Designing a Small Highly Available Virtualization Cluster

A worked example

// prepared by Neil Hanlon

// entity Shrug PW Consulting, LLC

// date September 2026

About this example

This isn't a real engagement — I invented the company, the workload, and the hardware. What's real is the reasoning. When a client asks me to "make it highly available," most of the useful work happens before anyone buys anything, and this is roughly how that part goes.

"Highly available" is one of those phrases everyone nods along to until you ask what it actually means, and then the room goes quiet.

The scenario

Small shop, about forty VMs on one tired hypervisor: a couple of line-of-business apps, two databases, the internal wiki nobody admits to loving, and the usual pile of supporting services. One bad power supply, or one Tuesday-night patch reboot, and all of it goes dark at the same time. They'd like that to stop. What they don't want is to hire a platform team or forklift everything into someone's cloud to get there.

Three modest servers, two switches. The ask is "high availability." My first job is to pin down what that actually buys them, because the phrase hides at least four different promises with four different price tags.

What "highly available" actually has to mean

Before I spec a single NIC, I want the goal in plain terms. "Highly available" usually smuggles in some mix of these, and they don't cost the same:

- A node can die, or just get patched, without taking the running VMs with it.
- Storage survives losing a node — actually survives it, not "we're pretty sure there's a copy somewhere."

- The cluster keeps quorum. It can tell “my neighbor crashed” apart from “I can’t reach my neighbor,” and it won’t split-brain and corrupt itself trying to be helpful.
- Recovery is bounded and boring: I can tell you how long until things come back, and what (if anything) you lose.

For this shop the honest scope is the first three, plus fast automatic restart for the fourth. True nobody-notices-a-node-died failover is a different and pricier promise, and nine times out of ten the place that conversation really belongs is the databases, not the hypervisor. If the database can’t come back cleanly from a hard restart, no amount of cluster cleverness upstream saves you.

Where it actually breaks

Three nodes, not two, and this is the hill I’ll die on: three is the smallest number that can hold a vote. Lose one and the survivors still have a majority, so they can make decisions instead of bickering. Two nodes plus a cheap external witness can also keep quorum, and I’ll price that if the budget’s tight — but three keeps the whole thing uniform and leaves room to actually run the workload with a host down.

Which is the part everyone skips. If all forty VMs have to keep running with a host missing, the cluster has to fit them on two hosts, not three. Size to N-1 on day one, or you’ve built something that looks like HA right up until the first time you lean on it, at which point it falls over very politely.

Storage is where the bodies are buried

Compute failover is the easy half; honestly it mostly configures itself. Storage is where the risk and the money actually live, so I pick it on purpose instead of taking whatever’s default.

Roughly three ways to go:

- Replicate across all three nodes, with the host as the failure boundary. Every volume keeps copies on different machines, so losing a box is a bad afternoon instead of a data-loss incident. It’s the most flexible option and the one that leans hardest on your network.
- Two-node replication with a witness. Cheaper, simpler, genuinely fine at this size. You pay for it later in clumsier placement and a worse growth story.
- One shared storage box that everything hangs off. Feels tidy, and it quietly drops a single point of failure right back into the middle of the project you started in order to remove one. Unless that box is itself redundant, which it won’t be, not on this budget.

The default I’ll defend is replication across the three nodes with the failure domain pinned to the host, so no single machine is ever holding the only copy of anything. The catch I say out loud every time: the moment storage replicates between nodes, your network is part of your storage. Treat it like plumbing and it behaves like plumbing.

The network just inherited a second job

Once storage is replicating, a rebuild after a node comes back and the traffic your users actually make are fighting over the same cables — unless the design keeps them apart. Two things I won't let share a lane: a storage rebuild can't be allowed to starve the apps, and the cluster heartbeat can't be starved by either of them. A cluster that loses its heartbeat under load starts making genuinely bad decisions on your behalf.

So, three planes instead of one: app traffic, storage replication, and cluster heartbeat. Split them logically with VLANs, and physically for storage if the budget stretches that far. The tempting shortcut is a single flat network. It's flawless in the demo and detonates during the first real rebuild.

The backup nobody wants to talk about

Replication protects you from a node dying. It does nothing for a fat-fingered delete, a bad deploy, or ransomware — those replicate to every copy, instantly and faithfully. So backups have to live outside the cluster entirely: a separate system, ideally offsite, that doesn't share the storage, the admin login, or the single bad day with production. A "backup" sitting on the same three nodes it's meant to protect isn't a backup, it's a rehearsal for disappointment.

Prove it before it carries anything real

The design isn't finished when it boots. It's finished when I've made it fail on purpose and liked what I saw:

- Baseline storage and network under normal load and under a rebuild, so when someone says "it feels slow" in March there's a real number to argue with.
- Pull power on a node and time what happens: how fast the VMs restart, what quorum does, how the storage reports itself degraded.
- Put the node back and watch the rebuild — copies land back on separate hosts, recovery traffic stays off the app lane.
- Actually restore a VM from the offsite backup. An untested backup is a wish, not a control.

The power-pull is the acceptance test for the whole thing. Not "look, a VM came back" — the entire architecture, behaving the way I told you it would.

What I'd nail down for real

On a live job the invented numbers turn back into questions: how many VMs, really, and how fast is that number growing; what the databases actually need to survive a hard restart (usually the thing that drives the whole design); how many network ports per host I've got to work with; where backups go and how long they have to stay; anything with a compliance or downtime clause hanging off it. The whole point of a design phase is to find the gap between the availability you asked for and the hardware you've got — on paper, cheaply, before production finds it for you.