SPRC - Tema 2 - Microservicii & Docker

Responsabil: Bogdan Mocanu

Termen: 27.11.2023 - 10.12.2023

Titular curs: Florin Pop

1 Obiectivele temei

În cadrul acestei teme veți realiza un API REST ce comunică pe baza unor contracte formulate folosind protocolul HTTP. Acesta este utilizat pentru stocarea într-o bază de date a unor date meteorologice si pentru furnizarea unor informații pe baza unor specificații geografice conform figurii 1.

Figura 1: Prezentare generală a soluției propuse.

Obiectivele principale ale acestei teme de laborator sunt:

- Familiarizarea cu arhitecturile bazate pe API-uri REST;
- Utilizarea tehnologiei Docker pentru realizarea unei aplicații;
- Utilizarea Docker Compose pentru rularea unei configuratii de containere in mod centralizat;
- Utilizarea bazelor de date (cum ar fi PostgreSQL, MongoDB) in aplicatii;
- Utilizarea obiectelor de tip JSON.

Cunoştinţele necesare rezolvării acestei teme de casă sunt următoarele:

- Arhitectura bazată pe REST API;
- Tehnologia Docker;
- Gestiunea bazelor de date;
- Utilizarea obiectelor JSON;
- Noțiuni medii de programare indiferent de limbajul utilizat.

2 Noțiuni teoretice

2.1 Arhitectura bazată pe REST API

Printr-un REST API se intelege un server web de tip blackbox ce comunica pe baza unor contracte formulate folosind cereri HTTP. Serverul este considerat blackbox deoarece consumatorii acestui API nu stiu nimic despre implementarea serverului, ci sunt pusi in fata unui contract de comunicare definit de cereri si raspunsuri HTTP.

Cel mai concret exemplu de REST API este serverul pe care l-ati implementat la laboratorul 3 de SPRC.

2.2 Tehnologia Docker

Docker este o tehnologie de containerizare bazata pe motorul containerd. Utilizand Docker, dezvoltatorii pot realiza aplicatii fara sa tina cont de sistemul de operare pe care acestea vor rula, fara sa tina cont de specificatii sau dependente, codul rulandu-se intr-un container specializat si izolat.

Containerele pot fi rulate, atat utilizand comenzi individuale, cat si in bazat pe o configuratie centralizata, folosind Docker Compose. Docker Compose foloseste fisiere de configuratie .yml pentru a rula mai multe containere concomitent. Aceleasi principii care se aplica in cadrul rularii individuale de containere, se aplica si in cazul rularii Docker Compose, insa trebuie sa fiti familiari cu formatul unui fisier .yml.

Un exemplu de fisier docker-compose.yml poate fi regasit în Figura 2.

```yaml
version: '3.3'

services:
  mysql:
    platform: linux/x86_64
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - mysql
    image: wordpress:latest
    volumes:
      - wp_data_data:/var/www/html
    ports:
      - "8081:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: mysql:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
    db_data: {}
    wp_data_data: {}
```

Figura 2: Exemplu de fișier docker-compose.yml.

Comenzi utile:

- pentru a rula o configuratie de containere, este nevoie sa aveti un fisier de configurare *.yml*. Comanda de rulare este *docker-compose -f fisier-compose.yml up –build*. Daca fisierul compose se numeste docker-compose.yml si sunteti in acelasi folder, nu mai este nevoie sa dati parametrul -f.;
- pentru a opri configuratia,este nevoie sa rulati comanda *docker-compose down*;
- Pentru a sterge configuratia, este nevoie sa rulati comanda *docker-compose rm*.

Cel mai concrete exemple de lucru cu tehnologia Docker sunt la laboratorul 4 de SPRC.

2.3 Gestiunea bazelor de date

Pentru această tema trebuie să aveți noțiuni privind configurarea bazelor de date și executarea operațiilor de baza (CRUD - Create/Read/Update/Delete).

MongoDB

MongoDB este o bază de date NoSQL open-source orientată pe documente. Acestă bază de date beneficiază de suport din partea companiei 10gen. MongoDB face parte din familia de sistemelor de baze de date NoSQL. Diferența principală constă în faptul că stocarea datelor nu se face folosind tabele precum într-o bază de date relațională, MongoDB stochează datele sub formă de documente JSON cu scheme dinamice.

Instalarea și configurarea MongoDB Comunity Edition pe sistemul de operare Ubuntu se realizează folosind instrucțiunile de la adresa Install MongoDB Community Edition on Ubuntu.

PostgreSQL

PostgreSQL este un sistem de baze de date relationale. Este disponibil gratuit sub o licentă open source de tip BSD. PostgreSQL nu este controlat de nici o companie, își bazează dezvoltarea pe o comunitate răspândită la nivel global, precum și câteva companii dezvoltatoare.

Instalarea și configurarea PostgreeSQL pe sistemul de operare Ubuntu se realizează folosind instrucțiunile de la adresa How To Install PostgreSQL on Ubuntu 20.04.

2.4 Utilizarea obiectelor JSON

JSON este un acronim în limba engleză pentru JavaScript Object Notation, și este un format de reprezentare și interschimb de date între aplicații informatice. Este un format text, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri de date și este folosit în special pentru a transmite date structurate prin rețea, procesul purtând numele de serializare. Informații suplimentare privind utilizarea JSON se regăsesc în The JavaScript Object Notation (JSON) Data Interchange Format.

3 Descrierea soluției cerute

3.1 Cerințe arhitecturale

Functionalitatea temei va fi aceea a unui server care retine date metereologice si furnizeaza informatiipe baza unor specificatii geografice.

Va trebui sa implementati o configuratie de containere Docker formata din cel puțin 3 microservicii astfel:

- **REST API** in orice limbaj de programare doriti;
- **Baza de date**;
- **Utilitar de gestiune al bazelor de date**.

3.2 Ceinte pentru baza de date

Baza de date trebuie sa aiba urmatoarele entitati (tabele, colectii, etc... – in functie de ce tehnologie utilizati):

- **[Tari]** : id, nume_tara, latitudine, longitudine (**unic(nume_tara)**);
- **[Orase]** : id, id_tara, nume_oras, latitudine, longitudine (**unic(id_tara, nume_oras)**);
- **[Temperaturi]** : id, valoare, timestamp, id_oras (**unic(id_oras, timestamp)**).

Asa cum am precizat mai sus, nu sunteti restrictionati la o tehnologie de baza de date. Puteti folosi PostgreSQL, MongoDB, etc...
Precizari baza de date:

- Structura trebuie respectata, numele coloanelor/campurilor;
- Trebuie respectate conditiile de unicitate;
- Timestamp trebuie sa se genereze automat la adaugare (nu se da in cererea HTTP).

Configuratia de mai sus va trebui sa ruleze folosind Docker Compose si sa se bazeze pe un build local (pentru codul scris de voi).

3.3 Cerinte pentru REST API

Rest API-ul va consuma doar obiecte de tip JSON si va trebui sa stie sa raspunda la urmatoarele rute:

- **Countries**;
  - **POST**/api/countries;
    - Adauga o tara in baza de date;
    - **Body:** {nume: Str, lat: Double, lon: Double} – obiect;
    - **Succes**: 201 si {id:Int};
    - **Error**: 400 sau 409.
  - **GET**/api/countries;
    - Returneaza toate tarile din baza de date;
    - **Succes**: 200 si [{id: Int, nume: Str, lat: Double, lon: Double}, {...}, ...] - lista de obiecte;
  - **PUT**/api/countries/:id;
    - Modifica tara cu id-ul dat ca parametru;
    - **Body:** {id: Int, nume: Str, lat: Double, lon: Double} – obiect;
    - **Succes**: 200;
    - **Error**: 400 sau 404.
  - **DELETE**/api/countries/:id;
    - Sterge tara cu id-ul dat ca parametru;
    - **Succes**: 200;
    - **Error**: 400 sau 404.
- **Cities**;
  - **POST**/api/cities;
    - Adauga un oras in baza de date;
    - **Body:** {idTara: Int, nume: Str, lat: Double, lon: Double} – obiect;
    - **Succes**: 201 si {id:Int};
    - **Error**: 400, 404 sau 409.
  - **GET**/api/cities;

    - Returneaza toate orasele din baza de date;
    - **Succes**: 200 si [{id: Int, idTara: Int, nume: Str, lat: Double, lon: Double}, {...}, ...] - lista de obiecte;
  - **GET**/api/cities/country/:id_Tara;
    - Returneaza toate orasele care apartin de tara primita ca parametru;
    - **Succes**: 200 si [{id: Int, idTara: Int, nume: Str, lat: Double, lon: Double}, {...}, ...] - lista de obiecte;
  - **PUT**/api/cities/:id;
    - Modifica orasul cu id-ul dat ca parametru;
    - **Body:** {id: Int, idTara: Int, nume: Str, lat: Double, lon: Double} – obiect;
    - **Succes**: 200;
    - **Error**: 400, 404 sau 409.
  - **DELETE**/api/cities/:id;
    - Sterge orasul cu id-ul dat ca parametru;
    - **Succes**: 200;
    - **Error**: 400 sau 404.

- **Temperatures**
  - **POST**/api/temperatures;
    - Adauga o temperatura in baza de date;
    - **Body:** {id_oras: Int, valoare: Double} – obiect;
    - **Succes**: 201 si {id:Int};
    - **Error**: 400, 404 sau 409.
  - **GET**/api/temperatures?lat=Double&lon=Double&from=Date&until=Date;
    - Returneaza temperaturi in functie de latitudine,longitudine,datadeinceput sisau data de final. Ruta va raspunde indiferent de ce parametri de cerere se dau. Daca nu se trimite nimic, se vor intoarce toate temperaturile. Daca se da doar o coordonata, se face match pe ea. Daca se da doar un capat de interval, se respecta capatul de interval;
    - **Succes**: 200 si [{id: Int, valoare: Double, timestamp: Date}, {...}, ...] - lista de obiecte;
  - **GET**/api/temperatures/cities/:id_oras?from=Date&until=Date;
    - Returneaza temperaturile pentru orasul dat ca parametru de cale in functie de data de inceput si/sau data de final. Ruta va raspunde indiferent de ce parametri de cerere se dau. Daca nu se trimite nimic, se vor intoarce toate temperaturile pentru orasul respectiv. Daca se da doar un capat de interval, se respecta capatul de interval;
    - **Succes**: 200 si [{id: Int, valoare: Double, timestamp: Date}, {...}, ...] - lista de obiecte;
  - **GET**/api/temperatures/countries/:id_tara?from=Date&until=Date;
    - Returneaza temperaturile pentru tara data ca parametru de cale in functie de data de inceput si/sau data de final. Ruta va raspunde indiferent de ce parametri de cerere se dau. Daca nu se trimite nimic, se vor intoarce toate temperaturile pentru tara respectiva. Daca se da doar un capat de interval, se respecta capatul de interva;
    -
    - **Succes**: 200 si [{id: Int, valoare: Double, timestamp: Date}, {...}, ...] - lista de obiecte;
  - **PUT**/api/temperatures/:id;
    - Modifica temperatura cu id-ul dat ca parametru;
    - **Body:** {id: Int, idOras: Int, valoare: Double} – obiect;
    - **Succes**: 200;

    - Error: 400, 404 sau 409.
  - **DELETE**/api/temperatures/:id;
    - Sterge temperatura cu id-ul dat ca parametru;
    - Succes: 200;
    - Error: 400 sau 404.

4 Menţiuni suplimentare

4.1 Punctare

- Fiecare ruta din **Countries**, din **Cities** si **Temperatures** va avea cate **0.5p** – total **7.5p**;
  - Punctajul acordat pe rute este **binar** (**0.5p** sau **0p**). Nu se acorda punctaj intermediar. Daca, de exemplu, nu tratati un caz de eroare, sau nu returnati ce trebuie in caz de succes, veti primi **0p** pentru ruta respectiva. Pentru a primi **0.5p** trebuie ca rutele sa functioneze conform cerintelor;
  - Daca ruta nu face ce trebuie (de ex, ruta de PUT nu face actualizare in baza de date), se vor acorda **0p**.
- Realizarea unei configuratii de containere care sa ruleze cu Docker Compose – **1.5p**;
- Optimizarea configuratiei de containere – **1.0p**
  - Utilizare de variabile de mediu, unde este cazul - **0.25p**;
  - Impartire logica in retele de Docker (nu toate containerele in aceeasi retea default) - **0.25p**. De exemplu, utilitarul de baze de date nu trebuie sa fie in aceeasi retea cu API-ul;
  - Utilizarea volumelor pentru persistenta, unde este cazul - **0.25p**;
  - Utilizarea named DNS in cadrul aplicatiei pentru putea referi containerele dupa numele lor si nu dupa IP - **0.25p**.

4.2 Mentiuni

- Formatul datei in cadrul rutelor de temepratura trebuie sa fie de tipul **AAAA-LL-ZZ**;
- Pentru codul sursa scris de voi, scrieti si un fisier .dockerignore pentru a evita copierea folderului cu dependente (de ex: node_modules) in interiorul containerului. Lipsa fisierului .dockerignore (sau a fisierelor, daca implementati mai mult de un singur microserviciu) duce la anularea punctajului pe tema;

4.3 Arhivarea temei

- Numele arhivei va trebui sa fie **Tema2_Nume_Prenume_Serie_2023.zip**;
- Va trebui sa incarcati **doar** codul sursa si fisierele .yml. **NU incarcati** folderele de dependente (precum node_modules). **NU incarcati** foldere ce au fost folosite ca si volume locale in docker.

4.4 Clarificări suplimentare

- În arhiva temei nu se vor include artefactele care se cer a fi ignorate (deoarece nu sunt relevante). De exemplu, pentru Python este uzuală crearea unui Virtual Environment (de obicei, pus în folderul venv) pentru instalarea dependinţelor, atunci când dezvoltăm rulând direct pe maşina fizică;
- Pentru rularea testelor, ordinea acestora este extrem de importantă. recomandăm aranjarea testelor în ordine conform Fig.3;

- Prin utilitar de gestiune a bazei de date ne referim la o soluție (un program) cu sursă deschisă (Open Source) care dintr-o interfață web să permită realizarea operațiilor obișnuite asupra bazei de date (creare/vizualizare/actualizare/ștergere de date).

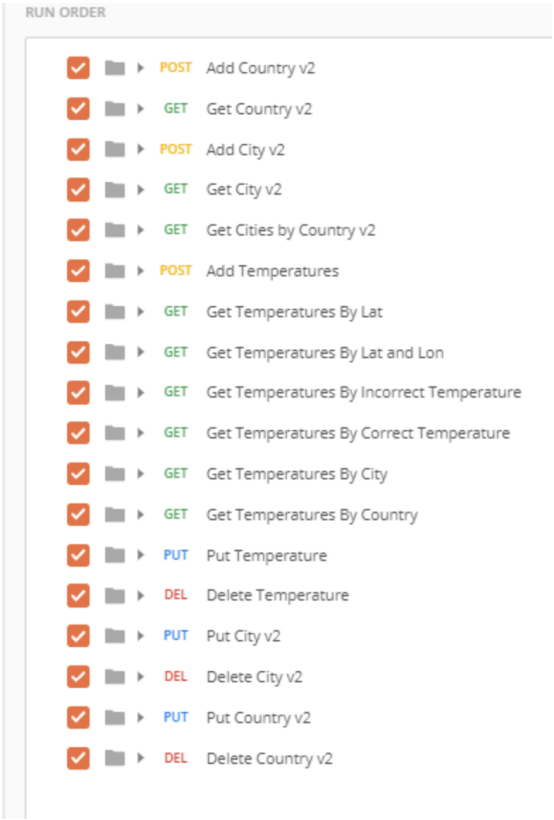

Figura 3: Ordinea de rulare a testelor.