Tema 2 – Servicii Web & Docker

responsabil: Alexandru Hogea

Premisa tema:

Daca in anii 2000, majoritatea aplicatiilor erau dezvoltate in sfera desktop, in jurul anului 2010 dezvoltatorii de aplicatii au fost pusi in fata unei schimbari de paradigma, domeniul preferat de lucru devenind domeniul web. Aceste schimbari au adus cu sine investitie tehnologica majora in procesele si uneltele din sfera web, conturandu-se astfel „cloud-ul", asa cum il stiti acum.

Obiectivele temei de casa:

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

Background Teoretic:

In cadrul acestei teme veti explora doua dintre aceste inovatii din domeniul web – cloud:

Arhitectura bazata 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.

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 mai jos[1]

```yaml
# docker-compose.yml
version: '3.8'
services:
  api:
    build: . # construieste imaginea dintr-un Dockerfile
    image: nume-imagine-registru:versiune # foloseste o imagine de pe registrul curent
    environment:
      NODE_ENV: development
      VARIABILA_DE_MEDIU: valoare
    ports:
     - "5000:80"
    networks:
     - network-laborator-3
  postgres:
    image: postgres:12
    environment:
      PGPASSWORD_FILE: /run/secrets/parola-mea-ultra-secreta
    volumes:
      - volum-laborator-3:/var/lib/postgresql/data
      - ./scripturi-initializare/init-db.sql:/docker-entrypoint-init.d/init-db.sql
    networks:
     - network-laborator-3

volumes:
  volum-laborator-3:

networks:
  network-laborator-3:

secrets:
  parola-mea-ultra-secreta:
    file: './parola-mea-nu-atat-de-secreta.txt'
```

(*nu este nevoie sa puneti si build si image impreuna, exemplul este ilustrativ*)

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 /cale/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`**

[1] https://ocw.cs.pub.ro/courses/cc/laboratoare/03

Cerintele temei:

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

Va trebui sa implementati o configuratie de containere formata din:

- (*cel putin*) **un REST API** in orice limbaj doriti
- **o baza de date**
- **utilitar de gestiune al bazelor de date**.

Baza de date va trebui 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 nu
- 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) si pe baza **imaginilor** pentru imaginile preluate de pe net (ex: baza de date)

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 }
- **Eroare:** 400 sau 409

➔ **GET** /api/countries
- Returneaza toate tarile din baza de date
- **Success**: 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`
- **Success**: 200
- **Eroare**: 400 sau 404

➔ **DELETE** /api/countries /:id
- Sterge tara cu id-ul dat ca parametru
- **Success**: 200
- **Eroare**: 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 }
- **Eroare:** 400 sau 409

➔ **GET** /api/ cities
- Returneaza toate orasele din baza de date
- **Success**: 200 si `[ {id: Int, idTara: Int, nume: Str, lat: Double, lon: Double}, {...}, ...] – lista de obiecte`

➔ **GET** /api/ cities /country/:idTara
- Returneaza toate orasele care apartin de tara primita ca parametru
- **Success**: 200 si `[ {id: 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`
- **Success**: 200
- **Eroare**: 400 sau 404

➔ **DELETE** /api/cities/:id
- Sterge orasul cu id-ul dat ca parametru
- **Success**: 200
- **Eroare**: 404

Temperatures:

➔ **POST** /api/temperatures
- Adauga o temperatura in baza de date
- **Body:** `{idOras: Int, valoare: Double} - obiect`
- **Succes:** 201 si { id: Int }
- **Eroare:** 400 sau 409

➔ **GET** /api/temperatures?lat=Double&lon=Double&from=Date&until=Date
- Returneaza temperaturi in functie de latitudine, longitudine, 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. Daca se da doar o coordonata, se face match pe ea. Daca se da doar un capat de interval, se respecta capatul de interval.
- **Success**: 200 si `[ {id: Int, valoare: Double, timestamp: Date}, {...}, ...] – lista de obiecte`

➔ **GET** /api/temperatures /cities/:idOras?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.
- **Success**: 200 si `[{id: Int, valoare: Double, timestamp: Date}, {...}, ...] – lista de obiecte`

➔ **GET** /api/temperatures /countries/:idTara?from=Date&until=Date
- Returneaza temperaturile pentru tara 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 tara respectiva. Daca se da doar un capat de interval, se respecta capatul de interval.
- **Success**: 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`
- **Success**: 200
- **Eroare**: 400 sau 404

➔ **DELETE** /api/temperatures /:id
- Sterge temperatura cu id-ul dat ca parametru
- **Success**: 200
- **Eroare**: 404

Punctare:

- Fiecare ruta din **Countries** si 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 – **1p**
    - Utilizare de variabile de mediu, unde este cazul
    - Impartire logica in retele de Docker (nu toate containerele in aceeasi retea default)
        - De exemplu, utilitarul de baza de date nu trebuie sa fie in aceeasi retea cu API-ul
    - Utilizarea volumelor pentru persistenta, unde este cazul
    - Utilizarea named DNS in cadrul aplicatiei pentru putea referi containerele dupa numele lor si nu dupa IP

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.

Arhiva Temei:

- Numele arhivei va trebui sa fie **Tema2_Nume_Prenume_Serie_2020.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
- Va trebuie sa incarcati un fisier **Readme** prin care sa motivati alegerea tehnologiei (atat pentru cod cat si pentru baza de date) si prin care sa motivati designul arhitectural utilizat
- Nerespectarea oricarei cerinte de mai sus, va duce la anularea punctajului pe tema