FreeOS DevOps Shadow Architecture

Autonomous Infrastructure Monitoring — Master Agent and Watchmen Design

March 2026

Prepared by: Henrik Ibsen eHealthBrains www.ehealthbrains.com

Classification: Internal Architecture Design **Audience:** Platform Engineering, DevOps, IT Leadership

---

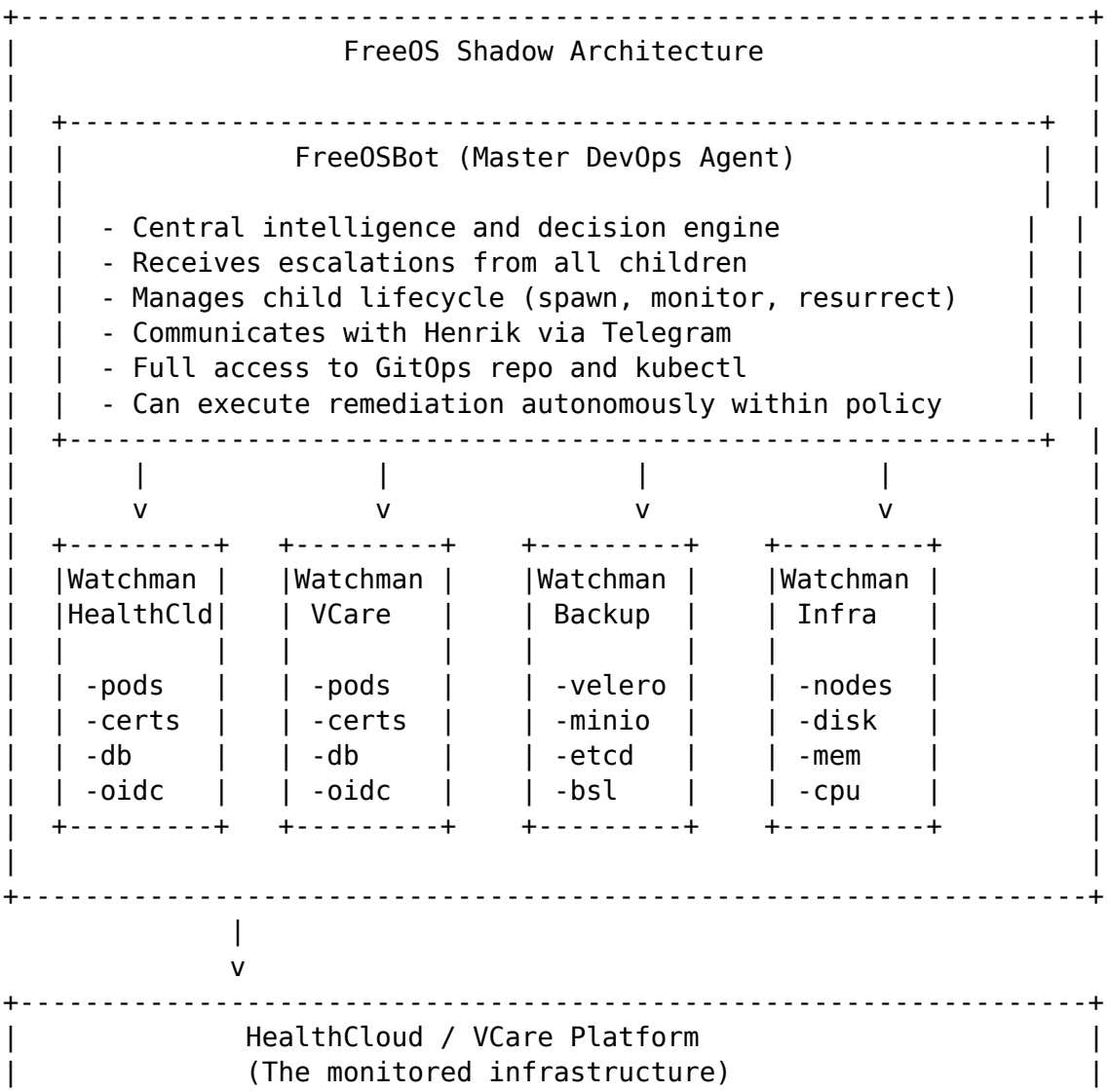
Executive Summary

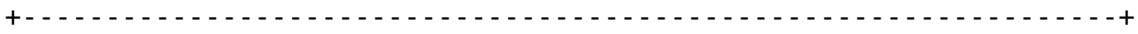
The FreeOS Shadow Architecture is a lightweight, resilient autonomous monitoring layer that runs alongside the HealthCloud and VCare platforms. It provides continuous infrastructure oversight through a hierarchy of simple agents — called **Watchmen** — supervised by a central **Master DevOps Agent** (FreeOSBot).

The design principle is **KISS — Keep It Simple and Stable.** Watchmen are not required to be intelligent. They are required to be **reliable, observable, and self-reporting.** The intelligence lives in the master. The watchmen are the eyes and ears.

A watchman that crashes is not a failure of the architecture. The master detects the outage, resurrects the child, and logs the event. The goal is self-healing at every level.

Architecture Overview





The Master Agent — FreeOSBot

Role and Responsibilities

FreeOSBot is the top-level autonomous DevOps agent. It is the only agent with:

- Full reasoning capability (Claude Sonnet 4.6 primary)
- Write access to the GitOps repository
- Permission to execute kubectl commands
- Ability to restart, reconfigure, or redeploy platform components
- Authority to contact Henrik for human escalation

What FreeOSBot Does

Function	Description
Child management	Spawns, monitors, and resurrects watchmen
Escalation handling	Receives alerts from watchmen and decides action
Autonomous remediation	Executes fixes within defined safe boundaries
Human escalation	Alerts Henrik via Telegram when outside safe boundaries
Memory maintenance	Keeps MEMORY.md and daily logs current
Heartbeat processing	Checks HEARTBEAT.md and executes scheduled tasks
Report generation	Generates platform status reports and PDFs
GitOps management	Commits, pushes, and monitors ArgoCD sync state

Autonomous Remediation Boundaries

FreeOSBot may act autonomously (without waking Henrik) for:

- Pod restart (non-cascading, isolated)
- ArgoCD force-sync (idempotent operations)
- Vault unseal (using stored keys)
- Certificate rotation (via cert-manager)
- Backup validation (read-only verification)
- DNS/CoreDNS configmap patch (reversible)
- Scaling adjustments within defined replica ranges

FreeOSBot **must escalate to Henrik** for:

- Any destructive operation (delete namespace, PVC, etcd snapshot restore)
- Cluster upgrades (k3s, RKE2, Helm chart major versions)
- CNI changes or network policy modifications

- Secret rotation or PAT updates
- Any change affecting more than 3 services simultaneously
- Any condition that has persisted unresolved for more than 2 hours

The Watchmen — Child Agents

Design Philosophy

Watchmen are simple. Watchmen are stable. Watchmen are replaceable.

A watchman has one job: observe a defined set of metrics or states, evaluate them against a fixed threshold table, and report any breach to the master.

A watchman does NOT: - Make architectural decisions - Execute remediation - Interpret complex failure patterns - Need a large context window or expensive model

A watchman DOES: - Poll specific endpoints or kubectl resources on a schedule - Compare current state against threshold table (plain logic) - Send a structured alert to the master if threshold is breached - Respond to a heartbeat ping with current health summary - Exit cleanly and wait for resurrection if it encounters an unrecoverable error

Model for children: `ollama/qwen2.5-coder:7b` (local, fast, cheap, sufficient)

Watchman Types

1. Cluster Watchman (per cluster)

Monitors Kubernetes cluster health. One instance per cluster (HealthCloud, VCare).

Checks (every 5 minutes):

Check	Threshold	Severity
Pod CrashLoopBackOff (any namespace)	Any	WARNING
Pod in Error state > 10 minutes	Any	WARNING
Pod OOMKilled	Any	WARNING
Deployment unavailable replicas	> 0 for 15 min	CRITICAL
Node NotReady	Any node	CRITICAL
ArgoCD app OutOfSync	> 1 app for 30 min	WARNING
ArgoCD app Degraded	Any	CRITICAL
PVC Pending / Lost	Any	CRITICAL
Namespace quota exceeded	> 90%	WARNING
k3s API unreachable	Any	CRITICAL

2. Certificate Watchman

Monitors TLS certificate expiry across all namespaces.

Checks (every 6 hours):

Check	Threshold	Severity
Certificate expiry	< 30 days	WARNING
Certificate expiry	< 7 days	CRITICAL
Certificate Ready = False	Any	CRITICAL

Check	Threshold	Severity
ClusterIssuer Ready = False	Any	CRITICAL

3. Backup Watchman

Monitors Velero backup jobs, MinIO storage, and etcd snapshots.

Checks (every hour):

Check	Threshold	Severity
Velero daily backup missed	Last backup > 26 hours	WARNING
Velero backup failed	Any failure	CRITICAL
BackupStorageLocation Available	Not Available	CRITICAL
MinIO pod not Running	Any	CRITICAL
MinIO storage utilization	> 80%	WARNING
MinIO storage utilization	> 95%	CRITICAL
etcd snapshot missed	Last snapshot > 26 hours	WARNING

4. Security Watchman

Monitors the security posture of the platform.

Checks (every 30 minutes):

Check	Threshold	Severity
Vault sealed	Sealed = true	CRITICAL
Vault pod not Running	Any	CRITICAL
Wazuh manager not Running	Any	WARNING
Harbor Trivy scan failed	Any	WARNING
New CRITICAL CVE in scanned images	Any	WARNING
NetworkPolicy missing from namespace	Any production ns	CRITICAL

5. Infrastructure Watchman

Monitors host-level resources on the bare metal node.

Checks (every 5 minutes):

Check	Threshold	Severity
Host CPU utilization	> 85% for 15 min	WARNING
Host CPU utilization	> 95% for 5 min	CRITICAL
Host memory utilization	> 85%	WARNING
Host memory utilization	> 95%	CRITICAL
Host disk utilization (root)	> 80%	WARNING
Host disk utilization (root)	> 90%	CRITICAL

Check	Threshold	Severity
Longhorn disk utilization	> 75%	WARNING
Longhorn disk utilization	> 90%	CRITICAL
Host load average (15m)	> 20	WARNING
Docker daemon not Running	Any	CRITICAL
OpenClaw container not Running	Any	CRITICAL

Escalation Framework

Severity Levels

Level	Definition	Watchman Action	Master Action
INFO	Normal operation, notable event	Log locally	None required
WARNING	Degraded state, no immediate impact	Alert master	Investigate + log
CRITICAL	Service impact, immediate attention needed	Alert master	Investigate + remediate or escalate
EMERGENCY	Data loss risk, cluster down, security breach	Alert master + retry	Wake Henrik immediately

Escalation Paths

```
Watchman detects threshold breach
  |
  +-- Severity = WARNING
  |     |
  |     +-> Alert Master (FreeOSBot)
  |           |
  |           +-> Assess: is this within autonomous remediation boundary?
  |                 |
  |                 +-- YES -> Remediate silently, log to daily notes
  |                 +-- NO  -> Log + monitor for escalation window (30 min)
  |                                |
  |                                +-- Resolved? -> Log "self-healed"
  |                                +-- Unresolved? -> Escalate to CRITICAL
  |
  +-- Severity = CRITICAL
  |     |
  |     +-> Alert Master (FreeOSBot)
  |           |
  |           +-> Assess: autonomous remediation possible?
  |                 |
  |                 +-- YES -> Attempt remediation (max 2 attempts)
  |                 |           +-- Success -> Notify Henrik (low priority)
  |                 |           +-- Fail x2 -> Escalate EMERGENCY
  |                 +-- NO  -> Notify Henrik via Telegram immediately
  |
  +-- Severity = EMERGENCY
        |
        +-> Alert Master
              |
```

```
            +-> Notify Henrik via Telegram (high priority message)
            +-> Document full context in daily log
        +-> Await instruction or timeout (30 min) then attempt safe recovery
```

Telegram Alert Format

```
[SEVERITY] FreeOSBot Alert - <cluster>/<namespace>

Issue: <description>
Component: <affected component>
Duration: <how long since detection>
Status: <what master has already tried>
Action required: <what Henrik needs to do, if anything>

Ticket: <daily log reference>
```

Child Agent Lifecycle Management

Normal State

```
Master spawns children on startup
Children run on schedule (cron-like internal loop)
Children heartbeat to master every 60 seconds
Master logs last-seen timestamp for each child
```

Child Unavailable (No Heartbeat)

```
Master notices child missed 2 consecutive heartbeats (2 minutes)
  |
  +-> Log: "Child <name> unresponsive — attempting resurrection"
  +-> Attempt session resurrection (sessions_spawn)
  +-> Wait 60 seconds for new child heartbeat
        |
        +-- Heartbeat received -> Log "Child <name> resurrected"
        +-- No heartbeat -> Retry resurrection (max 3 attempts, 5 min apart)
                              |
                             +-- Success within 3 attempts -> Log + monitor
                             +-- 3 failures -> Notify Henrik
                                   "Child <name> cannot be resurrected.
                                    Manual intervention required."
```

Child Dysfunctional (Responding but Wrong)

Signs: nonsensical alerts, wrong thresholds, reporting healthy when known broken.

```
Master detects anomalous output from child (validation check fails)
  |
  +-> Log: "Child <name> output invalid — quarantining"
  +-> Kill child session
  +-> Wait 30 seconds
  +-> Spawn fresh child from clean template
  +-> Validate first 3 outputs from new child
        |
        +-- Valid -> Log "Child <name> replaced, nominal"
        +-- Invalid again -> Escalate to Henrik
              "Child <name> persistently dysfunctional.
               Possible: template corruption, model issue, config drift."
```

Child Erratic (Flapping)

Signs: child oscillates rapidly between HEALTHY and CRITICAL on same component.

```
Master tracks alert state history per component (sliding 10-minute window)
  |
  +-- If same component flips state > 3 times in 10 minutes:
```

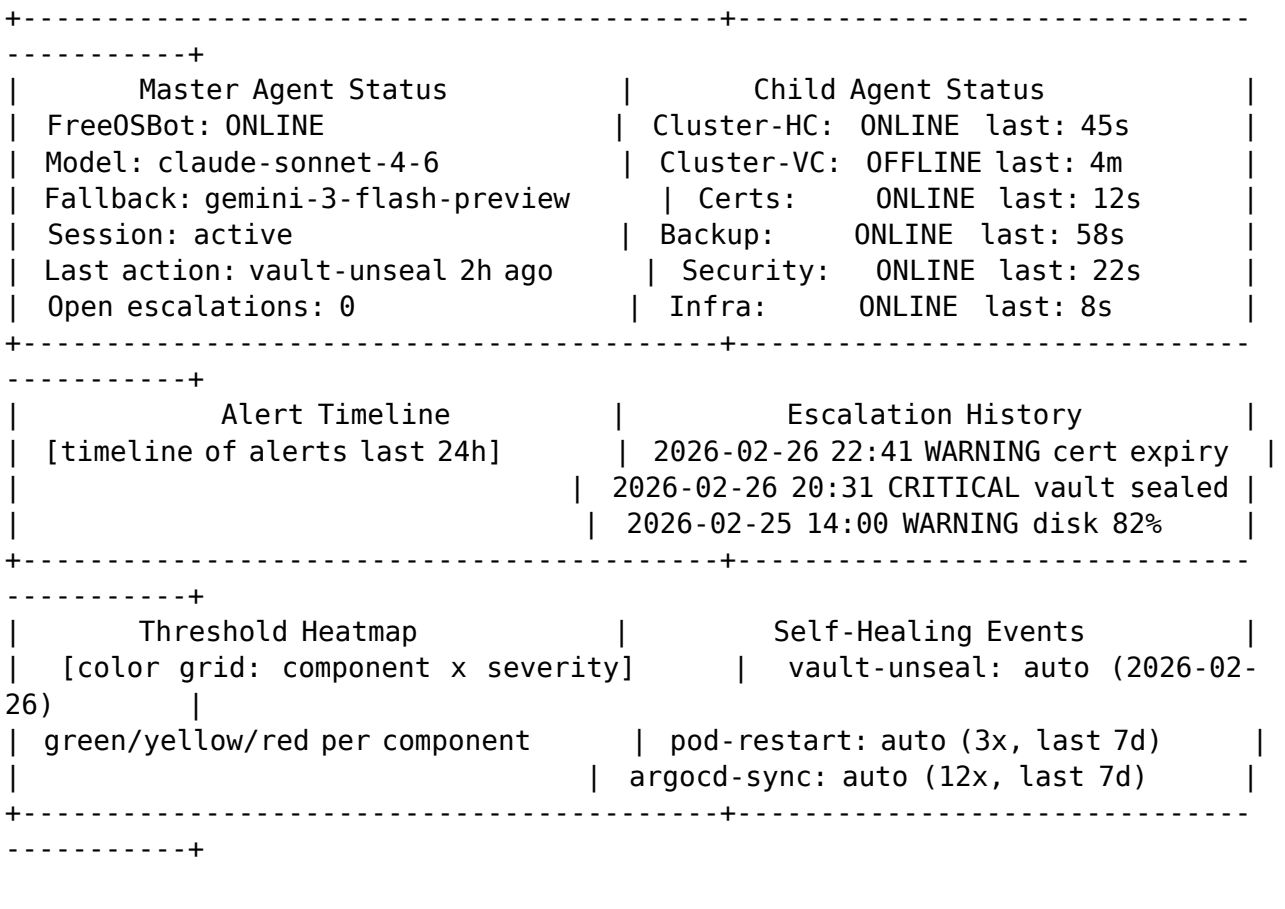
```
|
+-> Apply circuit breaker: suppress component alerts for 15 minutes
+-> Log: "Flap detected on <component> — circuit breaker active"
+-> After 15 min: re-evaluate with fresh check
      |
      +-- Stable HEALTHY -> Log "Flap resolved"
      +-- Stable CRITICAL -> Escalate normally
      +-- Still flapping -> Escalate to Henrik
            "Persistent flap on <component>. May indicate:
             intermittent network, resource contention, or
             race condition in health probe."
```

Visualization

Grafana Dashboard — Shadow Architecture

The Shadow Architecture state can be visualized in a dedicated Grafana dashboard within the existing kube-prometheus-stack deployment.

Proposed Panels



Metrics Exposed by Watchmen

Watchmen write structured state to a shared ConfigMap (`shadow-arch/watchman-state`) that Prometheus scrapes via a custom exporter:

```
shadow_watchman_up{name="cluster-hc"} 1
shadow_watchman_last_heartbeat_seconds{name="cluster-hc"} 45
shadow_alert_active{cluster="healthcloud", component="vault", severity="critical"} 0
shadow_alert_count_total{cluster="healthcloud", severity="warning"} 12
shadow_autoheal_total{action="vault-unseal"} 3
shadow_escalation_to_human_total 1
```

Alertmanager Rules

```yaml
- alert: WatchmanDown
  expr: shadow_watchman_up == 0
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "Watchman {{ $labels.name }} is down"

- alert: EscalationBacklog
  expr: shadow_alert_active{severity="critical"} > 0
  for: 30m
  labels:
    severity: emergency
  annotations:
    summary: "Unresolved critical alert for 30+ minutes"
```

Response Patterns Catalogue

Pattern 1: Pod CrashLoopBackOff (Isolated)

Detection: Single pod in CrashLoopBackOff, not affecting service.

Master response (autonomous): 1. Check pod logs for root cause 2. If OOM: check node memory pressure, attempt reschedule on different node 3. If config error: check last GitOps commit for change 4. Attempt pod delete (k8s will recreate) 5. If resolves within 5 min: log, no notification 6. If persists: notify Henrik with log excerpt

Pattern 2: Vault Sealed After Pod Restart

Detection: Security Watchman reports Vault sealed = true.

Master response (autonomous): 1. Confirm Vault pod is Running (not crashed) 2. Retrieve unseal keys from `memory/vault-init.md` 3. Execute 3 unseal operations via kubectl exec 4. Confirm Vault sealed = false 5. Notify Henrik: "Vault auto-unsealed after pod restart"

Pattern 3: Velero Backup Missed

Detection: Backup Watchman reports last backup > 26 hours.

Master response (autonomous): 1. Check Velero schedule status 2. Check MinIO BSL availability 3. Trigger manual backup: `velero backup create manual-$(date +%Y%m%d)` 4. Monitor backup completion (15 min timeout) 5. Notify Henrik with backup status

Pattern 4: ArgoCD App OutOfSync > 30 min

Detection: Cluster Watchman reports OutOfSync app.

Master response (autonomous): 1. Identify which app and why (last git commit, upstream chart change) 2. Check if change is intentional (recent commit by Henrik) 3. If unintentional drift: force sync 4. If intentional change: verify health after sync 5. Log result

Pattern 5: Node NotReady

Detection: Cluster Watchman reports node NotReady.

Master response (escalate immediately): 1. Collect node events and conditions 2. Check if workloads have migrated to other nodes 3. **Notify Henrik immediately** — node recovery requires human assessment 4. Provide context: last known state, running workloads, storage attachment status

Pattern 6: Disk > 90%

Detection: Infra Watchman reports host or Longhorn disk > 90%.

Master response (autonomous + escalate): 1. Identify largest consumers (`du -sh /var/lib/longhorn/*`) 2. Check for orphaned Longhorn volumes 3. Check Velero backup retention (purge expired) 4. Notify Henrik with disk usage breakdown and recommended actions 5. Do NOT delete anything without confirmation

Pattern 7: Security Breach Signal (Wazuh Alert)

Detection: Security Watchman reports high-severity Wazuh event.

Master response (escalate immediately): 1. Capture event details from Wazuh API 2. **Notify Henrik immediately via Telegram (EMERGENCY priority)** 3. Do NOT take autonomous action — security incidents require human judgment 4. Log full event context to daily notes

Implementation Roadmap

Phase 1 — Foundation (Immediate)

- ☐ Implement master heartbeat loop in HEARTBEAT.md
- ☐ Deploy Infra Watchman as OpenClaw sub-agent (local Qwen model)
- ☐ Deploy Cluster Watchman for HealthCloud
- ☐ Test child unavailable / resurrection flow
- ☐ Establish `shadow-arch/watchman-state` ConfigMap

Phase 2 — Full Coverage (Next Sprint)

- ☐ Deploy Backup Watchman
- ☐ Deploy Certificate Watchman
- ☐ Deploy Security Watchman
- ☐ Implement Grafana dashboard panels
- ☐ Add Prometheus metrics exporter for watchman state
- ☐ Add Alertmanager rules

Phase 3 — VCare Extension (When VCare Re-enabled)

- ☐ Deploy Cluster Watchman for VCare cluster
- ☐ Extend master to manage cross-cluster state
- ☐ Add platform-swap detection (HealthCloud/VCare active detection)

Phase 4 — Self-Improving (Future)

- ☐ Master learns from escalation history to improve thresholds
- ☐ Automated runbook generation from incident patterns
- ☐ Predictive alerting (disk growth rate, pod restart frequency trends)

Design Principles Summary

Principle	Application
KISS	Watchmen use simple polling logic, not AI reasoning
Fail-safe	A dead watchman does not cause data loss — master continues
Observable	Every agent action is logged; Grafana shows full state
Bounded autonomy	Master has a hard list of safe autonomous actions
Human-in-the-loop	EMERGENCY and destructive operations always escalate
Resilience-first	Architecture assumes children will fail; designed for recovery
Cost-conscious	Children use local Qwen model; master uses Claude only when needed
GitOps-consistent	All remediation is through the GitOps repo, not ad-hoc kubectl

---