BIG DATA: SECONDO PROGETTO

Environment Analysis and Monitoring with Lambda Architecture

Anno Accademico 22/23

Gruppo BARONATO:
Andrea de Donato, Matr. 536795
Pietro Baroni, Matr. 536373
GitHub Repository: https://github.com/dedo99/second_project_baronato

Contents

1 Introduzione

1.1 Architettura lambda

L'architettura Lambda è un modello di elaborazione dei dati che consente di gestire grandi volumi di flussi di dati sia tempo reale sia batch. Questo tipo di architettura si rivela estremamente efficace per l'elaborazione, l'analisi e il monitoraggio dei dati di sensori ambientali. Con la crescente disponibilità di sensori intelligenti per monitorare parametri ambientali come temperatura, umidità, inquinamento dell'aria e altro ancora, l'architettura Lambda offre una soluzione robusta per gestire e trarre valore da questi dati. La struttura di base dell'architettura Lambda prevede due pipeline differenti: una dedicata al processamento streaming e l'altra dedicata alla analisi e processamento batch dei dati. Le due pipeline si affidano a due differenti servizi DB, ovvero un Batch Serving DB e un Realtime Serving DB. Per lo streaming ci si può affidare ad un servizio distribuito di messaggistica che consente di consumare ad una specifica velocità i dati che provengono da più sorgenti. Le sorgenti dati sono spesso una fonte di streaming come Apache Kafka, che non è la fonte di dati originale di per sé, ma è un archivio intermedio che può contenere dati e distribuirli ai vari consumer.

1.2 La nostra architettura

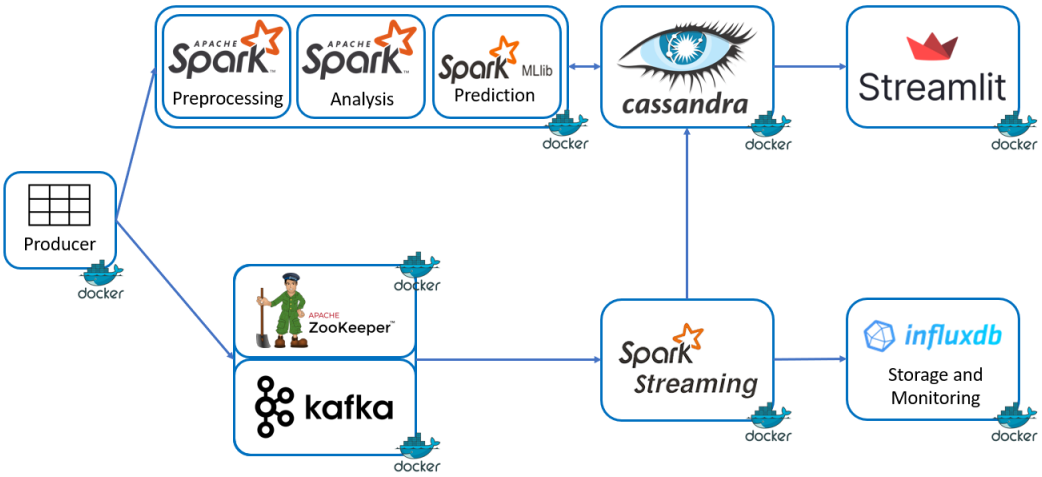

Figure 1: Architettura Lambda

L'architettura Lambda creata in questo progetto è mostrata in figura 1. Per favorire la portabilità e la riproducibilità di tale progetto, ogni componente dell'architettura ha un proprio ambiente isolato (container) in cui vengono soddisfatte tutte le dipendenze necessarie. Tutti i componenti comunicano tra di loro utilizzando una rete virtuale creata all'interno di Docker che consente di risolvere gli indirizzi IP anche attraverso i nomi che sono assegnati ai container e per la risoluzione degli indirizzi si ha che il Docker Engine si comporta come DNS. L'interazione con i componenti di questa architettura avviene attraverso il port-forwarding, che consente di mappare i servizi di determinate porte dei container sulle porte dell'host locale. Nell'architettura si possono distinguere 8 componenti:

- Producer
- Spark Batch
- Spark Streaming
- Zookeeper
- Kafka
- Cassandra
- InfluxDB
- Streamlit

Come si evince dalla figura dell'architettura le pipeline Batch e Streaming sono rispettivamente la parte superiore e la parte inferiore dell'architettura, ad eccezione del Producer che genera i dati per entrambe le pipeline. La descrizione accurata di ciascun componente dell'architettura viene rimandata ai capitoli successivi.

1.3 Tecnologie usate

Le principale tecnologie utilizzate in questo progetto, come visto nella figura 1, sono le seguenti:

- Spark è un framework open-source progettato per l'elaborazione distribuita di dati su larga scala, consentendo agli sviluppatori di scrivere applicazioni di analisi dei dati ad alte prestazioni. Spark può elaborare dati in batch e in tempo reale, fornendo un'ampia gamma di funzionalità per l'elaborazione, l'analisi e la manipolazione dei dati. Infatti, Spark comprende una libreria integrata che può acquisire e analizzare i dati in arrivo in tempo reale da diverse fonti, come log di server, social media, sensori, e altro ancora. Spark Streaming divide i flussi di dati in mini-batch, consentendo agli sviluppatori di applicare operazioni di trasformazione e analisi sui dati in tempo reale. Inoltre, Apache Spark contiene una libreria che offre un'ampia gamma di algoritmi e strumenti per la costruzione di modelli di machine learning su grandi set di dati. MLlib supporta algoritmi per la classificazione, la regressione, il clustering, il filtraggio collaborativo e molto altro ancora, semplificando lo sviluppo di modelli di machine learning scalabili.
- Cassandra è un sistema di gestione di database NoSQL distribuito ed altamente scalabile, progettato per gestire grandi volumi di dati su cluster di server. È noto per la sua capacità di fornire alta disponibilità e tolleranza ai guasti (le proprietà AP del CAP theorem), consentendo di gestire dati strutturati, semi-strutturati e non strutturati. Cassandra è ampiamente utilizzato per applicazioni che richiedono la memorizzazione e il recupero rapido di grandi quantità di dati. I suoi componenti principali sono Keyspace, Column Family, Partition, Row e Column. La consistenza è basata sul quorum e la sua architettura è Peer-to-Peer.
- Kafka è una piattaforma di streaming distribuito che permette la gestione e l'elaborazione di flussi di dati in tempo reale. È in grado di gestire milioni di messaggi al secondo,

garantendo una bassa latenza e una grande scalabilità. Kafka è spesso utilizzato come sistema di messaggistica per l'integrazione di sistemi distribuiti o come "backbone" per l'elaborazione di dati in tempo reale. Kafka mantiene feed di messaggi in categorie chiamate argomenti (topic). Il Producers produce messaggi (record) di uno o più argomenti (topic). Il Consumer sottoscrive ai topic ed elabora il feed dei messaggi pubblicati. Un topic può avere zero, uno o più consumatori che sottoscrivono i dati scritti su di esso. Per ogni topic, il cluster Kafka mantiene un log partizionato. A ciascuno dei record nelle partizioni viene assegnato un numero ID sequenziale chiamato offset che identifica in modo univoco ogni messaggio all'interno della partizione. Ogni nuovo record è appeso ad una sola partizione.

- ZooKeeper è un servizio di coordinamento distribuito che fornisce funzionalità di gestione e sincronizzazione per applicazioni distribuite. Può essere utilizzato per la gestione delle configurazioni, la scoperta dei servizi, la sincronizzazione dei nodi e altro ancora. ZooKeeper offre un'architettura scalabile e affidabile, consentendo alle applicazioni distribuite di coordinarsi e operare in modo affidabile.

- InfluxDB è un database di serie temporali ottimizzato per l'archiviazione e il recupero di dati temporali. È particolarmente adatto per l'archiviazione di dati generati da sensori, log di eventi e metriche di monitoraggio. InfluxDB offre una query linguaggio specifico per le serie temporali (Flux) ed è integrato con strumenti di visualizzazione dei dati.

- Streamlit è un framework open-source per la creazione di applicazioni web interattive per la visualizzazione e l'analisi dei dati. Consente agli sviluppatori di creare interfacce utente interattive e reattive utilizzando Python in modo tale da semplificare la creazione di applicazioni web per la presentazione e l'esplorazione dei dati.

- Docker è una piattaforma open-source per l'automazione della distribuzione di applicazioni all'interno di container. I container Docker consentono di incapsulare l'applicazione e le sue dipendenze in un'unità isolata, garantendo la portabilità e la riproducibilità dell'ambiente di esecuzione. Docker semplifica la distribuzione delle applicazioni su diversi ambienti di esecuzione, fornendo un metodo standardizzato e affidabile.

1.4 Il dataset

Il dataset utilizzato per questo progetto consiste in dati ambientali generati, una volta ogni minuto, da un'abitazione con una buona quantità di sensori IOT. Il dataset è stato ottenuto da Kaggle [1]. I sensori IOT installati all'interno dell'abitazione consentono di ottenere 32 misure (attributi) distinte:

- time: tempo espresso in Unix timestamp
- use [kW]: energia totale consumata
- gen [kW]: energia totale generata

[1] https://www.kaggle.com/datasets/taranvee/smart-home-dataset-with-weather-information

- House overall [kW]
- Dishwasher [kW]
- Furnace 1 [kW]
- Furnace 2 [kW]
- Home office [kW]
- Fridge [kW]
- Wine cellar [kW]
- Garage door [kW]
- Kitchen 12 [kW]
- Kitchen 14 [kW]
- Kitchen 38 [kW]
- Barn [kW]
- Well [kW]
- Microwave [kW]
- Living room [kW]
- Solar [kW]
- temperature
- icon: condizioni meteo complessive
- humidity
- visibility
- summary
- apparentTemperature: temperatura apparente (percepita)
- pressure
- windSpeed: velocità del vento
- cloudCover: copertura delle nuvole
- windBearing
- precipIntensity
- dewPoint: punto di rugiada
- precipProbability: probabilità di precipitazioni

In questo dataset sono presenti 29 attributi con variabili quantitative, mentre solo 3 attributi hanno variabili ordinali o nominali. Il dataset grezzo necessita di diverse operazioni per poter essere utilizzato tra cui la conversione del timestamp Unix in formato data, trasformazione dei numeri in modo da rimuovere la notazione scientifica e altre operazioni che sono descritte nei capitoli successivi.

2 Streaming Layer

2.1 Ingestion

Nella pipeline di streaming dell'architettura Lambda, mostrata in figura 2, la fase di ingestion è essenziale per acquisire e inviare i dati in tempo reale. In questo caso, il producer è responsabile di inviare i dati uno alla volta attraverso Kafka. I dati stessi provengono da un dataset in formato CSV.

In primo luogo, il Producer estrae i dati dal dataset CSV, prendendo un record alla volta. Ogni record viene quindi inviato al topic 'my-topic' all'interno di Kafka. Questo topic funge da canale di comunicazione tra il Producer e i consumatori successivi nella pipeline di elaborazione dei dati.

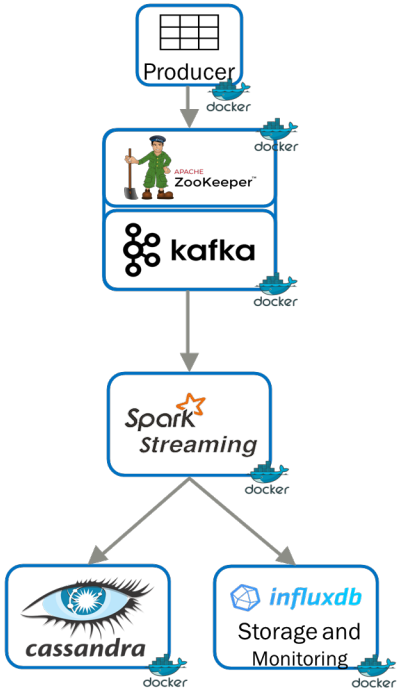

Figure 2: Pipeline streaming.

L'utilizzo di Kafka come intermediario per il flusso di dati consente una gestione efficiente e scalabile. I dati inviati al topic 'my-topic' sono pronti per essere consumati e elaborati da uno o più consumatori, che possono essere configurati per operare in parallelo.

Attraverso questa configurazione, l'architettura Lambda garantisce che i dati vengano acquisiti, trasmessi e resi disponibili per il consumo in tempo reale. La distribuzione uno alla volta dei dati dal producer al topic Kafka 'my-topic' consente un flusso continuo e una gestione affidabile dei dati all'interno della pipeline di streaming.

Uno dei vantaggi principali dell'utilizzo del sistema distribuito di messaging Kafka è che consente di conservare tutti i record pubblicati, indipendentemente dal fatto che siano stati consumati o meno, per un periodo di tempo configurabile perciò sia in caso di crash del Consumer o anche in caso di aggiunta di un nuovo Consumer non sarà necessario richiedere l'intervento del Producer per ottenere tali dati.

2.2 Spark Streaming

Il cuore della pipeline Streaming è il componente Spark Streaming ed è progettato per elaborare in tempo reale i dati provenienti dalla fase di ingestion. Utilizzando Spark, i messaggi vengono letti in streaming da Kafka e sottoposti a una serie di trasformazioni per estrarre e formattare i dati desiderati. Successivamente, viene eseguito un processo di preprocessing, che include la conversione delle temperature da Fahrenheit a Celsius e la formattazione dei numeri.

Una volta che i dati sono stati elaborati, vengono gestiti in modo diverso a seconda dell'obiettivo. Viene avviato uno stream di output che visualizza i dati sulla console per il monitoraggio in tempo reale. Inoltre, i dati vengono scritti sia su Cassandra, per l'archiviazione persistente, sia su InfluxDB, per la persistenza e la visualizzazione dei dati di serie temporali.

Per scrivere i dati su Cassandra, viene utilizzato il modulo Spark Cassandra Connector, che permette di specificare il keyspace e la tabella di destinazione. I dati vengono scritti in modo incrementale e mantenuti coerenti con gli aggiornamenti in arrivo. Per permettere la corretta connessione al database Cassandra è necessario specificarne tutti i parametri di connessione nel momento in cui si istanzia la SparkSession. I parametri che vengono specificati per la connessione a Cassandra sono:

- Host
- Porta
- Username
- Password

Per quanto riguarda InfluxDB, viene utilizzata la libreria influxdb-client-python per la connessione e la scrittura dei dati. Viene definita una funzione personalizzata "write_to_influxdb" che crea un oggetto di tipo Point per ogni record e lo scrive sul bucket di InfluxDB. L'oggetto Point costituisce un'astrazione della struttura dati necessaria per l'inserimento dei dati in Influx. Ci sono due tipi di dati offerti da Point che sono stati utilizzati: time e field. Time consente di assegnare un valore alla variabile tempo su cui si basa la memorizzazione di influxDB e qualora non viene assegnato nessun valore a tale variabile viene inserito automaticamente l'orario e la data attuale del momento dell'inserimento dei dati, ma nel nostro caso viene assegnato come valore time il momento di rilevazione del dato e non il momento attuale. Field consente di inserire i valori di un qualsiasi attributo e, infatti, sono stati inseriti tutti gli attributi del dataset ad eccezione di time.

Complessivamente, questa pipeline di Spark Streaming offre un'elaborazione dei dati in tempo reale efficiente e scalabile. Grazie alla sua capacità di gestire flussi continui di dati da Kafka, di eseguire trasformazioni complesse e di archiviare i risultati in database affidabili come Cassandra e InfluxDB, la pipeline supporta una varietà di casi d'uso, inclusi il monitoraggio dei dati, l'analisi e l'elaborazione dei dati di serie temporali.

2.3 Storage e Monitoring

La memorizzazione dei dati coinvolti nella pipeline Streaming avviene su due database NoSQL differenti: InfluxDB e Cassandra. Il database principale per la pipeline streaming è

InfluxDB poichè è un DB estremamente adatto per il salvataggio di serie temporali, ma i dati vengono salvati anche su Cassandra per consentire di arricchire la fase di analisi batch. InfluxDB consente di salvare i dati come record, dove all'interno di ogni record le componenti fondamentali del dato sono il tempo in cui è stato generato tale dato e il dato stesso (ad esempio temperatura o pressione o ...). Il linguaggio utilizzato per interrogare i dati è Flux.

La seguente query in Flux si occupa di mostrare tutti i dati presenti all'interno del bucket "iothome_bucket" e presenti all'interno del measurement "iothome_data". Inoltre, vengono mostrati solo i dati relativi alla temperatura e rilevati nel periodo temporale tra 01/01/2016 e 10/01/2016.

```
from(bucket: "iothome_bucket")
  |> range(start: 2016-01-01T00:00:00Z, stop: 2016-01-10T00:00:00Z)
  |> filter(fn: (r) => r._measurement == "iothome_data")
  |> filter(fn: (r) => r._field == "temperature")
```

Il bucket e il measurement possono essere ricondotti rispettivamente a quello che nei database relazionali sono lo schema (gruppo di tabelle) e la tabella.

La nuova versione di InfluxDB, ovvero la v 2.7.1, consente di creare delle dashboard pensate proprio per il monitoraggio in tempo reale dei dati. Ogni tipo di grafico che costituisce la dashboard si ottene a seguito di un'interrogazione effettuata sul database con Flux. Un esempio di dashboard ottenuta per il monitoraggio dei dati ambientali in tempo reale è i mostrato in figura 3.

Figure 3: Dashboard per il monitoraggio real time dei dati ambientali.

La dashboard mostrata consente di visualizzare tutti i dati di maggior interesse per verificare istantaneamente situazioni anomale, individuare trend e prevedere costi legati al consumo.

3 Batch Layer

3.1 Pre-processing

Le prime operazioni svolte nella pipeline di analisi batch, mostrata in figura 4, sono quelle di pre-processamento dei dati eseguite mediante Apache Spark, tali operazioni sono le stesse che vengono eseguite nella pipeline di streaming come la conversione del timestamp Unix in data, la conversione dei gradi Fahrenheit in gradi Celsius, la conversione dei valori numerici in notazione scientifica in valori in virgola mobile e l'eliminazione di alcune colonne quali 'icon' e 'cloudCover'. I dati pre-processati vengono poi salvati in una tabella del database NoSQL Cassandra come precedentemente fatto nella pipeline di streaming.

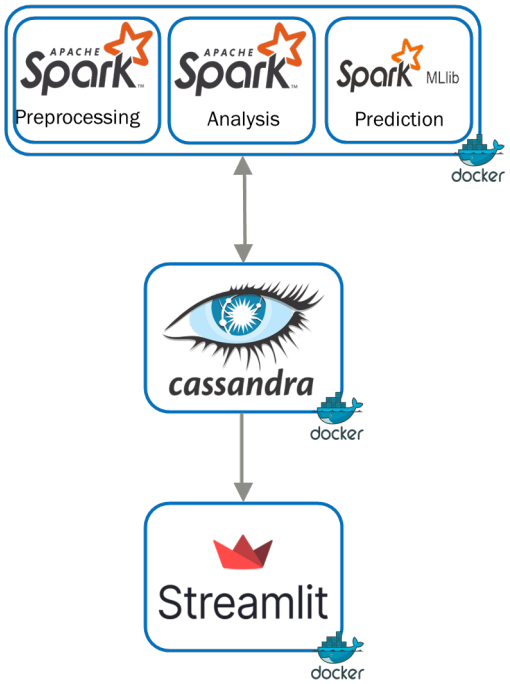

Figure 4: Pipeline Batch.

3.2 Batch Analysis

Una volta pre-processati i dati vengono recuperati dalla tabella di Cassandra e vengono eseguite le operazioni di analisi. Le analisi eseguite, sempre mediante Apache Spark, sono di due tipi e sono effettuate rispetto a quattro granularità temporali diverse quali:

- Totale;
- Mese;
- Giorno della settimana;
- Giorno dell'anno.

La prima analisi eseguita è relativa ai consumi energetici presenti nel dataset e relativi all'abitazione. Un esempio di analisi dei consumi effettuata rispetto al giorno della settimana è mostrato nel seguente codice:

```
kw_dayofweek_df = spark.sql("""
    SELECT DAYOFWEEK(time) AS day_of_week,
            SUM(use_kw) AS use_kw_total,
            SUM(gen_kw) AS gen_kw_total,
            AVG(use_kw) AS use_kw_avg,
            AVG(gen_kw) AS gen_kw_avg,
            MAX(use_kw) AS use_kw_max,
            MAX(gen_kw) AS gen_kw_max,
            MIN(use_kw) AS use_kw_min,
            MIN(gen_kw) AS gen_kw_min
    FROM df
    GROUP BY DAYOFWEEK(time)""")
```

La seconda è relativa ai dati atmosferici presenti nel dataset. Un esempio di analisi dei dati atmosferici effettuata rispetto al giorno dell'anno è mostrato nel seguente codice:

```
weather_day_df = spark.sql("""
    SELECT DAYOFYEAR(time) AS day,
           AVG(temperature) AS avgtemp,
           MIN(temperature) AS mintemp,
           MAX(temperature) AS maxtemp,
           AVG(humidity) AS avghum,
           MIN(humidity) AS minhum,
           MAX(humidity) AS maxhum,
           AVG(visibility) AS avgvis,
           MIN(visibility) AS minvis,
           MAX(visibility) AS maxvis,
           AVG(apparent_temperature) AS avgapptemp,
           MIN(apparent_temperature) AS minapptemp,
           MAX(apparent_temperature) AS maxapptemp,
           AVG(pressure) AS avgpress,
           MIN(pressure) AS minpress,
           MAX(pressure) AS maxpress,
           AVG(wind_speed) AS avgwspeed,
           MIN(wind_speed) AS minwspeed,
           MAX(wind_speed) AS maxwspeed,
           AVG(wind_bearing) AS avgwbear,
           MIN(wind_bearing) AS minwbear,
           MAX(wind_bearing) AS maxwbear,
           AVG(precip_intensity) AS avgprec,
           MIN(precip_intensity) AS minprec,
           MAX(precip_intensity) AS maxprec,
           AVG(dew_point) AS avgdpoint,
           MIN(dew_point) AS mindpoint,
           MAX(dew_point) AS maxdpoint
    FROM df
```

```
GROUP BY DAYOFYEAR(time)""")
```

Oltre a ciò nello script di analisi batch viene eseguita la preparazione del dataset per la predizione che verrà approfondita successivamente.
Tutte le tabelle generate dall'analisi vengono poi salvate nel database Cassandra.

3.3 Visualizzazione

Per la visualizzazione dei dati è stato scelto il framework Streamlit che consente una maggiore interattività per l'utente. Nel nostro caso l'utente può scegliere quale dataset vuole visualizzare, se quello completo, quelli relativi ai consumi energetici o quelli relativi ai dati atmosferici. Per ogni dataset scelto poi viene mostrato a schermo per ogni attributo un diverso grafico:

- Grafico di distribuzione: usato per mostrare la distribuzione dei valori di ogni attributo del dataset completo;
- Istogramma: usato nei casi di granularità mensile o del giorno della settimana per mostrare i valori medi, massimi e minimi degli attributi;
- Grafico a linea: usato nel caso di granularità giornaliera per mostrare la variazione dei valori medi di ogni attributo comparata con i valori massimi e minimi.

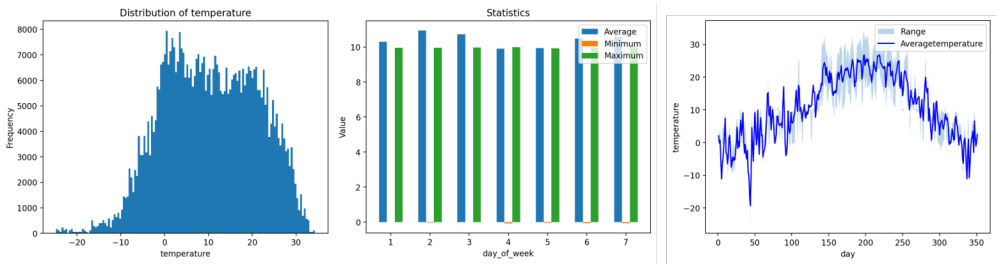

Figure 5: Esempio di grafici presenti

4 MLlib

L'interesse sempre più crescente per l'efficienza energetica e la gestione delle spese domestiche spinge molti utenti a cercare soluzioni smart per ottimizzare i consumi energetici all'interno delle proprie abitazioni. In questo contesto, MLlib di Apache Spark, libreria nata per utilizzare modelli di Machine Learning su grandi moli di dati, si rivela un ottimo strumento per effettuare predizioni accurate sulla temperatura e sui consumi di kW all'interno dell'abitazione. Grazie alla combinazione del lavoro di pre-processing e analisi svolto dall'architettura Lambda e ai potenti strumenti forniti da MLlib, è possibile creare modelli di Machine Learning in grado di analizzare i dati di temperatura, consumi di kW e altri fattori correlati al fine di generare predizioni accurate sulle tendenze future. Queste predizioni possono essere utilizzate per ottimizzare il consumo energetico, pianificare l'uso degli elettrodomestici e stimare le spese future relative ai consumi domestici.

In questo caso la predizione avviene su due attributi: la temperatura ed il consumo di kilowatt. Per avere una predizione il più accurata possibile è stato scelto di calcolare le medie di tutti gli attributi raggruppati per ogni ora in modo tale da avere la predizione della media oraria dei valori obiettivo.
Il dataset è quindi composto da circa 8400 righe e da 14 colonne: una per indicare l'ora, una per la data e le restanti per gli attributi del dataset originale tranne quelli dei consumi dei singoli elettrodomestici e degli attributi testuali.
Per eseguire la predizione è stata usata la libreria MLlib di Apache Spark, più precisamente l'algoritmo di Regressione Lineare.
Per la valutazione dei risultati invece sono state usate due metriche:

- Errore Quadratico Medio:

$$RMSE = \sqrt{\frac{1}{nT}\sum_{i,t}\left(\hat{y_{i,t}} - y_{i,t}\right)^2}$$

- R-squared:

$$R^2 = 1 - \frac{\sum\left(y_i - \hat{y_i}\right)^2}{\sum\left(y_i - \bar{y_i}\right)^2}$$

4.1 Weather prediction

I risultati per la predizione della temperatura sono i seguenti:

	RMSE	R^2
Train Set	1.078683	0.996807
Test Set	1.0614	0.996914

Come possiamo notare dai risultati e dal grafico in figura 6 la differenza media tra la predizione ed il valore reale è di circa un grado centigrado.

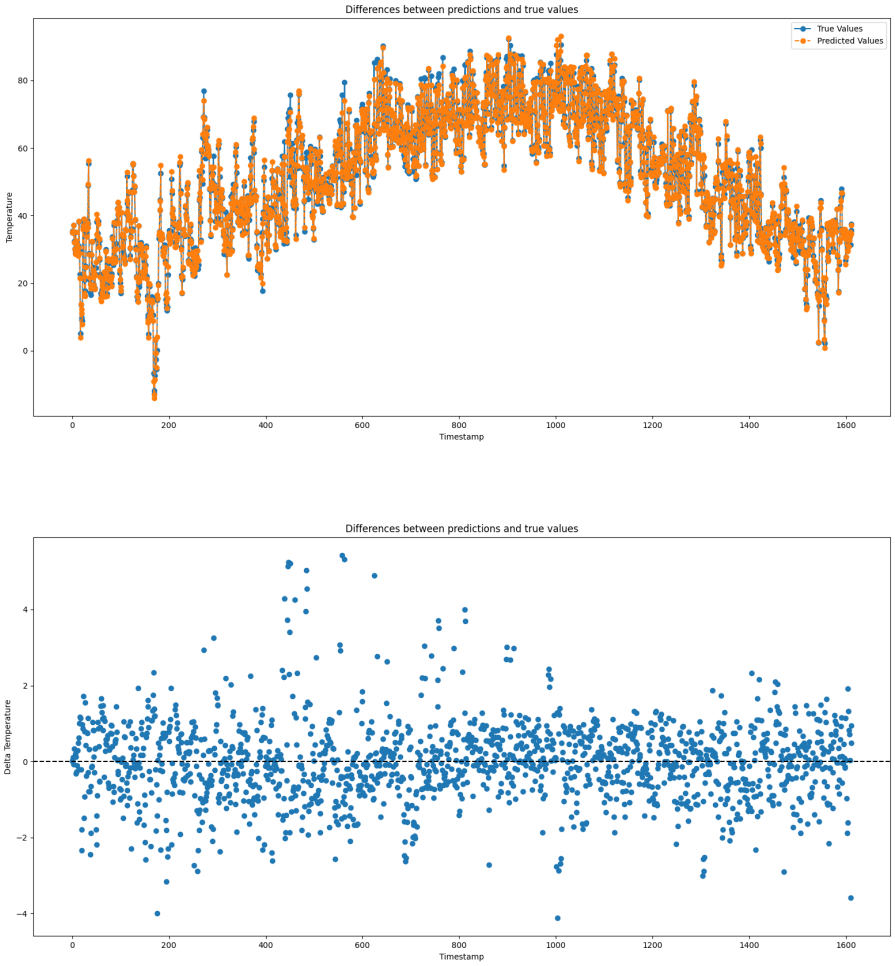

Figure 6: Differenza tra valore predetto e valore reale

4.2 KW usage prediction

I risultati per la predizione del consumo energetico orario sono i seguenti:

	RMSE	R^2
Train Set	0.000000	1.000000
Test Set	7.25604e-15	1

Come possiamo notare dai risultati e dalla scala del grafico in figura 7 la differenza media tra la predizione ed il valore reale è molto bassa, questo ci fa pensare alla presenza di overfitting oppure alla mancanza di features adeguate alla predizione di quel valore.

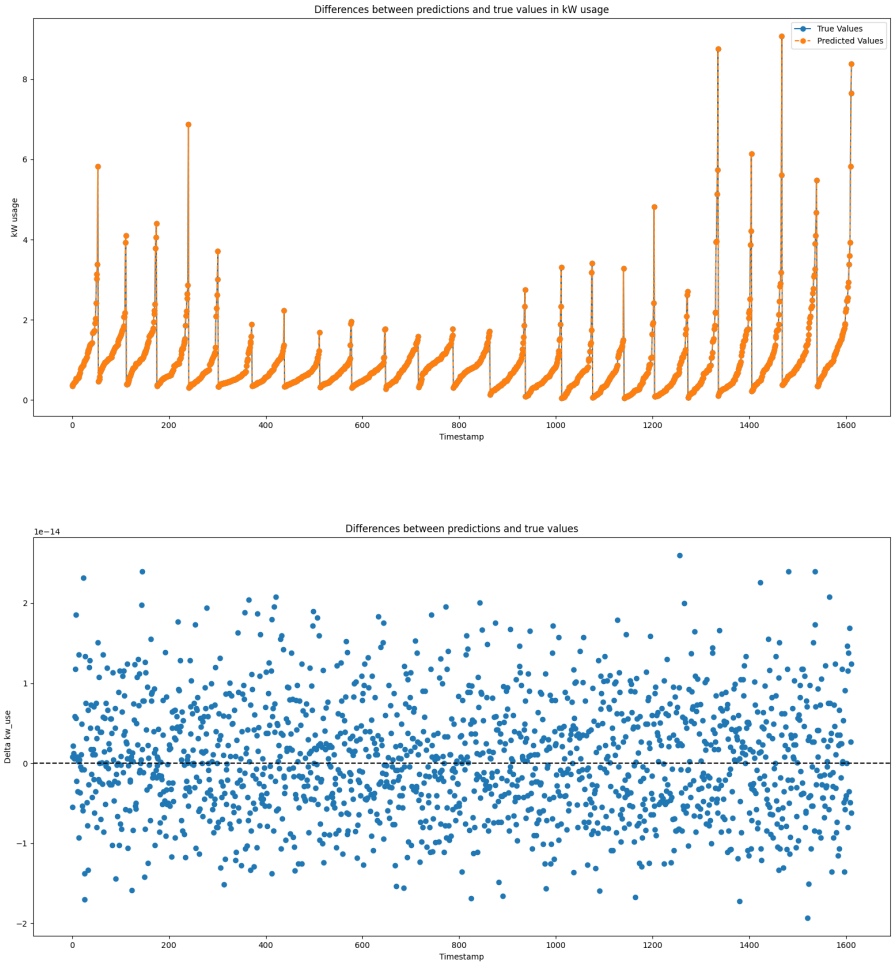

Figure 7: Differenza tra valore predetto e valore reale

5 Considerazioni finali

Durante lo sviluppo dell'architettura Lambda basata su Docker, con l'integrazione dei componenti Spark, Cassandra, InfluxDB, Kafka, ZooKeeper, Streamlit, MLlib di Spark e Spark Streaming, sono emerse diverse considerazioni importanti. Queste considerazioni riguardano sia gli aspetti positivi che le sfide affrontate durante l'implementazione e l'interconnessione dei diversi componenti. L'adozione di un'architettura Lambda offre numerosi vantaggi. La separazione chiara tra l'elaborazione batch e in tempo reale permette di gestire i dati in modo efficiente.

Docker ha semplificato notevolmente la distribuzione e l'interconnessione dei vari componenti dell'architettura e ha consentito di creare container indipendenti per ciascun componente, garantendo l'isolamento e la portabilità delle applicazioni. Tuttavia, interconnettere tutti i componenti in un ambiente Docker può risultare complesso. Infatti, la configurazione delle reti e dei collegamenti tra i container richiede una pianificazione accurata ed è fondamentale anche una corretta configurazione delle variabili d'ambiente, delle porte di rete e delle autorizzazioni differenti in base alla natura dello specifico servizio. Inoltre, sono state riscontrare difficoltà nel gestire tutti i componenti, ciascuno con un container differente, originati da immagini distinte, per motivi di limitata quantità sia di RAM che di storage.

La gestione dei dati è stata resa possibile grazie all'utilizzo di Cassandra e InfluxDB come database di archiviazione. Questi database hanno offerto la flessibilità necessaria per gestire sia i dati batch che quelli in tempo reale. Tuttavia, è importante definire una strategia di archiviazione dei dati ben strutturata, considerando aspetti come la scalabilità, la consistenza e la tolleranza alle partizioni per garantire prestazioni ottimali.

L'integrazione di Streamlit ha reso possibile la creazione di un'interfaccia utente interattiva per visualizzare e analizzare i dati. Streamlit ha semplificato lo sviluppo di applicazioni web per l'analisi e la visualizzazione dei dati, consentendo agli utenti di esplorare e comprendere meglio i risultati delle analisi sui dati storici.

Nonostante le sfide affrontate durante l'implementazione e l'interconnessione dei componenti, l'adozione di un'architettura Lambda con Docker, Spark, Cassandra, InfluxDB, Kafka, ZooKeeper, Streamlit, MLlib di Spark e Spark Streaming ha dimostrato di essere un approccio valido per la gestione delle spese e per promuovere uno stile di vita più sostenibile. Questa soluzione offre una visione globale dei dati energetici, supporta decisioni consapevoli e promuove l'efficienza energetica. Con una pianificazione attenta e un'adeguata configurazione, i benefici derivanti da questa architettura superano di gran lunga le difficoltà incontrate nel suo sviluppo e implementazione.