Monitoring plants

JUN 28, 2020

Univerzitet u Nišu

Elektronski fakultet

Profesor:

Dragan Stojanović

Studenti:

Miloš Đorđević 16079

Natalija Pešić 16458

REZIME PROJEKTA

1.Projektni zadatak

Cilj je projektovati i implementirati mikroservisnu arhitekturu IoT (Internet of Things) sistema, koja se sastoji iz *Device, Data, Analytics, Gateway i Command mikroservisa kao i WEB dashboard-a*. Bitan deo zadatka je da se svaki servis nalazi u okviru Docker container-a. Mikroservisi koriste ASP.NET Core i NodeJS (Molecular) i Python mikrosevise kao sto je i preporuceno. Za asinhronu komunikaciju izmedju mikroservisa po principu Publish/Subscribe koristi se NATS Message broker kao i REST (PUT, POST, GET). Neophodno je da se svaka baza podataka vezana za neki mikroservis kao i sam taj mikroservis nalazi u Docker container-u. U našem slucaju mikroservisi koji imaju svoju bazu podataka su Analytics i Data mikroservisi i oba koriste InfluxDB (TSDB, NoSQL).
Podatke koje generišu virtuelni senzori moguće je naći na internetu. Podaci koji se koriste (obradjuju) sadrže informacije kao što su:
Temperatura zemljišta, vlaga zemljišta, temperatura vazduha i relativna vlažnost vazduha za glavne vrste vegetacije u jednom delu Kanade.
Baza se sastoji od 5 .csv fajla i svaki odgovara jednoj vegetacionoj vrsti. Fajlove sa podacima kao i više detalja o ovim podacima možete da pronadjete na sledecem linku:

https://drive.google.com/drive/folders/1hNCFbkc2YcAorS0C0H7lEKkQyMXGpxlN?usp=sharing

Cilj našeg projekta je ilustrovati ovakav tip arhitekture na realnom problemu. Kao primer zadatka je uzet upravljanje uredjajima za navodnjavanje, za vlaznost vazduha i klima za modifikacije temperature vazduha. Praćenjem podataka kao trenutnim stanjem biljaka odredjeni uredjaji se aktiviraju sa ciljem postizanja idealnog staništa za posmatranu biljku.

2. Opis mikroservisa

1. Device Microservice

Svrha servisa je da upravlja jednim ili više Senzora ili Aktuatora.
U projektu je on implementiran kroz fajl services/**device.service.js** i koristi Moleculer (NodeJS).
Na samom pocetku fajla implementirane su odredjene akcije koje služe upravo za upravljanje modifikovanjem parametara. Definisani su parametri za aktuatore kao i za senzore.

```
created() {
    // actuator
    this.airCoolingOn = false;
    this.waterPumpOn = false;
    this.humidifierOn = false;

    this.humidifierHumidityLevel = 75;
    this.waterPumpLitersPerMinute = 3;
    this.airCoolerTemperature = 25; // Celsius
    // sensor
    this.sensorId = 1;
    this.paramsNumber = 4;
    this.dataIndex = 0;
    this.interval = 5000;
    // init
    this.readData();
    this.init();
}
```

Ovi parametri se modifikuju pomoću akcija kao sto su **setAirCoolerTemperature, turnHumidifierOnOrOff, changeReadInterval**...

Kako oni rade?

```
changeReadInterval: {
    params: {
        sensorId: { type: "number" },
        value: { type: "number" }
    },
    async handler(ctx) {
       console.log('Receieved "changeReadInterval" put request with payload: ', ctx.params);
       this.interval = ctx.params.value;
       return 'Success setting parameter!';
    }
}
```

Na primeru sa slike, akcija dobija json poruku u kojoj je sadržan ***value*** kojim ce da se setuje interval na novu vrednost.

Ovi uredjaji su virtuelni i predstavljeni objektima. Kako bi se “očitavali” podaci sa uređaja, ovom servisu je omogućeno da čita podatke iz datoteke sa postojećim senzorskim podacima. To upravo radi funkcija ***ReadData(),*** simulira očitavanja sa senzora.
Očitani podaci se periodično šalju Data Microservice-u. Funkcija koja obavlja ovaj zadatak postavlja interval za koji ce da uzima predhodno učitane podatke i obavlja slanje. Ova komunikacije se obavlja pomoću NATS broker-a. Postoji event koji je registrovan istim imenom u oba Microservice-a koji će da omogući da kada Device pročita podatak, emituje event i pošto je Data subscribe-ovan, kada mu stigne notifikacije on uzima te podatke.

2. Data Microservice

Dobija podatke sa Device Microservice-a i upisuje ih u InfluxDB (NoSQL) bazu podataka. U projektu je on implementiran kroz fajl services/***data.service.js*** i koristi Moleculer (NodeJS).
Po njegovom imenu može da se pretpostavi koja je njegova svrha, da izvšava akcije/operacije kao sto su upisivanje u bazu podataka ili rad sa query-jima pomoću kojih dolazi do željenih podataka kao što su ***soilTemperature, airTemperature...***

Bitna stvar jeste da on komunicira sa Analytics Microservice-om tako sto salje parametre preko POST zahteva kako bi ih on dalje obradio. Takođe, on je subscribe-ovan na Device Microservice sa kog prima podatke i upisuje u bazu podataka. Ova publish-subscribe komunikacije je napravljena pomoću NATS-a.

3. Analytics Microservice

Svrha ovog servisa je analiza dobijenih podataka od strane Data Microservice-a u cilju detektovanja događaja/stanja značajnih za domen aplikacije. Rezultati ovih analiza se skladiste u InfluxDB koja se nalazi u istom Docker container-u kao i ovaj servis. Servis je napisan u Python-u čime je omogućena proširivost sistema za kopleksnije analize podataka pomoću različitih biblioteka.

U projektu je implementiran, kroz fajl koji se nalazi u analytics/***app.py,*** tako što prima preko POST zahteva od Data Microservice-a podatke koje će kreiranom logikom proveriti i na osnovu rezultata odrediti koju akciju je potrebno preduzeti. Kada prepozna granični slučaj on se upisuje u bazu podataka i šalje odgovarajuci zahtev za izvršenje akcije na Command Microservice-u.
Na primer funkcija ***turnWaterPumpOnOrOff(...):***

- ispituje da li je vlažnost zemljišta povoljna za biljku na osnovu predhodno postavljenih graničnih podataka.

- **vrednosti nisu povoljne**
    - **Suvo zemljište** u tom slučaju uključujemo sistem za navodnjavanje

    - **Zasićeno zemljište** gasimo sistem za navodnjavanje.
- **vrednosti su povoljne** stanje je nepromenjeno

4. **Command Microservice**

Ovaj servis šalje komande Aktuator-u slanjem POST zahteva Device Microservice-u.
U projektu se nalaze vise fajlova koji čine ovaj servis ali najbitnija logika se vidi kroz fajl command/Controllers/***Command.cs*** i koristi ASP.NET Core.

Kako funkcioniše?

Kao što je već napomenuto u objašnjenju Analytics-a, Command prima zahtev od Analytics-a da obavi odredjenu akciju, odredjenu komandu. Na osnovu tipa akcije koju treba da izvrsi on obradi POST zahtev i pošalje komandu na Device Microservice preko Gateway-a.
Komande koje može da prepozna je da uključi i isključi odredjeni aktuator, postavi nivo aktuatora na odredjenu vrednost i moze da modifikuje vrednost tako sto smanji ili poveća za određeni offset. Sve promene koje se dešavaju kao što su npr. povećanje/smanjenje temperature vrše se na osnovu predhodno definisane vrednosti, offset-a.

Korisnik takođe ima mogućnost da pomoću Web dashboard-a upravlja upravo ovim parametrima koji utiču na aktuatore. Ovo je omogućeno PUT zahtevom.

Ovaj servis takodje omogućava da se preko GET-a vidi spisak komandi. Isto preko Get-a mogu da se za svaki aktuator, vide parametri za offset, minimalnu i maksimalnu vrednost.

Bitna stvar je da uvek kada primi zahtev od Analytics-a i kada treba da pošalje komandu (da se postavi na novu vrednost ili da se promeni offset) on ima logiku da proveri da li je vrednost promenjenog parametra u validnim granicama. Da li odgovara postavljenim granicama koje podržavaju uredjaj (min i max). Ako su dobre vrednosti on postavlja komandu, a ako nisu on šalje obaveštenje da vrednosti nisu dobre.

5. **Gateway service**

Preko ovog Microsevice-a se pristupa svim Microservice-ima, omogućava klijentima pristup, rutira zahteve. Ovaj servis u projektu je implementiran kroz fajl koji se nalazi u services/***gateway.service.js,*** koristi Moleculer (NodeJS).

Kako funkcioniše?

Pokrece express server koji “sluša” za http zahteve. Kao što je već pomenuto u ovom servisu se definisu GET, PUT i POST rute. Kada se preko njega pristupi ruti on to preusmerava i poziva odgovarajuci Microservice na osnovu rute koje je pozvana.

```
app.get("/actuatorsStatus", this.getActuatorsStatus); // Device
app.get("/soilTemperature", this.getSoilTemperature); // Data
app.get("/airTemperature", this.getAirTemperature); // Data
app.get("/RHpercent", this.getRHpercent); // Data
app.get("/waterContent", this.getWaterContent); // Data
app.get("/deviceInfo", this.getDeviceInfo); // Data
app.get("/commandList", this.getCommandList); // Command
app.get("/getWaterPumpParameters", this.getWaterPumpParameters); // Command
app.get("/getAirCoolerParameters", this.getAirCoolerParameters); // Command
app.get("/getHumidifierParameters", this.getHumidifierParameters); // Command
app.get("/notifications", this.getNotifications); // Analytics
app.put("/setOffset/waterPump", this.setWaterPumpOffset); // Command
app.put("/setOffset/humidifier", this.setHumidifierOffset); // Command
app.put("/setOffset/airCooler", this.setAirCoolerOffset); // Command
app.put("/setMax/humidifier", this.setHumidifierMax); // Command
app.put("/setMax/waterPump", this.setWaterPumpMax); // Command
app.put("/setMax/airCooler", this.setAirCoolerMax); // Command
app.put("/setMin/humidifier", this.setHumidifierMin); // Command
app.put("/setMin/waterPump", this.setWaterPumpMin); // Command
app.put("/setMin/airCooler", this.setAirCoolerMin); // Command
app.put("/setValue/airCooler", this.setAirCoolerTemperature); // Device
app.put("/setValue/waterPump", this.setWaterPumpLitersPerMinute); // Device
app.put("/setValue/humidifier", this.setHumidifierHumidityLevel); // Device
app.put("/addOffset/airCooler", this.addAirCoolerOffset); // Device
app.put("/addOffset/waterPump", this.addWaterPumpOffset); // Device
app.put("/addOffset/humidifier", this.addHumidifierOffset); // Device
app.put("/changeReadInterval", this.changeReadInterval); // Device
app.put("/turnOnOrOff/waterPump", this.turnWaterPumpOnOrOff); // Device
app.put("/turnOnOrOff/airCooler", this.turnAirCoolingOnOrOff); // Device
app.put("/turnOnOrOff/humidifier", this.turnHumidifierOnOrOff); // Device
app.post("/writeParametersToDatabase", this.writeParametersToDatabase); // Data
```

6. **WEB dashboard**

Omogućava pristup sistemu preko API Gateway-a za prikaz senzorskih podataka, notifikacija/upozorenja na bitne događaje kao i postavljanje parametara. U projektu je implementiran kroz fajl koji se nalazi u dashboard/**app.py,** koristi Python. Kako bi korisnik mogao da prati stanje sistema i po potrebi modifikuje parametre koji mogu da utiču na to, u dashboard-u su dodati grafovi koji prate stanje temperature vazduha, zemljišta i relativne vlažnosti vazduha, tabela koja prati notifikacije kao i forme koje omogućavaju korisniku da modifikuje parametre.

Kako ovo funkcioniše?

Za prikaz grafova i tabele iskorišćena je Bokeh biblioteka.
Podaci za grafove i tabele se dobijaju tako što je definisan Datasource u Bokeh koji periodično šalje GET zahtev na Gateway kako bi prikupio podatke. Sa Gateway-a uzima podatke o očitavanjima i notifikacijama. Očitavanja se preuzimaju iz baze Data Microservice-a preko Gateway-a, a notifikacije iz Analytics Microservice-a preko Gateway-a. Ovaj mikroservis se koristi preko browser-a, graficki se predstavjaju podaci i notifikacije i moguce je pomocu PUT zahteva modifikovati parametre očitavanja (periodu očitavanja podataka) i komandi (modifikacija offseta za uredjaje).

3. Opis REST API-a i topic-a

Koristi se NATS topic između data mikroservisa i device mikroservisa za prenos pročitanih podataka sa device na data mikroservis. Takođe NATS se koristi i kao transporter pri pozivanju funkcija između mikroservisa gateway, data i device zbog toga što su implementirani pomoću Moleculer-a.

Opis REST API-a za mikroservise sledi u nastavku:

http://localhost:3000/writeParametersToDatabase - POST metod za upis podataka u bazu podataka na data mikroservis

http://localhost:3000/turnOnOrOff/humidifier - PUT metod za uključivanje ili isključivanje humidifier aktuatora

http://localhost:3000/turnOnOrOff/airCooler - PUT metod za uključivanje ili isključivanje air cooler aktuatora

http://localhost:3000/turnOnOrOff/waterPump - PUT metod za uključivanje ili isključivanje water pump aktuatora

http://localhost:3000/changeReadInterval - PUT metod za modifikaciju vrednosti intervala na koji se očitavaju podaci u device mikroservisu

http://localhost:3000/addOffset/waterPump - PUT metod za povećavanje ili smanjivanje vrednosti nivoa na kom radi aktuator water pump

http://localhost:3000/addOffset/humidifier - PUT metod za povećavanje ili smanjivanje vrednosti nivoa na kom radi aktuator humidifier

http://localhost:3000/addOffset/airCooler - PUT metod za povećavanje ili smanjivanje vrednosti nivoa na kom radi aktuator air cooler

http://localhost:3000/setValue/airCooler - PUT metod za postavljanje vrednosti nivoa na kom radi aktuator air cooler

http://localhost:3000/setValue/humidifier - PUT metod za postavljanje vrednosti nivoa na kom radi aktuator humidifier

http://localhost:3000/setValue/waterPump - PUT metod za postavljanje vrednosti nivoa na kom radi aktuator water pump

http://localhost:3000/setMin/airCooler - PUT metod za postavljanje minimalne vrednosti parametra komande aktuatora air cooler

http://localhost:3000/setMin/airCooler - PUT metod za postavljanje minimalne vrednosti parametra komande aktuatora air cooler

http://localhost:3000/setMin/humidifier - PUT metod za postavljanje minimalne vrednosti parametra komande aktuatora humidifier

http://localhost:3000/setMin/waterPump - PUT metod za postavljanje minimalne vrednosti parametra komande aktuator water pump

http://localhost:3000/setMax/airCooler - PUT metod za postavljanje maksimalne vrednosti parametra komande aktuatora air cooler

http://localhost:3000/setMax/humidifier - PUT metod za postavljanje maksimalne vrednosti parametra komande aktuatora humidifier

http://localhost:3000/setMax/waterPump - PUT metod za postavljanje maksimalne vrednosti parametra komande aktuator water pump

http://localhost:3000/setOffset/humidifier - PUT metod za postavljanje vrednosti offset parametra komande aktuator humidifier

http://localhost:3000/setOffset/waterPump - PUT metod za postavljanje vrednosti offset parametra komande aktuator water pump

http://localhost:3000/setOffset/airCooler - PUT metod za postavljanje vrednosti offset parametra komande aktuator air cooler

http://localhost:3000/notifications - GET metod za dobijanje vrednosti notifikacija

http://localhost:3000/getHumidifierParameters - GET metod za dobijanje vrednosti parametara komandi humidifier aktuatora

http://localhost:3000/getAirCoolerParameters - GET metod za dobijanje vrednosti parametara komandi air cooler aktuatora

http://localhost:3000/getWaterPumpParameters - GET metod za dobijanje vrednosti parametara komandi water pump aktuatora

http://localhost:3000/commandList - GET metod za dobijanje vrednosti parametara svih komandi

http://localhost:3000/deviceInfo - GET metod za dobijanje vrednosti parametara senzora

http://localhost:3000/soilTemperature?id=1 - GET metod za dobijanje vrednosti očitavanja temperature tla

http://localhost:3000/airTemperature?id=1 - GET metod za dobijanje vrednosti očitavanja temperature vazduha

http://localhost:3000/RHpercent?id=1 - GET metod za dobijanje vrednosti očitavanja vlažnosti vazduha

http://localhost:3000/waterContent?id=1 - GET metod za dobijanje vrednosti očitavanja količine vode u tlu

http://localhost:3000/actuatorsStatus - GET metod za dobijanje vrednosti očitavanja parametara aktuatora

4. CLI komande za testiranje

U nastavku se može videti spisak curl komandi za testiranje REST API-a na gateway mikroservisu kome može pristupati klijent. Curl komande su generisane putem Postman aplikacije.

```
curl --location --request PUT 'http://localhost:3000/turnOnOrOff/humidifier' \

--header 'Content-Type: application/json' \
```

```
--data-raw '{ "sensorId": 1, "value": true }'

curl --location --request PUT 'http://localhost:3000/turnOnOrOff/airCooler' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "value": true }'

curl --location --request PUT 'http://localhost:3000/turnOnOrOff/waterPump' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "value": true }'

curl --location --request PUT 'http://localhost:3000/addOffset/humidifier' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "offset": 5 }'

curl --location --request PUT 'http://localhost:3000/addOffset/waterPump' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "offset": 5 }'

curl --location --request PUT 'http://localhost:3000/addOffset/airCooler' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "offset": 5 }'

curl --location --request PUT 'http://localhost:3000/setValue/humidifier' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "value": 10 }'
```

```
curl --location --request PUT 'http://localhost:3000/setValue/waterPump' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setValue/airCooler' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setMin/airCooler' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setMin/waterPump' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setMin/humidifier' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setMax/airCooler' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'
```

```
curl --location --request PUT 'http://localhost:3000/setMax/waterPump' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setMax/humidifier' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setOffset/airCooler' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setOffset/humidifier' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request PUT 'http://localhost:3000/setOffset/waterPump' \
--header 'Content-Type: application/json' \
--data-raw '{ "value": 10 }'

curl --location --request POST 'http://localhost:3000/writeParametersToDatabase' \
--header 'Content-Type: application/json' \
--data-raw '{ "sensorId": 1, "soilTemperature": 12, "airTemperature": 20, "RHpercent": 70, "waterContent": 0.5, "timeStamp": "20:35:45" }'
```

```
curl --location --request GET 'http://localhost:3000/notifications'

curl --location --request GET 'http://localhost:3000/getHumidifierParameters'

curl --location --request GET 'http://localhost:3000/getAirCoolerParameters'

curl --location --request GET 'http://localhost:3000/getWaterPumpParameters'

curl --location --request GET 'http://localhost:3000/commandList'

curl --location --request GET 'http://localhost:3000/deviceInfo'

curl --location --request GET 'http://localhost:3000/actuatorsStatus'

curl --location --request GET 'http://localhost:3000/soilTemperature?id=1'

curl --location --request GET 'http://localhost:3000/airTemperature?id=1'

curl --location --request GET 'http://localhost:3000/RHpercent?id=1'

curl --location --request GET 'http://localhost:3000/waterContent?id=1'
```