UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

VolleyLive

Sistema di monitoraggio, analisi e visualizzazione
delle partite di pallavolo in tempo reale

Studente:
Claudio Nuncibello

Relatore:
Prof. Salvatore Nicotra

Anno Accademico 2024/2025

Abstract

Il progetto VolleyLive nasce con l'obiettivo di monitorare e analizzare in tempo reale le partite di pallavolo, combinando tecnologie moderne per la gestione dei dati in streaming e la visualizzazione interattiva. Il sistema raccoglie snapshot live da API sportive, li trasforma e li distribuisce tramite una pipeline composta da Kafka, Logstash, Spark ed Elasticsearch. I dati vengono visualizzati in una dashboard frontend sviluppata in Next.js, offrendo funzionalità come la selezione dei match preferiti e l'analisi predittiva dell'esito delle partite. L'intero progetto è containerizzato tramite Docker e supporta l'estensibilità verso nuovi modelli e sport. Questa tesi descrive l'architettura, le scelte progettuali, e i risultati ottenuti attraverso l'implementazione del sistema.

Indice

Introduzione

Il presente elaborato descrive lo sviluppo di VolleyLive: un sistema per il monitoraggio e l'analisi in tempo reale delle partite di pallavolo. Il progetto nasce nell'ambito dell'analisi sportiva, oggi profonda trasformazione, favorita dalla crescente disponibilità di dati e dall'utilizzo di tecnologie per l'elaborazione in streaming. L'integrazione di API sportive, strumenti di monitoraggio e algoritmi analitici consente oggi non solo di osservare lo svolgimento di una competizione, ma anche di interpretarne dinamicamente l'andamento e prevederne l'evoluzione. L'analisi in tempo reale ha assunto un ruolo centrale in numerosi ambiti applicativi, dallo sport professionale alla logistica industriale. Nel mondo sportivo tuttavia strumenti avanzati di elaborazione live sono spesso riservati a discipline molto seguite (come calcio o basket), o a contesti professionali ad alto budget. Per molte altre discipline come la pallavolo esistono margini significativi per migliorare l'accessibilità e la fruibilità dei dati durante gli eventi. VolleyLive non si propone soltanto come una soluzione funzionale per il monitoraggio in tempo reale, ma getta le basi per un sistema estensibile e orientato al futuro delle competizioni sportive. Grazie all'adozione di tecnologie scalabili e open source, la piattaforma si presta a essere ampliata per integrare nuove fonti dati e modelli predittivi sempre più sofisticati. In questo senso, VolleyLive rappresenta una prima esplorazione concreta verso sistemi intelligenti di supporto all'analisi sportiva, potenzialmente applicabili non solo alla pallavolo ma a molte altre discipline. Il sistema adotta tecnologie all'avanguardia come Kafka per la gestione del flusso dati, Logstash per la trasformazione degli eventi, Spark per l'elaborazione in tempo reale, Elasticsearch per l'indicizzazione e la consultazione, e una web app per la visualizzazione lato utente. Il tutto è orchestrato tramite container Docker, in modo da garantire portabilità, isolamento e semplicità di gestione. Gli obiettivi principali del progetto sono: costruire una pipeline dati solida e scalabile, in grado di gestire flussi continui di snapshot aggiornati ogni dieci secondi; applicare modelli di analisi per stimare in tempo reale l'esito delle partite in corso; offrire un'interfaccia intuitiva e interattiva per la consultazione live dei match; garantire la tracciabilità storica dei dati raccolti, utile per analisi retrospettive e sviluppo futuro di modelli predittivi più avanzati. Dal punto di vista applicativo, VolleyLive è pensato per essere utilizzato sia da utenti generici interessati a seguire una competizione sia da figure più tecniche come data analyst sportivi o scout, che necessitano di uno strumento semplice ma flessibile per leggere, filtrare e confrontare partite in tempo reale. L'elaborato è articolata in sette capitoli. Nel capitolo primo capitolo si descriverà l'architettura generale del sistema e i componenti principali, soffermandosi sulle tecnologie adottate e sul loro ruolo nella pipeline. Il secondo capitolo approfondirà il funzionamento della raccolta dati live tramite API e il terzo illustra i meccanismi di ingestione, trasformazione ed elaborazione in streaming. Il quarto capitolo è dedicato alla visualizzazione e all'interazione tramite dashboard web. Il quinto capitolo presenta una serie di risultati ottenuti attraverso l'esecuzione del sistema, evidenziando esempi pratici e scenari d'uso. nelle conclusioni si raccolgono le riflessioni conclusive e suggerisce direzioni per futuri sviluppi ed estensioni. VolleyLive si propone quindi come una base concreta per sperimentare tecnologie moderne per lo streaming e l'analisi dei dati sportivi, ponendo particolare attenzione all'affidabilità del flusso, alla chiarezza della visualizzazione e alla possibilità di estensione verso altri sport o contesti analitici.

Architettura Generale del Sistema

Il sistema VolleyLive è stato concepito e realizzato seguendo un approccio modulare e scalabile, con l'obiettivo di garantire affidabilità, continuità del flusso dati e facilità di estensione. Questo capitolo descrive nel dettaglio l'architettura generale adottata, soffermandosi su ciascuna delle tecnologie impiegate, sul ruolo che ricoprono nella pipeline e sulle relazioni tra i vari moduli.

1.1 Panoramica dei Componenti Tecnologici

In questa sezione si fornisce una descrizione dettagliata dei principali componenti tecnologici che costituiscono il sistema VolleyLive, evidenziando il ruolo operativo di ciascun modulo all'interno della pipeline. Ogni tecnologia è presentata nel contesto del suo impiego pratico, illustrandone le funzionalità chiave e le modalità con cui interagisce con gli altri elementi del sistema.

1.1.1 Fonte Dati: SportDevs API

Il sistema VolleyLive si basa sui dati forniti dalle API pubbliche di SportDevs, che rappresentano la fonte principale per l'acquisizione delle informazioni relative alle partite in corso. Queste API espongono una serie di endpoint REST, interrogabili in formato `JSON`, attraverso cui è possibile ottenere dati grezzi aggiornati sulle partite live, dai quali il sistema costruisce, tramite successive trasformazioni, snapshot strutturati e coerenti, idonei a essere impiegati nei modelli di previsione.

Nel flusso di sistema, le API costituiscono il punto iniziale della pipeline: i dati vengono estratti regolarmente dallo script *producer*, che li interroga ogni dieci secondi. I dati ricevuti vengono normalizzati e strutturati per essere inoltrati a Kafka nel formato previsto. L'accesso avviene tramite chiave API e richiede attenzione nella gestione della frequenza di interrogazione per evitare di superare i limiti imposti dal servizio.

Nonostante l'ampia copertura e il buon livello di dettaglio, le API presentano alcune limitazioni. In particolare, durante le partite di campionati minori si riscontrano occasionali ritardi nell'aggiornamento dei punteggi, e la gestione del set di *tiebreak* può risultare imprecisa o mancante. Queste problematiche sono note e, al momento, vengono gestite a livello applicativo convivendo con le possibili incongruenze nel dato.

Nel complesso, SportDevs rappresenta una soluzione efficace per alimentare in tempo reale il sistema VolleyLive, pur richiedendo l'adozione di logiche resilienti in fase di raccolta e validazione dei dati.

1.1.2 Elaborazione e Normalizzazione: Logstash

Logstash è il primo componente della pipeline incaricato della trasformazione dei dati in arrivo dalle API di VolleyLive. Riceve direttamente i dati grezzi forniti dallo script Python (producer), li interpreta e li converte in un formato strutturato, adatto alla successiva pubblicazione su Kafka.

Le operazioni svolte da Logstash includono la normalizzazione dei nomi dei campi, l'estrazione di *timestamp* coerenti, la gestione di dati opzionali o incoerenti, e l'aggiunta di metadati utili (come gli identificativi univoci del match). Queste trasformazioni avvengono tramite pipeline configurabili in formato dichiarativo, che permettono di adattare con facilità le regole di parsing e filtraggio.

Una volta completata l'elaborazione, Logstash pubblica gli *snapshot* trasformati su un *topic* Kafka, rendendoli disponibili per le successive fasi di analisi e indicizzazione. In questo modo, Logstash funge da punto di ingresso intelligente per il flusso dati del sistema, garantendo uniformità e qualità nella base informativa di VolleyLive.

1.1.3 Gestione del Flusso Dati: Kafka

Apache Kafka è il sistema di messaggistica distribuito adottato da VolleyLive per la gestione affidabile e scalabile del flusso dati in tempo reale. All'interno dell'architettura, Kafka riceve i dati già trasformati da Logstash e li rende disponibili per i successivi moduli di analisi e indicizzazione.

Ogni snapshot elaborato viene pubblicato su un *topic* Kafka, da cui può essere consumato in parallelo da più servizi, come Apache Spark o eventuali altri moduli analitici. Questo modello basato su pubblicazione e sottoscrizione (*pub/sub*) consente un'elaborazione asincrona, modulare e facilmente estendibile.

La scelta di utilizzare Kafka è stata motivata dalla necessità di gestire un flusso continuo di eventi con frequenti aggiornamenti, mantenendo elevate garanzie di durabilità, ordinamento e possibilità di *replay* dei messaggi. La sua architettura nativamente distribuita lo rende particolarmente adatto a scenari *real-time* come quello di VolleyLive.

1.1.4 Analisi Streaming: Apache Spark

Apache Spark è il componente dedicato all'elaborazione avanzata dei dati in streaming nel sistema VolleyLive. Dopo che gli *snapshot* sono stati pubblicati su Kafka da Logstash, Spark li consuma in tempo reale per applicare logiche analitiche e generare output destinati all'indicizzazione.

L'implementazione sfrutta Spark Structured Streaming, un sistema che elabora i dati in tempo reale suddividendoli in piccoli blocchi temporali (*micro-batch*). Anche se i dati arrivano in modo continuo, Spark li raggruppa ogni pochi secondi e li processa in modo ordinato e affidabile. Questo approccio garantisce una *semantica end-to-end*, ovvero l'elaborazione completa, senza perdite o duplicazioni, di ogni blocco di dati dall'ingresso all'output finale.

Nel contesto di VolleyLive, Spark viene utilizzato per applicare filtri, trasformazioni strutturali e, in prospettiva, funzioni di analisi predittiva basate su modelli di *machine learning*.

Spark si colloca tra la distribuzione dei dati (Kafka) e la fase di visualizzazione (Elasticsearch), fungendo da nodo di elaborazione intermedio. Questa configurazione permette di isolare la logica analitica dal resto della pipeline, semplificando l'estensione futura del sistema verso algoritmi più sofisticati.

1.1.5 Indicizzazione e Ricerca: Elasticsearch

Elasticsearch è il motore di ricerca e indicizzazione adottato nel sistema VolleyLive per rendere interrogabili e visualizzabili gli *snapshot* elaborati in tempo reale. Dopo l'elaborazione su Apache Spark, ogni snapshot viene inviato a Elasticsearch, dove viene memorizzato come documento JSON strutturato.

Tutti i dati, inclusi gli score predittivi e le informazioni live, vengono inseriti all'interno di un unico indice. Questa scelta progettuale consente di ottenere, tramite una singola query,

tutte le informazioni aggiornate relative a un match, semplificando l'accesso lato frontend e migliorando l'efficienza delle chiamate.

L'indice è configurato con un *mapping* coerente, che definisce esplicitamente la struttura e i tipi dei campi per garantire interrogazioni affidabili e prestazioni ottimali. Questo approccio assicura una base dati consistente e facilmente estendibile anche in fase di evoluzione del sistema.

Infine, per attività di *debugging*, monitoraggio e ispezione manuale dei dati, il sistema fa uso di Kibana, uno strumento di visualizzazione integrato con Elasticsearch, utile in particolare durante le fasi di sviluppo.

1.1.6 Visualizzazione: Frontend React/Next.js

L'interfaccia utente di VolleyLive è sviluppata utilizzando il framework React in combinazione con Next.js, una soluzione moderna e performante per la realizzazione di applicazioni web dinamiche. Il frontend fornisce agli utenti una dashboard interattiva e reattiva per consultare lo stato in tempo reale delle partite, visualizzare punteggi, set e score predittivi.

Per accedere ai dati, il frontend comunica con un backend intermedio realizzato in FastAPI. Questo backend funge da strato di accesso alle informazioni archiviate in Elasticsearch, esponendo API REST personalizzate per effettuare query sui match live. Tale approccio consente di mantenere il database isolato, migliorare la sicurezza e semplificare la gestione delle richieste dal lato client.

La struttura della dashboard permette agli utenti di selezionare match preferiti live e visualizzare in dettaglio l'andamento di ciascun incontro. L'interfaccia è stata progettata per adattarsi ai diversi dispositivi.

Il frontend è containerizzato e integrato nell'infrastruttura tramite Docker, in modo da garantire coerenza tra ambiente di sviluppo e produzione e facilitare il deployment del sistema completo.

1.1.7 Containerizzazione: Docker e Visualizzazione dell'Architettura

Il sistema VolleyLive adotta una configurazione interamente containerizzata, implementata tramite Docker, al fine di garantire portabilità, isolamento e semplicità di gestione in tutte le fasi del ciclo di vita dell'applicazione. Ogni componente è incapsulato in un container indipendente, facilitando la manutenzione, il testing e il deployment multiambiente.

L'infrastruttura è orchestrata mediante `docker-compose`, che consente di avviare l'intero ecosistema tramite un'unica configurazione centralizzata. I servizi comunicano all'interno di una rete virtuale definita da Docker, riducendo le dipendenze esterne e i problemi di configurazione ambientale.

Tutti i moduli descritti nei paragrafi precedenti – inclusi il *producer* Python, i servizi di elaborazione e analisi, l'API backend e l'interfaccia web – sono containerizzati e integrati nell'infrastruttura, assicurando coerenza tra ambiente di sviluppo e produzione.

Lo schema architetturale riportato in Figura 1.1 illustra il flusso dei dati e le interazioni principali tra i servizi, dalla raccolta degli snapshot fino alla visualizzazione finale.

1.2 Scelte Progettuali e Motivazioni

La definizione dell'architettura di VolleyLive è frutto di un processo decisionale che ha bilanciato esigenze di efficienza, semplicità operativa e finalità analitiche. In questa sezione vengono descritte le principali alternative valutate e le motivazioni che hanno portato alla definizione dell'architettura presentata nel paragrafo precedente.

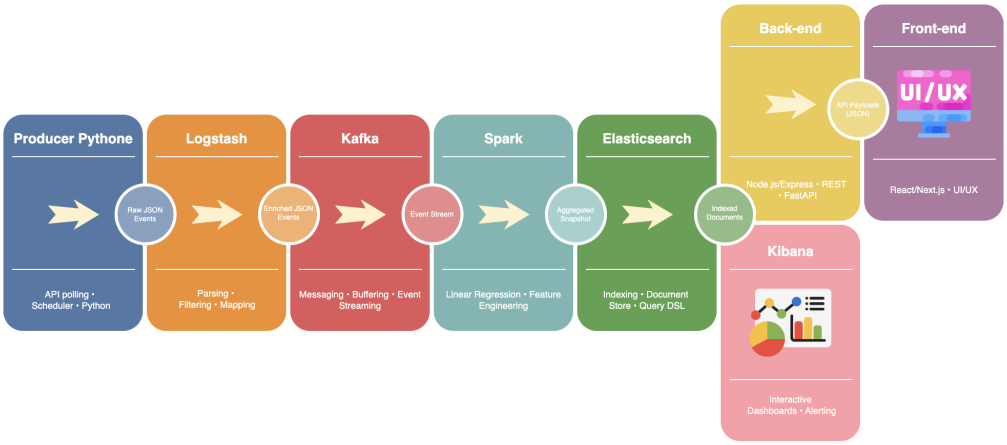


Figura 1.1: Architettura containerizzata del sistema VolleyLive

1.2.1 Pipeline Always-On vs On-Demand: Analisi Comparativa

Nel progettare il sistema VolleyLive, si è valutata la possibilità di attivare la pipeline dati solo in risposta a un'azione dell'utente (approccio *on-demand*) oppure di mantenerla costantemente attiva (approccio *always-on*). La modalità *on-demand*, sebbene più leggera dal punto di vista computazionale, avrebbe comportato una latenza iniziale, una complessità maggiore nel coordinamento tra moduli e una perdita della continuità storica. L'approccio *always-on* garantisce invece un flusso costante e affidabile di dati, semplificando il design complessivo e consentendo l'elaborazione in tempo reale anche in assenza di richieste attive.

1.2.2 Motivazioni della Scelta dell'Architettura Always-On

L'adozione dell'architettura *always-on* è stata guidata da due esigenze fondamentali: da un lato, garantire l'affidabilità e la continuità del flusso dati; dall'altro, costruire una base dati storica che consenta analisi retrospettive e l'applicazione di modelli predittivi. Inoltre, la natura del progetto – focalizzato sull'utilizzo di tecnologie di data streaming e containerizzazione – trova piena espressione in un sistema sempre attivo, che riflette in modo coerente gli obiettivi sperimentali della tesi.

Acquisizione e Gestione dei Dati in Real-Time

In questo capitolo viene descritta la prima parte della pipeline utilizzata da VolleyLive, dedicata alla raccolta e alla gestione iniziale dei dati provenienti da partite di pallavolo in corso. Il focus sarà sulle tecnologie e le metodologie impiegate per acquisire dati live, trasformarli in un formato standardizzato e distribuirli attraverso un flusso dati affidabile e scalabile.

I paragrafi di questo capitolo affrontano in maniera approfondita ciascuno dei passaggi fondamentali: dall'acquisizione iniziale tramite script Python, alla normalizzazione effettuata da Logstash, fino alla distribuzione tramite Kafka. Ogni fase sarà illustrata nel dettaglio, sottolineando le scelte tecniche effettuate e le loro motivazioni.

2.1 Acquisizione Dati: Producer Python

2.1.1 Introduzione agli Snapshot Live

Nella pipeline dati del progetto VolleyLive, lo snapshot live rappresenta il punto iniziale e fondamentale per ogni successiva elaborazione ed analisi. Per snapshot si intende una fotografia periodica e dettagliata dello stato corrente di ciascuna partita monitorata. Tali istantanee sono catturate a intervalli temporali regolari (ogni pochi secondi), al fine di garantire una visione costantemente aggiornata degli eventi sportivi in corso.

L'impiego degli snapshot live si rivela cruciale poiché essi costituiscono non solo la base per l'analisi in tempo reale, ma anche una risorsa fondamentale per alimentare il dataset storico, contribuendo così a migliorare nel tempo l'accuratezza delle predizioni effettuate dal sistema.

2.1.2 API SportDevs: Dati e Endpoint Utilizzati

La raccolta dei dati live viene effettuata interrogando direttamente le API pubbliche di SportDevs, che permettono di accedere a informazioni aggiornate in tempo reale sulle partite di pallavolo. Queste API restituiscono dati in formato JSON, facilmente integrabili all'interno del sistema VolleyLive. Il producer Python utilizza principalmente tre endpoint. Il primo è:

`https://volleyball.sportdevs.com/matches?status_type=eq.live`

Questo endpoint restituisce tutte le partite attualmente in corso e fornisce i dati essenziali per comporre uno snapshot live, come il punteggio per ciascun set, lo stato della partita, gli ID delle squadre coinvolte e altre informazioni contestuali. Il secondo endpoint utilizzato è:

`https://volleyball.sportdevs.com/matches?or=(home_team_id.eq.{team_id},away_team_id.eq.{team_id})&order=specific_start_time.desc&limit=5`

Questo viene impiegato per ottenere gli ultimi cinque incontri disputati da una specifica squadra, identificata dal parametro `team_id`. Tali informazioni possono essere utilizzate per arricchire gli snapshot con contesto storico immediato. Infine, il terzo endpoint è:

`https://volleyball.sportdevs.com/matches?or=(and(home_team_id.eq.{home_id},away_team_id.eq.{away_id}),and(home_team_id.eq.{away_id},away_team_id.eq.{home_id}))&order=specific_start_time.desc&limit=20`

Questo permette di recuperare lo storico degli scontri diretti (head-to-head) tra due squadre, fornendo un quadro comparativo utile tra avversarie ricorrenti.

Le richieste a questi endpoint vengono eseguite periodicamente, con gestione dei token di autenticazione quando necessario. I dati ottenuti vengono poi pre-processati dal producer per adattarli al formato previsto dalla pipeline.

2.1.3 Trasformazioni e Feature Addizionali

A valle della raccolta dei dati grezzi dagli endpoint SportDevs, lo script Python applica un processo di trasformazione che arricchisce ogni snapshot con una serie di colonne derivate, necessarie per semplificare l'analisi nei passaggi successivi e per arricchire la semantica delle informazioni trasmesse. Vengono innanzitutto calcolati i punteggi totali aggiornati per ciascuna squadra (`home_score_total` e `away_score_total`) e la loro differenza (`score_diff`), utile per valutare rapidamente l'andamento della partita. A questi si aggiunge la differenza tra il numero di set vinti dalle due squadre, rappresentata nella colonna `set_diff`. Lo stato del set in corso viene descritto tramite i punteggi parziali (`home_current_score` e `away_current_score`) e una rappresentazione sintetica della situazione (`set_info`), come ad esempio "Set 3: 18–22". L'arricchimento dello snapshot prosegue con l'inserimento di tre metriche derivate da partite storiche: `home_win_rate_last5` e `away_win_rate_last5` indicano le percentuali di vittorie delle due squadre negli ultimi cinque match disputati, mentre `head_to_head_win_rate_home` riflette il tasso di vittorie della squadra di casa negli scontri diretti più recenti contro l'avversaria.

Per quanto riguarda i dati storici, che richiederebbero interrogazioni multiple verso gli endpoint /matches. Per ottimizzare le prestazioni e ridurre la frequenza di chiamate HTTP, viene implementato un sistema di caching locale: quando le partite storiche di una determinata squadra o di una coppia di squadre sono già state scaricate in una precedente iterazione, i dati vengono riutilizzati evitando ulteriori richieste. Questo approccio consente di mantenere l'aggiornamento in tempo reale senza sovraccaricare l'infrastruttura remota.

Tutte le trasformazioni vengono applicate prima dell'invio dello snapshot a Logstash, contribuendo a mantenere leggerezza e modularità nelle componenti a valle della pipeline, che possono così concentrarsi sull'analisi e sull'indicizzazione dei dati già strutturati.

2.2 Elaborazione e Normalizzazione: Logstash

Una volta generati dal producer Python, gli snapshot arricchiti vengono inviati a Logstash, che rappresenta il primo nodo intermedio della pipeline. Il suo ruolo è quello di ricevere, pulire e uniformare i dati prima della loro distribuzione nel broker Kafka.

Logstash è configurato per ricevere i messaggi tramite input HTTP sulla porta 9090, utilizzando il codec JSON. Questo consente al producer di inviare i dati in formato strutturato direttamente tramite una richiesta POST.

All'interno del blocco di filtro, Logstash esegue una serie di operazioni di pulizia e arricchimento. In particolare, vengono rimossi i campi non rilevanti ereditati dalla richiesta HTTP e viene aggiunto un campo di metadati che memorizza il timestamp esatto in cui Logstash ha processato il messaggio.

I messaggi validati e normalizzati vengono infine inoltrati a Kafka sul topic `matchvolley`, mantenendo il formato JSON originale. Il collegamento è configurato per utilizzare il broker Kafka all'indirizzo interno della rete containerizzata.

La configurazione completa di Logstash è definita nel file `logstash/logstash.conf`. Il suo inserimento nella pipeline consente di rendere più flessibile e manutenibile il sistema, centralizzando le operazioni di normalizzazione in un punto dedicato. Questo permette, ad esempio, di apportare modifiche al formato dei messaggi in maniera indipendente rispetto al codice applicativo, favorendo una separazione netta tra acquisizione ed elaborazione iniziale.

2.3 Gestione del Flusso Dati: Kafka

Nel flusso dati di VolleyLive, Kafka svolge la funzione di snodo centrale attraverso cui transitano tutti gli snapshot live generati dal producer e normalizzati da Logstash. In questa fase della pipeline, il suo compito principale è quello di garantire una distribuzione fluida e affidabile dei dati verso i moduli di analisi in tempo reale.

Gli snapshot vengono pubblicati nel topic `matchvolley`, utilizzato attualmente con una configurazione a singola partizione. Questa scelta, adottata in fase di sviluppo, consente di mantenere il sistema semplice e lineare, evitando la necessità di coordinare il consumo parallelo da più partizioni. Si tratta di una configurazione pienamente adeguata al volume attuale di dati, che resta contenuto e gestibile in modo sequenziale.

Allo stesso tempo, l'infrastruttura rimane predisposta per un'evoluzione futura: aumentando il numero di partizioni, sarà possibile sfruttare il parallelismo interno a Kafka per scalare l'elaborazione su più istanze in maniera nativa. In questo modo, eventuali aumenti del carico dati potranno essere gestiti senza modificare l'architettura di base, ma semplicemente ricalibrando i parametri del topic.

Data Ingestion e Analisi Streaming

In questo capitolo descriviamo l'architettura di elaborazione in real-time basata su Spark Structured Streaming: dalla lettura dei topic Kafka contenenti sia le feature grezze che gli output del modello, fino all'indicizzazione su Elasticsearch.

3.1 Integrazione di Spark Structured Streaming con Kafka

Nel file `spark/spark.py` la pipeline di ingestione inizia con l'istanza di una `SparkSession`, a cui vengono aggiunte le dipendenze per i connettori Kafka e Elasticsearch. Inoltre, viene disabilitata la compressione Parquet, poiché la pipeline non prevede salvataggi persistenti in formato Parquet e punta a ridurre la latenza e il carico computazionale durante la gestione dei checkpoint interni di Spark Structured Streaming.

```
spark = SparkSession.builder \
    .appName("VolleyballSnapshotProcessor") \
    .config("spark.jars.packages",
        "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0,"
        "org.elasticsearch:elasticsearch-spark-30_2.12:8.7.1") \
    .config("spark.sql.parquet.compression.codec", "uncompressed") \
    .getOrCreate()
```

Subito dopo si definisce lo `snapshot_schema`, un oggetto `StructType` che mappa ciascun campo del JSON di input :

```
snapshot_schema = StructType(Seq(
  StructField("match_id", IntegerType, nullable = false),
  StructField("timestamp", StringType, nullable = false),
  ...
  StructField("head_to_head_win_rate_home", DoubleType, nullable = false)
))
```

Lo stream Kafka viene aperto con:

```
kafka_df = spark.readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "kafka-volley:9092")
  .option("subscribe", "matchvolley")
  .option("startingOffsets", "latest")
  .load()
```

Il DataFrame ottenuto espone i campi `key` e `value` come `binary`, oltre a metadati quali `topic`, `partition`, `offset` e `timestamp`. Per deserializzare il payload JSON:

```
parsed_df = kafka_df
  .select(from_json(col("value").cast("string"), snapshot_schema).alias("data"))
  .select("data.*")
```

Infine, per convertire la colonna `timestamp` da `StringType` a `TimestampType` mantenendo la precisione ai microsecondi, si applica:

```
processed_df = parsed_df.withColumn(
  "timestamp",
  date_format(
    to_timestamp(col("timestamp"), "yyyy-MM-dd HH:mm:ss.SSSSSS"),
    "yyyy-MM-dd HH:mm:ss.SSSSSS"
  )
)
```

Il DataFrame `processed_df` risultante contiene ora solo colonne tipizzate e pronte per le fasi successive di scoring e indicizzazione.

3.2 Modello predittivo: training, feature e scoring in tempo reale

3.2.1 Addestramento del modello

L'addestramento del modello predittivo avviene interamente all'avvio del container Spark: ad ogni riavvio del job viene caricato il dataset di snapshot etichettati da CSV, eseguito il parsing della durata di gioco e calcolate le feature avanzate (set diff, flag critici e pesature delle statistiche). Su questi dati si costruisce una pipeline MLlib modulare, composta da:

1. **Imputer**, che sostituisce i valori mancanti delle feature numeriche con le mediane calcolate sul training;
2. **StringIndexer**, che trasforma gli identificativi di squadra in indici numerici, gestendo le categorie non viste con `handleInvalid="keep"`;
3. **VectorAssembler**, che aggrega tutte le feature (imputate e one-hot) in un unico vettore;
4. **StandardScaler**, che centra e scala il vettore a media zero e deviazione standard uno;
5. **LogisticRegression**, che applica la regressione logistica con regolarizzazione L2 (`regParam=1.0`, `elasticNetParam=0.0`) e calcola la probabilità di vittoria (`probabilityCol="probability"`).

Al termine dell'addestramento il modello viene salvato in `/data/models/volley_logreg/latest` mediante `model.write().overwrite().save()`. In fase di scoring streaming, Spark non riaddestra più il modello: carica semplicemente il modello serializzato e le mediane per il fillna, applica in tempo reale le trasformazioni agli snapshot provenienti da Kafka e indicizza le probabilità predittive in Elasticsearch.

3.2.2 Feature derivate

Per migliorare la capacità predittiva del modello, oltre alle variabili direttamente disponibili negli snapshot (come punteggio, set vinti e durata della partita), lo script Spark calcola "on-the-fly" una serie di feature derivate definite nel file `spark.py` tramite UDF, sintetizzando informazioni più complesse e situazionali. Tra le principali feature si annoverano:

- `set_diff_current`, che ricava la differenza punti dell'ultimo set giocato analizzando il campo testuale `set_info`;
- `current_set_number`, ottenuta dal parsing del campo `match_status` per individuare il momento del match (inizio, set centrale, set decisivo);
- `set_importance`, un valore numerico (0.5 o 1.0) che misura l'importanza strategica del set corrente sull'esito finale;

- flag binari come `flag_3set_severo_home` e `flag_critico_base_away`, che segnalano situazioni potenzialmente critiche in base al punteggio e all'andamento;
- `home_win_rate_adj` e `away_win_rate_adj`, versioni adattate dei win rate storici, penalizzate dinamicamente in caso di svantaggio nel corso del set;
- `win_rate_diff`, che sintetizza in un'unica variabile il differenziale tra le due versioni adattate dei win rate.

Per dare peso maggiore alle condizioni di gioco correnti rispetto alle statistiche storiche, `set_diff_current` viene moltiplicato per un fattore 2, enfatizzando il differenziale istantaneo, mentre le feature basate su statistiche storiche (come `home_win_rate_last5`, `away_win_rate_last5` e `head_to_head_win_rate_home`) vengono scalate di un fattore 0.5, introducendo un decadimento temporale. Queste trasformazioni, eseguite direttamente da Spark nel flusso di elaborazione, hanno dimostrato di aumentare stabilità e precisione del modello rispetto all'uso delle sole feature originali, permettendo di catturare in modo più efficace fattori dinamici e situazionali del match.

3.2.3 Applicazione del modello in streaming

Durante l'esecuzione della pipeline Spark, il modello salvato viene caricato in memoria e applicato in tempo reale agli snapshot provenienti da Kafka. Prima della predizione, i dati vengono preparati replicando in Spark le trasformazioni calcolate in fase di addestramento, comprese le feature derivate come `set_diff_current`, `set_importance` e `win_rate_diff`.

Per ogni snapshot viene generata una stima della probabilità di vittoria della squadra di casa, memorizzata nella colonna `predicted_win`. Questa metrica rappresenta il livello di *scoring* attualmente integrato nel sistema e riflette una valutazione probabilistica aggiornata del match, basata su dati live e statistiche storiche.

3.3 Indicizzazione degli snapshot su Elasticsearch

Completata la fase di *scoring*, l'output finale viene scritto in tempo reale su Elasticsearch. Il `DataFrame` risultante, denominato `output_df`, include sia le feature originali dello snapshot, sia la colonna `predicted_win`, ottenuta tramite la funzione `extract_prob`, che converte il vettore `probability` prodotto da Spark in un singolo valore numerico rappresentante la probabilità di vittoria per la squadra di casa:

```
output_df = scored \
    .withColumn("predicted_win", extract_prob(col("probability"))) \
    .select(*final_cols)
```

Per l'indicizzazione viene utilizzato il connettore `elasticsearch-spark`, configurato tramite il metodo `writeStream`. I dati vengono scritti all'interno dell'indice `volleyball_matches`, attraverso connessione diretta al nodo `elasticsearch-volley` sulla porta `9200`. La modalità `append` consente di aggiungere nuovi documenti senza sovrascrivere i precedenti.

La configurazione prevede inoltre una directory di checkpoint (`/tmp/spark-es-logreg-checkpoint`), fondamentale per garantire l'esecuzione *exactly-once* e la tolleranza a guasti. Questo meccanismo permette al sistema di recuperare lo stato dello stream in caso di interruzione, evitando la duplicazione degli snapshot già processati.

La scrittura è avviata con il seguente blocco di codice:

```
es_query = output_df.writeStream \
    .format("org.elasticsearch.spark.sql") \
    .option("es.nodes", "elasticsearch-volley") \
    .option("es.port", "9200") \
    .option("es.resource", "volleyball_matches") \
    .option("es.write.operation", "index") \
    .option("checkpointLocation", "/tmp/spark-es-logreg-checkpoint") \
    .outputMode("append") \
    .start()

es_query.awaitTermination()
```

Con questa fase si conclude la pipeline di elaborazione streaming: ogni snapshot ricevuto da Kafka viene arricchito con una predizione e subito reso disponibile in Elasticsearch da cui viene interrogato tramite API per essere visualizzato in modo semplice e intuitivo nel frontend.

Visualizzazione e Interfaccia Utente

4.1 Architettura del Sito e Organizzazione del Codice

L'architettura del sito *VolleyLive* si basa su una chiara separazione tra interfaccia utente e logica *backend*, rispecchiata nella suddivisione delle directory all'interno della repository.

Il codice dell'interfaccia utente è contenuto nella cartella `/frontend`, sviluppata con il framework *Next.js*. Per lo styling è stato adottato Tailwind CSS, una libreria utility-first che consente di costruire interfacce responsive in modo rapido e modulare. Grazie a Tailwind, i componenti della UI sono stati progettati con uno stile coerente, facilmente personalizzabile e pienamente compatibile con dispositivi mobili e schermi di diverse dimensioni.

L'architettura del progetto segue le convenzioni di Next.js, con una struttura chiara e modulare che facilita l'estensione e la manutenzione del codice :

- `app/`: definizione delle rotte e delle pagine principali,
- `components/`: insieme di componenti UI riutilizzabili,
- `public/`: risorse statiche come immagini e icone,
- `styles/`: fogli di stile personalizzati tramite Tailwind CSS,
- `utils/`: funzioni di utilità condivise tra i vari componenti.

Il *backend* si trova nella directory `/backend` ed è sviluppato con *FastAPI*, fornendo un'interfaccia REST che consente al frontend di interrogare Elasticsearch:

- `main.py`: entrypoint dell'applicazione,
- `models/`: definizione dei modelli dati e delle classi di risposta.

4.2 Backend API: Connessione tra Elasticsearch e Frontend

Il backend del sistema VolleyLive è sviluppato in **FastAPI**, una libreria moderna e performante per la costruzione di API RESTful. Il suo ruolo principale è fornire un'interfaccia leggera tra il frontend e il database Elasticsearch, restituendo dati aggiornati in formato JSON da visualizzare nella dashboard.

Nel file `main.py`, situato nella directory `/backend`, viene inizializzata l'applicazione FastAPI e viene definita la connessione al servizio Elasticsearch, indirizzato al container `elasticsearch-volley`. La comunicazione avviene tramite richieste HTTP POST, con payload in formato JSON.

La funzione centrale del backend è `es_search(payload: dict)`, che incapsula la logica di invio delle richieste verso Elasticsearch e la gestione degli errori. Essa viene invocata dagli endpoint definiti nell'applicazione, come ad esempio:

```python
@app.get("/match/{match_id}")
def get_match_data(match_id: int):
    query = {
        "query": { "term": { "match_id": match_id } },
        "sort": [ { "timestamp": { "order": "desc" } } ],
        "size": 1
    }
    result = es_search(query)
    hits = result.get("hits", {}).get("hits", [])

    if not hits:
        raise HTTPException(status_code=404, detail="Match non trovato")

    return hits[0]["_source"]
```

Questo endpoint consente di ottenere l'*ultimo snapshot disponibile per una determinata partita*, identificata tramite `match_id`. Il risultato restituito contiene i dati completi della partita, tra cui punteggi, stato, set vinti e lo score predittivo `predicted_win`.

Il secondo endpoint, definito con

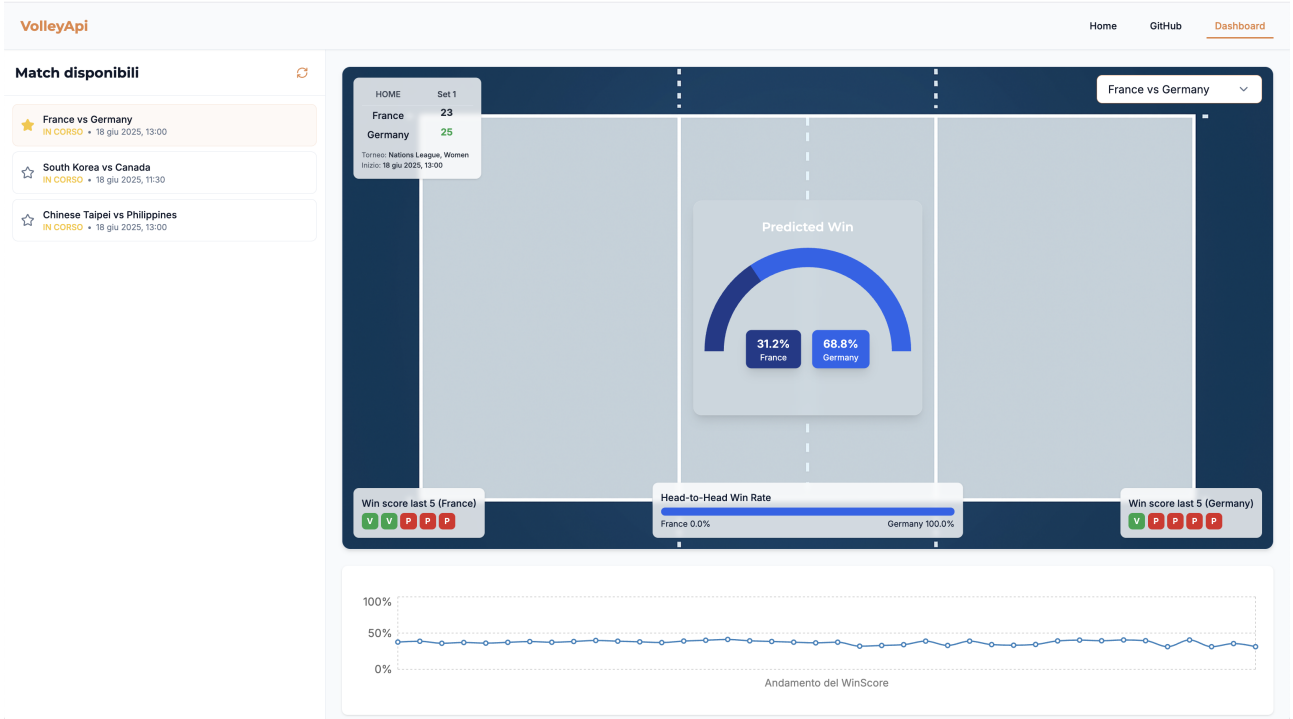
```python
@app.get("/matches/{match_id}/sets-predictions")
```

recupera invece l'intera sequenza di aggiornamenti per una data partita, ordinando i documenti per `timestamp` in ordine crescente. Dopo il fetch, il backend suddivide i risultati in due array distinti: `sets`, contenente gli oggetti `set_info` con relativo `timestamp`, e `predictions`, con i valori `predicted_win` e i rispettivi `timestamp`.

In fase di sviluppo, il middleware CORS è configurato per accettare richieste da qualsiasi origine, permettendo la comunicazione con il frontend anche in ambienti locali.

L'interfaccia API così progettata offre un punto di accesso semplice e modulare ai dati, ed è facilmente estendibile per supportare filtri aggiuntivi, endpoint aggregati o sistemi di caching futuri.

4.3 Struttura e Funzionalità della Dashboard



La dashboard del sistema VolleyLive è progettata per fornire una panoramica completa e interattiva delle partite in corso, aggiornandosi in tempo reale tramite le API interne. L'interfaccia si compone principalmente di due sezioni: una colonna laterale sinistra per la selezione dei match e un'area centrale che visualizza i dettagli del match attivo.

Nella colonna sinistra vengono elencate le partite disponibili, con informazioni essenziali come i nomi delle squadre, lo stato della partita (*"in corso"*) e l'orario d'inizio. Ogni riga è selezionabile e l'utente può anche marcare un match come preferito tramite l'icona a forma di stella. La lista si aggiorna dinamicamente interrogando l'API di backend, mostrando solo i match attualmente live.

La sezione principale della dashboard mostra nel dettaglio le informazioni della partita selezionata. In alto a sinistra è presente un box riassuntivo con:

- i nomi delle due squadre,
- i punteggi aggiornati set per set,
- il torneo e l'orario della partita.

Al centro dello schermo viene visualizzato il valore di **predicted win**, ovvero la probabilità stimata di vittoria calcolata dal modello di machine learning. Questo valore è rappresentato tramite un grafico a semicerchio (gauge) che mostra in modo chiaro l'andamento della probabilità tra squadra di casa e ospite.

Nella parte inferiore della schermata sono presenti tre blocchi distinti:

- A sinistra, una sintesi della forma recente della squadra di casa, con gli ultimi cinque risultati (vittorie e sconfitte) rappresentati da pallini colorati.
- Al centro, una barra che rappresenta le statistiche degli scontri diretti (*head-to-head win rate*) tra le due squadre, utile per contestualizzare la previsione.
- A destra, lo stesso indicatore di forma per la squadra ospite.

Infine, nella parte più bassa viene mostrato un grafico a linea che rappresenta l'**andamento temporale del punteggio predittivo**. Questo consente all'utente di osservare come si è evoluta la probabilità di vittoria nel corso del match, fornendo una visione più profonda sull'equilibrio o sul dominio tra le squadre.

Tutti i dati sono aggiornati automaticamente ogni pochi secondi, senza necessità di refresh manuale. Grazie a questo meccanismo di aggiornamento continuo, l'utente può seguire in tempo reale l'evoluzione della partita, con un'interfaccia chiara, responsiva e focalizzata sugli elementi più rilevanti.

Esecuzione del Sistema e Risultati Osservati

5.1 Esecuzione Completa della Pipeline

5.2 Valutazione del Modello Predittivo

Per misurare la capacità di generalizzazione del modello, i dati sono stati suddivisi per `match_id`: il 80 % degli snapshot è stato utilizzato per l'addestramento, mentre il restante 20 % è stato riservato al test, garantendo l'assenza di leak tra train e test.

Sul test set, costituito da 721 snapshot, si ottengono le seguenti metriche:

- **Accuracy:** 0.7462
  Il modello classifica correttamente circa 3 snapshot su 4.
- **AUC–ROC:** 0.8479
  Buon potere discriminante tra vittoria e sconfitta, indipendente dalla soglia di decisione.
- **Log Loss:** 0.5764
  Indica la qualità delle probabilità predette; valori più bassi denotano migliore calibrazione.
- **Brier Score:** 0.1939
  Misura la distanza quadratica tra probabilità predetta e risultato osservato.

La matrice di confusione sul test set è:

	Pred=0	Pred=1
Actual=0	328	117
Actual=1	66	210

Si riscontrano 117 falsi positivi e 66 falsi negativi, a indicare situazioni in cui il modello sovrastima o sottostima la probabilità di vittoria.

Va inoltre ricordato che il file `trainer_model.csv` è in continua crescita: ogni nuovo snapshot arricchisce il dataset e consente un costante miglioramento delle prestazioni del modello. Con l'aumentare dei dati raccolti, è ragionevole attendersi un incremento dell'AUC–ROC e una diminuzione di Log Loss e Brier Score.

Nel paragrafo seguente confronteremo queste prestazioni con l'evoluzione predittiva di un singolo match, analizzando come varia lo *score* predittivo in corrispondenza dei momenti chiave della partita.

5.3 Caso Studio: Evoluzione di un Match

5.4 Latenza e Prestazioni della Pipeline

Conclusioni e Sviluppi Futuri

Bilancio Finale e Competenze Sviluppate Estensioni e Sviluppi Futuri