OviOS Cluster

Administrator guide: configuration, fencing, quorum and failover

Covers the `ovios-cluster` engine at commit `eb8a654` (October 2026) and later. Written for the person who configures and operates the cluster.

This guide describes what the code does. Where a behaviour has been tested only with simulated hardware, or not tested on real hardware yet, it says so (see the last appendix).

> **Note** Contents: 1 Editing the configuration and every setting; 2 Fencing and the watchdog; 3 Tiebreaker or witness; 4 How the cluster decides (the algorithm); 5 Limits; 6 Quorum and why the number of voters should be odd; Appendices: command cheat sheet, log events, timeline with default numbers, example configurations, what is verified.

1. Editing the configuration

1.1 The workflow

The cluster configuration lives in `/etc/ovios-cluster/ovios-cluster.yaml` on every node. It is edited **only on the primary** and pushed from there to all other nodes (including the witness). In the OviOS shell, on the primary:

```
service ovios-cluster edit
```

What the command does, in order:

1. Checks that this node is the primary. On a replica it refuses and tells you which node is the primary.
2. Opens a **private copy** of the file in your editor (`$VISUAL`, then `$EDITOR`, then `nano`, then `vi`). The live file is not touched while you edit.
3. When you close the editor it validates the copy with `ovios-cluster sync --check --file <copy>`: YAML syntax, unknown keys, the consistency rules in this chapter, and the licence. If the copy is invalid, nothing is saved; you can edit again or discard.
4. If it is valid, it replaces the live file atomically and keeps the previous version as `ovios-cluster.yaml.bak`.
5. Asks whether to push it now. Pushing runs `ovios-cluster sync`, which copies the file to every node.

If somebody else changed the live file while you were editing, the command refuses to overwrite it and keeps your version next to a `.kept` file.

The same thing without the shell wrapper:

```
vi /etc/ovios-cluster/ovios-cluster.yaml
ovios-cluster sync --check     # validate only, pushes nothing
ovios-cluster sync             # validate and push to every node
```

What happens after a push

- Each daemon notices the new file on its next tick (a couple of seconds) and applies it. No restart is needed.
- The **YAML is strict**: an unknown key makes older daemons reject the whole file. Therefore upgrade **all** nodes, witness included, to the same version before you add a new key (for example `fencing:`).
- The daemon on each node validates the file again. A node that cannot accept it keeps running on the previous configuration and logs why.

> **Note** Changing `nodes`, `vip` or `interface` on a running cluster is possible but is best done with `ovios-cluster add` / `remove`, which also prepare ssh keys and certificates. Never edit the epoch/state file (`/var/lib/ovios-cluster/ovios-cluster.state`) by hand.

1.2 A complete example

This is a typical two-data-node cluster with a witness, VIP, SMB/NFS/iSCSI services and a watchdog. Every key is explained in section 1.3.

```
cluster_name: ovios-cluster
cluster_id: bdfefe9d-f412-4fd7-b80c-fd225f92519e   # assigned by init, never change
vip: 10.0.0.108
interface: bond0
vip_check_ports: [445, 2049, 3260]
vip_failure_seconds: 60
heartbeatport: "9064"

automatic_failover: true
heartbeat_interval: 2
lease_seconds: 15
max_replica_lag_seconds: 300
# tiebreaker: 10.0.0.1          # alternative to a witness node, see chapter 3

nodes:
  - id: 8e990c6e-cdef-4e0f-8898-9fe0890bb8d8
    ip: 10.0.0.107
```

```
    role: replica                # informational; the state file says who is primary
    fence_methods:
      - name: ssh
        cmd: ssh -o BatchMode=yes -o ConnectTimeout=3 root@$OVIOS_FENCE_IP "nohup sh -c 'sleep 1;
poweroff -f' >/dev/null 2>&1 &"
        timeout_seconds: 10
  - id: b1201ab3-8bf4-4f41-ad96-048dfea8ad8e
    ip: 10.0.0.134
    role: replica
    fence_methods:
      - name: ssh
        cmd: ssh -o BatchMode=yes -o ConnectTimeout=3 root@$OVIOS_FENCE_IP "nohup sh -c 'sleep 1;
poweroff -f' >/dev/null 2>&1 &"
        timeout_seconds: 10
  - id: 5d1f0c3a-0000-0000-0000-000000000003
    ip: 10.0.0.3
    role: witness                # votes only; holds no data

fencing:
  isolate_on_lease_loss: true
  failsafe_on_crash: true
  on_fence_failure: refuse       # or promote
  watchdog:
    enabled: true
    dry_run: false
    device: /dev/watchdog
    timeout_seconds: 10

datasets:
  exclude: [bpool, bpool/BOOT, rpool, rpool/HOME, rpool/ROOT]
replication:
  interval: 5
  snapshot_prefix: ovios
  workers: 4
  timeout_seconds: 3600
  retention: { keep_last: 12 }
  auto_rollback: true
  independent_datasets: true

services: [ovios-smb.service, ovios-nfs.service, ovios-iscsi.service]
hooks: { pre_promote: "", post_promote: "", pre_demote: "", post_demote: "" }
metrics_listen: ":9100"
binary_path: /usr/bin/ovios-cluster
```

1.3 Reference: every setting

"Default" is what the engine uses when the key is absent. Values are applied without a restart, except `heartbeatport` and `metrics_listen`, which are read when the daemon starts (`service ovios-cluster restart`).

Identity and network

Key	Default	Meaning and dependencies
`cluster_name`	set by init	Label shown in status and passed to hooks as `OVIOS_CLUSTER`.
`cluster_id`	set by init	Identifies this cluster for licence binding. Never change it; replacing nodes does not change it.
`heartbeatport`	`"9064"`	TCP port of the mutually authenticated (mTLS) heartbeat. Each node listens on its **own IP only**. Must be open between all nodes and the witness. Quote the number.
`nodes[].id`	required	Unique node id (a UUID kept in `/etc/ovios-cluster/node-id`). Commands accept id or IP.
`nodes[].ip`	required	Unique IPv4 address of the node, used for heartbeats and ssh.
`nodes[].role`	-	`primary` / `replica` are **informational only**: who is primary is decided by the epoch in the state file. `witness` is real and static: a vote-only node.

Key	Default	Meaning and dependencies
`nodes[].interface`	top-level `interface`	Overrides the VIP interface for this node when its NIC is named differently.

Virtual IP

Key	Default	Meaning and dependencies
`vip`	empty	The floating address clients use. The primary holds it and announces it (gratuitous ARP). Empty = the engine does not manage an address. A bare IP gets /32 (or /128).
`interface`	empty	Interface that carries the VIP. **Required when `vip` is set.**
`vip_check_ports`	none	TCP ports clients use on the VIP (SMB 445, NFS 2049, iSCSI 3260 are only the usual ones). If listed, the primary checks that each accepts a connection on the VIP. Requires `vip`; each port 1-65535, once.
`vip_failure_seconds`	60 (minimum 30)	How long VIP trouble (address not assignable, a checked port not answering, link down) may last before the primary **hands the role over** to a healthy replica. A missing interface or lost carrier is acted on after about 15 s (or this value if smaller). Needs `automatic_failover`.
`vip_handover_cooldown_seconds`	0 = 600	A node that took part in an automatic VIP handover will not start another for this long, so the role cannot bounce between two nodes that both have a VIP problem. Failovers and manual switchovers are not affected.

Failover timing and policy

Key	Default	Meaning and dependencies
`automatic_failover`	false	Lets a replica promote itself when the primary is gone. **Licensed feature.** Without a valid licence the cluster behaves as if it were false for promotion only (a primary still steps down, isolates itself and fences; replicas just wait for a manual `promote`). Accepted only if: a quorum can be formed (3+ voters), there are at least 2 data nodes, and every data node has a fence method, unless `fencing.on_fence_failure: promote` or a real watchdog is configured.
`heartbeat_interval`	2 s	Seconds between heartbeats and between engine ticks. A peer counts as **alive** if it answered within 3 intervals + 1 s (7 s by default).
`lease_seconds`	15 s	The primary lease. A primary that cannot prove a majority of voters for this long stops serving. **Must be at least 3 x `heartbeat_interval`.** Replicas wait this plus half a lease (22.5 s) before promoting.
`max_replica_lag_seconds`	300	A replica refuses to promote itself if its newest replicated snapshot is older than this (or if it has none). `promote --force` skips the check.
`tiebreaker`	empty	An IP address (usually the gateway) that every node pings once per heartbeat round; an answer counts as one extra vote. Alternative to a witness node. See chapter 3 for the important differences.

Fencing and watchdog

Explained in depth in chapter 2; listed here for completeness.

Key	Default	Meaning and dependencies
`nodes[].fence_cmd`	none	One shell command that powers this node off (typically IPMI). 90 s timeout. Counts as one fence method.
`nodes[].fence_methods[]`	none	Several independent ways to fence the node, all run at once; the first success wins. Each: `name` (required, unique per node), `cmd` (required), `timeout_seconds` (default 20).
`fencing.isolate_on_lease_loss`	true	The primary drops the VIP and kills the services as soon as it has been without quorum for a full lease. Needs a possible quorum (3+ voters).
`fencing.failsafe_on_crash`	true	If the daemon dies abnormally on the primary, systemd runs `ovios-cluster failsafe`, which drops the VIP and kills the services. Needs `automatic_failover` and the drop-in `failsafe.conf`.
`fencing.on_fence_failure`	`refuse`	`refuse` or `promote`. What an automatic promotion does when the old primary could not be fenced (or has no fence method). A real watchdog implies `promote`.
`fencing.watchdog.enabled`	false	Arm the watchdog device while this node is primary. Needs a possible quorum.
`fencing.watchdog.device`	`/dev/watchdog`	The watchdog device node. Use `/dev/watchdog1` etc. if the one you want is not the first.
`fencing.watchdog.timeout_seconds`	10	Programmed into the device. At least 2 and at most `lease_seconds - 2 x heartbeat_interval - 1` (10 with the defaults).
`fencing.watchdog.dry_run`	false	Log when the watchdog would stop being fed, but never open the device. Does not count as self-fencing.

Data: datasets and replication

Key	Default	Meaning and dependencies
`datasets.exclude`	none	Dataset names (and everything below them) that are never replicated. The receive guard on replicas refuses them too. **List the system pools** (for example `rpool`, `bpool` and their children) so they are never sent. `<pool>/.ovios-system`, the per-node metadata dataset, is always excluded whatever you write.
`replication.exclude`	none	Legacy; merged with `datasets.exclude`.
`replication.interval`	10 s	Seconds between replication cycles. This is also roughly your **recovery point**: an unplanned failover can lose up to one interval of writes plus whatever was in flight.
`replication.snapshot_prefix`	`ovios`	Name prefix of the engine's snapshots (`<prefix>-...`). Do not use it for your own snapshots.
`replication.workers`	4	Parallel dataset transfers.
`replication.timeout_seconds`	3600	Maximum time of one dataset transfer.
`replication.retention.keep_last`	12	Newest replication snapshots kept (pinned ones are always kept).
`replication.auto_rollback`	false	Lets the new primary roll back a returning **former** primary whose own post-failover writes were never replicated. **This discards those writes.** Off = the node is reported DIVERGED until you run `ovios-cluster resync <node> --discard-divergent-data`.

Key	Default	Meaning and dependencies
`replication.independent_datasets`	false	Each dataset transfers on its own, so a slow dataset does not hold up the others. Bounded by `workers`.

Services, hooks, metrics

Key	Default	Meaning and dependencies
`services`	none	Ordered list of systemd units. Started in this order on promotion; stopped in **reverse** order on an orderly demotion; stopped and SIGKILLed all at once by fast isolation and by the failsafe. Put dependencies first (for example a web UI last).
`hooks.pre_promote`	none	Command run before taking over. **If it fails, the promotion is aborted.**
`hooks.post_promote`	none	After taking over. A failure is logged and shown as a note.
`hooks.pre_demote` / `post_demote`	none	Before and after stepping down. Failures are logged.
`metrics_listen`	empty	Address for Prometheus metrics, for example `":9100"` (all addresses) or `"127.0.0.1:9100"`. Empty disables.
`binary_path`	`/usr/bin/ovios-cluster`	Where the binary lives on every node; the receive guard is run over ssh by this path.

Hooks run through `sh -c`, with a 90 second timeout, and see `OVIOS_NODE_ID`, `OVIOS_ROLE`, `OVIOS_EPOCH`, `OVIOS_VIP` and `OVIOS_CLUSTER` in the environment.

1.4 How the settings depend on each other

If you set...	...then
`vip`	`interface` is required. `vip_check_ports` is allowed. VIP handover (`vip_failure_seconds`) becomes active with automatic failover.
`automatic_failover: true`	Needs (a) a **quorum**: a witness node or a `tiebreaker` (3+ voters), (b) at least **2 data nodes**, (c) per data node a `fence_cmd` / `fence_methods` **or** `fencing.on_fence_failure: promote` **or** a real watchdog, (d) a valid **licence**.
`lease_seconds`	Must be >= 3 x `heartbeat_interval`. Also bounds `fencing.watchdog.timeout_seconds` (at most lease - 2 x heartbeat - 1). Lowering the lease makes failover faster **and** makes false failovers on network hiccups more likely.
`fencing.watchdog.enabled`	Needs 3+ voters. Counts as a fence (self-fencing) unless `dry_run`. Implies `on_fence_failure: promote`. Needs a working watchdog device on **every** data node (any of them can become primary).
`fencing.failsafe_on_crash`	Only useful with `automatic_failover` and the systemd drop-in `failsafe.conf` installed on every data node.
`fencing.isolate_on_lease_loss`	Only acts with 3+ voters (with two voters the engine cannot tell a dead peer from a partition, so a lease means nothing).
`services`	These units are killed by isolation and the failsafe. Do not list units that must keep running on a replica.
`replication.auto_rollback`	Convenient and destructive: pairs well with `interval` small, badly with workloads that cannot lose the last seconds of writes.

If you set...	...then
`datasets.exclude`	Must contain the system pools and anything node-specific. A dataset excluded here is also refused by the receive guard.

1.5 Common mistakes the validator catches

- `automatic_failover requires fence_cmd or fence_methods on every data node ...` : add a fence method, or choose `on_fence_failure: promote` / a real watchdog.
- `automatic_failover needs a way to form quorum` : add a witness (`ovios-cluster add --witness <ip>`) or set `tiebreaker`.
- `lease_seconds (N) must be at least 3x heartbeat_interval` : raise the lease or lower the interval.
- `vip is set but interface is empty`, `vip_check_ports ... without vip`, `vip_failure_seconds ... at least 30`.
- `fencing.watchdog.timeout_seconds (N) is too long ...` : the node would be reset while it still has quorum.
- Unknown key: usually a typo, or a node still running an older version.

2. Fencing and the watchdog

2.1 What fencing is for

OviOS clusters use **shared-nothing** replication: every node has its own copy of the data, replicated with ZFS send/receive. Two nodes can never corrupt one disk. The danger of a failover is therefore not damaged storage; it is **two primaries serving clients at the same time** (split brain), which makes the copies diverge and loses writes.

So "fencing" in this engine means: **make sure the old primary has stopped serving before the new one starts**. There are two ways to be sure:

- **Outside fencing**: another node (or device) switches the old primary off or cuts it from the network: IPMI, a smart power plug or PDU, an ssh `poweroff`. This is the classic STONITH ("shoot the other node in the head").
- **Self-fencing**: the old primary stops itself. After one lease without quorum it drops the VIP and kills its services; with a watchdog it also resets the whole machine. Replicas wait longer than the lease before promoting, so the old primary is already out of service when they act.

The engine uses both, in layers.

2.2 The layers, in the order they act

#	Layer	What it does and when
1	Fast isolation (always on)	A guard checks the lease every 500 ms on its own clock. A primary that has gone a full `lease_seconds` without proof that a majority of voters is reachable **drops the VIP and queues a stop plus SIGKILL for every unit in `services`, in parallel**, without waiting for orderly shutdowns. Setting: `fencing.isolate_on_lease_loss`.
2	Failsafe on daemon death	If the daemon is killed or crashes on the primary, nothing is left to enforce the lease. systemd then runs `ovios-cluster failsafe` (drop-in `failsafe.conf`), which drops the VIP and kills the services. It does nothing after a clean stop (`service ovios-cluster stop/restart`, upgrades), on a node that is not primary, or without `automatic_failover`. Setting: `fencing.failsafe_on_crash`.
3	Watchdog	While the node is primary it keeps a watchdog device armed and feeds it only while it can prove quorum. If it loses quorum, hangs, or the daemon dies, the **machine resets itself** within one lease. See 2.6.
4	Outside fencing before promotion	The promoting replica runs the old primary's `fence_cmd` / `fence_methods` before it bumps the epoch. See 2.3-2.5.
5	Epoch and receive guard	Even if everything above failed, a higher epoch always wins: the old primary demotes when it hears it, and replicas refuse replication streams from a sender with an older epoch (`recv-guard`, exit 3). This protects the data, not the clients.

2.3 fence_cmd and fence_methods: what they are and how they run

Both are shell commands, stored **on the node to be fenced** (the entry in `nodes:` describes the victim) and run **on the promoting node**, through `sh -c`, with these variables in the environment:

Variable	Value
`OVIOS_FENCE_NODE`	Id of the node being fenced.
`OVIOS_FENCE_IP`	Its IP address (from `nodes[].ip`). **You never set this yourself.**
`OVIOS_FENCE_METHOD`	Name of the method running (`fence_cmd` for the legacy key).

Variable	Value
`OVIOS_FENCE_TEST`	`1` only when started by `ovios-cluster fence-test --execute`.

fence_cmd: one command

```
nodes:
  - id: ...
    ip: 10.0.1.1
    fence_cmd: ipmitool -I lanplus -H 10.0.1.101 -U admin -P secret chassis power off
```

Use it when you have exactly one reliable way to switch the node off (most often IPMI/iDRAC/iLO). It has a 90 second timeout and counts as success when the command exits 0.

fence_methods: several commands in parallel

```
    fence_methods:
      - name: plug
        cmd: curl -s -m 5 http://10.0.2.5/relay/0?turn=off     # smart plug
        timeout_seconds: 10
      - name: ssh
        cmd: ssh -o BatchMode=yes -o ConnectTimeout=3 -o
UserKnownHostsFile=/etc/ovios-cluster/known_hosts root@$OVIOS_FENCE_IP "nohup sh -c 'sleep 1;
poweroff -f' >/dev/null 2>&1 &"
        timeout_seconds: 10
```

Use it when you have more than one way to fence a node, or want a fallback. **All methods start at the same moment and the first one that succeeds wins**; the promotion continues at once. A method that hangs or fails never delays one that works. Methods still running when one succeeds are left to finish within their own timeouts. If every method fails, the fence has failed (see 2.4).

	fence_cmd	fence_methods
Number of ways	One command	Any number, each named
Execution	Single	All in parallel; first success wins
Timeout	90 s	Per method, default 20 s
Typical use	IPMI / BMC	Smart plug, PDU, ssh poweroff, a BMC plus a fallback
Can be combined	Yes: it runs as one more method next to `fence_methods`	

Important If you combine a fast placeholder (for example `fence_cmd: echo fenced`) with a real method, the placeholder wins instantly and the real method becomes best-effort in the background. **Never leave a fake method configured** in a cluster that relies on fencing.

Writing a good method

- It must exit **0 only if the node is, or will certainly be, off**. A non-zero exit or a timeout is a failure.
- `ssh ... poweroff -f` powers the machine off without closing the connection, so ssh hangs until it times out and the method is wrongly counted as failed. Use the `nohup sh -c ...&` form shown above: ssh returns as soon as the command has been delivered.
- ssh must be non-interactive: `-o BatchMode=yes`, `-o ConnectTimeout=3`, and a known-hosts file that already contains the target (`/etc/ovios-cluster/known_hosts` holds the pinned keys).
- An ssh method **cannot work when the node is cut off from the network**. Pair it with an out-of-band method (BMC, plug, PDU) or with the watchdog.
- Keep timeouts short: the failover waits for the first success, so every second of a method's runtime is a second of outage.

Testing your methods: ovios-cluster fence-test

```
ovios-cluster fence-test <node-id-or-ip>            # checks each method's program exists; runs
nothing
ovios-cluster fence-test <node-id-or-ip> --execute   # RUNS every method in parallel and POWERS THE
NODE OFF
```

With `--execute` it reports, per method, whether it worked and how long it took, then watches whether the node stopped answering. The failover waits for the fastest working method, so that is the time it would have cost. Run it from another node, on a replica; power the node on again afterwards. It refuses to fence the node this machine is, a witness, or the current primary (unless `--force`, which makes the cluster fail over).

When is a fence run?

- **Automatic failover and manual `promote`**: just before the new epoch is created, after the pre_promote hook. A recent successful fence of a node that is still down is not repeated for 10 minutes.
- **Never** for a planned `switchover` or a VIP handover: the old primary is alive and has stopped serving on its own.
- Nothing is fenced when the node has no fence method; a manual promote then only warns, an automatic one follows `on_fence_failure` (next section).

2.4 When fencing fails: on_fence_failure

Suppose the old primary's network card died. The ssh method cannot reach it, there is no BMC, so every method fails. What should the replica do?

Value	Behaviour
`refuse` (default)	The promotion is refused: "refusing to promote while the old primary may still be writing". The replica retries after 10 s, 20 s, 40 s, then every minute, forever. The cluster stays down until the fence succeeds or you run `ovios-cluster promote --force` on the replica after confirming the old primary is really down. **Safest, but it can leave the cluster without a primary.**
`promote`	The replica goes ahead, and logs `FENCING <node> FAILED (...); promoting anyway because ...`. The automatic promotion happens only after the old primary has been silent for more than a lease; by then (with fast isolation, the failsafe and, if enabled, the watchdog) it has cut itself off. Choose this for shared-nothing clusters that accept self-fencing as the fence. With `promote`, `automatic_failover` no longer requires any fence method.

- A **real watchdog** (`enabled: true`, not `dry_run`) implies `promote` automatically.
- `on_fence_failure` affects **automatic** promotions only. A manual `promote` without `--force` always refuses after a failed fence.
- Your fence methods are still **tried first**; the policy decides only what happens if they all fail.

Important What `promote` does not cover: an old primary whose **daemon hangs but whose kernel keeps running**. It keeps its VIP and services, so clients keep writing to it while the replica promotes. A watchdog closes that gap, because a hung daemon stops feeding it and the machine resets. A completely frozen kernel cannot serve clients, so it is harmless to clients (only a hardware watchdog or outside fencing resets it).

2.5 Which to choose

Your situation	Recommended setup
Servers with IPMI / iDRAC / iLO	`fence_cmd` with `ipmitool ... power off`. Add `fence_methods` with an ssh fallback if you like. Keep `on_fence_failure: refuse`. Watchdog optional.
Commodity hardware with a smart plug / PDU	`fence_methods`: plug (out-of-band) + ssh (fast when the node is reachable). Add the watchdog if the machine has a working one.

Your situation	Recommended setup
No fencing hardware at all	Real watchdog (`softdog` if there is no working hardware timer) with `on_fence_failure: promote`, plus an ssh method as a bonus. State the risk to yourself: a frozen kernel is only covered by a hardware watchdog.
Two nodes + tiebreaker	Fencing that works **over a different path than the heartbeat** is strongly recommended (see chapter 3).
Shared storage (not the OviOS model)	Not supported by this engine. Do not use `on_fence_failure: promote` if two nodes could ever write to the same disk.

2.6 The watchdog in detail

What a watchdog is

A watchdog is a countdown timer that **resets the machine** when it reaches zero, unless software keeps restarting ("feeding") it. On a server it is usually a hardware timer in the chipset (for example Intel `iTCO_wdt`, AMD `sp5100_tco`, an IPMI watchdog); `softdog` is a software emulation inside the kernel. The kernel exposes it as `/dev/watchdog`.

How ovios-cluster uses it

1. The watchdog is **armed only while this node is the primary**. A replica never opens the device.
2. Once armed, the daemon programs the timeout (`timeout_seconds`, default 10 s) and **feeds the device once per second, but only while quorum evidence is fresh**: the age of the proof that a majority of voters is reachable must be at most `lease_seconds - timeout_seconds`.
3. If the node loses quorum, hangs, or the daemon dies, feeding stops. The hardware resets the machine within `timeout_seconds` of the last feed.
4. On an **orderly** stand-down (planned switchover, demotion that completes) and on a **clean daemon stop or restart**, the daemon disarms the watchdog with the "magic close" character, so upgrades never reset a healthy machine.
5. A daemon that **dies** (crash, `kill -9`) leaves the device armed on purpose. The machine resets after `timeout_seconds` unless a restarted daemon re-arms and feeds it in time.

The timing guarantee

The device resets `T` (= `timeout_seconds`) after the last feed. The daemon feeds only if the evidence of quorum is at most `L - T` old (`L` = `lease_seconds`). So the last feed happens at most `L - T` after the last proof of quorum, and the reset follows at most `T` later:

```
reset happens no later than (L - T) + T = L = lease_seconds after the last proof of quorum
```

That is **before** a replica may promote (it waits `L` plus half a lease, counted from its own last heartbeat from the primary). With the defaults (`L` = 15 s, `T` = 10 s):

Time after the last proof of quorum	What happens on the isolated primary
0 - 5 s	Evidence younger than 5 s: the watchdog is fed every second.
5 s	Evidence older than `L - T` = 5 s: feeding stops (log: `watchdog.starve`).
up to 15 s	The hardware timer expires: the machine resets. In the same second the lease guard (layer 1) isolates the node.
22.5 s	The replica may promote (if it holds quorum itself). The old primary is already gone.

> **Important** This relies on the device really resetting `T` seconds after the last feed. Some hardware may reset only after **two** timer periods (not verified here), and some firmware leaves the timer **frozen** so it never resets at all (observed on a Lenovo with `iTCO_wdt`, where the time left never decreased). Test every model (section 2.8). The daemon checks the timer when it arms: if the time left does not count down it reports `DEGRADED: watchdog: the timer does not count down ...` in `status`.

The settings

Setting	Notes
`enabled`	Turns the feature on. Every data node needs a working device, because any of them can become primary.
`device`	Default `/dev/watchdog`. If a hardware driver and `softdog` are both loaded, the first registered owns `/dev/watchdog`; the other is `/dev/watchdog1`. Prefer to blacklist the one you do not use.
`timeout_seconds`	Default 10, minimum 2, maximum `lease_seconds - 2 x heartbeat_interval - 1`. A larger value would reset the node while it still has quorum (normal evidence is up to about two heartbeats old). Smaller is faster to react but less tolerant of hiccups.
`dry_run`	Do everything except open the device; log `watchdog.starve` when the feed would stop. Use it to check the timings before you arm the real thing. Dry-run does not count as self-fencing.

Requirements on the machine

- Kernel: `CONFIG_WATCHDOG=y`, a driver for your hardware (`ITCO_WDT`, `SP5100_TCO`, `WDAT_WDT`, `IPMI_WATCHDOG`, ...) and/or `CONFIG_SOFT_WATCHDOG=m`, and **`CONFIG_WATCHDOG_NOWAYOUT` must be off**, otherwise a clean stop cannot disarm and every restart of the primary resets the machine.
- Nothing else may hold the device: `systemd`'s own `RuntimeWatchdogSec` must be unset; `fuser /dev/watchdog` should print nothing.
- For `softdog`: `modprobe softdog` (persist with `/etc/modules-load.d/softdog.conf`; blacklist a hardware driver whose timer does not work with `/etc/modprobe.d/`). On a PXE/live system these files must be part of the image or overlay.

What it does not do

- `softdog` is software: it cannot reset a machine whose kernel is frozen. A hardware watchdog can.
- The watchdog protects **against a primary that loses quorum or hangs**. It does not help when the primary still has quorum (see the tiebreaker case in chapter 3).
- A watchdog that could not be armed (no device, timer not counting) shows as `DEGRADED: watchdog:` in `ovios-cluster status` on that node. The replicas cannot see it, so check `status` on the primary.

2.7 Fast isolation and the failsafe, in short

- Isolation runs `ip addr del` for the VIP and, for every service in parallel, `systemctl stop --no-block <unit>` (so `Restart=` cannot revive it) immediately followed by `systemctl kill --signal=SIGKILL <unit>`. The orderly demotion (read-only datasets, hooks) follows afterwards, off the critical path.
- After an isolation the node stays **suspended**: it resumes as primary only when a majority of voters is reachable again and the cluster has not moved on to a newer epoch.
- The failsafe drop-in installs as `/usr/lib/systemd/system/ovios-cluster.service.d/failsafe.conf` (the OviOS package does this) with `ExecStopPost=-/usr/bin/ovios-cluster failsafe`. It is gated on systemd's `$SERVICE_RESULT`: only an abnormal result (signal, exit code, core dump, timeout, watchdog) triggers it.
- A quick restart of a crashed primary therefore costs a short service interruption, by design.

2.8 Testing the live fencing: summary

1. Prove the hardware: open the device without the magic character and confirm the machine resets within the expected time; read the time left twice and confirm it decreases.
2. `fence-test <node> --execute` for each node and method.
3. Dry-run watchdog: partition the primary (block heartbeats with a firewall rule that removes itself after a timeout) and watch `watchdog.starve`, `isolate` on the primary and `promote.done` on the replica, in that order.
4. Failsafe: `kill -9` the daemon on the primary; the VIP and services must drop at once.
5. Arm the real watchdog (`dry_run: false`): a clean `service ovios-cluster restart` must **not** reset the node; a partition must.

3. Tiebreaker or witness

Both exist for one reason: to give the cluster a **third vote**, so that two data nodes can tell "my partner is dead" from "I am cut off". Without a third voter no node can safely decide anything (see chapter 6). They are not equivalent.

3.1 What each one is

	Witness	Tiebreaker
What it is	A **third machine running the ovios-cluster daemon** in the `witness` role. It has no data, no VIP and is never promoted.	An **IP address** that answers ping (typically the default gateway or a switch). No software of yours runs on it.
How it votes	A normal voter in the heartbeat mesh (mutual TLS). Each data node exchanges heartbeats with it.	Every node pings it once per heartbeat round; an answer within the alive window counts as one vote. **Each node decides this for itself.**
Does it report what it sees?	**Yes.** Its heartbeat carries "I can (not) reach the primary". A replica will not promote while the witness still reaches the primary.	**No.** It only answers ping; it cannot say anything about the other node.
Configured as	`ovios-cluster add --witness <ip>`; entry in `nodes:` with `role: witness`.	`tiebreaker: <ip>` at the top level (`ovios-cluster init --tiebreaker`).
Cost	A small machine or VM (even a Raspberry Pi) that must be kept running and updated.	Nothing.
Needs a fence method?	No (it holds no data).	No.

3.2 Why the difference matters: the heartbeat-only failure

Imagine the cable or switch port **between the two data nodes** fails, but both nodes still reach the rest of the network.

With a witness

- The primary and the replica can no longer hear each other.
- The witness still hears both. Its heartbeat tells the replica "the primary is reachable".
- The replica's rule "promote only if no reachable voter still sees the primary" **blocks the promotion**. The primary keeps its lease with the witness as its second vote. Replication will fail while the link is down, but there is **no split brain**.

With a tiebreaker only

- The primary has itself + the tiebreaker = 2 of 3 votes: it keeps serving.
- The replica has itself + the tiebreaker = 2 of 3 votes, sees the primary silent for 22.5 s, and nobody can tell it that the primary is alive. It **promotes**.
- Now two nodes are primary until the old one is fenced or the link returns. The watchdog and the lease guard do not help here, because the old primary still has quorum. **Only a fence method that works over a different path (a BMC, a plug, a PDU) removes the old primary.**

> **Important** A tiebreaker is a vote that both sides of a partition can count at the same time. If you use one, fencing that does not depend on the heartbeat link is not optional. If you can afford a third machine, a witness is the stronger choice.

3.3 Which one to use

Situation	Use
You can run a third small machine in a different failure domain (another rack, another power feed, a cloud VM reachable over VPN)	**Witness.** Best protection against split brain.
Only two machines and a reliable gateway; you have a BMC/plug/PDU fence	**Tiebreaker** is acceptable. Keep `on_fence_failure: refuse`.
Only two machines, a gateway, and no fencing hardware	Tiebreaker + watchdog + `on_fence_failure: promote`, accepting the heartbeat-only case above; or add a witness.
Three data nodes	No extra voter is needed: three data nodes are three voters and tolerate the loss of one. Losing two of three loses the quorum, and a fourth voter would not change that (chapter 6).
Neither	Not allowed with `automatic_failover` (the validator refuses). Failover is then manual (`promote`).

- Choose a tiebreaker address that is **independent of the nodes**: the gateway is fine; a switch that is also the nodes' only uplink is a single point of failure.
- Do not use a tiebreaker address that sits behind one of the nodes (for example the node's own management controller).
- You can configure **both**, giving 4 voters, but that is not recommended: an even number adds no extra fault tolerance (chapter 6).

3.4 What each voter failure does (2 data nodes + 1 third voter)

Event	Result
Third voter (witness or tiebreaker) dies	The two data nodes still have 2 of 3. Nothing changes. The cluster has lost its safety margin: the next failure means no quorum.
Replica dies	Primary has itself + third voter. Keeps serving. Replication to the dead node errors until it returns.
Primary dies	Replica has itself + third voter = 2 of 3, waits 22.5 s, fences, promotes.
Both data nodes cannot see each other, third voter reaches both	Witness: no failover (it reports the primary alive). Tiebreaker: failover starts; fencing decides (3.2).
A data node is cut off from everything	It has 1 of 3: no quorum. If it is the primary it isolates itself after one lease; if it is the replica it never promotes.

4. How the cluster decides: the algorithm

4.1 The ingredients

Term	Meaning
Voter	Every entry in `nodes:` (data nodes and witnesses) plus the tiebreaker, if configured. A node always votes for itself.
Majority	More than half of all voters: 2 of 3, 3 of 4, 3 of 5. "Quorum" means "I can reach a majority".
Alive	A peer is alive if an authenticated heartbeat from it arrived within `3 x heartbeat_interval + 1 s` (7 s by default). The tiebreaker is alive if it answered ping in the same window.
Epoch	A number that increases by one at every promotion. It is written to disk **before** anything is exposed. The highest epoch always wins. Two nodes promoted at the same epoch: the one with the lexically smaller node id wins, so all nodes converge.
Lease	`lease_seconds` (15 s). The right of the primary to keep serving: renewed continuously while it reaches a majority.
Grace	Half a lease (7.5 s). The extra time a replica waits beyond the lease.
Candidate order	The order of the data nodes in `nodes:`. The first data node (other than the primary) that is alive promotes first.
Tick	The engine's main loop, every `heartbeat_interval`. The lease guard (every 500 ms) and the watchdog loop (every second) run beside it.

4.2 If you are the primary

A primary has one question: **can I still prove I reach a majority of voters?**

1. Every tick the engine counts alive voters. If the count is a majority, it refreshes "time of last quorum".
2. The **lease guard** (independent of the tick, every 500 ms) measures the exact age of the proof of quorum: the age of the freshest voter that completes a majority. When that age reaches `lease_seconds` it **isolates the node at once** (drops the VIP, kills the services). It does not apply to a planned switchover, and not to clusters that cannot form a quorum.
3. The slower path, the tick, demotes the node orderly when it has been without a majority for more than a lease: stops replication, runs `pre_demote`, stops the services in reverse order, drops the VIP, unmounts and sets the datasets read-only, runs `post_demote`. With the defaults this happens about 7 seconds later than the lease guard, because "alive" has a 7 s tolerance.
4. After stepping down for lack of quorum the node is **suspended**: it does not serve and does not take part in any election. When a majority is reachable again it resumes as primary **only if** the cluster has not moved on. If a higher epoch exists it becomes a replica instead.
5. When a node starts, it waits `3 x heartbeat_interval` before claiming the primary role, to learn about a newer epoch first.
6. A primary also stops serving (through a planned handover) if its VIP cannot be kept healthy for `vip_failure_seconds` (section 4.5).

> **Note** A cluster that cannot form a quorum (two nodes, no witness, no tiebreaker) never demotes a primary by itself and never promotes a replica automatically: it cannot tell a dead partner from a cut cable. This is why `automatic_failover` requires 3+ voters.

4.3 If you are a replica

A replica promotes itself only when **all** of the following are true, checked in this order every tick:

#	Condition	Why
1	The primary has been silent for at least `lease + grace` (22.5 s), counted from its last heartbeat	The old primary has already stepped down or been cut off (its lease is 15 s).
2	`automatic_failover` is on and licensed	Otherwise it only reports "primary down but automatic_failover is off" and waits for a manual `promote`.
3	A quorum is possible (3+ voters)	With two voters a silent peer could be a dead peer or a cut cable.
4	This node reaches a **majority** of voters	The isolated side must never promote.
5	**No** reachable voter reports that it still reaches the primary	If the witness still sees the primary, the problem is on our side. Only voters that run the daemon (nodes, witness) report this; the tiebreaker cannot.
6	This node is the **first alive candidate** in `nodes:` order	Prevents two replicas promoting at once. If the first candidate is dead, the next takes over.
7	Its data is recent enough: the newest replicated snapshot is not older than `max_replica_lag_seconds`, and there is one	Do not take over with stale data. `promote --force` overrides.

If all seven hold, the replica starts the promotion sequence:

1. **Pre-checks**: the VIP interface exists and has a link (otherwise it refuses *before* fencing anything, since fencing is irreversible).
2. `pre_promote` hook. If it fails, the promotion stops.
3. **Fence the old primary** with its `fence_cmd` / `fence_methods` (chapter 2). Skipped for a planned switchover. On failure: `on_fence_failure` decides.
4. **Write the new epoch** (epoch + 1) to disk. From this moment the cluster knows this node is the primary; heartbeats carry the new epoch to everyone.
5. Make the datasets writable and mounted; add the VIP and announce it (gratuitous ARP); start the `services` in order; start replication to the other nodes.
6. `post_promote` hook (a failure is logged only). The new state is then pushed to the other nodes in the background.

A failure before the epoch is written changes nothing. A failure while becoming primary (step 5) rolls the promotion back and **never reuses the epoch**. An automatic promotion that fails is retried after a pause of 10 s, doubling up to 1 minute. If the only problem is that the VIP cannot be assigned *after* the old primary was fenced, the node stays primary, shows `DEGRADED: virtual IP`, and keeps retrying the address.

4.4 After a failover: the old primary returns

- It starts, hears heartbeats with a higher epoch, adopts it and **becomes a replica**: services stopped, VIP dropped, datasets read-only.
- The new primary replicates to it from the last common snapshot. If the old primary wrote data after the last replicated snapshot, that data diverges: with `replication.auto_rollback: true` it is rolled back automatically (**those writes are discarded**); otherwise the node is shown as DIVERGED until you run `ovios-cluster resync <node> --discard-divergent-data`.
- Replication into a replica is guarded on the receiving side: streams from an older epoch, an unknown sender or for an excluded dataset are refused.

4.5 Planned changes: switchover and VIP handover

- **`ovios-cluster switchover`** (on the primary): loss-free. The chosen replica is checked first. The primary stops serving, sends a final sync, and the replica takes over at the next epoch. Nothing is fenced; if anything fails before the replica has taken over, the primary resumes.
- **VIP handover** (automatic, needs `automatic_failover`): if the VIP stays unhealthy for `vip_failure_seconds` (or the link is lost for about 15 s), the primary performs the same planned switchover to a healthy replica. It is rate-limited by `vip_handover_cooldown_seconds`.
- **`demote`** and **`promote`** are the manual tools for cases the cluster cannot decide itself.

4.6 Worked timeline: primary isolated from the network

Settings: heartbeat 2 s, lease 15 s, grace 7.5 s, watchdog timeout 10 s, a witness. At t = 0 the primary's network is cut. The last proof of quorum was at t = 0.

Time	Primary (isolated)	Replica (with the witness)
0 s	Heartbeats stop arriving.	Heartbeats from the primary stop arriving.
5 s	Watchdog feeding stops (`watchdog.starve`).	
7 s	Peers are no longer "alive" (7 s window); the tick sees no majority.	The primary is no longer "alive".
up to 15 s	Lease expires: **lease guard isolates** (VIP gone, services killed); the **watchdog resets** the machine (latest at 15 s).	Still waiting (needs 22.5 s).
22.5 s	(rebooting)	Conditions 1-7 hold. Fence attempt, then epoch + 1, datasets writable, VIP added, services started.
after 22.5 s		Clients reach the new primary once the datasets, the VIP and the services are up. The outage is the 22.5 s of silence plus the takeover time (fence, mounts, services starting).
later	Boots, learns the higher epoch, becomes a replica, catches up.	Replicates to it.

4.7 What each failure does

Failure	Result with default settings
Primary daemon crashes (kill -9, bug)	`failsafe` drops the VIP and kills the services immediately. systemd restarts the daemon after 2 s. With a watchdog, the machine resets after `timeout_seconds` unless the restarted daemon re-arms in time. A replica promotes after 22.5 s of silence only if the primary stays silent.
Primary daemon hangs (alive but stuck)	Without a watchdog the VIP stays up and the replica may promote after 22.5 s: **two primaries**. With a watchdog the machine resets. This is the main reason to enable the watchdog.
Primary machine loses power	Replica promotes after 22.5 s (plus fence time). The IPMI/plug method succeeds or finds it already off; an ssh method fails (then `on_fence_failure`).
Primary network cable out	Isolation at the lease, watchdog reset, replica promotes at 22.5 s (it needs the witness/tiebreaker to reach quorum).
Replica network cable out	The replica has 1 of 3 votes: it never promotes. The primary keeps serving; replication to it fails until it is back.
Heartbeat link between the two nodes only	Witness: no failover. Tiebreaker: failover with possible split brain until fenced (chapter 3).

Failure	Result with default settings
Witness or tiebreaker dies	No change, but no margin left.
Both data nodes die	Nothing is served until a node returns. A returning node keeps or takes the primary role only if it reaches a majority (so the third voter must be up). A returning replica promotes itself by the normal rules after 22.5 s of the primary's silence.
VIP interface fails on the primary	Planned handover after about 15 s (link) or `vip_failure_seconds` (other VIP problems).
Licence expires	Automatic promotion stops (replicas wait for a manual `promote`). A primary still steps down, isolates itself and fences as before.

5. Limits

5.1 Number of nodes

The engine has no hard-coded maximum number of nodes, and the licence does not limit it (the licence has one feature, automatic failover). The practical limits come from how the cluster works, and the engine has been exercised with up to three nodes in its automated tests; larger clusters are untested.

What grows with N nodes	Consequence
Heartbeat mesh	Every node exchanges heartbeats with every other node every `heartbeat_interval`: N x (N - 1) conversations. Fine for a handful of nodes; it grows quadratically.
Replication fan-out	There is **one primary at a time**. It sends every dataset to **every replica** itself. Each added replica multiplies the send bandwidth and CPU on the primary. Replicas do not chain.
Configuration pushes and ssh	Pushes and checks go to every node; the primary needs working root ssh key access to each.
Candidate order	Failover order is the order of `nodes:`. Only the first alive data node promotes; more nodes only add fallbacks.
Quorum	More voters tolerate more failures (chapter 6) but also need more of them reachable.

Recommended sizes

- **2 data nodes + 1 witness** (3 voters): the usual layout. Tolerates the loss of any one machine.
- **2 data nodes + a tiebreaker**: acceptable, with working fencing (chapter 3).
- **3 data nodes** (3 voters), or **3 data nodes + 2 witnesses** (5 voters) when you need to survive two simultaneous failures.
- Keep the total number of voters **odd** and small (3 or 5).

5.2 Other limits and boundaries

Limit	Detail
Storage	ZFS datasets only; shared-nothing, one writable copy. No shared storage and no active/active.
Replication is asynchronous	An unplanned failover can lose the writes since the last replicated snapshot: about `replication.interval` plus anything still in flight, or more if replication was lagging. A planned `switchover` loses nothing.
One VIP, one interface per node	A single floating address (`nodes[].interface` can differ per node). IPv4 for node addresses.
Timing floors	`lease_seconds` >= 3 x `heartbeat_interval`; `vip_failure_seconds` >= 30; `fencing.watchdog.timeout_seconds` >= 2 and <= `lease - 2 x heartbeat - 1`.
Failover time	At least `lease + grace` (22.5 s by default) plus the takeover time. Lowering the lease speeds it up and raises the risk of false failovers.
Automatic failover	Needs a licence, 3+ voters, 2+ data nodes and a way to fence (or a watchdog / `on_fence_failure: promote`).
Fencing coverage	Software layers cannot reset a frozen kernel; only outside fencing or a hardware watchdog does. A watchdog does not act when the primary still has quorum.
Watchdog hardware	The timer must really count down and reset the machine; verify every model.

Limit	Detail
Failsafe	A crash of the primary's daemon cuts its services even if systemd restarts the daemon two seconds later.
Heartbeat security	Mutual TLS with a per-cluster CA; node identity comes from the certificate. Ports must be reachable between all nodes.
Datasets	Any number of datasets and pools; system pools must be listed in `datasets.exclude`. `<pool>/.ovios-system` is always excluded.

6. Quorum, and why the number of voters should be odd

6.1 The problem quorum solves

A node that stops hearing its partner has two possible explanations: the partner is **dead**, or the network between them is **cut** and the partner is alive and still serving. From inside the node the two look identical. If it assumes "dead" and takes over while the partner is alive, both serve and the data diverges (split brain).

Quorum solves it with a simple rule: **a node may act only if it can reach a strict majority of all voters.** Two disjoint groups cannot both hold a majority, so at most one side of any partition can continue, and the other side stops.

6.2 A strict majority

With V voters the majority is `floor(V / 2) + 1`. The cluster survives the loss of `V - majority` voters.

Voters	Majority	Failures tolerated	Comment
1	1	0	No redundancy.
2	2	0	If either dies or is cut off, nobody has a majority. Cannot tell "dead" from "cut". No automatic failover.
3	2	1	The standard layout: 2 data nodes + witness, or 3 data nodes.
4	3	1	Same tolerance as 3, but one more machine, and a 2-2 split leaves **nobody** with a majority.
5	3	2	Survives two simultaneous failures.
6	4	2	Same tolerance as 5, plus the 3-3 deadlock.

6.3 Why odd

- **An even number adds cost and no tolerance.** Four voters tolerate one failure, exactly like three. Six tolerate two, like five.
- **An even number allows an exact tie.** If four voters split 2-2, neither half has a majority and the whole cluster stops. With an odd number one side always has more than half.
- **An odd number gives the cut-off minority a clear answer.** The side with fewer than half knows it is the minority and stands down; a primary there isolates itself after one lease.

6.4 How this engine counts

- Voters = every entry in `nodes:` (data nodes **and** witnesses) + the `tiebreaker` if set. In 2 data nodes + 1 witness, V = 3. In 2 data nodes + a tiebreaker, V = 3.
- A node counts itself, every peer that is alive (heartbeat within 7 s by default) and the tiebreaker if it answered ping.
- "Quorum possible" means V >= 3. With two voters the engine disables automatic failover (the validator refuses it) and never demotes or promotes by itself.
- The primary's lease is only as good as its proof of a majority; a replica's promotion needs a majority **and** "nobody still sees the primary". Chapter 4 has the exact rules.

6.5 Recommendation

Use **3 voters** for almost every OviOS cluster: two data nodes and a witness (preferred) or a tiebreaker. Use 5 only if you must survive two failures at once. Never add a fourth or sixth voter.

Appendix A. Command cheat sheet

Command	Purpose
`service ovios-cluster edit`	OviOS shell, on the primary: edit, validate, save and optionally push the configuration.
`ovios-cluster sync --check [--file F]`	Validate a configuration (YAML, unknown keys, rules, licence); pushes nothing.
`ovios-cluster sync`	Validate and push the configuration to every node.
`ovios-cluster status [--json]`	Cluster, votes, tiebreaker, peers, replication, VIP, `DEGRADED` notes.
`ovios-cluster init --vip IP [--tiebreaker IP]`	Create a cluster on this node.
`ovios-cluster add [--witness] [--fence-cmd C] IP`	Enrol a node. Needs root ssh key access. A fence command is required for a data node only while automatic failover is on and neither `on_fence_failure: promote` nor a real watchdog is set.
`ovios-cluster remove <node>`	Remove a node.
`ovios-cluster init --new-node-id`	Give a cloned machine its own identity (daemon stopped).
`ovios-cluster switchover [--host H] [--check] [-y] | --abort`	Planned, loss-free change of primary (run on the primary).
`ovios-cluster promote [--force]`	Make this replica the primary (manual failover). `--force` skips safety checks and continues if fencing fails.
`ovios-cluster demote [--force]`	Take this node out of the primary role by hand.
`ovios-cluster resync <node> --discard-divergent-data`	Discard a replica's divergent data and resend.
`ovios-cluster fence-test <node> [--execute] [--force]`	Check, or really run, a node's fence methods. `--execute` powers the node off.
`ovios-cluster license [install <file>]`	Show or install the licence (automatic failover is licensed).
`ovios-cluster failsafe`	Run by systemd (`ExecStopPost`) after an abnormal daemon exit; not meant to be run by hand.

Appendix B. Log events to look for

The daemon logs structured events to the journal: `journalctl -u ovios-cluster -f`.

Event	Meaning
`election.promote` / `promote.begin` / `promote.done`	A replica decided to promote; promotion started; finished (epoch shown).
`fence`	A fence was started for a node (number of methods). A method's outcome follows as an info or warning line.
`promote: FENCING <node> FAILED ...; promoting anyway`	`on_fence_failure: promote` or a real watchdog let a promotion go on after a failed fence.
`election.demote`, `demote.begin`	The primary is stepping down (lost quorum, or demoted by hand or switchover).

Event	Meaning
`isolate`	The lease guard cut the node off (age of the proof of quorum shown).
`failsafe`	The daemon died abnormally on the primary and the failsafe cut the node off.
`watchdog.arm`, `watchdog.disarm`	The watchdog was opened (the node became primary) or disarmed (orderly stand-down or clean daemon stop).
`watchdog.starve`, `watchdog.recovered`	Feeding stopped because quorum evidence became too old; feeding resumed. In dry-run nothing is reset.
`adopt`	A newer epoch from a peer was adopted.
`vip.handover`, `vip.handover.done`	An automatic VIP handover started / finished.
`suspend.cleared`, `resume`	A suspended node resumed or the suspension became irrelevant.

Appendix C. Timeline with the default numbers

Heartbeat 2 s, lease 15 s, watchdog timeout 10 s.

Quantity	Value and where it comes from
Peer considered alive	7 s without a heartbeat ends it (3 x 2 s + 1 s)
Daemon start before claiming primary	6 s (3 x heartbeat_interval)
Watchdog feeding stops	quorum evidence older than 5 s (lease - timeout)
Lease guard isolates	quorum evidence reaches 15 s (checked every 0.5 s)
Watchdog resets the machine	at the latest 15 s after the last proof of quorum
Orderly demotion by the tick	about 22 s after the last proof (lease + the 7 s alive tolerance)
Replica may promote	22.5 s after its last heartbeat from the primary (lease + grace)
Promotion retry after a failure	10 s, 20 s, 40 s, then every 60 s
Fence methods	fence_cmd 90 s; fence_methods 20 s unless set
VIP check	link every tick; ports and address every 30 s; handover after 15 s (link) or `vip_failure_seconds`
VIP handover cooldown	600 s (`vip_handover_cooldown_seconds`)
Full reconcile of the role	every 30 s

Appendix D. Example: two nodes, a tiebreaker, no IPMI

Two data nodes and the gateway as tiebreaker, an ssh fence method, a software watchdog and `promote` as the failure policy. Notes follow the example.

```
automatic_failover: true
tiebreaker: 10.0.0.1                # the gateway
nodes:
  - id: <node-a-id>
    ip: 10.0.0.107
    fence_methods:
      - name: ssh
```

```
        cmd: ssh -o BatchMode=yes -o ConnectTimeout=3 -o
UserKnownHostsFile=/etc/ovios-cluster/known_hosts root@$OVIOS_FENCE_IP "nohup sh -c 'sleep 1;
poweroff -f' >/dev/null 2>&1 &"
        timeout_seconds: 10
  - id: <node-b-id>
    ip: 10.0.0.134
    fence_methods: [ same as above ]
fencing:
  isolate_on_lease_loss: true
  failsafe_on_crash: true
  on_fence_failure: promote
  watchdog:
    enabled: true
    device: /dev/watchdog1           # softdog when a hardware driver owns /dev/watchdog
    timeout_seconds: 10
```

- On every data node: `softdog` loaded (and a hardware driver whose timer does not run blacklisted), the failsafe drop-in installed.
- Because the third voter is a tiebreaker, a heartbeat-only failure can still produce two primaries until the ssh method succeeds over another path; consider a witness instead.
- A frozen kernel is not covered. A hardware watchdog that counts down correctly, an IPMI or a smart plug would cover it.

Appendix E. What is verified and what is not

- **Automated tests (simulated hardware):** the election rules, the lease guard, fast isolation (command order), the failsafe gating, parallel fence methods and their timeouts, fence-test, the watchdog feeding rule and its states, the timer check, validation and the settings in this guide.
- **Verified on real hardware:** the parallel fence methods and `fence-test --execute` with an ssh method; `softdog` resets a machine; the Intel TCO timer on one Lenovo model did **not** count down (so it cannot be used there); and **the watchdog**: in the unplanned failover test of 6 November 2026 (three nodes on 1 Gb links, network cable pulled from the primary under load) the isolated primary was fenced by its watchdog in every test and the replica took over (see the OviOS HA Cluster Unplanned Failover Report).
- **Not yet verified on real hardware:** the failsafe under real systemd, smart-plug / PDU methods, and any cluster of more than three nodes. Test these on your own hardware before relying on them (section 2.8).
- Automatic failover is a licensed feature; check the licence date with `ovios-cluster license` before it lapses.