SmartDoctor

1. Introduzione e Obiettivi

SmartDoctor è un sistema proattivo e autonomo progettato per il monitoraggio delle performance di un server, con l'obiettivo principale di eliminare la "Alert Fatigue" generata dalle soglie statiche. Il sistema utilizza una logica di Autocorrezione che adatta dinamicamente le soglie di allerta al carico di lavoro reale del server.

Obiettivi Principali:

- **Automazione:** Azzerare l'intervento umano per la configurazione delle soglie.
- **Proattività:** Inviare notifiche solo per problemi critici reali, riducendo i falsi positivi.
- **Affidabilità:** Costruire un sistema containerizzato e una base dati solida per l'analisi storica.

2. Architettura

Il sistema è basato su un'architettura containerizzata, orchestrata tramite Docker Compose.

Componente	Tecnologia	Ruolo
Core Logic	PostgreSQL	Persistenza dei dati storici.
Database	Python 3+	Raccolta delle metriche e logica di alert/autocorrezione.
Interfaccia utente	Grafana	Strumento di Business Intelligence per la visualizzazione delle serie storiche.
Notifica	Telegram API	Canale di comunicazione immediato per gli alert.
Deployment	Docker/Docker Compose	Containerizzazione e gestione dei servizi.

Servizi Containerizzati:

1. **`PostgreSQL:`** Esegue il database.
2. **`smartdoctor_cron_job`**: Esegue gli script Python schedulati (simulando un cron job).
3. **`smartdoctor_grafana`**: Esegue l'interfaccia web per l'analisi dei dati storici.

3. Flusso Operativo e Ciclo di Vita del Dato

Il sistema opera in un ciclo continuo, gestito dal container `smartdoctor_cron_job`.

Raccolta Dati (`monitor.py`)

- **Frequenza:** Esegue l'azione schedulata ogni X minuti (definiti in `config.json`, tipicamente 10 minuti).
- **Azione:** Lo script `monitor.py` utilizza la libreria `psutil` per acquisire le metriche attuali del server (CPU %, RAM %, Disco %, Traffico di rete Sent/Recv in bytes).
- **Output:** I dati vengono immediatamente scritti nella tabella `system_metrics` del database PostgreSQL.

B. Analisi e Alerting (`alert_check.py`)

- **Frequenza:** Esegue l'azione schedulata ogni `x` minuti (tipicamente 10 minuti, dopo `monitor.py`).
- **Azione:** Lo script `alert_check.py` esegue una query SQL per recuperare i valori **massimi** (`MAX(cpu)`, `MAX(ram)`, ecc.) registrati nell'ultimo intervallo di tempo.
- **Logica:** Questi valori massimi vengono confrontati con le soglie attuali definite nel file **`config.json`**.

---

4. Funzionalità Distintiva: Autocorrezione (Adaptive Thresholding)

Questa logica è il cuore dell'Intelligenza di SmartDoctor e risiede nello script `alert_check.py`.

Condizione di Trigger per l'Autocorrezione

L'autocorrezione non scatta per un semplice superamento della soglia, ma solo se l'utilizzo massimo (`max_metric`) supera la soglia attuale (`current_threshold`) di oltre il 15%.

Sequenza di Azione

1. Calcolo Nuova Soglia: La nuova soglia è calcolata aggiungendo un margine del 5% all'utilizzo massimo osservato.

2. **Aggiornamento Configurazione:** Il nuovo valore viene scritto e salvato nel file **`config.json`** sul disco (aggiornando la configurazione dinamica).
3. **Notifica Autocorrezione:** Viene inviato un alert su Telegram per confermare l'aggiustamento automatico.

Alert Normale

Se la soglia viene superata, ma non abbastanza da attivare l'autocorrezione (cioè, il superamento è inferiore al 15%), viene inviato un normale ALERT CRITICO su Telegram.

5. Output e Formattazione

Tutte le comunicazioni avvengono tramite l'API di Telegram, utilizzando il formato Markdown per garantire chiarezza.

Tipi di Notifica

- **ALERT CRITICO (Es. CPU):** Segnalazione di un'anomalia.
- **AUTOCORREZIONE (Es. RAM):** Conferma dell'avvenuto adeguamento della soglia. Indica chiaramente la soglia vecchia e quella nuova.

6. Gestione della Configurazione e dei Tipi di Dato

- **File `config.json`:** Funge da database leggero per le soglie e le credenziali. È cruciale che i valori percentuali (CPU, RAM, DISK) siano sempre definiti come **numeri decimali (float)** (es. `1.0` anziché 1) per evitare errori nelle operazioni aritmetiche Python.
- **Persistenza Docker:** I volumi sono configurati per garantire che i dati di PostgreSQL e la configurazione di Grafana siano persistenti attraverso i riavvii del container.