Release and Update Methodology

Release and Update Methodology

This document is the canonical reference for how Hermes SEG (Docker era) is released, distributed, and upgraded. It covers both sides of the loop: the **developer side** (how a release is cut) and the **admin side** (how a running install is updated). Read this if you are:

- An operator deciding how to upgrade a running Hermes install
- A developer adding a schema change, a one-shot migration, or a service config edit and need to know where it goes
- A maintainer cutting a new release tag
- A future Claude session that needs the mental model

Linked work: #218 (release engineering pivot), #221 (`system_update_docker.sh` orchestrator), #231 (beta scope).

Core concepts

Calendar versioning

Versions are `vYYMMDD` — a **label** named for a target/planning date. The digits are **not** the ship date and **not** the git tag date; a release is often named early and tagged months later as the work lands.

```
v260119  →  named for 2026-01-19, actually tagged 2026-05-30   (first Docker release, the baseline)
```

```
v260609  →  named for 2026-06-09                                (backup/restore/DR + upgrade-
tooling release)
v270315  →  hypothetical: a release named for 2027-03-15
```

The label does **not** auto-increment — there is no `v260120` just because a day passed. For the **actual** release timing of any tag, read the git tag date (`git log -1 <tag>`) or the GitHub Release page — never infer it from the digits.

The version stamp is stored in two `system_settings` rows:

Parameter	Value	Meaning
`version_no`	`'Docker'`	Code train identifier (post-bare-metal era)
`build_no`	`'v260119'`	Specific release tag the install is currently at

Image registry

Container images live at `ghcr.io/deeztek/hermes-<service>:<tag>`. The registry hostname is configurable via the `IMAGE_REGISTRY` env var in `.env` so the legacy GitLab CR (`hub.deeztek.com/dedwards/hermes-seg-docker-gl`) can still be used during the bootstrap period before ghcr.io is fully populated.

Build/push via `Docker/build-all-ghcr.sh` + `Docker/push-all-ghcr.sh` (ghcr) or `Docker/build-all.sh` + `Docker/push-all.sh` (legacy hub). **When you add a new service container, add it to BOTH the build-all and push-all scripts of the relevant pair** (the `build_image` call and the `IMAGES`/echo lists) — otherwise the release build silently omits it.

ghcr push auth: pushing to `ghcr.io/deeztek` needs a token with the **`write:packages`** scope; a plain `gh auth login` does NOT include it (its default scopes are `repo`/`workflow`/`read:org`/`gist`). Add it with `gh auth refresh -h github.com -s write:packages` then `gh auth token | docker login ghcr.io -u deeztek --password-stdin`, or use a classic PAT created with `write:packages` + `read:packages`. A brand-new package (e.g. the first push of a new service) lands **private** by default — set it Public on the GitHub package settings page for a public release.

Distribution repo

Code lives on GitLab (dev). Distribution is via GitHub (`deeztek/Hermes-Secure-Email-Gateway`) — issues, releases, container packages all there. A two-remote `scripts/git_release.sh` helper codifies the dev-push / release-push split.

Artifact taxonomy — where does what go?

Hermes ships **three categories** of changes between releases. Each has a dedicated home in the repo.

```
+-----------------------------------------------------------------+
|  CATEGORY 1: BASELINE                                           |
|  Lives at: config/database/hermes_install.sql                   |
|  Purpose:  Self-contained snapshot of the schema AT v260119.    |
|            Imported once on fresh install. Never modified by    |
|            upgrades.                                            |
+-----------------------------------------------------------------+

+-----------------------------------------------------------------+
|  CATEGORY 2: PER-RELEASE ARTIFACTS                              |
|  Lives at: updates/v<DATE>/                                     |
|    ├── README.md                  (what this release does)      |
|    ├── sql/schema_updates.sql     (schema deltas)               |
|    ├── cfml/*.cfm                 (one-shot CFML migrations)    |
|    └── scripts/*.sh               (one-shot bash one-shots)     |
|  Purpose:  Everything that changed between the PREVIOUS release |
|            and THIS release. Applied once, in order, by the     |
|            update orchestrator. Discoverable: one directory     |
|            tells you everything that release does.              |
+-----------------------------------------------------------------+

+-----------------------------------------------------------------+
|  CATEGORY 3: PERSISTENT POST-UPGRADE HOOK                       |
|  Lives at: config/hermes/var/www/html/schedule/post_upgrade.cfm  |
|  Purpose:  Cross-release CFML migrations gated by a `migrations` |
|            table. Each named block runs ONCE EVER per install.   |
|            Used for retroactive backfills and migrations that    |
|            don't cleanly bind to a single release.              |
+-----------------------------------------------------------------+
```

Categories side-by-side

	Baseline	Per-release	Persistent hook
File location	`config/database/hermes_install.sql`	`updates/v<DATE>/`	`schedule/post_upgrade.cfm`
Scope	Whole v260119 schema	One release's deltas	Cross-release backfills
When applied	Once, at fresh install	Once, when upgrading past that version	At every upgrade; per-block gated
Lifecycle	Replaced wholesale at major rebaselines (rare)	Lives forever in repo as history	Blocks accumulate over time; stay as documentation
Idempotency	Required (`IF NOT EXISTS` / `INSERT IGNORE`)	Required per artifact	Required per block (`migrations` table check)
Discoverability	One file, mysqldump-style	One dir per release	One file with named blocks

Choosing the right category

Use this decision tree when adding a schema/data/config change:

```
Is this a one-off "we noticed something needs fixing across all installs"
that doesn't bind to any specific release?
  YES → Persistent post-upgrade hook (category 3)
  NO  → continue

Is this a schema change, seed data, or config that NEW INSTALLS should have?
  YES → Add to baseline (category 1) — ALSO add to the current in-flight
        release's updates/v<DATE>/ (category 2) so existing installs get it
        on upgrade.
  NO  → continue

Is this a one-shot migration that only matters when transitioning past a
specific release (e.g., re-encrypt rows after a key rotation)?
  YES → Per-release artifact (category 2):
          - Pure SQL → updates/v<DATE>/sql/schema_updates.sql
          - Needs CFML (encryption, file I/O, API calls) → updates/v<DATE>/cfml/<name>.cfm
          - Needs host shell (docker exec, file moves outside containers) →
updates/v<DATE>/scripts/<name>.sh
```

Idempotency rules

Every artifact, in every category, MUST be safely re-runnable. The orchestrator does not track "did I apply this already" at fine granularity — it relies on each artifact to no-op when there's nothing to do.

SQL artifacts

Statement	Idempotency pattern
`CREATE TABLE`	`CREATE TABLE IF NOT EXISTS`
`ALTER TABLE ADD COLUMN`	`ADD COLUMN IF NOT EXISTS`
`ALTER TABLE DROP COLUMN`	`DROP COLUMN IF EXISTS`
`ALTER TABLE MODIFY COLUMN`	Naturally idempotent (re-applying same type is no-op)
`CREATE INDEX`	`CREATE INDEX IF NOT EXISTS`
`INSERT`	`INSERT IGNORE` (relies on UNIQUE key) OR `INSERT ... SELECT ... WHERE NOT EXISTS`
`UPDATE`	Value-gated `WHERE` clause that only matches pre-fix state (e.g., `WHERE value = 'Fail'`)
`DELETE`	Targeted `WHERE` that becomes empty after first application
`CREATE USER` / `GRANT`	Wrap in `PREPARE` block that checks `mysql.user` / `mysql.db` first (see `config/database/hermes_install.sql` for canonical pattern)

When in doubt, look at how an existing block in `hermes_install.sql` or a prior `schema_updates.sql` handled the same pattern. The CLAUDE.md "Schema Updates" section also documents these patterns inline.

CFML artifacts

Each `.cfm` file in `updates/v<DATE>/cfml/` or each block in `post_upgrade.cfm` must:

1. **Check before doing**: query something (a row's encryption state, a file's existence, a setting's value) to decide whether to act.
2. **Act**: do the work.
3. **Record completion**: either update a row to a state the check in step 1 will reject, OR insert into the `migrations` table.

The `migrations` table (created by Phase 5 framework, see below):

```
CREATE TABLE IF NOT EXISTS `migrations` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
```

```
    `completed_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`id`),
    UNIQUE KEY `uq_name` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Bash artifacts

Each `.sh` file in `updates/v<DATE>/scripts/` must:

1. **Precondition check**: test whether the operation has already been performed (file exists, container running, value set, etc.).
2. **Act**: do the work.
3. **Exit 0 on no-op**: a script invoked when its preconditions say "already done" must exit 0, not error.

The update orchestrator (`scripts/system_update_docker.sh`)

This script (target: #221) is the **single command an admin runs** to upgrade a Hermes install. It runs five phases in order; each is independently idempotent.

Phase 1 — Pull new code

```
cd /opt/hermes-seg-docker-gl
git fetch --tags
git checkout <target-tag>
```

The target tag is either passed as an argument (`./system_update_docker.sh v260601`) or auto-resolved by polling GitHub Releases API for `releases/latest`. If the working tree is dirty, the script refuses to continue.

Phase 2 — Update containers

```
docker compose pull          # only when HERMES_DOCKER_IMG_VERSION in .env bumped
docker compose up -d         # always (idempotent: only restarts services whose config
changed)
```

Image registry comes from `IMAGE_REGISTRY` in `.env`. Containers whose images didn't change are NOT restarted by `compose up -d`.

Phase 3 — Apply per-release artifacts

The orchestrator globs `updates/v*/` directories, sorts chronologically (calendar versioning sorts correctly as a string), filters to versions newer than current `build_no`, and applies each release in order:

```
For each release dir D in (sorted release dirs above build_no):
    For each file F in D/sql/*.sql       (alphabetical):
        docker exec -i hermes_db_server mysql -u root hermes < F
    For each file F in D/cfml/*.cfm      (alphabetical):
        curl -sf http://localhost:8888/updates/<dir>/cfml/<name>.cfm
    For each file F in D/scripts/*.sh    (alphabetical):
        bash F
    # Each release's schema_updates.sql advances build_no at its end via
    #   UPDATE system_settings SET value='<DATE>' WHERE parameter='build_no';
    # giving the orchestrator a natural cursor across releases.
```

If any artifact fails, the orchestrator aborts. Because `build_no` advances at the END of each release's `sql/` step, a failed-mid-release re-run picks up at the failed release and re-applies its full set (idempotency makes that safe).

Why this ordering: SQL changes the schema first so CFML and bash artifacts that depend on new columns/tables work. CFML before bash because CFML migrations often need the schema landing but Lucee's lifecycle is already running. Bash last for host-level fixups.

Phase 4 — Standard finalize steps

```
- Restart hermes_commandbox so it picks up admin/2/, schedule/, schema changes
- NC version-drift detection (#264): if .env's NCVERSION differs from
  the live `occ status` versionstring → run `occ upgrade` + rehydrate
  Hermes-required NC apps (user_oidc, mail, twofactor_totp,
  twofactor_backupcodes, external) + verify needsDbUpgrade=false
- Re-render *.HERMES config templates (reminder only in MVP; v2 will
  detect modified template files and trigger each regen endpoint)
```

NC upgrade is fully automated — no manual `occ upgrade` step needed after a release that bumps `NCVERSION`. The rehydrate loop covers NC's tendency to auto-disable apps it thinks are incompatible

with a new core version. Service restart triggers (other than commandbox) will be detected in v2 by diffing the working tree against the previous tag.

Phase 5 — Persistent post-upgrade hook

Last step: orchestrator calls the persistent hook.

```
curl -sf http://localhost:8888/schedule/post_upgrade.cfm
```

`post_upgrade.cfm` contains N named migration blocks. Each block:

1. Queries `migrations` table: has this block run before?
2. If yes: log "skipped (already applied)" and move on.
3. If no: do the work, then `INSERT INTO migrations (name) VALUES ('<block-name>')`.

This hook is for migrations that:

- Don't bind cleanly to a release boundary
- Need to be retroactive across all existing installs regardless of when they upgrade
- Replace existing inline-in-CFML migrations (lazy backfills currently scattered across page handlers)

Examples (some hypothetical):

Block name	What it does
`encrypt-relay-credentials-v1`	Re-encrypts `parameters` rows whose values were stored as plaintext before the key-rotation policy landed
`move-pgp-keys-to-vault-dir`	One-time file move from `/opt/hermes/keys/` to `/opt/hermes/keys/pgp/` after a refactor
`backfill-mailbox-domain-id`	Populates a `domain_id` column added in a prior release where the existing data needs computed values

The admin update procedure

For the period BEFORE `system_update_docker.sh` ships (#221), updates are manual. After #221 lands, the same procedure collapses into one command.

Today (pre-#221)

```
cd /opt/hermes-seg-docker-gl
git fetch --tags
git checkout <new-tag>                                  # or: git reset --hard origin/main
docker compose pull
docker compose up -d
./scripts/install_hermes_docker.sh --apply-schema       # SQL deltas
# Manual steps below ONLY if release notes call them out:
docker exec hermes_commandbox occ upgrade               # if NCVERSION bumped
# Service-specific restarts per release notes
```

`install_hermes_docker.sh --apply-schema` globs `updates/v*/sql/schema_updates.sql` and applies any with version newer than current `build_no`. On a v260119-only install, it logs "No per-release schema_updates.sql files found" and exits cleanly.

After #221

```
cd /opt/hermes-seg-docker-gl
./scripts/system_update_docker.sh           # latest release
./scripts/system_update_docker.sh v260601   # specific tag
```

The orchestrator runs all 5 phases. Idempotent + re-runnable. Logs every step. Refuses to run if working tree is dirty.

Pre-upgrade: take a hypervisor snapshot

Before running `system_update_docker.sh`, take a hypervisor / VM snapshot of the entire Hermes host (Proxmox, VMware, KVM, AWS EBS, Azure Disk, GCE, Hyper-V). If the upgrade fails partway through, reverting the snapshot is the only currently-supported rollback path — Docker-aware backup/restore tooling is in development at #219 and #220 and is not yet shipped. See `docs/admin/01-system/backup-restore.md` for the full interim-backup guidance.

Post-upgrade: hard-refresh the browser (any release that bumps NCVERSION or admin/2 assets)

After the orchestrator finishes, **operators must hard-refresh their browser** (Ctrl-Shift-R on Linux/Windows, Cmd-Shift-R on macOS) on any open Hermes admin tab or Nextcloud tab. The

browser cache often serves the pre-upgrade CSS/JS bundle even though the server is now serving the new one, which presents as broken layouts — most visibly NC's top navbar collapsing to a vertical stack of icons when the NC 30 → 31 bundle swaps under it.

Release notes that bump `NCVERSION` or land changes under `config/hermes/var/www/html/admin/2/` should call this out explicitly. One sentence in the release notes prevents the "Nextcloud top bar broke" support ticket.

The release-cut procedure (developer side)

For maintainers preparing a release:

1. **Code complete on `main`**: all PRs for the release have landed on GitLab `main`.
2. **Create the release directory**: `mkdir -p updates/v<DATE>/{sql,cfml,scripts}` (only the subdirs you actually need). Add `updates/v<DATE>/README.md` documenting what changed.
3. **Add SQL deltas**: write `updates/v<DATE>/sql/schema_updates.sql`. ALSO add equivalent rows/columns to `config/database/hermes_install.sql` so fresh installs after this release ship with them baked in. **Last block of `schema_updates.sql`** must be:

   ```
   UPDATE system_settings SET value='v<DATE>' WHERE parameter='build_no';
   UPDATE system_settings SET value='Docker'  WHERE parameter='version_no';
   ```

4. **Add CFML / bash migrations** as needed under `updates/v<DATE>/cfml/` and `updates/v<DATE>/scripts/`.
5. **Update `.env.template`**: bump `HERMES_DOCKER_IMG_VERSION=v<DATE>` and (if NC bumped) `NCVERSION=...`. `NCVERSION` is release-managed per #261 — operators never edit it; bumps land here only after the integration check in step 6 passes.
6. **(If `NCVERSION` was bumped in step 5) Run the NC integration check on a Test box**:

   ```
   # On Test, after a fresh clone or pull that includes the new .env.template
   docker compose pull hermes_nextcloud
   docker compose up -d hermes_nextcloud
   sleep 30                                    # let NC finish first-start init
   ./scripts/test_nc_integration.sh
   ```

   The script (read-only `occ` queries + log scan) verifies the Hermes-NC integration surface: container responsive, `occ status` reports the expected version, no `needsDbUpgrade`, required apps enabled and not flagged incompatible (`user_oidc` / `mail` / `twofactor_totp` / `twofactor_backupcodes` / `external`), `trusted_domains` populated, theming URL set, `user_oidc` provider `Hermes_SEG` registered, and no ERROR/FATAL entries in the last 200

`nextcloud.log` lines. Exit code 0 if no FAIL.
If anything fails, fix the integration (or revert the `NCVERSION` bump and pin to the prior NC) before continuing. Do not publish a release that ships a failing NC integration.

7. **Draft the GitHub Release body for `v<DATE>`**: list every change in the release. Per-release notes live on the GitHub Release page (created when the tag is pushed) — the cumulative `RELEASE-NOTES.md` was retired because release notes belong to a specific tag, not an ever-growing file.
8. **Build + push images**: `./Docker/build-all-ghcr.sh && ./Docker/push-all-ghcr.sh` (or wait for GitHub Actions per #218 Session C).
9. **Tag + push**: `./scripts/git_release.sh --release v<DATE>` (pushes branch + tag to both GitLab and GitHub).
10. **Verify**: GitHub Release page exists, ghcr.io packages updated, run `./scripts/system_update_docker.sh v<DATE>` on Test box and confirm clean upgrade.

Common scenarios

"I'm adding a new feature that needs a new table."

1. Add `CREATE TABLE IF NOT EXISTS ...` to `config/database/hermes_install.sql` (baseline gets it for fresh installs).
2. Add the same `CREATE TABLE IF NOT EXISTS ...` to the in-flight release's `updates/v<DATE>/sql/schema_updates.sql` (so existing installs get it on upgrade).
3. Add seed rows the table needs in BOTH files following the same pattern.

"I'm adding a new system_settings row with a default value."

Same as above: baseline + per-release. Use auto-assigned IDs (omit the `id` column); UNIQUE KEY on `parameter` handles dedup. See [[feedback-no-hardcoded-row-ids-on-new-seeds]] in memory.

"I'm changing a default value an existing install might have customized."

`UPDATE ... WHERE value = '<old-default>'` only in `updates/v<DATE>/sql/schema_updates.sql`. The `WHERE` clause preserves admin customizations. Also update `hermes_install.sql` baseline to the new

default for fresh installs.

"I rotated an encryption key and need to re-encrypt rows."

Per-release `updates/v<DATE>/cfml/reencrypt-foo.cfm`. Pure SQL can't reach the encryption key. Inside the script, check whether each row is already at the new format before re-encrypting.

"I need to move a config file from `/old/path` to `/new/path` on the host."

Per-release `updates/v<DATE>/scripts/move-config.sh`. Precondition check: if `/new/path` exists, exit 0.

"I just realized all installs need a one-time backfill but I'm not cutting a release for it."

Add a named block to `schedule/post_upgrade.cfm`. It runs at the next upgrade (or at every upgrade, gated by `migrations` table).

"Fresh install at v260119, what runs?"

Just `config/database/hermes_install.sql`. The baseline is self-contained. `updates/v260119/` has no `sql/` subdirectory by design. `--apply-schema` finds nothing to apply and reports current `build_no=v260119`.

"Upgrading from v260119 to v260601 (hypothetical), what runs?"

In order:

1. `git checkout v260601`
2. `docker compose pull && docker compose up -d`

3. `updates/v260601/sql/schema_updates.sql` (advances `build_no` to `v260601` at its end)
4. `updates/v260601/cfml/*.cfm` (each)
5. `updates/v260601/scripts/*.sh` (each)
6. Standard finalize (restarts, occ upgrade, template regen)
7. `schedule/post_upgrade.cfm` (any unapplied blocks)

"Upgrading from v260119 to v260801, skipping v260601 (hypothetical)."

The orchestrator applies BOTH release directories in order:

1. Phase 1-2 once
2. Phase 3 walks `v260601/` first (sql → cfml → scripts), then `v260801/`
3. Phase 4-5 once at end

`build_no` advances after each release's `sql/` completes — `v260601` then `v260801`. Each release's artifacts are idempotent so the in-between cursor advancement is safe.

State of the methodology as of v260119

Piece	Status
Baseline (`hermes_install.sql`)	Self-contained at v260119 (audit completed 2026-05-26)
Per-release directory (`updates/v260119/`)	Empty by design — v260119 IS the baseline; first real per-release directory is the NEXT release
`install_hermes_docker.sh --init-db`	Imports baseline only; no schema_updates.sql call
`install_hermes_docker.sh --apply-schema`	Globs `updates/v*/sql/schema_updates.sql`, applies in version order; no-op on v260119-only
`scripts/system_update_docker.sh` (#221)	**MVP shipped 2026-05-26** — phases 1-5 functional; some v2 polish deferred (see "Known MVP limitations" below)
`schedule/post_upgrade.cfm` framework	Stub shipped 2026-05-26 — framework + helpers + `migrations` table in place, zero blocks registered (first one lands when first migration is needed)
`migrations` table	Added to baseline 2026-05-26
GitHub Actions release workflow	NOT YET BUILT — image pushes are manual via `Docker/push-all-ghcr.sh`

Piece	Status
Image registry (ghcr.io)	Empty as of 2026-05-25; bootstrap is a pre-Session B task

Known MVP limitations of `system_update_docker.sh`

The MVP that shipped 2026-05-26 is deliberately scoped. These limitations are tracked for v2:

Limitation	What happens today	v2 plan
Service-restart detection	Always restarts `hermes_commandbox` in Phase 4; logs a reminder that admin may need to re-save certain config pages	Diff config files between previous and new tag; only restart containers whose volume-mounted config changed
`*.HERMES` template re-render	Phase 4 logs a reminder; admin must re-save the corresponding admin page manually	Detect modified template files and invoke each one's regen endpoint via curl into commandbox
`occ upgrade`	**Phase 4 auto-detects NCVERSION drift, runs `occ upgrade`, rehydrates Hermes-required NC apps, verifies post-upgrade state** (#264)	— already MVP-complete —
`updates/v<DATE>/cfml/*.cfm` artifacts	Phase 3 logs a WARN and skips them	First release that ships a CFML migration must add a host→container mount for `updates/` plus a Lucee mapping; the orchestrator already curls the right URL pattern, just needs the mount to exist
Mid-upgrade resume	`set -e` fail-fast; operator re-runs from scratch and idempotency handles re-application	Track per-phase + per-release-artifact completion; resume from last successful step
GitHub Releases API auth	Anonymous polling; subject to GitHub's unauthenticated rate limit (60 req/hr per IP)	Honor a `GITHUB_TOKEN` env var if present

---

Revision #13
Created 2026-05-31 13:01:44 UTC by Dino Edwards
Updated 2026-06-13 12:30:28 UTC by Dino Edwards