Voltade

Volty: one agent, many businesses

The architecture behind Voltade's AI helpdesk builder, a managed, multi-tenant platform.

How Volty runs a dedicated business agent for every organization on one shared deployment: isolated by design, bounded by typed operations, auditable to the row, and routed to a tested model per task under zero data retention.

V0.3 | 4 November 2026
voltade.com

Contents

Read this if...

Executive summary

1 One deploy, many businesses

2 One agent per org, shaped, not a fleet of bots

3 The wake: how the agent actually runs

4 The operation registry: one capability, five surfaces

5 Automations and mini-apps: deterministic rails beside judgment

6 The drive: versioned knowledge the agent reads and writes

7 Tables and tasks: structured data, and work you can watch

8 Channels and platform lines

9 Inboxes: visibility and privacy by design

10 Models: the gateway, the choice per task, and zero retention

11 Authentication and per-action authority

12 How Volty is run

What this architecture buys you

Version 0.3, first published 18 June 2026, updated 4 November 2026.

The web edition is at voltade.com/resources/guides/volty-platform-architecture.

Read this if...

- You're evaluating Volty and want to know what's actually under the product, not just what the homepage claims.
- You need to know which AI models see your data, how they were chosen, and whether the providers keep it.
- Your team is asking the fair questions: where does our data live, who can see it, what stops the agent doing the wrong thing, and what happens when it does.
- You've used ChatGPT or Claude at work and want a precise answer to "why is a platform different from a chat window?"
- You run engineering and someone will ask you to sign off on whatever the business picks.

Executive summary

Volty is an AI helpdesk builder, run as a managed, pooled, multi-tenant platform. Every organization that signs up gets one dedicated business agent. It answers on WhatsApp, email, web chat, Messenger and Instagram, remembers its customers, learns from the team, and acts in the tools the business already uses. There is no per-tenant build and no per-tenant deployment: every organization runs on one shared codebase, one database, and one set of infrastructure, isolated from each other by design.

That single sentence hides four architectural commitments that took two years and 100+ deployed agents to settle on.

First, isolation is enforced at the data layer, not the application layer. Every row in the database is tagged with the organization that owns it, and the database itself refuses to return rows that don't belong to the organization making the request. One organization cannot see another's data even if the application code has a bug.

Second, every capability the agent has is a typed operation in our codebase. The agent cannot take an action that isn't a declared operation with a defined role and risk level. There is no hidden orchestration layer underneath, and no way for a prompt to invent a new power.

Third, the agent is supervised by default. Volty does nothing in an inbox until its owner chooses Off, Draft (Volty writes, a person sends) or Live, and the organization moves it to live on its own terms. Every customer-affecting action is recorded and bounded, most can be reverted, and the ones that cannot always wait for a person.

Fourth, each task runs on a model chosen and tested for it. A customer reply, a scanned invoice and a voice note go to different models, each with backups from a second AI company, all through one gateway under zero data retention, with the exceptions named in Section 10.

The result is the difference between a clever assistant you talk to and a colleague your operations team will actually let near a paying customer. The rest of this document is how it works.

1 One deploy, many businesses

Most "AI platforms" are really one instance per customer, stood up by hand. That model is slow to provision, expensive to run, and it drifts: every customer ends up on a slightly different version, and a fix in one place doesn't reach the others.

Volty takes the opposite stance: **one deployment, one main branch, one database, organization = tenant.** A new organization is created with a single record. There is no environment to spin up, no instance to configure, no migration to schedule. Everyone is always on the same version, and an improvement ships to every business at once.

The hard part of pooling many businesses into one system is keeping them apart. Volty does this with **row-level security (RLS)** in Postgres. Every table that holds tenant data carries an organization key, and an RLS policy on the table only returns rows matching the organization set for the current request. The organization is set per-transaction, never globally, so it cannot leak between requests that share a connection. Isolation is a property of the database, enforced on every read and write, not a filter the application has to remember to apply.

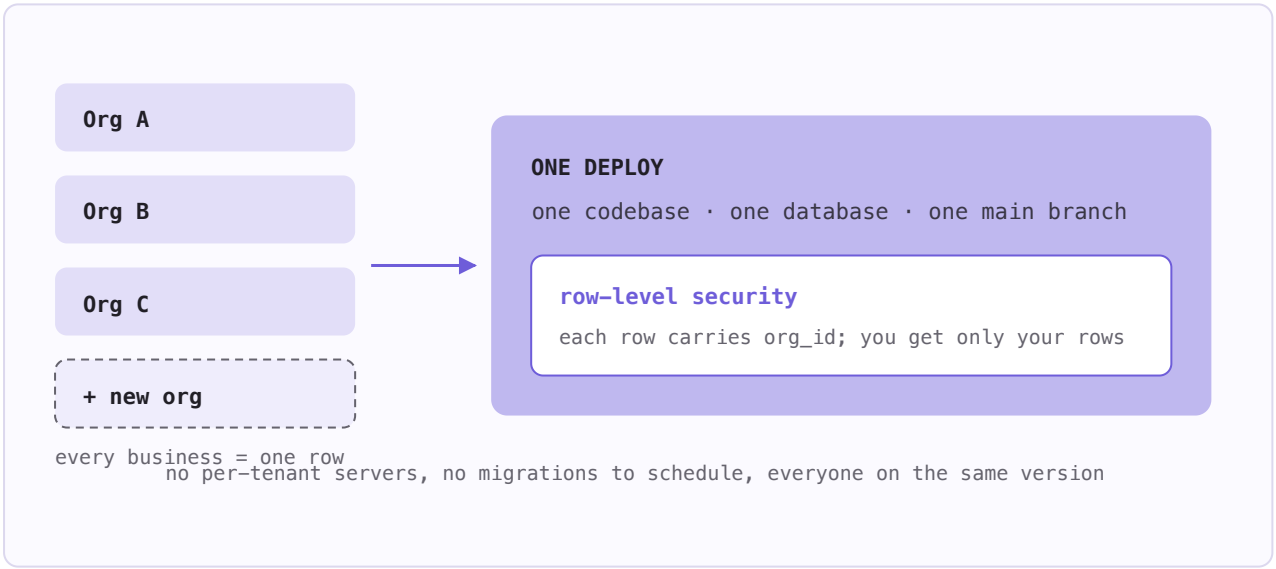


Figure 1 Every business is a row in one shared database, walled off by row-level security. A new tenant is an INSERT, not a deployment.

The same wall has a second floor inside it: an organization admin can see the organization's shared work, but cannot read into an individual staff member's private inbox. Since November 2026 every message row follows the visibility of its conversation in the database itself, not only in the application. Trust boundaries run between businesses **and** between roles within a business.

2 One agent per org, shaped, not a fleet of bots

A common instinct is to let a business build many bots, one per task. We tried that and removed it. Volty gives each organization **exactly one agent**, platform-provisioned and named "Volty," and lets the business shape its *behavior* rather than assemble a fleet.

Behavior is shaped by **placed instructions**: prose fields that live on the surface each one governs. There are three: organization-wide, team, and per-inbox. Each is addressed by where it sits rather than by a filename, so a rule about how one inbox answers is written on that inbox. Visibility follows the same wall as everything else: a member reads the organization's fields and the inbox fields of inboxes they can already see.

Two deliberate constraints sit on top. The fields are capped (12,000 characters for the organization, 8,000 for a team, 4,000 for an inbox). And they are read-only in the interface: the only way to change one is to ask Volty, which means every behavior change arrives through the same reviewed, recorded path as any other action rather than through a textarea. The platform remains the authority on identity, policy, and safety, and a placed field cannot override it.

Beside the placed fields sits one layer Volty writes itself: a short, capped list of **learned rules**, distilled from the team's corrections. A nightly review per organization reads the day's conversations and proposes changes to rules, knowledge and instructions, and an instruction change still waits for an admin.

This mirrors how the leading agent platforms actually work: one configured agent with layered personality, not a swarm. It keeps the mental model simple for the business, keeps the safety story coherent, and keeps the door open, because the underlying tables still allow multiple agents, neutered at the surface, so future collaboration roles need no migration. Identity and model are platform-owned. A custom reply name comes with every paid plan and removing the Volty branding from the web chat widget is a higher-tier entitlement, while the customer-facing disclosure line stays editable by an admin at every tier.

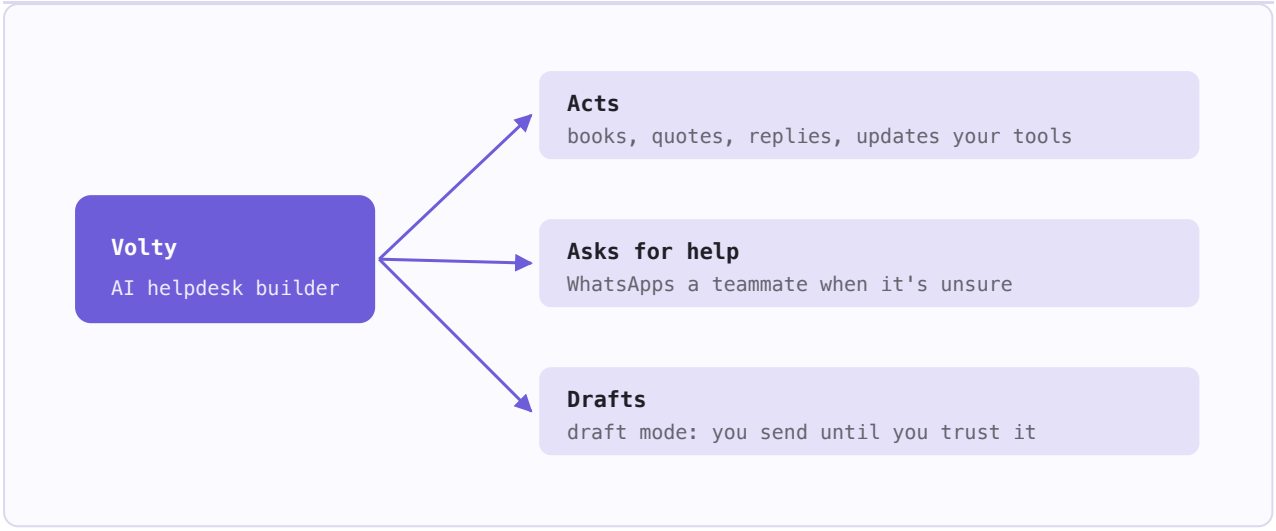


Figure 2 Volty acts on its own, asks a human when it is unsure (a WhatsApp DM to a teammate), and starts in draft mode, writing until you trust it to send on its own.

3 The wake: how the agent actually runs

Volty's agent is not a server waiting in a loop. It runs in **wakes**, bounded executions triggered by something that happened: an inbound message, a staff note, an approval decision, a scheduled follow-up, the nightly review. A customer-facing wake is scoped to one conversation in one organization; review and automation wakes are scoped to the organization. Every wake runs under that organization's data boundary.

A few properties make this safe at platform scale:

- **One wake per conversation at a time.** A lease prevents two wakes racing on the same conversation, so the agent never talks over itself. If a worker crashes mid-wake, the lease expires and the work is reclaimed rather than stranded.
- **Bursts are debounced.** A flurry of messages collapses into one wake rather than a storm of them.
- **Execution is durable.** Each step of a wake is journaled to Postgres, so a restart mid-flight resumes from the last save-point instead of losing work or repeating it.
- **The agent stays quiet when a human is handling it.** A staff reply starts a back-off that extends with each reply, so the agent observes rather than interrupts. Pressing Stop aborts the reply at once and hands the conversation to whoever pressed it.
- **One case, one session.** When a conversation is closed, the next message starts a fresh case, so an old thread's context does not leak into a new request.

The turn loop itself is built on a dedicated agent engine. Every model call is routed through the gateway (Section 10) under the organization's own key, so usage and cost stay tenant-scoped. The start of each session's prompt is frozen and its tools are kept in a stable order, so providers can cache it; Volty records the cache hit rate and alerts when it falls below 80%.

4 The operation registry: one capability, five surfaces

This is the heart of what makes Volty bounded. **Every capability is an operation**, a single typed declaration in our codebase. One declaration produces up to five surfaces from the same definition: a command the agent can run in its shell, a native tool the model can call, a remote CLI verb, an HTTP endpoint, and an automation SDK call. There are about 470 operations today.

Two things ride on every operation: a **role** (admin, staff, or contact) and a **risk class** (safe, customer-affecting, external, or budget-affecting). Every operation also declares the permission it needs (view, edit or manage over a resource). Those gates apply on every surface, every time. A turn a staff member starts runs with that person's own role, re-checked every turn. A turn a customer message or a schedule starts runs at staff rank, with admin-only operations off its surface and its data bounded by the inboxes the agent belongs to. A customer-affecting operation is checked for authority before it runs, whether it's the agent, a staff member, or an automation calling it. Roles are monotone: a caller satisfies an operation's floor when their role ranks at or above it.

A fixed floor of actions always waits for an owner or admin, whatever the settings: anything that spends money, sends to many people at once, or cannot be reverted, such as deleting contacts, merging them, or disconnecting a channel.

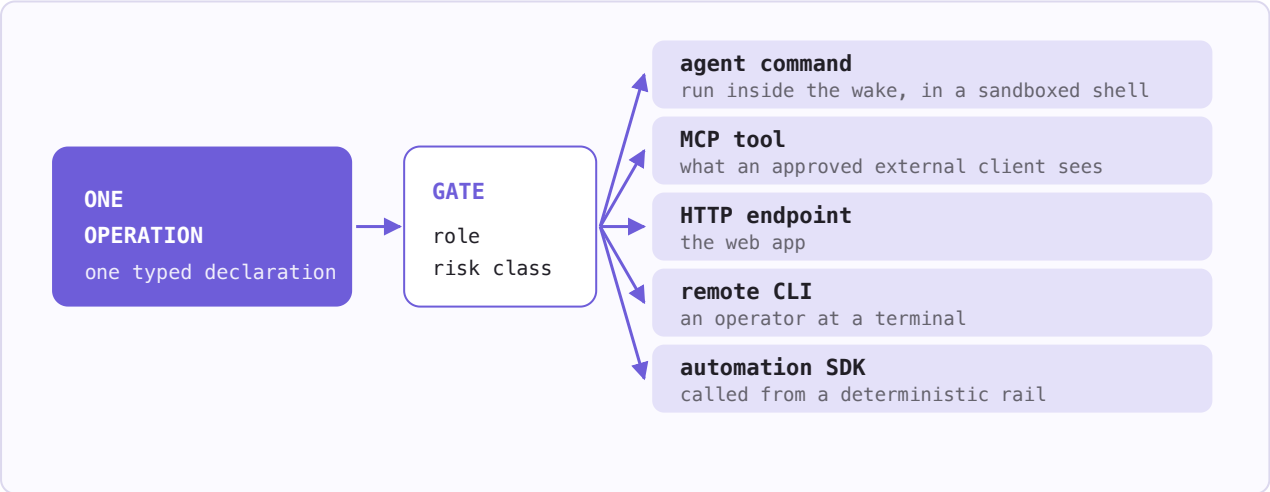


Figure 3 One typed declaration produces every way a capability can be called. The same role and risk gates sit in front of all five, so there is no surface where the rules are looser.

Declarations follow the shape of an MCP tool, so the catalog maps cleanly onto the protocol. Outbound calls are governed too: an organization can connect its own MCP servers, and a call out to one is classed as an external-risk operation and gated like any other, rather than becoming a side door around the registry.

The payoff: there is no second, hidden way for the agent to act. If a power exists, it's an operation, and it's governed. If it's not an operation, the agent can't do it.

5 Automations and mini-apps: deterministic rails beside judgment

SLA timers, scheduled digests, webhooks, and idempotent notifications belong in a fixed code path, not in an agent loop that re-decides them every time. Volty splits the two: the agent handles judgment, **automations** handle determinism.

An automation is per-organization code deployed to a sandboxed runtime, an isolate whose only network route is back through the platform, with the same journal-replay durability the wake loop uses, so a restart resumes rather than re-fires. The unusual part is who writes it. **Staff describe the rail in plain language and the agent authors it**, then staff see the rendered result and can run, pause, version, or roll it back. Staff can also paste code when they create one, but deployed code is shown read-only and never edited in place, which is what keeps the deployed rail and its description from drifting apart.

Two rules keep this from becoming an escape hatch around Section 4:

- **The approval gate is a dry-run.** An AI reviewer reads every filing for bugs and risk, then an operator approves against a simulated run on real data showing the concrete effect, with nothing sent. A rail classed safe that an org admin files goes live on its own. Anything the agent authored, and anything with an irreversible or customer-facing effect, waits for an org admin.
- **The judgment leaf is explicit.** An automation can hand one sub-decision to the model (classify this intent, draft this line) and then continue deterministically. Model use is pinned to named steps; the orchestration around them stays model-free.

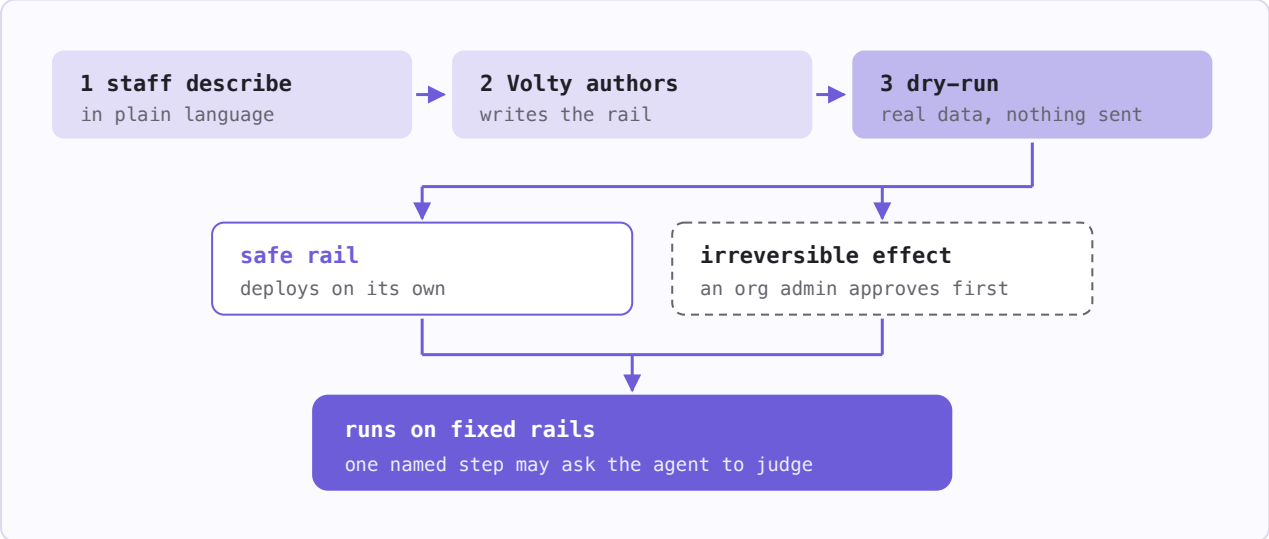


Figure 4 Staff describe a rail, Volty writes it, and what an operator approves against is a simulated run on real data. Safe rails deploy themselves; anything irreversible waits for an admin.

Mini-apps run on the same substrate: small internal tools an organization needs, authored the same way and governed by the same gates. A business can grow its own operational surface without a per-tenant deployment, and under the same approval record.

6 The drive: versioned knowledge the agent reads and writes

Volty's agent doesn't keep its knowledge in a prompt or a black-box vector store. It keeps it in a **drive**, a versioned file store that both humans and the agent share. Borrowing git's data model (content-addressed blobs, a working tree, immutable snapshots) without git itself, the drive gives every change a history.

The drive is partitioned into scopes: the organization's knowledge base and skills (the agent reads them), a per-contact memory (the agent writes what it learns about each customer), what the agent knows about each staff member, the agent's own skills, a private volume per staff member for their own uploads, a volume per automation or mini-app holding the versioned deploy artifacts for the code above, and smaller scopes for conversations, templates and broadcasts. Instructions are deliberately not in anything a wake reads as knowledge. They live in the placed fields in Section 2, which keeps the knowledge base a knowledge base and nothing else. Small text lives inline in the database. Everything larger lives in object storage, written ahead of the database record so a dangling reference is structurally impossible.

At the start of a wake the agent's workspace is materialised. At the end, a snapshot captures what changed, the agent's "commit" for that turn. That snapshot history is not a bolt-on audit log: it *is* the record of how the agent's understanding of your business evolved, who changed what, and when. This is also how the agent improves. A team's correction lands where it belongs: a learned rule, a fact in the knowledge base, or a note on the contact. The nightly review catches what the day missed. The agent gets better week on week without a retraining cycle, and no customer data is ever used to train a model.

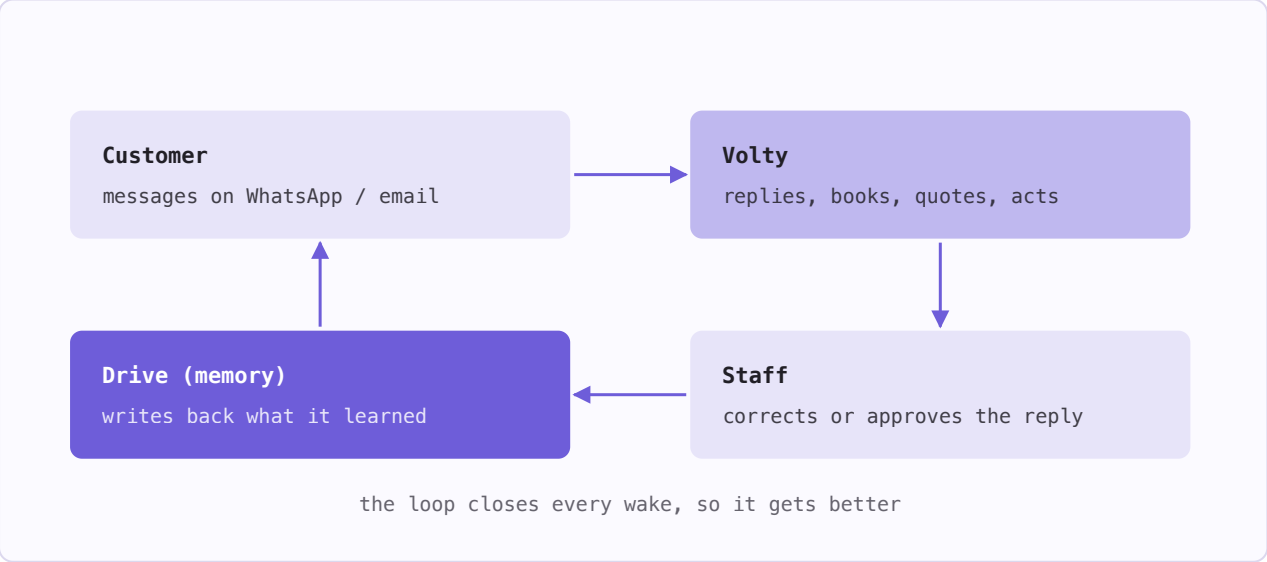


Figure 5 Volty learns from every correction: a staff reply becomes memory in the drive, so the next case is handled better. Week on week, without a retraining cycle.

7 Tables and tasks: structured data, and work you can watch

The drive holds what the agent knows in prose. Two other surfaces hold what a business needs in rows.

Tables are organization-defined structured data: typed column definitions over row storage, with one filter language shared by the agent's queries and the grid's address bar, so what a person filters and saves as a link and what the agent queries are the same thing. Every part of the surface has an operation behind it, which means the agent can create a table, populate it, and query it under exactly the gates a staff member faces. A business that needs a register, a roster, or a pipeline does not need a per-tenant schema change to get one.

Tasks are the platform's work ledger. Any multi-step operation that takes real time (importing a website, extracting an upload, syncing a drive source, standing up a WhatsApp number) writes one durable, organization-visible row with its progress and its outcome recorded separately, so "finished" and "finished successfully" are never the same field.

8 Channels and platform lines

Volty separates three things that are usually muddled together: the **agent** (the actor), the **channels** (the transports a business owns, its WhatsApp Business number, its email), and the **platform lines** (shared Voltade infrastructure).

Businesses bring their own channels: a WhatsApp Business number via Meta's official embedded signup, including numbers that keep using the WhatsApp Business app alongside (coexistence); Facebook Messenger and Instagram direct messages; a web chat widget on their own site; and email, either a forwarding address over SES or a connected Outlook mailbox, with Gmail awaiting Google's verification. These route into the organization's inbox, owned by the organization, walled by RLS.

Every inbound webhook is authenticated before it is read: Meta and the email and payment providers sign theirs, and Microsoft's carry a shared secret. Links in messages are checked against Google's list of harmful sites, and file types that can carry code are flagged, never opened by the platform, and can be discarded before storage.

Two shared platform lines sit alongside. A **Relay** line carries staff-facing traffic: notifications, approval cards, one-time sign-in codes, onboarding. A **Sandbox** line covers testing, where a tester can try any organization they're entitled to without a separate number being provisioned. One shared line serves everyone, and routing disambiguates who's who. A business never needs a number provisioned for staff traffic; a platform number is granted to an organization only when it asks to send broadcasts from one.

9 Inboxes: visibility and privacy by design

Channels are pipes. **Inboxes are containers**, and they're where the privacy model lives. An inbox is either private (one person's own mailbox or number, optionally with delegates) or shared (a team, with explicit membership). Membership is enforced by RLS: a person, including an organization admin, cannot see conversations in an inbox they're not a member of.

That means the privacy boundary isn't a setting that can be misconfigured into an open door. A staff member's individual inbox is invisible to the rest of the organization by default, and team inboxes show only to their members. Delegation is explicit and can expire. The agent is enrolled into inboxes the same way a person is, so visibility is uniform whether the member is human or machine. Admins can see who changed and who viewed each record and settings page.

Personal data gets its own wall. Phone numbers, email addresses, national ID numbers (Singapore, Malaysia, Indonesia and Hong Kong formats) and card numbers are replaced with placeholders when a message arrives, and the real values go into a vault encrypted with a per-organization key. The agent works with placeholders throughout; the real value is filled in only when a message leaves for the customer, the team, or a connected system. Staff see a locked chip, and every reveal is recorded. Just before any request reaches a model, a final scan masks any identifier that slipped through and pages the team. Contact fields marked sensitive never enter the agent's context unless it asks with a stated reason, which is recorded. Names, addresses written in prose, and the contents of photos and documents are not masked and are passed to the model as they are.

10 Models: the gateway, the choice per task, and zero retention

Volty is **model-neutral**, and it does not run every task on one model. A reply to a customer, a scanned invoice, a voice note and the nightly review each go to the models tested for that job.

10.1 One gateway, one approved list

Every model call goes through a single gateway, OpenRouter, that abstracts away which provider is behind it and meters it per organization. Each organization gets its own key, minted the first time it needs one, and every request writes a cost record against that organization, so usage and spend are tenant-scoped rather than estimated after the fact. Plans carry a weekly allowance, admins can cap each staff member and each automation, and each key has a monthly ceiling the gateway itself enforces.

Each request names the hosting companies allowed to serve it, and a host that is not on Volty's approved list cannot receive it. There is no second route to a model provider anywhere in the code.

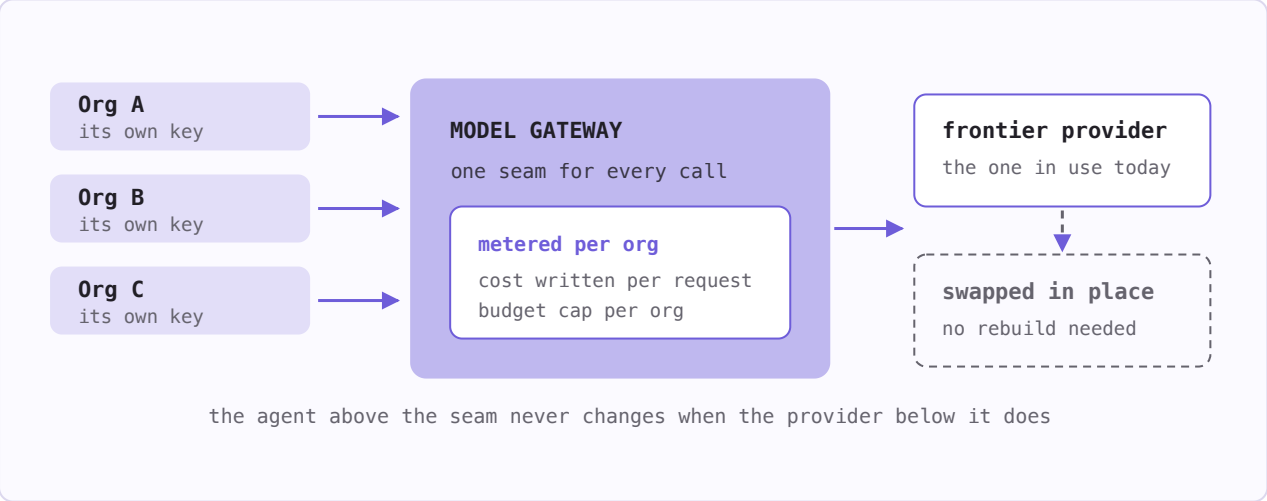


Figure 6 Every model call carries the organization's own key and writes a cost record against it. Because the providers sit behind one seam, the one underneath can be changed without touching the agent.

The seam has been exercised rather than merely asserted: production moved onto this gateway in 2026 with no per-tenant migration, and the previous gateway was switched off in September.

10.2 A ranked chain per task

Each task, which the code calls a purpose, has a chain of up to five models. The first is the one Volty asks for; if it is down, slow or refuses, the gateway moves to the next within the same request, so a customer still gets a reply. The chains in production on the date of this paper:

Task	What it does	Models, in order
Customer reply	Replies on WhatsApp, email, web chat, Messenger, Instagram	GPT-6 Luna (OpenAI), GLM-5.3 Flash (Z.ai), DeepSeek V4.1 Flash (DeepSeek)
Ask Volty	The team's assistant over the workspace	GLM-5.3 Flash, GPT-6 Luna, DeepSeek V4.1 Flash
Nightly review	Learns from the day and proposes fixes	GPT-6 Luna, Claude Haiku 5.5 (Anthropic), GLM-5.3
Second opinions	Reviews automations before deploy, confirms every High or Urgent priority, checks templates and broadcasts against policy	GPT-6 Sol (OpenAI), Kimi K3 (Moonshot), Claude Sonnet 5.5 (Anthropic)
Document reading	Invoices, statements, scans, charts	Gemini 3.8 Flash (Google), GPT-6 Sol, Kimi K3, Claude Sonnet 5.5
Photo captions	Makes Drive images searchable	GPT-5.6 Luna, Gemini 3.1 Flash Lite, MiMo V2.6 Flash (Xiaomi), GLM-5.3 Flash
Voice notes	Speech to text	Gemini 3.1 Flash Lite, MiMo V2.6 Flash
Web search	Looks things up on the public web	GPT-5.6 Luna, Gemini 3.1 Flash Lite
Short background tasks	Translation, handoff summaries, briefings, titles, triage, inbox sorting, suggestions	GPT-6 Luna, GLM-5.3 Flash, DeepSeek V4 Flash
Automation drafting	Writes an automation from a description	GLM-5.3 Flash, DeepSeek V4.1 Flash, GPT-5.6 Luna
File search	Embeddings for Drive search	text-embedding-3-small (OpenAI)
Decision steps	Yes or no questions inside smart inboxes and chat flows	Jev 1.14 (TypeSafe)

Reasoning effort is set per task too: minimal for sorting and titles, low for customer replies, medium for the team assistant and the nightly review. Admins can turn it up or down for their workspace.

10.3 How a model earns its place

A model is chosen for a task, never for the platform. For each task Volty weighs:

- **Cost against the task's shape.** A customer reply reads a long context and writes a short answer; a document read is mostly images. Each candidate is priced against the real mix of input and output that task produces.

- **Reasoning need.** Sorting a message needs little thought. Checking an automation before it touches customers needs a lot, so those tasks use larger models.
- **Modality.** Photos, pages and audio go only to models that read them; the chain is checked for this at build time.
- **Throughput and uptime.** A candidate must serve at least 50 tokens a second on an approved host with at least 99% uptime.
- **Benchmarks as a filter, never a reason.** Public leaderboards can rule a model out; they cannot put one in.
- **At least two vendors per chain**, so one company's outage or policy change never stops a task.
- **A price band on backups.** A backup may cost at most seven times the first choice on input and output, so an outage cannot quietly multiply the bill.

The structural rules are enforced by tests that fail the build: every chain entry is an admitted model, chains are one to five long, every rank can do the task, every chain spans two vendors and stays inside the price band.

10.4 How models are evaluated

Volty evaluates models on its own work, not on public test sets:

- **Golden scenarios.** Fixed suites run the real agent on real models against seeded organizations: 16 customer scenarios and 15 team-assistant scenarios. Each is graded against what happened in the database (was the contact updated, was the right reply sent, was the action held for approval), not by another model's opinion. Smart inbox filters have their own suite with a floor of 90% precision and 80% recall on every clause and a p95 latency under one second.
- **Measured trials for hard tasks.** Document reading and photo captioning were chosen by running candidates over real pages and photos and comparing accuracy, latency and cost.
- **Every reply, scored live.** Deterministic detectors score each production reply for known failure signs: garbled text, repetition loops, broken tool-call arguments. A host that starts failing is quarantined automatically, a reply that degenerates while streaming is cut off before it is sent, and any captured request can be replayed, masked, on another model or host to compare them side by side.
- **Incidents outrank benchmarks.** Hosts have been removed for corrupting tool calls and dropping letters from prose; a model was demoted for telling customers that actions had been taken when they had not; loop guards were added after a model repeated the same tool call. The backup price band exists because a failover once moved traffic onto a far more expensive model.

A routing change is a code change: it passes the same tests and review as any release, ships with the deploy, and is rolled back the same way. A week after a change, its observed cost and quality are compared against the old choice. Chains are revisited when a stronger model ships, a price moves, or a task misbehaves.

10.5 Zero data retention

The gateway account is set to **zero data retention**: an approved host processes a request in memory and keeps nothing once it has answered. Every request also carries a do-not-collect flag, so a host that would store or train on prompts is never offered the request. No provider Volty uses may train on customer data.

Open-weight models (GLM, DeepSeek, Kimi, MiMo) are served by independent hosts on the approved list that meet zero retention, never by the lab's own servers. Claude and Gemini are served through Amazon Bedrock and Google Vertex under the same rule. An hourly job reads each endpoint's published retention status, hosts that stopped meeting it have been removed from the list, and a model with no zero-retention host simply cannot be served.

Two exceptions are stated plainly:

- **OpenAI.** OpenAI models are exempt from the account-level setting so that OpenAI's own endpoint can serve them; the only zero-retention host for that family costs about twice as much. When OpenAI's endpoint answers, OpenAI may keep the request for a limited period for abuse monitoring under its API terms, and does not train on it. When Azure serves the same model, zero retention applies.
- **TypeSafe Jev**, the decision model behind smart inbox filters and chat flows, receives the do-not-collect flag; its host's retention terms are being confirmed in writing.

Every request has already been through the masking in Section 9, so a provider sees placeholders, not phone numbers or ID numbers. Volty keeps its own copy of each request and response for about 90 days for debugging, deleted sooner if the organization sets a shorter retention period.

10.6 What a business controls

Every plan runs the same chains; a cheaper plan does not get a weaker model, and a customer cannot pin a model, because a pinned model would skip the testing and the backups above. On the Max plan and above, an admin can rule out whole AI vendors, OpenAI included; a setting that would leave any task without a model is refused, and a turn whose every option is ruled out fails rather than using a banned vendor. Volty does not accept a customer's own model key, so every request stays under the rules in this section.

11 Authentication and per-action authority

Volty is passwordless. People sign in with Google, Microsoft, an email one-time code, or WhatsApp, and invitations carry a sign-in link that expires after 24 hours. A bot check guards the routes that send codes. The first three create an account on first sign-in; creating an organization goes through signup or an invitation. WhatsApp is deliberately not one of them: a code goes only to a number already linked to an account, and an unknown number gets the same "code sent" response and no message, so the sign-in endpoint cannot be used to test which numbers are registered. Identity is multi-tenant: a person can belong to more than one organization, and there are two admin tiers, platform admins (Voltade, across tenants) and organization admins (within their own business).

The subtle part is authority during a wake. When a staff member steers the agent, its permissions are re-derived **every turn** from that person, and enforced at the moment a tool runs, not assumed once at the start. If the person has left the organization, the turn fails closed. A staff member steering the agent in one organization can't have that authority bleed into another. There is no ambient "admin mode". There is only the authority of the specific caller, checked at the specific action. This is what closes the confused-deputy gap that most agent stacks leave open.

For approvals that arrive outside a logged-in session, such as an approval sent by email or WhatsApp, a one-time signed link carries just enough authority to decide that one action, and nothing more. API and CLI keys are named, scoped to one workspace, and revocable from the Security settings; CLI keys expire after 14 days.

12 How Volty is run

Volty runs on AWS, defined entirely as code (OpenTofu) across isolated production and staging environments. Data is hosted in the ap-southeast-1 (Singapore) region. Compute is on managed containers behind a load balancer that accepts traffic only from the CloudFront edge, over TLS 1.2 or newer. The database is managed Postgres fronted by a transaction-pooling proxy, the same proxy discipline that makes the per-transaction isolation in Section 1 reliable. Object storage holds the drive's larger files.

Secrets never sit in the repo in plain text: application secrets are encrypted in the repository and loaded into a secrets manager, internal credentials are generated by the infrastructure code, and the credentials an organization connects are each encrypted with AES-256 before they are stored. Engineers sign in through single sign-on. The production database is not on the public internet and is reachable only over a private network (Tailscale); the everyday route into it is read-only and refuses to open until the engineer states a reason, which is logged with every statement and kept for a year. The database is backed up continuously with 14 days of point-in-time recovery. Deploys are continuous from one branch, with database migrations run as a gated step, a freeze switch, and a one-step rollback.

None of this is visible to the businesses on Volty, which is the point. The infrastructure is the part you should never have to think about.

What this architecture buys you

Put the twelve pieces together and you get properties a chat window cannot have:

- **Isolation you can explain to your customers.** Data separated at the database, roles separated within the business.
- **A bounded agent.** It can only do declared operations, each gated by role and risk, with no hidden powers, and the same gates apply to automations and to external tools an organization connects.
- **An auditable record.** Every customer-affecting change and every shift in the agent's knowledge has a history, and long-running work has a ledger anyone in the organization can read.
- **Supervision by default.** Off until you choose, draft before live, a human able to take over at any point, and a fixed set of actions that always waits for a person.
- **Determinism where it belongs.** Fixed rails for the work that should never be re-decided, with a dry-run on real data as the approval gate.
- **The right model for each task.** Chains chosen and tested per task, backed by a second vendor, metered per business, served under zero data retention with the exceptions named, and a gateway we have already swapped in production.
- **No per-tenant drift.** One deployment, everyone current, improvements shipped to all at once.