Relazione progetto BigData: Covid-19 tracking

BigCovid team, Alessio Verdolini, Federico Gaglio

A.A 2019-2020

Indice

1 Introduzione al dominio

Il contesto in cui si inserisce il progetto è quello della raccolta ed analisi di dati inerenti alla diffusione dei contagi da corona virus, al secolo Covid-19.
Il tema, essendo estremamente attuale, presenta una vasta gamma di esempi di possibili analisi quantitative del numero dei contagi, decessi e guariti realizzate con differenti livelli di granularità.

Inoltre ogni stato ha provveduto a sviluppare sistemi, spesso OpenSource, volti a raccogliere informazione sui contatti tra individui al fine di poter ricostruire un grafo delle connessioni tra essi finalizzato alla prevenzione.
In questa direzione si sono mossi anche Big dell'informatica come Apple e Google che hanno unito gli sforzi per sviluppare, in tempo record, delle API condivise tra i due sistemi per smartphone più diffusi, IOS e Android, al fine di facilitare la raccolta di dati dagli utilizzatori finali.

L'obiettivo di questo progetto è di simulare alcune modalità di raccolta ed analisi dei dati al fine di valutarne le prestazioni. Il caso d'uso in cui ci poniamo è stato ispirato dal momento storico che stiamo vivendo e cioè quello del lockdown per regione. A tal fine si immagini che le persone possano muoversi "liberamente" solo all'interno della propria regione. Per rendere lo scenario più verosimile possibile immaginiamo che le informazioni vengano inviate in modo automatico dagli smartphone degli utenti ogni qual volta un utente entra in contatto con un altro utente.

1.1 Legenda del dominio

Prima di procedere con le prossime sezioni è utile dare definizione di alcuni concetti chiave all'interno del dominio.

- ***Utente***: è il componente elementare del sistema che compie delle azioni. Ogni dato raccolto ed elaborato fa riferimento ad uno o ad un insieme arbitrario di utenti.
- ***Positivo***: è un utente del sistema che ha contratto il virus e di conseguenza potrebbe trasmetterlo ad altri utenti.
- ***Contagio***: è l'azione che determina la transizione dallo stato sano allo stato positivo di un utente. Il contagio può avvenire a seguito del contatto tra due utenti uno dei quali risulta essere positivo.
- ***Contatto***: per contatto si intende un incontro prolungato tra due individui durante il quale potrebbe verificarsi il contagio di uno dei due ad opera dell'altro.
- ***Guarito***: è lo stato successivo al positivo. In questo modello semplificato non è contemplato il decesso degli individui ma solo la guarigione in modalità eventually.

2 Raccolta, tipologia e produzione dei dati

Per quanto riguarda la raccolta dei dati, per ovvi motivi di privacy, non abbiamo potuto avere accesso ai dati reali. Abbiamo quindi deciso di crearceli noi, in maniera coerente con le informazioni di reale interesse per lo sviluppo della nostra applicazione.

Per la generazione dei suddetti dati abbiamo utilizzato come tool Mockaroo. Mockaroo è un brillante strumento online che consente di risolvere i problemi di generazione dei dati in pochi istanti. Fornisce una GUI per creare alcuni dati che soddisfano le esigenze e consente di generare dati di test realistici nei formati CSV, JSON, SQL ed Excel. Genera dati da uno schema salvato che può essere utile per i test automatizzati.

Nel nostro caso, in particolare, abbiamo deciso di andare a generare tre file csv differenti, che nello specifico rappresentano tre regioni che abbiamo scelto per effettuare la nostra simulazione: Lombardia, Sicilia, Toscana.

I dati presenti nel sistema sono principalmente di tre tipologie che indicano:

- ***L'avvenuto contatto tra due utenti.*** In questa tipologia di dati le informazioni più rilevanti sono sicuramente quelle geo-spaziali. Come si può notare nell'esempio sottostante, i valori dei vari campi che rappresentano le informazioni spaziali sono coerenti tra loro. Ad esempio il cap 25129 ricade realmente all'interno del comune di Brescia, stessa cosa vale per la latitudine e la longitudine.


```
{
    "user_id_1": "L-133412",
    "user_id_2": "L-550863",
    "country": "IT",
    "region": "Lombardia",
    "city": "Brescia",
    "cap": 25129,
    "latitude": 45.4969452,
    "longitude": 10.2740428,
}
```


Figura 1: Esempio evento contatto tra due utenti

- ***La positività di un utente a seguito del test.*** In questo caso i dati sono volutamente molto più sintetici per evitare di andare ad appesantire il sistema di elaborazione. L'unico aspetto importante in questo caso è

la scelta coerente degli id da utilizzare. Per scelta coerente si intende utilizzare, per la produzione di eventi di una determinata regione, gli id degli utenti appartenenti solamente alla regione in questione in modo tale da poter riprodurre lo scenario di lockdown regionale di cui accennavamo all'inizio.


```
{
    "user_id": "L-133412"
}
```


Figura 2: Esempio evento utente positivo

- ***La guarigione di un utente.*** In questo caso le informazioni sono identiche, al livello sintattico, a quelle prodotte nel caso dei positivi. L'unica differenza in questo caso è il significato semantico degli eventi che rappresentato appunto la guarigione di un utente. L'unico costraint rilevante in questo caso riguarda la scelta degli id per la costruzione dell'evento che deve essere selezionato all'interno del pool dei positivi della regione di appartenenza.


```
{
    "user_id": "L-133412"
}
```


Figura 3: Esempio evento utente guarito

Per quanto riguarda la produzione dei dati, abbiamo implementato degli script in Python utilizzati per simulare i contatti, tra utenti, all'interno della regione di appartenenza. I dati vengono prodotti con cadenza semi randomica, al fine di rendere lo scenario più realistico possibile. Le informazioni vengono inserite dai singoli produttori sul canale trasmissivo che consente la comunicazione con il core del sistema, che si occuperà della loro elaborazione.

3 Architettura del sistema

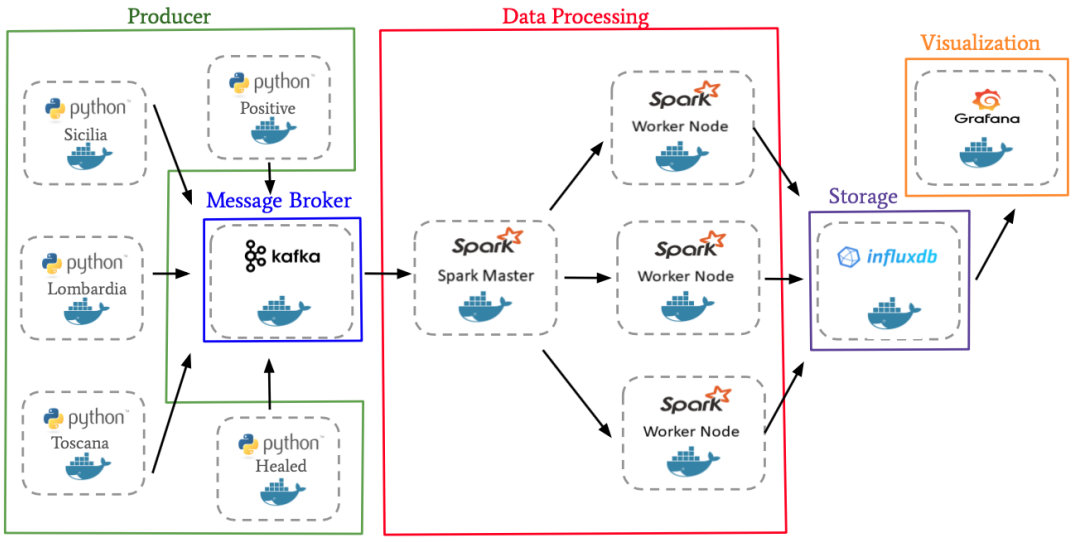

Figura 4: Architettura complessiva del sistema

Nel nostro progetto abbiamo deciso di utilizzare un'architettura che fondamentalmente ci permettesse di poter andare ad effettuare due operazioni fondamentali: Data Collection, per analisi batch e streaming processing per l'analisi near real time.
Nella realtà accade che la nostra applicazione raccoglie i dati sotto forma di flussi di messaggi del canale trasmissivo sul quale sono stati inseriti, va ad effettuare un operazione preliminare di pulizia e validazione dei dati in input e successivamente procede ad effettuare elaborazione dei flussi stessi per estrarre statistiche real time.
Infine i dati vengono salvati persistentemente su sistemi di storage al fine di consentire una successiva analisi in differenti modalità per poter individuare eventuali trend.
Una volta memorizzati i dati elaborati, mediante uno stumento di visualizzazione, è possibile ottenere statistiche di differenti tipoligie inerenti allo stato generare del dominio.

3.1 Docker & Docker-compose

Per poter riuscire a mettere su l'intera architettura, l'idea sostanzialmente è stata quella di utilizzare Docker, ed in particolare **docker compose**.
Docker compose è uno strumento per la definizione e l'esecuzione di applicazioni Docker multi-contenitore. Con Compose, si usa un file YAML per configurare i servizi della propria applicazione. Quindi, con un solo comando, si creano e avviano tutti i servizi dalla nostra configurazione.

3.1.1 Utilizzo

L'uso di Compose è fondamentalmente un processo in tre fasi:

1. Definizione dell'ambiente della propria app con un file Docker in modo che possa essere riprodotto ovunque.
2. Definizione dei servizi che compongono la propria app in docker-compose.yml in modo che possano essere eseguiti insieme in un ambiente isolato.
3. Esecuzione docker-compose up e Compose si avvia ed esegue l'intera app.

3.2 Produzione dei dati

Per quanto riguarda la produzione dei dati riguardanti i contatti sono stati implementati degli script Python che, leggendo le informazioni necessarie dai file .csv generati precedentemente, permettono di immettere all'interno del sistema tutti gli eventi di interesse che verranno utilizzati dai successivi livelli dell'infrastruttura.
I produttori sono stati realizzati in modo parametrico per permetterne il rilascio facilitato in ambiente di esecuzione. Ogni script prende in input la chiave della regione per cui effettuerà la produzione ed in funzione di quello ricostruirà sia il nome del file da cui estrarre le informazioni sia il topic su cui inoltrare l'evento generato.
Per permettere la connessione tra lo script Python e il message broker è stata utilizzata la libreria kafka-python che permette di effettuare la produzione degli eventi su uno dei topic presenti. (https://pypi.org/project/kafka-python/)
Il processo di invio degli eventi è intervallato in modo casuale per rendere lo scenario realistico.
Ogni producer è rilasciato in un container Docker differente in modo tale da permetterne uno scaling indipendente, facilitando notevolmente le operazioni di testing.

```
FROM python:3

ENV REGION=${REGION}

COPY contact-producer.py entrypoint.sh data/* ./

RUN pip install kafka-python

CMD ["sh", "-c", "bash entrypoint.sh $REGION"]
```

Figura 5: Immagine Docker dei producer di contatti

Per quanto riguarda invece gli eventi che prendono in considerazione la positività o la guarigione di un certo utente, le informazioni questa volta vengono estratte direttamente dal db sul quale sono stati salvati precedentemente i dati sui contatti.
Questa scelta si è resa necessaria per evitare che venissero generati positivi non ancora presenti all'interno del sistema oppure guariti che non sono mai risultati positivi. Ovviamente in uno scenario reale, dove i dati di input vengono dalle vere fonti e non da file di esempio, tutto ciò non è necessario.

3.3 Messaging layer

Una volta aver prodotto gli eventi, questi vengono presi in carico dal messaging layer, che nel nostro caso è stato implementato utilizzando Kafka e Zookeper. Come si può vedere nel file di configurazione del docker-compose il funzionamento di Kafka è subordinato alla presenza di Zookeper che si occupa del coordinamento delle istanze del cluster Kafka.
Inoltre in fase di bootstrap del container, vengono creati i topics elencati per la chiave KAFKA_CREATE_TOPICS che permetteranno successivamente sia ai producer che ai consumer di acecdere al canale trasmissivo per il trasferimento delle informazioni.
Il container che viene istanziato resterà in ascolto sulla porta 9092.
La scelta dell'utilizzo di questo message broker all'interno dell'infrastruttura è dettato dal bisogno di creare un'indirezione spaziale e temporale tra i produttori degli eventi e i consumatori, che nel caso specifico sono le varie applicazioni che leggeranno ed elaboreranno i vari eventi del dominio prodotti.
Inoltre uno dei vantaggi di Kafka è che, essendo pensato per sistemi distribuiti, scala facilmente su un numero molto elevato di richieste, offrendo ottime prestazioni anche a fronte di grandi carichi di lavoro.

```yaml
zookeeper:
  image: wurstmeister/zookeeper
  container_name: zookeeper
  ports:
    - "2181:2181"

kafka:
  image: wurstmeister/kafka:latest
  container_name: kafka
  depends_on:
    - "zookeeper"
  ports:
    - "9092:9092"
  environment:
    KAFKA_ADVERTISED_HOST_NAME: kafka
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_CREATE_TOPICS: "lombardia:1:1,sicilia:1:1,toscana:1:1,positive:1:1,healed:1:1"
    KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

Figura 6: Configurazione di Kafka e Zookeper

3.3.1 Kafka

Apache Kafka è un sistema open source di messaggistica istantanea, che consente la gestione di un elevato numero di operazioni in tempo reale da migliaia di client, sia in lettura che in scrittura. La piattaforma si dimostra ideale per la progettazione di applicazioni di alta fascia. Tra i punti di forza di Kafka vi sono l'incremento della produttività e l'affidabilità, fattori che hanno consentito al sistema di sostituire famosi broker di messaggistica come JMS e AMQP. Per un rapido apprendimento di Kafka è preferibile possedere una conoscenza approfondita di Java, di Scala e di Linux.

3.3.2 Zookeper

Un server open source che coordina in modo affidabile i processi distribuiti. Apache ZooKeeper fornisce servizi operativi per un cluster Hadoop. ZooKeeper fornisce un servizio di configurazione distribuita, un servizio di sincronizzazione e un registro dei nomi per i sistemi distribuiti. Le applicazioni distribuite utilizzano Zookeeper per archiviare e mediare gli aggiornamenti a importanti informazioni di configurazione.

3.4 Processing layer

Il processing layer è il core dell'intero sistema. In questo livello vengono processati ed elaborati tutti gli eventi che transitano sui vari topic di Kafka. La struttura di questa porzione dell'architettura è stata realizzata mediante un cluster Spark. Questo cluster prevede due componenti principali:

1. ***Nodi master:*** che si occupano sia della gestione delle richieste esterne sia dello scheduling delle operazioni sui vari nodi worker.

2. ***Nodi worker:*** che effettuano le operazioni richieste dai nodi master.

Per perseguire gli scopi del progetto sono state implementate tre diverse applicazioni che girano contemporaneamente sullo stesso cluster. Ad ognuna delle applicazioni sono affidate differenti responsabilità in termini di gestione delle informazioni:

1. ***Contact spark app:*** questa applicazione ha il compito di elaborare i messaggi che transitano sul topic dei contatti.
   Per prima cosa viene creato uno spark context mediante il quale effettuare le operazione necessarie. Successivamente viene creato uno streaming diretto con Kafka tramite il quale ottenere i messaggi presenti sui vari topic regionali.
   Nel nostro caso sono presenti 3 topic differenti: *Lombardia*, *Toscana*, *Sicilia*. Gli eventi vengono elaborati in batch di 10s creando degli oggetti JSON per facilitarne la lettura e la persistenza.
   La fase di validazione degli eventi prevede una pulizia dell'evento grezzo che viene consumato e la creazione di un entità normalizzata che faciliterà le successive elaborazioni.
   La persistenza viene effettuata su database di tipo Time Series (TSDB) che permettono di effettuare query temporali e soprattutto permettono di scrivere rapidamente dati in modalità append only.

2. ***Positive spark app:*** questa applicazione ha il compito di elaborare i messaggi che transitano sul topic dei positive.
   In questo caso l'elaborazione di questi eventi è molto più semplificata per il semplice motivo che l'evento contiene solamente l'identificativo dell'utente che risulta positivo. L'elaborazione viene effettuata sempre sfruttando mini batch da 10s e la persistenza viene effettuata sempre in modalità append only su un TSDB.

3. ***Healed spark app:*** Quest'ultima app è molto simile alla precedente in termini di funzionamento. L'unica differenza è che in questo caso il topic da cui consuma gli eventi e la relativa modalità di persistenza sul TSDB sono differenti dal precedente. Il topic in questo caso è healed. Per quanto riguarda la persistenza, questa viene sempre effettuata in modalità append only però inserendo uno stato dell'utente specifico. Questo permetterà in fase di retrive dell'aggregato di ricostruire il suo stato applicando gli eventi in ordine di arrivo e di conseguenza sovrascrivendo gli eventi più vecchi con quelli più nuovi. Questa modalità di gestione offre due vantaggi principali: la conoscenza completa dello storico degli eventi avvenuti e la possibilità di conoscere in tempo reale lo stato di un determinato aggregato.

3.4.1 Spark streaming

Apache Spark Streaming è un sistema di elaborazione in streaming scalabile e fault-tolerant che supporta nativamente carichi di lavoro in batch e in streaming.
Spark Streaming è un'estensione dell'API Spark principale che consente ai data engineers e ai data scientist di elaborare dati in tempo reale da varie fonti tra cui (ma non solo) Kafka, Flume e Amazon Kinesis. Questi dati elaborati possono essere inviati a file system, database e dashboard. La sua astrazione chiave è un flusso discreto o, in breve, un flusso DStream, che rappresenta un flusso di dati suddiviso in piccoli batch.
I flussi DS sono basati su RDD, l'astrazione dei dati core di Spark. Ciò consente a Spark Streaming di integrarsi perfettamente con qualsiasi altro componente Spark come MLlib e Spark SQL.
Spark Streaming è diverso da altri sistemi che dispongono di un motore di elaborazione progettato solo per lo streaming o hanno API batch e streaming simili ma compilano internamente su motori diversi. Il motore a singola esecuzione di Spark e il modello di programmazione unificata per batch e streaming portano alcuni vantaggi unici rispetto ad altri sistemi di streaming tradizionali.

3.5 Storage layer

Per quanto riguarda la scelta della tipologia di database da utilizzare, anche se in un primo momento si era scelto di utilizzare Cassandra, per il largo utilizzo dei timestamp che viene fatto, abbiamo scelto come database per la nostra applicazione InfluxDB.
La scelta è ricaduta su questo db sostanzialmente per due motivi:

1. I TSDB sono ottimali per la gestione di informazioni inerenti al monitoraggio nel tempo di domini, anche complessi. Questa ottimalità è garantita sia dalla velocità di scrittura sia dalla capacità di gestire, mediante snapshot o aggregazione dei record, moli di informazioni importanti.

2. Dopo aver capito che la nostra scelta verteva su un time-series db, ci siamo chiesti quale potesse essere il migliore da utilizzare; per capirlo, abbiamo notato che su https://db-engines.com/en/ranking/time+series+dbms , il più "quotato" era proprio influx, come si può notare anche dal grafico seguente.

☐ include secondary database models — 34 systems in ranking, July 2020

Rank Jul 2020	Rank Jun 2020	Rank Jul 2019	DBMS	Database Model	Score Jul 2020	Score Jun 2020	Score Jul 2019
1.	1.	1.	InfluxDB	Time Series	21.86	+0.68	+3.86
2.	2.	2.	Kdb+	Time Series, Multi-model	6.51	+0.66	+0.64
3.	3.	3.	Prometheus	Time Series	4.86	+0.27	+1.40
4.	4.	4.	Graphite	Time Series	3.83	+0.21	+0.40
5.	5.	5.	RRDtool	Time Series	3.05	+0.15	+0.28
6.	6.	↑ 8.	TimescaleDB	Time Series, Multi-model	2.32	+0.14	+1.06
7.	7.	↓ 6.	OpenTSDB	Time Series	2.10	+0.06	-0.20
8.	8.	↓ 7.	Druid	Multi-model	2.05	+0.10	+0.20
9.	9.	↑ 13.	FaunaDB	Multi-model	1.48	+0.28	+1.14
10.	10.	↓ 9.	KairosDB	Time Series	0.73	+0.12	+0.20
11.	↑ 12.	↑ 12.	GridDB	Time Series, Multi-model	0.61	+0.15	+0.24

Figura 7: Ranking InfluxDB rispetto ad altri database time-series

Un altro motivo fondamentale che ci ha portato a scegliere InfluxDB, è stato il fatto che, dopo aver elaborato in un primo momento i dati, ci siamo chiesti quale potesse essere il miglior modo per poter intuitivamente "guardare questi dati", ovviamente una volta fatto lo storage. Anche qui, tra le varie tecnologie, abbiamo visto che Grafana era ottimo per i nostri scopi ed inoltre aveva una fortissima compatibilità con Influx. InfluxDB è un database di tipo NoSQL, nello specifico un databesa column oriented, di conseguenza schemaless, e quindi molto comodo in situazione nelle quali la struttura delle informazioni da salvare risulta essere molto instabile.
Inoltre questa tipologia permette di effettuare agilmente modifiche alla struttura di un singolo record senza ricorrere alla modifica dell'intera base di dati. I concetti fondamentali all'interno di InfluxDB sono:

- ***points***: ovvero una serie di punti. In un certo senso sono assimilabili alle righe di un database relazionale. In particolare le serie temporali sono costituite da coppie "chiave-valore".
- ***measurement***: che corrisponde al concetto di tabella nel modello relazionale mentre la chiave primaria è costituita dal relativo timestamp che può essere generato automaticamente dal db stesso.
- ***tags e fields***: che sono assimilabili alle colonne dei database relazionali e sono molto utili nella costruzione delle query e di conseguenza nel filtraggio dei dati. Si differenziano per il fatto di essere campi indicizzati(tag) o meno(field). Ne deriva quindi il fatto che in grande vantaggio dell'andare ad utilizzare un campo tag è dovuto al fatto che permette di poter andare ad effettuare ricerche puntuali.

Nella nostra applicazione abbiamo scelto di utilizzare due diversi ***measurement***, a seconda dei differenti aspetti che volevamo analizzare, ed ognuno con caratteristiche differenti. I differenti measurement utilizzati sono rispettivamente:

- ***contact***: che sostanzialmente ci è utile dal punto di vista di analisi dei contatti. Ogni record all'interno di questo measurement rappresenta un contatto avvenuto in un determinato posto ad una determinata ora tra due utenti del sitema.
- ***positive***: che ha le stesse funzionalità di contact, solamente che in questo caso, il suddetto measurement, ci permette di poter andare ad effettuare una classificazione sia per utenti positivi, che per utenti guariti. Questo tipo di valutazioni sono rese possibili grazie alla ricostruzione temporale degli eventi. Quando si ha necessità di ricostruire la storia dei contatti di un determinato utente piuttosto che di una determinata regione è possibile rileggere gli eventi in ordine di arrivo, e di conseguenza eventi più recenti che fanno riferimento ad un determinato utente sovrascrivono quelli meno recenti.

Per quanto riguarda i *tags* utilizzati nei due differenti measurements, abbiamo utilizzato i seguenti campi:


```
{
    "results": [
        {
            "statement_id": 0,
            "series": [
                {
                    "name": "contact",
                    "columns": [
                        "tagKey"
                    ],
                    "values": [
                        [
                            "city"
                        ],
                        [
                            "country"
                        ],
                        [
                            "region"
                        ],
                        [
                            "user_1"
                        ],
                        [
                            "user_2"
                        ]
                    ]
                },
                {
                    "name": "positive",
                    "columns": [
                        "tagKey"
                    ],
                    "values": [
                        [
                            "country"
                        ],
                        [
                            "region"
                        ],
                        [
                            "status"
                        ],
                        [
                            "user_id"
                        ]
                    ]
                }
            ]
        }
    ]
}
```


Figura 8: Tag relativi ai vari measurements

Come si può notare, è evidente che si sta parlando dei campi tag, proprio perchè l'output della query sottomessa, restituisce in output i tagKey.
Per quanto riguarda invece i *fields*, anche in questo caso, mostriamo i campi relativi a un solo measurement,inevece che ad entrambi, perchè sono molto simili:

```
{
    "results": [
        {
            "statement_id": 0,
            "series": [
                {
                    "name": "positive",
                    "columns": [
                        "fieldKey",
                        "fieldType"
                    ],
                    "values": [
                        [
                            "country",
                            "string"
                        ],
                        [
                            "region",
                            "string"
                        ],
                        [
                            "status",
                            "string"
                        ],
                        [
                            "user_id",
                            "string"
                        ]
                    ]
                }
            ]
        }
    ]
}
```


Figura 9: Fields relativi al measurement positive

3.5.1 InfluxDB

InfluxDB è un database open source per serie temporali (TSDB Time Series Database in gergo) sviluppato dalla InfluxData e scritto interamente in linguaggio Go.

Un Time Series Database è una base di dati ottimizzata per lavorare con informazioni associate in modo univoco a una dato. Le time series infatti sono misure o eventi che sono raccolti, monitorati, post-elaborati e/o aggregati sulla base del tempo. Questi possono essere metriche di un server, dati di performance di una applicazione, dati network, valori in output da sensori, eventi, e ogni tipo di dati analitici. I time series database sono in grado di gestire grosse quantità di dati sia in scrittura che in lettura, di comprimere i dati salvati sul sistema in maniera efficiente così come gestirne l'accesso in maniera veloce grazie alla definizione di indici, tag e fields.

In un database time series tra l'altro il ciclo di vita dei dati non è necessariamente infinito come nei tradizionali database. Ogni informazione infatti è associabile a un retention policy che stabilisce per quando tempo il dato rimane valido prima di essere eliminato, evitando così di mantenere nel sistema dati di scarso interesse. Una delle sue peculiarità è la totale assenza di dipendenze esterne quindi si tratta di un singolo file eseguibile. Si presta bene a gestire grossi volumi di dati e quindi trova vasta applicazione nell'ambito del monitoraggio di server e sensori ma anche dell'analisi in real time.

La presenza di un linguaggio molto simile ad SQL rende semplicissima la scrittura di query anche molto complesse.

Va subito precisato che InfluxDB rientra nella categoria dei database NoSQL quindi dobbiamo farci un'idea ben precisa della sua struttura prima di sfruttarne tutte le potenzialità.

InfluxDB nasce nel 2013, ed oggi è il database time serie più diffuso e utilizzato a livello globale, anche grazie alla sua natura open source e la sua grande adattabilità che lo rende in grado di essere eseguito su qualsiasi sistema operativo. Alcuni dei punti di forza di InfluxDB database sono da individuarsi nelle sue alte prestazioni e ottimizzazione relativamente alle operazioni di scrittura dei dati e nella sua capacità di parallelizzare e gestire la concorrenza di molteplici sorgenti in scrittura in contemporanea. Inoltre è un database davvero semplice da scaricare, installare e utilizzare, anche grazie a una forte documentazione e community che ne tengono vivo lo sviluppo, aggiornamento e ne sostengono la diffusione.

InfluxDB è parte del TICK stack costituito anche dagli applicativi Telegraf, Chronograf e Kapacitor. 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 all'InfluxDB database a cui è connesso. Kapacitor è la componente che si occupa di post elaborare i dati memorizzati nel database time series, e di eseguire gli alert sulla base di soglie o comportamenti attesi, attingendo i dati dal database e inviando le notifiche via mail o applicativi terzi. Infine Chronograf è una applicazione web, tramite cui è possibile gestire tutti i componenti del tick

stack, accedere ai dati time series, aggiungere o rimuovere alerts, visualizzare in real time tramite dashboards e grafici i dati immagazzinati.

Usufruendo delle potenzialità offerte dai database time series, e in particolare da InfluxDB, molte aziende stanno aggiornando i propri sistemi, così da essere in grado di raccogliere informazioni e dati, da utilizzare in fase di monitoraggio e prevenzione sui guasti dei macchinari, e dei sistemi interni, sia per incrementare le proprie capacità predittive, sia per migliorare la qualità di utilizzo degli asset in azienda.

Fornendo l'accesso alla piattaforma tramite una interfaccia unificata, il TICK stack inizia anche ad essere utilizzato anche nel campo IOT, settore in cui si ha sempre più la necessità di elaborare una grande mole di dati sia in real-time che successivamente sulla base di campioni bufferizzati. Le fabbriche che intendono innovarsi devono prendere in considerazione anche questo aspetto!

3.6 Visualization layer

Il Visualization Layer, ricopre un ruolo fondamentale nella nostra applicazione per quanto riguarda la sottomissione di query al sistema e la conseguente visualizzazione dei risultati. L'obbiettivo di questo livello è permettere all'utente di effettuare valutazioni sia sui dati realtime che su dati aggregati eleborati in modalità batch.
Come strumento per la visualizzazione dei nostri risultati abbiamo utilizzato Grafana.
La scelta è ricaduto su questo strumento, per la grande espressività che presenta e l'enorme facilità che si ha nel suo utilizzo. Essendo uno strumento OpenSource ed avendo già una discreta platea di utilizzatori, sono disponibili numerosi plugin utili alla definizione di dashboard di ogni tipo.
E' anche possibile scrivere i propri plugin custom in numerosi linguaggi, favorendo ancor di più la sua estensione.
Per quanto riguarda la suddivisione vera e propria delle dashboards, ne abbiamo realizzate due differenti, intercambiabili attraverso un link.

Nella prima dashboard, mostrata in figura:

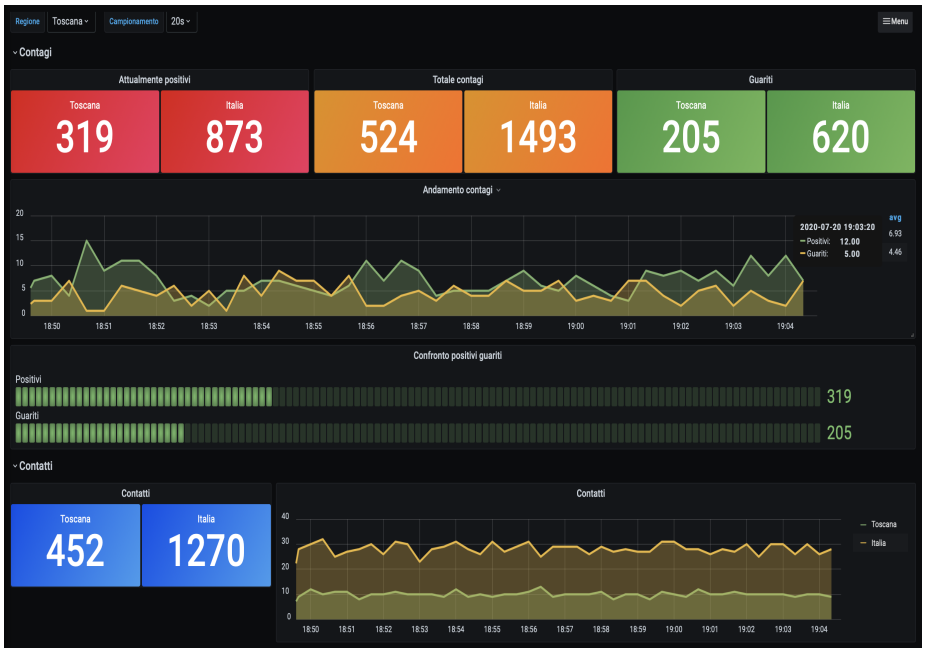

Figura 10: Vista prima dashboard

come si può notare, qui si è deciso di andare ad effettuare un'analisi regionale. Infatti in alto a sinistra della foto, è visibile un campo di selezione della regione di interesse (che ricordiamo sono: Toscana, Lombardia e Sicilia), con un'aggiunta di un campo che permette di poter andare a scegliere l'intervallo di

campionamento temporale utile per estrarre i dati.
Quello che si vuole mostrare con questa prima dashboard, sono sostanzialmente i contagi e i contatti per ogni regione per un determinato intervallo di campionamento temporale. Nella sezione relativa ai contagi, si è scelto di andare a studiare rispettivamente il numero degli *attualmente positivi*, *totale contagi*, *guariti* di ogni regione, a confronto con il totale relativo a tutta Italia. Inoltre si è scelto di effettuare un *confronto positivi guariti*, per permettere un'analisi relativa ad un'eventuale miglioramento del numero degli infetti da covid.
Nella sezione invece relativa ai contatti, si è invece deciso di andare a confrontare il numero dei contatti per regione, rispetto sempre al totale italiano; inoltre lo si è andato a graficare, in maniera tale da poter seguire i cambiamenti che avvengono real-time, al passare del tempo.
In alto a destra si può notare una voce "Menu", che è il link accennato in precedenza, che ci permette di passare alla seconda dashboard.
Diamo un'occhiata a come questa è strutturata:

Figura 11: Vista seconda dashboard

Lo scopo di questa dashboard, come si può facilmente capire, è quello di mettere a confronto tra di loro il numero dei guariti, contagiati e contatti, di ogni regione, in maniera tale da poter mettere a confronto tra di loro i vari trend regionali. Ovviamente anche in questo caso, la situazione può cambiare da un momento all'altro con il passare del tempo.
Finita l'analisi su questa dashboard, se si volesse tornare alla prima decsritta in precedenza, basta cliccare sull'apposito link "Home", in alto a destra.

3.6.1 Grafana

Grafana e' un software Open Source che consente di generare grafici e dashboard per il monitoraggio di ambienti e di sistemi. L'utente accede a Grafana con un normale browser: tipicamente il servizio risponde sulla porta 3000 [e' possibile modificare la porta ed utilizzare in alternativa i protocolli HTTP/HTTPS]. Grafana permette di interrogare, visualizzare, definire alert ed analizzare metriche con un'interfaccia semplice ed intuitiva. Grafana supporta diversi database temporali [TSDB: Time Series Database] quali Graphite, InfluxDB, ... e di recente anche diversi DB Relazionali quali MySQL e PostgreSQL.
Grafana e' un ottimo strumento per creare dashboard dinamiche su dati temporali: e' molto veloce, graficamente accattivante e di uso intuitivo. Una dashboard e' composta da pannelli, ciascuno dei quali esegue le ricerche su un database e visualizza i dati relativi. Le query vengono eseguite sull'intervallo temporale scelto dall'utente e, se richiesto, viene eseguito un refresh periodico. Le funzionalità di Grafana sono molte:
Dati dinamici: Grafana consente di analizzare i dati temporali in modo semplice ed intuitivo sia dal top menu che selezionando un intervallo temporale in un panel. Impostando un intervallo tutti i pannelli vengono aggiornati riportando i valori relativi al tempo scelto: le analisi temporali con Grafana sono particolarmente potenti e veloci. Oltre a questo e' possibile definire una o più variabili introducendo ulteriori filtri alle ricerche.
Grafica: Grafana supporta molteplici modalità di visualizzazione dei dati come ad esempio grafici a punti/linee/barre, gauge (singlestat), piechart, istogrammi, heatmap, geomap.
Profilazione: Grafana consente una profilazione sofisticata degli utenti all'interno di una o più Organization. Agli utenti possono essere associati Role (Admin, Editor, Viewer) per accedere con livelli differenti alle Dashboard ed ai Folder. E' possibile definire Team composti da un gruppo di utenti... insomma la profilazione e' molto completa. Oltre all'autenticazione nativa e' possibile utilizzare un'autenticazione LDAP esterna oppure utilizzare un web server come Apache in configurazione di reverse proxy e sfruttarne tutte le modalità di protezione. Collaborazione: Le dashboard ed i pannelli possono essere facilmente scambiati o condivisi tra gli utenti. Molto comoda e' anche la possibilità di annotare eventi inserendo commenti e tag.
Alert: Grafana permette, in modo semplice ed efficace, di definire soglie per l'invio di alert. Uno specifico Panel consente di visualizzare gli Alert attivi e quelli risolti.
Integrazione: fin dalle prime versioni Grafana ha integrato diversi database Time Series e strumenti per la collezione di dati (Graphite, Prometheus, InfluxDB). Nello stesso pannello possono essere visualizzati dati provenienti da sorgenti diverse in modo semplice e diretto. Dalle ultime versioni sono disponibili plugin per molteplici database relazionali e noSQL. Il backend di Grafana espone un'interfaccia HTTP API.
Estensioni: le funzionalita' Grafana si estendono facilmente con plugin esterni. I plugin possono fornire accesso a nuove basi dati, visualizzare i dati con

formati grafici differenti etc..
Open: da ultimo, ma non per importanza, Grafana e' un tool Open Source seguito e supportato da un'ampia community. Anche se vi sono alcune estensioni proprietarie (Premium Plugin) il loro numero e' molto basso e non limitano in nessun modo l'utilizzo dell'ambiente.

4 Conclusioni e sviluppi futuri

Il sistema nel complesso permettere di gestire ed analizzare i contagi e l'andamento dei positivi con granularità sia regionale che nazionale. Inoltre è facile scalare su altre regioni semplicemente inserendo all'interno del sistema nuovi produttori di eventi.
Per quanto riguarda le performance aumentando il numero di repliche dei nodi worker di Spark è possibile garantire velocità di elaborazione maggiori. La persistenza del DB essendo in modalità append only scala molto bene anche se il numero di richieste sale. Per quanto riguarda l'elaborazione e la visualizzazione lato utente finale tutte le elaborazioni attualmente vengono salvate su measurement differenti.
Alcuni miglioramenti o aggiunte che potrebbero essere effettuate in futuro al fine di incrementare le prestazioni e le funzionalità del sistema sono:

- Inserire una componente che effettua lo storage delle elaborazioni effettuate sottoforma di documenti al fine di poter offrire dei resoconti completi senza necessità di attendere troppo per la loro elaborazione.
  Questo potrebbere essere fatto mediante l'utilizzo di un documentDB come ad esempio MongoDb. Ogni qual volta viene generato un report questo potrebbe essere salvato persistentemente su db ed utilizzato in per le successive richieste limitando fortemente l'impiego di risorse per effettuare una nuova elaborazione.

- Utilizzare framework di ML al fine di fare predizione dei possibili contagi futuri oppure ricreare la rete di contatti anche in assenza di tutti i nodi. Questo potrebbe essere utile nel caso in cui alcuni utenti della rete non siano stati censiti e di conseguenza si potrebbero migliorare notevolmente le prestazioni in termini di accuratezza del sistema.

- Per quanto riguarda la parte relativa alla visualizzazione con Grafana, sarebbe interessante utilizzare un plugin chiamato Worldmap(la cui funzionalità è già stata inserita). Il pannello Worldmap è una mappa a riquadri del mondo che può essere sovrapposta a cerchi che rappresentano punti dati da una query. Può essere utilizzato con metriche di serie temporali, con dati geohash di Elasticsearch o dati nel formato tabellare.
  Grazie a questa aggiunta, si avrebbe quindi la possibilità di individuare punti relativi a persone, ad esempio contagiate, in una determinata area geografica.

Ecco un esempio di come la mappa appare quando viene utilizzata relativamente ai risultati delle query prodotte:

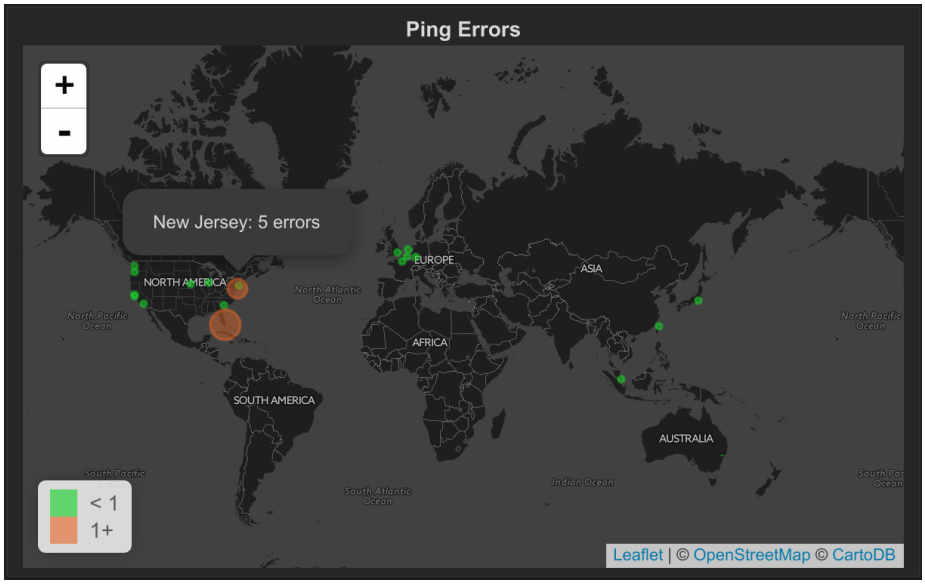

Figura 12: Esempio Worldmap

5 Riferimenti

- Documentazione Docker, `https://docs.docker.com`;
- Documentazione Docker compose, `https://docs.docker.com/compose/`;
- Documentazione Kafka, `https://kafka.apache.org/0110/documentation.html`;
- Documentazione Kafka-Python, `https://pypi.org/project/kafka-python/`;
- Documentazione ZooKeeper, `https://zookeeper.apache.org/documentation.html`;
- Documentazione Spark Streaming, `https://spark.apache.org/docs/latest/streaming-programming-guide.html`;
- Documentazione InfluxDB, `https://v2.docs.influxdata.com/v2.0/`;
- Documentazione Grafana, `https://grafana.com/docs/grafana/latest/`
- Mockaroo, `https://www.mockaroo.com`;