Building Data Lakes Successfully

Published: 11 July 2019 **ID:** G00378068

Analyst(s): Sumit Pal

Data lakes provide a flexible data store that helps organizations address data and analytic use cases, but building a data lake is challenging. This research helps data and analytics technical professionals circumvent challenges and align use cases with technologies and architectural best practices.

Key Findings

- Organizations are opting to build data lakes and leverage their data assets without thorough consideration to appropriate business use cases. This is often the main reason for data lakes turning into data swamps.
- Organizations often underestimate the technical complexities to develop and maintain a data lake and the associated expertise to overcome those complexities, resulting in a brittle environment that takes longer to solidify and results in higher costs.
- Organizations are thinking of using data lakes as a replacement for data warehouses. Data lakes are complementary to data warehouses; they do not replace them.
- Organizations that are successfully building and leveraging data lakes are using them for use cases such as offloading ETL, data discovery/exploration by data scientists, stream processing, and storing and enriching unstructured and multistructured datasets.

Recommendations

Technical professionals responsible for data management strategies should consider the following:

- “Begin with the end in mind.” Improve the chances of success by building data lakes for the specific requirements of certain business groups, sets of users or analytics use cases, rather than taking a “big bang,” enterprisewide approach.
- Learn lessons from data warehouse experiences. Do not build a data lake hoping that the enterprise will figure out how to use it.

- Incorporate effective governance early in the process to avoid potential pitfalls of data lakes. Pitfalls include poor accessibility, poor metadata management, poor data quality and lack of lineage and data security.
- Enable data democratization by establishing a metadata layer to enable governed data discovery, which is imperative to address data accessibility and for insights into context and analytics that will be delivered to the enterprise. It is essential to catalog data assets.

Table of Contents

List of Tables

List of Figures

Analysis

Organizations today are bursting at their seams with a data deluge. Organizations of all sizes are attempting to gain control over the data that resides in multiple formats and locations across the enterprise. Organizations are realizing that traditional technologies can't meet all of their new business needs, causing many organizations to invest in a scale-out data lake architectural pattern.

Organizations are investing massive amounts of effort to:

- Build and architect data lakes in response to today's data challenges.
- Maintain, organize and manage the varied data resources.

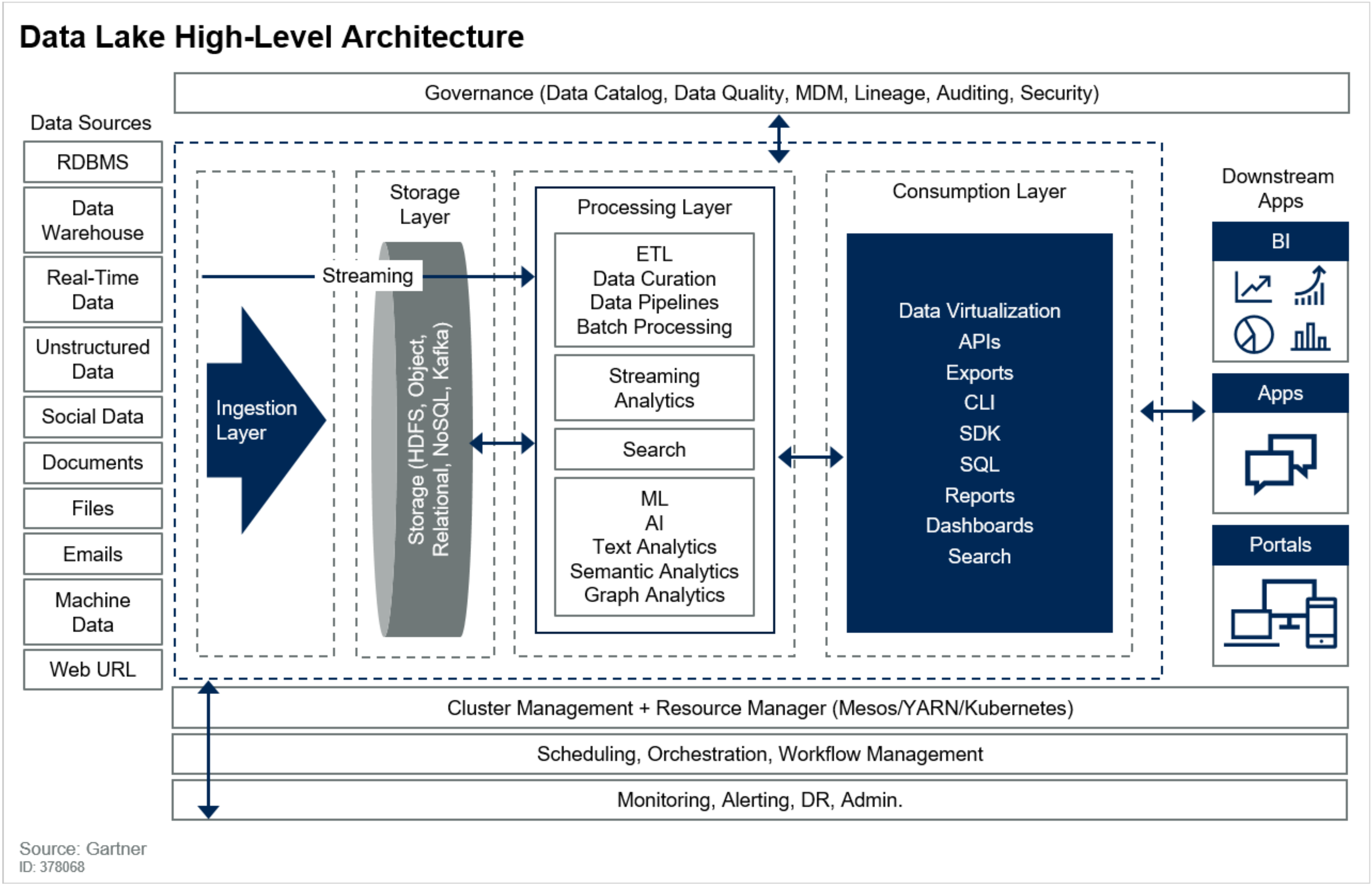
Data lakes are built on distributed architecture at massive scale, with the ability to store and process unstructured or multistructured datasets and built often in the cloud.

Data lakes need multiple components, frameworks and products from different vendors to work in tandem. Hence, enterprises need to architect data lakes with upfront planning, and correct architecture, because these potentially critical business resources could be underutilized with a lack of governance, inability to find the data, lack of security, and redundant and poor-quality data. The data lake stack is complex and requires critical decisions around data ingestion, storage, processing, consumption and governance.

Architecture

Data lakes are complex, and building them successfully involves integrating different technologies because there is no one end-to-end product in the market to implement enterprisewide data lakes. A high-level data lake architecture, along with key functional capabilities and major components needed to build a data lake, is shown in Figure 1.

Figure 1. Data Lake High-Level Architecture



When architecting a data lake, some of the key capabilities to keep in mind are listed in Table 1.

Table 1. Key Capabilities of a Data Lake

Capability	Description
Data Ingest	▪ Multiple ways of ingesting data into a data lake ▪ Make new data sources, both static and streaming, available for ingestion with defining schema during ingest
Data Storage	▪ Accommodate a variety of data storage platforms based on volume and variety of data types with different read and write patterns across multiple hardware types
Data Processing — Batch/ Streaming/Analytic Workloads	▪ Provide multiple ways of processing the variety of data types ingested and stored in the data lake for analytics and building data-driven applications
Data Quality Metrics and Continuous Measurement	▪ Multistage data cleaning, refinement and enrichment ▪ Automated data processing ▪ Continuous data profiling
Automated and Collaborative Data Cataloging Classification for Governance and Data Consumption Catalog Service	▪ Catalog and classify available data ▪ Search, discover and subscribe to data ▪ Data source registration, and automated data discovery and cataloging
Security	▪ Access controls to a wide category of users in the enterprise ▪ Data encryption, data redaction, data obfuscation and data masking
Operationalization and End-to-End Automation	▪ Design and configure workflows/scheduling/orchestration/automated scaling/self-healing
Data as a Service and Data Provisioning for Consumption	▪ Publish data and analytical services for consumption ▪ Provision refined, clean, trusted data for downstream consumption
Data Life Cycle Management	▪ Process of controlling, versioning and managing storage of data in the organization as it ages and progresses through its business and technical life cycle

Source: Gartner (July 2019)

Most organizations need to answer some key architectural and deployment questions before building a data lake. Should they:

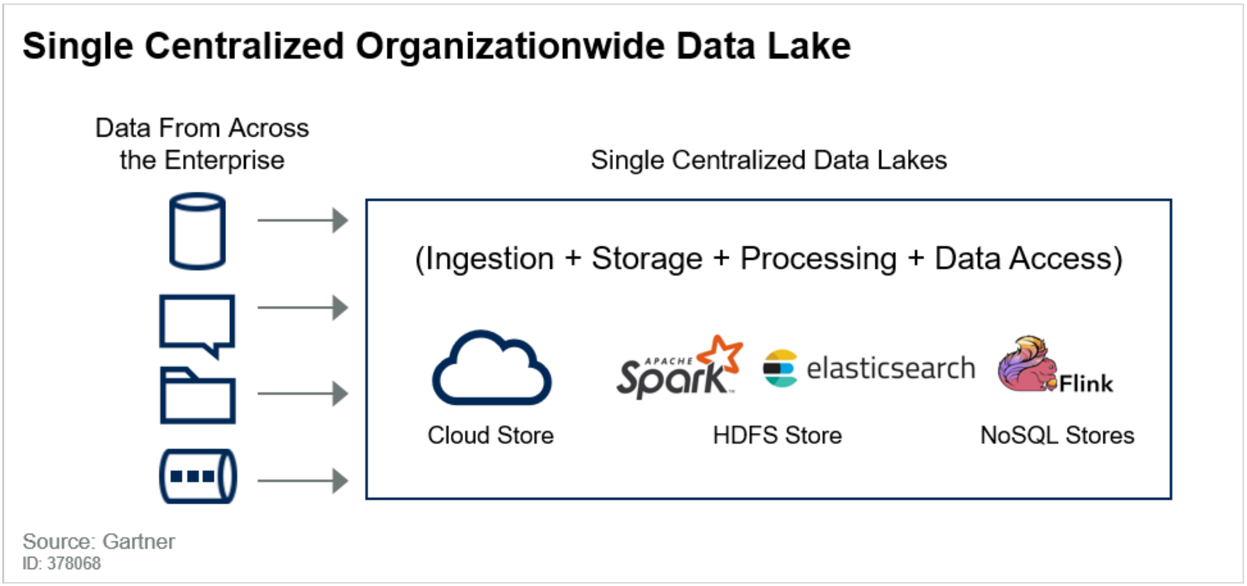
- Build a single centralized data lake for the entire organization?

- Build multiple data lakes that are localized to the data centers of the organization?
- Build an on-premises or in-cloud-based data lake or a hybrid or multicloud-based data lake?

These approaches to building a data lake are discussed below, along with their advantages and disadvantages.

Figure 2 shows a single centralized enterprisewide data lake.

Figure 2. Single Centralized Organizationwide Data Lake


Single Centralized Organizationwide Data Lake
Data From Across the Enterprise
Single Centralized Data Lakes
(Ingestion + Storage + Processing + Data Access)
Cloud Store
HDFS Store
NoSQL Stores
Apache Spark
elasticsearch
Flink
Source: Gartner
ID: 378068

Advantages

- A single centralized data lake is easier to manage and govern.
- All data is in one single location, which is easier for data access and for dealing with bandwidth constraints and provides a single source of truth.
- Costs can be controlled.

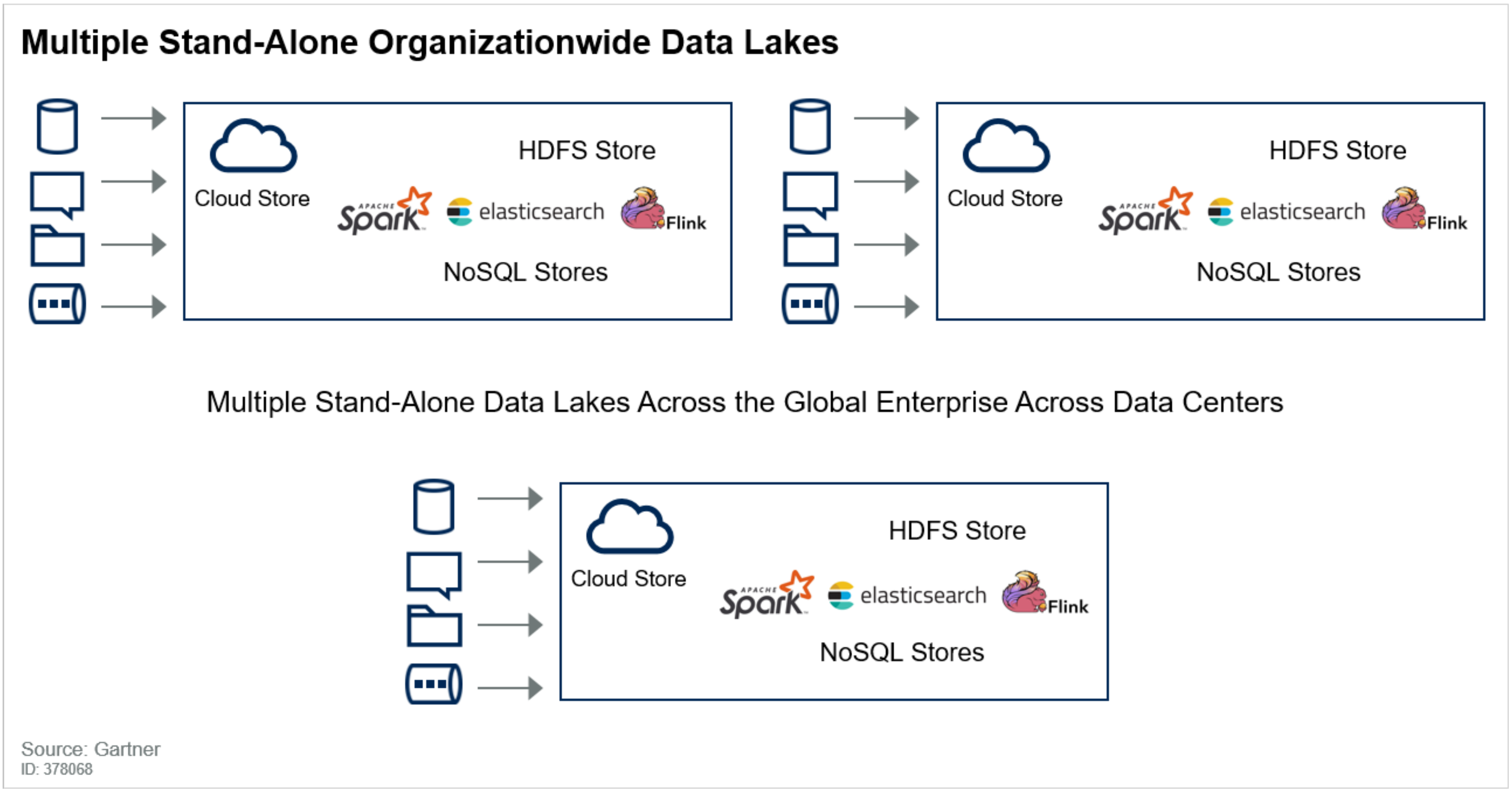
Disadvantages

This approach, though simple, may not be practical from a global enterprise's perspective. Some of the problems associated with such an architectural model include:

- Conflicting legislation/sovereignty on data residency, movement and access can hamper usability.
- Very high network capacity and speed is required to move everything to one place.
- This can be a single point of failure and a security risk.

An alternative approach is to build multiple stand-alone data lakes within the enterprise, with each localized to a data center. Figure 3 shows the architecture of such a data lake.

Figure 3. Multiple Stand-Alone Organizationwide Data Lakes



Advantages

- Data that is too big to move across the network into a single location can be handled within a local data lake.
- Legislation and regulations can be accommodated across geographical jurisdictions.

Disadvantages

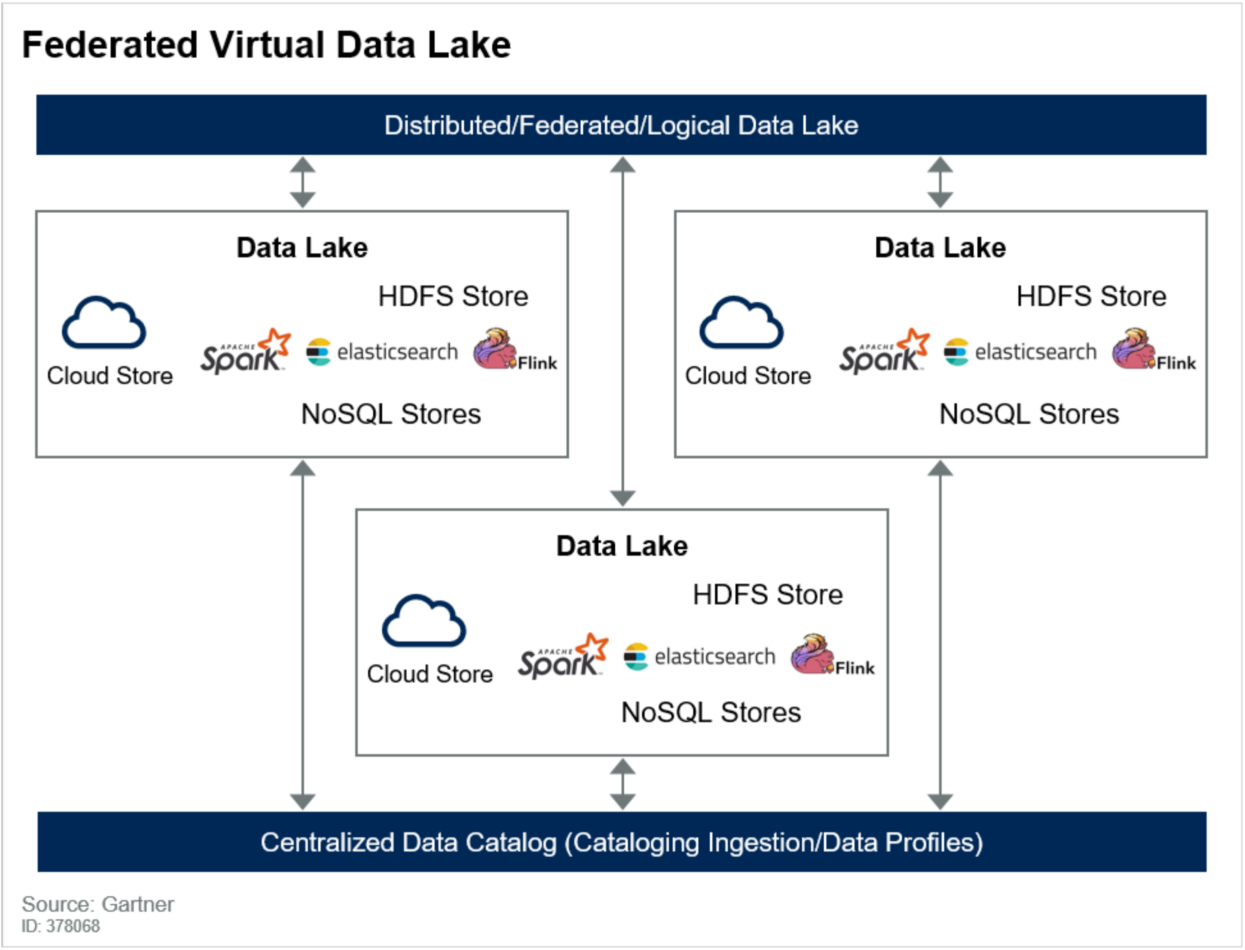
- This approach creates silos of data lakes.
- Inconsistencies of data governance policies and tools can occur.
- Synchronization of common data across multiple data lakes can be very difficult.
- Multiple heterogeneous technologies may exist across different data lakes.
- Costs can be high due to network bandwidth requirements for data movement.
- Data access can be challenging.

Another architectural approach gaining traction and usage across large organization is to build virtual/federated/logical data lakes.

The idea is to present siloed data lakes from across the organization as a single data lake, while managing the architectural details and capabilities separately for each individual data lake. In order to accomplish this sort of a data lake architecture, organizations need support from data federation tools as well as enterprisewide catalogs that can span across data lakes, whether they are deployed on-premises or in the cloud. The catalog becomes the interface for searching and finding data within the data lake, and the data federation maps the request to whichever data lake is appropriate. Figure 4 shows a high-level architecture of a federated data lake across the organization.

Data lakes can also be part of a large logical data warehouse (LDW). See “Adopt the Logical Data Warehouse Architecture to Meet Your Modern Analytical Needs.”

Figure 4. Federated Virtual Data Lake



Questions to Ask

Data lake architects and data engineers should ask the following questions when architecting a data lake:

- What are the workload processing requirements?

  The choice of components for handling different types of workloads varies. Batch workloads need tools like Apache's Sqoop, Spark and Hive for ingestion and data processing, while streaming workloads will need tools like Apache's NiFi, Kafka and Flink for ingestion and data processing.

- What are the nonfunctional requirements (NFRs) for the data lake?

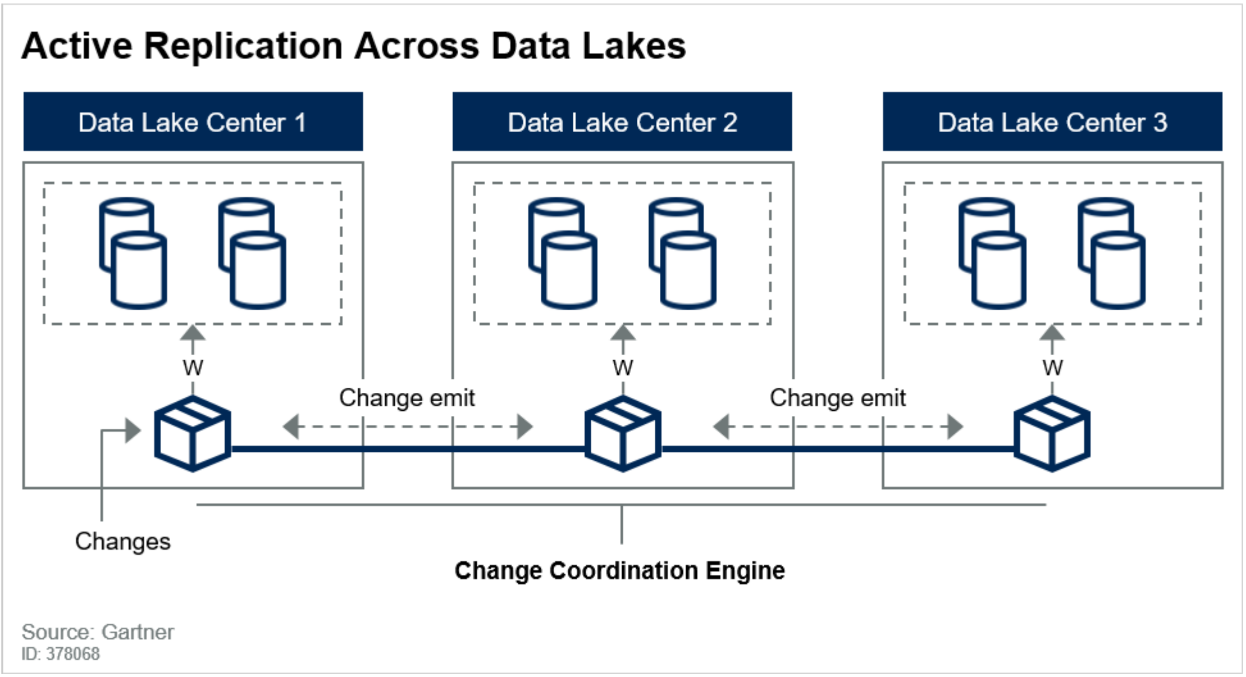
  Understanding the NFRs for a data lake is extremely important in determining the overall architecture, selection of tools, frameworks, hardware and capacity planning for the data lake. Each organization and business should answer these questions, which are specific to their business, data storage and processing needs:

- What are the scaling requirements across compute, storage, networking or I/O?
- What are the availability and recoverability requirements?
- What kind of compute and storage fault tolerance is desired?
- What are the performance metrics across workloads — SLAs/throughputs and latency?
- What are the disaster recovery requirements?
- What are the data replication requirements?
- What are the storage requirements based on throughput, latency and volume?
- What are the read/write access and patterns?
- What is the impact of data loss for the organization?

Best Practices

- For a highly available data lake, ensure each component within the data lake is highly available. For example, if you are using Kerberos distribution center, make sure it is set up for high availability (HA), or if you are using Hive metastore, make sure it is set up for HA (standby metastore with replication). If you are using Apache Hadoop-based components in a data lake, then use Hadoop Distributed File System (HDFS) 2.0 to make Name Node Highly Available and scale using federated NameNode architecture.
- Avoid tight coupling of components, decouple components with a layer of abstraction, and program to interfaces.
- For data lakes spanning across data centers, ensure active-active replication is set up, as shown in Figure 5.

Figure 5. Active-Active Replication Across Data Lakes



Ingestion

Before the data lake becomes useful for data analysis, the data lake needs to connect to various data sources and acquire data from them. The ingestion layer in the data lake enables the functionalities to achieve this. Data ingestion lays the foundation for data extraction from source data systems and orchestration of different ingestion strategies in a data lake.

Figure 6 shows the different ways data can be delivered into a data lake. Table 2 lists some of the major considerations before setting up data ingestion.

Figure 6. Data Delivery Techniques

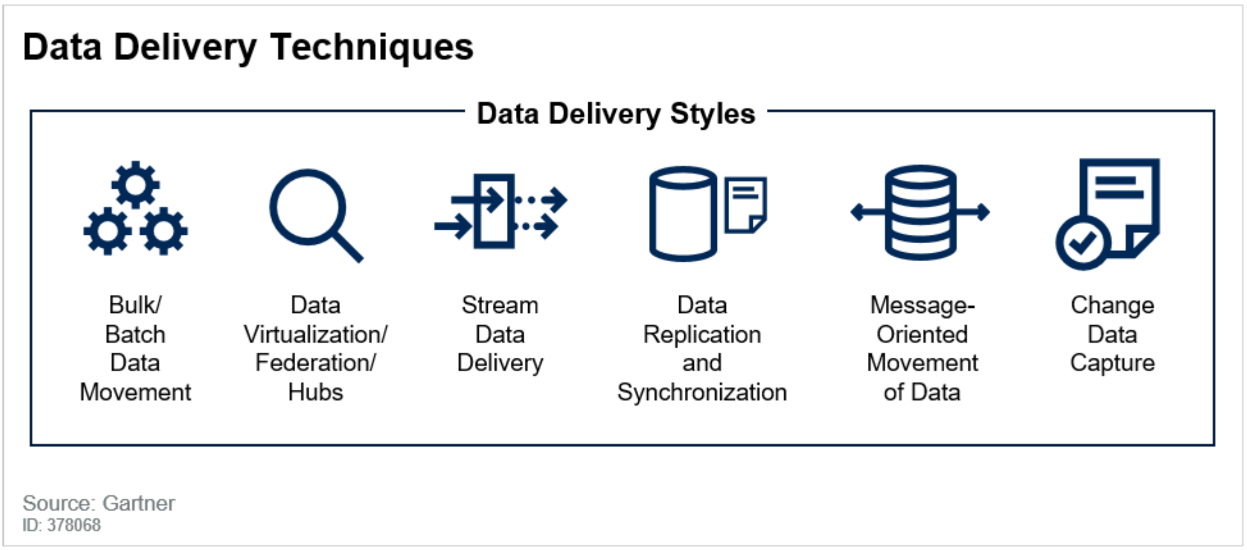

Data Delivery Techniques
Data Delivery Styles
Bulk/ Batch Data Movement
Data Virtualization/ Federation/ Hubs
Stream Data Delivery
Data Replication and Synchronization
Message-Oriented Movement of Data
Change Data Capture
Source: Gartner
ID: 378068

Table 2. Data Ingestion Requirements

Category	Type	Description
Technology	Schema	Identify the source and the associated schema. Have tools in place to identify data drift and schema drift.
Technology	Security	Classify the data and perform PII, data masking, data tokenization and data encryption.
Technology	Governance	Capture metadata at ingest and also have hooks in place to capture data lineage.
Organization	Owner	Define the owner of the data at source as well as the data lake.
Organization	Steward	Define the privacy, regulatory and compliance requirements of the ingested data.
Process	Monitoring	Ensure proper monitoring and alerting is in place to detect failures and problems during ingestion.
Process	Orchestration	Ensure proper workflows and scheduling is in place for timely data ingestion.

Source: Gartner (July 2019)

Data catalogs can help define some of the above requirements. Building a data ingestion strategy requires clear understanding of source systems and SLAs expected out of the ingestion framework. Ingestion mode can be batch, real time or a change data capture (CDC) — depending on the data transfer granularity and cadence:

- Batch-Based — Use batch-based ingestion techniques when throughput is more critical for your business decisions. Use batch-based techniques when SLAs are in the hours or days range. When using batch-based ingestion, pay particular attention to the batch size or batch granularity and the batch cadence. Often, batch size is an important criterion for tuning and performance optimization of batch ingestion jobs.
- Real Time — Use real-time ingestion techniques when latency is more critical for your business decisions than throughput. Organizations typically use real-time ingestion when SLAs are in the minutes or less range. Handling, backpressure, data loss and ability to autoscale based on incoming data flow is important when designing real-time data ingests.
- Change Data Capture — CDC has a few fundamental advantages over batch-based replication techniques:
  - Enables faster and accurate decisions based on more recent data
  - Minimizes disruptions to production workloads
  - Reduces costs and network bandwidth consumption because only incremental changes are transmitted

Questions to Ask

Below are some of the questions data lake architects should ask before architecting and building the data lake and choosing the components, frameworks and products for ingesting data into the data lake:

- What style of ingestion is required?

  Organizations should understand the type of ingestion that is required for their data lakes. Whether it is batch-based ingestion, real-time data ingestion or incremental, CDC-based ingestion required to populate the data lake. It is possible that an organization requires all these kinds of ingestion capabilities. Tools and framework selection should be based on the ingestion types because not all ingestion tools support all types of ingestion capabilities.
- What data formats and type of data should be supported for ingestion?

  Understanding the type of data that is being ingested is important because not all tools can support all types of data ingestion. Some tools, for example, cannot support ingested binary data like video and audio, and some tools can work only with structured datasets.
- What is the data velocity in terms of throughput and volume?

  Setup, configuration, capacity planning of the ingestion framework, and cluster and scalability of the ingestion tool should be guided by the above requirements.
- What is the ingestion frequency?

  Orchestration and workflow handling of the ingestion should be tied to the frequency. Ingestion processes should be deployed with the right scale to ensure each ingestion batch is properly sized for throughput.

- What is the source location of the source data systems and location of the data lake?

  Determine whether the data ingest will happen on-premises, from on-premises to cloud, from cloud to cloud, or from cloud to on-premises. Understanding the origin and destination for the data to be ingested is important for choosing the right toolsets, designing the right network bandwidth, and architecting failure-handling mechanisms. This is an important question to answer in order to select the right framework, tools and bandwidth requirements for data ingestion.

- How will you handle data ingestion failures and interruptions? Failures and errors can be different types:
  - Wrong data
  - Wrong schema
  - Hardware failure/software failure
  - How to handle exception records
  - How to detect file integrity (Is the file complete? Was it tampered with in transit?)
  - Data integrity (Is the data as expected? Are all fields present? Is there any data drift?)

Each organization should define policies and rules to handle these errors and failures. These can be handled at the framework level or can be hooked into the orchestration engine to schedule data ingestion. The Details section of this research dealing with ingestion outlines an architectural approach to handle these situations:

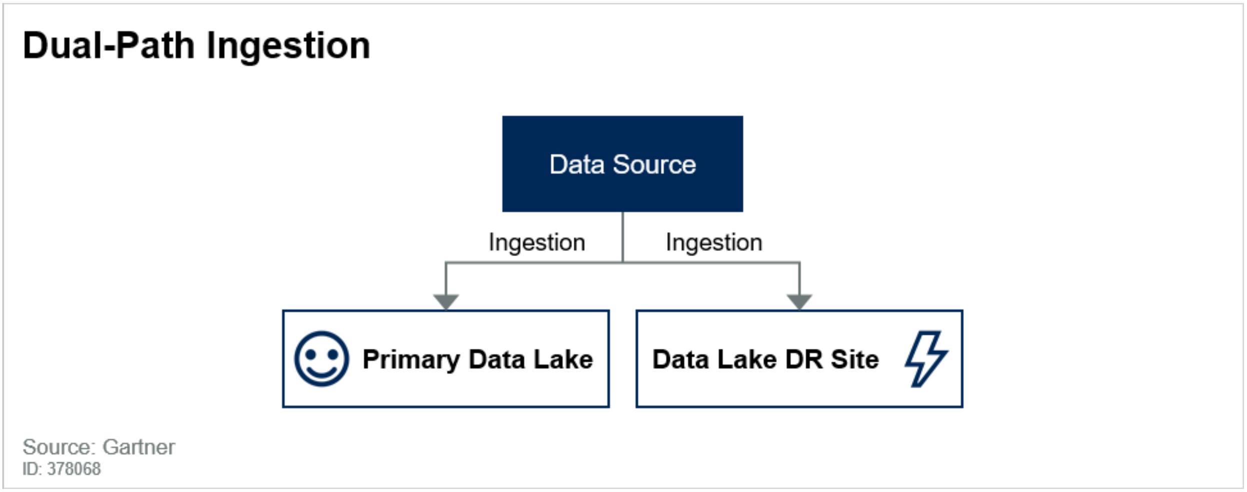
- Scheduling — How often does the data arrive? Was the data received on time? Does data delivery require acknowledgment?
- Schema management — Is there an associated schema with the data? Can the schema be inferred or else how is the schema communicated and how is it verified? How does the ingestion framework handle schema evolution?

Best Practices

- The ingestion framework should be able to handle additions and upgrades to existing and new data sources across a variety of data types and data velocity, such as streaming video and audio.
- For maintenance, it should be easy to add new ingestion jobs and update, modify, debug, troubleshoot and provide traceability for an already running job.
- The ingestion framework must be scalable on-demand based on varying load conditions. It should be able to scale on volume and velocity to minimize latency and to improve throughput to maintain the SLA.
- The ingestion framework must be capable of being fault-tolerant with fail-safety (recovery) as well as failover (resiliency) support.

- Ideally, the framework should support multithread and multievent execution.
- It should be able to quickly transform the acquired data structure into the target data formats as needed by the downstream processing layers.
- It should be capable of merging small files into larger files to reduce metadata management problems, as well as to enable faster downstream data processing.
- Avoid data lake indigestion by fixing data quality at ingestion time and by continuously monitoring data quality as data ingests into the data lake.
- Architect your ingestion framework to handle changes to incoming data — for example, fields getting reordered. The framework should be able to detect these kind of changes and self-correct or issue alerts and notifications.
- You should have a clear strategy to handle bad and corrupt data during ingestion.
- Integrate data ingestion with the data catalog to ensure data quality, data discovery and data classification at ingest time.
- Ensure automation of ingestion workflow and orchestration and scheduling of ingest workloads.
- Handle security at the source and set up handling PII on data that lands within the raw zone.
- Ensure the ingestion framework can throttle the source or apply backpressure when the data lake components are not able to keep up with the data flow.
- When architecting the ingestion layer, take into account the problems associated with distributed computing, like networks, which can be unreliable from data loss and data corruption perspectives.
- The framework should be integrated with the data catalog and should be able to detect and gracefully handle data drift and schema drift.
- If economics permit, ensure dual-path ingestion, as shown in Figure 7.

Figure 7. Dual-Path Ingestion



Storage

Following data ingestion, the data needs to be stored. Data storage within the data lake is the most critical component. Data storage choices and architecture determines the performance, scalability and accessibility of the data in a data lake. With myriad complex data types, growing data volumes and data variety, there is no single storage platform or choice that can suffice all storage requirements across different selection criteria.

Data lakes typically will have storage across any of these types of data stores:

- NoSQL data stores
- Relational data stores
- Streaming message/event stores
- New SQL data stores
- In-memory data stores
- Distributed File Systems
- Object data stores

For detailed coverage of these data stores, see the following Gartner research:

- "Assessing the Optimal Data Stores for Modern Architectures"
- "An Introduction to Graph Data Stores and Applicable Use Cases"

Depending on the consumption use cases, either the existing storage can be used or data moved to another form of storage like NoSQL data stores based on use cases and read and write patterns. Most organizations use HDFS — on-premises and cloud, or object store on cloud, for storing the data post ingestion, as well as for postprocessing. Object stores such as Amazon Simple Storage Service (S3) and Google Cloud Storage are overtaking Hadoop Distributed File System (HDFS) deployments rapidly. Data lake architects must understand the nuances of replacing a file system with an object store. Object stores have a different architecture than file systems. Patterns developed around HDFS primitives may not translate one to one when moving into an object store. Here are some of things to be careful about when working with object stores:

- An object store has a higher I/O variance than HDFS. When applications or data stores need consistent I/O requirements, like Apache HBase or another NoSQL database, object stores are a bad choice, although products like HBase have recently been ported to S3.
- Object stores do not support file appends and file truncation. Objects are immutable, and once uploaded, they cannot be changed.
- Object stores are not Portable Operating System Interface (POSIX)-compliant. Over the last couple of releases, HDFS has incorporated many POSIX-like features.

- Object stores do not expose all file system information because it abstracts the entire underlying storage management layer.
- Object storage may have greater request latency than HDFS.

Some advantages of object stores:

- **Lower costs:** Object stores use a pay-for-what-is-used model instead of buying hardware upfront and estimating storage requirements.
- **Decoupled compute and storage:** Data in an object store can be assessed from any other cluster, which makes it easier to tear down and set up new clusters with no data movement.
- **Interoperability:** Because of the decoupling, object stores enable interoperability between different processing engines and data stores.
- **High availability:** Object stores are architected to be highly available and globally replicated. They are not vulnerable to NameNode as a single point of failure.
- **Management overhead:** Object stores require very minimal maintenance regarding upgrades or rollbacks to previous versions of the file system and other routine administrative tasks.

Some the object store issues include:

- Directory listing is slow
- Eventually consistent
- Bucket renaming is not atomic because they are metadata pointers, not true directories
- Bucket metadata and permissions will not work as they do in HDFS

Choose HDFS for these workload types:

- Workloads that require low performance and fast metadata access or small file read.
- Workloads requiring POSIX-like features like strictly atomic directory renames, fine-grained HDFS permissions, or HDFS symlinks.

Questions to Ask

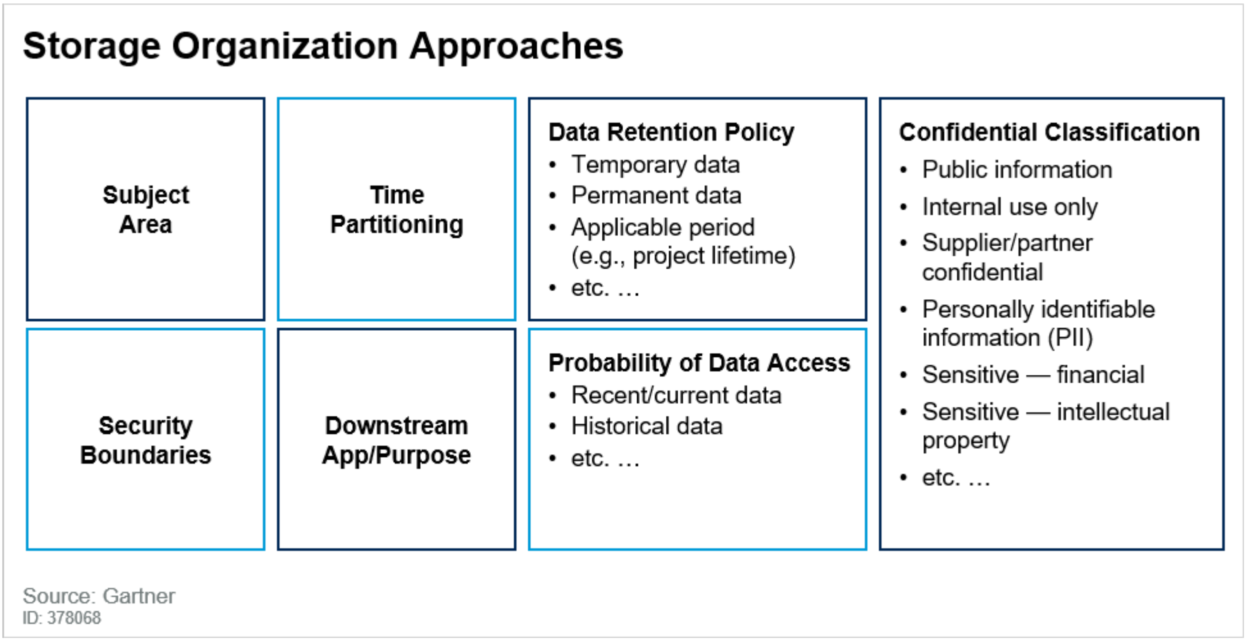
Below are some of the questions data architects should ask before architecting and building the data lake and choosing the components, frameworks and products for storing data in the lake:

- What are the datatype requirements for storage in terms of the type of data being stored?
- What are the velocity and throughput requirements of storage?
- What are the data storage volume requirements?
- What type of storage should be chosen — distributed files systems or object stores?
- What are the different storage zones?

- What are the enterprisewide data format, compression and data splitability requirements? Do you have an evaluation framework to choose the data format? Some questions to answer would be:
  - Is the data storage going to be row, column-based or both?
  - Does the data format need to be human-readable?
  - Does the data to be stored have an associated schema?
  - Can the schema be versioned and updated, and is it subject to change and evolution?
  - How does the data format affect the size of the data stored?
  - Is the data format splitable across the different nodes in a distributed system?
  - What compression capabilities should be supported?
  - What are the ways in which it can be integrated to other tools? How is it interoperable with other tools in the ecosystem?
  - Is the data format optimized for read/write-based patterns?
  - What are the read and write performance requirements? Is it sequential read/random reads or sequential write/random write?
- What are the hardware requirements on which the data is stored?
- What are the data partitioning requirements?
- How is the data storage organization to be done?

Some storage organization approaches are shown in Figure 8.

Figure 8. Storage Organization Approaches



Best Practices

As organizations are moving to the cloud, there is an increased move to adopting cloud-based object stores as the primary data storage for data lakes. For organizations that have decided to stick with HDFS, it best to move to HDFS 3.0, which uses erasure coding that results in better data storage savings and removes the need to replicate data three times. Also, evaluate Apache Ozone for uniform access to data stores irrespective of whether the underlying storage is HDFS or object store.

It is important to evaluate the file sizes and where they originate from. Contrary as it sounds, big data can be made up of a lot of small files, especially those originating from event-based streams from IoT devices, servers or applications, which typically arrive in kilobit (kb)-sized files. This can easily add up to hundreds of thousands of files being ingested in the data lake. Writing small files to an object storage is easy and fast; however, querying these small-sized files either using a SQL engine or for data processing engines like Spark drastically reduces performance and throughput.

Also, metadata collection — like file size, date of creation and so on — on small files is tedious and can reduce performance. Small files have the overhead of opening the file, reading metadata and closing it. Many files lead to noncontiguous disk seeks, which object storage is not optimized for. The remedy for the small-file problem is to merge them into larger files on a regular basis — called small-file compaction. Defining these compaction windows is also something that must be wisely tuned and scheduled. Compacting too often will be wasteful because files will still be pretty small and performance improvements will be very small. Files should be deleted postcompaction to save space and storage costs.

In distributed file systems, there is a lot of internode communication. MTU (maximum transmission unit) is an important metric that indicates the packet size that can be sent over TCP/IP. The default size for MTU is 1,500, which is small for most big data workloads. It is advisable to increase the MTU value to 6,000 to 9,000 to improve performance.

Leverage the performance and throughput improvements with data compaction, compression, partitioning and right file formats for data processing and query latency.

Data Processing

Other than the use case where data lakes are used just for data storage, data lakes provide value only when the data that has been ingested and stored can be processed to build data-driven applications. The versatility of data lakes is evident by the different data processing workloads it can support. For example:

- Batch
- Stream Workloads
- Machine Learning Workloads
- Graph Processing Workloads
- BI Workloads — SQL Query Processing

However, the challenge with data processing in a data lake is in the variety of available data processing engines. Each of the different data processing engines have different goals and are best-suited for different use cases. The criteria for selecting processing engines depends on the internal architecture of the engines, the best use-case fit for these engines, business SLAs, team expertise and other components used in the data lake.

Questions to Ask

When looking to architect the data processing layer of the data lake, try to answer the following questions before choosing a data processing platform or engine:

- How does the engine perform data processing and execute the plans?

  Directed acyclic graphs (DAGs) and execution plans express the underlying tasks and dependencies that a distributed engine performs. DAGs can be managed external to the execution engine — which is the simplest architecture (for example, Hadoop MapReduce) — or tightly coupled with the execution engine. Most processing engines of today no longer use the external model, though. MapReduce is used heavily in some of the core products in the Hadoop ecosystem, but is getting overshadowed by more modular and efficient engines like Apache Spark.
- How does the engine's system architecture distribute the processing for multiple users and handle resource allocation for concurrency and computation?

- How fast, both in terms of throughput and latency, is the engine at executing different types of use cases?
- How does the processing engine manage task and node failures?

  Different processing engines behave differently when they encounter either hardware or software failures. Analyze and understand these before choosing a processing engine for your use case.
- How is the processing engine architected — a large number of records or in single groups of events?
- What type of workloads does your organization plan to operate in the data lake? Are they batch, streaming, a combination of batch and streaming, machine learning, SQL access, or BI workloads.

Best Practices

Some of the recommended best practices when choosing data processing frameworks and engines for a data lake are given in this section.

Where and when to use MapReduce: MapReduce is a low-level framework that puts the onus on developers for handling minute details of operation and data flow, resulting in a lot of setup and boilerplate code. Some operations, such as file compaction and distributed file-copy, nicely fit into the MapReduce paradigm. MapReduce should be used by experienced developers who are comfortable with the paradigm, and should be applied to only those problems where minute control of the execution has significant advantages.

Organizations are using either a Lambda- or a Kappa-based architecture, with more moving to Kappa's stream-first architecture as the data processing paradigm. See "Stream Processing: The New Data Processing Paradigm."

There is a plethora of SQL processing engines built for large-scale data lakes and that support use cases related to SQL data processing, BI, ad hoc workloads and so on. However, most of the SQL engines degrade in terms of performance latency as the concurrency increases with complex analytic queries on large datasets. It is recommended not to use SQL processing engines on the data lake for extremely large SQL and BI workloads with high concurrency and highly complex analytical queries.

Consumption

Data lakes can ingest and process volumes of data, but the real value of a data lake is realized only when the processed data is consumed to build data-driven applications. Making the data available to downstream applications and consumers in the data lake lays the foundations for innovation and building data-driven applications. Easing the process of accessing the data lake allows business transformations to come from anyone in any line of business, not just your data scientists. One of the main requirements from a data lake is to cater to a variety of consumers. Each consumer has different requirements regarding the attributes they want and in what format.

Consumers of data from a data lake consist of two types:

- Processes and workloads *within* the data lake. For example:
  - ETL or ELT workloads
  - ML algorithms
  - Data-lake-based data marts
  - Search-based access
  - Data processing and analysis tools
  - Data governance, data catalog, security, metadata and MDM tools within the data lake
- Processes and workloads *outside* the data lake. For example:
  - Push the data to data warehouses and/or data marts
  - Access to data visualization tools and BI tools
  - Search-based access
  - Data analysis tools
  - Data governance, data catalog, security, metadata and MDM tools outside the data lake

Data from a data lake can be accessed in two ways:

- Data push:
  - Data exports
  - Data publish to message queue to be picked up by downstream processes
- Data pull:
  - Data services
  - Data views
  - SQL access
  - REST API
  - GraphQL
  - Data virtualization tools
  - Data marketplaces
  - Data services that provide a mechanism to deliver data based on a contract to a consuming application over lightweight protocols, as a pull-based mechanism from the consuming applications

Questions to Ask

Data lake architects and engineers should ask the following questions before architecting the consumption layer of a data lake:

- What are the data access throughput and latency requirements to the consuming applications?
- What are the SLAs for the throughout and low-latency reads and updates?
- What kind of access is required by the consuming application — random reads, sequential reads, random writes or sequential writes?

  An indexing capability allows for random reads and also serves a small portion of the huge dataset quite fast.
- How will you architect the data consumption layer to handle scalability (of data push or pull) and concurrency?
- How will you ensure fault tolerance and handle both hardware and software failures?
- How will you ensure that the data lake is capable of serving multiple data models?
- What are the data structures or data stores that needs to be created to make the access easier, faster, scalable and performant?
- Can the consumers shop for and prepare the data just like a data marketplace?
- Can the consumers search the data lake comprehensively either directly or through a catalog?
- Is the data access automatically governed for data security with the right kind of access policies?
- Is the data access logged, audited and monitored, and are metrics around these captured and stored?
- How will you expose the data quality and data profile results to the data consumers to gain their trust of the data in the data lake?

Best Practices

This section provides some of best practices that organizations, data lake architects and data engineers should follow when building the consumption layer of the data lake:

- Integrate the consumption layer with a data catalog/data governance and security tools to control access policies/authorization/authentication. Setting and enforcing security policies is essential for successful use of data within a data lake. The data owners configure the security access requirements of the data, detailing who can access what data. The security details are stored as part of metadata, which is used by data stewards to enforce these policies.
- Support the most common ways of data access from the data lake, including SQL, APIs, search, exports and bulk access.

- Self-service capabilities: Self-service consumption is essential for a successful data lake. Different types of users consume the data, and they are looking for different things — but each wants to access the data in a self-service manner, without the help of IT.
- Track lineage, auditing and logging of what data is consumed by whom, when, where and how.
- Maintain a data catalog that describes the content, data definition, and all the metadata that is captured as the data moves in the data pipeline from data ingestion to data enrichment, integration, transformation and consumption. This should be kept up-to-date with automated mechanisms and published by the data steward to the stakeholders.

Challenges

Some of the challenges that data lake architects and engineers face when designing the consumption layer include:

- Designing for high concurrent access to the data stored with a data lake across the data stores
- Management of the data model and schema, and avoiding data, model and schema inconsistencies
- Ensuring data findability by implementing the right kind of indexing, tagging and search capabilities
- Supporting self-service across the different consumers of the data lake
- Ensuring security of the data being accessed both by internal and external users
- Ensuring continuous maintenance of the quality of the data being access
- Enforcing logging, tracking and auditing of the data access, as well as integration with tools that support these capabilities

Operations

Building a data lake may be difficult and complex, but sustaining it is even more difficult. Managing a production data lake can easily become complex and foggy if the right kind of tools and efficient monitoring-and-alerting framework are not in place.

Due to the enormous complexity of building a data lake from ideation to production, organizations should realign their focus in building data lakes from a component-based architecture to an application-centric approach. The idea is to reduce the time to deployment, reduce the time to market and increase repeatability of the process.

This section briefly discusses the operationalization aspects of a data lake. For more details, please refer to “Operationalizing Big Data Workloads.”

Best Practices

- Improve efficiency by using a repeatable workflow that is configuration-driven, without writing code.
- Employ automated deployments with single-click deployments and the capability to roll back easily.
- Introduce proactive monitoring and alerting to keep constant check on platform availability and critical metrics. Provide operational intelligence and publish metrics to highlight availability, trends, SLA violations and so on.
- Perform incident analysis and a health checkup of the data lake in an automated way.
- Before building the data lake, perform the exercise related to cluster planning, capacity planning, and cluster design and component layouts. Have a map of the resources available on the data lake and the different tools, software and frameworks that are deployed.
- Ensure master processes for components that utilize a master-slave architecture are distributed across racks to minimize risk due to rack failures. Deploy multiple gateway nodes for load balancing and client operations.
- Automate the provisioning and deployment processes through tools like Chef, Puppet, Jenkins and Red Hat Ansible.
- All security components should be deployed in HA mode.
- Leverage DevOps and "DataOps" teams to use continuous integration, continuous delivery and continuous deployment principles to ensure a seamless data lake deployment process.
- Use containerization of workloads and apply container orchestration tools like Kubernetes to ensure better resource utilization, multitenancy, resource isolation, autoscaling and self-healing where appropriate.

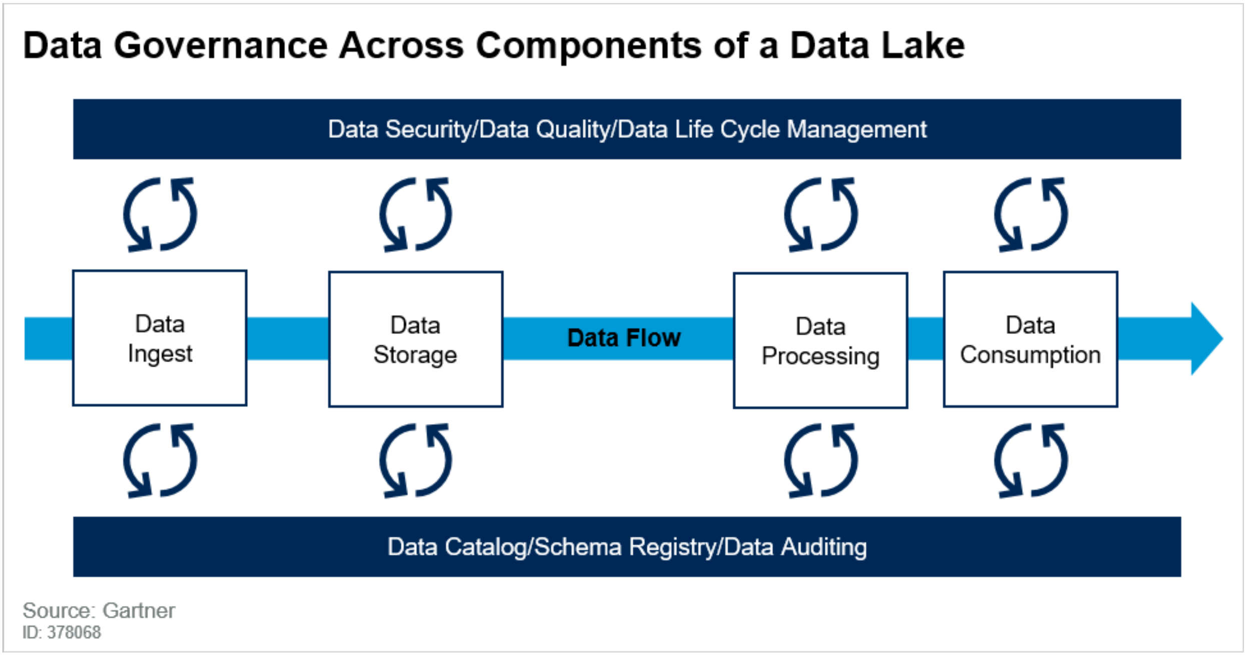
Governance

Data governance gives the right control and trust in the data in the data lake. This gives confidence to the consumers of data in the data lake to make business decisions. Data governance is a set of formal processes that ensure that data within the enterprise meets expectations, such as:

- Data is acquired from reliable sources.
- Data meets quality standards.
- Data conforms to well-defined business rules.
- Data is accessed and modified with the right policies, guidelines and access controls.
- Data follows a well-documented change control process.
- Trustworthiness of the data remains intact as the data flows through the data lake.

Figure 9 shows the different forms of data governance as it is applied across the different components of a data lake.

Figure 9. Data Governance Across Components of a Data Lake


Data Governance Across Components of a Data Lake
Data Security/Data Quality/Data Life Cycle Management
Data Ingest
Data Storage
Data Flow
Data Processing
Data Consumption
Data Catalog/Schema Registry/Data Auditing
Source: Gartner
ID: 378068

Data governance has four major pillars: data quality, data catalog, data security and data life cycle management. The following sections discuss details related to each of these pillars.

Ensure governance across all components of the data lake. Use data catalogs to ensure data quality at ingestion time, secure communication and every component within the data lake, and ensure the data life cycle is managed within the right kind of policies. Also, automate, configure, coordinate and control the operationalization of data lakes (see "Operationalizing Big Data Workloads").

Data Quality

The need for data quality is not new nor specific to data lakes. Data lakes make fixing data quality issues both urgent and challenging for a few reasons:

- The unstructured nature of the data ingested into the data lake
- Volume
- Velocity

This makes ensuring data quality in a data lake a more challenging proposition and sometimes more elusive. Correctness is difficult to determine when using data from external sources, and structural integrity can be difficult to test with unstructured and differently structured (nonrelational) data. Validating all data entering the lake is hard. Each source features unique technical challenges and a particular set of data issues, necessitating the development of specific data validation functions for each source.

Ensuring data quality involves:

- Data Quality Detection
- Data Quality Remediation (Ignore/Correct)
- Automated Detection, Flagging
- Setup Rules — Configuration-Driven
- Data Rejection
- Profiling and Classification of Data Based on Rules (Good, Bad or Ugly)
- Automate Data Quality Detection and Remediation

Questions to Ask

- How should you determine the data quality level for the data sources in the data lake?
- What tools should you use for data quality detection, data quality correction and automation?
- How often is the data quality to be assessed?
- Is the data accurate, complete, valid and/or consistent?
- How can you identify and implement data quality rules for monitoring and improvement?

Best Practices

- Data quality has to be ensured across all the layers of the data lake.
- Avoid data lake indigestion. Identify, fix and flag data quality problems at the point of ingest by building a robust set of data validation rules for each data source as it is loaded into the data lake. Validation rules compare the contents of each field in each record with a set of parameters and thresholds to determine whether that record contains quality data.
- Do not skip automatic data validation and profiling during the ingest process. This is the single root cause of failure of many data lake projects. Just because it is easy to ingest data into a data lake, do not fall into the trap of dumping bad quality data into a data lake for experimentation purposes.
- Data quality and profiling should be integrated closely with data cataloging tools by specifying data quality, profiling and distribution metrics in a data catalog and by cross-checking it during and after ingestion.
- Implement rules and systems to continually monitor data quality.

- With the help of data stewards, have controls in place and steps to be taken when data quality is not to the required level.
- Always ensure data quality as a precursor to the correlation of data across different siloed sources and before scheduling long-running data pipelines.
- Automate data quality detection when orchestrating the data pipelines. It does not make sense to run expensive data processing — which consume time, resources and money — on bad quality data.
- Automate data quality rules just like operators in a programming language. Organize rules into rule sets.

Challenges

- If the data in the data lake is not validated and profiled, it results in data liability, where the ingested data is questionable to use in terms of its quality and usefulness.
- Data quality issues in a data lake are generally tied to delays in making data available and (ultimately) the delivery of bad data.
- Not ensuring proper data quality within a data lake erodes consumer confidence.

Data Catalog

The single most important reason data lake implementations fail is because data lakes degenerate into data swamps due to of lack of proper metadata around them. Data lakes are reincarnated by making data catalogs an integral part of the ecosystem that is integrated across all layers and components. A data catalog should be able to answer the following for a data lake:

> What data is there, what does it mean, where is it used and who has responsibility for it?

Context is everything when it comes to data in the data lake. A data catalog helps companies build a map to organize and locate data stored in the data lake. It contains details on each dataset, both at a high level — in terms of its origin, date and time of ingest, and who ingested — and on a granular level regarding each piece of data, such as the data's profile and quality metrics, its data types, and lineage. The catalog is the first tool anyone looking to work with the data lake should use to find data to build insights.

A data catalog solves multiple problems, such as:

- Gives a comprehensive view of each piece of data across databases
- Makes the data easy to find
- Puts guardrails on the data and governs who can access it

- Enables the data lake to support different use cases such as self-service analytics, self-service data preparation, data virtualization and multicloud-based data architectures

Data catalog vendors are innovating by building catalogs powered by AI, ML and knowledge graphs that:

- Generate technical metadata
- Enable semantic inference and recommendations
- Catalog business context
- Discover entities within unstructured data
- Manage highly complex relationships, data quality and data lineage

Metadata management tools and business glossaries are being integrated and are evolving as holistic data catalog tools. The data catalog becomes an indispensable guide to data, unlocking its potential and enabling data-driven organizations.

The data catalog guides the producers and consumers of a data lake to find the right data for their interest. It provides clarity in the form of agreed definitions, supplied by subject matter experts, with a high-level view of the interconnectedness between data and the business processes that it supports. Data can be both a liability and an asset. A data catalog is an important component to becoming a data-driven organization. It is not enough to simply have lots of data, but the goal should be to maximize its findability and proper usage through ownership, understanding and trust.

Questions to Ask

- What are the data catalog access mechanisms?
  - REST APIs to integrate with other tools
  - Self-service capabilities
  - Search capabilities

    What kind of search capabilities are provided by the data catalog? With the volume and complexity of data in a data lake, it is extremely important to have a data catalog that can be searched efficiently in multiple ways across different tools and applications. It is best if the search capabilities of a data catalog can be integrated with other applications in an organization that needs to access the data lake. Nontechnical users need to search the catalog to find the data useful for their specific purpose.
- What kind of automated data discovery is possible?
  - Automated data profiling
  - Automated derivation of lineage
  - Automated field-level tagging

- Automate data quality detection based on rules and thresholds

- What are the relationships/dependencies that exist between the various datasets and the entities of the datasets?
- How do you determine the current data quality of the data in the data lake?

Best Practices

- Making sure all datasets are labeled, tagged and classified is imperative for successful usage of the data within the lake.
- Ideally, have a single, enterprisewide, scalable, multicloud data catalog. If this is not possible and your organization has multiple catalog tools, research into integrating them to reduce duplication and ambiguity.
- A data catalog should have an open API where tools, applications and other components of the data lake or LDW can seamlessly integrate with each other.
- Keep your data catalog current. Data lakes are constantly changing and evolving with new data sources being added continually. A data catalog should be an updated navigation map for the entire data lake and must be in sync with the data that currently exists in the data lake.
- Ideally, have a data catalog that integrates with data quality tools to measure data quality, enable correction of errors, and provide data that is fit for usage.
- When choosing a data cataloging tool for a data lake, consider these important capabilities:
  - Support native big data processing for performance and scalability.
  - Automate data discovery and classification, both for initial data loads and for ongoing discovery.
  - Catalog the data ingestion, data processing, data lineage and data security details and profiles.
  - Integrate with other enterprise metadata repositories.
  - Establish collaboration capabilities across a variety of users and the ability to annotate the contents of the data catalog at different granularity.
  - Keep an inventory of what data is there, where it is, its format, sensitivity, profile and time stamp, all built from the technical metadata.
  - Select a solution that can connect to the widest range of data sources.
  - Enable the data steward with tools to maintain data compliance.
  - Search for data across the data lake based on various search criteria including keywords, similarity searching, and facet-based and tag-based searching.

- Select a data catalog fueled by ML and AI, with automate data discovery and autogenerated recommendations to view and explore similar datasets.

Challenges

Organizations that neglect to build and integrate a data catalog within a data lake will eventually create a data swamp and underutilize the data within a data lake. This leads to:

- Lack of trust in data
- Poor awareness of what data actually exists
- Confusion around the meaning of data
- Data becoming obsolete due to lack of active ownership

Data Security

Organizations building data lakes with large amounts of data from a variety of sources need to carefully protect their data assets, just as with natural lakes, which have rules and regulations to ensure the safety of swimmers.

Important aspects of privacy and security of the data in a data lake include:

- A greater volume of data in a data lake implies a larger surface area of risk and attacks.
- Newer data types like sensor data, connected home data, fitness data, telematics and so on have different privacy and risk implications than traditional datasets.
- Performing data integration, enrichment and linkage with sensitive data in the lake can often unknowingly result exposure of private data.
- Combining data can inadvertently identify individuals in a way that the different types of aggregate data cannot.
- Data discovery, experimental and exploratory analysis in the data lake can create unanticipated risk exposure to private data.

Security solutions in a data lake should provide, at minimum, the following capabilities:

- Richness of access policies
- Built for scale and automation
- Data obfuscating and redaction capabilities
- Data AUDITABILITY
- Provides unified visibility — alerting, monitoring and logging

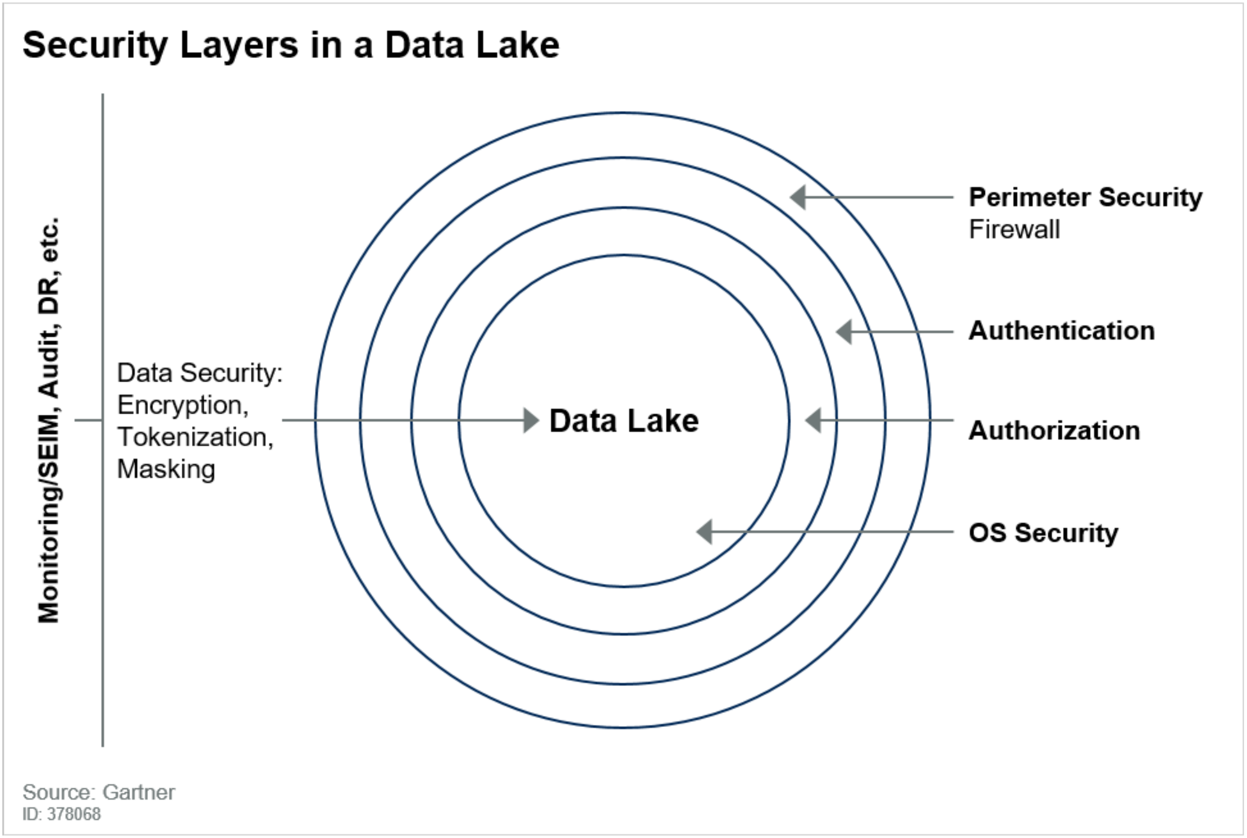
- Hybrid and multicloud-ready, governance, risk assessment, risk management, and compliance capabilities (a data lake would typically retain data for a long period of time, and that data might be sent across to multiple cloud providers and back and forth with on-premises solutions)
- Data disposal

Security in a data lake can be broadly split into three areas:

- **Authentication:** Determining the legitimacy of the identity of a user
- **Authorization:** The privileges that a user holds to perform specific actions
- **Access:** The security mechanisms used to protect data, both in transit and at rest

Figure 10 shows the different security layers in a data lake.

Figure 10. Security Layers in a Data Lake


Security Layers in a Data Lake
Monitoring/SEIM, Audit, DR, etc.
Data Security: Encryption, Tokenization, Masking
Data Lake
Perimeter Security
Firewall
Authentication
Authorization
OS Security
Source: Gartner
ID: 378068

The ingestion layer poses a high security risk because:

- Incoming raw data has not been evaluated of usefulness, and in its raw state, it is devoid of any classification about its eventual usage, authorizations and privacy.

- This raw data may contain personally identifiable information, and since it is untouched and no data masking has been applied, there is a higher propensity of exposing sensitive data.
- This raw data with sensitive information can be combined, linked and enriched with data in the data lake, resulting in security breaches.

Questions to Ask

Organizations, data architects and security engineers should ask these questions before building the data lake:

- Is the incoming data sensitive? Does it contain PII, protected health information (PHI) or Payment Card Industry (PCI) information?
- Who is entitled to read and/or write the data? Is the user allowed to see all the data? Are they limited to certain rows, or even certain parts of certain rows?
- Does this data need anonymization, sanitization or encryption?
- What kind of encryption is required? Is the same encryption to be used across all layers in the lake?
- Where does the data need to be protected/encrypted: at rest, in motion and/or in use?
- How and when can the data be disposed?
- How do you dispose the keys associated with encryption?
- Is the data safe when the user runs analytics?
- Is the data safe once the user has completed the data access? For example, there is no point in ensuring the data is safe across all layers of the data lake only to write plain text results to an unsecure area?
- Can conclusions be made from aggregated data?

Best Practices

Organizations, data architects and security engineers should follow these best practices when building the data lake:

- Security must be designed from the beginning.
- Apply the best security practices you would apply to legacy database implementations.
- Be aware of data types going into your data lake — sources, dependencies and levels of sensitivity.
- To mitigate the security risks in the ingestion layer, perform raw data analysis in a quarantined landing zone where only a small number of authorized users are provided access to all the data.
- Ideally, encryption should be done to:

    - Data at rest
    - Data in transit (shuffle/internode data transfers)
- Log and monitor data access across all layer in the data lake.
- Automate the infrastructure provisioning, and make it configuration-driven. Have security settings incorporated within the automation process rather than relying on manual intervention processes.
- Have separate security privileges across all personnel with access to the data lake, including data engineers, data scientists, QA engineers, DevOps and DataOps teams, and data and business analysts.
- Select the right type of encryption based on performance, data size and application transparency.
- Ensure that you encrypt ephemeral data storage across the cluster.
- For data on object storage, turn on object-level auditing.
- The key to successful data security is to implement the metadata-driven governance approach across all layers of the data lake. Use this approach to classify or tag data based on different security and protection requirements of the multiple types of data deposited in the data lake.
- Provision Hadoop and Spark deployments on IAM concepts. For structured data, use RBAC, roles and the data policy based on organizational policies. For unstructured data (text, audio and video), use coarse-grained platform security or add third-party layers that govern access.
- Limit the entry points into your cluster to specified edge nodes and authorized middleware.
- Ensure correct disposal of data. This is the most overlooked area of data security. Use policies with life cycle management tools to identify if data is simply out-of-date and no longer has value. Also, if data is encrypted, and is no longer required, simply throw the keys away.

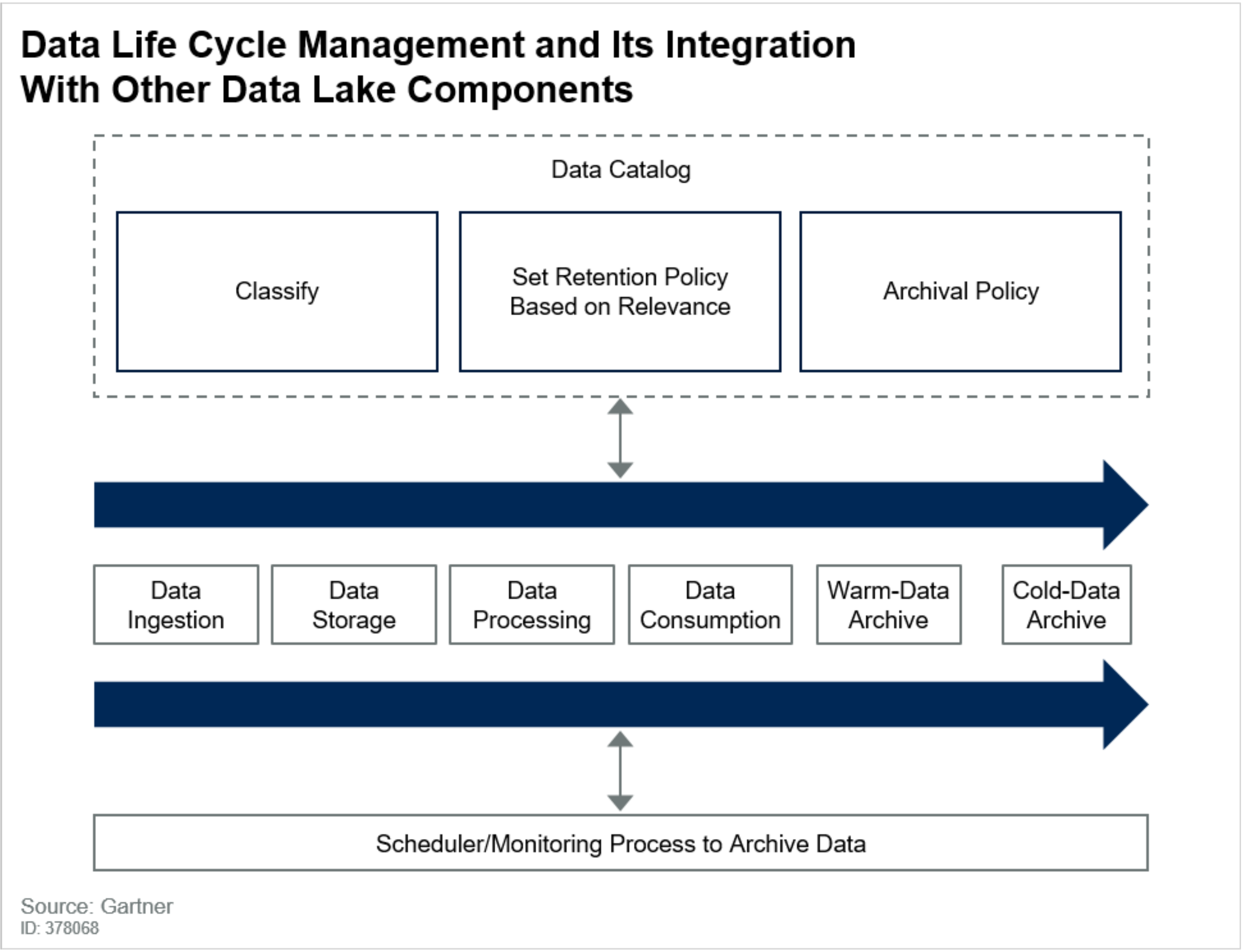
Data Life Cycle Management

Data Life Cycle Management (DLM) is a subprocess of data governance. The DLM layer strives to achieve strategy and policies for classifying which data is valuable and how long you should store a particular dataset in the data lake. The life span of data can be partitioned into multiple phases, categorized by different patterns of usage, and during different phases, the data can be stored differently. DLM helps to identify the true value of data over its lifetime and classifies it so data is stored, migrated or deleted according to its value. DLM provides a structured capability to classify data based on relevance and how the data evolves or grows over time.

Over periods of time, the value of data tends to decrease and the risks associated with storage increases.

Hence, it does not makes sense to keep the data in a data lake continuously without plans and policies in place to delete data that is past its “use by” date. Figure 11 shows how the DLM strategy and process should be integrated with the data flow, data catalog and orchestration tools.

Figure 11. Data Life Cycle Management and Its Integration With Other Data Lake Components


Data Life Cycle Management and Its Integration With Other Data Lake Components
Data Catalog
Classify
Set Retention Policy Based on Relevance
Archival Policy
Data Ingestion
Data Storage
Data Processing
Data Consumption
Warm-Data Archive
Cold-Data Archive
Scheduler/Monitoring Process to Archive Data
Source: Gartner
ID: 378068

Questions to Ask

- What changes happen to the data over a period of time? Will its value diminish? Is it OK to hold it in the data lake?
- What is the extent of the data protection and data availability needed for this data?
- What are the applicable legal policies?

Best Practices

- DLM should be defined across each layer of the data lake:
    - Ingestion Tier

  - Storage Tier
  - Consumption Tier

  Because the Consumption Tier deals with the distribution of data from the data lake to internal and external data customers, every transaction that distributes data is tracked, logged and monitored so that it adheres to the ILM policies of the organization.

- Data disposal — Secured data should have an agreed life cycle, set by a data authority or data steward who is knowledgeable both with the data and with the business and commercial context. A dataset labeled as "sensitive" may require encryption for the first year and "no encryption" thereafter before finally being disposed. DLM ensures that everyone knows exactly how the data is to be treated. The life span of the data is based on usage frequency, classification and relevance. For each classification of data, the age parameter may differ by the data's relevance or governmental norms.
- Data prioritization — Selection and classification of data should be the first exercise of DLM strategy.
- Ensure rules governing what can or cannot be stored in the data lake.
- Define your archival policy, and use automated rules based on data life cycle management, integrated with the data catalog. The frequency of access is one of the methods to find whether or not the data is relevant to the business.
- It is not a good approach to put a hard timeline for all objects in the data lake. It could have an adverse impact on the business relevance of data.
- Not monitoring the data usage can create data swamps over a period of time.
- Not integrating data lake life cycle management with the data catalog can cause confusion, inconsistencies and data loss.

Challenges

- Velocity and variety of data, as well as its source and veracity, adds to complexity in defining and understanding the governance and life cycle policies on data in a data lake.
- Unstructured data has challenges in terms of the eventual usability from an analytical-insights perspective. Most of this data is not bound by legal, regulatory and privacy norms. These are typically used in conjunction with the organization's own internal customer, financial and transaction data to produce insights, after data enrichment.
- Sheer volume of data in a data lake often poses challenges in enforcing DLM policies because volume provides a huge surface area for policy breaches and risks.

Strengths

- Data lakes are based on distributed scale-out architecture for compute, storage and I/O. This facilitates fault tolerance, HA and replication out of the box.

- Data lakes allow organization to better-utilize their resource consumption and utilization across different users and workloads.
- Data lakes provide the ability to work with multiple data types and data formats, which allows the best-of-breed selection of data formats, depending on workload SLAs.
- Data lakes decouple schema from data. Schema on read facilitates agility and flexibility and allows data lake users to be schema free or define multiple schemas for the same data.
- Data lakes provide capabilities to support a much broader range of possible use cases for enterprises — including data archival, data integration across a variety of data sources, ETL offloading from data warehouses, and complex data processing across batch, streaming and ML workloads.
- Data lakes can store and derive value from unlimited types of data — structured, unstructured, multistructured and so on.

Weaknesses

- Because of the architectural technical complexity and multiple moving pieces required to create a data lake, it often takes an enormous effort and to make data lakes work end to end, especially when leveraging open-source technologies. Hence, data lakes have a slow time to value. No end-to-end tools exist to do everything in a data lake, and a huge burden falls on development, operations, engineering and architecture.
- Data lake technologies are advancing at a rapid pace, and when trying to keep up with this pace, it can be extremely challenging to keep skill sets relevant and applicable.
- If data lakes are built without the right governance, as had been the case a few years back when organizations were building data lakes without any governance, they easily create data swamps. Although a large volume of data is available to users in the data lake, problems can arise when this data is not carefully managed, including:
    - Lack of data governance: Without the structure and controls to manage and maintain the quality, consistency and compliance of data, a data lake can rapidly devolve into a data swamp.
    - Poor accessibility: Although the data might be available, its value is limited because users are unable to find or understand the data.
    - Poor data quality and lineage: Users need to have the context of the data and know where it comes from to trust the data completely.
    - Lack of data security: Data loaded to a data lake without oversight can lead to compliance risks.
- Data lakes cannot be used for OLTP or operational workloads. Advances and developments are happening in this space, but the adoption curve is below the critical mass.
- Operationalization of data lakes is extremely challenging, complex and often error-prone. Promoting data pipelines from development into full enterprise production can be fraught with

perils, including wrong packaging, wrong configuration and wrong infrastructure. However, emerging tools in the DataOps space is helping to remediate this situation. See "Operationalizing Big Data Workloads."

- Unlike data warehouses, which have been around for 25-plus years, there are no industrywide-accepted best practices.

Guidance

To be successful with data lakes, technical professionals need to clearly define, understand and outline their requirements across the dimensions discussed in this section.

Data and Governance

Understanding the data to be ingested in a data lake, irrespective of whether it is external or internal data, is extremely imperative. Answer the following questions before building a data lake:

- What is the nature of the data in terms of volume, velocity, structure, source type and location?
- What is the data quality, its lineage, frequency of updates, and consistency and completeness?
- What are the security constraints of the data in terms of its usage and regulations?
- Who are owners, and what are the encryption requirements, access controls, and authorization authentication policies?
- What are the policies for data life cycle management?

Technology

Define the strategy and the high-level architectural components required in each layer needed. For example:

- What are the components, tools and frameworks being used across each of the layers in the data lake? For example, is the data lake going to be based on a distributed file system or an object store?
- What analytic tools will be required at each layer?
- What is the data ingestion and data integration strategy? How do you handle data drift and data schema changes and merge incremental data changes (CDC) into the full load?
- What is the I/O and memory model, and what are the compute, storage, latency and throughput requirements?
- How do you achieve code reusability across the different layers?

People

Build the right technical skill sets within the organization and continually develop those skill sets. Open-source technologies are continuously moving, changing and morphing.

Define, develop and hire across all the skill sets, including a data engineer, a data architect, data scientists, DevOps, DataOps and data analysts required to successfully architect and build data lake solutions.

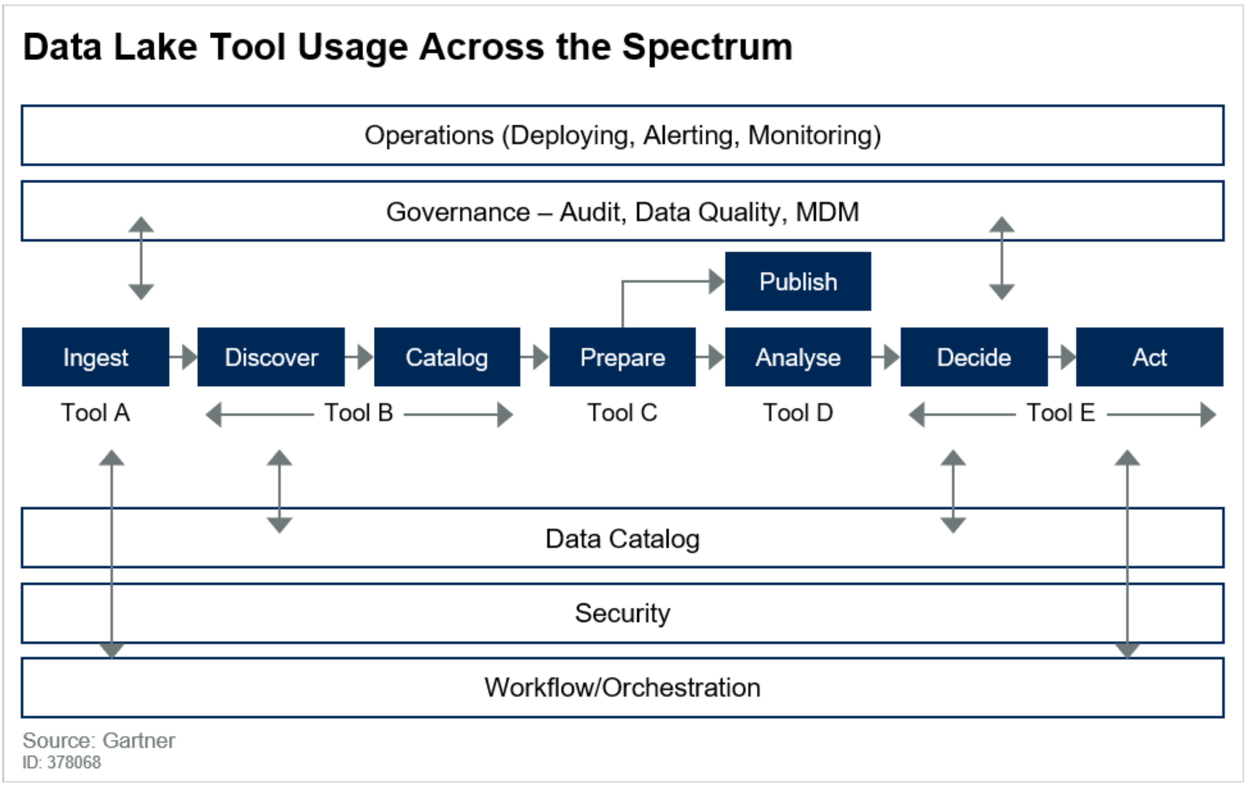
Operations

- Define and develop the data lake hosting strategy, whether it is in the cloud, on-premises, hybrid or multicloud. Plan and develop an end-to-end automation strategy across the different layers of the data lake, with self-service capabilities, and determine how you will leverage containerization and container-orchestration-based approaches.
- Invest in the right tools for building workflows while orchestrating and scheduling the end-to-end data flow processes. Invest in tools for monitoring, management, alerting and notifications.
- Operationalize end-to-end governance and continually collect and use operational metrics.
- Define and implement the nonfunctional requirements — such as DR, HA, fault tolerance and backup strategies.

Other Criteria:

- Manage, minimize and isolate complexity to reduce coupling, and build a configuration-driven system.
- Select the best-of-breed tools. There is no single end-to-end tool for building data lakes across its myriad layers and requirements. Develop components of the tool chain by selecting the best-of-breed tools at each layer and by integrating them with APIs and scripts to develop and end-to-end system. Figure 12 shows how different tools within the data lake ecosystem can integrate.

Figure 12. Data Lake Tool Usage Across the Spectrum



- Develop a strategy for build versus buy across all the tools required for your data-driven ecosystem.

The Details

This section takes a deeper dive into each of the components of a data lake and discusses architectural underpinnings of these components.

Ingestion

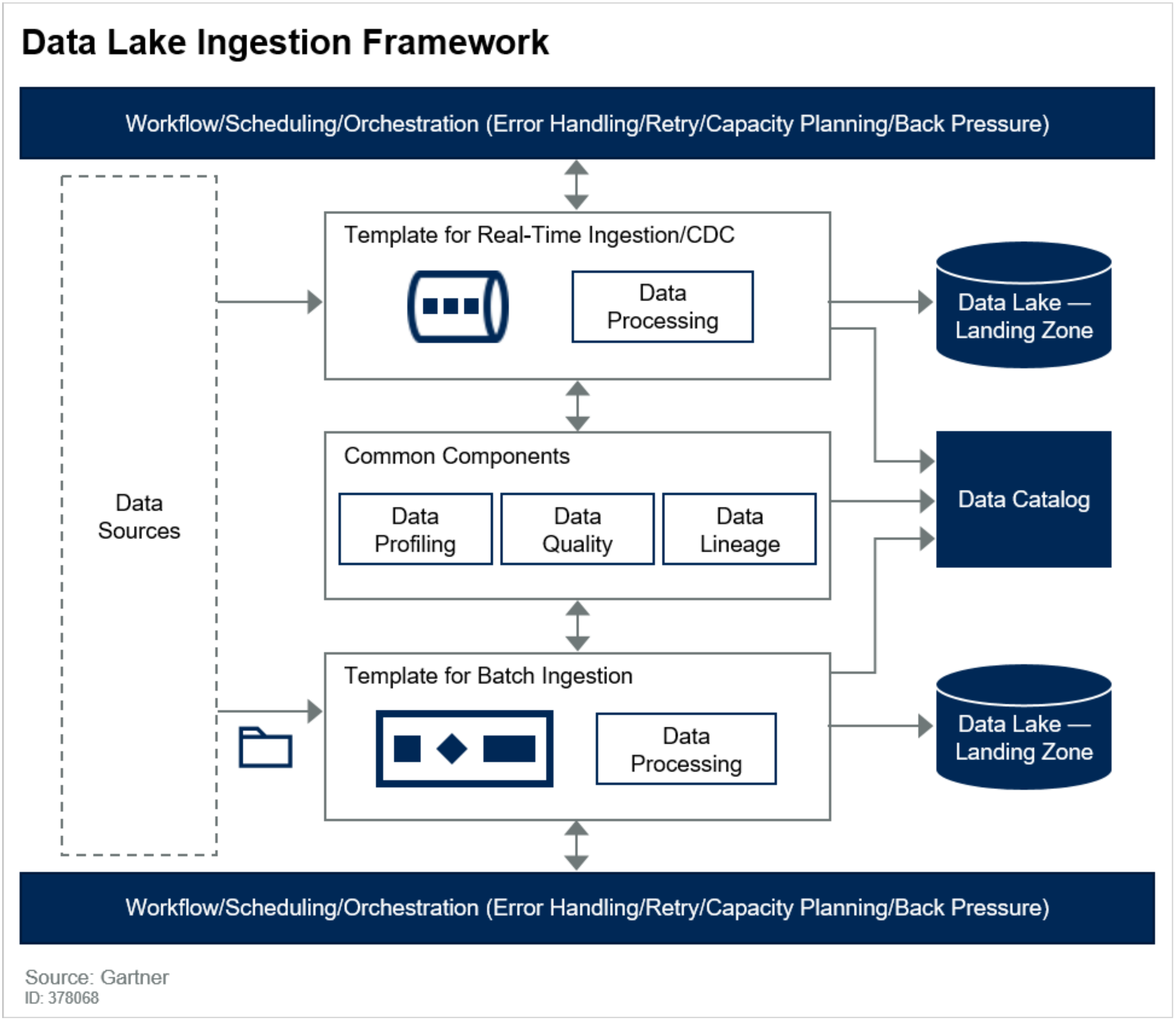
A data ingestion framework should have the following characteristics:

- A single framework to perform all data ingestions consistently into the data lake
- Metadata-driven architecture that captures the metadata of what datasets will be ingested, when they will be ingested and how often they need to be ingested; how to capture the metadata of datasets; and the credentials needed to connect to the data source systems

- Template design architecture to build generic templates that can read the metadata supplied in the framework and can automate the ingestion process for different formats of data, both in batch and in real time
- Tracking metrics, events and notifications for all data ingestion activities
- Single consistent method to capture all data ingestion along with technical metadata, data lineage and governance
- Proper data governance with "search and catalog" to find data within the data lake
- Data profiling to collect the anomalies in the datasets so that data stewards can look at them and come up with data quality and transformation rules

Figure 13 shows the high-level framework for ingestion based on the above framework.

Figure 13. Data Lake Ingestion Framework


Data Lake Ingestion Framework
Workflow/Scheduling/Orchestration (Error Handling/Retry/Capacity Planning/Back Pressure)
Data Sources
Template for Real-Time Ingestion/CDC
Data Processing
Data Lake — Landing Zone
Common Components
Data Profiling
Data Quality
Data Lineage
Data Catalog
Template for Batch Ingestion
Data Processing
Data Lake — Landing Zone
Workflow/Scheduling/Orchestration (Error Handling/Retry/Capacity Planning/Back Pressure)
Source: Gartner
ID: 378068

Batch-based ingestion tools include Sqoop, NiFi, Oracle Copy to BDA, Pivotal Greenplum (gphdfs) and HDFS Copy.

Real-time streaming, ingestion-based tools include, Apache Flume, Kafka, NiFi and StreamSets.

This research does not discuss batch and real-time data ingestion. The next section focuses on change data capture-based data ingestion into a data lake.

Change Data Capture

The different ways CDC can be accomplished are listed in Table 3.

Table 3. Different Ways to Accomplish CDC

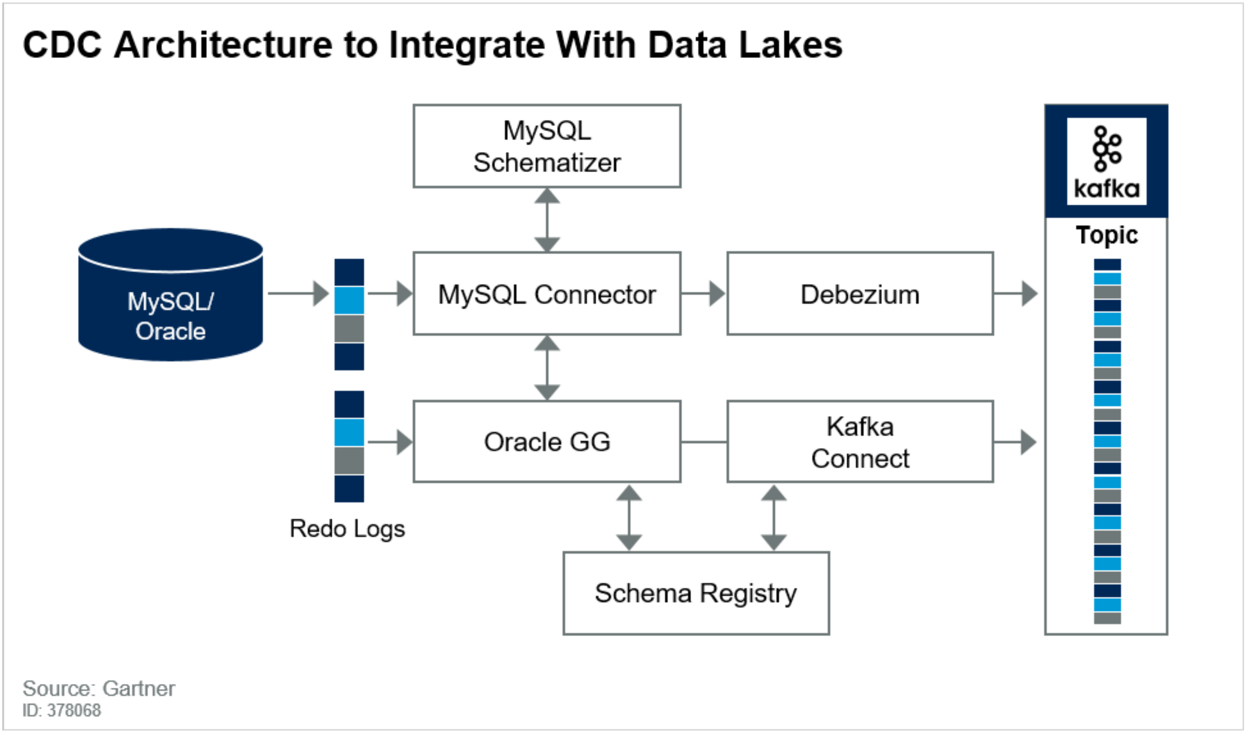
Capture Method	Description	Production Impact
Log Reader	Scans recovery transaction logs; use when access to logs is available	Minimal
Query	Flag new transaction with time stamp and version number	Low
Trigger	Source transaction triggers CDC; use when *no* access to logs is available	Medium

Source: Gartner (July 2019)

CDC Delivery Methods

- Transactional — CDC copies data updates/transactions in the same sequence in which they were applied to the source. This is appropriate when sequential integrity is more important to the analytics user than ultra-high performance.
- Aggregated/Batch — CDC bundles multiple source updates and sends them together to the target. This facilitates processing of high volumes of transactions when performance is more important than sequential integrity.
- Stream-Optimized — Stream-optimized CDC replicates source updates into a message stream that is managed by streaming platforms such as Kafka, Microsoft Azure Event Hubs, MapR-ES or Amazon Kinesis. This is data in motion. It supports a variety of new use cases, including real-time, location-based customer offers and analysis of continuous stock-trading data.
- Open Source — CDC can also be done with open-source tools like Debezium to ingest data from change logs into a distributed messaging platform like Kafka. This architecture is show in Figure 14.

Figure 14. CDC Architecture to Integrate With Data Lakes



A very common customer question is: How do you ingest data from applications generating data deployed on-premises to a data lake in the cloud. Figure 15 shows a high-level architecture on how to handle this. Tools like Attunity Replicate are architected for doing this kind of data ingestion.

Figure 15. Using CDC to Ingest Data From On-Premises to Cloud

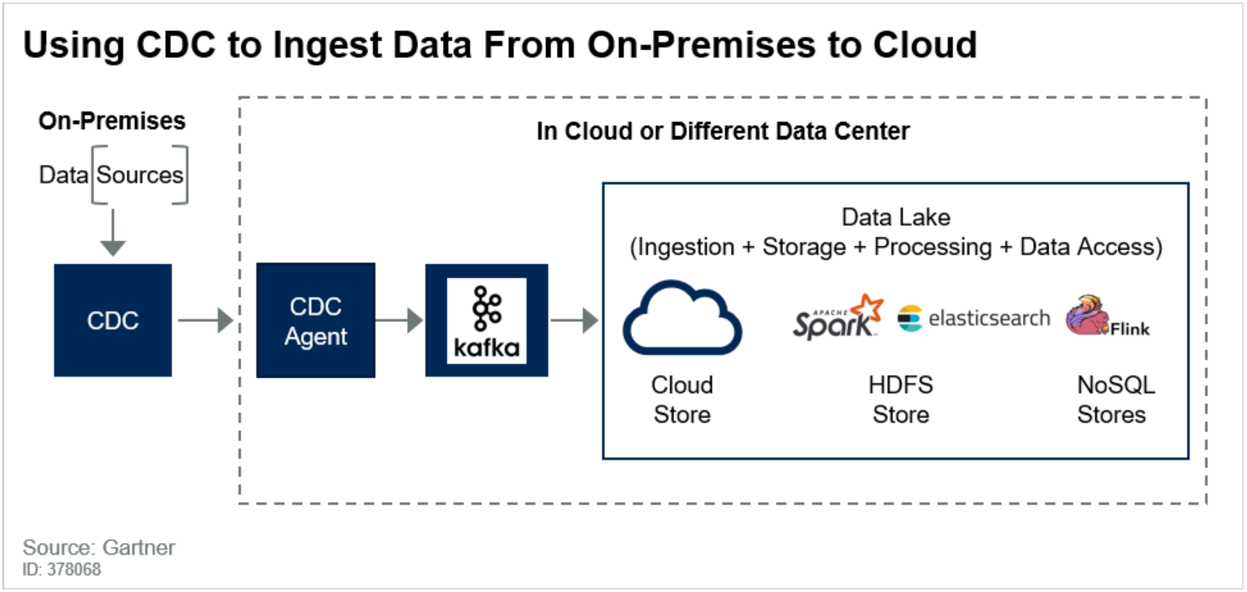

Using CDC to Ingest Data From On-Premises to Cloud
On-Premises
Data Sources
CDC
In Cloud or Different Data Center
CDC Agent
kafka
Data Lake
(Ingestion + Storage + Processing + Data Access)
Spark
elasticsearch
Flink
Cloud Store
HDFS Store
NoSQL Stores
Source: Gartner
ID: 378068

Table 4 shows some of the products, vendors and frameworks for data ingestion into a data lake.

Table 4. Some Products, Vendors and Frameworks for Data Ingestion Into a Data Lake

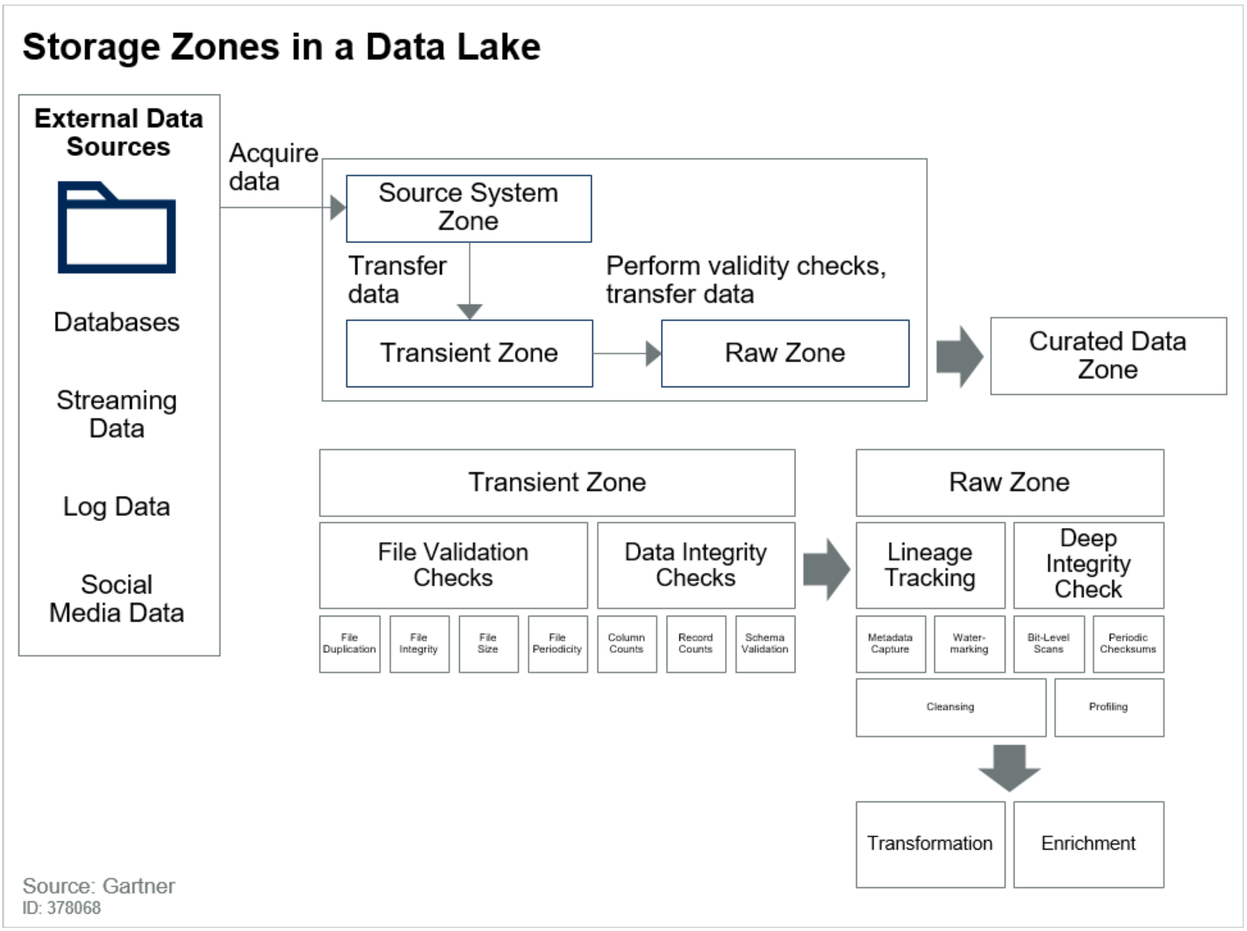
Batch	Sqoop, StreamSets, Apache Gobblin, Suro (Netflix)
Real time	NiFi, Kafka, Flume, StreamSets, Amazon Kinesis, Azure Event Factory, Google Cloud Pub/Sub, Streamlio, Gobblin, Suro (Netflix), Mozilla (Heka),
Log files	Suro (Netflix), Elastic Filebeat. Elastic Stack Logstash,
CDC	Databus (LinkedIn), Debezium, Attunity, IBM InfoSphere Data Replication (IIDR), Oracle GoldenGate for Big Data,
Replication	Attunity, SQData

Source: Gartner (July 2019)

Storage

The ingested data needs a landing place. This is where the data lake architects should design the zones of a data lake. Figure 16 shows the different zones in a data lake and some example functionalities that are performed in those zones.

Figure 16. Storage Zones in a Data Lake



The Source System Zone establishes the connectivity to the external data sources. It accesses the data, extracts the required data, and transfers the data into the transient landing zone for it to perform basic validity checks. After the validity checks have been performed, the data is moved into the Raw Zone to make it accessible for further processing. Once the data goes through data cleaning, data profiling, transformation and enrichment, the data is moved to the Curated Data Zone.

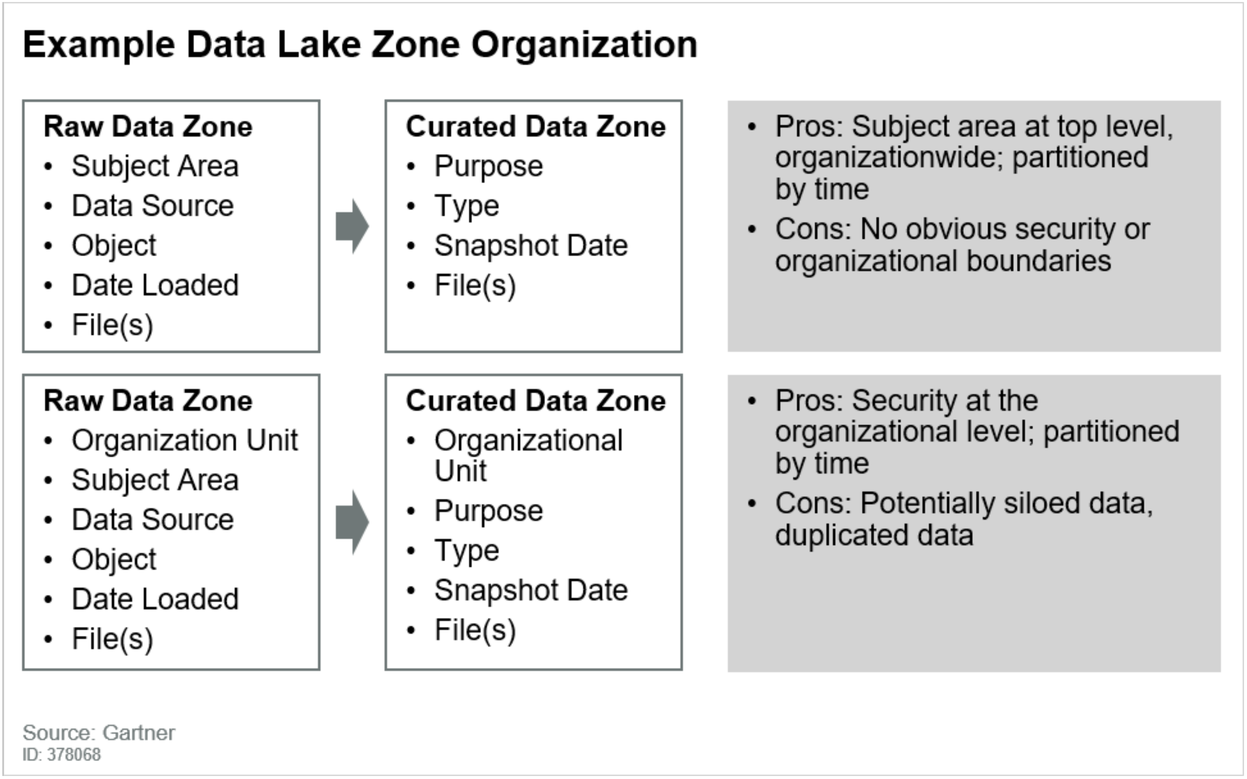
Transient Zone: This perform basic data quality checks just to make sure that the data ingested passes the minimum data quality thresholds before the expensive downstream data pipelines are started.

Raw Zone: The data in the Raw Zone should be immutable. It should retain the original data to accommodate future needs. It should be able to support any data, including batch, streaming, incremental and full load.

Curated Data Zone: This is the cleansed, organized, enriched data for the data consumption layer. Most downstream self-service data access should use the data from the Curated Data Zone.

Data in each of the zones can be organized in any of the ways shown in Figure 17. The objective is to avoid a chaotic data swamp. Plan the structure based on optimal data retrieval. The organization pattern should be self-documenting. The zones could be implemented as a directory or a bucket structure.

Figure 17. Example Data Lake Zone Organization



Another aspect of data lake storage design and architecture is the capacity planning around storage. Capacity planning becomes a function of some of these parameters:

- Initial data size — Historical and current data that will be moved into data lake
- YoY growth — Per year data growth rate
- Compression ratio — The factor by which the data gets compressed; measurement of compression factor varies by data types
- Replication factor — Number of replicas in a cluster (A higher replication factor impacts query performance and data availability. A replication factor of 3 is an optimum number that can balance availability with performance.)
- Intermediate data size — Hadoop creates multiple temporary files during intermediate stages; temporary data size accounts for 30% to 40% of raw data size

- A good rule of thumb is: Total storage required = (initial data size + YoY growth + intermediate data size) * replication factor * 1.2

Storage formats are an important consideration for optimum data lake performance in terms of storage footprint, I/O, and query and data processing performance. Table 5 outlines some of the storage formats and their usage patterns.

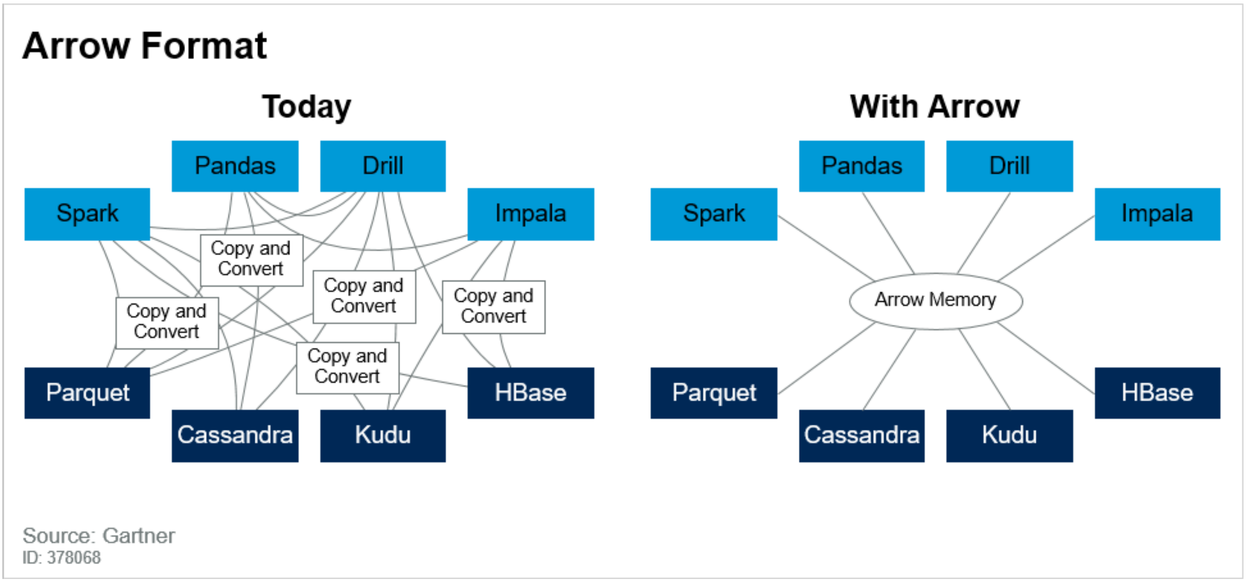
Table 5. Data Formats

Format Type	Features	Use
Apache Parquet	▪ Columnar and nested data support	▪ Slower write/update performance ▪ Optimum for analytic query
Apache Avro	▪ Row format data, nested data support, supports schema evolution	▪ Supports schema embedded within the file ▪ Supports splitting and block compression
Apache ORC	▪ Optimized record columnar files ▪ Row format data as key value ▪ Hybrid of row and columnar format	▪ High compression possible ▪ Slow write/update performance ▪ Used for analytic queries ▪ No schema evolution
SequenceFile	▪ Native support with MapReduce ▪ Splitable ▪ Transparent compression	▪ Limited schema evolution ▪ Supports block compression ▪ Used for intermediate file in MR
CSV/TSV Files	▪ Regular flat files with or without header information	▪ Easy to parse ▪ No block compression support ▪ No schema evolution
JSON	▪ Key-value-based, semistructured, nested and hierarchical data	▪ No block compression support ▪ Schema evolution is better than CSV

Source: Gartner (July 2019)

Apache Arrow is a new storage format that is gaining more usage across the industry. In-memory columnar data format with O(1) random access performance. It is highly optimized for numeric data and does zero-copy reads. Its layout is highly cache-efficient for analytics workloads and is efficient for fast data interchange between systems without serialization costs such as Apache Thrift, Avro and Google's protocol buffers. It can support complex data structures (hierarchical/nested) and dynamic schemas, and it works very well with modern CPU architectures — cache prefetching and avoiding CPU stalls. As shown in Figure 18, using Arrow in-memory format avoids any overhead for cross-system and cross-platform communication overheads.

Figure 18. Arrow Format



Data Processing

This section discusses some of the data processing paradigms and patterns that are applied for data processing in the data lake. These include MapReduce, Lambda architecture and Kappa architecture.

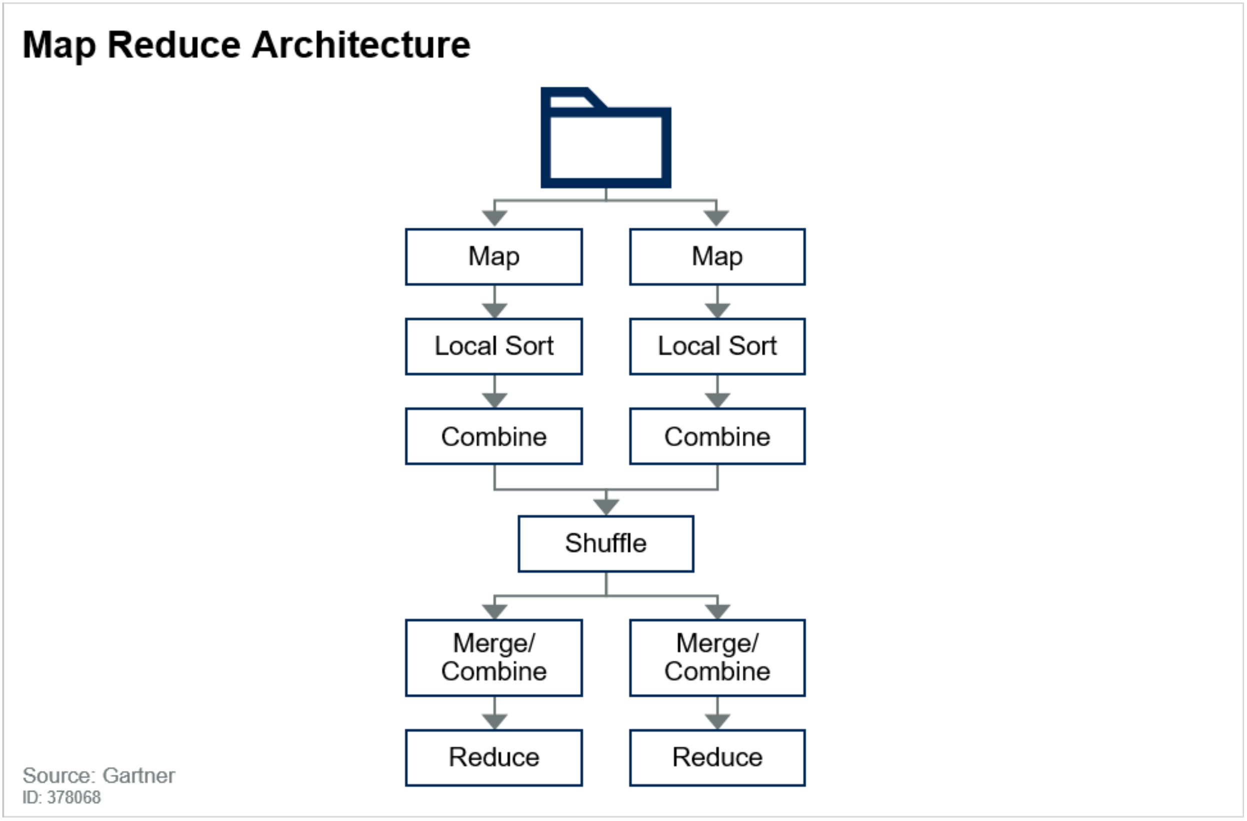
MapReduce

MapReduce is a framework that processes large-scale datasets by applying parallel and distributed algorithms on a distributed cluster. The primary methods of MapReduce are as follows:

- **Mappers:** Mappers read data from sources, apply transformations, and sort and partition the data again using data shuffle. Data shuffles are a fundamental part of distributed data processing and provide the underpinnings for doing operations like “joins,” “sorts” and “group by” on a distributed platform.
- **Reducers:** Reducers read data that has been processed by the mappers and shuffled, then they reprocess the data and apply operations like aggregations to generate final results.

A very high-level MR architecture is shown in Figure 19.

Figure 19. MapReduce Architecture



MapReduce has the following pain points:

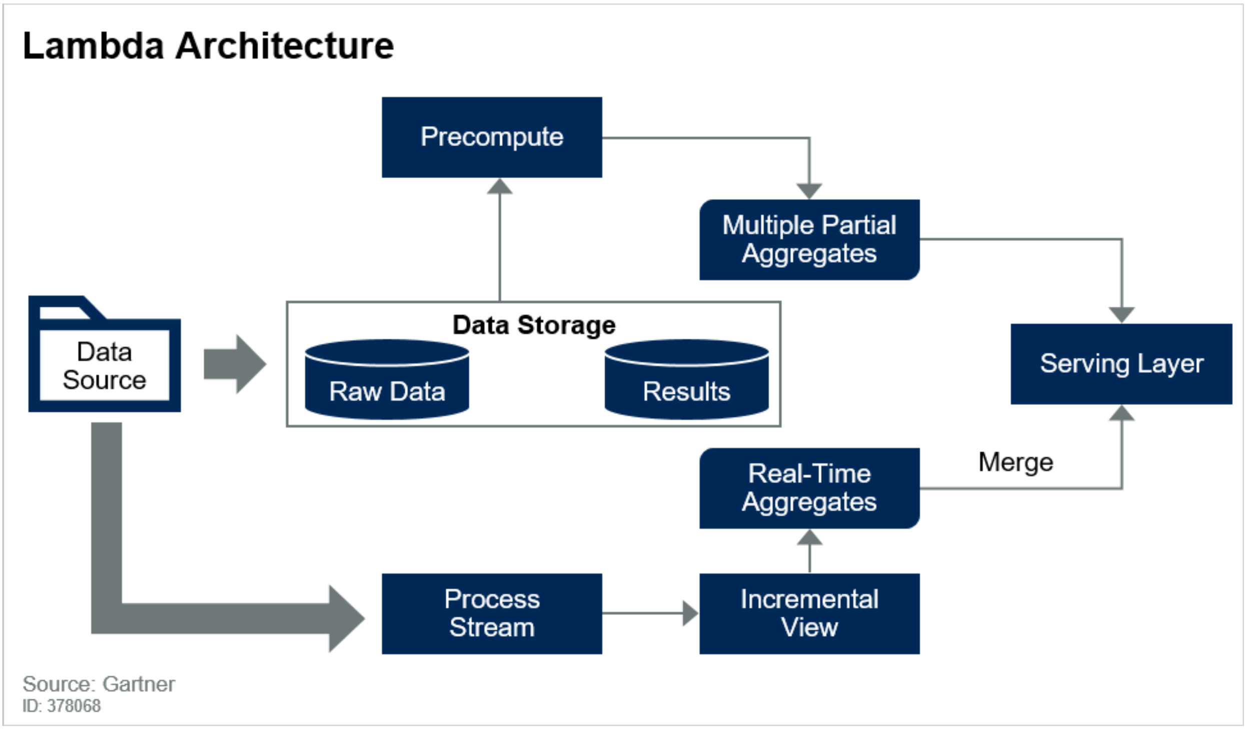
- Building complex pipelines requires a significant boilerplate, often separate program modules and multiple stages that are written to disk as files. This can be difficult to code and can result in large startup times due to JVM constraints.
- Writing intermediate results to disk between stages brings about huge performance and throughput bottlenecks.
- MapReduce architecture does not support building streaming data flows and pipelines.
- Exploratory analysis and data-discovery-based workloads incur significant delays due to slow disk reads and writes.

Lambda Architecture

Lambda architecture tries to mitigate the latencies of MapReduce, trying to balance latency, throughput and fault tolerance. The three primary layers are:

- **Batch layer:** This precomputes results using a distributed processing system output to the read-only data store and updates views by replacing the existing precomputed views. Data accuracy in the views is high with batch jobs (accuracy over latency).
- **Speed/real-time layer:** This processes data streams in real time, and the views are almost instantaneous, but they may have less data accuracy (latency over accuracy). However, those views can be updated later by batch methods (accuracy over latency).
- **Serving layer:** This stores outputs from the batch and speed layers to respond to ad hoc queries either by precomputed views or by new views from the processed data. A high level Lambda architecture is shown in Figure 20.

Figure 20. Lambda Architecture



Lambda architecture has the following pain points:

- The architecture requires coding and executing the same logic twice — once for the batch layer and once for the streaming layer, possibly on different frameworks and underlying engines. Due to so many moving pieces in the architecture, it is extremely complex to build, maintain, troubleshoot and debug. Keeping sync between these two layers incurs cost and effort, and this has to be handled in a careful, controlled manner. A very low code reuse across the layers is an inherent problem with this architecture.

- Building a Lambda-architecture-based data pipeline requires expertise in a number of technologies. Getting the all-encompassing skill sets can be extremely challenging for an organization.
- Implementing a Lambda architecture with open-source technologies and then deploying in the cloud can be troublesome. To avoid this, you could use cloud technologies to implement Lambda architecture, but by doing so, the enterprise automatically gets itself tied to a particular cloud provider, and that inherently is a disadvantage.
- Even though the architecture pattern has been around for quite some time, the tools are still immature and evolving. Cloud evolution has surely accelerated and innovated in this space, so it won't be long before mature solutions and tools appear in this space.
- The serving layer can be complex.

Kappa Architecture

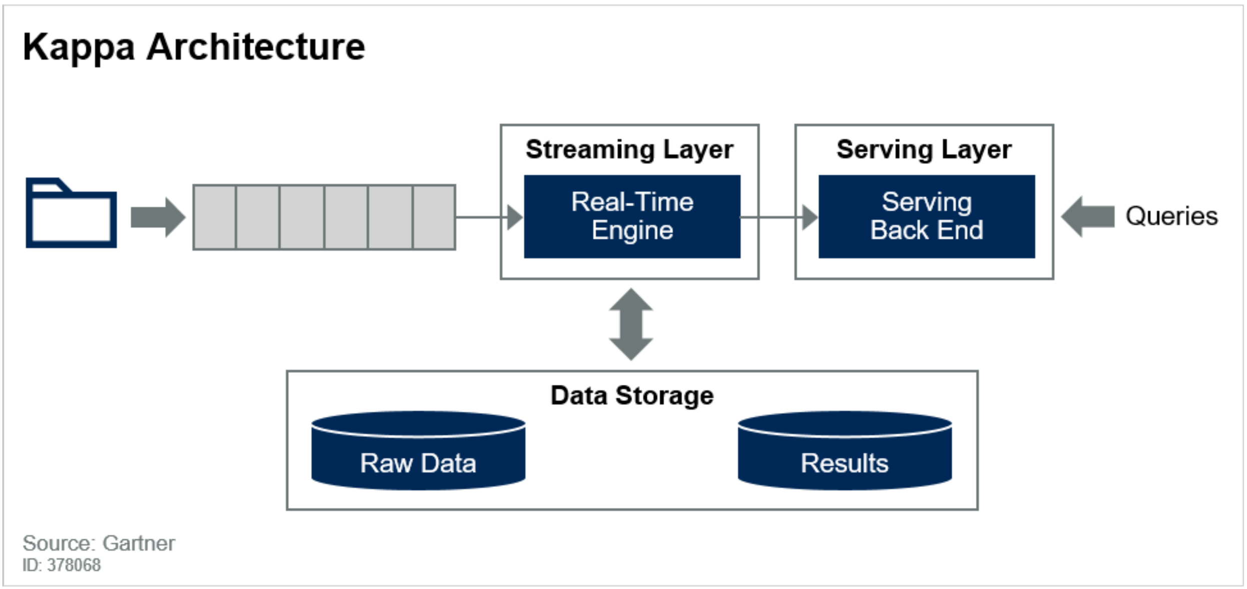
The Kappa architecture is based on the premise that batch is a special case of streaming. The batch layer results can be simulated using the streaming layer. Kappa architecture completely removes the batch layer and eliminates the need for the merging layer, thereby simplifying the overall architecture.

There are four underlying philosophies behind the Kappa architecture:

- **Everything is a stream:** Batching is a subset of stream, and hence, unifies everything as a stream.
- **Immutable data:** Raw data is stored and computations or views are derived. Because the raw data is stored and it never changes, the computations can be recomputed on demand.
- **Simplicity of the framework:** Kappa architecture relies on a single engine. There is no code duplication or working with multiple data pipelines within the same architecture and data flow.
- **Guarantee deterministic computation:** In the Kappa architecture, data processing can be rerun on the historical data, and the data pipeline must guarantee that the order of the events stays the same. This is the core to guaranteeing consistent results. For this reason, Kappa architecture requires a core component — a distributed messaging platform like Kafka — that provides event ordering and at-least-once delivery guarantees. Kafka can connect the output of one process. Apache Flink is a streaming data flow engine that provides the second component in the Kappa architecture the stream processing engine. Flink provides data distribution, interprocess communication, and fault tolerance for distributed streaming computations over event streams.

A high-level Kappa architecture is shown in Figure 21.

Figure 21. Kappa Architecture



Kappa architecture has the following pain points:

- Kappa architecture needs the messaging platform and the streaming platform — each of these components in itself are complex, both in terms of architectures and engineering.
- Advanced concepts — such as back-pressure handling, watermarking and windowing — need to be clearly understood, implemented and deployed.
- Building a Kappa-architecture-based data pipeline requires expertise across multiple products and frameworks.

For more details on data processing, see:

- “An Introduction to and Evaluation of Apache Spark for Big Data Architectures”
- “Enabling Streaming Architectures for Continuous Data and Events With Kafka”
- “Stream Processing: The New Data Processing Paradigm”
- This research will not be discussing SQL workload processing on data lakes and the associated products, tools and engines because this has been extensively discussed in “Selecting SQL Engines for Big Data Workloads.”

Data Governance

Data governance comprises data quality, data catalog, data security and privacy, data lineage tracking, and data life cycle management components. These components cut across all the layers of the data lake. This section explores the data quality, data catalog and data security aspects of data governance from an architectural lens.

Data Quality

For data democratization to be successful, organizations must ensure the data lake is filled with high-quality, well-governed data that is easy to find, easy to understand and easy to determine for quality and fitness. Some of the characteristics for ascertaining data quality:

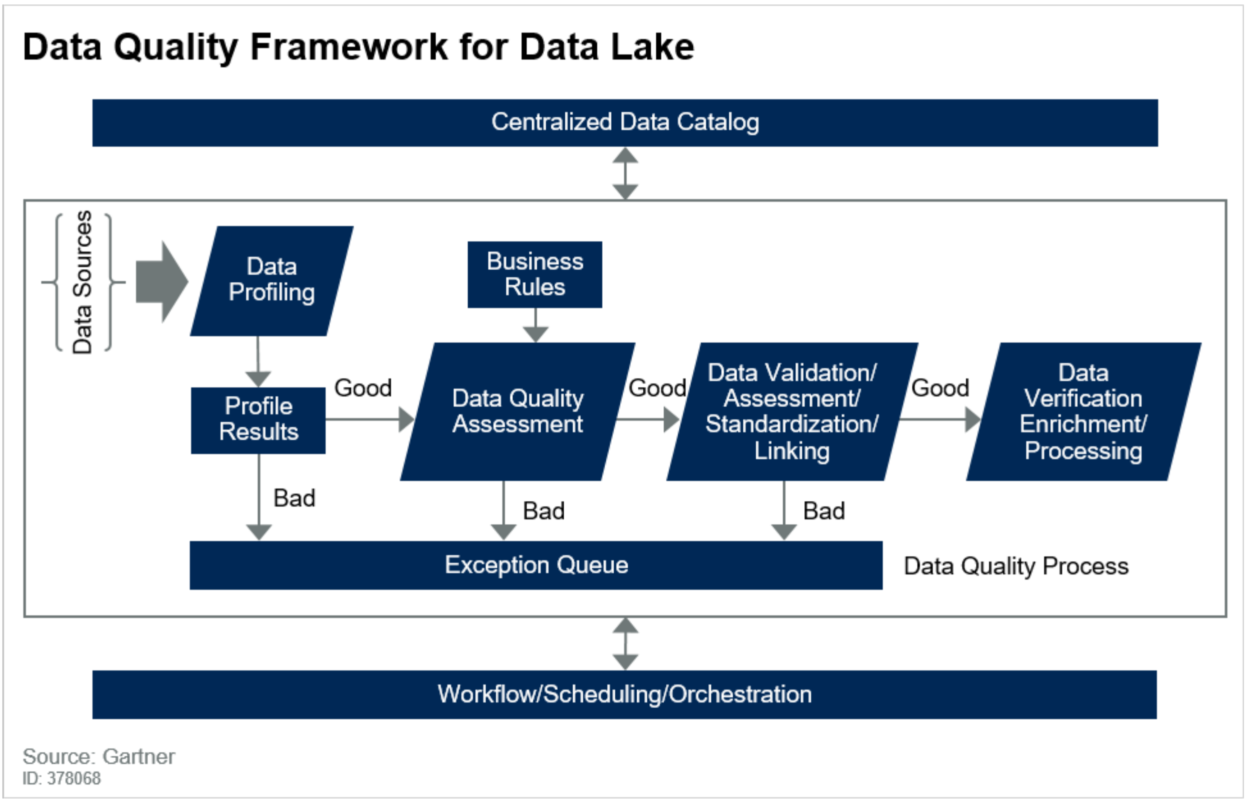
- Correctness/accuracy
- Completeness/coverage
- Consistency
- Timeliness
- Data lineage

Some common data quality issues that routinely arise in a data lake include:

- Embedded delimiters, where a value contains the delimiter that separates fields (e.g., commas)
- Corrupted records where an operational system may have inadvertently put control characters in values
- Data type mismatches, such as alphabetic characters in a numeric field
- Nonstandard representations of numbers and dates
- Multiple record types in a single file
- Mainframe file formats that use different character sets on legacy systems, which Hadoop does not know how to recognize or process

A high-level architecture of how a data quality system should be architected is shown in Figure 22.

Figure 22. Data Quality Framework for Data Lake



For more details on data quality, see:

- “Enabling Data Quality for Machine Learning and Artificial Intelligence”

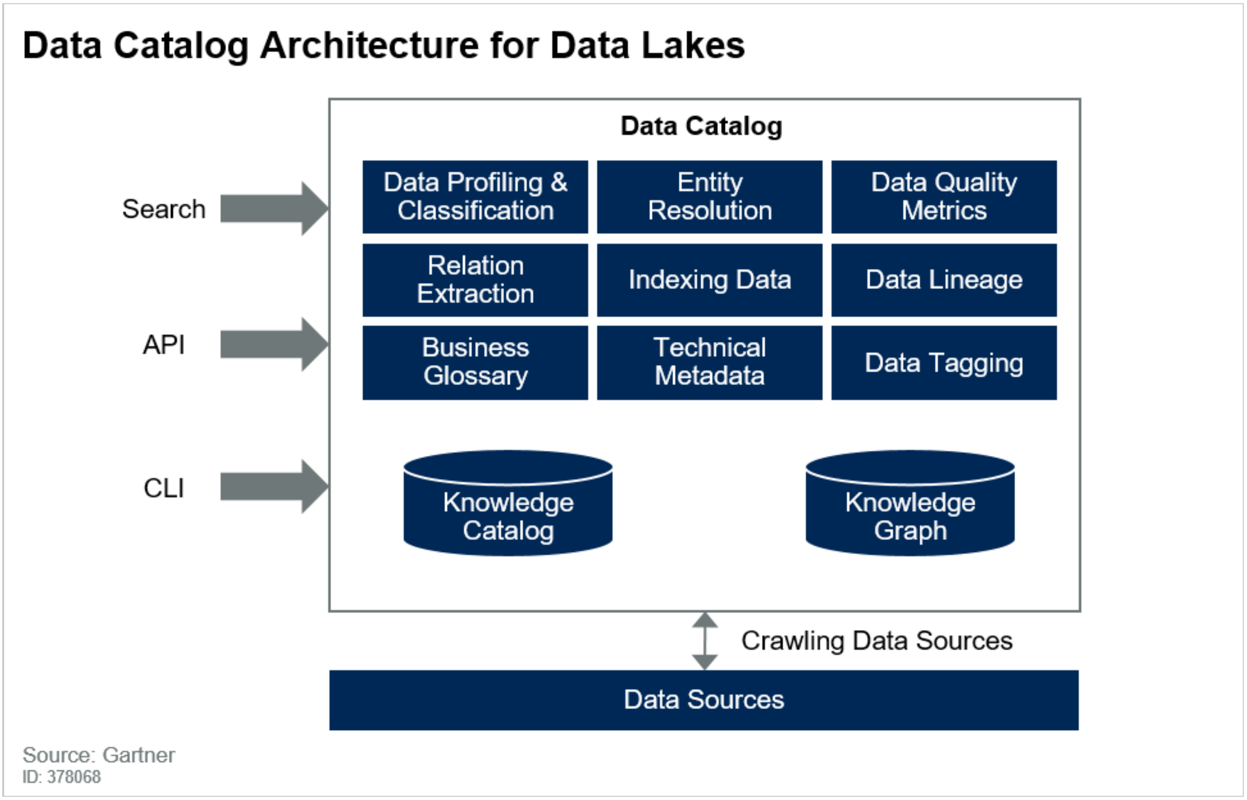
Data Catalog

For the data in the data lake to be used effectively with the downstream applications, the consumers need to understand the data and its structure, content and context. Various techniques can be applied on the data in the Raw Zone by the data catalog tools to identify semantic metadata and to enable efficient data discovery.

Data catalog tools should be able to extract entities from multistructured and unstructured data to build semantic metadata and to be able to identify relationships between entities. Additionally, data catalogs should also index the data to optimize and speed up data exploration and data discovery. Data catalog tools should leverage semantic technologies to identify relationships and standardize methods expressing relationships to improve visibility of the data and to reduce time to discover, integrate and analyze the data. Data catalog tools should identify latent topics from documents and then classify and annotate them based on the topics.

After data ingestion, data catalog tools crawl the datasets in the Transient Zone and Raw Zone to build the metadata. Enterprisewide data catalog tools that build a holistic enterprise catalog also crawl data sources outside a data lake and can also crawl to edge devices to gather technical and operational metadata. A high-level architecture of a data catalog is shown in Figure 23.

Figure 23. Data Catalog Architecture for Data Lakes



Some tools and vendors in the data catalog:

- Hortonworks Data Steward Studio
- Cloudera Navigator
- Infogix
- Waterline Data
- Oracle Enterprise Metadata Management
- Informatica Enterprise Data Catalog
- Collibra
- Alation
- Unifi

- Reltio
- IBM Watson Knowledge Catalog

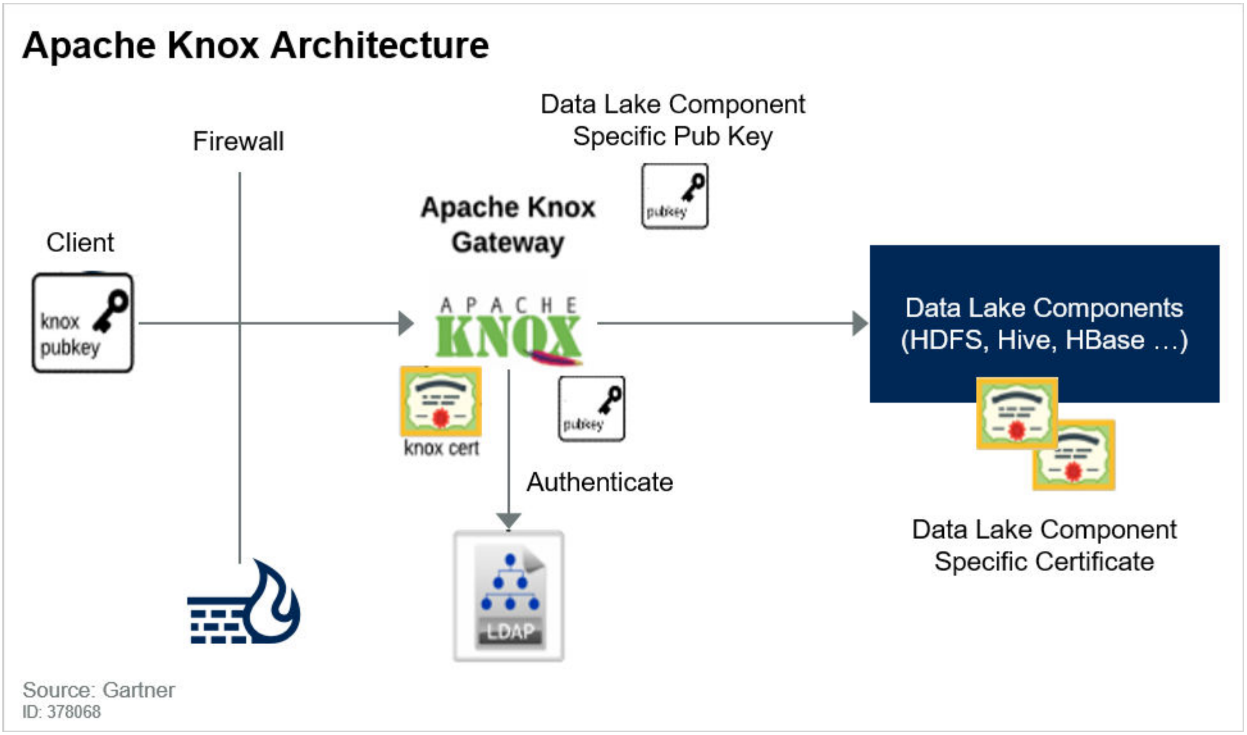
Security

This section details some of the commonly used products and provides an overview of their architecture to implement security across the different layers of the data lake.

Apache Knox is a reverse proxy that provides perimeter security to Hadoop clusters. It supports different policy enforcement like authentication, authorization and host mapping by chaining these as specified in a topology deployment descriptor.

Figure 24 shows how Apache Knox fits into the overall architecture of a data lake.

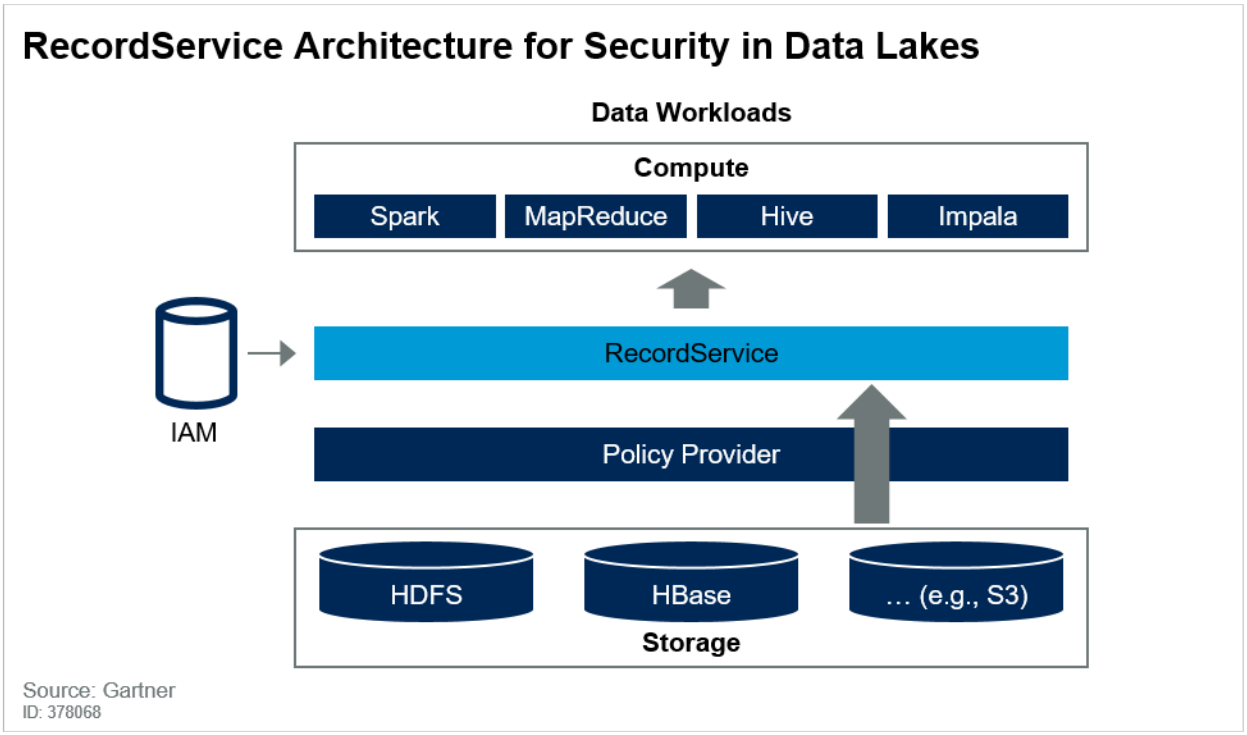
Figure 24. Apache Knox Architecture



As the data lake ecosystem grows, different products have different security implementations. Duplicate policies are often needed to provide the same user with a seamless access to different tools in the ecosystem. Cloudera's RecordService is a new security layer that sits between the storage managers and compute frameworks to provide a unified data access path, fine-grained data permissions and uniform enforcement across the stack to minimize duplication of policies.

Figure 25 shows how RecordService fits into the architecture.

Figure 25. RecordService Architecture for Security in Data Lakes

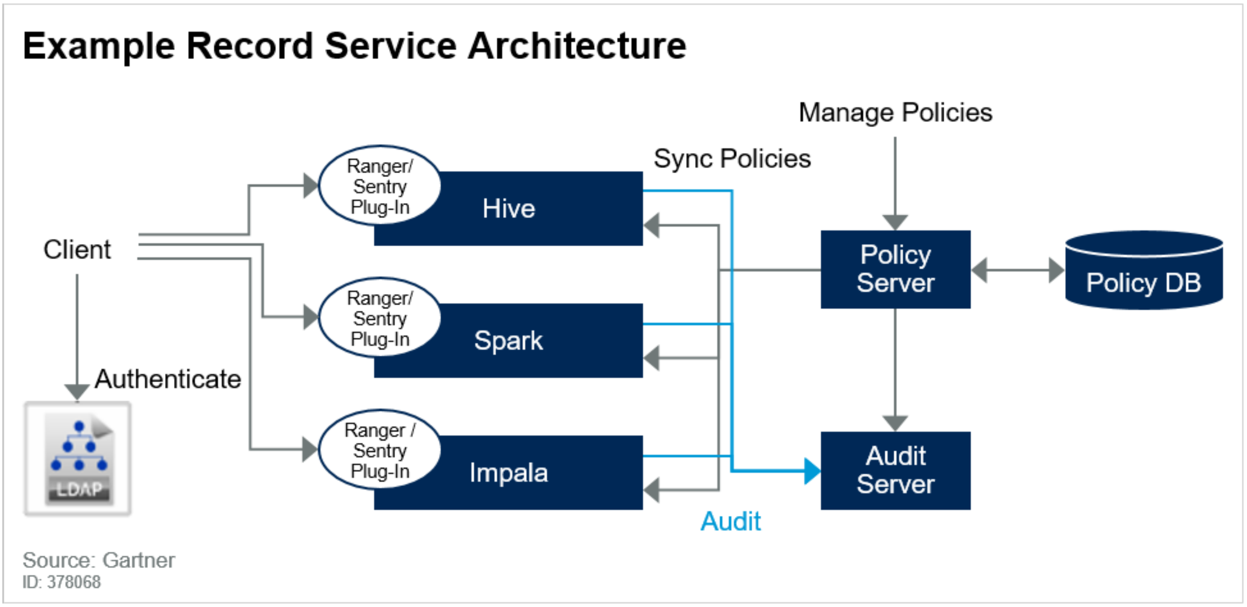


One example of a RecordService is Apache Ranger, which is a framework to enable, monitor and manage security across a Hadoop ecosystem. It provides authorization for a range of technologies in the ecosystem. It is based on attribute-based access control (ABAC). The Ranger plug-in is installed with the product — for that, authorization needs to be enforced. It synchronizes user data with enterprise directory (where user credentials are stored) and uses that to set up appropriate security policies.

When a user tries to access data for products for which the Ranger plug-in is installed, it retrieves the policies stored and does appropriate checks before users gain access to the data that they require.

Another example of a record service is Apache Sentry. Sentry is a system for enforcing fine-grained, role-based authorization to data and metadata. Sentry's plug-in is installed on any of the data processing technologies. Access to data is intercepted by the plug-in, and if it meets all the criteria defined in the policies, metadata access is allowed. Sentry's server manages authorization metadata. A high-level architecture of how Apache Ranger or Sentry integrates with the different components in a data lake is shown in Figure 26.

Figure 26. Example Record Service Architecture



Most components in the Hadoop ecosystem now include a built-in key management server (KMS) to secure the transport protocol over HTTP. It provides both client and server REST APIs for securing the communication channel. It is a Java application that includes support for the Java key store to hold multiple keys and an API to access and manage key metadata. It also includes access control list (ACL)-based access and support for multiple authentication and authorization protocols like Kerberos, Microsoft Active Directory and Lightweight Directory Access Protocol (LDAP) with Secure Sockets Layer (SSL). The Hadoop Key Management Server (KMS) includes end-to-end encryptions covering data at rest and in motion. Data written into HDFS is immediately encrypted using a specific algorithm and assigned a security zone.

Data lakes contain a large number of files, and setting permissions for each manually is not possible. Tag-based security in products like Cloudera Navigator and Ranger support tag-based policies. Instead of defining ACLs at a file level, these tools allow you to set up policies using tags by simply tagging files and folders instead of manually creating ACLs. Catalog tools allow you to set these tags, and they can get automatically applied by the policy-based access control tools.

This approach provides a powerful way to manage and organize the data without trying to shoehorn the organizational, hierarchy-based access rules into the file management structure. Tagging can be applied to data at the ingest layer and policies can be applied to restrict access to files.

Automation is the ideal solution for handling sensitive data and the access control management around it. Tools from vendors like Informatica and Waterline Data automatically scan ingested files and detect sensitive data with advanced ML algorithms and tag them. These tags are then used by the data catalog tools to enforce tag-based policies.

Vendors in the security space for data lake include Okera, Protegrity, Apache Metron, BigID, Dataguise, Privitar, Delphix (Data Masking), Vormetric and BlueTalon.

For more details, see "Securing the Big Data and Advanced Analytics Pipeline."

Gartner Recommended Reading

Some documents may not be available as part of your current Gartner subscription.

"Adopt the Logical Data Warehouse Architecture to Meet Your Modern Analytical Needs"

"Assessing the Optimal Data Stores for Modern Architectures"

"An Introduction to Graph Data Stores and Applicable Use Cases"

"Stream Processing: The New Data Processing Paradigm"

"Operationalizing Big Data Workloads"

"An Introduction to and Evaluation of Apache Spark for Big Data Architectures"

"Enabling Streaming Architectures for Continuous Data and Events With Kafka"

"Selecting SQL Engines for Big Data Workloads"

"Enabling Data Quality for Machine Learning and Artificial Intelligence"

"Securing the Big Data and Advanced Analytics Pipeline"

GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp