UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e Sviluppo di un sistema basato sul framework FIWARE nell'ambito del water re-use management

Design and Development of a system based on FIWARE framework in the water re-use management context

Relatore:
Dott. Adriano Mancini

Tesi di Laurea di:
Piero Castriota

Anno Accademico 2019-2020

Contents

1 Introduzione

Viviamo in un mondo che crea di continuo dati attorno a noi.
In una società in cui ormai il bene più prezioso sono le informazioni, saperle raccogliere, ma soprattutto saperle gestire ed interpretare è la chiave per una società efficiente.

Una diminuzione delle piogge, un aumento della temperatura, un cambiamento della qualità dell'aria sono tutte informazioni che in una *"Smart City"*, una città intelligente, non possono essere ignorate ma vanno e devono essere interpretate per successive previsioni o prevenzioni.
Si sente parlare sempre più spesso dei cambiamenti climatici e delle loro ripercussioni sull'ecosistema. Riprendendo un articolo del quotidiano "La Stampa" [9]: "Il 2019 è stato l'anno più caldo mai registrato in Europa, e già non va bene. Peggio, 11 dei 12 anni più caldi si sono verificati a partire dal 2000, confermando quindi una serie di incrementi della temperatura sempre più serrati. Nell'anno passato le concentrazioni di anidride carbonica e metano sono costantemente aumentate, tanto che se ne rintracciano di simili solo andando indietro di milioni di anni. In tutto ciò, le alte temperature e le ondate di caldo estivo hanno contribuito alla siccità nell'Europa centrale, mentre alla fine dell'anno si sono verificate forti piogge nell'Europa occidentale e meridionale. Questi i numeri più pesanti del rapporto Copernicus C3S. Lasciando l'Europa, invece, a livello globale, gli indicatori climatici mostrano che la temperatura media degli ultimi cinque anni risulta 1,1 gradi al di sopra della temperatura media dell'era preindustriale. In Europa risulta invece superiore di quasi 2 gradi rispetto alla media della seconda metà del XIX secolo."

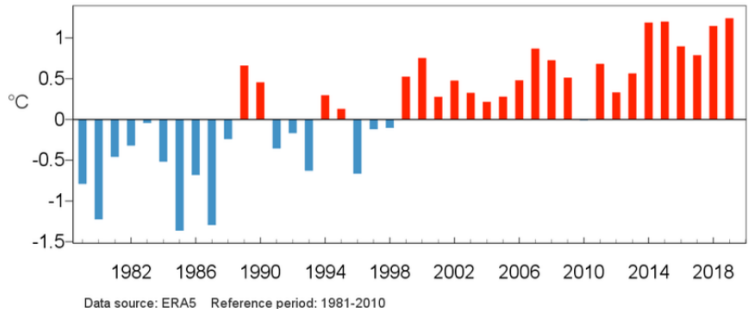

Figure 1: Anomalie annuali delle temperature europee 1979 - 2019

E le temperature non sono l'unico indice di preoccupazione, tutto il ciclo idrologico risulta alterato, comprese alluvioni e siccità. L'Europa non è un continente arido, ma per più della metà della popolazione dell'UE le fonti di approvvigionamento idrico rappresentano un vero e proprio fattore di preoccupazione.

Anche i dati relativi alle precipitazioni non sono rassicuranti. "Non c'è una

tendenza chiara nelle precipitazioni annuali europee", segnalano da Copernicus C3S [20], "il numero dei giorni di pioggia è stato fino a 30 giorni sopra la media nel nord, nell'ovest e nel sud, mentre aree dell'Europa centrale e orientale hanno registrato valori sotto la media". Preoccupa l'umidità del suolo, che mostra "una tendenza discendente, con i valori 2019 che sono i secondi più bassi almeno dal 1979".

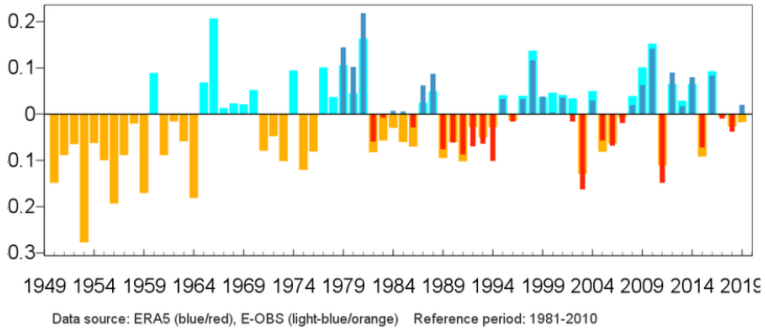

Figure 2: Anomalie annuali delle precipitazioni europee (mm/gg) 1979 - 2019

Il grafico riportato in Fig. 3 illustra la situazione in diversi paesi europei. L'indice di sfruttamento idrico (WEI) [3] indica il rapporto tra la quantità di acqua estratta ogni anno e il totale delle risorse di acqua dolce disponibili a lungo termine e riflette la pressione, cioè lo stress, cui sono sottoposte le riserve idriche Un indice di sfruttamento idrico superiore al 20 % implica una condizione di stress delle riserve, mentre valori oltre il 40 % riflettono uno stress idrico grave e un uso chiaramente insostenibile delle risorse disponibili.

Il consumo idrico annuale delle riserve idriche a lungo termine di Cipro, Bulgaria, Belgio, Spagna, Italia e Malta è attualmente del 20 % o più. A Cipro, dove si sono verificati episodi di grave siccità, il consumo delle risorse rinnovabili ha superato di gran lunga il 40 %.

Risulta dunque abbastanza chiara l'importanza che possiede un corretto utilizzo delle risorse idriche. Soprattutto l'importanza che assume un monitoraggio costante ed efficiente delle risorse, che sia in grado di prevenire e segnalare eventuali peggioramenti e/o problematiche. Il progetto si vuole porre quindi nel mondo dell' *"Internet of Things"* (IoT), un mondo in grado di raccogliere informazioni sull'ambiente che lo circonda, ma soprattutto un mondo in grado di trasmettere e confrontare queste informazioni.

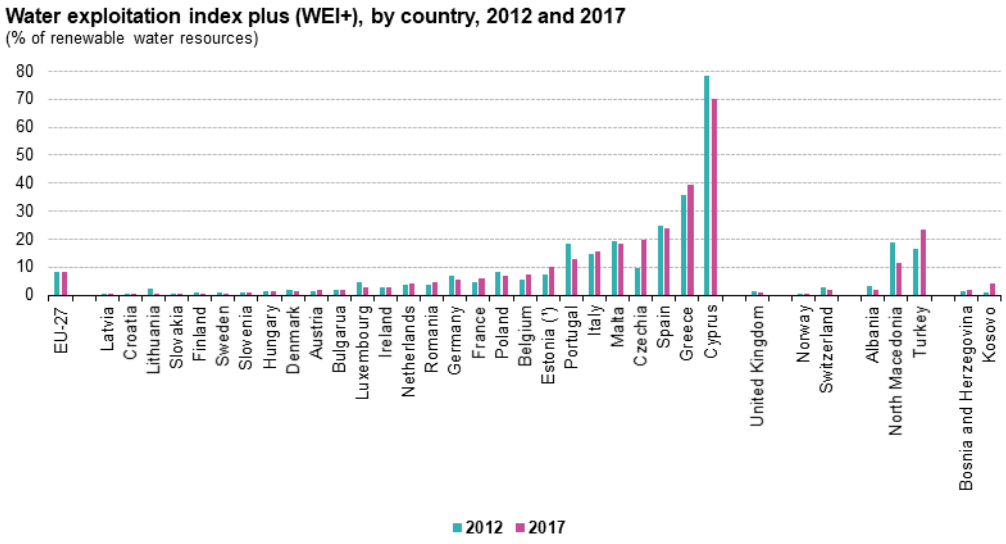

Figure 3: Indice WEI *Fonte: EEA*

1.1 Obiettivi

La digitalizzazione del controllo e monitoraggio delle acque, ma anche il potenziamento di tutte le infrastrutture sono argomenti sempre più di interesse nell'ambito internazionale. La *digital-water.city* (*DWC*)[7] è un'iniziativa europea che si pone principalmente tre obiettivi:

- tutela della salute;
- performance e guadagni;
- coinvolgimento del pubblico.

I fondi (45 miliardi nell'ultimo anno ma che si sperano di raddoppiare nei prossimi anni) vengono utilizzati per la creazione e attuazione di diverse soluzioni che spaziano tutti i nuovi campi della tecnologia, dalla realtà aumentata al *cloud computing*, dall'intelligenza artificiale al *real-time monitoring* che tutt'ora stanno portando grossi benefici alle cinque città in cui il progetto ha preso vita: Milano, Parigi, Berlino, Copenaghen e Sofia. [1]

Seguendo questa linea, si vuole sviluppare un sistema in grado di ricevere in input dati rilevati mediante diverse tipologie di sensori, principalmente adibiti al controllo della qualità delle acque. Si vuole inoltre salvare questi dati secondo opportuni protocolli in dei database e restituire in uscita una serie di *dashboard* grafiche che mostrano l'evoluzione delle rilevazioni nel tempo e la propria distribuzione geografica.

Un ulteriore compito importante che deve svolgere è quello dell' *"Early Warning"*, ovvero quello di una rapida avvertenza nel caso in cui alcuni valori rientrino all'interno di certi intervalli critici.

1.2 Stato dell'arte

La piattaforma FIWARE [21] a cui si è fatto riferimento nel titolo, e a cui riserveremo una trattazione approfondita più avanti, offre molti spunti e progetti funzionanti in ambiti diversi.

Con FIWARE4Water [11] FIWARE si sta immergendo nel mondo del *Water management* per portare nuovi standard che connetteranno la realtà fisica e quella digitale per nuove soluzioni nell'ambito dell'analisi delle acque. Sulla scia del successo di FIWARE in ambiti come le *Smart Cities* e seguendo le linee direttive date dalla commissione europea come *Connecting Europe Facility* (CEF), ci si sta muovendo verso una trasformazione digitale del settore idrico e si stanno dimostrando le capacità e potenzialità di FIWARE, che permettono una cooperazione e scambio d dati tra diversi domini.

Lo sviluppo di *modular smart applications* usando FIWARE e architetture API *open source* per la gestione *real-time* del sistema idrico connetterà gli utenti finali e le imprese fornitrici su una piattaforma *cross-domain* con associati 4 *demo cases* per le *smart water applications* e 3 *demo networks*.

Figure 4: Punti di forza FIWARE4Water

Questo è anche un tentativo da parte della community di FIWARE per ridurre gli effetti dei cambiamenti climatici e le ripercussioni che hanno sulle società, economie e specialmente sulle nostre vite di tutti i giorni. Ad oggi FIWARE può vantare un curriculum ricco di applicazioni e soluzioni concrete con un impatto positivo sul cambiamento climatico che presto crescerà con nuove *Smart Solutions* relative all'acqua.

2 Strumenti e Metodi

2.1 Strumenti utilizzati

Per sviluppare il sistema è stato necessario l'utilizzo di un diverso numero di strumenti. Di seguito se ne elencheranno i principali e si approfondiranno gli aspetti fondamentali di ognuno.

2.1.1 FIWARE

Figure 5: Logo FIWARE

FIWARE [21] è un'iniziativa *open source* nata con lo scopo di definire una serie di standard per la gestione dei *"context data"*, i dati d'ambiente, al fine di facilitare lo sviluppo di iniziative *startup* nell'ambito di *Smart Cities*, *Smart Industry*, *Smart Agrifood* e *Smart Energy*.

In tutte queste *smart solutions* la necessità principale è quella di raccogliere e gestire informazioni, processarle, e restituire output utili per gli utenti permettendo anche loro di modificarne o arricchirne il contenuto.

FIWARE vuole abbattere gli ostacoli dell'innovazione che un'impresa può trovare nella fase di avvio principalmente in due modi. Il primo è quello tecnologico: FIWARE mira a sviluppare delle tecnologie che rendano molto più semplice progettare future applicazioni internet con un'ampia gamma di strumenti che aiutano i programmatori ad abbandonare tutte le complicazioni come maneggiare dati su larga scala, la sincronizzazione con il cloud, la gestione di un gran numero di sensori, la connessione tra loro e all'IoT e non solo.

D'altra parte non è solo un aiuto tecnologico, FIWARE è anche creare un punto di incontro dove possano incontrarsi imprenditori da una parte e sponsor dall'altra, i primi hanno la possibilità di presentare le proprie idee in cerca di un finanziamento da parte di città o aziende che ne hanno la possibilità, i secondi possono invece incontrare questi imprenditori che sperano di coinvolgere a portare nuove idee al loro business.

Questo punto di incontro è ciò che viene chiamato FI-LAB. Un luogo tangibile dove le tecnologie FIWARE sono messe a disposizioni di sviluppatori che possono creare le loro applicazioni, materializzare le proprie idee.

Lo slogan è appunto *"Open APIs for Open Minds"*.

"Open APIs" è rivolto agli sviluppatori, con la promessa di distribuire soluzioni facili ed efficienti per risolvere problemi complessi. "Open Minds" si riferisce invece agli sponsor e investitori, perché per avanzare verso un nuovo livello di innovazione è necessaria la collaborazione ed il confronto tra più realtà differenti.

Un'applicazione che sfrutta le funzionalità di questa piattaforma viene definita *"Powered by FIWARE"*.

Il motore principale dell'architettura risiede nell' *Context Broker*.
Esso sarà il blocco principale della nostra struttura, al quale collegheremo tutte le successive componenti aggiuntive, le quali possono fornire dati da diverse sorgenti (social networks, applicazioni mobile o sensori IoT) oppure possono essere componenti utili per analizzare e visualizzare dati (Grafana[22]).

Se immaginassimo l'intero sistema come una città il *Context Broker* giocherebbe il ruolo dell'ufficio postale: è il punto di passaggio principale delle informazioni e svolge tra l'altro il compito di smistare correttamente i dati in entrata ed inviarli agli opportuni destinatari.
Consideriamo una semplice applicazione in cui l'*Orion Context Broker* è utilizzato per ricevere tramite richieste HTTP e salvare gli stessi in un database del tipo mongoDB [16].

Figure 6: Schema funzionamento Orion Context Broker

Anche questa applicazione è considerata *Powered by FIWARE*, infatti l'impiego del solo *Orion Context Broker* è sufficiente per considerarla tale.

Sfruttando le risorse che offre FIWARE si può avere accesso a molti benefici, come l'utilizzo di strumenti molto efficienti tra i quali citiamo le *subscription*, le *notification* e le *geo-queries*, che rendono molto semplici compiti che in realtà non lo sarebbero affatto.

2.1.2 FIWARE NGSI RESTful API

Un altro aspetto molto rilevante nel mondo FIWARE è l'utilizzo del *FIWARE NGSI RESTful API*, uno standard semplice quanto potente. Grazie al *FIWARE NGSI RESTful API* possiamo definire:

- un *data model* per le *context information*.
- un'interfaccia per i *context data* utile per permettere lo scambio di informazioni tra subscription, query e operazioni di update.

Il *data model* NGSI è definito da una struttura di tipo gerarchico, in cui troviamo gli elementi principali che sono le *entity*, le quali possiedono degli *attributes* che a loro volta sono dotati di *metadata*. Tale struttura viene illustrata nel grafico in Figura 7.

Le *entities* rappresentato un oggetto, che sia di tipo di tipo fisico o logico (un sensore, una persona, una stanza...) e viene riconosciuto unicamente tramite il

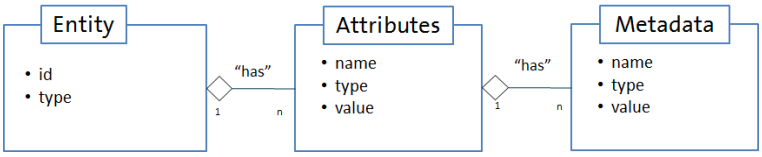

Figure 7: Diagramma entità relazioni

campo *id*. Possiedono inoltre un campo *type* il quale definisce il tipo di oggetto rappresentato dall'entity.
Gli *attributes* sono proprietà delle entities. Come ad esempio la temperatura o il livello di pH. Nel protocollo NGSI gli attributi sono composti dai campi: *name, type, value e metadata*.

- *Name* rappresenta il tipo di proprietà che descrive l'attributo.
- *type* definisce a quale tipo di valore NGSI fa' riferimento mentre
- *value* contiene il valore e opzionalmente i metadati, i quali descrivono le proprietà dell'attributo.

- Infine i *metadata* vengono usati in molte parti, una delle quali è all'interno del campo attributi visto prima. Un po' come gli attributi anche i metadati hanno: *name, type, value*.

Abbiamo dunque visto la struttura del data model NGSI, guardiamone ora la sintassi. Le *entities* sono rappresentate da un *JSON object* di cui in figura riportiamo un esempio:

```
{
  "id": "entityID",
  "type": "entityType",
  "attr_1": <val_1>,
  "attr_2": <val_2>,
  ...
  "attr_N": <val_N>
}
```

Figure 8: Formato JSON di una *entity*

Esistono dei tipi di rappresentazione standard che devono essere supportate nelle implementazioni:

- *keyValues mode*: gli attributi vengono rappresentati solo tramite il loro valore, vengono escluse le informazioni riguardanti il *type* e i metadata (Figura 10).

```
{
  "value": <...>,
  "type": <...>,
  "metadata": <...>
}
```

Figure 9: Formato JSON di un *attribute*

```
{
  "id": "R12345",
  "type": "Room",
  "temperature": 22
}
```

Figure 10: Esempio di *entity* in formato *keyValue*

- *values mode*: questa modalità rappresenta le *entities* come un array di valori degli attributi. Le informazioni riguardanti *id* e *type* sono lasciate fuori. L'ordine degli attributi nell'array è specificato nel parametro URI (es. attrs=branch,colour,engine). Se *attrs* non è specificato l'ordine è arbitrario.

```
[ 'Ford', 'black', 78.3 ]
```

Figure 11: Esempio di *attributes* in formato *values mode*

- *unique mode*: è come la *values mode* ma i valori non vengono mai ripetuti.

Un occhio di riguardo va inoltre riportato per alcuni tipi di attributo speciali:

- *DateTime*: identifica una data, in formato ISO8601[15], possono essere utilizzati per fare operazioni di query in cui siamo interessati solo a dati relativi a misurazioni antecedenti o successive ad una certa data.
- *geo:point*, *geo:line*, *geo:box*, *geo:polygon* e *geo:json*: Definiscono una posizione, hanno una semantica unica in relazione con il tipo di posizione.
- *Builtin Attributes*: *dateCreated*, *dateModified* e *dateExpires*. Da un punto di vista rappresentativo sono esattamente come dei normali attributi solo che non possono essere modificati direttamente dai clients ma solo essere usati dai server per garantire informazioni più accurate. Tutti questi tre attributi hanno un *type: DateTime*.

2.1.3 Componenti aggiuntive per FIWARE

FIWARE è un framework sviluppato e ricco di componenti di piattaforme open source che possono essere assemblate insieme e con ulteriori componenti di terze parti per velocizzare lo sviluppo di nuove *Smart Solutions*. Il componente principale, e l'unico ad essere obbligatoriamente presente, in ogni applicazione *"Powered by FIWARE"* è un *FIWARE Context Broker Generic Enabler* che è la chiave di volta di tutta l'architettura.

Il *FIWARE NGSI* è l'API esportato dal *FIWARE Context Broker* utilizzato per l'integrazione delle varie componenti con la piattaforma *"Powered by FIWARE"* e dall'applicazione stessa per aggiornare o utilizzare dati. Le specifiche del *FIWARE NGSI API* evolvono col tempo, attualmente viene utilizzato il *NGSIv2*. Attorno al FIWARE Context Broker possiamo aggiungere una ricca gamma di componenti che possono svolgere i seguenti compiti:

- Interfacciarsi con l'Internet of Things, Robots e sistemi di terze parti.
- Pubblicazione, gestione e monetizzazione di *context data*.
- Elaborazione, analisi e visualizzazione delle *context information*.

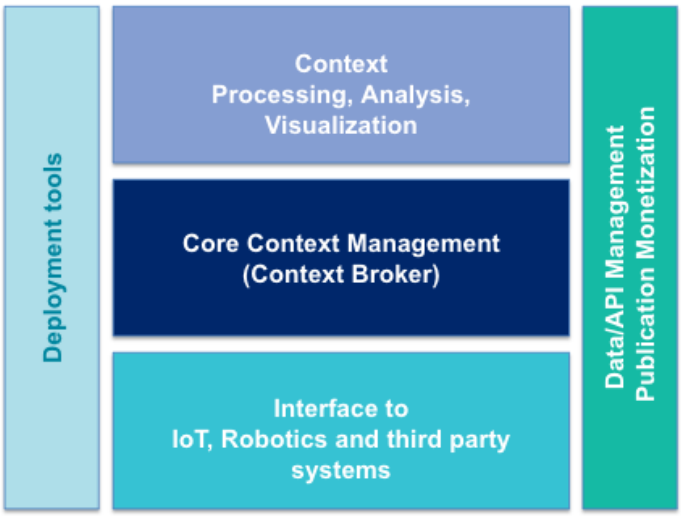

Figure 12: Schema componenti aggiuntive al *Context Broker*

Uno dei vantaggi di utilizzare una soluzione *"Powered by FIWARE"* è che non si è costretti ad utilizzare tutte le componenti, è possibile aggiungere al proprio progetto esclusivamente quelle di cui abbiamo bisogno se e quando vogliamo.

Come *Context Broker* principale useremo l'*Orion Context Broker* il quale fornisce la *FIWARE NGSI v2 API*, semplice e potente, che permette di utilizzare gli *updates*, le *queries* e le *subscription*.

Un ulteriore *generic enabler* che andremo ad utilizzare è "Quantum Leap", il quale supporta un sistema di archiviazione dei dati in alcuni *time series database* quali "CrateDB" o "Timescale".

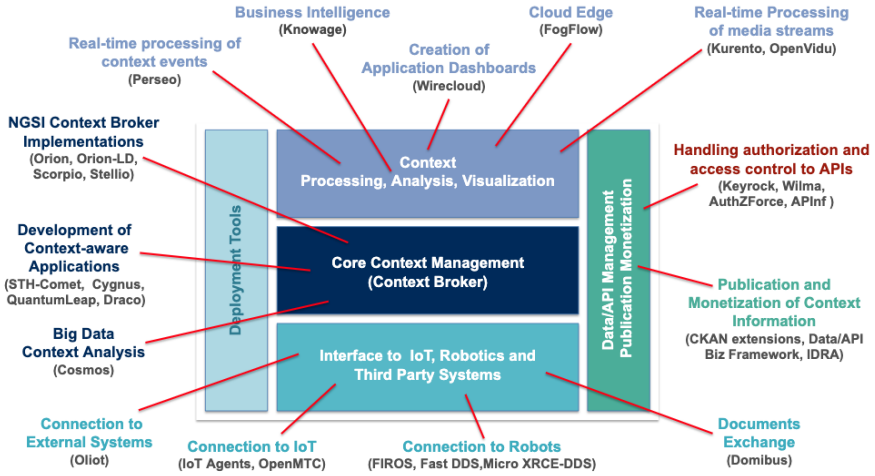

Figure 13: Schema dettagliato componenti aggiuntive al *Context Broker*

2.1.4 Data Model

Come è stato detto precedentemente uno degli scopi principali di FIWARE è quello di impostare una serie di protocolli standard al fine di rendere più veloce ed efficiente la comunicazione tra le varie componenti dell'IoT.

Uno dei passi svolti in questo senso è stata la definizione di alcuni "*data model*", dei veri e propri modelli a cui attenersi quando si vuole creare un'istanza appartenente ad una certa classe i quali garantiscono la portabilità dei dati tra le varie applicazioni.

FIWARE offre una gamma molto ampia di modelli, specifici per moltissimi campi: dagli *alerts* allo *Smart Enviroment*, dai trasporti ai dispositivi vengono definite tutte le classi di entità utili per l'interazione di una *Smart City*.

Per le finalità del progetto è stato utilizzato il *data model* relativo al *WaterQualityObserved* [5] del tipo riportato in Figura 14.

Il *data model* è strutturato come uno schema JSON con diversi campi:

- id : identificatore unico.

```
{
    "id": "waterqualityobserved:Sevilla:D1",
    "type": "WaterQualityObserved",
    "dateObserved": {
        "type": "DateTime",
        "value": "2017-01-31T06:45:00Z"
    },
    "temperature": {
        "value": 24.4
    },
    "NO3": {
        "value": 0.01
    },
    "location": {
        "type": "geo:json",
        "value": {
            "type": "Point",
            "coordinates": [-5.993307, 37.362882]
        }
    },
    "pH": {
        "value": 7.4
    },
    "measurand": {
        "value": ["NO3, 0.01, M1, Concentration of Nitrates"]
    },
    "conductivity": {
        "value": 0.005
    }
}
```


Figure 14: *WaterQualitybserved data model*

- type : tipo di *entity*. Deve essere uguale a "WaterQualityObserved".
- dateObserved : La data e l'ora della misurazione in formato UTC ISO8601. Può essere rappresentato da un istante specifico o da un intervallo ISO8601 [15].
  - Tipo di attributo: *Property*. DateTime o intervallo espresso in formato ISO8601. Obbligatorio.
- location : Luogo dov'è stata effettuata la rilevazione, rappresentato da un *GeoJSON Point*.
  - Tipo di attributo: *GeoProperty*. geo:json.
  - *Normative References: https://tools.ietf.org/html/draft-ietf-geojson-03*.
  - Obbligatorio se "*address*" non è presente.
- temperature : Temperatura.
  - Tipo di attributo: *Property*. *Number*.
  - Unità di misura standard: Gradi Celsius.
  - Opzionale.
- conductivity : conducibilità elettrica.
  - Tipo di attributo: *Property*. *Number*.
  - Unità di misura standard: Siemens per metro (S/m).
  - Opzionale.

- pH : Acidità o basicità di una soluzione acquosa.
  - Tipo di attributo: *Property. Number.*
  - Unità di misura standard: Negativo del logaritmo in base 10 dell'attività degli ioni di idrogeno.
  - Opzionale.

  Questo *data model* è in grado di includere diversi agenti chimici presenti nell'acqua. Le applicazioni devono dichiarare la lista degli agenti chimici la cui concentrazione sta venendo misurata. L'attributo "measurand" va utilizzato a tale scopo.

- measurand : Un array di stringhe che contiene dettagli sugli attributi misurati extra nella presente rilevazione.
  - Tipo di attributo: *Property. List of text.*
  - Valori consentiti: Ogni elemento dell'array deve essere una stringa con iil seguente formato: ¡measurand¿, ¡observedValue¿, ¡unitcode¿, ¡description¿, where:
    - measurand : corrisponde alla formula chimica, ex. CO.
    - observedValue : corrisponde al valore della misurazione in formato numerico.
    - unitCode: Il codice dell'unità di misurazione espressa come codice UN/CEFACT. Per esempio, M1 rappresenta milligrammi per litro.
    - description: breve descrizione dell'attributo misurato.
    - Esempio: "NO3,0.01, M1, Nitrates"
  - Opzionale.

- NO3 : Concentrazione di nitrati.
  - Tipo di attributo: *Property. Number.*
  - Unità di misura standard: Milligrammi per litro.
  - Opzionale.

Oltre agli *attributes* già presenti nel *data model* se ne è voluto inserire uno aggiuntivo al fine di verificare le potenzialità di espansione ed adattamento del sistema. Troveremo dunque tra i campi delle rilevazioni anche il campo "escherichiacoli" con un unico valore "*value*" di tipo numerico.

Un altro *data model* utilizzato è stato quello relativo allo *Smart Energy data model* [4], nello specifico il *ThreePhaseAcMeasurement*. Per motivi di efficienza è stato ridotto ad un numero di attributi inferiore.

2.1.5 Docker e i container

Docker [6] è principalmente una piattaforma per lo sviluppo di software e una sorta di tecnologia di virtualizzazione che rende facile per noi sviluppare e distribuire le app all'interno di "container" ovvero degli ambienti virtuali ordinati che contengono già tutti i requisiti necessari.

Ciò significa che non è importante dove si trovino o su quale macchina stiano girando, non avranno comunque problemi di compatibilità. Il software è quindi *system agnostic*, richiedendo meno lavoro per lo sviluppo e rendendoli più facili da utilizzare.

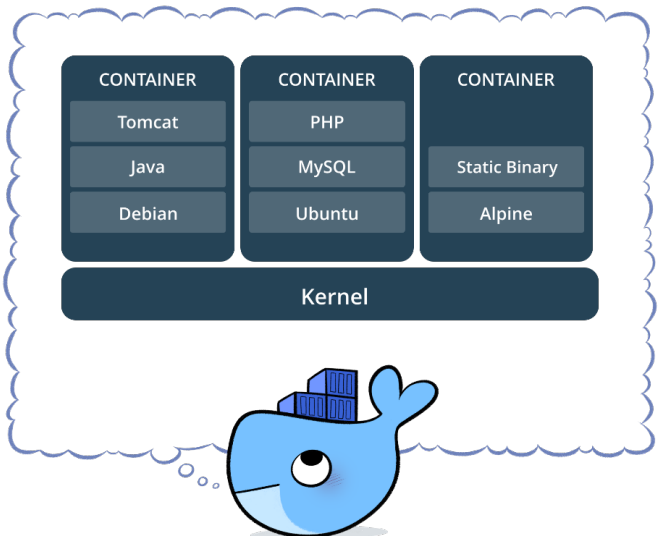

Figure 15: Schema struttura container

Quando un container è in esecuzione sul computer esso si comporta a sua volta come un micro computer, con un compito specifico, ognuno con il proprio sistema operativo, la propria CPU, memoria e risorse network. Anche per questo possono essere facilmente aggiunti, rimossi, fermati e fatti ripartire senza interferire gli uni con gli altri o con la macchina ospite.

Solitamente ogni container ha uno specifico task, come un database MySQL [17] o un'applicazione NodeJS [14], che vengono poi solitamente collegate tra loro.

Docker offre una sorta di virtualizzazione, ma a differenza delle macchine virtuali, le risorse sono condivise direttamente con l'host. Ciò rende possibile mandare in esecuzione diversi *docker containers* quando invece avremmo potuto usare solo poche macchine virtuali.

Una macchina virtuale deve riservarsi una gran quantità di risorse, emulare l'hardware e avviare un intero sistema operativo. A questo punto la macchina virtuale dovrà comunicare con il computer *host* attraverso un'applicazione di

nome "Hypervisor" che farà da traduttore. Docker invece comunica direttamente con il kernel del sistema, tagliando l'intermediario nelle macchine Linux ed anche in Windows10.

Inoltre Docker utilizza anche meno spazio su disco poiché è in grado di riutilizzare efficientemente i file, utilizzandone una sola copia in tutti i container dello stesso tipo che ne hanno bisogno. Le due strutture di virtualizzazione sono messe a confronto nella Fig. 16.

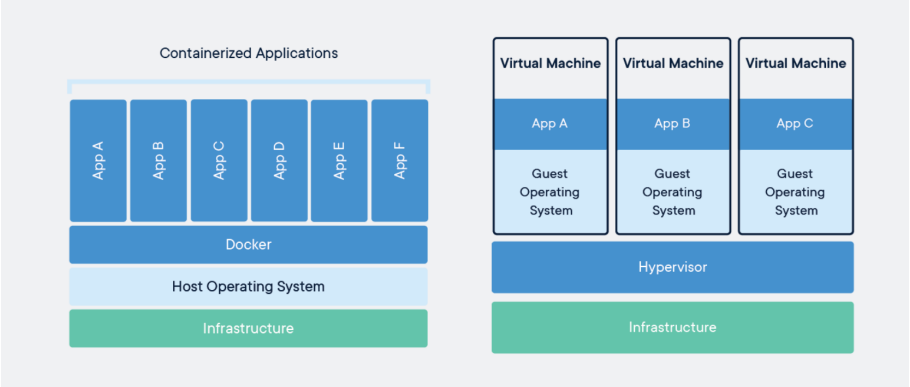

Figure 16: Confronto tra macchina virtuale e container

Per costruire un container si comincia dal Dockerfile, che consente di generare una *Docker image* che verrà poi eseguita come un *Docker container*. Il *Dockerfile* è un semplice file di testo in cui è dichiarato come la *Docker image* verrà costruita. Se non si vuole creare il proprio container ma scaricarne uno già esistente, se ne può scaricare uno da "DockerHub" con il comando *pull*.

Avviare un contenitore è abbastanza facile ma richiede di inserire molti comandi, e potremmo anche aver bisogno di avere molti container, per semplificare le cose ci viene incontro il *Docker Compose* [18].

Compose è uno strumento per definire ed eseguire un'applicazione Docker che richiede molteplici container. Con Compose si utilizza un file YAML [23] per configurare tutti i servizi dell'applicazione. Dopodiché basta un singolo comando per creare e far partire tutti i container che sono stati configurati.

L'utilizzo del Compose può essere riassunto fondamentalmente in 3 passi:

- Definire l'ambiente con un Dockerfile.
- Definire i servizi che si utilizzeranno nell'app all'interno del file *docker-compose.yml* così che possano essere avviati in un ambiente riservato.
- Eseguire *docker-compose up*, e avviare tutti i servizi.

2.1.6 Grafana

Grafana [22] è un software open source per la visualizzazione e l'analisi. Permette di cercare, visualizzare, ricevere avvisi ed esplorare i propri parametri indipendentemente da dove siano salvati. Offre inoltre diversi strumenti per trasformare un *time-series database* in utili e notevoli grafici. Allo scopo de lprogetto, utilizzeremo Grafana per rappresentare grafici che mostrino visivamente le variazioni dei valori rilevati dai sensori nel tempo e per creare delle mappe che mostrino la distribuzione geografica delle rilevazioni effettuate con i relativi valori.

2.1.7 CrateDB

CrateDB [13] è un SQL DBMS progettato per l'utilizzo con *l'Internet of Things* che si basa su una struttura NoSQL. È capace di immagazzinare un gran numero di dati e supporta il *real time querying*. Il database è inoltre progettato per l'esecuzione di compiti complessi come le *geospatial query* ed il *time series data*. Inoltre Crate è anche la miglior scelta per la collaborazione con Grafana, infatti proprio per queste peculiarità rende possibile creare grafici che mostrino l'andamento nel tempo e la distribuzione nello spazio.

2.1.8 Quantum Leap

QuantumLeap [8] è un servizio REST per archiviare, ricercare e recuperare dati spazio-temporali in formato NGSI v2. QuantumLeap converte dati NGSI semi-strutturati in formati tabulati e li mantiene in un *time-series database*, associando ogni registrazione con un indice temporale e, se presente nel dato NGSI, una locazione geospaziale. I *REST clients* potranno poi recuperare le *NGSI entities* filtrandole per intervalli spaziali e/o temporali.

Solitamente QuantumLeap acquisisce dati da fonti IoT, in forma di *NGSI entities*. Come detto sopra, quest'ultime verranno poi tradotti, tabulati e salvati in un database.

Affinché QuantumLeap riceva i dati da Orion, il *client* deve creare una *subscription* in Orion specificando rispetto quali *entities* dovranno essere notificate eventuali modifiche.

2.1.9 Flask

Flask [12] è un *micro-framework* web scritto in Python.

Con il termine *micro-framework* si vuole indicare la peculiare caratteristica di avere un nucleo semplice ma estendibile. Non c'è uno strato di astrazione per la base di dati, validazione dei formulari, o qualsiasi altra componente per fornire funzionalità comuni per le quali esistono già librerie di terze parti. A ogni modo, Flask supporta estensioni che possono aggiungere funzionalità a un'applicazione come se fossero implementate dallo stesso Flask. Ci sono per

esempio estensioni per la validazione dei formulari, la gestione del caricamento dei file, varie tecnologie di autenticazione e altro.Un'applicazione Flask minimale assomiglia a questa:

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
    return "Hello World!"
if __name__ == '__main__':
    app.run()
```

Nel progetto Flask sarà utilizzato per testare le potenzialità della *notification* per progettare un sistema efficiente di *Early Warning*.

2.2 Metodi utilizzati

2.2.1 Subscriptions e Notifications

All'interno della piattaforma FIWARE, un'entità rappresenta lo stato di un oggetto fisico o concettuale che esiste nel mondo reale. Ogni *Smart Solution* ha bisogno di sapere lo stato di questi oggetti in ogni istante di tempo.

Lo stato di ognuno di questi oggetti cambia continuamente. Per esempio, se consideriamo il mondo dello stoccaggio, avremo entità che cambiano ogni volta che apre un nuovo negozio, che si vende un prodotto, che cambiano i prezzi e così via. Per una *Smart Solution* che si basa su dati provenienti da sensori IoT, questo problema è ancora più pressante dato che il sistema reagirà continuamente ai cambiamenti del mondo reale.

Fino ad ora abbiamo parlato di entità statiche, ma l'*Orion Context Broker* offre un meccanismo di notifica asincrono, le applicazioni possono mantenersi aggiornati rispetto a cambiamenti delle *context information* così che possano essere informati quando accade qualcosa. Ciò significa che non abbiamo bisogno di restare in *polling* o di ripetere continuamente delle *query*.

L'utilizzo delle *subscriptions* ridurrà il numero delle richieste da effettuare ed anche il volume di traffico di dati. Questi vantaggi saranno d'effetto soprattutto nella reattività dell'intero sistema.

La *subscription* si effettua tramite una *POST request*, che per il progetto in questione verrà indirizzata sempre alla porta 1026, ovvero dove è reperibile l'*Orion Context Broker*, e nello specifico all'indirizzo:
http://localhost:1026/v2/subscriptions.
Il body della richiesta consiste di due parti, la prima è relativa al soggetto della richiesta (che definisce quale tipo di *entities* si sta considerando e rispetto a quali attributi bisogna essere aggiornati), la seconda è relativa alla *notification* che definisce, una volta che la *subscription* è stata innescata, quali attributi dovranno essere inviati e a quale indirizzo URL.

Si possono anche modificare le condizioni che soddisfano le condizioni della *subscription* per restare aggiornati solo nel caso in cui il valore di interesse rispecchi certe specifiche.

La logica che sta dietro queste condizioni va elaborata nel campo *expression* all'interno di *condition* attraverso l'attributo "q". In Figura 17 ne è riportato un esempio illustrativo.

In questo caso la *subscription* verrà innescata solo quando entrerà un'*entity* che contiene un campo di nome "NO3" il cui valore sarà superiore a 0.3 e contemporaneamente un campo di nome "conductivity" il cui valore sia minore di 0.15.

POST http://localhost:1026/v2/subscriptions/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "description": "Alert for level of escherichiacoli",
  "subject": {
    "entities": [
      {
        "idPattern": ".*"
      }
    ],
    "condition": {
      "attrs": [
        "escherichiacoli"
        ],
        "expression": {
          "q": "NO3>0.3; conductivity < 0.15"
        }
    }
  },
  "notification": {
    "http": {
      "url": "http://quantumleap:8668/v2/notify"
    },
    "attrs": [
      "escherichiacoli"
      ]
  },
  "throttling": 5
}
```

Figure 17: *Subscription* con campo limitato dall'*expression*

3 Progettazione e Sviluppo del Sistema

Ora che è stato definito il contesto in cui si lavora e gli strumenti che si andranno ad utilizzare, è possibile incominciare a vedere come è stata sviluppata la struttura. Inizialmente verrà definito ciò che il sistema effettivamente dovrà andare a fare mentre poi si andrà, passo per passo, a vedere come realizzarlo.

3.1 Analisi dei requisiti

Per portare a termine l'obiettivo di progettare un sistema in grado di prendere in input dei dati e restituire in output grafici ed avvisi avremo innanzitutto bisogno di una sorgente per i dati. Si suppone che il sistema ne riceva da sensori appositi, ma in mancanza, per lo scopo del progetto, verranno forniti dati fittizi per testare il funzionamento, attraverso delle *HTTP requests*. Successivamente, bisognerà andare a salvare questi dati in appositi database (affinché il grafico dia un'idea delle variazioni dei valori nel tempo sarà necessario un *Time Series Database*) ed utilizzarli, con Grafana, per ottenere i risultati desiderati. In aggiunta, si dovrà monitorare costantemente il flusso dei dati in entrata, in attesa di rilevare valori che cadano all'interno di intervalli critici.

3.2 Architettura del sistema

Per costruire la base del progetto si può prendere spunto dal catalogo di tutorial che FIWARE propone nell'apposita sezione, nello specifico, si farà riferimento al tutorial: *304. Querying Time Series Data (Crate-DB).*

Una volta clonata la directory di riferimento ed avviato tutti i servizi necessari tramite il docker compose, verrà creato un sistema funzionante con la seguente struttura:

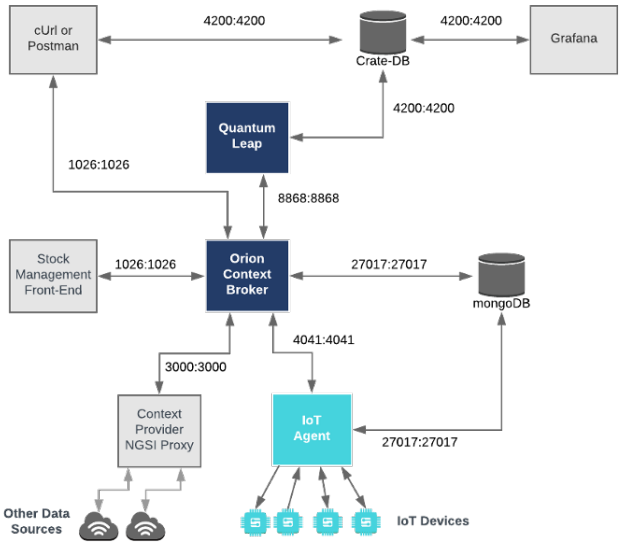

Figure 18: Visualizzazione schema dell'architettura

Come già precedentemente citato le uniche comunicazioni verranno effettuate esclusivamente con l'*Orion Context Broker* attraverso delle *HTTP requests* dal terminale di Postman [19]. Automaticamente le informazioni (che verranno fornite secondo il protocollo NGSI v2) verranno salvate su mongoDB. Contemporaneamente *QuantumLeap* sarà in ascolto sull'*Orion Context Broker* in attesa che una *subscription* rilevi dei dati di interesse. In tal caso, una copia dell'*entity* in questione verrà salvata su CrateDB. A questo punto potremo, tramite un container con Grafana sulla porta 3003, costruire i grafici desiderati. Sulla porta 3000 avremo inoltre il *Context Provider NGSI Proxy* che utilizzeremo successivamente come monitor per verificare l'effettivo invio degli *alert*. Infine, sulla porta 4041 è presente un *IoT Agent* che è stato utilizzato in fase di tutorial ma sul quale non ci soffermeremo.

Possiamo andare a verificare la creazione di tutti questi container dal terminale tramite il comando *docker ps*

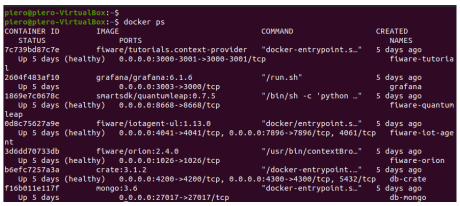

Figure 19: Lista dei container creati visti da terminale

3.3 Data ingestion

Nel file *docker-compose.yml* utilizzato per avviare i servizi, viene effettuata in automatico una chiamata con cui vengono create delle *entities* importandole dal file *import-data* (Figura 20) che sono relative ad una struttura che a noi non serve.

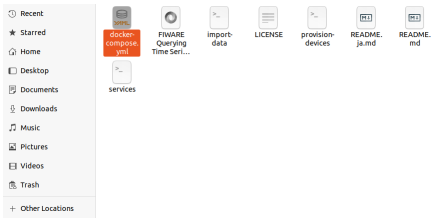

Figure 20: Repository clonata

Il file *import-data* è stato quindi svuotato di tutti gli elementi inutili al progetto e ripopolato con cinque nuovi esempi di *entities* affinché rispecchino il *WaterQualityObserved data model*, ed una relativa all'*Energy data model*.

Il file utilizzato nello sviluppo è riportato nell'Appendice A.

All'avvio dei container si avranno quindi già sei *entities* totali che popolano mongoDB. Ulteriori *entities* potranno essere aggiunte sempre tramite Postman, con una *POST request* all'indirizzo *http://localhost:1026/v2/entities*.

3.4 Popolamento time-series database

A questo punto è possibile utilizzare Postman per effettuare la prima *subscription* all'indirizzo *http://localhost:1026/v2/subscriptions* (Figura 21)

Si è già visto precedentemente il funzionamento delle *subscriptions*, ora ne verrà trattato il caso specifico. I primi due campi definiscono il tipo di dato in entrata, in questo tutti i dati in entrata del tipo "WaterQualityObserved". Nel caso in cui invece si fosse interessati solo ai dati relativi ad un'esclusiva sonda, potremmo modificare il campo *idPattern* per renderlo specifico al proprio indirizzo di provenienza.

Nel campo *condition* si inseriscono gli attributi i cui cambiamenti faranno attivare la *subscription*, per questo progetto è stato deciso di considerare come at-

```
{
  "description": "Notify Quantum Leap to check for WaterQualityObserved every five seconds",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "WaterQualityObserved"
      }
    ],
    "condition": {
      "attrs": [
        "NO3",
        "escherichiacoli",
        "location"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://quantumleap:8668/v2/notify"
    },
    "attrs": [
      "NO3",
      "escherichiacoli",
      "location"
    ],
    "metadata": ["dateCreated", "dateModified"]
  },
  "throttling": 5
}
```


Figure 21: Subscription relativa al WaterQualityObserved data model

tributi campione il campo "NO3", il campo "escherichiacoli" ed il campo "location" (utile quando vorremo fare le *geo-queries*). Nel campo *url* è stato indicato l'indirizzo a cui risponde *QuantumLeap* e sotto *attrs* sono stati definiti i campi da inviare (gli stessi della *condition* in questo caso). Infine sono stati definiti i metadati che verranno generati automaticamente ed impostiamo l'intervallo di tempo con cui si andrà ad aggiornare il controllo (il *throttling* a cinque secondi.

In Figura 22 è riportata una *subscription* analoga per l'*Energy model*.

```
{
  "description": "Notify Quantum Leap to check for WaterQualityObserved every five seconds",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "energy"


      }
    ],
    "condition": {
      "attrs": [
        "totalActivePower"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://quantumleap:8668/v2/notify"
    },
    "attrs": [
      "totalActivePower"
    ],
    "metadata": ["dateCreated", "dateModified"]
  },
  "throttling": 5
}
```


Figure 22: Subscription relativa all'Energy data model

Se il sistema ha funzionato correttamente a questo punto QuantumLeap ha tradotto e replicato le *entities* di partenza su CrateDB assegnando a ciascuna un indice relativo all'istante in cui sono state aggiunte.

Per verificare se le *subscriptions* precedenti sono attive è possibile eseguire attraverso Postman una *GET request* all'indirizzo

http://localhost:1026/v2/subscriptions. Il risultato di tale *request*, che viene riportato in "Appendice B", conferma che sono entrambe attive.

A questo punto si può verificare se i dati sono stati effettivamente riportati sul *time-series database* osservando che il container relativo a CrateDB è aperto sulla porta 4200 (Fig. 13).

Recandosi dunque all'indirizzo *http://localhost:4200*, nella sezione relativa alle tabelle si presenterà la schermata in Fig. 23 dove si può notare che sono state create correttamente entrambe le tabelle: una per l'*energy model* con un solo elemento, ed un'altra relativa al *waterqualityobserved* con tutte e cinque le registrazioni salvate. Inoltre nella parte inferiore sono specificati gli attributi salvati con i quali possiamo effettuare operazioni di *query*. Ne è già stata trattata la maggior parte di essi, di nuovo esiste l'attributo *time index*, il quale rappresenta l'indice di tempo che caratterizza l'ora di registrazione dei dati, peculiare dei database *time-series*

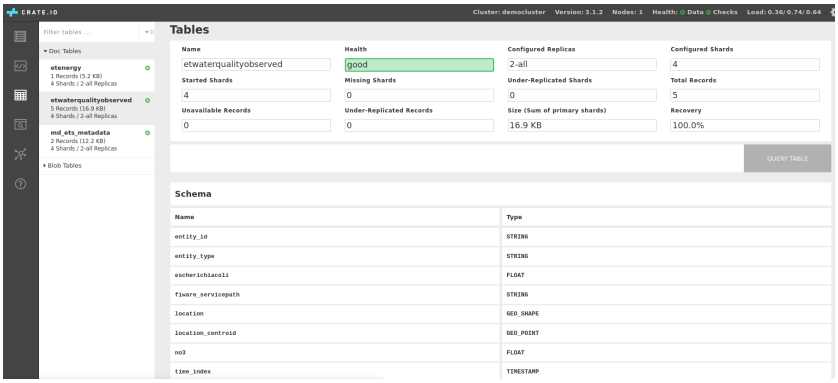

Figure 23: CrateDB

L'interfaccia indica già quante sono le voci registrate nel database, ma per assicurarsi che esse siano quelle di interesse si può effettuare una *query* all'interno di CrateDB, come riportato in Figura 24.

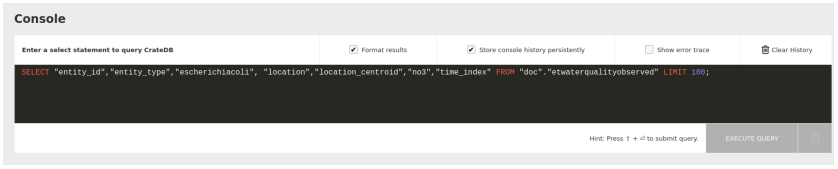

Figure 24: Query CrateDB

La risposta (Figura 25 conferma che i dati coincidono con quelli inseriti precedentemente.

Figure 25: Risultati query CrateDB

3.5 Creazione grafici

Ora che CrateDB è stato riempito con i dati d'interesse, si può procedere al passo successivo, quello di rappresentarli graficamente. Per fare ciò bisogna recarsi all'indirizzo *http://localhost:3003/login* ed effettuiamo l'accesso con le credenziali di default:

$$username : admin$$

$$password : admin$$

Una volta acceduti è necessario creare un nuovo *datasource* di tipo POSTGRE [2] con le seguenti specifiche:

- Name: "CrateDB"
- Host: "crate-db:5432"
- Database: "doc", questo campo dipende da quale sarà la sorgente di dati, in Figura 23 notiamo che la tabella "etwaterqualityobserved" e la tabella "etenergy" sono entrambe delle "doc tables".
- user: "crate"
- SSL mode: "disable"

Una volta impostato il *datasource* possiamo creare un nuovo grafico, premendo sul pulsante "+" e successivamente "Dashboard". Cominciamo con il creare una *query* assegnandole i seguenti valori:

- Queries to: "CrateDB", nome del *datasource* creato in precedenza
- FROM: "etwaterqualityobserved", nome della tabella relativa a CrateDB in cui sono contenuti i dati.
- `" Time column: time_index"`
- `"Metric column: entity_id"`

  . Questo viene utilizzato come nome delle singole rilevazioni.

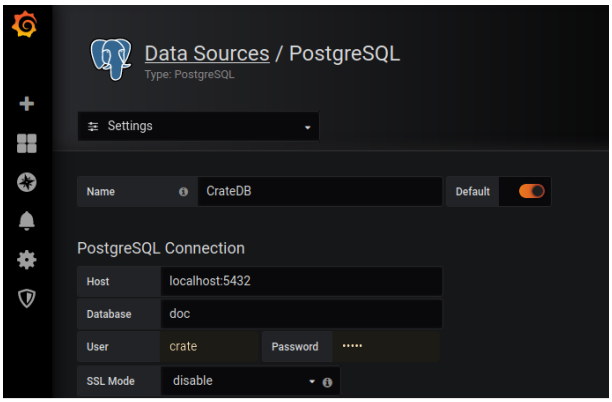

Figure 26: Specifiche datasource

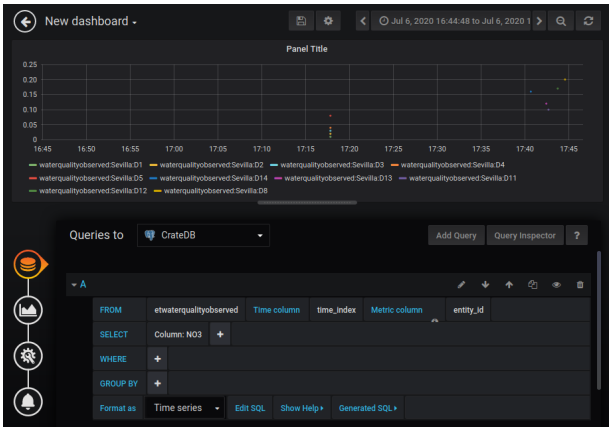

Figure 27: Specifiche datasource

- SELECT Column: "NO3", come valore di interesse per il grafico utilizziamo la concentrazione di NO3.

In Figura 27 osserviamo l'esistenza delle cinque rilevazione inserite in fase di creazione che hanno tutte lo stesso *timestamp*, successivamente sono state aggiunte cinque ulteriori rilevazioni tramite *POST requests* che sono visualizzate nel grafico.

Questo tipo di visualizzazione è utile se si vuole controllare l'andamento temporale di certi valori. Altri tipi di rappresentazione sono possibili tramite l'utilizzo dei *panel*, nello specifico si andranno ad utilizzare un *map panel*, utile per rappresentare la distribuzione geografica delle misurazioni. Il risultato finale è riportato in Figura 3.5, ogni valore è rappresentato geograficamente da un marker di colore rosso che riporta il valore del campo preso in considerazione. Per questo tipo di visualizzazione è stato scelto di considerare come valore di

campionamento l'attributo "escherichiacoli".

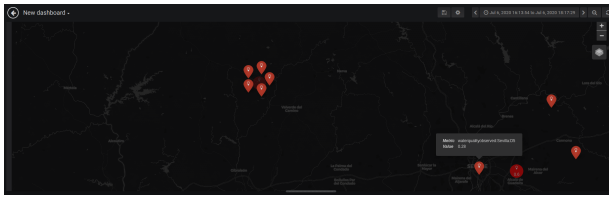

Figure 28: Map panel finale

Per realizzare ciò, dopo aver cominciato la creazione di un nuovo *Map panel* impostare come centralizzazione "Europe" e passare successivamente alla sezione *queries*. La compilazione di questa tabella è analoga a quella fatta precedentemente con la differenza che l'attributo di interesse è ora la concentrazione di *escherichiacoli*.

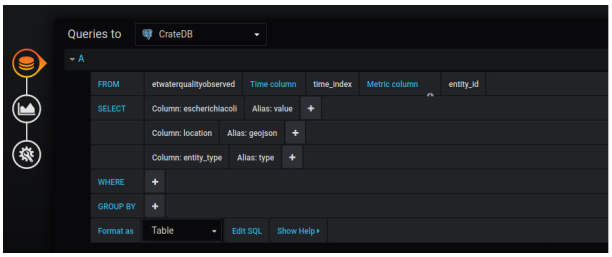

Figure 29: Queries per Map panel

Vanno inoltre aggiunti i valori *"location"* come formato *geojson* e *entity-type* come formato *type*.

A questo punto è stato definito il "cosa" deve essere rappresentato, quali dati, bisogna ora vedere il "come".

Tornando, tramite la barra laterale, alla sezione "*Visualization*" è possibile impostare il metodo di visualizzazione dei dati, la configurazione che è stata impostata è la seguente:

- Icon : lightbulb-o
- Cluster Type : average
- Color Type : fix
- Column for value : value
- Marker Color : red

3.6 Early Warning (monitor)

L'ultimo obiettivo che era stato prefissato era quello di essere notificati non appena si dovesse registrare un valore che rientri all'interno di certi intervalli critici.

Basandosi sempre sui servizi già offerti da *docker-compose.yml* è possibile utilizzare il monitor reperibile all'indirizzo *http://localhost:3000/app/monitor* per controllare eventuali rilevazioni di valori critici.

Il monitor verrà utilizzato come ricettore di notifiche inviate dall'*Orion Context Broker*.

Creiamo dunque una nuova *subscription*, al fine di rimanere sempre aggiornati sui dati in ingresso. Questa volta però essa avrà delle condizioni particolari per essere innescata.

Per il progetto è stato scelto di monitorare l'eventuale ingresso di dati che abbiano un valore del campo "escherichiacoli" superiore a 0.4.

Abbiamo già visto come strutturare una *subscription* e come regolarne la logica, impostiamo quindi come indirizzo URL di notifica quello relativo al monitor utilizzato. Per quanto riguarda la visualizzazione è stato scelto di passare come attributi di interesse solamente il campo "escherichaicoli" per rendere più rapida ed efficiente la visualizzazione sul monitor.

POST http://localhost:1026/v2/subscriptions/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "description": "Alert for level of escherichiacoli",
  "subject": {
    "entities": [
      {
        "idPattern": ".*"
      }
    ],
    "condition": {
      "attrs": [
        "escherichiacoli"
        ],
        "expression": {
          "q": "escherichiacoli>0.4"
        }
    }
  },
  "notification": {
    "http": {
      "url": "http://tutorial:3000/subscription/WARNING-Escherichiacoli-level-is-too-high"
    },
    "attrs": [
      "escherichiacoli"
      ]
  },
  "throttling": 5
}
```

Figure 30: Subscription per Alert Notification

All'invio, la *subscription* andrà a subito a controllare se alcuni dei dati già

presenti nel database soddisfino i requisiti. Con i dati inseriti tramite il file *import-data* lasciato in appendice, e con la suddetta *subscription*, il risultato che si visualizzerà sul monitor sarà il seguente.

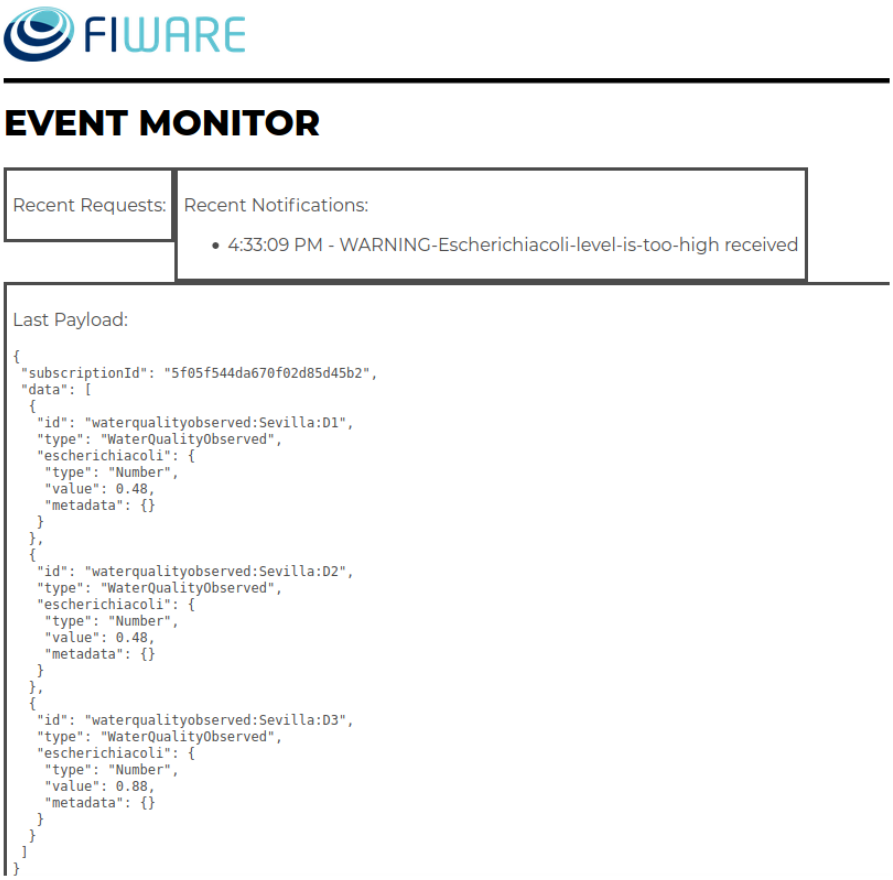

Figure 31: Risultato monitor

Successivamente, quando verranno inserite nuove componenti al database tramite *POST requests*, verrà visualizzato quali di esse soddisfano a loro volta le condizioni. Sul monitor apparirà il messaggio di allarme relativo alla specifica *subscription* (ciò significa che sarà anche possibile utilizzare molteplici *subscriptions* per rimanere aggiornati sotto diversi ambiti) con affianco l'ora della registrazione.

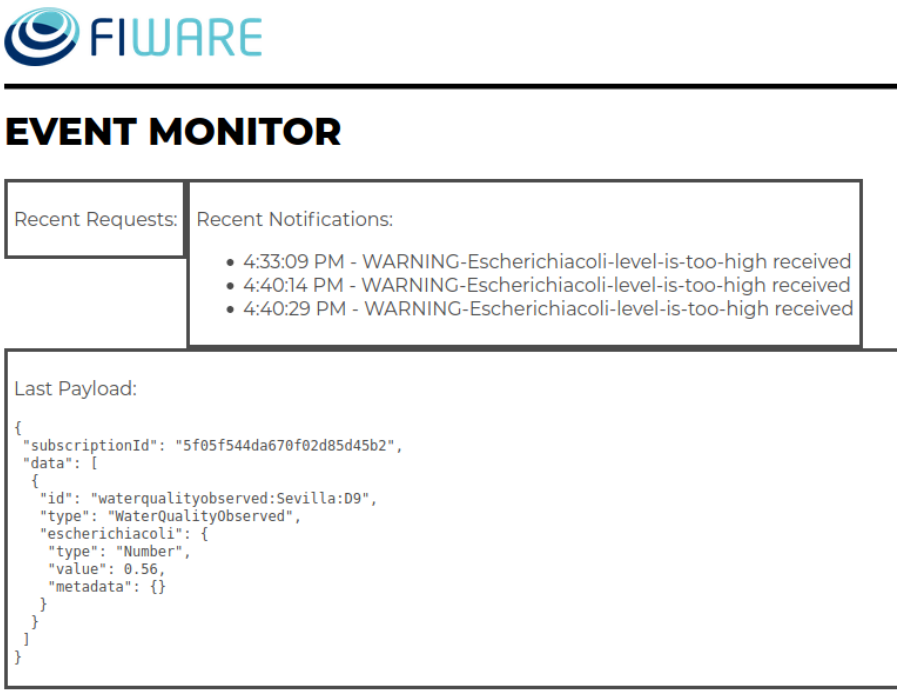

Figure 32: Risultato monitor

3.7 Early Warning (Flask)

Il monitor è uno strumento molto utile per il controllo delle misurazioni. Tuttavia non permette di eseguire operazioni aggiuntive oltre alla visualizzazione dell'*alert*.

Utilizziamo dunque il sopracitato Flask per creare un semplice web server che possa gestire il flusso di informazioni. Il codice che andremo ad utilizzare per il web server è quello riportato in Figura 33. I compiti che gli affidiamo sono abbastanza semplici:

- Salvare i valori che riceve tramite *POST requests*.
- Restituire la totalità delle informazioni salvate alal ricezione di una *GET request*.

Per facilitare le operazioni di comunicazione utilizziamo un *docker container*, sulla porta 5000, in cui andremo a far funzionare il web server. Possiamo rendere automatica la creazione di tale *container*, che chiameremo "notifymanager", andando a modificare il file *docker-compose.yml* aggiungendo nella sezione "Other services" le righe di codice necessarie, come riportato in Figura 34.

```python
from flask import Flask, request, jsonify # Imports the flask library modules
app = Flask(__name__) # Single module that grabs all modules executing from this file

languages = []

@app.route('/', methods=['GET'])
def test():
        return jsonify({'message' : 'It works!'})

@app.route('/monitor', methods=['POST'])
def addOne():
    languages.append(request.json)
    return jsonify({'languages' : languages})

@app.route('/monitor', methods=['GET'])
def hi():

    return jsonify({'languages' : languages})
```

Figure 33: Codice python

```
  # Other services
  grafana:
    image: grafana/grafana:6.1.6
    container_name: grafana
    depends_on:
      - crate-db
    ports:
      - "3003:${TUTORIAL_APP_PORT}"
    environment:
      - GF_INSTALL_PLUGINS=https://github.com/orchestracities/grafana-map-plugin/archive/-
master.zip;grafana-map-plugin,grafana-clock-panel,grafana-worldmap-panel
    volumes:
      - grafana:/var/lib/grafana

  notifymanager:
    image: notifymanager:001
    container_name: notifymanager
    ports:
      - "5000:5000"

networks:
  default:
    ipam:
      config:
        - subnet: 172.18.1.0/24
```

Figure 34: File .yml in cui è stato aggiunto un nuovo container

A questo punto è necessario che il sistema invii al web server le informazioni necessarie quando vengono rilevati dei dati critici. Risolviamo questo problema attraverso una nuova *subscription*, simile a quella relativa al monitor, solo che questa si andrà a comunicare con un differente URL, come illustrato in Figura 35.

Avendo registrato sul database solo i dati relativi al codice in Appendice A, eseguendo una prima *GET request* si otterrà il risultato in Figura 36

Se ora si procede con l'inserimento di una nuova *entity*, con il campo "escherichiacoli" che ha un valore maggiore di 0.4, verrà subito inviata una notifica al web server che procederà con l'aggiunta in lista del nuovo valore come mostrato in Figura 37.

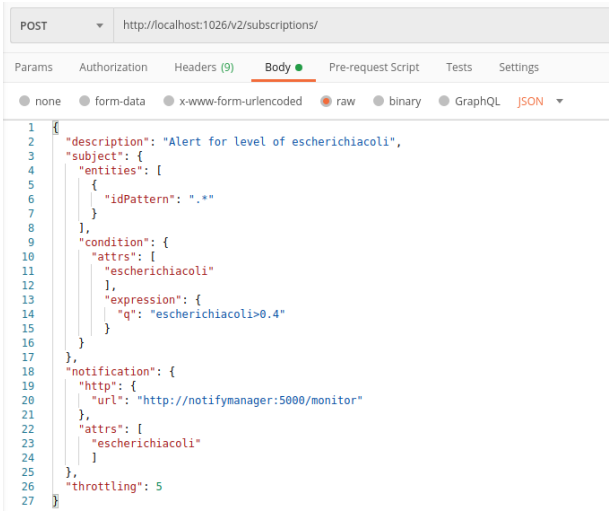

Figure 35: Subscription per web server


```
{
    "data": [
        {
            "escherichiacoli": {
                "metadata": {},
                "type": "Number",
                "value": 0.48
            },
            "id": "waterqualityobserved:Sevilla:D1",
            "type": "WaterQualityObserved"
        },
        {
            "escherichiacoli": {
                "metadata": {},
                "type": "Number",
                "value": 0.48
            },
            "id": "waterqualityobserved:Sevilla:D2",
            "type": "WaterQualityObserved"
        },
        {
            "escherichiacoli": {
                "metadata": {},
                "type": "Number",
                "value": 0.88
            },
            "id": "waterqualityobserved:Sevilla:D3",
            "type": "WaterQualityObserved"
        }
    ],
    "subscriptionId": "5f08908a8d89838a1d945ef0"
}
```


Figure 36: Risultato della prima GET

```
        {
            "escherichiacoli": {
                "metadata": {},
                "type": "Number",
                "value": 0.48
            },
            "id": "waterqualityobserved:Sevilla:D2",
            "type": "WaterQualityObserved"
        },
        {
            "escherichiacoli": {
                "metadata": {},
                "type": "Number",
                "value": 0.88
            },
            "id": "waterqualityobserved:Sevilla:D3",
            "type": "WaterQualityObserved"
        }
    ],
    "subscriptionId": "5f08908a8d89838a1d945ef0"
},
{
    "data": [
        {
            "escherichiacoli": {
                "metadata": {},
                "type": "Number",
                "value": 0.53
            },
            "id": "waterqualityobserved:Sevilla:D11",
            "type": "WaterQualityObserved"
        }
    ],
    "subscriptionId": "5f08908a8d89838a1d945ef0"
}
```


Figure 37: Risultato della GET successiva all'inserimento di nuovi dati

4 Conclusioni

Il presente lavoro di tesi aveva lo scopo di illustrare le diverse fasi progettuali che portano allo sviluppo di un sistema di monitoraggio e analisi di *context data* relativi alla qualità delle acque. A tal fine sono stati utilizzati molti strumenti, e sono stati esplorati molti ambienti diversi con l'obiettivo di unirli e farli comunicare.

4.1 Risultati

Gli obiettivi posti per il progetto sono stati soddisfatti: il sistema basato su FIWARE è in grado di gestire un flusso di dati in ingresso, di elaborarlo e di riprodurre risultati utili allo scopo di prevenzione e previsione.

Uno dei punti di forza del progetto è la sua flessibilità. Le specifiche imposte sono indirizzate al mondo del *water re-use management* ed anche il *data model* scelto seguiva delle linee guida già imposte, ciò non vuol dire però che esso sia un sistema chiuso. Come visto infatti, l'aggiunta di campi aggiuntivi (come è stato fatto per l'attributo "escherichiacoli") non ha portato problemi di elaborazione, ed anche le componenti aggiuntive (come CrateDB e Grafana) sono state in grado di gestirle. Si stima dunque che il sistema possa gestire *data model* anche più complessi, purché si rispettino le convenzioni di comunicazione poste da FIWARE.

I dati provenienti in formato *NGSI v2* sono stati tradotti, tabulati e salvati in database *time-series*, inoltre tali dati sono stati riprodotti in uscita tramite rappresentazioni grafiche e tramite le sofisticate *geo-queries*. Infine, il sistema è anche in grado di inviare un segnale di allerta nel caso in cui i valori che stiamo misurando non soddisfino dei vincoli prestabiliti, sia verso un monitor che avrà il solo scopo di rendere visualizzabile tale allerta, sia verso un piccolo web server che renderà invece possibile effettuare delle operazioni più complesse.

In un contesto di *Smart City*, in cui il monitoraggio della qualità delle acque è fondamentale, un sistema che rispecchi quello presentato in questa tesi apporterebbe un grosso aiuto, rendendone del tutto automatico e assolutamente comodo il monitoraggio ed il controllo. Tramite un semplice monitor si potrebbe essere in grado di sapere dove e quando le rilevazioni presentano dei valori critici, e tramite intuitivi ed efficienti grafici si sarebbe anche in grado di effettuare calcoli statistici utili alla previsione e prevenzione.

4.2 Sviluppi futuri

Per il progetto di tesi i dati sono stati inseriti manualmente, un passo successivo potrebbe essere quello di testare il sistema in funzione di effettivi sensori che inviino periodicamente dati relativi alle loro misurazioni.

In generale, sarebbe interessante vedere come il sistema, inserito all'interno del giusto contesto, possa rispondere a dati effettivi sul monitoraggio delle acque.

Un altro potenziamento che si può apportare al sistema potrebbe essere inoltre quello di utilizzare il web server per effettuare della logica e delle operazioni

con gli *alert* ricevuti.

5 Appendici

5.1 Appendice A

Codice relativo al file *import-data* che è stato modificando per creare delle *entities* utili alle finalità del progetto. Tramite una *POST request* vengono create cinque *entities* relative al *WaterQualityObserved data model* ed un'altra per l'*Energy data model*.

```
#!/bin/bash
#
#  curl commands to reload the data from the previous tutorial
#
#

set -e

printf "Loading context data "

#
# Create five WaterQualityObserved Entities in various locations across Sevilla
#
curl -s -o /dev/null -X POST \
  'http://orion:1026/v2/op/update' \
  -H 'Content-Type: application/json' \
  -g -d '{
  "actionType": "append",
  "entities": [{
     "id": "waterqualityobserved:Sevilla:D1","type": "WaterQualityObserved",
     "dateObserved": {"type": "DateTime","value": "2020-05-17T09:45:00Z"},
     "temperature": {"value": 24.4},
     "NO3": {"value": 0.01},
     "escherichiacoli": {"value": 0.48},
     "location": {"type": "geo:json","value": {"type": "Point","coordinates":    [-5.593307,
     "pH": {"value": 7.4},
     "measurand": {"value": ["NO3, 0.01, M1, Concentration of Nitrates"]},
     "conductivity": {"value": 0.005}
},
{
     "id": "waterqualityobserved:Sevilla:D2","type": "WaterQualityObserved",
     "dateObserved": {"type": "DateTime","value": "2020-05-17T10:45:00Z"},
     "temperature": {"value": 20.0},
     "NO3": {"value": 0.02},
     "escherichiacoli": {"value": 0.48},
     "location": {"type": "geo:json","value": {"type": "Point","coordinates":  [-5.694307, 3
     "pH": {"value": 6.4},
```

```
     "measurand": {"value": ["NO3, 0.01, M1, Concentration of Nitrates"]},
     "conductivity": {"value": 0.006}
},
{
     "id": "waterqualityobserved:Sevilla:D3","type": "WaterQualityObserved",
     "dateObserved": {"type": "DateTime","value": "2020-05-17T11:45:00Z"},
     "temperature": {"value": 25.5},
     "NO3": {"value": 0.03},
     "escherichiacoli": {"value": 0.88},
     "location": {"type": "geo:json","value": {"type": "Point","coordinates":  [-5.795307, 3
     "pH": {"value": 5.4},
     "measurand": {"value": ["NO3, 0.01, M1, Concentration of Nitrates"]},
     "conductivity": {"value": 0.007}
},
{
     "id": "waterqualityobserved:Sevilla:D4","type": "WaterQualityObserved",
     "dateObserved": {"type": "DateTime","value": "2020-05-17T12:45:00Z"},
     "temperature": {"value": 27.7},
     "NO3": {"value": 0.04},
     "escherichiacoli": {"value": 0.38},
     "location": {"type": "geo:json","value": {"type": "Point","coordinates":  [-5.889307, 3
     "pH": {"value": 7.6},
     "measurand": {"value": ["NO3, 0.01, M1, Concentration of Nitrates"]},
     "conductivity": {"value": 0.008}
},
{
     "id": "waterqualityobserved:Sevilla:D5","type": "WaterQualityObserved",
     "dateObserved": {"type": "DateTime","value": "2020-05-17T13:45:00Z"},
     "temperature": {"value": 28.8},
     "NO3": {"value": 0.08},
     "escherichiacoli": {"value": 0.28},
     "location": {"type": "geo:json","value": {"type": "Point","coordinates":  [-5.989307, 3
     "pH": {"value": 8.8},
     "measurand": {"value": ["NO3, 0.01, M1, Concentration of Nitrates"]},
     "conductivity": {"value": 0.008}
},
{
"id": "ThreePhaseAcMeasurement:LV3_Ventilation2","type": "energy",
    "dateEnergyMeteringStarted": {
        "type": "DateTime",
        "value": "2018-07-07T15:05:59.408Z"
    },
    "refDevice": {
        "type": "Relationship",
        "value": ["Device:eQL-EDF3GL-2006201705"]
    },
```

```
"name": {
    "value": "HKAPK0200"
},
"description": {
    "value": "measurement corresponding to the ventilation machine rooms"
},
"totalActiveEnergyImport": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        }
    },
    "value": 150781.96448
},
"totalReactiveEnergyImport": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        }
    },
    "value": 20490.3392
},
"totalActiveEnergyExport": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        }
    },
    "value": 1059.80176
},
"totalReactiveEnergyExport": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        }
    },
    "value": 93275.02176
},
"totalActivePower": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
```

```
            "value": "2019-01-24T22:00:00.173Z"
        },
        "measurementType": {
            "value": "average"
        },
        "measurementInterval": {
            "value": 1
        }
    },
    "value": 31700.269531
},
"activePower": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        },
        "measurementType": {
            "value": "average"
        },
        "measurementInterval": {
            "value": 1
        }
    },
    "type": "StructuredValue",
    "value": {
        "L1": 11996.416016,
        "L2": 9461.501953,
        "L3": 10242.351562
    }
},
"totalReactivePower": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        },
        "measurementType": {
            "value": "average"
        },
        "measurementInterval": {
            "value": 1
        }
    },
    "value": -7830.332031
},
```

```
"reactivePower": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        },
        "measurementType": {
            "value": "average"
        },
        "measurementInterval": {
            "value": 1
        }
    },
    "type": "StructuredValue",
    "value": {
        "L1": -2612.606934,
        "L2": -2209.906006,
        "L3": -3007.81958
    }
},
"totalApparentPower": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        },
        "measurementType": {
            "value": "average"
        },
        "measurementInterval": {
            "value": 1
        }
    },
    "value": 36019.089844
},
"apparentPower": {
    "metadata": {
        "timestamp": {
            "type": "DateTime",
            "value": "2019-01-24T22:00:00.173Z"
        },
        "measurementType": {
            "value": "average"
        },
        "measurementInterval": {
            "value": 1
```

```
            }
        },
        "type": "StructuredValue",
        "value": {
            "L1": 13201.412109,
            "L2": 10755.304688,
            "L3": 11941.094727
        }
    },
    "powerFactor": {
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": "2019-01-24T22:00:00.173Z"
            },
            "measurementType": {
                "value": "average"
            },
            "measurementInterval": {
                "value": 1
            },
            "onlyPositive": {
                "value": true
            }
        },
        "type": "StructuredValue",
        "value": {
            "L1": 0.908817,
            "L2": 0.879906,
            "L3": 0.859293
        }
    }
}

]
}'
echo -e " \033[1;32mdone\033[0m"
```

5.2 Appendice B

Risposta del server quando viene effettuata una *GET request* per verificare lo stato delle *subscriptions* presenti.

Notiamo che ve ne sono presenti due, ognuna con il relativo *id*. La presenza del *success code* assicura l'andamento a buon fine dell'operazione.


```
[
{
    "id": "5f02fe017cd769de466e593a",
    "description": "Notify Quantum Leap to check for WaterQualityObserved every five sec
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": ".*",
                "type": "WaterQualityObserved"
            }
        ],
        "condition": {
            "attrs": [
                "NO3",
                "escherichiacoli",
                "location"
            ]
        }
    },
    "notification": {
        "timesSent": 1,
        "lastNotification": "2020-07-06T10:33:37.00Z",
        "attrs": [
            "NO3",
            "escherichiacoli",
            "location"
        ],
        "onlyChangedAttrs": false,
        "attrsFormat": "normalized",
        "http": {
            "url": "http://quantumleap:8668/v2/notify"
        },
        "metadata": [
            "dateCreated",
            "dateModified"
        ],
        "lastSuccess": "2020-07-06T10:33:40.00Z",
        "lastSuccessCode": 200
    },
```

```
        "throttling": 5
    },
    {
        "id": "5f0302d27cd769de466e593b",
        "description": "Notify Quantum Leap to check for WaterQualityObserved every five sec
        "status": "active",
        "subject": {
            "entities": [
                {
                    "idPattern": ".*",
                    "type": "energy"
                }
            ],
            "condition": {
                "attrs": [
                    "totalActivePower"
                ]
            }
        },
        "notification": {
            "timesSent": 1,
            "lastNotification": "2020-07-06T10:54:10.00Z",
            "attrs": [
                "totalActivePower"
            ],
            "onlyChangedAttrs": false,
            "attrsFormat": "normalized",
            "http": {
                "url": "http://quantumleap:8668/v2/notify"
            },
            "metadata": [
                "dateCreated",
                "dateModified"
            ],
            "lastSuccess": "2020-07-06T10:54:11.00Z",
            "lastSuccessCode": 200
        },
        "throttling": 5
    }
]
```

References

[1] *About digital-water.city*, 2020 (ultimo accesso Luglio, 2020).

[2] *About PostgreSQL*, 2020 (ultimo accesso Luglio, 2020).

[3] *Carenza idrica e siccità nell'Unione europea*, 2020 (ultimo accesso Luglio, 2020).

[4] *Data model Energy*, 2020 (ultimo accesso Luglio, 2020).

[5] *Data model WaterQualityObserved*, 2020 (ultimo accesso Luglio, 2020).

[6] *Developers bring their ideas to life with Docker*, 2020 (ultimo accesso Luglio, 2020).

[7] *DIGITAL-WATER.city - Leading urban water management to its digital future*, 2020 (ultimo accesso Luglio, 2020).

[8] *DOCS - QUANTUMLEAP*, 2020 (ultimo accesso Luglio, 2020).

[9] *Europa, il continente è sempre più caldo*, 2020 (ultimo accesso Luglio, 2020).

[10] *FIWARE-NGSI v2 Specification*, 2020 (ultimo accesso Luglio, 2020).

[11] *Fiware4Water in a nutshell*, 2020 (ultimo accesso Luglio, 2020).

[12] *Flask. Web development one drop at time.*, 2020 (ultimo accesso Luglio, 2020).

[13] *Getting Started With CrateDB*, 2020 (ultimo accesso Luglio, 2020).

[14] *Informazioni su Node.js*, 2020 (ultimo accesso Luglio, 2020).

[15] *ISO 8601 DATE AND TIME FORMATl*, 2020 (ultimo accesso Luglio, 2020).

[16] *mongoDB database manual*, 2020 (ultimo accesso Luglio, 2020).

[17] *MySQL 8.0 Reference Manual*, 2020 (ultimo accesso Luglio, 2020).

[18] *Overview of Docker Compose*, 2020 (ultimo accesso Luglio, 2020).

[19] *The Postman API Platform*, 2020 (ultimo accesso Luglio, 2020).

[20] *Servizio relativo ai cambiamenti climatici di Copernicus*, 2020 (ultimo accesso Luglio, 2020).

[21] *WHAT IS FIWARE?*, 2020 (ultimo accesso Luglio, 2020).

[22] *What is Grafana?*, 2020 (ultimo accesso Luglio, 2020).

[23] *YAML Ain't Markup Language (YAML^{TM}) Version 1.2*, 2020 (ultimo accesso Luglio, 2020).