Distributed Banking System - Practice

In this exercise we will build a Distributed Banking System

**Background**

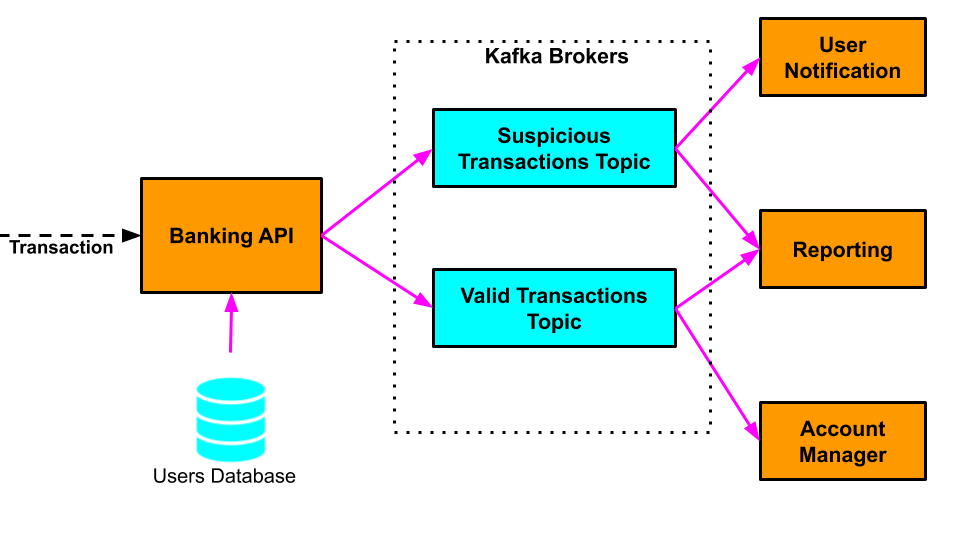
Let's assume we have a large number of users. Each user has an account with our bank and was also issued a credit card to make purchases in stores.

If our bank suspects a particular purchase transaction may be fraudulent, that transaction is flagged and sent to the user as a notification to approve or reject.

The way our security detects suspicious transactions is by comparing the user's country of residence and the location from which the transaction was made. If the user's residence country and the transaction's location don't match, the transaction is flagged.

**Architecture**

In our distributed system we will have multiple micro-services.



* ***Banking API Service*** - Receives credit card transactions from online and physical stores. Each **transaction** contains the following information
  1. **User** - the user who (allegedly) made the purchase in that particular store
  2. **Amount** - the amount of the purchase transaction in dollars
  3. **Transaction location**- the country in which that purchase has been made
* ***Users Database***- A database where we store each of our bank users' residence location
* ***User Notification Service***- for every transaction this service receives, it sends a notification to the user who allegedly made the purchase with a request to log in the banking web site and confirm or reject the transaction
* **Account Manager Service**- every transaction that this service receives is a valid transaction. The service authorizes the transaction and transfers the money from the user's account to the store where the purchase was made
* **Reporting Service** - For every transaction that has been processed by the *Banking API Service*, the *Reporting Service* stores the transaction for:
  1. Further investigation (if it is a suspicious transaction) or;
  2. For the user's monthly statement (if it is a valid transaction)

All the communication inside our Distributed System is done through Kafka Topics

**Assignment Part 1 - Launching Kafka**

1. Download Kafka to your computer from this [link](https://archive.apache.org/dist/kafka/2.2.0/kafka_2.12-2.2.0.tgz).
2. Navigate to the downloaded file, and unpack it.
3. Configure and start zookeeper as we learned in the lecture.
4. Launch 3 Kafka brokers listening on ports 9092, 9093 and 9094.
5. Create a new Kafka topic called **valid-transactions** with replication factor of 3 and 3 partitions.
6. Create a new Kafka topic called **suspicious-transactions** with replication factor of 3 and 2 partitions.
7. Validate that all the topics are created correctly.

**Assignment Part 2 - Implementation of the Microservices in Java**

1. Download the**banking-system-exercise.zip**from this lecture's resources
2. Fill in the code in the *Banking API Service, Account Manager Service, User Notification Service* and *Reporting Service*. (It is recommended to follow that order)
3. Build and package each application.
4. Run all the services with*Banking API Service* being last
5. Observe that the correct transactions are routed to the corresponding services.

See solution in the next lecture.

Have fun:)

Distributed Banking System - Solution

#### ****Distributed Banking System - Solution****

#### ****Part 1 - Launching Kafka****

# Instructions on Starting [Kafka](https://archive.apache.org/dist/kafka/2.2.0/kafka_2.12-2.2.0.tgz) to our computer we will start with configuring Apache Zookeeper:

1. Create **zookeeper-logs**directory :

mkdir zookeeper-logs

2. Open **config/zookeeper.properties** and modify **dataDir** to point to**zookeeper-logs**(full path)

3. Run Zookeeper

Unix: /bin/zookeeper-server-start.sh kafka/config/zookeeper.properties

Windows: /bin/windows/zookeeper-server-start.bat config/zookeeper.properties

4. Create 3 directories, one for each Kafka broker logs:

mkdir kafka-logs-0

mkdir kafka-logs-1

mkdir kafka-logs-2

Each directory will contain the corresponding Kafka broker's logs.

5. Create 2 copies of  **config/server.properties** and name them **config/server-1.properties** and **config/server-2.properties**. Rename the original**config/server.properties** to **config/server-0.properties**.

6. Change each configuration file's**log.dirs** to point to the corresponding **kafka-logs-X** directory:

In **config/server-0.properties**change**log.dirs**  to point to kafka-logs-0

In **config/server-1.properties**change**log.dirs**  to point to kafka-logs-1

In **config/server-2.properties**change**log.dirs**  to point to kafka-logs-2

7. In **server-1.properties** uncomment the #listeners=PLAINTEXT://:9092 and change the port to 9093

**listeners=PLAINTEXT://:9093**

8. In **server-2.properties** uncomment the #listeners=PLAINTEXT://:9092 and change the port to 9094

**listeners=PLAINTEXT://:9094**

Now our all our 3 Kafka broker's configuration files are ready so we go ahead to run the brokers

9. Run 3 Kafka brokers pointing to the corresponding properties files:

Unix:

bin/kafka-server-start.sh config/server-0.properties &

bin/kafka-server-start.sh config/server-1.properties &

bin/kafka-server-start.sh config/server-2.properties &

Windows:

bin/windows/kafka-server-start.bat config/server-0.properties &

bin/windows/kafka-server-start.bat config/server-1.properties &

bin/windows/kafka-server-start.bat config/server-2.properties &

Now that our Kafka cluster is running, lets create the Kafka topics:

10. Create **valid-transactions** Kafka topic:

Unix:

bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic valid-transactions

Windows:

bin/windows/kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic valid-transactions

11. Create **suspicious-transactions** Kafka topic:

Unix:

bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 2 --topic suspicious-transactions

Windows:

bin/windows/kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 2 --topic suspicious-transactions

We can check that our topics where indeed created by running:

Unix:

bin/kafka-topics.sh --list --bootstrap-server localhost:9092

Windows:

bin/windows/kafka-topics.bat --list --bootstrap-server localhost:9092

#### ****Part 2 - Launching Kafka****

Please see the attached **banking-system-solution.zip**for the implementation of theBanking API Service, Account Manager Service, User Notification Service and Reporting Service.