Estrutura do projeto

```
simple-ecommerce/
├── docker-compose.yml
├── producer/                  # Envia pedidos
│   └── src/main/java/com/lab/producer
│       └── PedidoProducer.java
├── consumer/                  # Processa pedidos
│   └── src/main/java/com/lab/consumer
│       └── PedidoConsumer.java
└── README.md
```

```
mkdir -p producer/src/main/com/lab/producer/
mkdir -p consumer/src/main/com/lab/consumer/
touch producer/src/main/com/lab/producer/PedidoProducer.java
touch consumer/src/main/com/lab/consumer/PedidoConsumer.java
```

Agora vamos criar o projeto producer. Execute:

```
cd producer
```

Crie o pom.xml:

O arquivo `pom.xml` é o arquivo de configuração principal do Maven. Ele define a estrutura do projeto, suas dependências, e outras informações importantes, como o identificador do grupo, o nome do artefato e a versão.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.lab</groupId>
  <artifactId>producer</artifactId>
  <!-- The <dependencies> section is used to declare external libraries
required by the project. -->
  <dependencies>
    <!-- JMS API -->
    <!-- JMS API: Provides the Java Message Service (JMS) API for sending
and receiving messages between distributed systems -->
    <!-- Note: Ensure to periodically update the version of dependencies to
include the latest features and security patches -->
      <groupId>jakarta.jms</groupId>
```

```
      <artifactId>jakarta.jms-api</artifactId>
      <version>3.1.0</version>
    </dependency>

    <!-- ActiveMQ Artemis JMS Client -->
      <groupId>org.apache.activemq</groupId>
      <artifactId>artemis-jms-client</artifactId>
      <version>2.41.0</version> <!-- Note: Ensure this version is kept up
to date with the latest stable release. Check the latest version at
https://activemq.apache.org/components/artemis/download -->
</project>
```

Crie um arquivo docker-compose.yml com este conteúdo:

```
services:
  artemis:
    image: apache/activemq-artemis:latest
    container_name: artemis
    ports:
      - "61616:61616"   # Porta JMS (TCP)
      - "8161:8161"     # Console web
    environment:
      ARTEMIS_USERNAME: admin
      ARTEMIS_PASSWORD: admin
    stdin_open: true
    tty: true
```

```
docker-compose up -d
```

Crie o arquivo PedidoProducer.java com este conteúdo:

```
package com.lab.producer;

import jakarta.jms.*;
import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class PedidoProducer {

    public static void main(String[] args) {
        // Endereço do broker JMS
        String brokerUrl = "tcp://localhost:61616";
        // Nome da fila onde os pedidos serão enviados
        String queueName = "lab.pedidos";

        // Conexão com o broker usando try-with-resources para fechar
corretamente
```

```
        try (ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(brokerUrl);
             JMSContext context = connectionFactory.createContext("admin",
"admin", JMSContext.AUTO_ACKNOWLEDGE)) {

            // Cria (ou obtém) a fila
            Queue filaDePedidos = context.createQueue(queueName);

            // Cria o produtor de mensagens
            JMSProducer producer = context.createProducer();

            // Simula um pedido simples (pode ser um JSON ou string
qualquer)
            String mensagemPedido = """
                {
                  "id": "PED123",
                  "cliente": "João Silva",
                  "produto": "Notebook",
                  "valor": 3500.00
                }
                """;

            // Envia a mensagem para a fila
            producer.send(filaDePedidos, mensagemPedido);

            System.out.println("Pedido enviado para a fila: " +
mensagemPedido);

        } catch (JMSException e) {
            System.err.println("Erro ao enviar pedido: " + e.getMessage());
        }
    }
}
```

📘 Explicação do Código

- **ActiveMQConnectionFactory**: Cria uma conexão com o broker ActiveMQ Artemis.
- **JMSContext**: Cria um contexto JMS para enviar mensagens.
- **Queue**: Define a fila onde as mensagens serão enviadas.
- **JMSProducer**: Cria um produtor de mensagens para enviar mensagens para a fila.
- **mensagemPedido**: Simula um pedido em formato JSON.
- **producer.send()**: Envia a mensagem para a fila.
- **try-with-resources**: Garante que os recursos sejam fechados automaticamente após o uso, evitando vazamentos de memória.
- **JMSException**: Captura exceções relacionadas ao JMS, como falhas de conexão ou envio de mensagens.
- **System.out.println()**: Exibe uma mensagem de confirmação no console após o envio do pedido.
- **JMSContext.AUTO_ACKNOWLEDGE**: Configuração para que o JMS reconheça automaticamente as mensagens após o envio.
- **"admin", "admin"**: Credenciais de autenticação para o broker. Você pode alterar conforme necessário.

- **"tcp://localhost:61616"**: URL do broker ActiveMQ Artemis. Certifique-se de que o broker esteja em execução nesse endereço e porta.
- **"lab.pedidos"**: Nome da fila onde os pedidos serão enviados. Você pode alterar conforme necessário.
- **mensagemPedido**: O pedido é representado como uma string JSON. Você pode modificar o conteúdo conforme necessário.
- **"PED123", "João Silva", "Notebook", 3500.00**: Exemplos de dados do pedido. Você pode personalizar esses valores conforme necessário.
- **"Erro ao enviar pedido: " + e.getMessage()**: Mensagem de erro exibida no console caso ocorra uma exceção durante o envio do pedido.
- **"Pedido enviado para a fila: " + mensagemPedido**: Mensagem de confirmação exibida no console após o envio do pedido.
- **"lab.pedidos"**: Nome da fila onde os pedidos serão enviados. Você pode alterar conforme necessário.

Linha	O que faz
`ActiveMQConnectionFactory`	Cria uma fábrica de conexões com o ActiveMQ Artemis.
`JMSContext`	Cria a sessão para produzir/consumir mensagens.
`createQueue`	Cria uma referência para a fila `lab.pedidos`.
`createProducer`	Cria o produtor de mensagens.
`send(...)`	Envia a string JSON como mensagem para a fila.
`try-with-resources`	Garante que a conexão seja fechada corretamente.

▶ Como compilar e rodar

No diretório producer, compile:

```
mvn clean package
```

Execute o programa:

```
java -cp target/producer-1.0-SNAPSHOT.jar com.lab.producer.PedidoProducer
```

O exec-maven-plugin permite que você execute sua aplicação Java diretamente com o Maven, sem precisar montar um JAR nem escrever comandos complicados. 🛠 Etapas para configurar o exec-maven-plugin

1. **Adicione o plugin ao seu `pom.xml`**:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>3.1.0</version>
```

```
      <configuration>
        <mainClass>com.lab.producer.PedidoProducer</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
mvn clean compile exec:java
```

Isso enviará um pedido para a fila `lab.pedidos` no ActiveMQ Artemis. Você pode verificar o envio do pedido acessando o console do ActiveMQ Artemis ou usando um consumidor JMS para ler as mensagens da fila.

Vamos empacotar o producer em uma Imagem Docker

Dockerfile para que o próprio Docker compile seu projeto Java com Maven e gere o .jar. Isso elimina a necessidade de ter Maven ou Java instalados localmente.

🐳 Dockerfile com Maven + Java (multi-stage build) Crie este Dockerfile dentro da pasta producer:

```
# Etapa 1: Build com Maven
FROM maven:3.9.6-eclipse-temurin-17 as builder
WORKDIR /app

# Copia tudo para o container
COPY . .

# Executa o build e empacota o JAR
RUN mvn clean package -DskipTests

# Etapa 2: Runtime com Java (sem Maven)
FROM eclipse-temurin:17-jdk
WORKDIR /app

# Copia o JAR gerado da etapa anterior
COPY --from=builder /app/target/producer-1.0-SNAPSHOT.jar app.jar

# Executa o Producer
ENTRYPOINT ["java", "-jar", "app.jar"]
```

No terminal, vá até a pasta producer.

Execute o comando para construir a imagem:

```
docker build -t producer .
```

Executando o container

```
docker run --rm -it --network host producer
```