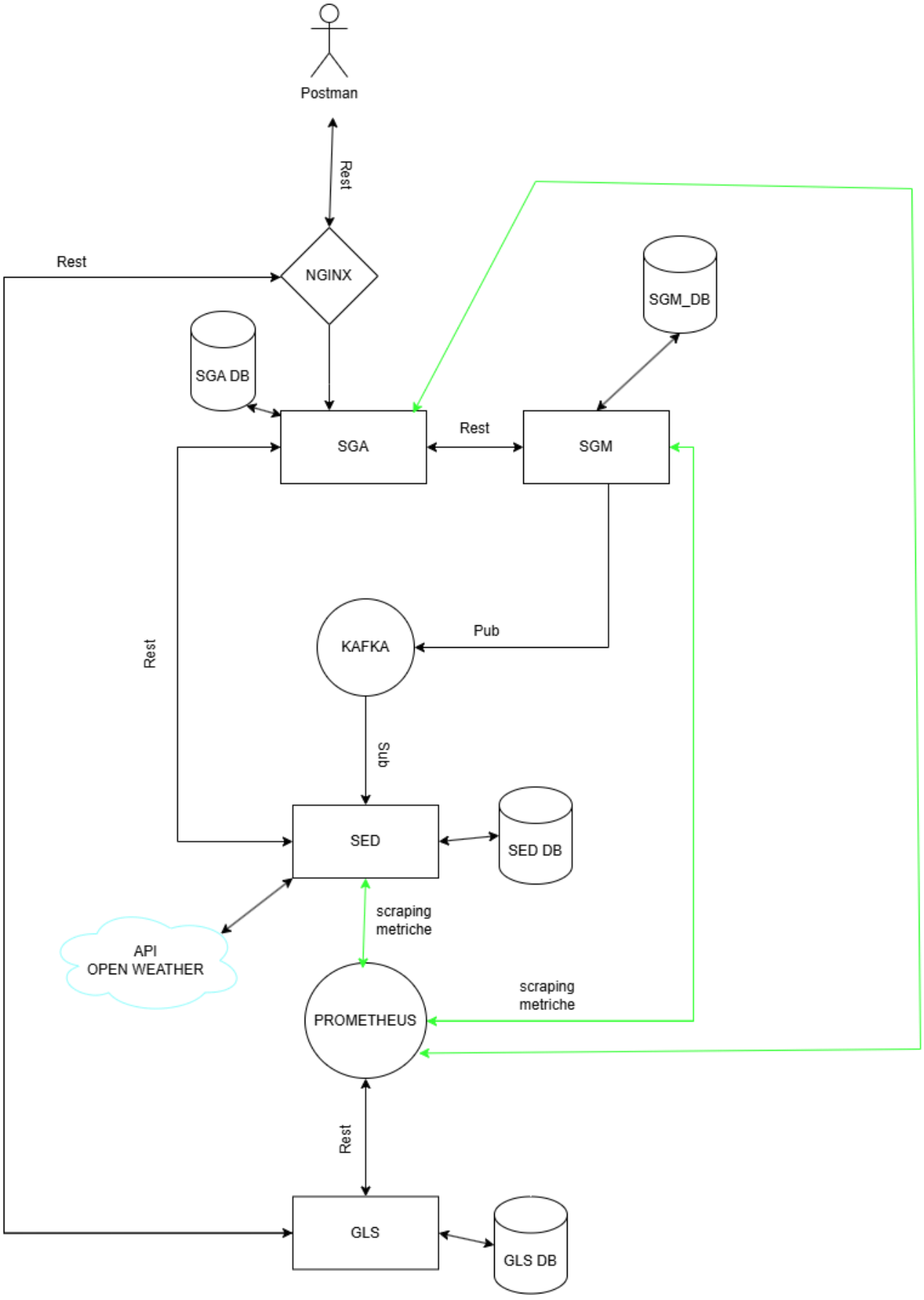
Sommario

1. ARCHITETTURA DEL SISTEMA



1.1 Microservizio SGA - Servizio Gestione Account

Implementato come microservizio indipendente, SGA fornisce un'API RESTful che consente agli utenti di registrarsi, autenticarsi e gestire il proprio account.

Funzionalità principali

1. Gestione account utente

- **Registrazione utente**: Creazione di nuovi account con validazione dell'email
- **Autenticazione**: Verifica delle credenziali e generazione di token JWT
- **Eliminazione account**: Rimozione completa dei dati utente dal sistema

2. Servizi di autenticazione

- **Emissione token JWT**: Creazione di token sicuri con scadenza impostata a 3 giorni
- **Verifica token**: Validazione delle richieste autenticate
- **Cifratura password**: Hashing SHA-256 per proteggere le credenziali utente

3. API di supporto agli altri microservizi

- **Risoluzione ID-email**: Traduzione bidirezionale tra ID utenti e indirizzi email
- **Verifica identità**: Supporto per SGM e SED nella verifica dell'identità degli utenti

Endpoint API:

- /registrazione (POST): Crea un nuovo account utente
- /login (POST): Autentica un utente e genera un token JWT
- /elimina_account (POST): Rimuove un account utente
- /utente/<id>/email (GET): Recupera l'email associata a un ID
- /utente/email/<email> (GET): Recupera l'ID associato a un'email
- /metriche (GET): Espone le metriche di monitoraggio per Prometheus

1.2 Microservizio SGM - Servizio Gestione Monitoraggio

Funge da intermediario tra l'interfaccia utente e il sistema di elaborazione dati meteorologici.

Funzionalità principali:

1. **Gestione delle regole di monitoraggio:**
   - **Creazione e aggiornamento delle regole**: Consente agli utenti di definire parametri meteorologici da monitorare per specifiche località
   - **Archiviazione delle configurazioni**: Mantiene le configurazioni personalizzate per ogni utente
   - **Eliminazione delle regole**: Permette agli utenti di rimuovere regole non più necessarie
2. **Elaborazione dei vincoli meteorologici:**
   - **Gestione parametri meteorologici**: Temperatura, umidità, pressione, nuvolosità, vento, precipitazioni
   - **Impostazione soglie**: Valori minimi e massimi per ciascun parametro
   - **Frequenza di controllo**: Personalizzazione del periodo di trigger per ogni regola
3. **Comunicazione asincrona:**
   - **Produzione messaggi Kafka**: Genera messaggi contenenti le regole da monitorare
   - **Consegna garantita**: Implementa meccanismi di conferma e retry per assicurare la consegna
   - **Pianificazione periodica**: Timer che attiva verifiche periodiche ogni 60 secondi

Endpoint API:

- /aggiorna_regole (POST): Crea o aggiorna regole meteo per una città
- /elimina_vincoli_utente (POST): Rimuove regole esistenti
- /mostra_regole (GET): Mostra le regole configurate dall'utente
- /metriche (GET): Espone metriche di monitoraggio

Schema database

- **Tabella città**: Archivia dati geografici delle località monitorate
  - id: Identificatore univoco (PK)
  - città: Nome della città
  - latitudine, longitudine: Coordinate geografiche della città
  - codice_postale, codice_stato
- **Tabella vincoli_utente**: Archivia le regole di monitoraggio
  - id: Identificatore univoco (PK)
  - id_utente: Riferimento all'utente (da SGA)
  - id_città: Riferimento alla città monitorata (FK)
  - regole: Struttura JSON con i parametri da monitorare
  - timestamp: Data/ora dell'ultimo aggiornamento
  - periodo_trigger: Intervallo di controllo in secondi

    - controllato: Flag che indica lo stato di elaborazione

Metriche di monitoraggio

- richieste_SGM: Contatore delle richieste ricevute
- richieste_SGM_Fallite: Contatore degli errori
- messaggi_kafka_SGM: Contatore dei messaggi prodotti
- regole_attive_SGM: Gauge delle regole attive nel sistema
- tempo_di_risposta_SGM: Latenza delle risposte API

Flusso operativo

1. L'utente si autentica attraverso SGA e riceve un token JWT
2. L'utente configura regole meteo attraverso l'endpoint /aggiorna_regole
3. SGM valida l'autenticazione e memorizza le regole nel database
4. Periodicamente (o su evento), SGM controlla le regole da elaborare
5. Le regole pronte vengono formattate e inviate a SED tramite Kafka
6. SED elabora le regole e notifica eventuali violazioni
7. L'utente può visualizzare/modificare le proprie regole in qualsiasi momento

1.3 Microservizio SED - Servizio Elaborazione Dati

Microservizio che si occupa del monitoraggio e della notifica delle condizioni meteorologiche in base alle regole definite dagli utenti.

Funzionalità principali

- **Monitoraggio meteorologico**: Interroga l'API OpenWeather per ottenere dati meteo in tempo reale
- **Verifica dei vincoli**: Confronta i dati meteorologici con le regole definite dagli utenti
- **Gestione delle notifiche**: Rileva le violazioni e invia notifiche via email agli utenti

Architettura interna

Il SED è organizzato in diversi componenti:

1. **Consumer Kafka**:
    - Sottoscrive il topic aggiornamento_eventi
    - Riceve regole di monitoraggio dal servizio SGM
    - Memorizza le regole ricevute nella tabella eventi_da_monitorare
2. **Monitoraggio**:
    - Processa le regole non ancora controllate
    - Interroga l'API OpenWeather per le condizioni meteo attuali
    - Verifica se i parametri meteo (temperatura, umidità, pressione, ecc.) violano i vincoli
3. **Thread Notificatore**:
    - Controlla periodicamente gli eventi da notificare
    - Richiede al servizio SGA le email degli utenti
    - Invia notifiche via email per le violazioni rilevate

Database

Il SED utilizza un database MySQL (SED_db) con due tabelle principali:

- eventi_da_monitorare: memorizza le regole da verificare
- eventi_da_notificare: registra le violazioni rilevate in attesa di notifica

Monitoraggio

Il SED espone metriche Prometheus per il monitoraggio operativo:

- Numero di richieste all'API OpenWeather
- Percentuale di richieste fallite
- Tempo di risposta dell'API
- Numero di eventi in attesa di notifica
- Statistiche sulle notifiche inviate/fallite

1.4 Microservizio GLS - Gestione Livello Di Servizio

Panoramica

Microservizio che si occupa del monitoraggio, dell'analisi e della previsione delle metriche operative dell'intero sistema. A differenza degli altri microservizi che gestiscono direttamente i dati meteorologici, il GLS si concentra sulla salute e sulle prestazioni del sistema, fornendo strumenti di monitoraggio avanzati e capacità predittive.

Funzionalità principali

1. **Autenticazione amministrativa:**
    - **Login dedicato**: Accesso riservato agli amministratori del sistema
    - **Sicurezza JWT**: Autenticazione basata su token con scadenza configurabile
    - **Gestione account admin**: Possibilità di aggiungere/rimuovere amministratori
2. **Monitoraggio delle metriche:**
    - **Registrazione di metriche**: Definizione delle metriche da monitorare con relativi target
    - **Configurazione dei parametri**: Impostazione di soglie minime e massime accettabili
    - **Verifica stato corrente**: Controllo in tempo reale delle violazioni
3. **Analisi storica delle violazioni:**
    - **Reportistica multi-periodo**: Analisi delle violazioni nelle ultime 1, 3 e 6 ore
    - **Statistiche di conformità**: Percentuale di tempo in cui le metriche violano i target
    - **Identificazione di pattern**: Rilevamento di trend problematici nel tempo
4. **Previsione delle violazioni future:**
    - **Modelli predittivi**: Utilizzo di algoritmi di forecasting (Exponential Smoothing)
    - **Visualizzazione grafica**: Generazione di grafici con dati storici e previsioni
    - **Stima probabilistica**: Calcolo della probabilità di violazioni future

Endpoint API

- /adminlogin (POST): Autenticazione amministrativa
- /aggiorna_metriche (POST): Crea/modifica metriche monitorate
- /elimina_metriche (POST): Rimuove metriche dal monitoraggio
- /stato_metriche (GET): Visualizza lo stato attuale delle metriche
- /violazioni_metriche (GET): Analizza violazioni storiche
- /previsione_violazioni (GET): Genera previsioni e grafici per future violazioni

Schema database

- **Tabella metriche**:
    - id: Identificatore univoco
    - nome: Nome della metrica (corrispondente a quella in Prometheus)
    - valore_target_min: Soglia minima accettabile
    - valore_target_max: Soglia massima accettabile
    - periodo: Parametro di stagionalità per il modello predittivo
- **Tabella admins**:
    - id: Identificatore univoco
    - email: Email dell'amministratore
    - password: Hash SHA-256 della password

Flusso operativo

1. L'amministratore si autentica tramite /adminlogin
2. Configura le metriche da monitorare con soglie appropriate
3. Utilizza l'endpoint /stato_metriche per verificare lo stato corrente
4. Analizza le tendenze con /violazioni_metriche
5. Genera previsioni future tramite /previsione_violazioni specificando la metrica e l'orizzonte temporale

1.5 NGINX

Caratteristiche e funzionalità di NGINX

NGINX rappresenta un web-server open-source ad alte prestazioni, ampiamente utilizzato nel campo dei sistemi distribuiti anche come reverse proxy. La sua popolarità è dovuta principalmente a due caratteristiche fondamentali: il basso consumo di risorse e l'elevata capacità di gestire richieste concorrenti.

Diversamente dai web server tradizionali che creano un nuovo processo o thread per ogni richiesta in arrivo, NGINX implementa un approccio event-driven (basato sugli eventi) e asincrono. Questo paradigma permette di gestire migliaia di connessioni simultanee all'interno di un singolo thread worker, ottimizzando drasticamente l'utilizzo delle risorse e aumentando la scalabilità del sistema.

Implementazione in Weather Sentinel

Nel contesto del progetto Weather Sentinel, NGINX è stato implementato come API Gateway, diventando il punto di ingresso unificato per tutte le richieste degli utenti verso i diversi microservizi del sistema. Questa configurazione garantisce due importanti proprietà di trasparenza:

1. **Location Transparency (Trasparenza di locazione)**: Gli utenti non necessitano di conoscere la posizione fisica o l'indirizzo specifico dei vari microservizi. Tutte le richieste vengono indirizzate a un unico endpoint NGINX, che si occupa dell'instradamento appropriato.
2. **Partition Transparency (Trasparenza di partizione)**: La suddivisione del sistema in microservizi risulta completamente invisibile agli utenti finali, che percepiscono Weather Sentinel come un'unica applicazione coesa.

Flusso operativo

Il processo operativo di NGINX all'interno dell'architettura Weather Sentinel segue questo schema:

1. Il client invia una richiesta REST API all'endpoint pubblico esposto da NGINX
2. NGINX riceve e analizza la richiesta
3. In base al pattern dell'URL e alla natura della richiesta, NGINX determina quale microservizio deve elaborarla

1.6 Apache Kafka: Sistema di Messaggistica Distribuita

Principi fondamentali

Apache Kafka rappresenta un sistema di messaggistica distribuita basato sul pattern publish-subscribe, progettato per offrire elevate prestazioni, alta scalabilità e persistenza durevole dei messaggi. La sua architettura è ottimizzata per gestire flussi di dati in tempo reale, consentendo la comunicazione asincrona tra componenti distribuiti senza necessità di accoppiamento diretto.

Nel paradigma di Kafka, ogni entità che interagisce con il sistema è classificata come client, che può assumere il ruolo di publisher (produttore), subscriber (consumatore), o entrambi contemporaneamente in base alle funzionalità svolte.

Modello di comunicazione

Il flusso comunicativo in Kafka segue questi principi:

1. I **publisher** generano messaggi e li pubblicano su specifici **topic**, che fungono da canali logici di categorizzazione
2. I **subscriber** si registrano ai topic di interesse e ricevono automaticamente tutti i messaggi pubblicati su di essi
3. La comunicazione avviene in modo **asincrono e indiretto**, eliminando la necessità che i processi comunicanti conoscano reciprocamente i propri indirizzi IP o endpoint
4. Ogni client stabilisce una connessione diretta esclusivamente con il **broker Kafka**, che gestisce l'inoltro e la persistenza dei messaggi

Struttura dei messaggi

I messaggi in Kafka sono strutture flessibili che possono contenere dati di qualsiasi natura o formato. Questa caratteristica conferisce elevata versatilità, lasciando ai client la responsabilità di definire lo schema e la struttura dei dati trasmessi.

Implementazione in Weather Sentinel

Nel contesto del progetto Weather Sentinel, Kafka svolge un ruolo cruciale come backbone comunicativo tra i diversi microservizi, permettendo lo scambio di informazioni in modo disaccoppiato e resiliente. La comunicazione asincrona garantisce che i servizi possano operare indipendentemente, migliorando la robustezza complessiva del sistema.

I flussi principali di comunicazione tramite Kafka includono:

- Il **SGM** (Sistema Gestione Monitoraggio) pubblica messaggi relativi ai vincoli meteorologici configurati dagli utenti
- Il **SED** (Sistema Elaborazione Dati) sottoscrive questi messaggi per elaborare i dati meteorologici e verificare eventuali violazioni

1.7 Prometheus: Sistema di Monitoraggio e Osservabilità

Caratteristiche fondamentali

Prometheus è un potente sistema open source dedicato al monitoraggio e all'osservabilità delle applicazioni, progettato specificamente per ambienti distribuiti e cloud-native. La sua architettura è ottimizzata per la raccolta, l'archiviazione e l'interrogazione di dati metrici in formato time series, offrendo una visione approfondita sullo stato e sulle prestazioni dei sistemi monitorati.

Uno dei principali punti di forza di Prometheus risiede nella sua capacità di conservare cronologicamente tutte le metriche raccolte, permettendo non solo il monitoraggio in tempo reale ma anche l'analisi storica dell'andamento delle prestazioni, fondamentale per l'identificazione di pattern e tendenze.

Modello di raccolta dati

Prometheus implementa un modello di raccolta dati basato sul "pull", in cui è il server Prometheus stesso ad occuparsi attivamente del recupero delle metriche. Questo approccio differisce significativamente dai sistemi tradizionali basati sul "push", in cui sono i componenti monitorati a inviare autonomamente i propri dati.

Nel contesto di Weather Sentinel, ogni microservizio (SGA, SGM, SED, GLS) espone un endpoint /metrics dedicato, contenente le metriche specifiche relative al proprio funzionamento. Prometheus esegue periodicamente, secondo intervalli configurabili, operazioni di "scraping" di queste metriche attraverso semplici richieste REST API agli endpoint esposti dai servizi.

Tipologie di metriche e interrogazione

Prometheus supporta diverse tipologie di metriche:

- **Counter**: valori cumulativi che possono solo incrementare (es. numero totale di richieste elaborate)
- **Gauge**: valori che possono sia aumentare che diminuire (es. utilizzo di memoria, connessioni attive)
- **Histogram**: campionamento di osservazioni in bucket predefiniti (es. latenza delle richieste)
- **Summary**: simile all'histogram ma con calcolo di quantili specifici

Tutte le metriche raccolte vengono archiviate nel database interno di Prometheus e sono interrogabili tramite PromQL (Prometheus Query Language), un potente linguaggio di interrogazione specifico per l'analisi di dati time series. Questo meccanismo consente di accedere alle metriche tramite l'API REST esposta da Prometheus stesso, facilitando l'integrazione con sistemi di visualizzazione come Grafana.

Implementazione in Weather Sentinel

Nel progetto Weather Sentinel, Prometheus rappresenta il nucleo centrale del sistema di monitoraggio, permettendo di:

1. Tracciare il numero di richieste ricevute da ciascun microservizio
2. Monitorare i tempi di risposta e l'efficienza delle elaborazioni
3. Rilevare anomalie nel comportamento del sistema
4. Generare alert in caso di soglie critiche superate
5. Fornire dati storici per l'analisi delle prestazioni nel tempo

2. SCELTE PROGETTUALI

2.1 Fault Tolerance

Principi e implementazione

In un'architettura a microservizi come Weather Sentinel, ogni componente è progettato per essere autonomo e indipendente. Tuttavia, questa indipendenza non elimina il rischio che il malfunzionamento di un singolo servizio possa innescare un effetto domino, compromettendo l'intero sistema. L'implementazione di meccanismi di fault tolerance rappresenta quindi una priorità per garantire che il sistema possa continuare a operare anche in presenza di guasti parziali.

Nel progetto Weather Sentinel, la fault tolerance è stata realizzata principalmente attraverso l'integrazione di database MySQL dedicati per ciascun microservizio. Questa scelta progettuale si rivela particolarmente cruciale per il **SED** (Sistema Elaborazione Dati) e il suo componente **ThreadNotificatore**, che fungono rispettivamente da elaboratore dati e sistema di notifica.

Gestione della persistenza nei componenti critici

Il SED, in qualità di consumer di messaggi Kafka, rappresenta un punto critico per la resilienza del sistema. Una volta che ha prelevato i dati meteorologici e analizzato i vincoli, esegue il commit su Kafka per segnalare l'avvenuta elaborazione. In questo momento, il SED diventa l'unico componente in possesso di queste informazioni, creando un potenziale single point of failure: un'interruzione in questa fase potrebbe comportare la perdita permanente dei dati elaborati.

Per mitigare questo rischio, è stato implementato un database MySQL dedicato al SED che memorizza i dati elaborati prima dell'esecuzione del commit su Kafka. Questo approccio preventivo consente di gestire efficacemente tre possibili scenari di fallimento:

1. **Fallimento dopo il commit su Kafka**: In caso di interruzione dopo aver eseguito il commit, il SED al riavvio può recuperare lo stato dal proprio database e riprendere normalmente le operazioni.
2. **Fallimento prima della memorizzazione nel database**: Se il SED si interrompe prima di salvare i dati nel database, Kafka stesso risolve il problema non avendo ricevuto conferma del commit. Al riavvio del SED, Kafka reinvierà automaticamente lo stesso messaggio.
3. **Fallimento tra la memorizzazione nel database e il commit su Kafka**: Questo scenario rappresenta il caso più delicato, poiché al riavvio il SED elaborerà prima il lavoro in sospeso dal database e successivamente riceverà nuovamente lo stesso messaggio da Kafka, portando a una doppia elaborazione.

Semantica di consegna e gestione delle duplicazioni

La scelta implementativa fondamentale risiede nell'adozione della semantica **"at least once"** di Kafka, privilegiando la garanzia che ogni messaggio venga elaborato almeno una volta piuttosto che rischiare di perdere completamente un'informazione. Nel contesto di Weather Sentinel, questa decisione comporta che in circostanze rare si possano verificare duplicazioni nelle notifiche inviate all'utente, un compromesso accettabile rispetto all'alternativa di mancata notifica di condizioni meteorologiche critiche.

È importante sottolineare che il SED pubblica messaggi verso il ThreadNotificatore solo quando rileva effettive violazioni dei parametri meteorologici indicati dall'utente. Questo significa che se il messaggio duplicato non contiene violazioni, non si verificherà alcuna duplicazione nelle notifiche.

Tracciamento delle notifiche

Il ThreadNotificatore implementa un ulteriore livello di resilienza attraverso il campo sent nel proprio database, che registra lo stato di ogni notifica. Questo meccanismo offre molteplici vantaggi:

1. Mantiene uno storico completo delle comunicazioni inviate
2. Associa un timestamp preciso a ciascuna notifica
3. Consente il recupero di notifiche non inviate in caso di riavvio del servizio

Al riavvio, il ThreadNotificatore esegue automaticamente un controllo delle notifiche pendenti nel database, garantendo che nessuna comunicazione importante rimanga non consegnata anche in caso di interruzioni del servizio.

2.2 Gestione delle Repliche

Principi di replicazione nel sistema distribuito

L'architettura di Weather Sentinel è stata progettata nativamente per supportare la replicazione dei suoi microservizi, un aspetto fondamentale per garantire elevata disponibilità, distribuzione efficiente del carico di lavoro e scalabilità orizzontale. La strategia implementativa varia in base alla tipologia di servizio, tenendo conto delle specifiche responsabilità e requisiti di stato di ciascun componente.

Identificazione univoca delle repliche per SED e ThreadNotificatore

Nel caso del SED (Sistema Elaborazione Dati) e del ThreadNotificatore, è stato implementato un sistema di identificazione basato su codici univoci (id_monitoraggio) che vengono registrati come campi nei rispettivi database. Questo meccanismo garantisce che:

1. Ogni replica possa identificare e operare esclusivamente sulle entries di propria competenza
2. Le query di database vengano eseguite in modo più efficiente, filtrando solo i record pertinenti
3. Tutte le repliche possano condividere un unico database centralizzato senza conflitti

Questa soluzione è particolarmente efficace poiché questi componenti non necessitano di mantenere uno stato persistente tra le diverse operazioni: i loro database sono principalmente utilizzati come meccanismo di fault tolerance piuttosto che come repository di stato.

Replicazione stateless del SGM (Sistema Gestione Monitoraggio)

Il SGM rappresenta un caso più complesso, poiché ha la responsabilità di mantenere l'elenco aggiornato delle regole di monitoraggio configurate da ciascun utente. In questo contesto, l'utilizzo di un identificativo di istanza avrebbe compromesso la trasparenza di locazione (Location Transparency) del sistema, costringendo a indirizzare le richieste di un utente sempre alla stessa replica specifica.

Per risolvere questa problematica, è stato implementato un approccio basato su timestamp nel database condiviso. Ogni regola include un campo timestamp che registra il momento in cui è stata presa in carico da una qualsiasi istanza del SGM per la costruzione di un messaggio Kafka da pubblicare sul topic event_update.

Il processo di gestione delle regole segue questo flusso:

1. Ogni istanza del SGM verifica periodicamente l'esistenza di entries per cui sia trascorso il trigger_period dal valore del timestamp
2. In caso positivo, l'istanza costruisce un messaggio Kafka contenente i dati delle entries identificate
3. Il timestamp viene aggiornato solo dopo la ricezione dell'acknowledgment da Kafka

Prevenzione delle elaborazioni duplicate

Una sfida significativa nella gestione delle repliche è la possibilità che, durante l'attesa dell'acknowledgment da Kafka, un'altra istanza del SGM possa accedere alle stesse entries e generare messaggi duplicati. Questo scenario potrebbe portare all'invio di notifiche duplicate agli utenti.

Per prevenire questa situazione, è stato introdotto un campo booleano *controllato* nella tabella *vincoli_utente* del database del SGM. Questo campo funge da semaforo, segnalando che una specifica entry è già stata presa in carico da un'istanza e un messaggio Kafka è in attesa di conferma.

Nessun'altra istanza elaborerà la stessa entry fino alla ricezione dell'acknowledgment e al conseguente reset del campo checked.

Replicazione del SGA con autenticazione stateless

Anche il SGA (Sistema Gestione Autenticazione) è stato progettato per supportare la replicazione attraverso un database condiviso tra tutte le sue istanze. L'elemento chiave che facilita questa replicazione è l'implementazione di un meccanismo di autenticazione basato su token JWT (JSON Web Token).

I token JWT vengono inclusi nell'header HTTP di ogni richiesta proveniente dal client, permettendo a qualsiasi replica del SGA di:

1. Autenticare l'utente in modo indipendente, senza bisogno di accesso a informazioni di sessione
2. Verificare l'integrità e l'autenticità del token senza comunicare con altre istanze
3. Eliminare la necessità di gestire sessioni distribuite, che rappresenterebbero un potenziale collo di bottiglia

3. Monitoraggio QoS

Il monitoraggio della Quality of Service (QoS) rappresenta un aspetto fondamentale di Weather Sentinel, garantendo il controllo costante delle prestazioni, dell'affidabilità e dell'utilizzo delle risorse. L'architettura di monitoraggio è basata su un server Prometheus, configurato per raccogliere e archiviare in modo persistente le metriche esposte dai vari microservizi. Queste metriche vengono successivamente rese disponibili per l'analisi da parte del microservizio GLS

Le metriche monitorate in Weather Sentinel sono organizzate sotto forma di serie temporali e si suddividono in tre categorie principali, ciascuna con scopi specifici di osservabilità:

Metriche di performance

Queste metriche forniscono informazioni dettagliate sui tempi di esecuzione e risposta dei vari componenti:

- **Tempi di esecuzione delle query ai database**, differenziati per microservizio (SGA, SGM, SED, GLS)
- **Tempi di risposta alle richieste client** da parte del SGM e del SGA
- **Latenza nelle notifiche utente**, misurata come il tempo intercorso tra la rilevazione di condizioni meteorologiche avverse e l'invio della notifica
- **Tempi di risposta del servizio esterno OpenWeather** per valutare la qualità dell'integrazione con API di terze parti

Metriche sullo stato di salute del sistema

Queste metriche forniscono una panoramica complessiva sul funzionamento del sistema:

- **Numero di richieste pervenute** al SGA e al SGM
- **Conteggio degli errori HTTP** classificati per codice di risposta
- **Numero di notifiche inviate** agli utenti
- **Errori verificatisi durante il processo di invio notifiche**
- **Richieste effettuate al servizio OpenWeather**
- **Numero di regole attive** nel SGM
- **Utenti registrati** nel sistema
- **Comunicazioni tra ThreadNotificatore e SGA**
- **Comunicazioni tra SGM e SGA**
- **Messaggi pubblicati su Kafka** da parte del SGM
- **Messaggi pubblicati su Kafka** da parte del SED
- **ACK ricevuti dal SGM** da parte di Kafka
- **ACK ricevuti dal SED** da parte di Kafka

Metriche di monitoraggio delle risorse dei container

Per monitorare le risorse utilizzate dai container, è stato integrato cAdvisor, uno strumento sviluppato da Google che produce numerose metriche relative all'hardware e alle prestazioni di rete. Le principali metriche raccolte includono:

- **container_memory_usage_bytes**: misurazione dell'utilizzo attuale della memoria per ogni container
- **container_cpu_system_seconds_total**: tempo cumulativo di esecuzione della CPU per ogni container
- **container_memory_cache**: bytes allocati nella memoria cache per container
- **container_network_transmit_bytes_total**: volume totale di dati trasmessi da ciascun container
- **container_network_receive_bytes_total**: volume totale di dati ricevuti da ciascun container

Metriche aggregate con Prometheus Rules

Il server Prometheus è stato configurato per calcolare metriche aggregate personalizzate attraverso le Prometheus Rules. Queste regole, specificate nel file di configurazione, generano le seguenti metriche derivate:

- **Richieste client completate con successo**, sia per il SGM che per il SGA
- **Richieste verso OpenWeather completate con successo** da parte del SED
- **Rate di richieste nell'ultima ora** per valutare il carico del sistema
- **Percentuali di successo delle richieste** verso i diversi componenti (SGM, SGA, OpenWeather)
- **Statistiche sui tempi di esecuzione delle query** (media, mediana, 90° percentile) per ogni microservizio con database

Forecasting

Il GLS espone un endpoint dedicato al forecasting delle metriche, consentendo di ottenere una previsione dell'andamento futuro per i prossimi minuti. L'API accetta come parametri della richiesta GET la metrica di interesse e l'orizzonte temporale desiderato in minuti.

L'implementazione del forecasting segue una metodologia rigorosa:

1. **Prelievo dei dati** relativi alla metrica selezionata dall'API di Prometheus
2. **Data Manipulation**:
    - Resample della serie temporale alla frequenza di scraping di Prometheus
    - Interpolazione per eliminare eventuali valori nulli nella serie
3. **Splitting dei dati** in training set (90%) e test set (10%)
4. **Addestramento del modello** sui dati di training
5. **Generazione delle previsioni** per l'orizzonte temporale richiesto

4. Deployment del Progetto

4.1 Docker

Step 1: Configurare nella directory docker il file .env con le relative password e API_KEY per il servizio openweather

.env M

docker > .env

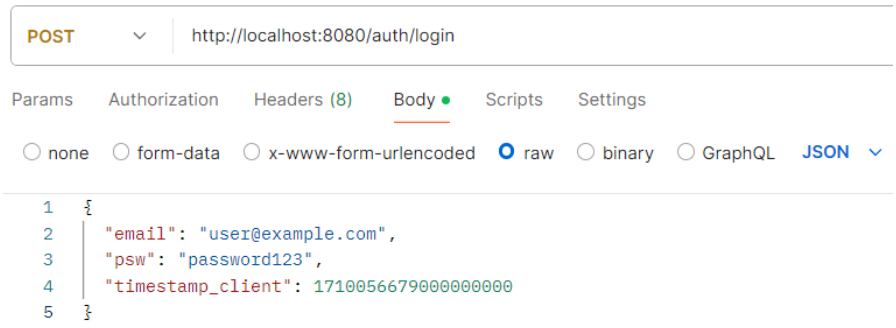
```
SGA_DB_PASSWORD="1234"
SGM_DB_PASSWORD = "1234"
SGA_DB_PASSWORD="1234"
GLS_DB_PASSWORD = "1234"
ADMIN_EMAIL: "admin@weathersentinel.com"
ADMIN_PASSWORD="1234"

OPENWEATHER_API_KEY="kncdkxzck343rlxdsvakxzln3rew"
```

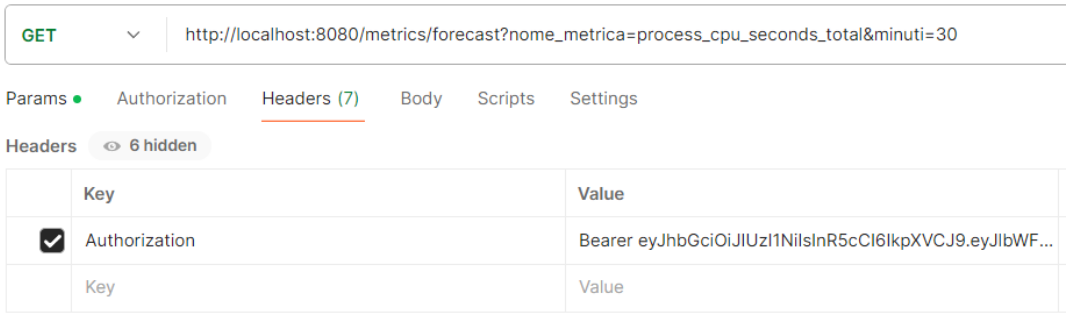
Step 2: Aprire un terminale nella directory docker ed eseguire il commando

```
docker-compose up -d
```

Step 3: Richieste tramite Postman all'API Gateway: http://localhost:8080

- Tutte le route sono mappate in nginx/nginx.conf





4.2 Kubernetes

1. Creazione del Namespace

```
# Verifica se il namespace esiste
kubectl get namespace weather-sentinel
# Se non esiste, crealo
kubectl create namespace weather-sentinel
```

2. Creazione ConfigMap e Secrets

```
# Configmap per i servizi
kubectl apply -f kubernetes/configmaps/sga-configmap.yaml
kubectl apply -f kubernetes/configmaps/sgm-configmap.yaml
kubectl apply -f kubernetes/configmaps/sed-configmap.yaml
kubectl apply -f kubernetes/configmaps/gls-configmap.yaml
# Secret per GLS Admin
kubectl apply -f kubernetes/secrets/gls-admin-secret.yaml
# Secret per le password dei database
kubectl apply -f kubernetes/secrets/app-secrets.yaml
```

3. Creazione PersistentVolumes e PersistentVolumeClaims

```
# Creazione dei PV per i database
kubectl apply -f kubernetes/persistence/sga-db-pv.yaml
kubectl apply -f kubernetes/persistence/sgm-db-pv.yaml
kubectl apply -f kubernetes/persistence/sed-db-pv.yaml
kubectl apply -f kubernetes/persistence/gls-db-pv.yaml
# Creazione dei PVC
kubectl apply -f kubernetes/persistence/sga-db-pvc.yaml
kubectl apply -f kubernetes/persistence/sgm-db-pvc.yaml
kubectl apply -f kubernetes/persistence/sed-db-pvc.yaml
kubectl apply -f kubernetes/persistence/gls-db-pvc.yaml
```

4. Creazione dei Servizi per i Database

```
kubectl apply -f kubernetes/services/sga-db-service.yaml
kubectl apply -f kubernetes/services/sgm-db-service.yaml
kubectl apply -f kubernetes/services/sed-db-service.yaml
kubectl apply -f kubernetes/services/gls-db-service.yaml
```

5. Creazione dei Database

```
kubectl apply -f kubernetes/deployments/sga-db-deployment.yaml
kubectl apply -f kubernetes/deployments/sgm-db-deployment.yaml
kubectl apply -f kubernetes/deployments/sed-db-deployment.yaml
kubectl apply -f kubernetes/deployments/gls-db-deployment.yaml
# Monitora lo stato
kubectl get pods -n weather-sentinel -l "app in (sga-db,sgm-db,sed-db,gls-db)" -w
```

6. Creazione dell'Infrastruttura di Supporto (Zookeeper, Kafka, Prometheus)

```
# Zookeeper

#creazione volumi
kubectl apply -f kubernetes/persistence/zookeeper-pv.yaml
kubectl apply -f kubernetes/persistence/zookeeper-pvc.yaml

kubectl apply -f kubernetes/services/zookeeper-service.yaml
kubectl apply -f kubernetes/deployments/zookeeper-deployment.yaml

# Kafka
#creazione volumi
kubectl apply -f kubernetes/persistence/kafka-pv.yaml
kubectl apply -f kubernetes/persistence/kafka-pvc.yaml

kubectl apply -f kubernetes/services/kafka-service.yaml
kubectl apply -f kubernetes/deployments/kafka-deployment.yaml
# Prometheus

#configmap
kubectl apply -f kubernetes/configmaps/prometheus-config.yaml
kubectl apply -f kubernetes/configmaps/prometheus-rules.yaml
kubectl apply -f kubernetes/services/prometheus-rbac.yaml

kubectl apply -f kubernetes/services/prometheus-service.yaml
kubectl apply -f kubernetes/deployments/prometheus-deployment.yaml
```

Configura Port-Forward per accedere all'interfaccia

```
# Port-forwarding per rendere Prometheus accessibile da localhost
kubectl port-forward -n weather-sentinel service/prometheus 9090:9090
```

Mantieni questo comando in esecuzione in un terminale separato e apri http://localhost:9090 nel browser.

```
# Monitora lo stato
kubectl get pods -n weather-sentinel -l "app in (zookeeper,kafka,prometheus)" -w
```

7. Creazione dei Servizi per i Microservizi

```
kubectl apply -f kubernetes/services/sga-service.yaml
kubectl apply -f kubernetes/services/sgm-service.yaml
kubectl apply -f kubernetes/services/sed-service.yaml
kubectl apply -f kubernetes/services/gls-service.yaml
```

8. Verifica e Costruzione delle Immagini Docker

```
# Verifica quali immagini devono essere costruite
minikube ssh -- docker images | grep -E 'sga|sgm|sed|gls'

# Costruisci le immagini mancanti
```

collegare minikube a docker ➔ minikube -p minikube docker-env | Invoke-Expression

```
docker build -t sga_service_image:latest -f SGA/Dockerfile .
docker build -t sgm_service_image:latest -f SGM/Dockerfile .
docker build -t sed_service_image:latest -f SED/Dockerfile .
docker build -t gls_service_image:latest -f GLS/Dockerfile .
# Carica le immagini in Minikube
minikube image load sga_service_image:latest
minikube image load sgm_service_image:latest
minikube image load sed_service_image:latest
minikube image load gls_service_image:latest
```

9. Deployment dei Microservizi

```
kubectl apply -f kubernetes/deployments/sga-deployment.yaml
kubectl apply -f kubernetes/deployments/sgm-deployment.yaml
kubectl apply -f kubernetes/deployments/sed-deployment.yaml
kubectl apply -f kubernetes/deployments/gls-deployment.yaml
# Monitora lo stato
kubectl get pods -n weather-sentinel -l "app in (sga-service,sgm-service,sed-
service,gls-service)" -w
```

10. Configurazione Ingress

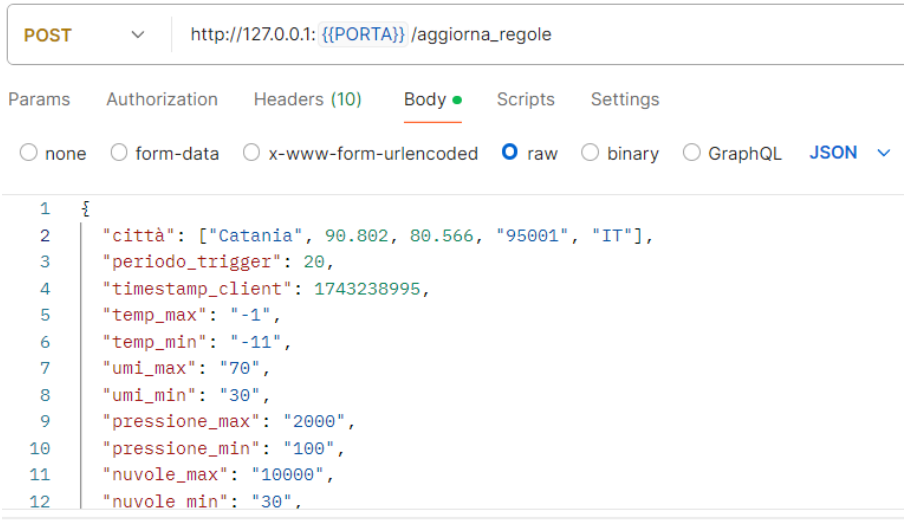
```
# Assicurati che l'addon Ingress sia abilitato ➔ ingress controller
minikube addons enable ingress
# Applica la configurazione Ingress
kubectl apply -f kubernetes/ingress/weather-sentinel-ingress.yaml
```

11. Verifica Complessiva dell'Infrastruttura

```
# Verifica tutti i pod
kubectl get pods -n weather-sentinel
# Verifica tutti i servizi
kubectl get svc -n weather-sentinel
```

```
# Verifica l'Ingress
kubectl get ingress -n weather-sentinel
```

ESEMPI DI CHIAMATA CON POSTMAN



Le route sono definete in:

kubernetes\ingress\weather-sentinel-ingress.yaml