Pacemaker Remote

Extending High Availablity into Virtual Nodes

Edition 1

David Vossel
Primary author

Red Hat
dvossel@redhat.com

Legal Notice

Copyright © 2009-2013 David Vossel.

The text of and illustrations in this document are licensed under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA")[1].

In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

In addition to the requirements of this license, the following activities are looked upon favorably:

1. If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.
2. All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.
3. Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy or CD-ROM expression of the author(s) work.

Abstract

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

The KVM and Linux Container walk-through tutorials will use:

1. Fedora 18 as the host operating system
2. Pacemaker Remote to perform resource management within virtual nodes
3. libvirt to manage KVM and LXC virtual nodes
4. Corosync to provide messaging and membership services on the host nodes
5. Pacemaker to perform resource management on host nodes

Table of Contents

List of Tables

Chapter 1. Extending High Availability Cluster into Virtual Nodes

Table of Contents

1.1. Overview

The recent addition of the **pacemaker_remote** service supported by **Pacemaker version 1.1.10 and greater** allows nodes not running the cluster stack (pacemaker+corosync) to integrate into the cluster and have the cluster manage their resources just as if they were a real cluster node. This means that pacemaker clusters are now capable of managing both launching virtual environments (KVM/LXC) as well as launching the resources that live withing those virtual environments without requiring the virtual environments to run pacemaker or corosync.

1.2. Terms

cluster-node - A baremetal hardware node running the High Availability stack (pacemaker + corosync)

remote-node - A virtual guest node running the pacemaker_remote service.

pacemaker_remote - A service daemon capable of performing remote application management within virtual guests (kvm and lxc) in both pacemaker cluster environments and standalone (non-cluster) environments. This service is an enhanced version of pacemaker's local resource manage daemon (LRMD) that is capable of managing and monitoring LSB, OCF, upstart, and systemd resources on a guest remotely. It also allows for most of pacemaker's cli tools (crm_mon, crm_resource, crm_master, crm_attribute, ect..) to work natively on remote-nodes.

LXC - A Linux Container defined by the libvirt-lxc Linux container driver. http://libvirt.org/drvlxc.html

1.3. Virtual Machine Use Case

The use of pacemaker_remote in virtual machines solves a deployment scenario that has traditionally been difficult to solve.

"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."

In the past, users desiring this deployment had to make a decision. They would either have to sacrifice the ability of monitoring resources residing within virtual guests by running the cluster stack on the baremetal nodes, or run another cluster instance on the virtual guests where they potentially run into corosync scalability issues. There is a third scenario where the virtual guests run the cluster stack and join the same network as the baremetal nodes, but that can quickly hit issues with scalability as well.

With the pacemaker_remote service we have a new option.

- The baremetal cluster-nodes run the cluster stack (paceamaker+corosync).
- The virtual remote-nodes run the pacemaker_remote service (nearly zero configuration required on the virtual machine side)
- The cluster stack on the cluster-nodes launch the virtual machines and immediately connect to the pacemaker_remote service, allowing the virtual machines to integrate into the cluster just as if they were a real cluster-node.

The key difference here between the virtual machine remote-nodes and the cluster-nodes is that the remote-nodes are not running the cluster stack. This means the remote nodes will never become the DC, and they do not take place in quorum. On the hand this also means that the remote-nodes are not bound to the scalability limits associated with the cluster stack either. **No 16 node corosync member limits** to deal with. That isn't to say remote-nodes can scale indefinitely, but the expectation is that remote-nodes scale horizontally much further than cluster-nodes. Other than the quorum limitation, these remote-nodes behave just like cluster nodes in respects to resource management. The

cluster is fully capable of managing and monitoring resources on each remote-node. You can build constraints against remote-nodes, put them in standby, or whatever else you'd expect to be able to do with normal cluster-nodes. They even show up in the crm_mon output as you would expect cluster-nodes to.

To solidify the concept, an example cluster deployment integrating remote-nodes could look like this.

- 16 cluster-nodes running corosync+pacemaker stack.
- 64 pacemaker managed virtual machine resources running pacemaker_remote configured as remote-nodes.
- 64 pacemaker managed webserver and database resources configured to run on the 64 remote-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.

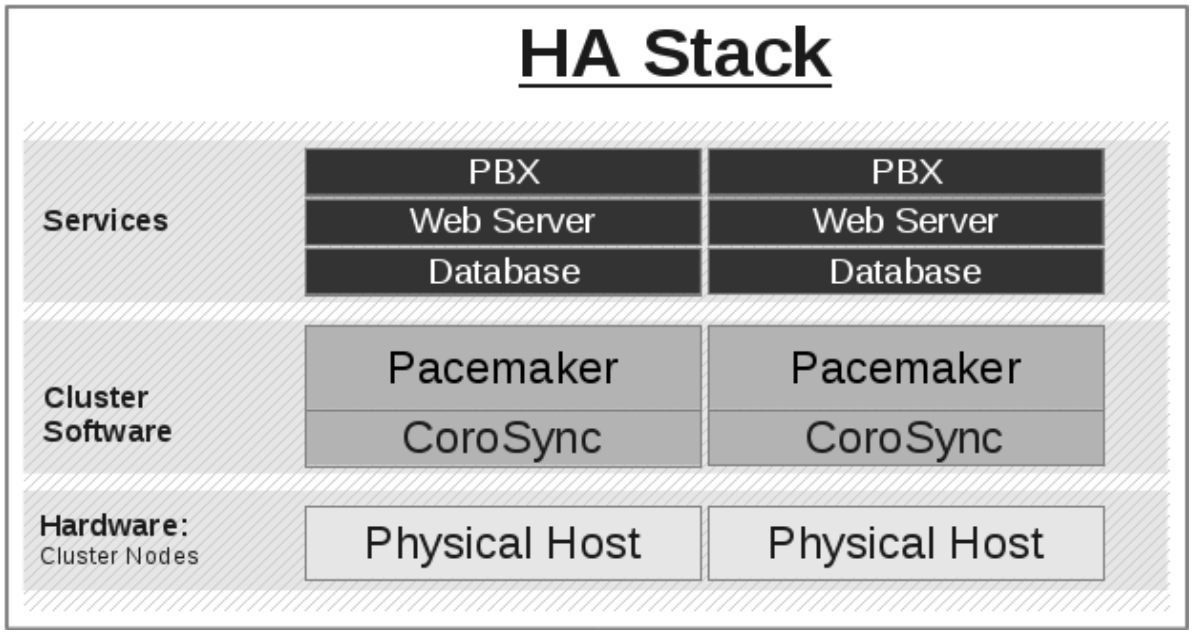
1.4. Linux Container Use Case

I want to isolate and limit the system resources (cpu, memory, filesystem) a cluster resource can consume without using virtual machines.

Using pacemaker_remote with Linux containers (libvirt-lxc) opens up some interesting possibilities for isolating resources in the cluster without the use of a hypervisor. We now have the ability to both define a contained environment with cpu and memory utilization limits and then assign resources to that contained environment all managed from within pacemaker. The LXC Walk-through section of this document outlines how pacemaker_remote can be used to bring Linux containers into the cluster as remote-nodes capable of executing resources.

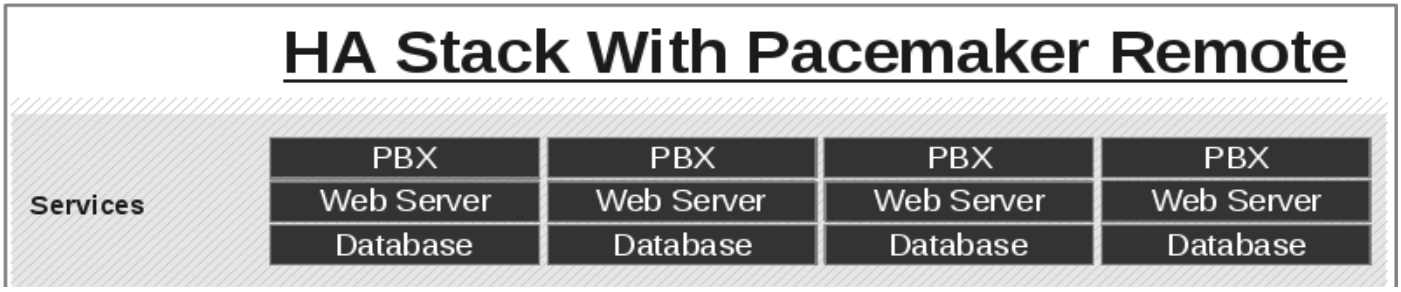
1.5. Expanding the Cluster Stack

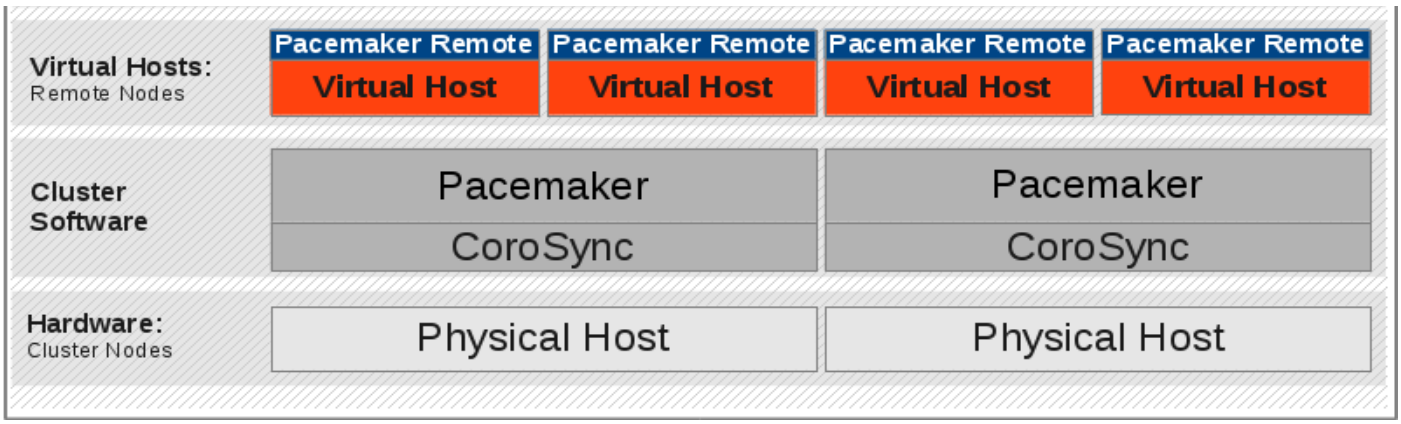
1.5.1. Traditional HA Stack



1.5.2. Remote-Node Enabled HA Stack

The stack grows one additional layer vertical so we can go further horizontal.




Virtual Hosts:
Remote Nodes
Pacemaker Remote
Virtual Host
Pacemaker Remote
Virtual Host
Pacemaker Remote
Virtual Host
Pacemaker Remote
Virtual Host
Cluster Software
Pacemaker
CoroSync
Pacemaker
CoroSync
Hardware:
Cluster Nodes
Physical Host
Physical Host

Chapter 2. Quick Example

Table of Contents

If you already know how to use pacemaker, you'll likely be able to grasp this new concept of remote-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.

2.1. Mile High View of Configuration Steps

- **Put an authkey with this path, /etc/pacemaker/authkey, on every cluster-node and virtual machine**. This secures remote communication and authentication.

Run this command if you want to make a somewhat random authkey.

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

- **Install pacemaker_remote packages every virtual machine, enable pacemaker_remote on startup, and poke hole in firewall for tcp port 3121.**

```
yum install pacemaker-remote resource-agents
systemctl enable pacemaker_remote
# If you just want to see this work, disable iptables and ip6tables on most distros.
# You may have to put selinux in permissive mode as well for the time being.
firewall-cmd --add-port 3121/tcp --permanent
```

- **Give each virtual machine a static network address and unique hostname**
- **Tell pacemaker to launch a virtual machine and that the virtual machine is a remote-node capable of running resources by using the "remote-node" meta-attribute.**

with pcs

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

raw xml

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

In the example above the meta-attribute *remote-node=guest1* tells pacemaker that this resource is a remote-node with the hostname *guest1* that is capable of being integrated into the cluster. The cluster will attempt to contact the virtual machine's pacemaker_remote service at the hostname *guest1* after it launches.

2.2. What those steps just did

Those steps just told pacemaker to launch a virtual machine called vm-guest1 and integrate that virtual machine as a remote-node called *guest1*.

Example crm_mon output after guest1 is integrated into cluster.

```
Last updated: Wed Mar 13 13:52:39 2013
Last change: Wed Mar 13 13:25:17 2013 via crmd on node1
Stack: corosync
Current DC: node1 (24815808) - partition with quorum
Version: 1.1.10
2 Nodes configured, unknown expected votes
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.conf op
monitor interval=30s
# pcs constraint webserver prefers guest1
```

Now the crm_mon output would show a webserver launched on the guest1 remote-node.

```
Last updated: Wed Mar 13 13:52:39 2013
Last change: Wed Mar 13 13:25:17 2013 via crmd on node1
Stack: corosync
Current DC: node1 (24815808) - partition with quorum
Version: 1.1.10
2 Nodes configured, unknown expected votes
2 Resources configured.

Online: [ node1 guest1]

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

2.3. Accessing Cluster from Remote-node

It is worth noting that after *guest1* is integrated into the cluster, all the pacemaker cli tools immediately become available to the remote node. This means things like crm_mon, crm_resource, and crm_attribute will work natively on the remote-node as long as the connection between the remote-node and cluster-node exists. This is particularly important for any master/slave resources executing on the remote-node that need access to crm_master to set the nodes transient attributes.

Chapter 3. Configuration Explained

Table of Contents

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 remote-nodes.

3.1. Resource Options

When configuring a virtual machine or lxc resource to act as a remote-node, these are the metadata options available to both enable the resource as a remote-node and define the connection parameters.

Table 3.1. Metadata Options for configurint KVM/LXC resources as remote-nodes

Option	Default	Description
remote-node	<none>	The name of the remote-node this resource defines. This both enables the resource as a remote-node and defines the unique name used to identify the remote-node. If no other parameters are set, this value will also be assumed as the hostname to connect to at port 3121. **WARNING** This value cannot overlap with any resource or node IDs.
remote-port	3121	Configure a custom port to use for the guest connection to pacemaker_remote.
remote-addr	**remote-node** value used as hostname	The ip address or hostname to connect to if remote-node's name is not the hostname of the guest.
remote-connect-timeout	60s	How long before a pending guest connection will time out.

3.2. Host and Guest Authentication

Authentication and encryption of the connection between cluster-nodes (pacemaker) to remote-nodes (pacemaker_remote) is achieved using TLS with PSK encryption/authentication on **tcp port 3121**. This means both the cluster-node and remote-node must share the same private key. By default this **key must be placed at "/etc/pacemaker/authkey" on both cluster-nodes and remote-nodes**.

3.3. Pacemaker and pacemaker_remote Options

If you need to change the default port or authkey location for either pacemaker or pacemaker_remote, there are environment variables you can set that affect both of those daemons. These environment variables can be enabled by placing them in the /etc/sysconfig/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
```

Chapter 4. KVM Walk-through

Table of Contents

What this tutorial is: 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 remote-node.

What this tutorial is not: 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.

4.1. Step 1: Setup the Host

This tutorial was created using Fedora 18 on the host and guest nodes. Anything that is capable of running libvirt and pacemaker v1.1.10 or greater will do though. An installation guide for installing Fedora 18 can be found here, http://docs.fedoraproject.org/en-US/Fedora/18/html/Installation_Guide/.

Fedora 18 (or similar distro) host preparation steps.

4.1.1. SElinux and Firewall

In order to simply this tutorial we will disable the selinux and the firewall on the host. **WARNING:** These actions will open a significant security threat to machines exposed to the outside world.

```
# setenforce 0
# sed -i.bak "s/SELINUX=enforcing/SELINUX=permissive/g" /etc/selinux/config
# systemctl disable iptables.service
# systemctl disable ip6tables.service
# rm '/etc/systemd/system/basic.target.wants/iptables.service'
# rm '/etc/systemd/system/basic.target.wants/ip6tables.service'
# systemctl stop iptables.service
# systemctl stop ip6tables.service
```

4.1.2. Install Cluster Software

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

4.1.3. Setup Corosync

Running the command below will attempt to detect the network address corosync should bind to.

```
# export corosync_addr=`ip addr | grep "inet " | tail -n 1 | awk '{print $4}' | sed
s/255/0/g`
```

Display and verify that address is correct

```
# echo $corosync_addr
```

In many cases the address will be 192.168.1.0 if you are behind a standard home router.

Now copy over the example corosync.conf. This code will inject your bindaddress and enable the vote quorum api which is required by pacemaker.

```
# cp /etc/corosync/corosync.conf.example /etc/corosync/corosync.conf
# sed -i.bak "s/.*\tbindnetaddr:.*/bindnetaddr:\ $corosync_addr/g"
/etc/corosync/corosync.conf
# cat << END >> /etc/corosync/corosync.conf
quorum {
           provider: corosync_votequorum
           expected_votes: 2
}
END
```

4.1.4. Verify Cluster Software

Start the cluster

```
# pcs cluster start
```

Verify corosync membership

```
# pcs status corosync

Membership information
    Nodeid      Votes Name
1795270848          1 example-host (local)
```

Verify pacemaker status. At first the *pcs cluster status* output will look like this.

```
# pcs status

 Last updated: Thu Mar 14 12:26:00 2013
 Last change: Thu Mar 14 12:25:55 2013 via crmd on example-host
 Stack: corosync
 Current DC:
 Version: 1.1.10
 1 Nodes configured, unknown expected votes
 0 Resources configured.
```

After about a minute you should see your host as a single node in the cluster.

```
# pcs status

 Last updated: Thu Mar 14 12:28:23 2013
 Last change: Thu Mar 14 12:25:55 2013 via crmd on example-host
 Stack: corosync
 Current DC: example-host (1795270848) - partition WITHOUT quorum
 Version: 1.1.8-9b13ea1
 1 Nodes configured, unknown expected votes
 0 Resources configured.

 Online: [ example-host ]
```

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

```
# pcs cluster stop
```

4.1.5. Install Virtualization Software

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

reboot the host

4.2. Step2: Create the KVM guest

I am not going to outline the installation steps required to create a kvm guest. There are plenty of tutorials available elsewhere that do that. I recommend using a Fedora 18 or greater distro as your guest as that is what I am testing this

tutorial with.

4.2.1. Setup Guest Network

Run the commands below to set up a static ip address (192.168.122.10) and hostname (guest1).

```
export remote_hostname=guest1
export remote_ip=192.168.122.10
export remote_gateway=192.168.122.1

yum remove -y NetworkManager

rm -f /etc/hostname
cat << END >> /etc/hostname
$remote_hostname
END

hostname $remote_hostname

cat << END >> /etc/sysconfig/network
HOSTNAME=$remote_hostname
GATEWAY=$remote_gateway
END

sed -i.bak "s/.*BOOTPROTO=.*/BOOTPROTO=none/g" /etc/sysconfig/network-scripts/ifcfg-eth0

cat << END >> /etc/sysconfig/network-scripts/ifcfg-eth0
IPADDR0=$remote_ip
PREFIX0=24
GATEWAY0=$remote_gateway
DNS1=$remote_gateway
END

systemctl restart network
systemctl enable network.service
systemctl enable sshd
systemctl start sshd

echo "checking connectivity"
ping www.google.com
```

To simplify the tutorial we'll go ahead and disable selinux on the guest. We'll also need to poke a hole through the firewall on port 3121 (the default port for pacemaker_remote) so the host can contact the guest.

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

# firewall-cmd --add-port 3121/tcp --permanent
```

If you still encounter connection issues just disable iptables and ipv6tables on the guest like we did on the host to guarantee you'll be able to contact the guest from the host.

At this point you should be able to ssh into the guest from the host.

4.2.2. Setup Pacemaker Remote

On the **HOST** machine run these commands to generate an authkey and copy it to the /etc/pacemaker folder on both the host and guest.

```
# mkdir /etc/pacemaker
# dd if=/dev/urandom of=/etc/pacemaker/authkey bs=4096 count=1
# scp -r /etc/pacemaker root@192.168.122.10:/etc/
```

Now on the **GUEST** install pacemaker-remote package and enable the daemon to run at startup. In the commands below you will notice the *pacemaker* and *pacemaker_remote* packages are being installed. The *pacemaker* package is not required. The only reason it is being installed for this tutorial is because it contains the a *Dummy* resource agent we will be using later on to test the remote-node.

```
# yum install -y pacemaker paceamaker-remote resource-agents
# systemctl enable pacemaker_remote.service
```

Now start pacemaker_remote on the guest and verify the start was successful.

```
# systemctl start pacemaker_remote.service

# 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_server:
Starting a tls listener on port 3121.
```

4.2.3. Verify Host Connection to Guest

Before moving forward it's worth going ahead and verifying the host can contact the guest on port 3121. Here's a trick you can use. Connect using telnet 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 telnet command on the host results in this output before disconnecting, the connection works.

```
# telnet guest1 3121
 Trying 192.168.122.10...
 Connected to guest1.
 Escape character is '^]'.
 Connection closed by foreign host.
```

If you see this, the connection is not working.

```
# telnet guest1 3121
Trying 192.168.122.10...
telnet: connect to address 192.168.122.10: No route to host
```

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.

4.3. Step3: Integrate KVM guest into Cluster.

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

4.3.1. 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 similar to this after about a minute.

```
Last updated: Thu Mar 14 16:41:22 2013
Last change: Thu Mar 14 16:41:08 2013 via crmd on example-host
Stack: corosync
Current DC: example-host (1795270848) - partition WITHOUT quorum
Version: 1.1.10
1 Nodes configured, unknown expected votes
0 Resources configured.


Online: [ example-host ]
```

Now enable the cluster to work without quorum or stonith. This is required just 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
```

4.3.2. Integrate KVM Guest as remote-node

If you didn't already do this earlier in the verify host to guest connection section, add the KVM guest's ip to the host's /etc/hosts file so we can connect by hostname. The command below will do that if you used the same ip address I used earlier.

```
# 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 > /root/guest1.xml
```

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

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

```
Last updated: Fri Mar 15 09:30:30 2013
Last change: Thu Mar 14 17:21:35 2013 via cibadmin on example-host
Stack: corosync
Current DC: example-host (1795270848) - partition WITHOUT quorum
Version: 1.1.10
2 Nodes configured, unknown expected votes
2 Resources configured.


Online: [ example-host guest1 ]

Full list of resources:

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

4.3.3. Starting Resources on KVM Guest

The commands below demonstrate how resources can be executed on both the remote-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 got started on the cluster-node, and some started on the remote-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 remote-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 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
```

4.3.4. Testing Remote-node Recovery and Fencing

Pacemaker's policy engine is smart enough to know fencing remote-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`
```

After a few seconds or so you'll see this in your *pcs status* output. The *guest1* node will be show as offline as it is being recovered.

```
Last updated: Fri Mar 15 11:00:31 2013
Last change: Fri Mar 15 09:54:16 2013 via cibadmin on example-host
Stack: corosync
Current DC: example-host (1795270848) - partition WITHOUT quorum
Version: 1.1.10
2 Nodes configured, unknown expected votes
7 Resources configured.


Online: [ example-host ]
OFFLINE: [ guest1 ]

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 (node=example-host, call=3, rc=7, status=complete): not running
```

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.

```
Last updated: Fri Mar 15 11:03:17 2013
Last change: Fri Mar 15 09:54:16 2013 via cibadmin on example-host
Stack: corosync
Current DC: example-host (1795270848) - partition WITHOUT quorum
Version: 1.1.10
2 Nodes configured, unknown expected votes
7 Resources configured.


Online: [ example-host 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

Failed actions:
    guest1_monitor_30000 (node=example-host, call=3, rc=7, status=complete): not running
```

4.3.5. Accessing Cluster Tools from Remote-node

Besides just 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) to work on remote nodes natively.**

Try it, run **crm_mon** or **pcs status** on the guest after pacemaker has integrated the remote-node into the cluster. These tools just work. These means resource agents such as master/slave resources which need access to tools like crm_master work seamlessly on the remote-nodes.

Chapter 5. Linux Container (LXC) Walk-through

Table of Contents

What this tutorial is: This tutorial demonstrates how pacemaker_remote can be used with Linux containers (managed by libvirt-lxc) to run cluster resources in an isolated environment.

What this tutorial is not: This tutorial is not a realistic deployment scenario. The steps shown here are meant to introduce users to the concept of managing Linux container environments with Pacemaker.

5.1. Step 1: Setup LXC Host

This tutorial was tested with Fedora 18. Anything that is capable of running libvirt and pacemaker v1.1.10 or greater will do though. An installation guide for installing Fedora 18 can be found here, http://docs.fedoraproject.org/en-US/Fedora/18/html/Installation_Guide/.

Fedora 18 (or similar distro) host preparation steps.

5.1.1. SElinux and Firewall Rules

In order to simply this tutorial we will disable the selinux and the firewall on the host. WARNING: These actions pose a significant security issues to machines exposed to the outside world. Basically, just don't do this on your production system.

```
# setenforce 0
# sed -i.bak "s/SELINUX=enforcing/SELINUX=permissive/g" /etc/selinux/config
# firewall-cmd --add-port 3121/tcp --permanent

# systemctl disable iptables.service
# systemctl disable ip6tables.service
# rm '/etc/systemd/system/basic.target.wants/iptables.service'
# rm '/etc/systemd/system/basic.target.wants/ip6tables.service'
# systemctl stop iptables.service
# systemctl stop ip6tables.service
```

5.1.2. Install Cluster Software on Host

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

5.1.3. Configure Corosync

Running the command below will attempt to detect the network address corosync should bind to.

```
# export corosync_addr=`ip addr | grep "inet " | tail -n 1 | awk '{print $4}' | sed
s/255/0/g`
```

Display and verify the address is correct

```
# echo $corosync_addr
```

In most cases the address will be 192.168.1.0 if you are behind a standard home router.

Now copy over the example corosync.conf. This code will inject your bindaddress and enable the vote quorum api which is required by pacemaker.

```
# cp /etc/corosync/corosync.conf.example /etc/corosync/corosync.conf
# sed -i.bak "s/.*\tbindnetaddr:.*/bindnetaddr:\ $corosync_addr/g"
/etc/corosync/corosync.conf
# cat << END >> /etc/corosync/corosync.conf
quorum {
           provider: corosync_votequorum
           expected_votes: 2
}
END
```

5.1.4. Verify Cluster

Start the cluster

```
# pcs cluster start
```

Verify corosync membership

```
# pcs status corosync

Membership information
    Nodeid      Votes Name
1795270848          1 example-host (local)
```

Verify pacemaker status. At first the *pcs cluster status* output will look like this.

```
# pcs status

 Last updated: Thu Mar 14 12:26:00 2013
 Last change: Thu Mar 14 12:25:55 2013 via crmd on example-host
 Stack: corosync
 Current DC:
 Version: 1.1.10
 1 Nodes configured, unknown expected votes
 0 Resources configured.
```

After about a minute you should see your host as a single node in the cluster.

```
# pcs status

 Last updated: Thu Mar 14 12:28:23 2013
 Last change: Thu Mar 14 12:25:55 2013 via crmd on example-host
 Stack: corosync
 Current DC: example-host (1795270848) - partition WITHOUT quorum
 Version: 1.1.8-9b13ea1
 1 Nodes configured, unknown expected votes
 0 Resources configured.

 Online: [ example-host ]
```

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

```
# pcs cluster stop
```

5.2. Step 2: Setup LXC Environment

5.2.1. Install Libvirt LXC software

```
# yum install -y libvirt libvirt-daemon-lxc wget
# systemctl enable libvirtd
```

At this point, restart the host.

5.2.2. Generate Libvirt LXC domains

I've attempted to simply this tutorial by creating a script to auto generate the libvirt-lxc xml domain definitions.

Download the script to whatever directory you want the containers to live in. In this example I am using the /root/lxc/ directory.

```
# mkdir /root/lxc/
# cd /root/lxc/
# wget https://raw.github.com/davidvossel/pcmk-lxc-autogen/master/lxc-autogen
# chmod 755 lxc-autogen
```

Now execute the script.

```
# ./lxc-autogen
```

After executing the script you will see a bunch of directories and xml files are generated. Those xml files are the libvirt-lxc domain definitions, and the directories are used as some special mount points for each container. If you open up one of the xml files you'll be able to see how the cpu, memory, and filesystem resources for the container are defined. You can use the libvirt-lxc driver's documentation found here, http://libvirt.org/drvlxc.html, as a reference to help understand all the parts of the xml file. The lxc-autogen script is not complicated and is worth exploring in order to grasp how the environment is generated.

It is worth noting that this environment is dependent on use of libvirt's default network interface. Verify the commands below look the same as your environment. The default network address 192.168.122.1 should have been generated by automatically when you installed the virtualization software.

```
# virsh net-list
Name                 State      Autostart     Persistent
----------------------------------------------------------
default              active     yes           yes

# virsh net-dumpxml default | grep -e "ip address="
<ip address='192.168.122.1' netmask='255.255.255.0'>
```

5.2.3. Generate the Authkey

Generate the authkey used to secure connections between the host and the lxc guest pacemaker_remote instances. This is sort of a funny case because the lxc guests and the host will share the same key file in the /etc/pacemaker/ directory. If in a different deployment where the lxc guests do not share the host's /etc/pacemaker directory, this key will have to be copied into each lxc guest.

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

5.3. Step 3: Integrate LXC guests into Cluster.

5.3.1. Start Cluster

On the host, start pacemaker.

```
# pcs cluster start
```

Wait for the host to become the DC. The output of *pcs status* should look similar to this after about a minute.

```
Last updated: Thu Mar 14 16:41:22 2013
Last change: Thu Mar 14 16:41:08 2013 via crmd on example-host
Stack: corosync
Current DC: example-host (1795270848) - partition WITHOUT quorum
Version: 1.1.10
1 Nodes configured, unknown expected votes
0 Resources configured.


Online: [ example-host ]
```

Now enable the cluster to work without quorum or stonith. This is required just 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
```

5.3.2. Integrate LXC Guests as remote-nodes

If you ran the *lxc-autogen* script with default parameters, 3 lxc domain definitions were created as .xml files. If you used the same directory I used for the lxc environment, the config files will be located in /root/lxc. Replace the *config* parameters in the following pcs commands if yours should be different.

The pcs commands below each configure a lxc guest as a remote-node in pacemaker. Behind the scenes each lxc guest is launching an instance of pacemaker_remote allowing pacemaker to integrate the lxc guests as remote-nodes. The meta-attribute *remote-node=<node-name>* used in each command is what tells pacemaker that the lxc guest is both a resource and a remote-node capable of running resources. In this case, the *remote-node* attribute also indicates to pacemaker that it can contact each lxc's pacemaker_remote service by using the remote-node name as the hostname. If you look in the /etc/hosts/ file you will see entries for each lxc guest. These entries were auto-generated earlier by the *lxc-autogen* script.

```
# pcs resource create container1 VirtualDomain force_stop="true" hypervisor="lxc:///"
config="/root/lxc/lxc1.xml" meta remote-node=lxc1
# pcs resource create container2 VirtualDomain force_stop="true" hypervisor="lxc:///"
config="/root/lxc/lxc2.xml" meta remote-node=lxc2
# pcs resource create container3 VirtualDomain force_stop="true" hypervisor="lxc:///"
config="/root/lxc/lxc3.xml" meta remote-node=lxc3
```

After creating the container resources you *pcs status* should look like this.

```
Last updated: Mon Mar 18 17:15:46 2013
Last change: Mon Mar 18 17:15:26 2013 via cibadmin on guest1
Stack: corosync
Current DC: example-host (175810752) - partition WITHOUT quorum
Version: 1.1.10
4 Nodes configured, unknown expected votes
6 Resources configured.

Online: [ example-host lxc1 lxc2 lxc3 ]

Full list of resources:

 container3     (ocf::heartbeat:VirtualDomain): Started example-host
 container1     (ocf::heartbeat:VirtualDomain): Started example-host
 container2     (ocf::heartbeat:VirtualDomain): Started example-host
```

5.3.3. Starting Resources on LXC Guests

Now that the lxc guests are integrated into the cluster, lets generate some Dummy resources to run on them.

Dummy resources are real resource agents used just for testing purposes. They actually execute on the node 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
```

After creating the Dummy resources you will see that the resource got distributed among all the nodes. The *pcs status* output should look similar to this.

```
Last updated: Mon Mar 18 17:31:54 2013
Last change: Mon Mar 18 17:31:05 2013 via cibadmin on example-host
Stack: corosync
Current DC: example=host (175810752) - partition WITHOUT quorum
Version: 1.1.10
4 Nodes configured, unknown expected votes
11 Resources configured.


Online: [ example-host lxc1 lxc2 lxc3 ]

Full list of resources:

 container3     (ocf::heartbeat:VirtualDomain): Started example-host
 container1     (ocf::heartbeat:VirtualDomain): Started example-host
 container2     (ocf::heartbeat:VirtualDomain): Started example-host
 FAKE1  (ocf::pacemaker:Dummy): Started lxc1
 FAKE2  (ocf::pacemaker:Dummy): Started lxc2
 FAKE3  (ocf::pacemaker:Dummy): Started lxc3
 FAKE4  (ocf::pacemaker:Dummy): Started lxc1
 FAKE5  (ocf::pacemaker:Dummy): Started lxc2
```

To witness that Dummy agents are running within the lxc guests browse one of the lxc domain's filesystem folders. Each lxc guest has a custom mount point for the '/var/run/'directory, which is the location the Dummy resources write their state files to.

```
# ls lxc1-filesystem/var/run/
Dummy-FAKE4.state  Dummy-FAKE.state
```

If you are curious, take a look at lxc1.xml to see how the filesystem is mounted.

5.3.4. Testing LXC Guest Failure

You will be able to see each pacemaker_remoted process running in each lxc guest from the host machine.

```
# ps -A | grep -e pacemaker_remote*
 9142 pts/2    00:00:00 pacemaker_remot
10148 pts/4    00:00:00 pacemaker_remot
10942 pts/6    00:00:00 pacemaker_remot
```

In order to see how the cluster reacts to a failed lxc guest. Try killing one of the pacemaker_remote instances.

```
# kill -9 9142
```

After a few moments the lxc guest that was running that instance of pacemaker_remote will be recovered along with all the resources running within that container.

Revision History

Revision 1 **Tue Mar 19 2013** **David Vossel**

Import from Pages.app

Index