LAB 13: QUARKUS REACTIVE REVIEW

Autor: José Díaz

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

1. Abre el proyecto **10-reactive-review-start**
2. Agrega la dependencia requerida para crear un endpoint reactivo que guarda datos en una base de datos PostgreSQL, y envía eventos a Apache Kafka.
3. Configura la aplicación para usar cuatro canales.
    a. Un incoming channel que consume eventos **SpeakerWasCreated** del canal **new-speakers-in**. Usa el tópico kafka **speaker-was-created** para consumir eventos. Establece la propiedad **offset.reset** del incoming channel a **earliest**, y deserializa los mensajes entraste con la clase **com.bcp.training.serde.SpeakerWasCreatedDeserializer**.
    b. Un outgoing channel que publica eventos **SpeakerWasCreated** al canal de salida **new-speakers-out**. Usa el tópico **speaker-was-created** para publicar eventos.
    c. Un outgoing channel que publica eventos **EmployeeSignedUp** al canal de salida **employees-out**. Usa el tópico **employees-signed-up** para publicar eventos.
    d. Un outgoing channel que publica eventos **UpstreamMemberSignedUp** al canal de salida **upstream-members-out**. Usa el tópico **upstream-members-signed-up** para publicar eventos.

4. Crear un endpoint reactive POST con los siguientes requisitos:
    a. Recibir un objeto **Speaker** como payload.
    b. Guarda el payload en la base de datos usando una transacción.
    c. Envía un evento **SpeakerWasCreated** al channel **new-speakers-out**.
    d. Retorna una respuesta 201 HTTP que incluye en el response header location el URI del elemento insertado. El URI debe seguir el patrón **/speakers/{id}.**
5. Crea un procesador de eventos que consume y filtra eventos **SpeakerWasCreated**.
    a. Si la afiliación es RED_HAT, entonces envía un evento **EmployeeSignedUp** al channel **employees-out**.
    b. Si la afiliación es GNOME_FOUNDATION, entonces envía un evento **UpstreamMemberSignedUp** al channel **upstream-members-out**.
    c. Siempre hacer acknowledge del mensaje
6. Ejecutar los tests para validar los cambios en el código. Opcionalmente, puedes usar el Swagger UI para manualmente validar los cambios antes de ejecutar las pruebas.

Solución

La aplicación para este ejercicio mantiene los registros de los speakers en la base de datos y usa reactive messaging para publicar eventos respecto a las acciones ocurridas en la aplicación. La aplicación usa Dev Services para iniciar una base de datos PostgreSQL y una instancia de Apache Kafka.

1. Abre el proyecto **10-reactive-review-start**
2. Agrega las dependencias requeridas para crear un endpoint reactive que guarda datos en una base de datos PostgreSQL, y envía eventos a Apache Kafka.
    a. Retorna a la terminal de windows, y usa el comando maven para instalar **quarkus-messaging-kafka**, **quarkus-hibernate-reactive-panache** y **quarkus-reactive-pg-client**.
    b. **mvn quarkus:add-extensions -Dextensions="rest,quarkus-messaging-kafka,hibernate-reactive-panache,reactive-pg-client"**
3. Configurar la aplicación para usar 4 channels:
    a. Un incoming channel que consume eventos **SpeakerWasCreated** del canal **new-speakers-in**. Usa el tópico kafka **speaker-was-created** para consumir eventos. Establece la propiedad **offset.reset** del incoming channel a **earliest**, y deserializa los mensajes entraste con la clase **com.bcp.training.serde.SpeakerWasCreatedDeserializer**.
    b. Un outgoing channel que publica eventos **SpeakerWasCreated** al canal de salida **new-speakers-out**. Usa el tópico **speaker-was-created** para publicar eventos.
    c. Un outgoing channel que publica eventos **EmployeeSignedUp** al canal de salida **employees-out**. Usa el tópico **employees-signed-up** para publicar eventos.
    d. Un outgoing channel que publica eventos **UpstreamMemberSignedUp** al canal de salida **upstream-members-out**. Usa el tópico **upstream-members-signed-up** para publicar eventos.

    3.1 Abre el **src/main/resources/application.properties** y luego configurar el incoming channel **new-speakers-in**.
    - Establece el nombre del incoming channel a **new-speakers-in**.
    - Usar el topico kafka **speaker-was-created** para consumir eventos.
    - Establecer la propiedad **offset.reset** del incoming channel a **earliest**.
    - Deserializar los mensajes entrantes con la clase **com.bcp.training.serde.SpeakerWasCreatedDeserializer**.

```
# Incoming Channels
mp.messaging.incoming.new-speakers-in.connector = smallrye-kafka
mp.messaging.incoming.new-speakers-in.topic = speaker-was-created
mp.messaging.incoming.new-speakers-in.auto.offset.reset = earliest
mp.messaging.incoming.new-speakers-in.value.deserializer =
com.bcp.training.serde.SpeakerWasCreatedDeserializer
```

    3.2 Configurar el outgoing channel **new-speakers-out**.
    - Establecer el nombre del outgoing channel a **new-speakers-out**
    - Usar el tópico kakfa **speaker-was-created** para publicar eventos.
    - Serializar los mensajes outgoing con

io.quarkus.kafka.client.serialization.ObjectMapperSerializer.

```
# Outgoing Channels
mp.messaging.outgoing.new-speakers-out.connector = smallrye-kafka
mp.messaging.outgoing.new-speakers-out.topic = speaker-was-created
mp.messaging.outgoing.new-speakers-out.value.serializer =
io.quarkus.kafka.client.serialization.ObjectMapperSerializer
```

3.3 Configurar el outgoing channel **employees-out**.

- Establecer el nombre del outgoing channel a **employees-out**
- Usar el tópico kakfa **employees-signed-up** para publicar eventos.
- Serializar los mensajes outgoing con **io.quarkus.kafka.client.serialization.ObjectMapperSerializer**.

```
mp.messaging.outgoing.employees-out.connector = smallrye-kafka
mp.messaging.outgoing.employees-out.topic = employees-signed-up
mp.messaging.outgoing.employees-out.value.serializer =
io.quarkus.kafka.client.serialization.ObjectMapperSerializer
```

3.4 Configurar el outgoing channel **upstream-members-out**.

- Establecer el nombre del outgoing channel a **upstream-members-out**
- Usar el tópico kakfa **upstream-members-signed-up** para publicar eventos.
- Serializar los mensajes outgoing con **io.quarkus.kafka.client.serialization.ObjectMapperSerializer**.

```
mp.messaging.outgoing.upstream-members-out.connector = smallrye-kafka
mp.messaging.outgoing.upstream-members-out.topic = upstream-members-signed-up
mp.messaging.outgoing.upstream-members-out.value.serializer =
io.quarkus.kafka.client.serialization.ObjectMapperSerializer
```

4. Crea un endpoint reactivo POST con los siguientes requisitos:

- Recibe un objeto **Speaker** como payload.
- Guarda el payload en la base de datos usando una transacción.
- Envía un evento **SpeakerWasCreated** al channel **new-speakers-out**.
- Retorna una respuesta HTTP 201 que incluye en el header de la respuesta el URI del elemento insertado. El URI debe seguir el patrón **/speakers/{id}.**

4.1. Abre la clase **SpeakerResource**, y luego agrega una variable Emitter para enviar eventos **SpeakerWasCreated** al channel **new-speakers-out**.

```
@Path("/speakers")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
```

```
public class SpeakerResource {

    @Channel("new-speakers-out")
    Emitter<SpeakerWasCreated> emitter;
```

4.2. Crea un endpoint POST que recibe instancias **Speaker** y retorna una instancia **Uni<Response>.**

```
@Path("/speakers")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class SpeakerResource {

    @Channel("new-speakers-out")
    Emitter<SpeakerWasCreated> emitter;

    @POST
    public Uni<Response> create(Speaker newSpeaker) {

    }
```

4.3. Crea la lógica del endpoint. Persiste la instancia **Speaker** en la transacción. En el item insertado, usa el emitter para enviar un evento **SpeakerWasCreated**, y luego retorna una respuesta created que incluye un URI del elemento insertado.

```
@POST
    public Uni<Response> create(Speaker newSpeaker) {
        return Panache
                .<Speaker>withTransaction(newSpeaker::persist)
                .onItem()
                .transform(
                        inserted -> {
                            emitter.send(
                                    new SpeakerWasCreated(
                                            inserted.id,
                                            newSpeaker.fullName,
                                            newSpeaker.affiliation,
                                            newSpeaker.email
                                    )
                            );

                            return Response.created(
                                    URI.create("/speakers/" + inserted.id)
                            ).build();
                        }
                );
    }
```

5. Crea un event processor que consume y filtra eventos **SpeakerWasCreated**.

   - Si la afiliación es RED_HAT, entonces envía un evento **EmployeeSignedUp** al channel **employees-out**.
   - Si la afiliación es GNOME_FOUNDATION, entonces envía un evento **UpstreamMemberSignedUp** al channel **upstream-members-out**.
   - Siempre hacer acknowledge del mensaje.

   5.1. Crear un deserializer que transforma mensajes de eventos de Apache Kafka a instancias **SpeakerWasCreated**.
   - Llama a la clase **SpeakerWasCreatedDeserializer**
   - Crea la entidad en el paquete **com.bcp.training.serde**.

```
package com.bcp.training.serde;

import com.bcp.training.event.SpeakerWasCreated;
import io.quarkus.kafka.client.serialization.ObjectMapperDeserializer;

public class SpeakerWasCreatedDeserializer
        extends ObjectMapperDeserializer<SpeakerWasCreated> {
    public  SpeakerWasCreatedDeserializer() {
        super(SpeakerWasCreated.class);
    }
}
```

   5.2. Abre la clase **com.bcp.training.reactive.NewSpeakersProcessor** y agrega 2 emitters.
   - Un emitter denominado **employeeEmitter** para enviar eventos **EmployeeSignedUp** al channel **employees-out**.
   - Un emitter denominado **upstreamEmitter** para enviar eventos **UpstreamMemberSignedUp** al channel **upstream-members-out**.

```
@ApplicationScoped
public class NewSpeakersProcessor {
    private static final Logger LOGGER =
Logger.getLogger(NewSpeakersProcessor.class);

    @Channel("employees-out")
    Emitter<EmployeeSignedUp> employeeEmitter;

    @Channel("upstream-members-out")
    Emitter<UpstreamMemberSignedUp> upstreamEmitter;
```

   5.3. Agrega un método llamado **sendEventNotifications** que procesa mensajes **SpeakerWasCreated** y retorna un valor **CompletionStage<Void>.**
   - Establece el incoming channel **new-speakers-in**.
   - Si la afiliación del speaker del evento incoming es RED_HAT, entonces envía un evento **EmployeeSignedUp** al channel **employees-out**.
   - Si la afiliación del speaker del evento incoming es GNOME_FOUNDATION, entonces envía un evento **UpstreamMemberSignedUp** al channel **upstream-**

members-out.

- Acknowledge los eventos
- Usa los métodos **logEmitEvent**() y **logProcessEvent**() para depurar la lógica.

```
@Incoming("new-speakers-in")
    public CompletionStage<Void>
            sendEventNotifications(Message<SpeakerWasCreated> message){
        SpeakerWasCreated event = message.getPayload();
        logProcessEvent(event.id);

        if(event.affiliation==Affiliation.RED_HAT){
            logEmitEvent("EmployeeSignedUp", event.affiliation);
            employeeEmitter.send(
                    new EmployeeSignedUp(event.id, event.fullName, event.email));
        }else if(event.affiliation ==Affiliation.GNOME_FOUNDATION){
            logEmitEvent("UpstreamMemberSignedUp", event.affiliation);
            upstreamEmitter.send(new
                    UpstreamMemberSignedUp(event.id, event.fullName, event.email));
        }

        return message.ack();
    }
```

6. Ejecutar los tests para validar los cambios en el código. Opcionalmente puedes usar el Swagger UI para manualmente validar los cambios antes de ejecutar los tests.
   6.1. Retorna a la terminal de windows y usa el siguiente comando maven para ejecutar las pruebas: **mvn clean test**.

```
[student@workstation reactive-review]$ mvn clean test
...output omitted...
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
...output omitted...
```

Enjoy!

José