Planning & Capacity

How the cluster was sized: CPU/RAM/disk math, memory overcommit on a swapless host, ZFS ARC capping, and the live VM right-sizing that paid for it.

Why size things at all?

It's a home lab. Why not just give every VM "plenty" of everything and move on?

Because the constraint is real, and pretending it isn't is how you wake up to an out-of-memory kill at 3am. One host has a fixed amount of RAM, CPU, and disk. Every VM you start writes a cheque against it. The interesting engineering — the part worth showing — is deciding *where* the budget goes, and noticing *which* resource you'll run out of first.

Spoiler for this lab: it's always memory. CPU overcommits gracefully (cores are time-shared), disk is plentiful, but RAM is hard — when it's gone, something dies.

> **Why we use this:** capacity planning is the least glamorous and most useful infrastructure skill. "Can we fit this?" is a question you should be able to answer with arithmetic before you provision, not discover afterwards. Every VM in this lab was sized by doing that arithmetic out loud.

The host, and the numbers

The whole lab runs on a single Proxmox host with, roughly:

- **64 CPU threads**
- **~125 GiB usable RAM**
- **A few hundred GB of fast local disk for the OS, plus a multi-TB ZFS pool for VM disks**

CPU stays comfortable — 64 threads is far more than the VMs will ever simultaneously peg — and the ZFS pool started with terabytes to spare. (Disk has since grown tight as the database and storage VMs claimed their space, so the pool is now a number worth watching too — more on that in the lessons — but it was never the *binding* constraint.) So the planning is almost entirely a RAM exercise.

There's one important wrinkle that shapes everything: **there is no swap on the host.** That's a deliberate choice for a virtualization host — you don't want the hypervisor swapping VM memory to disk, because the performance cliff is brutal and unpredictable. But it also means there's no safety cushion: if committed memory exceeds physical memory and the VMs actually use it, the kernel's OOM killer starts shooting processes. So the rule becomes simple and strict:

> **Committed VM memory + host overhead must stay under physical RAM. There is no margin to borrow from.**

The rest of this book is how that rule played out as the platform grew.

The thing that quietly eats your RAM: ZFS ARC

There's a second tenant on the host besides the VMs: **ZFS**. ZFS uses a chunk of RAM as a read cache called the **ARC (Adaptive Replacement Cache)**, and by default it will happily grow to use up to half (or more) of the machine's memory.

That's fine on a dedicated storage box. On a host that's also packing in VMs, an uncapped ARC is a hidden roommate eating the budget you wanted for guests. When this lab started filling up, the ARC was effectively uncapped.

The fix is to give ARC a fixed ceiling so the VM budget is predictable:

```
# /etc/modprobe.d/zfs.conf
options zfs zfs_arc_max=<bytes>
```

I capped it (first to 16 GiB, later trimmed to 12 GiB when more VMs needed the room) and applied it live as well as persisting it. ARC is *reclaimable* — it'll shrink under pressure — but on a swapless host you don't want to rely on "it'll shrink in time." A hard cap turns a fuzzy, dynamic consumer into a known constant you can plan around.

> **Lesson learned:** before you size VMs on a ZFS host, find out what ARC is allowed to take. An uncapped ARC is the single most common reason "the numbers added up but the host is still tight." Cap it, then do your VM math against what's left.

Making room: right-sizing live VMs

The cluster needed real memory: a control-plane node, three workers at 24 GiB each, a build runner, and later three database VMs. Adding all of that naively would have blown past physical RAM.

So before adding, I went looking for slack in what already existed. The original "core" service VMs were each handed 4 GiB out of habit. A quick look at what they actually used told a different story:

```
VM                used (real)     allocated
Git / Docs / ...   ~0.5-0.7 GiB     4 GiB     <- mostly page cache, not need
```

Those services were sitting on ~0.5 GiB of genuine usage with 4 GiB allocated. The "used" number in the hypervisor looked high only because Linux fills spare RAM with disk cache — which is *not* memory the VM needs. Reading the real figure (free/available, not used-including-cache) showed each could drop to 2 GiB with room to spare. That freed ~12 GiB.

Later, the three Kubernetes workers were trimmed from 24 to 22 GiB each — 6 GiB — to fund the first two database servers, and then trimmed again from 22 to 20 GiB to fund a third (a dedicated MySQL box). At 16 vCPU / 20 GiB that's 1.25 GiB per core, which is about as lean as I'd take these workers. Because these VMs have ballooning disabled, "trimming" means a real stop/start, so each round was done as a rolling operation (drain a node, resize it, bring it back, move to the next) to keep the cluster healthy throughout.

> **Lesson learned:** "used memory" lies. On Linux, free RAM is wasted RAM, so the OS caches aggressively and the *used* figure looks scary. Size against **available** memory and actual working set, not the headline number. Half this lab's growth was paid for by reclaiming allocation that was never really in use.

Lessons from sizing this lab

The short version, for anyone doing the same on their own host:

- **Find your binding constraint first.** Here it's RAM, every time. CPU overcommits fine; disk is cheap. Plan against the thing that actually runs out.
- **No swap means no second chances.** On a hypervisor that's the right setting, but it turns "a bit over budget" into "the OOM killer picks a victim." Keep committed memory honestly under physical.
- **Cap ZFS ARC.** Otherwise it's an invisible VM eating your budget.
- **Right-size from real usage, not allocation.** Most VMs are over-provisioned out of habit. Reclaiming that is free capacity.
- **Resizing isn't always live.** With memory ballooning off, changing a VM's RAM is a stop/start — plan a rolling, drain-aware procedure for anything that's part of a cluster.
- **Watch the second-tightest resource, too.** RAM was always the binding constraint — but quietly, thick-provisioned 250 GiB database disks pushed the ZFS pool past 90%. The resource that bites you is usually the one you stopped watching the moment you'd "solved" capacity.
- **Leave a margin.** I aimed to keep a few GiB of headroom on the host at all times. The last few percent of RAM is not worth the risk of an OOM cascade.

None of this is exotic. It's just arithmetic done before provisioning instead of after — which is the entire discipline.