Datenbank Spektrum (2017) 17:289–293
https://doi.org/10.1007/s13222-017-0272-7

KURZ ERKLÄRT

Data Lakes

Christian Mathis[1]

Published online: 6 October 2017

Abstract By moving data into a centralized, scalable storage location inside an organization – the data lake – companies and other institutions aim to discover new information and to generate value from the data. The data lake can help to overcome organizational boundaries and system complexity. However, to generate value from the data, additional techniques, tools, and processes need to be established which help to overcome data integration and other challenges around this approach. Although there is a certain agreed-on notion of the central idea, there is no accepted definition what components or functionality a data lake has or how an architecture looks like. Throughout this article, we will start with the central idea and discuss various related aspects and technologies.

1 Motivation

The term "data lake" was coined 2010 by James Dixon to distinguish between the approach to manage data on Hadoop and data marts or warehouses: "If you think of a data mart as a store of bottled water – cleansed and packaged and structured for easy consumption – the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples." [1] This informal "definition" leaves room for interpretation. As a result, although there is a certain agreed-on notion of the central idea, there is no accepted definition what components or functionality a data lake has or how an architecture looks like. Similar to terms like "Big Data" or "Cloud", the term is used in various contexts with different meanings. Often Apache Hadoop is used as a synonym for data lakes. However, many new technologies have emerged that fit into the picture. Throughout this article, we will start with the central idea and discuss various related aspects and technologies. Figure 1 summarizes the concepts as a topic map.

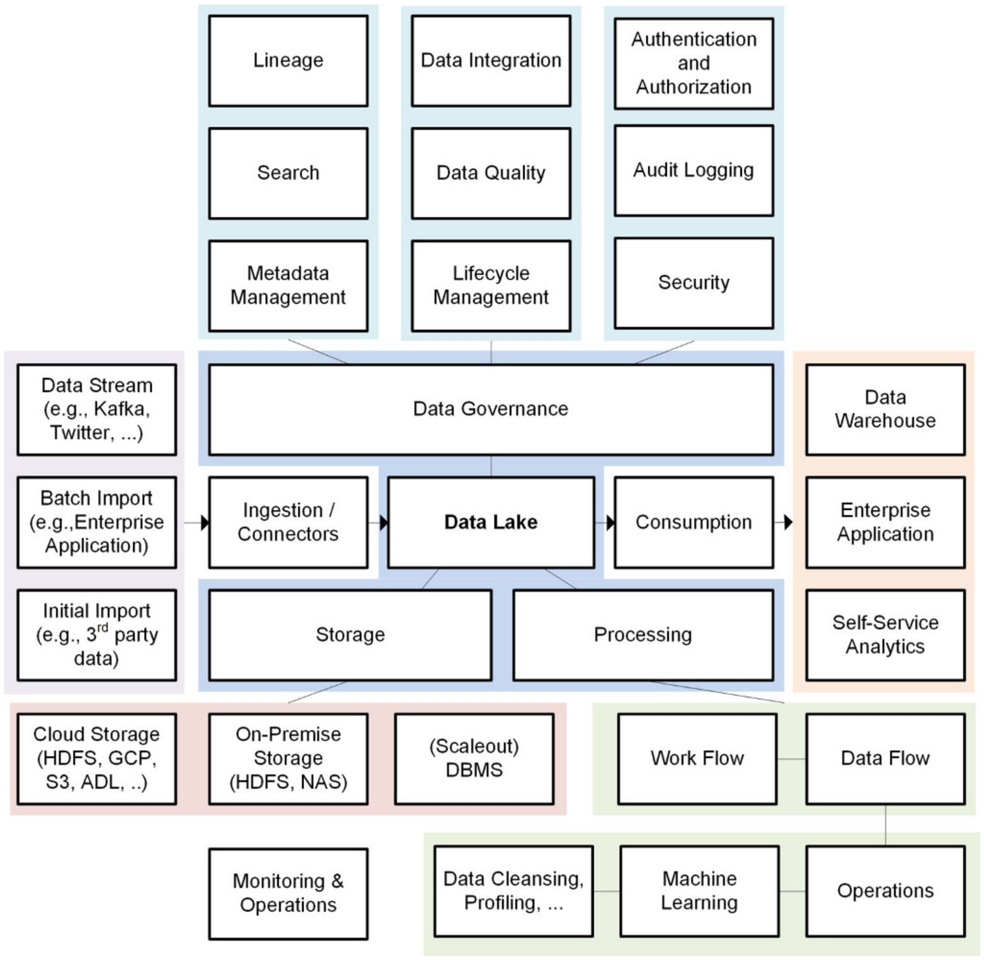
✉ Christian Mathis
christian.mathis@sap.com

[1] SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany

2 Data Lakes: The Idea

A data lake is usually thought of as a central storage location in an organization, enterprise, or institution, to which *any type* of data of *any size* can be copied *at any data rate* using *any import method* (initial, batch, streaming) in its *original* (native, raw) format. For example, this could be sensor streams, user interaction logs, captured social media feeds, image or video libraries, database snapshots from the enterprise's IT systems (ERP, CRM, etc.), among others. The goal is to derive value from the collected data by exploring the data, training statistical models, and transforming the data to be used in services and applications. For example, a manufacturer for home appliances might equip appliances with sensors that monitor customer usage patterns. The captured sensor data would be sent back to the manufacturer where it can be correlated with customer data or production data. On the centralized data set, the manufacturer can then implement predictive maintenance scenarios, extract information for marketing campaigns, or provide added services tailored to the customer. Often, the relevant data to implement such scenarios is stored in different systems: The sensor data might be recorded in some new time-series database operated by the research department, while the customer or production data is stored in an

Fig. 1 Data Lake Topic Map



ERP or CRM system operated by IT. By centralizing data into the lake, it can be correlated with other data without crossing system or organizational boundaries.

The advent of data lakes introduced a new persona: the *data scientist* – a person responsible to analyze the data on the lake [5]. A data scientist has a strong background in statistics and machine learning, is versed in analytical or machine learning software (Spark MLlib, Tensorflow, R, etc.), and usually can program one or more general purpose programming languages. The data scientist works together with other departments inside the organization which either can provide data, like the IT department, or which have a natural interest in analysis results like business analysts, the marketing department, or research and development. The data lake needs to support data scientists not only to analyze the data, but also to share results, for example, by publishing them into the data warehouse or enabling access to enterprise applications. Ideally, the data lake infrastructure and its tools also open up the data on the lake to non-technical users (*self-service analytics*) or enterprise applications. To simplify the following description, we will refer to the data scientist as "the" data lake user (although we are aware, that other personas interact with and build the data lake, like developers, administrators, business users, etc.).

3 Data Lake Technology

Data lake technology is being developed by a growing number of vendors who invest in Hadoop-related components but also develop proprietary software, for example, Google Cloud Platform, Amazon Cloud, Microsoft Azure, Teradata (Kylo), Cloudera, MapR, Hortonworks, SAP, to name a few. Because a comprehensive overview is beyond the scope of this article, we refer to open-source projects maintained by the Apache Software Foundation to highlight technologies that have been developed around data lakes (see Figure 2).

3.1 Storage

The idea of processing data with"any type, any size, and any rate using any import method (initial, batch, streaming)" poses challenges on the storage infrastructure of a data lake implementation. Additional requirements are the

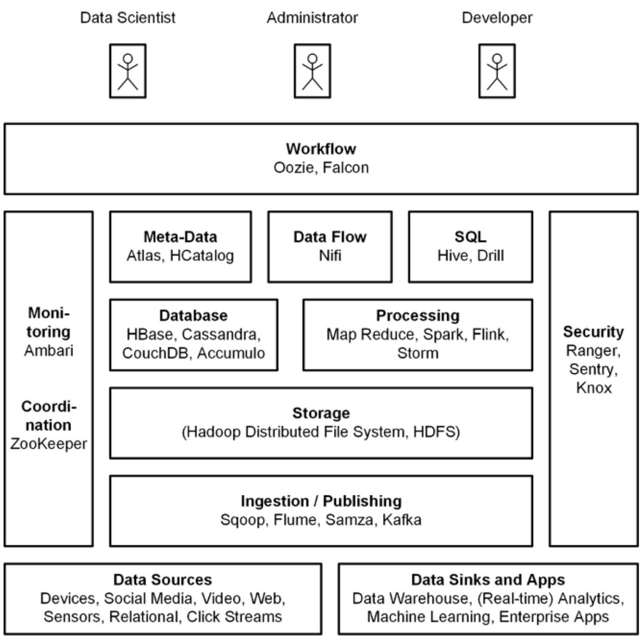

Data Scientist
Administrator
Developer
Workflow
Oozie, Falcon
Moni-toring
Ambari
Coordi-nation
ZooKeeper
Meta-Data
Atlas, HCatalog
Data Flow
Nifi
SQL
Hive, Drill
Database
HBase, Cassandra, CouchDB, Accumulo
Processing
Map Reduce, Spark, Flink, Storm
Storage
(Hadoop Distributed File System, HDFS)
Ingestion / Publishing
Sqoop, Flume, Samza, Kafka
Security
Ranger, Sentry, Knox
Data Sources
Devices, Social Media, Video, Web, Sensors, Relational, Click Streams
Data Sinks and Apps
Data Warehouse, (Real-time) Analytics, Machine Learning, Enterprise Apps

Fig. 2 Hadoop Technologies

usual price per unit of size, on-premise vs. Cloud storage, reliability, security, and scalability. Various storage infrastructures are often cited in the context of data lakes or are even sometimes referred to as "the" data lake: the Hadoop Distributed File System (HDFS) became popular for on-premise data lakes, because it is comparatively cheap and it is tailored to batch-processing workloads run by Hadoop tools (like MapReduce). Although there is a rich variety of Cloud-based Hadoop (and HDFS) offerings, Cloud (PaaS) providers have built additional storage solutions, like Google Cloud Storage, Amazon S3, Azure Data Lake Store [3], or OpenStack Swift. In contrast to HDFS, which implements a file-system-like API, Cloud stores implement an object-store API, which are a more natural fit to RESTful communication protocols. Although for some of these stores the internal architecture is not published, we can speculate that they rely on shared-nothing clusters built on commodity hardware (with standard hard disks) and that they achieve reliability, durability, and scalability by replicating and striping chunks of data across the nodes of the cluster. Bulk data uploads are usually supported via RESTful protocols or other HTTP-based APIs (e.g., WebHDFS). Other storage technologies have been proposed as alternatives to the standard HDFS implementation, mainly to address problems with performance or small/very large files, like MapR-FS, EMC Isilon (network-attached storage with an HDFS API for on-premise), Ceph [4], or IBM (GPFS).

3.2 Data Ingestion

Moving data from external sources into the lake (*data ingestion*) is often implemented with tools that extract data from source systems (pull) or that allow to receive data from external sources and streams (push). For example, Apache Sqoop is a tool that can connect to external database systems to bulk-load (pull) relational data to HDFS. Ingesting streaming data requires tools that can buffer the streams and convert the buffered data to batches before writing to some store like HDFS. Sometimes batching also involves aggregation, conversion, or cleansing directly on the stream. Solutions make use of scalable message queues (Kafka) and data stream engines (Samza, Flume, or Spark streaming) to condense and store streaming data or extracts thereof on the lake.

3.3 Data Profiling and Integration

Because data is stored in its original format on the lake, data scientists face a *data integration* problem (sometimes aggravated by inconsistent and incomplete data) [2]. Before analytical results can be extracted, the data scientist needs to profile the data to "understand" the structure, quality, and schema of the data. In the next step, relevant data can be extracted from the raw data and transformed into a format that allows correlation with other data. This approach is often described as "schema-on-read" [6] – the schema is defined at the point in time when the data scientist wants to work with the data. The data scientist might even define different schemas for the same data set, depending on the attributes she is interested in for the respective question. This approach contrasts the schema-on-write approach used by data warehouses, where ETL processes convert the data into a global schema during the write phase, before analysis tools can work with the data. The goal behind schema-on-read is flexibility (allowing multiple schemas per data set) and deferring the work to create a schema and run data transformations until it becomes necessary.

3.4 Processing

To extract information from the data stored on the lake, the data scientist usually implements the following three stages: 1) data preparation (to overcome the data integration challenge), 2) data analytics (or machine learning), and 3) result provisioning for consumption. Examples for general-purpose processing frameworks that allow to parallelize and scale processing jobs are Hadoop MapReduce, Apache Spark, Flink or Storm that allow (micro-)batch (MapReduce, Spark) or stream-based (Flink, Storm) processing.

There is a large variety of operations a data scientist might need to execute throughout preparation, analysis, and result provisioning: data profiling, parsing, filtering, format conversion, transformation, schema extraction, data cleansing and de-duplication, data masking, data mining, model training and fitting. For result provisioning, she might want to transfer data to an external system, like a business warehouse or an database system hosting self-service analytics or enterprise applications. In the beginning, the data scientist might need to execute these operations in an explorative manner. However, once some way of extracting meaningful information has been developed, a process needs to be established to automatically repeat the steps on new or updated data sets. Furthermore, the data scientist might want to automate repetitive tasks in the explorative phase. Therefore, these operations are often arranged in a *data flow*, where – conceptually – the operations are applied to chunks or streams of the data and intermediate results are stored and forwarded to the next operation. While such data flows can be implemented manually using general-purpose programming languages and various available tools, it often is easier and more flexible to model and configure these steps in a data flow graph using a UI and to execute the graph using a data flow engine (Apache Nifi).

A single data flow can be seen as a unit of work, for example: reading data from a Kafka queue, converting the data into a CSV file and storing the file on HDFS, or running data profiling and schema extraction on some CSV file and storing the result in a metadata repository. To automate and orchestrate these units of work, the data scientist might want to define a work flow (based on data flows and other activities as actions) and run the work flow via some work flow engine or scheduler (for example Apache Oozie or Falcon).

3.5 Encoding and Database Systems

Storing the raw data in a (virtual) file system or object store simplifies data ingestion, but poses challenges on data processing. For example, whenever a data scientist or an automated data flow would like to process a CSV file, the file needs to be parsed and the schema needs to be extracted. While this might be reasonable when the data is accessed only once, but it imposes unnecessary costs, when the data is accessed multiple times. Various data formats have been proposed to solve this problem, like Parquet, ORC, and AVRO. These formats can encode relational data (and nested data sets) into files using an internal columnar layout with compression. Such files also carry metadata (the schema) as well. Processing tools like Spark and Hive can implement specialized modules to read data from encoded files to only fetch the relevant portion (column fragments) or to execute predicates and filters while reading.

Encoded files provide for richer access methods to the data (e.g., SQL on files via Spark SQL). When even richer analytic capabilities or higher performance is required, it is often beneficial to load the data into a scalable (No)SQL database system, for example, HBase, Cassandra, or Accumulo. Some systems also support non-relational data types, like CouchDB and MongoDB (JSON), Neo4J (graphs), etc. These systems can be seen as an integral part of the data lake infrastructure and sometimes they also manage the primary data persistence. This is possible, because these database systems have their own fail-save, distributed and scalable data store, which can provide the same guarantees as an external file system or object store. An alternative to full-fledged distributed database systems are pure SQL engines like Hive or Drill. Those systems provide SQL as a higher abstraction level to data stored in key-value stores or distributed file systems.

Keeping data in encoded format instead of the raw format or even in a database system usually results in higher cost per unit of size. Therefore, the data scientist is challenged with the question, when to move data between raw files, encoded files, and database systems. Ideally, the data lake software (especially the data governance and monitoring components, see below) would support the data scientist to make the decision and data/work flows can be used to automate the tasks.

3.6 Connectors

Data lakes implement data movement: data is pulled from external systems (or pushed into the lake via public APIs), data is moved inside the lake, for example, from an object store into a NoSQL DBMS, and data is published or served to external systems, like a data warehouse or enterprise application. Many different systems can be involved in the data movements, which is why software in a data lake (the message queue, the data flow engine, or even a DBMS) usually provide a rich connector library.

3.7 Lambda and Kappa Architecture

So far, we have sketched the data scientist's working set rather as an immutable data snapshot of external systems. However, the need to derive real-time insights from data generated by sensors, social media, or other streams implies dynamic data processing. The data scientist needs to work with data streams from which results with minimal delay have to be generated. Buffering the data on storage and applying batch-processing techniques to the buffered chunks usually imposes a too large delay for near real-time insights. On the other hand, data scientist also want to store data from streams permanently for later analyses. The Lambda architecture has been proposed as a solution to the problem [8]. The data stream is both fed into a data stream processing system (like Apache Storm) and copied on a scalable file system, where it can be exposed to analyses (for example through Apache Spark). Results from both, the *batch* and the *speed* layer are published in a database system, from which they can be consumed by applications. The Lambda architecture has been criticized, because it requires the data scientist to write data processing logic twice: the speed and batch layer do not share the same "language". The Kappa architecture proposes to solely make use of a stream processing system (single layer) that can – in addition to on-stream processing – buffer the data long enough to process historic data using the same domain logic as the live stream [7].

3.8 Data Governance

Data lakes have often been criticized to quickly turn into "data swamps" (i. e., ungoverned data lakes). The risk stems from the potential lack of control on data (quality, structure, sources, intermediate results), processing pipelines, applications, and users. The lack of control can even have legal implications for an organization, for example, in the face of the new General Data Protection Regulations (GDPR). As a consequence, data lakes have to be integrated in *data governance* processes and methods by providing the necessary infrastructure for metadata management (data lineage, metadata search), data quality (data cleansing), data lifecycle management, and information security (authorization and audit logging). The Apache Atlas project aims to introduce data governance capabilities into Hadoop.

4 Summary and Outlook

Moving data into a central storage location to simplify analysis and to generate value from the data is main idea behind data lakes. However, to find new information from data on the lake requires techniques, tools, and processes which help data scientists, developers, and other stakeholders inside an organization to efficiently work with the data. This includes tools to accomplish data ingestion, discovery, profiling, cleansing, meta-data management, security, auditing, among others. Data lakes are still an evolving concept. Therefore, we expect further active development and research in this area.

Acknowledgements I would like to thank Christian Sengstock and Martin Hartig for feedback and discussions while writing this article.

References

1. Dixon J (2010) Pentaho, hadoop, and data lakes. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/
2. Dong XL, Srivastava D (2015) Big Data Integration. Morgan and Claypool Publishers, San Rafael, CA
3. Ramakrishnan R et al (2017) Azure data lake store: a hyperscale distributed file service for big data analytics. Proc ACM SIGMOD Int Conf Manag Data. https://doi.org/10.1145/3035918.3056100
4. Maltzahn C, Molina-Estolano E, Khurana A, Nelson AJ, Brandt SA, Weil S (2010) Ceph as a scalable alternative to the Hadoop distributed file system. login 35(4):38–49
5. Cohen J, Dolan B, Dunlap M, Hellerstein JM, Welton C (2009) MAD skills: new analysis practices for big data. Proc VLDB Endow 2009:1481–1492
6. Xin RS, Rosen J, Zahira M, Franklin MJ, Shenker S, Stoica I (2013) Shark: SQL and rich analytics at scale. Proc ACM SIGMOD Int Conf Manag Data 2013:13–24
7. Kreps J (2014) Questioning the lambda architecture. http://milinda.pathirage.org/kappa-architecture.com/. Accessed: 30. Sept. 2017
8. Marz N (2011) How to beat the CAP theorem. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html. Accessed: 30. Sept. 2017