HOW TO BUILD AN ENTERPRISE DATA LAKE: IMPORTANT CONSIDERATIONS BEFORE JUMPING IN

Mark Madsen

Third Nature Inc.

This report is underwritten by SnapLogic, Inc.

Table of Contents

A Place For Data: The Idea (and Ideals) of the Data Lake

The concept of a data lake evolved to address changes and challenges in enterprise IT and new business needs. The concept is simple: the data lake comprises a single system in which to store and process all the data the enterprise wants to use for analysis.

One of the key principles behind a data lake is that you should be able to collect and store any data desired. The format, structure, speed of arrival, size, rate of growth, origin, quality and use are irrelevant. A request for new data means someone has an idea of how to get value from it, so we want to make it fast and easy to collect and store any data. These ideas won't always prove to be valuable, but this is consistent with a new requirement: a place where one can explore and experiment with data and find new applications for it.

Collecting data is cheap. Not having data when you need it is expensive. Therefore a core principle of the lake is that you should collect as much data as you can successfully manage. A properly designed data lake should give you the infrastructure to do this.

> *"Collecting data is cheap. Not having data when you need it is expensive. Therefore a core principle of the lake is that you should collect as much data as you can successfully manage. A properly designed data lake should give you the infrastructure to do this."*

Because the data lake combines inexpensive storage with scalable parallel processing, it isn't only a place to store data. It's a place to work with data too. Data usually needs to be filtered, restructured, cleaned or at least reformatted, even for basic analysis. The lake provides the capability to do these things as well as to perform more complex processing associated with analytics.

There are different views on the purpose of a data lake, and therefore how it should be built. One way to separate these views is based on whether the applications in the lake are the business, or are supporting and enabling the business. To date, most discussions about data lakes frame them as integral to business operations, as they can be for many digital companies. The more common but less discussed need is to enable new practices that use data to support the business. This paper addresses the support and enablement roles.

There is confusion between the concept of the data lake as an application versus as infrastructure. An application is purpose-built to support specific requirements. Infrastructure is designed to support the needs of many different applications, so it generalizes some services while making others the responsibility of the applications. The focus of this paper is on building a data lake to serve as infrastructure for multiple applications and diverse uses.

Why Build a Lake When We Have a Perfectly Good Data Warehouse?

The idea behind the data lake isn't new. A centralized store for all the data in an organization is the same concept behind a data warehouse, but a warehouse is not a lake - just as a lake is not a warehouse. A data warehouse is built on several principles:

- There is a global data model designed to store all data that will be used
- It contains clean, consistent data
- All cleaning, reformatting and integrating of data is done prior to loading it
- The data is read-only and can't be altered
- All data resides in a relational database and the ideal means to access it is via SQL (or MDX) queries, either tool- or human-generated

These principles have endured for more than 25 years. They reflect the context in which the data warehouse was first developed as well as the period in which it subsequently evolved. Data was difficult to access, so only IT professionals could get to it. Compute and storage were expensive. There were only a handful of systems inside the organization that contained all the data. End-user tools were scarce. The rate of change in business and IT was low compared to today.

All those constraints have changed. Compute and storage are inexpensive. Many types of analysis tools are available. Data can be found everywhere, inside the organization, in cloud-based applications, public sources, and open data sources that anyone can access.

Most of this data never makes it into the data warehouse, nor should it. It takes a long time to assess, model, source, clean and load data into a warehouse. All data must go through these processes before it can be made available, thus a key reason to have a data lake: making data available without doing unnecessary work.

All warehouse data must be complete, of uniform quality and stored in tables, even if it will only be used once. It's like funneling all your data through a keyhole before it can be used. These requirements put such a large burden on the warehouse developers that there is always a backlog of work that doesn't get done, or the work takes so long that the business problem is solved in a different, less effective way.

If designed properly, the data lake strikes a balance between doing work in advance and deferring it until later. Sometimes you want to do the work in advance (as the warehouse requires) because the data will be used and reused often and by different people. Sometimes you want to defer the work because you aren't sure how the data will be used or whether the data has value. Instead of always doing all of the work up front, a data lake allows you to choose when to do it and how much to do, speeding the availability of data.

Possibly the most common reason to build a data lake is the need for analysis. The data warehouse was never intended to support the read-write workloads of today's analytics projects. The canonical warehouse definition begins with "…a read-only repository…" yet much analysis work involves integrating, preparing or deriving new data. A data lake is designed to support this work.

The Naïve Lake

The naive approach to a data lake is to collect and store all the data in its original format in a Hadoop cluster, then let analysts make sense of the data when they need it.

> *"The naive approach to a data lake is to collect and store all the data in its original format in a Hadoop cluster, then let analysts make sense of the data when they need it."*

This approach won't work, first and foremost because it doesn't scale organizationally. (It imposes performance and scalability penalties too, but these are not as important). Simply loading data provides little useful information. Everyone has to make sense of and integrate the data they want to use, so exploration and preparation of the same data is repeated by each person and use.

This is fine the first time someone analyzes data, but the approach results in little to no minimal reusability. Every time another person wants data they have to go to the lake and do all the work of finding, understanding and processing it, even if this involves replicating the work of others before them. Most organizations have little tolerance for redundant work, particularly when – owing to the complexity of preparing data for analysis – any duplication of effort can produce different answers to the same question.

Misconceptions About Data and Its Use

The naive approach to building a lake by simply storing raw data in the format the source application provides is based on several misconceptions:

- This approach assumes that people have the skills to find and process the data they need. Most analysts aren't developers. They don't want to or are not capable of getting, cleaning and shaping their own data. The technical nature of preparing and making sense of data re-creates the original scenario that led to data warehouses — the IT department becomes the gatekeeper for data.
- Creating a repository of raw data assumes that people want to spend time making sense of data. A data scientist may want to, but most business analysts don't. They want specific data in order to get their jobs done, and they want to be able to get it as quickly as possible.
- The approach ignores the realities of data use. One-off analysis is one type of use. After understanding a problem, an analyst may want to create an automated process to deliver that data. Doing this work directly from raw data can be inefficient and slow, particularly when several different processes are all using the same data. Some of these processes can be more important than others; all need to be managed and prioritized. End user tools support analysis but not the transition to operational processes. More is needed for this.

- The simple approach treats all data as if it is the same. Data isn't like water — uniform and therefore mixed together in the lake. Each set of data is unique, with it's own structure, format, quality and lineage. Different techniques are required to collect, catalog and store it so that it can be found and used. Building a data lake by simply putting all data in one place works about as well as enabling collaboration by storing everyone's documents on a file share. It's fine at first but after a while quality is suspect and nobody can find what they need.

The naive approach to building a data lake is not workable. You can't just load raw data and hope for the best. You can't expect people to focus on technical data preparation skills so they can do work that is peripheral to their jobs, unless their job is analysis. You can't install Hadoop and rely on it to solve all of these problems. A different approach is required.

The Architected Lake

The task of building a data lake is more difficult than building a system that supports a single use, such as data exploration or only running data pipelines for applications. There is a balance between doing all the processing of data before it can be used, as with a data warehouse, and none, as with a raw data dump into Hadoop.

When a particular set of data is reused often or the use of that data mandates strict quality control, doing the work in advance may be worth the time and effort required. When the value of data is unknown or reuse is low, it may be best to defer any work on the data.

The data lake must support both schema-on-write and schema-on-read approaches because it has proven to be impossible to anticipate and model all of an enterprise's data in advance. Therefore a new architecture is required, one that does not have the limitations of a data warehouse or a simple Hadoop installation.

There are three distinct needs a data lake must support: acquiring data, processing data, and delivering data for use. Rather than taking a monolithic approach, you should treat each of these needs as an independent design that is built on a shared platform, as shown in figure 1.

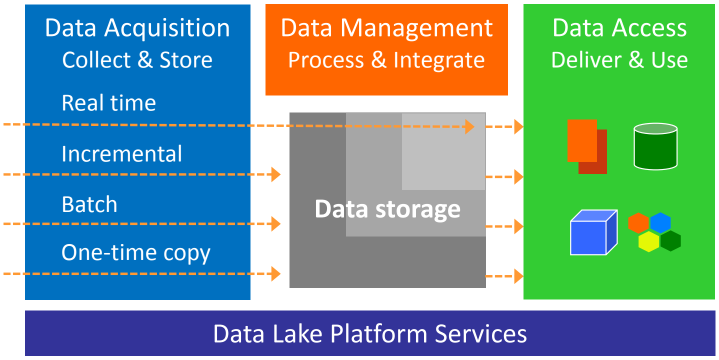

Figure 1: Data lake components. The acquisition component allows any data to be collected at any latency. The management component allows some data to be standardized and integrated. The access component provides access at any latency and via any means an application chooses. Processing can be done to any data at any time from any area.

Acquiring data should be independent from data management because you don't want to standardize and model data prematurely. Instead, you want to make collecting and storing new data as easy as possible so that it can be made available quickly.

Separating the acquisition from management allows you to collect data at one time and process it at another, as well as permitting different processes to use the data. This suggests that you should build the data acquisition subsystem with an immutable data store for the raw data. By doing so, you retain all of the original data and can store all of your changed or derived data separately. This permits easy reprocessing of any derived data, from data structure changes or cleaning rules to algorithmic output.

The advantage of an immutable layer is that you can collect data at any latency, from real-time streams to bulk file loads. All the extra work a relational database does to ensure consistency is not required because the data is recorded as-is and never changes. An immutable data store lets you write data as fast as it arrives without requiring a predefined schema. It likewise lets you do as little or as much processing as you want before writing the data.

In similar fashion, data access should be independent of both acquisition and management. Some people want to access raw data and should not be prevented from doing so. Some want standardized, structured data, while others want to process their own data. Humans aren't the only consumers of data - the lake must support applications that independently control processing of data.

This separation of data consumption from collection or management can only be accomplished by decoupling the part of the lake that supports access from the parts that support collection and management.

The data warehouse is a monolithic architecture designed around one primary workload: querying data. It merges data management and data access into one global data model. It provides processing only from sources into the data model, and makes the model immutable where it can only be accessed via SQL.

The data model imposes a common structure on all data; good for control, bad for adaptability. The problem is that different uses of data often require different data structures and models, so there can't be a restriction of one global data model at the data consumption layer.

The three-way separation of functions in a lake means repeatable use of data can be supported when repeatability is desired, rather than imposed on everyone up front. The separation also enables broader use across the organization because there is flexibility in access, from raw data that has not been changed to data that is standardized to data that is cleaned and structured for a specific use.

The secret to designing a data lake is knowing when to maintain control over data and when to relax it. Some data should be stored in raw form only. Other data should be standardized. Some data must be tightly controlled. The approach to doing this requires both a technical architecture and a data architecture, the two components that together define a data lake.

Reference Architecture

A reference architecture is abstract. It represents only the important design considerations and does not unnecessarily constrain implementation decisions. This means you don't use it as a blueprint that dictates technology choices. You must map your needs to the architecture and define the blueprint for implementation decisions.

To do this we must first define a set of principles that will govern what we design. These principles allow us to evolve an architecture suited to our needs rather than the accidental architectures that often result from a technology-driven approach. This evolutionary design is what we want a data lake to support because the uses of data by an organization change over time.

Instead of trying to model all data in advance, we should collect it. Instead of trying to control or prevent change, we should accept it as normal and design to accommodate change. Instead of trying to control what people can do with data, we should focus on enabling many uses but also make the resulting data and its use visible so it can be managed when necessary.

An architecture is more than code and components. It's a combination of three things: technology and data and process.

Technology Architecture

You don't buy a data lake, you build one. Building and maintaining a data lake is more complex than you might think. Installing Hadoop doesn't give you a data lake any more than installing a database gives you a data warehouse. These are technologies that provide a base platform to build on.

> *"Installing Hadoop doesn't give you a data lake any more than installing a database gives you a data warehouse."*

A lake is not just a passive repository. It's also a platform that provides data processing capabilities unavailable elsewhere. The high level tasks the lake needs to support include the ability to:

- enable collection of data at any latency, from streams to once-only file loads
- acquire data by passively receiving it or by actively retrieving it from other systems
- store any data desired, whether it is rows of numbers, human-authored text or images
- centralize datasets and their access with a minimum of configuration or modeling
- transform and integrate datasets into new structure or forms
- perform processing to filter, clean, structure or integrate data at multiple speeds and latencies
- run arbitrary algorithms against data, without constraining the type of data used or style computation
- publish officially curated data
- share data created or curated by individuals
- manage and archive collected, created and curated data
- act as a production platform for data pipelines and applications that requirement data processing or computation

- enable access and use of data beyond what is possible in SQL-only systems today

You need a technology architecture to define the required services or capabilities that support these tasks. As mentioned earlier, your specific tasks determine which services you build and which you can omit. These in turn define the implementation blueprint.

Core Services in a Data Lake

The core set of technology components you will most likely need in a data lake are shown in figure 2. The accompanying list defines the components in more detail. Ideally, you want all of them, but it takes time to build out the platform and evolve it to fit your requirements.

In reality, this is how the data warehouse was built. In the early days there were no blueprints or products. Developers built everything in code, from data extraction to the user interfaces, often omitting components that are now deemed vital such as BI tools and metadata.

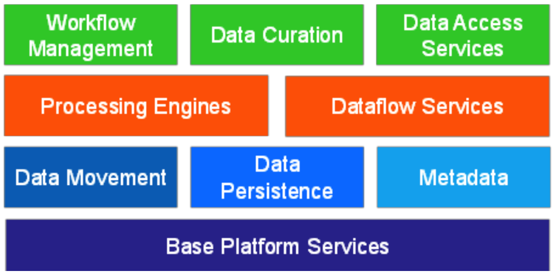

Figure 2: The main categories of services that make up the functionality of a data lake.

Data persistence. We use the term “persistence” because this is more than simple storage of data. Developers, tools and applications need access to higher level services that persist data for different durations, from long-term raw data storage to short-lived output caching.

Many descriptions of the data lake say that the data is immutable. This is not entirely correct. When data first enters the lake and is persisted, that data should be immutable for as long as it is retained. This allows you to reconstruct any downstream (processed) data at any time.

The downside of an immutable repository is that most data needs some processing before it can be used. Every time such data is accessed it must be re-processed, a potentially slow and inefficient task, in both machine resources and human labor.

This means other read-write persistence services will be required. There may be a need to write data at high volume streaming rates. There may be a need for high speed get-set operations of objects or documents. There may be a need for relational storage.

Data movement. There are different types of data movement. Data can be pushed to the lake or it can be pulled from elsewhere by the lake. Data can be collected from streams, services, files or databases.

The movement of data is bidirectional. The lake will probably need to publish data to other systems so the same inbound mechanisms must be available for outbound use. This means the lake will need to provide connectors of different types for different sources and targets.

Both collection and publishing may be one-time, periodic batch or streaming. The lake needs to be able to control these, throttling or speeding up as required by a particular task.

Data movement also happens within the platform. For example, a job run in one engine may provide input to another job that uses a different engine and file format. Ideally, the platform should provide services that hide as much of this work from the developer or user as possible.

Data access. The access services are distinct from the data movement services. They exist to allow outside tools or applications to access data stored in the lake, regardless of the format or type of persistence. There are many ways to access data, from whole sets to individual objects. File transfers, APIs, SQL queries, even search are all possible mechanisms for accessing data stored in the lake.

Different applications will have different requirements even though they may access the same underlying data. It's also possible that the same data may be persisted in more than one place, for example in Avro as files and in a relational tables. Although data may reside in multiple places, access to it should be transparent, as though it were centralized. SQL-on-Hadoop options, data virtualization and query federation are examples of technologies that can support many of these access requirements.

Processing engines. There are many different engines for different types of workloads because no single engine solves all problems equally. Some engines are batch, some are parallel pipeline, some are stream. There are engines that support data transformation, relational algebra, linear algebra, graph computation or other types of processing. Some engines support more than one type.

Many are available on Hadoop, for example MapReduce for batch, Spark or Tez for parallel pipelines, Presto, Impala or Hawk for SQL, Hive for batch-like query and transformation. The choice is determined by the applications that you want to support.

The advantage of Hadoop is that engines are easy to add. This is also a disadvantage because each has its own storage requirements and resource needs. When designing, it is important to try to keep the number of engines to a minimum to avoid unnecessary complexity and data duplication in the environment.

Dataflow tools. Data isn't used only in its raw form. Different applications will reformat, filter, clean, integrate, derive and compute data. Data must often be integrated across multiple sources and formats. Sometimes the data is continuously processed, other times it is processed periodically.

To date, this work has mostly been done by developers using languages not designed for dataflow problems, with the data pipelines constructed to run on a single engine. Dataflow tools abstract the problem of building pipelines so the developer can spend more time on the valuable work (the logic of the dataflow) and less on the redundant infrastructure, such as engine details and scaffolding.

Scheduling and workflow management. This is a necessary aspect of any processing beyond the simple persisting of data. Most data pipelines are composed of multiple dataflows in multiple jobs. These need to be coordinated, possibly across different execution engines, for example processing data in batch which is then transferred to a BSP (graph) engine that runs calculations and pushes the data to a SQL-accessible table. Scheduling and workflow tools allow developers to:

- Define and resolve scheduling priorities for jobs within and across workflows

- Manage dependencies in tasks, between jobs, on datasets and on events
- Test data pipelines on production data without causing production problems
- Debug multi-task or multi-job workflows (which often generate multiple different logs)
- Deploy workflows, particularly when they involve multiple tools and engines
- Detect failures at task, job, workflow and application levels
- Define recovery, restart and cleanup procedures after problems
- Monitor workflows to know what worked and what didn't, what met assertions and what didn't

Metadata. All data loaded into the lake should have metadata recorded with it, at both the dataset or source level and the individual element level. It's important to know when data arrived, what system it came from and what format it arrived in.

The lake must keep track of the schema implicit in each dataset or that data will become impossible to find over time. Despite the lack of a predefined data model, schema metadata is still important. The definition of the structure and format should be recorded with a dataset. This can sometimes be captured, for example from JSON or Avro-formatted data or from the table definition in a remote database. Sometimes it must be specified by a person.

A data lake is both a destination for data and a source of data as it runs workflows, so similar structure and lineage metadata should be generated for any data that is created from processing within the lake.

Likewise, usage metadata should be captured so any data accessed within the lake can be tracked. This is useful for everything from tracing dependencies to tracking the use of sensitive data.

The lake requires a metadata repository to store all of this information, as well as to track the history of changes to datasets over time. This isn't a trivial task because metadata services must be layered over the repository to support many different tools.

In order to access data, users and developers need the ability to find the data they are interested in. This is partly a technical problem solved through metadata indexing and access services. User interfaces to find data are also required, both for accessing data and for curating it.

Data curation. Self-service analysis is a myth everyone likes to hold onto. It only works in a curated environment. Unlike a data warehouse, the data lake contains many independent datasets. Some of these come from outside, while others are created within the platform. Some are created by developers, others by end users.

Metadata provides a mechanism for tracking all of these things, but people still have to manage them. Datasets need to be labeled and indexed, definitions written, data conflicts resolved, unused or obsolete data may need to be archived, and retention policies set for data that has legal restrictions. These are sometimes an IT responsibility, sometimes an end user's. Interfaces and underlying services are required to support these activities.

Platform services. Storage management, process coordination, resource management, encryption, authentication, basic monitoring, logging and security are all requirements. These are services that should be provided by the underlying platform the data lake is built on. If they don't exist then they also need to be built.

This list of components or subsystems describes what the data lake should provide at a high level. However, this is not sufficient. Data is what you want to use, not code. Without an appropriate data architecture, you will build the equivalent of a data swamp, albeit with nice code.

The Heart of the Lake: Data Architecture

The heart of the data lake is the data architecture that the platform and tools implement. It defines the arrangement of data to enable easy capture, easy use and easy reuse to avoid redundant processing and user effort.

> *"The data lake needs a data architecture that is not limiting the way a global warehouse schema is."*

The data lake needs a data architecture that is not limiting the way a global warehouse schema is. The lake needs to deal with change more easily and at scale. It should not impose requirements and models up front that prevent data collection. It should not limit the format or structure of data to only tables. It should be read-write rather than read-only, and provide both locations for shared data and for private or unpublished data.

The four zones for storing data in the lake

The data architecture is related to the component architecture shown in figure 1. It provides places to support the three major functions of the lake: data acquisition, data management and data use.

The goal is to isolate components that change at different rates from one another. They change at different rates due to different requirements. Data sources often change independently of downstream systems like the lake. Decoupling the storage and models used when collecting data from those used to standardize or deliver data makes it easier to adapt to change. Therefore there should be three discrete areas for storing data that match these functional components of the lake. The different data storage areas in the architecture are shown in figure 3.

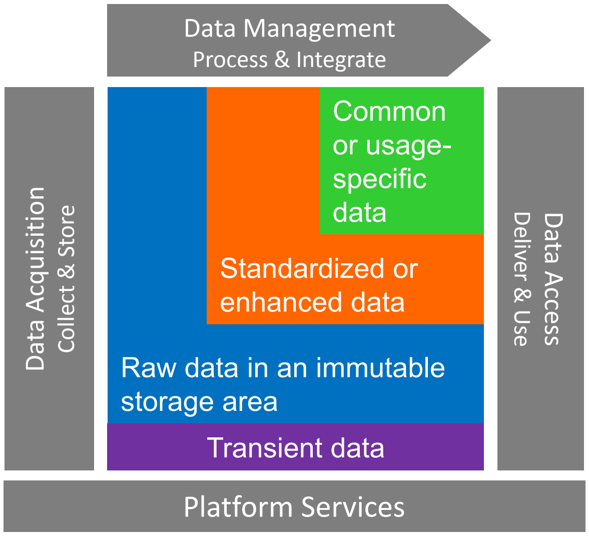

Figure 3: This shows the data architecture. There is one immutable storage area for collecting data, a second area where some datasets have been standardized to make them easier to integrate or use, a third area where data is structured for specific uses, and a working area for transient data that doesn't need to be retained.

The collection zone for raw data. This zone is the initial collection area. When new data requests arrive they should be done quickly so data is available in a timely fashion. This implies that there should be little to no data modeling or processing required.

The lake is schema-less in the sense that nobody defines or models all the data in advance before it is available. The lake is built to house many different datasets that are not all pre-integrated. Each dataset does have a schema, in the sense that any data structure has an implicit schema. The lake simply stores and keeps track of these datasets and schemas, hence the need for a metadata repository that is accessible by all applications.

By recording data in its original format and structure, the problem of data collection can be decoupled from the problem of managing data. Often, analysts want access to data to determine if it is useful. If it is, and use of it is repeatable, then cleaning and standardizing is worth the effort. Many times it isn't, or the data is infrequently used.

This area of the lake should be immutable; when data arrives it is written as-is with no attempt to clean or standardize it. Standardizing fields within or across datasets is the reason for the next zone

The management zone for standardized, enhanced and derived data. Some data is more valuable or is used more frequently than other data. When this is the case, the data can be standardized or cleaned and made available in another area of the lake. The purpose is to parse this data, making it more accessible and shareable.

Any datasets with common elements should have those elements standardized, for example a phone number will always be formatted the same way, or money is always recorded in a decimal format with a currency attached.

There are two goals. One is to represent common data in canonical formats to remove redundant, possibly conflicting rules or errors that lead to confusion. The second goal is to standardize identifiers so that integration across datasets is easier. As a side effect, the work analysts or users must do is reduced, similar to the way it is reduced when schemas are imposed in a data warehouse. Unlike that environment, the data lake does this selectively to datasets and to fields within them.

The delivery zone for common or usage-specific data. When data is common across a large part of the organization it is placed into this part of the lake. For example, financial transactions are core data that should be in a single, common format with standard names. This is usually a small portion of the total data available, but its use is widespread.

Data that is structured for a specific purpose also belongs in this area, for example, placing data into a particular file format for a graph database so it can be accessed by specialized tools, or building a file extract to be retrieved by Tableau.

Any application with specific needs can use the processing features of the lake to format, transform and publish the data it needs. Each application owns its own finalized data. The work done to get that data should be globally visible so the lineage of derived datasets can be traced.

The transient zone for short-lived or one-time data. One of the challenges users face is finding a place to work with data. People need a place that lets them work on and save data without affecting others. Sometimes the need is simple, such as loading an external file. Sometimes it's complex, such as preparing data for a machine learning model. This use is often ignored, or assumed to be something only developers need.

Using different zones enables agility. Using different areas to support separate functions of the lake makes it easier and faster to make data available. As use or value of specific data increases, time can be invested to make that data reusable or impose more control. In this way, many different applications can be supported on shared infrastructure. This can be done without using one global schema and the shortcomings that go with it.

While simple on the surface, this data architecture requires work to maintain. Data doesn't move itself. It takes work to transform and standardize data, to enhance and derive data with algorithms, and to put data into consumable formats for different uses. Rules are needed for when data should be placed in different areas.

The work is not isolated to the developers. While collecting data and making it available is largely their responsibility, deciding when to standardize data or integrate datasets is both a developer and user responsibility. The task of defining quality rules and data standards falls more with users. Enhancing data or placing subsets into usage-specific formats is more an analyst's responsibility.

Conclusion

A data lake is more than a simple dumping ground for data. It has a data architecture and a technology architecture. The architecture is built around several design principles:

> ***Change isolation:*** grouping features or classes of data that change frequently into different areas from those that seldom change. This minimizes the impact of rapidly changing components on more stable parts of the system.
>
> ***Services, not monoliths:*** A data warehouse is built on the idea of functional design with functions separated into layers. The layering means that functionality is hidden when looking across layers. This leads to a tightly coupled system that is resistant to change, the opposite of adaptable. The lake is designed with the assumption that services or data may be needed in different areas at different times, making the system more composable.
>
> ***Data architecture is as important as technology architecture:*** The data architecture is a mechanism for change isolation and decoupling. Some data and uses change frequently, others are more stable, so the architecture should manage the data in the same way one manages change

isolation in code. Data architecture also constrains what is possible, so design the data architecture to avoid the overly-controlled, less agile global schema approach.

You must use principles to guide your design. There are no simple specifications or patterns to follow because every organization has different needs, and much of what you want to support those needs requires custom construction.

The data lake enables a different design process for data. Instead of modeling data, then collecting it, then analyzing it, data can be collected and explored, but modeled only when it is worth the effort to do so. This is what should drive the internal data architecture; not waterfall design but evolutionary design.

An evolutionary design process makes data usable more quickly. A faster and more flexible process for making data usable allows the organization to try different things, experiment and evolve more quickly.

How Snaplogic Fits Within a Data Lake

SnapLogic is the only unified data and application integration platform as a service (iPaaS). The SnapLogic Elastic Integration Platform has 300+ pre-built connectors – called Snaps - to connect everything from AWS Redshift to Zuora and a streaming architecture that supports real-time, event-based and low-latency enterprise integration requirements plus the high volume, variety and velocity of big data integration in the same easy-to-use interface. SnapLogic's distributed, web-oriented architecture is a natural fit for consuming and moving large data sets residing on premises, in the cloud, or both and delivering them to and from the data lake.

The SnapLogic Elastic Integration Platform provides many of the core services of a data lake, including workflow management, dataflow, data movement, and metadata.

More specifically, SnapLogic accelerates development of a modern data lake through:

- Data acquisition: collecting and integrating data from multiple sources. SnapLogic goes beyond developer tools such as Sqoop and Flume with a cloud-based visual pipeline designer, and pre-built connectors for 300+ structured and unstructured data sources, enterprise applications and APIs.
- Data transformation: adding information and transforming data. Minimize the manual tasks associated with data shaping and make data scientists and analysts more efficient. SnapLogic includes Snaps for tasks such as transformations, joins and unions without scripting.

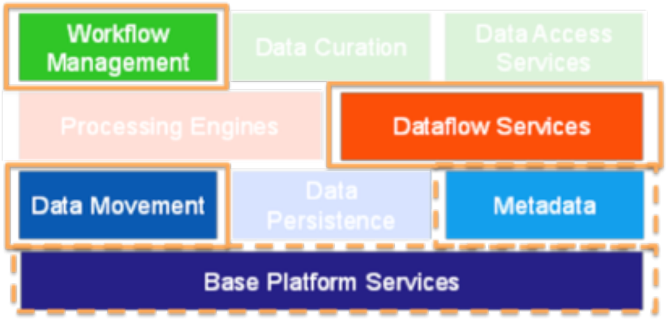

Figure 4: SnapLogic provides many of the core data lake services defined in figure 2, including workflow management, dataflow services, data movement, and some metadata and base platform services.

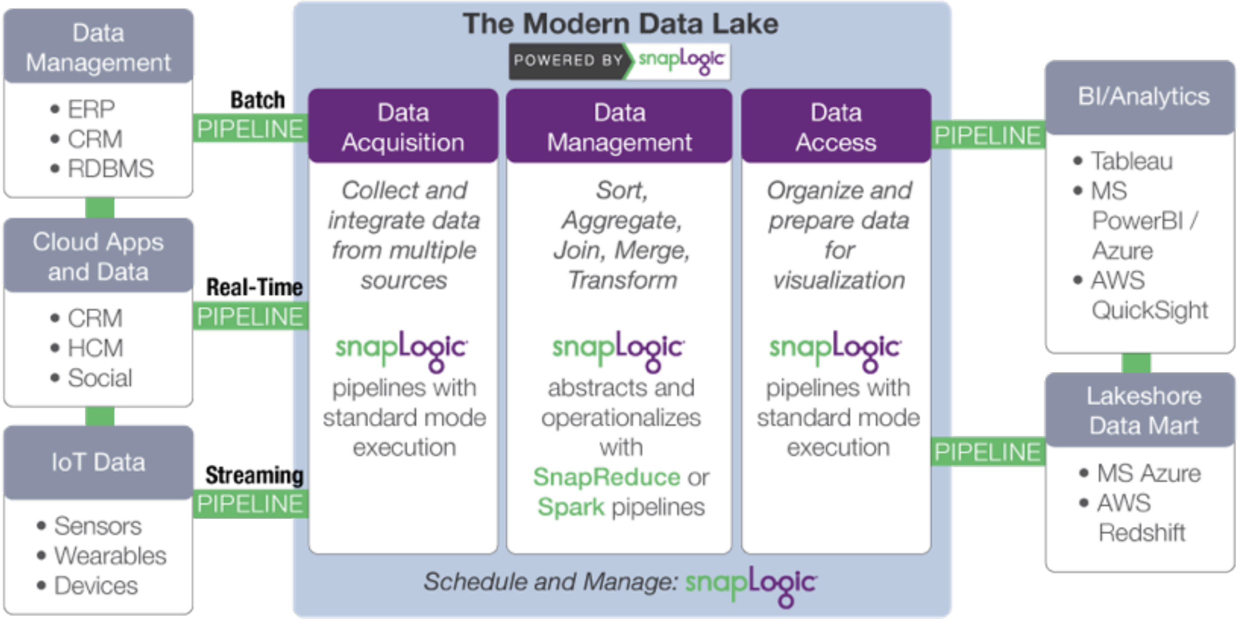

Figure 5: SnapLogic accelerates the development of an enterprise data lake by automating data acquisition, transformation and access.

- Data access: organizing and preparing data for delivery and visualization. Make data processed on Hadoop or Spark easily available to off-cluster applications and data stores such as statistical packages and business intelligence tools.

SnapLogic's platform-agnostic approach decouples data processing specification from execution. As data volume or latency requirements change, the same pipeline can be used just by changing the target data platform. SnapLogic's SnapReduce enables SnapLogic to run natively on Hadoop as a YARN-managed resource that elastically scales out to power big data analytics, while the Spark Snap helps users create Spark-based data pipelines ideally suited for memory-intensive, iterative processes. Whether MapReduce, Spark or other big data processing framework, SnapLogic allows customers to adapt to evolving data lake requirements without locking into a specific framework.

About SnapLogic

SnapLogic is the industry's first unified data and application integration platform as a service (iPaaS). The SnapLogic Elastic Integration Platform enables enterprises to connect to any source, at any speed, anywhere — whether on premises, in the cloud or in hybrid environments. The easy-to-use platform empowers self-service integrators, eliminates information silos, and provides a smooth onramp to big data. Founded by data industry veteran Gaurav Dhillon and backed by leading venture investors, including Andreessen Horowitz and Ignition Partners, SnapLogic is helping companies across the Global 2000 to connect faster. Learn more about SnapLogic for big data integration at *www.SnapLogic.com/bigdata*.

About Third Nature

Third Nature is a research and consulting firm focused on new and emerging technology and practices in analytics, information strategy and data management. Our goal is to help organizations solve problems using data. We offer education, consulting and research services to support business and IT organizations.