The Command Center

Technical Architecture

The technical companion to the essay "AI Runs My Back Office." That piece is the operator's view; this is the wiring.

An engineer's breakdown of a self hosted personal automation platform: hardware, networking, the software stack, the AI layer, the deployment pipeline, the data layer, failure modes, and every scheduled job.

1 System overview

The Command Center is a self hosted automation platform that runs the back office of several small operations from a single operator's hardware. It spans three macOS nodes joined by a mesh VPN. One node is always on and carries the compute and server tier. A second, portable node is the control surface from which all changes are deployed. A third, stationary node is the source of truth backup and the git origin. Running on top are a web application, local language models, a vector database, three supervised assistant processes, and roughly forty scheduled jobs.

The governing rule is human in the loop: the system gathers, drafts, formats, and prepares, but every outbound or irreversible action waits for explicit operator approval. Nothing is sent, posted, paid, or deleted autonomously.

Scope and non goals.

This is a single operator platform, not a multi tenant product, and it is deliberately not highly available. There is one compute node by design. That trade off is accepted in exchange for simplicity, low cost, and legibility. The goal is leverage for one person, not five nines of uptime. Where that single node is a risk, the risk is named and mitigated rather than engineered away (see section 9).

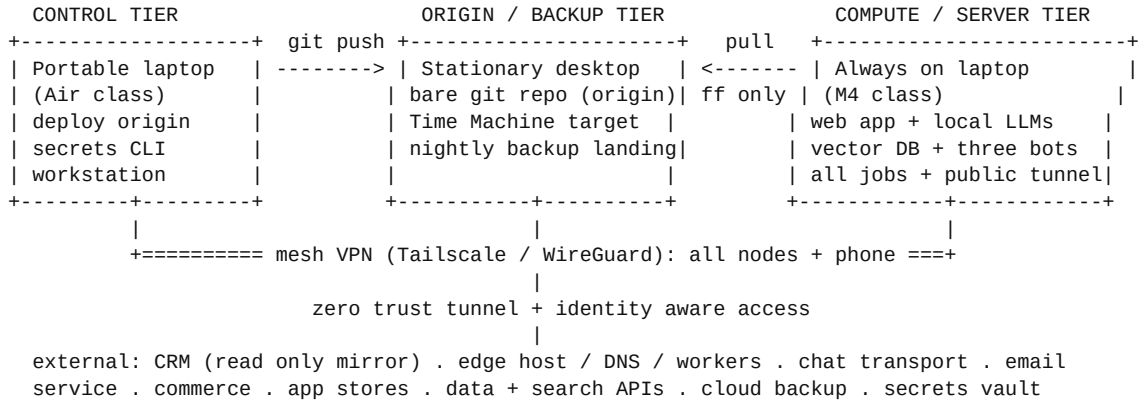

Figure 1. Three tier topology. The control node pushes code to the bare repository on the origin node; the compute node pulls from it with fast forward only. Every node reaches every other by stable name over the mesh, so the physical location of the always on compute node is irrelevant.

2 Hardware and compute topology

Node	Class and role	What it runs
Compute node	Apple Silicon laptop (M4 class), always on, never sleeps, auto restart after power loss	The application server, the local LLM runtime, the vector database, the three assistant bots, the public tunnel daemon, and every system level scheduled job. If a task must run on a fixed cadence, it lives here.

Node	Class and role	What it runs
Control node	Apple Silicon laptop (Air class), portable	The operator's daily workstation and the single origin for deployments. Holds the only signed in secrets manager CLI session. Travels with the operator.
Backup node	Intel desktop (iMac), stationary	The git origin (a bare repository), the Time Machine target, and the landing zone for the nightly mirror. Runs no production services.
External storage	Encrypted SSD plus cloud	A local Time Machine drive plus a continuous (sub hourly, on change), versioned, encrypted offsite cloud backup of the compute node. High churn caches and re downloadable model files are excluded; the rendered secrets file is excluded by path.

3 Networking, access, and security

- **Mesh VPN.** All three nodes plus a phone sit on one private tailnet (Tailscale, a WireGuard based mesh). Nodes address each other by stable name, not by location. Key expiry is disabled on the unattended nodes, the always on compute node and the stationary backup node, so neither silently drops off the network while no one is present.
- **Public ingress.** The web app is published through a zero trust tunnel rather than an open port. An identity aware access layer gates it, admitting only the operator's nodes and a signed mobile client that authenticates with a service token. DNS is managed at the same edge provider.
- **Secrets.** A password manager is the canonical secret store. A sync script renders a permission locked (chmod 600) environment file, which a launcher wrapper sources at job start. No plaintext secrets live in job definitions, and tokens are never written to logs.

Threat model and trust boundaries.

- The trust path for the dashboard is: public internet, then the identity aware access gate, then the tunnel, then the Node application. Access is the outer authorization boundary; the application also enforces its own session check behind it, so a tunnel reached directly still meets an auth wall (defense in depth).
- The control node is the highest value target because it holds the live secrets manager session. It is protected by full disk encryption, a biometric or password locked session, and a short idle timeout. Compromise of one mesh key exposes reachability, not the secrets vault, which requires a separate authenticated unlock.
- The write capable runner bot is the most privileged process. Its reach is bounded by guardrails that deny access to secrets, keys, and bot source, and it holds no credentials for sending mail, posting, or moving money. Its blast radius is file edits within allowed paths, all of which are version controlled and recoverable.

4 Software stack

The platform favors small, legible, dependency light components over heavy frameworks. The server is a hand written router, the client is framework free, and most automation is plain Python and shell. Dependencies are pinned: a Node lockfile, pinned Python requirements, fixed local model tags, and fixed database and runtime versions. Upgrades are tested on the control node before a deploy reaches the compute node.

Layer	Technology	Purpose
Application server	`Node.js (custom HTTP router)`	Single port service exposing a JSON API and the dashboard. Modular service layer; middleware for security headers, CORS, and access control. No web framework.
Client	`Vanilla JS single page app`	Framework free dashboard served as static assets. Card first responsive layout, dark canvas, a custom type system.

Layer	Technology	Purpose
Local inference	`Ollama (small model + fallback)`	Fast, private, low cost language tasks run on device without leaving the network. Caps spend on routine work.
Semantic search	`Qdrant vector database`	Embeds planning documents, journals, briefs, and records for retrieval behind the dashboard's archive search.
Assistant bots	`Python long running processes`	Three supervised chat bots on a messaging transport, with pluggable LLM providers and tool use.
Automation scripts	`Python and shell`	Data export, PDF generation, statement parsing, modeling, sync, and maintenance.
Document generation	`reportlab and poppler`	Print safe PDFs (action sheets, invoices, cards) and text or raster extraction from inbound PDFs.
Native bridge	`AppleScript (osascript)`	Drives Mail and Calendar on macOS for drafting and state capture.
CRM bridge	`Vendor CLI, read only`	Query only access to the system of record; results written to JSON mirrors.
Web and ventures	`Edge static hosting + workers`	Static sites and link routing workers, deployed by CLI; newsletter, commerce, and app store APIs.
Mobile apps	`Capacitor web wrappers`	Single file web apps wrapped to native and shipped through the app store pipelines.
Scheduling	`launchd + cloud cron agent`	System level timers on the compute node plus higher level cloud scheduled workflows.
Version control	`Git over the mesh VPN`	A self hosted bare repository on the stationary node is the deploy origin.

5 The AI layer

Inference is split between cheap local models for routine private work and hosted frontier models for reasoning and tool use. A vector store gives the dashboard and the bots semantic recall over the operator's own documents.

Three assistants, one job each

- **Assistant (chat operations).** A long running process on a messaging transport, backed by a hosted frontier model through a provider agnostic, OpenAI style chat completions API. The provider is swappable by configuration. Tools include hosted web search, directory listing, and read only file access, so it answers from the real record rather than guessing.
- **Strategist.** A second hosted model process dedicated to planning and analysis: weekly plans, idea pressure testing, build mapping.
- **Operator or runner.** A write capable file automation process with guardrails that block access to secrets, keys, and bot source.

Enforcement of human in the loop

The approval rule is structural, not advisory. The bots and services hold no credentials for sending mail, posting publicly, or moving money, and every outbound action is written to a draft or an approval queue that only a human action releases. A process cannot send what it has no key to send. This is why the rule holds even when a model is wrong.

Reliability and context

- Each bot uses per message isolation, exponential backoff, and a never silent guarantee, meaning every failure produces an operator visible message rather than a silent drop. Each bot is supervised for automatic relaunch. A daily maintenance job rotates and prunes a gateway session before it bloats and stalls the event loop.
- A single front door document is the canonical context every session reads first. It carries identity, current work, file locations, operating rules, and open threads, and a nightly job refreshes its live state section. Context is infrastructure here: a session that starts from a current front door starts informed.

6 Deployment pipeline and source control

Code is authored on the control node and never edited on the server directly. The deploy is a single script and a strict git flow.

```
DEPLOY FLOW
  author on control node  ->  git push to bare repo on origin node (over the mesh)
  compute node: git pull --ff-only  ->  copy data and config, sync content trees
  kick the server to reload   (live job definitions left untouched unless intended)

REQUEST FLOW
  operator device  ->  identity aware access gate  ->  tunnel  ->  Node app
  app reads JSON state / CRM mirror / vector DB ; calls an LLM provider as needed
  outbound results are drafts or queued actions ; nothing is sent without approval
```

Figure 2. The two primary flows. Deploys move control to origin to compute and never the reverse. Requests terminate at the application, which prepares drafts rather than dispatching actions.

- **Drift guard.** Before pushing, the deploy checks the compute node for uncommitted tracked changes and refuses to proceed if it finds any, so node canonical edits are never clobbered mid deploy.
- **Reverse sync.** A companion tool pulls compute node changes back into version control on demand (commit, push, pull), and lists untracked files for manual triage.
- **Fast forward only.** The server pull fails loudly on divergence instead of overwriting. Ignore rules keep non code artifacts out of the repository.

- **Canonical node rule.** Certain live state files and the bot processes are owned by the compute node and are never overwritten by a deploy.

7 Data layer and integrations

- **State files.** Operational records, goals, dashboard configuration, and the build pipeline live as JSON documents. Authoritative copies are cloud synced; runtime copies are per node local. Every system of record file is backed up before any edit.
- **System of record mirror.** A read only export (vendor CLI, SELECT only, never a write) runs on the compute node twice daily and writes compact JSON snapshots that are copied to the control node. Each snapshot carries a generation timestamp; consumers display the mirror age and never read through to the live system. The staleness window is at most one refresh interval (about eight hours at its widest, between the afternoon run and the following dawn run). Read only access is enforced by the connected application's permission set, not by convention.
- **Retention.** State snapshots and briefs are pruned on a rolling window; vault snapshots are kept to a fixed depth; generated PDFs age out of the working set. Sensitive operational records are minimized and never leave the owner's machines except inside the encrypted backup.

External service	Role in the system
Customer record platform (CRM)	System of record, accessed read only through a CLI; mirrored to JSON twice daily.
Edge platform (tunnel, DNS, static hosting, workers)	Public ingress for the dashboard, DNS, static venture sites, and link routing workers.
Messaging transport	Carries the three assistant bots and operator notifications.
Email service provider	Newsletter audiences and campaign sends, drafted by the system and sent by the operator.
Commerce platform	Storefront API behind a venture site.
App store and app marketplace	Release pipelines and daily metrics pulls using signed API tokens.
Data and search APIs	Hosted web search for the assistant; a sports data feed for a personal analytics engine.
Cloud backup and password manager	Offsite encrypted backup and the canonical secret store.

8 Automation and scheduled jobs

Two schedulers run the back office. System level timers (launchd) live on the always on compute node. Higher level workflows run on a cloud cron agent. The lists below are representative and total roughly forty recurring jobs. Jobs are written to be idempotent and guard against overlap with file locks, so a slow run does not stack on the next tick. A run missed during a reboot laps to the next scheduled tick rather than backfilling, except where a catch up is explicitly safe.

8.1 Compute node timers (launchd)

Job	Cadence	Function
Server, bots, vector DB	Always on (supervised)	The application server, the three bots, and the vector database relaunch on exit.
Public tunnel	Always on	Maintains the zero trust tunnel that publishes the dashboard.
Daily newspaper	06:00 daily	Generates a personal briefing PDF, with a weekend edition variant.
Record mirror export	05:30 and 13:00	Read only export of CRM contacts and deals to JSON; copied to the control node.
Daily action sheet	03:00 daily	Builds a printable outreach sheet for the day and files it to cloud storage.
State capture	every 30 min	Snapshots calendar, mail summary, and weather to local state.
Dispatch drainer	every 5 min	Flushes the outbound message queue to the chat transport.
Priority mail watcher	every 15 min, business hours	Surfaces high priority inbound mail.
Mail filing	periodic	Rule based auto filing of newsletters and automated replies.
End of day enforcer	17:00 weekdays	Applies the outreach lifecycle rules, including retire after N attempts.
Weekend re snap	Sat 23:55	Re schedules touches that drifted into the weekend.
Nightly backup mirror	02:00 daily	Mirrors the compute node to the stationary backup node.
Brief sync	06:10 daily	Pushes the morning brief from compute to control node.
Recurring greetings	daily and monthly	Birthday and milestone reminders for the relationship engine.
Meeting prep	weekly	Assembles a recurring one to one prep packet.
Overnight synthesis	03:15 daily	Reads the day's activity and produces a synthesis dig.
Vault snapshot	periodic	Point in time snapshot of the knowledge store.
Personal analytics engine	Wed and Sat, early AM	Runs a deterministic modeling pipeline and emails the output.
App metrics pull	06:15 daily	Pulls app store sales and trend reports via signed API.
Brief emailer	07:15 weekdays	Emails the morning brief, skipping weekends and holidays.
System health check	periodic	Watchdog that alerts over chat on an outage.
Front door refresh	21:00 daily	Regenerates the live state section of the canonical context document.
Gateway maintenance	04:00 daily	Rotates and prunes a bloated assistant gateway session.

8.2 Cloud agent workflows (cron)

Workflow	Cadence	Function
Morning brief	07:10 weekdays	Higher level daily briefing assembled by the cloud agent.

Workflow	Cadence	Function
End of day shutdown brief	17:00 weekdays	Closes out the day and stages tomorrow.
Lead aging sweep	Mon 08:00	Flags stale items in the pipeline.
Weekly wrap	Fri 16:30	Generates the weekly summary.
Monthly report prep	1st 08:00	Drafts the monthly results report and goal tracker.
Monthly invoice reminder	1st monthly	Triggers the invoicing run for the billing operation.
Consistency audit	Mon	Read only audit of cross file record consistency.
Evening inbox sweep	18:00 daily	Cross account inbox sweep; also feeds the weekly newsletter staging.
Weekly newsletter	Sun 18:00	Aggregates the week's captured items into a draft issue.
Venture newsletters	Fri	Stages weekly drafts for the publishing and affiliate properties.
Profile maintenance	Sun 19:00	Keeps the operator profile document current.
Periodic reminders	as configured	One off and recurring operator reminders.

9 Failure modes and recovery

The compute node is the platform's single point of failure by design. The table states what happens when each part fails, how it is detected, and the recovery path.

Component	Failure mode	Detection and recovery	RTO / RPO
Compute node	Hardware loss or extended outage. Runs the server, bots, vector DB, tunnel, and all jobs.	Health watchdog stops reporting and the dashboard is unreachable. Restore from the encrypted cloud backup onto replacement Apple Silicon hardware, rejoin the mesh, reload jobs.	RTO hours. RPO sub day (last nightly mirror plus continuous cloud backup).
Origin and backup node	Power or network loss (observed in practice).	Deploys fail loudly on push and the nightly mirror does not land. Production keeps running untouched. Deploys resume when the node returns, or the control node pushes straight to the compute node as a temporary origin.	RTO minutes once power returns. No production impact.
Mesh VPN control plane	Coordinator outage.	New authentication fails while existing tunnels keep routing on current keys. Wait out the outage and avoid re auth until restored.	No data impact.
Public tunnel and access edge	Edge provider outage.	The dashboard returns errors from outside. Local jobs and bots continue on the compute node; the dashboard returns when the edge recovers.	No data impact. Dashboard only.
Record mirror	Stale or failed export.	The snapshot timestamp ages and the export job exits nonzero. Re run the read only export; consumers respect the displayed age until then.	RPO up to one refresh interval.
A scheduled job	Crash or hang.	Nonzero exit is captured and the watchdog alerts over chat. Daemons relaunch; timers run on the next tick; lock files prevent overlap.	Per job. Most are idempotent.

10 Operational concerns

- **Observability and logging.** Every job writes timestamped logs to a per job file on the compute node, and the scheduler captures exit status, which the health watchdog surfaces over chat. Logs exclude secrets. A periodic read only audit can replay exit history and check cross file consistency.
- **Idempotency and locking.** Tight cadence jobs (a five minute dispatch drainer, a thirty minute state capture) are idempotent and use file locks so a slow run never stacks on the next tick.
- **Version pinning.** The Node lockfile, pinned Python requirements, fixed local model tags, and fixed database and runtime versions keep builds reproducible. Upgrades are staged on the control node first.
- **Secrets rotation.** Rotation is a one step operation: update the vault entry, re render the env file, reload the affected jobs. High value tokens carry calendar reminders for periodic rotation.
- **Cost and rate limits.** Routine, private work routes to the local models, reserving the metered hosted models for reasoning and tool use. Providers are called with per provider backoff and budget ceilings; a rate limited call degrades to a retry or a local fallback rather than failing the whole job.

11 Reliability, backup, and self healing

- **Backup posture (three two one).** Three copies across two media types with one offsite: a local Time Machine drive (external SSD), an encrypted offsite cloud copy, and a nightly mirror that lands on a node in a separate physical location from the compute node. The chmod 600 secrets file is excluded from the cloud backup by path. A backup that is not verified is not treated as a backup, so the system checks its own backup freshness.

- **Self healing.** Long running processes are supervised and relaunch on exit. The bots recover from network storms with retry and backoff. A periodic health check alerts over chat. A maintenance job keeps the assistant gateway from stalling.
- **Permanent fixes over patches.** Each failure yields a structural fix rather than a one off patch. A near loss of work produced the deploy drift guard and the reverse sync tool. A silent backup failure produced a freshness watcher. The system grows more resilient with each incident.
- **Self auditing.** The whole platform can be examined by a panel of read only agents that verify code health, job exit status, and cross file data consistency, then report a single health verdict.

12 Design principles

- **One source of truth per domain.** Mirror the authoritative system, never guess from a stale copy.
- **Human in the loop, enforced.** The machine prepares; releasing any outbound action requires a human, structurally, not just by policy.
- **Context is infrastructure.** A single, current front door document makes every AI session start informed.
- **Least privilege secrets.** A vault is canonical; rendered env files are permission locked; tokens never touch logs.
- **Canonical node ownership.** The compute node owns live state and bot processes; deploys never overwrite them.
- **Small and legible.** Hand written routers, framework free clients, plain Python. Easy to read is easy to fix.
- **Name the single points of failure.** One compute node by design; the risk is documented and recoverable, not hidden.

This document describes architecture only. It contains no personal, donor, patient, employer, or credential information. Component categories are named where it aids an engineer reading it; the operator's specific organizations and data are deliberately omitted.