



INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Informática

Enterprise Integration

MEIC / METI 2024-2025 – 4th Period

Enterprise Integration Report

IST number	Name	Email
103668	David Palma	david.palma@tecnico.ulisboa.pt
110948	Renato Custódio	renato.miguel.monteirinho.custodio@tecnico.ulisboa.pt

Contents

This report aims to explain the developed solution for the 2nd Sprint of the project, which builds upon the already existing codebase provided by Scenario 3 and the microservices implemented in the 1st Sprint. For this sprint, we leveraged those microservices and integrated them into business processes defined in Camunda, while also employing Kong as the API gateway and adding all necessary components to connect and execute these workflows. Additionally, we further developed Kafka, which was a bit underdeveloped in the first sprint. This report includes documentation on this project's testing, source code, terraform scripts files, installation procedures and parametrization.

Kafka Cluster

In order to ensure fault tolerance, availability, and reliable leader election, even in systems with low throughput, a Kafka cluster should be composed of at least three brokers. This configuration allows the cluster to maintain a quorum-based consensus, which is essential for operations such as topic management and partition leadership. With three brokers, Kafka can tolerate the failure of one node while continuing to operate normally, making the system resilient to downtime and enabling safe rolling updates. Additionally, it allows for balanced distribution of partitions, ensuring that no single broker becomes a bottleneck.

The opted solution to implement this was to simultaneously configure all the internal details of each one of the EC2 instances. It has the advantage of accelerating the manual configurations by automation, and at scale. The drawback is the complexity.

Therefore, after launching the cluster, the two global configurations that previously required manual setup on each EC2 instance (as shown in the image below) are now fully automated, eliminating the need for any manual configuration on individual machines.

At /usr/local/kafka/config/server.properties

```
#zookeeper connectivity (one per EC2 VM of this cluster)
zookeeper.connect=ec2-54-90-57-82.compute-1.amazonaws.com:2181,ec2-54-173-171-63.compute-1.amazonaws.com:2181,ec2-54-236-47-54.compute-1.amazonaws.com:2181
```

And at /usr/local/zookeeper/conf/zoo.cfg

```
server.1=ec2-54-90-57-82.compute-1.amazonaws.com:2888:3888
server.2=ec2-54-173-171-63.compute-1.amazonaws.com:2888:3888
server.3=ec2-54-236-47-54.compute-1.amazonaws.com:2888:3888
```

Kafka Topics

Regarding the creation of topics, all of these topics are now automatically created by the provisioning script.

- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 3 --topic DiscountCoupon`

This topic also uses 3 partitions, which is adequate for the anticipated low message volume. It allows for even distribution across the 3 brokers, maintaining fault tolerance while keeping the setup efficient for a lightweight service.

- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 4 --topic CrossSellingRecommendation`

We opted for 4 partitions to accommodate higher processing demands, as cross-selling recommendations may require more computational resources. This setup enables greater parallelism and better scalability for heavier workloads.

- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 6 --topic SelledProduct-Coupon`
- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 6 --topic SelledProduct-Customer`
- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 6 --topic SelledProduct-Location`
- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 6 --topic SelledProduct-LoyaltyCard`
- `sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <address>:<port> --replication-factor 3 - -partitions 6 --topic SelledProduct-Shop`

The topics for SelledProduct topics are configured with 6 partitions each to support higher throughput and concurrency.

These topics are expected to handle larger message volumes and more intensive analytics workloads, requiring greater parallelism.

To check if the topics were created:

- `sudo /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server <address>:<port>`

Kafka Automation:

To achieve this automation we use a `null_resource` in terraform. Once the EC2 instances have been successfully created by Terraform, the `null_resource` is responsible for applying additional configuration and setup tasks that cannot be performed directly through the `aws_instance` resource.

For each broker, this `null_resource` establishes a secure SSH connection to the newly deployed EC2 instance using the private key specified in the configuration. It then transfers a custom configuration script, rendered from a template and tailored to the specific broker's settings, to the EC2's temporary directory. After the script has been successfully copied, the `null_resource` executes it remotely on the EC2, allowing the broker to be configured and integrated into the Kafka cluster.

The shell script("configure.sh") that is applied remotely on each EC2 instance performs a series of configuration and startup tasks to bring the Kafka broker and Zookeeper into service and prepare the Kafka ecosystem for use.

The script starts by waiting for Zookeeper's configuration file (zoo.cfg) and Kafka's configuration file (server.properties) to become available. Once these files are present, it checks whether the required settings, namely the Zookeeper server IDs and the Zookeeper connect string, are already configured. If not, it appends these settings to the respective files to match the configuration of the current Kafka broker.

After making sure the configuration files are properly updated, the script restarts Zookeeper, first attempting a graceful shutdown and then starting it again. It performs health checks to verify Zookeeper is up and running. Once Zookeeper is healthy, the script restarts Kafka, first killing any existing Kafka process and then starting a new Kafka broker instance in daemon mode.

With both Zookeeper and Kafka up and running, the script then waits for all brokers to become available. It checks this by repeatedly retrieving the IDs of brokers registered in Zookeeper and comparing them against the expected number of brokers in the Kafka cluster. Once all brokers are present, the script proceeds to create a set of predefined Kafka topics. It performs this step from the broker acting as the Zookeeper leader to avoid attempting to create the same topics multiple times.

The topics it configures, such as DiscountCoupon, CrossSellingRecommendation, SoldProductByCoupon, and SoldProductByLocation, are all created with a replication factor matching the number of brokers in the Kafka cluster.

Finally, it's important to highlight that all automation is designed to adapt dynamically to the number of brokers. When verifying whether all brokers are up, it checks exactly the number of brokers that were launched, no more, no less. Similarly, when creating topics, the replication factor is set to match the broker count, ensuring it never exceeds the number of available brokers.

Kafka Cluster Testing:

```
null_resource.configure_kafka[2] (remote-exec): ⌚ Waiting for brokers... current IDs: [1, 2]
null_resource.configure_kafka[0] (remote-exec): ⌚ Waiting for brokers... current IDs: [1, 2]
null_resource.configure_kafka[1] (remote-exec): ✅ All 3 brokers are up: [1, 2, 3]
null_resource.configure_kafka[1] (remote-exec): Mode: leader
null_resource.configure_kafka[1] (remote-exec): Creating topics...
null_resource.configure_kafka[0] (remote-exec): ✅ All 3 brokers are up: [1, 2, 3]
null_resource.configure_kafka[0]: Creation complete after 2m27s [id=629970651908531627]
null_resource.configure_kafka[1] (remote-exec): Created topic LoyaltyCard-at-shop.
null_resource.configure_kafka[2] (remote-exec): ✅ All 3 brokers are up: [1, 2, 3]
null_resource.configure_kafka[2]: Creation complete after 2m28s [id=3362539075914375856]
null_resource.configure_kafka[1] (remote-exec): Created topic DiscountCoupon.
null_resource.configure_kafka[1]: Still creating... [2m30s elapsed]
null_resource.configure_kafka[1] (remote-exec): Created topic CrossSellingRecommendation.
null_resource.configure_kafka[1]: Creation complete after 2m33s [id=549920329987188945]
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS
[ec2-user@ip-172-31-93-244 ~]$ sudo /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server ec2-3-95-230-185.compute-1.amazonaws.com:9092
CrossSellingRecommendation
DiscountCoupon
LoyaltyCard-at-shop
```

```
LoyaltyCard-at-shop
[ec2-user@ip-172-31-93-244 ~]$ sudo /usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server ec2-3-95-230-185.compute-1.amazonaws.com:9092
Topic: CrossSellingRecommendation TopicId: NhD2sD7SRJ-JFd7y8b7Xlw PartitionCount: 4 ReplicationFactor: 3 Configs:
: N/A Topic: CrossSellingRecommendation Partition: 0 Leader: 3 Replicas: 3,1,2 Isr: 3,1,2 Elr: N/A LastKnownElr
: N/A Topic: CrossSellingRecommendation Partition: 1 Leader: 1 Replicas: 1,2,3 Isr: 1,2,3 Elr: N/A LastKnownElr
: N/A Topic: CrossSellingRecommendation Partition: 2 Leader: 2 Replicas: 2,3,1 Isr: 2,3,1 Elr: N/A LastKnownElr
: N/A Topic: CrossSellingRecommendation Partition: 3 Leader: 3 Replicas: 3,2,1 Isr: 3,2,1 Elr: N/A LastKnownElr
: N/A
Topic: LoyaltyCard-at-shop TopicId: UB0N1qmBQj-iW5A0UGo0xQ PartitionCount: 3 ReplicationFactor: 3 Configs:
Topic: LoyaltyCard-at-shop Partition: 0 Leader: 2 Replicas: 2,3,1 Isr: 2,3,1 Elr: N/A LastKnownElr: N/A
Topic: LoyaltyCard-at-shop Partition: 1 Leader: 3 Replicas: 3,1,2 Isr: 3,1,2 Elr: N/A LastKnownElr: N/A
Topic: LoyaltyCard-at-shop Partition: 2 Leader: 1 Replicas: 1,2,3 Isr: 1,2,3 Elr: N/A LastKnownElr: N/A
Topic: DiscountCoupon TopicId: luQRdpHgTwitofp924xwiA PartitionCount: 3 ReplicationFactor: 3 Configs:
Topic: DiscountCoupon Partition: 0 Leader: 1 Replicas: 1,2,3 Isr: 1,2,3 Elr: N/A LastKnownElr: N/A
Topic: DiscountCoupon Partition: 1 Leader: 2 Replicas: 2,3,1 Isr: 2,3,1 Elr: N/A LastKnownElr: N/A
Topic: DiscountCoupon Partition: 2 Leader: 3 Replicas: 3,1,2 Isr: 3,1,2 Elr: N/A LastKnownElr: N/A
```

As shown in the pictures above, all three Kafka brokers were successfully created. The topics are also being created correctly, and communication between the brokers is functioning as expected. This is confirmed by the use of a replication factor of 3 during topic creation, which ensures that all brokers are participating. Since no errors have occurred, we can conclude that the brokers are communicating properly.

As we can see in the pictures below, all topics that are automated are being created successfully.

```
[ec2-user@ip-172-31-81-24 ~]$ sudo /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092
CrossSellingRecommendation
DiscountCoupon
```

```
SelledProductByCoupon  
SelledProductByCustomer  
SelledProductByLocation  
SelledProductByLoyaltyCard  
SelledProductByShop
```

The consuming and producing of events are addressed later in this report, where the testing procedures and overall results of the system are presented.

Kong Automation

Since we opted to configure Kong by sending curl requests to set up the services and routes, we incorporated this process directly into the `deploy.sh` script. Once Kong is up and running, the script proceeds to configure its services and routes. It begins by printing the IP addresses of all relevant microservices, such as Customer, Purchase, Shop, Loyalty Card, Cross Selling Recommendation, Discount Coupon, Selled Product Analytics, and Ollama (AI). This step serves to verify that the environment variables holding the IP addresses are correctly set, providing visibility into the services that will be connected through Kong.

Before configuring each service, the script checks whether the corresponding environment variable containing the service's IP address is not empty. Only if the variable is set and contains a valid address does the script proceed to create the service and its route in Kong. This conditional check prevents attempts to configure routes for services that are not deployed or whose IPs are unavailable, thereby avoiding potential errors.

The script performs a health check by continuously attempting to connect to Kong's Admin API at `http://localhost:8001`, waiting in a loop and retrying every five seconds until Kong confirms it is ready to accept requests. Once Kong is confirmed operational, for each valid microservice, the script registers the service by specifying its name and the URL where it is hosted, usually combining the container's IP address with the appropriate port and path. Following this, it sets up a route that defines the path prefix through which the service can be accessed via Kong's API Gateway. This approach allows all microservices to be accessed in a unified manner through Kong, simplifying client interactions by consolidating endpoints under a single gateway.

Notably, for the Discount Analysis AI service, which relies on Ollama and may involve longer processing times, the script applies extended connection, read, and write timeouts of 180 seconds to ensure reliable communication despite potential latency.

To configure the variables that store the addresses of the microservices for Kong, we use the command `sed -i "/^<name_var>=/c <name_var>=\"\$addressMS\""/././kongVars.sh`. Here, addressMS is dynamically obtained by executing the command `$(terraform state show aws_instance.exampleDeployQuarkus | grep public_dns | sed "s/public_dns//g" | sed "s/=//g" | sed "s/\\//g" | sed "s/ //g" | sed "s/$esc[[[0-9;]*m//g")"` for each service. This command extracts and cleans the public DNS address from the Terraform state output. Before launching Kong with Terraform, we source KongVars.sh to update the variables accordingly. The variables are then exported within the script to Terraform, ensuring that Kong's service addresses remain up-to-date and accurately reflect the deployed infrastructure.

As we can see in the pictures below all the routes and services are being correctly set up.

Services

```
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-94-8 ~]$ curl -s http://localhost:8001/services
{"data":[{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/Purchase","updated_at":1749937953,"port":8080,"id":"047f3c4f-a3ee-4f56-ad8c-d16161c8a3b0","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"purchase-service","retries":5,"protocol":"http","host":"ec2-3-83-29-115.compute-1.amazonaws.com"},{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/SelledProductAnalytics","updated_at":1749937953,"port":8080,"id":"0ef1a6e7-eafe-4167-a9fc-5a8d6f285b7c","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"selledProductAnalytics-service","retries":5,"protocol":"http","host":"ec2-44-202-147-67.compute-1.amazonaws.com"},{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/Shop","updated_at":1749937953,"port":8080,"id":"241cdd9d-a5ea-40e9-8587-d718fa16c826","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"shop-service","retries":5,"protocol":"http","host":"ec2-3-90-50-18.compute-1.amazonaws.com"},{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/CrossSellingRecommendation","updated_at":1749937953,"port":8080,"id":"5c2cf292-e1c9-4053-a108-c8a971957845","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"crossSellingRecommendation-service","retries":5,"protocol":"http","host":"ec2-44-206-236-232.compute-1.amazonaws.com"},{"connect_timeout":180000,"read_timeout":180000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/api/generate","updated_at":1749937953,"port":11434,"id":"7d355650-7784-479f-bac2-e9cd969b293d","client_certificate":null,"enabled":true,"write_timeout":180000,"name":"ollama-service","retries":5,"protocol":"http","host":"ec2-54-163-63-209.compute-1.amazonaws.com"},{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/LoyaltyCard","updated_at":1749937953,"port":8080,"id":"89d5b9c2-4303-4a45-a879-587d04a73479","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"loyaltyCard-service","retries":5,"protocol":"http","host":"ec2-44-210-141-199.compute-1.amazonaws.com"},{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/Customer","updated_at":1749937953,"port":8080,"id":"b48cda70-afff-4894-80ea-830470239f0b","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"customer-service","retries":5,"protocol":"http","host":"ec2-3-84-53-224.compute-1.amazonaws.com"},{"connect_timeout":60000,"read_timeout":60000,"tls_verify":null,"tls_verify_depth":null,"created_at":1749937953,"tags":null,"ca_certificates":null,"path":"/DiscountCoupon","updated_at":1749937953,"port":8080,"id":"b51cb03d-3d4e-4dbb-ac8e-dbd4a0c51ca4","client_certificate":null,"enabled":true,"write_timeout":60000,"name":"discountCoupon-service","retries":5,"protocol":"http","host":"ec2-44-206-236-232.compute-1.amazonaws.com"}]}
```

Routes

```
[ec2-user@ip-172-31-94-8 ~]$ curl -s http://localhost:8001/routes
{"data":[{"strip_path":true,"service":{"id":"89d5b9c2-4303-4a45-a879-587d04a73479"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/loyaltyCard"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"loyaltyCard-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"0acaef55-01a7-4af6-96c5-d10f35a00f61"},"methods":null,"strip_path":true,"service":{"id":"241cdd9d-a5ea-40e9-8587-d718fa16c826"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/shop"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"shop-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"2b139558-3ab1-449d-99dc-5175f7426b68"},"method_s":null,"strip_path":true,"service":{"id":"5c2cf292-e1c9-4053-a108-c8a971957845"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/crossSellingRecommendation"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"crossSellingRecommendation-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"45d9841d-544e-4506-9266-517ee8833c85"},"methods":null,"strip_path":true,"service":{"id":"7d355650-7784-479f-bac2-e9cd969b293d"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/ollama"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"ollama-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"5b23b95a-9a69-4c33-bf57-4fe7c4cecb8d"},"methods":null,"strip_path":true,"service":{"id":"b48cda70-afff-4894-80ea-830470239f0b"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/customer"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"customer-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"6d073063-acf1-4e91-a97a-c59586310480"},"methods":null,"strip_path":true,"service":{"id":"b51cb03d-3d4e-4dbb-ac8e-dbd4a0c51ca4"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/discountCoupon"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"discountCoupon-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"047f3c4f-a3ee-4f56-ad8c-d16161c8a3b0"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/purchase"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"purchase-route","destinations":null,"protocols":["http","https"],"response_buffering":true,"id":"b34ad5fc-512a-4d8e-ac91-7cddf44244ef"},"methods":null,"strip_path":true,"service":{"id":"0ef1a6e7-eafe-4167-a9fc-5a8d6f285b7c"},"request_buffering":true,"snis":null,"preserve_host":false,"hosts":null,"tags":null,"paths":["/selledProductAnalytics"],"https_redirect_status_code":426,"updated_at":1749937953,"regex_priority":0,"headers":{"source":"v0"},"sources":null,"path_handling":"v0","created_at":1749937953,"name":"selledProductAnalytics-route","destinations":null,"protocols":["http","https"]}]}
```


Deployment

Initially, we had a single deployment script that was responsible for launching all the services at once. However, due to AWS account limitations, which restrict us to a maximum of nine instances per account, we decided to split the deployment into two separate scripts in order to distribute the services across two different accounts.

The first script handles the deployment of the RDS database, all microservices except for Ollama, and the Kafka cluster. We kept these components together because we already had an automated way to launch them collectively, and many of them needed to communicate directly with each other.

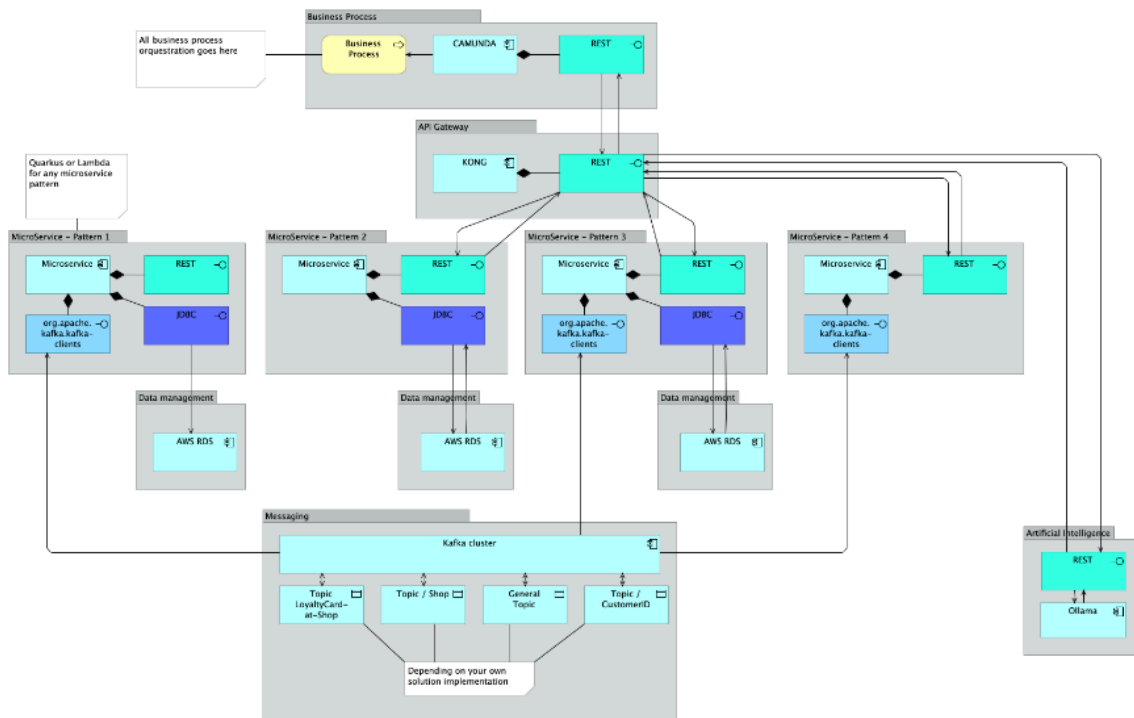
The second script covers the deployment of Ollama, Kong, and Camunda. To maintain the functionality of Kong's automated configuration, we transfer the "KongVars.sh" file, generated by the first script, to the second account. This way, when we launch the second script, all the necessary environment variables are already set, allowing Kong to be configured automatically and seamlessly integrate with the rest of the services.

All of these deployment scripts have matching undeployment scripts, making it easy to destroy and clean up the resources when necessary. All these scripts have been thoroughly tested and were successfully used to launch the system for the testing procedures described further down in this report.

Finally, it's also important to note that we added a new variable to the "access.sh" script to store the path to the SSH key used to access the remote machines in AWS. We use this key during the Kafka provisioning process to connect securely to those machines and perform the necessary setup.

Infrastructure

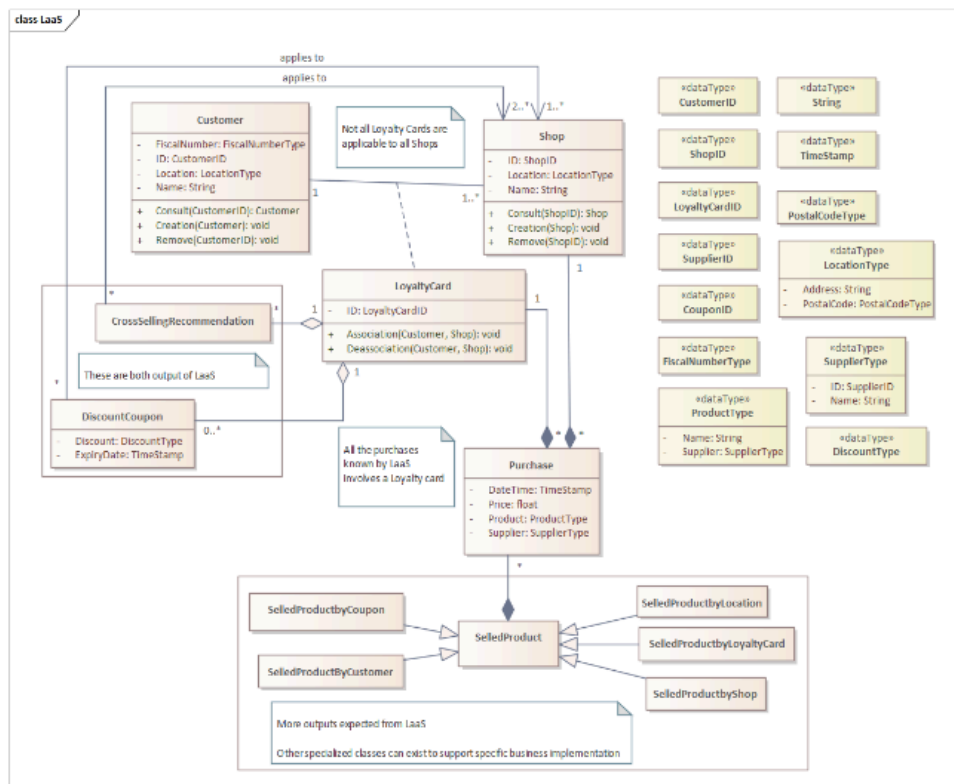
The adopted infrastructure was the one mentioned in the application integration reference architecture:



The business processes, running on Camunda Engine, make HTTP requests when needed to a Kong API Gateway, which in turn has configured Services and Routes that redirect the requests to the desired microservices. Each one of these microservices may then persist information in a shared AWS RDS or even produce Kafka events to a running Kafka cluster, as well as consume Kafka topics and be notified of specific events.

This infrastructure, albeit being simple, promotes microservice scalability and management, due to the decoupling of the microservices and to the advantages that the Kong API Gateway provides, such as the ease to add Services and Routes while running to redirect requests to additional microservices.

Domain Model



The adopted domain model was based on the already existing one provided by S3-Integration-Scenario. However, some changes were made in order for it to correctly meet the business specification.

Firstly, and the most important change, was regarding the mapping between the entities Customer, Shop and LoyaltyCard. In the provided codebase, a LoyaltyCard was associated with the pair CustomerID - ShopID. However, this went against the specified business requirements, where a Customer has a single, unique LoyaltyCard that is able to be used in a multitude of Shops. Therefore, the LoyaltyCard was updated to have a List of ShopIDs rather than a single ShopID. This was also reflected in the PUT /loyaltyCard/{id} endpoint, where now it receives a single ShopID which is then added to the already existing list of ShopIDs of the provided LoyaltyCard.

Secondly, regarding the entity Purchase, we added the field 'Long discountCouponId' with the objective of identifying whether a purchase was made using or not a DiscountCoupon. This proved useful in the SoldProductAnalytics and in the DiscountCouponAnalysisUsingAI BPMNs.

Thirdly, regarding the entity CrossSellingRecommendation, we added the field 'String recommendation' to effectively represent the recommendation of an item in several shops as described in the project statement.

Finally, regarding the entity Shop, we added the UNIQUE constraint to the Shop name, since we thought it made sense and it was useful to the SoldProductAnalytics BPMN.

Microservices

This section aims to describe all the developed microservices, as well as describe the modifications pertaining to the already existing microservices. As declared in the project statement, the available microservices are:

- Shop
- Customer
- LoyaltyCard
- Purchase
- DiscountCoupon
- CrossSellingRecommendation
- SoldProductAnalytics
- Discount Coupon analysis using Artificial Intelligence/Ollama

Shop

Provides CRUD operations regarding the Shop entity. This was an already existing microservice. An endpoint was added where a Shop is fetched by its name (3).

Endpoints:

1. GET /Shop
2. GET /Shop/{id}
3. GET /Shop/name/{ShopName}
4. POST /Shop
5. DELETE /Shop/{id}
6. PUT /Shop/{id}

Customer

Provides CRUD operations regarding the Customer entity. This was an already existing microservice.

Endpoints:

1. GET /Customer
2. GET /Customer/{id}
3. POST /Customer
4. DELETE /Customer/{id}
5. PUT /Customer/{id}

LoyaltyCard

Provides CRUD operations regarding the LoyaltyCard entity. This was an already existing microservice. As previously mentioned, the entity LoyaltyCard was modified to be single and unique to a customer, so some endpoints were modified. Firstly, the endpoint where a LoyaltyCard was fetched with the pair CustomerId-ShopId was replaced by an endpoint where only the former is provided (3). Furthermore, the endpoint where a LoyaltyCard was updated was modified to be able to add shops to a LoyaltyCard (6).

Endpoints:

1. GET /LoyaltyCard
2. GET /LoyaltyCard/{id}
3. GET /LoyaltyCard/customerId/{idCustomer}
4. POST /LoyaltyCard
5. DELETE /LoyaltyCard/{id}
6. PUT /LoyaltyCard/{id}

Purchase

Provides CRUD operations regarding the Purchase entity. This was an already existing microservice. As previously mentioned, a field was added to this entity, which justified an additional endpoint where purchases regarding a LoyaltyCard that used a DiscountCoupon are fetched (4). An endpoint where purchases are fetched by LoyaltyCard was also added (3).

Endpoints:

1. GET /Purchase
2. GET /Purchase/{id}
3. GET /Purchase/loyaltyCardId/{idLoyaltyCard}
4. GET /Purchase/loyaltyCardId/{idLoyaltyCard}/usedCoupons
5. POST /Purchase/Consume

DiscountCoupon

Provides CRUD operations regarding the DiscountCoupon entity. Basic CRUD operations were implemented. When creating a DiscountCoupon, an event is produced to the static Kafka topic 'DiscountCoupon' (6).

Endpoints:

1. GET /DiscountCoupon
2. GET /DiscountCoupon/{id}
3. GET /DiscountCoupon/loyaltyCardId/{idLoyaltyCard}
4. DELETE /DiscountCoupon/{id}
5. PUT /DiscountCoupon/{id}
6. POST /DiscountCoupon

CrossSellingRecommendation

Provides CRUD operations regarding the CrossSellingRecommendation entity. Basic CRUD operations were implemented. When creating a CrossSellingRecommendation, an event is produced to the static Kafka topic 'CrossSellingRecommendation' (5).

Endpoints:

1. GET /CrossSellingRecommendation
2. GET /CrossSellingRecommendation/{id}
3. DELETE /CrossSellingRecommendation/{id}
4. PUT /CrossSellingRecommendation/{id}
5. POST /CrossSellingRecommendation

SelledProductAnalytics

Provides CRUD operations regarding the SelledProductAnalytics entity. Basic CRUD operations were implemented. When creating a SelledProductAnalytics, an event is produced to 5 different static Kafka topics (5) :

- SelledProductByCoupon
- SelledProductByShop
- SelledProductByLocation
- SelledProductByCustomer
- SelledProductByLoyaltyCard

Endpoints:

1. GET /SelledProductAnalytics
2. GET /SelledProductAnalytics/{id}
3. GET /SelledProductAnalytics/purchase/{idPurchase}
4. DELETE /SelledProductAnalytics/{id}
5. POST /SelledProductAnalytics

Discount Coupon analysis using Artificial Intelligence/Ollama

Opposite to others, this microservice is not implemented in Quarkus but instead implemented by a single Ollama server. Only one endpoint is used in this microservice.

Endpoints:

1. POST /api/generate

Microservices Testing

Each microservice was tested manually using **CURLs** via the Postman app, both locally and when running on AWS instances. Additionally, each microservice has **JUnit** tests that aim to test the Resource Layer of the microservices, that is, the layer that exposes their endpoints.

Finally, their integration with **Kong** and **Camunda** was also successfully tested, as will be shown in the Camunda/BPMNs section. Furthermore, it was also observed that data was correctly persisted in the **AWS RDS**, as well as the whole interaction with **Kafka** which encompassed both the production and the consumption of Kafka Events to Kafka Topics.

Camunda/BPMNs

This section explains every developed BPMN, each aiming to provide business value to the LaaS. These were:

- **DiscountCouponEmission**
- **CrossSellingRecommendation**
- **SelledProductAnalytics**
- **DiscountCouponAnalysisUsingAI**

This section also approaches the modifications that were made to the already existing BPMNs, for these to align themselves with the business specification. The only modified BPMN was the **LoyaltyCardManagement** BPMN, which originated 2 additional BPMNs: **CreateLoyaltyCard** and **UpdateLoyaltyCard**.

DiscountCouponEmission

“consists in the emission of a discount coupon, encompassing the analysis of purchases of a given customer, that was associated with a loyalty card, to all shops. All the rules for the discount coupon emission are up to be decided by each working group. The resulting coupon is produced to one specific Kafka topic with all required elements to identify it”

In this BPMN, a discountCoupon is emitted to a specific customer. For that purpose, the customer catalog is firstly retrieved (1), and a user task assigns the desired customer based on the acquired list.

Afterwards, having chosen the customer, this BPMN retrieves the customer LoyaltyCard (2) and uses it to obtain the customer's purchases (3) .

Having obtained the customer purchases, a user task decides if the discountCoupon can be emitted or not. This is based on the rule that a discountCoupon can only be emitted if a customer has purchased **at least 3 products**.

If the discount can be emitted , a user task then decides the discount information, that is, the discount to be applied and its expiry date, tailored to the customer's purchases.

Finally, this BPMN creates the discountCoupon (4) with all the obtained information, and this action will both persist data in the AWS RDS as well as produce an event to the Kafka topic DiscountCoupon.

In this BPMN, the following endpoints were accordingly used:

1. GET http://<Kong>/customer
2. GET http://<Kong>/loyaltyCard/customerId/\${CustomerID}
3. GET http://<Kong>/purchase/loyaltyCardId/\${LoyaltyCardID}
4. POST http://<Kong>/discountCoupon

Below, screenshots were taken in order to demonstrate the ongoing process of emitting a discount-coupon.

Decide the Customer to emit the DiscountCoupon

DiscountCouponEmission

Set follow-up date Set due date Add groups Demo Demo

Form History Diagram Description

CustomerID

1

Save Complete

Decide Discount information

DiscountCouponEmission

Set follow-up date Set due date Add groups Demo Demo

Form History Diagram Description

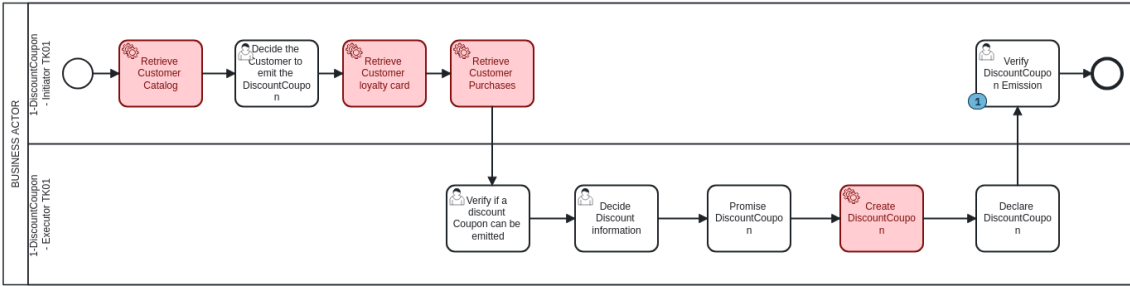
Discount

15%discount

ExpiryDate

2038-01-19T03:14:07

Save Complete



Variables	Incidents	Called Process Instances	User Tasks	Jobs	External Tasks
CustomerList			Json		Process (Discount...)
Discount			String		Process (Discount...)
ExpiryDate			String		Process (Discount...)
LoyaltyCardID			Integer		Process (Discount...)
PostStatusCode			String		Process (Discount...)
PurchasesList			Json		Process (Discount...)
emitCoupon			Boolean		Process (Discount...)

```
[ec2-user@ip-172-31-81-24 ~]$ /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092 --topic DiscountCoupon --from-beginning
{"Discount_Coupon":{"idLoyaltyCard":"S","discountType":"Percentage","expiryDate":"2038-01-19T03:14:07"}}
{"Discount_Coupon":{"idLoyaltyCard":"1","discountType":"15percentdiscount","expiryDate":"2038-01-19T03:14:07"}}
{"Discount_Coupon":{"idLoyaltyCard":"1","discountType":"15%discount","expiryDate":"2038-01-19T03:14:07"}}
```

CrossSellingRecommendation

“consists in the recommendation for cross selling of a given customer between different shops involving a single loyalty card, e.g., in collaborations, joint promotions, or shared marketing efforts. All the rules for the cross-selling recommendations are up to be decided by each working group. The resulting cross selling recommendation is produced to one specific Kaaa topic with all required elements to identify it”

In this BPMN, a crossSellingRecommendation is made to a specific customer, encompassing several shops. For that purpose, the customer catalog is firstly retrieved (1), followed by the shop catalog (2). A user task then assigns the both desired customer and shops based on the acquired lists.

Afterwards, having chosen the customer and its shops, this BPMN retrieves the customer LoyaltyCard (3) and uses it to obtain the customer's purchases (4) .

Having obtained the customer purchases, a user task decides the recommendation to be made, ideally tailored to the customer's purchases.

If the crossSellingRecommendation can be made , this BPMN creates the crossSellingRecommendation (5) with all the obtained information, and this action will both persist data in the AWS RDS as well as produce an event to the Kafka topic CrossSellingRecommendation.

In this BPMN, the following endpoints were accordingly used:

1. GET http://<Kong>/customer
2. GET http://<Kong>/shop
3. GET http://<Kong>/loyaltyCard/customerId/\${CustomerID}
4. GET http://<Kong>/purchase/loyaltyCardId/\${LoyaltyCardID}
5. POST http://<Kong>/crossSellingRecommendation

Below, screenshots were taken in order to demonstrate the ongoing process of making a crossSellingRecommendation.

Decide the data to order a CrossSellingRecommendation

CrossSellingRecommendation

Set follow-up date

Set due date

Add groups

Demo Demo

Form History Diagram Description

CustomerID

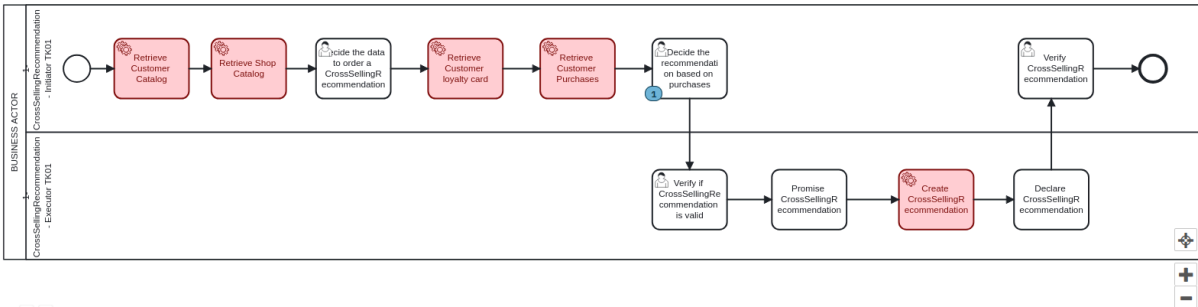
1

ShopID1

1

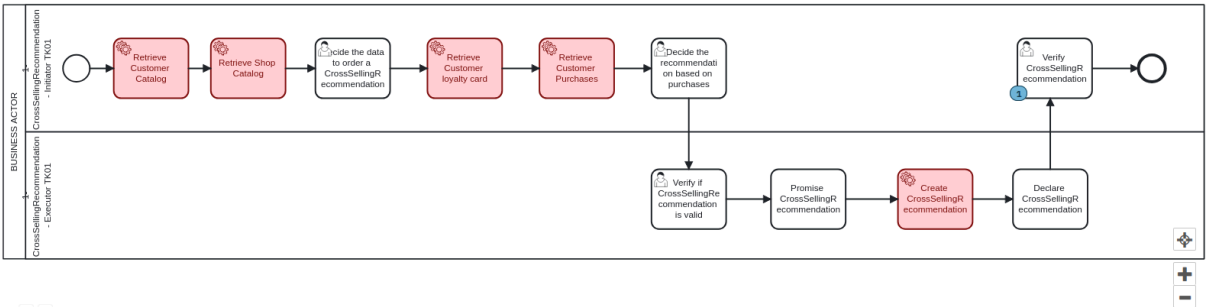
ShopID2

2



Variables Incidents Called Process Instances User Tasks Jobs External Tasks

CustomerID	String	1	Process (CrossSel...)	✓	✕
CustomerList	Json	{{"id":1,"fiscalNumber":123456,"location*":Lisb...	Process (CrossSel...)	✓	✕
LoyaltyCardID	Integer	1	Process (CrossSel...)	✓	✕
PurchasesList	Json	{{"id":1,"timestamp*":2038-01-19T03:14:07*,"p...	Process (CrossSel...)	✓	✕
ShopID1	String	1	Process (CrossSel...)	✓	✕
ShopID2	String	2	Process (CrossSel...)	✓	✕
ShopList	Json	{{"id":1,"location*":Lisboa*,"name*":ArcoCegoLis...	Process (CrossSel...)	✓	✕



Variables Incidents Called Process Instances User Tasks Jobs External Tasks

PostStatusCode	String	201	Process (CrossSel...)	✓	✕
PurchasesList	Json	{{"id":1,"timestamp*":2038-01-19T03:14:07*,"p...	Process (CrossSel...)	✓	✕
Recommendation	String	Coffee	Process (CrossSel...)	✓	✕
ShopID1	String	1	Process (CrossSel...)	✓	✕
ShopID2	String	2	Process (CrossSel...)	✓	✕
ShopList	Json	{{"id":1,"location*":Lisboa*,"name*":ArcoCegoLis...	Process (CrossSel...)	✓	✕
isRecommendationValid	Boolean	true	Process (CrossSel...)	✓	✕

SelledProductAnalytics

“consists in the quantitative analysis of the purchases. All the rules for analysis are up to be decided by each working group, the suggestions presented in Figure 2 can be extended to other metrics. The resulting analytics is produced to one specific Kafka topic with all required elements to identify it.”

In this BPMN, an analysis on a SelledProduct is made. For that purpose, the purchases catalog is firstly retrieved (1). A user task then assigns the desired purchase based on the acquired list.

Afterwards, having chosen the purchase, this BPMN retrieves the purchase information (2) and uses it to obtain both the customerId (3) and the shopLocation (4) associated with this purchase .

Having obtained all the required information to provide sufficient analytics for this purchase, a user task verifies if all the information is correct.

If everything is correct,, this BPMN creates the SelledProductAnalytics (5) with all the obtained information, and this action will both persist data in the AWS RDS as well as produce an event to the 5 different Kafka topics:

- SelledProductByCoupon
- SelledProductByShop
- SelledProductByLocation
- SelledProductByCustomer
- SelledProductByLoyaltyCard

In this BPMN, the following endpoints were accordingly used:

1. GET http://<Kong>/purchase
2. GET http://<Kong>/purchase/\${PurchaseID}
3. GET http://<Kong>/loyaltyCard/\${LoyaltyCardID}
4. GET http://<Kong>/shop/name/\${ShopName}
5. POST http://<Kong>/selledProductAnalytics

Below, screenshots were taken in order to demonstrate the ongoing process of making a crossSellingRecommendation.

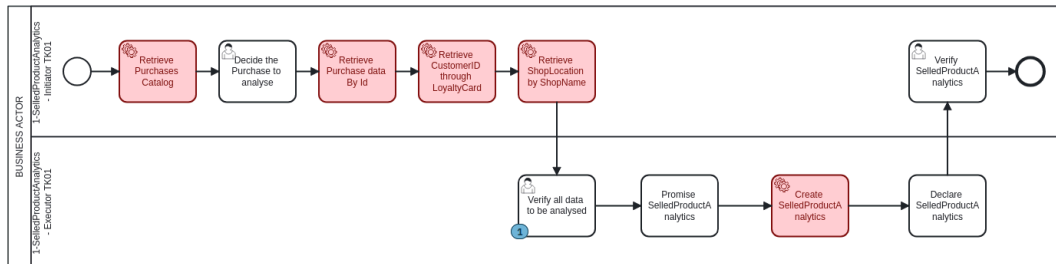
SelledProductAnalytics

 Set due date









 Demo Demo

Form History Diagram Description

Save Complete



Variables	Incidents	Called Process Instances	User Tasks	Jobs	External Tasks
CustomerID		Integer	1		Process (SelledPr...)
DiscountCouponID		Integer	2		Process (SelledPr...)
LoyaltyCardID		Integer	1		Process (SelledPr...)
PurchaseID		String	2		Process (SelledPr...)
PurchaseList		Json	{["id":1,"timestamp":"2038-01-19T03:14:07","p...}		Process (SelledPr...)
ShopLocation		String	Arco-do-Cego		Process (SelledPr...)
ShopName		String	Lisboa		Process (SelledPr...)

Variables	Incidents	Called Process Instances	User Tasks	Jobs	External Tasks
LooksGood			Boolean	true	Process (SelledPr...)  
LoyaltyCardID			Integer	1	Process (SelledPr...)  
PostStatusCode			String	201	Process (SelledPr...)  
PurchaseID			String	2	Process (SelledPr...)  
PurchaseList			Json	[{"id":1,"timestamp":"2038-01-19T03:14:07","p...	Process (SelledPr...)  
ShopLocation			String	Arco-do-Cego	Process (SelledPr...)  
ShopName			String	Lisboa	Process (SelledPr...)  

```
ec2-user@ip-172-31-81-24 ~]$ /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092 --topic SelledProductByCoupon --from-beginning
{"SelledProductAnalytics":{"idPurchase":"1","idCoupon":"8"}}
{"SelledProductAnalytics":{"idPurchase":"2","idCoupon":"2"}}
```

```
[ec2-user@ip-172-31-81-24 ~]$ ./usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092 --topic SelledProductByLocation --from-beginning
{"SelledProductAnalytics":{"idPurchase":"1","location":"Lisboa"}}
{"SelledProductAnalytics":{"idPurchase":"2","location":"Arco do Cego"}}
```

```
ec2-user@ip-172-31-81-24:~$ ./usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092 --topic SelledProductByCustomer --from-beginning
{"SelledProductAnalytics":{"idPurchase":"1","idCustomer":"10"}}
{"SelledProductAnalytics":{"idPurchase":"2","idCustomer":"11"}}
```

```
[ec2-user@ip-172-31-81-24 ~]$ /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092 --topic SoldProductByLoyaltyCard --from-beginning
{"SoldProductAnalytics":{"idPurchase":"1","idLoyaltyCard":"77"}}
{"SoldProductAnalytics":{"idPurchase":"2","idLoyaltyCard":"11"}}
```

```
[ec2-user@ip-172-31-81-24 ~]$ /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server ec2-34-201-52-83.compute-1.amazonaws.com:9092 --topic SoldProductByShop --from-beginning
{"SoldProductAnalytics":{"idPurchase":"2" "shopName":"Lisboa"}}
```

DiscountCouponAnalysisUsingAI

“consists in the quantitative analysis of the purchases that used the produced discount coupons. This is a business assessment whether the customers value the produced discount coupons by using them”

In this BPMN, an analysis is made regarding the discountCoupon usage of a customer. For that purpose, the customer catalog is firstly retrieved (1), and a user task assigns the desired customer based on the acquired list.

Afterwards, having chosen the customer, this BPMN retrieves the customer LoyaltyCard (2) and uses it to obtain the amount of discountCoupons that were emitted to his LoyaltyCard (3).

Having obtained the emitted discountCoupons to this customer, to observe if these were valued, this BPMN retrieves the amount of used DiscountCoupons (4).

If everything is correct, this BPMN aggregates this data and sends it to an Ollama server (5), which will then verify if these were or not valued based on redemption rate. This is the used prompt to achieve this purpose:

"You are a business-intelligence AI specialized in customer behavior analysis. You help companies assess the effectiveness and perceived value of their promotional coupons by examining quantitative purchase data. I have the following data for one customer: Total coupons issued: $\text{\$}\{\text{DiscountCouponsAmount}\}$ Coupons redeemed (purchases made with coupons): $\text{\$}\{\text{CouponsRedeemed}\}$ Determine whether this customer truly values the coupons based on redemption rate"

In this BPMN, the following endpoints were accordingly used:

1. GET `http://<Kong>/customer`
2. GET `http://<Kong>/loyaltyCard/customerId/{CustomerID}`
3. GET `http://<Kong>/discountCoupon/loyaltyCard/{LoyaltyCardID}`
4. GET `http://<Kong>/purchase/loyaltyCardId/{LoyaltyCardID}/usedCoupons`
5. POST `http://<Kong>/ollama`

Below, screenshots were taken in order to demonstrate the ongoing process of making a crossSellingRecommendation.

Decide the Customer to analyse discountCoupon usage

DiscountAnalysisUsingOllama

Set follow-up date

Set due date

Add groups

Demo Demo

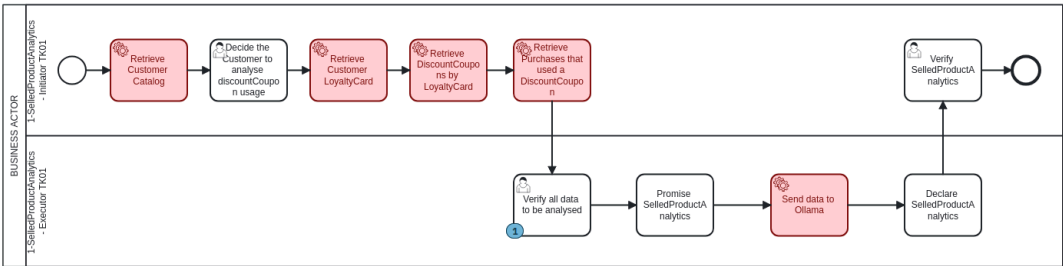
Form History Diagram Description

CustomerID

1

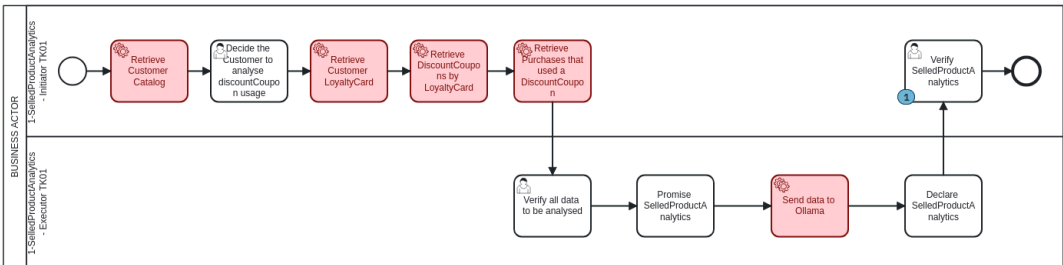
Save

Complete



Variables Incidents Called Process Instances User Tasks Jobs External Tasks

Add criteria				5			
Name	Type	Value	Scope	Actions			
CouponsRedeemed	Integer	2	Process (Discount...)				
CustomerID	String	1	Process (Discount...)				
CustomerList	Json	{["id":1,"fiscalNumber":123456,"location":"Lisb...	Process (Discount...)				
DiscountCouponsAmount	Integer	5	Process (Discount...)				
LoyaltyCardID	Integer	1	Process (Discount...)				



Variables Incidents Called Process Instances User Tasks Jobs External Tasks

CouponsRedeemed	Integer	2	Process (Discount...)			
CustomerID	String	1	Process (Discount...)			
CustomerList	Json	{{"id":1,"fiscalNumber":123456,"location":"Lisb...	Process (Discount...)			
DiscountCouponsAmount	Integer	5	Process (Discount...)			
LooksGood	Boolean	true	Process (Discount...)			
LoyaltyCardID	Integer	1	Process (Discount...)			
PostResponse	String	To determine if this customer truly values the co...	Process (Discount...)			

Inspect "PostResponse" variable

Value

Redemption rate:

$\text{Redemption Rate} = (\text{Number of Coupons Redeemed}) / (\text{Total Coupons Issued})$

In this case:

$\text{Redemption Rate} = 2 / 5$
 $= 0.4$ or 40%

A redemption rate of 40% indicates that this customer has found value in using their coupons, as they redeemed 40% of the total coupons issued.

Close

CreateLoyaltyCard

This BPMN is a modification to the previous LoyaltyCardManagement BPMN, with the purpose of correctly meeting the business specification regarding LoyaltyCards. Previously, a LoyaltyCard was the mapping between a customer and a shop, meaning that each customer could have several LoyaltyCards. However, the business specification states that each customer has a single, unique LoyaltyCard to be used in several shops.

That being said, each LoyaltyCard now holds a List of shopIds, but its creation is always regarding a single shop. An additional BPMN was developed to describe the business process of adding shops to a LoyaltyCard.

This BPMN deviates from the previous one in a single service task, 'Retrieve LoyaltyCardID', given that a LoyaltyCard is no longer obtainable by provisioning both CustomerId and a ShopId, being the former sufficient (5).

In this BPMN, the following endpoints were accordingly used:

1. GET http://<Kong>/shop
2. GET http://<Kong>/customer
3. POST http://<Kong>/loyaltyCard
4. GET http://<Kong>/shop/\${ShopID}
5. GET http://<Kong>/loyaltyCard/customerId/\${CustomerID}
6. POST http://<Kong>/purchase/Consume

Below, screenshots were taken in order to demonstrate the ongoing process of making a crossSellingRecommendation.

Decide the data to LoyaltyCard association order

BusinessActor1LoyaltyCardManagement

Set follow-up date

Set due date

Add groups

Demo Demo

Form History Diagram Description

CustomerID

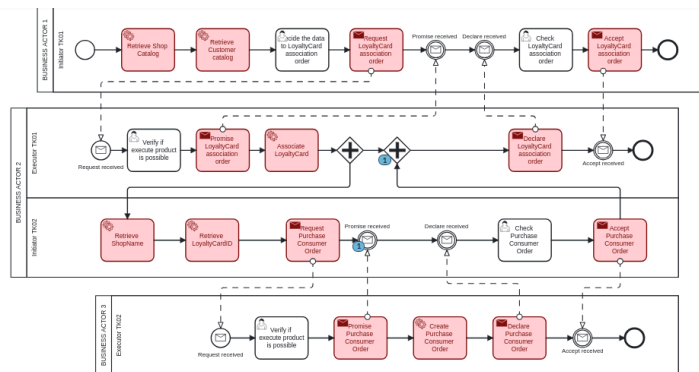
5

ShopID

1

Save

Complete



Variables	Incidents	Called Process Instances	User Tasks	Jobs	External Tasks
CustomerID		String	5		BusinessActor2LoyaltyCardManagement
LoyaltyCardID		Integer	5		BusinessActor2LoyaltyCardManagement
PostStatusCode		String	201		BusinessActor2LoyaltyCardManagement
ShopID		String	1		BusinessActor2LoyaltyCardManagement
ShopName		String	ArcoCegoLisbon		BusinessActor2LoyaltyCardManagement
businessKeyRequester		String	b		BusinessActor2LoyaltyCardManagement
promise		Boolean	true		BusinessActor2LoyaltyCardManagement

UpdateLoyaltyCard

This BPMN aims to complement the CreateLoyaltyCard BPMN. Given that the LoyaltyCard is a mapping between a customer and several shops, and it is instantiated with a single shop, we found it relevant to describe the business process where shops are added to a LoyaltyCard.

That being said, this BPMN is very similar to the previous one, but it differentiates from it on several points.

Firstly, the LoyaltyCard catalog is retrieved (1), followed by the shop catalog (2). A user task then assigns both the desired LoyaltyCard to update and the Shop to add based on the acquired lists.

Afterwards, having chosen the LoyaltyCard and the shop, this BPMN updates the LoyaltyCard, adding it the desired shop (3). The rest is identical to the previous BPMN, where a Purchase Consumer Order is issued for the pair LoyaltyCardId-ShopName.

In this BPMN, the following endpoints were accordingly used:

1. GET http://<Kong>/shop
2. GET http://<Kong>/loyaltyCard
3. PUT http://<Kong>/loyaltyCard/\${LoyaltyCardID}
4. GET http://<Kong>/shop/\${ShopID}
5. POST http://<Kong>/purchase/Consume

Below, screenshots were taken in order to demonstrate the ongoing process of making a crossSellingRecommendation.

Decide the data to update LoyaltyCard

BusinessActor1LoyaltyCardManagement

Set follow-up date

Set due date

Add groups

Demo Demo

Form History Diagram Description

LoyaltyCardID

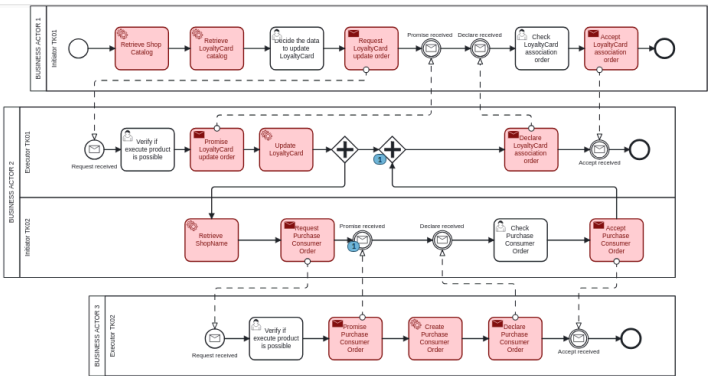
1

ShopID

4

Save

Complete



Variables Incidents Called Process Instances User Tasks Jobs External Tasks

Name	Type	Value	Scope	Actions
LoyaltyCardID	String	1	BusinessActor2LoyaltyCardManagement	
PutStatusCode	String	204	BusinessActor2LoyaltyCardManagement	
ShopID	String	4	BusinessActor2LoyaltyCardManagement	
ShopName	String	PracaDomFranciscoGomes	BusinessActor2LoyaltyCardManagement	
businessKeyRequester	String	g	BusinessActor2LoyaltyCardManagement	
promise	Boolean	true	BusinessActor2LoyaltyCardManagement	