ZephyrAB Administrator Manual

ZephyrAB 0.1.0 (preview) · September 2026

Table of contents

1. About this manual

ZephyrAB 0.1.0 (preview) · September 2026

This manual is for the person who installs and runs a ZephyrAB deployment, and for delegated administrators — tenant and domain admins — who manage their slice of it through the web console.

ZephyrAB is a multi-tenant mail and collaboration platform. One server binary (`zephyrd`) speaks SMTP, IMAP, POP3, JMAP, CalDAV, CardDAV and ManageSieve. Metadata lives in FoundationDB. Large message bodies live in a MinIO (S3-compatible) object store. A set of command-line tools handles provisioning, backups, migration and day-2 operations.

This manual covers:

- installing ZephyrAB, from one machine to a multi-host fleet
- the web console, page by page
- domains, DNS, and deliverability
- mail filtering, user security, backups, multi-cell operations, licensing and monitoring
- the command-line tools and environment variables
- migration from another mail server, and troubleshooting
- sizing a deployment and how the hosts fit together, and outbound webhooks for integration

Two companion manuals exist:

- the User Manual — webmail, mail clients, and self-service settings for end users
- the API Manual — the admin API (OAuth2, JSON) for automation

A note on tone: ZephyrAB documents its own limits. Where a feature has a known gap, this manual says so rather than hiding it. A limit you know about is one you can plan around.

2. Install ZephyrAB

Requirements

- **Ubuntu 24.04 LTS on amd64.** This is a hard requirement, not a preference: the published binaries reference GLIBC 2.39 symbols, so `dpkg` refuses to install them on anything older. Other distributions and architectures need a build from source.
- **A real host or VM with systemd.** The installer refuses containers by name — ZephyrAB manages systemd units, and a container has none.

- **Root or sudo.**
- **Memory:** 2 GiB is the hard floor for a single-box install (FoundationDB + MinIO + zephyrd on one machine); 4 GiB or more is the comfortable floor.
- **Disk:** 5 GiB free is the hard floor; 20 GiB or more is recommended. A full disk stops FoundationDB accepting writes, and it does not recover on its own at high fill.
- **Outbound port 25**, if you want to send mail. Most clouds block it by default and open it on request. The install proceeds without it — receiving is unaffected — but no mail can leave until it opens.
- **For a single-box (Solo) install: FoundationDB server.** The bootstrap script installs `foundationdb-clients` (the client library every binary links). A Solo install also runs its own cluster, so install both `foundationdb-clients` and `foundationdb-server` (7.3.x) before running `zephyrab install`. These are Apple's packages, not Ubuntu's, so plain `apt` will not find them; preflight names this and refuses until it is fixed.

Run the bootstrap

```
curl -fsSL https://zephyrab.com/install.sh | sh
```

This fetches the ZephyrAB package set, verifies every artifact against published SHA-256 checksums, and installs the packages. **Nothing starts.** The services ship disabled, because starting a mail server with an unconfigured environment is worse than not starting it.

The script deliberately does not run the setup: setup is an interview, and a script arriving through a pipe has no terminal — its stdin is the pipe. So it ends by naming the next command:

```
sudo zephyrab install
```

If you prefer to read before you run:

```
curl -fsSLO https://zephyrab.com/install.sh && less install.sh && sh
install.sh
```

Or fetch without installing: `sh install.sh --download-only ./zephyrab-debs`. Add `--with-dns` to also install the authoritative nameserver package (nameserver hosts only). `--version X` pins a version; by default the script reads the current version from the channel pointer at `/dl/latest.txt`.

Trust, stated honestly: artifacts and checksums travel over HTTPS from the same origin, so TLS to the download host is the trust anchor. The checksum step protects against corruption and mirror mix-ups, not against a compromised origin. A GPG-signed apt repository is the planned upgrade.

The plan and the interview

`zephyrab install` starts with an interview. Its output is a **plan file** — `zephyrab.plan.json`, JSON, versioned (`plan_version: 2`), reviewable and diffable. The plan never holds a secret; secrets are minted on the target machine during the install.

Unknown fields in a plan are refused on purpose: a typo'd key silently ignored is a setting you believe is in force and is not.

You can also run the pieces separately:

```
zephyrab plan new [--out FILE]        # interview only; write the plan
zephyrab plan show --plan FILE        # render it, with a topology
table
zephyrab plan validate --plan FILE    # parse and check; unknown
fields refuse
zephyrab preflight [--plan FILE]      # the host checks alone
zephyrab install --plan FILE --dry-run
zephyrab install --plan FILE
```

`--dry-run` prints every step's exact would-do — commands, generated files, all of it — and touches nothing. It is the review; run it first. The dry run is structural: with `--dry-run` the engine can only describe steps, never apply them.

The interview asks, in order:

1. **Topology preset** — solo, small, or standard (see below).
2. **Hosts and access** (multi-host presets only) — address, SSH user, key, port and roles per host. Keys only: the tool never takes or stores a password. If a host only has password auth today, run `ssh-copy-id user@host` first.
3. **The mail domain** — e.g. `example.com`.
4. **The mail hostname** — default `mail.<domain>`. This is the MX target, the TLS name, and the name your PTR record should match.
5. **The EHLO name** — default is the mail hostname. Announcing anything else is a spam signal.
6. **Certificate contact** — the ACME account email, default `postmaster@<domain>`.
7. **DNS mode** — `external` (you publish records at your DNS provider) or `zephyr-hosted`. The hosted-DNS wiring is not driven by the installer yet; the choice is recorded and behaves like external for now. Hosted DNS is configured separately (see section 7).
8. **Your own mailbox** — the admin address, default `admin@<domain>`. `postmaster@` and `abuse@` are refused here: they are role mailboxes created automatically, and they receive the platform's own operational mail.
9. **Data location** — default `/var/lib/zephyrab`.

The three presets

Preset	Hosts	What it is
solo	1 (this machine)	Everything on one box: FoundationDB (`single` redundancy), MinIO local, zephyrd. NATS is deliberately skipped — on one box, zephyrd's built-in

		FoundationDB-watch fallback covers IMAP IDLE and JMAP push. No SSH involved.
small	1–3	One host carrying roles `[edge, cell]` — it *is* the mail server (FoundationDB, MinIO, zephyrd and the Caddy web front all live there) — plus optional `obs` (observability) and `backup` hosts. Driven over SSH.
standard	5–10	3 or 5 `fdb` hosts, exactly one `blob` (MinIO) host, at least one `edge`, optional internal `cell` frontends, `obs` and `backup`. Driven over SSH.

The presets enforce their own shapes, and the refusals say why:

- **small refuses a split edge/cell.** In the small preset the edge and the cell are one host by construction of the plays it runs: zephyrd binds its JMAP/web listener on loopback and Caddy fronts it on the same machine, so a separate edge would proxy to a socket it cannot reach. Splitting the frontends from the database is what the standard preset is for.
- **small refuses two `[edge, cell]` hosts.** Each such host is a complete single-node cell, and two cells is the multi-cell shape — joining them is `zephyrab add-cell`, not a preset.
- **standard wants 3 or 5 `fdb` hosts.** FoundationDB runs `double` redundancy, which needs at least 3 nodes to survive a node loss and re-replicate onto the survivors. 5 is the measured shape: on the reference fleet, a 5-node cell held 1,900 messages per second for 24 hours where 3 nodes could not, because the dataset outgrew what 3 nodes' page cache could hold. Other counts are refused rather than guessed at.
- **standard refuses edge combined with `fdb` or `blob` on one host.** That would put the database (port 4500) or the blob store (port 9000) on a public machine with no firewall rules for them.

Choosing between them — how many machines, of what size, and which of them need public addresses — is section 19. Read it before you buy hardware; it carries the measured numbers these presets were shaped by.

Preflight

Preflight runs before anything changes, per host. A refusal names the fix and stops the install; a warning does not stop it and is restated at the end. The checks:

1. **os** — Ubuntu 22.04/24.04 only. A scope decision, not a technical wall: an installer that has never run on your distribution would be guessing with root.
2. **root** — the install writes `/etc`, creates a system user and manages units.
3. **systemd** — `/run/systemd/system` must exist. Its absence usually means a container.
4. **memory** — refuses below 2 GiB, warns below 4 GiB.
5. **disk** — refuses below 5 GiB free at the data path, warns below 20 GiB.
6. **ports** — the ports this host's roles need must be free. A port held by a foreign process is a refusal that names the holder; a port held by a ZephyrAB process is a warning (fine for a re-run, wrong for a fresh install).
7. **existing-mta** — an active or enabled Postfix, Exim, Sendmail, OpenSMTPD or nullmailer is a refusal. Two MTAs on one host fight over port 25 and each other's queues.
8. **outbound-25** — a live connect test toward real MX hosts. Blocked is a **warning**, not a refusal: receiving works either way, but no mail can leave until the provider opens it.
9. **ptr** — looks up the reverse DNS of this host's outbound address. Warnings for a missing, mismatched or provider-generic PTR: big receivers reject or junk mail whose forward and reverse DNS do not agree (FCrDNS).
10. **fdb-client** — Solo only: `libfdb_c` must be installed (see Requirements).
11. **clock** — NTP synchronization. DKIM signatures and TLS certificates carry validity windows; a wrong clock makes both fail in confusing ways.

Multi-host plans add two per-host checks first: **reach** (SSH answers with key auth, no prompts) and **sudo** (`sudo -n true` succeeds). If reach fails, the remaining checks for that host are collapsed into one line rather than eleven "could not run" lines.

What the Solo install does

Fourteen steps, each idempotent and postcondition-checked. Re-running after a failure resumes at the failed step rather than repeating. In order:

1. **system-user** — creates the `zephyr` system user and `/etc/zephyrab` (mode 2770, root:zephyr).
2. **data-dirs** — the data root and its `minio/` and `backup/` subdirectories.
3. **foundationdb** — enables the service and bootstraps the cluster (`configure new single ssd`) if no database exists. Every `fdbcli` call runs under a timeout, because against a cluster with no database `fdbcli` does not fail — it blocks indefinitely, looking exactly like progress.
4. **minio-binary** — downloads a pinned MinIO release and verifies its SHA-256. On an air-gapped host, pre-place the verified binary.
5. **minio-credentials** — mints `/etc/zephyrab/minio.env` with a random root credential. Never typed, never in the plan.
6. **minio-service** — a systemd unit binding MinIO to `127.0.0.1:9000`, and a wait for the socket.

7. **dkim-key** — generates an RSA-2048 DKIM signing key at `/etc/zephyrab/dkim.pem`. The public half is printed as a DNS record in the closing summary.
8. **tls-cert** — a self-signed certificate for the mail hostname. Clients will warn about it; that is what self-signed means. Replace it with a real certificate later (section 7, built-in ACME).
9. **admin-credentials** — mints `/etc/zephyrab/admin.env` with the admin API client credential. Without it the admin API fails closed (503).
10. **zephyrd-env** — renders `/etc/zephyrab/zephyrd.env` from the plan: listeners, identity, DKIM, TLS, S3 credentials, per-IP brakes. An existing file is left alone — your edits outrank a re-run; delete the file to regenerate it.
11. **zephyrd-service** — the systemd unit (runs as `zephyr`, `CAP_NET_BIND_SERVICE` for port 25, `ProtectSystem=strict`).
12. **listeners** — polls port 25 and port 8080 until they answer. A unit reading "active" is systemd's opinion of the process, not proof the sockets answer.
13. **first-boot-provisioning** — through the admin API with the minted credential (the audited, validated door; the installer never writes the database directly): a `Platform` tenant, your mail domain, the `postmaster@` and `abuse@` role mailboxes, your admin mailbox with a minted password **printed once**, and one invite batch.
14. **backup** — starts a continuous FoundationDB backup (`fdbbackup start -z`, 12-hour snapshots) into the data root, plus a daily retention timer. The summary states the scope plainly: this protects against operator error and corruption, not against losing the machine, and MinIO-stored bodies above 64 KiB are not in the FDB backup.

The closing screen lists where things are, the minted secrets, the admin password (printed once, stored only as a hash), the invite codes, the DNS records to publish — and the deliverability truth (see section 4).

Check the result

```
zephyrab doctor
```

Doctor is read-only, safe anywhere, and the support command: run it and send the output. It exits 1 on any failure. Its checks: the env file is readable; the `zephyrd` unit is active; FoundationDB answers and is available; every configured listener accepts a TCP connection; the SMTP banner starts `220`; an unauthenticated JMAP request answers 401 (a 200 there is treated as a serious failure — it is the shape of a cross-tenant breach); `zephyr_parser_panics_total` is zero; the accepted/delivered counters are present; the TLS certificate is valid and has more than 14 days left; a backup is running and restorable; and the license file evaluates cleanly (absent is a clean pass — community edition).

Air-gapped install

For hosts with no internet access, build a bundle on a connected machine:

```
deploy/pkg/build-bundle.sh     # produces zephyrab-bundle-
<version>.tar.gz
```

The bundle carries the .debs with their checksums, the pinned FoundationDB client package, an exact `ansible-core` dependency closure harvested by apt against clean Ubuntu 24.04, and `install.sh` itself. On the target:

```
tar -xzf zephyrab-bundle-<version>.tar.gz
sh install.sh --from-dir zephyrab-bundle-<version>
```

Zero network. Checksums are still enforced — where a file came from cannot change what it must be. The dependency closure is 24.04/amd64-specific, and the bundle's README says so.

Upgrade

```
sudo zephyrab upgrade [--version X] [--dry-run] [--yes]
```

Upgrade resolves the channel pointer (or the version you name), then runs four steps, each postcondition-checked:

1. **Fetch and verify** every artifact against its SHA-256 sums.
2. **The stored-filter scan, with the candidate binary.** The upgrade extracts `zephyr-sieve-scan` from the *fetched* tools package and runs that — not the installed copy — because the question is about the NEW parser. A stored Sieve filter that stops parsing under a new parser means a user's mail silently falls to INBOX instead of their folders. If the scan reports any such filter, the whole upgrade refuses, quoting the findings. `--skip-sieve-scan` exists and is named for what it skips; it is meant for hosts that run no mail store, not for hosts holding real filters.
3. **One apt transaction** installs the whole set, so the pinned = versions move level together. Version skew across the set is a documented hazard.
4. **Restart and prove health**: the unit is active; `/proc/<pid>/exe` is not a deleted inode (the kernel's own statement that the running process is the new binary); every configured listener is bound within 20 seconds; and `zephyr_parser_panics_total` is still zero. A zephyrd you left disabled is not started.

There is no automatic rollback, by design. An automatic rollback restarts the service again — doubling the outage — and can mask a configuration cause that would fail the old build just as hard. On a health failure, the exact downgrade command is printed instead:

```
sudo apt-get install -y --allow-downgrades zephyrab=<old> zephyrab-
core=<old> \
  zephyrab-tools=<old> zephyrab-installer=<old> zephyrab-web=<old>
sudo systemctl restart zephyrd
```

With `--plan`, upgrade rolls a fleet one host at a time in a safe order: non-edge cell hosts first, roleless hosts next, **edges last** — the user-facing surface moves only after the quiet hosts prove the build. The first failure stops the rollout and names the hosts not attempted. Hosts that run zephyrd from `/usr/local/bin` rather than from packages are refused by name: upgrading packages under a hand-copied binary would leave two zephyrds on PATH disagreeing about which one systemd runs.

Add a cell

```
zephyrab add-cell --plan FILE --cell-id ID --host NAME --address ADDR
\
  --ssh-user USER --public-hostname FQDN [--ca-dir DIR] [--init-ca]
[--mx-pref N] [--apply]
```

`add-cell` joins a new cell (a complete, separate mail store — see section 5) to an existing platform. **It is a dry run by default** — the inversion is deliberate, because a join touches every cell in the fleet and the recorded way this class of change goes wrong is silently: a peer list missing one entry does not error, it just never learns. The dry run prints every step: the mTLS feed certificate it will issue (subject, SAN, key usages), the environment lines for the new cell *and for every existing cell* (the cell map feed is pull-only, so every existing cell's peer list must gain the new one), the registration command, and — if you passed `--mx-pref` — the MX record, **printed and never published**. Pass `--apply` to execute. See section 12 for the multi-cell model.

Check DNS

```
zephyrab dns-check --domain example.com [--wait] [--timeout MINS]
```

`dns-check` asks the running platform which DNS records it expects for the domain — the same zone generator the nameserver serves from, so this tool cannot drift from the server — and prints them as a copy-paste table with a live status per record: `VERIFIED — seen in public DNS`, `not seen yet`, or `not checked (publish it; the platform does not probe this type)`. The platform probes the five mail-auth records; SRV, TLSA and the MTA-STS policy host are listed for you to publish but not probed. `--wait` polls every 30 seconds, asking the server to re-verify, up to `--timeout` minutes (default 30). Run it on the mail host — it reads the admin credential the install minted there.

3. What an installation contains

The six packages

Package	Contents
`zephyrab-core`	`zephyrd` (the server), its systemd unit (installed **disabled**), and `/etc/zephyrab/zephyrd.env.example`
`zephyrab-tools`	The 35 operator binaries: queue, cell, link and spam operations, account lifecycle, DMARC/TLS-RPT/ARF ingest, the backfills, backup and restore tooling, `zephyr-migrate` and `zephyr-tenant-move`. Installable without the server — a backup host wants the tools and not `zephyrd`.
`zephyrab-dns`	`zephyr-dnsd`, the authoritative DNS server with DNSSEC signing, and its unit (disabled). Nameserver hosts only; a mail-

	only host does not need it. It defaults to loopback port 5353 — binding a port does not publish a zone.
`zephyrab-installer`	The `zephyrab` CLI and the Ansible deployment payload under `/usr/share/zephyrab/ansible`. Depends on `ansible-core`.
`zephyrab-web`	The three web surfaces as self-contained HTML files under `/usr/share/zephyrab/www`: `mail.html` (webmail), `mail-ai.html` (the writing-assistant frame), `console.html` (the admin console). This package installs files, not a web server; a README beside them says how to front each one.
`zephyrab`	The metapackage. Pulls core + tools + installer + web at pinned = versions so the set stays level. `zephyrab-dns` is deliberately not in the set.

If you front the webmail yourself: `img-src` must allow `https:`

The installed presets get this right and you can skip this. It matters only if you serve `mail.html` behind a proxy you configured, with a Content-Security-Policy of your own.

The webmail shows a message's HTML inside a sealed frame that carries its own policy — `default-src 'none'`, and `img-src data:` until the reader clicks **Show images**, rebuilt for every message. That frame **inherits the surrounding page's policy and intersects it with its own**, so whatever the frame permits, the page's header can only take away.

The consequence is specific: if the page's header says `img-src 'self' data:`, the reader's **Show images** button does nothing at all — every remote image is refused however the frame's own policy reads. Nothing errors, nothing is logged, and the button still appears. Set the page's policy to:

```
img-src 'self' data: https:
```

That is not a relaxation of the frame. What enforces the reader's per-message choice is the frame's own policy, which is unchanged. What the page gives up is a blanket ban on remote images in the *surrounding* document — where no message markup is ever inserted. Leave `script-src`, `connect-src` and `form-action` alone; those are what limit what an injection on that page could execute or send.

The reference Caddyfile in the deployment payload carries the working header. A useful check after any proxy change: open a message with a remote image, click **Show images**, and confirm the image appears — the failure is silent and this is the only thing that shows it.

Services ship disabled

Both units (`zephyrd.service`, `zephyr-dnsd.service`) are installed disabled and are never started by the package scripts. Reason: the packages install a configuration *skeleton* (`.example` files), never a live configuration. A mail server that starts with an unconfigured environment binds public ports with defaults nobody reviewed. Configuring and enabling the services is `zephyrab install`'s job.

Important paths

Path	What lives there
`/etc/zephyrab/`	All configuration and keys. Mode 2770 root:zephyr — group-readable so services drop privileges, setgid so files created inside inherit the group. Holds `zephyrd.env`, `admin.env`, `minio.env`, `dkim.pem`, `tls.crt`/`tls.key`, ACME account/state, and `license.json` if licensed. Read it as root; a non-root `ls` reports files missing that exist.
`/usr/share/zephyrab/ansible/`	The embedded deployment payload.
`/usr/share/zephyrab/www/`	The web surfaces. dpkg owns these files; do not edit them in place — an upgrade replaces them.
`/var/lib/zephyrab/`	The default data root: `minio/` (blob store) and `backup/`.
`/var/lib/foundationdb/`	FoundationDB's own data directory.

Remove vs purge: `apt-get remove zephyrab-core` keeps `/etc/zephyrab`. `dpkg -P` (purge) removes what the package installed — the `.example` skeletons — but never deletes files you created there. That directory can hold the DNS master key, and a purge that deleted it would destroy something no reinstall can recreate.

Ports

Port	Protocol	Notes
25	SMTP	Mail from other servers. STARTTLS offered when TLS is configured.
465	SMTP submission, implicit TLS	For your users' mail clients. Only bound when TLS is configured.
587	SMTP submission, STARTTLS	For your users' mail clients.
143	IMAP, STARTTLS	Cleartext IMAP stays on

		loopback by default (RFC 8314: the TLS variants are the public ones).
993	IMAP, implicit TLS	The public IMAP port.
110	POP3	Loopback by default.
995	POP3, implicit TLS	The public POP3 port.
4190	ManageSieve, implicit TLS	Filter management.
443	HTTPS	The web front (Caddy in the installed presets): webmail, JMAP, DAV, the admin API and console. zephyrd's own HTTP listener is `127.0.0.1:8080` behind it.
8081	HTTP	The link-attachment download listener, if enabled — fronted by a **separate** origin (section 6, Domains → link attachments; section 9).
9000	HTTP	MinIO. Loopback only.
9464	HTTP	Prometheus metrics (`/metrics`). Keep it off the public internet.
53	DNS	Only on hosts running `zephyr-dnsd`, and only after you deliberately bind it — the daemon defaults to `127.0.0.1:5353`.

4. First hour

You have run `sudo zephyrab install` and `zephyrab doctor` passes. Do these next, in order.

Save the two things that cannot be recovered

- The **admin mailbox password** was printed once at the end of the install. It is stored only as a hash; nobody can read it back, including the installer. If it is lost, re-mint it with `zephyr-passwd admin@example.com`.
- If you later enable hosted DNS with DNSSEC, the **DNS master key** becomes the one secret whose loss is unrecoverable (section 7). Back it up the day you create it.

Sign in to the console

The console is `console.html` from the `zephyrab-web` package. Serve it on your **admin origin** — never on the mail hostname. The README beside the file shows the proxy shape; the rule is a security property: an admin control plane on the same origin users type their mail password into is a phishing surface nobody needs.

The console signs in with a **person's email address and password**, not a machine secret. What you can see and change is decided by the roles delegated to you; the session lives in the tab and a reload signs you out.

On a fresh install your admin mailbox has no roles yet. Grant yourself platform authority by adding one line to `/etc/zephyrab/zephyrd.env` and restarting:

```
ZEPHYR_ADMIN_PLATFORM_ACCOUNTS=admin@example.com
```

Accounts named there are treated as platform-wide administrators when they sign in. Everyone else gets authority only from stored delegations (section 5).

Create a tenant, a domain, and an account

In the console:

1. **Tenants → + New tenant.** A tenant is an organization: the ownership boundary for domains, accounts and delegation. (The install already created a `Platform` tenant holding your own domain; create separate tenants for separate customers.)
2. **Domains → + Add domain.** The domain is created in **pending verification**. Publish the records it asks for, then use **Verify**.
3. **Accounts → + Create account.** Local part, domain, optional display name. Set the password with the **Password** action (minimum 12 characters; a ticket reference is required for the audit log), or let the user sign up with an invite code.

Publish the DNS records

Open the domain's **Configure** drawer. The **Expected DNS records** panel lists every record the platform expects — MX, SPF, DKIM, DMARC and more — with a live `published` / `missing` / `not checked` status per record. Publish them at whoever answers the domain's NS records, then press **Verify**.

From the shell, the same loop is:

```
zephyrab dns-check --domain example.com --wait
```

It exits 0 once every probed record is live in public DNS. Three common mistakes it warns about up front: the Name column is fully qualified and some providers auto-append the domain (a doubled name resolves to nothing and looks exactly like propagation lag); DKIM TXT values over 255 characters must be published as quoted chunks; and `not checked` rows (SRV, TLSA, the MTA-STS policy host) still need publishing even though the probe does not examine them.

Mint invite codes

Signup is invite-gated (section 10 explains why open registration is refused). The Solo install minted one batch; mint more with:

```
zephyr-invite new --domain example.com --uses 5 --days 30
```

The code is printed once and is not recoverable afterwards — only its hash is stored, so reading the database cannot hand out working invites. `zephyr-invite list` shows records without codes, by construction. Hand a code to each person you want on the platform; they sign up in webmail.

Send and receive a test message

1. From an outside mailbox (any provider), send a message to `admin@example.com`. Watch it arrive in webmail (`/mail` on the mail origin) or over IMAP.
2. Send a reply outward, or use `zephyr-sendmail` on the host to submit a test message through the real signing and queue path.
3. `zephyrab doctor` again: accepted and delivered counters present, queue empty, parser panics zero.

The deliverability truth

Read this before you promise anyone a working mail system.

Receiving works immediately once the MX record is live and port 25 is open.

Sending reputation is earned, not installed. A new IP address and a new domain start in the spam folder at the big providers — even with SPF, DKIM and DMARC all passing, a valid PTR, and a perfect configuration score. This is how those providers treat all new senders, and no software setting changes it. The reference deployment scored 10/10 on external configuration tests and still spent weeks in Gmail's and Yahoo's spam folders.

The first-weeks checklist:

- Ask your provider for a **PTR record** matching your mail hostname, if preflight warned about it.
- Send real mail at low volume, consistently. Volume and time are the levers.
- Register with the **feedback loops and reputation portals** (Microsoft SNDS/JMRP, Yahoo CFL, Google Postmaster Tools) — enrolment is manual, at each provider's portal (section 8).
- Keep DMARC at `p=none` until the aggregate reports are clean, then tighten to `p=quarantine`. Keep MTA-STS in `testing` mode until TLS reports show nothing would break, then `enforce` (sections 7 and 8).

5. Concepts

The object hierarchy

- **Platform** — one ZephyrAB deployment, whatever its size. Platform-level settings and switches apply to everything below.
- **Tenant** — an organization. Owns domains, accounts and groups. Isolation between tenants is structural: every stored object carries its tenant, and no query path crosses tenants.
- **Domain** — a mail domain (`example.com`). Belongs to exactly one tenant. Carries its own DNS mode, DKIM keys, encryption policy, and per-domain settings.
- **Account** — a mailbox with an address, credentials, quota, filters, calendars and address books. **Aliases** are extra names for the same mailbox.
- **Group** — a distribution group: one address that fans out to many mailboxes. A group is not a mailbox and holds no mail of its own.

Addresses are exclusive by construction: an address cannot be both a mailbox and a group, because both contend for the same index key inside one transaction.

Cells

A **cell** is a complete, self-contained mail store: its own FoundationDB cluster, its own blob store, its own queue and frontends. A single-machine install is one cell. **A cell is a scaling unit, not a redundancy mechanism.** A domain lives in exactly one cell; if that cell is down, that domain's mail is down whether or not other cells exist. What a second cell buys: capacity past roughly a million mailboxes, blast-radius containment (a bad deploy or a hot tenant hits one cell's population), and tenant residency. Section 19 covers sizing a cell and how its hosts fit together; section 12 covers running more than one.

Admin scopes and delegations

The admin API and console use ten **scopes** — role labels that gate operations:

Scope	What it reaches
`platform:admin`	Everything. Implies every other scope.
`reseller:admin`	Reseller-level administration.
`tenant:admin`	Everything inside one tenant: domains, accounts, groups, settings, further delegations.
`domain:admin`	One domain: its DNS, encryption policy, settings, and the accounts on it.
`helpdesk:read`	Read accounts and groups. Cannot change anything.
`helpdesk:write`	Reset passwords and change account status. Cannot create or delete.

`compliance:read`	Read for audit and e-discovery purposes.
`compliance:write`	Retention and legal-hold actions.
`mailops:read`	Queue, alerts and deliverability — operational visibility, no mailbox content.
`mailops:write`	Act on the queue: retry, cancel, bounce, pause.

A **delegation** is a grant of a scope **bound to a subject** — a tenant or a domain. The pair is the whole point: `tenant:admin` bound to tenant A reaches tenant A and nothing else, and the scope cannot borrow reach from another grant. You cannot grant a scope you do not hold yourself. Cross-subject denials answer exactly like "not found", so a delegated admin cannot probe what else exists.

Grants are immutable. There is no edit. Changing one means creating the replacement grant and then revoking the old one — in that order, so there is no window with no access. The console's "Change permissions" dialog does exactly this and says so. Revocation takes effect on the **next request**, not when a token expires: subjects are re-read on every request.

Platform-wide authority for a *person* comes from `ZEPHYR_ADMIN_PLATFORM_ACCOUNTS` in the environment; machine callers use the OAuth2 client credential in `/etc/zephyrab/admin.env`, whose scopes come from `ZEPHYR_ADMIN_CLIENT_SCOPES` (default: full `platform:admin`).

Admin API tokens

Scopes say what you may do; a **token** is how you hold that authority for an hour. A machine caller exchanges the client credential at the token endpoint for an opaque bearer token — 256 random bits, stored by its SHA-256 so there is nothing to compare and nothing for timing to leak. Tokens live one hour and there is no refresh grant; getting another is one form POST.

Tokens are stored in the cell's FoundationDB, so every node honours every token. This matters the moment there is more than one node:

- **You can put the admin API behind a load balancer.** With a per-process store, roughly (N−1)/N of requests would meet a node that never issued that token, and the 401 is byte-identical to the one a forged token gets — deliberately, so nothing leaks — which means a client could not tell "wrong node" from "bad credential" and could not retry correctly.
- **Tokens survive a restart**, which is convenient and took away a revocation that used to come for free. It is given back explicitly: a machine token is bound to the credential that minted it, so changing `ZEPHYR_ADMIN_CLIENT_SECRET` stops every token minted against the old one, fleet-wide, at once. That is the revocation to reach for if a secret leaks.
- **Watch the startup line.** A node with the cluster store logs `admin token store: every node in this cell honours every issued token`. A node that kept the

in-process store logs a **warning** naming the consequence. Silence there is the failure mode: such a node works perfectly on its own and starts producing indistinguishable 401s the moment anything balances across it. Deploy the same build everywhere before putting a balancer in front.

- A store this node cannot **read** fails closed with 503, never 401 — an unreachable database and a bad token are different facts and must not read the same to a caller holding a good credential. A node with no store configured keeps the per-process behaviour, which fails safe: a node-local store can only ever honour *fewer* tokens, never more.

Scope masks and subjects are treated differently on purpose: the **scope mask is pinned into the token** (re-reading it would silently re-widen a token somebody deliberately narrowed at issue time), while **subjects are read live on every request** (a delegation is somebody else's decision and revoking it must not wait for a token to expire).

Known gap — the token endpoint's brute-force brake is wrong behind a proxy, in both directions. `ZEPHYR_ADMIN_TOKEN_FAIL_MAX` counts failed token requests per source address, and the source is the TCP peer, never a forwarded header (a client-supplied header would let an attacker both evade the limit and poison another operator's bucket). Behind a reverse proxy every request arrives from the proxy, so all operators share one bucket — ten failures from one script with a stale secret refuse token issuance for everyone behind that proxy for the window, and because the refusal is deliberately indistinguishable from a bad credential, the symptom is "my correct credential stopped working" with only the audit line saying otherwise. And the counter is still per process, so N nodes give a guesser N budgets. Neither is caused by the cluster store, but both stop being theoretical the moment you balance the admin API. Until it is fixed, the mitigation is deployment-shaped: **keep the admin API off the public proxy** (the reference deployment does not proxy it on the mail edge at all), or accept the shared bucket knowingly.

Settings layers

Many behaviors are **settings** resolved through three layers: **platform → domain → account**. The environment is the platform layer; a stored value overrides it; clearing a stored value falls back to the layer above. Every setting is one of two kinds, and the kind decides how the layers combine:

- **Protective** settings (safety booleans, e.g. `av.enabled`, `mailauth.enabled`, `mailauth.enforce`, `security.at_rest_encryption_required`): the layers form a **union**. Any layer can turn the protection ON; no layer can turn another layer's ON off. A domain can be stricter than the platform, never weaker. The reasoning: a wrongly-ON protection refuses loudly and gets fixed; a wrongly-OFF one stores someone's mail unprotected, silently and permanently.
- **Operational** settings (values and preferences, e.g. `spam.threshold`, `links.ttl_days`, `smtp.subaddress`): the **most specific layer wins** — account over domain over platform — except that a domain admin can **lock** a setting, and a locked domain value overrides the account layer, including values stored before the lock arrived.

Out-of-range values are refused, never clamped: a clamped setting is one an administrator believes is in force and is not.

Two timing facts to know: a platform switch change lands on every frontend within about 10 seconds (`ZEPHYR_SWITCH_REFRESH_SECS`); a domain-level change lands within about 60 seconds (`ZEPHYR_DOMAIN_SWITCH_REFRESH_SECS`). Nothing is instant, and the console says when each node last looked.

License tiers

ZephyrAB is open-core. The principle: **running your own mail is free; running mail as a business is licensed.**

- **Community** (no license file, forever): the entire mail platform — SMTP, submission, IMAP, POP3, JMAP, CalDAV/CardDAV, webmail, console, Sieve, groups, aliases — plus every protection (spam classifier, antivirus, RBL, greylisting, mail authentication, rate limits), at-rest encryption, backups and restore, migration import, and GDPR export. **One mail domain, one cell, and no mailbox limit** — the ceilings describe who the deployment serves rather than how big it is.
- **Licensed** adds the commercial features: **multi-cell** (joins, routing, secondary MX tiers, cross-cell tenant move), **delegated RBAC administration** (creating subject-bound admin grants), **commerce** (plans, usage metering, bulk provisioning), and **SSO** (LDAP / OAuth).

Nothing on the protection or data-safety rows is ever gated: gating backups, encryption or export would monetize user harm. Section 13 has the full rules, including what a lapsed license refuses and the one rule that outranks everything: an expired license never stops mail.

6. The web console, page by page

The console lives at `console.html`, served on the admin origin, talking to `/admin/api`. Sign in with your email address and password. Pages your roles cannot reach are dimmed, not hidden, with a notice naming the scopes they need. One convention runs through every page: when the console cannot read something, it shows a `cannot look` pill and says so — "could not read" is never rendered as "empty", because those are different facts.

Dashboard

Live platform state: tenant and domain counts, DKIM published, DMARC enforced, and a **Mail authentication across domains** panel (MX/SPF/DKIM/DMARC coverage), resolved live against public DNS. All figures come from the API; operations without a control-plane endpoint say so on their own pages.

Tenants

Create, edit, and deprovision tenants. **Deprovision is refused while a tenant still owns a domain or a mailbox**, and there is deliberately no cascade: a tenant is every mailbox a customer has, and one keystroke between an operator and that is not a feature. Accounts go first, one at a time, so each mailbox is purged in the ordering that settles blob refcounts and quota together. Suspending the tenant (from Edit) is the reversible step, and it is the one to take first. The deprovision dialog reads the counts before offering the confirmation, and an unreadable count is presented as "unknown", not as none.

Domains

The list shows each domain's tenant, status, live auth-record chips (MX, SPF, DKIM, DMARC, MTA-STS — failing chips are marked), and DKIM state. **Verify** re-checks a domain against public DNS after you publish records. **Configure** opens a drawer with six panels:

- **This domain** — status and its source. Only `disabled` can be set by hand; the other statuses are observations derived from DNS, because an operator who could assert "verified" would make this page report a healthy domain whose mail is failing. **Disable this domain…** makes RCPT answer a permanent `550 5.2.1` (senders are told the recipient is not accepting mail and will not retry); nothing is deleted, and it is reversible from the same panel.
- **Expected DNS records** — the generated record list with per-record `published` / `missing` / `not checked` status. The same list whether DNS is hosted here or elsewhere.
- **DNS** — the authority selector (`internal` = ZephyrAB is authoritative and generates every mail record; `external` = DNS lives elsewhere and we only verify it) and, for hosted domains, the custom-record editor. **Custom records are additive only**: a record that would collide with a mail record ZephyrAB manages — a second MX, a second SPF TXT at the apex — is **refused, naming the collision**, because accepting it would break your mail days later with nothing in the DNS looking wrong.
- **Encryption at rest** — the domain's policy (`optional` or `required`), with a readiness survey: how many accounts hold a usable key, and how many **would have mail refused** if the policy became required. Arming `required` needs a checkbox acknowledging exactly that consequence, and a truncated survey is reported as a floor, not a count. See section 10.
- **Settings** — the per-domain settings table (key, resolved value, which layer it came from, `protective` and `locked` pills), with **Set** and **Clear**. Values outside the allowed range are refused, never clamped.
- **Retire this domain** — removal, refused while any mailbox still exists on it. Key material is kept by default so the domain can be re-added without breaking DNSSEC; a separate checkbox destroys the keys, with a warning about DS records still published at the parent.

Rotate the DKIM key… starts a rotation and says plainly: it does not finish one. A new selector is minted and published, does not sign yet, begins signing only after a full hold, and

the old selector stays published until it is retired after another hold. Pressing it twice reports the rotation in flight rather than starting a second. Section 7 explains the phases.

Accounts

Per-tenant mailbox management. Row actions:

- **Details** — display name, quota (GB, 0 = unlimited), and the alias list (the whole set, replacing what is stored — emptying the box removes every alias).
- **Password** — set a new password (minimum 12 characters). A ticket reference is required for the audit log.
- **Filters** — the account's Sieve script. Saved scripts are **parsed before they are stored** — a script that does not parse is refused rather than silently ignored at delivery. Emptying the box removes the filter.
- **Sharing** — the shared-mailbox panel: what this account has shared out (mailbox, grantee, RFC 4314 rights letters) and what has been shared with it. Rights are the same letters IMAP's SETACL uses, so an ACL set here and one set by a mail client are the same thing. Sharing never crosses a tenant — both halves of a grant live in one tenant's keyspace, so a cross-tenant share is not something the system can express. Revocation is immediate: rights are re-read on every command, so an open IMAP session loses access at its next command. A message added by someone you share with counts against the **owner's** quota — which is why the `i` (insert) right is one to grant deliberately.
- **Status** — `active`, `suspended` or `maintenance`.
- **Restore…** — rebuilds the mailbox **as it stood at a moment in the past, from the cell's backup** — not from Trash and not from the user's own deleted-message window. The restored tree is appended alongside the live one under a folder of its own (default `RESTORED`); nothing live is moved or overwritten, and the user copies back what they want. Runs as a job; needs a ticket reference, because a restore reads somebody's deleted mail into a place they can see it.
- **Delete…** — purges every message, filter, calendar, address book, credential and address (aliases released back to the pool). Requires typing the address to confirm. Runs as a job, one transaction per message, in the ordering that settles blob refcounts, quota and tombstones together.
- ⇪ **Import many…** — bulk account creation, up to 10,000 rows (`localPart, display name, quota GB`). There is deliberately no password column: the rows sit in the job record until the job finishes, and ten thousand plaintext passwords at rest is a credential store nobody decided to build. **Dry run** is on by default the first time.

Groups

Distribution groups: one address that reaches many mailboxes. **An owner is required, and the requirement is the design**: mail arriving with a null sender — a bounce, or any other automatic message — goes to the **owner**, never expanded to the members, because fanning a delivery failure out to every member is the storm a list must not create. The owner is also the envelope sender on copies to members outside the platform. Copies carry

a `List-Id` header so clients and auto-responders can tell it is a list. Posting is `open` or `members only`. Deleting a group touches nobody's mailbox — a group holds no mail of its own.

Delegations

Create, replace and revoke scoped admin grants (see section 5 for the model). The grant form takes the admin's email address (it must match how they sign in), the scope, and what it is bound to — the tenant, or a single domain. `platform:admin` is deliberately absent from the list: platform-wide authority is configured on the server, not delegated through the console. **Change permissions** replaces a grant — a new one is created and the old one revoked, and if the revoke half fails the console tells you which grant is still live so you can revoke it by hand. **Revoke** takes effect at once.

Health and Security

The operations overview for this node:

- **Firing alerts** — relayed live from Alertmanager (configure `ZEPHYR_ALERTMANAGER_URL`). The empty state reads: "Nothing firing. (The relay REFUSES when it cannot reach Alertmanager, so this quiet is a verified quiet.)" — an unreachable Alertmanager is shown as `cannot look`, never as "no alerts".
- **License** — state, licensee, days left, used-of-limit rows and entitlement pills (section 13). The card's caption states the rule: limits gate provisioning, entitlements gate a few admin writes — mail flow, mailbox access and reads never consult the license.
- **Feature pills** — what is switched on: Inbound RBL, Mail antivirus, Outbound caps, SPF/DKIM/DMARC, Hygiene, At-rest encryption, Body search index, CalDAV/CardDAV, Link attachments, ACME TLS, OTel traces, Signup.
- **Live counters (this node, since restart)** — accepted/delivered, delivery failures, RBL and AV outcomes, **parser panics (alert on any)**, encryption refusals, push drops, blob-GC pending/due. Red means a number that should be zero is not. Counters reset on restart; the durable series live in Prometheus.

Mail Queue

The outbound queue: depth, in-flight count, and the entry table (queue id, sender, next hop, recipients, status, attempts, next retry). Terminal entries never appear — a delivered, bounced or cancelled entry has left the queue, which is the platform's core invariant doing its job. The page banner states the contract: "Retry and cancel take the same path as the `zephyr-queue-ops` CLI, so a cancel settles the entry exactly as retry-exhaustion would — ledger flip and the sender's bounce in one transaction. Cancelling needs a reason, and the reason reaches the sender. Both need the `mailops:write` scope, which nothing else uses."

- **Retry now** brings a deferred entry forward **without resetting its attempts** — the bounce clock belongs to the sender, and resetting attempts would move the moment they learn of a failure hours into the future.

- **Cancel…** does not quietly drop the message. It settles it as **bounced**, exactly as running out of retries would, and the sender receives a delivery-failure notification carrying the reason you type. A reason is required — the sender reads it. Cancel is refused while a worker holds a live lease on the entry, because bouncing under an in-progress delivery risks a message that was both delivered and reported as failed.
- **Pause outbound delivery…** is **fleet-wide**: it stops every node claiming new deliveries, not just this one. Queued messages stay queued and nothing is lost; senders simply wait. Bounce notifications for failures that already happened keep being delivered — pausing must not stop telling senders about mail that has already failed. A reason is required; whoever finds it paused will read it.
- **Trace a message** — by queue id or accept id. An RFC 5322 Message-ID cannot be resolved, and the refusal says why: the server keeps no index from a sender-chosen identifier to a delivery.

Jobs

Asynchronous work: bulk provisioning, mailbox restores, DKIM rotation, tenant deprovisioning, account purges. A worker holds a **lease**, not a lock: a job whose worker dies is reclaimed and run again, which is why every job type is written to be safe to run twice, and why attempts are shown beside the status. After the last attempt a job fails as **poison** rather than being reclaimed forever. The listing is scoped to what your roles cover.

Deliverability

Three panels (section 8 has the operational detail):

- **Blocklist self-check** — are *we* listed anywhere (Spamhaus zen for the sending IP, DBL for the domains), as recorded by the daily `zephyr-blocklist-watch` run. A never-recorded or stale state is flagged, with the wording "this is not the same as clean" — a check that never ran proves nothing.
- **DMARC aggregate reports** — what the world's receivers say about mail claiming to be from your domains, with a failing-sources table. The note under it is the decision rule: "A failing source is either a SPOOF being caught (the policy working) or a legitimate FORWARDER being discarded — the p=quarantine exit question."
- **Outbound abuse gate** — cap refusals and auto-locked accounts since this node's restart.

Plans

Class-of-service bundles for commerce. **A plan version is written once and never rewritten**: a closed billing period was scored against a version, so overwriting one would silently re-price a settled month. "Publish new version" appends; the previous version stays readable at its own number. Deleting is refused for any plan that has **ever** had a subscriber — retire it instead. ZephyrAB records allowances and rates and **computes no amount of money**: rounding, currency minor units, proration and tax are a billing system's decisions, and a wrong one here would be a wrong invoice.

Usage

Per-tenant metering for a period (`YYYY-MM`): mailboxes (peak), storage (peak, logical, before deduplication), inbound and outbound traffic, domains. Two flags govern whether you may bill from it, and the page shows both: **`final`** (false = the period is open or not yet frozen — a running total; the pill reads "NOT FINAL — do not invoice") and **`complete`** (false = a daily sample did not see every account; the stock figures are a floor). **Incomplete samples are never billable.** Physical storage is deliberately `not reported`: blobs are deduplicated across the whole platform, so a body shared between two tenants belongs to neither exclusively, and a fabricated number on an invoice is worse than an absent one. Section 5's stock/flow distinction and section 15's `zephyr-usage*` tools complete the picture.

Platform Settings

Two halves:

- **Protocol switches** — the runtime switches (RBL, antivirus, mail-auth enforcement, spam, recipient verification, subaddressing and the rest), each showing the value in force, its source ("set by operator" or "from the environment"), and Turn on / Turn off / Set / Clear actions. The banner is the contract: these change what this server does for every tenant, so they need `platform:admin`; **a change takes effect on every frontend within one refresh interval — it is not instant** — and the panel shows when each node last looked. Turning a switch **off** is always possible; turning one **on** needs the thing it drives to be configured (a blocklist needs a dedicated resolver, malware scanning needs a scanner address), and the panel says so rather than accepting a switch that would silently do nothing.
- **Mail-authentication posture** — per-domain MX/SPF/DKIM/DMARC/MTA-STS status resolved live from public DNS, plus each domain's at-rest encryption policy.

7. Domains and DNS

The records a domain needs

Every domain page (and `zephyrab dns-check`) generates the exact set, but the four that make mail work are:

Record	Example	What it does
MX	`example.com. MX 10 mail.example.com.`	Tells the world where to deliver mail for the domain.
SPF	`example.com. TXT "v=spf1 mx -all"`	Names the hosts allowed to send mail claiming to be from the domain. `mx` follows the MX records, so it covers your mail hosts

		automatically.
DKIM	`zab1._domainkey.example.com. TXT "v=DKIM1; k=rsa; p=..."`	Publishes the public half of the signing key. Values over 255 characters must be published as quoted chunks.
DMARC	`_dmarc.example.com. TXT "v=DMARC1; p=none; rua=mailto:postmaster@example.com"`	Tells receivers what to do with mail that fails SPF/DKIM alignment, and where to send aggregate reports. Start at `p=none`.

Additional records the generator lists (SRV autodiscovery, MTA-STS, TLS-RPT, TLSA) are not probed by Verify; publish them anyway.

Hosted DNS vs external DNS

Each domain has a DNS authority mode, set in the console's domain drawer:

- **external** — DNS lives at your registrar or DNS provider. ZephyrAB generates the record list and verifies it against public DNS, but publishes nothing. It still mints the domain a DKIM key, so the record list can tell you what to publish; **publish that record and the domain signs as itself** (see below).
- **internal** — ZephyrAB is authoritative. `zephyr-dnsd` serves the zone, generates every mail record itself, and mints a **per-domain DKIM key** (selector `zd1`) on first serve, publishing its TXT automatically. Nothing further to do.

Which key signs a domain's mail

A message is signed as the domain in its **From** line, using that domain's own key, whenever the platform holds one and has seen it published. Otherwise it falls back to the deployment key (`ZEPHYR_DKIM_KEY`, selector `ZEPHYR_DKIM_SELECTOR`).

This matters as soon as you host more than one domain. DMARC checks that the signing domain lines up with the From address, so a message from one domain signed by another **fails alignment** — and a receiver told to quarantine or reject on failure will act on it. With a single deployment key, every domain but one was mis-signed.

For an external domain the sequence is:

1. `zephyrab dns-check --domain agency.example` — asks the platform what to publish. The first call for a domain with no key generates one, so the DKIM line is there to copy.
2. Publish that record wherever the domain's NS records point.
3. Nothing else. The platform re-checks the domain's DNS on its own, and once it sees its own key there it starts signing as that domain. Re-run `dns-check` (or `--wait`) to watch it flip.

Signing is gated on the record being visible in that domain's DNS, not on your say-so, and that is the safety property. A signature whose key nobody can look up fails at *every* receiver — worse than no signature at all — so until the record resolves, mail goes out under the deployment key exactly as before. If the record is later removed, the platform notices, logs it, and falls back rather than signing into a hole.

The From domain is also checked against the account that is sending, so putting somebody else's domain in a From line does not borrow their signature: that message is signed with the deployment key.

Custom records on a hosted domain are **additive only**. A record that collides with a managed mail record is refused with the collision named — a customer adding their own MX or a second SPF TXT would break their mail days later, with nothing in the zone looking wrong.

Run the nameserver

`zephyr-dnsd` (package `zephyrab-dns`) has two modes:

- **Direct mode** — the daemon reads the domain registry from FoundationDB and generates zones itself.
- **Feed mode** — set `ZEPHYR_DNS_FEED_HOST`, and the daemon pulls signed zones from the control plane (`zephyrd`'s zone feed, `ZEPHYR_ZONE_FEED_ADDR`) over mutual TLS, and **holds no database session at all**. This is the production shape: a nameserver answers unauthenticated UDP from the whole internet, and the only safe amount of database access for it is none.

Two properties worth knowing before you rely on it:

- **Binding a port does not publish a zone.** The daemon defaults to `127.0.0.1:5353`. The world reaches it only when NS records delegate a zone to it — a separate, deliberate act. Response-rate limiting is on by default (`ZEPHYR_DNS_RRL`, only the literal `off` disables it).
- **Stale beats dark.** The zone feed's contract is install-never-clear: a zone missing from a feed pull means *keep what you have*, never *stop serving*. A nameserver cut off from the control plane keeps answering with authoritative, signed, slightly stale zones and logs no errors — safe, and silent. The signal that it is cut off is **SOA serial divergence between your nameservers**, not query failures. Watch serials, not just reachability.

DNSSEC

Set `ZEPHYR_DNSSEC=sign` (direct mode; in feed mode the control plane signs) and give it `ZEPHYR_DNS_MASTER_KEY_FILE`. ZephyrAB then signs every hosted zone: per-zone KSK and ZSK, NSEC3 denial of existence, automatic serial management.

Key custody: zone keys live in FoundationDB, sealed under an operator-held master key supplied from a file at startup — a file, not an environment variable, and never stored beside the data it protects. Two consequences:

- **BACK THE MASTER KEY UP.** Without it the stored zone keys are unreadable. Every other loss on this platform is recoverable; this one is not. Once a DS record is published at a parent zone, a lost master key means a registrar visit and a validation outage.
- A stored key record that will not decrypt is a **hard error, never a regeneration**. Regenerating on decrypt failure would look like graceful recovery and would be an accidental key rollover — with a DS at the parent, that takes the domain dark for every validating resolver.

The ZSK (zone-signing key) rolls automatically in pre-publish → switch → retire phases. The KSK (the key the parent's DS record commits to) rolls only by explicit operator steps, because the parent is involved:

```
zephyr-dns-keys show     --domain example.com            # state + the
DS record to publish
zephyr-dns-keys roll-ksk-start  --domain example.com --master-key-file
<path>
zephyr-dns-keys ds-check --domain example.com [--mark]   # did the DS
appear at the parent?
zephyr-dns-keys roll-ksk-switch --domain example.com     # refuses
until ds-check verified it
```

`roll-ksk-switch` refuses to run until the incoming key's DS has been verified at the parent (`--unverified` exists for parents that hold no DS, and you must spell it out). `ds-check` treats "could not look" as an error, never as "not published" — a switch gated on that misreading would run early.

Rotate a DKIM key

DKIM rotation is three phases, not a swap, because a receiver verifies a signature by fetching `{selector}._domainkey.{domain}` — possibly hours after the mail was sent. A selector must be resolvable **before** it signs and must **stay** resolvable after it stops.

Phase	Incoming key	Outgoing key
PrePublish	published in DNS, not signing	signing
(hold)		
Switch	signing	still published, not signing
(hold)		
Retire	signing	removed from DNS

The hold (default 7200 seconds) must exceed the DKIM TXT's TTL (3600 in generated zones). The default key lifetime is 90 days. **The gates refuse early runs**: ticking during a hold is refused with the time it is holding until — proven behavior, not an assumption. Selectors are never reused (`zd1` → `zd2` → `zd3`), so a cached old TXT can never verify new mail against an old key.

Start a rotation from the console (Domains → Configure → Rotate the DKIM key…) or the shell:

```
zephyr-dkim-rotate show --domain example.com
zephyr-dkim-rotate tick --domain example.com --master-key-file <path>
```

`show` needs no master key — public halves are stored in the clear so tooling works without unsealing anything.

MTA-STS in one paragraph

MTA-STS (RFC 8461) lets you tell sending servers "always use verified TLS to my MX hosts". It is two artifacts: a TXT record at `_mta-sts.example.com` and a policy file served over HTTPS at `https://mta-sts.example.com/.well-known/mta-sts.txt` listing your MX hosts and a mode. Run in `testing` mode first: senders report problems but still deliver. Flip to `enforce` only after your TLS reports are clean — an enforcing sender **filters out any MX host the policy does not list** and refuses to deliver when it cannot fetch or validate the policy, so an incomplete MX list or a broken policy host becomes lost mail. On the outbound side, `ZEPHYR_MTA_STS=off|testing|enforce` controls whether ZephyrAB honours *other* domains' policies when sending.

TLS-RPT in one paragraph

TLS-RPT (RFC 8460) is the reporting channel for MTA-STS and DANE: a TXT record at `_smtp._tls.example.com` asking every sending server to mail you a daily report when it could not negotiate TLS to you. This is the one deliverability signal that is invisible from your side — a sender that cannot verify your certificate defers or downgrades entirely at their end, and without these reports the first you hear is a user saying a partner's mail stopped arriving. `zephyr-tlsrpt-ingest` (daily timer) reads the reports out of the postmaster mailbox and stores a trend; `--report` prints it grouped by failure type and MX host, which is the pair that identifies a fix. A clean report is the normal case, and it is the evidence you need before flipping MTA-STS to `enforce`.

DANE in one paragraph

DANE (RFC 7672) publishes a TLSA record — a hash of your mail server's public key — in DNS, protected by DNSSEC, so a sending server can verify your certificate without trusting any certificate authority. Its sharp edge: a validating sender **fails rather than falls back** when the TLSA record goes stale, so a certificate re-key without a DNS update is silently lost mail from every DANE-validating sender. Publish TLSA only when the key underneath it is stable — ZephyrAB's built-in ACME deliberately **reuses its private key across renewals** so the published TLSA stays true — and keep a drift check. On the outbound side, `ZEPHYR_DANE=off|testing|enforce` controls whether ZephyrAB validates other domains' TLSA records when sending; it ships off by default (the shipped configuration keeps it off pending an upstream DNS-resolver fix).

Get a real certificate

The Solo install seeds a self-signed certificate. Replace it either by fronting the TLS ports yourself, or with zephyrd's built-in ACME client (`ZEPHYR_ACME=on`): DNS-01 challenges, so it needs no port 80 or 443, publishes only `_acme-challenge` TXT records, renews at 60 days of age (`ZEPHYR_ACME_RENEW_DAYS`), and hot-swaps the renewed certificate into every TLS listener with no restart. Two DNS backends exist for the challenge (`ZEPHYR_ACME_DNS=godaddy|zephyr`); the `zephyr` backend writes the challenge through the platform's own hosted DNS and requires `ZEPHYR_ACME_DNS_CHECK` — a list of your own nameservers to certify propagation against before asking the CA to validate. The key-reuse behavior requires the key in PKCS#8 format; a non-PKCS#8 key is refused with conversion guidance rather than silently replaced, because a fresh key changes the public-key hash and breaks a published TLSA record.

8. Deliverability operations

The acronyms, one sentence each

- **SPF** — a DNS record naming which hosts may send mail for your domain.
- **DKIM** — a cryptographic signature on each message, verified against a public key in your DNS.
- **DMARC** — a policy telling receivers what to do when SPF/DKIM fail to align with the From header, plus a reporting address.
- **PTR / FCrDNS** — the reverse-DNS name of your sending IP; forward-confirmed means the name resolves back to the same IP. Big receivers junk or refuse mail without it.
- **MTA-STS** — an HTTPS-published policy demanding verified TLS to your MX hosts (section 7).
- **TLS-RPT** — daily reports from senders about TLS failures reaching you (section 7).
- **DANE / TLSA** — DNSSEC-protected certificate pinning for SMTP (section 7).
- **RBL / DNSBL** — DNS-queried blocklists of IPs and domains known to send spam.
- **ARF** — the standard format for abuse complaints from feedback loops.

The monitoring that ships

Four ingest tools are packaged binaries, each idempotent (re-running over the same mailbox overwrites rather than double-counting) and read-only against the mailbox:

Tool	What it reads	Run it
`zephyr-blocklist-watch`	Checks **your own** sending IP against Spamhaus zen and your domains against the DBL, through your local resolver. Exit 1 = listed; exit 3 = **could not check, which**	Daily timer. Feeds the console's Blocklist self-check panel; a state that never recorded reads "this is not the same as clean".

	also fails — a self-check that silently checks nothing is worse than none.	
`zephyr-dmarc-ingest`	DMARC aggregate reports from the `rua` mailbox (`postmaster@`). `--report` prints the per-day pass/fail trend and a failing-sources table (source IP, DKIM/SPF verdicts, disposition).	Daily timer.
`zephyr-tlsrpt-ingest`	TLS reports from the postmaster mailbox. `--report` groups failures by (result, MX host).	Daily timer.
`zephyr-arf-ingest`	RFC 5965 feedback-loop complaints from the `abuse@` mailbox, attributed to the sending account by the recorded envelope — never by a header a stranger could forge. Deliberately **not** wired to enforcement: `abuse@` is unauthenticated, and feeding it to the auto-lock would let anyone lock any account by mail.	

The repository's `deploy/deliverability/` directory carries additional probe scripts with systemd units (a reputation watch, an inbox-placement probe, a daily digest) for deployments built from the repo; they are not in the packages.

The DMARC decision rule, printed by the ingest and the console alike: the exit condition for tightening `p=none` to `p=quarantine` is **not** "failures = 0" but "every failing source is identified and is not something we want delivered". A failing source you do not recognise is usually spoofing — an argument *for* tightening. A failing source that is a legitimate forwarder (mailing lists, `.forward` rules) breaks DKIM alignment by design, and `p=quarantine` would start discarding real mail from it.

ZephyrAB can also **send** DMARC aggregate reports to domains that publish a `rua` (`ZEPHYR_DMARC_REPORT=off|record|send` plus `ZEPHYR_DMARC_REPORT_FROM`), which is the polite half of the same ecosystem.

The role mailboxes are load-bearing

`postmaster@` and `abuse@` exist on every domain, are reserved (signup can never hand them out), and must be **read**: `abuse@` is where other operators and the feedback loops tell you one of your users is attacking them, and `postmaster@` is your DMARC and TLS-RPT reporting target. A reasonable service bar: a human acknowledges `abuse@` within 24 hours and `postmaster@` within 72. Machine-generated aggregate reports do not count as human work — the ingest tools parse those.

Reputation feeds — enrolment is manual

Register at each provider's portal; no software can do it for you:

- **Microsoft SNDS** (Smart Network Data Services) — per-IP reputation data, and **JMRP**, their complaint feedback loop (deliver complaints to `abuse@`).
- **Yahoo CFL** — their complaint feedback loop, keyed on your DKIM `d=` domain.
- **Google Postmaster Tools** — domain reputation dashboards; needs a DNS verification record.

Outbound protections

These protect your sending reputation from your own compromised or misbehaving accounts:

- **Per-account caps** — `ZEPHYR_OUTBOUND_MSGS_PER_HOUR` and `ZEPHYR_OUTBOUND_RCPTS_PER_DAY` (0 = off), also settable at platform and domain scope as `outbound.msgs_per_hour` / `outbound.rcpts_per_day`. A refused submission never consumes allowance.
- **Auto-lock** — `ZEPHYR_OUTBOUND_AUTOLOCK=on`. The trigger is deliberately not "hit the limit" (bursty humans do that): an account is locked when it has **burned an entire hourly allowance on refusals and is still hammering**. No human retry loop does that; a script does. The lock is the ordinary account lock, which every auth path refuses. The auto-lock setting itself is deliberately not overridable per account — an outbound limit an attacker who owns the mailbox can switch off is not a limit.
- **Content parity** — the virus gate runs on submission as well as ingress (section 9), so your platform does not emit what it would refuse.

Counters: `zephyr_outbound_cap_refused_total`, `zephyr_outbound_auto_locked_total` — both on the console's Deliverability page.

Route outbound through one host

Sending reputation is per IP address and takes months to build. A fleet where every node sends directly has as many cold reputations as nodes — and cold mail is accepted and filed to spam, which is invisible from the sending end.
`ZEPHYR_SMARTHOST=relay.example.com` routes **all** of a node's outbound mail through one warmed host. `ZEPHYR_SMARTHOST_TLS=verify` (the default) requires a valid certificate for the smarthost's own name and **defers** rather than downgrades when it

cannot get one; `opportunistic` is the other honest position, and there is deliberately no "require TLS but do not verify it" middle. A smarthost that is set but unusable refuses at startup rather than silently falling back to direct delivery — the fallback would send mail from exactly the address the smarthost exists to keep it off. Loops are guarded twice: a smarthost naming this node's own EHLO name is refused at startup, and past 25 `Received:` hops a message is bounced as a detected loop. Watch `zephyr_smarthost_loops_total` — any non-zero value means two nodes are relaying through each other. Known limit: the hop speaks port-25 SMTP without authentication, so a smarthost demanding SMTP AUTH on 587/465 cannot be used.

Warm-up honesty

There is no shortcut. Placement services measure; they do not warm. A new IP earns inbox placement through weeks of consistent, wanted mail at gradually rising volume — and manufacturing fake volume teaches receivers a traffic shape that then vanishes, which is worse than no history. Do not "warm up" hosts that do not send (secondary MX relays have no reputation to warm because they originate nothing). Concentrate sending on one address, watch the feedback loops, and give it time.

9. Mail filtering

The spam classifier

A learning Bayes classifier, trained by your users, per tenant. It is staged, and the staging is the design:

```
ZEPHYR_SPAM = off | train | on
```

A Bayes classifier with an empty corpus is not a weak filter — it is a confident and wrong one. Every ordinary word has only been seen in whichever class happened to be trained first. So there is a middle state:

Mode	Trains	Scores and stamps `X-Spam-Status`	Files into Junk
`off`	no	no	no
`train`	yes	yes	**no**
`on`	yes	yes	yes

The rollout order: deploy with it off (nothing changes); set `train` and let users' Junk filings accumulate; wait until `zephyr-spam-ops stats --tenant <id>` reports the corpus usable — **both classes** need `spam.min_corpus` examples (default 50 each; 500 spams and no ham is refused, deliberately); read what it *would* have done (`X-Spam-Status: Yes` on stored mail); only then set `on`, or store `spam.autofile=true` through the settings API, which lands without a restart.

How it learns — five doors, one set of eligibility rules so none can be skipped:

- **IMAP**: moving a message into Junk trains spam; moving it back out trains ham (and retracts the earlier label — one gesture does both).
- **JMAP**: the `$junk` / `$notjunk` keywords.
- **`POST /account/spam/train`** — an explicit "report spam" door for clients.
- **Autolearn** (`spam.autolearn`, off by default): a DMARC quarantine verdict trains spam; outbound mail can train ham.
- **Migration**: `zephyr-migrate --train-spam` turns a migrating user's years of filing into a corpus — the fastest path to a usable corpus that exists.

The silences are as deliberate as the signals: Junk → Trash trains nothing (deleting spam is not a claim it was wanted), and a move that does not cross the Junk boundary says nothing. Training marks are keyed by the message's content digest, not its location, so moving a message retracts cleanly.

What it never does: it never **refuses** a message on a learned score (filing to Junk is reversible and visible; an SMTP reject would turn a model's mistake into permanently undelivered mail); it never overrides an explicit Sieve rule; it never trains from an encrypted mailbox; it never acts below the minimum corpus. The threshold (`spam.threshold`, default 90) has a floor of 50 — below a coin flip, filing is a bug someone configured, and the floor is enforced in code.

Operate it with `zephyr-spam-ops` (section 15): `stats`, `prune` (drop one-off tokens when the vocabulary nears its 200,000 cap — past it the model refuses to rebuild rather than load a truncated, confidently-wrong half), `rebuild` (reconstruct the corpus from the per-message training marks; also the migration path for per-account personalisation), `forget` (drop a user's reply-tracking correspondents). Watch `zephyr_spam_model_errors_total` — scoring fails safe, so a broken corpus is silent: mail keeps flowing and the filter quietly stops improving.

Per-account personalisation exists on top of the tenant corpus: an account's own markings overlay the shared model for that account only. Accounts that never trained score exactly as before.

Antivirus

ZephyrAB speaks directly to **clamd** (ClamAV's daemon). One scanner serves two paths, and they deliberately fail in opposite directions:

- **The mail path fails OPEN** (`ZEPHYR_MAIL_AV=on`, scanner at `ZEPHYR_AV_CLAMD`). An infected message is refused at SMTP time with a `554` naming the signature — before the platform ever accepts responsibility for it. But if the scanner is unreachable or times out, the mail is **delivered unscanned and counted**. Refusing all mail because clamd restarted is worse than missing one scan: live mail cannot wait, and senders' retry windows expire.
- **The download path fails CLOSED** (link attachments, section 6 of the User Manual; `links.av_fail_open` defaults to false). A stored file can wait for the scanner to come

back; serving it unscanned buys nothing but risk. A domain admin may opt a domain into fail-open via the setting.

Messages larger than the scan budget (`ZEPHYR_MAIL_AV_MAX_SCAN_BYTES`, default 25 MiB) are delivered unscanned and counted as **skipped**, separate from **errors** — an oversized message is expected traffic, and folding it into the paging counter teaches operators to ignore the pager. Alert on `zephyr_mail_av_errors_total` rising: it means the scanner is silently doing nothing. The scan also runs on submission, so the platform does not send what it would refuse. Keep `freshclam` running so signatures stay current. Requesting AV without a scanner address is a startup error, not a silent no-op.

DNS blocklists (RBL)

`ZEPHYR_RBL=off|observe|enforce` checks connecting servers against DNS blocklists (`ZEPHYR_RBL_ZONES`, default `zen.spamhaus.org`) at MAIL FROM time. `observe` logs and counts what enforce would have refused; `enforce` answers `554 5.7.1` naming the zone.

It requires a local recursive resolver (`ZEPHYR_RBL_RESOLVER`, e.g. `127.0.0.1:5335` running unbound). This is not a preference: the major blocklists refuse queries arriving via public resolvers (Google, Cloudflare) — the query gets no answer or a refusal code, and an RBL pointed at a public resolver **silently does nothing**. zephyrd therefore refuses to start with RBL on and no resolver configured. The classifier also refuses three classic misreadings: a `127.255.255.x` answer is the provider refusing the query (an error, never a listing); a non-`127.x` answer is a wildcarding or dead zone (the expired-blocklist trap that "lists" the whole internet); a timeout is "could not look", never "not listed". The check **fails open** in every error case — a missed check must not cost a stranger's mail — which is why the metric to alert on is `zephyr_rbl_errors_total` rising: errors climbing means the RBL is silently off.

One expected consequence of `enforce` on the standard zones: residential IPs are policy-listed by design, so direct-to-MX mail from home connections will be refused. Your own users are unaffected — they submit on 587/465, which is never RBL-checked.

Greylisting

`hygiene.greylist` (off by default): first contact from an unrecognised (client IP, sender domain) pair is answered with a temporary failure; a legitimate server retries after the delay (default 60 seconds) and is remembered. Spam cannons often never retry. It is **adaptive**: senders with accumulated positive reputation bypass the delay, and the null sender (bounces) is never greylisted, per standard guidance. Cost: first mail from every new correspondent is delayed by minutes. If the greylist store cannot be read, the check accepts — availability errors never refuse mail.

Rate limits and brakes

Per-IP brakes at the frontends, all **off by default** (0) so a misconfigured cap can never silently drop legitimate mail; set them on public-facing hosts:

Variable	What it limits
`ZEPHYR_LIMIT_CONNS_PER_IP`	Concurrent sessions per source IP (421 beyond).
`ZEPHYR_LIMIT_CONN_RATE`	New connections per second per IP (token bucket, burst = value).
`ZEPHYR_LIMIT_AUTH_FAILS`	Failed logins per IP per 15 minutes, refused **before** the expensive password hash — protects the CPU pool.
`ZEPHYR_LIMIT_RCPT_FAILS`	Unknown-recipient rejections per IP per 15 minutes, then tarpit — kills mailbox enumeration and dictionary attacks.

These are per-connection-source protections; the per-account outbound caps (section 8) protect reputation, and the per-account login lockout (section 10) protects credentials. Three layers, three different keys.

An external filter (milter)

`ZEPHYR_MILTER=inet:host:port` (or a unix socket path) hands each inbound message to an external filter — the hook used by rspamd and similar. `ZEPHYR_MILTER_ON_ERROR=defer|accept` sets the posture when the filter fails; `defer` is the default, and messages larger than `ZEPHYR_MILTER_MAX_BYTES` are not sent to the filter. Off when unset.

10. Users and security

Why signup is invite-gated

Self-service signup (`ZEPHYR_SIGNUP=on`, off by default) requires an invite code. Open registration is refused as a design decision, and the reason is reputation: your domain and IP pass every authentication check and *still* earn inbox placement slowly (section 4). Open registration hands that reputation to whoever finds the page — one spammer sending through your DKIM-signed outbound path puts your IP on a blocklist in a day, and IP reputation is far easier to lose than rebuild.

Invite codes are bearer credentials: minted with `zephyr-invite new`, shown once, stored only as hashes. The signup endpoint is rate-limited per source before any expensive work, and its refusals are deliberately indistinguishable: a taken address and a reserved one return the same answer (otherwise the form is an address-enumeration oracle), and an unknown, expired, or exhausted invite all return the same answer (saying "expired" confirms a code was once real). Reserved local parts — `postmaster`, `abuse`, and the other standard role names — can never be claimed: `postmaster@` receives other operators' complaints and your own DMARC reports, and handing it to a stranger would intercept them.

Password rules

- Self-service (signup, webmail change-password): **at least 12 characters, and that is the only rule.** Length is the one requirement that measurably helps; composition rules mostly produce `Password1!`.
- The admin API password endpoint enforces a lower floor (8) for operator-set passwords; the console's own dialog asks for 12.
- Passwords are stored as argon2id hashes only. Nobody can read one back — not the operator, not the tooling. A lost password is reset (`zephyr-passwd`, or the console), never recovered.

Recovery addresses

A recovery address is where a password-reset link goes. Its failure is silent and permanent: an account whose recovery address never worked cannot be recovered by anyone, and the user is shown nothing but success until the day they need it. So ZephyrAB **checks the address before storing it**:

1. The domain must publish an **MX record** — deliberately stricter than the SMTP standard, which falls back to the A record. The case that justified it was real: a recovery address at a domain whose A record is a web host that never answers on port 25. Every "MX or A" check passes it; mail to it defers for days and bounces.
2. Some listed exchanger must **answer on port 25** with a mail-server greeting.

The probe is bounded, connects only to public addresses (an MX resolving to a private or loopback address is refused without connecting, so signup cannot be used to scan your own network), and is skipped for domains this platform hosts.
`ZEPHYR_RECOVERY_PROBE=full|dns|off` tunes it — `dns` for environments that block outbound 25, `off` as the last resort; an unrecognised value means `full`, because a typo must not silently disable a check.

Storing an address is not the end: the user must click the verification link mailed to it. Settings shows three states, and the middle one matters — **awaiting confirmation is not recovery**. `ZEPHYR_RECOVERY_FROM` must name a sender address (e.g. `postmaster@example.com`) or recovery mail cannot be sent and the feature refuses rather than pretending.

Two-factor authentication (TOTP)

Standard authenticator-app codes: six digits, thirty seconds. Enrolment is three states — off, **pending**, on — and pending is enforced nowhere: someone who scanned the wrong QR code, or whose phone clock is wrong, must find out while they still have a way in.

Once TOTP is on, the account password stops working everywhere — not just in the browser. Letting the raw password through on IMAP would leave the second factor as decoration, because a mail client is exactly where somebody would go to avoid it. Mail clients cannot be prompted for codes, so each protocol answers with text telling the user to use an application password instead.

Ways back in: ten single-use recovery codes minted at enrolment (shown once, stored hashed; a new enrolment invalidates the old set), and the operator command:

```
zephyr-totp-reset user@example.com --confirm user@example.com
```

A password reset deliberately does **not** remove TOTP. A reset that also cleared the second factor would be a takeover path: compromise the helpdesk, reset the password, walk past the factor that exists to stop exactly that.

Application passwords

Per-device credentials for mail clients: minted in webmail, shown once, individually revocable, at most 24 per account. Two scopes — `mail` (IMAP, POP3, SMTP submission, JMAP, ManageSieve) and `dav` (calendar and contacts).

The rule the feature rests on: **an application password can never sign in to webmail or the console.** Webmail is where the second factor is asked for; if a device credential worked there, anyone holding it would walk past TOTP. There is no code path in which one authenticates an interactive surface.

Useful operational details: a valid credential at the wrong door answers "wrong scope", not "wrong password" — and a wrong-scope attempt does **not** count toward the lockout, because the holder is the account owner with a misconfigured client. Minting one requires the account's current password, so a leaked device credential cannot mint more. **A password change revokes every application password and session in the same transaction** — someone changing their password after a scare expects everything cut off, and every configured client will need a new credential. That cost is stated next to the button in webmail.

Webmail sign-in itself exchanges the password (plus a code, when required) for a twelve-hour session token, so the account password crosses the wire once per session, not on every request.

OAuth (external identity provider)

ZephyrAB can validate sign-in tokens issued by an external identity provider — Google Workspace, Entra, Keycloak, Okta — so that the platform never holds a password for a federated user.

ZephyrAB is a resource server only. It is not an authorization server — no authorize endpoint, no consent screen, no client registry, and no token-exchange or impersonation endpoint. Consequence, stated plainly: a standalone deployment with no external IdP cannot do OAuth. Organisations that want single sign-on already have somewhere it lives. A companion runbook (`docs/runbooks/identity-federation.md` in the source tree) covers integration in detail, including how to broker a provider that speaks neither OIDC nor SAML.

A minimum configuration is three lines:

```
ZEPHYR_OAUTH_ISSUER=https://accounts.example.com/
ZEPHYR_OAUTH_AUDIENCE=zephyrab
ZEPHYR_OAUTH_JWKS_URI=https://accounts.example.com/jwks
```

The full set:

Variable	Required	Default	What it is
`ZEPHYR_OAUTH_ISSUER`	yes	—	The exact `iss` a token must carry
`ZEPHYR_OAUTH_AUDIENCE`	yes	—	The value that must appear in `aud`
`ZEPHYR_OAUTH_JWKS_URI`	yes	—	Where the provider publishes its signing keys
`ZEPHYR_OAUTH_ADDRESS_CLAIM`	no	`email`	Which claim names the mailbox. Some providers use `upn` or a custom claim; the value must contain an `@`
`ZEPHYR_OAUTH_REAUTH_MAX_AGE_SECS`	no	`300`	How recently the person must have authenticated for a token to stand in for re-entering a password

The first three are all-or-nothing: setting some but not all refuses to start, logs `ZEPHYR_OAUTH_* is set but incomplete` naming what is missing, and exits. The alternative is a deployment that believes single sign-on is on, advertises nothing, and finds out when a user cannot log in with the missing variable named nowhere. Unset entirely is off, and behaves exactly as every build before the feature existed.

Choose the address claim deliberately — this is the one setting here that can become an account-takeover path. `ZEPHYR_OAUTH_ADDRESS_CLAIM` names the claim ZephyrAB reads as *which mailbox is this*. It must carry an address your organisation controls, and it must not be a value the person can change upstream on their own. The trap is a provider whose most obvious-looking claim holds a **personal** address rather than the organisational one — the case that made this concrete was a national single-sign-on service that returns both a stable internal identifier and the citizen's private email address in the same profile. Map on the private address and whoever controls that private mailbox controls the hosted one, with nobody in your organisation involved and nothing in the logs looking wrong.

Where a provider's claims cannot be trusted for this, the fix is upstream rather than here: have the identity provider (or a broker in front of it) resolve its stable identifier through *your* user table and sign a claim carrying the ZephyrAB address, then point

`ZEPHYR_OAUTH_ADDRESS_CLAIM` at that claim. ZephyrAB refuses a claim value with no `@`, and refuses an address it does not already host — but neither check can tell a correct mailbox from a plausible wrong one, so the mapping is yours to get right.

Which surfaces accept a token:

Surface	How
IMAP, POP3, SMTP submission, ManageSieve	SASL `XOAUTH2` / `OAUTHBEARER`
JMAP	`Authorization: Bearer`
Self-service `/account/*` — the whole webmail settings surface	`Authorization: Bearer`
CalDAV / CardDAV	**No.** Use a `dav`-scoped application password
Admin API	Identity token via the RFC 7523 grant (below)
Web console	**No** — it signs in with a password

The self-service surface matters more than its size suggests. Without it a federated user could read their mail and reach none of their own settings — password, recovery address, application passwords, two-factor, encryption, filters, links — which would make an identity-provider deployment require a ZephyrAB password anyway, purely to administer the account. That is precisely the shadow credential federating exists to remove.

The admin API accepts an identity token, and it is a separate decision from federating mailboxes. The control plane's authorization model — scopes, subject-bound delegations — is ZephyrAB's own, and federating it changes only who may *prove they are the operator*, never what that operator may do. Post the provider's identity token to `/admin/api/oauth/token` with `grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer` and the token in `assertion`; it is verified by the same code that verifies one on IMAP or JMAP — one opinion about what a valid identity is — and the server mints its own short-lived opaque token in exchange.

The assertion proves who, and is not the session. Authority still comes from delegations read on every request, so a revoked grant stops working on the next call rather than when the provider's token expires; and the provider's token lifetime never becomes an admin session lifetime. Freshness is not required for ordinary sign-in — demanding it would refuse administrators on every provider that omits `auth_time` — but it is required for the operations that re-authenticate.

The shipped web console is the exception, and it is a real one: it signs in with a password. Obtaining an identity token in a browser needs an authorization-code redirect with PKCE, which ZephyrAB does not implement and which belongs to whoever owns the provider relationship. So on a federated deployment, either keep one administrator with a real ZephyrAB password, drive the platform with the machine client credential, or use your own front end — which has everything it needs.

Mechanisms are advertised only when a provider is configured. A client that sees `AUTH=XOAUTH2` will *prefer* it, so a server that offered the mechanism and then refused every token would be worse than one offering only `PLAIN`: the user meets an authentication failure on a server whose password login works perfectly. The same rule governs the self-service `WWW-Authenticate` challenge, which names `Bearer` only where a token could actually be honoured.

The safety rules are structural: the signature algorithm comes from the provider's key, never from the token (the `alg: none` family of attacks); the issuer and audience must match exactly, because a token minted by the right provider for a *different* application is a perfectly valid token; a password is never tried as a token nor a token as a password; an address claim the provider marks unverified is not an identity; and an account is never created from a token — a claim naming no mailbox here is refused, so provision first. A bearer token does not additionally require ZephyrAB's own TOTP: the IdP is the authentication authority, and that is where an organisation's MFA policy lives. A deployment that wants *this* server's second factor enforced should not enable the resource server.

Sensitive settings changes need a recent sign-in. Changing a password, setting a recovery address, minting an application password, enrolling or disabling two-factor, and the at-rest encryption routes all demand a second proof. Under a password that is `currentPassword` in the request body. A bearer token has no second value to carry, so what is demanded instead is freshness: the provider must say the person authenticated within `ZEPHYR_OAUTH_REAUTH_MAX_AGE_SECS`, read from `auth_time` (preferred) or `iat`, and a token carrying neither is refused rather than assumed recent. Raising the number weakens those operations for everyone; there is no value that switches the check off, and a zero or an unparseable value falls back to the default rather than to "no limit" — a typo in a number must not be the way a security check gets disabled.

What an identity-provider outage costs. Signature validation survives it: the key set is fetched at most every five minutes and cached, and a failed refresh **keeps the previous set** rather than emptying it — stale keys still verify every token the provider is currently issuing, where no keys verify none. (An empty key set is refused rather than installed, for the same reason.) What stops is *issuance*: no new sign-ins for as long as the provider is down. And if no keys have ever been fetched, every token is refused — "we could not check" must never read as "it was fine". For a service where mail access matters during an identity outage, keep one non-federated break-glass path: an operator account with a password, or application passwords already issued.

Provision federated accounts with no password. `POST /tenants/{id}/accounts` takes `password` as an optional field; omit it and the account exists with no credential at all, so password authentication fails for it by construction — nothing to leak, go stale, or outlive the upstream identity. Application passwords and sessions remain available and are not the account password: they are per-device, individually revocable, and refused on the interactive surface.

The honest gap: an account that **already has** a password keeps it, and there is no operation that clears one. Migrating existing accounts to federation therefore leaves the old credential in place and still working — the shadow credential is still there, just unused. A per-domain "password login disabled" setting is the natural fix and is not built. Until it is, the closest thing is `zephyr-passwd` (section 15) setting each migrated account to a fresh random value nobody records, which makes the password unusable without removing it. Treat that as a workaround, not a feature.

LDAP directory mode

For organisations with an existing directory. For the domains listed in `ZEPHYR_LDAP_DOMAINS`, the directory becomes the authority for who exists and what their password is: the password check is a real LDAP bind as the user (ZephyrAB never sees or stores a hash), and a person the directory vouches for gets a local account on first login (`ZEPHYR_LDAP_AUTO_PROVISION`, on by default).

The rules to know before enabling it:

- **`ZEPHYR_LDAP_DOMAINS` is the tenancy boundary** and is required. It is the only thing keeping one tenant's users from being authenticated against another tenant's directory.
- **The directory replaces the local password entirely** for its domains. The stored hash is not consulted, and setting a password on such an account is refused with the reason — two authorities for one credential is how an account keeps opening with a password the directory revoked. Application passwords, sessions and ZephyrAB's TOTP stay local and keep working; they are how a directory user configures a phone.
- **A directory that is down is not a wrong password.** An unreachable directory answers as a *temporary* failure on every protocol and never touches the lockout counter — otherwise one directory outage locks out the whole organisation while telling each user their password is wrong.
- TLS is not optional, there is no accept-any-certificate switch (`ZEPHYR_LDAP_CA_FILE` exists so nobody wants one), an empty password is refused before anything is sent (LDAP defines it as a successful anonymous bind — the most dangerous default in the protocol), and a search matching more than one entry is refused rather than picking the first.
- A half configuration refuses to start, like OAuth.

Alert on `zephyr_ldap_unavailable_total`: an unreachable directory refuses nobody by design, so from outside an hour of directory downtime looks like users with flaky mail clients. This counter is the only thing that says otherwise.

Account lockout

Five failed password attempts arm a fifteen-minute lockout for that address. The lockout gate runs before the expensive hash work, so a guessing script cannot burn CPU. Things that deliberately do **not** count: wrong-scope application-password attempts, and directory

outages. Setting a new password clears the counter — without that, a reset appears to do nothing for fifteen minutes.

At-rest encryption per account or domain

An account can opt in to at-rest encryption (OpenPGP or S/MIME — one format per account): every message delivered to it is encrypted to the user's own public key before storage. The server keeps no private key and cannot decrypt what it stores. A domain can make it mandatory via the `security.at_rest_encryption_required` setting — a Protective setting, so the domain or the account can turn it on and neither can turn the other's choice off.

The consequences are presented **before** the switch, because they are irreversible in the direction that matters:

- **The operator cannot read the mail back. Neither can support.** Recovery without the private key does not exist.
- **Body search stops working** for that mailbox. Searches over headers still work; body and full-text searches are *refused by name* rather than answering an empty result that looks like "no matches".
- **Mail to an opted-in account with no usable key is REFUSED, and the sender is told**: a permanent `550 5.2.1` at RCPT time stating that at-rest encryption is required and no usable key is on file. It is never stored unencrypted as a fallback.

The fail-closed posture is the design: wrongly refusing is loud and gets fixed the same day; wrongly storing plaintext is silent, permanent, and defeats the reason the user opted in. Two guards keep the refusal from surprising anyone: an account cannot opt in without a usable key on file, and a domain cannot arm `required` without the console's readiness survey plus an explicit acknowledgement that keyless accounts will have mail refused. A truncated survey never counts as a pass.

Honest boundaries: message headers (From, To, Subject, dates) remain unencrypted — that is how the format works, and pretending otherwise would be worse. Mail that arrives already encrypted by the sender is stored as-is even when the account's own key is missing — refusing it would punish the one sender who did the right thing. Certificate expiry is checked at upload, deliberately not at delivery: fail-closed plus a delivery-time expiry check would mean every message starts bouncing on a particular morning with nothing having changed.

11. Backups and recovery

What is backed up, and the rule that makes it consistent

Two stores back one product:

Store	Holds	Backed up by

FoundationDB	All metadata — mailboxes, message rows, flags, quota, provisioning, calendars/contacts, the acceptance ledger — **and the bytes of every message at or under the inline threshold (64 KiB by default)**	`fdbbackup` continuous backup (versioned, cluster-consistent)
MinIO	Message bodies above the threshold, content-addressed, reference-counted	An append-only file mirror, every 5 minutes

Because small bodies live inside FoundationDB, an FDB-only restore already reproduces most mail byte-for-byte. The blob side is where the consistency problem lives, and **no ordering of the two backups is correct on its own**: blobs-first misses a blob created in the window; metadata-first references a blob already collected in the window. The fix is in the source, not the ordering — **blob deletion is deferred** longer than the backup window:

```
ZEPHYR_BLOB_GC_GRACE_SECS  >  blob-mirror interval  +  longest mirror pass
```

Deleting a message writes a tombstone in the same transaction instead of deleting the object; `zephyr-backup gc-sweep` (on a timer) collects objects whose grace has expired. With the grace in place, every restore point in the retention window provably has every blob it references. The Solo install sets a 24-hour grace. zephyrd warns at startup when the grace is 0.

The safe restore point is `min(latest restorable FDB version, start of the last completed blob-mirror pass)`, and the tooling refuses to certify a point past the blob horizon. Recovery-point objective: seconds for metadata and small bodies; the mirror interval (5 minutes) for large bodies.

Restore verification — nightly, by restoring

A backup nobody has restored from is a hope, not a backup. `zephyr-backup verify` (the nightly `zephyr-restore-verify` timer) picks random mailboxes, restores them into an **isolated scratch cluster**, and diffs the result against live: body hashes, sizes, flags, dates, folders, presence. A silent pass is impossible by construction — nothing restored, nothing compared, or no blob pass all FAIL. It distinguishes "the source lost this body too" from "the backup lost it", because a backup wrongly accused is a backup nobody trusts.

Two safety rules the tooling enforces: the source cluster file and the scratch cluster file are never interchangeable — `zephyr-backup` compares their coordinator lists and aborts on overlap, because a restore **clears its target ranges** and a drill that could clobber

production is not a drill; and a restore is executed by backup agents attached to the *destination* cluster — without them it reports "running" forever and moves nothing.

Alerts that matter: `zephyr_restore_verify_ok == 0` (a verified restore failed — page); the last-success timestamp older than 36 hours (the nightly job may not be running at all, which looks identical to good news if you only alert on failures); the backup's restorable point lagging; the blob mirror falling behind.

Restore one mailbox, side by side

```
zephyr-backup restore      --account user@example.com --target-version
<V>
zephyr-backup side-by-side --account user@example.com --prefix
Restored-2026-08-06
```

The restored tree appears **beside** the live one — `Restored-2026-08-06/INBOX` next to `INBOX` in any client — and nothing live is moved or overwritten. The user copies back what they want. Re-running is a no-op rather than a second copy. The console's per-account **Restore…** action queues the same operation as a job; a node with no restore procedure configured (`ZEPHYR_RESTORE_COMMAND`) refuses before queueing anything.

Deleted-message recovery

Trash catches an ordinary delete. This catches the one after it — the user emptied Trash, or the client expunged:

```
ZEPHYR_UNDELETE_DAYS=7    # unset or 0 = off (the default)
```

With it on, a purged message leaves a recovery record behind, and the user restores it themselves (webmail's "Recently deleted", or `GET /account/deleted` + a restore call). A restored message returns to its mailbox under a new uid — clients see it as a message arriving.

The design decision that makes the promise real: the record **transfers** the message's blob reference rather than releasing and re-acquiring it. Between those two moments the blob GC grace would expire and collect the bytes — and a record that promises a message it can no longer produce is worse than offering no recovery, because the user is told their mail is safe right up until they ask.

What it costs: a recoverable message keeps its bytes after the user's quota already went back to them. Budget roughly `deletion rate × retention window × mean message size`. **Install the sweeper** (`zephyr-undelete-sweep`, daily, with `--apply`), or the retention window is a claim rather than a mechanism. The sweeper's `--days` and the server's `ZEPHYR_UNDELETE_DAYS` are separate numbers on purpose — the sweeper can trim a deployment that just shortened its window — but a sweeper set shorter than the server is a server promising recovery it no longer has; keep them in step. There is deliberately no delete-permanently verb on the recovery records: it would be a way to destroy evidence in one request, and waiting out the window achieves the same thing.

Export an account

```
zephyr-account-export --address user@example.com --out /path/to/export
zephyr-account-export --address user@example.com --out /path/to/export
--verify
```

Everything the platform holds about one account, in standard formats its owner can use: mail as `.eml`, calendars as `.ics`, contacts as `.vcf`, filters as Sieve, metadata as JSON. The manifest carries a SHA-256 per file and is written **last**, so a truncated export cannot pass as complete. `--verify` re-reads **the store**, not the export — walking the export would only prove it is self-consistent. Deliberately not exported: the password hash (tells the subject nothing and makes the file worth stealing), recovery tokens, and the platform's acceptance ledger. The output directory must not already exist — an export merged into anything else would be a disclosure.

Delete an account

```
zephyr-account-delete --address user@example.com
# dry run
zephyr-account-delete --address user@example.com --confirm
user@example.com
```

Dry run is the default; it surveys and prints what would be removed. `--confirm` must equal the **canonical** address — deleting via an alias still requires typing the real one.

Why deletion is not a simple range clear, even though every account key sits under one prefix: message rows are the only record of which blob each message referenced, and blob reference counts are global because deduplication spans accounts. A range clear drops the rows without decrementing, leaving every refcount permanently too high — those blobs become immortal, and nothing can repair it afterwards. So messages leave one transaction at a time through the ordinary purge, which settles refcount, quota and tombstones together.

The order is the safety property: fence the account (status Deleted) → revoke credentials and tokens → purge messages → remove calendars and address books → verify quota is zero → clear the residue → **unprovision the address last**. The asymmetry: address gone with data left means mail bounces and the data is orphaned but whole; data gone with the address still provisioned means SMTP keeps accepting and quietly re-materialises the account. The second is worse, so the address goes last. A crash mid-run leaves a fenced, resumable account — never half a deletion.

12. Multi-cell operations

Multi-cell is a licensed feature (section 13). Read section 5's definition first: a cell is a scaling unit, **not** redundancy — a domain lives in exactly one cell, and losing that cell takes that domain's mail down regardless of how many other cells exist.

When a second cell makes sense

- **Capacity** — a cell's practical envelope is around a million mailboxes; past it, add a cell rather than growing one without bound.
- **Blast radius** — a bad deploy, a hot tenant or a corrupted store hits one cell's population. Canary new builds cell by cell.
- **Residency** — a tenant whose data must live in a particular place gets a cell there.

At small scale it buys none of these; do not add one for comfort.

The routing map and zephyr-cell-ops

Every cell holds a replica of the **cell map**: which cells exist, their endpoints, and which cell each domain is bound to. `zephyr-cell-ops` is the operator window onto it:

```
zephyr-cell-ops list                          # every registered cell
zephyr-cell-ops show   --cell <id>
zephyr-cell-ops put    --cell <id> [--smtp h:25] [--https URL] [--imap
h:993] ... [--yes]
zephyr-cell-ops bind   --domain example.com --cell <id> --yes
zephyr-cell-ops unbind --domain example.com --yes
zephyr-cell-ops route  --domain example.com
zephyr-cell-ops pauses | pause | resume    # bounded write-pauses
(tenant moves)
```

- `put` is a read-modify-write: correcting one endpoint cannot silently blank the others, and a blank endpoint is unroutable. Endpoints are TLS ports carrying **names**, never IP literals — the cross-cell proxies verify the certificate name, and an IP is a thing anyone can take over.
- `bind` **moves a domain's mail** and requires `--yes` after printing the consequence: from the moment it commits, the named cell serves the domain and the previous owner fences it. **There is deliberately no way to hand-write an epoch** (the version number bindings carry): the increment happens inside one transaction, so two operators racing on different cells cannot both believe they won, and a hand-typed number could outrank and silently reverse a real move.
- `unbind` is for a domain **leaving the platform**, not for moving one — on a cell with peers, an unbound domain is refused by every protocol surface. To move a domain, `bind` it to the new cell. Unbinding writes a replicating tombstone so a peer's stale copy cannot resurrect the binding.

Cells learn each other's maps over a mutual-TLS feed (`ZEPHYR_CELL_FEED_*`, `ZEPHYR_CELL_PEERS`). **The feed is pull-only, so the peer mesh must be stated in both directions** — every cell lists every other cell; a hub shape silently fails to propagate. And the safe join order is identity → register and bind → peers, because the moment a cell gains a peer, any domain it hosts but has not bound is refused with a permanent error. `zephyrab add-cell` (section 2) encodes all of this as a reviewed step list.

Secondary MX

Cells can back each other up at the SMTP layer: publish the other cells as lower-preference MX records, and a cell receiving mail for a domain it does not own **accepts, queues and relays** it to the owner — so a sender never has to hold mail through the owner's reboot. Four prerequisites before publishing such an MX, each of which prevents a way of damaging your own reputation:

1. **Recipient digests** — each cell publishes a compact digest of the addresses it accepts, replicated with the map. A relay checks it at RCPT: a miss is definitive and answered with a real `550 no such user`; a hit accepts and relays. Without this, a relay accepts dictionary-spam recipients, takes the owner's rejection, and bounces to forged senders — **backscatter from your own IP**. If a peer's digest is missing or stale, the relay answers a temporary `450`, never an accept: accepting what cannot be verified is exactly the hole. Alert on `zephyr_relay_rcpt_unverifiable_total` — sustained non-zero means a peer's digest is stale and its inbound mail is deferring.
2. **Content parity** — the relay must run the same malware scanning as the primary, or it becomes the weaker path that accepts what the owner then rejects, and bounces it.
3. **PTR/FCrDNS and SPF** for the relay hosts — a relay eventually originates mail (a bounce, after the retry schedule is exhausted).
4. **The MTA-STS policy must list every MX host** before the mode ever leaves `testing` — an enforcing sender filters out any MX the policy omits.

Move a tenant between cells

`zephyr-tenant-move` moves one tenant online, in explicit phases you invoke separately — you decide when the pause happens:

```
zephyr-tenant-move survey  --tenant <id> --src <cluster-file> --dst
<cluster-file> --src-cell a --dst-cell b
zephyr-tenant-move sync    ... --rounds 3     # repeat until a round
changes nothing
zephyr-tenant-move cutover ... --yes
zephyr-tenant-move verify  ...
```

What the user experiences: nothing, until the cutover — and during it they can still read their mail. New mail is **deferred, not refused**: senders get a temporary `451` and retry on their own schedule, so a bounded pause costs latency and never a message. The window is minutes, and **it ends by itself** even if the mover dies. Afterwards users sign in again — a session is a bearer credential scoped to one cell.

`sync` reconciles rather than copies (a message expunged between rounds is expunged at the destination too), and rounds shrink. The cutover pauses writes, waits out the fence grace, copies the final delta, checks its time budget **before** flipping, binds each domain destination-first (so there is never an instant where no cell claims the domain — an unclaimed domain is the one answer that loses mail), and lifts the pause. Any failure lifts the pause and leaves the source authoritative. `verify` re-reads both cells and exits 1 on

any discrepancy. After a completed cutover, the way back is another `bind` — the source still holds all the data until you delete it deliberately.

This is an operator-privileged act from a temporarily trusted host. The mover needs simultaneous access to both FoundationDB clusters, and FoundationDB has no per-key authorization — whoever runs it holds unrestricted read/write to both cells for the duration. Run it from a host you already trust with both, open the destination's cluster port to that host only, and **close the opening when done** — write that into the change record before you open it, or it does not get closed.

`survey` prints a **STAYS BEHIND** list; read it. The acceptance ledger, the outbound queue (the mover blocks on queued outbound mail rather than guessing), sessions and tokens, and link-attachment records all stay in the source cell, each for a stated reason.

The version-skew rule

Deploy the same binary to every cell before using a new map feature. The precedent that made this a rule: the unbind tombstone. An older binary does not know the tombstone field, reads the record as a live binding, and honours it — so a tombstone written while any cell ran the older build would be honoured by some cells and ignored by others. That is a split brain arriving through version skew, with every cell internally consistent. The packaged upgrade path (`zephyrab upgrade --plan`) exists to move a fleet level in one pass; the same rule applies to the outbound next-hop field and any future addition to replicated records.

How clients reach the right cell

For completeness: inbound SMTP routes by DNS (a hosted domain's MX names its cell); JMAP answers a misdirected client with the owning cell's URLs (the protocol's own redirect); IMAP, POP3 and ManageSieve **proxy the session** to the owning cell before authentication, over TLS that verifies a name — which is what lets credentials stay cell-local, so compromising one cell does not let anyone authenticate as another cell's users. Download links carry the cell in the URL path and are proxied the same way. None of this needs operator action beyond a correct cell map.

13. Licensing

The two editions

Section 5 has the tier table. In operation: **community** is the platform with no license file — the complete mail system, every protection, every data-safety feature, one mail domain and one cell, unlimited mailboxes, forever. A **license file** adds the commercial features: `multi-cell`, `delegated-rbac` (creating delegation grants), `commerce` (plans, usage, bulk provisioning), and `sso` (LDAP/OAuth), plus whatever limits the license names.

Nothing that protects users is ever gated. Spam filtering, antivirus, encryption, backups, restore, migration and export work unlicensed, forever. Gating them would monetize user harm.

Install a license

Drop the signed license file at `/etc/zephyrab/license.json` (or point `ZEPHYR_LICENSE_FILE` elsewhere). No restart: zephyrd re-reads it on the switches refresh tick, so the state flips within seconds. Renewing is the same act — replace the file. The license is verified offline against keys embedded in the binaries; there is no activation server and no phone-home.

License states

State	Meaning	Administration	Mail
community	No license file	Community rules (one mail domain, one cell, unlimited mailboxes; no commercial features)	Normal
valid	Signed, in date	Everything the license names	Normal
expiring soon	30 days or less left	Everything, with warnings in the console, doctor and logs	Normal
grace	Expired, within 14 days	Everything, loud warnings	Normal
lapsed	Past the grace	**New** accounts, domains and cells are refused; everything existing keeps working; reads work	Normal
invalid	Present but unreadable, unverifiable, or revoked	Community rules, plus an alertable state naming the error	Normal

Note the last row: a present-but-broken license is **never** silently treated as community — the state is `invalid` and the reason is the fix. And a revoked license id (the revocation list ships inside the binaries, so it reaches a deployment at its next upgrade) evaluates as invalid with a message naming the issuer.

What a refusal looks like

Two kinds, distinct in the response and the logs:

- **`license limit`** — a provisioning ceiling: "the license for Example Corp allows 100 domains and this would be the 101st — raise the limit or retire a domain." Checked where things are *created*; never on reads, never on delivery. The account counter behind the mailbox limit fails **open** when it cannot be read (counted and logged; repair with `zephyr-backfill-account-count`) — a broken counter must not block provisioning.
- **`license entitlement`** — a feature the license does not list: "the license for Example Corp does not include commerce — reading the catalogue and existing subscriptions keep working." An enterprise license holds exactly what its `features` list names; nothing is implied, because an ambiguous license is a dispute waiting to happen.

Refusals name the licensee and the numbers in full sentences. An operator hitting a ceiling should never have to guess whether it is a bug.

The rules that outrank everything

- **An expired license never stops mail.** Nothing in the delivery path consults the license — not for performance, but on principle: holding users' correspondence hostage to a commercial dispute with their operator is not a failure mode this platform will have.
- **Access is never revoked by license state.** Existing delegations keep being honoured in every state, community included (dropping stored grants on a downgrade would *widen* access, not narrow it). Revoking a delegation is never gated — taking access away must never require a license.
- **SSO is never cut off.** LDAP/OAuth configured without the `sso` entitlement logs a loud warning and sets the `zephyr_license_sso_unentitled` gauge — and keeps working, because cutting sign-on locks users out of their existing mail.

Watch it

`GET /license` (and the console's License card) reports state, licensee, days left, usage against effective limits, and the entitlement map. Metrics: `zephyr_license_state` (numeric; the metric's help text carries the coding) and `zephyr_license_days_left` — a community deployment exports 36500 days, so an expiry alert can never fire for a deployment with nothing to renew. Alert at 30 and 7 days; a renewal should never be a surprise.

14. Monitoring

Metrics

zephyrd exports Prometheus metrics at `/metrics` on `ZEPHYR_METRICS_ADDR` (default `127.0.0.1:9464`). Keep this listener off the public internet. Counters reset on process

restart; the durable series is whatever your Prometheus retains. The backup and sweep jobs write node-exporter textfile metrics, so run a node exporter with the textfile collector on hosts that run them. The repository ships example alert rules under `deploy/observability/` covering the platform, backups, cells and licensing.

The alerts that matter most, in plain words

Signal	What it means when it fires
`zephyr_parser_panics_total` non-zero	Hostile mail is exercising a live parser bug right now. The panic is contained per message, but alert on **any** non-zero value: capture the traffic and treat it as an incident, not a statistic.
`zephyr_relay_rcpt_unverifiable_total` rising	A peer cell's recipient digest is missing or stale, so that cell's inbound mail is being **deferred** at your relays. Senders retry, so nothing is lost yet — but it does not fix itself.
`zephyr_blob_gc_due` climbing without bound	The blob garbage-collection sweeper is behind (or not installed). `due` normally sawtooths between sweeps; the failure shape is a monotone climb. Alert on the trend, never on `pending`, whose steady state is legitimately large.
`zephyr_license_state` / `zephyr_license_days_left`	License trouble ahead of time: expiring, in grace, lapsed, or invalid. Community deployments report 36500 days left, so the expiry alert stays quiet where it should.
`zephyr_restore_verify_ok == 0`, or its last success older than 36 h	A verified restore failed — page — or the nightly restore-verification job has stopped running, which looks identical to good news if you only alert on failures.
`zephyr_spam_model_errors_total` rising	The spam model cannot rebuild. Scoring fails safe, so mail keeps flowing and the filter silently stops improving — this counter is the only tell.
`zephyr_mail_av_errors_total` rising	The virus scanner is unreachable. The mail path fails open, so mail is being delivered **unscanned** while everything else looks healthy.
`zephyr_rbl_errors_total` rising	The blocklist check is failing open — the RBL is silently off.
`zephyr_ldap_unavailable_total` rising	The directory is down. Nobody is being refused (by design), so from outside it

	looks like flaky mail clients.
`zephyr_smarthost_loops_total` non-zero	Two nodes are relaying through each other; every counted message was bounced to its sender.
`zephyr_outbound_auto_locked_total` moving	An account was auto-locked for outbound abuse — usually a compromised account. Investigate before unlocking.
`zephyr_switch_domain_sweep_failures_total` rising	Per-domain settings are not being refreshed: every domain is quietly running on platform values while everything else looks healthy.

One command

`zephyrab doctor` is the health check and the support command: read-only, safe anywhere, exit 1 on any failure. Run it first, and attach its output to any support request.

The in-product status surfaces

The console's **Health & Security** page is the operator's live view: firing alerts (relayed from Alertmanager when `ZEPHYR_ALERTMANAGER_URL` is set — and shown as "cannot look" when it is unreachable, never as "no alerts"), the license card, feature pills, and this node's counters. For request tracing, set `ZEPHYR_OTEL_ENDPOINT` and zephyrd exports OpenTelemetry spans for the accept → delivery path. The repository's `deploy/observability/` tree carries the example Prometheus, Alertmanager and dashboard configurations it was all built against, including the machinery for a public status page if you want one.

15. Command-line tools reference

All tools read the same `/etc/zephyrab` environment files the services use. Most need the environment loaded to find the database and blob store:

```
set -a; . /etc/zephyrab/zephyrd.env; set +a
zephyr-queue-ops list
```

House rules the tools share: destructive commands are dry-run by default or demand `--yes`/`--confirm`; refusals happen before anything is touched and name the fix; secrets are never taken as command-line arguments (arguments are visible in `ps` and shell history).

All packaged tools

Tool	One line
`zephyrd`	The server. One binary; SMTP, submission, IMAP, POP3, JMAP, DAV, ManageSieve, metrics and the outbound queue-runner.

`zephyr-dnsd`	The authoritative DNS server, DNSSEC included. Nameserver hosts only.
`zephyrab`	The installer/lifecycle CLI: install, plan, preflight, doctor, dns-check, upgrade, add-cell.
`zephyr-account-delete`	Remove one account and all its data, in the order that keeps blob refcounts correct. Dry run by default.
`zephyr-account-export`	Export everything held about one account as standard formats, with a verifiable manifest.
`zephyr-alarm-index`	Schedule calendar alarms for events stored before the alarm feature existed.
`zephyr-arf-ingest`	Read feedback-loop complaints out of `abuse@` and attribute each to the sending account.
`zephyr-backfill-account-count`	Establish or repair the platform account counter the license gate reads.
`zephyr-backfill-counts`	Establish the per-mailbox message-count key for mailboxes that predate it.
`zephyr-backfill-domainids`	Assign stable ids to domain records that predate them.
`zephyr-backfill-fts`	Build the full-text search index over existing mail.
`zephyr-backfill-internaldate`	Stamp an arrival date onto mail stored before the field existed.
`zephyr-backfill-threads`	Build conversation-thread links for existing mail.
`zephyr-backup`	Point-in-time restore, nightly restore verification, side-by-side restore, and the blob-GC sweeper.
`zephyr-blocklist-watch`	Check your own IP and domains against DNS blocklists; a check that cannot run fails loudly.
`zephyr-cell-ops`	Read and write the cell routing map: list, show, put, bind, unbind, route, pause, resume.
`zephyr-dkim-rotate`	Operator window onto a domain's DKIM rotation: `show` and `tick`.
`zephyr-dmarc-ingest`	Read DMARC aggregate reports out of the rua mailbox; `--report` prints the trend

	and failing sources.
`zephyr-dns-keys`	DNSSEC key operations: `show`, `tick`, KSK rollover, DS verification at the parent.
`zephyr-invite`	Mint, list and revoke signup invite codes. Codes print once; only hashes are stored.
`zephyr-link-ops`	List, inspect, audit and revoke link-attachments.
`zephyr-link-sweep`	Reclaim the storage behind expired link-attachments.
`zephyr-mark-scratch`	Mark a FoundationDB cluster as a scratch (test) cell — integration suites refuse to run against anything unmarked. Never mark production.
`zephyr-migrate`	Migration from IMAP, Maildir, mbox, Sieve, calendars and contacts, with fidelity verification and cutover/rollback (section 17).
`zephyr-passwd`	Set an account's password interactively, echo off. There is no `--password` flag, on purpose.
`zephyr-queue-ops`	Inspect the outbound queue; retry, bounce, pause and resume delivery.
`zephyr-seed`	Bulk-provision synthetic mailboxes for load testing, through the live delivery path. Test cells only.
`zephyr-sendmail`	Submit one message from stdin through the real signing and queue path — the outbound test tool.
`zephyr-sieve-scan`	Re-parse every stored filter with **this** binary's parser. The pre-upgrade gate.
`zephyr-spam-ops`	Inspect, prune, rebuild and trim a tenant's spam corpus.
`zephyr-tenant-move`	Move one tenant between cells, online, with a bounded write-pause (section 12).
`zephyr-tenant-purge`	Remove test-residue tenants from a cell.
`zephyr-tlsrpt-ingest`	Read TLS reports out of the postmaster mailbox; `--report` prints the trend by failure and MX.
`zephyr-totp-reset`	Turn an account's second factor off, as an operator. Reports by default; the address is typed twice.

`zephyr-undelete-sweep`	Release deleted-message recovery records past their window. Dry run without `--apply`.
`zephyr-upload-sweep`	Reclaim storage behind abandoned webmail attachment uploads.
`zephyr-usage`	Read a tenant's per-day usage meters (the flow half of billing evidence).
`zephyr-usage-rollup`	Close a billing period and freeze it, once, after the grace window.
`zephyr-usage-sample`	Record today's stock sample (mailboxes, storage, domains) for every tenant. Daily timer.

The ten you will use most

zephyr-invite

```
zephyr-invite new --domain <domain> [--uses N] [--days N]
                  [--quota-mb N] [--note <text>] [--tenant <id>]
zephyr-invite list
zephyr-invite show --code <code>
zephyr-invite revoke --code <code> --yes
zephyr-invite revoke --id <id> --yes
zephyr-invite reserved
```

Defaults: 1 use, 14 days, 1024 MB quota; the tenant is resolved from the domain. The code prints once and is not recoverable — only its hash is stored. `reserved` prints the local parts signup can never claim. `revoke` demands `--yes`.

zephyr-passwd

```
zephyr-passwd <address>
```

Prompts twice with echo off. Refuses a `--password` flag by name — an argument is visible in `ps`, in shell history, and in any transcript. Minimum 12 characters. Reading from a pipe works but warns.

zephyr-queue-ops

```
zephyr-queue-ops list [--max N]
zephyr-queue-ops show   --qid <qid>
zephyr-queue-ops retry  --qid <qid>
zephyr-queue-ops bounce --qid <qid> [--reason TEXT] [--yes]
zephyr-queue-ops status
zephyr-queue-ops pause  --reason TEXT [--yes]
zephyr-queue-ops resume
```

`bounce` settles the entry exactly as retry-exhaustion would — ledger reconciled, and a delivery-failure notification to the sender — and refuses an entry under a live lease. `retry`

brings a deferred entry forward **without** resetting its attempts. `pause` stops every node sharing the queue, not just this one; bounce notifications for earlier failures keep going out.

zephyr-spam-ops

```
zephyr-spam-ops stats   --tenant <id> [--min-corpus 50]
zephyr-spam-ops prune   --tenant <id> [--min-total 2] [--apply]
zephyr-spam-ops rebuild --tenant <id> [--apply]
zephyr-spam-ops forget  --tenant <id> --account <addr> [--older-than-
days 730] [--apply]
```

Dry run is the default everywhere; only `--apply` writes. `rebuild` reconstructs the corpus from the per-message training marks — the evidence users actually gave — and reports how many marked messages are gone before you apply. `forget` requires `--account` so there is no tenant-wide sweep.

zephyr-cell-ops

```
zephyr-cell-ops list | show --cell <id> | route --domain <name> |
pauses
zephyr-cell-ops put    --cell <id> [--status active|draining|down] [--
smtp h:25] ... [--yes]
zephyr-cell-ops bind   --domain <name> --cell <id> --yes
zephyr-cell-ops unbind --domain <name> --yes
zephyr-cell-ops pause  --domain <name> --secs N --reason <why> --yes
zephyr-cell-ops resume --domain <name>
```

`bind` moves a domain's mail and says so before `--yes`. `unbind` is for a domain leaving the platform — to move one, bind it to the new cell. A `--status` typo is refused, never defaulted. Needs `ZEPHYR_CELL_ID` in the environment; the refusal prints the `set -a; . /etc/zephyrab/zephyrd.env; set +a` incantation.

zephyr-backup

```
zephyr-backup verify        [--sample N] [--account ADDR] [--seed N]
[--metrics-out PATH]
zephyr-backup restore       --account ADDR [--target-version V | --
target-timestamp T]
zephyr-backup side-by-side  --account ADDR --prefix Restored-YYYY-MM-
DD
zephyr-backup gc-sweep      [--limit N]
zephyr-backup ranges        --account ADDR
```

Common options: `--cell-cluster-file` (the source, read-only), `--scratch-cluster-file` (the isolated restore target), `--backup-url`, `--blob-backup-dir`, `--state-dir` — each also settable by environment. It refuses to restore into a cluster that shares coordinators with the source, and refuses to certify a restore point past the blob-mirror horizon (`--allow-past-horizon` overrides, by name). `side-by-side` is additive by construction; re-running is a no-op.

zephyr-account-export

```
zephyr-account-export --address user@example.com --out /path/to/dir
[--verify]
```

The output directory must not already exist. `--verify` re-reads the store against the manifest. Refuses to run when the blob store is unreachable — an export that silently omitted large bodies would be worse than no export.

zephyr-account-delete

```
zephyr-account-delete --address user@example.com                    # dry run
zephyr-account-delete --address user@example.com --confirm <canonical> [--keep-address]
```

Dry run surveys and prints what would be removed. `--confirm` must equal the canonical address. `--keep-address` deletes the data but leaves the address provisioned and fenced. A mid-run failure stops and leaves the account fenced and resumable rather than stranding blob refcounts.

zephyr-usage

```
zephyr-usage --tenant <uuid> [--days N]
zephyr-usage --address <addr> [--days N]
```

Read-only. `--address` resolves to a tenant the way delivery does, because tenant ids are uuids and asking an operator to type one invites the mistake. Default window 30 days.

zephyr-migrate

```
zephyr-migrate run|verify|plan        --source-host HOST --accounts FILE ...
zephyr-migrate import-local|verify-local --address ADDR --maildir PATH | --mbox PATH ...
zephyr-migrate import-sieve           --address ADDR --sieve PATH [--dry-run]
zephyr-migrate import-dav             --address ADDR --calendars PATH --contacts PATH [--dry-run]
zephyr-migrate phase|rollback|status  --address ADDR ...
```

Section 17 covers it in full. Exit code 0 means the run **passed** verification; anything else is non-zero, so a pipeline cannot mistake a partial migration for a finished one.

16. Environment variables reference

The variables an operator actually sets, grouped. The environment is the **platform layer** of the settings system: a value stored through the console or API overrides it, and clearing the stored value falls back to the environment. Defaults below are what an unset variable means. An empty value is not the same as unset for most variables — leave a line out rather than setting it empty.

Core identity and listeners

Variable	Default	What it does
`ZEPHYR_HOSTNAME`	`dev.zephyrab.local`	The server's own name

		(banner, authentication-results). Set it.
ZEPHYR_HELO	mail.zephyrab.com	The EHLO name and reporting identity. Set it to your mail hostname.
ZEPHYR_SMTP_ADDR	127.0.0.1:2525	SMTP ingress bind. Production: 0.0.0.0:25.
ZEPHYR_SUBMISSION_ADDR	127.0.0.1:2587	Submission (STARTTLS) bind. Production: 0.0.0.0:587.
ZEPHYR_SUBMISSIONS_ADDR	unbound	Submission over implicit TLS (465). Needs TLS configured.
ZEPHYR_IMAP_ADDR / ZEPHYR_IMAPS_ADDR	127.0.0.1:1143 / unbound	IMAP and IMAP-over-TLS (993) binds.
ZEPHYR_POP3_ADDR / ZEPHYR_POP3S_ADDR	127.0.0.1:1110 / unbound	POP3 and POP3-over-TLS (995) binds.
ZEPHYR_MANAGESIEVE_ADDR / ZEPHYR_MANAGESIEVE_TLS_ADDR	127.0.0.1:14190 / unbound	ManageSieve binds (4190 for the TLS one).
ZEPHYR_JMAP_ADDR	127.0.0.1:8080	The HTTP listener: JMAP, webmail API, /admin/api, /dav. Keep it loopback behind your proxy.
ZEPHYR_METRICS_ADDR	127.0.0.1:9464	Prometheus metrics.
ZEPHYR_JMAP_BASE_URL	empty	Public base URL advertised to JMAP clients and used in signup/recovery links, e.g. https://mail.example.com.
ZEPHYR_CLIENT_HOST	empty = advertise nothing	The hostname handed to mail clients in setup instructions.
ZEPHYR_MAX_MESSAGE_BYTES	104857600 (100 MiB)	Largest accepted message.
ZEPHYR_TCP_KEEPALIVE_SECS	120 (off disables)	Keepalive on long-lived IMAP/POP3 sessions.
ZEPHYR_LOG_ADDRESSES	full	redacted replaces local parts in logs with fingerprints.
ZEPHYR_OTEL_ENDPOINT	off	OTLP trace export endpoint.

ZEPHYR_ALERTMANAGER_URL	none	Alertmanager the console's alerts panel reads.

TLS and ACME

Variable	Default	What it does
ZEPHYR_TLS_CERT / ZEPHYR_TLS_KEY	none = no TLS	PEM paths. Both or neither; without them no TLS port binds.
ZEPHYR_ACME	off	on runs the built-in renewal loop (DNS-01, key-reusing, hot-swap).
ZEPHYR_ACME_DOMAINS	$ZEPHYR_HELO	Certificate identifiers, comma-separated.
ZEPHYR_ACME_DIRECTORY	production	staging, production, or a directory URL. Prove it against staging first.
ZEPHYR_ACME_DNS	godaddy	Challenge backend: godaddy (needs ZEPHYR_ACME_ZONE, GODADDY_KEY, GODADDY_SECRET) or zephyr (hosted DNS; needs ZEPHYR_ACME_DNS_CHECK, your nameservers to certify propagation against).
ZEPHYR_ACME_RENEW_DAYS	60	Renew when the certificate is older than this.
ZEPHYR_ACME_CONTACT	none	ACME account contact address.

Storage and delivery

Variable	Default	What it does
ZEPHYR_S3_ENDPOINT	http://localhost:9000	Blob store endpoint.
ZEPHYR_S3_BUCKET / ZEPHYR_S3_REGION	zephyr-blobs / us-east-1	Bucket and region.
ZEPHYR_S3_ACCESS_KEY / ZEPHYR_S3_SECRET_KEY	dev placeholders	Blob credentials. The installer mints real ones.
ZEPHYR_BLOB_INLINE_MAX	65536	Bodies at or under this ride inside FoundationDB; larger go to the blob store.
ZEPHYR_BLOB_GC_GRACE_SECS	0 = immediate delete	Deferred blob deletion. **Set it above your blob-mirror**

		window (the installer uses 86400) or backups cannot be consistent (section 11).
`ZEPHYR_NATS_URL`	off	NATS push bus for IMAP IDLE / JMAP push at scale; unset uses the built-in database-watch fallback.
`ZEPHYR_DELIVER_BATCH` / `ZEPHYR_DELIVER_WORKERS`	`64` / `8`	Delivery batching. Leave alone unless you are doing throughput work.
`ZEPHYR_MBOX_COUNT`	`off`	`trust` reads the per-mailbox counter key. Run `zephyr-backfill-counts` first — the order is load-bearing.
`ZEPHYR_UNDELETE_DAYS`	`0` = off	Deleted-message recovery window, in days (section 11).
`ZEPHYR_SWITCH_REFRESH_SECS`	`10`	How often each frontend re-reads the platform switches.
`ZEPHYR_DOMAIN_SWITCH_REFRESH_SECS`	`60`	How often the per-domain settings overlay is swept.

Mail posture and outbound

Variable	Default	What it does
`ZEPHYR_MAILAUTH`	`observe`	SPF/DKIM/DMARC on ingress: `observe` stamps Authentication-Results; `enforce` acts on DMARC verdicts; `off`.
`ZEPHYR_RCPT_VERIFY`	`on`	Refuse unknown recipients at RCPT.
`ZEPHYR_SUBADDRESS`	`on`	`user+detail@` delivers to `user@` when the exact address does not exist.
`ZEPHYR_DKIM_KEY` / `ZEPHYR_DKIM_DOMAIN` / `ZEPHYR_DKIM_SELECTOR`	none / `zephyrab.com` / `zab1`	The deployment signing key. Without a key this node cannot originate mail (relay still works). Set domain and selector to your own.
`ZEPHYR_ARC_SEAL`	`on`	ARC-seal forwarded copies, reusing the DKIM key.

ZEPHYR_MTA_STS / ZEPHYR_DANE	off / off	Honour recipients' MTA-STS policies / TLSA records when sending: testing or enforce.
ZEPHYR_SMARTHOST / ZEPHYR_SMARTHOST_TLS	none / verify	Route all outbound through one host (section 8). verify or opportunistic; a typo is a startup error.
ZEPHYR_SUBMISSION_TRACE	off	Received-header on submitted mail: private (no client IP) or full.
ZEPHYR_OUTBOUND_MSGS_PER_HOUR / ZEPHYR_OUTBOUND_RCPTS_PER_DAY	0 = off	Per-account outbound caps.
ZEPHYR_OUTBOUND_AUTOLOCK	off	Auto-lock an account that burns a whole hourly allowance on refusals and keeps hammering.
ZEPHYR_DNS_SERVER	system resolver	Explicit resolver for MX lookups, ip or ip:port.

Inbound filtering

Variable	Default	What it does
ZEPHYR_HYGIENE	scan	Inbound hygiene: scan, greylist (adds greylisting), or off.
ZEPHYR_RBL	off	observe or enforce DNS blocklists. Requires ZEPHYR_RBL_RESOLVER — a **local** recursive resolver, e.g. 127.0.0.1:5335; on without it is a startup error.
ZEPHYR_RBL_ZONES	zen.spamhaus.org	Blocklist zones, comma-separated.
ZEPHYR_MAIL_AV	off	on scans mail through clamd at ZEPHYR_AV_CLAMD (tcp://host:port or a unix socket path). On without an address is a startup error.

ZEPHYR_MAIL_AV_MAX_SCAN_BYTES	26214400 (25 MiB)	Larger messages are delivered unscanned and counted as skipped. Keep it in step with clamd's own StreamMaxLength.
ZEPHYR_MILTER	off	External filter at inet:host:port or a socket path; ZEPHYR_MILTER_ON_ERROR=defer|accept (default defer).
ZEPHYR_SPAM	off	The learning classifier: off, train, on (section 9).
ZEPHYR_SPAM_THRESHOLD / ZEPHYR_SPAM_MIN_CORPUS	90 / 50	Filing threshold (floor 50) and per-class corpus minimum.
ZEPHYR_SPAM_AUTOLEARN	off	Train from DMARC quarantine verdicts and outbound ham.
ZEPHYR_LIMIT_CONNS_PER_IP / ZEPHYR_LIMIT_CONN_RATE / ZEPHYR_LIMIT_AUTH_FAILS / ZEPHYR_LIMIT_RCPT_FAILS	0 = off	The per-IP brakes (section 9). Set them on public edges.

Features

Variable	Default	What it does
ZEPHYR_DAV	off	CalDAV/CardDAV at /dav on the HTTP listener.
ZEPHYR_FTS	off	Full-text search: off → index → run zephyr-backfill-fts → on. The staging order is load-bearing.
ZEPHYR_THREADS	off	Conversation threading: same off/index/on staging, with zephyr-backfill-threads.
ZEPHYR_ALARMS	off	Email reminders for calendar alarms. Needs

		ZEPHYR_RECOVERY_FROM.
ZEPHYR_RECOVERY_FROM	none	Sender for recovery, signup-verification and alarm mail, e.g. postmaster@example.com. Unset = those mails cannot be sent, and the features refuse rather than pretend.
ZEPHYR_RECOVERY_PROBE	full	Recovery-address liveness check: full (MX + port-25 probe), dns (MX only), off.
ZEPHYR_SIGNUP	off	The invite-gated signup endpoint.
ZEPHYR_P11	off	At-rest encryption surfaces (section 10).
ZEPHYR_ASSIST	off	The writing-assistant endpoints.
ZEPHYR_AI_ENDPOINT / ZEPHYR_AI_MODEL / ZEPHYR_AI_API_KEY / ZEPHYR_AI_CONSENT_REF	off	The AI backend for classification/assist. An external backend (key set) without a consent reference is configured and unusable, by design.
ZEPHYR_LINK_DOMAIN / ZEPHYR_LINK_ADDR	off	Link-attachments: the **separate** download origin and the download listener bind. Both are needed; the separate origin is the security argument.
ZEPHYR_LINK_AV_CLAMD	none = links not scanned	clamd for download scanning (fails closed by default via links.av_fail_open=false).
ZEPHYR_JMAP_WS	off	JMAP over WebSocket.
ZEPHYR_JOBS	off	The async job store and worker (bulk imports, restores, purges). The console's job-backed actions need it.
ZEPHYR_RESTORE_COMMAND	none	Absolute path to the restore program the job worker

		runs for mailbox restores.
`ZEPHYR_WEBHOOKS`	`off`	Outbound event callbacks (section 20). Off is a byte-for-byte no-op; the `/webhooks` routes answer 501.
`ZEPHYR_WEBHOOK_KEY_FILE`	none	The master key that seals endpoint secrets. **Back it up** — without it every registered endpoint stops being delivered to. A node with webhooks on and no key file still delivers; it cannot register.

Admin API, sessions and identity

Variable	Default	What it does
`ZEPHYR_ADMIN_CLIENT_ID` / `ZEPHYR_ADMIN_CLIENT_SECRET`	none = admin API answers 503	The OAuth2 client credential. Fail-closed by design; `ZEPHYR_ADMIN_AUTH=off` is the explicit dev-only escape.
`ZEPHYR_ADMIN_CLIENT_SCOPES`	`platform:admin`	The machine client's grants, e.g. `tenant:admin@tenant:<uuid>`.
`ZEPHYR_ADMIN_PLATFORM_ACCOUNTS`	none	Accounts treated as platform admins when a person signs in to the console.
`ZEPHYR_ADMIN_TOKEN_FAIL_MAX`	`10` per 15 min per source	Token-endpoint brute-force brake.
`ZEPHYR_CRED_CACHE_SECS` / `ZEPHYR_CRED_CACHE_MB`	`300` / `4`	The verified-credential cache (skips repeated password hashing on busy JMAP). Raise both together on busy hosts.
`ZEPHYR_ARGON2_THREADS`	CPU count	Size of the password-hashing pool.
`ZEPHYR_OAUTH_ISSUER` / `ZEPHYR_OAUTH_AUDIENCE` /	off	External IdP token validation, on the mail protocols, JMAP and the

`ZEPHYR_OAUTH_JWKS_URI`		self-service `/account/*` surface — never the admin API. All three or startup is refused.
`ZEPHYR_OAUTH_ADDRESS_CLAIM`	`email`	Which token claim names the mailbox. The value must contain an `@`.
`ZEPHYR_OAUTH_REAUTH_MAX_AGE_SECS`	`300`	How recently the IdP must have authenticated the person for a bearer token to satisfy the sensitive self-service operations. Zero or unparseable falls back to the default, never to "no limit".
`ZEPHYR_LDAP_URL`, `ZEPHYR_LDAP_DOMAINS`, `ZEPHYR_LDAP_BASE` or `ZEPHYR_LDAP_USER_DN`, `ZEPHYR_LDAP_BIND_DN/_PASSWORD`, `ZEPHYR_LDAP_CA_FILE`, `ZEPHYR_LDAP_AUTO_PROVISION`	off	Directory mode (section 10). A partial configuration refuses to start.

Multi-cell

Variable	Default	What it does
`ZEPHYR_CELL_ID`	`cell-dev-1`	This cell's identity in the routing map. Set it before registering or binding anything.
`ZEPHYR_CELL_FEED_ADDR` / `_CA` / `_CERT` / `_KEY`	off	The mTLS listener serving this cell's map to peers. No unauthenticated mode exists.
`ZEPHYR_CELL_PEERS`	none = not federated	Peers to pull the map from, `host:port:server_name`, comma-separated. Full mesh — every cell lists every other.
`ZEPHYR_CELL_PROXY_CA`	none	Trust anchor for cross-cell session proxying (the issuer of peers' public TLS certificates).

ZEPHYR_CELL_SERVING_STALENESS_SECS	21600 (6 h)	How stale this cell's map may be before it stops claiming domains.
ZEPHYR_RECIPIENT_DIGEST_SECS	300	Rebuild interval for the recipient digest relays check.

DNS serving (zephyr-dnsd and the control plane)

Variable	Default	What it does
ZEPHYR_DNS_ADDR	127.0.0.1:5353	The nameserver's bind. Publishing a zone is delegation, not binding.
ZEPHYR_DNS_FEED_HOST / _PORT / _CA / _CERT / _KEY	unset = direct mode	Feed mode: pull zones from the control plane over mTLS; the daemon then holds no database session. Port default 8443.
ZEPHYR_DNSSEC	off	sign enables DNSSEC (direct mode). A requested-but-broken configuration refuses at startup rather than serving unsigned.
ZEPHYR_DNS_MASTER_KEY_FILE	none	The DNSSEC master key file. **Back it up** (section 7). On the control plane, also enables signing for the feed.
ZEPHYR_DNS_RRL	on	Response-rate limiting; only the literal off disables it.
ZEPHYR_DNS_NS / ZEPHYR_DNS_HOSTMASTER / ZEPHYR_DNS_MAIL_HOST / ZEPHYR_DNS_MAIL_IPV4	derived per domain	Zone shape: NS names, SOA contact, MX target and its address.
ZEPHYR_DNS_DMARC_POLICY / ZEPHYR_DNS_DMARC_RUA / ZEPHYR_DNS_TLSRPT_RUA / ZEPHYR_DNS_MTA_STS_ID	none / postmaster@{domain} / none / 1	Generated policy records for hosted domains.
ZEPHYR_ZONE_FEED_ADDR / _CA / _CERT / _KEY	off	The control-plane end of the zone feed (set on zephyrd). All required together.

Licensing and backup tooling

Variable	Default	What it does
`ZEPHYR_LICENSE_FILE`	`/etc/zephyrab/license.json`	The license file path. Missing = community; unreadable = invalid, loudly.
`ZEPHYR_CELL_CLUSTER_FILE`	`/etc/zephyrab/cell.cluster`	For `zephyr-backup`: the source cell's cluster file (read-only use).
`ZEPHYR_SCRATCH_CLUSTER_FILE`	`/etc/foundationdb/fdb.cluster`	For `zephyr-backup`: the isolated restore target. Never the same cluster as the source; the tool checks.
`ZEPHYR_FDB_BACKUP_URL`	`file:///data/zephyr-backup/fdb`	The fdbbackup container URL.
`ZEPHYR_BLOB_BACKUP_DIR` / `ZEPHYR_BACKUP_STATE_DIR`	`/data/zephyr-backup/blobs` / `.../state`	The append-only blob mirror and the state directory.

17. Migration from another server

`zephyr-migrate` moves mailboxes into ZephyrAB from a live IMAP server or from files on disk, plus filters, calendars and contacts. One tool, one set of rules, whatever the source.

The rules every source shares

- **The source is read-only, by construction.** The IMAP client issues only reading commands — a test asserts its command log contains nothing that could mutate — and the file readers never open anything for writing. A Maildir reader is conventionally supposed to move messages from `new/` to `cur/`; this one deliberately does not, because that would modify a mailbox that may still be live and destroy the unread state the import exists to preserve.
- **Messages are never re-encoded.** The bytes stored are the bytes the source held.
- **Idempotent and resumable.** A resume mark rides the same transaction as the message, so "stored" and "recorded as stored" cannot come apart. Re-running a job copies nothing already copied; keep `--job` stable to resume.
- **Verification re-reads both ends.** `verify` fetches the source again, reads the destination back through the same call mail clients use, and takes its index of what was written from the database — never from the migrator's memory. It compares body hashes, flags, dates, folders, and then the **count**, which catches a folder where nothing was copied at all. The verdict is pass only with zero discrepancies and every source message accounted for. There is no tolerance.
- **Exit code 0 means the run passed.** Anything else is non-zero, so a pipeline cannot mistake a partial migration for a finished one.

From a live IMAP server

```
zephyr-migrate run --source-host imap.oldhost.example --accounts
roster.tsv \
    --job cutover-2026 --report report.json
```

`roster.tsv` is `source_user <TAB> source_password <TAB> dest_address`. For Gmail and Microsoft 365 use `--source-auth xoauth2`; the password column then carries an OAuth access token (this tool does not mint tokens). `plan` is `run` with dry-run forced on.

The throttle defaults protect a production source: 20 messages per second **globally** across the whole run (`--rate`), 2 accounts in parallel (`--account-concurrency`). Raise them only after measuring the source. Verification re-reads every body, so run-plus-verify costs the source about twice the reads.

Folder mapping honours special-use attributes at any depth (so a provider's "Sent Mail" becomes Sent), with `--folder-map SRC=DST` and `--exclude` for the rest.

From files on disk

```
zephyr-migrate import-local --address alice@example.com --maildir
/srv/vmail/alice/Maildir
zephyr-migrate import-local --address alice@example.com --mbox
/var/mail/alice --mbox-variant mboxrd
zephyr-migrate import-sieve --address alice@example.com --sieve
/home/alice/sieve
zephyr-migrate import-dav   --address alice@example.com --
calendars ./cal --contacts ./vcf
```

- **Maildir** — Maildir++ layout only (folders as dot-prefixed siblings, the Courier/Dovecot default). The nested-directory layout is **refused by name**: in it, a directory holding `cur/` is a folder, while in Maildir++ the same directory is somebody's stray backup that must not be imported as mail, and nothing in the tree distinguishes them. `tmp/` is never read. A message in `new/` is unseen regardless of its filename. Delivery dates come from the filename's timestamp, falling back to file mtime — that order matters, because a tree copied with `cp -r` has every mtime set to the afternoon of the copy.
- **mbox** — the case where a wrong split corrupts mail *silently*: a body line beginning `From ` is indistinguishable from a message boundary unless the writer escaped it, and the escaping convention differs by variant in ways the file itself does not declare. **`--mbox-variant auto` (the default) decides only when the decision cannot matter**: it scans for any `>From `-style line, and if one exists — meaning the file's contents genuinely depend on which tool wrote it — the import **stops and asks** rather than guessing and corrupting. `mboxrd` is the usual answer for qmail/getmail spools. Two further refusals: a `From ` line with no blank line before it (either reading loses mail; `--mbox-allow-unescaped-from` takes one reading, named for what it does), and a `Content-Length` header that disagrees with the scan. Read/answered/flagged state is read from the `Status:`/`X-Status:` headers.

- **Sieve** — every script is parse-gated with the same parser the delivery path uses; a script that does not parse is reported and not stored. **Nothing is activated by guessing**: a single file or a single script is obviously the active one; otherwise the source must say (an active-marker symlink), or no script is activated and the report says why.
- **Calendars and contacts** — a directory of files per collection. A multi-card `.vcf` is split byte-exactly (each card is a contiguous slice of the file). A `.ics` holding more than one event UID is **refused**: the events share one calendar wrapper and its timezone definitions, so splitting means re-serialising — which drops exactly the properties an importer does not model. Export one event per file, or migrate the calendar over CalDAV.

Add `--train-spam` (and `--train-ham-from-sent`) to turn the source's own filing into a spam corpus — the fastest path to a usable classifier that exists, taken at the one moment the user's years of filing decisions are in your hands. Refused unless the classifier is enabled in the migration's environment, rather than silently training nothing.

Coexistence: dual delivery, cutover, rollback

Per-account phases let you migrate gradually while both servers are live:

```
zephyr-migrate phase    --address alice@example.com --phase dual --
relay-to alice@legacy.example.com
zephyr-migrate phase    --address alice@example.com --phase cutover
zephyr-migrate rollback --address alice@example.com
zephyr-migrate status   --address alice@example.com
```

Phases: `pending → migrating → dual → cutover → rolled_back`. In **dual**, mail delivered here is also relayed to the legacy address (enable `ZEPHYR_COEXISTENCE=on` on the frontend), so the user can keep working on the old server while you verify the new one. **cutover** stops relaying — ZephyrAB is now authoritative. **rollback** replays exactly the mail that arrived after the cutover back to the legacy host, so backing out loses nothing. `--relay-host` pins the relay's next hop when the legacy MX does not point where the mail should go; its TLS follows `ZEPHYR_SMARTHOST_TLS`.

Honest limit: no real-world Dovecot or Courier spool corpus ships with the test suite — the fidelity tests build their own adversarial trees — so treat your first large import as an exercise of the tool as well as of your data, and read the report.

18. Troubleshooting

The admin API answers 503

Fail-closed, by design: no admin client credential is configured. The API refuses everything rather than running open. Check that `/etc/zephyrab/admin.env` exists with `ZEPHYR_ADMIN_CLIENT_ID` and `ZEPHYR_ADMIN_CLIENT_SECRET`, and that the zephyrd unit loads it as an environment file. The Solo installer mints it (step 9).

`ZEPHYR_ADMIN_AUTH=off` is the explicit development-only escape; never set it on a host that faces anyone.

A 403 mentioning the license

Two distinct titles, and each names its own fix:

- **`license limit`** — a provisioning ceiling was reached ("...allows 100 domains and this would be the 101st — raise the limit or retire a domain"). Check the console's License card for used-of-limit rows. A mailbox row reading "not yet counted" means the counter needs `zephyr-backfill-account-count`.
- **`license entitlement`** — the operation needs a feature the license does not list ("...does not include commerce — reading the catalogue and existing subscriptions keep working"). Community deployments hit this on delegation creation, plans, bulk provisioning, and multi-cell registration.

Neither ever affects mail flow, mailbox access, or reads.

I changed a switch and nothing happened

Nothing is instant, by design. A platform switch lands on every frontend within one refresh interval (about 10 seconds); a per-domain setting lands within the domain sweep interval (about 60 seconds). The Platform Settings page's footer shows when this node last read the switches — "never read on this node" is its own finding. Also check the other direction: turning a switch **on** requires the thing it drives to be configured. A blocklist switch with no local resolver, or a malware switch with no scanner address, is stored with a warning naming what is missing — the console shows the warning rather than pretending.

The upgrade refuses on the stored-filter scan

`zephyrab upgrade` ran the **candidate** binary's `zephyr-sieve-scan` and found stored Sieve filters that would stop parsing under the new parser. If it upgraded anyway, those users' mail would silently land in INBOX instead of their folders, with nothing telling anyone. Run `zephyr-sieve-scan` yourself to list the failing scripts and accounts, fix or remove them (the account's Filters panel in the console), then upgrade again. `--skip-sieve-scan` exists for hosts that run no mail store; on a host holding real filters it is how this hazard ships.

Verifying from outside the host

A server often **cannot reach its own public name**: many providers put the public address on a NAT layer that does not hairpin, so `curl https://mail.example.com` from the mail host itself times out — looking exactly like an outage that is not one. Verify from a different network vantage point. From the host itself, `curl --resolve mail.example.com:443:127.0.0.1 https://mail.example.com/...` tests the local service against the real certificate — useful, but it proves less than an external check.

Related trap: **unauthenticated probes lie.** The webmail page returns 200 and SMTP still greets while the database is unavailable, because neither touches it. Check an authenticated path, or just run `zephyrab doctor`, whose FoundationDB check asks directly. When the database *is* unavailable, the recorded usual suspect is a **full disk** — FoundationDB stops accepting writes without free headroom and does not recover on its own at high fill.

An account is locked

Five failed password attempts lock an address for fifteen minutes. Setting a new password clears the counter immediately — a reset that appeared "to do nothing" for fifteen minutes is why. Before resetting, ask what was failing: a misconfigured client retrying a stale password is the common cause, and a device credential used at the wrong door (a mail app pointed at a calendar credential) deliberately does **not** count toward the lockout — so if the user swears nothing is retrying, look for a real guesser in the logs. A directory outage never locks anyone (section 10).

DNS records are not propagating

`zephyrab dns-check --domain example.com --wait` polls and prints transitions. If a record stays "not seen yet" for more than an hour, it is usually not propagation:

- The record went into the wrong zone, or the **name doubled** — the tool prints fully-qualified names, and providers that auto-append the domain turn `_dmarc.example.com` into `_dmarc.example.com.example.com`, which resolves to nothing and looks exactly like lag.
- A DKIM TXT value over 255 characters was pasted as one string instead of quoted chunks.
- The record type is one the platform lists but does not probe (SRV, TLSA, the MTA-STS policy host) — `not checked` means nothing looked, not that it is missing.

Verify at the domain's actual authority (whoever answers its NS records) before blaming caches.

The signup form says the invite code is not valid

Unknown, expired and fully-used invites all answer identically — deliberately, so a leaked code list is not worth grinding (section 10). `zephyr-invite list` on the host shows the truth: uses remaining and expiry per invite. Mint a fresh one.

Where the logs live

```
journalctl -u zephyrd -n 100
journalctl -u zephyr-dnsd -n 100
```

Log verbosity follows `RUST_LOG` (the shipped units set `info`). Address logging can be redacted with `ZEPHYR_LOG_ADDRESSES=redacted`. One more reading trap: configuration under `/etc/zephyrab` is mode 0640 root:zephyr, so a non-root `ls` or `test -f` reports files missing that exist — check config as root, or you will "discover" that nothing is configured.

19. Sizing and architecture

This section is for the question "what machines do I need, and how do they fit together". Section 2 gives you the three installer presets; this one gives you the reasoning behind them, the measured numbers they rest on, and what changes as a deployment grows.

A warning about numbers before any of them. Everything here is either a design assumption from the architecture specification (labelled as such) or something measured on one reference deployment (labelled as such). They are not the same kind of fact. A measurement from one fleet on one provider's storage tells you the *shape* of a constraint reliably and the *magnitude* only approximately. Where nothing has been measured, this section says so rather than offering a plausible figure.

What you are sizing: the cell

The unit of sizing is the **cell** — one complete mail store, as section 5 defines it: a FoundationDB cluster, a blob store, a queue, and one or more frontends. Everything below is about sizing *one* cell, and then about when to stop and add another.

The architecture specification's design envelope is **about one million mailboxes per cell**, with the growth rule stated plainly: *add cells beyond ~1M mailboxes rather than scaling one cell past its tested envelope*. That number is a decision, not a hard limit — it is the population the reference design was modelled and load-tested against.

A cell is a scaling and blast-radius unit. It is not redundancy, and sizing it as though it were will disappoint you. A domain lives in exactly one cell. If that cell is down, that domain's mail is down, and no number of other cells changes that. What more cells buy is capacity past the envelope, containment (a bad deploy or a hot tenant hits one cell's population, which is why releases canary cell by cell), and residency. Redundancy *inside* a cell comes from FoundationDB replication and from having more than one frontend — not from the cell count.

The reference load model

The architecture specification models a cell of one million active mailboxes like this. These are **planning assumptions to be validated by load test**, and the specification says so in the same breath:

Assumption	Value
Inbound messages per user per day (pre-filter, including spam)	40
Accepted messages per user per day	15
Outbound messages per user per day	8
Peak factor	4
Average mailbox size	4 GB
Fraction of users holding an idle IMAP	0.2

connection

Which derives:

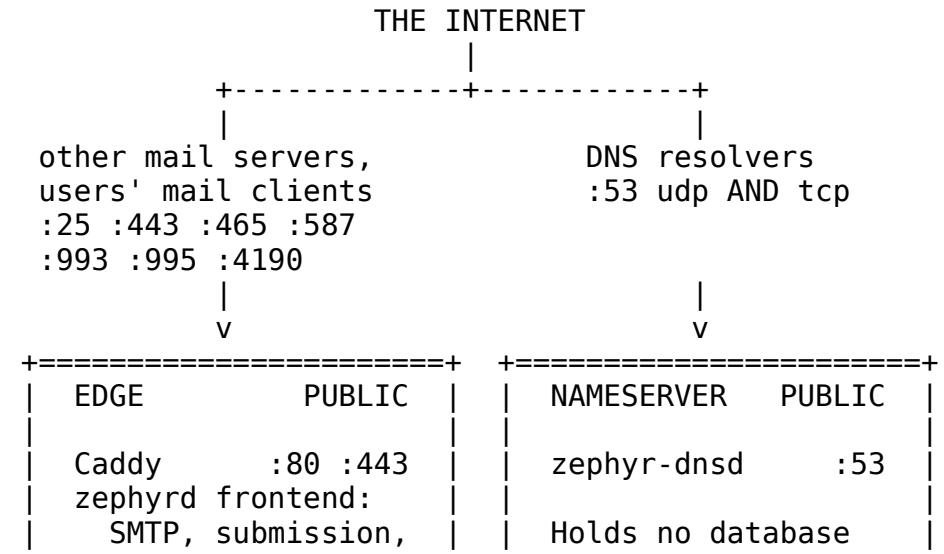
Derived figure	Value
Sustained inbound	**463 messages/second** (1,000,000 × 40 ÷ 86,400)
Peak inbound	**1,850 messages/second** (463 × 4)
Peak outbound	370 messages/second
Concurrent IMAP connections	200,000
Logical mail storage	4.0 PB
Physical after dedup and compression	2.4 PB (a ~0.6 factor for a typical corporate mix)
Metadata in FoundationDB	~15 TB (roughly 2–4 KB per message, plus indexes)

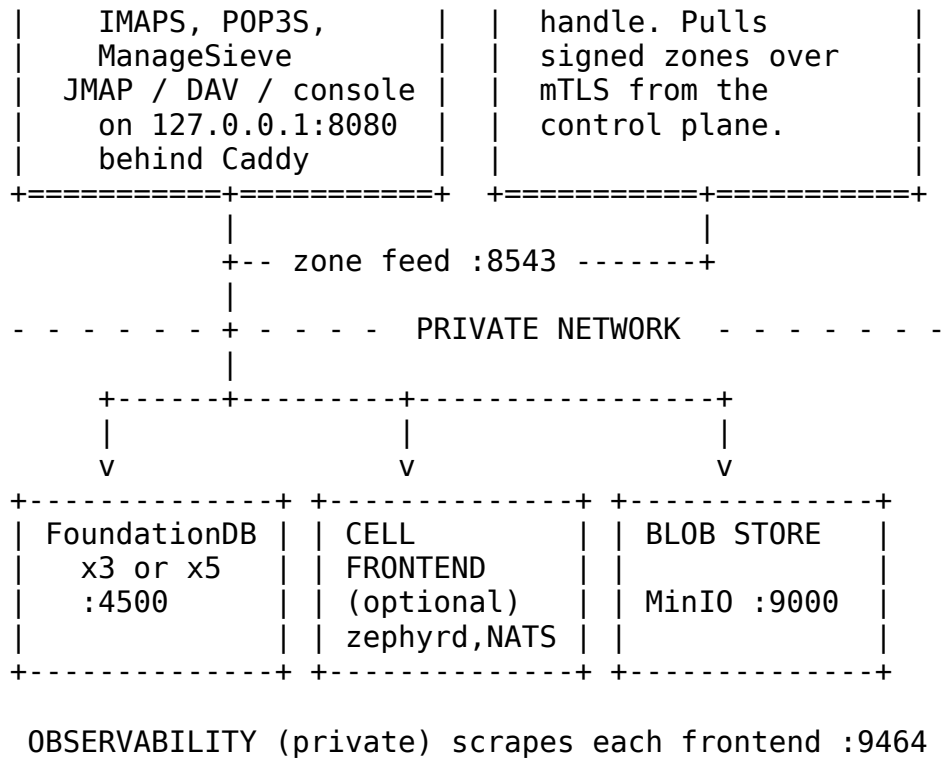
Read those two rates together, because they are the ones people confuse. **463/s is what the design sustains; 1,850/s is the busiest moment of a modelled day.** Sizing a cell for 1,850/s continuously would model no deployment that exists.

Two of these numbers are **assumptions that have never been validated at scale here**, and you should treat them as the least reliable rows in the table: the 2.4 PB physical figure and the 0.6 dedup factor. The reference deployment's load testing used synthetic bodies that deduplicated heavily, so its storage path was never the stressed one — a fact recorded at the time precisely so the throughput results would not later be quoted as storage results.

How the pieces fit together

One cell, in the shape the `standard` preset builds. Every box is labelled with whether it needs a public address.





The load path in words: mail and clients arrive at an **edge**, which terminates TLS and runs a `zephyrd` frontend. Frontends are **stateless** — they hold no mail — so they can be added, removed and restarted freely, and any of them can serve any request. All state is in the **FoundationDB** cluster (metadata, and message bodies at or under the 64 KiB inline threshold) and the **blob store** (bodies above it). **NATS** carries push notifications between frontends so that an IMAP IDLE or JMAP push session on one frontend learns about a delivery that landed on another. The **nameserver** is deliberately not part of that path at all.

Public and private: which host needs which address

This is the part that is easiest to get wrong and most expensive to get wrong, so it is a rule rather than a preference: **only the edge and the nameserver need public addresses. Everything else belongs on a private network.**

Role	Address	Listens publicly on	Notes
Edge	**Public**, with forward *and* reverse DNS that agree	25, 80, 443, 465, 587, 993, 995, 4190	The only host that must accept SMTP from the internet.
Nameserver	**Public**	53 **TCP and UDP**	Only if you run hosted DNS. Holds no database handle.
FoundationDB node	Private	nothing	Port 4500 on the private network, or loopback on a

			single-node cell.
Blob store	Private	nothing	MinIO on 9000, private or loopback.
Cell frontend (non-edge)	Private	nothing	Binds its protocol ports on **all interfaces**, on the assumption that its only interface is private. Do not give one a public address.
Observability	Private	nothing	Scrapes the frontends' metrics port.
Backup destination	Private	nothing	Should not be the machine it is backing up.

Cleartext IMAP (143) and POP3 (110) are the ones to check, because the two frontend shapes handle them differently and only one of them is safe on a public machine. A frontend built as a **public edge** binds them on loopback. An **internal** frontend binds them on all interfaces, because its only interface is meant to be private. A **single-node cell** — one machine that is both edge and cell — also binds them on all interfaces and relies on the firewall rather than the bind address to keep the internet off them, which is a thinner margin than loopback and worth knowing about if you ever edit that host's firewall by hand.

Three specific traps, each of which has cost somebody real time:

- **A public FoundationDB port is a total compromise, not a partial one.** FoundationDB has no per-key authorisation. Any client that can open the cluster can read every mailbox in it. This is why the `standard` preset **refuses** a host carrying both `edge` and `fdb` (or `blob`) roles by name — the plays open no firewall for those services, because they were written for hosts that are not reachable from the internet. The same reasoning applies to MinIO, whose only access control is its credential.
- **Opening port 53 is two rules, not one.** DNS is primarily UDP. A firewall rule listing 53 without saying which protocol opens only the TCP half — and the zone still resolves, because every resolver falls back to TCP after a timeout. The symptom is a nameserver that is mysteriously slow rather than one that is obviously broken. The Ansible layer keeps public TCP and public UDP ports in **separate lists** for exactly this reason.
- **Adding a firewall rule is the easy direction; removing one silently does not happen.** Rules that are meant to be gone must be *stated* as gone, not merely deleted from a list — a list is applied additively, so shortening it closes nothing, and the only

place the truth shows is the firewall's own status output. The deployment layer keeps a retired-ports list for this and applies it as explicit deletions.

Between cells, two ports are **peer-scoped** — open to named peer addresses only, never to the world: the cell-map feed (8443, mutual TLS) and the metrics port (9464). Cross-cell client session proxying deliberately needs no extra port: it lands on 993, 995 and 4190, which are public already, because the proxy dials TLS immediately and what crosses that connection is a user's password.

There is one property the generator enforces that is worth knowing about, because it catches a whole class of mistake: **generation fails if the firewall would open a port that nothing binds**, or if a cleartext protocol port would be bound on all interfaces *and* opened publicly. Those two things had genuinely drifted apart once — ports opened that nothing listened on, and cleartext logins bound on all interfaces with only a firewall default between a password and the network.

Four shapes

1. Single server (the `solo` and `small` presets)

Everything on one machine: FoundationDB at `single` redundancy, MinIO, `zephyrd`, and on the `small` preset a Caddy web front too.

Hosts	1 (`solo` installs onto the machine you run it on; `small` drives one host over SSH, plus optional observability and backup hosts)
Floors enforced by preflight	Refuses below **2 GiB RAM** and below **5 GiB free disk**; warns below **4 GiB RAM** and below **20 GiB free disk**
Disk floor asserted by the plays	8 GB for the database, 8 GB for the blob store — small enough that an ordinary root filesystem passes
Good for	One organisation, one or a few domains. The interview describes it as the right choice for one domain and up to a few thousand mailboxes.
What it does not give you	Any redundancy at all. `single` redundancy means the machine is the failure domain — the installer says so in as many words. Your recovery story is entirely the backup.

`solo` deliberately skips NATS: on one machine, `zephyrd`'s built-in FoundationDB-watch fallback covers IMAP IDLE and JMAP push, and a message bus between one process and itself buys nothing.

"A few thousand mailboxes" is a judgement, not a measurement. No single-machine cell has been load-tested to a ceiling here. What is certain is the direction of the first constraint you will meet — see *The numbers that actually bind*, below.

2. Three FoundationDB nodes (`standard`)

Hosts	3 `fdb`, exactly 1 `blob`, at least 1 `edge`, optionally `cell` frontends, `obs`, `backup` — so 5 hosts minimum
FoundationDB redundancy	`double`
Disk floors asserted by the plays	50 GB per database node, 100 GB for the blob store — explicitly *floors for a production start, not recommendations*
Good for	The smallest shape that survives losing a database node
What it does not give you	Headroom. Three is the minimum at which `double` redundancy can survive a node loss **and re-replicate onto the survivors**.

Three nodes is a real and supported shape. It is also the shape where the reference deployment met its most instructive wall, and you should size against that finding rather than rediscover it — see the next tier.

3. Five FoundationDB nodes (`standard`)

Same shape, five database nodes. This is the configuration the reference deployment certified.

Measured: a five-node cell held **1,900 messages/second for 24.7 hours** — the full modelled peak, with 200,000 concurrent IMAP IDLE connections, delivery p95 averaging 0.35 s and never exceeding 0.74 s against a 5 s target, and every one of 171 million accepted messages reconciled. On the *same* workload, three nodes **could not**: durability lag climbed monotonically (5 s → 36 → 132 → 413 s) inside twenty-five minutes, and the run diverged.

The differentiator was memory, and specifically page cache against dataset size. With three nodes the dataset reached roughly 143 GB per node against about 25 GB of cacheable memory, so storage-server reads fell to disk at 1–3 ms each and the servers could no longer serve reads and flush writes at the same time. The durability queue then grows without bound. With five nodes the same dataset was about 73 GB per node and stayed cacheable.

Two things follow, and they are the most useful sizing facts in this section:

- **It is monotonic in data volume**, which is why short tests pass and long ones fail. A twenty-minute run on a small dataset proves nothing about a week. This is also why the installer refuses database-node counts other than 3 or 5: those are the shapes that have actually been built and measured here, and it declines to guess at others.

- **The lever is RAM per node and node count, not tuning.** Everything else was tried and refuted by measurement on that deployment: storage-process count, filesystem, proxy and resolver counts, transaction-log placement. If a cell is falling behind and the dataset has outgrown memory, the answers are more memory per node, more nodes, less per-message work, or a lower rate.

Good for	The modelled million-mailbox cell, including its peak
What it does not give you	Cross-site survival. Five nodes in one place is still one failure domain for anything that takes the site out.

What the reference deployment actually ran

Since the tiers above give floors and not sizes, here is the hardware behind the 24.7-hour result, measured on the machines rather than quoted from a provisioning note. **This is one fleet on one provider, on network-backed volumes. It is a data point, not a specification** — read it with the caveats that follow, which matter more than the numbers.

Role	vCPU	RAM	Data volume
FoundationDB node (×5)	32 or 64	125 GiB	~492 GB, five volumes striped, ext4
Frontend	48	188 GiB	50 GB
Blob store	32	62 GiB	6.4 TB
Observability	32	62 GiB	50 GB
Edge	16	31 GiB	100 GB

Four things to take from that table, in order of usefulness:

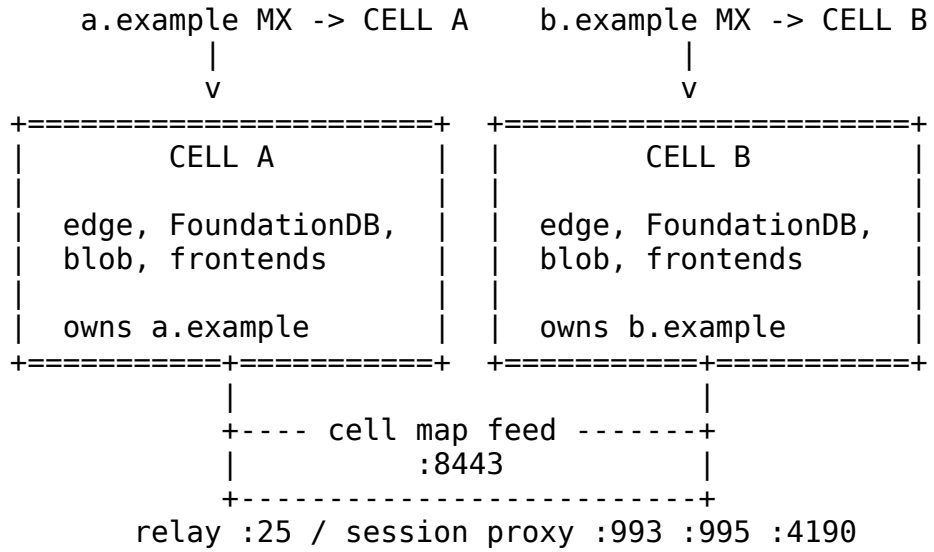
- **RAM on the database nodes is the number that matters**, and it is the one to scale against your expected dataset, not against your message rate. The tier-3 finding is entirely a memory finding.
- **CPU was never the binding constraint.** Those database nodes sat near-idle throughout; the cluster was not even uniform (some 32 vCPU, some 64) and it never showed. Do not read 32–64 vCPU as a requirement, and do not reach for more CPU first when a cell is struggling.
- **The frontend's 188 GiB is not a requirement either.** Frontends are stateless; their memory is working set, and the largest single consumer is password hashing, which is bounded by a fixed-size pool (`ZEPHYR_ARGON2_THREADS`) rather than by load.
- **The blob store's 6.4 TB was mostly unused** at the time of the run, because bodies at or under the inline threshold never reach it. Size it from your expected large-message volume, not from total mail volume.

Scaling this down is reasonable and untested here. Scaling the **database nodes' memory** down is the one change most likely to cost you the endurance result, because that is the constraint the result turned on.

4. More than one cell

Section 12 covers running multiple cells in full — the routing map, `zephyr-cell-ops`, secondary MX, tenant moves, and the version-skew rule. Sizing-wise:

When	Past roughly a million mailboxes; or to contain blast radius; or for data residency
What each cell is	A complete cell in its own right — any of the shapes above. Cells need not be the same size as each other.
What it costs	Each cell is a separate database, blob store, backup and restore story. Two cells is two of everything to operate, not one bigger thing.
What it does not give you	**Redundancy for a domain.** This bears repeating because it is the commonest misreading: a domain is bound to exactly one cell. A second cell does not make the first one's domains survive its loss.



Inbound routing is **DNS**: a hosted domain's MX names the cell that owns it. The link between the cells is the backstop — a cell that receives mail, or a client login, for a domain it does not own hands it to the cell that does (section 12).

Cells learn each other's routing map over a **pull-only** mutual-TLS feed. Two consequences for anyone laying out a topology: the peer mesh must be stated in **both** directions (a hub shape silently fails to propagate — one cell listing two peers while they list only each other looks like a hub and is not one), and the feed carries **pointers only** — cell records and

domain bindings. It carries no account, no credential and no message. A peer that pulls it learns where things are, never what they contain.

Multiple edges

Frontends are stateless, so a cell can have more than one edge, and the `standard` preset allows it (`at least one edge`). What each additional edge buys and costs is worth separating carefully, because inbound and outbound are not symmetric.

Inbound: more edges is straightforwardly good. Publish them as MX records for the domain and senders will use them. Each is a full `zephyrd` frontend of the same cell, reading and writing the same database, so any of them can serve any user. Losing one costs capacity, not availability.

Outbound: more sending addresses is a cost, not a benefit, and it is the one people get backwards. Sending reputation is per IP address and is earned over months of consistent, wanted mail. A fleet where every node sends directly has as many *cold* reputations as it has nodes — and cold mail is accepted and filed to spam, which is invisible from the sending end. So:

> **Split inbound from outbound.** A cell needs its own inbound face, because inbound routes by DNS. It does **not** need its own outbound face. Route every node's outbound mail through one warmed host with `ZEPHYR_SMARTHOST` (section 8), and keep one reputation to maintain instead of N.

That is not a theoretical preference. On the reference deployment, weeks of work went into one sending address — reverse DNS agreeing both ways, authentication, feedback-loop enrolment, and the spam-placement fight with two large receivers. A new sending address starts all of it at zero, and starts it cold, which is the state large receivers treat with the most suspicion.

Two related facts about network layout that are easy to miss:

- **A host with no public address may still reach the internet through NAT, and it will do so wearing somebody else's reputation.** On the reference deployment the private hosts egress through the edge's public address. That is convenient and it is a hazard: anything one of those hosts sends to the internet carries the edge's reverse DNS, its SPF authorisation and its accumulated reputation. A test or staging cell emitting junk that way damages the production domain. Point such a host's queue at a smarthost, or leave it with no signing key so its queue runner does not start.
- **Many providers do not hairpin their own NAT**, so a host cannot reach its own public address. This is a property of the provider, not of your configuration, and it is not fixable from inside the host. It matters because it makes "verify from the machine itself" unreliable — see section 18 for how to verify from outside instead. Do **not** work around it with a hosts-file entry pointing the public name at loopback: that makes a check silently bypass the NAT, the firewall and the proxy binding while still looking like an external check.

Certificates on multiple edges: the A record must name every edge. Every edge serves the *same* public name, each runs its own Caddy, and each will try to answer an ACME challenge for it. So the name has to resolve to all of them — or to a load balancer in front of them. **An edge the name does not resolve to never obtains a certificate, and Caddy retries for ever with every log line looking like progress**, which is the worst shape a failure can take. The installer raises this as an advisory when a plan carries more than one edge; do not skim past it.

Two related certificate facts. TLS on the mail listeners **arrives after DNS, not at the end of the install**: until the name resolves to an edge and answers a challenge, `zephyrd` binds port 25 (mail arrives in the clear) and port 587 *refuses* AUTH rather than prompting for a password over an unencrypted channel — so nobody can send yet. That is the honest state, not a fault; the hourly certificate-sync timer picks the certificate up on its own. And if you publish DANE/TLSA records, note that they commit to a *key*, so any renewal that re-keys invalidates them unless key reuse is configured (section 7).

The numbers that actually bind

These are the measured constraints, in the order you are likely to meet them. Each is from one reference deployment on one provider's storage; trust the shape, verify the magnitude.

Disk fills faster than the database's own metric says. FoundationDB's storage engine sits on top of the logical key-value volume with its own B-tree, write amplification and free-page churn. Measured: **28.9 GB of logical KV occupied 88.7 GB of disk — about 3.1×**. So the rule is absolute: **size capacity from the disk series, never from the logical KV metric.** A run sized from the KV figure once filled its disks and stopped accepting writes, and FoundationDB does not recover on its own at high fill. Measured growth on the five-node cell at 1,900 msg/s was about 3.54 GB/hour per node — which over seven days is 594 GB per node, against 492 GB volumes. At the sustained 463 msg/s the same arithmetic gives 145 GB per node over a week, or 29% of those volumes. Do that arithmetic for your own rate and retention before you buy disks.

Memory on database nodes is the ceiling that ends long runs. See tier 3 above. The failure is not a crash; it is a durability queue that grows and never recovers, and it appears only once the dataset outgrows cache — which is to say, only in long runs.

Storage barriers are the write-path currency, and the filesystem changes them. Measured on an identical five-disk striped volume: **ext4 249 barriers/second (p50 3.62 ms), xfs 220 (4.36 ms), btrfs 193 (4.88 ms)** — ext4 about 29% ahead of btrfs. FoundationDB prefers ext4 or xfs, and the deployment layer builds ext4. This is a modest difference, not a transformative one; do not expect a filesystem change to rescue a cell that is short of memory.

Striping buys capacity, not IOPS. Measured aggregate per-VM I/O budget on that provider was about **27,000–29,000 IOPS**, and it was the same whether one volume or five striped volumes carried the load. Five volumes gave five times the space and no more operations per second. If your provider meters I/O per instance, adding volumes will not raise the ceiling.

CPU was never the constraint. On the reference fleet the database nodes sat near-idle throughout; the constraints were memory and disk. That is one deployment's experience, not a law — but it does mean CPU is the last thing to buy more of when a cell is struggling, not the first.

Single-node object storage saturates on small writes. Measured: about **1,750 PUT operations/second** before the blob host became the bottleneck. This is much less alarming than it sounds, because of the inline threshold: bodies at or under `ZEPHYR_BLOB_INLINE_MAX` (64 KiB by default) never reach the object store at all — they ride inside the delivery transaction in FoundationDB. So the blob store sees only large messages. Lowering the threshold moves load from the database to the object store and raising it does the reverse; both directions have been measured to have a real cost, and the default is the balance point that was chosen.

Distributed object storage is not a settled recommendation here. Erasure-coded and multi-node MinIO have been measured, the results were not conclusive, and the deployment layer builds single-node. If you need the blob store to survive losing its host, that is a design decision to make deliberately with your own testing — the reference deployment has not made it for you.

What to measure before you believe a size

Nothing above substitutes for measuring your own deployment, and the tools ship with it:

What	How
Is this deployment healthy right now	`zephyrab doctor` (section 2) — read-only, safe anywhere, exits non-zero on any failure
What is it actually doing	The metrics endpoint and the shipped alert rules (section 14)
How much is each tenant using	`zephyr-usage` (section 15)
Disk growth over time	Sample free space on the database nodes on a timer and keep the series. **Do this before a long run, not during one** — an earlier run's death was a surprise precisely because nothing had recorded its growth curve.
Does a proposed run fit	Compute `rate × duration × bytes-per-message × replication` against free space on every database node, and refuse to start if it does not fit. Capacity is a pre-flight check, not something to notice at 95% full.
Will it hold	A long run. **A short hold can demonstrate instability but can never prove**

sustainability — the page-cache wall is monotonic in data volume, so twenty minutes on a small dataset tells you nothing about a week.

One last piece of discipline, learned expensively: when you change a shared resource — a filesystem, a node count, a setting — and measure the result, ask first what the metric you are watching is a property **of**. Changing one of two writers to a shared database and observing no change proves nothing about the change. Vary one thing at a time, and know which thing the number belongs to.

20. Outbound webhooks

ZephyrAB can call an HTTPS endpoint of yours when something happens, so an external provisioning or billing system does not have to poll. This is off by default and is a byte-for-byte no-op when off: the provisioning, domain and job transactions run exactly as they did before, and the `/webhooks` routes answer `501`.

Turning it on

```
# /etc/zephyrab/zephyrd.env
ZEPHYR_WEBHOOKS=on
ZEPHYR_WEBHOOK_KEY_FILE=/etc/zephyrab/webhook-master.hex
```

Mint the key and lock it down:

```
openssl rand -hex 32 > /etc/zephyrab/webhook-master.hex
chmod 600 /etc/zephyrab/webhook-master.hex
```

Back that key up. Endpoint secrets are sealed under it; without it they are unreadable, every registered endpoint stops being delivered to, and every integrator has to be issued a new secret.

You may point `ZEPHYR_WEBHOOK_KEY_FILE` at the same file as `ZEPHYR_DNS_MASTER_KEY_FILE`. They are separate variables so that this is a decision rather than a default — a node holding the DNSSEC master key for an unrelated reason should not silently become able to mint webhook secrets.

A node with webhooks on and **no key file still delivers**; it just cannot **register**, and says so with a 501 naming the variable. That is the shape of a multi-node deployment where one host holds the key material.

The startup line to look for is `webhook delivery on; endpoint snapshot warmed`. If you instead see `the endpoint list could not be read at boot; events are DROPPED until a refresh succeeds`, events are being **lost, not delayed**.

The events

Event	When

`job.completed`	An async job reaches a terminal state
`account.created`	An account is provisioned — signup, admin API, bulk, or LDAP auto-provision
`account.deleted`	An account is unprovisioned
`account.status_changed`	An account's status is set (carries `from` and `to`)
`domain.created`	A domain is registered
`domain.deleted`	A domain is removed
`domain.verified`	**Subscribable and never fires** — see below

`job.completed` carries the bare `Job` object with no envelope, because that is what the API specification declared before any of this existed and an integrator may already have written against it. Every other event carries an envelope with `id`, `event`, `occurredAt`, `subject` and `data`. **Dispatch on the `X-Webhook-Event` header, never on the body's shape.**

`domain.verified` never fires, as of this version. A domain's verification status is derived live from public DNS on every read and is never stored — deliberately, because a stored observation would let the control plane report a domain as verified while its mail was failing. So there is no transition to fire on. It stays in the catalogue, parseable and documented as unfired, so that a subscriber whose stored list names it keeps working on the day that changes.

Verifying a delivery

Every delivery carries three headers:

Header	Meaning
`X-Signature`	`t=<unix-seconds>,v1=<hex>`
`X-Webhook-Event`	The event name
`X-Webhook-Id`	The delivery id — **stable across retries**, and therefore your deduplication key

`v1` is `HMAC-SHA256(secret, "{t}.{raw body}")`. The timestamp is inside the signed material, so a captured body cannot be replayed indefinitely and the timestamp cannot be edited without breaking the signature. It is deliberately the same scheme several well-known services use, so a library you already have will verify it.

Two things receivers get wrong: **sign the raw body**, before any parse-and-re-serialise (a round-tripped body is not the bytes that were signed); and **the secret is the hex string as issued**, used as ASCII key material, not the 32 bytes it decodes to.

What is promised

At-least-once, and unordered.

- A receiver that answers `200` after its network drops the response *will* be sent the same delivery again. **Deduplicate on `X-Webhook-Id`.**
- Retries mean a later event can arrive before an earlier one. **Order by `occurredAt`**, never by arrival.
- Retry is exponential from 2 seconds, capped at an hour, **12 attempts** (roughly a day), then abandoned. `2xx` is success. A `3xx` is **not followed and not a success** — a redirect is "ask somewhere else", which is exactly what must not be honoured. A `4xx` other than `408/429` is abandoned immediately: retrying an unauthorised or malformed delivery twelve times changes nothing and looks like an attack from your side.

Constraints on the URL

- **HTTPS only**, verified against the **system trust roots**. A private-CA endpoint will fail and there is no way to add a CA for this. Terminate on a publicly trusted certificate.
- **The hostname must resolve entirely to global addresses.** Loopback, RFC 1918, link-local (including the cloud metadata address), CGNAT and unique-local are refused — checked at registration *and again before every delivery*, because a hostname can be re-pointed at any time afterwards. If *any* address in the answer is non-global the delivery is refused; a mixed answer is a rebinding attack, not a coincidence.
- **Exactly one request, no redirects.**
- A URL carrying userinfo (`https://user@host/`) is refused rather than parsed: it means different things to different parsers, and a URL whose host depends on which library reads it cannot be checked.

Managing endpoints

Registration, listing, delivery history and deletion are admin API calls; the API manual has the request shapes. The operational facts:

- **The secret appears in the registration response and nowhere else.** It is sealed under the master key bound to the endpoint's id and there is no path that renders it again. Rotating means registering a new endpoint and deleting the old one.
- **An endpoint hears about its own subject**: `platform` hears everything, `tenant:<id>` that tenant's events, `domain:<id>` that domain's. You may only register a subject you administer. Deleting an endpoint drops everything queued for it.
- **Bounds**: 256 endpoints per deployment, a 64 KiB payload cap, and at least one event per subscription (an endpoint subscribed to nothing would never be called, so it is refused).
- A newly registered endpoint can miss events for a moment. The enqueue path works from a snapshot refreshed every 30 seconds, because reading the endpoint list inside the transaction that commits the thing being announced would put a conflict range on that list in every provisioning transaction. Registration republishes the snapshot immediately, so the window is small. **Deletion is exact even so** — the worker re-

reads the endpoint to get its secret, so a delivery for an endpoint that has gone is dropped rather than sent.

Known defect, as of this version: a tenant-scoped endpoint may miss `domain.created` and `domain.deleted`. Register a platform-scoped endpoint if you want domain events. The cause is not in webhooks: a domain record's tenant field holds a tenant *name* when the domain was created through the global domains route and a tenant *uuid* when it was created through the tenant-scoped one. Where it holds a name, no grant can name it and a correctly registered tenant endpoint silently receives nothing. For the same reason, do not correlate domain events with account events on `tenantId`. Platform-scoped endpoints are unaffected. The fix belongs in the domain record rather than here, and may well have landed by the time you read this — check the release notes.

Operating it

Metric	What it means
`zephyr_webhook_abandoned_total`	**Alert.** A subscriber was given up on after the full backoff — their integration has silently stopped.
`zephyr_webhook_refused_total`	**Alert.** A registered URL now resolves inside a private network: DNS rebinding, or an endpoint re-pointed somewhere it should not reach. No request was made.
`zephyr_webhook_dropped_total`	Events **lost, not delayed** — an oversize payload, or a node whose endpoint snapshot has never been read.
`zephyr_webhook_failed_total`	Expected traffic (a receiver restarting). Watch as a *rate against* delivered, not absolutely.
`zephyr_webhook_delivered_total` / `_queued_total`	On a node with no endpoints these stay at zero forever, which is the whole no-op claim in two numbers.

`consecutiveFailures` and `lastError` on the endpoint record are the per-subscriber view; a success clears them, so a non-zero value means *now*.

A queue that is not draining: read `zephyr_webhook_refused_total` and the endpoint's `lastError` first. The commonest causes are a certificate that no longer validates and a hostname that now resolves privately, and neither produces a log line at the receiver. Individual stuck deliveries need nothing done — deliveries are leased for 60 seconds and a worker that dies holding one leaves it to be re-claimed, which is the same mechanism that makes every delivery possibly arrive twice.

21. Glossary

- **Account** — a mailbox: an address, credentials, quota, filters, calendars and contacts. **Alias**: an extra address for the same mailbox.
- **ARF** — Abuse Reporting Format; the standard shape of complaint reports from feedback loops. Read by `zephyr-arf-ingest`.
- **Blob** — a stored message body above the inline threshold, kept in the object store, addressed by its content hash and shared between identical copies (deduplication). The **inline threshold** (64 KiB by default) is the size at or under which a body is stored inside FoundationDB instead.
- **Cell** — one complete mail store: a FoundationDB cluster, a blob store, a queue and frontends. A scaling unit; a domain lives in exactly one cell.
- **clamd** — the ClamAV scanning daemon ZephyrAB talks to for malware scanning.
- **DANE / TLSA** — DNSSEC-protected certificate pinning for mail transport: a DNS record committing to a mail server's public key.
- **Delegation** — an admin grant: a scope bound to a subject (a tenant or a domain). Immutable; changed by issuing a new one and revoking the old.
- **DKIM** — a per-message cryptographic signature verified against a public key in the sender's DNS. A **selector** names which key.
- **DMARC** — the policy record telling receivers what to do with mail failing SPF/DKIM alignment, and where to send aggregate reports (**rua**).
- **DNSBL / RBL** — DNS-queried blocklists of sending IPs and domains.
- **Doctor** — `zephyrab doctor`, the read-only health check and support command.
- **FCrDNS** — forward-confirmed reverse DNS: the sending IP's PTR name resolves back to the same IP. A baseline requirement at large receivers.
- **FoundationDB** — the transactional key-value database holding all metadata and small message bodies. A **cluster file** identifies one cluster; the **scratch cluster** is the isolated one restores are drilled into.
- **Greylisting** — answering first contact from an unknown sender with a temporary failure; real servers retry, many spam senders do not.
- **Invite code** — a single- or multi-use bearer credential that gates signup. Printed once, stored only as a hash.
- **JMAP** — the modern JSON mail protocol (RFC 8620/8621); what the webmail speaks.
- **Lease** — how queue and job workers hold work: a claim with an expiry, so a dead worker's job is reclaimed instead of stuck.
- **Maildir / mbox** — on-disk mailbox formats other servers use; import sources for `zephyr-migrate`.
- **ManageSieve** — the protocol (port 4190) mail clients use to manage Sieve filters.
- **Milter** — the mail-filter socket protocol; ZephyrAB can hand inbound mail to an external filter through it.
- **MinIO** — the S3-compatible object store holding large message bodies.
- **MTA-STS** — an HTTPS-published policy demanding verified TLS to a domain's MX hosts.

- **MX record** — the DNS record naming which host receives a domain's mail, with a preference number (lower = tried first).
- **Plan (deployment)** — `zephyrab.plan.json`, the reviewable description of a deployment the installer works from. **Plan (commerce)** — a class-of-service bundle in the console's Plans page. Context tells them apart.
- **Preset** — one of the installer's deployment shapes: solo, small, standard.
- **Protective / Operational setting** — the two kinds in the settings registry: protections combine as a union (any layer may turn one on, none may turn another's off); operational values resolve most-specific-wins, subject to domain locks.
- **PTR record** — reverse DNS: the name a sending IP resolves to.
- **Recipient digest** — a compact per-cell summary of accepted addresses, replicated between cells so a secondary MX can refuse nonexistent recipients without asking the owner.
- **Recovery address** — an external address a password-reset link goes to; checked for a working mail system before it is stored, and useless until confirmed.
- **Scope** — an admin role label (`tenant:admin`, `helpdesk:read`, …); always paired with a subject in a delegation.
- **Sieve** — the standard mail-filtering language; every account's filters are Sieve scripts, whether written by hand or generated by the vacation/forwarding form.
- **Smarthost** — a single host all outbound mail is routed through, concentrating sending reputation.
- **SPF** — the DNS record naming hosts allowed to send for a domain.
- **Stock / flow** — the two kinds of usage evidence: stock is a point-in-time count (mailboxes, storage), sampled daily; flow is events over time (messages, bytes), counted as they happen and unrecoverable if missed.
- **Subaddress** — `user+detail@example.com`: delivered to `user@example.com`, with the detail available to their filters.
- **Tenant** — an organization: the ownership and isolation boundary for domains, accounts and groups.
- **TLS-RPT** — daily reports from sending servers about TLS failures reaching your domain.
- **Tombstone** — a record marking something deleted so replication cannot resurrect it (domain unbinding), or deferring a deletion (blob garbage collection).
- **TOTP** — time-based one-time passwords; the second factor.
- **Webhook** — an HTTPS callback ZephyrAB makes to a system of yours when an account, domain or job event happens, so it does not have to poll. Signed, at-least-once, deduplicated on a delivery id (section 20).
- **Zone** — one domain's complete DNS record set, as served by an authoritative nameserver.