



**Politecnico  
di Torino**

**POLITECNICO DI TORINO**

Master's degree in Computer engineering

Academic Year 2022/2023

**Design and implementation of data  
science pipelines**

A new paradigm based on analytics engineers

Supervisor

Prof. Paolo GARZA

Candidate

Ferdinando MICCO

July 2023



# Abstract

Data represents an increasingly critical strategic asset for companies of all sectors and sizes. Without a solid foundation of Analytics engineering, one risks having poor quality data, manual and fragmented processes, unreliable analysis, and long delivery times.

Fortunately, there are tools that help implement the best Analytics engineering practices efficiently and at scale. One of these is dbt (data build tool), an open-source platform that simplifies the transformation, documentation, and testing of data models.

The main focus of the thesis is to implement a modern pipeline solution that incorporates all best practice of analytics engineering. The inclusion of an analytics engineer within a data team represents a new paradigm in data-driven organizations. The study aims to show the feasibility of such a solution and the potential improvements of adopting such a solution in terms of increased efficiency, higher quality data, and faster time to insights.

Moreover, this project has served as the starting point for a collaboration with a company that has specific requirements in the area of data quality. The collaboration has provided valuable insights into the practical implementation of the pipeline solution and has helped tailor the approach to address the company's data quality needs.

The proposed solution will involve the use of cutting-edge tools and techniques to transform, document, and test data models, such as dbt. The whole architecture will be implemented serverless on a cloud computing system to provide the required elasticity, scalability, and cost-effectiveness.

The improved reliability of data analysis, coupled with the faster time to insights, will allow organizations to make data-driven decisions more quickly and confidently.

## ACKNOWLEDGMENTS

I would like to offer my special and sincere thanks to my colleague Clemente Cetera for his presence, his assistance and his advice during every stage of this project. I would also like to thank my manager Andrea D'Amelio and prof. Paolo Garza for their guidance during my internship and for giving me the opportunity to work on this thesis.

# Table of Contents

<b>1</b>	<b>Introduction</b>	6
1.1	Motivation . . . . .	6
1.2	Problem . . . . .	6
1.3	Purpose and goal . . . . .	7
1.4	Methodology . . . . .	7
1.5	Outline . . . . .	7
<b>2</b>	<b>State of the Art</b>	8
2.1	Data pipeline . . . . .	8
2.1.1	Extract, Transform and Load . . . . .	8
2.1.2	Extract, Load and Transform . . . . .	9
2.1.3	Comparison and Cloud computing scenario . . . . .	10
2.2	Importance of Cloud Computing Services . . . . .	11
2.2.1	Advantages and benefits offered by Cloud computing services	11
2.3	Technologies Exploited in Cloud Computing Services . . . . .	12
2.3.1	Virtualization and resource pooling . . . . .	12
2.3.2	Distributed computing . . . . .	12
2.3.3	Networking and connectivity . . . . .	12
2.4	Analytics engineering . . . . .	13
2.4.1	What is Analytics engineering . . . . .	13
2.4.2	The analytics engineer . . . . .	13
2.4.3	dbt : Data Build Tool . . . . .	14
2.4.4	dbt models . . . . .	14
<b>3</b>	<b>dbt Project</b>	17
3.1	Introduction . . . . .	17
3.2	Data source . . . . .	17
3.3	dbt project Set-up . . . . .	18
3.4	Model creation . . . . .	20
3.4.1	Staging layer . . . . .	20
3.4.2	Intermediate layer . . . . .	22

3.4.3	Marts layer . . . . .	23
3.5	Testing and Validation . . . . .	26
3.6	Data Lineage and Documentation . . . . .	28
<b>4</b>	<b>AWS data pipeline with dbt serverless</b>	<b>30</b>
4.1	Technologies . . . . .	30
4.1.1	S3 - Simple Storage Service . . . . .	30
4.1.2	Redshift . . . . .	30
4.1.3	Glue Data Catalogue . . . . .	31
4.1.4	Docker . . . . .	32
4.1.5	AWS ECR . . . . .	32
4.1.6	AWS Fargate . . . . .	33
4.1.7	ECS . . . . .	33
4.1.8	Lambda . . . . .	34
4.1.9	CloudWatch . . . . .	34
4.2	Overall Architecture . . . . .	35
4.3	Data Ingestion Phase . . . . .	36
4.4	Data Storage and Management . . . . .	36
4.5	Data Transformation with dbt . . . . .	37
4.6	Data Processing and Analysis . . . . .	39
4.7	Security and Data Governance . . . . .	43
<b>5</b>	<b>Speculative Analysis : comparison between proposed and traditional paradigm</b>	<b>45</b>
5.1	Process Analysis: Model creation for business insights . . . . .	46
5.2	Design comparison and final thoughts . . . . .	49
<b>6</b>	<b>Conclusion</b>	<b>51</b>
	<b>Bibliography</b>	<b>54</b>
	<b>List of Figures</b>	<b>57</b>
	<b>Acronyms</b>	<b>58</b>



# Chapter 1

## Introduction

### 1.1 Motivation

In today's business landscape, Big Data refers to the vast amount of structured and unstructured data that inundates organizations on a daily basis. The ability to store, process, and analyze such data is crucial for extracting insights and making informed decisions. Companies generate and collect large volumes of data from various sources, including user interactions, enterprise resource planning systems, and other software applications. The goal is to leverage this data to gain business insights and derive value from it. However, without reliable and relevant datasets, business intelligence insights may be incomplete, inaccurate, or biased, leading to misguided strategic choices.

### 1.2 Problem

In the modern paradigm of data driven organizations, data engineers focus on building and managing all the infrastructure of data pipelines, while data analysts utilize advanced statistical and analytical techniques to derive insights from data. However, There is a misalignment between the technical expertise of Data engineer and the analytical skills of Data analyst may lead to difficulties in finding proper coordination and collaboration.

To address this challenge, a new role is needed to bridge the gap. Analytics engineers serve as a connection between technology and business, with the goal of providing useful, clean, and accurate datasets for business needs.

## 1.3 Purpose and goal

dbt (Data Build Tool) is an open-source software developed by Fishtown Analytics, specifically designed to equip analytics engineers with robust tools for dataset creation and validation.

The purpose of this thesis is to design and implement an architecture that embodies the new paradigm, wherein analytics engineers play a central role in the transformation phase.

The goal of this thesis is to build a comprehensive dbt project from scratch, starting with a database, in order to showcase the key characteristics that facilitate the transition from the old paradigm to the new one. Moreover, a proof-of-concept solution has been developed to seamlessly integrate the dbt project into a serverless data pipeline hosted on AWS.

The deliverables include the thesis, the code and the documentation to explain the code.

## 1.4 Methodology

The methodology of this project can be considered as system research on dbt and AWS. The first step would be to figure out the components related to this project, and the interactions between them. The next step is to build a concept about how to realize the goals. The concept should be demonstrated whether it is feasible or not. In this case, it is to generate a design about how to run a serverless DBT project on AWS. Then, the design should be implemented to check its feasibility. According to the situation, minor changes could be applied to adjust the design in order to achieve a better result.

## 1.5 Outline

Chapter 2 introduces the necessary background for the project, including the introduction of applied technologies. Chapter 3 and 4 explains the implementation of the work. Chapter 5 contains a qualitative analysis to evaluate the effectiveness of the new paradigm compared to the old one. Chapter 6 gives the conclusion of the project.

# Chapter 2

## State of the Art

### 2.1 Data pipeline

Data pipelines serve as the backbone of modern data-driven environments. In this context processing, analysis, and decision-making is enabled by the efficient and reliable flow of data from various sources to target systems. A data pipeline encompasses a series of stages or steps that collectively handle the movement, transformation, and storage of data.

There are two main paradigms for implementing data pipelines: Extract, Transform, Load (ETL) and Extract, Load, Transform (ELT). [1]

#### 2.1.1 Extract Transform and Load

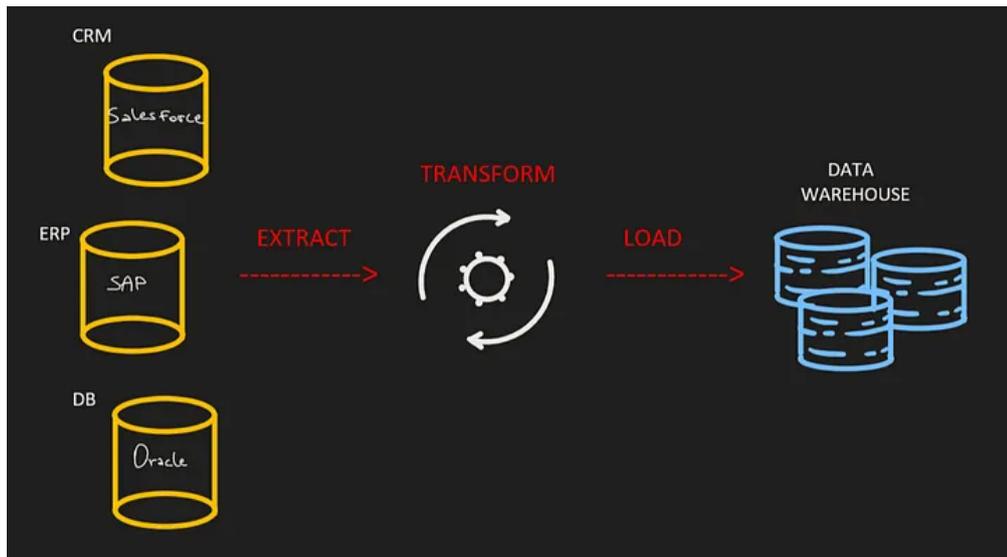
The ETL paradigm is a traditional approach to data pipeline implementation. It involves the following three stages:

In the "Extract" stage, data is extracted from various sources such as databases, files, APIs, or streaming platforms. This is why this stage can be considered as the collection of data from inputs.

Once the data is extracted, it undergoes a series of transformations. These include validation, standardization, data cleaning, and aggregation. The goal is to grant that the data is in the desired format and structure for analysis and loading.

In the last stage, the transformed data is loaded into a target system, such as a data warehouse, data mart, or a specific analytics platform. In the loading process the transformed data is mapped to the target schema and inserted or updated.

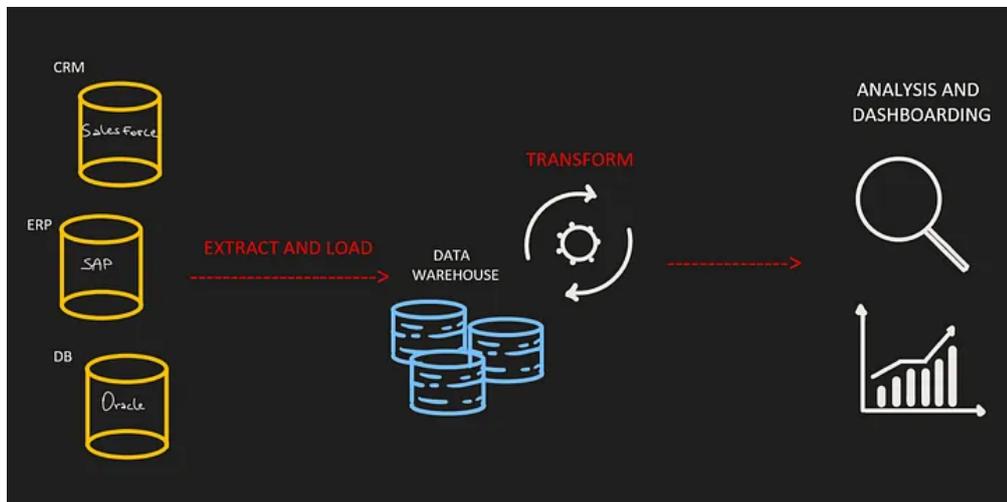
The ETL paradigm is often used in scenarios where data integration, data quality management, and complex transformations are necessary before loading the data into the target system. It allows for data consolidation, harmonization, and enrichment, enabling organizations to derive valuable insights from disparate data sources. [2]



**Figure 2.1:** Logic scheme of an etl data pipeline

### 2.1.2 Extract, Load and Transform

The ELT paradigm is an alternative approach to data pipeline implementation that has gained popularity with the emergence of cloud-based data platforms and powerful data processing technologies. It involves the following stages:



**Figure 2.2:** Logic scheme of an elt data pipeline

Concerning the ELT, the Extraction stage is identical to the ETL process. Data is collected from different sources, but instead of immediately applying

transformations, it is prepared for the loading phase.

The extracted data is directly loaded into a target data storage or data warehouse. This means that the data arrives in the storage infrastructure without any manipulation. This is commonly referred to as "raw data."

Once the data is loaded into the target system, transformations are performed within the target environment. This approach takes advantage of the flexibility and computational resources available in the Cloud system. During this stage, we can begin by transforming the raw data, initially focusing on cleaning any data imperfections, and then proceed to create models based on relevant data aggregation.

The ELT paradigm offers the advantage of leveraging the full processing power of the target system for data transformations. It enables organizations to handle large volumes of data and perform complex analytical tasks directly within the target environment, reducing the need for pre-processing and enabling faster insights.[3]

### 2.1.3 Comparison and Cloud computing scenario

The choice between ETL and ELT depends on several factors, including data requirements, target system capabilities, performance considerations, and data governance needs. Here are some points of comparison between the two paradigms:

Complexity of transformations: ETL is typically suited for scenarios that require extensive data transformations before loading into the target system. ELT, on the other hand, is advantageous when the target system has robust processing capabilities and can handle complex transformations efficiently.

Scalability and performance: ELT leverages the scalability and performance capabilities of cloud-based data platforms, allowing for high-volume data processing and real-time analytics. ETL may require additional infrastructure and processing resources to handle large-scale transformations.

Data governance and privacy: ETL provides more control over data governance processes, as transformations can be performed before loading the data into the target system. ELT, however, requires careful consideration of privacy and security

In conclusion ETL is the typical solution adopted for small relational database which require complex transformation that have been predetermined as being relevant to the analysis goal. This is why Data teams using this approach have to know in advance how the data will be used before any analysis is even performed. The problem is that they have poor visibility on transformations applied to data and their life cycle since engineering teams typically own the Extraction and Transformation steps. This makes it hard for them to understand exactly what the data represents, often leading to incorrectly drawn conclusions.

Instead in ELT approach the amount of raw data stored into the warehouse will increase and so the amount of storage needed. For the same reason the

computational power needed to transform and model data will be bigger. Such resources are very expensive and this has been a limiting factor for many years until the arrival of Cloud computing.

Cloud computing reduces the costs of building and maintaining Data Center. It allows to dynamically scale the amount of resources needed both for storage and computation (more space when needed and less space when not to avoid waste of resources and costs) and offers super computer to quickly transform great amount of data. The major cons of ELT pipelines are now compensated by cloud computing services leading them to become commonplace. [4]

In this ELT scenario the Transformation phase takes places as the last step of the process so it's unbound from the Extraction and the Load. Such an isolated process, within the Datawarehouse environment, makes the operation executable with more flexibility and frequency without loss of performances. These pave the way to a new figure designated to accomplish these functions, the "Analytics engineer".

## 2.2 Importance of Cloud Computing Services

Cloud computing services have emerged as a revolutionary paradigm in the field of information technology. The rapid advancement of digital technologies and the increasing demand for scalable, flexible, and cost-effective computing solutions have led to the widespread adoption of Cloud computing services across various industries and sectors. This section aims to provide an introduction to Cloud computing services, their fundamental concepts, and their impact on the modern technology landscape.[5]

### 2.2.1 Advantages and benefits offered by Cloud computing services

What cloud computing systems offer is the possibility to access hardware resources without physically owning them. In fact, this allows companies to avoid buying and managing resources. There are several resources available for different purposes. The main ones are computational units, storage systems, and networking capacities. Computational units are CPUs and GPUs used to execute various kinds of operations, such as running a query. The storage system refers to all the available space for data collection, and networking capacity refers to the ability to handle a certain amount of traffic and requests on a resource. The main advantages of cloud computing are the great flexibility that these systems offer. The amount of resources can auto-scale according to business necessities, ensuring that: - The need for resources can always be met. - Unused resources are instantly freed, avoiding any waste or unnecessary expenses. [6]

## 2.3 Technologies Exploited in Cloud Computing Services

Cloud computing services rely on a variety of underlying technologies to deliver their capabilities. These technologies work together to provide scalable, reliable, and efficient computing solutions. Understanding these foundational technologies is essential to comprehending the inner workings of Cloud computing services.

### 2.3.1 Virtualization and resource pooling

Two fundamental technologies employed by cloud computing system are Virtualization and resource pooling. The first concern the possibility to abstract hardware, operating systems, storage, and network resources into virtual instances. They allow an efficient management of the physical resources and ensure isolation and flexibility. Resource pooling instead is the management mechanism to allocate hardware resources. Resource pooling involves aggregating resources, such as processing power, storage, and network bandwidth, into a shared infrastructure. This shared infrastructure is dynamically assigned to users as needed. Resource management technologies, including orchestration and auto-scaling systems, ensure optimal resource utilization and efficient allocation based on demand.[7]

### 2.3.2 Distributed computing

Distributed computing is a core concept in Cloud computing services. It involves the use of multiple interconnected computers or servers to work together as a unified system. Distributed computing enables the processing of tasks across a network of resources, enabling parallel processing, fault tolerance, and improved performance. Distributed file systems, distributed databases, and distributed processing frameworks, such as Apache Hadoop and Apache Spark, are key components of Cloud computing services.[8]

### 2.3.3 Networking and connectivity

Robust and high-performance networks are essential for efficient communication between different components of the cloud infrastructure. In order to adapt to varying traffic volumes, a network must be capable of swiftly managing network configurations. Ad-hoc technologies such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV) provide the agility and programmability required to manage and configure network resources in cloud environments. Connectivity options, including virtual private networks (VPNs), load balancers, and content delivery networks (CDNs), ensure secure and efficient data transfer

between cloud service providers and their clients. Moreover, technologies such as encryption, authentication, and access control mechanisms are implemented to safeguard data and ensure privacy. Secure communication protocols and secure multi-tenancy models provide isolation between different clients utilizing the cloud infrastructure. To mitigate potential security threats, several techniques exist, including continuous monitoring, vulnerability assessments, and intrusion detection.

## 2.4 Analytics engineering

This thesis delves into the topic of analytics engineering, a term coined by the company 'Fishtown analytics', whose toolkit, dbt, is revolutionizing the field of data analytics. The immense power of this tool has propelled it to become a leading technology, increasingly adopted by numerous companies to bolster their data pipelines.

The dbt toolkit stands out due to its user-friendly nature, seamless integration with various cloud system services, and the ability to adhere to diverse Data Quality best practices. These qualities are highly sought after by data teams seeking to optimize their processes.

As explored in this thesis, the evolving landscape of data teams has prompted a restructuring of their responsibilities, with the transformation phase now falling under the purview of the "Analytics engineer." This role takes charge of managing and executing the various tasks involved in the data transformation process.

### 2.4.1 What is Analytics engineering

Analytics engineering is the term used to identify the discipline of data transformation. It includes all the practices, activities and tools that contribute to the creation of clean datasets. Its aim is to model the content and structure of data in order to empower end-users to answer their own questions and to better respond to the needs of analysis.

### 2.4.2 The analytics engineer

Nowadays, with the massive use of Cloud data warehouses, a new role is needed to connect data engineers and data analysts. Analytics engineers represent a bridge between Technology and business - new data team members with the aim of providing the most useful, clean, and accurate datasets for business.

The Data Engineer is no more in charge of Data transformation, his role is to build and maintain the data pipeline. Data content is fully managed by the Analytics Engineer that provides clean, transformed data ready for analysis. He

applies software engineering best practices to analytics code (ex. version control, testing, continuous integration) and maintains data documentation & definitions. He also interacts with the Data Analyst to supply required models for deep insights work, build critical dashboards and forecasting.[9]

DATA ENGINEERS	ANALYTICS ENGINEERS	DATA ANALYSTS
- Build custom data ingestion Integrations	- Provide clean, transformed data Ready for analysis	- Deep insights work
- Manage overall pipeline orchestration	- Apply Software engineering practices to analytics code (ex. Version control, testing, continuous integration)	- Work with business users to understand data requirements
- Develop and deploy machine learning endpoints	- Maintain data documentation & Definitions	- Build critical dashboards
- Build and maintain the data platform	- Train business users on how to use a data platform data visualization tools	- Forecasting
- Data Warehouse performance optimization		

**Figure 2.3:** Tasks assigned to each of the role within a data Team.

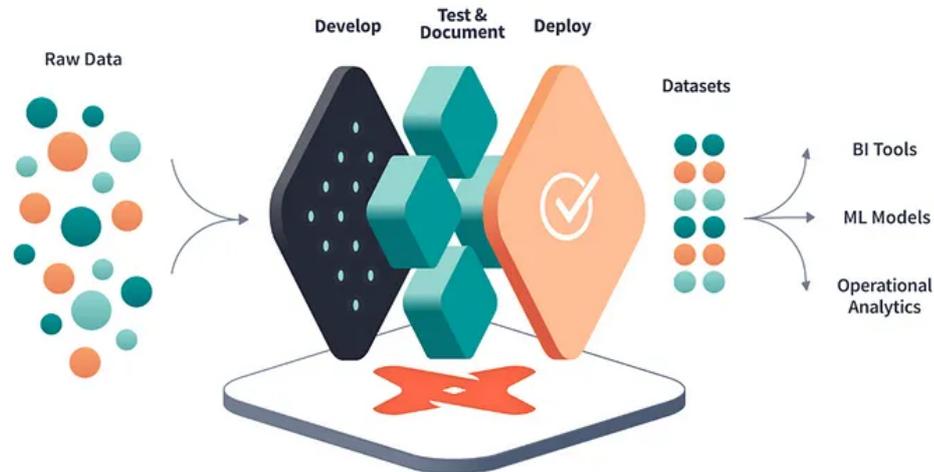
### 2.4.3 dbt : Data Build Tool

In order for the analytics engineer to accomplish his tasks a new tool has been created : dbt (Data Build Tool). Dbt enables analytics engineers to transform data in their warehouses by simply writing select statements. dbt handles turning these select statements into tables and views.” Further, “dbt does the T in ELT (Extract, Load, Transform) processes — it doesn’t extract or load data, but it’s extremely good at transforming data that’s already loaded into your warehouse. -Dbt Labs

At the most basic level, dbt has two building blocks: a compiler and a runner. Users choose whatever text editor to write dbt code and then invoke dbt from the command line. dbt compiles all code into raw SQL and executes that code against the configured data warehouse.[10]

### 2.4.4 dbt models

Data transformation is simplified by dbt models that allow to do all the work in SQL select statement avoiding boilerplate DDL and DML code. Furthermore dbt is a complete Jinja compiler and SQL is far more powerful when it’s paired with a fully-featured templating language.



**Figure 2.4:** The role of dbt inside a data pipeline

dbt promotes a modular and reusable approach to model creation. It allows analysts and data engineers to define reusable SQL-based models, commonly referred to as dbt models. These models encapsulate the logic required to transform and shape data into meaningful analytical assets. By designing models as modular units, dbt enables reusability across different analyses, reducing redundancy and promoting consistency. This modular approach enhances collaboration, as models can be shared and reused by team members, fostering a more efficient and scalable modeling process.

dbt also supports incremental model building, which is a key feature for managing and updating models as new data arrives. With dbt, users can define incremental models that only process new or modified data, allowing for faster model refreshes and reducing unnecessary processing of unchanged data. Incremental model building in dbt is achieved through the use of timestamp-based or state-based mechanisms, providing flexibility and efficiency in data processing.

dbt offers built-in features for data lineage and documentation, which are crucial for understanding and maintaining models. Data lineage in dbt provides visibility into the flow of data, tracking the dependencies between different models and their sources. This enables users to trace the origin of data and understand how changes in upstream data sources impact downstream models. Additionally, dbt facilitates the generation of documentation, automatically creating documentation pages that describe the purpose, structure, and dependencies of each model. This documentation feature promotes better understanding, collaboration, and maintainability of models over time.[1]

In the above example we can see the model that shows, for each city, how many

```

kpi_sales_per_city_category.sql
Raw
1  with sales as (
2
3      select * from {{ ref('stg_tickit_sales') }}
4
5  ),
6
7  events as (
8      select * from {{ ref('dim_events') }}
9  )
10
11  {% set categories = dbt_utils.get_column_values(ref('stg_tickit_categories'), column='cat_name') %}
12
13  select
14      e.venue_city,
15      sum(s.qty_sold) as tickets_sold,
16      sum(s.price_paid) as amount_paid,
17      {% for category in categories %}
18      sum(case when e.cat_name = '{{ category }}' then s.qty_sold end) as {{category}}_tickets_sold
19      {% if not loop.last %},{% endif %}
20      {% endfor %}
21
22  from
23      sales as s
24      join events as e on s.event_id = e.event_id
25  group by
26      e.venue_city
27

```

Jinja syntax is contained within curly brackets

Dbt.utils package function used to retrieve all values contained in a column

Example of iterations and if statement

**Figure 2.5:** select statement example to create a new dbt model

tickets have been sold for each event's category. We can observe how Jinja is useful to write in a simpler way complex query by setting variables, using if else statements or iterations.

Jinja also offers the possibility to create custom and reusable functions. Some of those are available in packages that can be imported as project's dependencies, some can be written by the analytics engineer and some other are dbt built-in functions. Two of those are the 'source()' and 'ref()' functions that allow to build models upon others previously created.[11]

# Chapter 3

## dbt Project

### 3.1 Introduction

The next chapter will describe the created dbt project and all its functionalities.

The chapter will be structure as follow :

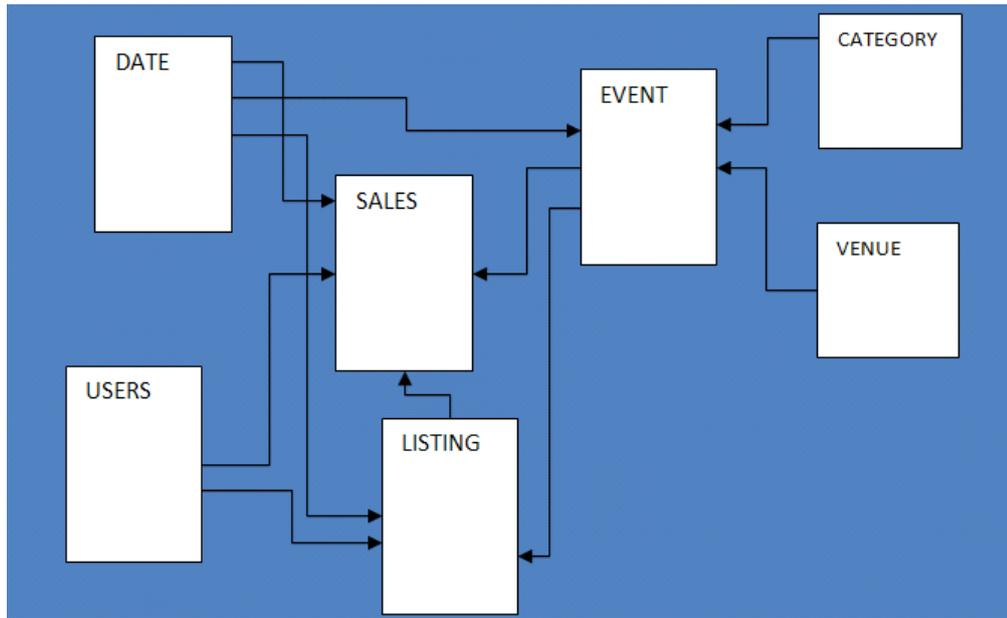
1. Data source description.
2. dbt project set-up.
3. Model creation.
4. Testing and validating
5. Documentation

### 3.2 Data source

In order to provide a better understanding of further data transformations we will first describe the data used for modelling.

The database used in the project consists of a collection of sales activity for the fictional TICKIT web site, where users buy and sell tickets for sporting events, shows, concerts, musicals, and opera . Analysts can use this information to provide incentives to buyers and sellers who frequent the site, to attract new users, and to drive advertising and promotions.

The database is open-source and provided by AWS. It has the following ER schema

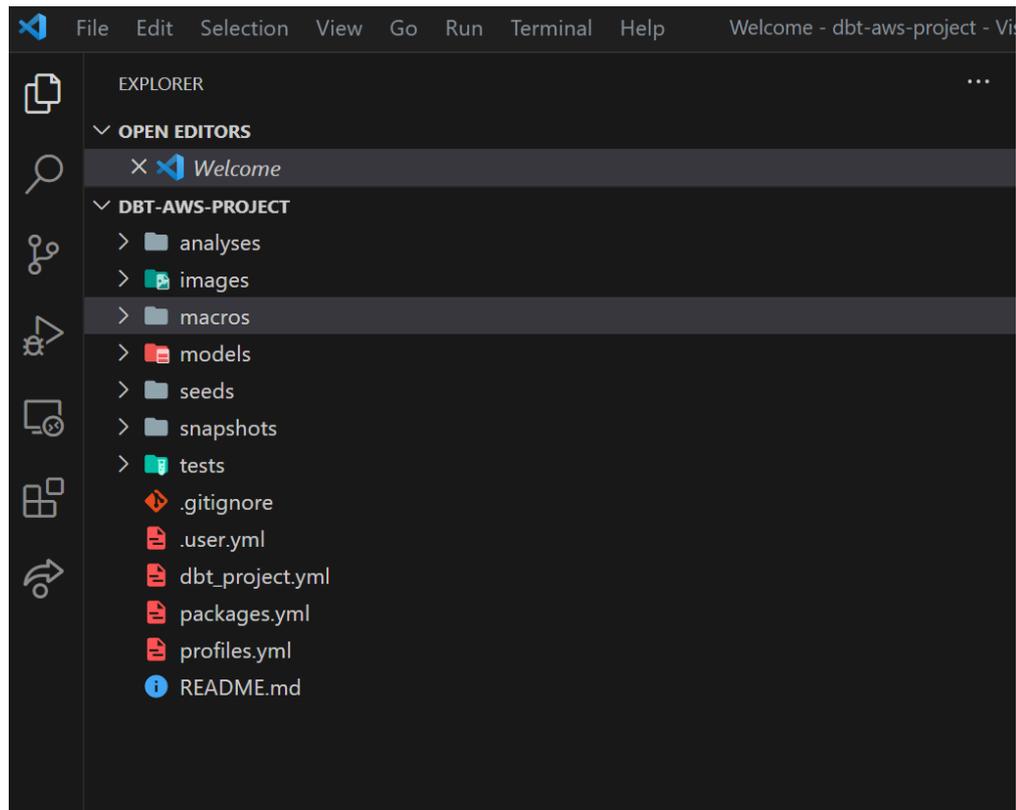


**Figure 3.1:** ER schema for the Tickit relational database

### 3.3 dbt project Set-up

The first think to do when starting a new dbt project is to create a Git-hub repository and open it on a IDE. For this project VSCode has been used.[12]

From now on it is possible to interact with the terminal offered by VSCode to execute all the command provided by DBT CLI for project initialization. This is how a newly initialized dbt project looks like after running the "dbt init" command[13]:



**Figure 3.2:** newly created dbt project

The GitHub project's `packages.yml` contains the packages that are the project's dependencies. They have been installed with the command "dbt deps".

In this case the following packages have been included:

1. package: dbt-labs/dbt-external-tables version: 0.8.2
2. package: dbt-labs/codegen version: 0.9.0
3. package: dbt-labs/dbt-utils version: 1.0.0
4. package: dbt-labs/redshift version: 0.8.0

After that the dbt project have been connected to an existing Datawarehouse containing the raw data of the above mentioned Ticket database. This is why has been created a Redshift datawarehouse instance. To connect dbt to Amazon Redshift, it has been used the Glue adapter. The Glue adapter is a plugin for dbt that allows you to interact with Redshift as a target database for running your dbt models. The Glue adapter leverages AWS Glue DataBrew and AWS Glue DataCatalog to provide seamless integration with Redshift. [14]

The connection between a dbt-project and a datawarehouse is configured in the profile.yml file :

```
yaml Copy code
# dbt_project.yml

...

models:
  my_dbt_project:
    target: dev
    adapter: glue

    schema: my_redshift_schema
    database: my_redshift_database

warehouse:
  host: my_redshift_host
  port: my_redshift_port
  user: my_redshift_user
  password: my_redshift_password
```

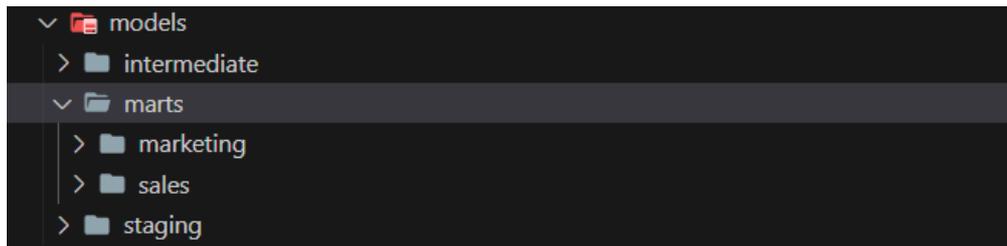
Figure 3.3: profile.yml datasource configuration file

## 3.4 Model creation

Once the datasource has been configured the analytics engineer can start modelling the rawdata. This project follows many of the best practices outlined in dbt Labs' Best Practice Guide. All data models that are going to be created will be organized into three different subdirectories, also known as layers : staging, intermediate, and marts.

### 3.4.1 Staging layer

According to the principles outlined in the dbt best practice guide, the staging layer can be viewed as the consolidation and refinement of raw material into individual building blocks that will later be used to construct more sophisticated and valuable structures.



**Figure 3.4:** models folder structure in the dbt project

In this project, the staging data models play a fundamental role as foundational tables and views, serving as the building blocks for more complex aggregations and analytics queries in the Redshift environment. Specifically, within the `models/staging/ticket/` subdirectory, the `schema.yml` file defines the creation of seven late-binding views in Amazon Redshift. These views are meticulously modeled by dbt, ensuring their accuracy and effectiveness.

Here, we provide an illustration of the `stg-tickit-sales` model (`stg-tickit-sales.sql`) as an example. This model retrieves data from the external sale table in the `external-table` schema, and subsequently performs column renaming and basic calculations. [14]

**Listing 3.1:** `stg-tickit-sales` model (`stg-tickit-sales.sql`)

```

1  {{ config(materialized='view', bind=False) }}
2
3  with source as (
4
5      select * from {{ source('tickit_external', 'sale') }}
6
7  ),
8
9  renamed as (
10
11     select
12         saleid as sale_id,
13         listid as list_id,
14         sellerid as seller_id,
15         buyerid as buyer_id,
16         eventid as event_id,
17         dateid as date_id,
18         qtysold as qty_sold,
19         round(pricepaid / qtysold, 2) as ticket_price,
20         pricepaid as price_paid,
21         round((commission / pricepaid) * 100, 2) as commission_prcnt,
22         commission,
23         (pricepaid - commission) as earnings,
24         saletime as sale_time

```

```

25     from
26         source
27     where
28         sale_id IS NOT NULL
29     order by
30         sale_id
31
32 )
33
34 select * from renamed

```

In order to construct subsequent models based on the staging layer, the execution of the "dbt run" command is necessary. According to dbt Labs, this command enables the execution of compiled SQL model files on the designated target database. By connecting to the target database, dbt runs the relevant SQL queries required to materialize all data models, employing the specified materialization strategies.[15]

### 3.4.2 Intermediatelayer

The intermediate layer, as explained in the dbt best practice guide, is characterized as a collection of purpose-built transformation steps. It is recommended to consider action-oriented verbs (such as pivoted, aggregated-to-user, joined, fanned-out-by-quantity, funnel-created, etc.) when designing the intermediate layer, serving as a guiding principle for its construction.

Within the intermediate layer of this project, there are two models that pertain to users. In the sample TICKIT database, all users are combined into a single table. However, for the purpose of analytics, marketing teams may be interested in different user personas, such as buyers, sellers, sellers who also make purchases, and non-buyers (users who have never bought tickets). The two models within the intermediate layer are designed to filter and create separate views for buyers and sellers, enabling distinct perspectives on user personas.

**Listing 3.2:** int-buyer-extracted-from-user model (int-buyer-extracted-from-user.sql)

```

1  {{ config(materialized='view', bind=False) }}
2
3  with sales as (
4
5      select * from {{ ref('stg_tickit__sales') }}
6
7  ),
8
9  users as (
10
11     select * from {{ ref('stg_tickit__users') }}

```

```
12 ),
13 ),
14
15 first_purchase as (
16     select min(date(sale_time)) as first_purchase_date, buyer_id
17     from sales
18     group by buyer_id
19 ),
20
21 final as (
22     select distinct
23         u.user_id,
24         u.username,
25         cast((u.last_name||', ' || u.first_name) as varchar(100)) as full_name,
26         f.first_purchase_date,
27         u.city,
28         u.state,
29         u.email,
30         u.phone,
31         u.like_broadway,
32         u.like_classical,
33         u.like_concerts,
34         u.like_jazz,
35         u.like_musicals,
36         u.like_opera,
37         u.like_rock,
38         u.like_sports,
39         u.like_theatre,
40         u.like_vegas
41     from
42         sales as s
43         join users as u on u.user_id = s.buyer_id
44         join first_purchase as f on f.buyer_id = s.buyer_id
45     order by
46         user_id
47
48 )
49
50
51 select * from final
```

### 3.4.3 Marts layer

The marts layer in this project aligns with the recommendations provided in the dbt best practice guide. This layer serves as the culmination point where all the staging models (atoms) and intermediate models (molecules) are organized into fully developed entities with clear identities and purposes. It is sometimes referred

to as the entity layer or concept layer, emphasizing the representation of specific entities or concepts at their unique granularity.

Within the project's marts layer, there are four data models focusing on marketing and sales. These models manifest as two dimension tables and two fact tables. Although traditionally these would be labeled as star schema dimension (dim-) or fact (fct-) tables, it is important to note that the fact tables in this project are actually wide, denormalized tables. Wide tables are known to enhance analytics performance in modern data warehouses, as affirmed by Fivetran and other industry sources.

**Listing 3.3:** fct-sales model (fct-sales.sql)

```
1 {{ config(materialized='table', sort='sale_id', dist='sale_id') }}
2
3 with categories as (
4     select * from {{ ref('stg_tickit__categories') }}
5
6 ),
7
8 dates as (
9     select * from {{ ref('stg_tickit__dates') }}
10
11 ),
12
13 events as (
14     select * from {{ ref('stg_tickit__events') }}
15
16 ),
17
18 listings as (
19     select * from {{ ref('stg_tickit__listings') }}
20
21 ),
22
23 sales as (
24     select * from {{ ref('stg_tickit__sales') }}
25
26 ),
27
28 sellers as (
29     select * from {{ ref('int_sellers_extracted_from_users') }}
30
31
32
33
34
35
36
```

```
37 ),
38
39 buyers as (
40     select * from {{ ref('int_buyers_extracted_from_users') }}
41 )
42 ),
43
44 event_categories as (
45     select
46         e.event_id,
47         e.event_name,
48         c.cat_group,
49         c.cat_name
50     from events as e
51     join categories as c on c.cat_id = e.cat_id
52 )
53 ),
54
55 final as (
56     select
57         s.sale_id,
58         s.sale_time,
59         d.qtr,
60         ec.cat_group,
61         ec.cat_name,
62         ec.event_name,
63         b.username as buyer_username,
64         b.full_name as buyer_name,
65         b.state as buyer_state,
66         b.first_purchase_date as buyer_first_purchase_date,
67         se.username as seller_username,
68         se.full_name as seller_name,
69         se.state as seller_state,
70         se.first_sale_date as seller_first_sale_date,
71         s.ticket_price,
72         s.qty_sold,
73         s.price_paid,
74         s.commission_prcnt,
75         s.commission,
76         s.earnings
77     from
78         sales as s
79         join listings as l on l.list_id = s.list_id
80         join buyers as b on b.user_id = s.buyer_id
81         join sellers as se on se.user_id = s.seller_id
82         join event_categories as ec on ec.event_id = s.event_id
```

```
86         join dates as d on d.date_id = s.date_id
87     order by
88         sale_id
89 )
90
91 select * from final
92
93 select * from final
```

## 3.5 Testing and Validation

dbt makes data integrity and data quality pretty effortless. Test are written and then assigned right next to the model we have just created in a configuration .yml file. Each time we run a new model the command dbt test is executed and all tests related to that model will be launched.

dbt incorporate a set of functionalities dedicated to desting in order to allow users to define different kind of tests. The main goal is to validate the correctness and integrity of models which must conform to expected behaviour and produce accurate results. The testing framework includes data type validations, uniqueness checks, referencial integrity tests, and custom SQL-based tests.[16]

```

1  models:
2  - name: fct_sales
3    description: All sales with details
4    columns:
5      - name: sale_id
6        description: primary key
7        tests:
8          - unique
9          - not_null
10     - name: price_paid
11       tests:
12         - test_sale_amount
13
14     - name: kpi_sales_per_city_category
15       description: Indicators of tickets' sales by city per each category
16       columns:
17         - name: tickets_sold
18           tests:
19             - test_generic_assert_positive_value
20         - name: amount_paid
21           tests:
22             - test_generic_assert_positive_value

```

**Figure 3.5:** YAML file showing testing configurations

As shown in the figure above, we can have three different types of test: built-in, single and generic tests. The built-in tests do not need to be written because they have been already created and included in the dbt project. The single test must be written like a SQL select statement, if the select statement returns any value the test is not passed. This is the logic for testing in a dbt project.

```

1  select *
2  from {{ ref('fct_sales') }} as s
3  where s.ticket_price * s.qty_sold != s.price_paid

```

**Figure 3.6:** Tasks assigned to each of the role within a data Team.

Finally there is the generic test. It is written like a Jinja function that takes as input the name of the model and the column we want to test. This means that we

can apply the same test more than once to whatever model saving a lot of time.

```
test_generic_assert_positive_value.sql

1  {% test test_generic_assert_positive_value(model, column_name) %}
2  select
3      {{column_name}}
4  from {{ model }}
5  where {{column_name}} < 0
6  {% endtest %}
```

Figure 3.7: Tasks assigned to each of the role within a data Team.

### 3.6 Data Lineage and Documentation

dbt offers built-in features for data lineage and documentation, which are crucial for understanding and maintaining models. Data lineage in dbt provides visibility into the flow of data, tracking the dependencies between different models and their sources. This enables users to trace the origin of data and understand how changes in upstream data sources impact downstream models. Additionally, dbt facilitates the generation of documentation, automatically creating documentation pages that describe the purpose, structure, and dependencies of each model. This documentation feature promotes better understanding, collaboration, and maintainability of models over time.

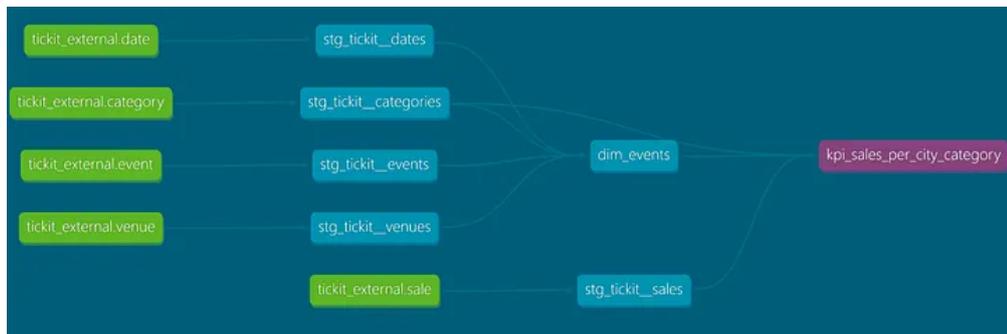


Figure 3.8: example of an Auto-generated data-lineage

dbt automates the generation of documentation around descriptions(.yml configuration files), model dependencies (ref() functions), model SQL, sources, and tests. The documentation displays existing models, relevant database objects, and detailed information about each model.

By default, dbt generates documentation in HTML format with the "dbt docs generate" command. This docs can be visualized on the auto-generated link that is inserted in the project folder after the above command has been run. The following is a view of the website for our project documentation.[17]

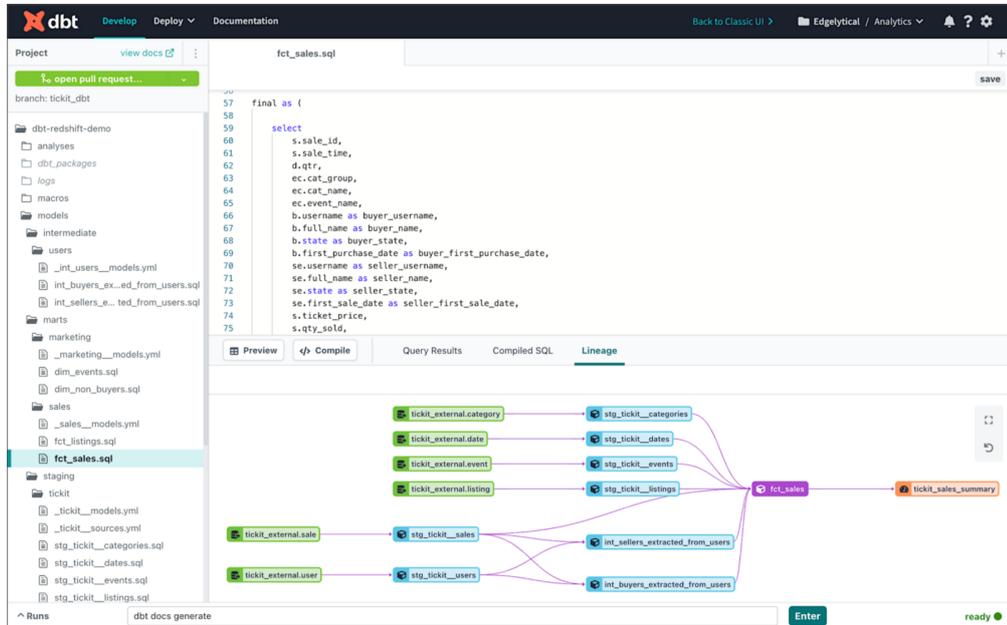


Figure 3.9: dbt project documentation website

## Chapter 4

# AWS data pipeline with dbt serverless

The next chapter will describe the whole project developed during my internship in Data Reply srl. The project concerns the implementation of an ELT data pipeline leveraging AWS cloud computing services as a Lakehouse and dbt for the data transformation phase.

### 4.1 Technologies

#### 4.1.1 S3 - Simple Storage Service

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Users of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

In this project has been used to store the original datasource of the Ticket database. Once in a bucket we have a direct access to this sources to be manipulated and loaded in the Data Warehouse.[18]

#### 4.1.2 Redshift

Amazon Redshift is a fast, fully managed, petabyte-scale data warehouse service that makes it simple and cost-effective to efficiently analyze all your data using your existing business intelligence tools. Its optimization cover different usecases of

datasets ranging from a few hundred gigabytes to a petabyte or more and costs less than \$1,000 per terabyte per year, a tenth the cost of most traditional data warehousing solutions.

Redshift is provided by Amazon Web Services (AWS). Its design allow to handle large-scale data analytics workloads and provides high-performance querying and processing capabilities.

Key features and characteristics of Amazon Redshift include:

1. columnar storage: by utilizing columnar storage, data blocks in Amazon Redshift are designed to store values of a single column for multiple rows. When new records are introduced into the system, they are automatically transformed into columnar storage for each corresponding column.

Columnar storage allows each data block to accommodate column field values for up to three times the number of records compared to row-based storage. Consequently, retrieving the same number of column field values for the same number of records necessitates only one-third of the I/O operations compared to row-wise storage. Notably, in real-world scenarios with tables featuring numerous columns and a significant number of rows, the storage efficiency is even more pronounced.

Another notable benefit is that since each block holds homogeneous data, it becomes feasible to employ compression techniques tailored specifically to the data type of the column, resulting in further reduction in disk space and I/O. For additional details regarding compression encodings based on data types, further references can be consulted.

2. Massive Parallel Processing (MPP): Redshift leverages a distributed and parallel processing architecture. It automatically distributes data and query execution across multiple nodes, allowing for high scalability and performance.
3. Query Optimization: Redshift includes an optimizer that analyzes queries and generates efficient query plans. It can parallelize and distribute query execution across multiple nodes to achieve optimal performance.
4. package: dbt-labs/redshift version: 0.8.0

[redshift]

In this thesis, it has been chosen as the Data Warehouse to which to connect and configure dbt.

### 4.1.3 Glue Data Catalogue

AWS Glue Data Catalog is a metadata repository provided by Amazon Web Services (AWS). It is a central catalog that stores and organizes metadata information about

various data sources, such as databases, tables, and schemas, within an AWS environment. As metadata repository AWS Glue Data Catalogue collects and store informations such as table schemas, data types, partitions and statistics.

To rapidly collect this metadata automated crawling has been implemented. This technique is based on the ability to analyze each sort of data source, such as S3 buckets, Redshift, JDBC/ODBC databases, extracting and classifying metadata. Automated Crawling: Glue Data Catalog can automatically discover and crawl data sources to infer their schemas and extract metadata. It supports a wide range of data sources, including Amazon S3, Amazon RDS, Amazon Redshift, and other JDBC/ODBC compatible databases.

Moreover The Data Catalog supports schema evolution, allowing it to handle changes in data structures over time. It can detect and manage updates to tables, including new columns, deleted columns, and changed data types. It make easier to discover and access data by providing a metadata-driven approach. It enables users to search and query the catalog to find relevant datasets and their associated metadata.[19]

In this project the service has been used to automatically detect, thanks to the crawling mechanism, all existing attributes of the Source file to create and populate external tables in the AWS Redshift Datawarehouse

#### 4.1.4 Docker

Docker is an open-source framework based on containerization, whose main goal is to simplify the building, deploying, and running of applications. Applications run inside docker virtual environments. The word environment refers to the operative system and software dependencies needed by the application to run. Runtime environments are usually independent from the underlying hardware infrastructure on which are host.

To run an application on a Docker environment, user must create an image that is a static template of the application which should include its dependencies, runtime libraries and configuration files. Images are created from Dockerfiles, which are declarative configuration files specifying the steps to build an image. Images allow high portability and fast deploying of applications.

A running instance of a Docker image within an environment is called Container. Container are considered as single and isolated process inside the virtual system even if they share the underlying kernel and resources with other containers[20]

#### 4.1.5 AWS ECR

Amazon ECR is a container image registry service provided by AWS, offering a secure, scalable, and dependable solution. It facilitates private repositories with

resource-based permissions utilizing AWS IAM. This ensures that designated users or Amazon EC2 instances can access your container repositories and images. You have the flexibility to employ your preferred CLI for pushing, pulling, and managing various types of images, including Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

ECR serves as a private, highly available, and scalable registry for storing Docker container images. It allows you to push, pull, and manage container images using the Docker CLI or any other Docker-compatible tools.

In this architecture an ECR instance has been use to host the docker image fetched to be run on a fargate virtual environment in order to execute dbt command on the thesis's dbt project. [21]

#### 4.1.6 AWS Fargate

AWS Fargate is a serverless compute engine for containers provided by Amazon Web Services (AWS). It allows you to run containers without managing the underlying infrastructure. With Fargate, you can focus on designing and deploying containerized applications without worrying about server provisioning, scaling, or patching.[22]

This is why it was used to automatically configure an environment capable of running our ECR container. In particular Fargate offer a way to communicate with the container while an instance of the environment is running, they are called ECS TAsK and are described in the next subsection

#### 4.1.7 ECS

ECS stands for Elastic Container Service, and it is an AWS service that allows the user to easily deploy their applications on a scalable and manageable underlying infrastructure. In fact ECS provide a user interface to set-up all the requisites and characteritics of the desired environment and a way to manage their scaling and scheduling.

In this project it has been used, for sake of simplicity, a more powerful ECS service called Fargate. With Fargate, you can define your containerized applications, specify their resource requirements, and let Fargate handle the scaling, scheduling, and management of the underlying infrastructure. This makes it easier to deploy and manage containers without the need for manual provisioning or configuration of EC2 instances.[23]

### **4.1.8 Lambda**

Lambda, offered by Amazon Web Services (AWS), is a serverless computing service that enables you to execute your code without the need to provision or manage servers. It is specifically designed to execute functions, which are small units of code, in response to various events or triggers. With Lambda, you can effortlessly run your code while abstracting away the complexities of server management.[24]

In our system Lambda function has been used to respond to any eventual failure in the system. So while running dbt model creation or dbt test we can monitor the logs outputted by an ECS Task and react to it by launching a predefined AWS Lambda function.

### **4.1.9 CloudWatch**

CloudWatch is a service provided by Amazon Web Services (AWS) that helps monitor and observe your data. It enables you to collect, store, and analyze logs, metrics, and events from various AWS resources and applications.

In this project i exploited the Logs monitoring and alarm mechanism as explained in the previous subsection about Lambdas. Logs enables you to aggregate, store, and analyze logs generated by your applications and AWS services. You can centralize logs from different sources, search and filter log data, and set up alarms and notifications based on log events.[25]

## 4.2 Overall Architecture

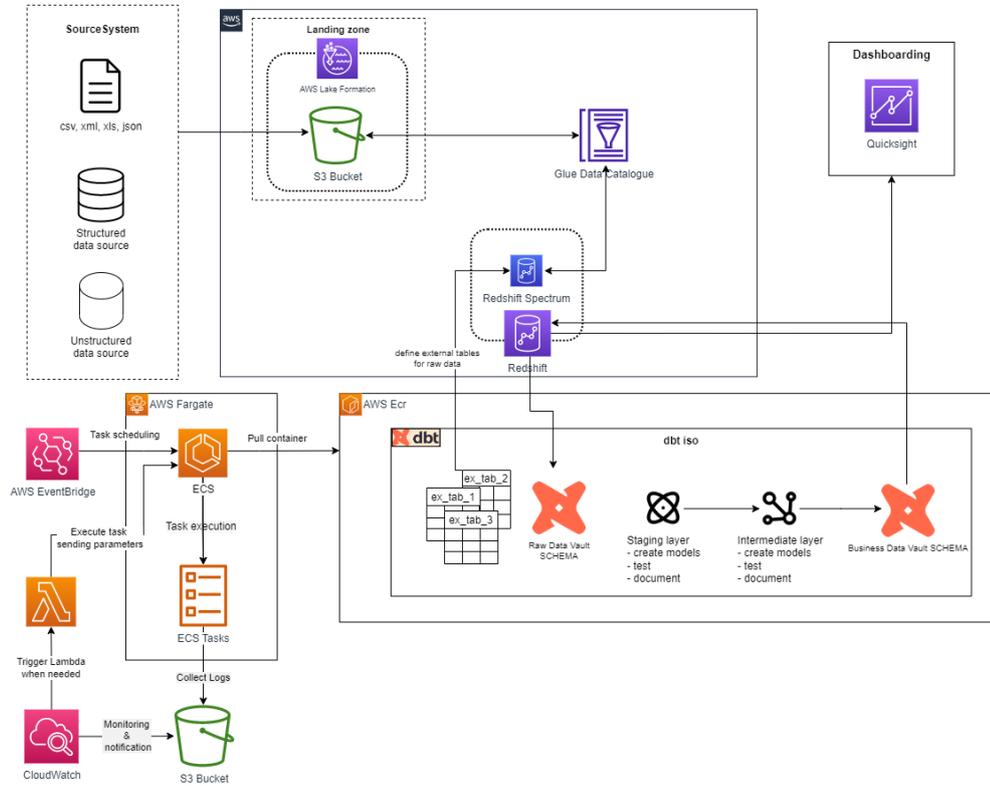


Figure 4.1: Data pipeline diagram

The above architecture depicts a simple ELT pipeline in which raw data is loaded directly from sources into an S3 bucket. These raw data are then used to populate external tables in the Redshift data warehouse, which serve as the source tables for the dbt project. All the transformation phases, including testing and documentation, are managed by dbt. I have implemented a serverless version of dbt using a dbt Docker image hosted in ECR. A cron schedule has been configured using the AWS EventBridge service to run ECS tasks that execute any dbt CLI command. Logs from ECS task execution are then monitored using Cloudwatch, and Lambda functions can be used to respond to any potential failures.

Once the data has been transformed and cleaned, it is ready to be displayed and analyzed in Quicksight

## 4.3 Data Ingestion Phase

The sources have not been implemented, instead data have been manually loaded in the s3 bucket as .txt files for simplicity. Thus Data Ingestion has been avoided since it was not the main theme to be treated in this thesis. Nevertheless the architecture allow different techniques of data ingestion and can manage whatever kind of data both structure and unstructured.

## 4.4 Data Storage and Management

The Raw data is collected in the S3 instances. The main goal is to load all this raw data into the Data Warehouse so that they are reachable and modified depending on the analysis needs. The technique used leverages external tables. They are a feature that allows you to query and join data stored in external data sources directly from your Redshift cluster without having to load the data into the cluster's storage. Instead, the data remains in its original location, such as Amazon S3, and Redshift treats it as an external table.

In Redshift text editor and query runner the following instruction has been written and run:

**Listing 4.1:** fct-sales model (fct-sales.sql)

```

1 create external schema tickit_external
2 from data catalog
3 database 'tickit_dbt'
4 iam_role 'arn:aws:iam::<your_aws_account_id>:role/ClusterPermissionsRole'
5 create external database if not exists ;

```

What is done with this command is to create a new schema in the Glue Data Catalog called "tickit-external". This schema will be the place where to store all meta data of our external tables.

To build external tables, it is needed to operate on the dbt project. The tickit-sources.yml file in the models/staging/tickit/external-tables/ model's subdirectory defines the schema and S3 location for each of the seven external TICKIT database tables: category, date, event, listing, sale, user, and venue.

**Listing 4.2:** Category external table schema definition (Category external table schema definition)

```

1 version: 2
2
3 sources:
4   name: tickit_external
5   description: Sales activity for the fictional TICKIT web site, where users buy and
   sell tickets online for sporting events, shows, and concerts.

```

```

6  database: demo
7  schema: tickit_external
8  loader: s3
9  tables:
10     name: category
11     description: dimension table TICKIT categories
12     external:
13     location: "s3://<your_s3_bucket_name>/raw_tickit_data/category/"
14     row_format: >
15     serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
16     with serdeproperties (
17         'separatorChar'=|'
18     )
19     table_properties: "('skip.header.line.count'='1')"
20     columns:
21     name: catid
22     data_type: int
23     description: primary key
24     tests:
25         unique
26         not_null
27     name: catgroup
28     data_type: varchar(20)
29     name: catname
30     data_type: varchar(20)
31     name: catdesc
32     data_type: varchar(50)

```

To create the seven external tables in the AWS Glue Data Catalog it was executed the command, `dbt run-operation stage-external-sources`. This command is part of the `dbt-external-tables` package we installed earlier. It iterates through all source nodes, creates the tables if missing, and refreshes metadata.

Focus must be paid to "schema" field which is the one previously created and to the field "location". This package will look for the table name inside the AWS Glue Data Catalogue schema and if not present it will generate all the metadata defined in the `tickit-sources.yml` file.[14]

At this point the Datawarehouse is configured to access data source by means of the external tables. So it is possible to create whatever models needed for Data analysis.

## 4.5 Data Transformation with dbt

As shown in Chapter 3 - Model creation, all the transformation phase is handled on the dbt project by an analytics engineer. It's up to the data engineer now to create a serverless environment on AWS that can host the dbt project.

In the proposed solution the following steps has been executed :

1. Creation of a Docker image of the dbt project
2. Loading of the image in a ECR instance
3. Configuration of the environment on AWS Fargate
4. AWS ECS Task definition
5. Cron schedule on AWS Event Bridge

The docker image has been built from the following dockerfile :

```
1 FROM python:3.9
2
3 # Set the working directory inside the container
4 WORKDIR /app
5
6 # Copy the dbt project files to the container
7 COPY . /app
8
9 # Install dependencies
10 RUN pip install dbt
11
12 # Set the entrypoint command to run dbt with the profiles.yml file as a parameter
13 ENTRYPOINT ["dbt", "run", "--m", "/root/.dbt/profiles.yml"]
```

The docker image generated has been loaded onto an ECR instance.

Then the Fargate environment has been configured with the following steps :

1. Configure Task Definition: In the AWS Management Console, a new task definition has been created in the ECS service. The task definition must be set with the required CPU and memory resources.
2. Under "Container Definitions", is added a container that uses the ECR image pushed in step 1.
3. Environment variables and volume mounts required by your dbt project has been specified.
4. Fargate Service: After creating the task definition, an ECS Task has been created to run the dbt project.
5. Fargate launch type definition. Configured the service with the desired number of tasks and the cluster where to run them.

6. Specify the task definition you created in step 2. Security Groups and VPC Configured : Fargate tasks must have the necessary security group and VPC configurations to access the required resources, such as the Data Warehouse.

An interesting solution has been adopted to make the execution of some dbt commands iterated according to a cron schedule. It as been considered in our use case that new raw data arrive from our sources on a daily basis and our dbt models must be re-run to keep the DataWarehouse updated. This functionalities has been implemented exploiting the AWS EventBridge services that allow to generate a Cron schedule of ECS Tasks.

## 4.6 Data Processing and Analysis

The final models, the ones inside the "marts layer," have been used to generate analysis about the sales of tickets on the fictional Tickit website. The BI services adopted for this aim has been Quicksight.

In the first panel, various graphics have been generated to visualize the number of tickets sold with respect to the category of the event and the period of the year.

An other graph instead shows the geographical distribution of sales in all the American states.

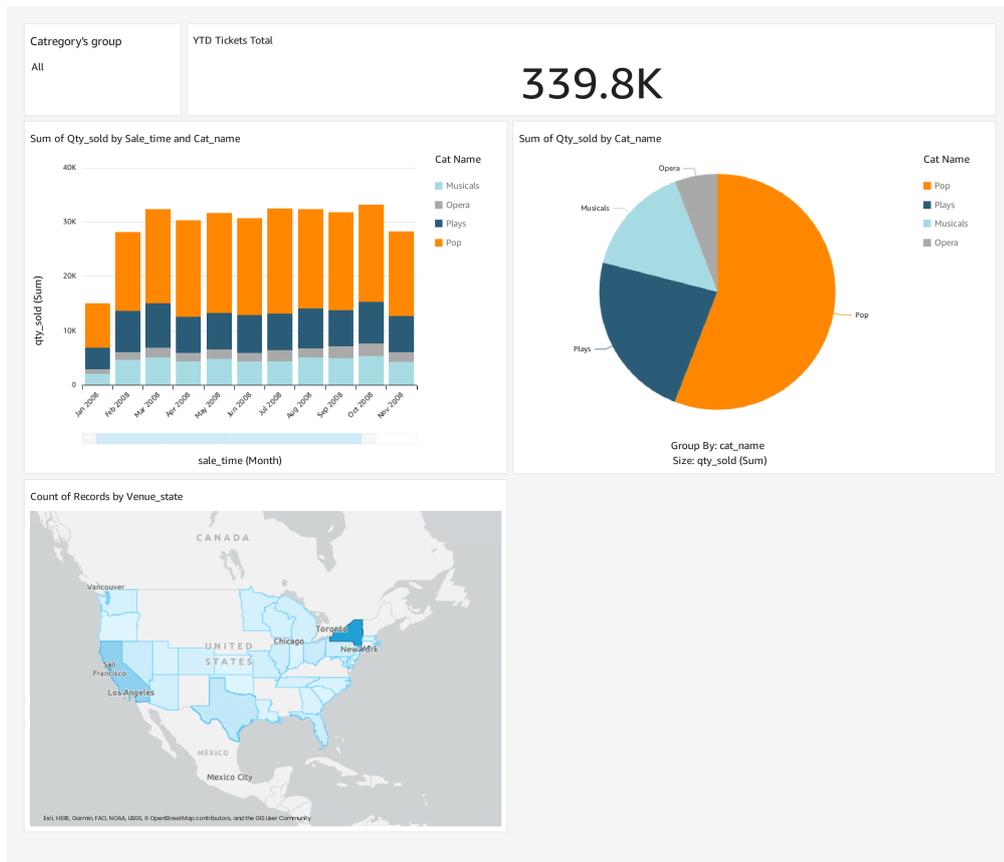


Figure 4.2: Quicksight statistics about the number of tickets sold

The second panel shows the statistics about the total revenues, net earnings, and commissions.

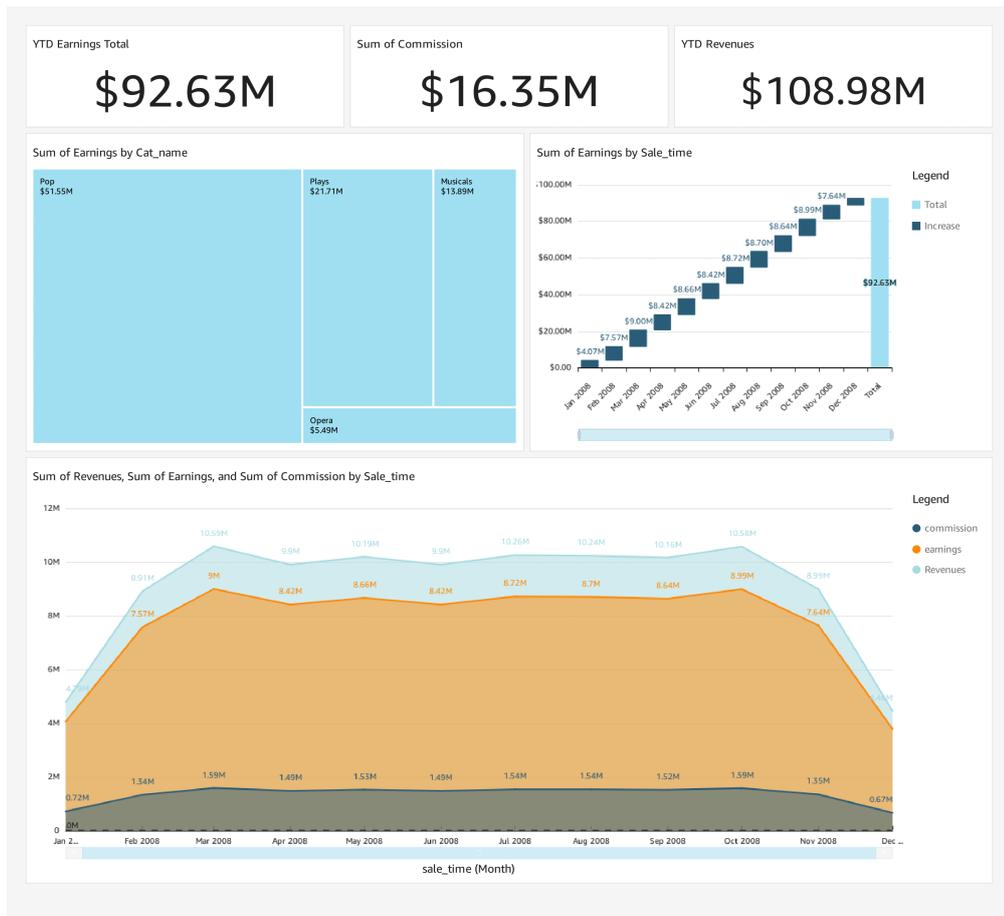


Figure 4.3: Quicksight statistics about Earnings

The last one instead is more oriented to analyzing the users of the website. Both buyer and sellers can be grouped by their interests and level of activity on the site.

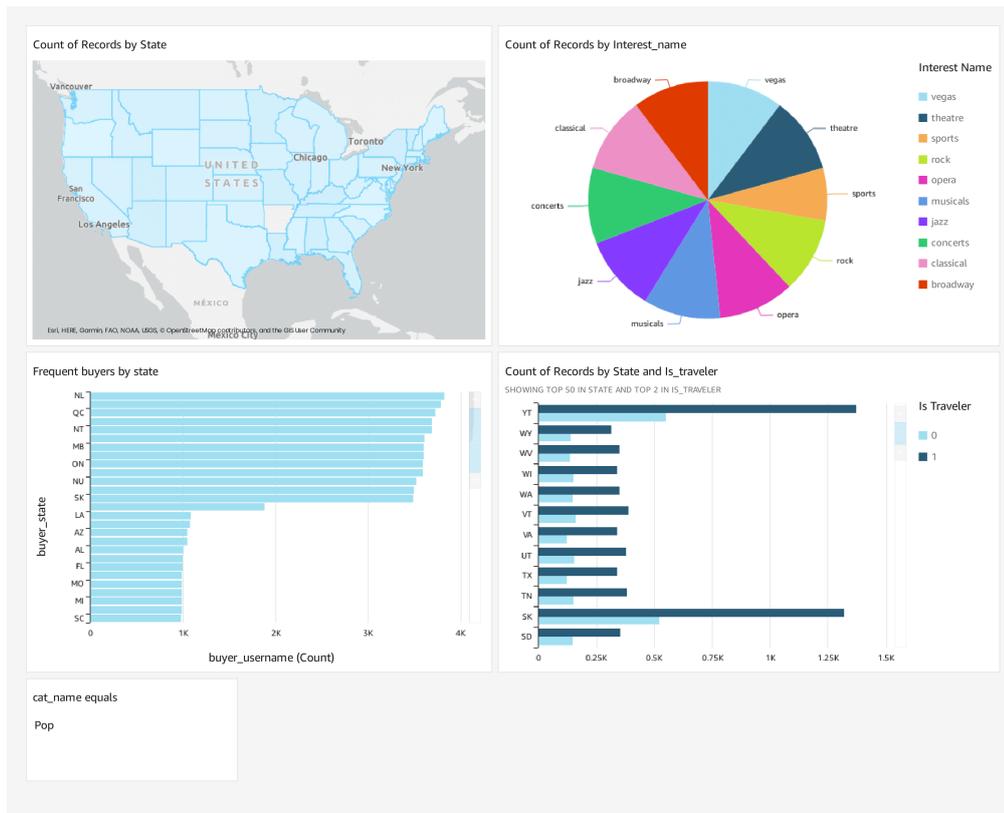
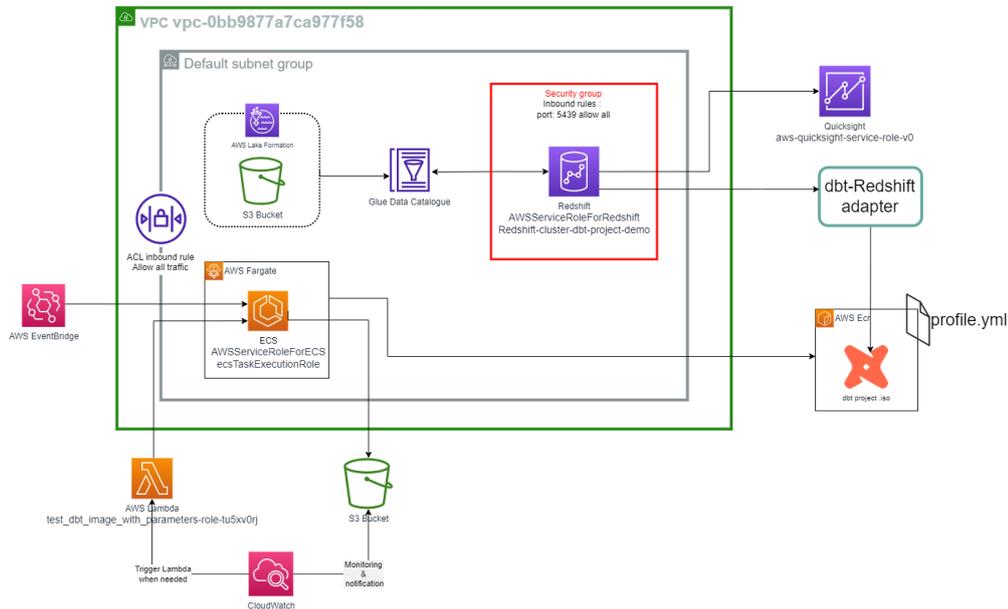


Figure 4.4: QuickSight statistics about Users

## 4.7 Security and Data Governance



**Figure 4.5:** AWS network and policies adopted the project

The above schema depicts the AWS VPC and all the internal subnets, security groups, and ACLs involved in the pipeline, including all the roles assumed by each service to operate within it.

The Virtual Private Cloud (VPC) encompasses all the resources in use. It is termed "virtual" because there is no dedicated physical hardware solely operating for us. The underlying hardware is shared among multiple users, but the VPC ensures that the allocated resources remain private and accessible only with the appropriate permissions.[26]

Various technologies are employed to ensure privacy and security within the VPC. The diagram above illustrates two such technologies: ACL and Security Group.

ACL (Access Control List) is commonly utilized in routers, switches, and firewalls to regulate the flow of network traffic. It consists of a set of rules that determine the allowance or denial of network packets within a group. Operating at both the network layer (Layer 3) and transport layer (Layer 4) of the OSI model, ACL controls inbound and outbound traffic. It is stateless, meaning it does not remember the previous state of network connections.[27]

On the other hand, the Security Group functions as a virtual firewall for instances (virtual machines) within the VPC. It establishes rules solely for inbound traffic,

enabling instances to communicate exclusively with authorized senders. Unlike ACL, the Security Group is stateful, meaning it remembers previously authorized senders, allowing bidirectional communication while maintaining security.

By combining ACL and Security Group, the VPC achieves robust privacy and security measures, ensuring controlled access and secure communication within the virtual network environment.

## Chapter 5

# Speculative Analysis : comparison between proposed and traditional paradigm

As starting point of this chapter the main problem of the traditional paradigma must be stressed : the chasm from business to IT.

In a previous era of data analysis, an individual had the opportunity to acquire all the necessary expertise to become an expert in both the specific domain and the relevant data technology. The size and diversity of data were limited, and people typically developed the required data skills (such as Excel, basic SQL, SAS) as they encountered problems. The datasets were small in scale and focused, allowing this approach to work effectively. It was as simple as obtaining a CSV file and diving into the analysis.

However, this approach became inadequate with the emergence of the modern data stack. Presently, the potential for analysis has expanded exponentially, but so have the range and depth of skills required. Consequently, a single domain expert sitting down at a computer to explore data is no longer sufficient to obtain answers. In the contemporary data ecosystem, answering questions necessitates collaborative teamwork, a meticulously designed workflow, and purpose-built tools that facilitate this collaboration.

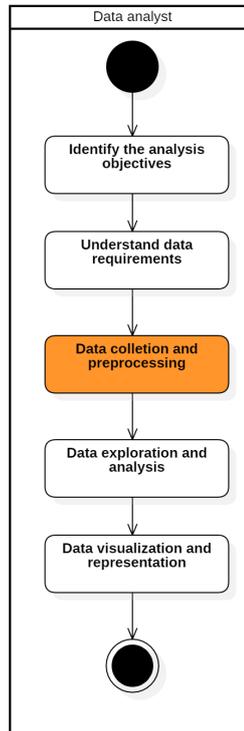
In the next paragraph, the process of creating a new dataset for analysis purposes has been deconstructed into individual activities for both the new and traditional paradigms. Additionally, a design comparison has been conducted to emphasize the significant differences and advantages of the proposed solution.

## **5.1 Process Analysis: Model creation for business insights**

When there is a need for a new data model for analysis purposes, usually the following steps must be executed:

1. Identify the analysis objective: Clearly define the purpose and goal of the data model. Determine the specific insights or questions that need to be addressed.
2. Understand the data requirements: Collaborate with stakeholders, such as business users or domain experts, to understand the data needed to achieve the analysis objective. Determine the data sources and variables required for the model.
3. Data collection and preprocessing: Acquire the necessary data from the identified sources. Clean and preprocess the data to ensure accuracy, consistency, and completeness. Handle missing values, outliers, and perform data transformations as needed.
4. Dataset creation, validation and refinement: Evaluate the performance of the dataset using appropriate metrics and validation techniques. Refine the dataset as necessary to improve accuracy and reliability.
5. Data exploration and analysis: Conduct exploratory data analysis to gain a deeper understanding of the data. Perform descriptive statistics, data visualization, and identify patterns, trends, and correlations within the dataset.
6. Data representation and visualization

To conduct the process analysis two activity diagrams have been created. The first diagram illustrates a scenario where the Data analyst independently executes all the steps without involving the Technology teams.

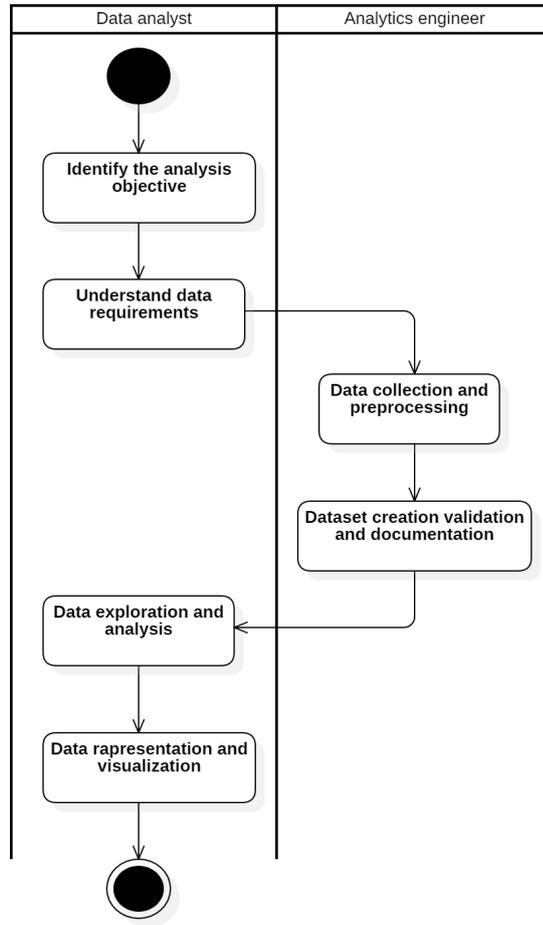


**Figure 5.1:** Activity diagram describing model creation with Data analyst

However, challenges arise in this scenario due to the Data analyst’s primarily business-oriented skill set. The most critical step here is the Data collection and preprocessing phase, which requires a deep understanding of the data infrastructure. It involves selecting the appropriate Data sources, ensuring data quality, handling data transformations, and addressing any inconsistencies. This can lead to potential errors or limitations in the resulting data model, affecting the quality and reliability of the insights derived from it.

Therefore, collaboration with Technology teams, such as Analytics engineers or Data engineers, who possess the necessary technical expertise, can help mitigate these challenges and enhance the effectiveness of the data model creation process.

In contrast, the second diagram represents a scenario where either a Data analyst or a Data engineer is involved in performing these operations.



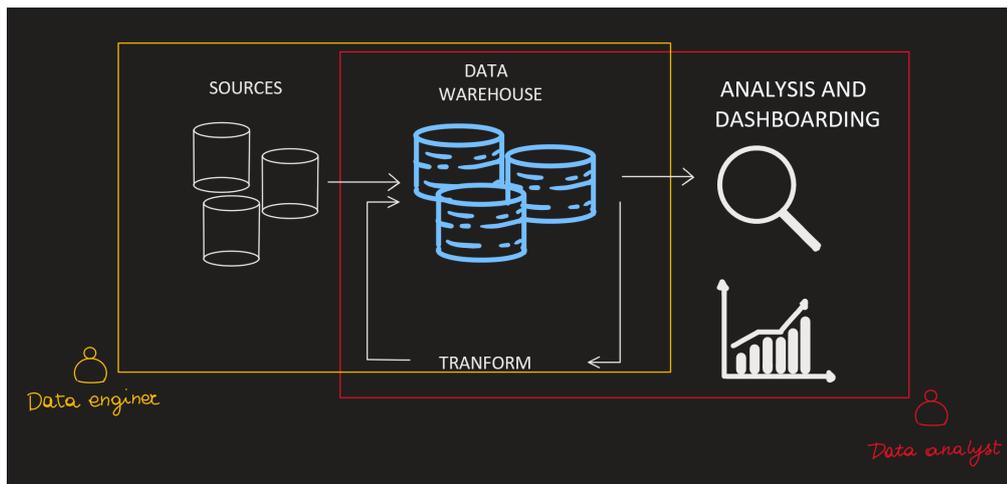
**Figure 5.2:** Activity diagram describing model creation with Analytics engineer

Analytics engineer should not be replaced by a Data engineer. Data engineering team in this case have few understanding of the business needs and all the data collection process risk to be not accurate or not relevant. Data engineering teams usually provide data models that highlight only KPIs but are not able to provide real insight; furthermore, they are not experts in the discovery process of data analysis. Their job is to build and maintain the Data pipeline and manage its overall orchestration. This means that if a data analyst makes a request to data engineers, it will be added to the list of scheduled tasks to be completed. The lack of autonomy and the unpredictable waiting time significantly impair the efficiency of the analysts' work. Lloyd, an individual associated with Looker, refers to the concept of "data breadlines" to illustrate a scenario where data consumers are required to wait in line for assistance from the technology team before they can carry out specific analyses. This waiting period significantly hampers the speed of

analysis and diminishes the enthusiasm of data consumers.

## 5.2 Design comparison and final thoughts

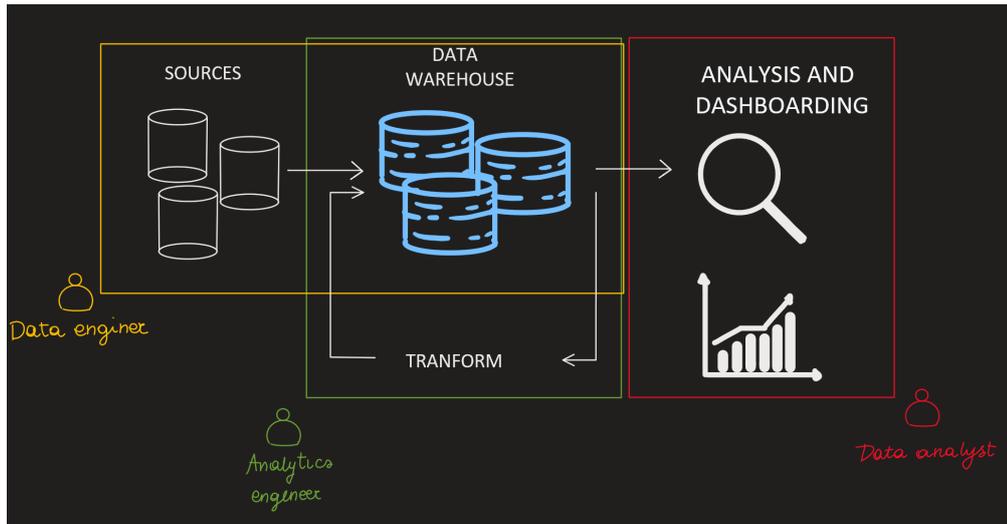
For the sake of comparison both the pipelines has been For the purpose of comparison, both pipelines have been simplified, highlighting the main stages and indicating the responsible parties for each stage.



**Figure 5.3:** Roles in traditional data pipelines

For the purpose of comparison, both pipelines have been simplified, highlighting the main stages and indicating the responsible parties for each stage.

One significant observation is the overlapping of roles during the transformation phase, indicating a lack of clear definition regarding the process of dataset creation, which serves as the foundation for analytics..



**Figure 5.4:** Roles in modern data pipelines

In the proposed paradigm, Analytics engineers are knowledge specialists: like librarians, they curate an organization's knowledge. They acquire, codify and make sure that all the knowledge is reliable and current. They represent the bridge between business and technology that fill this gap between existing roles and competences. These are the individuals who have been absent from our data projects, and this is the practice that we must refine if we are to address the fundamental dysfunction of data. With a well-established analytics engineering practice, knowledge is gradually accumulated by numerous individuals through incremental contributions.[28]

## Chapter 6

# Conclusion

The thesis project offers a practical use case where the new paradigm introduced by Analytics Engineering has been adopted and experimented with. Data engineers and data analysts can now focus on their primary tasks, relying on the expertise of analytics engineers for the creation of data models.

The final solution is a cloud-based data pipeline implemented on AWS. The data sources are stored as raw data in the Simple Storage Service (S3) and made accessible through Redshift Data Warehouse as external tables. The entire data transformation phase has been managed using dbt, which has been extensively tested and proven to be a comprehensive toolkit for analytics engineering. Within the dbt project, model creation, testing, validation, and documentation are fully supported and partially automated to ensure accurate and reliable datasets.

The dbt project has been deployed as a Docker image on an ECR instance and executed within a pre-configured virtual environment on AWS Fargate. To utilize the dbt command-line interface (CLI) and execute any type of command within the project, a cron schedule of ECS tasks has been established using EventBridge. Furthermore, an automated system for failure detection has been implemented through log monitoring with CloudWatch. Lambda functions have proven to be a reliable service for executing functions that respond appropriately to these failures.

Moving forward, several potential improvements can be considered for future enhancements. Firstly, exploring the integration of additional data sources could broaden the scope and depth of insights derived from the pipeline. Secondly, enhancing the error handling and recovery mechanisms within the system would further improve the pipeline's reliability. Additionally, implementing real-time or near real-time data processing capabilities could enable more dynamic and timely analytics. Lastly, integrating a comprehensive data governance framework and security measures would ensure data privacy and compliance with regulations.

By addressing these future improvements and incorporating these elements, the data pipeline with dbt can continue to evolve and provide even more valuable

insights for data engineers, data analysts, and other stakeholders involved in the analytics process.



# Bibliography

- [1] *Introduction to dbt*. <https://docs.getdbt.com/docs/introduction> (cit. on pp. 8, 15).
- [2] Aiswarya Raj, Jan Bosch, Helena Holmström Olsson, and Tian J. Wang. «Modelling Data Pipelines». In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 13–20. DOI: 10.1109/SEAA51224.2020.00014 (cit. on p. 8).
- [3] IBM. *IBM ELT Documentation*. 2021. URL: <https://www.ibm.com/topics/elt> (visited on 07/07/2023) (cit. on p. 10).
- [4] IBM. *ELT vs ETL: What's the Difference*. 2021. URL: <https://www.ibm.com/cloud/blog/elt-vs-etl-whats-the-difference> (visited on 07/07/2023) (cit. on p. 11).
- [5] Matthew N.O. Sadiku, Sarhan M. Musa, and Omonowo D. Momoh. «Cloud Computing: Opportunities and Challenges». In: *IEEE Potentials* 33.1 (2014), pp. 34–36. DOI: 10.1109/MPOT.2013.2279684 (cit. on p. 11).
- [6] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. «Cloud Computing: An Overview». In: *Cloud Computing*. Ed. by Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 626–631. ISBN: 978-3-642-10665-1 (cit. on p. 11).
- [7] Zhuangdi Zhu, Alex X. Liu, Fan Zhang, and Fei Chen. «FPGA Resource Pooling in Cloud Computing». In: *IEEE Transactions on Cloud Computing* 9.2 (2021), pp. 610–626. DOI: 10.1109/TCC.2018.2874011 (cit. on p. 12).
- [8] Zryan Najat Rashid, Subhi R. M. Zebari, Karzan Hussein Sharif, and Karwan Jacksi. «Distributed Cloud Computing and Distributed Parallel Computing: A Review». In: *2018 International Conference on Advanced Science and Engineering (ICOASE)*. 2018, pp. 167–172. DOI: 10.1109/ICOASE.2018.8548937 (cit. on p. 12).
- [9] dbt. *What is Analytics Engineering?* 2021. URL: <https://www.getdbt.com/what-is-analytics-engineering/> (visited on 07/07/2023) (cit. on p. 14).

- [10] dbt. *What Exactly is dbt?* 2021. URL: <https://www.getdbt.com/blog/what-exactly-is-dbt/> (visited on 07/07/2023) (cit. on p. 14).
- [11] dbt. *dbt Documentation - dbt Jinja Functions*. 2021. URL: <https://docs.getdbt.com/reference/dbt-jinja-functions> (visited on 07/07/2023) (cit. on p. 16).
- [12] dbt. *Customizing CI/CD*. <https://docs.getdbt.com/guides/orchestration/custom-cicd-pipelines/1-cicd-background>. 2023 (cit. on p. 18).
- [13] dbt. *dbt Documentation - dbt init*. 2021. URL: <https://docs.getdbt.com/reference/commands/init> (visited on 07/07/2023) (cit. on p. 18).
- [14] Programmatic Ponderings. *Lakehouse Data Modeling using dbt, Amazon Redshift, Redshift Spectrum, and AWS Glue*. 2022. URL: <https://programmaticponderings.com/2022/08/19/lakehouse-data-modeling-using-dbt-amazon-redshift-redshift-spectrum-and-aws-glue/> (visited on 07/07/2023) (cit. on pp. 19, 21, 37).
- [15] dbt. *dbt Documentation - dbt run*. 2021. URL: <https://docs.getdbt.com/reference/commands/run> (visited on 07/07/2023) (cit. on p. 22).
- [16] dbt. *dbt Documentation - dbt test*. 2021. URL: <https://docs.getdbt.com/reference/commands/test> (visited on 07/07/2023) (cit. on p. 26).
- [17] dbt. *dbt Documentation - dbt docs*. 2021. URL: <https://docs.getdbt.com/reference/commands/cmd-docs> (visited on 07/07/2023) (cit. on p. 29).
- [18] Amazon Web Services. *Amazon Simple Storage Service (S3) Documentation*. 2021. URL: <https://docs.aws.amazon.com/s3/index.html> (visited on 07/07/2023) (cit. on p. 30).
- [19] Amazon Web Services. *AWS Glue Data Catalog Documentation*. 2021. URL: <https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html> (visited on 07/07/2023) (cit. on p. 32).
- [20] Docker. *Docker - Get Started Overview*. 2021. URL: <https://docs.docker.com/get-started/overview/> (visited on 07/07/2023) (cit. on p. 32).
- [21] Amazon Web Services. *AWS Elastic Container Registry (ECR) Documentation*. 2021. URL: <https://docs.aws.amazon.com/ecr/index.html> (visited on 07/07/2023) (cit. on p. 33).
- [22] Amazon Web Services. *AWS Fargate Documentation*. 2021. URL: <https://docs.aws.amazon.com/fargate/index.html> (visited on 07/07/2023) (cit. on p. 33).
- [23] Amazon Web Services. *AWS Elastic Container Service (ECS) Documentation*. 2021. URL: <https://docs.aws.amazon.com/ecs/index.html> (visited on 07/07/2023) (cit. on p. 33).

## BIBLIOGRAPHY

---

- [24] Amazon Web Services. *AWS Lambda Documentation*. 2021. URL: <https://docs.aws.amazon.com/lambda/index.html> (visited on 07/07/2023) (cit. on p. 34).
- [25] Amazon Web Services. *Amazon CloudWatch Documentation*. 2021. URL: <https://docs.aws.amazon.com/cloudwatch/index.html> (visited on 07/07/2023) (cit. on p. 34).
- [26] Amazon Web Services. *AWS VPC Documentation*. 2021. URL: <https://docs.aws.amazon.com/vpc/index.html> (visited on 07/07/2023) (cit. on p. 43).
- [27] Amazon Web Services. *AWS ACL Documentation*. 2021. URL: [https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC\\_ACLS.html](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_ACLS.html) (visited on 07/07/2023) (cit. on p. 43).
- [28] dbt. *Analytics Engineering for Everyone*. 2021. URL: <https://www.getdbt.com/blog/analytics-engineering-for-everyone/> (visited on 07/07/2023) (cit. on p. 50).

# List of Figures

2.1	Logic scheme of an etl data pipeline . . . . .	9
2.2	Logic scheme of an elt data pipeline . . . . .	9
2.3	Tasks assigned to each of the role within a data Team. . . . .	14
2.4	The role of dbt inside a data pipeline . . . . .	15
2.5	select statement example to create a new dbt model . . . . .	16
3.1	ER schema for the Tickit relational database . . . . .	18
3.2	newly created dbt project . . . . .	19
3.3	profile.yml datasource configuration file . . . . .	20
3.4	models folder structure in the dbt project . . . . .	21
3.5	YAML file showing testing configurations . . . . .	27
3.6	Tasks assigned to each of the role within a data Team. . . . .	27
3.7	Tasks assigned to each of the role within a data Team. . . . .	28
3.8	example of an Auto-generated data-lineage . . . . .	28
3.9	dbt project documentation website . . . . .	29
4.1	Data pipeline diagram . . . . .	35
4.2	Quicksight statistics about the number of tickets sold . . . . .	40
4.3	Quicksight statistics about Earnings . . . . .	41
4.4	Quicksight statistics about Users . . . . .	42
4.5	AWS network and policies adopted the project . . . . .	43
5.1	Activity diagram describing model creation with Data analyst . . .	47
5.2	Activity diagram describing model creation with Analytics engineer	48
5.3	Roles in traditional data pipelines . . . . .	49
5.4	Roles in modern data pipelines . . . . .	50

# Acronyms

API	Application Programming Interface
AWS	Amazon Warehouse System
dbt	Data Buld Tool
ECR	Elastic container repository
ECS	Elastic container service
ETL	Extract Transform and Load
ELT	Extract Load and Transform
ERP	Enterprise Resource Planning
SQL	Structured Query Language
S3	Simple Storage Service
YAML	YAML Ain't Markup Language. It is a human-readable data serialization format used for representing structured data