

Testando uma aplicação em uma arquitetura orientada a eventos

Henrique Schmidt

@henriquels25



<https://www.linkedin.com/in/henriquelschmidt/>



<https://github.com/henriquels25>



<https://henriquels25.github.io/>

Henrique Luis Schmidt

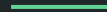
Engenheiro de Software no Sicredi

Agenda

- Arquitetura orientada a eventos
- Cenário de negócio
- Arquitetura do Software
- Testando um consumidor
- Testando um transformador de eventos
- Testando um produtor
- Testando envio para DLQ
- Teste de aceitação

Arquitetura orientada a eventos

Utilização de eventos para
comunicação entre
serviços desacoplados



O que é um evento?

Representa alguma alteração no estado do sistema.

Exemplos:

- Item adicionado a um carrinho de compras;
- Solicitação de transferência de dinheiro entre contas;
- Compra entregue para destinatário.

Componentes principais da arquitetura

1. Produtores de eventos
2. Event routers (roteadores de eventos)
3. Consumidores de eventos

Produtores de eventos

O produtor “sente” o evento e publica para o sistema.

Produtores de eventos

Exemplos:

- Sistema de recebimento de pedidos publica evento de criação de pedido;
- Sistema de recuperação de crédito publica evento de dívida paga;

Consumidores de eventos

O consumidor recebe o evento e reage conforme sua regra de negócio.

Consumidores de eventos

Exemplos:

- Sistema de envio de notificações recebe evento de compra efetuada com cartão de crédito;
- Sistema de análise de fraudes recebe evento de transferência efetuada.

Produtores e Consumidores de eventos

Tipicamente, são microserviços.

Uma mesma aplicação pode ser tanto um produtor e um consumidor para diferentes eventos.

Consumidores de eventos

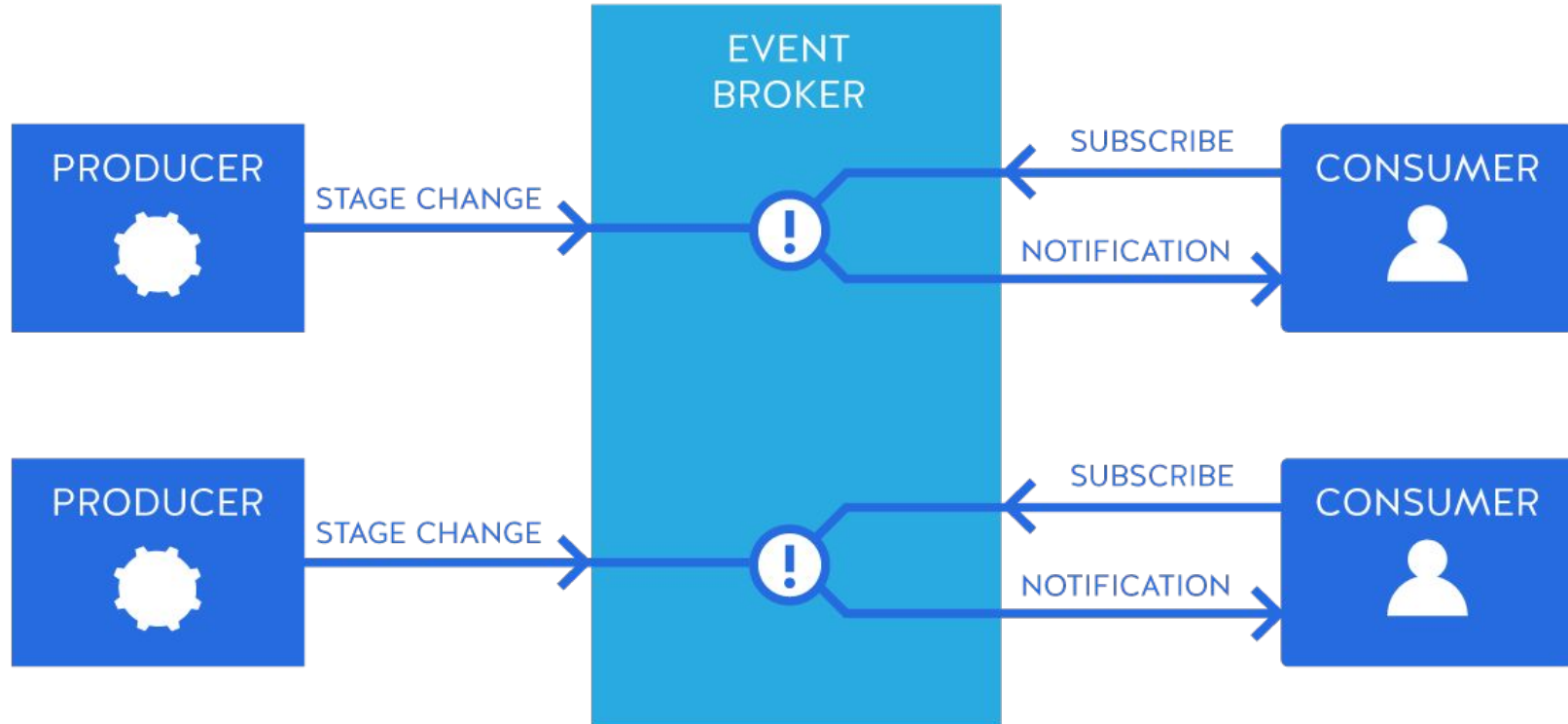
Produtores e Consumidores **NÃO** se conhecem.

Event routers (roteadores de eventos)

Como os consumidores e produtores dos eventos não se conhecem, deve haver algo entre eles, um **middleware**.

Também são chamados de **event brokers**.

Event routers (roteadores de eventos)



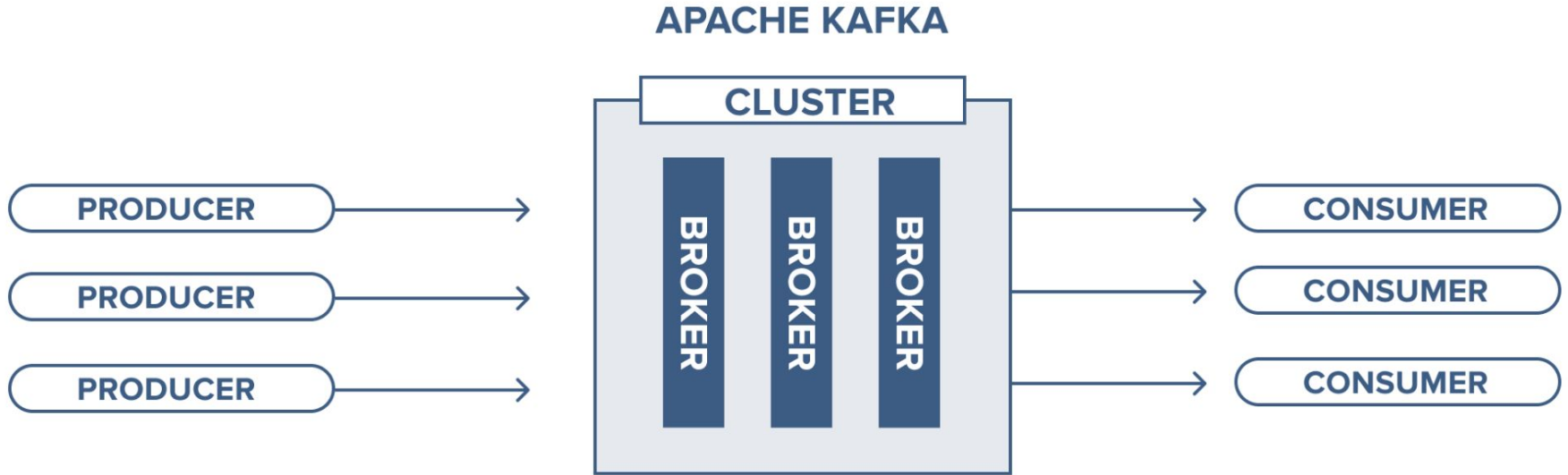
Apache Kafka

O Kafka é um broker de mensagens muito utilizado.

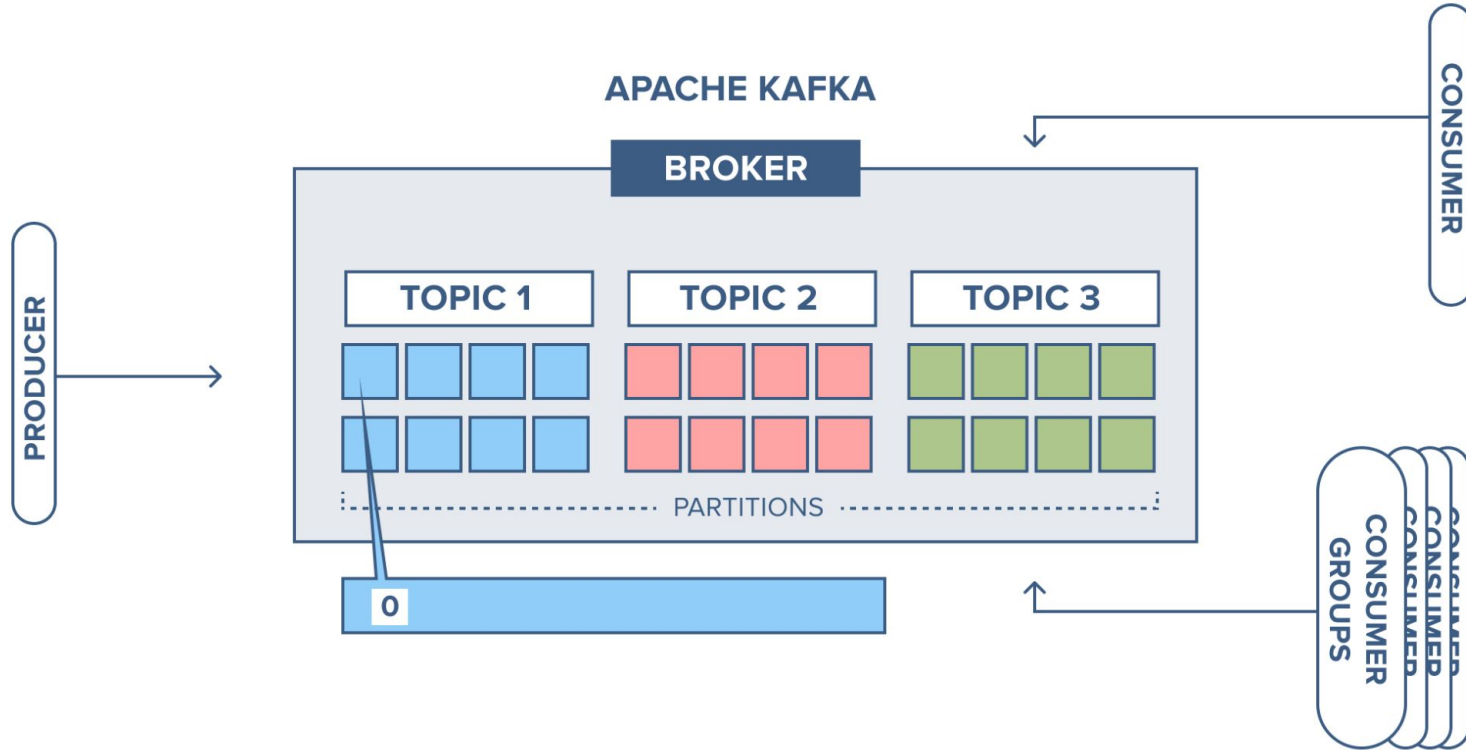
Cada evento é enviado para um **tópico** e consumidores se inscrevem para receberem eventos.



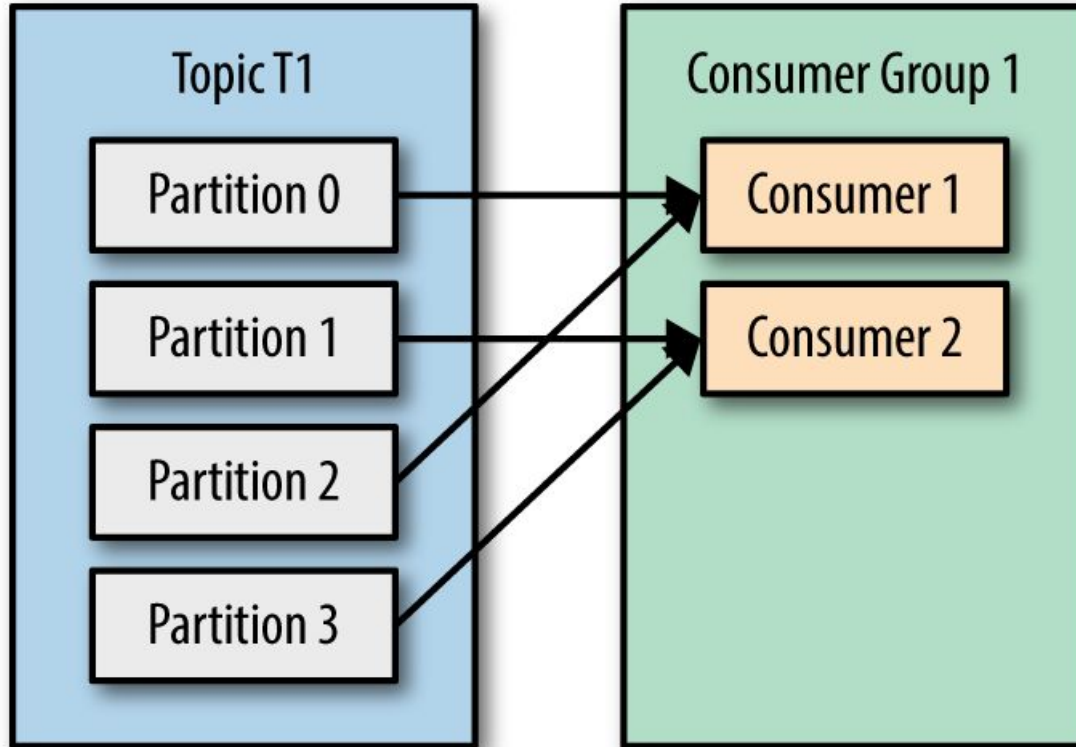
Apache Kafka



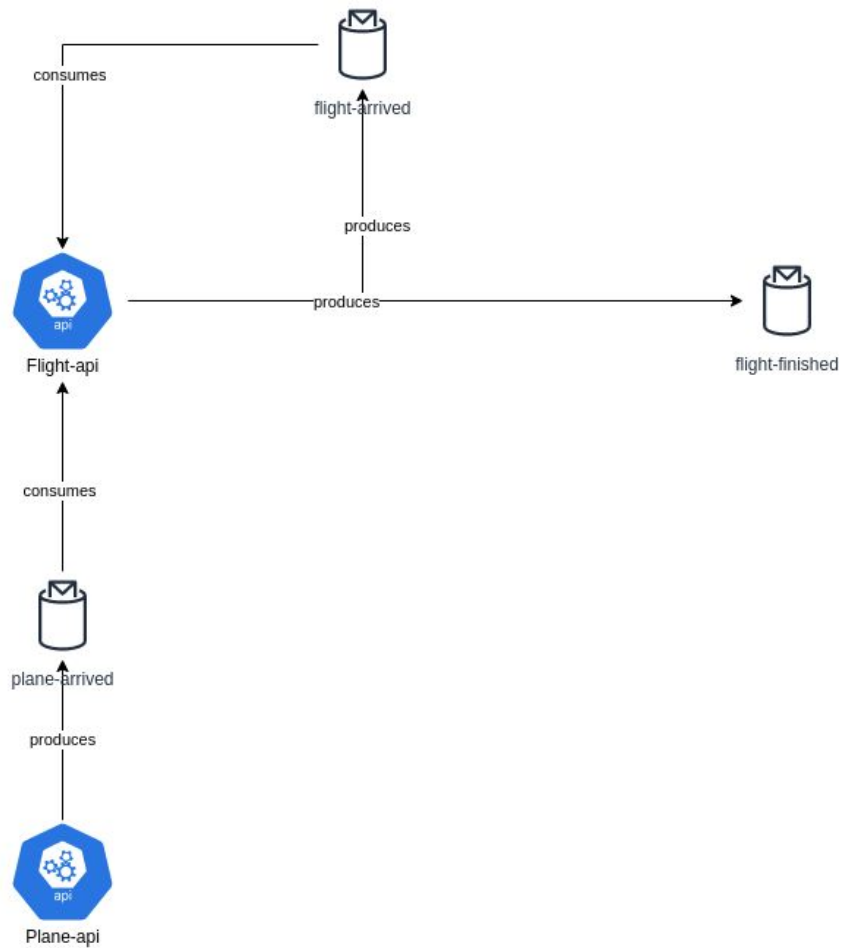
Apache Kafka



Apache Kafka



Cenário de Negócio

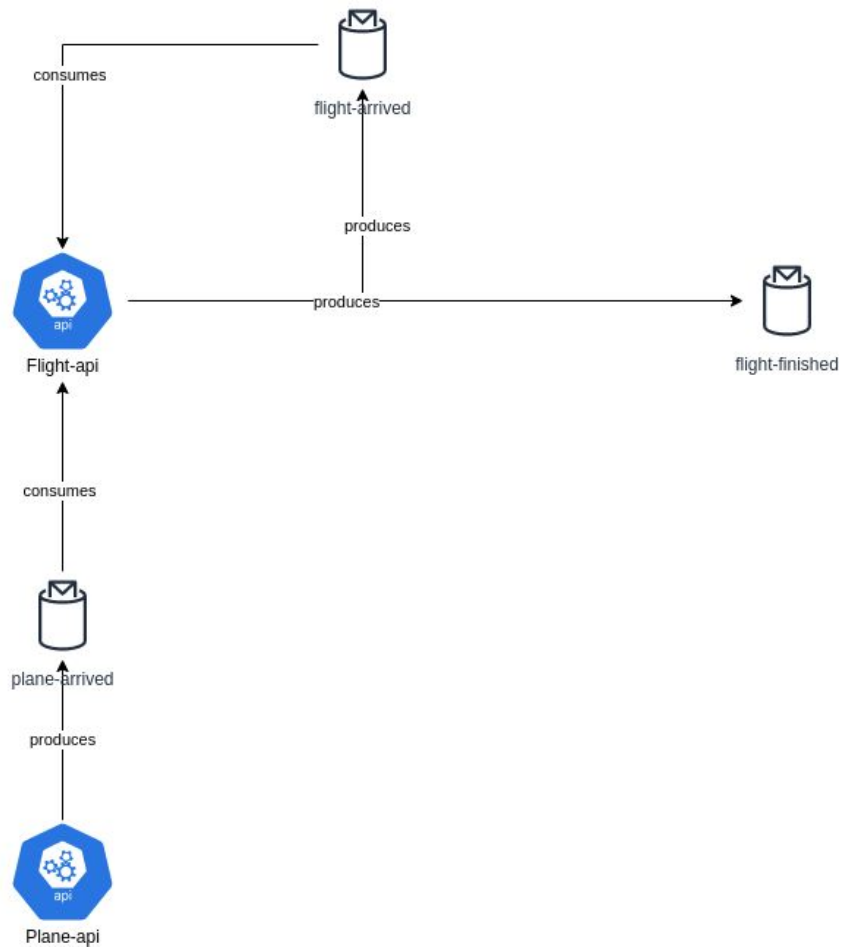


Plane-arrived: Avião pousou

Flight-arrived: Vôo pousou

Flight-finished: Vôo chegou no destino

<https://github.com/henriquels25/spring-cloud-stream-sample/>



Exemplo:
Voo POA-CNH

Primeiro evento:
Plane-arrived - avião pousou em CNH

Segundo evento:
Flight-arrived - vôo pousou em CNH

Terceiro evento:
Flight-finished - vôo pousou no destino

<https://github.com/henriquels25/spring-cloud-stream-sample/>

Arquitetura do Software

Arquitetura Hexagonal

Também chamada de Ports&Adapters

Arquitetura Hexagonal

Criada em 2005, por Alistair Cockburn.

<https://alistair.cockburn.us/hexagonal-architecture/>



@TotherAlistair

Arquitetura Hexagonal

Objetivo #1

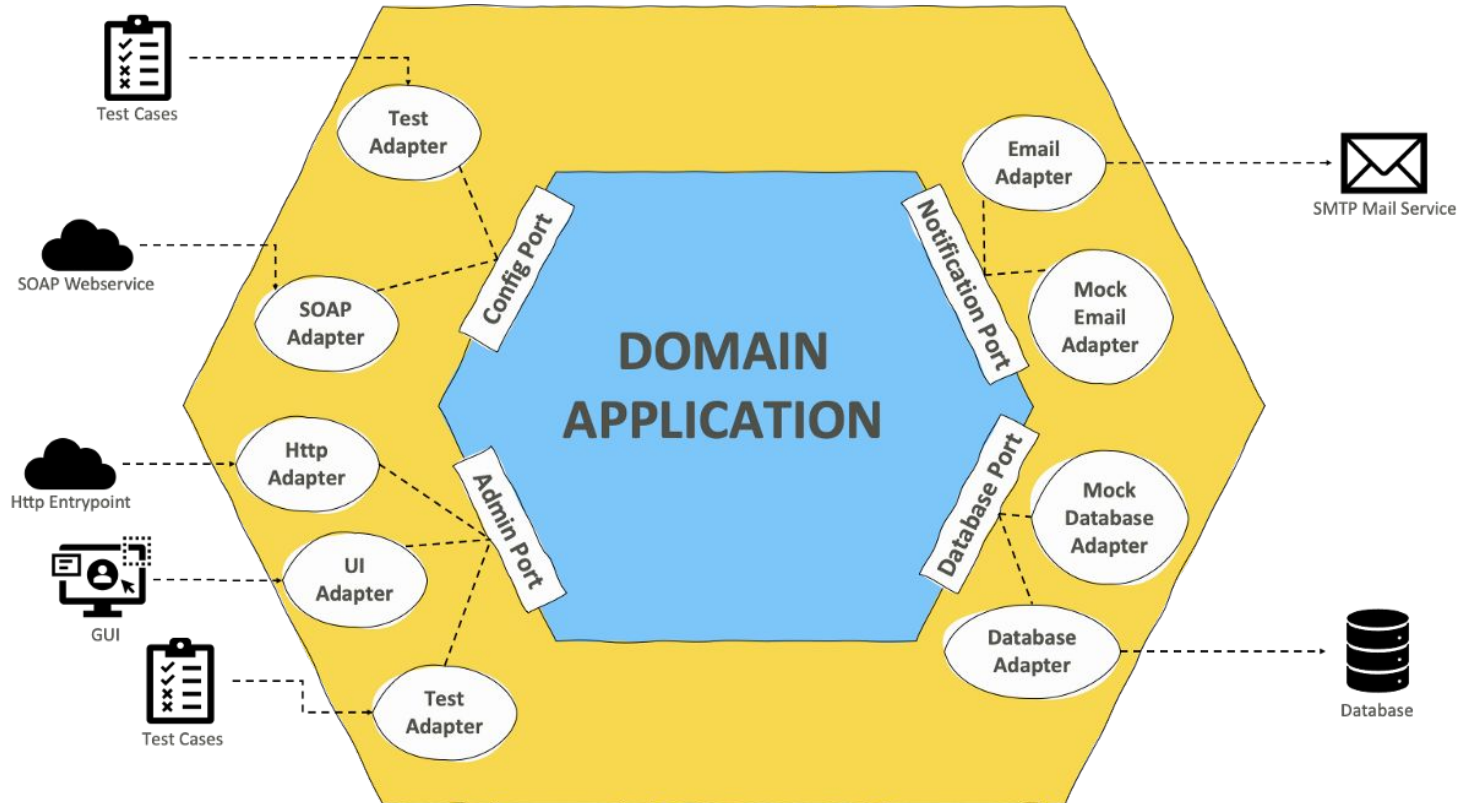
Permitir que uma aplicação seja igualmente guiada por usuários, programas, testes automatizados ou scripts.

Arquitetura Hexagonal

Objetivo #2

Permitir que uma aplicação seja desenvolvida de forma isolada dos banco de dados e dispositivos necessários na hora da execução.

Arquitetura Hexagonal

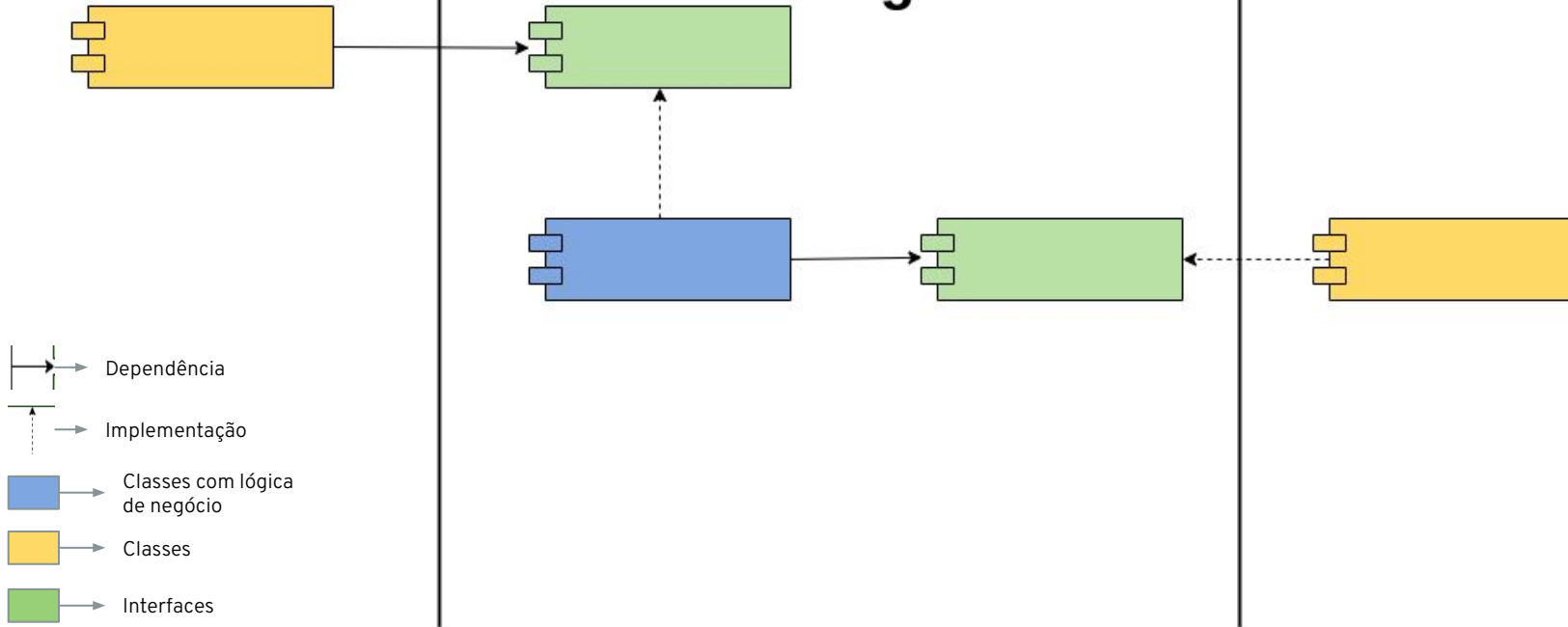


Arquitetura Hexagonal

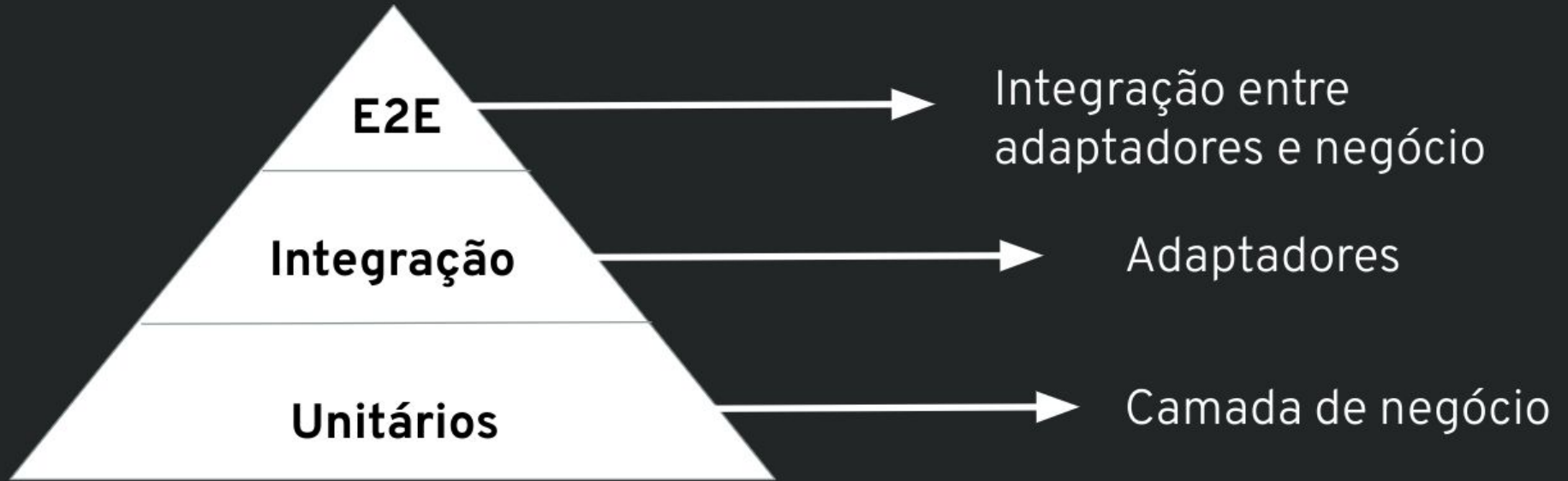
Adaptadores primários

Lógica de Negócio

Adaptadores secundários



Testes Locais

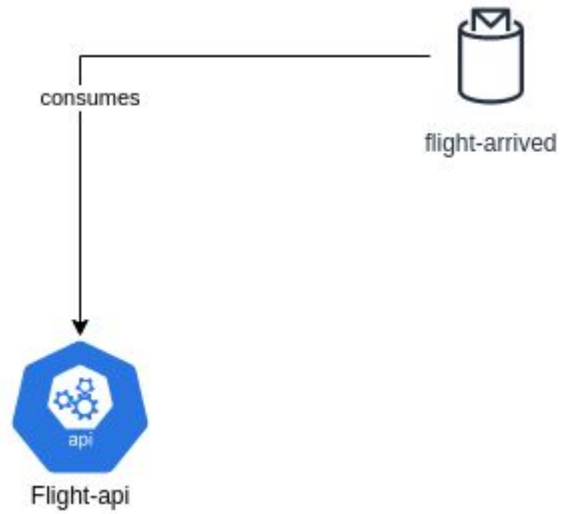


Código Fonte



<https://github.com/henriquels25/spring-cloud-stream-sample/>

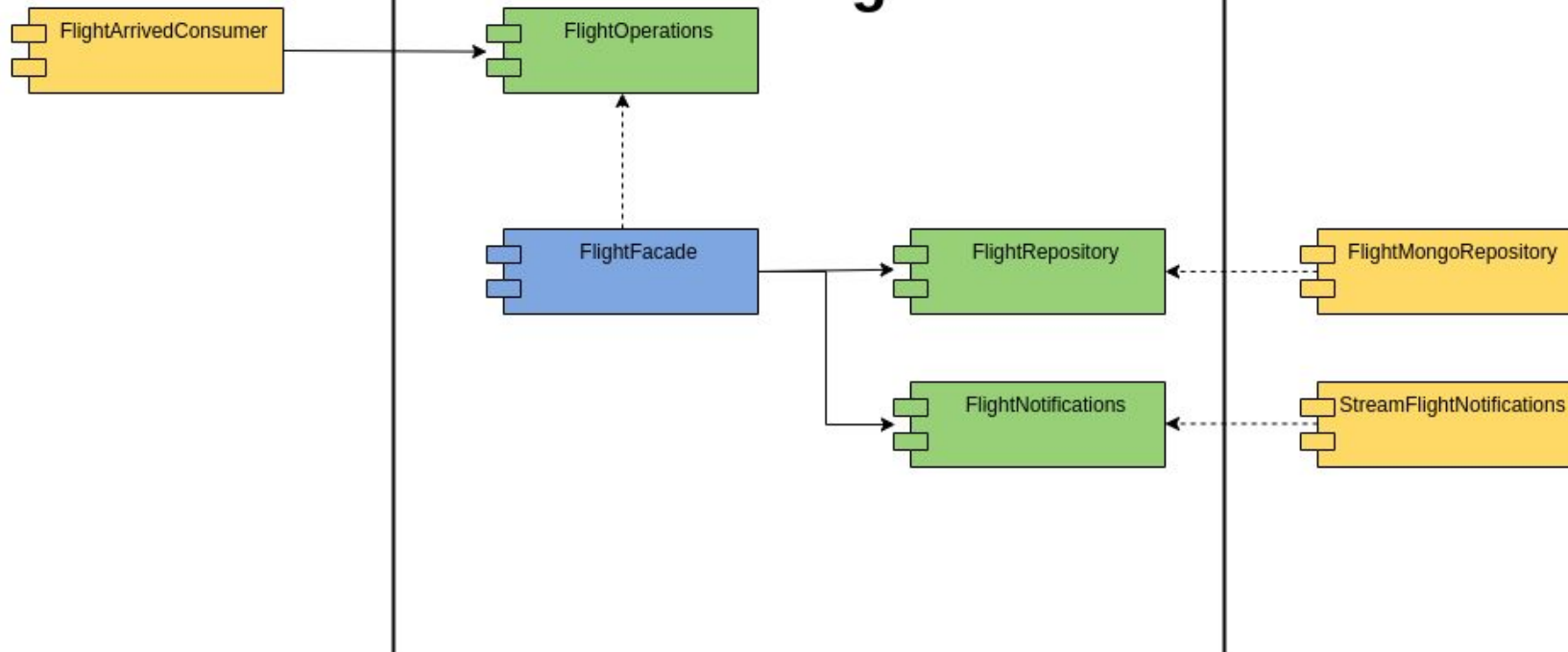
Testando um consumer



Adaptadores primários

Lógica de Negócio

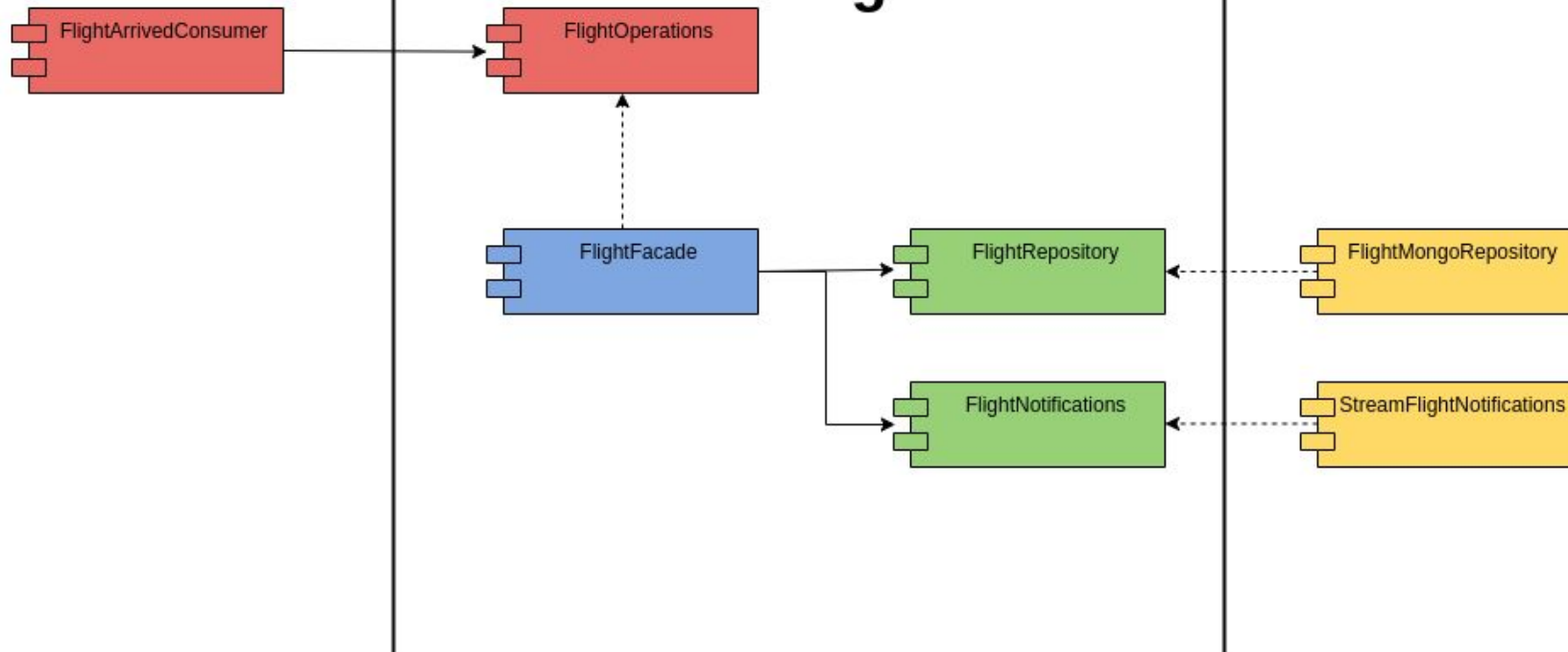
Adaptadores secundários



Adaptadores primários

Lógica de Negócio

Adaptadores secundários



Passos

1 - Levantar um Kafka em memória

Levantando um Kafka em memória

No Spring temos a seguinte alternativa:

```
32      testImplementation 'org.springframework.kafka:spring-kafka-test'
```


[flight-api/build.gradle](#)

```
9      @EmbeddedKafka(topics = {"plane-arrived-v1",  
10          "flight-arrived-v1", "plane-arrived-dlq-v1", "flight-arrived-dlq-v1", "flight-finished-v1"},  
11          bootstrapServersProperty = "spring.cloud.stream.kafka.binder.brokers",  
12          partitions = 1)
```

[flightapi/messaging/utis/EmbeddedKafkaWithTopics](#)

Levantando um Kafka em memória

Em outras tecnologias, poderia ser utilizado testContainers.

 **Testcontainers**

Search

testcontainers-java
1.16.3 5.7k 1.1k


Testcontainers
Home
Quickstart >
Features >
Modules >
 Databases >
 Azure Module
 Docker Compose Module
 Elasticsearch container

Kafka Containers

Testcontainers can be used to automatically instantiate and manage [Apache Kafka](#) containers.
More precisely Testcontainers uses the official Docker images for [Confluent OSS Platform](#)

Benefits

- Running a single node Kafka installation with just one line of code

**Table of contents**
Benefits
Example
Options
 Using external Zookeeper
Multi-container usage
Adding this module to your project dependencies

Passos

1 - Levantar um Kafka em memória

2- Subir a aplicação apontando para o Kafka em memória

```
24 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE)
```

```
25 @EmbeddedKafkaWithTopics
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

```
9 @EmbeddedKafka(topics = {"plane-arrived-v1",  
10     "flight-arrived-v1", "plane-arrived-dlq-v1", "flight-arrived-dlq-v1", "flight-finished-v1"},  
11     bootstrapServersProperty = "spring.cloud.stream.kafka.binder.brokers",  
12     partitions = 1)  
13 public @interface EmbeddedKafkaWithTopics {  
14 }
```

[flightapi/messaging/utis/EmbeddedKafkaWithTopics](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Enviar a mensagem para o tópico que será consumido**

Enviando a mensagem

```
28     @Autowired
29     private EmbeddedKafkaBroker broker;
30
31
32
33
34     private KafkaTestUtils kafkaTestUtils;
35
36
37
38
39     @BeforeEach
40     void prepare() {
41         this.kafkaTestUtils = new KafkaTestUtils(broker);
42     }
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Enviando a mensagem

```
51     public void sendMessage(String topic, String json) {  
52         Producer<String, String> producer = createProducer();  
53  
54         producer.send(new ProducerRecord<>(topic, json));  
55     }  
56
```

```
46     String flightEvent = new JSONObject().put("flightId", FLIGHT_ID)  
47         .put("currentAirport", AIRPORT).toString();  
48  
49     kafkaTestUtils.sendMessage("flight-arrived-v1", flightEvent);  
50
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Enviar a mensagem para o tópico que será consumido
- 4- Validar que a mensagem foi consumida e a camada de negócio foi chamada**

Awaitility

Como todo processamento via mensageria é assíncrono, o awaitility ajuda com uma espera inteligente.



Validando o consumo

30

31 @MockBean

32 private FlightOperations flightOperations;

33

49 kafkaTestUtils.sendMessage("flight-arrived-v1", flightEvent);

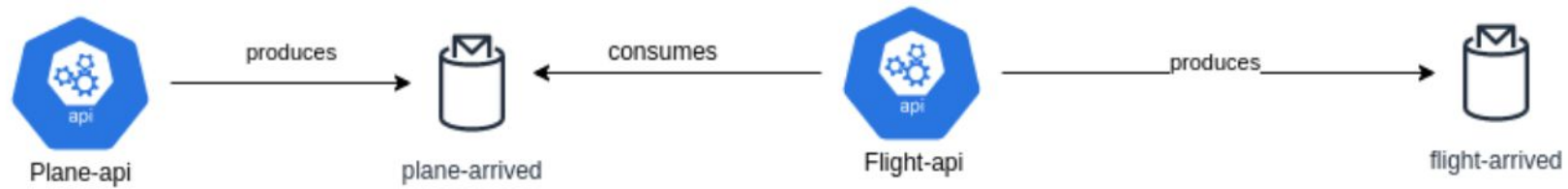
50

51 await().untilAsserted(() -> verify(flightOperations).

52 flightArrivedIn(FLIGHT_ID, new Airport(AIRPORT)));

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

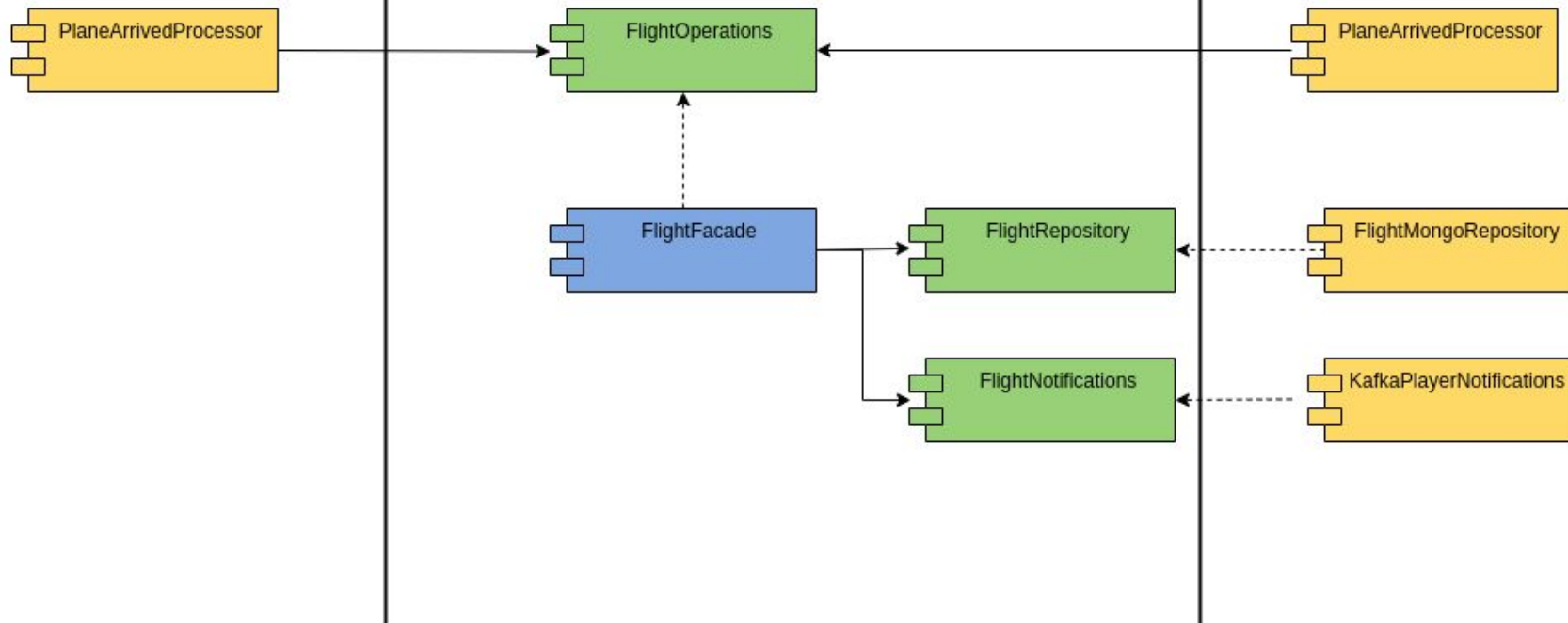
Testando um transformador de eventos



Adaptadores primários

Lógica de Negócio

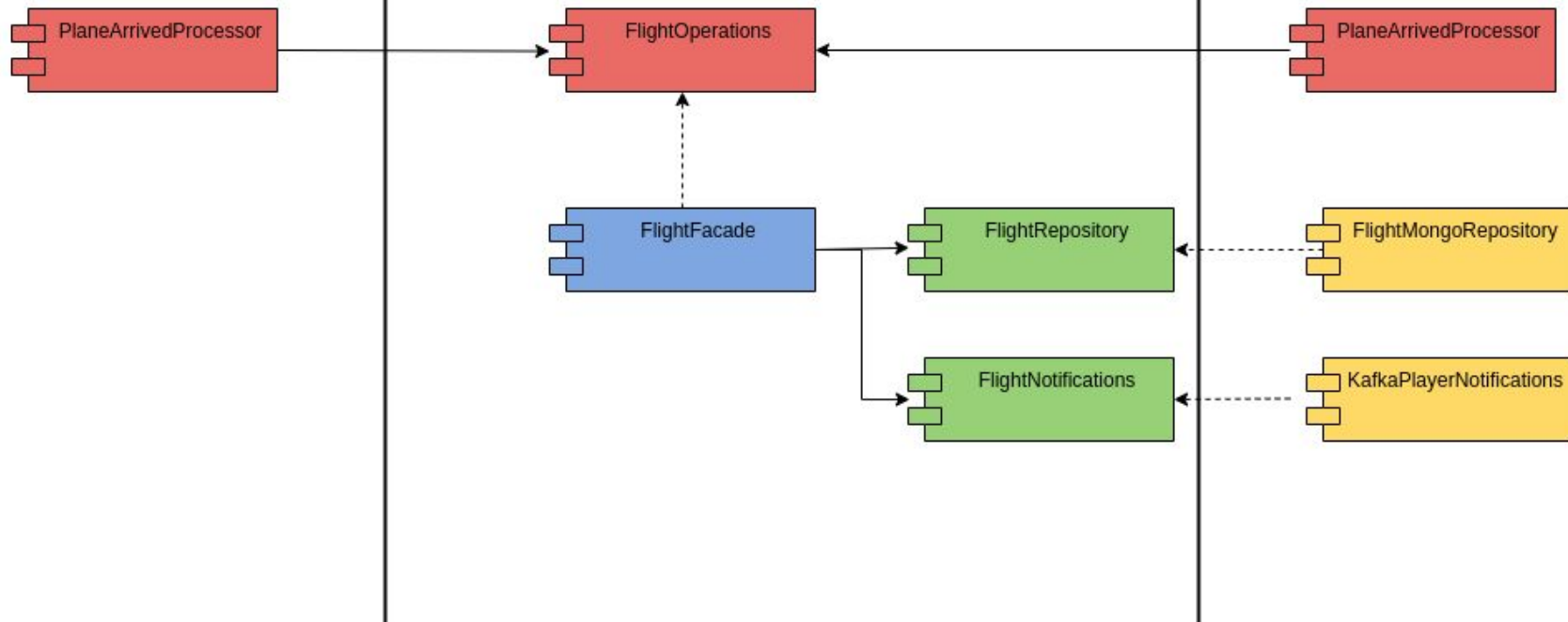
Adaptadores secundários



Adaptadores primários

Lógica de Negócio

Adaptadores secundários



Passos

1 - Levantar um Kafka em memória

2- Subir a aplicação apontando para o Kafka em memória

```
23 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE)
24 @EmbeddedKafkaWithTopics
25 class PlaneEventProcessorTest {
```

[flightapi/plane/infra/stream/PlaneEventProcessorTest](#)

Passos

1 - Levantar um Kafka em memória

2- Subir a aplicação apontando para o Kafka em memória

3- Preparar mock

Preparando Mock

```
42         when(flightOperations.findConfirmedFlightByPlaneId(PLANE_ID)).  
43             thenReturn(Optional.of(FLIGHT_WITH_ID));  
44
```

[flightapi/plane/infra/stream/PlaneEventProcessorTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Preparar mock
- 4- Envia mensagem**

Enviando a mensagem

```
47         String planeEvent = new JSONObject().put("planeId", PLANE_ID)
48             .put("currentAirport", CNH_CODE).toString();
49
50         kafkaTestUtils.sendMessage("plane-arrived-v1", planeEvent);
51     }
```

[flightapi/plane/infra/stream/PlaneEventProcessorTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Preparar mock
- 4- Envia mensagem
- 5- Valida que mensagem foi enviada para o destino**

Validação da mensagem

```
52     ConsumerRecord<String, String> record = kafkaTestUtils.getLastRecord(consumer, "flight-arrived-v1");
53
54     var jsonFlightEvent = new JSONObject(record.value());
55     assertThat(jsonFlightEvent.get("currentAirport")).isEqualTo(CNH_CODE);
56     assertThat(jsonFlightEvent.get("flightId")).isEqualTo(FLIGHT_ID);
57
58     assertThat(record.key()).isEqualTo(FLIGHT_ID);
59 }
```

[flightapi/plane/infra/stream/PlaneEventProcessorTest](#)

Testando um produtor de eventos

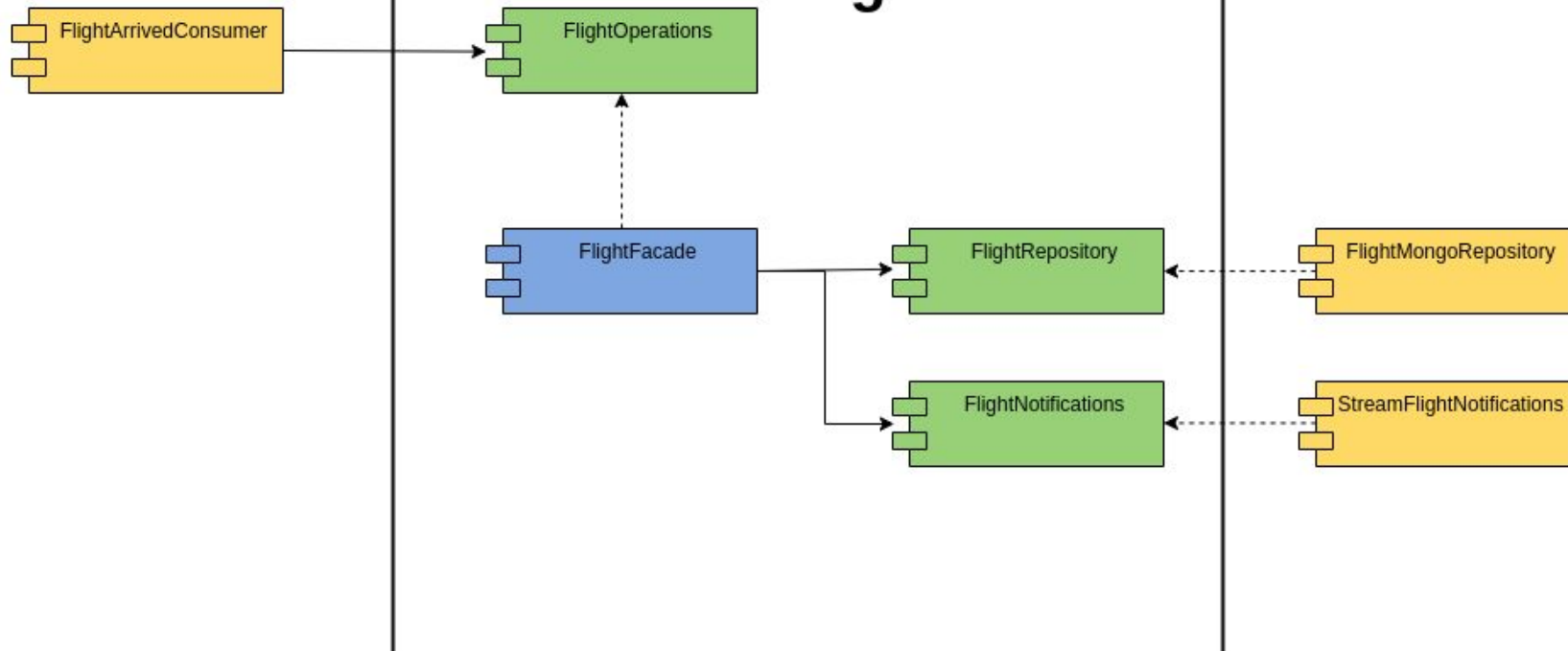


O evento apenas é enviado para flight-finished quando o voo aterrissa no destino

Adaptadores primários

Lógica de Negócio

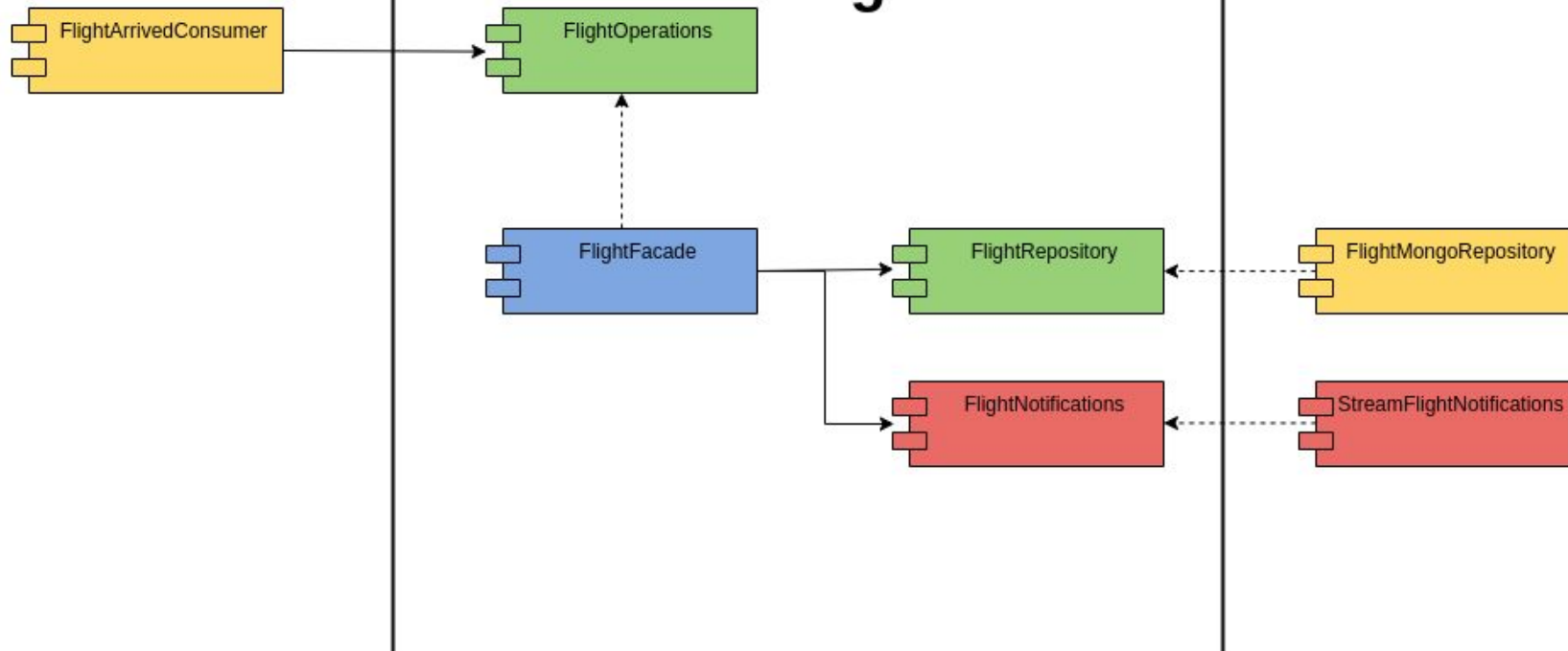
Adaptadores secundários



Adaptadores primários

Lógica de Negócio

Adaptadores secundários



Passos

1 - Levantar um Kafka em memória

2- Subir a aplicação apontando para o Kafka em memória

```
18 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE)
19 @EmbeddedKafkaWithTopics
20 class StreamFlightNotificationsTest {
```

[flightapi/flight/infra/stream/producer/StreamFlightNotificationsTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Injetar classe que faz o envio e chamar método**

Injetando e chamando o método

```
25      @Autowired
26      private StreamFlightNotifications notifications;
27
38
39      notifications.flightFinished(FLIGHT_ID);
40
```

[flightapi/flight/infra/stream/producer/StreamFlightNotificationsTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Injetar classe que faz o envio e chamar método
- 4- Valida que mensagem está no tópico**

Validando que mensagem está no tópico

```
41     ConsumerRecord<String, String> record = kafkaTestUtils.getLastRecord(consumer, "flight-finished-v1");
42
43     var jsonFlightEvent = new JSONObject(record.value());
44     assertThat(jsonFlightEvent.get("flightId")).isEqualTo(FLIGHT_ID);
45 }
```

[flightapi/flight/infra/stream/producer/StreamFlightNotificationsTest](#)

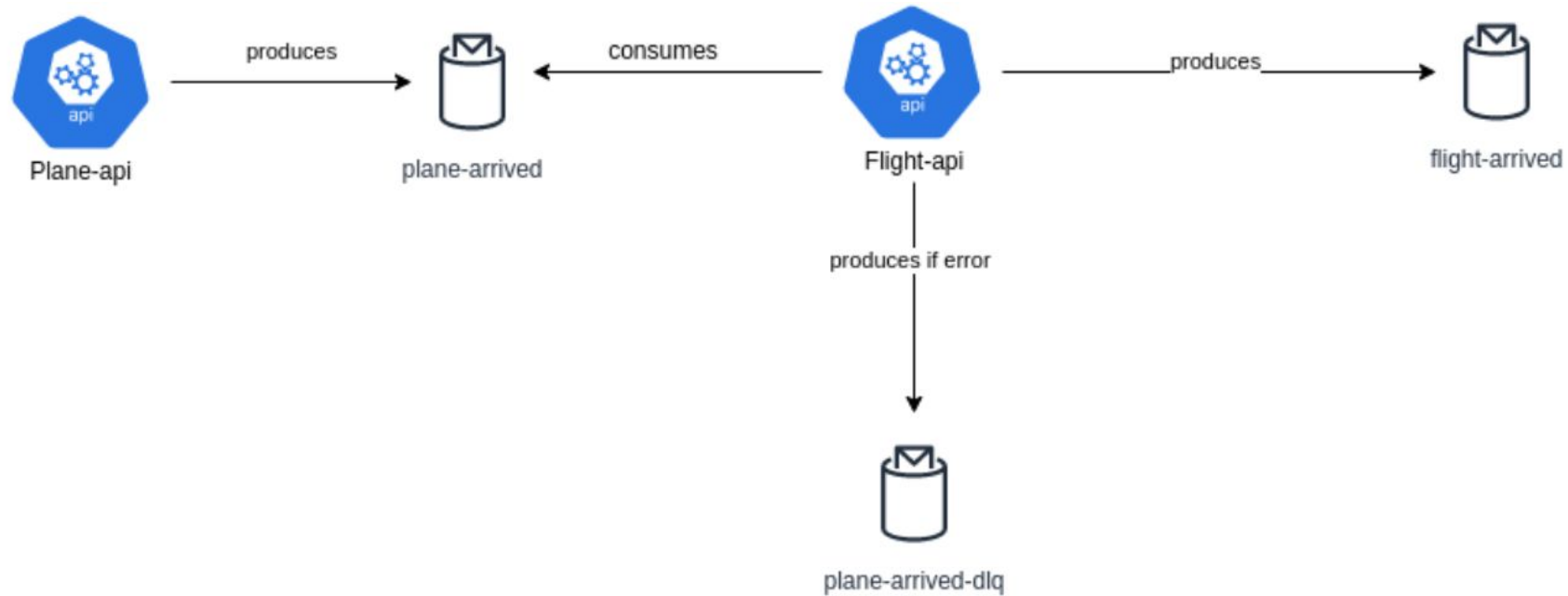
Testando envio para Dead Letter Queue

Dead Letters Queue (DLQ)

Quando a mensagem não é processada corretamente por algum problema, ela pode ser enviada para um tópico especial para posterior análise e reproprocessamento.

Retries

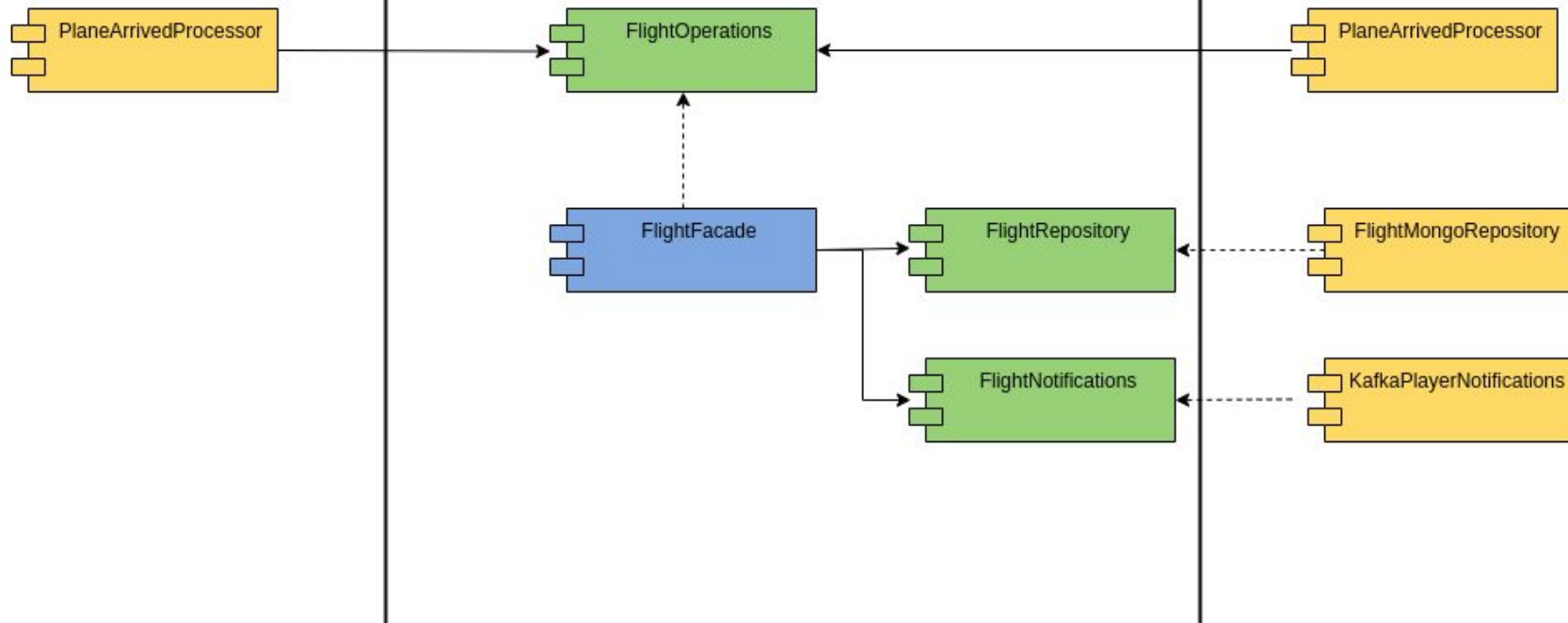
As mensagens também podem receber tentativas de processamento.



Adaptadores primários

Lógica de Negócio

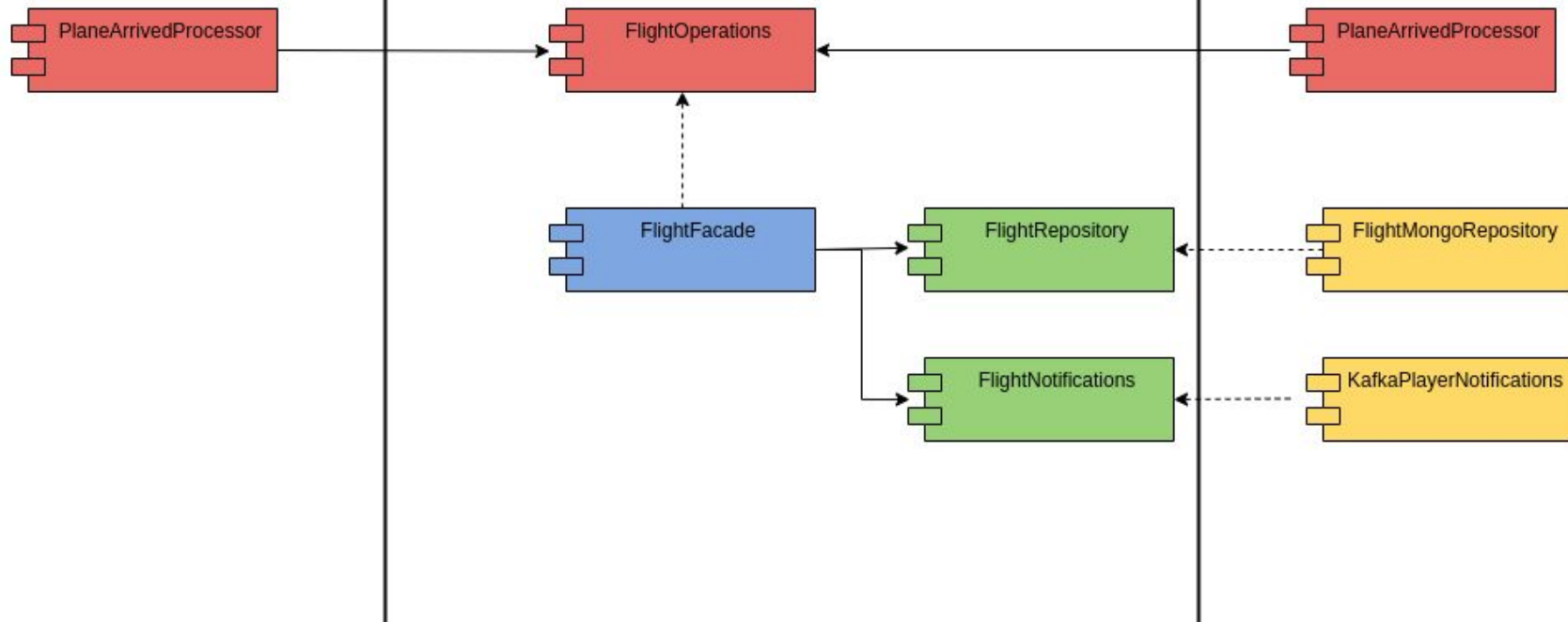
Adaptadores secundários



Adaptadores primários

Lógica de Negócio

Adaptadores secundários



Passos

1 - Levantar um Kafka em memória

2- Subir a aplicação apontando para o Kafka em memória

```
24 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE)
25 @EmbeddedKafkaWithTopics
26 class FlightArrivedConsumerTest {
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Preparar mock simulando o erro**

Preparando mock simulando um erro

```
58         doThrow(RuntimeException.class)
59             .when(flightOperations).flightArrivedIn(FLIGHT_ID, new Airport(AIRPORT));
60
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Preparar mock simulando o erro
- 4- Envia mensagem**

Enviando a mensagem

```
63         String flightEvent = new JSONObject().put("flightId", FLIGHT_ID)
64             .put("currentAirport", AIRPORT).toString();
65
66         kafkaTestUtils.sendMessage("flight-arrived-v1", flightEvent);
67
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Preparar mock simulando o erro
- 4- Envia mensagem
- 5- Valida que a mensagem chegou na DLQ**

Valida que a mensagem está na DLQ

```
68     ConsumerRecord<String, String> record = kafkaTestUtils.getLastRecord(consumer, "flight-arrived-dlq-v1");
69
70     var jsonFlightEvent = new JSONObject(record.value());
71     assertThat(jsonFlightEvent.get("currentAirport")).isEqualTo(AIRPORT);
72     assertThat(jsonFlightEvent.get("flightId")).isEqualTo(FLIGHT_ID);
73     String exceptionMessage = new String(record.headers().lastHeader("x-exception-message").value());
74     assertThat(exceptionMessage).contains("RuntimeException");
75
```

[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Valida que foram executadas três tentativas

```
76         verify(flightOperations, times(3)).flightArrivedIn(FLIGHT_ID,  
77             new Airport(AIRPORT));
```

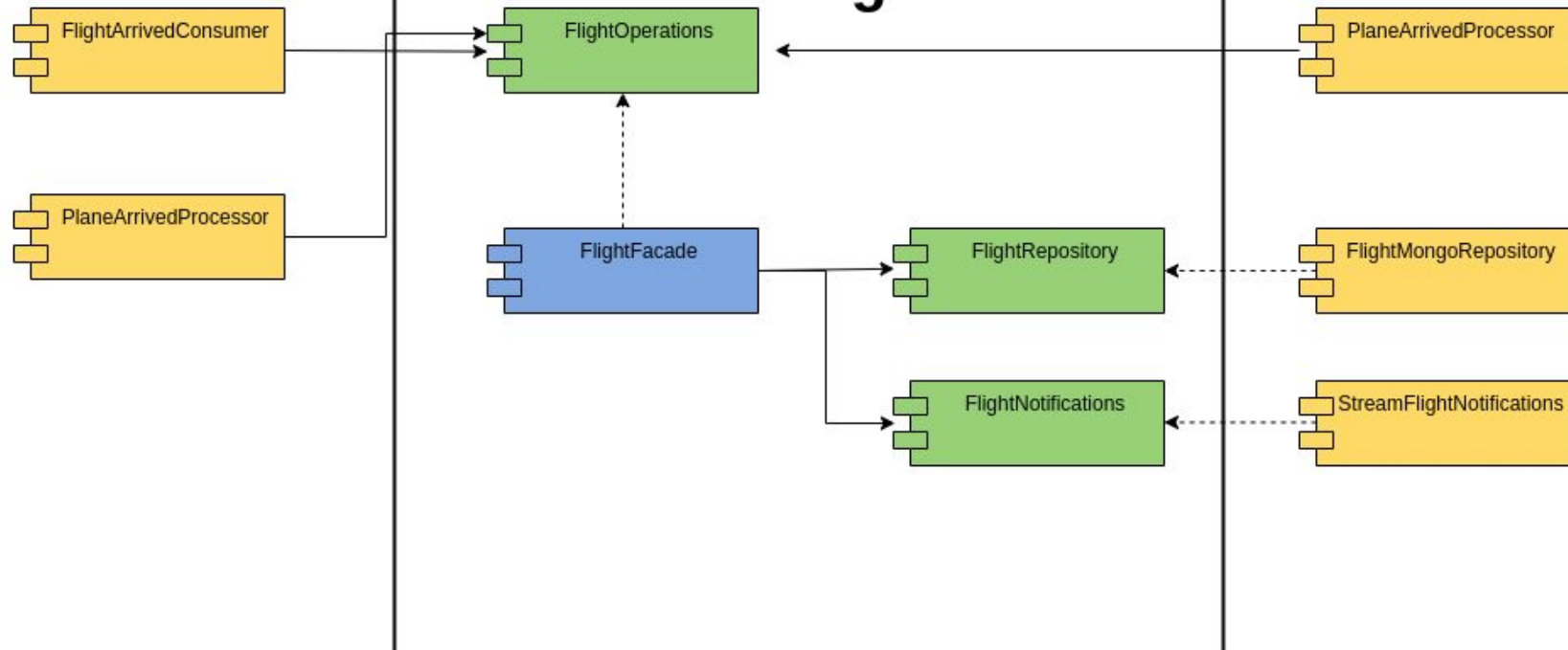
[flight/infra/stream/consumer/FlightArrivedConsumerTest](#)

Teste de aceitação

Adaptadores primários

Lógica de Negócio

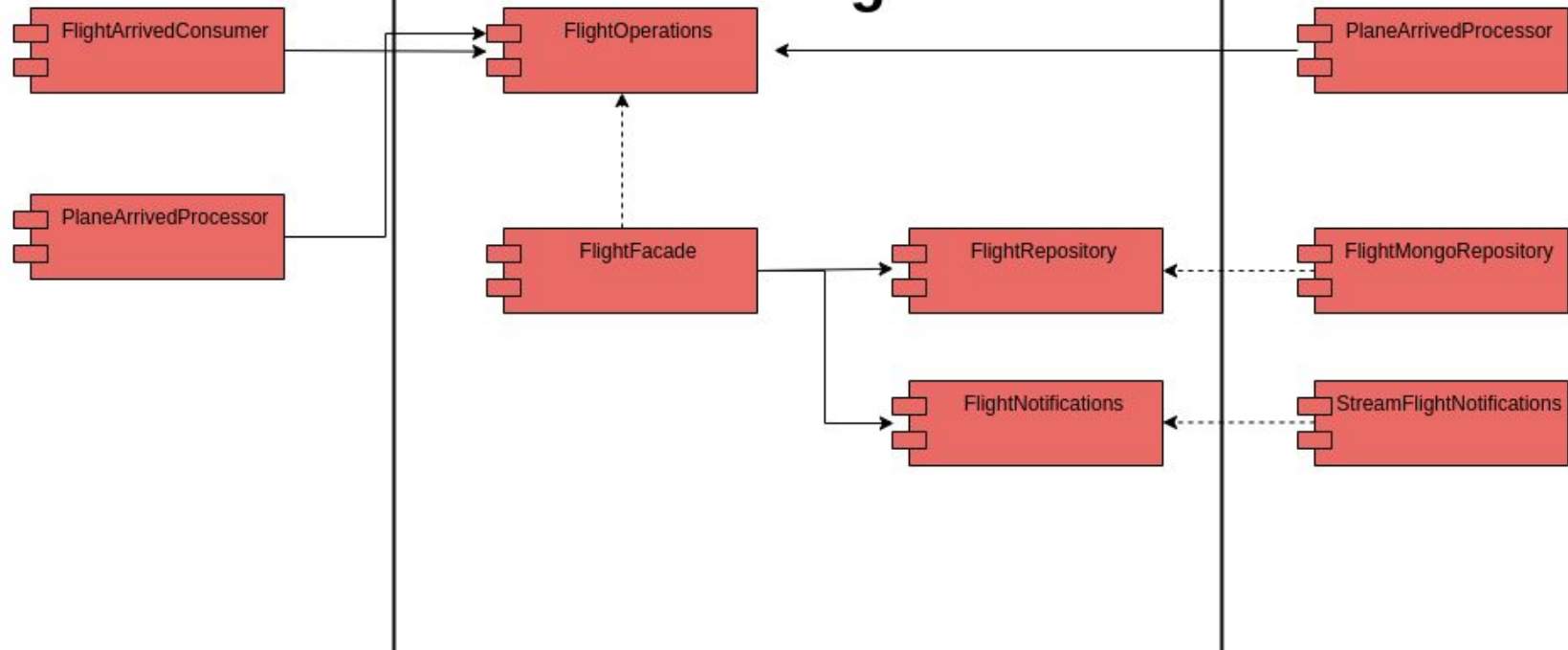
Adaptadores secundários



Adaptadores primários

Lógica de Negócio

Adaptadores secundários



Passos

1 - Levantar um Kafka em memória

2- Subir a aplicação apontando para o Kafka em memória

```
25  @SpringBootTest
26  @EmbeddedKafkaWithTopics
27  @AutoConfigureMockMvc
28  class FlightApiAcceptanceTest {
```

[flightapi/e2e/FlightApiAcceptanceTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Criar um vôo no banco de dados**

Criando um vôo

```
49     var jsonFlightEvent = new JSONObject();
50     jsonFlightEvent.put("planeId", PLANE_ID);
51     jsonFlightEvent.put("origin", POA_CODE);
52     jsonFlightEvent.put("destination", CNH_CODE);
53
54     String locationHeader = mockMvc.perform(post("/flights").contentType(MediaType.APPLICATION_JS
55                                     .content(jsonFlightEvent.toString()))
56                                     .andExpect(status().isCreated())
57                                     .andReturn().getResponse().getHeader("location"));
58
59     String flightId = locationHeader.substring(locationHeader.lastIndexOf("/") + 1);
60
```

[flightapi/e2e/FlightApiAcceptanceTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Criar um vôo no banco de dados
- 4- Enviar evento de “avião pousou”**

Enviando evento de avião pousou

```
61     var planeEvent = new JSONObject();
62     planeEvent.put("planeId", PLANE_ID);
63     planeEvent.put("currentAirport", CNH_CODE);
64
65     kafkaTestUtils.sendMessage("plane-arrived-v1", planeEvent.toString());
66
```

[flightapi/e2e/FlightApiAcceptanceTest](#)

Passos

- 1 - Levantar um Kafka em memória
- 2- Subir a aplicação apontando para o Kafka em memória
- 3- Criar um voo no banco de dados
- 4- Enviar evento de “avião pousou” no aeroporto destino
- 5- Validar que evento de “voo finalizado” foi enviado**

Validando evento de voo finalizado

```
67         ConsumerRecord<String, String> record = kafkaTestUtils
68             .getLastRecord(consumer, "flight-finished-v1");
69
70         JSONObject jsonFlightFinished = new JSONObject(record.value());
71
72         assertThat(jsonFlightFinished.get("flightId")).isEqualTo(flightId);
73     }
74 }
```

[flightapi/e2e/FlightApiAcceptanceTest](#)

Dúvidas ou comentários?



<https://www.linkedin.com/in/henriquelschmidt/>



<https://github.com/henriquels25>



<https://henriquels25.github.io/>