Università degli studi di Catania

Distributed Systems & Big Data 2023/24

Relazione progetto di **Francesco Pandolfo**, matricola 1000008982

Etna events notifier

Abstract

L'applicazione, accedendo in modo esclusivo ai dati di proprietà dell'I.N.G.V di Catania, prevede due funzioni, entrambe utilizzabili tramite un interfaccia REST: la prima è quella di fornire serie temporali in formato Pandas Dataframe (Python) specificando un range di date (inizio e fine) oppure indicando l'ultimo intervallo temporale a partire dal momento in cui si fa la richiesta; la seconda funzione è quella di permettere agli utenti interessati di sottoscriversi agli eventi vulcanici dell'Etna, in particolare alla deformazione del suolo. Gli utenti in fase di sottoscrizione forniscono le constraint di deformazione (espressa in mm) e il tempo minimo per cui l'evento si debba verificare. Al verificarsi delle condizioni specificate il sistema restituisce all'utente un alert tramite email. A tale scopo, ad intervalli di 30 secondi, il sistema recupera la serie temporale di interesse all'utente e le elabora sulla base delle constraint specificate. La qualità del servizio offerto (QoS) è garantita grazie al monitorato costante eseguito dall'applicazione Prometheus sulla base di un *Service Level Agreement* configurabile che permette di definire uno SLA-set delle metriche da monitorare.

Descrizione dell'applicazione

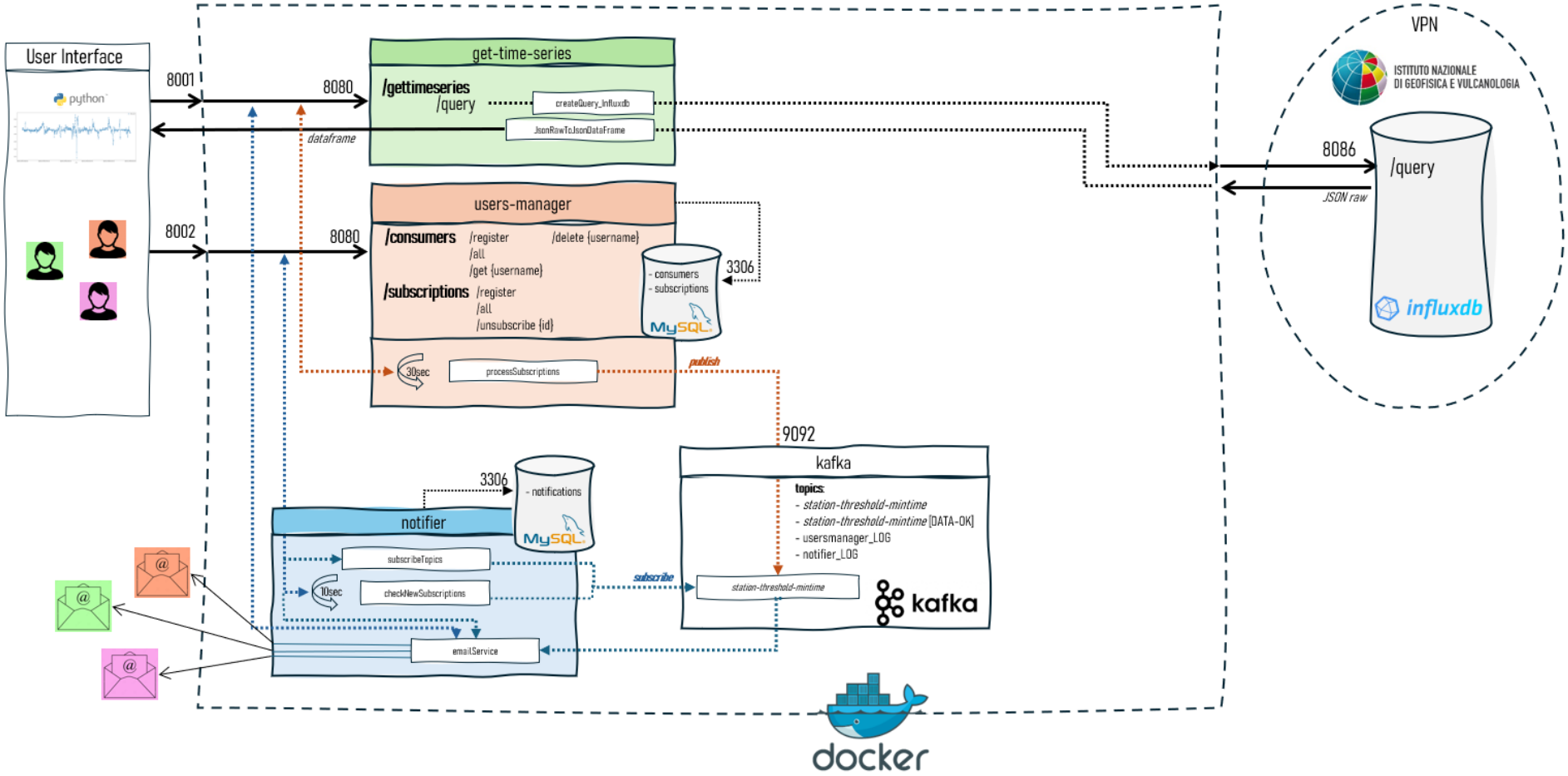
Architettura e tecnologie

L'applicazione è stata sviluppata su piattaforma **Docker**, composta pertando da più microservizi deployati come docker container. Il programma sviluppato all'interno di ciscun container è stato realizzato su framework **Spring**, il linguaggio scelto pertanto è **Java**. Per testare la prima funzione dell'applicazione è stata realizzato un breve script in **Python** per l'invio della richiesta di serie temporale e la successiva visualizzazione del Pandas dataframe ricevuto. Due dei container che compongono l'applicazione memorizzano i dati da loro gestiti in database **MySql** a cui accedono in maniera esclusiva. Per la gestione delle notifiche agli utenti iscritti al servizio è stato scelto di utilizzare il broker **Apache Kafka.**

Il monitoraggio delle performance del sistema è realizzato tramite **Prometheus**, e per una più rapida e ottimizzata visualizzazione delle metriche monitorate è stato inserto anche l'applicazione **Grafana**.

Prima parte – il sistema

Di seguito viene mostrato lo schema architetturale della parte del sistema che si occupa dell'acquisizione delle serie temporali memorizzate nel database **InfluxDB** (reso accessibile tramite VPN) di proprietà dell'Istituto Nazionale di Geofisica e Vulcanologia di Catania, nonché della gestione delle sottoscrizioni degli utenti e le relative notifiche:


User Interface
python
8001
8080
get-time-series
/gettimeseries
/query
createQuery_Influxdb
JsonRawToJsonDataFrame
dataframe
8002
8080
users-manager
/consumers
/register
/all
/get {username}
/delete {username}
/subscriptions
/register
/all
/unsubscribe {id}
- consumers
- subscriptions
MySQL
3306
30sec
processSubscriptions
publish
VPN
ISTITUTO NAZIONALE
DI GEOFISICA E VULCANOLOGIA
8086
/query
JSON raw
influxdb
9092
kafka
topics:
- station-threshold-mintime
- station-threshold-mintime [DATA-OK]
- usersmanager_LOG
- notifier_LOG
3306
- notifications
MySQL
notifier
subscribeTopics
10sec
checkNewSubscriptions
emailService
subscribe
station-threshold-mintime
kafka
docker

get-time-series

Il container denominato get-time-series ha lo scopo di ricevere richieste di serie temporali, creare la query necessaria per interrograre il database InfluDB ed inoltrargli la richiesta, creare un JSON sulla base di quello ricevuto da InfluxDB (chiamato JSON raw) in modo che sia acquisibile come Pandas dataframe a chi ne ha fatto richiesta. L'accessibilità a tale funzione è resa possibile dall'**esterno** tramite la porta **8001**, ma anche resa disponibile agli altri container all'**interno** di docker tramite la porta **8080.**

API

/gettimeseries/query

Unico endpoint realizzato per fare la richiesta di serie termporali

users-manager

Il container denominato users-manager ha lo scopo di gestire gli utenti che interndono usufruire del servizio di notifica dell'evento vulcanico. L'utente deve prima specificare il proprio **username** e contatto **email**, successivamente può inserire una o più stazione di interesse **stazione** GNSS che acquisisce la posizione del sito monitorato) specificando per ciascuna di esse la **soglia** espressa in millimetri e il **tempo minimo** che deve trascorrere affinchè la deformazione resti sopra soglia prima che scatti la notifica. Il container memorizza tali informazioni in un database MySql di uso esclusivo.

Ad intervalli di 30 secondi vengono processate tutte le sottoscrizioni presenti: per ciascuna di esse viene fatta una chiamata REST al container get-time-series con l'intervallo di tempo indicato nella sottoscrizione. Se dai dati ricevuti si verifica che la deformazione del suolo è sopra soglia per tutto il tempo indicato (non deve quindi essere presente nemmeno un'epoca in cui il valore scende sotto soglia) viene pubblicato un **topic** sul broker kafka, il cui nome è dinamico ed è costruito dalla seguente successione: ***stazione-soglia-tempo_minimo***; dal nome del topic si evince il fatto che se più utenti hanno interesse per la stessa triade non saranno pubblicati tanti topic quanti sono gli utenti, ma solamente uno: sarà il notificatore (altro container) a gestire gli utenti collegati alla medesima triade.

In maniera complementare al topic visto sopra, che notifica il superamento della soglia, viene inviato un topic con la stessa triade e con l'aggiunta della scritta [DATA-OK] nel caso quindi in cui non c'è superamento della soglia: questo topic sarà utile al notificatore per capire quando l'evento notificato può considerarsi chiuso. Ad ogni processamento

L'accesso al container è consentito dall'**esterno** tramite la porta **8002**, mentre all'**interno** di docker tramite la porta **8080**; il database denominato **mysql-usersmanager** è connesso al container tramite porta **3306**. I topic per **kafka** sono inviati internamente sulla porta **9092** del container in cui è installato.

API

/consumers

/register	registra o aggiorna un un nuovo utente (consumer)
/all	restituisce tutti gli utenti registrati
/get {username}	restituisce l'utente identificato dal suo username
/delete {username}	cancella l'utente, solo se non à sottoscrizioni registrate

/subscriptions

/register	registra una nuova sottoscrizione per l'utente *username*
/all	restituisce tutte le sottoscrizioni registrate
/unsubscribe {id}	cancella la sottoscrizione identificata dal suo *id*

kafka

Container in cui è installato il broker Apache Kafka. E' raggiungibile **internamente** a docker tramite la porta **9092**.

L'interazione tra Spring e Kafka avviene grazie alla libreria java **kafka-clients-3.6.1.jar**

TOPICS

- ***station-threshold-mintime***
  - stazioni che superano la soglia;
- ***station-threshold-mintime* [DATA-OK]**
  - stazioni che non superano la soglia
- **usersmanager_LOG**
  - alcuni messaggi di log del container *users-manager* per monitorarne il funzionamento regolare o se viene sollevata qualche eccezione;
- notifier_LOG
  - alcuni messaggi di log del container *notifier* per monitorarne il funzionamento regolare.

notifier

Verifica la presenza di nuove notifiche da inviare agli utenti essendo collegato al broker kafka tramite la sottoscrizione ai topics su cui pubblica *users-manager*. A intervalli di 10 secondi verifica la presenza di nuove sottoscrizioni degli utenti tramite una chiamata REST a *users-manager* all'endpoint **/subscriptions/all**, creando di fatto la sottoscrizione al relativo topic. Quando riceve una notifica dal broker recupera l'indirizzo email dell' utente tramite una chiamata REST a *user-manager* all'endpoint **/consumers/get**, eper aggiungere qualche informazione in più relativa alla serie termporale in oggetto fa una chiamata REST a get-time-series per ricavare il valori medio e massimo nell'intervallo di riferimento; tutte queste informazioin vengono inviate tramite email agli utenti interessati.

La notifica all'utente avviene nel momento in cui la constraint viene violata tramite l'invio di una sola email; questo significa che per tutto il tempo in cui il valore della constraint resta sopra soglia (quindi users-manager continua a pubblicare sul topic di riferimento ogni 30 secondi) non saranno inviate altre email (sarebbe una ridondanza inutile). Quando il valore della constraint scende sotto soglia, quindi users-manager pubblicherà sul topic ...[DATA-OK], il notifier invierà la mail di chiusura all'utente.

Utilizza in maniera esclusiva un database MySql denominato **mysql-notifier** (raggiunto tramite la sua porta **3306**) per tenere traccia delle notifiche aperte (quindi non inviare altre notifiche oltre alla prima di apertura), chiudere le notifiche (inviare la mail di chiusura), nonché conservare lo storico delle notifiche gestite.

Non ha un'interfaccia REST, quindi **non è accessibile** ne dall'esterno ne dall'interno di docker.

Seconda parte – monitoraggio del sistema

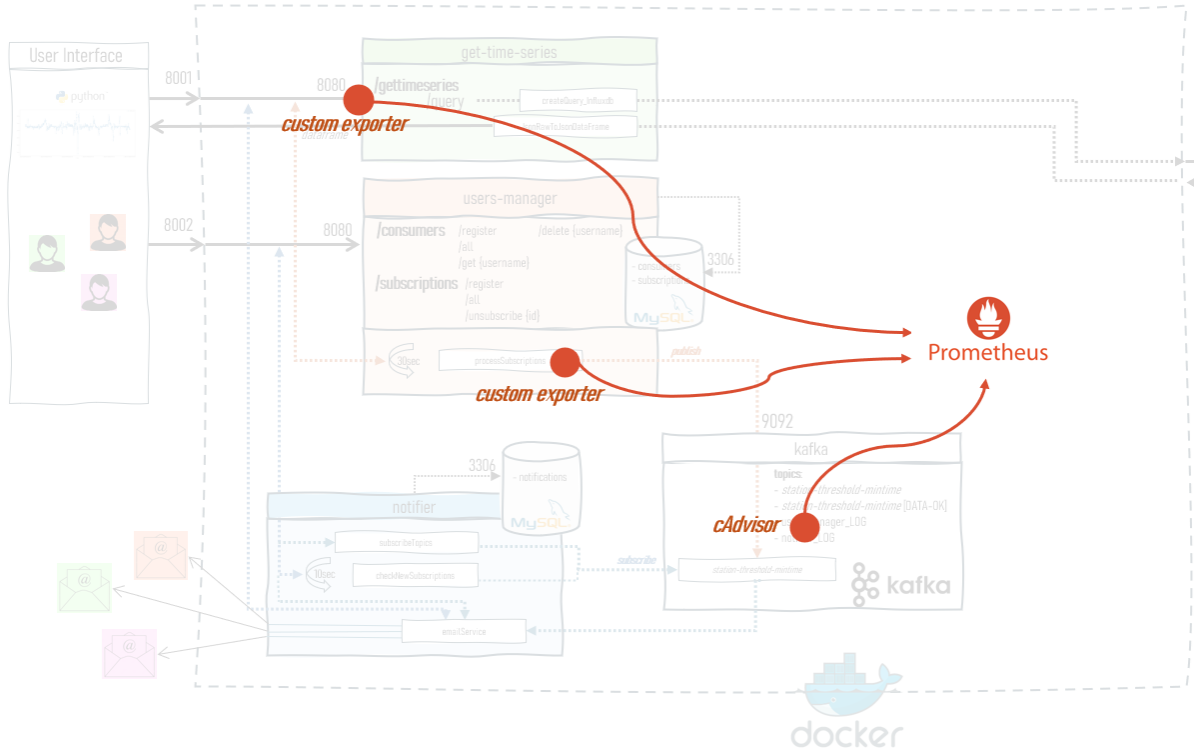
Il monitoraggio delle performance del sistema è stato realizzato con l'applicazione **Prometheus** , contenuta in un apposito container dentro docker, che si mette in ascolto sui punti interessati grazie alla creazione di specifici exporter presenti dentro il sitema:

Le metriche esposte così da Prometheus sono consultabili tramite l'interfaccia web dello stesso raggiungibile all'**esterno** tramite la porta **8003**, alla quale si collegano:

- l'applicazione **Grafana,** installata dentro un ulteriore containder dentro docker,
- un container denominato **sla-manager** il cui scopo è quello di definire uno SLA-set tramite il quale verificare lo stato di qualità del sistema in ogni momento.

Prometheus

Effettua lo scraping delle metriche ogni 5 secondi. Sono stati configurati 3 jobs distinti su cui pubblicare le metriche, una per ciascun miscroservizio interessato. La creazione degli exporter su Spring e stata fatta grazie alla libreria **io.micrometer** [micrometer-registry-prometheus]:



Le metriche studiate per questo progetto, che costituiscono lo **SLA-set**, sono:

get-time-series:

query_time_seconds_max{ job="gettimeseries", method="query" }

users-manager:

SubscriptionsService_seconds_max{job="usersmanager", method="processSubscriptions"}

UsersManagerApplication_seconds_count{job="usersmanager", method="exceptionManager"}

kafka:

process_cpu_seconds_total{job="kafka"}

grafana

Accessibile dall'**esterno** tramite la porta **8005**. E' stata realizzata una dashboard riassuntiva con le 4 metriche dello SLA-set, in modo da avere una consultazione rapida ed efficace sullo stato dell'applicazione:

sla-manager

Questo container ha lo scopo di gestire la qualità del servizio del nostro sistema tramite l'interazione con un interfaccia REST accessibile dall'**esterno** tramite la porta **8004**. Usa in maniera esclusiva il database **mysql-sla**. Il suo funzionamento è descritto tramite i suoi endpoint indicati di seguito.

ENDPOINTS

/slamanager

/discoveryJobs: restituisce l'elenco dei job disponibili, ciascuno riferito al microservizio monitorato.

/discoveryMetrics: indicando il job di interesse, vengono estratte tutte le metriche monitorate

/metrics

/register: registra una nuova metrica per lo SLA-set

/delete {name}&{job}&{method}: cancella la metrica identificata tramite i tre attributi name, job e method

/all: restituisce tutte le metriche dello SLA-set

/get {name}&{job}&{method}: recupera la singola metrica dello SLA-set

/getState: mostra lo stato di tutte le metriche rispetto allo SLA: **OK**(non violata), **KO**(violata)

/violations:mostra il numero di violazioni per tutte le metriche dello SLA-set, nelle ultime 1, 3, 6 ore

/violations/probability/{x}: restituisce la probabilità di violazione nei prossimi x minuti.

Schema finale dell'applicazione "*Etna events notifier*"