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 —