Table of Contents

Fencing/Stonith

Configuring fencing depends on the final deployment. The fencing operation relies on a script installed on the cluster node. Each cluster technology supports several kind of fencing mechanisms, but you can create your own if necessary.

Before configuring the fencing mechanism, it is a good practice to do a research about the available fencing script, to decide which fits better on a particular deployment and make sure it works before configuring it on the cluster.

The virtual lab is configured through *Vagrant*, and this application uses *VirtualBox* to create and manage the the VMs. *Vagrant* and *VirtualBox* are running on *Mac OS X*. For this particular deployment, using *VirtualBox* as a fence mechanism makes sense, as this will always be running in order to keep the VMs running.

This is similar to a *VMWare vSphere* environment where *vCenter* manages the VMs of the cluster nodes. For *VMWare*, *RedHat* provides a fencing mechanism called `fence_vmrare_soap`, which uses the *SOAP* Interface of *vCenter*.

Unfortunately, for *VirtualBox* there is no fencing agent, which means it should be created.

VirtualBox

RHEL/CentOS 6

There is no fencing device for *VirtualBox*, so I've created one in *Python* using *RedHat's Fencing API* (see Appending A). The script must be placed on `/usr/sbin`, and must have `0755` for its permissions.

The following command can list all the VMs with their *IDs* (`VBoxManage` is executed on the host machine, in my case *Mac OS X*):

```
$ /Applications/VirtualBox.app/Contents/MacOS/VBoxManage list vms
"opennms-cluster-onmssrv01" {5c467fe9-2611-4e5f-99f6-d1c6aa17527e}
"opennms-cluster-onmssrv02" {a379c5c0-2ce0-482a-be49-7237aa67f662}
"opennms-cluster-pgdbsrv01" {5ebfd17b-1325-4cd7-b2ee-ac3347e38306}
"opennms-cluster-pgdbsrv02" {b74dbaff-46ae-402b-bd50-5cebdfb81cfa}
"opennms-cluster-nfssrv01" {8379a446-d3a4-44a8-954c-408f0c236a20}
```

After creating the `/usr/sbin/fence_vbox` (own by root with `0755` for its permissions), you should be able to see the following:

```
[root@onmssrv01 ~]# /usr/sbin/fence_vbox -h
Usage:
fence_vbox [options]
Options:
   -o, --action=<action>          Action: status, reboot (default), off or on
   -a, --ip=<ip>                  IP address or hostname of fencing device
   -l, --username=<name>          Login name
   -p, --password=<password>      Login password or passphrase
   -S, --password-script=<script> Script to run to retrieve password
   -x, --ssh                      Use ssh connection
   -n, --plug=<id>                Physical plug number on device or
                                        name of virtual machine
   -k, --identity-file=<filename> Identity file (private key) for ssh
   -4, --inet4-only               Forces agent to use IPv4 addresses only
   -6, --inet6-only               Forces agent to use IPv6 addresses only
   -c, --command-prompt=<prompt>  Force command prompt
   -v, --verbose                  Verbose mode
   -D, --debug-file=<debugfile>   Debugging to output file
   -V, --version                  Output version information and exit
   -h, --help                     Display this help and exit
   --power-timeout <seconds>      Test X seconds for status change after ON/OFF
   --shell-timeout <seconds>      Wait X seconds for cmd prompt after issuing command
   --login-timeout <seconds>      Wait X seconds for cmd prompt after login
   --power-wait <seconds>         Wait X seconds after issuing ON/OFF
   --delay <seconds>              Wait X seconds before fencing is started
   --retry-on <attempts>          Count of attempts to retry power on
```

Here is a sample execution:

```
[root@onmssrv01 ~]# /usr/sbin/fence_vbox -a 192.168.205.1 -l vboxuser -p vboxpasswd -n opennms-cluster-
onmssrv01 -o status
Status: ON
```

On the above example, the *VirtualBox* application is running on 192.168.205.1, and the user called *vboxuser* (with password, *vboxpasswd*) has privileges to run the `VBoxManage` command on *MacOS X*, and it will retrieve the status of the VM called *opennms-cluster-onmssrv01*.

You can use either the name of the *VM* or its *UUID*:

```
[root@onmssrv01 ~]# /usr/sbin/fence_vbox -a 192.168.205.1 -l vboxuser -p vboxpasswd -n '5c467fe9-2611-
4e5f-99f6-d1c6aa17527e' -o status
Status: ON
```

After copying the file to all the cluster nodes and verify they work, configure the fencing mechanism in the cluster.

Add the fence device (using `virtualbox_dev` as the device's name):

```
[root@onmssrv01 ~]# ccs --addfencedev virtualbox_dev agent=fence_vbox ipaddr="192.168.205.1"
login="onms_user" passwd="onms_passwd"
```

Add a method to every cluster node (using `virtualbox_m` as the method's name):

```
[root@onmssrv01 ~]# ccs --addmethod virtualbox_m onmssrv01.local
Method virtualbox_m added to onmssrv01.local.
[root@onmssrv01 ~]# ccs --addmethod virtualbox_m onmssrv02.local
Method virtualbox_m added to onmssrv02.local.
```

Add the fence instance to each cluster node:

```
[root@onmssrv01 ~]# ccs --addfenceinst virtualbox_dev onmssrv01.local virtualbox_m port="opennms-cluster-
onmssrv01"
[root@onmssrv01 ~]# ccs --addfenceinst virtualbox_dev onmssrv02.local virtualbox_m port="opennms-cluster-
onmssrv02"
```

As an alternative, you can use the *UUIDs* in you want, instead of the *VM* names:

```
[root@onmssrv01 ~]# ccs --addfenceinst virtualbox_dev onmssrv01.local virtualbox_m port="5c467fe9-2611-
4e5f-99f6-d1c6aa17527e"
[root@onmssrv01 ~]# ccs --addfenceinst virtualbox_dev onmssrv02.local virtualbox_m port="a379c5c0-2ce0-
482a-be49-7237aa67f662"
```

The advantage of using the *UUID* is that the *VM* name can be changed, while the *UUID* should be fixed. Remember to use the command shown at the beginning of the section to obtain the *VM* names or *UUIDs*.

Propagate the cluster configuration and activate it:

```
[root@onmssrv01 ~]# ccs --sync --activate
```

To test the fencing, execute the following command (assuming that the `onms_svc` service is active on *onmssrv02*):

```
[root@onmssrv01 ~]# fence_node -vv onmssrv02.local
fence onmssrv02.local dev 0.0 agent fence_vbox result: success
agent args: port=a379c5c0-2ce0-482a-be49-7237aa67f662 nodename=onmssrv02.local agent=fence_vbox
ipaddr=10.0.1.9 login=agalue passwd=9224153
fence onmssrv02.local success
```

RHEL/CentOS 7

Similar to *RHEL/CentOS 6*, there is no fencing device for *VirtualBox*, so I've created one in *Python* using *RedHat's Fencing API* (See Appending A). The script must be placed on `/usr/sbin`, and must have `0755` for its permissions.

The script `fence_vbox` is slightly different than the one used on *RHEL/CentOS 6* because the *API* was changed, so keep in mind they are not interchangeable. Make sure you use the correct one (see Appending B).

After copying the file to all the cluster nodes at `/usr/sbin`, and verify they work, configure the fencing mechanism in the cluster.

The following command can be used to get the list of parameters of a given fencing script:

```
[root@onmssrv11 ~]# pcs stonith list | grep vbox
fence_vbox - Fence agent for VirtualBox over SSH

[root@onmssrv11 ~]# pcs stonith describe fence_vbox
Stonith options for: fence_vbox
  ipport: TCP/UDP port to use for connection with device
  ipaddr (required): IP Address or Hostname
  port (required): Physical plug number, name of virtual machine or UUID
  secure: SSH connection
  cmd_prompt: Force Python regex for command prompt
  inet6_only: Forces agent to use IPv6 addresses only
  identity_file: Identity file for ssh
  inet4_only: Forces agent to use IPv4 addresses only
  passwd_script: Script to retrieve password
  passwd: Login password or passphrase
  ssh_options: SSH options to use
  action (required): Fencing Action
  login (required): Login Name
  verbose: Verbose mode
  debug: Write debug information to given file
  version: Display version information and exit
  help: Display help and exit
  separator: Separator for CSV created by operation list
  power_wait: Wait X seconds after issuing ON/OFF
  login_timeout: Wait X seconds for cmd prompt after login
  power_timeout: Test X seconds for status change after ON/OFF
  delay: Wait X seconds before fencing is started
  shell_timeout: Wait X seconds for cmd prompt after issuing command
  retry_on: Count of attempts to retry power on
  stonith-timeout: How long to wait for the STONITH action to complete per a stonith device.
  priority: The priority of the stonith resource. Devices are tried in order of highest priority to
lowest.
  pcmk_host_map: A mapping of host names to ports numbers for devices that do not support host names.
  pcmk_host_list: A list of machines controlled by this device (Optional unless pcmk_host_check=static-
list).
  pcmk_host_check: How to determine which machines are controlled by the device.
```

As you can see the parameters are similar but not exactly the same.

For example:

```
[root@onmssrv01 ~]# /usr/sbin/fence_vbox -a 192.168.205.1 -l onms_user -p onms_passwd -o list | grep onms
opennms-cluster-onmssrv01,5c467fe9-2611-4e5f-99f6-d1c6aa17527e
opennms-cluster-onmssrv02,a379c5c0-2ce0-482a-be49-7237aa67f662
```

The first thing to do is re-enable stonith:

```
[root@onmssrv01 ~]# pcs property set stonith-enabled=true
```

Then, create a stonith entry for each node:

```
[root@onmssrv01 ~]# pcs stonith create vbox_1 fence_vbox ipaddr=192.168.205.1 \
login=onms_user passwd=onms_passwd \
pcmk_host_map="onmssrv01:opennms-cluster-onmssrv01" \
pcmk_host_list="onmssrv01"

[root@onmssrv01 ~]# pcs stonith create vbox_2 fence_vbox ipaddr=192.168.205.1 \
login=onms_user passwd=onms_passwd \
pcmk_host_map="onmssrv02:opennms-cluster-onmssrv02" \
pcmk_host_list="onmssrv02"
```

The key parameters here are `pcmk_host_map` and `pcmk_host_list`.

- `pcmk_host_list`, contains the list of cluster nodes that are going to be fenced with the fence device.

- `pcmk_host_map`, contains the map list of each cluster node that is going to be fenced with the fence device.

As mentioned before, the *VM* names on *VirtualBox* might not be the same as the names used for the nodes (and keep in mind that the name of the cluster nodes could be the FQDN or the short name, depending on how you've added them to the cluster). So, the value for this attribute has the following format:

```
pcmk_host_map="{cluster_node_name_X}:{vm_name_in_virtualbox_X}"
```

Now, it is important to tell the cluster where the stonith resources must be allocated. The vbox_1 is defined to fence *onmssrv01*, so it cannot be running on the same node for obvious reasons. Because of this, we need to define a location constraint for each snotty resource.

```
[root@onmssrv01 ~]# pcs constraint location add fence_onmssrv01_loc vbox_1 onmssrv01 -INFINITY
[root@onmssrv01 ~]# pcs constraint location add fence_onmssrv02_loc vbox_2 onmssrv02 -INFINITY

fence_onmssrv01_loc means, start vbox_1 anywhere but onmssrv01.
fence_onmssrv02_loc means, start vbox_2 anywhere but onmssrv02.
```

You can check the configured settings with the following command:

```
[root@onmssrv02 ~]# pcs stonith show --full
 Resource: vbox_1 (class=stonith type=fence_vbox)
  Attributes: ipaddr=192.168.205.1 login=onms_user passwd=onms_passwd pcmk_host_map=onmssrv01:opennms-
cluster-onmssrv01 pcmk_host_list=onmssrv01
  Operations: monitor interval=60s (vbox_1-monitor-interval-60s)
 Resource: vbox_2 (class=stonith type=fence_vbox)
  Attributes: ipaddr=192.168.205.1 login=onms_user passwd=onms_passwd pcmk_host_map=onmssrv02:opennms-
cluster-onmssrv02 pcmk_host_list=onmssrv02
  Operations: monitor interval=60s (vbox_2-monitor-interval-60s)
```

Use the following command to verify that each stonith resource is associated with the proper cluster node:

```
[root@onmssrv01 ~]# stonith_admin -l onmssrv01
 vbox_1
1 devices found
[root@onmssrv01 ~]# stonith_admin -l onmssrv02
 vbox_2
1 devices found
```

If you see a different output, check the stonith configuration.

Now, to validate the constraints:

```
[root@onmssrv01 ~]# pcs constraint show --full
Location Constraints:
  Resource: onms_app
    Enabled on: onmssrv01 (score:INFINITY) (role: Started) (id:cli-prefer-onms_app)
  Resource: vbox_1
    Disabled on: onmssrv01 (score:-INFINITY) (id:fence_onmssrv01_loc)
  Resource: vbox_2
    Disabled on: onmssrv02 (score:-INFINITY) (id:fence_onmssrv02_loc)
Ordering Constraints:
Colocation Constraints:
```

The first is related with the resource group, and the last two are associated with the fencing resources.

The cluster status shows the following:

```
[root@onmssrv01 ~]# pcs status
Cluster name: cluster_onms
Last updated: Wed Jul 29 20:06:57 2015
Last change: Wed Jul 29 20:05:56 2015
Stack: corosync
Current DC: onmssrv01 (1) - partition with quorum
Version: 1.1.12-a14efad
2 Nodes configured
8 Resources configured

Online: [ onmssrv01 onmssrv02 ]

Full list of resources:

 Resource Group: onms_app
     virtual_ip(ocf::heartbeat:IPaddr2):Started onmssrv01
     onms_etc(ocf::heartbeat:Filesystem):Started onmssrv01
     onms_var(ocf::heartbeat:Filesystem):Started onmssrv01
     pgpool_etc(ocf::heartbeat:Filesystem):Started onmssrv01
     pgpool_bin(systemd:pgpool):Started onmssrv01
     onms_bin(systemd:opennms):Started onmssrv01
 vbox_1(stonith:fence_vbox):Started onmssrv02
 vbox_2(stonith:fence_vbox):Started onmssrv01

PCSD Status:
  onmssrv01: Online
  onmssrv02: Online

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

Pacemaker doesn't have a `fence_node` command to test fencing like *rgmanager* on *RHEL/CentOS 6*. For this reason, testing fence in this case is different.

At this point you should be able to test fencing with `stonith_admin -F;` or simulate a node problem with `killall -9 corosync`, on the active cluster node.

```
[root@onmssrv02 ~]# stonith_admin -B onmssrv01
```

After running the above command, *onmssrv01* will be rebooted and all the resources will be moved to *onmssrv02*.

Once you start *onmssrv01* again (through *vCenter*), you can see it will join the cluster.

VMWare

As mentioned before, *RedHat* provides a fencing script for *VMWare* called `fencing_vmware_soap`. In order to use it, the *SOAP* Interface must be enabled on *vCenter* for a user designated for fencing. This user must be created with the following permissions on *vCenter*:

```
System.Anonymous
System.View
VirtualMachine.Interact.PowerOff
VirtualMachine.Interact.PowerOn
```

Once the user is configured, you can test the fencing script to be sure it works. The first thing to do is retrieving the list of VMs using `fencing_vmware_soap`. This will confirm that the credentials are correct, and will give us all the information required to configure fencing on the cluster.

```
[root@onmssrv01 ~]# /usr/sbin/fence_vmware_soap -a vcentersrv01.local -l onms_user -p onms_passwd -z -o
list | grep onms
onmssrv01,42332447-59ef-4d71-a200-6f3c61b549a9
onmssrv02,423317fe-8ee0-e442-7529-c1f9e3eede38
```

As you can see, you should provide the *IP* or *FQDN* of the *vCenter* server (`-a`), the user credentials with the required permissions to manipulate *VMs* as mentioned before (`-l`, `-p`), the command (`-o`), and either the name (`-n`) or the *UUID* (`-U`) of the *VM* you want to manipulate. Also, it is common that *SSL* is enabled (`-z`).

Looking at the output of the list command, the *VM's* name are displayed at the left side, and their *UUIDs* appear at the left side. Keep in mind that not necessarily the *FQDN* of the server is the name of the *VM* in *vCenter*. For this reason, it is important to use the above command, or look at the *vCenter UI* to pick the correct names. Also, it is extremely important to avoid changing the *VM* name after configuring the fencing if you are planing to use them for fencing instead of the *UUIDs*.

In order to retrieve the status, you could use either of them, for example:

```
[root@onmssrv01 ~]# /usr/sbin/fence_vmware_soap -a vcentersrv01.local -l onms_user -p onms_passwd -z -o
status -n "onmssrv02"
Status: ON

[root@onmssrv01 ~]# /usr/sbin/fence_vmware_soap -a vcentersrv01.local -l onms_user -p onms_passwd -z -o
status -U "423317fe-8ee0-e442-7529-c1f9e3eede38"
Status: ON
```

If you want to test further you can request a power-off by using `-o off` on any of above commands.

Once the script works without issues, you can configure the fencing mechanism on the cluster.

RHEL/CentOS 6

The following command can be used to identify the available parameters of a supported fencing script:

```
[root@onmssrv01 ~]# ccs --lsfenceopts fence_vmware_soap
fence_vmware_soap - Fence agent for VMWare over SOAP API
  Required Options:
  Optional Options:
    option: No description available
    action: Fencing Action
    ipaddr: IP Address or Hostname
    login: Login Name
    passwd: Login password or passphrase
    passwd_script: Script to retrieve password
    ssl: SSL connection
    port: Physical plug number or name of virtual machine
    uuid: The UUID of the virtual machine to fence.
    ipport: TCP port to use for connection with device
    verbose: Verbose mode
    debug: Write debug information to given file
    version: Display version information and exit
    help: Display help and exit
    separator: Separator for CSV created by operation list
    power_timeout: Test X seconds for status change after ON/OFF
    shell_timeout: Wait X seconds for cmd prompt after issuing command
    login_timeout: Wait X seconds for cmd prompt after login
    power_wait: Wait X seconds after issuing ON/OFF
    delay: Wait X seconds before fencing is started
    retry_on: Count of attempts to retry power on
```

The mapping between the parameter names and their equivalent switches are the following:

```
-a : ipaddr
-l : login
-p : passwd
-o : action
-z : ssl
-n : port
-U : uuid
```

Add the fence device (using `vmware_dev` for the device's name):

```
[root@onmssrv01 ~]# ccs --addfencedev vmware_dev agent=fence_vmware_soap ipaddr= vcentersrv01.local
login="onms_user" passwd="onms_passwd"
```

Add a method to every cluster node (using vmware_m for the method's name):

```
[root@onmssrv01 ~]# ccs --addmethod vmware_m onmssrv01.local
Method vmware_m added to onmssrv01.local.
[root@onmssrv01 ~]# ccs --addmethod vmware_m onmssrv02.local
Method vmware_m added to onmssrv02.local.
```

Add the fence instance to each cluster node:

```
[root@onmssrv01 ~]# ccs --addfenceinst vmware_dev onmssrv01.local vmware_m ssl="on" port="onmssrv01"
[root@onmssrv01 ~]# ccs --addfenceinst vmware_dev onmssrv02.local vmware_m ssl="on" port="onmssrv02"
```

As an alternative, you can use the *UUIDs* in you want, instead of the *VM* names:

```
[root@onmssrv01 ~]# ccs --addfenceinst vmware onmssrv01.local vmware ssl="on" uuid="423317fe-8ee0-e442-
7529-c1f9e3eede38"
[root@onmssrv01 ~]# ccs --addfenceinst vmware onmssrv02.local vmware ssl="on" uuid="42332447-59ef-4d71-
a200-6f3c61b549a9"
```

The advantage of using the *UUID* is that the VM name can be changed at any time by the administrators, while the *UUID* should be fixed.

Remember to use the command shown at the beginning of the section to obtain the *VM* names or *UUIDs*.

Propagate the cluster configuration and activate it:

```
[root@onmssrv01 ~]# ccs --sync --activate
```

1. Test Fencing

There are several ways to test it:

- Block the network access of the active *VM* through *vCenter*.
- Create an iptables rule to block all the IN/OUT traffic on the machine.
- Use the `fence_node` command to trigger the fencing on a given node using the information from `cluster.conf`.

I'm going to use the third option:

```
[root@onmssrv02 ~]# fence_node -vv onmssrv01.local
fence onmssrv01.local dev 0.0 agent fence_vmware_soap result: success
agent args: port=onmssrv01 ssl=on nodename=centos1 agent=fence_vmware_soap ipaddr= vcentersrv01.local
login=onms_user passwd=onms_passwd
```

I tested both configuration options: using the *VM* names, and then using the *UUIDs*. Both were working properly.

RHEL/CentOS 7

The following command can be used to get the list of parameters of a given fencing script:

```
[root@onmssrv01 ~]# pcs stonith list | grep vmware
fence_vmware_soap - Fence agent for VMWare over SOAP API

[root@onmssrv01 ~]# pcs stonith describe fence_vmware_soap
Stonith options for: fence_vmware_soap
  ipport: TCP/UDP port to use for connection with device
  notls: Disable TLS negotiation, force SSL 3.0
  ssl_secure: SSL connection with verifying fence device's certificate
  port (required): Physical plug number, name of virtual machine or UUID
  inet6_only: Forces agent to use IPv6 addresses only
  ipaddr (required): IP Address or Hostname
  inet4_only: Forces agent to use IPv4 addresses only
  passwd_script: Script to retrieve password
  passwd: Login password or passphrase
  ssl: SSL connection
  ssl_insecure: SSL connection without verifying fence device's certificate
  action (required): Fencing Action
  login (required): Login Name
  verbose: Verbose mode
  debug: Write debug information to given file
  version: Display version information and exit
  help: Display help and exit
  separator: Separator for CSV created by operation list
  power_wait: Wait X seconds after issuing ON/OFF
  login_timeout: Wait X seconds for cmd prompt after login
  power_timeout: Test X seconds for status change after ON/OFF
  delay: Wait X seconds before fencing is started
  shell_timeout: Wait X seconds for cmd prompt after issuing command
  retry_on: Count of attempts to retry power on
  stonith-timeout: How long to wait for the STONITH action to complete per a stonith device.
  priority: The priority of the stonith resource. Devices are tried in order of highest priority to
lowest.
  pcmk_host_map: A mapping of host names to ports numbers for devices that do not support host names.
  pcmk_host_list: A list of machines controlled by this device (Optional unless pcmk_host_check=static-
list).
  pcmk_host_check: How to determine which machines are controlled by the device.
```

As you can see the parameters are similar but not exactly the same.

Actually the *VMWare* fencing script is more strict on *RHEL/CentOS 7* so it might complain about the *SSL Certificate* if it cannot be verified. If this is the case, you should see the following message when running the script:

```
Server side certificate verification failed
```

If you see the above message, add `--ssl-insecure` to the command and it should work; for example:

```
[root@onmssrv01 ~]# /usr/sbin/fence_vmware_soap -a vcentersrv01.local -l onms_user -p onms_passwd -o list
-z --ssl-insecure | grep onms
/usr/lib/python2.7/site-packages/urllib3/connectionpool.py:769: InsecureRequestWarning: Unverified HTTPS
request is being made. Adding certificate verification is strongly advised. See:
https://urllib3.readthedocs.org/en/latest/security.html
  InsecureRequestWarning)
onmssrv01,42332447-59ef-4d71-a200-6f3c61b549a9
onmssrv02,423317fe-8ee0-e442-7529-c1f9e3eede38
```

Now, the first thing to do is re-enable stonith:

```
[root@onmssrv01 ~]# pcs property set stonith-enabled=true
```

Then, create a stonith entry for each node:

```
[root@onmssrv01 ~]# pcs stonith create vmware_1 fence_vmware_soap \
ipaddr=vcentersrv01.local login=onms_user passwd=onms_passwd \
ssl=1 ssl_insecure=1 \
pcmk_host_map="onmssrv01:onmssrv01" \
pcmk_host_list="onmssrv01"

[root@onmssrv01 ~]# pcs stonith create vmware_2 fence_vmware_soap \
ipaddr=vcentersrv01.local login=onms_user passwd=onms_passwd \
ssl=1 ssl_insecure=1 \
pcmk_host_map="onmssrv02:onmssrv02" \
pcmk_host_list="onmssrv02"
```

NOTE If you don't have problems with the *SSL Certificate*, you can remove the `ssl_insecure` parameter from the above commands.

The key parameters here are `pcmk_host_map` and `pcmk_host_list`.

- `pcmk_host_list`, contains the list of cluster nodes that are going to be fenced with the fence device.
- `pcmk_host_map`, contains the map list of each cluster node that is going to be fenced with the fence device.

As mentioned before, the *VM* names on *vCenter* might not be the same as the names used for the nodes (and keep in mind that the name of the cluster nodes could be the *FQDN* or the short name, depending on how you've added them to the cluster). So, the value for this attribute has the following format:

```
pcmk_host_map="{cluster_node_name_X}:{vm_name_on_vcenter_X}"
```

Now, it is important to tell the cluster where the stonith resources must be allocated. The vbox_1 is defined to fence *onmssrv01*, so it cannot be running on the same node for obvious reasons. because of this, we need to define a location constraint for each snotty resource"

```
[root@onmssrv01 ~]# pcs constraint location add fence_onmssrv01_loc vmware_1 onmssrv01 -INFINITY
[root@onmssrv01 ~]# pcs constraint location add fence_onmssrv02_loc vmware_2 onmssrv02 -INFINITY

fence_onmssrv01_loc means, start vmware_1 anywhere but onmssrv01.
fence_onmssrv02_loc means, start vmware_2 anywhere but onmssrv02.
```

You can check the configured settings with the following command:

```
[root@onmssrv02 ~]# pcs stonith show   full
 Resource: vmware_1 (class=stonith type=fence_vmware_soap)
  Attributes: ipaddr=vcentersrv01.local login=onms_user passwd=onms_passwd ssl=1 ssl_insecure=1
pcmk_host_map=onmssrv01:onmssrv01 pcmk_host_list=onmssrv01
  Operations: monitor interval=60s (vmware_1-monitor-interval-60s)
 Resource: vmware_2 (class=stonith type=fence_vmware_soap)
  Attributes: ipaddr=vcentersrv01.local login=onms_user passwd=onms_passwd ssl=1 ssl_insecure=1
pcmk_host_map=onmssrv02:onmssrv02 pcmk_host_list=onmssrv02
  Operations: monitor interval=60s (vmware_2-monitor-interval-60s)
```

Use the following command to verify that each stonith resource is associated with the proper cluster node:

```
[root@onmssrv01 ~]# stonith_admin -l onmssrv01
 vmware_1
1 devices found
[root@onmssrv01 ~]# stonith_admin -l onmssrv02
 vmware_2
1 devices found
```

If you see a different output, check the stonith configuration.

Now, to validate the constraints:

```
[root@onmssrv01 ~]# pcs constraint show --full
Location Constraints:
  Resource: onms_app
    Enabled on: onmssrv01 (score:INFINITY) (role: Started) (id:cli-prefer-onms_app)
  Resource: vmware_1
    Disabled on: onmssrv01 (score:-INFINITY) (id:fence_onmssrv01_loc)
  Resource: vmware_2
    Disabled on: onmssrv02 (score:-INFINITY) (id:fence_onmssrv02_loc)
Ordering Constraints:
Colocation Constraints:
```

The first is related with the resource group, and the last two are associated with the fencing resources.

The cluster status shows the following:

```
[root@onmssrv01 ~]# pcs status
Cluster name: cluster_onms
Last updated: Fri Jul 24 19:40:51 2015
Last change: Fri Jul 24 19:29:12 2015
Stack: corosync
Current DC: onmssrv02 (2) - partition with quorum
Version: 1.1.12-a14efad
2 Nodes configured
8 Resources configured

Online: [ onmssrv01 onmssrv02 ]

Full list of resources:

 Resource Group: onms_app
     virtual_ip(ocf::heartbeat:IPaddr2):Started onmssrv01
     onms_etc(ocf::heartbeat:Filesystem):Started onmssrv01
     onms_var(ocf::heartbeat:Filesystem):Started onmssrv01
     pgpool_etc(ocf::heartbeat:Filesystem):Started onmssrv01
     pgpool_bin(systemd:pgpool):Started onmssrv01
     onms_bin(systemd:opennms):Started onmssrv01
 vmware_1(stonith:fence_vbox):Started onmssrv02
 vmware_2(stonith:fence_vbox):Started onmssrv01

PCSD Status:
  onmssrv01: Online
  onmssrv02: Online

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

Pacemaker doesn’t have a `fence_node` command to test fencing like *rgmanager* on *RHEL/CentOS 6*. For this reason, testing fence in this case is different.

At this point you should be able to test fencing with `stonith_admin -F;` or simulate a node problem with `killall -9 corosync`, on the active cluster node.

```
[root@onmssrv02 ~]# stonith_admin -B onmssrv01
```

After running the above command, *onmssrv01* will be rebooted and all the resources will be moved to *onmssrv02*.

Once you start onmssrv01 again (through *vCenter*), you can see it will join the cluster.