LAB 19: QUARKUS HEALTH

Autor: José Díaz

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

Abre el proyecto **tolerance-health**.

Instructions

1. Open the Quarkus project located at `~/DO378/tolerance-health` with an editor, such as VSCodium or vim.

    1.1. In a terminal window, navigate to the project directory.

```
[student@workstation ~]$ cd ~/DO378/tolerance-health
```

    1.2. Open the project with your editor and examine the files.

    - The `com.redhat.training.service.StateService` is a bean that controls whether the application is alive.
    - The `com.redhat.training.SolverResource` class exposes a REST endpoint that solves math equations.

2. Include the Quarkus extensions required to integrate health checks in the application, and to deploy the application to RHOCP.

    2.1. Return to the terminal window, and then use the Maven command to install the `quarkus-smallrye-health` and `quarkus-openshift` extensions.

```
[student@workstation tolerance-health]$ mvn quarkus:add-extension \
-Dextensions="smallrye-health,openshift"
...output omitted...
[INFO] [SUCCESS] ...  Extension io.quarkus:quarkus-openshift has been installed
[INFO] [SUCCESS] ...  Extension io.quarkus:quarkus-smallrye-health has been
 installed
...output omitted...
```

3. Create a liveness health check endpoint.

   3.1. Open the `LivenessHealthResource` class.

   3.2. Annotate the class with the `@Liveness` annotation.

```
package com.redhat.training;

...code omitted...

@Liveness
@ApplicationScoped
public class LivenessHealthResource {

    private final String HEALTH_CHECK_NAME = "Liveness";

    @Inject
    StateService applicationState;
}
```

   3.3. Implement the `HealthCheck` interface.

```
package com.redhat.training;

...code omitted...

@Liveness
@ApplicationScoped
public class LivenessHealthResource implements HealthCheck {

    private final String HEALTH_CHECK_NAME = "Liveness";

    @Inject
    StateService applicationState;
}
```

   3.4. Implement the liveness logic of the application by overriding the `call()` method. Use the `StateService` class to determine whether the application is up or down.

```
package com.redhat.training;

...code omitted...

@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);
    }
}
```

4. Create a readiness health check endpoint.

4.1. Open the `ReadinessHealthResource` class.

4.2. Annotate the class with the `@Readiness` annotation.

```
package com.redhat.training;

...code omitted...

@Readiness
@ApplicationScoped
public class ReadinessHealthResource {

    private final String HEALTH_CHECK_NAME = "Readiness";
}
```

4.3. Implement the `HealthCheck` interface.

```
package com.redhat.training;

...code omitted...

@Readiness
@ApplicationScoped
public class ReadinessHealthResource implements HealthCheck {

    private final String HEALTH_CHECK_NAME = "Readiness";
}
```

4.4. Implement the readiness logic of the application by overriding the `call()` method. The first ten calls to the readiness endpoint must return a down health check response.

```
package com.redhat.training;

...code omitted...
```

```
@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);
    }
}
```

5. Verify the implementation of the liveness and readiness health checks. The first ten requests to the health endpoint must return that the readiness status is `DOWN`.

5.1. Return to the terminal window, and then use the command `mvn quarkus:dev` to start the application in development mode.

```
[student@workstation tolerance-health]$ mvn quarkus:dev
...output omitted...
... INFO [io.quarkus] ... Listening on: http://localhost:8080
...output omitted...
```

5.2. Open a new a terminal, and then use the combination of `watch` and `curl` commands to verify that the `/q/health` endpoint returns `DOWN` as the current status of the application.

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

Wait until the readiness count reaches the limit specified in the application logic, and the readiness health check reports `UP`.

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

5.3. Open a new terminal window, and then use the `curl` command to call the `/crash` endpoint.

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

5.4. Close the terminal window.

5.5. Return to the terminal window where the `watch` command is running, and verify the response of the health checks. The status of the liveness health check must be `DOWN`.

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

5.6. Stop the `watch` command by pressing `Ctrl+C`, and then close the terminal window.

6. Deploy the application to RHOCP.

6.1. Return to the terminal window where the Quarkus application is running and stop the application by pressing **q**.

6.2. Log in to RHOCP with your developer user.

```
[student@workstation tolerance-health]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful
...output omitted...
```

6.3. Verify that you are using the `tolerance-health` project.

```
[student@workstation tolerance-health]$ oc project tolerance-health
Already on project "tolerance-health" on server "https://
api.ocp4.example.com:6443".
```

6.4. Deploy the application to RHOCP by using the Maven command. Use the -`DskipTests` flag to skip the tests.

```
[student@workstation {gls_lab_script}]$ mvn clean package \
-Dquarkus.kubernetes.deploy=true \
-Dquarkus.openshift.expose=true \
-DskipTests
...output omitted...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...output omitted...
```

6.5. Monitor the deployment by using the `oc get pods` command. Use the `-w` option to continuously display the status changes. Wait until the number of ready pods is `1/1`.

```
[student@workstation tolerance-health]$ oc get pods -w
NAME                             READY  STATUS    RESTARTS  AGE
...output omitted...
quarkus-calculator-1-POD_SUFFIX  0/1    Running   0         7s
quarkus-calculator-1-POD_SUFFIX  1/1    Running   0         22s
...output omitted...
```

Copy the pod name for later use, and then stop the command by pressing `Ctrl+C`.

6.6. Verify the configuration of the liveness and readiness endpoints in the pod by using the `oc describe pod` command.

```
[student@workstation tolerance-health]$ oc describe \
pod quarkus-calculator-1-POD_SUFFIX
Name:          quarkus-calculator-1-POD_SUFFIX
Namespace:     tolerance-health
...output omitted...
    Liveness:       http-get http://:8080/q/health/live ...
    Readiness:      http-get http://:8080/q/health/ready ...
...output omitted...
```

6.7. Use the `curl` command to call the `/crash` endpoint in the deployed application. The application must report the status of the liveness health check as `DOWN` and RHOCP restarts the pod.

```
[student@workstation tolerance-health]$ curl \
http://quarkus-calculator-tolerance-health.apps.ocp4.example.com/crash
Service not alive
```

6.8. Use the `oc get pods` command to monitor the restarting of the application pod. The RESTARTS value for the application pod must be greater than zero.

⅄

```
[student@workstation tolerance-health]$ oc get pods -w
NAME                                READY   STATUS      RESTARTS       AGE
quarkus-calculator-1-build          0/1     Completed   0              5m14s
quarkus-calculator-1-deploy         0/1     Completed   0              4m49s
quarkus-calculator-1-POD_SUFFIX     0/1     Running     1 (4s ago)     4m46s
quarkus-calculator-1-POD_SUFFIX     1/1     Running     1 (16s ago)    4m58s
```

Stop the command by pressing `Ctrl+C`.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish tolerance-health
```

This concludes the section.

enjoy!

Jose