Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Time-series Database: progettazione e sviluppo di una libreria di supporto al tool Influx 2.0

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Tommaso Segato

II Sessione - primo appello
Anno Accademico 2019/2020

Keywords:

InfluxDB

Time Series

TSDB

Time Series Database

API

Dedica
Alla mia Famiglia

Ringraziamenti:

Ringrazio tutti quelli che mi sono stati vicini in questi anni. Dalla mia famiglia, agli amici, all'amore. Il supporto di tutti è stato fondamentale.

Ringrazio il Prof. Di Felice dell'aiuto datomi nello sviluppo della tesi.

Indice

Introduzione

Il sempre più vivo interesse nei confronti dei big-data e delle serie temporali, spinto anche dai maggiori investimenti in campi come l'IoT, ha aumentato la necessità degli utenti di poter utilizzare strumenti adeguati per la gestione di queste particolari tipologie di dati. Si è, quindi, ritenuto di definire questo progetto di tesi per andare a fornire degli strumenti di semplice utilizzo e generare un supporto ad applicazioni ed utenti in fase di accesso e manipolazione delle serie temporali.

L'obiettivo di questa tesi sarà quello di andare ad implementare una libreria di funzioni JavaScript per l'utilizzo delle API del database InfluxDB v2.0. In particolare verranno implementate le funzioni per quanto concerne gli ambiti dei bucket, dati, task, check, notificantion endpoint e notification rule.

La tesi è pertanto divisa in 5 capitoli:

- Nel Capitolo 1 (Stato dell'Arte) sarà introdotto il tema delle serie temporali associato, fornendo esempi di applicazione. Verrà poi discusso il tema dell'Internet of Things illustrandone lo sviluppo storico e applicazioni. Infine sarà presentato un approfondimento sui time series database con una digressione sulle varie tipologie.

- Nel capitolo 2 (InfluxDB) verrà introdotto il database le cui funzionalità sono state utilizzate durante il lavoro di tesi. La discussione partirà da un'introduzione generale per arrivare a parlare delle caratteristiche delle versioni 1.x del database e di quelle della versione 2.x. In particolare per la versione 2.x ci sarà un approfondimento teorico sul funzionamento delle principali componenti dei database.

- Nel capitolo 3 (Progettazione e Implementazione) verranno introdotti gli obiettivi della tesi e verrà impostata un'analisi delle principali tecnologie utilizzate durante la fase di sviluppo con associata una descrizione delle scelte progettuali. In seguito verranno descritte le funzionalità inserite riportando degli esempi di codice.

- Nel capitolo 4 (Validazione) verrà presentato e descritto il caso d'uso utilizzato per validare le funzioni della libreria. Con descrizione delle funzioni utilizzate.

- Nel Capitolo 5 (Conclusioni e Sviluppi Futuri) verrà riepilogato il lavoro di tesi e verranno discussi alcuni possibili sviluppi futuri.

Capitolo 1

Stato dell'Arte

1.1 Serie Temporale

Una serie temporale è costituita da un set di dati indicizzati in ordine temporale [1]. L'informazione più significativa collegato ad un dato facente parte della serie temporale è il time stamp. I dati delle serie temporali hanno un ordinamento temporale naturale basato sull'evento o sull'intervallo che viene scelto in relazione al tipo di fenomeno che si vuole tracciare e da cui si vogliono ottenere informazioni. I time series data possono essere distinti in due tipi a seconda che le misurazioni vengano campionate ad intervalli regolari, oppure se vengano campionate ad intervalli irregolari, in questo caso sono chiamati eventi.

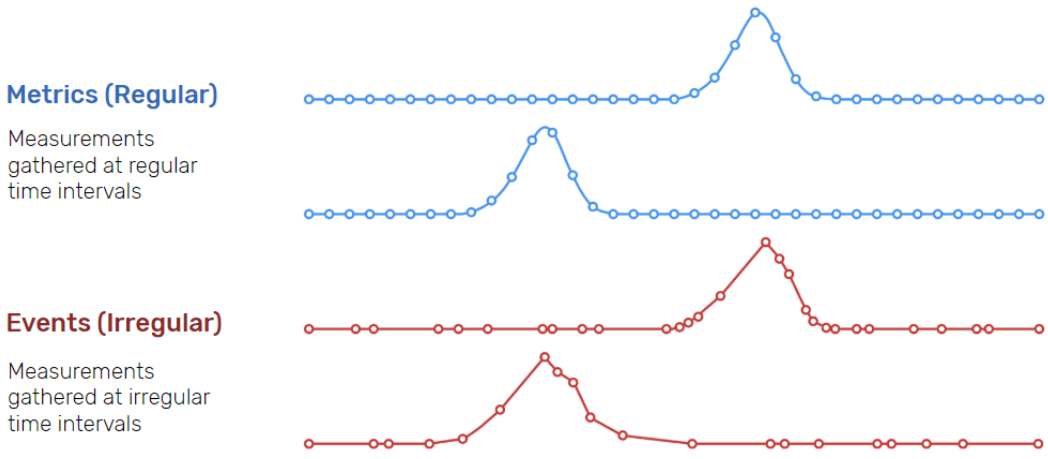

Figura 1.1: Differenti tipologie di serie temporali. [2].

Questi tipi di dataset hanno delle caratteristiche precise per quanto concerne la gestione dei dati e del dataset. I dati in arrivo sono salvati come nuove entry e in ordine temporale, sono considerati, tendenzialmente, immutabili i dataset sono mantenuti nel tempo e la coordinata primaria risulta essere il tempo [1].

Nel momento in cui i time series data sono considerati nella loro totalità è possibile ottenere una rappresentazione delle variazione e del comportamento del processo e del sistema che si sta analizzando. Su questi tipi di dati è possibile definire ed effettuare vari tipi di analisi attraverso metodi per analizzare i dati delle serie temporali al fine di estrarre statistiche significative e altre caratteristiche dei dati. Il time series forecasting indaga sul valore futuro tramite tecniche di predizione tenendo presente che il futuro può essere solo stimato sulla basa di ciò che già è accaduto [3]. L'interrupted time series analysis è un metodo di analisi statistica che prevede il monitoraggio di un periodo a lungo termine prima e dopo un punto di intervento per valutare gli effetti dell'intervento [4].

Le caratteristiche descritte fin qui fanno capire come le serie temporali siano lo strumento da adottare quando l'obiettivo è osservare un certo fenomeno in un dato intervallo. Infatti consentono di operare un'indagine precisa e proficua [5]:

- Rappresentando in modo chiaro le variazioni a cui il sistema analizzato è soggetto nel corso del tempo.
- Consentendo l'individuazione delle caratteristiche generali del sistema analizzato.
- Facilitando la rilevazione di anomalie e la nascita di nuove tendenze.
- Fornendo uno strumento utile e valido per la modellazione predittiva e le previsioni.

1.1.1 Casi d'Uso

Nella complessa situazione attuale le aziende si trovano ad avere necessità di analizzare enormi quantità di dati. Questi dati prendono il nome di big-data e sono caratterizzati da [6]:

- Volume: Dimensione insieme, nell'ordine dei Petabytes.
- Velocità: Elevate velocità di generazione e analisi.
- Varietà: Dati eterogenei sia nel contenuto che nella struttura.
- Valore: Capacità di generare valore a seguito di analisi.

In questo contesto l'analisi del fattore temporale può rivelarsi fondamentale per riuscire a determinare la giusta modalità di interpretazione e di approccio e ottenere così una visione completa. Spesso l'analisi viene affiancata ad una rappresentazione grafica facilitata dall'utilizzo degli assi cartesiani. Solitamente troviamo nell'asse delle ascisse il tempo e nell'asse delle ordinate il valore che si sta considerando.

I dati delle serie temporali sono raccolti, salvati ed analizzati in una molteplicità di settori e contesti. In ognuno dei casi di seguito analizzati è evidente il ruolo

centrale della coordinata temporale. In particolare è importante sottolineare che la visualizzazione degli eventi nell'ordine di arrivo è una caratteristica chiave dei dati di serie temporali, in quanto esiste una capacità di ordinamento temporale naturale [5].

1.2 Internet of Things

Con il termine Internet of Things si intende l'insieme di tutti i dispositivi connessi ad internet (smartphone, computer, automobili) che inviano e/o ricevono dati e che grazie all'ausilio di sensori effettuano delle rilevazioni del contesto. Di base tutte le differenti tipologie di informazioni che è possibile rilevare ed immagazzinare vanno ad ampliare l'ambito di applicazione dell'IoT e, di conseguenza, delle serie storiche [7].

L'estensione agli oggetti dei benefici dell'uso di internet che per anni era rimasto limitati alle persone ha permesso di rendere disponibili i dati necessari a comprendere meglio il mondo reale e a ottenere informazioni rilevamenti per le decisioni aiutando le aziende a sviluppare percorsi per ottenere un'innovazione digitale dei business.

Nel corso del tempo ad aumentare non è solamente il numero di informazioni misurabili ma anche la quantità delle misurazioni stesse. Basti pensare alla variazione delle informazioni prodotte dai telefoni di inizio secolo rispetto a quelli attuali. Non sono aumentate solamente le tipologie di informazioni di cui tenere traccia ma anche il numero delle misurazione dello stesso dato e la quantità dei dispositivi da cui ottenere queste informazioni.

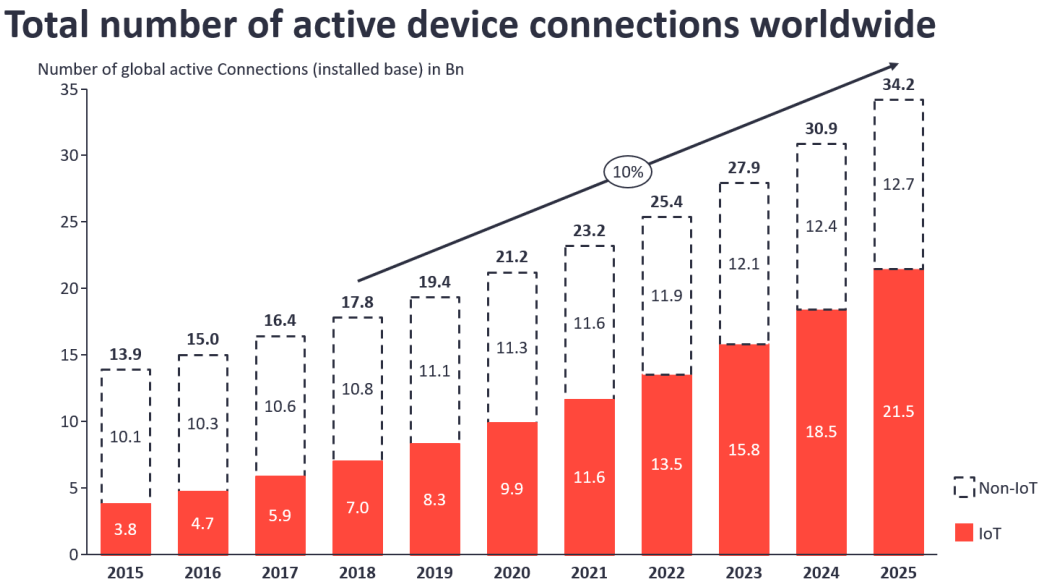

Figura 1.2: Il numero dei dispositivi connessi (17 bilioni) paragonato a Iot dispositivi (7 bilioni). Il numero non include smartphone, tablets, computer. [8]

L'insieme dei dati che viene fornito e raccolto dai dispositivi IoT trovano nelle serie temporale la loro naturale forma di archiviazione e analisi. Infatti questa tipologia di dati una volta raccolta viene inserita e non modificata e, solitamente, visualizzata in ordine cronologico con lo scopo di individuare tendenze, anomalie e caratteristiche del sistema indagato [9]. L'applicazione delle serie temporali nell'ambito del Internet of Things è un esempio piuttosto calzante in quanto consente di capire a pieno la necessità di individuare e sviluppare strutture di raccolta dati sempre più performanti e adeguate per lo storage e analisi di grandi quantità di dati.

1.2.1 Casi d'Uso

La sempre più diffusa applicazione dell'Internet of Things dipende da due fattori chiave, ovvero dagli sviluppi nei campi dell'elettronica e della comunicazione wireless e dai notevoli vantaggi che si riescono ad ottenere attraverso la raccolta e elaborazione di dati tipiche di questa tipologia di tecnologia. I protagonisti diretti delle reti IoT sono i sensori. L'insieme di tutti i device che consentono la raccolta dei dati fra i quali troviamo videocamere, rilevatori di umidità, rilevatori di calore e sensori di movimento [10]. Il numero di device capaci di inserirsi nell'ecosistema aumenta velocemente insieme alle possibilità di applicazione delle informazioni raccolta ed, ad oggi, risulta essere presente in molti settori.

Smart Health

L'IoT è sempre più applicato anche in contesto medico sanitario. Attraverso la digitalizzazione del sistema sanitario è possibile ottenere una raccolta e un'analisi dei dati che contribuisce allo sviluppo della ricerca e al monitoraggio dei pazienti. I dispositivi IoT consentono il monitoraggio e l'archiviazione delle informazioni riguardanti i parametri vitali, come il battito cardiaco e la pressione sanguigna, le radiografie e la risposta ai farmaci [11]. Questo genera un sistema capace di fornire una risposta più rapida ed efficace capace di gestire in modo migliore anche i quadri clinici complessi fornendo trattamenti e piani personalizzati e accurati.

Trasporti

Tra i settori che più si avvantaggiano delle tecnologie in ambito IoT ci sono l'industria dell'automotive e i trasporti. L'IoT riesce a migliorare l'integrazione delle comunicazione, il controllo e anche l'elaborazione dei dati relativi ai vari sistemi di trasporto. Troviamo esempi di applicazioni nel controllo della posizione, della velocità, gestione dei servizi in caso di incidente, controllo e supervisione delle condizioni del veicolo. Risulta evidente come l'applicazione dell'IoT pervade tutti gli aspetti del trasposto, dal veicolo al conducente [12]. Le informazioni ottenute, se combinate con tecniche di apprendimento automatico, danno la possibilità di ottenere dei miglioramenti in ambito costruttivo e di sicurezza.

1.3 Time Series Database

Un database di serie temporali (TSDB) è un database ottimizzato per la gestione di eventi e/o misurazioni basate su data e ora e delle serie temporali. È studiato e implementato per misurare e analizzare i cambiamenti nel tempo essendo che le proprietà delle serie temporali richiedono capacità nella gestione del ciclo di vita dei dati, il riepilogo e l'analisi di una grande quantità di informazioni [13].

1.3.1 Cenni Storici e Sviluppo

Le prime generazioni di TSDB si occupavano della gestione delle serie temporali ottenuti dal mondo della finanza e del trading.

Inizialmente per la gestione delle serie temporali venivano utilizzati i database relazionali. Questi erano caratterizzati da limitazioni tecniche e mancanza di ottimizzazione per ciò che riguarda la manipolazione dei dati basati sul tempo come caratteristica primaria. Per questo motivo, e per il costante aumento della quantità dei dati, si è arrivati allo definizione dei primi TSDB general-purpose caratterizzati da una buona efficienza di gestione ed elaborazione delle serie temporali.

Da questi si è passati alla situazione attuale dove sono presenti TSDB verticali capaci di alte prestazioni a basso costo. Questi database per la gestione di serie temporali hanno capacità di elaborazione dei dati più avanzate, algoritmi di compressione più efficienti e motori di archiviazione più adatti alle caratteristiche delle serie storiche [14].

Oggi, con la diffusione delle mondo IoT e la strumentalizzazione di svariati componenti per l'ottenimento di informazioni si ha la necessità di operare su sistemi più ottimizzati per la gestione di grandi quantità di dati eterogenei fra loro. Questo risulta evidente anche dal tasso di crescita dei Time Series Database rispetto alle altre tipologie.

1.3.2 Tipologie di Time Series Database

L'individuazione dei dati da campionare era il punto di partenza standard per la gestione dei Time Series Database. Questi venivano raccolti e studiati per periodi più o meno lunghi di tempo al fine di produrre la documentazione necessaria, andando poi ad eliminare molti dei dati raccolti [1]. La necessità di eliminare i dati raccolti era dovuta al fatto che la quantità di questi ultimi sopraffacesse le capacità dei database tradizionali in poco tempo e andasse ad aumentare i costi per il mantenimento e la gestione.

Ad oggi il metodo di archiviazione è cambiato. In particolare grazie ai nuovi sistemi NoSQL e ai nuovi strumenti è possibile mantenere, memorizzati e analizzare dati raccolti in anni di ricerca senza andare in contro a costi troppo elevati per la gestione dei sistema. In questo modo tutti quei dati che nei periodi passati venivano eliminati vengono mantenuti in memoria e utilizzati nelle analisi dando la

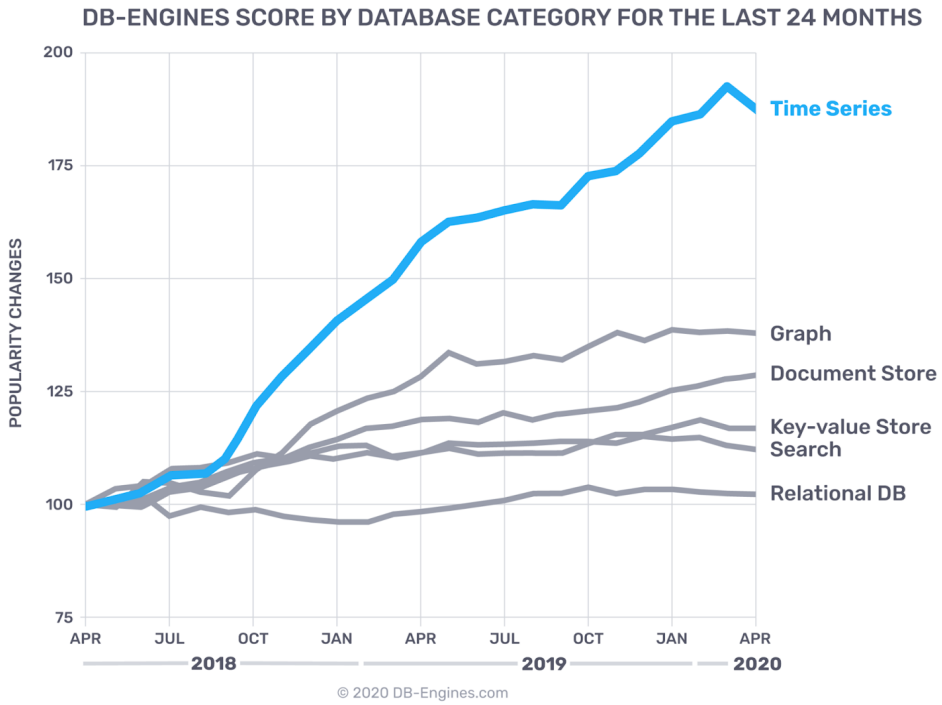

Figura 1.3: Trend crescita database dal 2018 al 2020. [13]

possibilità di effettuare delle ricerche più precise in termine di anomalie, trend e nuove tendenze.

Il risultato di questa innovazione è l'ampliamento delle situazione in cui i dati vengono collezionati come serie temporali e gestiti tramite i Time Series Database. Ci sono tre principali tipologie di database: NoSQL, RDBM e i Times Series Management System.

Database Management System Relazionali

L'utilizzo dei database relazionali nella gestione delle serie temporali è fonte di discussioni da lungo tempo. Prima degli anni duemila sono stati svolti alcuni tentativi con l'obiettivo di ottenere un ottimizzatore di query che permettesse l'utilizzo di sequenze nel database al posto delle tuple [15]. La non efficacia ed i problemi di performance dei RDBMS nella manipolazione delle serie temporali dipendono dall'elevato numero di funzionalità di cui sono forniti. Questo riguarda in particolar modo l'inserimento di dati. Per aumentare la velocità è necessario eseguire l'ingestione tramite vettori memorizzati come valori delta e raggruppati da un cluster primary key, questo produce come effetto l'impossibilità di utilizzare query SQL standard e la necessità di andare a definire come i dati saranno condivisi, indicizzati e raggruppati [16]. Da questo si evince come sia possibile utilizzare i database relazionali per la gestione delle serie temporali se pure con alcune limitazioni.

```
CREATE TABLE IF NOT EXISTS METEO(
    Timestamp TIMESTAMP,
    TemperaturaMax INT,
    TemperaturaMin INT,
    Pressione INT,
    PRIMARY KEY(Timestamp)
    )ENGINE = INNODB;
```

L'esempio di codice riportato propone una possibile tabella per la gestione dei time series data su MySQL. Il valore primario risulta essere il timestamp necessario per indicizzare in ordine temporale i dati memorizzati sul DB.

NoSQL Management System

I sistemi di gestione NoSQL e RDBMS hanno come obiettivi primari il salvataggio, recupero dei dati e coordinazione dei cambiamenti tuttavia tra i due vi è una sostanziale differenza. Un sistema NoSQL va a perdere alcune funzionalità dei RDBMS per aumentare la scalabilità e le performance generali, in particolare:

- Eliminazione e/o riduzione delle funzionalità supportate del linguaggio SQL.
- Eliminazione ottimizzazione delle query SQL.

Questo comporta dei benefici che includono un aumento della semplicità generale, la possibilità di manipolare dati semi strutturati e denormalizzati e l'aumento della scalabilità del sistema. Dall'altra parte viene meno parte dell'astrazione fornita dal ottimizzatore di query quindi il linguaggio SQL e le transazioni e nasce la necessità per lo sviluppatore di preventivare l'ottimizzazione. Andare ad eliminare l'ottimizzazione è vantaggioso in alcuni ambiti specifici poiché consente allo sviluppatore di ottenere performance più prevedibili [1].

```
document = {nome: "rilevazione",
            orario: new Date("2020-09-29T16:00:00Z"),
            input:[...]
            };

db.meteo.insert(document);
```

L'esempio riportato serve per chiarire come, nonostante nei sistemi NoSQL non sia richiesta una struttura fissa del database, sia necessario fornire sempre il timestamp da memorizzare per riuscire a gestire i time series data.

Time Series Management System

Un TSMS è uno strumento utile per salvare molteplici Time Series Data, in modo tale che le query possano essere ottimizzate per selezionare uno specifico intervallo

di tempo. Le applicazioni in cui vengono considerati gli intervalli di tempo sono tra le migliori candidate per implementare ed usare un TSMS. L'implementazione dei Time Series Database si basa sul fatto che i Time Series Data sono raramente modificato, e spesso vengono recuperati leggendo una serie continua di campionamenti che vengono, solitamente, gestiti senza modifiche. Tra le caratteristiche più importanti vi è l'elevato ingestion rate (processo di acquisizione e importazione dei dati per uso immediato o archiviazione), inoltre vi è la possibilità per il sistema di supportare l'analisi ammortizzata ed hanno una gestione del sistema che consentono una di ottenere una grande scalabilità. A queste caratteristiche si aggiunge la possibilità di poter effettuare downsampling per adattare il risultato su un periodo di tempo più lungo, questo è uno degli elementi che rendono i TSMS un ottimo strumento per la gestione delle serie temporali [1].

La famiglia dei TSMS può essere suddivisa, non considerando quelli proprietari di cui non è possibile conoscere la struttura in tre tipologie [15] [16]:

- **Internal Data Stores**: L'insieme dei TSMS implementati per avere una profonda integrazione tra componenti di archiviazione e di elaborazione. La memorizzazione dei dati è autonoma nonostante vi sia l'utilizzo di DBMS per salvare metadati e informazioni addizionali. Sono sistemi centralizzati e raramente distribuiti.

- **External Data Stores**: L'insieme dei TSMS implementati mediante il collegamento ad un DBMS già esistente. Il vantaggio principale è la possibilità di poter usare le configurazione del sistema a cui è appoggiato mentre lo svantaggio principale riguarda la limitazione alle modalità di archiviazione che devono rispettare quelle del DBMS di riferimento. Sono sistemi distribuiti e vengono impiegati quando le serie temporali sono persistenti nel sistema.

- **RDBMS Extensions**: Derivano dai RDBMS esistenti i quali sono estesi attraverso l'aggiunta di funzionalità per rendere l'archiviazione e l'analisi delle serie temporali più efficiente. Si ha il vantaggio di avere l'analisi e l'archiviazione in riferimento ad un unico sistema tuttavia ciò è limitante per quanto riguarda la possibilità di modifica alle funzionalità. I sistemi di questo tipo tendono a non essere distribuiti perché mancano di buona scalabilità.

1.3.3 Criteri di Valutazione

Al fine di individuare il TSMS più adatto alle necessità del problema che si deve affrontare è necessario scegliere sulla base delle caratteristiche che sono ritenute più importanti in relazione al progetto [15] [16]:

- **Caso d'Uso**: È necessario scegliere il sistema più in linea con le esigenze dell'Indagine che si vuole effettuare.

- **Continuos Calculation**: Possibilità di eseguire in modo continuativo funzioni basate sui dati ricevuti memorizzando il risultato.

- **Scalabilità**: Possibilità di miglioramento delle performance in seguito all'aumento del numero dei nodi. Definisce la grandezza del più grande dataset ed il numero di nodi che un TSDB può gestire.
- **Long-Term Storage**: Necessario per gestire grandi quantità di dati sul lungo periodo. È possibile utilizzare dei dati longevi oppure si possono utilizzare le serie temporali matriciali che hanno due o più timestamp per valore.
- **API ed Interfacce**: Definizione metodi interazione con il sistema (linguaggi di query, client libraries, plugin, interfacce web etc).
- **Maturità**: basata su tre livelli
  - *Proof-of-concept implementations*: Solo funzionalità necessarie per valutazione nuove tecniche.
  - *Demonstration system*: Funzionalità necessarie all'utente per interagire con il sistema. Con possibili applicazioni in casi d'uso reali.
  - *Mature system*: Strutture robuste applicabili a casi d'uso reali. Sostenute da comunità open source o da supporto commerciale.
- **Load Balancing**: Possibilità di distribuire le query ai nodi ottenendo un flusso di lavoro identico nei vari nodi del TSMS.
- **Storage Layout**: Modalità memorizzazione delle serie temporali nel sistema.
- **Granularità**: La minore distanza esistente tra due serie temporali memorizzate.
- **Operazioni Consentite**: Lista delle funzionalità che è possibile utilizzare
  - *Select*: Selezione dei data points.
  - *Insert*: Inserimento dei data points.
  - *Delete*: Eliminazione dei data points.
  - *Append*: Aggiunta dei data points alla fine di una serie temporale.
  - *Update*: Aggiornamento dei data points che fanno già parte della serie temporale.
  - *Aggregate*: Calcolo degli aggregati partendo da una serie temporale.
  - *Join*: Unione di più serie temporali sul timestamp o sul valore.
  - *Over*: Esecuzione di calcoli su una serie temporale attraverso funzioni del sistemo o definite dall'utente.
  - *Data Analysis*: Analisi dei dati memorizzati.

Summary of the Surveyed Systems in Terms of Functionality

	Select	Insert	Append	Update	Delete	Aggregates	Join	Over	Analytics
Internal Stores									
tsdb [30]	●		●	●					
FAQ [31]	●					●			Jaccard, Top-K, AQP, etc
WearDrive [32], [33]	●		●		●			●	
RINSE [34], [35], [36]	●	●	●	●	●				NN-Search, AQP
Unnamed [37]						●			AQP
Plato [38]	●	●	●	●	●	●	●		Interpolation, AQP
Chronos [39], [40]	●	●	●	●	●	●			
Pytsms [41], [42]	●		●			●			
PhilDB [43]	●	●	●	●					Pandas
External Stores									
TSDS [44]	●					●			AQP
SciDB [45], [46], [47]	●	●	●	●	●	●	●	●	Linear Algebra, AQP
Respawn [48], [49]	●		●			●			
SensorGrid [50]	●		●			●		●	
Unnamed [51], [52], [53]	●		●						AQP
Tristan [54], [55]	●		●			●			AQP
Druid [56]	●		●			●			AQP
Unnamed [57]	●	●	●				●		AQP
Unnamed [58]	●		●			●		●	
Bolt [59]	●		●		●				AQP
Storacle [60], [61]	●		●			●			
Gorilla [62]	●		●						
Unnamed [63]	●		●			●			Frequency est, AQP, etc
servIoTicy [64], [65]	●	●	●	●	●			●	
BTrDB [66], [67]	●	●	●	●	●	●			
RDBMS Extensions									
TimeTravel [68], [69]	●	●	●	●	●	●	●	●	Forecasting, AQP
F^2DB [70], [71]	●	●	●	●	●	●	●	●	Forecasting, AQP
Unamed [72], [73], [74]	●	●	●	●	●	●	●	●	Interpolation, AQP

Figura 1.4: Operazioni di alcuni TSDB. [15]

Capitolo 2

InfluxDB

InfluxDB è un database di serie temporali open source che nasce nel 2013 e che, ad oggi, risulta essere il time series database più diffuso e utilizzato a livello globale. Alcuni dei punti di forza di InfluxDB derivano dall'ottimizzazione del processo di archiviazione e di recupero di grandi quantità di dati e nella sua capacità di parallelizzare e gestire la concorrenza di molteplici sorgenti in scrittura in contemporanea. Per queste, ed altre, sue caratteristiche risulta essere ampiamente utilizzato in campi come l'Internet of Things e nell'analisi in tempo reale.

2.1 InfluxDB v1.x

InfluxDB 1.x è il database facente parte del pacchetto TICK (Telegraf, InfluxDB, Chronograf, Kapacitor). Consente la compressione, l'esecuzione e l'acquisizione di query in tempo reale, è scritto interamente in Go senza dipendenze esterne. Fornisce funzionalità di scrittura e query tramite riga di comando, API HTTP e un set di librerie. Funziona tramite l'utilizzo di InfluxQL, un linguaggio di query SQL-like usato per interagire con i dati, che fornisce funzionalità specifiche per l'archiviazione e l'analisi dei dati delle serie temporali. Data la possibilità di InfluxDB di poter gestire grandi quantità di dati il sistema è stato dotato di politiche di downsampling e conservazione dei dati. InfluxDB compatta automaticamente i dati per ridurre lo spazio di archiviazione utilizzato e può eseguire il downsampling dei dati, ovvero riesce a conservare dati grezzi ad alta precisione per un periodo di tempo limitato e può conservare dati a bassa precisione per un periodo di tempo maggiore e, potenzialmente, senza limiti temporali [17]. A supporto di questi meccanismi ci sono le funzionalità di query continue (continuous query) e politiche di conservazione (retention policies) rispettivamente query eseguite automaticamente e periodicamente su un database utili per l'ottimizzazione dei dati e componente di InfluxDB necessaria a definire l'arco temporale durante il quale tenere memorizzati i dati del database [18].

Le caratteristiche principali posso essere riassunte in [19]:

- Archivio dati personalizzato ad alte prestazioni scritto specificamente per i dati di serie temporali.
- Scritto interamente in Go. Si compila in un unico binario senza dipendenze esterne.
- API HTTP di scrittura e query semplici e ad alte prestazioni.
- Linguaggio di query SQL-like progettato per eseguire facilmente query sui dati.
- I tag consentono di indicizzare le serie per query veloci ed efficienti.
- Le politiche di conservazione fanno scadere automaticamente i dati obsoleti.
- Le query continue calcolano automaticamente i dati aggregati per rendere più efficienti le query frequenti.

2.1.1 Pacchetto TICK

Il pacchetto TICK è un insieme di progetti open source combinati per fornire gli strumenti necessari alla gestione e all'analisi di generose quantità di informazioni caratterizzate dalla presenza di un timestamp [17].

Telegraf

Telegraf svolge il compito di data collector. Installato nel macchinario o in generale su un server o dispositivo remoto, si occupa della raccolta dei dati attraverso l'utilizzo di input plugins e dunque di inviare le time series al database a cui è connesso dopo averli tradotti nel formato line protocol di InfluxDB [20].

Chronograf

Chronograf è l'interfaccia utente tramite cui gestire le componenti del pacchetto TICK. Semplifica la gestione e il monitoraggio includendo librerie per creare velocemente dashboard con visualizzazione in tempo reale dei dati e per definire regole e modalità di avviso automatiche [21].

Kapacitor

Kapacitor è un framework di elaborazione dati che si occupa di elaborare i dati memorizzati nel time series database. Include la possibilità di rilevare anomalie, eseguire avvisi sulla base di soglie o comportamenti attesi operando sui dati in tempo reale [22].

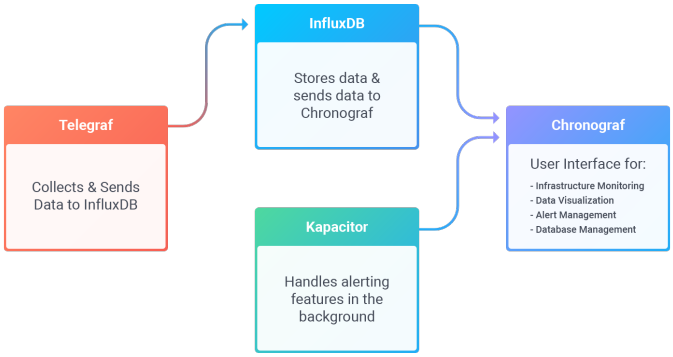

Figura 2.1: Architettura pacchetto TICK. [23]

2.2 InfluxDB v2.0

All'inizio del 2019 il team di InfluxDB ha introdotto la definizione di una nuova versione di InfluxDB che potesse rappresentare un toolkit essenziale per la gestione delle serie temporali consentendo l'utilizzo di dashboard, query, tasks tutti in un'unica piattaforma [24].

Il nuovo sistema velocizza tutto il processo di gestione e di analisi dei dati e va ad introdurre dei cambiamenti concettuali rispetto alla versione precedente. In particolare i cambiamenti sono: L'integrazione del pacchetto TICK in un'unica soluzione ovvero InfluxDB v2, il linguaggio Flux sostituisce InfluxQL come linguaggio di default, le query continue sono sostituite dai tasks e i database di InfluxDB 1.x diventano i bucket di InfluxDB v2.

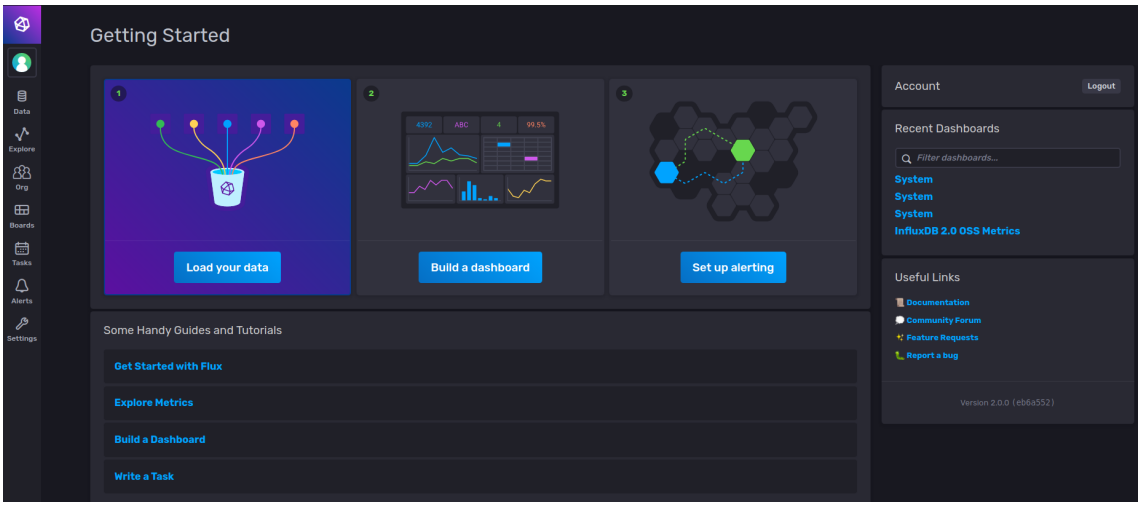

Figura 2.2: GUI Web InfluxDb v2.0.

2.2.1 Measurement e Time Series Data

In InfluxDB non è presente la classica divisione in tabelle caratterizzate dalla presenza di righe e colonne dei database relazionali, al loro posto ci sono i measurement. I measurement, letteralmente misurazione, è un nome utilizzato come identificativo per indicare i contenitori di time series data i quali sono scritti come stringhe nel database. Nei measurement la componente principale sono le colonne che consentono di organizzare dei tipi di dato.

- Una colonna time obbligatoria necessaria a registrare il timestamp al momento dell'inserimento dei dati.
- Una o più colonne field necessarie per contenere il valore del dato, o dei dati nel caso di più colonne, rilevanti per l'indagine.
- Da zero a più colonne tag contenenti informazioni aggiuntive sul dato in esame.

Ogni TSD è costituito da molteplici valori, l'insieme dei valori salvati prende il nome di point. La struttura dei point riprende quella dei measurement, quindi sono presenti: un timestamp in relazione alla colonna time, almeno un valore field e uno o più valori aggiuntivi per le colonne tag [25]. Nel campo field deve sempre essere presente almeno un valore che corrisponde al dato rilevato che si vuole esaminare, questo viene salvato seguendo la struttura fieldKey=fieldValue interpretato come stringa=tipoDato. La stessa struttura si presenta per il campo tag dove si trova tagKey=tagValue, la presenza dei tag non è obbligatoria ma raccomandata poiché aumentano la quantità di informazione connessa al dato salvato e per Time Series con molti point si può incorrere in dati con espressività ridotta senza la presenza di tag. Il campo time serve a salvare il timestamp nel formato desiderato o nel formato standard RFC3339, è utilizzato come chiave primaria poiché essendo i dati salvati uno alla volta ogni point avrà un timestamp differente.

Una caratteristiche importante riguarda la struttura dei measurement, questa non è vincolata e limitata dai primi point salvati -quindi dai tag e field presenti- ma ogni point può avere un numero di attributi differenti anche in relazione allo stessa tipologia di misurazione. Questa caratteristica garantisce una minore necessità di manutenzione derivante da cambi di specifiche dando al database la possibilità di evolvere senza una stringente necessità di frequenti aggiornamenti della struttura.

2.2.2 Flux

Flux è un linguaggio di scripting nato con il contributo della comunity open source guidata dall'interesse nei confronti dei time serie data. È specifico per interrogare, analizzare e intervenire sui dati in un'ottica di usabilità, flessibilità e condivisione [26]. Queste caratteristiche trovano la loro concretizzazione nella presenta di un'interfaccia a linea di comando e di una UI web da cui è possibile eseguire

_time	_value	_field	_measurement	city
2020-09-08 15:40:50 GMT+2	11	temperature	weather	Bologna
2020-09-08 15:41:00 GMT+2	15	temperature	weather	Bologna
2020-09-08 15:41:10 GMT+2	13	temperature	weather	Bologna
2020-09-08 15:41:21 GMT+2	12	temperature	weather	Bologna

Figura 2.3: Esempio di points nel database.

comandi point-and-click, nella possibilità per gli sviluppatori di definire nuove funzionalità del linguaggio per casi d'uso particolari che possono, una volta testate, essere aggiunte all'elenco delle funzioni del linguaggio, nell'abilità di integrare e collaborare con sistemi esterni come database, APIs di terze parti e tools di analisi [27].

Con Flux sono stati introdotti due concetti chiave, ovvero la tabella e l'operatore pipe-forward |>. La tabella è alla base del funzionamento di Flux, in particolare è l'oggetto che viene manipolato, elaborato e generato tramite le funzioni definite dall'utente. L'operatore pipe-forward viene utilizzato come strumento base per concatenare le operazione da eseguire sulle tabelle, che sono l'output delle funzioni e operazioni in Flux. Grazie all'utilizzo di questo operatore è possibile indirizzare la tabella risultante da un'operazione nella successiva dove i dati possono essere sottoposti ad un'ulteriore manipolazione. Quindi questo operatore consente di creare query sofisticate senza complicare il processo di concatenamento [26].

Sintassi Base

In questa sezione verrà presentata la sintassi base di Flux [28].
È possibile assegnare una variabile tramite l'utilizzo dell'operatore =.

```
{
    > s = "stringa" // string
    > i = 1 // integer
    > f = 2.0 // float
}
```

Con lo stesso operatore è possibile andare a dichiarare degli oggetti. I valori al loro interno possono avere tipologie di dati differenti.

```
{
    > obj = {nome: "Carlo", eta: 34, peso: 72.5}
}
```

È possibile anche creare un oggetto con un'annotazione ridotta, dove il valore verrà considerato come il nome della chiave.

```
{
    > obj2 = {a, b, f}
    // equivalente a dire obj2 = {a: a, b: b, f: f}
}
```

Gli oggetti creati possono essere acceduti tramite due tipologie di notazione: la notazione tramite punto e quella tramite parentesi quadre. Quella tramite parentesi quadre è consigliata in presenza di chiavi con spazi bianchi o caratteri speciali.

```
{
    > obj.nome
    Carlo
    > obj["peso"]
    72.5
}
```

Oltre agli oggetti Flux prevede la presenza degli arrays dove i dati devono avere lo stesso tipo di dato.

```
{
    > arr = [5, 10, 15, 10]
}
```

Tramite Flux è possibile anche implementare e utilizzare delle funzioni con o senza parametri di input.

```
{
    > square = (n) => {
        return n * n
    }
    > square(n:3)
    9
}
```

Come anticipato in precedenza tramite l'operatore pipe-forward è possibile andare a concatenare una serie di operazioni o funzioni. Il contenuto della risposta della prima manipolazione viene passato come input nella successiva.

```
{
    data |> funzione() |> funzione2()
}
```

Interrogazione e Manipolazione Dati

In questa sezione sono presentate, tramite una query di esempio, le componenti base delle query in Flux, ovvero: data source, time range e data filters [29].

Tramite la finzione *from()* viene definita la fonte del flusso di dati, il bucket. Tramite l'operatore pipe-forward è possibile passare il risultato di questa funzione alla funzione *range()* la quale serve per definire l'intervallo temporale dei dati da considerare.

```
{
    from(bucket:"bucketEsempio")
        |> range(start:-1h)
        ...
}
```

A questo punto i dati che rientrano nel range sono passati alla funzione *filter()* per ottenere un insieme di dati basato su attributi o colonne. La funzione ha un parametro *fn* che attende una funzione anonima. I dati sono passati alla funzione *filter()* come un record *r*. La funzione anonima valuta se i dati rispettano i filtri imposti. I filtri sono concatenati dall'operatore **and**.

```
{
        ...
        |> filter(fn: (r) =>
            r._measurement == "cpu" and
            r.host == "serverA"
        )
        ...
}
```

In fine vi è la funzione *yield()* necessaria per generare la tabella con i dati filtrati come output. Questa funzione è obbligatoria nel caso in cui sono presenti più queries nella stessa query.

```
{
        ...
        |> yield(name:"result")
}
```

Si è arrivati a definire una query che prende i dati da un bucket chiamato "bucketEsempio", di questi dati considera solo quelli che sono stati salvati da non più di un'ora. Infine mantiene validi solo quelli dove i valori di measurement e host corrispondono a ciò che è indicato nel filtro.

In InfluxDB oltre a eseguire query per filtrare i dati è spesso necessario manipolarli. È possibile utilizzare sia funzioni già presenti in Flux che crearne di nuove [30]. Riprendo l'esempio della query precedente prima di chiamare la funzione *yield()* è possibile andare a manipolare i dati grazie alla funzione *aggregateWindow()*. Questa funzione prende un insieme di valore della tabella, li aggrega e li trasforma in un nuovo valore.

```
{
        ...
        |> aggregateWindow(every: 10m, fn: mean)
        |> yield(name:"result")
}
```

In questo esempio i dati sono divisi in partizioni ogni dieci minuti e viene calcolata la media di ogni partizione come nuovo dato. Durante questa operazione il dato creato manca del proprio timestamp, per questo la funzione prevede un meccanismo automatico di generazione del timestamp sulla base di quello dei valori che sono stati utilizzati per individuare il valore.

2.2.3 Alert

Un Alert è un meccanismo di monitoraggio e notifica dei time series data messo a disposizione da InfluxDB v2.0 il cui funzionamento dipende da tre componenti: check, notification rule e notification endpoint. La creazione di un alert prevede tre passaggi che partono dalla creazione del check, necessario a monitorare i dati e assegnare uno stato, passando per l'aggiunta di un notification endpoint, utilizzato per inviare notifiche a terze parti, arrivando alla creazione di una notification rule per controllare gli stati e inviare notifiche ai notification endpoints personali [31].

I check in InfluxDB v2.0 servono ad interrogare i dati ed applicare uno stato un un livello a ciasun point sulla base delle condizioni specificate. Esistono due tipologie di check:

- Threshold checks assegano uno stato in base a un valore superiore, inferiore, interno o esterno alle soglie definite.
- Deadman checks assegnano uno stato ai dati quando una serie o un gruppo non genera rapporti per un periodo di tempo specificato.

2.2.4 Task

Un Task è uno script in flux pianificato, con una cadenza decisa dall'utente, che permette la gestione di un flusso di dati. Il flusso di dati in input corrisponde ad una serie temporale e, di base, viene modificato e/o analizzato grazie al task attraverso cui è possibile anche memorizzare i dati in un nuovo bucket o eseguire altre operazioni. I Task, in InfluxDB v2.0, possono essere gestiti tramite operazioni di creazione, lettura, aggiornamento ed eliminazione [32].

Ogni task è caratterizzato da quattro componenti essenziali:

- Task options: Definisce le informazioni necessarie per la creazione ed il funzionamento del task:

  - Name: Nome del task.
  - Every: Intervallo di esecuzione del task.
  - Cron: Cadenza di esecuzione del task.
  - Offset: Ritardo da applicare all'esecuzione del task.
  - Concurrency: Numero esecuzioni task consentite contemporaneamente.
  - Retry: Numero tentativi esecuzione del task prima di considerarlo fallito.

- Data source: Utilizzato per indicare la fonte da cui ricavare il flusso di dati e gli eventuali filtri da applicare.
- Data processing or transformation: Definisce il processo di modifica o trasformazione da applicare sul flusso di dati tramite uno script in flux.
- Destination: Indica la posizioni in cui salvare le informazioni ottenuto dalla manipolazioni dei dati. Può essere un bucket o un measurement.

Capitolo 3

Progettazione e Implementazione

3.1 Obiettivi

Lo scopo principale della tesi è quello di generare una collezione di funzioni in JavaScript. fornite di documentazione, corrispondenti ad alcune delle APIs di InfluxDB v2.0 .

Tramite l'utilizzo di questa libreria l'utente avrà modo di interfacciarsi con le APIs fornite da InfluxDB v2.0 in maniera più semplice e rapida in quanto, per l'utente, viene eliminata la questione riguardante il completamento dei campi e l'effettuazione delle chiamate HTTP ai servizi offerti da InfluxDB v2.0.

3.2 Tecnologie

Lo sviluppo del lavoro ha reso necessaria l'integrazione di più componenti al fine di ottenere le funzionalità prefissate. Di seguito sono analizzate le tecnologie utilizzate con riguardo nei confronti della descrizione delle scelte progettuali effettuate.

HTTP

HTTP è un protocollo generico e stateless che lavora con un'architettura di tipo client-server, ci sono quindi due ruoli principali: il client che attiva la connessione e richiede dei servizi e il server che accetta la connessione e restituisce la risposta. La comunicazione presenta due tipi di messaggi quelli di richiesta e quelli di risposta, in entrambi i casi, data la natura di protocollo generico, vi è la possibilità di trasmettere qualsiasi tipologia di risorsa

I messaggi HTTP sono costituiti da un intestazione (header) ed un corpo (body). Il body può contenere le informazioni da trasmettere tramite la connessione oppure rimanere vuoto. L'header contiene metadati, come ad esempio le informazioni di codifica e i metodi HTTP in caso di richiesta.

HTTP differisce da altri protocolli di livello 7 per il fatto che le connessioni vengono generalmente chiuse una volta che una particolare richiesta, oppure una

serie di richieste correlate, è stata soddisfatta. Questo comportamento rende il protocollo HTTP ideale per il World Wide Web, in cui le pagine molto spesso contengono dei collegamenti a pagine presenti su server esterni diminuendo così il numero di connessioni attive limitandole a quelle effettivamente necessarie con aumento quindi di efficienza sia lato server che client. Talvolta però pone problemi agli sviluppatori di contenuti web, perché la natura stateless della sessione di navigazione costringe ad utilizzare dei metodi alternativi, tipicamente basati sui cookie, per conservare lo stato dell'utente

JavaScript

JavaScript è un linguaggio interpretato, dove il codice non viene compilato ma è eseguito direttamente. Ha una sintassi simile ai linguaggi di programmazione C, C++ e Java e funzionalità tipiche dei linguaggi di programmazione ad alto livello, inoltre, è un linguaggio orientato agli oggetti. Viene comunemente utilizzato nella programmazione web sia lato client che lato server

In JavaScript lato client è presente un aspetto importante che da una forte caratterizzazione a questo linguaggio. Troviamo la particolarità che il codice lato client non viene eseguito sul server ma direttamente sul client, questo approccio porta un vantaggio nel momento in cui si presentano script particolarmente complessi evitando un sovraccarico di lavoro nei confronti del server grazie alla delega del lavoro verso il client [33].

API RESTful

L'acronimo API sta per Application Programming Interface è racchiude l'insieme delle applicazione che, attraverso modalità standard e predefinite forniscono le proprie funzionalità ad applicazioni terze. Attraverso il loro utilizzo viene semplificato il rapporto di dialogo tra più applicazioni poiché, favorendo la riusabilità. danno la possibilità di evitare ridondanze ed evitare la replicazioni di codice generando un ambiente più coeso e comprensibile.

Con il tempo accanto al concetto di API si è sviluppato il modello architetturale REST -REpresentational State Transfer- il quale è uno stile di architettura software per sistemi distribuiti. Un principio importante di un sistema REST è rappresentato dell'interfaccia uniforme, ovvero il principio secondo il quale le convenzioni uniformi sono alla base del rapporto di interazione tra le componenti di un sistema. Per ottenere un sistema REST è necessario rispettare dei vincoli precisi:

- identificazione delle risorse: deve essere definito un meccanismo per l'identificazione delle risorse presenti nel sistema.

- manipolazione delle risorse tramite rappresentazione: le risorse non possono essere accedute direttamente, è il sistema che deve fornire delle loro rappresentazioni.

- hypermedia come il motore per la gestione dello stato dell'applicazione: ogni risorsa contribuisce alla rappresentazione dello stato dell'applicazione e deve fornire l'accesso ad eventuali risorse collegate

Queste caratteristiche definiscono un sistema dove il funzionamento base vede un client richiedere una rappresentazione di una risorsa al server ed il server restituire come risposta la risorsa associata ai collegamenti di altre risorse collegate. Questa comunicazione è resa possibile dall'utilizzo combinato di una struttura degli URL definita insieme ai verbi HTTP per il recupero (GET) e per la modifica (POST, PUT, PATCH, DELETE).

Funzioni Asincrone

Attraverso la programmazione asincrona è possibile consentire ad un'unità di lavoro di funzionare separatamente rispetto al thread principale, notificando quando avrà finito l'esecuzione. Un thread può essere definito come una sequenza di istruzioni da eseguire che nel contesto di JavaScript sono necessariamente sequenziali data la sua natura single thread.

Ad oggi, in JavaScript, le funzioni asincrone vengono gestite tramite l'utilizzo degli operatori async e await che si basano sul meccanismo delle Promise, ovvero operazioni incomplete al momento corrente ma che saranno completata in futuro. Una Promise può avere 3 stati gestiti tramite i metodi then() e catch():

- Pending: Promise in esecuzione, stato sconosciuto.
- Fulfilled: Promise ha ottenuto un risultato visibile nel blocco then().
- Rejected: Promise ritorna un errore visibile nel blocco catch().

Abbiamo quindi che la parola async consente di definire una funzione come asincrona, ovvero specifica la presenza al suo interno di una funzione asincrona, mentre la parola await sospende l'esecuzione di una funzione in attesa che la Promise associata ad un'attività asincrona venga risolta o generi un errore [33].

Fetch

Fetch è uno standard JavaScript usato per la gestione delle richieste HTTP dove è possibile andare a specificare:

- Method: Stringa che specifica quale metodo HTTP verrà utilizzato nella richiesta.
- Headers: Oggetto utilizzato per indicare gli headers da includere nella richiesta HTTP.
- Body: Stringa contenente tutti i dati che verranno forniti al server tramite la richiesta.

È caratterizzato da una sintassi semplice e da una buona integrazione nel sistema ad oggetti proposto da JavaScript e può essere utilizzato per la gestione delle chiamate asincrone grazie alle Promise. In particolare, in caso di successo, la Promise sarà risolta e la risposta verrà fornita tramite l'oggetto Response il quale fornisce vari metodi di accesso alla risposta.

Metodo	Descrizione
text()	Restituisce il contenuto sotto forma di testo
json()	Effettua il parsing del contenuto e lo restituisce sotto forma di oggetto
blob()	Restituisce il contenuto sotto forma di blob (dato binario con tipo)
arrayBuffer()	Restituisce il contenuto strutturato in un arrayBuffer

È importante precisare che la Promise restituita da fetch viene risolta sempre in presenza di qualsiasi risposta fornita dal server e non solamente in presenza di un codice di stato 2xx. Risulta quindi importante andare a gestire questa caratteristica tramite sistemi di verifica appositi [34].

OpenAPI e Swagger

OpenAPI e Swagger sono due entità che operano coese per consentire la creazione di documentazione per le API RESTful.

OpenAPI è il nome ufficiale della specifica il cui sviluppo è promosso da più di 30 organizzazioni tra cui sono presenti Google, Microsoft e IBM. Definisce un'interfaccia standard per le API RESTful, la quale può essere scritta in YAML o JSON, dando la possibilità di includere nella descrizione: gli endpoint con le operazioni associate, i parametri di input e output e metodi di autenticazione [35]. Ad oggi è presente la versione 3.0 di OpenAPI questa, rispetto alla versione 2.0 (nota anche come Swagger 2) propone dei cambiamenti importanti [36]:

- Introduzione della possibilità di inserire hosts multipli.
- Ridefinizione del concetto di components che ha reso il sistema più modulare e riutilizzabile.
- Possibilità di definire il corpo di una richiesta e della risposta separatamente rispetto ai parametri della richiesta.
- Introduzione del supporto ai link per consentire la descrizione del collegamento tra percorsi.
- Nuovo sistema di descrizione degli esempi di risposte e richieste.

Swagger, invece, è il nome associato ad alcune delle tecnologie e degli strumenti maggiormente utilizzati per generare i documenti contenenti l'implementazione della specifica OpenAPI [37].

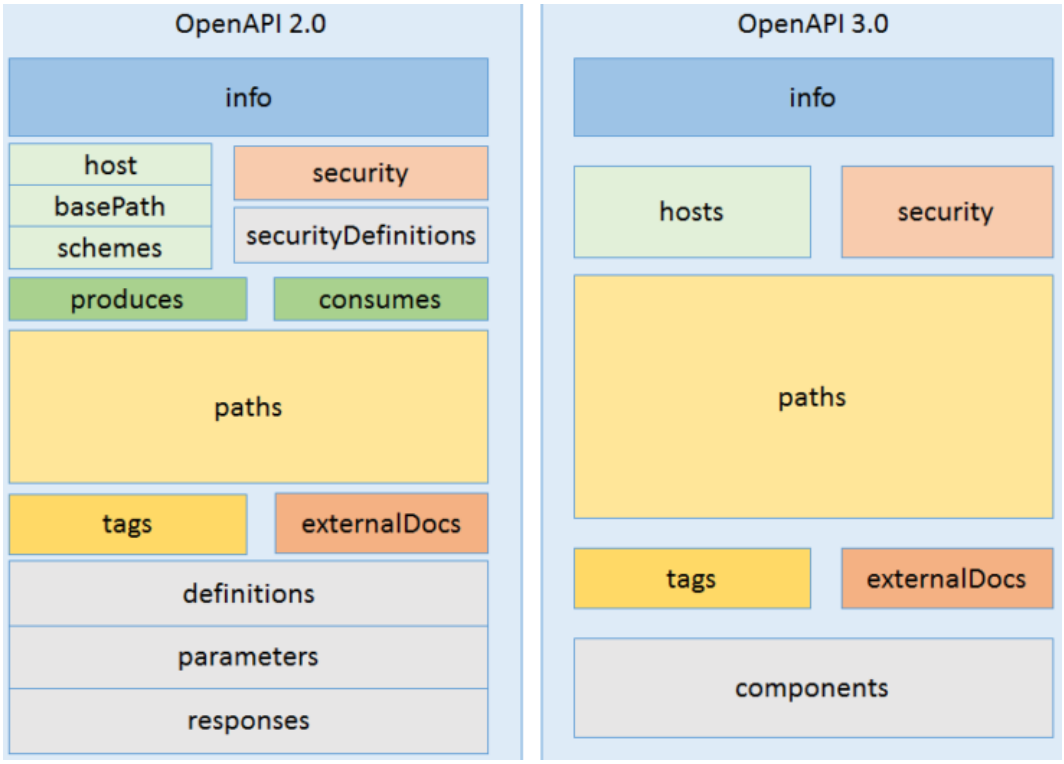

Figura 3.1: Differenze strutturali swagger 2 vs OpenAPI 3.0. [38]

3.3 Scelte Effettuate

Per effettuare le richieste HTTP è stato scelto di utilizzare Fetch in quanto è una tecnologia nativa dei browser e quindi non è necessario integrarla da fonti esterne, consente di usare le Promise in sostituzione dei callback ed è compatibile con la maggioranza dei browser comunemente utilizzati. Decidendo di utilizzare Fetch sono state anche accettate le sue funzionalità negative tra cui troviamo:

- Il processo in due step per gestire i JSON. Dove il primo serve a fare la richiesta e il secondo a chiamare il metodo .json() sulla risposta per ottenerla nel formato desiderato.
- La gestione degli errori. Le Promise ritornate da Fetch hanno la particolarità di gestire gli errori solo lato network e non gli errori HTTP anche se ed essere presente è un codice di stato delle classi 4xx o 5xx.

Di seguito è riportato il codice comune a tutte le funzione per gestire gli errori che possono insorgere durante la richiesta HTTP.

```
async function handlerError(response) {
    if (!response.ok) {
```

```
        let resp = await response.json();
        return resp;

    } else {

        try {

            let resp = await response.json();

            let obj = {"json" : resp,
                       "response" : response.ok};
            return obj;

        } catch (error) {

            let obj = {"response" : response.ok};
            return obj;

        };
    }
}
```

Questo codice è richiamato da ogni funzione della libreria ed in particolare è presente nel return. Questa tipologia di implementazione consente di andare a verificare il contenuto della risposta lato server per individuare l'effettivo risultato della richiesta.
Nel if viene controllato il codice di stato della risposta generata, se questo non è della classe 2xx allora la Promise viene risolta e il suo contenuto restituito alla funzione chiamante, mentre se il codice di stato risulta essere della classe 2xx allora si entra in un costrutto try/catch. Nel try viene risolta la Promise, se questa risulta avere un body allora viene creato un oggetto contenente il body e il codice di stato che viene restituito alla funzione chiamante, mentre, se non è presente il body ad essere restituito è solamente il codice di stato.

Per il progetto è stato scelto di utilizzare la versione 3.0 di OpenAPI rispetto a Swagger 2. Questa scelta è stata dettata dalla preferenza nei confronti dell'approccio più modulare e riutilizzabile usato da OpenAPI 3.0 per definire le API. Questo, infatti, consente di descrivere i modelli di richiesta, di risposta, i dettagli dei componenti e la gestione delle impostazioni di sicurezza in maniera maggiormente versatile.

3.4 Funzionalità

Il progetto di tesi è stato impostato per implementare le funzioni relative alle funzionalità riguardanti quattro macro categorie: Bucket, Dati, Task, Alert.

Bucket

In InfluxDB v2.0 un bucket è un nome che viene attribuito ad una posizione (può essere inteso come la tabella del modello relazionale) in cui sono salvati e conservati i dati delle serie temporali. Tutti i bucket sono caratterizzati da una politica di conservazione attraverso cui viene definito l'intervallo di tempo durante il quale i dati saranno conservati e da un'organizzazione, ovvero un spazio di lavoro, di appartenenza.
Attraverso l'utilizzo della libreria è possibile andare ad ottenere l'elenco dei bucket di un'organizzazione, creare, modificare ed eliminare un bucket. Per ottenere queste funzionalità è necessario fornire le giuste informazioni specificate nella documentazione.

```
{
  "orgID": "string",
  "name": "string",
  "description": "string",
  "rp": "string",
  "retentionRules": [
    {
      "type": "expire",
      "everySeconds": 30000
    }
  ]
}
```

In esempio è riportata la struttura dell'oggetto che deve essere passato alla funzione necessaria ad ottenere la creazione di un bucket. Tra i valori presenti non tutti sono obbligatori, in particolare ad essere obbligatori sono: name e description. Per retentionRules è possibile fare un discorso a parte, infatti questo è obbligatorio al fine della creazione ma può essere omesso e in questo caso i dati di quel bucket verranno considerati come senza scadenza.

Dati

Per quanto concerne la gestione dei dati è possibile fare due tipologie principali di operazioni: scrittura di nuovi dati ed interrogazione di dati salvati nei bucket presenti su di una specifica organizzazione.

```
{
  "extern": {...},
  "query": "string",
  "type": "flux",
  "dialect": {...},
  "now": "2020-09-15T10:37:22.648Z"
}
```

In esempio è riportata la struttura dell'oggetto che deve essere passato alla funzione necessaria ad ottenere l'interrogazione dei dati. La componente che caratterizza maggiormente questo oggetto è la query dove deve essere inserita una stringa corrispondente allo script in Flux, linguaggio specifico di InfluxDB v2.0, che si vuole eseguire.

Task

Un task in InfluxDB v2.0 è uno script in Flux pianificato con la finalità di prendere un flusso di dati in input con l'obiettivo di elaborare o trasformare i dati contenuti nel flusso. I dati solitamente sono poi archiviati in un nuovo bucket o manipolati nuovamente, tuttavia, cosa esattamente succede viene definito dal caso d'uso specifico.
Attraverso l'utilizzo della libreria è possibile andare ad ottenere l'elenco dei task presenti, creare, modificare ed eliminare un task.

```
{
  "orgID": "string",
  "org": "string",
  "status": "active",
  "flux": "string",
  "description": "string"
}
```

In esempio è riportata la struttura dell'oggetto che deve essere passato alla funzione necessaria ad ottenere la creazione di un task. Non tutti le voci sono obbligatorie, la più rilevante è quella riguardante flux, infatti è qua che dovrà essere inserito lo script in Flux che si vuole far eseguire dal task.

Alert

In InfluxDB v2.0 un alert serve a monitorare i time series data e a notificare attraverso l'uso combinato di tre componenti: check, notification rule e notification endpoint. Le funzioni create sono state divise su questi tre componenti quindi, attraverso l'utilizzo della libreria, è possibile:

- Ottenere l'elenco dei check presenti, creare, modificare ed eliminare un check e ottenere informazioni specifiche su di un unico check.

- Ottenere l'elenco delle notification rules presenti, creare, modificare ed eliminare una notification rule e ottenere informazioni specifiche su di una unica notification rules.

- Ottenere l'elenco dei notification endpoint presenti, creare, modificare ed eliminare un notification endpoint e ottenere informazioni specifiche su di un unico notification endpoint.

```
{
       "name": "string",
       "orgID": "string",
       "query": {...},
       "status": "active"
       "description": "string",
       "labels": [...],
       "type": "threshold",
       "thresholds": [...],
       "every": "string",
       "offset": "string",
       "tags": [...],
       "statusMessageTemplate": "string"
}
```

In esempio è riportata la struttura dell'oggetto che deve essere passato alla funzione necessaria ad ottenere la creazione di un check di tipo threshold.

3.4.1 Elenco Funzionalità

Di seguito viene riportata una tabella riassuntiva delle funzionalità presenti nella libreria.

Nome	Descrizione	Input	Output
insertData	Scrive time series data in InfluxDB	obj	json
queryDataJson	Esegue query in InfluxDB	obj	json
allBuckets	Ottiene elenco buckets	obj	json
createBucket	Crea un bucket	obj	json
updateBucket	Aggiorna un bucket	obj	json
deleteBucket	Elimina un bucket	obj	json
allTask	Ottiene elenco tasks	obj	json
createTask	Crea un task	obj	json
updateTask	Aggiorna un task	obj	json
deleteTask	Elimina un task	obj	json
allCheck	Ottiene elenco checks	obj	json
singleCheck	Ottiene informazioni singolo check	obj	json
createCheck	Crea un check	obj	json
updateCheck	Aggiorna un check	obj	json
deleteCheck	Elimina un check	obj	json
allRules	Ottiene elenco rules	obj	json
singleRule	Ottiene informazioni singola rule	obj	json
createRule	Crea una rule	obj	json
updateRule	Aggiorna una rule	obj	json
deleteRule	Elimina una rule	obj	json
allEndpoints	Ottiene elenco endpoints	obj	json
singleEndpoint	Ottiene informazioni singolo endpoint	obj	json
createEndpoint	Crea un endpoint	obj	json
updateEndpoint	Aggiorna un endpoint	obj	json
deleteEndpoint	Elimina un endpoint	obj	json

Capitolo 4

Validazione

In questo capitolo viene presentato lo scenario attraverso cui sono state testate alcune delle funzioni JavaScript contenuto nella libreria del progetto di tesi.

4.1 Caso d'Uso

Per andare a testare il corretto funzionamento delle APIs descritte è stato implementato un semplice sito web con una compatta struttura HTML. All'interno del file HTML sono stati inseriti il collegamento al file CSS (`main.css`) dove è stata definita l'interfaccia grafica, il collegamento al file JavaScript principale del sito (`main.js`) dove sono codificate le sue funzionalità e le funzioni necessarie all'ottenimento dei dati sul clima oltre a quelle necessarie per utilizzare le funzioni della libreria e, infine, il collegamento al file JavaScript contenente la libreria InfluxDB (`InfluxWrap.js`).

Figura 4.1: Home page del sito.

Tramite il sito web si ha la possibilità di visionare il meteo della città di Bologna il quale viene aggiornato ogni dieci secondi. Per implementare questa funzionalità ed ottenere le informazioni sui dati meteorologici il sito utilizza l'API di OpenWeatherMap, un servizio online che fornisce dati meteorologici. In particolare tramite una chiamata fetch() presente all'interno della funzione `getResults()` si ottengono le informazioni in formato JSON. Dalla risposta, grazie alla funzione `displayResults()`, vengono prese solamente le informazioni rilevanti per il caso in esame, ovvero: la città, la data, la temperatura, la condizione meteorologica, l'intervallo minimo e massimo della temperatura.

Una volta ottenute queste informazioni viene aggiornata l'interfaccia grafica con i nuovi dati e viene settato un timer, dalla durata di dieci secondi, per andare a richiamare la funzione `getResults()` così da generare un ciclo di dati necessari per testare le funzionalità della libreria.

```
function getResults(query) {

  fetch('${api.base}weather?q=${query}&units=metric&APPID=${api.key
     ↪ }')
    .then(weather => {
      return weather.json();
    }).then(insert);
}
function displayResults(weather) {

  let city = document.querySelector('.location .city');
  city.innerText = `${weather.name}, ${weather.sys.country}`;

  let now = new Date();
  let date = document.querySelector('.location .date');
  date.innerText = dateBuilder(now);

  let temp = document.querySelector('.current .temp');
  temp.innerHTML = '${Math.round(weather.main.temp)}<span>c</span>'

  let weather_el = document.querySelector('.current .weather');
  weather_el.innerText = weather.weather[0].main;

  let hilow = document.querySelector('.hi-low');
  hilow.innerText = '${Math.round(weather.main.temp_min)}c / ${Math.
     ↪ round(weather.main.temp_max)}c';

  setTimeout(function() {getResults(api.city);}, interval);
}
```

Per testare le funzionalità della libreria al caricamento del file HTML vengono chiamate le funzioni nel file `main.js` attraverso l'utilizzo di `'DOMContentLoaded'`, questo evento viene eseguito solo dopo che la pagina HTML è stata caricata e analizzata, senza aspettare il foglio di stile e le immagini.

Nelle funzioni presenti in `main.js` viene definita la struttura dell'oggetto che verrà successivamente passato alla funzione della libreria. La struttura necessariamente cambia da funzione a funzione in relazione ai dati che è necessario fornire, le caratteristiche dei dati sono specificate e documentate nella documentazione OpenAPI.

```
function newBucket() {

  let objectCreateBucket = {
    headers : {
      "token" : influx.token,
      "zapTraceSpan" : ""
    },
    body : {
      "description" : "",
      "name" : influx.bucket,
      "orgID" : influx.orgID,
      "rp" : "",
      "retentionRules": [...]
    }
  }

  createBucket(objectCreateBucket, influx.url).then(result => {
    console.log(result);
  }).catch(error => {
    console.log(error);
  });
}
```

In esempio è riportata la funzione `newBucket()` utilizzata per andare a chiamare la funzione `createBucket()` presente nella libreria, è possibile notare come sia necessario fornire il token di autenticazione proprio dell'utente e altre informazioni tra cui il nome del bucket da creare e l'ID dell'organizzazione in cui creare il bucket.

4.2 Funzionalità Inserite

Di seguito vengono descritte le funzioni testate attraverso il sito web, ovvero: creazione bucket, scrittura dati, creazione task, creazione check, creazione notification rules, creazione notification endpoints.

Tramite le funzioni è stato deciso di effettuare la creazione di due bucket nominati *bologna* e *temperatureMean*. Un bucket è un nome che viene attribuito ad una posizione dove vengono salvati i time series data, può essere inteso come una tabella del modello relazionale. Nei bucket creati sono salvati rispettivamente i dati sulla temperatura ottenuti tramite la chiamata alla API di OpenWeatherMap e la media della temperatura ottenuta dal lavoro di monitoraggio del task.

_time	_value	_field	_measurement	city
2020-09-29 06:41:51 GMT+2	11	temperature	weather	Bologna
2020-09-29 06:42:01 GMT+2	11	temperature	weather	Bologna
2020-09-29 06:42:11 GMT+2	11	temperature	weather	Bologna
2020-09-29 06:42:22 GMT+2	11	temperature	weather	Bologna

Figura 4.2: Visualizzazione dati tramite GUI Web InfluxDB.

Dopo la creazione dei bucket viene creato il task *temperatureMean* per eseguire lo script Flux riportato sotto.

```
option task = {
    name: "temperatureMean",
    every: 1m
}

data = from(bucket: "bologna")
    |> range(start: -task.every)
    |> filter(fn: (r) => (
        r._measurement == "weather" and
        r._field == "temperature")
    )

data
    |> aggregateWindow(
        every: task.every,
        fn: mean
    )
    |> to(bucket:"temperatureMean", org:"tesi")
```

Questo script andrà a definire un task per monitorare i dati presenti nel bucket *bologna* ad intervalli di sessanta secondi calcolando la media dei nuovi dati inseriti sulla temperatura e salvandola nel bucket *temperatureMean*.
In seguito alla definizione del task vengono create tutte le componenti necessarie per la definizione di un alert, ovvero un check associato a un notification endpoint e una notification rule. Il check eseguirà lo script in Flux, riportato in seguito, tramite

il quale verrà imposto al check di monitorare il bucket *temperatureMean* ogni sessante secondi per segnalare la presenza di valori di temperatura non conformi alle specifiche, nel caso in esame segnalerà tutti i valori superiori ad 8 tramite un messaggio di errore con template `Check: check_name is: r._level` .

```
from(bucket: "temperatureMean")
    |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
    |> filter(fn: (r) =>
        r["_measurement"] == "weather")
    |> filter(fn: (r) =>
        r["_field"] == "temperature")
    |> aggregateWindow(every: 1m, fn: mean)
    |> yield(name: "mean")
```

La gestione della notifica è gestita dal notification endpoint chiamato *temepratureEndpoint* e dalla notificationRule chiamata *temperatureRule*.

TIME	LEV...	CHECK	NOTIFICATION RULE	NOTIFICATION ENDPOI...
2020-09-16 13:32:00	crit	temperatureCheck	temperatureRule	temperatureEndpoint
2020-09-16 13:30:40	crit	temperatureCheck	temperatureRule	temperatureEndpoint

Figura 4.3: Elenco valori individuati fuori schema definito.

Come anticipato nel capitolo sulla progettazione e implementazione per la definizione delle funzioni è stato adottato un approccio asincrono tramite l'utilizzo delle strutture async/await in combinazione con Fetch. Essendo, per loro natura, le funzioni della libreria simili è stato deciso di discutere il funzionamento di una come esempio generale per tutte le altre.

Creazione bucket

Come esempio è proposta la funzione utilizzata per la creazione di un bucket. I parametri di input sono:

- Oggetto contenente le specifiche per la creazione del bucket, questo può contenere riferimenti ai valori presenti nel header, nel body e parametri di query.
- Url di riferimento a cui verrà aggiunta la parte corrispondente alla funzione invocata e i parametri della richiesta se presenti .

Nella funzione viene effettuata una chiamata asincrona, utilizzando Fetch, posta all'interno di un costrutto try/catch per la gestione degli errori. La gestione degli

errori è gestiti da due componenti: il try/catch esposto nel codice seguente e dalla funzione `handlerError()` discussa nel capitolo sulla progettazione ed implementazione; grazie all'uso combinato di queste componenti si riescono a gestire e catturare tutte le tipologie di errori che possono nascere sia lato server che lato client. Seguendo le specifiche di Fetch ogni funzione ha esposto il proprio metodo HTTP necessario per la chiamata, in questo caso POST ma sono presenti anche DELETE, PATCH e GET. In seguito vengono gestiti i dati e passati come input nel body, quando presente e sempre in formato JSON, e nel header nel quale è sempre necessario il token di autenticazione per eseguire la richiesta.

```
async function createBucket(obj, url) {

    var url = new URL(url + "/buckets");

    try {
        let response = await fetch(url, {
            method : "POST",
            headers: {
                "Authorization" : obj.headers.token,
                "Zap-Trace-Span" : obj.headers.zapTraceSpan
            },
            body : JSON.stringify(obj.body)
        })

        return handlerError(response);

    } catch (error) {
        console.log(error);
        return error;
    }
}
```

Capitolo 5

Conclusioni e Sviluppi Futuri

Nel progetto di tesi è stato presentato il processo che ha portato alla definizione e creazione di una libreria JavaScript per l'utilizzo di API fornite da InfluxDB v2.0. La presentazione parte descrivendo e analizzando il contesto attuale per quando riguarda le tematiche teoriche sulle serie temporali e sul mondo del Internet of Things, tutto associato ad una presentazione di casi d'uso realistici per presentare una analisi più completa. Prosegue con un approfondimento, sempre teorico, sui time series database, andando ad analizzare la situazione attuale e proponendo un excursus sulle tipologie di database maggiormente diffuse ponendo enfasi sui criteri di valutazione di questa tipologia di database. In seguito viene introdotto un capitolo specifico su InfluxDB dove sono descritte le principali funzionalità del sistema, andando a porre una distinzione tra le versioni antecedenti alla 2.x e la versione 2.0. Sono quindi descritti i principali componenti caratterizzanti InfluxDB, ovvero, per le versioni prima dalla 2.x viene discusso il pacchetto TICK mentre, per la 2.0, vengono analizzati il linguaggio Flux, il sistema di monitoraggio tramite gli alert e la struttura interna del database con i measurement e i time series data. Si arriva quindi a parlare della parte di progettazione e validazione riassumendo gli obiettivi e descrivendo le principali tecnologie utilizzate, sono poi descritte le scelte progettuali effettuate e introdotte le funzionalità del progetto attraverso l'utilizzo di esempi i codice. Nel capitolo sulla validazione viene descritto il caso d'uso utilizzato per controllare il corretto funzionamento della libreria e viene discusso in modo accurato il processo implementato descrivendo in parallelo le funzioni utilizzate per ottenerlo.

5.1 Sviluppi Futuri

Lo sviluppo naturale di questo progetto di tesi consiste nell'andare ad implementare tutte le API di InfluxDB v2.0 non facenti parte della libreria creata. Andandole a testare in casi d'uso alternativi per confermare il corretto funzionamento anche in presena di operazione intensive sui time series data. Sarebbe inoltre opportuno mantenere aggiornato i metodi di effettuazione e gestione delle chiamate HTTP

per fornire sempre un sistema funzionante e aggiornato oltre a fornire il supporto per altri TSDB.

Bibliografia

[1] E. Friedman and T. Dunning, *Time Series Databases: New Ways to Store and Access Data*. O'Reilly Media.

[2] What is time series data? - influxdata. [Online]. Available: https://www.influxdata.com/what-is-time-series-data/

[3] C. Chatfield, *The Analysis of Time Series: An Introduction*. Routledge.

[4] Imdadullah, "Time series analysis."

[5] Dati di serie temporali - azure architecture center. [Online]. Available: https://docs.microsoft.com/it-it/azure/architecture/data-guide/scenarios/time-series

[6] Bigdata - caratteristiche. [Online]. Available: https://www.ibmbigdatahub.com/infographic/four-vs-big-data#:~:text=IBM%20data%20scientists%20break%20big,%2C%20variety%2C%20velocity%20and%20veracity.

[7] Internet of things iot: Overview. [Online]. Available: https://www.internet4things.it/iot-library/internet-of-things-gli-ambiti-applicativi-in-italia//

[8] State of the iot 2018: Number of iot devices now at 7b – market accelerating. [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[9] Architecture for big data - azure architecture center. [Online]. Available: https://docs.microsoft.com/it-it/azure/architecture/data-guide/big-data/#internet-of-things-iot

[10] Internet of things iot: che cos'è, come funziona internet delle cose. [Online]. Available: https://www.zerounoweb.it/analytics/big-data/internet-of-things-iot-come-funziona/

[11] C. André da Costa, C. F. Pasluosta, B. Eskofier, D. Bandeira da Silva, and R. da Rosa Righi, "Internet of health things: Toward intelligent vital signs monitoring in hospital wards."

[12] K. Mahmud, G. E. Town, S. Morsalin, and M. Hossain, "Integration of electric vehicles and management in the internet of energy."

[13] Time series database (tsdb) explained — influxdb. [Online]. Available: https://www.influxdata.com/time-series-database/

[14] What are time series databases?. in this article, we will introduce the... — by alibaba cloud — data driven investor. [Online]. Available: https://medium.com/datadriveninvestor/what-are-time-series-databases-a3e847608f91

[15] S. K. Jensen, T. Bach Pedersen, and C. Thomsen, "Time series management systems: A survey."

[16] A. Bader, O. Kopp, and M. Falkenthal, "Survey and comparison of open source time series databases."

[17] Influxdb 1.x. [Online]. Available: https://www.influxdata.com/time-series-platform/

[18] Influxdb - continuous query and retention policy. [Online]. Available: https://docs.influxdata.com/influxdb/v1.8/guides/downsample_and_retain/#:~:text=Retention%20policy%20(RP)%20is%20the,RPs%20are%20unique%20per%20database

[19] Influxdb - key features. [Online]. Available: https://docs.influxdata.com/influxdb/v1.8/

[20] Influxdb - telegraf. [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/

[21] Influxdb - chronograf. [Online]. Available: https://www.influxdata.com/time-series-platform/chronograf/

[22] Influxdb - kapacitor. [Online]. Available: https://www.influxdata.com/time-series-platform/kapacitor/

[23] Influxdb - tick graph. [Online]. Available: https://docs.influxdata.com/platform/getting-started/

[24] Influxdb v2 - overview. [Online]. Available: https://www.influxdata.com/products/influxdb-overview/influxdb-2-0/

[25] Influxdb - continuous query. [Online]. Available: https://docs.influxdata.com/influxdb/v1.8/query_language/continuous_queries/

[26] Get started with flux. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/query-data/get-started/

[27] Flux open source query language. [Online]. Available: https://www.influxdata.com/products/flux/

[28] Influxdb v2.0 - sintassi flux. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/query-data/get-started/syntax-basics/

[29] Influxdb v2.0 - query with flux. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/query-data/get-started/query-influxdb/

[30] Influxdb v2.0 - transform data with flux. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/query-data/get-started/transform-data/

[31] Influxdb v2.0 - alert. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/monitor-alert/

[32] Influxdb v2.0 - task. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/process-data/get-started/#define-a-data-source

[33] Javascript - overview. [Online]. Available: https://javascript.info/

[34] Javascript - fetch. [Online]. Available: https://javascript.info/fetch

[35] Openapi - overview. [Online]. Available: https://swagger.io/specification/

[36] Openapi 3.0 - news. [Online]. Available: https://blog.restcase.com/6-most-significant-changes-in-oas-3-0/

[37] Swagger - overview. [Online]. Available: https://swagger.io/docs/specification/about/

[38] Openapi 3.0 - overview. [Online]. Available: https://www.openapis.org/news/blogs/2016/10/tdc-structural-improvements-explaining-30-spec-part-2

Elenco delle figure