SOT-1d Plan-Review — Code/Data Volume Separation

Datum: 2026-05-14 16:03 UTC | Repo: `Steve-TradingBot` | master HEAD: `85c7dbf` | Bot-Image: `63b198ad61ef` | Status: **PLAN-PHASE** | Implementation: **NO-GO sofort**

Executive Summary

Befund: Das named Volume `steve-tradingbot_clawbot-openclaw` mountet auf `/home/node/.openclaw/` (Volume-Root) und überdeckt damit den gesamten Image-Codepfad `/home/node/.openclaw/workspace/trading/` (Image: **2.5 MB Code**, Live-Container: **197 MB** wegen Daten/Logs). Trennung: Image = Code (`*.py` , `requirements.txt` , Subdir-Module), Volume = Daten (`.env` , `logs/` , `__pycache__/`).

Root-Cause: Mount-Punkt liegt zu hoch in der Hierarchie. Volume sollte nur Daten-Subpfade halten, nicht den ganzen `workspace` .

Empfehlung: Option C-modifiziert (Daten-Subpfad-Volumes ohne Code-Touch). Alternative: Option B (Code-Pfad in `/opt/` , sauberere Trennung, aber +Code-Touch in 4 Files). **NO-GO sofort** — Plan-Phase nur, GO für SOT-1d-Umsetzung nach Operator-Entscheidung D1–D4.

Live-State: ✓ master `85c7dbf` , working tree clean. Bot PID 3331968 healthy (image `63b198ad61ef`), Worker 3335179, GUI 3335999, GUI HTTPS=200. BINANCE_TESTNET=true, cmd 13 cancelled, managed_proposals+history 0/0, decision_logs +144/5min, position_snapshots +12/5min, 0 Tracebacks. `position_snapshots status=closed total=0` (kein Close seit DATA-CLEANUP-1-Cutover, erwartet).

1. Live-State Preflight

Item	Wert
master HEAD	`85c7dbfddbbfc819b7b2644f058c172baaa07a83`
git status	working tree clean
Bot	host-PID 3331968, healthy, restartCount 0, image `63b198ad61ef`
Bot main.py	host-PID 3337375, scan-loop active
Worker	host-PID 3335179, running, restartCount 0
GUI	host-PID 3335999, running, image `259300694a28`
GUI HTTPS /admin/login	HTTP 200
BINANCE_TESTNET	`true`
cmd 13	cancelled
managed_proposals / history	0 / 0
decision_logs / 5 min	+144 (latest 15:53:48+)
position_snapshots / 5 min	+12 (latest 15:53:27)
position_snapshots status=closed total	0 (erwartet, DATA-CLEANUP-1 gerade aktiviert, kein Close-Event seitdem)
bot_statuses / 2 min	+4
Tracebacks post-Cutover	0

2. Aktuelle Mount-Architektur

Bot `clawbot`

Source	Destination	Mode	Code/Data
`steve-tradingbot_clawbot-openclaw`	**`/home/node/.openclaw`**	rw	**MIX (Problem)**
`steve-tradingbot_clawbot-workspace`	`/home/node/workspace`	rw	unused
`steve-tradingbot_clawbot-data`	`/home/node/.clawbot`	rw	legacy
`steve-tradingbot_clawbot-memu`	`/home/node/.openclaw/memUdata`	rw	legacy/empty
`steve-tradingbot_clawbot-workflows`	`/home/node/.openclaw/workflows`	rw	legacy/empty
bind `/projekte/Steve-TradingBot/config`	`/home/node/.openclaw/host-config`	ro	config (openclaw.json, searxng)
bind `/var/run/docker.sock`	`/var/run/docker.sock`	ro	legacy

Worker `clawbot-worker`

- `clawbot-openclaw` rw → `/home/node/.openclaw` (gleicher Mix!)
- `clawbot-workspace` rw → `/home/node/workspace`
- bind `config` ro → `/home/node/.openclaw/host-config`

GUI `steve-tradingbot-gui`

- `clawbot-openclaw` **ro** → `/bot-data/openclaw` (read-only Daten-Konsum)
- bind `/projekte/Steve-TradingBot/logs` ro → `/bot-data/logs`
- bind `/projekte/Steve-TradingBot/config` ro → `/bot-data/config`
- bind `/projekte/Steve-TradingBot/gui` → `/var/www/html`

Code-vs-Daten Grenze in `workspace/trading/`

Was	Wer schreibt	Wer liest	Im Image?
`*.py` (19 files), Subdirs `execution/news/scanner/tier3/strategies/state/config/`, `requirements.txt`, Doku- `.txt`	nie	Bot + Worker	**JA (Code, 2.5 MB)**
`.env`	Operator / Deploy	Bot (TELEGRAM_*, BINANCE_*, GUI_DB_*, BINANCE_TESTNET=true)	nein (gitignored, dockerignored)
`logs/` (192.9 MB: bot_stdout.log, scanner.log, trading.log, errors.log, bot_poller_state.json, live_portfolio.json, bot.pid, command_worker.pid, daily_performance.csv, *.bak, bugswarm/)	Bot + Worker	GUI ro, Operator	nein
`workspace/gui/command_bus_versions.json` (266 B)	Operator / Deploy	Worker	nein
`__pycache__/`	Python runtime	—	nein (regenerable)

3. Problem exakt

- **Warum überdeckt das Volume den Image-Code?** Docker mountet named volumes auf einen Zielpfad und macht damit alle Image-Inhalte unter diesem Pfad unsichtbar. Volume mountet auf `/home/node/.openclaw/` — somit ist der gesamte Subbaum incl. `workspace/trading/` aus dem Volume zu sehen, nicht aus dem Image. Beim ersten Mount eines leeren named Volumes kopiert Docker den Image-Inhalt einmalig ins Volume — danach hat das Volume Vorrang.
- **Was muss künftig aus Image kommen?** Alle `*.py`, Subdirs (execution/news/scanner/tier3/strategies/state/config/backtest/docs/reports/notifications/tests/scripts/), `requirements.txt`, Doku- `.txt`.
- **Was muss im Volume bleiben?** `logs/` (live data), `.env` (config/secrets), `workspace/gui/command_bus_versions.json` (Worker-Config).
- **Wer braucht Zugriff auf was?**
  - Bot: Code rw → ro; `.env` ro; `logs/` rw
  - Worker: Code ro; `command_bus_versions.json` ro; `logs/command_worker.pid` rw
  - GUI: `logs/` ro (über Daten-Volume); Code-Pfad braucht GUI nicht

4. Optionen-Vergleich

Option A — Volume auf Trading-Subpfaden

Pro	Con
Keine Pfad-Änderung im Code	Fragmentierte Mounts (`.env` als bind, `logs/` als named volume subpath); Operator-Edits am Host nur indirekt

Verdict: machbar, aber meist nicht der eleganteste Pfad.

Option B — Code wandert auf `/opt/steve-tradingbot/trading`

Pro	Con
Physisch komplett getrennt (Code-Pfad ≠ Daten-Pfad), kein Volume-Overlay-Risiko mehr	Code-Touch in **4 Files**: • `Dockerfile` : `COPY trading/ /opt/steve-tradingbot/trading/` • `entrypoint.sh` : `DASHBOARD=/opt/steve-tradingbot/trading/dashboard.py` • `bot_watchdog.sh` : `BOT_DIR=/opt/steve-tradingbot/trading` + Log-Pfad • `worker_watchdog.sh` : command_worker.pid-Pfad bleibt im Daten-Volume `main.py:95` `log_dir` bleibt auf `/home/node/.openclaw/workspace/trading/logs` (Daten-Pfad), kein Code-Edit nötig wenn CWD korrekt gesetzt

Verdict: architektonisch am saubersten, Code-Touch klein aber nicht null.

Option C — Separates `bot-data` Volume (Subpath-Mounts, Image-Pfad unverändert) — EMPFEHLUNG

Pro	Con
0× Code-Touch. Pfade unverändert. `.env` + `command_bus_versions.json` als Host-Bind-Mounts (operator-edierbar, im Host-Backup mit drin). Künftige Repo-Code-Änderung → `docker compose build && up -d clawbot` reicht.	3 Mounts statt 1. `.env` -Bind braucht 1× Operator-Aufgabe (cp aus Volume nach `/projekte/Steve-TradingBot/.env-runtime/bot.env`).

Mount-Plan (Compose-Diff Vorschau):

```
clawbot:
  volumes:
    # Daten-Subpfade (NEUES Volume)
    - clawbot-bot-data:/home/node/.openclaw/workspace/trading/logs
    - type: bind
      source: /projekte/Steve-TradingBot/.env-runtime/bot.env
      target: /home/node/.openclaw/workspace/trading/.env
      read_only: true
    - type: bind
      source: /projekte/Steve-TradingBot/gui/command_bus_versions.json
      target: /home/node/.openclaw/workspace/gui/command_bus_versions.json
      read_only: true
    # Nicht mehr: das alte clawbot-openclaw-Root-Volume
```

Subpath-Mount-Verifikation: Docker Engine 26.1.3 — named volume auf Subpfad (z. B. `clawbot-bot-data:/home/node/.openclaw/workspace/trading/logs`) wird unterstützt. Bei leerem Volume + nicht-leerem Image-Pfad copy-on-init; bei leerem Image-Subpath (durch `.dockerignore` `logs/` ausgeschlossen) startet das Volume leer und wird live aufgefüllt.

Verdict: minimale Invasivität, Code-Pfad unchanged, klar reproduzierbar.

5. Empfehlung: Option C

1. **0× Code-Touch:** kein Dockerfile / main.py / entrypoint / watchdog-Edit nötig.
2. **Reproducible Build "live"**: `docker compose build clawbot && up -d --no-deps clawbot` macht Repo-Edit sofort live (ohne Volume-Rebuild-Tanz wie heute zweimal nötig).
3. **Daten klar isoliert:** Logs/State im einen Volume, Secrets/Config als Bind-Mounts (Host-side sichtbar).
4. **Worker analog:** dieselben 3 Mounts auf Worker, command_worker.pid landet im neuen Daten-Volume.
5. **GUI:** mountet neues `clawbot-bot-data:/bot-data/openclaw-data:ro` für read-only Daten-Konsum.

6. Auswirkungen pro Subsystem (Option C)

Subsystem	Impact
Bot entrypoint.sh	unchanged
Bot main.py	unchanged (`log_dir` weiter `/home/node/.../logs/`)
Host bot_watchdog.sh	unchanged (`BOT_DIR` gleich)
Worker command_worker	unchanged (`COMMAND_BUS_VERSIONS_PATH` env unchanged)
Host worker_watchdog.sh	unchanged
GUI /bot-data/openclaw	wird `/bot-data/openclaw-data` ; 1 Pfad-Edit nötig falls GUI-Code hardcoded — TODO im Spike
Dashboard 8050	unchanged
Healthcheck	unchanged
Backup/Restore	einfacher: nur 1 kleines Daten-Volume tar (~200 MB), .env als Host-File bei Repo-Backup automatisch mit
Image-Rebuilds	**jetzt wirklich live** — Repo → Image → Container ohne Volume-Eingriff
DATA-CLEANUP-1	bereits live (Cutover heute), künftig keine Wiederholung nötig
Mainnet	unverändert (BINANCE_TESTNET=true)

7. Migrationsplan SOT-1d-impl (geschätzt 15–25 min Downtime)

1. **Preflight Snapshot** (Bot/Worker/GUI/master/Telemetry/cmd-13)
2. **Watchdog-Freeze** (crontab `# CUTOVER_FREEZE` Pattern wie DATA-CLEANUP-1-LIVE)
3. **Frisches Volume-Backup** `clawbot-openclaw → /root/sot-1d-backup/clawbot-openclaw-<ts>.tar.gz` mode 600
4. **Vorab-Bind-Quellen vorbereiten:**
   - `mkdir -p /projekte/Steve-TradingBot/.env-runtime/` mode 700
   - 1× `.env` aus Volume nach Host kopieren via `docker run --rm -v clawbot-openclaw:/src busybox cat /src/workspace/trading/.env > .env-runtime/bot.env` (mode 600)
   - `gui/command_bus_versions.json` ist bereits im Repo
5. **Docker-Compose-Edit** (clawbot + clawbot-worker Volume-Block) — Replace `clawbot-openclaw:/home/node/.openclaw` durch:
   - `clawbot-bot-data:/home/node/.openclaw/workspace/trading/logs`
   - bind `.env-runtime/bot.env → .../trading/.env:ro`
   - bind `gui/command_bus_versions.json → .../workspace/gui/command_bus_versions.json:ro`

   Volume-Block: `clawbot-bot-data` hinzufügen, `clawbot-openclaw` -Deklaration entfernen
6. **GUI docker-run-Pattern** aktualisieren (orphan run): `-v clawbot-bot-data:/bot-data/openclaw-data:ro`
7. **Coordinated Stop** Bot + Worker + GUI
8. **Volume rm** `clawbot-openclaw` (alte Generation entfernen)
9. **Compose recreate** Bot + Worker — Image `63b198ad61ef` initialisiert `workspace/trading/` frisch aus Image; neues `clawbot-bot-data` startet leer
10. **Restore aus Backup**: `logs/` -Inhalt aus altem Backup ins neue Daten-Volume (Subpath `./workspace/trading/logs/` → Volume-root)
11. **GUI recreate** via docker run (Pattern aus GUI-RESTORE-1, neuer Mount)
12. **main.py spawn** + Verifikation: 3-Way MD5 Repo==Image==Container (alle 4 Pflicht-Files); Tracebacks/OpenClaw/psycopg2 = 0; decision_logs/position_snapshots schreiben; Worker-Heartbeat frisch
13. **End-to-End Smoke-Beweis**: Repo-Edit (z. B. `trading/version_marker.txt`) → `docker compose build clawbot` → `up -d --no-deps clawbot` → File im Container sofort sichtbar **ohne** Volume-Tanz. Marker via `git restore` wieder entfernen
14. **Watchdog re-enable**
15. **Closure**

8. Rollback-Plan

- Watchdog gefreezed lassen
- Volume `clawbot-bot-data` rm
- Compose-Edit revert (git apply revert ODER vorbereitete `docker-compose.yml.pre-sot-1d`)
- Volume `clawbot-openclaw` aus Backup restore: `docker volume create` → tar xzf in `/var/lib/docker/volumes/clawbot-openclaw/_data` via busybox
- Compose `up -d`
- bot_watchdog spawn main.py
- DATA-CLEANUP-1 würde dann wieder **NICHT live** sein (pre-SOT-1d-Verhalten)

9. DATA-CLEANUP-1 Live-Pfad

> *DATA-CLEANUP-1 ist bereits live (commit `85c7dbf`, image `63b198ad61ef`, container md5 `79b0b5aadf...`). SOT-1d ist hier schadensfreie Architektur-Verbesserung — verhindert Wiederholung des Volume-Rebuild-Tanzes bei künftigen Repo-Änderungen.*
>
> *Nach SOT-1d-impl: Repo-Edit `trading/main.py` → `git commit` → `docker compose build clawbot && up -d --no-deps clawbot` → live. Punkt. Keine Volume-Operations mehr.*

10. Operator-Decisions

ID	Frage	Empfehlung
D1	Welche Option (A/B/C)?	**C** (0× Code-Touch, kleinster Cutover, klar reproduzierbar)
D2	`.env`-Quelle: Repo-Host-Bind (`/projekte/.../.env-runtime/bot.env`) ODER named volume `clawbot-bot-env`?	Host-Bind mit `.env-runtime/` mode 700 — Host-sichtbar, leichter zu rotieren
D3	Recreate-Pfad GUI: weiter orphan docker run ODER gleich mit GUI-COMPOSE-1 verbinden?	GUI-COMPOSE-1 als **separate Phase** danach; SOT-1d-impl macht GUI nur Mount-Update
D4	Subpath-Mount für `logs/` als named volume — Engine 26.1.3 unterstützt es; Sandbox-Test vorab?	Optional. 5-min `docker run busybox`-Test reicht; bei Sensibilität ja, sonst direkt impl

11. Stop-Regeln (alle eingehalten in Plan-Phase)

Regel	Status
Secret sichtbar	✓ keiner ausgegeben
unklarer Container-Zweck	✓ alle 3 identifiziert
Worker/GUI Datenbedarf unklar	✓ Worker → command_bus_versions.json + command_worker.pid; GUI → /bot-data/openclaw ro
DATA-CLEANUP-1 würde wieder nicht live	✓ Plan stellt Code-Pfad sicher
Mainnet nicht klar blockiert	✓ BINANCE_TESTNET=true durchgehend

12. Boundaries (alle eingehalten)

0× Code, 0× Container-Stopp, 0× Volume-Edit, 0× docker cp, 0× DB-Migration, 0× Push, 0× Mainnet, 0× Secrets im Output, 0× env-Dump, 0× compose config mit Secrets.

13. GO / NO-GO

Phase	Status
Plan-Review	**DONE**
SOT-1d-impl	**NO-GO sofort** — Operator-Entscheidungen D1–D4 nötig

STOP — Operator-GO für SOT-1d-impl mit Entscheidungen D1–D4 erforderlich.