

Backup Process

1. Overview & Architecture

This server (`ul`) implements an automated, multi-tiered backup system designed around high reliability, cryptographic security, redundancy, and verified recoverability. The backup infrastructure covers 43 containerized and native host services, maintaining local PAR2-protected rollbacks and GPG-encrypted multi-cloud offsite copies.

Key Philosophy

- **Self-Contained Service Scripts:** Each service has dedicated backup and restore scripts encapsulating its exact data dependencies (databases, ACLs, Quadlet definitions, and persistent volumes).
- **Parity & Error Correction:** Local backups utilize PAR2 (Parity Volume Set 2) with 10% redundancy to protect against bit-rot and silent disk corruption.
- **Zero-Trust Offsite Storage:** Cloud backups are client-side encrypted using GPG (AES-256 / SHA-512) before leaving local disk, preventing cloud storage providers from inspecting sensitive server state or database contents.
- **Strict Verification Before Retention Pruning:** Old backups (local and remote) are pruned **only** after a new backup run completes and passes 100% integrity verification checks (`rcclone check` / `par2 verify`).

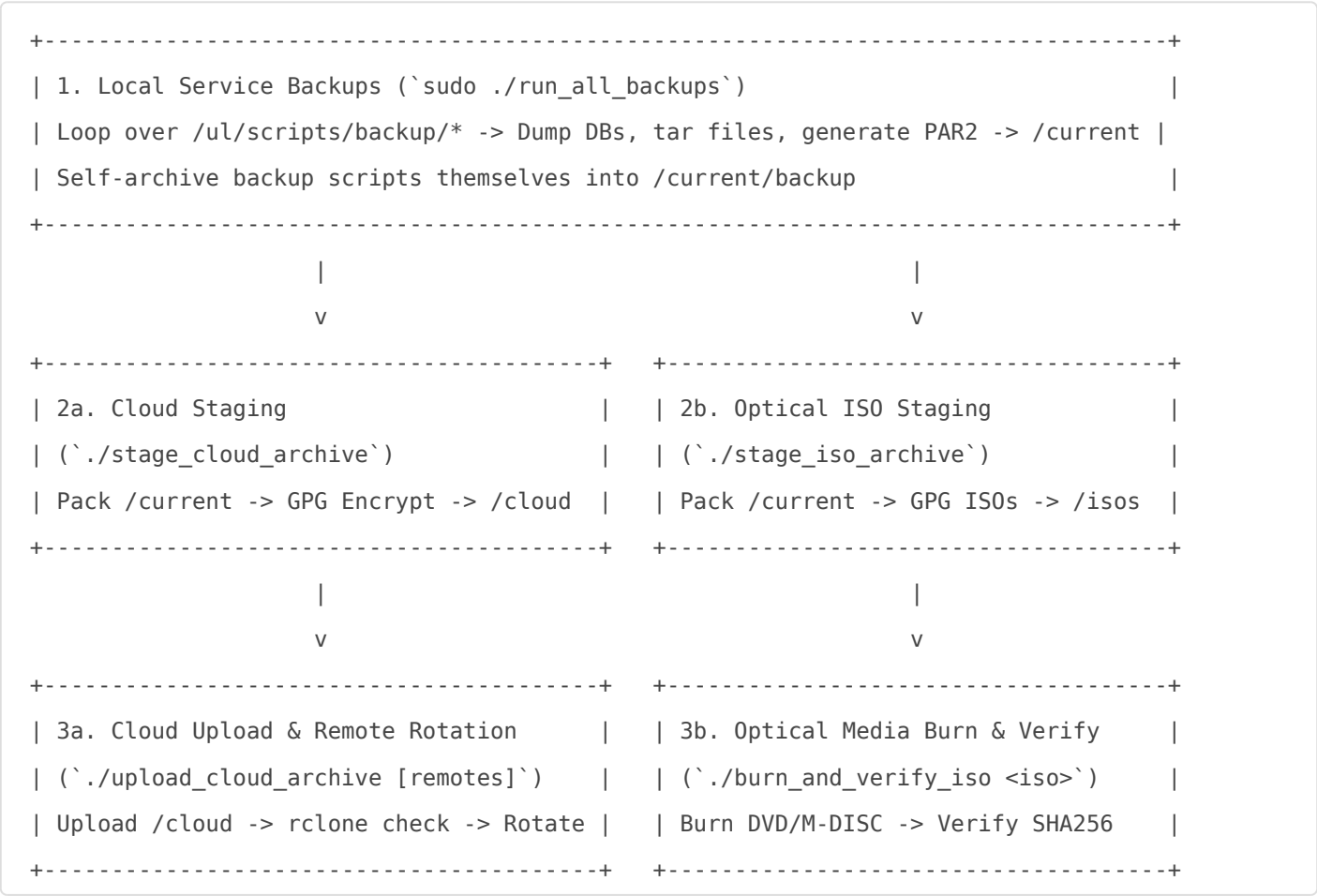
2. Directory & Data Structure

Path	Purpose	Access Control
<code>/home/luke/git/backups/ul/scripts/</code>	Shell and Python backup/restore execution scripts (GPG-signed git repository)	Executable
<code>/data/backups/services/current/<service>/</code>	Most recent local backup run for each service	<code>700 luke:luke</code>
<code>/data/backups/services/archive/<service>/</code>	Previous local backup run(s) for point-in-time recovery	<code>700 root:root</code>
<code>/data/backups/cloud/</code>	Staged GPG-encrypted net archive ready for cloud upload	<code>700 luke:luke</code>

Path	Purpose	Access Control
/data/backups/isos/	Staged optical media ISO images (when generating disk media)	700 luke:luke

3. End-to-End Backup Workflow

Local service archive creation, cloud staging, cloud upload, and optical ISO creation are separated into distinct commands under `/home/luke/git/backups/ul/scripts/`:



Commands & Flags

- `sudo ./run_all_backups`: Creates local archives for all services in `/data/backups/services/current/<service>/` and archives the backup scripts themselves into `/data/backups/services/current/backup/` (default: local archives only, no cloud staging or upload).
- `sudo ./run_all_backups --upload [remotes]`: Runs all local service backups and then executes `stage_cloud_archive` and `upload_cloud_archive [remotes]`.

- `./stage_cloud_archive [--upload [remotes]]`: Stages the GPG-encrypted cloud archive in `/data/backups/cloud/` (and optionally uploads it immediately if `--upload` is passed).
- `./upload_cloud_archive [remotes]`: Uploads the staged `/data/backups/cloud/` directory to all configured cloud remotes, or to specific comma-separated remotes (e.g., `./upload_cloud_archive gdrive,protondrive`).
- `./stage_iso_archive`: Packs and GPG-encrypts `/data/backups/services/current/` into 4.7GB optical DVD ISO images in `/data/backups/isos/`.

4. Service-Level Backups (`/ul/scripts/backup/`)

Every service backup script performs atomic snapshots using the following standard pattern:

1. **Root Verification & Timestamping:** Requires root permissions (`EUID == 0`) to preserve subordinate container UID/GID ownership (`1100`, `624288--689823`), POSIX ACLs (`--acls`), and extended attributes (`--xattrs`). Timestamp format: `YYYY-MM-DD_HHMM.SS`.
2. **Local Archive Rotation:** Moves existing `/current/<service>/*` directories to `/archive/<service>/`.
3. **Consistent Database & Cache Snapshots:**
 - **Native Host PostgreSQL** (`authentik`, `anchor`, `forgejo`, `immich`, `nextcloud_db`, `vikunja`, `wipstudio`): Staged to `/tmp/<fn>.dump` first via `sudo -u postgres pg_dump -F c -b -v` before moving into `/data/backups/services/current/<service>/`.
 - **Containerized PostgreSQL** (`penpot-postgres`, `funkwhale`): Executed rootlessly via `(cd /tmp && sudo -u docker-data podman exec penpot-postgres pg_dump -U penpot -F c -b penpot)` and `podman exec -u funkwhale funkwhale pg_dumpall -U funkwhale`.
 - **Containerized MariaDB** (`bookstack_db`, `romm-db`, `yourls-db`): `(cd /tmp && sudo -u docker-data podman exec <db_container> mariadb-dump --single-transaction --routines --triggers ...)`.
 - **SQLite WAL Checkpoints** (`navidrome`, `subocr`, `vaultwarden`): Consistent online `.backup` or `PRAGMA wal_checkpoint(TRUNCATE)`.
 - **Native Valkey** (`valkey.service`): Triggers `BGSAVE` with dedicated read/save-only ACL user `backup` and archives `/var/lib/valkey/dump.rdb` + `/etc/valkey/`.
4. **Configuration, Quadlet & Volume Compression:**
 - Compressed using multi-threaded XZ (`XZ_OPT="-T0" tar --acls --xattrs -cJf ...`).
 - Preserves `.env`, `<service>.container`, `<service>.network`, application data/assets, and host configs (`systemd` archives `/etc/systemd`, `/etc/containers`, `/etc/haproxy`, `/etc/nftables.conf`, and `/home/docker-data/.config/containers/systemd`).
5. **PAR2 Recovery Generation:**
 - Generates 10% parity redundancy (`par2 create -r10`) directly alongside every database dump and tarball.

6. **Execution Logging:** Captures output to `backup_execution.log` inside each timestamped service backup folder.

Backed-Up Services (43 Targets)

`acme`, `amux`, `anchor`, `audiomass`, `authentik`, `bentopdf`, `bookstack`, `cam`, `craft`, `crowdsec`, `degoog`, `demucs`, `drawpile`, `excalidraw`, `forgejo`, `funkwhale`, `grill`, `immich`, `jellyfin`, `mc_manager`, `minecraft_router`, `navidrome`, `nextcloud`, `nginx`, `penpot`, `rclone`, `romm`, `scontrol`, `sopro`, `subocr`, `sunshine`, `systemd`, `tldraw`, `transmission`, `valkey`, `vaultwarden`, `vikunja`, `wavacity`, `webrcade`, `webshell`, `wick`, `wipstudio`, `wireguard`, `yourls`.

5. Cloud Archiving & Encryption (`archive_pack.py`)

Local service backups are consolidated into a secure net package by `archive_pack.py` invoked via `stage_cloud_archive`:

- **Input Directory:** `/data/backups/services/current`
 - **Output Directory:** `/data/backups/cloud/`
 - **GPG Recipient:** Key fingerprint `9F9AC4567EDC8E2B86E5CB9475D6600BEF4E8E8F`
 - **Cryptographic Specifications:** AES-256 cipher, SHA-512 digest.
 - **Manifests & Hashes:** Generates `manifest.json`, `manifest.txt`, `SHA256SUMS`, and `DECRYPT_INSTRUCTIONS.txt` mapping original file sizes, staged sizes, SHA-256 hashes, and GPG metadata.
 - **Optical Media Support:** `archive_pack.py` also supports `--mode disk` (for CD/DVD/Blu-ray ISO image staging with optional splitting and PAR2 protection via `stage_iso_archive`).
-

6. Cloud Orchestration & Provider Synchronization

Cloud synchronization is orchestrated by `upload_cloud_archive` calling provider scripts in `upload/`:

Configured Providers

1. **Google Drive** (`upload/gdrive`)
2. **Proton Drive** (`upload/protondrive`)

Upload Reliability Features

- **Exponential Backoff & Retries:** Up to 3 main attempt loops with exponential delay scaling (10s/15s initial delays). Low-level HTTP retries configured up to 20 attempts.
- **Self-Healing Draft Cleanup:** If `rclone copy` encounters mid-transfer errors or failed file copies, the upload script parses `rclone` logs and explicitly deletes failed/corrupted draft files from the remote before retrying.
- **Strict Integrity Verification:** Post-upload, `rclone check --one-way --size-only` verifies that remote file sizes match local files exactly.
- **Atomic Failure Safety Guard:** If upload or verification fails across all attempts, the script deletes the partial remote upload directory (`rclone purge`) and **skips retention rotation** to preserve existing valid cloud backups.

Remote Retention Policy

- **Max Remote Backups:** Retains the newest **3** successful cloud backups (`MAX_BACKUPS=3`).
 - **Automated Proton Trash Cleaning:** Executes custom Go binary (`proton-trash-cleaner`) to automatically purge deleted backup files from Proton Drive's cloud trash bin.
-

7. Restoration Practice & Recovery Procedure

Restoration scripts reside in `/home/luke/git/backups/ul/scripts/restore/`. Each service script handles full state recovery:

Standard Restore Procedure

1. **Target Selection:** Accepts an explicit backup directory or timestamp as argument. If omitted, defaults to the latest backup in `/data/backups/services/current/<service>/`.
2. **PAR2 Integrity Check & Self-Repair:** Runs `par2 verify` on database dumps and file tarballs prior to performing any system modifications. Automatically executes `par2 repair` if corruption is detected.
3. **Interactive Confirmation:** Prompts user for explicit confirmation (`yes/no`) before stopping services or modifying state.

4. **Service Suspension:** Gracefully stops active rootless Podman Quadlet units (`sudo -u docker-data XDG_RUNTIME_DIR=/run/user/1100 systemctl --user stop <service>.service`) or native systemd units.
 5. **Rollback Snapshot Creation:** Preserves current configurations in a timestamped rollback folder (`<service_dir>_rollback_<timestamp>`).
 6. **File & Database Restoration:**
 - Extracts compressed tarball over service data directory with POSIX ACL/xattr and subordinate UID/GID preservation.
 - Re-creates target database schema and imports the SQL dump or PostgreSQL custom binary dump.
 7. **Service Resumption:** Reloads and restarts rootless Quadlets (`sudo -u docker-data XDG_RUNTIME_DIR=/run/user/1100 systemctl --user daemon-reload && sudo -u docker-data XDG_RUNTIME_DIR=/run/user/1100 systemctl --user start <service>.service`).
-

8. Retention Pruning Practice (`prune_old_local_backups`)

To prevent local storage exhaustion on the server, local service backup folders are managed by `/home/luke/git/backups/ul/scripts/prune_old_local_backups`:

- **Local Retention Limit:** Retains the **2** most recent backups per service (`KEEP_COUNT=2`) across `/data/backups/services/current` and `/data/backups/services/archive`.
 - **Safe Dry-Run Flag:** Supports `--dry-run` (`-n`) to inspect directories eligible for pruning without performing deletions.
-

Revision #6

Created 2026-08-16 16:08:16 UTC by admin

Updated 2026-10-20 03:45:58 UTC by luKe