NAME

stonith_sbd – Stonith Block Device

DESCRIPTION

*** Data Protection**

The SLE-HA cluster stack's highest priority is protecting the integrity of data. This is achieved by preventing uncoordinated concurrent access to data storage - such as mounting an XFS file system more than once in the cluster, but also preventing OCFS2 from being mounted if coordination with other cluster nodes is not available. In a well-functioning cluster, Pacemaker will detect if resources are active beyond their concurrency limits and initiate recovery; further, its policy engine will never exceed these limitations.

However, network partitioning or software malfunction could potentially cause scenarios where several coordinators are elected. If this so-called split brain scenario were allowed to unfold, data corruption might occur. Hence, several layers of protection have been added to the cluster stack to mitigate this.

Server fencing via STONITH (Shoot the other node into the head) is the primary component contributing to this goal, since they ensure that, prior to storage activation, all other access is terminated. cLVM2 (cluster Logical Volume Manager 2) exclusive activation or OCFS2 (Oracle Cluster File System 2) file locking support are other mechanisms, protecting against administrative or application faults. Combined appropriately for your setup, these can reliably prevent split-brain scenarios from causing harm.

This chapter describes an IO fencing mechanism that leverages the storage itself, following by a description of an additional layer of protection to ensure exclusive storage access. These two mechanisms can even be combined for higher levels of protection.

*** Storage-based Fencing**

In scenarios where shared storage is used one can leverage said shared storage for very reliable server fencing and avoidance of split-brain scenarios.

This mechanism has been used successfully with the Novell Cluster Suite and is also available in a similar fashion for the SLE-HA product using the "external/sbd" STONITH agent.

In an environment where all nodes have access to shared storage, a small partition is formated for use with SBD (STONITH block device). The sbd daemon, once configured, is brought online on each node before the rest of the cluster stack is started, and terminated only after all other cluster components have been shut down - ensuring that cluster resources are never activated without SBD supervision.

The daemon automatically allocates one of the message slots on the partition to itself, and constantly monitors it for messages to itself. Upon receipt of a message, the daemon immediately complies with the request, such as initiating a power-off or reboot cycle for fencing.

The daemon also constantly monitors connectivity to the storage device, and commits suicide in case the partition becomes unreachable, guaranteeing that it is not disconnected from fencing message. (If the cluster data resides on the same logical unit in a different partition, this is not an additional point of failure; the work-load would terminate anyway if the storage connectivity was lost.)

SBD supports one, two, or three devices. This affects the operation of SBD as follows:

**** One device**

In its most simple implementation, you use one device only. (Older versions of SBD did not support more.) This is appropriate for clusters where all your data is on the same shared storage (with internal redundancy) anyway. From a general perspcective the SBD device does not introduce an additional single point of failure then.

But since the timeouts are usually shorter for SBD and watchdog than for other resources, a temporary outage of the SBD device might lead to a non-wanted shutdown. To cover this, current versions of SBD could be configured to tolerate an inacessible SBD deviced as long as the corosync link between the cluster nodes is functional.

If the SBD device is not accessible, the daemon will fail to start and inhibit pacemaker startup.

**** Two devices**

This configuration is a trade-off, primarily aimed at environments where host-based mirroring is used, but no third storage device is available.

SBD will not commit suicide if it loses access to one mirror leg; this allows the cluster to continue to function even in the face of one outage.

However, SBD will not fence the other side while only one mirror leg is available, since it does not have enough knowledge to detect an asymmetric split of the storage. So it will not be able to automatically tolerate a second failure while one of the storage arrays is down. (Though you can use the appropriate crm command to acknowledge the fence manually.)

If devices are configured different, the cluster will not start. If no header is on the devices, the cluster starts and keeps looking for a valid header.

**** Three devices**

In this most reliable configuration, SBD will only commit suicide if more than one device is lost; hence, this configuration is resilient against one device outages (be it due to failures or maintenance). Fencing messages can be successfully relayed if at least two devices remain up.

If one device out of three is completely missing at cluster start, the cluster will start. If one device out of three is available, but mis-configured, the cluster will not start. If two devices are completely missing, the cluster will also not start.

This configuration is appropriate for more complex scenarios where storage is not confined to a single array.

Host-based mirroring solutions could have one SBD per mirror leg (not mirrored itself), and an additional tie-breaker on iSCSI.

*** Pre-Requisites**

The environment must have shared storage reachable by all nodes. You must dedicate a small partition of each as the SBD device. This shared storage segment must not make use of host-based RAID, cLVM2, nor DRBD.

The SBD device can be connected via Fibre Channel, Fibre Channel over Ethernet, or even iSCSI. Thus, an iSCSI target can become a sort-of network-based quorum server; the advantage is that it does not require a smart host at your third location, just block storage.

However, using storage-based RAID and multipathing is recommended for increased reliability.

*** SBD Partition**

It is recommended to create a tiny partition at the start of the device. In the rest of this text, this is referred to as "/dev/<SBD>" or "/dev/<SBD_n>", please substitute your actual pathnames (e.g. "/dev/disk/by-id/scsi-1494554000000000036363600000000000000000000000000-part1") for this below.

The size of the SBD device depends on the block size of the underlying device. SBD uses 255 slots. Thus, 1MB is fine on plain SCSI devices and SAN storages with 512 byte blocks. On the IBM s390x architecture disks could have larger block sizes, as 4096 bytes. Therefor 4MB or more are needed there.

After having made very sure that this is indeed the device you want to use, and does not hold any data you need - as the sbd command will overwrite it without further requests for confirmation -, initialize the sbd device.

If your SBD device resides on a multipath group, you may need to adjust the timeouts sbd uses, as MPIO's path down detection can cause some latency: after the msgwait timeout, the message is assumed to have been delivered to the node. For multipath, this should be the time required for MPIO to detect a path failure and switch to the next path. You may have to test this in your environment. The node will perform suicide if it has not updated the watchdog timer fast enough; the watchdog timeout must be shorter than half the msgwait timeout - half the value is a good estimate. This can be specified when the SBD device is initialized.

Sharing one SBD device among several clusters is not recommended. Even if it is possible to run up to 127 two-node clusters on one single SBD device, this should never be done. For production, each cluster should have its own SBD device(s).

*** Testing and Starting the SBD Daemon**

The sbd daemon is a critical piece of the cluster stack. It must always be running when the cluster stack is up, or even when the rest of it has crashed, so that it can be fenced. The pacemaker init script starts and stops SBD if configured.

On SLE-HA 12 add the following to /etc/sysconfig/sbd:

```
===
#/etc/sysconfig/sbd
# SBD devices (no trailing ";"):
SBD_DEVICE="/dev/<SBD_1>;/dev/<SBD_2>;/dev/<SBD_3>"
# Watchdog, pacemaker snooping, startup:
SBD_WATCHDOG="yes"
SBD_PACEMAKER="yes"
SBD_STARTMODE="clean"
#
===
```

On SLE-HA 11 add the following to /etc/sysconfig/sbd:

```
===
#/etc/sysconfig/sbd
# SBD devices (no trailing ";"):
SBD_DEVICE="/dev/<SBD_1>;/dev/<SBD_2>;/dev/<SBD_3>"
# Watchdog, pacemaker snooping, startup:
SBD_OPTS="-W -P -S 1"
#
===
```

Some notes on SLE-HA 11 SBD_OPTS:

-W: SBD always needs a watchdog to be save. Therefor "-W" is required, which tells the sbd daemon to start the watchdog timer.

-P: The SBD fencing is not needed as long as the corosync communication is working and pacemaker has control over all nodes. In that situation a cluster node not self-fence just because a short outage of the SBD devices. This behaviour is choosen with "-P".

-S 1: If the SBD device becomes inaccessible from a node, this could cause the node to enter an infinite reboot cycle. That is technically correct, but depending on your administrative policies, might be considered a nuisance. The option "-S" controls how sbd behaves if a reset request is found on startup in the node's sbd slot on disk. "-S 1" tells sbd not to start if a reset message is found in the node's slot. Alternatively vou may wish to not automatically start up openais on boot in such cases.

Before proceeding, ensure that SBD has indeed started on all nodes through "rcopenais restart" (SLE-HA 11) or "systemctl pacemaker restart" (SLE-HA 12). Once the sbd and pacemaker service has started, your cluster is now successfully configured for shared-storage fencing, and will utilize this method in case a node needs to be fenced.

The command sbd can be used to read and write the sbd device, see sbd(8) .

*** SBD-related parameters inside the CIB**

To complete the sbd setup, it is necessary to activate SBD as a STONITH/fencing mechanism in the CIB. The SBD mechanism normally is used instead of other fencing/stonith mechanisms. Please disable any others you might have configured before. An escalating STONITH topology usually makes no sense with stonith/sbd.

A sbd device might be entered into the CIB like this:

```
===
primitive rsc_stonith_sbd stonith:external/sbd \
 params pcmk_action_limit="-1" pcmk_delay_max="15"
===
```

Instead of the parameter pcmk_delay_max, an option start-delay is needed before SLE-HA 11 SP3. In case of split-brain this makes the DC winning the fencing race. Recent versions of SLE-HA know the option pcmk_delay_max. This adds a random delay for STONITH actions on the fencing device.

The pcmk_action_limit was introduced with SLE-HA 11 SP4 to allow parallel fencing of multiple nodes in clusters with more than two nodes. This is usefull e.g. for HANA scale-out systems. To make this work, an additional parameter concurrent-fencing is needed in the CIB's property section:

```
===
stonith-enabled="true" \
stonith-timeout="187" \
stonith-action="reboot" \
concurrent-fencing="true" \
===
```

For two-node clusters, the no-quorum-policy="ignore" has to be set as well.

*** Hardware Watchdog**

Increased protection is offered through watchdog support. Modern systems support a "hardware watchdog" that has to be updated by the software client, or else the hardware will enforce a system restart. This protects against failures of the sbd process itself, such as dieing, or becoming stuck on an IO error.

It is highly recommended that you set up your Linux system to use a watchdog. Please refer to the SLES manual for this step.

This involves loading the proper watchdog driver on system boot. On HP hardware, this is the "hpwdt" module. For systems with an Intel TCO, "iTCO_wdt" can be used. Dell and Fujitsu machines usually fall into this category. Machines with an IPMI based watchdog need the "ipmi_watchdog". This might apply for recent Lenovo machines. Inside a VM on z/VM on an IBM mainframe, "vmwatchdog" might be used. Inside a Xen VM (aka DomU) "xen_wdt" is a good choice. Inside a KVM VM "i6300esb" might be used. The corresponding PCI device has to be emulated by the KVM host in that case. "softdog" is the most generic driver, but it is recommended that you use one with actual hardware integration. See /lib/modules/.../kernel/drivers/watchdog in the kernel package for a list of choices.

No other software must access the watchdog timer. Some hardware vendors ship systems management software that use the watchdog for system resets (e.g. HP ASR daemon). Such software has to be disabled if the watchdog is used by SBD.

SBD can be configured in /etc/sysconfig/sbd to use the systems' watchdog.

*** Timeout Settings**

If your SBD device resides on a multipath group, you may need to adjust the timeouts sbd uses, as MPIO's path down detection can cause some latency: after the msgwait timeout, the message is assumed to have been delivered to the node. For multipath, this should be the time required for MPIO to detect a path failure and switch to the next path. You may have to test this in your environment. The node will perform suicide if it has not updated the watchdog timer fast enough; the watchdog timeout must be shorter than the msgwait timeout - half the value is a good estimate. This can be specified when the SBD device is initialized.

If you want to avoid MD mirror splitting in case of IO errors, the watchdog timeout has to be shorter than the total MPIO failure timeout. Thus, a node is fenced before the MD mirror is splitted. On the other hand, the time the cluster waits for SAN and storage to recover is shortened.

In any case, the watchdog timeout must be shorter than sbd message wait timeout. The sbd message wait timeout must be shorter than the cluster stonith-timeout.

If the sbd device recovers from IO errors within the watchdog timeout, the sbd daemon could reset the watchdog timer and save the node from being fenced. To allow re-discovery of a failed sbd device, at least the primary sbd retry cycle should be shorter than the watchdog timeout. Since this cycle is currently hard-coded as ten time the loop timeout, it has to be set by choosing an apropriate loop timeout.

It might be also wise to set a start delay for the cluster resource agent in the CIB. This is done to overcome situations where both nodes fence each other within the sbd loop timeout, see sbd(8).

Putting it all together:
- How long a cluster survives a storage outage depends on the watchdog
  timeout and the sbd retry cycle. All other timeouts should be aligned
  with this settings. That means they have to be longer.
- Storage resources - as Raid1, LVM, Filesystem - have operation
  timeouts. Those should be aligned with the MPIO settings. This avoids
  non-needed failure actions, but does not define how long the cluster
  will survive a storage outage.
- SBD must always be used together with a watchdog.

FILES

/usr/sbin/sbd
: the daemon (and control command).

/usr/lib64/stonith/plugins/external/sbd
: the STONITH plugin.

/etc/sysconfig/sbd
: the SBD configuration file.

/etc/sysconfig/kernel
: the kernel and initrd configuration file.

/etc/rc.d/rc3.d/K01openais
: stop script to prevent stonith during system shutdown (SLE-HA 11).

/dev/<SBD>
: the SBD block device(s).

/dev/watchdog
: the watchdog device node.

/lib/modules/<kernel-version>/kernel/drivers/watchdog/
: the watchdog modules.

BUGS

To report bugs for a SUSE product component, please use
https://www.suse.com/support/report-a-bug/ .

SEE ALSO

sbd(8), **stonith**(8), **cs_add_watchdog_to_initrd**(8), **crm_no_quorum_policy**(7), **cs_disable_other_watchdog**(8), **cs_make_sbd_devices**(8), **dasdfmt**(8), **SAPHanaSR-ScaleOut**(7), **ha_related_acronyms**(7),
http://www.linux-ha.org/wiki/SBD_Fencing , http://www.mail-archive.com/pacemaker@oss.clusterlabs.org/msg03849.html , https://github.com/l-mb/sbd/blob/master/src/sbd.sysconfig , http://www.suse.com/documentation/sle_ha/book_sleha/?page=/documentation/sle_ha/book_sleha/data/part_config.html , https://www.suse.com/documentation/sle_ha/book_sleha/?page=/documentation/sle_ha/book_sleha/data/part_storage.html , https://www.suse.com/documentation/sle_ha/book_sleha/data/sec_ha_storage_protect_fencing.html , https://www.suse.com/support/kb/doc/?id=7004306 , https://www.suse.com/support/kb/doc/?id=7007616 , https://www.suse.com/support/kb/doc/?id=7008921 , https://www.suse.com/support/kb/doc/?id=7009485 , https://www.suse.com/support/kb/doc/?id=7009737 , https://www.suse.com/support/kb/doc/?id=7010879 , https://www.suse.com/support/kb/doc/?id=7010931 , https://www.suse.com/support/kb/doc/?id=7010933 , https://www.suse.com/support/kb/doc/?id=7011346 , https://www.suse.com/support/kb/doc/?id=7011426 , https://www.suse.com/support/kb/doc/?id=7016042 , https://www.suse.com/support/kb/doc/?id=7016305 , https://www.suse.com/support/kb/doc/?id=7016880 , https://www.suse.com/support/kb/doc/?id=7021158 , https://www.suse.com/support/kb/doc/?id=7022255

AUTHORS

The content of this manual page was mostly derived from online documentation mentioned above.

COPYRIGHT