Servisno-orijentisane arhitekture

Upotreba DeviceHive platforme za IoT

Mentor:

Prof. dr Dragan Stojanović

Student:

Miloš Živković 15976

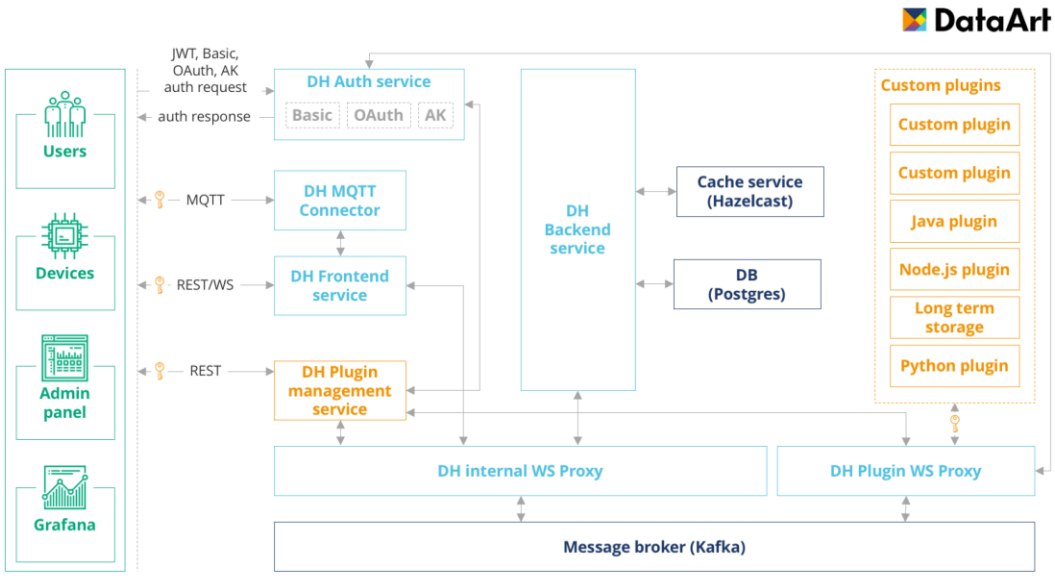
Sadržaj

1. Uvod

DeviceHive je IoT platforma koja služi za integraciju uređaja putem MQTT-a, WebSocket-a, ili REST API-ja. Sama platforma je „device-agnostic“, to jest DeviceHive nudi postojeće klijentske biblioteke napisane u JS-u, na Javi, Python-u, GoLang-u, takođe postoje biblioteke razvijene u iOS i Android tehnologiji. Deployment je moguće izvršiti pomoću Kubernetes-a ili Docker-a, tako da skaliranje ne zahteva previše truda. Osnova za analitiku podržana je kroz softverska rešenja kao što su: ElasticSearch, Apache Spark, Cassandra i Kafka, za razvoj demo aplikacije je iskorišćena funkcionalnost Apache Kafka servisa. Platforma je open source, implementacija i arhitektura je dostupna na zvaničnom GitHub-u.

2.Arhitektura

Platforma je razvijena kao mikro servisni sistem sa dobrom podrškom za skaliranje i dostupnost samog sistema. Sama arhitektura je projektovana tako da može da podrži na stotine uređaja uz odgovarajuću bezbednost samih podataka i platforme.



Slika 0. Arhitektura platforme

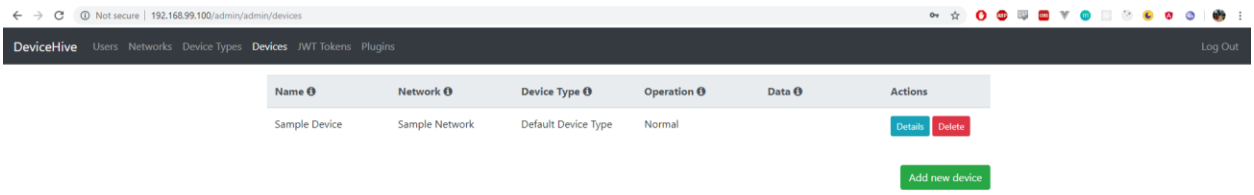
3. Docker Deployment

Platformu je moguće pokrenuti lokalno uz pomoć Docker kontejnera. Za potrebe razvoja demo aplikacije dodat je i plugin servis. Kontejneri su kreirani na osnovu komande:

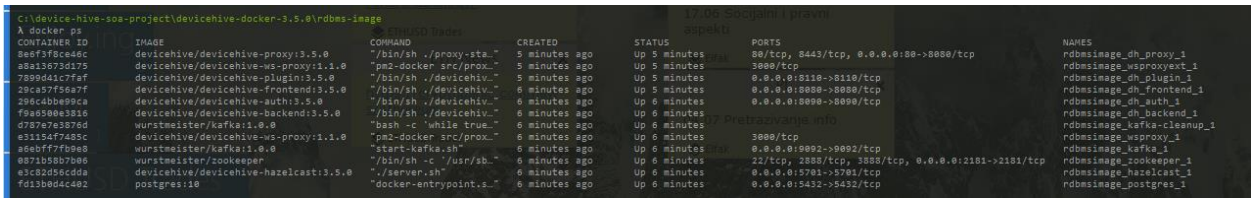
```
docker-compose -f docker-compose.yml -f dh_plugin.yml up -d
```

Izvršenje ove komande postavlja okruženje koje se može podešavati kroz GUI koji se nalazi na *docker-machine-ip*/admin. Pored admin konzole pokrenuti su i sledeći servisi:

- Frontend Service API
- Auth Service API
- Plugin Service API
- Kafka



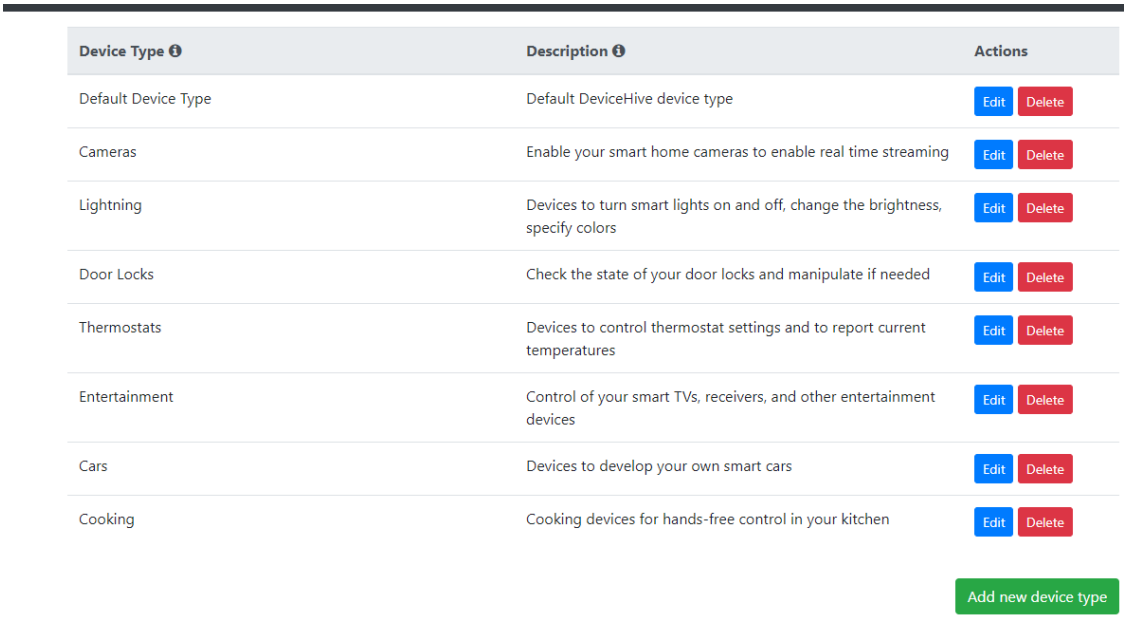
Slika 1. Izgled admin konzole



Slika 1.1. DeviceHive Docker kontejneri

4.Admin GUI

U okviru korisničkog interfejsa DeviceHive platforme moguće je izvršiti osnovne CRUD operacije nad korisnicima, mrežama, tipovima uređaja, uređajima, JWT tokenima, i u slučaju da je pokrenut plugin servis i plugin-a.



Slika 2. Tipovi podržanih uređaja

Za korisnike je moguće dodati kojoj mreži pripadaju i odobriti im pristup istoj. U sekciji za JWT je moguće kreirati tokene koji će poslužiti kao autentifikacija klijenata koji pristupaju DeviceHive platformi. Za kreiranje uređaja je neophodno dodati kog je tipa uređaj i kojoj mreži pripada.

5. Kafka

Kafka je distribuirana streaming platforma, ovim je omogućeno da se vrši publish/subscribe na određene tokove podataka. Load balancing i komunikaciju između servisa podržava Apache Kafka. Kafka podržava kreiranje topic-a, na topic-e se mogu prijaviti više korisnika sa ciljem konzumiranja podataka. Kafka rešava perzistenciju podataka pomoću message buffer-a, koji se čuvaju na određeni vremenski period. Za čišćenje i monitoring topic-a je integrisan Apache ZooKeeper. Za potrebe demo aplikacije je iskorišćen plugin menadžer mikroservis, koji omogućava slanje komandi i notifikacija u Kafka topic. Komponente servisa su:

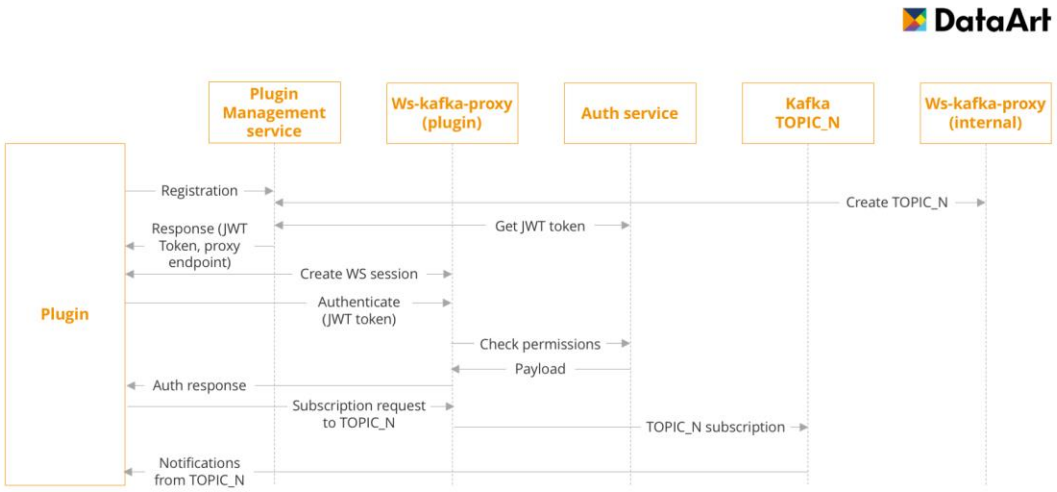
- Plugin Menadžer UI
- Auth servis
- WebSocket Kafka proxy
- Kafka topic

WebSocket Kafka proxy je NodeJS servis koji služi kao omotač za brokerisanje poruka, to jest komunikaciju sa Kafkom preko WebSocket-a. Plugin služi kao komunikator sa Kafka proxy-ijem, trenutni zvanični templejti za razvoj plugin-a su:

- Java template
- NodeJS plugin core
- Python library

6.Bezbednost

Autentifikacija plugin-a se vrši pomoću JWT tokena, prilikom kreiranja plugin-a dobija se par access i refresh token kao i id topic-a na koji je moguće konektovati plugin. Za odgovorajući plugin je moguće podesiti mrežu, uređaj i tip uređaja. Na slici je dat prikaz arhitekture plugin menadžer mikroservis-a, demo aplikacija predstavlja **plugin** dok je **internal** omotač za Kafka sistem.



Slika 3. Proces autentifikacije plugin-a

7. Specifičan NodeJS Plugin

Demo aplikacija je napravljena na osnovu zvaničnog šablona sa zvaničnog sajta za NodeJS. Konekcija ka DeviceHive platformi se vrši preko WebSocket-a, i zato postoji odgovarajuća konfiguracija u config fajlu. U konfiguraciji je bitno navesti endpoint ka platformi koji je u obliku:

```
ws://192.168.99.100/plugin/proxy
```

Konfiguracija između ostalog sadrži i URL servisa za autentifikaciju, ime topic-a na koji se plugin treba prijaviti, tokene za korisnika i tokene za plugin. Za razvoj glavne aplikacije je korišćena biblioteka devicehive-plugin-core, pomoćna biblioteka za interfejs poruka devicehive-proxy-message i fetch API za NodeJS zbog slanja GET zahteva.

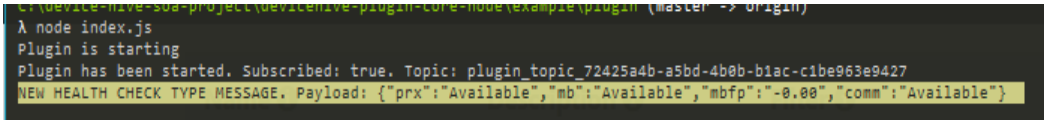
Za razvoj plugina je proširena klasa DeviceHivePlugin koja pomoću statičke metode start uz prosleđenu modifikovanu klasu i konfiguraciju. Metode plugina koje ne treba predefinisati:

```
sendMessage(message) {}

subscribe(subscriptionGroup) {}

unsubscribe() {}
```

U specifičnom plugin-u je dodata metoda handleMessage koja služi da obradi poruke različitog tipa koje mogu stići iz bafera poruka određenog topic-a. Tip poruka koje obrađujemo u okviru plugin-a jesu health i notification tip poruka.



Slika 4. Health tip poruke

Health tip poruke sadrži:

- **prx** – proxy status
- **mb** – message buffer status
- **mbfp** – procenat popunjenosti message buffer
- **comm** – status brokera poruka

Notification tip poruke se kreira pomoću message builder-a koji je pomoćna biblioteka i priprema tip poruke koji je prihvaćen od strane proksija i za normalizaciju poruke kad ona dođe do klijenta.

```
setInterval(() => {
            fetch(api.API_URL)
                .then(res => res.json())
                .then((messages) => {
                    const notificationMsg =
MessageBuilder.createNotification(
                        {
                            topic: me.topic,
                            message: JSON.stringify({
                                b: { notification: messages }
                            }),
                            partition: 1,
                            type: "latest"
                        }, 44);
                    service.sendMessage(notificationMsg);
                })
        }, minutePassed);
```