Raspberry Pi Lab Architecture

Architecture case study, ADRs and playbook

Hamish Burke
hamish@burke.kiwi | Github | Linkedin | Blog/Portfolio

3 September 2025

Last updated: 3rd September 2025

TL;DR

Single-host production lab on a Raspberry Pi 4 (8 GB) serving a dashboard, file manager, photo service, and Minecraft server.

Key metrics

Metric	Value
Host	Raspberry Pi 4 Model B, 8 GB RAM
Users	2 primary human users; occasional Minecraft spikes.
Public access	Cloudflare Tunnel (TLS) + Caddy (HTTP).
Admin access	Tailscale (mesh VPN).
Backup remote	Backblaze B2 via ‘rclone crypt‘.
RTO (target)	2 hours (critical web/file services).
RPO (target)	6 hours (acceptable data loss for user files).
Current backup data (measured)	96 GB (as of 15 August 2025).
Projected data footprint	$\approx$ 1 TB (projected).
Last restore test	15 August 2025.
Monthly backup cost (measured)	\$2.82/month for 96 GB (August 2025).

What I owned / outcomes

I design and operate the single-host environment. Responsibilities include system design, orchestration and deployment, backup architecture, access control configuration, and the recovery playbook. A full restore completed in 1 hour 45 minutes on 15 August 2025. Monthly Backblaze B2 cost measured at \$2.82 for 96 GB in August 2025 (varies with API transactions and object count). Immich ML tasks peak at 75% CPU utilisation with 1.5 GB RAM usage during photo processing. Configuration files and directory structure are available in this project’s repo.

Contents

1 Executive summary

Overview. A single-node Raspberry Pi 4 (8 GB) runs the Homer dashboard, FileGator, Immich, and a Minecraft server, with Portainer, Radicale, and SMB. Public access is via Cloudflare Tunnel fronted by Caddy; administrative access is via Tailscale. The design favors clarity, security, and rapid recovery over high availability.

Architecture. Caddy handles host-based routing, TLS headers, and logging; Cloudflare Tunnel terminates TLS at the edge and connects to Caddy, which proxies HTTP to containers. Public services (Homer dashboard, Immich, Radicale, FileGator) are behind the Tunnel; admin services (Portainer, SMB, SSH, PostgreSQL, Minecraft) use Tailscale. Secrets live in a gitignored '.env' with high-sensitivity values in Docker secrets; SSO is not used and may be added in future if additional services require SSO.

Operations and security. StatusCake probes check defined URLs and status codes; weekly backup reports go to email. Images are pinned to semantic versions and staged via a separate Compose project/profile before production. Caddy enforces HSTS, X-Content-Type-Options, X-Frame-Options, Referrer-Policy, and optional CSP. FileGator Basic Auth is bcrypt-hashed at rest and protected by TLS in transit.

Backups and DR. 'rclone crypt' to Backblaze B2 with 4-hour PostgreSQL dumps, nightly 'rclone sync' to a "current" path, and weekly dated snapshots. ' /.config/rclone/rclone.conf' keys are escrowed offline with checksums and a documented recovery process. Full restore tested 15 Aug 2025 in 1h45m (meets RTO); schedule meets 6-hour RPO. Cost: $2.82/month for 96 GB (August 2025).

Performance and capacity. Container memory caps total about 7.26 GB with approximately 0.74 GB headroom. Immich ML tasks are scheduled for low-use hours and can be offloaded via 'MACHINE_LEARNING_URL' if needed. External service spend capped at $15/month.

Notable trade-offs. Single-node design mitigated by encrypted offsite backups and a tested DR playbook; dual networking (Cloudflare Tunnel + Tailscale) is documented with explicit service mapping; exFAT limitations are handled by mapping to '/srv/will_mapped' to simulate ownership semantics.

2 Constraints and measurable goals

Explicit constraints make trade-offs measurable. The following list defines the working constraints for the environment.

2.1 Assumptions and non-goals

- Single node; no clustering or horizontal scaling.
- No high-availability (HA) targets.
- No 24/7 paging or on-call rotation.

2.2 Hardware

- Single Raspberry Pi 4 Model B, 8 GB RAM, chosen for low power consumption (5 W typical) and broad ARM64 container image support
- Boot from 32 GB SanDisk microSD; persistent data and configuration on Samsung EVO 256 GB external SSD via USB 3.0 to avoid SD card wear on write-heavy workloads
- 1 TB external HDD (exFAT-formatted; must remain as-is for owner requirements)

2.3 Users and load

- Two human users: owner and flatmate; Minecraft server for occasional spikes
- Regular web traffic expected to be light: dashboard, photo service, and file manager

2.4 Network

- Consumer ISP behind CGNAT with no static IPv4
- Reliable remote access requires a tunnel or VPN solution

2.5 Storage and backups

- Estimated data footprint 1 TB; encrypted off-site backups are mandatory
- RTO: 2 hours for critical web and file services
- RPO: at most 6 hours for user files

2.6 Security and compliance

- External access must be encrypted and access-controlled
- Secrets must not be committed to public repositories

2.7 Operational

- Maintenance windows acceptable during weekday evenings (8–10 pm local)
- External service costs should not exceed $15/month
- Basic monitoring for service health; no 24/7 alerting required; further observability may be explored later

3 High level solution

This section summarises the chosen stack, the reasons for those choices, and the intended operational flow

3.1 Chosen stack and roles

- Host OS: Raspberry Pi OS (Debian derivative) with controlled update process
- Orchestration: Docker Compose for simplicity and low resource overhead
- Reverse proxy: Caddy for local TLS, host-based routing, and security headers
- External access: Cloudflare Tunnel with hostname-based routing to Caddy
- Administrative access: Tailscale for secure mesh networking and low-latency connections
- Storage: External SSD mounted at '/srv/hamish' and '/srv/will' for user data, 'Docker volumes' for application state
- SMB access: SMB shares over Tailscale for file access, using the same directories as FileGator (for example, '/srv/hamish/files/' and '/srv/will/files/', or mapped equivalents like '/srv/will_mapped')

Core services: Immich (photos with ML), FileGator (file management), Homer dashboard, Minecraft server, Portainer (container management), Radicale (CalDAV), SMB (file sharing over Tailscale)

3.2 External access architecture

Cloudflare Tunnel provides secure external access without port forwarding

- Tunnel ID: '66f90ca2-d062-4ecd-8290-e6688beeb85e'
- Public hostnames (placeholders), standardised mapping:
  - 'dash.hamishburke.dev:443' → Caddy ':8080' (Homer dashboard)
  - 'immich.hamishburke.dev:443' → Caddy ':8081' (Immich photos)
  - 'calendar.hamishburke.dev:443' → Caddy ':8083' (Radicale CalDAV)
  - 'files.hamishburke.dev:443' → Caddy ':8084' (FileGator)

Caddy provides security headers (HSTS, XSS protection, content-type options) and request logging.

3.3 Resource allocation

- Memory limits (targeted caps):
  - Immich ML: 2 GB
  - Immich server: 768 MB
  - PostgreSQL: 1 GB
  - Minecraft: 2 GB
  - FileGator: 512 MB
  - Homer dashboard: 128 MB
  - Caddy: 128 MB
  - Cloudflared: 128 MB
  - Radicale: 128 MB
  - Samba (SMB): 256 MB
  - Portainer: 256 MB
- Aggregate requested memory: $\approx$ 7.26 GB out of 8 GB RAM
- Headroom: $\approx$ 0.74 GB (9.25%) reserved for the host OS, kernel page cache, and burst
- CPU limits: Best-effort scheduling with Minecraft given higher shares during gameplay

3.4 Authentication strategy

- Public services: Individual service authentication (Immich accounts, FileGator basic auth)
- Administrative and private access: Tailscale network-level access control

- No unified SSO to maintain service independence. May be added in future if additional services require SSO

3.5 System Architecture

See Figure 1 for a container-level architecture diagram of the environment.

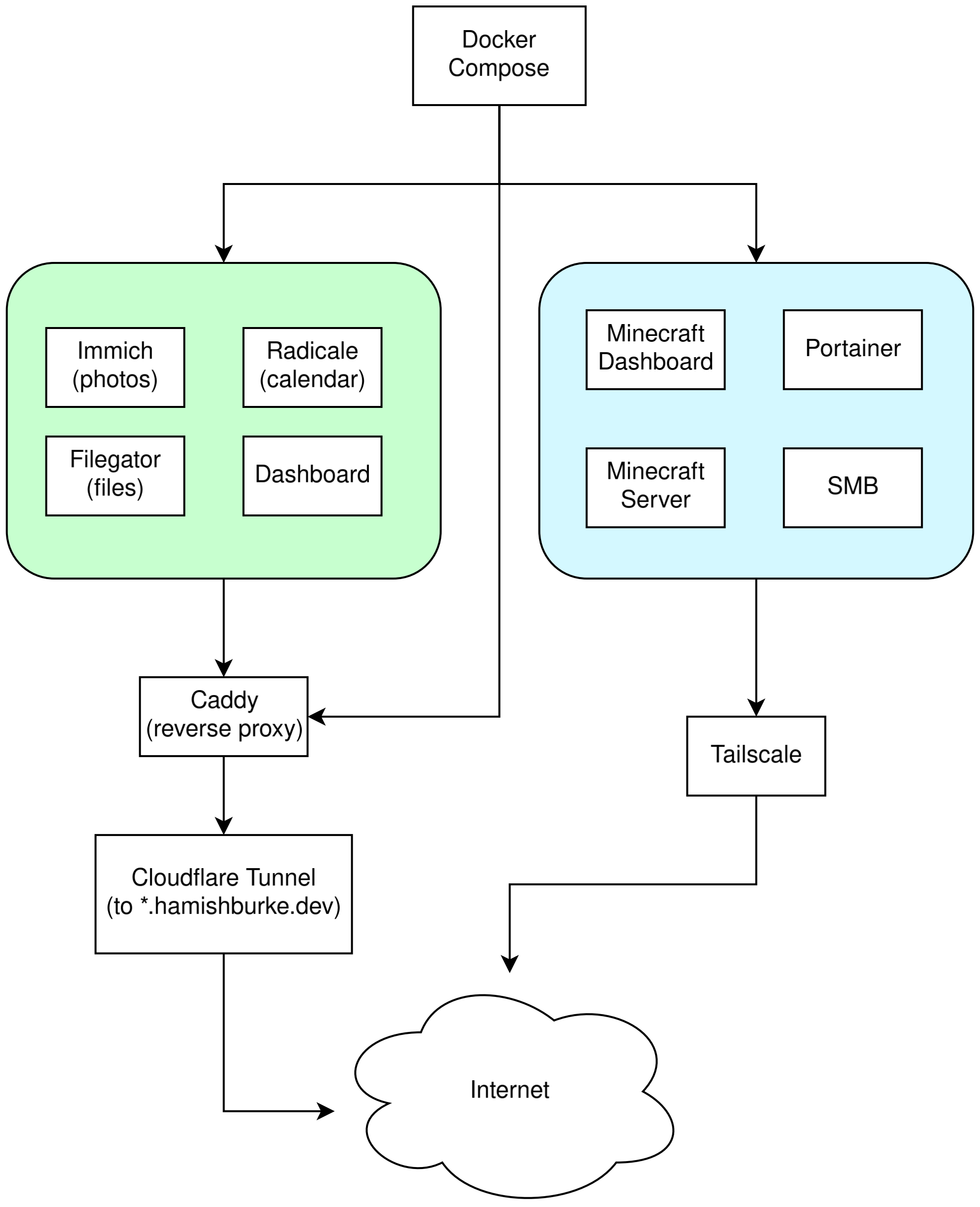

Figure 1: System architecture — container view of major services grouped by connection method.

4 Security Architecture

4.1 Access Control Policy

- **Public web services** exposed via Cloudflare Tunnel and fronted by Caddy for TLS, host-based routing, and integration with Cloudflare Access
- **Administrative interfaces** and maintenance access use Tailscale for low-latency peer-to-peer connectivity; SMB file shares are restricted to Tailscale-authenticated devices, using the same directories as FileGator for seamless access
- **SSH access** is key-based only with password logins disabled

4.2 Secrets and configuration management

- Secrets inventory (placeholders shown, no real values)
  - Immich: ‘IMMICH_VERSION‘, ‘IMMICH_UPLOAD_LOCATION‘, ‘IMMICH_ML_CACHE‘
  - Database: ‘DB_USERNAME‘, ‘DB_PASSWORD‘, ‘DB_DATABASE_NAME‘
  - Caddy: ‘CADDY_EMAIL‘, HTTP basic auth credentials hashed in ‘Caddyfile‘
  - Minecraft: ‘EULA‘, ‘MC_MEMORY‘, ‘MC_VERSION‘, and related variables
  - Cloudflare Tunnel credentials stored at ‘/etc/cloudflared/<tunnel-id>.json‘
- ‘.env‘ file used for non-secret configuration and environment variables; ensure ‘.env‘ is ignored via ‘.gitignore‘
- Prefer Docker secrets for high-sensitivity values (‘DB_PASSWORD‘, basic auth hashes) and mount as files or environment at runtime
- Repository contains sanitised configuration templates only; no real credentials are committed
- File permissions: ‘0600‘ for credential files and ‘0644‘ for configuration files

4.3 Security Headers and TLS

- TLS path: Client HTTPS → Cloudflare edge (TLS termination) → Cloudflare Tunnel (‘cloudflared‘, mTLS/QUIC) → Caddy (local reverse proxy) → backend containers
- Intra-container hops use HTTP on the Docker network by default; optional local TLS between Caddy and services can be enabled if required
- Caddy security headers applied to public origins
  - Strict-Transport-Security (HSTS): ‘max-age=31536000; includeSubDomains; preload‘
  - X-Content-Type-Options: ‘nosniff‘
  - X-Frame-Options: ‘DENY‘ (or ‘SAMEORIGIN‘ where embedding is required)
  - Referrer-Policy: ‘no-referrer‘ (or ‘strict-origin-when-cross-origin‘ for analytics compatibility)
  - Content-Security-Policy (if applied): locked-down ‘default-src ’self’‘ with explicit script/-style allowances for each UI (for example, Homer dashboard, FileGator)
- Health checks and status endpoints on each service enable Caddy to make informed routing decisions

4.4 Risks and mitigations

- Single-node, no HA introduces a single point of failure → mitigate with regular tested restores, documented recovery steps, and conservative change control
- Exposure via public endpoints (Tunnel or Caddy misconfiguration) → mitigate with Cloudflare Access, least-privilege API tokens, header hardening, and periodic config reviews
- Secret leakage through repository or logs → mitigate with ‘.gitignore‘ for ‘.env‘, Docker secrets for sensitive values, restricted file permissions, and log scrubbing
- Intra-host HTTP between reverse proxy and services → mitigate with Docker network isolation; enable local TLS or mTLS for sensitive services if threat model requires

- Brute-force and abuse against public services → mitigate with Cloudflare rate limiting/WAF and Caddy request limits; enforce strong passwords and 2FA where available

5 Backup and Disaster Recovery

A clear backup policy is essential. This section lists what to back up, the cadence, encryption and storage details, and the precise restore steps.

5.1 What is backed up

- **Immich PostgreSQL database** (with ‘pgvector‘): Full dumps, suitable for point-in-time restore
- **User files**: ‘/srv/hamish/files/‘ and ‘/srv/will/files/‘
- **Immich upload libraries**: Included within the user file directories above
- **Radicale CalDAV data**: Calendar and contacts volumes
- **Minecraft world data**: World saves and configuration
- **Docker volumes**: ‘portainer_data‘ and ‘filegator_data‘ for application state
- **Configuration files**: ‘compose.yaml‘, ‘Caddyfile‘, Cloudflare Tunnel config at ‘/etc/cloudflared/<tunnel-id>.json‘, and ‘.env‘ (sanitised template only in repo)
- **Excluded**: Immich ML cache (can be rebuilt)

5.2 Backup Strategy

- **Tooling**: ‘pg_dump‘/‘pg_restore‘ for PostgreSQL; ‘rclone sync‘/‘rclone copy‘ to Backblaze B2 (crypt remote)
- **Directory layout in B2**:
  - ‘b2:pi-backups/current/‘ — current state mirrored nightly via ‘rclone sync‘
  - ‘b2:pi-backups/snapshots/YYYY-MM-DD/‘ — dated weekly snapshots via ‘rclone copy‘ from ‘current/‘
  - Example substructure: ‘current/db/‘, ‘current/files/hamish/‘, ‘current/files/will/‘, ‘current/volumes/filegator_data/‘, ‘current/config/‘
- **Cadence**:
  - Database dumps every 4 hours: ‘pg_dump -Fc‘ produced under a local staging path (for example, ‘/srv/backups/db/‘) then synced
  - Nightly file sync at 03:30: ‘rclone sync‘ user files, Docker volumes, and configuration into ‘current/‘
  - Weekly snapshots (for example, Sunday 04:00): ‘rclone copy b2:pi-backups/current b2:pi-backups/snapshots/YYYY-MM-DD‘ to create immutable restore points
  - Backblaze B2 lifecycle rules: keep all versions for 30 days; retain weekly ‘snapshots/‘ for 12 weeks
- **Example cron** (times local):
  - ‘0 */4 * * *‘ — ‘pg_dump‘ to ‘/srv/backups/db/‘ with timestamped filename, then ‘rclone sync /srv/backups/db b2:pi-backups/current/db‘
  - ‘30 3 * * *‘ — ‘rclone sync /srv/hamish/files b2:pi-backups/current/files/hamish‘ and ‘rclone sync /srv/will/files b2:pi-backups/current/files/will‘
  - ‘30 3 * * *‘ — ‘rclone sync /var/lib/docker/volumes/filegator_data/_data b2:pi-backups/current/volume and similarly for ‘portainer_data‘
  - ‘0 4 * * 0‘ — ‘rclone copy b2:pi-backups/current b2:pi-backups/snapshots/$(date +%F)‘

5.3 Encryption and Remote Storage

- **Encryption**: ‘rclone crypt‘ defaults for data and names (NaCl Secretbox — XSalsa20-Poly1305; salted filename encryption)

- **Remote**: Backblaze B2 bucket with versioning enabled; directory structure as above
- **Key material location**: ‘ /.config/rclone/rclone.conf‘
- **Escrow and recovery for ‘rclone.conf‘**:
  - Create two offline copies of ‘ /.config/rclone/rclone.conf‘ (encrypted removable media) and store separately
  - Record ‘rclone config file‘ output location, a SHA-256 checksum of the file, and the crypt remote name in the recovery runbook
  - Store the crypt passphrase and salt in a password manager with a break-glass procedure for the second admin; print a sealed copy in physical escrow if required
  - Recovery procedure: provision new host → install ‘rclone‘ → place ‘rclone.conf‘ in ‘ /.config/rclone/‘ → verify checksum → ‘rclone ls crypt:pi-backups‘ to validate access
- **Cost**: $2.82/month for 96 GB (measured August 2025; varies with object count and API operations)

5.4 Disaster Recovery Playbook

Restore order: volumes and files → database → services

1. Prepare host
   - Boot target machine or attach storage; install Docker and the Docker Compose plugin: ‘sudo apt-get update && sudo apt-get install -y docker.io docker-compose-plugin‘
   - Clone the repository and place production ‘.env‘ alongside ‘compose.yaml‘
2. Recreate named volumes and restore data
   - Create volumes: ‘docker volume create filegator_data && docker volume create portainer_data‘
   - Restore user files:
     - ‘rclone sync crypt:pi-backups/current/files/hamish /srv/hamish/files/‘
     - ‘rclone sync crypt:pi-backups/current/files/will /srv/will/files/‘
   - Restore volume contents using a helper container:
     - ‘rclone sync crypt:pi-backups/current/volumes/filegator_data /tmp/restore/filegator_data‘
     - ‘docker run –rm -v filegator_data:/data -v /tmp/restore/filegator_data:/restore alpine sh -c "cp -a /restore/. /data/"‘
     - Repeat for ‘portainer_data‘ if needed
3. Start only the database to restore it
   - ‘docker compose up -d postgres‘
   - Fetch the desired dump: ‘rclone copy crypt:pi-backups/snapshots/<DATESTAMP>/db/immich-<DATESTAMP>.dump /tmp/‘
   - Ensure ‘pgvector‘ is available: ‘docker compose exec -T postgres psql -U "$DB_{U}SERNAME$”$-d$”DB_DATABASE_NAME" -c "CREATE EXTENSION IF NOT EXISTS pgvector;"‘
   - Restore database (custom format): ‘docker compose exec -T postgres bash -lc ’PGPASSWORD="$DB_PASSWORD" pg_restore –clean –if-exists -U "$DB_USERNAME" -d "$DB_DATABASE_NAME" /tmp/immich-<DATESTAMP>.dump’‘
   - If using plain SQL instead: ‘docker compose exec -T postgres bash -lc ’PGPASSWORD="$DB_PASSWORD" psql -U "$DB_USERNAME" -d "$DB_DATABASE_NAME" -f /tmp/immich-<DATESTAMP>.sql’‘
4. Start remaining services
   - ‘docker compose up -d‘
5. Validate
   - Check service health UIs and logs
   - Spot-check Immich libraries, FileGator access, Radicale sync, and Minecraft world load

Tested restore procedure: Completed in 1 hour 45 minutes on 15 August 2025 (minor DNS propagation delay 15 minutes)

5.5 Risks and mitigations

- Accidental deletion via ‘rclone sync‘ — enable B2 file versioning, use weekly ‘snapshots/‘, and require peer review for destructive changes
- Loss of ‘rclone.conf‘ keys — maintain dual offline escrow copies with checksum verification and a documented recovery drill
- Inconsistent backups during writes — use database dumps for PostgreSQL, schedule file syncs during low activity, and consider application quiescence hooks for large restores
- Cloud or network outage during restore — keep a local copy of the most recent snapshot for critical data when feasible and test restores quarterly
- Cost growth with data expansion — monitor Backblaze B2 usage monthly; prune obsolete snapshots per policy and adjust lifecycle rules

6 Trade-offs and Mitigations

6.1 Minecraft Server Resource Limits

- **Trade-off**: Default JVM settings can consume all available RAM.
- **Mitigations**: Pin resource usage with environment variables ‘MEMORY=$MC_MEMORY‘, ‘USE_AIKAR_FLAGS=true‘ for optimised garbage collection, and ‘MAX_TICK_TIME=-1‘ to prevent server shutdown during temporary CPU spikes.

6.2 Immich ML Performance Impact

- **Trade-off**: Face detection and ML processing can saturate CPU and consume significant RAM; PostgreSQL with ‘pgvector‘ adds memory overhead.
- **Mitigations**: Run ML in a separate container with a cache volume, use health checks with 30-second intervals, and schedule ML tasks at low-use hours. Optionally point ‘MACHINE_LEARNING_URL‘ to an external service if local performance is insufficient.

6.3 FileGator Basic Auth Security

- **Trade-off**: HTTP Basic Auth is simple but transmits credentials on each request and is not inherently replay-resistant.
- **Mitigations**: Credentials are bcrypt-hashed at rest in ‘Caddyfile‘ and are protected in transit by TLS (Cloudflare Tunnel → Caddy). Enforce HSTS, disable plaintext HTTP, and enable Cloudflare rate limiting/WAF to reduce brute-force and replay risk. Consider placing Cloudflare Access in front (signed tokens) or upgrading to OAuth/OIDC for stronger, replay-resistant authentication in the future.

6.4 Tunnel Plus Mesh Complexity

- **Trade-off**: Running both Cloudflare Tunnel and Tailscale increases flexibility but adds debugging complexity.
- **Mitigations**: Document service paths explicitly to reduce ambiguity:
  - Cloudflare Tunnel (public HTTP): Homer dashboard, Immich, Radicale, FileGator (all via Caddy).
  - Tailscale (administrative/P2P): Portainer, SMB, SSH, PostgreSQL, and Minecraft server.

6.5 Docker Compose Simplicity

- **Trade-off**: Docker Compose is lightweight but lacks cluster features.
- **Mitigations**: Retain Compose for single-host operations. If scaling is planned, maintain a tested migration path to k3s (for example: convert ‘compose.yaml‘ to Helm/‘kompose‘ manifests

→ stand up k3s on a new node → deploy Caddy/Cloudflare Tunnel ingress equivalents → migrate volumes using ‘rclone‘/‘restic‘ → cut over DNS).

6.6 exFAT HDD Permission Issues

- **Trade-off**: exFAT (required for the flatmate's HDD) lacks native Linux permission support, causing ownership issues for FileGator and SMB.
- **Mitigations**: Mapped the directory to ‘/srv/will_mapped‘ to simulate ownership semantics for exFAT, allowing consistent access for FileGator and SMB without reformatting.

6.7 Risks and mitigations

- Resource contention on an 8 GB host during concurrent ML and Minecraft loads → enforce container memory caps, schedule ML at low-use hours, and prioritise Minecraft CPU shares during gameplay.
- Authentication weaknesses of Basic Auth → keep bcrypt-hashed credentials, enforce TLS/HSTS, add Cloudflare WAF/rate limits, and plan migration to OAuth/OIDC or Cloudflare Access.
- Dual networking planes (Tunnel + Tailscale) causing routing confusion → keep an explicit service mapping inventory and standardise troubleshooting runbooks.
- exFAT filesystem edge cases (ownership, metadata) → continue using mapped directory semantics; document limitations and avoid POSIX-permission-dependent workflows on that path.
- Future scale constraints with Compose → retain a documented, tested k3s migration path and rehearse data migration for critical volumes.

7 Operational Practices

7.1 Monitoring and Alerting

- Caddy exposes a ‘/health‘ endpoint aggregating service health checks.
- External uptime monitoring via StatusCake (5-minute intervals) with explicit checks:
  - ‘https://dash.hamishburke.dev/health‘ — expect HTTP ‘200‘; alert routing: email to admins.
  - ‘https://immich.hamishburke.dev/api/server-info‘ — expect HTTP ‘200‘; alert routing: email to admins.
  - ‘https://files.hamishburke.dev/‘ — expect HTTP ‘200‘ (HEAD/GET); alert routing: email to admins.
  - ‘https://calendar.hamishburke.dev/.well-known/caldav‘ — expect HTTP ‘401‘ (protected CalDAV endpoint); alert routing: none.
- Weekly automated backup status reports with success/failure notifications via email to admins.

7.2 Update Strategy

- OS updates: Monthly manual review during maintenance windows.
- Staging before production using a separate Compose project or profiles:
  - Separate project: ‘docker compose -f compose.yaml -p staging –env-file .env.staging up -d‘ for smoke tests on Tailscale-only ports.
  - Profiles: declare ‘profiles: ["staging"]‘ on services to be trialled, then ‘docker compose –profile staging up -d‘.
  - Promote by pulling and restarting production: ‘docker compose pull && docker compose up -d‘; roll back by reverting tags in ‘.env‘/‘compose.yaml‘ and re-deploying.
- Automatic security updates enabled for base OS packages only.

7.3 Capacity Management

- Disk usage alerts when SSD exceeds 80% capacity.
- Weekly capacity reports tracking photo library and world data growth (current growth: ≈ 1 GB/month).

8 Architecture decision records (summary)

A one-line summary table of ADRs for quick scanning; full ADRs are in the repo at ADRs.md.

ADR	Decision	Status	Rationale
001	External access via Cloudflare Tunnel and Caddy	Accepted	No port forwarding; simple TLS and routing.
002	Orchestration: Docker Compose rather than Kubernetes	Accepted	Lightweight single-host orchestration; minimal overhead.
003	Backups: rclone crypt to Backblaze B2	Accepted	Reliable offsite storage, encrypted at rest.
004	Host OS: Raspberry Pi OS with controlled update process	Accepted	Stable base; predictable kernel and drivers.
005	Administrative access: Tailscale for mesh VPN and admin operations	Accepted	Zero-config mesh; device ACLs; low latency.
006	Storage layout: external SSD at '/srv' and Docker volumes for state	Accepted	Separate system and data; reduce SD wear.
007	Database: local PostgreSQL container with persistent volume and scheduled dumps	Accepted	Simple local DB; predictable backups.
008	Authentication: service-level auth, no unified SSO (for now)	Accepted	Reduce complexity; small user base fit.
009	Secrets management: Docker secrets and encrypted files, avoid committing to repo	Accepted	Gitignored '.env'; Docker secrets; no repo creds.
010	Container management UI: Portainer restricted to admin network	Accepted	Admin-only via Tailscale; reduce exposure.
011	Resource allocation: memory and CPU limits per container	Accepted	Prevent contention on 8 GB host.
012	ML processing: run photo ML on device where feasible	Accepted	Adequate performance; schedule low-use hours.
013	Monitoring and logging: lightweight health checks, log rotation and offsite archive	Accepted	Catch outages; control disk growth.
014	Updates and change management: controlled upgrades, snapshots and rollback plan	Accepted	Reduce downtime risk; enable rollback.
015	Backup retention: tiered retention and lifecycle rules on B2	Accepted	Balance cost with restore objectives.
016	Privacy and data governance: minimise external processing and document data flows	Accepted	Respect user data; limit third-party exposure.

9 Conclusion and Future Directions

The setup has worked well. The most difficult part was handling different drive formats and working within that constraint (mapping the HDD) rather than reformatting. A key lesson learnt was deciding when a service should be accessible via Cloudflare Tunnel versus via the Tailscale mesh. For administrative services, where only a select number of devices should have access, I will continue to use Tailscale. For files, photos, and similar services, I value the flexibility of subdomain access from any device without extra setup. SMB over Tailscale complements this by enabling direct file sharing on the same directories as FileGator, improving usability for private access.

In the short term, improvements focus on observability and polish. While Portainer is useful for container health and logs, I plan to augment Portainer with Prometheus and Grafana for metrics and dashboards, and selectively add OpenTelemetry where supported to gain cross-service traces.

10 Reproducibility resources

- Repository root: github.com/Slaymish/RaspberryPiHomeLab.
- ADR directory/file: ADRs.md.
- Sample Caddyfile: Caddyfile.
- Sample docker-compose.yml: docker-compose.yml.