Fencing and Stonith

Dejan Muhamedagic, dejan@suse.de

v0.9

Fencing is a very important concept in computer clusters for HA (High Availability). Unfortunately, given that fencing does not offer a visible service to users, it is often neglected.

Fencing may be defined as a method to bring an HA cluster to a known state. But, what is a "cluster state" after all? To answer that question we have to see what is in the cluster.

1 Introduction to HA clusters

Any computer cluster may be loosely defined as a collection of cooperating computers or nodes. Nodes talk to each other over communication channels, which are typically standard network connections, such as Ethernet.

The main purpose of an HA cluster is to manage user services. Typical examples of user services are an Apache web server or, say, a MySQL database. From the user's point of view, the services do some specific and hopefully useful work when ordered to do so. To the cluster, however, they are just things which may be started or stopped. This distinction is important, because the nature of the service is irrelevant to the cluster. In the cluster lingo, the user services are known as resources.

Every resource has a state attached, for instance: "resource r1 is started on node1". In an HA cluster, such state implies that "resource r1 is stopped on all nodes but node1", because an HA cluster must make sure that every resource may be started on at most one node.

A collection of resource states and node states is the cluster state.

Every node must report every change that happens to resources. This may happen only for the running resources, because a node should not start resources unless told so by somebody. That somebody is the Cluster Resource Manager (CRM) in our case.

So far so good. But what if, for whatever reason, we cannot establish with certainty a state of some node or resource? This is where fencing comes in. With fencing, even when the cluster doesn't know what is happening on some node, we can make sure that that node doesn't run any or certain important resources.

If you wonder how this can happen, there may be many risks involved with computing: reckless people, power outages, natural disasters, rodents, thieves, software bugs, just to name a few. We are sure that at least a few times your computer failed unpredictably.

2 Fencing

There are two kinds of fencing: resource level and node level.

Using the resource level fencing the cluster can make sure that a node cannot access one or more resources. One typical example is a SAN, where a fencing operation changes rules on a SAN switch to deny access from a node.

The resource level fencing may be achieved using normal resources on which the resource we want to protect would depend. Such a resource would simply refuse to start on this node and therefore resources which depend on it will be unrunnable on the same node as well.

The node level fencing makes sure that a node does not run any resources at all. This is usually done in a very simple, yet brutal way: the node is simply reset using a power switch. This may ultimately be necessary because the node may not be responsive at all.

The node level fencing is our primary subject below.

3 Node level fencing devices

Before we get into the configuration details, you need to pick a fencing device for the node level fencing. There are quite a few to choose from. If you want to see the list of stonith devices which are supported just run:

```
stonith -L
```

Stonith devices may be classified into five categories:

- UPS (Uninterruptible Power Supply)
- PDU (Power Distribution Unit)
- Blade power control devices
- Lights-out devices
- Testing devices

The choice depends mainly on your budget and the kind of hardware. For instance, if you're running a cluster on a set of blades, then the power control device in the blade enclosure is the only candidate for fencing. Of course, this device must be capable of managing single blade computers.

The lights-out devices (IBM RSA, HP iLO, Dell DRAC) are becoming increasingly popular and in future they may even become standard equipment of of-the-shelf computers. They are, however, inferior to UPS devices, because they share a power supply with their host (a cluster node). If a node stays without power, the device supposed to control it would be just as useless. Even though this is obvious to us, the cluster manager is not in the know and will try to fence the node in vain. This will continue forever because all other resource operations would wait for the fencing/stonith operation to succeed.

The testing devices are used exclusively for testing purposes. They are usually more gentle on the hardware. Once the cluster goes into production, they must be replaced with real fencing devices.

4 STONITH (Shoot The Other Node In The Head)

Stonith is our fencing implementation. It provides the node level fencing.

NB The stonith and fencing terms are often used interchangeably here as well as in other texts.

The stonith subsystem consists of two components:

- stonithd
- stonith plugins

4.1 stonithd

stonithd is a daemon which may be accessed by the local processes or over the network. It accepts commands which correspond to fencing operations: reset, power-off, and power-on. It may also check the status of the fencing device.

stonithd runs on every node in the CRM HA cluster. The stonithd instance running on the DC node receives a fencing request from the CRM. It is up to this and other stonithd programs to carry out the desired fencing operation.

4.2 Stonith plugins

For every supported fencing device there is a stonith plugin which is capable of controlling that device. A stonith plugin is the interface to the fencing device. All stonith plugins look the same to stonithd, but are quite different on the other side reflecting the nature of the fencing device.

Some plugins support more than one device. A typical example is ipmilan (or external/ipmi) which implements the IPMI protocol and can control any device which supports this protocol.

5 CRM stonith configuration

The fencing configuration consists of one or more stonith resources.

A stonith resource is a resource of class stonith and it is configured just like any other resource. The list of parameters (attributes) depend on and are specific to a stonith type. Use the stonith(1) program to see the list:

```
$ stonith -t ibmhmc -n
ipaddr
$ stonith -t ipmilan -n
hostname  ipaddr  port  auth  priv  login  password reset_method
```

NB It is easy to guess the class of a fencing device from the set of attribute names.

A short help text is also available:

```
$ stonith -t ibmhmc -h
STONITH Device: ibmhmc - IBM Hardware Management Console (HMC)
Use for IBM i5, p5, pSeries and OpenPower systems managed by HMC
  Optional parameter name managedsyspat is white-space delimited
list of patterns used to match managed system names; if last
character is '*', all names that begin with the pattern are matched
  Optional parameter name password is password for hscroot if
passwordless ssh access to HMC has NOT been setup (to do so,
it is necessary to create a public/private key pair with
empty passphrase - see "Configure the OpenSSH client" in the
redbook for more details)
For more information see
http://publib-b.boulder.ibm.com/redbooks.nsf/RedbookAbstracts/SG247038.html
```

> **You just said that there is stonithd and stonith plugins. What's with these resources now?**
> Resources of class stonith are just a representation of stonith plugins in the CIB. Well, a bit more: apart from the fencing operations, the stonith resources, just like any other, may be started and stopped and monitored. The start and stop operations are a bit of a misnomer: enable and disable would serve better, but it's too late to change that. So, these two are actually administrative operations and do not translate to any operation on the fencing device itself. Monitor, however, does translate to device status.

A dummy stonith resource configuration, which may be used in some testing scenarios is very simple:

```
configure
primitive st-null stonith:null \
        params hostlist="node1 node2"
clone fencing st-null
commit
```

NB

All configuration examples are in the crm configuration tool syntax. To apply them, put the sample in a text file, say sample.txt and run:

```
crm < sample.txt
```

The configure and commit lines are omitted from further examples.

An alternative configuration:

```
primitive st-node1 stonith:null \
        params hostlist="node1"
primitive st-node2 stonith:null \
        params hostlist="node2"
location l-st-node1 st-node1 -inf: node1
location l-st-node2 st-node2 -inf: node2
```

This configuration is perfectly alright as far as the cluster software is concerned. The only difference to a real world configuration is that no fencing operation takes place.

A more realistic, but still only for testing, is the following external/ssh configuration:

```
primitive st-ssh stonith:external/ssh \
        params hostlist="node1 node2"
clone fencing st-ssh
```

This one can also reset nodes. As you can see, this configuration is remarkably similar to the first one which features the null stonith device.

What is this clone thing?

Clones are a CRM/Pacemaker feature. A clone is basically a shortcut: instead of defining n identical, yet differently named resources, a single cloned resource suffices. By far the most common use of clones is with stonith resources if the stonith device is accessible from all nodes.

The real device configuration is not much different, though some devices may require more attributes. For instance, an IBM RSA lights-out device might be configured like this:

```
primitive st-ibmrsa-1 stonith:external/ibmrsa-telnet \
        params nodename=node1 ipaddr=192.168.0.101 \
        userid=USERID passwd=PASSW0RD
primitive st-ibmrsa-2 stonith:external/ibmrsa-telnet \
```

```
        params nodename=node2 ipaddr=192.168.0.102 \
        userid=USERID passwd=PASSWORD
# st-ibmrsa-1 can run anywhere but on node1
location l-st-node1 st-ibmrsa-1 -inf: node1
# st-ibmrsa-2 can run anywhere but on node2
location l-st-node2 st-ibmrsa-2 -inf: node2
```

Why those strange location constraints?
There is always certain probability that the stonith operation is going to fail. Hence, a stonith operation on the node which is the executioner too is not reliable. If the node is reset, then it cannot send the notification about the fencing operation outcome. The only way to do that is to assume that the operation is going to succeed and send the notification beforehand. Then, if the operation fails, we are in trouble. Given all this, we decided that, by convention, stonithd refuses to kill its host.

If you haven't already guessed, configuration of a UPS kind of fencing device is remarkably similar to all we have already shown.

All UPS devices employ the same mechanics for fencing. What is, however, different is how the device itself is accessed. Old UPS devices, those that were considered professional, used to have just a serial port, typically connected at 1200baud using a special serial cable. Many new ones still come equipped with a serial port, but often they also sport a USB interface or an Ethernet interface. The kind of connection we may make use of depends on what the plugin supports. Let's see a few examples for the APC UPS equipment:

```
$ stonith -t apcmaster -h

STONITH Device: apcmaster - APC MasterSwitch (via telnet)
NOTE: The APC MasterSwitch accepts only one (telnet)
connection/session a time. When one session is active,
subsequent attempts to connect to the MasterSwitch will fail.
For more information see http://www.apc.com/
List of valid parameter names for apcmaster STONITH device:
        ipaddr
                login
                password

$ stonith -t apcsmart -h

STONITH Device: apcsmart - APC Smart UPS
```

```
 (via serial port - NOT USB!).
 Works with higher-end APC UPSes, like
 Back-UPS Pro, Smart-UPS, Matrix-UPS, etc.
 (Smart-UPS may have to be >= Smart-UPS 700?).
 See http://www.networkupstools.org/protocols/apcsmart.html
 for protocol compatibility details.
For more information see http://www.apc.com/
List of valid parameter names for apcsmart STONITH device:
                ttydev
                hostlist
```

The former plugin supports APC UPS with a network port and telnet protocol. The latter plugin uses the APC SMART protocol over the serial line which is supported by many different APC UPS product lines.

> **So, what do I use: clones, constraints, both?**
> It depends. Depends on the nature of the fencing device. For example, if the device cannot serve more than one connection at the time, then clones won't do. Depends on how many hosts can the device manage. If it's only one, and that is always the case with lights-out devices, then again clones are right out. Depends also on the number of nodes in your cluster: the more nodes the more desirable to use clones. Finally, it is also a matter of personal preference.
> In short: if clones are safe to use with your configuration and if they reduce the configuration, then make cloned stonith resources.

The CRM configuration is left as an exercise to the reader.

6 Monitoring the fencing devices

Just like any other resource, the stonith class agents also support the monitor operation. Given that we have often seen monitor either not configured or configured in a wrong way, we have decided to devote a section to the matter.

Monitoring stonith resources, which is actually checking status of the corresponding fencing devices, is strongly recommended. So strongly, that we should consider a configuration without it invalid.

On the one hand, though an indispensable part of an HA cluster, a fencing device, being the last line of defense, is used seldom. Very seldom and preferably never. On the other, for whatever reason, the power management equipment is known to be rather fragile on the communication side. Some devices were known to give up if there was too much

broadcast traffic on the wire. Some cannot handle more than ten or so connections per minute. Some get confused or depressed if two clients try to connect at the same time. Most cannot handle more than one session at the time. The bottom line: try not to exercise your fencing device too often. It may not like it. Use monitoring regularly, yet sparingly, say once every couple of hours. The probability that within those few hours there will be a need for a fencing operation and that the power switch would fail is usually low.

7 Odd plugins

Apart from plugins which handle real devices, some stonith plugins are a bit out of line and deserve special attention.

7.1 external/kdumpcheck

Sometimes, it may be important to get a kernel core dump. This plugin may be used to check if the dump is in progress. If that is the case, then it will return true, as if the node has been fenced, which is actually true given that it cannot run any resources at the time. kdumpcheck is typically used in concert with another, real, fencing device. See README_kdumpcheck.txt for more details.

7.2 external/sbd

This is a self-fencing device. It reacts to a so-called "poison pill" which may be inserted into a shared disk. On shared storage connection loss, it also makes the node commit suicide. See //www.linux-ha.org/SBD_Fencing for more details.

7.3 meatware

Strange name and a simple concept. `meatware` requires help from a human to operate. Whenever invoked, `meatware` logs a CRIT severity message which should show up on the node's console. The operator should then make sure that the node is down and issue a `meatclient(8)` command to tell `meatware` that it's OK to tell the cluster that it may consider the node dead. See `README.meatware` for more information.

7.4 null

This one is probably not of much importance to the general public. It is used in various testing scenarios. `null` is an imaginary device which always behaves and always claims that it has shot a node, but never does anything. Sort of a happy-go-lucky. Do not use it unless you know what you are doing.

7.5 suicide

`suicide` is a software-only device, which can reboot a node it is running on. It depends on the operating system, so it should be avoided whenever possible. But it is OK on one-node clusters. `suicide` and `null` are the only exceptions to the "don't shoot my host" rule.

What about that stonithd? You forgot about it, eh?
The stonithd daemon, though it is really the master of ceremony, requires no configuration itself. All configuration is stored in the CIB.

8 Resources

//linux-ha.org/STONITH

//linux-ha.org/fencing

//linux-ha.org/ConfiguringStonithPlugins

//linux-ha.org/CIB/Idioms

//www.clusterlabs.org/mediawiki/images/f/fb/Configuration_Explained.pdf

//techthoughts.typepad.com/managing_computers/2007/10/split-brain-quo.html