

Architecture & Design Philosophy

Product vision, high level design and reference architecture.

Velio — Architecture & Design Philosophy

Product vision, high level design and reference architecture

Status: in development · **Platform version:** 3.2.1

Foreword

Velio grew out of my conviction that professional communication does not have to be complicated.

In my work I saw how implementations kept getting more complex, while what many organisations actually needed was simple: reliable communication, without being dependent on a single vendor or an unnecessarily complicated platform.

That is why I deliberately designed Velio differently. Not with as many features as possible, but with a clear architecture, open standards and reproducible deployments as the starting point. I believe software should remain understandable — for administrators, partners and developers alike.

You see that philosophy reflected in every design choice. Setting up a new Velio environment should not be a project of days or weeks, but something that can be operational within hours. The infrastructure is reproducible, the configuration largely automated, and a complete environment can be rebuilt without complex manual steps. As a result, implementations stay predictable, upgrades remain manageable, and recovery after an incident is fast and reliable.

Velio is still very much in development. This document describes the current architecture, the design choices made, and the direction in which the platform is evolving.

How to read this document

The document opens with the vision and design principles, followed by two parts for two kinds of readers:

- **Part A — Product description.** For a broad audience: what Velio is, who it is for, and why you would choose it. No technical background required.

- **Part B — High Level Design.** For a technical audience (architects, administrators, a customer already running Nextcloud): reference architecture, security, interfaces, deployment and the backup/restore strategy, with schematic diagrams in Mermaid.

Status markers throughout:  available ·  in development ·  designed (not yet built).

1. Why does Velio exist?

Modern communication platforms keep growing in scope, and with that often become more complex to implement and operate.

Velio deliberately chooses simplicity.

The platform is built around open standards and gives organisations the freedom to make their own choices: where the platform runs, how identities are managed, and which systems it integrates with.

Velio therefore integrates excellently with platforms such as Nextcloud, but remains fully usable on its own. Integrations are a choice, not a requirement.

2. Design principles

Understandable software. Every architectural choice is weighed for simplicity, maintainability and predictability.

Open standards. SIP, WebRTC, OIDC and PostgreSQL form the foundation.

Reproducible infrastructure. New environments are rolled out in an automated way so that every installation has the same quality.

Deployability. A new customer environment should be operational within hours. Not because speed is a goal in itself, but because a reproducible deployment leads to fewer errors, simpler operations and faster recovery.

Single tenant. Every customer gets a fully dedicated environment.

The backend is the source of truth. Business logic does not belong in the PBX.

Security by design. Security is part of the design, not an afterthought.

Part A — Product description

A1. What is Velio?

Velio is a **communication platform**. Whether you simply want a **reception / phone system**, want to roll out a full **contact center for your service desk**, or **both** — Velio covers that range with one integrated product.

Velio's strength lies in simplicity, fast rollout, and a platform that remains understandable and manageable by a small team. It is deliberately not built to serve contact centers with hundreds of agents — that choice keeps it light, fast and easy to oversee.

A2. Who it is for

- **Reception & operator** — answering, transferring, consulting, parking calls.
- **Customer contact & service desk** — queues, menus, opening hours, callbacks.
- **Omnichannel contact center** — voice and webchat in one workflow and one set of reports.

A3. Why Velio

- **Simplicity.** One coherent product instead of a sprawl of modules and SKUs.
- **Fast restore & rollout.** The entire environment is rolled out reproducibly; after an incident you recover by redeploying and restoring your data, rather than maintaining an expensive standby environment. This is a core promise — see B7.
- **Open standards.** SIP, WebRTC, OIDC — no vendor lock-in, no closed protocol.
- **Fully browser-based.** Agents call through a WebRTC softphone; nothing to install.
- **Omnichannel.** Voice and webchat share queue, presence and reporting.
- **Nextcloud-friendly.** Identity, directory and presence via Nextcloud/OIDC; voice to Talk colleagues through Velio's own Talk bridge ✔.
- **Privacy & security by design.** NL hosting, encryption, MFA, immutable audit trail.

A4. Functional overview

Domain	Functionality
Telephony ✓	Inbound routing, queues, menus (IVR), opening hours/holidays/ad-hoc, blind & consult transfer, parking, hold with music-on-hold
Agent workspace (/advisor) ✓	WebRTC softphone, presence, wrap-up (call coding), after-call work, directory & search
Administration (/manage) ✓	Teams, phone numbers, contact centers (Lite Builder wizard), agents, security, resources
Webchat ✓	Embeddable widget, triage, conversation shared with voice, anti-abuse
Reporting ✓	Realtime dashboard + public wallboard, historical reporting, channel-aware (voice/chat)
Identity & access ✓	OIDC (Nextcloud), RBAC (admin/supervisor/agent), MFA with recovery codes
Quality monitoring & recordings ⚙	Call recordings and QM as a managed, privacy-sensitive RBAC object

A5. Licensing direction

Simple and sellable: **named seats** (licensed users/agents) plus a limit on the number of **concurrent inbound calls**. The limit counts at call level, not at PBX-internal call legs or channels.

A6. Positioning

Velio differentiates itself not on feature volume but on **simplicity and fast recovery/rollout times**, on **open standards**, and on seamless alignment with the **Nextcloud ecosystem** — hosted in the Netherlands/EU. It is a platform for organisations that want professional customer contact without the complexity and lock-in of the large suites.

Part B — High Level Design

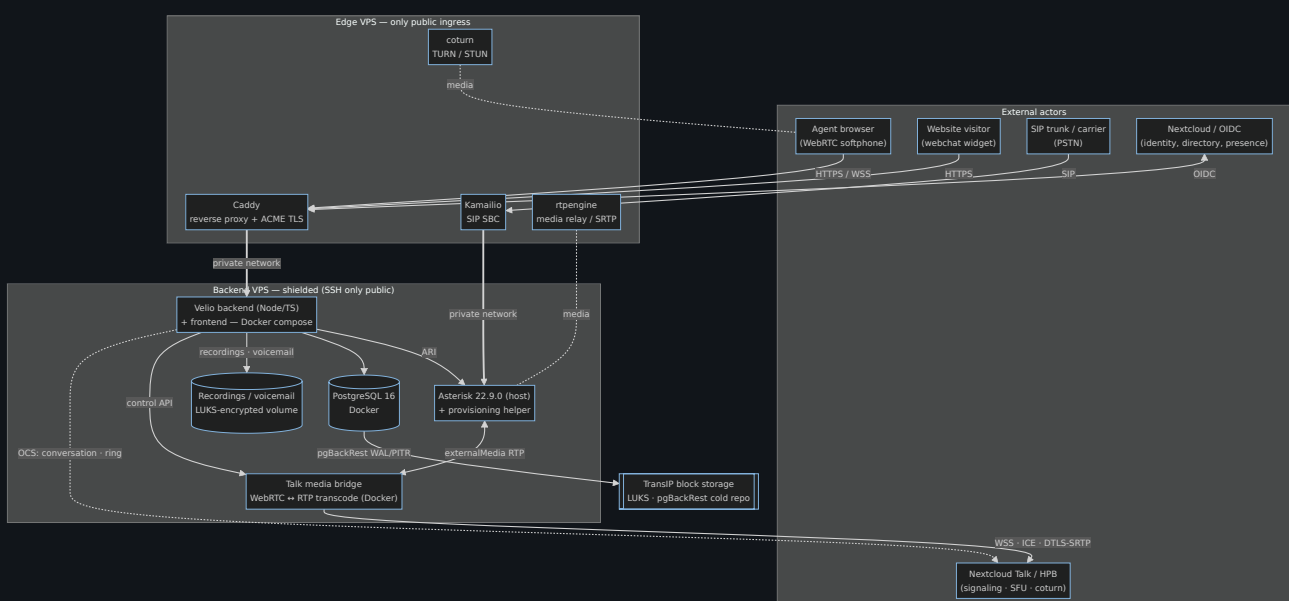
B1. Architectural principles

The design principles from chapter 2 translate technically into the following choices:

- **The backend is the source of truth.** Call state, agent state, routing, queueing, business rules, audit and events live in the Velio core. The PBX provides signalling and media, not the business logic.
- **Asterisk-first, behind an adapter boundary.** Asterisk is the fixed execution engine; a PBX adapter (`PbxProvider`) translates a **canonical telephony model** to/from ARI. The core does not speak vendor concepts.
- **Event-driven & realtime.** PBX → adapter → core → frontend, with WebSocket/server-push to the UI; polling only as an explicit fallback.
- **Open standards for media.** WebRTC, SIP, RTP/SRTP, SDP, ICE/STUN/TURN.
- **Single-tenant per customer.** Each customer runs its own, isolated Velio environment.
- **Predictability over complexity.** No active HA clustering; the environment is built identically and repeatably via Ansible. That results in reproducible deployments instead of complex HA solutions — recovery is via redeploy + data restore (B7), not failover.

B2. Reference architecture (2 VPS)

A customer environment consists of two VPSes, connected over a **private network** (`vpriv0`). The **edge** is the only public ingress; the **backend** (app, data, telephony) is shielded and only reachable publicly over SSH.



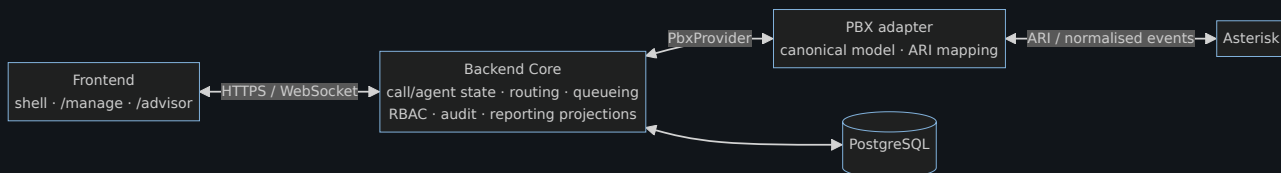
Edge VPS (firewall: 80/443 public, SIP 5080/udp + media ranges, SSH restricted):

- **Caddy** — reverse proxy with automatic TLS (ACME) for the web UI, API and WebSocket.
- **Kamailio** — SIP SBC for carrier trunks (the outward signalling boundary).
- **rtengine** — media relay / SRTP termination.
- **coturn** — TURN/STUN for WebRTC NAT traversal from the agent browser.

Backend VPS (firewall: only SSH public, the rest on loopback/bridge/private):

- **Velio backend + frontend** — Node/TS runtime and static frontend in Docker compose.
- **PostgreSQL 16** — in Docker; the source of truth for product and operational data.
- **Asterisk 22.9.0** — on the host (pinned baseline), with a restricted provisioning helper.
- **Talk media bridge** — a separate Docker component (WebRTC ↔ RTP transcode) that couples an Asterisk channel to a Nextcloud Talk conversation; talks to Asterisk over loopback and only **outbound** (WSS + ICE/TURN) to the Nextcloud HPB. Always deployed, active only when the Talk bridge is switched on (B4).
- **Disk encryption (LUKS2)** on the dedicated data volumes (pgBackRest repo and recordings/voicemail); the key is the per-customer Ansible vault key, DR-recoverable (B5).
- **Healthcheck** — `/health`, pjsip transports, ARI status, edge reachability.

B3. Logical layers



The frontend is thin and PBX-agnostic (a browser shell with separate `/manage` and `/advisor` modules). The core owns the domain; the adapter isolates all Asterisk-specific execution. Asterisk provisioning (numbers, agent endpoints) is a **synchronisation path**, not an availability condition: if Asterisk is temporarily down, the management layer remains usable.

B4. Interfaces

Interface	Protocol / technology	Status
Carrier trunks	SIP via Kamailio SBC	✓
Agent media	WebRTC (DTLS-SRTP, ICE, coturn/rtpengine)	✓
Identity & directory	OIDC against Nextcloud; presence	✓
Webchat	Embeddable widget over HTTPS/WSS	✓
Voice to Talk colleagues	Velio's own Talk bridge (headless WebRTC); official SIP bridge (Enterprise) as a reserved path	✓
Reporting / CDR export	Public Velio Core API	✓
Backup storage	pgBackRest → TransIP block storage (LUKS)	✓
Provisioning / fleet management	Partner Portal (Ansible, TransIP API)	⦿

Constraint — E2EE calls. Transferring from Velio to a Nextcloud Talk participant requires the conversation involved to **not be end-to-end encrypted**: a gateway that transcodes audio (to Asterisk/PSTN) must be able to decrypt the media, which is inherently incompatible with E2EE. This is not a Velio limitation — it applies to **any** Talk↔telephony link, including the official **SIP bridge** (E2EE calls do not support SIP dial-in/-out) and every PSTN party (a phone is never an E2EE participant). E2EE **per conversation** (opt-in) coexists fine: the transfer conversation is simply non-E2EE, while internal Talk↔Talk calls can still be E2EE. Only **globally enforced E2EE** is an **unsupported configuration** for telephony integration.

B5. Security architecture

Network

- The **edge is the only public ingress**; the backend VPS is shielded (only SSH public), internal traffic runs over the private network.
- **nftables** with default-deny on both hosts; **fail2ban**, automatic security updates and a pinned timezone via the `common` hardening.
- **Caddy** handles end-to-end TLS with automatic certificates.

Identity & access

- **OIDC** (Nextcloud) for login; **RBAC** enforced at both API and UI level with roles `admin` / `supervisor` / `agent` and team scope.
- **Single-session policy** per account (a second login takes over the session in a controlled way).
- **MFA** for local accounts: TOTP with single-use **recovery codes**; disabling MFA requires a **step-up** with a current code. Externally managed (OIDC) accounts are out of scope — their security runs through the IdP.

Data & secrets

- **Passwords** hash-only with **Argon2id** (never readable back).
- **Technical secrets** (SIP/provisioning, MFA secrets) **AES-256-GCM encrypted-at-rest**, with a deployment key **outside** the database, in a shielded secret-store boundary — not in broad product config, not in the UI, not in logs.
- **Parameterised queries** everywhere; an **immutable audit trail** for management changes.

Storage

- **Disk encryption by default (LUKS2)** on all dedicated data volumes — the pgBackRest repo and the recordings/voicemail volume. The key is the **per-customer Ansible vault key** (`/etc/velio/disk.key`, root-only); escrowed by construction, hence **DR-recoverable**. TransIP's off-site copy is block-level and therefore copies the LUKS ciphertext — no unencrypted intermediate.

Webchat anti-abuse

- Proof-of-work pre-queue gating, rate limits, concurrency caps, idle/abandon timeouts, a temporary IP block ("mark as spam") and a URL content gate. Deliberately **WAF-independent**; a customer can put Cloudflare in front if desired.

Media

- WebRTC with **DTLS-SRTP**, ICE and TURN; G.711 (PCMU/PCMA) as the interoperable baseline.

B6. Deployment & operations

Every customer environment is rolled out in an identical way. This makes implementations predictable, upgrades manageable, and incidents easier to resolve.

- **Ansible, 2-VPS, idempotent.** `edge.yml` and `backend.yml` (shared via `site.yml`); an intake wizard (`velio-deploy`) generates the inventory and runs the apply.
- **Docker** for backend + frontend (compose) and PostgreSQL; **Asterisk** on the host (pinned 22.9.0). Releases are shipped as rsynced tar images to the backend VPS.

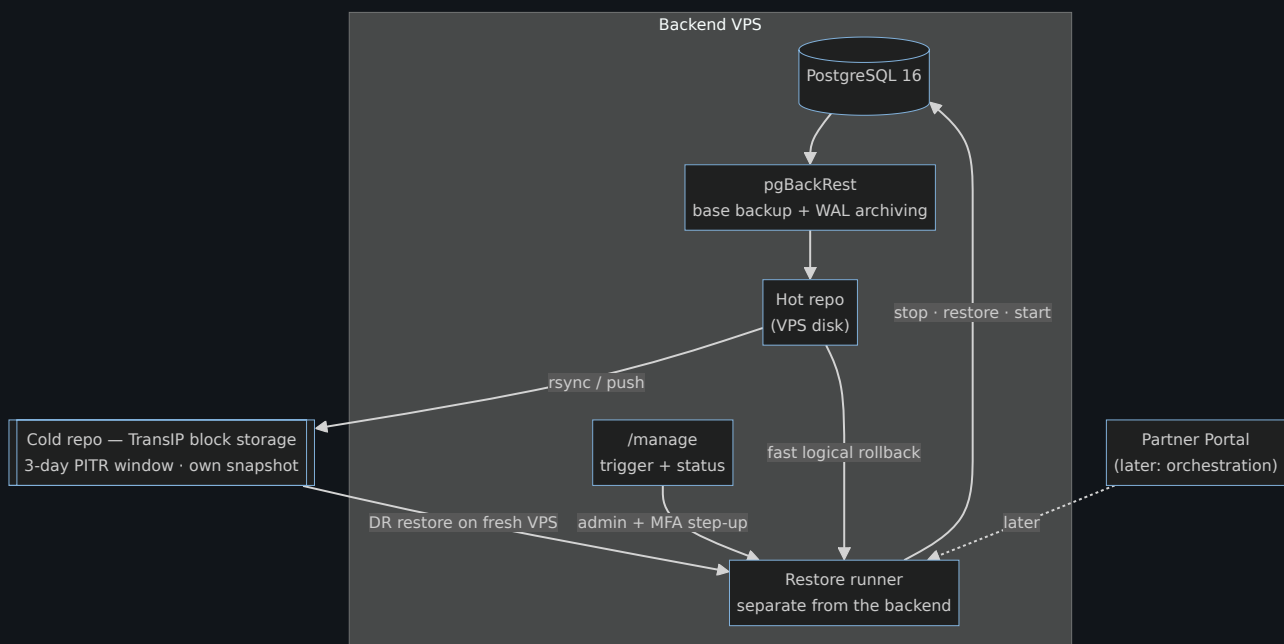
- **Migrations run at boot** of the backend — no manual database steps.
- **Healthcheck role** monitors `/health`, pjsip transports, ARI status and edge reachability.
- **Private network** (`vpriv0`, static IP, no gateway) between edge and backend.

Lead time, realistically. The Velio environment itself is usually operational within hours. The biggest variable lies outside the platform: **porting or activating phone numbers** with the carrier is an external dependency that can determine the eventual lead time.

Direction. The ultimate goal is a fully automated deployment, where infrastructure is provisioned via the **TransIP API** and the complete environment can be rolled out without manual configuration. In time, deployment, lifecycle management and tenant monitoring will be centralised in the **Partner Portal**; every Velio installation publishes a standardised `/health` endpoint for that purpose.

B7. Backup, restore & continuity

The "fast restore" promise from Part A, technically underpinned. Ansible reconstructs **code + config**; this layer covers the **state**.



Engine. `pgBackRest` (MIT license) with **WAL archiving** → **point-in-time recovery**, down to the second before an incident. Runs as a separate tool alongside PostgreSQL-in-Docker.

Two tiers.

- **Hot** — base + WAL on the VPS disk; fast PITR for logical mistakes ("deleted by accident").

- **Cold — TransIP block storage**, decoupled from compute (VPS gone $\neq$ data gone), optionally in another datacenter, with its own built-in snapshot as an extra safety net.

Retention. A PITR window of **3 days** — recent enough for DR, generous enough to recover a "discovered later" mistake over a weekend; going further back adds nothing. A single config line, adjustable later.

Restore architecture. The backend talks to the very database being restored and therefore cannot perform its own restore. **Trigger and execution are separated:** `/manage` is the **trigger + status** (admin only, with **MFA step-up** and explicit destructive confirmation), a **restore runner** separate from the backend does the work (maintenance $\rightarrow$ PostgreSQL stop $\rightarrow$ `pgbackrest restore` $\rightarrow$ start $\rightarrow$ backend back up). The runner is **source-agnostic** (hot-local for rollback, cold-external for DR on a fresh VPS). The Partner Portal later takes over the same runner — no wasted work.

Critical coupling. The data repo and the **AES encryption key** (Ansible vault) are bound together: a restore against a lost/different key makes MFA and provider secrets unreadable (a silent, half restore). The restore routine puts **both** back and versions them together.

Out of scope. Call recordings are not backed up — they fall under data retention and disappear with the VPS. Only **QM material** is later retained durably and separately.

Compliance retention of CDRs/audit belongs in the application (reporting/archive layer), not in the backup window — that keeps the window short.

Status: backup, WAL/PITR, the LUKS repo on block storage and the `/manage` observability are **built and in production** ✅ (since release 2.0.5, DR run exercised). The fully automated, Partner Portal-orchestrated restore-to-throwaway remains the outstanding step 🔄.

B8. Deliberate non-goals

- **No active HA clustering** — predictability + fast restore instead of a continuously running standby environment (less maintenance, less hidden state).
- **No mega contact centers** (hundreds of agents) — that scale calls for a different kind of platform; Velio deliberately chooses simplicity and clarity.
- **No generic PBX shell** — Asterisk-first, with a sharp adapter boundary.
- **No proprietary media SDKs** — open standards only.

Appendix — sources

- `docs/architectuur/ARCHITECTUUR.md` — extensive internal architecture (domain model, PBX adapter, media, telephony model).
- `docs/releases/` — release notes per version.
- `infra/ansible/` — the 2-VPS deployment (roles `edge_*`, `asterisk_host`, `postgres`, `velio_app`, `healthcheck`).