SUSE

Configuring Disk-Based SBD in an Existing High Availability Cluster

WHAT?

How to use the CRM Shell to configure disk-based SBD in a High Availability cluster that is already installed and running.

WHY?

To be supported, all SUSE Linux Enterprise High Availability clusters *must* have node fencing configured. SBD provides a node fencing mechanism without using an external power-off device.

EFFORT

Configuring disk-based SBD in an existing cluster only takes a few minutes and does not require any downtime for cluster resources.

GOAL

Protect the cluster from data corruption by fencing failed nodes.

REQUIREMENTS

- An existing SUSE Linux Enterprise High Availability cluster
- Shared storage accessible from all cluster nodes
- A hardware watchdog device on all cluster nodes

If the SBD service is already running, see *Changing the Configuration of SBD* (https://documentation.suse.com/sle-ha/16.0/html/HA-sbd-changing-configuration/).

Publication Date: 07 Nov 2025

Contents

1 What is node fencing?

In a *split-brain scenario*, cluster nodes are divided into two or more groups (or *partitions*) that do not know about each other. This might be because of a hardware or software failure, or a failed network connection, for example. A split-brain scenario can be resolved by *fencing* (resetting or powering off) one or more of the nodes. Node fencing prevents a failed node from accessing shared resources and prevents cluster resources from running on a node with an uncertain status. This helps protect the cluster from data corruption.

To be supported, all SUSE Linux Enterprise High Availability clusters *must* have at least one node fencing device configured. For critical workloads, we recommend using two or three fencing devices. A fencing device can be either a physical device (a power switch) or a software mechanism (SBD in combination with a watchdog).

1.1 Components

pacemaker-fenced

The `pacemaker-fenced` daemon runs on every node in the High Availability cluster. It accepts fencing requests from `pacemaker-controld`. It can also check the status of the fencing device.

Fence agent

Each type of fencing device can be controlled by a specific *fence agent*, a `stonith`-class resource agent that acts as an interface between the cluster and the fencing device. Starting or stopping a fencing resource means registering or deregistering the fencing device with the `pacemaker-fenced` daemon and does not perform any operation on the device itself. Monitoring a fencing resource means logging in to the device to verify that it works.

Fencing device

The fencing device is the actual physical device that resets or powers off a node when requested by the cluster via the fence agent. The device you use depends on your budget and hardware.

1.2 Fencing devices

Physical devices

- *Power Distribution Units (PDU)* are devices with multiple power outlets that can provide remote load monitoring and power recycling.
- *Uninterruptible Power Supplies (UPS)* provide emergency power to connected equipment in the event of a power failure.
- *Blade power control devices* can be used for fencing if the cluster nodes are running on a set of blades. This device must be capable of managing single-blade computers.
- *Lights-out devices* are network-connected devices that allow remote management and monitoring of servers.

Software mechanisms

- *Disk-based SBD* fences nodes by exchanging messages via shared block storage. It works together with a watchdog on each node to ensure that misbehaving nodes are really stopped.
- *Diskless SBD* fences nodes by using only the watchdog, without a shared storage device. Unlike other node fencing mechanisms, diskless SBD does not need a fence agent.
- The *fence_kdump* agent checks if a node is performing a kernel dump (`kdump`). If a `kdump` is in progress, the cluster acts as if the node was fenced, because the node will reboot after the `kdump` is complete. If a `kdump` is not in progress, the next fencing device fences the node. This fence agent must be used together with a physical fencing device. It cannot be used with SBD.

1.3 For more information

For more information, see https://clusterlabs.org/projects/pacemaker/doc/3.0/Pacemaker_Explained/html/fencing.html ↗.

For a full list of available fence agents, run the **`crm ra list stonith`** command.

For details about a specific fence agent, run the `crm ra info stonith:fence_AGENT` command.

2 What is SBD?

SBD (STONITH Block Device) provides a node fencing mechanism without using an external power-off device. The software component (the SBD daemon) works together with a watchdog device to ensure that misbehaving nodes are fenced. SBD can be used in disk-based mode with shared block storage, or in diskless mode using only the watchdog.

Disk-based SBD uses shared block storage to exchange fencing messages between the nodes. It can be used with one to three devices. One device is appropriate for simple cluster setups, but two or three devices are recommended for more complex setups or critical workloads.

Diskless SBD fences nodes by using only the watchdog, without relying on a shared storage device. A node is fenced if it loses quorum, if any monitored daemon is lost and cannot be recovered, or if Pacemaker determines that the node requires fencing.

2.1 Components

SBD daemon

The SBD daemon starts on each node before the rest of the cluster stack and stops in the reverse order. This ensures that cluster resources are never active without SBD supervision.

SBD device (disk-based SBD)

A small logical unit (or a small partition on a logical unit) is formatted for use with SBD. A message layout is created on the device with slots for up to 255 nodes.

Messages (disk-based SBD)

The message layout on the SBD device is used to send fencing messages to nodes. The SBD daemon on each node monitors the message slot and immediately complies with any requests. To avoid becoming disconnected from fencing messages, the SBD daemon also fences the node if it loses its connection to the SBD device.

Watchdog

SBD needs a watchdog on each node to ensure that misbehaving nodes are really stopped. SBD "feeds" the watchdog by regularly writing a service pulse to it. If SBD stops feeding the watchdog, the hardware enforces a system restart. This protects against failures of the SBD process itself, such as becoming stuck on an I/O error.

2.2 Limitations and recommendations

Disk-based SBD

- The shared storage can be Fibre Channel (FC), Fibre Channel over Ethernet (FCoE), or iSCSI.
- The shared storage must *not* use host-based RAID, LVM, Cluster MD, or DRBD.
- Using storage-based RAID and multipathing is recommended for increased reliability.
- If a shared storage device has different `/dev/sdX` names on different nodes, SBD communication will fail. To avoid this, always use stable device names, such as `/dev/disk/by-id/DEVICE_ID`.
- An SBD device can be shared between different clusters, up to a limit of 255 nodes.
- When using more than one SBD device, all devices must have the same configuration.

Diskless SBD

- Diskless SBD cannot handle a split-brain scenario for a two-node cluster. This configuration should only be used for clusters with more than two nodes, or in combination with QDevice to help handle split-brain scenarios.

2.3 For more information

For more information, see the man page `sbd` or run the **`crm sbd help`** command.

3 Setting up the SBD watchdog

SBD needs a watchdog on each node to ensure that misbehaving nodes are really stopped. SBD "feeds" the watchdog by regularly writing a service pulse to it. If SBD stops feeding the watchdog, the hardware enforces a system restart. This protects against failures of the SBD process itself, such as becoming stuck on an I/O error.

Hardware-specific watchdog drivers are available as kernel modules. However, sometimes the wrong watchdog module loads automatically. Use this procedure to make sure the correct module is loaded.

Important: `softdog` limitations

If no hardware watchdog is available, `crmsh` automatically configures the software watchdog (`softdog`) when configuring SBD. This watchdog can be used for testing purposes, but is *not* recommended for production environments.

The `softdog` driver assumes that at least one CPU is still running, so if all CPUs are stuck, `softdog` cannot reboot the system. Hardware watchdogs work even if all CPUs are stuck.

Perform this procedure on *all* nodes in the cluster:

1. List the drivers that are installed with your kernel version:

   ```
   > rpm -ql kernel-VERSION | grep watchdog
   ```

   To help you find the correct driver for your hardware, see *Table 1, "Commonly used watchdog drivers"*. However, this is not a complete list and might not be accurate for your specific system. Check your system's hardware configuration if possible, or ask your hardware or system vendor for details about system-specific watchdog configuration.

2. Check whether any watchdog modules are already loaded in the kernel:

   ```
   > lsmod | egrep "(wdt|dog)"
   ```

   If the correct watchdog module is already loaded, you can skip to *Step 7*.

3. If the wrong watchdog module is loaded, you can unload it with the following command:

   ```
   > sudo rmmod WRONG_MODULE
   ```

4. Enable the watchdog module that matches your hardware:

```
> sudo bash -c "echo WATCHDOG_MODULE > /etc/modules-load.d/watchdog.conf"
```

   If you run this command as the `root` user, you can omit **`bash -c`** and the quotes (""):

```
# echo WATCHDOG_MODULE > /etc/modules-load.d/watchdog.conf
```

5. Reload the kernel modules:

```
> sudo systemctl restart systemd-modules-load
```

6. Check whether the watchdog module is loaded correctly:

```
> lsmod | egrep "(wdt|dog)"
```

7. Verify that at least one watchdog device is available:

```
> sudo sbd query-watchdog
```

   If no watchdog device is available, you might need to use a different driver.

8. Verify that the watchdog device works:

```
> sudo sbd -w /dev/WATCHDOG_DEVICE test-watchdog
```

   If the test is successful, the node reboots.

Important: Accessing the watchdog timer

SBD must be the only software that accesses the watchdog timer. Some hardware vendors ship systems management software that uses the watchdog for system resets (for example, the HP ASR daemon). If this is the case, disable the additional software.

TABLE 1: COMMONLY USED WATCHDOG DRIVERS

Hardware	Driver
HP	`hpwdt`
Dell, Lenovo (Intel TCO)	`iTCO_wdt`
Fujitsu	`ipmi_watchdog`

Hardware	Driver
LPAR on IBM Power	`pseries-wdt`
VM on IBM z/VM	`vmwatchdog`
VM on VMware vSphere	`wdat_wdt`

4 Setting up disk-based SBD

Disk-based SBD fences nodes by exchanging messages via shared block storage. It works together with a watchdog on each node to ensure that misbehaving nodes are really stopped. You can configure up to three SBD devices.

This procedure explains how to configure SBD after the cluster is already installed and running, not during the initial cluster setup.

Important: Cluster restart required

In this procedure, the script checks whether it is safe to restart the cluster services automatically. If any non-`stonith` resources are running, the script warns you to restart the cluster services manually. This allows you to put the cluster into maintenance mode first to avoid resource downtime. However, be aware that the resources will not have cluster protection while in maintenance mode.

Warning: Overwriting existing data

Make sure any device you want to use for SBD does not hold any important data. Configuring a device for use with SBD overwrites the existing data.

REQUIREMENTS

- An existing High Availability cluster is already running.
- The SBD service is not running.
- Shared storage is configured and accessible on all nodes.

- The path to the shared storage device is consistent across all nodes. Use stable device names such as `/dev/disk/by-id/DEVICE_ID`.
- All nodes have a watchdog device, and the correct watchdog kernel module is loaded.

Perform this procedure on only one cluster node:

1. Log in either as the `root` user or as a user with **`sudo`** privileges.
2. Run the SBD stage of the cluster setup script, using the option `--sbd-device` (or `-s`) to specify the shared storage device:

   ```
   > sudo crm cluster init sbd --sbd-device /dev/disk/by-id/DEVICE_ID
   ```

   ADDITIONAL OPTIONS

   - You can use `--sbd-device` (or `-s`) multiple times to configure up to three SBD devices. Each SBD device must use a different shared storage device.
   - If multiple watchdogs are available, you can use the option `--watchdog` (or `-w`) to choose which watchdog to use. Specify either the device name (for example, `/dev/watchdog1`) or the driver name (for example, `iTCO_wdt`).

   The script initializes SBD on the shared storage device, creates a `fence_sbd` cluster resource, and updates the SBD configuration file and timeout settings. The script also checks whether it is safe to restart the cluster services automatically. If any non-`stonith` resources are running, the script warns you to restart the cluster services manually.
3. If you need to restart the cluster services manually, follow these steps to avoid resource downtime:

   a. Put the cluster into maintenance mode:

      ```
      > sudo crm maintenance on
      ```

      In this state, the cluster stops monitoring all resources. This allows the services managed by the resources to keep running while the cluster restarts. However, be aware that the resources will not have cluster protection while in maintenance mode.

   b. Restart the cluster services on all nodes:

      ```
      > sudo crm cluster restart --all
      ```

c. Check the status of the cluster:

```
> sudo crm status
```

The nodes will have the status `UNCLEAN (offline)`, but will soon change to `Online`.

d. When the nodes are back online, put the cluster back into normal operation:

```
> sudo crm maintenance off
```

4. Check the SBD configuration:

```
> sudo crm sbd configure show
```

The output of this command shows the SBD device's metadata, the enabled settings in the `/etc/sysconfig/sbd` file, and the SBD-related cluster settings.

5. Check the status of SBD:

```
> sudo crm sbd status
```

The output of this command shows the type of SBD configured, information about the SBD watchdog, and the statuses of the SBD service, disk, and cluster resource.

5 Testing SBD and node fencing

Verify that SBD works as expected by performing one or more of the following tests:

5.1 Checking SBD communication

Check whether the SBD device can send and receive messages between the nodes. This procedure uses example nodes called `alice` and `bob`.

1. On either node, list the node slots and their current messages from the SBD device:

```
> sudo sbd -d /dev/disk/by-id/DEVICE_ID list
0      alice  clear
1      bob    clear
```

2. On bob, send a test message to alice:

```
> sudo sbd -d /dev/disk/by-id/DEVICE_ID message alice test
```

3. On alice, check /var/log/messages for the message from bob:

```
> sudo cat /var/log/messages | grep "test"
[...]
Received command test from bob on disk /dev/disk/by-id/DEVICE_ID
```

This confirms that SBD is running and ready to receive messages.

5.2 Testing cluster failures

The **crm cluster crash_test** command simulates cluster failures and reports the results. To test SBD and node fencing, you can run one or more of the tests --fence-node, --kill-sbd and --split-brain-iptables.

The command supports the following checks:

--fence-node *NODE*
: Fences a specific node passed from the command line.

--kill-sbd/--kill-corosync/ --kill-pacemakerd
: Kills the daemons for SBD, Corosync, or Pacemaker. After running one of these tests, you can find a report in the directory /var/lib/crmsh/crash_test/. The report includes a test case description, action logging, and an explanation of possible results.

--split-brain-iptables
: Simulates a split-brain scenario by blocking the Corosync port, and checks whether one node can be fenced as expected. You must install iptables before you can run this test.

For more information, run the **crm cluster crash_test --help** command.

This example uses nodes called alice and bob, and tests fencing bob. To watch bob change status during the test, you can log in to Hawk and navigate to *Status* › *Nodes*, or run **crm status** from another node.

EXAMPLE 1: MANUALLY TRIGGERING NODE FENCING

```
admin@alice> sudo crm cluster crash_test --fence-node bob
```

```
==============================================
Testcase:           Fence node bob
Fence action:       reboot
Fence timeout:      95

!!! WARNING WARNING WARNING !!!
THIS CASE MAY LEAD TO NODE BE FENCED.
TYPE Yes TO CONTINUE, OTHER INPUTS WILL CANCEL THIS CASE [Yes/No](No): Yes
INFO: Trying to fence node "bob"
INFO: Waiting 95s for node "bob" reboot...
INFO: Node "bob" will be fenced by "alice"!
INFO: Node "bob" was fenced by "alice" at DATE TIME
```

6 Legal Notice

A GNU Free Documentation License

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

HA glossary

active/active, active/passive

How resources run on the nodes. Active/passive means that resources only run on the active node, but can move to the passive node if the active node fails. Active/active means that all nodes are active at once, and resources can run on (and move to) any node in the cluster.

arbitrator

An *arbitrator* is a machine running outside the cluster to provide an additional instance for cluster calculations. For example, *QNetd* provides a vote to help *QDevice* participate in *quorum* decisions.

CIB (cluster information base)

An XML representation of the whole cluster configuration and status (cluster options, nodes, resources, constraints and the relationships to each other). The CIB manager (`pacemaker-based`) keeps the CIB synchronized across the cluster and handles requests to modify it.

clone

A *clone* is an identical copy of an existing node, used to make deploying multiple nodes simpler.

In the context of a cluster *resource*, a clone is a resource that can be active on multiple nodes. Any resource can be cloned if its resource agent supports it.

cluster

A *high-availability* cluster is a group of servers (physical or virtual) designed primarily to secure the highest possible availability of data, applications and services. Not to be confused with a *high-performance* cluster, which shares the application load to achieve faster results.

Cluster logical volume manager (Cluster LVM)

The term *Cluster LVM* indicates that LVM is being used in a cluster environment. This requires configuration adjustments to protect the LVM metadata on shared storage.

cluster partition

A cluster partition occurs when communication fails between one or more nodes and the rest of the cluster. The nodes are split into partitions but are still active. They can only communicate with nodes in the same partition and are unaware of the separated nodes. This is known as a *split brain* scenario.

cluster stack

The ensemble of software technologies and components that make up a cluster.

colocation constraint

A type of *resource constraint* that specifies which resources can or cannot run together on a node.

concurrency violation

A resource that should be running on only one node in the cluster is running on several nodes.

Corosync

Corosync provides reliable messaging, membership and quorum information about the cluster. This is handled by the Corosync Cluster Engine, a group communication system.

CRM (cluster resource manager)

The management entity responsible for coordinating all non-local interactions in a High Availability cluster. SUSE Linux Enterprise High Availability uses *Pacemaker* as the CRM. It interacts with several components: local executors on its own node and on the other nodes, non-local CRMs, administrative commands, the fencing functionality, and the membership layer.

`crmsh` (CRM Shell)

The command-line utility `crmsh` manages the cluster, nodes and resources.

Csync2

A synchronization tool for replicating configuration files across all nodes in the cluster.

DC (designated coordinator)

The `pacemaker-controld` daemon is the cluster controller, which coordinates all actions. This daemon has an instance on each cluster node, but only one instance is elected to act as the DC. The DC is elected when the cluster services start, or if the current DC fails or leaves the cluster. The DC decides whether a cluster-wide change must be performed, such as fencing a node or moving resources.

disaster

An unexpected interruption of critical infrastructure caused by nature, humans, hardware failure, or software bugs.

disaster recovery

The process by which a function is restored to the normal, steady state after a disaster.

Disaster Recovery Plan

A strategy to recover from a disaster with the minimum impact on IT infrastructure.

DLM (Distributed Lock Manager)

DLM coordinates accesses to shared resources in a cluster, for example, managing file locking in clustered file systems to increase performance and availability.

DRBD

DRBD® is a block device designed for building High Availability clusters. It replicates data on a primary device to secondary devices in a way that ensures all copies of the data remain identical.

existing cluster

The term *existing cluster* is used to refer to any cluster that consists of at least one node. An existing cluster has a basic *Corosync* configuration that defines the communication channels, but does not necessarily have resource configuration yet.

failover

Occurs when a resource or node fails on one machine and the affected resources move to another node.

failover domain

A named subset of cluster nodes that are eligible to run a resource if a node fails.

fencing

Prevents access to a shared resource by isolated or failing cluster members. There are two classes of fencing: *resource-level* fencing and *node-level* fencing. Resource-level fencing ensures exclusive access to a resource. Node-level fencing prevents a failed node from accessing shared resources and prevents resources from running on a node with an uncertain status. This is usually done by resetting or powering off the node.

GFS2

Global File System 2 (GFS2) is a shared disk file system for Linux computer clusters. GFS2 allows all nodes to have direct concurrent access to the same shared block storage. GFS2 has no disconnected operating mode, and no client or server roles. All nodes in a GFS2 cluster function as peers. GFS2 supports up to 32 cluster nodes. Using GFS2 in a cluster requires hardware to allow access to the shared storage, and a lock manager to control access to the storage.

group

Resource groups contain multiple resources that need to be located together, started sequentially and stopped in the reverse order.

Hawk (HA Web Konsole)

A user-friendly Web-based interface for monitoring and administering a High Availability cluster from Linux or non-Linux machines. Hawk can be accessed from any machine that can connect to the cluster nodes, using a graphical Web browser.

heuristics

QDevice supports using a set of commands (*heuristics*) that run locally on start-up of cluster services, cluster membership change, successful connection to the *QNetd* server, or optionally at regular times. The result is used in calculations to determine which partition should have *quorum*.

knet (kronosnet)

A network abstraction layer supporting redundancy, security, fault tolerance, and fast fail-over of network links. In SUSE Linux Enterprise High Availability 16, *knet* is the default transport protocol for the *Corosync* communication channels.

local cluster

A single cluster in one location (for example, all nodes are located in one data center). Network latency is minimal. Storage is typically accessed synchronously by all nodes.

local executor

The local executor is located between *Pacemaker* and the resources on each node. Through the `pacemaker-execd` daemon, Pacemaker can start, stop and monitor resources.

location

In the context of a whole cluster, *location* can refer to the physical location of nodes (for example, all nodes might be located in the same data center). In the context of a *location constraint*, *location* refers to the nodes on which a resource can or cannot run.

location constraint

A type of *resource constraint* that defines the nodes on which a resource can or cannot run.

meta attributes (resource options)

Parameters that tell the *CRM (cluster resource manager)* how to treat a specific *resource*. For example, you might define a resource's priority or target role.

metro cluster

A single cluster that can stretch over multiple buildings or data centers, with all sites connected by Fibre Channel. Network latency is usually low. Storage is frequently replicated using mirroring or synchronous replication.

network device bonding

Network device bonding combines two or more network interfaces into a single bonded device to increase bandwidth and/or provide redundancy. When using *Corosync*, the bonded device is not managed by the cluster software. Therefore, the bonded device must be configured on every cluster node that might need to access it.

node

Any server (physical or virtual) that is a member of a cluster.

order constraint

A type of *resource constraint* that defines the sequence of actions.

Pacemaker

Pacemaker is the *CRM (cluster resource manager)* in SUSE Linux Enterprise High Availability, or the “brain” that reacts to events occurring in the cluster. Events might be nodes that join or leave the cluster, failure of resources, or scheduled activities such as maintenance, for example. The `pacemakerd` daemon launches and monitors all other related daemons.

parameters (instance attributes)

Parameters determine which instance of a service the *resource* controls.

primitive

A primitive resource is the most basic type of cluster resource.

promotable clone

Promotable clones are a special type of *clone* resource that can be promoted. Active instances of these resources are divided into two states: promoted and unpromoted (also known as “active and passive” or “primary and secondary”).

QDevice

QDevice and *QNetd* participate in *quorum* decisions. The `corosync-qdevice` daemon runs on each cluster node and communicates with QNetd to provide a configurable number of votes, allowing a cluster to sustain more node failures than the standard quorum rules allow.

QNetd

QNetd is an *arbitrator* that runs outside the cluster. The `corosync-qnetd` daemon provides a vote to the `corosync-qdevice` daemon on each node to help it participate in quorum decisions.

quorum

A *cluster partition* is defined to have quorum (be *quorate*) if it has the majority of nodes (or “votes”). Quorum distinguishes exactly one partition. This is part of the algorithm to prevent several disconnected partitions or nodes (“split brain”) from proceeding and causing data and service corruption. Quorum is a prerequisite for fencing, which then ensures that quorum is unique.

RA (resource agent)

A script acting as a proxy to manage a *resource* (for example, to start, stop or monitor a resource). SUSE Linux Enterprise High Availability supports different kinds of resource agents.

ReaR (Relax and Recover)

An administrator tool set for creating *disaster recovery* images.

resource

Any type of service or application that is known to *Pacemaker*, for example, an IP address, a file system, or a database. The term *resource* is also used for *DRBD*, where it names a set of block devices that use a common connection for replication.

resource constraint

Resource constraints specify which cluster nodes resources can run on, what order resources load in, and what other resources a specific resource is dependent on.

See also *colocation constraint*, *location constraint* and *order constraint*.

resource set

As an alternative format for defining location, colocation or order constraints, you can use *resource sets*, where primitives are grouped together in one set. When creating a constraint, you can specify multiple resources for the constraint to apply to.

resource template

To help create many resources with similar configurations, you can define a resource template. After being defined, it can be referenced in primitives or in certain types of constraints. If a template is referenced in a primitive, the primitive inherits all operations, instance attributes (parameters), meta attributes and utilization attributes defined in the template.

SBD (STONITH Block Device)

SBD provides a node *fencing* mechanism through the exchange of messages via shared block storage. Alternatively, it can be used in diskless mode. In either case, it needs a hardware or software *watchdog* on each node to ensure that misbehaving nodes are really stopped.

scheduler

The scheduler is implemented as `pacemaker-schedulerd`. When a cluster transition is needed, `pacemaker-schedulerd` calculates the expected next state of the cluster and determines what actions need to be scheduled to achieve the next state.

split brain

A scenario in which the cluster nodes are divided into two or more groups that do not know about each other (either through a software or hardware failure). *STONITH* prevents a split-brain scenario from badly affecting the entire cluster. Also known as a *partitioned cluster* scenario.

The term *split brain* is also used in *DRBD* but means that the nodes contain different data.

SPOF (single point of failure)

Any component of a cluster that, if it fails, triggers the failure of the entire cluster.

STONITH

Another term for the *fencing* mechanism that shuts down a misbehaving node to prevent it from causing trouble in a cluster. In a *Pacemaker* cluster, node fencing is managed by the fencing subsystem `pacemaker-fenced`.

switchover

The planned moving of resources to other nodes in a cluster. See also *failover*.

utilization

Tells the CRM what capacity a certain *resource* requires from a node.

watchdog

SBD (STONITH Block Device) needs a watchdog on each node to ensure that misbehaving nodes are really stopped. SBD “feeds” the watchdog by regularly writing a service pulse to it. If SBD stops feeding the watchdog, the hardware enforces a system restart. This protects against failures of the SBD process itself, such as becoming stuck on an I/O error.