Exercício 1: Implementação de um Microsserviço com Spring Boot

Descrição:

Implemente um microsserviço em Spring Boot que gerencie uma lista de produtos, permitindo operações CRUD através de uma API RESTful.

Passo a Passo:

1. Inicialização do Projeto

Commit 1: `chore: Inicializar projeto Spring Boot com dependências básicas`

- **Ação:**
  - Utilize o Spring Initializr para gerar o projeto com as seguintes dependências:
    - Spring Web
    - Spring Data JPA
    - H2 Database
    - Lombok (opcional)
- **Automação:**
  - Execute o comando abaixo para gerar o projeto via linha de comando (usando `curl`):

```
curl https://start.spring.io/starter.zip \
  -d dependencies=web,data-jpa,h2,lombok \
  -d name=produtos-microsservico \
  -d packageName=com.exemplo.produtos \
  -o produtos-microsservico.zip
unzip produtos-microsservico.zip -d produtos-microsservico
cd produtos-microsservico
git init
git add .
git commit -m "chore: Inicializar projeto Spring Boot com
dependências básicas"
```

2. Configuração do Banco de Dados

Commit 2: `chore: Configurar banco de dados H2 e JPA`

- **Ação:**
  - Atualize `src/main/resources/application.properties`:

```
spring.datasource.url=jdbc:h2:mem:produtosdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.hibernate.ddl-auto=update
```

- **Automação:**
  - Crie um script de configuração:

```
cat <<EOL > src/main/resources/application.properties
spring.datasource.url=jdbc:h2:mem:produtosdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.hibernate.ddl-auto=update
EOL
git add src/main/resources/application.properties
git commit -m "chore: Configurar banco de dados H2 e JPA"
```

3. Criação da Entidade Produto

Commit 3: `feat: Criar entidade Produto`

- **Ação:**
  - Implemente a classe `Produto` em `src/main/java/com/exemplo/produtos/model/Produto.java`:

```
package com.exemplo.produtos.model;

import javax.persistence.*;
import lombok.Data;

@Entity
@Table(name = "produtos")
@Data
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private String categoria;

    @Column(nullable = false)
    private Double preco;

    @Column(nullable = false)
    private Integer quantidade;
}
```

- **Automação:**
  - Use um script para criar a classe:

```
mkdir -p src/main/java/com/exemplo/produtos/model
cat <<EOL >
src/main/java/com/exemplo/produtos/model/Produto.java
package com.exemplo.produtos.model;

import javax.persistence.*;
import lombok.Data;

@Entity
@Table(name = "produtos")
@Data
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private String categoria;

    @Column(nullable = false)
    private Double preco;

    @Column(nullable = false)
    private Integer quantidade;
}
EOL
git add src/main/java/com/exemplo/produtos/model/Produto.java
git commit -m "feat: Criar entidade Produto"
```

4. Criação do Repositório ProdutoRepository

Commit 4: `feat: Criar repositório ProdutoRepository`

- **Ação:**
  - Implemente `ProdutoRepository` em `src/main/java/com/exemplo/produtos/repository/ProdutoRepository.java`:

```
package com.exemplo.produtos.repository;

import com.exemplo.produtos.model.Produto;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProdutoRepository extends
JpaRepository<Produto, Long> {
}
```

- **Automação:**
  - Use um script:

```
mkdir -p src/main/java/com/exemplo/produtos/repository
cat <<EOL >
src/main/java/com/exemplo/produtos/repository/ProdutoRepositor
y.java
package com.exemplo.produtos.repository;

import com.exemplo.produtos.model.Produto;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProdutoRepository extends
JpaRepository<Produto, Long> {
}
EOL
git add
src/main/java/com/exemplo/produtos/repository/ProdutoRepositor
y.java
git commit -m "feat: Criar repositório ProdutoRepository"
```

5. Implementação do Serviço ProdutoService

Commit 5: `feat: Implementar serviço ProdutoService`

- **Ação:**
  - Crie `ProdutoService` em `src/main/java/com/exemplo/produtos/service/ProdutoService.java`:

```
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.repository.ProdutoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class ProdutoService {

    @Autowired
    private ProdutoRepository produtoRepository;

    public Produto criarProduto(Produto produto) {
        return produtoRepository.save(produto);
    }

    public List<Produto> listarProdutos() {
        return produtoRepository.findAll();
    }

    public Optional<Produto> buscarProdutoPorId(Long id) {
        return produtoRepository.findById(id);
    }

    public Produto atualizarProduto(Long id, Produto
produtoAtualizado) {
        Produto produto = produtoRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Produto
não encontrado"));
        produto.setNome(produtoAtualizado.getNome());

produto.setCategoria(produtoAtualizado.getCategoria());
        produto.setPreco(produtoAtualizado.getPreco());

produto.setQuantidade(produtoAtualizado.getQuantidade());
        return produtoRepository.save(produto);
    }

    public void deletarProduto(Long id) {
        produtoRepository.deleteById(id);
    }
}
```

- **Automação:**
    - Use um script:

```
mkdir -p src/main/java/com/exemplo/produtos/service
cat <<EOL >
src/main/java/com/exemplo/produtos/service/ProdutoService.java
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.repository.ProdutoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class ProdutoService {

    @Autowired
    private ProdutoRepository produtoRepository;

    public Produto criarProduto(Produto produto) {
        return produtoRepository.save(produto);
    }

    public List<Produto> listarProdutos() {
        return produtoRepository.findAll();
    }

    public Optional<Produto> buscarProdutoPorId(Long id) {
        return produtoRepository.findById(id);
    }

    public Produto atualizarProduto(Long id, Produto
produtoAtualizado) {
        Produto produto = produtoRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Produto
não encontrado"));
        produto.setNome(produtoAtualizado.getNome());

produto.setCategoria(produtoAtualizado.getCategoria());
        produto.setPreco(produtoAtualizado.getPreco());

produto.setQuantidade(produtoAtualizado.getQuantidade());
        return produtoRepository.save(produto);
    }

    public void deletarProduto(Long id) {
        produtoRepository.deleteById(id);
```

```
    }
}
EOL
git add
src/main/java/com/exemplo/produtos/service/ProdutoService.java
git commit -m "feat: Implementar serviço ProdutoService"
```

6. Criação do Controlador ProdutoController

Commit 6: `feat: Criar controlador REST ProdutoController`

- **Ação:**
  - Implemente `ProdutoController` em `src/main/java/com/exemplo/produtos/controller/ProdutoController.java`:

```java
package com.exemplo.produtos.controller;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.service.ProdutoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/produtos")
public class ProdutoController {

    @Autowired
    private ProdutoService produtoService;

    @PostMapping
    public ResponseEntity<Produto> criarProduto(@RequestBody
Produto produto) {
        Produto criado = produtoService.criarProduto(produto);
        return new ResponseEntity<>(criado,
HttpStatus.CREATED);
    }

    @GetMapping
    public ResponseEntity<List<Produto>> listarProdutos() {
        List<Produto> produtos =
produtoService.listarProdutos();
        return new ResponseEntity<>(produtos, HttpStatus.OK);
    }

    @GetMapping("/{id}")
```

```
    public ResponseEntity<Produto>
buscarProdutoPorId(@PathVariable Long id) {
        Produto produto =
produtoService.buscarProdutoPorId(id)
            .orElseThrow(() -> new RuntimeException("Produto
não encontrado"));
        return new ResponseEntity<>(produto, HttpStatus.OK);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Produto>
atualizarProduto(@PathVariable Long id, @RequestBody Produto
produto) {
        Produto atualizado =
produtoService.atualizarProduto(id, produto);
        return new ResponseEntity<>(atualizado,
HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deletarProduto(@PathVariable
Long id) {
        produtoService.deletarProduto(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
```

- **Automação:**
  - Use um script:

```
mkdir -p src/main/java/com/exemplo/produtos/controller
cat <<EOL >
src/main/java/com/exemplo/produtos/controller/ProdutoControlle
r.java
package com.exemplo.produtos.controller;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.service.ProdutoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/produtos")
public class ProdutoController {

    @Autowired
```

```
    private ProdutoService produtoService;

    @PostMapping
    public ResponseEntity<Produto> criarProduto(@RequestBody
Produto produto) {
        Produto criado = produtoService.criarProduto(produto);
        return new ResponseEntity<>(criado,
HttpStatus.CREATED);
    }

    @GetMapping
    public ResponseEntity<List<Produto>> listarProdutos() {
        List<Produto> produtos =
produtoService.listarProdutos();
        return new ResponseEntity<>(produtos, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Produto>
buscarProdutoPorId(@PathVariable Long id) {
        Produto produto =
produtoService.buscarProdutoPorId(id)
            .orElseThrow(() -> new RuntimeException("Produto
não encontrado"));
        return new ResponseEntity<>(produto, HttpStatus.OK);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Produto>
atualizarProduto(@PathVariable Long id, @RequestBody Produto
produto) {
        Produto atualizado =
produtoService.atualizarProduto(id, produto);
        return new ResponseEntity<>(atualizado,
HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deletarProduto(@PathVariable
Long id) {
        produtoService.deletarProduto(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
EOL
git add
src/main/java/com/exemplo/produtos/controller/ProdutoControlle
r.java
git commit -m "feat: Criar controlador REST ProdutoController"
```

7. Tratamento Global de Exceções

Commit 7: `feat: Implementar tratamento global de exceções`

- **Ação:**
  - Crie `GlobalExceptionHandler` em `src/main/java/com/exemplo/produtos/exception/GlobalExceptionHandler.java`:

```
package com.exemplo.produtos.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.ControllerAdvice;
import
org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<?>
handleRuntimeException(RuntimeException ex, WebRequest
request) {
        Map<String, String> errorDetails = new HashMap<>();
        errorDetails.put("message", ex.getMessage());
        return new ResponseEntity<>(errorDetails,
HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<?> handleGlobalException(Exception
ex, WebRequest request) {
        Map<String, String> errorDetails = new HashMap<>();
        errorDetails.put("message", ex.getMessage());
        return new ResponseEntity<>(errorDetails,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

- **Automação:**
  - Use um script:

```
```

```
mkdir -p src/main/java/com/exemplo/produtos/exception
cat <<EOL >
src/main/java/com/exemplo/produtos/exception/GlobalExceptionHa
ndler.java
package com.exemplo.produtos.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.ControllerAdvice;
import
org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<?>
handleRuntimeException(RuntimeException ex, WebRequest
request) {
        Map<String, String> errorDetails = new HashMap<>();
        errorDetails.put("message", ex.getMessage());
        return new ResponseEntity<>(errorDetails,
HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<?> handleGlobalException(Exception
ex, WebRequest request) {
        Map<String, String> errorDetails = new HashMap<>();
        errorDetails.put("message", ex.getMessage());
        return new ResponseEntity<>(errorDetails,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
EOL
git add
src/main/java/com/exemplo/produtos/exception/GlobalExceptionHa
ndler.java
git commit -m "feat: Implementar tratamento global de
exceções"
```

8. Documentação e Finalização do Exercício 1

Commit 8: `docs: Adicionar instruções para execução e testes manuais`

- **Ação:**
  - Atualize `README.md` com instruções de execução e testes:

````
# Produtos Microsserviço

## Descrição
Microsserviço em Spring Boot para gerenciamento de produtos
com operações CRUD via API RESTful.

## Tecnologias Utilizadas
- Spring Boot
- Spring Data JPA
- H2 Database
- Lombok

## Como Rodar

### Requisitos
- Java 11+
- Maven

### Passos
1. Clone o repositório:
    ```bash
    git clone <URL_DO_REPOSITORIO>
    cd produtos-microsservico
    ```
2. Execute a aplicação:
    ```bash
    mvn spring-boot:run
    ```
3. Acesse o console do H2 em [http://localhost:8080/h2-
console](http://localhost:8080/h2-console)
    - JDBC URL: `jdbc:h2:mem:produtosdb`
    - User: `sa`
    - Senha: (vazia)

## Endpoints

- **Criar Produto:** `POST /api/produtos`
- **Listar Produtos:** `GET /api/produtos`
- **Buscar Produto por ID:** `GET /api/produtos/{id}`
- **Atualizar Produto:** `PUT /api/produtos/{id}`
- **Deletar Produto:** `DELETE /api/produtos/{id}`
````

- **Automação:**
  - Use um script:

```
cat <<EOL > README.md
# Produtos Microsserviço

## Descrição
Microsserviço em Spring Boot para gerenciamento de produtos
com operações CRUD via API RESTful.

## Tecnologias Utilizadas
- Spring Boot
- Spring Data JPA
- H2 Database
- Lombok

## Como Rodar

### Requisitos
- Java 11+
- Maven

### Passos
1. Clone o repositório:
    \`\`\`bash
    git clone <URL_DO_REPOSITORIO>
    cd produtos-microsservico
    \`\`\`
2. Execute a aplicação:
    \`\`\`bash
    mvn spring-boot:run
    \`\`\`
3. Acesse o console do H2 em [http://localhost:8080/h2-
console](http://localhost:8080/h2-console)
    - JDBC URL: \`jdbc:h2:mem:produtosdb\`
    - User: \`sa\`
    - Senha: (vazia)

## Endpoints

- **Criar Produto:** \`POST /api/produtos\`
- **Listar Produtos:** \`GET /api/produtos\`
- **Buscar Produto por ID:** \`GET /api/produtos/{id}\`
- **Atualizar Produto:** \`PUT /api/produtos/{id}\`
- **Deletar Produto:** \`DELETE /api/produtos/{id}\`
EOL
git add README.md
git commit -m "docs: Adicionar instruções para execução e
testes manuais"
```

9. Finalização do Exercício 1

Commit 9: `chore: Finalizar Exercício 1 - Microsserviço de Produtos`

- **Ação:**
    - Revisão final do código e documentação.
- **Automação:**
    - Use um script:

```
git commit --allow-empty -m "chore: Finalizar Exercício 1 - Microsserviço de Produtos"
```

Exercício 2: Implementação de Testes Unitários e Integração Contínua

Descrição:

- Adicione testes unitários para a aplicação criada no Exercício 1 utilizando JUnit e Mockito.
- Configure um pipeline de integração contínua (CI) utilizando GitHub Actions para automatizar a execução dos testes.
- O pipeline deve executar os testes a cada push e fornecer um relatório dos resultados.

Passo a Passo:

1. Adição de Dependências para Testes

Commit 1: `chore: Adicionar dependências para testes unitários com JUnit e Mockito`

- **Ação:**
    - Atualize `pom.xml` para incluir dependências de teste:

```
<dependencies>
    <!-- Dependências existentes -->

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- **Automação:**
    - Use um script:

```
cat <<EOL >> pom.xml

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <scope>test</scope>
    </dependency>
EOL
git add pom.xml
git commit -m "chore: Adicionar dependências para testes
unitários com JUnit e Mockito"
```

2. Implementação de Testes Unitários para ProdutoService

Commit 2: `test: Adicionar testes unitários para ProdutoService`

- **Ação:**
    - Crie `ProdutoServiceTest` em `src/test/java/com/exemplo/produtos/service/ProdutoServiceTest.java`:

```
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.repository.ProdutoRepository;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
```

```
public class ProdutoServiceTest {

    @Mock
    private ProdutoRepository produtoRepository;

    @InjectMocks
    private ProdutoService produtoService;

    @Test
    public void testCriarProduto() {
        Produto produto = new Produto(null, "Produto A",
"Categoria A", 100.0, 10);
        Produto salvo = new Produto(1L, "Produto A",
"Categoria A", 100.0, 10);


when(produtoRepository.save(produto)).thenReturn(salvo);

        Produto resultado =
produtoService.criarProduto(produto);

        assertNotNull(resultado.getId());
        assertEquals("Produto A", resultado.getNome());
        verify(produtoRepository, times(1)).save(produto);
    }

    @Test
    public void testListarProdutos() {
        Produto p1 = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);
        Produto p2 = new Produto(2L, "Produto B", "Categoria
B", 200.0, 20);
        List<Produto> produtos = Arrays.asList(p1, p2);


when(produtoRepository.findAll()).thenReturn(produtos);

        List<Produto> resultado =
produtoService.listarProdutos();

        assertEquals(2, resultado.size());
        verify(produtoRepository, times(1)).findAll();
    }

    @Test
    public void testBuscarProdutoPorId() {
        Produto p = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);

when(produtoRepository.findById(1L)).thenReturn(Optional.of(p)
);

        Optional<Produto> resultado =
```

```
produtoService.buscarProdutoPorId(1L);

        assertTrue(resultado.isPresent());
        assertEquals("Produto A", resultado.get().getNome());
        verify(produtoRepository, times(1)).findById(1L);
    }

    @Test
    public void testAtualizarProduto() {
        Produto existente = new Produto(1L, "Produto A",
"Categoria A", 100.0, 10);
        Produto atualizado = new Produto(1L, "Produto A+",
"Categoria A+", 150.0, 15);


when(produtoRepository.findById(1L)).thenReturn(Optional.of(ex
istente));

when(produtoRepository.save(existente)).thenReturn(atualizado)
;

        Produto resultado =
produtoService.atualizarProduto(1L, atualizado);

        assertEquals("Produto A+", resultado.getNome());
        assertEquals("Categoria A+",
resultado.getCategoria());
        assertEquals(150.0, resultado.getPreco());
        assertEquals(15, resultado.getQuantidade());
        verify(produtoRepository, times(1)).findById(1L);
        verify(produtoRepository, times(1)).save(existente);
    }

    @Test
    public void testDeletarProduto() {
        doNothing().when(produtoRepository).deleteById(1L);
        produtoService.deletarProduto(1L);
        verify(produtoRepository, times(1)).deleteById(1L);
    }
}
```

- **Automação:**
  - Use um script:

```
mkdir -p src/test/java/com/exemplo/produtos/service
cat <<EOL >
src/test/java/com/exemplo/produtos/service/ProdutoServiceTest.
java
package com.exemplo.produtos.service;
```

```
import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.repository.ProdutoRepository;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
public class ProdutoServiceTest {

    @Mock
    private ProdutoRepository produtoRepository;

    @InjectMocks
    private ProdutoService produtoService;

    @Test
    public void testCriarProduto() {
        Produto produto = new Produto(null, "Produto A",
"Categoria A", 100.0, 10);
        Produto salvo = new Produto(1L, "Produto A",
"Categoria A", 100.0, 10);


when(produtoRepository.save(produto)).thenReturn(salvo);

        Produto resultado =
produtoService.criarProduto(produto);

        assertNotNull(resultado.getId());
        assertEquals("Produto A", resultado.getNome());
        verify(produtoRepository, times(1)).save(produto);
    }

    @Test
    public void testListarProdutos() {
        Produto p1 = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);
        Produto p2 = new Produto(2L, "Produto B", "Categoria
B", 200.0, 20);
        List<Produto> produtos = Arrays.asList(p1, p2);


when(produtoRepository.findAll()).thenReturn(produtos);
```

```
        List<Produto> resultado =
produtoService.listarProdutos();

        assertEquals(2, resultado.size());
        verify(produtoRepository, times(1)).findAll();
    }

    @Test
    public void testBuscarProdutoPorId() {
        Produto p = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);

when(produtoRepository.findById(1L)).thenReturn(Optional.of(p)
);

        Optional<Produto> resultado =
produtoService.buscarProdutoPorId(1L);

        assertTrue(resultado.isPresent());
        assertEquals("Produto A", resultado.get().getNome());
        verify(produtoRepository, times(1)).findById(1L);
    }

    @Test
    public void testAtualizarProduto() {
        Produto existente = new Produto(1L, "Produto A",
"Categoria A", 100.0, 10);
        Produto atualizado = new Produto(1L, "Produto A+",
"Categoria A+", 150.0, 15);


when(produtoRepository.findById(1L)).thenReturn(Optional.of(ex
istente));

when(produtoRepository.save(existente)).thenReturn(atualizado)
;

        Produto resultado =
produtoService.atualizarProduto(1L, atualizado);

        assertEquals("Produto A+", resultado.getNome());
        assertEquals("Categoria A+",
resultado.getCategoria());
        assertEquals(150.0, resultado.getPreco());
        assertEquals(15, resultado.getQuantidade());
        verify(produtoRepository, times(1)).findById(1L);
        verify(produtoRepository, times(1)).save(existente);
    }

    @Test
    public void testDeletarProduto() {
        doNothing().when(produtoRepository).deleteById(1L);
```

```
        produtoService.deletarProduto(1L);
        verify(produtoRepository, times(1)).deleteById(1L);
    }
}
EOL
git add
src/test/java/com/exemplo/produtos/service/ProdutoServiceTest.
java
git commit -m "test: Adicionar testes unitários para
ProdutoService"
```

3. Implementação de Testes Unitários para ProdutoController

Commit 3: `test: Adicionar testes unitários para ProdutoController`

- **Ação:**
  - Crie `ProdutoControllerTest` em `src/test/java/com/exemplo/produtos/controller/ProdutoControllerTest.java`:

```
package com.exemplo.produtos.controller;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.service.ProdutoService;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.WebMvc
Test;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import
org.springframework.test.context.junit.jupiter.SpringExtension
;
import org.springframework.test.web.servlet.MockMvc;

import java.util.Arrays;
import java.util.Optional;

import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyLong;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBui
lders.*;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatch
```

```
ers.*;

@ExtendWith(SpringExtension.class)
@WebMvcTest(ProdutoController.class)
public class ProdutoControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ProdutoService produtoService;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    public void testCriarProduto() throws Exception {
        Produto produto = new Produto(null, "Produto A",
"Categoria A", 100.0, 10);
        Produto salvo = new Produto(1L, "Produto A",
"Categoria A", 100.0, 10);


Mockito.when(produtoService.criarProduto(any(Produto.class))).
thenReturn(salvo);

        mockMvc.perform(post("/api/produtos")
            .contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(produto)))
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.id").value(1))
            .andExpect(jsonPath("$.nome").value("Produto A"));
    }

    @Test
    public void testListarProdutos() throws Exception {
        Produto p1 = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);
        Produto p2 = new Produto(2L, "Produto B", "Categoria
B", 200.0, 20);


Mockito.when(produtoService.listarProdutos()).thenReturn(Array
s.asList(p1, p2));

        mockMvc.perform(get("/api/produtos"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.length()").value(2))
            .andExpect(jsonPath("$[0].nome").value("Produto
A"))
            .andExpect(jsonPath("$[1].nome").value("Produto
B"));
```

```
    }

    @Test
    public void testBuscarProdutoPorId() throws Exception {
        Produto p = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);


Mockito.when(produtoService.buscarProdutoPorId(1L)).thenReturn
(Optional.of(p));

        mockMvc.perform(get("/api/produtos/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nome").value("Produto A"));
    }

    @Test
    public void testAtualizarProduto() throws Exception {
        Produto atualizado = new Produto(1L, "Produto A+",
"Categoria A+", 150.0, 15);


Mockito.when(produtoService.atualizarProduto(Mockito.eq(1L),
any(Produto.class))).thenReturn(atualizado);

        mockMvc.perform(put("/api/produtos/1")
            .contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(atualizado)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nome").value("Produto A+"))

.andExpect(jsonPath("$.categoria").value("Categoria A+"))
            .andExpect(jsonPath("$.preco").value(150.0))
            .andExpect(jsonPath("$.quantidade").value(15));
    }

    @Test
    public void testDeletarProduto() throws Exception {

Mockito.doNothing().when(produtoService).deletarProduto(1L);

        mockMvc.perform(delete("/api/produtos/1"))
            .andExpect(status().isNoContent());
    }
}
```

- **Automação:**
    - Use um script:

```

```

```
mkdir -p src/test/java/com/exemplo/produtos/controller
cat <<EOL >
src/test/java/com/exemplo/produtos/controller/ProdutoControlle
rTest.java
package com.exemplo.produtos.controller;

import com.exemplo.produtos.model.Produto;
import com.exemplo.produtos.service.ProdutoService;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.WebMvc
Test;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import
org.springframework.test.context.junit.jupiter.SpringExtension
;
import org.springframework.test.web.servlet.MockMvc;

import java.util.Arrays;
import java.util.Optional;

import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyLong;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBui
lders.*;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatch
ers.*;

@ExtendWith(SpringExtension.class)
@WebMvcTest(ProdutoController.class)
public class ProdutoControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ProdutoService produtoService;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    public void testCriarProduto() throws Exception {
        Produto produto = new Produto(null, "Produto A",
"Categoria A", 100.0, 10);
```

```
        Produto salvo = new Produto(1L, "Produto A",
"Categoria A", 100.0, 10);


Mockito.when(produtoService.criarProduto(any(Produto.class))).
thenReturn(salvo);

        mockMvc.perform(post("/api/produtos")
            .contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(produto)))
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.id").value(1))
            .andExpect(jsonPath("$.nome").value("Produto A"));
    }

    @Test
    public void testListarProdutos() throws Exception {
        Produto p1 = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);
        Produto p2 = new Produto(2L, "Produto B", "Categoria
B", 200.0, 20);


Mockito.when(produtoService.listarProdutos()).thenReturn(Array
s.asList(p1, p2));

        mockMvc.perform(get("/api/produtos"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.length()").value(2))
            .andExpect(jsonPath("$[0].nome").value("Produto
A"))
            .andExpect(jsonPath("$[1].nome").value("Produto
B"));
    }

    @Test
    public void testBuscarProdutoPorId() throws Exception {
        Produto p = new Produto(1L, "Produto A", "Categoria
A", 100.0, 10);


Mockito.when(produtoService.buscarProdutoPorId(1L)).thenReturn
(Optional.of(p));

        mockMvc.perform(get("/api/produtos/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nome").value("Produto A"));
    }

    @Test
    public void testAtualizarProduto() throws Exception {
        Produto atualizado = new Produto(1L, "Produto A+",
```

```
"Categoria A+", 150.0, 15);


Mockito.when(produtoService.atualizarProduto(Mockito.eq(1L),
any(Produto.class))).thenReturn(atualizado);

        mockMvc.perform(put("/api/produtos/1")
            .contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(atualizado)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nome").value("Produto A+"))

.andExpect(jsonPath("$.categoria").value("Categoria A+"))
            .andExpect(jsonPath("$.preco").value(150.0))
            .andExpect(jsonPath("$.quantidade").value(15));
    }

    @Test
    public void testDeletarProduto() throws Exception {

Mockito.doNothing().when(produtoService).deletarProduto(1L);

        mockMvc.perform(delete("/api/produtos/1"))
            .andExpect(status().isNoContent());
    }
}
EOL
git add
src/test/java/com/exemplo/produtos/controller/ProdutoControlle
rTest.java
git commit -m "test: Adicionar testes unitários para
ProdutoController"
```

4. Configuração do Pipeline de Integração Contínua com GitHub Actions

Commit 4: `ci: Configurar pipeline de CI com GitHub Actions`

- **Ação:**
    - Crie o diretório `.github/workflows` e adicione `ci.yml`:

```
name: Java CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

```
jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'

    - name: Build with Maven
      run: mvn clean install

    - name: Run tests
      run: mvn test
```

- **Automação:**
  - Use um script:

```
mkdir -p .github/workflows
cat <<EOL > .github/workflows/ci.yml
name: Java CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'

    - name: Build with Maven
```

```
        run: mvn clean install

      - name: Run tests
        run: mvn test
EOL
git add .github/workflows/ci.yml
git commit -m "ci: Configurar pipeline de CI com GitHub
Actions"
```

5. Testes do Pipeline

Commit 5: `test: Verificar execução dos testes via pipeline de CI`

- **Ação:**
  - Faça um push das alterações:

    ```
    git push origin main
    ```

  - Verifique no GitHub Actions se o pipeline foi executado corretamente e os testes foram aprovados.
- **Automação:**
  - Como a verificação é manual via GitHub, este passo não requer script.

6. Finalização do Exercício 2

Commit 6: `chore: Finalizar Exercício 2 - Testes e CI`

- **Ação:**
  - Revisão final dos testes e configuração de CI.
- **Automação:**
  - Use um script:

    ```
    git commit --allow-empty -m "chore: Finalizar Exercício 2 -
    Testes e CI"
    ```

Exercício 3: Escalabilidade e Mensageria com AWS e RabbitMQ

Descrição:

- Modifique o microsserviço criado no Exercício 1 para publicar mensagens em uma fila RabbitMQ sempre que um novo produto for criado.
- Implemente um consumidor que leia as mensagens da fila e faça o log dessas mensagens.
- Configure a aplicação para rodar na AWS, utilizando serviços como AWS ECS ou AWS Lambda.

Passo a Passo:

1. Adição de Dependências para RabbitMQ

Commit 1: `chore: Adicionar dependências para RabbitMQ`

- **Ação:**
  - Atualize `pom.xml` para incluir Spring AMQP:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

- **Automação:**
  - Use um script:

```
cat <<EOL >> pom.xml

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
EOL
git add pom.xml
git commit -m "chore: Adicionar dependências para RabbitMQ"
```

2. Configuração do RabbitMQ

Commit 2: `chore: Configurar RabbitMQ no application.properties`

- **Ação:**
  - Atualize `src/main/resources/application.properties`:

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

```
spring.rabbitmq.template.exchange=exchange-produtos
spring.rabbitmq.template.routing-key=routing-produtos
```

- **Automação:**
  - Use um script:

```
cat <<EOL >> src/main/resources/application.properties

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
spring.rabbitmq.template.exchange=exchange-produtos
spring.rabbitmq.template.routing-key=routing-produtos
EOL
git add src/main/resources/application.properties
git commit -m "chore: Configurar RabbitMQ no
application.properties"
```

3. Implementação do Publisher de Mensagens

Commit 3: `feat: Implementar publisher para enviar mensagens ao RabbitMQ`

- **Ação:**
  - Crie `MensagemPublisher` em `src/main/java/com/exemplo/produtos/service/MensagemPublisher.java`:

```
package com.exemplo.produtos.service;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MensagemPublisher {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    public void enviarMensagem(String mensagem) {
        rabbitTemplate.convertAndSend("exchange-produtos",
"routing-produtos", mensagem);
    }
}
```

- Atualize `ProdutoService` para usar `MensagemPublisher`:

```
// Adicione a injeção de dependência
@Autowired
private MensagemPublisher mensagemPublisher;

public Produto criarProduto(Produto produto) {
    Produto salvo = produtoRepository.save(produto);
    mensagemPublisher.enviarMensagem("Produto criado: " +
salvo.getNome());
    return salvo;
}
```

- **Automação:**
  - Use um script:

```
mkdir -p src/main/java/com/exemplo/produtos/service
cat <<EOL >
src/main/java/com/exemplo/produtos/service/MensagemPublisher.j
ava
package com.exemplo.produtos.service;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MensagemPublisher {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    public void enviarMensagem(String mensagem) {
        rabbitTemplate.convertAndSend("exchange-produtos",
"routing-produtos", mensagem);
    }
}
EOL
git add
src/main/java/com/exemplo/produtos/service/MensagemPublisher.j
ava

# Atualize ProdutoService.java
sed -i '/public class ProdutoService {/a \
\n    @Autowired\n    private MensagemPublisher
mensagemPublisher;\n\
'
src/main/java/com/exemplo/produtos/service/ProdutoService.java
```

```
sed -i '/return produtoRepository.save(produto);/a \
\n        mensagemPublisher.enviarMensagem("Produto criado: "
+ salvo.getNome());\n\
'
src/main/java/com/exemplo/produtos/service/ProdutoService.java

git add
src/main/java/com/exemplo/produtos/service/ProdutoService.java
git commit -m "feat: Implementar publisher para enviar
mensagens ao RabbitMQ"
```

4. Implementação do Consumidor de Mensagens

Commit 4: `feat: Implementar consumidor para ler mensagens do RabbitMQ`

- **Ação:**
  - Crie `MensagemConsumer` em `src/main/java/com/exemplo/produtos/service/MensagemConsumer.java`:

```java
package com.exemplo.produtos.service;

import
org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Service;

@Service
public class MensagemConsumer {

    @RabbitListener(queues = "fila-produtos")
    public void receberMensagem(String mensagem) {
        System.out.println("Mensagem recebida: " + mensagem);
    }
}
```

  - Crie configuração de filas em `RabbitMQConfig`:

```java
package com.exemplo.produtos.config;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
```

```
public class RabbitMQConfig {

    @Bean
    public DirectExchange exchange() {
        return new DirectExchange("exchange-produtos");
    }

    @Bean
    public Queue filaProdutos() {
        return new Queue("fila-produtos");
    }

    @Bean
    public Binding binding(Queue filaProdutos, DirectExchange
exchange) {
        return
BindingBuilder.bind(filaProdutos).to(exchange).with("routing-
produtos");
    }
}
```

- **Automação:**
  - Use um script:

```
mkdir -p src/main/java/com/exemplo/produtos/service
cat <<EOL >
src/main/java/com/exemplo/produtos/service/MensagemConsumer.ja
va
package com.exemplo.produtos.service;

import
org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Service;

@Service
public class MensagemConsumer {

    @RabbitListener(queues = "fila-produtos")
    public void receberMensagem(String mensagem) {
        System.out.println("Mensagem recebida: " + mensagem);
    }
}
EOL
git add
src/main/java/com/exemplo/produtos/service/MensagemConsumer.ja
va

# Crie RabbitMQConfig.java
mkdir -p src/main/java/com/exemplo/produtos/config
cat <<EOL >
```

```
src/main/java/com/exemplo/produtos/config/RabbitMQConfig.java
package com.exemplo.produtos.config;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitMQConfig {

    @Bean
    public DirectExchange exchange() {
        return new DirectExchange("exchange-produtos");
    }

    @Bean
    public Queue filaProdutos() {
        return new Queue("fila-produtos");
    }

    @Bean
    public Binding binding(Queue filaProdutos, DirectExchange
exchange) {
        return
BindingBuilder.bind(filaProdutos).to(exchange).with("routing-
produtos");
    }
}
EOL
git add
src/main/java/com/exemplo/produtos/config/RabbitMQConfig.java
git commit -m "feat: Implementar consumidor para ler mensagens
do RabbitMQ"
```

5. Criação do Dockerfile

Commit 5: `chore: Adicionar Dockerfile para containerização`

- **Ação:**
  - Crie `Dockerfile` na raiz do projeto:

```
FROM openjdk:11-jre-slim
VOLUME /tmp
COPY target/produtos-microsservico-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

- **Automação:**
  - Use um script:

```
cat <<EOL > Dockerfile
FROM openjdk:11-jre-slim
VOLUME /tmp
COPY target/produtos-microsservico-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
EOL
git add Dockerfile
git commit -m "chore: Adicionar Dockerfile para
containerização"
```

6. Deployment na AWS ECS

Commit 6: `chore: Configurar deployment na AWS ECS`

- **Ação:**
  - **Pré-requisitos:**
    - **AWS CLI** configurado.
    - **Docker** instalado.
    - **AWS ECR** criado para armazenar a imagem Docker.
  - **Passos Automatizados:**

```
# Substitua <account_id> e <region> pelos valores apropriados
AWS_ACCOUNT_ID=<account_id>
AWS_REGION=<region>
REPO_NAME=produtos

# Login no ECR
aws ecr get-login-password --region $AWS_REGION | docker login
--username AWS --password-stdin
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com

# Criar repositório no ECR
aws ecr create-repository --repository-name $REPO_NAME --
region $AWS_REGION

# Tag e push da imagem
docker build -t produtos .
docker tag produtos:latest
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:l
atest
docker push
```

```
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:l
atest
```

- **Configuração do ECS:**
  - Crie um cluster ECS via AWS CLI ou console.
  - Defina uma task definition utilizando a imagem do ECR.
  - Crie um serviço ECS associado à task definition.

- **Automação:**
  - Crie um script `deploy-ecs.sh`:

```
cat <<EOL > deploy-ecs.sh
#!/bin/bash

set -e

AWS_ACCOUNT_ID=<account_id>
AWS_REGION=<region>
REPO_NAME=produtos
CLUSTER_NAME=produtos-cluster
SERVICE_NAME=produtos-service
TASK_DEFINITION=produtos-task

# Login no ECR
aws ecr get-login-password --region $AWS_REGION | docker login
--username AWS --password-stdin
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com

# Criar repositório no ECR (ignore se já existir)
aws ecr create-repository --repository-name $REPO_NAME --
region $AWS_REGION || echo "Repositório já existe"

# Build e push da imagem
docker build -t $REPO_NAME .
docker tag $REPO_NAME:latest
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:l
atest
docker push
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$REPO_NAME:l
atest

# Registrar task definition
aws ecs register-task-definition \
    --family $TASK_DEFINITION \
    --network-mode awsvpc \
    --requires-compatibilities FARGATE \
    --cpu "256" \
    --memory "512" \
    --container-definitions '[
```

```
        {
            "name": "produtos",
            "image":
"'"$AWS_ACCOUNT_ID"'.dkr.ecr.'"$AWS_REGION"'.amazonaws.com/'"$
REPO_NAME"':latest",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 8080,
                    "hostPort": 8080,
                    "protocol": "tcp"
                }
            ]
        }
    ]'

# Criar cluster ECS
aws ecs create-cluster --cluster-name $CLUSTER_NAME || echo
"Cluster já existe"

# Criar serviço ECS
aws ecs create-service \
    --cluster $CLUSTER_NAME \
    --service-name $SERVICE_NAME \
    --task-definition $TASK_DEFINITION \
    --desired-count 1 \
    --launch-type FARGATE \
    --network-configuration '{
        "awsvpcConfiguration": {
            "subnets": ["subnet-xxxxxxxx"],
            "securityGroups": ["sg-xxxxxxxx"],
            "assignPublicIp": "ENABLED"
        }
    }' || echo "Serviço já existe"
EOL

chmod +x deploy-ecs.sh
git add deploy-ecs.sh
git commit -m "chore: Configurar deployment na AWS ECS"
```

- **Nota:**
  - Substitua `<account_id>`, `<region>`, `subnet-xxxxxxxx` e `sg-xxxxxxxx` pelos valores apropriados.

7. Testes de Mensageria e Deployment

Commit 7: `test: Testar mensageria e deployment na AWS ECS`

- **Ação:**

- Execute o script de deployment:

```
./deploy-ecs.sh
```

- Verifique no AWS ECS se o serviço está rodando.
- Crie um produto via API e verifique se a mensagem é publicada e consumida corretamente.

- **Automação:**
  - Este passo envolve ações manuais de verificação via AWS Console e logs.

8. Finalização do Exercício 3

Commit 8: `chore: Finalizar Exercício 3 - Escalabilidade e Mensageria com AWS e RabbitMQ`

- **Ação:**
  - Revisão das implementações de mensageria e deployment na AWS.
  - Atualizar `README.md` com informações sobre RabbitMQ e deployment na AWS.
- **Automação:**
  - Use um script:

```
git commit --allow-empty -m "chore: Finalizar Exercício 3 - Escalabilidade e Mensageria com AWS e RabbitMQ"
```

Exercício 4: Aplicação dos Princípios SOLID

Descrição:

- Revise o código do microsserviço desenvolvido nos exercícios anteriores e refatore-o para garantir que ele esteja aderente aos princípios SOLID.
- Adicione comentários explicando as modificações feitas e como elas melhoram o código em termos de design e manutenibilidade.

Passo a Passo:

1. Revisão e Refatoração para SRP (Responsabilidade Única)

Commit 1: `refactor: Aplicar SRP - Separar lógica de negócios da mensageria`

- **Ação:**
  - Certifique-se de que `ProdutoService` lida apenas com operações CRUD.

- Mensageria já está separada em `MensagemPublisher`.

- **Automação:**
  - Se já separado, apenas confirmar:

```
git commit --allow-empty -m "refactor: Aplicar SRP - Separar lógica de negócios da mensageria"
```

2. Aplicação do OCP (Aberto/Fechado) com Interfaces

Commit 2: `refactor: Aplicar OCP - Utilizar interfaces para serviços`

- **Ação:**
  - Defina interfaces para os serviços.

```
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;
import java.util.List;
import java.util.Optional;

public interface ProdutoServiceInterface {
    Produto criarProduto(Produto produto);
    List<Produto> listarProdutos();
    Optional<Produto> buscarProdutoPorId(Long id);
    Produto atualizarProduto(Long id, Produto produto);
    void deletarProduto(Long id);
}
```

  - Implemente a interface em `ProdutoService`:

```
@Service
public class ProdutoService implements ProdutoServiceInterface
{
    // Implementação existente
}
```

- **Automação:**
  - Use um script:

```
# Crie a interface
mkdir -p src/main/java/com/exemplo/produtos/service
```

```
cat <<EOL >
src/main/java/com/exemplo/produtos/service/ProdutoServiceInter
face.java
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;
import java.util.List;
import java.util.Optional;

public interface ProdutoServiceInterface {
    Produto criarProduto(Produto produto);
    List<Produto> listarProdutos();
    Optional<Produto> buscarProdutoPorId(Long id);
    Produto atualizarProduto(Long id, Produto produto);
    void deletarProduto(Long id);
}
EOL

# Atualize ProdutoService.java para implementar a interface
sed -i 's/public class ProdutoService {/public class
ProdutoService implements ProdutoServiceInterface {/'
src/main/java/com/exemplo/produtos/service/ProdutoService.java

git add
src/main/java/com/exemplo/produtos/service/ProdutoServiceInter
face.java
src/main/java/com/exemplo/produtos/service/ProdutoService.java
git commit -m "refactor: Aplicar OCP - Utilizar interfaces
para serviços"
```

3. Garantia do LSP (Substituição de Liskov)

Commit 3: `refactor: Garantir LSP - Implementações corretas das interfaces`

- **Ação:**
  - Verifique se todas as implementações de `ProdutoServiceInterface` seguem o contrato sem alterar comportamentos esperados.
- **Automação:**
  - Se as implementações já seguem, apenas confirmar:

```
git commit --allow-empty -m "refactor: Garantir LSP -
Implementações corretas das interfaces"
```

4. Segregação de Interfaces (ISP)

Commit 4: `refactor: Aplicar ISP - Segregar interfaces de serviço`

- **Ação:**
  - Se necessário, segmente `ProdutoServiceInterface` em interfaces menores (e.g., leitura e escrita).
  - Exemplo:

```
public interface ProdutoReadService {
    List<Produto> listarProdutos();
    Optional<Produto> buscarProdutoPorId(Long id);
}

public interface ProdutoWriteService {
    Produto criarProduto(Produto produto);
    Produto atualizarProduto(Long id, Produto produto);
    void deletarProduto(Long id);
}
```

  - Atualize `ProdutoService` para implementar as interfaces segregadas.
- **Automação:**
  - Use um script para criar novas interfaces e atualizar o serviço:

```
# Crie ProdutoReadService.java
cat <<EOL >
src/main/java/com/exemplo/produtos/service/ProdutoReadService.
java
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;
import java.util.List;
import java.util.Optional;

public interface ProdutoReadService {
    List<Produto> listarProdutos();
    Optional<Produto> buscarProdutoPorId(Long id);
}
EOL

# Crie ProdutoWriteService.java
cat <<EOL >
src/main/java/com/exemplo/produtos/service/ProdutoWriteService
.java
package com.exemplo.produtos.service;

import com.exemplo.produtos.model.Produto;

public interface ProdutoWriteService {
    Produto criarProduto(Produto produto);
```

```
    Produto atualizarProduto(Long id, Produto produto);
    void deletarProduto(Long id);
}
EOL

# Atualize ProdutoService.java para implementar as novas interfaces
sed -i 's/public class ProdutoService implements ProdutoServiceInterface {/public class ProdutoService implements ProdutoReadService, ProdutoWriteService {/' src/main/java/com/exemplo/produtos/service/ProdutoService.java

git add src/main/java/com/exemplo/produtos/service/ProdutoReadService.java
src/main/java/com/exemplo/produtos/service/ProdutoWriteService.java
src/main/java/com/exemplo/produtos/service/ProdutoService.java
git commit -m "refactor: Aplicar ISP - Segregar interfaces de serviço"
```

5. Inversão de Dependência (DIP)

Commit 5: `refactor: Aplicar DIP - Inverter dependências com injeção de interfaces`

- **Ação:**
  - Assegure que `ProdutoService` depende de interfaces, não de implementações concretas.
  - Já implementado ao utilizar `ProdutoReadService` e `ProdutoWriteService`.
- **Automação:**
  - Confirmar sem alterações adicionais:

```
git commit --allow-empty -m "refactor: Aplicar DIP - Inverter dependências com injeção de interfaces"
```

6. Adição de Comentários Explicativos

Commit 6: `docs: Adicionar comentários sobre refatorações SOLID`

- **Ação:**
  - Adicione comentários no código explicando as refatorações e melhorias.
- **Automação:**

- Inserir comentários manualmente ou via script (exemplo para `ProdutoService.java`):

```
// ProdutoService.java

@Service
public class ProdutoService implements ProdutoReadService,
ProdutoWriteService {

    @Autowired
    private ProdutoRepository produtoRepository;

    @Autowired
    private MensagemPublisher mensagemPublisher;

    /**
     * Cria um novo produto e publica uma mensagem para
RabbitMQ.
     * Aplicação do SRP: Serviço dedicado para operações de
escrita.
     */
    @Override
    public Produto criarProduto(Produto produto) {
        Produto salvo = produtoRepository.save(produto);
        mensagemPublisher.enviarMensagem("Produto criado: " +
salvo.getNome());
        return salvo;
    }

    // Outros métodos com comentários similares...
}
```

- Use um script para adicionar um comentário de exemplo:

```
sed -i '/public class ProdutoService implements
ProdutoReadService, ProdutoWriteService {/a \
\n    /**\n    * Cria um novo produto e publica uma mensagem
para RabbitMQ.\n    * Aplicação do SRP: Serviço dedicado para
operações de escrita.\n    */\n\
'
src/main/java/com/exemplo/produtos/service/ProdutoService.java

git add
src/main/java/com/exemplo/produtos/service/ProdutoService.java
git commit -m "docs: Adicionar comentários sobre refatorações
SOLID"
```

7. Finalização do Exercício 4

Commit 7: `chore: Finalizar Exercício 4 - Aplicação dos Princípios SOLID`

- **Ação:**
  - Revisão final das refatorações e documentação.
- **Automação:**
  - Use um script:

```
git commit --allow-empty -m "chore: Finalizar Exercício 4 - Aplicação dos Princípios SOLID"
```

Exercício 5: Integração de Banco de Dados e Migrações

Descrição:

- Integre uma base de dados MySQL com o microsserviço, incluindo a configuração de migrações de banco de dados usando Flyway.
- Crie um script de migração inicial que configure a tabela de produtos.
- Adicione um endpoint na API que permita buscar produtos com filtros específicos (ex: por nome, categoria, preço).

Passo a Passo:

1. Adição de Dependências para MySQL e Flyway

Commit 1: `chore: Adicionar dependências para MySQL e Flyway`

- **Ação:**
  - Atualize `pom.xml`:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
```

- **Automação:**
  - Use um script:

```
cat <<EOL >> pom.xml

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.flywaydb</groupId>
        <artifactId>flyway-core</artifactId>
    </dependency>
EOL
git add pom.xml
git commit -m "chore: Adicionar dependências para MySQL e
Flyway"
```

2. Configuração do MySQL no application.properties

Commit 2: `chore: Configurar conexão com MySQL no application.properties`

- **Ação:**
  - Atualize `src/main/resources/application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/produtosdb
spring.datasource.username=root
spring.datasource.password=senha
spring.jpa.hibernate.ddl-auto=validate
spring.flyway.enabled=true
spring.flyway.locations=classpath:db/migration
```

- **Automação:**
  - Use um script:

```
cat <<EOL >> src/main/resources/application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/produtosdb
spring.datasource.username=root
spring.datasource.password=senha
spring.jpa.hibernate.ddl-auto=validate
spring.flyway.enabled=true
spring.flyway.locations=classpath:db/migration
EOL
git add src/main/resources/application.properties
git commit -m "chore: Configurar conexão com MySQL no
application.properties"
```

3. Criação do Script de Migração Inicial com Flyway

Commit 3: `chore: Criar script de migração inicial para tabela de produtos`

- **Ação:**
  - Crie `src/main/resources/db/migration/V1__Create_Produto_Table.sql`:

```
CREATE TABLE produtos (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    categoria VARCHAR(255) NOT NULL,
    preco DOUBLE NOT NULL,
    quantidade INT NOT NULL
);
```

- **Automação:**
  - Use um script:

```
mkdir -p src/main/resources/db/migration
cat <<EOL >
src/main/resources/db/migration/V1__Create_Produto_Table.sql
CREATE TABLE produtos (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    categoria VARCHAR(255) NOT NULL,
    preco DOUBLE NOT NULL,
    quantidade INT NOT NULL
);
EOL
git add
src/main/resources/db/migration/V1__Create_Produto_Table.sql
git commit -m "chore: Criar script de migração inicial para
tabela de produtos"
```

4. Atualização da Entidade Produto para MySQL

Commit 4: `feat: Adaptar entidade Produto para MySQL`

- **Ação:**
  - Verifique se a entidade `Produto` está alinhada com a tabela de migração.
  - Adicione validações se necessário (já implementado anteriormente).
- **Automação:**

- Confirmação sem alterações:

```
git commit --allow-empty -m "feat: Adaptar entidade Produto
para MySQL"
```

5. Implementação de Métodos de Busca com Filtros no Repositório

Commit 5: `feat: Implementar métodos de busca com filtros no ProdutoRepository`

- **Ação:**
  - Atualize `ProdutoRepository` para incluir métodos de busca:

```
import java.util.List;

public interface ProdutoRepository extends
JpaRepository<Produto, Long> {
    List<Produto> findByNomeContaining(String nome);
    List<Produto> findByCategoria(String categoria);
    List<Produto> findByPrecoBetween(Double precoMin, Double
precoMax);
}
```

- **Automação:**
  - Use um script:

```
sed -i '/public interface ProdutoRepository extends
JpaRepository<Produto, Long> {/a \
\n    List<Produto> findByNomeContaining(String nome);\n
List<Produto> findByCategoria(String categoria);\n
List<Produto> findByPrecoBetween(Double precoMin, Double
precoMax);\n\
'
src/main/java/com/exemplo/produtos/repository/ProdutoRepositor
y.java

git add
src/main/java/com/exemplo/produtos/repository/ProdutoRepositor
y.java
git commit -m "feat: Implementar métodos de busca com filtros
no ProdutoRepository"
```

6. Atualização do ProdutoService para Suporte a Filtros

Commit 6: `feat: Adicionar métodos de busca com filtros no ProdutoService`

- **Ação:**
    - Adicione métodos no `ProdutoService`:

```
public List<Produto> buscarProdutosPorNome(String nome) {
    return produtoRepository.findByNomeContaining(nome);
}

public List<Produto> buscarProdutosPorCategoria(String
categoria) {
    return produtoRepository.findByCategoria(categoria);
}

public List<Produto> buscarProdutosPorPreco(Double precoMin,
Double precoMax) {
    return produtoRepository.findByPrecoBetween(precoMin,
precoMax);
}
```

- **Automação:**
    - Use um script:

```
sed -i '/public class ProdutoService implements
ProdutoReadService, ProdutoWriteService {/a \
\n    public List<Produto> buscarProdutosPorNome(String nome)
{\n        return
produtoRepository.findByNomeContaining(nome);\n    }\n\
\n    public List<Produto> buscarProdutosPorCategoria(String
categoria) {\n        return
produtoRepository.findByCategoria(categoria);\n    }\n\
\n    public List<Produto> buscarProdutosPorPreco(Double
precoMin, Double precoMax) {\n        return
produtoRepository.findByPrecoBetween(precoMin, precoMax);\n
}\n\
'
src/main/java/com/exemplo/produtos/service/ProdutoService.java

git add
src/main/java/com/exemplo/produtos/service/ProdutoService.java
git commit -m "feat: Adicionar métodos de busca com filtros no
ProdutoService"
```

7. Adição de Endpoints para Busca com Filtros no ProdutoController

Commit 7: `feat: Adicionar endpoints para busca de produtos com filtros`

- **Ação:**

- Atualize `ProdutoController`:

```
@GetMapping("/buscar")
public ResponseEntity<List<Produto>> buscarProdutos(
    @RequestParam(required = false) String nome,
    @RequestParam(required = false) String categoria,
    @RequestParam(required = false) Double precoMin,
    @RequestParam(required = false) Double precoMax) {

    List<Produto> produtos;

    if (nome != null) {
        produtos = produtoService.buscarProdutosPorNome(nome);
    } else if (categoria != null) {
        produtos =
produtoService.buscarProdutosPorCategoria(categoria);
    } else if (precoMin != null && precoMax != null) {
        produtos =
produtoService.buscarProdutosPorPreco(precoMin, precoMax);
    } else {
        produtos = produtoService.listarProdutos();
    }

    return new ResponseEntity<>(produtos, HttpStatus.OK);
}
```

- **Automação:**
  - Use um script:

```
sed -i '/public class ProdutoController {/a \
\n    @GetMapping("/buscar")\n    public
ResponseEntity<List<Produto>> buscarProdutos(\n
@RequestParam(required = false) String nome,\n
@RequestParam(required = false) String categoria,\n
@RequestParam(required = false) Double precoMin,\n
@RequestParam(required = false) Double precoMax) {\n\n
List<Produto> produtos;\n\n        if (nome != null) {\n
produtos = produtoService.buscarProdutosPorNome(nome);\n
} else if (categoria != null) {\n            produtos =
produtoService.buscarProdutosPorCategoria(categoria);\n
} else if (precoMin != null && precoMax != null) {\n
produtos = produtoService.buscarProdutosPorPreco(precoMin,
precoMax);\n        } else {\n            produtos =
produtoService.listarProdutos();\n        }\n\n        return
new ResponseEntity<>(produtos, HttpStatus.OK);\n    }\n\
'
src/main/java/com/exemplo/produtos/controller/ProdutoControlle
r.java
```

```
git add
src/main/java/com/exemplo/produtos/controller/ProdutoControlle
r.java
git commit -m "feat: Adicionar endpoints para busca de
produtos com filtros"
```

8. Testes Manuais e Verificação das Migrações

Commit 8: `test: Verificar migrações com Flyway e testes manuais de endpoints`

- **Ação:**
  - Rode a aplicação:

    ```
    mvn clean install
    mvn spring-boot:run
    ```

  - Acesse o MySQL e verifique se a tabela `produtos` foi criada.
  - Utilize **Postman** ou **cURL** para testar os endpoints CRUD e de busca com filtros.
- **Automação:**
  - Embora os testes manuais não sejam automatizados, certifique-se de que Flyway aplica as migrações corretamente ao iniciar a aplicação.

9. Finalização do Exercício 5

Commit 9: `chore: Finalizar Exercício 5 – Integração de Banco de Dados e Migrações`

- **Ação:**
  - Revisão final das integrações e testes.
  - Atualizar `README.md` com informações sobre a integração com MySQL e Flyway.
- **Automação:**
  - Use um script:

    ```
    git commit --allow-empty -m "chore: Finalizar Exercício 5 –
    Integração de Banco de Dados e Migrações"
    ```

Questão Complementar: Implementação de Autoscaling e Balanceamento de Carga na AWS

Descrição:

- Utilize o microsserviço desenvolvido nos exercícios anteriores.
- Configure o autoscaling e o balanceamento de carga para o microsserviço utilizando os serviços da AWS (ex: Elastic Load Balancing (ELB) e Auto Scaling).

Passo a Passo:

1. Preparação do Ambiente

Commit 1: `chore: Criar Dockerfile e fazer upload para AWS ECR`

- **Ação:**
  - Certifique-se de que o `Dockerfile` está criado (feito no Exercício 3).
  - Execute o script de deployment no Exercício 3 para push da imagem para ECR.
- **Automação:**
  - Já realizado no Exercício 3.

2. Configuração do Amazon ECS

Commit 2: `chore: Configurar cluster ECS e task definition`

- **Ação:**
  - Utilize o script `deploy-ecs.sh` criado anteriormente para configurar o cluster ECS e registrar a task definition.
- **Automação:**
  - Use o script de deployment:

```
./deploy-ecs.sh
```

3. Configuração de Autoscaling

Commit 3: `chore: Configurar política de autoscaling no ECS`

- **Ação:**
  - Configure políticas de autoscaling no AWS ECS baseado em métricas de CPU.
  - Utilize AWS CLI ou Console para definir as políticas:
    - **Aumentar Instâncias:** CPU > 70%
    - **Diminuir Instâncias:** CPU < 30%
- **Automação:**
  - Adicione ao script `deploy-ecs.sh`:

```
# Adicionar ao deploy-ecs.sh após a criação do serviço
aws application-autoscaling register-scalable-target \
    --service-namespace ecs \
    --resource-id service/$CLUSTER_NAME/$SERVICE_NAME \
    --scalable-dimension ecs:service:DesiredCount \
    --min-capacity 1 \
    --max-capacity 10

aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --resource-id service/$CLUSTER_NAME/$SERVICE_NAME \
    --scalable-dimension ecs:service:DesiredCount \
    --policy-name cpu-scaling-policy-up \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration
file://policy-up.json

aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --resource-id service/$CLUSTER_NAME/$SERVICE_NAME \
    --scalable-dimension ecs:service:DesiredCount \
    --policy-name cpu-scaling-policy-down \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration
file://policy-down.json
```

- Crie `policy-up.json` e `policy-down.json`:

```
mkdir -p policies
cat <<EOL > policies/policy-up.json
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification": {
        "PredefinedMetricType":
"ECSServiceAverageCPUUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60
}
EOL

cat <<EOL > policies/policy-down.json
{
    "TargetValue": 30.0,
    "PredefinedMetricSpecification": {
        "PredefinedMetricType":
"ECSServiceAverageCPUUtilization"
    },
    "ScaleOutCooldown": 60,
```

```
    "ScaleInCooldown": 60
}
EOL
```

- Atualize `deploy-ecs.sh` para incluir as políticas:

```
# Adicione ao deploy-ecs.sh
aws application-autoscaling register-scalable-target \
    --service-namespace ecs \
    --resource-id service/$CLUSTER_NAME/$SERVICE_NAME \
    --scalable-dimension ecs:service:DesiredCount \
    --min-capacity 1 \
    --max-capacity 10

aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --resource-id service/$CLUSTER_NAME/$SERVICE_NAME \
    --scalable-dimension ecs:service:DesiredCount \
    --policy-name cpu-scaling-policy-up \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration
file://policies/policy-up.json

aws application-autoscaling put-scaling-policy \
    --service-namespace ecs \
    --resource-id service/$CLUSTER_NAME/$SERVICE_NAME \
    --scalable-dimension ecs:service:DesiredCount \
    --policy-name cpu-scaling-policy-down \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration
file://policies/policy-down.json
```

- Atualize o repositório:

```
git add deploy-ecs.sh policies/policy-up.json policies/policy-
down.json
git commit -m "chore: Configurar política de autoscaling no
ECS"
```

4. Configuração de Balanceamento de Carga

Commit 4: `chore: Configurar Application Load Balancer (ALB)`

- **Ação:**
  - Crie um ALB via AWS CLI ou Console.
  - Configure listeners e regras para direcionar o tráfego para o serviço ECS.

- **Automação:**
  - Adicione ao script `deploy-ecs.sh`:

```
# Criar ALB
aws elbv2 create-load-balancer --name produtos-alb --subnets
subnet-xxxxxxxx subnet-yyyyyyyy --security-groups sg-xxxxxxxx
--scheme internet-facing --type application

# Criar Target Group
aws elbv2 create-target-group --name produtos-tg --protocol
HTTP --port 8080 --vpc-id vpc-xxxxxxxx

# Criar Listener
aws elbv2 create-listener --load-balancer-arn <ALB_ARN> --
protocol HTTP --port 80 --default-actions
Type=forward,TargetGroupArn=<TG_ARN>

# Registrar serviço ECS com ALB
aws ecs update-service \
    --cluster $CLUSTER_NAME \
    --service $SERVICE_NAME \
    --load-balancers "targetGroupArn=
<TG_ARN>,containerName=produtos,containerPort=8080"
```

  - **Nota:** Substitua `<ALB_ARN>` e `<TG_ARN>` com os ARNs retornados pelos comandos anteriores.
  - Atualize `deploy-ecs.sh` com placeholders ou automatize a captura dos ARNs.
  - Atualize o repositório:

```
git add deploy-ecs.sh
git commit -m "chore: Configurar Application Load Balancer
(ALB)"
```

5. Teste e Monitoramento

Commit 5: `test: Testar autoscaling e balanceamento de carga na AWS`

- **Ação:**
  - **Testar Autoscaling:**
    - Simule cargas altas usando ferramentas como **Apache JMeter**:

```
```

```
# Exemplo de comando JMeter (assumindo que o teste está configurado)
jmeter -n -t teste-load.jmx -l resultados.jtl
```

- **Verificar Balanceamento de Carga:**
  - Envie múltiplas requisições e verifique se as respostas vêm de diferentes instâncias.

```
for i in {1..10}; do curl http://<ALB_DNS>/api/produtos; done
```

- **Monitoramento:**
  - Utilize **Amazon CloudWatch** para acompanhar métricas de CPU, memória e tráfego no ALB.
  - Configure dashboards no CloudWatch para visualização.

- **Automação:**
  - Scripts para simular carga podem ser adicionados, mas geralmente envolvem ferramentas externas.

6. Finalização da Questão Complementar

Commit 6: `chore: Finalizar Questão Complementar - Autoscaling e Balanceamento de Carga`

- **Ação:**
  - Revisão das configurações de autoscaling e balanceamento de carga.
  - Atualizar `README.md` com informações sobre AWS ECS, ALB e políticas de autoscaling.

- **Automação:**
  - Use um script:

```
git commit --allow-empty -m "chore: Finalizar Questão Complementar - Autoscaling e Balanceamento de Carga"
```

Instruções Gerais para Todos os Exercícios

1. **Criação de Repositórios Separados:**
   - Cada exercício deve ser um repositório Git separado.

- Nomeie os repositórios de forma descritiva, ex: `produtos-microsservico`, `produtos-microsservico-tests`, etc.

2. **Boas Práticas de Programação:**

   - Utilize **nomenclatura clara** e consistente.
   - Mantenha o **código limpo** e bem **documentado**.
   - Adote **padrões de projeto** para garantir a escalabilidade e manutenção.

3. **Documentação:**

   - Atualize o `README.md` de cada projeto com:
     - Descrição do projeto.
     - Instruções de configuração e execução.
     - Detalhes sobre como testar a aplicação.
     - Informações sobre o pipeline de CI/CD, se aplicável.

4. **Versionamento e Branching:**

   - Utilize o modelo **GitFlow** ou similar.
   - Crie branches para novas funcionalidades, correções de bugs e melhorias.
   - Realize **pull requests** para integrar mudanças nas branches principais, garantindo revisão de código.

5. **Automação e CI/CD:**

   - Configure pipelines de CI/CD para automatizar builds, testes e deploys.
   - Utilize ferramentas como **GitHub Actions**, **GitLab CI** ou **Jenkins**.
   - Assegure que o pipeline inclua etapas de build, execução de testes e deploy (se aplicável).

6. **Segurança:**

   - Não exponha informações sensíveis (como senhas ou chaves de API) no código-fonte.
   - Utilize variáveis de ambiente ou serviços de gerenciamento de segredos para armazenar credenciais.
   - Implemente **autenticação** e **autorização** robustas nos microsserviços.

7. **Monitoramento e Logging:**

   - Integre ferramentas de monitoramento como **Prometheus** e **Grafana** para acompanhar o desempenho dos microsserviços.
   - Configure logs estruturados para facilitar a análise e detecção de problemas.
   - Utilize soluções como **ELK Stack** (Elasticsearch, Logstash, Kibana) para centralizar e visualizar logs.

Considerações Finais

Este guia simplificado e automatizado permite a implementação eficiente dos exercícios propostos, garantindo que cada etapa seja realizada de forma organizada e aderente às boas práticas de

desenvolvimento de microsserviços com Spring Boot no ecossistema Java. Ao seguir este passo a passo, você poderá construir um sistema robusto, escalável e bem testado, preparado para atender às demandas de negócios e tecnológicos.

Nota: Adapte os scripts e comandos conforme as necessidades específicas do seu ambiente e substitua os placeholders (`<account_id>`, `<region>`, etc.) pelos valores apropriados.

Se precisar de mais detalhes ou assistência em alguma das etapas, sinta-se à vontade para perguntar!