The whole project has been broken down into several microservices.
Following is a brief summary of the microservices:

1.IncomingTradesToKafka Microservice :

This microservice reads the input trades from the XML files, converts them into String format and publishes those strings to a Kafka topic “GenericTradeTopic” on the kafka cloud server being used for this project.

The conversion of XML files to strings is done using various classes and libraries including the DocumentBuilder, Transformer, StringUtils etc.

The concept of multithreading has been used in order to achieve parallel processing of the XML files and publishing into Kafka topic.

A WatcherService has also been incorporated in order to monitor the directory where the trade XML files are stored. This service monitors the directory for creation of new XML files and if any XML file is created in the directory(symbolising a new incoming trade being encountered), it performs the same operations on that file and publishes the trade to the kafka topic in a similar fashion.

JUnit test case has been written for the Microservice.

The service is configured using Spring Cloud Config.

The source code has been uploaded on the GitHub repository whose link has been submitted.

2.GenericTradeObject Microservice:

The microservice consumes the Trade string form Kafka topic “GenericTradeTopic” and convert it to the java object of GenericTrade and publishes the object to the Kafka topic “GenericTradeObject” on the Kafka cloud server being used for this project.

This microservice has a Kafka Consumer which consumes to the Generic trade string and makes a call the Controller which further calls the ControllerService to convert the string to java object, then the controller calls the producer to publish the object to the kafka topic.

Jackson library is used to make a custom serializer class for the producer.

3.ExceptionListCreator Microservice:

This microservice can be seen as a combination of two different microservices

3.1 Kafka to Cache Microservice:

This microservice reads the generic Trade Object Stored in kafka topic which for further ease is stored in Cache.Out of different possible caching mechanisms,Redis cache has been chosen into which the trade object will be written as a hash set.This microservice is like a simple pathway to direct the object in kafka to cache.

3.2 Cache to Kafka Microservice:

This process consists of three steps:

1.Reading trade data that is being written to the cache within a span of time.

2.Comparing the read data with the mongoDB database and preparing a list of tradeIds which are already present in the database.

3.Processing the list of duplicate trades and sending the serialized trade objects to the kafka so that trades could be sent back to the user.

Technologies used:

- Spring Boot Module
- Redis cache mechanism
- MongoDB
- Kafka

The respective microservice is intended to be deployed to the cloud at the end.
Configuration for this microservice is done using SpringCloud Config.
The source code is uploaded on Github as well.

4.AckNack Generating Microservice:

This microservice consumes the exception trade object from the exception topic and generates a XML file i.e ack/nack and sends it to another kafka topic.

This microservice has a Kafka Listener which listens to the exception topic and when it detects new trade in the topic it makes a call to controller class which calls convert service that converts the trade object received into XML of required format using JAXB marshalling.

The final XML contains the full name of the firm along with asset description instead of the firm code and asset code present in the input trade.These details are fetched from a cache containing the names of all firms and assets.

Now the controller calls the producer to produce this XML to the ack-nack topic.This sums up the role of this microservice.

Jackson library is used to make a custom deserializer class for the consumer.

JUnit test case has been written for the Microservice.

Configuration for this microservice is done using SpringCloud Config

5.ReferenceStroring Microservice:

This microservice takes XML files containing the list of firm names and description and asset names and description and stores them in mongodb database.This database is connected to a redis cache and the AckNack Generation Microservice fetches the firm and asset description from this redis cache.
The microservice clears the cache after writing to the database.

JUnit test case has been written for the Microservice.

Configuration for this microservice is done using SpringCloud Config.

6.KafkaToFileConnector Microservice:

This microservice takes in the final Ack or Nack XML String to a Kafka Topic and stores them into XML files.

This has a Kafka listener which listens to the AckNack topic for getting the ack nack XML Strings.

These Strings are then written to an XML file using the FileWriter class in Java.

JUnit test case has been written for the Microservice.

Configuration for this microservice is done using SpringCloud Config

The source code is uploaded in the GitHub repository.

7.TradeProcessingConfigServer Microservice:

This is a server application for Spring Cloud Config.

This points to the directory where the Config files for each microservice are stored. The path of that directory needs to be provided in the properties file of this service.

The config files are Uploaded on github within a folder ‘ConfigDir’

The source code is uploaded on Github as well.

Rule Engine -> A Test Project to learn something new

We created a basic and lightweight rule engine to validate the data coming in trade.xml project ….for example
TradeId-must be alphanumeric,of length at least 20,Mandatory
DateTime-Time and Date should be in correct format and date should be a valid date
We implemented rule engine in two languages:

Python Implementation:

Name- RuleEngineInPython

Just import it in Spyder IDE and run SimpleRuleEngine.py file

Java Implementation:

Name- RuleEngineInJava

Just import it in STS and run as a Spring Boot Application.