Progetto Finale Big Data
Air Quality Lambda Architecture

Matteo de Santis (500355) e Kaio Cesar Alves Reis (484997)

Sommario

1 Introduzione e obiettivi

Il progetto ha come obiettivo la realizzazione di un'architettura lambda per l'analisi streaming e l'analisi batch di dati provenienti da stazioni di monitoraggio della qualità dell'aria.
Tali dati sono relativi a stazioni di monitoraggio situate in diverse città degli Stati Uniti, che riportano ogni giorno informazioni riguardanti la concentrazione di diversi inquinanti (Biossido di Azoto, Biossido di Zolfo, Monossido di Carbonio e Ozono) nell'aria.

Un altro obiettivo è quello di sperimentare gli strumenti e le tecnologie che rappresentano lo stato dell'arte in ambito di Big Data Analytics, e quindi l'integrazione tra queste diverse tecnologie.

2 Architettura e tecnologie utilizzate

La Lambda Architecture è un'architettura progettata per gestire grossi flussi di dati, combinando una tradizionale pipeline batch e una pipeline di stream in real-time. Questo approccio permette di bilanciare la latenza, il throughput e la tolleranza ai guasti fornendo attraverso la parte batch viste dettagliate delle grosse quantità di dati e contemporaneamente la parte streaming permette di avere risposte alle query con bassa latenza.

I dati vengono forniti ai layer batch e streaming attraverso l'ingestion layer, che dev'essere costituito da un sistema che permette l'inoltro di grosse quantità di dati in streaming anche in modo asincrono.

Le viste batch e i dati streaming analizzati vengono infine memorizzati in un sistema DBMS e visualizzate in una dashboard.

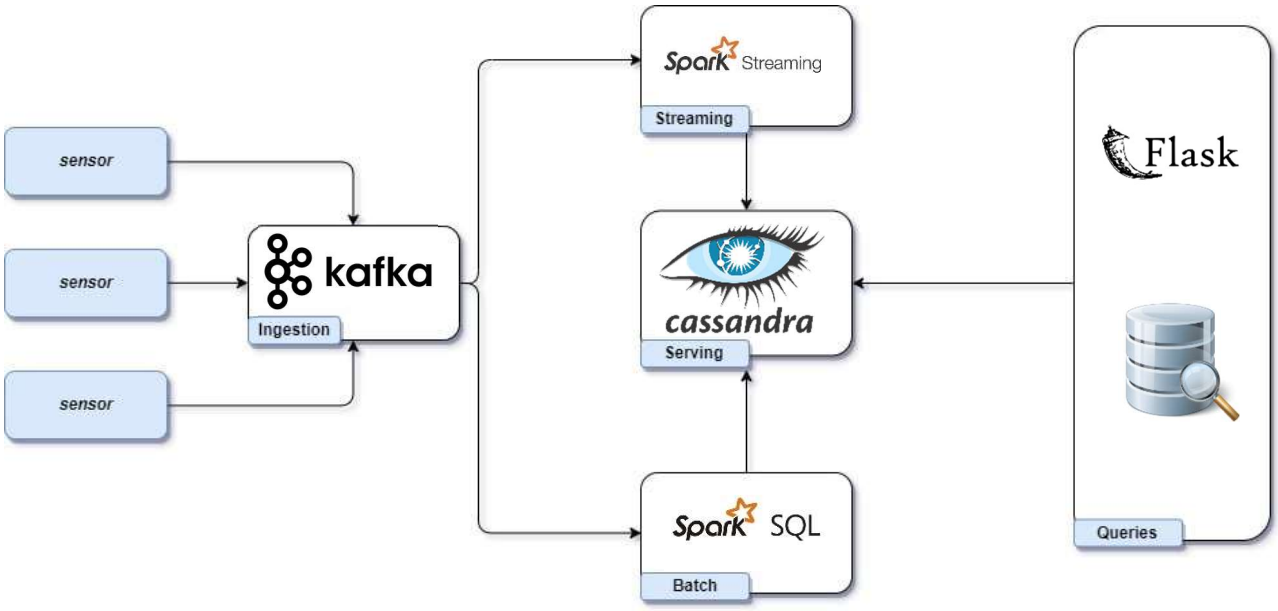


Figura 1: Architettura e scelte tecnologiche del progetto

2.1 Apache Kafka

Per l'ingestion dei dati viene utilizzato Apache Kafka, una piattaforma open-source per il data streaming basata sul concetto di publishing e subscribing, che permette di archiviare ed elaborare flussi di record in tempo reale. In breve, consente di spostare grandi quantità di dati da un punto qualsiasi a un altro nello stesso momento.

Uno dei vantaggi di Kafka è che permette di disaccoppiare chi produce i dati da chi li legge e quindi la possibilità di inviare/leggere i dati in maniera totalmente asincrona, garantendo alte prestazioni, in quanto non avverranno mai situazioni di collo di bottiglia.

I concetti fondamentali in Kafka sono:

- Topic: astrazione indirizzabile usata per mostrare interesse in un dato flusso di dati. Un topic può essere pubblicato e sottoscritto, ed è un livello di astrazione che viene utilizzato dall'applicazione per mostrare interesse in un dato flusso di dati.

- Persistenza: Kafka gestisce un cluster di server che mantiene in modo duraturo i record mentre vengono pubblicati. Il cluster usa un timeout di mantenimento configurabile per determinare il tempo di vita dei messaggi indipendentemente dal consumo.

- Producers: un produttore di dati definisce su quale topic un dato record/messaggio dovrebbe essere pubblicato.

- Consumatori: sono le entità che elaborano i record/messaggi dopo essersi sottoscritte a uno topic o più topic. I consumatori possono essere configurati per lavorare indipendentemente su carichi di lavoro individuali o in modo collaborativo con altri consumatori su un dato carico di lavoro (bilanciamento del carico).

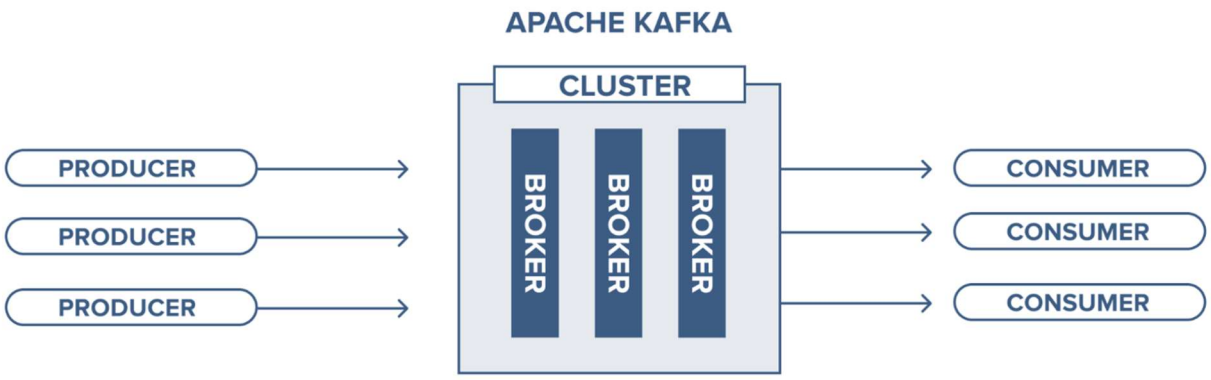


Figura 2: Architettura Kafka

2.2 Apache Spark

Come stream e batch engine è stato utilizzato Apache Spark, un framework di elaborazione parallela open source per l'esecuzione di applicazioni di analisi di dati su larga scala.

Spark è progettato specificamente per fornire velocità di calcolo e scalabilità per il flusso di dati, dati grafici, apprendimento automatico e applicazioni di intelligenza artificiale.

Spark elabora i dati da vari repository come HDFS, database NOSQL e Apache Hive. Inoltre, supporta l'elaborazione in memoria per migliorare le prestazioni delle applicazioni di analisi dei dati. Tuttavia, può anche eseguire l'elaborazione basata su disco quando i set di dati sono troppo grandi per adattarsi alla memoria di sistema disponibile.

Spark ha un'architettura gerarchica. Il driver Spark è il nodo master che gestisce i nodi in esecuzione e controlla l'amministratore del cluster.

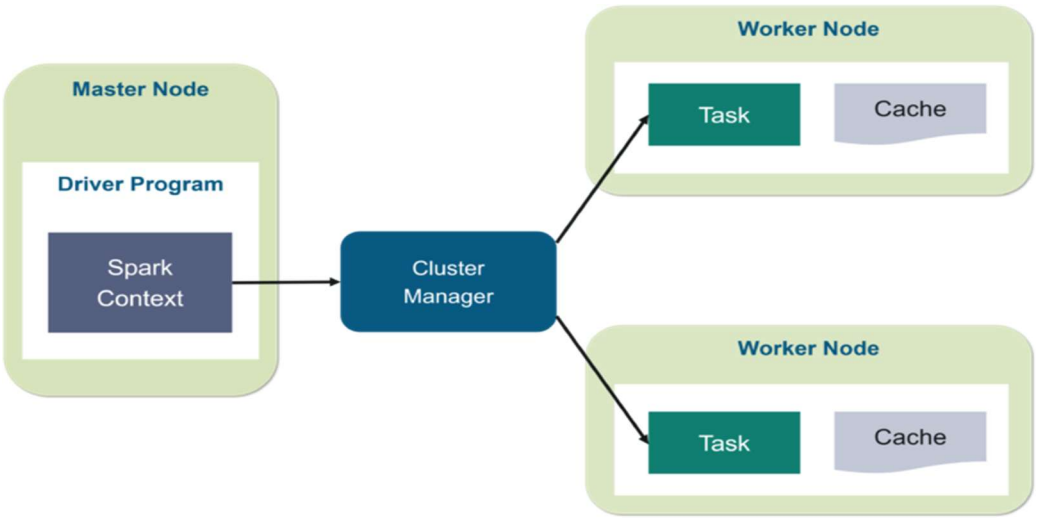


Figura 3: Architettura gerarchica di Spark

Spark utilizza un modello di struttura di dato distribuito e flessibili chiamati Resilient Distributed Dataset.

I Resilient Distributed Dataset hanno una struttura progettata per nascondere la complessità computazionale agli utenti. Spark raccoglie i dati e li sposta in un archivio dati diverso dove possono essere calcolati in seguito. Quindi lo divide in un cluster di server che può essere eseguito tramite modello analitico.

Tra i componenti fondamentali di Spark, nel progetto vengono utilizzati: Spark SQL e Spark Streaming.

- Spark SQL – una delle librerie più utilizzate, Spark SQL offre un'interfaccia SQL a Spark, consentendo di eseguire query sui dati tramite SQL oppure in HQL, la variante SQL di Apache Hive.

- Spark Streaming – è un componente di Spark che consente l'elaborazione e la presentazione di flussi dati in tempo reale.

2.3 Apache Cassandra

Per lo storage dei dati viene utilizzato Cassandra, un database NoSQL open source ottimizzato per la gestione di grandi flussi di dati, e che offre un modello di dati basato sul modello ColumnFamily.

Grazie alla buona scalabilità, questi database possono essere distribuiti su diversi cluster, il che significa che Cassandra non è legato a un singolo server. Usando quindi un'architettura peer-to-peer rispetto alla configurazione master/slave utilizzata nei RDBMS, Cassandra ha la capacità di sopravvivere a intere interruzioni del data center. Tutti i cluster hanno i medesimi diritti e possono elaborare tutte le interrogazioni del database, cosa che aumenta significativamente l'efficienza del sistema.

Offre inoltre un'elevata disponibilità e tolleranza alle partizioni, secondo il Teorema CAP esso è un sistema AP.

Apache Cassandra dispone di un proprio linguaggio di interrogazione denominato Cassandra Query Language (CQL) che, seppur simile a SQL, viene preferito dagli sviluppatori in ragione del fatto che è adattato alle caratteristiche di Cassandra.

2.4 Flask

La dashboard del progetto è una Web Application sviluppata utilizzando le funzionalità offerte da Flask.

Flask è un framework web, che fornisce strumenti, librerie e tecnologie che permettono la creazione di applicazioni web. Fa parte della categoria dei micro-framework, che comprende tutti quei framework che hanno ridotte dipendenze a librerie esterne. I vantaggi di avere ridotte dipendenze é di avere framework estremamente leggeri e di facile utilizzo, mentre i contro è che se si necessità di creare applicazioni più complesse, tutto il lavoro dev'essere fatto "a mano".

Flask è basato sullo strumento Werzeug Web Server Gateway Interface (WSGI), uno standard per lo sviluppo di applicazioni web Python. WSGI quindi è una specifica di interfaccia universale tra il web server e le web app. Werzeug è uno

dei più avanzati moduli WSGI che contiene diversi tools e utilities che facilitano lo sviluppo di web app Python.

Infine, Flask utilizza come template engine Jinja2, che permette la creazione di file HTML, XML, o altri formati di markup, che vengono restituiti all'utente mediante una risposta HTTP. Presenta una sintassi consistente e, in aggiunta, rispetto ad altri motori di template, permette da un lato di inserire il codice all'interno dei template, dall'altro di essere sviluppato.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Figura 4. Semplice esempio di app Flask

3 Analisi dei dati

Il dataset originale di proprietà dell'U.S. Enviromental Protection Agency è stato pre-elaborato da un utente di Kaggle in modo da facilitare le operazioni di analisi.
La versione utilizzata per lo svolgimento del progetto si chiama U.S. Pollution Data e contiene dati relativi ai livelli di inquinamento in diverse città degli Stati Uniti. In particolare, ogni ennupla mostra la concentrazione giornaliera dal 2000 al 2016, dei quattro maggiori inquinanti dell'aria, per città:

- Biossido di Azoto (NO2)
- Biossido di Zolfo (SO2)
- Monossido di Carbonio (CO)
- Ozono (O3)

Il dataset contiene all'incirca +2M di ennuple ed è strutturato in 29 campi:

- Attributi di posizione della stazione di monitoraggio
- Attributo temporale
- Ogni inquinante ha cinque campi specifici

Di seguito una breve descrizione di ciascun campo che riguarda gli inquinanti:

- Unità di misura: unità di misura usata per esprimere i livelli di concentrazione di un elemento chimico (parti per milione o parti per miliardo)
- Media giornaliera: media giornaliera del dato inquinante
- 1st Max Value: valore massimo giornaliero del dato inquinante
- 1st Max Hour: valore massimo per 1 ora di misurazione del dato inquinante
- AQI: indice di qualità dell'aria per il dato inquinante

Particolare attenzione verrà riservata ai campi che riguardano l'AQI.

L'Air Quality Index (AQI) è un numero adimensionale che varia da 0 a 500. Maggiore è il suo valore, maggiore è il livello di inquinamento dell'aria e quindi maggiori sono i rischi per la salute. Per esempio, un AQI di 50 o più basso rappresenta in genere una buona qualità dell'aria, mentre AQI che superano la soglia di 300 identifica scarsa qualità dell'aria e quindi pericoloso per la salute.

L'indice è suddiviso in 6 categorie, ciascuna della quali corrisponde a un diverso livello di rischio:

Daily AQI Color	Levels of Concern	Values of Index	Description of Air Quality
Green	Good	0 to 50	Air quality is satisfactory, and air pollution poses little or no risk.
Yellow	Moderate	51 to 100	Air quality is acceptable. However, there may be a risk for some people, particularly those who are unusually sensitive to air pollution.
Orange	Unhealthy for Sensitive Groups	101 to 150	Members of sensitive groups may experience health effects. The general public is less likely to be affected.
Red	Unhealthy	151 to 200	Some members of the general public may experience health effects; members of sensitive groups may experience more serious health effects.
Purple	Very Unhealthy	201 to 300	Health alert: The risk of health effects is increased for everyone.
Maroon	Hazardous	301 and higher	Health warning of emergency conditions: everyone is more likely to be affected.

Figura 5. Tabella di rischio in base al valore di AQI

4 Progetto

4.1 Ingestion dei dati

In questa fase dell'architettura si ha la necessità di uno strumento che "produca" dei flussi di dati ad intervalli regolari. Come già spiegato precedentemente la scelta architetturale per l'ingestion dei dati è stata Kafka.

Per poter inviare messaggi un producer deve essere creato un canale o topic. Il topic principale del progetto è 'sensors-data', ed è anche l'unico, in quanto avendo una sola tipologia di messaggio da produrre, si è ritenuto inutile la presenza di ulteriori canali.

Una prima fase di valutazione del dataset, ci ha portato a cancellare alcune colonne ridondanti ai fini del task. Il dataset utilizzato per svolgere il progetto contiene sempre +2M di ennuple ma con un numero ridotto di campi (circa 17). Ognuna delle ennuple contiene informazioni relative alla stazione che ha effettuato la misurazione dei livelli di inquinamento e quindi i relativi valori individuati. Tali campi sono:

- State Code: codice dello stato dove è stata effettuata la misurazione
- State: nome dello stato
- County Code: codice della contea dove è stata effettuata la misurazione
- County: nome della contea
- City: nome della città dove è dove è stata effettuata la misurazione
- Date Local: giorno nel quale è avvenuta la misurazione
- NO2 Units: unità di misura utilizzata per il biossido di azoto
- NO2 Mean: media giornaliera di concentrazione del biossido di azoto
- NO2 AQI: indice di qualità dell'aria con riferimento al biossido di azoto
- SO2 Units: unità di misura utilizzata per il biossido di azoto
- SO2 Mean: media giornaliera di concentrazione del biossido di azoto
- SO2 AQI: indice di qualità dell'aria con riferimento al biossido di azoto
- CO Units: unità di misura utilizzata per il biossido di azoto
- CO Mean: media giornaliera di concentrazione del biossido di azoto
- CO AQI: indice di qualità dell'aria con riferimento al biossido di azoto
- O3 Units: unità di misura utilizzata per il biossido di azoto
- O3 Mean: media giornaliera di concentrazione del biossido di azoto
- O3 AQI: indice di qualità dell'aria con riferimento al biossido di azoto

Attraverso Kafka quindi si è simulata la cattura di eventi e quindi la successiva produzione di messaggi su base giornaliera, ponendo una condizione sul campo Date Local. Il producer produce dei mini-batch di record in base alla data in esame: quando deve inviare un messaggio che è riferito a un diverso giorno, aspetta 30 secondi e quindi invia un secondo mini-batch riferito al giorno successivo e così via.

Questa simulazione si è potuta effettuare in quanto il dataset è una serie storica, e quindi ordinato su in modo temporale.

4.2 Consumers

I dati giornalieri prodotti dal kafka producer vengono consumati da due ascoltatori sottoscritti all'unico topic disponibile, sensors-data:

- **stream-processor**: che esegue una prima elaborazione sui dati, e popola le tabelle del database con i dati non aggregati.

- **daily-streamer:** che elabora i dati in base al giorno, ed esegue una prima fase di analisi.

4.3 Streaming Layer

La streaming analisys quindi è eseguita attraverso i dati consumati dal daily-streamer.
Il flusso dati viene passato a una prima fase di preelaborazione, dove vengono eliminate le ennuple contenenti valori mancanti, e successivamente, viene applicato un filtro che seleziona le misurazioni che oltrepassano una determinata soglia.
Tale soglia è quella relativa al 'rischio moderato' (Figura 5), e fissata a un valore di 51 per tutti i quattro inquinanti.

```
most_polluted_by_day = dataframe \
    .filter((col('no2aqi') > THRESHOLD) |
            (col('so2aqi') > THRESHOLD) |
            (col('coaqi') > THRESHOLD)  |
            (col('o3aqi') > THRESHOLD))\
    .select('state', 'county', 'city', 'day', 'month', 'year', 'no2aqi', 'so2aqi', 'coaqi', 'o3aqi')
```

Figura 6: esempio di selezione delle ennuple che superano una determinata soglia

Spark rende tutte queste operazioni molto semplici, grazie all'utilizzo dei dataframe, possono essere eseguite query sul flusso dati e creare tabelle che hanno la stessa struttura delle tabelle nel database, rendendo semplice l'integrazione tra lo streamer e il database.

Le ennuple vengono quindi memorizzate sulla relativa tabella in Cassandra (grazie all'utilizzo di specifici driver) in modalità append.

```
to_append = most_polluted_by_day.writeStream \
    .trigger(processingTime="5 seconds")\
    .format("org.apache.spark.sql.cassandra")\
    .option("keyspace", "airqualitykeyspace")\
    .option("checkpointLocation", '/tmp/daily/checkpoint/')\
    .option("table", "daily_pollution")\
    .outputMode("append")\
    .start()
```

Figura 7: esempio di memorizzazione dei dati in Cassandra

Nel database quindi si avrà una tabella, a cui vengono appese ennuple che soddisfanno per il giorno, le condizioni descritte in precedenza; ossia se almeno uno degli inquinanti supera la soglia di rischio moderato. Le ennuple vengono appese a tale tabella con un ritardo di 30 secondi, lo stesso ritardo con cui il producer invia i messaggi relativi a due giorni successivi.

4.4 Batch Layer

La batch analysis viene eseguita sui dati non aggregati salvati in un primo momento sul database dallo **stream-processor**.

I protagonisti della batch analisys sono due programmi pyspark, che leggono i dati 'grezzi' dalla relativa tabella sul database. Grazie all'ottima integrazione tra Spark e Cassandra, è possibile riprendere dati scritti nel database ed eseguire alcune elaborazioni su tali dati.

Tali programmi, quindi importano la tabella dei dati grezzi sottoforma di dataframe. Su tale dataframe poi vengono eseguite operazioni di aggregazioni in base a determinati campi scelti.

- **Viste batch mensili:** le batch views mensili vengono create da month-most-polluted.
  In fase di progettazione si è deciso di creare quattro view mensili, una per ciascun inquinante.
  Il dataframe che contiene i dati grezzi viene aggregato dapprima per mese e anno e successivamente per stato. Viene quindi selezionata la città che ha il livello di concentrazione più alto (per un determinato inquinante), per stato del mese.

```
no2_most_polluted = pollution \
    .groupBy('state', 'month', 'year') \
    .agg(avg('no2aqi').alias('no2aqi')) \
    .select('state', 'month', 'year', 'no2aqi')
```

Figura 8: creazione della vista mensile per SO2

Vengono quindi creati quattro diversi dataframe, ciascuno relativo a uno dei quattro inquinanti, che vengono memorizzati poi in quattro diverse tabelle in Cassandra.

- **Viste batch annuali:** la creazione delle batch views annuali non si discosta molto dalla creazione di quelle mensili. Il processo è praticamente lo stesso, solo con un diverso tipo di granularità dell'aggregazione, ossia solo per anno.

```
no2_most_polluted = pollution \
    .groupBy('state', 'year') \
    .agg(avg('no2aqi').alias('no2aqi')) \
    .select('state', 'year', 'no2aqi')
```

Figura 9: creazione della vista annuale per SO2

Il programma pyspark dedicato a tale scopo è year-most-polluted, che anche in questo caso ha in output quattro dataframe che vengono salvati anch'essi in quattro tabelle diverse su Cassandra, una per ciascun inquinante.

Questi due processi vengono rilanciati ripetutamente per tutta la durata della simulazione con un diverso ritardo ciascuno; la creazione delle viste mensili avviene ogni 300 minuti, mentre quelle annuali sono create ogni 10 minuti. Questi sono anche i tempi necessari al producer per produrre rispettivamente un intero mese e un intero anno di messaggi e quindi allo **stream-processor** per memorizzarli nella relativa tabella nel database.

4.5 Serving Layer

Il serving layer infine è la parte dell'architettura dove i flussi di dati provenienti dal batch layer e dallo streaming layer si incontrano.

La scelta tecnologica fatta per questo layer è Apache Cassandra, in quanto ha un'ottima integrazione sia con Kafka sia con Spark. Per connettere i tre strumenti, infatti, è necessaria solamente l'importazione di alcuni driver.

Per poter memorizzare i dati in Cassandra è necessaria la definizione di un **keyspace** e di tutte le tabelle che andranno a comporre il database. Tale definizione può essere eseguita attraverso un file con estensione '.cql', all'interno del quale vi sono i comandi necessari alla creazione del keyspace e delle tabelle.

In totale le collezioni utilizzate per il progetto sono 10:

- airqualitykeyspace.**pollution:** è la tabella destinataria di tutti i recordi non aggregati consumati dallo stream-processor e che quindi attendono di essere rielaborati dai batch layer
- airqualitykeyspace.**daily_pollution:** memorizza tutti i record elaborati dal daily_streamer, contiene quindi le città che per un determinato giorno in esame superano la soglia di 'rischio moderato'

E infine le viste batch mensili e annuali per ogni inquinante:

- airqualitykeyspace.**no2_month** e airqualitykeyspace.**no2_year** per il biossido di azoto
- airqualitykeyspace.**so2_month** e airqualitykeyspace.**so2_year** per il biossido di zolfo
- airqualitykeyspace.**co_month** e airqualitykeyspace.**co_year** per il monossido di carbonio
- airqualitykeyspace.**o3_month** e airqualitykeyspace.**o3_year** per l'ozono

year	month	state	city	so2aqi	so2mean
2010	5	Arkansas	North Little Rock	2	1.03
2010	5	Colorado	Welby	1	0.04
2010	5	District Of Columbia	Washington	4	2.36
2010	5	Florida	Winter Park	0	0.01
2010	5	Indiana	Indianapolis (Remainder)	7	2.58
2010	5	Kansas	Kansas City	5	1.62
2010	5	Louisiana	Baton Rouge	1	0.47
2010	5	Maryland	Essex	5	0.73
2010	5	Massachusetts	Boston	3	1.02
2010	5	New Hampshire	Manchester	0	0.16
2010	5	New York	New York	6	2.27
2010	5	North Carolina	Charlotte	1	0.47
2010	5	Oklahoma	Cherry Tree	1	0.88
2010	5	Oregon	Portland	2	1.15
2010	5	Virginia	Alexandria	4	1.37

year	state	city	so2aqi	so2mean
2010	Arkansas	North Little Rock	3	1.45
2010	Colorado	Welby	7	1.03
2010	District Of Columbia	Washington	8	3.38
2010	Florida	Winter Park	1	0.15
2010	Indiana	Indianapolis (Remainder)	10	3
2010	Kansas	Kansas City	14	5.21
2010	Louisiana	Baton Rouge	7	1.36
2010	Maryland	Essex	7	2.17
2010	Massachusetts	Boston	4	1.6
2010	Nevada	Reno	1	0.43
2010	New Hampshire	Manchester	8	0.98
2010	North Carolina	Charlotte	3	0.92
2010	Oregon	Portland	2	1.35

Figura 10: vista mensile e annuale, rispettivamente per SO2

Tutte le tabelle hanno un ordine di clustering decrescente, cosicché record relativi a giorni recenti stiano in testa alla tabella. Questa operazione è necessaria in quanto rende il lavoro più semplice alla successiva fase di visualizzazione dei dati attraverso la dashboard.

4.6 Dashboard

La dashboard del progetto è una semplice web application creata utilizzando le funzionalità di Flask. Grazie a Flask è possibile, attraverso poche righe di codice, avviare un web server attraverso il comando 'flask run'.
Il codice Flask è racchiuso tutto in un unico file, denominato app.py, che ha al suo interno le specifiche di routing della web app.

Il metodo principale da definire è il 'main'. Tale metodo ha come path di rounting il root '/' e permette chiamate http GET al server.
Il main, inoltre, ha la facoltà di instaurare la connessione con il cluster Cassandra e richiedere una prima versione delle tabelle all'interno del database. Il main infine ritorna un oggetto creato dal metodo 'render_template'. Nel metodo 'render_template' è possibile specificare sia il template da mostrare graficamente, sia gli oggetti che devono essere oggetti da restituire.

```
@app.route('/', methods=["GET"])
def main():

    cluster = Cluster()
    session = cluster.connect("airqualitykeyspace")
    daily_pollution = session.execute("SELECT * FROM daily_pollution")

    return render_template('index.html', daily_data = daily_pollution)
```

Figura 11: metodo main in app.py

Infine, attraverso le funzionalità di Jinja2, è stato possibile mostrare tutti i dati nella web app, semplicemente iterando nelle collezioni ritornate dal metodo 'render_template'.

In fase di progettazione volevamo avere una dashboard che non fosse statica, e che quindi si aggiornasse con i nuovi dati memorizzati periodicamente sul database. Per fare ciò si sono definiti diversi metodi che interrogano periodicamente il database attraverso chiamate AJAX.

```python
@app.route('/daily-data' , methods=["GET", "POST"])
def daily_data():

    daily_pollution = session.execute("SELECT * FROM daily_pollution")
    data = daily_pollution.all()

    return json.dumps(data)
```

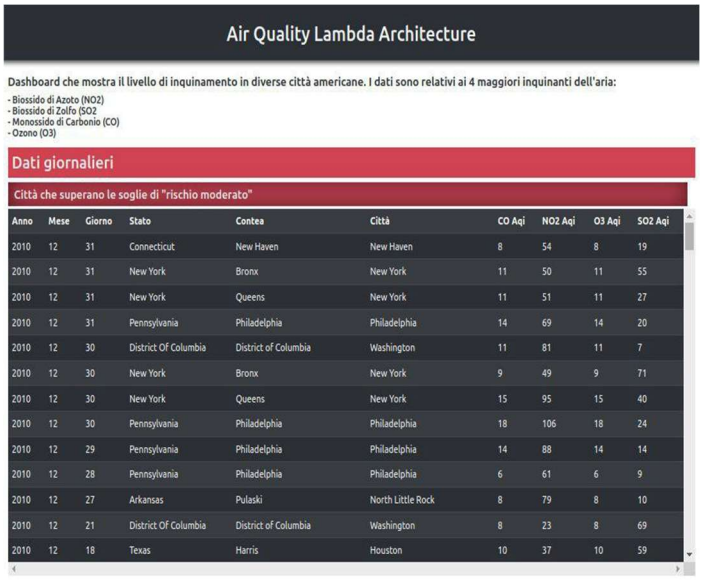
Figura 12: metodo per l'update della table giornaliera

In questo modo è possibile settare un intervallo di tempo per tali richieste attraverso degli script scritti in Javascript. L'intervallo di tempo viene settato con il metodo setInterval:

- La tabella dei dati giornalieri nella dashboard viene aggiornata ogni 15 secondi, mostrando così le città che hanno superato la soglia di rischio per l'ultimo giorno analizzato.

- Le tabelle mensili nella dashboard vengono aggiornate ogni 300 secondi (5 min), e quindi mostrano i dati aggregati riferiti all'ultimo mese analizzato.

- Le tabelle annuali nella dashboard vengono aggiornate ogni 600 secondi (10 min), e quindi mostrano i dati aggregati riferiti all'ultimo anno analizzato.



Air Quality Lambda Architecture

Dashboard che mostra il livello di inquinamento in diverse città americane. I dati sono relativi ai 4 maggiori inquinanti dell'aria:

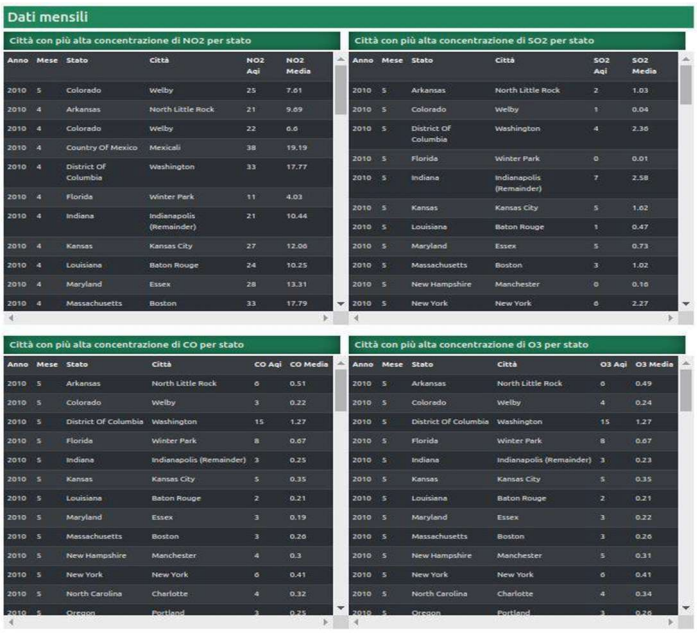
- Biossido di Azoto (NO2)
- Biossido di Zolfo (SO2
- Monossido di Carbonio (CO)
- Ozono (O3)

Dati giornalieri

Città che superano le soglie di "rischio moderato"

Anno	Mese	Giorno	Stato	Contea	Città	CO Aqi	NO2 Aqi	O3 Aqi	SO2 Aqi
2010	12	31	Connecticut	New Haven	New Haven	8	54	8	19
2010	12	31	New York	Bronx	New York	11	50	11	55
2010	12	31	New York	Queens	New York	11	51	11	27
2010	12	31	Pennsylvania	Philadelphia	Philadelphia	14	69	14	20
2010	12	30	District Of Columbia	District of Columbia	Washington	11	81	11	7
2010	12	30	New York	Bronx	New York	9	49	9	71
2010	12	30	New York	Queens	New York	15	95	15	40
2010	12	30	Pennsylvania	Philadelphia	Philadelphia	18	106	18	24
2010	12	29	Pennsylvania	Philadelphia	Philadelphia	14	88	14	14
2010	12	28	Pennsylvania	Philadelphia	Philadelphia	6	61	6	9
2010	12	27	Arkansas	Pulaski	North Little Rock	8	79	8	10
2010	12	21	District Of Columbia	District of Columbia	Washington	8	23	8	69
2010	12	18	Texas	Harris	Houston	10	37	10	59

Figura 13: tabelle dei dati giornalieri



Dati mensili

Città con più alta concentrazione di NO2 per stato

Anno	Mese	Stato	Città	NO2 Aqi	NO2 Media
2010	5	Colorado	Welby	25	7.61
2010	4	Arkansas	North Little Rock	21	9.69
2010	4	Colorado	Welby	22	6.6
2010	4	Country Of Mexico	Mexicali	38	19.19
2010	4	District Of Columbia	Washington	33	17.77
2010	4	Florida	Winter Park	11	4.03
2010	4	Indiana	Indianapolis (Remainder)	21	10.44
2010	4	Kansas	Kansas City	27	12.06
2010	4	Louisiana	Baton Rouge	24	10.25
2010	4	Maryland	Essex	28	13.31
2010	4	Massachusetts	Boston	33	17.79

Città con più alta concentrazione di SO2 per stato

Anno	Mese	Stato	Città	SO2 Aqi	SO2 Media
2010	5	Arkansas	North Little Rock	2	1.03
2010	5	Colorado	Welby	1	0.04
2010	5	District Of Columbia	Washington	4	2.36
2010	5	Florida	Winter Park	0	0.01
2010	5	Indiana	Indianapolis (Remainder)	7	2.58
2010	5	Kansas	Kansas City	5	1.62
2010	5	Louisiana	Baton Rouge	1	0.47
2010	5	Maryland	Essex	5	0.73
2010	5	Massachusetts	Boston	3	1.02
2010	5	New Hampshire	Manchester	0	0.16
2010	5	New York	New York	8	2.27

Città con più alta concentrazione di CO per stato

Anno	Mese	Stato	Città	CO Aqi	CO Media
2010	5	Arkansas	North Little Rock	6	0.51
2010	5	Colorado	Welby	3	0.22
2010	5	District Of Columbia	Washington	15	1.27
2010	5	Florida	Winter Park	8	0.67
2010	5	Indiana	Indianapolis (Remainder)	3	0.25
2010	5	Kansas	Kansas City	5	0.35
2010	5	Louisiana	Baton Rouge	2	0.21
2010	5	Maryland	Essex	3	0.19
2010	5	Massachusetts	Boston	3	0.26
2010	5	New Hampshire	Manchester	4	0.3
2010	5	New York	New York	6	0.41
2010	5	North Carolina	Charlotte	4	0.32
2010	5	Oregon	Portland	3	0.25

Città con più alta concentrazione di O3 per stato

Anno	Mese	Stato	Città	O3 Aqi	O3 Media
2010	5	Arkansas	North Little Rock	6	0.49
2010	5	Colorado	Welby	4	0.24
2010	5	District Of Columbia	Washington	15	1.27
2010	5	Florida	Winter Park	8	0.67
2010	5	Indiana	Indianapolis (Remainder)	3	0.23
2010	5	Kansas	Kansas City	5	0.35
2010	5	Louisiana	Baton Rouge	2	0.21
2010	5	Maryland	Essex	3	0.22
2010	5	Massachusetts	Boston	3	0.26
2010	5	New Hampshire	Manchester	5	0.31
2010	5	New York	New York	6	0.41
2010	5	North Carolina	Charlotte	4	0.34
2010	5	Oregon	Portland	3	0.26

Figura 14: tabelle dei dati mensili

Figura 15: tabelle dei dati annuali

5 Problemi riscontrati

5.1 Implementazione dell'architettura con Docker

Sono state realizzate due versioni dell'architettura descritta nella sessione riguardante il Progetto, tra cui una versione implementata con Docker.
Ci sono stati diverse problematiche che ci hanno fatto desistere dall'idea di presentarla in questa relazione, ma è comunque disponibile nel repository del progetto su GitHub.

Tali problemi sono riassunti di seguito:

- Non sempre gli spark-worker riuscivano a stabilire una connessione con il cluster Cassandra, era necessario quindi riavviare la fase di analisi batch diverse volte e sperare che venisse portata a termine.

- La comunicazione in parallelo tra più spark-worker con Cassandra causava la perdita di un nodo random relativo del cluster, con successiva perdita di interazione con il resto dell'architettura. Abbiamo notato che tale problema è dovuto alla scarsa disponibilità di risorse di memoria. Supponiamo che per l'esecuzione dell'architettura su Docker sia necessaria una quantità maggiore di memoria.

6 Conclusioni

L'architettura lambda si rivela molto utile e efficace per l'analisi di grossi flussi di dati provenienti da sensori. Tale modello architetturale permette di bilanciare in modo ottimale il trade-off tra bassa latenza e alto throughput per l'analisi di dati su larga

scala. Con riferimento al caso d'uso studiato, l'architettura si comporta egregiamente e l'interazione tra le varie tecnologie utilizzate avviene in maniera rapida e intuitiva.

7 Riferimenti

Repo GitHub del progetto - MatteoDeSantisS/Big_Data_Lambda_Architecture (github.com)

Dataset - U.S. Pollution Data | Kaggle

Fonti AQI - AQI Basics | AirNow.gov

Apache Kafka - Apache Kafka

Apache Spark - Apache Spark™ - Unified Analytics Engine for Big Data

Apache Cassandra - Apache Cassandra | Apache Cassandra Documentation

Flask - Welcome to Flask — Flask Documentation (2.0.x) (palletsprojects.com)