Měřící systémy
IoT HUB

Petr Dvořáček

27. května 2020

Abstrakt

Díky velké škále možnosti použití IoT zařízení lidstvo získává obrovské toky dat. Vzniká potřeba tyto toky dat dostat z periferie (IoT zařízení) na cloud. Tato práce popisuje způsob jakým lze zpracovat data z rozdílných IoT zařízení s odlišným komunikačním rozhraním. Cílem je uložit data ve formátu čitelném pro člověka do databáze a vytvoření vizualizace.

1 Úvod

V této sekci bude rozebrán úvod do problematiky a technologie použité pro řešení problému.

IoT zařízení mohou po síti komunikovat s využitím různých protokol (HTTP, MQTT, UDP) a to buďto přes bránu (IoT Gateway), anebo jako napřímo [1]. Cílem této práce je vytvořit IoT HUB který bude přes odpovídající adaptéry zpracovávat odpovídající protokoly a informace následně uloží v jednotném formátu do databáze.

Celé softwarové řešení se dá rozložit na 3 samostatné aplikace

1. BE – Backend přijímá data z vnějšího prostředí, vkládá do DB.
2. DB – Databáze přijímá a uchovává data z BE.
3. FE – Frontend slouží jako vizualizace pro uživatele.

2 Rozbor technologií

V této sekci budou detailně rozebrány technlogie použité pro realizaci projektu. Jedná se technologii Node.js, Grafanu, InfluxDB a Docker.

[1] Gateway je v takovém případě součástí IoT senzoru

2.1 BE

Pro realizování BE byla zvolena technologie Node.js [**?**] postavená na JavaScriptového běhového prostředí V8 [**?**]. Tato technologie byla zvolena zejména pro efektivnost implementace [2]. Node.js vyniká ve zvládání vysokého zatížení díky možnstí vytváření asynchronnich (neblokujících) funkcí jednoduchým a pro JavaScript jednoduchým způsobem[**?**].

Multithreading Při spuštění funkce v jiném vlákně není blokováno hlavní vlákno programu. V případě že ve funkci je blokující operace (IO zápis do databáze, souborového systému počítače) tak toto vlákno stojí a čeká – toto je nehospodárné. Ve vláknu by měla běžen výpočetně náročná operace, nikoliv IO blokující operace.

Async Při asynchronním spuštění funkce není vytvářeno nové vlákno, namísto toho je zavoláno z V8 api (Prohlížeče, nebo operačního systému) a přes toto api jsou vykonány požadované funkce. Od zavolání api je funkce ukončena a Event Loop JavaScriptu pokračuje ve vykonávání dalších funkcí. Jakmile je funkce zavolaná z api dokončena, je její výsledek vložen do Callback Queue [**?**]. V této technologii je zřejmý návrhový model producenta a kunzumenta, kde konzumentem je Call Stack a producentem je api, přičemž data si předávaji přes Callback Queue.

Konkrétně pro vytvoření webového serveru byla použita minimalistická knihovna `express` pro svou jednoduchost a flexibilnost[**?**]

2.2 DB

Databázová technologie InfluxDB [**?**] byla použita pro svůj real-tile charakter, otevřený zdrojový kód a pro fakt že se jedná o technologii primárně určenou ke zpracovávání časových řad (což je také výstup ze senzoru).

V kontextu technologie SQL lze chápat jeden senzor v databázi InfluxDB jako SQL tabulku, kde hlavní index (klíč) je čas a tagy a hodnoty představují sloupce.

Rozdíl oproti technologii je ten, že v InfluxDB je přiřozené mít miliony senzorů (tedy v kontextu SQL tabulek)

Syntaxe jazyka použitého pro ovládání databáze je velmi podobná jazyku SQL.

Pro manipulaci s databází z JavaScriptu lze využít knihovny `influx`. Ta nám umožňuje přidat jakákoliv data přes formát JSON.

Pro propojení influx databáze byla použita knihovna `influx` [**?**].

2.3 FE

Pro vizualizaci byla použita technologie Grafana zejména pro jednoduchost použití a flexibilnost. Uživateli umožňuje zobrazit jakoukoliv řadu z databáze

[2] Implementace BE není složitá a zároveň je flexibilní a výsledný server je výkonný

pomocí Query příkazů.

2.4 Propojení celků

Pro propojení jednotlivých funkčních celků byl použit nástroj `docker-compose`. Tento nástroj vytvoří izolovanou síť pro dané kontejnery a zároveň vybrané porty tuneluje do hostujícího stroje. Výhodou tohoto přístupu je fakt, že veškerá propojení, čísla portů, závislosti služeb jsou definované v textu (viz. 1). Text je poté možno jednoduše verzovat a debugovat.

3 Vypracování

Nejdříve se zaměřmě na vývojové prostředí ve kterém bude celá aplikace tvořena a lazena. Rozhodl jsem se celou aplikaci již odpočátku vyvýjet v docker síti[3]. To znamená že celé prostředí aplikace je odděleno od mého operačního systému, zároveň je veškerá infrastruktura zachycená v kódů[4]. Tento přístup je výhodný zejména v reprodukcibilitě prostředí (celé prostředí se všemi balíčky v kódu) a pozdějším nasazení – na produkčním serveru stačí aby byl nainstalován docker.

Infrastruktura Klíčový soubor `docker-compose` který představuje infrastrukturu prostředí je na fig. 1. Je z něj zřejmé, že aplikace bude složena ze tří služeb, `influxdb`, `grafana` a `server_http`. Služba `server_http` je založena na souboru `Dockerfile` který je na fig. 2. Dále stojí za zmínění parametr `volumes` služby `server_http` – tento nám napojí vývojové prostředí na lokálním počítači do kontejneru. Budeme tedy schopni server aktualizovat "za pochodu"při psaní kódu na lokálním stroji.

Server Implementace serveru je velmi jednoduchá, přijme `POST` který musí mít `body` ve stanoveném formátu a data v objektu `fields` uloží do databáze. Objekt `fields` nesmí obsahovat žádné podobjekty! Pouze klíče a triviální hodnoty (string, int, float ...) viz Fig. 3

Vizualizace Frontend aplikace je postaven na Grafaně. Služba Grafany běží na portu 3000, stačí se připojit na VPN školní sítě a jít na adresu `rc111.vsb.cz:3000`, uživatel a heslo by měl být *admin*.

[3] Respektive v docker conteinerech propojených pomocí docker-compose.

[4] Hlavním důvodem proč jsem se tak rozhodl je ušetření místa na disku. Cele vývojové prostředí mám nadefinované v kódu který zabírá zlomek místa. V době kdy vyvíjím dané prostředí postavím (veškeré balíčky se nainstalují) a po skončení programování jej můžu zase smazat (balíčky se smažou, nezabírají paměť). Jedná se o "good practice".

```
version: "3"
services:
  influxdb:
    image: influxdb:latest
    container_name: influxdb
    ports:
      - "8083:8083"
      - "8086:8086"
      - "8090:8090"
    env_file:
      - "env.influxdb"
    volumes:
      - ./data_influxdb:/var/lib/influxdb

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"
    env_file:
      - "env.grafana"
    user: "0"
    links:
      - influxdb
    volumes:
      - ./data_grafana/:/var/lib/grafana

  server_http:
    build: ./server_http/
    env_file:
      - .env
    ports:
      - "$HTTP_IP:$HTTP_PORT:80" #localip:localport:containerport
    links:
      - influxdb
    volumes:
      - "./server_http/src/:/server_http/src"
```

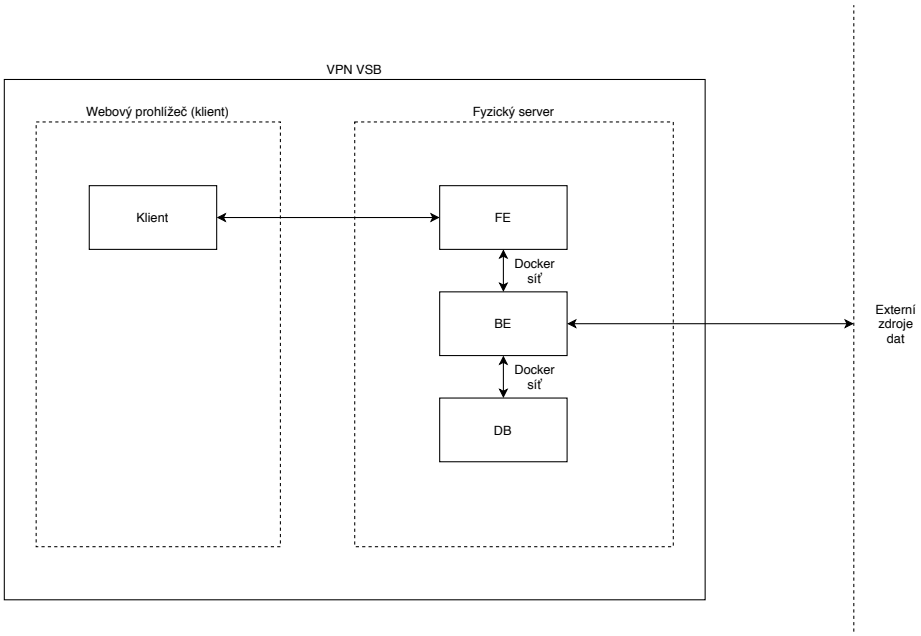
Obrázek 1: Infrastruktura vývojového a běhového prostředí zachycena syntaxí `docker-compose`.

```
FROM node:10

WORKDIR /server_http/

COPY package.json ./

RUN npm install
# RUN npm ci --only=production
RUN npm install supervisor -g

COPY ./src/ ./src/

EXPOSE 5000

CMD [ "supervisor", "./src/server.js" ]
# CMD [ "tail", "-f", "/dev/null" ]
```

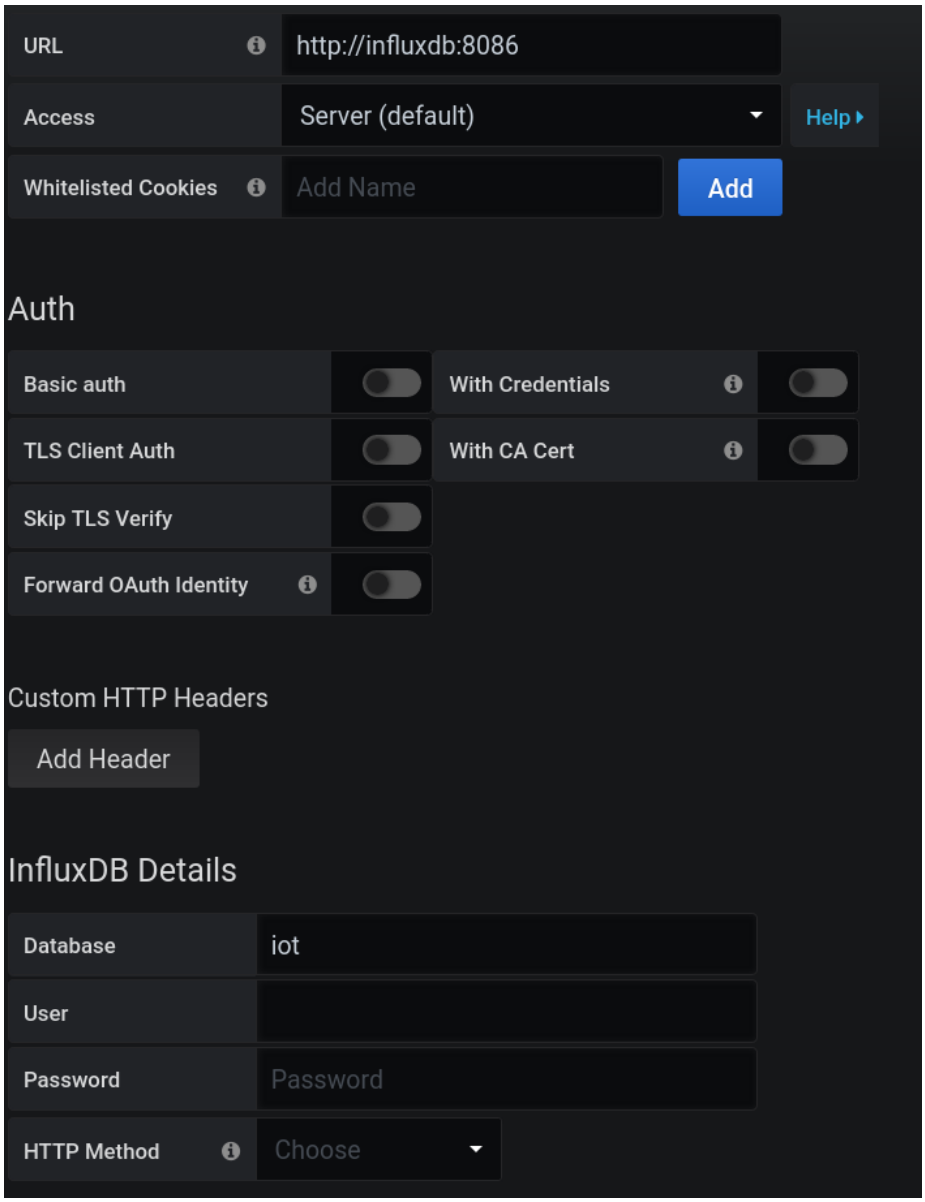
Obrázek 2: Infrastruktura běhového prostředí služby `server` vyjádřena syntaxí `Dockerfile`. Tato je dále propojena s ostatními službami pomocí `docker-compose`

```
    {
    "measurement": "id_zarizeni_test",
    "fields":{
        "teplota": 10,
        "vlhkost": 10.12,
        "cas_mereni": 100918283
    }

{
    "measurement": "id_zarizeni_test",
    "fields":{
        "teplota": 13,
        "vlhkost": 10.16,
        "cas_mereni": 100918403,
        "necekany_objekt": {
            "oh": 1,
            "outstanding move": -1
        }
    }
}
```

Obrázek 3: Nahoře: validní body HTTP POST které do databáze zapíše hodnoty klíče `fields`. Dole: Nevalidní zapsání `body` ve kterém objekt `fields` obsahuje klíč s anonymním objektem (tento nelze uložit do DB)



Obrázek 4: Diagram propojení jednotlivých komponent aplikace.



Obrázek 5: Funkční konfigurace zdroje dat v Grafaně.