Microservice Data Warehouse with Dynamic Semantic Workflows

Adam Žák[1], Martin Bobák[2]* and Ladislav Hluchý[2]

[1]Institute of Informatics, Slovak Academy of Sciences, Dúbravská cesta 9, 845 07 Bratislava, Slovakia
e-mail: xzaka@is.stuba.sk

[2]Institute of Informatics, Slovak Academy of Sciences, Dúbravská cesta 9, 845 07 Bratislava, Slovakia
e-mail: {name.surname}@savba.sk

*Corresponding author: M. Bobák (e-mail: martin.bobak@savba.sk)
https://orcid.org/0000-0002-2905-4999

Abstract: The paper proposes a microservice-based data warehouse with a semantic layer to enhance system efficiency. Semantics in the data warehouse offers improved business process modeling, better resource utilization, and streamlined service management. Our solution integrates semantics into a Dockerized data warehouse built with Ruby, using Neo4j and Redis for ontology and tags. System evaluation demonstrates that semantics brings benefits to both users and administrators. Ontologies enable precise pipeline selection, while tags require fewer resources. Additionally, semantics facilitate automatic deployment and optimize resource usage during idle periods. With dynamic processing pipelines and automation, the system becomes less sensitive to input errors and is better prepared to handle new data sources.

Keywords: microservice; data warehouse; semantic workflows

1 Introduction

Microservices are one of the current leading trends in software development, offering advantages like scalability and efficient resource management [1] [2], but they also bring challenges in data processing within distributed architectures [3] [4] (e.g., correct data routing can be more complex). The proposed methodology explores the integration of semantics into a microservice system to improve its efficiency and resource utilization. Semantics is mainly used to understand the context and meaning of the stored data [5]. The proposed approach uses semantic techniques to dynamically analyze incoming data and ensure appropriate service

activation within microservice-based data warehouses. Semantic annotation improves microservice usage, making systems more comprehensible and efficient. Traditional business modeling is obsolete and costly, whereas dynamic semantic workflows offer flexibility and reusable code without hard-coded service configurations [6]. Our approach follows a data-driven microservice approach, addressing challenges in ETL pipelines by dynamic pipelines for improved data processing.

2 Background and Related Work

2.1 Data Warehouses and Semantics

The main purpose of data warehouses is to integrate, store, and process data from heterogeneous sources [7], serving as key decision-support tools [8]. By managing large, non-volatile data, they provide insights across different time periods, aiding strategic decision making. While primarily handling structured data, approaches for unstructured data also exist [9]. Additionally, data warehouses act as research repositories [10] (e.g. in medical fields, they consolidate DNA data, genetic predispositions, and past illnesses to support further discoveries).

Semantics brings many advantages in different fields of computer science, including recommendation systems [11], document indexing [12], and the semantic web [6], which can bring benefits to data warehouses. Unlike traditional relational databases, semantic data warehouses store and process data and provide semantic annotations, enriching data management and analysis. However, traditional ETL pipelines do not inherently support semantic information, thus requiring architectural modifications [13]. This challenge is addressed by integrating a semantic layer, typically applied after data processing and storage, rather than within the ETL pipeline.

2.1.1 Integration of Semantics in Data Warehouses

Enabling data integration is a key aspect of semantic usage with connection to data warehouses. Semantic integration functions on a higher system level, enabling the creation of a single scheme [13] for multiple schemes or data sources. Semantic integration can understand the data and combine them by their contextual message or meaning. A unified layer enables users to query data from multiple sources at once, which enables access to a wider data range. However, achieving perfect integration is not possible and brings problems that need to be solved. Different table names, column names, or data formats can cause problems in integration. In the past, manual approaches were used for comparison of schemes.

These approaches were based on rules, which were shown to be problematic for some sort of standardization. Implementation of these rules for each specific problem was difficult and demanding. As a result, semantic integration based on these rules was not achieving desired levels of accuracy.

2.1.2 Semantic Layer in Data Warehouses

The **semantic layer** transforms data representation from a model-based approach to a business-oriented approach, enhancing readability and analysis. This shift aligns data structure with operational and analytical needs, making business entities and their relationships more intuitive. It also unifies data across an organization, improving decision making [14].

Key Advantages of the Semantic Layer

- **Accessibility**: Improves data availability by enabling retrieval through semantic markers (e.g., *users*) instead of complex queries.
- **Query Speed**: Enhances performance using views, materialized views, or pre-executed queries, reducing redundant computations.
- **Single Source of Truth**: Ensures all departments work with the same semantic definitions, fostering consistency.
- **Security & Authorization**: Supports access control mechanisms, restricting sensitive data (e.g., *salaries*, *addresses*) to authorized users.

The **semantic layer** can also handle pre-processing and annotation of structured and unstructured data, grouping them by meaning. It enables seamless integration of multiple data sources while simplifying relationships and processing. Additionally, a standardized semantic language across the organization prevents inconsistencies, ensuring clarity and reducing conflicts.

2.2 State of the Art

This section provides an overview and comparison of data warehouse solutions and business intelligence tools (BI), highlighting their integration of semantic layers. Data warehouses fall into two main categories: cloud (widely adopted) and local (less common due to maintenance challenges). Effective management is essential but time-consuming.

Semantic layers enhance data analysis and comprehension, though not originally part of data warehouses. Their widespread adoption varies across systems, mainly applied after the data is imported. However, SETL introduces a unique approach, directly influencing data, aligning with this paper's approach.

AtScale [15] integrates a **semantic layer** into data warehouses, enhancing analysis and decision making. It employs:

- **Dimensional modeling**: Uses a star schema to structure data around business concepts, improving clarity.
- **Metadata management**: Centralized metadata storage ensures consistency across the organization.
- **Data virtualization**: Enables virtual data views, reducing physical movement and improving performance.

AtScale is compatible with major data warehouses (Google BigTable [16], Redshift [17], Azure [18], Snowflake [19]) and offers **unified SQL access**, streamlining data management.

Dremio [20] is a data warehouse with a semantic layer that organizes data into semantic spaces, allowing multiple groups with different access rights. The data warehouse has the following key features:

- **Semantic Metadata Catalog**: Stores metadata on tables, views, data types, and relationships, enhancing data comprehension.
- **User Accessibility**: Simplifies data access for non-developers by structuring data in a more business-oriented way.
- **Semantic Grouping**: Groups data by meaning to create new, easily interpretable tables.
- **Limited cloud support**: Cloud-native performance could be improved.
- **Limited version management**: Upgrades can be challenging, making rollbacks complex.

Looker [21] is a data analysis and visualization platform with a semantic layer, focusing on shifting data from a model-based to a business-oriented approach. It connects to existing cloud or local databases rather than functioning as a standalone data warehouse. Now part of Google Cloud [22], Looker integrates seamlessly with Google BigTable. Its semantic model categorizes data, enhances business metrics, and simplifies analysis by acting as a bridge between data sources and analytical tools. Looker has the following key features and limitations:

- **Strengths**: Scalable, flexible, and well-integrated with Google services.
- **Challenges**: Limited visualization capabilities and high pricing, especially for smaller businesses.
- **Dependency**: Performance relies on the efficiency of the underlying data warehouse—slow databases lead to slow Looker queries.

Datameer [23] is a cloud-based data integration and processing tool designed for user-friendly data exploration and analysis. It supports Snowflake databases by default but offers alternatives at additional costs. Datameer has the following key features:

- **Data Transformation**: Converts raw data into a structured format for storage in a data warehouse.
- **Semantic Layer**: Enables grouping, documentation, and categorization, improving data integrity and accessibility.
- **User Accessibility**: Aims to be intuitive for non-technical users but has complex UI and limited documentation, which may hinder usability.

Semantic Data Warehouse (SETL) [24] integrates a semantic layer into the ETL (Extract, Transform, Load) process, unlike traditional semantic approaches that focus only on categorization. Evaluated with datasets from Danish Agricultural and Business domains, this system directly transforms and stores data in RDF Triple format within the data warehouse, ensuring semantic consistency at the storage level. SETL has the following key features:

1) **Semantic Data Extraction**: Collects data from multiple sources (databases, XML, web services) based on semantic metadata.
2) **Semantic Data Transformation**: Converts data into RDF format for better interoperability and consistency.
3) **Semantic Data Loading**: Stores transformed data in a semantic data warehouse, enabling advanced querying and analysis.

Unlike commercial solutions, SETL modifies data at the processing stage, ensuring semantic accuracy before storage, an approach similar to the one used in this paper.

3 Design of Semantic-driven Data Warehouse

Different types of data can be understood by the system in various ways. In the context of this system, it means the ability to figure out what is in the data without any prior knowledge. Factors such as file type, coding method, or size can assist in decision making by the system. Two main approaches will be used for this purpose: first, the approach involving specific data identifiers and other key file information is managed within an ontology system. Secondly, a similar approach uses tags instead of an ontology to track the data and its movement through the system. The system will implement both semantic approaches.

The approaches will generate processing pipelines that will be used for processing in two different ways: classical and dynamic. In the classical method, all the required containers will be deployed and waiting for the data to process them. After

processing, they will return to idle mode and wait for data once again. The dynamic option will load the containers needed for processing from the pipeline, start instances of these containers, process the required data, and then stop these containers.

This covers all instances where semantics will be employed to manage the system: firstly, in data recognition, secondly, in pipeline creation, and lastly, in container management. Each part of the system has its specific perspective of implementation and will be described in its respective section.

3.1 Data Recognition

The first step to accurately generate effective processing pipelines or to correctly initiate containers based on pipeline requirements is to identify the data the system is processing. Key data identifiers are collected from the observed data to achieve this. This system will not implement any automatic collection of these identifiers, but it presents an interesting option for future research and development.

This new system component removes the burden on users who previously needed to manually select the origin of the data they were importing in the original design of the data warehouse. This part of the system will perform this task automatically based on the semantic understanding of the data. To accomplish this, a new service will be introduced that is responsible for collecting metadata. This design suggests these metadata:

1) **file endings**: easily obtainable and can greatly narrow down the number of possible suppliers
2) **mime-types:** offer option to understand data format more precisely, as file endings can be lost/edited and overall much more misleading
3) **charset:** identifies coding used in the file, can help distinguish between two similar data files
4) **country/language:** identifying the country or region will help separate different groups of data files
5) **header columns:** header columns are representing specific data stored in files, and as such they should represent most unique identifiers, even though its not true for all column names (PSC, Address, etc.)

One possible option is to use a containerized version of Apache Tika, which will be tasked with metadata collection. Apache Tika is an excellent choice because of its ability to extract a wide range of metadata from various file types, such as xlsx, csv, xml, and txt in our case. It provides comprehensive and robust community support, offering numerous deployment options, making it an ideal docker container for this system.

3.2 Ontology

Crucial part of this solution is the ontology. As mentioned in the state of the art, there are many ways to implement the ontology. The current state of the art in computer representations is graph representation. Native ways of representing graph data are graph databases, which offer a more intuitive way of representing these types of data. There are many existing solutions that can be used in this system such as GraphDB [25], Neo4j [26], CosmosDB [27], RedisGraph [28], and so on. Each of these tools has its own strengths. The proposed solution opts to use Neo4j, which is the leading graph database. It will implement the semantics module that brings semantics standards such as RDF to the native graph database, which is not designed with semantics in mind. Neo4j will be responsible for finding neighbors based on relations among nodes.

The ontology will be implemented using a docker container, as expected from the system. For communication, the Ruby library will be used on the open port of Neo4j. The advantage of this tool is the graphical management tool accessible at port 7474, which offers a clear management tool for the addition or edit of new nodes or removal of unwanted nodes or relationships.

This service will be responsible for managing queries for data recognition and also for managing possible processing paths for dynamic pipelines. The service then becomes a key part of the semantic system and, as such, will require proper scaling and resource utilization, so it will not be a bottleneck of the whole system.

3.3 Tags

Same approach is applied in the proposed solution for Tags management. The proposed plan is to use the Redis stack server container for tag memory storing and index searching. Redis is selected as the leading option in the field of in-memory databases with support for tags. This support brings benefits and functionality for tag matching searching and indexing. The system will be communicating with the Redis container through the Ruby library implemented in the *tags_manager*. This manager will manage the queries creating, manage index creating and searching, manage results, and process them in the form that is used in the rest of the system.

This part will include some sort of management UI, for tag editing and overall CRUD operations, it can be implemented in the *admin_service* container as a unified form of management and communication point. Another option is to include the RedisInsight container, which can be used to manage Redis and, it can also manage the data stored there directly.

3.4 Semantic Graphs

Various approaches to representing semantic knowledge or ontology come with their own complexities. The widely adopted method for ontology representation is typically the graph model. Its popularity arises from its effective visualization of connections and nodes, which aligns well with human cognitive processes and understanding. This makes it easier to understand, show detailed knowledge, and explain complicated relationships. However, a notable challenge lies in the maintenance of this representation. Graphs can expand significantly, introducing complexities when performing routine tasks such as adding new nodes. Navigating through an extensive array of nodes to identify new relationships poses a substantial difficulty. Although automation can address certain aspects, human oversight remains essential to ensure the accurate and meaningful representation of knowledge within the ontology.

However, tags offer a different approach altogether. They provide a simple way of showing semantics, where important information is communicated mainly through keywords. This simplicity greatly reduces the complexities of maintaining tag representations. It enables for easier automated node management, giving more flexibility with fewer strict rules. However, tags can have problems to represent all knowledge about data in proper way, for example, relationships can represent challenge. These "small" information, lost in tags creation because they were lost in tags creation can be crucial for semantics in some situations. Without these connections, specific information can lack integrity and context. The lack of clear visualizations and representations in the tag representation can create challenges for those trying to fully understand its complexities and information. This can lead to lesser accuracy in conclusions created from tags in comparison to ontology with much more complex structure.

3.5 Semantic Data Warehouse Architecture

The resulting architecture will look as demonstrated in Figure 1. This figure shows the planned structure of a data warehouse after the introduction of semantics. Each of the blocks represents a docker container deployed at least once (with the exception of processors and optional processors that do not have to be deployed when automatic deployment is set). Containers are divided into four main categories, divided by color. Each color has its own meaning:

- Blue containers are containers that were already existing in previous iteration of data warehouse, without the semantic layer (see [29]).
- Purple containers represent the tools used in this paper. They are used to process (Apache Tika), store (Redis/PostgreSQL) data more effectively, automate tasks for efficiency (Redis - background jobs) and manage semantic knowledge, in tags (Redis) and ontology (Neo4j) representations.

- Green containers represent three special managers, each of them responsible for management of some sort of process. The Tag Manager communicates with the tags stored in Redis and constructs paths. The same applies to the ontology manager that talks to the ontology and selects the best paths. Docker manager is responsible for communication with the Docker API, deployed on the host machine. It starts and stops containers required in processing pipeline, if automatic deployment is started.
- Red container is only for the semantic manager. This service is special because it manages the whole semantic processing in the system. It sends requests on selected semantic approaches, gathers data, communicates with the docker manager, and finally sends the data to the created processing pipeline.

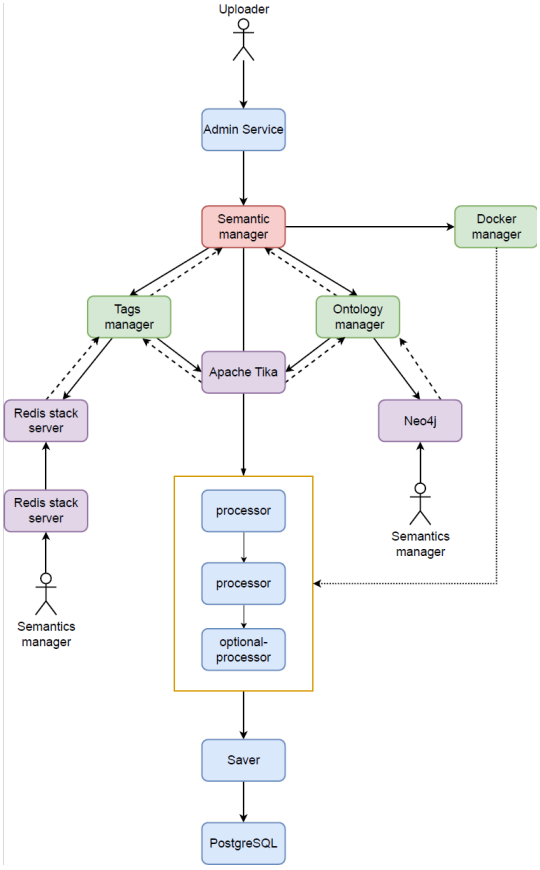

Figure 1
Semantic Data Warehouse architecture

The same logic is used also for the connections, which represent the communication among the containers. These are split into three different line types:

- Default line represents requests from one container to another.
- The dashed line is for the "return" request, which represents response to the request. This cannot be skipped, and if not obtained, system will not correctly continue in processing.
- The dotted line represents one special usecase, when automatic deployment is enabled. In that case, the Docker manager communicates with the processing containers in order to start/stop them. If this processing is not enabled, these requests will not be sent.

In general, the structure will be based on the semantics manager, which takes responsibility for the management of the system. In order to do this, it will also use a number of new services. Important containers are used for ontology, tags, and docker management. Thanks to them, semantics manager will be able to obtain information about data it is processing, and then select the best processing pipeline for successful processing and data storing.

4 Platform for Data Warehouse Deduplication

This section describes the steps required for the replication and realization of the system developed in this paper. It is divided into multiple sections aimed at new parts of the system. These chapters focus on the additional new services (managers) that are responsible for semantic implementation together with the tools that are used for their functioning. In addition, the deployment and the data model for semantics are shown and explained.

For a better understanding of the changes made in this paper, it is necessary to explain key concepts from previous paper done and outlined in [29] [30]. Previous paper aimed at the construction and testing of modular data warehouses based on microservice architecture. This system introduced containerized data pipelines, which allowed easier data sharing between multiple different endpoints and containers designated for data processing and storing. The primary components of the processing pipeline included Data Loaders, Processors, and Savers. Each of these parts of pipeline are designed for a specific task, which are explained below:

- Data Loaders were designed to download specific data (from APIs or other sources) or receive it through user input.
- Processors containers received raw, unprocessed data and transformed them into a ready-to-store version.
- Savers received processed data, determining where and how to store it while managing data storage.

Although the system worked as intended, a major drawback was its static architecture construction, which made changes to the system very difficult. Users had to identify the origin of the files they were uploading for successful processing. The system then works with the assumption that the data are of the origin, selected by the user. If the user does not know the origin or selects the wrong one, it creates errors and leads to data loss. The next problem is that the processing pipelines use a fixed architecture, lacking dynamic adjustments to satisfy diverse data needs. This is the challenge addressed in the current paper.

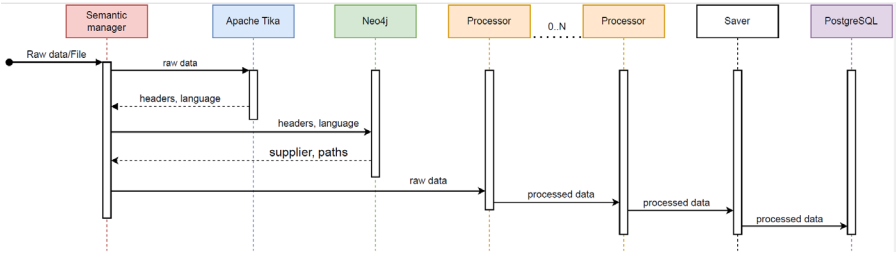

Figure 2
Sequence diagram which demonstrates flow of data in system

4.1 Semantic Manager

Central point of the newly introduced semantic layer is the semantic manager, which serves as the key point for all semantic management within the system. Thanks to the modular microservice architecture of the original data warehouse, this manager seamlessly integrates this layer into the solution as a new service (or containers in our case). The key difference between the original data warehouse and the new system using the semantic layer involves redirecting requests previously shared directly from data loaders to processors. Now, these requests are first routed to the semantic manager, which then constructs the best processing pipelines. This process is repeated for each request and creates a dynamic flow of data in the system, based on the identified supplier. This new processing flow is illustrated in Figure 3.

The first data routing displays the original flow of the data in the application, where a fixed processing pipeline was used for the data flow. The lower data flow demonstrates the new approach used in this paper. The new service, semantic manager, is colored green and is placed after data collection (yellow service/s). This service in the processing pipeline is very important for semantics operation, as it has to occur as soon as possible after data loading to identify given data.

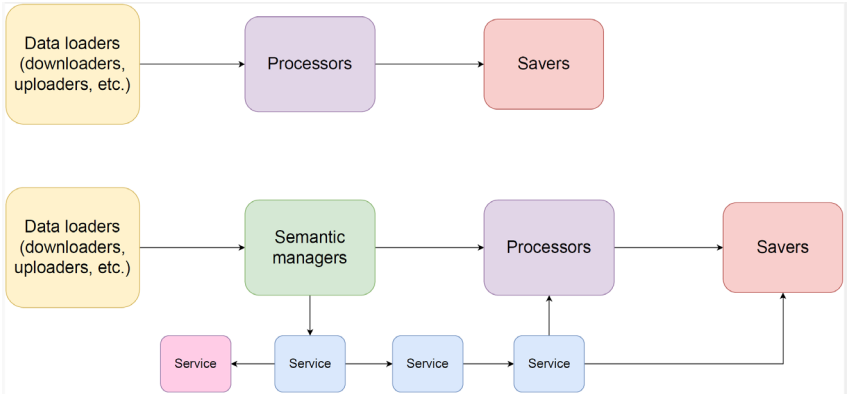

Figure 3
Original static processing pipeline (upper) and new dynamic processing pipeline (lower)

After successful identification of the received data, the next step for the semantics manager is to create an appropriate processing pipeline that has an optimal combination of best services and length. One of possible processing pipelines is mirroring the original static processing path, where data are flowing from the identification to processor, and from processing to storing. Other alternative options, colored blue, can bring other services into play, which can be placed at different places in the pipeline. This new pipeline can completely bypass processors, can be placed between them, or even can go directly to saver if that is desired behavior of specific data. Another flow option is represented by the pink service, which serves as the final step in the pipeline, providing an additional choice.

Searching of correct data flows is based on provided parameters for request, which can be changed and selected by user, but need to be set in semantics as an option. Possible processing paths are defined in the data ontology, which is represented as a graph in Neo4j.

This operation returns an array of conditions that should be fulfilled by the processing pipeline if possible. After that, the system produces all possible paths; for each path, an array of conditions is constructed, which are collected into an array, which is then iterated over, and checks how much of a requested conditions from request are inside of the given array. Counts the number of hits and stores the index of the array with most hits (hits will be explained later). The resulting path on given index is then returned for additional processing.

4.2 Tag Manager

This paper implements an alternative semantic approach, which is using tags. Tags are, in a very simple way, simple strings that are used as values for identification and recognition. They usually contain key entity characteristics they represent.

While it is preferred to have unique tags, it is not a strict requirement. The main advantage and significance of tags is in their count. More tags enable for more precise identification as a larger number of attributes can be matched. The bigger number of tags also comes at the price of higher memory and storage consumption.

For this implementation, the system uses Redis as an in-memory storage solution because of its superior speed compared to traditional data storage. This choice is particularly advantageous, as it works with a relatively smaller amount of data stored in memory, minimizing the impact on overall memory consumption. When the application is not active, the data are persistently stored in a text file, which is loaded upon the application startup. The representation of the data is as follows:

For this implementation, system uses Redis as an in-memory database, mainly due to its high processing speed because data are stored in the memory, not in the storage. This tool is very beneficial, mainly because the number of tags for suppliers is based on the number of suppliers, which are not an infinitely growing set of values. Data are loaded into Redis from the text file which contains Redis commands which are then executed. For example:

HSET <id> name <name> tags <tags.....>

The whole query is constructed as a full text search on supplierIndex where the tags field contains given attributes and the payload also has the attributes, return score, and also uses a special scorer named HITS_SCORER which is explained later. The query is then executed, and the results are processed.

When it comes to the stop-words, systems approach is to not eliminate them. Stop words are something that usually occurs in the normal text, but this system works with the column headers and file types, which often do not contain stop words and automatic removal could potentially remove something important from the tag. Also, the index specifically requests HASH data format instead of JSON data storing, with "supplier" prefix for the identifier. At the end, the command setups the data scheme, which setups name field as TEXT and tags field as TAG field.

System stores endpoints data in classical key-value format. This simple approach is based on idea that one endpoint has one condition that is specific for it and can be identified by that condition. This information enables more efficient structure in this key-value format

One key part of the solution already mentioned before is custom scorer. Redis enables tag searching and also provides scoring for the obtained results. It offers multiple scorers, which can be used, but all of them are used for full text processing and things such as TF-IDF, which is default scorer, are not optimal for this task as it calculates many unnecessary operations and also lowers and boost score based on terms frequency, etc. which is opposite to what this system needs.

For this reason, the system has its own custom scorer named HITS_SCORER implemented in C. It uses payload option for data loading and it simply compares strings received in payload part and data loaded from Redis.

5 Experiments and Evaluation

The proposed evaluation examines the impact of the semantic layer on system resource utilization, performance, and accuracy. Performance tests analyze potential negative effects, ensuring implementation costs do not outweigh benefits.

5.1 Testing Environment

To ensure a consistent and reproducible evaluation, the experiments were conducted on a system with the following specifications. Table 1 outlines the key hardware and software parameters, including the operating system, CPU details, memory capacity, and Docker and Docker Compose versions.

Table 1
Testbed specification

OS	Ubuntu 22.04.4 LTS
Architecture	x86_64
CPU	Intel i5-7300HQ
Cores	4 @ 3.500GHz
Memory	16GB
Docker version	26.0.0
Docker-compose version	2.3.3

5.2 Platform Performance

The additional semantic layer brings new overhead to the system. In this part of the evaluation, the performance of the system is compared in setup with semantic layer and without semantic layer, between different semantic approaches (see Figure 4).

Each group in the graph represents three tests for a specific data load, organized by series. In particular, the difference between the green and yellow bars, which signify processing times for uploading a single invoice (with green indicating the default system and yellow representing the semantic system using ontology), is relatively small. In this comparison, the difference is only one second, with the average processing time for the default system being 8 seconds and for the semantic system being 9 seconds.

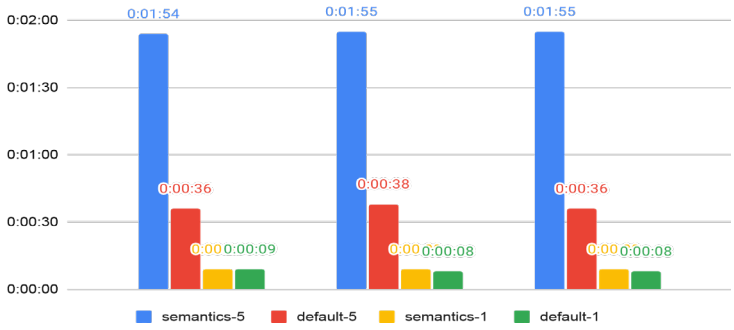

Figure 4

Processing time for default system setup and semantic data warehouse using ontology

5.3 Platform Memory Utilization

Resource utilization carried out on both semantic approaches used in the system. The results are compared among these approaches and also with the results gathered from the original data warehouse without the use of semantics.

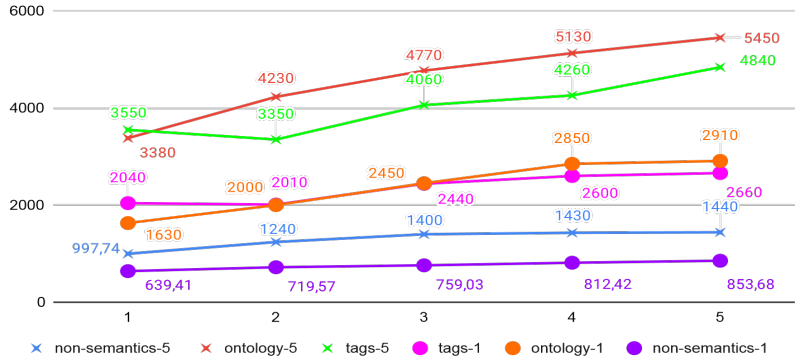

Figure 5

Memory consumption of system over time in different configurations and different number of uploads. Vertical axis values are in megabytes and horizontal axis describes order of uploads.

Figure 5 shows multiple lines with different colors and also two groups, one with crosses and one with dots or points. Lines with dots represent memory consumption for different setups of data warehouse and semantics, and both for a single uploaded invoice. The lines with crosses represent the same setups but for five uploaded invoices at once. The lowest line in the graph (purple) represents the data warehouse without any form of semantics and without additional containers running for semantics. The system in this configuration uses the lowest amount of memory through multiple uploads of one invoice. The system in such a setup increased its memory consumption by around 200 megabytes. The next blue line is for the same

setup, but this time 5 invoices are uploaded in each step. Memory requirements are higher, but this is for obvious reasons, as the system works with much more data, which consumes more memory.

5.4 Evaluation of Dynamic Pipeline Construction

This experiment evaluates the accuracy of semantic data understanding, a critical aspect of system functionality. If accuracy is insufficient, the system fails to operate correctly. The system assesses pipeline selection accuracy based on provided conditions, ensuring optimal processing for each semantic type.

This subsection introduces conditions which are values that are passed to the system when receiving data files. They can represent options for the user to interact with the system; for example, when they want to send email, we represent this option with *send_email* string in the input field on the upload page. Conditions also represent marks given by the system to specific data types. For example, when users want the system to schedule CRON job when data are received, they can add this condition to the data node (in both Tags or Ontology) and as such it will automatically add itself to the system.

The conditions are then used in the correct path creation/selection. Both ontology and tags use different approaches to pipeline creation/selection. Tags are using a much simpler approach where the processing path is not selected among existing options, but rather created based on the values received in the data recognition and conditions. Ontology, on the other hand, rather selects the best option based on identification and conditions. It holds pre-existing network of possible processing paths in graph. For identified supplier are then selected all possible paths that can be used. These are then compared, and the system selects best, based on the number of hits. This means that ontology can select processing pipeline that contains endpoints that are not in conditions or selected pipeline can miss some of these endpoints for situations where better processing pipeline is not possible.

Figure 6 shows the experiment results. Each option is described with number and has two bars describing hits percentage for different values calculated and resulting (green) average of these two values. In this scenario, system has 7 possible paths. The numbers on the x-axis are used for describing the possible paths, as these would be very long for simple showing.

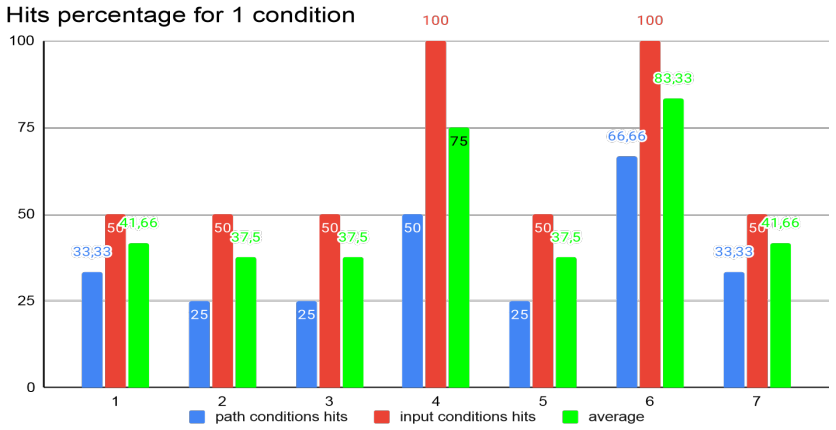

Figure 6

Comparison of hits percentage for path conditions and input conditions (1 received from user) and their resulting average score

Conclusions

The integration of a semantic layer into microservice-based data warehouses shows a significant advancement in the automation and adaptability of data processing pipelines. By enabling context-aware service selection and reducing manual configuration, the system enhances both usability and operational efficiency. The paper presented and evaluated two semantic approaches: 1) ontology-based and 2) tag-based.

The evaluation demonstrated that while the semantic layer causes tolerable overhead, it offers considerable gains in pipeline flexibility and data recognition accuracy. The ontology-based method enables deeper semantic reasoning and precise workflow alignment, but it comes at the cost of increased system complexity and performance limitations, particularly due to the use of Neo4j as the backend. In contrast, the tag-based approach, though less expressive, proved to be lightweight and effective in typical usage scenarios.

The experiments show that the choice of semantic backend has a significant impact on performance and scalability. Neo4j, while powerful for general-purpose graph processing, is not optimized for semantic reasoning and RDF triple management. Future work will address this limitation by exploring alternative triplestore systems, such as GraphDB or CosmoDB, which offer native support for SPARQL and enhanced inference capabilities. This improvement can optimize ontology-driven processing and increase the accuracy and efficiency of dynamic pipelines.

While the paper is primarily focused on structured data sources such as invoices, the presented approach is modular and extensible. By incorporating metadata extraction tools like Apache Tika and expanding the semantic rules or tag sets, the system can be extended to accommodate unstructured or semi-structured data

formats such as PDFs, emails, XML documents, or sensor logs. Future work will explore dynamic recognition and processing of these formats, potentially integrating natural language processing (NLP) techniques for context extraction and semantic tagging.

Acknowledgment

Funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I05-03-V02-00055. This work was also supported by APVV grant no. APVV-23-0430, and VEGA grant no. 2/0131/23.

References

[1] M. Bobak, L. Hluchy and V. Tran, "Abstract model of k-cloud computing," *Proceedings - IEEE 11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014,* pp. 710-714, 2014

[2] M. Bobak, L. Hluchy and V. Tran, "Methodology for intercloud multicriteria optimization," *Proceedings - IEEE 12th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2015,* pp. 1786-1791, 2015

[3] M. Bobak, B. Somoskoi, M. Graziani, M. Heikkurinen, M. Hob, J. Schmidt, L. Hluchy, A. Belloum, R. Cushing, J. Meizner, P. Nowakowski, V. Tran, O. Habala and J. Maassen, "Reference exascale architecture," *Proceedings - IEEE 15th International Conference on eScience, eScience 2019,* pp. 479-487, 2019

[4] J. Meizner, P. Nowakowski, J. Kapala, P. Wojtowicz, M. Bubak, V. Tran, M. Bobák and M. Höb, "Towards exascale computing architecture and its prototype: Services and infrastructure," *Computing and Informatics,* Vol. 39, No. 4, p. 860-880, 2021

[5] A. Formica and F. Taglino, "Semantic Similarity in a Taxonomy by Refining the Relatedness of Concept Intended Senses," *Computing and Informatics,* Vol. 42, No. 1, p. 191-209, 5 2023

[6] F. Z. Amara, M. Hemam, M. Djezzar and M. Maimor, "Semantic Web and Internet of Things: Challenges, Applications and Perspectives," *Journal of ICT Standardization,* Vol. 10, No. 2, pp. 261-292, 2022

[7] A. Dhaouadi, K. Bousselmi, M. M. Gammoudi, S. Monnet and S. Hammoudi, "Data warehousing process modeling from classical approaches to new trends: Main features and comparisons," *Data,* Vol. 7, No. 8, p. 113, 2022

[8] A. Al-Okaily, M. Al-Okaily, A. T. -. E. J. o. … and u. 2022, "An empirical study on data warehouse systems effectiveness: the case of Jordanian banks

in the business intelligence era," *EuroMed Journal of Business,* Vol. 18, No. 4, pp. 489-510, 10 2023

[9] j. Zemnickis, "Data Warehouse Data Model Improvements from Customer Feedback.," *Baltic Journal of Modern Computing,* Vol. 11, No. 3, pp. 475-499, 2023

[10] S. Visweswaran, B. Mclay, N. Cappella, M. Morris, J. T. Milnes, S. E. Reis, J. C. Silverstein and M. J. Becich, "An atomic approach to the design and implementation of a research data warehouse," *Journal of the American Medical Informatics Association,* Vol. 29, No. 4, pp. 601-608, 2022

[11] S. Jain, "Product discovery utilizing the semantic data model," *Multimedia Tools and Applications,* Vol. 82, No. 6, pp. 9173-9199, 3 2023

[12] A. Sharma and S. Kumar, "Machine learning and ontology-based novel semantic document indexing for information retrieval," *Computers & Industrial Engineering,* Vol. 176, p. 108940, 2023

[13] R. P. Deb Nath, O. Romero, T. B. Pedersen, K. Hose and P. Cudre-Mauroux, "High-level ETL for semantic data warehouses," *Semantic Web,* Vol. 13, No. 1, pp. 85-132, 5 2021

[14] R. K. Stirewalt and M. Búr, "The RAI Way: A Technical Analysis and Design Method for Building Enterprise Semantic Layers," *Lecture Notes in Business Information Processing,* Vol. 482, pp. 74-79, 2023

[15] "GigaOm Sonar Report for Semantic Layers and Metrics Stores," *Analyst Report,* 2024

[16] V. Pamisetty, S. Rao Challa, V. Bhardwaj Komaragiri, K. Challa, K. Chava and A. Professor, "Scalability and Efficiency in Distributed Big Data Architectures: A Comparative Study," *Metallurgical and Materials Engineering,* Vol. 31, No. 3, pp. 40-49, 3 2025

[17] P. Borra, "An overview of cloud data warehouses: Amazon Redshift (AWS), Azure Synapse (Azure), and Google BigQuery (GCP).," *International Journal of Advanced Research in Computer Science,* Vol. 15, No. 3, pp. 23-27, 4 2024

[18] A. Kumar, A. Mishra and S. Kumar, Architecting a modern data warehouse for large enterprises: Build multi-cloud modern distributed data warehouses with azure and AWS, Apress Media LLC, 2023, pp. 1-368

[19] R. Steelman, "The Snowflake Data Cloud," Apress, Berkeley, CA, 2024, pp. 7-14

[20] T. Shiran, J. Hughes and A. Merced, Apache Iceberg: The Definitive Guide: Data Lakehouse Functionality, Performance, and Scalability on the Data Lake, O'Reilly Media, 2024

[21] P. Borra, "Evaluation of Top Cloud Service Providers' Bi Tools: A Comparison of Amazon Quicksight, Microsoft Power Bi, and Google Looker," *International Journal of Computer Engineering and Technology (IJCET),* Vol. 15, pp. 150-156, 6 2024

[22] S. R. Sukhdeve and S. S. Sukhdeve, Google Cloud Platform for Data Science: A Crash Course on Big Data, Machine Learning, and Data Analytics Services, Apress Media LLC, 2023, pp. 1-219

[23] S. Kaur, A. Bala and A. Garg, "Data Wrangling Dynamics," *Data Wrangling: Concepts, Applications and Tools,* pp. 53-70, 7 2023

[24] R. P. Deb Nath, K. Hose and T. B. Pedersen, "owards a programmable semantic extract-transform-load framework for semantic data warehouses," *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP,* pp. 15-24, 2015

[25] D. Greco, F. Osborne, S. Pusceddu and D. R. Recupero, "Modelling big data platforms as knowledge graphs: the data platform shaper," *Journal of Big Data,* Vol. 12, No. 1, pp. 1-64, 2025

[26] Y. Chen and X. Xing, "Constructing Dynamic Knowledge Graph Based on Ontology Modeling and Neo4j Graph Database," *2022 IEEE 5th International Conference on Artificial Intelligence and Big Data, ICAIBD 2022,* pp. 522-525, 2022

[27] N. Freitas, D. Vaqueira, A. Rocha, J. Barata, F. Serrano, S. L and M. M, "A Novel Pipeline for Data Management and Analysis that Integrates Data Lakehouse Architecture into the Aeronautics Industry," *International Conference on Innovative Intelligent Industrial Production and Logistics,* pp. 410-424, 2024

[28] N. Kanakaris, D. Michail and I. Varlamis, "A Comparative Survey of Graph Databases and Software for Social Network Analytics: The Link Prediction Perspective," in *Graph Databases*, CRC Press, 2023, pp. 36-55

[29] A. Zak and M. Bobak, "Modular e-Commerce Data Warehouse using Microservices," *Proceedings of the 31st International Conference on Cybernetics and Informatics, K&I 2022,* pp. 1-6, 2022

[30] J. Hlavačka, M. Bobák and L. Hluchý, "Big Data Deduplication in Data Lake," *Acta Polytechnica Hungarica,* Vol. 21, No. 11, pp. 307-328, 2024

[31] A. Calatrava, H. Asorey, J. Astalos, A. Azevedo, F. Benincasa, I. Blanquer, M. Bobak, F. Brasileiro, L. Codó, L. del Cano, B. Esteban, M. Ferret, J. Handl, T. Kerzenmacher, V. Kozlov, A. Křenek, R. Martins, M. Pavesio, A. J. Rubio-Montero and J. Sánchez-Ferrero, "A survey of the European Open Science Cloud services for expanding the capacity and capabilities of multidisciplinary scientific applications," *Computer Science Review,* Vol. 49, p. 100571, 2023

[32] M. Staňo, L. Hluchý, M. Bobák, P. Krammer and V. Tran, "Federated Learning Methods for Analytics of Big and Sensitive Distributed Data and Survey," *Proceedings - IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI),* pp. 000705-000710, 2023

[33] M. Barbella and G. Tortora, "A semi-automatic data integration process of heterogeneous databases," *Pattern Recognition Letters,* Vol. 166, pp. 134-142, 2023