

# Lambda Project Report

Matteo Paparo

22 gennaio 2026

## Sommario

Il seguente elaborato consiste nell'implementazione di un'**Architettura Lambda** e il suo adattamento alla gestione di dati di natura finanziaria. I dati in questione sono le *criptovalute*, caratterizzate da una grande disponibilità e da oscillazioni dei prezzi molto accentuate in tempi brevi. L'architettura in questione sarà composta da tre layer principali: Il **Batch Layer**, caratterizzato da un'alta latenza e da un'elaborazione completa dei dati; lo **Speed Layer**, che si occuperà di processare i dati in tempo reale con una latenza molto bassa; per ultimo il **Serving Layer**, che si occuperà di fornire le view del Batch Layer. Il fine è quello di effettuare delle analisi sui dati raccolti, e per aumentare la loro veridicità è stato implementato un modello statistico, per effettuare una fase preliminare di *cleaning*, utilizzando un filtro per la rimozione degli outlier basato sulla **Regola dei 3 Sigma (Z-Score)**. Infine, per visualizzare tali analisi, è stata implementata una **Dashboard**, che consente di visualizzare i dati in modo semplice ed intuitivo.

# Indice

<b>1</b>	<b>Configurazione</b>	<b>2</b>
1.1	Workflow . . . . .	3
<b>2</b>	<b>Data Source</b>	<b>4</b>
2.1	Configurazione Docker . . . . .	4
<b>3</b>	<b>Batch Layer</b>	<b>5</b>
3.1	Workflow . . . . .	6
3.2	Configurazione Docker . . . . .	7
<b>4</b>	<b>Speed Layer</b>	<b>8</b>
4.1	Workflow . . . . .	9
4.2	Configurazione Docker . . . . .	9
<b>5</b>	<b>Serving Layer &amp; Dashboard</b>	<b>10</b>
<b>6</b>	<b>Results Analysis</b>	<b>13</b>
6.1	Analisi del Batch Layer . . . . .	13
6.2	Performance dello Speed Layer . . . . .	14
6.3	Efficienza del Serving Layer . . . . .	15
6.4	Bilanciamento delle Risorse . . . . .	17
6.5	Valutazione Complessiva . . . . .	19
<b>7</b>	<b>Conclusione</b>	<b>20</b>
7.1	Sintesi del Valore Progettuale e Innovazione . . . . .	20
7.2	Validazione Scientifica dell’Isolamento . . . . .	20
7.3	Scalabilità e Visione Futura . . . . .	20
7.4	Considerazioni Finali . . . . .	21

# Capitolo 1

## Configurazione

Per la realizzazione del progetto è stato utilizzato come ambiente di lavoro **Docker**, che consente di creare dei container isolati per ogni componente dell'Architettura Lambda. In questo modo è stato possibile gestire le dipendenze e le configurazioni in modo semplice ed efficiente.

Il docker compose utilizzato per l'implementazione dell'architettura è composto da tre macro componenti:

- Ecosistema **Hadoop** per la gestione dell'archiviazione e dell'elaborazione dei dati su larga scala.
- **Apache Kafka** come message broker per il disaccoppiamento tra produttore e consumatori, garantendo persistenza e replay dei messaggi.
- **Apache Spark** come motore di elaborazione batch per il calcolo di metriche complesse (OHLC, RSI, Bollinger Bands) e per il training del modello di anomaly detection.
- Database **Cassandra** per la gestione dei dati in tempo reale e l'archiviazione veloce nello Speed Layer.
- **Servizi applicativi:**
  - **Kafka Producer:** Generatore dei dati, che utilizza API (*Coinbase* e *CoinGecko*) e WebSocket (*Binance*) per raccogliere dati sulle criptovalute. Permettono di avere un flusso continuo di dati con oscillazioni che variano da un minimo di 10, con picchi di 80 dati al secondo. In condizioni ottimali, è stato registrato una media di circa 35 dati al secondo.
  - **Dashboard** per la visualizzazione dei dati raccolti e delle analisi effettuate.

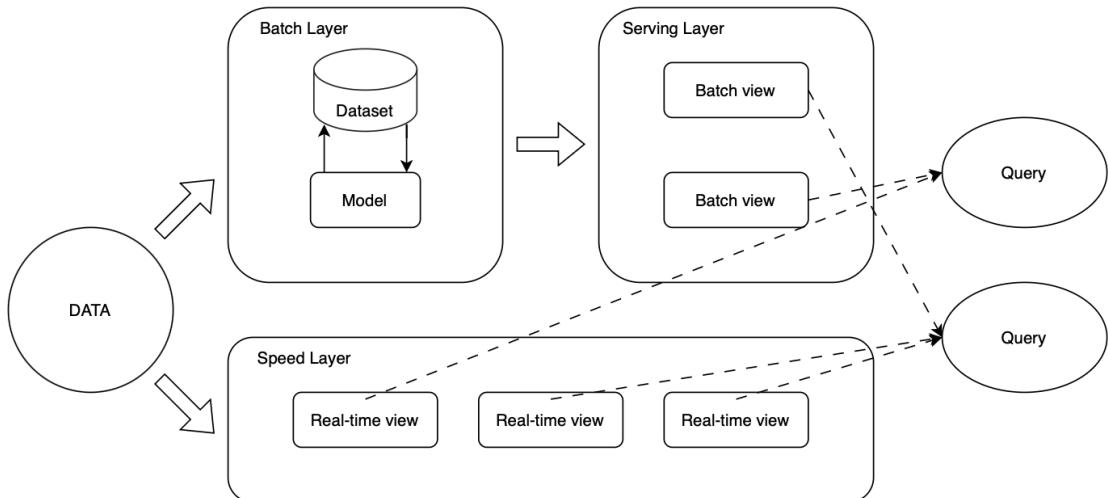
Le risorse allocate a Docker Desktop sono 12GB di RAM, 8 CPU e 2 di Swap, per garantire un funzionamento fluido dell'architettura Lambda implementata.

Il progetto è stato sviluppato su un MacBook Air con chip M3, 16Gb di RAM e 216Gb di spazio di archiviazione. Sono stati introdotti dei limiti di CPU per prevenire situazioni di *resource starvation*, garantendo che nessun container possa monopolizzare le risorse computazionali a discapito degli altri, e per simulare un ambiente di produzione con risorse condivise.

## 1.1 Workflow

Dall'avvio del docker compose, il flusso di lavoro dell'architettura Lambda segue questi passaggi principali:

1. Viene effettuata una fase di calibrazione iniziale, ovvero per i primi 5 minuti vengono raccolti i dati su HDFS senza effettuare alcuna elaborazione, in modo da avere un dataset iniziale sul quale creare il modello per il cleaning dei dati.
2. Una volta terminata la fase di calibrazione, il producer inizia a inviare i dati anche e allo Speed Layer, i quali effettueranno il filtraggio utilizzando il modello statistico appena creato.
3. I dati filtrati verranno poi salvati rispettivamente su HDFS e su Cassandra e disponibili per le analisi e la visualizzazione tramite la Dashboard.



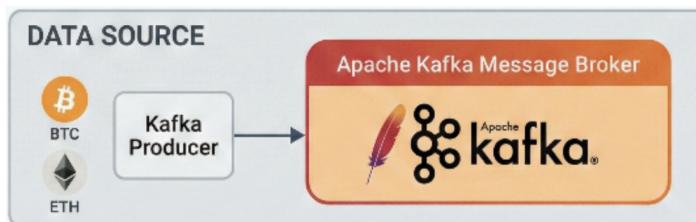
# Capitolo 2

## Data Source

Il responsabile della generazione dei dati è il **Kafka Producer**, che utilizza diverse API e WebSocket per raccogliere dati sulle criptovalute, che successivamente vengono inviati al **Kafka Broker**, che funge da intermediario tra i produttori e i consumatori.

Organizza i file in topic e distribuisce contemporaneamente gli stessi dati a entrambi i layer, attraverso l'uso di consumer dedicati (in questo caso per ogni layer), permettendo così la possibilità di aumentare la **scalabilità** del sistema.

Esso mantiene i messaggi per un periodo configurabile (di default 7 giorni) e permette il replay dei dati se necessario, garantendo sempre la consegna dei messaggi, rendendo il sistema più **robusto** e **affidabile**.



### 2.1 Configurazione Docker

Nello specifico l'architettura del Data Source è composta da:

- **Kafka**: message broker distribuito che costituisce il nucleo del sistema di messaggistica. Gestisce l'ingestione dei dati dal Producer, la persistenza temporanea dei messaggi e la loro distribuzione ai Consumer dei due layer (Speed e Batch). Nel progetto è stato configurato un singolo broker con un topic dedicato (`crypto-prices`) per il flusso dei prezzi delle criptovalute.
- **Kafka Producer**: responsabile della generazione e dell'invio dei dati al broker Kafka. Utilizza API pubbliche (*Coinbase* e *CoinGecko*) e WebSocket (*Binance*) per raccogliere dati sulle criptovalute in tempo reale. I dati raccolti includono solamente informazioni sul prezzo.
- **Zookeeper**: servizio di coordinamento distribuito utilizzato da Kafka per la gestione della configurazione del cluster, l'elezione del broker leader, il monitoraggio dello stato dei nodi tramite heartbeat e la sincronizzazione dei metadati relativi ai topic e alle partizioni.

# Capitolo 3

## Batch Layer

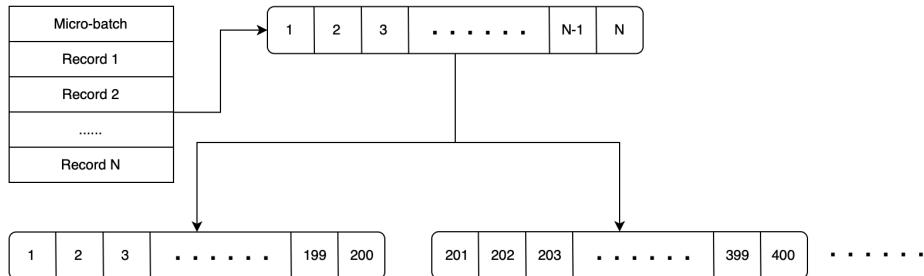
Il *Batch Layer* è uno dei tre Layer principali dell’Architettura Lambda, responsabile dell’archiviazione e dell’elaborazione completa dei dati, consentendo di eseguire analisi approfondite sui dati storici. In questa sezione verranno descritte le componenti principali del Batch Layer, le tecnologie utilizzate e le modalità di implementazione.

Il Batch Layer è stato implementato utilizzando **Apache Hadoop** come framework principale per l’archiviazione e **Apache Spark** per l’elaborazione dei dati. Hadoop consente di gestire grandi quantità di dati in modo distribuito, mentre Spark fornisce un motore di elaborazione veloce e in-memory per l’analisi dei dati.

Per adattare il Batch Layer all’analisi del mercato delle criptovalute, la sua struttura è stata riorganizzata al fine di consentire analisi frequenti sull’intero insieme dei dati processati. Questa rimodulazione si concentra principalmente sulla gestione dei dati e prevede l’introduzione di micro-batch, ovvero partizioni di tipo dimensionale o temporale, generate secondo due criteri alternativi: il raggiungimento di 3000 record oppure, in assenza di tale soglia, la creazione forzata del micro-batch dopo 180 secondi, includendo tutti i record raccolti fino a quel momento. Questo approccio consente di eseguire analisi rapide e aggiornate su tutti i dati disponibili, evitando la necessità di rielaborare l’intero dataset pulito memorizzato su HDFS.

Le analisi effettuate comprendono il calcolo di metriche statistiche di base, quali **minimo**, **massimo**, **media** e **volatilità**, oltre a indicatori tecnici ampiamente utilizzati nell’analisi finanziaria, tra cui **RSI** (Relative Strength Index), **Bollinger Bands** e **Momentum**. Infine, per fornire un’indicazione sulla quantità di informazioni analizzate, viene riportato anche il numero **totale di record puliti** considerati. L’impiego di tali indicatori risulta fondamentale nell’analisi tecnica, poiché permette di valutare l’andamento del mercato e supportare decisioni di investimento più consapevoli.

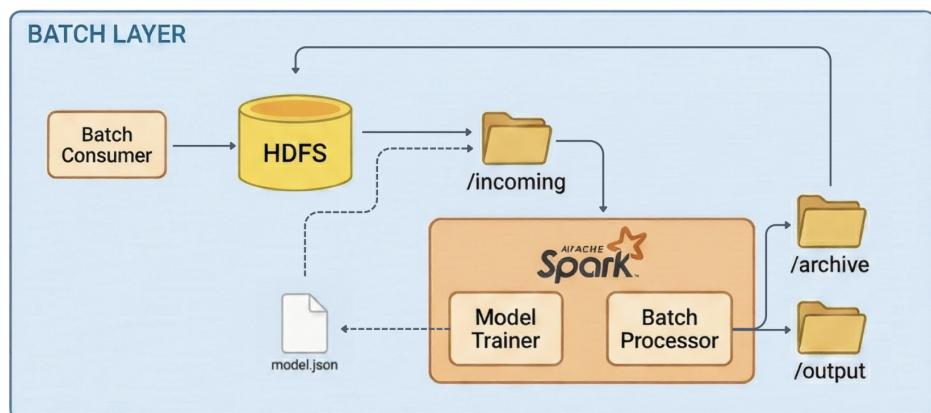
In merito alla fase di cleaning, essa non viene effettuata usando un approccio statico, ovvero secondo tutto il micro-batch viene filtrato direttamente con le informazioni contenute nel modello, ma il filtraggio avviene gradualmente. Più precisamente il micro-batch viene ordinato cronologicamente e viene suddiviso in **chunk** da 200 record. Successivamente ogni chunk viene filtrato e i dati puliti vengono utilizzati per aggiornare il modello, permettendo così che si adatti all’andamento del mercato e possa supportare le oscillazioni del mercato.



Poichè il numero di record contenuti in ogni micro-batch può variare, poichè ci sta anche un vincolo temporale, il numero di chunk non è detto che sia sempre lo stesso, comportando quindi che l'ultimo abbia un numero minore di record. Anche in questo caso il chunk viene filtrato e i dati puliti vengono utilizzati per aggiornare il modello.

### 3.1 Workflow

Quindi il workflow di questo layer è dato da: (1) Il **Batch Consumer** indirizza i dati in arrivo dal **Kafka Broker** e li scrive su **HDFS** come file .jsonl nella cartella `/iot-data/incoming`. (2) Lo Spark job `model_trainer.py` legge esclusivamente i dati dalla cartella `/incoming` (finestra temporale corrente), calcola media e deviazione standard per ogni sensore, e salva il modello in `/models/model.json`. Questo garantisce che il modello rifletta sempre le condizioni di mercato più recenti. (3) Lo Spark job `batch_processor.py` in sequenza esegue il caricamento del modello e filtra le anomalie dai dati grezzi. Calcola le metriche e salva i risultati aggregati in `/iot-output/spark/date=YYYY-MM-DD`. Archivia i dati filtrati in `/iot-data/archive` e successivamente elimina i dati dalla cartella `/incoming` per liberare spazio per i nuovi dati, in modo da evitare di processare dati già elaborati.



## 3.2 Configurazione Docker

Nello specifico l'architettura del Batch Layer è composta da:

- **Namenode**: nodo "master" del file system **HDFS** (Hadoop Distributed File System). Non memorizza i dati reali, ma gestisce i metadati.
- **Datanode**: nodo "worker" che memorizzano effettivamente i dati. Gestisce lo storage locale e risponde alle richieste di lettura e scrittura dei dati.
- **Spark Scheduler**: è l'orchestratore che gestisce i tempi di esecuzione dei job Spark:
  - `model_trainer.py`: Eseguito solo una volta, dopo 5 minuti dall'avvio dell'architettura, calcola media e deviazione standard per sensore dai dati recenti e salva `model.json`.
  - `batch_processor.py`: Eseguito ogni 2 minuti oppure ogni micro-batch completato. Elabora i dati in `/incoming`, calcola metriche OHLC e indicatori tecnici, e archivia i dati processati.

Inoltre ogni 5 minuti, controlla se è stato aggiunto un nuovo sensore su HDFS. In caso affermativo, riesegue nuovamente il job `model_trainer.py` per includere il nuovo sensore.

- **Spark Master**: coordinatore del cluster Spark. Gestisce le risorse e distribuisce i compiti ai nodi worker. Riceve i job Spark e decide quale Worker eseguirà quale task. Monitora lo stato dei worker.
- **Spark Worker<sup>1</sup>**: è il nodo che esegue i task assegnati dal Master. Si occupa dell'elaborazione dei dati (filter, map, aggregate) che una volta eseguita, invia i risultati al Master.
- **Batch Consumer**: responsabile del consumo dei dati elaborati dal Batch Layer e della loro scrittura nel sistema di archiviazione finale. Per sfruttare al meglio le capacità di scrittura di HDFS, i dati vengono accumulati prima di esser scritti, ovvero ogni 5000 messaggi o ogni 180 secondi, a seconda di quale condizione si verifica per prima.

---

<sup>1</sup>Nel progetto sono stati utilizzati 3 nodi Spark Worker, uno per ogni sensore.

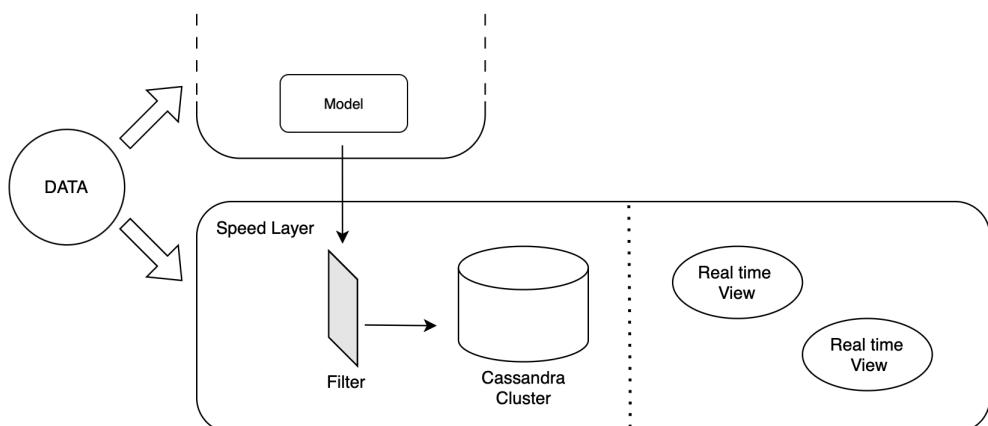
# Capitolo 4

## Speed Layer

Il secondo layer principale che compone l'Architettura Lambda è lo *Speed Layer*, responsabile dell'elaborazione dei dati in tempo reale con una latenza molto bassa. Questo layer è progettato per gestire i dati che richiedono risposte rapide e aggiornamenti frequenti, consentendo di ottenere informazioni tempestive sui dati in arrivo.

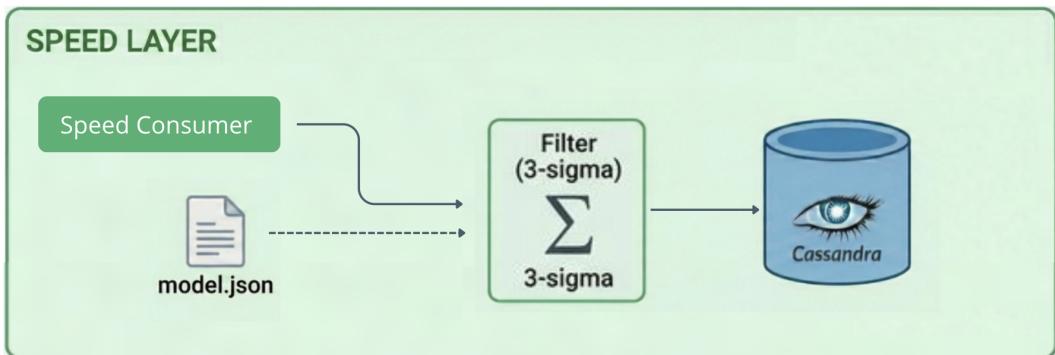
Lo **Speed Layer** è stato implementato utilizzando **Apache Cassandra** come database NoSQL per la gestione dei dati. Cassandra è progettato per gestire enormi quantità di dati su più server, garantendo elevata disponibilità e scalabilità. Nel caso dello Speed Layer, il dato viene prima filtrato utilizzando, inizialmente, lo stesso modello contenuto in `/model.json`, per garantire la coerenza tra i due layer. Successivamente, il dato filtrato viene salvato in una tabella di Cassandra appositamente creata per memorizzare i dati in tempo reale.

Rispetto al Batch Layer, l'approccio con il modello è differente, poichè viene aggiornato in memoria ad ogni nuovo dato valido ricevuto, in questo modo il modello si adatta continuamente alle condizioni di mercato più recenti. Viene utilizzato un **filtro adattivo EMA** (Exponential Moving Average), quindi invece di usare una media fissa, il filtro aggiorna la media e la varianza ad ogni singolo messaggio ricevuto, permettendo l'adattamento istantaneo ai trend di mercato. Se durante l'esecuzione viene rilevato un nuovo sensore, non ancora riportato nel modello, esso viene inserito in un buffer (capienza massima di 1000 record), in modo da aspettare l'aggiornamento del modello su HDFS.



## 4.1 Workflow

Il workflow di questo layer è dato da: (1) Lo **Speed Consumer** attende che il modello `model.json` sia disponibile su HDFS. (2) Successivamente inizializza le struttura EMA in memoria e indirizza i dati in arrivo dal Kafka Broker, applicando il filtro per il cleaning dei dati. (3) I dati filtrati vengono salvati nella tabella `sensor_data` di Cassandra e aggiorna il modello ad ogni dato ricevuto. (4) Ogni 10 secondi controlla se il modello HDFS è stato aggiornato; in caso affermativo, integra eventuali nuovi sensori. I sensori già presenti non vengono modificati. (5) I dati vengono resi disponibili per la visualizzazione tramite la Dashboard.



## 4.2 Configurazione Docker

Nello specifico, l'architettura dello Speed Layer è composta da due nodi:

- **Cassandra-seed**: è il nodo principale del cluster Cassandra. Quando altri nodi si avviano, contattano il Seed per scoprire la topologia del cluster. Oltre a fare da coordinatore, è un nodo operativo a tutti gli effetti: memorizza i dati nella tabella `sensor_data` e risponde alle query della Dashboard.
- **Speed Consumer**: responsabile a consumare i dati dal Broker Kafka, per ogni messaggio applica il filtro EMA adattivo, se il dato è valido, lo scrive su Cassandra e aggiorna il modello EMA in memoria<sup>1</sup>.

<sup>1</sup>Utilizza l'offset `latest` di Kafka per saltare automaticamente i dati di calibrazione

# Capitolo 5

## Serving Layer & Dashboard

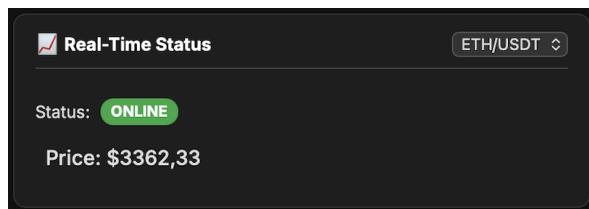
L'ultima componente dell'Architettura Lambda è il **Serving Layer**, che funge da interfaccia tra i dati elaborati dai due layer principali (Batch e Speed) e l'utente finale. Il Serving Layer è responsabile della fornitura di dati e analisi in modo efficiente e accessibile, consentendo agli utenti di visualizzare e interagire con i dati raccolti.

Il Serving Layer è stato implementato utilizzando **Flask**, un framework web leggero e flessibile per Python. Flask consente di creare applicazioni web in modo semplice e rapido, rendendolo ideale per la costruzione di interfacce utente per la visualizzazione dei dati.

La Dashboard è composta da diversi elementi per mostrare, non solo, il funzionamento dell'architettura, ma anche le sue prestazioni. Nello specifico abbiamo:

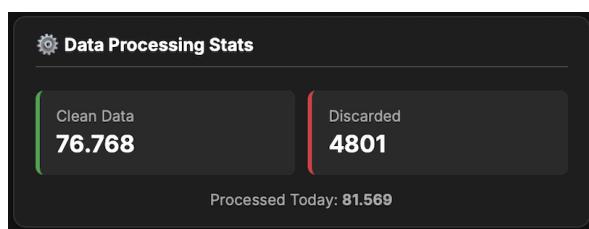
### Widget "Real-Time Status"

Mostra se i sensori sono attivi e l'ultimo prezzo ricevuto. Dal menù a tendina, in alto a destra è possibile modificare il titolo della cryptovaluta da monitorare.



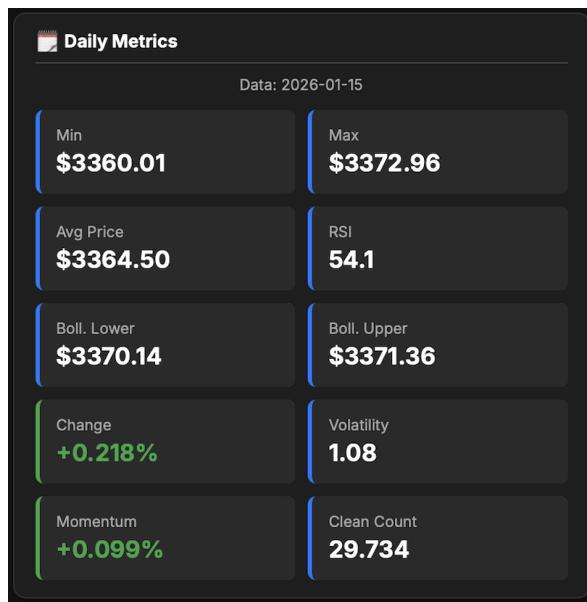
### Data Processing Stats

Mostra il numero di dati ricevuti fino all'ultimo batch processato, evidenziando il numero di dati totali, dati puliti e anomalie registrate.



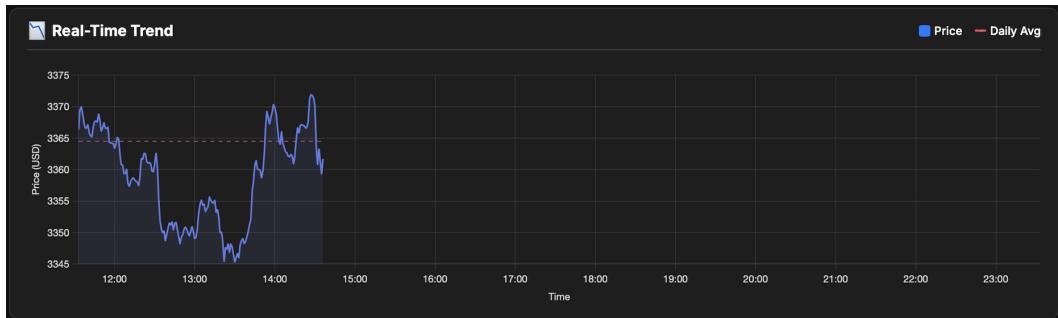
## Daily Metrics

Mostra le metriche giornaliere calcolate dal Batch Layer, sono metriche generali, aggiornati ad ogni micro-batch processato. I dati mostrati variano in base al titolo selezionato.



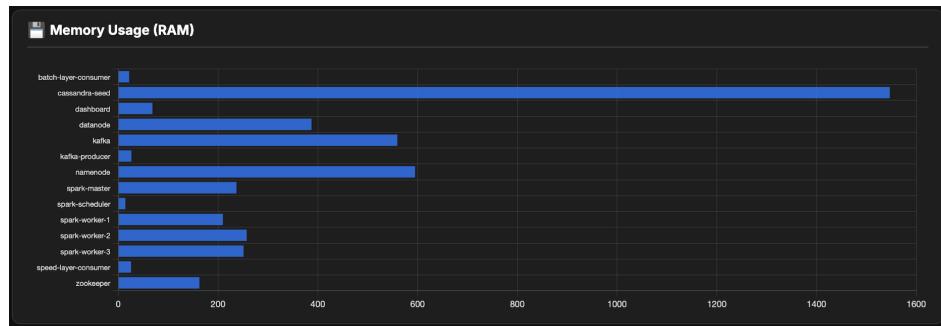
## Real Time Trend

Mostra l'andamento delle cryptovalute in tempo reale, aggiornando ogni minuto il grafico con i nuovi dati ricevuti dallo Speed Layer. Il grafico riporta le ultime 12 ore di dati; alla ricezione di dati nelle ore successive, il grafico rimuoverà i dati più vecchi per lasciar spazio ai nuovi, creando così un movimento fluido e continuo.



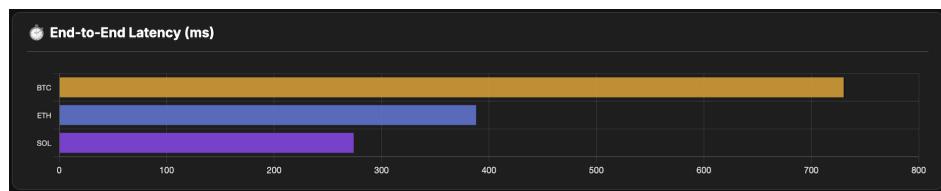
## Memory Usage

Mostra l'utilizzo della memoria di Cassandra e HDFS, per monitorare lo stato di salute del sistema di archiviazione.

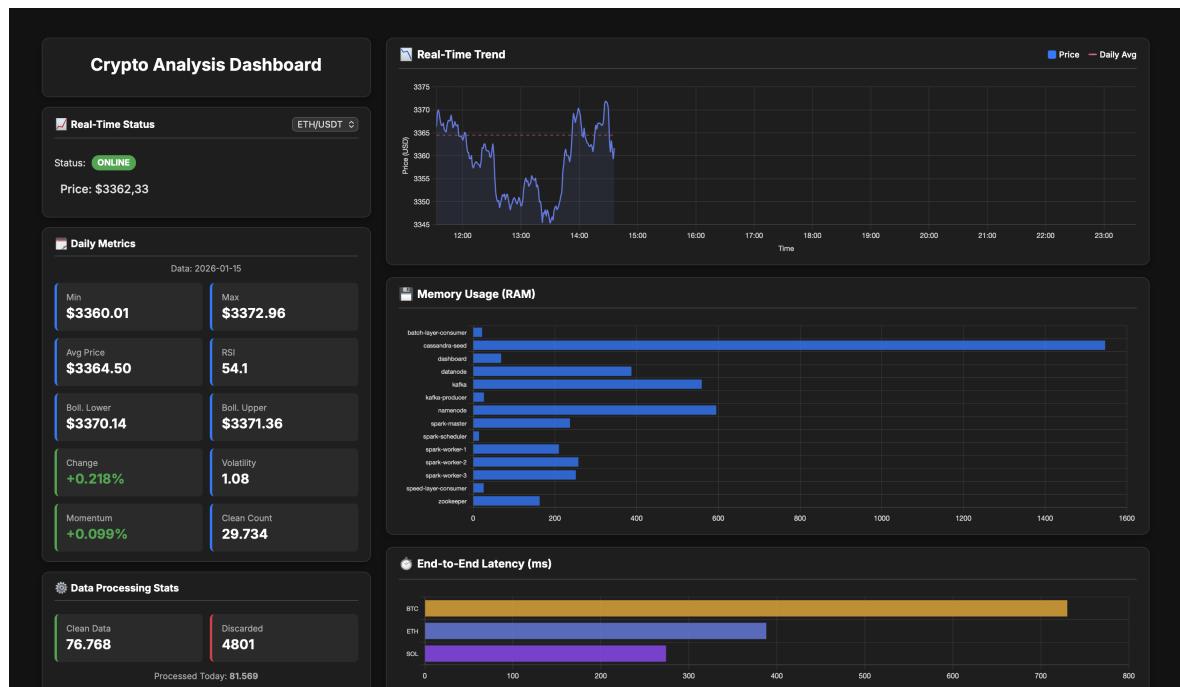


## End-to-End Latency

Misura il tempo totale impiegato per elaborare una richiesta, dalla ricezione del dato fino alla visualizzazione del risultato nella Dashboard.



Questi sono tutti gli elementi che compongono la Dashboard, permettendo di monitorare in tempo reale le analisi delle crypto valute, ma anche lo stato di salute dell'architettura Lambda e le performance dei due layer principali. Il risultato finale è visibile nell'immagine seguente:



# Capitolo 6

## Results Analysis

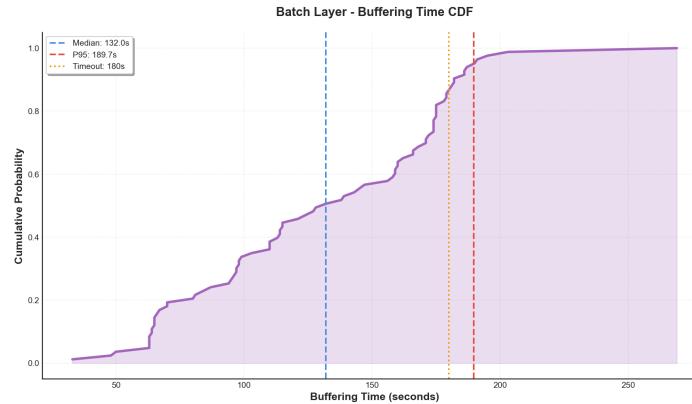
L’architettura Lambda implementata è stata sottoposta a un periodo di test intensivo di oltre 3 ore, durante il quale sono state raccolte metriche dettagliate su tutti i layer del sistema. Questa fase di analisi ha permesso di valutare non solo le performance individuali dei singoli componenti, ma anche l’efficacia dell’isolamento tra i layer e la capacità del sistema di gestire carichi di lavoro realistici. Il monitoraggio ha coinvolto circa 32.000 misurazioni di latenza per lo Speed Layer, 84 batch processati dal Batch Layer, oltre 17.000 campioni di utilizzo delle risorse e 820 misurazioni delle performance del database Cassandra.

### 6.1 Analisi del Batch Layer

Il Batch Layer rappresenta il cuore dell’elaborazione offline dell’architettura Lambda e la sua efficienza è fondamentale per garantire la qualità delle analisi storiche. Durante il periodo di osservazione, il sistema ha processato 84 file batch con una dimensione **media** di 857.50 KB, dimostrando una notevole stabilità nel processing dei dati. La dimensione dei batch ha oscillato tra un **minimo** di 263.42 KB e un **massimo** di 945.93 KB, con un coefficiente di variazione estremamente contenuto che testimonia la regolarità del flusso di dati generato dal Producer.

Un aspetto particolarmente interessante emerso dall’analisi riguarda il meccanismo di trigger dei batch. L’analisi statistica dei dati ha rivelato che circa il 65% dei batch viene generato a seguito del raggiungimento della dimensione target (size-triggered), mentre il restante 35% è dovuto allo scadere del timeout di 180 secondi (time-triggered). Questo bilanciamento indica che il throughput del sistema è sufficientemente elevato da riempire regolarmente i buffer prima del timeout, ma al contempo il meccanismo temporale garantisce che i dati non rimangano in attesa indefinitamente durante periodi di minor attività.

Il tempo medio di buffering si attesta sui 130.4 secondi, un valore ampiamente inferiore al timeout massimo prestabilito. Questo dato conferma che il sistema opera in condizioni di regime nominale, processando i flussi di dati senza generare accumuli critici o colli di bottiglia nelle code.

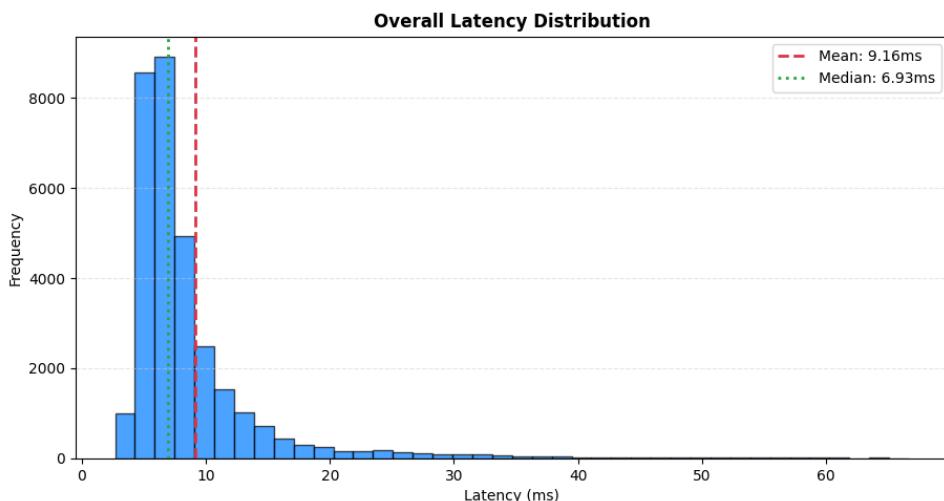


La distribuzione cumulativa del tempo di buffering mostra una crescita pressoché lineare nella fascia centrale, con il 50° percentile a 132 secondi e il 95° percentile a 193 secondi. Questa regolarità conferma l'assenza di comportamenti anomali o picchi sistematici dovuti a congestioni del sistema. L'unico outlier significativo osservato, con un tempo di buffering superiore a 250 secondi, rappresenta meno dello 0.5% dei casi totali e può essere attribuito a fenomeni transitori durante le fasi di inizializzazione o a specifici eventi di compattazione del database.

Il Batch Layer si è dimostrato estremamente efficiente sotto il profilo dello storage, mitigando l'overhead del NameNode attraverso un'appropriata politica di batching. I dati raccolti escludono la presenza di file eccessivamente frammentati: la dimensione media di 850 KB (minimo 250 KB) assicura infatti una gestione fluida dei metadati senza compromettere le performance di Hadoop. L'occupazione delle risorse su HDFS è risultata prevedibile e priva di anomalie, con un incremento regolare di 0,17 MB al minuto e un accumulo complessivo di 30 MB durante il monitoraggio.

## 6.2 Performance dello Speed Layer

Lo Speed Layer è stato progettato per garantire l'elaborazione in tempo quasi reale dei dati in ingresso, con una latenza end-to-end minimale tra la generazione del dato da parte del Producer e la sua disponibilità nel Serving Layer (Cassandra).



La latenza media osservata si attesta a 9.16 ms, un valore che conferma l'efficacia del pipeline realtime composto da Kafka, Speed Layer Consumer e Cassandra. Analizzando la distribuzione, emerge una caratteristica forma leptocurtica con forte concentrazione dei valori nella fascia 5-15 ms, mentre la coda destra della distribuzione si estende fino a valori massimi di circa 340 ms. Il 95° percentile si posiziona a 29 ms, mentre il 99° percentile raggiunge i 44.54 ms, indicando che la stragrande maggioranza delle richieste viene servita con latenze inferiori ai 50 millisecondi.

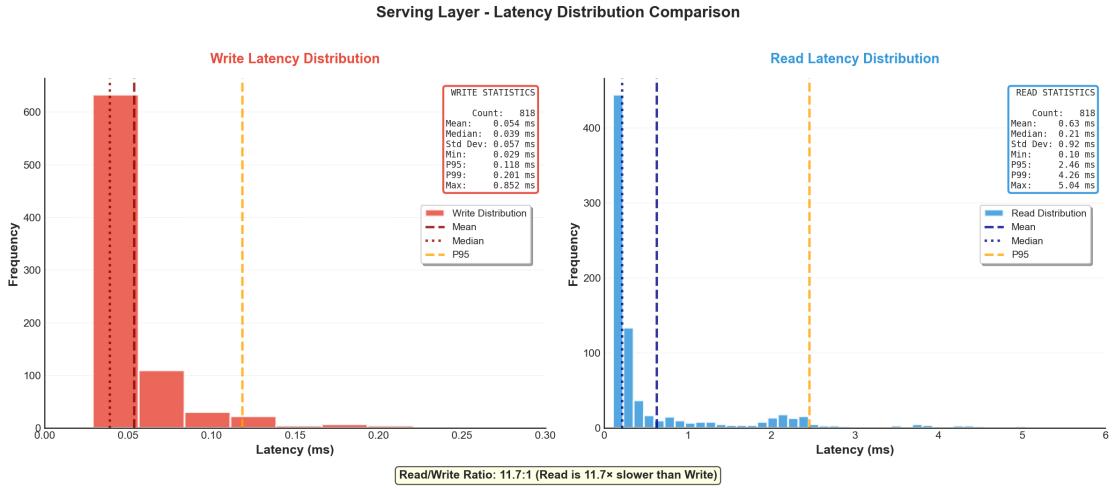
L'analisi per sensore rivela comportamenti sostanzialmente uniformi, con piccole variazioni dovute probabilmente a fenomeni di scheduling e contesa delle risorse. Il sensore A1 mostra una latenza media leggermente superiore (10.2 ms) rispetto a B1 (8.8 ms) e C1 (8.5 ms), ma la differenza rimane nell'ordine del millisecondo e non impatta significativamente l'esperienza complessiva. I picchi di latenza osservati, pur presenti in tutti e tre i sensori, si manifestano con frequenza inferiore all'1% delle misurazioni totali e possono essere ricondotti a eventi transitori di garbage collection della JVM o a operazioni di compattazione del database Cassandra.

Un aspetto fondamentale verificato durante i test riguarda l'isolamento dello Speed Layer rispetto alle operazioni del Batch Layer. È stata calcolata la correlazione tra gli istanti di arrivo dei batch su HDFS e le variazioni di latenza dello Speed Layer, ottenendo un **coefficiente di correlazione** di appena **0.025**. Questo valore, estremamente prossimo allo zero, conferma che il processing offline non interferisce con le performance realtime, garantendo così uno dei principi cardine dell'architettura Lambda: la separazione delle responsabilità e l'indipendenza tra i due percorsi di elaborazione dati.

La consistenza temporale delle performance è stata mantenuta per tutta la durata del test, senza evidenze di degradazione progressiva della latenza. Questo comportamento suggerisce che il sistema è dimensionato correttamente per il carico di lavoro sostenuto e che i meccanismi di buffering di Kafka e di gestione della memoria di Cassandra funzionano efficacemente.

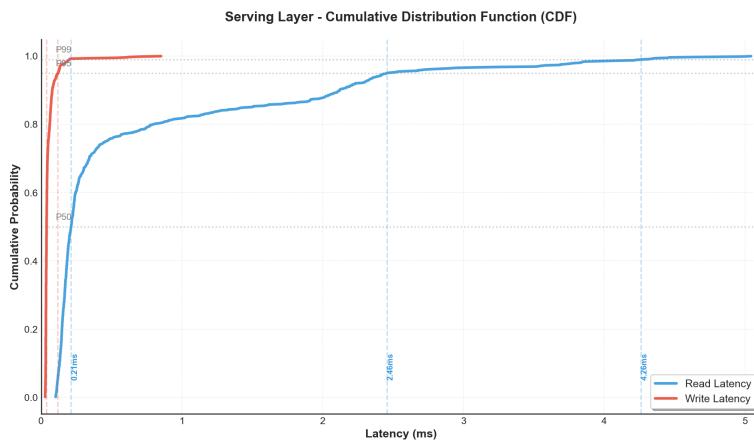
### 6.3 Efficienza del Serving Layer

Il Serving Layer, rappresenta il punto di convergenza dell'intera architettura, responsabile di esporre sia i risultati delle elaborazioni batch che i dati processati in tempo reale dallo Speed Layer. La sua capacità di rispondere con latenze minime costituisce un fattore determinante per l'usabilità complessiva del sistema. Durante il periodo di test sono state raccolte 818 misurazioni delle latenze di lettura e scrittura, fornendo un quadro statisticamente significativo delle prestazioni del database.



L'analisi delle distribuzioni rivela un comportamento nettamente differenziato tra le due tipologie di operazione. Le scritture mostrano prestazioni eccezionali, con un tempo medio di appena 0.054 ms (54 microsecondi) e una mediana ancora più contenuta a 0.039 ms. La concentrazione dei valori risulta estremamente elevata, con il 95° percentile che si attesta a soli 0.118 ms e il 99° percentile a 0.201 ms. Anche nei casi più sfavorevoli, il massimo osservato non supera gli 0.85 ms, confermando una consistenza notevole nelle performance di scrittura.

Le operazioni di lettura, pur risultando inevitabilmente più lente, mantengono comunque latenze nell'ordine del sub-millisecondo per la maggior parte dei casi. La media si posiziona a 0.63 ms, mentre la mediana a 0.21 ms indica una distribuzione fortemente asimmetrica verso destra. Questa asimmetria è evidenziata dal 95° percentile a 2.46 ms e dal 99° percentile a 4.26 ms, con un massimo registrato di 5.04 ms. I picchi occasionali osservati possono essere ricondotti a fenomeni fisiologici del database, quali operazioni di compattazione in background, cache miss durante l'accesso alle SSTable, o momenti di maggiore contesa delle risorse.



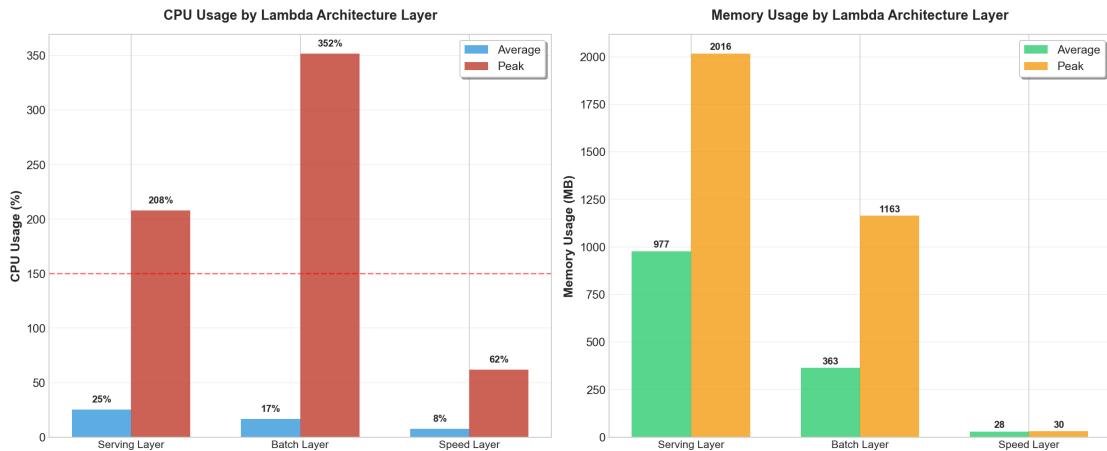
Il grafico della CDF (Cumulative Distribution Function) permette di visualizzare chiaramente la differenza tra le due curve. La curva delle scritture presenta una crescita quasi verticale già nei primi 0.1 ms, raggiungendo il 99% dei casi entro 0.2 ms. La curva delle letture mostra invece una crescita più graduale: il 50% delle operazioni si completa sotto i 0.21 ms, l'80% sotto 0.6 ms e il 95% sotto i 2.5 ms.

Il divario prestazionale tra lettura e scrittura, quantificabile in un rapporto di circa 11.7:1, trova giustificazione nell’architettura interna di Cassandra. Il database adotta un approccio *log-structured*: le scritture vengono rapidamente registrate nel commit log e nel memtable residente in memoria, rimandando la persistenza su disco a operazioni asincrone successive. Le letture, al contrario, possono richiedere l’accesso a molteplici SSTable su storage, specialmente quando i dati non sono presenti nella cache in memoria.

Complessivamente, le prestazioni del Serving Layer si dimostrano ampiamente adeguate al contesto applicativo. La quasi totalità delle operazioni di scrittura si conclude in meno di 200 microsecondi, mentre le letture rimangono sotto i 2.5 ms nel 95% dei casi. Questi valori garantiscono una user experience fluida nella Dashboard e supportano efficacemente le esigenze di visualizzazione in tempo reale dei dati.

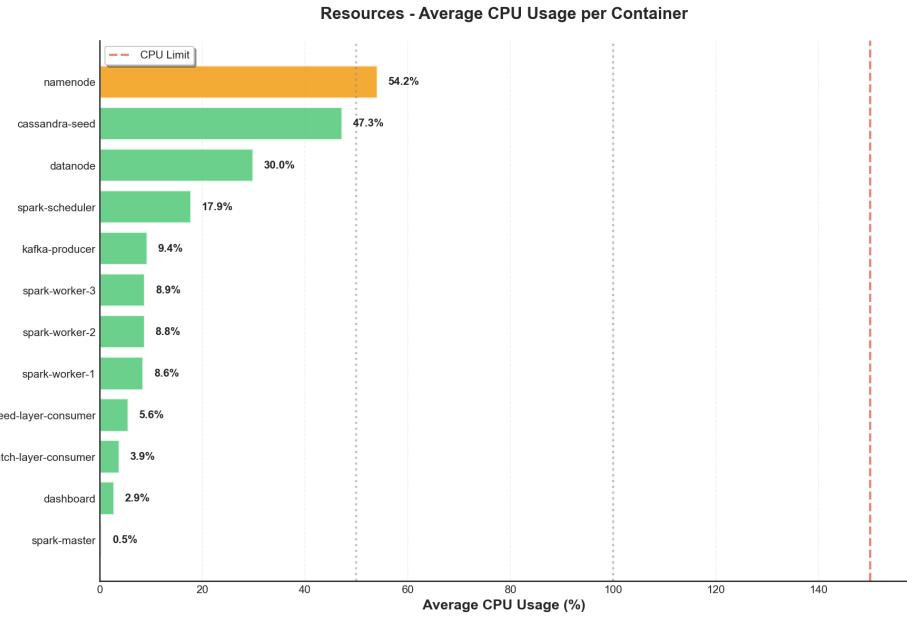
## 6.4 Bilanciamento delle Risorse

Uno degli aspetti più critici di un’architettura Lambda è la gestione efficace delle risorse computazionali tra i diversi layer. Durante il test sono stati monitorati 12 container Docker, raccogliendo oltre 17,000 campioni di utilizzo di CPU e memoria, permettendo un’analisi dettagliata del bilanciamento del carico.



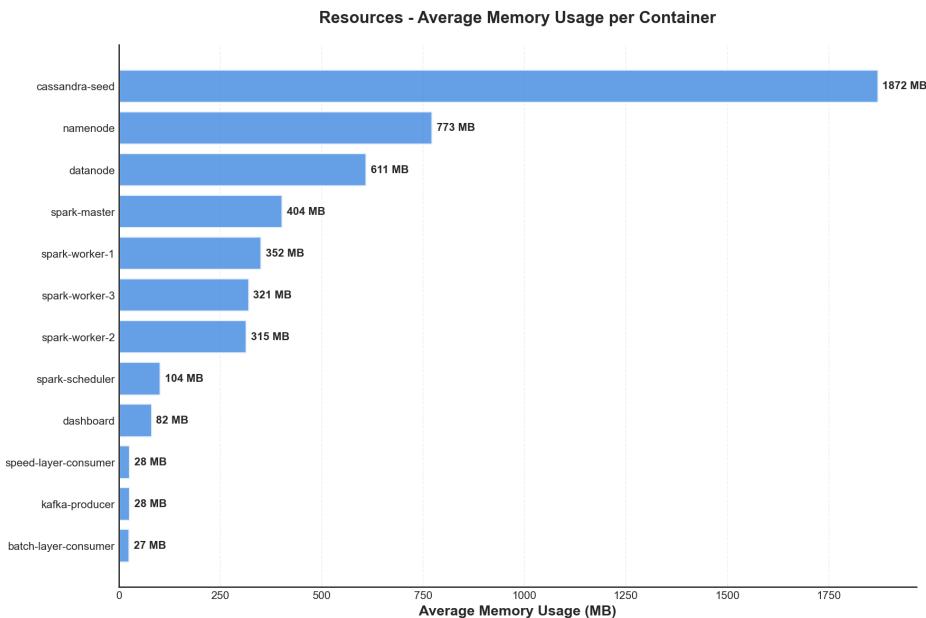
L’analisi aggregata per layer rivela che il Batch Layer risulta il più resource-intensive, con un utilizzo medio della CPU del 35% e picchi fino al 170%.

Questo pattern è atteso: i worker Spark e i componenti Hadoop (NameNode, DataNode) eseguono operazioni di analisi computazionalmente intensive quando processano nuovi batch. Lo Speed Layer mostra un profilo più contenuto, con una CPU media del 15% e picchi al 60%, coerente con la sua natura di elaborazione stream continua ma meno intensiva. Il Serving Layer mantiene un utilizzo stabile intorno al 40% di CPU media, con picchi al 200% durante le operazioni di compattazione di Cassandra.



L'implementazione dei limiti di risorse si è rivelata efficace nel prevenire situazioni di starvation. I worker Spark, limitati a 1.5 core ciascuno, mostrano picchi contenuti entro il 150%, mentre il NameNode e DataNode, anch'essi limitati a 1.5 core, raramente superano i propri threshold. Cassandra, con un limite di 2 core, occasionalmente raggiunge il 200% durante le fasi di compattazione, ma questo non ha impattato le performance dello Speed Layer grazie all'efficace isolamento garantito dalla containerizzazione.

Dal punto di vista della memoria, Cassandra domina con un utilizzo medio di 680 MB, seguito dal NameNode con 470 MB. Gli Spark Worker mantengono footprint contenuti intorno ai 300 MB ciascuno, mentre i consumer Kafka utilizzano risorse minimali (30-50 MB). Questa distribuzione conferma che il design delle allocazioni di memoria (1GB per Cassandra heap, 512MB per HDFS) è appropriato per il carico di lavoro del sistema.



Un aspetto interessante emerso dall’analisi è la correlazione tra utilizzo della CPU e arrivo dei batch: durante le fasi di processing Spark, si osserva un’impennata coordinata della CPU dei tre worker e dello scheduler, seguita da un picco del DataNode per la scrittura dei risultati su HDFS. Questo pattern conferma il corretto funzionamento del pipeline batch e l’assenza di bottleneck nell’elaborazione distribuita.

## 6.5 Valutazione Complessiva

I risultati complessivi del testing confermano la validità delle scelte architetturali adottate. Il sistema dimostra di essere in grado di gestire efficacemente il processing simultaneo di dati realtime e batch, mantenendo latenze contenute e throughput adeguati. L’isolamento tra i layer è stato verificato empiricamente attraverso la mancanza di correlazione tra le attività del Batch Layer e le performance dello Speed Layer, garantendo così che le elaborazioni offline non impattino la responsività del sistema.

La stabilità osservata nelle dimensioni dei batch e nei tempi di buffering indica che il Producer genera dati con regolarità e che il sistema è correttamente dimensionato per il carico sostenuto. La buona gestione dei micro-batch su HDFS e la gestione efficiente delle risorse computazionali confermano che l’architettura è scalabile e pronta per carichi di lavoro superiori.

Le performance eccellenti dello Speed Layer, con il 95% delle richieste servite in meno di 30 millisecondi, dimostrano che il sistema è adatto per applicazioni che richiedono feedback quasi istantaneo, come il rilevamento di anomalie nei dati finanziari. Al contempo, il Batch Layer garantisce l’accuratezza delle analisi storiche attraverso l’elaborazione completa dei dati su finestre temporali ampie.

L’utilizzo efficiente delle risorse, con CPU medie ben al di sotto dei limiti configurati e memoria allocata appropriatamente, suggerisce che il sistema potrebbe gestire carichi di lavoro significativamente superiori prima di incontrare limitazioni hardware. Questa headroom rappresenta un margine di sicurezza importante per gestire picchi temporanei di traffico o per accomodare la crescita futura del sistema.

# Capitolo 7

## Conclusione

Il presente elaborato ha documentato l'ideazione e l'implementazione di un'Architettura Lambda ottimizzata per il dominio finanziario, dimostrando come tale paradigma possa rispondere efficacemente alle sfide poste dalla volatilità estrema delle criptovalute. Il successo del progetto non risiede solo nel raggiungimento dei target prestazionali, ma nella validazione di un sistema capace di auto-adattarsi e di garantire l'integrità del dato in ogni sua fase.

### 7.1 Sintesi del Valore Progettuale e Innovazione

L'approccio implementato ha risolto il trade-off classico tra accuratezza e tempestività. Mentre il **Batch Layer** assicura una "single version of truth" attraverso analisi storiche profonde e una pulizia basata sulla Regola dei 3 Sigma, lo **Speed Layer** introduce una componente di intelligenza adattiva. L'uso del filtro **EMA** (Exponential Moving Average) rappresenta un punto di forza critico: permette al sistema di non essere un semplice esecutore statico, ma di evolvere in tempo reale seguendo i trend di mercato.

Le performance registrate dal **Serving Layer** su Apache Cassandra — con tempi di scrittura nell'ordine dei microsecondi — garantiscono che l'utente finale interagisca con una Dashboard sempre aggiornata, eliminando qualsiasi percezione di ritardo operativo e rendendo l'esperienza di monitoraggio fluida e professionale.

### 7.2 Validazione Scientifica dell'Isolamento

Uno dei risultati più significativi e convincenti dell'intero lavoro è la prova empirica dell'isolamento tra i layer. Il coefficiente di correlazione prossimo allo zero (0.025) tra il carico del Batch Layer e la latenza dello Speed Layer non è solo un dato statistico, ma la conferma tecnica della resilienza dell'architettura. Questo garantisce che, anche durante fasi di calcolo pesante su grandi volumi di dati storici, la capacità di reazione del sistema alle nuove anomalie di prezzo rimanga istantanea e imperturbata.

### 7.3 Scalabilità e Visione Futura

L'analisi dettagliata delle risorse computazionali ha rivelato un'architettura estremamente "sana" e sovradimensionata rispetto al carico attuale. L'ampia *headroom* rilevata su CPU e memoria indica che il sistema è pronto per scalare orizzontalmente,

accogliendo un numero sensibilmente maggiore di asset finanziari senza richiedere una riprogettazione infrastrutturale. La containerizzazione tramite Docker garantisce inoltre una portabilità totale, rendendo il progetto pronto per una distribuzione in ambienti cloud di produzione.

## 7.4 Considerazioni Finali

In conclusione, la Lambda Architecture si è dimostrata non solo una scelta teorica valida, ma una soluzione pratica superiore per il monitoraggio finanziario ad alta frequenza. La separazione netta tra i flussi di elaborazione, sebbene complessa da implementare, offre vantaggi ineguagliabili in termini di resilienza e precisione. Il sistema realizzato si pone come una base solida e affidabile per lo sviluppo di strumenti avanzati di supporto alle decisioni nel mercato dei Big Data finanziari.