MENTAT

SECURITY WHITEPAPER

# Single-Tenant VPC Deployment

Where your data lives, and what leaves your network.

INTEGRITY EVOLUTION INC

AIMENTAT.DEV

**Audience:** the security team reviewing Mentat for deployment inside your own cloud account.

**Scope:** Mentat's single-tenant, in-VPC deployment. It answers the question a security review actually asks: where does our data live, and what leaves our network.

> Every egress claim below traces to a specific line of application source, and a continuous-integration drift gate fails our build if a claim and the code disagree, or if a named config gate stops existing. If your review wants to check the citations against the code directly, ask for the annotated source.

---

## 1. Deployment model and the data-residency boundary

Mentat runs entirely inside your VPC.

**Everything stateful stays in your cloud account:** PostgreSQL with pgvector, every document and chunk and embedding, the knowledge graph, interview transcripts, the audit ledgers, and the credentials for any connectors you configure. Those credentials live only in your database inside your VPC, and Mentat holds none of them. The connector secrets that support encryption at rest, such as repository, Linear, Azure DevOps, and Freshdesk credentials, are Fernet-encrypted under a key ( `INSIGHT_ENCRYPTION_KEY` ) that you generate and hold.

**Two things cross the boundary, both to destinations you own and control:**

1. Model inference to Google Gemini, covering both generation and embeddings over the same host and your own enterprise no-training-tier API key.
2. Outbound email through your own SMTP relay, for magic-link sign-in and notifications.

That is the whole boundary. There is no Mentat-operated control plane, no license server, no telemetry pipeline, and no update daemon reaching back to us. Sections 2 through 6

substantiate that claim one destination at a time.

**Your prompts and responses are not training data.** Inference runs on the paid enterprise tier, whose terms exclude customer prompts and responses from model training, under an API key you hold. The optional Anthropic fallback, off unless you turn it on with your own key, runs under Commercial terms that likewise exclude training. Section 8 states the terms in full.

The approved way to state this, which we hold ourselves to:

> Single-tenant, in-VPC deployment. Your data and knowledge graph stay in your cloud account. Inference runs against enterprise no-training API tiers, or fully self-hosted on request.

---

## 2. The complete external-egress inventory

Every destination your deployment can reach is enumerated here, one row at a time. The list is generated from a machine-readable inventory checked in beside this document (`vpc-egress-inventory.yaml`), and a test fails when the two disagree or a named config gate stops existing. The single exception is Mentat's own SaaS telemetry, which your deployment never contacts and which we summarize rather than enumerate; its full destination list is in the annotated source.

**Columns:** Destination · What triggers it · Config gate · Class (core or optional) · State under the VPC profile.

### Core: required for the product to function

| Destination | Trigger | Config gate | Class | State in VPC profile |
|---|---|---|---|---|
| generativelanguage.googleapis.com | Every interview turn, extraction, synthesis, RAG | GEMINI_API_KEY | CORE | Active, your key |

| Destination | Trigger | Config gate | Class | State in VPC profile |
|---|---|---|---|---|
| generativelanguage.googleapis.com | Document and query embedding | GEMINI_API_KEY | CORE | Active, your key |
| Your SMTP relay | Magic-link sign-in, invites, digests | SMTP_HOST | CORE | Active, your relay |

### Optional model-provider fallback: off unless you opt in

| Destination | Trigger | Config gate | Class | State in VPC profile |
|---|---|---|---|---|
| api.anthropic.com | Gemini failover, only when enabled | LLM_FALLBACK_ENABLED | OPTIONAL | Off by default; on only with your own key |

### Mentat's own SaaS telemetry: disabled and boot-enforced off

Mentat's hosted-SaaS build talks to standard product-analytics and error-reporting services and sends itself a signup notification. **None of that is part of your deployment.** Under the VPC profile every one of them is disabled: the server refuses to boot if any server-side telemetry credential is set (section 7), and the browser analytics and error SDKs are compiled out of your client image entirely, so end-user browsers phone home to nothing. The specific destinations and their config gates are enumerated in the annotated source, which we provide to your review on request.

### Web import: off under the VPC profile

| Destination | Trigger | Config gate | Class | State in VPC profile |
|---|---|---|---|---|
| r.jina.ai | A user imports an arbitrary web page | WEB_IMPORT_ENABLED | OPTIONAL | Off; request rejected before any HTTP client is built |
| en.wikipedia.org | A user imports a Wikipedia article | WEB_IMPORT_ENABLED | OPTIONAL | Off; same gate |

When `WEB_IMPORT_ENABLED` is false, the import endpoints return an error before the outbound HTTP client is even constructed, so the egress is provably absent rather than merely unused.

### Billing and support: off, local by default

| Destination | Trigger | Config gate | Class | State in VPC profile |
|---|---|---|---|---|
| api.stripe.com | Checkout, billing portal, subscription changes | STRIPE_SECRET_KEY | OPTIONAL | Off; billing is a local signed file in this deployment |
| Your configured URL | An admin explicitly uploads a support bundle | SUPPORT_BUNDLE_UPLOAD_ENABLED | OPTIONAL | Off; no Mentat receiver, download is default |

## 3. Per-integration egress, on opt-in only

The connectors below become an egress only when one of your admins connects that integration. Until then, no code path calls them. Each row names the destination host so you can allowlist per integration at your egress proxy. The gate for all of them is a per-org connection stored in your database, not a process environment variable.

| Integration | Destination host(s) | Gate | State |
|---|---|---|---|
| Slack | slack.com | per-org connection | Egress only after you connect it |
| Notion | api.notion.com | per-org connection | Egress only after you connect it |
| Confluence | api.atlassian.com | per-org connection | Egress only after you connect it |
| Confluence (authorization) | auth.atlassian.com | per-org connection | Egress only after you connect it; token exchange and every refresh |
| GitHub (issues/PRs) | api.github.com | per-org connection | Egress only after you connect it |
| GitHub (authorization) | github.com | per-org connection | Egress only after you connect it; OAuth token exchange |
| Git repository ingest | The repo's own clone host | per-org connection | Egress only after you connect a repo |
| Azure DevOps | dev.azure.com | per-org connection | Egress only after you connect it |

| Integration | Destination host(s) | Gate | State |
|---|---|---|---|
| Freshdesk | your {subdomain}.freshdesk.com | per-org connection | Egress only after you connect it |
| Linear | api.linear.app | per-org connection | Egress only after you connect it |
| Microsoft Teams (reads) | graph.microsoft.com | per-org connection | Egress only after you connect it |
| Microsoft Teams (inbound signature check) | login.botframework.com | per-org connection | Egress only after you connect it; OpenID metadata and signing keys |
| Microsoft Teams (token acquisition, reads **and** replies) | login.microsoftonline.com | per-org connection | Egress only after you connect it; your own tenant for a single-tenant bot. Needed for reading as well as replying — every Graph call mints a token here first, so allowlisting graph.microsoft.com alone breaks ingestion |
| Microsoft Teams (replies) | the Bot Connector `serviceUrl` Microsoft supplies per conversation | per-org connection | Egress only after you connect it; see the note below this table |

Three details in that table are worth stating rather than leaving to be discovered while writing a firewall rule.

**The Teams reply destination is not a host we choose.** Mentat posts each reply to the `serviceUrl` carried on the stored conversation reference, and Microsoft sets that value per conversation. In the public cloud it is observed under `smba.trafficmanager.net`, but that is an observation about Microsoft's routing rather than a value this software pins, so an allowlist written against one literal host can break when Microsoft changes it. The same applies in the other direction to the Bot Framework signing keys: the key URL is read out of the OpenID metadata document rather than hardcoded. If your egress proxy needs fixed entries, allowlist the Microsoft service endpoints Microsoft publishes for Teams and the Bot Framework, not the values you observe on one connection.

**Connecting an integration reaches its authorization host as well as its API host.** Atlassian mints and refreshes tokens at `auth.atlassian.com`, and GitHub exchanges its OAuth code at `github.com`, neither of which is the API host in the row above it. An allowlist covering

only the API host produces a connector that fails at connect time and, for Atlassian, again at every token refresh.

**On-premise source control is split across two connectors.** The Git ingest clone host is derived from the repository's own clone URL, so a GitHub Enterprise or self-hosted repository is cloned from inside your network and that traffic never leaves it. The issue and pull-request connector is a separate path with fixed `api.github.com` and `github.com` constants and no Enterprise host setting, so it addresses github.com or it does not run.

Two further destinations are ones you choose rather than ones Mentat introduces, and they are in the machine-verified set for completeness. Single sign-on authenticates against your own identity provider. The experimental Buzz messenger channel is off by default and connects only to a relay you configure.

| Destination | Trigger | Gate | State |
|---|---|---|---|
| Your Entra tenant (`login.microsoftonline.com`) | Microsoft single sign-on, if you configure it | per-org connection | Your own identity provider, not a Mentat one |
| A relay URL you configure | The experimental Buzz channel, if you enable it | `BUZZ_AGENT_ENABLED` | Off by default; connects only to your relay |

## 4. License posture

The license is an offline, Ed25519-signed file. Mentat verifies it locally against a public key embedded in the build. It has **zero network dependency**, because there is nothing for it to call: no license server exists. An expired license degrades the instance to read-only; it never destroys or exfiltrates data. Three conditions refuse start rather than run in an unknown state: a tampered or forged license, a missing or unreadable license file while enforcement is enabled, and a host clock that predates the license's issuance beyond tolerance. The first is a security failure; the other two are ops-visible config errors, and the boot message names which one occurred.

## 5. Support bundle posture

Nothing is ever sent automatically. When you need to share diagnostics, an admin generates a support bundle explicitly. Its contents are allowlisted in code and secret-redacted by a scrubber pass, and the admin inspects the sanitized bundle locally before anything leaves. The default path is download-and-send-yourself; there is no Mentat endpoint that receives it. Code, graph data, and interview transcripts are excluded by construction: no collector in the bundle reads them. Direct identifiers such as email addresses that may appear in log text are masked by the scrubber pass before staging, which is best-effort redaction rather than a structural guarantee, which is why the admin reviews the sanitized bundle before sending it.

---

## 6. Billing posture

No usage telemetry streams out. Seat usage is tracked locally and exported as a signed file that you download and send at renewal. Your day-to-day operation does not depend on any billing call leaving your network.

---

## 7. In-VPC security properties

Some properties from Mentat's general architecture carry directly into the single-tenant deployment. They matter to your review, so they are restated here.

**Phone-home is provably off, not merely idle.** Under the VPC profile, the server validates its own configuration at startup and refuses to boot if any Mentat-infrastructure credential is set, whether product-analytics, log-export, error-reporting, in-app issue-report, billing, or signup-notification (the full list is in the annotated source). The same check refuses any configuration value that still points at a Mentat domain, and refuses a configuration that leaves the tenant open in ways a single tenant must not be — single-tenant mode off, or

public self-serve signup available (again, the conditions are enumerated in the annotated source, which is the list the code executes). A dirty configuration fails at migration time, before the server ever serves a request, with one error listing every offending setting.

The signup condition is the billing one reached from the other side, and it is worth stating separately because it is the condition an already-running installation is most likely to meet on upgrade rather than at install. Your deployment is contract-billed, so there is no self-serve purchase path in it — the billing credential is empty by the check above, and a signup funnel ending at a checkout would have nothing to sell.

Do not read these conditions as things a configuration satisfies by staying silent. A setting the environment file omits does not arrive unset: the container definition substitutes a default, and for many settings it does so on an explicitly empty value too. Where that default is the wrong one — a Mentat domain, or simply the wrong setting — omission is what the check catches. This is why the supplied template assigns settings rather than leaving them out, and why it is the thing to install from. The server refuses to start rather than run a tenant whose configuration contradicts its billing model, and the error names every setting at fault. Upgrading an existing installation may need a line added to the environment file; the install runbook's upgrade procedure carries the step.

**The audit ledgers are append-only, enforced by database privilege.** There are two: `mcp_call_log` and `document_audit_event`. The application's database role can insert into them but cannot update or delete, because a separate maintenance role owns them. You can verify this on your own instance in about ten seconds: connect, `SET SESSION AUTHORIZATION` to the application role, run `DELETE FROM mcp_call_log WHERE false`, and observe `permission denied for table mcp_call_log`; repeat against `document_audit_event`. The guarantee is a database grant, not an application promise.

**Authentication** is magic-link plus JWT.

**Authorization has three layers, and one deliberate limit you should know before you connect a source.**

The first layer is the organization boundary. Document retrieval and semantic search over captured knowledge both constrain results to the acting member's organization in the

query itself rather than filtering after the fact, and under the VPC profile there is no second organization in the instance for a missed filter to reach.

The second is roles. Each member carries a role, defaulting to the least privileged, and administrative endpoints sit behind a single shared dependency that rejects any non-admin caller rather than each route re-implementing the check.

The third applies to onboarding records, where "who may read this" depends on relationship rather than rank. The acting member resolves to a role relative to the record — the hire's manager, the hire, a matched buddy, another admin, any other member of the organization — and that pairing is looked up in a deny-by-default matrix. An object type nobody has mapped yet returns no access rather than falling through to permitted, and a cross-organization read is refused before the matrix is consulted at all. That refusal is deliberately indistinguishable from the record not existing, so a probe cannot confirm that another tenant's record is there.

**The limit: Mentat does not mirror the permissions of the systems it reads from.** When an admin connects a source, what Mentat ingests from it becomes readable by that organization — Mentat does not reproduce per-channel, per-page, or per-repository access rules from Slack, Notion, Confluence, or your source control, and it does not attempt to infer them. This follows from what the product is: a shared knowledge base for a team, where a captured answer is useful precisely because the next person can find it. It has a direct consequence for your rollout, so we would rather you read it here than discover it later. Connect sources whose contents are appropriate for everyone in the organization, and treat a restricted channel or a private page as a deliberate decision to be made, not a default to be inherited. If your requirement is that retrieval enforce the source system's own ACLs per user, this deployment does not do that today, and you should raise it with us before a pilot rather than after.

**Single-tenancy is structural:** the cross-organization aggregate loops that exist in the multi-tenant SaaS are disabled by the `SINGLE_TENANT` setting, which the boot validator requires to be on. There is no second tenant in your instance to leak toward.

**The host is reachable only through AWS Systems Manager, never SSH.** The instance has no public IP address and no inbound SSH port; its security group accepts inbound only from the load balancer, on 443. Egress is 443, plus one narrow exception: if you point Mentat at an external SMTP relay, its submission port (587 by default) opens to that relay's address alone, nowhere else. Operators reach the host through SSM Session Manager, authenticated against your IAM and logged to CloudTrail. There is no standing shell exposed to the network. The instance metadata service is IMDSv2-only, and both the root and data volumes are encrypted.

**The stack is deployed under a scoped, least-privilege identity you can read before it runs.** Standing the instance up is Terraform, applied under a dedicated IAM role whose complete permission set is defined in the deployment code, not a broad administrator. Everything nameable (its IAM roles, secrets, storage bucket, and log group) is scoped to this one stack's resources; the compute and load-balancer permissions that AWS does not let you scope below the account are bounded to a single region and called out for review. `terraform output deployer_policy_json` renders the exact policy document, so your security team approves the privileges the deployment will use before a single resource is created.

**The deployment is reproducible, reversible, and validated end to end on real AWS.** The same Terraform that stands the stack up tears it down. Before shipping we ran the full lifecycle under that same scoped role: apply, boot to a healthy instance serving only through the load balancer, confirm the phone-home and audit-ledger checks above on the running box, then destroy. The identity is proven against what it creates and what it removes alike. One safeguard is deliberate: the encrypted data volume is protected against an accidental teardown and must be released as a separate, explicit step, so rebuilding the instance never takes your data with it.

Three properties describe the turnkey AWS deployment: SSM-only host access, the scoped deploy identity, and the validated apply-to-teardown lifecycle. If you provision the host yourself on another cloud, host access and the deploy identity are yours to define to your own standards, and the equivalent controls are your platform's to apply. The data-residency boundary, the egress inventory, and every property in sections 1 through 6 hold

identically regardless of cloud, because they are properties of the application, not of the infrastructure it runs on.

## 8. Model-provider terms

Gemini inference runs on the paid enterprise tier, whose terms exclude your prompts and responses from model training. The optional Anthropic fallback, off unless you enable it with your own key, runs under Commercial terms that likewise exclude training. Prompt and response bodies are never logged to any third-party observability service; the observability integrations, where enabled at all, carry operational metadata, not conversation content. Under the VPC profile the third-party observability integrations are off entirely (section 2).

## 9. Scope of the guarantee

We would rather state the boundary precisely than oversell it, because a claim that fails your review costs more than a claim we never made.

**What we claim:** single-tenant, in-VPC deployment; your data and knowledge graph stay in your cloud account; inference runs against enterprise no-training API tiers, or fully self-hosted on request; and every other external destination requires an explicit action of yours before it can ever fire. Which action depends on the destination: the SaaS phone-home set is pinned empty and boot-enforced under the VPC profile; the browser telemetry is absent from your build; web import is refused at request time when disabled; and the opt-in connectors, single sign-on, the Anthropic fallback, the Buzz channel, and the support-bundle upload each stay dark until you connect them or set their flag. None of these is on by default.

**What we do not claim.** This is not an air-gapped deployment, and we do not tell you that "nothing leaves your network" or that Mentat is "fully VPC compliant" as an unconditional slogan. Two destinations you own are in the path by design: Google Gemini for inference and your SMTP relay for mail. If your requirement is zero external egress, including model inference, that is a different, fully self-hosted deployment, and we should talk about it directly rather than stretch this one's claim to cover it.

**We also do not claim permission-aware retrieval.** Reads are scoped to your organization and to the roles and relationships described in section 7, not to each user's individual access in whichever system a document came from. Section 7 states what that means for deciding which sources to connect.

**Where to check us.** Every row in section 2 names a config gate you can read in your own environment and a destination you can confirm at your egress proxy. The audit-ledger property in section 7 is checkable in ten seconds. The point of this document is that you do not have to take any of it on faith.