



Data Lakehouse Reference Architecture

JUNE 2025

1.	Introduction	3
2.	What is a Data Lakehouse?	3
3.	Conceptual Architecture	4
4.	A Concrete Architecture	6
5.	A Few Words on Data Sources	7
6.	The Ingestion Layer	7
7.	The Data Storage Layer	9
8.	The Processing Layer	11
9.	The Optional Semantic Layer	12
10.	The Consumption Layer	13
11.	Data Lakehouse Security	14
12.	Summary	16



1. Introduction

As businesses strive to unlock the full potential of their data assets, the need for a flexible, scalable, and unified approach to data storage and analytics has become top of mind for enterprise architects who need to build out infrastructure that supports the needs of the business. A data lakehouse architecture addresses this need by integrating a data lake for storing unstructured data with a data warehouse for structured data. This whitepaper serves as a guide to understanding and implementing a data lakehouse architecture.

This paper is organized as follows. Section 2 will provide a brief definition of the data lakehouse. Section 3 will present a conceptual architecture. Section 4 will show how this conceptual architecture could be implemented with open-source tools. Section 5 will discuss data sources, their varying data formats and the different techniques they may use to send their data to a data lakehouse. Sections 6 through 10 will present the five layers of a data lakehouse (Ingestion Layer, Storage Layer, Processing Layer, Semantic Layer and the Consumption Layer). Finally, Section 11 will cover Data Lakehouse Security.

2. What is a Data Lakehouse?

A data lakehouse is one-half data warehouse and one-half data lake and uses [object storage](#) for everything. This may sound like a marketing trick - put two products in one package and call it a new product. However, the data warehouse presented in this paper is better than a conventional data warehouse since it uses object storage; therefore, it provides all the benefits of object storage in terms of scalability and performance. Furthermore, organizations that adopt this approach will only pay for what they need (facilitated by the scalability of object storage), and if blazing speed is required, they can equip their underlying object store with NVMe drives connected by a high-speed network.

The use of object storage in this fashion has been made possible by the rise of Open Table Formats (OTFs) like Apache Iceberg, Apache Hudi, and Delta Lake, which are specifications that, once implemented, make it seamless for object storage to be used as the underlying storage solution for a data warehouse. These specifications also provide features that may not exist in a conventional data warehouse - for example, time travel, schema evolution, partitions, partition evolution, and zero-copy branching. They are also capable of ACID transactions, which makes them an excellent replacement for Hadoop, HDFS (Hadoop Distributed File System), and Hive, which are starting to show their age.

As stated above, the data lakehouse is more than just a fancy data warehouse - it also contains a data lake for unstructured data. The OTFs also provide integration to external data



in the data lake. This integration allows external data to be used as a SQL table if needed, or the external data can be transformed and routed to the data warehouse using high-speed processing engines (described later) and familiar SQL commands. For the remainder of this paper, whenever a data warehouse is mentioned, the reference is to an OTF-based data warehouse.

So the data lakehouse is more than just a data warehouse and a data lake in one package with a different name. Collectively, they provide more value than what can be found in a conventional data warehouse and a standalone data lake.

3. Conceptual Architecture

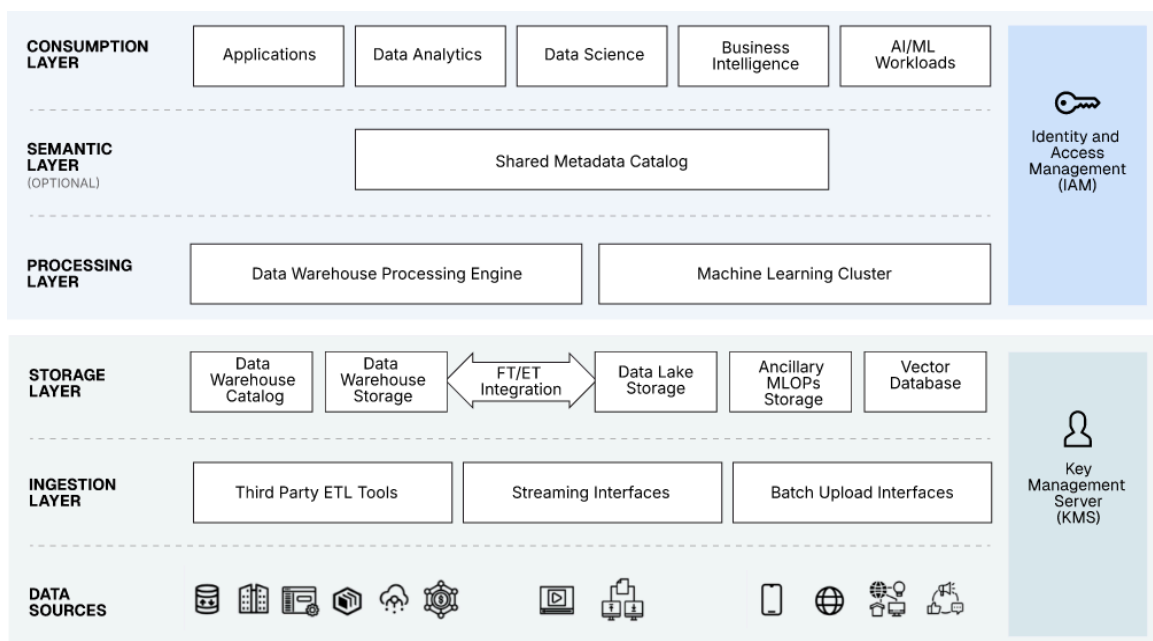
Layering is a convenient way to present all the components and services needed for a data lakehouse solution. Layering provides a clear way to group services that provide similar functionality. It also establishes a hierarchy, with consumers on top and data sources (with their raw data) on the bottom. The layers of the data lakehouse from top to bottom are:

- Consumption Layer - Contains the tools used by power users to analyze data. Also contains applications and AI/ML workloads that will programmatically access the data lakehouse.
- Semantic Layer - An optional metadata layer for data discovery and governance.
- Processing Layer - Contains the compute clusters needed to query the data warehouse. It also contains compute clusters used for distributed model training that access the data lake. Complex transformations can occur in the processing layer by taking advantage of the storage layer's integration between the data lake and the data warehouse.
- Storage Layer - Object storage is the primary storage service for the data lakehouse; however, MLOP tools may need other storage services, such as relational databases. Also, if you are pursuing generative AI, you will need a vector database, which is also in this layer.
- Ingestion Layer - Contains the services needed to receive data. The data lakehouse should support a variety of protocols. It should also support data arriving in streams and batches. Advanced ingestion layers will be able to retrieve data based on a schedule. Simple and complex data transformations can occur in the ingestion layer.



- **Data Sources** - The data sources layer is technically not a part of the data lakehouse solution, but it is included in this paper because a well-constructed data lakehouse must support a variety of data sources with varying capabilities for sending data.

The diagram below visually depicts all the layers described above and all the capabilities that may be needed to implement these layers. This is an end-to-end architecture where the heart of the platform is a data lakehouse. Rather than focusing on just the processing layer and the storage layer, this architecture also shows capabilities needed to ingest, transform, discover, govern, and consume data. The tools required to support important workloads that depend on a data lakehouse are also included, such as MLOps storage, vector databases, and machine learning clusters.

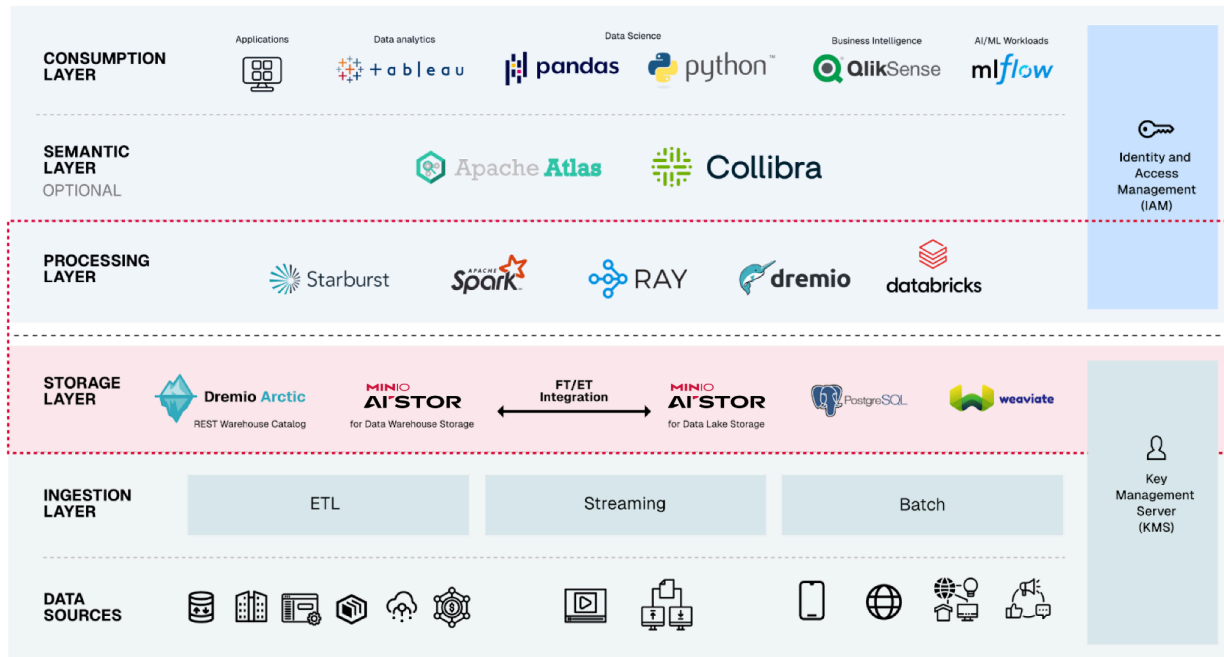


The conceptual nature of the approach used in this paper is important. If the diagram above used product names, then the meaning would be lost. Product names are rarely chosen for meaning - rather, they are selected for brand awareness and memory retention. To this end, our conceptual architecture uses simple nouns where the feature provided is intuitive. The following section will give an example of a concrete implementation for the reader familiar with the more popular big data projects and products in the market today.

Finally, there are no arrows. Arrows typically depict data flow and dependencies. Showing all possible data flows and dependencies would unnecessarily complicate the diagram. A better approach is to look at data flow and dependencies in the context of a use case. Once a few components are isolated in the context of a use case, then data flow and dependencies can be more clearly illustrated.

4. A Concrete Architecture

The purpose of this section is to ground the design of our reference architecture with concrete open-source examples. For the architect eager to dive in and start building, the projects and products shown below are free to use in a proof of concept. When your POC graduates to a funded project that will one day run in production, be sure to check open source licenses and terms of use for all software used in your POC.



A complete list of data lakehouse commercial products is listed below. These companies have products that are data lakehouse examples using AIStor for both sides of the data lakehouse.

- [Dremio](#)
- [Starburst](#)
- [Trino](#)



5. A Few Words on Data Sources

The applications, devices, and vendors that feed your data lakehouse come in a variety of flavors, and so does their data. On-premise modern applications may be able to stream well-structured data in real time using formats such as AVRO and Parquet. On the other hand, older legacy applications may only be able to send simple files in batches, such as XML, JSON, and CSVs. Data vendors may not send data at all, expecting their customers to retrieve data.

Mobile apps, websites, IOT Devices, and social media apps will typically send application logs and other telemetry (usage statistics) to your ingestion layer. Log analytics is a popular use case for a data lakehouse. Additionally, they may send images and audio files to be used within AI/ML workloads.

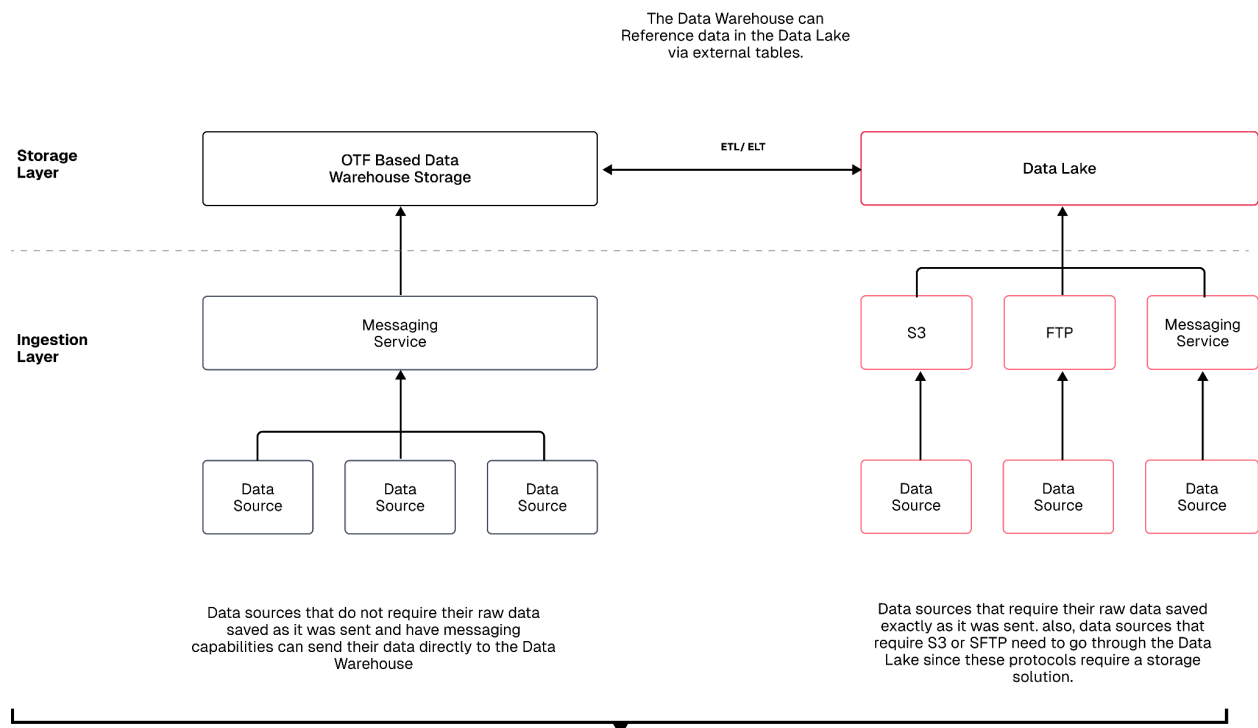
Finally, organizations looking to take advantage of generative AI and agentic AI will need to store documents found in file shares and portals such as SharePoint Portal Server and Confluence in the Data Lake.

The data lakehouse needs to interface with all these data sources efficiently and reliably, storing unstructured data in the data lake side of the data lakehouse and structured data in the data warehouse. Onboarding all this data is the primary purpose of the Ingestion Layer of our architecture. This requires your ingestion layer to support a variety of protocols capable of receiving streamed and batched data. Let's investigate the components of this layer next.

6. The Ingestion Layer

The ingestion layer is the on-ramp to your data lakehouse. It is responsible for ingesting data into the Data Storage Layer. Structured data from sources that designed their feeds for the data warehouse can bypass the data lake and send their data directly to the data warehouse. On the other hand, sources that did not design their feeds in such a fashion will need to have their data sent to the data lake, where it can be transformed before being ingested into the data warehouse.





Data Lakehouse = OTF Based Data Warehouse + Data Lake

The ingestion layer should be able to receive and retrieve data. Internal lines of business (LOB) applications may have been given the mandate to send their data via streaming or batching. For these applications, the ingestion layer needs to provide an endpoint for receiving the data. However, Data Vendors and other external data sources may not be so willing to deliver data. The ingestion layer should also provide scheduled retrieval capabilities. For example, a data vendor may provide new datasets at the beginning of every month. Scheduled retrieval capabilities will allow for the ingestion layer to connect and download data at the correct time.

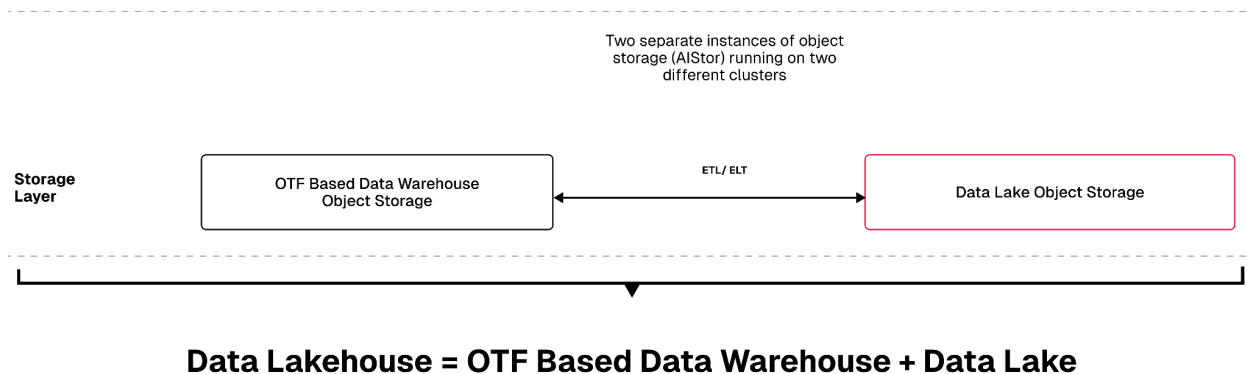
Streaming is the best way to transmit data to a data lakehouse or to any destination for that matter. Streaming implies using a messaging service deployed in a way that makes it resilient, available and highly performant. The messaging service usually provides a queuing mechanism that acknowledges the receipt of a message only upon successful storage of the message. The service then provides “exactly once” delivery to a downstream service responsible for saving the data in the message to either the data warehouse or the data lake. (Note: Some message services provide “at least once” delivery, requiring downstream services to implement idempotent updates to the data source. It is important to check the fine print of the service you use.) What is especially nice about this style of ingestion is that if the downstream service fails and does not acknowledge the successful processing of a

message, then the message will reappear in the queue for future ingestion. Messaging services also provide “dead letter queues” for messages that repeatedly fail.

Streaming ingestion is great, but in many cases, real-time insights are not needed. In these situations, batch or mini-batch processing works fine and can be considerably simpler to implement. For batch uploads, the S3 API is your best option. AIStor is fully S3 compliant, and any data source currently sending batch data to an S3 endpoint will work “as-is” with only a connection change once you switch over to the AIStor data lake. However, many organizations may still prefer FTP/SFTP for its simplicity and ability to run in highly constrained environments. AIStor also has support for FTP and SFTP. This interface allows a data source to send data to AIStor the same way it would send data to an FTP server. From an application or user's perspective, moving data onto AIStor using SFTP is seamless since everything is essentially the same - from policies, security, etc.

7. The Data Storage Layer

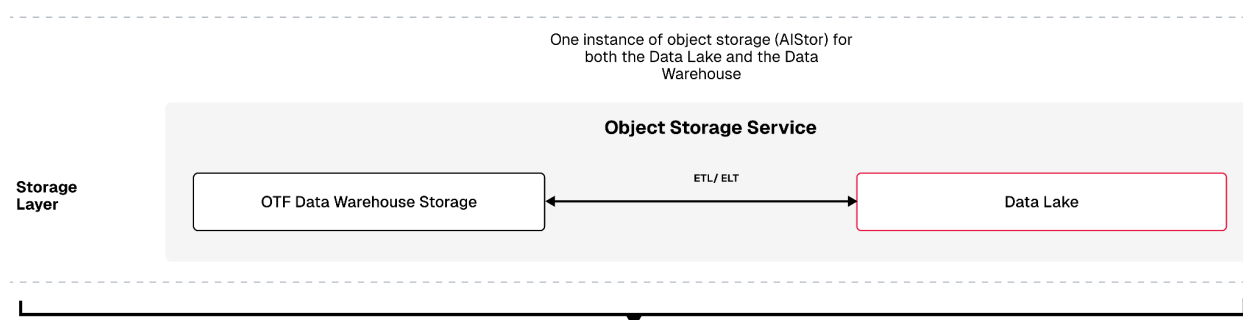
The Data storage layer is the bedrock that all other layers depend upon. Its purpose is to store data reliably and serve it efficiently. There could be an object storage service for the data lake side of the data lakehouse and a separate object storage service for the data warehouse. This option uses two separate namespaces for each half of the data lakehouse.



Another option is to use one physical instance of an object store (one namespace) by using buckets to keep the data warehouse storage separate from data lake storage. However, consider keeping them separate and installed on different hardware if the processing layer will be putting different workloads on these two storage services. For example, a common data flow is to have all new data land in the data lake. Once in the data lake, it can be transformed and ingested into the data warehouse, where it can be consumed by other applications and used for the purpose of data science, business intelligence, and data analytics. If this is your data flow, then your data lakehouse will be putting more load on your



data warehouse, and you will want to make sure it is running on high-end hardware (storage devices, storage clusters, and network).



Data Lakehouse = OTF Based Data Warehouse + Data Lake

External table functionality allows data warehouses and processing engines to read objects in the data lake as if they were SQL tables. If the data lake is used as the landing zone for raw data, then this capability, along with the data warehouse's SQL capabilities, can be used to transform raw data before inserting it into the data warehouse. Alternatively, the external table could be used "as-is" and joined with other tables and resources inside the data warehouse without it ever leaving the data lake. This pattern can help save on migration costs and can overcome some data lakehouse security concerns by keeping the data in one place while, at the same time, making it available to outside services.

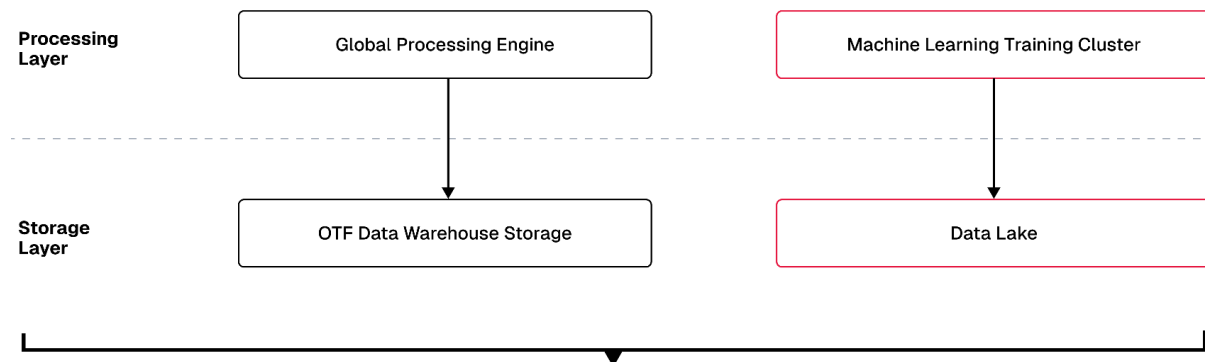
Machine Learning Operations (MLOps) tools use this layer as well. These tools use a combination of a data lake and a relational database to support MLOps functionality. For example, an MLOp tool should store training metrics, hyperparameters, model checkpoints, and dataset versions. Models and datasets should be stored in the data lake, while metrics and hyperparameters will be more efficiently stored in a relational database.

If you are pursuing Generative AI, you will need to build a custom corpus for your organization. It should contain documents with knowledge that no one else has and only documents that are true and accurate should be used. Furthermore, your custom corpus should feed a vector database. A vector database indexes, stores, and provides access to your documents alongside their vector embeddings, which are numerical representations of your documents. Vector databases facilitate semantic search, which is needed for Retrieval Augmented Generation (RAG) - a technique generative AI utilizes to marry information in your custom corpus to an LLMs trained parametric memory.



8. The Processing Layer

The processing layer contains the compute needed for all the workloads supported by the data lakehouse. At a high level, compute comes in two flavors: Processing engines for the data warehouse and clusters for distributed machine learning.



Data Lakehouse = OTF Based Data Warehouse + Data Lake

The data warehouse processing engine supports the distributed execution of SQL commands against the data in data warehouse's storage. Transformations that are part of the ingestion process may also need the compute power in the processing layer. For example, some data warehouses may wish to use a medallion architecture, while others may choose a star schema with dimensional tables. These designs often require substantial ETL against the raw data during ingestion.

The data warehouse used within a data lakehouse disaggregates compute from storage. So, if needed, multiple processing engines can exist for a single data warehouse. (This differs from a conventional relational database where compute and storage are tightly coupled, and there is one compute resource for every storage device.) A possible design for your processing layer is to set up one processing engine for each entity in the consumption layer. For example, a processing cluster for business intelligence, a separate cluster for data analytics, and yet another for data science. Each processing engine would query the same data warehouse storage service; however, since each team has its own dedicated cluster, they do not compete with each other for compute. If the business intelligence team is running month-end reports that are compute-intensive, then they will not interfere with another team that may be running daily reports.



Machine learning models, especially Large Language Models (LLMs), can be trained faster if training is done in a distributed fashion. The Machine learning cluster supports distributed training. Distributed training should be integrated with an MLOps tool for experiment tracking and checkpointing.

9. The Optional Semantic Layer

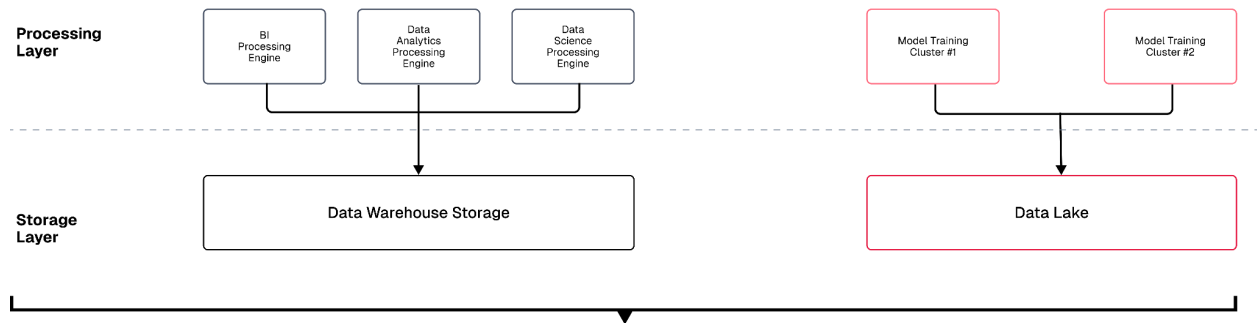
A semantic layer helps the business understand its data. The semantic layer sits between the processing layer, which serves up the data from the storage layer, and the Consumption layer, which contains the tools and applications looking for data. It acts like a translator that bridges the gap between the language of the business and the technical terms used to describe data. It also helps both data professionals and business users find relevant data for either end-user reports or dataset creation for AI/ML.

In its simplest form, the Semantic layer could be a data catalog or an organized inventory of data. A data catalog typically includes the original data source location (lineage), schema, short description, and long description. A more robust Semantic layer can provide security, privacy, and governance by incorporating policies, controls, and data quality rules.

This layer is optional. Organizations with few data sources and well-structured feeds may not need a semantic layer. A well-structured feed contains intuitive field names and accurate field descriptions that can be easily extracted from data sources and loaded into the data warehouse. Well-structured feeds should also implement data quality checks at the source so that only quality data is transmitted to the data lakehouse.

However, large organizations with many data sources where metadata was an afterthought when schemas and feeds were designed should consider implementing the semantic layer. Many of the products that can be used in this layer provide features that help an organization populate a metadata catalog. Also, organizations that operate in complex industries should consider a semantic layer. For example, industries like Financial Services, Healthcare and Legal heavily use terms that are not everyday words. When these domain-specific terms are used as table names and field names, the underlying meaning of the data can be hard to ascertain.





Data Lakehouse = OTF Based Data Warehouse + Data Lake

10. The Consumption Layer

Let's conclude our presentation of the data lakehouse layers by looking at the workloads run in the topmost layer, the Consumption Layer, and discussing how the layers below support their specific use cases. Many of these workloads are often used interchangeably or synonymously - this is unfortunate because it is better to have precise definitions when investigating their needs. In the discussion below, a precise description for each workload will be given along with a description of which parts of this reference architecture support it.

Applications - Custom applications can programmatically send SQL Queries to the data warehouse to create custom views and reports for end users. Applications may also save and retrieve data from the data lake using S3. These may be the same applications that submitted raw data as data sources at the bottom of the diagram. A use case that a data lakehouse should support is to allow applications to submit raw data, clean it, combine it with other data and finally serve it up quickly. Applications may also use models trained with data from the data lakehouse. This is another use case that the data lakehouse should support. Applications should be able to send raw data to the data lakehouse, get it processed, and sent to model training pipelines - from there, the trained models can be used to make predictions within the application.

Data Science is the study of data. Data scientists design the datasets and potentially the models that will be trained and used for inference. Datasets made up of structured data will be in the data warehouse and unstructured datasets will be in the data lake. Data scientists use techniques from mathematics and statistics for the purpose of feature engineering. Feature engineering is a technique for improving datasets used to train a model. A very slick feature that data lakehouses possess is Zero-copy branching, which allows data to be branched the same way code can be branched within a Git repository. As the name suggests,



this feature does not make a copy of the data - rather, it makes use of the metadata layer of the open table format used to implement the data warehouse to create the appearance of a unique copy of the data. Data scientists can experiment with a branch - if their experiments are successful, then they can merge their branch back into the main branch for other data scientists to use.

Business Intelligence is often retrospective, providing insights into past events. It involves the use of reporting tools, dashboards, and key performance indicators (KPIs) to provide a view into business performance. Much of the data needed for BI is aggregations, which can require a fair amount of compute to create. Business intelligence is usually supported by the data warehouse.

Data analytics, on the other hand, involves the analysis of data to extract insights, identify trends, and make predictions. It is more forward-looking and aims to understand why certain events occurred and what might happen in the future. Since data analytics is about predicting the future it should also use AI. Data analytics is usually supported by the data warehouse.

Machine Learning - the machine learning workload is where AI/ML engineers train models. As they run their experiments during model training they will use MLOps tooling to track their experiments and checkpoint their models. This workload is primarily supported by the data lake which needs to serve data in a high performance fashion to the GPUs used to train the models. MLOps tooling also uses the data lake to save unstructured artifacts created during model training such as model checkpoints. Many MLOps tools also use a relational database to save metrics created during model training and testing. In theory, this data could be stored in the data warehouse but most MLOps in the industry today use a relational database of their choice.

11. Data Lakehouse Security

Data Lakehouse security must provide authentication and authorization for users and services. It should also offer encryption for data at rest and data in motion. This section will look into these aspects of data lakehouse security.

The data lake and the data warehouse must support an Identity and Access Management (IAM) solution that facilitates authentication and authorization. Both halves of the data lakehouse should use the same directory service to track users and groups, allowing users to present their corporate credentials when signing into the user interface for both the data lake and the data warehouse. Since each product requires a different connection type for programmatic access, the credentials that need to be presented for authentication will be different. Likewise, the policies used for authorization will also be different, as the underlying



resources and actions are different. The data lake requires authorization for buckets and objects. It also needs authorization for bucket and object actions. On the other hand, the data warehouse needs tables and table-related actions to be authorized.

Data Lake Authentication

Every connection to the data lake requires verification of identity and the data lake should integrate with the organization's identity provider. Since the data lake is an object store that is S3 compliant, the [AWS Signature Version 4 protocol](#) should be used. For programmatic access, this means that each service wishing to access an administrative API or an S3 API, such as PUT, GET, and DELETE operations, must present a valid access key and secret key.

Data Lake Authorization

Authorization is the act of restricting the actions and resources of an authenticated client. An S3-compliant object store should use Policy-Based Access Control (PBAC), where each policy describes one or more rules that describe the permissions of a user or group of users. The data lake should support S3-specific [actions](#) and [conditions](#) when creating policies. By default, AIStor denies access to actions or resources not explicitly referenced in a user's assigned or inherited policies.

Data Warehouse Authentication

Similar to the data lake, every connection to the data warehouse must be authenticated and the data warehouse should integrate with the organization's identity provider for authenticating users. A data warehouse may provide the following options for programmatic access: ODBC connection, JDBC connection, or REST session. Each will require an access token.

Data Warehouse Authorization

A data warehouse should support user, group, and role level access controls for tables, views, and other objects found in the data warehouse. This allows access to individual objects to be configured based on the user's ID, a group, or a role.



Key Management Server

The data lakehouse uses a Key Management Server (KMS) for security at rest and in transit. A KMS is a service responsible for generating, distributing, and managing cryptographic keys used for encryption and decryption.

12. Summary

There you have it, the five layers of a data lakehouse from data sources to consumption. This paper explored a conceptual reference architecture for data lakehouses. The goal is to provide organizations with a strategic blueprint for building a platform that efficiently manages and extracts value from their vast and diverse data sets. The data lakehouse combines the strengths of traditional data warehouses and flexible of data lakes, offering a unified and scalable data lakehouse solution for storing, processing, and analyzing data.