

Weather Event Notifier

Elaborato DSDB

Mirco Antona

A.A. 2023/2024

REPOSITORY

<https://github.com/mircoantona1998/WeatherEventNotifier>

ABSTRACT

L'elaborato è pensato per offrire un servizio agli utenti di avviso mediante Mail, Telegram e/o notifica push su un applicazione mobile.

L'utente una volta fatto accesso all'applicazione mediante autenticazione username-password, può creare e gestire delle configurazioni.

Una configurazione è formata da:

- Latitudine della zona interessata
- Longitudine della zona interessata
- Metrica (temperatura massima, temperatura minima, probabilità di pioggia, ecc..) in accordo con quelle di OpenWeather
- Frequenza che indica ogni quanto l'applicazione deve fare verifica della condizione (ogni ora, ogni 2 ore, ogni 4 ore, ogni 8 ore, una volta al giorno)
- Valore per cui deve scattare la notifica
- Simbolo (<,>,<=,>=,==) che serve per confrontare la misura effettiva della metrica scelta con il valore di soglia

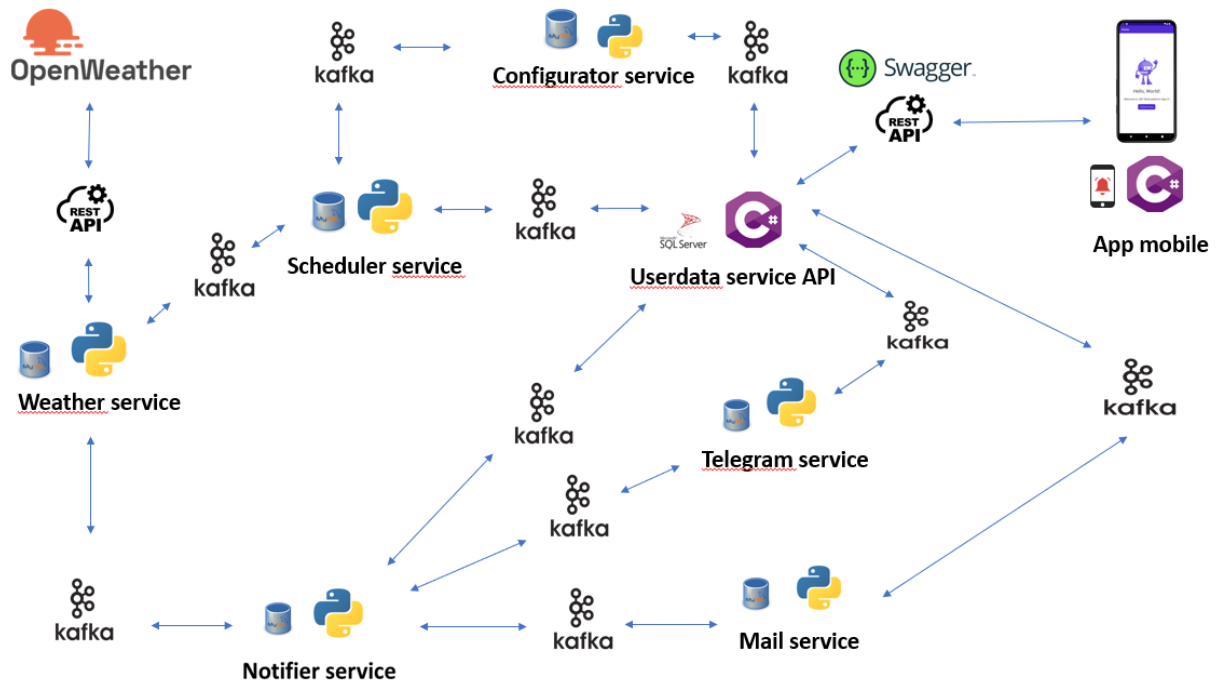
L'utente può scegliere di disattivare/attivare i canali su cui vuole ricevere le informazioni che sono 3:

- Telegram (passando un id che rappresenta il proprio chat_id)
- Mail (configurando nel proprio profilo la propria mail)
- Notifiche push (che riceverà all'interno dell'applicazione mobile nella sezione notifiche)

L'applicazione invierà con frequenza inserita nella configurazione, un messaggio sui canali configurati dell'utente, se la condizione della configurazione si verifica.

ARCHITETTURA:

L'architettura prevede un api gateway, microservizi, e un app mobile sviluppata solo per completezza.



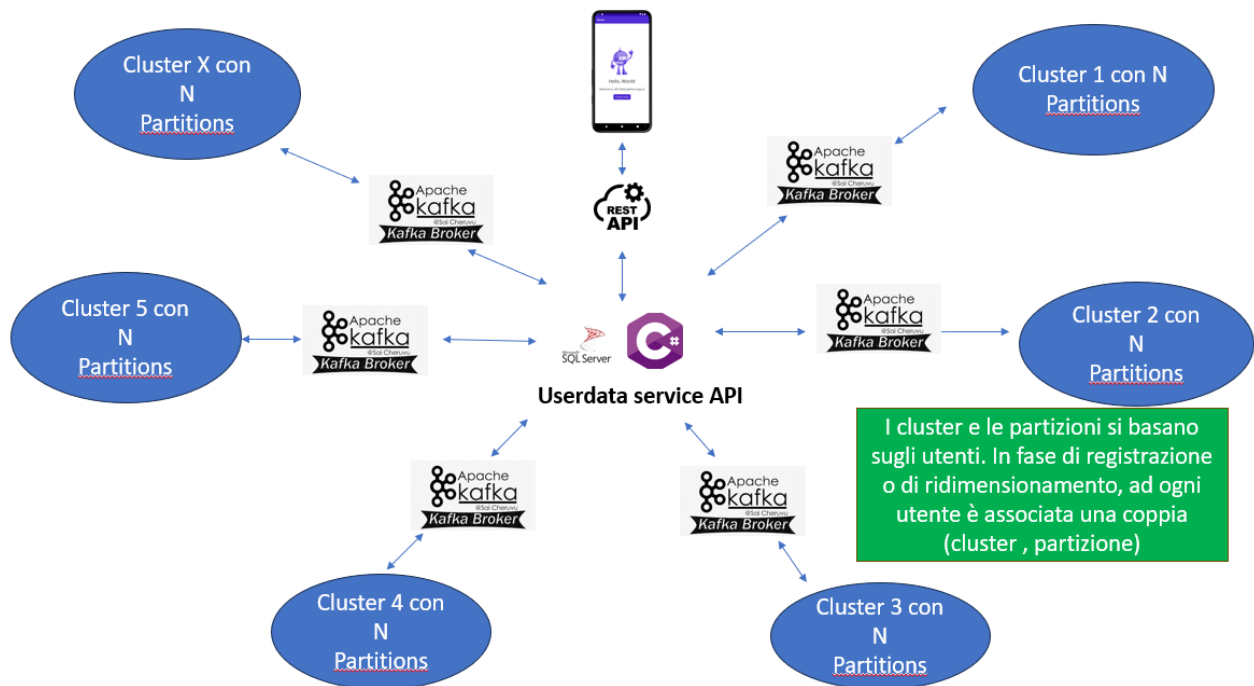
- **Userdata service API:** dotnet (aspNet.core) con modulo di autenticazione. Esso si appoggia su database Sql Server. Il suo compito è di rispondere alle richieste dell'utente in prima battuta e smistare verso i microservizi. Nel database a cui si appoggia vengono memorizzate solo le info degli utenti nella tabella Users utili per autenticazione
- **App mobile:** dotnet (Maui). App che utilizza utente.
- **Configurator service:** python su database mysql. Il suo compito è di memorizzare le info sulle metriche e le frequenze per restituirle all'utente e gestire le configurazioni
- **Scheduler service:** python su database mysql. Il suo compito è di preparare le schedulazioni, cioè una volta al giorno, sulla base delle configurazioni degli utenti, prepara i controlli sulle misure effettive che devono essere fatte in giornata. Ovviamente ne aggiunge quando durante l'arco della giornata vengono create nuove configurazioni. Durante il giorno poi, ogni ora attiva il processo di calcolo delle varie schedulazioni
- **Notifier service:** python con database mysql. Il suo compito è di gestire le schedulazioni che riceve dallo scheduler. Quando riceve una schedulazione, verifica la misura effettiva della metrica attraverso il weather service, e se capisce che deve scattare la notifica la genera e la gira ai microservizi per l'invio
- **Weather service:** python con database mysql. Il suo compito è di verificare la misura di una schedulazione e dare un responso al notifier se deve generare una notifica oppure no. Utilizza le api di OpenWeather
- **Telegram service:** python con database mysql. Il suo compito è di inviare un messaggio su telegram qualora riceve un messaggio dal notifier e utente in questione ha il canale attivo
- **Mail service:** python con database mysql. Il suo compito è di inviare una mail qualora riceve un messaggio dal notifier e utente in questione ha il canale mail attivo

La **comunicazione** fra i vari microservizi avviene attraverso Kafka su questi topic:

topic_to_scheduler,topic_to_weather:,topic_to_mail,topic_to_telegram,topic_to_notifier,topic_to_configuration,topic_to_userdata; mentre tra utente e api gateway Rest, come anche tra weather service e provider OpenWeather.

Processi di **scale up** e **scale down** sono gestiti da aumenti di cluster e partizioni:

Si basa sul concetto di “Cluster” come insieme di microservizi (configurator, scheduler, weather, notifier, telegram e mail) e “Partition” relativa alle partizioni kafka. Ogni utente avrà associato all’atto di registrazione oppure di scale up o down del sistema, una coppia (cluster,partition) che permette la gestione e l’affidamento dell’utente ad una parte del sistema.



PROCESSO DI FUNZIONAMENTO

Un utente registrandosi contatta **Userdataservice API**.

Registrato un utente viene salvato in DB con un "cluster" e una "partition" specifica affidatagli.

L'utente può:

- Configurare e abilitare una mail e una chat telegram per ricevere le notifiche
Userdataservice API contatta i microservizi "TelegramService" o "MailService" sul kafka del cluster dell'utente e sul topic kafka "topic_to_telegram" o "topic_to_mail" alla partizione dell'utente per gli aggiornamenti fatti dall'utente
- Visualizzare le notifiche sia da app o da rest API
Userdataservice API contatta il microservizio "NotifierService" sul kafka del cluster dell'utente e topic kafka "topic_to_notifier" alla partizione dell'utente per gli aggiornamenti fatti dall'utente
- Aggiungere e gestire delle configurazioni del meteo come descritto nell'abstract
Userdataservice API contatta il microservizio "ConfiguratorService" sul kafka del cluster dell'utente e topic kafka "topic_to_configurator" alla partizione dell'utente per gli aggiornamenti fatti dall'utente

Una volta che la configurazione dell'utente si trova nel database del Configurator, sarà attivato lo "SchedulerService" del cluster dell'utente sul topic kafka "topic_to_scheduler" alla partizione kafka dell'utente di preparare nella tabella "Schedule" tutte le righe che devono essere elaborate sulla base della frequenza della configurazione scelta dall'utente.

Sarà lo "SchedulerService" a inviare sul cluster kafka dell'utente al topic "topic_to_weather" alla partizione dell'utente un messaggio al "WeatherService" di elaborarlo e verificare se scatta la notifica o no.

Il "WeatherService" mediante verifica presenza previsione in db o chiamata rest al provider OpenWeather verifica se la schedulazione genera notifica oppure no. Se la genera invia un messaggio al "NotifierService" sul cluster kafka dell'utente al topic "topic_to_notifier" alla partizione dell'utente.

Il "NotifierService" salva la notifica in db e la smista al "TelegramService" o "MailService" sul cluster kafka dell'utente al topic "topic_to_telegram" o "topic_to_mail" alla partizione dell'utente.

LISTA DELLE API

La lista delle api è raggiungibile da swagger su “<http://localhost:8080/swagger/index.html>” una volta avviato docker compose

<http://localhost:8080/swagger/index.html>

Swagger
Select a definition: ExposeAPI v1

ExposeAPI WeatherEventNotifier ^{v1} ^{0.0.0}

<http://localhost:8080/swagger/v1/swagger.json>

Authorize

Auth

- POST /Auth/Registration
- POST /Auth/Login

Configuration

- GET /Configuration/Get
- POST /Configuration/Add
- PATCH /Configuration/Patch

Configuration/Delete

- DELETE /Configuration/Delete

Frequency

- GET /Frequency/Get

Mail

- GET /Mail/Get

Metric

- GET /Metric/Get

Notifier

- GET /Notifier/Get

Scheduler

- GET /Scheduler/Get

Telegram

- GET /Telegram/Get

UserMail

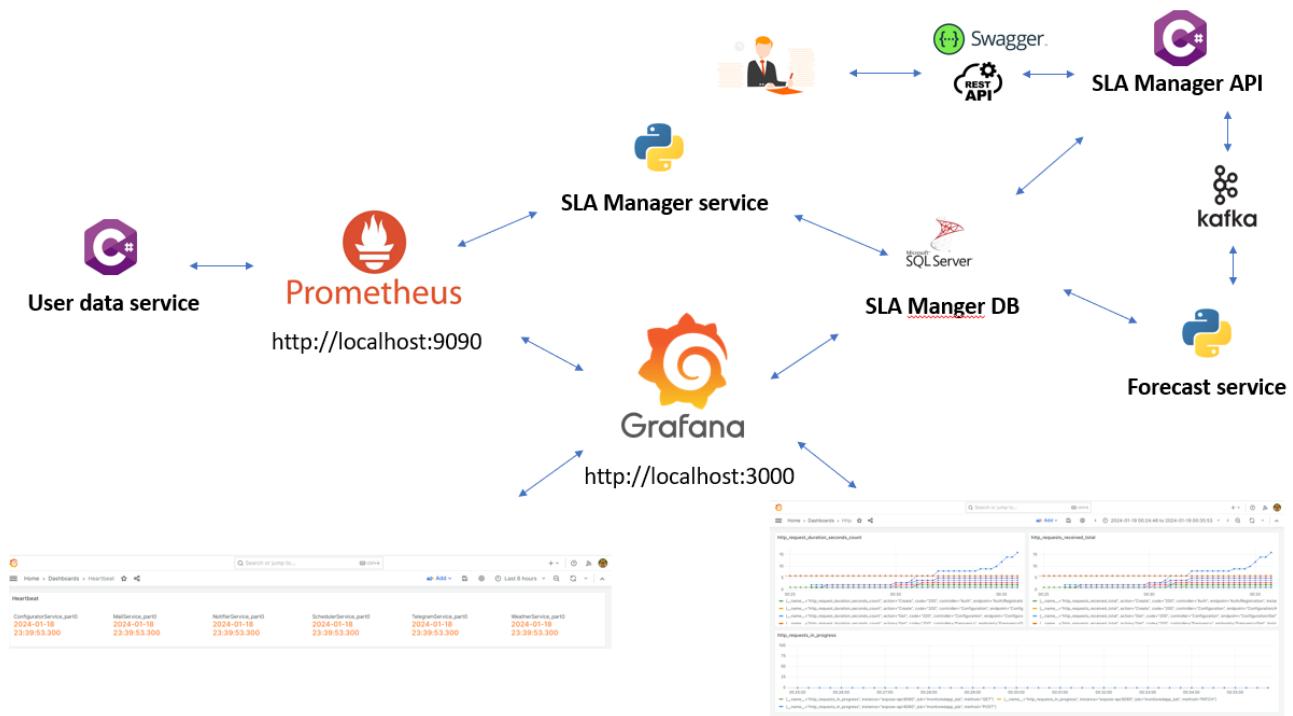
- GET /UserMail/Get
- POST /UserMail/Add
- PATCH /UserMail/Patch
- DELETE /UserMail/Delete

UserTelegram

- GET /UserTelegram/Get
- POST /UserTelegram/Add
- PATCH /UserTelegram/Patch
- DELETE /UserTelegram/Delete

SLA MANAGER

Per lo sla manager è stato sviluppato per monitorare le metriche esposte dall' **Userdata service API**.




Le metriche da prometheus vengono storicizzate in database Sql server attraverso un microservizio SLA Manager Service. SLA Manager service è responsabile dopo aver storicizzato una metrica di verificare se genera una violazione per le SLA configurate nel DB.

Si possono gestire le sla attraverso lo **Sla manager API**. Inoltre sono presenti:

- **Forecast Service**: per calcolare le probabilità di generazione violazione nei prossimi x minuti da parte di SLA
- **Grafana**: per visualizzare le metriche da prometheus e qualche pannello come ad esempio la raccolta di heartbeat da parte dei microservizi


<http://localhost:8081/swagger/index.html>

 Swagger
Empowering SMART BEAR

Select a definitionSLAManager v1

SLAManager WeatherEventNotifier ^{v1} OAS3

<http://localhost:8081/swagger/v1/swagger.json>

Authorize 

Auth ^

POST /Auth/Registration

POST /Auth/Login

Heartbeat ^

POST /Heartbeat/Send

MetricData ^

GET /MetricData/Get

MonitoringMetric ^

GET /MonitoringMetric/Get

Sla ^

GET /Sla/Get

POST /Sla/Add

PATCH /Sla/Patch

DELETE /Sla/Delete

SlaMetricStatus ^

GET /SlaMetricStatus/Get

SlaMetricViolation ^

GET /SlaMetricViolation/Get

GET /SlaMetricViolation/GetCount

SlaMetricViolationForecast ^

SlaMetricViolationForecast ^

GET /SlaMetricViolationForecast/Get

Status ^

GET /Status/Get

