Scaling Witchly.host to 10,000 Users on Bare Metal

A Systems Engineering Case Study

Project	Witchly.host — Free Game Server Hosting Platform
Role	Founder & Lead Systems Engineer
Founded	August 2021; relaunched with rebuilt stack in 2026
Status	Bootstrapped | Profitable | 10,000+ Registered Users

10,000+	€100/mo	50 Gbps+	$3,000+/mo
Registered Users	Infrastructure Cost	DDoS Mitigated	Equivalent AWS Cost

1. The Problem

Game server hosting sits in an awkward gap. Free platforms like Aternos and Minehut are severely resource-constrained and unreliable under load. Commercial hosts charge $15–30 per server per month, pricing out hobbyists and small communities entirely. Nobody was offering production-grade, low-latency infrastructure at zero cost.

Witchly was built to close that gap — and to do it on zero external funding. Every architectural decision was made to maximise raw compute per euro, with the goal of making a free tier economically viable long-term.

2. The Stack

Hardware — Data Plane

The compute layer runs on Hetzner bare-metal servers. Bare metal was a deliberate choice over managed cloud — for a workload with predictable density (game containers), the cost delta is too large to ignore.

Component	Specification
Provider	Hetzner Bare Metal
CPU	Intel Core i9-9900K
Memory	128 GB RAM
Storage	2 TB NVMe SSD
OS	Ubuntu 20.04 LTS
Locations	Germany, Finland

Orchestration

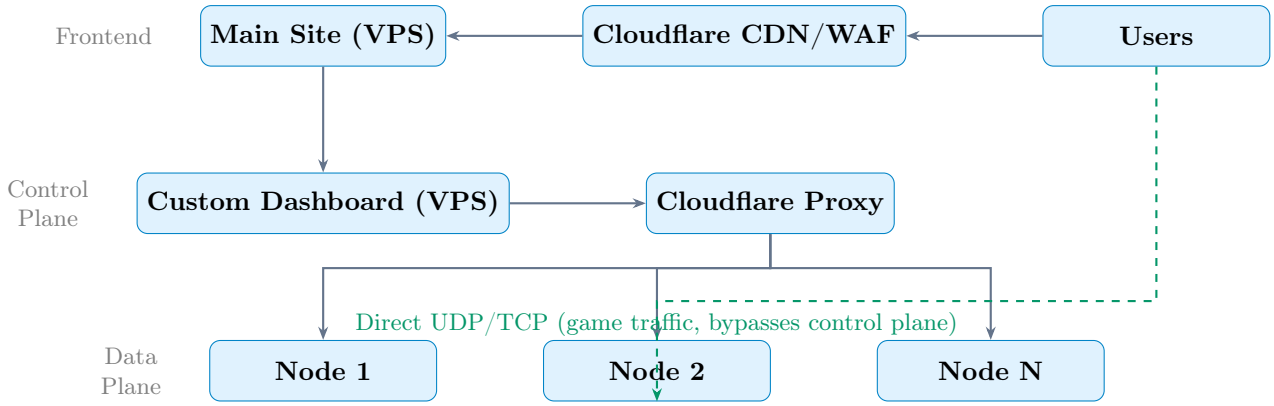
Game servers run as isolated Docker containers managed through Pterodactyl. Resource isolation is enforced at the kernel level:

- **CPU pinning** via cgroup cpusets — each tenant is allocated dedicated physical cores.
- **RAM hard limits** enforced per container to prevent OOM cascading across tenants.
- **Custom game environment configs** for niche engines and modpacks requested by the commu-

nity.

Architecture

The system is split into three fully separated tiers, each independently scalable:



Game traffic connects directly to bare-metal nodes over raw UDP/TCP, completely bypassing the control plane. This architectural separation proved critical during the DDoS incident described in Section 3 — active game servers stayed alive even while the control plane was under attack.

Custom Dashboard

Rather than relying on an off-the-shelf panel, a custom dashboard was built from scratch to handle the full user lifecycle:

- **Server provisioning** — users can create, configure, and manage game servers directly from the dashboard.
- **Stripe integration** — subscription billing and plan management handled natively, without any third-party billing portal.
- **Resource visibility** — per-server CPU, RAM, and storage usage surfaced in real time.
- **REST API integration** — dashboard communicates with the game server backend via internal APIs for provisioning and lifecycle management.

Building the dashboard in-house rather than wrapping an existing panel gave full control over UX, feature velocity, and the billing flow — all of which are differentiators in a space where competing free hosts offer clunky, generic interfaces.

3. Engineering Spotlight: DDoS Mitigation

The Incident

During peak traffic, the infrastructure absorbed a coordinated **50 Gbps+ L7 DDoS attack**. The attack targeted the control plane's authentication layer with volumetric request floods, overwhelming the server and causing repeated crashes every few minutes. Dashboard access was effectively down for all users.

Critically, active game servers continued running without interruption. The direct UDP/TCP routing for game traffic meant the attack on the control plane had zero impact on players already in-session — a direct payoff of the plane-separation architecture.

The Response

Full service was restored in under 20 minutes with zero data loss through a layered mitigation approach:

1. **Attack analysis.** Log correlation identified the attack vector and the botnet's origin profile quickly, enabling targeted responses rather than blanket blocking.

2. **Edge-level mitigation.** Cloudflare's WAF was configured to intercept and challenge suspicious traffic at the network edge before it reached any origin server.
3. **Network-level filtering.** Malicious traffic sources were blocked at the autonomous system level at the CDN edge, cutting off the bulk of attack volume.
4. **Host-level hardening.** Firewall rules on bare-metal nodes were tightened to drop all non-whitelisted inbound traffic, protecting game ports running outside the proxy layer.

The multi-layer approach — edge, network, and host — was essential. Any single layer alone would have been insufficient against an attack at this scale.

4. Cost Efficiency

Component	Witchly	AWS EC2 Equivalent
Full compute cluster	€100/mo	$3,000+/mo
DDoS protection	$0 (Cloudflare free tier)	$3,000+/mo (Shield Advanced)

Running the entire cluster at €100/month is what makes a permanently free tier viable. Equivalent compute on AWS EC2 runs $3,000+ per month — a gap large enough that Witchly's free-tier model would be impossible on managed cloud without significant outside funding.

5. 2026 Relaunch & Current State

Witchly originally ran from 2021 to 2024, validating the infrastructure approach and accumulating 10,000+ registered users. The 2026 relaunch rebuilt the stack from the ground up — new custom dashboard, improved provisioning flow, and a cleaner frontend.

Monetisation: Programmatic advertising via NitroPay across the dashboard and management interface. The core revenue challenge is geographic ad fill rate: the majority of Witchly's user base comes from non-US/EU regions where CPMs are substantially lower. Current growth focus is on English-language gaming communities (via Reddit, SEO, and Minecraft creator outreach) to shift the traffic mix toward higher-value markets.

6. Key Takeaways

1. **Bare metal beats cloud for predictable compute.** Hetzner bare-metal at €100/mo versus AWS EC2 at $3,000+/mo is the delta that makes the business model work. For compute-heavy workloads with predictable density, managed cloud is simply the wrong tool.
2. **Separate your planes.** Routing game UDP/TCP traffic directly to bare-metal nodes and keeping it entirely off the control plane was the single best architectural decision made. A 50 Gbps DDoS attack on the dashboard had zero effect on active game sessions.
3. **Build your own tooling.** Shipping a custom dashboard with native Stripe billing — rather than wrapping an existing panel — gave full control over UX, monetisation, and feature velocity. At a platform level, your interface is your product.
4. **Layer your defences.** The DDoS response worked because edge, network-level, and host-level mitigations engaged together. At 50 Gbps volumetric scale, single-layer protection is not enough.