# LALO — A Virtual Data Lake Zone for Composing Tailor-Made Data Products on Demand

Christoph Stach, Yunxuan Li, Laura Schuiki, and Bernhard Mitschang

Institute for Parallel and Distributed Systems, University of Stuttgart, 70569 Stuttgart, Germany
{firstname.lastname}@ipvs.uni-stuttgart.de

**Abstract.** The emerging paradigm of data products, which has become increasingly popular recently due to the rise of data meshes and data marketplaces, also poses unprecedented challenges for data management. Current data architectures, namely data warehouses and data lakes, are not able to meet these challenges adequately. In particular, these architectures are not designed for a just-in-time provision of highly customized data products tailored perfectly to the needs of customers. In this paper, we therefore present a virtual data lake zone for composing tailor-made data products on demand, called LALO. LALO uses data streaming technologies to enable just-in-time composing of data products without allocating storage space in the data architecture permanently. In order to enable customers to tailor data products to their needs, LALO uses a novel mechanism that enables live adaptation of data streams. Evaluation results show that the overhead for such an adaptation is negligible. Therefore, LALO represents an efficient solution for the appropriate handling of data products, both in terms of storage space and runtime.

**Keywords:** Data Product · Virtual Data Lake Zone · Data Stream Adaptation

## 1 Introduction

Novel data management concepts such as *data meshes* [4] and *data marketplaces* [7] envision data not just as valuable raw materials but as fully-fledged company assets. If refined appropriately, data can be transformed into *data products* [22]. A data product is defined as a self-contained, discoverable, and reusable dataset. To this end, it encompasses not only the raw data, but also the metadata, code, and infrastructure required to process and access the included information. The most important characteristic, however, is that it has a business value [20]. Therefore, the market principles that apply to physical products also apply to these data products [19]. With the Fourth Industrial Revolution, these market principles were supplemented by just-in-time production of goods in order to save storage costs, as well as virtually unlimited customization of products so that they can be tailored perfectly to the needs of customers [26].

Common data architectures, namely *data warehouses* [16] and *data lakes* [15] as well as *lakehouses* [2], which are a combination of the former two, however, are incapable of meeting these requirements adequately: Data warehouses lack the required flexibility to enable tailor-made data products, as they are geared towards a few use cases that are clearly defined in advance. The data transformations required for these use cases are applied directly to the raw data and only the resulting data are stored in the data

In: Strauss C., Amagasa T., Manco G., Kotsis G., Tjoa A M., and Khalil I. (Eds.) 35th International Conference, DEXA 2024, Naples, Italy, August 26–28, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol 14911. Springer, Cham, pp. 288–305, 2024.  10.1007/978-3-031-68312-1_22.

warehouse. As data are therefore only available in this aggregated form, it is not possible to define additional use cases at a later date and reprocess the underlying raw data to meet the requirements of these new use cases. Data lakes overcome this by applying the required data transformations only when data are accessed. Besides unprocessed raw data, data are stored redundantly in different pre-processing stages, which leads to high storage costs. This is made even worse by the fact that, unlike physical products, data products are not "consumed" (i.e., depleted) when being used. As a result, the amount of data to be managed increases significantly with each additional data product, especially as a lot of metadata has to be maintained besides the actual data. Enabling fully customized data products would therefore escalate storage costs.

In order to address these shortcomings and thus enable the appropriate handling of data products, we introduce a virtual data lake zone called *LALO*[1]. In particular, our approach enables the just-in-time provision of customizable data products in data lakes without incurring significant storage and processing costs. To this end, we make the following two contributions: a) We extend the data lake concept by adding a **virtual data zone**. In this zone, data products are made available in a non-materialized form. Instead, the products are generated live on access using data stream technologies. In this way, there are **no additional storage costs** for data products. b) To handle this efficiently, we introduce a novel mechanism that enables **live adaptation of data streams**. This facilitates the adaptation of the manufacturing process of data products at runtime, e.g., due to changing customer needs. After a negligible reconfiguration time, the new and **fully tailored data product** is then generated and provided to the customer. It has to be noted that LALO is not only suitable for traditional, i.e., centralized data architectures but also for modern distributed approaches such as data meshes.

The remainder of this paper is structured as follows: Initially, in Sect. 2, we present today's prevailing data architectures and discuss their shortcomings with regard to the handling of data products. Subsequently, we look at research approaches that facilitate the management of data products in Sect. 3. In Sect. 4, we introduce LALO, a novel approach that enables the just-in-time provision of tailor-made data products by means of a virtual data lake zone and the live adaptation of data streams. We evaluate in Sect. 5 whether LALO is effective in terms of storage space consumption and efficient in terms of runtime costs. Section 6 concludes this paper and provides a brief outlook.

## 2 Data Architectures

As data are a valuable resource for enterprises, various data architectures have been developed to efficiently support the collection and processing of data for specific forms of use. Data warehouses, data lakes, and lakehouses have proven to be particularly effective and durable in this regard, making these data architectures widely used in practice [11]. From an enterprise perspective, however, the paradigm of data products is becoming increasingly attractive, i.e., offering highly processed data tailored to a value-adding business case to internal and external customers [6]. Yet, data products require agile data management to meet the challenges of just-in-time production and unlimited

[1] The name LALO refers to the composer Édouard Lalo. Just as Lalo composed his music tailored to different countries, LALO enables composing data products tailored to different customers.

customizability. Therefore, we discuss whether the aforementioned data architectures are suitable for this. We also look at how data meshes and data marketplaces, which have popularized the paradigm of data products, address these data management challenges.

*Data Warehouse.* A data warehouse is a central data management solution designed to gather consolidated information from various sources. It is characterized by the fact that data from heterogeneous sources are brought into a homogeneous structured form by means of an ETL process (extract, transform, load). The underlying relational data schema is geared towards the intended analytical tasks. However, as relational database systems are designed for vertical scaling, data are only stored in a highly aggregated form—in accordance with the intended use cases—to reduce storage costs. This comes at the expense of flexibility, as raw data are not available, and therefore it is not possible to make ex-post adaptations for other use cases. Moreover, the ETL process is time-consuming, which is why the data warehouse is not populated with new data in real-time [16].

*Data Lake.* Such rigid data schemas and high storage costs are inadequate when faced with big data. Data lakes therefore use NoSQL technologies to store structured as well as unstructured data in their raw format. A data lake is therefore much more flexible than a data warehouse, as its data corpus can be used for any analytical task. This is enabled by an ELT process (extract, load, transform), i.e., data are ingested in their original data schema, and use-case-dependent adjustments are only applied when the data are accessed. The high level of flexibility is thus achieved at the cost of complex data access. To reduce these costs, data lakes often store variants of the data in different pre-processing stages in addition to the raw data. Yet, this pre-processing is rather generic, i.e., further refinement and customization steps are still required when accessing the data [15].

*Lakehouse.* A lakehouse therefore combines the flexibility and scalability of data lakes with the optimized data structure and efficient query capabilities of data warehouses. To this end, structured and unstructured data are stored in their raw format, while a wide range of metadata increases the effectiveness and efficiency of query processing. These metadata include information about the actual stored data, the underlying data formats and schemas as well as access structures. In addition, transformation information is also stored for the data, which describes how the data have to be pre-processed for specific use cases. Similar to a data warehouse, this enables a structured view on the data however without having to materialize it inevitably. Instead, the specified transformation steps can be applied on the fly when accessing the data [2].

*Data Mesh.* All of these approaches have in common that they are based on centralized data management for an entire enterprise. However, since data understanding as well as insights into value-adding business cases—i.e., knowledge about data products in demand—are concentrated in the respective domains, the data mesh architecture attempts to reflect this fact. Instead of a single central data architecture, the data mesh transfers data responsibility to the individual domains. Data are exchanged only in the form of highly processed data products. Yet, the data mesh is primarily an organizational and not a technical approach. This means in particular that each domain is self-administering, i.e., they have to take care of the generation, quality assurance, and provision of their data products. To this end, data warehouses, data lakes, and lakehouses are still used locally in a data mesh approach for the management of data products [19].

*Data Marketplace.* A data marketplace takes the idea of a data product that is offered to customers one step further. This approach offers a platform that brings data owners and customers together. That is, it represents a central interface for providing, discovering, and accessing data. Yet, it is primarily focused on the management of metadata about data products, as a data catalog is required that describes which products are available, what properties they have, and how they can be used. Data lakes are the predominant solution for managing the actual data products in data marketplaces due to their flexibility [7].

*Synopsis.* Two aspects are particularly relevant for the appropriate handling of data products: On the one hand, just-in-time generation must be possible to reduce storage costs. On the other hand, customers should be able to customize data products. Yet, all five discussed data architectures offer only limited support in this regard. The time-consuming ETL process in data warehouses prevents just-in-time generation and the rigid schemas impede subsequent adaptation to customer requirements. Although the ELT process in data lakes offers more flexibility, data are only transformed when accessed, i.e., adaptation to use cases is entirely handled by the customer. Although it is possible to keep pre-processed versions of the data in addition to the raw data to reduce the workload when accessing it, such a redundant data strategy causes high storage costs. The lakehouse aims to create the best of both worlds by offering the flexibility of a data lake and providing a view on refined data similar to a data warehouse using metadata. The required refinement steps are automatically applied to the data when accessed. However, as the metadata are also geared towards a schema, customizing the data products is still not possible.

**Table 1:** Comparison of Existing Data Architectures in Terms of Usability for Data Products.

| **Data Architecture** | **Just-in-Time Generation** | **Customized Data Products** |
|---|---|---|
| *Data Warehouse* | The time-consuming ETL process prevents just-in-time data product generation. | The rigid schemas are designed for specific use cases; subsequent adaptation is not feasible. |
| *Data Lake* | Use case-specific data preparation is not envisaged in the concept, i.e., any data product is generated just in time from raw data. | As all data products are generated when they are accessed due to the ELT process, they can theoretically be customized at will. |
| | Use-case-specific data transformation, i.e., customization of data products, must be handled externally by customers. If data are provided in various pre-processed stages to facilitate this, high storage costs incur. | |
| *Lakehouse* | As only a non-materialized view on data is provided, data products are generated just-in-time by default. | The underlying metadata are designed for a specific schema and adjustments are not feasible. |
| *Data Mesh* | A data mesh is an organizational approach to promote data decentralization and data product provision but does not offer any technical solutions. | |
| *Data Marketplace* | A data marketplace provides techniques to facilitate providing, discovering, and accessing data but does not address the creation of data products. | |

The required metadata and access structures also cause increased storage costs. While data meshes and data marketplaces put the spotlight on data products, neither offer any solutions for the management of data products and fall back on other data architectures, typically data lakes. Our main findings regarding the usability of the discussed data architectures for the management of data products are summarized in Table 1.

## 3 Related Work

As the foregoing discussion has revealed that current data architectures do not provide sufficient support for data products, we look at the state of research in the following.

*Specification of Data Products.* As data products are an abstract conceptual model, Hasan and Legner [13] attempt to create a harmonized understanding. Key aspects with regard to management are that no redundant data copies should be maintained when administering data products and that a dedicated data product cannot be created for every use case for reasons of scalability. This technical limitation is therefore contrary to market principles, especially in terms of customization. Therefore, metadata must clearly define data products so that customers can still find the most suitable ones. A formal metamodel for data products, such as PROMOTE [5], can be used to this end. Based on such a template, data owners can then use, e.g., a visual inquiry tool to design their data products [12].

*Handling of Data Products.* But even with such metadata, there is an irresolvable asymmetry in terms of the decision as to which products are being offered as well as information sovereignty regarding the products. By means of dedicated metrics, e.g., quality metrics, at least the latter can be addressed to improve the findability of suitable data products [10]. Applying the serverless computing paradigm, customers can then be provided with these products on demand [21]. Technical measures can also be implemented to increase the trustworthiness of data products by ensuring the validity and veracity of the source data. To this end, the lineage of data products can be traced by means of provenance data [25]. This enables customers to inspect the composition of data products. Yet, it would be preferable if customers were able to shape their products themselves from the outset.

*Management of Data Products.* An extended zone architecture for data lakes can be used for managing data products. Besides use-case-independent zones, in which generically pre-processed data are stored alongside unprocessed raw data, use-case-dependent zones can also be introduced. The latter stores highly specialized data that are tailored to a specific use case [9]. However, such an architecture inevitably leads to redundant data storage and thus to high storage costs, which increase even further with each additional use case or data product. Moreover, due to the involved batch processing, it takes a long time until the huge amounts of data are processed, i.e., until a new data product is available. The Delta architecture tackles this problem. For this, the Delta Lake, a framework for building lakehouses [1], is complemented by structured streaming functionalities. The combination of batch and stream processing enables a continuous data flow model. As a result, in addition to accessing data that are already stored in the data lakes, the latest data from the sources can also be accessed in near real-time [17]. Yet, its sole purpose is to be able to provide the latest data as quickly as possible and not to reduce storage costs—all data are still stored redundantly in the data lake zones in the long run. This

is where data fabric, an integrated layer connecting data sources and data consumers, comes into play. With this, a unified end-to-end data platform is established that provides any kind of emerging use case with its required data products. Ingestion, transformation, orchestration, preparation, and curation of data products are fully automated and realized just in time. Albeit data fabric in this form is only a vision for data management [14].

*Findings.* As data mesh and thus the data product paradigm is a recent approach, there is limited work on the management of data products [27]. In particular, there is a lack of concepts that enable the just-in-time production and full customization of data products.

## 4 Introduction of the LALO Approach

To overcome the identified limitations of current data architectures and the state of research in terms of managing data products, namely just-in-time production to reduce storage costs and customization capabilities to fully meet customer needs, we introduce LALO. LALO is intended to enable the provision of tailor-made data products on demand.

For this purpose, we draw on the current state of the art—namely data lakes—as well as our previous work—namely BARENTS [23]. Looking at the data lake concept outlined in Sect. 2 and its extensions discussed in Sect. 3, one notices that this data architecture has undergone an evolution: Starting as a repository for raw data, to a zone architecture for storing pre-processed data, culminating in lakehouses where a small number of proprietary data products are offered via a structured access layer on top of the data lake. This evolution of the data lake approach and its implications for the

**Table 2:** Evolution of the Data Lake Approach and its Ramifications Regarding Data Products.

| Data Lake Approach | Supported Data Products | Ramification on Storage Costs | Ramification on Customization |
|---|---|---|---|
| *Basic Data Lake (Raw Data Only)* | no support | — | — |
| *Data Lake with Predefined Zones* | proprietary data preparation | fixed costs for pre-processed data | no customization |
| *Data Lake with Data Warehouse (e.g., Lakehouse or Data Fabric)* | proprietary data products are implicitly supported due to the data warehouse layer | depends on whether or not the data warehouse layer is materialized | little scope for customization due to the rigid data warehouse schema |
| *Data Lake with Customizable Materialized Zones (BARENTS)* | explicit support for prefabricated tailor-made data products | high storage costs per data product | full customization; yet long lead times due to batch processing |
| *Data Lake with Customizable Virtual Zones (LALO)* | explicit support for tailor-made data products on demand | no storage costs due to stream processing | full customization at all times due to live adaptability |

management of data products is reflected in Table 2. However, all of these approaches have limited support for tailor-made data products.

For this reason, we developed BARENTS as a complement to the zone architecture. It enables users to define custom preparation steps in an ontology, which are then applied to the relevant data via batch processing. For simplicity's sake, the definition of the processing steps can be assumed as a sequence of triples: source, operator, and sink. Source and sink are data lake zones in which the required data can be found and in which the results are to be stored. The operator is a single command—complex processing steps can be realized by concatenating several such triples, whereby the sink of a triple represents the source of the subsequent triple. An excerpt of such an ontology is given in Listing 1. In it, three data preparation steps are applied consecutively: First, all raw data with an index that is not divisible by 5 are selected (lines 2, 3, and 4). Then, data with an empty "notes" attribute are filtered out (lines 8 and 9). Finally, the "revenue" attribute is removed, and the result is stored in the "feature_selection" zone (lines 13, 14, and 15).

This user-defined ontology enables demand-driven data provision in data lakes. Yet, this comes at a high price: a) Due to the materialization of all intermediate data products, BARENTS entails high storage costs. For each data preparation step, an additional zone has to be provided in the data lake in which the results can be stored. This also leads to increased administration costs, as the lifecycle of the data products must be monitored so that zones can be removed when a product is no longer required. In addition to persistent data storages, BARENTS also supports in-memory databases. Yet, they merely shift the problem of rising storage costs from disk storage to main memory. b) Although data products can be fully customized with BARENTS, just-in-time production is not possible due to its static batch-processing approach. This is further complicated by the fact that

**List. 1:** Excerpt from a BARENTS Ontology to Define Exemplary Data Preparation Steps.

```
1  ...
2  <rdf:Description rdf:about="http://barents.dl/raw_zone">
3     <dl:function>lambda x : x.index % 5 != 0</dl:function>
4     <dl:type>filter</dl:type>
5     <dl:partOf rdf:resource="http://barents.dl/data_selection"/>
6  </rdf:Description>
7  <rdf:Description rdf:about="http://barents.dl/data_selection">
8     <dl:function>lambda x : x.notes</dl:function>
9     <dl:type>filter</dl:type>
10    <dl:partOf rdf:resource="http://barents.dl/data_cleansing"/>
11 </rdf:Description>
12 <rdf:Description rdf:about="http://barents.dl/data_cleansing">
13    <dl:function>lambda x : remove_attribute(x.revenue)</dl:function>
14    <dl:type>map</dl:type>
15    <dl:partOf rdf:resource="http://barents.dl/feature_selection"/>
16 </rdf:Description>
17 ...
```

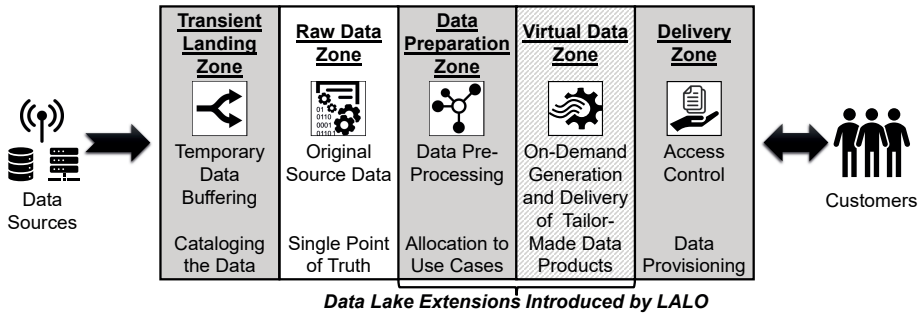



**Fig. 1:** Schematic Data Lake Zone Architecture of LALO Adapted from Stach et al. [23].

the composition of a data product is carried out incrementally—i.e., each processing instruction is handled separately instead of considering the entire process from raw data to data product. It is therefore not possible to optimize the process. This results in long production times before a data product can be accessed.

Hence in LALO, we introduce a novel virtual data lake zone that does not incur any permanent storage costs due to stream-based processing (see Sect. 4.1). Moreover, we introduce a novel mechanism for the live adaptation of data streams (see Sect. 4.2), which facilitates alterations to the data processing at runtime. To this end, we compose a script from all the data preparation steps defined in the ontology for a data product, which is then loaded into the data processor of the stream.

### 4.1 A Virtual Data Lake Zone

Our virtual data lake zone is geared towards the BARENTS zone architecture [23]. In our zone architecture, the data from the source systems are initially ingested and cataloged by the transient landing zone. They are then stored as is in the raw data zone. In LALO, the raw data zone serves as single point of truth and as sole data source. Unlike other zone architectures, we do not store data redundantly in different pre-processing stages. The raw data zone is followed by a data preparation zone, which is configured using an ontology of processing rules and applies those rules to the raw data. This zone would be followed by a multitude of use-case-dependent zones, which serve as storage for the processed results of the data preparation zone. Yet, as discussed before, such an approach is not suitable for data products as the storage costs would be too high. Therefore, the use-case-dependent zones are fully virtualized in LALO by using stream processing. LALO thus adheres completely to the Kappa architecture [18], in which there is only a single processing channel for both, batch and streaming data. All data sources are treated as data streams and processed in real-time. Customers access data products via the delivery zone. Here they can specify which products they are looking for, which adjustments need to be made to existing products, or how new products are to be designed. The schematic representation of the resulting LALO zone architecture is shown in Fig. 1.

The enhanced data preparation zone and the novel virtual data zone represent the extensions introduced by LALO. Their implementation and the interactions with the upstream and downstream zones are depicted in Fig. 2.

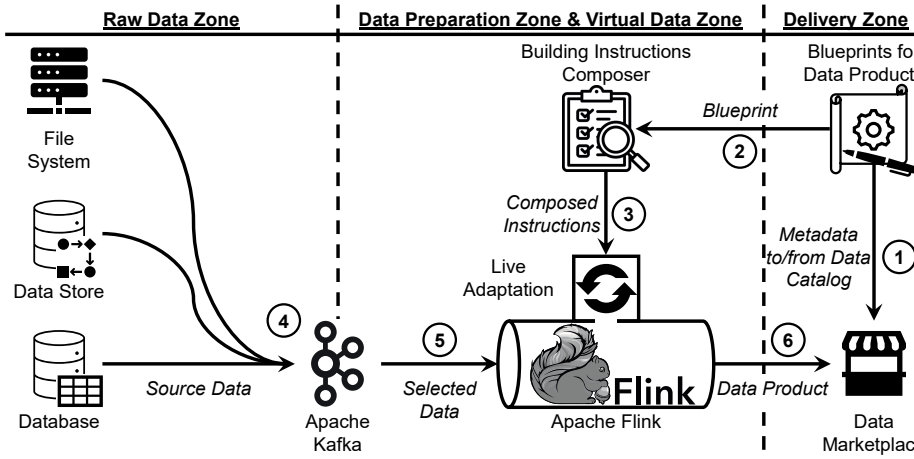



**Fig. 2:** Implementation of the Virtual Data Zone in LALO Based on the Kappa Architecture.

① A customer can use a data marketplace as conceived by Eichler et al. [7], to discover which data products are available. In line with the just-in-time strategy, however, this is not a materialized view on the available products, but rather an overview of the blueprints stored in the data catalog. These blueprints are the sequences of processing steps defined in the ontology that transform raw data into a data product. If none of the blueprints meet the customer's requirements, the customer can make adjustments to an existing blueprint, resulting in a fork in the ontology, or define a completely new blueprint, which adds an entirely new branch to the ontology.

② Once a blueprint has been selected (respectively an existing one adapted or a new one created), it is forwarded to the composer, which assembles the building instructions. As the processing rules in the ontology are single-step commands, all commands that are required to build the selected data product are consolidated in a script by the composer. This is feasible as the source-operator-sink rules can be composed in such a way that the input of a rule is the output of its predecessor [24]. As a result, the commands are not handled independently and optimizations such as lazy evaluation can be applied.

③ The composed script is then forwarded to the stream processing system—in LALO we use Apache Flink for this. Our novel mechanism for the live adaptation of data streams ensures that the script is automatically applied to the data after a negligible reconfiguration time. For more details on this mechanism, see Sect. 4.2. For reasons of scalability, it is possible to run several jobs in parallel in accordance with the Kappa architecture, i.e., more than one type of data product can be produced at the same time.

④ A continuous data ingestion process extracts all data from the source systems that are deemed relevant and loads them into the raw data zone. Therefore, the raw data zone contains the latest data available for the production of a data product. Appropriate metadata management ensures that these data can be accessed by all subsequent zones.

⑤ In LALO, the streaming platform Apache Kafka is used to retrieve the data from the raw data zone, assign a topic to them, and partition them accordingly. The data

required to produce the requested data product (identified via the assigned topics) are forwarded to the stream processing system via Kafka brokers for processing.

⑥ Each ready-to-use data product is delivered directly to the customer via the data marketplace. In this way, the products are at the disposal of the customer as soon as they are finalized, and no storage costs incur for the products in the data architecture.

### 4.2 A Mechanism for the Live Adaptation of Data Streams

To realize such a virtual data lake zone, a stream processing system is required that provides the flexibility needed for such an agile production. In the traditional Kappa architecture, however, a static processing logic is assumed that is to be applied to all data from its source systems. A dynamic adaptation of the processing logic at runtime, e.g., due to evolving customer requirements, is not envisaged. If a revision of the processing logic is necessary, a new instance of the stream processing job with the updated logic must be created first, then started, and all data have to be re-processed by this modified instance. Yet, this leads to high reconfiguration costs for each adaptation. As an alternative, a dedicated job could be started in parallel for each existing blueprint from the outset. However, this does not only represent an unnecessarily high consumption of resources, as presumably not all data products are required at the same time but also limits flexibility—this approach implies that all data products are already known in advance and no adaptations can be made to them. To this end, LALO introduces a mechanism for the live adaptation of data streams for Apache Flink, as shown in Fig. 3.

❶ Incoming data from the raw data zone (represented by $D_{t+1}$) are initially cached in the stream processing system. A FIFO buffer (first in, first out) is used for this purpose in LALO in order to maintain the sequential order of the data.

❷ The FIFO buffer forwards the data immediately for processing unless it receives a retention instruction. Step ❻ reveals why such a retention may be necessary.

❸ The raw data are transformed into data products by the data processor. This component continuously monitors whether new building instructions are available. All currently active production plans are stored in the building instructions buffer as Python scripts that can be applied to the incoming data by the data processor (represented by $D_t$).

❹ If the building instructions have been fully applied to a data item, i.e., the requested data product has been finalized, it is forwarded to the delivery zone (represented by $D_{t-1}$).

❺ If a customer makes alterations to the building instructions, the resulting script is stored in the building instructions buffer and is therefore noticed by the data processor.

❻ If it is necessary to modify the data processing logic, the data processor instructs the FIFO buffer to withhold the data until the update has been completed. The new script is then automatically loaded into the data processor and processing can continue. From then on, all data products are generated and provided according to the altered blueprint. As the dataflow in Apache Flink is modeled as a directed acyclic graph [3], however, such upstream communication is not supported. In LALO, we therefore use Apache Kafka as a message broker for communication between the data processor and the FIFO buffer.

For each data product, a separate data stream is initiated when a customer requests that data product. This allows multiple data products to be generated in parallel.

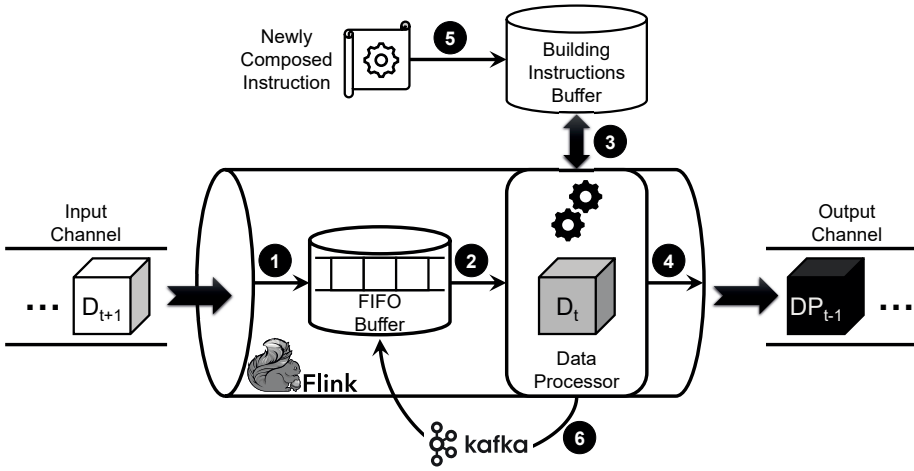



**Fig. 3:** Implementation of the Live Adaptation Mechanism for Data Streams in LALO.

## 5 Discussion of the LALO Approach

After presenting LALO and especially its main components, namely the virtual data lake zone and the mechanism for the live adaptation of data streams, our approach is critically discussed in the following. First, we assess in Sect. 5.1 to what extent the concept of the virtual data lake zone benefits our research goals, i.e., the reduction of storage costs and the just-in-time composing of data products. We then evaluate the performance of our technical implementation of this concept in the form of the live adaptation of data streams in Sect. 5.2. Finally, we summarize our lessons learned in Sect. 5.3.

### 5.1 Assessment of the Virtual Data Lake Zone

LALO is designed to enable the appropriate management of data products. To this end, two requirements have to be addressed:

*Reduction of Storage Costs.* To support data pre-processing geared to the needs of any given use case, it is essential to maintain all raw data. This amount of data can therefore be regarded as the minimum baseline for a data architecture that supports tailor-made data products. However, current data architectures that facilitate this, such as data lakes, store these base data redundantly in various pre-processing stages, which leads to high storage costs. In terms of data products, such a strategy implies that each single data product permanently requires storage space, as they are not consumed when being used.

In LALO, we take advantage of this fact, as not only data products but also raw data are not consumed when being used. That is, it is possible to generate a data product on demand, as these raw data can never be out of stock. Our virtual data lake zone achieves a full virtualization of data storage. Instead of materialized data products, LALO only provides blueprints for data products. If a customer requests a data product, it is generated from raw data just in time using the blueprints and delivered directly to the customer

using stream processing. As a result, data products only incur storage costs on top of the baseline (i.e., the raw data storage) temporarily during production.

*Just-in-Time Composing.* Data products offer the most benefit when they are perfectly geared to their intended use. A generic one-size-fits-all data pre-processing, as applied in data lakes, therefore necessitates considerable additional effort in terms of data refinement by customers. Due to the sheer number of potential use cases, however, dedicated data pre-processing for a few specific use cases, as it is common in lakehouses, is also not an option. Instead, customers must be enabled to design their own tailor-made data products.

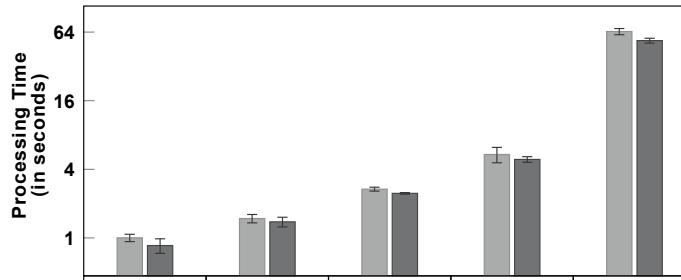
Since LALO adopts the ontology-based specification of the processing rules from BARENTS, both the redesign and customization of such rules are straightforward. Unlike in BARENTS, however, in LALO these rules are not applied as a sequence of isolated commands, but rather as a composed script, which makes optimizations possible. Any changes made to the script are applied to the data stream almost immediately.

*Privacy.* The focus of LALO is solely on functional aspects in order to comply with market principles. However, there is also a need for LALO from a legal perspective. Data protection laws, such as the European General Data Protection Regulation (GDPR), pose special challenges to the management of personal data [8]. Such data are very valuable, making them indispensable to any company. Therefore, companies must comply with the principles of data minimization (Art. 5(1)(c)) and storage limitation (Art. 5(1)(e)), which are core features of LALO. Furthermore, purpose limitation (Art. 5(1)(b)) stipulates that different privacy requirements apply to data processing depending on the customer. This is facilitated by the just-in-time composing of the processing logic in LALO. Finally, data subjects have the right to rectification (Art. 16) or the right to erasure (Art. 17). If all data are stored redundantly in different pre-processing stages and are contained in multiple data products, this represents an administrative nightmare. By keeping the raw data as a single point of truth and generating data products just in time, LALO facilitates all this, as required by Article 25, by design and default. That is, LALO also provides support for compliance with data protection laws in addition to its functional benefits.

### 5.2 Evaluation of the Live Adaptation of Data Streams

As LALO possesses the functional capabilities to enable appropriate management of data products, we now evaluate whether it is also efficient. To this end, we consider data throughput, as this is a key metric in stream processing. Yet, as throughput depends largely on the complexity of the processing logic—i.e., it differs from one data product to the next—we introduce a representative example. Algorithm 1 shows the five processing steps that are necessary for our exemplary data product. In accordance with Listing 1, a revenue dataset of a company is initially retrieved (step 1) and cleansed (steps 2 and 3). Then, the mean shipped units per division and the accumulated notes on these shipments are determined (steps 4 and 5). The script includes selections (selection of tuples), projections (selection of attributes), and aggregations combined with a group-by operator to represent a wide range of typical pre-processing tasks.

We implemented these five instructions as ontology-based processing rules, which can be processed by LALO. We also generated five synthetic revenue datasets with




| | 1,000 k | 2,000 k | 4,000 k | 8,000 k | 16,000 k |
|---|---|---|---|---|---|
| BARENTS | 1.00238 s | 1.48085 s | 2.68044 s | 5.39692 s | 64.69793 s |
| LALO | 0.85748 s | 1.38411 s | 2.46149 s | 4.88507 s | 53.87698 s |

Size of Processed Dataset (in tuples)

**Fig. 4:** Evaluation Results for the Five Datasets: Mean Processing Time and Standard Deviation.

1,000 k to 16,000 k data items, i.e., a data volume between approximately 320 MB and 5 GB. In each dataset, we increased the amount of data items by a factor of two. The integer-based index of the data items is consecutive and starts at 0—i.e., in step 1, 20% of the tuples are filtered out. 80% of the data items contain random notes with a length of 128 bytes each, while the other notes are empty—i.e., another 20% of the tuples are filtered out in step 2. All remaining attributes are filled with random values. There are five distinct divisions in the database—i.e., five groups are formed in step 4.

To determine whether LALO is efficient, we compare it with BARENTS. For this evaluation, each dataset is processed once by LALO and BARENTS according to Algorithm 1 and we determine the processing time. In our prototype, we use Python 3.12.2 as the runtime environment and Polars 0.20 as the execution engine. The measurements were carried out on a desktop computer with an Intel Core i7-1165G7 with four cores and 16 GB DDR4-3200 of main memory. We ran these measurements ten times. The mean processing time and the standard deviation are shown in Fig. 4.

It can be seen that a significant speedup of almost 17% can be achieved due to the code optimizations in LALO. It also has to be noted that we have factored out the additional I/O costs of BARENTS—actually, BARENTS would store all intermediate results after each single command and then retrieve these data again to apply the next command. In this light, the performance advantage of LALO is even more impressive.

**Algorithm 1:** Exemplary Blueprint for a Data Product Used for Evaluation.

**Data:** *dataset:* (index, division, shipped_units, revenue, notes)
**Result:** *data product:* mean shipped units per division and accumulated notes

1. filter out all tuples with an index that is divisible by 5;
2. remove all tuples which do not include additional notes;
3. drop the revenue attribute;
4. group the tuples by their division;
5. aggregate the number of shipped units (*mean*) and the notes (*concatenate*);

We also used our prototype to determine the costs incurred by the live adaptation of data streams. The overhead for preparing the data processing (i.e., retrieving a new script and parsing the contained commands) is only 2.4 ms. This can be considered negligible.

### 5.3 Lessons Learned

The performance measurements have shown that our approach using composed scripts achieves a distinct speedup compared to the execution of a sequence of isolated commands. The overhead incurred when adapting a script is also negligible. Admittedly, the just-in-time generation of data products naturally takes more time than if they are already available ready-made in the data architecture. However, this is not reasonable given the market principles for appropriate management of data products. For this, just-in-time production as well as complete customization are required—both of which are provided by LALO. Furthermore, long-term and redundant storage of preprocessed data and finalized data products is not an option for privacy reasons as well. Accordingly, LALO is not only an effective solution for composing tailor-made data products on demand but also facilitates compliance with data protection laws.

Although LALO is explicitly designed for data lakes, this is not a limitation of our approach. Combined with data fabric, data lakes provide the basis for a flexible lakehouse. Data lakes are also a commonly used data architecture in data meshes at the domain level. They also represent an important data backend for data marketplaces. LALO therefore makes a substantial contribution to all current data architectures.

## 6 Conclusion and Outlook

The new understanding of data as a valuable commodity that can be transformed into a profitable data product if processed appropriately implies that the same market principles apply to data as to physical goods. This poses new challenges for the management of data products which current data architectures are unable to cope with. In particular, this concerns the just-in-time preparation of data products to reduce storage costs as well as the full customization of such products so that they have the best utility for customers.

LALO addresses these challenges: a) By introducing a virtual data zone for data lakes, we enable the virtualization of data storage. It is possible to provide blueprints instead of materialized data products and do the actual generation just in time. This is realized using the Kappa architecture and data streams, i.e., there accrue no storage costs. b) A key component that makes this possible is our mechanism for live adaptation of data streams. By composing building instructions and deploying them at runtime, we achieve a 17% speedup in the generation of fully customized data products. This makes LALO a valuable solution for composing tailor-made data products on demand.

Although LALO is a worthwhile contribution to the management of data products, it marks only the first cornerstone. Our next steps focus on three aspects: First, a hybrid BARENTS/LALO solution may be useful. Frequently used or standardized data products could then be provided in a materialized form, while highly individualized data products could be composed on demand. Second, a cross-domain virtualization layer could be considered when LALO is used in a data mesh at a domain level to provide data products.

This involves combining and harmonizing blueprints from multiple domains to support just-in-time generation of data products that are composed of data products from several domains. Third, the introduction of dedicated privacy operators for the blueprints to obfuscate personal information when creating data products could further facilitate compliance with the GDPR. However, these conceptions are just food for future work.

**Acknowledgments.** This paper is part of the SofDCar research project (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK).

## References

1. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Łuszczak, A., undefinedwitakowski, M., Szafrański, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., Xin, R., Zaharia, M.: Delta lake: high-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment **13**(12), 3411–3424 (2020), https://doi.org/10.14778/3415478.3415560
2. Armbrust, M., Ghodsi, A., Xin, R., Zaharia, M.: Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In: Proceedings of the 11th Annual Conference on Innovative Data Systems Research, CIDR (2021), URL http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf
3. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache Flink™: Stream and Batch Processing in a Single Engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **36**(4), 28–38 (2015)
4. Dehghani, Z.: Data Mesh: Delivering Data-Driven Value at Scale. O'Reilly (2022)
5. Driessen, S., den Heuvel, W.J.v., Monsieur, G.: ProMoTe: A Data Product Model Template for Data Meshes. In: Proceedings of the 42nd International Conference on Conceptual Modeling, ER (2023), https://doi.org/10.1007/978-3-031-47262-6_7
6. Eichler, R., Gröger, C., Hoos, E., Schwarz, H., Mitschang, B.: From Data Asset to Data Product – The Role of the Data Provider in the Enterprise Data Marketplace. In: Proceedings of the 16th Symposium and Summer School on Service-Oriented Computing, SummerSOC (2022), https://doi.org/10.1007/978-3-031-18304-1
7. Eichler, R., Gröger, C., Hoos, E., Stach, C., Schwarz, H., Mitschang, B.: Introducing the enterprise data marketplace: a platform for democratizing company data. Journal of Big Data **10**, 173 (2023), https://doi.org/10.1186/s40537-023-00843-z
8. Forgó, N., Hänold, S., Schütze, B. (eds.): New Technology, Big Data and the Law. Springer (2017)
9. Giebler, C., Gröger, C., Hoos, E., Schwarz, H., Mitschang, B.: A Zone Reference Model for Enterprise-Grade Data Lake Management. In: Proceedings of the 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, EDOC (2020), https://doi.org/10.1109/EDOC49727.2020.00017
10. Guggenberger, T.M., Altendeitering, M., Schlueter Langdon, C.: Design Principles for Quality Scoring—Coping with Information Asymmetry of Data Products. In: Proceedings of the Hawaii International Conference on System Sciences 2024, HICSS (2024), URL https://hdl.handle.net/10125/106928
11. Harby, A.A., Zulkernine, F.: From Data Warehouse to Lakehouse: A Comparative Review. In: Proceedings of the 2022 IEEE International Conference on Big Data, Big Data (2022), https://doi.org/10.1109/BigData55660.2022.10020719

12. Hasan, M.R., Legner, C.: Data Product Canvas: A Visual Inquiry Tool Supporting Data Product Design. In: Proceedings of the 18th International Conference on Design Science Research in Information Systems and Technology, DESRIST (2023), https://doi.org/10.1007/978-3-031-32808-4_12
13. Hasan, M.R., Legner, C.: Understanding Data Products: Motivations, Definition, and Categories. In: Proceedings of the 2023 European Conference on Information Systems, Research Papers, ECIS (2023), URL https://aisel.aisnet.org/ecis2023_rp/229
14. Hechler, E., Weihrauch, M., Wu, Y.: Data Fabric and Data Mesh Approaches with AI: A Guide to AI-based Data Cataloging, Governance, Integration, Orchestration, and Consumption. Apress (2023)
15. Inmon, B.: Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump. Technics Publications (2016)
16. Inmon, W.H.: Building the Data Warehouse. Wiley, 4th edn. (2005)
17. Kraetz, D., Morawski, M.: Architecture Patterns—Batch and Real-Time Capabilities. In: Liermann, V., Stegmann, C. (eds.) The Digital Journey of Banking and Insurance, Volume III: Data Storage, Data Processing and Data Analysis, pp. 89–104, Springer (2021), https://doi.org/10.1007/978-3-030-78821-6_6
18. Kreps, J.: Questioning the Lambda Architecture. Radar Article, O'Reilly (2014), URL https://www.oreilly.com/radar/questioning-the-lambda-architecture/
19. Machado, I.A., Costa, C., Santos, M.Y.: Data Mesh: Concepts and Principles of a Paradigm Shift in Data Architectures. Procedia Computer Science **196**, 263–271 (2022), https://doi.org/10.1016/j.procs.2021.12.013
20. Majchrzak, J., Siwiak, M., Balnojan, S.: Data Mesh in Action. Manning (2023)
21. Sedlak, B., Pujol, V.C., Donta, P.K., Werner, S., Wolf, K., Falconi, M., Pallas, F., Dustdar, S., Tai, S., Plebani, P.: Towards Serverless Data Exchange Within Federations. In: Proceedings of the 17th Symposium and Summer School on Service-Oriented Computing, SummerSOC (2023), https://doi.org/10.1007/978-3-031-45728-9_9
22. Stach, C.: Data Is the New Oil–Sort of: A View on Why This Comparison Is Misleading and Its Implications for Modern Data Administration. Future Internet **15**(2), 71:1–71:49 (2023), https://doi.org/10.3390/fi15020071
23. Stach, C., Bräcker, J., Eichler, R., Giebler, C., Mitschang, B.: Demand-Driven Data Provisioning in Data Lakes: BARENTS — A Tailorable Data Preparation Zone. In: Proceedings of the 23rd International Conference on Information Integration and Web Intelligence, iiWAS (2021), https://doi.org/10.1145/3487664.3487784
24. Stach, C., Eichler, R., Schmidt, S.: A Recommender Approach to Enable Effective and Efficient Self-Service Analytics in Data Lakes. Datenbank-Spektrum **23**(2), 123–132 (2023), https://doi.org/10.1007/s13222-023-00443-4
25. Subramanian, G., Nagabushanam, H.: Governance of Data Product in Multi-layered IoT system. In: Proceedings of the 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering, ICECCME (2022), https://doi.org/10.1109/ICECCME55909.2022.9987960
26. Tien, J.M.: Toward the Fourth Industrial Revolution on Real-Time Customization. Journal of Systems Science and Systems Engineering **29**(2), 127–142 (2020), https://doi.org/10.1007/s11518-019-5433-9
27. Wider, A., Verma, S., Akhtar, A.: Decentralized Data Governance as Part of a Data Mesh Platform: Concepts and Approaches. In: Proceedings of the 2023 IEEE International Conference on Web Services, ICWS (2023), https://doi.org/10.1109/ICWS60048.2023.00101