Politecnico di Torino

Politecnico di Torino

DATA SCIENCE AND ENGINEERING

A.a. 2024/2025

Graduation Session July 2025

Development and Orchestration of a Scalable and Efficient Automated Data Ingestion Workflows and Pipelines for Multi-Domain at MSC Technology Italia

Data Ingestion and Workflows Management

supervisor: Paolo Garza

candidate: Babelle Tchoumi Yomi

Ringraziamenti

I would like to express my deepest gratitude to Professor Paolo Garza, whose guidance, expertise, and constant support have been instrumental throughout the entire journey of my thesis. His constructive criticism, insightful advice, and unwavering encouragement have profoundly shaped the quality and direction of my research. Working under his supervision has been both an honor and a privilege.

I am also thankful to the Politecnico di Torino, and in particular the Department of Computer Engineering, Film and Mechatronics, for providing a dynamic and intellectually stimulating environment. The academic resources and high-quality teaching offered by the institution have been essential to the development of my skills and knowledge in the field of Data Science and Engineering.

My sincere thanks go to MSC Technology, and in particular to Mr. Roberto Novaro, Business Intelligence Manager, and Ms. Valentina Mauro, Team Leader, for their warm welcome and guidance throughout my thesis project. I am deeply grateful for the opportunity to be part of their team and for the rich, professionally rewarding experience I gained. The collaboration not only allowed me to apply theoretical knowledge to real-world challenges but also offered valuable insights that significantly enhanced the academic dimension of my work. I especially appreciate the trust, support, and mentorship I received during my time at MSC Technology.

I would also like to extend my heartfelt appreciation to my family. To my parents, for their unconditional love, patience, and sacrifices that have laid the foundation for my academic path. To my brothers and sisters, whose encouragement and presence have always given me strength and perspective, even in the most challenging moments.

Completing this thesis has been a journey marked by learning, growth, and perseverance, and I am truly grateful to all those who have contributed to it, directly or indirectly.

Indice

Elenco delle figure VII

1 Introduction 1
- 1.1 Goal . . . 1
- 1.2 Thesis structure . . . 2

2 Technologies 3
- 2.1 Goal . . . 3
- 2.2 Key Technologies Used . . . 3
  - 2.2.1 Informatica PowerCenter . . . 3
  - 2.2.2 Azure Synapse Analytics . . . 4
  - 2.2.3 Microsoft Fabric . . . 5
  - 2.2.4 Azure DevOps . . . 5
  - 2.2.5 Automic Automation . . . 5
  - 2.2.6 Power BI . . . 6
  - 2.2.7 Dagster . . . 6
- 2.3 Integration Strategy . . . 6

3 Integrating Data with Informatica PowerCenter 8
- 3.1 Goal . . . 8
- 3.2 Understanding the Data and Architecture . . . 8
  - 3.2.1 Source and Target Overview . . . 8
  - 3.2.2 Loading Frequency and Volume Management . . . 9
- 3.3 Setting Up Tables and Workflows . . . 9
  - 3.3.1 Getting Access and Preparing the Systems . . . 9
  - 3.3.2 Using CDC and Automating Workflow Creation . . . 10
  - 3.3.3 Initial Data Load . . . 11
- 3.4 Tracking Changes and Incremental Loading with PowerCenter . . . 12
  - 3.4.1 Designing the Mappings . . . 12
  - 3.4.2 Managing the Workflow . . . 13

4 Ingesting Data Using Azure Synapse 16
4.1 Introduction 16
4.2 How I Designed the Pipelines 16
4.3 Step-by-Step Pipeline Logic 17
4.4 Making Pipelines Reusable Across Environments 19
4.5 Benefits of this Pipeline Design 19
4.6 Orchestration with the IBOX_TN Master Pipeline 20

5 Modern Ingestion with Microsoft Fabric 21
5.1 Goal 21
5.2 Pipeline Overview 21
5.3 Failure Notification Mechanism 23
5.3.1 Notify Teams Channel Pipeline 23
5.3.2 Pipeline Structure and Logic 24
5.3.3 MasterPipeline 25
5.3.4 Key Benefits 25

6 Establishing the Real-Time Data Pipeline 27
6.1 Design and Role of Silver Tables 28
6.1.1 Automated and Dynamic Update Policy 28
6.1.2 Technical Considerations and Logic 31
6.1.3 Benefits of This Approach 31
6.1.4 Limitations and Considerations 31
6.1.5 Data Recovery Process After Schema Change 32

7 Workflow Automation and Scheduling 34
7.1 Execution of the Workflow via Automic Software Pre-Process Stage 34
7.1.1 Initialization of Variables 34
7.1.2 Retry Management 35
7.1.3 Environment Initialization 35
7.1.4 Workflow List Retrieval 35
7.2 Post-Process Execution 36
7.2.1 Job Workflow Design in Automic Software 37
7.2.2 Integration into Finance Team Daily Schedule 39
7.2.3 Results Interpretation 40

8 Versioning and Deployment Process 43
8.1 Goal 43
8.1.1 Table Deployment 43
8.1.2 Running Build and Validation (CI/CD Pipeline) 44
8.2 Automic Workflow Deployment Using Azure DevOps 45
8.2.1 Azure DevOps Pipeline Overview 45

8.3 Deployment Script Functionality 46
8.3.1 Creating the Deployment Package 46
8.3.2 Deploying to the Target Environment 48
8.3.3 Deployment Validation and Results Interpretation 49

9 Monitoring and Reporting 51
9.1 Overview of Data Discrepancy Report 51
9.1.1 Purpose and Structure 51
9.1.2 Key Metrics and Numerical Analysis 52
9.1.3 Visualization Components 53
9.1.4 Impact and Usage 54
9.2 Log Analysis Report 54
9.2.1 Purpose and Structure 54
9.2.2 Key Metrics and Visualizations 55
9.2.3 Numerical Interpretation 55
9.2.4 Impact and Usage 56

10 Reconciliation and Discrepancy Fixing Using Dagster 57
10.1 Reconciliation Workflow Design 57
10.1.1 High-Level Architecture 58
10.2 The `generic_reconciliation_fixer` Job 59
10.2.1 Job Overview 59
10.2.2 Graph Components 59
10.2.3 Detailed Steps 59
10.2.4 Execution Process 60
10.3 Use Case: Interlink_LinkMSCNet `Agency_User` Table 61
10.4 Advantages of Using Dagster for Reconciliation 61

11 Possible Improvements and Future Developments 62
11.0.1 Integration of Microsoft Data Activator for proactive detection 63
11.1 Incorporating Machine Learning for Anomaly Detection 64

12 Conclusion 65

Bibliografia 67

Elenco delle figure

3.1 Change Data Capture Generator Tool 10
3.2 SQL Server Import and Export Wizard 11
3.3 Change Data Capture Mechanism 12
3.4 CDC Mapping for BunkerManagementSystem Tables 13
3.5 BunkerManagementSystem Daily Workflow 14

4.1 IBOX$_T$*Npipeline* 17
4.2 Incremental load Pipeline 18

5.1 Dynamic Ingestion Pipeline 22
5.2 Teams Notification Pipeline 24
5.3 Master Pipeline with Error Handling 25
5.4 Failure Notification in Microsoft Teams 26

6.1 Full Eventstream pipeline from ASBonOVAsource through transformation and routing into Eventhouse bronze tables. 27

7.1 Workflow Structure of the JOBP in Automic 37
7.2 Workflow execution 38
7.3 Integration of BunkerManagementSystem JOBP into Finance Team Daily Schedule 39
7.4 Execution Results of Finance Team Daily Workflows 41

8.1 Validation Step: Checking for Duplicate Table Declarations in the ODS Project 44
8.2 Azure DevOps Pipeline Overview 45
8.3 Deployment Validation Across Preproduction (PRE) and Production (PROD) Environments 49

9.1 Power BI Report: Overview of Data Discrepancies 52
9.2 Power BI Report: Log Analysis of Workflow Executions 55

10.1 High-Level Workflow of Reconciliation and Discrepancy Fixing . . . 58

11.1 Dynamic Ingestion Pipeline 62

Capitolo 1

Introduction

1.1 Goal

As part of my thesis at MSC Technology Italia, I had the opportunity to contribute to a large-scale data integration initiative aimed at modernising and improving the efficiency of data ingestion processes across multiple business domains (Finance, Commercial,Logistics,Liners, CRM, Operation etc.).

The main objective of this thesis is to develop and optimise automated data flow mechanisms capable of handling large-scale multi-domain ingestion, including domains such as finance, CRM, logistics, operations and cruise ships. These pipelines must guarantee high performance, reliability and scalability. By leveraging modern data ingestion tools, orchestration platforms and continuous deployment practices, the project aims to reduce manual intervention, minimise operational risk and increase confidence in data-driven reporting.

Specifically, the project focuses on:

- Establishing a seamless ingestion process using PowerCenter, Azure Synapse, and Microsoft Fabric.
- Automating workflow orchestration via Automic for scheduling and execution control.
- Implementing a discrepancy detection and reconciliation layer using Dagster to ensure data consistency between source and target systems.
- Integrating CI/CD pipelines through Azure DevOps to streamline deployment and maintain version control of data workflows.
- Providing visualizzation report using Power BI

The ultimate goal is to deliver a scalable, maintainable, and production-ready data framework that supports operational analytics and business intelligence at MSC Technology Italia, with Power BI serving as the primary data visualization layer.

1.2 Thesis structure

Initially, I worked on data ingestion using **Informatica PowerCenter**, which is an ETL (Extract, Transform, Load) tool. This phase involved transferring data from various source systems into a centralized **Operational Data Store (ODS)** hosted on SQL Server. PowerCenter allows me to use Change Data Capture (CDC) mechanisms and build robust workflows for tracking source changes and ensuring up-to-date data synchronization across different environments, such as Development, Test, Preproduction, and Production.

However, as data volumes grew and business requirements evolved, the need for a more scalable and flexible solution became apparent. Therefore, I transitioned to working with **Azure Synapse Analytics**. In this phase, I designed dynamic and parameterized pipelines to ingest data incrementally from Oracle databases into the ODS. I implemented a **watermark-based incremental loading mechanism**, which enabled efficient identification and extraction of newly updated data only, significantly reducing processing time and system load. Azure Synapse offered a more cloud-native approach, providing better integration capabilities, improved scalability, and easier management of complex data flows.

With the continuous evolution of cloud technologies and the increasing need for unified data platforms, MSC Technology Italia started migrating towards **Microsoft Fabric**. This platform allowed me to take a step further by leveraging **Lakehouse architecture** and **Delta tables**. I implemented a **watermark approach combined with a CDC logic** in Fabric pipelines, leveraging the flexibility of notebooks and PySpark alongside structured copy activities and scripts. The pipelines are fully parameterized to ensure reusability across multiple databases and tables, providing a scalable, cost-efficient, and integrated solution for data ingestion.

This thesis report details the methodologies, tools, and technologies I used to achieve a robust, flexible, and scalable data integration ecosystem. It highlights the evolution from traditional ETL approaches to cloud-native, unified data platforms, illustrating how each technological shift contributed to improving data management at MSC Technology Italia.

Capitolo 2

Technologies

This chapter introduces the main tools, platforms, and frameworks used to build a fully automated data workflows at MSC Technology Italia. The goal is to explain the roles and interactions of each component within the overall architecture and to provide context for the implementation described in later chapters.

2.1 Overview of Data Engineering Concepts

At MSC, the Business Intelligence architecture team relies on data engineering to collect, process, and deliver clean, usable data to multiple business domains. A robust data pipeline must handle:

Data ingestion from heterogeneous sources (e.g., operational databases, APIs, flat files)

Workflow orchestration to manage dependencies and execution order

Deployment automation to ensure reproducibility and scalability

Data reconciliation to ensure consistency and trustworthiness

Visualization and reporting to communicate insights

Automation, data quality, and DevOps practices are now integral to these processes.

2.2 Key Technologies Used

2.2.1 Informatica PowerCenter

Informatica is a well-known software company. It offers many tools for enterprise-level data management. These include data integration, data quality, and master data management.

In this case, we will focus on Informatica PowerCenter, which is Informatica's main product for ETL (Extract, Transform, Load) processes.

PowerCenter helps users:

Extract data from different sources,

Transform the data based on business needs,

Load the data into a target system like a data warehouse (DWH) or operational data store (ODS).

PowerCenter is widely used for:

Data integration,

Data migration,

Data governance,

And managing data warehouses.

2.2.2 Azure Synapse Analytics

Azure Synapse Analytics is a data platform from Microsoft. It is used for data warehousing, big data processing, and data integration. It also allows you to explore logs and time series data using a tool called Azure Data Explorer.

You can bring data from many sources into a lakehouse (a mix of data lake and data warehouse). Once the data is in the lakehouse, you can choose how to work with it using SQL, Spark, or Data Explorer. You can do everything in one place using Azure Synapse Studio.

With Synapse, you can manage data integration, monitor your system, and apply security rules all in one platform.

Benefits of Azure Synapse Analytics

- Many powerful features with just a few clicks
- Flexible and works well with other tools
- One user interface to manage everything

Security, governance, and admin features are all in one place. They work together smoothly. Synapse also works well with Power BI. You can use Microsoft Information Protection labels to secure your data. These labels stay with the data from Synapse to Power BI—even when exporting to Excel, which many users like.

Microsoft has been a trusted leader in this field for many years. That's why Synapse is a strong platform choice. It connects all the parts from the lakehouse to data analytics and reporting, and even machine learning. It brings the full power of Azure and Microsoft 365 into one powerful tool.

2.2.3 Microsoft Fabric

Microsoft Fabric is a unified data analytics platform. It offers one product, one user experience, one architecture, and one business model. All the data in your organization is stored in one central place a SaaS data lake.

The data is saved in an open format. This means you can use the same data to train machine learning models, build reports, run SQL queries, and use it in the data warehouse.

Microsoft Fabric brings all the tools you need into one place. You can:

Create data pipelines to move data,

Train machine learning models,

Build semantic models to define key business metrics,

And much more.

For business users, Fabric connects data to Microsoft 365. This helps teams work together and do quick data analysis easily.

Security, governance, and compliance are built into the platform. With Copilot, the AI assistant, users can be more productive. It can help write SQL, build reports, and create automated workflows based on events.

In short: all your data, all your teams, all in one place.

Microsoft Fabric covers many areas like data engineering, data science, and data governance. It brings everything together in one platform.

Compared to Azure Synapse Analytics, Microsoft Fabric has many updates and improvements. The design is cleaner, and the experience is much better overall.

2.2.4 Azure DevOps

Azure DevOps is a comprehensive suite of development tools and services from Microsoft that enables teams to plan, develop, test, deliver, and monitor software efficiently. It supports the full software development lifecycle and encourages collaboration, automation, and continuous delivery.
empowers development teams to deliver high-quality software faster and more reliably, through automation, collaboration, and efficient project tracking.

2.2.5 Automic Automation

Azure DevOps is a complete platform that helps turn an idea into a working software product.

It provides all the tools you need for software development:

You can plan projects using agile tools,

Manage your source code with Git,

Create and run test plans from the web,

Deploy your apps using a powerful, cross-platform CI/CD system.

Azure DevOps also gives full traceability and clear visibility across all development steps. This helps teams stay organized and work better together.

2.2.6 Power BI

Power BI is a business intelligence tool used to create dashboards and reports. It is tightly integrated with Microsoft Fabric and Azure Synapse. In this project, Power BI is used to visualize the results of discrepancies checks.

2.2.7 Dagster

Dagster is a modern data orchestrator specifically designed for building data-aware pipelines. It supports modular pipeline design, type-checking, observability, and testability. In this thesis, Dagster is used for automating the data reconciliation process: detecting and fixing discrepancies between source and target systems.

Dagster offers several advantages that made it an ideal choice for this project:

- **Strong Typing and Validation**: Dagster enforces type checking and schema validation at the operator level, ensuring higher reliability and better debugging.
- **Composable Workflows**: Its graph-based pipeline design allows easy composition of modular workflows, facilitating the reuse and scaling of reconciliation jobs across different datasets.
- **Integrated Monitoring and Logging**: Dagster's native observability tools (Dagit UI) provide fine-grained insights into each step of the process.
- **Flexible Deployment**: Dagster can be deployed on Kubernetes, Docker, or serverless environments, offering flexibility depending on operational requirements.

2.3 Integration Strategy

The tools above are combined into a cohesive ecosystem:

- PowerCenter, Azure Synapse, and Fabric perform ingestion and transformation.
- Automic and Dagster orchestrate and automate workflows.
- Azure DevOps manages CI/CD and version control.
- Power BI delivers final insights and alerts to business users.

Each tool plays a specific role in ensuring that the data pipeline is automated, reliable, and maintainable. In the next chapter, we will explore the design and implementation of the ingestion processes in more detail.

Capitolo 3

Integrating Data with Informatica PowerCenter

3.1 Goal

In this chapter, I explain how I integrated a new data source called OVHBMS, which runs on Microsoft SQL Server, into our target database: the Operational Data Store (ODS).

My main goal was to design a reliable and scalable process for loading data from several source tables into the ODS. To do that, I used a technique called Change Data Capture (CDC), which helps track and load only new or modified records. Although this started with tables from the BunkerManagementSystem database, I made sure the approach could also support future ingestion projects within MSC.

I will walk through the tools and steps I used, explain the design decisions I made, and share how I ensured that the data remained consistent and trustworthy during the transfer.

3.2 Understanding the Data and Architecture

3.2.1 Source and Target Overview

The BunkerManagementSystem database contains important business data like market indexes, product orders, and contract details. Each type of data is stored in a specific schema. The tables requested for ingestion were:

- `market.MarketIndex`
- `market.MarketIndexValue`

- `order.ProductOrder`
- `order.ProductOrderLine`
- `contract.PurchaseContract`
- `contract.PurchaseContractLocation`
- `contract.PurchaseContractIndex`

This data needed to be moved to the ODS which is used by all MSC Business Intelligence teams. The ODS is present in several environments: Development (DEV), Testing (TEST), Preproduction (PRE), and Production (PROD) all running on Microsoft SQL Server.

To keep things organized and aligned with the source structure, I created a new schema named `BunkerManagementSystem` in the ODS. This mirrored schema made it easier to trace data from source to destination and ensured consistent integration.

3.2.2 Loading Frequency and Volume Management

Because the amount of data is not quite large, I decided to schedule data loads once a day instead of every four hours or in real time. This frequency allowed us to keep the data in the ODS up to date every day.

I applied a `TOP clause` to limit each load to 50,000 rows instead of 500,000 or many more. This helped prevent performance issues and avoid overwhelming the network or database during the ETL process. It is a tradeoff between data freshness and system stability.

3.3 Setting Up Tables and Workflows

3.3.1 Getting Access and Preparing the Systems

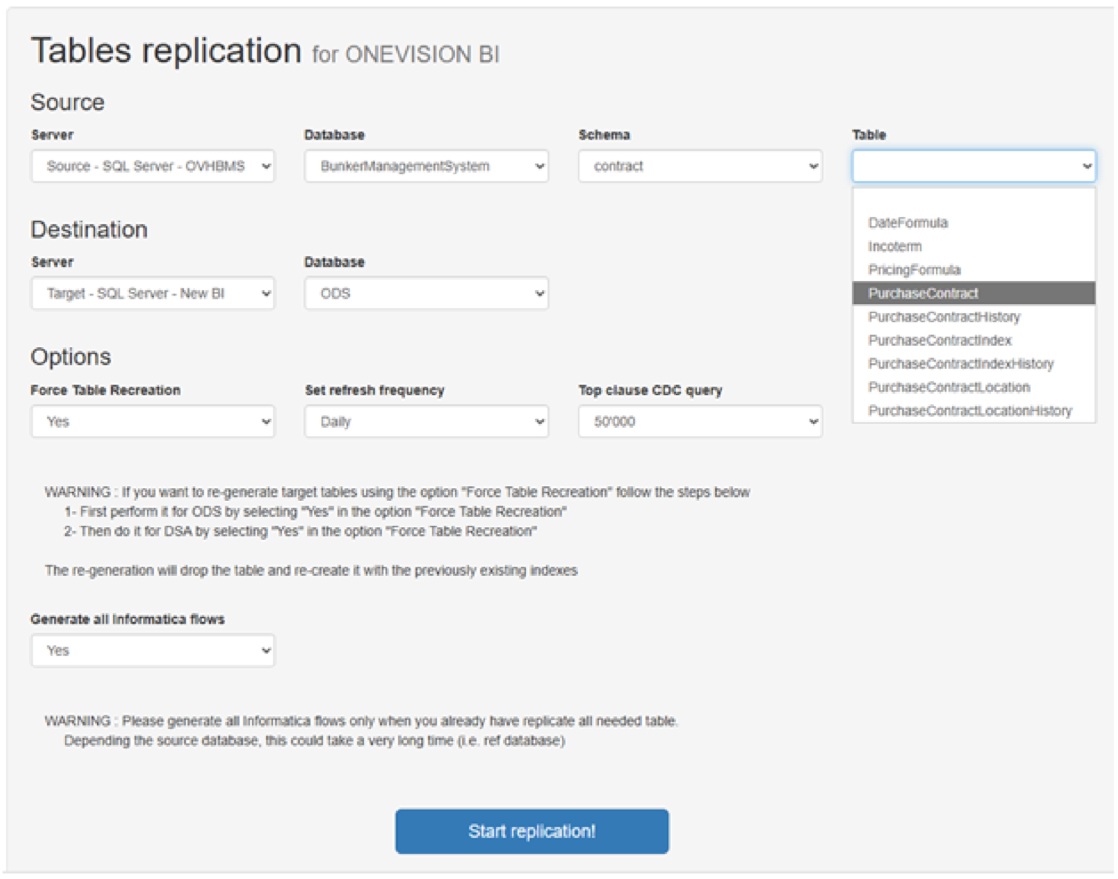
Before I could start building workflows, I submitted three important support tickets:

- I requested access to the OVHBMS source database across all environments (DEV, TEST, PRE, PROD).
- I asked for CDC to be enabled on the source tables. This was essential for tracking data changes.
- I also requested new relational connections in Informatica PowerCenter to allow communication between the source and the ODS.

Taking care of these early helped avoid issues later in the project.

3.3.2 Using CDC and Automating Workflow Creation

Once CDC was up and running, I used the CDC Generator tool (see Figure 3.1) to automatically create the ingestion workflows. I set the batch size to 50,000 rows and scheduled the updates to happen daily.



Tables replication for ONEVISION BI

Source

Server	Database	Schema	Table
Source - SQL Server - OVHBMS	BunkerManagementSystem	contract	

Destination

Server	Database
Target - SQL Server - New BI	ODS

Options

Force Table Recreation	Set refresh frequency	Top clause CDC query
Yes	Daily	50'000

WARNING : If you want to re-generate target tables using the option "Force Table Recreation" follow the steps below
1- First perform it for ODS by selecting "Yes" in the option "Force Table Recreation"
2- Then do it for DSA by selecting "Yes" in the option "Force Table Recreation"

The re-generation will drop the table and re-create it with the previously existing indexes

Generate all Informatica flows: Yes

WARNING : Please generate all Informatica flows only when you already have replicate all needed table.
Depending the source database, this could take a very long time (i.e. ref database)

Figura 3.1: Change Data Capture Generator Tool

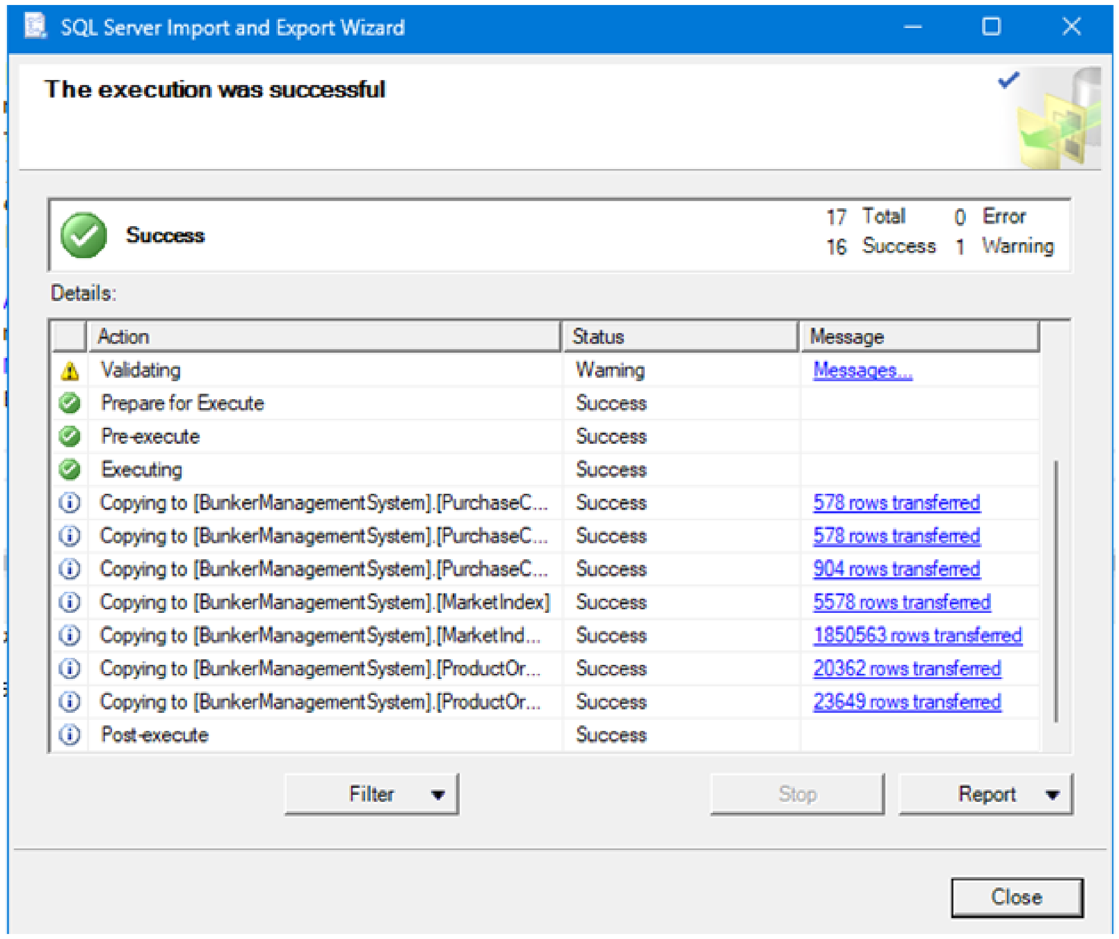
The CDC Generator handled the following tasks:

- Dropping any existing target tables that had the same name.
- Recreating the target tables to match the source schema.
- Generating Informatica workflows to manage the ETL process.

This automation saved time and ensured that the integration process was consistent and easy to maintain.

3.3.3 Initial Data Load

After setting up the workflows, I loaded historical data into the ODS using the SQL Server Import and Export Wizard.



The execution was successful

Success

17	Total	0	Error
16	Success	1	Warning

Details:

	Action	Status	Message
⚠	Validating	Warning	Messages...
✓	Prepare for Execute	Success	
✓	Pre-execute	Success	
✓	Executing	Success	
ⓘ	Copying to [BunkerManagementSystem].[PurchaseC...	Success	578 rows transferred
ⓘ	Copying to [BunkerManagementSystem].[PurchaseC...	Success	578 rows transferred
ⓘ	Copying to [BunkerManagementSystem].[PurchaseC...	Success	904 rows transferred
ⓘ	Copying to [BunkerManagementSystem].[MarketIndex]	Success	5578 rows transferred
ⓘ	Copying to [BunkerManagementSystem].[MarketInd...	Success	1850563 rows transferred
ⓘ	Copying to [BunkerManagementSystem].[ProductOr...	Success	20362 rows transferred
ⓘ	Copying to [BunkerManagementSystem].[ProductOr...	Success	23649 rows transferred
ⓘ	Post-execute	Success	

Figura 3.2: SQL Server Import and Export Wizard

I mapped each source table to its corresponding target table and launched the data transfer. Here's a summary of what was loaded:

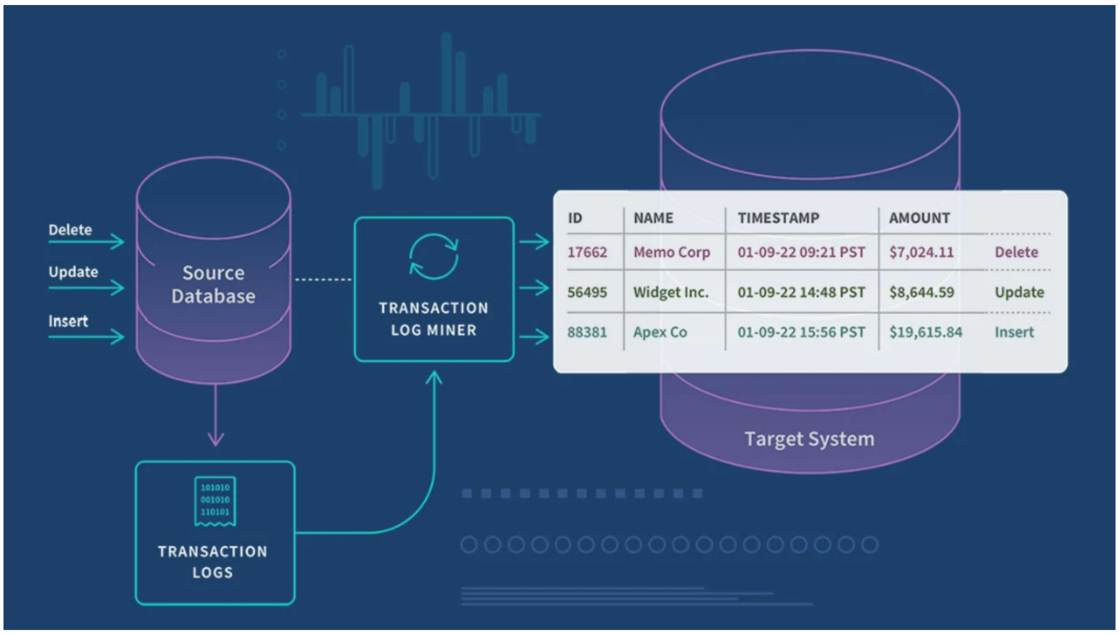
- `PurchaseContract`: 578 rows
- `PurchaseContractLocation`: 578 rows
- `PurchaseContractIndex`: 904 rows
- `MarketIndex`: 5,578 rows
- `MarketIndexHistory`: 1,850,563 rows

- `ProductOrder`: 20,362 rows
- `ProductOrderLine`: 23,649 rows

This full load gave us a solid historical base to work from before switching to daily incremental updates.

3.4 Tracking Changes and Incremental Loading with PowerCenter

After the full load was complete, I needed a way to keep the target tables up to date without reloading everything. That's where CDC really proved useful. It tracks every insert, update, or delete in the source database, and PowerCenter lets me extract and apply these changes efficiently.



ID	NAME	TIMESTAMP	AMOUNT	
17662	Memo Corp	01-09-22 09:21 PST	$7,024.11	Delete
56495	Widget Inc.	01-09-22 14:48 PST	$8,644.59	Update
88381	Apex Co	01-09-22 15:56 PST	$19,615.84	Insert

Figura 3.3: Change Data Capture Mechanism

3.4.1 Designing the Mappings

Each mapping pulls only changed records from the CDC tables. These records include a special field called `__ $operation` that tells me what kind of change happened:

- 1 = Delete
- 2 = Insert
- 3 = Before Update
- 4 = After Update

In the target tables, I added three technical fields:

- `TEC_IsDeleted`
- `TEC_CreatedDate`
- `TEC_LastUpdatedDate`

I filtered out unnecessary records (like operation 3), then applied the proper action depending on the operation type, inserting new rows or updating existing ones.

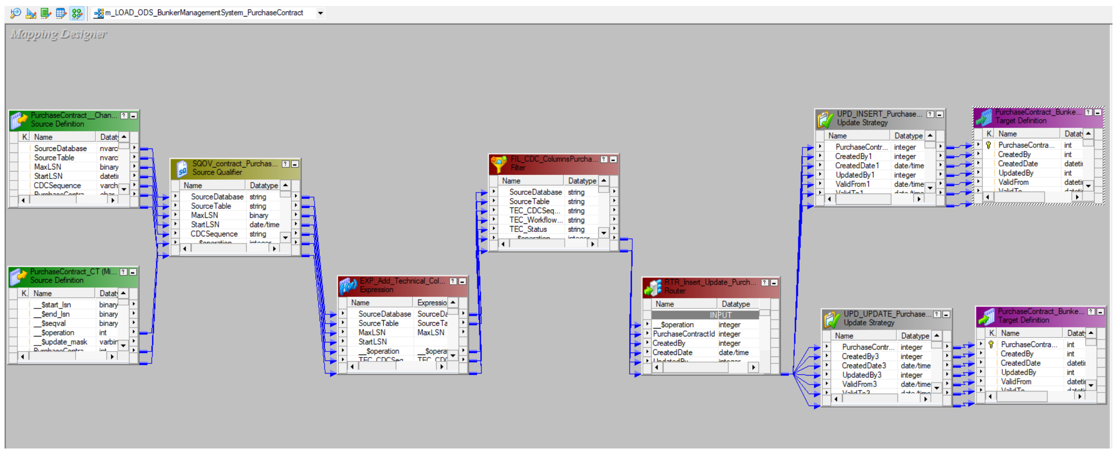

m_LOAD_ODS_BunkerManagementSystem_PurchaseContract
Mapping Designer
PurchaseContract__Chan... Source Definition
PurchaseContract_CT (Mi... Source Definition
SQOV_contract_Purchas... Source Qualifier
EXP_Add_Technical_Col... Expression
FIL_CDC_ColumnsPurcha... Filter
RTR_Insert_Update_Purch... Router
UPD_INSERT_Purchase... Update Strategy
UPD_UPDATE_Purchase... Update Strategy
PurchaseContract_Bunke... Target Definition
PurchaseContract_Bunke... Target Definition

Figura 3.4: CDC Mapping for BunkerManagementSystem Tables

3.4.2 Managing the Workflow

The workflow, `wf_ODS_CDC_Loading_BunkerManagementSystem_Daily_1`, consolidates all mappings pertinent to the **BunkerManagementSystem** tables. Given the current scope, which encompasses fewer than 15 tables, a single workflow is sufficient to manage the daily CDC-based data refresh operations. The data ingestion pipeline is now fully configured to refresh data on a daily basis, thereby ensuring that the ODS consistently reflects the latest information available from the source OVHBMS.

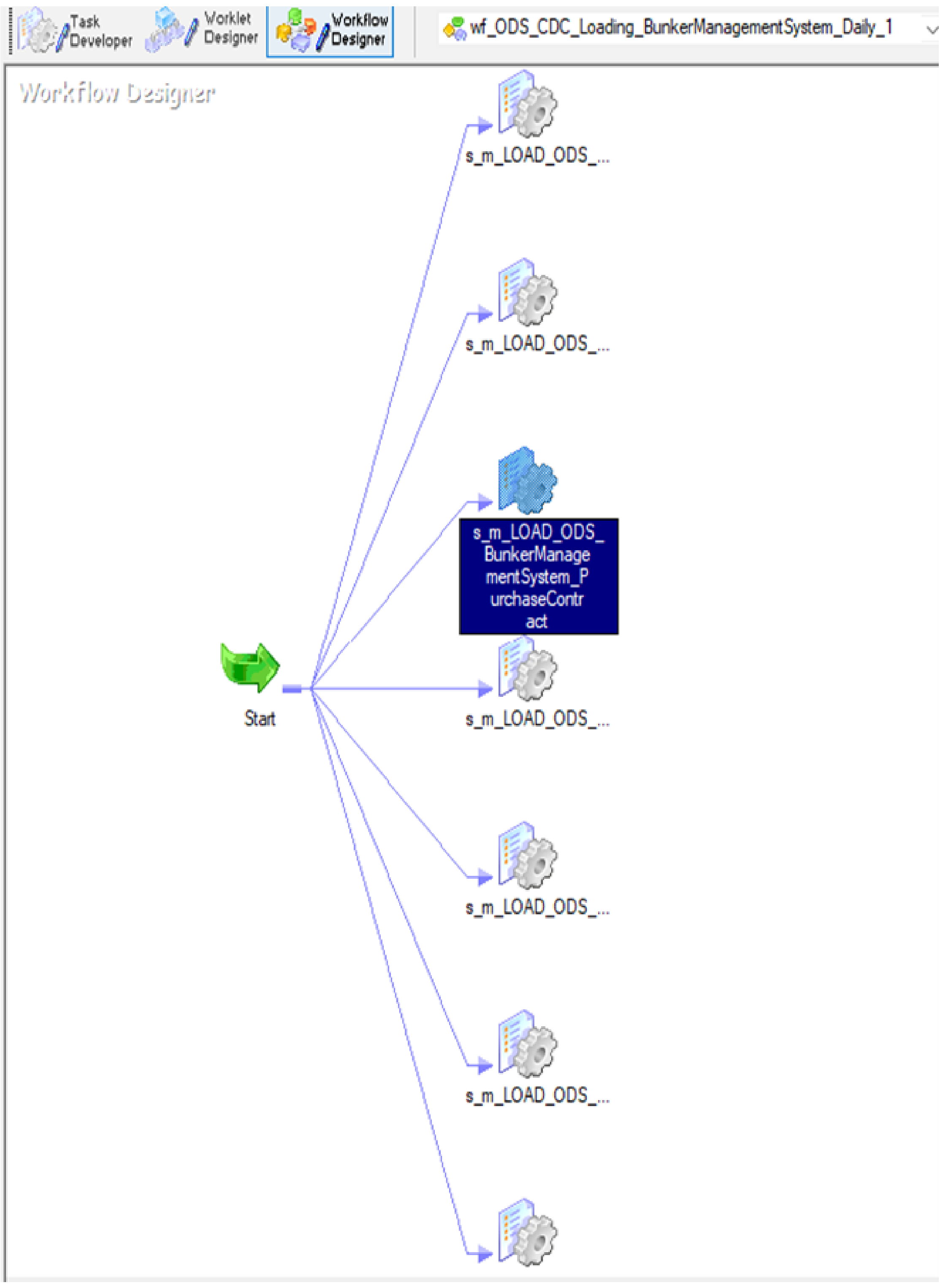


Figura 3.5: BunkerManagementSystem Daily Workflow

The ingestion process implemented with Informatica PowerCenter provided a strong foundation for integrating data from the OVHBMS source into the ODS. However, as MSC technology needs expanded new source from another relational database management systems as Oracle which needed to be ingested with more flexibility and at scale, it became necessary to explore more modern, cloud-native tools.

This led to the adoption of Azure Synapse Pipelines, a solution that supports dynamic configurations, simplified maintenance, and seamless integration with the broader Azure ecosystem. The next chapter presents in detail how I designed and implemented these pipelines to meet the evolving requirements of the organization.

Capitolo 4

Ingesting Data Using Azure Synapse

4.1 Introduction

In this chapter, I present the design and implementation of a custom ETL pipeline that I developed using **Azure Synapse** to ingest critical business data from the `IBOX_TN` Oracle database into a centralized **SQL Server Operational Data Store (ODS)**.

The solution I built is tailored to address essential requirements such as scalability, reusability, and data integrity. I designed it to operate seamlessly across different environments Development (DEV), Testing (TEST), Preproduction (PREPROD), and Production (PROD) while processing key business tables such as `BL_Containers`, `BL_Details`, `Customers_Master`, and `Voyage`.

By combining dynamic parameterization and an incremental loading mechanism based on watermarking, I ensured that the pipeline delivers accurate, up-to-date data while minimizing system load. In the following lines, I will outline the technical architecture, design choices, and practical benefits of this cloud-native ingestion process.

4.2 How I Designed the Pipelines

I built the pipelines with two key goals in mind:

- **Load only new or updated data** to save resources and improve performance.
- **Make them reusable and adaptable** across different tables and environments.

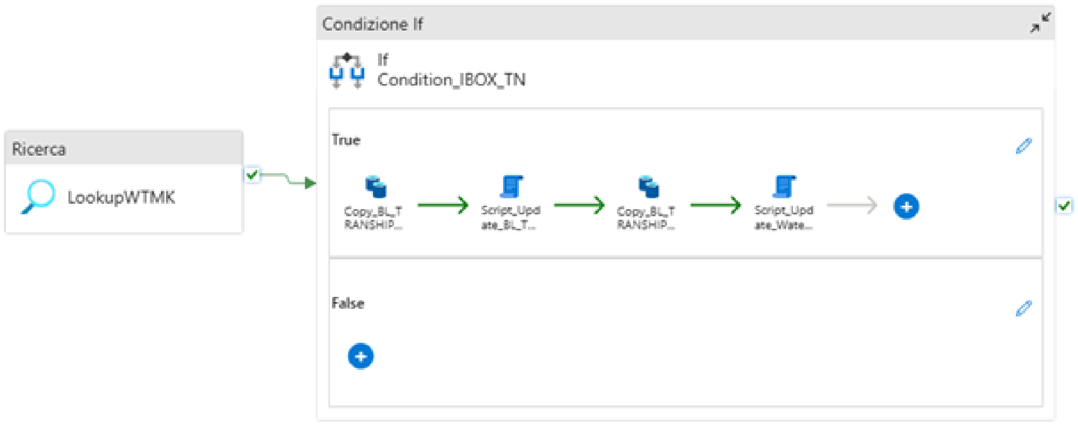


Figura 4.1: IBOX$_T$*Npipeline*

4.3 Step-by-Step Pipeline Logic

Here's how the pipeline was designed and how it works step by step:

Step 1: extract the max Watermark date

To begin, I created a dedicated watermark table named `WTMK_IBOX_TN` in the ODS environment across all stages DEV, TEST, PREPROD, and PROD. I initialized it with a default date value of `1900-01-01` to ensure a controlled starting point for the first execution.

For the development work, I created a dedicated branch in the Azure Synapse DEV workspace named `IBOX_TN_ingestion`. This branch was based on the production workspace to maintain the production state while isolating my changes.

I then created a new pipeline named `IBOX_TN_incremental_daily`, which begins with a **Lookup** activity. This activity queries the `WTMK_IBOX_TN` table to retrieve the latest successful load date for a given table. This retrieved date serves as the lower boundary for the incremental data extraction.

Step 2: Check for New Data

To avoid unnecessary processing, I used an **If Condition** activity that compares the retrieved watermark date with the current date.

- If the watermark date is equal to today's date, it means the data is already up to date, and the pipeline terminates early.
- If the watermark date is earlier than today, it means new or updated data exists, and the pipeline proceeds with the ingestion steps.

When the condition is True, I proceed with the folowing steps

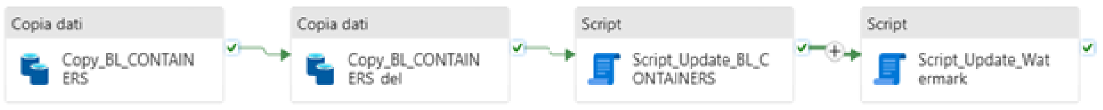


Figura 4.2: Incremental load Pipeline

Step 3: Load New or Updated Records

Inside the `If True` branch, I added a **Copy activity** to extract and load incremental data from the Oracle source into the ODS. The logic includes:

- A dynamic SQL query that fetches only records modified between the last watermark date and the current date.
- The addition of three technical columns to the target ODS tables, in order to track changes and maintain historical data:
  - `TEC_IsDeleted`: initialized to `0`, indicating active records.
  - `TEC_CreatedDate`: stores the date when the record was first loaded.
  - `TEC_LastUpdatedDate`: records the most recent update timestamp based on primary key changes.

Step 4: Load Deleted Records into a Working Table

In parallel, I implemented a second **Copy activity** to extract records marked as deleted from the Oracle source. These records are loaded into a dedicated staging table located in a separate database named `WORKING`. This staging area temporarily stores deleted rows for post-processing.

Step 5: Apply Merge and Soft Delete Logic

Once both actual and deleted data are loaded, I used a **Script activity** to update the corresponding records in the ODS.

For each primary key found in the delete staging table:

- I set the `TEC_IsDeleted` field to `1`, marking the record as deleted.
- I also updated the `TEC_LastUpdatedDate` to reflect the deletion timestamp.

I uuse thus strategy in order to retain historical data instead of physically removing rows, which is essential for auditability and data lineage tracking.

Step 6: Update the Watermark

To complete the process, I executed another **Script activity** that inserts the current timestamp into the `WTMK_IBOX_TN` watermark table. This ensures that the next execution of the pipeline will only process changes that occurred after the current run, maintaining consistency across all future loads.

4.4 Making Pipelines Reusable Across Environments

One of the strengths of Azure synapse pipeline is its high level of **parameterization**. I used parameters for:

- Oracle and SQL Server connection details (host, port, service name, username)
- Secure credentials stored in Azure Key Vault
- Table names and dynamic queries

Thanks to this approach, I could easily deploy this pipeline in other environments (TEST, Preprod and PROD) just by changing parameter values no need to modify pipeline logic.

4.5 Benefits of this Pipeline Design

Here's what I gained from this architecture:

- **Performance**: Incremental loads meant less data was transferred each time.
- **Reliability**: The watermark system made sure I didn't miss or duplicate data.
- **Flexibility**: I could reuse the same logic for multiple tables and projects.
- **Cost-efficiency**: I avoided running the pipeline unnecessarily, saving compute time.

If a step fails, the orchestration stops. This prevents corrupt or partial data from spreading downstream.

4.6 Orchestration with the IBOX_TN Master Pipeline

To coordinate the execution of all individual `IBOX_TN` table ingestion pipelines, I designed a master orchestration pipeline named `IBOX_TN_Overall_Orchestration`.

Within this orchestration pipeline, I used multiple `ExecutePipeline` activities to invoke each table-specific ingestion pipeline. For each call, I passed a set of dynamic parameters, including:

- Oracle and SQL Server connection settings
- Secret names used for secure credential retrieval from Azure Key Vault

The execution flow is fully sequential: each sub-pipeline is triggered in the correct order and only starts once the previous one has successfully completed. This ensures data consistency across dependent tables and facilitates easier error tracking and operational monitoring.

Why It's Useful

- It gives me one central place to manage the ingestion process.
- It's easy to add or remove tables as business needs change.
- It works across environments with no code duplication.

Overall, Synapse offered a more modern and flexible solution that aligned with MSC Technology Italia's move toward the cloud. It allowed me to build a robust, scalable ingestion pipeline that is future-proof and well-integrated with the rest of the company's data platform.

Capitolo 5

Modern Ingestion with Microsoft Fabric

5.1 Goal

In a constantly changing world, **Microsoft** keeps improving its products to meet customer needs. That's how **Microsoft Fabric** was created an all-in-one data analytics platform. It helps bring together data collection, processing, storage, and analysis in one place. It uses **OneLake**, a unified storage system designed to gather data from different sources in the cloud.

Since MSC mainly works with Microsoft technologies, the company naturally started moving gradually to the Microsoft Fabric ecosystem. This smart choice offers many benefits: better scalability, built-in cloud management, advanced analytics, and lower costs.

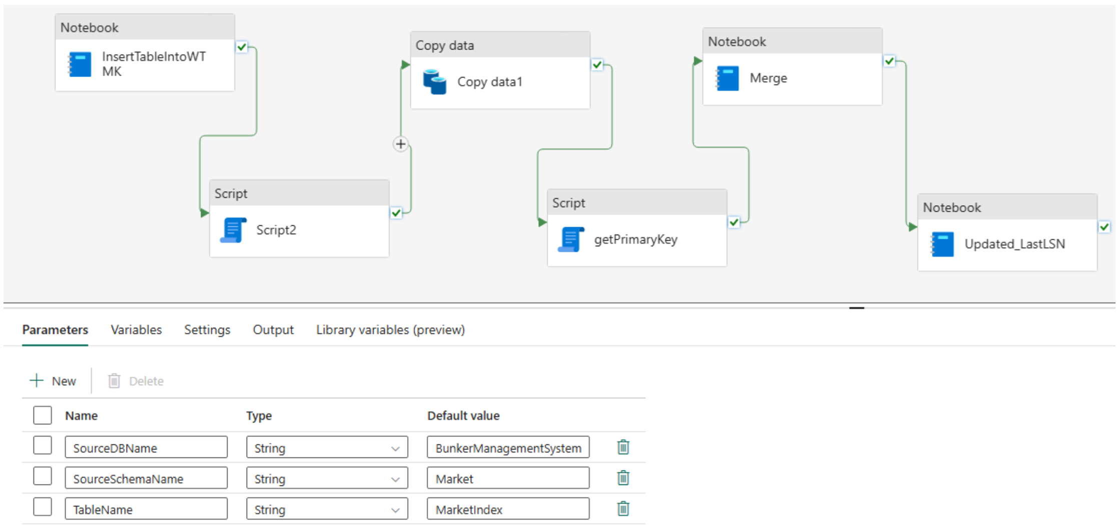
In this context, the goal was to design and build a more flexible, reusable, and scalable data pipeline. This pipeline can handle large amounts of different types of data. It collects data from many sources, using a method similar to the classic **Change Data Capture (CDC)**, but adapted for the cloud. It is based on modern concepts such as **watermarks** to manage data flows, **notebooks** to run processing tasks with **PySpark**, Delta tables to ensure data reliability and traceability, and unified storage in **OneLake**. All of this is part of a **Lakehouse** architecture, which combines the flexibility of a Data Lake with the structure of a data warehouse.

5.2 Pipeline Overview

The pipeline is fully parameterized, allowing dynamic adjustment for:

- **Source Database Name**

- **Source Schema Name**
- **Table Name**



Name	Type	Default value
SourceDBName	String	BunkerManagementSystem
SourceSchemaName	String	Market
TableName	String	MarketIndex

Figura 5.1: Dynamic Ingestion Pipeline

This design ensures high reusability, as the same pipeline can be invoked for different datasets without requiring manual modifications.
This pipeline executes a series of operations to process daily data loads, aiming to identify and apply changes from the source table to the target. To ensure conformity, I utilized the Delta Lake format for efficient storage and change tracking.

The main steps include

- **InsertTableIntoWTMK:** A Notebook that manages and retrieves the latest update timestamp (`LastLSNDate`) using a Delta Lake table as a watermark store.
  - If the watermark table exists, it loads and verifies whether an entry for the specified `TableName` is present.
  - If the watermark table exists but lacks the `TableName`, it appends an initial record.
  - If the table does not exist, it creates a new `WatermarkTable` in the Data Lake.

  This Notebook extracts the maximum `LastLSNDate` and returns it as an output.

- **List__of__columns:** An SQL script that dynamically conructs and retrieves a list of columns from the specified source table, excluding system-generated columns, in order to use it in the CDC query.

- **getPrimaryKey:** Another SQL Script that retrieves the primary key columns of the specified table within the database schema.

- **Copy data:** A Copy Activity that transfers the changed data from SQL Server to the Lakehouse in Parquet format. The SQL query is dynamically generated to fetch only changes using CDC logic.

- **Merge:** A Notebook that performs the merge operation on the copied CDC records, handling updates, deletions, and insertions into the Delta table.

- **Updated__LastLSN:** A Notebook that updates the `Watermark_CDC Delta table` with the latest captured `LastLSNDate`, ensuring the incremental load pipeline is ready for the next run.

5.3 Failure Notification Mechanism

As part of my efforts to improve the **observability** and **reliability** of our data integration workflows, I designed and implemented a **failure alert mechanism**. This system is based on two interconnected pipelines: a main pipeline, called **MasterPipeline**, which runs the critical ingestion workflows, and a second pipeline, **NotifyTeamsChannelPipeline**, which is triggered in case of failure.

The main goal of this alert system is to **automatically notify** the different BI teams (Finance, Commercial, Liners, Operation, CRM, Architecture, etc.) via a dedicated**Microsoft Teams** channel that I created whenever one or more pipelines responsible for ingesting their respective databases fail. This helps reduce the time between the appearance of the issue and its resolution. It eliminates the need for **manual checks** of the workflow status and greatly improves **data governance**.

Throughout the design of these pipelines, I relied on the **official Microsoft documentation** to implement this alert mechanism effectively.

5.3.1 Notify Teams Channel Pipeline

I used the **NotifyTeamsChannelPipeline** to structure the message sent to the Teams channel, which I named **Failure Notification**. The goal is to automatically alert the BI teams in case of a failure in a pipeline responsible for updating or loading the data they manage.

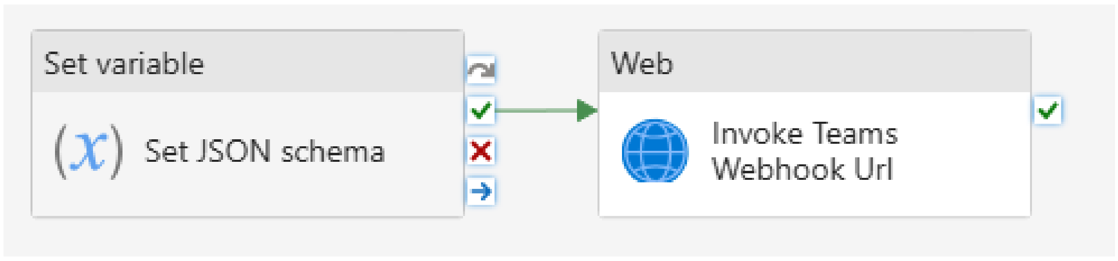


Figura 5.2: Teams Notification Pipeline

5.3.2 Pipeline Structure and Logic

The core logic of the **NotifyTeamsChannelPipeline** is composed of two main steps:

- **MessageCard JSON Schema:** I first initialize a pipeline variable named `messageCard` containing a structured JSON schema that conforms to the Microsoft `MessageCard` schema, ensuring compatibility with Teams channels.

  The message card includes the following execution details:

  - **Pipeline Run ID:** A unique identifier for the pipeline execution.
  - **Pipeline Name:** The name of the pipeline that encountered the failure.
  - **Status:** Execution status of the pipeline (e.g., `Failed`).
  - **Start and End Time (UTC):** Timestamps marking the beginning and end of the execution.
  - **Execution Duration:** Total time taken by the pipeline before the failure.
  - **Error Message:** The error message that helps to understand why the pipeline failed.
  - **Workspace Name:** The environment or project within which the pipeline was executed.
  - **Notification Timestamp (UTC):** The exact time when the notification is sent.

- **Invoke Teams Webhook:** Once the message card is constructed, the pipeline calls a Microsoft Teams webhook URL (**Failure Notification**) that I pre-configured before using a `Web Activity`. The webhook sends the structured message, which results in an immediate notification being posted into the designated Teams channel.

In the next section, I will describe how the `MasterPipeline` orchestrates data ingestion flows and integrates failure detection logic to seamlessly trigger the `NotifyTeamsChannelPipeline` when necessary.

5.3.3 MasterPipeline

I created a pipeline called **MasterPipeline**. In this pipeline, I use two **Invoke Pipeline** activities. The first one is used to call any pipeline for which we want to receive a failure notification. In this specific case, I invoke the **Incremental_watermark_BunkerManagementSystem_pipeline**. I connected the second Invoke activity in case of the failure of the first one; it calls the **NotifyTeamsChannelPipeline**, which is responsible for sending the notification.

I constructed the **Teams MessageCard** dynamically in order to capture the execution details of the previous pipeline.

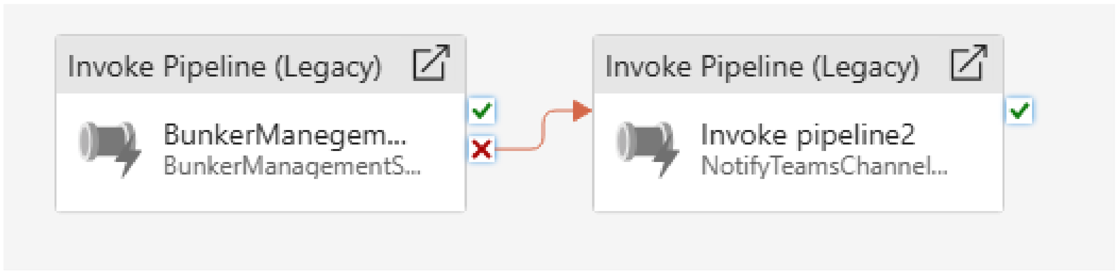


Figura 5.3: Master Pipeline with Error Handling

This pipeline is responsible for executing key data workflows and invoking the Teams notification pipeline if an error is detected. The structure consists of:

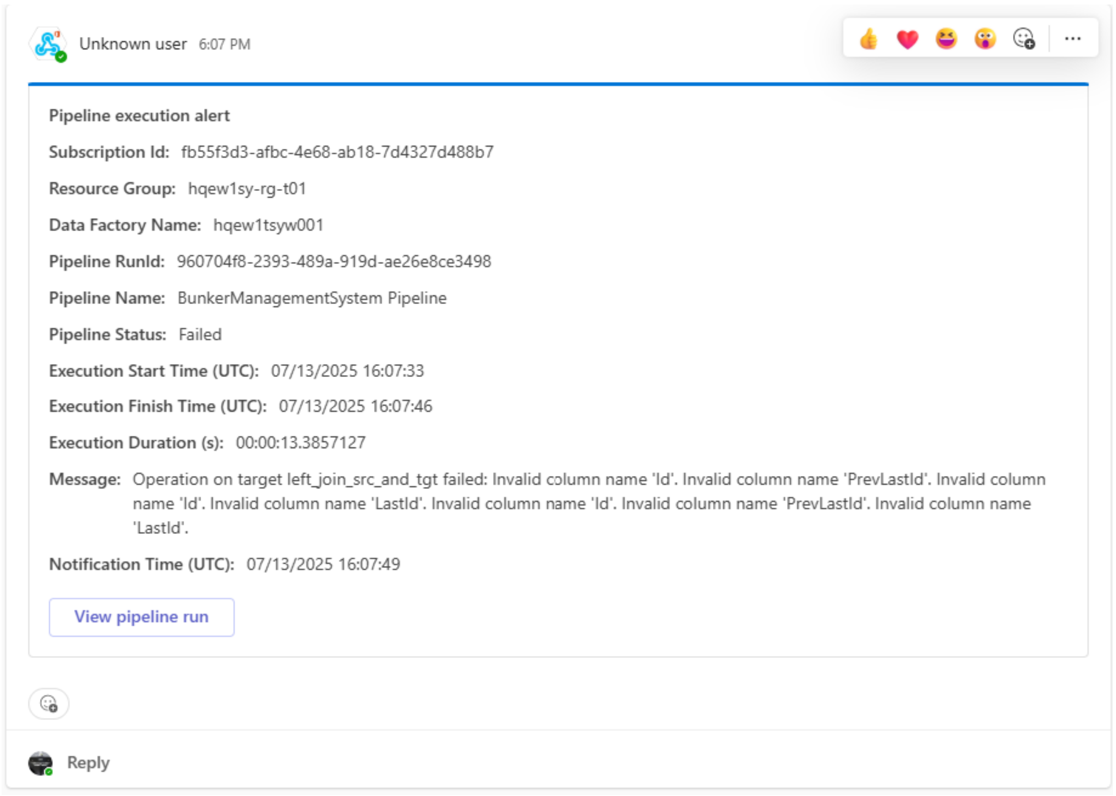
- **Primary Workflow Execution:** Runs the **BunkerManagementSystem Pipeline** data integration pipeline under controlled execution.
- **Conditional Failure Handling:** In the case of failure, the pipeline proceeds to trigger the **NotifyTeamsChannelPipeline**, passing necessary metadata for the alerting process.

This orchestrated approach ensures high reliability, improved observability, and timely notifications in case of critical failures during data processing.

5.3.4 Key Benefits

Through this modular and robust setup, I achieved several operational advantages:

- **Immediate Visibility into Failures:** guarantees that MSC BI teams are notified without delay when a critical data processing job fails, effectively reducing downtime and accelerating incident response.
- **Automated Alerting:** Eliminates manual intervention for monitoring and error notifications.



Pipeline execution alert

Subscription Id: fb55f3d3-afbc-4e68-ab18-7d4327d488b7

Resource Group: hqew1sy-rg-t01

Data Factory Name: hqew1tsyw001

Pipeline RunId: 960704f8-2393-489a-919d-ae26e8ce3498

Pipeline Name: BunkerManagementSystem Pipeline

Pipeline Status: Failed

Execution Start Time (UTC): 07/13/2025 16:07:33

Execution Finish Time (UTC): 07/13/2025 16:07:46

Execution Duration (s): 00:00:13.3857127

Message: Operation on target left_join_src_and_tgt failed: Invalid column name 'Id'. Invalid column name 'PrevLastId'. Invalid column name 'Id'. Invalid column name 'LastId'. Invalid column name 'Id'. Invalid column name 'PrevLastId'. Invalid column name 'LastId'.

Notification Time (UTC): 07/13/2025 16:07:49

Figura 5.4: Failure Notification in Microsoft Teams

- **Reusable Pipeline Logic:** The structure is generic and can be reused across various data flows by simply modifying pipeline references and parameters.
- **Structured and Actionable Alerts:** Alerts are formatted for readability and include all relevant execution metadata, making them actionable for operational teams.

Capitolo 6

Establishing the Real-Time Data Pipeline

To begin, I created a dedicated Lakehouse environment named **EH__ASBonOVA** in Microsoft Fabric. Within this lakehouse, I defined the following:

- **KQL Database: EH__ASBonOVA** used to store structured and semi-structured data.
- **KQL Queryset: EH__ASBonOVA__queryset** used to define and execute KQL queries, including transformation logic and update policies.

A Lakehouse is a modern data architecture that merges the scalability of data lakes with the performance and schema governance of data warehouses, making it ideal for managing semi-structured formats such as XML.

Kusto Query Language (KQL) is a read-optimized language developed by Microsoft for querying large datasets in real time. Its analytical capabilities make it particularly well-suited for event-based and structured data processing.

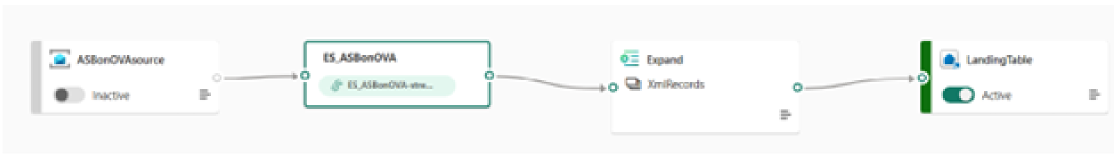


Figura 6.1: Full Eventstream pipeline from ASBonOVAsource through transformation and routing into Eventhouse bronze tables.

Data Source Integration

The ingestion pipeline begins with **ASBonOVAsource**, a simulated Azure Service Bus source configured to emit real-time messages. Each message contains a payload structured in XML, encapsulating various types of business events such as charges, equipment statuses, local BI updates etc... By activating this source, I ensured that Eventstream began receiving these messages in real time, forming the ingestion point of the pipeline.

Eventstream Processing

The centerpiece of the pipeline is the Eventstream named **ES_ASBonOVA**. which functions as the ingestion and transformation engine. It receives the raw XML payloads and applies logic to make the data suitable for downstream consumption.

I used an **Expand** transformation which targets the **XmlRecords** field, which contains a list of embedded XML fragments within each message. The Expand operation flattens this list into individual records, transforming the nested structure into a stream of XML events.

I created a destination table named **LandingTable** in the **Eventhouse** which act as a Bronze layer. In the following section, I describe how I implemented update policies to parse and promote this data into silver tables.

6.1 Design and Role of Silver Tables

Silver tables act as the intermediate layer in the medallion architecture. While bronze tables store raw and semi-structured XML data, silver tables are responsible for parsing and structuring it into query-ready formats. This step is crucial for Structuring unstructured XML.

I implemented this using **KQL update policies**, which automate the transformation of bronze data as it arrives.

6.1.1 Automated and Dynamic Update Policy

I implemented an automated and dynamic update policy for **charge table**. It is important to note that this code is designed to be generic and adaptable to any target table, provided that the source structure and column patterns are similar.

This flexibility is enabled by the use of dynamic schema extraction techniques, particularly through the use of functions such as**bag_unpack()** and **bag_remove_keys()**, which allow me to handle both known and unknown attributes. Specifically, the transformation logic does not depend on a fixed schema. Instead, it dynamically interprets the contents of XML records, extracts attribute-value pairs, and constructs a flexible structure where known fields are explicitly projected and unknown fields are safely isolated into a dynamic **Additional_columns** object. This ensures that any evolution in source data such as the addition of new attributes does not break the transformation process or require major code changes.

The use of this dynamic design allows me to reapply the same update policy logic to any other table in the data platform, simply by modifying the filter condition for XML tags and adjusting the list of known columns. As a case study, I filtered the charge data because it contains well-structured financial attributes that are relevant for billing and cost analysis. However, I could have just as easily applied this logic to other XML-driven datasets such as invoice, payment, or shipment, by changing the tag filter (e.g., <BILocalData instead of <charge) and adapting the column projection accordingly.

The underlying function, **ExtractFromLandingTable_charge()**, begins by expanding the array of XML records from the LandingTable, converting XML elements to strings, and filtering only those that match a specific pattern in this case, those beginning with <charge . After cleaning and reformatting the XML content into comma-separated key-value strings, I extract each attribute and pack them into a key-value bag. This bag is then unpacked into columns, grouped by entity, and deduplicated using **arg_max()** on the **LM_DT** timestamp to keep only the latest version of each record.

The solution remains highly scalable, with low-latency updates and built-in deduplication logic. One of the most valuable aspects of this solution is its adaptability. The logic for cleaning, parsing, and typing the data remains stable across different tables; only a few parameters like the tag name and the expected output columns need to be adapted. This modular design significantly reduces the effort required to onboard new data sources and supports future scalability as the data environment grows.
while I chose the charge table for demonstration purposes, this policy architecture is not limited to it. The methodology I used enables broad adaptability and consistent data quality enforcement across multiple tables in the silver layer, all while maintaining resilience to schema drift and unknown attributes.

```
.alter table Silver_charge policy update ‘‘‘[{"IsEnabled": true,"Source": "LandingTable","Query": "ExtractFromLandingTable_charge()","IsTransactional": true,"PropagateIngestionProperties": false}]‘‘‘

.create-or-alter function with (folder = "Policies") ExtractFromLandingTable_charge()
{
LandingTable
| mv-expand record = ArrayValue_XmlRecords
| extend XML = tostring(record.XMLData)
| where XML startswith "<charge "
| extend XML_Clean = replace_string( replace_string(XML, "<charge ", ""),"  />","")
| extend XML_Clean = replace_string(XML_Clean, "\" ", "\",")
| extend kv_pairs = split(XML_Clean, ",")
| mv-expand kv = kv_pairs
| extend kv_split = split(kv, "=")
| extend key = tostring(kv_split[0]), raw_value = tostring(kv_split[1])
| extend value = trim(’"’, raw_value)
| summarize kv_bag = make_bag(pack(key, value)) by XML,Agency_Id
| evaluate bag_unpack(kv_bag)
| extend Additional_columns = bag_remove_keys(parse_json(tostring(kv_bag)), KnownColumns)
| summarize arg_max(LM_DT, *) by Primary_Key_Id, Agency_Id
| project-away XML
| project tolong(Primary_Key_Id) ,
          todatetime(LM_DT)     ,
          tolong(Cargo_Shipment_id),
          tolong(Cargo_Shipment_Container_id),
          tolong(Cargo_id),
          tolong(Bill_Of_Lading_id),
          tolong(Auto_Chg_For_Charge_id),
          tostring(Charge_Definition),
          toint(Adjust_Curency_GB_DBID_PK),
          toint(Paid_At_Location_GB_DBId_PK),
          todecimal(Adjust_Param_Std),
          toreal(Adjust_Percentage),
          toint(Adjust_Number_of_Units),
          todynamic(Additional_colums)
}
```

Listing 6.1: KQL Update Policy for Silver_charge Table

6.1.2 Technical Considerations and Logic

- **mv-expand:** This command expands the array of XML records so that each fragment is processed individually.
- Upsert strategy: By using **Primary_Key_Id** and **LM_DT**, the silver table ensures that newer records overwrite older versions for the same ID, preserving only the most up-to-date view.
- **IsTransactional:** IsTransactional is set to true, ensuring atomic updates that maintain data integrity.

6.1.3 Benefits of This Approach

By leveraging KQL update policies:

- I eliminated the need for scheduled batch transformations.
- Real-time XML parsing occurs as soon as data lands in bronze.
- Silver tables provide clean, typed data that is immediately usable for BI dashboards and analytics.

6.1.4 Limitations and Considerations

Despite the strengths of using KQL update policies and Microsoft Fabric's Real-Time Intelligence tools for XML data processing, several limitations have emerged during implementation. These are important to acknowledge for future scalability and maintainability of the solution:

Lack of Function Parameterization and Reusability

The function is statically defined and tightly coupled to a specific XML schema and target table. It does not support the use of dynamic parameters, which limits its adaptability. As a result, the function cannot be reused across different silver tables or XML types without duplicating and manually modifying the logic. This lack of flexibility hinders generalization and increases maintenance overhead in environments dealing with multiple, heterogeneous data formats.

Schema Rigidity and Evolution

The current approach assumes a stable and predefined XML schema. If new fields are introduced or the structure changes, the update policy and extraction logic must be manually revised. There is no built-in schema discovery or dynamic field mapping, which limits the pipeline's ability to adapt to evolving business requirements.

6.1.5 Data Recovery Process After Schema Change

One critical limitation in working with static schemas and update policies in KQL is that newly added columns are not retroactively populated for historical records. Once a column such as **Result_Std** for example is introduced in the **Silver_charge** table, existing rows that were ingested prior to this schema change will have null values for that column. This can result in incomplete or misleading data when performing analytics.

To address this, I implemented a manual backfill and recovery process using KQL. The process consists of two key steps: deleting the incomplete records, and re-appending them with the newly added column properly extracted and populated.

```
.delete table Silver_charge records <|
Silver_charge
   | where isnull(Result_Std)
```

Listing 6.2: KQL Update Policy for Silver_charge Table

In this step, I first identify and delete all records in the **Silver_charge** table where the **Result_Std** column is null. This ensures that only outdated, incomplete records are removed and that no duplicate primary keys will exist after re-insertion.

```
.append Silver_charge <|
Silver_charge
    | where isnull(Result_Std)
    | join kind=inner (
        LandingTable
        ...
    ) on Primary_Key_Id, LM_DT
```

Listing 6.3: KQL Update Policy for Silver$_{c}hargeTable$

The second part re-ingests the relevant records by joining them with the raw data available in the LandingTable. The logic includes:

XML parsing: The raw XML string is cleaned using **replace_string()** and then exploded using string splitting logic to extract key-value pairs dynamically.

Bag unpacking: I use **make_bag()** and **bag_unpack()** to convert the extracted key-value pairs into a tabular format.

Aggregation: The **arg_max()** function ensures that only the most recent version of each record (based on `LM_DT`) is retained during the join.

Join operation: The re-parsed data is joined back with the incomplete records using `Primary_Key_Id` and `LM_DT` to maintain alignment.

Projection: Finally, I re-select all relevant columns—including the newly added column and append the corrected data back into the `Silver_charge` table.

This method not only restores missing values for new columns but also ensures data consistency and continuity without disrupting the existing pipeline logic.

In this chapter, I detailed the strategies and mechanisms I implemented to handle real-time ingestion, transformation, and schema evolution challenges within Microsoft Fabric. By leveraging KQL update policies, I was able to automate the structuring and upserting of XML data into silver tables, ensuring high data quality and near real-time visibility.

Additionally, I established a recovery process to retroactively populate new fields in existing records, maintaining data integrity without disrupting the operational flow. These practices not only strengthened the robustness of the pipeline but also laid a scalable foundation for future enhancements.

In the following chapter, I will extend this work by focusing on the implementation of analytical queries and real-time dashboards built on top of the silver layer, providing actionable insights into the ingested business events.

Capitolo 7

Workflow Automation and Scheduling

7.1 Execution of the Workflow via Automic Software Pre-Process Stage

To automate the execution of Informatica PowerCenter workflows (figure 3.5) and and Azure Synapse pipelines (figure 4.1), I configured a **JOBS** object inside **Automic Software**. The automation process starts with the **Pre-Process** tab, where I set up all the necessary information that the job will use before running the workflow itself.

7.1.1 Initialization of Variables

First, I initialized a set of important variables that will guide the execution:

- I set the `&parentnr` variable, which keeps track of the parent task number. This helps in organizing and tracing the job's execution.
- I used the `GET_PUBLISHED_VALUE` function to dynamically retrieve values such as:
  - `&DATABASE_NAME`: the name of the database where the workflows will run,
  - `&DATABASE_VARA_COLUMN_NUMBER`: a reference to the column in the variable object,
  - `&FOLDER_NAME`: the folder where the workflows are stored,
  - `&user_INFA`: the Informatica user running the workflows,
  - `&WAIT`: a setting for controlling wait times,

  - `&LOOPING`: which indicates whether the execution should loop or not.

By setting these variables dynamically, I made the job flexible and adaptable to different environments or databases without needing manual changes.

7.1.2 Retry Management

To prepare for possible errors during workflow execution, I initialized retry-related variables:

- `&HASFAILED` starts empty; it will later store any failure messages.
- `&number_of_failures` is set to `0` by default, tracking how many times the job might fail.
- `&MESSAGE` is prepared to capture any error messages if something goes wrong.

This way, I ensured that if something fails, the job can react appropriately and retry if needed.

7.1.3 Environment Initialization

Then, I set up the environment variables:

- If the integration service (`&IS`) is not forced, I set the variable `&VARA_NAME` to the environment-specific variable object `OVBINT.BI.CDC_ENVIRONMENT.VARA`.
- I retrieved the environment name using the `get_var` function, ensuring that the job is aware of whether it is running in DEV, TEST, PREPROD, or PROD.

This step made sure the job would adjust its behavior according to the environment, reducing the risk of misconfiguration.

7.1.4 Workflow List Retrieval

Finally, I prepared the list of workflows that needed to be executed:

- I assigned `&VARA_NAME` with the appropriate workflow variable object, either the normal daily workflow list or a looping list, depending on whether looping was enabled.
- I read the actual list of workflows using the `get_var` function and split them based on a separator (`;`).

With this setup, the job can process multiple workflows in one go, looping if necessary, or running just once if that's what's needed.

In this Pre-Process step, I focused on creating a flexible, environment-aware, and robust setup. By initializing everything properly at the start, I ensured the workflow execution would be smooth, with minimal risk of failure or manual intervention.

7.2 Post-Process Execution

After the main execution of the workflow, the job transitions into the **Post-Process** phase. In this stage, I implemented several important steps to ensure that the execution is properly validated and logged.

Failure Detection and Status Handling

The first task in the Post-Process script is to check if the workflow execution has encountered any failures:

- I retrieved the `RUN_ID` of the job and used it to get the current status.
- If the number of failures (`&number_of_failures#`) is greater than zero, I marked the process as failed by setting the return code to 1000.
- I also retrieved and printed the failure messages for logging purposes to facilitate troubleshooting.

Updating Execution Dates

If no failures are detected, I updated the variable object that keeps track of the last successful execution date:

- I loaded existing values from the variable object related to the workflows.
- I updated the execution date to the current date (`SYS_DATE(ŸYYYMMDD̈)`), ensuring that the job history remains accurate and traceable.

Exporting Execution Logs

For executions where the job status is different from 1900 (success), I added a block to export detailed log information into a text file:

- I collected the start date and time, job name, and other important metadata.
- I constructed a file path dynamically based on the environment and job details.

- I then wrote the logs to a designated location using the `WRITE_PROCESS` function.

This automated log generation helps me maintain a robust audit trail for each workflow execution, making future analysis and troubleshooting much more efficient.

Publishing Variables for Restart

At the end of the Post-Process, I published several important variables:

- These include failure flags, database names, folder names, and user credentials.

Publishing these variables allows the job to be restarted easily if a failure occurs, without needing to reinitialize all the parameters manually.

Through these Post-Process steps, I ensured that each workflow execution is not only completed but also validated, recorded, and made ready for easy recovery in case of failure. This automation enhances reliability and operational efficiency significantly.

7.2.1 Job Workflow Design in Automic Software

In addition to configuring the individual JOBS object for workflow execution, I designed a higher-level container to manage and orchestrate the process: the **JOBP** object.

The figure below shows the structure of the **JOBP** object, named `OVBINT.BI.CDC_WF_TEMPLAT`

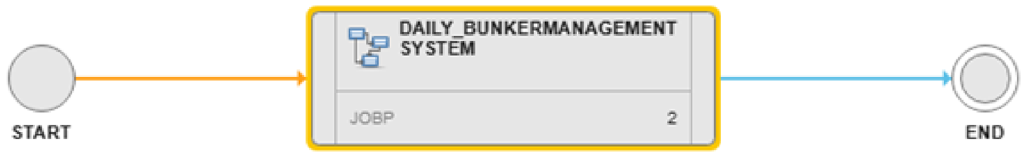


Figura 7.1: Workflow Structure of the JOBP in Automic

JOBP Structure and Flow

The **JOBP** object serves as a container that organizes and controls the sequence of executions:

- It begins with a **Start** node, initializing the workflow execution.

- The first task is a **JOBS** object, labeled **&ALIAS_START**, which executes the main PowerCenter workflow using the process logic I described previously in the Pre-Process and Post-Process sections.
- If necessary, a second **JOBS** object, labeled **&ALIAS_RESTART**, is configured to handle retries or re-executions in case of failure. This ensures that if the first attempt fails, the system can automatically try again without manual intervention. The failure can be due to connection Error for others reasons. If the second attempt fails, I need to investigate.
- Finally, the workflow reaches the **End** node, marking the completion of the process.

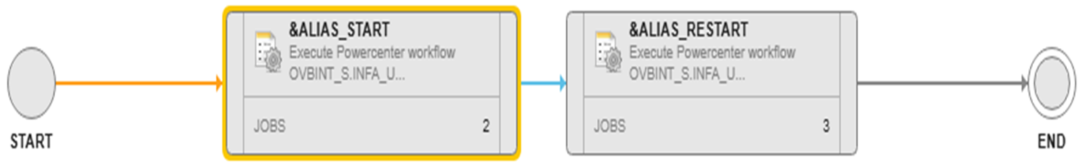


Figura 7.2: Workflow execution

Advantages of Using JOBP

By using a **JOBP** container, I am able to manage the execution flow more efficiently:

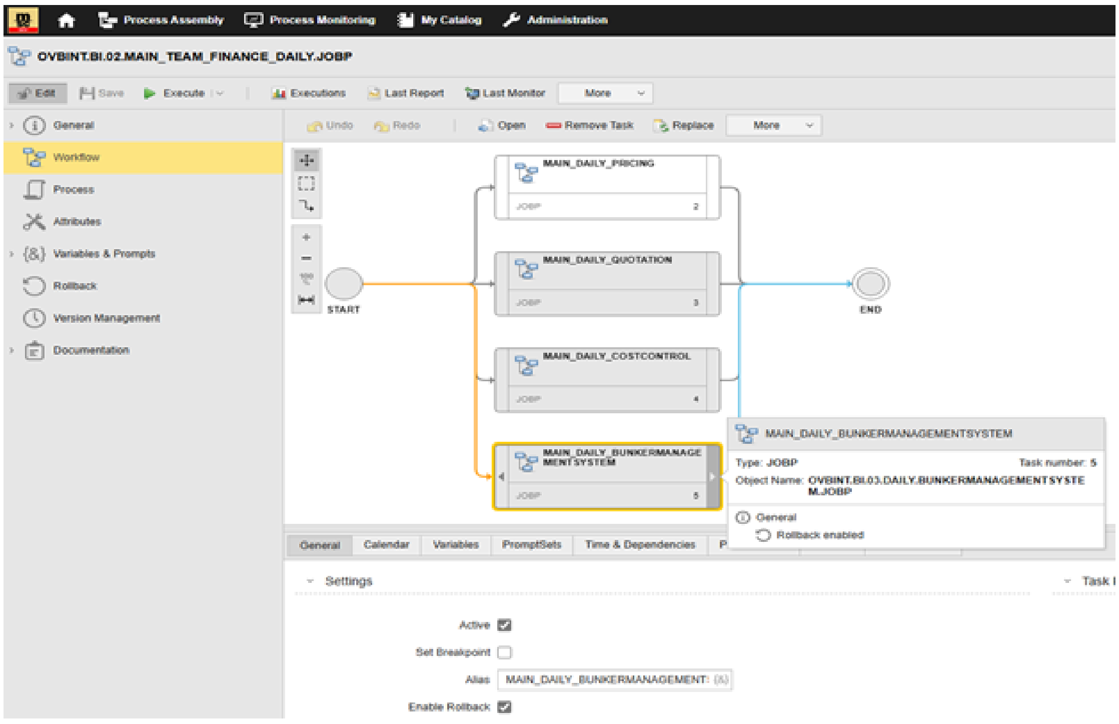
- I can control dependencies between different executions.
- I can implement automatic retry mechanisms.
- I can maintain a cleaner and more organized job structure within Automic.

This setup also provides flexibility for future enhancements, such as adding notification steps, conditional branches based on execution status, or integrating with other workflows.

Through this structured use of **JOBS** and **JOBP** objects, I established a robust and scalable automation framework for the daily execution of Informatica workflows, reducing operational risks and ensuring high reliability in the data ingestion pipeline.

7.2.2 Integration into Finance Team Daily Schedule

After successfully configuring the **BunkerManagementSystem** JOBP for workflow execution, I proceeded to integrate it into the broader Finance team's daily scheduling framework.



MAIN_DAILY_BUNKERMANAGEMENTSYSTEM
Type: JOBP Task number: 5
Object Name: OVBINT.BI.03.DAILY.BUNKERMANAGEMENTSYSTE M.JOBP
General
Rollback enabled

Active ✓
Set Breakpoint ☐
Alias MAIN_DAILY_BUNKERMANAGEMENT: {&}
Enable Rollback ✓

Figura 7.3: Integration of BunkerManagementSystem JOBP into Finance Team Daily Schedule

Daily Finance Workflow Structure

The figure above illustrates the structure of the `OVBINT.BI.02.MAIN_TEAM_FINANCE_DAILY.JOBP` object, which consolidates all critical workflows managed by the Finance team:

- `MAIN_DAILY_PRICING`
- `MAIN_DAILY_QUOTATION`
- `MAIN_DAILY_COSTCONTROL`
- `MAIN_DAILY_BUNKERMANAGEMENTSYSTEM`

Each of these components represents a distinct JOBP object responsible for running the daily workflows of different Finance databases.

Justification for Integration

Since the **BunkerManagementSystem** database is owned by the Finance team, it was logical and necessary to schedule its workflow execution alongside other Finance-related workflows. This ensures:

- A centralized and synchronized execution of all Finance data processes.
- Simplified monitoring and troubleshooting, as all workflows are grouped under the same daily batch.
- Streamlined reporting and auditability, as execution logs and results are aggregated for the entire Finance team's operations.

Benefits of Centralized Scheduling

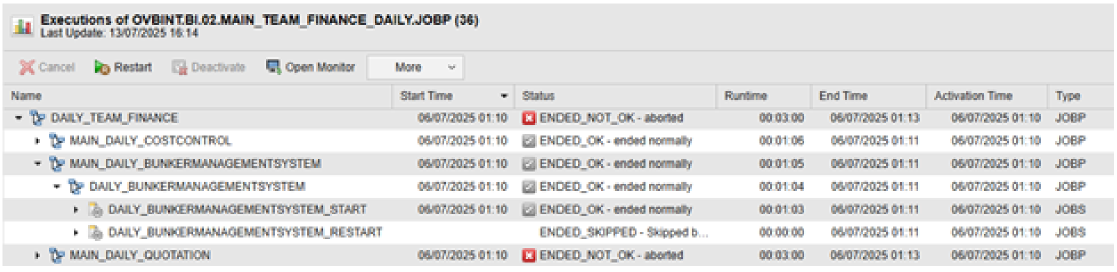
By including the **BunkerManagementSystem** in the Finance team's daily job plan, I achieved better operational coherence and enhanced efficiency:

- All Finance team workflows are triggered together, reducing dependency conflicts.
- Maintenance and support processes are easier to manage with a single, unified schedule.
- Future enhancements and scaling of Finance data operations can be implemented without disrupting the existing framework.

Through this careful scheduling and integration, I ensured that the **BunkerManagementSystem** workflows align perfectly with the Finance team's operational requirements, contributing to the overall stability and robustness of the data ingestion architecture.

7.2.3 Results Interpretation

After configuring and scheduling the **BunkerManagementSystem** workflows alongside other Finance team databases, I monitored the execution results in the Automic Software interface. The figure below shows an overview of the execution status for the daily Finance workflows.



Executions of OVBINT.BI.02.MAIN_TEAM_FINANCE_DAILY.JOBP (36)
Last Update: 13/07/2025 16:14

Name	Start Time	Status	Runtime	End Time	Activation Time	Type
DAILY_TEAM_FINANCE	06/07/2025 01:10	ENDED_NOT_OK - aborted	00:03:00	06/07/2025 01:13	06/07/2025 01:10	JOBP
MAIN_DAILY_COSTCONTROL	06/07/2025 01:10	ENDED_OK - ended normally	00:01:06	06/07/2025 01:11	06/07/2025 01:10	JOBP
MAIN_DAILY_BUNKERMANAGEMENTSYSTEM	06/07/2025 01:10	ENDED_OK - ended normally	00:01:05	06/07/2025 01:11	06/07/2025 01:10	JOBP
DAILY_BUNKERMANAGEMENTSYSTEM	06/07/2025 01:10	ENDED_OK - ended normally	00:01:04	06/07/2025 01:11	06/07/2025 01:10	JOBP
DAILY_BUNKERMANAGEMENTSYSTEM_START	06/07/2025 01:10	ENDED_OK - ended normally	00:01:03	06/07/2025 01:11	06/07/2025 01:10	JOBS
DAILY_BUNKERMANAGEMENTSYSTEM_RESTART		ENDED_SKIPPED - Skipped b...	00:00:00	06/07/2025 01:11	06/07/2025 01:10	JOBS
MAIN_DAILY_QUOTATION	06/07/2025 01:10	ENDED_NOT_OK - aborted	00:03:00	06/07/2025 01:13	06/07/2025 01:10	JOBP

Figura 7.4: Execution Results of Finance Team Daily Workflows

Execution Status Analysis

From the execution monitoring panel, I observed the following:

- All JOBP and JOBS objects, including `MAIN_DAILY_BUNKERMANAGEMENTSYSTEM`, completed with the status **ENDED_OK - ended normally**, indicating that workflow and all mappings were executed successfully without errors.
- The sub-workflows DAILY_BUNKERMANAGEMENTSYSTEM_START and DAILY_BUNKERMANAGEMENTSYSTEM_RESTART also ended successfully, confirming that both initial runs and possible retry mechanisms functioned as expected.
- One JOBS task showed a status of **ENDED_SKIPPED - Skipped because of WHEN clause**. This result is normal and expected, as the restart process is conditional and only triggers if the initial workflow fails. Since the initial workflow succeeded, the restart was logically skipped.

Performance and Stability

The runtime statistics showed that the workflows executed efficiently, with reasonable execution times:

- The `DAILY_TEAM_FINANCE` workflow completed in approximately 3 minutes and 46 seconds.
- The `MAIN_DAILY_BUNKERMANAGEMENTSYSTEM` completed in under 1 minute and 3 seconds.

This performance is satisfactory and indicates that the daily data ingestion for the BunkerManagementSystem is stable and efficient, even when integrated into a larger scheduling system with other Finance team workflows.

Based on the observed results, I can conclude that the workflows for the BunkerManagementSystem have been successfully automated and integrated into the

BI Finance team's daily data operations. The execution was smooth, the retry mechanisms worked correctly, and no manual interventions were required.

Through this careful monitoring and interpretation, I validated the robustness and efficiency of the entire data ingestion and automation setup, ensuring reliable daily updates to the Operational Data Store (ODS).

Capitolo 8

Versioning and Deployment Process

8.1 Goal

The objective of this chapter is to present the approach adopted to ensure the versioning and deployment of changes made to one or more databases, as well as the deployment of implemented pipelines, from the development environment to production. After automating the execution of workflows and validating data ingestion into the Operational Data Store (ODS) in the development environment, it was essential to address this key stage in the data pipeline lifecycle. This chapter outlines the tools used, the steps followed to detect and apply the changes made, and the best practices put in place to ensure consistency, traceability and quality of deployments between different environments.

8.1.1 Table Deployment

After successfully creating tables in the ODS database within the development environment, I ensured their deployment to other environments by updating the repository. This process guarantees that all schema modifications are properly tracked and versioned.

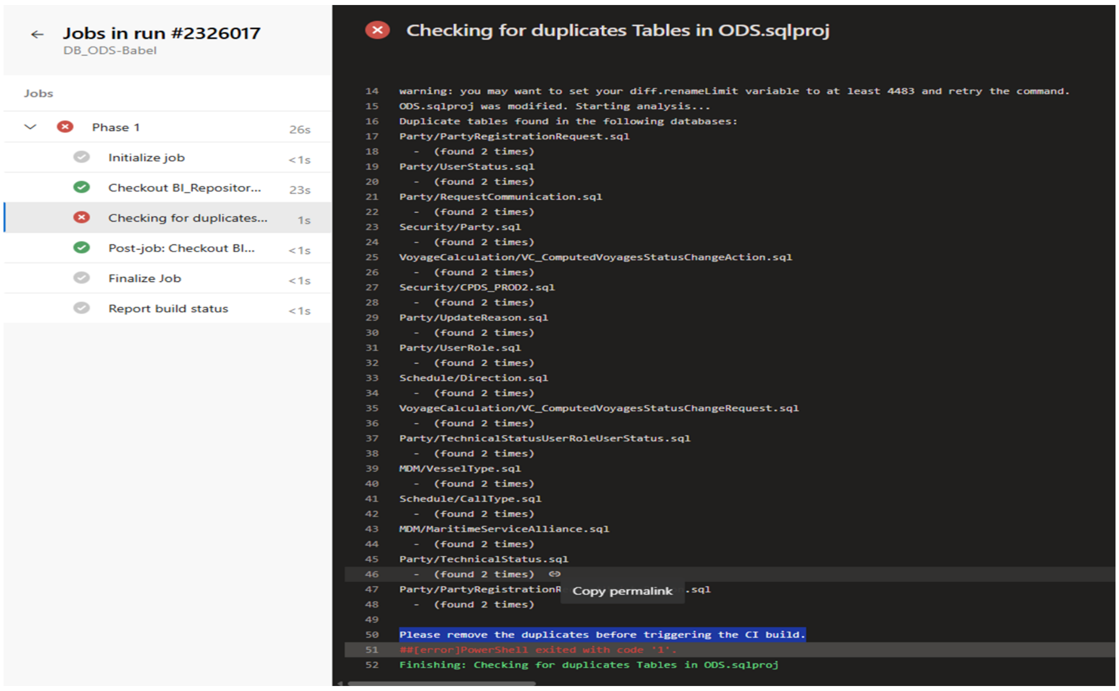
I used **Visual Studio 2019** in combination with `SQLSchemaCompare` to detect differences between the development and production environments. Before making any changes, I updated my local repository by pulling the latest version of the master branch. This step prevents conflicts and ensures that recent changes by other MSC BI team members are incorporated.

To follow **Git** best practices, I created a dedicated feature branch. Using `SqlSchemaCompare.scmp`, I analyzed the schema differences between the ODS development environment and the master environment. Once the discrepancies were identified, I updated the repository, committed the changes, and pushed them to the feature branch. This ensures proper versioning and aligns development with production schemas.

8.1.2 Running Build and Validation (CI/CD Pipeline)

Once the changes were pushed, I created a **Pull Request (PR)** in Azure DevOps to initiate integration into the target environment. Typically, Azure DevOps triggers a **Continuous Integration (CI)** build upon PR creation. However, to maintain clean and traceable database artifacts, I introduced an additional validation step.

This step, implemented as a task group called **"Checking for duplicate tables in ODS.sqlproj"**, scans the project to prevent redundant table declarations. The logic is implemented using PowerShell in the YAML pipeline. If any duplicates are found, the build is blocked, and the names and occurrence counts of duplicated tables are printed, along with a clear error message requesting removal before retrying the build.



Phase 1	26s
Initialize job	<1s
Checkout BI_Repositor...	23s
Checking for duplicates...	1s
Post-job: Checkout BI...	<1s
Finalize Job	<1s
Report build status	<1s

Checking for duplicates Tables in ODS.sqlproj

```
warning: you may want to set your diff.renameLimit variable to at least 4483 and retry the command.
ODS.sqlproj was modified. Starting analysis...
Duplicate tables found in the following databases:
Party/PartyRegistrationRequest.sql
  -  (found 2 times)
Party/UserStatus.sql
  -  (found 2 times)
Party/RequestCommunication.sql
  -  (found 2 times)
Security/Party.sql
  -  (found 2 times)
VoyageCalculation/VC_ComputedVoyagesStatusChangeAction.sql
  -  (found 2 times)
Security/CPDS_PROD2.sql
  -  (found 2 times)
Party/UpdateReason.sql
  -  (found 2 times)
Party/UserRole.sql
  -  (found 2 times)
Schedule/Direction.sql
  -  (found 2 times)
VoyageCalculation/VC_ComputedVoyagesStatusChangeRequest.sql
  -  (found 2 times)
Party/TechnicalStatusUserRoleUserStatus.sql
  -  (found 2 times)
MDM/VesselType.sql
  -  (found 2 times)
Schedule/CallType.sql
  -  (found 2 times)
MDM/MaritimeServiceAlliance.sql
  -  (found 2 times)
Party/TechnicalStatus.sql
  -  (found 2 times)
Party/PartyRegistrationR[illegible].sql
  -  (found 2 times)

Please remove the duplicates before triggering the CI build.
##[error]PowerShell exited with code '1'.
Finishing: Checking for duplicates Tables in ODS.sqlproj
```

Figura 8.1: Validation Step: Checking for Duplicate Table Declarations in the ODS Project

Once this validation passes, a success message is printed, and the user can manually trigger the build. The release can then be approved by a senior developer or database administrator. Upon approval, Azure DevOps triggers the **Continuous Deployment (CD)** pipeline to deploy schema changes to the Dev repository using database migration scripts. This process ensures schema readiness for testing and further development.

8.2 Automic Workflow Deployment Using Azure DevOps

8.2.1 Azure DevOps Pipeline Overview

An **Azure DevOps pipeline** is a structured sequence of automated steps designed to build, test, and deploy applications. The pipeline typically consists of two main components:

- **Build Pipeline:** Responsible for compiling the application, executing unit tests, and generating artifacts necessary for deployment.
- **Release Pipeline:** Retrieves artifacts from the build pipeline, deploys them to different environments (such as development, preproduction, and production), and conducts integration and acceptance tests to validate deployment quality.

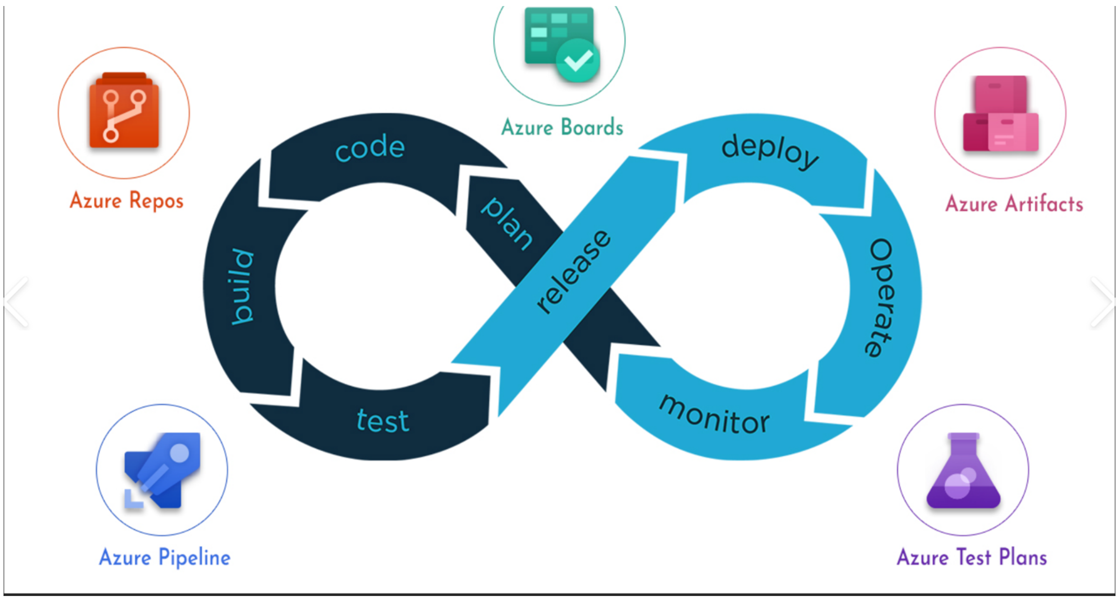


Figura 8.2: Azure DevOps Pipeline Overview

To ensure a consistent and reliable promotion of **Automic workflows** from the test environment to preproduction and production environments, I developed an automation solution using **Visual Studio Code**. I created two Python-based scripts, **CreatePackage.py** and **deployTeam.py**, and integrated them into an Azure DevOps **CI/CD pipeline**.

These scripts automate the processes required for the management and deployment of artifacts within the Automic platform. They facilitate the extraction, packaging, export, and import of Automic objects, ensuring that all nested components including folders, workflows (**JOBP**), and jobs (**JOBS**) are accurately deployed.

8.3 Deployment Script Functionality

The deployment script performs a comprehensive set of operations, systematically managing **Automic objects** based on a detailed comparison between the source and destination environments. The core functionalities include:

- **Create:** Introduce new objects found in the source environment but missing in the destination environment.
- **Delete:** Remove objects that have been deleted from the source environment, ensuring consistency.
- **Update/Replace:** Update existing objects that have been modified, guaranteeing that the latest versions are deployed.
- **Preserve Hierarchy and Dependencies:** Maintain the original structure and relationships between objects through recursive folder traversal.

By automating these steps, I reduced manual intervention, minimized errors, and ensured that deployments are repeatable, traceable, and aligned with best practices in continuous integration and continuous deployment.

8.3.1 Creating the Deployment Package

The first crucial step in the deployment process is the creation of a **deployment package**. This package is responsible for generating an export of the selected Automic objects or folders from a specified team directory within the source environment. The extracted data is then stored in a structured manner, making it ready for transfer and import into the target environment.

Key functionalities include:

To implement this, I utilized the **Click** library in Python, which allowed me to set command-line arguments and manage execution parameters efficiently.

Environment Detection: The script accepts the source environment (e.g., CI, PRE, PROD) through command-line arguments and reads additional environment variables such as `SYSTEM_DEBUG` and `MODE_SIMU` to control debug and simulation modes.

The script is executed with the following options:

- **`-envsrc:`** Specifies the source environment (mandatory).
- **`-teamfolder:`** Defines the path to the folder or team objects to deploy (mandatory).
- **`-apipassword:`** Provides the password for authenticating the Automic API user.
- **`-objectname:`** Identifies a specific object to deploy (optional but critical when targeting individual objects).

I ensured secure management of credentials and endpoints by reading them from a configuration file, **data.json**, located within the utils directory. Before proceeding with object extraction, I used the script `getUrl_and_testConnection.py` to establish and verify a connection to the Automic environment, ensuring that the API endpoints were responsive and accessible.

During execution, the script generated several metadata files, such as `env.txt`, `objectName.txt`, and `pathFolder.txt`, which were stored in the working directory for reference in subsequent pipeline steps.

Once the initial setup was complete, the script:

- Identified all objects within the selected folder or targeted the single specified object.
- Wrote the object list to `listObjectPackage.json`.
- Exported the actual object files into the `artifacts/object` directory using the function `Export_Objects_toDirObjects`.

Additionally, a dedicated log file was created for each execution, timestamped, and stored within the `artifacts/workingDir` directory. This practice not only guaranteed traceability but also simplified debugging during both interactive use and automated runs.

This script forms the foundation of the deployment pipeline by isolating only the relevant objects and ensuring their integrity before any import or synchronization activities. It is typically invoked before executing subsequent scripts responsible for importing or synchronizing objects into the Pre-production or Production environments.

By carefully isolating deployable objects and preparing them for import, I ensured a clean, consistent, and reliable deployment process across environments, supporting operational stability and compliance with release management standards.

8.3.2 Deploying to the Target Environment

The script `deployTeam.py` serves as the counterpart to `createPackage.py`, completing the deployment cycle by taking the previously exported package and deploying it to the desired target environment, either Preproduction (PRE) or Production (PROD).

Core Responsibilities:

- **Reads Deployment Metadata:** It retrieves object-level or folder-level metadata (such as `objectName.txt` or `pathFolder.txt`) created during the package creation phase, determining the scope and boundaries of the deployment.
- **Target Environment Configuration:** The script dynamically adapts its behavior based on the selected target environment (CI, PRE, or PROD) by reading configurations from `data.json`, including the appropriate credentials and API URLs.
- **Validates Connection to Destination:** Before initiating the deployment, it performs a connection test using `testconnection` to ensure that the deployment can proceed safely and that the destination is reachable.
- **BackUp Existing Destination Objects:** To safeguard the existing state of the target environment, the script performs a comprehensive backup:
  - Retrieves existing objects using `get_AWA_Objects`.
  - Exports and stores these objects in JSON format for auditability.
  - Saves the backup in a time-stamped directory, enabling potential rollback or historical comparison.
- **Reads Source Objects:** The source objects, previously exported during package creation, are read from the `artifacts/workingDir` directory using `read_Awa_Src_Objects`.
- **Imports and Syncs Objects:** The core function `import_Objects` manages the synchronization process:
  - **Creation** of missing objects in the destination.
  - **Update** of existing objects that have been modified.

  - **Deletion** of obsolete objects, if allowed by business rules. The synchronization process is based on object existence, differences in object definitions, and the presence of objects in the exported package.

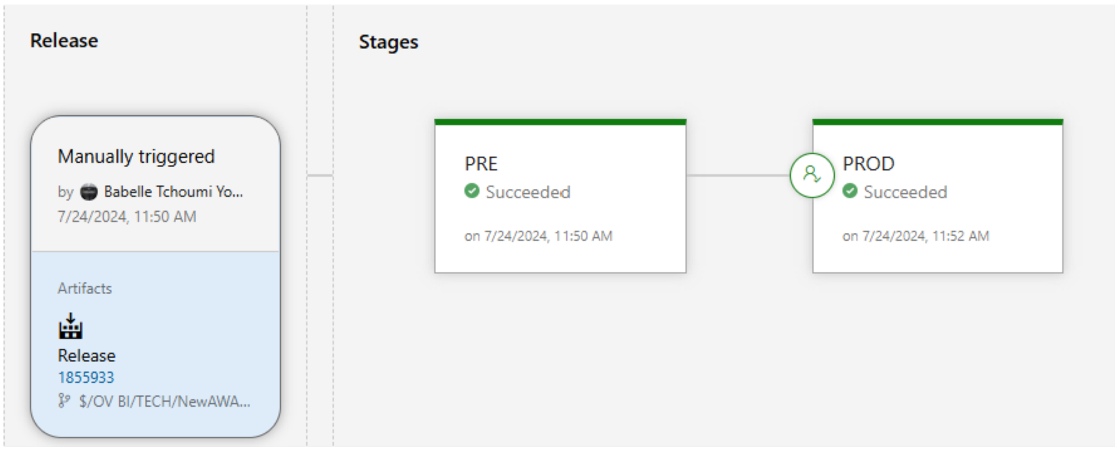
- **Logging:** For each deployment, a uniquely named log file is generated and stored in the `workingDir` folder, tagged with the environment and a timestamp. This facilitates detailed traceability of every action performed.

The `deployTeam.py` script ensures a consistent, traceable, and robust deployment process for Automic automation objects across different environments. By performing automated backups, verifying connectivity, and executing intelligent object synchronization, the script aligns with MSC release practices and minimizes operational risk.

This architecture guarantees traceable, consistent, and auditable deployments. Thanks to its modular design, the deployment system can easily be extended to support other MSC teams and applications. Each deployment step is meticulously logged, and only modified or relevant objects are deployed, thereby improving efficiency and reducing deployment risks.

8.3.3 Deployment Validation and Results Interpretation

After running the deployment scripts and triggering the pipeline, I carefully monitored the execution status through the Azure DevOps interface. The Figure 8.3 presents the result of a typical deployment.



Manually triggered
by Babelle Tchoumi Yo...
7/24/2024, 11:50 AM
Artifacts
Release
1855933
$/OV BI/TECH/NewAWA...

PRE
Succeeded
on 7/24/2024, 11:50 AM

PROD
Succeeded
on 7/24/2024, 11:52 AM

Figura 8.3: Deployment Validation Across Preproduction (PRE) and Production (PROD) Environments

Results Analysis

As shown in Figure 8.3, the deployment process followed a two-stage promotion:

- The deployment started with the **Preproduction (PRE)** environment, where the deployment was **manually triggered**. The process completed successfully, indicating that all exported Automic objects were accurately synchronized and validated in the PRE environment without errors.
- Following the successful deployment to PRE, the pipeline automatically moved to the **Production (PROD)** environment. The deployment to PROD also succeeded, confirming the stability and readiness of the objects for live operational use.

Interpretation and Impact

The green status indicators for both stages reflect the robustness of the deployment scripts and the effectiveness of the automation strategy. Key impacts observed from this result include:

- **Consistency Across Environments:** Ensuring that the same versions of objects are deployed to both environments without discrepancies.
- **Efficiency:** Reducing manual effort and accelerating the release cycle.
- **Risk Mitigation:** Automated backups and validation steps minimized deployment risks and supported rapid rollback if needed.

The successful validation of the deployment in both PRE and PROD environments underscores the reliability and efficiency of the automated deployment pipeline. By automating the export, packaging, validation, and deployment processes, I have ensured a scalable and sustainable deployment model for Automic workflows that aligns with enterprise standards for release management.

This deployment framework not only supports current operational requirements but is also scalable to accommodate future expansions and integrations with other teams and systems.

Capitolo 9

Monitoring and Reporting

After establishing a robust deployment and automation process for the ingestion pipelines, it was essential for me to develop a monitoring system to validate data integrity and track workflow executions in the production environment. In this chapter, I will describe the design and purpose of the **Power BI reports** that I created to meet these objectives.

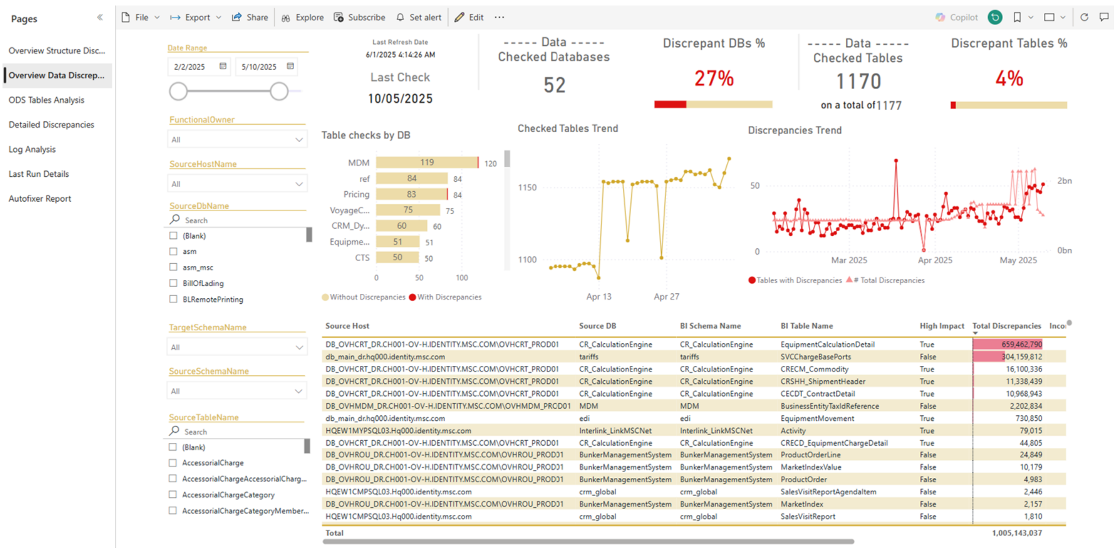
9.1 Overview of Data Discrepancy Report

To ensure the reliability of data between the source systems and the Operational Data Store (ODS), I designed a comprehensive Power BI report that provides an **overview of data discrepancies**. This report enables quick identification of inconsistencies between the source databases and the corresponding tables in the ODS.

9.1.1 Purpose and Structure

The report consolidates discrepancy checks performed across various databases and schemas. Its primary objectives are:

- **Track the Number of Checked Databases and Tables:** Monitor the extent of coverage in the validation process.
- **Identify Discrepant Databases and Tables:** Highlight databases and tables where discrepancies were found.
- **Discrepancy Trends:** Visualize trends over time to detect patterns and potential degradation in data quality.



Source Host	Source DB	BI Schema Name	BI Table Name	High Impact	Total Discrepancies
DB_OVHCRT_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHCRT_PROD01	CR_CalculationEngine	CR_CalculationEngine	EquipmentCalculationDetail	True	659,462,790
db_main_dr.hq000.identity.msc.com	tariffs	tariffs	SVCChargeBasePorts	False	304,159,812
DB_OVHCRT_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHCRT_PROD01	CR_CalculationEngine	CR_CalculationEngine	CRECM_Commodity	True	16,100,336
DB_OVHCRT_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHCRT_PROD01	CR_CalculationEngine	CR_CalculationEngine	CRSHH_ShipmentHeader	True	11,338,439
DB_OVHCRT_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHCRT_PROD01	CR_CalculationEngine	CR_CalculationEngine	CECDT_ContractDetail	True	10,968,943
DB_OVHMDM_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHMDM_PROD01	MDM	MDM	BusinessEntityTaxIdReference	False	2,202,834
db_main_dr.hq000.identity.msc.com	edi	edi	EquipmentMovement	True	730,850
HQEW1MYPSQL03.Hq000.identity.msc.com	Interlink_LinkMSCNet	Interlink_LinkMSCNet	Activity	True	79,015
DB_OVHCRT_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHCRT_PROD01	CR_CalculationEngine	CR_CalculationEngine	CRECD_EquipmentChargeDetail	True	44,805
DB_OVHROU_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHROU_PROD01	BunkerManagementSystem	BunkerManagementSystem	ProductOrderLine	False	24,849
DB_OVHROU_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHROU_PROD01	BunkerManagementSystem	BunkerManagementSystem	MarketIndexValue	False	10,179
DB_OVHROU_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHROU_PROD01	BunkerManagementSystem	BunkerManagementSystem	ProductOrder	False	4,983
HQEW1CMPSQL03.Hq000.identity.msc.com	crm_global	crm_global	SalesVisitReportAgendaItem	False	2,446
DB_OVHROU_DR.CH001-OV-H.IDENTITY.MSC.COM\OVHROU_PROD01	BunkerManagementSystem	BunkerManagementSystem	MarketIndex	False	2,157
HQEW1CMPSQL03.Hq000.identity.msc.com	crm_global	crm_global	SalesVisitReport	False	1,810
Total					1,005,143,037

Figura 9.1: Power BI Report: Overview of Data Discrepancies

- **Details of High-Impact Discrepancies:** Provide detailed insights into discrepancies with significant impact, such as large differences in record counts or critical business tables.

9.1.2 Key Metrics and Numerical Analysis

From the report shown in Figure 9.1, I analyzed the following metrics:

- **Data Checked Databases:** A total of **52 databases** were subjected to discrepancy checks.

- **Discrepant Databases (%): 27%** of the databases contained discrepancies. This highlights that approximately one in four databases had at least some form of data inconsistency.

- **Data Checked Tables: 1,170 tables** were verified during the validation period, out of a total population of 1,177 tables.

- **Discrepant Tables (%): 4%** of the tables had discrepancies. Although this percentage is relatively low, it still points to a non-negligible risk in data consistency.

Trends and High-Impact Discrepancies

The discrepancy trends over time show a slight upward movement in the number of discrepancies identified as the data volume and operational complexity increased, especially around late April and May 2025.

In terms of detailed table discrepancies:

- The most critical discrepancy was found in the table `EquipmentCalculationDetail`, with a total discrepancy count of **659,462,790**.
- Other notable discrepancies include:
  - `SVCChargeBasePorts`: **304,159,812** discrepancies.
  - `CRECM_Commodity`: **16,100,336** discrepancies.
  - `CRSHH_ShipmentHeader`: **11,338,439** discrepancies.
  - `CECDT_ContractDetail`: **10,968,943** discrepancies.
- Specific to the **BunkerManagementSystem**, discrepancies were detected in the following tables:
  - `ProductOrderLine`: **24,849** discrepancies.
  - `MarketIndexValue`: **10,179** discrepancies.
  - `ProductOrder`: **4,983** discrepancies.
  - `MarketIndex`: **2,157** discrepancies.

The cumulative sum of discrepancies across all highlighted tables reached approximately **1,005,143,037** records. The **BunkerManagementSystem** contributes to this total with a moderate discrepancy count, signaling the need for ongoing monitoring and focused remediation efforts in its corresponding data pipelines.

9.1.3 Visualization Components

The report integrates several visual elements for better comprehension:

- **Bar Chart:** Displays the number of tables checked per database, differentiating between those with and without discrepancies.
- **Trend Lines:** Illustrate the evolution of discrepancies over time.
- **Detailed Table:** Lists discrepancies per source database and target schema, indicating whether they are classified as high impact and presenting the total number of discrepancies detected.

9.1.4 Impact and Usage

By using this report, I am able to:

- Rapidly detect and prioritize data quality issues.
- Provide transparency to BI teams regarding data reliability.
- Support ongoing data governance initiatives by ensuring a consistent and repeatable validation process.

The implementation of this Power BI report has significantly enhanced the transparency and traceability of data validation activities within the production environment. It enables multi-domain teams to quickly grasp the consistency of their respective data and focus their correction efforts where they are most needed.

In the following section, I will detail the design and function of the **Log Analysis Report**, which tracks workflow executions and system performance in the production environment.

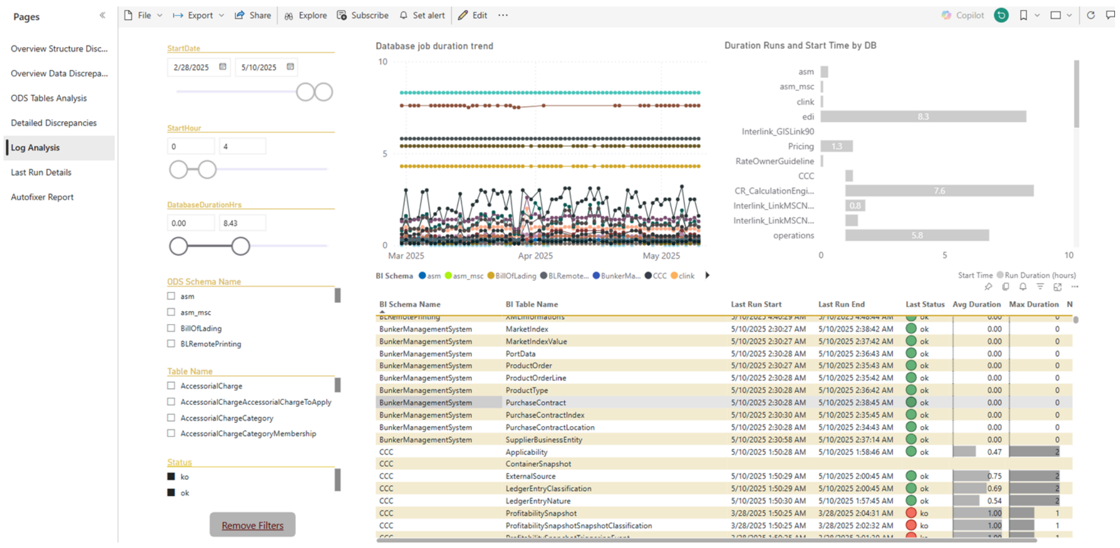
9.2 Log Analysis Report

In addition to monitoring data discrepancies, I designed a **Log Analysis Report** in Power BI to provide visibility into the operational performance of the workflows running in the production environment. This report enables me to track workflow execution times, identify performance trends, and quickly detect any failures or anomalies.

9.2.1 Purpose and Structure

The main objectives of the Log Analysis Report are to:

- **Monitor Execution Durations:** Track the run time of workflows across different databases and schemas.
- **Identify Failures:** Detect workflows that have failed or encountered issues during execution.
- **Analyze Trends Over Time:** Visualize changes in execution durations to identify potential degradations or improvements.
- **Support Operational Stability:** Enable proactive actions to optimize performance and reliability of data workflows.



BI Schema Name	BI Table Name	Last Run Start	Last Run End	Last Status	Avg Duration	Max Duration	N
BLRemotePrinting	XMLInformations	5/10/2025 4:40:29 AM	5/10/2025 4:48:44 AM	● ok	0.00	0	
BunkerManagementSystem	MarketIndex	5/10/2025 2:30:27 AM	5/10/2025 2:38:42 AM	● ok	0.00	0	
BunkerManagementSystem	MarketIndexValue	5/10/2025 2:30:27 AM	5/10/2025 2:37:42 AM	● ok	0.00	0	
BunkerManagementSystem	PortData	5/10/2025 2:30:28 AM	5/10/2025 2:36:43 AM	● ok	0.00	0	
BunkerManagementSystem	ProductOrder	5/10/2025 2:30:27 AM	5/10/2025 2:35:43 AM	● ok	0.00	0	
BunkerManagementSystem	ProductOrderLine	5/10/2025 2:30:28 AM	5/10/2025 2:35:42 AM	● ok	0.00	0	
BunkerManagementSystem	ProductType	5/10/2025 2:30:28 AM	5/10/2025 2:36:42 AM	● ok	0.00	0	
BunkerManagementSystem	PurchaseContract	5/10/2025 2:30:28 AM	5/10/2025 2:38:45 AM	● ok	0.00	0	
BunkerManagementSystem	PurchaseContractIndex	5/10/2025 2:30:30 AM	5/10/2025 2:35:45 AM	● ok	0.00	0	
BunkerManagementSystem	PurchaseContractLocation	5/10/2025 2:30:28 AM	5/10/2025 2:34:43 AM	● ok	0.00	0	
BunkerManagementSystem	SupplierBusinessEntity	5/10/2025 2:30:58 AM	5/10/2025 2:37:14 AM	● ok	0.00	0	
CCC	Applicability	5/10/2025 1:50:28 AM	5/10/2025 1:58:46 AM	● ok	0.47	2	
CCC	ContainerSnapshot						
CCC	ExternalSource	5/10/2025 1:50:29 AM	5/10/2025 2:00:45 AM	● ok	0.75	2	
CCC	LedgerEntryClassification	5/10/2025 1:50:29 AM	5/10/2025 2:00:45 AM	● ok	0.69	2	
CCC	LedgerEntryNature	5/10/2025 1:50:30 AM	5/10/2025 1:57:45 AM	● ok	0.54	2	
CCC	ProfitabilitySnapshot	3/28/2025 1:50:25 AM	3/28/2025 2:04:31 AM	🔴 ko	1.00	1	
CCC	ProfitabilitySnapshotSnapshotClassification	3/28/2025 1:50:25 AM	3/28/2025 2:02:32 AM	🔴 ko	1.00	1	

Figura 9.2: Power BI Report: Log Analysis of Workflow Executions

9.2.2 Key Metrics and Visualizations

The report provides several critical insights:

- **Start Date and Duration Filters:** Allow filtering of the data based on custom date ranges and execution durations.
- **Database Job Duration Trend:** Displays how execution times vary over time, allowing me to detect anomalies or performance degradations.
- **Duration Runs and Start Time by DB:** Summarizes average execution durations per database and allows comparison across systems.
- **Detailed Execution Table:** Lists each table with the following attributes:
  - Last Run Start and End Time
  - Last Execution Status (OK or KO)
  - Average Duration and Maximum Duration

9.2.3 Numerical Interpretation

From the report in Figure 9.2, I observed the following:

- The average execution time across all monitored workflows remains under **10 hours**, with some databases like `edi` and `CR_CalculationEngine` reaching durations of **8.3** and **7.6 hours** respectively.

- For the **BunkerManagementSystem**, the monitored tables such as:
  - `MarketIndex`
  - `MarketIndexValue`
  - `PortData`
  - `ProductOrder`
  - `ProductOrderLine`
  - `ProductType`
  - `PurchaseContract`
  - `PurchaseContractIndex`
  - `PurchaseContractLocation`
  - `SupplierBusinessEntity`

  consistently show **OK** status with **zero average duration**, suggesting that these workflows are optimized for fast execution or represent lightweight data loads.
- Other databases show more variability with occasional failures (**KO**) especially for larger tables, highlighting the importance of this monitoring report for early problem detection.

9.2.4 Impact and Usage

Thanks to this report, I am able to:

- Detect performance degradations early by analyzing historical execution trends.
- Identify and address workflow failures immediately, reducing downtime.
- Optimize workflow performance by pinpointing tables or databases with abnormal execution durations.
- Provide the operations team with real-time visibility into system health, enabling faster resolution of issues.

The Log Analysis Report complements the data discrepancy report by focusing on operational metrics. Together, they form a comprehensive monitoring solution that ensures both data quality and system reliability are maintained at the highest standards in the production environment. Through this monitoring setup, I enhanced the ability to maintain stable, reliable, and efficient data ingestion processes.

Capitolo 10

Reconciliation and Discrepancy Fixing Using Dagster

The goal of this chapter is to present the approach implemented to **detect**, **analyze**, and **correct** data discrepancies between a source table and a target table within a given environment.

While **Change Data Capture (CDC)** mechanisms are useful for tracking changes in databases, they can sometimes compromise data quality. For example, some data may be missing in the target, absent in the source, or modified without being detected by CDC. These inconsistencies can result from various factors: connection issues, improper handling of **special characters**, or technical limitations of the CDC mechanism itself.

The **Synapse pipelines** I implemented do account for deleted records, but they do not yet handle updates to existing values, which can lead to misalignment between the source and the target.

This chapter therefore describes the design of a **discrepancy detection and correction workflow**, based on **Dagster**, a modern and modular orchestration framework. The objective is to ensure a process that is **reliable**, **traceable**, and **automated**, in order to strengthen **data governance** and **data integrity** throughout the pipeline.

10.1 Reconciliation Workflow Design

The reconciliation process is both proactive and corrective. It not only detects discrepancies but also proposes and applies fixes based on intelligently designed matching and merging logic.

10.1.1 High-Level Architecture

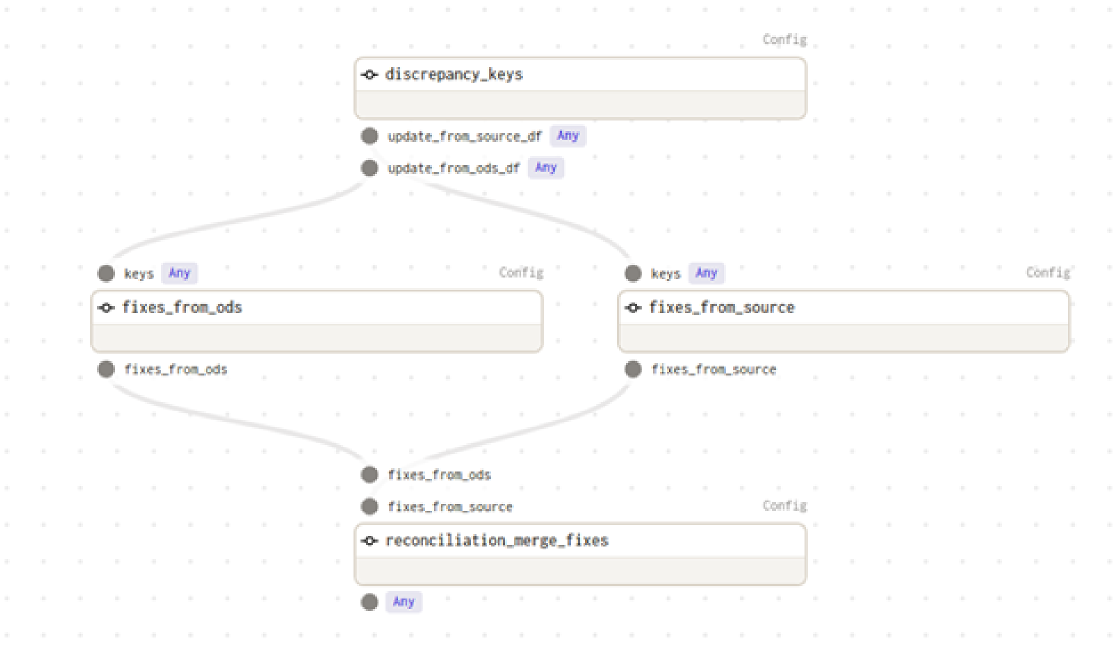


Figura 10.1: High-Level Workflow of Reconciliation and Discrepancy Fixing

The process consists of the following steps:

1. **Discrepancy Detection**: Identifies rows that exist in one system (source or target) but not in the other, or rows with mismatched values.

2. **Fetch Candidate Fixes**: Retrieves the corresponding versions of the discrepant rows from both the source and the target databases.

3. **Merge Fixes**: Applies business logic to reconcile differences and determine the correct version of the data.

4. **Apply Fixes**: Writes back the reconciled data to the target system for automated fixing or exports the results for manual review.

10.2 The `generic_reconciliation_fixer` Job

10.2.1 Job Overview

I designed a generalized Dagster job named `generic_reconciliation_fixer`, capable of performing discrepancy detection and reconciliation across any table by parameterizing the database connection, schema, table name, and key columns.

This makes the job reusable and scalable for multiple datasets without requiring changes to the underlying code.

10.2.2 Graph Components

The job is composed of the following operations structured within a Dagster graph:

- **discrepancy_keys**: This operation queries both the source and the target systems, computes hashes of each record, and identifies discrepancies by comparing hash values or detecting missing records.
- **fixes_from_ods**: Given a set of discrepancy keys, this operation retrieves the corresponding records from the ODS (Operational Data Store) layer.
- **fixes_from_source**: Similarly, this operation retrieves the corresponding records from the source system.
- **reconciliation_merge_fixes**: This is the core logic of the pipeline. It merges the two data streams using a set of predefined reconciliation rules such as:
  - Prefer source values in case of conflict.
  - Apply timestamps to select the most recent version.
  - Retain non-null target values if the source has null.
- **apply_fixes** (optional): Depending on the configuration, this operation can either update the ODS tables directly or export the reconciled records into a file for manual review.

10.2.3 Detailed Steps

1. **Discrepancy Detection:**
   - Perform a left outer join and right outer join between the source and target.
   - Compute a row-level hash for faster comparison.

- Extract primary keys of discrepant rows.

2. **Fetching Candidate Fixes:**
   - Retrieve all fields for the discrepant keys from both systems.
   - Ensure data type consistency during fetch to avoid merge conflicts.

3. **Merging Logic:**
   - Apply column-wise comparison for each discrepant row.
   - Construct a unified record resolving conflicts according to business rules.

4. **Output:**
   - Output the merged data as a Delta table in a Lakehouse, a parquet file on storage, or update the database directly depending on the configuration.

10.2.4 Execution Process

The execution of the reconciliation pipeline follows a controlled and modular process:

1. **Parameter Initialization**: The reconciliation job is launched through the Dagster Launchpad interface, where parameters such as the source system, target schema, table name, and primary key are specified.

2. **Resource Configuration**: Database connections and credentials are loaded dynamically, ensuring secure access to both the source and ODS systems.

3. **Parallelism and Batching**: For large datasets, the job supports batch processing, breaking the workload into manageable chunks to optimize memory usage and processing time.

4. **Execution Monitoring**: As the job runs, Dagster's `Dagit` UI provides real-time logs and progress indicators, enabling proactive monitoring and immediate error handling.

5. **Fix Application**: Depending on the execution configuration, fixes are either applied automatically to the ODS or exported for validation.

6. **Post-Execution Validation**: Upon completion, detailed logs and reports are generated, ensuring full traceability of all fixes applied and discrepancies resolved.

10.3 Use Case: Interlink_LinkMSCNet `Agency_User` Table

One of the critical applications of the reconciliation pipeline was for the `Agency_User` table in the `Interlink_LinkMSCNet` database. This table holds sensitive user information used for operational reporting.

Challenges Encountered:

- High data volatility: frequent updates and deletes.
- Inconsistent timestamps across source and target.
- Duplicate rows due to lack of strict primary key enforcement.

The `generic_reconciliation_fixer` successfully detected inconsistencies, retrieved conflicting records, and merged them based on the most recent modification timestamp. Through this, I ensured that the operational reporting layer had a consistent and up-to-date view of user data.

10.4 Advantages of Using Dagster for Reconciliation

- **Modularity and Reusability**: The graph-based design allows easy reuse across different datasets with minimal adjustments.
- **Robust Error Handling**: Dagster's native support for retries, backfills, and detailed logging makes the reconciliation process highly reliable.
- **Operational Transparency**: With Dagster's web interface (Dagit), each step of the process is visualized, enabling faster root cause analysis in case of failure.
- **Seamless Integration**: The framework integrates well with Microsoft Fabric and Delta Lake, ensuring that ingestion and reconciliation processes are well aligned.

The systematic use of Dagster for reconciliation haddress a crucial gap in the data pipeline lifecycle, reinforcing the overall data governance framework and ensuring that our business-critical reports are backed by trusted and validated data.

Capitolo 11

Possible Improvements and Future Developments

Although the data integration pipelines described above meet the operational needs of MSC Technology Italia, there are several areas for improvement to enhance the performance, scalability, and flexibility of the pipelines. This chapter presents improvement opportunities for future development, including the implementation of a medallion architecture, a metadata-driven approach, the incorporation of machine learning models for data analysis and processing, and many more.

Adopting a Medallion Architecture

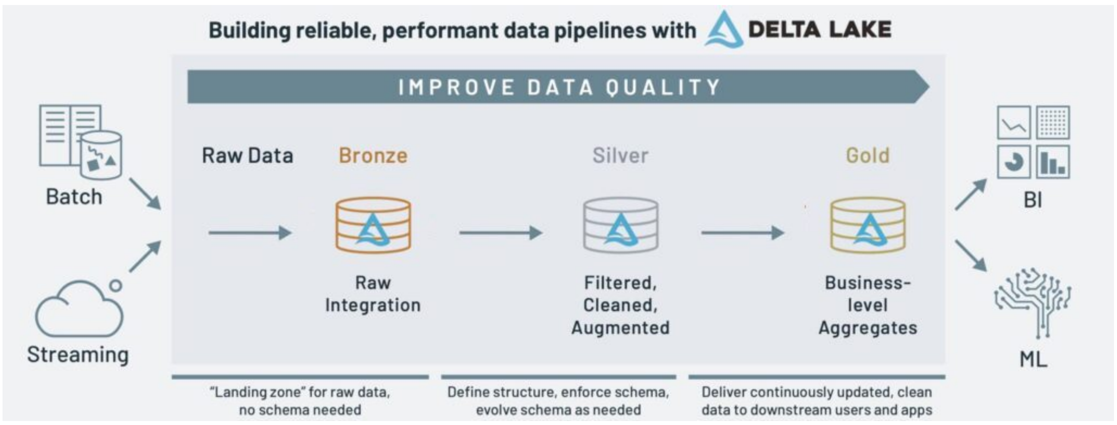

Building reliable, performant data pipelines with DELTA LAKE
IMPROVE DATA QUALITY
Batch
Streaming
Raw Data
Bronze
Raw Integration
Silver
Filtered, Cleaned, Augmented
Gold
Business-level Aggregates
BI
ML
"Landing zone" for raw data, no schema needed
Define structure, enforce schema, evolve schema as needed
Deliver continuously updated, clean data to downstream users and apps

Figura 11.1: Dynamic Ingestion Pipeline

One major enhancement would be the implementation of a medallion architecture, which organizes data processing into logical layers:

- **Bronze**: Raw and unstructured data directly ingested from the source.
- **Silver**: Cleaned and structured data, with management of formats, types and inconsistent values.
- **Gold**: Curated, business-ready data prepared for reporting, dashboards, and analytics.

This layered structure improves data traceability, facilitates troubleshooting, and enforces data governance across the entire pipeline.

Implementing a Metadata-Driven Approach

A key improvement would be to adopt a metadata-driven approach. Instead of hard-coding rules and logic, pipelines can be controlled by centralised configuration tables containing:

- Source and target table names
- Columns to include or exclude
- Filtering or transformation rules
- Ingestion settings (paths, formats, frequency, etc.).

This would reduce hard-coded code, facilitate multi-environment deployment (DEV, TEST, PROD), and significantly improve the reusability and maintainability of workflows. This approach is particularly suited to multi-source ingestion contexts with scalable data structures.

11.0.1 Integration of Microsoft Data Activator for proactive detection

To take automation and responsiveness further, integrating Microsoft Data Activator would be a strategic move. This tool allows you to monitor data in Microsoft Fabric in real time and automatically trigger actions when a specific event or condition is detected (abnormal value, lack of update, etc.).

For example, if a significant discrepancy in the data or a missed deadline in a pipeline is detected, Data Activator could automatically send an alert, trigger a fix, or notify a specific team. This would strengthen proactive governance and reduce response time to incidents.

11.1 Incorporating Machine Learning for Anomaly Detection

As data volumes increase, traditional rule-based checks may not be sufficient for detecting subtle issues. A potential evolution of the platform would be to integrate **machine learning models** that detect data anomalies. These models could be trained on historical data and alert BI teams to irregularities in ingestion volume, latency, or content.

Capitolo 12

Conclusion

This report gives an overview of the data ingestion pipelines I created during my Thesis at MSC Technology Italia. I worked with different tools, from classic ETL solutions like Informatica PowerCenter to more modern platforms like Azure Synapse and Microsoft Fabric.

During this project, I solved many problems. I made sure data was consistent, built flexible and reusable pipelines, and handled both batch and real-time data. I also set up automated processes with good monitoring. I always focused on making the pipelines scalable, fast, and easy to maintain.

This project also taught me how to manage the full data process, from extraction to deployment. I followed best practices and adapted to the tools used by the company.

After used Informatica PowerCenter Azure Analytics, and Microsoft Fabric tools for data ingestion, it's evident that Microsoft Fabric is the most adapted for the company's needs. It brings all data services into one place and is fully cloud-based. Thanks to OneLake, all data is stored in a single and unified storage layer, which avoids duplication and makes access faster and easier. The platform is simple to use, well integrated with Power BI, and fits well with the existing Microsoft ecosystem. It also reduces the number of tools needed, which makes development and maintenance easier.

Beyond the technical implementations, this project also gave me valuable experience in managing data workflows end-to-end from extraction and transformation to deployment, versioning, monitoring, and discrepancy fixing. I applied best practices in data engineering while continuously adapting to the evolving technological landscape of the organization.

In short, this project not only allowed me to contribute meaningfully to the modernization of MSC Technology Italia's data infrastructure but also helped me grow as a data engineer technically, methodologically, and professionally. I look forward to build a pipeline with metadata-driven initiatives.

Bibliografia

[1] Informatica LLC. *Informatica PowerCenter 10.x Documentation.* ETL tool used for data ingestion workflows. 2023. URL: https://docs.informatica.com/data-integration/powercenter.html.

[2] Microsoft Docs. *Change Data Capture (CDC) in SQL Server.* Official documentation for CDC in SQL Server. 2023. URL: https://learn.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server.

[3] Microsoft Docs. *SQL Server Import and Export Wizard.* Used to transfer historical data to the ODS. 2023. URL: https://learn.microsoft.com/en-us/sql/integration-services/import-export-data/import-and-export-data-with-the-sql-server-import-and-export-wizard.

[4] Microsoft Docs. *Azure Synapse Pipelines Documentation.* Used to design and orchestrate scalable ingestion pipelines. 2024. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/pipelines/overview.

[5] Broadcom Inc. *Automic Automation Documentation.* Used to orchestrate and schedule Informatica workflows. 2024. URL: https://docs.automic.com.

[6] Microsoft. *What is Microsoft Fabric?* https://learn.microsoft.com/en-us/fabric/get-started/. 2024.

[7] Microsoft. *Send notifications to Microsoft Teams from Azure Data Factory.* https://learn.microsoft.com/en-us/azure/data-factory/how-to-send-notifications-to-teams?tabs=data-factory. 2024.

[8] Microsoft. *Use Eventstream in Microsoft Fabric to process real-time data.* https://learn.microsoft.com/en-us/fabric/real-time/eventstream-overview. 2024.

[9] Microsoft. *Kusto Query Language (KQL) documentation.* https://learn.microsoft.com/en-us/azure/data-explorer/kusto/query/. 2024.

[10] Elementl. *Dagster Documentation.* https://docs.dagster.io/. 2024.

[11] Broadcom Inc. *Automic Automation Documentation.* `https://docs.automic.com/documentation/webhelp/english/all/components/DOCU/21.0/index.htm`. 2024.

[12] Databricks. *What is the medallion architecture?* `https://www.databricks.com/glossary/medallion-architecture`. 2024.

[13] Microsoft. *Get started with Microsoft Data Activator.* `https://learn.microsoft.com/en-us/fabric/data-activator/overview`. 2024.