
Quarkus Health

quarkus-smallrye-health

```
[user@host myapp]$ mvn quarkus:add-extension \
-Dextensions=smallrye-health
```

Health Checks

In the MicroProfile Health specification, a *health check* is a condition that verifies the health status of an application. The most commonly used health checks are *liveness* and *readiness* checks.

Liveness checks

Liveness checks verify whether an application is healthy. An application is typically considered healthy if it works as expected, without critical or fatal errors. An example of an unhealthy scenario is an application that fails due to memory errors.

Readiness checks

Readiness checks verify whether a service is ready to serve requests. This type of health check usually considers the external dependencies required for the application to work properly. For example, for applications with a database, a readiness check should typically wait for the database to become available.

The MicroProfile Health specification uses annotations to identify and resolve health checks, such as `@Liveness` and `@Readiness`. You, as a developer, are responsible for implementing beans that contain your specific readiness and liveness health conditions, and annotate such beans with the pertinent annotations.

Health Endpoints

/q/health/ready

/q/health/live

/q/health/started

/q/health

```
{
  "status": "UP",
  "checks": [
      {
          "name": "Cache readiness check",
          "status": "UP"
      },
      {
          "name": "Database readiness check",
          "status": "UP"
      },
      {
          "name": "App liveness check",
          "status": "UP"
      },
  ]
}
```

/q/health-ui/

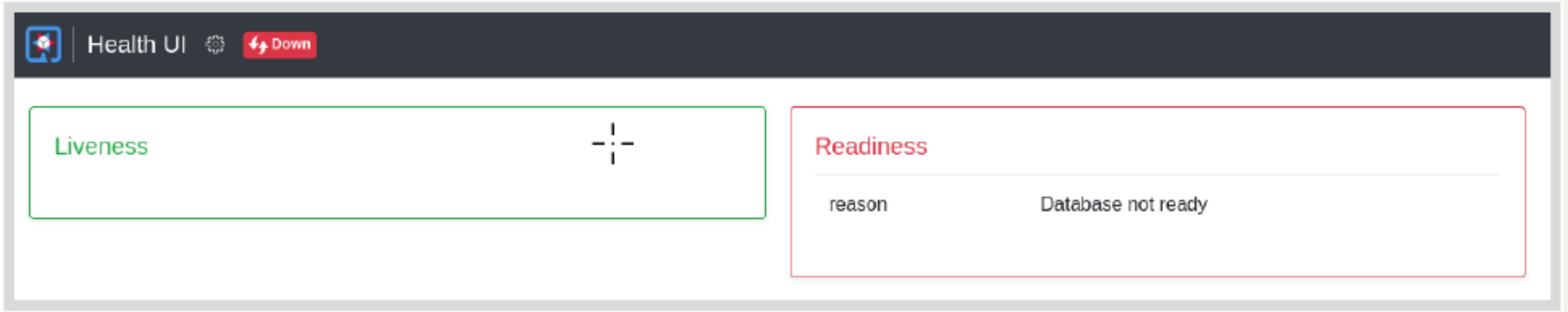

Figure 7.2: The health UI

Health Implementation

```java
@FunctionalInterface
public interface HealthCheck {
    HealthCheckResponse call();
}
```

```java
@Liveness
public class MyLivenessCheck implements HealthCheck {

    @Override
    public HealthCheckResponse call() {
        // Contruct the health response
    }
}
```

```java
@Readiness
public class MyReadinessCheck implements HealthCheck {

    @Override
    public HealthCheckResponse call() {
        // Contruct the health response
    }
}
```

HealthCheckResponse

```
@Override
public HealthCheckResponse call() {
  return HealthCheckResponse
            .named( "my-health-check" )
            .up()
            .build();
}
```

The abbreviated approach uses the `up()` and `down()` static methods of the `HealthCheckResponse` class to create a named response.

```
@Override
public HealthCheckResponse call() {
  return HealthCheckResponse.up( "my-health-check" );
}
```

HealthCheckResponse + Data

```
@Override
public HealthCheckResponse call() {
   return HealthCheckResponse.named( "load-average" )
      .withData( "average", "0,19" )
      .up()
      .build();
}
```

Health Checks - OpenShift

```
quarkus.openshift.liveness-probe.initial-delay=20s 1
quarkus.openshift.liveness-probe.period=45s 2
quarkus.openshift.readiness-probe.initial-delay=2s 3
quarkus.openshift.readiness-probe.period=15s 4
```

1. Time to delay the first request to check the service liveness.
2. Time between liveness probes.
3. Time to delay the first request to check the service readiness.
4. Time between readiness probes.