Project Tenzik — Viability & Build Plan (v0.1)

Executive Summary

Thesis: Tenzik = verifiable, federated edge compute where *events carry code* (WASM capsules) and sensitive workflows can be *proven* (zk receipts) without central control. The market pull is real: agent swarms, privacy-by-default, and interop-first architectures.

Viability (today): 8/10. Becomes 9/10 if we (a) drop Tent legacy, (b) make ZK optional/deferred, (c) lead with DevEx, (d) ship 1 killer demo.

MVP: A minimal federated node that runs 3–5 KB capsules, emits signed/hashed **execution receipts**, and (optionally) attaches a delayed zk-proof. Includes a *Quest Capsule* demo (invite/riddle → verified unlock) **or** a *Verifiable Webhook Router* demo (transform + receipt).

What to avoid: Tokenized economy in v1; mandatory ZK; protocol maximalism; over-scoped federation. Win trust with great tooling + a delightful demo.

Product & Service Offerings (v0 → v1)

v0 (MVP / Free OSS):

- **OSS Core:** Runtime + node + CLI, Apache-2.0.
- **Capsule Templates & Studio:** Local builder, size-budget guardrails, `wasm-opt -Oz` baked in.
- **Bridges Pack (starter):** Webhook adapter, ActivityPub MVP, MQTT MVP.
- **Receipt Explorer (lite):** Local web UI to browse/verify receipts.
- **Docs & Quickstarts:** Copy-pasteable examples; VS Code snippets.

v1 (Paid / Hosted add-ons):

- **Tenzik Verify:** Managed proof service (queues, GPU fleet), proof caching with SLAs.
- **Bridges Pro:** Hosted adapters (retry/queue/observability), curated transform library.
- **Enterprise add-ons:** SAML/OIDC, audit export (SIEM), support SLAs.
- **Optional Hosted Node:** One-click demo / small teams; self-host still first-class.
- **Compliance Packs:** PII redaction policies, data-retention helpers.

Non-Negotiables

- **User/Data Sovereignty:** Self-hostable node; portable data; export-first UX.
- **Security Model:** Sandboxed WASM, capability-based imports, resource ceilings.
- **Verifiability:** Receipts (hashes + signatures) everywhere; zk as a *pluggable backend*.
- **Interop:** Bridges before dogma (ActivityPub/ATProto for social, MQTT for IoT).
- **DevEx:** `tenzik deploy/run` in < 5 minutes; templates that actually work.

Architecture (v0 Core)

1) Runtime (Rust + Wasmtime):

- Load/validate/execute capsules with fuel metering & timeouts.
- Capability map (allowed imports ↔ declared capabilities in capsule metadata).
- Execution metrics → emit **ExecutionReceipt** (capsule_id, input/output commits, metrics, signer, nonce).

2) Federation (Minimal DAG):

- Local sled store of events/receipts.
- Gossip-lite: start with manual peer add; later libp2p/mdns.
- Event = {header (parents, ts, seq), payload (receipt|state|announce), sig}.

3) Proof Backends (Pluggable):

- **Phase A:** Mock ZK (dev mode) + delayed proof job queue.
- **Phase B:** Risc0/SP1 integration behind trait (`ProofBackend`).
- **Alt:** TEE attestation backend for low-latency "trust-but-verify" deployments.

4) Economic Layer (Defer):

- Start **fiat/credits** only; API keys + quotas. Token later *if ever*.

5) Capsules (3–5 KB target):

- Templates in AssemblyScript + TinyGo; Rust for heavier paths.
- Size budget enforcers: `wasm-opt -Oz`, stripped names, host-provided primitives.
- Component Model ready, but ship MVP with a flat ABI: `run(ptr,len) -> out_len`.

MVP Scope (12 weeks total)

Week 1–2 — Runtime Hardening

- Finish validation: required exports, import whitelist, size caps, fuel/epoch checks.
- CLI: `tenzik test <capsule.wasm> "input"` shows output + metrics.
- Receipt signer (ed25519) + blake3 commitments; local log.

Week 3–4 — Federation Seed

- Event DAG: tip selection, local store; `tenzik node start --peer <addr>`.

- **Publish/subscribe for Receipts and NodeAnnouncements.**

Gate A: Two nodes exchange a Receipt end-to-end.

Week 5–6 — Proofs (Optional/Deferred)

- Trait `ProofBackend` with mock impl; background job queues a delayed proof.
- **`tenzik receipt verify` accepts: (a) sig only, or (b) sig + zk.**

Gate B: One execution produces a delayed zk-proof that verifies.

Week 7–8 — Interop + Demo

- Adapter: Webhook Router (HTTP in → capsule → HTTP out) **or** ActivityPub bridge (note → capsule transform → note).
- Demo #1 ready: Quest Capsule or Verifiable Webhook Router.

Week 9–10 — DevEx + Docs

- VSCode snippet + `tenzik new capsule --template <name>`.
- Hosted demo site (static) + screencast + Quickstart.

Week 11–12 — Beta

- Invite 10 testers; tighten logs, metrics, and panic traps.
- Publish roadmap + Contribution Guide.

Killer Demo Options

A) Quest Capsule (pitch/marketing)

- 3–5 KB WASM: riddle → derives unlock code → emits receipt.
- Page gates deck on **receipt presence**, not just passphrase.
- Agents can auto-deliver capsule links; receipts are publicly checkable.

B) Verifiable Webhook Router (devtools)

- Transform Stripe/GitHub payloads via capsule; emit receipt with input/output commitments.
- Optional zk for PII-safe transforms.

C) IoT Threshold Filter (industrial)

- Edge node runs capsule: `(temp > 90°F)`; only alerts federated; receipts create audit trail.

Pick 1 for MVP.

WASM Capsule Footprint Reality Check

Target: 3 KB is feasible for tiny logic in AS/TinyGo when:

- No dynamic alloc; minimal stdlib; no panics; `-0z`; stripped names.
- Host provides helpers (hash, base64, datetime) to avoid in-capsule bloat.

If you keep missing 3 KB:

- Relax to **5–8 KB** for nontrivial transforms; or
- Use **Two-Stage Capsules**: 2–3 KB loader + host primitives; loader chains micro-ops; or
- **Alt runtimes:** eBPF (super tiny; great for filter/transform), QuickJS isolates (JS text, bigger but ubiquitous), or **TEE-only** for heavy code w/ attestation.

Rule of thumb: Ship with *host-provided opcodes* (e.g., `hash_commit`, `jwt_verify`, `json_path`) so business logic stays small.

Proof Strategy (Pragmatic)

- **Every run → signed Receipt:** Capsule + input/output commitments, metrics, node signature.
- **When needed → zk:** Queue proof generation; attach later (OK if minutes). Batch nightly to amortize.
- **Backends (pluggable):** `Mock` → `Risc0/SP1` → `TEE attestation` → hybrids.
- **Proof Cache:** Keyed by `(capsule_id, input_commitment)` with LRU + TTL; precompute hot paths; store URI (CID) pointer on receipt.
- **Invalidation:** Bump `capsule_rev` or `policy_id` to force cache miss; receipts include these in public inputs.

Receipt & Proof Contract (public inputs)

Receipt

```
{ capsule_id: blake3([wasm bytes]),
  input_commit: blake3(input),
  output_commit: blake3(output),
  exec: { gas, mem, ms },
  node_id: ed25519 pubkey,
  nonce,
  receipt_sig }
```

Proof

```
{ backend: "risc0|sp1|tee", version,
  public_inputs: { capsule_id, capsule_rev, input_commit, output_commit,
```

```
policy_id? },
  proof: bytes|cid,
  proof_sig? }
```

Interop Strategy (Don’t fight gravity)

- ActivityPub/ATProto for social, MQTT for IoT, Webhooks for SaaS.
- *Translate* external events into Tenzik Receipts; don’t replace their worlds.

Risk Register & Mitigations

1. **ZK perf/complexity** → Optional, deferred, batched; mock-first backend.
2. **Dev adoption** → Templates, `tenzik new/test/deploy`, great docs + demo.
3. **Security** → Strict import allowlist; fuel/time ceilings; no I/O by default.
4. **Regulatory (tokens)** → No token at MVP. Credits via fiat billing only.
5. **Scope creep** → One demo; bridges minimal; proofs optional.

Success Metrics

- T-4 weeks: Two nodes exchanging Receipts; CLI happy path.
- T-8 weeks: Demo app live; 5 external devs run it.
- T-12 weeks: 10 testers; 1 interop adapter; README stars/interest.

Decision Log (initial)

- Drop Tent legacy; **Tenzik Core** spec is new.
- ZK = pluggable/optional; receipts always-on.
- No token/economics in MVP; metering via quotas/credits.
- Capsule target = 3–5 KB with host op primitives.
- First demo TBD (leaning **Webhook Router** for B2B appeal).

Immediate Next Actions (This Week)

- ✅Finalize validation rules & tests in Runtime.
- ✅Define `ExecutionReceipt` and sign/verify flow.
- ✅Pick demo A or B; sketch the user journey.
- ✅Create capsule op primitives list (host functions) to keep sizes tiny.

Open Questions for Brian

1. **Demo pick:** Quest Capsule (viral) or Webhook Router (dev-painkiller)?
2. **Audience first:** Investors/PMs, dev-tools crowd, or IoT folks?
3. **Hosting posture:** Local-only to start, or a tiny hosted demo node too?
4. **License & brand:** MIT/Apache? Keep “Tenzik”? (It’s good.)
5. **Security envelope:** Any must-have capabilities or strict bans (e.g., no network I/O in capsules at all for MVP)?

Appendix — Capsule Size Budgeting

- **AssemblyScript:** great for tiny demos; easy to hit \~3–5 KB.
- **TinyGo:** slightly larger, but predictable; often \~4–10 KB for nontrivial logic.
- **Rust:** best safety/perf; smallest only with `no_std` /careful features (often 10 KB+), use for host/runtime.
- **Tricks:** `wasm-opt -Oz`; strip names; avoid fmt/alloc; precompute tables; host JSON helpers.

TL;DR: Ship receipts and a delightful demo; make ZK optional; win with DevEx and interop. Capsules stay tiny by leaning on host ops. We expand proofs & federation only when the use cases demand it.

✅Decisions Locked for MVP

- **Brand:** Keep **Tenzik**.
- **Demo: Verifiable Webhook Router** (B2B painkiller) for MVP; Quest Capsule as a marketing sidecar later.
- **Audience (first):** Dev-tools / B2B teams that need signed/traceable transforms (Stripe/GitHub/Slack payloads).
- **Hosting posture: Local-first** (CLI + self-host), with optional public tunnel for demos; tiny hosted node later if needed.
- **License: Apache-2.0** (enterprise-friendly + permissive).
- **Security envelope:** No network/file/clock access inside capsules for MVP. Strict import allowlist. Fuel + epoch timeouts.

🟩Sprint 1 (Weeks 1–2) — Runtime Hardening + Receipts

Objective: Execute capsules safely and emit signed **ExecutionReceipts**.

Work items (with DoD)

- **RT-003 — WASM validation logic**\ *DoD:* Fails capsules missing required exports (`run`, `memory`), exceeding size, or importing disallowed symbols.\ *Tests:* Unit tests for each failure mode + a passing "hello world".

- **RT-004 — Capability mapper / import allowlist**\ *DoD:* Declarative map from capsule `capabilities: []` → permitted imports. MVP allows only `env::abort`.\ *Tests:* Capsule with extra import is rejected.

- **RT-005 — Resource limits enforcement**\ *DoD:* Fuel metering + epoch timeouts; configurable `ResourceLimits`.\ *Tests:* Long-loop capsule is halted; metrics show cutoff.

- **ECON-002 — ExecutionReceipt (sign/verify)**\ *DoD:* `ExecutionReceipt { capsule_id, input_commit, output_commit, metrics, node_id, nonce, sig }` with ed25519 signer; verify path.\ *Tests:* Happy path + signature tamper fail + deterministic commits.

- **CLI-101 — `` happy path**\ *DoD:* `tenzik test build/capsule.wasm '{"x":1}' --metrics` prints output + metrics; non-UTF8 output handled.

Acceptance demo

Run `tenzik test` on a tiny transform capsule and show a printed **ExecutionReceipt** (JSON or pretty-table) plus a `tenzik receipt verify <file>` that returns OK.

🌐Sprint 2 (Weeks 3–4) — Minimal Federation (DAG) + Node Announce

Objective: Two nodes exchange receipts end-to-end.

Work items

- **FED-003 — Event DAG + local store**\ Keys: blake3 event IDs, parent links, tip selection. sled backing store.
- **FED-004 — NodeAnnouncement + handshake**\ Minimal peer add, announce capabilities, exchange tips.
- **CLI-102 — **``\ Boot a node, add peer, see receipts propagate.

Gate A

Two nodes publish/receive a receipt over the wire; local stores agree on DAG tips.

📵Sprint 3 (Weeks 5–6) — Pluggable Proof Backend (Optional/Deferred)

Objective: Stub ZK with a background queue; attach proof later.

Work items

- **PROOF-001 — `` trait + MockProof**
- **PROOF-002 — Proof job queue** (deferred generation; link proof CID/hash to receipt)
- **CLI-201 — **`` accepts sig-only or sig+zk.

Gate B

One execution produces a delayed proof that verifies with the mock backend.

🛄MVP Demo Spec — Verifiable Webhook Router

Flow: HTTP payload → (adapter) → optional **HMAC verify** (`X-Signature: sha256=...`) → capsule run → receipt emit → HTTP response (`{ output, receipt_id }`).

Components:

- **Adapter service** (Axum): `/webhook/:route` → selects capsule → encodes input → calls runtime → returns `{ output, receipt_id }`. Supports optional shared-secret HMAC verification.
- **Capsule template:** `json-transform` (e.g., rename field, filter keys, compute hash).
- **Receipt log:** file or sled; `GET /receipts/:id` returns JSON for verification.
- **Receipt Explorer (lite):** tiny static UI to paste an ID, fetch & pretty-render the receipt, and run local verification.

CLI UX:

```
# 1) Build & test capsule locally
cd capsules/templates/json-transform && npm i && npm run build
tenzik test build/capsule.wasm '{"email":"a@b.com"}' --metrics

# 2) Run adapter service (local node)
cargo run -p tenzik-adapter -- --port 8787 --hmac-secret $SECRET

# 3) Send a webhook with signature
SIG=$(printf '%s' '{"email":"a@b.com"}' | openssl dgst -sha256 -hmac "$SECRET" -
binary | xxd -p -c 256)
curl -X POST localhost:8787/webhook/sanitize \
  -H 'content-type: application/json' \
```

```
  -H "X-Signature: sha256=$SIG" \
  -d '{"email":"a@b.com"}'
# → { "output": {...}, "receipt_id": "..." }

# 4) Verify receipt
tenzik receipt verify <receipt_id>
```

Definition of Done:

- Deterministic input/output commitments (blake3).
- Signature verifies; receipt retrievable by ID; adapter returns 2xx with IDs.
- **If HMAC enabled:** bad/missing signature → 401; good → 2xx.
- Capsule size ≤ 5 KB (stretch: 3 KB) with `-Oz`; no forbidden imports.
- Receipt Explorer can fetch + verify a receipt offline.

🧳 Host Ops (to keep capsules tiny)

- `host.hash_commit(bytes) -> [32]`
- `host.json_path(bytes, path) -> bytes`
- `host.base64_encode(bytes) -> bytes`
- `host.hmac_sha256(key, bytes) -> [32]` *(for webhook sig verification paths) (No network/file/clock access in MVP.)*

♒ Ticket Bank (initial)

- RT-003, RT-004, RT-005 (Runtime hardening)
- ECON-002 (Receipts)
- **ECON-003 (Metering hook):** account/quota counters + CLI `tenzik credits top`.
- CLI-101/102/201 (CLI flows)
- FED-003/FED-004 (DAG + announce)
- PROOF-001/002 (backend + queue)
- **PROOF-003 (Proof cache + invalidation)**
- **SEC-004 (HMAC verify helper + threat model note)**
- DEMO-001 (Adapter) / DEMO-002 (json-transform capsule)
- **DEMO-003 (Receipt Explorer lite UI)**
- **BRIDGE-101 (ActivityPub MVP)**
- **IDENT-001 (DID/VC compatibility spike)**
- DOCS-001 (Quickstart) / DOCS-002 (Security model)

⚪Daily Rhythm (20h/week)

- **Mon:** plan sprint tasks (2h) → build (3h)
- **Wed:** "wild card" ideation (30m) → build (4h) → checkpoint
- **Fri:** test & demo (3h) → ship notes/readme updates (1h)

🚳Ready to Execute — Command Cheatsheet

```
# Build workspace
yarn --version >/dev/null 2>&1 || echo "(npm ok)"
cargo build --workspace

# Template → build capsule
cd capsules/templates/hello-world && npm i && npm run build

# Validate + run capsule locally
tenzik test build/capsule.wasm '{"x":1}' --metrics

# Start node A (terminal 1)
TENZIK_DB=.data/a tenzik node start --peer none

# Start node B (terminal 2)
TENZIK_DB=.data/b tenzik node start --peer 127.0.0.1:9000

# Verify a receipt
tenzik receipt verify path/to/receipt.json
```

🏧Comms & Positioning (for when we demo)

- One-liner: **“Tenzik runs tiny capsules at the edge and gives you cryptographic receipts for every transform.”**
- Proof stance: **Receipts now, ZK when it matters.**
- Security stance: **No I/O in capsules; capability-based host ops; strict limits.**

Next actionable step for you: run the Cheatsheet’s first three commands and drop me the console output. I’ll triage immediately and we’ll iterate fast.