

Progetto GAINS

(Generative AI for Network Sustainability)

Deliverable D3.1

Technical Report su Progettazione e Implementazione Network Monitor

Giordano, A. - Basile, L. - De Giacomo, A. - Haider, B - Paesani, M.
2026-01

Work Package: WP3 Progettazione, Implementazione Prototipale e Valutazione degli Agenti

Stato: Definitivo | Versione: 1.0

Classificazione: Uso interno di progetto Condivisione controllata con partner su necessità

Abstract

Questo deliverable presenta la progettazione e il blueprint di implementazione del Network Monitor del progetto GAINS, un agente software orientato a trasformare la telemetria di rete in conoscenza operativa verificabile, replicabile e pronta per l'adozione industriale. In infrastrutture telco e datacenter, la disponibilità di dati non coincide con la capacità di prendere decisioni affidabili: le sorgenti sono eterogenee, i segnali sono spesso non confrontabili nel tempo e le sintesi prodotte in emergenza mancano di tracciabilità. Il Network Monitor nasce per colmare questo divario, imponendo un modello canonico dei dati, una catena di trasformazione esplicita e una governance dell'evidenza che consenta audit, riproducibilità e miglioramento continuo. Nel complesso, il Network Monitor si propone come fondazione ingegneristica per osservabilità avanzata e sostenibile: un agente capace di rendere la rete non solo misurata, ma anche interpretabile, verificabile e governabile. L'architettura è "read-only by design": l'agente non modifica configurazioni né stato di rete e opera esclusivamente tramite connettori in sola lettura verso sistemi di monitoraggio, database e storici time-series. Ogni risultato analitico include trace_id ed evidenze riferibili alle sorgenti e alle finestre temporali impiegate, abilitando una diagnostica trasparente e una responsabilità tecnica misurabile. La pipeline analitica è intenzionalmente deterministica: normalizzazione di metrica e unità, quality gates, calcolo di baseline robuste (quantili p50/p95), scoring di health e rilevazione anomalie con detector versionati. Questa scelta privilegia affidabilità e controllabilità, riducendo falsi positivi e rendendo i risultati ripetibili a parità di input e configurazione. Il sistema separa in modo rigoroso la telemetria raw ad alta cardinalità dalla memoria a lungo termine: lo storico rimane nei sistemi nativi, mentre il knowledge store conserva solo derivati a bassa cardinalità come summary, eventi di anomalia, note operatore, registry delle evidenze e input per analisi di sostenibilità. In particolare, il Network Monitor introduce l'oggetto canonico FootprintInputs, che produce proxy di carico e segnali di efficienza attribuibili per device e aggregabili per PoP, corredati da metodo, confidenza, assunzioni dichiarate e riferimenti alle evidenze. Questo

abilita un approccio di progressive enhancement: anche in assenza di misure fisiche di potenza device-level, l'agente fornisce output confrontabili e auditabili, migliorabili quando diventano disponibili segnali diretti. L'integrazione con un endpoint LLM esterno OpenAI-compatibile è prevista ma non autoritativa: viene impiegata in modo opzionale per embedding e explain grounded, mantenendo separati i meccanismi deterministici che governano health, severità e detection. Il deliverable include inoltre contratti API stabili con envelope standard ed error model coerente, oltre a un blueprint di deployment in Docker riproducibile, con segregazione dei secrets, mount read-only per le sorgenti time-series, policy di caching e retention conservative e una strategia di test basata su contract testing e golden tests per garantire determinismo e regressione controllata. Per la fase PoC successiva sarà adottato LibreNMS come NMS di riferimento, già operativo nella rete target con misure reali; tale scelta accelera l'integrazione e la validazione su dati autentici, impattando principalmente sui connettori e sul mapping verso il modello canonico, senza alterare invarianti architetturali e contratti. [1][2][3][4][5][19]

Indice

1. Introduzione

- 1.1 Scopo del deliverable e perimetro
- 1.2 Contesto operativo e principi di sicurezza
- 1.3 Assunzioni e dipendenze esterne

2. Requisiti del Network Monitor

- 2.1 Requisiti e criteri di tracciabilità
- 2.2 Requisiti funzionali
- 2.3 Requisiti non funzionali
- 2.4 Requisiti climate e vincoli di misurabilità

3. Architettura dell'agente Network Monitor

- 3.1 Principi architetturali
- 3.2 Vista logica dei componenti
- 3.3 Component model e responsabilità
- 3.4 Confini di responsabilità e interfacce
- 3.5 Workflow principali

4. Modello dei dati e memoria dell'agente

- 4.1 Data model canonico adottato dal Monitor
- 4.2 Domini dati e livelli di memoria
- 4.3 Oggetti persistiti, indicizzazione e policy di retention

- 4.4 EvidenceRef, trace_id e audit trail
- 5. Acquisition, normalizzazione e pipeline ETL
 - 5.1 Sorgenti e metodi read-only
 - 5.2 Envelope interno e normalizzazione unità/naming
 - 5.3 Mapping device identity e enrichment metadata
 - 5.4 Pipeline versionabili e criteri di data quality
- 6. Analisi deterministica: baseline, scoring e anomaly detection
 - 6.1 Finestre temporali e step
 - 6.2 Baseline (p50/p95) e drift
 - 6.3 Quality gates e gestione missing data
 - 6.4 Detector: segnali, regole e severità
 - 6.5 Output AnomalyEvent e correlazione
- 7. Climate support: produzione di FootprintInputs
 - 7.1 Catena segnali fisici e proxy logici
 - 7.2 Confidence, assunzioni e comparabilità
 - 7.3 Evidenze e limiti della stima
- 8. Contratti API e modelli di risposta
 - 8.1 Famiglie di endpoint
 - 8.2 Response envelope e invarianti
 - 8.3 Error model standardizzato
 - 8.4 OpenAPI e contract testing
- 9. Blueprint di deployment in Docker
 - 9.1 Obiettivo e perimetro del blueprint
 - 9.2 Componenti containerizzati e dipendenze esterne
 - 9.3 Struttura repository e gestione artefatti
 - 9.4 Docker compose di riferimento
 - 9.5 Bootstrap e smoke test
- 10. Implementazione reference: dettagli costruttivi
 - 10.1 Scelte implementative e moduli interni
 - 10.2 Connectors (API/DB/RRD) e policy read-only
 - 10.3 Configurazioni versionate (detectors.yml, mappings.yml)

10.4 Schema init per knowledge store e “bring your own vectors”

10.5 Logging strutturato, metriche e tracing

10.6 Testing deterministico e golden tests

10.7 Sicurezza: secrets, redaction, minimizzazione

11. Conclusioni e lessons learned

Bibliografia

Appendici