

skidbladnir — documentation

version 0.1.3

generated 2026-07-03

rendered by **nornir** · typst engine

Contents

1	README	4
2	Skidbladnir	5
2.1	Mashup — one sealed bundle onto a zero-network target	5
2.2	Read the real docs	5
3	Skidbladnir — complete overview	6
3.1	What it does — pack, move, unfold	6
3.2	Quickstart	7
3.3	The packet format — tar + zstd default, zippy behind a feature	7
3.4	Capabilities	7
3.5	Design	7
3.6	License	7
4	The Install Tree — a backend-neutral service-install abstraction for Skíðblaðnir	8
4.1	0. Why a tree	8
4.2	PART A — the modern service/init landscape (research)	8
4.2.1	A.1 systemd — system scope	8
4.2.2	A.2 systemd — <code>--user</code> scope	9
4.2.3	A.3 OpenRC	9
4.2.4	A.4 runit	9
4.2.5	A.5 s6 / s6-rc	9
4.2.6	A.6 dinit	9
4.2.7	A.7 SysV init (the universal fallback)	10
4.2.8	A.8 OCI / Docker	10
4.2.9	A.9 Kubernetes-adjacent — Podman quadlet / systemd-nspawn	10
4.2.10	A.10 Windows service (SCM) — for completeness	10
4.2.11	A.11 The cross-cutting abstraction these all share	11
4.3	PART B — the tree design	11
4.3.1	B.1 Shape	11
4.3.2	B.2 The types (landed in the spike, <code>src/service.rs</code>)	12
4.3.3	B.3 The two implemented renderers	12
4.3.4	B.4 Why OpenRC was the chosen second backend (not systemd-user)	13
4.3.5	B.5 How the existing <code>ComponentSpec</code> folds in	13
4.4	PART C — multi-OS install test matrix + the KVM introspection probe	13
4.4.1	C.1 Goal	14
4.4.2	C.2 The matrix axes	14
4.4.3	C.3 Reuse the existing <code>MachineController</code> (KVM + container)	14
4.4.4	C.4 The machine-injected introspection probe	15
4.4.5	C.5 Should the matrix REPORT to <code>workspace_skidbladnir</code> ?	15
4.4.6	C.6 Build order for Part C	15
4.5	PART D — tunnr: part of Skíðblaðnir, or a cooperating sibling?	16
4.5.1	D.1 What tunnr is (from the references — the repo is not yet checked out here)	16
4.5.2	D.2 Recommendation — cooperating sibling, composed by Skíðblaðnir; NOT absorbed	16
4.6	PART E — product install lifecycle (install vs update vs uninstall; data + log provisioning)	16
4.6.1	E.1 The three verbs	17
4.6.2	E.2 API (<code>src/install.rs</code>)	17
4.6.3	E.3 Seams (honest)	17
4.6.4	E.4 Tests (<code>src/install.rs</code> , offline into tempdirs)	18
4.7	Appendix — spike status	18
5	skidbladnir — crate card	19
5.1	Reverse-engineered (from source — verify, don't trust the name)	19
5.2	Fill in (one line each — only the human knows these)	19

6	skidbladnir <code>ZnippyArchive</code> → real <code>znippy-core</code> (one canonical archive format)	20
6.1	Good news: clean extraction	20
6.2	The real format (the truth)	20
6.3	Plan	20
6.4	Gate (compat, not perf)	20
7	skidbladnir-probe — crate card	21
7.1	Reverse-engineered (from source — verify, don't trust the name)	21
7.2	Fill in (one line each — only the human knows these)	21

README



Skidbladnir

Skidbladnir (Skiðblaðnir) is a 100% Rust airgap install framework: pack a whole deployment into one sealed, content-addressed bundle, carry it across the gap, and on the far side it creates the service, unpacks the zippy archive, and provisions the data & log dirs — starting every component in dependency order with no network and no shell on the target.

Mashup — one sealed bundle onto a zero-network target

Skidbladnir carries the *whole* deployment — infra **and** software — over an airgap in one content-addressed bundle and brings it up with **no agent, no SSH, no shell** on the target (it drives systemd straight over D-Bus). The rivals each cover only a slice:

capability	Skidbladnir	Ansible	Packer	cloud-init
One sealed, content-addressed bundle — infra + software + images + config + data, verify-on-receipt	☑	✗	● (image only)	✗
Zero network, zero shell on the target — drives systemd over D-Bus, no SSH/agent	☑	✗ (SSH + Python)	✗	● (first-boot shell)
Provision the machine (Redfish BMC · Terraform · KVM) and install the software, one tool	☑	●	✗	✗

The full seven-capability matrix — the `ServiceSpec` → PKI/systemd renderer tree, atomic version-swap + rollback, all-Rust offline — is in [the full overview](#).

Read the real docs

- ☑ [The complete manual — docs/book.pdf](#) — every subsystem chapter (pack · archive · move · unfold, PKI, layout, the Redfish/Terraform/KVM backends).
- ☑ [Full overview — .nornir/README-full.md](#) — the detailed pitch, quickstart, the full capability matrix, packet format, and design.
- ☑ [docs/book.md](#) — the manual, as Markdown.

Skidbladnir — complete overview

The **real** README: the full pitch, quickstart, capabilities and design. The repo-root [README.md](#) is the slim face (title · mashup · pointers); this chapter is the heavy detail, and it is also folded into the manual [docs/book.pdf](#) by `nornir docs book`.

100% Rust — airgap a **whole system** (the infrastructure *and* the software on it) across a gap with no network, no shell, no manual steps: pack everything into one sealed, content-addressed bundle, carry it across, and on the far side it provisions the machine, sets up PKI + systemd, and starts every component in dependency order.

Why Skiðblaðnir — it carries the *whole* deployment (infra **and** software) over an airgap in one bundle and brings it up with no agent on the target — the rows marked **NA** are categories the rivals have no concept for:

capability	Skidbladnir	Ansible	Packer	cloud-init
One sealed, content-addressed bundle — infra + software + images + config + data, verify-on-receipt	☒	✗	● (image only)	✗
Zero network, zero shell on the target — drives systemd over D-Bus, no SSH/agent	☒	✗ (SSH + Python)	✗	● (first-boot shell)
Provision the machine (Redfish BMC · Terraform · KVM) and install the software, one tool	☒	●	✗	✗
Typed ServiceSpec → renderer tree — PKI + user + systemd unit + ports + dated layout from one declaration	☒	NA	NA	NA
Dependency-ordered bring-up with atomic version-swap + rollback	☒	●	✗	✗
100% Rust , builds & runs fully offline (lean core pulls no network client)	☒	✗ (Python)	● (Go)	✗
Mature module ecosystem / huge community / multi-distro coverage	●	☒	☒	☒

Rows marked **NA**: the rival has no such category — not a missing feature, a missing concept.

Think of it as a **moving company for a deployment**. The flyttbil (moving van) packs *all your belongings* — every binary, config, container image and data file — into one sealed crate. But unlike a normal mover, it also brings the **cement mixer**: on the far side it pours the foundation first (provisions the machine, the OS, the network) and *then* unpacks the furniture (installs, configures and starts each component) — in the right order. You roll the crate across the airgap; the far side comes up running.

What it does — pack, move, unfold

- **pack** — gather everything the target needs: artifacts, target-identical dependencies (it can build a component in a container so it is byte-alike with the destination), images, config, data.
- **archive** — seal it into one movable bundle with a content-addressed manifest (SHA-256 per entry), so the far side can **verify-on-receipt** that nothing changed in transit.
- **move** — carry the single bundle across the gap by any means (it is just bytes — a disk, a one-way diode, a courier).
- **unfold** — on arrival: lay the foundation (provision hardware/VMs), unpack into a versioned software dir, set up the PKI, create the service user and systemd unit, open the firewall, and **start every component in dependency order**. No shell scripts — it talks to systemd over D-Bus directly.

It packs **and** unpacks: the same crate goes both ways.

Named for **Skíðblaðnir**, the ship in Norse myth that always has a fair wind and folds up small enough to fit in a pouch. The bundle folds an entire deployment into one thing you can carry.

Quickstart

```
# internet side — pack a staging tree into a signed tar.zst bundle + per-entry SHA-256
manifest:
nornir airgap pack ./staging --out lake.tar.zst

# ...carry lake.tar.zst across the gap (a disk, a one-way diode, a courier)...

# secure side — ZERO network on the target:
nornir airgap fetch lake.tar.zst      # verify-on-receipt against the manifest
nornir airgap unfold kvm my-vm        # show the provisioning plan (kvm | terraform |
redfish; --run to drive)
nornir airgap start lake.spec.toml    # dependency-ordered start waves → systemd over D-
Bus
```

The packet format — tar + zstd default, znippy behind a feature

The default archive is **tar + zstd**: pure Rust, always works offline, the dependable fallback. **znippy** — the content-cache + delta packet format (read without full decompress, ship only what changed) — is **fully implemented behind the znippy cargo feature** (`--features znippy`), not the default. Turn it on to use it; the tar+zstd path keeps working regardless.

Capabilities

- **platform** — per-OS prep, verify-on-receipt, unpack into the software dir.
- **pki** — `SelfGenerated` ed25519 certs (pure Rust, offline) or a Vault source.
- **layout** — version-dated `software/<date>/<product>` dirs with an atomic symlink swap on upgrade (and rollback).
- **product** — service user + systemd unit + configure + open ports.
- **unfold backends** — Redfish (bare-metal BMC) · Terraform (cloud/VM) · KVM (local test).
- **orchestrate** — dependency-ordered bring-up, emitting `airgap_events` .

Design

Lean by default: the offline core (pack · archive · layout · orchestrate · events) needs only a small always-offline dep set (`anyhow / serde /tar/zstd/sha2/chrono/tempfile/zbus`), so it **always builds offline**. PKI, the znippy engine, and the Redfish/Terraform/KVM backends sit behind cargo features — the lean core never pulls a network client it doesn't use.

Part of the [nornir](#) ecosystem. See `nornir/.nornir/airgap-install.md` for the full design + guide.

License

MIT OR Apache-2.0

The Install Tree — a backend-neutral service-install abstraction for Skíðblaðnir

One declarative `ServiceSpec` → every modern (2015+) Linux/embedded init & service system, with a multi-OS KVM install **test matrix** and a machine-injected introspection **probe** that reports “installed + started + healthy” as DATA.

Status (2026-06-30): design + landed spike. The spike (branch `agent/install-tree-spike`, worktree `/home/rickard/git/install-tree-spike-wt`) ships `src/service.rs`: the neutral `ServiceSpec` + the `ServiceRenderer` trait + **two real renderers** (systemd `systemd/ - -user` and OpenRC). `cargo test` green (82 passed), clippy clean. Everything below the spike (the other 5 init renderers, the test matrix, the probe) is design.

0. Why a tree

`src/systemd.rs` is excellent — but it is **one backend**. Its `SystemdServiceSpec` renders straight to a systemd `.service` unit and nothing else. Today every nordisk product (nornir, holger) couples its install story to systemd. That is fine on Ubuntu/Fedora/RHEL, but Skíðblaðnir’s whole pitch is “*airgap a whole system across a gap with no network, no shell, no manual steps*” — and the systems you airgap onto in 2026 are not all systemd:

- **Alpine / postmarketOS / embedded** → OpenRC (and increasingly **dinit**).
- **Void Linux / many appliance images** → runit.
- **Gentoo / hardened / s6 shops** → s6 + s6-rc.
- **Artix / Devuan / Chimera** → OpenRC | runit | s6 | **dinit**.
- **Ancient/locked RHEL6-era or busybox** → SysV fallback.
- **Container hosts** → OCI restart policies / Podman **quadlet** / `systemd-nspawn`.
- **Windows targets** → the SCM (already half-covered by `msi::WindowsServiceSpec`).

Hard-coupling to systemd re-creates exactly the per-init bespoke install scripts Skíðblaðnir set out to delete (the `x14-datalabb` Ansible problem). The fix is a **tree**: one declarative spec at the root, one renderer per backend at the leaves.

PART A — the modern service/init landscape (research)

A 2015+ install tree must cover the following. Each row is what a renderer has to emit and how the lifecycle verbs map. (The right-hand “abstraction” column is what §B’s `ServiceSpec` distils.)

A.1 systemd — system scope

- **Artifact:** INI unit `/etc/systemd/system/<name>.service` with `[Unit]` (`Description=`, `After=`, `Wants=` / `Requires=`), `[Service]` (`Type=`, `ExecStart=`, `User=` / `Group=`, `Environment=` / `EnvironmentFile=`, `Restart=`, `RestartSec=`, hardening: `NoNewPrivileges=`, `ProtectSystem=...`), `[Install]` (`WantedBy=multi-user.target`).
- **Verbs:** `systemctl daemon-reload`, `enable`, `start`, `is-active / status`, `stop`, `disable`. Skíðblaðnir issues these over **D-Bus** (`org.freedesktop.systemd1`, the existing `systemd::Systemd` zbus client) — no `systemctl` subprocess.
- **Scope:** system (root, dedicated user).
- **Logging:** `journald` (`journalctl -u <name>`).
- **Ordering:** `After=` / `Before=` (ordering) vs `Wants=` / `Requires=` (pull-in).
- **Socket activation:** a sibling `<name>.socket` unit (`ListenStream=`) + `Type=notify`.
- *Already implemented by* `src/systemd.rs`.

A.2 systemd — `--user` scope

- **Artifact:** same INI, in `~/.config/systemd/user/<name>.service`; omit `User=` / `Group=` (the unit already runs as you — systemd rejects them); `[Install] WantedBy=default.target`.
- **Verbs:** `systemctl --user ...`, over the per-user `session` bus.
- **Scope:** the invoking login user; no root, no `/etc`. State under `$HOME`.
- *Already implemented by* `src/systemd.rs` (`Scope::User`).

A.3 OpenRC

- **Artifact:** an `#!/sbin/openrc-run` shell-ish service script in `/etc/init.d/<name>`; settings as variables (`command=`, `command_args=`, `command_user=`, `pidfile=`, `directory=`, `command_background=true`). Env is auto-sourced from `/etc/conf.d/<name>`. Ordering/needs live in a `depend()` function (`need net`, `after <svc>`, `use <svc>`).
- **Restart supervision:** modern OpenRC uses `supervise-daemon` (`supervisor="supervise-daemon"`, `respawn_delay=`, `respawn_max=`) — the `Restart=` / `RestartSec=` twin.
- **Verbs:** `rc-update add <name> default` (enable at boot), `rc-service <name> start|status|stop`.
- **Scope:** system; user services exist (`/etc/init.d/.../user`) but are rare.
- **Logging:** stdout/err to syslog or `output_log=` / `error_log=`.
- **Socket activation:** none native (the service binds its own port).

A.4 runit

- **Artifact:** a service **directory** `/etc/sv/<name>/` with an executable `run` script (`exec chpst -u <user> <cmd> 2>&1`), optional `finish`, and a `log/run` for the logging chain (`svlogd`). Enable = symlink into the supervised dir `/var/service` (or `/run/runit/service`).
- **Restart:** automatic — runit *always* respawns `run` (to NOT restart you exit the script or use a `down` file / `check`). `RestartPolicy::No` $\Rightarrow$ a `run` that execs once and a `down` marker.
- **Verbs:** `ln -s /etc/sv/<name> /var/service` (enable), `sv up|status|down <name>`.
- **Scope:** system (per-user via `runsvdir` in a login session is possible).
- **Logging:** the `log/run` $\rightarrow$ `svlogd` directory chain.
- **Ordering:** no declarative graph — encode via `sv check` / `sv start <dep>` at the top of `run`, or a oneshot ordering service. (The renderer emits a `run` that blocks on deps with `sv check`.)
- **Socket activation:** none native.

A.5 s6 / s6-rc

- **Artifact (source form):** a **definition directory** per service holding a `type` file (`oneshot` | `longrun` | `bundle`), a `run` executable for longruns (`#!/bin/execlineb` or shell), optional `finish`, `notification-fd` (readiness), and a `dependencies` file (one dep service per line). A `bundle` groups services via a `contents.d/` dir. The source tree is **compiled** with `s6-rc-compile` into a read-only **compiled** DB.
- **Verbs:** `s6-rc-compile <compiled> <source>` (build), `s6-rc-update` (swap live DB), `s6-rc -u change <bundle>` (start/up), `s6-rc -d change <bundle>` (down), `s6-svstat` (status).
- **Restart:** s6-supervise always restarts a longrun; tune with `down-signal`, `max-death-tally`, a `finish` that exits non-zero to stop.
- **Scope:** system (per-user supervision trees are idiomatic for s6).
- **Logging:** a dedicated `<name>/log` longrun piping to `s6-log`.
- **Ordering:** the `dependencies` file (hard) — s6-rc never starts A before B is up.
- **Socket activation:** via `s6-ipcserver` / `s6-tcpserver` superservers in `run`.
- *Note:* s6-rc needs a **compile step**, so its renderer emits a source tree + records `s6-rc-compile` as a build verb (the only backend with a compile phase).

A.6 dinit

- **Artifact:** a textual service description in `/etc/dinit.d/<name>` (system) or `/etc/dinit.d/user` (user). Keys: `type = process | bgprocess | scripted | internal | triggered`, `command = ...`,

- **stop-command** = , **run-as** = , **restart** = yes|no , **pid-file** = , **logfile** = , **working-dir** = , **env-file** = , and three dependency kinds: **depends-on**: (hard — stop-propagates), **depends-ms**: (milestone — needed to start, but not stop-propagated), **waits-for**: (soft — may fail).
- **Verbs**: `dinitctl enable <name>` (and `start|status|stop|disable`).
- **Restart**: `restart = yes + restart-delay / restart-limit-count`.
- **Scope**: `system + user ( /etc/dinit.d/user , dinitctl --user )`.
- **Logging**: `logfile =` or `console`.
- **Ordering**: `depends-on / depends-ms / waits-for` — the **richest** 3-tier dep model of the lot (maps cleanly to `needs` vs `after`).
- **Socket activation**: not native; `triggered` type covers event-start.

A.7 SysV init (the universal fallback)

- **Artifact**: an LSB `/etc/init.d/<name>` shell script with an `### BEGIN INIT INFO ... ### END INIT INFO` header (`Provides`: , `Required-Start`: , `Default-Start`: 2 3 4 5) and `start|stop|restart|status` cases driven by `start-stop-daemon`.
- **Verbs**: `update-rc.d <name> defaults / chkconfig <name> on` (enable), `service <name> start|status|stop`.
- **Restart**: no supervision — needs a `respawn` line in `/etc/inittab` or a watchdog. `RestartPolicy` is best-effort here (documented limitation).
- **Scope**: system only. **Logging**: whatever the script redirects.
- **Ordering**: the LSB header's `Required-Start`: + runlevel symlink order.
- This is the lowest-common-denominator target; the renderer is the “it’ll always boot *something*” safety net.

A.8 OCI / Docker

- **Artifact**: not a file on disk but a **container run spec** — image ref, env, published ports, mounts, and a **restart policy** (`--restart=on-failure:5 / always / unless-stopped`) the runtime’s own supervisor honors. The unit fields map to `docker run / podman run` flags or a compose service.
- **Verbs**: `podman create|start` , `podman ps / inspect` (status), `stop`.
- **Scope**: system or rootless (rootless podman ≈ the `--user` analogue).
- **Logging**: `podman logs` (json-file / journald driver).
- **Ordering**: compose `depends_on` / pod start order. **Socket activation**: systemd socket → container, or the app binds inside.

A.9 Kubernetes-adjacent — Podman quadlet / systemd-nspawn

- **Artifact**: a **quadlet** `<name>.container` file in `/etc/containers/systemd/` (or `~/.config/containers/systemd/`) that `podman-systemd` (the `podman-system-generator`) renders into a transient systemd `.service` at daemon-reload. Keys: `[Container]` (`Image=` , `Exec=` , `PublishPort=` , `Environment=` , `User=`), `[Service]` (`Restart=`), `[Install]` (`WantedBy=`). This is the clean bridge “declarative container ⇒ systemd-managed”.
- **Verbs**: `systemctl daemon-reload` then `systemctl start <name>` (the generated unit). **systemd-nspawn**: `machinectl start <name>` over a container image in `/var/lib/machines`.
- This is the highest-leverage container target because it **reuses the systemd renderer’s verbs** — a quadlet is a systemd unit once generated.

A.10 Windows service (SCM) — for completeness

- **Artifact**: an SCM registration (the existing `msi:WindowsServiceSpec`: account `LocalSystem`, `Start = Auto|Manual|Disabled`, `delayed-auto`, env block in `HKLM\SYSTEM\CurrentControlSet\Services\<name>`). Built into the MSI by WiX.
- **Verbs**: `sc create|start|query|stop / New-Service` (PowerShell); `msiexec /i ... /qn` for the packaged install.
- **Scope**: machine (`LocalSystem`) or a named service account.

- **Logging:** Windows Event Log. **Ordering:** DependOnService.

A.11 The cross-cutting abstraction these all share

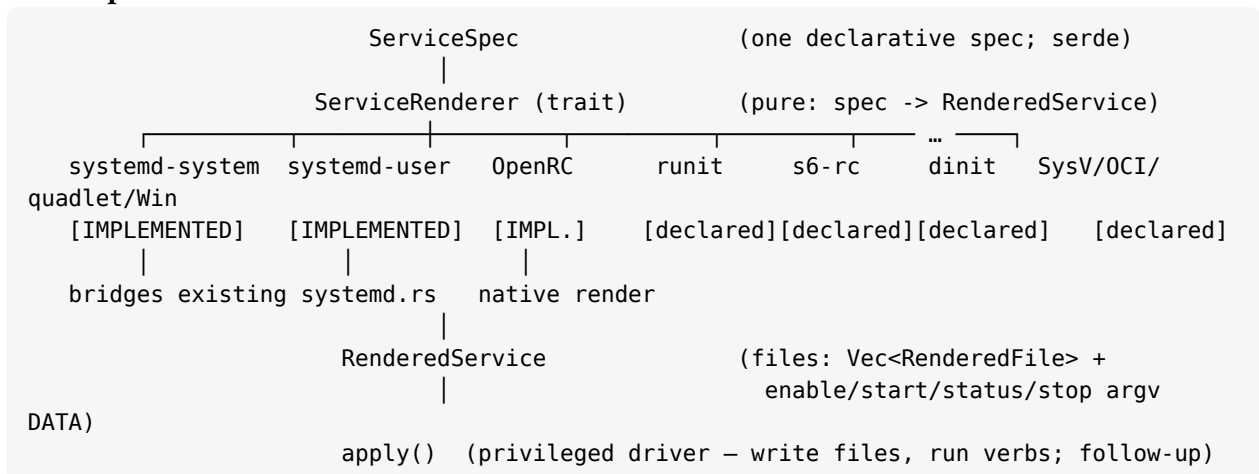
Strip the syntax and **every** backend is the same six facts plus verbs:

Neutral concept	systemd	OpenRC	runit	s6-rc	dinit	SysV	OCI/quadlet	Windows
what to run	ExecStart=	command= + command_args=	run exec	run	command=	start) case	Exec= / cmd	service binary
as whom	User= / Group=	command_u	chpst -u	s6-setuid	run-as=	--chuid	User= / rootless	account
scope	system / --user	system	system	system/ user	system/ user	system	system/ rootless	machine/ account
env	Environment= / Environment<n>	/etc/conf.d/<n>	chpst -e dir	s6-envdir	env-file=	export in script	Environment=	env block
ordering	After=	depend() {after}	sv check	dependenc	waits-for:	Required-Start:	depends_on	DependOnService
hard need	Requires= + After=	depend() {need}	block in run	dependenc	depends-on:	Required-Start:	pod/compose	DependOnService
restart	Restart= + RestartSec=	supervisord daemon + respawn_delay	always (default)	always (default)	restart= +delay	inittab respawn	--restart=	recovery actions
status verb	is-active	rc-service status	sv status	s6-svstat	dinitctl status	service status	podman inspect	sc query

That table is the `ServiceSpec`. One declarative spec carries the left column; each renderer knows its own row.

PART B — the tree design

B.1 Shape



The tree is a strict **generalization** of `src/systemd.rs`, not a replacement: the systemd renderer *bridges* to the existing tested `systemd::systemd_unit_scoped`, so the install limb that already ships is reused, never forked. `ServiceSpec` reuses `systemd::{Scope, RestartPolicy}` as shared vocabulary.

B.2 The types (landed in the spike, `src/service.rs`)

```
pub enum Backend {
    SystemdSystem, SystemdUser, OpenRc, // implemented
    Runit, S6Rc, Dinit, SysV, Oci, Quadlet, WindowsService, // declared
}
impl Backend { fn all() -> &'static [Backend]; fn slug(self) -> &'static str; }

pub struct ServiceSpec { // ONE declarative spec
    name, description, exec, args,
    user: Option<String>, // None => run as caller (required for Scope::User)
    scope: Scope, // System | User (reused from systemd)
    working_dir: Option<String>,
    env: Vec<(String,String)>,
    after: Vec<String>, // ordering edges
    needs: Vec<String>, // hard requires
    restart: RestartPolicy, // No | OnFailure | Always (reused)
    restart_sec: u32,
    sockets: Vec<Socket>, // socket activation / declared ports
}

pub struct RenderedFile { path: String, contents: String, mode: u32 }
pub struct RenderedService { // the render output: files + verbs AS DATA
    backend: Backend,
    files: Vec<RenderedFile>,
    enable: Vec<String>, start: Vec<String>, status: Vec<String>, stop: Vec<String>,
}

pub trait ServiceRenderer { fn backend(&self)->Backend; fn render(&self,&ServiceSpec)->RenderedService; }
pub fn renderer_for(b: Backend) -> Option<Box<dyn ServiceRenderer>>; // None = honest
"not yet"
```

Two design choices worth calling out:

1. **Verbs as data.** `RenderedService` carries `enable/start/status/stop` as argv `Vec<String>`, *not* hidden inside an `apply()`. This makes the whole tree unit-testable with zero privilege (“OpenRC enables with `rc-update add holger default`” is an assertion, not a side effect) and lets a single privileged driver run them uniformly across every backend. systemd’s verbs additionally route through the existing D-Bus `Systemd` client rather than `systemctl`.
2. **renderer_for returns Option.** Declared-but-unrendered backends resolve to `None` — an honest “not yet”, never a faked success. This mirrors the repo’s law (cf. the KVM `exec` “not yet” stub). `Backend::all()` + `slug()` give the coverage matrix its backend axis.

B.3 The two implemented renderers

- **SystemdRenderer { scope }** — projects `ServiceSpec` → `SystemdServiceSpec` and calls the existing `systemd_unit_scoped`. `after` → `After=`; `needs` → `After= + Wants=`; user scope drops `User= / Group=` and targets `default.target`; verbs become `systemctl [--user] <verb> <name>.service`.
- **OpenRcRenderer** — emits `/etc/init.d/<name>` (`#!/sbin/openrc-run`, `supervisor="supervise-daemon"` for restart) + `/etc/conf.d/<name>` (`env, mode 0600`); `depend()`

carries `need net` (the `network-online.target` twin) + the spec's `need / after` edges; verbs become `rc-update / rc-service`.

B.4 Why OpenRC was the chosen second backend (not systemd-user)

The deliverable offered “systemd-user or OpenRC”. systemd-user is **nearly free** — it is the same INI minus `User=`, already handled by the existing `Scope::User`. Choosing it would prove little. **OpenRC is a genuinely different format** (a shell service script, not INI; `depend()` not `After=`; `rc-update` not `systemctl`), so it *proves the abstraction is init-neutral* rather than “systemd with a coat of paint”. The spike therefore lands **three** renderers (systemd-system, systemd-user *for free via the bridge*, and OpenRC) — the strongest demonstration per unit of code. Next-highest-value renderers, in order: **dinit** (richest dep model, modern, embedded-relevant), **runit** (Void/appliances), **quadlet** (reuses systemd verbs), then `s6-rc` (needs the compile step), then SysV (fallback).

B.5 How the existing ComponentSpec folds in

`product::ComponentSpec` (`name/binary/user/ports/after/restart/health`) stays the *airgap* unit-of-installation; it gains a `to_service_spec(&layout, backend) -> ServiceSpec` (sibling to today's `to_systemd_spec`), so the whole AIRGAP6 install path can pick a backend at install time. **Landed (2026-07-01)**: the bridge is `ComponentSpec::to_service_spec(&self, layout, backend)` — it wires `working_dir` + `exec` through the `current` symlink (using the `FULL bundle_binary` relative path, because the lifecycle path unfolds the whole product tree with `extract_all`, not a flattened single binary), exposes the data + log dirs as `DATA_DIR / LOG_DIR` env (plus the legacy `<NAME>_DATA`), maps `after` → ordering edges and each health/listen port → a `Socket`, and picks `Scope::User` for `Backend::SystemdUser`. `LakeSpec` is unchanged — the dep graph + start-wave orchestration is backend-blind.

Layout note (2026-07-01): `layout::Layout` gained a `log_dir` root (e.g. `/var/log`) beside `data_dir`, a `product_log(name) → log_dir/<name>` path builder, and a `prepare_product(name)` that idempotently provisions ALL THREE per-product dirs — `software/<version>/<name>`, `data/<name>` AND `log/<name>` — returning them as `(software, data, log)`. `Layout::dated` now takes `(data_dir, log_dir, software_dir)`. Root `prepare()` also creates the `log_dir` root. This is the “create data dir + create log dir” half of a product install (see PART E).

PART C — multi-OS install test matrix + the KVM introspection probe

Status (2026-06-30): LANDED + LIVE. `src/matrix.rs` (feature `matrix`) + the injected `probe/` crate (static-musl `skidbladnir-probe` + `skidbladnir-testsvc`)

- the finished `src/machine.rs` KVM seam implement this whole section. A live run booted **4 OS guests** and the probe round-tripped a real `ProbeReport` from each:

OS	renderer	scope	result
Ubuntu 24.04	systemd-system	system	PASS
Debian 12	systemd-system	system	PASS
Fedora 42	systemd-system	system	PASS
Alpine 3.21	OpenRC	system	PASS

Channels are pure-Rust + *airgap-correct*: the cloud-init seed is a **FAT image** (`fatfs`, no `genisoimage`), the payload (`probe+testsvc+units`) rides a second FAT image, and the probe reports back over an `org.skidbladnir.report` **virtio-serial** port (host reads a file — works

under `NetMode::Isolated`). Rows sink to `workspace_skidbladnir/probe_reports.jsonl` + `airgap_events` (`op=matrix_probe`). The only external process is `qemu` (the hypervisor). Run: `cargo test --features matrix --test install_matrix_live -- --ignored --nocapture`.

C.1 Goal

Assert, as **data**, that a real product (holger / nornir) **installs + starts + stays healthy** across multiple OSES ($\Rightarrow$ multiple init backends) inside KVMs. Not “the renderer produced a plausible string” (that is $\S$ B’s unit tests) but “the string, written into a real Alpine/Void/Debian guest, actually brought the service up”.

C.2 The matrix axes

OS image	× init backend	× scope	= a matrix cell
Ubuntu 24.04	systemd-system	system	
Fedora 41	systemd-system	system	
Debian 12	systemd / --user	system+user	
Alpine 3.20	OpenRC	system	
Void (glibc/musl)	runit	system	
Artix	OpenRC runit s6 dinit		(declared backends)
Chimera	dinit	system	
(Windows Server)	SCM	machine	(out-of-band, MSI)

A cell is `(image, backend, scope, ServiceSpec)`. The matrix is the cartesian product filtered to `renderer_for(backend).is_some()` — so today it runs the `systemd` + `OpenRC` cells and **skips** (honestly, as `Skipped{reason:"no renderer"}`) the rest, growing automatically as renderers land. This is the same honesty law as the coverage gate in `nornir-testmatrix`.

C.3 Reuse the existing MachineController (KVM + container)

`src/machine.rs` already has exactly the harness: `MachineController` trait (`start / stop / exec / wait_ready`), `MachineSpec` (`image`, `NetMode::Isolated` for true airgap, `Provision::CloudInit`, `mounts`, `Readiness`), the content-addressed `ImageCache` (verify-on-receipt, `import_znippy_set`), `KvmController`, `ContainerController`, and `needs_real_machine()` ($\rightarrow$ KVM for `systemd/init/airgap`, container for `fast/stateless`). **The matrix is built on this — no new hypervisor code.** Per cell:

1. resolve the OS base image from `ImageCache` (znippy-carried across the gap)
2. `MachineController::start(MachineSpec{ image, net: Isolated, provision: CloudInit{...} })`
3. `wait_ready(SystemdUnit | TcpPort)` // real init came up
4. push the airgap bundle (mount or copy) + the renderer's `RenderedService` files
5. run the `RenderedService.enable/start` verbs in-guest (exec)
6. drop + run the PROBE (next $\S$); collect its JSON report (exec)
7. assert probe says Running + correct PID/port/health; `stop()`

Gap to close first (already tracked in `machine-control-design.md` §6): `KvmController::exec` and `wait_ready(SystemdUnit)` are **deferred** — they need a guest channel (`qemu-guest-agent QMP guest-exec`, or an ssh key seeded via `cloud-init`). The matrix needs this. **The probe (below) is the cheapest way to close it:** instead of a general `guest-exec`, inject one static probe binary and have it report over a known channel, sidestepping the need for arbitrary in-guest `exec` for the assert step. Steps 4–5 still need *a* push/exec; `cloud-init runcmd` (rendered into the NoCloud seed ISO — also deferred but small) can run the install + the probe at first boot, so the matrix needs **zero** interactive guest `exec`: the guest self-installs and the probe self-reports.

C.4 The machine-injected introspection probe

A **tiny static Rust executable** (`skidbladnir-probe` , musl static, no deps beyond `std`) injected into each KVM. It introspects the just-installed service and emits **one JSON line** of ground truth:

```
{
  "service": "holger", "backend": "openrc", "os": "alpine-3.20",
  "running": true, "pid": 1417,
  "ports": [{"port": 7070, "open": true}],
  "health": {"kind": "tcp", "ok": true},
  "logs_tail": ["...last N lines..."],
  "asserted_at": "2026-06-30T12:00:00Z"
}
```

How it introspects, backend-neutrally (it reads the SAME `ServiceSpec` it was shipped with, so it knows the unit name, user, ports, health check):

- **running? / pid?** — query the active init: systemd via the `/run/systemd` D-Bus `ActiveState / MainPID` (reuse the `systemd::Systemd` client!); OpenRC via `rc-service <name> status` / the pidfile; runit via `sv status`; dinit via `dinitctl status`. The probe carries a *mini* version of `renderer_for`'s status knowledge. Lowest-common-denominator fallback: scan `/proc` for the exec path.
- **port open?** — a `TcpStream::connect("127.0.0.1:<port>")` from inside the guest (the probe is *in* the netns, so this is true even for `NetMode::Isolated`).
- **health** — run the `ComponentSpec::HealthCheck` (Tcp connect / Http GET /health).
- **logs** — tail `journald` (`sd-journal`) or the backend's logfile for context on failure.

Report channel (how the probe gets data back out of an airgapped guest):

1. **Primary** — `virtio-serial` / `fw_cfg` **write-back**: the probe writes its JSON to a virtio-serial port (`/dev/virtio-ports/org.skidbladnir.probe`) that the host `KvmController` reads — works with `NetMode::Isolated` (no network needed). This is the clean airgap-correct channel and is a *small* `KvmController` addition (add one `-device virtserialport` to the argv + read the host-side socket).
2. **Fallback** — **shared mount**: the probe writes `/<mount>/probe-report.jsonl`; the host reads it off the bind mount after `stop()`. Zero new device code (mounts already exist in `MachineSpec`).
3. **Container cells**: trivial — `ContainerController::exec` already works, so the probe just prints to stdout and the controller captures it.

The probe is **the introspection seam**: it converts “did the install work?” from a human eyeballing `journalctl` into a typed `ProbeReport` row the matrix asserts on.

C.5 Should the matrix REPORT to `workspace_skidbladnir`?

Yes. Reuse the existing `events::AirgapEvent / JsonlSink` (already “persist-to-warehouse” by design — one row per pack/fetch/unfold/install/start boundary). Add an `op::MATRIX_PROBE` event carrying the `ProbeReport` per cell, sunk to `<workspace_skidbladnir>/airgap_events.jsonl`. `nornir`'s autonom completeness gate (`nornir/src/autonom.rs` , the `surface_coverage` Iceberg table) can then ingest the matrix the same way it ingests other surfaces — a (backend × os) install-coverage axis: green ⇔ every implemented backend has a passing install+probe on at least one OS. This makes the install tree's completeness a **mechanically-checked datum**, not a claim, exactly matching the nordisk functional-matrix philosophy. Concretely: the matrix writes `ProbeReport` rows to `workspace_skidbladnir`; `nornir`'s testmatrix reads them as an `InstallCoverage` surface. `Skidbladnir` stays the *producer*; `nornir` stays the *verdict engine*.

C.6 Build order for Part C

1. `ProbeReport` type + `skidbladnir-probe` bin (static, reads `ServiceSpec` , `/proc` + TCP introspection) — pure, unit-testable on the host first.

- virtio-serial write-back in `KvmController` (+ cloud-init seed-ISO render — the two deferred `machine.rs` items, both small).
- The `Matrix` driver: cartesian (image × backend), `renderer_for` filter, per-cell `MachineController` run, collect `ProbeReport`s.
- Sink `ProbeReport` → `AirgapEvent` → `workspace_skidbladnir`; `nornir` ingests.

PART D — tunnr: part of Skíðblaðnir, or a cooperating sibling?

D.1 What tunnr is (from the references — the repo is not yet checked out here)

`tunnr` is Rickard’s invention for **thin, OS-less laptops/desktops**: a pure-Rust UEFI appliance (its “binary” in the airgap bundle is a **disk image**, not an ELF). The references are consistent: `product::ComponentSpec::tunnr()` declares it as `run --appliance`, user `_tunnr`, HTTPS admin on 8443, HTTP `/healthz`; the install-system doc calls it `type: Appliance` (vs `Service`), writes the image to a block device / loop file, and registers it with a **tunnr KVM supervisor** (itself a service that manages VMs). It is the “OS-less client” endpoint of the stack — a machine that boots straight into the appliance image with no general-purpose OS.

D.2 Recommendation — cooperating sibling, composed by Skíðblaðnir; NOT absorbed

`tunnr` should remain its **own leaf repo**, a sibling of Skíðblaðnir, for the same reason Skíðblaðnir was extracted out of `nornir` (the Cargo.toml comment says it plainly: “so leaf consumers — `nornir`, `holger`, **tunnr** — can pull the airgap framework without a release-cascade”). Folding `tunnr` into Skíðblaðnir would re-couple the very things that extraction decoupled, and would drag a UEFI/appliance image-build toolchain into the lean airgap core. They compose cleanly **without** merging:

- Skíðblaðnir is the mover; tunnr is one kind of cargo.** `tunnr` is *described* by a spec (`ComponentSpec::tunnr()` already exists) and *carried* by the bundle — exactly like `holger`, which is also a sibling, not a dependency. Skíðblaðnir never links `tunnr`’s code; it ships its image and renders its install.
- tunnr is an Appliance, which the tree should model as a first-class kind.** Add `ServiceSpec`’s sibling for non-service installs: an `ApplianceSpec` (write image to block dev / loop, register with the `tunnr` KVM supervisor, health via `tunnr status <vm> == BOOT-OK`). The install tree then has two arms — **service** (this doc, all the init backends) and **appliance** (`tunnr`’s image-to-metal). The `MachineController` / `ImageCache` that Part C uses to *test* installs is the SAME machinery `tunnr`’s supervisor uses to *run* appliances — so they share the KVM/image layer without sharing a crate.
- The dependency direction is one-way and clean:** `tunnr` → depends on → Skíðblaðnir (pulls the airgap framework + the install tree + `MachineController`). Skíðblaðnir → only *describes* `tunnr` (a registry spec). No cycle.
- Why not absorb, concretely:** (1) release cadence — an appliance image format changes on a different clock than the airgap engine; (2) leanness — the embedded Skíðblaðnir client must not pull a UEFI image builder; (3) blast radius — `tunnr` is an *endpoint product*, Skíðblaðnir is *infrastructure* every product depends on.

Verdict: keep `tunnr` a sibling. Have Skíðblaðnir grow an `ApplianceSpec` arm (parallel to `ServiceSpec`) and let `tunnr` consume Skíðblaðnir’s install tree + machine layer. They cooperate; they do not merge. When the `tunnr` repo lands, the first concrete step is moving `ComponentSpec::tunnr()`’s appliance knowledge into an `ApplianceSpec` that `tunnr` owns and Skíðblaðnir renders — the same “product-owns-its-spec, engine-applies-it” split `holger` already follows.

PART E — product install lifecycle (install vs update vs uninstall; data + log provisioning)

Status (2026-07-01): LANDED. `src/install.rs` turns install from a binary-swap into a first-class **product** lifecycle over the neutral service tree (PART B) + the AIRGAP5 layout (now with a per-product LOG dir, see the B.5 layout note). Every op is **verbs-as-data**: the lifecycle fn does the UNPRIVILEGED filesystem work (dirs,

unfold, unit files under a fs `root`) and returns the privileged service-manager argv as data — a thin `apply()` runs them. That split makes the whole lifecycle unit-testable offline into a tempdir.

E.1 The three verbs

Rickard's framing — install ≠ update ≠ uninstall:

- **install** = (1) create the service, (2) unpack the airgapped product from its znippy archive, (3) create the data dir + create the log dir.
- **update** = swap exe / new generation, **preserve** data + log, restart.
- **uninstall** = delete the service; optionally purge data + log.

E.2 API (`src/install.rs`)

```
pub enum InstallOp { Install, Update, Uninstall }

pub struct InstallOutcome {
    pub op: InstallOp,
    pub product: String,
    pub backend: Backend,
    pub version: String,
    pub dirs: Vec<PathBuf>,          // created/ensured: software/<ver>/<name>, data/
    <name>, log/<name>
    pub removed: Vec<PathBuf>,      // purged data/log + deleted unit files (uninstall)
    pub files_written: Vec<PathBuf>, // unit/script/env files (under the fs `root`)
    pub identity: Option<String>,   // minted (install) / preserved (update) wire
    string
    pub rendered: RenderedService,  // file bodies + enable/start/status/stop verbs
    pub verbs: Vec<Vec<String>>,    // the ORDERED privileged argv to run for this op
}

pub fn install_product(spec, layout, bundle, archive, backend, root) ->
Result<InstallOutcome>;
pub fn update_product (spec, layout, bundle, archive, backend, root) ->
Result<InstallOutcome>;
pub fn uninstall_product(spec, layout, backend, root, purge: bool) ->
Result<InstallOutcome>;
pub fn apply(&InstallOutcome) -> Result<()>; // the thin privileged driver (runs
`verbs`)
```

- **root** is the filesystem root the absolute unit paths (`/etc/systemd/system/...`, `/etc/init.d/...`) are re-rooted under — `/` in production, a tempdir in tests, so writes are assertable without privilege.
- **install_product**: `layout.prepare_product` (software+data+log) → `archive.extract_all` (unfold the WHOLE product) → `+x` the binary → `layout.activate()` (atomic `current` swap) → `identity::ensure` under the data dir → render for `backend` → write unit files → `verbs = [enable, start]`.
- **update_product**: same, but `layout` carries the NEW `version`; `prepare_product` is idempotent so an existing `data/<name> + log/<name>` is **kept, never clobbered**, and `identity::ensure` LOADS the existing id (the reinstall-stable install id survives). Old generation retained for rollback. `verbs = [stop, start]` (restart; the unit points through `current`, so it never churns → no daemon-reload).
- **uninstall_product**: render → remove the unit/script/env files → if `purge`, `remove_dir_all` the `data/<name> + log/<name>` (else retain) → `verbs = [stop, disable]`. Never touches the shared software generation.

E.3 Seams (honest)

- No native `StateDirectory=` / `LogsDirectory=`. `SystemdServiceSpec` has no such field, so the data + log dirs are surfaced as `Environment=DATA_DIR/LOG_DIR + WorkingDirectory=` rather than the native

systemd directives; the dirs are provisioned explicitly by `prepare_product`, so nothing is lost — but a future pass could add real `StateDirectory=` / `LogsDirectory=` on the systemd renderer.

- **extract_all vs extract_one.** The lifecycle unfolds the whole product tree (`extract_all`, tree preserved), so `to_service_spec`'s `exec` is the full `bundle_binary` relative path under `current/<name>`. The legacy AIRGAP6 `SystemdInstaller` / `to_systemd_spec` path FLATTENS via `extract_one` to `current/<name>/<binfile>` (`installed_binary`). Both are correct for their own unpack; they are deliberately not merged here.
- **No firewall step.** `install_product` does not open ports (the AIRGAP6 `SystemdInstaller` did); ports are declared as `Socket` s on the neutral spec and can be opened by the same `open_ports` helper when a privileged driver wires it.

E.4 Tests (`src/install.rs`, offline into tempdirs)

- `install_product_on_systemd_system`, `install_product_on_openrc` — full inject-assert: all three dirs exist, whole product unfolded + binary +x, `identity.toml` under data, `current` resolves, unit files reference the data + log + exec paths, verbs = `[enable, start]` (OpenRC: `rc-update add` + `init.d / conf.d` files).
- `update_product_preserves_data_and_log_and_swaps_version` — writes real state into data + log, updates to a new gen, asserts both survive, install id preserved, `active_version` swapped, old gen retained, verbs = `[stop, start]`.
- `uninstall_product_purge_flag_controls_data_retention` — `purge=false` keeps data+log and removes the unit (verbs = `[stop, disable]`); `purge=true` removes data+log too.
- `install_outcome_json_roundtrips` — the outcome is plain serde data.
- Layout: `prepare_product_creates_software_data_and_log_dirs` (idempotent, never clobbers). Bridge: `component_spec_renders_neutral_service_spec_with_data_and_log`.

Appendix — spike status

- **Branch:** `agent/install-tree-spike` (worktree `/home/rickard/git/install-tree-spike-wt`). **Not pushed.**
- **Landed:** `src/service.rs` (the tree) + a one-line `lib.rs` module hook.
- **Renderers:** `systemd-system`, `systemd-user` (both via the existing `systemd_unit_scoped` bridge), OpenRC (native).
- **Tests:** 6 inject-assert tests incl. `one_spec_renders_to_systemd_and_openrc` (the headline: ONE `ServiceSpec` → both backends, each in its native shape). `cargo test` green (82 passed, 1 ignored), `cargo clippy` clean.
- **Not done (design only):** the other 5 init renderers, the `apply()` privileged driver, `ComponentSpec::to_service_spec`, the matrix driver, the probe binary, the `virtio-serial/cloud-init` machine.rs additions, the `ApplianceSpec` arm.

skidbladnir — crate card

Reverse-engineered (from source — verify, don't trust the name)

- **One-line pitch:** Skiðblaðnir — a 100% Rust framework to pack a product (or whole product lake) into an air-gapped bundle, move it across the gap, unfold the target (Redfish / Terraform / KVM), and install+configure+start every product in dependency order — no shell glue.
- **Does:** Implements the pipeline `pack → archive → fetch ||airgap|| → unfold → install → start(ordered)`. Modules map to AIRGAP steps: `pack` (staging tree + VERSION), `archive` (tar+zstd bundle + content-addressed manifest; zippy engine behind a feature), `fetch` (verify-on-receipt, zero network), `platform` (os-release detect/unpack), `pki` (`PkiSource` : SelfGenerated ed25519 / ExistingVault / TempVault, real X.509 via rcgen), `layout` (version-dated dirs + atomic symlink swap), `product` / `install` (ComponentSpec + generalized Installer, install/update/uninstall over a neutral service tree), `service` (backend-neutral ServiceSpec → systemd/OpenRC renderers), `systemd` (pure-Rust systemd via zbus, no `systemctl`), `identity` (UUIDv7 install_id + BLAKE3 machine_fp), `unfold` (Redfish/Terraform/KVM backends), `machine` / `matrix` (KVM/container lifecycle + live multi-OS install matrix), `orchestrate` (topo-sorted start waves), `events`, `msi` (WiX-in-podman + UUIDv5 GUIDs). `config::LakeSpec` is the declarative descriptor.
- **Key deps:** always-on pure-Rust core — `anyhow`, `serde` / `serde_json`, `tar` + `zstd`, `sha2`, `blake3`, `uuid v7`, `hex`, `chrono`, `zbus` (systemd D-Bus). Feature-gated: `ed25519-dalek` / `rcgen` / `x509-parser` / `rustls-pki-types` / `time` (`pki`, default on), `request` + `base64` (`vault` / `redfish`), `fatfs` + `libc` (`kvm`, pure-Rust FAT authoring for cloud-init seed/payload). No sub-crates in this workspace (probe is separate).
- **Shape:** lib only, large (19 modules; `unfold` 45K, `systemd` 45K, `archive` 64K, `pki` 36K, `product` 35K, `machine` 59K). Standalone workspace (resolver 3), extracted from `nornir`; published as `skidbladnir` (renamed from `nornir-airgap`).
- **Observed role in the workspace:** The `airgap` install framework that leaf consumers (`nornir`, `holger`, `tunnr`) pull to pack/move/install themselves without a release-cascade through `nornir`. The `matrix` feature injects the sibling `skidbladnir-probe` binary into KVM guests to assert install success.

Fill in (one line each — only the human knows these)

- Primary consumer(s):
- Why it exists (vs the obvious alternative):
- Hard constraint (`airgap` / no-C-deps / no-FFI / pure-Rust / etc.):
- Release cadence & independence (rides a parent? own repo someday?):
- Status (production / prototype / experimental / stubbed):
- Must-NOT-change invariant:
- Biggest known gap or TODO:
- Anything a newcomer would get wrong about it:

skidbladnir `ZnippyArchive` → real `znippy-core` (one canonical archive format)

skidbladnir hand-rolled a **JSON-indexed** archive (`src/archive.rs:354–765` , magic `ZNIPGAP1`) that imitates `znippy` and whose doc **falsely claims** it mirrors `znippy`'s Arrow-IPC layout. Replace it with the REAL format via a new lean `znippy-core` crate, so bundles are `znippy/DuckDB/Polars`-readable. ****** = not done.

Good news: clean extraction

- skidbladnir's `ZnippyArchive` is behind the **znippy** feature (OFF by default) — never shipped, **zero back-compat debt**. Default stays `TarZstd` .
- The real format's index code (`znippy/znippy-common/src/index.rs`) is **already lean**: deps = arrow 58 + blake3 + fst + serde — **no rayon, no znippy-cli, no compress engine**. So a `znippy-core` extraction is additive, not a churn.

The real format (the truth)

[blobs...][Arrow-IPC index][8-byte LE index_offset footer] (v0.6); v0.7 adds sub-indexes + a manifest + `ZNIPYIDX` magic + reserved modules (lookup/trie/sign). Index schema (`index.rs:39–70`): `relative_path.chunk_seq.fdata` `ZNIPPY_FORMAT_VERSION=3` in schema metadata; `check_format_version` already guards it.

Plan

Phase 1 — new `znippy-core` crate (znippy workspace, additive): re-home the lean index/footer surface: `compose_index_schema` , `build_metadata_batch` , `build_arrow_metadata_for_config` , `check_format_version` , `ManifestEntry` + reserved-module consts, `read_footer` / `write_footer` , `BlobMeta` / `ChunkMeta` . Deps: arrow/anyhow/serde/log/once_cell only. **No compress, no CLI, no rayon**. (Real `znippy-common` can later re-export from it.) **Phase 2 — skidbladnir consumes it**: add `znippy-core` dep behind the existing `znippy` feature; implement `RealZnippyArchive: Archive` that writes the REAL Arrow-IPC format (`pack` / `read_entry` O(1) seek/ `delta` / `apply`), delete the hand-rolled `ZnippyArchive` (`archive.rs:354–765`). Callers (`machine.rs:413` `ImageCache` , `fetch.rs` `verify_entries`) see no change (Archive trait seam). Fix the false doc claims. **Phase 3 (optional, holger)**: holger's parallel `.tar.zst` engine (`agent/lib/src/operations.rs:254–506`) can gain a `transfer_to_znippy()` export (holger already deps `znippy-compress`) — `tar.zst` stays for non-nordisk interop.

Gate (compat, not perf)

Round-trip: skidbladnir packs a tree → the real **znippy** CLI lists/extracts it byte-identically AND the Arrow index parses (DuckDB `SELECT * FROM <archive>`). O(1) selective read stays O(1). No back-compat to preserve (feature was off). Source: read-only scout 2026-06-28.

skidbladnir-probe — crate card

Reverse-engineered (from source — verify, don't trust the name)

- **One-line pitch:** The machine-injected introspection probe (plus a trivial test service) for skidbladnir's install matrix — a static-musl binary that reports install ground-truth as one JSON line.
- **Does:** Ships two std-only bins. `skidbladnir-probe` is dropped into each KVM matrix guest, introspects the just-installed service and prints one JSON line `{running?, pid, ports, health, logs}` for the host to assert on; it carries mini per-backend status knowledge (systemd-system / systemd-user / openrc) plus a `/proc` fallback (`--service/--backend/--os/--exec/--port/--health` args, hand-rolled JSON escaping, no serde). `skidbladnir-testsvc` is the trivial “product” the matrix installs as a real service — it binds a TCP port and answers `GET /health` with 200 so the probe has something to assert healthy.
- **Key deps:** none but `std` — deliberately dependency-free so it cross-compile cleanly and statically to `x86_64-unknown-linux-musl` (the parent skidbladnir crate's ring/zbus/rcgen deps cannot).
- **Shape:** bin only, 2 tiny binaries (`src/bin/skidbladnir-probe.rs`, `skidbladnir-testsvc.rs`). Own `[workspace]`, `publish = false`, release profile tuned for size (opt-level z, LTO, strip, panic=abort).
- **Observed role in the workspace:** Consumed by skidbladnir's `matrix` feature — one artifact injected into every guest (glibc Ubuntu/Debian/Fedora and musl Alpine) to turn the install-matrix cells into machine-checkable pass/fail rows.

Fill in (one line each — only the human knows these)

- Primary consumer(s):
- Why it exists (vs the obvious alternative):
- Hard constraint (airgap / no-C-deps / no-FFI / pure-Rust / etc.):
- Release cadence & independence (rides a parent? own repo someday?):
- Status (production / prototype / experimental / stubbed):
- Must-NOT-change invariant:
- Biggest known gap or TODO:
- Anything a newcomer would get wrong about it: