Politecnico di Torino

Politecnico di Torino
Corso di Laurea Magistrale in Ingegneria Gestionale

CAP: Customer Attributes Platform.
Progettazione e sviluppo di una piattaforma distribuita per analizzare e processare Big Data

Relatrice:
Prof.ssa Tania Cerquitelli

Laureando:
Vincenzo Cuccaro

Anno Accademico 2023/2024

Indice

Capitolo 1

Introduzione

Nell'odierno panorama competitivo, i dati assumono sempre più rilevanza nei contesti enterprise, a tal punto che la maggior parte dei processi aziendali di successo sono guidati da strategie data-driven. Per estrarre valore dai dati bisogna realizzare infrastrutture e applicazioni efficienti e mantenibili, in grado di far fronte a una crescente complessità e volumi di dati sempre più consistenti.

In particolare, per le aziende del settore fashion & luxury, la velocità nei processi decisionali e nella gestione dei clienti diventa un fattore critico di successo.

L'obiettivo di questo elaborato è la presentazione di CAP (Customer Attributes Platform), una piattaforma di calcolo distribuito progettata per l'analisi dei clienti in un'azienda del settore dell'alta moda e di articoli di lusso. CAP, in senso più ampio, consiste in un'infrastruttura più estesa, che gestisce non solo il calcolo dei KPI, ma anche la loro propagazione verso diversi applicativi con funzioni di reporting, marketing e CRM (Customer Relationship Management). Attualmente i dati prodotti vengono utilizzati da diversi brand che appartengono alla stessa holding.

1.1 Il problema

CAP mira a risolvere tre problemi principali:

- decentralizzazione del calcolo e della storicizzazione dei KPI relativi ai clienti
- tempi di calcolo dei KPI eccessivamente lunghi
- impossibilità di scalare facilmente le risorse in caso di necessità

Prima di CAP, i KPI dei clienti erano calcolati in diverse applicazioni già esistenti nell'ecosistema dell'organizzazione aziendale. Uno degli aspetti determinanti che hanno promosso lo sviluppo di CAP, infatti, è proprio l'obiettivo di avere un'unica fonte di verità, capace di distribuire a tutte le applicazioni che ne necessitano vecchi e nuovi KPI. I vecchi sistemi on-premise di calcolo soffrivano del costante aumento dei dati da elaborare e giorno dopo giorno il carico sulle prestazioni era sempre più difficile da gestire.

1.2 La soluzione

La soluzione sviluppata consiste nell'integrazione di un nuovo sistema capace di calcolare e storicizzare i KPI, totalmente integrato con i sistemi applicativi target e basato principalmente su infrastruttura cloud, in modo da rendere flessibile la scalabilità delle risorse utilizzate. Il nuovo sistema CAP è diviso in due parti:

- CAP Backend (Engine): calcola, storicizza, propaga e aggiorna gli attributi dei costumer nei sistemi target.

  È stato integrato usando il servizio AWS EMR (Elastic MapReduce), che sfrutta le potenzialità di Apache Spark per processare e gestire ogni giorno un gran numero di dati. Ogni KPI viene calcolato usando come sorgente dati le informazioni dei clienti e del loro comportamento di acquisto. I dati sono storicizzati su AWS S3, che rappresenta il principale sistema di data storage del Datalake dell'infrastruttura. I KPI possono essere calcolati e aggiornati giornalmente, settimanalmente o mensilmente, in base alle esigenze di business. Il calcolo può essere lanciato in modalità "full" (ogni volta vien considerato tutto il set dei dati) o in modalità "incremental" (considerando solo il set di dati aggiornati rispetto il calcolo precedente). In tutti i casi, dopo il calcolo, gli attributi dei clienti vengono storicizzati su AWS S3 e successivamente scritti su un'istanza AWS DocumentDB e altri sistemi target che hanno bisogno di ricevere queste informazioni.

- CAP Frontend (API & UI): è un applicazione che espone via API i dati precedentemente calcolati e storicizzati su DocumentDB dal CAP Backend, in modo che siano visualizzabili nel CAP Manager, ossia l'interfaccia web utilizzata in varie aree dell'organizzazione (come ad esempio gli store e i team di supporto clienti).

1.3 Contenuto dell'elaborato

L'oggetto di questa tesi si focalizzerà sul CAP Engine. Verrà introdotta inizialmente l'architettura con uno schema ad alto livello, per poi entrare più in dettaglio sui singoli componenti. Lungo il testo verrà spiegato come l'architettura di CAP è integrata nell'ecosistema dell'organizzazione e come i KPI vengono calcolati dall'applicazione. Alla fine, con il supporto di uno use case, verrà mostrata la trasformazione del dato tra i diversi layer di calcolo e come esso viene propagato verso i sistemi target.

Capitolo 2

Architettura

Questo capitolo descrive dove CAP è collocato all'interno dell'ecosistema. In particolare viene fatto un overview su: il sistema sorgente da cui i dati d'input vengono letti, dove i dati sono storicizzati, quali sono i sistemi target e come CAP comunica con loro. A seguire c'è un focus sull'architettura del CAP Engine, dunque sui componenti dell'applicazione e su come questi ultimi interagiscono tra loro. In conclusione verranno mostrate che tipo di trasformazioni vengono applicate ai dati.

2.1 Schema ad alto livello: CAP nell'Ecosistema

Nella fig. 2.1 è possibile vedere che il sorgente del CAP Engine è il Datalake, costruito su AWS S3. Il Datalake contiene informazioni legate a più aree dell'organizzazione, ma quelle utilizzate da CAP sono prevalentemente quelle relative alle vendite, ai clienti e agli store. La tipologia di informazioni necessarie per il calcolo dei KPI dipende dalle specifiche business di ogni attributo, ma in generale i più rilevanti sono quelli legati al comportamento di acquisto del cliente. Sul fronte opposto invece sono rappresentati i diversi sistemi target, i quali giocano un ruolo diverso nella struttura aziendale. Nel seguente paragrafo verranno presentati la maggior parte dei sistemi mostrati in figura, con un focus particolare al Datalake. Questo focus speciale è giustificato dal fatto che il team CAP, anche se appartiene a tutt'altra area e organizzazione rispetto al team Datalake, utilizza la maggior parte delle tecnologie e servizi gestiti dal loro team. Come è possibile notare in figura, l'accesso ai dati prodotti da CAP può essere fatto sfruttando diverse tecnologie, in base ovviamente a come i sistemi target sono integrati nell'infrastruttura di CAP. In generale i modi più utilizzati sono API, SFTP, JDBC e via condivisione file su AWS S3.

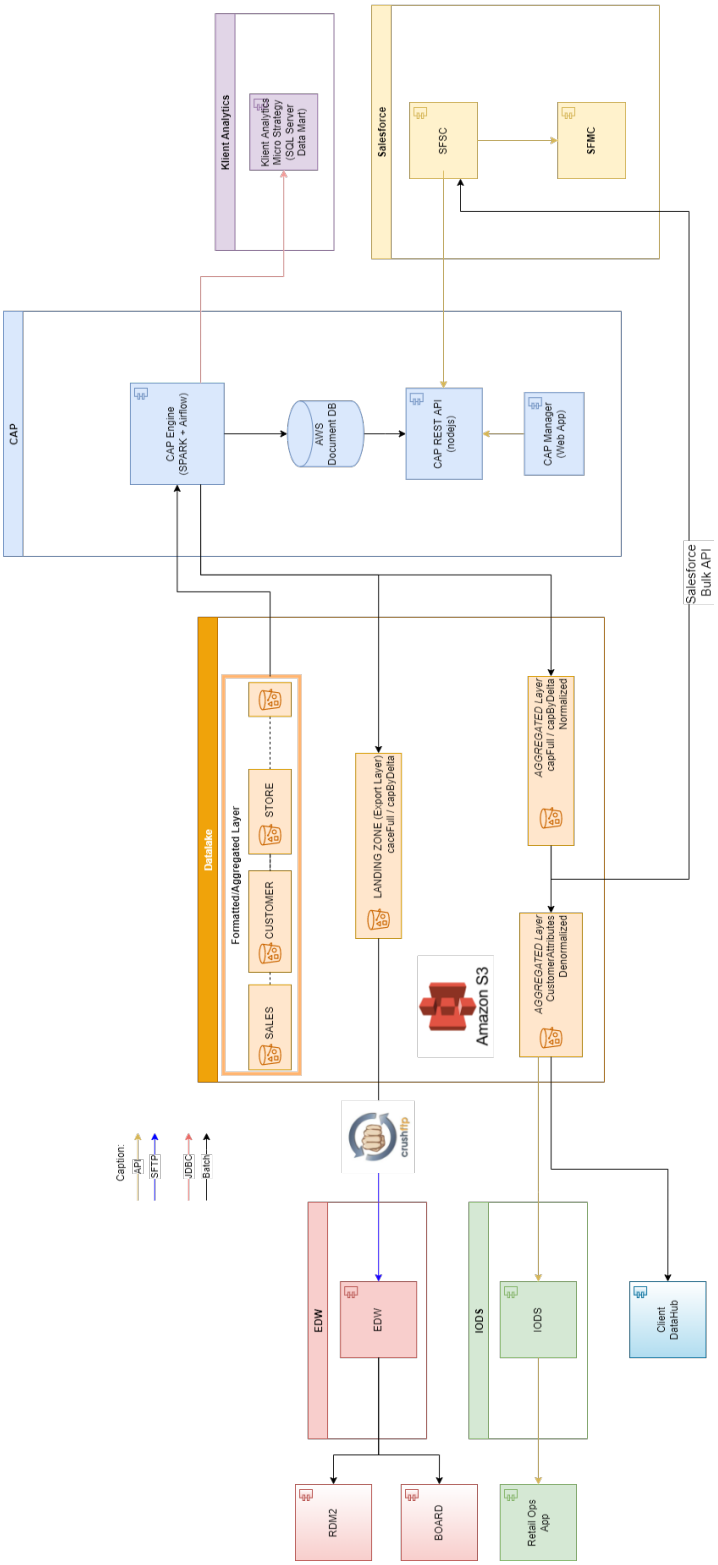

Figura 2.1: Schema ad alto livello di CAP

2.1.1 Datalake

"Un data lake è un repository centralizzato progettato per archiviare, elaborare e proteggere grandi quantità di dati strutturati, semistrutturati e non strutturati." [6]. Nonostante i datalake offrono una soluzione di centralizzazione dei dati, essi si differiscono dai tradizionali datawarehouse. La differenza principale riguarda proprio la tipologia di struttura dati, che nei datalake è più flessibile. Nei datawarehouse siamo abituati ad un'organizzazione delle informazioni di tipo tabellare, in quanto le loro finalità sono specifiche e già predeterminate nell'organizzazione aziendale. Spesso nei datalake invece, il dato viene storicizzato ancor prima di definirne l'impiego, e messo quindi a disposizione in forma del tutto flessibile a progetti ed esigenze future.[11]

Il datalake dell'ecosistema in cui CAP è integrato è gestito tramite AWS S3 Bucket. Il bucket principale dell'ecosistema è stato organizzato in quattro diversi layer, inoltre esistono ulteriori layer di import/export storicizzati in un altro S3 Bucket chiamato *landingzone*. I quattro layer differiscono principalmente per organizzazione, accessibilità, formato e storage key. Essi sono:

- RAW: è organizzato per brand / source system. Il dato è storicizzato così come vengono integrati dai sorgenti esterni; lo scopo di questo layer infatti è quello di salvare tutte le versioni (nella maggioranza dei casi una versione al giorno) del sistema sorgente. I formati utilizzati sono solitamente: CSV, AVRO, JSON, Parquet. La storage key è: data/raw/⟨brand⟩/⟨source⟩/⟨dataEntity⟩. Solitamente la storage key è composta da un ulteriore sottolivello /$date, ed è quindi presente per ogni data la fotografia del sistema sorgente. Le policy di accesso sono delimitate al solo team "Datalake", che è il responsabile della gestione di ingestion dati su S3.

- FORMATTED: è organizzato per brand / source system. Il dato è storicizzato in Parquet o Deltalake format. Questo layer viene utilizzato per convertire il formato dei dati nello standard definito dall'organizzazione del Datalake. La storage key è: data/formatted/⟨brand⟩/⟨source⟩/⟨dataEntity⟩. Anche in questo caso la storage key può essere composta da un ulteriore sottolivello /$date o /latest (in caso di deltalake). Rispetto al layer RAW esistono delle logiche di retention che cancellano dati più vecchi di un limite definito. Ogni team tecnico ha accesso alla sola porzione di storage key che compete al suo progetto. Su questo layer, è presente anche il ⟨brand⟩ "corp", che contiene i dati di tutti i brand.

- AGGREGATED: è organizzato per brand / domain / Business Objects. Il dato è storicizzato in formato Parquet. Questo layer viene utilizzato per ar-

ricchire il dato da informazioni ricavate da altri sorgenti e per creare ciò che è definito come "business object". Le policy di accesso dati sono le medesime del layer precedente. La storage key è: data/aggregated/⟨brand⟩/⟨domain⟩/⟨dataEntity⟩.

- USAGE: è organizzato per brand / usage. In questo layer si hanno le regole di business (aggregazioni, filtri, join ...) che sono applicate per un unico specifico Datamart / Usage / Usecase. Lo storage in questo layer può essere gestito come file Parquet S3, ma ci sono casi specifici in qui il dato viene scritto direttamente su altri databese, come: Cassandra, DocumentDB, Amazon Aurora. Le regole di accesso dipendono dai sistemi in cui il dato è storicizzato. In particolare nel caso di storicizzazione su S3, la storage key è: data/usage/⟨brand⟩/⟨usage⟩/⟨dataEntity⟩.

La figura 2.2 mostra come i layer sono suddivisi e quali sono i flussi permessi tra i vari layer. Nella figura fig.2.3 invece è mostrato come il bucket principale interagisce con i layer di import/export del bucket landingzone. Quest'ultimo bucket è progettato per storicizzare i dati temporaneamente, in quanto si tratta di dati non ancora processati che vengono replicati sul layer di raw. La landingzone può essere accessibile via SFTP, sia in import che in export, sfruttando un'integrazione con *CrushFTP*, un server di trasferimento file che semplifica la configurazione di connessioni sicure con gli utenti.

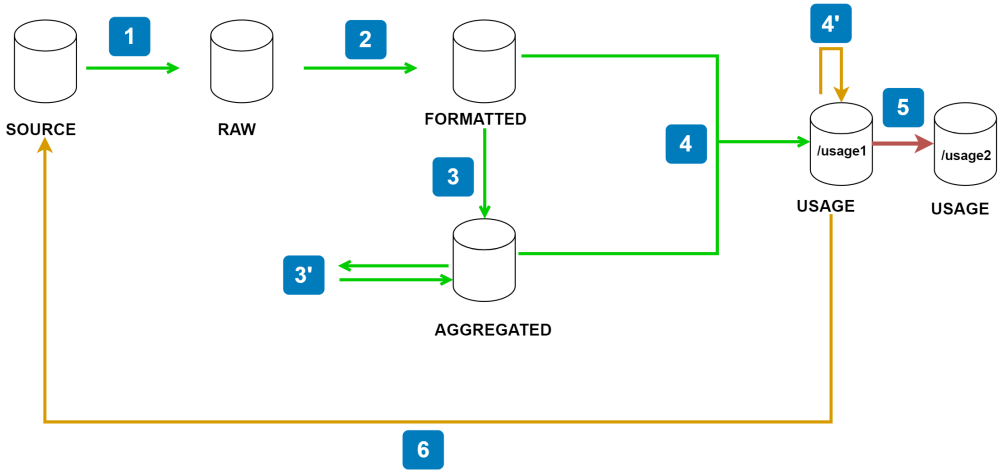

Figura 2.2: Flussi permessi tra Layer del Datalake

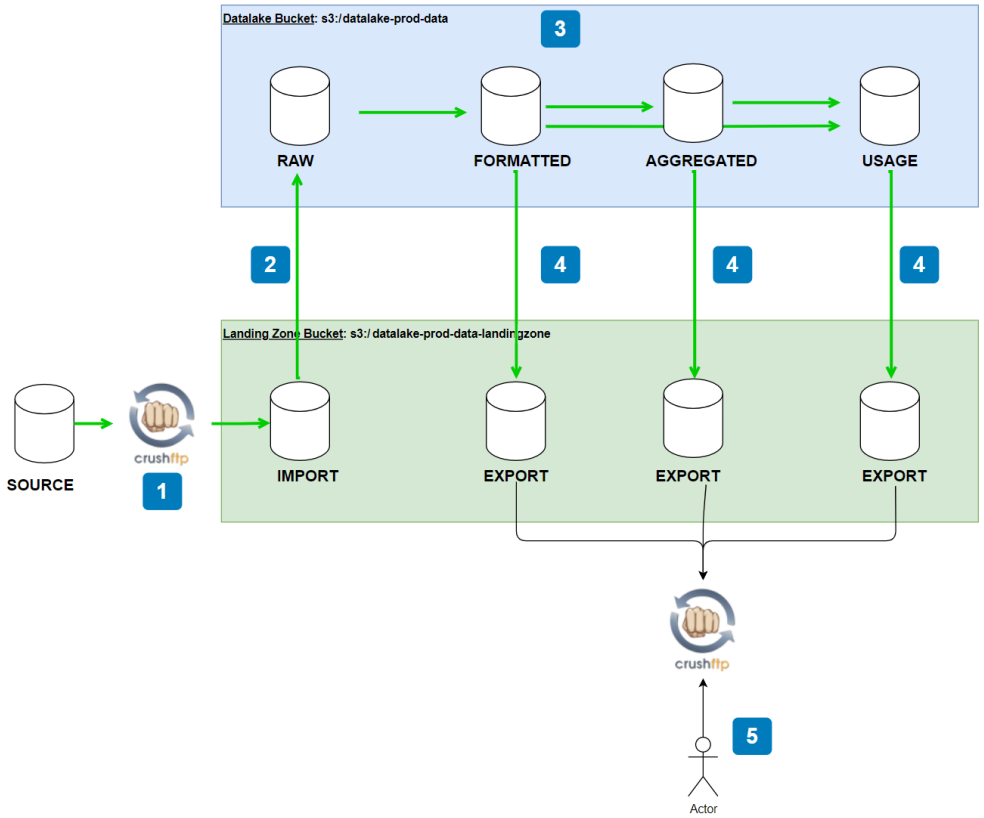

Figura 2.3: Integrazioni della Landingzone Import/Export del Datalake

CAP è una parte integrata del Datalake e utilizza la stessa infrastruttura, le stesse applicazioni e gli stessi tool. La principale differenza è che CAP è business oriented e il team si interfaccia direttamente con le richieste espresse dai brand. CAP utilizza come input dati già integrati dal team Datalake sui layer di formatted o aggregated, e scrive invece sui layer di aggregated e usage, o direttamente su database target esterni. Nei prossimi paragrafi sono descritti singolarmente i database target principale su cui CAP espone i suoi KPI.

2.1.2 Klient Analytics

Klient Analytics è un sistema di reporting che, come suggerisce il nome, produce analitiche relative ai clienti di ogni brand. Per analytics si intende l'insieme di tutte le analisi sui dati utilizzate per scoprire, interpretare e condividere le informazioni presenti nei dati. Klient Analytics è suddiviso in una parte backend, costruita su Microsoft SQL Server, e da una parte frontend, integrata con Microstrategy. "MicroStrategy è uno strumento di analisi e reporting per data warehouse aziendale altamente capace di generare report informativi e completi. È essenzialmente un software di business intelligence che supporta dashboard interattive, formattazione avanzata nei report, scorecard e diverse altre funzionalità relative alla generazione, ordinamento e distribuzione di report che forniscono informazioni sulle tendenze aziendali passate, presenti e future."[8]

Gli attributi calcolati da CAP risultano quindi una fonde determinante dei dati utilizzati su Microstrategy da Klient Analytics, che offre circa 130 report su Client Analysis, Sales Assistant KPIs, Retention Analysis, Segmentation Analysis, Cross Selling Analysis, Marketing Campaign Analytics e altro.

CAP si interfaccia con Klient Analytics connettendosi tramite JDBC direttamente all'istanza di Microsoft SQL Server. CAP scrive il full o i soli aggiornamenti giornalieri su una tabella di staging dedicata che contiene i KPI, in modo che questi ultimi siano disponibili ai sviluppatori backend del sistema.

2.1.3 Salesforce

Salesforce è un'infrastruttura di molteplici servizi per il Customer Relationship Management. Viene utilizzato per la gestione delle relazioni con i clienti che aiuta i team di marketing, vendite, commercio, assistenza e IT a connettersi con i propri clienti.[10] Nell'organizzazione Salesforce viene utilizzato anche nello specifico per fare analitiche per l'area CRM, per cui è integrato con CAP in due modi, in base al modulo Salesforce utilizzato: leggendo dal layer di aggregated su AWS S3 o collegandosi via API all'istanza di AWS DocumentDB dedicata ai KPI di CAP.

2.1.4 EDW: Enterprise Data Whareouse

L'enterprise data warehouse (EDW) è una tecnologia che storicizza e centralizza dati strutturati integrati da una moltitudine di sistemi sorgenti e applicazioni, rendendoli disponibili per le analytics o per altri usi predefiniti dell'organizzazione.[4] EDW legge i dati da CAP via SFTP sfruttando *CrushFTP* che è connesso al bucket di landingzone. L'utilizzo degli attributi di CAP su EDW sono molteplici e dipendono dal tipo di KPI calcolato. Essi possono essere utilizzati come filtri o termini di aggregazione durante i flussi batch di trasformazioni dati o direttamente come valori visualizzati su strumenti di dashboarding o reporting.

2.1.5 Client Data Hub

Un Data Hub è un sistema centralizzato che facilita la condivisione, l'accesso e l'utilizzo dei dati all'interno di un'organizzazione, sfruttando tecnologie di storage, di processamento e mediazione dati moderni. Ha lo scopo principale di centralizzare il dato, rendendo più flessibile la data governance e la gestione dei permessi degli utenti. Si offre quindi come punto di incontro tra i dati strutturati, che solitamente sono gestiti in Data Warehouse, e i dati eterogenei, dispersi invece nel Data Lake. Nell'organizzazione il Data Hub è presente solo per uno dei brand più rilevanti e incide non solo sulla gestione dei clienti ma su tutte le funzioni aziendali. L'infrastruttura è costituita da una Data Platform, costruita su Cloudera, che gestisce lo storage e i flussi batch, e da una Application Platform, basata su servizi di Amazon Web Service, che gestisce la parte più applicativa e i flussi real-time. Con CAP, al momento, il Client Data Hub supporta solo l'ingestion dei dati via batch tramite il bucket S3 Aggregated Denormalized, ma in futuro sono previste nuove integrazioni real-time.

2.1.6 IODS

IODS sta per Indexed Operational Data Store. Generalmente per Operational Data Store si intende un repository centralizzato di dati operazionali all'interno di un organizzazione. I dati al suo interno sono integrati in real-time, per permettere a tool di Business Intelligence di fare analisi su dati operazionali. Risulta essere quindi il ponte tra i sistemi transazionali e i sistemi per analisi dati, e può essere utilizzato in particolare per i sistemi di Customer Relationship Management. [9] Nello specifico, nell'architettura in cui lavora CAP, IODS ha lo scopo di rendere disponibili dati relativi a clienti, vendite, negozi e stock su API, in modo che essi siano raggiungibili da tutti i sistemi che supportano il sistema standard di API dell'organizzazione, e lo fa con un sistema di indicizzazione delle entità per garantire altissime performance in fase di lettura. IODS quindi è integrato direttamente con

CAP, leggendo i KPI calcolati dei clienti direttamente dal set di dati *customerAttributes* sul layer aggregato di AWS S3, rendendoli poi disponibili all'applicazione Retail Ops presente in tutti i negozi.

2.2 Schema a basso livello: il flusso del CAP Engine

La figura 2.4 mostra il flusso del CAP Engine. Verrà fornito prima un'overview iniziale e successivamente una descrizione di ogni modulo del flusso. I moduli in questione sono tre:

- Calculator Module: in questo modulo è presente solo un job Spark, chiamato *fmt_to_agg*. Ogni giorno vengono lanciate diverse istanze di questo job, una per ogni brand della holding. Lo scopo del job è quello di calcolare tutti gli attributi dei clienti per il relativo brand. I dati sorgenti sono principalmente estratti dal layer di formatted del Datalake, mentre l'output viene scritto nel layer di aggregated, nello storage key con ⟨domain⟩ pari a *CAPFull* (nel caso il job sia stato lanciato con la configurazione full) oppure *CAPByDelta* (nel caso il job sia lanciato con la configurazione incrementale).

- Denormalizer Module: questo modulo è composto da due job Spark, chiamati *agg_to_agg_customerAttributes* and *agg_to_agg_CAPDenormalized*.
  Entrambi leggono i sottolivelli *CAPFull* e *CAPByDelta*, e il loro scopo è quello di denormalizzare il dato. Il primo job processa i dati per ogni brand, mentre il secondo processa i dati *corp*, in modo tale che nello stesso dataset ci siano i dati di tutti i brand. In questo modulo il flusso legge e scrive sul layer di aggregated, per cui l'output può essere utilizzato da altri team tecnici di altri progetti, o da applicazioni come IODS.

- Integrator Module: questo modulo è composto dai job Spark che integrano i dati prodotti dal "calculator module" nei sistemi target. Ci sono tre diversi job, che si chiamano rispettivamente *agg_to_usg*, *agg_to_usg_KA* and *agg_to_usg_landingzone*. Il primo scrive i dati sull'istanza di AWS DocumentDB, a cui sono connesse le API di CAP frontend, il secondo scrive i dati sull'istanza Microsoft SQL Server di Klient Analytics, mentre l'ultimo è usato per scrivere i dati sul bucket di Landingzone, che è usato per esportare i dati sull'Enterprise Data Warehouse.

Tutti i flussi sono scritti in linguaggio Scala e vengono eseguiti come job Apache Spark, che è un multi-language engine utilizzato nell'ambito del data engineering, data science e machine learning, sfruttando macchine a singoli nodi o cluster a più

nodi. Le esecuzioni di ogni job sul cluster Hadoop sono gestite da Apache Airflow, un piattaforma open-source di workflow management. Airflow utilizza quelli che son definiti "directed acyclic graphs" (DAG) per gestire l'orchestrazione dei flussi. Il DAG è uno dei concetti centrale di Airflow: è uno strumento per definire le attività da eseguire, descrivendo le loro dipendenze e le loro relazioni, in modo da indicare come dovrebbero essere eseguite. Il DAG stesso non si preoccupa di ciò che accade all'interno delle attività; si preoccupa semplicemente di come eseguirli: l'ordine di eseguirli, quante volte riprovarli, se hanno timeout e così via. La regola principale del DAG è che il grafo deve essere aciclico, ossia, partendo da un qualsiasi nodo deve essere impossibile ritornare ad esso seguendo gli archi del grafo. [2] I task e le dipendenze sono definite in linguaggio Python, per qui Airflow si limita a gestire lo scheduling e le esecuzioni dei flussi. In questo caso i DAG sono schedulati usando un tool chiamato *mediator*, un tool sviluppato dal "Datalake Team", che controlla se tutti i dati sorgenti necessari per un determinato DAG sono presenti, inviando non appena tutti essi sono disponibili un evento ad Airflow, che a sua volta esegue i DAG.

Figura 2.4: Schema a basso livello di CAP

2.2.1 Apache Airflow

Per la gestione dell'orchestrazione dei job Spark sul cluster viene utilizzato Apache Airflow. Airflow è una piattaforma open source per creare, schedulare e monitorare i workflow in modo programmatico.[1]

Durante il proof of concept di CAP, come tool di orchestrazione dei job Spark è stato utilizzato Apache NiFi. Nonostante NiFi offrisse una vasta gamma di componenti e connettori predefiniti per diversi casi d'uso, presentava alcune limitazioni in termini di:

- Manutenzione: La gestione dei flussi risultava complessa e poco intuitiva, richiedendo un impegno considerevole per la manutenzione e l'aggiornamento dei flussi. NiFi è si costruito in Java, ma vincola l'utente a uno sviluppo su interfaccia grafica. Airflow dall'altro lato, è costruito in Python, e permette una configurazione parametrica più orientata allo scripting.[12]
- Monitoring: Le funzionalità di monitoraggio integrate erano limitate per il caso d'uso di CAP, rendendo difficoltoso l'identificazione e la risoluzione di eventuali anomalie.

A valle del proof of concept, rimettendo in discussione l'utilizzo di NiFi per le motivazioni sopraindicate, si valutò l'utilizzo Airflow e della gestione dei job via DAG, soluzione già consolidata dal team Datalake su altri fronti.

Nella sua versione finale CAP quindi gestisce i propri flussi via DAG. Ogni job ha il suo DAG corrispondente, che è costituito da tre nodi standard:

1. launch_spark_job: questo nodo invia semplicemente via API il lancio della Spark application sul resource manager. L'intero processo della Spark application è conteinerizzata in un file jar.
2. wait_spark_job: questo nodo aspetta il feedback del resourse manager relativo alla richiesta di lancio dell'applicazione
3. handle_spark_job_status: questo nodo gestisce la risposta del resourse manager. Se quest'ultimo ha risposto con un fallimento dell'applicazione manderà in fallimento l'intero DAG. In caso contrario, il DAG termina con successo ed esegue, se esiste, il task successivo.

Nella fig.2.5 è mostrata una rappresentazione grafica del DAG. In caso di successo o fallimento, una mail viene mandata ad un indirizzo di mail dedicato per notificare lo stato del DAG. Il lancio di un job Spark necessità di alcuni parametri, i quali sono

sottomessi usando un file JSON versionato per sicurezza su Gitlab. I parametri dipendono da: l'ambiente in cui deve essere lanciata l'applicazione (sviluppo o produzione), l'insieme delle risorse utilizzate del cluster e il brand per il quale l'attributo deve essere calcolato.

Figura 2.5: DAG Graph View

Airflow viene solitamente utilizzato per semplificare la gestione dei task e dei job, infatti in questo ecosistema gioca il ruolo di orchestratore. Per questo motivo l'istanza sul quale è implementato non gode di particolari risorse. Difatti, il calcolo degli attributi, che richiedono un grande utilizzo di risorse, sono lasciati ai job Spark.

2.2.2 Apache Spark

Apache Spark è un sistema di elaborazione distribuito open source utilizzato per carichi di lavoro di big data, eseguendo in cache il framework MapReduce in un cluster di server. L'architettura cluster è costituita principalmente da due core-service:

- Hadoop Ditributed File System HDFS: un sistema distribuito di data storage che permette un accesso ad alte performance ai dati tra i vari server del cluster. Quando l'HDFS raccoglie i dati li organizza in più blocchi che poi memorizza replicando le stesse sui nodi del cluster, in modo da consentire un'elaborazione parallela altamente efficiente e da garantire che i dati siano sempre disponibili prevenendo potenziali perdite.
- Yet Another Resource Negotiar YARN: il componente di coordinamento del cluster, responsabile dell'organizzazione e della gestione delle risorse sottostanti e della pianificazione del calcolo da eseguire

MapReduce è un modello di programmazione per l'elaborazione di grandi set di dati con operazioni in parallelo e distribuite. Il framework consiste in quattro step:

- lettura dei dati in input su HDFS
- funzione map, che trasforma i dati in input in una serie di coppia chiave-valore

- funzione reduce, che raccoglie le coppie chiave-valore prodotte dalla funzione map ed elabora i valori ad essi associati creando come output una o più coppia chiave-valore
- scrittura dell'output su HDFS

Spark è stato creato per affrontare i limiti di MapReduce, eseguendo l'elaborazione in memoria e riutilizzando i dati in cache, rispetto invece il tradizionale Hadoop che utilizza il disco in lettura e scrittura.

Il riutilizzo dei dati su Spark si ottiene attraverso la creazione di DataFrame, un'evoluzione del Resilient Distributed Dataset (RDD). L'RDD rappresenta una raccolta distribuita dei dati tra i nodi del cluster: per permettere operazioni di tipo relazionale e facilitare su Spark la manipolazione dei dati, ai classici RDD sono state aggiunte le informazioni di schema (struttura tabellare), formando quindi quello che si definisce DataFrame.[5][7]

Spark fornisce API di sviluppo in Java, Scala, Python e R. Nel progetto in particolare le applicazioni Spark sono scritte in Scala, e sono coinvolte nei processi di lettura e trasformazione dei dati sorgenti presente nel Datalake.

In questa tesi facciamo funzionalmente riferimento ai "Spark jobs", ma tecnicamente parlando è più corretto chiamarli "Spark application". Una Spark application è un programma costruito su Spark tramite l'utilizzo delle proprie API. Consiste nell'utilizzo di un "driver" e di una serie di "executor" sul cluster manager. Le applicazioni girano sul cluster dell' Hadoop YARN manager. In sintesi il driver è responabilità di generare e controllare:

- Job: un job consiste in un calcolo parallelo di più task, generati come risposta alle Spark action (esempio, save(), collect()...). Quando le action di una Spark application vengono eseguite, il driver lancia uno o più Spark job. Ogni Job costituisce un DAG, ossia una serie di uno o più blocchi sequenziali, chiamati Stage.
- Stage: lo stage è uno dei componenti sequenziali della catena di un Job. Ogni stage difatti, come in una catena, dipende dall'altro. Non tutte le operazioni su Spark possono essere eseguite come un singolo stage; in quel caso esse vengono divise in più stage.
- Task: il task è una singola unità di lavoro che può essere assegnata ad un executor. Uno stage può essere assegnato a più executor proprio grazie alla suddivisione in task. Ogni task corrisponde a una singolo core e lavora su una singola partizione di dati.

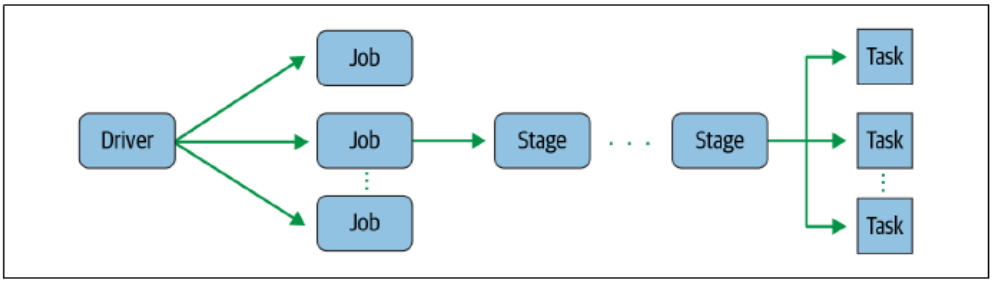

Figura 2.6: Spark Application: Workflow

Più in dettaglio possiamo quindi descrivere driver ed executor in questo modo:

- Driver: è un processo Java dove il metodo main () del nostro programma Scala, Java, Python gira. Esso esegue il codice dell'utente e crea una SparkSession o un SparkContext, in cui è possibile gestire i dati tramite Dataframe. DataSet, RDD, esecuzioni SQL
- Executor: è il processo del nodo worker incaricato ad eseguire task in un determinato Spark job. Una volta che l'executor conclude un task invia il risultato al driver.

Quando lanciamo un'applicazione Spark su un cluster abbiamo bisogno di settare alcuni parametri di configurazione per indicare al resourse manager quante risorse sono necessarie per l'esecuzione dell'applicazione. I parametri più significativi sono:

- spark.app.name: il nome identificativo della tua applicazione
- spark.jars: La lista dei jar da includere come librerie nel driver e nell'executor
- spark.driver.cores: numero di core da utilizzare per il processo del driver
- spark.driver.memory: la memoria da utilizzare per il processo del driver
- spark.driver.memoryOverhead: la memoria addizionale che deve essere allocata per il processo del driver
- spark.executor.instances: numero di exector da lanciare
- spark.executor.memory: la memoria da utilizzaew per il processo degli executor
- spark.executor.memoryOverhead: la memoria addizzionale che deve essere allocata per il processo degli executor

- spark.executor.cores: numero di core che devono utilizzare gli executor

Il numero di executor e la quantità di memoria per ogni executor indica al cluster quanti *Virtual Machine* sono ncessarie per il calcolo. È importante dosare il numero giusto di risorse quando lanciamo un job così da non sprecarle inutilmente.[3]

2.3 Calculator Module

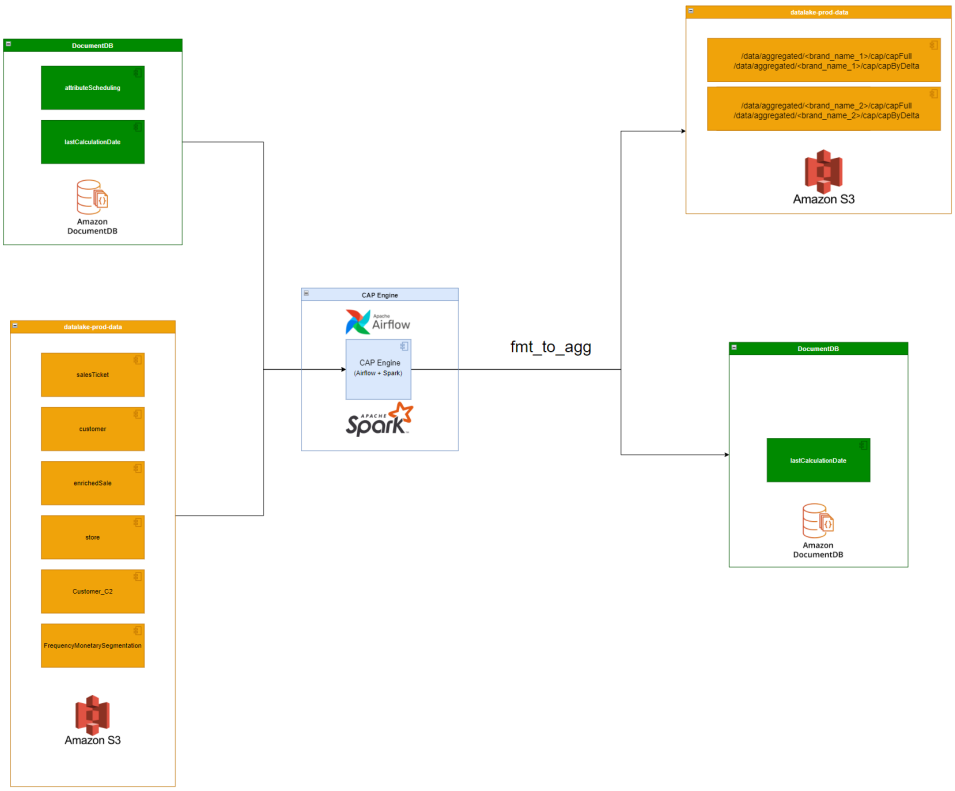

Figura 2.7: Calculator Module: Workflow

Nella figura 2.7 è rappresentato il calculator module. Il job *fmt_to_agg* è responsabile del calcolo dei KPI dei clienti e del loro upload su AWS S3, sia in modalità full, sia in quella delta. Il job segue i seguenti step:

1. legge le sorgenti dati (formatted layer), incluso i KPI calcolati nelle sessioni precedenti (aggregated layer), da AWS S3

2. legge le tabelle parametriche su DocumentDB: *lastCalculationDate* e *attributeScheduling*.

3. esegue il calcolo full per gli attributi e li scrive su AWS S3 (precisamente nel percorso dei calcoli full) come *CAPFull* aggregated layer

4. esegue un confronto tra l'attuale e il calcolo della precedente sessione per definire le variazioni (il delta), che vengono poi scritte su AWS S3 aggregate layer come *CAPByDelta*

5. aggiorna la tabella parametrica *lastCalculationDate* su DocumentDB

Nella successiva sezione è spiegato qual è la funzione delle tabelle parametriche e come esse sono strutturate. Quindi, sono date in seguito dettaglio aggiuntivi riguardo il job appena descritto e il data model *CAPFull/CAPByDelta*

2.3.1 attributeScheduling: Parametric Tabel

Il job è eseguito giornalmente per ogni brand, ma non tutti gli attributi sono calcolati ogni giorno. Per settare se un attributo deve essere calcolato in uno specifico job si utilizza una tabella parametrica su DocumentDB chiamata *attributeScheduling + brand_name*. Gli attributi dei clienti sono divisi in gruppi. Un *AttributeGroup* è un gruppo di attributi che dipendono gli uni dagli altri o che condividono una gran parte delle logiche di calcolo, per cui è meglio ottimizzare il calcolo processandoli tutti insieme. La tabella parametrica ha la struttura come nella fig. 2.1:

Field Name	**Type**
_id	Text
ActiveFlag	Bool
AttributeGroup	Text
AttributeName	Text
DayOfWeek	Text
Frequency	Text

Tabella 2.1: attributeScheduling: Data Model

- _id: è lo unique identifier (UUID) generato da DocumentDB
- ActiveFlag: può essere true o false, e viene utilizzato per attivare o disattivare lo scheduling del calcolo di un gruppo di attributi
- AttributeGroup: è nome del gruppo di attributi da calcolare

- AttributeName: è un array contenente il nome di tutti gli attributi da calcolare nel gruppo
- DayOfWeek: è valorizzato solo nel caso in cui il campo Frequency è settato a 'Weekly', e specifica quale giorno della settimana il calcolo deve essere processato per il gruppo di attributi
- Frequency: descrive la frequenza di calcolo del gruppo di attributi. Può essere'Daily' se il calcolo deve essere fatto giornalmente, o 'Weekly', se deve essere fatto una volta a settimana

2.3.2 lastCalculationDate: Parametric Table

Per poter produrre il calcolo in delta, abbiamo bisogno di confrontare l'attuale calcolo full con il calcolo dell'esecuzione precedente. Siccome non tutti gli attributi sono calcolati ogni giorno, è stato necessario creare una dedicata tabelle parametrica per tracciare l'ultima data di calcolo per ogni attributo e per ogni brand. La tabella parametrica ha la struttura come nella fig. 2.2:

Field Name	**Type**
_id	Text
AttributeGroup	Text
lastDateSynchronization	Text
lastDateCalculation	Text
lastDateExecution	Text

Tabella 2.2: lastCalculationDate: Data Model

- _id: è lo unique identifier (UUID) generato da DocumentDB
- AttributeGroup: il nome del gruppo di attributi calcolato
- lastDateCalculation: l'ultima data in cui è stato processato il calcolo del gruppo. È utile soprattuto per il gruppo di attributi che non sono calcolati giornalmente
- lastDateSynchronization: l'ultima data di sincronizzazione degli attributi. Per sincronizzazione si intende il processo di copia dell'ultimo calcolo processato per un gruppo nella cartella del giorno corrente su AWS S3. Questa procedura permette di aver a disposizione per tutti gli attributi, anche quelli in cui il calcolo non è schedulato, l'ultima versione disponibile dei KPI nella cartella della giornata corrente, requisito necessario per alcuni sistemi target fruitori dei dati calcolati da CAP.

- lastDateExecution: l'ultima data quando l'attributo è stato calcolato o sinconizzato. È il massimo valore tra lastDateCalculation e lastDateSynchronization.

Per permettere il calcolo di un nuovo attributo abbiamo bisogno di censire questa tabella con una nuova riga, settando il valore 'InitialLoad' nel parametro *lastDate*. Alla fine del job questo parametro verrà aggiornato con la data corrente per tutti gli *AttributeGroup* calcolati.

2.3.3 Spark job: fmt_to_agg

Questo job calcola tutti gli attributi dei clienti. Ogni attributo è calcolato con una sua logica e viene storicizzato nei *CAPFull* e *CAPByDelta* dataset. Il modello di questi dataset è descritto nella fig. 2.3:

Field Name	**Type**
_id	Text
Brand	Text
AttributeName	Text
AttributeValue	Text
CustomerId	Text
Timestamp	Datetime
TriggerTimestamp	Datetime
CurrentValue	Boolean

Tabella 2.3: CAPFull & CAPByDelta: Data Model

- _id: è lo unique identifier (UUID) generato dal job come sha1 della concatenazione dei campi: Brand, AttributeName, AttributeValue, CustomerId, Timestamp. Questo formato standard è utilizzato per permettere il processo di UPDATE su chiave.
- Brand: il nome del brand
- AttributeName: il nome dell'attributo calcololato. (*preferred_store*, *segmentation*, *contactable_email_address_flag* ...)
- AttributeValue: valore dell'attributo calcolato
- CustomerId: è l'ID che identifica univocamente il cliente
- Timestamp: timestamp corrispondente al momento di primo calcolo dell'attributo

- TriggerTimestamp: di default corrisponde al timestamp di calcolo. Potrebbe essere aggiornato con un timestamp più recente per indicare che il record non è più valido da quella data
- CurrentValue: se è valorizzato true indica che il valore dell'attributo è attualmente valido, altrimenti indica che quel valore non è più valido

Questo data model è utilizzato per avere per ogni cliente tutta la storia dei suoi attributi. Ci dà la possibilità di controllare in modo semplice tutte le variazione che ci sono state per un attributo e in quale data ci sono state. Questo data model viene alimentato dal job Spark fmt_to_agg, i quali step sono descritti nella fig. 2.8.

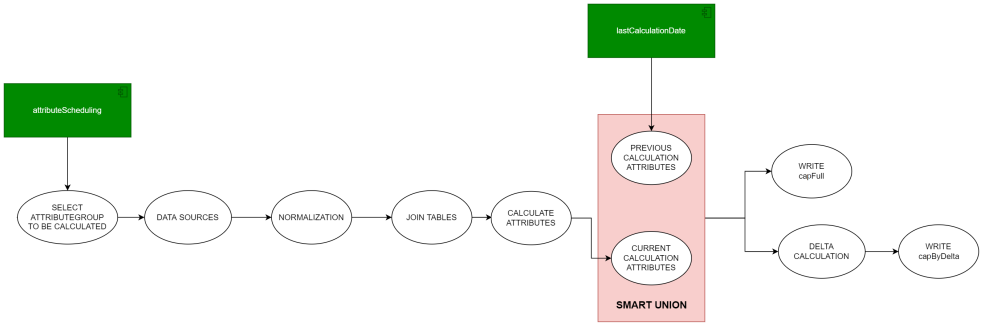

Figura 2.8: fmt_to_agg Step

- Select *AttributeGroup* to be calculated: viene letta la tabella parametrica *attributeScheduling* per identificare quali sono i gruppi di attributi da calcolare
- Data Sources: in questo step viene fatto l'accesso a tutti i dati necessari per il calcolo degli attributi, ossia i dati sorgenti in AWS S3 formatted e i calcoli delle esecuzioni precedenti in AWS S3 aggregated.
- Normalization: viene fatta una normalizzazione dei dati sorgenti, applicando le trasformazioni necessarie per preparare le strutture sorgenti al calcolo e alle join.
- Join Table: le tabelle normalizzate vengono unite tra loro tramite join
- Calculate Attributes: viene fatto il calcolo degli attributi per ogni cliente
- Current Calculation Attributes: è l'output dello step precedente
- Previous Calculation Attributes: viene recuperato l'output dell'ultimo calcolo processato, come riportato sulla tabella parametrica *lastCalculationDate*

- Smart Union: è un processo che ritorna come risultato l'attuale calcolo, compreso della parte del calcolo precedente da invalidare (in quanto il valore dell'attributo di un cliente è variato rispetto l'ultimo calcolo). In questo step viene aggiornato il *TriggerTimestamp* e *CurrentValue* per la vecchia versione dell'attributo da invalidare. Lo step di smart union scrive poi il risultato finale nel percorso AWS S3 *CAPFull*.

- Delta Calculation: in questo step viene calcolato il dataset *CAPByDelta*, ossia una porzione di dati estratta dal *CAPFull*, selezionando tutti gli attributi aventi un *TriggerTimestamp* pari alla data corrente di calcolo.

Negli ultimi due step scriviamo quindi l'output del calcolo su AWS S3 nell'aggregated layer del Datalake. Il delta e il full sono calcolati e scritti su file parquet, in percorsi dedicati con path *data/aggregated/+brand_name+/client/* seguito da:

- *CAPFull/runDate/attributeName=attribute_name*

- *CAPByDelta/date=runDate/attributeName=attribute_name*

La struttura dei path permette di facilitare l'accesso a tutti gli attributi o a un solo sottoinsieme. Infatti possiamo leggere tutti gli attributi accedendo al percorso *CAPFull/runDate/** oppure ai singoli attributi usando *CAPFull/runDate/attributeName=attribute_name*.

2.4 attributeRouting: Parametric Table

La tabella parametrica *attributeRouting* ha due obiettivi. Il primo è dare una fotografia riassuntiva attuale di quali attributi sono calcolati specificando quali sono i sistemi target per cui essi sono calcolati. Il secondo è permettere di configurare dinamicamente, senza dover quindi cambiare codice sorgente dell'applicazione, le interazioni tra il Calculator Module e gli Integrator e Denormalizer module, semplifcando la manutenzione degli stessi. La tabella parametrica permette in breve di propagare su uno dei possibili canali target uno specifico attributo: si minimizza quindi l'effort di sviluppo e manutenzione del codice quando un attributo già calcolato viene richiesto da un nuovo sistema target. La tabella è utilizzata dai job di moduli di Integrator e Denormalizer module per selezionare, applicare il cast del data type e rinominare gli attributi che sono presenti nei dataset *CAPFull* e *CAPByDelta*. Il modello dati di questa tabella parametrica è nella fig. 2.4

Field Name	Type
_id	Text
brandDesc	Text
brandCode	Text
attributeName	Text
deprecationFlag	Text
deprecationDate	Datetime
isFastGrowing	Datetime
targetSystems	Array[Struct]
targetJobs	Array[Struct]
type	Text

Tabella 2.4: attributeRouting: Data Model

- _id: è lo unique identifier (UUID).
- brandDesc: breve descrizione del brand
- brandCode: codice del brand
- attributeName: nome dell'attributo
- deprecationFlag: flag che indica se l'attributo è ancora utilizzato
- deprecationDate: indica, in caso l'attributo non sia più utilizzato, la data di ultimo utilizzo
- isFastGrowing: assume valore Y se la frequenza di variazione dei valori nella storia di un cliente è alta, altrimenti assume valore N
- targetSystems: indica per quali sistemi target l'attributo è attualmente attivo. Non è singole valore, ma è un array di strutture, ciascuna delle quali ha i seguenti campi: *targetName*, *attributeName*, *isCurrent*
- targetJobs: indica se l'attributo deve essere selezaionato nei job dell'Integrator e del Denormalizer module. Questo campo è un array di strutture, con lo stesso modello del campo targetSystems
- type: indica il data type dell'attributo

In generale, per ogni target job, potrebbero esserci più target system. Per esempio: abbiamo un target job che genera il *customerAttributes* in the datalake, che però può essere letto da diversi sistemi target, ognuno con una sua naming convention, proprio come censito nella tabella parametrica.

2.5 Integrator Module

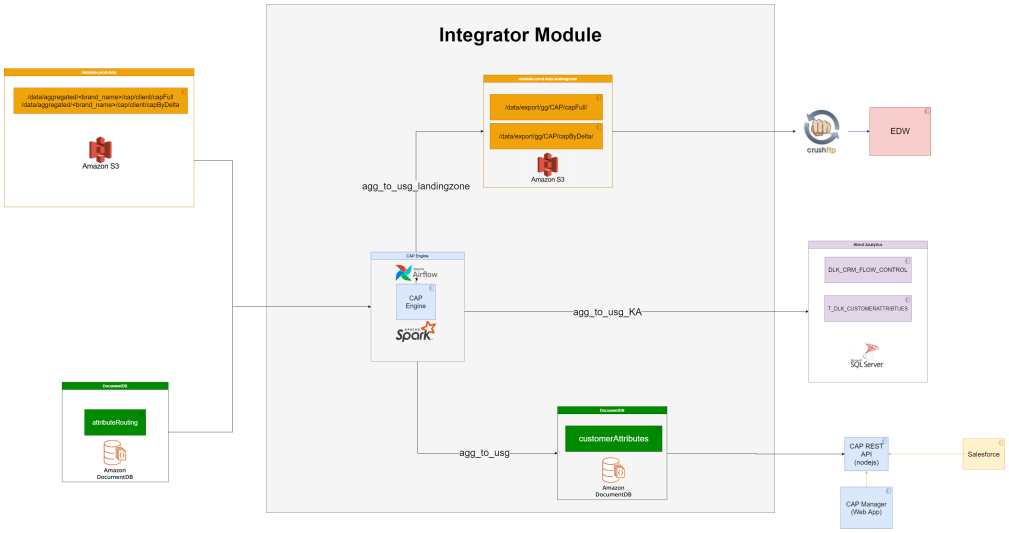

Figura 2.9: Integrator Module: Workflow

Nella fig. 2.9 è rappresentato l' Integrator module. Il modulo è costituito da tre diversi job, ognuno responsabile di inserire e aggiornare dati in diversi database/sistemi. I System Owner di questi sistemi hanno richiesto solo alcuni degli attributi calcolati da CAP, quindi la gestione delle singole integrazioni viene fatta configurando opportunamente la tabella parametrica attributeRouting.

2.5.1 Spark job: agg_to_usg

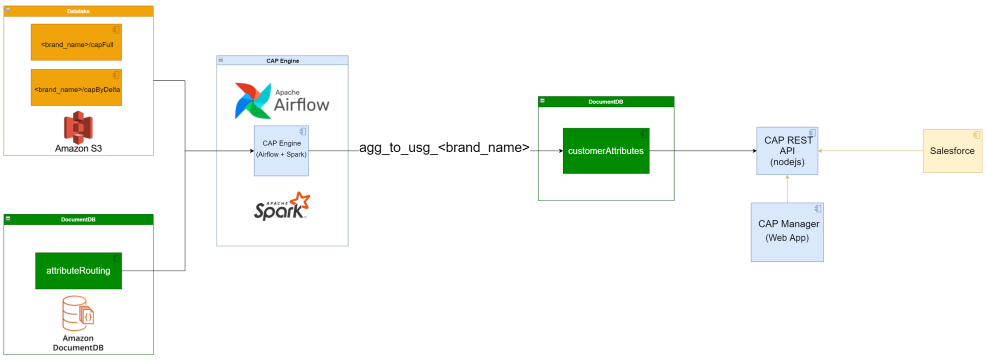

Figura 2.10: agg_to_usg Flow

Il job nella fig. 2.10 ha l'obiettivo di aggiornare la collection *customerAttributes* su DocumentDB. Esiste un job per ogni brand e la prima esecuzione di questo job comporta un inserimento full su DocumentDB. Giornalmente vengono prodotte delle variazioni rispetto al calcolo del giorno precedente, per cui, dopo la prima esecuzione, solo il delta verrà inserito sulla collection, tramite una UPSERT. Tutti gli step sono rappresentati nel seguente schema: 2.11.

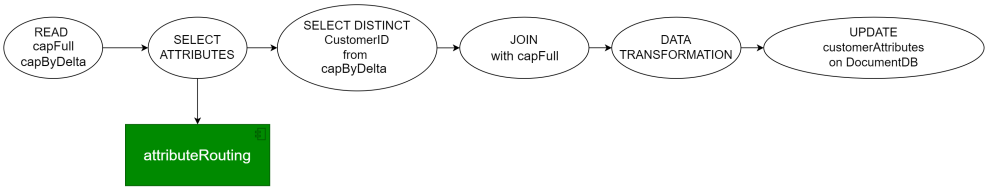

Figura 2.11: agg_to_usg Steps

L'importante step di questo job è la data trasformation, in quanto riformatta i dati *CAPFull* in modo da mantenere la storia di tutti i valori di un attributo per ogni cliente, secondo il modello dati della fig. 2.5:

Field Name	**Type**
_id	Text
Brand	Text
insertDate	Datetime
customerId	Text
attributes	Array[Struct]
oldAttributes	Array[Struct]
operationFlag	Text

Tabella 2.5: customerAttributes: Data Model

- _id: è lo unique identifier (UUID) generato come lo sha1 della contenazione di CustomerId e Brand. È utilizzato come indice per permettere la procedura di UPSERT sulla collection
- Brand: il nome del brand
- insertDate: la data di inserimento del primo attributo calcolato per un cliente
- customerId: l'ID che identifica univocamente un cliente
- attributes: i valori correnti di tutti gli attributi calcolati per quel cliente (*CurrentValue = true da CAPFull*) con la struttura: *name, value, lastUpdateDate*.

- oldAttributes: i valori storici degli attributi calcolati da un clientr (*CurrentValue = false da CAPFull*) con la struttura: *name, value, lastUpdateDate.*
- operationFlag:ultima operazione fatta sul record. Può assumere i valori: I (insert), U (update), D (delete).

Con questa struttura quindi, a differenza del modello CAPFull, abbiamo un solo document per ogni cliente e la storicizzazione degli attributi è gestita nel campo oldAttributes.

CAP REST API & Manager

I dati contenuti nella collection di DocumentDB sono esposti tramite le CAP REST API. È quindi possibile recuperare il valore degli attributi dal CAP Engine via richiesta HTTP, cercando il cliente per nome o per id. Le API sono usate dalle applicazioni di Salesforce e dal CAP Manager, che è una applicazione web che visualizza gli attributi di tutti i clienti per tutti i brand.

2.5.2 Spark job: agg_to_usg_KA

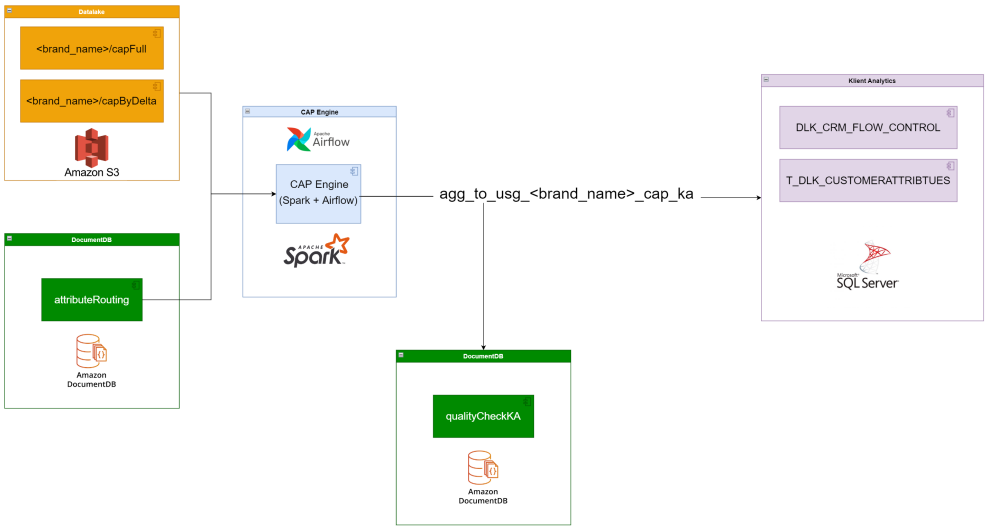

Figura 2.12: agg_to_usg_KA Flow

Il flusso mostrato nella fig. 2.12 carica gli attributi dei clienti nel database di Klient Analytics, che è un tool di reportistica sviluppato con Microstrategy. Questo flusso legge i dati *CAPFull* e *CAPByDelta*, che va poi a scrivere in una tabella *MSQL*

SERVER chiamata *T_DLK_CUSTOMERATTRIBUTES*. Il job scrive prima e dopo la scrittura degli attributi anche sulla tabella semaforica *DLK_CRM_FLOW_CONTROL*, per loggare i timestamp di inizio e fine flusso, indicando il brand e il numero di record inseriti e aggiornati.

Nella figura 2.13 sono riportati tutti gli step che il flusso esegue:

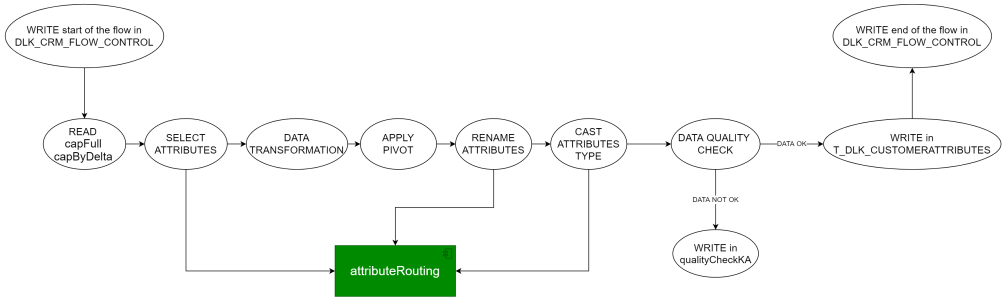

Figura 2.13: agg_to_usg_KA Steps

1. viene scritto un record di "inizio flusso" nella tabella semaforica *DLK_CRM_FLOW_CONTROL* valorizzando con il timestamp di inizio flusso il campo *Transfer_id*, il quale identifica univocamente il trasferimento del dato su KlientAnalytics

2. legge i dati su *CAPFull* e *CAPByDelta*

3. controlla sulla tabella parametrica *attributeRouting* quali attributi devono essere propagati su Klient Analytics

4. viene aggiunto il valore *Transfer_id* alla struttura dati di scrittura e viene rinominato il *Brand* seguendo una naming convention

5. applica una Pivot sui dati sorgenti. La Pivot permette di accorpare in un'unica riga più record sorgenti, andando a valorizzare su più colonne il dato che fa riferimento alla stessa entità. In questo caso quindi applichiamo la Pivot su *AttributeName raggruppando per CustomerId, Brand, TransferId, ed aggregando su AttributeValue*. Il risultato è che abbiamo una colonna per *AttributeName* avente *AttributeValue* come valore.

6. viene recuperato dalla tabella *attributeRouting* l'informazione di come rinominare l'attributo

7. siccome sui dataset *CAPFull* e *CAPByDelta* utilizziamo solo campi di tipo stringa in questo step applichiamo anche il cast del data type per il campo *AttributeValue*, come descritto sulla tabella *attributeRouting*

8. viene fatto uno step di data quality per verificare che il dato sia conforme a delle regole di business
9. se il dato non è valido per le regole di business, esso viene scritto su una collection dedicata su DocumentDB chiamata *qualityCheckKA*.
10. se il dato è valido per le regole di business viene scrito nella tabella MSQL: *T_DLK_CUSTOMERATTRIBUTES*

Il data model della tabella *T_DLK_CUSTOMERATTRIBUTES* è mostrato nel seguente schema 2.6.

FieldName	**Type**
Brand	Text
TransferId	Text
CustomerId	Text
attributes	one attribute for each column

Tabella 2.6: T_DLK_CUSTOMERATTRIBUTES: Data Model

- Brand: il nome del brand
- TransferId: l'ID del trasferimento dati, necessario per fare AUDITING sul flusso
- CustomerId: l'ID che identifica univocamente il cliente
- attributes: c'è una colonna aggiuntiva per ogni attributo calcolato per Klient Analytics, per esempio *preferred_store*, *segmentation*, ecc...

Control Flow Table

In questa tabella è tracciato lo stato di ogni job che scrive su KlientAnalytics, il cui data model segue lo schema 2.7:

FieldName	**Type**
TRANSFER_ID	Text
STATUS	Text
FLOW_NAME	Text
LOAD_START_DATETIME	DateTime
LOAD_END_DATETIME	DateTime
PROCESSING_START_DATETIME	DateTime
PROCESSING_END_DATETIME	DateTime

Tabella 2.7: DLK_CRM_FLOW_CONTROL: Data Model

All'inizio del job viene scritto un record con i parametri settati come sotto:

- TRANSFER_ID: primary key e identificatore del batch di trasferimento
- STATUS: 'LOADING IN PROGRESS'
- FLOW_NAME: 'CUSTOMER_ATTRIBUTES'
- LOAD_START_DATETIME: timestamp di inizio job
- LOAD_END_ENDTIME: blank
- PROCESSING_START_DATETIME: blank
- PROCESSING_END_DATETIME: blank

Alla fine del job lo stesso record viene aggiornato come segue:

- STATUS: 'LOADING FINISHED' o 'LOADING ERROR'
- LOAD_END_ENDTIME: timestamp della fine del job

Gli ultimi due campi saranno valorizzati dal job del sistema target quando farà l'ingestion dei dati dalla tabella di frontiera su cui CAP ha scritto i dati.

Data Quality Check

È stato aggiunto un controllo di data qualilty su i campi *Brand* e *CustomerId*:

- length(*Brand*) less or equal than 10
- length(*CustomerId*) less or equal than 25

Se una delle due condizione non è rispettata, i record sono inviata alla collection *qualityCheckKA* di DocumentDB, che condivide la stessa struttura della tabella customerAttributes, con in aggiunta il campo *CheckTimestamp*, sul quale è valorizzato il timestamp in cui il record è stato inserito sulla tabella degli scarti.

2.5.3 Spark job: agg_to_usg_landingzone

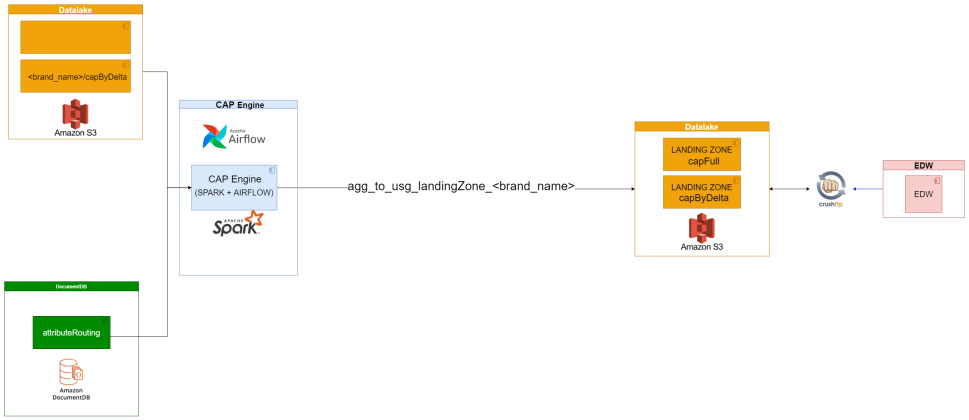

Figura 2.14: agg_to_usg_landingzone Flow

Nel job rappresentato nella figura2.14 vengono letti i dati da *CAPFull* e *CAPByDelta*, sul quale applichiamo alcune trasformazioni, prima di scriverli sul layer di landingzone del datalake. Questo layer è utilizzato per permettere l'export dei dati verso quei sistemi che possono direttamente connettersi tramite *CrushFTP*, tra cui l'*Enterprise Data Warehouse*. Il data model della tabella esportata verso *EDW* segue lo schema 2.8:

FieldName	**Type**
customer_id	Text
brand_desc	Text
brand_code	Text
operation_flag	Text
update_date	DateTime
attributes	one attribute for each column

Tabella 2.8: CAP landingzone: Data Model

- customer_id: è il *CustomerId* recuperato da *CAPFull*
- brand_desc: descrizione breve del brand
- brand_code: codice identificativo del brand
- operation_flag: l'ultima operazione processata sul record. Può assumere i valori I (insert), U (update), D (delete)

- update_date: ultima data di aggiornamento degli attributi del cliente
- attributes: c'è una colonna aggiuntiva per ogni attributo calcolato ed esposto a EDW, per esempio *preferred_store*, *segmentation*, ecc...

Gli step richiesti per produrre dati su questo modello dati sono rappresentati nella seguente fig. 2.15

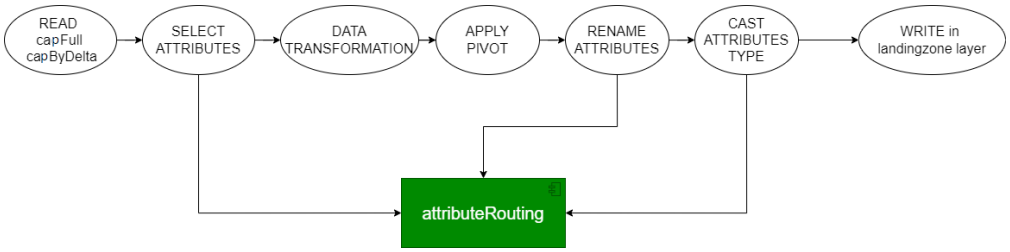

Figura 2.15: agg_to_usg_landingzone Steps

2.6 Denormalizer Module: Workflow

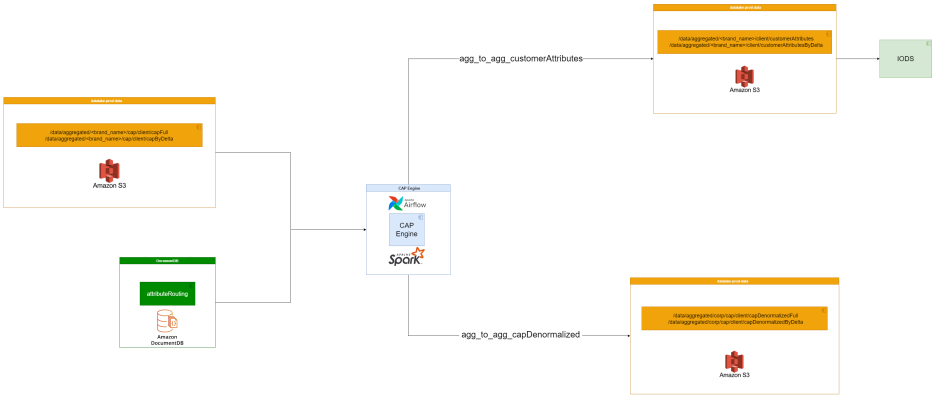

Figura 2.16: Denormalizer Module: Workflow

La figura 2.16 mostra il modulo Denormalizer. Esso è composto da due job che leggono i dati da *CAPFull* e *CAPByDelta* nel layer aggregated. I dati vengono resi disponibili per ogni team/sistema, in modo che siano leggibili e utilizzabile rispetto alle diverse esigenze di ogni user.

2.6.1 Spark job: agg_to_agg_customerAttributes

Questo job viene eseguito ogni giorno per ogni brand, il quale legge i dati da *CAPFull* e *CAPByDelta* e genera il dataset *customerAttributes*, che è letto da IODS in modo che sia reso disponibile alle applicazioni frontend utilizzate in tutti gli store della holding. Il job recupera dalla tabella parametrica *attributeRouting* tutti gli attributi che devono essere inclusi sulla struttura esposta a IODS. Il data model di questa struttura segue lo schema 2.9:

FieldName	**Type**
client_id	Text
brand	Text
brand_code	Text
date_calculation	DateTime
attributes	one attribute for each column

Tabella 2.9: customerAttributes: Data Model

- client_id: *CustomerId* delle tabelle di CAP
- brand: breve descrizione del brand
- brand_code: codice identificativo del brand
- date_calculation: data di ultimo calcolo per il cliente (*max TriggerTimestamp per ogni CustomerId*)
- attributes: c'è una colonna aggiuntiva per ogni attributo calcolato ed esposto a EDW, per esempio *preferred_store*, *segmentation*, ecc...

Per ottenere questo tipo di dataset il job segue gli step della seguente fig. 2.17

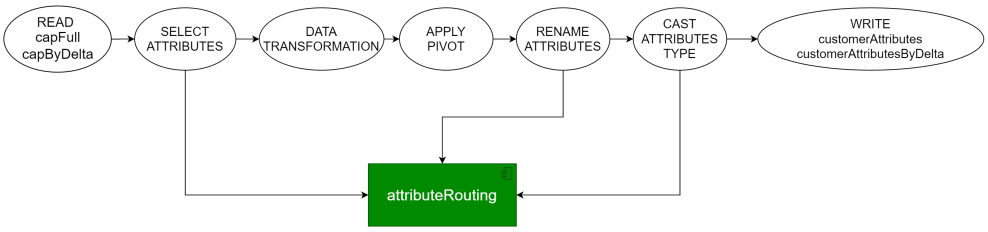

Figura 2.17: agg_to_agg_customerAttributes Steps

1. legge i dati *CAPFull* e *CAPByDelta*

2. verifica sulla tabella parametrica *attributeRouting* quali attributi devono essere selezionati

3. Data Transformation:

   (a) Renaming: *CustomerId* → *client_id*; *Brand* → *brand*

   (b) Aggiunta dei campi *brand_code* usando una naming convenction

   (c) Calcolo del campo *date_calculation* come *max(TriggerTimestamp)* per ogni cliente

4. applica una Pivot sui dati sorgenti. La Pivot permette di accorpare in un'unica riga più record sorgenti, andando a valorizzare su più colonne il dato che fa riferimento alla stessa entità. In questo caso quindi applichiamo la Pivot su *AttributeName raggruppando per CustomerId, Brand, TransferId, ed aggregando su AttributeValue*. Il risultato è che abbiamo una colonna per *AttributeName* avente *AttributeValue* come valore.

5. viene recuperato dalla tabella *attributeRouting* l'informazione di come rinominare l'attributo

6. siccome sui dataset *CAPFull* e *CAPByDelta* utilizziamo solo campi di tipo stringa in questo step applichiamo anche il cast del data type per il campo *AttributeValue*, come descritto sulla tabella *attributeRouting*

7. viene scritto il full e il delta della tabella *customerAttributes* sull'aggregated layer del Datalake.

2.6.2 Spark job: agg_to_agg_CAPDenormalized

Questo job legge i dati di *CAPFull* e *CAPByDelta* e genera il dataset *CAPDenormalized*. La principale differenza dal job precedente è che legge il dataset di CAP di tutti i brand e li unisce sotto un nuovo dataset con lo stesso modello dati e con gli stessi step del job *agg_to_usg*. Viene prodotto quindi un unico dataset su S3 contenente i dati prodotti da CAP per tutti i brand, in modo che sia disponibile al team Datalake.

Capitolo 3

Attributi dei clienti

Gli attributi dei clienti calcolati in CAP possono essere divisi in diverse categorie. In questo capitolo è presentata un overview di ogni categoria e le logiche dietro il loro calcolo. In generale, per lo sviluppo di ogni attributo sono stati seguiti i seguenti step:

- Analisi dei requisiti
- Comprensione del significato funzionale dei dati
- Traduzione dei requisiti in un diagramma ad albero decisionali, o direttamente in linguaggio SQL per gli attributi caratterizzati da logiche di calcolo più semplici
- Validazione del codice SQL o dell'albero decisionale con l'utente finale
- Traduzione del codice SQL o dell'albero decisionale in linguaggio Scala per Spark
- Estrazione dei dati di test
- Test interni
- Test con l'utente
- Calcolo degli attributi in *CAPFull* e in *CAPByDelta*
- Propagazione degli attributi calcolati verso il sistema target

Nel flusso di lavoro, in caso di correzioni o rework necessari, è sempre possibile ritornare agli step precedenti per rivedere e modificare gli algoritmi di calcolo.

3.1 Categorie degli attributi

Gli attributi sono divisi in quattro categorie principali:

- Preferenze di contatto e privacy: informazioni sui metodi di contatto del cliente (ad esempio, indirizzo e-mail o numero di telefono) e preferenze correlate (ad esempio, se i metodi di contatto sono contattabili, per quale utilizzo, in quale periodo di tempo preferito)
- Preferenze di clientela e fidelizzazione: informazioni su come il cliente preferisce interagire con il marchio. Include principalmente entità di preferenza (ad esempio, negozio preferito e assistente alle vendite preferito).
- Qualità dei dati e tracciabilità: informazioni di verifica sui dati dei clienti e sulla qualità dei dati (ad esempio, flag, metriche, log di pulizia dei dati), include principalmente attributi relativi alla contattabilità e ai metodi di contatto validi (ad esempio, validità del contatto via telefono, validità del contatto via mail).
- Informazioni di profilazione: qualsiasi aggregazione (ad esempio numero di acquisti negli ultimi "n" mesi) o categoria (ad esempio, tag, segmentazione del cliente, cluster).

In base alle regole di business da seguire, il livello di complessità degli algoritmi di calcolo può variare notevolmente. Di seguito è presentato, per ogni categoria, un attributo di esempio, concentrando il focus sul processo di sviluppo seguito su CAP.

3.2 Attributo *contactable email address flag*

L'attributo *contactable_email_address_flag* fa parte della categoria *Preferenze di contatto e privacy*. Indica se un cliente può essere contattato tramite e-mail, tenendo conto del suo indirizzo e-mail valido e del consenso al marketing. L'indirizzo e-mail operativo considererà tutti gli aspetti necessari per contattare i clienti con il mezzo appropriato e il relativo consenso. Questo attributo contribuisce a garantire l'affidabilità dei dati dei clienti.

3.2.1 Regola di business

In questo caso la regola è stata condivisa direttamente dall'utente finale, tramite il seguente tracciato:

```
IF {

    Email is valid
    AND Consent for Marketing = 'Y'
    AND Contact via Email = 'Y'

}
THEN contactable_email_address_flag = 'Y'
ELSE contactable_email_address_flag = 'N'

Email is valid - Email contains @ and .
```

3.2.2 Informazioni di input

Il primo passo è capire quali sono le fonti di dati e i campi necessari per il calcolo. Per questo attributo, tutti i campi di input sono contenuti nel dataset dell'anagrafica cliente disponibile nel datalake, chiamata "customer". Essi sono:

- La "Email" è contenuta nel campo *email.address*
- "Consent for Marketing" è contenuto nel campo *typology.flg.prv_mark*
- "Contact via Email" è contenuto nel campo *contact.flg.email*

La notazione utilizzata per il mapping *structure.element* descrive la posizione precisa di dove il dato è salvato nel dataset. I dati sorgenti sono sempre storicizzati in file strutturati, i cui campi, a differenza di strutture tradizionali tabellari, possono essere a loro volta strutture o array. Una volta che il mapping è stato riscritto in linguaggio SQL, esso viene tradotto in linguaggio Scala per poter essere calcolato tramite Spark.

3.2.3 CAP Output

Un volta che il calcolo è stato lanciato e concluso, l'output viene scritto nei dataset *CAPFull* e *CAPByDelta*. Un esempio è mostrato nella figura 3.1

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	Y	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	N	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 3.1: *contactable_email_address_flag*: Output in CAPFull & CAPByDelta

3.3 Attributo *preferred store*

L'attributo *preferred_store* fa parte degli attributi di categoria *Preferenze di clientela e fidelizzazione*. Il preferred store è definito considerando il comportamento di acquisto del cliente e un set di regole predefinite. Questa informazione è utilizzata per le strategie di marketing e di contatto del cliente, determinando principalmente quale store dovrebbe contattare il cliente. È uno degli attributi con il numero più elevato di logiche complesse. Ogni brand definisce delle regole di definizione del preferred store diverse. Sotto sono riportate le logiche richieste di uno dei brand della holding. Per dare un visione più chiara di insieme, in fondo a questa sezione, è riportato l'albero decisionale che semplifica la lettura e la comprensione dell'algoritmo.

3.3.1 Regola di business

La procedura considera solo i clienti per i quali il campo "preferred_store_flag" dell'anagrafica clienti è settato ad "N". Il flag a N indica in particolare che il preferred store non è stato assegnato manualmente, e che quindi deve essere calcolato secondo le logiche condivise dal business. Lo store preferito deve innanzitutto soddisfare i seguenti criteri:

- STORE TYPE: Per questo brand in particolare, tutti i tipi di store devono essere calcolati, tra cui i regular, gli outlet e gli e-commerce.
- STORE STATUS: Devono essere considerati solo gli ACTIVE store, ossia i negozi attualmente in attività. Nel caso in cui un cliente abbia fatto acquisti in solo CLOSED store, verrà assegnato come store preferito uno store di default.
- STORE COUNTRY: Il paese dello store deve coincide con il paese di residenza del cliente. Se il cliente non ha fatto nessun acquisto nel suo paese di residenza, nel calcolo devono essere considerati gli store di tutti i paesi. La regola deve seguire quindi i due possibili scenari:
  - Local client: Se esiste almeno una transazione fatta nel paese di residenza del cliente, la procedura di calcolo del preferred store deve considerare solo le transazioni fatte nel paese di residenza.
  - Tourist client: Se il cliente non ha nessuna transazione nel suo paese di residenza, allora per il calcolo del preferred store bisogna considerare tutte le transazioni fatte nel mondo.

Considerando tutti gli store che soddisfano questi requisiti, bisogna applicare le seguenti regole in sequenza:

1. FREQUENCY OF VISITS: Il sistema considera il numero di visite del cliente per ogni store. Il numero di visite del cliente viene calcolato contando tra tutte le transazioni del cliente il numero di triplette 'CUSTOMER ID'; 'STORE CODE'; 'TRANSACTION DATE' distinte, ignorando le transazioni ITEM LINES TRANSACTION TYPE = RETURN. Sotto questa regola, se un cliente fa due o più acquisti diversi (con quindi due o più scontrini fiscali) nella stessa giornata, il numero di visite viene contato una sola volta, ignorando in ogni caso le transazioni di rimborso. Se un cliente nel periodo di analisi, ha solo transazioni di rimborso viene assegnato lo store di default come preferred store.

2. TOTAL SPENDING VALUE: Se la frequenza di visite è uguale tra due o più store, il sistema considera la spesa totale netta (ossia la spesa totale, senza IVA e sconti). Nel conteggio sono considerato sia le transazioni di vendita che di rimborso (ovviamente con simbolo negativo).

3. STORE RECENCY: Se anche la spesa totale netta è uguale tra due o più store, il sistema considera lo store dove, tra quelli rimasti, il cliente ha fatto la transazione più recente. La regola deve considerare solo le transazioni di acquisto, e non di rimborso.

In caso di logiche complesse, il primo step è tradurre i requisiti in un diagramma ad albero decisionale. Nella figura 3.3 è mostrato l'albero decisionale in cui i requisiti precedentemente descritti sono stati tradotti. Ogni divisione nell'albero decisionale è mutualmente esclusiva, quindi ogni cliente può finire in una e una sola foglia dell'albero. L'albero decisionale ha regole commerciali aggiuntive che non sono state riportate esplicitamente, sia per privacy aziendali, sia per facilitare la comprensione al lettore.

3.3.2 Informazioni di input

Per il calcolo di questo attributo, tre diversi dataset sono stati necessari: anagrafica clienti, scontrini fiscali (transazioni di acquisto e rimborso) e anagrafica degli store. Dall'anagrafica cliente sono stati utilizzati i campi:

- *client_id*: identifica il cliente
- *activity.registr_store*: store in cui il cliente è stato registrato per la prima volta

- *pref_store.flg*: il flag che indica se il preferred store è stato già assegnato manualmente
- *typology.flg.emp*: il flag che indica se il cliente è un dipendente dell'organizzazione
- *typology.flg.unknown*: il flag che indica se un cliente è stato anonimizzato
- *typology.flg.prsp*: il flag che indica se il cliente è un "prospect", ossia un potenziali cliente che non ha ancora fatto nessun acquisto
- *client_address_country*: indica qual è il paese di residenza del cliente

Dal dataset degli scontrini fiscali sono stati utilizzati i campi:

- *business_date*: indica in quale data è stato fatto l'acquisto (si differisce dal campo transaction_date, in quanto è esente dall'informazione di fuso orario, che dipende dalla regione in cui l'acquisto è stato fatto),
- *client_id*: identifica il cliente a cui la transazione è assegnata
- *ticket_lines.net_amt*: indica l'ammontare netto speso dal cliente
- *sale_flg*: indica se lo scontrino è uno scontrino di rimborso (uno scontrino è uno scontrino di rimborso, se tutte le transazioni a livello di riga, e cioè a livello di articolo, sono rimborsi)
- *store_code*: identifica in quale store è stato fatto l'acquisto

Dal dataset di anagrafica store sono stati utilizzati i campi:

- *id*: identifica la store
- *store_status_code*: indica se lo store è in attività o è stato chiuso
- *store_subtype_code*: indica il tipo di store, che può essere: e-commerce, outlet, retail (o regular)

Una volta letti i dataset sorgenti, vengono applicate le join e le aggregazioni per calcolare la frequenza, la spesa aggregata e a quale store è associata l'ultima transazione.

3.3.3 CAP Output

In base alle regole di business e gli algoritmi di calcolo tradotti in Scala per Spark, per ogni cliente viene ogni giorno assegnato, o ricalcolato, il preferred store. Un esempio di dati in output sono mostrati nella fig. 3.2

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	12345	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	67893	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 3.2: *preferred_store*: Output in CAPFull & CAPByDelta

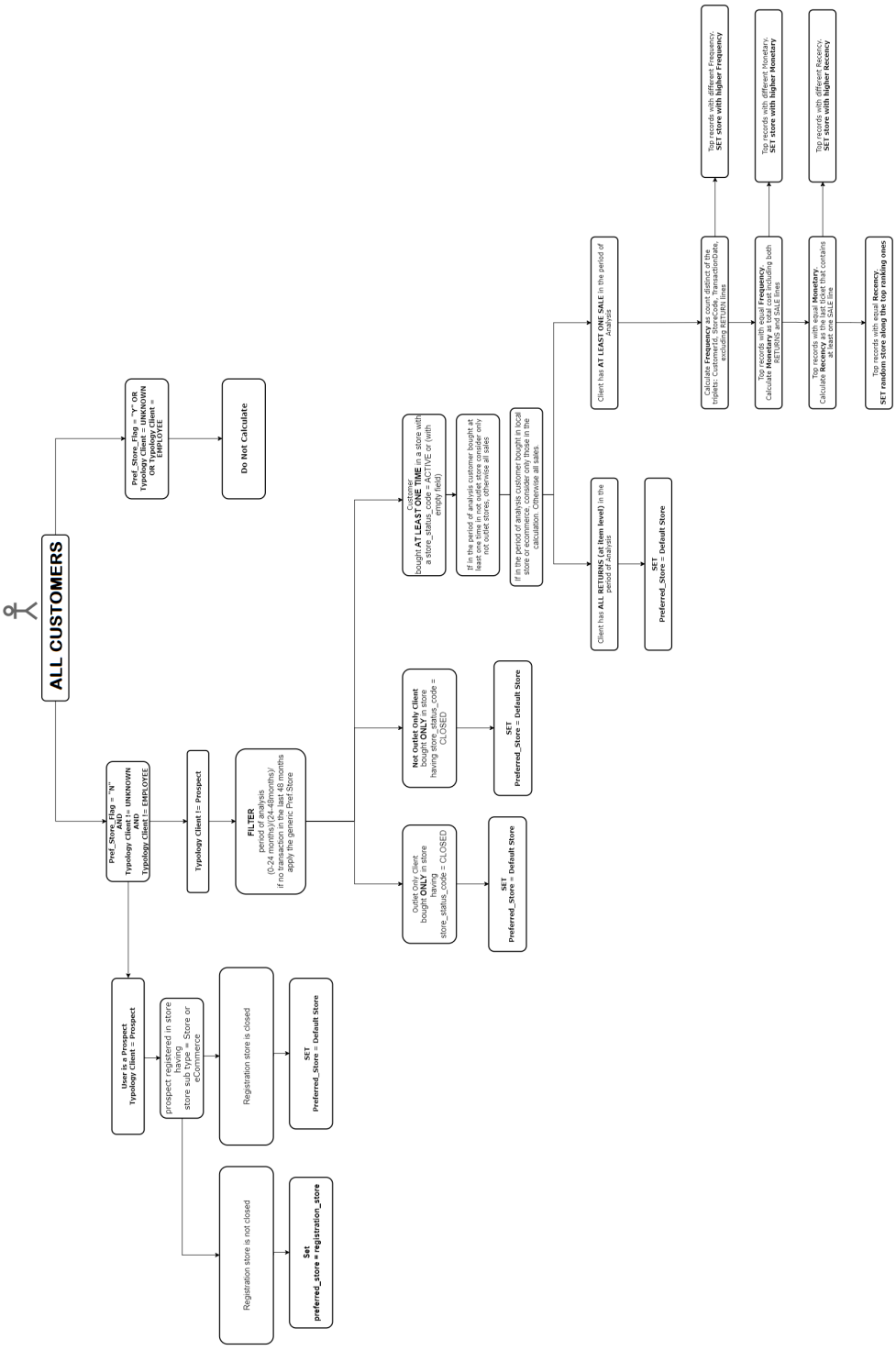

Figura 3.3: *preferred store*: decision tree

3.4 Attributo *gender title common consistent flag*

L'attributo *gender_title_common_consistent_flag* fa parte della categoria *Qualità dei dati e tracciabilità*. Indica se il titolo del cliente è consistente rispetto il genere riportato in anagrafica. Nella tabella 3.1 tutte le combinazioni che rende consistente il titolo di un cliente rispetto il suo genere:

Gender	Title
M (Male)	MR.
F (Female)	MISS. MS. MRS.
N (Neutral)	PREFER NOT SAY
C (Company)	COMPANY
A (Absent)	ABSENT

Tabella 3.1: Gender Title Mapping

3.4.1 Regola di business

In questo caso la regola di business è stata condivisa direttamente tramite il seguente tracciato:

```
IF {

Gender = 'M' AND Title (Common) != 'MR.'
OR Gender = 'F' AND Title (Common) IS NOT IN ('MISS.', 'MS.', 'MRS.')
OR Gender = 'N' AND Title (Common) != 'PREFER NOT SAY'
OR Gender = 'C' AND Title (Common) != 'COMPANY'
OR Gender = 'A' AND Title (Common) != 'ABSENT'

}
THEN Gender and Title (Common) Consistent Flag == 'N'
ELSE Gender and Title (Common) Consistent Flag == 'Y'
```

3.4.2 Informazioni di input

Per questo attributo abbiamo bisogno solo di recuperare due campi del dataset dell'anagrafica clienti:

- Gender è valorizzato nel campo *gender*
- Title (Common) è valorizzato nel campo *name.per_title*

Come per gli altri attributi con logiche poco complesse, la regola prima di essere scritta in linguaggio Scala è stata semplicemente tradotta in linguaggio SQL per validazione, senza rappresentarla su diagramma.

3.4.3 CAP Output

Il dato è scritto nei dataset *CAPFull* e *CAPByDelta* come nella figura 3.4.

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	Y	gender_title_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	N	gender_title_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 3.4: *gender title common consistent flag*: Output in CAPFull & CAPByDelta

3.5 Attributo *segmentation*

La *segmentation* fa parte della categoria *Informazioni di profilazione*. L'attributo serve per assegnare al cliente un determinato valore di segmentazione, ossia un attributo che riassume in una categoria la capacità (in termini quantitativi) di acquisto di un cliente. La segmentazione è basata su alcuni criteri relativi all'acquisto del cliente e sulla finestra temporale di analisi (che varia da brand a brand). Sono riportate sotto, come esempio, le regole di business di un brand casuale.

3.5.1 Regola di business

In questo caso le regole di business sono molto complesse, quindi il focus in questo elaborato sarà piuttosto sulle specifiche funzionali estratte dalla raccolta dei requisiti e dall'analisi con l'utente finale. Le logiche della *segmentation* sono applicate per selezionare uno dei seguenti valori:

- DREAMER
- EXPLORER
- FAN
- STAR

- ELITE

Il valore della *segmentation* è calcolato considerando il comportamento di acquisto degli ultimi 12 mesi di merchandising. Due criteri principali guidano la selezione del valore di *segmentation*:

1. Spesa: calcola l'ammontare della spesa del cliente considerando tutte le transazioni fatte nel periodo di analisi. Quindi viene calcolata la somma della spesa di ogni scontrino associato al cliente.

2. Visite: conta il numero di visite fatte dal cliente nel periodo di analisi. A differenza del preferred store, se il cliente ha acquistato due volte nello stesso giorno entrambe le visite vengono conteggiate.

La tabella 3.2 mostra le relazioni tra i valori delle segmentazioni e i valori dei due criteri appena descritti.

SEGMENT	**CRITERIA**
ELITE	The calculation is based on the below criteria: 1. Revenue >10000€
STAR	The calculation is based on the below criteria: 1. Revenue >5000€ and <= 10000€ 2. Visit >= 2
FAN	the calculation is based on the below criteria: 1. Revenue >= 1000€ and <= 5000 2. Visit >= 2
EXPLORER	The calculation is based on the below criteria: 1. Revenue >= 100€ and <= 10000 2. Visit = 1
DREAMER	The calculation is based on the below criteria: 1. Revenue <1000€

Tabella 3.2: Criteri di mappatura per l'attributo Segment

Da questo tipo di analisi non è necessario tradurre i requisiti aziendali in codice SQL. Sarebbe necessario un ulteriore livello di astrazione per descrivere completamente le logiche implementate, ma non rientra nel focus dell'elaborato. Una volta terminata l'analisi a partire dalle logiche condivise, esse vengono tradotte in linguaggio Scala per costruire l'algoritmo di calcolo.

3.5.2 Informazioni di input

Per la segmentazione, e quindi per il calcolo dei *ricavi* e delle *visite*, sono necessari perlopiù i dati relativi al dataset degli scontrini. I campi utilizzati nello specifico sono:

- *client_id*: identificatore del cliente a cui è associato lo scontrino
- *amt*: totale speso dal cliente nello scontrino
- *business_date*: data in cui l'acquisto è stato fatto

3.5.3 CAP Output

Una volta recuperati i dati sorgenti viene applicato un filtro sul campo *business_date* per considerare gli acquisti fatti negli ultimi 12 mesi di merchandising. Quindi calcoliamo per ogni *client_id* le due metriche:

- *Ricavi*: somma di *amt*
- *Visite*: count(*) dei record del dataset scontrini

A questo punto, in base alla tabella di mapping 3.2 viene assegnato il valore di Segmentation. Esempi di output sono presentati nella fig. 3.5.

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	ELITE	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	FAN	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 3.5: *segmentation*: Output in CAPFull & CAPByDelta

Capitolo 4

Use Case

In questa sezione è presentato un esempio di un output *CAPFull* e di un output *CAPByDelta* e come i dati sono trasformati da tutti i job del flusso. Nello use case mostrato è descritto anche come sono configurate le tabelle parametriche necessarie per i job e come loro impattano con le trasformazioni. Verranno presi come oggetto tre clienti di un brand casuale, simulando il calcolo dei quattro attributi descritti nel capitolo precedente. La simulazione di questo use case consiste in quattro step:

1. Configurazione iniziale delle tabelle parametriche
2. Primo calcolo full degli attributi e aggiornamento della tabella *lastCalculationDate*;
3. Secondo calcolo full ed estrazione del delta giornaliero;
4. Trasformazione del delta e propagazione verso i sistemi target;

4.1 Configurazione iniziale delle tabelle parametriche

Prima di lanciare il primo job del flusso c'è bisogno di configurare due tabelle parametriche su *DocumentDB*: la *attributeScheduling + brand_name* e *lastCalculationDate + brand_name*. Immaginando di avere un brand di nome *XX* avremo quindi due tabelle parametriche: *attributeSchedulingXX* e *lastCalculationDateXX*. Le due tabelle hanno un document per ogni attributeGroup, per cui, siccome nel nostro caso sono stati presi quattro attributi che appartengono ad attributeGroup tutti diversi, avremo in totale quattro attributeGroup. Nella tab. 4.1 e nella

tab.4.2 è mostrato un esempio di configurazione per un il gruppo *PreferredGroup*, relativo all'attributo *preferred_store* (gli altri document hanno valori simili).

Field Name	**Field Value**
_id	(...)
AttributeGroup	PreferredGroup
LastDateSynchronization	InitialLoad
LastDateCalculation	InitialLoad
LastDateExecution	InitialLoad

Tabella 4.1: PreferredGroup: lastCalculationDateXX Parametric Table

Inizialmente, è stata configurata la tabella con tutti i valori dei campi *LastDate* settati a *InitialLoad*. Tale configurazione indica al calcolatore che il gruppo di attributi non è mai stato calcolato prima e che non c'è bisogno di accedere al calcolo precedente per il riconoscimento

Field Name	**Field Value**
_id	(...)
ActiveFlag	True
AttributeGroup	PreferredGroup
AttributeName	[preferred_store, preferred_sales_assistant]
DayOfWeek	null
Frequency	Daily

Tabella 4.2: PreferredGroup: attributeSchedulingXX Parametric table

Nella tabella sotto abbiamo due attributi nell'array AttributeName. Infatti il PreferredGroup è composto da due attributi, ma ci sono gruppi per i quali l'array potrebbe avere oltre 10 attributi.

4.2 Primo calcolo full degli attributi e aggiornamento della tabella *lastCalculationDate*

In questa sezione vedremo un esempio di un calcolo full e come la tabella *lastCalculationDateXX* viene aggiornata. Per questo esempio è stata lanciata una simulazione di initial load, prendendo il 5 Giugno 2023 come data di calcol e i seguenti quattro attributi come oggetto di calcolo:

- *contactable_email_address_flag*
- *preferred_store*
- *gender_title_common_consistent_flag*
- *segmentation*

Il risulato è mostrato nella fig.4.1. Come è possibile vedere il *Timestamp* e il *TriggerTimestamp* hanno lo stesso valore, in quanto si tratta di un initial load. Un altro elemento che ci suggerisce che si tratta di un primo calcolo è che tutti i record hanno il campo *CurrentValue* valorizzato a true. Il campo *_id* non è stato riportato, ma sarebbe costituito semplicemente da un identificatore univoco del record, calcolato come SHA1 della concatenazione dei diversi campi che compongono la chiave naturale della tabella.

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	Y	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	12345	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	Y	gendertitle_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	ELITE	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	N	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	67893	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	N	gendertitle_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	FAN	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 4.1: CAPFull Dataset Simulation

Una volta che il calcolo è terminato, il job aggiorna la collection *lastDateCalculationXX* come raffigurato sulla tabella 4.3. Come è possibile vedere i campi *LastDateExecution* e *LastDateCalculation* sono stati aggiornati con la data del calcolo. Dato il fatto che il gruppo di attributi è stato calcolato e non sincronizzato dal calcolo precedente, il valore sul campo *LastDateSynchronization* non è variato.

Field Name	Field Value
_id	(...)
AttributeGroup	PreferredGroup
LastDateSynchronization	InitialLoad
LastDateCalculation	20230605
LastDateExecution	20230605

Tabella 4.3: lastCalculationDateXX: Update Simulation

4.3 Secondo calcolo full ed estrazione del delta giornaliero

Un secondo calcolo full è stato lanciato durante la giornata successiva, il 6 Giugno 2023. In questo caso il calcolo è stato confrontato con il calcolo precedente del 5 Giugno 2023, in modo da dare evidenza dei soli valori degli attributi che hanno subito una variazione e di estrarli come calcolo delta. Come è possibile vedere nella figura 4.2 il cliente *123* ha subito una variazione su due valori, infatti nei record precedenti il campo *CurrentValue* è passato a False e il campo *TriggerTimestamp* è stato aggiornato con la nuova data. Il cliente *789* è rientrato per la prima volta nel perimetro dei clienti da calcolare. Questo può essere successo perché il cliente è stato aggiunto solo in questa data in anagrafica, o nello specifico perché è rientrato nei filtri dei clienti per il quale è necessario calcolare quel tipo di attributo. Per il nuovo cliente, trattandosi di attributi calcolati per la prima volta, vediamo quindi i campi *Timestamp* e *TriggerTimestamp* coincidenti con la data corrente di calcolo. Usando questa strategia, è possibile storicizzare i valori storici degli attributi per ogni cliente e dare evidenza di qual è il valore attualmente valido dell'attributo e cosa notificare al sistema target, sia in termini di nuovo valore valido e sia in termini di invalidazione dei vecchi valori.

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	Y	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	False	(...)
123	xx	N	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	12345	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	False	(...)
123	xx	67893	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	Y	gendertitle_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	ELITE	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	N	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	67893	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	N	gendertitle_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
456	xx	FAN	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	Y	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	34567	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	N	gendertitle_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	DREAMER	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 4.2: Second CAPFull Dataset Simulation

Dal secondo calcolo full quindi è possibile estrarre il delta prendendo tutti i valori con *TriggerTimestamp* uguali alla data corrente di calcolo, che in questo caso corrisponde a *2023-06-06T02:00:00.000Z*. Quindi, il delta è estratto come riportato nella 4.3, e come atteso il cliente *456* non è presente, in quanto non ha subito alcuna variazione per gli attributi calcolati. Sia il *CAPFull* che il *CAPByDelta* sono storicizzati su S3 e sono pronti per essere inviati al sistema target dopo le necessarie trasformazioni.

CustomerId	Brand	AttributeValue	AttributeName	Timestamp	TriggerTimestamp	CurrentValue	_id
123	xx	Y	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	False	(...)
123	xx	N	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
123	xx	12345	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	False	(...)
123	xx	67893	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	Y	contactable_email_address_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	34567	preferred_store	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	N	gendertitle_common_consistent_flag	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)
789	xx	DREAMER	segmentation	2023-06-15T02:00:00.000Z	2023-06-15T02:00:00.000Z	True	(...)

Figura 4.3: Delta Dataset Simulation

4.4 Trasformazione del delta e propagazione verso i sistemi target

In conclusione, possiamo vedere come il delta è trasformato e quindi propagato a sistemi target. In questo elaborato è stato deciso di presentare la propagazione del delta e non quella del full, in quanto è quello che giornalmente viene propagato ed esprime maggiormente le funzionalità del motore di calcolo.

4.4.1 Tabella parametrica *AttributeRouting*

Come già descritto nei precedenti capitoli, la *AttributeRouting* è una tabella parametrica su *DocumentDB*, utilizzata per indirizzare quali attributi devono essere inviati a quali sistemi target e se c'è bisogno di applicare un cast del data type o una renaming dell'attributo. L'utilizzo di questa tabella è fondamentale per permettere una dinamicità sull'indirizzamento dell'attributo senza cambiare il codice sorgente.
Nelle prossime pagine è mostrato un esempio di due document della tabella parametrica. Sono omessi i valori del campo *targetSystem* in quanto non hanno un interesse relativo agli aspetti tecnici della simulazione.

```
{

  "_id": "ObjectId(\"(...)\")",
  "brandDesc": "xx",
  "brandCode": "99",
  "attributeName": "preferred_store",
  "deprecationFlag": "N",
  "deprecationDate": "",
  "isFastGrowing": "N",
  "targetSystems": [(..)],
  "targetJobs": [

     {

       "targetName": "agg_to_usg",
       "attributeName": "preferred_store",
       "isCurrent": "Y"
    },
    {

       "targetName": "agg_to_usg_KA",
       "attributeName": "PreferredStore",
       "isCurrent": "Y"
    },
    {

       "targetName": "agg_to_usg_landingzone",
       "attributeName": "preferred_store",
       "isCurrent": "Y"
    },
    {

       "targetName": "agg_to_agg_customerAttributes",
       "attributeName": "",
       "isCurrent": "N"
    },
    {

       "targetName": "agg_to_agg_CAPDenormalized",
       "attributeName": "",
```

```
41          "isCurrent": "N"
42        }
43    ],
44    "type": "string"
45
46  }
```

```
{

  "_id": "ObjectId(\"(...)\")",
  "brandDesc": "xx",
  "brandCode": "99",
  "attributeName": "segmentation",
  "deprecationFlag": "N",
  "deprecationDate": "",
  "isFastGrowing": "N",
  "targetSystems": [(..)],
  "targetJobs": [

    {

      "targetName": "agg_to_usg",
      "attributeName": "",
      "isCurrent": "N"
    },
    {

      "targetName": "agg_to_usg_KA",
      "attributeName": "",
      "isCurrent": "N"
    },
    {

      "targetName": "agg_to_usg_landingzone",
      "attributeName": "",
      "isCurrent": "N"
    },
    {

      "targetName": "agg_to_agg_customerAttributes",
      "attributeName": "segmentation",
      "isCurrent": "Y"
    },
    {

      "targetName": "agg_to_agg_CAPDenormalized",
      "attributeName": "segmentation",
```

```
41          "isCurrent": "Y"
42        }
43    ],
44    "type": "string"
45
46  }
```

Nei document l'attributo *preferred_store* è richiesto in tre sistemi, mentre l'attributo *segmentation* è richiesto in altri due sistemi (ciò è osservabile controllando il valore del campo *isCurrent*). In alcuni casi viene fatta una renaming del nome dell'attributo in base alla naming convention del sistema destinatario. In nessuno dei casi riportati viene fatto un cast del type data, che rimane stringa come in fase di calcolo e scrittura su *CAPByDelta*. Per gli altri due attributi, per sintesi, assumiamo che siano presenti in tutti i sistemi target, con eventuali renaming.

4.4.2 Delta in *agg_to_usg*

Il job agg_to_usg è responsabile dell'inserimento e dell'aggiornamento degli attributi dei clienti su *DocumentDB*. Il modello è trasformato come il seguente json:

```
{

  "_id": "(...)",
  "customerId": "123",
  "brand": "xx",
  "updateDate": "2023-06-06T03:00:00.000Z",
  "insertDate": "2023-06-05T03:00:00.000Z",
  "operationFlag": "U",
  "attributes": [

    {

      "name": "contactable_email_address_flag",
      "value": "N",
      "lastUpdateDate": "2023-06-06T03:00:00.000Z"
    },
    {

      "name": "preferred_store",
      "value": "67893",
      "lastUpdateDate": "2023-06-06T03:00:00.000Z"
    },
    {

      "name": "gender_title_common_consistent_flag",
      "value": "Y",
      "lastUpdateDate": "2023-06-05T03:00:00.000Z"
    },
  ],
  "oldAttributes": [

    {

      "name": "contactable_email_address_flag",
      "value": "Y",
      "lastUpdateDate": "2023-06-06T03:00:00.000Z"
```

```
37      },
38      {
39
40        "name": "preferred_store",
41        "value": "12345",
42        "lastUpdateDate": "2023-06-06T03:00:00.000Z"
43      }
44    ]
45
46  }
```

Il document è stato aggiornato dal job in particolare si da evidenza che:

- Gli attributi *preferred_store* e *contactable_email_address_flag* hanno subito una variazione. I vecchi valori sono storicizzati nell'array *oldAttributes*, mentre i nuovi sono scritti nell'array *attributes*.
- Il campo *lastUpdateDate* da la possibilità di ricostrure il tutta la storia di ogni attributo.
- Lo schema è statico: anche se noi aggiungiamo più attributi, l'unico impatto è sulla dimensione dell'array. Questa funzionalità permette una facile costruzione dell'API che accede al document.
- L'attributo *segmentation* non è presente, come specificato sulla tabella *attributeRouting*.

4.4.3 Delta in *agg_to_usg_KA*

Per il job *agg_to_usg_KA* abbiamo bisogno di rimodellare la struttura come nella tabella 2.6. Prima di scrivere i dati sulla tabella di frontiera di Klient Analytics, abbiamo bisogno di registrare lo start del job sulla tabella semaforica *DLK_CRM_FLOW_CONTROL*. Quindi, inseriremo su di essa un record con questi valori 4.4:

Field Name	Field Value
TRANSFER_ID	(...)
STATUS	LOADING IN PROGRESS
FLOW_NAME	CUSTOMER ATTRIBUTES
LOAD_START_DATETIME	2023-06-06T03:00:00.000Z
LOAD_END_DATETIME	DateTime
PROCESSING_START_DATETIME	
PROCESSING_END_DATETIME	

Tabella 4.4: DLK_CRM_FLOW_CONTROL: Starting Signal

A questo punto viene processata la trasformazione del delta, in modo che esso venga inviato a Klient Analytics come nella fig. 4.4

Brand	TransferId	CustomerId	PreferredStore	ContactableEmailAddressFlag	GenderTitleCommonConsistentFlag
xx	(..)	123	67893	N	Y
xx	(..)	789	67893	Y	N

Figura 4.4: *T_DLK_CUSTOMERATTRIBUTES*: Delta Update

Come è possibile vedere solo i due clienti che hanno subito la variazione e che sono contenuti nel delta sono presenti. In particolare possiamo osservare le seguenti trasformazioni:

- Pivot degli attributi: ora c'è un solo record per cliente, con una colonna aggiuntiva per ogni attributo
- Renaming degli attributi: la naming convention è passata da uno snake case a un camel case
- Recupero degli attributi dall'ultimo calcolo: per il cliente 123 solo due attributi sono cambiati, per cui il delta conteneva solo quei due attributi. Per il modello dei dati richiesto da Klient Analytics c'è bisogno però di inserire un record contenente tutti gli attributi del cliente, per cui i restanti non inclusi nel delta vengono recuperati dal calcolo full.

Alla fine del job abbiamo bisogno di aggiornare il record precedentemente inserito sulla tabella semaforica, così come nella figura 4.5. Il campo *STATUS* è stato aggiornato in LOADING FINISHED ed è stato inserito il timestamp di fine processo nel campo *LOAD_END_DATETIME*.

Field Name	Field Value
TRANSFER_ID	(...)
STATUS	LOADING FINISHED
FLOW_NAME	CUSTOMER ATTRIBUTES
LOAD_START_DATETIME	2023-06-06T03:00:00.000Z
LOAD_END_DATETIME	2023-06-06T03:20:00.000Z
PROCESSING_START_DATETIME	
PROCESSING_END_DATETIME	

Tabella 4.5: DLK_CRM_FLOW_CONTROL: Ending Signal

4.4.4 Delta in *agg_to_usg_landingzone*

Nel job *agg_to_usg_landingzone* sono inviati gli attributi dei clienti all'*Enterprise Data Warehouse*. Prima di processare l'inserimento abbiamo bisogno di trasformare il delta in un nuovo modello dati come mostrato nella figura 4.5.

customer_id	brand_desc	brand_code	operation_flag	update_date	contactable_email_address_flag	preferred_store	gender_title_common_consistent_flag
123	xx	99	U	2023-06-15T02:00:00.000Z	N	67893	Y
789	xx	99	I	2023-06-15T02:00:00.000Z	Y	67893	N

Figura 4.5: *landingZone*: Delta Update

La tabella raffigurata è una rappresentazione del dataframe *Spark* prima della sua conversione in file CSV. Quindi il CSV è stato caricato sul layer di landingzone del datalake layer così che possa essere importata dal warehouse tramite *CrushFTP*.

4.4.5 Delta in *agg_to_agg_customerAttributes*

Il delta nel job *agg_to_agg_customerAttributes* viene trasformato in un modello simile a quello inviato a *Klient Analytics*. La differenza è che la storicizzazione viene fatta in uno dei layer aggregated del datalake. Il delta viene quindi rimodellato come nella figura 4.6.

client_id	brand	brand_code	date_calculation	segmentation	contactable_email_address_flag	gender_title_common_consistent_flag
123	xx	99	2023-06-15	ELITE	N	Y
789	xx	99	2023-06-15	DREAMER	Y	N

Figura 4.6: *customerAttributes*: Delta Update

Quello che possiamo osservare è che è stata applicata una pivot per avere un record per cliente, con una colonna per ogni attributo. Il campo *date_calculation* rappresenta l'ultima data in cui un cliente ha subito una variazione sull'ultimo attributo variato.

4.4.6 Delta in *agg_to_agg_CAPDenormalized*

Per il job *agg_to_agg_CAPDenormalized* abbiamo una struttura simile a quella prodotta nel job *agg_to_usg* job. La differenza, come prima, è che il dato dopo la trasformazione è salvato in un layer di aggregated del datalake. Qui abbiamo diversi attributi, come descritto sulla tabella parametrica *attributeRouting*. Per semplicità, qui è mostrato un json di rappresentazione per un solo cliente.

```
{

  "_id": "(...)",
  "customerId": "123",
  "brand": "xx",
  "updateDate": "2023-06-06T03:00:00.000Z",
  "insertDate": "2023-06-05T03:00:00.000Z",
  "operationFlag": "U",
  "attributes": [

    {

      "name": "contactable_email_address_flag",
      "value": "N",
      "lastUpdateDate": "2023-06-06T03:00:00.000Z"
    },
    {

      "name": "segmentation",
      "value": "ELITE",
      "lastUpdateDate": "2023-06-05T03:00:00.000Z"
    },
    {

      "name": "gender_title_common_consistent_flag",
      "value": "Y",
      "lastUpdateDate": "2023-06-05T03:00:00.000Z"
    },
  ]

}
```

Capitolo 5

Conclusione

In questo elaborato sono state descritte tutte le funzionalità del motore di calcolo CAP. Sfruttando gli strumenti AWS e Apache disponibili nell'ecosistema del cliente, è stato possibile sviluppare un motore di calcolo molto più performante rispetto quelli on-premise già presenti nell'organizzazione.

Dai grafici in figura 5.1 è possibile notare come i tempi di elaborazione dei vari KPI hanno subito una consistente diminuzione, in particolare sugli attributi il cui calcolo è caratterizzato da logiche complesse e da aggregazioni di un enorme quantità di dati. L'attributo "PreferredStore" ad esempio, per il brand più grande della holding, viene calcolato su un set di trenta milioni di clienti, trasformando e aggregando circa un miliardo di righe di vendite.

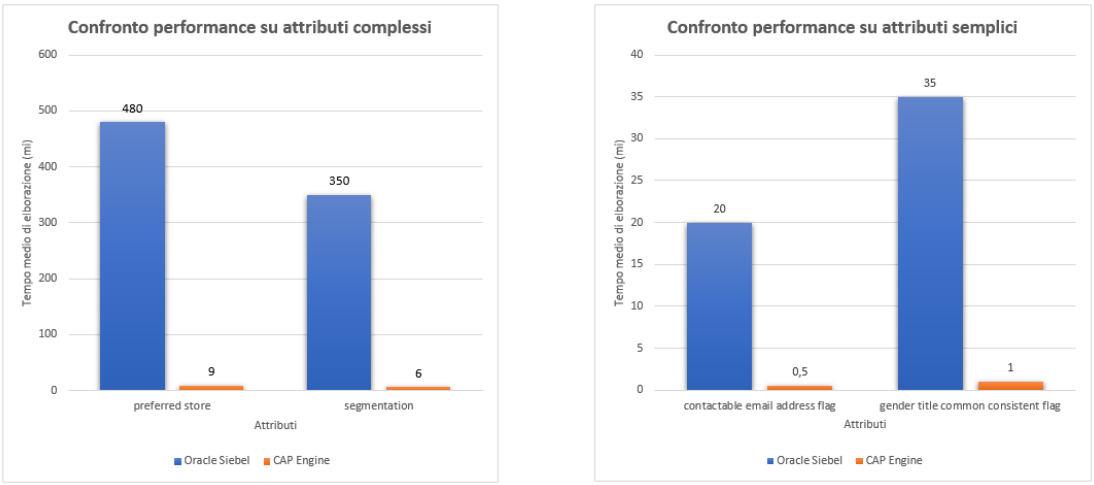

Figura 5.1: Confronto tempi di elaborazione medi tra Oracle Siebel e CAP Engine

Oltre un beneficio sulle performance, il vantaggio più grande è che CAP può essere facilmente scalato grazie alla natura cloud-based dell'infrastruttura. Infatti, se il volume di dati o il numero di KPI da calcolare per ogni brand aumenta, possiamo aggiungere risorse a ciascuna applicazione Spark per garantire il corretto funzionamento del servizio, assicurando sempre i tempi di calcolo concordati con gli utenti finali.

Grazie all'ambiente datalake già presente nell'organizzazione, CAP è in grado di memorizzare gli attributi calcolati quotidianamente su *AWS S3* e su *DocumentDB*. Tutti i job sono in grado di diffondere le informazioni anche a tutti quei sistemi target che non hanno accesso diretto al datalake, soddisfando a pieno il requisito di centralità del dato, diventando quindi il sistema master degli attributi del cliente.

Il sistema di sviluppo è pronto a introdurre in tempi ridotti nuovi KPI richiesti dai brand, indirizzando ogni attributo al sistema target dinamicamente tramite il sistema di routing presentato. In futuro, sarà possibile replicare o utilizzare lo stesso motore per calcolare i KPI relativi ad aree diverse dell'organizzazione. Ciò significa che il servizio è un prodotto autonomo, non direttamente collegato all'area CRM, utilizzabile anche quindi in altre funzioni aziendali. Per estendere il servizio a nuovi sistemi è già in fase di sviluppo un modulo in grado di inviare gli attributi a nuovi sistemi di destinazione tramite *Apache Kafka*, un servizio di streaming di messaggi che implementa il modello produttore-consumatore.

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Apache Software Foundation. Apache airflow documentation, 2023. https://airflow-fork-tedmiston.readthedocs.io/en/latest/ [Ultimo accesso: (2023)].

[2] Apache Software Foundation. Core concepts: Dags, 2023. https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html. [Ultimo accesso: (2023)].

[3] Apache Software Foundation. Spark configuration, 2023. https://spark.apache.org/docs/latest/configuration.html [Ultimo accesso: (2024)].

[4] Inc Gartner. Data warehouse, 2023. https://www.gartner.com/en/information-technology/glossary/data-warehouse [Ultimo accesso: (2023)].

[5] IBM. Cos'è apache spark?, 2023. https://www.ibm.com/it-it/topics/apache-spark [Ultimo accesso: (2023)].

[6] Google LLC. Che cos'è un data lake?, 2023. https://cloud.google.com/learn/what-is-a-data-lake?hl=it#:~:text=A%20data%20lake%20is%20a,data%20lake%20on%20Google%20Cloud. [Ultimo accesso: (2023)].

[7] Naveen (NNK). Difference between spark worker vs executor, 2023. https://sparkbyexamples.com/spark/difference-between-spark-worker-vs-executor/ [Ultimo accesso: (2023)].

[8] MindMajix Pooja Mishra. What is microstrategy, 2023. https://mindmajix.com/what-is-microstrategy [Ultimo accesso: (2023)].

[9] Aditi Prakash. Operational data stores (ods): An overview and use cases, 2023. https://airbyte.com/data-engineering-resources/operational-data-stores#:~:text=An%20ODS%20is%20a%20centralized,ODS%20in%20their%20raw%20formats. [Ultimo accesso: (2023)].

[10] Anzhelika Serhiienko. What is salesforce? expert insight on what salesforce is used for, 2023. https://ascendix.com/blog/what-is-salesforce-what-salesforce-is-used-for/ [Ultimo accesso: (2024)].

[11] Talend. Data lake e data warehouse, 2023. https://www.talend.com/it/resources/data-lake-vs-data-warehouse/ [Ultimo accesso: (2023)].

[12] Ansam Yousry. Apache airflow vs apache nifi: A comprehensive comparison, 2023. https://medium.com/illumination/apache-airflow-vs-apache-nifi-a-comprehensive-comparison-b7f55d5998f4 [Ultimo accesso: (2023)].

Ai miei cari genitori,

Vi dedico questo traguardo con immenso affetto e gratitudine.

Mi mancate ogni giorno e vorrei tanto condividere con Voi questo momento di gioia.

Vi voglio bene.