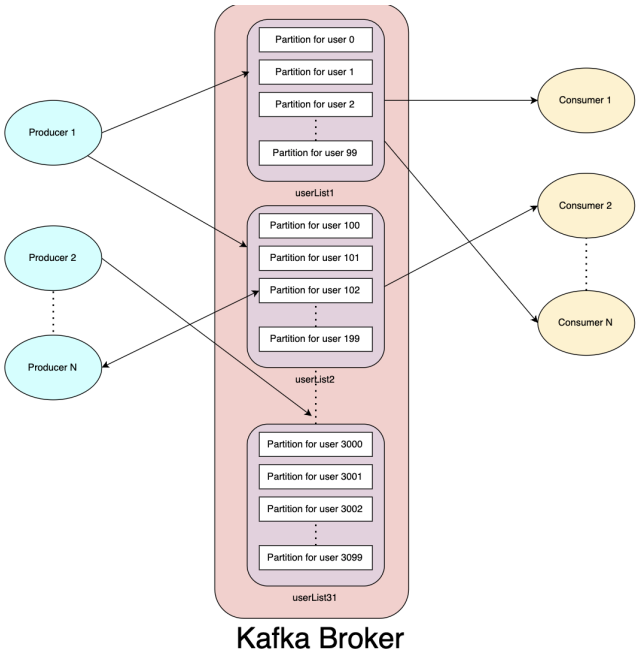
project_detail

Kafka Structure and Simulated Data



In this Tech Challenge, I used Kafka's standalone mode, which means there is only 1 Kafka Broker. There are 100 partitions for each topic, and these partitions are organized based on user IDs. For instance, if a user has the ID 210, their data will be stored in the 11th partition of the "userList2" topic. (userList2 starts with userID 200) Similarly, if a user has the ID 350, their information will be found in the 51st partition of the "userList3" topic. Each partition represent a specific user and contain all of their historical transaction records. Due to limitations in my computer's performance during testing, I've created 31 topics with a total of 3100 partitions. Each partition holds transaction data spanning from January to July of 2022, with the number of transactions ranging from 1000 to 3000 per month. The transaction records adhere to the specified format provided by the website's guidelines.

Kafka Search

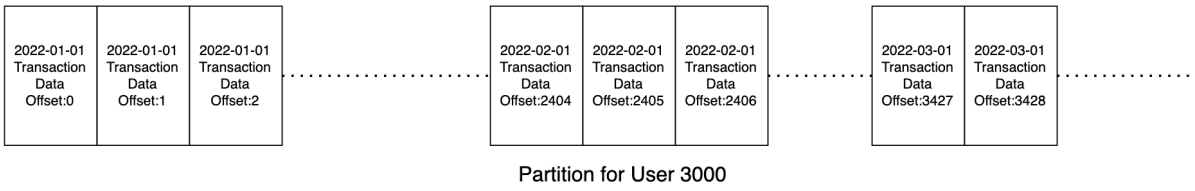
To search for a month’s transaction records, it is inefficient to loop through all transaction datas in a partition. Instead, my solution first checks the offset 0 and find the transaction date at offset 0. Then it compares with the date you want to search for and calculate the month difference. It then goes to month_diff * 1000 offset and find the date at that offset. If the month difference is at least 2, then offset += (month_diff-1). Repeat the process until the offset does not change any more.

This iterative process continues until the offset stabilizes. Subsequently, the offset is incremented by 100 each time to check if the transaction date at that particular offset corresponds to the desired

month. If a match is found, it indicates that the start offset for the desired date falls within the range of `current_offset - 100` to `current_offset`.

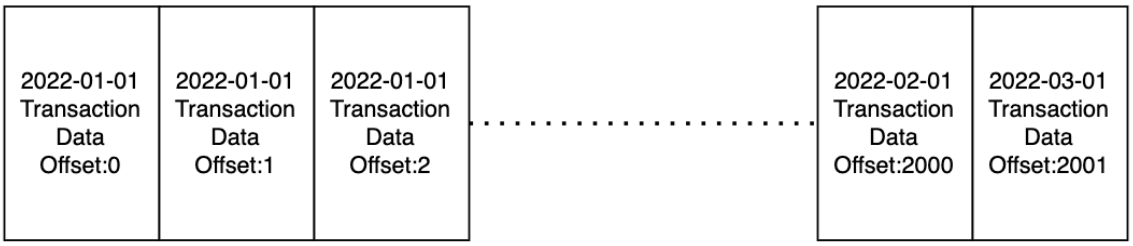
The final step involves fetching 100 records at a time and identifying the starting offset. Additionally, this starting offset is stored in Redis for a specific partition and month. This enables the system to quickly retrieve the information for subsequent searches. If the Redis cache contains the required key, the system can directly return the stored start offset.

For instance, consider the partition for user 3000. If the objective is to search for transactions in March 2022, the process begins at offset 0, which contains data from January 2022. Calculating a month difference of 2, the program moves to offset 2000. At offset 2000, the transaction is still in January and still has a month difference of 2, so the program lands on offset 3000(2000+(2-1)*1000), which still holds data from February 2022. This offset serves as a reference point, and the offset is then incremented by 100 for further checks. Eventually, at offset 3500, the transaction date matches the target month, indicating that the start offset falls between 3400 and 3500. By examining records within this range, the starting offset is determined to be 3427.



Why (month_diff-1) * 1000 instead of month_diff * 1000

To prevent edge cases such as below. If we want to find the transaction start from 2022-03-01. We start from offset 0, and calculate the month difference with 2022-03-01 and 2022-01-01. We get month_diff as 2 and go to offset 2 * 1000 = 2000. At this offset, the date is 2022-02-01, and still has a 1 month difference with 2022-03-01. If we use month_diff * 1000, the offset will be jumped to 2000 + 1 * 1000 = 3000. However, the correct offset is 2001.



Partition Example

Paginated List Return

The system is designed to provide 10 records per page. When retrieving a particular page number for a given month, the system initiates by determining the starting offset for that month and adding `10 * page_number` to it. This approach ensures that the system accurately captures the desired page's starting point. To address scenarios where the page number exceeds the total number of pages for the specific month, the program also calculates the starting offset for the subsequent month. This calculation assists in defining the limit and enables the system to handle cases where the page number extends beyond the available data for the chosen month.

Currency Exchange API

The system use third party provider for retrieving real time currency exchange rate. The url is `http://api.exchangeratesapi.io` and it can retrieve over 100 currency rates based on Euro.


```
{
    "success": true,
    "timestamp": 1692497583,
    "base": "EUR",
    "date": "2023-08-20",
    "rates": {
        "AED": 3.999374,
        "AFN": 91.564377,
        "ALL": 106.17029,
        "AMD": 419.882504,
        "ANG": 1.957252,
        "AOA": 901.564314,
        "ARS": 380.089039,
        "AUD": 1.707423,
        "AWG": 1.959922,
        "AZN": 1.85536,
        "BAM": 1.955253,
        "BBD": 2.192786,
        "BDT": 118.866737,
        "BGN": 1.961995,
```


Currency Exchange process

The system undertakes the conversion of total credit and debit values into the current exchange rate of the Hong Kong Dollar. Recognizing the relatively stable nature of currency exchange rates within shorter time intervals, the system optimizes efficiency by maintaining a repository of all exchange rates within Redis. This repository is regularly updated with real-time exchange rate data every 2 hours. Following the retrieval of the paginated list of a user's transactions for a designated month, the system then engages with Redis to access and convert the exchange rate between the currency associated with each transaction and the Hong Kong Dollar (HKD).

Authentication and Authorization

The system uses Spring Security and Spring Oauth2 to generate JWT token.

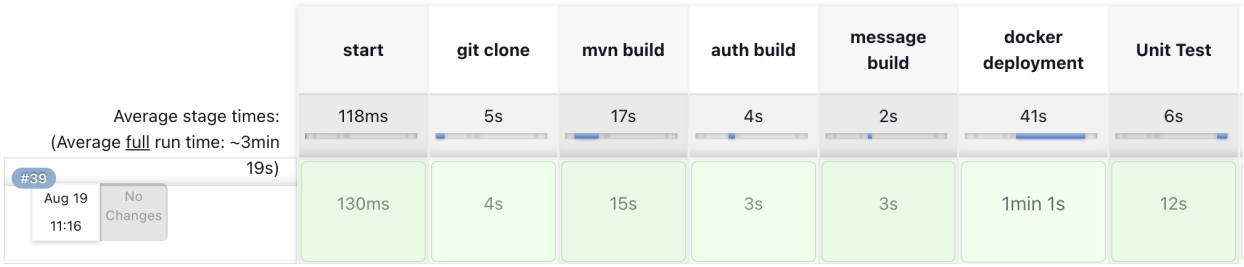
To retrieve transaction records, users **must** attach their jwt tokens in http header as “Authorization: bearer jwt_token”.

Unit Test

The current unit test covers 19 cases, and it covers normal transactions, out of page, out of month, out of year, out of userID transactions. The unit test is located inside Transaction/src/test/java/com/synpulse/DeployUnitTest.java.

Jenkins Pipeline

The system employs a Jenkins pipeline to facilitate seamless continuous integration. The pipeline encompasses several critical stages, beginning with the cloning of the project's source code from GitHub. Subsequently, the pipeline starts the compilation and installation of the project to ensure its proper setup. Moreover, the pipeline creates Docker images for both the authentication (auth) and transaction components, subsequently deploying these images within the Docker environment. The final phase involves the execution of comprehensive unit tests to rigorously validate the integrity and functionality of the developed solution. This end-to-end Jenkins pipeline integration ensures a systematic and automated approach to project development and validation. The code for Jenkins pipeline is attached in appendix



Appendix

```
pipeline {
    agent any
```

```
    stages{
        stage('start') {
            steps {
                echo "start passed successfully"
            }
        }
        stage('git clone') {
            steps {
                git branch: 'main', url: 'git@github.com:hu611/Kafka_For_Message_Retrieval.git'
            }

        }
        stage('mvn build') {
            steps {
                script {
                    dir('Parent') {
                        sh "mvn clean install -DskipTests"
                    }
                }
            }
        }
        stage('auth build') {
            steps {
                script {
                    echo "building auth images"
                    sh "docker build -t auth_image ./Auth"
                }
            }
        }
        stage('message build') {
            steps {
                script {
                    echo "building message images"
                    sh "docker build -t transaction_image ./Transaction"
                }
            }
        }
        stage('docker deployment') {
            steps {
                script {
                    echo "deploying auth"
                    sh "nohup docker run -p 3002:3002 auth_image &"
                    echo "deploying transaction"
                    sh "nohup docker run -p 3012:3012 transaction_image &"
                    sleep(time: 60, unit: 'SECONDS')
                }
            }
        }
        stage('Unit Test') {
            steps {
                script {
                    dir('Transaction') {
                        def testExitCode = sh(script: "mvn -Dtest=DeployUnitTest test", returnStatus: true)
                        if (testExitCode != 0) {
                            error "Unit tests failed"
                        }
                    }

                }
            }
        }
        stage('User Input') {
```

```
            steps {
                input "Please confirm to proceed"
            }
        }
    }
}
```