Zyvor Fabric Product Overview

Zyvor Fabric — Enterprise VM Management Platform

The Problem

Organizations running Linux infrastructure need a unified way to manage virtual machines. Existing solutions are either:

- **Too heavy** — traditional virtualization stacks require complex multi-server deployments, dedicated storage infrastructure, and specialized operations teams
- **Too basic** — Manual QEMU/KVM management with shell scripts doesn't scale and lacks security, monitoring, or multi-user access
- **Too locked-in** — Cloud-only solutions (AWS, Azure, GCP) create vendor dependency with unpredictable costs

Zyvor Fabric fills the gap — a single-binary VM management platform that runs on any Linux server with systemd, providing enterprise features without enterprise complexity.

What Is Zyvor Fabric?

Zyvor Fabric is a production-grade virtual machine management platform built in Rust. It drives VMs through FluxVM with a complete management layer:

- **One binary, one config file, one systemd service** — deploys in under 5 minutes
- **520+ REST API endpoints** with JWT authentication, RBAC, and audit logging
- **4 management interfaces** — CLI, web dashboard, Kubernetes operator, Terraform provider
- **Enterprise features** — HA clustering, live migration, GPU passthrough, backup/restore, network policies

```
sudo systemctl enable --now zyvor-fabricd
# That's it. Platform is running.
```

Key Differentiators

1. Built on FluxVM (Not a Custom Hypervisor Stack)

Zyvor Fabric launches guests through FluxVM — a disposable-VM engine with no systemd dependency of its own. This means:

- No custom kernel modules or hypervisor patches
- VMs are first-class systemd units with journal logging, socket activation, and watchdog support
- Works on Fedora, Ubuntu, Debian, RHEL, SUSE — any distro with systemd 256+
- Upstream-maintained VM lifecycle, not a fork

2. Single Binary, Zero Dependencies

One 15MB binary, one config file, a SQLite user store, and about 5 minutes to set up. It runs on any Linux.

3. Security-First Architecture

The entire codebase has undergone a **30-round security audit** with 174 issues identified and fixed (9 consecutive clean rounds — audit complete):

- **Zero unsafe Rust** — memory-safe by construction
- **Zero shell pipelines** — all subprocess calls use safe argument passing
- **JWT + RBAC** on every API endpoint (Admin/User/Viewer roles)
- **bcrypt password hashing** with auto-generated secrets (0600 file permissions)
- **Rate limiting** on authentication (5 attempts/5 min)
- **Input validation** on every user-facing parameter
- **Audit logging** on all VM lifecycle operations
- **Path traversal protection** with canonicalization
- **SQL injection prevention** — all queries parameterized

4. Rust Performance

- **Sub-millisecond API response times** — async Axum + Tokio runtime
- **Low memory footprint** — ~20MB RSS for the daemon
- **No garbage collection pauses** — predictable latency
- **Safe concurrency** — per-VM mutexes prevent race conditions

---

Feature Overview

VM Lifecycle

- Create, start, stop, restart, pause, resume, delete, **per-VM backup**
- Full and linked cloning with CoW support
- **Hibernate (suspend-to-disk)** and resume from snapshot
- Templates for rapid deployment

- Declarative config via YAML (`fabricctl apply -f config.yaml`)
- Multiple disk formats: qcow2, raw, vmdk, vdi
- **VM import** from VMDK, VDI, VHD (auto-convert to qcow2)
- **Online disk resize** (qemu-img + QMP block_resize)

Storage

- **6 backends**: Local, NFS, LVM, LVM-thin, ZFS, Ceph/RBD
- Volume CRUD with attach/detach, resize, clone
- Snapshot create/restore with retention policies
- ZFS replication with incremental send/receive
- Ceph cluster health monitoring and RBD image management
- **Storage live migration** — move VM disks between pools without downtime
- **Cloud image downloader** — built-in catalog (Ubuntu, Fedora, Debian, Alma)
- **ISO management** — download, list, delete ISO images

Networking

- **Network Policies** — Cilium-style label-based ingress/egress rules
- **VM Firewall** — Per-VM firewall profiles and zones via nftables
- **Service Mesh** — Virtual IP load balancing (round-robin, least-conn, random, IP-hash)
- **QoS / Traffic Shaping** — Guaranteed/max rate, burst, priority-based bandwidth
- **DNS Policy** — Zone management, upstream servers, domain blocking
- **VPN Mesh** — WireGuard tunnels (point-to-point, hub-spoke, full-mesh)
- **Packet Mirror** — Traffic capture for debugging
- **NAT Gateway** — Masquerade, SNAT, DNAT, hairpin NAT
- **Network Monitor** — Per-VM bandwidth tracking with threshold alerts

Security & Identity

- JWT authentication with configurable token expiration
- **LDAP and OIDC/OAuth2** integration for enterprise SSO
- 3-tier RBAC (Admin / User / Viewer) enforced on every endpoint
- **Multi-tenancy** — project isolation with member roles and quotas
- API keys for service-to-service authentication
- TLS/HTTPS with certificate management
- Audit logging with JSON/CSV export
- Encryption at rest
- Rate limiting on authentication and API keys
- **30-round security audit** — 174 issues fixed, 9 consecutive clean rounds, audit complete
- **Storage pool name validation** — LVM, ZFS, Ceph pool names validated
- **SSRF prevention** on all user-provided URLs
- **Entity ID sanitization** in state store
- **Credential redaction** in API responses
- **WebSocket authentication** on console and VNC endpoints

High Availability

- Predictive DRS (Distributed Resource Scheduling)

- **Affinity / anti-affinity rules** for VM placement constraints
- Fault tolerance with automatic failover and fencing
- Distributed storage replication
- Site recovery with failover/reprotect workflows
- **Resource overcommit policies** (CPU/memory/storage ratios)

Monitoring & Automation

- Prometheus metrics exporter (`/metrics` endpoint)
- **Metrics retention policies** with configurable cleanup
- Analytics dashboard with historical data
- Multi-channel notifications: Email, Slack, **Webhook with retry + backoff**, Microsoft Teams
- VM scheduling (once, daily, weekly)
- Backup/restore with retention policies and incremental backups
- **Per-VM backup** from web UI (single or bulk)
- **Automated daily backups** via an internal scheduler inside the daemon (2:00 AM), not a separate systemd timer
- **Automated weekly state store cleanup** via the same internal scheduler (Sunday 3:00 AM)
- **Backup configuration** via the `[backup]` section in `zyvor-fabricd.toml`
- **Post-install auto-verify** — smoke test runs automatically after deploy/reinstall (API, auth, VM CRUD, backups)
- **Deep health check** — API, disk space, DB integrity, credentials, timers, memory, KVM
- **TLS certificate generation** — self-signed certs with SAN via `fabricctl tls`
- **Shell completions** — Bash tab completion for `fabricctl` and `fabricctl`
- Resource quotas, pools, and datacenter abstractions
- **Database schema migrations** with version tracking

Console Access

- WebSocket terminal via xterm.js (browser-based SSH)
- VNC graphical console via noVNC proxy
- Authenticated with same JWT tokens as API

Cloud & Virtualization

- cloud-init integration (NoCloud datasource)
- TPM/vTPM support (1.2 and 2.0 via swtpm)
- GPU passthrough (NVIDIA, AMD, Intel GVT-g)
- **Live migration** with iterative rsync pre-copy and cutover
- CPU pinning and NUMA optimization
- **IPv6 support** — dual-stack nftables (ip + ip6)
- **API versioning** — all endpoints under `/api/` and `/api/v1/`

Management Interfaces

CLI (`fabricctl`)

Scriptable command-line tool with JSON/YAML/table output:

```
fabricctl list -o json
fabricctl create myvm --image=ubuntu.qcow2 --cpus=4 --memory=4G
fabricctl start myvm
fabricctl apply -f infrastructure.yaml
fabricctl policy list
fabricctl ceph health my-pool
```

Terminal UI (removed)

The former `fabricctl` has been removed. Use the web console under `/app` or `fabricctl`.

Per-VM actions: `s` start, `t` stop, `r` restart, `b` backup, `d` delete.

Web Dashboard

React-based UI with 37+ pages, command palette (Ctrl+K), light Apple console, real-time WebSocket updates, and bulk operations. Per-VM actions: start, stop, pause, backup, console, details. Bulk actions: start, stop, backup, delete.

Kubernetes Operator

Manage VMs as `VirtualMachine` CRDs with automatic reconciliation:

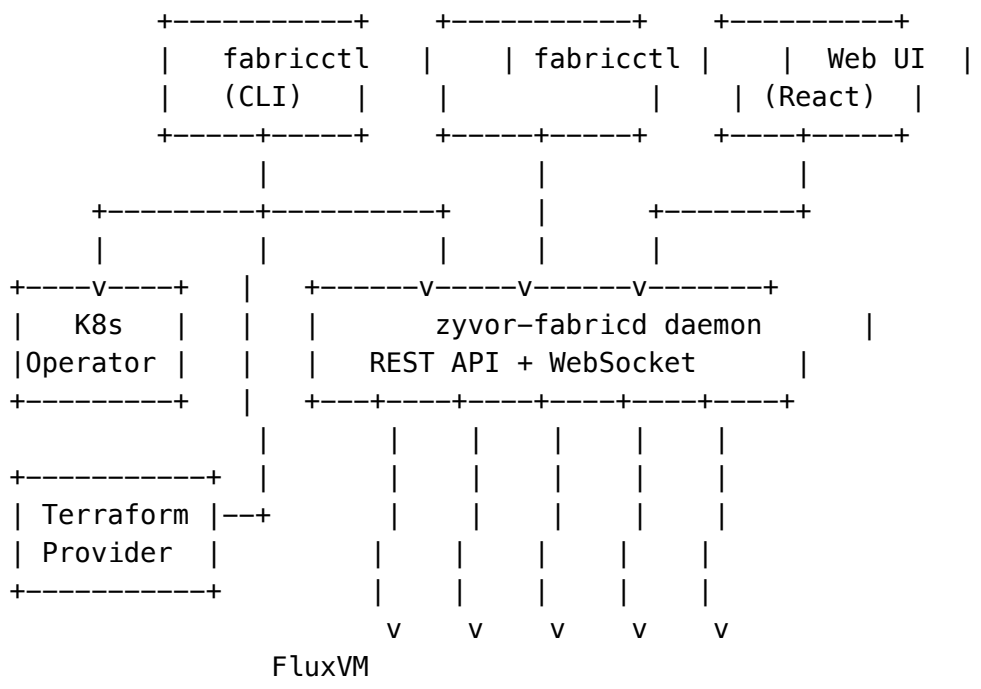
```yaml
apiVersion: Zyvor Fabric.io/v1
kind: VirtualMachine
metadata:
  name: web-server
spec:
  image: ubuntu-22.04.qcow2
  cpus: 4
  memory: 4096
```

Terraform Provider

Declarative VM provisioning with full plan/apply workflow:

```
resource "zyvor_fabric_vm" "web" {
  name   = "web-server"
  image  = "ubuntu-22.04.qcow2"
  cpus   = 4
  memory = 4096
}
```

Architecture



Deployment Models

Single Server

One Linux server running Zyvor Fabric with local storage. Suitable for development, testing, small teams, and edge deployments.

Requirements: Linux with systemd 256+, 4GB RAM minimum

Multi-Node Cluster

Multiple Zyvor Fabric nodes with etcd-based clustering, shared storage (NFS/Ceph), live migration, and HA failover.

Requirements: 3+ nodes, shared storage, etcd cluster

Kubernetes-Managed

Zyvor Fabric nodes managed by the Kubernetes operator. VMs defined as CRDs alongside containerized workloads.

Requirements: Kubernetes cluster with zyvor-fabricd operator deployed

Technology Stack

Layer	Technology
Language	Rust (2021 edition)
Async Runtime	Tokio 1.44
Web Framework	Axum 0.8
Web console	

Getting Started

One-Command Deployment

```
# Deploy everything (auto-sudo — no manual sudo needed)
./fabricctl deploy

# That's it. Zyvor Fabric is running.
```

Step-by-Step

```
# Install dependencies (auto-sudo)
./fabricctl deps

# Build
./fabricctl build

# Run tests
./fabricctl test

# Install and start (auto-sudo)
./fabricctl install
./fabricctl start

# Read admin password
./fabricctl password

# Run interactive demo
./fabricctl demo

# Create your first VM
fabricctl create myvm --image=/path/to/image.qcow2 --
        cpus=4 --memory=4096
fabricctl start myvm

# Open web dashboard
open http://localhost:9095
```

Layer	Technology

Management Commands

```
./fabricctl status      # Check service status
./fabricctl verify      # Post-install smoke test (API,
        auth, VM CRUD, backups)
./fabricctl health      # Deep health check (disk, DB,
        timers, resources)
./fabricctl logs        # Follow logs
./fabricctl restart     # Restart service (auto-sudo)
./fabricctl reinstall   # Rebuild + reinstall + auto-
        verify (auto-sudo)
./fabricctl upgrade     # Git pull + reinstall (auto-
        sudo)
./fabricctl uninstall   # Remove everything (auto-sudo)
./fabricctl doctor      # System readiness check
./fabricctl tls         # Generate self-signed TLS
        certificate
```

Backup Commands

```
./fabricctl backup now      # Run backup immediately
./fabricctl backup enable   # Enable daily backup timer
        (2:00 AM)
./fabricctl backup disable  # Disable backup timer
./fabricctl backup status   # Show timer state +
        storage info
./fabricctl backup logs     # Follow backup logs
```

For technical details, see the Architecture Guide, API Reference, and Security Documentation.