2023

Guide for designing business reference architectures aimed for using NoSQL databases

UNWE | Sofia

Introduction

Nowadays, when change and digital transformation are a necessity for the businesses to remain competitive, companies that fail to keep up with change, risk being left behind or out of business. Technologies developing at a faster pace from simple phones to sensors collecting data and being used almost everywhere in our daily lives to computers, robotics, Artificial Intelligence, cloud technologies, Big Data, Internet of Things, and more are considered the basis of digital transformation in various spheres.

Digitization and digital transformation cover various types of processes in our lives, progressing at an extremely fast pace. The adaptation of new technologies helps to increase efficiency and productivity in the digitization of manual processes. There are countless benefits to digital transformation, but one of the most important factors is that it helps businesses become more efficient and organized. This process has the potential to make organizations more agile, efficient and customer centric.

In the last decade, managing exponentially growing amounts of data has become an increasingly challenging task. With the development of technologies, the digitization of processes and the digital transformation of businesses, it leads to the generation of huge amounts of data that must be stored, managed, and analysed through appropriate methods. Technology development and data generation represent the initial and easiest step. The challenges start with storing, processing, analysing and deriving results from the collected data.

In the past, when the development of information technology was initiated, the main type of data collected was structured. With the digitization of more and more businesses and digital transformation, the generation of different types of data - semi-structured and unstructured - has also started. Unlike structured data, the other two types are extremely challenging to store and process. The usual relational databases used for the storage of structured data are not suitable for semi-structured and unstructured, which leads to the need to create and develop new ways of storing different types of data. One of the key requirements for Big Data storage is that it must handle massive amounts of data, with storage capacity continuing to grow without disrupting the workflow.

In recent years, the financial sector has been developing and digitizing more and more, which leads to the need for changes. For financial institutions to meet the demands, needs and expectations of their customers, it is of utmost importance to make changes and adapt quickly to evolving technologies. In addition to this, these types of institutions must manage extremely large volumes of data while making changes to their systems without causing data loss or problems for their customers. Financial services offered by banks, credit unions, accounting companies, insurance companies, investment funds, stock exchanges and others need databases that can adapt to the automation needs that non-relational databases can meet. NoSQL databases can handle the storage and processing of large amounts of data, with their scalability and providing better data processing performance.

Business Reference Architecture for NoSQL Databases for Financial Services

A reference architecture is a document or set of documents that provide a recommended structure and integrations of IT products to form a solution. RA incorporates industry best practices, usually offering the optimal method for specific technologies. The reference architecture offers IT best practices in an easy-to-understand format that guides the implementation and use of complex technology solutions.

Reference architectures add value to companies in the following ways. Eliminate possible confusion through standardization, make problem solving easier by applying precise and clear guidelines. They provide resources for the design of IT architectures, teams and systems. Reference architectures save time, effort, and money by using already available resources, as well as optimize problem resolution by applying standard best practices and support compatibility and reuse of components.

Using best practices when creating reference architectures improves efficiency, meets regulatory requirements, and reduces the chance for errors. A reference architecture helps make decisions about choosing the best model and way to create a software architecture to meet business goals.

An approach for building reference architecture

Creating a reference architecture is an extremely challenging task in the absence of predefined steps or a process to create it. There are five main steps that must be taken: identifying the purpose, formulating principles, setting technical rules and standards, building rules and standards, and applying context.

- In **identifying the purpose**, one must define the domain and scope of reference, who are the stakeholders, how the architecture will be used, what are the constraints, assumptions and environment with which it is associated.
- **Formulation of principles** occurs after the goal is successfully identified, the components required for the architecture must be included. Elements must be aligned with the company's core culture statements and values.
- **Setting technical rules and standards** happens after the formulation of the principles. The next step is to decide on the frameworks and models that will need to be followed in the company. In this part, all the necessary rules are laid down to achieve the principles formulated earlier.
- **Rules and standards** must be established in the different departments of the company. All possible challenges should be listed and the right standards should be selected for each possible scenario.
- Rules and standards include the **context** of each situation. Most companies incorporate a logical process, making it easy and possible to incorporate good practices.

When using best practices to create an enterprise reference architecture, it should be ensured that the company and customers can reap its benefits. It can improve efficiency, meet regulatory requirements and reduce the possibility of errors. Some of the good practices that can be applied are:

- competitive advantage;
- comparative indicators;
- compliance standards;
- reuse of components and management

Designing a Business Reference Architecture for Financial Services with NoSQL

Every single financial institution dealing with different types of financial services generates, collects, processes and analyzes data of different types – structured, semi-structured and unstructured in large volumes, which in turn leads to different needs of the institutions. Depending on the type of financial institution depends on the type of data that is generated, which in turn leads to different requirements for storage space, processing methods and analysis tools.

With the implementation of non-relational databases in financial institutions and adapting their historical systems to the new databases, it leads to the development and need of services in the field of finance to maintain their competitive advantage, as well as to get ahead of their competitors in the field.

The purpose of creating a reference architecture, whether for use only in one organization or for universal and adaptation across multiple financial institutions, is to ensure consistency and applicability of the use of given technologies in a particular organization.

Continuing the example of personal finance and more specifically banking or services directly related to the customer, the following elements can be considered, which are based on the various data collected about customers:

- Every financial institution has batch processes that do not require real-time or near-real-time execution, but after exceeding a certain volume, their processing continues during the institution's working hours, when activities need to happen quickly, and in this case it delays the main work and customer service. This may lead to a point where these data processes will not have finished before they need to start again. For many institutions, in-memory processing is the most efficient way to achieve the required levels of performance and scalability. In this type of processing, large volumes of data can be stored in memory and massively parallel processing can be used to provide up to 1000 times faster performance for applications built on disk-based bases. When data is stored and processed in memory, data traffic across the network is reduced or eliminated completely. Traditional relational databases are not designed to process huge amounts of data in real time, so a choice must be made about which workload to optimize. They can handle either just the operational workload, which refers to day-to-day business transactions, or the analytical workload, which refers to business intelligence systems and analytics. It is impossible to implement the 2 types of load at the same time, because relational databases cannot handle mixing them. SQL databases are designed to be specialized at the cost of flexibility. On the other hand, with non-relational databases, the performance and performance of database operations such as reading and writing, data retrieval, and data analysis can be improved. This happens because in-memory processing enables faster access to data, reducing the latency associated with traditionally disk-based storage. This allows for much faster data access and processing time, making it useful in processing time-sensitive financial services such as securities trading, risk management, and fraud detection. In-memory processing enables fast, real-time processing of large amounts of data, which can improve decision-making, increase efficiency and reduce costs in financial services.

- Financial institutions maintain customer files composed of blobs (semi-structured and unstructured data - documents related to the customer-product hierarchy) classified according to certain characteristics. A characteristic feature, however, is that blobs have a lot of context because they originate and are stored in relation to a specific customer and banking/financial transaction registered in a computer system. No processing is done on them that requires analysis of their content, only searching for signature verification and visualization/printing during on-demand service

Based on the requirements that may arise from financial services, together with the theoretical literature review, as well as the practical one of existing financial services solutions on various non-relational databases, a model of a universal component reference architecture for financial services was created. services

➢ Conceptual model

The conceptual model of the reference architecture for financial services presented in Figure 1 represents the general structure and elements of which it consists, as well as the necessary data and systems to achieve the business requirements, which means the maintenance of company processes, recording of business events and performance tracking. An approach to building a reference architecture is defined, which consists of five main steps. The created conceptual model of the business reference architecture for financial services is the result of the first two steps - the identification of the goal and the formulation of the goal. The purpose of the Business Reference Architecture for Financial Services is to adapt the modern ways of storing Big Data for different types – semi-structured and unstructured data – to financial institutions that continue their rapid development and digital transformation.

The reference architecture consists of 3 layers: Data Storage System, Integration Layer and Data Storage Server, which are connected to financial services processing systems .

❖ Data storage system

The data storage system is the system that stores the data after it has been received from the financial services processing systems.

❖ Integration layer

The integration layer in the reference architecture refers to the components responsible for the integration of data storage systems and their storage servers. This layer provides a centralized platform for management, data movement and communications between different systems. The integration layer consists of data connectors between the storage system and the server.

❖ **Data Storage Server**

The data storage server is where the data that arrives through the integration layer and the data connectors from the data warehousing systems that enter the data warehousing system from the financial services processing systems is stored.

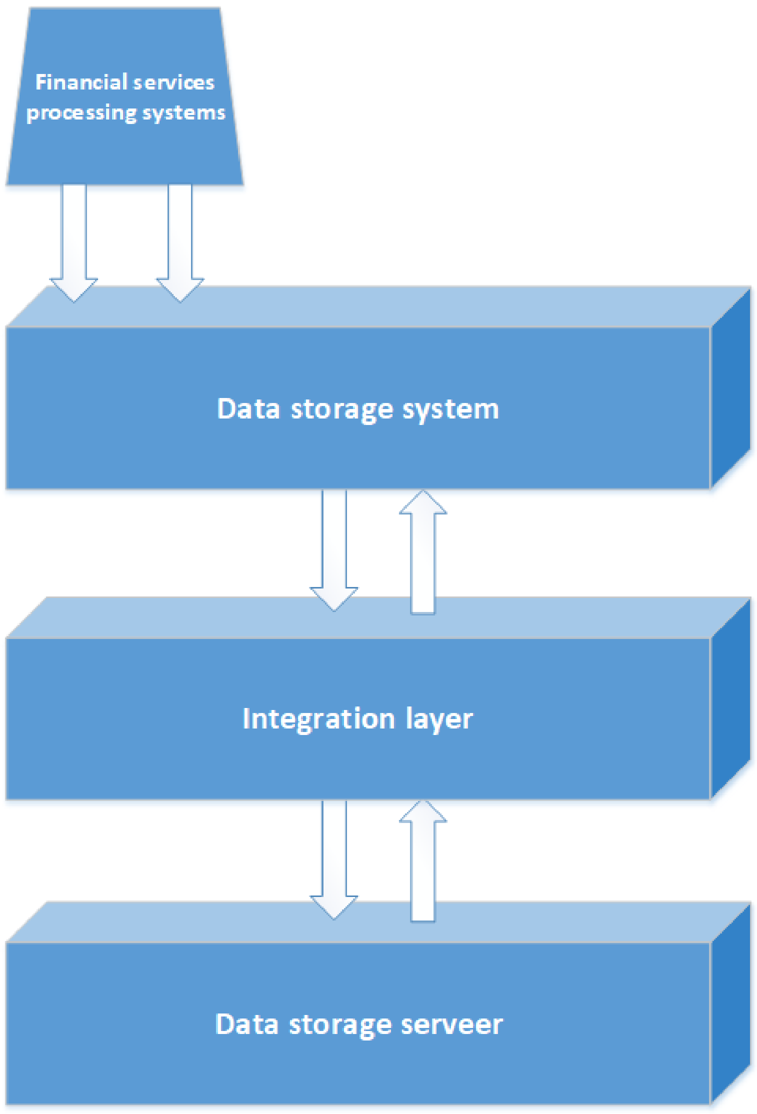


Figure 1 Conceptual model

➢ Logical model

The logical model lays down the technical rules and standards for the creation of the business reference architecture, and it lays down the rules so that the set principles can be achieved. Through the financial services processing system, data from financial institutions enters, which are subsequently stored, processed and analyzed in the remaining layers of the reference architecture. Figure 2 presents the logical model of the financial services business reference architecture, with each of its layers discussed in detail below.

❖ Data storage system

In the first layer of the reference architecture, the so-called A data storage system can host 1 or more of the four types of non-relational databases – document-oriented, key-value, wide-column, and graph databases. The types of NoSQL databases were discussed at a theoretical level, such as what they are capable of in Chapter I, and in this chapter, they were discussed at a practical level - what applications related to financial services are built on specific non-relational databases, to help the selection of specific base

The choice of such is realized based on the following elements, which should be considered carefully before proceeding to a specific NoSQL database:

- The financial institution's existing system(s) that will need to be migrated to the new base
- Type of financial services in which the financial institution deals
- Types of data that the financial institution generates based on the financial services it provides – structured, semi-structured and unstructured data

❖ Integration layer

Based on the non-relational base that is selected in the data storage system based on the considered elements, the way to connect to a specific NoSQL is selected, which is positioned in the integration layer of the reference architecture.

❖ Dedicated NoSQL database server

Depending on the selected non-relational database in the data storage system, the subsequent selection of a way to connect a specific NoSQL database, a specific NoSQL server is reached in which the data of the financial institution is stored.

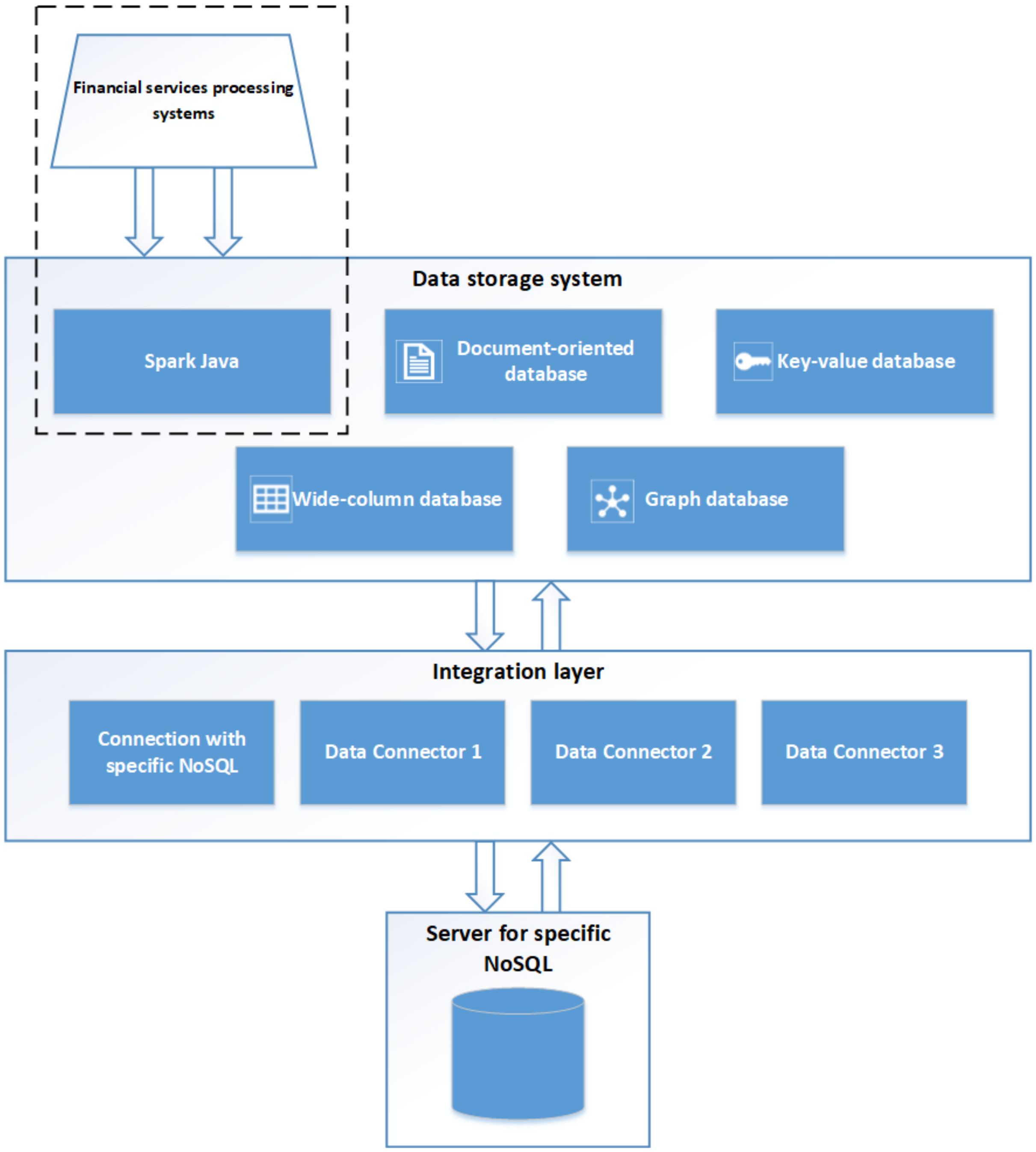


Figure 2 Logical model

➢ Physical model

The created physical model in Figure 3 is implemented with concrete examples of non-relational databases and connectors for them to Hadoop. Each of these elements can be replaced according to the needs of the financial institution that wants to adapt this reference architecture with NoSQL databases for its own needs.

In the first layer, which is the data storage system, the four types of non-relational databases - document-oriented, key-value, wide-column and graph databases - can be located, and in this layer the particular database that is selected is located. through the evaluation and selection method of NoSQL databases for financial services. Bases of each type are located in the physical model:

- Document – oriented database: **MongoDB**
- Key-value database: **Amazon DynamoDB**
- Wide – column database: **HBase**
- Graph database: **Orient DB**

In the second layer, the so-called integration layer is where the connecting elements with the different NoSQL databases are located, e.g. NiFi, Spark Connector and other types of Data Connectors suitable for the other types of non-relational databases. The specific data connectors from the data storage system to the server of the specific NoSQL database are as follows:

- For the document-oriented **MongoDB** database, the **Mongo Spark Connector** can be used
- **Amazon DynamoDB** key-value database can be used **Amazon EMR Connector**
- For wide-column **HBase**, the **Spark HBase Connector** can be used
- For the **Orient DB** graph database, the **Orient DB Spark Connector** can be used

The third layer is a server of the specific NoSQL base, and Hadoop has been chosen for the implementation of the physical model of the reference architecture for financial services.

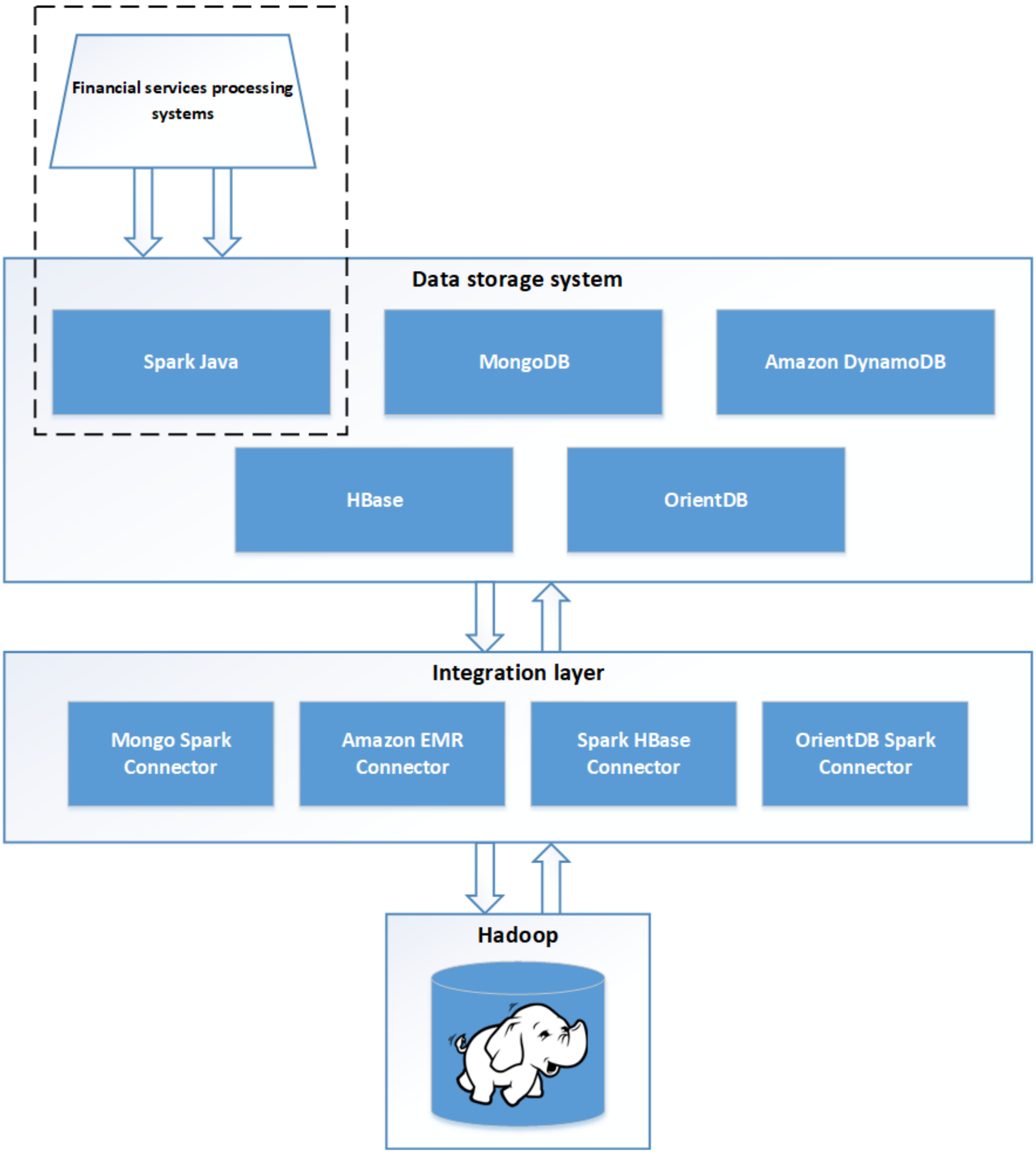


Figure 3 Physical model

An approach to the use of the designed reference architecture leading to the creation of an ICT architecture

The ICT Architecture (Information and Communication Technologies) provides a conceptual model, specifying at a basic level the elements of the ICT architecture (application, databases, technological ICT elements), as well as the relationships between them. Based on the reference architecture for financial services with NoSQL databases, which was created in the previous point and demonstrated in Figure 3, a specific ICT architecture will be created, which will be implemented and tested in the following points.

The developed physical model for the reference architecture contains 3 layers - a data storage system, an integration layer and a NoSQL-specific layer, and in the previous point we discussed in detail the different types of non-relational databases, with different types of data connectors that can be connected with the NoSQL specific layer. In the creation of the ICT architecture for the data storage system (the first layer), the document-oriented non-relational database MongoDB will be used, since it has been selected through the method created in the previous point and meets the requirements and has already created a financial service. which uses the data type with which the architecture will be tested. The second integration layer will use the Apache Spark Connector for MongoDB. Due to the fact that support for MongoDB's direct connector to Hadoop is deprecated, we will use this one through Apache Spark. The third layer remains for the Hadoop big data system that we will use for storage – Figure 4.

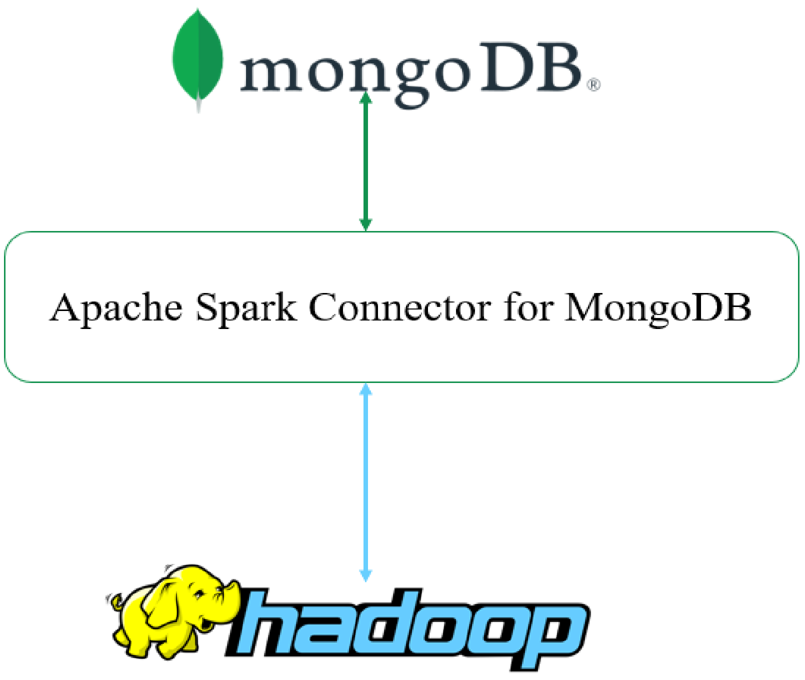


Figure 4 Specific ICT architecture

Application of a reference architecture design method for financial services

With the increasing automation and digitization of processes, and the influx of larger volumes of data of various types – mostly semi-structured and unstructured, businesses face the enormous challenge of collecting, storing and analyzing unstructured data. Financial institutions and the services they offer are fast-growing businesses, and for this reason they must adapt quickly and in a timely manner to the unfolding technologies and to the increasing demands of customers, as well as to constant competition.

Based on the detailed analysis of scientific literature related to non-relational databases and their use for the implementation of financial services systems, as well as on the basis of the developed business solutions, a business reference architecture was developed, consisting of a data storage system, an integration layer and a specific NoSQL server that was developed and reached the realization of a physical model. Each of the four types of non-relational databases can be deployed in this model, either in combination together or separately.

The ICT architecture that was tested practically consists of a data storage system – MongoDB, an integration layer – Apache Spark Connector for MongoDB and a specific NoSQL server, in this case the Hadoop distributed file system was used.

Hadoop is a technology designed to store large volumes of data distributed across different clusters. MongoDB, on the other hand, is an extremely powerful document-oriented NoSQL database. Combined together, Hadoop and MongoDB can create a complete application for handling and analyzing Big Data. Hadoop takes the data that is stored in MongoDB, merges with the data it has and thus generates analytics as well as machine learning models. The results are fed back into MongoDB, thereby being used to create better customer offers, to better identify and detect fraud attempts, to better predict changes in exchanges, and more.

The MongoDB Connector for Spark provides the integration between MongoDB and Apache Spark, and from there, data is then extracted from Hadoop. Through the connector, the user has access to all Apache Spark libraries that can be used with the data that is stored in the NoSQL databases

Figure 5 demonstrates how the connection is made through the MongoDB connector for Apache Spark, as well as all the libraries it accesses.

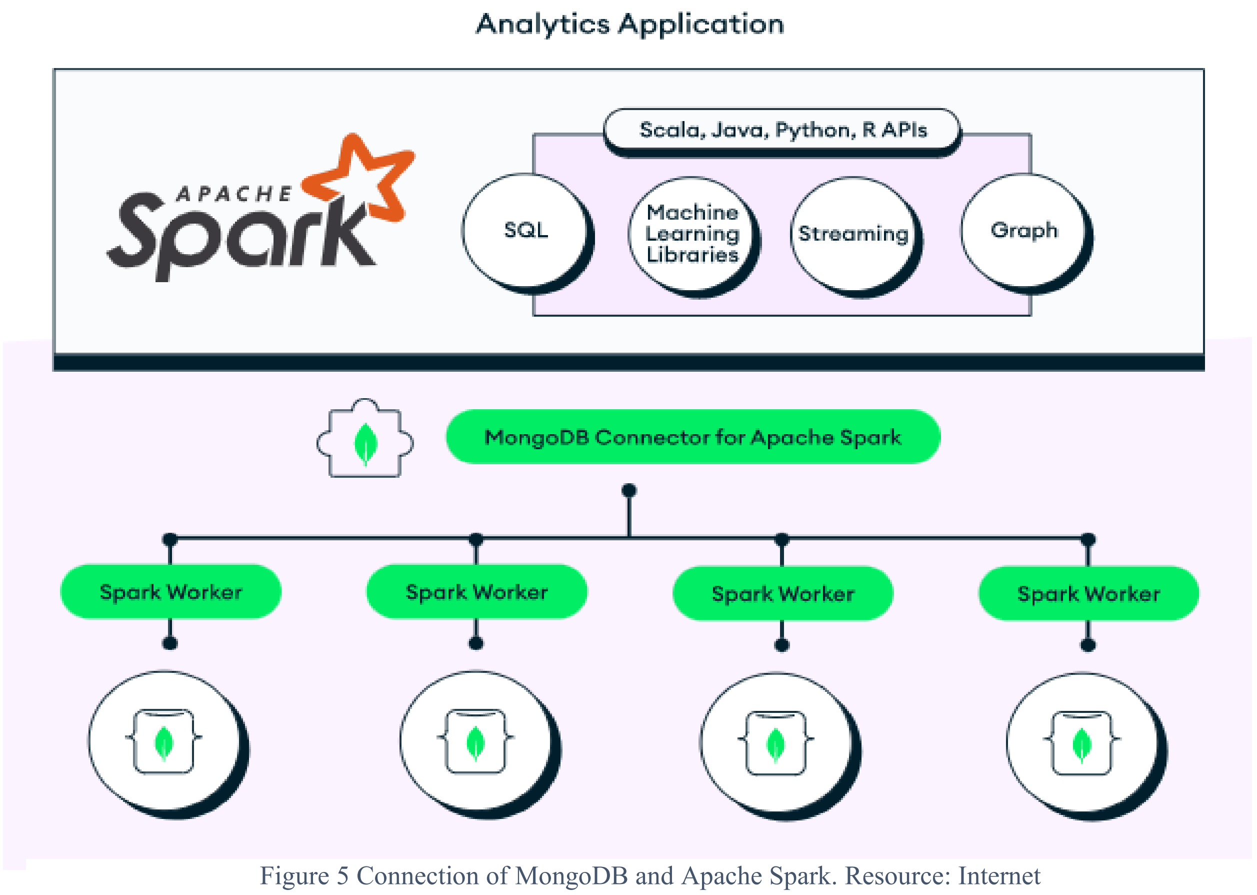


Figure 5 Connection of MongoDB and Apache Spark. Resource: Internet

The MongoDB data connector over Spark enables the integration of the document-oriented database MongoDB and Apache Spark, allowing users to implement complex analyzes with large data sets. In addition to enabling read operations, the connector also enables data to be written back to MongoDB through the Spark connector. This provides an outstanding opportunity to record results from the processing performed in Spark to be written back to MongoDB and then used for further analysis.

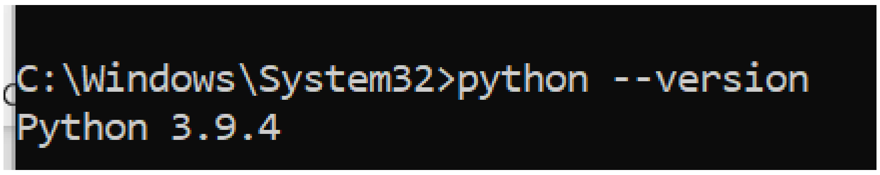
The following components are required for the implementation of the physical model of the business reference architecture:

- Hadoop

- Apache Spark connector for MongoDB

- Python

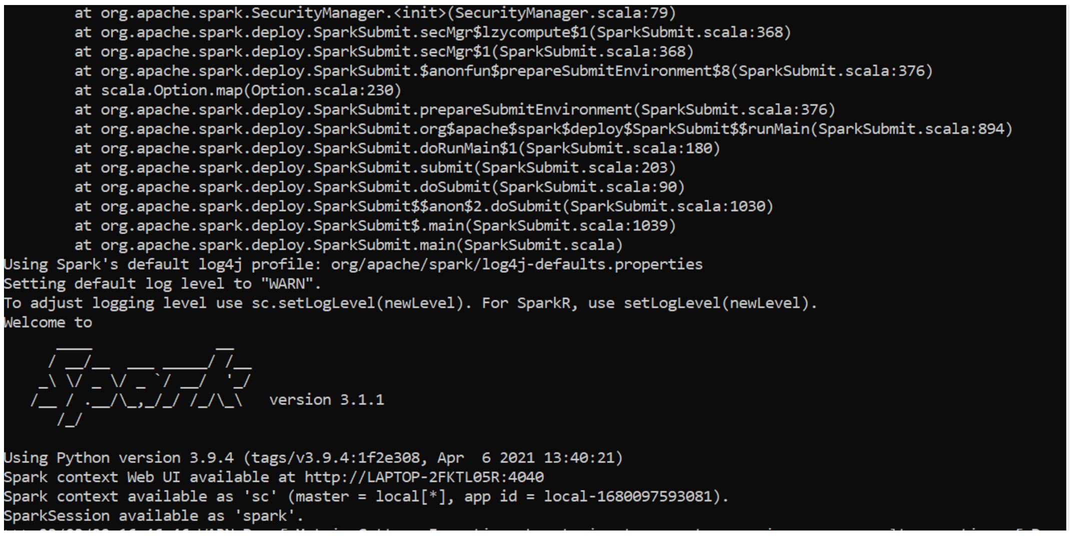
At the moment, the ICT architecture is implemented at the local level - a MongoDB connection through the Apache Spark connector to Hadoop. It starts with a local Python installation, as PySpark will be used. To install and configure Python, the necessary files are downloaded, which are run on the local machine, using standard installation software available. After the completion of and, the available version of Python on the machine is checked to ensure that the process completed correctly and work on the next steps can be continued. In the Command Prompt, type the following python –version command, and if the installation process was successful, the following result should appear - Figure 6.



```
C:\Windows\System32>python --version
Python 3.9.4
```

Figure 6 Result of correct installation of Python

Installing Spark locally is done after selecting the configurations and downloading the necessary files to the machine. Upon proper installation of Spark and entering the following command in the Command Prompt: C:\Spark\spark-3.1.1-bin-hadoop2.7\bin\spark-shell, Apache Spark is started and the screen below is displayed – Figure 7.

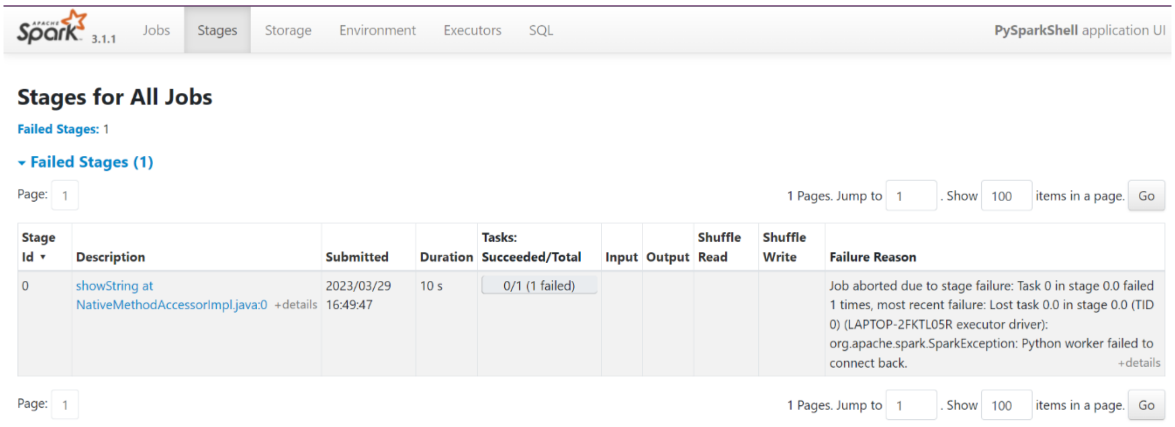


```
        at org.apache.spark.SecurityManager.<init>(SecurityManager.scala:79)
        at org.apache.spark.deploy.SparkSubmit.secMgr$lzycompute$1(SparkSubmit.scala:368)
        at org.apache.spark.deploy.SparkSubmit.secMgr$1(SparkSubmit.scala:368)
        at org.apache.spark.deploy.SparkSubmit.$anonfun$prepareSubmitEnvironment$8(SparkSubmit.scala:376)
        at scala.Option.map(Option.scala:230)
        at org.apache.spark.deploy.SparkSubmit.prepareSubmitEnvironment(SparkSubmit.scala:376)
        at org.apache.spark.deploy.SparkSubmit.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:894)
        at org.apache.spark.deploy.SparkSubmit.doRunMain$1(SparkSubmit.scala:180)
        at org.apache.spark.deploy.SparkSubmit.submit(SparkSubmit.scala:203)
        at org.apache.spark.deploy.SparkSubmit.doSubmit(SparkSubmit.scala:90)
        at org.apache.spark.deploy.SparkSubmit$$anon$2.doSubmit(SparkSubmit.scala:1030)
        at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:1039)
        at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Python version 3.9.4 (tags/v3.9.4:1f2e308, Apr  6 2021 13:40:21)
Spark context Web UI available at http://LAPTOP-2FKTL05R:4040
Spark context available as 'sc' (master = local[*], app id = local-1680097593081).
SparkSession available as 'spark'.
```

Figure 7 Result of correct installation of Apache Spark

When Apache Spark is properly installed and started, in addition to the Command prompt message, we are provided with a browser UI that can be used to monitor the various processes being performed - Figure 8, at the following address: http://localhost :4040/.



Stages for All Jobs

Failed Stages: 1

Failed Stages (1)

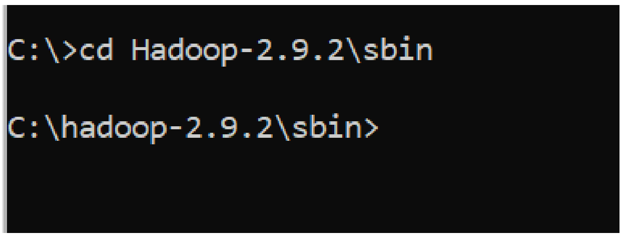
Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
0	showString at NativeMethodAccessorImpl.java:0 +details	2023/03/29 16:49:47	10 s	0/1 (1 failed)					Job aborted due to stage failure: Task 0 in stage 0.0 failed 1 times, most recent failure: Lost task 0.0 in stage 0.0 (TID 0) (LAPTOP-2FKTL05R executor driver): org.apache.spark.SparkException: Python worker failed to connect back. +details

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Figure 7 A local browser-level interface for monitoring the execution of Apache Spark tasks

The next step to implement the physical model of the business reference architecture that needs to be done is the Hadoop installation. It is downloaded, the necessary machine configurations are made, as well as additional configurations are added to some of the Hadoop files that can be found in the appendices of this thesis. To verify that the installation was successful, type the following command in the machine's Command prompt: cd Hadoop-2.9.2\sbin , and the following should appear in Figure 9.

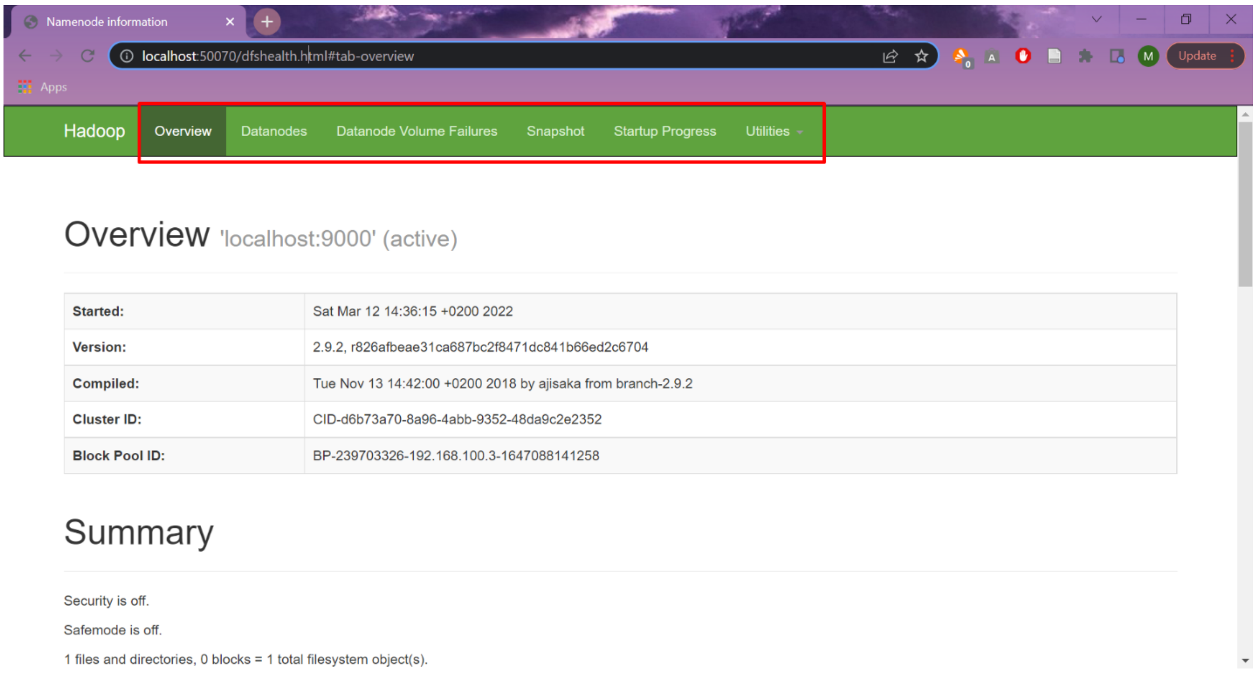


```
C:\>cd Hadoop-2.9.2\sbin

C:\hadoop-2.9.2\sbin>
```

Фигура 8 Successful installation of Hadoop

In the browser, the following address: http://localhost:50070/ opens the user interface through which, if Hadoop is started correctly, the data clusters and others can be traced.



Overview 'localhost:9000' (active)

Started:	Sat Mar 12 14:36:15 +0200 2022
Version:	2.9.2, r826afbeae31ca687bc2f8471dc841b66ed2c6704
Compiled:	Tue Nov 13 14:42:00 +0200 2018 by ajisaka from branch-2.9.2
Cluster ID:	CID-d6b73a70-8a96-4abb-9352-48da9c2e2352
Block Pool ID:	BP-239703326-192.168.100.3-1647088141258

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Figure 9 Interface of Hadoop

The last step to implement the physical architecture is the installation of the Apache Spark connector for MongoDB, which is implemented using Pyspark and the following --packages commands, and then from the options provided, mongo-spark-connector is selected, which is installed on the machine.

The initialization of the connections happens through the following code:

```
./bin/pyspark                                                                    --conf
"spark.mongodb.read.connection.uri=mongodb://localhost:27017/FinancialData.Stocks?readPreferenc
e=primaryPreferred"

          --conf
"spark.mongodb.write.connection.uri=mongodb://localhost:27017/FinancialData.Stocks"

          --packages org.mongodb.spark:mongo-spark-connector_2.12:10.1.1
```

- **Application of typical unstructured data in financial services**

Every single financial service works with different types of data – structured, semi-structured and unstructured, collected from different sources in different formats. The established reference architecture for financial services with the document-oriented database

MongoDB connected through an Apache Spark connector to Hadoop works with data in a document version.

This type of unstructured data is typical of many services in core banking, e-payments, stocks, bonds, personalization, e-wallet, insurance, customer clearing, customer management and others. For this reason, the reference architecture created has been tested with data such as contracts, slips, emails, reports, stock, and bond data, and more. Part of this data is mock-up, generated additionally, because the data with which financial services work is extremely sensitive and cannot be found with free access.

- **Testing reference architecture with financial data**
  - **Source of financial data – shares**

Data about and related to customers that is generated, used and stored in financial institutions is extremely sensitive and should not be freely distributed, and should also be stored according to the requirements for the protection of personal data. Financial data can be very hard to find freely on the internet, and for testing purposes, freely available stock data from Forbes, Nasdaq, Nyse and SP500 in json format was used.

Forbes, Nasdaq, Nyse and SP500 data contain the following fields:

Field	Value
currency	Съответната валута
symbol	Символ
exchangeName	Обменно име
instrumentType	Тип инструмент
firstTradeDate	Първа дата на продаване
regularMarketTime	Регулярно време на пазара
gmtoffset": -18000	
timezone	Времева зона
exchangeTimezoneName	Времева зона на обмен
regularMarketPrice	Редовна цена на пазара
chartPreviousClose	
priceHint	Подсказка за цената
currentTradingPeriod	Настоящ период за търгуване
pre	
timezone	Часова зона на предварителната продажба
start	Стартова дата

end	Крайна дата
gmtoffset	Разлика от времето по Гринуич
regular	
timezone": "EST",	Часова зона на регулярна продажба
"start": 1670855400,	Стартова дата
"end": 1670878800,	Крайна дата
gmtoffset	Разлика от времето по Гринуич
post	
timezone	Часова зона след продажба
start	
end	Крайна дата
gmtoffset	Разлика от времето по Гринуич
dataGranularity	Детайлност на данните
range	Диапазон
validRanges	Валидни времеви диапазони

Table 1 Fields with of Forbes, Nasdaq, Nyse и SP500

Stock data from the various sources - Forbes, Nasdaq, Nyse and SP500 is over 10GB, which will be loaded into the architecture that is realized in this chapter.

- **Loading financial data**

The created architecture can exchange data from MongoDB through the MongoDB connector to Apache Spark and vice versa. Loading data from Hadoop to Mongo is implemented using PySpark.

To do this, it is first necessary to add the necessary libraries and create a Spark session with the following code:

```
spark = SparkSession.builder.FinancialTest("HadoopToMongoDB")
.config("spark.mongodb.output.uri","mongodb://localhost:27017/FinancialData.Stocks ")
.getOrCreate()
```

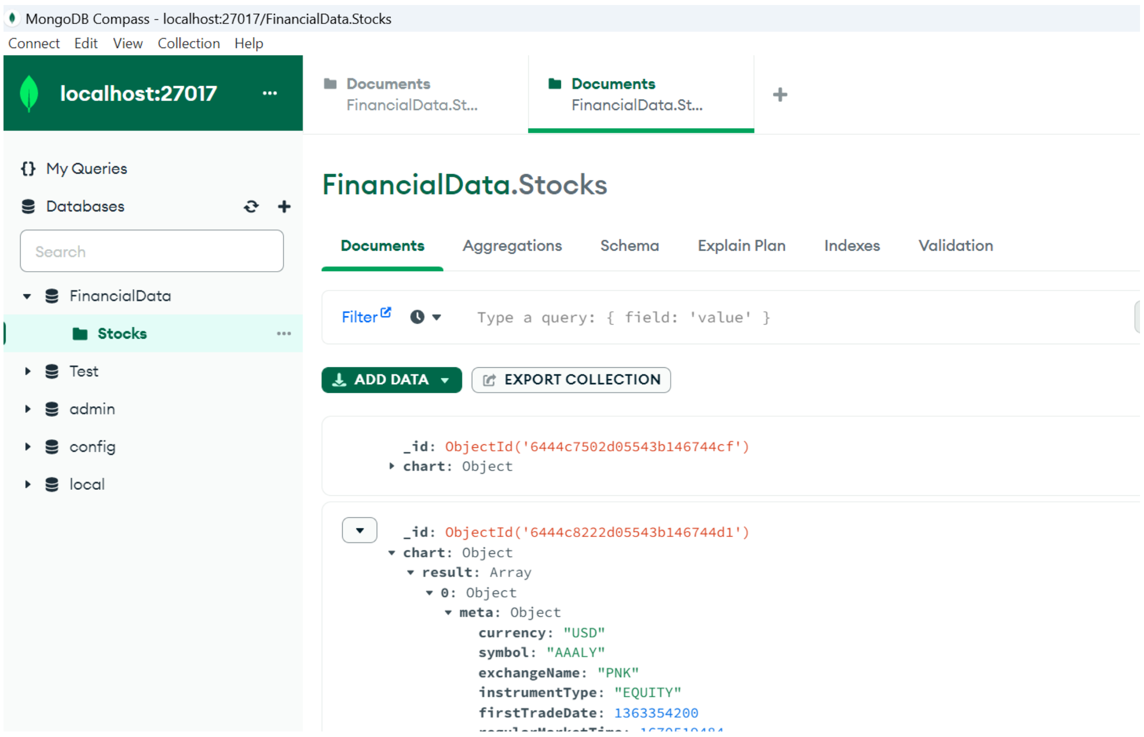
With the code below, we load the data from Hadoop into PySpark Frame:

```
df = spark.read.format("com.mongodb.spark.sql.DefaultSource").option("uri", "mongodb://
http://localhost:50070/FinancialData.Stocks.load()
```

Finally, we write the data from Hadoop to MongoDB, using the following code:

```
df.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").option("uri", "mongodb://localhost:27017/FinancialData.Stocks") .save()
```

Figure 11 visualizes the data loaded through the Apache Spark connector to MongoDB from Hadoop.



```
_id: ObjectId('6444c7502d05543b146744cf')
chart: Object

_id: ObjectId('6444c8222d05543b146744d1')
chart: Object
  result: Array
    0: Object
      meta: Object
        currency: "USD"
        symbol: "AAALY"
        exchangeName: "PNK"
        instrumentType: "EQUITY"
        firstTradeDate: 1363354200
```

Figure 10 Loaded data from Hadoop to MongoDB

```
client = MongoClient()

db = client['Financial_data']

collection = db['stocks']


spark = SparkSession.builder

    .appName("Financial Stock")

    .getOrCreate()


df = spark.read.format("com.mongodb.spark.sql.DefaultSource")

    .option("database", "Financial_data")

    .option("collection", "stocks")
    .load()



df = df.filter((col("symbol") == "ACGL") & (col("firstTradeDate") >= "2017-05-02") &
(col("firstTradeDate") <= "2017-10-10"))



result = df.groupBy(year("date").alias("year"), month("date").alias("month"))
    .agg({"close": "avg"})  .orderBy("year", "month")


result.write.format("com.mongodb.spark.sql.DefaultSource").option("database",
"Financial_data") .option("collection", "stock_analysis") .mode("overwrite") .save()


client.close()
spark.stop()
```

The generated PySpark program uses the stock data that was loaded at the start of the reference architecture experiment from Hadoop to MongoDB. The first step is to connect to MongoDB and then open the Spark session. The data is filtered by the name of the shares, and subsequently only those whose first sale date is between the set interval are searched. The program then groups the result by date and outputs the average value of the price of the particular stock for the entire period in the interval. The data is saved to the MongoDB database and the session is closed.

Източници

Kate Blumberg, “7 Big Data Use Cases in Financial Services and Benefits of Data Science,” SAFEGRAPH, Nov. 04, 2021. https://www.safegraph.com/blog/top-big-data-use-cases-financial-services (Opened on Feb 24, 2022)

“Big Data in Finance - Your Guide to Financial Data Analysis,” Talend. https://www.talend.com/resources/big-data-finance/ (Opened on June 15, 2022)

Mark Smallcombe, “Structured vs Unstructured Data: 5 Key Differences,” Integrate.io, Feb. 16, 2023. https://www.integrate.io/blog/structured-vs-unstructured-data-key-differences/ (Opened on June 18, 2022)

“What is Apache Hadoop?,” IBM. https://www.ibm.com/analytics/hadoop (Opened on August 29, 2022)

“Advantages of Hadoop | Disadvantages of Hadoop,” RF Wireless World. https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Hadoop.html

“Apache Hadoop Architecture Explained,” Phoenixnap, May 25, 2020. https://phoenixnap.com/kb/apache-hadoop-architecture-explained (Opened on September16, 2022)

“Banking industry architecture,” IBM. https://www.ibm.com/cloud/architecture/architectures/banking/reference-architecture (Opened on October 2, 2022)

“MongoDB for Financial Services,” MongoDB. https://www.mongodb.com/industries/financial-services (Opened on October 2, 2022)

“Hadoop and MongoDB,” MongoDB. https://www.mongodb.com/hadoop-and-mongodb (Opened on February 11, 2023)

“DATA PLATFORMS IN FINANCIAL SERVICES.” Medici. [Online]. Available: https://content.dataversity.net/rs/656-WMW-918/images/Data-Platforms-in-Financial-Services-NoSQL-Edge-Whitepaper.pdf (Opened on February 27, 2023)

“MongoDB Connector for Apache Spark,” MongoDB. https://www.mongodb.com/products/spark-connector (Opened on March 11, 2023)

MEDICI, “Why Financial Services Should Look to NoSQL.” [Online]. Available: http://pages.aerospike.com/rs/229-XUE-318/images/Aerospike_Wp_Why_Financial_Services_Should_Look_to_NoSQL.pdf (Opened on March 11, 2023)