APACHE KUDU BASICS

By QuickTechie.com

Chapter-1

Introduction

Initially we have learned that Hadoop framework would work with only the HDFS (Hadoop Distributed File System) but now things have changed and there are various different kind of file system which are supported by the Hadoop framework for exam below

- HDFS
- AWS S3
- Azure Data Lake
- Apache Kudu

Isolation of Compute and Storage

There is another major change done in the Hadoop Big Data world, because most of the Big Data Solution provided now need to support the public, private cloud as well as on-prem data center setup. And to accommodate this they are changing the basic philosophy of the Hadoop framework. If you remember initial MapReduce solution works by sending the code to the data. And in that case, all the computation was running on the Data Node in the Hadoop cluster.

This has been changed by most of the platform, if they are running the Big Data Solution in the cloud. As data remain in the Cloud Storage for example Google Storage, AWS S3, MS Azure Data Lake. This helps in isolating the compute and storage. This certainly has some impact on the performance. But the compute engine now used are much faster than the native MapReduce solution for example Cloudera Impala, Apache Spark etc. They have separate compute engine underline. So if you are preparing for the interview then keep in mind that this concepts has changed.

Enterprise Data Cloud

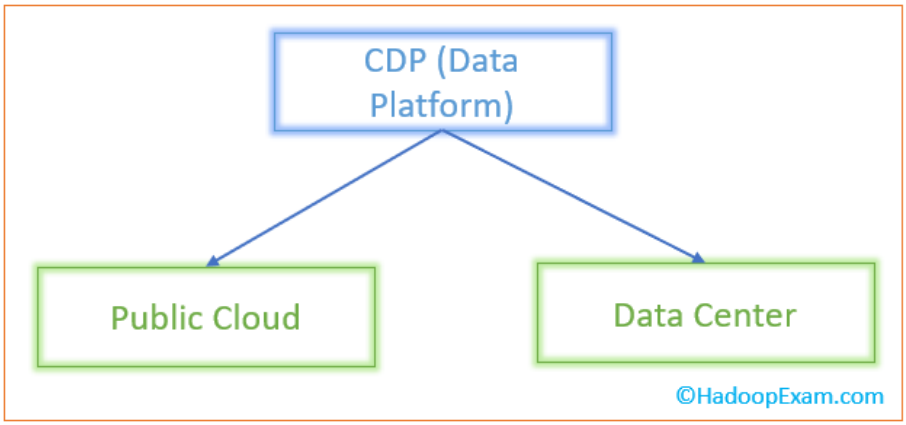
Previously whenever you are working with the BigData only things which comes in mind is the Hadoop framework. This is no more the case; every enterprise is coming with their own solution for solving the Big Data problems and Big Data platform should support at least following solutions

- Data Ware House
- Data Mart
- Business Intelligence
- Data Science
- Machine Learning
- ETL
- Interactive Query on the Big Data
- Batch processing of the Data

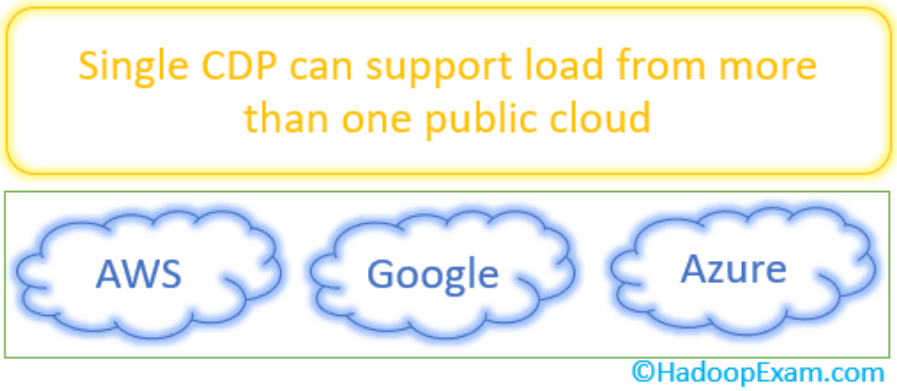
- Real-time and Near real time stream data processing

And single Hadoop framework is not capable of handling all this situation efficiently. And need to integrate many open source and close source products together to create a solution which can work in the public/private cloud as well as on-premise data center. Below are the two companies which are leading in this fields are

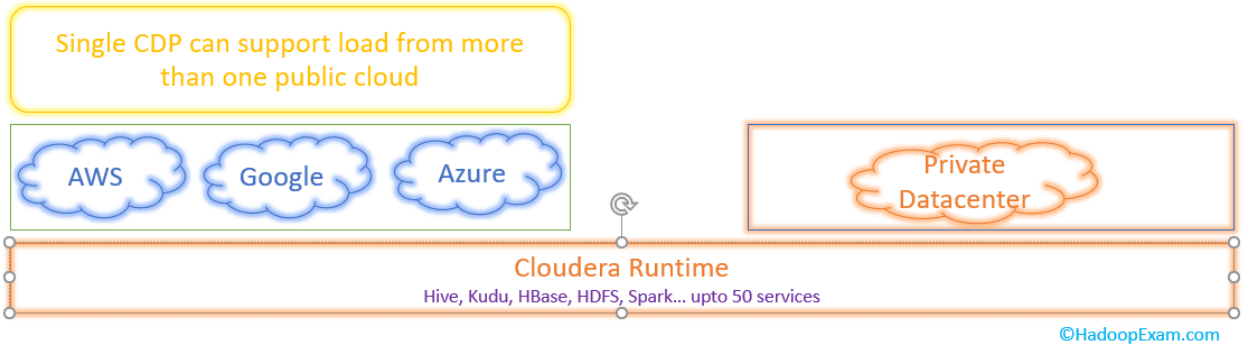
- Cloudera Inc
- MapR Inc an HPE Company



Cloudera has come up with the new platform called Cloudera Data Platform which has two different model for the public Cloud known as "Cloudera Data Platform Public Cloud" and another solution is "Cloudera Data Platform Data Center". On the similar fashion MapR created the solution. These are basically integrated software solution. However, to provide the support in the public cloud and on-prem they have to create different solutions and as an end user or System administrator you should know the differences. http://HadoopExam.com had launched a separate book for the Cloudera Data Platform which you can access from this link.



Most of the Enterprise Cloud solution are build using the Open Source Solution, if you see Cloudera Data Platform include around 50 open source technology and create a single platform which can be supported in the public cloud as well as in the on-premise data center and this still evolving. That's the reason there is a huge demand for such platform in the industry which are working in the Big Data. There are hardly few companies which provide such an excellent solution.

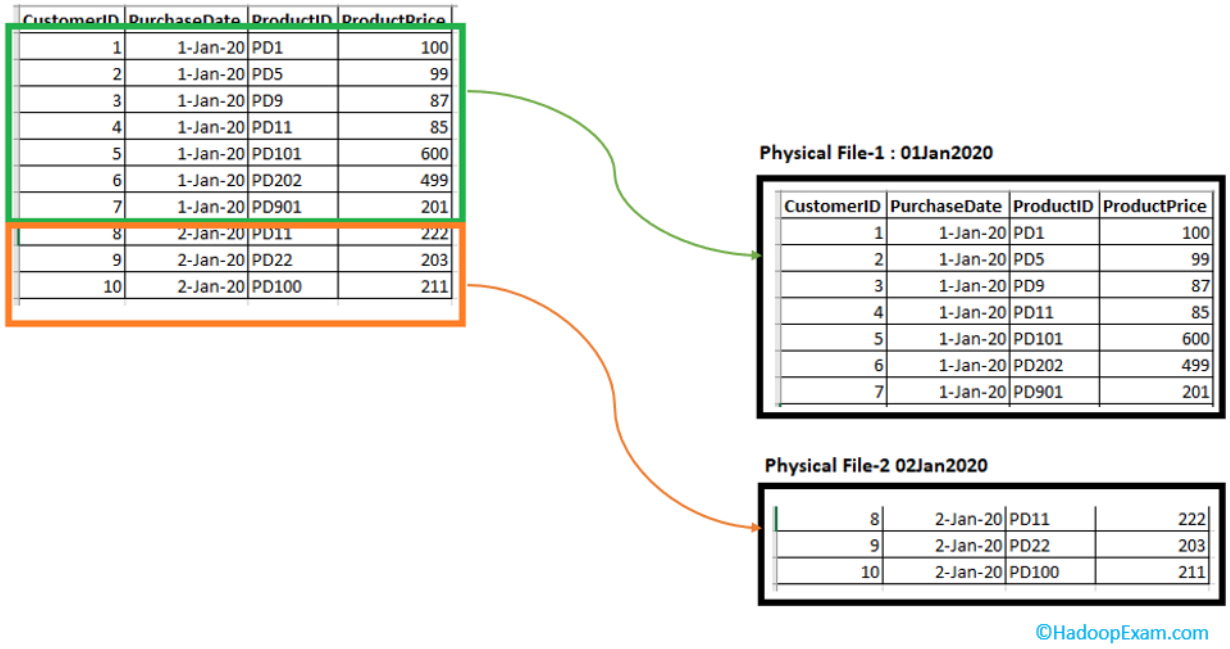


Column Storage Format

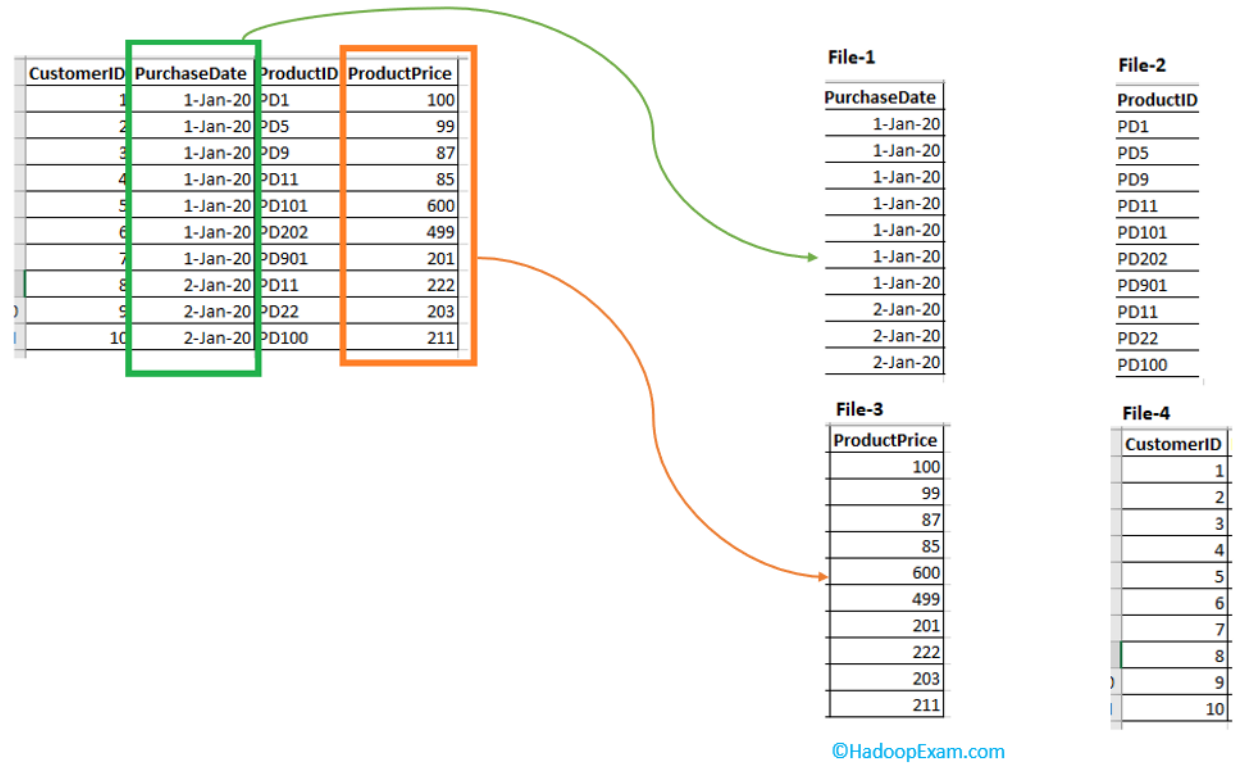
If you have seen analytical queries mostly done on few of the columns from the table. For example in the below given dataset (assume it's a huge table with many 100K's of records). You wanted to simply calculate the average product purchase price for each date. Then you are interested only two columns "PurchaseDate" and "ProductPrice".

CustomerID	PurchaseDate	ProductID	ProductPrice
1	1-Jan-20	PD1	100
2	1-Jan-20	PD5	99
3	1-Jan-20	PD9	87
4	1-Jan-20	PD11	85
5	1-Jan-20	PD101	600
6	1-Jan-20	PD202	499
7	1-Jan-20	PD901	201
8	2-Jan-20	PD11	222
9	2-Jan-20	PD22	203
10	2-Jan-20	PD100	211

If you see traditional databases which stores data row wise in the file system as depicted below, if partition by date.



This is a good storage solution if you want all the columns for a particular date and it simply access the individual file and give you the entire data for that date. But for our analytical query this is not an ideal storage solution, where we want to access only two columns PurchaseDate and ProductPrice (consider huge volume of data, just to depict we have taken only 10 rows). We need to store each column individually in a separate file as depicted below.



In this processing engine has to access only two separate file and less amount of data and all the data sit together on the single file so calculation is much faster. There is much more optimization is done like creating a folder for each date and then storing physical file in that folder so that it can easily calculate the average, total price etc. By this you would have got the idea what is column storage means. Column storage is best solution where we have analytical use cases and as in above case, we need to query almost exclusively use a subset of the columns in the queried table and aggregates the values over broad range of rows as in our example “ProductPrice” column.

On the contrary, operational use-cases mostly access all of the columns in a row and better served using row-oriented storage. And Kudu is using column-oriented storage because it is primarily used for analytical queries.

About Apache Kudu

Apache Kudu is a storage solution developed for the Hadoop platform and the major difference is that it is based on the columnar storage manager. And similar to Hadoop framework this also supports distributed storage and runs on the commodity hardware and horizontally scalable which support the highly available operations.

Benefits of the Apache Kudu

There are many benefits of the using Apache Kudu storage, some of the major ones are listed below.

- **Analytics Solution:** Kudu works well for the OLAP (Online Analytics Processing) and much faster for such load.
- **Integration with compute:** Apache Kudu is well integrated with the existing compute solution like
    - MapReduce
    - Spark
    - Flume (Cloudera removing support for Flume from their CDP platform)
    - Apache Impala
- **Alternate to Parquet**: Apache Kudu has tight integration with the Impala and this is making it a good alternative for using HDFS with Parquet.
- **Consistency**: In the Apache Kudu you can have strong as well as flexible consistency model (Similar to the Apache Cassandra) and based on your requirement you can choose consistency for each individual request as well. It also supports the strict serialized consistency.
- **Performance:** Apache Kudu is highly performant for both running sequential and random workloads simultaneously.

- **Integrated with Cloudera Manager:** If you are using Cloudera CDP platform then using the Cloudera Manager you can administer, configure the Apache Kudu. However, it comes with its own configuration files and tools.
- **Structured:** Apache Kudu support the structured data model.
- **High Availability:** As this is a distributed framework and for that reason it is created keeping high availability keeping in mind.
- **Close to RDBMS:** As we have seen there are professionals who knows RDBMS SQL quite well and having this knowledge make their work using Apache Kudu much easier. Because Kudu is able to provide very similar functionality and data model as RDBMS engines provide.
- **CRUD Operations:** Kudu provides the same insert, update, delete and select statement as provided in RDBMS.

If you see all the above features are not available based on the available storage solution whether you consider the HDFS, HBase, S3, Azure Data Lake, Google Storage etc.

Application for Kudu

Enterprise Cloud Data Solution provider already considering and integrated Apache Kudu in their platform for providing the support for the following kind of applications.

- **Real Time Reports**: Applications where new data must be immediately available for end users.
- **Time-series data**: You have seen that there are huge volume of historic data is stored based on the date basis best example is trading data, e-commerce purchase history etc. Apache Kudu supports the queries on the huge volume of Historic Data and at the same time you can access small entity with highly granular data access.
- **Predictive Modeling**: In the recent growth of the Machine Learning world where predictive models must process the data in real-time to make the instantaneous decision. And regularly refreshing predictive models based on new real-time data which were created using the historic data.
- **Legacy source**: Even using the Kudu you can access and query data from any legacy sources or formats using Impala.

Apache Kudu and Cloudera Data Platform

For the both the deployment model of CDP (Public cloud) and CDP-DC(Data Center) Kudu is available as part of the Cloudera Runtime services and part of the Operations Data Mart template. If you are using CDP public cloud then to use the Apache Kudu you have to create a Data Hub cluster using the Cloudera Management Console and then select a template which is Operations and Data Mart in the Cluster Definition Dropdown Menu.

CDP cluster created using the Data Hub cluster and Operations Data Mart template has an instance of following components or services

- Apache Kudu
- **Apache Impala:** Manages the hierarchical storage between Kudu and Object storage. You can use SQL Query features like “UNION ALL” or “VIEW” to query the data from Kudu and Parquet together.
- Spark
- Knox
- YARN
- Hue

You can use Cloudera manager to manage the cluster. These all components are using the shared resources present with the Data Lake.

HBase v/s Kudu

Kudu and HBase has some common characteristics couple of them are below

- Real-time store
- Support for the Key-indexed record lookup and mutation

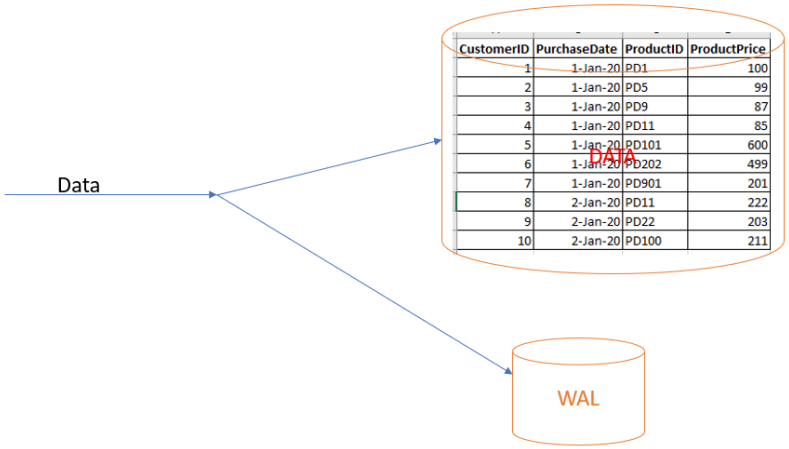
Below are the few fundamental differences existing between HBase and Kudu

- **Schema**: HBase is schema less but Kudu’s data model is more traditionally relational.
- **Storage**: As we have already discussed Kudu follows purely columnar storage while HBase not. HBase underline storage is very different storage design.

If the same change needs to be done in the HBase solution then this would have required lot of work and does not seems easy. There are some use cases where HBase fits better than the Kudu. Hence, it was decided to create new storage solution altogether.

Kudu Storage Device

Kudu uses the EXT4 or XFS storage devices using the local file system. Kudu does not require the RAID solution. To avoid data loss Kudu uses the Write Ahead Log (WAL) very similar to the HBase and this can be stored on the separate locations from the actual data files. We will discuss the entire mechanism later on.



We can even take the advantage of the SSD, and store all the WAL data in SSD which is separate location from the actual data files (same is shown in above image) and with the WAL on SSD we can have lower latency writes by involving SSD for WAL and magnetic disks for the data files.

Parquet v/s Kudu

If you see Kudu's on disk data format is quite similar to the Parquet, with some subtle differences for providing random data access as well as the updates. And if you want to query the data you can directly access from the file but you need the Kudu's client API. The storage for the Kudu is very efficient.

There is not a single compression strategy which can fit for all the use cases, it is more of use case dependent and you need to choose based on your CPU utilization and storage efficiency. Compression codec selection is completely based on the use cases.

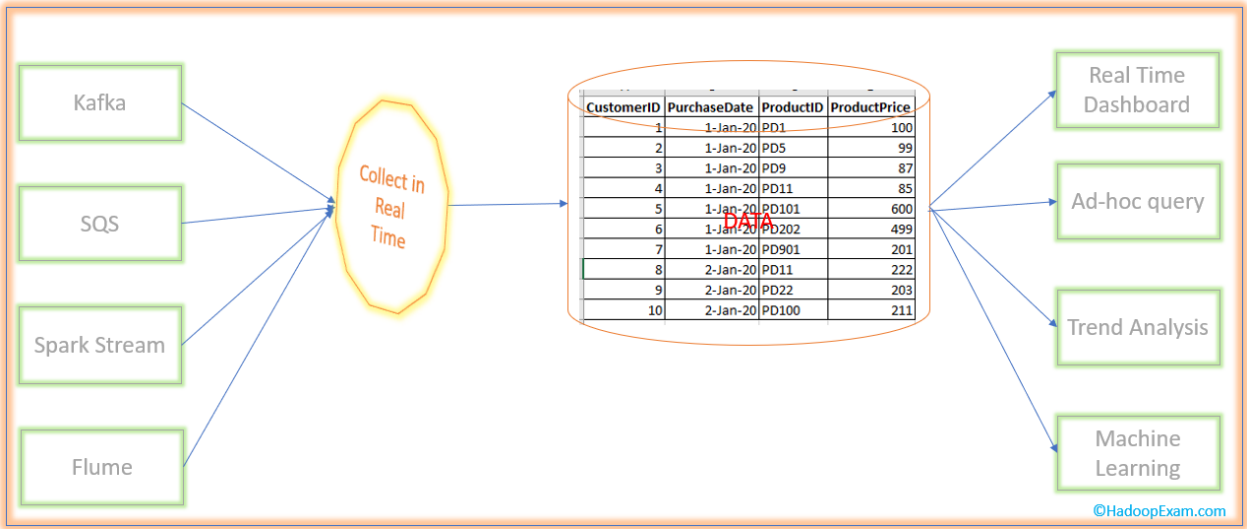
In-memory database and Kudu

As we have been discussing that data is stored on the disk and WAL (SSD's), so it is clearly not an in-memory database. There is an option available if you use the Kudu's C++ implementation which can scale at very large heaps. And couple it with the CPU-efficient design, and if your dataset can fit in the memory then it can give very good performance.

Real-time Data Processing without Kudu

Since last few years, we have seen there is a huge demand for processing data in real-time and there are various tools which provides the facility to collect the data in real time. Some of the which are more popular below

- Kafka Messaging
- Spark-Structured Streaming
- Messaging Engine like SQS
- AWS Kinesis
- Flume



There are tools available for collecting and querying the data in real-time. But at the same time, we need storage layer which can support the extraction and query the data in real-time as well, without even loosing the data. Data which is collected are in GB's every hour. Hence, storage layer should also be big enough which can hold and process the data as fast as they are delivered. There are very few such storage layers are available and Apache Kudu is one of them.

Initially we could have thought for using HDFS as storage with the file format like Avro, Parquet. The major difference between Avro and Parquet is that Avro is a row-oriented file format while Parquet is a column-oriented file format. And for streaming data Avro is better fit then Parquet, when you write the data. Then we can think of creating managed/external Hive table on that data by providing required schema.

What problem you see with this design, as we are doing good while writing the data. Because for every batch we are creating a new file. But problem is with the other side while querying or reading this data. Because there are so many tiny files created in HDFS that querying data across so many tiny files is big performance bottleneck. It even does not matter how good is your engine e.g. Apache Tez, Apache Impala, or Apache Spark any of this would not be able to perform better. Because the problem is not with the processing engine but with the Storage Layer.

If you know HDFS better than you can think of reducing the number of time files and the process which can be used to reduce the number of files is known as compaction. There are some disadvantages of the compaction process

- Compaction process can not overwrite the files in the same partition and can cause the failure of already running query and you may have to ask team not to query the data while compaction process is running.
    - You would first compact the data at separate location
    - Then copy back the data in the existing partition location.
- As you can see this is a two-step process, so consistency is not possible.

Real-time Data De-duplication

As we know in the distributed system data duplication is very common problem. Most of the streaming solution (other than Spark Structured Streaming) possibly deliver the same data more than once to avoid data loss. And it is your responsibility to remove the duplicated data. As we have discussed previously, we need to merge the various tiny files into bigger files and with that process we can introduce one more step for removing the duplicate data.

Late Data problem

There is another problem with the distributed system that, we can receive duplicate data. But it is also possible that your data arrives quite late. So, what criteria you would follow to process late data so that it can fall in the same merged large file. Hence, we have to delay the compaction process to consider the late data. Now question is how much delay we need to consider. (I would highly recommend you learn Spark Structured Streaming, this would take care of these two problems of late arrival of data and delivering data only once, check training at this link).

HBase NoSQL Database

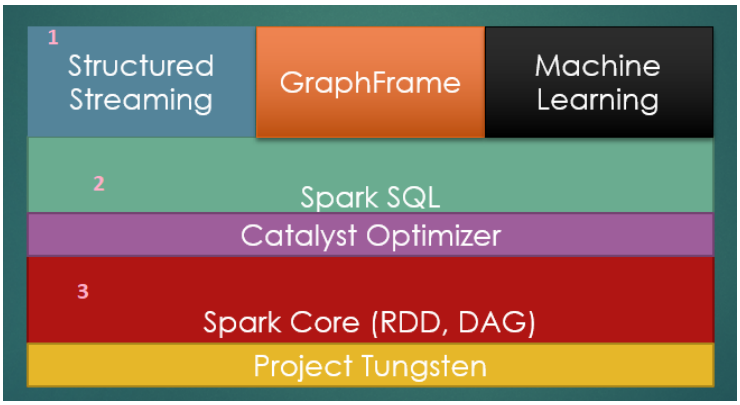
I am sure you might have thought why we are not using the NoSQL storage rather than HDFS based file system. Yes, this is certainly a good thought. But I wanted you to know some basic problems on the distributed systems. And how the Distributed systems are evolving time to time. HBase is not as good as Parquet files in the HDFS for analytical queries. HBase is good for inserting or writing the data. But not good for analytical query.

Spark-structured streaming

To process the data in real-time or near real time a completely new framework is developed on the Spark. Previously it was using DStream framework, which is quite complicated for the developer to understand as well as to develop application. Hence, again Spark Development team had created a new

Framework for the structured data, this is known as “Structured Streaming”. If you can convert your data in well structured (can define a schema) then use the structured streaming which has various in-built feature. One of the well-known features is process exactly once. If you are using any other streaming solution then you would have to write a solution your own to avoid duplicate data processing, in structured streaming this is in-built.

If we take a look on the Spark 2.x components architecture then we can find that each component is built on one another, same exam section is also decided in order.



In above block diagram three components (marked as 1,2 and 3) would be tested or evaluated in the real exam of HDPSCD Spark. This exam is developer oriented as name suggest (Hortonworks Spark Certified Developer). And it is expected from you that you are having some good hands on, with the programming language from one of the below.

- Spark 2.x using Scala
- Spark 2.x using Python

However, it is not expected from you that you are very proficient in any of the programming language. You can consider the trainings provided by “HadoopExam.com” for Scala and Python to have good hands on with these programming languages and good enough to work with the Spark framework. You should be able to write Spark application using one of the programming knowledge. In this exam, it is not expected from you that you have knowledge for both the programming knowledge. Knowing only one programming language either Scala or Python is good enough. Similarly, we have created separate certification preparation material for the HDPSCD using Scala and HDPSCD using Python.

We would be discussing each topic of the syllabus in detail as we move ahead. Syllabus remain same whether you use Python or Scala. However, Spark application can also be written using Java and R language also. But there is no certification available for R and Java language.

To understand the Apache Kudu architecture lets first learn some important terminology and then discuss the architectural design.

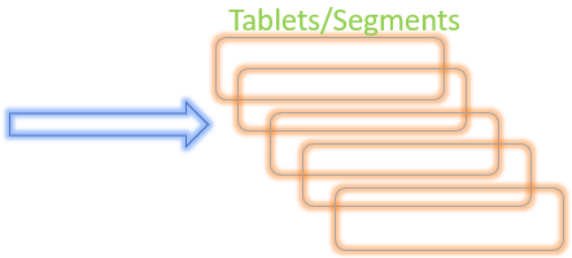
Terminology and concepts

- *Table*

  Every IT professional knows what is the table in the database and in case of Kudu also this is the same thing. Table is a place where Kudu store the data and a table would have defined schema and totally ordered primary key. Each individual table is further divided in the segment which is called tables.

CustomerID	PurchaseDate	ProductID	ProductPrice
1	1-Jan-20	PD1	100
2	1-Jan-20	PD5	99
3	1-Jan-20	PD9	87
4	1-Jan-20	PD11	85
5	1-Jan-20	PD101	600
6	1-Jan-20	PD202	499
7	1-Jan-20	PD901	201
8	2-Jan-20	PD11	222
9	2-Jan-20	PD22	203
10	2-Jan-20	PD100	211

Kudu Table



- *Tablets*

  As we have already discussed table is further divided in the segments. However, the point to be noted is that this is a contiguous segment of a table. If you know the partitioning in the relational database then this is the very same things. If we partition our data based on the date column then entire data for the same date would go in the same segment if size can fit on a single disk. And a given tablet is replicated on multiple tablet servers, and at a given point in time, one of these replicas become the leader tablet.
  - **Read Request:** Any replica can server for the read request.
  - **Write request:** This require consensus among the set of tablet servers serving the tablet.

- *Tablet Server*

  Tablet servers are the place where the tablets are stored and serves the tablets to the client. For the given tablet, one tablet server acts as a leader and remaining acts as a follower replica for the tablet. Only leader can service the write request and either follower or leader can serve the read request.

  - **Leader selection**: Leader selection is done using the “Raft consensus Algorithm”

  A single tablet server generally serves more than one tablet to the clients and one single tablet can be served by multiple tablet servers.

- *Catalog table:*

  Catalog table is a central location for metadata in Kudu and keep the information related to tables and tablets. You can not directly read or write the catalog table directly. And this is accessible only via metadata operations exposed in the client API. There are following two types of metadata

  - **Tables Metadata**: It stores the table schemas, locations, and states.
  - **Tablets Metadata**: This stores the list of existing tablets, which tablet servers have replicas for each of the tablet, state of the current tablet, start and end keys.