TP6 bis: réalisation d'une solution full stack avec docker-compose

Le but de ce TP est de créer une application full stack avec :

- tomcat
- apache2
- activemq
- mariadb
- rabbitmq

Eléments fournis

Serveur backend (tomcat)

- Fichier WAR fourni : hello-world.war
- Fichier Dockerfile

Code source du backend :

```
package mypackage;

import java.io.IOException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.lang.Exception;
import java.sql.*;
import java.net.*;
import java.util.Base64;
import java.nio.charset.StandardCharsets;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.eclipsesource.json.JsonObject;

@WebServlet("/helloworld/*")
public class HelloworldServlet extends HttpServlet {
    private static final long serialVersionUID = -7759593256585062849L;
    private static final Logger LOG = LoggerFactory.getLogger(HelloworldServlet.class);

    static final String JDBC_DRIVER = "org.mariadb.jdbc.Driver";
    static final String DB_URL = "jdbc:mariadb://db";
    static final String USER = "root";
    static final String PASS = "root";

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws IOException {
        LOG.info("serving request");
        LOG.info("special format for request {}", req);
        resp.setContentType("application/json");
        String json = new JsonObject()
                .add("msg", "Hello!")
                .add("db", testDB())
                .add("mail", testMail())
                .add("log", testLog())
                .toString();
        resp.getWriter().print(json);
    }
```

```
    private boolean testDB(){
        try {
            Class.forName("org.mariadb.jdbc.Driver");
            Connection con = DriverManager.getConnection(DB_URL, USER, PASS);
            LOG.info("Connected database successfully...");
            return true;
        } catch (Exception e){
            LOG.info("OOOOOPS \n " + e.getMessage());
            return false;
        }
    }

    private int testMail(){
        try{
            String uri = "amqp://guest:guest@mail:5672";
            com.rabbitmq.client.ConnectionFactory factory = new com.rabbitmq.client.ConnectionFactory();
            factory.setUri(uri);
            factory.setConnectionTimeout(5000);
            com.rabbitmq.client.Connection connection = factory.newConnection();
            return 200;
        } catch(Exception e){
            LOG.info("Fail to join log service \n " + e.getMessage());
            return 500;
        }
    }

    private int testLog(){
        try {
            URL url = new URL("http://log:8161");
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setRequestMethod("GET");
            return con.getResponseCode();
        } catch(Exception e){
            LOG.info("Fail to join log service \n " + e.getMessage());
            return 500;
        }
    }

}
```

Ce code renvoie en format JSON l'état des connections via les autres composants via l'appel à la resource HTTP "GET /helloworld"

- mariadb : nom de l'hôte de la base
- root/root : identifiants mariadb
- log : nom de l'hôte de activeMQ
- mail : nom de l'hôte de rabbitMQ

Application cliente (html)

Configuration fournie pour apache2

- httpd.conf (configuration apache, modules chargés)
- default.conf (configuration du virtualhost pour l'application cliente)

Fichier index.html (application qui appelle en AJAX le webservice GET /helloworld du serveur). Ouvir la console JS pour voir la réponse du serveur.

Exercice

Réaliser un fichier docker-compose qui lie les différents composants de la stack et permet d'accéder à l'application cliente sur http://localhost:7001