Pacemaker Remote

Scaling High Availablity Clusters

, Written by the Pacemaker project contributors

Pacemaker Remote: Scaling High Availablity Clusters

by

Abstract

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

The example commands in this document will use:

1. CentOS 7.4 as the host operating system
2. Pacemaker Remote to perform resource management within guest nodes and remote nodes
3. KVM for virtualization
4. libvirt to manage guest nodes
5. Corosync to provide messaging and membership services on cluster nodes
6. Pacemaker 1.1.16 [1] to perform resource management on cluster nodes
7. 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.

[1] While this guide is part of the document set for Pacemaker 2.0, it demonstrates the version available in the standard CentOS repositories

Table of Contents

List of Figures

List of Tables

Chapter 1. Scaling a Pacemaker Cluster

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.

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_remote	A small service daemon that allows a host to be used as a Pacemaker node without running the full cluster stack. Nodes running pacemaker_remote 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_remote daemon is an enhanced version of Pacemaker's local resource management daemon (LRMD).
remote node	A physical host running pacemaker_remote. Remote nodes have a special resource that manages communication with the cluster. This is sometimes referred to as the *baremetal* case.
guest node	A virtual host running pacemaker_remote. 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 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.

[1] See the Pacemaker documentation [http://www.clusterlabs.org/doc/], especially *Clusters From Scratch* and *Pacemaker Explained*, for basic information about high-availability using Pacemaker

- 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_remote to make it a guest node, allowing the cluster to manage services inside it. The virtual machine is *transparent* to the cluster.

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_remote, 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_remote:

- The physical hosts are cluster nodes (running the full cluster stack).
- The virtual machines are guest nodes (running the pacemaker_remote service). Nearly zero configuration is required on the virtual machine.
- The cluster stack on the cluster nodes launches the virtual machines and immediately connects to the pacemaker_remote service 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_remote 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_remote can scale to these lengths and possibly much further depending on the specific scenario.

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.

Expanding the Cluster Stack

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

Figure 1.1. Traditional HA Stack

Figure 1.2. HA Stack With Guest Nodes

Chapter 2. Guest Node Quick Example

If you already know how to use Pacemaker, you’ll likely be able to grasp this new concept of guest nodes by reading through this quick example without having to sort through all the detailed walk-through steps. Here are the key configuration ingredients that make this possible using libvirt and KVM virtual guests. These steps strip everything down to the very basics.

Mile-High View of Configuration Steps

- Give each virtual machine that will be used as a guest node a static network address and unique hostname.

- Put the same authentication key with the path `/etc/pacemaker/authkey` on every cluster node and virtual machine. This secures remote communication.

  Run this command if you want to make a somewhat random key:

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

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

  ```
  yum install pacemaker-remote resource-agents
  systemctl enable pacemaker_remote
  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.

- Create a Pacemaker resource to launch each virtual machine, using the **remote-node** meta-attribute to let Pacemaker know this will be a guest node capable of running resources.

  ```
  # pcs resource create vm-guest1 VirtualDomain hypervisor="qemu:///system" config=
  ```

  The above command will create CIB XML similar to the following:

  ```
        <primitive class="ocf" id="vm-guest1" provider="heartbeat" type="VirtualDom
          <instance_attributes id="vm-guest-instance_attributes">
            <nvpair id="vm-guest1-instance_attributes-hypervisor" name="hypervisor"
            <nvpair id="vm-guest1-instance_attributes-config" name="config" value="
          </instance_attributes>
          <operations>
            <op id="vm-guest1-interval-30s" interval="30s" name="monitor"/>
          </operations>
          <meta_attributes id="vm-guest1-meta_attributes">
            <nvpair id="vm-guest1-meta_attributes-remote-node" name="remote-node" v
          </meta_attributes>
        </primitive>
  ```

In the example above, the meta-attribute **remote-node="guest1"** tells Pacemaker that this resource is a guest node with the hostname **guest1**. 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.

Using a Guest Node

Guest nodes will show up in `crm_mon` output as normal:

Example `crm_mon` output after guest1 is integrated into cluster.

```
Stack: corosync
Current DC: node1 (version 1.1.16-12.el7_4.5-94ff4df) - partition with quorum
Last updated: Fri Jan 12 13:52:39 2018
Last change: Fri Jan 12 13:25:17 2018 via pacemaker-controld on node1

2 nodes configured
2 resources configured

Online: [ node1 guest1]

vm-guest1     (ocf::heartbeat:VirtualDomain): Started node1
```

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

```
# pcs resource create webserver apache params configfile=/etc/httpd/conf/httpd.con
# pcs constraint location webserver prefers guest1
```

Now, the crm_mon output would show:

```
Stack: corosync
Current DC: node1 (version 1.1.16-12.el7_4.5-94ff4df) - partition with quorum
Last updated: Fri Jan 12 13:52:39 2018
Last change: Fri Jan 12 13:25:17 2018 via pacemaker-controld on node1

2 nodes configured
2 resources configured

Online: [ node1 guest1]

vm-guest1     (ocf::heartbeat:VirtualDomain): Started node1
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_master` to set transient attributes.

Chapter 3. 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-based nodes.

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 metadata parameters.

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 metadata options available to enable a resource as a guest node and define its connection parameters.

Table 3.1. 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.

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. This can be an ip address or hostname.
  port: tcp port to connect to.
  reconnect_interval: Interval in seconds at which Pacemaker will attempt to
        reconnect to a remote node after an active connection to
        the remote node has been severed. When this value is
        nonzero, Pacemaker will retry the connection
        indefinitely, at the specified interval.
```

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.

Example defining a remote node with the hostname **remote1**:

```
# pcs resource create remote1 remote
```

Example defining a remote node to connect to a specific IP address and port:

```
# pcs resource create remote1 remote server=192.168.122.200 port=8938
```

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 [https://en.wikipedia.org/wiki/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_remote 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 a custom directory for finding the authkey.
PCMK_authkey_location=/etc/pacemaker/authkey
#
# Specify a custom port for Pacemaker Remote connections
PCMK_remote_port=3121
```

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.

Chapter 4. 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.

Configure the Physical Host

Note

For this example, we will use a single physical host named **example-host**. A production cluster would likely have multiple physical hosts, in which case you would run the commands here on each one, unless noted otherwise.

Configure Firewall on Host

On the physical host, 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/sel
[root@pcmk-1 ~]# systemctl mask firewalld.service
[root@pcmk-1 ~]# systemctl stop firewalld.service
[root@pcmk-1 ~]# iptables --flush
```

Install Cluster Software

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

Configure Corosync

Corosync handles pacemaker's cluster membership and messaging. The corosync config file is located in `/etc/corosync/corosync.conf`. That config file must be initialized with information about the cluster nodes before pacemaker can start.

To initialize the corosync config file, execute the following `pcs` command, replacing the cluster name and hostname as desired:

```
# pcs cluster setup --force --local --name mycluster example-host
```

Note

If you have multiple physical hosts, you would execute the setup command on only one host, but list all of them at the end of the command.

Configure Pacemaker for Remote Node Communication

Create a place to hold an authentication key for use with pacemaker_remote:

```
# mkdir -p --mode=0750 /etc/pacemaker
# chgrp haclient /etc/pacemaker
```

Generate a key:

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

Note

If you have multiple physical hosts, you would generate the key on only one host, and copy it to the same location on all hosts.

Verify Cluster Software

Start the cluster

```
# pcs cluster start
```

Verify corosync membership

```
# pcs status corosync

Membership information
----------------------
    Nodeid      Votes Name
         1          1 example-host (local)
```

Verify pacemaker status. At first, the output will look like this:

```
# pcs status
Cluster name: mycluster
WARNING: no stonith devices and stonith-enabled is not false
Stack: corosync
Current DC: NONE
Last updated: Fri Jan 12 15:18:32 2018
Last change: Fri Jan 12 12:42:21 2018 by root via cibadmin on example-host

1 node configured
0 resources configured

Node example-host: UNCLEAN (offline)
```

```
No active resources

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

After a short amount of time, you should see your host as a single node in the cluster:

```
# pcs status
Cluster name: mycluster
WARNING: no stonith devices and stonith-enabled is not false
Stack: corosync
Current DC: example-host (version 1.1.16-12.el7_4.5-94ff4df) - partition WITHOUT q
Last updated: Fri Jan 12 15:20:05 2018
Last change: Fri Jan 12 12:42:21 2018 by root via cibadmin on example-host

1 node configured
0 resources configured

Online: [ example-host ]

No active resources

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

Disable STONITH and Quorum

Now, enable the cluster to work without quorum or stonith. This is required for the sake of getting this tutorial to work with a single cluster node.

```
# pcs property set stonith-enabled=false
# pcs property set no-quorum-policy=ignore
```

Warning

The use of `stonith-enabled=false` 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 STONITH disabled. We disable STONITH here only to focus the discussion on pacemaker_remote, and to be able to use a single physical host in the example.

Now, the status output should look similar to this:

```
# pcs status
Cluster name: mycluster
Stack: corosync
Current DC: example-host (version 1.1.16-12.el7_4.5-94ff4df) - partition with quor
Last updated: Fri Jan 12 15:22:49 2018
Last change: Fri Jan 12 15:22:46 2018 by root via cibadmin on example-host

1 node configured
```

```
0 resources configured

Online: [ example-host ]

No active resources

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

Go ahead and stop the cluster for now after verifying everything is in order.

```
# pcs cluster stop --force
```

Install Virtualization Software

```
# yum install -y kvm libvirt qemu-system qemu-kvm bridge-utils virt-manager
# systemctl enable libvirtd.service
```

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.

Configure the KVM guest

Create Guest

We will not outline here the installation steps required to create a KVM guest. There are plenty of tutorials available elsewhere that do that. Just 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).

Configure Firewall on Guest

On each guest, allow cluster-related services through the local firewall, following the same procedure as in the section called “Configure Firewall on Host”.

Verify Connectivity

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

Configure pacemaker_remote

Install pacemaker_remote, and enable it to run at start-up. 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 pacemaker-remote resource-agents
# systemctl enable pacemaker_remote.service
```

Copy the authentication key from a host:

```
# mkdir -p --mode=0750 /etc/pacemaker
# chgrp haclient /etc/pacemaker
# scp root@example-host:/etc/pacemaker/authkey /etc/pacemaker
```

Start pacemaker_remote, and verify the start was successful:

```
# systemctl start pacemaker_remote
# systemctl status pacemaker_remote

  pacemaker_remote.service - Pacemaker Remote Service
   Loaded: loaded (/usr/lib/systemd/system/pacemaker_remote.service; enabled)
   Active: active (running) since Thu 2013-03-14 18:24:04 EDT; 2min 8s ago
 Main PID: 1233 (pacemaker_remot)
   CGroup: name=systemd:/system/pacemaker_remote.service
    ##1233 /usr/sbin/pacemaker-remoted

  Mar 14 18:24:04 guest1 systemd[1]: Starting Pacemaker Remote Service...
  Mar 14 18:24:04 guest1 systemd[1]: Started Pacemaker Remote Service.
  Mar 14 18:24:04 guest1 pacemaker-remoted[1233]: notice: lrmd_init_remote_tls_ser
```

Verify Host Connection to Guest

Before moving forward, it's worth verifying that the host can contact the guest on port 3121. Here's a trick you can use. Connect using ssh from the host. The connection will get destroyed, but how it is destroyed tells you whether it worked or not.

First add guest1 to the host machine's `/etc/hosts` file if you haven't already. This is required unless you have DNS setup in a way where guest1's address can be discovered.

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

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
```

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

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. The output of `pcs status` should look as it did in the section called “Disable STONITH and Quorum”.

Integrate as Guest Node

If you didn’t already do this earlier in the verify host to guest connection section, add the KVM guest’s IP address to the host’s `/etc/hosts` file so we can connect by hostname. For this example:

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

We will use the **VirtualDomain** resource agent for the management of the virtual machine. This agent requires the virtual machine’s XML config to be dumped to a file on disk. To do this, pick out the name of the virtual machine you just created from the output of this list.

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     guest1                         shut off
```

In my case I named it guest1. Dump the xml to a file somewhere on the host using the following command.

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

Now just register the resource with pacemaker and you’re set!

```
# pcs resource create vm-guest1 VirtualDomain hypervisor="qemu:///system" \
    config="/etc/pacemaker/guest1.xml" meta remote-node=guest1
```

Note

This example puts the guest XML under /etc/pacemaker because the permissions and SELinux labeling should not need any changes. If you run into trouble with this or any step, try disabling SELinux with `setenforce 0`. If it works after that, see SELinux documentation for how to troubleshoot, if you wish to reenable SELinux.

Note

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

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.

```
# pcs status
Cluster name: mycluster
Stack: corosync
Current DC: example-host (version 1.1.16-12.el7_4.5-94ff4df) - partition with quor
Last updated: Fri Jan 12 18:00:45 2018
Last change: Fri Jan 12 17:53:44 2018 by root via crm_resource on example-host

2 nodes configured
2 resources configured

Online: [ example-host ]
GuestOnline: [ guest1@example-host ]

Full list of resources:

 vm-guest1 (ocf::heartbeat:VirtualDomain): Started example-host

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

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 example-host
 FAKE1 (ocf::pacemaker:Dummy): Started guest1
 FAKE2 (ocf::pacemaker:Dummy): Started guest1
 FAKE3 (ocf::pacemaker:Dummy): Started example-host
 FAKE4 (ocf::pacemaker:Dummy): Started guest1
 FAKE5 (ocf::pacemaker:Dummy): Started example-host
```

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 example-host
 FAKE1 (ocf::pacemaker:Dummy): Started guest1
 FAKE2 (ocf::pacemaker:Dummy): Started guest1
 FAKE3 (ocf::pacemaker:Dummy): Started guest1
 FAKE4 (ocf::pacemaker:Dummy): Started example-host
 FAKE5 (ocf::pacemaker:Dummy): Started example-host
```

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
Stack: corosync
Current DC: example-host (version 1.1.16-12.el7_4.5-94ff4df) - partition with quor
Last updated: Fri Jan 12 18:08:35 2018
Last change: Fri Jan 12 18:07:00 2018 by root via cibadmin on example-host

2 nodes configured
7 resources configured

Online: [ example-host ]

Full list of resources:

 vm-guest1 (ocf::heartbeat:VirtualDomain): Started example-host
 FAKE1 (ocf::pacemaker:Dummy): Stopped
 FAKE2 (ocf::pacemaker:Dummy): Stopped
 FAKE3 (ocf::pacemaker:Dummy): Stopped
 FAKE4 (ocf::pacemaker:Dummy): Started example-host
 FAKE5 (ocf::pacemaker:Dummy): Started example-host

Failed Actions:
* guest1_monitor_30000 on example-host 'unknown error' (1): call=8, status=Error,
    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
```

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.

```
Cluster name: mycluster
Stack: corosync
Current DC: example-host (version 1.1.16-12.el7_4.5-94ff4df) - partition with quor
Last updated: Fri Jan 12 18:18:30 2018
Last change: Fri Jan 12 18:07:00 2018 by root via cibadmin on example-host

2 nodes configured
7 resources configured

Online: [ example-host ]
GuestOnline: [ guest1@example-host ]

Full list of resources:

 vm-guest1 (ocf::heartbeat:VirtualDomain): Started example-host
 FAKE1 (ocf::pacemaker:Dummy): Started guest1
 FAKE2 (ocf::pacemaker:Dummy): Started guest1
 FAKE3 (ocf::pacemaker:Dummy): Started guest1
 FAKE4 (ocf::pacemaker:Dummy): Started example-host
 FAKE5 (ocf::pacemaker:Dummy): Started example-host

Failed Actions:
* guest1_monitor_30000 on example-host 'unknown error' (1): call=8, status=Error,
    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
```

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`, `crm_master`, 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_master`) 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.

Chapter 5. 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.

This tutorial requires three machines: two to act as cluster nodes, and a third to act as the remote node.

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 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:

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

Configure pacemaker_remote on Remote Node

Install the pacemaker_remote daemon on the remote node.

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

Create a location for the shared authentication key:

```
# mkdir -p --mode=0750 /etc/pacemaker
# chgrp haclient /etc/pacemaker
```

All nodes (both cluster nodes and remote nodes) must have the same authentication key installed for the communication to work correctly. If you already have a key on an existing node, copy it to the new remote node. Otherwise, create a new key, for example:

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

Now start and enable the pacemaker_remote daemon on the remote node.

```
# systemctl enable pacemaker_remote.service
# systemctl start pacemaker_remote.service
```

Verify the start is successful.

```
# systemctl status pacemaker_remote
pacemaker_remote.service - Pacemaker Remote Service
   Loaded: loaded (/usr/lib/systemd/system/pacemaker_remote.service; enabled)
   Active: active (running) since Fri 2018-01-12 15:21:20 CDT; 20s ago
 Main PID: 21273 (pacemaker_remot)
   CGroup: /system.slice/pacemaker_remote.service
           ##21273 /usr/sbin/pacemaker-remoted

Jan 12 15:21:20 remote1 systemd[1]: Starting Pacemaker Remote Service...
Jan 12 15:21:20 remote1 systemd[1]: Started Pacemaker Remote Service.
Jan 12 15:21:20 remote1 pacemaker-remoted[21273]: notice: crm_add_logfile: Additio
Jan 12 15:21:20 remote1 pacemaker-remoted[21273]: notice: lrmd_init_remote_tls_ser
Jan 12 15:21:20 remote1 pacemaker-remoted[21273]: notice: bind_and_listen: Listeni
```

Verify Connection to Remote Node

Before moving forward, it's 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.

First, 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
```

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, move on to setting up Pacemaker on the cluster nodes.

Configure Cluster Nodes

Configure Firewall on Cluster Nodes

On each cluster node, allow cluster-related services through the local firewall, following the same procedure as in the section called “Configure Firewall on Remote Node”.

Install Pacemaker on Cluster Nodes

On the two cluster nodes, install the following packages.

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

Copy Authentication Key to Cluster Nodes

Create a location for the shared authentication key, and copy it from any existing node:

```
# mkdir -p --mode=0750 /etc/pacemaker
# chgrp haclient /etc/pacemaker
# scp remote1:/etc/pacemaker/authkey /etc/pacemaker/authkey
```

Configure Corosync on Cluster Nodes

Corosync handles Pacemaker’s cluster membership and messaging. The corosync config file is located in `/etc/corosync/corosync.conf`. That config file must be initialized with information about the two cluster nodes before pacemaker can start.

To initialize the corosync config file, execute the following pcs command on both nodes, filling in the information in <> with your nodes' information.

```
# pcs cluster setup --force --local --name mycluster <node1 ip or hostname> <node2
```

Start Pacemaker on Cluster Nodes

Start the cluster stack on both cluster nodes using the following command.

```
# pcs cluster start
```

Verify corosync membership

```
# pcs status corosync
Membership information
----------------------
    Nodeid      Votes Name
         1          1 node1 (local)
```

Verify Pacemaker status. At first, the `pcs cluster status` output will look like this.

```
# pcs status
Cluster name: mycluster
Stack: corosync
Current DC: NONE
```

```
Last updated: Fri Jan 12 16:14:05 2018
Last change: Fri Jan 12 14:02:14 2018

1 node configured
0 resources configured
```

After about a minute, you should see your two cluster nodes come online.

```
# pcs status
Cluster name: mycluster
Stack: corosync
Current DC: node1 (version 1.1.16-12.el7_4.5-94ff4df) - partition with quorum
Last updated: Fri Jan 12 16:16:32 2018
Last change: Fri Jan 12 14:02:14 2018

2 nodes configured
0 resources configured

Online: [ node1 node2 ]
```

For the sake of this tutorial, we are going to disable stonith to avoid having to cover fencing device configuration.

```
# pcs property set stonith-enabled=false
```

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.

Define the remote node connection resource to our remote node, **remote1**, using the following command on any cluster node.

```
# pcs resource create remote1 ocf:pacemaker:remote
```

That's it. After a moment you should see the remote node come online.

```
Cluster name: mycluster
Stack: corosync
Current DC: node1 (version 1.1.16-12.el7_4.5-94ff4df) - partition with quorum
Last updated: Fri Jan 12 17:13:09 2018
Last change: Fri Jan 12 17:02:02 2018

3 nodes configured
1 resources configured

Online: [ node1 node2 ]
RemoteOnline: [ remote1 ]

Full list of resources:
 remote1 (ocf::pacemaker:remote): Started node1

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

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* [http://clusterlabs.org/doc/] document for examples of resource creation.

Warning

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

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.

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`, `crm_master`, 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_master`) 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.

Chapter 6. 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.

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.

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.

Containers as Guest Nodes

Containers,[1] 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 [http://libvirt.org/drvlxc.html] driver for LXC containers.

[1] https://en.wikipedia.org/wiki/Operating-system-level_virtualization

- 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.

Appendix A. Revision History

Revision History

Revision 1-0	Tue Mar 19 2013	DavidVossel`<davidvossel@gmail.com>`
Import from Pages.app		
Revision 2-0	Tue May 13 2013	DavidVossel`<davidvossel@gmail.com>`
Added Future Features Section		
Revision 3-0	Fri Oct 18 2013	DavidVossel`<davidvossel@gmail.com>`
Added Baremetal remote-node feature documentation		
Revision 4-0	Tue Aug 25 2015	KenGaillot`<kgaillot@redhat.com>`
Targeted CentOS 7.1 and Pacemaker 1.1.12+, updated for current terminology and practice		
Revision 5-0	Tue Dec 8 2015	KenGaillot`<kgaillot@redhat.com>`
Updated for Pacemaker 1.1.14		
Revision 6-0	Tue May 3 2016	KenGaillot`<kgaillot@redhat.com>`
Updated for Pacemaker 1.1.15		
Revision 7-0	Mon Oct 31 2016	KenGaillot`<kgaillot@redhat.com>`
Updated for Pacemaker 1.1.16		
Revision 7-1	Fri Jan 12 2018	KenGaillot`<kgaillot@redhat.com>`
Update banner for Pacemaker 2.0 and content for CentOS 7.4 with Pacemaker 1.1.16		
Revision 7-2	Mon Jan 29 2019	JanPokorný`<jpokorny@redhat.com>`
Minor reformatting		
Revision 7-3	Tue Oct 15 2019	KenGaillot`<kgaillot@redhat.com>`
Minor update for dynamic recheck interval		

Index