Politecnico di Torino

POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

A.a. 2022/2023

Progettazione e implementazione di una Data Platform basata su Data Virtualization e Data mesh

Relatore

Prof. Paolo GARZA

Candidato

Claudio COMO

Sessione di Laurea Luglio 2023

Sommario

In questo progetto è stato realizzato il Proof of Concept (POC) di una piattaforma che mirasse a migliorare lo stato delle data platoform tradizionali che sempre più spesso si trovano ad avere a che fare con il problema della centralizzazione dei dati, dovuto, in parte, anche dal debito tecnico accumulato nel tempo. Quello del debito tecnico è un problema sistemico e generalmente viene tenuto in considerazione durante la realizzazione di qualsiasi progetto, in questo ambito, data l'evoluzione costante dei dati, potrebbe essere accumulato però molto velocemente. Prendendo ispirazione da alcune tecnologie, o paradigmi, creati per migliorare lo stato attuale delle data platform, è stata progettata una piattaforma open-source che potesse realizzare il paradigma ETL, e non solo, su dati provenienti da sorgenti differenti in modo facile e veloce, fornendo anche uno strumento di visualizzazione dall'uso intuitivo anche per utenti non tecnici. La piattaforma è basata sull'idea che possa evolversi nel tempo, permettendo l'evoluzione e l'adattamento alle novità in maniera consistente.

Ringraziamenti

Vorrei ringraziare innanzitutto il mio relatore, il professor Paolo Garza per avermi seguito con dedizione durante la realizzazione di questa tesi, sempre aperto al confronto e disponibile all'ascolto di tutte le mie richieste. Il mio tutor aziendale Emanuele Gallo per avermi fornito feedback e pareri durante il progetto, ponendo fiducia nella sua realizzazione.

Vorrei ringraziare i miei amici, da quelli conosciuti da bambino che sono rimasti sempre con me e che mi hanno sostenuto con il loro instancabile supporto, a quelli conosciuti all'università, durante la triennale o la magriastrale, che sono stati fondamentali per superare i momenti difficili, le sessioni interminabili, riuscendo ad alleggerire uno dei percorsi più difficili che mi ha riservato finora la vita. Quindi grazie Sara, Lia, Vito, Piero, Gianno, Danzio, Carletto, Cico, Nzino, Irene, Ale, Agatina, Pozzy, Gianluca, Alexandro e tutte le altre persone che mi sono ritrovato vicino fino ad ora, perchè siete entrati nella mia vita in momenti differenti, ma spero ci rimarrete.

I ringraziamenti, quelli più importanti, vanno alla mia famiglia. I miei genitori, che hanno fatto tanti sacrifici per permettermi di avere il futuro che desideravo, uno dei sacrifici più grandi proababilmente è stato quello di avermi lasciato andare a vivere in una città molto distante dalla loro, sacrificio che tuttora un po' patiscono ma che reggono grazie all'amore che provano per me. E mia sorella, la mia alleata, il mio opposto, la mia fonte di ispirazione più grande, la colonna portante della mia vita. Per loro non esisterebbero sufficienti ringraziamenti.

Infine, ringrazio due persone speciali, le mie nonne, Rosetta e Caterina, che si prendono cura di me da sempre, in modi differenti ma fondamentali in egual maniera.

L'ultimo pensiero va a te, nonno Giuseppe, sei stata la persona a cui mi sono affidato il giorno che ho dato il mio primo esame, sperando che da qualche parte lassù potessi veramente aiutarmi. Fu il primo esame che superai.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione generale

1.1 Problema

Guardando alla storia dell'enterprise data management possiamo notare che sin dall'inizio le enterprise data platform erano state pensate per centralizzare ed integrare dati provenienti da fonti diverse e da diversi sistemi transazionali. Nel tempo, però, i dati analitici e quelli transazionali sono stati molto spesso divisi. Ad ora, però, ritorna la necessità di ricongiungere questi dati per apprezzarne e utilizzarne il pieno valore. In questa tesi verranno analizzate alcune tecnologie che sono state adottate negli anni e che sono adottate tutt'ora nell'enterprise data management con il focus su delle nuove tecniche e tecnologie che stanno prendendo piede di recente e che puntano a raggiungere l'intento originale delle data platform. Ovvero, ottenere maggiore controllo sui dati al fine di seguire un approccio data-oriented.

Infatti, uno dei problemi presenti generalmente all'interno delle aziende è quello del Technical Debt[1, 2](debito tecnico) dovuto ad una pratica assai comune, quella di volere tutto e subito, a discapito di quello che sarà nel futuro. Spesso vengono fatte delle scelte di progettazione e implementazione poco valide portando le aziende a pagarne un caro prezzo nel futuro. La cosa più grave è che, con questo mondo in costante evoluzione, il futuro non è più "domani" ma, costantemente, "oggi". Quindi un'azienda si ritrova a progettare una data platform che magari in pochissimi anni diventerà obsoleta. Per cercare di non farla diventare obsoleta, rischiando quindi di perdere gli investimenti fatti precedentemente, si cerca di mettere delle "pezze" aumentando sempre di più il debito tecnico, finchè non diventa ingestibile e poco comprensibile ai più. Naturalmente, per evitare tutto questo, i vari Cloud Provider provvedono sempre più spesso ad offrire delle soluzioni per ovviare a questi problemi, consapevoli che troveranno dei clienti. Questo problema, insieme all'evoluzione tecnologica, è quindi ciò che porta maggiore difficoltà oggi a sfruttare i dati in

modo corretto. Non potendo agire sull'avanzamento tecnologico l'altra soluzione che rimane è quella di agire quindi sull'infrastruttura. Inoltre, le piattaforme attuali richiedono di spostare fisicamente i dati per essere esplorati e valutarne la loro utilità, il che richiede costi, oltre che lo spreco di spazio inutilmente.

1.2 Metodologia

Per questo progetto sono state studiate e analizzate le varie data platform tradizionali e la loro evoluzione. Successivamente sono state esaminate alcune tecnologie, o paradigmi, creati con l'obbiettivo di migliorare lo stato attuale delle data platform, come: Data Mesh, Data Fabric e Data Virtualization. Prendendo quindi ispirazione da alcune piattaforme presenti sul mercato come Starburst, Dremio o Denodo, si è cercato di creare una piattaforma open source che potesse mirare a risolvere il problema.

1.3 Soluzione

La soluzione proposta prevede una piattaforma nata con l'intento di evolversi e che si possa adattare al futuro. Dove non sia necessario mettere una "pezza", ma aggiungere solo un altro pezzo all'ingranaggio. Durante lo svolgimento della tesi in Data Reply SRL ho realizzato quindi un Proof of Concept (POC) di questa piattaforma che prende ispirazione dalle tecniche e i paradigmi sopra citati e apre una strada verso di essi. Inoltre, queste tecniche possono collaborare insieme e nessuna esclude l'altra. Anzi, molto spesso l'utilizzo ibrido di queste tecniche ne migliora i risultati.

Inoltre, durante questo periodo, ho avuto modo di appurare che sempre più spesso i clienti richiedono di avere una piattaforma che gli permetta di avere un facile accesso unificato ai dati per migliorarne la gestione e l'utilizzo consapevole. Sono presenti sul mercato alcune piattaforme che permettono di raggiungere questo risultato, ma l'intenzione era quella di crearne una versione più personalizzabile e adattabile alle necessità dell'azienda. Questa piattaforma è basata su tre tecnologie che ne definiscono il cuore pulsante: Trino, DBT e Apache Airflow. A queste tecnologie ne sono state aggiunte altre a supporto per dimostrarne le potenzialità. Grazie all'utilizzo di Trino è infatti possibile leggere i dati direttamente dalla sorgente, il che permette di realizzare un'esplorazione dei dati senza la necessità di effettuare una reale ingestione di essi. DBT permette invece di effettuare delle trasformazioni sui dati, sfruttando Trino che gli permetterà di leggere i dati dalla sorgente, e di effettuare dei test sulla qualità dei dati. Questo permetterà di fare l'ingestione solo dei dati veramente utili per l'azienda, il tutto verrà orchestrato da Airflow che permetterà di eseguire i flussi creati tramite i modelli di DBT. Questa

piattaforma, grazie all'utilizzo di Trino e DBT, permetterà anche di applicare i paradigmi del Data Mesh creando tramite DBT dei Data Product basandosi sulle varie sorgenti di dati.

Sicuramente la parola chiave migliore per definire questa piattaforma è ADATTABILITÀ.

Capitolo 2

Cos'è una Data Platform e quali sono le Data Platform tradizionali

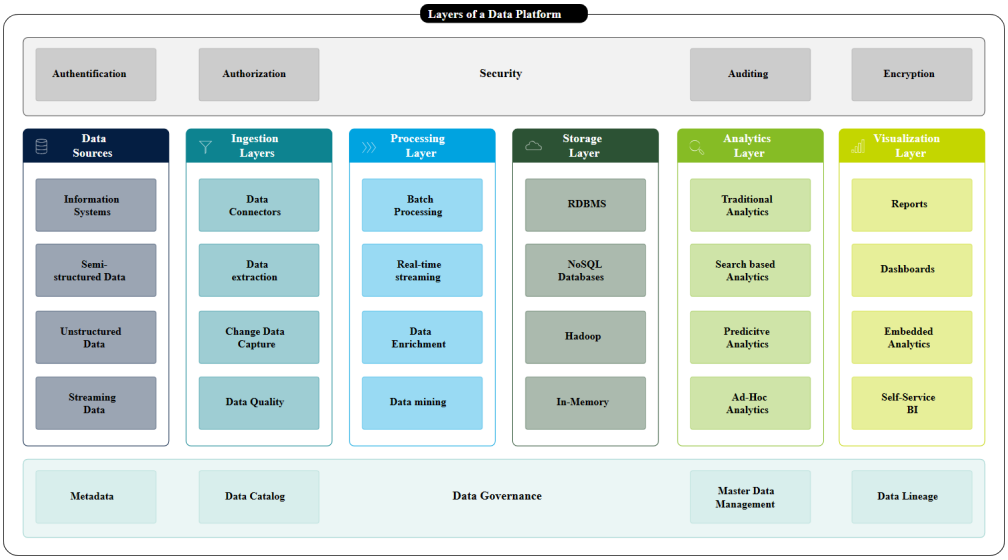

Figura 2.1: Esempio di una data platform generica [3]

Una data platform è una soluzione che nasce dalla necessità di sfruttare i dati in modo utile ed efficacie in ambito business. I tre obbiettivi chiave sono ingestione, processamento e analisi dei dati.

Una data platform si occupa di tutti i dati di un'azienda facilitandone l'utilizzo attraverso tutte le componenti dell'azienda stessa, dall' IT alla BI, centralizzando i

dati tramite una singola piattaforma.

Al giorno d'oggi i dati, infatti, vengono gestiti utilizzando servizi differenti che mirano a risolvere problemi individuali, questi dati però variano gli uni dagli altri rendendo impossibile l'integrazione degli stessi e portando a quello che è definito come data silos. Ovviamente, per fare ciò, una data platform dovrebbe prendere i dati quasi dalla sorgente senza aggiungere ulteriore complessità. Le data platform, inoltre, sono caratterizzate da 3 pilastri:

- Volume: La quantità di dati generati e immagazzinati
- Variety: Il tipo e la natura dei dati
- Velocity: La velocità con cui sono generati e processati i dati

In questo capitolo verrà spiegata inizialmente com'è fatta una data platform e successivamente verranno analizzate le data platform che sono state realizzate nel tempo[4].

2.1 Com'è fatta una Data Platform?

Come è possibile vedere dalla figura 2.1, le data platform attuali hanno 5 livelli principali, ognuno di questi livelli compie una parte fondamentale nell'estrazione e l'utilizzo dei dati. In questo paragrafo verranno spiegati brevemente i vari livelli di cui è composta una data platform.

2.1.1 Ingestion Layer

Il primo livello è quello che si occupa della fase di ingestione dei dati. I dati generalmente sono provenienti da più sorgenti e in più formati, per questa ragione vengono offerti strumenti dalle piattaforme per farne l'estrazione, in modo che i dati vengano successivamente processati dal successivo livello. Durante la fase di ingestione, i dati possono essere estratti, per esempio, ad intervalli regolari o tramite streaming real-time. A volte possono essere anche leggermente trasformati, puliti o filtrati prima di passare al livello successivo. L'ingestion layer dovrebbe essere anche adatto ad una facile scalabilità in base alla mole di dati che riceve, può includere dei meccanismi di validazione dei dati e potrebbe anche occuparsi della gestione dell'aggiornamento dei metadati relativi ai dati che sta estraendo. I metadati verranno spiegati successivamente parlando anche del perché sono così importanti[3].

2.1.2 Processing Layer

Questo livello si occupa di processare i dati che sono stati estratti durante la fase di ingestione. Al suo interno i dati vengono ripuliti del tutto e trasformati in base alla necessità. Possono essere effettuate anche operazioni di aggregazione o deduplicazione dei dati con l'obbiettivo di migliorarne la loro consistenza e qualità. Inoltre, questo livello viene utilizzato per recuperare informazioni utili dai dati come metriche, indicatori o report. Utilizzando anche delle join tra i dati estratti o generando aggregazioni con diversa granularità, come per esempio in base a data, ora o giorni di un mese. I dati vengono generalmente trasformati utilizzando dei frameworks come Apache Spark, che permette di processare i dati in modo parallelo e distribuito. L'obbiettivo finale del livello di processamento è quello di fornire quindi dei dati di qualità che possano essere analizzati dai livelli successivi, ovvero il Visualization Layer o l'Analytic Layer. Tutto questo, facendo rispettare le politiche di data governance e le misure di sicurezza che devono essere attuate. Che possono includere criptazione dei dati, tecniche di anonimizzazione o meccanismi di monitoraggio e protezione dei dati sensibili[3].

2.1.3 Storage Layer

Il livello di archiviazione, che all'apparenza pare il più scontato al giorno d'oggi, è un punto chiave all'interno di una data platform. Questo livello non si occupa semplicemente di archiviare i dati ma è importante prestare attenzione a come questi dati vengano archiviati. La scelta dei giusti strumenti di archiviazione, infatti, cambia in modo abbastanza radicale la qualità di una data platform. I principali compiti che svolge il livello di archiviazione sono, oltre l'utilizzo di tecnologie specifiche per archiviare i dati[3]:

- La definizione di modelli di dati o schemi per organizzare i dati in relazione al tipo di strumento utilizzato per archiviarli.
- La scelta di come partizionare o dividere i dati all'interno dell'archivio soprattutto quando vengono gestiti grandi volumi di dati.
- Implementare misure che mirino a rafforzare la data governance e la sicurezza dei dati.
- Fare da tramite tra i livelli di ingestione e processamento dei dati, e le applicazioni interessate ai dati che vengono generati.

2.1.4 Analytics Layer

Altro livello molto importante all'interno di una data platform, utilizzato per estrarre informazioni rilevanti dai dati che vengono immagazzinati nel livello di

archiviazione. Al suo interno vengono adoperate diverse tecniche e strumenti per analizzare i dati e fornire spunti per migliorare il decision-making per l'azienda. Possono essere eseguiti diversi tipi di analisi, da quella puramente esplorativa dei dati che punta a comprendere le relazioni tra i dati, scoprire i trend e i pattern, a quella descrittiva che si occupa di mettere insieme uno storico dei dati per comprendere cosa è accaduto in un determinato evento o periodo. Oltre a queste, naturalmente, possono essere fatte analisi predittive applicando algoritmi matematici o modelli di machine learning, o analisi prescrittive per dare indicazioni e raccomandazioni su come agire al fine di migliorare le entrate nel futuro. L'Analytics Layer viene spesso integrato con degli strumenti di visualizzazione dei dati, ragione per cui può essere unito anche al Visualization Layer[3].

2.1.5 Visualization Layer

Il livello di visualizzazione ha il compito di mostrare spunti e informazioni rilevanti in modo semplice e intuitivo tramite l'utilizzo di dashboard, grafici, diagrammi o mappe. È possibile, come detto precedentemente, che il Visualization Layer possa essere integrato anche con l'Analytics Layer per fornire un'ambiente unico in cui analizzare e visualizzare i dati. Naturalmente per integrare i due ambienti è necessario valutare bene lo strumento da utilizzare in modo che possano essere effettuate tutte le operazioni necessarie ad entrambi gli ambienti. Generalmente il livello di visualizzazione utilizza delle meccaniche semplici essendo mirato ad utenti non troppo tecnici che possano sfruttarlo per creare dei report o delle presentazioni da condividere facilmente con altri utenti. Può fornire anche strumenti per personalizzare lo stile delle visualizzazioni create, come: schemi colorati, fonts, etichette o possibilità di configurare il layout[3].

2.2 Data warehouse

La prima generazione di data platform che nacque fu la data warehouse con lo scopo di integrare i dati transazionali per supportare le attività di business intelligence, in particolare gli analytics.

Un data warehouse immagazzina e consolida grandi quantità di dati da fonti diverse e le sue capacità analitiche sono utili alle aziende per ricavare informazioni importanti per migliorare il processo decisionale. Questo tipo di data platform fornisce dei record storicizzati che possono essere molto preziosi per data scientist e business analytics che possono consultarli utilizzando facilmente strumenti basati su SQL o strumenti di BI. I data warehouse sono subject-oriented, tendono quindi ad analizzare dati su uno specifico argomento o area funzionale, riuscendo ad unire dati provenienti da origini molto differenti come ERP, CRM, siti web, applicazioni mobile. I dati tra l'altro non sono volatili, dal momento che sono in un data

warehouse e se ne può analizzare la variazione nel tempo. Se un data warehouse è ben progettato permetterà di fare delle query molto velocemente, consentendo la gestione di un flusso di dati elevato e offrendo agli utenti la possibilità di fare analisi più o meno approfondite.

Per queste ragioni un data warehouse è pensato per immagazzinare le meta information. Ovvero, informazioni statistiche e parametri riguardanti i propri prodotti. Questi parametri possono essere usati, per esempio, per fare delle previsioni. Ovviamente i dati presenti nel data warehouse non sono dati "freschi" perché è necessario molto tempo per catalogarli quando arrivano in ingresso al data warehouse. Quindi, possibilmente, sarebbe più comodo utilizzarlo come un repository in cui salvare copie dei database[4, 5, 6].

2.2.1 Architettura

Un data warehouse generalmente include un database relazionale, usa una soluzione di ELT (estrazione, caricamento e trasformazione), funzionalità di analisi statistiche, reporting e data mining e altre applicazioni analitiche che generano informazioni a cui verranno applicati algoritmi di data science e artificial intelligence. La sua architettura viene determinata dalle esigenze specifiche dell'organizzazione. Le più comuni sono:

- **Semplice**: struttura di base dove metadati, dati di riepilogo e dati non elaborati sono archiviati nel repository centrale. Il repository riceve i dati dalle fonti da una parte e le mette a disposizione degli utenti finali dall'altra[7].

- **Semplice con area di gestione temporanea**: come l'architettura semplice ma con un ulteriore livello che pulisce i dati e li elabora prima di inserirle nel warehouse[7].

- **Hub e spoke**: vengono aggiunti dei data mart tra repository centrale e utenti finali fornendo un'organizzazione personalizzata dei dati agli utenti in modo da poter gestire diverse linee di business[7].

- **Sandbox**: aree private protette e sicure che consentono alle aziende di testare nuovi metodi di analisi o esplorare un set di dati in modo informale senza sottostare ai protocolli del data warehouse[7].

2.2.2 Vantaggi

- Dati più organizzati e integrati che permettono ai ricercatori di navigarci attraverso molto più facilmente[7].

- Fornisce accesso ai dati di tutta l'azienda in un solo punto migliorando la qualità del decision making in ambito business[7].
- Riduce il tempo speso dagli impiegati nel recuperare informazioni, tempo che può essere impiegato nell'analisi portando di conseguenza ad un miglioramento della produttività[7].

2.2.3 Svantaggi

- I data warehouse funzionano molto bene con i dati strutturati ma non con i dati non strutturati come quelli provenienti da social media e streaming o con i log[7].
- Possono esistere dei problemi di compatibilità, infatti i data warehouse combinano dati da più database differenti che possono contenere misure differenti o titoli differenti per indicare lo stesso tipo di dato, oppure i dati integrati potrebbero non contenere tutti i campi richiesti[7].
- I costi possono essere molto alti, infatti preparare e gestire questa grande mole di dati richiede molto tempo, oltre che denaro. Inoltre più dati storici verranno contenuti nel data warehouse, maggiore naturalmente sarà il suo costo per mantenerlo[7].
- Per il tipo di dati che vengono contenuti nei data wharehouse, questi, sono soggetti molto spesso ad attacchi hacker, il che necessita una grande attenzione, nonché ulteriore spesa, per tenere alto il livello di sicurezza[7].

2.3 Data Lake

Il Data Lake fa parte della seconda generazione di data platform e altro non è che un repository centralizzato, creato utilizzando un cluster di hardware poco costosi e scalabili, che permette di archiviare qualsiasi tipo di dato (strutturato, semi-strutturato, non strutturato o binario) nel suo stato nativo. Il principio su cui si basa l'immagazzinamento dei dati viene chiamato schema on read, infatti quando i dati vengono ricevuti dal data lake non vengono adattati ad uno schema ben preciso prima di essere immagazzinati. Data la sua natura, il data lake è molto utile per investigazioni e verifiche di nuove ipotesi[8, 9].

2.3.1 Implementazione 'on Cloud' o 'on Premise'

I data lakes generalmente venivano implementati 'on-premise' con storage come Hadoop Distributed File System (HDFS) e processati con clusters Hadoop. Hadoop

è scalabile, low-cost, e offre buone performance poiché la computazione e la posizione dei dati sono nello stesso punto. Ovviamente non è semplice creare un'infrastruttura 'on-premise' poiché è necessario tenere in considerazione multipli fattori come lo spazio necessario per i vari server ed il setup degli stessi. Inoltre un sistema del genere è difficile da scalare perché richiede tempo, spese e ulteriore spazio. Per questa ragione è molto importante fare una stima dei requisiti più accurata possibile. Dall'esperienza di implementazione 'on-premise' nasce la soluzione 'on-cloud' che cerca di risolvere o comunque contenere i problemi che si riscontrano nella versione 'on-premise'. Infatti sono molto più semplici da creare e avviare (con un avvio generalmente incrementale). Tra l'altro si paga in base a quanto viene utilizzato ed è molto facile da scalare, principale comodità dell'ambiente cloud[8, 9].

2.3.2 Architettura

Nel paragrafo precedente si evince come sia possibile implementare un data lake in più modi, infatti è importante ricordare che un data lake è composto da due componenti principali: lo storage e la computazione. Entrambe queste componenti possono essere implementate sia 'on premise' che 'on cloud' e questo permette di realizzare diversi tipi di architetture differenti. Generalmente l'architettura di una data platform basata su un data lake prevede:

- Un livello di ingestione dei dati che permette di estrarre i dati dalle varie sorgenti.
- Un livello di archiviazione in cui verranno utilizzati dei sistemi di memorizzazione dei dati HDFS o sistemi cloud come Amazon S3 o Azure Data Lake Storage. Naturalmente dei sistemi in cloud offrono maggiore scalabilità rispetto a dei sistemi on Premise, tutto dipende dalle proprie necessità naturalmente.
- Un livello di Metadati che si occuperà di gestire le informazioni legate ai dati che sono all'interno del datalake per facilitarne il loro utilizzo e la loro reperibilità.
- Un meccanismo di Data Governance che si occupa di garantire la qualità, la sicurezza e la conformità dei dati all'interno del Data Lake.
- Un livello di Data Processing e uno di Data Analytics come quelli citati precedentemente.
- Un livello di Data Access che permette l'accesso ai dati contenuti all'interno della piattaforma

La peculiarità, come detto precedentemente, è che ai dati viene dato uno schema solo al momento della necessità. In questo caso infatti la fase di ingestion si occupa

di inglobare tutti i dati all'interno del Data Lake per decide successivamente come utilizzarli[8, 9].

2.3.3 Rischi

Una delle sfide da affrontare quando si usano i Data Lake on cloud è però legata al tipo di dati che si vogliono tenere on cloud, infatti alcune aziende per esempio preferiscono tenere quelli che sono i dati sensibili o confidenziali on Premise per una questione di sicurezza. Nonostante i fornitori di data lake cloud-based stanno investendo molto nella sicurezza rimane il timore di un furto dei dati. Un altro dei rischi maggiori nell'utilizzo dei data lake è quello di trasformare il data lake in un data swamp (palude). Si definiscono data swamp quei data lake che accumulano dati su dati senza un sistema di Data Governance che permetta di sapere che dati sono contenuti nel data lake e dove trovarli[8, 9].

2.3.4 Vantaggi

I vantaggi dei Data Lake sono[8, 9]:

- Le ragioni per scegliere un Data Lake sono svariate, una fra tante però è quella legata al numero di dati prodotti al giorno d'oggi che è in constante crescita e provenienti dalle più disparate fonti. Molto spesso è fondamentale immagazzinare questi dati senza perder tempo, concentrandosi di più su algoritmi e usi che è possibile fare di questi dati in futuro.
- Il costo dei data lake, data la loro natura, è inferiore rispetto a quello di un data warehouse

2.3.5 Svantaggi

Gli svantaggi dei Data Lake sono[8, 9]:

- Utilizzare i tools di BI potrebbe essere difficile nei data lakes se i dati in quest'ultimi non vengono gestiti correttamente. In più le query potrebbero non essere del tutto accurate visto che la struttura dei dati non è consistente.
- Risulta molto difficile implementare delle misure di sicurezza per i dati sensibili data la mancanza di consistenza dei dati.

2.4 Data LakeHouse

Terza generazione di data platforms, il data lakehouse è un'architettura che combina un data lake e un data warehouse, il tentativo è quello di prendere il meglio che

c'è di un'architettura e dell'altra. L'obbiettivo del data lakehouse è lo stesso delle altre data platform, ingerire grandi flussi di dati al fine di utilizzarli per decisioni importanti nel contesto business. Quest'ultima però offre la capacità di gestione dei dati dei data warehouse combinata alla scalabilità e flessibilità dei data lake. Inoltre aiuta a ridurre la duplicazione dei dati visto che offre una singola piattaforma su cui lavorare a tutti gli utilizzatori, il rapporto costo-efficienza è molto buono[10, 11, 12].

2.4.1 Obbiettivi

Ci sono 4 obbiettivi principali a cui mira un data lakehouse[10, 11, 12]:

- Risolvere i problemi legati ai data silos provvedendo un repository centralizzato per immagazzinare e gestire grandi moli di dati strutturati e non.
- Ridurre lo spostamento dei dati, effettuando infatti un'analisi preliminare dei dati all'interno del data lake è possibile decidere a piori quali dati utilizzare perché significativi.
- Permettere alle organizzazioni di effettuare data processing in modo veloce ed efficiente, rendendo possibile un'analisi veloce e decision-oriented.
- Rendere possibile alle organizzazioni di gestire facilmente l'accesso ai loro dati nonostante la costante crescita degli stessi.

2.4.2 Funzionamento

Un data lakehouse è basato su un'architettura multi-layer:

1. Il primo strato è l'Ingestion Layer che riceve una grande mole di dati non trattati, che possono essere: strutturati, semi-strutturati o non strutturati, che vengono aggiunti al data lake tramite metodi di batch e streaming.
2. Lo strato successivo è lo Storage Layer che immagazzina, grazie ad un repository centralizzato, i dati in object store a basso costo come AWS S3 o Azure Blob Storage v2.
3. A seguire, il Metadata Layer, in questo strato viene creato un catalogo con informazioni riguardanti i dati contenuti nel data lake. Grazie a questo strato è possibile ricreare una delle caratteristiche tipiche del data warehouse, le ACID transactions, ma anche fare cache, indicizzazione ed estrazione dei dati.
4. Il quarto strato è l'API Layer, in questo strato risiedono le API che permettono ai vari utenti di accedere ai dati più facilmente o processare dei tasks più velocemente.

5. L'ultimo strato è il Data consumption Layer, in questo strato sono presenti vari tool o applicazioni utilizzati in ambito analitycs. Come anche metodi per lanciare dei tasks di machine learning o queries SQL.

Proprio grazie all'uso del Metadata Layer è possibile creare una centralizzazione dei dati con un singolo punto di accesso, sia per i BI analytics sia per gli utenti che vogliono utilizzare i dati per modelli di machine learning[10, 11, 12].

2.4.3 Vantaggi

I vantaggi di un Data lake house sono[10, 11, 12]:

- Il costo di questa architettura non è molto alto poiché è basato su un singolo livello di storage, il che alleggerisce e focalizza il lavoro dei team per gestirlo e mantenerlo (vantaggio che viene dal data lake).
- Grazie sempre al fatto che i dati siano disponibili a tutti tramite un singolo data storage viene ridotta di molto la duplicazione dei dati.
- È possibile accedere ai dati presenti nel data lakehouse con qualsiasi tool, favorendo anche le app che possono essere utilizzate solo su dati strutturati.
- Supporta l'utilizzo di analytics tool avanzati come PowerBI o Tableau. Inoltre visto che utilizza file in Open Data format (e.g. Parquet, OCR) supporta le APIs e le librerie di machine learning.
- Grazie al Metadata Layer introdotto nei data lakehouse è possibile migliorare la governance dei dati e la sicurezza.

2.4.4 Svantaggi

Gli svantaggi di un Data Lake House sono[10, 11, 12]:

- Potrebbe essere difficile progettare e mantenere un design monolitico come quello del data lakehouse, inoltre un design universale potrebbe portare a minori funzionalità rispetto a soluzioni create ad hoc.
- Essendo una tecnologia relativamente nuova ci vorrà tempo e risorse prima che se ne potrà sfruttare il pieno potenziale.

	Data Lake	**Data Warehouse**	**Data Lakehouse**
Tipo di dato	Semi-strutturato e non strutturato	Strutturato	Semi-strutturato, non strutturato e strutturato
Obbiettivo	Utilizzabile per machine learning e artificial intelligience	Data analytics e BI, ma limitato	Può essere usato per data anlytics, BI, machine learning e AI
Conformità con ACID	Non conforme ad ACID, problemi di integrità dei dati	Conforme ad ACID: assicura l'integrità dei dati	Conforme ad ACID: assicura consistenza dei dati sia in lettura che scrittura su multiple sorgenti
Costo dello storage	Buon rapporto costo-efficienza, veloce e flessibile	Costoso, richiede molto tempo	Buon rapporto costo-efficienza, facile, permette molta flessibilità, riduce la duplicazione dei dati

Tabella 2.1: Confronto tra Data Platform[11]

Capitolo 3

Approcci utilizzati per la gestione dei dati

Ad oggi, oltre all'evoluzione delle data platform si tende a pensare ad alternative utili per la gestione di queste data platform, e dei dati in generale, che possa essere ottimale. Soprattutto se si considera l'intenzione da parte di un'azienda di utilizzare una singola data platform. Un'azienda al suo interno ha però diversi business domain che utilizzano dati differenti. A volte vengono persino utilizzati data lake differenti, forniti da cloud provider differenti, anche in regioni differenti. Inoltre, i dati di business domain differenti potrebbero anche essere tenuti all'interno dello stesso Data Lake, per esempio, che può portare ad una certa confusione, specie se non viene fatta una corretta governance dei dati che potrebbe portare un Data Lake a trasformarsi in un Data Swamp. In sintesi, avere una singola data platofrm non sempre è sufficiente a raggiungere i risultati attesi. Da qui la necessità di trovare dei nuovi meccanismi. Per ora, infatti, si parla molto di Data Mesh, Data Fabric e Data Virtualization come possibili soluzioni a diversi problemi che si presentano principalmente nella fase di integrazione dei dati. La cosa interessante è che ogni soluzione pensata può essere integrata con le altre portando ad ulteriori miglioramenti. In questo capitolo si parlerà di queste soluzioni e dei loro ambiti applicativi.

3.1 Data Mesh

Sempre più spesso, di recente, le aziende si rendono conto che il modo in cui i loro dati sono immagazzinati e gestiti non è in linea con le loro necessità. Questo perché nonostante esistano nuove data platform, come i Data Lake che permettono di immagazzinare tutti i tipi di dato, questi dati vengono, a volte, immagazzinati in maniera così destrutturata da non renderne semplice il loro impiego. Così, di

recente, è nato questo nuovo paradigma chiamato Data Mesh che ha l'intento di spostare le priorità sulla gestione dei dati, priorità che finora si sono incentrate principalmente nel migliorare le infrastrutture che gestissero i dati, ponendo meno peso, o interesse, su dei paradigmi che avessero come punto di forza proprio i dati. Questo approccio infatti viene definito *data-driven*[13].

3.1.1 I pilastri del Data Mesh

Zhamak Dehghani, una technology consultant che ha teorizzato quali possono essere le basi architetturali e logiche per poter realizzare questo paradigma, sostiene che ci sono 4 pilastri fondamentali su cui si basa[14]:

- Domain-oriented decentralized data ownership and architecture
- Data as a product
- Self-serve data platform
- Federated computational governance

Il primo pilastro punta all'idea che la gestione e la proprietà dei dati dovrà essere autonoma per ogni business domain, ovvero ogni area di lavoro all'interno dell'azienda. In questo modo la responsabilità sui dati verrà decentralizzata spostandosi così dall'attuale architettura monolitica. Per ovviare alle necessità dei vari business domain dovrà essere naturalmente adattata l'architettura monolitica attuale con lo scopo di dare ad ogni business domain la possibilità offrire/servirsi dei dati degli altri business domain. Questo è dovuto al fatto che attualmente si riscontra molta difficoltà nella ricerca e scoperta di dati di qualità[13, 14, 15].

Il primo pilastro favorisce quindi il concetto del secondo pilastro, ovvero l'idea di vedere i dati come prodotto. Infatti, ad ora, per quanto le aziende dicano di avere come priorità proprio i dati non trattano quest'ultimi nei dovuti modi. Da qui nasce la necessità di porre come postulato che i dati debbano essere trattati proprio come un prodotto e che quindi debbano rispondere a tutte le necessità di un consumatore. Non basta più quindi fare una semplice governance dei dati ma è necessario fare in modo che questi dati siano facilmente utilizzabili dai consumatori, questo naturalmente richiede un'infrastruttura adatta che metta in contatto sia i produttori dei dati che i consumatori[13, 14, 15].

Il secondo pilastro pone le basi per il terzo pilastro, "Self-serve data platform". Ogni business domain probabilmente si troverà ad utilizzare tecnologie differenti per gestire i propri dati, una self-serve platform avrà quindi il compito di offrire completa interoperabilità tra le varie tecnologie e infrastrutture utilizzate dai vari business domain minimizzando la difficoltà di accesso a questi dati e rendendola semplice, in modo che ognuno possa servirsi autonomamente dei dati senza che

debba essere specializzato in una specifica tecnologia utilizzata da un altro business domain[13, 14, 15].

Infine, per poter mantenere un'organizzazione generale così ben curata e mirata ai dati è necessario istituire un modello che sia comune a tutti e che dia delle regole sul come i dati debbano essere offerti, favorendo così l'interoperabilità discussa al punto precedente. Per questo motivo Zhamak Dehghani denomina questo modello "federated computational governance"[13, 14, 15].

3.1.2 Come realizzare il Data Mesh

Una delle possibili implementazioni del Data Mesh è proposta da Zhamak Dehghani. In particolare, lei sostiene che sia necessario prima di tutto definire due tipi di domini. Il dominio sorgente ed il dominio consumatore. Il dominio sorgente dovrebbe occuparsi di fare estrazione dei dati e mantenerli in formato grezzo, il dominio consumatore prenderà i dati grezzi per trasformarli e strutturarli in base alle sue necessità. Ovviamente questo sarà possibile creando un ambiente condiviso in cui il dominio sorgente offre dei dati che siano immagazzinati seguendo le regole imposte proprio dal modello federeted computational governance così da essere utilizzabili dal dominio consumatore. Sul come realizzare tutto questo Zhamak Dehghani propone l'utilizzo di un tipo di architettura definita "architettura quantum". Questa architettura viene definita dall' Evolutionary Architecture [16] come la più piccola unità architetturale che può essere distribuita in maniera indipendente, con un'elevata coesione funzionale e che include tutti gli elementi strutturali necessari per la sua funzione. In pratica i Data Product sono visti come singoli nodi realizzati come architettura quantum e contengono al loro interno tre componenti strutturali[13, 14]:

- Il **codice**:
  - Il codice responsabile per l'estrazione, la trasformazione e la distribuzione dei dati provenienti dalle sorgenti.
  - Le API necessarie per accedere ai dati offerti dal nodo, come anche la sintassi degli schemi, le metriche e altri metadati che possano offrire informazioni sui dati prodotti.
  - Altre parti di codice che si occupano delle politiche di accesso ai dati e tecnologie di contorno.
- I **dati** e i **metadati**:
  - In questo caso si parla di dati poliglotti, questo perché si immagina che i dati possano essere serviti in tutte le forme necessarie al consumatore. Questo sarebbe possibile proprio grazie agli schemi e i modelli che vengono

definiti nel modello di federation computational governance che viene usato per definire i metadati legati ai dati. Ad ora vengono già usati i metadati per facilitare l'accesso ai dati ma avere un modello standard su cui fare affidamento ne permetterebbe ovviamente un'ottimizzazione e un miglioramento nell'accesso ai dati.

- L'**infrastruttura**:
  - Altro punto cardine di questa architettura è proprio l'infrastruttura utilizzata per permettere di contenere tutte queste informazioni. Infatti, è importante che l'infrastruttura utilizzata permetta build, deploy e running del nodo, come anche l'accesso a tutti i vari dati contenuti in esso.

3.2 Data Fabric

Se il Data Mesh da un canto cerca di decentralizzare la responsabilità sui dati e la gestione degli stessi, il Data fabric cerca di fare in qualche maniera tutto l'opposto. Infatti, il data fabric è pensato come una soluzione che integra dati provenienti da più sorgenti fornendo un singolo punto di accesso a tutti i processi che vogliono utilizzare quei dati. Nello specifico per la definizione data da Gartner un data fabric: *"è un progetto emergente di gestione dei dati per ottenere pipeline, servizi e semantica di integrazione dei dati flessibili, riutilizzabili e aumentati. Un data fabric supporta casi d'uso sia operativi che analitici, forniti attraverso molteplici piattaforme e processi di distribuzione e orchestrazione. I data fabric supportano una combinazione di diversi stili di integrazione dei dati e sfruttano i metadati attivi, i knowledge graph, la semantica e il ML per aumentare la progettazione e la fornitura dell'integrazione dei dati."*[17] Un data fabric sfrutta quindi l'aiuto sia dell'uomo che della macchina per fare integrazione dei dati, a cui viene fatto accesso in loco. Lo scopo è quello di trovare delle relazioni, tra dati provenienti da contesti totalmente differenti, e offrire informazioni utili di business. La parte più interessante è che tutto questo viene fatto in maniera interattiva offrendo queste informazioni praticamente in tempo reale, cosa che generalmente non sarebbe possibile visto che i dati subiscono i vari processi di ingestione che richiedono tempo.

3.2.1 I pilastri del Data Fabric

Il primo pilastro riguarda il modo in cui i dati dovrebbero essere collezionati, infatti dovrebbe essere utilizzato un meccanismo che permetta al data fabric di accedere, identificare e utilizzare qualsiasi tipo di metadato, da quelli tecnici a quelli operazionali o di business. Questo perché concettualmente il Data Fabric

è pensato proprio per fare integrazione di qualsiasi tipo di dato, se non avesse la possibilità di interfacciarsi con essi verrebbe a mancare uno dei suoi punti cardine.

Per facilitare l'utilizzo di questi dati è necessario anche rendere i metadati da passivi ad attivi, questo è il punto del secondo pilastro. Infatti, si dovrebbe il più possibile creare un ambiente che analizzi continuamente i metadati basandosi su metriche e statistiche chiave in modo da creare un modello a grafo che renda i metadati facili da comprendere basandosi sulle loro relazioni che siano uniche e rilevanti dal punto di vista di business. Infine, proprio grazie ai metadati attivi si abiliterebbero gli algoritmi di Machine Learning e Artificial Intelligence che nel tempo si specializzerebbero nella comprensione di quali dati siano rilevanti.

Uno dei terzi punti chiave della Data Fabric è quello di realizzare quelli che vengono definiti "knowledge graph", ovvero delle reti semantiche che mettono in relazione i dati provenienti da varie sorgenti rendendo più semplice la comprensione di quelle che possono essere delle relazioni utili per il proprio business. Ormai esistono diversi strumenti che permettono di avere un facile accesso ai knowledge graph i quali, utilizzati sia dagli algoritmi di AI/ML, sia dai data anlytics, possono fornire una mappa più chiara del valore dei dati. Infine, ultimo ma non ultimo, uno dei pilastri più importanti per la Data Fabric è proprio una struttura molto robusta che si occupi di fare integrazione dei dati, che non sia limitato solo a quello che si pensa sia relativo all'integrazione dei dati. Infatti, in questo caso si parla di integrazione dei dati a livello più ampio, e questo sistema dovrebbe permettere sia ad utenti che non hanno competenze tecniche sia a quelli che ne hanno di usufruire e gestire facilmente i dati di cui loro hanno bisogno[18].

3.2.2 Data Virtualization

Contrariamente al Data Mesh ed il Data Fabric, si parla da più tempo di data virtualization ma ci sono diverse proposte sul modo in cui si dovrebbe applicare, uno dei punti chiave a cui mira la data virtualization però è molto chiaro: limitare, quando possibile, lo spostamento fisico dei dati dalle loro sorgenti (quindi limitare, se non evitare del tutto, la fase di ingestion) e favorire un accesso diretto ai dati sorgente. Questo singolo punto di acceso a tutti i dati sorgente dovrebbe permettere a chiunque di utilizzare i dati senza sapere necessariamente la loro provenienza. Come si può notare, concettualmente la data virtualization mira un po' a quello a cui mirano anche il Data Mesh ed il Data Fabric. Con le dovute differenze, l'urgenza principale rimane quella di avere la possibilità di poter gestire qualsiasi tipo di dato aldilà della sua provenienza e struttura in modo unificato e comodo[19, 20].

3.2.3 I pilastri della Data Virtualization

La data virtualization consta di 2 fasi principali[19, 20]:

1. La prima fase riguarda l'identificazione delle sorgenti di dati e dei suoi attributi. Si valuta quali sorgenti sono più affidabili a parità di dati offerti. I tool di data virtualization verranno usati poi per definire dei data model che definiscono l'entità coinvolte e creano un mapping fisico con i dati reali. Questi, mirano alla creazione di business object model basati sulla rappresentazione object oriented, permettendo di definire anche le relazioni tra i vari oggetti (ovvero i dati).

2. La seconda fase è quella relativa all'applicazione di questi data model e l'utilizzo di questi tramite qualsiasi linguaggio supportato dallo strumento di data virtualization.

3.2.4 Capacità della Data Virtualization

Le possibilità offerte dalla Data Virtualization sono molteplici, infatti, oltre alla possibilità di offrire al consumatore i dati senza l'onere di sapere quale sia la loro struttura originaria, viene anche migliorata la possibilità di effettuare trasformazione e aggregazione di dati, migliorando così la qualità dei dati offerti. Inoltre si dovrebbe considerare anche il fatto che i dati vengono forniti al consumatore solo quando vengono richiesti, evitando per l'appunto una reale fase di ingestion[19, 20].

3.3 Data platform e tecnologie proposte per venire incontro a questi nuovi approcci

Come da titolo del paragrafo, verranno trattate alcune proposte attualmente presenti sul mercato, per parlare successivamente, nel prossimo capitolo, delle tecnologie utilizzate per realizzare la POC pensata per questo progetto.

3.3.1 Denodo

[Denodo] Piattaforma di virtualizzazione dei dati che permette di accedere, integrare e fornire una vista unificata dei dati provenienti da diverse sorgenti, sia interne all'organizzazione che esterne. Denodo è pensato per agire come uno strato di virtualizzazione dei dati, consentendo agli utenti e alle applicazioni di accedere ai dati in modo trasparente, senza tenere conto della loro posizione o del loro formato. Denodo mantiene una vista logica dei dati ed accede ad essi direttamente alla sorgente senza replicare i dati fisicamente, permette di effettuare operazioni di

trasformazione, aggregazione o arricchimento dei dati in tempo reale utilizzando delle query distribuite e delle modalità di caching dei dati. Offre inoltre sicurezza sui dati e la possibilità di gestire l'accesso ad essi. Molto utile per avere un accesso unificato ai dati in tempo reale[21]. Il lato negativo è che, in presenza di una mole di dati molto vasta, il sistema potrebbe rallentarsi non utilizzando una replica fisica dei dati. Questo lo rende uno strumento molto valido ma per aziende che gestiscono una mole di dati contenuta, inoltre è una piattaforma che non prevede piani gratuiti, se non per prova.

3.3.2 Dremio

[Dremio] Piattaforma Data lakehouse moderna e open-source progettata per semplificare l'accesso e l'analisi dei dati in ambienti di data lake distribuiti. È stata creata per offrire un'esperienza self-service per gli utenti aziendali, consentendogli di esplorare, interrogare e analizzare i dati in modo veloce e intuitvo. Anche Dremio utilizza la virtualizzazione dei dati fornendo una visione unificata dei dati provenienti da diversi fonti e formati. Si basa su tecniche di ottimizzazione delle query e caching dei dati e permette di essere scalato orizzontalmente nel caso il volume di dati da gestire cresca o ci siano dei carichi di lavoro complessi. Inoltre, offre delle funzionalità di auto-discovery dei dati che gli permettono di riconoscere automaticamente la struttura e lo schema dei dati. Infine, fornisce anche dei meccanismi di sicurezza avanzati per proteggere i dati sensibili[22].

3.3.3 Starburst

[Starburst] Sturburst è una piattaforma di query SQL distribuita basata su Trino. Infatti, Starburst Data, l'azienda produttrice di Trino, nasce da dei creatori di Trino con l'intento di spingere al massimo le potenzialità di quest'ultimo. Offre prestazioni elevate durante l'interrogazione dei dati, una vasta gamma di connettori che gli permettono di essere collegato a svariati servizi, è progettato per essere scalabile orizzontalmente in modo da gestire, anch'esso, grandi carici di lavoro e di dati. Starburst fornisce anche delle funzionalità avanzate per gestire la sicurezza e la governance dei dati. Supporta l'autenticazione degli utenti, l'autorizzazione basata sui ruoli e la crittografia dei dati in transito e a riposo. È possibile anche integrare a Starburst diversi strumenti di visualizzazione dati come Tableau e Power BI ed è disponibile sia in una versione a pagamento chiamata Starburst Enterprise che in una versione gratuita chiamata Starburst Community[23].

Capitolo 4

Tecnologie utilizzate per la realizzazione del POC

4.1 Introduzione

In questo capitolo parlerò delle varie tecnologie in cui mi sono imbattuto durante la realizzazione del POC e cosa ha portato alla decisione di integrare queste tecnologie all'interno della piattaforma. Questa piattaforma è stata pensata con la possibilità di integrare il più possibile, anche nel futuro, diverse tecnologie, in modo che possa adattarsi facilmente e velocemente a qualsiasi necessità da parte dell'azienda. Uno dei problemi che si riscontra spesso durante la progettazione di una data platform è la paura che invecchi in fretta o che si possano riscontrare delle difficoltà nel caso in cui si vogliano aggiungere nuovi pezzi al "puzzle". Da questo nasce la voglia di offrire affidabilità quasi a costo 0, considerato che la piattaforma è realizzata principalmente con tecnologie open source. Nella figura 4.1 è possibile vedere la struttura del progetto.

4.2 Il "cuore" della piattaforma: Trino

Il cuore della piattaforma è Trino, un distributed query engine che processa i dati attraverso più server. Per capire di cosa si tratta è necessario prima capire l'architettura di questa piattaforma[24, 25].

4.2.1 Architettura

Trino utilizza due tipi di server, i coordinator ed i worker[25]:

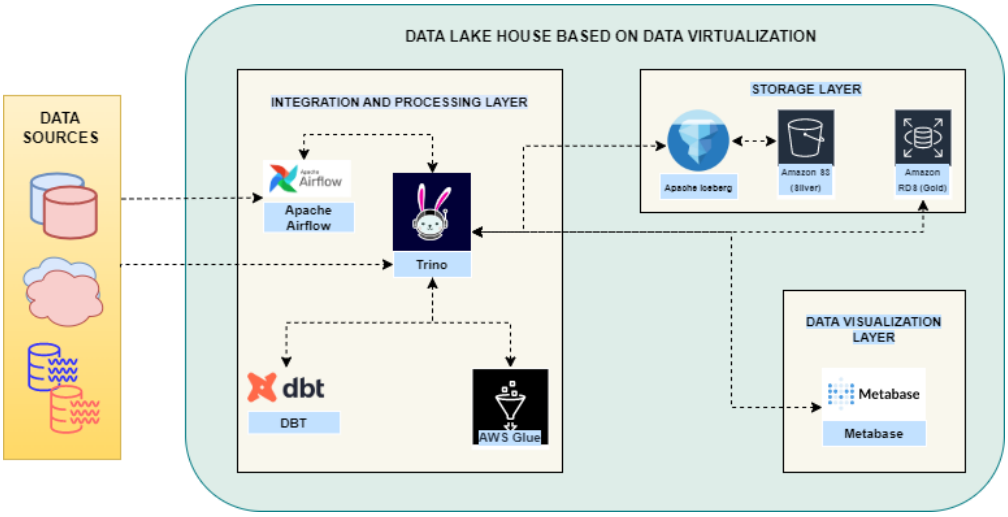

Figura 4.1: Data Platform realizzata nel POC

- **Coordinator**: Il coordinator altro non è che il "cervello" di un cluster Trino. È il punto di contatto per il client ed è quello che riceve e analizza le query lanciate dal client e che decide il piano d'azione per gestire quelle query. Decisi i vari step da attuare invia indicazioni ai worker che effettueranno queste operazioni. Un cluster trino necessita sempre di almeno un coordinator e un worker, che in ambiente di testing possono essere anche realizzati come singola unità. Come viene fatto nell'ambito di questa POC.
- **Worker**: il worker è incaricato di eseguire i task ricevuti dal coordinator e di processare quindi i dati, nel farlo mantiene una comunicazione attiva con gli altri worker. Quando ottiene il risultato lo invia al coordinator che a sua volta lo invia al client che aveva fatto la richiesta.

4.2.2 Come si accede ai dati

Il meccanismo di accesso ai dati si basa su connector, catalog e schema:

- **Connector**: Su Trino è possibile accedere ai dati tramite quelli che vengono definiti come connettori. Si può pensare ad essi come una sorta di driver che permettono di interagire con le API delle altre tecnologie con cui si interfaccia, comprese le sorgenti dati.
- **Catalog**: All'interno di Trino il connettore viene utilizzato per definire quello che si chiama Catalog. Infatti, per definire un catalog viene creato un file di configurazione in cui viene indicato il tipo di connettore che si vuole utilizzare e altri parametri che possono essere utili per connettersi alla sorgente dei

dati (come, per esempio, il metodo ed i parametri per l'autenticazione). È possibile usare più connettori dello stesso tipo per connettersi a sorgenti dati che usano la stessa tecnologia (ad esempio se ci si vuole connettere a due o più PostgresDB verrà utilizzato lo stesso connettore ma con parametri differenti).

- **Schema**: gli schema sono un insieme di tabelle, ogni catalog può avere più schema differenti in cui vengono raggruppate le tabelle.

In una query su trino per accedere ad una tabella specifica verrà utilizzata una sintassi del tipo *catalog.schema.table*. È necessario specificare la differenza tra quello che è uno statement(dichiarazione) e una query(interrogazione). In Trino, infatti, lo statement è scritto in SQL e viene interpretato dal server coordinator che lo trasforma in una query, ovvero un insieme di: task, stage e trasformazioni che portano poi al risultato finale richiesto nello statement. I vari step che portano al risultato vengono effettuati dai vari worker che vengono orchestrati dal coordinator. Tutto il processo è basato su vari stage organizzati con una struttura ad albero, ogni stage ha degli stage "foglia" che aggregano dati e li passano allo stage "radice". Gli stage vengono utilizzati dal coordinator per creare quello che viene definito "query plan", ossia l'insieme di operazioni da effettuare per soddisfare la query. Ma l'esecuzione dei vari stage non avviene all'interno del server coordinatore, ma all'interno dei workers. Gli stage, infatti, non sono altro che un insieme di task che vengono eseguite nei vari workers. I task, a loro volta, sono composti da uno o più driver paralleli che sfruttano più operatori per fare delle operazioni sui dati. Per esempio, se la query prevede un filtro sui dati esiste un operator che si occupa di filtrare i dati. Ogni operator ha dei dati in ingresso, su cui svolge l'operazione e poi fornisce i dati al prossimo operator. Ogni task agisce su un blocco di dati e il risultato dei vari task viene via via aggregato sui vari stage, come detto precedentemente, per fornire il risultato finale della query al coordinator. Tutti i vari scambi di dati tra operator, task e stage vengono effettuati tramite un exchange client interno. Ovviamente tutte queste operazioni essendo realizzate in parallelo sui vari worker rendono tutto il processo molto veloce[25].

4.3 DBT

Un altro strumento fondamentale per la realizzazione di questa data platform è dbt, questo strumento realizzato da dbt Labs è pensato per gestire i processi di trasformazione dei dati, non solo cercando di ottimizzarli ma fornendo diversi strumenti per migliorarli e gestirli più facilmente. Il servizio è fornito in due versioni: dbt Core, che può essere installato e utilizzato gratuitamente, e dbt Cloud a pagamento, gestito, per l'appunto, in cloud da parte dalla compagnia, in cui vengono forniti anche un'interfaccia grafica e altri servizi che migliorano osservabilità

e interazione con altri tool, uno scheduler per i job(l'insieme di operazioni effettuate da questo servizio) e un ambiente di sviluppo integrato[26].

4.3.1 Come funziona DBT

In pratica dbt è basato su quelli che lui definisce data models, i data models sono scritti tramite linguaggio SQL ma non solo, è possibile infatti utilizzare anche un linguaggio chiamato Jinja utilizzato per fare tamplating. Infatti, in fase di compilazione, le parti di linguaggio scritte in Jinja verranno tradotte nei parametri a cui si fa riferimento al loro interno. Essendo un linguaggio di tamplating molto avanzato è possibile realizzare *for loop* o dichiarazioni condizionali al suo interno che rendono la creazione di questi modelli molto dinamica. Oltre a questo, dbt permette anche di richiamare delle funzioni Python che vengono definite all'interno di pacchetti che vengono aggiunti al progetto. Per finire, permette anche di definire dei test che devono essere applicati ai dati in uscita dal modello, di appoggiarsi ad un repository per permettere il versioning durante la realizzazione dei modelli e di far riferimento tramite la funzione ref ad altri modelli, evitando così di dover riscrivere, per esempio, più volte lo stesso modello che parta da dati grezzi, comprendendo da sé le dipendenze tra un modello e l'altro. Tra le altre funzionalità offerte da dbt c'è anche la possibilità di aggiungere quei tipi di dati chiamati seed semplicemente tramite un file CSV e l'utilizzo del comando dbt seeds. I seed sono dati utilizzati per mappare informazioni all'interno di altri db. Per esempio, quando un db contiene dei codici postali e li si vuole mappare con il nome della città si può creare questo db fittizio che contiene i nomi delle città in base al codice postale. Dopo si potrà utilizzare questo db come tutte le altre sorgenti[27].

4.3.2 Strategie di salvataggio dei dati

Dbt offre diversi metodi per rendere o meno persistenti i risultati offerti dai modelli. Queste strategie vengono definite Materializations. Esistono 4 tipi di "materializzazioni"[28]:

- **View**: le view non salvano dati all'interno del db e vengono generate ad ogni run del modello, rendendole molto utili nel caso in cui si debbano fare operazioni semplici come base per altri modelli. Nel momento in cui una view deve fare operazioni più complesse e su una mole di dati maggiore conviene utilizzare una strategia differente.
- **Table**: i modelli che usano questa strategia ricreano la tabella ad ogni run ma le query fatte nei confronti di queste tabelle sono molto veloci. Naturalmente se la tabella comprende delle operazioni complesse impiegherà più tempo per essere ricreata quindi si consiglia di utilizzare questo tipo di strategia

per offrire i dati all'utente finale o come base per molti modelli che devono sfruttare lo stesso tipo di dati e dovrebbero fare molte operazioni complesse prima di poterne avere accesso.

- **Incrmental**: i modelli incrementali sono pensati per inserire o aggiornare delle righe rispetto alle run precedenti. Sono molto potenti ma richiedono una configurazione avanzata dei modelli.
- **Ephemeral**: questo tipo di modelli sono pensati per non scrivere direttamente sul db; invece, il codice di questo modello viene interpolato nei modelli dipendenti da esso. Questa strategia permette di ridurre il disordine nel data warehouse. Il contro di questo tipo di modelli è che non è possibile fare una select direttamente su di essi e che non si possa fare riferimento ad essi tramite la funzione ref durante l'uso delle Operations.

È possibile scrivere i modelli anche in python ma le uniche strategie supportate ad ora sono table e incremental.

4.4 Apache Airflow

Il terzo strumento, anch'esso uno dei 3 pilastri di questa architettura, è Apache Airflow, una piattaforma open source pensata per sviluppare e monitorare i flussi di dati, generalmente utilizzata per fare ingestion e processamento dei dati. La logica di Airflow si basa su quelli che vengono chiamati DAG, ovvero Directed Acyclich Graph. Grazie all'utlizzo di script in Python, infatti, vengono creati questo tipo di grafi aciclici dove ogni nodo è un task, ogni task svolge delle funzioni e può essere connesso ad un task successivo[29].

4.4.1 L'architettura alle spalle di Apache Airflow

L'architettura alle spalle di Airflow è composta da[30]:

- **Scheduler**: questo componente si occupa della programmazione dei DAG, infatti ogni DAG deve avere una data di inizio, in cui verrà eseguito la prima volta, e un intervallo di programmazione che lo scheduler utilizza per programmare la prossima esecuzione del DAG, queste informazioni vengono passate all'executor che si occuperà di eseguire i Task di cui sono composti i DAG.
- **Executor**: questo componente si occupa di eseguire i task all'interno dei DAG e in ambiente di test può essere contenuto all'interno dello scheduler, in ambiente di produzione sarebbe meglio gestire l'esecuzione dei task su

più worker (si potrebbero definire come dei sotto-ambienti sui quali vengono installate tutte le librerie necessarie ad eseguire i task).

- **Webserver**: gestisce l'interfaccia web di Airflow in cui è possibile eseguire i DAG, monitorarli o fare debug di essi. Fornisce tantissime informazioni ed è uno strumento fondamentale.
- **DAGs folder**: in questa cartella vengono contenuti tutti gli script Python dei DAG, tra l'altro Airflow riesce a leggere anche dentro le cartelle compresse ed è in grado di capire se uno script python contiene un DAG o meno.
- **Metadata database**: viene usato da tutti i componenti precedenti per detenere informazioni riguardo i DAG, come logs, stato dei task (fallito o andato a buon fine), date di programmazione e quant'altro.

Ogni DAG è composto principalmente da 3 tipi di elementi:

- Gli **Operator** : uno dei punti di forza più grandi di Airflow, gli operator sono dei task predefinite che permettono di interfacciarsi con tantissime altre tecnologie facilmente, senza dover scrivere ogni volta una Task ad hoc.
- I **Sensors**: un tipo di operator che viene utilizzato generalmente per tenere sotto controllo lo stato di un altro task o di un altro DAG e intraprendere un'azione in base al suo stato (Per esempio, se un secondo DAG deve essere attivato quando un primo DAG ha raggiunto la fine, viene utilizzato un Sensor sul secondo DAG che attende la fine del primo per "attivare" il secondo).
- I **@task**: nelle ultime versioni di airflow è stato aggiunto il decorator @task che può essere utilizzato per definire più facilmente quelli che erano conosciuti come Python Operator ovvero degli operatori utilizzati per eseguire "pezzi" di codice python

All'interno dei DAG i vari Task possono avere delle dipendenze, per esplicitare la dipendenza di due o più task vengono utilizzati "»" o "«", se si scrive "task1 » task2" si indica che a seguito del task1 verrà eseguito il task2, scrivendo "task1 « task2" si indica la stessa informazione, ovvero il task2 dipende dal task1.

4.4.2 Le potenzialità di Apache Airflow

Quello descritto finora è solo il meccanismo con cui funziona Airflow, ma al suo interno racchiude tantissime capacità e possibilità. Essendo una piattaforma open source viene costantemente aggiornata e mantenuta da molti membri, grazie a questo viene infatti migliorata costantemente sia l'interfaccia utente per una maggiore fruibilità da parte degli utenti di strumenti di debug e monitoring dei

loro flussi, sia il numero di operatori e connettori messi a disposizione degli utenti. Airflow permette infatti di creare variabili di ambiente personalizzate che possono essere utilizzate all'interno dei DAG e connessioni verso tantissime sorgenti di dati facilitando quindi l'interazione con essi. Inoltre visto che viene utilizzato python per creare i DAG si ha la possibilità di sfruttare anche le potenzialità di python stesso nella gestione e pulizia dei dati. Come se non bastasse esistono anche operatori Bash che permettono di eseguire script in bash all'interno del DAG o Branch Operator che permettono di scegliere quale task eseguire successivamente in base al risultato che si è ottenuto o si vuole ottenere. Data quindi la sua dinamicità è possibile riuscire a non farsi trovare impreparati nel momento in cui si presenta una nuova sorgente di dati[30].

4.5 Apache Iceberg

Nasce con l'intento di migliorare i metodi di archiviazione tradizionali, è un framework open source che permette di gestire dati di grandi dimensioni. È possibile eseguire cancellazioni, aggiornamenti o inserimenti nelle tabelle senza comprometterne la consistenza. Inoltre Apache Iceberg supporta una forma di partizionamento nascosta dei dati che organizza autonomamente dati in base a criteri scelti come data. Tutto questo naturalmente rende non solo più semplice l'interrogazione dei dataset, ma anche più veloce e meno prona agli errori da parte dell'utente. Un altro dei punti a suo favore è l'integrazione con diversi strumenti tra cui Spark, Hive e Trino e la possibilità di utilizzare un linguaggio SQL-like per svolgere operazioni sulle tabelle. In più, è possibile interrogare delle versioni precedenti delle tabelle, o la tabella in una specifica data. Questo può essere molto utile nel caso in cui si vuole monitorare la variazione dei dati nel tempo o si vuole mantenere lo storico di questi dati[31, 32, 33].

4.6 Metabase

Si tratta di una piattaforma open source pensata per facilitare la visualizzazione di dati anche ad utenti che non hanno grande manualità con il linguaggio SQL, ma fornisce comunque la possibilità di utilizzarlo e per questo è molto potente. È possibile collegare a Metabase diverse sorgenti dati, infatti fornisce diversi connettori tra cui quelli per Trino. Quando la sorgente viene collegata, Metabase fa già un'analisi preliminare dei dati e dà la possibilità all'utente di avere già alcune informazioni sui dati senza dover fare nulla. Ovviamente dà la possibilità di creare dashboard interattive in modo semplice e veloce e di effettuare operazioni di aggregazione, filtraggio o ordinamento (che comunemente si effettuerebbero tramite codice SQL) tramite un'interfaccia molto intuitiva e facile da utilizzare. Inoltre,

Metabase offre anche delle funzionalità per la condivisione dei report che consentono agli utenti di condividere facilmente dashboard e report con i membri del team o con altre parti interessate. Fornisce anche la possibilità di collegare un canale Slack (piattaforma utilizzata molto spesso per la comunicazione e l'organizzazione all'interno dei team) per automatizzare alcuni tipi di comunicazione. Infine, è possibile gestire la sicurezza ed il controllo degli accessi. L'admin infatti può decidere chi può trattare alcuni tipi di dato, modificare le dashboard o avere visibilità su specifiche tabelle piuttosto che altre. Come detto inizialmente, Metabase è un progetto open source che viene supportato e sviluppato attivamente dalla sua comunità e offre una versione sia su Docker che come applicazione web eseguibile in locale[34].

4.7 PostgreSQL

PostgreSQL, chiamato comunemente Postgres, nasce all'incirca negli anni '80 nell'Università della California a Berkeley sotto il nome inizialmente di INGRES. È un sistema di gestione di database relazionali, open source, molto potente e molto affidabile. È conosciuto per[35]:

1. Affidabilità e robustezza: è un sistema molto stabile che supporta le transizioni ACID e sul quale è possibile configurare replicazioni dei dati e backup per garantirne una maggiore disponibilità dei dati.

2. Le sue funzionalità: infatti PostgreSQL offre molte funzionalità che mirano a migliorare la sua usabilità permettendo di effettuare facilmente join complessi, funzioni aggregate complesse, subquery e altro.

3. Linguaggi utilizzabili: su Postgres viene utilizzato un linguaggio chiamato PL/pgSQL, che permette di definire funzioni personalizzate, procedure e operazioni di analisi dei dati complesse tramite un linguaggio SQL like, in più supporta anche altri tipi di linguaggi come PL/ Python, PL/Perl, PL/Java e altri. Tutto questo lo rende molto comodo e potente per l'utente.

4. Scalabilità: una delle peculiarità, che rende Postgres molto apprezzato all'interno delle aziende, è il fatto che si adatta molto bene anche a grandi moli di dati, offrendo meccanismi per il partizionamento dei dati e quindi una migliore gestione negli scenari di elevato carico di lavoro.

5. Comunità: è un progetto con una comunità molto attiva e numerosa che lo supporta. Offrendo documentazione, aggiornamenti e consigli sul come utilizzarlo.

6. Compatibilità: è compatibile con una vasta gamma di piattaforme, tra cui Linux, macOS e Windows, e anche per questo motivo viene scelto da una grande varietà di aziende, dalle più piccole alle più grandi.

4.8 Tecnologie AWS

4.8.1 AWS EC2

Amazon Elastic Compute Cloud (Amazon EC2) è un servizio da Amazon Web Services (AWS) che consente di eseguire le applicazioni su server virtuali configurabili in modo flessibile, questi server sono chiamati EC2. Le EC2 sono scalabili e flessibili, infatti è possibile modificare lo spazio di archiviazione, le risorse di calcolo o la larghezza di banda in modo rapido ed efficiente dall'interfaccia utente offerta da AWS. Inoltre, è possibile scegliere l'istanza più adatta all'uso che se ne deve fare e i prezzi variano anche in base al tipo di macchina utilizzata, oltre al tipo di macchina è possibile anche scegliere il tipo di sistema operativo che si vuole utilizzare sulla macchina, alcuni dei sistemi operativi disponibili sono: Amazon Linux, Windows Server, ubuntu, CentOS. È possibile pagare in base all'utilizzo della macchina e arrestarla quando nel tempo in cui non verrà utilizzata, ovviamente si continuerà a pagare lo spazio di archiviazione. Ovviamente l'EC2 si integra bene con gli altri servizi di Amazon come Amazon Elastic Block Store (EBS) che fornisce gli spazi di archiviazione persistente, o Amazon Simple Storage Service (S3) per l'archiviazione di oggetti, o la VPC (Virtual Private Cloud) per isolare la rete, e così via. La sicurezza nelle istanze EC2 è gestita tramite l'utilizzo di chiavi crittografiche, la gestione dei gruppi di sicurezza per il controllo degli accessi, la crittografia dei dati in transito e a riposo e la possibilità di configurare le regole di firewall per proteggere le istanze[36].

4.8.2 AWS S3

Amazon Simple Storage Service è un servizio che permette di archiviare dati come oggetti. È scalabile e sicuro ed è fornito da AWS, è progettato per gestire grandi quantità di dati e permette un'archiviazione ed un recupero facile e veloce. AWS S3 archivia i dati in buckets, ovvero "secchielli" che contengono oggetti con una chiave univoca all'interno del bucket. Le ragioni per cui si scelgono degli storage in Cloud, generalmente, sono proprio la necessità di scalabilità, disponibilità, affidabilità e durabilità. Tutte caratteristiche che AWS S3 riesce ad offrire consentendo di:

- Aumentare o ridurre la capacità dei bucket, in base alla necessità, senza interruzioni.

- Assicurare la durabilità replicando automaticamente i dati in zone diverse della stessa regione

In aggiunta, offre un accesso sicuro ai dati e la possibilità di gestire gli accessi ai dati, permettendo di decidere chi può accedere ai dati e cosa può farne, grazie anche all'utilizzo di AWS IAM. Infine, permette di gestire anche il ciclo di vita dei dati automatizzando il passaggio di dati tra diverse classi di archiviazione, come Standard, Infrequent Access (IA), Archive e Glacier, in base alle esigenze di accesso e costo. Naturalmente si integra bene con gli altri servizi di AWS, tra cui AWS Glue e Amazon Athena, e viene utilizzato in una grande varietà di casi d'uso. Dal backup dei dati alla distribuzione di contenuti o la condivisione di file. Può essere addirittura utilizzato anche per l'hosting di siti web statici[37].

4.8.3 AWS RDS

RDS sta per Relational Database Service, questo servizio di AWS infatti offre un database relazionale completamente gestito e fornito da AWS. È una soluzione semplice per configurare, gestire e scalare database relazionali in ambienti cloud. Supporta una vasta gamma di database relazionali tra cui Amazon Aurora, MySQL, PostgreSQL, Oracle Database e Microsoft SQL Server. Questo permette di scegliere il database più appropriato per il proprio caso d'uso. Amazon RDS si occupa di attività come provisioning, backup, patching e monitoraggio delle prestazioni. Questo permette all'utente di non doversi preoccupare di queste pratiche insidiose e prone agli errori, dedicandosi così al semplice utilizzo dei suoi dati. Ovviamente questo porta con sé una grande tollerabilità ai guasti e grazie alla sua scalabilità è in grado di gestire e istanziare in maniera autonoma le risorse necessarie per il carico di lavoro a cui è sottoposto. Come gli altri servizi di AWS offre sicurezza e integrazione con gli altri servizi AWS, alcuni degli strumenti più interessanti che offre però sono quelli di monitoraggio, che permettono di rintracciare e intervenire tempestivamente in caso di eventuali problemi[37].

4.8.4 AWS IAM

Servizio di gestione degli accessi fornito da Amazon Web Services, Identity and Access Management (IAM) consente di controllare in modo centralizzato gli utenti, i gruppi e le autorizzazioni per l'accesso e la gestione delle risorse AWS. È possibile creare utenti e gruppi a cui affidare alcuni tipi di permessi. Ogni servizio o risorsa all'interno di AWS ha un set di permessi che possono essere dati o revocati. Permettendo una gestione sicura dell'accesso alle risorse. IAM supporta l'autenticazione a più fattori offrendo così un ulteriore livello di sicurezza. Inoltre, permette di monitorare l'attività degli utenti IAM e rilevare eventuali attività sospette[38].

4.8.5 AWS Glue

Servizio di elaborazione dei dati gestito e fornito da AWS. È progettato per semplificare e automatizzare il processo di estrazione, trasformazione e caricamento (ETL) dei dati per l'analisi. AWS Glue fornisce un catalogo dei dati centralizzato in cui vengono registrate le informazioni sullo schema e sulla struttura dei dati. Il catalogo consente di scoprire, comprendere e gestire i dati in modo più efficiente. Inoltre, permette di definire e automatizzare il flusso di lavoro di trasformazione dei dati. Utilizza un'interfaccia visuale per definire le operazioni di trasformazione e genera automaticamente il codice sottostante necessario per eseguire le trasformazioni. Permette di analizzare i dati all'interno di varie sorgenti identificando in maniera autonoma gli schemi e le relazioni dei dati. Questo permette di accelerare la creazione dei flussi di lavoro per la trasformazione dei dati. Naturalmente, come tutti i servizi in cloud, è scalabile e lo fa in maniera autonoma, in base alla necessità. È serverless, quindi non è necessario gestire l'infrastruttura sottostante, permettendo di avere un focus sulle attività di ETL. Sin integra bene con gli altri servizi offerti da AWS e la sicurezza dei dati viene gestita da AWS. È possibile aggiungere, anche in questo caso, delle politiche di accesso ai dati con AWS IAM[39].

4.9 Docker

Si tratta di una piattaforma open source che permette di creare, distribuire e gestire delle applicazioni all'interno di container. In pratica, l'idea dietro Docker nasce dal fatto che, molto spesso, quando viene creata un'applicazione nel proprio terminale questa funzioni grazie a dipendenze e tecnologie presenti su di esso. Nel momento in cui la stessa applicazione viene però spostata in un altro ambiente probabilmente si riscontreranno problemi nella sua esecuzione. Da qui la necessità di creare delle immagini (vengono così definite in Docker) che possono essere eseguite dentro dei container (ambiente indipendente), evitando così di essere dipendenti dalla macchina in cui vengono eseguite. Questo naturalmente ne facilita la distribuzione e l'utilizzo. In ogni immagine docker vengono definite le tecnologie e le dipendenze necessarie ad eseguire l'applicazione. Quando questa immagine viene eseguita dentro il container verranno quindi create tutte le risorse necessarie alla sua esecuzione. È possibile definire le immagini Docker attraverso un Dockerfile e distribuirle tramite una repository pubblica offerta da Docker chiamata Docker Hub. Infine, uno strumento molto utile offerto da Docker è Docker Compose. Questo strumento permette di creare un file YAML in cui vengono indicati più applicazioni o servizi all'interno di un progetto, come ad esempio un server ed un client, che verranno eseguiti su container differenti. Questo è possibile elencando all'interno del file YAML il nome dell'immagine docker da utilizzare con annesse dipendenze verso gli altri servizi o reti virtuali che i container possono utilizzare per comunicare.

Il concetto di applicazioni all'interno di container offerto da Docker sta un po' alla base di un paradigma sempre più utilizzato per le applicazioni moderne che cercano sempre di più di allontanarsi da sistemi monolitici per spostarsi su sistemi basati su microservizi, ovvero ambienti in cui un'applicazione è composta da tante sotto applicazioni che possono esistere in maniera indipendente dalle altre[40].

Capitolo 5

Realizzazione POC

5.1 Introduzione

Uno dei problemi più comuni al giorno d'oggi, come citato all'inzio, è proprio quello dell'evoluzione tecnologica e dell'aumento costante della produzione di dati, come se non bastasse però non solo vengono prodotti sempre più dati ma la loro tipologia e i vari formati cambiano e si evolvono in base alla necessità. Questo spesso porta le aziende a fare scelte di progettazione che inizialmente sembrano vincenti e adeguate alle loro necessità, in breve tempo queste scelte però diventano obsolete e si corre ai ripari. Le aziende si trovando quindi a cercare soluzioni, spesso parziali, che possano mettere una pezza finché dura. Tutto questo contribuisce all'aumento del "debito tecnico", generalmente durante la realizzazione di un progetto si prevede anche il "debito tecnico", in questi casi però rischia di crescere esponenzialmente e fuori controllo. Di recente, quindi, si sta cominciando a cercare delle soluzioni che possano essere dinamiche, o soluzioni che prevedano dei paradigmi rigidi che possano limitare il più possibile il caos. Per questo motivo, prima di realizzare questo POC, ho deciso di studiare alcune delle soluzioni proposte da alcune aziende. Una di queste è stata poi grande fonte di ispirazione per la realizzazione di questo POC. In generale queste aziende tendono a "fare l'occhiolino" alla Data Virtualization. In questo capitolo analizzerò i vari layer che compongono una data platform cercando di spiegare cosa mi ha portato alla scelta di una tecnologia piuttosto che un'altra. Infine, spiegherò i possibili utilizzi della piattaforma, i suoi punti di forza.

5.2 Livelli della Data Platform

Come detto precedentemente, una data platform è composta da più livelli, ognuno dei quali svolge un compito fondamentale per ottenere dati di qualità e che abbiano valore economico per un'azienda. Nella mia POC ho deciso di unire due dei livelli

che generalmente vengono mantenuti divisi. Mi riferisco al livello che si occupa dell'ingestione di dati e quello che si occupa del loro processamento. Essendo Trino il cuore di questa piattaforma lo si può immaginare come il punto di riferimento per le altre tecnologie che gli girano attorno quindi in questo caso era difficile, ma soprattutto insensato, dividere questi due livelli che sfrutteranno le potenzialità di Airflow e DBT tramite Trino. Gli altri due layer fondamentali naturalmente sono quello di:

- **Archiviazione**: che include i servizi e le tecnologie utilizzate per immagazzinare i dati
- **Visualizzazione**: che permette di visualizzare i dati in modo semplice e chiaro.

Come detto precedentemente, questi livelli avranno come filo conduttore Trino. Trino ha una grande varietà di connettori che gli permettono di collegarsi a svariate sorgenti di dati, ma non solo. Infatti, grazie alla sua vasta community, sono state realizzate sempre più librerie che possano permettere di integrarlo in altri strumenti. Questo è il caso di:

- **Airflow**: in cui è stato creato un connettore e degli operatori per Trino.
- **DBT** in cui è stata creata una libreria chiamata dbt-trino.
- **Metabase** per cui Starburst ha creato dei drivers, ma considerato che Starburst è basato su Trino vanno ugualmente bene per quest'ultimo.

Questi naturalmente sono solo alcuni dei connettori che sono stati creati per integrare Trino. Godendo anche di una vasta community e popolarità continuerà probabilmente anche ad evolversi.

5.2.1 Livello di Ingestione e Processamento

Il livello di ingestione e processamento verrà gestito da Airflow e DBT, che usano come struttura alle spalle proprio Trino. In questo paragrafo andrò a spiegare per entrambe le tecnologie cosa mi ha portato alla loro scelta. Per quanto riguarda Airflow, gli altri framework o piattaforme valutate erano Dagster, Azure Data Factory e AWS Glue. Tutte queste piattaforme si occupano di gestire l'estrazione dei dati, con la differenza che Glue, come spiegato precedentemente, permette di gestire anche la parte di trasformazione e caricamento dei dati. Ho valutato Azure Data Factory ma il suo costo e la relativa capacità di personalizzazione dei flussi di lavoro, nonché i suoi vincoli con la piattaforma Azure, mi hanno portato a scartarla molto presto. A seguire ho valutato Glue ma anche in questo caso avevo inizialmente scartato questa risorsa perché anch'essa legata al mondo AWS, la mia

idea inizialmente era quella di realizzare una piattaforma che avesse più componenti open source possibili e che potesse essere facilmente collegabile a qualsiasi risorsa sorgente e che avesse anche meno vincoli possibile con le piatteforme utilizzate per lo storage 'on Cloud'. Così la scelta si è ridotta a due tecnologie che di recente hanno preso molto piede, la prima è Dagster, l'altra è Apache Airflow. La differenza tra Airflow e Dagster non è molta, ma quanto basta per avermi fatto preferire l'una all'altra. Anche Dagster infatti si occupa di orchestrare e gestire i flussi di lavoro e l'automazione di operazioni di dati. Con la differenza che[41, 29]:

AIRFLOW	**DAGSTER**
Ha molta più libertà nell'orchestrazione e generazione dei flussi, dando spazio a flussi di lavoro basati su task e DAG temporizzati.	Si basa su un paradigma di orchestrazione di dati concentrandosi sulla creazione di pipeline di dati affidabili e scalabili con un occhio mirato alle dipendenze sulle operazioni e la gestione di dati in input e output.
Si basa su DAG ma composti da task che possono essere eseguiti in sequenza o in parallelo	Si basa anch'esso sui DAG ma ha una logica più stringente e meno "libera" basata sulle dipendenze delle operazioni e dei dati
Non ha un focus sull'affidabilità dei dati, dovrebbe essere implementato manualmente	Ha un focus sull'affidabilità dei dati
I DAG vengono scritti in Python ma i vari operatori permettono di utilizzare diversi linguaggi all'interno del DAG, in più permette di utilizzare file yaml per la configurazione	I DAG vengono scritti in python
Vasta gamma di connettori e integrazione con altri strumenti e servizi	Ecosistema in crescita ma poche integrazioni predefinite ancora
Supporta l'integrazione con DBT	Ha un'ottima integrazione con DBT nativa

Tabella 5.1: Confronto tra Apache Airflow e Dagster

La tabella 5.1 ci porta a parlare a questo punto della scelta, perché scegliere DBT e non Apache Spark (altro strumento offerto da Apache per gestire dataframe, stream o HDFS). E se l'integrazione con DBT è nativa su Dagster, perché scegliere Airflow? La risposta a queste due domande in realtà è semplice. DBT è uno strumento nuovo e "fresco" che permette di gestire la sicurezza e l'affidabilità dei dati, andando a sopperire questa mancanza di Airflow (che invece è presente in

Dagster). D'altra parte però può avere delle mancanze in fatto di integrazioni o librerie che possono essere sopperite a loro volta da Airflow vista la sua vasta quantità di operatori e connettori. Questa dualità permette alla piattaforma di essere stabile e pronta a gestire le novità sia ora che in futuro. Inoltre un' eventuale migrazione a Dagster in futuro sarebbe anche possibile magari visto che possono essere scritti i dag partendo dai models, è una porta lasciata aperta si può dire. Ma nel breve periodo Airflow sembra la scelta più affidabile, oltre il fatto che sempre più aziende cominciano ad integrarla al suo interno. Come per esempio Microsoft Azure che ha realizzato una versione del suo servizio Data Factory con una integrazione di Airflow, chiamato Managed Airflow. Un altro punto a favore di DBT è il fatto che i suoi models potrebbero essere un ottimo strumento per realizzare i così detti Data Product di cui si parla nel Data Mesh.

5.2.2 Livello di archiviazione

Per il livello di archiviazione mi è stata offerta dall'azienda in cui ho svolto la tesi la possibilità di utilizzare un servizio AWS RDS che ho utilizzato per la realizzazione del database PostgreSQL e un AWS S3 come object storage per il datalake. Avevo valutato inizialmente l'utilizzo di altre due macchine, una in cui installare il database PostgreSQL e un'altra dove installare un object storage open source chiamato Minio, in questo caso però gestire la loro eventuale scalabilità come anche l'affidabilità poteva essere scomodo e poco fruttuoso, quindi la scelta di questi servizi era la più valida. Inoltre, il S3 viene utilizzato tramite Apache Iceberg, che a sua volta sfrutta AWS Glue come Hive metastore.

Partiamo però dal parlare di cos'è un Hive Metastore e a cosa serve.

Un Hive Metastore è uno strumento utilizzato per mappare i dati che vengono inseriti all'interno del datalake, è uno strumento fondamentale perché permette di interrogare i dati come se fossero all'interno di un database tradizionale, utilizzando quindi anche un linguaggio SQL-like. Un Hive metastore contiene quindi una serie di informazioni che tengono traccia di dove sono locati i dati, qual è la loro struttura e come raggiungerli. Per quale motivo quindi utilizzare Apache Iceberg appoggiandosi ad AWS Glue? Le potenzialità di Apache Iceberg sono tantissime, come abbiamo visto precedentemente. Sempre più spesso infatti si cerca di utilizzare strumenti di questo tipo e alcune aziende hanno realizzato degli strumenti simili come Google BigQuery e Snowflake, una cosa che manca ad Apache Iceberg è la possibilità di fare query distribuite, capacità integrate in BigQuery e Snowflake. Qui entra in gioco Trino, che supporta le query distribuite e sopperisce a questa mancanza in Apache Iceberg. Apache Iceberg per essere collegato a Trino e sfruttare l'S3 come spazio di archiviazione che ha bisogno di un hive metasotre, AWS Glue fa quindi da metastore per Apache Iceberg. In pratica quando Apache tenta di creare nuove tabelle/file, AWS Glue intercetta la richiesta e salva informazioni utili

per il recupero dei dati mentre questi vengono inseriti nello storage S3. Al posto di AWS Glue era possibile utilizzare un hive metastore creato ad hoc, che avesse alle spalle un database come redis, ma anche in questo caso poteva essere oneroso da gestire nel futuro. Nelle varie data platform moderne viene spesso utilizzata anche un'architettura chiamata "medallion architecture". La Medallion architecture infatti prevede generlmente l'utilizzo di più livelli di archiviazione. Comunemente sono presenti 3 livelli: Bronzo, Argento e Oro. Nel livello Bronzo vengono salvati i dati grezzi, che vengono successivamente puliti parzialmente per essere spostati nel livello Argento, infine i dati subiscono un ulteriore processamento per essere spostati al livello oro, che comprende dati che abbiano realmente un valore di business. Naturalmente la scelta di quanti livelli utilizzare dipende molto da quanto tempo e soldi si hanno a disposizione, subendo tre fasi di processamento infatti i dati richiedono molte risorse, non solo economiche visto che bisogna mantenere 3 livelli di archiviazione, ma anche di tempo richiesto per tutto il processo al fine di avere dati puliti[42]. Anch'io per la piattaforma ho deciso di utilizzare questo tipo di architettura, eliminando però il livello Bronzo. Successivamente verrà spiegata la ragione di questa scelta.

5.2.3 Livello di visualizzazione

Per il livello di visualizzazione dei dati la scelta era vasta visto che esistono tantissimi strumenti per la visualizzazione dei dati tra cui i più famosi sono PowerBI offerto da Microsoft e Tableau. Ovviamente entrambi gli strumenti richiedono una sottoscrizione a pagamento, per questo cercando ho trovato uno strumento open source che sempre più cerca di farsi strada tra le grandi aziende, si tratta di Metabase. Questa piattaforma ha delle potenzialità enormi visto che, oltre ad offrire una versione a pagamento in Cloud, ne offre anche una gratis, installabile come applicazione java. L'alternativa a Metabase che ho tenuto inizialmente in considerazione era Apache Superset. La differenza chiave tra le due tecnologie è l'interfaccia utente. Sembrerebbe una ragione banale, ma l'esperienza utente e la facilità di utilizzo di uno strumento cambia drasticamente le carte in tavola, soprattutto se si considera che chi dovrà utilizzare quello strumento non ha conoscenze tecniche molto approfondite. Oltre a questo, Metabase offre già un buon quantitativo di connettori ed utilizzarli per collegarsi ad una nuova sorgente è molto semplice e intuitivo, al contrario di Apache Superset che richiede qualche conoscenza in più. Naturalmente l'altro lato della medaglia è che Apache Superset permette di fare alcune operazioni più complesse che Metabase non permette di fare, ad ora. Vorrei sottolineare "ad ora" perché questo strumento è in costante crescita, quindi nel tempo potrebbe sicuramente ricevere degli aggiornamenti che ne migliorino la qualità. Perché, però, Trino è molto utile con uno strumento di data visualization di questo tipo? È presto detto, Trino infatti fornendo la possibilità di

accedere a tutte le sorgenti di dati ad esso collegato permetterebbe all'utilizzatore di questo strumento di monitorare e osservare i dati anche prima che essi siano stati "ingeriti" dalla piattaforma, o nel loro livello intermedio, oppure semplicemente nel loro livello finale. In pratica apre la possibilità ad interrogare o aggregare i dati in qualsiasi momento. Se ci fossero dei dati di cui non avremmo necessità di fare ingestione per fare delle aggregazioni sarebbe possibile farlo, evitando di occupare spazio nella piattaforma magari. Come se non bastasse, come detto precedentemente, Metabase permette di "passare ai raggi-x" (dicitura utilizzata nel programma) le varie sorgenti di dato e le varie tabelle, fornendo sin da subito delle informazioni utili su di esse.

5.3 Le potenzialità della piattaforma

Durante la realizzazione della piattaforma ho tenuto in considerazione più casi d'uso della stessa, che andrò ad affrontare durante questo paragrafo. Proprio grazie alle potenzialità di Trino, in concerto con l'uso degli altri strumenti, questa piattaforma dovrebbe sopperire alla maggior parte delle richieste da parte delle aziende che vogliono gestire dati provenienti da più fornitori o business unit differenti. Naturalmente anche questa piattaforma ha delle limitazioni, che discuteremo nel capitolo conclusivo insieme a delle possibili migliorie che potrebbero essere apportate ad essa.

5.3.1 Più tipi di ETL

L'ETL all'interno di questa piattaforma può essere gestita in più modi. La scelta di che tipo di ETL eseguire dovrebbe dipendere dalla quantità di dati che arrivano dalla sorgente e dal tipo di dati. Se prevediamo che la tabella di cui fare ingestion sia una tabella nell'ordine dei MB non è il caso di far passare questa tabella da un livello Argento ma è possibile creare un Data Model per questa e fare l'ingestion ed il processamento di essa direttamente tramite DBT ed Airflow, saltando il livello Argento e salvando i dati direttamente nel livello Oro. Questa metodologia, essendo una tabella di piccole dimensioni, risparmia spazio e tempo, inoltre DBT fornisce la possibilità di salvare alcune tabelle come "ephemeral", quindi se dovesse essere necessario utilizzare la stessa solo con il fine di fare aggregazione potrebbe essere utilizzata nello step iniziale e poi scartata prima di essere salvata. Un'altra possibilità è quella invece di utilizzare Apache Iceberg, che si trova nel livello Argento, dove verranno salvati i dati provenienti dalle varie sorgenti a seguito di parziali pulizie su di essi. Da lì poi verrà utilizzato DBT per processare i dati e spostarli sul livello Oro. La fase di ingestion da dati sorgente a livello Argento può essere gestita tramite Airflow, che con la sua varietà di connettori permette di fare ingestion di qualsiasi tipo di dato da qualsiasi sorgente, che siano essi semplici

file csv, xmls o json, o che si tratti di API. In realtà anche DBT è pensato per lavorare con Apache Iceberg, sfortunatamente le librerie utilizzate per farlo hanno ancora qualche difetto che non rende stabile questo processo, quindi, nonostante abbia tentato di sfruttare questa possibilità, ho deciso di considerare solo Airflow nella fase iniziale di ingestion per mantenere solida la struttura della piattaforma. Durante la seconda fase dell'ETL, da Argento a Oro, può essere utilizzato DBT per processare i dati. Questo tipo di ETL è consigliata soprattutto in presenza di grandi moli di dati provenienti dalla sorgente. Dove trasportare tanti dati dalla sorgente più volte durante una giornata potrebbe anche diventare oneroso e dispendioso, quindi è più sicuro ed economico spostare i dati prima su un livello Argento. Ma come funziona nello specifico questo processo?

Analizziamo il processo per step:

1. DBT basa tutto sui dbt project. Un dbt project viene definito tramite un file YAML nel quale vengono indicati alcuni parametri di configurazione tra cui sorgente e destinazione dei dati. Essendo un tool pensato per fare trasformazione, per il database sorgente e destinazione è possibile usare una singola connessione sorgente, ad esempio se utilizzo un connettore sorgente di RedShift le tabelle verranno prese da RedShift, trasformate e reinserite all'interno di RedShift, c'è la possibilità di scegliere un database differente come target ma rimane vincolato sempre alla stessa connessione RedShift. Essendo Trino però collegato a più sorgenti è possibile offrire a dbt la connessione di Trino facendogli credere che la connessione sia unica, ma in realtà verranno utilizzati database, schemi e tabelle presenti nelle altre sorgenti tramite la connessione di Trino. In questo modo si hanno due migliorie:

   - DBT permette di fare trasformazione cross-connection.
   - I dati vengono interrogati più velocemente e in modo sicuro da Trino

   In questo progetto ad esempio viene indicato come:

   - Database sorgente **iceberg**: object storage S3 che è stato collegato a Trino tramite AWS Glue, come spiegato precedentemente.
   - Database destinazione **business**: PostgreSQL realizzato utilizzando AWS RDS e collegato a Trino tramite il connettore per PostgreSQL.

   Per definire questi due database è necessario definire un ulteriore file YAML chiamato profiles. Come mostrato nell'immagine 5.1. In questo caso ho voluto inserire anche il datalake in cui utilizzo Iceberg come altro database da cui prendere i dati, agli occhi di trino sarà un'altra sorgente e un altro profilo, nonostante siano la stessa sorgente. Lo scopo naturalmente era solo a fine di testing. Visto che tutte le risorse vengono eseguite all'interno della stessa

```yaml
business:
  target: dev
  outputs:
    dev:
      type: trino
      method: none  # optional, one of {none | ldap | kerberos | oauth | jwt | certificate}
      user: test
      host: localhost
      port: 8080
      schema: public
      database: business
      threads: 1
      retries: 3 # default: 3

datalake:
  target: dev
  outputs:
    dev:
      type: trino
      method: none  # optional, one of {none | ldap | kerberos | oauth | jwt | certificate}
      user: test
      host: localhost
      port: 8080
      schema: "\"sep-demo-db\""
      database: datalake
      threads: 1
      retries: 3 # default: 3

iceberg:
  target: dev
  outputs:
    dev:
      type: trino
      method: none  # optional, one of {none | ldap | kerberos | oauth | jwt | certificate}
      user: test
      host: localhost
      port: 8080
      schema: "\"sep-demo-db\""
      database: iceberg
      threads: 1
      retries: 3 # default: 3
```

Figura 5.1: Profili di DBT

macchina non sono stati utilizzati metodi di autenticazioni, naturalmente dovrebbero essere implementati in caso di un utilizzo reale. Si può notare come tutte le sorgenti usino Trino come punto di connessione, se utilizzassimo due connessioni differenti DBT non ce lo permetterebbe. La figura 5.2 mostra com'è fatto, in questo caso, il file YAML per il dbt project.

2. Dopo aver definito il dbt project ed i profili, è possibile indicare in un altro file YAML le sorgenti dei dati (come in figura 5.3, funzionalità di DBT che permette di indicare schemi e tabelle delle sorgenti compreso il nome dei database di cui fanno parte

3. Infine è necessario definire i modelli, nel progetto sono stati definiti dei modelli che fanno delle semplici SELECT dalla sorgente per spostare i dati nella destinazione(Figura 5.4), ovviamente DBT permetterebbe di creare vere e proprie pipeline che comprendono diverse trasformazioni al suo interno. Lo scopo nel mio caso era dimostrare che fosse possibile interagire con le varie

```yaml
# This setting configures which "profile" dbt uses for this project.
profile: business


# These configurations specify where dbt should look for different types of files.
# The `model-paths` config, for example, states that models in this project can be
# found in the "models/" directory. You probably won't need to change these!
model-paths: ["models"]
seed-paths: ["seeds"]
test-paths: ["tests"]
analysis-paths: ["analysis"]
macro-paths: ["macros"]

target-path: "target"  # directory which will store compiled SQL files
clean-targets:         # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"
  - "logs"


require-dbt-version: [">=1.0.0", "<2.0.0"]

# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models

# In this example config, we tell dbt to build all models in the example/ directory
# as tables. These settings can be overridden in the individual model files
# using the `{{ config(...) }}` macro.
models:
  business:
    +materialized: table

dispatch:
  - macro_namespace: dbt_utils
    search_order: ['trino_utils', 'dbt_utils']
```

Figura 5.2: File YAML del DBT Project

sorgenti all'interno di Trino. In questo caso vengono creati una serie di modelli che prendono dati dalla sorgente iceberg e li spostano in business, o altri modelli che prendono da business per reimmettere in business. All'interno dei modelli per fare riferimento alle sorgenti definite precedentemente viene utilizzato il linguaggio Jinja con le sue doppie parentesi graffe. Naturalmente sarebbe possibile fare una query SQL come se fossimo su Trino, ma è meglio sfruttare le comodità offerte da DBT. Inoltre, visto che i modelli integrano Jinja, è possibile aggiungere delle operazioni preliminari che si possono fare prima di eseguire la query. Inizialmente avevo provato ad inserire i dati all'interno di Iceberg tramite DBT, sfortunatamente sono ancora presenti dei bug, quindi avevo provato ad ovviare al problema inserendo proprio una parte di codice in Jinja che eliminasse i dati all'interno di una tabella in Iceberg prima di reinserirli. Non essendo una soluzione ottimale ho deciso poi di inserire i dati all'interno di Iceberg direttamente tramite Airflow. Configurate le varie componenti del progetto basterebbe eseguire in una shell il comando dbt run.

```yaml
version: 2

sources:
  - name: business
    catalog: business
    schema: public
    tables:
      - name: dim_product
      - name: employees
      - name: fact_pre_discount
      - name: fact_sales_monthly
  - name: datalake
    catalog: datalake
    schema: "\"sep-demo-db\""
    tables:
      - name: dim_customer
      - name: fact_gross_price
      - name: factory_manufacturing_cost
      - name: primes
      - name: customers
      - name: orders
      - name: sessions
      - name: payments
  - name: iceberg
    catalog: iceberg
    schema: "\"sep-demo-db\""
    tables:
      - name: customers_ice
```

Figura 5.3: File delle sorgenti in DBT

```
SELECT * FROM {{ source('iceberg', 'customers_ice') }}
```

Figura 5.4: Un dbt model

Ma l'intenzione era quella di rendere possibile l'esecuzione automatizzata e programmata. Per questo motivo nasce la necessità di utilizzare Airflow, oltre che per la sua integrazione con Trino che permetterebbe di fare ingestion di dati all'interno del datalake su cui è posto Iceberg. Trino viene eseguito in una macchina virtuale EC2, invece Airflow viene eseguito all'interno di un container Docker su cui vengono esposte le porte di rete 8081 per l'interfaccia grafica, generalmente Airflow utilizza la porta 8080, ma ho dovuto fare una modifica al file di configurazione perché è la stessa porta utilizzata da Trino, naturalmente anche Docker è eseguito all'interno della macchina virtuale. Per far interagire Airflow, dbt e trino è stato necessario installare la libreria

dbt-trino sul container Docker in cui viene eseguito Airflow.

4. Airflow, come dicevamo precedentemente, possiede una cartella per i DAG, all'interno di questa cartella verrà posta la cartella contente il dbt project e gli altri file necessari a DBT.

5. All'interno della DAG folder ho creato un DAG composto da più task:

   (a) Il primo task crea una tabella in iceberg tramite un operatore Trino. Utilizzando una query SQL è possibile creare una tabella che abbia le colonne della tabella che vorremmo importare in Iceberg.

   (b) Il secondo task anch'esso un operatore Trino inserisce dei dati tramite il catalog tpch, connesso a Trino. Questo catalog è un algoritmo che permette di generare dati randomici tramite un algoritmo euristico, è possibile decidere il tipo di dati che si vogliono inserire e la dimensione delle tabelle che si vogliono generare. In questo caso è stata generata una tabella di circa 10MB.

   (c) Il terzo task si occupa di utilizzare un comando della shell che aggiorna il manifest del dbt project, documento che contiene tutte le informazioni riguardanti il progetto che dovrebbe essere eseguito dal comando dbt run. Lo si potrebbe immaginare come un file prodotto da un compilatore dove sono stati fatti i vari bindings con le librerie.

   (d) Il quarto task prendendo le informazioni dal manifest genera dinamicamente dei task che seguono l'esecuzione del modello ed il test dello stesso (non sono stati scritti test nel mio caso ma se ce ne fossero verrebbero eseguiti alla fine dell'esecuzione del modello). Ogni sequenza di task modello-test viene eseguita parallelamente all'interno di questo task group.

   La peculiarità di questo DAG è che si aggiorna in maniera dinamica in base ai modelli che andremo ad inserire o eliminare dal nostro progetto dbt. In questo modo i dati possono essere estratti e trasformati partendo da diverse fonti e i modelli possono essere programmati per essere eseguiti con una cadenza specifica. È possibile naturalmente creare dbt project differenti in base alla necessità. Nelle figure 5.5 e 5.6 è possibile vedere come sono scritti i DAG in Python e come sono visibili nell'interfaccia.

5.3.2 Migrazione dei dati

E se un giorno si decidesse di spostare i dati che stiamo utilizzando da un Cloud provider ad un altro? O se volessimo spostare i dati contenuti nel nostro database PostgreSQL in un datalake? Questa piattaforma è pensata anche per questo,

```python
@task_group(group_id='etl',dag=dag)
def load_compli():
    data = load_manifest()

    dbt_tasks = {}
    for node in data["nodes"].keys():
        if node.split(".")[0] == "model":
            node_test = node.replace("model", "test")

            dbt_tasks[node] = make_dbt_task(node, "run")
            dbt_tasks[node_test] = make_dbt_task(node, "test")

    for node in data["nodes"].keys():
        if node.split(".")[0] == "model":
            # Set dependency to run tests on a model after model runs finishes
            node_test = node.replace("model", "test")
            dbt_tasks[node] >> dbt_tasks[node_test]



            # Set all model -> model dependencies
            for upstream_node in data["nodes"][node]["depends_on"]["nodes"]:
                upstream_node_type = upstream_node.split(".")[0]
                if upstream_node_type == "model":
                    dbt_tasks[upstream_node] >> dbt_tasks[node]
next=load_compli()


create_iceberg_table = TrinoOperator(
    task_id = 'create_iceberg_table',
    sql = """CREATE TABLE IF NOT EXISTS iceberg."sep-demo-db".customers_ice WITH (format = 'ORC', location = 's3://datareply-sep-demo-data/customers_ice') AS SELECT * FROM tpch.tiny.customer""",
    handler = list,
    dag=dag
)
read_data = TrinoOperator(
    task_id = 'read_data',
    sql = """INSERT INTO iceberg."sep-demo-db".customers_ice SELECT * FROM tpch.tiny.customer""",
    handler = list,
    dag=dag
)
create_iceberg_table >> read_data >> manifestUpdate>> next
```

Figura 5.5: Alcuni dei task all'interno del DAG

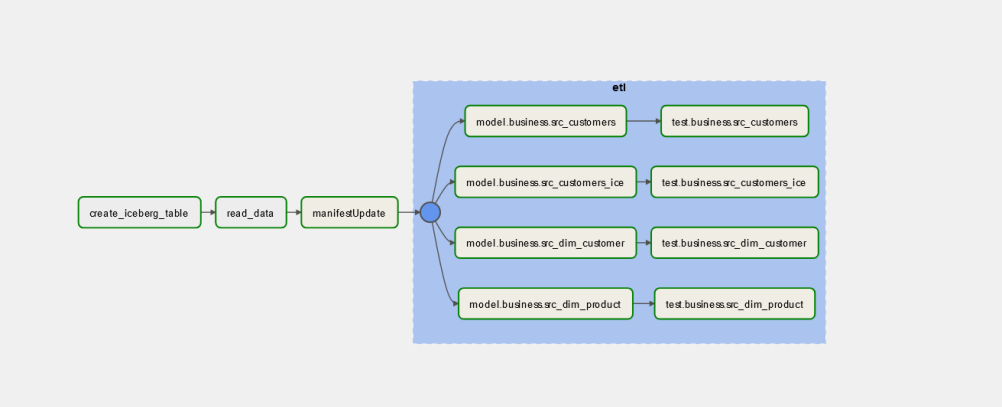

Figura 5.6: Visuale del DAG nell'interfaccia utente di Apache Airflow

infatti, proprio grazie a Trino e DBT, spostare i dati sarà semplicissimo. Basterà aggiungere la destinazione ai catalog di Trino e scrivere un progetto in DBT che abbia come sorgente il database che vogliamo spostare e come destinazione il database di destinazione. Infine, scrivere dei modelli per questo progetto DBT dove verranno selezionati le tabelle sorgente con una " SELECT * " sulla tabella che vogliamo importare e avviare la pipeline su Airflow, da quel momento i dati cominceranno ad essere spostati da una parte all'altra.

5.3.3 Visualizzazione dei dati facile e veloce

Generalmente ai tool di visualizzazione di dati vengono collegati singolarmente i vari database provenienti da sorgenti differenti. Questa pratica, oltre a risultare a volte complicata per utenti che non hanno delle competenze tecniche ha anche lo svantaggio di non permettere di fare query trasversali tra le varie sorgenti, a mala pena è possibile fare query tra database differenti all'interno della stessa sorgente. Qui vengono fuori le potenzialità Trino unite a Metabase. È possibile infatti collegare Trino come singola sorgente a Metabase, a questo punto sarà possibile visualizzare tutti i database contenuti all'interno di trino con annessi schemi e tabelle. È possibile interrogare le tabelle all'interno dei singoli database o fare join tra database presenti su sorgenti differenti(come in Figura 5.7). Il tutto con una semplicissima query SQL, quasi come se fossero classiche tabelle. Oltre a questo è possibile sfruttare le capacità offerte di base da Metabase per fare analisi sui dati, creare Dashboard da condividere e utilizzare ampiamente la piattaforma come più lo si desidera. Nelle immagini seguenti vengono mostrate alcune delle visualizzazioni permesse da Metabase. I dati visibili nella Figura 5.8 sono presi da un dataset scaricato da Kaggle che avevo usato inizialmente per testare la piattaforma. Inoltre, Metabase offre la possibilità di creare degli 'snippets', ovvero 'pezzi' di codice richiamabili con una semplice chiave per facilitare la scrittura delle query, com'è possibile vedere nella figura 5.9.

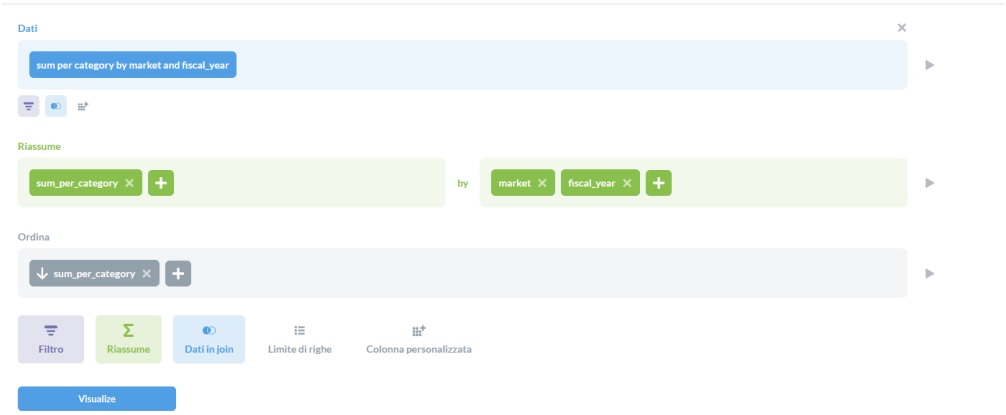

Figura 5.7: Strumento di Metabase per la realizzazione di query semplificate

Infine, le figure 5.10 e 5.11 mostrano rispettivamente i dati all'interno di iceberg e i dati all'interno di Postgres in seguito all'esecuzione del flusso che è stato spiegato precedentemente.

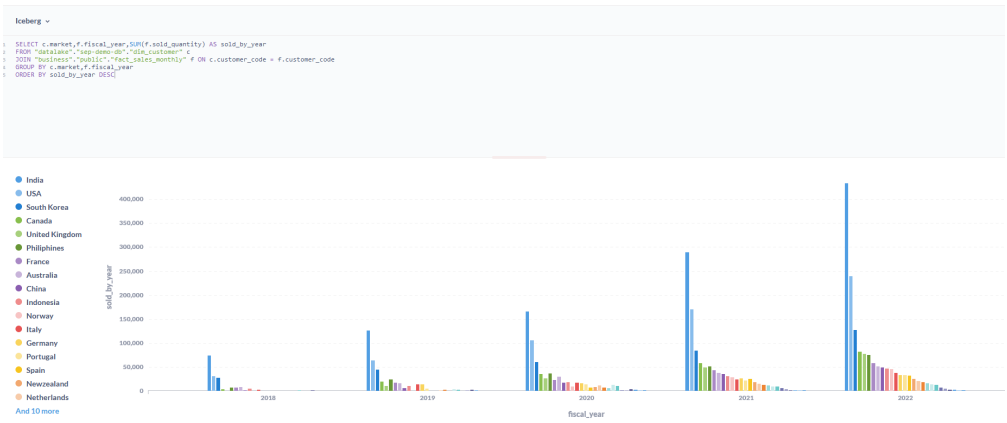

Figura 5.8: Query e visualizzazione dei dati tramite istogramma

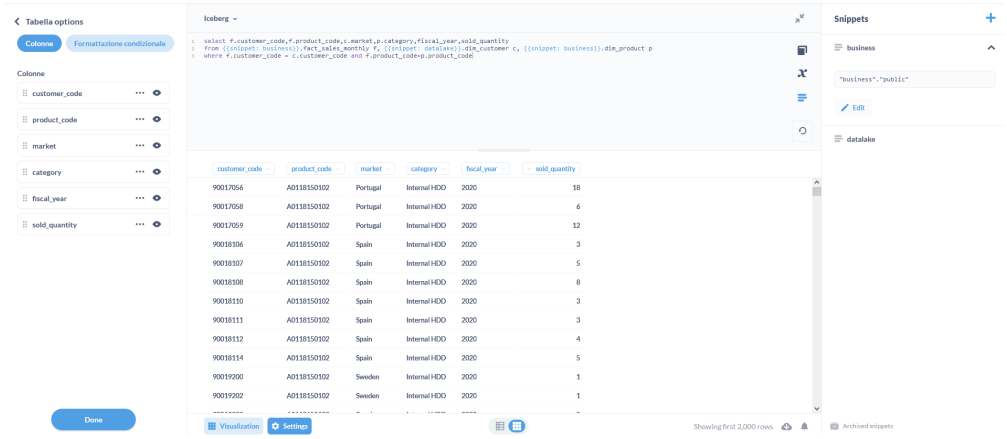

Figura 5.9: Utilizzo degli 'Snippets' per semplificare la scrittura delle query

Figura 5.10: Tabella all'interno di iceberg prima dell'ingestion

Figura 5.11: Tabella all'interno di Postgres in seguito all'ingestion

Capitolo 6

Conclusione

Durante questi capitoli ho parlato dello stato attuale e delle potenzialità di questa piattaforma, naturalmente questo progetto è un POC, con evidenti limitazioni dovute anche all'ambiente in cui è realizzato. Se questo progetto dovesse essere spostato in produzione dovrebbe essere realizzato sicuramente in un cluster Kubernetes fornito da un Cloud Provider per poter gestire le varie risorse, gestire la connessione delle risorse per farle comunicare e gestire le connessioni con i vari servizi a Trino. Messa da parte quindi la parte infrastrutturale da gestire è necessario considerare che la POC è stata realizzata senza includere l'ingestione di stream di dati in real time ma la scelta di Trino è nata proprio per le sue capacità di adattamento, per questo motivo ho valutato un possibile ampliamento della piattaforma integrando nella piattaforma l'utilizzo di **Apache Kafka**, un framework open source pensato per fare ingestion di stream di dati in real time, naturalmente Trino dispone di un connettore anche per quello. Altre considerazioni fatte per migliorare la piattaforma sono relative all'integrazione di altri strumenti che possano migliorare la piattaforma, che andrò adesso ad elencare:

- **Alluxio**: strumento open source nato con l'idea di realizzare un file-system distribuito virtuale che possa fare da caching layer per i dati che vengono letti più spesso. Per quanto Trino possa sfruttare già delle meccaniche di caching, Alluxio è pensato proprio per questo. Anche lui funziona con un meccanismo basato su un 'master' e degli 'slave', un po' come Trino con coordinator e worker. In questo caso però è possibile distribuire un nodo Alluxio a tutte le risorse che hanno necessità di fare accesso diretto ai dati risparmiando tempo e denaro durante la richiesta dei dati alla sorgente. Questo perché i singoli nodi di Alluxio sono collegati al Master che a sua volta è collegato alla sorgente. Ad ogni lettura e scrittura i dati verranno scritti tramite il nodo, la risorsa penserà di parlare con la sorgente ma in realtà sarà Alluxio a rispondere, a sua volta i nodi di Alluxio verranno mantenuti sincronizzati

con le sorgenti in caso di variazione da o verso la sorgente. Nella figura 6.1 è possibile vedere dove sarebbe posizionato Alluxio all'interno del progetto. Sicuramente l'utilizzo di Alluxio potrebbe risparmiare non solo delle letture dalle sorgenti ma potrebbe fornire anche una risposta più veloce migliorando la qualità della piattaforma[43].

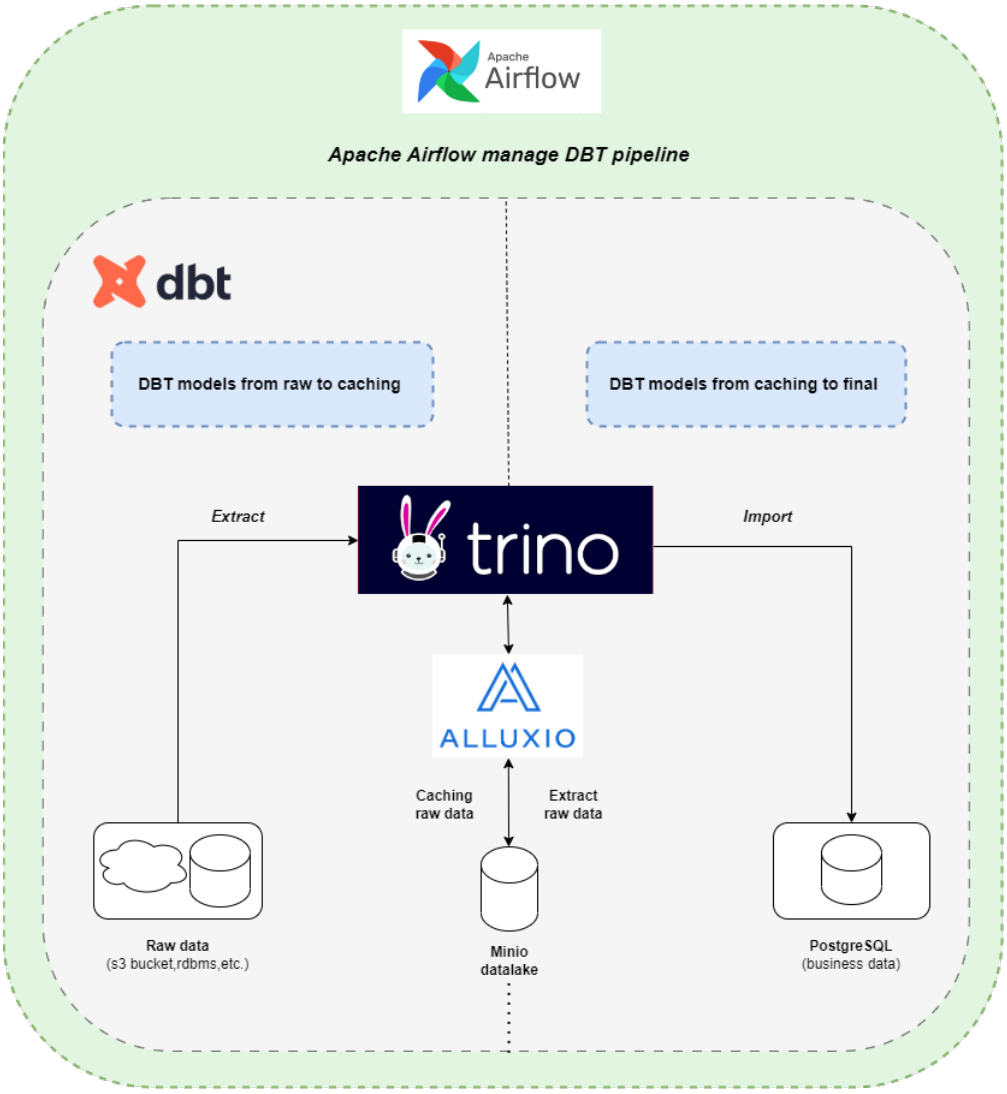

Figura 6.1: Possibile progetto che sfrutta Alluxio come livello intermedio

- **CubeJS**: un altro strumento tenuto in considerazione per ampliare l'offerta della piattaforma. CubeJS supporta l'integrazione sia con Trino che con Metabase, è pensato per la creazione di API al fine di analizzare dati in tempo reale. È possibile collegare una o più sorgenti e CubeJS permetterà

di generare query e aggregazioni partendo da esse, facendo anch'esso caching delle query per velocizzare i risultati e in grado di generare le RESTful API per applicazioni frontend che vogliono analizzare i dati. Idealmente potrebbe essere aggiunto come livello intermedio tra Trino e Metabase[44].

- **CI/CD**: dbt è predisposto per essere integrato con ambienti di CI/CD, molto utili per permettere a più persone di lavorare contemporaneamente sui vari dbt project e modelli[45].

Come spiegato precedentemente, sul mercato sono disponibili altre tecnologie come Dremio che mirano agli stessi obbiettivi di questa piattaforma, con uno stato attuale sicuramente ottimizzato rispetto alla piattaforma da me realizzata. In compenso, la piattaforma da me realizzata è sicuramente molto personalizzabile e ampliabile, con strumenti che potrebbero migliorare nel tempo il suo valore. Inoltre, potrebbe permettere ad un'azienda, che abbia intenzione di effettuare una transizione a questa piattaforma, di poter preservare gli investimenti fatti su altre tecnologie. Come ad esempio Apache Airflow, che sempre di più viene utilizzato all'interno delle aziende per gestire i flussi di lavoro. Infine, durante la realizzazione del progetto mi è stata offerta da parte dell'azienda l'opportunità di provare ad utilizzare Starburst Enterprise ed essendo Starburst una "evoluzione" di Trino, all'interno di questa piattaforma si potrebbe sostituire Trino con Starburst mantenendo invariato il funzionamento delle altre tecnologie utilizzate e ottimizzando la piattaforma. Sicuramente due dei nemici principali di questa piattaforma sono la latenza ed il fatto che l'aggiunta di un catalog a Trino comporti il riavvio dello stesso. Non è un grande problema, ma in alcuni casi potrebbe incidere negativamente, soprattutto se si fa ingestion di flussi di dati in real time. Come detto inizialmente questa piattaforma è pensata per evolversi. Non mettendo una pezza ma aggiungendo pezzi che la rendano migliore in ciò che fa. Spero di poterne ampliare in futuro le sue capacità perché credo abbia un grosso potenziale. Soprattutto per piccole aziende che non possono permettersi costi elevati da parte dei vari Cloud Provider.

Bibliografia

[1] Johannes Holvitie, Sherlock A. Licorish, Rodrigo O. Spínola, Sami Hyrynsalmi, Stephen G. MacDonell, Thiago S. Mendes, Jim Buchan e Ville Leppänen. «Technical debt and agile software development practices and processes: An industry practitioner survey». In: *Information and Software Technology* 96 (2018), pp. 141–160. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2017.11.015. URL: https://www.sciencedirect.com/science/article/pii/S0950584917305098 (cit. a p. 1).

[2] Towards Data Science. *Understand and fight technical debts in your data warehouse*. https://towardsdatascience.com/understand-and-fight-technical-debts-in-your-data-warehouse-d455afed770e. 2023 (cit. a p. 1).

[3] Deloitte. *Layered architecture for data platforms*. https://www2.deloitte.com/nl/nl/pages/data-analytics/articles/layered-architecture-for-data-platforms.html. 2022 (cit. alle pp. 4–7).

[4] Splunk. *What is a Data Platform?* https://www.splunk.com/en_us/data-insider/what-is-a-data-platform.html. 2022 (cit. alle pp. 5, 8).

[5] Praveen Kumar e Dr kavita. «Data Warehouse Concept and Its Usage». In: (apr. 2021) (cit. a p. 8).

[6] Mohammad Rifaie, Keivan Kianmehr, Reda Alhajj e Mick J. Ridley. «Data warehouse architecture and design». In: *2008 IEEE International Conference on Information Reuse and Integration*. 2008, pp. 58–63. DOI: 10.1109/IRI.2008.4583005 (cit. a p. 8).

[7] Oracle. *What is a Data Warehouse?* https://www.oracle.com/it/database/what-is-a-data-warehouse/. 2022 (cit. alle pp. 8, 9).

[8] Amazon Web Services. *What is a data lake?* https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/. n.d. (Cit. alle pp. 9–11).

[9] Talend. *What is a Data Lake?* https://www.talend.com/resources/what-is-data-lake/. n.d. (Cit. alle pp. 9–11).

[10] Databricks. *How Data Lakehouses Solve Common Issues with Data Warehouses*. https://www.databricks.com/blog/2021/02/04/how-data-lakehouses-solve-common-issues-with-data-warehouses.html. 2021 (cit. alle pp. 12, 13).

[11] Serokell. *Data Warehouse vs. Data Lake vs. Lakehouse*. https://serokell.io/blog/data-warehouse-vs-lake-vs-lakehouse. 2021 (cit. alle pp. 12–14).

[12] Ahmed Harby e Farhana Zulkernine. «From Data Warehouse to Lakehouse: A Comparative Review». In: dic. 2022. DOI: 10.1109/BigData55660.2022.10020719 (cit. alle pp. 12, 13).

[13] Inês Araújo Machado, Carlos Costa e Maribel Yasmina Santos. «Data mesh: concepts and principles of a paradigm shift in data architectures». In: *Procedia Computer Science* 196 (2022), pp. 263–271 (cit. alle pp. 16, 17).

[14] Zhamak Dehghani. *From Data Monolith to Mesh*. https://martinfowler.com/articles/data-monolith-to-mesh.html. 2021 (cit. alle pp. 16, 17).

[15] Zhamak Dehghani. *Data Mesh Principles and Logical Architecture*. https://martinfowler.com/articles/data-mesh-principles.html. 2021 (cit. alle pp. 16, 17).

[16] N. Ford, R. Parsons e P. Kua. *Building Evolutionary Architectures: Support Constant Change*. O'Reilly Media, 2017. ISBN: 9781491986691. URL: https://books.google.it/books?id=pYI2DwAAQBAJ (cit. a p. 17).

[17] Gartner. *Data Fabric*. https://www.gartner.com/en/information-technology/glossary/data-fabric. 2022 (cit. a p. 18).

[18] Ana-Maria Ghiran e Robert Andrei Buchmann. «The model-driven enterprise data fabric: A proposal based on conceptual modelling and knowledge graphs». In: *International conference on knowledge science, engineering and management*. Springer. 2019, pp. 572–583 (cit. a p. 19).

[19] Alexander Bogdanov, Mikhail Kuznetsov e Natalia Kuznetsova. «Big Data Virtualization: Why and How?» In: *International Journal of Computer Science and Network Security* 20.3 (2020), pp. 1–7 (cit. alle pp. 19, 20).

[20] Atlan. *Data Fabric vs Data Virtualization: What's the Difference?* https://atlan.com/data-fabric-vs-data-virtualization/. 2021 (cit. alle pp. 19, 20).

[21] Denodo. *Denodo Platform 8.0 Documentation*. https://community.denodo.com/docs/html/browse/8.0/en/. 2023 (cit. a p. 21).

[22] Dremio. *Intro Guides*. https://www.dremio.com/resources/intro/. 2022 (cit. a p. 21).

[23] Starburst. *Starburst Architecture*. https://docs.starburst.io/get-started/architecture.html. 2023 (cit. a p. 21).

[24] Trino. *Use Cases — Trino 421 Documentation*. https://trino.io/docs/current/overview/use-cases.html. 2022 (cit. a p. 22).

[25] Trino. *Trino concepts — Trino 421 Documentation*. https://trino.io/docs/current/overview/concepts.html. 2022 (cit. alle pp. 22, 24).

[26] *Introduction to dbt*. https://docs.getdbt.com/docs/introduction (cit. a p. 25).

[27] *Projects*. https://docs.getdbt.com/docs/building-a-dbt-project/projects (cit. a p. 25).

[28] *Materializations*. https://docs.getdbt.com/docs/building-a-dbt-project/materializations (cit. a p. 25).

[29] *Apache Airflow*. https://airflow.apache.org/docs/apache-airflow/stable/index.html (cit. alle pp. 26, 36).

[30] *Overview*. https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html (cit. alle pp. 26, 28).

[31] *Getting Started*. https://iceberg.apache.org/docs/latest/getting-started/ (cit. a p. 28).

[32] *Iceberg's Hidden Partitioning*. https://iceberg.apache.org/docs/latest/partitioning/#icebergs-hidden-partitioning (cit. a p. 28).

[33] *Time Travel*. https://iceberg.apache.org/docs/latest/spark-queries/#time-travel (cit. a p. 28).

[34] *Documentation*. https://www.metabase.com/docs/latest/ (cit. a p. 29).

[35] *About PostgreSQL*. https://www.postgresql.org/about/ (cit. a p. 29).

[36] *Concepts*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (cit. a p. 30).

[37] *Amazon S3 Developer Guide*. https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html (cit. a p. 31).

[38] *AWS Identity and Access Management User Guide*. https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html (cit. a p. 31).

[39] *AWS Glue Developer Guide*. https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html (cit. a p. 32).

[40] Y. Zhang e J. Liang. «An Introduction to Docker and Analysis of its Performance». In: *International Journal of Computer Science and Network Security* 17.3 (2017), pp. 1–7 (cit. a p. 33).

[41] *Getting Started*. `https://docs.dagster.io/getting-started` (cit. a p. 36).

[42] Databricks. *Medallion Architecture*. `https://www.databricks.com/glossary/medallion-architecture`. 2023 (cit. a p. 38).

[43] Alluxio. *Alluxio Documentation*. `https://docs.alluxio.io/os/user/stable/en/Overview.html`. 2023 (cit. a p. 50).

[44] Cube.js. *Cube.js Documentation*. `https://cube.dev/docs/`. 2023 (cit. a p. 51).

[45] dbt. *Customizing CI/CD*. `https://docs.getdbt.com/guides/orchestration/custom-cicd-pipelines/1-cicd-background`. 2023 (cit. a p. 51).