Diplomski studij

Informacijska i komunikacijska tehnologija
Telekomunikacije i informatika

Računarstvo

Programsko inženjerstvo i informacijski sustavi
Računarska znanost

Raspodijeljeni sustavi

Upute za izradu laboratorijske vježbe

Mikrousluge

Ak. god. 2020./2021.

Sadržaj

1 Uvod

CILJ LABORATORIJSKE VJEŽBE

U praksi utvrditi i ponoviti gradivo s predavanja iz područja mikrousluga i kontejnera. Studenti će naučiti implementirati raspodijeljeni sustav mikrousluga koristeći programski okvir *Spring Boot* [1] te alat za kontejnerizaciju *Docker*[2].

ZADATAK

Ova laboratorijska vježba sastoji se od sljedećih koraka:

1. proučavanje primjera s predavanja i video materijala,
2. programiranje mikrousluga za dohvaćanje temperature i vlage,
3. programiranje mikrousluge za dohvaćanje agregiranih podataka temperature i vlage,
4. programiranje konfiguracijskog poslužitelja i dohvaćanje konfiguracije prilikom pokretanja mikrousluga,
5. programiranje registracijskog poslužitelja i registracija mikrousluga na poslužitelj i
6. pakiranje stvorenih mikrousluga u kontejnere i pokretanje cijelog sustava pomoću alata *Docker Compose*.

PREDAJA

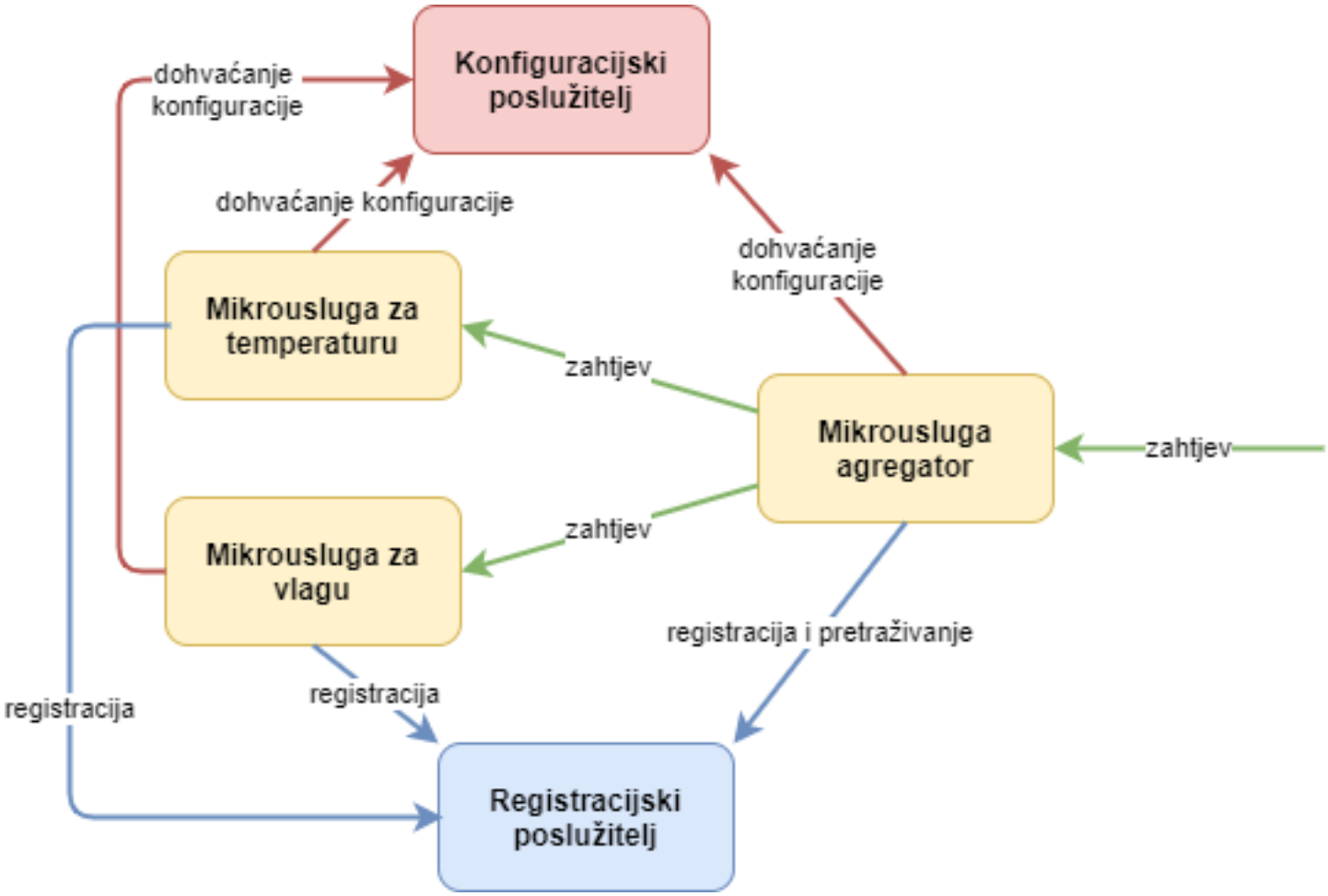
Studenti su dužni u zadanom roku putem sustava Moodle predati arhivu koja se sastoji od sljedećih dijelova:

1. izvorni kod svih mikrousluga, konfiguracijskog i registracijskog poslužitelja,
2. ***docker-compose.yml*** datoteka čijim pokretanjem se stvaraju i pokreću sve usluge implementiranog sustava u obliku Docker kontejnera.

Osim predaje datoteka u digitalnom obliku, bit će organizirana i usmena predaja putem sustava Microsoft Teams na kojoj će se ispitivati razumijevanje koncepata potrebnih za izradu domaće zadaće te poznavanje vlastitog programskog koda.

Sav programski kod piše se u programskom jeziku Java koristeći programski okvir **Spring Boot** te *build* alat **Gradle**.

2 Arhitektura sustava



Raspodijeljeni sustav služi za praćenje očitanja temperature i vlage čije su usluge implementirane zasebno te se korisniku sustava izlaže samo jedna usluga koja služi za agregaciju očitanih podataka. Uloga usluge za agregaciju jest primiti zahtjev za dohvaćanje agregiranih mjerenja temperature i vlage nakon čega se šalju 2 odvojena zahtjeva uslugama za temperaturu i vlage, zatim se dohvaćena mjerenja agregiraju i zajedno šalju kao odgovor na zahtjev korisniku. Izlaganjem jedne usluge korisniku se skriva raspodijeljenost sustava jer sa strane korisnika postoji samo jedna usluga kojoj se šalje zahtjev i s koje dolazi odgovor.

Kako bi se osigurala konzistentnost konfiguracijskih podataka, poput identifikatora mikrousluga ili mjernih jedinica mjerenja, potrebno je implementirati konfiguracijski poslužitelj. Tako će se prilikom pokretanja svake mikrousluge automatski učitati konfiguracija s poslužitelja.

Za raspodijeljeni sustav mikrousluga poželjno je da lokacija mikrousluge nije statički definirana kako bi se omogućila transparentnost, skalabilnost i otpornost na ispade. Stoga je potrebno implementirati registracijski poslužitelj kako bi klijenti (u našem slučaju mikrousluga agregator) mogli otkriti gdje se nalazi neka usluga.

3 Mikrousluge za dohvaćanje temperature i vlage

Sve mikrousluge u ovoj vježbi bit će implementirane pomoću programskog okvira *Spring Boot* koji je obrađen na predavanjima. Preporučujemo da prvo proučite video snimke o *Springu* profesora Kušeka koja se nalaze na Youtubeu na poveznici [3].

Vaš je zadatak pomoću programskog okvira Spring Boot implementirati dvije REST usluge koje na zahtjev HTTP GET na putanju */current-reading* vraćaju vrijednost trenutnog mjerenja temperature, odnosno vlage.

Za bazu podataka potrebno je koristiti *in-memory* bazu H2. Kako biste napunili bazu sa simuliranim mjerenjima dostupna je datoteka *mjerenja.csv* u kojoj se nalaze simulirana mjerenja temperature (u stupnjevima Celzijusa), tlaka, vlažnosti te vrijednosti plinova CO, NO_2 i SO_2 u zraku. Prilikom pokretanja Spring Boota potrebno je učitati datoteku i spremiti mjerenja u bazu podataka tako da se svakom mjerenju dodijeli ID u rasponu od 0 do broj mjerenja (na primjer: prvi redak imat će ID=0, drugi redak ID=1 itd.).

Nakon učitavanja podataka u bazu pokreće se Spring Boot poslužitelj koji prilikom dolaska zahtjeva HTTP GET na putanju */current-reading* dohvaća mjerenje iz baze podataka pomoću ID-a koji se računa sljedećim izrazom:

$$ID = 4 \cdot Sat(trenutno\_vrijeme) + \frac{Minuta(trenutno\_vrijeme)}{15}$$

Primjer: u 15.50 sati ID mjerenja koje se dohvaća bit će $15 \cdot 4 + 50/15 = 60 + 3 = 63$

Na kraju ovog koraka imat ćete dvije mikrousluge:

- *temperature-microservice* – Spring Boot mikrousluga koja vraća trenutnu vrijednost temperature kao odgovor na zahtjev HTTP GET na putanju */current-reading*
- *humidity-microservice* - Spring Boot mikrousluga koja vraća trenutnu vrijednost vlage kao odgovor na zahtjev HTTP GET na putanju */current-reading*

4 Mikrousluga za dohvaćanje agregiranih podataka temperature i vlage

U ovom koraku Vaš je zadatak pomoću programskog okvira Spring Boot implementirati REST uslugu koja na zahtjev HTTP GET na putanju */readings* vraća trenutne vrijednosti temperature i vlažnosti zraka. Usluga treba biti implementirana tako da u pozadini poziva mikrouslugu za dohvaćanje temperature i mikrouslugu za dohvaćanje vlažnosti zraka te agregirane rezultate vraća korisniku kao odgovor na zahtjev. Primijetite kako ćete u ovom koraku morati statički definirati IP adresu i vrata mikrousluge na koju se povezujete, a taj problem riješit ćemo kasnije dodavanjem registracijskog poslužitelja.

U zasebnu konfiguracijsku datoteku unutar projekta (primjerice *application.yml*) izdvojite podatak o mjernoj jedinici temperature koju je potrebno vratiti korisniku. Također, potrebno je implementirati logiku koja će, ako je potrebno korisniku vratiti vrijednost temperature u Kelvinima, nakon dohvaćanja temperature pretvoriti vrijednost iz stupnjeva Celzijusa u Kelvine.

5 Konfiguracijski poslužitelj i dohvaćanje konfiguracije prilikom pokretanja

Sljedeći zadatak je, kao što je već spomenuto, implementirati konfiguracijski poslužitelj. Za implementaciju poslužitelja koristit ćete projekt Spring Cloud Config [4] koji je obrađen na predavanjima.

Git repozitorij na kojem se nalazi konfiguracija možete implementirati lokalno ili na Githubu, no pazite da nakon stvaranja repozitorija na Githubu promijenite naziv *default* grane iz *main* u *master* jer se Spring Cloud Config automatski povezuje na granu *master*. Unutar repozitorija stvorite konfiguracijsku datoteku u koju ćete unijeti podatak o mjernoj jedinici temperature.

Nakon što ste implementirali konfiguracijski poslužitelj potrebno je promijeniti kod mikrousluga tako da konfiguraciju čitaju s konfiguracijskog poslužitelja. Nakon što ste povezali sve tri mikrousluge s poslužiteljem, provjerite da mikrousluga za agregaciju podataka učitava mjernu jedinicu temperature s konfiguracijskog poslužitelja.

6 Registracijski poslužitelj i registracija mikrousluga

Kao što je objašnjeno ranije, kako bi se mogli povezati na mikrouslugu bez da znamo njenu točnu lokaciju (preciznije IP adresu i vrata usluge) potrebno je implementirati registracijski poslužitelj. Vaš je zadatak implementirati registracijski poslužitelj Spring Cloud Netflix Eureka [5] koji je obrađen na predavanjima.

Nakon što ste implementirali registracijski poslužitelj potrebno je promijeniti kôd mikrousluga tako da se automatski prilikom pokretanja registriraju na registracijski poslužitelj te da ga koriste za pretraživanje drugih usluga. Kao naziv usluge prilikom registracije za mikrouslugu za agregiranje koristite *aggregator-ms*, mikrouslugu za temperaturu *temperature-ms* i mikrouslugu za vlagu *humidity-ms*. Nakon toga promijenite logiku unutar mikrousluge za agregaciju tako da zahtjev prema mikrouslugama za temperaturu i vlagu ne ide preko statički definirane IP adrese i vrata, nego da se ime mikrousluge čita iz konfiguracijske datoteke nakon čega se po tom imenu preko poslužitelja Eureka pristupa uslugama.

7 Pokretanje usluga pomoću kontejnera

U posljednjem koraku laboratorijske vježbe potrebno je sve usluge u sustavu (tri mikrousluge te konfiguracijski i registracijski poslužitelj) pokrenuti unutar Docker kontejnera. Prije nego što započnete s posljednjim korakom preporučuje se proći kroz videopredavanja o Dockeru profesora Kušeka koja se nalaze na Youtubeu na poveznici [6].

Za početak je potrebno stvoriti Docker slike iz svakog projekta za što je **obavezno** koristiti Gradle *plugin* **JIB** za čije korištenje se upute također nalaze na poveznici [6]. Nakon što ste stvorili sve slike sljedeći korak je pokrenuti sve usluge pomoću Docker kontejnera. Kako se kontejneri ne bi morali pokretati zasebno te kako bi se omogućilo jednostavno umrežavanje kontejnera, potrebno je stvoriti datoteku za Docker Compose koja će pokrenuti sve usluge unutar kontejnera. S obzirom da mikrousluge za agregaciju, temperaturu i vlagu ovise o konfiguracijskom i registracijskom poslužitelju, u Docker Compose dodajte ovisnosti *depends_on* te koristite skriptu *wait-for-it.sh* koju možete preuzeti na poveznici [7]. Skripta *wait-for-it.sh* je *bash* skripta koja u pozadini samo čeka da se otvore TCP vrata na željenom računalu, što se može primijeniti u scenariju laboratorijske vježbe jer mikrousluge moraju čekati da se pokrenu konfiguracijski i registracijski poslužitelj kako bi mogle ispravno raditi. Primjer korištenja skripte možete pronaći na poveznici [8].

8 Konačna struktura

Konačna datotečna struktura koja se predaje unutar arhive mora izgledati ovako:

```
config
├── application.yml
└── docker-compose.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
```

Primijetite kako se nakon izvršavanja naredbe **"gradle clean build"** veličina direktorija u kojem se nalazi Gradle projekt naglo povećala (sada iznosi nekoliko desetaka megabajta). Kako ne biste morali u Moodle učitati datoteke od nekoliko stotina megabajta, potrebno je smanjiti veličinu Gradle projekata koje predajete. Postoje 2 načina kako to možete učiniti:

1. Pokrenuti Bash skriptu **remove_transient_data.sh** koju možete preuzeti na web-stranici predmeta. Važno je naglasiti da vam je potrebna Bash ljuska (na Windowsima možete instalirati Git BASH i unutar nje ju pokretati) za njeno izvršavanje te da morate pratiti gore navedenu datotečnu strukturu. Skriptu pokrećete iz direktorija u kojem se nalaze vršni direktoriji strukture *config* i *microservices*. Skripta ulazi unutar direktorija svakog od projekata i briše tranzijentne datoteke;
2. Pokrenuti naredbu **"gradle clean"** nad svakim projektom te nakon toga izbrisati direktorij **".gradle"** unutar svakog projekta.

Nakon što ste smanjili veličine projekata, dodajte direktorije *config* i *microservices* u arhivu pod nazivom **"rassus_lti_grupa_X.zip"** gdje X predstavlja broj vaše grupe.

Literatura

[1] https://spring.io/projects/spring-boot

[2] https://www.docker.com/

[3] https://www.youtube.com/playlist?list=PLy0T81VDh93YLJEEe5AxydDlXxUPrPs_B

[4] https://cloud.spring.io/spring-cloud-config/reference/html/

[5] https://spring.io/guides/gs/service-registration-and-discovery/

[6] https://www.youtube.com/playlist?list=PLy0T81VDh93awNP1X91-WPNgOykME_2CI

[7] https://github.com/vishnubob/wait-for-it

[8] https://docs.docker.com/compose/startup-order/