EFM Codex: Operator's Guide

Deployment, Monitoring, and Response

Appendix O to Volumes I–II

Entropica SPC
Yology Research Division

Version 1.1 — December 2025

Audience: This guide is for operators, SREs, and governance teams responsible for deploying and running an EFM forest. It is *not* for proving invariants or modifying the specification.
Scope: Summarizes normative behavior from Volumes I–II and translates it into operational concepts and procedures. For formal definitions and proofs, see the source volumes.

Contents

1 Core Concepts in Operator Terms

1.1 Entity Glossary

EFM Term	**Operator Analog**	**Description**
Capsule	Managed agent instance	Individual autonomous unit with role and permissions
Vault (Layer 0)	System-wide safety baseline	Immutable constraints; cannot be modified at runtime
Reflex Engine (L0.5)	Per-capsule circuit breaker	Sub-10ms safety interrupt; blocks unsafe actions
Arbiter Layer (L2)	Distributed safety council	Deliberative consensus for ambiguous decisions
d-CAM	Consensus protocol	Byzantine-fault-tolerant voting among Arbiters
Forest (L3)	Fleet topology	Swarm structure with dialect lineages
Trunk	Main branch	Primary dialect shared by capsule group
Branch	Canary/variant	Diverged dialect subset under evaluation
Gardener	Human operator	Override authority with cryptographic keys

Table 1: EFM concepts mapped to operator-familiar terms.

1.2 The Two Primary Health Signals

Operators monitor two key metrics:

1. ΔS **(Behavioral Entropy)** — Per-capsule (Vol. I §3). Measures how "strange" an individual capsule is behaving relative to its baseline. Threshold: τ (role-dependent, typically 0.3–0.5).

2. **SCI (Swarm Coherence Index)** — Per-trunk/branch (Vol. II §3.2). Measures how "aligned" a population is behaving collectively. Range: 0–1. Threshold: θ_{SCI} (deployment-dependent, typically 0.65–0.85).

Key Insight: A swarm can have all capsules with $\Delta S < \tau$ (all individually "safe") while $SCI < \theta_{SCI}$ (collectively drifting). Both signals are necessary.

2 Standing Up an EFM Forest

2.1 Pre-Deployment Checklist

1. **Define Vault Commandments** — Immutable safety rules. Once deployed, cannot be changed without full system rebuild.
2. **Configure Genesis Parameters** — Role archetypes, τ/λ defaults, permission matrices.
3. **Provision Capsules** — Deploy initial capsule fleet with Genesis Protocol. Verify each capsule's Vault Hash Chain links to system root.
4. **Deploy Arbiter Set** — Minimum 5 Judicial Capsules for $f = 2$ Byzantine tolerance. Configure d-CAM quorum parameters.
5. **Initialize d-CTM** — Distributed ledger storage. Verify append-only property and ZK-SP integration.
6. **Configure Gardener Keys** — Generate and securely store Gardener override keys. Test override on non-critical verdict.
7. **Enable SCI Computation** — Configure d_{behav} components, sampling strategy (if $> 10^4$ capsules), and alert thresholds.

2.2 First Forest Verification

Before production traffic:

Check	**Expected**	**Command/Method**
Capsules online	$\geq N_{min}$	Dashboard: Active Capsules
d-CTM reachable	All nodes	`dctm-health -all`
SCI computation	Value $\in [0, 1]$	Dashboard: SCI gauge
Reflex latency	< 10ms p99	Metrics: `reflex_latency_p99`
Arbiter quorum	≥ 5 available	`dcam-status -quorum`
Gardener override	Success	Manual test on staging verdict

Table 2: First-forest verification checklist.

3 Day-to-Day Monitoring

3.1 Key Metrics and Alerts

Wire up the following to your monitoring stack (Prometheus, Datadog, etc.):

Metric	**Alert Threshold**	**Action**
`delta_s_p95`	$> \tau - 0.05$	Review high-entropy capsules
`sci_per_trunk`	$< \theta_{alert}$	Investigate drift pattern
`reflex_halt_rate`	$>$ 10/min sustained	Check for misconfiguration
`arbiter_verdict_latency`	$>$ 5s p99	Scale Arbiter pool
`orphan_count`	> 0	Verify Orphan Protocol active
`fork_rate`	$>$ 2/hour	Possible instability
`branch_count`	$> B_{max} - 2$	Approaching branch limit

Table 3: Recommended monitoring configuration.

3.2 Dashboard Layout

Recommended panels:

1. **Fleet Health** — Capsule count by state (ACTIVE, PROBATION, QUARANTINE, PURGED)
2. **Entropy Distribution** — Histogram of ΔS across all capsules
3. **SCI Timeseries** — Per-trunk SCI with threshold lines
4. **Arbiter Activity** — Verdict rate, latency, decision breakdown
5. **Forest Topology** — Visual tree of trunks/branches with capsule counts
6. **Gardener Actions** — Recent overrides and approvals

4 Response Playbooks

4.1 Entropy Spike (S2–S3 Severity)

Playbook: High ΔS on Individual Capsule

Trigger: $\Delta S \geq \tau$ for capsule C, Reflex halt issued.
Steps:

1. **Inspect History:** Query d-CTM for C's recent action log. Identify which ΔS component spiked (action, resource, goal).
2. **Classify:**
   - *Expected workload change* → Adjust τ via Threshold Governance (Vol. II §2.6)
   - *Legitimate anomaly* → Leave halt in place, monitor
   - *Misconfiguration* → Fix configuration, restart capsule
   - *Potential compromise* → Escalate to Arbiter for QUARANTINE
3. **Document:** Log decision rationale in d-CTM via Gardener annotation.

Escalation: If pattern repeats across multiple capsules, investigate swarm-level cause (see SCI Dip playbook).

4.2 SCI Below Threshold

Playbook: SCI Dip Below θ_{fork}

Trigger: $SCI < \theta_{fork}$ for $> T_{debounce}$ ticks.
Steps:

1. **Identify Divergence:** Use DDI breakdown to find which capsule subset is drifting. Check behavioral distance matrix.
2. **Review Arbiter Verdicts:** Query recent FORK proposals. If Arbiter already deliberating, wait for verdict.
3. **Validate Fork Decision:**
   - If FORK approved: Verify branch created correctly, SCI improved per-group
   - If FORK rejected: Review DCG rationale; consider manual Gardener review
4. **Post-Fork Monitoring:** Track branch stability. Prepare for eventual Merge if branch proves valuable.

Escalation: If SCI continues dropping post-fork, investigate deeper (possible adversarial drift or systemic issue).

4.3 Orphan Detection

Playbook: Orphan Capsules Detected

Trigger: `orphan_count` > 0.
Steps:

1. **Verify Protocol Active:** Confirm Orphan Protocol is running (capsules should be in QUARANTINE).
2. **Check Recovery Attempts:** Query d-CTM for re-sync attempts. Are compatible trunks being tried?
3. **Monitor Timeout:** If approaching $N_{orphan\_timeout}$, decide:
   - *Recoverable* → Manually assign to trunk via Gardener action
   - *Unrecoverable* → Allow PURGE, review forensics
4. **Post-Purge Review:** Examine d-CTM entry for PURGE. Verify compliance with E2 Purge Justification constraint (Vol. II §3.9.5).

Escalation: Multiple orphans simultaneously may indicate trunk corruption or network partition.

4.4 Catastrophic Failure: Arbiter Unavailability

Runbook 1: ALL Arbiters Unavailable

Trigger: Arbiter availability $< 20\%$ or all Arbiters unreachable.
Severity: S4 (CRITICAL)
Immediate Actions:

1. **Verify Scope:** Confirm this is not a monitoring failure. Check network connectivity to Arbiter nodes.
2. **Emergency Spawn:** If truly unavailable, spawn emergency Arbiters from standby pool:

   ```
   gardener spawn-arbiters --emergency --count=3
   ```

3. **Queue Freeze:** Escalation queue will grow. Set queue priority to CRITICAL-only:

   ```
   efm-cli escalation --mode=critical-only
   ```

4. **Capsule Guidance:** Broadcast to capsules: "Arbiters recovering, reduce escalations"

Recovery Criteria:

- Arbiter availability $> 50\%$
- Escalation queue < 100 pending

- No new Arbiter failures for 10K ticks

Post-Incident: Root cause analysis required. Check for common cause (resource exhaustion, coordinated attack, network partition).

4.5 Catastrophic Failure: d-CTM Compromise

Runbook 2: d-CTM Integrity Compromised

Trigger: d-CTM hash verification failure OR write latency > 1000ms sustained.
Severity: S5 (EMERGENCY)
Immediate Actions:

1. **Halt All Operations:** This is a Layer 0 violation. Issue forest-wide FREEZE:

   ```
   gardener forest --action=FREEZE --reason=DCTM_COMPROMISE
   ```

2. **Isolate d-CTM:** Prevent further writes until integrity verified:

   ```
   efm-cli dctm --mode=read-only
   ```

3. **Integrity Check:** Run full verification against backup:

   ```
   efm-cli dctm --verify --against=backup
   ```

4. **If Corrupted:** Initiate rollback to last known-good state (see Appendix J §8).

Recovery Criteria:

- d-CTM integrity verified (hash match)
- Write latency < 100ms
- Gardener sign-off on recovery

Post-Incident: Mandatory forensic replay (Appendix M). Identify attack vector or corruption source.

4.6 Catastrophic Failure: Runaway Spawn

Runbook 3: Runaway Spawn Event

Trigger: Spawn rate $> 2 \times R_{max}$ OR depth exceeds D_{max} anywhere.
Severity: S4 (CRITICAL)
Immediate Actions:

1. **Emergency Lockdown:** ASG should auto-trigger, but verify:

   ```
   efm-cli spawn --check-lockdown
   ```

2. **If Not Locked:** Manually trigger:

   ```
   gardener spawn --lockdown --R_max=0 --tau=1.0
   ```

3. **Identify Source:** Query d-CTM for spawn lineage:

   ```
   efm-cli lineage --depth-exceeded --last=1000
   ```

4. **Quarantine Branch:** Isolate the runaway lineage:

   ```
   gardener quarantine --lineage=<ROOT_ID>
   ```

Recovery Criteria:

- Spawn rate $\leq R_{max}$
- All capsules $D \leq D_{max}$
- ASG parameters restored to operational bounds

Post-Incident: Review ASG configuration. Was R_{max} appropriate? Adjust deployment profile if needed.

4.7 Cascading Failure Decision Tree (Gap 7)

Critical Operational Guidance

Real-world failures cascade. When multiple symptoms appear simultaneously, use this decision tree to identify the **primary failure** and execute the correct runbook.

Runbook 0: Cascading Failure Diagnosis

Trigger: Multiple failure symptoms detected simultaneously (e.g., Arbiter failures + escalation queue growth + d-CTM slowdown).
The Problem:

```
Tick 0:    Arbiter failure -> Escalation queue grows
Tick 1K:   Queue growth -> Memory pressure -> d-CTM writes slow
Tick 2K:   d-CTM slow -> Health monitoring delayed -> False readings
Tick 3K:   False health -> ASG reduces spawn -> System underutilized
Tick 4K:   Multiple symptoms: Which runbook?
```

Step 1: Identify Primary Failure
Run diagnostics in this **exact priority order**:

1. **d-CTM Integrity Check** (highest priority):

   ```
   efm-cli dctm --verify-integrity
   ```

If CORRUPTED → **Runbook 2 IMMEDIATELY** (this is Layer 0)

2. **Reflex Engine Health**:

```
efm-cli reflex --status
```

   If DOWN → **EMERGENCY HALT** (Layer 0 violation)

3. **Arbiter Availability**:

```
efm-cli arbiters --status
```

   If $< 50\%$ → **Runbook 1** (Arbiter recovery)

4. **Spawn Rate Anomaly**:

```
efm-cli spawn --check
```

   If $> R_{max}$ → **Runbook 3** (spawn lockdown)

Step 2: Execute Primary Runbook
Follow the procedures for whichever failure was identified first. Do NOT try to address secondary effects until primary is resolved.
Step 3: Monitor Secondary Effects
Every 1000 ticks, re-run Step 1 diagnostics:

- If all checks pass → Recovery in progress
- If new failure detected → Escalate to Gardener (human decision needed)
- If same failure persists → Continue current runbook

Step 4: Root Cause Analysis
After recovery, mandatory forensic replay (Appendix M):

- Which component failed first?
- How did failure propagate?
- How to prevent cascade in future?

Decision Matrix:

Symptom Combination	**Likely Primary**	**Action**
Arbiter ↓ + Queue ↑	Arbiter failure	Runbook 1
d-CTM slow + Everything else	d-CTM issue	Runbook 2
Spawn ↑ + Depth ↑	Runaway spawn	Runbook 3
All symptoms at once	d-CTM first	Runbook 2 then reassess
Reflex down + Anything	Reflex failure	EMERGENCY HALT

4.8 Partial System Failure

Playbook: Partial Arbiter Failure (20–50%)

Trigger: Arbiter availability between 20% and 50%.
Severity: S3 (HIGH)
Note: This is NOT "all Arbiters failed" (Runbook 1) but partial degradation.
Steps:

1. **Assess Severity:**
   - $> 50\%$ available $\rightarrow$ Minor degradation, monitor only
   - $20 - 50\%$ available $\rightarrow$ This playbook
   - $< 20\%$ available $\rightarrow$ Runbook 1 (emergency)
2. **Spawn Replacement Arbiters:**
   ```
   gardener spawn-arbiters --count=<FAILED_COUNT/2>
   ```
3. **Redistribute Load:**
   ```
   efm-cli arbiters --rebalance
   ```
4. **Investigate Common Cause:**
   - Network partition? Check connectivity between Arbiter nodes
   - Resource exhaustion? Check memory/CPU on failed nodes
   - Coordinated attack? Check for suspicious patterns in d-CTM
5. **Alert Gardener:** Partial failures often indicate emerging problems

Recovery Criteria:

- Arbiter availability $> 90\%$
- No new failures for 5K ticks
- Escalation queue stable

5 Configuration and Tuning

5.1 Tunable Parameters

Parameter	**Default**	**Tuning Method**	**Constraint**
τ (entropy threshold)	Role-dependent	Threshold Governance	$\tau \in [\tau_{min}, \tau_{max}]$
λ (heuristic tolerance)	0.1	Threshold Governance	$\lambda \in [0, \lambda_{max}]$
θ_{SCI} (fork threshold)	0.7	Operator config	Per deployment regime
$T_{debounce}$	100 ticks	Operator config	> 0
$N_{orphan\_timeout}$	5000 ticks	Operator config	$> T_{recovery}$
B_{max} (branch limit)	10	Operator config	> 0
ATP horizon K_{max}	5000 ticks	Operator config	> 0

Table 4: Tunable parameters and constraints.

5.2 Safe Change Workflow

All parameter changes follow this workflow:

1. **Propose:** Submit change request with justification
2. **Validate:** For τ/λ changes, run Monotonic Sensitivity check (Vol. I Theorem 3.1)
3. **Approve:** Gardener signs change (or Arbiter for Threshold Governance)
4. **Log:** Change recorded to d-CTM with ZK-SP proof
5. **Monitor:** Watch affected metrics for $T_{observation}$ ticks
6. **Rollback:** If regression detected, revert via same workflow

Never Modify Directly

The following are **immutable** and cannot be changed at runtime:

- Vault Commandments (Layer 0)
- Reflex-Core rules (Layer 0.5)
- Genesis root hash
- d-CTM append-only property

Changes to these require full system rebuild.

6 Governance and Audits

6.1 Gardener Responsibilities

1. **Override Authority:** Can override any d-CAM verdict (subject to Vault constraints)
2. **DCG Review:** Review Deliberation Context Graphs for high-impact decisions (Vol. II §2.3)
3. **Micro-Heuristic Approval:** Sign off on Reflex-Heuristic updates derived from Arbiter precedent
4. **Conflict Arbitration:** Resolve disputes between safety and performance
5. **Audit Cooperation:** Provide d-CTM access for compliance reviews

6.2 Incident Reconstruction

To reconstruct any incident from audit trail:

1. **Identify Verdict:** Query d-CTM by capsule ID, timestamp, or verdict type
2. **Retrieve DCG:** Use `dcg_hash` to fetch full reasoning graph (Vol. II Definition 2.3)
3. **Verify ZK-SP:** Confirm proof chain is valid (tamper detection)
4. **Trace Lineage:** Follow Vault Hash Chain to genesis for affected capsules
5. **Document:** Produce audit report showing invariant compliance

6.3 Compliance Checklist

For "meaningful human control" (EU AI Act Article 14 and similar):

Requirement	**EFM Mechanism**	**Evidence**
Human can understand system behavior	DCG, d-CTM logs	Audit reports
Human can intervene	Gardener override	Override logs in d-CTM
Human can stop system	Gardener PURGE authority	Purge records
Decisions are traceable	ZK-SP chain	Proof verification
System cannot override human	Vault constraint on overrides	Architecture

Table 5: Regulatory compliance mapping.

7 Quick Reference

7.1 Common Commands

Task	Command
Check capsule status	`efm-ctl capsule status <id>`
Query d-CTM	`dctm-query -filter "capsule=<id>"`
View SCI for trunk	`efm-ctl forest sci <trunk_id>`
List active branches	`efm-ctl forest branches`
Initiate Gardener override	`gardener-ctl override <verdict_id> -reason "..."`
Check Arbiter quorum	`dcam-status -quorum`
Export audit trail	`dctm-export -from <ts> -to <ts> -format json`

Table 6: Common operator commands (implementation-specific; adjust for your deployment).

7.2 Severity Levels

Level	ΔS Range	Action	Response Time
S0 (Normal)	$< 0.5\tau$	None	—
S1 (Elevated)	0.5τ – 0.8τ	Monitor	Next shift
S2 (Warning)	0.8τ – τ	Investigate	1 hour
S3 (Critical)	$\geq \tau$	Reflex halt, review	Immediate

Table 7: Severity levels and response expectations.

8 Implementation Guidance

This section provides working examples for key operational concepts.

8.1 Micro-Heuristic Deployment

Micro-Heuristics are compact behavioral patterns deployed to the Reflex-Heuristic layer:

[language=Python, caption=Micro-Heuristic deployment example.] from efm.reflex import MicroHeuristic, ReflexHeuristic

Define a new micro-heuristic for resource hoarding detection hoarding$_detector = MicroHeuristic(name = "resource_hoarding_v1", pattern = b"1234...56", 64 - bytesignatureseverity = Severity.S2, response = Response.THROTTLE, metadata = "author" : "discovery_stack", "source" : "golden_heuristic_GH - 2847", "validated" : True)$

Deploy via Validation Gate (ensures Monotonic Sensitivity) reflex = ReflexHeuristic.current() result = reflex.deploy(hoarding$_detector$)

if result.success: print(f"Deployed: result.signature$_id")print(f"Newsensitivity : result.new_sensitivity")else : print(f"Rejected : result.reason")e.g., "monotonicity_violation"$

Monotonic Sensitivity: The Validation Gate ensures that new micro-heuristics only *increase* detection sensitivity. If a proposed heuristic would reduce sensitivity (fewer detections), it is rejected. See Vol. I §3.5.

8.2 ZK-SP Audit Path Verification

Operators can verify the ZK-SP audit trail for any capsule:

[language=Python, caption=ZK-SP audit path verification.] from efm.audit import ZK-SPVerifier, AuditPath

Retrieve audit path for a capsule capsule$_{i}d = "C - 2024 - 12 - 15 - 00042" path = AuditPath.for_{c}apsule(capsule_{i}d)$

Verify the complete chain verifier = ZKSPVerifier() result = verifier.verify$_{c}hain(path)$

print(f"Chain length: len(path.proofs)") print(f"Genesis anchor: path.genesis$_{h}ash[: 16]...") print(f" Current anchor : path.current_{h}ash[: 16]...") print(f" Verified : result.valid")$

if not result.valid: print(f"Break at proof result.break$_{i}ndex") print(f" Reason : result.reason")$

Verify specific state at tick tick = 50000 state$_{p}roof = verifier.verify_{s}tate_{a}t_{t}ick(capsule_{i}d, tick) print(f" State at t = tick : state_{p}roof.state_{h}ash[: 16]...")$

Table 8: ZK-SP verification CLI commands.

Command	**Description**
`efm-cli zksp verify -capsule=<id>`	Verify capsule audit chain
`efm-cli zksp anchor -tick=<n>`	Get anchor hash at tick
`efm-cli zksp replay -from=<t1> -to=<t2>`	Forensic replay
`efm-cli zksp export -capsule=<id>`	Export proof chain

8.3 Deployment Hooks

Pre-deployment validation hooks ensure system integrity:

[language=Python, caption=Deployment hook configuration.] from efm.deploy import DeploymentHooks, ValidationResult

class ProductionHooks(DeploymentHooks): """Custom hooks for production deployment."""

def pre$_{g}enesis(self, config) - > ValidationResult : """ValidatebeforeGenesisceremony.""" Checkallspawnconditionsareconfigurableifnotconfig.spawn_{c}ondit$
$returnValidationResult.FAIL("Missingspawnconditions")$

Verify ZK-SP infrastructure if not self.zksp$_{a}vailable() : returnValidationResult.FAIL("ZK - SPnotinitialized")$

return ValidationResult.PASS

def post$_{g}enesis(self, root_{c}apsule) - > ValidationResult : """ValidateafterGenesis.""" VerifyVaultbindingifnotroot_{c}apsule.vault_{b}ound() : returnValidationResult.FAIL("Vaultbindingfailed")$

Verify d-CTM initialized if not root$_{c}apsule.dctm_{a}ctive() : returnValidationResult.FAIL("d - CTMnotactive")$

return ValidationResult.PASS

def pre$_{s}pawn(self, parent, child_{s}pec) - > ValidationResult : """Validatebeforeanyspawn.""" Delegatetostandardspawngatereturnself.spawn_{g}ate.validate(parent, child_{s}$

Register hooks deploy = Deployment(profile=Profile.PRODUCTION) deploy.register$_{h}ooks(ProductionHooks())$

Changelog

v1.2 (December 2025) — *Catastrophic Failure Runbooks*

- **Runbook 1:** Arbiter unavailability procedures (§4.4)
- **Runbook 2:** d-CTM compromise response (§4.5)
- **Runbook 3:** Runaway spawn event handling (§4.6)
- **Gap 7 (Added):** Cascading failure decision tree (§4.7)
- **Added:** Partial system failure playbook (§4.8)
- 5 new operational playbooks total

v1.0 (December 2025)

- Initial operator's guide
- Core concepts and glossary
- Day-to-day monitoring procedures
- Response playbooks (Entropy Spike, SCI Dip, Orphan Detection)
- Configuration and tuning guidance
- Governance and audit requirements

— *End of Operator's Guide* —

For formal definitions and proofs, see Volumes I–II.
For Constitutional Kernel details, see Appendix J.
For Adaptive Spawn Governance, see Appendix N.