MicroProfile Health

Health checks are used to determine if service is running, shutdown, lack of disk space or maybe issues with the database connection. Because we added all the MicroProfile dependencies in our services, we have an endpoint called `/health` by default and if we visit that endpoint it will show us `UP` indicating that the service is up and running. We can add custom health checks if we want. In your terminal windows, start the `BookStoreClient` service if not already running and type the following to invoke the `/health` endpoint:

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8081/restapi/health
```

`Output`

```
{"checks":[],"outcome":"UP","status":"UP"}
```

Create a new file called `BookStoreClientHealthCheck.java` inside `com.kodnito.bookstore.rest` package and make it look like the following

```
package com.kodnito.bookstore.rest;

import javax.enterprise.context.ApplicationScoped;
import org.eclipse.microprofile.health.Health;
import org.eclipse.microprofile.health.HealthCheck;
import org.eclipse.microprofile.health.HealthCheckResponse;

@Health
@ApplicationScoped
public class BookStoreClientHealthCheck implements HealthCheck {

    @Override
    public HealthCheckResponse call() {
        return HealthCheckResponse.
                named("diskspace").
                up().
                withData("free", "900MB").
                build();
    }

}
```

Beans annotated with `@Health` and paired with `@ApplicationScoped` are discovered automatically. We implement the `HealthCheck` interface, and we override the `call()` method. Restart the `BookStoreClient` service and invoke the `/health` endpoint.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8081/restapi/health
```

Output

```
{"checks":[{"data":{"free":"900MB"},"name":"diskspace","state":"UP"}],"outcome":"UP","
status":"UP"}
```

Now we have our data in the checks list. We are not limited with hardcoded data, this is for test purpose only. We could add a check for database connection, disk space, or we could invoke `BookStore` service and check if the service is in maintenance or the service is down. Add the following property to the `microprofile-config.properties` file.

```
bookservice.url=http://localhost:8080/restapi
```

And now update the `BookStoreClientHealthCheck.java` to look like the following

```
@Health
@ApplicationScoped
public class BookStoreClientHealthCheck implements HealthCheck {

    @Inject
    @ConfigProperty(name = "bookservice.url")
    private String bookServiceUrl;

    @Override
    public HealthCheckResponse call() {

        boolean isHealthy = false;

        try {
            Client client = ClientBuilder.newClient();
            Response response =
client.target(bookServiceUrl).request(MediaType.APPLICATION_JSON)
                    .get();
            if (response.getStatus() != 200) {
                isHealthy = false;
            }
            isHealthy = true;
        } catch (Exception e) {
            isHealthy = false;
        }

        if (!isHealthy) {
            return HealthCheckResponse.named("bookservice")
                    .withData("service", "not available")
                    .down().build();
        }

        return HealthCheckResponse.named("bookservice")
                .withData("service", "available")
                .up().build();
    }

}
```

Here we inject the `bookservice.url` property, and then we invoke the `BookStore` service. We check the response code we get back and if it's not 200 (`OK`) then we return `service not available`. Kill the `BookService` and start the `BookServiceClient`, if it's not already started. In your terminal window type the following to invoke the `/health` endpoint.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8081/restapi/health
```

`Output`

```
{"checks":[{"data":{"service":"not
available"},"name":"bookservice","state":"DOWN"}],"outcome":"DOWN","status":"DOWN"}%
```

Now, because we stopped the `BookService`, we see that `service is not available` and the status is `DOWN`, if we start the `BookService` and invoke the `/health` again then we see that the output shows that the service is available and status is `UP`.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8081/restapi/health
```

`Output`

```
{"checks":[{"data":{"service":"available"},"name":"bookservice","state":"UP"}],"outcom
e":"UP","status":"UP"}
```

Here is an example of how to check database connection health:

```
@Log
@Health
@ApplicationScoped
public class MembershipHealthCheck implements HealthCheck {

    @Inject
    private DataSource datasource;

    @Override
    public HealthCheckResponse call() {

        HealthCheckResponseBuilder responseBuilder =
HealthCheckResponse.named("membership");
        try {
            Connection connection = datasource.getConnection();
            boolean isValid = connection.isValid(timeout);

            DatabaseMetaData metaData = connection.getMetaData();

            responseBuilder = responseBuilder
                    .withData("databaseProductName",
metaData.getDatabaseProductName())
                    .withData("databaseProductVersion",
metaData.getDatabaseProductVersion())
                    .withData("driverName", metaData.getDriverName())
                    .withData("driverVersion", metaData.getDriverVersion())
                    .withData("isValid", isValid);

            return responseBuilder.state(isValid).build();


        } catch(SQLException  e) {
            log.log(Level.SEVERE, null, e);
            responseBuilder = responseBuilder
                    .withData("exceptionMessage", e.getMessage());
            return responseBuilder.down().build();
        }
    }

    @Inject @ConfigProperty(name = "health.membership.dbtimeout", defaultValue = "5")
    private int timeout;


}
```

Example

Phillip Krüger, **GitHub**

Example

Example to check free memory

```
@Override
public HealthCheckResponse call() {
    return HealthCheckResponse
            .named("book-store-client")
            .state(true)
            .withData("memory", Runtime.getRuntime().freeMemory())
            .build();
}
```

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8081/restapi/health
```

Output

```
{"checks":[{"data":{"memory":129470808},"name":"book-store-
client","state":"UP"}],"outcome":"UP","status":"UP"}
```

Summary

In this chapter, we learned how to use MicroProfile Health in our application.