Pacemaker Remote

Release 2.1.2

the Pacemaker project contributors

Nov 24, 2021

CONTENTS

Scaling High Availablity Clusters

CHAPTER

ONE

ABSTRACT

This document exists as both a reference and deployment guide for the Pacemaker Remote service.

The example commands in this document will use:

- CentOS Stream 8 as the host operating system
- Pacemaker Remote to perform resource management within guest nodes and remote nodes
- KVM for virtualization
- libvirt to manage guest nodes
- Corosync to provide messaging and membership services on cluster nodes
- Pacemaker 2 to perform resource management on cluster nodes
- pcs as the cluster configuration toolset

The concepts are the same for other distributions, virtualization platforms, toolsets, and messaging layers, and should be easily adaptable.

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Scaling a Pacemaker Cluster

2.1.1 Overview

In a basic Pacemaker high-availability cluster[1] each node runs the full cluster stack of Corosync and all Pacemaker components. This allows great flexibility but limits scalability to around 16 nodes.

To allow for scalability to dozens or even hundreds of nodes, Pacemaker allows nodes not running the full cluster stack to integrate into the cluster and have the cluster manage their resources as if they were a cluster node.

2.1.2 Terms

cluster node A node running the full high-availability stack of corosync and all Pacemaker components. Cluster nodes may run cluster resources, run all Pacemaker command-line tools (`crm_mon`, `crm_resource` and so on), execute fencing actions, count toward cluster quorum, and serve as the cluster's Designated Controller (DC).

pacemaker-remoted A small service daemon that allows a host to be used as a Pacemaker node without running the full cluster stack. Nodes running `pacemaker-remoted` may run cluster resources and most command-line tools, but cannot perform other functions of full cluster nodes such as fencing execution, quorum voting, or DC eligibility. The `pacemaker-remoted` daemon is an enhanced version of Pacemaker's local executor daemon (pacemaker-execd).

pacemaker_remote The name of the systemd service that manages `pacemaker-remoted`

Pacemaker Remote A way to refer to the general technology implementing nodes running `pacemaker-remoted`, including the cluster-side implementation and the communication protocol between them.

remote node A physical host running `pacemaker-remoted`. Remote nodes have a special resource that manages communication with the cluster. This is sometimes referred to as the *bare metal* case.

guest node A virtual host running `pacemaker-remoted`. Guest nodes differ from remote nodes mainly in that the guest node is itself a resource that the cluster manages.

Note: *Remote* in this document refers to the node not being a part of the underlying corosync cluster. It has nothing to do with physical proximity. Remote nodes and guest nodes are subject to the same latency

[1] See the https://www.clusterlabs.org/doc/ Pacemaker documentation, especially *Clusters From Scratch* and *Pacemaker Explained*.

requirements as cluster nodes, which means they are typically in the same data center.

Note: It is important to distinguish the various roles a virtual machine can serve in Pacemaker clusters:

- A virtual machine can run the full cluster stack, in which case it is a cluster node and is not itself managed by the cluster.
- A virtual machine can be managed by the cluster as a resource, without the cluster having any awareness of the services running inside the virtual machine. The virtual machine is *opaque* to the cluster.
- A virtual machine can be a cluster resource, and run `pacemaker-remoted` to make it a guest node, allowing the cluster to manage services inside it. The virtual machine is *transparent* to the cluster.

2.1.3 Guest Nodes

"I want a Pacemaker cluster to manage virtual machine resources, but I also want Pacemaker to be able to manage the resources that live within those virtual machines."

Without `pacemaker-remoted`, the possibilities for implementing the above use case have significant limitations:

- The cluster stack could be run on the physical hosts only, which loses the ability to monitor resources within the guests.
- A separate cluster could be on the virtual guests, which quickly hits scalability issues.
- The cluster stack could be run on the guests using the same cluster as the physical hosts, which also hits scalability issues and complicates fencing.

With `pacemaker-remoted`:

- The physical hosts are cluster nodes (running the full cluster stack).
- The virtual machines are guest nodes (running `pacemaker-remoted`). Nearly zero configuration is required on the virtual machine.
- The cluster stack on the cluster nodes launches the virtual machines and immediately connects to `pacemaker-remoted` on them, allowing the virtual machines to integrate into the cluster.

The key difference here between the guest nodes and the cluster nodes is that the guest nodes do not run the cluster stack. This means they will never become the DC, initiate fencing actions or participate in quorum voting.

On the other hand, this also means that they are not bound to the scalability limits associated with the cluster stack (no 16-node corosync member limits to deal with). That isn't to say that guest nodes can scale indefinitely, but it is known that guest nodes scale horizontally much further than cluster nodes.

Other than the quorum limitation, these guest nodes behave just like cluster nodes with respect to resource management. The cluster is fully capable of managing and monitoring resources on each guest node. You can build constraints against guest nodes, put them in standby, or do whatever else you'd expect to be able to do with cluster nodes. They even show up in `crm_mon` output as nodes.

To solidify the concept, below is an example that is very similar to an actual deployment we test in our developer environment to verify guest node scalability:

- 16 cluster nodes running the full Corosync + Pacemaker stack
- 64 Pacemaker-managed virtual machine resources running `pacemaker-remoted` configured as guest nodes

- 64 Pacemaker-managed webserver and database resources configured to run on the 64 guest nodes

With this deployment, you would have 64 webservers and databases running on 64 virtual machines on 16 hardware nodes, all of which are managed and monitored by the same Pacemaker deployment. It is known that `pacemaker-remoted` can scale to these lengths and possibly much further depending on the specific scenario.

2.1.4 Remote Nodes

"I want my traditional high-availability cluster to scale beyond the limits imposed by the corosync messaging layer."

Ultimately, the primary advantage of remote nodes over cluster nodes is scalability. There are likely some other use cases related to geographically distributed HA clusters that remote nodes may serve a purpose in, but those use cases are not well understood at this point.

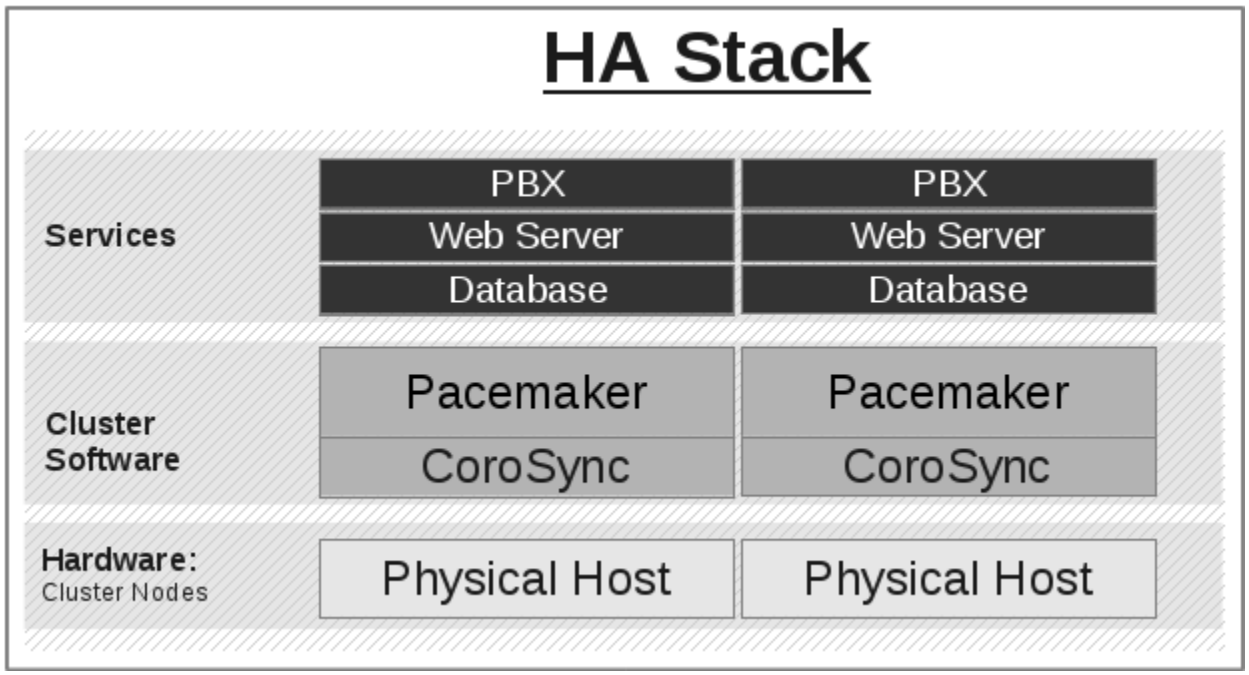
Like guest nodes, remote nodes will never become the DC, initiate fencing actions or participate in quorum voting.

That is not to say, however, that fencing of a remote node works any differently than that of a cluster node. The Pacemaker scheduler understands how to fence remote nodes. As long as a fencing device exists, the cluster is capable of ensuring remote nodes are fenced in the exact same way as cluster nodes.

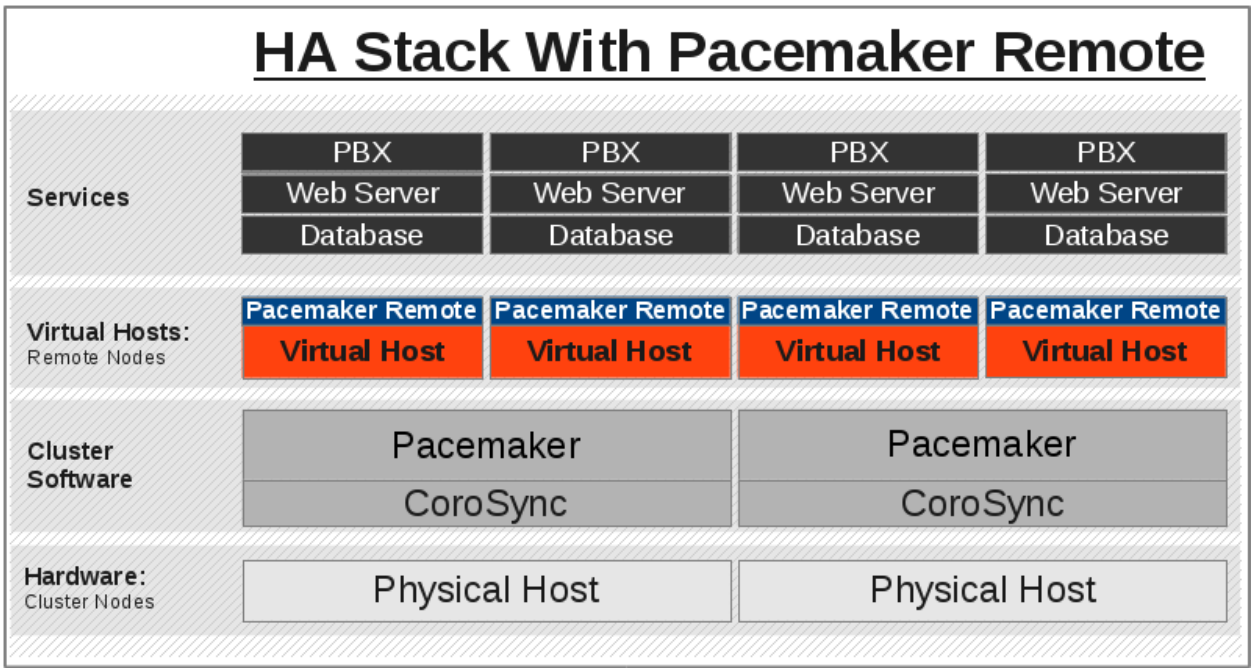
2.1.5 Expanding the Cluster Stack

With `pacemaker-remoted`, the traditional view of the high-availability stack can be expanded to include a new layer:

Traditional HA Stack



HA Stack With Guest Nodes



2.2 Configuration Explained

The walk-through examples use some of these options, but don't explain exactly what they mean or do. This section is meant to be the go-to resource for all the options available for configuring Pacemaker Remote.

2.2.1 Resource Meta-Attributes for Guest Nodes

When configuring a virtual machine as a guest node, the virtual machine is created using one of the usual resource agents for that purpose (for example, **ocf:heartbeat:VirtualDomain** or **ocf:heartbeat:Xen**), with additional meta-attributes.

No restrictions are enforced on what agents may be used to create a guest node, but obviously the agent must create a distinct environment capable of running the pacemaker_remote daemon and cluster resources. An additional requirement is that fencing the host running the guest node resource must be sufficient for ensuring the guest node is stopped. This means, for example, that not all hypervisors supported by **VirtualDomain** may be used to create guest nodes; if the guest can survive the hypervisor being fenced, it may not be used as a guest node.

Below are the meta-attributes available to enable a resource as a guest node and define its connection parameters.

Table 1: **Meta-attributes for configuring VM resources as guest nodes**

Option	Default	Description
remote-node	none	The node name of the guest node this resource defines. This both enables the resource as a guest node and defines the unique name used to identify the guest node. If no other parameters are set, this value will also be assumed as the hostname to use when connecting to pacemaker_remote on the VM. This value **must not** overlap with any resource or node IDs.
remote-port	3121	The port on the virtual machine that the cluster will use to connect to pacemaker_remote.
remote-addr	'value of' `remote-node`	The IP address or hostname to use when connecting to pacemaker_remote on the VM.
remote-connect-timeout	60s	How long before a pending guest connection will time out.

2.2.2 Connection Resources for Remote Nodes

A remote node is defined by a connection resource. That connection resource has instance attributes that define where the remote node is located on the network and how to communicate with it.

Descriptions of these instance attributes can be retrieved using the following `pcs` command:

```
# pcs resource describe remote
ocf:pacemaker:remote - remote resource agent

Resource options:
  server: Server location to connect to (IP address or resolvable host name)
  port: TCP port at which to contact Pacemaker Remote executor
  reconnect_interval: If this is a positive time interval, the cluster will attempt to
                      reconnect to a remote node after an active connection has been
                      lost at this interval. Otherwise, the cluster will attempt to
                      reconnect immediately (after any fencing needed).
```

When defining a remote node's connection resource, it is common and recommended to name the connection resource the same as the remote node's hostname. By default, if no `server` option is provided, the cluster will attempt to contact the remote node using the resource name as the hostname.

2.2.3 Environment Variables for Daemon Start-up

Authentication and encryption of the connection between cluster nodes and nodes running pacemaker_remote is achieved using with TLS-PSK encryption/authentication over TCP (port 3121 by default). This means that both the cluster node and remote node must share the same private key. By default, this key is placed at `/etc/pacemaker/authkey` on each node.

You can change the default port and/or key location for Pacemaker and `pacemaker_remoted` via environment variables. How these variables are set varies by OS, but usually they are set in the `/etc/sysconfig/pacemaker` or `/etc/default/pacemaker` file.

```
#==#==# Pacemaker Remote
# Use the contents of this file as the authorization key to use with Pacemaker
# Remote connections. This file must be readable by Pacemaker daemons (that is,
# it must allow read permissions to either the hacluster user or the haclient
# group), and its contents must be identical on all nodes. The default is
# "/etc/pacemaker/authkey".
# PCMK_authkey_location=/etc/pacemaker/authkey

# If the Pacemaker Remote service is run on the local node, it will listen
# for connections on this address. The value may be a resolvable hostname or an
# IPv4 or IPv6 numeric address. When resolving names or using the default
# wildcard address (i.e. listen on all available addresses), IPv6 will be
# preferred if available. When listening on an IPv6 address, IPv4 clients will
# be supported (via IPv4-mapped IPv6 addresses).
# PCMK_remote_address="192.0.2.1"

# Use this TCP port number when connecting to a Pacemaker Remote node. This
# value must be the same on all nodes. The default is "3121".
# PCMK_remote_port=3121

# Use these GnuTLS cipher priorities for TLS connections. See:
#
#   https://gnutls.org/manual/html_node/Priority-Strings.html
#
# Pacemaker will append ":+ANON-DH" for remote CIB access (when enabled) and
# ":+DHE-PSK:+PSK" for Pacemaker Remote connections, as they are required for
# the respective functionality.
# PCMK_tls_priorities="NORMAL"

# Set bounds on the bit length of the prime number generated for Diffie-Hellman
# parameters needed by TLS connections. The default is not to set any bounds.
#
# If these values are specified, the server (Pacemaker Remote daemon, or CIB
# manager configured to accept remote clients) will use these values to provide
# a floor and/or ceiling for the value recommended by the GnuTLS library. The
# library will only accept a limited number of specific values, which vary by
# library version, so setting these is recommended only when required for
# compatibility with specific client versions.
#
# If PCMK_dh_min_bits is specified, the client (connecting cluster node or
# remote CIB command) will require that the server use a prime of at least this
# size. This is only recommended when the value must be lowered in order for
# the client's GnuTLS library to accept a connection to an older server.
# The client side does not use PCMK_dh_max_bits.
#
# PCMK_dh_min_bits=1024
# PCMK_dh_max_bits=2048
```

2.2.4 Removing Remote Nodes and Guest Nodes

If the resource creating a guest node, or the **ocf:pacemaker:remote** resource creating a connection to a remote node, is removed from the configuration, the affected node will continue to show up in output as an offline node.

If you want to get rid of that output, run (replacing `$NODE_NAME` appropriately):

```
# crm_node --force --remove $NODE_NAME
```

Warning: Be absolutely sure that there are no references to the node's resource in the configuration before running the above command.

2.3 Guest Node Walk-through

What this tutorial is: An in-depth walk-through of how to get Pacemaker to manage a KVM guest instance and integrate that guest into the cluster as a guest node.

What this tutorial is not: A realistic deployment scenario. The steps shown here are meant to get users familiar with the concept of guest nodes as quickly as possible.

2.3.1 Configure Cluster Nodes

This walk-through assumes you already have a Pacemaker cluster configured. For examples, we will use a cluster with two cluster nodes named pcmk-1 and pcmk-2. You can substitute whatever your node names are, for however many nodes you have. If you are not familiar with setting up basic Pacemaker clusters, follow the walk-through in the Clusters From Scratch document before attempting this one.

You will need to add the remote node's hostname (we're using **guest1** in this tutorial) to the cluster nodes' `/etc/hosts` files if you haven't already. This is required unless you have DNS set up in a way where guest1's address can be discovered.

Execute the following on each cluster node, replacing the IP address with the actual IP address of the remote node.

```
# cat << END >> /etc/hosts
192.168.122.10    guest1
END
```

Install Virtualization Software

On each node within your cluster, install virt-install, libvirt, and qemu-kvm. Start and enable libvirtd.

```
# yum install -y virt-install libvirt qemu-kvm
# systemctl start libvirtd
# systemctl enable libvirtd
```

Reboot the host.

Note: While KVM is used in this example, any virtualization platform with a Pacemaker resource agent can be used to create a guest node. The resource agent needs only to support usual commands (start, stop, etc.); Pacemaker implements the **remote-node** meta-attribute, independent of the agent.

2.3.2 Configure the KVM guest

Create Guest

Create a KVM guest to use as a guest node. Be sure to configure the guest with a hostname and a static IP address (as an example here, we will use guest1 and 192.168.122.10). Here's an example way to create a guest:

- Download an .iso file from the CentOS Mirrors List into a directory on your cluster node.
- Run the following command, using your own path for the **location** flag:

```
# virt-install \
  --name guest-vm \
  --ram 1024 \
  --disk path=./guest-vm.qcow2,size=1 \
  --vcpus 2 \
  --os-type linux \
  --os-variant centos-stream8\
  --network bridge=virbr0 \
  --graphics none \
  --console pty,target_type=serial \
  --location <path to your .iso file> \
  --extra-args 'console=ttyS0,115200n8 serial'
```

Configure Firewall on Guest

On each guest, allow cluster-related services through the local firewall.

Verify Connectivity

At this point, you should be able to ping and ssh into guests from hosts, and vice versa.

Configure pacemaker_remote on Guest Node

Install the pacemaker_remote daemon on the guest node. Here, we also install the `pacemaker` package; it is not required, but it contains the dummy resource agent that we will use later for testing.

```
# yum install -y pacemaker-remote resource-agents pcs pacemaker
```

2.3.3 Integrate Guest into Cluster

Now the fun part, integrating the virtual machine you've just created into the cluster. It is incredibly simple.

Start the Cluster

On the host, start Pacemaker.

```
# pcs cluster start
```

Wait for the host to become the DC.

Integrate Guest Node into Cluster

We will use the following command, which creates the VirtualDomain resource, creates and copies the key, and enables pacemaker_remote:

```
# pcs cluster node add-guest guest1
```

Once the **vm-guest1** resource is started you will see **guest1** appear in the `pcs status` output as a node. The final `pcs status` output should look something like this, and you can see that it created the VirtualDomain resource:

```
# pcs status
Cluster name: mycluster

Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.0.5-8.el8-ba59be7122) - partition with quorum
  * Last updated: Wed Mar 17 08:37:37 2021
  * Last change:  Wed Mar 17 08:31:01 2021 by root via cibadmin on pcmk-1
  * 3 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]
  * GuestOnline: [ guest1@pcmk-1 ]

Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain):  pcmk-1

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

How pcs Configures the Guest

To see that it created the key and copied it to all cluster nodes and the guest, run:

```
# ls -l /etc/pacemaker
```

To see that it enables pacemaker_remote, run:

```
# systemctl status pacemaker_remote

 pacemaker_remote.service - Pacemaker Remote executor daemon
   Loaded: loaded (/usr/lib/systemd/system/pacemaker_remote.service; enabled; vendor preset:␣
↪disabled)
   Active: active (running) since Wed 2021-03-17 08:31:01 EDT; 1min 5s ago
     Docs: man:pacemaker-remoted
           https://clusterlabs.org/pacemaker/doc/
 Main PID: 90160 (pacemaker-remot)
    Tasks: 1
   Memory: 1.4M
   CGroup: /system.slice/pacemaker_remote.service
           90160 /usr/sbin/pacemaker-remoted
```

(continues on next page)

(continued from previous page)

```
Mar 17 08:31:01 guest1 systemd[1]: Started Pacemaker Remote executor daemon.
Mar 17 08:31:01 guest1 pacemaker-remoted[90160]:  notice: Additional logging available in /var/log/
↪pacemaker/pacemaker.log
Mar 17 08:31:01 guest1 pacemaker-remoted[90160]:  notice: Starting Pacemaker remote executor
Mar 17 08:31:01 guest1 pacemaker-remoted[90160]:  notice: Pacemaker remote executor successfully␣
↪started and accepting connections
```

Note: Pacemaker will automatically monitor pacemaker_remote connections for failure, so it is not necessary to create a recurring monitor on the **VirtualDomain** resource.

2.3.4 Starting Resources on KVM Guest

The commands below demonstrate how resources can be executed on both the guest node and the cluster node.

Create a few Dummy resources. Dummy resources are real resource agents used just for testing purposes. They actually execute on the host they are assigned to just like an apache server or database would, except their execution just means a file was created. When the resource is stopped, that the file it created is removed.

```
# pcs resource create FAKE1 ocf:pacemaker:Dummy
# pcs resource create FAKE2 ocf:pacemaker:Dummy
# pcs resource create FAKE3 ocf:pacemaker:Dummy
# pcs resource create FAKE4 ocf:pacemaker:Dummy
# pcs resource create FAKE5 ocf:pacemaker:Dummy
```

Now check your `pcs status` output. In the resource section, you should see something like the following, where some of the resources started on the cluster node, and some started on the guest node.

```
Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain):  Started pcmk-1
  * FAKE1   (ocf::pacemaker:Dummy):  Started guest1
  * FAKE2   (ocf::pacemaker:Dummy):  Started guest1
  * FAKE3   (ocf::pacemaker:Dummy):  Started pcmk-1
  * FAKE4   (ocf::pacemaker:Dummy):  Started guest1
  * FAKE5   (ocf::pacemaker:Dummy):  Started pcmk-1
```

The guest node, **guest1**, reacts just like any other node in the cluster. For example, pick out a resource that is running on your cluster node. For my purposes, I am picking FAKE3 from the output above. We can force FAKE3 to run on **guest1** in the exact same way we would any other node.

```
# pcs constraint location FAKE3 prefers guest1
```

Now, looking at the bottom of the *pcs status* output you'll see FAKE3 is on **guest1**.

```
Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain):  Started pcmk-1
  * FAKE1   (ocf::pacemaker:Dummy):  Started guest1
  * FAKE2   (ocf::pacemaker:Dummy):  Started guest1
  * FAKE3   (ocf::pacemaker:Dummy):  Started guest1
  * FAKE4   (ocf::pacemaker:Dummy):  Started pcmk-1
  * FAKE5   (ocf::pacemaker:Dummy):  Started pcmk-1
```

2.3.5 Testing Recovery and Fencing

Pacemaker's scheduler is smart enough to know fencing guest nodes associated with a virtual machine means shutting off/rebooting the virtual machine. No special configuration is necessary to make this happen. If you are interested in testing this functionality out, trying stopping the guest's pacemaker_remote daemon. This would be equivalent of abruptly terminating a cluster node's corosync membership without properly shutting it down.

ssh into the guest and run this command.

```
# kill -9 $(pidof pacemaker-remoted)
```

Within a few seconds, your `pcs status` output will show a monitor failure, and the **guest1** node will not be shown while it is being recovered.

```
# pcs status
Cluster name: mycluster

Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.0.5-8.el8-ba59be7122) - partition with quorum
  * Last updated: Wed Mar 17 08:37:37 2021
  * Last change:  Wed Mar 17 08:31:01 2021 by root via cibadmin on pcmk-1
  * 3 nodes configured
  * 7 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]
  * GuestOnline: [ guest1@pcmk-1 ]

Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain):  pcmk-1
  * FAKE1   (ocf::pacemaker:Dummy):  Stopped
  * FAKE2   (ocf::pacemaker:Dummy):  Stopped
  * FAKE3   (ocf::pacemaker:Dummy):  Stopped
  * FAKE4   (ocf::pacemaker:Dummy):  Started pcmk-1
  * FAKE5   (ocf::pacemaker:Dummy):  Started pcmk-1

Failed Actions:
* guest1_monitor_30000 on pcmk-1 'unknown error' (1): call=8, status=Error, exitreason='none',
    last-rc-change='Wed Mar 17 08:32:01 2021', queued=0ms, exec=0ms

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

Note: A guest node involves two resources: the one you explicitly configured creates the guest, and Pacemaker creates an implicit resource for the pacemaker_remote connection, which will be named the same as the value of the **remote-node** attribute of the explicit resource. When we killed pacemaker_remote, it is the implicit resource that failed, which is why the failed action starts with **guest1** and not **vm-guest1**.

Once recovery of the guest is complete, you'll see it automatically get re-integrated into the cluster. The final `pcs status` output should look something like this.

```
# pcs status
Cluster name: mycluster

Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.0.5-8.el8-ba59be7122) - partition with quorum
  * Last updated: Wed Mar 17 08:37:37 2021
  * Last change:  Wed Mar 17 08:31:01 2021 by root via cibadmin on pcmk-1
  * 3 nodes configured
  * 7 resource instances configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]
  * GuestOnline: [ guest1@pcmk-1 ]

Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain):  pcmk-1
  * FAKE1   (ocf::pacemaker:Dummy):  Stopped
  * FAKE2   (ocf::pacemaker:Dummy):  Stopped
  * FAKE3   (ocf::pacemaker:Dummy):  Stopped
  * FAKE4   (ocf::pacemaker:Dummy):  Started pcmk-1
  * FAKE5   (ocf::pacemaker:Dummy):  Started pcmk-1

Failed Actions:
* guest1_monitor_30000 on pcmk-1 'unknown error' (1): call=8, status=Error, exitreason='none',
    last-rc-change='Fri Jan 12 18:08:29 2018', queued=0ms, exec=0ms

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

Normally, once you've investigated and addressed a failed action, you can clear the failure. However Pacemaker does not yet support cleanup for the implicitly created connection resource while the explicit resource is active. If you want to clear the failed action from the status output, stop the guest resource before clearing it. For example:

```
# pcs resource disable vm-guest1 --wait
# pcs resource cleanup guest1
# pcs resource enable vm-guest1
```

2.3.6 Accessing Cluster Tools from Guest Node

Besides allowing the cluster to manage resources on a guest node, pacemaker_remote has one other trick. The pacemaker_remote daemon allows nearly all the pacemaker tools (`crm_resource`, `crm_mon`, `crm_attribute`, etc.) to work on guest nodes natively.

Try it: Run `crm_mon` on the guest after pacemaker has integrated the guest node into the cluster. These tools just work. This means resource agents such as promotable resources (which need access to tools like `crm_attribute`) work seamlessly on the guest nodes.

Higher-level command shells such as `pcs` may have partial support on guest nodes, but it is recommended to run them from a cluster node.

Guest nodes will show up in `crm_mon` output as normal. For example, this is the `crm_mon` output after **guest1** is integrated into the cluster:

```
Cluster name: mycluster

Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.0.5-8.el8-ba59be7122) - partition with quorum
  * Last updated: Wed Mar 17 08:37:37 2021
  * Last change:  Wed Mar 17 08:31:01 2021 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ pcmk-1 ]
  * GuestOnline: [ guest1@pcmk-1 ]

Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain):  Started pcmk-1
```

Now, you could place a resource, such as a webserver, on **guest1**:

```
# pcs resource create webserver apache params configfile=/etc/httpd/conf/httpd.conf op monitor␣
↪interval=30s
# pcs constraint location webserver prefers guest1
```

Now, the crm_mon output would show:

```
Cluster name: mycluster

Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.0.5-8.el8-ba59be7122) - partition with quorum
  * Last updated: Wed Mar 17 08:38:37 2021
  * Last change:  Wed Mar 17 08:35:01 2021 by root via cibadmin on pcmk-1
  * 2 nodes configured
  * 3 resource instances configured

Node List:
  * Online: [ pcmk-1 ]
  * GuestOnline: [ guest1@pcmk-1 ]

Full List of Resources:
  * vm-guest1       (ocf::heartbeat:VirtualDomain): Started pcmk-1
  * webserver       (ocf::heartbeat::apache):       Started guest1
```

It is worth noting that after **guest1** is integrated into the cluster, nearly all the Pacemaker command-line tools immediately become available to the guest node. This means things like `crm_mon`, `crm_resource`, and `crm_attribute` will work natively on the guest node, as long as the connection between the guest node and a cluster node exists. This is particularly important for any promotable clone resources executing on the guest node that need access to `crm_attribute` to set promotion scores.

2.3.7 Mile-High View of Configuration Steps

The command used in *Integrate Guest Node into Cluster* does multiple things. If you'd like to each part manually, you can do so as follows. You'll see that the end result is the same:

- Later, we are going to put the same authentication key with the path `/etc/pacemaker/authkey` on every cluster node and on every virtual machine. This secures remote communication.

Run this command on your cluster node if you want to make a somewhat random key:

```
# dd if=/dev/urandom of=/etc/pacemaker/authkey bs=4096 count=1
```

- To create the VirtualDomain resource agent for the management of the virtual machine, Pacemaker requires the virtual machine's xml config file to be dumped to a file – which we can name as we'd like – on disk. We named our virtual machine guest1; for this example, we'll dump to the file /etc/pacemaker/guest1.xml

```
# virsh dumpxml guest1 > /etc/pacemaker/guest1.xml
```

- Install pacemaker_remote on the virtual machine, and if a local firewall is used, allow the node to accept connections on TCP port 3121.

```
# yum install pacemaker-remote resource-agents
# firewall-cmd --add-port 3121/tcp --permanent
```

Note: If you just want to see this work, you may want to simply disable the local firewall and put SELinux in permissive mode while testing. This creates security risks and should not be done on a production machine exposed to the Internet, but can be appropriate for a protected test machine.

- On a cluster node, create a Pacemaker VirtualDomain resource to launch the virtual machine.

```
[root@pcmk-1 ~]# pcs resource create vm-guest1 VirtualDomain hypervisor="qemu:///system"␣
↪config="vm-guest1.xml" meta
Assumed agent name 'ocf:heartbeat:VirtualDomain' (deduced from 'VirtualDomain')
```

- Now use the following command to convert the VirtualDomain resource into a guest node which we'll name guest1. By doing so, the /etc/pacemaker/authkey will get copied to the guest node and the pacemaker_remote daemon will get started and enabled on the guest node as well.

```
[root@pcmk-1 ~]# pcs cluster node add-guest guest1 vm-guest1
No addresses specified for host 'guest1', using 'guest1'
Sending 'pacemaker authkey' to 'guest1'
guest1: successful distribution of the file 'pacemaker authkey'
Requesting 'pacemaker_remote enable', 'pacemaker_remote start' on 'guest1'
guest1: successful run of 'pacemaker_remote enable'
guest1: successful run of 'pacemaker_remote start'
```

- This will create CIB XML similar to the following:

```
<primitive class="ocf" id="vm-guest1" provider="heartbeat" type="VirtualDomain">
  <meta_attributes id="vm-guest1-meta_attributes">
    <nvpair id="vm-guest1-meta_attributes-remote-addr" name="remote-addr" value="guest1
↪"/>
    <nvpair id="vm-guest1-meta_attributes-remote-node" name="remote-node" value="guest1
↪"/>
  </meta_attributes>
  <instance_attributes id="vm-guest1-instance_attributes">
    <nvpair id="vm-guest1-instance_attributes-config" name="config" value="vm-guest1.xml
↪"/>
    <nvpair id="vm-guest1-instance_attributes-hypervisor" name="hypervisor" value=
↪"qemu:///system"/>
  </instance_attributes>
  <operations>
```

(continues on next page)

(continued from previous page)

```
    <op id="vm-guest1-migrate_from-interval-0s" interval="0s" name="migrate_from"␣
↪timeout="60s"/>
    <op id="vm-guest1-migrate_to-interval-0s" interval="0s" name="migrate_to" timeout=
↪"120s"/>
    <op id="vm-guest1-monitor-interval-10s" interval="10s" name="monitor" timeout="30s"/
↪>
    <op id="vm-guest1-start-interval-0s" interval="0s" name="start" timeout="90s"/>
    <op id="vm-guest1-stop-interval-0s" interval="0s" name="stop" timeout="90s"/>
  </operations>
</primitive>
```

```
[root@pcmk-1 ~]# pcs resource status
  * vm-guest1 (ocf::heartbeat:VirtualDomain): Stopped

[root@pcmk-1 ~]# pcs resource config
 Resource: vm-guest1 (class=ocf provider=heartbeat type=VirtualDomain)
  Attributes: config=vm-guest1.xml hypervisor=qemu:///system
  Meta Attrs: remote-addr=guest1 remote-node=guest1
  Operations: migrate_from interval=0s timeout=60s (vm-guest1-migrate_from-interval-0s)
              migrate_to interval=0s timeout=120s (vm-guest1-migrate_to-interval-0s)
              monitor interval=10s timeout=30s (vm-guest1-monitor-interval-10s)
              start interval=0s timeout=90s (vm-guest1-start-interval-0s)
              stop interval=0s timeout=90s (vm-guest1-stop-interval-0s)
```

The cluster will attempt to contact the virtual machine's pacemaker_remote service at the hostname **guest1** after it launches.

Note: The ID of the resource creating the virtual machine (**vm-guest1** in the above example) 'must' be different from the virtual machine's uname (**guest1** in the above example). Pacemaker will create an implicit internal resource for the pacemaker_remote connection to the guest, named with the value of **remote-node**, so that value cannot be used as the name of any other resource.

2.3.8 Troubleshooting a Remote Connection

Note: This section should not be done when the guest is connected to the cluster.

Should connectivity issues occur, it can be worth verifying that the cluster nodes can contact the remote node on port 3121. Here's a trick you can use. Connect using ssh from each of the cluster nodes. The connection will get destroyed, but how it is destroyed tells you whether it worked or not.

If running the ssh command on one of the cluster nodes results in this output before disconnecting, the connection works:

```
# ssh -p 3121 guest1
ssh_exchange_identification: read: Connection reset by peer
```

If you see one of these, the connection is not working:

```
# ssh -p 3121 guest1
ssh: connect to host guest1 port 3121: No route to host
```

```
# ssh -p 3121 guest1
ssh: connect to host guest1 port 3121: Connection refused
```

If you see this, then the connection is working, but port 3121 is attached to SSH, which it should not be.

```
# ssh -p 3121 guest1
kex_exchange_identification: banner line contains invalid characters
```

Once you can successfully connect to the guest from the host, you may shutdown the guest. Pacemaker will be managing the virtual machine from this point forward.

2.4 Remote Node Walk-through

What this tutorial is: An in-depth walk-through of how to get Pacemaker to integrate a remote node into the cluster as a node capable of running cluster resources.

What this tutorial is not: A realistic deployment scenario. The steps shown here are meant to get users familiar with the concept of remote nodes as quickly as possible.

2.4.1 Configure Cluster Nodes

This walk-through assumes you already have a Pacemaker cluster configured. For examples, we will use a cluster with two cluster nodes named pcmk-1 and pcmk-2. You can substitute whatever your node names are, for however many nodes you have. If you are not familiar with setting up basic Pacemaker clusters, follow the walk-through in the Clusters From Scratch document before attempting this one.

You will need to add the remote node's hostname (we're using **remote1** in this tutorial) to the cluster nodes' `/etc/hosts` files if you haven't already. This is required unless you have DNS set up in a way where remote1's address can be discovered.

Execute the following on each cluster node, replacing the IP address with the actual IP address of the remote node.

```
# cat << END >> /etc/hosts
192.168.122.10    remote1
END
```

2.4.2 Configure Remote Node

Configure Firewall on Remote 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 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:

```
# setenforce 0
# sed -i.bak "s/SELINUX=enforcing/SELINUX=permissive/g" /etc/selinux/config
# systemctl mask firewalld.service
# systemctl stop firewalld.service
```

Configure pacemaker_remote on Remote Node

Install the pacemaker_remote daemon on the remote node.

```
# yum install -y pacemaker-remote resource-agents pcs
```

Integrate Remote Node into Cluster

Integrating a remote node into the cluster is achieved through the creation of a remote node connection resource. The remote node connection resource both establishes the connection to the remote node and defines that the remote node exists. Note that this resource is actually internal to Pacemaker's controller. A metadata file for this resource can be found in the `/usr/lib/ocf/resource.d/pacemaker/remote` file that describes what options are available, but there is no actual **ocf:pacemaker:remote** resource agent script that performs any work.

Before we integrate the remote node, we'll need to authorize it.

```
# pcs host auth remote1
```

Now, define the remote node connection resource to our remote node, **remote1**, using the following command on any cluster node. This command creates the ocf:pacemaker:remote resource, creates and copies the key, and enables pacemaker_remote.

```
# pcs cluster node add-remote remote1
```

That's it. After a moment you should see the remote node come online. The final `pcs status` output should look something like this, and you can see that it created the ocf:pacemaker:remote resource:

```
# pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync
  * Current DC: pcmk-1 (version 2.0.5-8.el8-ba59be7122) - partition with quorum
  * Last updated: Wed Mar  3 11:02:03 2021
  * Last change:  Wed Mar  3 11:01:57 2021 by root via cibadmin on pcmk-1
  * 3 nodes configured
  * 1 resource instance configured

Node List:
  * Online: [ pcmk-1 pcmk-2 ]
  * RemoteOnline: [ remote1 ]

Full List of Resources:
  * remote1 (ocf::pacemaker:remote):         Started pcmk-1
```

2.4.3 How pcs Configures the Remote

To see that it created the key and copied it to all cluster nodes and the guest, run:

```
# ls -l /etc/pacemaker
```

To see that it enables pacemaker_remote, run:

```
# systemctl status pacemaker_remote
  pacemaker_remote.service - Pacemaker Remote executor daemon
    Loaded: loaded (/usr/lib/systemd/system/pacemaker_remote.service; enabled; vendor preset:␣
↪disabled)
    Active: active (running) since Tue 2021-03-02 10:42:40 EST; 1min 23s ago
      Docs: man:pacemaker-remoted
            https://clusterlabs.org/pacemaker/doc/
 Main PID: 1139 (pacemaker-remot)
     Tasks: 1
    Memory: 5.4M
    CGroup: /system.slice/pacemaker_remote.service
             1139 /usr/sbin/pacemaker-remoted

Mar 02 10:42:40 remote1 systemd[1]: Started Pacemaker Remote executor daemon.
Mar 02 10:42:40 remote1 pacemaker-remoted[1139]:  notice: Additional logging available in /var/log/
↪pacemaker/pacemaker.log
Mar 02 10:42:40 remote1 pacemaker-remoted[1139]:  notice: Starting Pacemaker remote executor
Mar 02 10:42:41 remote1 pacemaker-remoted[1139]:  notice: Pacemaker remote executor successfully␣
↪started and accepting connections
```

2.4.4 Starting Resources on Remote Node

Once the remote node is integrated into the cluster, starting resources on a remote node is the exact same as on cluster nodes. Refer to the Clusters from Scratch document for examples of resource creation.

Warning: Never involve a remote node connection resource in a resource group, colocation constraint, or order constraint.

2.4.5 Fencing Remote Nodes

Remote nodes are fenced the same way as cluster nodes. No special considerations are required. Configure fencing resources for use with remote nodes the same as you would with cluster nodes.

Note, however, that remote nodes can never 'initiate' a fencing action. Only cluster nodes are capable of actually executing a fencing operation against another node.

2.4.6 Accessing Cluster Tools from a Remote Node

Besides allowing the cluster to manage resources on a remote node, pacemaker_remote has one other trick. The pacemaker_remote daemon allows nearly all the pacemaker tools (`crm_resource`, `crm_mon`, `crm_attribute`, etc.) to work on remote nodes natively.

Try it: Run `crm_mon` on the remote node after pacemaker has integrated it into the cluster. These tools just work. These means resource agents such as promotable resources (which need access to tools like `crm_attribute`) work seamlessly on the remote nodes.

Higher-level command shells such as `pcs` may have partial support on remote nodes, but it is recommended to run them from a cluster node.

2.4.7 Troubleshooting a Remote Connection

Note: This section should not be done when the remote is connected to the cluster.

Should connectivity issues occur, it can be worth verifying that the cluster nodes can contact the remote node on port 3121. Here's a trick you can use. Connect using ssh from each of the cluster nodes. The connection will get destroyed, but how it is destroyed tells you whether it worked or not.

If running the ssh command on one of the cluster nodes results in this output before disconnecting, the connection works:

```
# ssh -p 3121 remote1
ssh_exchange_identification: read: Connection reset by peer
```

If you see one of these, the connection is not working:

```
# ssh -p 3121 remote1
ssh: connect to host remote1 port 3121: No route to host
```

```
# ssh -p 3121 remote1
ssh: connect to host remote1 port 3121: Connection refused
```

Once you can successfully connect to the remote node from the both cluster nodes, you may move on to setting up Pacemaker on the cluster nodes.

2.5 Alternative Configurations

These alternative configurations may be appropriate in limited cases, such as a test cluster, but are not the best method in most situations. They are presented here for completeness and as an example of Pacemaker's flexibility to suit your needs.

2.5.1 Virtual Machines as Cluster Nodes

The preferred use of virtual machines in a Pacemaker cluster is as a cluster resource, whether opaque or as a guest node. However, it is possible to run the full cluster stack on a virtual node instead.

This is commonly used to set up test environments; a single physical host (that does not participate in the cluster) runs two or more virtual machines, all running the full cluster stack. This can be used to simulate a larger cluster for testing purposes.

In a production environment, fencing becomes more complicated, especially if the underlying hosts run any services besides the clustered VMs. If the VMs are not guaranteed a minimum amount of host resources, CPU and I/O contention can cause timing issues for cluster components.

Another situation where this approach is sometimes used is when the cluster owner leases the VMs from a provider and does not have direct access to the underlying host. The main concerns in this case are proper

fencing (usually via a custom resource agent that communicates with the provider's APIs) and maintaining a static IP address between reboots, as well as resource contention issues.

2.5.2 Virtual Machines as Remote Nodes

Virtual machines may be configured following the process for remote nodes rather than guest nodes (i.e., using an **ocf:pacemaker:remote** resource rather than letting the cluster manage the VM directly).

This is mainly useful in testing, to use a single physical host to simulate a larger cluster involving remote nodes. Pacemaker's Cluster Test Suite (CTS) uses this approach to test remote node functionality.

2.5.3 Containers as Guest Nodes

Containers and in particular Linux containers (LXC) and Docker, have become a popular method of isolating services in a resource-efficient manner.

The preferred means of integrating containers into Pacemaker is as a cluster resource, whether opaque or using Pacemaker's `bundle` resource type.

However, it is possible to run `pacemaker_remote` inside a container, following the process for guest nodes. This is not recommended but can be useful, for example, in testing scenarios, to simulate a large number of guest nodes.

The configuration process is very similar to that described for guest nodes using virtual machines. Key differences:

- The underlying host must install the libvirt driver for the desired container technology – for example, the `libvirt-daemon-lxc` package to get the libvirt-lxc driver for LXC containers.
- Libvirt XML definitions must be generated for the containers. The `pacemaker-cts` package includes a script for this purpose, `/usr/share/pacemaker/tests/cts/lxc_autogen.sh`. Run it with the `--help` option for details on how to use it. It is intended for testing purposes only, and hardcodes various parameters that would need to be set appropriately in real usage. Of course, you can create XML definitions manually, following the appropriate libvirt driver documentation.
- To share the authentication key, either share the host's `/etc/pacemaker` directory with the container, or copy the key into the container's filesystem.
- The **VirtualDomain** resource for a container will need **force_stop="true"** and an appropriate hypervisor option, for example **hypervisor="lxc:///"** for LXC containers.

CHAPTER

THREE

INDEX

- genindex
- search

INDEX