

Vault Cube — API Endpoint Specification

Version: 0.2 (Draft — Updated with Quarantine, Training, Eval, Developer Mode) **Date:** February 2026 **Status:** Scoping / Pre-Development

Overview

All client interactions with the Vault Cube pass through a single FastAPI gateway. The vLLM inference engine is never exposed directly. The API serves three audiences: **end users** (via chat UI or SDK), **IT administrators**, and **the management dashboard frontend**.

Base URL: <https://vault-cube.local/api>

Authentication: Local API keys passed via `Authorization: Bearer <key>` header. Admin endpoints require an admin-scoped key.

1. Inference (Industry-Standard LLM API)

These endpoints follow the de facto industry-standard LLM API format (originated by OpenAI, now adopted universally by Anthropic, Google, and every major open-source inference engine). Customers can use any compatible client library by swapping the base URL — no vendor lock-in.

Method	Endpoint	Description	Auth
POST	/v1/chat/completions	Chat completion (streaming + non-streaming). Proxied to vLLM. Core inference endpoint.	User
POST	/v1/completions	Legacy text completion. Proxied to vLLM.	User
POST	/v1/embeddings	Generate text embeddings (if embedding model loaded).	User
GET	/v1/models	List available models. Returns model ID, size, quantization, status (loaded/available).	User

GET	/v1/models/{model_id}	Model details: parameters, context window, VRAM usage, capabilities.	User
-----	-----------------------	--	------

Notes

- `/v1/chat/completions` supports `stream: true` via SSE, matching OpenAI behavior exactly.
- Model IDs map to what's in `/opt/vault/models/` — e.g., `qwen2.5-32b-awq`, `llama-3.3-70b-q4`.
- The gateway injects audit metadata (user ID, timestamp, token count) before proxying. The user never sees this.
- Unsupported OpenAI parameters (e.g., `logprobs` if not enabled) return a clear error rather than silently ignoring.

2. Model Management

Control which models are loaded, swap between them, and manage the model library.

Method	Endpoint	Description	Auth
GET	/vault/models	List all models on disk with <code>status: loaded</code> , <code>available</code> , <code>downloading</code> .	User
GET	/vault/models/{model_id}	Detailed model info: size on disk, VRAM required, quantization, benchmark scores, description.	User
POST	/vault/models/{model_id}/load	Load a model into GPU memory. Unloads current model if single-model mode.	Admin
POST	/vault/models/{model_id}/unload	Unload model from GPU memory.	Admin
GET	/vault/models/active	Returns currently loaded model(s) and their GPU allocation.	User
		Import a new model from	

POST	/vault/models/import	USB/mounted drive. Accepts path, validates format.	Admin
DELETE	/vault/models/{model_id}	Delete a model from disk. Refuses if model is currently loaded.	Admin

Notes

- Loading a model restarts the vLLM container with the new model config. Expect 30-90 seconds downtime during swap.
- Multi-model serving (running 2+ models simultaneously) is a post-MVP feature. For MVP, one model at a time.
- Import validates safetensors format, checks available disk space, and verifies VRAM fit before accepting.

3. Conversations & History

Local conversation storage for the chat UI. All data stays on-device.

Method	Endpoint	Description	Auth
GET	/vault/conversations	List user's conversations. Paginated, sorted by last activity.	User
POST	/vault/conversations	Create a new conversation. Optional: title, system prompt, model override.	User
GET	/vault/conversations/{id}	Get full conversation with all messages.	User
PUT	/vault/conversations/{id}	Update conversation metadata (title, system prompt, pinned status).	User
DELETE	/vault/conversations/{id}	Delete a conversation and all its messages.	User
POST	/vault/conversations/{id}/messages	Add a message to conversation. Triggers	User

		inference if role is <code>user</code> .	
GET	<code>/vault/conversations/{id}/export</code>	Export conversation as JSON or Markdown.	User

Notes

- Conversations are stored in local SQLite, scoped per API key / user.
 - System prompts are stored per-conversation, allowing different personas or instructions per thread.
 - Export enables customers to move conversation history off the device for archival.
 - Search across conversations is a post-MVP feature.
-

4. Documents & RAG (Post-MVP, Scoped Here for Planning)

File upload and retrieval-augmented generation. Users upload documents, the system indexes them, and the LLM can reference them in conversations.

Method	Endpoint	Description	Auth
POST	<code>/vault/documents</code>	Upload a document (PDF, DOCX, TXT, MD). Queued for indexing.	User
GET	<code>/vault/documents</code>	List uploaded documents with indexing status.	User
GET	<code>/vault/documents/{id}</code>	Document metadata and indexing status.	User
DELETE	<code>/vault/documents/{id}</code>	Remove document and its index entries.	User
POST	<code>/vault/documents/search</code>	Semantic search across indexed documents. Returns ranked chunks.	User
POST	<code>/vault/collections</code>	Create a named collection of documents (e.g., "Q3 Contracts").	User
GET	<code>/vault/collections</code>	List collections.	User
PUT	<code>/vault/collections/{id}</code>	Update collection (add/remove documents, rename).	User

Notes

- RAG pipeline: upload → chunk → embed (local embedding model) → store in vector DB (likely ChromaDB or pgvector).
 - This is the feature that makes the Vault Cube sticky for law firms and healthcare — “upload your contracts/records and ask questions.”
 - MVP can ship without RAG. Phase 2 feature, but API shape should be planned now to avoid breaking changes.
 - Document storage is local, encrypted at rest.
-

5. System Health & Monitoring

Endpoints the dashboard and admin tools use to display system status.

Method	Endpoint	Description	Auth
GET	/vault/system/health	Overall system health. Returns status of each service (vLLM, API, DB, monitoring).	User
GET	/vault/system/gpu	GPU status: per-GPU utilization, VRAM used/total, temperature, power draw, fan speed.	User
GET	/vault/system/resources	CPU usage, RAM usage, disk space (OS drive + model drive), uptime.	User
GET	/vault/system/inference	Inference stats: requests/min, avg latency, tokens/sec, active requests, queue depth.	User
GET	/vault/system/services	Status of all managed services: vLLM, Prometheus, Grafana,	Admin

		Cockpit, API gateway.	
POST	/vault/system/services/{name}/restart	Restart a specific service.	Admin
GET	/vault/system/logs	Paginated system logs. Filterable by service, severity, time range.	Admin
GET	/vault/system/logs/stream	WebSocket endpoint for real-time log streaming.	Admin

Notes

- `/vault/system/gpu` calls `nvidia-smi` or uses NVML Python bindings. Cache for 2-3 seconds to avoid hammering.
 - These endpoints power the Grafana dashboards AND the custom management UI. Dual purpose.
 - Inference stats are aggregated from the gateway's own request log, not from vLLM directly.
-

6. Administration

User management, API keys, configuration, and audit trail.

6.1 API Key Management

Method	Endpoint	Description	Auth
GET	/vault/admin/keys	List all API keys (shows prefix, label, scope, created date — never full key).	Admin
POST	/vault/admin/keys	Generate new API key. Set label, scope (user/admin), rate limit, expiry. Full key shown once.	Admin
PUT	/vault/admin/keys/{key_id}	Update key metadata: label, rate limit, active/disabled.	Admin

DELETE	/vault/admin/keys/{key_id}	Revoke an API key. Immediately effective.	Admin
--------	----------------------------	---	-------

6.2 User Management (Post-MVP: LDAP/SSO Integration)

Method	Endpoint	Description	Auth
GET	/vault/admin/users	List users. MVP: derived from API keys. Post-MVP: synced from LDAP.	Admin
POST	/vault/admin/users	Create a user account (post-MVP with LDAP).	Admin
PUT	/vault/admin/users/{id}	Update user: role, permissions, rate limit.	Admin
DELETE	/vault/admin/users/{id}	Deactivate user.	Admin

6.3 Audit Log

Method	Endpoint	Description	Auth
GET	/vault/admin/audit	Query audit log. Filter by user, action type, time range, endpoint. Paginated.	Admin
GET	/vault/admin/audit/export	Export audit log as CSV or JSON for compliance reporting.	Admin
GET	/vault/admin/audit/stats	Aggregate stats: requests per user, tokens consumed, most-used models.	Admin

6.4 System Configuration

Method	Endpoint	Description	Auth
GET	/vault/admin/config	Current system configuration (network, TLS, default model, rate limits, etc.).	Admin
PUT	/vault/admin/config	Update configuration. Validates before applying. Some changes	Admin

		require service restart.	
GET	/vault/admin/config/network	Network settings: hostname, IP, DNS, proxy (if applicable).	Admin
PUT	/vault/admin/config/network	Update network config.	Admin
GET	/vault/admin/config/tls	TLS certificate info: issuer, expiry, type (self-signed/custom).	Admin
POST	/vault/admin/config/tls	Upload custom TLS certificate and private key.	Admin

Notes

- Audit log records: timestamp, user/key ID, endpoint, method, model used, token count (input + output), response time, status code.
- Audit log does NOT store request/response content by default. Optional "full logging" mode for compliance (admin toggle, with storage implications warning).
- Config changes that require restart return `202 Accepted` with a `restart_required: true` flag.

7. Updates & Maintenance

Air-gapped update lifecycle.

Method	Endpoint	Description	Auth
GET	/vault/updates/status	Current system version, last update date, update history.	Admin
POST	/vault/updates/scan	Scan for update bundle on mounted USB/external drive.	Admin
GET	/vault/updates/pending	Details of detected update: version, changelog, components affected, size.	Admin
POST	/vault/updates/apply	Apply pending update. Returns job ID for progress tracking.	Admin

GET	/vault/updates/progress/{job_id}	Update progress: percent complete, current step, logs.	Admin
POST	/vault/updates/rollback	Rollback to previous version. Only available if previous image retained.	Admin
GET	/vault/updates/history	Full update history with versions, dates, and results.	Admin

Notes

- Update bundles are GPG-signed. `/scan` verifies signature before showing anything to admin.
- `/apply` is a long-running operation. Frontend polls `/progress` or uses WebSocket.
- Rollback retains the previous container images and config. One version back only for MVP.
- Update can include: new container images, new/updated models, OS security patches, API gateway code, dashboard code.

8. First-Boot / Onboarding

Used only during initial setup. These endpoints are unauthenticated (protected by first-boot state flag).

Method	Endpoint	Description	Auth
GET	/vault/setup/status	Returns setup state: <code>pending</code> , <code>in_progress</code> , <code>complete</code> . If <code>complete</code> , all other setup endpoints return 404.	None
POST	/vault/setup/network	Configure hostname, static IP or DHCP.	None*
POST	/vault/setup/admin	Create admin account and generate first admin API key.	None*
POST	/vault/setup/tls	Choose TLS mode: generate self-signed or upload custom cert.	None*

POST	/vault/setup/model	Select default model from pre-loaded options. Triggers model loading.	None*
GET	/vault/setup/verify	Run system verification: GPU check, inference test, service health. Returns results.	None*
POST	/vault/setup/complete	Finalize setup. Locks setup endpoints permanently. Starts all services in production mode.	None*

**None = unauthenticated, but only accessible before setup is marked complete. After setup, these return 404.*

Notes

- First-boot wizard in the frontend calls these sequentially.
 - `/setup/complete` is irreversible. To re-run setup, admin must factory reset from Cockpit.
 - The verify step actually sends a test prompt through the full stack: API → vLLM → response. If it works, the system is healthy.
-

9. WebSocket Endpoints

For real-time features in the dashboard.

Endpoint	Description	Auth
<code>ws://vault-cube.local/api/ws/inference</code>	Real-time inference stream (alternative to SSE for chat).	User
<code>ws://vault-cube.local/api/ws/system</code>	Live system metrics push: GPU util, temps, request rate. Dashboard subscribes on load.	User
<code>ws://vault-cube.local/api/ws/logs</code>	Live log stream for admin console.	Admin
<code>ws://vault-cube.local/api/ws/updates</code>	Update progress stream during update apply.	Admin

10. Quarantine & Data Ingestion Security

Every file entering the Vault Cube — whether uploaded via browser, imported from USB, or submitted as training data — passes through a multi-stage quarantine pipeline before reaching production storage. This applies to document uploads (RAG), training datasets, model imports, and software updates.

10.1 Quarantine Pipeline Endpoints

Method	Endpoint	Description	Auth
POST	/vault/quarantine/scan	Submit files for quarantine scanning. Accepts multipart upload or path to mounted USB volume. Returns job ID.	User
GET	/vault/quarantine/scan/{job_id}	Scan progress and results per file: stage, status, findings.	User
GET	/vault/quarantine/held	List all files currently in quarantine hold (flagged by any scan stage). Paginated.	Admin
GET	/vault/quarantine/held/{id}	Details for a held file: which stage flagged it, why, risk severity, preview if safe.	Admin
POST	/vault/quarantine/held/{id}/approve	Admin override: approve a flagged file and move to production storage. Requires reason (logged to audit).	Admin
POST	/vault/quarantine/held/{id}/reject	Reject and delete a flagged file. Logged to audit.	Admin
GET	/vault/quarantine/signatures	Current ClamAV and YARA signature versions, last updated	Admin

		date.	
GET	/vault/quarantine/stats	Aggregate stats: files scanned, flagged, approved, rejected, by time period.	Admin
PUT	/vault/admin/config/quarantine	Configure quarantine behavior: strictness level, auto-approve thresholds, PII action (flag/redact/block).	Admin

10.2 Quarantine Pipeline Stages

Files pass through four stages sequentially. Any stage can flag a file for quarantine hold.

Stage 1 — File Integrity & Validation: Magic byte verification, file structure validation per format, size/count limits, archive bomb detection. Pure code, no signatures needed.

Stage 2 — Malware Scanning (Offline): ClamAV with pre-loaded signatures, custom YARA rules, known-bad hash blacklist. Signatures refreshed via software update bundles.

Stage 3 — Content Sanitization: PDFs stripped of JavaScript/executables and rebuilt clean. Office docs stripped of macros/ActiveX. Images re-encoded. All metadata scrubbed. Uses dangerzone-style pixel-based sanitization (runs entirely offline).

Stage 4 — AI-Specific Safety Checks: Training data format/quality validation, prompt injection pattern detection, data poisoning heuristics (statistical outlier detection), PII scanning via regex + local NER model, model file validation (safetensors only — pickle can execute arbitrary code).

Notes

- Quarantine runs asynchronously. Large USB imports may take minutes.
- Files in quarantine hold are stored in an isolated directory, not accessible to production services.
- Admin approve/reject actions are logged to audit trail with admin ID and stated reason.
- PII scanning action is configurable: `flag` (alert admin), `redact` (auto-mask), or `block` (reject file).
- Signature staleness is an inherent air-gap trade-off. Dashboard shows signature age with visual warnings (green < 30 days, yellow < 90 days, red > 90 days).

11. Training & Fine-Tuning (Phase 2)

Job-based API for fine-tuning models using LoRA/QLoRA. Long-running GPU-intensive operations managed through the gateway for resource scheduling, audit logging, and progress tracking. Can be triggered via API or through guided conversational flow in the chat UI.

Method	Endpoint	Description	Auth
POST	/vault/training/jobs	Submit fine-tuning job: base model, dataset (quarantine-cleared file ID), hyperparameters. Returns job ID.	Admin
GET	/vault/training/jobs	List all training jobs. Filterable by status, model, date.	User
GET	/vault/training/jobs/{id}	Job detail: progress %, current epoch, loss curve, estimated time remaining, logs.	User
POST	/vault/training/jobs/{id}/cancel	Cancel a running training job.	Admin
GET	/vault/training/adapters	List trained LoRA adapters with metadata: base model, training summary, creation date, metrics.	User
POST	/vault/training/adapters/{id}/activate	Load a LoRA adapter onto its base model for inference.	Admin

POST	/vault/training/adapters/{id}/deactivate	Remove adapter, revert to base model.	Admin
DELETE	/vault/training/adapters/{id}	Delete adapter from disk. Refuses if currently active.	Admin
POST	/vault/training/validate	Dry-run validation: format check, size estimate, VRAM projection, estimated time. No training started.	User

Notes

- Wraps Hugging Face `trl` / `peft` libraries — no custom training framework.
- Training datasets must have passed quarantine before being accepted.
- Chat UI can trigger these endpoints via guided conversation: user says “train a custom model,” assistant walks them through it.
- GPU scheduling: training jobs queue behind active inference. Admin can configure priority or dedicate GPUs.

12. Evaluation & Benchmarking (Phase 2)

Job-based API for running evaluation suites against models and adapters.

Method	Endpoint	Description	Auth
POST	/vault/eval/jobs	Submit eval job: model/adapter, eval dataset, metrics to compute. Returns job ID.	User
GET	/vault/eval/jobs	List all eval jobs. Filterable by model, date.	User
GET	/vault/eval/jobs/{id}	Eval results: aggregate scores, per-example breakdown, confidence intervals.	User
GET	/vault/eval/compare	Side-by-side comparison of 2+	User

		models/adapters on same eval dataset.	
POST	/vault/eval/quick	Quick synchronous eval: small batch of test prompts, inline results. For chat UI interactive testing.	User

Notes

- Wraps `lm-evaluation-harness` for standard benchmarks, custom pipeline for user-supplied datasets.
 - Eval jobs run at lower priority than inference.
 - Quick eval limited to ~50 test cases (synchronous). Full eval is async/job-based.
-

13. Developer Mode (Phase 3)

Unlocks direct hardware access for power users (research labs, ML engineers). Admin-controlled toggle with explicit resource allocation.

Method	Endpoint	Description	Auth
POST	/vault/admin/devmode/enable	Enable developer mode with GPU allocation (e.g., reserve GPU 3 for dev). Shows warnings.	Admin
POST	/vault/admin/devmode/disable	Disable, reclaim all GPUs for managed inference. Terminates active sessions.	Admin
GET	/vault/admin/devmode/status	State: enabled/disabled, GPU allocation map, active sessions, resource usage.	Admin
POST	/vault/admin/devmode/jupyter	Launch JupyterHub container on allocated GPU(s). Returns URL and access token.	Admin
DELETE	/vault/admin/devmode/jupyter	Shut down JupyterHub container.	Admin

Notes

- Dev mode is a conscious trade-off: raw access vs. managed stability. Enable flow

includes explicit acknowledgment.

- JupyterHub runs as isolated container with access only to allocated GPU(s).
 - Sessions still logged to audit trail.
-

Endpoint Count Summary

Domain	Endpoints	Phase	Priority
Inference (Industry-Standard API)	5	MVP	Sprint 1
Model Management	7	MVP	Sprint 2
Conversations & History	7	MVP	Sprint 2
System Health & Monitoring	8	MVP	Sprint 1
Administration (Keys, Audit, Config)	14	MVP	Sprint 2
Updates & Maintenance	7	MVP	Sprint 2
First-Boot / Onboarding	7	MVP	Sprint 1
WebSockets	4	MVP / Phase 2	Sprint 3
Quarantine & Data Security	9	MVP*	Sprint 2
Documents & RAG	8	Phase 2	Sprint 4
Training & Fine-Tuning	9	Phase 2	Sprint 5
Evaluation & Benchmarking	5	Phase 2	Sprint 5
Developer Mode	5	Phase 3	Sprint 6+
Total	95		

*Quarantine Stages 1-3 are MVP. Stage 4 (AI-specific checks) is Phase 2.

MVP: ~57 endpoints | Phase 2: +27 | Phase 3: +5

Error Response Format

All errors follow a consistent schema:

```
{  
  "error": {  
    "code": "model_not_loaded",  
    "message": "The requested model qwen2.5-32b-awq is not currently loaded.",  
    "status": 503,  
    "details": {  
      "available_models": ["llama-3.3-8b-q4"],  
      "suggestion": "Use POST /vault/models/qwen2.5-32b-awq/load to load this mod  
    }  
  }  
}
```

Standard HTTP status codes: 400 (bad request), 401 (missing/invalid key), 403 (insufficient scope), 404 (not found), 409 (conflict — e.g., model already loaded), 422 (validation error), 429 (rate limited), 500 (internal error), 503 (service unavailable).

Authentication Model

- **API Keys:** Generated locally, stored as SHA-256 hashes in SQLite.
 - **Scopes:** `user` (inference + conversations + document upload) and `admin` (everything).
 - **Key format:** `vault_sk_` prefix + 48 random chars. Shown once at creation.
 - **Rate limiting:** Per-key, configurable. Default: 60 req/min for user, unlimited for admin.
 - **No cloud auth, no OAuth, no SSO for MVP.** LDAP integration planned for post-MVP.
-

Implementation Priority

Sprint 1 (MVP Core)

- Inference pass-through (`/v1/chat/completions` , `/v1/models`)
- API key auth middleware
- System health endpoints
- First-boot setup flow

Sprint 2 (MVP Complete)

- Conversations CRUD
- Model management (load/unload/active)
- Audit logging
- Admin key management
- Quarantine pipeline Stages 1-3 (file integrity, malware, sanitization)

Sprint 3 (MVP Polish)

- WebSocket endpoints (system metrics, log streaming)
- Update mechanism (USB scan, apply, progress)
- Advanced config endpoints (network, TLS)

Sprint 4 (Phase 2 — Data Platform)

- Documents & RAG pipeline
- Quarantine Stage 4 (AI-specific checks, PII scanning)
- LDAP/SSO integration

Sprint 5 (Phase 2 — Training Platform)

- Training & fine-tuning job API
- LoRA adapter management
- Evaluation & benchmarking
- Chat UI guided training flow

Sprint 6+ (Phase 3 — Research Platform)

- Developer mode (GPU allocation, JupyterHub)
- SSH access management
- Custom container launching
- Multi-model simultaneous serving