Autonomous IT Incident Resolution through AIOps with Local LLMs: A Practical Framework and PoC

Konrad Soares
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00218708@mytudublin.ie

Introduction

Problem. Modern monitoring detects incidents but rarely closes the loop. Engineers face alert fatigue, inconsistent runbooks, and long MTTR, especially in constrained or air-gapped environments.
Aim. Design and evaluate a *policy-gated, local-LLM* framework that goes from alert → triage → safe action → verification with full auditability and zero data egress.
Approach. Prometheus/Alertmanager trigger a Python orchestrator with a local LLM (Ollama). Policies allowlist actions and enforce budgets/cooldowns. Outcomes are recorded in `/state/case-*.json` and surfaced via Telegram.

Framework / Policy Model

1. Closed-loop design

Alert → Case → LLM triage → Policy gate → Actions → Resolution (alert clears) or Escalation.

2. Policy model

Allowlist per hint (e.g., `frontend_down`, `disk_full`); restart budgets; cooldowns; max attempts; unsafe-action blocks; optional silencing to avoid storms.

3. Execution flow

Triage proposes a plan (e.g., restart; prune → df → restart). Policy validates scope/limits; dispatcher executes; verification via cleared alert; otherwise escalate and draft runbook.

PoC Environment & Tooling

1. Stack

Prometheus + Alertmanager (webhook) · Python orchestrator · Ollama (local LLM) · Frontend + sidecars · Telegram · `/state/case-*.json`.

2. Config & policies

Alert labels → policy `hint`; YAML allowlist; budgets/cooldowns/max-attempts; env ports/thresholds; silencing rules.

3. Reproducibility & security

Deterministic scenarios (Appendix E); zero egress; least privilege; timestamps for MTTA/MTTR and audit.

Observability, Scenarios & Metrics

1. Observability

System-of-record: `/state/case-*.json` (actions, notes, lifecycle).
Operator timeline via Telegram.
Alertmanager state used as verification source.

2. Scenarios (Appendix E)

S1: ENOSPC. Prune → df snapshot → restart; resolve on alert clear.
S2: Missing container. 3 attempts → escalate; runbook draft.
S3: FrontendDown. Restart with backoff; dedupe/suppression.

3. Metrics

Detection latency (startsAt−T0), MTTA, MTTR, Autonomy %,
guardrail compliance (attempts, budgets, cooldowns).

Key Results (KPIs)

- **MTTR:** Reduced in covered paths (ENOSPC, FrontendDown).
- **Autonomy %:** High where runbooks/policies exist; escalate on missing container.
- **Noise:** Fewer repeats via silencing/dedupe.
- **Audit:** `/state/case-*.json` (actions, notes, lifecycle).

Safety & Governance

- Allowlist per hint (`frontend_down`, `disk_full`).
- Budgets, cooldowns, max attempts; restart caps.
- Escalate on denial or failed verification.
- Zero egress (local LLM); least privilege.

How to Reproduce (PoC)

```
$ docker stop frontend1
```

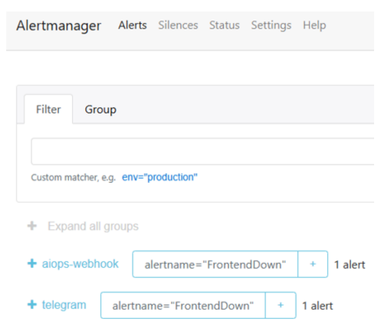

Fig. 1 Alert firing

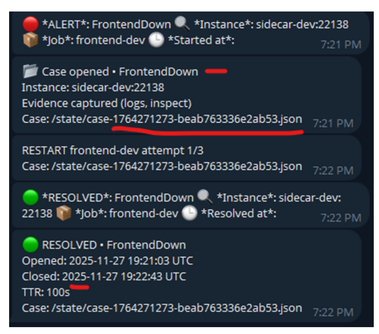

Fig. 2 Telegram message / Case / MTTR

Topic Overview

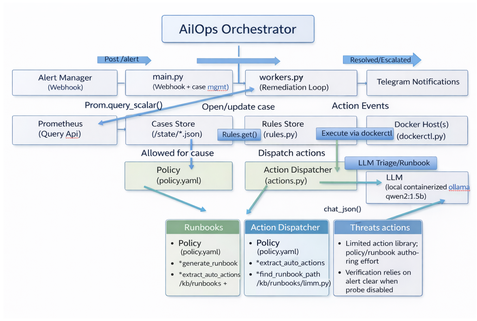

Fig. 3. System architecture

Conclusions

Conclusions.

- Closed-loop, policy-gated automation reduced MTTR in covered cases.
- Zero data egress with full audit trail (`case-*.json`) suits edge/air-gapped environments.
- Guardrails (allowlist, budgets, cooldowns) bounded risk; unresolved paths escalated with low noise.

QR Code for Recording