Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 06, June-2023

Data Lake Implementation

Diwakar Manna
Associate Principal – Architecture, Data & Analytics,
LTIMindtree,
Peninslyvania, USA.

Abstract: Data Lake implementation: Data acquisition approaches and considerations. Natural or man-made, I find lakes beautiful, peaceful, and re-energizing places. It must be the relaxing effect of water! The water journey to a lake is a different story, though: rivers, or sometimes creeks, may be steep, narrow, with lots of twists, or, more likely, a combination of those characteristics.

Introduction: Data lakes, like those containing water, can also provide you that relaxation effect: knowing that you have all your data available, in the same place, ready to be consumed in whichever way you may need. The analogy extends to establishing the ways for your data to get to the lake as well: the path between a given data source and the lake may be a well-built canal, a river, or perhaps just a small creek.

Let's cover some aspects of the water journey to the lake.

1 DATA ACQUISITION INTERFACES INTO THE DATA LAKE

An interface defines the boundary between two objects. In this case, the two objects are content repositories. First, the source of data: the water spring or the snowpack. Second, the target data repository: the lake itself.

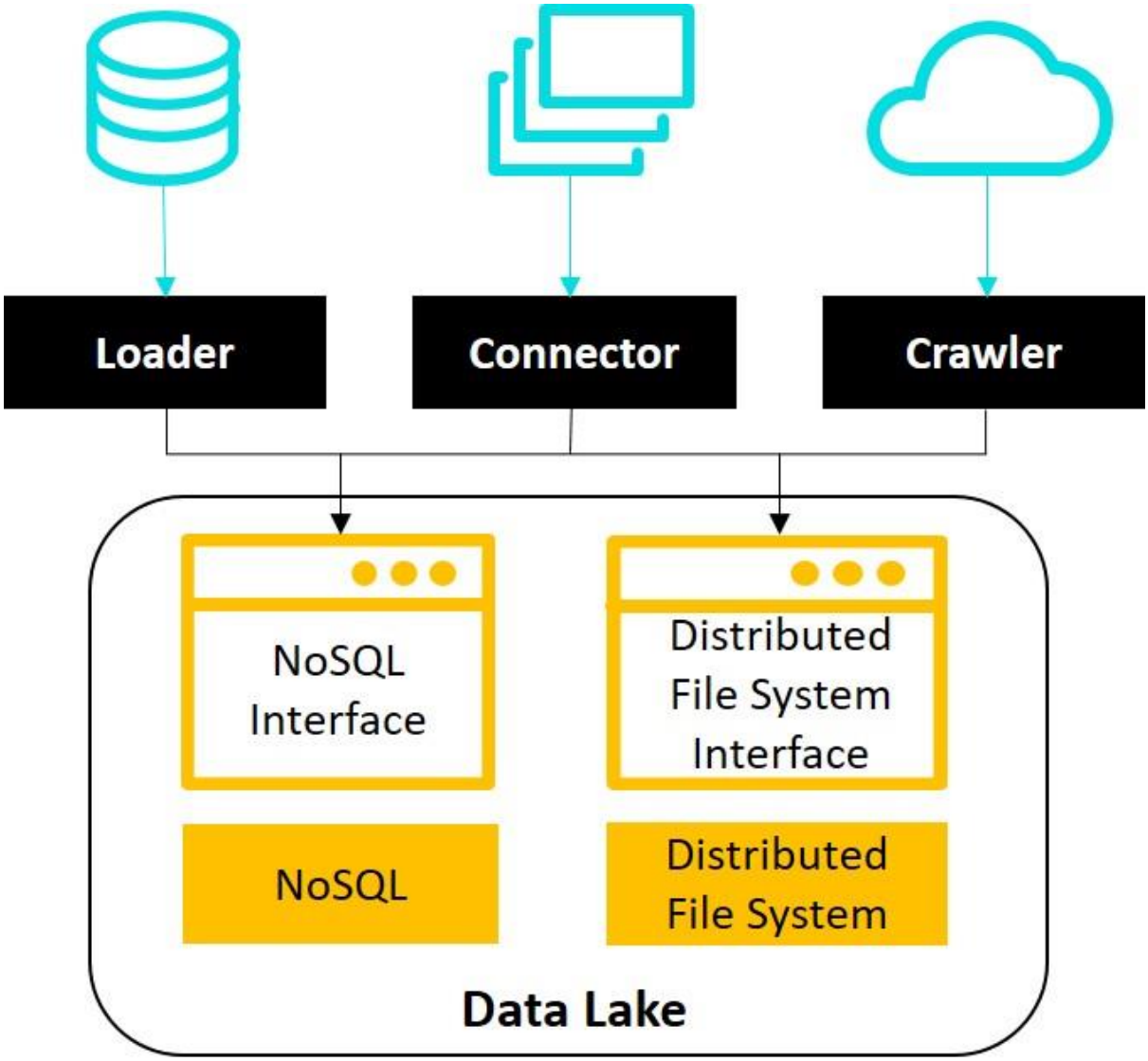
Each data repository has its own interface, or boundary, to hold the data in. A part of our job in a data lake implementation is to provide effective mechanisms for the data to be copied from one repository to the other.

The mechanisms for connecting two repositories typically implement two interfaces:

- On one side are the interfaces that read or accept the data from the content sources. The interface may vary between one source and the next.
- The second interface is on the target side, to write or send the data to the lake.

In the diagram below, the data sources are shown at the top and the data lake at the bottom. I've used different common names for the mechanisms to integrate the two. To prevent confusion, I prefer to choose one name and use it across all content sources in an application. In the rest of this document, I'll stick to the term **"crawler."**

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 06, June-2023



Data lake content ingestion workflow

It is common to limit the crawler function to getting the data as-is. This simplifies the integration between the repositories and minimizes the time it takes to copy data over. This means that crawlers only consist of the 'E' and the 'L' of the famous **ETL** acronym: **"Extract"** from the source, and **"Load"** to the data lake. Your data lake should have much more capacity to process data once it is inside of the lake than while it is just flowing into it. Therefore, we'll leave the **"Transformation"** part of the ETL acronym as a separate subject for another blog.

2 DATA ACQUISITION LIMITATIONS

The data may flow like water in a river or slowly, in little amounts, like in a creek, depending on the interfaces, capacity, and connectivity between the source and the lake. Some limitations are imposed on the content source side, which you probably can do little-to-nothing about. Here are examples of possible constraints in interfacing with the content sources:

- An old system, with limited integration support.
- An old system, only capable of running on an old, specific version of an operating system.
- Limited capacity in memory and CPU in the machine(s) hosting the content source.
- Limited network bandwidth to the content source.
- Limited capacity for integration so that active users of the source system are not affected by crawling activity; or

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 06, June-2023

- An API that produces multiple events or messages for each action in the source system, all of which must be processed to identify those that are relevant to the crawling needs.

Other limitations may come from the implementation of your crawlers or the data lake itself. You likely have control over these, though you may need to work with other internal departments or create projects to improve the platform, reconfigure portions of the architecture, or even redesign or reimplement some of it. Here are some examples of changes that may be necessary to enhance the acquisition of content into the data lake:

- Implement multi-threading in the crawler.
- Increase crawler machine resources, network bandwidth, storage throughput capacity, or other similar infrastructure limitations.
- Move crawler or interface components to a more quickly scalable platform, such as cloud-based service offerings from AWS, Google Cloud, Microsoft Azure, or similar platforms.
- Change APIs used in passing content so that data is compressed during transferring, files are uploaded in multiple parts in parallel, or other similar performance enhancement techniques; or
- Re-engineer the current bottleneck components that prevent your system from being highly performant or highly available.

3 DATA ACQUISITION IMPLEMENTATION OPTIONS

Knowing about the limitations that may exist in integrating your repositories is important as it helps you evaluate implementation options. Some sources, particularly the older ones, may not offer more than a single interface for integration. If you do have options, further understand the constraints and features offered by each option before you choose one over the others.

Let's first briefly describe the possibilities for implementation of interfaces on either side:

- **Application Programming Interfaces (API)** - Modern repositories tend to have available APIs used by the software itself and published for others to integrate their own software products. This doesn't necessarily mean that the APIs would do everything that you need for your specific integration. This is a preferred approach as future updates to the software would likely respect the API. You get the benefits of source system upgrades without breaking your crawler, or with minimal changes to it.
- **Software Development Kits (SDK)** - An SDK is a set of software tools that can be used for multiple needs around the specific software that the SDK was created for. It is a do-it-yourself approach that is supported by the software manufacturer. It is likely a preferred alternative for crawler integration when an API is not available or doesn't fully satisfy your requirements.
- **Direct** - By direct, I mean that there is no tool or API from the software vendor for the integration. Still, you may have access to the database(s) or the underlying files used by the software. Let's assume the software uses a database. Through database drivers, like Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC), the crawler could be implemented to retrieve the desired data. In addition to writing the connectivity from scratch, you may need to first infer the relationships in the data store. This is particularly difficult if no documentation is available from the software manufacturer.
- **Third-party crawlers** - There are various products and frameworks out there that can be used to acquire data from various popular sources since interfacing with data sources is not a new requirement. For example, through many client engagements, we've developed a range of connectors that can efficiently support data lake content acquisition and security requirements. A third-party crawler might have been implemented using a direct approach, except that you only need to configure it rather than building it yourself (though some further customization may still be necessary!). There may still be limitations on these software tools for your specific needs. A disadvantage may be that you would not be using the original software's integration mechanism if one is available. But a possible advantage of a third-party crawler is that it can be updated sooner than your in-house developed version, if you had one, because the third-party developer may need to make it available sooner for its other clients.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 06, June-2023

- **A combination of the approaches above** - Some content sources may have enough API support for some of your integration use cases but not all. When an integration use case is not supported by the source API or its SDK, you may need to develop or integrate a direct-based integration. For example, assume that you can get metadata and files via a source API but not the permissions associated with them. Further assume that you could get those permissions directly from the source's database via a direct integration. This means that you would need a crawler that uses a combination of mechanisms to get the data.

4 Which approach works for me?

"So how do I know which approach(es) to select?" – you may ask.

The preferred approach would be to integrate your crawler via the source system's provided API or SDK, when either is available. Furthermore, a third-party crawler that uses those API or SDK mechanisms may be a better choice when it is well-supported and documented. Not only would you save time in developing your own integration with the API or SDK but you'd also benefit of the testing and features developed to support a larger set of use cases by that third party's clients.

A crawler-based on the direct implementation mechanism may break if the source software is updated in the future. You may not have implemented all the relationships between the objects in your first versions of the crawler. The source system stability or performance may be affected by the crawler.

The direct integration approach is the least recommended one, as it is not supported by the provider of the content source. Yet, sometimes, this may be your only option. Here are just a few considerations in case you are faced with an integration in which this crawling approach is required:

- How are the data repository objects organized?
- How are the objects related to one another?
- How does data change: in what order, under what conditions, etc.?
- Are there flags to identify active/enabled data? Or snapshot id values or similar for latest data?
- Are there versions of the data? How do you select the appropriate one to read?
- Are there status values that you should use to retrieve the appropriate data?
- Are there periods of time during which the crawler may not read data? (to prevent effects on performance for users of the source system or to allow for source data processes to complete so that the crawler reads up-to-date information)
- In what format is the data stored? Any conversions necessary to read the data or portions of it?

5 DATA ACQUISITION REQUIREMENTS

Every lake and its surroundings are different from other lakes. We've covered some of the differences in the acquisition options: the data lake surrounding crawler implementation options. It's time to go over requirements for the crawlers. Although you may need different crawlers for different types of data, the same crawler requirements likely apply to most, if not all, of your crawlers – regardless of the crawler option you choose to implement!

Clearly defining those requirements will help you evaluate the options and choose the most appropriate one for each of your data source acquisition needs. You'd need to discard, adjust, or extend from the sample subset of requirements below as applicable to your own situation:

- Use a service account for the crawler. You may call it a faceless or system account. Do not use any one person's account as that person may move onto some other adventures in the future.
- The crawler account must have access to all the data to be crawled.
- The crawler account must have read-only access.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 06, June-2023

- Crawling can occur in parallel, either through multi-threading or by deploying multiple instances of the crawler working on different portions of the data at the same time.
- The crawler must have a feature to perform a full-crawl (bulk load) to acquire all data from the source. This is necessary to fully copy the data over to the lake when it is first on-boarded. It is likely that you'd need to run this more than once after that initial load. Some possible reasons for re-crawling all content are:

    - You may have missed some metadata fields the first time around.
    - There are changes to most or all of the source content: a new metadata field; content was re-organized; etc.
    - A bug fix to the crawler that requires re-loading all content.
    - Switching to a new data lake storage that requires re-loading all content from the source.

- The crawler must be able to retrieve frequent data changes to ensure the data lake is in sync with the content source. These delta updates are often referred to as incremental crawls. Depending on the source system, some field (like a timestamp, status, or flag column in a table), some event (like an alert or notification) or other mechanisms (like Change Tokens in SharePoint) exists to identify changes in the source. In other cases, your crawler may need to maintain its own snapshot mechanism (often the case for a web crawler) to detect when content changed at the source.

- The crawler must store the data as-is. Data normalization, cleansing, and enhancements must be deferred to subsequent processing within the data lake.
- The crawler must store the source field names as-is. This is particularly important during the initial implementation or while troubleshooting data differences. It would allow you to quickly correlate the data to the source.
- The crawler must be able to get all required metadata from the content source.
- When available, the crawler must be able to retrieve files stored in the content source. Some content sources store different file formats that you may need to copy over to the data lake along with the metadata of the file or the object it is associated with.
- The crawler must be able to get the Access Control Lists (ACLs) in the content source. This may be used for ensuring that access to content in the data lake (or one of its client applications) is restricted to only those users with access to the content at the source.
- The crawler must be able to get user-to-groups associations at the source. Often, access to the source is granted to groups rather than specific users. Some sources may not use the concept of groups but that of roles, folders, entities, or something else for data security. The crawler must be able to retrieve all relevant associations if the same data security is to be enforced in the data lake or its clients.
- Choose or build a data lake content acquisition framework, whenever possible. This would allow you to re-use common components across your multiple crawlers: logging, error handling, configuration, uploading to the data lake, etc.

6 MAINTAINING THE LAKE AND ITS FEEDERS

Like their water counterparts, data lakes should be a live ecosystem. As sources change, so do the lake and the flows that feed it. Overflows may occur on the way to the lake or within the lake itself, both of which must be dealt with or prevented, if possible. Eventually, new crawlers would be implemented for loading additional content sources into the data lake. Sometimes, sources may go dry. The data would likely stay in the lake, though the crawlers should be decommissioned, and resources reallocated or scaled down. There will be different difficulties and unexpected turns in connecting sources to your lake, even with the same software on which two or more deployments are made (for different business units or other reasons). Still, each integration would be better than the prior one, particularly when you can implement data acquisition with a crawler framework.

AUTHOR BIBLIOGRAPHY:

Diwakar Manna has completed **Masters of Engineering in Computer Science and Engineering** at Madras University. Has over 20 years of experience in **Architecture, Solution Design and Delivery of Enterprise Data Integration, Information Management and Business Intelligence solutions** for global customers in various roles such as **Data Architect, Solution Architect and Data Integration**. Architected and solutioned the aggregated, integrated computing environment made up of clinical, operational data content, and external data housed in an industry-standard clinical hosted environment. End to end implementation expertise in the areas of Data Integration (ETL/ELT), Information Management covering Master Data Management, data architecture, and data modeling.