**Projeto: Curso Udemy - Comunicação entre Microsserviços**

Repositório contendo o projeto desenvolvido do curso Comunicação entre Microsserviços, ministrado por mim para a plataforma Udemy.

## Tecnologias

* **Java 11**
* **Spring Boot**
* **Javascript ES6**
* **Node.js 14**
* **ES6 Modules**
* **Express.js**
* **MongoDB (Container e Cloud MongoDB)**
* **API REST**
* **PostgreSQL (Container e Heroku Postgres)**
* **RabbitMQ (Container e CloudAMQP)**
* **Docker**
* **docker-compose**
* **JWT**
* **Spring Cloud OpenFeign**
* **Axios**
* **Heroku**
* **Coralogix Logging**
* **Kibana**

## Arquitetura Proposta

No curso, desenvolveremos a seguinte aquitetura:

Diagrama

Descrição gerada automaticamente

Teremos 3 APIs:

* **Auth-API**: API de Autenticação com Node.js 14, Express.js, Sequelize, PostgreSQL, JWT e Bcrypt.
* **Sales-API**: API de Vendas com Node.js 14, Express.js, MongoDB, Mongoose, validação de JWT, RabbitMQ e Axios para clients HTTP.
* **Product-API**: API de Produtos com Java 11, Spring Boot, Spring Data JPA, PostgreSQL, validação de JWT, RabbitMQ e Spring Cloud OpenFeign para clients HTTP.

Também teremos toda a arquitetura rodando em containers docker via docker-compose.

### Fluxo de execução de um pedido

O fluxo para realização de um pedido irá depender de comunicações **síncronas** (chamadas HTTP via REST) e **assíncronas** (mensageria com RabbitMQ).

O fluxo está descrito abaixo:

* 01 - O início do fluxo será fazendo uma requisição ao endpoint de criação de pedido.
* 02 - O payload (JSON) de entrada será uma lista de produtos informando o ID e a quantidade desejada.
* 03 - Antes de criar o pedido, será feita uma chamada REST à API de produtos para validar se há estoque para a compra de todos os produtos.
* 04 - Caso algum produto não tenha estoque, a API de produtos retornará um erro, e a API de vendas irá lançar uma mensagem de erro informando que não há estoque.
* 05 - Caso exista estoque, então será criado um pedido e salvo no MongoDB com status pendente (PENDING).
* 06 - Ao salvar o pedido, será publicada uma mensagem no RabbitMQ informando o ID do pedido criado, e os produtos com seus respectivos IDs e quantidades.
* 07 - A API de produtos estará ouvindo a fila, então receberá a mensagem.
* 08 - Ao receber a mensagem, a API irá revalidar o estoque dos produtos, e caso todos estejam ok, irá atualizar o estoque de cada produto.
* 09 - Caso o estoque seja atualizado com sucesso, a API de produtos publicará uma mensagem na fila de confirmação de vendas com status APPROVED.
* 10 - Caso dê algum problema na atualização, a API de produtos publicará uma mensagem na fila de confirmação de vendas com status REJECTED.
* 11 - Por fim, a API de pedidos irá receber a mensagem de confirmação e atualizará o pedido com o status retornado na mensagem.

## Logs e Tracing da API

Todos os endpoints necessitam um header chamado **transactionid**, pois representará o ID que irá percorrer toda a requisição no serviço, e, caso essa aplicação chame outros microsserviços, esse **transactionid** será repassado. Todos os endpoints de entrada e saída irão logar os dados de entrada (JSON ou parâmetros) e o **transactionid**.

A cada requisição pra cada microsserviço, teremos um atributo **serviceid** gerado apenas para os logs desse serviço em si. Teremos então o **transactionid** que irá circular entre todos os microsserviços envolvidos na requisição, e cada microsserviço terá seu próprio **serviceid**.

Fluxo de tracing nas requisições:

**POST** - **/api/order** com **transactionid**: ef8347eb-2207-4610-86c0-657b4e5851a3

service-1:

transactionid: ef8347eb-2207-4610-86c0-657b4e5851a3

serviceid : 6116a0f4-6c9f-491f-b180-ea31bea2d9de

|

| HTTP Request

|----------------> service-2:

transactionid: ef8347eb-2207-4610-86c0-657b4e5851a3

serviceid : 4e1261c1-9a0c-4a5d-bfc2-49744fd159c6

|

| HTTP Request

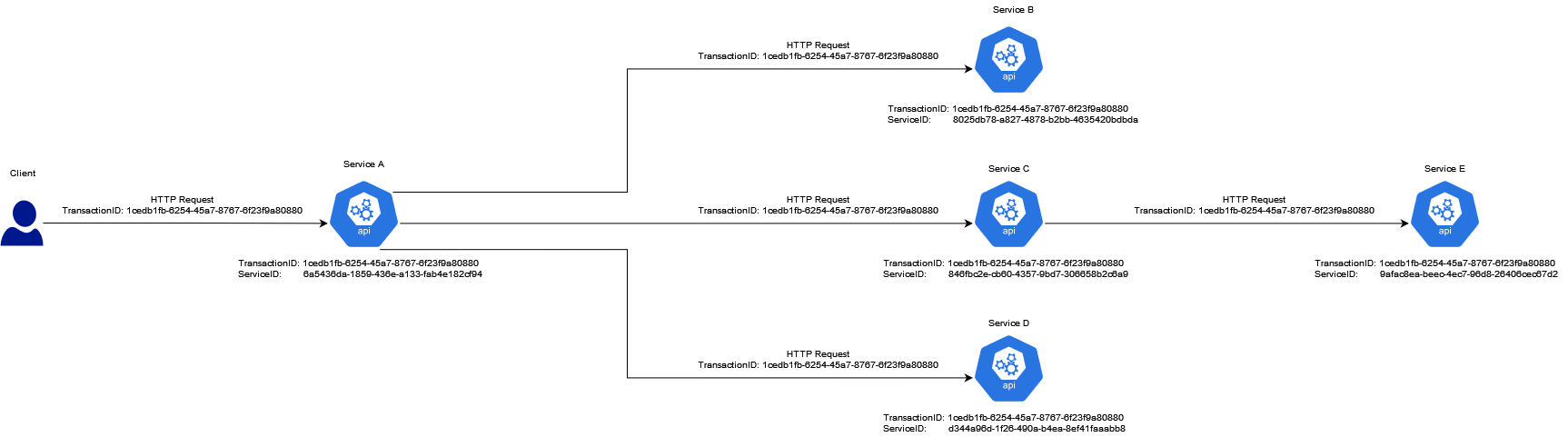
|----------------> service-3: /api/check-stock

transactionid: ef8347eb-2207-4610-86c0-657b4e5851a3

serviceid : b4fbc082-a49a-440d-b1d6-2bd0557fd189

Como podemos ver no fluxo acima, o **transactionid** ef8347eb-2207-4610-86c0-657b4e5851a3 manteve-se o mesmo nos 3 serviços, e cada serviço possui seu próprio **serviceid**.

Exemplo de um fluxo completo chamando 5 serviços e gerando **transactionid** e **serviceid**:



Exemplo de logs nas APIs desenvolvidas:

Auth-API:

Request to POST login with data {"email":"testeuser1@gmail.com","password":"123456"} | [transactionID: e3762030-127a-4079-9dee-ba961d7e77ce | serviceID: 6b07b6c2-009e-4799-be96-3bf972338b17]

Response to POST login with data {"status":200,"accessToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRoVXNlciI6eyJpZCI6MSwibmFtZSI6IlVzZXIgVGVzdCAxIiwiZW1haWwiOiJ0ZXN0ZXVzZXIxQGdtYWlsLmNvbSJ9LCJpYXQiOjE2MzQwNTE4ODQsImV4cCI6MTYzNDEzODI4NH0.NJ-h2i5XPT8NwZyZ\_43bif1NIS00ROfCtRecBkxy5A8"} | [transactionID: e3762030-127a-4079-9dee-ba961d7e77ce | serviceID: 6b07b6c2-009e-4799-be96-3bf972338b17]

Product-API:

Request to POST product stock with data {"products":[{"productId":1001,"quantity":1},{"productId":1002,"quantity":1},{"productId":1003,"quantity":1}]} | [transactionID: 8817508e-805c-48fb-9cb4-6a1e5a6e71e9 | serviceID: ea146e74-55cf-4a53-860e-9010d6e3f61b]

Response to POST product stock with data {"status":200,"message":"The stock is ok!"} | [transactionID: 8817508e-805c-48fb-9cb4-6a1e5a6e71e9 | serviceID: ea146e74-55cf-4a53-860e-9010d6e3f61b]

Sales-API:

Request to POST new order with data {"products":[{"productId":1001,"quantity":1},{"productId":1002,"quantity":1},{"productId":1003,"quantity":1}]} | [transactionID: 8817508e-805c-48fb-9cb4-6a1e5a6e71e9 | serviceID: 5f553f02-e830-4bed-bc04-8f71fe16cf28]

Response to POST login with data {"status":200,"createdOrder":{"products":[{"productId":1001,"quantity":1},{"productId":1002,"quantity":1},{"productId":1003,"quantity":1}],"user":{"id":1,"name":"User Test 1","email":"testeuser1@gmail.com"},"status":"PENDING","createdAt":"2021-10-12T16:34:49.778Z","updatedAt":"2021-10-12T16:34:49.778Z","transactionid":"8817508e-805c-48fb-9cb4-6a1e5a6e71e9","serviceid":"5f553f02-e830-4bed-bc04-8f71fe16cf28","\_id":"6165b92addaf7fc9dd85dad0","\_\_v":0}} | [transactionID: 8817508e-805c-48fb-9cb4-6a1e5a6e71e9 | serviceID: 5f553f02-e830-4bed-bc04-8f71fe16cf28]

RabbitMQ:

Sending message to product update stock: {"salesId":"6165b92addaf7fc9dd85dad0","products":[{"productId":1001,"quantity":1},{"productId":1002,"quantity":1},{"productId":1003,"quantity":1}],"transactionid":"8817508e-805c-48fb-9cb4-6a1e5a6e71e9"}

Recieving message with data: {"salesId":"6165b92addaf7fc9dd85dad0","products":[{"productId":1001,"quantity":1},{"productId":1002,"quantity":1},{"productId":1003,"quantity":1}],"transactionid":"8817508e-805c-48fb-9cb4-6a1e5a6e71e9"} and TransactionID: 8817508e-805c-48fb-9cb4-6a1e5a6e71e9

Sending message: {"salesId":"6165b92addaf7fc9dd85dad0","status":"APPROVED","transactionid":"8817508e-805c-48fb-9cb4-6a1e5a6e71e9"}

Recieving message from queue: {"salesId":"6165b92addaf7fc9dd85dad0","status":"APPROVED","transactionid":"8817508e-805c-48fb-9cb4-6a1e5a6e71e9"}

## Documentação dos endpoints

A documentação da API se faz presente no arquivo [API\_DOCS.md](https://github.com/vhnegrisoli/curso-udemy-comunicacao-microsservicos/blob/master/API_DOCS.md).

## Deploy no Heroku

As 3 APIs foram publicadas no Heroku, o repositório que foram publicados são esses:

* Auth-API - <https://github.com/vhnegrisoli2018/auth-api> (PostgreSQL e Coralogix Logging)
* Product-API - <https://github.com/vhnegrisoli2018/product-api> (Coralogix Logging, Cloud MongoDB e CloudAQMP)
* Sales-API - <https://github.com/vhnegrisoli2018/sales-api> (Coralogix Logging Heroku Postgres e CloudAQMP)

As URL base são:

* Auth-API - <https://microsservicos-auth-api.herokuapp.com/>
* Product-API - <https://microsservicos-product-api.herokuapp.com/>
* Sales-API - <https://microsservicos-sales-api.herokuapp.com/>

## Tracing com Coralogix Logging e Kibana

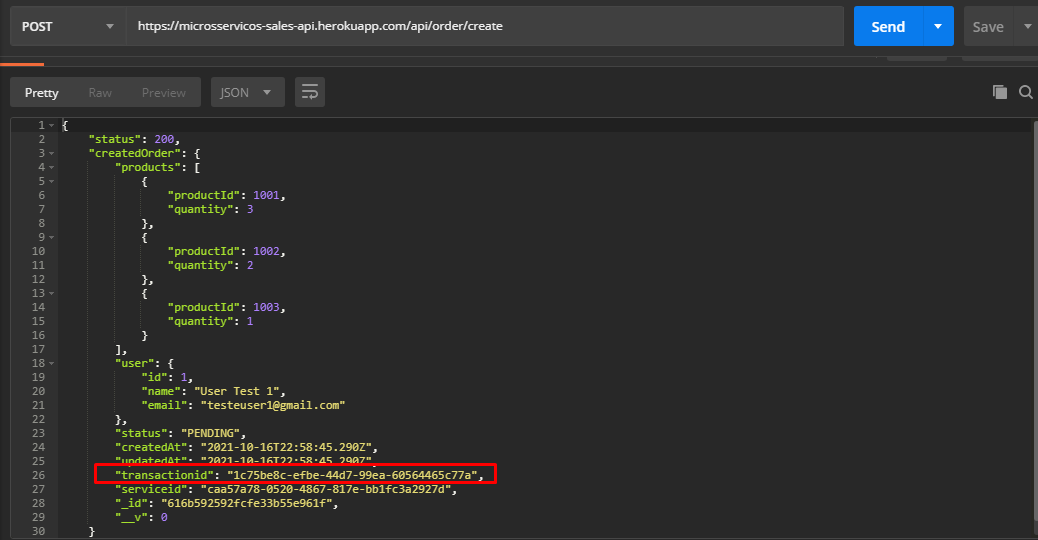
O Coralogix Logging é um add-on do Heroku para adicionarmos um dashboard de status e visualização de logs das aplicações.

Exemplo do dashboard do Coralogix Logging da aplicação Product-API:

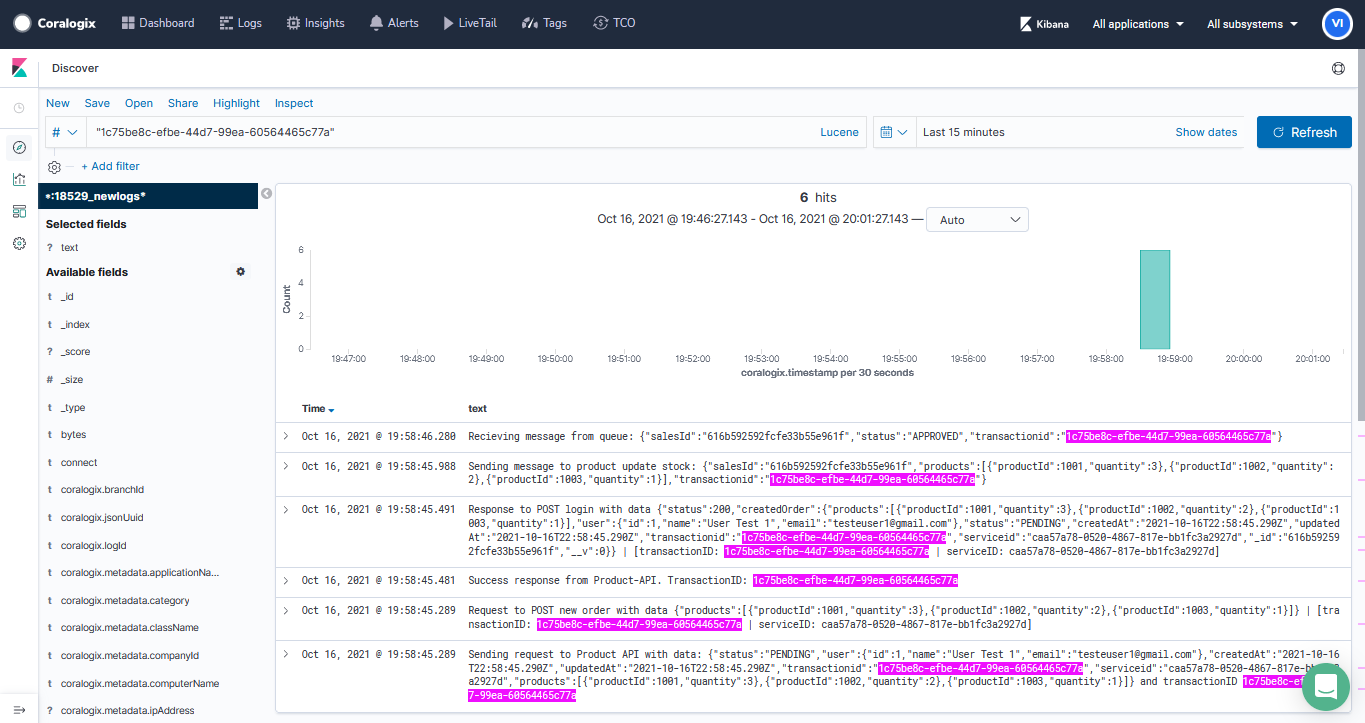


No Heroku, conseguimos realizar o tracing da aplicação através do nosso header **TransactionID** que é obrigatório em todos os endpoints.

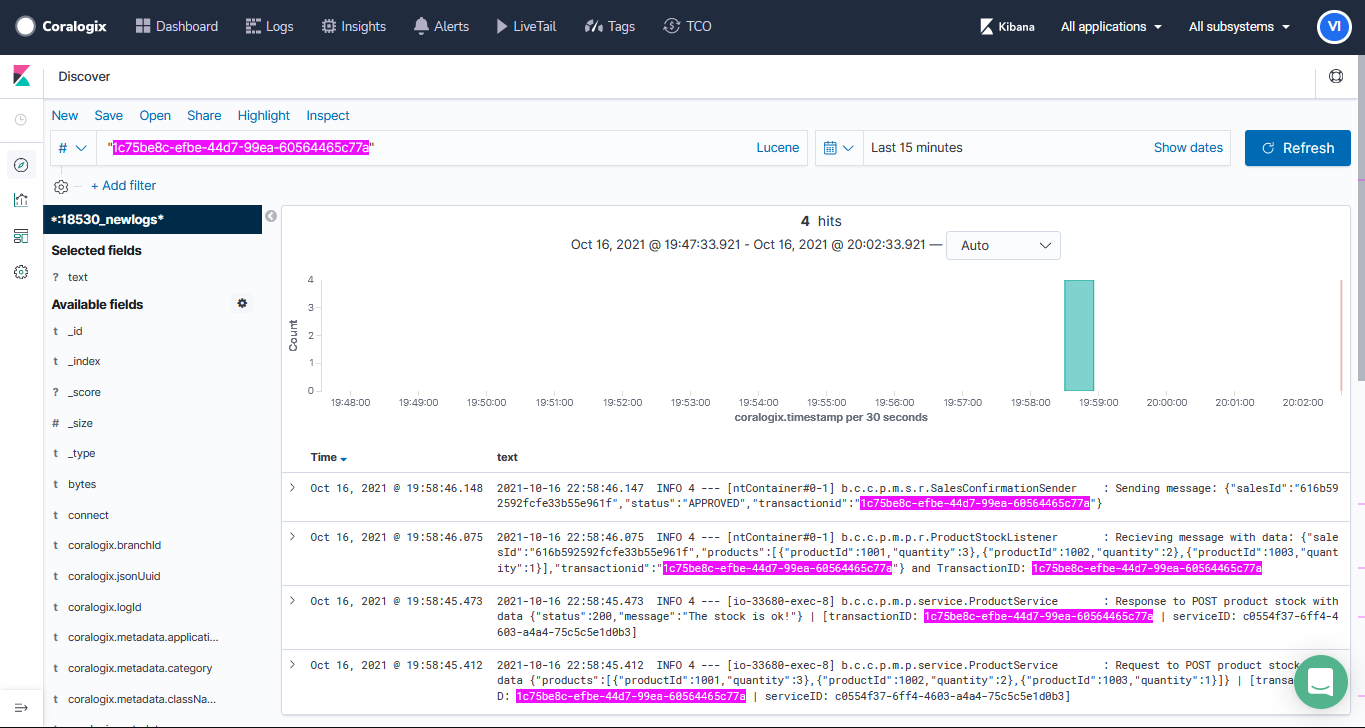
Abaixo foi mostrado um exemplo de tracing realizado com um pedido criado para o **TransactionID** com valor **1c75be8c-efbe-44d7-99ea-60564465c77a**.



Após realizada a requisição, vamos ao nosso Kibana disponibilizado pelo Coralogix Logging da aplicaçãode Sales-API e pesquisaremos os logs pelo valor **1c75be8c-efbe-44d7-99ea-60564465c77a**:



Podemos ver vários logs de entrada e saída, contendo o JSON de entrada e saída. Também podemos visualizar que foi feita uma chamada ao microsserviço de Product-API via HTTP REST, e também uma comunicação via mensagem do Rabbit, e conseguimos visualizar esses logs sendo recebidos lá na aplicação de Product-API:



Com isso, conseguimos rastrear todos os dados de entrada e saída dos endpoints, o ID da transação que circula entre eles via chamada REST e via mensageria, facilitando no acompanhamento de logs de uma requisição específica e, principalmente, no processo de troubleshooting.

## Comandos Docker

Abaixo serão listados alguns dos comandos executados durante o curso para criação dos containers dos bancos de dados PostgreSQL, MongoDB e do message broker RabbitMQ:

#### Container Auth-DB

docker run --name auth-db -p 5432:5432 -e POSTGRES\_DB=auth-db -e POSTGRES\_USER=admin -e POSTGRES\_PASSWORD=123456 postgres:11

#### Container Product-DB

docker run --name product-db -p 5433:5432 -e POSTGRES\_DB=product-db -e POSTGRES\_USER=admin -e POSTGRES\_PASSWORD=123456 postgres:11

#### Container Sales-DB

docker run --name sales-db -p 27017:27017 -p 28017:28017 -e MONGODB\_USER="admin" -e MONGODB\_DATABASE="sales" -e MONGODB\_PASS="123456" -v c:/db tutum/mongodb

#### Conexão no Mongoshell

mongo "mongodb://admin:123456@localhost:27017/sales"

#### Container RabbitMQ

docker run --name sales\_rabbit -p 5672:5672 -p 25676:25676 -p 15672:15672 rabbitmq:3-management

### Execução docker-compose

docker-compose up --build

Para ignorar os logs, adicione a flag -d.

## Autor

### Victor Hugo Negrisoli

### Desenvolvedor de Software Back-End

Fonte: <https://github.com/vhnegrisoli/curso-udemy-comunicacao-microsservicos>