FER3

Diplomski studij

Raspodijeljeni sustavi

Treća laboratorijska vježba — Mikroservisi

Ak. god. 2023./2024.

1 Uvod

Cilj laboratorijske vježbe je izgraditi jednostavan raspodijeljeni sustav baziran na mikroservisima i kontejnerima kako je definirano u poglavlju 2. Studenti će naučiti implementirati raspodijeljene mikroservise koristeći okvir Spring Boot[1] i Docker[2] alat za kontejnerizaciju. Ova se vježba sastoji od:

- proučavanja priloženih primjera s predavanja.
- oblikovanja i implementacije mikroservisa za dohvaćanje temperature i vlage.
- oblikovanja i implementacije mikroservisa za dohvaćanje agregiranih vrijednosti temperature i vlažnosti.
- oblikovanja i implementacije konfiguracijskog poslužitelja za dohvaćanje konfiguracijskih podataka prilikom pokretanja mikroservisa.
- oblikovanja i implementacije registracijskog poslužitelja za registraciju i otkrivanje mikroservisa.
- kontejnerizacije stvorenih mikroservisa i njihovo izvršavanje pomoću alata Docker Compose[3].

U sklopu zadaće dostupna su **Često postavljena pitanja** 9.

1.1 Predaja

Studenti samostalno rješavaju programski zadatak i dužni su putem sustava Moodle predati arhivu koja se sastoji od:

- izvornog kôda svih mikroservisa
- izvornog kôda konfiguracijskog poslužitelja
- izvornog kôda registracijskog poslužitelja
- datoteka potrebnih za pokretanje sustava pomoću Docker Compose alata.

Napomena: Komponente moraju biti implementirane kao odvojeni projekti u odabranom razvojnom okruženju (npr. Eclipse, NetBeans, Visual Studio, IntelliJ Idea).

Arhiva se mora predati do roka koji će biti objavljen na stranici predmeta. Studenti koji predaju vježbu nakon navedenog roka dobit će 0 bodova iz vježbe.

Uz predaju digitalne arhive, organizirat će se **usmena prezentacija** predanog rješenja kada će studenti morati objasniti implementaciju i pokazati svoje razumijevanje primijenjenih komunikacijskih tehnologija, a kako bi se rješenje moglo ocijeniti. Termin i provedba usmenog ispita, tj. kolokviranje vježbe, bit će objavljeni na stranici predmeta. Kolokviranje je nužno kako bi predano rješenje bilo ocijenjeno.

Sve komponente sustava **moraju** se implementirati pomoću programskog jezika **Java** ili **Kotlin**. Svi mikroservisi u ovoj vježbi implementirani su pomoću Spring Boota, a pretpostavljeni alat za izgradnju aplikacije **Gradle** (dozvoljeno je koristiti i **Maven**). Preporučujemo da prvo proučite povezane materijale i predavanja koja se mogu pronaći na Youtubeu[4]. Arhiva koja sadrži izvorne kôdove mora biti nazvana **ime_prezime**. Dopušteno je slanje zip arhive koja sadrži izvoz projekta ako je implementiran u IDE-u (Eclipse, NetBeans itd.). Točna struktura arhive obrađena je u poglavlju 8.

Konzultacije se održavaju **utorkom od 10:00 do 11:00**, uz prethodnu **najavu** na MS Teamsu (kanal Laboratorijske vježbe) ili e-mailom.

[1] https://spring.io/projects/spring-boot
[2] https://www.docker.com/
[3] https://docs.docker.com/compose/
[4] `https://www.youtube.com/watch?v=Wtjx-75w5CY&list=PLy0T81VDh93YLJEEe5AxydDlXxUPrPs_B`

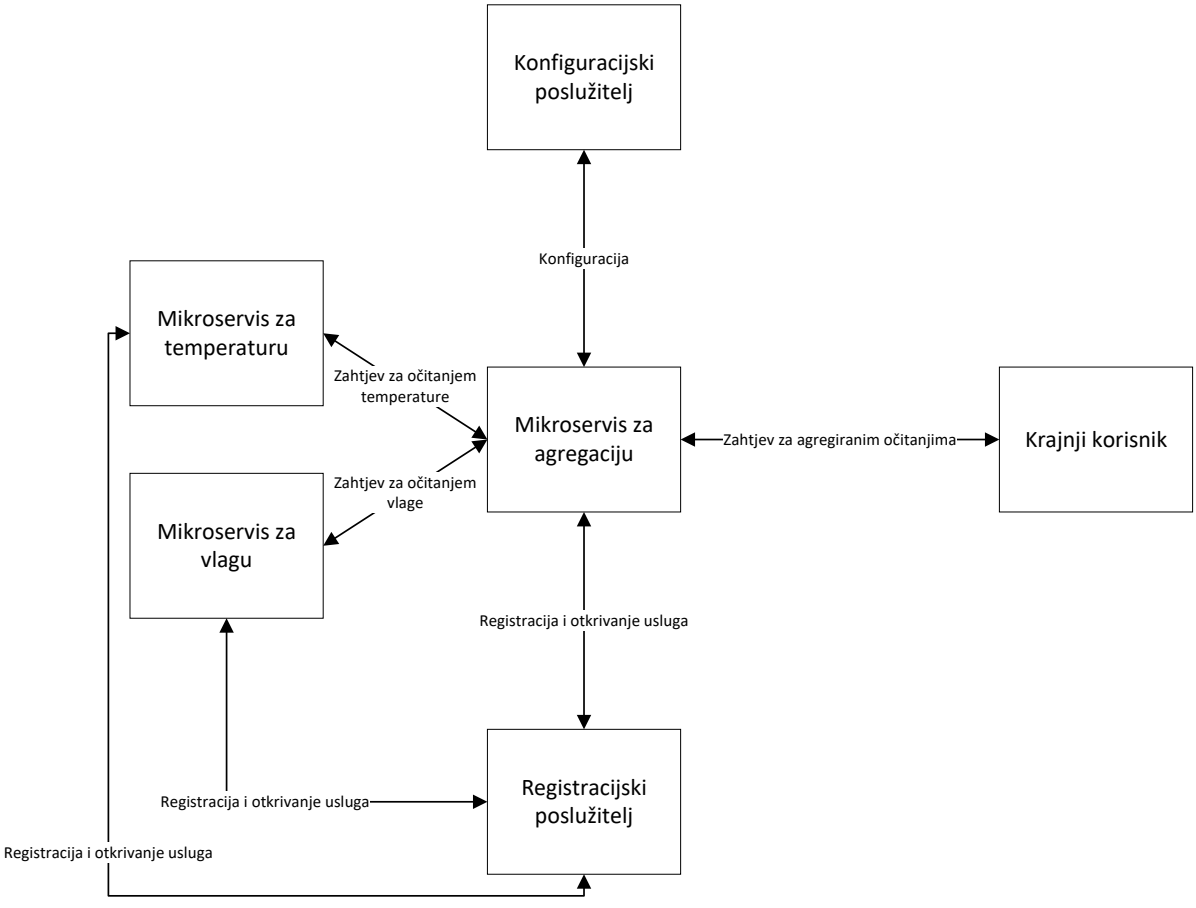
2 Arhitektura raspodijeljenog sustava

Raspodijeljeni sustav se koristi za praćenje očitanja **temperature** i **vlažnosti**, čiji se mikroservisi implementiraju zasebno. Funkcionalnost mikroservisa temperature i vlažnosti detaljno je objašnjena u poglavlju 3.

Nadalje, raspodijeljeni sustav sadrži mikroservis za **agregaciju** koji dohvaća podatke prikupljene od mikroservisa za vlagu i mikroservisa za temperaturu. Mikroservis za agregaciju zaprima zahtjev za dohvaćanje agregiranih očitanja temperature i vlažnosti. Zatim šalje 2 zasebna zahtjeva, jedan mikroservisu za temperaturu i jedan mikroservisu za vlažnost, nakon čega se primljena očitanja **agregiraju i zajedno šalju** krajnjem korisniku kao odgovor na zahtjev. Nadalje, mikroservis za agregaciju može se konfigurirati tako da odgovara očitanjem temperature u Celzijusima ili Kelvinima. Funkcionalnost mikroservisa za agregaciju je detaljno objašnjena u poglavlju 4.

Kako bi se osigurala konzistentnost konfiguracijskih podataka, odnosno mjerne jedinice temperature, te kako bi se izbjeglo ponovno pokretanje mikroservisa kada se konfiguracija promijeni, sustav koristi **konfiguracijski poslužitelj**. Mikroservisi koji imaju dinamičku konfiguraciju (u našem primjeru, mikroservis za agregaciju) moraju automatski učitati konfiguraciju s konfiguracijskog poslužitelja pri pokretanju. Funkcionalnost konfiguracijskog poslužitelja detaljno je objašnjena u poglavlju 5.

Kako bi omogućio sustavu da automatski otkrije nove mikroservise i kako bi sustav bio transparentniji, skalabilniji i tolerantniji na greške, sustav koristi **registracijski** poslužitelj koji omogućuje ostalim mikroservisima da se međusobno otkriju. Funkcionalnost konfiguracijskog poslužitelja detaljno je objašnjena u poglavlju 6. Pregled arhitekture raspodijeljenog sustava prikazan je na slici 1.



Slika 1: Arhitektura raspodijeljenog sustava

Kako bi se pojednostavila implementacija i korištenje sustava, tri stvorena mikroservisa, poslužitelj za konfiguraciju i poslužitelj za registraciju pakirani su u **Docker kontejnere** i mogu se pokrenuti pomoću **Docker Compose** alata. Pojedinosti o kontejnerizaciji pokrivene su u poglavlju 7.

3 Mikroservisi za temperaturu i vlagu

U ovom koraku vaš je zadatak implementirati dva mikroservisa koristeći programski radni okvir Spring Boot. Mikroservis za temperaturu mora izložiti **HTTP sučelje** koje vraća vrijednost trenutne temperature u Celzijusima. Primjer odgovora prikazuje JSON 1. HTTP zahtjev možete definirati proizvoljno.

JSON 1: Primjer odgovora na zahtjeva za očitanjem temperature u JSON formatu

```
{
  "name": "Temperature",
  "unit": "C",
  "value": 27
}
```

Mikroservis za vlagu izlaže **HTTP sučelje** koje vraća vrijednost trenutne vlažnosti u postocima. Primjer odgovora prikazuje JSON 2. HTTP zahtjev možete definirati proizvoljno.

JSON 2: Primjer odgovora na zahtjeva za očitanjem vlage u JSON formatu

```
{
  "name": "Humidity",
  "unit": "%",
  "value": 48
}
```

Budući da pravi senzori nisu spojeni na mikroservise, njihova se funkcija emulira pripremljenim očitanjima iz ulazne datoteke **readings.csv**, gdje očitanja (redci) imaju više vrijednosti (stupaca) koje predstavljaju očitanja. Za generiranje očitanja iz ulazne datoteke upotrijebite jednadžbu 1 za dobivanje broja retka koji ćete koristiti za očitavanje temperature i vlažnosti barometarskog tlaka, relativne vlažnosti i koncentracije plinova CO i NO2 ili SO2 iz ulazne datoteke:

$$red \leftarrow (brojAktivnihSekundi \ \% \ 100) + 1 \tag{1}$$

Varijabla *brojAktivnihSekundi* je razlika između trenutnog vremena i ponoći, 1. siječnja 1970. UTC, mjerena u **minutama**[5]. Prilikom pokretanja se pročita datoteka i sva očitanja za pojedini mikroservis se **moraju** zapisati u **bazu podataka** pojedinog mikroservisa (na primjer, baza podataka H2).

4 Mikroservis za agregaciju

U ovom koraku vaš je zadatak koristiti Spring Boot okvir za implementaciju mikroservisa koji izlaže **HTTP sučelje** koja vraća trenutne vrijednosti temperature i vlažnosti. HTTP zahtjev možete definirati proizvoljno. Primjer odgovora prikazuje JSON 3.

Mikroservis mora biti implementiran tako poziva mikroservis za temperaturu i vlagu, agregira odgovore i vraća rezultate **bez** pohranjivanja agregiranih očitanja u bazu podataka.

Napomena: da u ovom koraku trebate statički postaviti IP adresu mikroservisa na koji se povezujete. Ovaj problem je riješen u poglavlju 6 dodavanjem registracijskog poslužitelja.

[5] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#currentTimeMillis()

JSON 3: Primjer odgovora na zahtjeva za agregiranim očitanjem u JSON formatu

```
[
  {
    "name": "Humidity",
    "unit": "%",
    "value": 48
  },
  {
    "name": "Temperature",
    "value": 27,
    "unit": "C"
  }
]
```

U zasebnoj konfiguracijskoj datoteci unutar projekta (na primjer, **application.yml**) izdvojite jedinicu temperature koja se koristi pri generiranju agregiranog odgovora, koji može biti u Kelvinima ili u Celzijusima. Mikroservis mora implementirati logiku koja pretvara Celzijuse u Kelvine. Primjer odgovora u Kelvinu prikazuje JSON 4

JSON 4: Primjer odgovora na zahtjeva za agregiranim očitanjem u Kelvinima u JSON formatu

```
[
  {
    "name": "Humidity",
    "unit": "%",
    "value": 48
  },
  {
    "name": "Temperature",
    "value": 300.15,
    "unit": "K"
  }
]
```

Napomena: promjene u željenoj mjernoj jedinici temperature zahtijevaju ponovno pokretanje mikroservisa. Ovaj je problem riješen u poglavlju 5 dodavanjem konfiguracijskog poslužitelja.

5 Konfiguracijski poslužitelj

Sljedeći zadatak je implementacija konfiguracijskog poslužitelja. Projekt Spring Cloud Config[6] je obrađen u predavanjima i koristit će se kao konfiguracijski poslužitelj.

Git repozitorij gdje se nalazi konfiguracija možete implementirati **lokalno** ili na **udaljenom poslužitelju** (npr. Github). Na repozitoriju pohranite konfiguracijsku datoteku u koju unosite mjernu jedinicu za temperaturu.

Nakon što implementirate konfiguracijski poslužitelj, trebate **modificirati mikroservis za agregaciju** da čita konfiguraciju s konfiguracijskog poslužitelja. Nakon što povežete mikroservis s konfiguracijskim poslužiteljem, provjerite ažurira li mikroservis temperaturnu jedinicu s konfiguracijskog poslužitelja bez ponovnog pokretanja mikroservisa.

[6] https://spring.io/projects/spring-cloud-config

6 Registracijski poslužitelj

Kao sljedeći korak, kako bi se spojili na mikroservis bez poznavanja njegove točne lokacije (točnije, IP adrese i vrata), sustavu se dodaje registracijski poslužitelj. Registracijski poslužitelj koji će se koristiti je Spring Cloud Netflix Eureka[7], koji je obrađen u sklopu predavanja.

Nakon što implementirate registracijski poslužitelj, trebate **modificirati sve mikroservise** da se automatski **registriraju** na registracijskom poslužitelju pri pokretanju i koristiti ga za otkrivanje drugih usluga. Zatim promijenite logiku unutar mikroservisa za agregaciju tako da zahtjev za mikroservisima temperature i vlažnosti **ne koristi statički definiranu IP adresu i port**, već koristi naziv mikroservisa koji je registriran na Eureka poslužitelju.

7 Kontejnerizacija

Posljednji korak je migracija svih mikroservisa sustava (temperature, vlažnosti, mikroservisa za agregaciju, konfiguracijskog poslužitelja i poslužitelja za registraciju) u Docker kontejnere. Preporučujemo da prvo proučite povezane materijale i predavanja koja se mogu pronaći na Youtubeu[8]. Spring Boot pruža dodatke (*plugin*) za Maven i Gradle[9] koji se mogu koristiti za kontejnerizaciju. Na primjer, Gradle dodatak ima zadatak (*task*) **bootBuildImage**. Dodatke nije obavezno koristiti.

Kako bi se izbjeglo pojedinačno pokretanje kontejnera, koristi se Docker Compose. Budući da mikrousluge za agregaciju, temperaturu i vlagu ovise o konfiguracijskom i registracijskom poslužitelju, prvo se moraju pokrenuti konfiguracijski i registracijski poslužitelj. Definirajte konfiguracijsku datoteku koju Docker Compose koristi za pokretanje samo konfiguracijskog i registracijskog poslužitelja (na primjer, kreiranjem datoteke **docker-compose-infrastructure.yml** i pokretanjem naredbe **docker-compose -f docker-compose-infrastructure.yml up**). Također, definirajte konfiguracijsku datoteku koju Docker Compose koristi za pokretanje mikroservisa za agregaciju, temperaturu i vlagu (na primjer, kreiranjem datoteke **docker-compose-services.yml** i pokretanjem naredbe **docker-compose -f docker-compose-services.yml up**). Provjerite može li se distribuirani sustav pokrenuti pomoću Docker kontejnera.

[7] https://spring.io/projects/spring-cloud-netflix
[8] https://www.youtube.com/watch?v=kUiJA0_Uhe0&list=PLy0T81VDh93awNP1X91-WPNgOykME_2CI
[9] https://spring.io/guides/topicals/spring-boot-docker/

8 Konačna struktura laboratorija

Konačna struktura datoteka predanih unutar arhive mora izgledati kako slijedi:

```
├── config
│   └── application.yml
├── microservices
│   ├── aggregator-microservice
│   │   ├── build.gradle
│   │   ├── gradle
│   │   ├── gradlew
│   │   ├── gradlew.bat
│   │   ├── settings.gradle
│   │   └── src
│   ├── config-server-microservice
│   │   ├── build.gradle
│   │   ├── gradle
│   │   ├── gradlew
│   │   ├── gradlew.bat
│   │   ├── settings.gradle
│   │   └── src
│   ├── eureka-server
│   │   ├── build.gradle
│   │   ├── gradle
│   │   ├── gradlew
│   │   ├── gradlew.bat
│   │   ├── settings.gradle
│   │   └── src
│   ├── humidity-microservice
│   │   ├── build.gradle
│   │   ├── gradle
│   │   ├── gradlew
│   │   ├── gradlew.bat
│   │   ├── settings.gradle
│   │   └── src
│   └── temperature-microservice
│       ├── build.gradle
│       ├── gradle
│       ├── gradlew
│       ├── gradlew.bat
│       ├── settings.gradle
│       └── src
├── docker-compose-infrastructure.yml
└── docker-compose-services.yml
```

Napomena: nakon pokretanja naredbe **gradle clean build**, veličina direktorija koji sadrži Gradle projekt značajno se povećala. Kako biste izbjegli predavanje datoteka značajne veličine, **morate** smanjiti veličinu Gradle projekata koje predajete. Prije predaje, pokrenite naredbu **gradle clean** u svakom Gradle projektu, a zatim izbrišite direktorij **.gradle** unutar svakog projekta.

9 Često postavljena pitanja

1. Treba li sva mjerenja spremiti u bazu ili samo temperaturu/vlagu?
    - Svejedno je, bitno je samo koristiti bazu podataka.
2. Može li se koristiti application.properites umjesto application.yml?

- Može.

3. Treba li se ažuriranje temperaturne jedinice s konfiguracijskog poslužitelja odvijati bez ponovnog pokretanja servisa?
   - Ne treba, dovoljno je da se servis ponovno pokrene.
4. Može li registracijski poslužitelj imati statičku IP adresu unutar kontejnera?
   - Može, a može i imati simboličku adresu iz docker networka, odnosno svejedno je.
5. Treba li se koristiti specifične zahtjeve (npr. POST, GET) ili specifičan URL?
   - Ne mora, zahtjev definirajte proizvoljno.
6. Koje verzije razvojnog okruženja je preporučeno koristiti?
   - Java verzija 17 ili 21
   - id ’org.springframework.boot’ verzija ’3.2.0’
   - id ’io.spring.dependency-management’ verzija ’1.1.4’
   - ’springCloudVersion’ version ’23.0.0’
   - gradle verzija 8.5

   Minimalne preporučene verzije:
   - Java verzija 17
   - id ’org.springframework.boot’ verzija ’3.0.0’
   - id ’io.spring.dependency-management’ verzija ’1.1.0’
   - ’springCloudVersion’ version "2022.0.0-RC2"
   - gradle verzija 7.6
7. **Pitanja studenata prijašnjih godina**
   - Q1: Imam problem kod dohvaćanja adresa od mikroservisa za temperaturu i vlažnost. Adrese dohvaćam sljedećim naredbama:

     ```
     private DiscoveryClient discoveryClient;
     ServiceInstance tempService = discoveryClient.getInstances("temperature-microservice").get(0);
     String tempUrl = tempService.getUri().toString() + "/humidity";
     ```

     I ako pokrenem samostalno iz Intelij-a, radi sve kako treba. Agregator pošalje zahtjeve i dobije odgovore.

     Ako pokrenem sve pomoću Dockera, agregator ne uspije poslati zahtjeve (dobijem grešku da host nije prepoznat)

     Koliko sam shvatio, ne dobije dobru adresu s Eureka poslužitelja.
   - A1: Pokušajte adresu Eureka poslužitelja na Agregatorskom mikroservisu postaviti na simboličko ime Eureke, ili na hardkodiranu ip adresu Eureke u docker podmreži. Moguće je da Vaš Agregator ne razlučuje host.docker.internal adresu ispravno.
   - A2: Spajanje na Eureka poslužitelja je radilo, koliko sam shvatio problem je bio što je u ServiceInstance objektu kao hostName postavljen ID kontejnera u Dockeru. Ako netko naiđe na sličan problem, ovo mi je pomoglo: `https://stackoverflow.com/questions/63871795/why-application-in-docker-register-in-eureka-like-some-hash-instead-of-ip-addres`