Data Lake Architecture -

A Comprehensive Design Document

Medical Data Processing Company

# Tracker

## Revision, Sign off Sheet and Key Contacts

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 03/11/2023 | QUANG HUY CHU | 0.1 | Initial submission |

## Reviewers / Approval

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
| FirstName LastName | 1.0 | Udacity Reviewer  Enterprise Data Lake Architect |  |

## Key Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Role | Team | email |
| QUANG HUY CHU | Data Architect | Medical Data Processing | cqhofsns@gmail.com |

# Purpose

In this document, I have outlined and explained in detail the Data Lake Architecture design for Medical Data Processing Company.

This architecture has the goal of solving the problem of data overload in both storing and processing data that the company is facing. Moreover, by using the Data Lake Architecture instead of the current RDBMS SQL server architecture, we hope to be able to provide the company with more new insights such as real-time data processing or discover new insights by applying Machine Learning that company are planing.

This document is specially designed for technical audiences who have a high level of technical knowledge. So, if you don't need to learn about technical designs or don't have the technical knowledge, please have a look at the document "DataLakeExecutivePresentation.pptx" attached in the same folder.

# Requirements

In the problem statement, we acknowledged that the company facing many problems, such as:

1. Data storage and processing is not as expected because the amount of data is and will be so large that the current system cannot meet the resource needs even the customer upgraded their hardware to maximum. This lead to system overload and slow ETL processes.
2. There is no fault tolerance since the system is just a single-node SQL server and can not handle situation like system failure or crashed.
3. The company currently can only run batch processing jobs at night due to the compute capacity limitation. This affects not only the flexibility of the system but also makes the system unable to meet the desired needs (tracking patient health, bed availability, etc.)

Also, the customer shared with us their system’s current specification, the current data amount, flow rate, and expected growth as follows:

**2.1. Customer existing technical environment (Single SQL server node):**

* 1 Master SQL DB Server
* 1 Stage SQL DB Server
  + 64 core vCPU
  + 512 GB RAM
  + 12 TB disk space (70% full, ~8.4 TB)
  + 70+ ETL jobs running to manage over 100 tables
* 3 other smaller servers for Data Ingestion (FTP Server, data and API extract agents)
* Series of web and application servers (32 GB RAM Each, 16 core vCPU)

**2.2. Current data volume:**

* Data coming from over 8000 facilities
* 99% zip files size ranges from 20 KB to 1.5 MB
* Edge cases - some large zip files are as large as 40 MB
* Each zip files when unzipped will provide either CSV, TXT, XML records
* In case of XML zip files, each zip file can contain anywhere from 20-300 individual XML files, each XML file with one record
* Average zip files per day: 77,000
* Average data files per day: 15,000,000
* Average zip files per hour: 3500
* Average data files per hour: 700,000
* Data Volume Growth rate: 15-20% YoY

With the above specifications as well as the problems encountered, the customer also provided business and technical requirements to guide us in the design of the new system:

**2.3. Customer business requirement:**

* Improve uptime of overall system
* Reduce latency of SQL queries and reports
* System should be reliable and fault tolerant
* Architecture should scale as data volume and velocity increases
* Improve business agility and speed of innovation through automation and ability to experiment with new frameworks
* Embrace open source tools, avoid proprietary solutions which can lead to vendor lock-in
* Metadata driven design - a set of common scripts should be used to process different types of incoming data sets rather than building custom scripts to process each type of data source.
* Centrally store all of the enterprise data and enable easy access

**2.4. Customer technical requirement:**

* Ability to process incoming files on the fly (instead of nightly batch loads today)
* Separate the metadata, data and compute/processing layers
* Ability to keep unlimited historical data
* Ability to scale up processing speed with increase in data volume
* System should sustain small number of individual node failures without any downtime
* Ability to perform change data capture (CDC), UPSERT support on a certain number of tables
* Ability to drive multiple use cases from same dataset, without the need to move the data or extract the data
  + Ability to integrate with different ML frameworks such as TensorFlow
  + Ability to create dashboards using tools such as PowerBI, Tableau, or Microstrategy
  + Generate daily, weekly, nightly reports using scripts or SQL
* Ad-hoc data analytics, interactive querying capability using SQL

# Data Lake Architecture design principles

To design the Data Lake, first we need to analyze some considerable design principles based on both customer’s technical and business requirement:

1. **Embrace open source tools:** This is the most important requirement in my subjective opinion. Customers ask to use open-source tools to avoid using services and software controlled by third parties, which can also imply that it is difficult to access cloud-based services with this requirement. Since the customer did not specifically say which open-source tools, I recommend that we use the applications and services of the Apache organization, specifically here their Hadoop Ecosystem.
2. **The system should has capability to handle, process, and deliver Big Data :** The customer's data source in my opinion is being evaluated as Big Data because it is meeting the 5V standard. Therefore, the new system needs to be able to handle a large amount of data from many sources continuously, but at the same time, it can still ensure the accuracy of each type of data. Apache Hadoop ecosystem has provided us many tools to perform with this requirement.
3. **Single point of failure is not allowed:** In the old (current) system, when experiencing data overload or system failure, a single SQL server node will immediately shut down and affect other services. To avoid the single point of failure, in the new system, we should consider using distributed computing/storage. This has three advantages:

1. Solve expensive hardware problems with only commodity hardware.

2. Accelerate ETL or even provide real-time data analytics. A good example of distributed computing is Apache Spark.

3. Distributed Storage like HDFS provides durability and availability to prevent data loss.

1. **System can provide new insight and new services:** Since the data in this project includes medical data, real-time data monitoring is paramount for end users to help them make accurate and timely decisions. In addition, using a large amount of historical data is also essential in serving needs such as reporting or new insight discovery using Machine Learning. Therefore, this new system needs to be designed to ensure that all three requirements are met.

# Assumptions

With the designed Data Lake architecture, there are some assumptions to be considered:

* Since the customer did not specifically say which open-source tools, Apache Hadoop Ecosystem’s services and application is recommended.
* Customer can lower their technical environment to save cost with compatible hardware, but should not lower than the tools requirement. For example, Apache Spark requires 8 GB or more for each machine.
* According to the “Current Architecture” description, the main source for the Data Lake is coming from two sources: Files or FTP server, and API from EMR sensors. These two sources will be the main source that will ingest the data into this new Data Lake system. But in the initial step, there will be the third temporary source, which is the SQL server. Apache Nifi also can handle migrating data from the SQL server to the new system.
* This Data Lake mainly used for BI approach but still customer can use create specific ETL scripts using Python, Scala, Java, R on Apache Spark to create data set for Machine Learning, other Business Analyst purposes.
* Instead of using web-based portal to create dashboard or report, this Data Lake is designed to allow customer to create more interactive dashboard using BI tools such as Tableau, Power BI, Looker, or Microstrategy.

# Data Lake Architecture for Medical Data Processing Company

From the requirement and many considerations mentioned above. I have design a Data Lake system architecture for this use case, as follows:

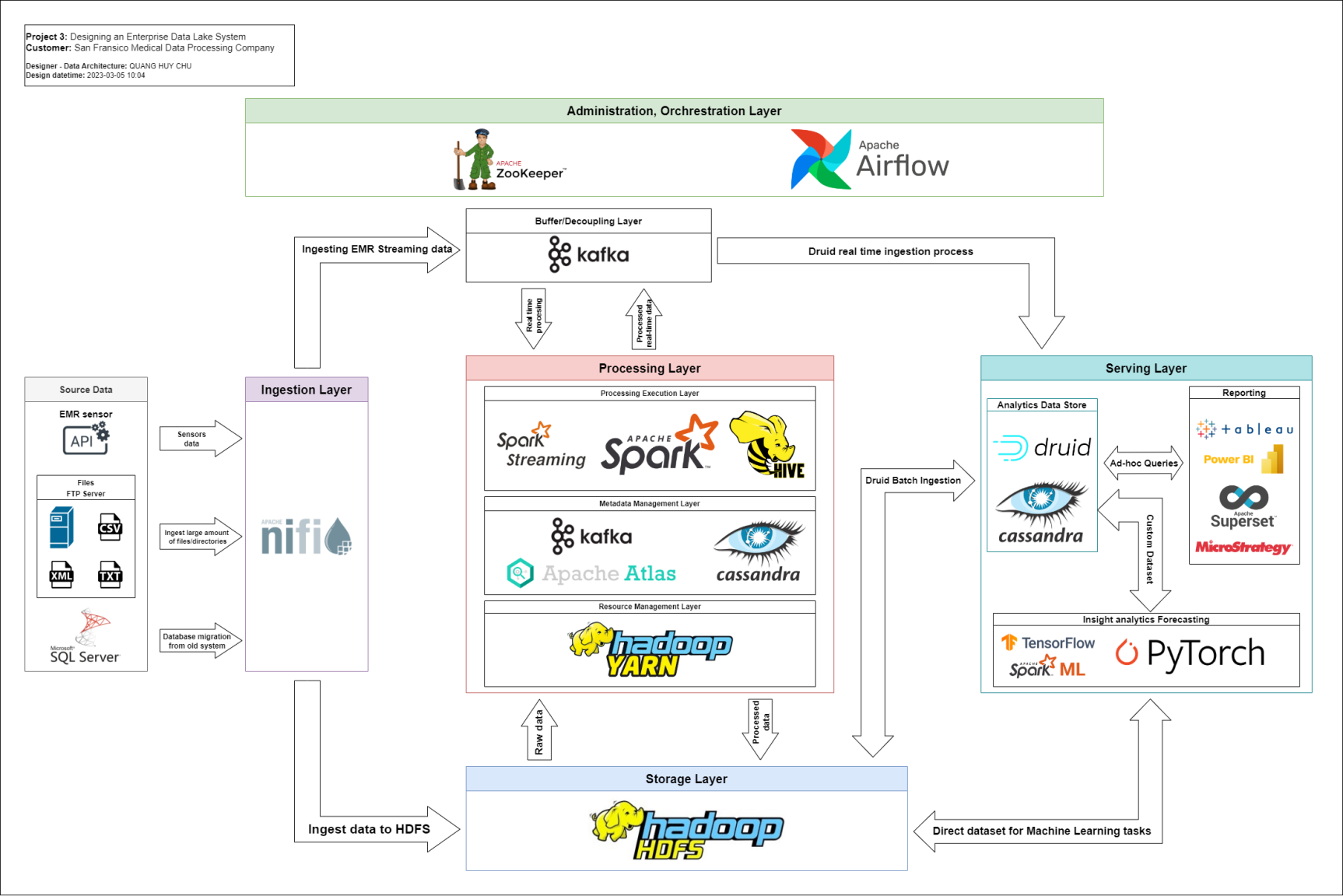


Figure 1. New Data Lake system architecture

# Design Considerations and Rationale

According to the Figure 1, the architecture has total 6 layers:

## Ingestion Layer

As I mentioned, in this layer, Apache Nifi is chosen for the ingestion layer, as it can handle perfectly the incoming data for this use case:

* **User-friendly**: Apache Nifi provides a Web UI that helps developers, or customers, monitor and manage ingestion tasks more intuitively.
* **Support various sources ingestion**: Apache NiFi can be used to ingest data from various sources such as file systems, databases, HTTP/HTTPS, MQTT, Kafka, and more. In this case, files or FTP server, API data from EMR sensor, or even SQL server database can ingest to Apache Nifi with different process.
* **Handle large amount of data**: Apache Nifi has a master-slave node architecture that is designed to handle data with large volume and high velocity allow storing data across nodes.
* **Support Change Data Capture (CDC), UPSERT**: Apache Nifi use CDC to capturing changes made to the data sources (INSERTs, UPDATEs, and DELETEs) then processing them in real-time and replicate the changes to the Storage Layer. Apache Nifi also supports UPSERT for certain database, including SQL Server of the current system.
* **Fault-tolerance**: Apache Nifi has a cluster architecture which help it not only have a capability to handle large amount of data but also support fault-tolerance thanks to its data replication and backup, which ensure data is not lost in the event of node failure.
* **Support data compression**: Apache Nifi supports data compression at the output, making data storage more efficient and less memory-intensive than uncompressed data storage.

In general, with Apache Nifi, we can easily receive and ingest data from various sources with high speed and high volume without worrying about data loss in the event of hardware failure.

## Buffer / Decoupling Layer:

The reason I added this layer is to consolidate the ability of process real-time data. Apache Kafka is chosen since it provide many feature that meet the requirement of handling real-time data in this case, such as:

* **Scalability**: Apache Kafka has a cluster architecture of brokers. This makes Kafka can handle a large amount of data in real time, which can satisfy the need of processing real-time data from 8000 facilities.
* **Durability**: Apache Kafka provides durability guarantees by replicating data across broker, ensure data is not loss if a broker fails.
* **Decoupling**: This is the most important reason why the Buffer/Decoupling layer is necessary for this case. Apache Kafka acts as a decoupling layer between Ingestion and Processing layers and between Processing and Serving layer for real-time analysis, allowing them to operate independently in the case when any layer experiencing failure or issue.

## Storage Layer:

This layer is considered the most important part of the system, this is where the data is stored, the key to the system. To avoid the difficulties that the legacy system is facing, Hadoop Distributed File System (HDFS) is considered to take over this role. HDFS provides a distributed storage system that:

* **Ensure fault tolerance and durability**: HDFS has a mechanism to replicate data across nodes. Ensure that if one node fails, the remaining nodes can still use the data copied at the failed node.
* **Scalability**: HDFS is designed to be able to store and manage big data that can be distributed on a cluster. As the data grows larger, we just need to add more nodes to the cluster, these nodes do not need to have a strong configuration, just commodity hardware can become a node. This helps customer to solve the storage of large amounts of data that is and will increase in the future as well as solve the problem of customer's hardware costs.
* **Performance**: HDFS provide Data Locality, which will try to keep data closer to the processing node to minimize the data transfer over the network. Thus, can improve the performance.

In addition to providing many features to support Big data. HDFS also has many features that make data security even better. Some of the features include:

* **Access Control**: a built-in access control mechanism that helps restrict access to specific data or files. Access can be granted or denied based on users, groups, and IP addresses.
* **Encryption**: Hadoop Key Management Server (Hadoop KMS) and Transparent Data Encryption (TDE) that can be used with HDFS. Encryption ensures that data is secured even if the physical storage is damaged or stolen.

## Processing Layer

This layer serves three different purposes:

1. Perform ETL on large amounts of data in real-time.

2. Improve the flexibility and the execution speed of batch ETL processes on large amounts of data at certain times or when data changes (CDC), not necessarily late at night and ensure fault-tolerance.

3. Data Warehousing for multi-purpose data analytics such as data visualization or creating data sets for Machine Learning tasks.

In addition to meeting the above three requirements, this layer is considered as the heart and brain of the whole system, so it is designed to be more complex than other layers. In the Processing layer, there are three sub-layers including:

1. **Processing Execution Sub-Layer:**

This layer is where batch or real-time data processing jobs are carried out. For this use case, Apache Spark and Apache Hive were considered to be added to this layer.

* 1. **Apache Spark:**
* Apache Spark is an in-memory distributed computing engine designed for fast data processing and iterative algorithms. Apache Spark splits data and stores it in worker nodes, data processing tasks will be performed in parallel to speed up computation and help Apache Spark to handle large amounts of data.
* Provides APIs for many different languages like Java, Scala, Python, or R to make complex data processing and analytics easier for developers.
* Apache Spark is best suited not only for batch processing but also for real-time data processing. Spark Streaming is a component of Apache Spark, designed to optimize data processing with low latency and high throughput.
  1. **Apache Hive:**
* Apache Hive is a Data Warehouse software that facilitates querying and managing large-scale data.
* Apache Hive provides Hive Structured Query (HQL), a SQL-like query language to enable data analytics and processing.
* Optimized queries using MapReduce, hence Apache Hive is best suited for batch processing, and data warehousing tasks.
* Apache Hive also has master-slave node architecture which helps it handle well on large amounts of data in parallel.

Overall, Apache Spark is best suited for real-time and batch processing, complex data processing and analysis, and iterative algorithms, while Apache Hive is best suited for batch processing, large-scale data processing, and data warehousing tasks.

In our customer's case, Apache Spark provides fast, accurate computation on large amounts of data with high fault tolerance with RDDs. Apache Spark offers both batch and real-time processing, meeting the needs of high complexity compared to existing ETL batches and satisfying real-time data processing capabilities. Apache Hive acts as a engine providing large-scale data analysis with SQL-like queries, it reinforces the power of historical data analysis on big data. In addition, processed data from Apache Spark can also be saved and analyzed in combination with Apache Hive to diversify data analysis capabilities.

1. **Metadata Management Sub-Layer:**

Although Apache Spark's master node and Apache Hive's metastore are both designed to store metadata, storing a lot of metadata of a large system on Master nodes and Metastore can affect their performance. So in the Processing Layer, we add a metadata management sub-layer to ensure this task, thus, Apache Atlas, Apache Kafka, and Apache Cassandra was chosen to handle the role. In this sub layer, Apache Atlas serves as a metadata management platform for managing all the metadata of Apache Spark (including Spark Streaming) and Apache Hive. In the case when this system expands, the number of nodes will increase, which means that the amount of metadata also increases, then Apache Cassandra will act as a metadata store to help Apache Atlas manage more smoothly because Apache Cassandra is a NoSQL that supports high availability, high scalability suitable for large metadata storage, it also provides high read and write throughput, suitable for continuous metadata access in real-time processing scenarios. Finally, we use another Apache Kafka cluster to act as a message broker between Apache Spark, Apache Hive from Processing Layer to Apache Atlas, and Apache Atlas to Apache Cassandra. Apache Kafka, with its properties suitable for streaming data, will make it easier and faster to provide metadata information between entities.

1. **Resource Management Sub-Layer:**

On top of HDFS in the Storage Layer and at the bottom of the Processing Layer is the Apache YARN. Apache YARN is primarily designed for resource management like CPU, RAM, etc. for Apache Spark and Apache Hive, making the resource sharing when executing tasks more flexible. In addition, Apache Atlas, Apache Cassandra, and Apache Kafka also have mechanisms to integrate with YARN to further improve their resource management.

## Serving Layer

In this architecture, the serving layer will take care of delivering processed data from the Processing Layer to the BI Tools or Machine Learning tools where the end user will create new reporting dashboards or insights search algorithms.

In this layer, there are three main components: Analytics Data Store, Reporting, and Insight analytics or Forecasting. The most important of which is the Analytics Data Store including Apache Druid and Apache Cassandra:

Apache Druid is a real-time, column-oriented data store that supports both real-time ingestion and querying capabilities on large-scale data, and also supports SQL that can be used as a real-time OLAP engine for fast querying. Druid can ingest historical processed data from HDFS through batch ingestion, as well as real-time processed data through real-time ingestion.

In addition, Apache Druid backed by Apache Cassandra makes it easier to store and process large-scale historical data, with high durability, high scalability, and fault-tolerance. One thing to keep in mind is that Apache Cassandra is an AP (Availability and Partition Tolerance), but you can still balance data consistency by tweaking Cassandra's consistency level.

End-user with BI tools such as Tableau, Microsoft Power BI, MicroStrategy, or Apache Superset can perform SQL ad-hoc queries to Apache Druid for retrieving and visualizing real-time dashboards or historical dashboards for reporting. Note that some BI tools need connector such as ODBC or JDBC connector in order to connect to Apache Druid.

With the problem of insight discovery or forecasting using machine learning algorithms. End-users can use processed datasets from Apache Hive or Apache Spark stored in HDFS or can take advantage of Apache Druid to feature engineering datasets. In additional, end user can use Apache Spark ML for creating Machine Learning models, or they can use other free frameworks, libraries such as Keras, Tensorflow, or Scikit-learn.

Overall, this layer is the most top layer of the whole system, it interacts with end-user and acts as a hub between the Data Lake and the Analytics/Reporting/Machine learning layer.

## Administration/Orchestration Layer

With a large system, governance not only with nodes but also with processing tasks running in the whole system is paramount. So this layer was added to reinforce the above. Apache Zookeeper and Apache Airflow will provide the necessary services to make it easier to manage the Data Lake system. Apache Zookeeper helps to manage and coordinate all the nodes of various components of this architecture, while Apache Airflow makes monitoring and creating processing tasks easier and more intuitive by setting up a many Direct Acyclic Graph (DAGs).

# 7. Conclusion

In this document, I have designed and explained the architecture of the new Data Lake system to replace the SQL server system that is overloaded and cannot meet the needs of the customer. This architecture is designed to withstand high loads from huge amounts of data, can be responsible for data processing in real-time and batch-processing with flexible execution time not only at night, and can overcome system failure that could not be recovered immediately.

This is an architecture that is based on documentation from the customer, so in the next meetings, this architecture may change depending on the opinions and requirements of the customer.

# 8. References

**A. Apache Ecosystem:**

1. Apache Nifi documentation:

<https://nifi.apache.org/>

2. Apache Kafka documentation:

<https://kafka.apache.org/>

3. Apache Hadoop documentation:

<https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>

4. Apache Hadoop YARN:

<https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

5. Apache Atlas:

<https://atlas.apache.org/#/>

6. Apache Cassandra:

<https://cassandra.apache.org/_/index.html>

7. Apache Spark (including Spark Streaming):

<https://spark.apache.org/>

8. Apache Hive:

<https://hive.apache.org/>

1. Apache Druid:

<https://druid.apache.org/>

11. Apache Zookeeper:

<https://zookeeper.apache.org/>

1. Apache Airflow:

<https://airflow.apache.org/>

**B. BI Tools:**

1. Tableau:

<https://www.tableau.com/>

2. Microsoft Power BI:

<https://powerbi.microsoft.com/en-us/>

3. Apache Superset:

<https://superset.apache.org/>

4. MicroStrategy:

<https://www.microstrategy.com/en>

1. **Machine Learning frameworks/libraries:**
2. Tensorflow:

<https://www.tensorflow.org/>

1. Spark ML (Apache Spark):

<https://spark.apache.org/docs/latest/ml-guide>

1. Keras:

<https://keras.io/>

1. PyTorch:

<https://pytorch.org/>