LAB 19: QUARKUS HEALTH

Autor: José Díaz

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

Abre el proyecto **16-tolerance-health-start**.

1. Abre el Proyecto con tu editor favorito.
2. Revisa los siguientes archivos:
   a. El **com.bcp.training.service.StateService** es un bean que controla si la aplicación esta viva.
   b. El **com.bcp.training.SolverResource** es una clase que expone un endpoint REST que soluciona ecuaciones matemáticas.

3. Incluye las extensiones Quarkus requeridas para integrar health checks en la aplicación, y desplegar la aplicación a K8S.

   a. Retorna a la terminal windows, y luego usa el comando maven para instalar la extensión **quarkus-smallrye-health**.

   mvn quarkus:add-extension -Dextensions=smallrye-health

4. Crea un liveness health check endpoint.

   a. Abre la clase **LivenessHealthResource**.
   b. Anota la clase con la anotación **@Liveness** e implementa la interface **HealthCheck**. Sobreescribe el método **call()** usando **StateService** para determinar si la aplicación esta viva (**up**) o no (**down**).

```
@Liveness
@ApplicationScoped
public class LivenessHealthResource implements
HealthCheck {

    private final String HEALTH_CHECK_NAME =
"Liveness";

    @Inject
    StateService applicationState;
```

```
    @Override
    public HealthCheckResponse call() {
        return applicationState.isAlive()
                ?
HealthCheckResponse.up(HEALTH_CHECK_NAME)
                :
HealthCheckResponse.down(HEALTH_CHECK_NAME);
    }
}
```

5. Crea un readiness health check endpoint.

    a. Abre la clase **ReadinessHealthResource**.
    b. Anota la clase con la anotación **@Readiness**, implementa la interface **HealthCheck** sobreescribiendo el método **call()**. Las primeras 10 llamadas del endpoint readiness debe retornar una respuesta **down** health check.

```
@Readiness
@ApplicationScoped
public class ReadinessHealthResource implements
HealthCheck {

    private final String HEALTH_CHECK_NAME =
"Readiness";

    private int counter = 0;

    @Override
    public HealthCheckResponse call() {
        return ++counter >= 10
                ?
HealthCheckResponse.up(HEALTH_CHECK_NAME)
                :
HealthCheckResponse.down(HEALTH_CHECK_NAME);
    }
}
```

6. Verificar la implementación de los liveness y readiness health checks. Los primeros 10 requests a el endpoint health deben retornar el estatus **DOWN**.

    a. Retorna a la terminal de windows, y luego usa el comando mvn quarkus:dev para iniciar la aplicación en modo desarrollo.

    mvn quarkus:dev

b. Abre una nueva terminal, y luego usa la combinación de **watch** y **curl** para verificar que el endpoint **/q/health** retorna **DOWN** como el status actual de la aplicación.

watch -d -n 2 curl -s http://localhost:8080/q/health

```
[student@workstation ~]$ watch -d -n 2 curl -s http://localhost:8080/q/health
{
    "status": "DOWN",
    "checks": [
        {
            "name": "Liveness",
            "status": "UP"
        },
        {
            "name": "Readiness",
            "status": "DOWN"
        }
    ]
}
```

Espera hasta que el contador del readiness llegue al limite especifico en la logica de la aplicación, y reporte **UP**.

c. Abre una nueva terminal y usa el comando curl para llamar al endpoint **/crash**.

```
[student@workstation ~]$ curl http://localhost:8080/crash
Service not alive
```

d. Cierra la terminal de windows.
e. Reejecuta el comando **watch** y verifica la respuesta a los health checks. El status de los liveness checks deben ser **DOWN**.

```
{
    "status": "DOWN",
    "checks": [
        {
            "name": "Liveness",
            "status": "DOWN"
        },
        {
            "name": "Readiness",
            "status": "UP"
        }
    ]
}
```

f. Deten el comando **watch** presionando **CTRL+C**, y luego cierra la terminal de windows.

Esto termina el laboratorio.

Enjoy!

José