{"@timestamp":"2024-05-28T04:28:54.297Z","@version":"1","message":"Produced: {\"requestId\": \"b8ed4087-6304-412c-ae30-23e9c31063d2\", \"customerId\": \"1701921677932589\", \"paymentTypeLimits\": {\"duitnow\_acc\_intrabank\": \"5000.00\", \"duitnow\_qr\": \"5000.00\"}, \"categoryTypeLimits\": {\"cat\_duitnow\_acc\_proxy\": \"50000.00\"}, \"status\": \"SUCCESS\"}","logger\_name":"com.arbm.digibank.accounts.savingaccount.service.SavingAccountParameterService","thread\_name":"org.springframework.kafka.KafkaListenerEndpointContainer#1-0-C-1","level":"INFO","level\_value":20000,"springAppName":"deposit-account-service","spanId":"2c969d72afd8c144","bc":"deposits"}

//Start sending reports, a mark that the processing ended

//Its log time indicates roughly the time the emails were sent

fields *@timestamp*, *@message*, kubernetes.container\_name

| filter kubernetes.container\_name = "deposit-commodity-trading-service"

| filter strcontains(*@message*, "Send trade order request:")

| sort *@timestamp* desc

| display *@timestamp*, kubernetes.container\_name

| limit 1000

//See the customer command records received on the internal Kafka topics

//The number of records corresponds to the number of customers

fields *@timestamp*, *@message*, kubernetes.container\_name

| filter kubernetes.container\_name = "deposit-commodity-trading-service"

| filter strcontains(*@message*, "Received command:")

| sort *@timestamp* desc

| display *@timestamp*, kubernetes.container\_name

| limit 1000

//To see the processing of each customer

fields *@timestamp*, *@message*, kubernetes.container\_name

| filter kubernetes.container\_name = "deposit-commodity-trading-service"

| filter strcontains(*@message*, "Calculated Customer Murabaha for customer")

| parse log "stdout F \*" as logBody

| display *@timestamp*, kubernetes.container\_name, logBody

| sort *@timestamp* desc

| limit 1000

* Everything happens in the PreProd environment so we can definitely test it there. Our goal will be that everything literally stays the same, but it happens faster.
* The Flow starts with the schedular called “CommodityTradingSchedular” starting to initiate the trade order by calling “initiateTradeOrder” function.
* For each customer a Kafka message is sent to the “customer-murabaha-command.topic” topic which is an internal topic that no one else uses except the microservice.
* A set of operations is performed on the command of the customer every single customer, and then this data is then saved to the database.
* The following steps are taken before sending the report, each one can potentially be improved:
  + Collecting all Customers who have balances. (line 36 in MurabhaService)
  + Check if customer has been processed (line 53 MurabhaService)
  + Get Customer balances (line 54 Murabha Service)
  + Saving in CustomerDailyTradeRepository after mapping (line 56 MurabhaService)
  + Checking if the last message in the customer data is the command (line 57 MurabhaService)
  + Aggregate Customer Murabha (line 58 MurabhaService)