2.4 Configure Communication Between Nodes

Configure Host Name Resolution

Confirm that you can communicate between the two new nodes:

```
[root@pcmk-1 ~]# ping -c 3 192.168.122.102
PING 192.168.122.102 (192.168.122.102) 56(84) bytes of data.
64 bytes from 192.168.122.102: icmp_seq=1 ttl=64 time=1.22 ms
64 bytes from 192.168.122.102: icmp_seq=2 ttl=64 time=0.795 ms
64 bytes from 192.168.122.102: icmp_seq=3 ttl=64 time=0.751 ms

--- 192.168.122.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.751/0.923/1.224/0.214 ms
```

Now we need to make sure we can communicate with the machines by their name. Add entries for the machines to `/etc/hosts` on both nodes. You can add entries for the machines to your DNS server if you have one, but this can create a single-point-of-failure (SPOF) if the DNS server goes down[1]. If you add entries to `/etc/hosts`, they should look something like the following:

```
[root@pcmk-1 ~]# grep pcmk /etc/hosts
192.168.122.101 pcmk-1.localdomain  pcmk-1
192.168.122.102 pcmk-2.localdomain  pcmk-2
```

We can now verify the setup by again using `ping`:

```
[root@pcmk-1 ~]# ping -c 3 pcmk-2
PING pcmk-2.localdomain (192.168.122.102) 56(84) bytes of data.
64 bytes from pcmk-2.localdomain (192.168.122.102): icmp_seq=1 ttl=64 time=0.295 ms
64 bytes from pcmk-2.localdomain (192.168.122.102): icmp_seq=2 ttl=64 time=0.616 ms
64 bytes from pcmk-2.localdomain (192.168.122.102): icmp_seq=3 ttl=64 time=0.809 ms

--- pcmk-2.localdomain ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.295/0.573/0.809/0.212 ms
```

Configure SSH

SSH is a convenient and secure way to copy files and perform commands remotely. For the purposes of this guide, we will create a key without a password (using the `-N` option) so that we can perform remote actions without being prompted.

> **Warning:** Unprotected SSH keys (those without a password) are not recommended for servers exposed to the outside world. We use them here only to simplify the demo.

Create a new key and allow anyone with that key to log in:

[1] You can also avoid this SPOF by specifying an `addr` option for each node when creating the cluster. We will discuss this in a later section.

Creating and Activating a New SSH Key

```
[root@pcmk-1 ~]# ssh-keygen -f ~/.ssh/id_rsa -N ""
Generating public/private rsa key pair.
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:h5AFPmXsGU4woOxRLYHW9lnU2wIQVOxpSRrsXbo/AX8 root@pcmk-1
The key's randomart image is:
+---[RSA 3072]----+
|   o+*BX*.       |
| .oo+.+*O o      |
| .+. +=% O o     |
| . .  =o%.o .    |
|  .    .S+..     |
|        ..o E    |
|         . o     |
|          o      |
|           .     |
+----[SHA256]-----+

[root@pcmk-1 ~]# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Install the key on the other node:

```
[root@pcmk-1 ~]# ssh-copy-id pcmk-2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host 'pcmk-2 (192.168.122.102)' can't be established.
ED25519 key fingerprint is SHA256:QkJnJ3fmszY7kAuuZ7wxUC5CC+eQThSCF13XYWnZJPo.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:1: 192.168.122.102
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
↪already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to
↪install the new keys
root@pcmk-2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'pcmk-2'"
and check to make sure that only the key(s) you wanted were added.
```

Test that you can now run commands remotely, without being prompted:

```
[root@pcmk-1 ~]# ssh pcmk-2 -- uname -n
pcmk-2
```

Finally, repeat this same process on the other node. For convenience, you can also generate an SSH key on your administrative machine and use `ssh-copy-id` to copy it to both cluster nodes.

2.3 Set up a Cluster

2.3.1 Simplify Administration With a Cluster Shell

In the dark past, configuring Pacemaker required the administrator to read and write XML. In true UNIX style, there were also a number of different commands that specialized in different aspects of querying and updating the cluster.

In addition, the various components of the cluster stack (Corosync, Pacemaker, etc.) had to be configured separately, with different configuration tools and formats.

All of that has been greatly simplified with the creation of higher-level tools, whether command-line or GUIs, that hide all the mess underneath.

Command-line cluster shells take all the individual aspects required for managing and configuring a cluster, and pack them into one simple-to-use command-line tool.

They even allow you to queue up several changes at once and commit them all at once.

Two popular command-line shells are `pcs` and `crmsh`. Clusters from Scratch is based on `pcs` because it comes with AlmaLinux, but both have similar functionality. Choosing a shell or GUI is a matter of personal preference and what comes with (and perhaps is supported by) your choice of operating system.

2.3.2 Install the Cluster Software

Fire up a shell on both nodes and run the following to activate the High Availability repo.

```
# dnf config-manager --set-enabled highavailability
```

Important: This document will show commands that need to be executed on both nodes with a simple `#` prompt. Be sure to run them on each node individually.

Now, we'll install `pacemaker`, `pcs`, and some other command-line tools that will make our lives easier:

```
# dnf install -y pacemaker pcs psmisc policycoreutils-python3
```

Note: This document uses `pcs` for cluster management. Other alternatives, such as `crmsh`, are available, but their syntax will differ from the examples used here.

2.3.3 Configure the Cluster Software

Allow cluster services through firewall

On each node, allow cluster-related services through the local firewall:

```
# firewall-cmd --permanent --add-service=high-availability
success
# firewall-cmd --reload
success
```

Note: If you are using `iptables` directly, or some other firewall solution besides `firewalld`, simply open the following ports, which can be used by various clustering components: TCP ports 2224, 3121, and 21064, and UDP port 5405.

If you run into any problems during testing, you might want to disable the firewall and SELinux entirely until you have everything working. This may create significant security issues and should not be performed on machines that will be exposed to the outside world, but may be appropriate during development and testing on a protected host.

To disable security measures:

```
[root@pcmk-1 ~]# setenforce 0
[root@pcmk-1 ~]# sed -i.bak "s/SELINUX=enforcing/SELINUX=permissive/g" /etc/selinux/config
[root@pcmk-1 ~]# systemctl mask firewalld.service
[root@pcmk-1 ~]# systemctl stop firewalld.service
[root@pcmk-1 ~]# iptables --flush
```

Enable `pcs` Daemon

Before the cluster can be configured, the `pcs` daemon must be started and enabled to start at boot time on each node. This daemon works with the `pcs` command-line interface to manage synchronizing the Corosync configuration across all nodes in the cluster, among other functions.

Start and enable the daemon by issuing the following commands on each node:

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
Created symlink from /etc/systemd/system/multi-user.target.wants/pcsd.service to /usr/lib/systemd/
↪system/pcsd.service.
```

The installed packages will create an `hacluster` user with a disabled password. While this is fine for running `pcs` commands locally, the account needs a login password in order to perform such tasks as syncing the Corosync configuration, or starting and stopping the cluster on other nodes.

This tutorial will make use of such commands, so now we will set a password for the `hacluster` user, using the same password on both nodes:

```
# passwd hacluster
Changing password for user hacluster.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Note: Alternatively, to script this process or set the password on a different machine from the one you're logged into, you can use the `--stdin` option for `passwd`:

```
[root@pcmk-1 ~]# ssh pcmk-2 -- 'echo mysupersecretpassword | passwd --stdin hacluster'
```

Configure Corosync

On either node, use `pcs host auth` to authenticate as the `hacluster` user:

```
[root@pcmk-1 ~]# pcs host auth pcmk-1 pcmk-2
Username: hacluster
Password:
pcmk-2: Authorized
pcmk-1: Authorized
```

Next, use `pcs cluster setup` on the same node to generate and synchronize the Corosync configuration:

```
[root@pcmk-1 ~]# pcs cluster setup mycluster pcmk-1 pcmk-2
No addresses specified for host 'pcmk-1', using 'pcmk-1'
No addresses specified for host 'pcmk-2', using 'pcmk-2'
Destroying cluster on hosts: 'pcmk-1', 'pcmk-2'...
pcmk-2: Successfully destroyed cluster
pcmk-1: Successfully destroyed cluster
Requesting remove 'pcsd settings' from 'pcmk-1', 'pcmk-2'
pcmk-1: successful removal of the file 'pcsd settings'
pcmk-2: successful removal of the file 'pcsd settings'
Sending 'corosync authkey', 'pacemaker authkey' to 'pcmk-1', 'pcmk-2'
pcmk-1: successful distribution of the file 'corosync authkey'
pcmk-1: successful distribution of the file 'pacemaker authkey'
pcmk-2: successful distribution of the file 'corosync authkey'
pcmk-2: successful distribution of the file 'pacemaker authkey'
Sending 'corosync.conf' to 'pcmk-1', 'pcmk-2'
pcmk-1: successful distribution of the file 'corosync.conf'
pcmk-2: successful distribution of the file 'corosync.conf'
Cluster has been successfully set up.
```

Note: If you'd like, you can specify an `addr` option for each node in the `pcs cluster setup` command. This will create an explicit name-to-address mapping for each node in `/etc/corosync/corosync.conf`, eliminating the need for hostname resolution via DNS, `/etc/hosts`, and the like.

```
[root@pcmk-1 ~]# pcs cluster setup mycluster \
    pcmk-1 addr=192.168.122.101 pcmk-2 addr=192.168.122.102
```

If you received an authorization error for either of those commands, make sure you configured the `hacluster` user account on each node with the same password.

The final `corosync.conf` configuration on each node should look something like the sample in *Sample Corosync Configuration*.

2.3.4 Explore pcs

Start by taking some time to familiarize yourself with what `pcs` can do.

```
[root@pcmk-1 ~]# pcs

Usage: pcs [-f file] [-h] [commands]...
Control and configure pacemaker and corosync.

Options:
    -h, --help         Display usage and exit.
    -f file            Perform actions on file instead of active CIB.
                       Commands supporting the option use the initial state of
                       the specified file as their input and then overwrite the
```

(continues on next page)

(continued from previous page)

```
                       file with the state reflecting the requested
                       operation(s).
                       A few commands only use the specified file in read-only
                       mode since their effect is not a CIB modification.
    --debug            Print all network traffic and external commands run.
    --version          Print pcs version information. List pcs capabilities if
                       --full is specified.
    --request-timeout  Timeout for each outgoing request to another node in
                       seconds. Default is 60s.
    --force            Override checks and errors, the exact behavior depends on
                       the command. WARNING: Using the --force option is
                       strongly discouraged unless you know what you are doing.

Commands:
    cluster     Configure cluster options and nodes.
    resource    Manage cluster resources.
    stonith     Manage fence devices.
    constraint  Manage resource constraints.
    property    Manage pacemaker properties.
    acl         Manage pacemaker access control lists.
    qdevice     Manage quorum device provider on the local host.
    quorum      Manage cluster quorum settings.
    booth       Manage booth (cluster ticket manager).
    status      View cluster status.
    config      View and manage cluster configuration.
    pcsd        Manage pcs daemon.
    host        Manage hosts known to pcs/pcsd.
    node        Manage cluster nodes.
    alert       Manage pacemaker alerts.
    client      Manage pcsd client configuration.
    dr          Manage disaster recovery configuration.
    tag         Manage pacemaker tags.
```

As you can see, the different aspects of cluster management are separated into categories. To discover the functionality available in each of these categories, one can issue the command `pcs <CATEGORY> help`. Below is an example of all the options available under the status category.

```
[root@pcmk-1 ~]# pcs status help

Usage: pcs status [commands]...
View current cluster and resource status
Commands:
    [status] [--full] [--hide-inactive]
        View all information about the cluster and resources (--full provides
        more details, --hide-inactive hides inactive resources).

    resources [<resource id | tag id>] [node=<node>] [--hide-inactive]
        Show status of all currently configured resources. If --hide-inactive
        is specified, only show active resources.  If a resource or tag id is
        specified, only show status of the specified resource or resources in
        the specified tag. If node is specified, only show status of resources
        configured for the specified node.

    cluster
        View current cluster status.
```

(continues on next page)

(continued from previous page)

```
    corosync
        View current membership information as seen by corosync.

    quorum
        View current quorum status.

    qdevice <device model> [--full] [<cluster name>]
        Show runtime status of specified model of quorum device provider.  Using
        --full will give more detailed output.  If <cluster name> is specified,
        only information about the specified cluster will be displayed.

    booth
        Print current status of booth on the local node.

    nodes [corosync | both | config]
        View current status of nodes from pacemaker. If 'corosync' is
        specified, view current status of nodes from corosync instead. If
        'both' is specified, view current status of nodes from both corosync &
        pacemaker. If 'config' is specified, print nodes from corosync &
        pacemaker configuration.

    pcsd [<node>]...
        Show current status of pcsd on nodes specified, or on all nodes
        configured in the local cluster if no nodes are specified.

    xml
        View xml version of status (output from crm_mon -r -1 -X).
```

Additionally, if you are interested in the version and supported cluster stack(s) available with your Pacemaker installation, run:

```
[root@pcmk-1 ~]# pacemakerd --features
 Pacemaker 2.1.2-4.el9 (Build: ada5c3b36e2)
 Supporting v3.13.0: agent-manpages cibsecrets corosync-ge-2 default-concurrent-fencing default-
↪resource-stickiness default-sbd-sync generated-manpages monotonic nagios ncurses remote systemd
```

2.4 Start and Verify Cluster

2.4.1 Start the Cluster

Now that Corosync is configured, it is time to start the cluster. The command below will start the `corosync` and `pacemaker` services on both nodes in the cluster.

```
[root@pcmk-1 ~]# pcs cluster start --all
pcmk-1: Starting Cluster...
pcmk-2: Starting Cluster...
```

Note: An alternative to using the `pcs cluster start --all` command is to issue either of the below command sequences on each node in the cluster separately:

```
# pcs cluster start
Starting Cluster...
```

or

```
# systemctl start corosync.service
# systemctl start pacemaker.service
```

Important: In this example, we are not enabling the `corosync` and `pacemaker` services to start at boot. If a cluster node fails or is rebooted, you will need to run `pcs cluster start [<NODENAME> | --all]` to start the cluster on it. While you can enable the services to start at boot (for example, using `pcs cluster enable [<NODENAME> | --all]`), requiring a manual start of cluster services gives you the opportunity to do a post-mortem investigation of a node failure before returning it to the cluster.

2.4.2 Verify Corosync Installation

First, use `corosync-cfgtool` to check whether cluster communication is happy:

```
[root@pcmk-1 ~]# corosync-cfgtool -s
Local node ID 1, transport knet
LINK ID 0 udp
        addr        = 192.168.122.101
        status:
                nodeid:          1: localhost
                nodeid:          2: connected
```

We can see here that everything appears normal with our fixed IP address (not a `127.0.0.x` loopback address) listed as the `addr`, and `localhost` and `connected` for the statuses of nodeid 1 and nodeid 2, respectively.

If you see something different, you might want to start by checking the node's network, firewall, and SELinux configurations.

Next, check the membership and quorum APIs:

```
[root@pcmk-1 ~]# corosync-cmapctl | grep members
runtime.members.1.config_version (u64) = 0
runtime.members.1.ip (str) = r(0) ip(192.168.122.101)
runtime.members.1.join_count (u32) = 1
runtime.members.1.status (str) = joined
runtime.members.2.config_version (u64) = 0
runtime.members.2.ip (str) = r(0) ip(192.168.122.102)
runtime.members.2.join_count (u32) = 1
runtime.members.2.status (str) = joined

[root@pcmk-1 ~]# pcs status corosync

Membership information
----------------------
    Nodeid      Votes Name
         1          1 pcmk-1 (local)
         2          1 pcmk-2
```

You should see both nodes have joined the cluster.

2.4.3 Verify Pacemaker Installation

Now that we have confirmed that Corosync is functional, we can check the rest of the stack. Pacemaker has already been started, so verify the necessary processes are running:

```
[root@pcmk-1 ~]# ps axf
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
...lots of processes...
17121 ?        SLsl   0:01 /usr/sbin/corosync -f
17133 ?        Ss     0:00 /usr/sbin/pacemakerd
17134 ?        Ss     0:00  \_ /usr/libexec/pacemaker/pacemaker-based
17135 ?        Ss     0:00  \_ /usr/libexec/pacemaker/pacemaker-fenced
17136 ?        Ss     0:00  \_ /usr/libexec/pacemaker/pacemaker-execd
17137 ?        Ss     0:00  \_ /usr/libexec/pacemaker/pacemaker-attrd
17138 ?        Ss     0:00  \_ /usr/libexec/pacemaker/pacemaker-schedulerd
17139 ?        Ss     0:00  \_ /usr/libexec/pacemaker/pacemaker-controld
```

If that looks OK, check the `pcs status` output:

```
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster

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

Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-2 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:09:55 2022
  * Last change:  Wed Jul 27 00:07:08 2022 by hacluster via crmd on pcmk-2
  * 2 nodes configured
  * 0 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * No resources

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

Finally, ensure there are no start-up errors from `corosync` or `pacemaker` (aside from messages relating to not having STONITH configured, which are OK at this point):

```
[root@pcmk-1 ~]# journalctl -b | grep -i error
```

Note: Other operating systems may report startup errors in other locations (for example, `/var/log/messages`).

Repeat these checks on the other node. The results should be the same.

2.4.4 Explore the Existing Configuration

For those who are not of afraid of XML, you can see the raw cluster configuration and status by using the `pcs cluster cib` command.

The last XML you'll see in this document

```
[root@pcmk-1 ~]# pcs cluster cib
```


```
<cib crm_feature_set="3.13.0" validate-with="pacemaker-3.8" epoch="5" num_updates="4" admin_
↪epoch="0" cib-last-written="Wed Jul 27 00:07:08 2022" update-origin="pcmk-2" update-client=
↪"crmd" update-user="hacluster" have-quorum="1" dc-uuid="2">
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <nvpair id="cib-bootstrap-options-have-watchdog" name="have-watchdog" value="false"/>
        <nvpair id="cib-bootstrap-options-dc-version" name="dc-version" value="2.1.2-4.el9-
↪ada5c3b36e2"/>
        <nvpair id="cib-bootstrap-options-cluster-infrastructure" name="cluster-infrastructure"␣
↪value="corosync"/>
        <nvpair id="cib-bootstrap-options-cluster-name" name="cluster-name" value="mycluster"/>
      </cluster_property_set>
    </crm_config>
    <nodes>
      <node id="1" uname="pcmk-1"/>
      <node id="2" uname="pcmk-2"/>
    </nodes>
    <resources/>
    <constraints/>
    <rsc_defaults>
      <meta_attributes id="build-resource-defaults">
        <nvpair id="build-resource-stickiness" name="resource-stickiness" value="1"/>
      </meta_attributes>
    </rsc_defaults>
  </configuration>
  <status>
    <node_state id="2" uname="pcmk-2" in_ccm="true" crmd="online" crm-debug-origin="do_state_
↪transition" join="member" expected="member">
      <lrm id="2">
        <lrm_resources/>
      </lrm>
    </node_state>
    <node_state id="1" uname="pcmk-1" in_ccm="true" crmd="online" crm-debug-origin="do_state_
↪transition" join="member" expected="member">
      <lrm id="1">
        <lrm_resources/>
      </lrm>
    </node_state>
  </status>
</cib>
```


Before we make any changes, it's a good idea to check the validity of the configuration.

```
[root@pcmk-1 ~]# pcs cluster verify --full
Error: invalid cib:
(unpack_resources)  error: Resource start-up disabled since no STONITH resources have been defined
```

(continues on next page)

(continued from previous page)

```
(unpack_resources)  error: Either configure some or disable STONITH with the stonith-enabled option
(unpack_resources)  error: NOTE: Clusters with shared data need STONITH to ensure data integrity
crm_verify: Errors found during check: config not valid

Error: Errors have occurred, therefore pcs is unable to continue
```

As you can see, the tool has found some errors. The cluster will not start any resources until we configure STONITH.

2.5 Configure Fencing

2.5.1 What is Fencing?

Fencing protects your data from being corrupted, and your application from becoming unavailable, due to unintended concurrent access by rogue nodes.

Just because a node is unresponsive doesn't mean it has stopped accessing your data. The only way to be 100% sure that your data is safe, is to use fencing to ensure that the node is truly offline before allowing the data to be accessed from another node.

Fencing also has a role to play in the event that a clustered service cannot be stopped. In this case, the cluster uses fencing to force the whole node offline, thereby making it safe to start the service elsewhere.

Fencing is also known as STONITH, an acronym for "Shoot The Other Node In The Head", since the most popular form of fencing is cutting a host's power.

In order to guarantee the safety of your data[1], fencing is enabled by default.

Note: It is possible to tell the cluster not to use fencing, by setting the `stonith-enabled` cluster property to false:

```
[root@pcmk-1 ~]# pcs property set stonith-enabled=false
[root@pcmk-1 ~]# pcs cluster verify --full
```

However, this is completely inappropriate for a production cluster. It tells the cluster to simply pretend that failed nodes are safely powered off. Some vendors will refuse to support clusters that have fencing disabled. Even disabling it for a test cluster means you won't be able to test real failure scenarios.

2.5.2 Choose a Fence Device

The two broad categories of fence device are power fencing, which cuts off power to the target, and fabric fencing, which cuts off the target's access to some critical resource, such as a shared disk or access to the local network.

Power fencing devices include:

- Intelligent power switches
- IPMI
- Hardware watchdog device (alone, or in combination with shared storage used as a "poison pill" mechanism)

[1] If the data is corrupt, there is little point in continuing to make it available.

Fabric fencing devices include:

- Shared storage that can be cut off for a target host by another host (for example, an external storage device that supports SCSI-3 persistent reservations)
- Intelligent network switches

Using IPMI as a power fencing device may seem like a good choice. However, if the IPMI shares power and/or network access with the host (such as most onboard IPMI controllers), a power or network failure will cause both the host and its fencing device to fail. The cluster will be unable to recover, and must stop all resources to avoid a possible split-brain situation.

Likewise, any device that relies on the machine being active (such as SSH-based "devices" sometimes used during testing) is inappropriate, because fencing will be required when the node is completely unresponsive. (Fence agents like `fence_ilo_ssh`, which connects via SSH to an HP iLO but not to the cluster node, are fine.)

2.5.3 Configure the Cluster for Fencing

1. Install the fence agent(s). To see what packages are available, run `dnf search fence-`. Be sure to install the package(s) on all cluster nodes.
2. Configure the fence device itself to be able to fence your nodes and accept fencing requests. This includes any necessary configuration on the device and on the nodes, and any firewall or SELinux changes needed. Test the communication between the device and your nodes.
3. Find the name of the correct fence agent: `pcs stonith list`
4. Find the parameters associated with the device: `pcs stonith describe <AGENT_NAME>`
5. Create a local copy of the CIB: `pcs cluster cib stonith_cfg`
6. Create the fencing resource: `pcs -f stonith_cfg stonith create <STONITH_ID> <STONITH_DEVICE_TYPE> [STONITH_DEVICE_OPTIONS]`

   Any flags that do not take arguments, such as `--ssl`, should be passed as `ssl=1`.
7. Ensure fencing is enabled in the cluster: `pcs -f stonith_cfg property set stonith-enabled=true`
8. If the device does not know how to fence nodes based on their cluster node name, you may also need to set the special `pcmk_host_map` parameter. See `man pacemaker-fenced` for details.
9. If the device does not support the `list` command, you may also need to set the special `pcmk_host_list` and/or `pcmk_host_check` parameters. See `man pacemaker-fenced` for details.
10. If the device does not expect the target to be specified with the `port` parameter, you may also need to set the special `pcmk_host_argument` parameter. See `man pacemaker-fenced` for details.
11. Commit the new configuration: `pcs cluster cib-push stonith_cfg`
12. Once the fence device resource is running, test it (you might want to stop the cluster on that machine first): `pcs stonith fence <NODENAME>`

2.5.4 Example

For this example, assume we have a chassis containing four nodes and a separately powered IPMI device active on `10.0.0.1`. Following the steps above would go something like this:

Step 1: Install the `fence-agents-ipmilan` package on both nodes.

Step 2: Configure the IP address, authentication credentials, etc. in the IPMI device itself.

Step 3: Choose the `fence_ipmilan` STONITH agent.

Step 4: Obtain the agent's possible parameters:

```
[root@pcmk-1 ~]# pcs stonith describe fence_ipmilan
fence_ipmilan - Fence agent for IPMI

fence_ipmilan is an I/O Fencing agentwhich can be used with machines controlled by IPMI.This agent␣
↪calls support software ipmitool (http://ipmitool.sf.net/). WARNING! This fence agent might␣
↪report success before the node is powered off. You should use -m/method onoff if your fence␣
↪device works correctly with that option.

Stonith options:
  auth: IPMI Lan Auth type.
  cipher: Ciphersuite to use (same as ipmitool -C parameter)
  hexadecimal_kg: Hexadecimal-encoded Kg key for IPMIv2 authentication
  ip: IP address or hostname of fencing device
  ipport: TCP/UDP port to use for connection with device
  lanplus: Use Lanplus to improve security of connection
  method: Method to fence
  password: Login password or passphrase
  password_script: Script to run to retrieve password
  plug: IP address or hostname of fencing device (together with --port-as-ip)
  privlvl: Privilege level on IPMI device
  target: Bridge IPMI requests to the remote target address
  username: Login name
  quiet: Disable logging to stderr. Does not affect --verbose or --debug-file or logging to syslog.
  verbose: Verbose mode. Multiple -v flags can be stacked on the command line (e.g., -vvv) to␣
↪increase verbosity.
  verbose_level: Level of debugging detail in output. Defaults to the number of --verbose flags␣
↪specified on the command line, or to 1 if verbose=1 in a stonith device configuration (i.e., on␣
↪stdin).
  debug_file: Write debug information to given file
  delay: Wait X seconds before fencing is started
  disable_timeout: Disable timeout (true/false) (default: true when run from Pacemaker 2.0+)
  ipmitool_path: Path to ipmitool binary
  login_timeout: Wait X seconds for cmd prompt after login
  port_as_ip: Make "port/plug" to be an alias to IP address
  power_timeout: Test X seconds for status change after ON/OFF
  power_wait: Wait X seconds after issuing ON/OFF
  shell_timeout: Wait X seconds for cmd prompt after issuing command
  stonith_status_sleep: Sleep X seconds between status calls during a STONITH action
  ipmitool_timeout: Timeout (sec) for IPMI operation
  retry_on: Count of attempts to retry power on
  use_sudo: Use sudo (without password) when calling 3rd party software
  sudo_path: Path to sudo binary
  pcmk_host_map: A mapping of host names to ports numbers for devices that do not support host␣
↪names. Eg. node1:1;node2:2,3 would tell the cluster to use port 1 for node1 and ports 2 and 3␣
↪for node2
  pcmk_host_list: A list of machines controlled by this device (Optional unless pcmk_host_
↪check=static-list).
  pcmk_host_check: How to determine which machines are controlled by the device. Allowed values:␣
↪dynamic-list (query the device via the 'list' command), static-list (check the pcmk_host_list␣
↪attribute), status
                  (query the device via the 'status' command), none (assume every device can␣
↪fence every machine)
  pcmk_delay_max: Enable a delay of no more than the time specified before executing fencing␣
↪actions. Pacemaker derives the overall delay by taking the value of pcmk_delay_base and adding a␣
↪random delay value
```

(continues on next page)

(continued from previous page)

```
                  such that the sum is kept below this maximum. This prevents double fencing when␣
↪using slow devices such as sbd. Use this to enable a random delay for fencing actions. The␣
↪overall delay is
                  derived from this random delay value adding a static delay so that the sum is␣
↪kept below the maximum delay.
  pcmk_delay_base: Enable a base delay for fencing actions and specify base delay value. This␣
↪enables a static delay for fencing actions, which can help avoid "death matches" where two nodes␣
↪try to fence each
                   other at the same time. If pcmk_delay_max is also used, a random delay will be␣
↪added such that the total delay is kept below that value. This can be set to a single time value␣
↪to apply to any
                   node targeted by this device (useful if a separate device is configured for␣
↪each target), or to a node map (for example, "node1:1s;node2:5") to set a different value per␣
↪target.
  pcmk_action_limit: The maximum number of actions can be performed in parallel on this device␣
↪Cluster property concurrent-fencing=true needs to be configured first. Then use this to specify␣
↪the maximum number
                      of actions can be performed in parallel on this device. -1 is unlimited.

Default operations:
  monitor: interval=60s
```

Step 5: `pcs cluster cib stonith_cfg`

Step 6: Here are example parameters for creating our fence device resource:

```
[root@pcmk-1 ~]# pcs -f stonith_cfg stonith create ipmi-fencing fence_ipmilan \
      pcmk_host_list="pcmk-1 pcmk-2" ipaddr=10.0.0.1 login=testuser \
      passwd=acd123 op monitor interval=60s
[root@pcmk-1 ~]# pcs -f stonith_cfg stonith
  * ipmi-fencing    (stonith:fence_ipmilan):        Stopped
```

Steps 7-10: Enable fencing in the cluster:

```
[root@pcmk-1 ~]# pcs -f stonith_cfg property set stonith-enabled=true
[root@pcmk-1 ~]# pcs -f stonith_cfg property
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: mycluster
 dc-version: 2.0.5-4.el8-ba59be7122
 have-watchdog: false
 stonith-enabled: true
```

Step 11: `pcs cluster cib-push stonith_cfg --config`

Step 12: Test:

```
[root@pcmk-1 ~]# pcs cluster stop pcmk-2
[root@pcmk-1 ~]# pcs stonith fence pcmk-2
```

After a successful test, login to any rebooted nodes, and start the cluster (with `pcs cluster start`).

2.6 Create an Active/Passive Cluster

2.6.1 Add a Resource

Our first resource will be a floating IP address that the cluster can bring up on either node. Regardless of where any cluster service(s) are running, end users need to be able to communicate with them at a consistent address. Here, we will use `192.168.122.120` as the floating IP address, give it the imaginative name `ClusterIP`, and tell the cluster to check whether it is still running every 30 seconds.

> **Warning:** The chosen address must not already be in use on the network, on a cluster node or elsewhere. Do not reuse an IP address one of the nodes already has configured.

```
[root@pcmk-1 ~]# pcs resource create ClusterIP ocf:heartbeat:IPaddr2 \
    ip=192.168.122.120 cidr_netmask=24 op monitor interval=30s
```

Another important piece of information here is `ocf:heartbeat:IPaddr2`. This tells Pacemaker three things about the resource you want to add:

- The first field (`ocf` in this case) is the standard to which the resource agent conforms and where to find it.
- The second field (`heartbeat` in this case) is known as the provider. Currently, this field is supported only for OCF resources. It tells Pacemaker which OCF namespace the resource script is in.
- The third field (`IPaddr2` in this case) is the name of the resource agent, the executable file responsible for starting, stopping, monitoring, and possibly promoting and demoting the resource.

To obtain a list of the available resource standards (the `ocf` part of `ocf:heartbeat:IPaddr2`), run:

```
[root@pcmk-1 ~]# pcs resource standards
lsb
ocf
service
systemd
```

To obtain a list of the available OCF resource providers (the `heartbeat` part of `ocf:heartbeat:IPaddr2`), run:

```
[root@pcmk-1 ~]# pcs resource providers
heartbeat
openstack
pacemaker
```

Finally, if you want to see all the resource agents available for a specific OCF provider (the `IPaddr2` part of `ocf:heartbeat:IPaddr2`), run:

```
[root@pcmk-1 ~]# pcs resource agents ocf:heartbeat
apache
conntrackd
corosync-qnetd
.
. (skipping lots of resources to save space)
.
VirtualDomain
Xinetd
```

If you want to list all resource agents available on the system, run `pcs resource list`. We'll skip that here.

Now, verify that the IP resource has been added, and display the cluster's status to see that it is now active. Note: There should be a stonith device by now, but it's okay if it doesn't look like the one below.

```
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:37:28 2022
  * Last change:  Wed Jul 27 00:37:14 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-2

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

On the node where the `ClusterIP` resource is running, verify that the address has been added.

```
[root@pcmk-2 ~]# ip -o addr show
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
1: lo    inet6 ::1/128 scope host \       valid_lft forever preferred_lft forever
2: enp1s0    inet 192.168.122.102/24 brd 192.168.122.255 scope global noprefixroute enp1s0\      ␣
↪valid_lft forever preferred_lft forever
2: enp1s0    inet 192.168.122.120/24 brd 192.168.122.255 scope global secondary enp1s0\      ␣
↪valid_lft forever preferred_lft forever
2: enp1s0    inet6 fe80::5054:ff:fe95:209/64 scope link noprefixroute \       valid_lft forever␣
↪preferred_lft forever
```

2.6.2 Perform a Failover

Since our ultimate goal is high availability, we should test failover of our new resource before moving on.

First, from the `pcs status` output in the previous step, find the node on which the IP address is running. You can see that the status of the `ClusterIP` resource is `Started` on a particular node (in this example, `pcmk-2`). Shut down `pacemaker` and `corosync` on that machine to trigger a failover.

```
[root@pcmk-2 ~]# pcs cluster stop pcmk-2
pcmk-2: Stopping Cluster (pacemaker)...
pcmk-2: Stopping Cluster (corosync)...
```

Note: A cluster command such as `pcs cluster stop <NODENAME>` can be run from any node in the cluster, not just the node where the cluster services will be stopped. Running `pcs cluster stop` without a `<NODENAME>` stops the cluster services on the local host. The same is true for `pcs cluster start` and many other such commands.

Verify that pacemaker and corosync are no longer running:

```
[root@pcmk-2 ~]# pcs status
Error: error running crm_mon, is pacemaker running?
  Could not connect to pacemakerd: Connection refused
  crm_mon: Connection to cluster failed: Connection refused
```

Go to the other node, and check the cluster status.

```
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:43:51 2022
  * Last change:  Wed Jul 27 00:43:14 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ pcmk-1 ]
  * OFFLINE: [ pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1

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

Notice that pcmk-2 is OFFLINE for cluster purposes (its pcsd is still active, allowing it to receive pcs commands, but it is not participating in the cluster).

Also notice that ClusterIP is now running on pcmk-1 – failover happened automatically, and no errors are reported.

Quorum

If a cluster splits into two (or more) groups of nodes that can no longer communicate with each other (a.k.a. *partitions*), *quorum* is used to prevent resources from starting on more nodes than desired, which would risk data corruption.

A cluster has quorum when more than half of all known nodes are online in the same partition, or for the mathematically inclined, whenever the following inequality is true:

```
total_nodes < 2 * active_nodes
```

For example, if a 5-node cluster split into 3- and 2-node paritions, the 3-node partition would have quorum and could continue serving resources. If a 6-node cluster split into two 3-node partitions, neither partition would have quorum; Pacemaker's default behavior in such cases is to stop all resources, in order to prevent data corruption.

Two-node clusters are a special case. By the above definition, a two-node cluster would only have quorum when both nodes are running. This would make the creation of a two-node cluster pointless. However, Corosync has the ability to require only one node for quorum in a two-node cluster.

The `pcs cluster setup` command will automatically configure `two_node: 1` in `corosync.conf`, so a two-node cluster will "just work".

Note: You might wonder, "What if the nodes in a two-node cluster can't communicate with each other? Wouldn't this `two_node: 1` setting create a split-brain scenario, in which each node has quorum separately and they both try to manage the same cluster resources?"

As long as fencing is configured, there is no danger of this. If the nodes lose contact with each other, each node will try to fence the other node. Resource management is disabled until fencing succeeds; neither node is allowed to start, stop, promote, or demote resources.

After fencing succeeds, the surviving node can safely recover any resources that were running on the fenced node.

If the fenced node boots up and rejoins the cluster, it does not have quorum until it can communicate with the surviving node at least once. This prevents "fence loops," in which a node gets fenced, reboots, rejoins the cluster, and fences the other node. This protective behavior is controlled by the `wait_for_all: 1` option, which is enabled automatically when `two_node: 1` is configured.

If you are using a different cluster shell, you may have to configure `corosync.conf` appropriately yourself.

Now, simulate node recovery by restarting the cluster stack on `pcmk-2`, and check the cluster's status. (It may take a little while before the cluster gets going on the node, but it eventually will look like the below.)

```
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:45:17 2022
  * Last change:  Wed Jul 27 00:45:01 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1

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

2.6.3 Prevent Resources from Moving after Recovery

In most circumstances, it is highly desirable to prevent healthy resources from being moved around the cluster. Moving resources almost always requires a period of downtime. For complex services such as databases, this period can be quite long.

To address this, Pacemaker has the concept of resource *stickiness*, which controls how strongly a service prefers to stay running where it is. You may like to think of it as the "cost" of any downtime. By default,[1]

[1] Zero resource stickiness is Pacemaker's default if you remove the default value that was created at cluster setup time, or if

Pacemaker assumes there is zero cost associated with moving resources and will do so to achieve "optimal"[2] resource placement. We can specify a different stickiness for every resource, but it is often sufficient to change the default.

In AlmaLinux 9, the cluster setup process automatically configures a default resource stickiness score of 1. This is sufficient to prevent healthy resources from moving around the cluster when there are no user-configured constraints that influence where Pacemaker prefers to run those resources.

```
[root@pcmk-1 ~]# pcs resource defaults
Meta Attrs: build-resource-defaults
  resource-stickiness=1
```

For this example, we will increase the default resource stickiness to 100. Later in this guide, we will configure a location constraint with a score lower than the default resource stickiness.

```
[root@pcmk-1 ~]# pcs resource defaults update resource-stickiness=100
Warning: Defaults do not apply to resources which override them with their own defined values
[root@pcmk-1 ~]# pcs resource defaults
Meta Attrs: build-resource-defaults
resource-stickiness=100
```

2.7 Add Apache HTTP Server as a Cluster Service

Now that we have a basic but functional active/passive two-node cluster, we're ready to add some real services. We're going to start with Apache HTTP Server because it is a feature of many clusters and is relatively simple to configure.

2.7.1 Install Apache

Before continuing, we need to make sure Apache is installed on both hosts. We will also allow the cluster to use the `wget` tool (this is the default, but `curl` is also supported) to check the status of the Apache server. We'll install `httpd` (Apache) and `wget` now.

```
# dnf install -y httpd wget
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload
```

Important: Do **not** enable the `httpd` service. Services that are intended to be managed via the cluster software should never be managed by the OS. It is often useful, however, to manually start the service, verify that it works, then stop it again, before adding it to the cluster. This allows you to resolve any non-cluster-related problems before continuing. Since this is a simple example, we'll skip that step here.

2.7.2 Create Website Documents

We need to create a page for Apache to serve. On AlmaLinux 9, the default Apache document root is `/var/www/html`, so we'll create an index file there. For the moment, we will simplify things by serving a static site and manually synchronizing the data between the two nodes, so run this command on both nodes:

you're using an older version of Pacemaker that doesn't create this value at setup time.

[2] Pacemaker's default definition of "optimal" may not always agree with yours. The order in which Pacemaker processes lists of resources and nodes creates implicit preferences in situations where the administrator has not explicitly specified them.

```
# cat <<-END >/var/www/html/index.html
 <html>
 <body>My Test Site - $(hostname)</body>
 </html>
END
```

2.7.3 Enable the Apache Status URL

Pacemaker uses the **apache** resource agent to monitor the health of your Apache instance via the **server-status** URL, and to recover the instance if it fails. On both nodes, configure this URL as follows:

```
# cat <<-END >/etc/httpd/conf.d/status.conf
 <Location /server-status>
    SetHandler server-status
    Require local
 </Location>
END
```

Note: If you are using a different operating system, **server-status** may already be enabled or may be configurable in a different location. If you are using a version of Apache HTTP Server less than 2.4, the syntax will be different.

2.7.4 Configure the Cluster

At this point, Apache is ready to go, and all that needs to be done is to add it to the cluster. Let's call the resource `WebSite`. We need to use an OCF resource agent called **apache** in the **heartbeat** namespace[1]. The script's only required parameter is the path to the main Apache configuration file, and we'll tell the cluster to check once a minute that Apache is still running.

```
[root@pcmk-1 ~]# pcs resource create WebSite ocf:heartbeat:apache  \
      configfile=/etc/httpd/conf/httpd.conf \
      statusurl="http://localhost/server-status" \
      op monitor interval=1min
```

By default, the operation timeout for all resources' start, stop, monitor, and other operations is 20 seconds. In many cases, this timeout period is less than a particular resource's advised timeout period. For the purposes of this tutorial, we will adjust the global operation timeout default to 240 seconds.

```
[root@pcmk-1 ~]# pcs resource op defaults
No defaults set
[root@pcmk-1 ~]# pcs resource op defaults update timeout=240s
Warning: Defaults do not apply to resources which override them with their own defined values
[root@pcmk-1 ~]# pcs resource op defaults
Meta Attrs: op_defaults-meta_attributes
timeout: 240s
```

Note: In a production cluster, it is usually better to adjust each resource's start, stop, and monitor timeouts to values that are appropriate for the behavior observed in your environment, rather than adjusting the global

[1] Compare the key used here, `ocf:heartbeat:apache` with the one we used earlier for the IP address, `ocf:heartbeat:IPaddr2`.

default.

Note: If you use a tool like `pcs` to create a resource, its operations may be automatically configured with explicit timeout values that override the Pacemaker built-in default value of 20 seconds. If the resource agent's metadata contains suggested values for the operation timeouts in a particular format, `pcs` reads those values and adds them to the configuration at resource creation time.

After a short delay, we should see the cluster start Apache.

```
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:47:44 2022
  * Last change:  Wed Jul 27 00:47:23 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 3 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-2

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

Wait a moment, the `WebSite` resource isn't running on the same host as our IP address!

Note: If, in the `pcs status` output, you see the `WebSite` resource has failed to start, then you've likely not enabled the status URL correctly. You can check whether this is the problem by running:

```
wget -O - http://localhost/server-status
```

If you see `Not Found` or `Forbidden` in the output, then this is likely the problem. Ensure that the `<Location /server-status>` block is correct.

2.7.5 Ensure Resources Run on the Same Host

To reduce the load on any one machine, Pacemaker will generally try to spread the configured resources across the cluster nodes. However, we can tell the cluster that two resources are related and need to run on the same host (or else one of them should not run at all, if they cannot run on the same node). Here, we instruct the cluster that `WebSite` can only run on the host where `ClusterIP` is active.

To achieve this, we use a *colocation constraint* that indicates it is mandatory for `WebSite` to run on the same node as `ClusterIP`. The "mandatory" part of the colocation constraint is indicated by using a score of `INFINITY`. The `INFINITY` score also means that if `ClusterIP` is not active anywhere, `WebSite` will not be permitted to run.

Note: If `ClusterIP` is not active anywhere, `WebSite` will not be permitted to run anywhere.

Note: `INFINITY` is the default score for a colocation constraint. If you don't specify a score, `INFINITY` will be used automatically.

Important: Colocation constraints are "directional", in that they imply certain things about the order in which the two resources will have a location chosen. In this case, we're saying that `WebSite` needs to be placed on the same machine as `ClusterIP`, which implies that the cluster must know the location of `ClusterIP` before choosing a location for `WebSite`

```
[root@pcmk-1 ~]# pcs constraint colocation add WebSite with ClusterIP INFINITY
[root@pcmk-1 ~]# pcs constraint
Location Constraints:
Ordering Constraints:
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
Ticket Constraints:
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:49:33 2022
  * Last change:  Wed Jul 27 00:49:16 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 3 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1

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

2.7.6 Ensure Resources Start and Stop in Order

Like many services, Apache can be configured to bind to specific IP addresses on a host or to the wildcard IP address. If Apache binds to the wildcard, it doesn't matter whether an IP address is added before or after Apache starts; Apache will respond on that IP just the same. However, if Apache binds only to certain IP address(es), the order matters: If the address is added after Apache starts, Apache won't respond on that address.

To be sure our `WebSite` responds regardless of Apache's address configuration, we need to make sure `ClusterIP` not only runs on the same node, but also starts before `WebSite`. A colocation constraint ensures only that the resources run together; it doesn't affect order in which the resources are started or stopped.

We do this by adding an ordering constraint. By default, all order constraints are mandatory. This means, for example, that if `ClusterIP` needs to stop, then `WebSite` must stop first (or already be stopped); and if WebSite needs to start, then `ClusterIP` must start first (or already be started). This also implies that the recovery of `ClusterIP` will trigger the recovery of `WebSite`, causing it to be restarted.

```
[root@pcmk-1 ~]# pcs constraint order ClusterIP then WebSite
Adding ClusterIP WebSite (kind: Mandatory) (Options: first-action=start then-action=start)
[root@pcmk-1 ~]# pcs constraint
Location Constraints:
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
Ticket Constraints:
```

Note: The default action in an order constraint is `start` If you don't specify an action, as in the example above, `pcs` automatically uses the `start` action.

Note: We could have placed the `ClusterIP` and `WebSite` resources into a **resource group** instead of configuring constraints. A resource group is a compact and intuitive way to organize a set of resources into a chain of colocation and ordering constraints. We will omit that in this guide; see the Pacemaker Explained document for more details.

2.7.7 Prefer One Node Over Another

Pacemaker does not rely on any sort of hardware symmetry between nodes, so it may well be that one machine is more powerful than the other.

In such cases, you may want to host the resources on the more powerful node when it is available, to have the best performance – or you may want to host the resources on the **less** powerful node when it's available, so you don't have to worry about whether you can handle the load after a failover.

To do this, we create a location constraint.

In the location constraint below, we are saying the `WebSite` resource prefers the node `pcmk-2` with a score of `50`. Here, the score indicates how strongly we'd like the resource to run at this location.

```
[root@pcmk-1 ~]# pcs constraint location WebSite prefers pcmk-2=50
[root@pcmk-1 ~]# pcs constraint
Location Constraints:
  Resource: WebSite
    Enabled on:
      Node: pcmk-2 (score:50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
Ticket Constraints:
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
```

(continues on next page)

(continued from previous page)

```
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:51:13 2022
  * Last change:  Wed Jul 27 00:51:07 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 3 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1

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

Wait a minute, the resources are still on `pcmk-1`!

Even though `WebSite` now prefers to run on `pcmk-2`, that preference is (intentionally) less than the resource stickiness (how much we preferred not to have unnecessary downtime).

To see the current placement scores, you can use a tool called `crm_simulate`.

```
[root@pcmk-1 ~]# crm_simulate -sL
[ pcmk-1 pcmk-2 ]

fence_dev   (stonith:some_fence_agent):      Started pcmk-1
ClusterIP   (ocf:heartbeat:IPaddr2):         Started pcmk-1
WebSite     (ocf:heartbeat:apache):  Started pcmk-1

pcmk__native_allocate: fence_dev allocation score on pcmk-1: 100
pcmk__native_allocate: fence_dev allocation score on pcmk-2: 0
pcmk__native_allocate: ClusterIP allocation score on pcmk-1: 200
pcmk__native_allocate: ClusterIP allocation score on pcmk-2: 50
pcmk__native_allocate: WebSite allocation score on pcmk-1: 100
pcmk__native_allocate: WebSite allocation score on pcmk-2: -INFINITY
```

2.7.8 Move Resources Manually

There are always times when an administrator needs to override the cluster and force resources to move to a specific location. In this example, we will force the WebSite to move to `pcmk-2`.

We will use the `pcs resource move` command to create a temporary constraint with a score of `INFINITY`. While we could update our existing constraint, using `move` allows `pcs` to get rid of the temporary constraint automatically after the resource has moved to its destination. Note in the below that the `pcs constraint` output after the `move` command is the same as before.

```
[root@pcmk-1 ~]# pcs resource move WebSite pcmk-2
Location constraint to move resource 'WebSite' has been created
Waiting for the cluster to apply configuration changes...
Location constraint created to move resource 'WebSite' has been removed
```

(continues on next page)

(continued from previous page)

```
Waiting for the cluster to apply configuration changes...
resource 'WebSite' is running on node 'pcmk-2'
[root@pcmk-1 ~]# pcs constraint
Location Constraints:
  Resource: WebSite
    Enabled on:
      Node: pcmk-2 (score:50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
Ticket Constraints:
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 00:54:23 2022
  * Last change:  Wed Jul 27 00:53:48 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 3 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-2
  * WebSite (ocf:heartbeat:apache):  Started pcmk-2

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

To remove the constraint with the score of 50, we would first get the constraint's ID using `pcs constraint --full`, then remove it with `pcs constraint remove` and the ID. We won't show those steps here, but feel free to try it on your own, with the help of the `pcs` man page if necessary.

2.8 Replicate Storage Using DRBD

Even if you're serving up static websites, having to manually synchronize the contents of that website to all the machines in the cluster is not ideal. For dynamic websites, such as a wiki, it's not even an option. Not everyone can afford network-attached storage, but somehow the data needs to be kept in sync.

Enter DRBD, which can be thought of as network-based RAID-1[1].

2.8.1 Install the DRBD Packages

DRBD itself is included in the upstream kernel[2], but we do need some utilities to use it effectively.

[1] See http://www.drbd.org for details.
[2] Since version 2.6.33

AlmaLinux does not ship these utilities, so we need to enable a third-party repository to get them. Supported packages for many OSes are available from DRBD's maker LINBIT, but here we'll use the free ELRepo repository.

On both nodes, import the ELRepo package signing key, and enable the repository:

```
[root@pcmk-1 ~]# rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
[root@pcmk-1 ~]# dnf install -y https://www.elrepo.org/elrepo-release-9.el9.elrepo.noarch.rpm
```

Now, we can install the DRBD kernel module and utilities:

```
# dnf install -y kmod-drbd9x drbd9x-utils
```

DRBD will not be able to run under the default SELinux security policies. If you are familiar with SELinux, you can modify the policies in a more fine-grained manner, but here we will simply exempt DRBD processes from SELinux control:

```
# dnf install -y policycoreutils-python-utils
# semanage permissive -a drbd_t
```

We will configure DRBD to use port 7789, so allow that port from each host to the other:

```
[root@pcmk-1 ~]# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" \
source address="192.168.122.102" port port="7789" protocol="tcp" accept'
success
[root@pcmk-1 ~]# firewall-cmd --reload
success
```

```
[root@pcmk-2 ~]# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" \
source address="192.168.122.101" port port="7789" protocol="tcp" accept'
success
[root@pcmk-2 ~]# firewall-cmd --reload
success
```

Note: In this example, we have only two nodes, and all network traffic is on the same LAN. In production, it is recommended to use a dedicated, isolated network for cluster-related traffic, so the firewall configuration would likely be different; one approach would be to add the dedicated network interfaces to the trusted zone.

Note: If the `firewall-cmd --add-rich-rule` command fails with `Error: INVALID_RULE: unknown element` ensure that there is no space at the beginning of the second line of the command.

2.8.2 Allocate a Disk Volume for DRBD

DRBD will need its own block device on each node. This can be a physical disk partition or logical volume, of whatever size you need for your data. For this document, we will use a 512MiB logical volume, which is more than sufficient for a single HTML file and (later) GFS2 metadata.

```
[root@pcmk-1 ~]# vgs
  VG               #PV #LV #SN Attr   VSize   VFree
  almalinux_pcmk-1   1   2   0 wz--n- <19.00g <13.00g

[root@pcmk-1 ~]# lvcreate --name drbd-demo --size 512M almalinux_pcmk-1
```

(continues on next page)

(continued from previous page)

```
  Logical volume "drbd-demo" created.
[root@pcmk-1 ~]# lvs
  LV        VG              Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  drbd-demo almalinux_pcmk-1 -wi-a----- 512.00m
  root      almalinux_pcmk-1 -wi-ao----   4.00g
  swap      almalinux_pcmk-1 -wi-ao----   2.00g
```

Repeat for the second node, making sure to use the same size:

```
[root@pcmk-1 ~]# ssh pcmk-2 -- lvcreate --name drbd-demo --size 512M cs_pcmk-2
 Logical volume "drbd-demo" created.
```

2.8.3 Configure DRBD

There is no series of commands for building a DRBD configuration, so simply run this on both nodes to use this sample configuration:

```
# cat <<END >/etc/drbd.d/wwwdata.res
resource "wwwdata" {
  device minor 1;
  meta-disk internal;

  net {
    protocol C;
    allow-two-primaries yes;
    fencing resource-and-stonith;
    verify-alg sha1;
  }
  handlers {
    fence-peer "/usr/lib/drbd/crm-fence-peer.9.sh";
    unfence-peer "/usr/lib/drbd/crm-unfence-peer.9.sh";
  }
  on "pcmk-1" {
    disk "/dev/almalinux_pcmk-1/drbd-demo";
    node-id 0;
  }
  on "pcmk-2" {
    disk "/dev/almalinux_pcmk-2/drbd-demo";
    node-id 1;
  }
  connection {
    host "pcmk-1" address 192.168.122.101:7789;
    host "pcmk-2" address 192.168.122.102:7789;
  }
}
END
```

Important: Edit the file to use the hostnames, IP addresses, and logical volume paths of your nodes if they differ from the ones used in this guide.

Note: Detailed information on the directives used in this configuration (and other alternatives) is available in the DRBD User's Guide. The guide contains a wealth of information on such topics as core DRBD concepts, replication settings, network connection options, quorum, split- brain handling, administrative tasks, troubleshooting, and responding to disk or node failures, among others.

The `allow-two-primaries: yes` option would not normally be used in an active/passive cluster. We are adding it here for the convenience of changing to an active/active cluster later.

2.8.4 Initialize DRBD

With the configuration in place, we can now get DRBD running.

These commands create the local metadata for the DRBD resource, ensure the DRBD kernel module is loaded, and bring up the DRBD resource. Run them on one node:

```
[root@pcmk-1 ~]# drbdadm create-md wwwdata
initializing activity log
initializing bitmap (16 KB) to all zero
Writing meta data...
New drbd meta data block successfully created.
success

[root@pcmk-1 ~]# modprobe drbd
[root@pcmk-1 ~]# drbdadm up wwwdata
















  --==  Thank you for participating in the global usage survey  ==--
The server's response is:

you are the 25212th user to install this version
```

We can confirm DRBD's status on this node:

```
[root@pcmk-1 ~]# drbdadm status
wwwdata role:Secondary
  disk:Inconsistent
  pcmk-2 connection:Connecting
```

Because we have not yet initialized the data, this node's data is marked as `Inconsistent` Because we have not yet initialized the second node, the `pcmk-2` connection is `Connecting` (waiting for connection).

Now, repeat the above commands on the second node, starting with creating `wwwdata.res`. After giving it time to connect, when we check the status of the first node, it shows:

```
[root@pcmk-1 ~]# drbdadm status
wwwdata role:Secondary
  disk:Inconsistent
  pcmk-2 role:Secondary
    peer-disk:Inconsistent
```

You can see that `pcmk-2 connection:Connecting` longer appears in the output, meaning the two DRBD nodes are communicating properly, and both nodes are in `Secondary` role with `Inconsistent` data.

To make the data consistent, we need to tell DRBD which node should be considered to have the correct data. In this case, since we are creating a new resource, both have garbage, so we'll just pick `pcmk-1` and run this command on it:

```
[root@pcmk-1 ~]# drbdadm primary --force wwwdata
```

Note: If you are using a different version of DRBD, the required syntax may be different. See the documentation for your version for how to perform these commands.

If we check the status immediately, we'll see something like this:

```
[root@pcmk-1 ~]# drbdadm status
wwwdata role:Primary
  disk:UpToDate
  pcmk-2 role:Secondary
    peer-disk:Inconsistent
```

It will be quickly followed by this:

```
[root@pcmk-1 ~]# drbdadm status
wwwdata role:Primary
  disk:UpToDate
  pcmk-2 role:Secondary
    replication:SyncSource peer-disk:Inconsistent
```

We can see that the first node has the `Primary` role, its partner node has the `Secondary` role, the first node's data is now considered `UpToDate`, and the partner node's data is still `Inconsistent`.

After a while, the sync should finish, and you'll see something like:

```
[root@pcmk-1 ~]# drbdadm status
wwwdata role:Primary
  disk:UpToDate
  pcmk-1 role:Secondary
    peer-disk:UpToDate
[root@pcmk-2 ~]# drbdadm status
wwwdata role:Secondary
  disk:UpToDate
  pcmk-1 role:Primary
    peer-disk:UpToDate
```

Both sets of data are now `UpToDate`, and we can proceed to creating and populating a filesystem for our `WebSite` resource's documents.

2.8.5 Populate the DRBD Disk

On the node with the primary role (pcmk-1 in this example), create a filesystem on the DRBD device:

```
[root@pcmk-1 ~]# mkfs.xfs /dev/drbd1
meta-data=/dev/drbd1             isize=512    agcount=4, agsize=32765 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=131059, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
Discarding blocks...Done.
```

Note: In this example, we create an xfs filesystem with no special options. In a production environment, you should choose a filesystem type and options that are suitable for your application.

Mount the newly created filesystem, populate it with our web document, give it the same SELinux policy as the web document root, then unmount it (the cluster will handle mounting and unmounting it later):

```
[root@pcmk-1 ~]# mount /dev/drbd1 /mnt
[root@pcmk-1 ~]# cat <<-END >/mnt/index.html
 <html>
  <body>My Test Site - DRBD</body>
 </html>
END
[root@pcmk-1 ~]# chcon -R --reference=/var/www/html /mnt
[root@pcmk-1 ~]# umount /dev/drbd1
```

2.8.6 Configure the Cluster for the DRBD device

One handy feature pcs has is the ability to queue up several changes into a file and commit those changes all at once. To do this, start by populating the file with the current raw XML config from the CIB.

```
[root@pcmk-1 ~]# pcs cluster cib drbd_cfg
```

Using pcs's -f option, make changes to the configuration saved in the drbd_cfg file. These changes will not be seen by the cluster until the drbd_cfg file is pushed into the live cluster's CIB later.

Here, we create a cluster resource for the DRBD device, and an additional *clone* resource to allow the resource to run on both nodes at the same time.

```
[root@pcmk-1 ~]# pcs -f drbd_cfg resource create WebData ocf:linbit:drbd \
     drbd_resource=wwwdata op monitor interval=29s role=Promoted \
     monitor interval=31s role=Unpromoted
[root@pcmk-1 ~]# pcs -f drbd_cfg resource promotable WebData \
     promoted-max=1 promoted-node-max=1 clone-max=2 clone-node-max=1 \
     notify=true
[root@pcmk-1 ~]# pcs resource status
 * ClusterIP        (ocf::heartbeat:IPaddr2):       Started pcmk-1
 * WebSite  (ocf::heartbeat:apache):                Started pcmk-1
```

(continues on next page)

(continued from previous page)

```
[root@pcmk-1 ~]# pcs resource config
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: cidr_netmask=24 ip=192.168.122.120
  Operations: monitor interval=30s (ClusterIP-monitor-interval-30s)
              start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
 Resource: WebSite (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
  Operations: monitor interval=1min (WebSite-monitor-interval-1min)
              start interval=0s timeout=40s (WebSite-start-interval-0s)
              stop interval=0s timeout=60s (WebSite-stop-interval-0s)
```

After you are satisfied with all the changes, you can commit them all at once by pushing the `drbd_cfg` file into the live CIB.

```
[root@pcmk-1 ~]# pcs cluster cib-push drbd_cfg --config
CIB updated
```

Note: All the updates above can be done in one shot as follows:

```
[root@pcmk-1 ~]# pcs resource create WebData ocf:linbit:drbd \
    drbd_resource=wwwdata op monitor interval=29s role=Promoted \
    monitor interval=31s role=Unpromoted \
    promotable promoted-max=1 promoted-node-max=1 clone-max=2  \
    clone-node-max=1 notify=true
```

Let's see what the cluster did with the new configuration:

```
[root@pcmk-1 ~]# pcs resource status
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-2
  * WebSite (ocf:heartbeat:apache):  Started pcmk-2
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 ]
    * Unpromoted: [ pcmk-2 ]
[root@pcmk-1 ~]# pcs resource config
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: cidr_netmask=24 ip=192.168.122.120
  Operations: monitor interval=30s (ClusterIP-monitor-interval-30s)
              start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
 Resource: WebSite (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
  Operations: monitor interval=1min (WebSite-monitor-interval-1min)
              start interval=0s timeout=40s (WebSite-start-interval-0s)
              stop interval=0s timeout=60s (WebSite-stop-interval-0s)
 Clone: WebData-clone
  Meta Attrs: clone-max=2 clone-node-max=1 notify=true promotable=true promoted-max=1 promoted-
↪node-max=1
  Resource: WebData (class=ocf provider=linbit type=drbd)
   Attributes: drbd_resource=wwwdata
   Operations: demote interval=0s timeout=90 (WebData-demote-interval-0s)
               monitor interval=29s role=Promoted (WebData-monitor-interval-29s)
               monitor interval=31s role=Unpromoted (WebData-monitor-interval-31s)
               notify interval=0s timeout=90 (WebData-notify-interval-0s)
```

(continues on next page)

(continued from previous page)

```
                promote interval=0s timeout=90 (WebData-promote-interval-0s)
                reload interval=0s timeout=30 (WebData-reload-interval-0s)
                start interval=0s timeout=240 (WebData-start-interval-0s)
                stop interval=0s timeout=100 (WebData-stop-interval-0s)
```

We can see that `WebData-clone` (our DRBD device) is running as `Promoted` (DRBD's primary role) on `pcmk-1` and `Unpromoted` (DRBD's secondary role) on `pcmk-2`.

Important: The resource agent should load the DRBD module when needed if it's not already loaded. If that does not happen, configure your operating system to load the module at boot time. For AlmaLinux 9, you would run this on both nodes:

```
# echo drbd >/etc/modules-load.d/drbd.conf
```

2.8.7 Configure the Cluster for the Filesystem

Now that we have a working DRBD device, we need to mount its filesystem.

In addition to defining the filesystem, we also need to tell the cluster where it can be located (only on the DRBD Primary) and when it is allowed to start (after the Primary was promoted).

We are going to take a shortcut when creating the resource this time. Instead of explicitly saying we want the `ocf:heartbeat:Filesystem` script, we are only going to ask for `Filesystem`. We can do this because we know there is only one resource script named `Filesystem` available to Pacemaker, and that `pcs` is smart enough to fill in the `ocf:heartbeat:` portion for us correctly in the configuration. If there were multiple `Filesystem` scripts from different OCF providers, we would need to specify the exact one we wanted.

Once again, we will queue our changes to a file and then push the new configuration to the cluster as the final step.

```
[root@pcmk-1 ~]# pcs cluster cib fs_cfg
[root@pcmk-1 ~]# pcs -f fs_cfg resource create WebFS Filesystem \
    device="/dev/drbd1" directory="/var/www/html" fstype="xfs"
Assumed agent name 'ocf:heartbeat:Filesystem' (deduced from 'Filesystem')
[root@pcmk-1 ~]# pcs -f fs_cfg constraint colocation add \
    WebFS with Promoted WebData-clone
[root@pcmk-1 ~]# pcs -f fs_cfg constraint order \
    promote WebData-clone then start WebFS
Adding WebData-clone WebFS (kind: Mandatory) (Options: first-action=promote then-action=start)
```

We also need to tell the cluster that Apache needs to run on the same machine as the filesystem and that it must be active before Apache can start.

```
[root@pcmk-1 ~]# pcs -f fs_cfg constraint colocation add WebSite with WebFS
[root@pcmk-1 ~]# pcs -f fs_cfg constraint order WebFS then WebSite
Adding WebFS WebSite (kind: Mandatory) (Options: first-action=start then-action=start)
```

Review the updated configuration.

```
[root@pcmk-1 ~]# pcs -f fs_cfg constraint
Location Constraints:
  Resource: WebSite
    Enabled on:
```

(continues on next page)

(continued from previous page)

```
      Node: pcmk-1 (score:50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
  promote WebData-clone then start WebFS (kind:Mandatory)
  start WebFS then start WebSite (kind:Mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
  WebFS with WebData-clone (score:INFINITY) (rsc-role:Started) (with-rsc-role:Promoted)
  WebSite with WebFS (score:INFINITY)
Ticket Constraints:
```

After reviewing the new configuration, upload it and watch the cluster put it into effect.

```
[root@pcmk-1 ~]# pcs cluster cib-push fs_cfg --config
CIB updated
[root@pcmk-1 ~]# pcs resource status
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-2
  * WebSite (ocf:heartbeat:apache):  Started pcmk-2
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-2 ]
    * Unpromoted: [ pcmk-1 ]
  * WebFS   (ocf:heartbeat:Filesystem):      Started pcmk-2
[root@pcmk-1 ~]# pcs resource config
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: cidr_netmask=24 ip=192.168.122.120
  Operations: monitor interval=30s (ClusterIP-monitor-interval-30s)
              start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
 Resource: WebSite (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
  Operations: monitor interval=1min (WebSite-monitor-interval-1min)
              start interval=0s timeout=40s (WebSite-start-interval-0s)
              stop interval=0s timeout=60s (WebSite-stop-interval-0s)
 Clone: WebData-clone
  Meta Attrs: clone-max=2 clone-node-max=1 notify=true promotable=true promoted-max=1 promoted-
↪node-max=1
  Resource: WebData (class=ocf provider=linbit type=drbd)
   Attributes: drbd_resource=wwwdata
   Operations: demote interval=0s timeout=90 (WebData-demote-interval-0s)
               monitor interval=29s role=Promoted (WebData-monitor-interval-29s)
               monitor interval=31s role=Unpromoted (WebData-monitor-interval-31s)
               notify interval=0s timeout=90 (WebData-notify-interval-0s)
               promote interval=0s timeout=90 (WebData-promote-interval-0s)
               reload interval=0s timeout=30 (WebData-reload-interval-0s)
               start interval=0s timeout=240 (WebData-start-interval-0s)
               stop interval=0s timeout=100 (WebData-stop-interval-0s)
 Resource: WebFS (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/drbd1 directory=/var/www/html fstype=xfs
  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)
```

2.8.8 Test Cluster Failover

Previously, we used `pcs cluster stop pcmk-2` to stop all cluster services on `pcmk-2`, failing over the cluster resources, but there is another way to safely simulate node failure.

We can put the node into *standby mode*. Nodes in this state continue to run `corosync` and `pacemaker` but are not allowed to run resources. Any resources found active there will be moved elsewhere. This feature can be particularly useful when performing system administration tasks such as updating packages used by cluster resources.

Put the active node into standby mode, and observe the cluster move all the resources to the other node. The node's status will change to indicate that it can no longer host resources, and eventually all the resources will move.

```
[root@pcmk-1 ~]# pcs node standby pcmk-2
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 05:28:01 2022
  * Last change:  Wed Jul 27 05:27:57 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 6 resource instances configured

Node List:
  * Node pcmk-2: standby
  * Online: [ pcmk-1 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 ]
    * Stopped: [ pcmk-2 ]
  * WebFS   (ocf:heartbeat:Filesystem):      Started pcmk-1

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

Once we've done everything we needed to on `pcmk-2` (in this case nothing, we just wanted to see the resources move), we can unstandby the node, making it eligible to host resources again.

```
[root@pcmk-1 ~]# pcs node unstandby pcmk-2
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 05:28:50 2022
  * Last change:  Wed Jul 27 05:28:47 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 6 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
```

(continues on next page)

(continued from previous page)

```
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 ]
    * Unpromoted: [ pcmk-2 ]
  * WebFS   (ocf:heartbeat:Filesystem):      Started pcmk-1

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

Notice that `pcmk-2` is back to the `Online` state, and that the cluster resources stay where they are due to our resource stickiness settings configured earlier.

2.9 Convert Storage to Active/Active

The primary requirement for an active/active cluster is that the data required for your services is available, simultaneously, on both machines. Pacemaker makes no requirement on how this is achieved; you could use a Storage Area Network (SAN) if you had one available, but since DRBD supports multiple Primaries, we can continue to use it here.

2.9.1 Install Cluster Filesystem Software

The only hitch is that we need to use a cluster-aware filesystem. The one we used earlier with DRBD, xfs, is not one of those. Both OCFS2 and GFS2 are supported; here, we will use GFS2.

On both nodes, install Distributed Lock Manager (DLM) and the GFS2 command- line utilities required by cluster filesystems:

```
# dnf config-manager --set-enabled resilientstorage
# dnf install -y dlm gfs2-utils
```

2.9.2 Configure the Cluster for the DLM

The DLM control daemon needs to run on both nodes, so we'll start by creating a resource for it (using the `ocf:pacemaker:controld` resource agent), and clone it:

```
[root@pcmk-1 ~]# pcs cluster cib dlm_cfg
[root@pcmk-1 ~]# pcs -f dlm_cfg resource create dlm \
    ocf:pacemaker:controld op monitor interval=60s
[root@pcmk-1 ~]# pcs -f dlm_cfg resource clone dlm clone-max=2 clone-node-max=1
[root@pcmk-1 ~]# pcs resource status
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 ]
    * Unpromoted: [ pcmk-2 ]
  * WebFS   (ocf:heartbeat:Filesystem):      Started pcmk-1
```

Activate our new configuration, and see how the cluster responds:

```
[root@pcmk-1 ~]# pcs cluster cib-push dlm_cfg --config
CIB updated
[root@pcmk-1 ~]# pcs resource status
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 ]
    * Unpromoted: [ pcmk-2 ]
  * WebFS   (ocf:heartbeat:Filesystem):      Started pcmk-1
  * Clone Set: dlm-clone [dlm]:
    * Started: [ pcmk-1 pcmk-2 ]
[root@pcmk-1 ~]# pcs resource config
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: cidr_netmask=24 ip=192.168.122.120
  Operations: monitor interval=30s (ClusterIP-monitor-interval-30s)
              start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
 Resource: WebSite (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
  Operations: monitor interval=1min (WebSite-monitor-interval-1min)
              start interval=0s timeout=40s (WebSite-start-interval-0s)
              stop interval=0s timeout=60s (WebSite-stop-interval-0s)
 Clone: WebData-clone
  Meta Attrs: clone-max=2 clone-node-max=1 notify=true promotable=true promoted-max=1 promoted-
↪node-max=1
  Resource: WebData (class=ocf provider=linbit type=drbd)
   Attributes: drbd_resource=wwwdata
   Operations: demote interval=0s timeout=90 (WebData-demote-interval-0s)
               monitor interval=29s role=Promoted (WebData-monitor-interval-29s)
               monitor interval=31s role=Unpromoted (WebData-monitor-interval-31s)
               notify interval=0s timeout=90 (WebData-notify-interval-0s)
               promote interval=0s timeout=90 (WebData-promote-interval-0s)
               reload interval=0s timeout=30 (WebData-reload-interval-0s)
               start interval=0s timeout=240 (WebData-start-interval-0s)
               stop interval=0s timeout=100 (WebData-stop-interval-0s)
 Resource: WebFS (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/drbd1 directory=/var/www/html fstype=xfs
  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)
 Clone: dlm-clone
  Meta Attrs: interleave=true ordered=true
  Resource: dlm (class=ocf provider=pacemaker type=controld)
   Operations: monitor interval=60s (dlm-monitor-interval-60s)
               start interval=0s timeout=90s (dlm-start-interval-0s)
               stop interval=0s timeout=100s (dlm-stop-interval-0s)
```

2.9.3 Create and Populate GFS2 Filesystem

Before we do anything to the existing partition, we need to make sure it is unmounted. We do this by telling the cluster to stop the `WebFS` resource. This will ensure that other resources (in our case, `WebSite`) using `WebFS` are not only stopped, but stopped in the correct order.

```
[root@pcmk-1 ~]# pcs resource disable WebFS
[root@pcmk-1 ~]# pcs resource
```

(continues on next page)

(continued from previous page)

```
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Stopped
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 ]
    * Unpromoted: [ pcmk-2 ]
  * WebFS   (ocf:heartbeat:Filesystem):      Stopped (disabled)
  * Clone Set: dlm-clone [dlm]:
    * Started: [ pcmk-1 pcmk-2 ]
```

You can see that both `WebSite` and `WebFS` have been stopped, and that `pcmk-1` is currently running the promoted instance for the DRBD device.

Now we can create a new GFS2 filesystem on the DRBD device.

Warning: This will erase all previous content stored on the DRBD device. Ensure you have a copy of any important data.

Important: Run the next command on whichever node has the DRBD Primary role. Otherwise, you will receive the message:

```
/dev/drbd1: Read-only file system
```

```
[root@pcmk-1 ~]# mkfs.gfs2 -p lock_dlm -j 2 -t mycluster:web /dev/drbd1
It appears to contain an existing filesystem (xfs)
This will destroy any data on /dev/drbd1
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/drbd1
Block size:                4096
Device size:               0.50 GB (131059 blocks)
Filesystem size:           0.50 GB (131055 blocks)
Journals:                  2
Journal size:              8MB
Resource groups:           4
Locking protocol:          "lock_dlm"
Lock table:                "mycluster:web"
UUID:                      19712677-7206-4660-a079-5d17341dd720
```

The `mkfs.gfs2` command required a number of additional parameters:

- `-p lock_dlm` specifies that we want to use DLM-based locking.
- `-j 2` indicates that the filesystem should reserve enough space for two journals (one for each node that will access the filesystem).
- `-t mycluster:web` specifies the lock table name. The format for this field is `<CLUSTERNAME>:<FSNAME>`. For `CLUSTERNAME`, we need to use the same value we specified originally with `pcs cluster setup --name` (which is also the value of `cluster_name` in `/etc/corosync/corosync.conf`). If you are unsure what your cluster name is, you can look in `/etc/corosync/corosync.conf` or execute the command `pcs cluster corosync | grep cluster_name`.

Now we can (re-)populate the new filesystem with data (web pages). We'll create yet another variation on our home page.

```
[root@pcmk-1 ~]# mount /dev/drbd1 /mnt
[root@pcmk-1 ~]# cat <<-END >/mnt/index.html
<html>
<body>My Test Site - GFS2</body>
</html>
END
[root@pcmk-1 ~]# chcon -R --reference=/var/www/html /mnt
[root@pcmk-1 ~]# umount /dev/drbd1
[root@pcmk-1 ~]# drbdadm verify wwwdata
```

2.9.4 Reconfigure the Cluster for GFS2

With the `WebFS` resource stopped, let's update the configuration.

```
[root@pcmk-1 ~]# pcs resource config WebFS
 Resource: WebFS (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/drbd1 directory=/var/www/html fstype=xfs
  Meta Attrs: target-role=Stopped
  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)
```

The fstype option needs to be updated to `gfs2` instead of `xfs`.

```
[root@pcmk-1 ~]# pcs resource update WebFS fstype=gfs2
[root@pcmk-1 ~]# pcs resource config WebFS
 Resource: WebFS (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/drbd1 directory=/var/www/html fstype=gfs2
  Meta Attrs: target-role=Stopped
  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)
```

GFS2 requires that DLM be running, so we also need to set up new colocation and ordering constraints for it:

```
[root@pcmk-1 ~]# pcs constraint colocation add WebFS with dlm-clone
[root@pcmk-1 ~]# pcs constraint order dlm-clone then WebFS
Adding dlm-clone WebFS (kind: Mandatory) (Options: first-action=start then-action=start)
[root@pcmk-1 ~]# pcs constraint
Location Constraints:
  Resource: WebSite
    Enabled on:
      Node: pcmk-2 (score:50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
  promote WebData-clone then start WebFS (kind:Mandatory)
  start WebFS then start WebSite (kind:Mandatory)
  start dlm-clone then start WebFS (kind:Mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
  WebFS with WebData-clone (score:INFINITY) (rsc-role:Started) (with-rsc-role:Promoted)
  WebSite with WebFS (score:INFINITY)
```

(continues on next page)

(continued from previous page)

```
  WebFS with dlm-clone (score:INFINITY)
Ticket Constraints:
```

We also need to update the `no-quorum-policy` property to `freeze`. By default, the value of `no-quorum-policy` is set to `stop` indicating that once quorum is lost, all the resources on the remaining partition will immediately be stopped. Typically this default is the safest and most optimal option, but unlike most resources, GFS2 requires quorum to function. When quorum is lost both the applications using the GFS2 mounts and the GFS2 mount itself cannot be correctly stopped. Any attempts to stop these resources without quorum will fail, which will ultimately result in the entire cluster being fenced every time quorum is lost.

To address this situation, set `no-quorum-policy` to `freeze` when GFS2 is in use. This means that when quorum is lost, the remaining partition will do nothing until quorum is regained.

```
[root@pcmk-1 ~]# pcs property set no-quorum-policy=freeze
```

2.9.5 Clone the Filesystem Resource

Now that we have a cluster filesystem ready to go, we can configure the cluster so both nodes mount the filesystem.

Clone the `Filesystem` resource in a new configuration. Notice how `pcs` automatically updates the relevant constraints again.

```
[root@pcmk-1 ~]# pcs cluster cib active_cfg
[root@pcmk-1 ~]# pcs -f active_cfg resource clone WebFS
[root@pcmk-1 ~]# pcs -f active_cfg constraint
Location Constraints:
  Resource: WebSite
    Enabled on:
      Node: pcmk-2 (score:50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
  promote WebData-clone then start WebFS-clone (kind:Mandatory)
  start WebFS-clone then start WebSite (kind:Mandatory)
  start dlm-clone then start WebFS-clone (kind:Mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
  WebFS-clone with WebData-clone (score:INFINITY) (rsc-role:Started) (with-rsc-role:Promoted)
  WebSite with WebFS-clone (score:INFINITY)
  WebFS-clone with dlm-clone (score:INFINITY)
Ticket Constraints:
```

Tell the cluster that it is now allowed to promote both instances to be DRBD Primary.

```
[root@pcmk-1 ~]# pcs -f active_cfg resource update WebData-clone promoted-max=2
```

Finally, load our configuration to the cluster, and re-enable the `WebFS` resource (which we disabled earlier).

```
[root@pcmk-1 ~]# pcs cluster cib-push active_cfg --config
CIB updated
[root@pcmk-1 ~]# pcs resource enable WebFS
```

After all the processes are started, the status should look similar to this.

```
[root@pcmk-1 ~]# pcs resource
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 pcmk-2 ]
  * Clone Set: dlm-clone [dlm]:
    * Started: [ pcmk-1 pcmk-2 ]
  * Clone Set: WebFS-clone [WebFS]:
    * Started: [ pcmk-1 pcmk-2 ]
```

2.9.6 Test Failover

Testing failover is left as an exercise for the reader.

With this configuration, the data is now active/active. The website administrator could change HTML files on either node, and the live website will show the changes even if it is running on the opposite node.

If the web server is configured to listen on all IP addresses, it is possible to remove the constraints between the `WebSite` and `ClusterIP` resources, and clone the `WebSite` resource. The web server would always be ready to serve web pages, and only the IP address would need to be moved in a failover.

2.10 Configuration Recap

2.10.1 Final Cluster Configuration

```
[root@pcmk-1 ~]# pcs resource
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 pcmk-2 ]
  * Clone Set: dlm-clone [dlm]:
    * Started: [ pcmk-1 pcmk-2 ]
  * Clone Set: WebFS-clone [WebFS]:
    * Started: [ pcmk-1 pcmk-2 ]
```

```
[root@pcmk-1 ~]# pcs resource op defaults
Meta Attrs: op_defaults-meta_attributes
  timeout=240s
```

```
[root@pcmk-1 ~]# pcs stonith
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
```

```
[root@pcmk-1 ~]# pcs constraint
Location Constraints:
  Resource: WebSite
    Enabled on:
      Node: pcmk-2 (score:50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory)
  promote WebData-clone then start WebFS-clone (kind:Mandatory)
  start WebFS-clone then start WebSite (kind:Mandatory)
  start dlm-clone then start WebFS-clone (kind:Mandatory)
```

(continues on next page)

(continued from previous page)

```
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY)
  WebFS-clone with WebData-clone (score:INFINITY) (rsc-role:Started) (with-rsc-role:Promoted)
  WebSite with WebFS-clone (score:INFINITY)
  WebFS-clone with dlm-clone (score:INFINITY)
Ticket Constraints:
```

```
[root@pcmk-1 ~]# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.1.2-4.el9-ada5c3b36e2) - partition with quorum
  * Last updated: Wed Jul 27 08:57:57 2022
  * Last change:  Wed Jul 27 08:55:00 2022 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 9 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]

Full List of Resources:
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
  * ClusterIP       (ocf:heartbeat:IPaddr2):         Started pcmk-1
  * WebSite (ocf:heartbeat:apache):  Started pcmk-1
  * Clone Set: WebData-clone [WebData] (promotable):
    * Promoted: [ pcmk-1 pcmk-2 ]
  * Clone Set: dlm-clone [dlm]:
    * Started: [ pcmk-1 pcmk-2 ]
  * Clone Set: WebFS-clone [WebFS]:
    * Started: [ pcmk-1 pcmk-2 ]

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

```
[root@pcmk-1 ~]# pcs config
Cluster Name: mycluster
Corosync Nodes:
 pcmk-1 pcmk-2
Pacemaker Nodes:
 pcmk-1 pcmk-2

Resources:
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: cidr_netmask=24 ip=192.168.122.120
  Operations: monitor interval=30s (ClusterIP-monitor-interval-30s)
              start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
 Resource: WebSite (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
  Operations: monitor interval=1min (WebSite-monitor-interval-1min)
              start interval=0s timeout=40s (WebSite-start-interval-0s)
              stop interval=0s timeout=60s (WebSite-stop-interval-0s)
 Clone: WebData-clone
  Meta Attrs: clone-max=2 clone-node-max=1 notify=true promotable=true promoted-max=2 promoted-
↪node-max=1
```

(continues on next page)

(continued from previous page)

```
  Resource: WebData (class=ocf provider=linbit type=drbd)
   Attributes: drbd_resource=wwwdata
   Operations: demote interval=0s timeout=90 (WebData-demote-interval-0s)
               monitor interval=29s role=Promoted (WebData-monitor-interval-29s)
               monitor interval=31s role=Unpromoted (WebData-monitor-interval-31s)
               notify interval=0s timeout=90 (WebData-notify-interval-0s)
               promote interval=0s timeout=90 (WebData-promote-interval-0s)
               reload interval=0s timeout=30 (WebData-reload-interval-0s)
               start interval=0s timeout=240 (WebData-start-interval-0s)
               stop interval=0s timeout=100 (WebData-stop-interval-0s)
 Clone: dlm-clone
  Meta Attrs: interleave=true ordered=true
  Resource: dlm (class=ocf provider=pacemaker type=controld)
   Operations: monitor interval=60s (dlm-monitor-interval-60s)
               start interval=0s timeout=90s (dlm-start-interval-0s)
               stop interval=0s timeout=100s (dlm-stop-interval-0s)
 Clone: WebFS-clone
  Resource: WebFS (class=ocf provider=heartbeat type=Filesystem)
   Attributes: device=/dev/drbd1 directory=/var/www/html fstype=gfs2
   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)

Stonith Devices:
 Resource: fence_dev (class=stonith type=some_fence_agent)
  Attributes: pcmk_delay_base=pcmk-1:5s;pcmk-2:0s pcmk_host_map=pcmk-1:almalinux9-1;pcmk-
↪2:almalinux9-2
  Operations: monitor interval=60s (fence_dev-monitor-interval-60s)
Fencing Levels:

Location Constraints:
  Resource: WebSite
    Enabled on:
      Node: pcmk-2 (score:50) (id:location-WebSite-pcmk-2-50)
Ordering Constraints:
  start ClusterIP then start WebSite (kind:Mandatory) (id:order-ClusterIP-WebSite-mandatory)
  promote WebData-clone then start WebFS-clone (kind:Mandatory) (id:order-WebData-clone-WebFS-
↪mandatory)
  start WebFS-clone then start WebSite (kind:Mandatory) (id:order-WebFS-WebSite-mandatory)
  start dlm-clone then start WebFS-clone (kind:Mandatory) (id:order-dlm-clone-WebFS-mandatory)
Colocation Constraints:
  WebSite with ClusterIP (score:INFINITY) (id:colocation-WebSite-ClusterIP-INFINITY)
  WebFS-clone with WebData-clone (score:INFINITY) (rsc-role:Started) (with-rsc-role:Promoted)␣
↪(id:colocation-WebFS-WebData-clone-INFINITY)
  WebSite with WebFS-clone (score:INFINITY) (id:colocation-WebSite-WebFS-INFINITY)
  WebFS-clone with dlm-clone (score:INFINITY) (id:colocation-WebFS-dlm-clone-INFINITY)
Ticket Constraints:

Alerts:
 No alerts defined

Resources Defaults:
  Meta Attrs: build-resource-defaults
    resource-stickiness=100
Operations Defaults:
  Meta Attrs: op_defaults-meta_attributes
```

(continues on next page)

(continued from previous page)

```
    timeout=240s

Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: mycluster
 dc-version: 2.1.2-4.el9-ada5c3b36e2
 have-watchdog: false
 last-lrm-refresh: 1658896047
 no-quorum-policy: freeze
 stonith-enabled: true

Tags:
 No tags defined

Quorum:
  Options:
```

2.10.2 Node List

```
[root@pcmk-1 ~]# pcs status nodes
Pacemaker Nodes:
 Online: pcmk-1 pcmk-2
 Standby:
 Standby with resource(s) running:
 Maintenance:
 Offline:
Pacemaker Remote Nodes:
 Online:
 Standby:
 Standby with resource(s) running:
 Maintenance:
 Offline:
```

2.10.3 Cluster Options

```
[root@pcmk-1 ~]# pcs property
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: mycluster
 dc-version: 2.1.2-4.el9-ada5c3b36e2
 have-watchdog: false
 no-quorum-policy: freeze
 stonith-enabled: true
```

The output shows cluster-wide configuration options, as well as some baseline- level state information. The output includes:

- `cluster-infrastructure` - the cluster communications layer in use
- `cluster-name` - the cluster name chosen by the administrator when the cluster was created
- `dc-version` - the version (including upstream source-code hash) of `pacemaker` used on the Designated Controller, which is the node elected to determine what actions are needed when events occur

- `have-watchdog` - whether watchdog integration is enabled; set automatically when SBD is enabled
- `stonith-enabled` - whether nodes may be fenced as part of recovery

Note: This command is equivalent to `pcs property config`.

2.10.4 Resources

Default Options

```
[root@pcmk-1 ~]# pcs resource defaults
Meta Attrs: build-resource-defaults
  resource-stickiness=100
```

This shows cluster option defaults that apply to every resource that does not explicitly set the option itself. Above:

- `resource-stickiness` - Specify how strongly a resource prefers to remain on its current node. Alternatively, you can view this as the level of aversion to moving healthy resources to other machines.

Fencing

```
[root@pcmk-1 ~]# pcs stonith status
  * fence_dev       (stonith:some_fence_agent):      Started pcmk-1
[root@pcmk-1 ~]# pcs stonith config
 Resource: fence_dev (class=stonith type=some_fence_agent)
  Attributes: pcmk_delay_base=pcmk-1:5s;pcmk-2:0s pcmk_host_map=pcmk-1:almalinux9-1;pcmk-
↪2:almalinux9-2
  Operations: monitor interval=60s (fence_dev-monitor-interval-60s)
```

Service Address

Users of the services provided by the cluster require an unchanging address with which to access it.

```
[root@pcmk-1 ~]# pcs resource config ClusterIP
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: cidr_netmask=24 ip=192.168.122.120
  Operations: monitor interval=30s (ClusterIP-monitor-interval-30s)
              start interval=0s timeout=20s (ClusterIP-start-interval-0s)
              stop interval=0s timeout=20s (ClusterIP-stop-interval-0s)
```

DRBD - Shared Storage

Here, we define the DRBD service and specify which DRBD resource (from `/etc/drbd.d/\*.res`) it should manage. We make it a promotable clone resource and, in order to have an active/active setup, allow both instances to be promoted at the same time. We also set the notify option so that the cluster will tell the `drbd` agent when its peer changes state.

```
[root@pcmk-1 ~]# pcs resource config WebData-clone
 Clone: WebData-clone
  Meta Attrs: clone-max=2 clone-node-max=1 notify=true promotable=true promoted-max=2 promoted-
↪node-max=1
  Resource: WebData (class=ocf provider=linbit type=drbd)
   Attributes: drbd_resource=wwwdata
   Operations: demote interval=0s timeout=90 (WebData-demote-interval-0s)
               monitor interval=29s role=Promoted (WebData-monitor-interval-29s)
               monitor interval=31s role=Unpromoted (WebData-monitor-interval-31s)
               notify interval=0s timeout=90 (WebData-notify-interval-0s)
               promote interval=0s timeout=90 (WebData-promote-interval-0s)
               reload interval=0s timeout=30 (WebData-reload-interval-0s)
               start interval=0s timeout=240 (WebData-start-interval-0s)
               stop interval=0s timeout=100 (WebData-stop-interval-0s)
[root@pcmk-1 ~]# pcs constraint ref WebData-clone
Resource: WebData-clone
  colocation-WebFS-WebData-clone-INFINITY
  order-WebData-clone-WebFS-mandatory
```

Cluster Filesystem

The cluster filesystem ensures that files are read and written correctly. We need to specify the block device (provided by DRBD), where we want it mounted and that we are using GFS2. Again, it is a clone because it is intended to be active on both nodes. The additional constraints ensure that it can only be started on nodes with active DLM and DRBD instances.

```
[root@pcmk-1 ~]# pcs resource config WebFS-clone
 Clone: WebFS-clone
  Resource: WebFS (class=ocf provider=heartbeat type=Filesystem)
   Attributes: device=/dev/drbd1 directory=/var/www/html fstype=gfs2
   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)
[root@pcmk-1 ~]# pcs constraint ref WebFS-clone
Resource: WebFS-clone
  colocation-WebFS-WebData-clone-INFINITY
  colocation-WebSite-WebFS-INFINITY
  colocation-WebFS-dlm-clone-INFINITY
  order-WebData-clone-WebFS-mandatory
  order-WebFS-WebSite-mandatory
  order-dlm-clone-WebFS-mandatory
```

Apache

Lastly, we have the actual service, Apache. We need only tell the cluster where to find its main configuration file and restrict it to running on a node that has the required filesystem mounted and the IP address active.

```
[root@pcmk-1 ~]# pcs resource config WebSite
 Resource: WebSite (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
  Operations: monitor interval=1min (WebSite-monitor-interval-1min)
              start interval=0s timeout=40s (WebSite-start-interval-0s)
              stop interval=0s timeout=60s (WebSite-stop-interval-0s)
[root@pcmk-1 ~]# pcs constraint ref WebSite
```

(continues on next page)

(continued from previous page)

```
Resource: WebSite
  colocation-WebSite-ClusterIP-INFINITY
  colocation-WebSite-WebFS-INFINITY
  location-WebSite-pcmk-2-50
  order-ClusterIP-WebSite-mandatory
  order-WebFS-WebSite-mandatory
```

2.11 Sample Corosync Configuration

Sample corosync.conf for two-node cluster created by pcs.

```
totem {
    version: 2
    cluster_name: mycluster
    transport: knet
    crypto_cipher: aes256
    crypto_hash: sha256
    cluster_uuid: e592f61f916943978bdf7c046a195980
}

nodelist {
    node {
        ring0_addr: pcmk-1
        name: pcmk-1
        nodeid: 1
    }

    node {
        ring0_addr: pcmk-2
        name: pcmk-2
        nodeid: 2
    }
}

quorum {
    provider: corosync_votequorum
    two_node: 1
}

logging {
    to_logfile: yes
    logfile: /var/log/cluster/corosync.log
    to_syslog: yes
    timestamp: on
}
```

2.12 Further Reading

- Project Website https://www.clusterlabs.org/

- SuSE has a comprehensive guide to cluster commands (though using the `crmsh` command-line shell rather than `pcs`) at: https://www.suse.com/documentation/sle_ha/book_sleha/data/book_sleha.html
- Corosync http://www.corosync.org/

CHAPTER

THREE

INDEX

- genindex
- search

INDEX