A Cloud-Native Lakehouse Architecture for Using Knowledge Graphs in Aeronautical Information Management

Bashar Ahmad and Christoph G. Schuetz
Institute of Business Informatics – Data & Knowledge Engineering
Johannes Kepler University Linz
Linz, Austria
{ahmad,schuetz}@dke.uni-linz.ac.at

***Abstract*—A knowledge graph (KG) represents real-world entities as well as their properties and relationships in a machine-readable format, which can be employed in decision support systems across various domains. For example, in ATM, a KG can contain information about various aspects of the current state of the air traffic network, including infrastructure, important events, and flight trajectories. Traditional monolithic KGs face scalability challenges as the volume, variety, and velocity of the data increase, limiting real-time responsiveness. However, many applications require access only to subsets of a KG rather than to the entire KG at once. For example, a pilot briefing for a particular flight within Central Europe does not need information about the entire European air traffic network. Based on this observation, we propose a cloud-native data lakehouse architecture for KG management, optimized for ingesting and indexing large volumes of a variety of data arriving at high velocity. The main contribution of this paper is the design of a modular and scalable architecture for data ingestion and the on-demand generation of contextualized KGs from the ingested data. We further provide a proof-of-concept implementation using open-source technologies, demonstrated on a real-world use case of decision support in ATM. A comparison against a traditional monolithic pipeline shows that the proposed architecture achieves superior ingestion rates, with horizontal scaling further increasing the throughput.**

***Index Terms*—Knowledge graph management system, data lakehouse, big data, microservice architecture, contextualized knowledge graphs**

I. Introduction

A knowledge graph (KG) represents a wide variety of facts about real-world entities, incorporating both instance data and ontological knowledge [1]. Organizations increasingly build *enterprise KGs* from diverse data sources, including relational databases and unstructured data, to gain a comprehensive view of their operations [2]. Their flexibility and adaptability to various tasks make KGs a potential unified foundation for business analytics and decision support [1], [3]. In air traffic management (ATM), where air traffic controllers and pilots rely on various sources to gain situational awareness of the air traffic network, (contextualized) KGs have been proposed as an integrated representation of the information required for decision support [4], [5], [6], [7].

Research on KG data architectures suitable for business analytics and decision support remains limited. Existing KG architectures and implementations focus primarily on data stores and query engines (cf. [8]). However, these platforms do not adequately address the *Three Vs* of big data—namely, volume, variety, and velocity [9]—which are critical for supporting applications in business analytics and decision support. Platforms such as Wikibase [10] and data stores such as GraphDB [11] and RDFox [12], [13] facilitate efficient management of monolithic, possibly contextualized KGs, but will nevertheless find a challenge in real-world settings where large volumes of a variety of data arrive at high velocity and have to be integrated into the KG, e.g., in the ATM domain. Another proposed implementation of a KG system explicitly geared towards scalability for the purposes of analytics [14] primarily emphasizes querying while neglecting comprehensive architectural solutions for data ingestion. Those approaches could be used in combination with the data lakehouse architecture proposed in this paper, which allows for rapid ingestion and on-demand KG construction for different applications.

The underlying assumption of the proposed approach is that many applications do not require access to the entire KG at once. For example, in ATM, the pilot briefing for a flight from Munich to Vienna does not require information on the Spanish airspace. A contextualized KG (CKG), which partitions knowledge facts into multiple contexts [15], facilitates the selection of relevant knowledge facts. The proposed architecture allows for ingesting heterogeneous raw data, producing contextualized RDF graphs structured by time, location, and other (topic) dimensions, enabling semantic reasoning and analytics.

Modern data architectures for business analytics and decision support include *data warehouse*, *data lake*, and *data lakehouse* [16]. Data warehouses store structured, consistent data optimized for recurring and routine analyses, typically summarized in reports or dashboards. Data lakes archive raw, unprocessed data for more flexible, exploratory analyses and future use. However, the *schema-on-read* nature of the data lake requires complex cleaning and transformation, making access inefficient for large datasets. Data lakehouses [17] combine the flexibility of data lakes with the structured

nature of data warehouses by leveraging metadata, indexes, and caching to support complex analyses without sacrificing storage efficiency and query flexibility.

In this paper, we propose the *KG Lakehouse*, a cloud-native data lakehouse architecture designed to ingest large volumes of heterogeneous, high-velocity data streams. While prior work such as [18], [19] have applied big data platforms to KG processing, the novelty of the proposed architecture lies in integrating KG-OLAP [20] into a cloud-native lakehouse architecture for contextualized, on-demand KG construction tailored to specific tasks. A key feature of the architecture is the on-demand construction of smaller CKGs that are tailored to support specific tasks in business analytics and decision support, e.g., in ATM. In this architecture, each ingested file is indexed and linked to a context derived from the file's content. This indexing mechanism facilitates efficient querying, filtering, and KG construction by maintaining semantic metadata about the stored raw data. The constructed CKGs are *KG-OLAP cubes* [20], which organize KGs into contexts. The contexts are hierarchically organized along multiple context dimensions, which can be used to aggregate the contained knowledge by merging different contexts. While our implementation builds on KG-OLAP cubes, the proposed architecture is modular and extensible, allowing for the integration of alternative CKG formalisms.

Figure 1, in the upper part, provides a conceptual overview of the proposed *KG Lakehouse*, highlighting the main steps of the processing pipeline. The KG Lakehouse processes incoming high-velocity data streams, which may include a variety of structured, semi-structured, or unstructured files. The incoming files are ingested, the content of the ingested files analyzed to extract relevant semantic metadata about the context of the comprised information, e.g., time, location, topic, and importance, before the ingested raw files are stored and indexed in the data lakehouse. Rather than maintaining a single, huge, monolithic CKG, with all its scalability issues [20], where most of the facts are anyway not needed by an application to perform a specific task, smaller CKGs containing only the contexts and facts relevant for a specific application and task, e.g., for reporting, data mining, or reasoning, are constructed on demand from the indexed raw files and supplied to the corresponding application.

The proposed architecture can be employed in the ATM domain for aeronautical information management. In particular, we consider information filtering and summarization for pilot briefings [21], [22], trajectory prediction [23], and adaptive ATM software that provides air traffic controllers with the information required to perform their tasks [6], [7]. Figure 1, in the lower part, shows an example instantiation of the KG Lakehouse for the ATM domain. This KG Lakehouse receives continuous streams of messages about infrastructure, e.g., a temporary update of a navigation aid's usage type or frequency, updates of flight plans, weather reports, flight trajectories, eye-tracking of controllers to determine controllers' attention on specific tasks [24], and voice data, possibly as textual transcripts. The content of the incoming files is analyzed upon ingestion, to extract information regarding the semantics of the content, matching the files, for example, to certain time spans, locations (airspace segments, flight information regions, or airports), importance/criticality levels, or topics. A pilot briefing for a flight from Munich to Vienna on 14 May 2025 at 14:00, or an air traffic controller directing that particular flight, would require a different CKG than a flight on a different date or a flight connecting different cities. Furthermore, a pilot briefing requires a completely different CKG in terms of the topics represented compared to trajectory prediction, conflict detection, or adaptive ATM software as proposed by the AISA and AWARE projects [6], [7] that considers the controllers' attention to display precisely the information that the controller currently requires to complete a task. Such an ATM software will frequently request CKGs with the required subset of knowledge facts valid at different times.

The work presented in this paper is part of AWARE [7], an exploratory research project in the Single European Sky ATM Research (SESAR) joint undertaking within the EU Horizon Europe program. The AWARE project investigates the extension of ATM software with artificial intelligence (AI) features that are aware of a controller's focus and information requirements, the controller's current tasks, and the current air traffic situation. A KG serves as an integrated data repository, which comprises information about infrastructure, flight plans and flight trajectories, events, etc. On the one hand, declarative rules formulated using SPARQL [25] or Datalog [26] allow for deterministic derivation of new facts from the existing KG. On the other hand, probabilistic machine-learning models may serve for prediction of links between different entities or properties of individual entities in the KG, e.g., for conflict detection.

We provide an online appendix [27] with detailed documentation of the KG Lakehouse system. A companion repository [28] documents the experimental setup, infrastructure, datasets, benchmarking procedures, and performance results, along with evaluation logs and Jupyter notebooks for reproduction of experiments and analyses. The AIXM-based dataset generator [29] used to create the datasets used in the experiments is openly available. Finally, the source code of the prototype implementation [30], including deployment scripts and service-level configurations, is also provided.

The remainder of this paper is organized as follows. Section II introduces the proposed architecture. Section III presents an experimental evaluation of the scalability of the proposed architecture. Section IV reviews related work. Section V concludes the paper with a summary and an outlook on future work.

II. Architecture

A typical KG management system (KGMS) is monolithic, designed to build and manage a single centralized KG. Although KGMS implementations can leverage cloud infrastructure and distributed computing for storage and processing, the KG itself grows as a single entity, complicating querying and management as scale and complexity increase. In dynamic

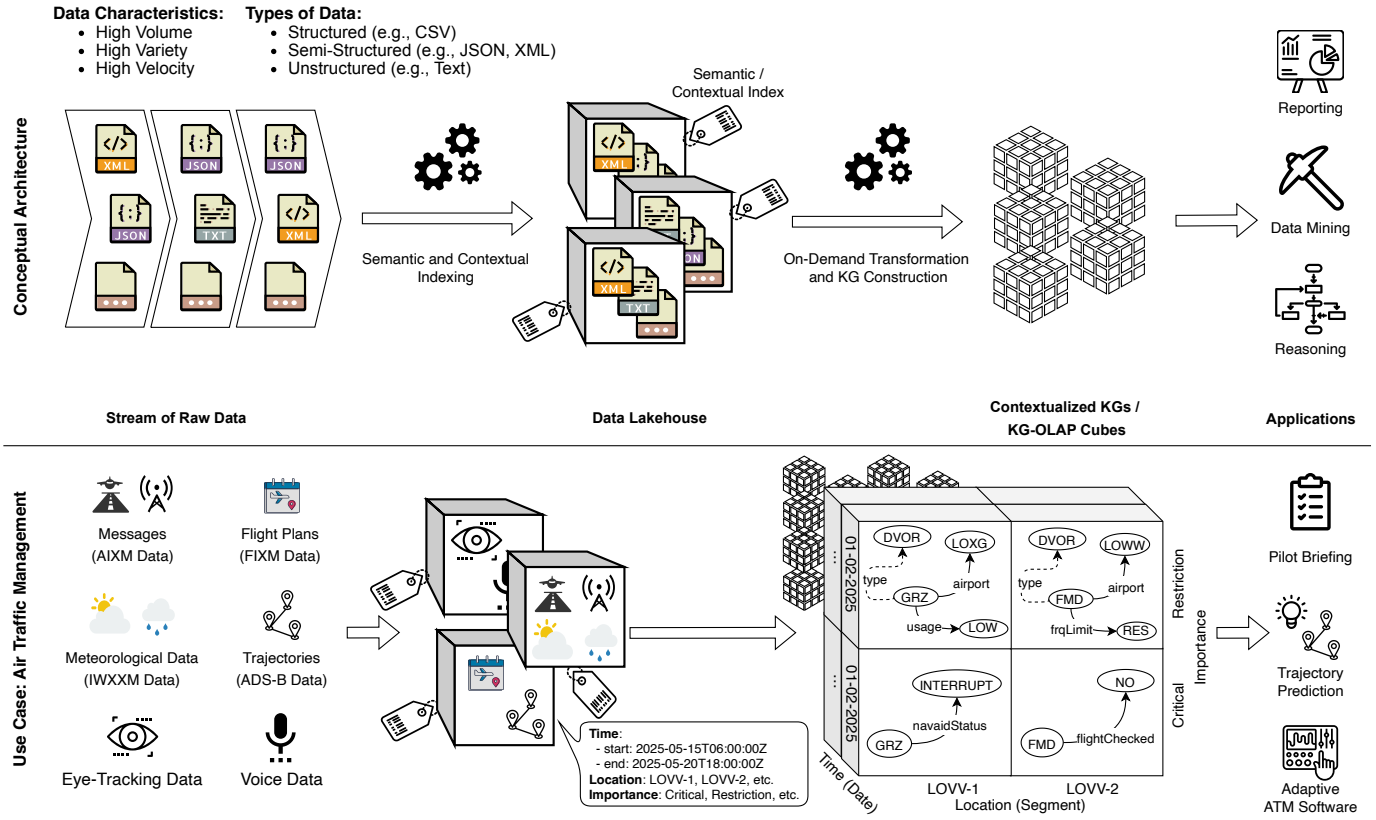


Figure 1. Overview of the proposed cloud-native data lakehouse architecture and an instantiation for ATM

and data-rich domains such as ATM, where continuous data streams are generated, analytical tasks often focus on specific subsets of the KG, e.g., specific geographic regions, time periods, or individual topics. For individual tasks, e.g., a pilot briefing for a particular flight or an air traffic controller's directing of an individual flight, access to the monolithic KG for the entire airspace is not required. A modular approach with the generation of context-specific KGs then improves performance and scalability. Figure 2 shows the main components and processing steps for the proposed *KG Lakehouse* architecture. The design reflects real-world constraints such as data velocity and heterogeneity, guiding key architectural trade-offs. The architecture reflects key lakehouse principles, including separation of storage and compute (via MinIO and microservices), schema-on-read (during KG construction), and metadata-driven indexing (using context hierarchies in Cassandra); the online supplementary material [27] provides further information.

The *Surface Service* acts as the unified entry point, providing a RESTful API to handle interactions from users, applications, and data streams. In general, we distinguish between *data ingestion* and on-demand *KG construction*. The modular architecture with a separation of ingestion tasks and graph construction (query) tasks allows independent scaling of each component to meet fluctuating data volumes and processing demands. Thus, resource utilization is optimized by scaling only the necessary services, which allows efficient on-demand generation of contextualized KGs that overcome the limitations of a traditional monolithic KGMS. In the following, we describe the main components and processing steps for data ingestion and KG construction.

A. Data Ingestion

The KG Lakehouse employs an event-driven pipeline for scalable, resilient, and flexible data ingestion in cloud-native systems [31]. The *Ingestion Scheduler* and *Ingestion Service* manage the asynchronous data ingestion. The Ingestion Scheduler creates ingestion tasks, annotated with metadata. The metadata for a task consist of a description of the data source and content type, e.g., Aeronautical Information Exchange Model (AIXM) data, which are then used to select the appropriate domain-specific *Content Analyzer*. The raw files are persisted in the *Storage Service*, the ingestion tasks are pushed to a shared queue.

Figure 3 shows the high-level ingestion process conducted by the *Ingestion Service*. The Ingestion Service pulls tasks from the *Queue Service*, retrieves the data (raw file) associated with an ingestion task from the *Storage Service*, and analyzes the data using the appropriate content analyzers. The appropriate *content analyzer* is determined based on the metadata extracted during task creation. Domain-specific content analyzers designed to extract structured information from heterogeneous sources can be provided in a *plug-and-*

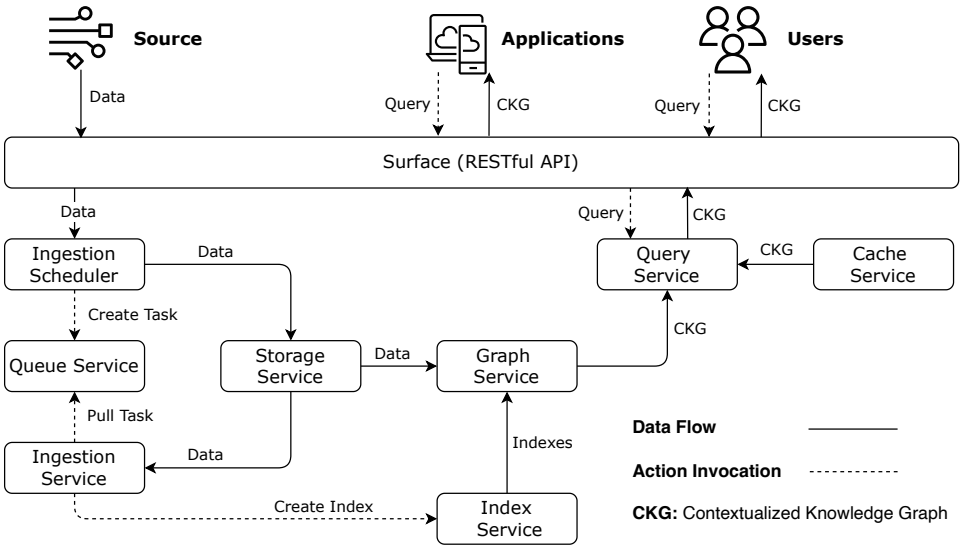


Figure 2. The proposed cloud-native data lakehouse architecture

play fashion. In this paper, we demonstrate the approach with a content analyzer for AIXM data, which represent messages about events in the air traffic network.

The content analyzers serve to annotate the ingested files with domain-specific contextual and semantic metadata. These metdata describe the *context* of validity for the information contained in the files. The context can be determined temporal intervals that the information is valid for, geographic locations that the information concerns, and the described topics, e.g., navigation aid restrictions or airspace closure. Thus, the extracted context from a file is *multidimensional*. The context dimensions, in turn, are *hierarchical*. The contexts of a file are *upserted*, i.e., inserted or updated, into the *Index Store* to maintain accurate contextual indexes that can later be queried to retrieve files for KG construction.

Figure 4 illustrates the process of content analysis and context extraction using an example from the ATM domain. In this example, an `AIXMBasicMessage` is processed. This message follows the AIXM standard [32] for exchanging aeronautical information. The index service extracts information describing the *time* of validity (1 January 2018), the *location* with which the information is concerned (Vienna Airport or LOWW) and the *topic* (*AirportHeliport*). This information is hashed and stored, linking the ingested file to the derived context for efficient retrieval in the process of KG construction.

B. Knowledge Graph Construction

The KG construction process retrieves relevant files for specific contexts on demand, based on a user-defined KG specification ("query"), and generates a contextualized KG (CKG) in form of a KG-OLAP cube. Figure 5 illustrates the KG construction process, detailing the interactions between *Query Service* and *Graph Service* in the construction of task-specific CKGs.

The KG construction process starts with the submission (via the Surface) of a KG specification to the *Query Service* by a user or application. The KG specification defines the temporal and spatial applicability as well as other dimensional attributes, e.g., the topic, of the desired CKG, depending on the configuration of the system and the target domain. For example, a request for *AirportHeliport* data for Austrian airspace during January 2018 would constrain the time to 2018 at the year level and January at the month level, the location to Austria, including relevant flight information regions (FIRs) and segments, and the topic to the *AirportHeliport* category.

The Query Service validates and parses a KG specification to extract context hierarchies (*SliceDiceContext*) and dimension mappings (*MergeLevels*). Specific contexts are directly derived from the specification, while general contexts are determined based on overlapping or shared dimensions across existing contexts. The service then iterates through all relevant combinations to identify the required contexts. The *Query Service* prepares queries for the required contexts and retrieves the file names for the contexts from the *Index Store*, ensuring both specific and general contexts are included. The *Graph Service* constructs the KG by retrieving raw data files from the *Storage Service*, extracting relevant triples of subject, predicate, and object for each context, and merging the triples into the contexts based on the dimension mappings to create a CKG. The final step is the aggregation of the results according to the KG specification, with caching mechanisms optimizing performance by retrieving pre-computed portions of the CKG, if available. This design instantiates the formal KG-OLAP model [20], where context dimensions form well-defined hierarchies that guarantee the safety of OLAP operations such as slicing, dicing, and merging.

III. Evaluation

We conducted two series of experiments to evaluate the scalability of our proposed architecture. The first series of experiments evaluates data ingestion using a predefined load,

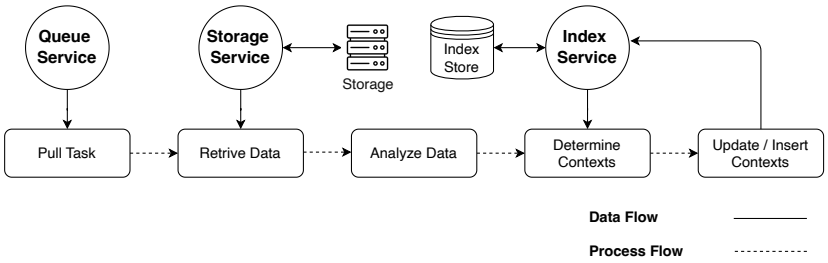


Figure 3. Overview of the data ingestion process

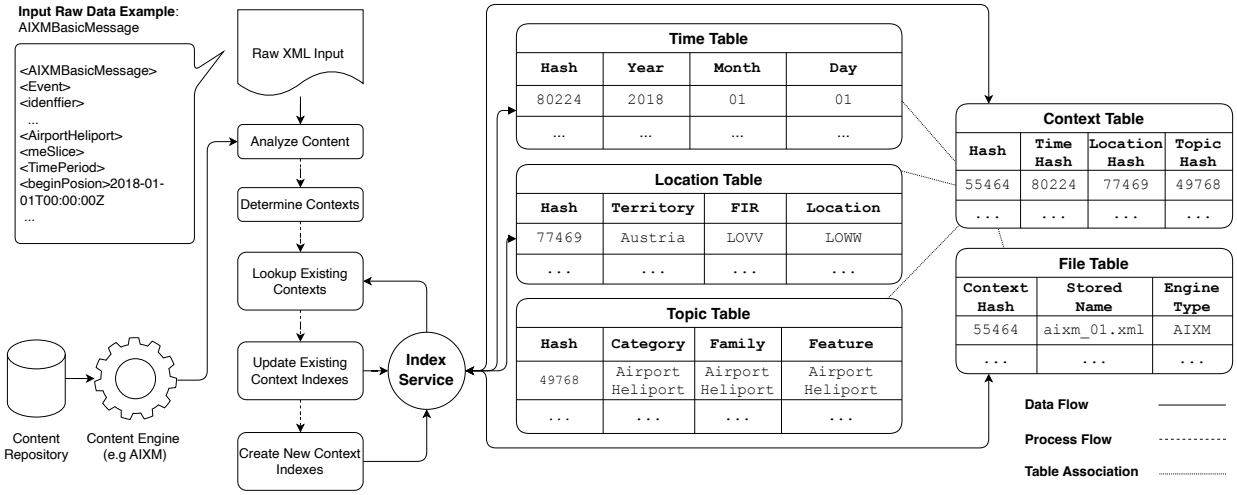


Figure 4. Overview and example of the data indexing process

denoted as **L0** (Table I), which is repeatedly sent to the system. We vary the scale of data ingestion components deployed on the Kubernetes cluster to examine how increased scaling impacts ingestion speed. The results demonstrate that increasing the scale of ingestion components considerably improves data ingestion speed. The second series of experiments evaluates the system's ability to construct KGs of varying sizes on demand from different file loads stored and indexed by the system. The KGs (Table II) are constructed from variously-sized datasets, from **L1** to **L5**, with multiple repeats of the KG construction process to observe how dataset size influences response time. For these experiments, the KG Lakehouse was deployed on a Kubernetes cluster [33] with eight nodes: two server nodes and six worker nodes. Each server node was configured with four virtual CPUs, 8 GB RAM, and 48 GB disk space. Each worker node was configured with eight virtual CPUs, 16 GB RAM, and 100 GB disk space. All nodes ran Ubuntu-22.04 Server (minimal) and k3s-1.24.6.

A. Datasets

We conduct experiments with realistic synthetic datasets from the ATM domain. Standard benchmark datasets lack the contextual and hierarchical structure required by KG-OLAP. Therefore, we implemented an AIXM generator to produce realistic, reproducible XML test data [29] according to the AIXM standard [32]. Instructions for reproducing the datasets are provided in the online appendix [28]. Each file represents a message (e.g., AIXM `BasicMessage`) and typically contains 20–100 RDF-equivalent triples. Details and examples are provided in the online documentation [27]. The files are organized along three context dimensions: *Location*, *Topic*, and *Time*. The *Location* dimension is hierarchically organized into territories, flight information regions (FIRs), and locations, where locations roll up to FIRs, and FIRs roll up to territories. The *Topic* dimension is hierarchically organized into topic categories, topic families, and topic features, where features roll up to families, and families roll up to categories. Finally, the *Time* dimension is hierarchically organized into year, month, and day. Days roll up to months, and months roll up to years. We considered three territories: Austria, Germany, and France. The Austrian territory has one FIR with three locations. The German territory have two FIRs, each with four locations. French territory has two FIRs, each with three locations. From the AIXM model, we considered five categories: **AirportHeliPort**, **Routes**, **Airspace**, **NavaidsPoints**, and **Procedure**. **AirportHeliPort** has two families, each with four features. **Routes** has two families, each with two features. **Airspace**, **NavaidsPoints**, and **Procedure** each have one family with one feature. We generated five datasets according to this structure. Table I lists the number of files, quads, and contexts for each dataset.

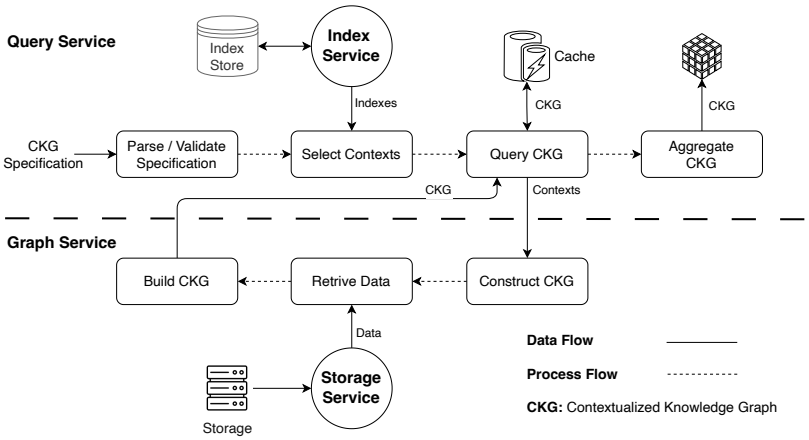


Figure 5. Overview of the KG construction process conducted by the Query Service and the Graph Service

TABLE I
CHARACTERISTICS OF THE DATASETS USED IN EXPERIMENTS

Load	Files	Quads	Contexts
L0	0.16M	3.18M	0.08M
L1	0.82M	15.92M	0.41M
L2	1.65M	31.84M	0.82M
L3	2.47M	47.76M	1.24M
L4	3.79M	73.24M	1.89M
L5	4.61M	89.16M	2.31M

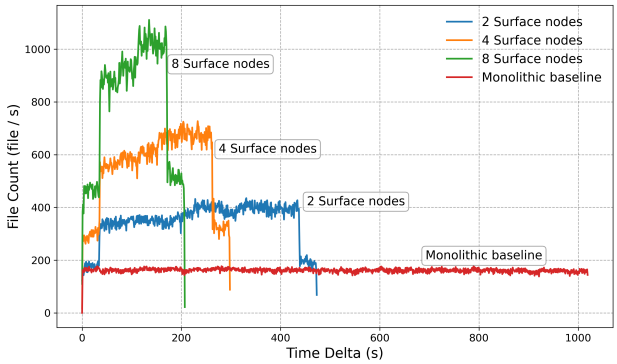


Figure 6. Ingestion rate with different numbers of surface nodes

B. Baseline

To provide context for the evaluation, we conducted baseline experiments using a monolithic pipeline to ingest and construct KGs similar to the ones obtained from the KG Lakehouse. The pipeline, implemented in *Python*, for processing AIXM XML files, using the same dataset used in our prototype evaluation, and storing the resulting graph in Apache Fuseki [34], created a contextualized knowledge graph similar to the KG Lakehouse architecture. Fuseki was selected as it supports RDF and SPARQL; however, like other existing systems, it assumes a precomputed KG, limiting scalability. Our approach is orthogonal: it builds contextualized KGs on demand from raw data, avoiding these limits while remaining compatible with such systems. The pipeline was deployed on a single machine with eight virtual CPUs, 32 GB RAM, and a 100 GB SSD, running on Ubuntu-22.04 Server. While functional, the monolithic setup demonstrated significant limitations in scalability and performance. For example, processing **L0**, the smallest dataset, took the monolithic setup considerably longer than the KG Lakehouse, despite leveraging multiprocessing. Resource monitoring revealed underutilized resources and bottlenecks in the monolithic graph construction and storage design, further emphasizing the scalability advantages of the distributed KG Lakehouse architecture.

C. Data Ingestion

The purpose of the experiment is to demonstrate the prototype's ability to quickly ingest large volumes of data, and assess CPU and memory usage. We used a client to send as many ingestion requests as possible to the framework with dataset **L0** (refer to Table I). The server setup included a Cassandra cluster with three nodes, an ElasticMQ cluster with one node, one ingestion scheduler, and two ingestion services. Three tests were performed with the same dataset, doubling the service instances each time (two, four, and eight, respectively). Each test was repeated three times for consistency. Figure 6 shows data ingestion throughput in files per second. In each run, we scaled only the *Surface Service* instances, keeping the other configurations constant. The results indicate that as we increased the number of service instances, the ingestion rate increased from 10 MB/s with two nodes to 35 MB/s with eight nodes. The time to ingest the entire dataset decreased accordingly. These results demonstrate that increasing service instances enables the system to handle more ingestion requests simultaneously. In addition, the resource footprint was minimal, optimizing resource usage.

Figure 6 also shows file ingestion rates for the monolithic pipeline. The monolithic pipeline peaks at 70 files per second, compared to 400 files per second for the *lakehouse architecture* at the lowest scaling configuration. In addition, ingesting and processing the entire dataset takes approximately 2 500

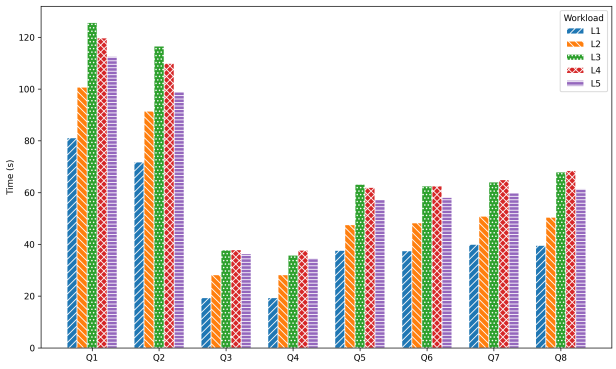


Figure 7. Run times of on-demand construction of different CKGs, specified by Q1–Q8, with various workloads (L1–L5), averaged over three runs

TABLE II
RESULT SIZE OF THE EVALUATION OF THE KG SPECIFICATIONS ("QUERIES") IN TERMS OF RETURNED CONTEXTS AND QUADS

KG	Contexts	Quads
Q1	6 975	269 700
Q2	225	197 025
Q3	2 790	107 880
Q4	1	50 220
Q5	4 392	177 144
Q6	1	83 448
Q7	4 392	166 164
Q8	36	128 304

seconds with the monolithic approach versus just under 600 seconds with the *lakehouse architecture*. To ingest and construct the same knowledge graph, the *lakehouse architecture* requires roughly 650 seconds in total, while the monolithic pipeline, despite pre-constructing the knowledge graph, still requires at least 2 500 seconds, excluding retrieval time.

D. Knowledge Graph Construction

We conducted experiments to demonstrate the prototype's ability to construct KGs on demand in reasonable time frames. We defined KGs of various sizes to represent different slice-and-dice operations and tested them on five datasets of varying sizes (see Table I). Table II outlines the characteristics of each KG. The online appendix [28] provides more details on KGs design. Dataset **L1** is small, resulting in short run times. Larger datasets **L2** and **L3** showed increased run times, while further expansion to **L4** stabilized run times.

For indexing, we use Apache Cassandra [35], a distributed NoSQL database designed for managing large-scale datasets across distributed environments. Cassandra's architecture natively supports horizontal scalability and fault tolerance. In our experiments, we observed that as the index size grows, query response times remain stable and in some cases even improve due to optimized partitioning and caching mechanism. Figure 7 compares the response times for different KG constructions against datasets **L1**, **L2**, **L3**, **L4**, and **L5** (see Table I for more details).

IV. RELATED WORK

Existing KGMS provide the infrastructure for integrating, storing, and scaling KGs, enabling their application across diverse domains. Systems like Neo4j [36] and Blazegraph [37] specialize in handling graph data using property graphs and RDF models, respectively. These systems enable complex graph analytics and semantic reasoning but are often constrained by static data representations. Distributed frameworks such as Apache Spark GraphX [38] and Amazon Neptune [39] address scalability through partitioning, distributed querying, and hybrid reasoning methods [1]. Despite these advancements, these systems lack the flexibility to dynamically generate KGs based on user-defined queries in real-time, a key feature of the proposed KG Lakehouse architecture.

Recent advances in KG storage and analytics have introduced innovative frameworks that address scalability, dynamic updates, and the integration of diverse data types. Li et al. [19] introduced GraphAr, a storage scheme optimized for managing Labeled Property Graphs (LPGs) within data lakes. While GraphAr focuses on efficient graph data storage, the KG Lakehouse architecture emphasizes dynamic KG generation, extending beyond static storage to support application-specific requirements. Similarly, Stream2Graph [40] constructs dynamic KGs from streaming data, enabling real-time updates and online learning. However, while Stream2Graph excels in provessing evolving data streams, the KG Lakehouse architecture incorporates advanced indexing and contextualization, ensuring efficient querying and retrieval for dynamically composed KGs for analytical tasks.

Chen et al. [41] propose a system for constructing KGs from unstructured text using deep active learning, thereby reducing the dependence on labeled data. This method could complement the KG Lakehouse architecture by providing advanced data extraction and learning capabilities to the ingestion pipeline for improved automation and adaptability.

Several frameworks specifically address large-scale KG processing and analytics. P-OLAP [42] extends OLAP principles to graph analytics, leveraging Apache Hadoop and MapReduce [43] for scalability. SANSA [18] offers a unified semantic stack for processing RDF data at scale using Apache Spark, supporting SPARQL, reasoning, and machine learning. While SANSA is optimized for batch processing of static RDF datasets, our architecture focuses on real-time ingestion, contextual indexing, and on-demand construction of task-specific CKGs using KG-OLAP. These approaches are complementary: Our system could integrate with SANSA for downstream KG processing, and we plan to explore this integration in future work. Trinity [44], by contrast, efficiently handles online and offline queries using distributed memory storage. While both frameworks are powerful, they target specific formats or rely on static KG structures.

V. SUMMARY AND FUTURE WORK

The increasing use of knowledge graphs (KGs) as a data source, coupled with the high volume, velocity, and variety of data required for their construction, highlights the limitations

of monolithic architectures in supporting efficient analytical applications. In this paper, we proposed a cloud-native data lakehouse architecture for KG management, designed to address these challenges by enabling efficient data ingestion, indexing, and on-demand generation of contextualized KGs tailored to specific tasks.

We presented a proof-of-concept prototype using open-source frameworks and evaluated its performance using realistic synthetic data from the ATM domain. The results demonstrated that the architecture efficiently handles large-scale data ingestion and adapts dynamically to different load conditions. Ingestion rates increased near-linearly with the number of service instances, reaching up to 400 files per second with eight nodes (Fig. 6), clearly outperforming the 70 files per second achieved by a monolithic pipeline. KG construction response times remained stable and reasonable, even as dataset sizes increased, demonstrating the system's scalability and reliability.

The evaluation was a laboratory exercise and was not conducted in a representative environment. Due to the exploratory nature of the AWARE project, validation exercises with large-scale data from various real-world sources to support air traffic control operations were out of scope of the project. Thus, the investigation of key parameters and safety implications as well as alignment with relevant EU and ICAO regulations could be part of a future exploratory or industrial research project.

This paper focused on efficient data ingestion and the on-demand generation of contextualized KGs for individual tasks. Future work will extend the presented approach with scalable graph operations over KG-OLAP cubes—such as abstraction, pivoting, and reification [20]—to enable the transformation and aggregation of subgraphs within individual cube cells. These operations will support advanced analytics tasks, including link prediction, KG completion, and context-aware summarization.

Acknowledgment

This work was conducted as part of the AWARE project, which has received funding from the SESAR Joint Undertaking under grant agreement no. 101167442 under the European Union's Horizon Europe research and innovation program. The views expressed in this work are those of the author.

References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *ACM Computing Surveys*, vol. 54, no. 4, 2021.

[2] J. Sequeda and O. Lassila, *Designing and building enterprise knowledge graphs*. Springer, 2022.

[3] M.-E. Papadaki, Y. Tzitzikas, and M. Mountantonakis, "A brief survey of methods for analytics over rdf knowledge graphs," *Analytics*, vol. 2, no. 1, pp. 55–74, 2023.

[4] R. M. Keller, "Building a knowledge graph for the air traffic management community," in *Companion Proceedings of the 2019 World Wide Web Conference*, 2019, pp. 700–704.

[5] C. G. Schuetz, B. Neumayr, M. Schrefl, E. Gringinger, A. Vennesland, and S. Wilson, "The case for contextualized knowledge graphs in air traffic management," in *Joint Proceedings of the International Workshops on Contextualized Knowledge Graphs, and Semantic Statistics co-located with 17th International Semantic Web Conference (ISWC 2018)*, ser. CEUR Workshop Proceedings, S. Capadisli, F. Cotton, J. M. Giménez-García, A. Haller, E. Kalampokis, V. Nguyen, A. P. Sheth, and R. Troncy, Eds., vol. 2317. CEUR-WS.org, 2018. [Online]. Available: https://ceur-ws.org/Vol-2317/article-10.pdf

[6] "AISA: AI Situational Awareness Foundation for Advancing Automation." [Online]. Available: https://doi.org/10.3030/892618

[7] "AWARE: Achieving Human–Machine Collaboration with Artificial Situational Awareness." [Online]. Available: https://doi.org/10.3030/101167442

[8] W. Ali, M. Saleem, B. Yao, A. Hogan, and A.-C. N. Ngomo, "A survey of rdf stores & sparql engines for querying knowledge graphs," *The VLDB Journal*, vol. 31, no. 3, pp. 603–628, 2021.

[9] D. Laney, "3D data management: Controlling data volume, velocity and variety," *META Group Research Note*, vol. 6, 2001.

[10] D. Diefenbach, M. D. Wilde, and S. Alipio, "Wikibase as an infrastructure for knowledge graphs: The eu knowledge graph," in *ISWC 2021*, A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, and H. Alani, Eds. Springer, 2021, pp. 631–647.

[11] Ontotext, "GraphDB," accessed: 2025-05-13. [Online]. Available: https://graphdb.ontotext.com/

[12] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee, "RDFox: A highly-scalable RDF store," in *ISWC 2015 – Part II*, ser. LNCS, M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, Eds., vol. 9367. Springer, 2015, pp. 3–20.

[13] Oxford Semantic Technologies, "RDFox," accessed: 2025-05-13. [Online]. Available: https://www.oxfordsemantic.tech/rdfox

[14] S. Purohit, N. Van, and G. Chin, "Semantic property graph for scalable knowledge graph analytics," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 2672–2677.

[15] L. Serafini and M. Homola, "Contextualized knowledge repositories for the semantic web," *Journal of Web Semantics*, vol. 12-13, pp. 64–87, 2012, reasoning with context in the Semantic Web.

[16] A. A. Harby and F. Zulkernine, "From data warehouse to lakehouse: A comparative review," in *2022 IEEE International Conference on Big Data*, 2022, pp. 389–395.

[17] M. Armbrust, M. Zaharia, A. Ghodsi, and R. Xin, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *11th Conference on Innovative Data Systems Research, CIDR 2021*, 2021. [Online]. Available: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

[18] V. Janev, D. Graux, H. Jabeen, and E. Sallinger, *Knowledge Graphs and Big Data Processing*, 01 2020.

[19] X. Li, W. Zeng, Z. Wang, D. Zhu, J. Xu, W. Yu, and J. Zhou, "Graphar: An efficient storage scheme for graph data in data lakes," 2024. [Online]. Available: https://arxiv.org/abs/2312.09577

[20] C. G. Schuetz, L. Bozzato, B. Neumayr, M. Schrefl, and L. Serafini, "Knowledge Graph OLAP: A multidimensional model and query operations for contextualized knowledge graphs," *Semantic Web*, vol. 12, no. 4, pp. 649–683, 2021.

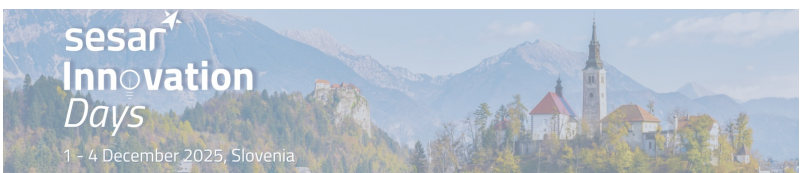
[21] D. Steiner, I. Kovacic, F. Burgstaller, M. Schrefl, T. Friesacher, and E. Gringinger, "Semantic enrichment of dnotams to reduce information overload in pilot briefings," in *2016 Integrated Communications Navigation and Surveillance (ICNS)*, 2016, pp. 6B2–1–6B2–13.

[22] C. G. Schuetz, B. Neumayr, M. Schrefl, E. Gringinger, and S. Wilson, "Semantics-based summarisation of atm information: Managing information overload in pilot briefings using semantic data containers," *The Aeronautical Journal*, vol. 123, no. 1268, p. 1639–1665, 2019.

[23] H. Georgiou, N. Pelekis, S. Sideridis, D. Scarlatti, and Y. Theodoridis, "Semantic-aware aircraft trajectory prediction using flight plans," *International Journal of Data Science and Analytics*, vol. 9, no. 2, pp. 215–228, 2020.

[24] C. Vetter, K. Samardžić, I. Tukarić, T. Radišić, and R. H. Hermann, "Gaze analysis of air traffic contoller using AI-based conflict detection," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024.

[25] O. Hartig, A. Seaborne, R. Taelman, G. Williams, and T. Pellissier Tanon, "SPARQL 1.2 Query Language – w3c working



draft 15 november 2025," W3C, Tech. Rep., 2025. [Online]. Available: https://www.w3.org/TR/2025/WD-sparql12-query-20251115/
[26] T. Eiter, G. Gottlob, and H. Mannila, "Disjunctive datalog," *ACM Transactions on Database Systems*, vol. 22, no. 3, pp. 364–418, 1997.
[27] B. Ahmad and C. G. Schuetz, "Online appendix for KG lakehouse experiments," https://doi.org/10.5281/zenodo.17629301, accessed: 2025-05-13.
[28] ——, "Experimental results and benchmarking scripts," https://doi.org/10.5281/zenodo.17629455, accessed: 2025-05-13.
[29] ——, "AIXM data generator," https://doi.org/10.5281/zenodo.17629545, accessed: 2025-05-13.
[30] B. Ahmad, D. Haunschmied, and C. G. Schuetz, "KG lakehouse source code," https://doi.org/10.5281/zenodo.17629660, accessed: 2025-05-13.
[31] A. Singh, V. Singh, A. Aggarwal, and S. Aggarwal, "Event driven architecture for message streaming data driven microservices systems residing in distributed version control system," in *2022 International Conference on Innovations in Science and Technology for Sustainable Development (ICISTSD)*, 2022, pp. 308–312.
[32] AIXM, "Aeronautical information exchange model," https://www.aixm.aero, 2019.
[33] Kubernetes, "Kubernetes," https://kubernetes.io, accessed: 2023-02-01.
[34] Apache Jena, "Fuseki," accessed: 2024-12-16. [Online]. Available: https://jena.apache.org/documentation/fuseki2/
[35] Apache Cassandra, "Cassandra," https://cassandra.apache.org, accessed: 2023-02-01.
[36] Neo4j, "Neo4j," accessed: 2025-05-13. [Online]. Available: https://neo4j.com/
[37] Blazegraph, "Blazegraph Database," accessed: 2025-05-13. [Online]. Available: https://blazegraph.com/
[38] Apache Spark, "GraphX," accessed: 2025-05-13. [Online]. Available: https://spark.apache.org/graphx/
[39] Amazon, "Neptune," accessed: 2025-05-13. [Online]. Available: https://aws.amazon.com/neptune/
[40] M. Barry, A. Bifet, R. Chiky, S. El Jaouhari, J. Montiel, A. El Ouafi, and E. Guerizec, "Stream2graph: Dynamic knowledge graph for online learning applied in large-scale network," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 2190–2197.
[41] A. Pradhan, K. K. Todi, A. Selvarasu, and A. Sanyal, "Knowledge graph generation with deep active learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.
[42] A. Beheshti, B. Benatallah, and H. R. Motahari Nezhad, "Scalable graph-based olap analytics over process execution data," *Distributed and Parallel Databases*, vol. 34, 2015.
[43] Apache Hadop, "Hadoop," https://hadoop.apache.org/, accessed: 2025-12-01.
[44] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," 06 2013, pp. 505–516.