LAB 20: QUARKUS TOLERANCE REVIEW

Autor: José Díaz

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

Abre el proyecto **17-tolerance-review-start**

Instrucciones

Este laboratorio usa dos servicios:

session
Un servicio que mantiene una lista de sesiones de speaking. Este contiene un cache local de speakers en cada sesión. Adicionalmente, este enrique la información de cada speaker utilizando el servicio speaker.

Los speakers contienen información como first name y surname. Los speakers cacheados solo contienen el first name.

speaker
Un servicio que mantiene un registro completo de speakers.
Tu puedes usar este servicio para opcionalmente probar el servicio session.

Para completar este laboratorio hay que asegurarnos que el servicio session pase las pruebas.

1. Agrega los liveness y readiness probes al microservicio session.
   Retorna las siguientes respuestas:
   - Liveness probe: Service is alive
   - Readiness probe: Service is ready

2. El endpoint **GET /sessions** del servicio session llama al servicio speaker para enriquecer la data del speaker.
   Implementa el método **SessionResource#allSessionsFallback** para usar el método **SessionStore#findAllWithoutEnrichment** para retornar las sessions sin enviar requests al servicio speaker.
   Luego configura el endpoint para responder sin enviar requests al servicio speaker cuando el servicio speaker no este disponible.
3. El endpoint **PUT /sessions/{sessionId}/speakers/{speakerName}** debe completarse.
   Implementa una política de reintento para solicitar una vez por segundo por 60 segundos en caso de una excepción **InternalServerErrorException**.
4. El endpoint **GET /session/{sessionId}** del servicio session usa el servicio speaker para enriquecer la data de los speakers.
   Implementa el método **SessionResource#retrieveSessionFallback** para usar el **SessionStore#findByIdWithoutEnrichment** para retornar un objeto Response que contenga la session sin enviar requests al servicio speakers.
   Luego configura el endpoint para responder sin enviar requests al servicio speaker

cuando el servicio speaker no esta disponible.
Adicionalmente, cuando dos requests consecutivos al metodo **retrieveSession** fallan, retornar respuestas fallback en los siguientes 30 segundos.

5. El endpoint **GET /sessions/{sessionId}/speakers** debe responder en no mas de un Segundo. El endpoint usa el método **findSessionSpeakers** que llama al servicio speaker.
   Configura el metodo para arrojar una excepción si el servicio speaker toma mas de un segundo en responder.

SOLUCION

1. Agrega los liveness y readiness probes del microservicio session.

   a. Abre el archivo **src/main/java/com/bcp/training/conference/session/LivenessCheck.java**, e implementa la interface **HealthCheck**. Agrega la anotación **@Liveness**.

```
@Liveness
@ApplicationScoped
public class LivenessCheck implements HealthCheck
{
    @Override
    public HealthCheckResponse call() {
        return HealthCheckResponse.up("Service is
alive");
    }
}
```

   b. Abre el archivo **src/main/java/com/bcp/training/conference/session/ReadinessCheck.java** e implementa la interface **HealthCheck**. Agrega la anotación **@Readiness**.

```
@Readiness
@ApplicationScoped
public class ReadinessCheck implements HealthCheck
{
    @Override
    public HealthCheckResponse call() {
        return HealthCheckResponse.up("Service is
ready");
    }
}
```

   c. Verifica que el **testLivenessProbe** y **testReadinessProbe** pasan.

mvn clean test

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testLivenessProbe,SessionResourceTest#testReadinessProbe
...output omitted...
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

2. El endpoint **GET /sessions** del servicio **session** llama al servicio **speaker** para obtener data adicional.
   Implementar el método SessionResource#allSessionsFallback para usar el método **SessionStore#findAllWithoutEnrichment** para retornar las sessions sin enviar requests al servicio speaker.
   Luego configura el endpoint para responder sin enviar requests al servicio **speaker** cuando el servicio **speaker** no esta disponible.
   a. Abre el archivo **src/main/java/com/bcp/training/conference/session/SessionResource.java**. Luego implementa el método **allSessionsFallback**.

```
public Collection<Session> allSessionsFallback()
throws Exception {
    logger.warn("Fallback for GET /sessions");
    return
sessionStore.findAllWithoutEnrichment();
}
```

   b. Usa la anotación **@Fallback** para configurar que el método **allSessions** use el método **allSessionsFallback** durante las fallas.

```
@GET
@Fallback(fallbackMethod="allSessionsFallback")
public Collection<Session> allSessions() throws
Exception {
    return sessionStore.findAll();
}
```

   c. Verifica que el test **testAllSessionsFallback** pase.

   mvn clean test -Dtest=SessionResourceTest#testAllSessionsFallback

```
[student@workstation session]$ mvn clean test \
  -Dtest=SessionResourceTest#testAllSessionsFallback
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

3. El endpoint **PUT /sessions/{sessionId}/speakers/{speakerName}** debe completarse.
   Implementa una política de reintento para reintentar el request una vez por segundo por 60 segundos en caso de la excepcion **InternalServerErrorException**.

a. Agrega la anotación **@Retry** para ambos método endpoint. Usa el **maxRetries** y **delay** para configurar un reintento por segundo por 60 segundos.

```
@PUT
@Path("/{sessionId}/speakers/{speakerName}")
@Transactional
@Retry(maxRetries=60, delay=1_000,
retryOn=InternalServerErrorException.class)
public Response
addSessionSpeaker(@PathParam("sessionId")
final String sessionId,

@PathParam("speakerName") final String
speakerName) {
    final Optional<Session> result =
sessionStore.findByIdWithoutEnrichmentMaybeFai
l(sessionId);

    if (result.isPresent()) {
        final Session session = result.get();

sessionStore.addSpeakerToSession(speakerName,
session);
        return Response.ok(session).build();
    }

    throw new NotFoundException();
}
```

b. Verifica que el **testAddSpeakerToSession** pase.

mvn clean test -Dtest=SessionResourceTest#testAddSpeakerToSession

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testAddSpeakerToSession
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

4. El endpoint **GET /session/{sessionId}** del servicio session usa al servicio **speaker** para enriquecer la data del speaker.
Implementa el método **SessionResource#retrieveSessionFallback** para usar el método **SessionStore#findByIdWithoutEnrichment** para retornar un objeto Response que contenga la session sin enviar requests al servicio speaker.
Luego configura el endpoint para responder sin enviar requests al servicio Speaker cuando el servicio speaker no esta disponible.
Adicionalmente, cuando dos requests consecutivos al metodo **retrieveSession** fallen, retorna una respuesta fallback por los siguientes 30 segundos.

a. Implementa el método **retrieveSessionFallback**.

```
public Response retrieveSessionFallback(final
String sessionId) {
    logger.warn("Fallback for GET
/sessions/"+sessionId);
    return
sessionStore.findByIdWithoutEnrichment(sessionId)
            .map(s -> Response.ok(s).build())
            .orElseThrow(NotFoundException::new);
}
```

b. Usa la anotación **@Fallback** para configurar el método **retrieveSession** y usar el método **allSessionsFallback** durante las fallas.
Adicionalmente, usa la anotación **@CircuitBreaker** para usar el método fallback despues de 2 fallas.

```
@GET
@Path("/{sessionId}")
@Fallback(fallbackMethod="retrieveSessionFal
lback")
@CircuitBreaker(requestVolumeThreshold = 2,
failureRatio = 1, delay = 30_000)
public Response
retrieveSession(@PathParam("sessionId")
final String sessionId) {
    final Optional<Session> result =
sessionStore.findById(sessionId);

    return result.map(s ->
Response.ok(s).build()).orElseThrow(NotFound
Exception::new);
}
```

c. Verifica que **testSessionCircuitBreaker** pase.

mvn clean test -Dtest=SessionResourceTest#testSessionCircuitBreaker

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testSessionCircuitBreaker
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

5. El endpoint **GET /sessions/{sessionId}/speakers** debe responder en no mas de un Segundo. El endpoint usa el método **findSessionSpeakers** que llama al servicio speaker.
   Configura el método para arrojar una excepción si el servicio speaker toma mas de un segundo en responder.
   a. Usa la anotación **@Timeout** en el método **findSessionSpeakers**. Usa un valor de parametro de 1000 ms.

```
@Timeout(1000)
public Optional<Session>
findSessionSpeakers(String sessionId) {
    return sessionStore.findById(sessionId);
}
```

   b. Verifica que el test **testSessionsSpeakerFallback** pase.

mvn clean test -Dtest=SessionResourceTest#testSessionSpeakerFallback

```
[student@workstation session]$ mvn clean test \
-Dtest=SessionResourceTest#testSessionSpeakerFallback
...output omitted...
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

Esto termina el laboratorio.

Enjoy!

José