RECON Managed Holdings — Sub-Roadmap v2

Version: v2 (2026-05-10) · **Status:** Architektur-Closure mit Q-MH-Decisions integriert · **13 Files konkateniert** (12 Roadmap + 1 Q-MH-Session-Audit-Trail)

Boundaries: keine Implementierung, kein Code, keine Migration, kein Bot-Restart, kein Worker, kein Mainnet, kein Push.

📄 `99_master_boundaries.md`

RECON-MH · Master Boundaries

Status: verbindlicher zentraler Vertrag für alle RECON-Managed-Holdings-Phasen **Version:** v2 (2026-05-10) **Geltungsbereich:** alle Files unter `docs/roadmap/recon_managed_holdings/` und alle daraus abgeleiteten Implementierungsphasen **Bezug:** ergänzt — überschreibt nicht — die globalen Regeln aus `memory/recon_status_pin.md` (DR-1..DR-13, SR-1..SR-10)

---

1 · Globale Durable Rules (für alle MH-Phasen verbindlich)

ID	Regel	Quelle
G-DR-1	Frozen-by-default: jedes Wallet-Asset ohne expliziten Eintrag ist `frozen` (Bot rührt nicht an)	RECON-2.1 + DR-2
G-DR-2	Operator ist final authority bei jeder State-Transition `frozen ↔ managed`	DR-11
G-DR-3	CommandBus-only: jede Mutation an `baseline_holdings.json` / `managed_state.json` / `risk_proposals/*.json` läuft AUSSCHLIESSLICH über CommandBus + Worker, nie aus PHP, nie via direct file-write	DR-1 erweitert
G-DR-4	TESTNET-first dauerhaft: alle MH-Phasen entwickeln und drillen ausschließlich auf testnet; Mainnet bleibt blockiert bis explizite RECON-2.5 Sign-Off	DR-3
G-DR-5	Audit-heavy: jede State-Transition + jeder Operator-Klick + jeder Bot-Drift-Detect schreibt mindestens 1 audit_event-Row + (ab SM-6) 1 immutables Snapshot-File	DR-6 + DR-13
G-DR-6	Backup-before-mutate: jeder Writer-Apply / -Clear / -State-Transition erstellt sha256-verifiziertes Backup vor Schreibvorgang	DR-5 + DR-9
G-DR-7	Operator-Multi-Step-Wizard für Promote: kein One-Click-Transfer; mindestens 5 Schritte inklusive Risk-Summary + Hard-Confirm	UX-2 (Reviewer)
G-DR-8	`'managed'` darf NIE `default_policy_for_unlisted` sein; nur explizite Asset-Markierung kann den State setzen	DR-2
G-DR-9	Bot darf NIE autonom `managed_*` → `frozen` zurückführen; einzig zulässige autonome Bot-Transitionen sind `managed_active → managed_drift_alert` und `managed_active → exit_executed`	DR-11
G-DR-10	Risk-Proposals MÜSSEN versioniert sein (`proposal_version` + `risk_model_version`); unversionierte Proposals werden vom System abgelehnt	DR-12
G-DR-11	Audit-Event-Prefixes strict separiert: `managed.*` ↔ `baseline.*` ↔ `runtime_config.*` ↔ `apply_profile_testnet.*` — keine Wiederverwendung	DR-13
G-DR-12	Wallet-Signature-Hash schließt `tradable_quote` -Slots aus (USDT-drift darf signature nicht brechen)	DR-7
G-DR-13	Watchdog-Skript MUSS Container-Namen `clawbot` verwenden (nie `steve-tradingbot`); vor jedem Restart manueller Dry-Run-Check	DR-8
G-DR-14	**Policy-Wechsel ist nur erlaubt zwischen `{frozen, managed}` . Übergänge zu/von `tradable_quote` sind hard-blocked.** Quote-Asset (USDT/USDC etc.) bleibt durable Quote — fundamentale Bot-Annahme. Promote / Release schreiben `managed_state.json` UND `baseline_holdings.json` **transactional** in einer DB-Transaction (Two-File-Atomic-Pattern). Verhindert wallet-signature-Drift bei Policy-Wechsel.	DR-7-Resolution Q-MH-Session 2026-05-10

2 · Globale Stop-Regeln

ID	Regel	Auslöser
G-SR-1	Wenn `baseline_holdings.json` oder `managed_state.json` außerhalb CommandBus mutiert wird → STOP, Forensik	filewatcher / source-grep test
G-SR-2	Wenn proposal_engine ohne `risk_model_version` Output liefert → STOP, abort proposal	engine-Validator
G-SR-3	Wenn Promote-Versuch ohne dazugehöriges `managed.asset_promoted`-Audit von einem `actor=user_*` → STOP	DR-11 enforcement
G-SR-4	Wenn Bot-Worker autonom `managed_active → frozen` versucht → STOP + audit `managed.policy_change_blocked`	DR-11
G-SR-5	Wenn `command_type_registry_version` zwischen GUI-DB und `command_bus_versions.json` driftet → STOP	G6.5 invariant
G-SR-6	Wenn Bot-PID ohne dokumentierten Phasen-Trigger wechselt → STOP, externer Eingriff prüfen	DR-10
G-SR-7	Wenn Drift-Detection einen autonomen Sell triggert → STOP, Bot ist Berater bei Drift, nicht Executor	SM-1 (Drift-Freeze)
G-SR-8	Wenn `bot_watchdog.sh` auf `steve-tradingbot` driftet → STOP, `git checkout` erforderlich vor jedem Restart	DR-8 + RECON-2.2b Befund
G-SR-9	Wenn Confidence-Score < Threshold UND Operator klickt Approve ohne Override-Confirm → STOP	SM-4
G-SR-10	Wenn Volatility > Kill-Switch-Schwelle bei managed_active → autonome Transition zu `managed_paused` (kein Sell)	SM-5
G-SR-11	Wenn Mainnet-Apply-Versuch in irgendeinem Layer (Settings.BINANCE_TESTNET, Writer-ALLOWED_ENVIRONMENTS, Worker-ALLOWED_ENVIRONMENTS, payload-validator, GUI hardcode) → STOP, alle 5 Layer müssen halten	DR-3
G-SR-12	Wenn `confidence_score` Feld in `risk_proposals/*.json` fehlt → Proposal als `proposal_aborted` markieren	DR-12 + SM-4

3 · Restart-Policy

Grundregel: Bot-Restarts sind teuer (PID-Wechsel + Cycle-Reset + State-Reload). Jede MH-Phase markiert explizit `Restart required: yes/no`.

Phasen mit Restart-Pflicht (a priori): - Phase die `baseline_bootstrap.bootstrap_baseline()` Verhalten ändert (`auto_import_managed_holdings`) - Phase die `main.py` startup-Sequenz ändert - Phase die `LiveTrader.__init__` settings-snapshot ändert (z.B. Outage-Resilience config)

Phasen ohne Restart-Pflicht: - alle Filament-UI-Phasen (Browser-Reload genügt) - alle Schema/Validator/Service-Phasen die nur `commands` + `audit_events` schreiben - alle Reader-Phasen (`*.snapshot()` liest pro Cycle frisch) - alle proposal_engine-Phasen solange engine standalone callable

Restart-Workflow (durable, analog RECON-2.2b): 1. Pre-Backup-Pflicht (G-DR-6) 2. sha256-Verify Host vs Container pre-cp 3. `docker cp` der modifizierten Files 4. sha256-Verify post-cp 5. Watchdog-Status-Check (G-DR-13) 6. SIGTERM PID + watchdog-getriggerter Neustart 7. Post-Cycle-Verify: keine Tracebacks, keine Verhaltens-Drift, audit_events-Stream weiterhin sauber

4 · Migration-Policy

Grundregel: keine DB-Migration im Rahmen einer einzelnen MH-Phase außer ausdrücklich gekennzeichnet `Migration required: yes`.

Erlaubte Migrationen (durch dedizierte Phase): - neue Tabellen `managed_proposals`, `managed_assets_history` (siehe 05_commandbus_worker) - neue Spalten in bestehenden Tabellen NUR mit Backfill-Plan + Rollback-Plan

Nicht erlaubte Migrationen (in keiner MH-Phase): - Schema-Änderungen an `audit_events` (Tabelle ist live, Pattern stabil) - Schema-Änderungen an `commands` (G6.5 invariant) - Schema-Änderungen an `bot_statuses` / `decision_logs` / `position_snapshots` (T-SPLIT-Pattern)

Migration-Workflow: 1. eigene Phase mit `Migration required: yes` 2. Backup der GUI-DB vor Migrate 3. Migration läuft NUR in TESTNET-DB; Mainnet-DB wird nicht betroffen 4. Rollback-Migration als Companion (artisan migrate:rollback) 5. Tests verifizieren old + new Schema (via `--pretend` für Mainnet)

---

5 · Audit-Policy

Grundregel: jede Mutation generiert mindestens einen audit_events-Row.

Pflicht-Felder pro audit_events-Row: - `event_type` aus enum unter `managed.*`-Prefix (siehe Liste in 03_state_machine) - `user_id` falls Operator-Action, sonst NULL (System/Bot) - `subject_type` = `'ManagedAsset'` oder `'BaselineHoldings'` - `subject_id` = Asset-Symbol (z.B. `'SOL'`) oder `'managed_state'` - `summary` = human-readable 1-Zeile - `metadata.command_id` = UUID des auslösenden Commands (NULL nur bei System-Auto-Transitions) - `metadata.from_state` + `metadata.to_state` für State-Transitions - `metadata.actor` = `'user_<id>'` / `'bot'` / `'system'` (für DR-11 enforcement)

Zusätzliche Snapshot-Files (SM-6, ab Phase 05): - pro State-Transition: `audit_snapshots/<event_id>.json` immutable - enthält vollen `managed_state.json` + relevantes `risk_proposals/<id>.json`-Excerpt - Snapshot-Files NIE überschrieben, NIE gelöscht (nur via separater Archive-Phase)

Sanity: - alle audit_events durch `AuditMetadataScrubber` (G9-2) bereinigen vor Insert - keine Tokens / Secrets / Passwörter in `metadata` - bei Verstoß: source-grep test schlägt fehl

---

6 · CommandBus-Policy

Grundregel: alle Mutationen an State-Files ausschließlich über `commands`-Tabelle.

Erlaubte Command-Types für RECON-MH (Stand v1, erweiterbar): - `request_managed_proposal` - `approve_managed_proposal` - `reject_managed_proposal` - `pause_managed_asset` - `resume_managed_asset` - `release_managed_asset` - `flag_managed_drift` (bot-self-emitted via internal channel) - `dry_run_promote_managed` (SM-7)

Idempotency: - `approve_managed_proposal` + `reject_managed_proposal` haben `idempotency_key = "mh:decide:<proposal_id>"` - Pause/Resume haben distinct keys pro Klick (kein dedup) - request_managed_proposal hat key `"mh:propose:<asset>:<YmdHis>"` (verhindert Doppel-Klicks innerhalb derselben Sekunde)

Forbidden: - direct `managed_state.json` write aus PHP - direct `risk_proposals/*.json` write aus PHP - Worker autonom Command erstellen (außer `flag_managed_drift`)

---

7 · TESTNET-first Policy

Grundregel: alle MH-Phasen entwickeln, testen, drillen auf TESTNET. Mainnet wird in keiner Phase berührt außer `07_mainnet_future` (BACKLOG, BLOCKIERT).

Verbindlich: - jede Phase prüft via Settings.BINANCE_TESTNET=True (Bot-Side) - jede Phase prüft via Writer ALLOWED_ENVIRONMENTS = {'testnet'} - jede Phase prüft via payload validator environment='testnet' - jeder Bot-Worker-Handler enforced testnet-only

TESTNET bleibt permanent erhalten auch wenn ein Mainnet-Feature später aktiviert wird: - Operator kann jederzeit auf TESTNET zurückwechseln - TESTNET-Datenbank + Wallet bleiben separat - (siehe `feedback_testnet_permanent_paper_rename.md` durable rule aus 2026-05-09)

---

8 · Mainnet-Block-Policy (5-Layer)

Grundregel: Mainnet ist durch 5 unabhängige Layer blockiert. Alle 5 müssen halten. Single-Layer-Failure führt nicht zu Mainnet-Activation, weil weitere Layer greifen.

Layer	Komponente	Mechanismus
L1	Settings (`.env`)	`BINANCE_TESTNET=true`
L2	BaselineHoldingsWriter	`ALLOWED_ENVIRONMENTS = {'testnet'}` (Hard-Constant)
L3	command_worker.py	`ALLOWED_ENVIRONMENTS['*'] = {'testnet'}` für alle MH-Command-Types
L4	payload-validator (CommandTypeRegistry)	`environment === 'testnet'` Check
L5	GUI Filament	hardcoded `environment='testnet'` in Service-Layer

Mainnet-Activation erfordert: - explizites RECON-2.5 Sign-Off - alle 5 Layer durch Mehrpersonen-Audit verifiziert - B-FEE-FIX 1-4 + B-OUTAGE-RESILIENCE-1 + Operator-Drill 502 alle grün - Q-MH-1..18 alle entschieden - siehe `07_mainnet_future.md`

---

9 · Drift-Policy (SM-1)

Grundregel: Bot detektiert Drift, alarmiert Operator, **handelt nicht autonom**.

Drift-Triggers (alle drei lösen `managed_drift_alert` aus): - **qty drift:** `|state['positions'].qty - real_wallet.qty| / real_wallet.qty > 5%` - **price drift:** `|current_price - synthetic_entry| > 2 × ATR_14d` - **regime drift:** Markt-Regime wechselt von BEAR ↔ BULL bei aktivem managed Asset

Bot-Verhalten bei Drift: - managed_active → managed_drift_alert (autonom, einzige erlaubte autonome Transition außer SL/TP-Hit) - audit_event `managed.drift_detected` mit `drift_kind ∈ {qty, price, regime, wallet_external}` - KEINE neuen BUYs auf das Asset solange drift_alert - bestehende SL/TP bleiben aktiv (Operator kann override per pause) - KEIN autonomer Sell

Operator-Optionen bei drift_alert: - `managed_drift_alert → managed_active` (override, drift accepted) - `managed_drift_alert → release_pending` (graceful exit) - `managed_drift_alert → managed_paused` (Bot inaktiv, Position bleibt)

---

10 · Rollback-Policy

Grundregel: jede destruktive Operation hat einen Rollback-Pfad.

Operation	Rollback
`apply_baseline_holdings` (RECON-2.3)	`clear_baseline_holdings` (durable, getestet)
`approve_managed_proposal`	`release_managed_asset` (graceful) — kein Auto-Rollback
`pause_managed_asset`	`resume_managed_asset`
`release_managed_asset`	kein Rollback (terminal); Operator muss neu propose'n
Migration	dedizierte rollback-Migration
Bot-Restart	watchdog-getriggerter Neustart auf alte File-Stände nur bei docker cp -Reverse
Worker-Daemon-Aktivierung	`docker compose down clawbot-worker`

Snapshot-basierter Rollback (SM-6): - jeder State-Übergang hat audit-Snapshot - Operator kann theoretisch via dedizierte Phase `restore_managed_state_snapshot` State zurücksetzen - aber: Wallet-Realität ist nicht zurücksetzbar (echte Buy/Sell-Trades sind irreversible) - daher: Rollback nur für State-Files, nie für Position-State

---

11 · Backup-Policy (durable rule aus 2026-05-09)

Vor jeder live-nahen Aktion (`docker cp` / Bot-Restart / Worker-Start / Apply / Migration / .env-Mutation): - pg_dump GUI-DB - `live_portfolio.json` Backup - `.env` Backup - `state/` snapshot - Health-Snap (Bot-PID + cmdline + container-status + .env mtime + key-State-Files presence) - Memory tar.gz Snapshot

Erlaubt ohne Backup: - Code-Edit (host-side) - Tests - lokale Commits (kein push) - Doc-Updates

Backup-Ziel: `/srv/shares/backups/<phase_name>_pre/` mit TS-stamped Filenames + MANIFEST mit sha256.

---

12 · Worker-Policy

Grundregel (Stand v1, erweiterbar): Worker läuft heute manuell als `--once`. Daemon-Aktivierung ist eigene Phase (Q-MH-14 entscheidet zeitliche Einordnung).

Manueller Worker (heute): - aufgerufen via `docker exec -w /home/node/.openclaw/workspace/trading clawbot python3 command_worker.py --once` - claimt 1 command via `FOR UPDATE SKIP LOCKED` - prozessiert + exits - nächster Operator-Klick erfordert nächsten manuellen Aufruf

Daemon-Worker (Backlog, Q-MH-14): - läuft als `docker compose --profile workers up -d clawbot-worker` (G10-1 prepared path) - pollt commands-Tabelle in Loop - Pause/Resume-Commands werden so synchron verarbeitet (R4-Race aus Architektur-Doc gelöst) - erforderlich für sichere managed-Lifecycle-UX (UX-2 Multi-Step-Wizard kann sonst zu lang werden)

Forbidden: - Worker-Daemon-Aktivierung außerhalb dedizierter Phase - Worker schreibt direkt in `state['positions']` außer via dokumentierte Handler

---

13 · GUI-Policy

Grundregel: GUI ist read-mostly + Action-Trigger. **Niemals** direkt File-Writer am Bot-State.

Erlaubt für GUI: - read von DB-Tabellen (`audit_events` , `commands` , `managed_proposals` , `managed_assets_history` , `bot_statuses` , etc.) - read von Bot-Side State-Files via dediziertem Read-Endpoint (z.B. `bot_statuses.metadata_json.managed_summary`) - INSERT in `commands` + `audit_events` über Service-Layer - INSERT in `managed_proposals` + `managed_assets_history` über Service-Layer (Cache-Tabellen) - Filament-Pages, Resources, Widgets, Modals, Hard-Confirm-Forms - Polling alle 30s

Forbidden für GUI: - direct file-write (`baseline_holdings.json` , `managed_state.json` , `runtime_config.json` , `risk_proposals/*.json` , `state['positions']` , etc.) - direct ccxt-API-Calls - direkte Bot-Process-Interaktion (kill / signal) - direkte `.env` -Mutation - direct Worker-Spawn

Admin-only Surfaces: - Apply-/Clear-/Approve-/Reject-/Pause-/Resume-/Release-Actions sind admin-only - Operator-Role darf: view + request_proposal - canAccess() pro Page admin-only verifiziert (analog G7-3 / G9-3 / G10-6)

---

14 · State-File-Policy

Vier kanonische State-Files unter `trading/state/` :

File	Owner	Schreibt	Liest
`baseline_holdings.json`	BaselineHoldingsWriter	Worker (apply/clear)	BaselineHoldingsReader, baseline_bootstrap
`managed_state.json`	ManagedStateWriter (Phase 04+)	Worker (alle managed.* Handler)	ManagedStateReader, baseline_bootstrap, universe filter, execute_buy guards
`risk_proposals/*.json`	ProposalWriter (Phase 02)	proposal_engine	ProposalReader, GUI
`audit_snapshots/*.json`	AuditSnapshotWriter (Phase 05+)	Worker bei jeder State-Transition	Restore-Phase (Backlog)

Schreibe-Pattern für ALLE 4: 1. atomic write: tmp-file + fsync + os.replace 2. Backup-before-mutate (sha256 + Backup-File) 3. canonical-hash idempotency (no_effect=true bei identischem Inhalt) 4. ALLOWED_ENVIRONMENTS = {testnet} hard-Constant 5. MainnetBlockedError raise bei environment != 'testnet' 6. emit audit_event BEFORE return (transactional)

Forbidden Schreibe-Pattern: - direct `.write_text()` ohne tmp + fsync + replace - direct file mutation aus PHP (G-DR-3) - silent overwrites ohne Backup

14.1 · JSON ↔ DB Source-of-Truth-Hierarchie (Q-MH-15 Resolution, 2026-05-10)

managed_state.json + baseline_holdings.json + risk_proposals/*.json sind durable Single Source of Truth (SoT) für den Bot. GUI-DB-Tabellen (`managed_proposals`, `managed_assets_history`) sind **Read-Cache** für Filament-UI.

Schreibe-Reihenfolge bei jeder managed-State-Mutation:

1. Worker validiert Payload + Guard
2. Worker schreibt **JSON zuerst** (atomic + sha256 + Backup)
3. Worker schreibt **DB-Cache in selber DB-Transaction**
4. bei JSON-Write-Failure: keine DB-Mutation
5. bei DB-INSERT-Failure nach JSON-Write: Worker emittet `managed.cache_drift_detected` audit + Retry-Logik beim nächsten Cycle

Forbidden: - GUI schreibt direkt in DB-Cache (Filament Service-Layer wird nur über CommandBus aktiv) - Bot liest DB-Cache (Bot liest IMMER aus JSON) - DB-Cache wird Source of Truth (kein Read-Path Bot ↔ DB-Cache)

14.2 · Two-File-Atomic-Pattern (G-DR-14 Resolution, 2026-05-10)

Anwendbar: alle Phasen die `managed_state.json` UND `baseline_holdings.json` gleichzeitig mutieren (insbesondere `approve_managed_proposal` + `release_managed_asset`).

Sequenz:

1. Worker prüft Guard (G-DR-14: keine tradable_quote-Crossings)
2. Worker schreibt baseline_holdings.json (atomic + sha256 + Backup, neue policy + neuer canonical_hash + neuer account_hash)
3. Worker schreibt managed_state.json (atomic + sha256 + Backup, neuer state)
4. Worker schreibt audit_event + audit_snapshot mit beiden Pre/Post-States in derselben DB-Transaction
5. bei Failure auf einem der Schritte: Backup-Restore von baseline_holdings.json + managed_state.json + audit `managed.two_file_atomic_rollback`

Verhindert: signature-Drift beim nächsten Bot-Restart (account_hash bleibt konsistent gegen baseline.holdings).

---

15 · Cross-Reference-Header (verbindlich pro Phase-File)

Jede der 11 phase-`.md`-Files unter diesem Verzeichnis MUSS am Anfang folgende Tabelle enthalten:

```
## Phase Cross-Reference

| Field | Value |
|---|---|
| Dependencies | <Liste anderer .md-Files die vorher gelesen sein müssen> |
| Required previous phases | <RECON-MH-X.Y abgeschlossen> oder "none" |
| Restart required | yes / no |
| Migration required | yes / no |
| Mainnet-touch | always: no (außer 07_mainnet_future) |
| Risk-Level | low / medium / high |
```

Plus die folgenden 4 verbindlichen Sektionen:

- `## Allowed` — phasenspezifisch, nicht generisch (z.B. „neue Reader OK", NICHT „Code OK")
- `## NOT Allowed` — phasenspezifisch (z.B. „kein Worker-Daemon-Start", NICHT „kein Code")
- `## Memory / Durable Rules` — Liste der DR/G-DR/SR/G-SR-IDs die in dieser Phase betroffen sind
- `## Expected Test Surface` — geschätzte Testanzahl + Test-Typen + bestehende Suites die betroffen sind

---

16 · Was dieser Vertrag NICHT regelt

- Konkrete Implementierung (Code) → erst in dedizierten Phasen
- Konkrete UI-Designs (Pixel) → kommen via Claude-Design-Iteration

- Konkrete Testimplementierung (test files) → erst wenn Phase aktiviert
- Operator-Antworten zu Q-MH-1..18 → siehe `08_open_questions.md`

---

17 · Versionierung dieses Vertrags

Version	Datum	Änderung
v1	2026-05-10	Initial-Erstellung nach Konsolidierung von Architektur-Doc + Reviewer-Bewertung + User-Roadmap-Split-Prompt + Master-Boundaries-Antwort
v2	**2026-05-10**	**Q-MH-Session-Decisions integriert:** (1) neue **G-DR-14** (tradable_quote-Lock + Two-File-Atomic-Pattern); (2) §14 erweitert um §14.1 JSON-SoT-Hierarchie (Q-MH-15 Hybrid C); (3) §14 erweitert um §14.2 Two-File-Atomic-Pattern (DR-7 Resolution). Q-MH-2/11/13/14/15 + DR-7 alle decided.

Bei Änderungen an einer DR / G-DR / SR / G-SR / Layer-Liste / Policy: Version bumpen + Changelog-Eintrag.

— Ende Master Boundaries v1 —

📄 `00_overview.md`

00 · RECON Managed Holdings — Overview

Status: Architektur-Closure (read-only) **Phase-Key:** RECON-MH-0 **Vorbild-Pattern:** G10-Apply-State-Machine + RECON-2.1/2.2/2.3 + CommandBus G6.5 **Bezug:** zentrale Eintrittsstelle für die gesamte RECON-MH-Sub-Roadmap

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md` zwingend zuerst lesen
Required previous phases	none — Eintritts-Phase
Restart required	no
Migration required	no
Mainnet-touch	no
Risk-Level	low (read-only)

Allowed

- Architektur-Dokumentation
- Verweise auf andere `.md`-Files in dieser Sub-Roadmap
- State-Machine-Diagramme als ASCII / Mermaid
- Verlinkung zu bestehenden Memory-Files (`recon_status_pin.md`, `recon_managed_holdings_architecture.md`)
- Q-MH-Cross-References zu `08_open_questions.md`
- Roadmap-Phasen-Tabelle mit Status (closed/open/backlog)

NOT Allowed

- Konkrete Implementierungs-Hinweise (gehören in Phase-Files 01-06)
- Detaillierte Schemas (gehören in 01_foundation / 02_risk_proposal_engine / 03_state_machine)
- Pixel-Design-Beschreibungen (gehören in 04_gui_operator_flow)
- Q-MH-Antworten (gehören in 08_open_questions)
- Cross-Phase-Referenzen ohne Verlinkung (jede Referenz braucht expliziten File-Path)

Memory / Durable Rules

ID	Inhalt
G-DR-1	Frozen-by-default
G-DR-2	Operator final authority
G-DR-4	TESTNET-first dauerhaft
G-DR-7	Multi-Step Wizard für Promote
G-DR-9	Bot darf nicht autonom managed → frozen
G-DR-13	Watchdog-clawbot-Konvention
feedback_testnet_permanent	TESTNET bleibt permanent erhalten

Expected Test Surface

- 0 Tests (read-only doc)
- aber: existence-Tests in 09_test_strategy referenzieren dieses File via Index-Check
- bestehende Suite: keine

1 · Gesamtvision

Operator möchte bestehende Spot-Holdings explizit an den Bot übergeben können. Der Bot übernimmt **niemals** automatisch. Stattdessen:

```
frozen → Bot analysiert → risk_proposal → Operator bestätigt → managed_active
```

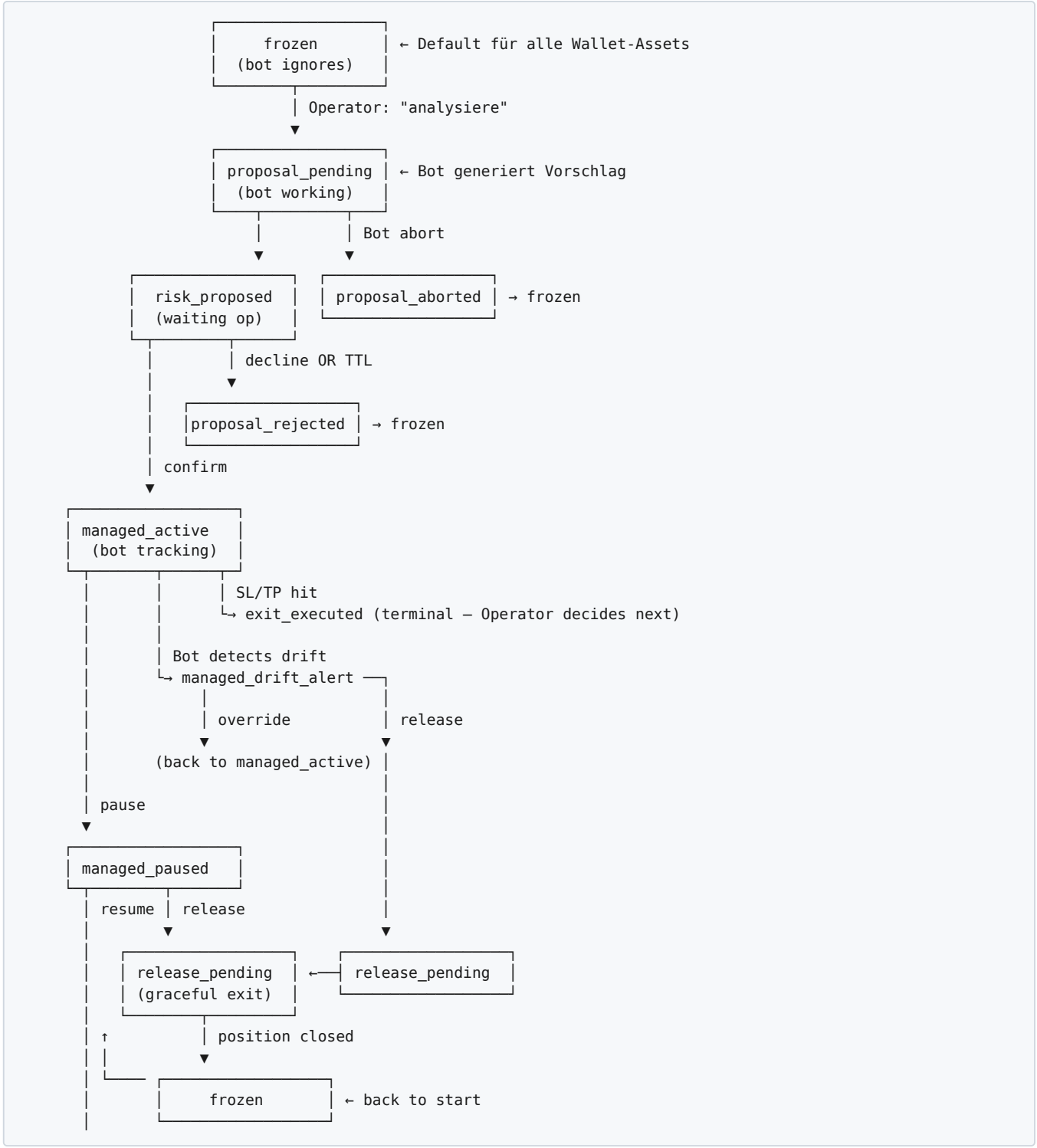
Bot ist Berater. Operator ist final authority. Mainnet bleibt blockiert.

2 · Architektur-Zusammenfassung

Schicht	Komponenten
Bot-Side State-Files	`baseline_holdings.json` (existing), `managed_state.json` (neu), `risk_proposals/*.json` (neu), `audit_snapshots/*.json` (neu, ab Phase 05)
Bot-Side Module	`BaselineHoldings{Reader,Writer}` (existing), `ManagedState{Reader,Writer}` (neu), `proposal_engine.py` (neu), `proposal_{reader,writer}.py` (neu), erweiterung von `baseline_bootstrap.py`, `command_worker.py`, `scanner/universe.py`, `execution/paper_trade.py`, `execution/live_trade.py`
GUI-Side Services	`ApplyBaselineService` (existing), `ProposalService` (neu), `ManagedStateService` (neu), `ProposalSchemaRegistry` (neu)
GUI-Side UI	`BaselineHoldings` Page (existing), `ManagedHoldings` Page (neu), `ManagedProposalResource` (neu)
GUI-Side DB-Cache	`audit_events` (existing), `commands` (existing), `managed_proposals` (neu), `managed_assets_history` (neu)
CommandBus Types (neu)	`request_managed_proposal`, `approve_managed_proposal`, `reject_managed_proposal`, `pause_managed_asset`, `resume_managed_asset`, `release_managed_asset`, `flag_managed_drift`, `dry_run_promote_managed`

Detailbeschreibungen in den entsprechenden Phase-Files (siehe §6 unten).

3 · State-Machine (10 States)



Vollständige State-Tabelle + Übergangs-Regeln in `03_state_machine.md`.

---

4 · Gesamt-Phasen-Roadmap

Phase	Inhalt	Restart	Mig	Mainnet	Risk
MH-0	Architektur-Closure (dieses File + 11 weitere)	–	–	no	low
MH-0.5	**Worker-Daemon-Aktivierung** (`docker compose --profile workers up -d clawbot-worker`) + Healthcheck + 24h-Stabilitäts-Beobachtung. Vorbedingung für MH-1. Q-MH-14 decided 2026-05-10. Reversibel via `docker compose down clawbot-worker` .	no (Daemon ist eigener Container)	no	no	low-medium
MH-1	PHP CommandTypeRegistry + 8 neue Command-Types + Tests	no	no	no	low
MH-2	Bot-Side `managed_state_reader` + `proposal_reader` (dormant) + Tests	no	no	no	low
MH-3	`proposal_engine.py` standalone + Tests	no	no	no	medium
MH-4	PHP `ProposalService` + `ManagedStateService` + Tests	no	yes (DB-Cache-Tabellen)	no	medium
MH-5	Filament `ManagedHoldings` Page + `ManagedProposalResource` + Multi-Step Wizard + Tests	no	no	no	medium
MH-6	Bot-Worker Handler für 8 Command-Types + Tests	no	no	no	medium
MH-7	Bot-Side Wiring (bootstrap / universe / execute_buy / drift-hook) + Tests	**yes**	no	no	high
MH-8	Testnet-Drill (frozen-only erst, dann managed-promote E2E)	indirekt	no	no	high
MH-9	Worker-Daemon-Aktivierung + scheduled jobs (TTL-expiry / reconciliation) + Mainnet-Pre-Sign-Off-Audit	yes	no	no	high

→ 9 Code-Phasen plus Architektur-Phase 0 plus Daemon-Vorbereitungs-Phase 0.5. Jede einzeln Approval-Required.

Q-MH-Decisions (2026-05-10) verankert: - Q-MH-14 → MH-0.5 (Worker-Daemon vor MH-1) - Q-MH-15 → JSON ist Bot-SoT, DB ist GUI-Cache (Hybrid C); §14.1 in Master-Boundaries - Q-MH-13 → 3 Mainnet-Disables (aggressive/DCA/t2_pump_dump-recommended) - Q-MH-2 → Engine 3 Varianten, Output env-abhängig (Testnet 3, Mainnet 2) - Q-MH-11 → Hard-Confirm `<asset>:<variant>` + Override-Pattern - DR-7 → G-DR-14 + Two-File-Atomic-Pattern; §14.2 in Master-Boundaries

5 · Globale Stop-Regeln (gekürzt — vollständig in `99_master_boundaries.md`)

- G-SR-1: Mutation außerhalb CommandBus → STOP
- G-SR-2: Proposal ohne `risk_model_version` → STOP
- G-SR-3: Promote ohne user-actor-Audit → STOP (DR-11)
- G-SR-4: Bot autonom managed → frozen → STOP (DR-11)
- G-SR-7: Drift-Detection autonomous-Sell → STOP
- G-SR-11: Mainnet-Apply-Versuch → STOP (5-Layer halten)

6 · Verweise zu Teilphasen

File	Inhalt	Status
01_foundation.md	baseline_holdings + managed_state Schemas + Reader/Writer	architektur-defined
02_risk_proposal_engine.md	proposal_engine + Confidence + Volatility-Kill + Dry-Run	architektur-defined
03_state_machine.md	10 States + Übergänge + Drift-Freeze + Cooldown + TTL	architektur-defined
04_gui_operator_flow.md	Multi-Step Wizard + 5 UX-Regeln + Filament-Pages	architektur-defined
05_commandbus_worker.md	Apply-Flow + Audit-Snapshot + Restart-Strategie	architektur-defined
06_testnet_drill.md	Drill-Plan + Acceptance-Criteria	architektur-defined
07_mainnet_future.md	Mainnet-BACKLOG, BLOCKIERT	backlog
08_open_questions.md	Q-MH-1..18 (alle 18 offenen Operator-Fragen)	open
09_test_strategy.md	Test-Plan + Boundary-AST + Drift-Tests	architektur-defined
10_backlog_future_extensions.md	CAP-1 / T-SPLIT-5/6 / Multi-wallet / Copy-Trading / Tax-Layer	backlog
99_master_boundaries.md	zentraler Vertrag: durable rules + stop-regeln + 14 Policies	verbindlich

7 · Cross-Reference zu globaler RECON-Roadmap

Diese Sub-Roadmap ist Teil der größeren RECON-Roadmap. Globaler Stand siehe:

- `memory/recon_status_pin.md` — RECON-2.1 ☑ → 2.2a ☑ → 2.2b ☑ → 2.3 ☑ → 2.3c ☑ → **RECON-2.3-DEPLOY** (BACKLOG) → **RECON-2.4 Frozen-only Drill** (BACKLOG) → **RECON-MH-1..9** (diese Sub-Roadmap) → **RECON-2.5 Mainnet-Readiness** (BACKLOG)
- `docs/roadmap/recon_managed_holdings_architecture.md` — monolithisches Quell-Doc dieser Sub-Roadmap (574 Zeilen, bleibt als Synthesis-Referenz erhalten)
- `docs/roadmap/mainnet_preflight_recon.md` — originaler RECON-Phasen-Plan + Q-1..Q-12

8 · TESTNET-first + Mainnet-Block (Kurzfassung)

- **TESTNET** ist permanente Entwicklungsumgebung — bleibt auch nach späteren Mainnet-Aktivierungen erhalten (durable rule)
- **Mainnet** ist 5-Layer-blockiert: Settings + Writer + Worker + Validator + GUI (siehe §8 in `99_master_boundaries.md`)
- Mainnet-Aktivierung ausschließlich über `RECON-2.5 Mainnet-Readiness Review` mit:
- 2-Personen-Sign-Off
- alle 5 Layer manuell + automatisiert verifiziert
- Q-MH-1..18 entschieden
- B-FEE-FIX 1-4 + B-OUTAGE-RES-1 + Operator-Drill-502 alle grün
- dry-run + read-only Mainnet-Apply zuerst
- 5-min-Rollback-Plan getestet

9 · Onboarding-Pfad für neue Entwickler

Reihenfolge zum Lesen:

1. **`99_master_boundaries.md`** (verbindlicher Vertrag — alles andere baut darauf auf)
2. Dieses File (`00_overview.md`) — Big Picture
3. `08_open_questions.md` — was entschieden ist + was offen
4. `01_foundation.md` — Schemas + Datenfluss
5. `03_state_machine.md` — Lebenszyklus
6. weitere Files je nach Aufgabe

— Ende Overview —

📄 `01_foundation.md`

01 · Foundation — Schemas, Reader, Writer, Hashing

Status: architektur-defined **Phase-Key:** RECON-MH-1 (PHP-Registry) + RECON-MH-2 (Bot-Reader) **Bezug:** baut auf RECON-2.1 + RECON-2.3 (PHP-Side baseline_holdings) auf

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `00_overview.md`
Required previous phases	RECON-2.1 ☑, RECON-2.3 ☑ (PHP-Side BaselineHoldings), RECON-2.3c ☑ (Bot-Worker Handler)
Restart required	no
Migration required	yes (Phase MH-4: neue Tabellen `managed_proposals`, `managed_assets_history`)
Mainnet-touch	no
Risk-Level	low (Reader+Schema), medium (Writer+Migration in MH-4)

Allowed

- neue Reader-Klassen (`ManagedStateReader`, `ProposalReader`) — pure SELECT, kein File-Mutate
- neue Writer-Klassen (`ManagedStateWriter`, `ProposalWriter`) — analog `BaselineHoldingsWriter`-Pattern
- neue JSON-State-Files (`managed_state.json`, `risk_proposals/<id>.json`) — leer/dormant bis Phase MH-4
- neue Eloquent-Models (`ManagedProposal`, `ManagedAssetHistory`) für DB-Cache
- Migration für DB-Cache-Tabellen (eigene Migration-Phase, nicht inline)
- Tests für Schema-Validation, Hash-Determinism, Idempotency
- Cross-Reference zu G10-RuntimeConfig-Pattern als Vorbild
- atomic-write + sha256-Backup-Pattern

NOT Allowed

- direct file-write aus PHP (G-DR-3)
- Schema-Änderungen an `audit_events` / `commands` / `bot_statuses` / `decision_logs`
- Reader gibt MUTABLE-Strukturen zurück (alles muss frozen/immutable sein)
- Writer ohne Backup-before-mutate
- Writer ohne canonical-hash idempotency
- Hashing-Algorithmus von `compute_account_hash` ändern (DR-7 invariant — tradable_quote bleibt excluded)
- `'managed'` als `default_policy_for_unlisted` zulassen (G-DR-8)
- environment != 'testnet' an irgendeiner Stelle (G-DR-4)

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-1	Frozen-by-default	`default_policy_for_unlisted = 'frozen'` Default
G-DR-3	CommandBus-only Mutation	Writer-API only callable from `command_worker.py` Handler
G-DR-5	Audit-heavy	jede Apply/Clear schreibt audit_event vor return
G-DR-6	Backup-before-mutate	tmp-file + sha256 + Backup-Datei vor `os.replace`
G-DR-8	managed nie als default	hard-Constant `ALLOWED_DEFAULTS_FOR_UNLISTED = {frozen, tradable_quote}`
G-DR-10	Risk-Proposals versioniert	`proposal_version` + `risk_model_version` Pflicht-Felder
G-DR-12	Wallet-Signature excludes tradable_quote	`compute_account_hash` skipt `policy='tradable_quote'` Slots
RECON-2.1-pattern	atomic-write Pattern	tmp + fsync + os.replace; Backup-File `.<TS>.bak`

Expected Test Surface

- ~30 Tests Bot-Side
- 10 Reader: snapshot/None/policy_for/wallet_signature_matches/managed_assets/frozen_assets
- 10 Writer: apply/clear/no_effect/idempotency/MainnetBlocked/ValidationError/Backup-creation/restore_from_backup
- 5 Hash-Determinism: account_hash + canonical_hash mit/ohne tradable_quote, key-order-stable
- 5 Boundary-AST: keine direct file_write außerhalb Writer-Klassen, keine `'managed'` als default
- ~25 Tests PHP-Side
- 10 ManagedProposal Eloquent + Validators
- 10 ManagedStateService methodologische Tests
- 5 Migration up/down Tests
- bestehende Suites NICHT betroffen (additive Erweiterung)

---

1 · `baseline_holdings.json` (existing — Erinnerung)

Schema bleibt unverändert (RECON-2.1):

```
{
  "_meta": {
    "schema_version": "recon-2.1",
    "captured_via": "operator_apply",
    "canonical_hash": "<64-hex sha256>",
    "command_id": "<uuid>"
  },
  "captured_at": "2026-05-10T...",
  "environment": "testnet",
  "default_policy_for_unlisted": "frozen | tradable_quote",
  "wallet_signature": {
    "exchange": "binance",
    "account_hash": "sha256:<64-hex>",
    "total_assets": 47
  },
  "holdings": {
    "BTC":  { "qty_at_baseline": 1.0,  "policy": "frozen" },
    "SOL":  { "qty_at_baseline": 5.0,  "policy": "managed" },
    "USDT": { "qty_at_baseline": null, "policy": "tradable_quote" }
  }
}
```

Policy-Konstanten (hard, an 4 Stellen identisch): - `ALLOWED_POLICIES = {frozen, managed, tradable_quote}` (Bot-Writer + PHP-Allowlist + Bot-Reader-Validator + Worker-Handler) - `ALLOWED_DEFAULTS_FOR_UNLISTED = {frozen, tradable_quote}` (G-DR-8) - `ALLOWED_ENVIRONMENTS = {testnet}` (G-DR-4)

---

2 · `managed_state.json` (NEU)

Eigene Datei — Lifecycle-State separiert von baseline.json. Vermeidet Vermischung „Operator-Default-Konfiguration" mit „Live-Lifecycle-State".

```
{
  "_meta": {
    "schema_version": "recon-mh-1",
    "captured_at": "2026-05-10T...",
    "environment": "testnet"
  },
  "managed_assets": {
    "SOL": {
      "state": "managed_active",
      "current_proposal_id": "uuid-...",
      "synthetic_entry": 145.32,
      "synthetic_entry_method": "current_price | cost_basis | vwap | operator_override",
      "stop_loss": 130.0,
      "take_profit": 165.0,
      "max_allocation_usdt": 500.0,
      "strategy_group": "t1_core",
      "promoted_at": "2026-05-10T...",
      "promoted_by_user_id": 42,
      "history": [
        {
          "ts": "2026-05-10T...",
          "from": "frozen",
          "to": "proposal_pending",
          "actor": "user_42",
          "event_type": "managed.asset_proposal_requested"
        }
      ]
    }
  },
  "pending_proposals": {
    "DOGE": {
      "proposal_id": "uuid-...",
      "state": "proposal_pending | risk_proposed",
      "requested_at": "...",
      "requested_by_user_id": 42,
      "expires_at": "..."
    }
  }
}
```

Wichtige Eigenschaft: `managed_state` speichert NICHT die Live-`qty`. Live-qty kommt immer aus `state['positions']`. Reconciliation-Job vergleicht beides per-cycle (siehe `03_state_machine` Drift-Detection).

---

3 · `risk_proposals/<proposal_id>.json` (NEU)

Eine Datei pro Proposal — durable Audit, NIE überschrieben, NIE gelöscht (außer nach TTL-Expiry-Cleanup-Job mit Archive-Move).

```
{
  "proposal_id": "uuid-...",
  "asset": "SOL",
  "proposal_version": 1,
  "risk_model_version": "phase1-cautious-v1",
  "generated_at": "2026-05-10T...",
  "generated_by": "bot_proposal_engine",
  "expires_at": "2026-05-17T...",

  "market_context": {
    "current_price": 145.32,
    "atr_14d": 8.20,
    "volatility_30d_pct": 12.4,
    "volume_24h_usdt": 1234567890,
    "spread_pct": 0.05,
    "liquidity_score": 0.85,
    "regime": "BEAR",
    "regime_confidence": 0.72,
    "support_level": 138.0,
    "resistance_level": 158.0
  },

  "strategy_recommendation": {
    "strategy_group": "t1_core",
    "rationale": "...",
    "alternative_groups": []
  },

  "risk_recommendation": {
    "synthetic_entry_method": "current_price",
    "synthetic_entry_value": 145.32,
    "stop_loss": 130.00,
    "stop_loss_pct": 10.5,
    "take_profit": 165.00,
    "take_profit_pct": 13.5,
    "trailing_stop_enabled": false,
    "rr_ratio": 1.29,
    "max_allocation_usdt": 500.0,
    "max_allocation_pct_of_portfolio": 5.0,
    "dca_recommended": false,
    "dca_settings": null
  },

  "context_warnings": {
    "is_blacklisted": false,
    "is_stablecoin_or_peg": false,
    "concentration_warning": false,
    "correlation_with_existing_positions": [],
    "regime_compatibility": "ok",
    "circuit_breaker_state": "closed"
  },

  "confidence": {
    "overall_score": 0.65,
    "explanation": "..."
  },

  "alternative_proposals": [
    { "variant": "conservative", "stop_loss": 120.0, "take_profit": 155.0, "max_allocation_usdt": 250.0 },
    { "variant": "aggressive",   "stop_loss": 138.0, "take_profit": 175.0, "max_allocation_usdt": 750.0 }
  ]
}
```

Versionierung (G-DR-10): - `proposal_version` ist semantischer Schema-Version (1, 2, 3 ...) — bumpen wenn Top-Level-Struktur ändert - `risk_model_version` ist HARDCODED in der Engine als String — bumpen bei jedem Engine-Algorithmus-Update - Beispiele: `"phase1-cautious-v1"`, `"phase1-cautious-v2"`, `"phase2-ml-v1"`

---

4 · GUI-DB Tabellen (Cache + GUI-Read, eigene Migration in MH-4)

`managed_proposals`

```
CREATE TABLE managed_proposals (
    proposal_id           UUID PRIMARY KEY,
    asset                 VARCHAR(32) NOT NULL,
    state                 VARCHAR(32) NOT NULL,               -- proposal_pending|risk_proposed|...
    proposal_json         JSON,                               -- Inhalt der proposal-Datei
    generated_at          TIMESTAMP,
    expires_at            TIMESTAMP,
    decided_at            TIMESTAMP NULL,
    requested_by_user_id  BIGINT NULL,
    decided_by_user_id    BIGINT NULL,
    decision              VARCHAR(16) NULL,                   -- approved|rejected|expired
    created_at            TIMESTAMP DEFAULT now(),
    updated_at            TIMESTAMP DEFAULT now()
);

CREATE INDEX idx_managed_proposals_state    ON managed_proposals(state);
CREATE INDEX idx_managed_proposals_asset    ON managed_proposals(asset);
CREATE INDEX idx_managed_proposals_expires  ON managed_proposals(expires_at);
```

`managed_assets_history`

```
CREATE TABLE managed_assets_history (
    id            BIGSERIAL PRIMARY KEY,
    asset         VARCHAR(32) NOT NULL,
    state_from    VARCHAR(32),
    state_to      VARCHAR(32) NOT NULL,
    actor         VARCHAR(64) NOT NULL,         -- 'user_<id>' | 'bot' | 'system'
    event_type    VARCHAR(64) NOT NULL,
    metadata_json JSON,
    ts            TIMESTAMP NOT NULL
);

CREATE INDEX idx_managed_history_asset_ts ON managed_assets_history(asset, ts);
CREATE INDEX idx_managed_history_event    ON managed_assets_history(event_type);
```

Source of truth bleibt JSON: `managed_state.json` + `risk_proposals/*.json` Bot-Side. DB ist Read-Cache. Bot-Worker schreibt nach jedem State-Update sowohl JSON als auch DB-Cache (transactional via Worker).

Q-MH-15 Decision (2026-05-10): Hybrid C — JSON ist SoT für Bot, DB ist Cache für GUI. Schreibe-Reihenfolge verbindlich (siehe `99_master_boundaries.md §14.1`): 1. Worker validiert Payload + Guard 2. Worker schreibt JSON zuerst (atomic + sha256 + Backup) 3. Worker schreibt DB-Cache in selber DB-Transaction 4. bei JSON-Failure: keine DB-Mutation 5. bei DB-INSERT-Failure nach JSON-Write: Worker emittet `managed.cache_drift_detected` audit + Retry-Logik

Bot liest IMMER JSON (kein Bot ↔ DB-Cache Read-Pfad). GUI liest IMMER DB-Cache (kein GUI ↔ JSON Read-Pfad).

G-DR-14 (Hard-Lock) seit 2026-05-10: Policy-Wechsel ist nur erlaubt zwischen `{frozen, managed}`. Übergänge zu/von `tradable_quote` sind **hard-blocked**. BaselineHoldingsAllowlist-Validator (Phase MH-1) prüft das bei jedem Apply-Versuch.

---

5 · `compute_account_hash` (durable, RECON-2.2a-Refinement)

Algorithmus (DR-7): 1. holdings nach Asset-Symbol sortieren (lexikographisch) 2. für jede Position: - falls `policy == 'tradable_quote'` → SKIP - sonst: emit `"<asset>=<qty>"` Zeile 3. Zeilen mit `\n` joinen 4. sha256 → `"sha256:<64-hex>"`

```
def compute_account_hash(holdings: dict) -> str:
    parts = []
    for asset in sorted(holdings.keys()):
        slot = holdings[asset] or {}
        if isinstance(slot, dict) and slot.get("policy") == "tradable_quote":
            continue
        qty = slot.get("qty_at_baseline") if isinstance(slot, dict) else None
        parts.append(f"{asset}={qty}")
    raw = "\n".join(parts).encode("utf-8")
    return "sha256:" + hashlib.sha256(raw).hexdigest()
```

Warum tradable_quote ausschließen: - USDT/USDC ändern sich bei jedem Trade - inklusive würde signature_match nach jedem Cycle brechen - frozen + managed sind die meaningful Invariante

Cross-Phase-Konflikt (offen): wenn Operator promote frozen → managed via baseline-Apply, ändert sich Asset's Policy → nicht aber qty (gleiches Asset, gleiche menge) → Hash bleibt gleich. ABER: wenn Operator parallel `tradable_quote → managed` umschaltet, kommt qty von null auf qty → Hash ändert sich. Siehe Q-MH-2 in `08_open_questions.md`.

---

6 · canonical_hash (Idempotency)

Verschieden vom account_hash. Nutzt **vollen Inhalt** für Idempotency-Check:

```
def canonical_checksum(holdings: dict, default_policy: str) -> str:
    parts = [f"_default={default_policy}"]
    for asset in sorted(holdings.keys()):
        slot = holdings[asset] or {}
        policy = slot.get("policy")
        qty = slot.get("qty_at_baseline")
        synth_entry = slot.get("synthetic_entry")
        parts.append(f"{asset}|policy={policy}|qty={qty}|entry={synth_entry}")
    return hashlib.sha256("\n".join(parts).encode()).hexdigest()
```

Verhalten: - identische Apply-Calls (gleiche holdings + default_policy) → gleicher canonical_hash → Writer setzt `no_effect=True` - Bot's `_meta.canonical_hash` Feld speichert den Hash für nächsten Vergleich - Operator-getriggerter zweimaliger Apply derselben Datei = no-op

---

7 · Backup-before-mutate

Pattern (durable, RECON-2.1):

```
def apply(self, holdings, environment, ...):
    # 1. Validate (raises before any IO)
    _validate_holdings(holdings)
    _validate_environment(environment)

    # 2. Compute canonical_hash, check no_effect
    new_hash = canonical_checksum(holdings, default_policy)
    if self._target.is_file():
        old_meta = json.loads(self._target.read_text())["_meta"]
        if old_meta.get("canonical_hash") == new_hash:
            return ApplyResult(ok=True, no_effect=True, ...)

    # 3. Backup if target exists
    backup_path = None
    if self._target.is_file():
        ts = datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%SZ")
        backup_path = self._target.with_suffix(f".{ts}.bak")
        shutil.copy2(self._target, backup_path)

    # 4. Atomic write
    tmp = self._target.with_suffix(".tmp")
    body = {...}  # full payload
    tmp.write_text(json.dumps(body, indent=2, sort_keys=True))
    os.fsync(tmp.open().fileno())
    os.replace(tmp, self._target)

    # 5. Return result with backup_path for audit
    return ApplyResult(
        ok=True, no_effect=False, target_path=self._target,
        backup_path=backup_path, canonical_hash=new_hash, ...
    )
```

---

8 · Reader-API (pure SELECT)

ManagedStateReader (neu):

```
class ManagedStateReader:
    def __init__(self, state_dir: Optional[Path] = None) -> None: ...
    @property
    def target_path(self) -> Path: ...

    def snapshot(self) -> Optional[ManagedStateSnapshot]: ...
    def asset_state(self, asset: str) -> Optional[str]: ...
    def is_managed_active(self, asset: str) -> bool: ...
    def is_managed_lifecycle(self, asset: str) -> bool: ...   # any state außer frozen
    def proposal_for(self, asset: str) -> Optional[str]: ...   # current_proposal_id
    def history_for(self, asset: str) -> list[dict]: ...

    # NEVER: write, mutate, delete
```

ProposalReader (neu):

```
class ProposalReader:
    def __init__(self, proposals_dir: Optional[Path] = None) -> None: ...

    def get(self, proposal_id: str) -> Optional[ProposalDoc]: ...
    def list_pending_for_asset(self, asset: str) -> list[ProposalDoc]: ...
    def list_expired(self, before_ts: datetime) -> list[ProposalDoc]: ...
    def is_decided(self, proposal_id: str) -> bool: ...
```

Beide returnen frozen `dataclass`-Snapshots, niemals mutable dicts.

---

9 · Restore-from-Backup

Analog `BaselineHoldingsWriter.restore_from_backup()`:

```
@staticmethod
def restore_from_backup(backup_path: Path, target_path: Path) -> dict:
    """Restore the target_path from a backup. Returns {'ok': bool, 'restore_status': '...'}.
    Does NOT modify backup_path itself."""
    if not backup_path.is_file():
        return {"ok": False, "restore_status": "failed", "error": "backup_missing"}
    shutil.copy2(backup_path, target_path)
    return {"ok": True, "restore_status": "success"}
```

Operator-Phase „restore_managed_state_snapshot" ist Backlog (siehe `10_backlog_future_extensions.md`).

---

10 · Phasen-Sequenz für 01_foundation

MH-1 (PHP): - Erweitere `CommandTypeRegistry` um 8 neue Types (siehe `05_commandbus_worker.md`) - Validators schreiben — keine Service-Implementation - ~25 PHP-Tests

MH-2 (Bot, dormant): - `managed_state_reader.py` + `proposal_reader.py` erstellen - pure SELECT, nie aufgerufen aus aktivem Pfad - ~30 Bot-Tests

MH-4 (PHP, Migration): - DB-Migration für `managed_proposals` + `managed_assets_history` - ManagedStateService + ProposalService implementieren - ~30 Tests

→ kein Bot-Restart in dieser Foundation-Phase.

— Ende Foundation —

📄 `02_risk_proposal_engine.md`

02 · Risk Proposal Engine

Status: architektur-defined **Phase-Key:** RECON-MH-3 **Bezug:** ist die zentrale Bot-Side Analyse-Komponente für Managed-Holdings-Lifecycle

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `01_foundation.md`, `03_state_machine.md`
Required previous phases	RECON-MH-1 ☐, RECON-MH-2 ☐
Restart required	no (Engine ist standalone callable, kein Wiring ins main.py)
Migration required	no
Mainnet-touch	no
Risk-Level	medium (computational complexity + multiple dependencies)

Allowed

- neuer Python-Modul `trading/proposal_engine.py` als reine Berechnungs-Komponente
- read-only Zugriff auf:
- `fetch_balance()` (ccxt)
- `fetch_ticker()` (ccxt)
- `fetch_ohlcv()` (ccxt für ATR/volatility)
- `fetch_order_book()` (ccxt für depth/liquidity)
- `MarketRegime.detect()` (existing)
- `state['positions']` (read-only für correlation)
- `BaselineHoldingsReader.snapshot()` (existing)
- Schreiben von `risk_proposals/<id>.json` via `ProposalWriter` (Phase MH-2 ergänzt)
- Audit-Event `managed.asset_proposal_generated` Emit
- Tests mit Mocks für ccxt + market data
- Confidence-Score-Berechnung als deterministische Funktion
- Volatility-Kill-Switch-Berechnung
- Dry-Run-Modus (= alle Berechnungen, KEIN file-write)

NOT Allowed

- direct write nach `managed_state.json` (das ist Worker-Job)
- direct CommandBus-INSERT (Engine ist Berechnung, nicht Command-Erzeuger)
- Schreibe-Vorgang ohne `risk_model_version` (G-DR-10)
- Schreibe-Vorgang mit `confidence.overall_score` fehlend (G-SR-12)
- ccxt order placement (`create_market_buy_order`, `create_order`, ...)
- t3_copy_trading als `recommended` strategy_group (G-SR-14 — T-SPLIT-3 hat t3 als „planned · disabled" gepinnt)
- Mainnet-Daten-Fetch ohne `BINANCE_TESTNET=true` Check
- Bot-State-Mutation jeglicher Art

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-4	TESTNET-first	Engine prüft Settings.BINANCE_TESTNET
G-DR-10	Risk-Proposals versioniert	jeder Output trägt `proposal_version` + `risk_model_version`
G-DR-11	Audit-Prefix-Separation	Engine emittet `managed.asset_proposal_generated`, nie `baseline.*`
G-SR-12	confidence_score Pflichtfeld	Engine raises `ProposalAbortedError` bei fehlenden Score
G-SR-14	t3_copy_trading-Block	Validator vor dem Schreiben prüft strategy_group != t3_copy_trading
SM-4	Confidence-Threshold Hard-Gate	Engine setzt `confidence.requires_operator_override = true` bei Score < 0.X
SM-5	Volatility-Kill-Switch	Engine setzt `context_warnings.volatility_kill_switch = true` ab Schwelle
SM-7	Dry-Run-Modus	Engine kann mit `dry_run=True` Param laufen — keine Datei-Outputs

Expected Test Surface

- ~35 Bot-Tests
- 5 schema-conform: alle Required-Felder gesetzt
- 5 versioning: `risk_model_version` ∈ Whitelist + `proposal_version` ≥ 1
- 5 market-context-Berechnung: ATR, volatility, spread, liquidity_score
- 5 risk-recommendation-Berechnung: SL/TP/RR mit verschiedenen ATR-Werten
- 5 confidence-score: deterministisch + reproduzierbar bei gleichen Inputs
- 5 t3-blacklist: t3_copy_trading wird auf t1_core fallback reduziert
- 5 dry-run: kein File-Write, aber gleicher Output-Inhalt
- 5 boundary-AST: keine ccxt order-API-Aufrufe in Engine-Source
- bestehende Suites NICHT betroffen
- Engine ist standalone aufrufbar in Tests via `proposal_engine.generate(asset, dry_run=True, mock_exchange=...)`

1 · Engine-Aufruf-Signatur

```
class RiskProposalEngine:
    def __init__(
        self,
        exchange: ccxt.Exchange,
        risk_model_version: str = "phase1-cautious-v1",
        confidence_threshold: float = 0.50,        # SM-4
        volatility_kill_threshold_pct: float = 25.0, # SM-5
        baseline_reader: BaselineHoldingsReader,
        market_regime: MarketRegime,
    ) -> None: ...

    def generate(
        self,
        asset: str,                          # z.B. "SOL"
        operator_id: int,
        captured_via: str = "operator_request",
        dry_run: bool = False,
    ) -> ProposalResult: ...
```

ProposalResult:

```
@dataclass(frozen=True)
class ProposalResult:
    ok: bool                       # False bei aborted
    aborted_reason: Optional[str]  # 'no_balance' | 't3_blocked' | 'circuit_breaker_open' | …
    proposal_id: Optional[str]     # uuid, None bei aborted
    proposal_path: Optional[Path]  # None bei dry_run oder aborted
    payload: dict                  # full proposal-JSON content
    written_audit_event: bool      # False bei dry_run oder aborted
```

2 · Engine-Pipeline (10 Schritte)

```
1. validate inputs (asset, operator_id)
2. fetch wallet balance → if qty <= 0 → abort('no_balance')
3. fetch market context (price, OHLCV, OB, regime)
4. compute market_context block (ATR, volatility, spread, liquidity, support/resistance)
5. compute strategy_recommendation
   - default: t1_core
   - t3_copy_trading → fallback to t1_core + warning (G-SR-14)
6. compute risk_recommendation (SL/TP/RR/max_alloc)
   - apply confidence_threshold check (SM-4)
   - apply volatility_kill_threshold check (SM-5)
7. compute context_warnings (blacklist, peg, concentration, correlation)
8. compute confidence (overall_score 0..1 + explanation)
9. compute alternative_proposals (3 Varianten — siehe §3)
10. emit:
    - if dry_run: return payload, no file write, no audit
    - else: ProposalWriter.apply(...) + audit_event managed.asset_proposal_generated
```

3 · Drei Varianten pro Proposal

recommended (default): - SL = current_price - 1.5 × ATR_14d - TP = current_price + 2.0 × ATR_14d (RR ≈ 1.33) - max_allocation_usdt = computed from portfolio_concentration_target

conservative (engerer SL, kleinerer Position): - SL = current_price - 1.0 × ATR_14d - TP = current_price + 1.5 × ATR_14d (RR = 1.5) - max_allocation_usdt = recommended × 0.5

aggressive (weiterer SL, größere Position): - SL = current_price - 2.5 × ATR_14d - TP = current_price + 3.5 × ATR_14d (RR = 1.4) - max_allocation_usdt = recommended × 1.5 - ⚠ auf Mainnet-Profil **disabled** (siehe Q-MH-13)

→ Operator wählt im Wizard (UX-2, siehe `04_gui_operator_flow.md`) eine Variante; kann zusätzlich Parameter overriden.

Q-MH-2 Decision (2026-05-10): Engine berechnet **immer 3 Varianten** intern. Output-Field `alternative_proposals` wird **env-abhängig gefiltert**: - auf **Testnet:** `[conservative, aggressive]` (2 Alternativen + recommended = 3 sichtbar) - auf **Mainnet:** `[conservative]` (1 Alternative + recommended = 2 sichtbar; aggressive ist Q-MH-13 Hard-Disable)

Q-MH-13 Decision (2026-05-10) — 3 Mainnet-Hard-Disables: 1. **aggressive Variante** wird auf Mainnet aus `alternative_proposals[]` entfernt 2. `dca_recommended` ist auf Mainnet hardcoded `false` (egal was Engine-Logik sonst sagen würde) 3. `strategy_group: 't2_pump_dump'` als `recommended` Output → hard-fallback auf `t1_core` mit Warning-Log + audit `managed.t2_recommendation_blocked_mainnet`

Operator-Override (Q-MH-3) erlaubt das **Override nur für SL/TP/max_alloc/trailing_stop** — NIE für strategy_group oder variant-Switch.

4 · Confidence-Score (SM-4)

Deterministische Funktion mit klar gewichteten Inputs:

```
def compute_confidence(market_ctx, regime_ctx, warnings) -> Confidence:
    score = 0.5  # Baseline
    # Liquidity
    if market_ctx.liquidity_score >= 0.8: score += 0.15
    elif market_ctx.liquidity_score >= 0.5: score += 0.05
    else: score -= 0.10
    # Spread
    if market_ctx.spread_pct < 0.05: score += 0.05
    elif market_ctx.spread_pct < 0.20: pass
    else: score -= 0.10
    # Regime-Compatibility
    if warnings.regime_compatibility == "ok": score += 0.05
    elif warnings.regime_compatibility == "weak": score -= 0.10
    else: score -= 0.20
    # Concentration warning
    if warnings.concentration_warning: score -= 0.15
    # Volatility (if NOT kill-switch but borderline)
    if 15 <= market_ctx.volatility_30d_pct < 25: score -= 0.05
    # Clamp
    return Confidence(
        overall_score=round(max(0.0, min(1.0, score)), 2),
        explanation=...,  # human-readable Liste der angewandten Modifier
        requires_operator_override=score < confidence_threshold,
    )
```

Reproduzierbarkeit: identische Inputs → identischer Score (deterministisch, kein Random). **Versionierung:** Algorithmus-Änderung = `risk_model_version` bump.

---

5 · Volatility-Kill-Switch (SM-5)

```
def check_volatility_kill_switch(market_ctx, threshold_pct) -> bool:
    return market_ctx.volatility_30d_pct >= threshold_pct
```

Default-Schwelle: 25 % 30d-Volatility (entscheidet Operator via Q-MH-18).

Auswirkung beim Promote-Versuch: - `context_warnings.volatility_kill_switch = true` - Wizard zeigt **harten Stop**: „Asset volatility too high — promote disabled" - Operator kann nicht approven (Hard-Gate, kein Override-Path) - empfohlene Operator-Action: Asset bleibt frozen + erneute Analyse später

Auswirkung im Lifecycle (managed_active): - siehe `03_state_machine.md` autonome Transition `managed_active → managed_paused` bei Volatility > Schwelle

---

6 · Dry-Run-Modus (SM-7)

```
result = engine.generate(asset="SOL", operator_id=42, dry_run=True)

# result.payload enthält volle Proposal-Struktur
# result.proposal_path ist None
# result.written_audit_event ist False
# kein File geschrieben
# kein audit_event eingefügt
```

Verwendung: - Operator kann im Wizard „Dry-Run-Preview" klicken bevor er sich für request_managed_proposal entscheidet - Dry-Run-Output wird in DB-Cache geschrieben für GUI-Read, **aber NICHT** als physisches `risk_proposals/<id>.json` -File - Confidence + Warnings sind sichtbar - Kein Engagement, kein State-Change

Test-Strategie: - Engine-Tests laufen IMMER im dry-run (keine Side-Effects in unit tests) - Integration-Tests verifizieren dass dry_run=True keine Files schreibt + keine audit-events emittet

---

7 · Abort-Pfade

Reason	Trigger	Nachfolge-Action
`no_balance`	wallet qty == 0	audit `managed.asset_proposal_aborted`, state → frozen, kein Proposal-File
`circuit_breaker_open`	LiveTrader._circuit_breaker.state == 'open'	abort + audit; Operator wartet auf Recovery
`t3_blocked`	strategy_recommendation wäre t3_copy_trading	fallback auf t1_core, warning, **NICHT** abort (siehe §3)
`mainnet_blocked`	Settings.BINANCE_TESTNET == false	RAISE MainnetBlockedError, kein Output
`expired_in_pipeline`	TTL überschritten WHILE generating	abort + audit
`peg_or_stable`	asset in N7.2-Blocklist	abort, peg-asset darf nie managed werden

8 · Engine-Boundaries

Hard-Constraints (durch Tests gepinnt): - Engine darf KEINE `commands`-Tabelle-INSERTs - Engine darf KEINE direct `managed_state.json` mutation - Engine darf KEINE ccxt-order-API-Aufrufe (`create_market_*`, `cancel_order`, ...) - Engine schreibt AUSSCHLIESSLICH: - `risk_proposals/<id>.json` (via ProposalWriter) - 1 audit_event-Row `managed.asset_proposal_generated` - Engine ist NICHT in `main.py` Cycle-Loop wired (kein per-Cycle-Aufruf) — wird nur vom Worker getriggert

Source-Grep-Test-Patterns (für 09_test_strategy): - `grep "create_market" trading/proposal_engine.py` → leer - `grep "managed_state\.json" trading/proposal_engine.py` → leer (außer Doc-Comments) - `grep "INSERT INTO commands" trading/proposal_engine.py` → leer

9 · Phase-Sequenz für 02_risk_proposal_engine

Voraussetzungen: - MH-2 abgeschlossen (ProposalWriter existiert) - Q-MH-7 entschieden (Drift-Schwellen für Engine-Berechnung) - Q-MH-8 entschieden (Volatility-Kill-Threshold-Wert) - Q-MH-9 entschieden (Confidence-Threshold-Wert)

Implementation: 1. Engine-Skelett mit dataclasses (ProposalResult, MarketContext, RiskRecommendation, Confidence) 2. Mock-Exchange-Tests für jede Pipeline-Stufe 3. Real-Exchange-Tests in TESTNET (manuell verified) 4. Boundary-AST-Tests 5. Versioning-Tests 6. Dry-Run-Tests

Lieferzustand: Engine ist callable, schreibt Proposals, aber **kein** Wiring in CommandBus (das ist MH-6).

→ kein Bot-Restart in dieser Phase. Engine steht standalone bereit für MH-6.

— Ende Risk Proposal Engine —

📄 `03_state_machine.md`

03 · State Machine — 10 Zustände + Übergänge + Guards + Rollback

Status: architektur-defined **Phase-Key:** verteilt über RECON-MH-2..7 (Reader-Schema, Worker-Handler, GUI-Wiring) **Bezug:** zentrale Zustandsmaschine für Managed-Holdings-Lifecycle

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `01_foundation.md`, `02_risk_proposal_engine.md`
Required previous phases	RECON-MH-2 ☐ (Reader für State-Lookup)
Restart required	no (Schema-only); ja in MH-7 (Wiring in main.py)
Migration required	no
Mainnet-touch	no
Risk-Level	high — Kern-Zustandsmodell, fehlerhafte Übergänge brechen Lifecycle

Allowed

- 10 fix definierte States (siehe §1) als Schema-Konstanten
- Übergangs-Tabelle mit Actor-Spalte (user_*/bot/system)
- Guards (Validatoren pro Übergang) als Bot-Worker + PHP-Service-Layer Code
- Drift-Detection als per-cycle Hook in `main.py` (Phase MH-7)
- TTL-Expiry als scheduled Job (Phase MH-9)
- Cooldown-Logik nach release/exit (Phase MH-7)
- Rollback-Pfade pro Übergang dokumentieren
- Audit-Event pro Übergang (G-DR-5)
- DR-11 Enforcement im Worker

NOT Allowed

- neue States ohne Vertrag-Update (`99_master_boundaries.md` v-bumpen)
- Bot-Worker autonom `managed_*` → `frozen` (G-DR-9)
- silent state mutation ohne audit
- Übergangs ohne Guard
- direct `managed_state.json` mutation außerhalb Worker
- terminal-States `exit_executed` ohne Operator-Action zu `frozen` zurück (G-DR-2)
- `proposal_pending` Übergang OHNE TTL (MUSS expiry haben)

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-2	Operator final authority bei state-transitions	Worker-Validator: managed→frozen requires actor=user_*
G-DR-9	Bot darf nicht autonom managed→frozen	DR-11 enforcement Test
G-DR-11	audit_event-Prefix `managed.*` strict	Boundary-Test
SM-1	Drift-Freeze	autonomer Übergang `managed_active → managed_drift_alert` (kein Sell!)
SM-2	Cooldown nach release/exit	system-managed lock auf re-propose innerhalb X Stunden
SM-3	Proposal-TTL	scheduled job → `proposal_pending|risk_proposed → proposal_rejected`
SM-5	Volatility-Kill-Switch	autonomer Übergang `managed_active → managed_paused` (kein Sell!)

Expected Test Surface

- ~25 Bot-Tests
- 10 Übergangs-Tests pro Pfad (jeder Übergang: ok-path + invalid-actor-reject)
- 5 DR-11-Enforcement: Bot-actor wird abgelehnt
- 5 TTL-Expiry: scheduled job markiert expired Proposals
- 5 Drift-Detection: Trigger schreibt managed_drift_alert + audit
- ~15 PHP-Tests
- Service-Layer-Validators für approve/reject/pause/resume/release
- Idempotency: zweiter approve auf decided proposal → precondition_rejected
- bestehende Suites NICHT betroffen

1 · Die 10 Zustände

State	Bedeutung	Wer setzt?	Terminal?
`frozen`	Default; Bot ignoriert	Operator / System / Default	Nein (Re-Entry möglich)
`proposal_pending`	Bot generiert Vorschlag	Operator triggert	Nein
`risk_proposed`	Vorschlag liegt vor, Operator-Action erwartet	Bot finalisiert	Nein
`proposal_aborted`	Bot konnte nicht generieren	Bot (kein Balance / Markt down / blacklisted)	Ja → frozen
`proposal_rejected`	Operator declined oder TTL expired	Operator / System	Ja → frozen
`managed_active`	Bot trackt + tradet Asset	Operator confirmed	Nein
`managed_paused`	Operator-pausiert (oder SM-5 Volatility-Kill)	Operator / **Bot (SM-5)**	Nein
`managed_drift_alert`	Bot detected Drift	**Bot (SM-1)**	Nein — Operator entscheidet
`release_pending`	Graceful exit angefordert	Operator	Nein
`exit_executed`	SL/TP gehit	Bot (per Proposal-SL/TP)	Ja — Operator entscheidet ob neu propose'n oder bleibend frozen

2 · Übergangs-Matrix (vollständig)

From	To	Actor	Trigger	Audit-Event	Guard
frozen	proposal_pending	`user_*`	Operator-Klick „Analyse"	`managed.asset_proposal_requested`	wallet qty > 0; baseline-aware
proposal_pending	risk_proposed	`bot`	Engine fertig	`managed.asset_proposal_generated`	proposal_version + risk_model_version + confidence Pflicht
proposal_pending	proposal_aborted	`bot`	Engine abort	`managed.asset_proposal_aborted`	aborted_reason ∈ enum
risk_proposed	managed_active	`user_*`	Operator-Approve	`managed.asset_promoted` + `managed.synthetic_entry_set`	Hard-Confirm `<asset>:<variant>`; confidence_override-flag falls Score < threshold
risk_proposed	proposal_rejected	`user_*`	Operator-Reject	`managed.asset_proposal_rejected`	none (Reject ist immer erlaubt)
risk_proposed	proposal_rejected	`system`	TTL-Expiry-Job	`managed.asset_proposal_expired`	proposal.expires_at < now
proposal_aborted	frozen	`system`	Auto-Cleanup	–	–
proposal_rejected	frozen	`system`	Auto-Cleanup nach Cooldown	–	Cooldown abgelaufen (SM-2)
managed_active	managed_paused	`user_*`	Operator-Pause	`managed.asset_paused`	none
managed_active	managed_paused	`bot`	Volatility-Kill (SM-5)	`managed.asset_paused` mit reason='volatility_kill_switch'	volatility_30d > threshold
managed_paused	managed_active	`user_*`	Operator-Resume	`managed.asset_resumed`	none
managed_paused	release_pending	`user_*`	Operator-Release	`managed.asset_release_requested`	none
managed_active	managed_drift_alert	`bot`	Drift-Detection (SM-1)	`managed.drift_detected` mit `drift_kind`	qty/price/regime drift threshold gerissen
managed_drift_alert	managed_active	`user_*`	Operator-Override	`managed.operator_override`	Hard-Confirm
managed_drift_alert	release_pending	`user_*`	Operator-Release	`managed.asset_release_requested`	none
managed_active	exit_executed	`bot`	SL oder TP gehit	`managed.exit_executed`	position SELL via update_prices
managed_active	release_pending	`user_*`	Operator-Release	`managed.asset_release_requested`	none
release_pending	frozen	`system`	position closed (Bot SELL oder Operator manual)	`managed.asset_released`	qty in position == 0
exit_executed	frozen	`user_*`	Operator decide	`managed.asset_released`	none — explizite Operator-Action
exit_executed	proposal_pending	`user_*`	Operator re-	`managed.asset_proposal_requested`	Cooldown abgelaufen

From	To	Actor	Trigger	Audit-Event	Guard
			engage		

Hard-Rule: für jede `managed_* → frozen`-Transition MUSS `actor` ein `user_*` oder `system` sein. **NIE** `bot`. (G-DR-9, G-SR-4)

Two-File-Atomic-Hard-Rule (G-DR-14, decided 2026-05-10): Übergänge die `frozen ↔ managed` involvieren (insbesondere `risk_proposed → managed_active` und `managed_*/release_pending → frozen`) schreiben `managed_state.json` **UND** `baseline_holdings.json` **transactional in einer DB-Transaction**. Siehe `99_master_boundaries.md §14.2`. Bei einem File-Failure: Backup-Restore beider Files + audit `managed.two_file_atomic_rollback`.

Hard-Lock G-DR-14: `tradable_quote ↔ frozen/managed` ist verboten. Transitions die diese Grenze kreuzen würden, werden vom Worker-Guard reject'd mit audit `managed.policy_change_blocked_dr14`.

---

3 · Drift-Detection (SM-1)

Per-cycle Hook in `main.py`:

```
# Phase MH-7
def detect_managed_drift(trader, baseline_reader, managed_reader):
    snap = managed_reader.snapshot()
    if snap is None:
        return  # nothing to check
    for asset, slot in snap.managed_assets.items():
        if slot["state"] != "managed_active":
            continue
        symbol = f"{asset}/USDT"
        # qty drift
        real_qty = exchange.fetch_balance().get(asset, {}).get("total", 0)
        state_qty = trader.state["positions"].get(symbol, {}).get("quantity", 0)
        if abs(real_qty - state_qty) / max(real_qty, 0.0001) > 0.05:
            emit_drift(asset, drift_kind="qty", real_qty=real_qty, state_qty=state_qty)
            continue
        # price drift
        current_price = exchange.fetch_ticker(symbol)["last"]
        synth_entry = slot["synthetic_entry"]
        atr = ...  # cached or recomputed
        if abs(current_price - synth_entry) > 2 * atr:
            emit_drift(asset, drift_kind="price", current=current_price, entry=synth_entry, atr=atr)
            continue
        # regime drift
        if regime_changed_from_promote_time(slot["promoted_at"]):
            emit_drift(asset, drift_kind="regime")
```

Konsequenz: Worker schreibt managed_state.state = `managed_drift_alert` + audit. **KEIN Bot-Sell** (G-SR-7).

---

4 · Cooldown nach release/exit (SM-2)

Schwelle (Q-MH-12 entscheidet): - Default-Vorschlag: 24h auf TESTNET, 7 Tage auf Mainnet (Mainnet später)

Mechanismus: - nach `managed.asset_released` oder `managed.exit_executed`: System schreibt in managed_state.json `released_assets[asset].last_release_ts` - bei `request_managed_proposal` für gleiches Asset: Worker prüft `now - last_release_ts < cooldown` → reject mit `cooldown_active` - Audit-Event `managed.asset_proposal_request_blocked_by_cooldown`

Operator-Override: - per Q-MH-16 entscheiden: darf Operator Cooldown override'n? (Vorschlag: ja, mit extra Hard-Confirm)

---

5 · Proposal-TTL (SM-3)

Default-TTL: 7 Tage (Q-MH-1).

Implementierung: - jeder `risk_proposals/<id>.json` enthält `expires_at` - scheduled job (Phase MH-9) läuft 1× täglich - für jedes proposal mit `state ∈ {proposal_pending, risk_proposed}` und `expires_at < now`: - state → `proposal_rejected` - audit `managed.asset_proposal_expired` - Operator-Notification (Telegram-Channel optional)

Race-Vermeidung: - TTL-Job schreibt mit row-level lock auf managed_assets[asset] - wenn parallel Operator approved während TTL-Job läuft: - approve schreibt zuerst → state = managed_active - TTL-Job sieht state != risk_proposed → skip

---

6 · Volatility-Kill-Switch (SM-5)

Trigger im managed_active-Lebenszyklus: - per-cycle: berechne 30d-volatility für jedes managed_active Asset - wenn `volatility_30d_pct >= kill_threshold` (Default 25%, Q-MH-18): - state → `managed_paused` mit reason `volatility_kill_switch` - audit `managed.asset_paused` mit metadata.reason - **KEIN Sell** (Position bleibt offen, nur kein neuer Trade) - Operator-Notification

Operator-Optionen nach Volatility-Kill: - `managed_paused → managed_active` resume (overrides Volatility-Kill, neue Volatility-Berechnung muss < threshold sein) - `managed_paused → release_pending` graceful exit

---

7 · Guards (Validatoren pro Übergang)

Bot-Worker-Side Validators (Phase MH-6):

```
class ManagedStateGuard:
    @staticmethod
    def can_transition(from_state, to_state, actor, payload) -> Result:
        # 1. State-existence
        if from_state not in ALLOWED_STATES or to_state not in ALLOWED_STATES:
            return Reject("invalid_state")
        # 2. Transition-existence
        if (from_state, to_state) not in TRANSITION_MATRIX:
            return Reject("transition_not_allowed")
        # 3. Actor-Match
        required_actor_types = TRANSITION_MATRIX[(from_state, to_state)]["allowed_actors"]
        if actor.type not in required_actor_types:
            return Reject(f"actor_not_allowed_for_transition:{actor.type}")
        # 4. DR-11: Bot autonom managed→frozen ist HARD-blocked
        if from_state.startswith("managed_") and to_state == "frozen":
            if actor.type != "user" and actor.type != "system":
                return Reject("dr11_bot_cannot_demote_managed_to_frozen")
        # 5. Phase-spezifische Guards
        if to_state == "managed_active":
            # Confidence-Override-Flag check (SM-4)
            if payload.confidence_score < CONFIDENCE_THRESHOLD and not payload.operator_override_confirmed:
                return Reject("confidence_below_threshold_no_override")
        # 6. Cooldown check (SM-2)
        if from_state == "frozen" and to_state == "proposal_pending":
            if last_release_ts and (now - last_release_ts) < cooldown:
                return Reject("cooldown_active")
        return Accept()
```

Source-Grep-Test: alle Übergänge laufen durch `ManagedStateGuard.can_transition` — kein direkter state-write außerhalb.

---

8 · Rollback-Pfade

Übergang	Rollback-Pfad
frozen → proposal_pending	Operator klickt „Cancel Proposal Request" → Worker setzt zurück frozen + audit
proposal_pending → risk_proposed	kein Rollback (Bot-Output ist immutable)
risk_proposed → managed_active	`release_managed_asset` → release_pending → frozen
managed_active → managed_paused	`resume_managed_asset` zurück
managed_paused → managed_active (resume)	`pause_managed_asset` zurück
managed_active → exit_executed	**kein Rollback** — Sell ist real, Wallet hat sich geändert
exit_executed → frozen	trivial — Operator-decide
any → release_pending	kein Rollback — graceful exit ist gewählt

Snapshot-basierter State-Rollback (BACKLOG, separate Phase): - pro Übergang Snapshot-File (SM-6, siehe `05_commandbus_worker.md`) - Operator kann via separater Phase „restore_managed_state_snapshot" managed_state.json auf Snapshot zurücksetzen - ABER: Wallet-Realität ist NICHT zurücksetzbar (echte Trades irreversible) - Snapshots-Restore nur für State-Files, nicht für Position-State

---

9 · UML-State-Diagramm (Mermaid)

```
stateDiagram-v2
    [*] --> frozen
    frozen --> proposal_pending: operator: analyze
    proposal_pending --> risk_proposed: bot: generated
    proposal_pending --> proposal_aborted: bot: abort
    proposal_aborted --> frozen
    risk_proposed --> managed_active: operator: approve
    risk_proposed --> proposal_rejected: operator: reject
    risk_proposed --> proposal_rejected: system: ttl_expiry
    proposal_rejected --> frozen: system: cooldown_done
    managed_active --> managed_paused: operator: pause
    managed_active --> managed_paused: bot: volatility_kill
    managed_paused --> managed_active: operator: resume
    managed_paused --> release_pending: operator: release
    managed_active --> managed_drift_alert: bot: drift_detected
    managed_drift_alert --> managed_active: operator: override
    managed_drift_alert --> release_pending: operator: release
    managed_active --> release_pending: operator: release
    managed_active --> exit_executed: bot: sl_or_tp_hit
    release_pending --> frozen: system: position_closed
    exit_executed --> frozen: operator: decide
    exit_executed --> proposal_pending: operator: re_engage
```

---

10 · Phase-Sequenz für 03_state_machine

Voraussetzungen: - MH-2 abgeschlossen (Reader für State-Lookup) - Q-MH-7 entschieden (Drift-Schwellen) - Q-MH-1 entschieden (Proposal-TTL Wert) - Q-MH-12 entschieden (Cooldown-Schwelle)

Implementation-Ort: - Bot-Side: `managed_state_writer.py` enthält Guards (MH-2 erweitert in MH-7) - Bot-Side: `command_worker.py` Handler nutzen Guards (MH-6) - PHP-Side: `ManagedStateService` enthält Pre-Validation (MH-4) - main.py per-cycle hook für Drift-Detection (MH-7) + Volatility-Kill (MH-7)

Bot-Restart-Notwendigkeit: - MH-7 erfordert Restart, weil main.py erweitert wird - alle anderen Phasen (Reader/Writer/Service/UI) sind restart-frei

---

— Ende State Machine —

📄 `04_gui_operator_flow.md`

04 · GUI Operator Flow — Multi-Step Wizard, Filament Pages, UX-Regeln

Status: architektur-defined **Phase-Key:** RECON-MH-5 **Bezug:** Filament v4 UI für Managed-Holdings-Lifecycle

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `01_foundation.md`, `02_risk_proposal_engine.md`, `03_state_machine.md`, `05_commandbus_worker.md`
Required previous phases	RECON-MH-1 ☐ (Registry), RECON-MH-4 ☐ (Services + DB-Cache)
Restart required	no
Migration required	no (DB-Cache-Tabellen kommen aus MH-4)
Mainnet-touch	no
Risk-Level	medium — Operator-UX entscheidet ob System sicher bedienbar ist

Allowed

- neue Filament v4 Page `ManagedHoldings` (admin-only canAccess)
- neue Filament Resource `ManagedProposalResource` (Detail-View pro Proposal)
- neue Blade-Components für policy-Badge, wallet-signature-Chip, etc. (analog Iter-2 Design)
- Multi-Step Wizard via Filament Form Steps
- Hard-Confirm-Forms mit dynamic-string-validation
- Polling 30s für Live-State (analog `RuntimeConfigStatusWidget`)
- Service-Layer-Aufrufe über Dependency Injection
- audit_events-Read für Timeline (analog G10-6 Pattern)
- read-only Charts für Risk-Summary (Exposure-Pie, Confidence-Bar)

NOT Allowed

- direct file-write (`managed_state.json`, `risk_proposals/*.json`, etc.) — G-DR-3
- direct ccxt-Calls aus PHP
- direkte Bot-Process-Kontrolle
- Operator-Role-Sichtbarkeit für admin-only Actions
- Auto-Submit ohne Hard-Confirm
- One-Click-Promote (UX-1 verbietet das)
- Wizard-Skip (jeder Step muss durchlaufen sein bevor Submit)
- Session-State zwischen Wizard-Steps verlieren (UX-Frust)
- Mainnet-Banner suggerieren das Mainnet near-future-aktivierbar wäre

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-2	Operator final authority	jede Mutation läuft durch Hard-Confirm-Form
G-DR-7	Multi-Step Wizard	mindestens 5 Steps, Single-Modal verboten
G-DR-13	GUI ist read-mostly + Action-Trigger	INSERT in commands + audit_events, kein File-Write
UX-1	NIE One-Click-Transfer	Wizard mit min. 5 Steps + Hard-Confirm
UX-2	Multi-Step Wizard	Filament Form Steps
UX-3	Risk-Summary visuell hervorheben	dedizierter Step mit Confidence + Warnings
UX-4	Exposure-Impact visualisieren	Pie + Concentration-Warning vor Submit
UX-5	Mainnet immer markieren	sticky Mainnet-Banner permanent
Hard-Confirm-Patterns	siehe `iteration_3_briefing.md` §6.6	dynamische String-Validation

Expected Test Surface

- ~30 PHP-Tests
- 5 Page admin-only canAccess
- 5 Wizard step navigation (no skip, all steps required)
- 5 Hard-Confirm-String dynamic validation
- 5 Polling-Refresh + Stale-Threshold
- 5 Audit-Trail-Read (audit_events filtered for managed.*)
- 5 Boundary-AST: keine direct File-Mutation, keine ccxt-Calls in Filament-Code
- ~10 Browser-Tests (Filament test helpers)
- End-to-End Wizard Flow für Approve, Reject, Pause, Resume, Release
- bestehende Suites NICHT betroffen (additiv)

---

1 · Page-Inventar

Filament Surface	Inhalt
`Filament\Pages\ManagedHoldings`	Live-Lifecycle-Tabelle (alle Wallet-Assets + State + Aktion)
`Filament\Resources\ManagedProposalResource`	Detail-View pro Proposal mit Approve/Reject Action
`Filament\Widgets\ManagedHoldingsKpiWidget`	KPIs: total managed, total frozen, drift_alerts pending, proposals expired-soon
`Filament\Widgets\ManagedAuditTimelineWidget`	letzte 50 managed.*-Events im rechten Panel

Alle admin-only via `canAccess()` (analog `BaselineHoldings::canAccess`).

---

2 · Multi-Step Wizard für Promote (UX-2 zentral)

5-Step Wizard für `request_managed_proposal` → `approve_managed_proposal`:

```
Step 1: Asset & Intent
  – Asset-Symbol + aktueller Wallet-qty (read-only)
  – Operator schreibt Begründung (Pflicht-Textfield, ≥ 20 chars)
  – Trigger button: "Bot-Analyse anfordern"
  – Service-Call: createRequestManagedProposalCommand

[Bot-Engine läuft asynchron via Worker]

Step 2: Bot-Proposal Review
  – Markt-Kontext-Card (Preis, ATR, Volatility, Volumen, Regime)
  – Strategy-Empfehlung-Card (group + rationale)
  – Confidence-Score-Pill (grün/gelb/rot mit Threshold-Anzeige)
  – Context-Warnings (rot wenn problematisch – concentration, peg, regime, blacklist)
  – if confidence < threshold: "ACHTUNG – Bot empfiehlt nicht" Banner

Step 3: Variant Selection + Operator Override
  – Variant-Selector env-abhängig (Q-MH-2 decided 2026-05-10):
      Testnet:  3 Karten (conservative / recommended / aggressive)
      Mainnet:  2 Karten (conservative / recommended) – aggressive ausgeblendet (Q-MH-13)
  – pro Variante: SL / TP / RR / max_alloc Anzeige
  – Custom-Override-Form falls Operator gewählte Variante anpassen will
      Override erlaubt (Q-MH-3): SL/TP/max_alloc/trailing_stop
      Override verboten (Q-MH-13): strategy_group, variant-Switch
  – Hint auf Mainnet: "aggressive disabled (Mainnet-Policy)"

Step 4: Risk-Summary + Exposure-Impact (UX-3 + UX-4)
  – Pie-Chart aktuelle Portfolio-Allokation
  – Konfetti-Pie nach Promote (vergleicht "vorher" vs "nachher")
  – Concentration-Warning falls neue Position > 20% des Portfolios
  – Correlation-Liste (z.B. "starke Korrelation 0.83 mit BTC/USDT")
  – ggf. Volatility-Kill-Switch-Hard-Stop
  – Final-Bestätigung-Button enabled NUR wenn alle Warnings akzeptiert (Checkboxen)

Step 5: Hard-Confirm
  – Hard-Confirm-Input mit dynamischem String <asset>:<variant>
  – z.B. "SOL:recommended"
  – Submit disabled bis exact match
  – falls confidence < threshold: zusätzliches Override-Confirm-Field "I-OVERRIDE-BOT"
  – Submit triggert createApproveManagedProposalCommand
```

Filament v4 Implementation:

```php
public function form(Form $form): Form
{
    return $form->schema([
        Wizard::make([
            Wizard\Step::make('Asset & Intent')
                ->schema([...])
                ->beforeValidation(fn() => $this->triggerProposalRequest()),
            Wizard\Step::make('Bot-Proposal Review')
                ->schema([...]),
            Wizard\Step::make('Variant Selection')
                ->schema([...]),
            Wizard\Step::make('Risk-Summary')
                ->schema([...]),
            Wizard\Step::make('Hard-Confirm')
                ->schema([...]),
        ])
        ->skippable(false)  // UX-1 enforcement
        ->submitAction(...)
    ]);
}
```

---

3 · Approval-Pfade (kompakt)

Action	Trigger	Worker-Pfad	Audit-Event
Approve	Wizard Step 5 Submit	`approve_managed_proposal` Command	`managed.asset_promoted`
Reject	Operator clicks Reject im Step 2-4	`reject_managed_proposal` Command	`managed.asset_proposal_rejected`
Defer	Operator schließt Wizard ohne Submit	kein Command	nur audit `managed.asset_proposal_deferred` (optional)
Pause	Operator-Action auf managed_active Asset	`pause_managed_asset` Command	`managed.asset_paused`
Resume	Operator-Action auf managed_paused Asset	`resume_managed_asset` Command	`managed.asset_resumed`
Release	Operator-Action auf managed_*/drift_alert	`release_managed_asset` Command	`managed.asset_release_requested`
Override Drift	Operator auf managed_drift_alert	`approve_managed_proposal` (re-confirm)	`managed.operator_override`

4 · UX-Regeln (verbindlich, Reviewer SM)

UX-1 · NIE One-Click-Transfer

- Wizard hat mindestens 5 Steps
- jeder Step braucht entweder Operator-Input ODER explizite Bestätigung
- skippable=false (Filament v4)
- jeder Step zeigt Progress-Indicator

UX-2 · Multi-Step Wizard

- siehe §2 — 5 Steps verbindlich
- pro Step: Header + Body + Next/Back-Button
- Back-Button bricht NICHT Proposal-Request ab (proposal bleibt im commands-Tabelle)

UX-3 · Risk-Summary hervorheben

- Step 4 hat mindestens 4 visuelle Sub-Cards:
- Confidence-Score (Bar mit Threshold-Markierung)
- Risk-Recommendation Tabelle (SL/TP/RR/max_alloc)
- Warnings-Panel (rot wenn ≥ 1 Warning)
- Exposure-Pie (Portfolio vorher/nachher)
- alle Warnings sind dismissable nur durch Checkbox-Akzeptanz
- Final-Submit-Button disabled bis alle Warnings akzeptiert

UX-4 · Exposure-Impact visualisieren

- Pie-Chart der aktuellen Portfolio-Allokation in USDT-Werten
- nach Promote: simulierter neuer Anteil
- wenn neuer Anteil > 20% des Portfolios → Concentration-Warning
- wenn neuer Anteil > 5 ähnliche Korrelation-Assets → Concentration-Cluster-Warning
- Operator sieht visuell: "verlässt du dein Portfolio-Konzept?"

UX-5 · Mainnet immer markieren

- sticky Mainnet-Banner top permanent
- Text exakt: `"TESTNET · Mainnet durably blocked · BINANCE_TESTNET=true"`
- ROT/danger color
- nicht dismissable
- alle Action-Buttons in der Page haben zusätzlich `mode-badge` „TESTNET"

5 · Filament Components

Custom Blade Components (vom Designer Iter-3 v3):

Component	Verwendung
`<x-stb.policy-badge :policy="..." />`	Asset-Tabelle, Wizard, Audit
`<x-stb.wsig :state="..." />`	Page-Header
`<x-stb.cb :state="..." />`	KPI-Widget, falls B-OUTAGE-Status angezeigt
`<x-stb.proposal-card :proposal="..." />`	Wizard Step 2
`<x-stb.confidence-bar :score="..." :threshold="..." />`	Wizard Step 4
`<x-stb.exposure-pie :before="..." :after="..." />`	Wizard Step 4
`<x-stb.audit-row :event="..." />`	Audit-Timeline
`<x-stb.hard-confirm :expected="..." />`	Wizard Step 5 + alle anderen Hard-Confirms

6 · Hard-Confirm-Patterns (zwei!)

Aus `iteration_3_briefing.md` §6.6:

Action	Pattern	Beispiel
approve_managed_proposal	`<asset>:<variant>`	`SOL:recommended`
approve_managed_proposal (Override-Path)	`<asset>:override:<variant>`	`SOL:override:aggressive`
reject_managed_proposal	statisch `reject`	`reject`
pause_managed_asset	statisch `pause-<asset>`	`pause-SOL`
resume_managed_asset	statisch `resume-<asset>`	`resume-SOL`
release_managed_asset	statisch `release-<asset>`	`release-SOL`

Q-MH-11 Decision (2026-05-10): approve-Pattern ist `<asset>:<variant>` mit dynamischer Variant-Auswahl pro Klick. Override-Pattern `<asset>:override:<variant>` macht Override-Audit eindeutig nachvollziehbar. Pattern bricht Muskel-Memory + erzwingt bewusste Variant-Wahl.

Implementation: Filament TextInput rule mit closure die den expected String dynamisch berechnet (analog G10-6 `confirm_profile_name`-Pattern).

7 · Audit-Timeline-Section

ManagedHoldings Page hat im rechten Panel:

```
Letzte 50 managed.*-Events:
─────────────────────────────────────────────────────────
2026-05-10 14:33:41  managed.asset_proposal_requested  user_42  SOL  command_id 8a3f...
2026-05-10 14:34:12  managed.asset_proposal_generated  bot      SOL  proposal_id 7c2e...
2026-05-10 14:35:50  managed.asset_promoted            user_42  SOL  variant=recommended
[expand metadata for each row]
```

Implementation: - Filament `TableWidget` mit polling 30s - Filter: `event_type LIKE 'managed.%'` - Sortierung: `created_at DESC` - Limit: 50 - Row-click expand zeigt `metadata` als formatted-JSON-Block

8 · Live-State-Tabelle (Hauptscreen)

```
ManagedHoldings Tabelle:
| Asset | Wallet-Qty | Policy-Badge | State | Action |
| BTC   | 1.0        | frozen       | frozen           | [Analyse] |
| SOL   | 5.0        | managed      | managed_active   | [Pause] [Release] |
| DOGE  | 12.5       | frozen       | proposal_pending | (Spinner) |
| ETH   | 1.0        | frozen       | risk_proposed    | [Review →] |
| SHIB  | 1M         | frozen       | managed_drift_alert | [Override] [Release] |
```

- Filter: nach Policy + nach State
- Sort: nach Asset, nach State, nach last_change
- Pro Row: action-buttons je nach state
- live-Polling 30s

---

9 · Operator-Notifications

innerhalb GUI: - Filament Notification (toast) bei jeder Command-Successful-Insertion - Notification bei drift_detected events (red, sticky) - Notification bei TTL-expiry (yellow)

außerhalb GUI (optional, Phase MH-9): - Telegram-Bot-Channel über Reporter-Token - Email-Notification bei `managed.drift_detected` + `managed.exit_executed` - KEINE Notification über GUI bei Promote/Pause/Resume — Operator hat bereits geklickt

---

10 · Mobile-Behandlung

Out-of-Scope für RECON-MH-5: - Mobile-Operator-Use ist BACKLOG (siehe `iteration_3_briefing.md` §10) - ManagedHoldings Page ist Desktop-only - Wizard funktioniert auf Mobile aber nicht Mobile-optimiert (Touch-Targets ≥44px in späterer Phase)

---

11 · Phasen-Sequenz für 04_gui_operator_flow

Voraussetzungen: - MH-4 abgeschlossen (Services + DB-Cache-Tabellen existieren) - Q-MH-2 entschieden (3 oder 2 Varianten?) - Q-MH-11 entschieden (Hard-Confirm-Pattern verbindlich) - Q-MH-3 entschieden (Operator-Override-Tiefe) - Designer Iter-3 v3 abgeliefert (siehe `iteration_3_briefing.md`)

Implementation: 1. ManagedHoldings Page Skelett + canAccess + nav-icon 2. Live-State-Tabelle mit Polling 3. Wizard mit 5 Steps (skippable=false) 4. ManagedProposalResource + Detail-View 5. Audit-Timeline Widget 6. Tests (PHP + Browser-tests)

Lieferzustand: GUI komplett funktional, aber Worker-Daemon fehlt → Pause/Resume haben Latenz (siehe Q-MH-14 / `05_commandbus_worker.md`).

→ kein Bot-Restart in dieser Phase.

— Ende GUI Operator Flow —

📄 `05_commandbus_worker.md`

05 · CommandBus & Worker — 8 Command-Types, Audit-Snapshots, Restart-Strategie

Status: architektur-defined **Phase-Key:** RECON-MH-1 (Registry) + RECON-MH-4 (Services) + RECON-MH-6 (Worker-Handler) + RECON-MH-9 (Worker-Daemon) **Bezug:** Datenfluss Operator → CommandBus → Worker → State-Files

---

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `01_foundation.md`, `02_risk_proposal_engine.md`, `03_state_machine.md`
Required previous phases	RECON-MH-2 ☐ (Reader), RECON-MH-3 ☐ (Engine)
Restart required	yes (in MH-7 wenn Wiring) — die reine Worker-Erweiterung in MH-6 ist restart-frei (Worker-Code wird beim nächsten `--once` neu geladen)
Migration required	yes (DB-Cache-Tabellen aus MH-4)
Mainnet-touch	no
Risk-Level	high — zentrale Mutation-Pipeline

Allowed

- 8 neue CommandTypeRegistry Einträge (siehe §2)
- Worker-Handler in `command_worker.py` für jeden Type
- Service-Layer-Klassen (`ProposalService`, `ManagedStateService`) als Command-Builder
- Audit-Snapshot-Files (SM-6) — immutable JSON pro State-Übergang
- Idempotency-Keys für approve/reject (G6.5 Pattern)
- Worker `--once` invocation (manuell, durch Operator)
- Worker-Daemon-Aktivierung (MH-9, separate Phase)
- DB-Cache-Updates (managed_proposals, managed_assets_history)

NOT Allowed

- direct file-write außerhalb Worker-Handler (G-DR-3)
- GUI-Service-Layer mutiert state-files direkt (G-DR-3)
- Worker spawned eigenständig Commands (außer `flag_managed_drift` für Bot-Self-Detection)
- Migration ohne dedizierte Phase
- Audit-Snapshot überschreiben oder löschen (immutable)
- Idempotency-Key ohne `proposal_id` für approve/reject
- Mainnet-Apply (5-Layer-Block, G-SR-11)
- Worker-Daemon-Aktivierung außerhalb MH-9
- Bot-Worker autonom managed→frozen (G-DR-9)
- Schreibe-Operation ohne `command_id` in audit-metadata

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-3	CommandBus-only Mutation	jeder Writer-Call kommt aus Worker, nie aus PHP
G-DR-5	Audit-heavy	jede Mutation schreibt audit_event + audit_snapshot
G-DR-6	Backup-before-mutate	Writer-Pattern (RECON-2.1)
G-DR-9	Bot autonom managed→frozen verboten	Worker-Validator
G-DR-11	audit-Prefix `managed.*` strict	Boundary-Test
G-SR-1	außerhalb-CommandBus-Mutation	filewatcher / source-grep
G-SR-3	Promote ohne user-actor → STOP	Worker-Validator-DR-11
G-SR-11	Mainnet-Block 5-Layer	env=testnet hard, jede Schicht prüft
SM-6	Audit-Snapshot-Files	Worker schreibt nach jedem State-Change
SM-7	Dry-Run-Promote	eigener Command-Type `dry_run_promote_managed`

Expected Test Surface

- ~50 Bot-Tests (Worker-Handler, je 6-7 pro Type)
- ~30 PHP-Tests (Service-Builder + Idempotency + Validator)
- ~10 DB-Migration-Tests (up/down/Backfill)
- ~5 Boundary-AST-Tests
- ~5 Audit-Snapshot-Tests (immutable, nicht überschreibbar)
- bestehende Suites NICHT betroffen (additiv)

---

1 · Datenfluss-Pipeline

```
Operator-Klick (GUI)
  │
  ▼
Filament Action (admin-only)
  │
  ▼
Service-Layer (PHP, e.g. ProposalService)
  │ → INSERT in `commands` (status=pending)
  │ → INSERT in `audit_events` (managed.*_requested)
  │ → INSERT in `managed_proposals` (DB-Cache, optional)
  ▼
Worker --once (manuell oder Daemon)
  │
  ▼
claim_next() (FOR UPDATE SKIP LOCKED)
  │ → UPDATE commands SET status='claimed'
  │ → INSERT command_audit_log (claimed)
  ▼
_handle_<command_type>() (Bot-Worker)
  │ → ManagedStateGuard.can_transition()
  │ → ManagedStateWriter.apply() / ProposalWriter.apply() / ProposalReader.get()
  │ → audit_event managed.* (succeeded/failed)
  │ → audit_snapshot file (SM-6)
  │ → DB-Cache update (managed_proposals.state, managed_assets_history insert)
  ▼
mark_succeeded(commands.result_json)
```

---

2 · Die 8 Command-Types

Command-Type	Trigger	Worker-Handler	Idempotency-Key
`request_managed_proposal`	Operator Wizard Step 1 Submit	`_handle_request_managed_proposal`	`mh:propose:<asset>:<YmdHis>`
`approve_managed_proposal`	Operator Wizard Step 5 Submit	`_handle_approve_managed_proposal`	`mh:decide:<proposal_id>`
`reject_managed_proposal`	Operator Reject-Action	`_handle_reject_managed_proposal`	`mh:decide:<proposal_id>`
`pause_managed_asset`	Operator Pause-Action	`_handle_pause_managed_asset`	`mh:pause:<asset>:<YmdHis>`
`resume_managed_asset`	Operator Resume-Action	`_handle_resume_managed_asset`	`mh:resume:<asset>:<YmdHis>`
`release_managed_asset`	Operator Release-Action	`_handle_release_managed_asset`	`mh:release:<asset>:<YmdHis>`
`flag_managed_drift`	**Bot self-emit** via internal channel	`_handle_flag_managed_drift`	`mh:drift:<asset>:<YmdHis>`
`dry_run_promote_managed`	Operator-Klick „Dry-Run im Wizard" (SM-7)	`_handle_dry_run_promote_managed`	`mh:dryrun:<asset>:<YmdHis>`

Alle: - admin-only (außer evtl. `request_managed_proposal` für operator-Role mit Q-MH-3-Antwort) - testnet-only (`allowedEnvironments: ['testnet']`) - Mainnet hard-blocked an 5 Layern

Idempotency-Pattern: - `approve` und `reject` teilen denselben Key → das verhindert dass beides nacheinander auf dasselbe Proposal läuft - alle anderen haben distinct Keys pro Klick (kein dedup, mehrere pause/resume möglich)

---

3 · Worker-Handler-Skelett (analog RECON-2.3c Pattern)

```
def _handle_request_managed_proposal(self, cmd: dict) -> dict:
    cid = cmd["command_id"]
    payload = cmd["payload_json"]
    asset = payload["asset"]

    self._g10_3b_emit_audit(cid, "managed.asset_proposal_requested_started", {...})
    try:
        # 1. Validate payload (env=testnet, asset format, no cooldown)
        # 2. Load BaselineHoldingsReader; verify asset is currently 'frozen'
        # 3. Check Cooldown (SM-2): last_release_ts of this asset
        # 4. Trigger proposal_engine.generate(asset, dry_run=False)
        # 5. ManagedStateWriter: pending_proposals[asset] = proposal_pending
        # 6. Audit: managed.asset_proposal_generated (oder _aborted bei abort)
        # 7. Audit-Snapshot file: audit_snapshots/<event_id>.json
        # 8. DB-Cache: INSERT INTO managed_proposals
        return {"ok": True, "proposal_id": ..., "state": "risk_proposed"}
    except Exception as exc:
        self._g10_3b_emit_audit(cid, "managed.asset_proposal_failed", {...})
        raise
```

Pattern für alle 8 Handler: 1. emit `*_started` audit 2. validate payload (env, asset, sender-actor, format) 3. load relevant readers (BaselineHoldingsReader, ManagedStateReader, ProposalReader) 4. ManagedStateGuard.can_transition() → reject mit audit `managed.policy_change_blocked` falls illegal 5. delegate to relevant Writer/Engine 6. emit success/abort/failed audit 7. write audit_snapshot file (SM-6) 8. update DB-Cache 9. return result_json für commands.result_json

Two-File-Atomic-Pattern (G-DR-14, decided 2026-05-10) für approve_managed_proposal + release_managed_asset:

```
1. Worker prüft G-DR-14: keine tradable_quote-Crossings
2. Backup beider Files (managed_state.json + baseline_holdings.json) via sha256 + .bak
3. atomic-write managed_state.json (neuer state)
4. atomic-write baseline_holdings.json (policy-Wechsel, neuer canonical_hash + account_hash)
5. DB-Cache update + audit_event + audit_snapshot in selber DB-Transaction
6. bei Failure auf einem der Schritte: Backup-Restore beider Files + audit `managed.two_file_atomic_rollback`
```

→ verhindert wallet-signature-Drift beim nächsten Bot-Restart. Detail in `99_master_boundaries.md §14.2`.

JSON-SoT + DB-Cache Schreibe-Reihenfolge (Q-MH-15 Hybrid C, decided 2026-05-10): - Worker schreibt **JSON zuerst** (atomic + sha256 + Backup) - danach DB-Cache in selber DB-Transaction - bei JSON-Failure: keine DB-Mutation - bei DB-INSERT-Failure nach JSON-Write: audit `managed.cache_drift_detected` + Retry beim nächsten Cycle

→ Bot liest IMMER JSON (kein DB-Read aus Bot). GUI liest IMMER DB-Cache (kein File-Read aus GUI).

---

4 · Audit-Snapshot-Files (SM-6)

Verzeichnis: `trading/state/audit_snapshots/<command_audit_log.id>.json`

Inhalt pro Snapshot:

```
{
  "snapshot_id": "uuid-...",
  "event_id": <command_audit_log.id>,
  "event_type": "managed.asset_promoted",
  "command_id": "uuid-...",
  "ts": "2026-05-10T...",
  "actor": "user_42",

  "from_state": "risk_proposed",
  "to_state": "managed_active",

  "asset": "SOL",

  "managed_state_before": {
    "managed_assets": {...},  // full file before write
    "pending_proposals": {...}
  },
  "managed_state_after": {
    "managed_assets": {...},  // full file after write
    "pending_proposals": {...}
  },

  "proposal_excerpt": {
    "proposal_id": "uuid-...",
    "asset": "SOL",
    "risk_recommendation": {...},
    "confidence": {...}
  },

  "metadata": {
    "idempotency_key": "...",
    "request_origin": "filament_wizard"
  }
}
```

Properties: - IMMUTABLE — niemals überschrieben, niemals gelöscht (außer dedizierter Archive-Phase) - pro State-Übergang genau 1 Snapshot - pro Apply / Reject / Pause / Resume / Release / Drift / Exit genau 1 Snapshot - Verzeichnis hat Cron-Cleanup nach 90 Tagen → Move zu `attic/` (separate Phase, nicht Phase MH)

Restore-Path (Backlog, BACKLOG): - Operator kann via dedizierter Phase „restore_managed_state_snapshot" managed_state.json auf einen Snapshot zurücksetzen - ABER: Wallet-Realität bleibt (Trades sind echt) - nur State-Files werden zurückgesetzt - Restore selbst ist eigener Command-Type + eigene Phase

---

5 · Idempotency-Detail

approve/reject auf gleicher proposal_id:

```
-- approve INSERT
INSERT INTO commands (command_id, command_type, idempotency_key, payload_json, ...)
VALUES (uuid_generate_v4(), 'approve_managed_proposal', 'mh:decide:<proposal_id>', ...)
ON CONFLICT (idempotency_key) DO NOTHING;
```

- erste approve schreibt → ok
- zweite approve oder erste reject auf gleiches proposal: `ON CONFLICT DO NOTHING` → existing command zurück
- Service-Layer wirft `proposal_already_decided` Error wenn existing command status != pending

pause/resume distinct keys: - Operator kann legitim 5× pause/resume klicken - jeder Klick = neuer Command (kein dedup) - Audit-Trail zeigt Click-History

---

6 · Worker-Daemon vs Manual --once (Q-MH-14 zentral)

Status quo (Stand v1): Worker läuft manuell als `--once`.

Konsequenz für Pause/Resume-UX: - Operator klickt Pause → Command in DB - Bot's main.py update_prices Cycle liest managed_state.json am Cycle-Anfang - bis Worker `--once` läuft, bleibt Command in `pending` - Bot-Cycle sieht alten state → potentielle Race (R4 aus Architektur-Doc)

Worker-Daemon-Aktivierung (MH-9): - via `docker compose --profile workers up -d clawbot-worker` - pollt `commands` jede X Sekunden - Pause-Commands werden synchron verarbeitet → Bot-Cycle sieht aktuellen state - R4-Race vermieden

Q-MH-14 Decision (2026-05-10): Worker-Daemon ist **Vorbedingung** für MH-1. Eigene Phase **MH-0.5** zwischen Architektur-Closure und MH-1: - `docker compose --profile workers up -d clawbot-worker` - Healthcheck-Verify - 24h-Stabilitäts-Beobachtung - klein (2-3h Operator-Zeit), reversibel via `docker compose down clawbot-worker` - löst R4-Race vor jeder Code-Phase

Daemon-Lifecycle: eigener Container-Service `clawbot-worker`. Restart-Policy: unless-stopped. Bei Crash + Watchdog-Failure: Operator-Notification via Telegram + Commands stecken in `pending` (Audit-Trail durabel).

Polling-Intervall: 5s (Default — entscheidet Q-MH-Folgefrage). Bot-Cycle ist 120s; Worker-Latency damit < 5s, R4-Race effektiv auf < 5s reduziert.

Forbidden in MH-1..6: Daemon ist bereits aktiv (MH-0.5), aber Worker-Code-Updates werden weiter manuell deployed (`docker cp` + Daemon-Restart). MH-7 (Bot-Side-Wiring) ist die einzige Phase die Bot-Restart erfordert.

---

7 · Restart-Strategie

Phase	Restart needed?	Begründung
MH-1 PHP-Registry	no	reine PHP-Erweiterung; Worker lädt erst beim nächsten `--once`
MH-2 Bot-Reader (dormant)	no	nicht aufgerufen aus aktivem Pfad
MH-3 proposal_engine standalone	no	nicht in main.py wired
MH-4 PHP-Services + DB-Migration	no	DB-Migration läuft via artisan, kein Bot-Touch
MH-5 Filament-UI	no	Browser-Reload genügt
MH-6 Worker-Handler	no	Worker-Code lädt erst beim nächsten `--once`. Bot-PID unverändert
MH-7 Bot-Side Wiring	**yes**	bootstrap_baseline + universe + execute_buy + drift-hook in main.py erweitert
MH-8 Drill	indirect (im Drill-Plan)	wenn Drill managed-Promote testet, ist Restart Teil des Drill-Plans
MH-9 Worker-Daemon-Aktivierung	**yes (Daemon)**	docker compose --profile workers up -d

Restart-Workflow MH-7 (analog RECON-2.2b Pattern): 1. Backup-Pflicht (G-DR-6 + DR-9): pg_dump + live_portfolio + .env + state/ + Memory 2. sha256-Verify Host vs Container 3. `docker cp` der modifizierten Files (main.py, baseline_bootstrap.py, scanner/universe.py, paper_trade.py, live_trade.py, command_worker.py) 4. Watchdog-Status-Check (G-DR-13 — `bot_watchdog.sh` muss `clawbot` enthalten) 5. SIGTERM Bot-PID → watchdog-getriggerter Neustart 6. Post-Cycle-Verify: 0 Tracebacks, kein Verhalten-Drift, audit_events-Stream sauber

8 · Service-Layer-Skelett (PHP, Phase MH-4)

```
class ProposalService
{
    public function createRequestCommand(string $asset, int $userId, string $intent): Command
    {
        return DB::transaction(function () use ($asset, $userId, $intent) {
            // 1. Validate Asset Format + nicht in Cooldown
            // 2. INSERT commands (status=pending, idempotency_key=mh:propose:...)
            // 3. INSERT audit_events (managed.asset_proposal_requested)
            // 4. INSERT managed_proposals (DB-Cache)
            return $command;
        });
    }

    public function createApproveCommand(string $proposalId, int $userId, string $variant, array $overrides):
Command
    {
        return DB::transaction(function () use ($proposalId, $userId, $variant, $overrides) {
            // 1. Validate proposal exists + state in {risk_proposed}
            // 2. Validate Hard-Confirm-String matches <asset>:<variant>
            // 3. Validate Confidence-Override-Flag if confidence < threshold
            // 4. INSERT commands (idempotency_key=mh:decide:<proposalId>)
            // 5. INSERT audit_events (managed.asset_promoted)
            return $command;
        });
    }

    public function createRejectCommand(string $proposalId, int $userId, ?string $reason): Command { ... }
}

class ManagedStateService
{
    public function createPauseCommand(string $asset, int $userId): Command { ... }
    public function createResumeCommand(string $asset, int $userId): Command { ... }
    public function createReleaseCommand(string $asset, int $userId, string $strategy): Command { ... }
}
```

9 · Drift-Detection (Bot-Self-Emit, `flag_managed_drift`)

Per-cycle Hook in main.py (Phase MH-7):

```
def detect_managed_drift_cycle(trader, managed_reader):
    snap = managed_reader.snapshot()
    if snap is None:
        return
    for asset, slot in snap.managed_assets.items():
        if slot["state"] != "managed_active":
            continue
        drift_kind = compute_drift_kind(asset, trader, slot)
        if drift_kind is None:
            continue
        # Bot self-emit Command
        client = CommandBusClient()
        client.create(
            command_type="flag_managed_drift",
            payload={"asset": asset, "drift_kind": drift_kind, "drift_details": ...},
            requested_by=None,  # System/Bot
        )
```

Bot-Side Validator (Worker-Handler): - `flag_managed_drift` Command: actor=bot/system zulässig - transition managed_active → managed_drift_alert - audit `managed.drift_detected` - KEIN sell

Boundary: `flag_managed_drift` ist die EINZIGE Bot-self-emitted Command-Type. Alles andere kommt vom Operator über GUI.

---

10 · Phasen-Sequenz für 05_commandbus_worker

MH-1 (PHP-Registry): - 8 neue Command-Types in CommandTypeRegistry - Validators für Payload-Format - ~25 Tests - Bump `command_bus_versions.json` auf v6

MH-4 (PHP-Services): - ProposalService + ManagedStateService - DB-Migration für `managed_proposals`, `managed_assets_history` - ~30 Tests

MH-6 (Bot-Worker-Handler): - 8 Handler in `command_worker.py` - ManagedStateGuard für DR-11 enforcement - audit-Snapshot writer - ~50 Tests

MH-9 (Worker-Daemon): - separate Phase - compose service `clawbot-worker` aktivieren - monitoring + healthcheck - siehe `06_testnet_drill.md` für Drill-Plan

→ Bot-Restart erst in MH-7. Vor MH-7 alles restart-frei.

— Ende CommandBus & Worker —

📄 `06_testnet_drill.md`

06 · Testnet Drill — Acceptance, Restore, Failure-Szenarien

Status: architektur-defined **Phase-Key:** RECON-MH-8 **Bezug:** End-to-End Verifikations-Drill auf TESTNET

Phase Cross-Reference

Field	Value
Dependencies	alle vorigen MH-Phasen abgeschlossen + `99_master_boundaries.md`
Required previous phases	RECON-MH-1..7 ☐
Restart required	yes (mehrere im Verlauf des Drills, dokumentiert)
Migration required	no (Migration ist in MH-4 abgeschlossen)
Mainnet-touch	NEIN, absolut
Risk-Level	high — erstes echtes End-to-End auf TESTNET

Allowed

- Pre-Drill Backup gemäß G-DR-6
- TESTNET-Bot-Restart (mehrere geplant)
- Worker `--once` Aufrufe (geplant)
- echtes `apply_baseline_holdings` mit frozen-only baseline (Frozen-only Drill, BACKLOG separater RECON-2.4)
- echtes `request_managed_proposal` für ein einzelnes Test-Asset
- echtes `approve_managed_proposal` mit voller Wizard-Durchlauf
- echtes `release_managed_asset` mit Cleanup
- `clear_baseline_holdings` Cleanup
- pre-/post-Drill Health-Snapshot
- Operator-Checkliste für jede Phase

NOT Allowed

- Mainnet-Touch jeglicher Art
- Worker-Daemon-Aktivierung im Drill (MH-9 ist separat danach)
- mehrere Assets gleichzeitig promoten (Q-MH-5 zuerst entscheiden, default Bulk später)
- Produktiv-DB-Migration (alles auf testnet-DB)
- echte Operator-Risk auf Mainnet-äquivalente Werte (TESTNET-Werte sind unkritisch)
- Drill-Modifikationen außerhalb dokumentierter Sequenz (kein adhoc-Probing)
- Drill ohne Pre-Backup
- Drill ohne explizites GO pro Sub-Phase

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-4	TESTNET-first	Settings.BINANCE_TESTNET=true verifiziert pre-Drill
G-DR-6	Backup-before-mutate	Pre-Drill Backup unter `/srv/shares/backups/recon_mh_drill_pre/`
G-DR-13	Watchdog clawbot	`bot_watchdog.sh` Stand verifiziert vor jedem Restart
G-SR-11	Mainnet-Block 5-Layer	alle 5 Layer pre-Drill verifiziert
RECON-2.2b	Restart-Pattern	sha256-Verify, SIGTERM, watchdog-getriggerter Neustart

Expected Test Surface

- Drill = manueller Smoketest, keine automatisierten Tests in dieser Phase
- ABER: Acceptance-Criteria in Form von SQL-Queries + Log-Greps verifiziert
- bestehende Suites NICHT betroffen
- post-Drill-Verify-Tests können in `09_test_strategy.md` als End-to-End-Suite dokumentiert werden

---

1 · Drill-Plan in 4 Sub-Phasen

```
Sub-Phase A: Frozen-Only Drill   (= RECON-2.4)
Sub-Phase B: Single-Asset Promote
Sub-Phase C: Drift-Simulation
Sub-Phase D: Cleanup + Audit-Verify
```

Jede Sub-Phase: explizites GO vom Operator vor Start.

---

2 · Pre-Drill (verbindlich für ALLE Sub-Phasen)

```
1. Backup-Pflicht (analog RECON-2.2b 6-Phase-Drill):
   - pg_dump GUI-DB nach /srv/shares/backups/recon_mh_drill_pre/
   - cp live_portfolio.json
   - cp .env (read-only)
   - cp -r state/ snapshot
   - Health-Snap (Bot-PID, container, .env mtime, key files presence)
   - tar memory/

2. Boundary-Checks:
   - Settings.BINANCE_TESTNET == true
   - Bot-PID läuft (kein Crash recent)
   - Watchdog-clawbot-Konvention OK
   - 5-Layer-Mainnet-Block intakt
   - command_bus_versions.json registry-Version stimmt mit DB-Cache

3. Pre-State-Snapshot:
   - SELECT COUNT(*) FROM commands GROUP BY status
   - SELECT COUNT(*) FROM audit_events WHERE event_type LIKE 'managed.%'
   - ls trading/state/baseline_holdings.json (existing or absent?)
   - ls trading/state/managed_state.json (absent expected)
   - ls trading/state/risk_proposals/ (empty expected)

4. Test-Asset auswählen:
   - low-cap, hohe Liquidität (z.B. SOL)
   - aktuelle Wallet-qty > 0
   - kein bestehender state['positions'][SOL/USDT]
   - nicht in N7.2-Blocklist (kein Stable, kein Peg)
```

---

3 · Sub-Phase A — Frozen-Only Drill (= RECON-2.4)

Ziel: Operator triggert via GUI das erste echte `apply_baseline_holdings`-Command mit ausschließlich `frozen` policies (kein managed). Verifiziert: - baseline-Workflow End-to-End - Universe-Filter wirksam (`reject_reason='baseline_frozen'`) - Bot-Verhalten bei present baseline - Bot-Restart NICHT erforderlich (Reader liest pro Cycle frisch)

Schritte: 1. Backup (siehe §2) 2. Operator öffnet `/admin/baseline-holdings` Filament Page 3. Operator paste'd Holdings JSON (z.B. 5-10 Assets, alle policy=frozen, default_policy=frozen) 4. Operator durchläuft Hard-Confirm `<count>:<sha8>` 5. Submit → Command in `commands` (status=pending) 6. Verify in DB: `sql SELECT * FROM commands WHERE command_type='apply_baseline_holdings' AND status='pending'; SELECT * FROM audit_events WHERE event_type='baseline.apply.requested' ORDER BY id DESC LIMIT 1;` 7. Manuell: `docker exec -w /home/node/.openclaw/workspace/trading clawbot python3 command_worker.py --once` 8. Verify Worker-Output: - command status='succeeded' - audit_event `baseline_holdings.apply_succeeded` - `trading/state/baseline_holdings.json` written - `trading/state/baseline_holdings.<TS>.bak` exists (= no, weil erste Apply) 9. Cycle-Verify (im laufenden Bot): - Decision-Log zeigt `0 baseline_frozen` reject_reason wenn Universe keine frozen Bases im Auswahlbereich hatte - oder >0 wenn Bases die im baseline frozen sind, Decision-Logs

gezeigt werden 10. Operator-Sanity: - Bot-PID unverändert - .env mtime unverändert - keine neuen Tracebacks im stdout-Log - wallet_signature bei nächstem Bot-Restart match (NICHT in dieser Sub-Phase)

Acceptance-Criteria Sub-Phase A: - ✓ command status='succeeded' - ✓ audit_event baseline_holdings.apply_succeeded vorhanden - ✓ baseline_holdings.json existiert mit korrekten 8 Sektionen - ✓ Bot läuft stabil - ✓ Universe-Filter zeigt baseline_frozen Hit-Count > 0 (wenn Bases gefroren)

---

4 · Sub-Phase B — Single-Asset Promote

Ziel: Operator promotet 1 einzelnes Test-Asset von frozen → managed_active.

Voraussetzung: Sub-Phase A erfolgreich abgeschlossen (Frozen-baseline existiert).

Schritte: 1. Backup (siehe §2) 2. Operator öffnet `/admin/managed-holdings` (neue Page aus MH-5) 3. Wallet-Tabelle zeigt Test-Asset (SOL) mit policy=frozen, state=frozen 4. Operator klickt „Analyse" → Step 1 Wizard öffnet 5. Operator schreibt Begründung (≥ 20 Zeichen) 6. Operator klickt „Bot-Analyse anfordern" → `request_managed_proposal` Command insert 7. Worker `--once` ausführen → proposal_engine generiert risk_proposal 8. Verify: - `risk_proposals/<id>.json` exists - `managed_proposals` DB-Cache row state='risk_proposed' - audit_event `managed.asset_proposal_generated` mit confidence-score 9. Wizard Step 2 zeigt Bot-Proposal Review (Polling 30s sieht es) 10. Operator durchläuft Step 2-5 (alle Wizard-Steps, keine Skip) 11. Step 5: Hard-Confirm-String `<asset>:<variant>` exakt eingeben (z.B. `SOL:recommended`) 12. Submit → `approve_managed_proposal` Command insert 13. Worker `--once` → managed_state.json wird geschrieben mit SOL=managed_active 14. Verify: - `managed_state.json[managed_assets][SOL].state == 'managed_active'` - `managed_state.json[managed_assets][SOL].synthetic_entry > 0` - audit_event `managed.asset_promoted` - audit_snapshot file vorhanden - DB-Cache `managed_assets_history` row inserted 15. **HIER ist Bot-Restart-Entscheidung:** wenn `managed.asset_promoted` einen Auto-Import-Trigger im Bot benötigt (synthetische Position in state['positions']), MUSS Bot restartet werden. Wenn das durch Worker direkt geschrieben wird, ist Restart optional. - Variante 1 (Restart): SIGTERM + Watchdog → neue PID → bootstrap_baseline → managed-Auto-Import läuft - Variante 2 (No Restart): Worker schreibt direkt in trader.state['positions'] (BAD — verletzt Worker-Boundary; vermutlich nicht der Pfad) - Final: Variante 1 = Bot-Restart in diesem Sub-Phase B 16. Post-Restart-Verify: - state['positions'][SOL/USDT] hat synthetic_entry + SL + TP - update_prices Cycle trackt SOL/USDT korrekt - kein Auto-Sell, kein Auto-Buy

Acceptance-Criteria Sub-Phase B: - ✓ Wizard durchlaufen ohne Skip - ✓ Hard-Confirm-String validiert - ✓ managed_state.json korrekt geschrieben - ✓ audit_snapshot erzeugt (immutable) - ✓ Bot tracked SOL als managed Position nach Restart - ✓ keine Tracebacks

---

5 · Sub-Phase C — Drift-Simulation

Ziel: drift-Mechanismus testen — qty-drift / price-drift / regime-drift einzeln.

Schritte: 1. Backup 2. **qty-drift:** Operator macht manuelles Sell auf Binance-TESTNET (z.B. 50% von SOL) - Bot's nächster update_prices Cycle detektiert qty-drift - flag_managed_drift Command (Bot self-emit) - state → managed_drift_alert - Verify: audit_event `managed.drift_detected` mit drift_kind='qty' - **Kein Auto-Sell** 3. Operator entscheidet: Override → state managed_active (Hard-Confirm) ODER Release → release_pending 4. **price-drift:** simulierbar via TESTNET-Preis-Manipulation (oder warte auf natürliche Bewegung > 2 ATR) - drift_kind='price', gleicher Pattern 5. **regime-drift:** schwer simulierbar; Drill-Plan definiert dies als optional bzw. via gemockten Regime-State 6. Cleanup: alle drift_alert states → managed_active oder release_pending

Acceptance-Criteria Sub-Phase C: - ✓ Bot detected qty-drift - ✓ flag_managed_drift Command Bot-self-emitted (kein Operator-Klick) - ✓ state managed_drift_alert - ✓ kein Auto-Sell (DR-9 enforcement) - ✓ Operator-Override funktioniert - ✓ audit_snapshot pro Drift-Übergang

---

6 · Sub-Phase D — Cleanup + Audit-Verify

Ziel: Drill-State zurück zu „leerer baseline" + Audit-Trail vollständig.

Schritte: 1. Backup 2. Operator klickt `release_managed_asset` für SOL → release_pending 3. Worker `--once` → Bot graceful exit (oder Operator entscheidet keine Sell) 4. state SOL → frozen 5. Operator triggert `clear_baseline_holdings` Command 6. Worker `--once` → baseline_holdings.json removed (mit backup) 7. Verify: - baseline_holdings.json absent - managed_state.json zeigt SOL=frozen oder Datei absent - audit_events zeigen

volle Lifecycle-Sequenz 8. Audit-Trail-Verify (SQL): `sql SELECT event_type, summary, created_at FROM audit_events WHERE created_at > '<drill_start_ts>' AND (event_type LIKE 'managed.%' OR event_type LIKE 'baseline.%') ORDER BY created_at;` - 1 row managed.asset_proposal_requested - 1 row managed.asset_proposal_generated - 1 row managed.asset_promoted - ≥ 1 row managed.drift_detected - 1 row managed.asset_release_requested - 1 row managed.asset_released - 1 row baseline.apply.requested + baseline_holdings.apply_succeeded - 1 row baseline.clear.requested + baseline_holdings.clear_succeeded 9. audit_snapshot files: zähle pro Übergang 1 Snapshot 10. Bot-State final-Verify: - Bot-PID stable (kann sich gegenüber Drill-Start geändert haben aufgrund Sub-Phase B Restart) - state['positions'] enthält keine SOL/USDT mehr (oder Operator hat manuell nicht released — dokumentieren)

Acceptance-Criteria Sub-Phase D: - ✓ baseline + managed_state cleanly removed - ✓ vollständiger Audit-Trail abrufbar - ✓ pro State-Übergang 1 audit_snapshot - ✓ Bot stabil

---

7 · Restore-Plan (bei Failure)

Falls Drill in einer Sub-Phase fehlschlägt:

1. **Stop sofort** — keine weiteren Operator-Klicks
2. Backup-Verify: pre-Drill-Backup unter `/srv/shares/backups/recon_mh_drill_pre/`
3. Restore-Optionen: - **State-File-Restore:** kopiere `baseline_holdings.<TS>.bak` zurück - **DB-Restore:** pg_restore aus `gui_db_<TS>.sql.gz` (TESTNET-DB only) - **State-Snapshot-Restore:** kopiere `live_portfolio_<TS>.json` zurück - **Memory-Restore:** untar memory backup
4. Bot-Restart auf restored state via Watchdog
5. Verify pre-Drill state wiederhergestellt
6. Failure-Analyse: Was war die Ursache? Audit dokumentieren.

---

8 · Failure-Szenarien (geplant durchspielbar)

Szenario	Erwartetes Verhalten
Worker-`--once` schlägt fehl (z.B. JSON parse error)	Command status=failed mit error_message; State unverändert
Filament Wizard Step 5 Submit ohne Hard-Confirm-Match	Form-Validation reject; kein Command insert
Operator versucht zweites approve auf gleiche proposal_id	precondition_rejected via idempotency_key
Bot detected drift während Operator Pause klickt	beide Commands queued; Worker verarbeitet in Reihenfolge; State eventually consistent
TTL-Expiry läuft während Operator approve klickt	row-lock verhindert Race; erstes Schreiben gewinnt
Mainnet-Versuch (synthetisch — env=mainnet im Payload)	5-Layer-Block greift; payload-validator reject + audit precondition_rejected

---

9 · Operator-Checkliste (Drill-Buchstabe)

```
□ Pre-Drill Backup vollständig
□ /srv/shares/backups/recon_mh_drill_pre/ MANIFEST geschrieben
□ Settings.BINANCE_TESTNET == true verifiziert
□ Watchdog clawbot-Konvention OK
□ Sub-Phase A frozen-only Drill grün
□ Sub-Phase B Single-Asset Promote grün (inkl. Bot-Restart)
□ Sub-Phase C Drift-Simulation grün
□ Sub-Phase D Cleanup + Audit-Verify grün
□ Bot-PID Final dokumentiert
□ post-Drill Health-Snap geschrieben
□ Audit-Trail Excerpt nach /srv/shares/backups/recon_mh_drill_post/
```

---

10 · Phasen-Sequenz für 06_testnet_drill

Voraussetzungen: - **MH-0.5 Worker-Daemon aktiv** (Q-MH-14 decided 2026-05-10) — Pause/Resume/Release synchron verarbeitet, kein R4-Race - alle MH-1..7 abgeschlossen - Q-MH-1..18 entschieden (architektur-prägende: ☑ 2026-05-10; verbleibende 13 entscheidbar während Drill-Vorbereitung) - Operator hat 2-3h Zeit für vollen Drill - Test-Asset gewählt (vermutlich SOL — liquide, mid-cap, weder Stable noch Peg) - Testnet-Wallet hat Test-Asset mit qty > 0 - **G-DR-14 enforcement aktiv** — Test-Asset darf nicht `tradable_quote` sein (USDT/USDC/etc.)

Sequenz: 1. Pre-Drill (siehe §2) 2. Sub-Phase A 3. **Operator-GO Pause + Review der Acceptance-Criteria** 4. Sub-Phase B (inkl. Bot-Restart) 5. **Operator-GO Pause + Review** 6. Sub-Phase C 7. **Operator-GO Pause + Review** 8. Sub-Phase D 9. Post-Drill Closure-Checkliste

Lieferzustand nach Drill: managed_state.json absent (Cleanup), Bot stable, alle Audit-Events dokumentiert. Drill-Bericht als Memory-File pinned.

— Ende Testnet Drill —

📄 `07_mainnet_future.md`

07 · Mainnet Future — Voraussetzungen, Sicherheitsregeln, Limits

Status: BACKLOG · BLOCKIERT **Phase-Key:** RECON-MH-future / RECON-2.5 Mainnet-Readiness Review **Bezug:** Erst bearbeiten NACH RECON-MH-1..9 + RECON-2.5 Sign-Off

Phase Cross-Reference

Field	Value
Dependencies	ALLE vorigen MH-Phasen abgeschlossen + RECON-2.5 abgeschlossen
Required previous phases	RECON-MH-1..9 ☐, RECON-2.5 ☐ (Mainnet-Readiness Review)
Restart required	yes (mehrere)
Migration required	tbd (vermutlich nein)
Mainnet-touch	**JA** — diese Phase ist die einzige in der gesamten MH-Roadmap die Mainnet überhaupt anfasst
Risk-Level	**kritisch** — größte Gefahr in der gesamten Roadmap

⚠ Wichtigster Hinweis

Diese Datei ist **reine Backlog-Dokumentation**. Keine der hier beschriebenen Aktionen darf vor RECON-2.5 Mainnet-Readiness Review Sign-Off ausgeführt werden.

Mainnet bleibt durch 5-Layer-Block durable blockiert (siehe `99_master_boundaries.md` §8). Diese Datei beschreibt **was eines Tages passieren würde**, nicht **was als nächstes ansteht**.

Allowed (in dieser Phase, NACH Sign-Off)

- 5-Layer-Mainnet-Block-Audit (manuell + automatisiert)
- Settings.BINANCE_TESTNET=false aktivieren — **NUR nach 2-Personen-Sign-Off**
- Mainnet-Connection-Test (read-only fetch_balance, kein Order)
- erste Mainnet-Apply mit dry_run=true (read-only, schreibt kein State)
- Mainnet wallet-signature Erstellung
- Mainnet-Operator-Drill (Operator-Drill 502 Pattern, aber mit Mainnet-API)
- 5-Min-Rollback-Plan testen
- alle B-FEE-FIX 1-4 verifiziert grün auf Mainnet-Symbol-Sample
- B-OUTAGE-RESILIENCE-1 verifiziert auf Mainnet-Connection

NOT Allowed (auch in dieser Phase)

- Mainnet-Switch ohne 2-Personen-Sign-Off
- Mainnet-Switch ohne abgeschlossene RECON-2.5 Mainnet-Readiness Review
- Mainnet-Switch ohne abgeschlossene B-FEE-FIX 1-4
- Mainnet-Switch ohne B-OUTAGE-RESILIENCE-1 Operator-Drill grün auf Mainnet
- Mainnet-Switch ohne erfolgreichen RECON-MH-8 Testnet-Drill
- Mainnet-Switch ohne dokumentiertem 5-Min-Rollback-Plan
- Mainnet aggressive Variante in proposal_engine (Q-MH-13 disabled-on-Mainnet)
- Mainnet ohne Mainnet-spezifische Cooldowns (Q-MH-12 → 7 Tage default)
- Mainnet-Beträge die im Verlust-Fall unbeherrschbar wären (Q-MH-13 enforcement)
- ad-hoc-Mainnet-Switch ohne Backup
- Mainnet-Switch ohne Telegram-Push-Channel aktiv (Operator-Notification-Path muss vor Mainnet stehen)

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-4	TESTNET-first dauerhaft	TESTNET bleibt parallel zu Mainnet erhalten
G-SR-11	Mainnet-Block 5-Layer	alle 5 Layer manuell verifiziert pre-Switch
feedback_testnet_permanent	TESTNET permanent	TESTNET wird auch nach Mainnet-Aktivierung erhalten
Bot-PID-Timeline	jede Mainnet-Aktivierung schreibt neue PID in Memory	siehe DR-10

Expected Test Surface

- 0 automated Tests in dieser Phase
- ABER: vor jeder Aktion 100% manuelle Verifikation der 5 Layer
- post-Switch Health-Monitoring durch B-OUTAGE-RESILIENCE-1 Patterns

---

1 · Voraussetzungen vor Mainnet-Switch (alle erforderlich)

1.1 Phasen-Status

- ✓ RECON-MH-1..7 alle abgeschlossen (Code + Tests grün)
- ✓ RECON-MH-8 Testnet-Drill **erfolgreich** durchgeführt (alle 4 Sub-Phasen)
- ✓ RECON-MH-9 Worker-Daemon aktiviert + 24h+ stabil ohne Crash
- ✓ RECON-2.5 Mainnet-Readiness Review formal abgeschlossen
- ✓ Q-MH-1..18 alle entschieden + dokumentiert in `08_open_questions.md`
- ✓ alle DR-1..13 + G-DR-1..13 + G-SR-1..12 grün

1.2 Bot-Side Health

- ✓ B-FEE-FIX 1-4 verifiziert auf Mainnet-Symbol-Sample (mindestens 5 Mainnet-Symbole)
- ✓ B-OUTAGE-RESILIENCE-1 Operator-Drill 502 erfolgreich auf Mainnet-Connection
- ✓ N7.2 Stablecoin/Peg-Block aktiv und getestet
- ✓ G10-4.2 Apply-Path stabil (RuntimeConfig-Override-Pattern)
- ✓ Watchdog `clawbot` Convention durabel
- ✓ Backup-Strategie aktiv (DB-Backup-Cron läuft, Drill durchgeführt)
- ✓ Reporter-Telegram-Channel funktioniert (Push-Notifications)

1.3 Operator-Vorbereitung

- ✓ 2-Personen-Sign-Off-Process etabliert
- ✓ Operator hat Mainnet-API-Keys bereit (read-only zuerst)
- ✓ Mainnet-Wallet ist mit kontrollierter Test-Allokation befüllt (z.B. nur 1-5% des Gesamt-Vermögens für erstes Engagement)
- ✓ 5-Min-Rollback-Plan auf TESTNET geübt
- ✓ Telegram-Notification-Channel für Mainnet-Aktivierung dokumentiert
- ✓ Rollback-Backup-Pfade festgelegt

1.4 Code-Zustand

- ✓ master HEAD = `<dokumentierter commit>` (mind. RECON-MH-9 abschluss)
- ✓ alle PHP-Tests grün (volle Suite)
- ✓ alle Bot-Tests grün (volle Suite)
- ✓ keine pending TODOs in MH-Code
- ✓ keine experimental flags (Feature-Flags, etc.)

---

2 · 5-Layer-Mainnet-Block (verbindlich)

Bevor Mainnet aktiviert wird, MUSS jeder Layer manuell + automatisiert verifiziert sein:

Layer	Komponente	Pre-Switch-Verify
L1	`.env` Settings	`BINANCE_TESTNET=true` derzeit; Switch zu false ist die einzige hier erlaubte Mutation
L2	`BaselineHoldingsWriter.ALLOWED_ENVIRONMENTS`	Konstante = `{'testnet'}`; Mainnet-Erweiterung würde gleichzeitige Code-Änderung + Test bedeuten
L3	`command_worker.ALLOWED_ENVIRONMENTS`	gleiche Konstante; Mainnet-Erweiterung muss synchron mit L2 sein
L4	`CommandTypeRegistry::validateApplyBaselineHoldings` payload-Check	hardcoded `environment === 'testnet'`; Mainnet-Erweiterung muss synchron mit L1-L3 sein
L5	GUI Filament hardcode	`ApplyBaselineService::APPLY_ENVIRONMENT = 'testnet'`; Mainnet-Erweiterung muss synchron mit L1-L4 sein

Aktivierungs-Sequenz (analog Layer-für-Layer-Removal):

```
Phase 0:    Audit aller 5 Layer manuell + automatisiert
Phase 1:    L2-L5 erlauben sowohl 'testnet' als auch 'mainnet' (additive Erweiterung)
Phase 2:    L1 Settings.BINANCE_TESTNET=false (einzige Mainnet-spezifische Mutation)
Phase 3:    Bot-Restart mit Mainnet-API-Keys (read-only zuerst)
Phase 4:    Mainnet-Connection-Test (fetch_balance, fetch_ticker)
Phase 5:    Mainnet-Operator-Drill (analog 502 Pattern, aber mit Mainnet-API)
Phase 6:    erste Mainnet-Apply mit dry_run=true (read-only, kein State-Write)
Phase 7:    erste Mainnet-Apply mit dry_run=false (frozen-only baseline)
Phase 8:    erste Mainnet-Promote mit single Asset, kontrollierte allocation (z.B. ≤ 100 USDT-Wert)
Phase 9:    24h Monitoring, Bot-Verhalten Cross-Check
Phase 10:   Operator-Recap + Sign-Off + Mainnet-Lifecycle-Phase startet
```

Jede Phase einzeln Approval-Required durch 2-Personen-Sign-Off.

---

3 · Mainnet-Sicherheitsregeln (MH-spezifisch)

Regel	Beschreibung
MN-SR-1	aggressive Variante in proposal_engine **HARD-DISABLED** auf Mainnet (Q-MH-13 decided 2026-05-10). Engine entfernt `aggressive` aus `alternative_proposals[]`-Output wenn env=mainnet. Wizard zeigt nur 2 Karten.
MN-SR-2	DCA-Recommendation in proposal_engine **HARD-DISABLED** auf Mainnet (Q-MH-13 decided 2026-05-10). `dca_recommended=False` hardcoded auf Mainnet, egal was Engine-Logik sagen würde.
MN-SR-3	t2_pump_dump als `recommended` strategy_group **HARD-FALLBACK auf t1_core** mit Warning + audit `managed.t2_recommendation_blocked_mainnet` (Q-MH-13 decided 2026-05-10).
MN-SR-3.1	**G-DR-14 (decided 2026-05-10) gilt auf Mainnet doppelt streng:** Two-File-Atomic-Pattern für policy-Wechsel ist Pflicht. Bei einem File-Failure → Backup-Restore beider Files + Mainnet-Stop + Operator-Notification (Telegram).
MN-SR-4	Cooldown auf Mainnet = 7 Tage (Q-MH-12)
MN-SR-5	Max-Allocation pro Asset auf Mainnet = 5% des Gesamt-Wallet-Werts (Q-MH-13 erweitert)
MN-SR-6	Max-Anzahl gleichzeitig managed Assets auf Mainnet = 3 (vergleichsweise zu höheren TESTNET-Werten)
MN-SR-7	Confidence-Threshold auf Mainnet erhöht: 0.65 (statt 0.50 auf TESTNET, Q-MH-9)
MN-SR-8	Volatility-Kill-Threshold auf Mainnet niedriger: 15% 30d-vol (statt 25% TESTNET, Q-MH-18)
MN-SR-9	Mainnet-Hard-Confirm zusätzlich `MAINNET-CONFIRMED-<asset>` als zweiten String beim Promote
MN-SR-10	Mainnet Operator-Drill 502 alle 30 Tage wiederholen (analog regulatorisches Drill)
MN-SR-11	Mainnet-DB-Backup täglich + 30-Tage-Retention (B-OUTAGE-RESILIENCE-Pattern)
MN-SR-12	bei jedem Mainnet-Mainnet-Crash: pre-Crash Bot-State-Snapshot in Memory speichern

4 · 5-Min-Rollback-Plan

Vor jeder Mainnet-Aktivierung dokumentieren + üben:

```
T+0min  Operator detected Issue
T+1min  Operator klickt "Emergency Pause All Managed" Button
        → schreibt managed_state.json[*].state = managed_paused
        → Worker prozessiert sofort (Daemon)
        → Bot stops new BUYs, hält offene SL/TP
T+2min  Operator entscheidet: Restore-from-Backup oder Continue
T+3min  IF Restore: pg_restore TESTNET-DB-Backup vom Vortag
        IF Continue: Operator entscheidet manuell pro Asset
T+4min  Operator klickt "Switch to TESTNET" Button
        → .env Settings.BINANCE_TESTNET=true
        → Bot-Restart via Watchdog
T+5min  Bot läuft auf TESTNET, Mainnet-Connection geschlossen
```

Phase muss vor Mainnet-Switch auf TESTNET geübt sein: - mit Mock-Mainnet-Issue (z.B. simulierte 502) - Wall-Time-Verify: < 5 min - Audit-Trail-Verify: alle Schritte sind audit_event-tracked

5 · Mainnet Lifecycle (nach erfolgreichem Switch)

Erste 30 Tage (Probebetrieb): - max 1 Asset gleichzeitig managed - max 100 USDT-Wert pro Position (oder 1% Wallet, whichever lower) - tägliche Operator-Review der `managed_state.json` - wöchentlicher Sign-Off-Sitzung - KEIN aggressive variant - KEIN DCA

Tag 31-90 (Erweiterung): - max 3 Assets gleichzeitig managed (MN-SR-6) - Allocation-Limits gelockert auf 5% des Wallets (MN-SR-5) - Recommended Variante regulär verfügbar - Conservative Variante regulär verfügbar -

aggressive bleibt disabled

Nach Tag 90 (Stable): - regulärer Lifecycle - Operator hat freie Hand innerhalb der MN-SR-1..12 - regelmäßiges Drill alle 30 Tage (MN-SR-10)

---

6 · Was BEWUSST nicht in dieser Phase ist

- Multi-wallet (mehrere Mainnet-Konten gleichzeitig) — separate Phase
- Copy-Trading auf Mainnet (T-SPLIT-3 t3_copy_trading bleibt disabled bis eigene Phase)
- Tax-Layer für Mainnet-Trades — separate Phase
- Profit-Reinvestment-Strategie (CAP-1) — separate Phase
- Mainnet-only-Strategien (Bot ist TESTNET-first, alle Strategien testnet-equal)

Siehe `10_backlog_future_extensions.md` für diese Themen.

---

7 · Rollback nach Mainnet-Aktivierung (Notfall)

Wenn Mainnet-Switch im Nachhinein als Fehler erkannt wird:

1. Sofort Emergency-Pause-All-Managed (siehe §4)
2. Operator entscheidet: alle Managed → release_pending → frozen
3. Wallet-Sells via Operator manuell (NICHT automatisch — Bot ist Berater)
4. Settings.BINANCE_TESTNET=true zurück
5. Bot-Restart via Watchdog
6. Audit-Trail-Verify: alle Mainnet-Events dokumentiert
7. Post-Mortem-Memory-Eintrag mit Lessons-learned
8. Re-Mainnet-Aktivierung NUR nach erneutem 2-Personen-Sign-Off + neuem Drill

---

8 · Cross-Reference

Memory / Doc	Inhalt
`recon_status_pin.md` §3.4 + §5 SR-9	RECON-2.5 Mainnet-Readiness BACKLOG
`b_fee_fix_3_status.md` (Memory)	B-FEE-FIX-3 Mainnet-Blocker durable rule
`operator_drill_outage_502_status.md`	Drill-Pattern für Mainnet-Drill-Wiederholung (MN-SR-10)
`feedback_testnet_permanent_paper_rename.md`	TESTNET bleibt parallel zu Mainnet erhalten
`cap_1_preflight_backlog.md`	Profit-Reinvestment auf Mainnet, separate Phase

---

9 · Final-Hinweis

Diese Datei ist nicht handlungsanleitend. Sie beschreibt einen Zukunfts-Zustand. Wenn jemand diese Datei liest und denkt „lass uns das machen" — STOP. Erst RECON-MH-1..9 + RECON-2.5 abschließen, dann erneut hier ankommen, dann 2-Personen-Sign-Off einholen, dann eine Phase nach der anderen mit der vorhergehenden Verifikation aktivieren.

Mainnet ist die größte Gefahr im gesamten Trading-Bot-Stack. Diese Sub-Roadmap ist explizit so gebaut, dass Mainnet am Ende kommt — nicht am Anfang.

— Ende Mainnet Future (BACKLOG, BLOCKIERT) —

📄 `08_open_questions.md`

08 · Open Questions — Q-MH-1..18 mit Default-Empfehlungen

Status: open — keine Antworten verbindlich entschieden bis hier `decided` als Status steht **Phase-Key:** Voraussetzung für RECON-MH-1..7 **Bezug:** zentrale Q&A-Liste; jede Antwort wird in den entsprechenden Phase-File propagiert

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `00_overview.md`
Required previous phases	none — kann jederzeit beantwortet werden
Restart required	no
Migration required	no
Mainnet-touch	no
Risk-Level	low (Q&A-Doc), aber HIGH wenn Antworten falsch sind (beeinflusst Code-Phasen)

Allowed

- Operator-Antworten zu jeder Q-MH ergänzen
- Status-Update pro Frage: `open / decided / deferred / blocked`
- Begründung pro Antwort
- Cross-Reference zu betroffenen Phase-Files
- neue Q-MH-Fragen ergänzen falls Architektur-Iterationen Bedarf bringen

NOT Allowed

- bestehende Antworten überschreiben ohne Versions-Bump
- Q-MH löschen (nur als `deprecated` markieren mit Begründung)
- Code-Beispiele oder Implementierungs-Details (gehören in entsprechende Phase-Files)

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
G-DR-2	Operator final authority	Antworten kommen vom Operator, nicht vom Bot/Architekt
G-DR-7	UX-Multi-Step-Wizard	Q-MH-2 + Q-MH-3 + Q-MH-11 betreffen Wizard-Aufbau
G-DR-10	Risk-Proposals versioniert	Q-MH-9 betrifft Confidence-Threshold + risk_model_version

Expected Test Surface

- 0 Tests (Q&A-Doc)
- aber: Tests in `09_test_strategy.md` referenzieren beantwortete Werte als Test-Inputs

Q-MH-1 · Proposal-TTL

Frage: Wie lange darf ein `risk_proposed`-Proposal liegen, bevor es auto-rejected wird?

Optionen: - A) 24h - B) 7 Tage - C) konfigurierbar pro Asset

Default-Empfehlung: B — 7 Tage

Auswirkung wenn beantwortet: - `03_state_machine.md` SM-3 schreibt TTL-Wert in scheduled-job - `02_risk_proposal_engine.md` setzt `expires_at` im Proposal-File - `05_commandbus_worker.md` TTL-Expiry-Job läuft 1× täglich

Status: `open`

Q-MH-2 · Wie viele Varianten generiert Bot?

Frage: Soll Bot 1, 2 oder 3 Varianten pro Proposal liefern (recommended / conservative / aggressive)?

Optionen: - A) nur recommended (1) - B) recommended + conservative (2) - C) recommended + conservative + aggressive (3)

Default-Empfehlung: C auf TESTNET, B auf Mainnet (aggressive ist Mainnet-disabled per Q-MH-13)

Auswirkung: - `02_risk_proposal_engine.md` §3 generiert entsprechend - `04_gui_operator_flow.md` Wizard Step 3 zeigt entsprechend - `07_mainnet_future.md` MN-SR-1 bleibt enforced

Status: `decided` 2026-05-10 — Operator hat Default akzeptiert. Engine berechnet 3 Varianten intern; `alternative_proposals[]`-Output env-abhängig: Testnet `[conservative, aggressive]`, Mainnet `[conservative]`. Vollständige Begründung in `sessions/q_mh_session_2026-05-10_architecture_priming.md` §Q-MH-2.

Q-MH-3 · Operator-Override-Tiefe

Frage: Welche Felder darf Operator bei Approve überschreiben?

Optionen: - A) minimal: nur SL/TP/max_alloc - B) medium: + trailing_stop, dca_settings - C) maximal: alles inkl. strategy_group

Default-Empfehlung: B — medium

Begründung: strategy_group ist Engine-Entscheidung; Operator-Override würde Engine-Logik aushebeln.

Auswirkung: - `04_gui_operator_flow.md` Wizard Step 3 erlaubt entsprechend - `05_commandbus_worker.md` approve-Validator akzeptiert entsprechend Felder im Override

Status: `open`

Q-MH-4 · release_pending Behavior

Frage: Bei `release_managed_asset`:

Optionen: - A) graceful (warte SL/TP, dann frozen) - B) immediate (Bot inactive, Position bleibt offen, frozen) - C) choice im UI (Operator entscheidet pro release)

Default-Empfehlung: C (mit default graceful)

Auswirkung: - `03_state_machine.md` Übergangs-Tabelle release_pending → frozen - `04_gui_operator_flow.md` Release-Modal hat Wahl-Option

Status: `open`

Q-MH-5 · Multi-Asset-Bulk-Proposal

Frage: Soll Operator mehrere Assets gleichzeitig zur Analyse markieren?

Optionen: - A) ja, als Bulk-Action (1 Command für alle) - B) nein, single-asset only (vermeidet R3 Race) - C) Bulk-Markierung mit Einzel-Workflows pro Asset

Default-Empfehlung: C (markiert mehrere, jeder bekommt eigenen Proposal-Workflow)

Auswirkung: - `04_gui_operator_flow.md` Tabelle erlaubt Mehrfach-Selektion - `05_commandbus_worker.md` request_managed_proposal Command bleibt single-asset (Bulk wird in PHP-Service zu N Commands aufgeteilt)

Status: `open`

Q-MH-6 · Bot self-initiated Proposals

Frage: Darf Bot selbst initiieren?

Optionen: - A) nein, nur Operator - B) ja, Bot darf vorschlagen via Alert; Operator-Confirm bleibt - C) ja, Auto-Promote bei score > X (gefährlich!)

Default-Empfehlung: B — Bot darf alerten, aber Operator entscheidet

Auswirkung: - `02_risk_proposal_engine.md` Engine kann via Cron-Job triggered werden mit `captured_via='bot_alert'` - `03_state_machine.md` Pfad frozen → proposal_pending OPENS für actor=bot_alert - KEIN Auto-Promote (G-DR-2 bleibt)

Status: `open`

Q-MH-7 · Drift-Schwellen

Frage: Welche Werte triggern `managed_drift_alert`?

Optionen: - qty drift: > 5% / 10% / 15%? - price drift: > 1.5 ATR / 2 ATR / 2.5 ATR? - regime drift: BEAR↔BULL / BEAR↔RANGING / BULL↔RANGING?

Default-Empfehlung: qty>5% OR price out 2-ATR OR regime BEAR↔BULL

Auswirkung: - `03_state_machine.md` Drift-Detection-Schwellen - `02_risk_proposal_engine.md` Vergleichswerte für drift-Detection - `09_test_strategy.md` Drift-Tests nutzen diese Werte

Status: `open`

Q-MH-8 · Pause-Verhalten

Frage: Bei `managed_paused`:

Optionen: - A) SL/TP-Tracking aktiv aber kein neuer Trade - B) SL/TP off (Position bleibt offen, kein automatischer Exit) - C) Choice pro Pause

Default-Empfehlung: A (SL/TP bleibt aktiv — Operator kann sich auf Stops verlassen)

Auswirkung: - `03_state_machine.md` Pause-Verhalten dokumentiert - main.py update_prices Cycle berücksichtigt managed_paused-state

Status: `open`

Q-MH-9 · Re-Engage nach exit_executed

Frage: Wenn SL/TP gehit:

Optionen: - A) state → frozen, Operator muss neu propose'n - B) state → "exit_executed_can_re_engage", Bot kann auto-Proposal generieren

Default-Empfehlung: A (frozen, sauber)

Status: `open`

Q-MH-10 · synthetic_entry Method

Frage: Bot's synthetic_entry-Berechnung:

Optionen: - A) current_price (heute) - B) cost_basis (from trade history if available) - C) VWAP über letzten X Tagen - D) Operator-choice bei jedem Promote (default current_price)

Default-Empfehlung: D — Operator-choice mit current_price-Default

Auswirkung: - `02_risk_proposal_engine.md` synthetic_entry_method-Feld in Proposal - `04_gui_operator_flow.md` Wizard Step 3 zeigt Wahl - `01_foundation.md` managed_state.synthetic_entry_method dokumentiert Wahl

Status: `open`

Q-MH-11 · Hard-Confirm-String approve

Frage: Format des Hard-Confirm-Strings für `approve_managed_proposal`?

Optionen: - A) nur Asset (`SOL`) - B) `<asset>:<variant>` (`SOL:recommended`) - C) `proposal_id`-suffix (`SOL:a3f9c7`)

Default-Empfehlung: B — `<asset>:<variant>`

Auswirkung: - `04_gui_operator_flow.md` §6 Hard-Confirm-Patterns Tabelle - `05_commandbus_worker.md` payload-validator - `02_risk_proposal_engine.md` proposal-Output enthält `variant`-Feld

Status: `decided` 2026-05-10 — Operator hat Default akzeptiert. Pattern: `<asset>:<variant>` (z.B. `SOL:recommended`). Override-Pfad: `<asset>:override:<variant>` (z.B. `SOL:override:aggressive`). Vollständige Begründung in `sessions/q_mh_session_2026-05-10_architecture_priming.md` §Q-MH-11.

Q-MH-12 · Cool-down nach reject/release

Frage: Cool-down nach `proposal_rejected` oder `released_back_to_frozen`:

Optionen: - A) nein, sofort neu propose'n erlaubt - B) 24h - C) konfigurierbar - D) 7 Tage auf Mainnet, 24h auf TESTNET

Default-Empfehlung: D — TESTNET 24h, Mainnet 7d

Auswirkung: - `03_state_machine.md` SM-2 Cooldown-Mechanismus - `05_commandbus_worker.md` request_managed_proposal-Validator

Status: `open`

Q-MH-13 · Mainnet-Disable-Features

Frage: Welche Features sind Mainnet-disabled?

Optionen: - aggressive variant? - DCA in proposal? - t2_pump_dump als rec?

Default-Empfehlung: alle drei: disabled auf Mainnet

Auswirkung: - `07_mainnet_future.md` MN-SR-1, MN-SR-2, MN-SR-3 - `02_risk_proposal_engine.md` env=mainnet checks

Status: `decided` 2026-05-10 — Operator hat Default akzeptiert. **Drei Hard-Disables auf Mainnet:** (1) aggressive Variante in proposal_engine; (2) `dca_recommended` Output hardcoded auf `false`; (3) `t2_pump_dump` als `recommended` strategy_group → hard-fallback to `t1_core` mit Warning. Operator-Override (Q-MH-3) erlaubt Override **nur für SL/TP/max_alloc/trailing_stop**, NIE für strategy_group oder variant. Vollständige Begründung in `sessions/q_mh_session_2026-05-10_architecture_priming.md` §Q-MH-13.

Q-MH-14 · Worker-Daemon-Aktivierung als Vorbedingung

Frage: Ist Worker-Daemon required für sichere managed-Lifecycle-UX?

Optionen: - A) optional (Pause/Resume haben Latenz) - B) required für RECON-MH (sonst R4-Race)

Default-Empfehlung: B — required

Begründung: R4 (Bot-Cycle vs Pause-Command) nicht akzeptabel ohne Daemon. Worker-Daemon-Aktivierung wird MH-0.5 vor MH-1.

Auswirkung: - `05_commandbus_worker.md` §6 Worker-Daemon-Sektion - `00_overview.md` MH-0.5 als zwingender Schritt vor MH-1 - `06_testnet_drill.md` Daemon ist Drill-Voraussetzung

Status: `decided` 2026-05-10 — Operator hat **Daemon-Twist** akzeptiert: Worker-Daemon wird **MH-0.5 vor MH-1** (klein, reversibel via `docker compose down clawbot-worker`). Healthcheck-Verify + 24h-Stabilitäts-Beobachtung vor MH-1-Start. Löst R4-Race vor jeder Code-Phase. Vollständige Begründung in `sessions/q_mh_session_2026-05-10_architecture_priming.md` §Q-MH-14.

Q-MH-15 · managed_state Source of Truth

Frage: managed_state Source of Truth:

Optionen: - A) JSON-File (parity mit baseline_holdings.json) - B) GUI-DB Tabelle (parity mit ConfigProfile) - C) beide (JSON für Bot, DB als Cache)

Default-Empfehlung: C (JSON für Bot, DB als Cache + GUI-Read)

Auswirkung: - `01_foundation.md` Datenmodell - `05_commandbus_worker.md` Worker schreibt beides transactional - `04_gui_operator_flow.md` GUI liest aus DB für Performance

Status: `decided` 2026-05-10 — Operator hat **Hybrid C** akzeptiert: JSON ist SoT für Bot, DB ist Read-Cache für GUI. Worker schreibt **transactional double-write** (zuerst JSON atomic + sha256 + Backup, dann DB-Cache in selber DB-Transaction). Bei JSON-Write-Failure: keine DB-Mutation. Bei DB-INSERT-Failure nach JSON-Write: Worker emittet `managed.cache_drift_detected` audit + retry-Logik beim nächsten Cycle. Vollständige Begründung in `sessions/q_mh_session_2026-05-10_architecture_priming.md` §Q-MH-15.

Q-MH-16 · Cooldown-Override durch Operator

Frage (NEU aus Reviewer-Ergänzung): Darf Operator die Cooldown-Sperre überschreiben?

Optionen: - A) nein, Cooldown ist hart - B) ja, mit extra Hard-Confirm - C) admin-only Override

Default-Empfehlung: B — mit extra Hard-Confirm `OVERRIDE-COOLDOWN-<asset>`

Status: `open`

Q-MH-17 · Confidence-Threshold-Wert

Frage (NEU aus Reviewer SM-4): Bei welchem Confidence-Score greift der Hard-Gate?

Optionen: - A) 0.50 (50%) - B) 0.60 (60%) - C) konfigurierbar pro Asset / pro env

Default-Empfehlung: 0.50 auf TESTNET, 0.65 auf Mainnet

Auswirkung: - `02_risk_proposal_engine.md` confidence_threshold-Param - `04_gui_operator_flow.md` Wizard Step 4 zeigt Override-Hint - `07_mainnet_future.md` MN-SR-7

Status: `open`

Q-MH-18 · Volatility-Kill-Schwelle

Frage (NEU aus Reviewer SM-5): Bei welcher Volatility greift der Kill-Switch?

Optionen: - A) 25 % 30d-volatility (default) - B) 30 % 30d-volatility - C) konfigurierbar pro Asset / pro env

Default-Empfehlung: 25% auf TESTNET, 15% auf Mainnet

Begründung: Mainnet niedriger weil Mainnet-Verluste real sind.

Auswirkung: - `02_risk_proposal_engine.md` volatility_kill_threshold-Param - `03_state_machine.md` Volatility-Kill-Switch-Trigger - `07_mainnet_future.md` MN-SR-8

Status: `open`

DR-7 · Wallet-Signature-Konflikt bei Policy-Wechsel

Frage: Wie verhindern wir, dass `wallet_signature.account_hash` beim Policy-Wechsel zwischen `tradable_quote` und `frozen/managed` bricht und damit Bot-Restart crasht?

Default-Vorschlag: C + D Hybrid — neue **G-DR-14 (Hard-Lock)** + **Two-File-Atomic-Pattern**.

Konkret: 1. **G-DR-14 NEU:** Policy-Wechsel ist nur erlaubt zwischen `{frozen, managed}`. Übergänge zu/von `tradable_quote` sind hard-blocked. 2. **Two-File-Atomic-Pattern** für approve_managed_proposal + release_managed_asset: Worker schreibt `managed_state.json` UND `baseline_holdings.json` in derselben DB-Transaction. 3. Test-Pinning: Boundary-Test G-DR-14, Two-File-Atomic-Test, signature-match-after-promote-Test.

Status: `decided` 2026-05-10 — Operator hat **C+D Hybrid** akzeptiert. **G-DR-14 in `99_master_boundaries.md` ergänzen.** Vollständige Begründung in `sessions/q_mh_session_2026-05-10_architecture_priming.md` §DR-7.

---

2 · Status-Übersicht

Q-MH	Frage-Topic	Status	Default verfügbar?
1	Proposal-TTL	open	✓ B (7d)
2	**Bot-Vorschlag-Modi**	**decided** (2026-05-10)	✓ C/B Hybrid env-abhängig
3	Operator-Override-Tiefe	open	✓ B
4	release_pending Behavior	open	✓ C
5	Multi-Asset-Bulk	open	✓ C
6	Bot self-init	open	✓ B
7	Drift-Schwellen	open	✓ qty>5%, price>2ATR, regime BEAR↔BULL
8	Pause-Verhalten	open	✓ A
9	Re-Engage nach exit	open	✓ A
10	synthetic_entry Method	open	✓ D
11	**Hard-Confirm-String**	**decided** (2026-05-10)	✓ B (`<asset>:<variant>`)
12	Cool-down	open	✓ D (24h/7d)
13	**Mainnet-Disable**	**decided** (2026-05-10)	✓ 3 Disables (aggressive/DCA/t2_pump_dump-recommended)
14	**Worker-Daemon required**	**decided** (2026-05-10)	✓ MH-0.5 vor MH-1
15	**managed_state SoT**	**decided** (2026-05-10)	✓ Hybrid C
16	Cooldown-Override	open	✓ B
17	Confidence-Threshold	open	✓ 0.50/0.65
18	Volatility-Kill	open	✓ 25%/15%
DR-7	**Wallet-Signature-Konflikt**	**decided** (2026-05-10)	✓ G-DR-14 + Two-File-Atomic

Stand: 6 entschieden (5× Q-MH + DR-7), 13 offen. Architektur-prägende Fragen sind durch — verbleibende 13 sind asynchron während Code-Phasen entscheidbar.

---

3 · Workflow für Q-MH-Beantwortung

Empfohlen: 1. Operator liest dieses File komplett 2. pro Frage: liest betroffene Phase-Files (Cross-Reference oben) 3. entscheidet pro Frage A/B/C/D oder Custom 4. Status `open → decided` mit Datum + Begründung 5. wenn alle 18 entschieden: Code-Phasen MH-1..7 können starten 6. spätere Änderungen: Q-MH-Antwort mit Versions-Bump + Audit-Trail

Asynchron-Workflow (empfohlen für komplexe Fragen): - Operator entscheidet ad-hoc beim Implementieren der Phase - Code-Phase markiert TBD-Stellen mit `# Q-MH-<N>: pending` Comment - nach Phase-Completion: Q-MH wird `decided` markiert + Begründung

---

4 · Cross-Reference

File	Betroffene Q-MH
`01_foundation.md`	Q-MH-15 (SoT)
`02_risk_proposal_engine.md`	Q-MH-2, 7, 10, 17, 18
`03_state_machine.md`	Q-MH-1, 4, 7, 8, 9, 12, 16, 18
`04_gui_operator_flow.md`	Q-MH-2, 3, 5, 11, 14
`05_commandbus_worker.md`	Q-MH-6, 11, 14, 15
`06_testnet_drill.md`	Q-MH-5 (Bulk im Drill?)
`07_mainnet_future.md`	Q-MH-2, 12, 13, 17, 18

— Ende Open Questions —

📄 `09_test_strategy.md`

09 · Test Strategy — Test-Plan, Boundary-AST, Drift-Tests

Status: architektur-defined **Phase-Key:** verteilt über alle MH-Phasen, Test-Surface pro Phase **Bezug:** zentrale Test-Übersicht für die gesamte MH-Sub-Roadmap

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, `00_overview.md`, alle anderen Phase-Files
Required previous phases	none — kann jederzeit referenziert werden
Restart required	no (Tests sind code-only)
Migration required	no
Mainnet-touch	no
Risk-Level	low (Doc-Layer), aber high wenn Tests fehlen / unzulänglich

Allowed

- pro Phase Test-Suite mit klarer Naming-Convention
- Boundary-AST-Tests (source-grep auf File-Level)
- Idempotency-Tests
- Drift-Detection-Tests
- Restart-Tests (post-Bot-Restart Boundary-Verify)
- Integration-Tests (PHP + Bot via DB-Cache)
- GUI-Tests (Filament Browser-Tests)
- Worker-Handler-Tests (mock-Exchange, mock-DB)
- Drill-Verify-Tests als End-to-End-Suite

NOT Allowed

- Tests die auf Mainnet-Connection ausgehen
- Tests die Bot-State live mutieren (Tests müssen tempdir / mock nutzen)
- Tests ohne psycopg2-Stub (host environment kann ohne pg laufen)
- Tests die andere bestehende Suites brechen (additiv, nicht regressiv)
- Tests die Worker-Daemon spawnen (`--once` Pattern verwendbar)
- Tests die Backups in Produktiv-Pfaden anlegen
- Tests die `.env` mutieren

Memory / Durable Rules

ID	Inhalt	Wo durchgesetzt
RECON-2.1-Pattern	Boundary-Tests via source-grep	strip-Docblock-Pattern aus G10_3_5_ApplyCommandTest
G-DR-11	Audit-Prefix-Separation	Tests prüfen `managed.*` vs `baseline.*`
G-DR-13	GUI ist read-mostly	Tests prüfen kein file_put_contents in Filament-Code
RECON-2.2a-Pattern	psycopg2-Stub für Bot-Tests	host-environment lauffähig
AST-Test-Pattern	siehe G10-CLEANUP-MINOR M2	invariants-Drift-Detection

Expected Test Surface

- **Total geschätzt: ~280 Tests** über alle MH-Phasen

- Bot-Side: ~165 Tests (~30+25+30+25+30+25 pro Phase)
- PHP-Side: ~115 Tests (~25+30+25+15+10+10 pro Phase)

---

1 · Test-Architektur (analog G10 / RECON-2.x)

Bot-Side (Python / unittest):

```
trading/tests/
  test_recon_mh_state_writer.py            # ~15 Tests
  test_recon_mh_proposal_engine.py         # ~30 Tests
  test_recon_mh_state_machine_guards.py    # ~25 Tests
  test_recon_mh_worker_handlers.py         # ~50 Tests (8 Handler × ~6)
  test_recon_mh_drift_detection.py         # ~15 Tests
  test_recon_mh_audit_snapshots.py         # ~10 Tests
  test_recon_mh_boundary.py                # ~10 Tests (source-grep)
  test_recon_mh_e2e_dryrun.py              # ~10 Tests
```

PHP-Side (Pest / PHPUnit):

```
gui/tests/Feature/
  Recon_MH_RegistryTest.php                 # ~20 Tests
  Recon_MH_ProposalServiceTest.php          # ~25 Tests
  Recon_MH_ManagedStateServiceTest.php      # ~20 Tests
  Recon_MH_FilamentManagedHoldingsTest.php  # ~25 Tests
  Recon_MH_WizardFlowTest.php               # ~15 Tests
  Recon_MH_AuditTimelineTest.php            # ~10 Tests
```

---

2 · Test-Kategorien

2.1 Boundary-AST-Tests (durable Schutz vor späteren Drift)

Pattern aus G10-CLEANUP-MINOR M2 + RECON-2.3:

```
def test_proposal_engine_no_command_creation(self):
    """proposal_engine.py darf NIE in commands-Tabelle inserten."""
    src = (TRADING_DIR / "proposal_engine.py").read_text()
    src_no_docstrings = strip_docstrings(src)
    self.assertNotIn("INSERT INTO commands", src_no_docstrings)
    self.assertNotIn("Command::create", src_no_docstrings)

def test_proposal_engine_no_order_apis(self):
    """proposal_engine.py darf KEINE ccxt order-API-Aufrufe enthalten."""
    src = strip_docstrings((TRADING_DIR / "proposal_engine.py").read_text())
    for forbidden in ("create_market_buy_order", "create_market_sell_order", "create_order"):
        self.assertNotIn(forbidden, src)
```

Zu pinnende Boundaries: - proposal_engine: keine commands-INSERTs, keine order-APIs - managed_state_writer: keine ccxt-imports - managed_state_reader: keine file-mutation (write_text/unlink/rename) - ProposalService.php: kein file_put_contents auf state-files - ManagedHoldings.php Filament Page: kein direct ccxt - Worker-Handler: jeder Handler hat audit-Emit am Anfang + Ende - **G-DR-14 (decided 2026-05-10):** BaselineHoldingsAllowlist-Validator + Worker-Guard prüfen `tradable_quote ↔ frozen/managed`-Crossings. Boundary-Test: jeder Apply mit Crossing-Versuch wird reject'd mit `managed.policy_change_blocked_dr14`. - **Two-File-Atomic (decided 2026-05-10):** approve_managed_proposal + release_managed_asset Worker-Handler schreiben `managed_state.json` UND `baseline_holdings.json` in einer Transaction. Boundary-Test: Failure auf einem File → Backup-Restore beider + audit `managed.two_file_atomic_rollback`. - **JSON-SoT (Q-MH-15 decided 2026-05-10):** Source-grep-Test pinnt: Bot liest NIE aus DB-Cache (kein psycopg2-Read in Bot-Reader-Klassen außer ManagedStateReader read von JSON); GUI liest NIE aus JSON-File (kein file_get_contents auf state-files in PHP).

2.2 Idempotency-Tests

Pattern aus G10-2 + RECON-2.3:

```
def test_apply_managed_proposal_idempotent_on_proposal_id(self):
    """Zweiter approve auf gleiche proposal_id → kein zweiter Command."""
    cmd1 = service.createApproveCommand(proposal_id, user_id, ...)
    cmd2 = service.createApproveCommand(proposal_id, user_id, ...)
    self.assertEqual(cmd1.id, cmd2.id)
```

Zu pinnende Idempotencies: - approve_managed_proposal: gleicher proposal_id → gleicher Command - reject_managed_proposal: gleicher proposal_id → gleicher Command - approve nach reject (oder umgekehrt): zweiter `precondition_rejected` - pause/resume: distinct keys (kein dedup, multiple möglich) - request_managed_proposal: `mh:propose:<asset>:<YmdHis>` verhindert Doppel-Klick innerhalb 1 Sekunde

2.3 State-Machine-Tests (DR-11 enforcement)

```
def test_dr11_bot_cannot_demote_managed_to_frozen(self):
    """Bot-actor wird abgelehnt für managed_* → frozen Übergang."""
    guard = ManagedStateGuard()
    result = guard.can_transition(
        from_state="managed_active", to_state="frozen",
        actor=Actor(type="bot"),
        payload={...}
    )
    self.assertFalse(result.ok)
    self.assertIn("dr11", result.reject_reason)

def test_user_can_demote_managed_to_frozen(self):
    """user_*-actor ist erlaubt."""
    result = guard.can_transition(
        from_state="managed_active", to_state="release_pending",
        actor=Actor(type="user", id=42),
        payload={...}
    )
    self.assertTrue(result.ok)
```

Zu pinnende Übergänge: - alle 18 Übergänge aus `03_state_machine.md` §2 - pro Übergang: erlaubte Actor + verbotene Actor

2.4 Drift-Detection-Tests (SM-1)

```
def test_qty_drift_triggers_drift_alert(self):
    """qty drift > 5% → state managed_drift_alert"""
    setup_managed_active(asset="SOL", state_qty=5.0)
    mock_real_qty(asset="SOL", real_qty=4.6)  # 8% drift
    detect_drift_cycle()
    snap = managed_reader.snapshot()
    self.assertEqual(snap.managed_assets["SOL"]["state"], "managed_drift_alert")

def test_qty_drift_below_threshold_no_alert(self):
    """qty drift < 5% → kein alert"""
    setup_managed_active(asset="SOL", state_qty=5.0)
    mock_real_qty(asset="SOL", real_qty=4.85)  # 3% drift
    detect_drift_cycle()
    snap = managed_reader.snapshot()
    self.assertEqual(snap.managed_assets["SOL"]["state"], "managed_active")
```

Zu pinnende Drifts: - qty drift Schwelle (Q-MH-7 — default 5%) - price drift Schwelle (default 2 ATR) - regime drift (BEAR ↔ BULL) - alle 3 zusammen - kein autonomer Sell bei drift (G-SR-7)

2.5 Worker-Handler-Tests (analog RECON-2.3c Pattern)

```
def test_handle_approve_managed_proposal_success_path(self):
    """approve creates managed_active state + audit + snapshot"""
    setup_proposal_in_db(proposal_id, state="risk_proposed")
    cmd = make_command("approve_managed_proposal", payload={...})

    result = worker._handle_approve_managed_proposal(cmd)

    self.assertTrue(result["ok"])
    self.assertEqual(result["state"], "managed_active")

    # audit
    audits = [e for _, e, _ in worker._captured_audit]
    self.assertIn("managed.asset_promoted", audits)
    self.assertIn("managed.synthetic_entry_set", audits)

    # snapshot
    self.assertTrue(snapshot_dir / f"<event_id>.json".exists())
```

Pro Handler ~6-7 Tests: - success path - precondition_rejected (z.B. proposal already decided) - mainnet_blocked - DR-11 enforcement (für approve/reject/pause/resume/release) - audit snapshot file written - DB-Cache update transactional - result_json shape

2.6 Restart-Tests (Bot-Side)

```
def test_after_restart_bootstrap_imports_managed_only(self):
    """Bot-Restart triggert auto_import NUR für managed_active assets,
       nicht für proposal_pending / risk_proposed / managed_paused / drift_alert."""
    write_managed_state({
        "SOL":  {"state": "managed_active",   "synthetic_entry": 145.0},
        "ETH":  {"state": "managed_paused",   ...},
        "DOGE": {"state": "proposal_pending", ...},
    })
    bootstrap_baseline(trader, ...)
    self.assertIn("SOL/USDT", trader.state["positions"])
    self.assertNotIn("ETH/USDT", trader.state["positions"])
    self.assertNotIn("DOGE/USDT", trader.state["positions"])
```

Zu pinnende Restart-Effekte: - nur managed_active Auto-Import - pause/drift/exit_executed ignoriert beim Bootstrap - Bot mismatch der wallet-signature → sys.exit(1)

2.7 GUI-Tests (Filament v4)

```
public function test_admin_can_open_managed_holdings_page(): void
{
    $this->actingAs(User::factory()->admin()->create());
    Livewire::test(ManagedHoldings::class)
        ->assertSuccessful();
}

public function test_operator_cannot_access_admin_actions(): void
{
    $this->actingAs(User::factory()->operator()->create());
    Livewire::test(ManagedHoldings::class)
        ->assertActionDoesNotExist('promote'); // nur view-only
}

public function test_wizard_step_5_hard_confirm_validates_dynamic_string(): void
{
    Livewire::test(ManagedHoldings::class)
        ->fillForm(['hard_confirm' => 'SOL:wrong'])
        ->call('submit')
        ->assertHasFormErrors(['hard_confirm']);

    Livewire::test(ManagedHoldings::class)
        ->fillForm(['hard_confirm' => 'SOL:recommended'])
        ->call('submit')
        ->assertSuccessful();
}
```

2.8 Audit-Snapshot-Tests (SM-6)

```
def test_audit_snapshot_file_immutable(self):
    """Snapshot-File ist immutable — kein write nach Erstellung"""
    snap_path = audit_snapshots_dir / f"{event_id}.json"
    self.assertTrue(snap_path.is_file())
    original_content = snap_path.read_text()

    # try to overwrite
    with self.assertRaises(FileExistsError):
        write_audit_snapshot(event_id, ...)

    # content unchanged
    self.assertEqual(snap_path.read_text(), original_content)
```

---

3 · Tests pro Phase

Phase	Bot-Tests	PHP-Tests	Boundary	Total
MH-1 (Registry)	0	~25	5	~30
MH-2 (Reader)	~25	0	5	~30
MH-3 (proposal_engine)	~30	0	5	~35
MH-4 (Services + Migration)	0	~30	5	~35
MH-5 (Filament UI)	0	~25	5	~30
MH-6 (Worker-Handler)	~50	0	5	~55
MH-7 (Bot-Wiring)	~25	0	5	~30
MH-8 (Drill)	0 (manual)	0 (manual)	0	manual
MH-9 (Daemon)	~10	~10	5	~25
Total	**~165**	**~115**	**~40**	**~320**

(zzgl. ~50 indirekte Regressionstests in bestehenden Suites die durch additive Erweiterung weiter laufen)

---

4 · Test-Lauf-Strategie

Pro Phase: 1. Phase-spezifische neue Tests laufen lassen (alle grün) 2. RECON-Test-Suite Regression: `python -m unittest tests.test_recon_*` — keine Failures 3. Vollsuite Bot: `cd trading && python -m unittest discover -s tests` — keine neuen Failures 4. Vollsuite PHP: `php artisan test` — keine neuen Failures 5. Phase-Closure-Snapshot der Test-Counts in Memory-Status-File

Pro Bot-Restart (MH-7 + MH-9): - Pre-Restart: Test-Counts dokumentieren - Post-Restart: Test-Counts vergleichen (sollten gleich sein) - post-Cycle-Verify (4 Cycles ohne Tracebacks)

Pro Drill (MH-8): - automatisierte Test-Suite-Run vorher - Drill ist manueller Smoketest, keine automatisierten Tests in Drill - post-Drill Test-Suite-Run nochmal als Regression

---

5 · CI/CD-Integration

Existing-Pattern: - `gui/phpunit.xml` listet alle PHP-Tests - `trading/tests/` discoverable via unittest - daily_bugswarm.py drone führt regression checks - pre-commit hooks (optional) prüfen no-broken-tests

MH-spezifisch: - Test-File Naming `test_recon_mh_*.py` und `Recon_MH_*Test.php` für discoverbarkeit - Boundary-Tests laufen bei jedem commit (keine separate Phase nötig) - Drift-Tests laufen täglich via Cron (BACKLOG, separate Phase)

---

6 · Bestehende Suites die NICHT brechen dürfen

Suite	Begründung
`test_recon_2_1_baseline.py`	Foundation-Schemas; MH ist additiv
`test_recon_2_2a_baseline_aware.py`	Bot-Side wiring; MH erweitert nicht überschreibt
`test_recon_2_3c_baseline_handlers.py`	Worker-Handler bleibt unangetastet, MH-Handler additive
`test_g10_*`	G10-Pattern; MH adoptiert Pattern, ändert nicht
`test_n7_2_*`	N7.2 Stablecoin/Peg-Block bleibt aktiv
`Recon_2_3_BaselineHoldingsTest.php`	RECON-2.3 PHP-Side; MH additive
`G10_*ApplyCommandTest.php`	G10-Apply-Pattern bleibt
`G6.5 Etappe Tests`	CommandBus invariant

→ vor jedem Phase-commit: full suite läuft. Bei rot: rollback bevor commit.

---

7 · Phase-Sequenz für 09_test_strategy

Tests sind nicht eigene Phase, sondern Teil jeder Code-Phase: - MH-1 enthält Registry-Tests - MH-2 enthält Reader-Tests - usw.

Aber: Boundary-AST-Tests + Drift-Tests + Audit-Snapshot-Tests können **eigene Phase** sein wenn sie zu komplex werden: - MH-9.1: Drift-Test-Suite (separate Phase nach MH-9) - MH-9.2: Audit-Snapshot-Test-Suite (separate Phase) - MH-9.3: full-Suite Boundary-AST-Test (separate Phase)

→ Tests sind durable. Mit jedem MH-Code wachsen sie.

— Ende Test Strategy —

📄 `10_backlog_future_extensions.md`

10 · Backlog · Future Extensions

Status: BACKLOG · alle Themen explizit aus aktueller MH-Roadmap ausgenommen **Phase-Key:** kein eigener — separate Phasen pro Thema, NACH MH-9 + RECON-2.5 **Bezug:** Themen die bewusst NICHT in MH-1..9 gehören, aber später wichtig werden

---

Phase Cross-Reference

Field	Value
Dependencies	`99_master_boundaries.md`, alle MH-Phasen abgeschlossen
Required previous phases	mind. RECON-MH-9 ☐
Restart required	tbd pro Phase
Migration required	tbd pro Phase (vermutlich ja)
Mainnet-touch	tbd pro Phase
Risk-Level	high (jede Phase hat eigene Risk-Profile)

Allowed (in dieser Datei selbst)

- BACKLOG-Themen sammeln + grob skizzieren
- Cross-Reference zu existing Phasen (G10, T-SPLIT, CAP-1, etc.)
- offene Architektur-Fragen pro Backlog-Thema
- Prioritäts-Hinweise

NOT Allowed

- Detail-Architektur pro Backlog-Thema (gehört in eigene Phase-Doc wenn aktiviert)
- Code-Skizzen
- Test-Pläne
- Implementierungs-Reihenfolgen

Memory / Durable Rules

Diese Datei selbst löst keine Mutation aus, aber jede aktivierte Backlog-Phase MUSS folgende durable rules anwenden:

ID	Inhalt	Wo durchgesetzt (sobald Backlog-Phase aktiviert)
G-DR-1	Frozen-by-default	jede Backlog-Phase die State-Files berührt
G-DR-2	Operator final authority	CAP-1-Compound, Auto-Hedging, Re-Engage-Pfade — kein autonomous-Promote
G-DR-3	CommandBus-only Mutation	Tax-Layer-Trigger, Restore-from-Snapshot, Mobile-Notifications — alle Mutationen über Bus
G-DR-4	TESTNET-first	jede Backlog-Phase entwickelt + drilled auf TESTNET zuerst
G-DR-7	Multi-Step-Wizard	Mobile-UX bekommt Mobile-Wizard (kein Mobile-One-Click)
G-DR-9	Bot autonom managed→frozen verboten	Auto-Hedging-Backlog-Phase darf NICHT auto-demoten
G-DR-10	Risk-Proposals versioniert	Risk-Engine V2 ML bekommt eigenen `risk_model_version="phase2-ml-v1"` String
G-DR-11	Audit-Prefix-Separation	jede neue Backlog-Phase bekommt eigenen audit-prefix (z.B. `tax.*`, `mobile.*`, `restore.*`) — strict separation
G-DR-12	Wallet-Signature excludes tradable_quote	bei Multi-wallet-Erweiterung: Signature pro Wallet getrennt
G-SR-7	Drift-Detection autonomous-Sell verboten	Auto-Hedging-Phase darf NICHT auf Drift autonom hedgen
G-SR-11	Mainnet-Block 5-Layer	jede Backlog-Phase die Mainnet anfasst MUSS RECON-2.5 Sign-Off durchlaufen
feedback_testnet_permanent	TESTNET bleibt parallel zu Mainnet	Multi-wallet, Tax-Layer, ML-Engine — TESTNET stays
feedback_backup_before_live_actions	Backup-Pflicht	Tax-Layer-Migration, Restore-Phase, ML-Model-Deploy alle Backup-pflichtig

Zusätzliche durable rules die pro Backlog-Phase ergänzt werden müssen:

- **B-DR-1 (CAP-1):** managed_state.max_allocation ist Soft-Cap, CAP-1.investable_capital ist Hard-Cap. Sizer kapppt bei investable_capital + audit `managed.allocation_capped`.
- **B-DR-2 (Tax-Layer):** Tax-Lots sind immutable nach exit_executed. Audit-Snapshot-Files (SM-6) sind Tax-Beweis.
- **B-DR-3 (Multi-wallet):** managed_state.json pro Wallet getrennt. account_hash NIE über Wallets gemischt.
- **B-DR-4 (Restore-Phase):** Restore mutiert nur State-Files, NIE state['positions'] oder Wallet. Audit-Event `managed.state_restored_from_snapshot` Pflicht.
- **B-DR-5 (ML-Engine V2):** Explainability-Pflicht — jeder ML-Output enthält Top-N Feature-Importance, sonst proposal_aborted.

Expected Test Surface

Diese Datei selbst: 0 Tests (reine Backlog-Übersicht, keine Implementierung).

Aber pro aktivierter Backlog-Phase (sobald aus BACKLOG → aktiv): eigene Test-Suite analog `09_test_strategy.md`-Pattern. Schätzung pro Backlog-Phase:

Backlog-Phase	Geschätzte Tests	Test-Typen	Bestehende Suites die NICHT brechen dürfen
1. CAP-1 Integration	~50	Sizer-Capping-Tests, allocation_capped-Audit-Tests, investable_capital-Berechnung, Soft/Hard-Cap-Boundary, Cycle-Recompute	`test_recon_mh_*`, CAP-1-Preflight-Tests, Sizer-Tests
2. T-SPLIT-5/6 Integration	~30	strategy_group-Erweiterung, t3-Aktivierung, proposal_engine-Allowlist-Update	`test_recon_mh_proposal_engine`, T-SPLIT-Tests
3. Multi-wallet	~80	per-wallet managed_state, signature-pro-wallet, exchange-Routing, race-conditions zwischen Wallets	gesamte MH-Suite (Boundary für single-wallet bleibt)
4. Copy-Trading auf Mainnet (T-SPLIT-3)	~60	externe Wallet-Trades-Mirror, Konflikt managed-vs-copy, t3_copy_trading-Aktivierung	`test_recon_mh_state_machine`, T-SPLIT-3-Tests
5. Exchange-Abstraction	~40	ccxt-Adapter-Tests pro Exchange, Symbol-Mapping, Order-Format-Differences	gesamte Bot-Suite
6. Tax-Layer	~50	FIFO/LIFO/HIFO Tax-Lot-Tracking, Audit-Snapshot-as-Tax-Proof, CSV-Export, Steuer-API-Integration	`test_recon_mh_audit_snapshots`, B-FEE-FIX-Suite
7. Profit-Reinvestment (CAP-1+)	~30	Auto-Re-Engage bei positiver PnL, Operator-Override-Pfad, cap_1_compound_mode	CAP-1-Tests, MH-State-Machine-Tests
8. ML-basierte Risk-Engine V2	~60	Model-Output-Schema, Determinismus-Replacement (ML ist non-deterministic), Explainability, Model-Versioning, A/B-Testing-Pfad	`test_recon_mh_proposal_engine` (parallel laufend)
9. Mobile UX	~25	Touch-Target ≥44px, Filament-Mobile-Layout, Wizard-Mobile-Tauglichkeit, Latency-Tests	GUI-Suite
10. Notifications	~20	Telegram-Push, Email, Mute/Snooze, Escalation-Logik	Reporter-Tests
11. Restore-from-Snapshot	~30	Snapshot-Read, State-File-Restore, Audit `state_restored_from_snapshot`, NIE Position-Mutation	`test_recon_mh_audit_snapshots`, MH-State-Machine-Tests
12. Performance-Monitoring	~20	Aggregations-Job, Cold-Storage, Export	`test_recon_mh_*` (alle für Metriken-Quellen)
13. Korrelations-V2	~25	Cluster-Analyse, Concentration-Cluster-Warning, Wizard Step 4 Erweiterung	`test_recon_mh_proposal_engine`
14. Auto-Hedging	(out-of-scope für Operator-Final-Authority — keine Test-Schätzung)	–	–

Gesamt-Test-Schätzung Backlog (alle Phasen kumuliert): ~520 Tests zusätzlich zur Kern-MH-Suite (~280 aus `09_test_strategy.md`).

Test-Strategie pro Backlog-Phase (durable): - Boundary-AST-Tests pro Phase analog SM-6-Pinning - Idempotency-Tests pflichtbestandteil - Restart-Tests bei Bot-Side-Phasen - Drift-Tests bei Multi-wallet, Auto-Hedging, Re-Engage - bestehende `test_recon_mh_*` Suite + `test_recon_2_*` Suite + G10-Suite + T-SPLIT-Suite dürfen NICHT regressiv sein - pro Phase eigene Test-File-Naming `test_<phase_short_name>_*.py` und `<PhaseShortName>Test.php` - CI/CD-Integration analog 09_test_strategy.md §5

---

1 · CAP-1 Integration (Profit-Reinvestment / Profit-Reserve)

Was: Operator entscheidet Mainnet=reserve_profits, TESTNET=compound. Bot's `investable_capital` ist NICHT identisch zu `total_value` oder `cash`.

Bezug zur MH-Sub-Roadmap: - managed_state.max_allocation_usdt ist **Soft-Cap** - CAP-1.investable_capital ist **Hard-Cap** - Sizer kapppt bei investable_capital + Audit `managed.allocation_capped`

Offene Fragen: - wie wird `investable_capital` per-cycle aktualisiert? - wie reagiert managed-Allocation bei investable_capital-Drop? - was wenn managed Asset überschreitet investable_capital nach Promote (Markt-Bewegung)?

Priorität: mittel — wird relevant nach MH-7 wenn echtes managed-Trading auf TESTNET läuft.

Memory-Cross-Reference: `cap_1_preflight_backlog.md` (Memory) hat 6 Korrekturen K1-K6.

Empfohlene Phase-Bezeichnung: `RECON-MH-CAP-1-INTEGRATION` (eigene Sub-Phase NACH MH-9)

---

2 · T-SPLIT-5 / T-SPLIT-6 Integration

Was: T-SPLIT-3 hat strategy_groups durable definiert (`t1_core / t2_pump_dump / t3_copy_trading / legacy_unknown`). t3 ist heute „planned · disabled".

Bezug zur MH-Sub-Roadmap: - proposal_engine darf KEINE t3-recommendation ausgeben (G-SR-14) - managed_state.strategy_group ist auf {t1_core, t2_pump_dump} beschränkt - T-SPLIT-5 könnte t3 aktivieren — dann müssen wir managed-state-Validatoren erweitern

Offene Fragen: - wann wird t3 aktiviert? - gilt für managed Assets eine andere strategy_group-Logik als für autonom-eröffnete? - T-SPLIT-6 könnte neue strategy_groups hinzufügen (z.B. t4_arbitrage)

Priorität: niedrig — t3 ist BACKLOG, T-SPLIT-5/6 weiter BACKLOG.

Empfohlene Phase-Bezeichnung: `RECON-MH-TSPLIT-EXTENSION` (eigene Sub-Phase)

---

3 · Multi-wallet (mehrere Mainnet-Konten gleichzeitig)

Was: Operator hat mehrere Binance-Konten oder mehrere Exchange-Konten gleichzeitig.

Bezug zur MH-Sub-Roadmap: - baseline_holdings.json ist heute single-exchange-single-wallet - managed_state.json ist heute single-wallet - wallet_signature ist heute single-account-hash

Architektur-Fragen: - ein managed_state pro wallet (multi-file)? - pro wallet eigene baseline? - exchange-Abstraction Layer nötig?

Priorität: niedrig — single-wallet reicht für aktuelles Vorhaben.

Empfohlene Phase-Bezeichnung: `RECON-MH-MULTIWALLET` (separate große Phase, vermutlich Jahre weg)

---

4 · Copy-Trading auf Mainnet

Was: T-SPLIT-3 t3_copy_trading-strategy_group für copy-Trading von externen Wallets.

Bezug zur MH-Sub-Roadmap: - copy-Trading ist NICHT managed-Holdings (anderer Lifecycle) - aber: copy-Trading-Position könnte aus „Operator-managed" upgrade-Pfad bekommen - Datenmodell-Konflikt: externe Wallet-Trades vs eigene managed-state

Architektur-Fragen: - copy-Trading hat keinen Operator-Approval-Flow (fully autonomous) - managed-Holdings hat Operator-Approval (semi-discretionary) - diese zwei Modelle KOLLIDIEREN potentiell wenn beide für gleiches Asset aktiv

Priorität: niedrig — t3 ist BACKLOG.

Empfohlene Phase-Bezeichnung: `T-SPLIT-3-IMPLEMENTATION` (T-SPLIT-Roadmap, nicht MH)

---

5 · Exchange-Abstraction (Coinbase, Kraken, etc.)

Was: Bot supportet heute nur Binance. Andere Exchanges könnten managed-Holdings haben.

Bezug zur MH-Sub-Roadmap: - baseline_holdings.json hat `exchange: 'binance'` hardcoded in wallet_signature - proposal_engine.market_context fetcht via ccxt — exchange-agnostic theoretisch - managed-Lifecycle braucht exchange-spezifische SL/TP-Mechanik

Architektur-Fragen: - ccxt unified API reicht? oder Exchange-spezifische Pfade? - managed-Asset über Exchange-Grenzen hinweg (rare)?

Priorität: niedrig — Binance reicht.

Empfohlene Phase-Bezeichnung: `EXCHANGE-ABSTRACTION` (separate Roadmap)

---

6 · Tax-Layer für Mainnet-Trades

Was: Mainnet-Trades haben steuerliche Konsequenzen. Operator muss bilanzieren.

Bezug zur MH-Sub-Roadmap: - managed.exit_executed schreibt PnL — kann in Tax-Reports einfließen - managed.synthetic_entry vs cost_basis: wichtig für Steuerlich relevante Berechnung - audit_snapshot files könnten Steuer-Beweise sein

Architektur-Fragen: - FIFO / LIFO / HIFO für Position-Tracking? - pro Promote/Release Tax-Lot-Tracking? - Export nach CSV / Steuer-API?

Priorität: mittel — relevant SOBALD Mainnet aktiv ist.

Empfohlene Phase-Bezeichnung: `TAX-LAYER` (separate Phase, parallel zu Mainnet-Aktivierung)

---

7 · Profit-Reinvestment-Strategie (CAP-1 erweitert)

Was: automatisches Re-Compound von Profits in managed-Lifecycle.

Bezug zur MH-Sub-Roadmap: - nach `managed.exit_executed` mit positivem PnL → option re-engage same asset - Re-Engage könnte Auto-Trigger via `cap_1_compound_mode` werden - aber: G-DR-2 sagt Operator final authority

Architektur-Fragen: - darf Bot autonom re-engagen wenn `cap_1_compound_mode=auto`? - oder bleibt Operator-Confirm zwingend (default)?

Priorität: niedrig — sehr späte Phase, vermutlich weit nach Mainnet-Stable.

Empfohlene Phase-Bezeichnung: `CAP-1-COMPOUND-MANAGED` (kombiniert)

---

8 · ML-basierte Risk-Engine

Was: proposal_engine v2 könnte ML-Modelle für Confidence-Score / Strategy-Empfehlung nutzen.

Bezug zur MH-Sub-Roadmap: - aktuelle Engine ist deterministisch (Q-MH-2-Phase1) - v2 mit ML würde `risk_model_version="phase2-ml-v1"` bekommen - Versions-System (G-DR-10) ist darauf vorbereitet

Architektur-Fragen: - welches Modell? (klassisch ML / Transformer / RL) - Training auf welchen Daten? - Latency vs Accuracy Trade-off? - Explainability (Operator muss verstehen warum Bot empfiehlt)

Priorität: niedrig — Phase 1 ist Heuristik, ML ist Phase 2+.

Empfohlene Phase-Bezeichnung: `RISK-ENGINE-V2-ML` (separate Roadmap, vermutlich nach 1+ Jahr Live-Daten)

---

9 · Mobile UX

Was: Operator kann mobile auf managed-Holdings reagieren.

Bezug zur MH-Sub-Roadmap: - ManagedHoldings Filament Page ist heute Desktop-only - Wizard hat 5 Steps — Mobile-tauglich? - Touch-Targets ≥44px

Architektur-Fragen: - Filament v4 hat Mobile-Layout (responsive) - aber: Wizard-Form-Handling auf Mobile nicht getestet - Operator-Klicks-Latenz im Mobile-Context kritisch

Priorität: mittel — relevant wenn Operator unterwegs ist.

Empfohlene Phase-Bezeichnung: `RECON-MH-MOBILE` (eigene Phase nach MH-Stable)

---

10 · Operator-Notification-Channels

Was: Telegram + Email + Push-Notifications für managed-Lifecycle-Events.

Bezug zur MH-Sub-Roadmap: - Reporter (Bot-Side) hat heute Telegram-Channel für Trade-Events - managed.drift_detected sollte sofort an Operator gepusht werden (Mobile + Desktop) - managed.exit_executed sollte gepusht werden - proposal_pending → risk_proposed Transition pushed?

Architektur-Fragen: - welche Events sind „push-worthy"? - mute-/snooze-Mechanismen? - escalation bei keiner Operator-Reaktion auf drift_alert?

Priorität: mittel — relevant für sichere Mainnet-Operation.

Empfohlene Phase-Bezeichnung: `MH-NOTIFICATIONS` (kann mit MH-9 kombiniert werden)

---

11 · Restore-from-Snapshot Phase

Was: Operator kann via dedizierter Phase einen managed_state.json-Snapshot zurückrollen.

Bezug zur MH-Sub-Roadmap: - SM-6 erzeugt audit_snapshot files pro State-Übergang - Restore-Path ist BACKLOG (nicht in MH-1..9) - ABER: Restore-Path ist sinnvoll für Test/Drill/Notfall

Architektur-Fragen: - Restore mutiert nur State-Files, nicht state['positions'] (DR-2) - Operator-Approval verbindlich - Restore erzeugt selbst audit_event `managed.state_restored_from_snapshot`

Priorität: niedrig — Notfall-Mechanismus, nicht alltäglich.

Empfohlene Phase-Bezeichnung: `RECON-MH-RESTORE` (separate Phase)

---

12 · Performance-Monitoring + Metrics-Dashboard

Was: managed-Lifecycle-Metriken in einem zentralen Dashboard.

Bezug zur MH-Sub-Roadmap: - KPIs: managed-PnL, drift-Rate, exit-Rate, average-time-to-promote - Dashboard analog `MonitoringDashboard.php` (existing) - Charts via Filament Widgets

Architektur-Fragen: - Aggregations-Job pro Tag / pro Stunde? - Cold-storage für historische Metriken? - Export (CSV / JSON / API)?

Priorität: mittel — wird wichtig je mehr managed Assets aktiv sind.

Empfohlene Phase-Bezeichnung: `MH-METRICS-DASHBOARD` (kann mit MH-9 kombiniert werden)

---

13 · Korrelations-/Concentration-Detection

Was: advanced Berechnung von Portfolio-Korrelationen vor Promote.

Bezug zur MH-Sub-Roadmap: - aktuelle proposal_engine berechnet einfache Korrelationen (rho) - v2 könnte Cluster-Analyse machen (z.B. Layer-1-Cluster, DeFi-Cluster) - Wizard Step 4 zeigt Cluster-Concentration-Warnings

Priorität: niedrig — Phase 1 ist single-asset.

Empfohlene Phase-Bezeichnung: `RISK-ENGINE-CORRELATION-V2` (mit ML-Engine kombinierbar)

---

14 · Auto-Hedging

Was: wenn managed long Asset und Markt-Regime kippt → automatischer Hedge.

Bezug zur MH-Sub-Roadmap: - KOLLIDIERT mit G-DR-2 (Operator final authority) - Hedge ist Trading-Action, nicht nur State-Mutation - nur via Operator-Approval (kein autonomous hedging)

Priorität: niedrig — komplexes Feature, nicht im Roadmap-Scope.

---

15 · Cross-Reference zu existing Memory

Memory-File	Bezug
`cap_1_preflight_backlog.md`	CAP-1 Integration
`t_split_status.md`	t3_copy_trading Hard-Block
`g10_runtime_config_operator_runbook.md`	G10-Apply-Pattern als Vorbild
`feedback_testnet_permanent_paper_rename.md`	TESTNET permanent durable
`feedback_backup_before_live_actions.md`	Backup-Pflicht
`recon_managed_holdings_architecture.md`	monolithisches Quell-Doc dieser Sub-Roadmap

---

16 · Priorisierung-Empfehlung

Wenn alle MH-1..9 erfolgreich abgeschlossen + RECON-2.5 Mainnet-Readiness:

Priorität	Phase
1 (sofort danach)	CAP-1 Integration (mittel)
2	MH-Notifications + Metrics-Dashboard (mittel)
3	Mobile UX (mittel)
4	Tax-Layer (mittel — wenn Mainnet aktiv)
5	Restore-from-Snapshot (niedrig)
6	T-SPLIT-5/6 Integration (niedrig — wartet auf T-SPLIT-Phase)
7	Risk-Engine V2 ML (sehr niedrig — Phase 2+)
8	Multi-wallet (niedrig — Jahre weg)
9	Exchange-Abstraction (niedrig)
10	Auto-Hedging (out of scope für Operator-Final-Authority)

---

17 · Was hier NICHT steht

- Konkrete Implementierungs-Schritte pro Backlog-Phase
- Code-Skizzen
- Test-Pläne pro Backlog-Phase
- Operator-Antworten

→ wenn ein Backlog-Item aktiviert wird: eigene Architecture-Phase wie diese (eigenes `.md` unter eigener Sub-Roadmap), eigener Reviewer-Bewertung, eigene Phasen-Splittung.

— Ende Backlog Future Extensions —

📄 `sessions/q_mh_session_2026-05-10_architecture_priming.md`

Q-MH-Antwort-Session — 5 architektur-prägende Fragen + DR-7-Konflikt

Datum: 2026-05-10 **Phase:** Pre-MH-1, Operator-Q&A-Vorbereitung **Status:** Vorlage zur Entscheidung — keine Antworten verbindlich bis Operator pro Frage `decided` markiert **Boundaries:** keine Implementierung, kein Code, keine Migration, kein Bot-Restart, kein Worker, kein Mainnet, kein Push

Format pro Frage: **Problem · Optionen · Empfehlung · Risiko · Phase-Auswirkungen · Default**. Eintragung in `08_open_questions.md` und betroffene Phase-Files erst nach Operator-Entscheidung.

Q-MH-14 · Worker-Daemon als Vorbedingung?

1. Problem

Aktueller Worker (`command_worker.py`) wird **manuell** als `--once` gestartet. Operator klickt Pause auf managed_active Asset → Command landet in `commands` (status=pending) → bleibt dort bis jemand `docker exec ... command_worker.py --once` aufruft. Wenn Bot-Cycle parallel SL/TP triggert, sieht Bot den noch alten state (`managed_active`) statt des gewünschten `managed_paused` → **R4-Race aus Architektur-Doc**: Bot verkauft trotz Pause-Klick.

2. Optionen

A · Optional bleiben (status quo) Operator triggert manuell `--once`. R4-Race bleibt akzeptiert.

B · Daemon als Vorbedingung (Pre-Req für MH-1) Vor MH-1 muss `clawbot-worker` als Daemon-Compose-Service laufen. Worker pollt `commands` jede X Sekunden (z.B. 5s) → Pause/Resume/Release synchron verarbeitet.

C · Hybrid mit Latenz-Toleranz Daemon erst ab MH-7 (Bot-Side-Wiring); MH-1..6 bleiben mit `--once`. Akzeptiert R4-Race nur bis MH-7.

3. Empfehlung — B

- managed-Lifecycle braucht **synchrones Verarbeiten** von Pause/Resume — sonst ist UX-2 Multi-Step-Wizard sinnlos (Operator klickt Pause, Bot ignoriert)
- `flag_managed_drift` (Bot-self-emit) braucht Daemon, sonst hängt Drift-Alert in pending bis manueller Worker-Run
- Daemon-Aktivierung ist eigene kleine Phase (G10-1 hat den Pfad bereits vorbereitet via `docker compose --profile workers`)

4. Risiko

- **A:** akzeptiert dauerhaften R4-Race → Operator-Erwartung weicht von Bot-Verhalten ab → Vertrauensverlust + potenzielle Verluste bei live-nahen Mainnet-Phasen.
- **B:** Daemon-Container muss eigene Lifecycle-Patterns haben (Healthcheck, Restart-Policy, Graceful-Shutdown). Wenn Daemon abstürzt + Watchdog ihn nicht wiederbringt → Commands stecken fest (Bot weiß davon nichts).
- **C:** „nur halb" — MH-1..6 sind Foundation-Phasen ohne live-Pause/Resume-Use-Case → R4 unkritisch, aber Operator könnte versehentlich vor MH-7 GUI nutzen.

5. Auswirkungen auf spätere Phasen

- `05_commandbus_worker.md` **§6:** Daemon-Konzept fest verankern (statt „status quo + Backlog-Empfehlung")
- `00_overview.md`**:** MH-9 (Worker-Daemon-Aktivierung) wird zu MH-0.5 (vor MH-1) **oder** bleibt als MH-9 mit harter Vorbedingung-Klausel für MH-7
- `03_state_machine.md` **§4 Cooldown / §5 TTL:** TTL-Job + Drift-Detection brauchen Daemon → ohne Daemon = manueller Cron oder gar nicht
- `06_testnet_drill.md` **Sub-Phase B/C:** `--once` ersetzt durch live-Daemon → Drill-Schritte ändern sich

6. Default-Vorschlag

Option B mit kleinem Twist: Daemon-Aktivierung als „MH-0.5" zwischen Architektur-Closure und MH-1.

Heißt:

- Phase MH-0.5 = nur `docker compose --profile workers up -d clawbot-worker` + Healthcheck-Verify + 24h-Stabilitäts-Beobachtung
- klein (2-3 Stunden Operator-Zeit), reversibel (`docker compose down clawbot-worker`)
- löst R4-Race vor jeder Code-Phase
- bringt Cron-getriggerte TTL-Expiry + Drift-Detection später ohne Architektur-Drift

---

Q-MH-15 · managed_state Source of Truth — JSON oder DB?

1. Problem

Lifecycle-State eines managed Assets (z.B. SOL ist `managed_active`, synthetic_entry=145.32, stop_loss=130.0, history=[…]) muss **irgendwo** persistent liegen. Optionen sind nicht orthogonal: GUI braucht schnellen Read-Zugriff (DB-Index), Bot braucht atomic-write + sha256-verify (JSON-File), und der Reader muss „pro Cycle frisch" lesen ohne 50ms Latenz.

2. Optionen

A · JSON only (parity mit baseline_holdings.json) `managed_state.json` ist Single Source of Truth. GUI liest **direkt** aus dem File via Bot-API-Endpoint oder shared volume.

B · DB only (parity mit ConfigProfile) `managed_assets` / `managed_proposals` / `managed_assets_history` Tabellen. Bot liest via psycopg2-Query pro Cycle.

C · Hybrid: JSON ist SoT, DB ist Read-Cache Worker schreibt **transactional in beide** (JSON für Bot-Reader, DB für GUI-Read). Bot ignoriert DB. GUI ignoriert JSON.

3. Empfehlung — C

Begründung mit konkretem Repo-Bezug:

- Bot-Reader-Pattern (RECON-2.1/2.2a) ist **etabliert** auf JSON-File mit `snapshot()` pro Cycle (kein Cache, atomic-read). Weg von dem Pattern wäre teurer Bruch.
- GUI-Read aus File braucht entweder shared volume (verletzt Container-Isolation und G-DR-3 wenn versehentlich auch GUI-write erfolgt) oder API-Endpoint (zusätzliche Komponente, neue Failure-Surface).
- DB-Cache löst beides: Bot bleibt im JSON-Pattern, GUI nutzt DB-Indexes für Filament Tables.
- Worker schreibt in beide **transactional** (nicht: PHP schreibt in DB, Bot in JSON — das wäre wirklich Drift).

4. Risiko

- **A:** GUI muss File über Bot-API lesen → entweder neuer Endpoint (Bot wird zum HTTP-Server) oder shared volume mit read-only mount → Container-Isolation-Bruch.
- **B:** Bot muss psycopg2-Query pro Cycle → 1-5ms Latenz pro Asset → bei 50 managed Assets sind das spürbar 50-250ms je Cycle. Plus: DB-down → Bot-Cycle-Crash (während JSON nur „file absent" wäre).
- **C:** Worker muss JSON+DB transactional aktualisieren — wenn JSON-Write succeed aber DB-INSERT fail → Drift. Mitigation: G-DR-6 Backup-before-mutate + Worker-Retry-Logik. Plus: pro State-Change zwei Writes (JSON + DB-Cache) → leicht erhöhte Latenz, aber pro-Klick einmalig (nicht per-Cycle).

5. Auswirkungen auf spätere Phasen

- `01_foundation.md` **§4 (DB-Tabellen):** bleibt wie geplant (Cache-Tabellen)
- `05_commandbus_worker.md` **§3:** Worker-Handler muss **transactional double-write** haben (JSON-Writer + DB-INSERT in einer DB-Transaction; bei JSON-Failure: Rollback)
- `04_gui_operator_flow.md` **§8:** GUI-Live-Tabelle liest aus DB-Cache, polling 30s — sauber
- **Q-MH-15 selbst hat keine Phase-Reihenfolge-Auswirkung** — beide Schreib-Pfade landen im selben Worker-Handler

6. Default-Vorschlag

Option C — JSON ist SoT für Bot, DB ist Cache für GUI.

Konkrete Regel die in `99_master_boundaries.md §14 State-File-Policy` ergänzt wird:

> Bei managed_state-Änderungen: Worker schreibt zuerst JSON (atomic + sha256 + Backup), dann DB-Cache in selber DB-Transaction. Bei JSON-Write-Failure: keine DB-Mutation. Bei DB-INSERT-Failure nach erfolgreichem JSON-Write: Worker emittet `managed.cache_drift_detected` Audit + retry-Logik beim nächsten Cycle.

Q-MH-13 · Welche Features sind Mainnet-disabled?

1. Problem

Mainnet ist real money. proposal_engine könnte aggressive Strategien empfehlen (weite SL = große Verluste), DCA-Reinvest auf fallendem Markt empfehlen (Capital-Lock-In), oder t2_pump_dump-Strategy-Group für Pump-Coins empfehlen (extreme Volatility). Auf TESTNET ist all das harmlos. Auf Mainnet kann es viele Tausend Euro kosten.

2. Optionen

A · Nichts disabled — Engine läuft identisch auf Testnet + Mainnet. Operator entscheidet bei Approve, ob er der Empfehlung folgt.

B · Konservatives Set disabled — auf Mainnet:

- aggressive Variante in proposal_engine
- DCA-Recommendation
- t2_pump_dump als `recommended` strategy_group

C · Strict Mainnet-Lockdown — alle Features disabled, nur 1 Variante (recommended-conservative), nur t1_core Strategy-Group, kein DCA, kein Trailing-Stop.

3. Empfehlung — B mit drei Disables (siehe Architektur-Doc default)

Begründung:

- A ist verantwortungslos: Operator hat 18 Q-MH-Fragen + 14 G-DR-Regeln im Kopf, kann nicht zusätzlich pro Promote 5 weitere Risiko-Punkte abwägen → Hard-Gate ist sicherer
- C ist überprotektiv: würde managed-Holdings-Konzept aushöhlen (Operator wäre auf TESTNET frei, auf Mainnet kastriert → Frust + Drill-Result-Übertragbarkeit fragwürdig)
- B trifft die durable-Linie: Bot empfiehlt nur „balanced" Risk-Profile auf Mainnet, Operator kann mehr (via Operator-Override aus Q-MH-3) wenn er bewusst will

4. Risiko

- **A:** Bot generiert aggressive Variante, Operator klickt unbedacht durch Wizard → Mainnet-Verluste in 1-3% Range pro Tag möglich.
- **B:** Operator könnte sich von disabled aggressive variant „eingeengt" fühlen → könnte versuchen via Operator-Override (Q-MH-3) ähnliche Werte selbst eintragen → Override-Audit zeigt das, aber blockiert nicht.
- **C:** managed-Holdings auf Mainnet wird unterausgenutzt → Operator macht Trades manuell außerhalb des Systems → kein Audit, kein Risk-Tracking.

5. Auswirkungen auf spätere Phasen

- `02_risk_proposal_engine.md` **§3:** `if env=mainnet: alternative_proposals = [conservative, recommended]` (kein aggressive). DCA-Recommendation `dca_recommended=False` hardcoded auf Mainnet. t2_pump_dump-Vorschlag fallback to t1_core mit Warning.
- `07_mainnet_future.md` **MN-SR-1, MN-SR-2, MN-SR-3:** explizit benennen welche 3 Features disabled sind (statt vager „Mainnet-disable-Features").
- `04_gui_operator_flow.md` **Wizard Step 3:** Variant-Selector zeigt aggressive auf Mainnet als greyed-out mit Tooltip „Mainnet-policy: aggressive disabled".
- `08_open_questions.md` **Q-MH-13:** Status open → decided.

6. Default-Vorschlag

Option B mit exakt diesen 3 Disables auf Mainnet:

#	Feature	Mainnet	Testnet
1	aggressive Variante in proposal_engine	DISABLED	enabled
2	dca_recommended Output	hardcoded `false`	engine-decided
3	t2_pump_dump als recommended strategy_group	hard-fallback to t1_core mit warning	enabled

Operator-Override (via Q-MH-3) erlaubt das Override **nur für SL/TP/max_alloc/trailing_stop**, NIE für strategy_group oder variant-Switch — diese 3 Disables sind hart.

Q-MH-2 · Wie viele Risk-Proposal-Varianten generiert Bot?

1. Problem

Bot kann zu jedem Asset eine `recommended` Variante generieren plus 0-2 Alternativen (`conservative` und/oder `aggressive`). Das beeinflusst:

- Wizard-UX (3 Buttons vs 1 Button)
- Engine-Compute-Time (3× ATR-Multiplier-Pfade vs 1×)
- Operator-Cognitive-Load (3 Optionen vergleichen vs 1)
- Mainnet-Q-MH-13-Konflikt (aggressive auf Mainnet disabled — was zeigt Wizard?)

2. Optionen

A · Nur recommended (1 Variante) — schlankeste UX, Engine berechnet 1×.

B · recommended + conservative (2 Varianten) — Operator hat Risk-Wahl ohne aggressive-Falle. Mainnet + Testnet identisch.

C · recommended + conservative + aggressive (3 Varianten) — volle Bandbreite auf Testnet, aggressive auf Mainnet aus Q-MH-13 disabled (in v3 Wizard greyed-out).

3. Empfehlung — C auf Testnet, B auf Mainnet (= durable Default aus Architektur-Doc)

Begründung:

- C auf Testnet: Operator lernt Risk-Spektrum kennen, kann aggressive ausprobieren ohne reales Geld zu verlieren
- B auf Mainnet: aggressive ist Q-MH-13-disabled → konsistent mit MN-SR-1
- Engine-Code ist identisch (3 Berechnungspfade), nur das `alternative_proposals`-Feld wird env-abhängig gefiltert
- Wizard zeigt env-abhängig 3 oder 2 Cards

4. Risiko

- **A:** Operator kann nicht zwischen Risk-Profilen wählen → muss Operator-Override (Q-MH-3) für jede Anpassung nutzen → Override-Audit-Spam.
- **B:** identisch über Envs ist klar, aber Testnet-Operator hat aggressive nicht zur Hand → Drill-Tests können aggressive-Pfad nicht durchspielen.
- **C:** Inkonsistenz Testnet vs Mainnet — Drill-Sicherheit vs Mainnet-Sicherheit-Asymmetrie. Operator könnte aggressive Settings auf Testnet gewohnt sein, dann beim ersten Mainnet-Promote irritiert sein dass es nicht da ist.

5. Auswirkungen auf spätere Phasen

- `02_risk_proposal_engine.md` **§3:** Engine berechnet 3 Varianten, filtert env-abhängig im Output
- `04_gui_operator_flow.md` **§2 Wizard Step 3:** zeigt 2 oder 3 Cards je nach env
- `06_testnet_drill.md` **Sub-Phase B Step 11:** Drill-Pfad muss eine Variante wählen — Default `recommended`, optional aggressive nur auf Testnet

- `07_mainnet_future.md` **MN-SR-1:** aggressive disabled (siehe Q-MH-13)
- `08_open_questions.md` **Q-MH-2:** Status open → decided mit Verweis auf env-Differenzierung

6. Default-Vorschlag

Option C/B Hybrid: 3 Varianten auf Testnet, 2 (rec + cons) auf Mainnet.

Konkret: `proposal_engine` immer berechnet 3 Varianten intern. Output-Field `alternative_proposals` enthält:

- auf Testnet: `[conservative, aggressive]` (2 Alternativen + 1 recommended = 3 sichtbar)
- auf Mainnet: `[conservative]` (1 Alternative + 1 recommended = 2 sichtbar)

Q-MH-11 · Hard-Confirm-Pattern für approve

1. Problem

Hard-Confirm-Strings sollen Operator zwingen, bewusst zu tippen statt durchzuklicken. Bestehende Patterns im Repo:

- G10-6 Apply Profile: `confirm_profile_name === $record->name` (Profilname, z.B. `g10-3-test-profile`)
- RECON-2.3 Apply Baseline: `<count>:<sha8>` (z.B. `47:b25e09fb`)

Für approve_managed_proposal: was ist der String? Optionen unterscheiden sich in **Eindeutigkeit pro Action** (verhindert Muskel-Memory) und **Lesbarkeit** (Operator soll den String verstehen).

2. Optionen

A · Nur Asset-Symbol — `SOL`. Kurz, leicht zu tippen, aber Operator könnte 5 verschiedene SOL-Approves hintereinander machen ohne nachzudenken.

B · `<asset>:<variant>` — `SOL:recommended` / `SOL:conservative` / `SOL:aggressive`. Pro Klick variabel; Operator muss explizit Variante tippen.

C · `<asset>:<sha8>` (proposal_id-suffix) — `SOL:a3f9c7`. Maximal eindeutig pro Proposal; aber Operator kann sha8 nicht im Kopf behalten, muss copy-pasten.

3. Empfehlung — B

Begründung:

- A: zu schwach, zwingt Operator nicht zur bewussten Variant-Auswahl
- B: Variante ist semantisch tragend (Operator weiß was er gewählt hat: recommended/conservative/aggressive), und ändert sich pro Klick → Muskel-Memory bricht
- C: technisch sauberster, aber Operator kopiert eh nur den String aus dem Modal-Helper-Text → kein „Cognitive Slow-Down" mehr

4. Risiko

- **A:** Operator klickt 3 Approves hintereinander, alle mit Asset-Symbol getippt → kein Cognitive-Slow-Down → Fehlentscheidung-Risiko.
- **B:** Operator könnte versucht sein die Variant zu wechseln nach Wizard Step 3 ohne den Confirm-String zu aktualisieren → Filament Form-Validation greift, aber UX leicht frustig.
- **C:** Maximal sicher, aber alle Operator copy-pasten → effektiv kein Cognitive-Slow-Down, dafür „nervig in der Hand".

5. Auswirkungen auf spätere Phasen

- `04_gui_operator_flow.md` **§6:** Hard-Confirm-Patterns-Tabelle erweitert
- `05_commandbus_worker.md` **§3:** payload-validator prüft Hard-Confirm-Format
- `02_risk_proposal_engine.md` **Output-Schema:** `variant` ist Pflichtfeld in `alternative_proposals[]` damit Wizard den String konstruieren kann
- `08_open_questions.md` **Q-MH-11:** Status open → decided

6. Default-Vorschlag

Option B — `<asset>:<variant>`

Konkrete Confirm-Strings:

- `SOL:recommended`
- `SOL:conservative`
- `SOL:aggressive` (nur Testnet, Mainnet via Q-MH-13 disabled)
- bei Operator-Override (Q-MH-3): `SOL:override:recommended` (zusätzliches `override:` Segment macht Override-Audit eindeutig nachvollziehbar)

DR-7 · Wallet-Signature-Konflikt bei Policy-Wechsel

1. Problem

`compute_account_hash()` (RECON-2.2a) berechnet `wallet_signature.account_hash` über alle Holdings **außer** `policy='tradable_quote'`. Begründung: USDT-Drift soll nicht jeden Cycle die Signature brechen.

Aber: wenn Operator promoted SOL von `frozen` → `managed`, ändert sich SOL's policy aber nicht die qty. Hash bleibt identisch. Soweit OK.

Konflikt: wenn Operator USDT umstellt von `tradable_quote` → `managed` (selten, aber möglich für ein Stablecoin-Yield-managed-Setup), kommt die qty plötzlich rein → Hash ändert sich → bei nächstem Bot-Restart `wallet_signature_matches=False` → `sys.exit(1)`. Auch: bei Promote `frozen → tradable_quote` (z.B. Operator gibt managed quote-asset frei) verschwindet qty → Hash ändert sich.

Folge: jede policy-Änderung *zwischen tradable_quote und einer der anderen 2* bricht das Vertrauen → Bot crasht beim Restart.

2. Optionen

A · Hash-Algorithmus erweitern: schließe nur dynamisch-veränderliche Slots aus — `tradable_quote` mit qty=null ist „strukturell stable" (nichts zu hashen), während `tradable_quote` mit numerischem qty (rare) als „material" zählt. Aber: macht Algorithmus komplexer und Edge-Case-anfällig.

B · Hash über Asset-Symbole + Policy (qty raus) — Hash über `[(asset, policy)]` Tupel-Liste, ignoriert qty komplett. Drift in qty bricht nichts; Drift in policy bricht. Preis: Wallet-Verkauf eines frozen Assets (qty fällt) wird nicht mehr detektiert.

C · Promote-Operation muss baseline_holdings.json transactional mit-updaten — wenn Operator SOL `frozen → managed` promotet, schreibt der Worker beide Files (managed_state.json + baseline_holdings.json mit aktualisierter policy + neuer account_hash). Hash bleibt valid, weil Bot-Restart die neue Hash gegen die neue baseline rechnet.

D · Hard-Constraint: tradable_quote ↔ managed/frozen ist verboten — Operator darf nur `frozen ↔ managed` switchen, nie über tradable_quote-Grenze. Quote-Asset bleibt durable quote.

3. Empfehlung — C + D Hybrid

Begründung:

- D ist die durable-Regel: tradable_quote (USDT/USDC etc.) ist eine fundamentale Bot-Annahme — Bot braucht spendable Quote zum Trading. Umstellung quote → managed wäre semantisch ein anderer Bot.
- C löst das eigentliche Problem: `frozen ↔ managed` Switch via baseline-Apply (transactional), Hash bleibt damit konsistent. Worker macht Apply auf baseline_holdings.json **und** managed_state.json in einer DB-Transaction.
- A wäre eine Algorithmus-Komplikation ohne klaren Mehrwert
- B verliert wichtige Drift-Detection (qty-Drift auf frozen Assets ist genau was wir detektieren wollen)

4. Risiko

- **A:** Algorithmus-Komplexität → Tests müssen alle Edge-Cases abdecken → mehr Surface-Area für Bugs.

- **B:** verliert Drift-Detection-Granularität → managed_drift_alert (Q-MH-7 qty-drift) kann nicht aus signature-mismatch abgeleitet werden, braucht separate qty-Diff-Berechnung pro Cycle.
- **C:** Worker-Handler komplexer (zwei File-Writes transactional, beide mit eigenen canonical_hashes); aber das ist nur 1× pro promote, nicht per-cycle.
- **D:** Operator könnte sich eingeschränkt fühlen wenn er einen Bot-Rebalancing-Use-Case hat („mein USDT-Bestand soll auch managed sein"); aber dieses Szenario ist konstruiert und kollidiert mit fundamentaler Quote-Asset-Annahme.

5. Auswirkungen auf spätere Phasen

- `99_master_boundaries.md`: neue G-DR-14 ergänzen: „tradable_quote ↔ managed/frozen Übergang verboten. baseline-Apply darf policy-Wechsel nur innerhalb {frozen, managed} zulassen."
- `01_foundation.md`: BaselineHoldingsAllowlist-Validator erweitert: bei policy-Wechsel-Apply prüft Validator dass keine tradable_quote-Slot zu/von wechselt
- `05_commandbus_worker.md`: approve_managed_proposal-Handler schreibt managed_state.json **und** baseline_holdings.json transactional — das Promote-Pattern wird zum Two-File-Atomic-Update
- `03_state_machine.md` **Übergangs-Tabelle:** keine Änderung (managed_active ist nur über baseline-Apply erreichbar; baseline-Apply ist eigene Operation)
- `07_mainnet_future.md`: MN-SR ergänzen — auf Mainnet ist die Two-File-Atomic noch wichtiger, weil signature-mismatch beim Bot-Restart Mainnet-Bot crashen lässt

6. Default-Vorschlag

Option C + D Hybrid:

1. **G-DR-14 (NEU, durable):** „Policy-Wechsel ist nur erlaubt zwischen `{frozen, managed}`. Übergänge zu/von `tradable_quote` sind hard-blocked. Quote-Asset-Bestand bleibt durable quote."

2. **Worker-Handler-Pattern für approve_managed_proposal:** schreibt beide Files transactional in derselben DB-Transaction: - `managed_state.json` (state=managed_active gesetzt) - `baseline_holdings.json` (policy für SOL: frozen → managed, neue canonical_hash + account_hash) - Audit-Snapshot (SM-6) erfasst beide File-States vor + nach der Mutation - bei einem File-Write-Failure: Rollback beider Writes via Backup-Restore (G-DR-6)

3. **release_managed_asset Pattern (analog):** schreibt beide Files transactional: - `managed_state.json` (state=frozen) - `baseline_holdings.json` (policy: managed → frozen)

4. **Test-Pinning:** - Boundary-Test pinnt G-DR-14 (kein tradable_quote ↔ frozen/managed) - Two-File-Atomic-Test pinnt Worker-Handler-Verhalten - signature-match-after-promote-Test pinnt dass Bot-Restart nach Promote nicht crasht

---

Konsolidierter Default-Set (alle 6 Empfehlungen auf einen Blick)

#	Frage	Default-Antwort
Q-MH-14	Worker-Daemon required?	**Ja**, als eigene Phase MH-0.5 vor MH-1 (klein, reversibel via `docker compose down`)
Q-MH-15	managed_state SoT?	**Hybrid C** — JSON ist SoT für Bot, DB ist Cache für GUI; Worker schreibt beide transactional
Q-MH-13	Mainnet-Disable	**3 Disables**: aggressive Variante + dca_recommended + t2_pump_dump-as-recommended
Q-MH-2	Anzahl Varianten	**Engine berechnet 3, Output env-abhängig: Testnet 3 sichtbar, Mainnet 2 sichtbar**
Q-MH-11	Hard-Confirm-Pattern	`<asset>:<variant>` (z.B. `SOL:recommended`); Override-Pfad mit `<asset>:override:<variant>`
DR-7	Wallet-Signature bei Policy-Wechsel	**G-DR-14 NEU** + Two-File-Atomic-Pattern (managed_state + baseline_holdings transactional)

Was nach Operator-Antworten passiert (kein Implement, nur Doc-Update)

1. Eintrag in `08_open_questions.md` Status `open → decided` für Q-MH-14, 15, 13, 2, 11
2. Ergänzung in `99_master_boundaries.md`: neue **G-DR-14** (tradable_quote-Lock)
3. Phase-File-Updates pro betroffener Datei (durch Cross-Reference-Tabelle in `08_open_questions.md` ermittelbar)
4. Re-Build der ZIP wenn gewünscht, oder Commit-Phase

Operator-Antwort-Schablone (zum Ausfüllen)

#	Frage	✅ Default akzeptieren	✏️ Custom-Antwort	❓ Nachfragen
Q-MH-14	Worker-Daemon?	☐	_____	☐
Q-MH-15	managed_state SoT?	☐	_____	☐
Q-MH-13	Mainnet-Disable?	☐	_____	☐
Q-MH-2	Varianten-Anzahl?	☐	_____	☐
Q-MH-11	Hard-Confirm-Pattern?	☐	_____	☐
DR-7	Wallet-Signature-Konflikt?	☐	_____	☐

Boundaries gehalten: keine Implementierung, kein Code, keine Migration, kein Bot-Restart, kein Worker, kein Mainnet, kein Push.

— Ende Q-MH-Antwort-Session 2026-05-10 —