Backup/Restore

Backup/Restore

Admin path: **System > Backup/Restore** (`view_system_backup.cfm`).

> **CLI-only by design.** Backup and restore run from the Docker host's shell, not from the admin console. The admin console's *Backup/Restore* page is a read-only info surface (CLI examples + a list of backups detected on disk + a link back to this doc). There are no buttons. Long-running operations + web UIs is a known footgun (page reload kills progress, browser timeouts, race conditions); the CLI is the canonical interface.

What ships in this release

Two scripts under `scripts/`:

Script	Purpose
`system_backup.sh`	**Hot mode by default — zero application downtime.** Uses application-native hot-backup primitives: `mariadb-dump --single-transaction`, `slapcat`, and live tar of mail tiers (Dovecot, Amavis, Postfix all use atomic-rename writes safe for live tar). Toggles `occ maintenance:mode --on` briefly during Nextcloud file tar to pause NC user writes (mail flow unaffected). `--cold` flag stops the full stack for legal-hold / forensic snapshots that need absolute byte-level consistency.

Script	Purpose
`system_restore.sh`	**Always cold on the restore side** (we're overwriting tier contents — concurrent reads/writes would corrupt). Verifies the manifest + per-archive SHA256 BEFORE any destructive action, **auto-remaps** tiers to this host's paths (refuses only on a build-version mismatch unless `FORCE_VERSION_MISMATCH=1`), restores DBs via socket auth, restores OpenLDAP via `slapadd`, stream-extracts in-scope tiers directly to their mount paths, reconciles the Nextcloud DB user, restarts the stack, and on a cross-host restore offers to run `system_rehost.sh`.

Backup scopes

The `-B` flag chooses what to back up. Pick the scope that matches your need — there's no reason to back up 500 GB of vmail every night if only the DBs and configs are churning.

Scope	Includes	Typical cadence	Hot-mode duration
`system`	Config tier + Data tier + 6 DB dumps + LDAP slapcat	Nightly	seconds to a few minutes (dominated by `/mnt/data` tar size; DB+LDAP dumps are fast)
`archive`	Archive tier (Amavis quarantine)	Weekly or per retention policy	proportional to archive size; mail intake continues uninterrupted
`vmail`	Vmail tier (Dovecot mailboxes)	Weekly	proportional to mailbox size; mail flow continues uninterrupted
`nextcloud`	Nextcloud tier (NC files)	Weekly	proportional to NC file size; NC web UI shows "under maintenance" during the tar; mail unaffected
`all`	Everything above	Periodic full-DR snapshot	sum of all of the above

Hot-mode safety per component

Why we don't need downtime:

Component	Hot-backup technique	Why it's safe

MariaDB	`mariadb-dump --single-transaction --routines --triggers --events --databases <db>`	InnoDB MVCC gives a consistent point-in-time snapshot. No table locks. Stored procedures, triggers, and scheduled events captured.
OpenLDAP	`slapcat -b dc=hermes,dc=local` inside `hermes_ldap`	Standard hot LDIF export.
Dovecot (vmail)	tar `/mnt/vmail` live	maildir/sdbox writes are atomic-rename (write to temp filename, atomic `mv` to final name). No torn files. Worst case: messages arriving during the tar window may land after the tar's snapshot — they're durable upstream (postfix queue, sender's MX retries) and captured by the next backup.
Amavis (archive)	tar `/mnt/archive` live	Amavis quarantine writes are atomic-rename. Same as Dovecot.
Nextcloud (files)	tar `/mnt/files` live, with `occ maintenance:mode --on` toggled around the tar	NC writes are atomic, but the filesystem ↔ `oc_filecache` DB table can drift if a user uploads mid-tar. Maintenance mode pauses NC user writes — the NC web UI shows "under maintenance" briefly, but mail flow is unaffected. Use `--no-nc-maintenance` to skip the toggle if needed.
Postfix (data tier)	tar `/mnt/data/postfix` live	Postfix queue files are atomic-rename.
Service logs (data tier)	tar live	Append-only. A torn last line is cosmetic, not data loss.
MariaDB / LDAP / ClamAV raw files	**Excluded** from the data tier tar	DB dumps + LDAP slapcat are the authoritative restore sources, so the on-disk InnoDB tablespace files and slapd data files are redundant. ClamAV signatures are regenerable, not worth the backup space.

Hot mode is the daily backup. **Cold mode (`--cold`) is the escape hatch for use cases where absolute byte-level consistency matters more than uptime** — legal hold, forensic snapshots, regulatory archive. Cold mode does `docker compose stop` for the full duration.

Backup

Backup quick start

```
sudo /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B system --yes
```

The script creates a **backup directory** at `/mnt/backups/hermes-backup-<scope>-<build_no>-<UTC-timestamp>/` (e.g. `hermes-backup-all-v260609-20260609T183616Z/`). It is written under a `.staging-…` name and **atomic-renamed** into place only on success. **There is no outer tarball** — the per-tier archives sit directly in the directory, so the restore verifies and stream-extracts each one in place without unpacking a wrapper first (no ~2× scratch space). Read `manifest.json` directly to inspect a backup before restoring.

Output layout

Inside the backup directory (only the archives relevant to the chosen scope are present):

```
manifest.json                ← scope, mode (hot/cold), topology, source hostname,
                               build_no, SHA256 per archive
backup.log                   ← the backup run's own log
databases.tar.gz             ← 6 .sql files; system / all scopes only
ldap.ldif.gz                 ← slapcat output; system / all scopes only
config.tar.gz                ← Config tier USER-DATA subdirs only (keys, .gnupg,
                               ssl, templates, sa-bayes, sa-learn, dkim, arc,
                               conf_files) — NOT .env / secrets / compose / scripts
                               (those are host-specific and excluded by design);
                               system / all scopes only
data.tar.gz                  ← Data tier user-data only (excludes mysql/ ldap/
                               clamav/ — captured by dumps / slapcat / regenerable);
                               system / all scopes only
archive.tar.gz               ← Archive tier; archive / all scopes only
vmail.tar.gz                 ← Vmail tier; vmail / all scopes only
nextcloud.tar.gz             ← Nextcloud tier; nextcloud / all scopes only
```

Backup flags

Flag	Purpose
`-P <path>`	**Required.** Output directory. Must exist and be writable.
`-B <scope>`	**Required.** One of: `system`, `archive`, `vmail`, `nextcloud`, `all`.
`--cold`	Stop the full stack for the duration of the backup. Use for legal-hold / forensic snapshots. Default is HOT mode (zero application downtime).

Flag	Purpose
`--no-nc-maintenance`	Skip the brief `occ maintenance:mode --on` that hot-mode nextcloud / all backups use to pause NC user writes during the file tar. Without it, file uploads happening mid-tar may be missed by the backup.
`--yes` (or `-y`)	Skip the interactive confirmation prompt. Use for cron / Ofelia.
`--dry-run` (or `-n`)	Print what would happen without changing anything.
`--help` (or `-h`)	Show usage.

Scheduling

For nightly automated backups, use **host cron** on the Docker host. `system_backup.sh` is a host-level script (it runs `docker compose stop`, reads `.env` from the host, writes to `/mnt/backups` on the host) — host cron is the natural fit. Example `/etc/cron.d/hermes-backup`:

```
# m h dom mon dow user  command
0 3 * * *      root  /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B
system    --yes >> /var/log/hermes-backup.log 2>&1
0 4 * * 0      root  /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B
vmail     --yes >> /var/log/hermes-backup.log 2>&1
0 5 1 * *      root  /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B
all       --yes >> /var/log/hermes-backup.log 2>&1
```

A typical cadence:

Cadence	Scope	Why
Nightly	`system`	Small + fast. Captures DBs, LDAP, configs, install-root state. Run with hot mode = zero downtime.
Weekly	`vmail` (or `archive` or `nextcloud`, rotated)	Larger but slower-changing.
Monthly	`all`	Full disaster-recovery snapshot.

The script's exit code reflects success (0) or failure (non-zero). For built-in email alerting, use the `--notify-email=ADDR` flag (see below). For "Hermes is so dead it can't even tell you" cases, see External monitoring.

> **Why host cron and not Ofelia?** Ofelia runs as a container (`hermes_ofelia`). Its job model (`job-exec` into a named container, `job-local` on the Ofelia container itself) doesn't fit `system_backup.sh` cleanly — the script needs host-level `docker compose` access, root, and write access to `/mnt/backups`. Ofelia's image lacks `docker compose` plugin and root host access. Native Ofelia integration is deliberately NOT on the roadmap; the existing **System > Scheduled Tasks** admin page lists Ofelia jobs but does NOT support adding new ones from the UI today.

Failure / success email alerting

Use `--notify-email=ADDR` to receive an email on backup completion. By default emails on **failure only** (the "noisy on failure, silent on success" pattern most operators want). Add `--notify-on-success` to also email on success — useful for "daily I-am-alive confirmation" use cases.

```
# Email on failure only (typical)
sudo /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B system --yes \
  --notify-email=admin@example.com

# Email on both failure AND success
sudo /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B all --yes \
  --notify-email=admin@example.com --notify-on-success
```

Subject lines are bracketed for easy scanning in a mail client:

- Success: `[SUCCESS] Hermes backup on <hostname> (scope=<scope>)`
- Failure: `[FAILURE] Hermes backup on <hostname> (scope=<scope>)`

Failure bodies include the timestamp, scope, mode, reason, log file path, and the last 50 lines of the log. Success bodies include the timestamp, scope, mode, output filename, file size, and run duration.

How it works: the script shells out to `docker exec -i hermes_postfix_dkim sendmail -t` and pipes the message into the Postfix container's `sendmail` binary. Postfix queues and delivers it like any other outbound mail from Hermes. No host MTA configuration is needed — Hermes's own Postfix does the work.

Verify the path before wiring into cron — `--test-notify` sends one `[TEST] [SUCCESS]` sample and one `[TEST] [FAILURE]` sample to the address you give, then exits without running a backup:

```
sudo /opt/hermes-seg-docker-gl/scripts/system_backup.sh --test-notify \
  --notify-email=admin@example.com
```

Both test messages have a `[TEST]` prefix in the subject so any ops-alert filters watching for `[FAILURE]` are not tripped. If both arrive, your notification path is good. If neither arrives, check `hermes_postfix_dkim` is running and look at the log file the script prints for sendmail errors.

Caveat — needs Hermes to be at least partially healthy: if the failure cause is "the Postfix container is down" or "the Docker daemon is down", `docker exec` has nothing to talk to and the email won't go out. The script logs the failure-to-notify as a warning and exits with the original non-zero status, but you won't get the email. This is the gap external monitoring fills — see below.

External monitoring (strongly recommended)

Built-in email alerting covers the "backup ran but something went wrong" case (the 99% case). It does NOT cover "Hermes itself is so broken it can't send any email at all" — Docker daemon crashed, host out of disk, container restart loop, network partition, etc. For that, you need an external monitoring tool that lives off the Hermes host and tells YOU when Hermes goes dark.

Strongly recommended for every production install. Common choices:

Tool	Pattern	Best for
Zabbix	Agent on the Hermes host reports up/down, disk, container health, custom metrics	Self-hosted, comprehensive; common in business / mid-size deployments
Nagios / Icinga	NRPE plugin or similar	Self-hosted, classic; many existing operator setups already have it
healthchecks.io	Cron pings a URL on success; if the ping doesn't arrive on schedule, healthchecks alerts you	Dead simple; free tier; cron-native pattern
Uptime Kuma	Self-hosted ping monitor with web UI	Free, self-hosted alternative to healthchecks.io
PRTG / Datadog / New Relic / etc.	Commercial monitoring	If you already have one, integrate Hermes alongside your other infrastructure

The healthchecks.io pattern works nicely alongside cron-based backups:

```
# Pings healthchecks.io on success only (curl wraps the backup; ping is the URL of your check)
0 3 * * *  root  /opt/.../system_backup.sh -P /mnt/backups -B system --yes \
                 --notify-email=admin@example.com \
                 && curl -fsS --retry 3 https://hc-ping.com/<your-uuid> >/dev/null
```

If the backup fails, the `--notify-email` sends the failure email (assuming Postfix is up). If the backup succeeds, healthchecks.io gets the ping. If the WHOLE HOST is down (no ping, no email), healthchecks.io alerts you after the scheduled interval. Three-layer coverage with minimal moving parts.

Off-site copy

`system_backup.sh` writes to the local `-P` path only. Off-site copy is left to your existing tooling — `rclone`, `rsync` to remote storage, `aws s3 cp`, `restic`, whatever you already use. Typical pattern:

```
sudo /opt/hermes-seg-docker-gl/scripts/system_backup.sh -P /mnt/backups -B system --yes \
  && rclone sync /mnt/backups remote:hermes-backups/
```

Restore

Restore quick start

```
sudo /opt/hermes-seg-docker-gl/scripts/system_restore.sh -F /mnt/backups/hermes-backup-system-v260609-20260601T103000Z
```

`-F` takes the backup **directory** (not a tarball).

The restore replaces the data in the backup's scope and leaves other scopes alone. Restoring a `system` backup overwrites the install root + Data tier + DBs + LDAP; the Vmail / Archive / Nextcloud tiers are untouched. Restoring a `vmail` backup overwrites only `/mnt/vmail`. The stack is stopped for the duration of the restore (always — even hot-mode backups are restored cold).

Safety: SHA256 + version gates, topology auto-remap

Two gates fire BEFORE any destructive action, plus automatic topology handling:

1. **Manifest SHA256 verification.** Every archive's SHA256 is checked against `manifest.json` (verified in place — no unpacking). If any byte of the backup is corrupt or tampered with, the restore aborts BEFORE stopping the stack or touching any data.

2. **Hermes build-version match.** The backup's `build_no` (captured at backup time from `system_settings.build_no`) is compared against the current host's `build_no`. If they differ, restore refuses unless `FORCE_VERSION_MISMATCH=1` is set. Schema migrations between Hermes builds make cross-version restore unsafe — restoring an older DB dump onto a newer host leaves the schema in a state the running code does not expect, which breaks silently when something hits a missing or renamed column. **The correct procedure is to install Hermes at the matching build first (`git checkout <build>`), restore, then upgrade forward via `scripts/system_update_docker.sh`.**
3. **Storage-topology auto-remap.** If the backup's recorded mount paths (`/mnt/data`, `/mnt/vmail`, etc.) differ from this host's current mount paths in `.env` — typical when restoring onto different hardware — the restore **automatically retargets each tier to this host's paths** and prints a `REMAP` line per tier. No flag is needed; the old `FORCE_REMAP=1` gate was retired as needless friction for new-hardware DR.

Disaster-recovery flow (different host)

1. Install Hermes fresh on the new host using `install_hermes_docker.sh`. The install root + `.env` need to exist before restore can succeed.
2. Make the backup directory reachable on the new host — **either** mount the backup storage (off-site / NAS share) on the new host (recommended: the restore stream-extracts in place, so there's no need to copy the whole backup), **or** `scp -r` the backup directory across to local disk.
3. Run `system_restore.sh -F /path/to/hermes-backup-<scope>-<build>-<ts>`. Storage-topology differences are **auto-remapped** to this host's paths; a build-version difference still requires `FORCE_VERSION_MISMATCH=1` (better: install the matching build first).
4. When the restore detects a **cross-host** restore (backup hostname ≠ this host), it **offers to run `system_rehost.sh` for you** — accept it to rewire host identity (`.env`, DB rows, all rendered configs, and the Nextcloud DB user).
5. Verify the admin console loads and a test message flows end-to-end.

> **A cross-host restore needs more than the restore itself.** The restored data carries the *source* host's identity and credentials, so several things must be reconciled by hand — run `system_rehost.sh`, re-activate the Pro license, and re-save the Content Checks pages to re-apply the milter chain. Follow the full checklist: **Post-Restore Steps**.

Restore flags

Flag	Purpose

`-F <path>`	**Required.** Path to the backup **directory** produced by `system_backup.sh`.
`--yes` (or `-y`)	Skip the interactive confirmation prompt (and auto-accept the rehost offer on a cross-host restore).
`--dry-run` (or `-n`)	Show what would happen without changing anything.
`--only=<scope>`	Restore only one scope out of an `all` backup (e.g. `--only=vmail`).
`--help` (or `-h`)	Show usage.
`FORCE_VERSION_MISMATCH=1` (env)	Override the build-version refusal. Topology differences auto-remap — no flag needed.

When to use hypervisor snapshots instead

The cold-mode escape hatch (`--cold`) covers byte-level-consistency use cases that the cold-mode scripts can satisfy. For two other cases, **hypervisor snapshots** are the right tool, not the Hermes scripts:

1. **Pre-upgrade safety net.** Always take a hypervisor snapshot before running `system_update_docker.sh` — that gives you a working rollback if the upgrade fails mid-flight. The methodology doc codifies this.
2. **Zero-downtime full-host snapshot.** If you want a single consistent point-in-time image of the entire Hermes host (every storage tier, the Docker daemon state, the host OS), a hypervisor snapshot is the only tool that captures all of that atomically.

Per-hypervisor snapshot mechanisms:

Platform	Mechanism
Proxmox VE	Datacenter > Backup, or Snapshot from the VM's right-click menu
VMware vSphere / ESXi	VM > Snapshots > Take Snapshot
KVM / libvirt	`virsh snapshot-create-as <domain> <name> --disk-only --atomic`
AWS EC2	EBS volume snapshot (or AMI for full image)
Azure VMs	Disk snapshot, or Recovery Services Vault
Google Compute Engine	Disk snapshot
Hyper-V	Checkpoint

What you should NOT do

Do NOT run the legacy bare-metal scripts on a Docker host

The pre-Docker `config/hermes/opt/hermes/scripts/system_backup.sh` and `system_restore.sh` are kept in the repo for reference and for the legacy-to-Docker migration path. **Do not run them on a Docker install.** The legacy `system_restore.sh` does `cd / && tar -xvzf <backup-file>` — extracts the backup tarball relative to the host filesystem root and will overwrite host directories with files from a layout that does not match the Docker host's reality. Hermes services fail to start, host OS may become unbootable.

Do NOT tar a running storage tier with `tar` directly

If for some reason you reach for `tar` directly instead of `system_backup.sh`, do NOT tar `/mnt/data`, `/mnt/vmail`, `/mnt/files`, or `/mnt/archive` while the stack is running **without using the hot-backup primitives the script uses**. Specifically:

- `/mnt/data` contains MariaDB's tablespace files — tar'ing them while `hermes_db_server` is running produces a backup MariaDB will reject as inconsistent on restore. Use `system_backup.sh` (which excludes `mysql/` from the data tar and captures DBs via `mariadb-dump`) instead.
- Without `slapcat`, raw tar of `/mnt/data/ldap` mid-write captures inconsistent slapd database files.

The Hermes scripts handle all of this correctly. Use them.

Do NOT trust an untested restore procedure

Whatever backup strategy you adopt, **practice the restore at least once on a non-production system before you rely on it.** Take a backup of your live Hermes host, spin up a second VM, run the restore, verify you can log into the admin console and send a test message. A backup procedure that has never been restored from is not a backup procedure — it is wishful thinking.

What's coming in Phase B

The Phase A scripts cover the common cases (hot daily system backup, scoped tier backups, cold-mode forensic snapshot, scope-aware restore). The Phase B refactor (post-Link-Guard) will add:

- **Retention pruning** (`--retain-last=N` deletes older backups beyond N)
- **Per-tier** `--remap-tiers <old>:<new>` to override individual tiers (today's default is whole-backup auto-remap to this host's paths)
- **Selective container restart** instead of full `compose down` on the restore side (faster restart, smaller blast radius)
- **Filesystem-snapshot integration** (LVM / ZFS / btrfs detection): if a tier lives on a snapshot-capable filesystem, take a filesystem snapshot and tar the snapshot rather than the live mount, for use cases where "best-effort hot tar" isn't good enough but `--cold` is too disruptive

Not on the Phase B roadmap (deliberately dropped):

- **Native Ofelia integration**. Cron is the right tool. Ofelia's job model (`job-exec` into a named container, `job-local` on the Ofelia container) doesn't fit a host-level script cleanly. Forcing it would mean a custom Ofelia image with `docker compose` plugin + Docker socket + root access, plus admin-page UI work to add jobs — all to honor a pattern that doesn't fit. Host cron is the answer.
- **Admin-UI launch button**. Long-running operations + web UIs is a footgun; the admin who runs a backup is already in SSH. The Backup/Restore admin page stays read-only / informational, by design.

Failure / success notification is a separate discussion — see the Scheduling section above. Today the answer is cron's `MAILTO=` / pipe exit code into existing alerting; if operators ask for native built-in notification, it's a small Phase B addition.

Tracking: #219 for the backup-side enhancements, #220 for the restore-side.

Migrating from a legacy bare-metal install

A separate tool exists at `scripts/migrate_legacy_to_docker.sh` for operators moving from a legacy bare-metal install to the Docker install. It consumes a backup produced by the **legacy** `system_backup.sh` (which is correct in the bare-metal context where it ran) and restores it into the Docker layout via a translation step — NOT the same as running the legacy restore script directly. See the migration section of the

v260119 release notes for current scope.

Cross-references

- Storage Topology — the five-tier layout the backup operates on
- Release & Update Methodology — recommends taking a hypervisor snapshot before running `system_update_docker.sh`
- scripts/migrate_legacy_to_docker.sh — separate from backup/restore; for one-time bare-metal-to-Docker migration only

---

Revision #48
Created 2026-05-31 12:51:56 UTC by Dino Edwards
Updated 2026-06-20 13:32:56 UTC by Dino Edwards