ZYVORAI LABS · USER DOCUMENTATION

Zyvor Fabric

A private cloud control plane with no systemd dependency.

2026-09-02

zyvor.dev · Zyvor Production License v1.0

Zyvor Fabric is a complete Rust control plane giving you VM-platform and KubeVirt-style capabilities without a heavyweight stack. Manage the same infrastructure four ways — CLI, web dashboard, Kubernetes operator, or Terraform — over a single daemon exposing 480+ REST endpoints and live WebSocket channels. VM lifecycle runs through FluxVM, a disposable-VM engine with no systemd dependency of its own (`driver.fluxvm_url` in `zyvor-fabricd.toml`). There are no custom hypervisor patches or kernel modules to maintain, and the daemon itself runs fine under systemd or any other supervisor — it's no longer a hard requirement.

480+ REST API endpoints · **4** management interfaces · **40+** Rust backend crates · **6** storage backends · **37+** web dashboard pages · **1** binary, one config, one service

This is the user-facing onboarding guide — how to access the product, your first workflows, and how to use every feature. A print-ready PDF of the same content sits alongside this file.

Contents

0. Getting started — access & first workflows
1. VM Lifecycle & Provisioning
2. Storage Orchestration
3. Software-Defined Networking
4. Security & Identity
5. High Availability & Disaster Recovery
6. Compute, GPU & Virtualization
7. Monitoring, Automation & Operations
8. Interfaces & Automation Surfaces
9. Fleet, Cost & Governance

Getting started

How to access it

- **Web:** React dashboard served by the `zyvor-fabricd` daemon at http://localhost:9095 — light Apple-style console with marketing site. It shares the daemon's origin (no separate web server); generate the TLS cert with `./zyvor-fabricd-ctl tls`.
- **CLI:** `fabricctl` — scriptable client with table/JSON/YAML output (`--output json`). Examples: `fabricctl list`, `fabricctl create web-01 --image fedora-41 --cpus 2 --memory 4096`, `fabricctl start web-01`, `fabricctl apply -f config.yaml`.
- **API:** REST + WebSocket exposed by the `zyvor-fabricd` daemon (480+ endpoints under `/api/...`). Obtain a token with `POST /api/auth/login`, then pass `Authorization: Bearer` on every call. The same API also backs the Terraform provider, the Kubernetes `VirtualMachine` CRD operator, and the Rust/Python/Ansible SDKs.
- **Login:** Username `admin`; the initial password is auto-generated on first run — read it with `./zyvor-fabricd-ctl password` (or `sudo cat /var/lib/zyvor-fabricd/.admin_password`). JWT tokens last 24h by default (`auth.token_expiration_hours`); 3-tier RBAC (admin/user/viewer) is enforced on every endpoint, with optional TOTP 2FA.
- **Needs:** A Linux host with systemd 256+ and KVM; install and bring the daemon up with `sudo systemctl enable --now zyvor-fabric`.

Your first workflows

- **Launch your first VM in five minutes**
  1. Start the daemon: `sudo systemctl enable --now zyvor-fabric`, then read the admin password with `./zyvor-fabricd-ctl password`.
  2. Pull a cloud image from the built-in catalog (`POST /api/images/cloud/download` with `{"name":"fedora-41"}`), or list options first with `GET /api/images/cloud`.
  3. Create the VM: `fabricctl create web-01 --image fedora-41 --cpus 2 --memory 4096`.
  4. Start it: `fabricctl start web-01`, then confirm `state: running` with `fabricctl list`.
  5. Open a console from the web dashboard (http://localhost:9095).
- **Manage VMs declaratively (GitOps)**
  1. Describe one or more VMs (name, cpus, memory, disk, image, tags/labels) in a YAML file.
  2. Reconcile them into the fabric: `fabricctl apply -f config.yaml`.
  3. Commit `config.yaml` to git as the source of truth; re-run `fabricctl apply -f config.yaml` to converge after edits.
  4. For K8s shops, express the same VMs as `VirtualMachine` CRDs and let the Helm-installed operator reconcile them continuously.
- **Wire up VM networking**
  1. Create a bridge (virtual switch): `POST /api/networkd/bridges` with a name, address, and MTU — every change writes `.netdev` / `.network` files and reloads networkd.
  2. Segment traffic with a VLAN: `POST /api/networkd/vlans` (`{name, id, parent}`), or bond NICs via `POST /api/networkd/bonds`.
  3. Hand out addresses: `POST /api/networkd/dhcp` with the bridge name and a pool range.
  4. Expose a guest service with a port-forward: `POST /api/networkd/port-forwards`.

5. Do the same from the Web Network Security page (9 Cilium-style tabs).

- **Make a VM reachable from your laptop, the simple way**
  1. In the Create VM wizard's Advanced Options, either expose a port under NAT (one click for SSH via the Expose SSH (22) preset, or any custom host→guest mapping — `POST /api/vms/:name` with `port_forwards`), or switch to Bridged mode for a VM with its own address on the LAN.
  2. Under Bridged, leave it on DHCP for an auto-assigned address, or check "Assign the IP statically via cloud-init" for a fixed address baked in at boot.
  3. Forwarded ports bind `0.0.0.0`, so they're reachable from any machine on the network — not just the host.
  4. Add or remove forwards later without recreating the VM: `POST` / `DELETE /api/vms/:name/port-forwards` · Web VM detail Network tab.

- **Protect a VM with snapshots and backups**
  1. Before a risky change, take a snapshot: `POST /api/vms/web-01/snapshots` (use `snapshot_type: Full` to include memory state).
  2. Roll back if needed: `POST /api/vms/web-01/snapshots//revert`.
  3. Create a durable backup: `POST /api/backups` (`{vm_name, backup_type: full|incremental, retention_days}`).
  4. Automate it with a policy: `POST /api/backups/policies` (daily/weekly, matched by VM tag) — scheduled via systemd timers.
  5. Restore any time with `POST /api/backups/restore`, or drive all of this from the Web Backups page.

- **Turn a golden image into a fleet**
  1. Build or download a base image, create and configure one VM, then customize first boot via `POST /api/vms/base/cloud-init` (users, packages, SSH keys).
  2. Capture it as a reusable template: `POST /api/templates` (`{name, source_vm}`).
  3. Stamp out instances: `POST /api/templates//deploy` per VM, or `POST /api/vms/base/clone` for full/linked copy-on-write clones.
  4. Deploy from templates and clone VMs directly in the Web Templates / VM Cloning pages.

1. VM Lifecycle & Provisioning

Create, run, and reshape virtual machines with declarative or interactive workflows, backed by FluxVM.

- **Full VM Lifecycle** — Create, start, stop, restart, pause, resume, hibernate, and delete VMs backed by KVM, via FluxVM. — *One consistent lifecycle across CLI, web, Terraform, and Kubernetes.*
  - **How:** CLI `fabricctl start|stop|restart|delete <name>` (pause/resume are REST/Web only) · Web dashboard VM-list quick actions VMs view (`s` / `t` / `r` / `d`) · REST `POST /api/vms/:name/{start,stop,restart,pause,resume}` and `DELETE /api/vms/:name`.
- **Declarative Apply** — Define VMs in YAML and reconcile them with fabricctl apply -f config.yaml. — *GitOps-friendly infrastructure without a control-plane rewrite.*
  - **How:** CLI `fabricctl apply -f config.yaml` reconciles a YAML spec (version it in git); equivalent imperative path is REST `POST /api/vms`, or a `VirtualMachine` CRD via the K8s operator.
- **Cloning & Templates** — Full and linked copy-on-write clones plus reusable templates for rapid deployment. — *Stand up fleets from a golden image in seconds, not minutes.*
  - **How:** REST `POST /api/vms/:name/clone` (`linked_clone: true|false`) · templates via `POST /api/templates` and `POST /api/templates/:id/deploy` · Web Templates / VM Cloning pages (REST/Web only — no `fabricctl` clone command).
- **Hibernate & Checkpoint** — Suspend-to-disk hibernate, resume from snapshot, and VM checkpoint/restore and forking. — *Pause idle workloads and restore exact machine state on demand.*
  - **How:** Capture machine state with a full snapshot: REST `POST /api/vms/:name/snapshots` (`snapshot_type: Full`) then `POST .../snapshots/:id/revert` · Web VM detail Snapshots tab (REST/Web only — no `fabricctl` snapshot command).
- **Live Hotplug** — Hotplug CPU, memory, disk, and NIC into running VMs without a reboot. — *Scale a VM to demand while it keeps serving traffic.*
  - **How:** Web VM detail Hotplug tab (live CPU/memory/disk/NIC) · REST resource endpoints under `/api/system/vms/:name/...` · emits `cpu_hotplug` / `memory_hotplug` / `disk_attached` events on the SSE stream.
- **Disk Import & Conversion** — Import VMs from VMDK, VDI, and VHD with auto-conversion to qcow2, and online disk resize via QMP. — *Bring existing VMDK, VDI and VHD machines over without downtime.*
  - **How:** REST `POST /api/images/import` (VMDK/VDI/VHD → qcow2) and online resize `POST /api/images/:id/resize` · Web VM Create from imported image (REST/Web only — no `fabricctl` image command).
- **Golden Images** — Turn any configured VM's current disk into a new, independent, catalog-registered base image — not a template's config recipe or a live copy-on-write fork, an actual standalone qcow2 file other VMs can boot from even after the source VM is gone. — *Get a workload exactly the way you want it once, then stamp out fleets from it.*
  - **How:** REST `POST /api/images/from-vm/:name` (async job, poll `GET /api/images/convert/:id`) · Web Create VM → "Create golden image from a VM." Each image is certified with GuestKit's offline `doctor` boot-readiness analysis (not qemu-guest-agent — Zyvor Fabric uses GuestKit for all guest/image inspection) as soon as conversion finishes; the score attaches to the job once ready and requires no agent running inside the guest.

- **OS Image Catalog** — Download ready-made Ubuntu, Fedora, Debian, AlmaLinux, or Flatcar cloud images straight into the local catalog, reusing an already-downloaded copy instead of re-fetching it. — *Skip hunting for cloud-image URLs before your first VM.*
  - **How:** REST `GET /api/images/cloud` (catalog) and `POST /api/images/cloud/download` (async job) · Web Create VM → "Download an OS image."

> VMs are ordinary systemd units — journalctl, watchdogs, and socket activation work exactly as operators already expect.

2. Storage Orchestration

Six pluggable backends, live disk mobility, and a built-in cloud-image catalog.

- **Six Storage Backends** — Pool and volume management across Local, NFS, LVM, LVM-thin, ZFS, and Ceph/RBD. — *Use the storage you already run — no dedicated SAN required.*
  - **How:** REST `POST /api/storage/pools/{local,nfs,lvm,lvm-thin,zfs,ceph}`, list with `GET /api/storage/pools` · Web Storage page Storage view (type auto-detected).
- **Volume Management** — Full volume CRUD with attach/detach, online resize, and clone operations. — *Reshape storage for a workload without recreating the VM.*
  - **How:** Volume CRUD + attach/detach/resize/clone via the `/api/storage/...` volume endpoints · Web Storage → Volumes (capacity + attachment info) · `fabricctl ceph` covers Ceph pools/images specifically; other backends and generic volume CRUD are REST/Web only.
- **Snapshots & Retention** — Create and restore snapshots with configurable retention policies. — *Roll back a bad change in seconds and prune old state automatically.*
  - **How:** REST `POST /api/vms/:name/snapshots`, `GET .../snapshots/tree`, `POST .../snapshots/:id/revert` · Web VM Snapshots tab · retention applied per backup/snapshot policy.
- **Storage Live Migration** — Move VM disks between pools with no downtime, guided by SDRS recommendations. — *Rebalance or evacuate storage while VMs stay online.*
  - **How:** Trigger a disk move between pools (SDRS-recommended) via the datastore-cluster/storage-migration REST endpoints · Web Storage policies / Site Operations.
- **ZFS Replication** — Incremental ZFS send/receive replication between hosts and sites. — *Efficient, block-level DR copies that only ship the delta.*
  - **How:** Configure incremental send/receive between hosts via the replication REST endpoints · Web replication page · requires ZFS pools on both ends.
- **Cloud Image & ISO Catalog** — Built-in downloader for Ubuntu, Fedora, Debian, and Alma images plus ISO download/list/delete. — *Boot a fresh distro without hunting for images.*
  - **How:** REST `GET /api/images/cloud`, `POST /api/images/cloud/download`, ISOs under `GET /api/images/iso` · Web Content Library / Images · files land in `/var/lib/zyvor-fabricd/images`.

3. Software-Defined Networking

A full SDN stack — policies, firewalling, load balancing, VPN mesh, and observability.

- **Network Policies** — Cilium-style label-based ingress/egress rules enforced through nftables. — *Segment east-west traffic with identity, not brittle IP lists.*
  - **How:** Label-selector rules via the network-policy REST endpoints · Web Network Security → Policies (direction/priority/enforcement badges) Net Security → Policies (`S` sync / `d` delete).
- **Per-VM Firewall** — Firewall profiles and zones applied per VM via nftables, with IPv6 dual-stack support. — *Ship each workload with its own hardened perimeter.*
  - **How:** REST `GET/POST /api/firewall-profiles` (compiled to nftables) · Web Network Security → Firewall rule builder (protocol/port/CIDR/action) + zones + VM assignments Firewall tab.
- **Service Mesh** — Virtual-IP load balancing with round-robin, least-connection, random, and IP-hash strategies plus health checks. — *Front a pool of VMs behind one resilient endpoint.*
  - **How:** Virtual-IP service + backend pool via the service/load-balancer REST endpoints (algorithm selectable) · Web Network Security → Services Services tab.
- **WireGuard VPN Mesh** — Point-to-point, hub-spoke, and full-mesh WireGuard overlay tunnels. — *Securely stitch VMs across sites without external appliances.*
  - **How:** WireGuard tunnels/networks via the VPN REST endpoints · Web Network Security → VPN peer editor + topology selector VPN tab.
- **QoS Traffic Shaping** — Guaranteed and maximum rates, burst, and priority queuing via Linux tc. — *Protect critical workloads from noisy-neighbor bandwidth spikes.*
  - **How:** tc-based guaranteed/max rate, burst, and priority via the QoS REST endpoints · Web Network Security → QoS QoS tab.
- **DNS Policy & NAT Gateway** — Zone management, upstream servers, domain blocking, plus SNAT/DNAT/hairpin NAT gateways. — *Own DNS and egress routing for every tenant network.*
  - **How:** DNS zones/upstreams/blocking + SNAT/DNAT/hairpin via the DNS and `/api/networkd/...` REST endpoints (`POST /api/networkd/port-forwards` , `POST /api/networkd/dhcp`) · Web Network Security → DNS / NAT tabs.
- **Packet Mirror & Net Monitor** — Mirror sessions for traffic capture and per-VM bandwidth tracking with threshold alerts. — *Debug and meter network behavior without leaving the fabric.*
  - **How:** Mirror sessions via the mirror REST endpoints; per-VM bandwidth via `GET /api/network-metrics/:vm` with threshold alerts · Web Network Security → Mirror / Monitor tabs Net Security.

4. Security & Identity

JWT auth, enterprise SSO, RBAC, multi-tenancy, and encryption on every endpoint.

- **JWT Auth + RBAC** — JWT authentication with configurable expiry and 3-tier RBAC (Admin/User/Viewer) enforced on every endpoint. — *Least-privilege access is the default, not an add-on.*
  - **How:** `POST /api/auth/login` returns a bearer JWT (`auth.token_expiration_hours` , default 24); roles admin/user/viewer are enforced per endpoint · Web login form · CLI/SDK send the token in `Authorization` .
- **Enterprise SSO** — LDAP and OIDC/OAuth2 integration plus API keys for service-to-service auth. — *Plug into existing identity providers instead of a new user silo.*
  - **How:** Configure LDAP/OIDC in `zyvor-fabricd.toml` `[auth]` ; issue API keys for service-to-service calls · Web Administration → users/roles.
- **Multi-Tenancy** — Project isolation with member roles and per-project quotas. — *Give teams their own bounded slice of the fabric.*
  - **How:** Projects with member roles and per-project quotas via the tenant/project REST endpoints · Web Quotas + Administration pages.
- **Encryption & Secrets** — Encryption at rest, per-VM disk encryption with key rotation, and an encrypted secrets store with access policies. — *Protect data and credentials without external KMS scaffolding.*
  - **How:** Encrypted secrets store via `GET/POST/PUT/DELETE /api/secrets` (values always redacted in responses); per-VM disk encryption + key rotation · Web Administration → encryption keys.
- **Audit Logging** — Audit trail on all VM lifecycle operations with JSON/CSV export. — *Answer 'who did what, when' for compliance and RCA.*
  - **How:** Audit trail via `GET /api/audit/logs` with JSON/CSV export · Web Audit page (filter by user/action/resource/time) Logs view.
- **PKI & Certificate Manager** — CA creation, certificate issue/renew/revoke, automated rotation, and hardware attestation. — *Run internal TLS without a separate PKI product.*
  - **How:** CA create + cert issue/renew/revoke + rotation via the certificate REST endpoints; `./zyvor-fabricd-ctl tls` generates the web-server cert · Web Administration → certificates.
- **Compliance Scanning** — Built-in CIS, STIG, and PCI-DSS profiles with per-VM findings and remediation guidance. — *Prove and improve posture against recognized benchmarks.*
  - **How:** REST `GET /api/compliance/profiles` , `POST /api/compliance/scan/:vm` (`profile_id` cis-level1/cis-level2/stig/pci-dss/hipaa), `GET /api/compliance/results` · Web Compliance page · `[compliance]` config enables auto-scan.

> The codebase carries a documented multi-round security audit: zero unsafe Rust, no shell pipelines, parameterized queries, SSRF and path-traversal protection, and bcrypt-hashed credentials.

5. High Availability & Disaster Recovery

Live migration, fault tolerance, and multi-site recovery.

- **Live Migration** — Move running VMs between hosts with iterative rsync pre-copy and cutover, with progress tracking and cancel. — *Drain a host for maintenance without stopping workloads.*
  - **How:** Migrate a running VM between hosts (iterative rsync pre-copy + cutover) via the migration REST endpoints with progress/cancel · Web Site Operations.
- **Fault Tolerance & Fencing** — Continuous VM replication with automatic failover detection, fencing, and FT metrics. — *Survive a node loss with minimal recovery time.*
  - **How:** Continuous replication + automatic failover detection + fencing via the fault-tolerance REST endpoints (FT metrics exposed) · Web Site Operations → Fault tolerance.
- **Predictive DRS** — Distributed resource scheduling with demand forecasting, proactive placement, and affinity/anti-affinity rules. — *Keep clusters balanced before hotspots become outages.*
  - **How:** DRS with demand forecasting + affinity/anti-affinity rules via the DRS REST endpoints · Web Site Operations → DRS configuration & recommendations.
- **Site Recovery** — Recovery plans with planned migration, disaster failover, test failover, and reprotection workflows. — *Rehearse and execute cross-site DR with confidence.*
  - **How:** Build recovery plans (planned migration, disaster/test failover, reprotect) via the site-recovery REST endpoints · Web Site Operations → Site recovery.
- **Multi-Site Replication** — Cross-site VM replication with sync scheduling, RPO monitoring, and recovery instances. — *Meet recovery-point targets you can actually measure.*
  - **How:** Cross-site VM replication with sync scheduling + RPO monitoring + recovery instances via the replication REST endpoints · Web Site Operations.

6. Compute, GPU & Virtualization

Low-level control over CPU topology, memory, accelerators, and firmware.

- **GPU Passthrough** — Pass a physical NVIDIA or AMD GPU through to a VM via VFIO once it's bound to `vfio-pci` (generic PCI passthrough — no vGPU/mediated-device support). — *Run AI, rendering, and CAD workloads at near-bare-metal speed.*
  - **How:** REST `GET /api/system/pci-devices`, `POST` / `DELETE /api/vms/:name/devices/pci` · Web VM detail Devices tab; requires matching IOMMU/hardware and manually binding the device to `vfio-pci` first (no `fabricctl` command for this).
- **CPU Pinning & NUMA** — CPU topology control, pinning, NUMA-aware placement, and nested virtualization. — *Squeeze predictable performance from latency-sensitive VMs.*
  - **How:** REST `GET /api/system/numa-topology`, placement hint `GET /api/system/numa/placement`, and `POST /api/system/vms/:name/cpu-pinning` (Auto/NumaNode/Socket/Explicit) · Web VM advanced CPU settings.
- **Memory Optimization** — Memory ballooning, hugepages, and KSM page deduplication via a system resource manager. — *Fit more VMs per host without starving any of them.*
  - **How:** REST `POST /api/system/vms/:name/memory-ballooning`, `/memory-limit`, and `POST /api/system/hugepages/allocate`; KSM handled by the resource manager · Web VM settings.
- **vTPM & Secure Boot** — TPM 1.2/2.0 via swtpm with per-VM isolated state, UEFI, and Secure Boot firmware management. — *Support BitLocker, LUKS, and measured boot inside guests.*
  - **How:** Pass `tpm: true` / `secure_boot` in VMStartOptions on `POST /api/vms/:name/start` (swtpm gives each VM isolated state) · Web VM Create advanced boot/display settings · requires swtpm on the host.
- **cloud-init Provisioning** — NoCloud datasource generation for users, packages, network config, and SSH key injection. — *Boot fully configured VMs on first start, hands-free.*
  - **How:** REST `POST /api/vms/:name/cloud-init` (users, packages, runcmd, write_files → NoCloud ISO) · Web VM detail Cloud-init tab · applied on next start/restart.
- **OVA/OVF & Content Library** — OVA/OVF export and import plus a content library with cross-site image/template sync and customization specs. — *Standardize and share golden images across every site.*
  - **How:** Export via `POST /api/vms/:name/export` (OVA) and import to bring appliances in; cross-site sync + customization specs in the Content Library · Web Content Library page.

7. Monitoring, Automation & Operations

Metrics, scheduling, notifications, and self-checks that keep the fabric healthy.

- **Prometheus Metrics** — A /metrics endpoint exposing VM-count and lifecycle metrics (total/running/stopped, start/stop/create/delete counters), with a prebuilt Grafana dashboard. — *Drop into your existing observability stack instantly.*
  - **How:** Scrape `GET /metrics` (no auth) into Prometheus; live per-VM CPU/memory/disk/network detail via `GET /api/vms/:name/metrics` (not yet exported as Prometheus series) · Web Monitoring page · import the bundled Grafana dashboard.
- **Scheduling & Auto Backups** — Once/daily/weekly VM schedules plus automated daily backups and weekly state-store cleanup via systemd timers. — *Routine operations run themselves, on time, every time.*
  - **How:** Once/daily/weekly schedules via the schedule REST endpoints (backed by systemd timers) · Web Scheduling page (cron-style + one-time, with execution history).
- **Backup & Restore** — Per-VM and bulk backups with retention policies and incremental backups from web UI. — *Recover a single VM or the whole fleet on your own terms.*
  - **How:** REST `POST /api/backups` (full/incremental + retention), `POST /api/backups/restore`, policies via `POST /api/backups/policies` · Web Backups page (bulk).
- **Multi-Channel Notifications** — Email, Slack, Microsoft Teams, and webhook alerts with retry and backoff. — *The right people hear about problems the moment they happen.*
  - **How:** REST `POST /api/notifications/channels` (email/slack/webhook/teams) + `/rules`, verify with `POST .../channels/:id/test` · Web Notifications page · exponential-backoff retry (max 10 attempts).
- **Health & Auto-Verify** — Deep health checks (API, disk, DB, timers, KVM) and post-install smoke tests of API, auth, VM CRUD, and backups. — *Catch a broken deploy before your users do.*
  - **How:** Run `./zyvor-fabricd-ctl health` (API/disk/DB/timers/KVM) and `./zyvor-fabricd-ctl verify` (post-install smoke tests of API, auth, VM CRUD, backups) · health also exposed over REST.
- **Config Snapshots & Events** — Versioned config-snapshot API, retained lifecycle events, and an SSE event stream for time-machine correlation. — *Diff infrastructure over time and reconstruct incidents.*
  - **How:** Versioned config-snapshot REST API + recent events `GET /api/events` and live SSE `GET /api/events/stream` (created/started/stopped/migrated/error/...) · Web activity feed.

8. Interfaces & Automation Surfaces

Four first-class ways to drive the same daemon — pick per task, not per product.

Interface	Best for	Highlights
fabricctl CLI	Scripting & automation	JSON/YAML/table output, apply -f
Web dashboard	Operators & teams	80+ console pages under `/app` plus marketing routes, Ctrl+K palette, bulk ops
K8s operator	GitOps / K8s shops	VirtualMachine CRD reconciliation
Terraform / SDK	Infra-as-code	plan/apply, typed Rust + Python + Ansible

- **fabricctl CLI** — Scriptable command-line client with JSON/YAML/table output covering VM, policy, storage, and network operations. — *Automate anything the platform can do from a shell script.*
  - **How:** Install the `fabricctl` binary and point it at the daemon: `fabricctl list`, `fabricctl create web-01 --image fedora-41 --cpus 2 --memory 4096`, `fabricctl apply -f config.yaml`, with `--output json|yaml|table` (or `-o`).
- **Web Dashboard** — Hybrid marketing + `/app` console, Ctrl+K palette, light Apple UI, live WebSocket updates, and bulk operations. — *Give operators a full GUI without giving up the API.*
  - **How:** Browse to http://localhost:9095, sign in at `/sign-in`, open `/app` and log in as `admin`; `Ctrl+K` command palette, 80+ console pages under `/app` plus marketing routes, bulk ops, and live WebSocket/SSE updates — served by the daemon itself. A fresh install with no VMs yet shows a "Getting Started" panel instead of an empty table, linking straight to VM creation, templates, the playground, and access control. Deleting a VM is undoable for a few seconds via an Undo bar before the delete actually fires.
- **Console & VNC** — Browser terminal via xterm.js over WebSocket and graphical VNC via a noVNC proxy, authenticated with the same JWT. — *Reach any VM's console without exposing raw ports.*
  - **How:** Web VM Console (xterm.js) / VNC (noVNC) buttons · WebSocket `ws:///api/vms/:name/console?token=` (also `/ws/vnc/:name`) · `websocat` from the CLI.
- **Kubernetes Operator** — Manage VMs as VirtualMachine CRDs with continuous reconciliation via a Helm-installable operator. — *Define VMs alongside containers in the same GitOps flow.*
  - **How:** Install the Helm-packaged operator, then declare `VirtualMachine` CRDs; the operator continuously reconciles them against the Fabric API.
- **Terraform, SDK & Ansible** — A Terraform provider with full plan/apply, a typed Rust zyvor-fabric-sdk, plus Python and Ansible SDKs. — *Provision the fabric from whatever IaC tooling your team already uses.*
  - **How:** Use the Terraform provider (`terraform plan` / `apply`), the typed Rust `zyvor-fabric-sdk`, or the Python / Ansible SDKs — all target the same REST API.

9. Fleet, Cost & Governance

Datacenter hierarchy, resource pools, chargeback, and lifecycle compliance at scale.

- **Datacenter Hierarchy** — Model datacenters, clusters, and hosts with registration, heartbeat, maintenance mode, and auto-discovery. — *Organize sprawling infrastructure into a navigable topology.*
  - **How:** Register datacenters/clusters/hosts (heartbeat, maintenance mode, auto-discovery) via the datacenter REST endpoints · Web Administration → datacenter / hosts.
- **Resource Pools & Quotas** — CPU/memory/storage reservations with admission control, overcommit ratios, and pool-level quotas. — *Guarantee capacity to teams while preventing runaway usage.*
  - **How:** Pool reservations + overcommit ratios + admission control via the resource-pool/quota REST endpoints · Web Quotas page (usage-vs-limit visualization).
- **Billing & Chargeback** — Per-VM metering, configurable pricing tiers, invoice generation, and chargeback reports. — *Show every team the true cost of what they run.*
  - **How:** REST `GET/PUT /api/billing/pricing`, `GET /api/billing/usage`, `POST /api/billing/invoice/:tenant_id` · Web Monitoring/Analytics · rates set in `[billing]` config.
- **Lifecycle Manager** — Host baseline definitions, compliance scanning, remediation, and rolling updates with pause/advance. — *Keep every host on a known-good, patched baseline.*
  - **How:** Define host baselines and run remediation + rolling updates (pause/advance) via the lifecycle REST endpoints · Web Administration → lifecycle.
- **Distributed Storage Policies** — Datastore clusters, storage policies, SDRS recommendations, and compliance checking. — *Enforce storage placement rules automatically across pools.*
  - **How:** Define datastore clusters + storage policies and act on SDRS recommendations via the storage-policy REST endpoints · Web Storage policies page.

> Deployment scales from a single 4GB edge server to a 3+ node etcd cluster with shared storage and HA failover — same binary, same API.

Getting started

1. **Install in one command** — Clone the repo and run make build && sudo make install, or ./zyvor-fabricd-ctl deploy for an auto-sudo end-to-end setup.
2. **Start the service** — Run sudo systemctl enable --now zyvor-fabric, then read the auto-generated admin password with ./zyvor-fabricd-ctl password.
3. **Create your first VM** — Use fabricctl create web-01 --image fedora-41 --cpus 2 --memory 4096, or declare it in YAML and run fabricctl apply -f config.yaml.
4. **Open your interface of choice** — Open the web dashboard at http://localhost:9095 to manage the fleet.
5. **Verify and monitor** — Run ./zyvor-fabricd-ctl verify and ./zyvor-fabricd-ctl health, then scrape /metrics into Prometheus and import the bundled Grafana dashboard.

> **Good to know:** Zyvor Fabric requires Linux with systemd 256+ (Fedora, Ubuntu, Debian, RHEL, or SUSE) and KVM; it is not a hosted or Windows-server product. Some enterprise capabilities carry environmental prerequisites — swtpm for vTPM, an etcd cluster and shared/replicated storage for HA and live migration, and matching hardware/IOMMU for GPU passthrough. Multi-node HA is designed for 3+ nodes; single-server deployments run standalone. The published endpoint and page counts (480+ REST endpoints, 37+ web pages) reflect current documentation and may vary by release, and the macOS Machina workbench is a separate desktop product that consumes the Fabric API rather than part of this daemon.

---

*Zyvor Fabric is developed by ZyvorAI Labs. Contact **info@zyvor.dev** · Zyvor Production License v1.0.*