FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Doporučovací systém pro chytré domácnosti
Student:	Bc. Cyril Urban
Vedoucí:	Ing. Radim Štěpaník
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem diplomové práce je vytvoření nezávislé služby pro systémy domácí automatizace, která dokáže na základě dat doporučovat nastavení chytré domácnosti.

• Nastudujte a popište existující řešení a standardy.

• Navrhněte stabilní škálovatelnou architekturu pomocí AWS serverless technologií.

• Navrhněte procesy zpracování vstupních dat a zpracujte datový model. Připravená data budou sloužit pro vytváření doporučení.

• Navrhněte alespoň jeden scénář, jehož funkčnost ověříte implementací prototypu.

• Implementujte službu, která dokáže zpracovávat vstupní data a poskytovat doporučení nastavení chytré domácnosti.

• Připravte testovací data a ověřte funkcionalitu řešení.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 1. září 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Doporučovací systém pro chytré domácnosti

Bc. Cyril Urban

Katedra softwarového inženýrství
Vedoucí práce: Ing. Radim Štěpaník

29. dubna 2021

Poděkování

Rád bych poděkoval vedoucímu této práce, panu Ing. Radimovi Štěpaníkovi za skvělé vedení práce, plné soustavné konstruktivní kritiky, která vedla k úspěšnému cíli. Dále bych chtěl poděkovat paní Ing. Magdě Friedjungové za konzultaci algoritmů strojového učení.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. dubna 2021 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Urban, Cyril. *Doporučovací systém pro chytré domácnosti.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato diplomová práce se zabývá vytvořením nezávislé služby pro systémy domácí automatizace, která dokáže na základě IoT senzorických dat doporučovat nastavení chytré domácnosti. Hlavním účelem této služby je poskytovat aplikační rozhraní pro příjem IoT dat, periodicky spouštět algoritmy pro tvorbu doporučení a tyto doporučení poskytovat. Výpočet doporučení se opírá o algoritmy shlukové analýzy. V systému je kladen důraz na stabilitu a škálovatelnost s využitím AWS serverless technologií. Výsledkem je funkční prototyp.

Klíčová slova Doporučovací systém, Strojové učení, Shluková analýza, AWS, Serverless, IoT, Node.js, TypeScript, Infrastruktura jako kód, DynamoDB, Lambda, Step Functions, CDK

Abstract

This Master's Thesis deals with the creation of an independent service for home automation systems that can recommend smart home settings based on IoT sensor data. The main purpose of this service is to provide an application interface for receiving IoT data, periodically run algorithms for creating recommendations, and provide these recommendations. The calculation of

recommendations is based on cluster analysis algorithms. The system emphasizes stability and scalability using AWS serverless technologies. The result is a functional prototype.

Keywords Recommender system, Machine learning, Cluster analysis, AWS, Serverless, IoT, Node.js, TypeScript, Infrastructure as code, DynamoDB, Lambda, Step Functions, CDK

Obsah

Seznam obrázků

Úvod

Myšlenka vytvořit chytrou domácnost, která lidem pomáhá v jejich každodenním životě je už poměrně stará. Samotný pojem chytré domácnosti se stal populárním již v 80. letech 20. století [1]. Od té doby zájem lidí o chytrou domácnost každým rokem stoupá a masivní nárůst nás v budoucnu nejspíše teprve čeká [2].

Pod pojmem chytrá domácnost si lze představit domácnost, kterou je možné monitorovat a řídit v ní zařízení pro ovládání osvětlení, žaluzií, teploty, vlhkosti a tak dále. Nicméně aby domácnost začala být doslovně chytrá je potřeba, aby se sama přizpůsobovala chování uživatele. Tedy, aby sledovala jeho návyky a tyto návyky mu pomohla zautomatizovat. Například pokud uživatel každý všední den vstává v 7:00 a po probuzení si vytahuje žaluzie, zapíná si rádio, kávovar a rozsvěcuje si v kuchyni, tak uživateli tyto činnosti zautomatizujme. Nebo pokud při západu slunce pravidelně zatahuje žaluzie, rozsvěcuje si lampu a zapíná si televizi není problém ani tyto činnosti dělat automaticky. Kromě usnadnění života automatizace vede k lépe energeticky použitelným domům.

V rámci této práce se budu zabývat návrhem webové služby, která bude schopna přijímat informace o chování uživatele chytré domácnosti a na základě navržených algoritmů tvořit doporučení na automatizaci činností, které bude služba poskytovat přes aplikační rozhraní.

Jedny z klíčových faktorů webové služby, bude její stabilita a škálovatelnost, které bude zajištěno pomocí cloudových technologií. Konkrétně pomocí AWS serverless technologií.

V této práci bude definován cíl práce a testovací data. Dále bude zanalyzována problematika doporučovacích systémů a cloudových AWS technologií. Následně bude proveden návrh na řešení realizace služby, která by splňovala dané charakteristiky. Poté budou prezentovány problémy a zjištěné skutečnosti, které vznikly při samostané realizaci navrženého prototypu. Další kapitola bude věnována testování a cenovému zhodnocení provozu služby.

V předposlední části bude diskutován potenciální další rozvoj a integrace systému. Poslední kapitola patří závěru.

Kapitola 1

Cíl práce

Cílem této diplomové práce je vytvoření nezávislé služby pro systémy domácí automatizace, která dokáže na základě IoT senzorických dat doporučovat nastavení chytré domácnosti.

Úkolem bude nastudovat, vysvětlit a popsat existující řešení a standardy. Následně navrhnout a implementovat službu, která dokáže přijímat IoT senzorická data, zpracovat je a vytvořit z nich doporučení na nastavení chytré domácnosti. Tyto doporučení bude služba poskytovat přes API. Při implementaci webové služby bude kladen důraz na stabilitu a škálovatelnost pomocí AWS serverless technologií.

Dále bude úkolem navrhnout a implementovat alespoň jeden scénář, který více zkonkretizuje doporučení.

Systém pomocí testovacích dat ověří funkcionalitu řešení.

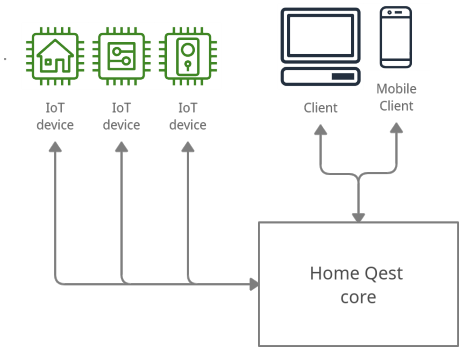
Kapitola 2

Testovací data

Za účelem analýzy, návrhu a implementace prototypu bylo nutné získat testovací data, na kterých budou demonstrovány použité myšlenky. Testovací data byla poskytnuta společností Qest automation s.r.o., která vyvíjí mimojiné software pro domácí automatizaci s názvem Home Qest.

Home Qest má v produkčním prostředí v provozu zhruba 20 aktivních domácností, z nichž každá používá několik chytrých zařízení, která může uživatel ovládat nebo si jejich ovládání automatizovat.

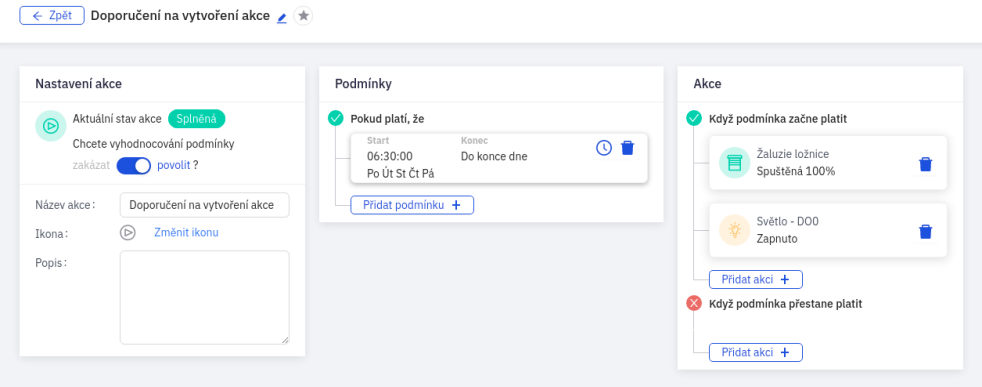
Události o manuálním ovládání jednotlivých zařízení jsou logovány. Cílem bude právě na základě těchto logů najít opakující se vzor a uživateli doporučit tyto činnosti zautomatizovat.



Obrázek 2.1: Architektura (zjednodušená) systému Home Qest

2.1 Home Qest systém

Home Qest je systém, který umožňuje ovládat připojená chytrá zařízení pomocí webové či mobilní aplikace. Pod chytrými zařízeními si lze představit například žaluzie, světla, termostat, klimatizace a tak dále. Zařízení



Obrázek 2.2: Snímek obrazovky z webové aplikace Home Qest - nastavení podmínkové akce

může uživatel ovládat buď manuálně (jednorázově), kupříkladu manuálně vytáhnout žaluzie anebo si může některé činnosti zautomatizovat, například každý den v 8:00 vytáhnout žaluzie.

Při automatizaci činností uživatel definuje časovou podmínku, jakých zařízení se to má týkat a do jakých stavů se daná zařízení mají aktivovat. V časové podmínce lze definovat dny v týdnu, ve kterých se událost má stát a k tomu jeden z následujících údajů:

- čas v rámci dne (například 8:00)
- při východu slunce
- při západu slunce slunce

Podmínka podle východu/západu slunce funguje jen pro domy, které mají v systému nastavenou adresu. Podle adresy systém zjišťuje, kdy je u daného domu východ/západ slunce.

2.2 Logy o uživatelských událostech

Když uživatelé udělají v domě nějakou manuální akci, tak se vytvoří log[1], který se ukládá v rámci Home Qest core (2.1). Z logu lze vyčíst informace o tom, kde byl log pořízen (dům, případně patro a místnost), jakého zařízení se to týkalo (jeho unikátní ID, typ a jméno), v jakém čase byla akce vykonána a jakého stavu dané zařízení nabývalo. Hodnota stavu je různého formátu pro různé typy zařízení.

[1]Log se tvoří i za předpokladu, že je akce zautomatizovaná, to je ale atribut *ManualUntil* ve stavu null. Lze tedy snadno oddělit logy, které byly vytvořeny manuálně.

```
{
    "_id": "5c98b7cb5b4a330001521a29",
    "HomeId": "5c98b3115b4a330001521a27",
    "FloorId": "5c4f2704494bdc0001173ea2",
    "RoomId": "5c4f2735494bdc0001173ea5",
    "DeviceId": "5c503b194f89eb0001094fa5",
    "DeviceType": "windowBlind",
    "DeviceName": "Zaluzie 1",
    "State": {
        "_t": "WindowBlindState",
        "Position": {
            "Shift": 0,
            "Tilt": 100
        },
        "ManualUntil": "2200-01-01T00:00:00.000Z"
    },
    "CreationUtc": "2019-03-25T11:13:15.144Z"
}
```

Zdrojový kód 2.1: Ukázka logu uživateké manuální akce

Logy jsou ukládány do MongoDB kolekce *DeviceRecords*. Pro účely analýzy a vývoje prototypu byl poskytnut dump (historická kopie) databáze.

2.3 Údaje o chytrých domech

Data, která jsou k dispozici jsou anonymizovaná, nicméně je k dispozici ještě kolekci *Homes*, u které znám kromě identifikátoru jediný parametr a to je PSČ.

2.4 Motivace pro řešení

Webová služba, která vzniká v rámci této diplomové práce má za cíl dokázat přijímat přes API logy o manuálních aktivitách uživatele v chytrém domě a právě na jejich základě vypočítat doporučení, které bude v podstatě podklad pro vytvoření automatizované, podmínkové akce.

Výsledné doporučení by tak ideálně mělo obsahovat informace o **časových údajích** (případně východu/západu slunce), **dnech v týdnu** a **seznamu zařízení se stavy** do kterých se má akce aktivovat.

Tato doporučení by se měla periodicky přepočítávat a být poskytována přes API. API pak v budoucnu může konzumovat systém Home Qest core, který by uživateli v rámci aplikace navrhl doporučení na vytvoření podmínkové akce, tedy na automatizaci jeho činností. Alternativně by mohla být doporučení rovnou automaticky aplikována. Tak jako tak, by se dům po integraci přizpůsoboval chování uživatele. To by vedlo na vyšší komfort a nižší spotřebu energie domu.

Kapitola 3

Analýza

Definovaný problém a jeho požadované řešení je to, že chceme v datech, které máme na vstupu uložené v databázi (později přijímané přes API) najít nějaký opakující se vzor, respektive novou znalost. Tato znalost bude sloužit jako doporučení. Z toho je rychle zřejmé, že tento typ úlohy dobře zapadá do oblasti **získávání znalostí z databází**. V první části analýzy bude blíže popsána tato problematika.

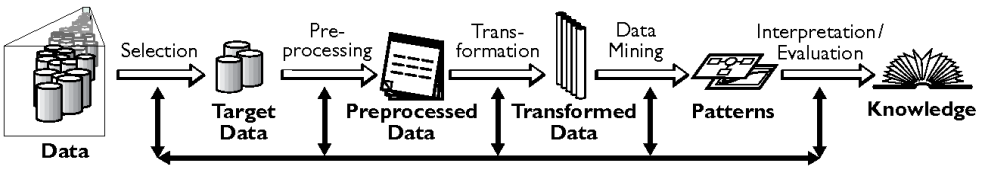
Nestačí pouze zanalyzovat algoritmy, které povedou k vyřešení problému. Další velkou problematikou je samotná architektura systému, tedy to v jakém prostředí budou algoritmy pracovat a jak bude zaručena stabilita a škálovatelnost systému. Případně jak provést návrh architektury, aby cena provozu byla co nejnižší? Přesně na tyto otázky bude odpovídat druhá část analýzy. Ze zadání plyne, že by na tento problém měly být použity AWS serverless **cloudové technologie**. Bude zde také provedena analýza toho, jestli je to správná volba a jaké to přináší výhody a nevýhody.

3.1 Získávání znalostí z databází

Získávání znalostí z databází (KDD) vzniklo v 90. letech po nárůstu rozsáhlých a všudypřítomných databází [3]. Vzhledem k tomu, že toto rostoucí množství dat mělo ve své surové podobě velmi malou hodnotu a stalo se nezvládnutelné pro tehdejší techniky analýzy dat, byly potřebné nové metody pro získávání znalostí.

Původní proces KDD je definován jako netriviální proces identifikace platných, nových, potenciálně užitečných a nakonec srozumitelných vzorců v datech [4]. Samotná data tedy nejsou znalostmi, ale po identifikaci těchto vzorců, informací z dat je lze interpretovat do znalostí.

Pojem získávání nebo někdy také dolování dat je často chápáno jako označení celého procesu získávání znalostí. Jindy je ale chápán pouze jako jeden z kroků v tomto procesu. Tento proces zobrazuje obrázek 3.1.



Obrázek 3.1: Proces získávání znalostí z databází (převzato z [5])

Jak lze na obrázku 3.1 pozorovat, získávání znalostí z dat je posloupnost několika kroků, které na sebe postupně navazují, mezi tyto kroky lze zahrnout:

- **Selekce dat** (Selection) značí volbu pouze těch dat, která jsou pro danou dolovací úlohu vhodná a relevantní.
- **Předzpracování dat** (Pre-processing) označuje část procesu, ve které jsou data vyčištěna (odstraněna z nich chybějící, zašuměná nebo odlehlá data). Do tohoto kroku je taká zahrnuta integrace dat, která se řeší v případě, že máme více datových zdrojů a potřebujeme je sjednotit (v tomto kroku je někdy nezbytné vyřešit nekonzistenci dat, která vznikla sjednocením).
- **Transformace dat** (Transformation) definuje převod dat do formátu, který je vhodný pro danou dolovací úlohu. Mezi transformace patří normalizace, vyhlazování, agregace, generalizace či konstrukce nových atributů (například z adresy může být vytvořena zeměpisná délka a šířka).
- **Dolování z dat** (Data Mining) je klíčový proces, v němž nastává převod předzpracovaných a transformovaných dat do potenciálně užitečných vzorů. K tvorbě vzorů můžou posloužit metody jako shluková analýza, klasifikace, analýza asociací a další metody.
- **Interpretace a hodnocení vzorů** (Interpretation and Evaluations) značí proces výběru a interpretace těch vzorů, které jsou zajímavé a tím tak reprezentují znalost.
- **Znalost a její prezentace** (Knowledge) je poslední část procesu, která danou znalost vhodně prezentuje a poskytuje uživatelům.

Proces získávání znalostí z databází může být iterativní proces, ve kterém lze nalezené vzory znovu zapojit do začátku procesu, zdokonalit těžbu, integrovat a transformovat nová data, abychom získali odlišné a vhodnější výsledky.

3.1.1 Dělení dolovacích metod

V této části bude popsáno a přiblíženo několik hlavních typů dolovacích metod v oblasti získávání znalostí z databází.

3.1.1.1 Vzorkování

Vzorkování není přímo algoritmem řešící nějakou zadanou dolovací úlohu, ale je to jedna ze základních technik dolování dat, která umožňuje dostat výsledek ve velmi rychlém čase.

Vzorkování je založeno na výběru omezené množiny dat ze vstupních dat. Nejjednodušším způsobem vzorkování je náhodný výběr, jehož účelem je jen zmenšení objemu zpracovávaných dat a tím tak vede k urychlení výpočtů. Složitější metody vzorkování jako je například výběr stejného počtu záznamů daného typu, umožňují redukci počtu dat s minimální ztrátou požadované přesnosti výsledku.

Vzorkování může být použito v kombinaci s některou z dále zmíněných metod.

3.1.1.2 Analýza asociačních pravidel

Nejběžnějším použitím analýzy asociačních pravidel a zároveň jejím ilustrativním příkladem, je tzv. analýza nákupního košíku. Ta analyzuje kombinaci produktů, které se ve vstupních datech, tedy v nákupním košíku zákazníků, objevují významně častěji. Odhalení takových kombinací pomáhá například marketingovému oddělení v organizování nabídky či tvorbě balíčků, které obsahují sadu produktů.

3.1.1.3 Klasifikace a predikce

Obecně je klasifikace metodou pro dělení dat do souboru skupin podle jistých kritérií. Klasifikační kritéria jsou předem známa, alespoň pro část dat. Pomocí prediktivního modelování je vyvinut model jehož výstupem je klasifikační proměnná. Příkladem může být klasifikace toho jestli je e-mail nevyžádaná pošta nebo ne. Dalším příkladem může být přiřazení diagnózy danému pacientovi na základě pozorovaných charakteristik pacienta (pohlaví, krevní tlak, přítomnost nebo nepřítomnost určitých příznaků atd.). Klasifikace je příkladem rozpoznávání vzorů.

Klasifikace bývá spojována také s predikcí, s kterou nese mnoho společných rysů. Rozdíl mezi klasifikací a predikcí je ten, že klasifikace zařazuje data do některé z tříd, které jsou předem definované. Kdežto predikce určuje hodnotu, která je spojitá. Tato hodnota je nejpravděpodobnější výstup pro předem neznámé kombinace vstupních hodnot.

Data pro klasifikaci a predikci bývají typicky rozdělena do tří množin. První množina je **trénovací množina**, tam se algoritmus učí klasifikovat. Další množina je **testovací množina**, která slouží pro ověření úspěšnosti. Na zbylou, poslední množinu metoda aplikuje své naučené dovednosti klasifikace. Klasifikace často používá metody založené na **rozhodovacím stromu**, **regresi** nebo na **neuronových sítích** [6].

3.1.1.4 Shluková analýza

Shluková analýza **nezná předem definovaný soubor tříd** a úkolem je právě takové třídy nalézt a vytvořit. Příkladem může být nalezení skupin prodejen na základě jejich druhu zboží, obratu či typu zákazníků. Nalezené skupiny lze pak použít například pro specifikaci marketingové kampaně zaměřené na jednotlivé skupiny prodejen.

Shluková analýza má několik typů dělení, kde každý typ chápe shluky trochu jiným způsobem. Pro každou z nich je ale podstatné, aby podobnost jednotlivých objektů v rámci shluku byla co největší. Podrobněji se bude shlukové analýze věnovat kapitola 3.1.2.

3.1.2 Shluková analýza

Jak již bylo nastíněno v minulé podkapitole, cílem shlukové analýzy je seskupit konečnou množinu objektů do konečné a diskrétní množiny datových struktur, takovým způsobem, že objekty ve stejné výsledné množině (nazývané shluk) jsou si navzájem více podobné než v jiných množinách (shlucích) [7].

Na vstupu metody nejsou žádné popsané třídy, do kterých bychom jednotlivé objekty mohli zařadit. Právě tvorba nových tříd (shluků) je předmětem shlukové analýzy.

3.1.2.1 Datová matice

Datová matice slouží k reprezentaci vstupní datové množiny. Tato 2D matice má rozměry **n x p**, kde **n** značí počet objektů a **p** počet dimenzí (zkoumaných atributů, které budou hrát roli v hledání podobnosti).

$$\begin{bmatrix} x_{11} & \dots & x_{1f} & \dots & x_{1p} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & \dots & x_{if} & \dots & x_{ip} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nf} & \dots & x_{np} \end{bmatrix}$$

Obrázek 3.2: Datová matice (překresleno z [8])

3.1.2.2 Shluk

Shluk lze definovat jako množinu objektů, které jsou si navzájem podobné a objekty z jiných shluků si nejsou podobné [7]. Nyní je třeba definovat, jak zjistit, že jsou si některé objekty podobné. To zjistíme pomocí měr podobnosti, mezi které typicky patří tzv. **vzdálenostní funkce**. Ty se používají především u dat, která jsou numerického typu. Pokud nejsou data numerického typu, lze využít tzv. podobnostní funkce.

3.1.2.3 Vzdálenostní funkce

Vzdálenostní funkce nám slouží pro počítání vzdáleností mezi jednotlivými objekty. Vzdálenost **d**, mezi objekty **p** a **q** s dimenzemi numerického typu se počítá nejčastěji pomocí Euklidovské vzdálenosti:

$$d\left(p,q\right)=\sqrt{\sum_{i=1}^{n}\left|q_i-p_i\right|^2} \tag{3.1}$$

Alternativní způsob pro výpočet vzdálenosti **d**, mezi objekty **p** a **q** a je pomocí Manhattanské (city block) vzdálenosti:

$$d\left(p,q\right)=\sum_{i=1}^{n}\left|q_i-p_i\right| \tag{3.2}$$

3.1.3 Dělení algoritmů shlukové analýzy

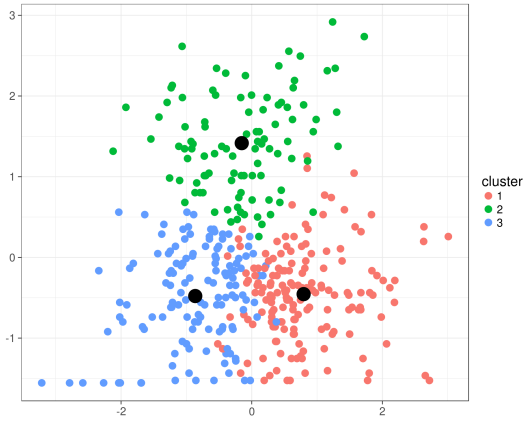
V literatuře existuje mnoho algoritmů pro shlukovou analýzu. Je obtížné poskytnout přesnou kategorizaci metod shlukování, protože tyto kategorie se mohou překrývat. Je však užitečné představit relativně organizovaný obraz metod shlukování. Obecně lze hlavní metody shlukování rozdělit do následujících kategorií, o nichž pojednává zbytek této sekce [6].

3.1.3.1 Metody založené na rozdělování

Tyto metody rozdělují data do **k** shluků, přičemž **k** parametr je předem znám a zároveň platí, že $\mathbf{k} \leq \mathbf{n}$, kde **n** značí počet vstupních objektů. To znamená, že rozděluje data do **k** skupin tak, že každá skupina musí obsahovat alespoň jeden objekt. Základní metody dělení obvykle používají výlučné oddělení shluků. To znamená, že každý objekt musí patřit přesně do jedné skupiny [6].

Většina metod založených na rozdělování je postavena na vzdálenosti. Vzhledem k počtu shluků (k), které se mají vytvořit, vytvoří metoda počáteční rozdělení. Poté použije **iterativní techniku přemístění**, která se pokusí zlepšit rozdělení pomocí přesunu objektů z jednoho shluku do druhého.

Nejpopulárnější představitel této metody je **k-means**. K-means předpokládá, že objekty lze chápat jako body v nějakém eukleidovském prostoru a že počet shluků **k** je předem dán. Případně lze vyzkoušet různá k a najít nejoptimálnější, například přes *elbow metodu* [9]. Shluky jsou definovány centroidy, což jsou body ve stejném prostoru jako shlukované objekty. Objekty se přiřazují do shluku, jehož centroidu jsou nejblíže. Algoritmus postupně zlepšuje kvalitu shlukování a přibližují se lokálnímu optimu.

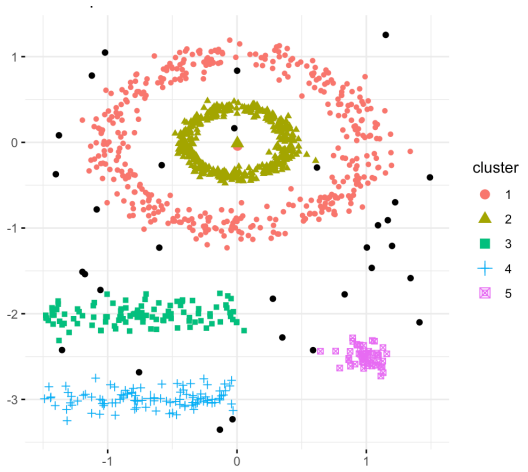


Obrázek 3.3: Příklad shluků rozdělených pomocí metody k-means (převzato z [10])

3.1.3.2 Metody založené na hustotě

Většina metod shlukové analýzy rozděluje objekty na základě vzdálenosti mezi objekty. Takové metody mohou najít pouze shluky sférického tvaru a setkat se s obtížemi při objevování shluků libovolných tvarů. Tento problém řeší metody založené na hustotě. Klíčovou myšlenkou těchto metod je postupné zvětšování shluku, do té doby dokud se v okolí hraničního bodu nachází dostatečné množství bodů [6].

Takovou metodu lze použít k odfiltrování šumu nebo odlehlých hodnot a k objevení shluků libovolného tvaru. Na takovémto principu funguje i metoda **DBSCAN (Density-Based Spatial Clustering)**.



Obrázek 3.4: Příklad shluků rozdělených pomocí metody DBSCAN (převzato z [11])

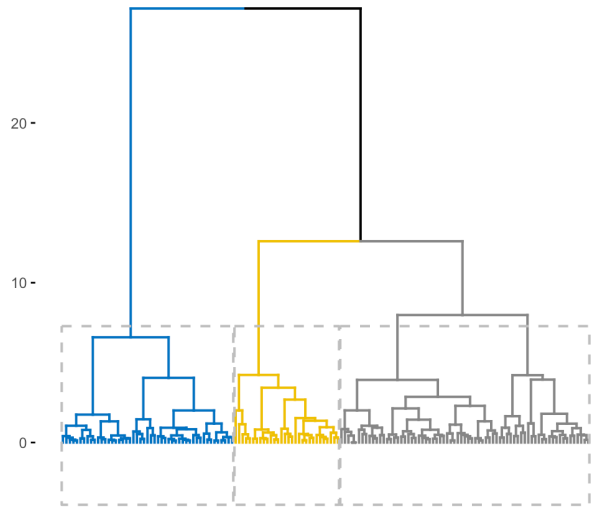
3.1.3.3 Metody založené na hiearchii

Hierarchická metoda vytváří hierarchický rozklad datových objektů. Hierarchickou metodu lze rozdělit na aglomerativní a rozdělovací na základě toho, jak se hierarchický rozklad vytváří [6].

Aglomerativní přístup, nazývaný také přístup zdola nahoru, začíná tím, že každý objekt tvoří samostatnou skupinu. Postupně slučuje objekty nebo skupiny blízko sebe, dokud nejsou všechny skupiny sloučeny do jedné (nejvyšší úroveň hierarchie) nebo dokud nebude platit podmínka ukončení.

Rozdělovací přístup, nazývaný také přístup shora dolů, začíná u všech objektů ve stejném shluku. V každé po sobě jdoucí iteraci je shluk rozdělen na menší shluky, dokud nakonec není každý objekt v jednom shluku, nebo dokud nebude zachována podmínka ukončení.

Hierarchické metody trpí tím, že jakmile je proveden krok (sloučení nebo rozdělení), nelze jej nikdy vrátit zpět, tedy nemohou opravit chybná rozhodnutí. Tato vlastnost je ale na druhou stranu užitečná v tom, že vede k nižším nákladům na výpočet, protože nemůže nastat kombinační počet různých možností.



Obrázek 3.5: Příklad shluků rozdělených pomocí metody založené na hiearchii (převzato z [11])

3.1.3.4 Metody založené na mřížce

Metody založené na mřížce rozdělují prostor objektů na konečný počet buněk, které tvoří strukturu mřížky. Všechny operace shlukování se provádějí na struktuře mřížky (tj. na kvantizovaném prostoru).

Hlavní výhodou tohoto přístupu je jeho rychlá doba zpracování, která je obvykle nezávislá na počtu datových objektů a závisí pouze na počtu buněk

v každé dimenzi v kvantizovaném prostoru [6].

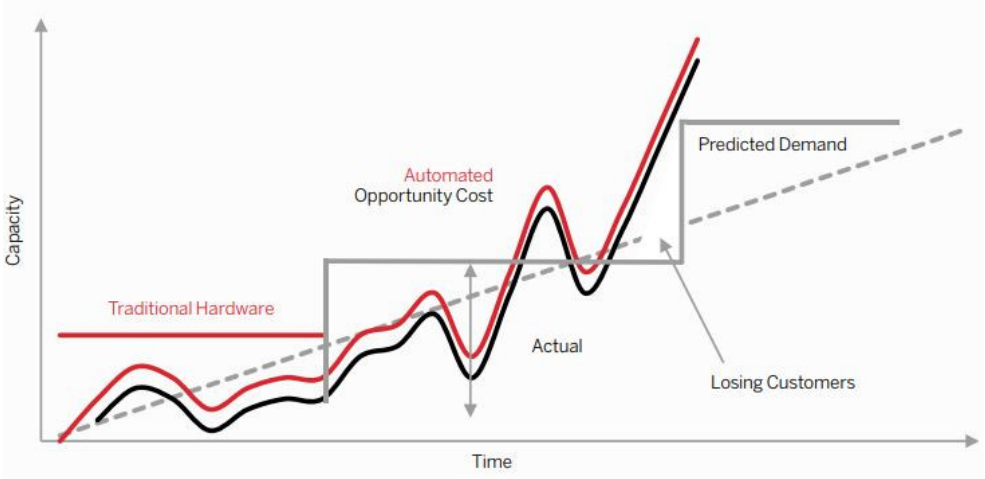
Používání mřížek je často efektivním přístupem k mnoha problémům s těžbou prostorových dat, včetně shlukování. Proto mohou být metody založené na mřížce integrovány s jinými metodami shlukování, jako jsou metody založené na hustotě či hierarchické metody.

3.2 Cloudové služby

V dnešním vysoce konkurenčním prostředí hledají podniky způsoby a prostředky, jak efektivně fungovat tak, aby **snižovaly náklady** a **maximalizovali zisk**. Objevilo se nové paradigma výpočetní techniky, cloud computing, které mění staré způsoby výpočtu. Cloud computing a cloudové služby obecně se ukázaly jako jedna z cest, která umožňuje světu informačních technologií efektivněji využívat počítačové zdroje [12].

Jednou z hlavních výhod cloudových služeb je flexibilnilní škálování výpočetního výkonu podle aktuální zátěže. Například webový portál poskytující sportovní videopřenosy během olympiských her dosahuje citelně větších počtů návštěv na svých stránkách, což vede k vyšším nárokům na výkon. Na druhou stranu jindy provoz značně opadá. Pokud by si firma provozovala webový portál na svém vlastním hardwaru a nechtěla by se dostat do situace, že by jejich hardware nestačil náporu uživatelů v největší špičce, musela by nakoupit výpočetní zařízení, které by bylo po většinu času využité jen z části. I přesto by nikdo zcela jistě nedokázal zaručit to, že při masivní návštěvnosti budou tyto prostředky dostatečné.

Cloud computing umí vyřešit právě tento problém. Cloud díky nastavenému škálování dokáže hardwarové prostředky snadno dynamicky přizpůsobovat na základě aktuální zátěže, což ilustruje obrázek 3.6. Tato flexibilita v podobě škálování hardwarových prostředků dokáže firmě mimojiné také šetřit náklady, jelikož uživatel platí jen za to, co skutečně využívá.



Obrázek 3.6: Cloud computing a jeho hlavní benefit - (automatické) škálováníi (převzato z [13])

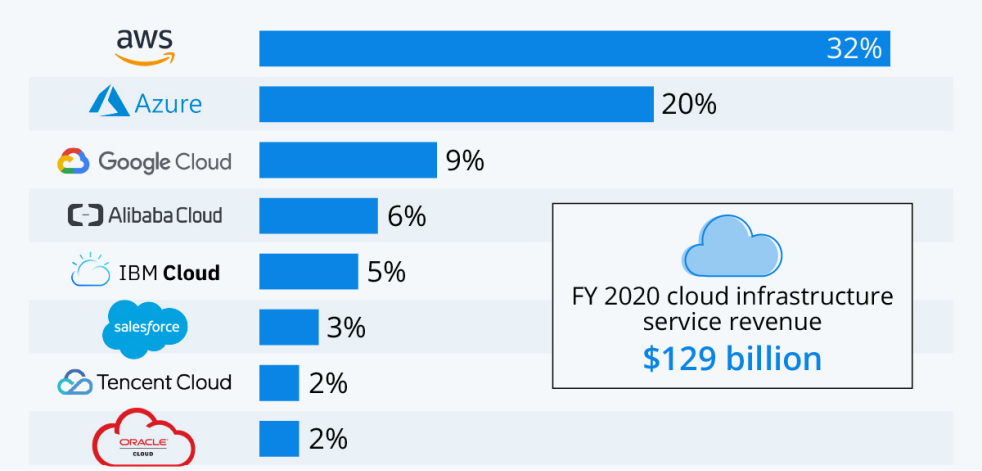
Provozovat software v cloudu nese velké výhody hlavně do začínajících podnikatelských projektů, které díky cloudu nemusí na začátku svého podnikání investovat nemalé částky do poskytování vlastních serverů. Výhodou je, že v cloudu se typicky **cena odvíjí od velikosti zátěže**, to tedy v praxi znamená, že pokud má firma v začátcích například jen málo uživatelů, tak cena za cloud computing bude velmi malá a začne růst teprve se zvyšujícím se počtem jejich uživatelů.

Mezi navýhodu lze zahrnout problematiku legislativy, konkrétně **ochranu osobních údajů**, neboť zákony, zabývající se ochranou těchto údajů, se v mnoha zemích velmi liší. A právě umístění serverů, na kterých jsou naše data uloženy, hrají významnou roli v legislativě [14].

3.2.1 Poskytovatelé cloudových služeb

Na trhu existuje celá řada poskytovatelů cloudových služeb. Přičemž lídrem v oblasti cloudových služeb (z pohledu obratu) je už dlouhodobě **Amazon Web Services**. Podle odhadů Synergy Research Group [15] činil podíl Amazon Web Services na celosvětovém trhu cloudové infrastruktury ve čtvrtém čtvrtletí roku 2020 32 procent, což stále převyšuje kombinovaný tržní podíl jeho dvou největších konkurentů, **Microsoftu Azure** a **Google Cloud**.

Mezi další poskytovatele cloudových služeb patří **Alibaba Cloud**, **IBM Cloud**, **Salesforce**, **Tencent Cloud** či **Oracle Cloud**. Jak ukazuje následující graf, Amazon a Microsoft představovaly ve 4. čtvrtletí 2020 více než polovinu podílu na trhu cloudových služeb, přičemž osm největších poskytovatelů ovládalo zhruba 80 procent trhu.



Obrázek 3.7: Celosvětový podíl na trhu předních poskytovatelů služeb cloudové infrastruktury ve 4. čtvrtletí 2020 (převzato z [16])

U většina z hlavních poskytovatelů cloudových služeb, kteří nyní zveřejnili své údaje o příjmech za 4. čtvrtletí Synergy Research Group odhaduje [15], že čtvrtletní obraty cloudových služeb (včetně IaaS, PaaS a hostovaných privátních cloudových služeb) činily 37,1 miliardy dolarů, což je o 35% více než ve čtvrtém čtvrtletí minulého roku. Za celý rok 2020 má tento sektor obrat 129 miliard dolarů.

Geograficky cloudový trh nadále silně roste ve všech regionech světa. Trvalo pouhých devět čtvrtletí, než se trh zdvojnásobil, což je na tak velkém trhu neobvyklé.

3.2.2 Amazon Web Services

Amazon Web Services je dceřinou společností společnosti Amazon poskytující cloudové služby na vyžádání. Tyto služby využívají jednotlivci, společnosti či vlády prostřednictvím platebního modelu založeném na průběžných poplatcích.

Amazon Web Services poskytuje řadu základních abstraktních technických infrastruktur, stavebních bloků a nástrojů pro distribuované výpočty. Jednou z těchto služeb je **Amazon Elastic Compute Cloud (EC2)**, který umožňuje mít uživatelům k dispozici virtuální server, ke kterému lze přistoupit přes internet. Virtuální server emuluje většinu atributů skutečného počítače, včetně hardwarových procesorových jednotek (CPU), grafických procesorů (GPU), RAM paměti, úložiště na pevném disku, výběr operačních systémů, síťování nebo předinstalovaný aplikační software jako jsou webové servery, databáze či řízení vztahů se zákazníky (CRM).

Poplatky jsou založeny na kombinaci využití hardwaru, operačního systému, softwaru nebo síťových propojení, které se starají o dostupnost, redundanci, zabezpečení a tak dále.

Technologie AWS je implementována na serverových farmách po **celém světě** a je udržována společností Amazon. Jak ilustruje obrázek 3.8 AWS působí v mnoha globálních geografických oblastech, z toho je 6 v Evropě (dále jsou 2 Evropské serverové farmy ve výstavbě).

Obrázek 3.8: Globální mapa serverových farem Amazon Web Services (převzato z [17])

V současné době zahrnuje Amazon Web Services **více než 200 produktů a služeb** [18], včetně virtuálních serverů, úložiště, sítí, databází, analytických služeb, aplikačních služeb, nástrojů pro nasazení a správu a nástrojů pro internet věcí.

Mezi nejoblíbenější patří Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (Amazon S3) a AWS Lambda.

Většina služeb není poskytována přímo koncovým uživatelům, ale místo toho je využívají prostředním aplikačního rozhraní vývojáři, kteří na cloudové architektuře vytvářejí software pro koncové uživatele. Služby Amazon Web Services jsou přístupné přes HTTP pomocí architektonického stylu REST a dále pomocí protokolu SOAP.

Dále Amazon Web Services vydalo a udržuje řadu nástrojů, které pomáhají vývojářům s vývojem cloudových aplikací. Konkrétně se jedná například o nástroje pro **automatizaci nastavení infrastruktury** (jako je AWS Cloud Development Kit) nebo **SDK moduly** do jednotlivých programovacích jazyků (jako je například TypeScript), skrze které lze napřímo z programového kódu

provolávat služby Amazon Web Services (například přistupovat k Amazon Simple Storage Service).

3.2.3 AWS serverless

serverless je způsob, jak popsat služby, postupy a strategie, které umožní vytvářet agilnější aplikace, tak aby bylo možné rychleji inovovat a reagovat na změny. Se serverless jsou práce spojené se správou infrastruktury, jako je zajišťování kapacity hardwarových prostředků odstraněno a je zajišťováno cloudovým poskytovatelem. Takže je možné soustředit se pouze na psaní kódu, který souvisí se samotnou logikou aplikace.

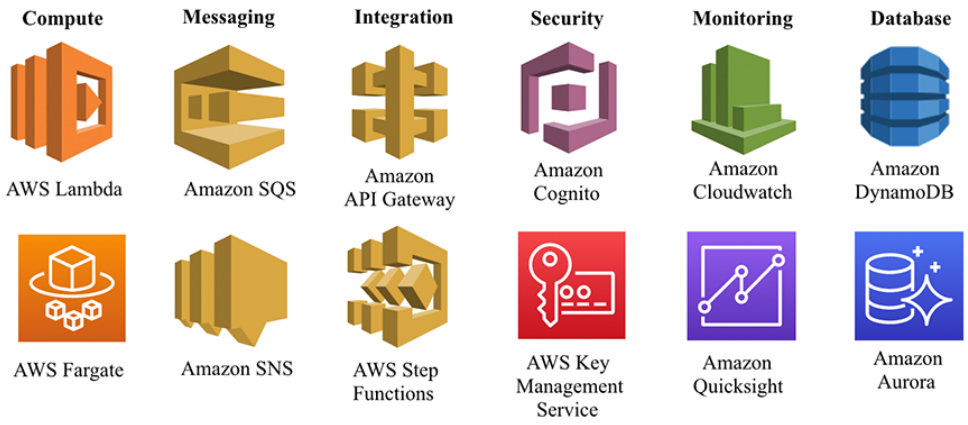
AWS serverless technologie, jejimž nejvýraznějším představitelem je AWS Lambda, přicházejí s **automatickým škálováním**, integrovanou vysokou dostupností a typickým cenovým modelem s platbou jen za počet použití[2]. Lambda je výpočetní služba řízená událostmi, která umožňuje spouštět kód v reakci na události z více než 150 nativně integrovaných zdrojů AWS - to vše bez správy jakýchkoli serverů [19].

Tento přístup tedy přináší řadu benefitů jako **rychlé uvedení nápadu v realitu**, jelikož není potřeba složitě definovat serverovou infrastrukturu. Dále nám serverless technologie **automaticky škálují**, což nám přináší obrouvskou výhodu v podobě flexibility a stability. Plus je tu jeden z nejvyšších benefitů v podobě **flexibilního platového modelu**, v němž se platí u jednotlivých služeb jen za počet sekund či počet použití dané služby. To má za výsledek, že užívání služby při nízkém provozu stojí extrémně málo peněz.

3.2.3.1 AWS serverless technologie

V rámci AWS serverless lze použít řadu technologií, níže budou představeny ty nejvíce klíčové [19][20].

[2]Detailnější popis platebního modelu AWS serverless technologií lze načíst například zde: `https://techbeacon.com/enterprise-it/economics-serverless-computing-real-world-test`



Obrázek 3.9: Přehled hlavních AWS serverless technologií (převzato z [20])

Výpočetní výkon

- S **AWS Lambda** lze spustit kód jako funkci spouštěnou různými zdroji pomocí událostí, jako jsou například HTTP requesty na AWS API Gateway, aktualizace souborů na S3 či spouštění alarmů v AWS Cloudwatch. Lambda je řešení typu FaaS, kde lze spustit kód pro jakýkoliv typ aplikace nebo backend služby. Lambda spouští kód na plně spravované, vysoce dostupné infrastruktuře a provádí všechny administrativní úlohy výpočetních prostředků, včetně údržby serveru a systému, zajišťování kapacity a automatického škálování a monitorování kódu. Platí se pouze za výpočetní čas, který aplikace skutečně spotřebuje. Lambda lze kombinovat s mnoha dalšími službami AWS.
- S **AWS Fargate** lze spouštět Docker kontejnery bez jakékoli správy serverů nebo klastrů. Jedná se o řešení orchestrace kontejnerů, které usnadňuje nasazení, správu a škálování kontejnerových aplikací. Pro škálování instancí nemusíte definovat typy EC2 instancí, spravovat plánování klastrů, optimalizovat využití serveru nebo definovat metriky pro monitoring.

Messaging

- **AWS Simple Notification Service** (SNS) je plně spravovaná služba pro pub/sub messaging, která umožňuje oddělit mikroslužby, distribuované systémy či aplikace. Je možné odesílat zprávy mezi různými službami, aplikacemi, zařízeními a platformami prostřednictvím více protokolů. SNS umožňuje odesílat zprávy z jedné aplikace velkému počtu subscriberů pro paralelní zpracování.

- Služba **Amazon Simple Queue Service** (SQS) je plně spravovaná služba pro distribuované frontování zpráv (producer-consumer problém), která umožňuje oddělit a škálovat distribuované mikroslužby.

Integrace

- **Amazon API Gateway** je plně spravovaná služba pro vytváření, publikování, údržbu, monitorování a zabezpečení rozhraní REST API. Zpracovává requesty až stovek tisíc souběžných volání na API. Platí se pouze za přijaté requesty na API a za množství odchozích dat.
- S **AWS Step Functions** lze orchestrovat komponenty aplikace jako sérii kroků, kde jsou jednotlivé kroky realizované pomocí Lambda funkcí a představují stavový diagram. Pomocí grafického prohlížeče lze tyto komponenty vizualizovat a zkontrolovat tok provádění v reálném čase. Všechny kroky jsou automaticky spouštěny, monitorovány a opakovány v případě chyb.

Autorizace a bezpečnost

- **Amazon Cognito** je plně spravovaná, škálovatelná a nákladově efektivní služba pro registraci a přihlašování, která poskytuje ověřování, autorizaci a správu uživatelů pro webové a mobilní aplikace. Registrace a přihlášení lze realizovat na základě svého vlastního nebo externího poskytovatele identity, který implementuje protokol SAML, OAuth2 nebo OpenID Connect (jako je Facebook či Google). Pro přihlášení uživatele lze použít přizpůsobitelné webové uživatelské rozhraní, které lze integrovat do stávající webové stránky.
- **AWS Key Management Service** (KMS) je plně spravovaná služba, která umožňuje vytvářet a spravovat bezpečnostní klíče a řídit používání šifrování v celé řadě služeb AWS.

Monitoring

- **AWS CloudWatch** je služba pro monitoring a správu AWS zdrojů či vlastních aplikací. CloudWatch umožňuje shromažďovat metriky ze všech aplikací a AWS služeb. Poskytuje monitorování v reálném čase a také nabízí srozumitelné přehledy. Kromě toho lze definovat alarmy na základě metrik z CloudWatch a provádět akce na základě jejich hodnoty.
- **AWS Quicksight** je plně spravovaná služba Business Intelligence, která umožňuje vytvářet vizualizace dat a navrhovat interaktivní řídící panely pro analýzu aplikací. K těmto panelům lze přistupovat z jakéhokoli mobilního zařízení nebo webového prohlížeče. Lze je vložit do aplikací,

portálů nebo webových stránek a poskytnout tak výkonnou samoobslužnou analýzu.

Databáze a úložiště

- **Amazon DynamoDB** je key-value NoSQL databázová služba. Jelikož se jedná o plně spravovanou službu, tak je zajištěno poskytování hardwaru, nastavení, konfigurace, replikace, zálohován nebo škálování. Dále DynamoDB nabízí latenci v milisekundách. Díky cenovému modelu s platbou za dotaz a díky snadné integraci s mnoha dalšími AWS službami je DynamoDB skvělou databázovou službou pro mnoho serverless aplikací. Uložení dat v DynamoDB připomíná relační přístup - požívají se zde Tabulky a primární klíče, nicméně na pozádí je jsou data uložena stylem klíč-hodnota (key-value). Více o této databázi bude popsáno v rámci 4.3.4.
- **Amazon Aurora** je plně spravovaný relační databázový server, který je kompatibilní s MySQL a PostgreSQL. Automatizuje a standardizuje klastrování a replikaci databází, aby odstranil časově náročné administrativní úlohy, jako je zajišťování hardwaru, automatické škálování úložiště, nastavení databáze, opravy a zálohy.
- **Amazon Simple Storage Service** (Amazon S3) je služba pro ukládání objektů, která nabízí špičkovou škálovatelnost, dostupnost dat, zabezpečení a výkon. Lze ji využít pro servírování webu, zálohování/obnovení, archivaci, pro tvorbu datových jezer nebo pro mnoho dalších případů užití.

Kapitola 4

Návrh technologií

V následující kapitole bude věnována pozornost návrhu použitých technologií, knihoven a postupů pro realizaci prototypu. Kapitola je rozdělena do 4. částí. V první části jsou popsány navržené **obecné použité technologie** jako je programovací jazyk. Další podkapitola patří návrhu knihoven pro realizaci **shlukové analýzy**. 3. část se věnuje použitým **službám v rámci Amazon Web Services**. Závěrečná podkapitola se věnuje návrhu přístupu s názvem **infrastruktura jako kód**.

4.1 Obecné použité technologie

Jako hlavní programovací jazyk byl zvolen **JavaScript**. Mezi důvody patří kompatibilita se zadáním v podobě možnosti napsat webovou službu, která bude pomocí algoritmů shlukové analýzy tvořit doporučení pro nastavení chytré domácnosti - JavaScript má k dispozici velké množství knihoven v rámci balíčkovacího systému NPM a to včetně knihoven pro shlukovou analýzu. Dále je JavaScript dobře podporován v rámci AWS serverless technologií, které budou použity pro stavbu služby. Alternativou by mohl být programovací jazyk Python jehož výhoda je taktéž mimojiné velká základna existujících knihoven a to především v oblasti strojového učení. Nicméně jeden z faktorů pro rozhodování byl i fakt, že s jazykem JavaScript mám už dlouholeté zkušennosti.

V první části tedy popíšu jazyk **JavaScript**. Pro větší robustnost kódu byl použit **TypeScript**, o kterém bude další část. Následovat bude popis o **Node.js**, což je prostředí pro běh JavaScriptu. Závěr této části bude patřit již zmíněnému balíčkovacímu systému **NPM**.

4.1.1 JavaScript

JavaScript je interpretovaný, multiparadigmatický, který je nejvíce známý jako skriptovací jazyk pro webové stránky, ale používá se také v mnoha jiných prostředích než jenom v prostředích pro webové prohlížeče. Mezi tyto další

prostředí patří například serverové prostředí (Node.js), mobilní prostředí (React Native) či prostředí desktopových aplikací (Electron). Jedná se o **dynamický prototypovací skriptovací jazyk** s více paradigmaty, který podporuje objektově orientované, imperativní a funkcionální programovací styly.

JavaScript může fungovat jako procedurální i objektově orientovaný jazyk. Objekty jsou vytvářeny za běhu programově v JavaScriptu připojením metod a vlastností k jinak prázdným objektům, na rozdíl od definic syntaktických tříd v běžných kompilovaných jazycích, jako jsou C++ a Java. Jakmile je objekt zkonstruován, může být použit jako **prototyp** pro vytváření podobných objektů [21].

4.1.2 Typescript

TypeScript je open-source programovací jazyk, který je postaven na JavaScriptu, do ktereho přidává **definice statických typů**. Typy poskytují způsob, jak popsat tvary objektů, proměnných či výstupních parametrů funkcí. To umožňuje TypeScriptu ověřit, že kód funguje správně ještě před spuštěním. To je obzvláště výhodné pro psaní velkých aplikací, u kterých je tak zajištěna větší robustnost, stabilita a celkově funguje dobře jako prevence chyb.

Protože je TypeScript nadmnožinou JavaScriptu, jsou stávající programy JavaScriptu také platnými programy TypeScriptu. Kód TypeScriptu je transpilován do JavaScriptu pomocí kompilátoru TypeScript nebo nástroje Babel [22].

TypeScript má i řadu dalších funkcí jako *interface, dekorátory, generiky* nebo *stringové enumy.*

4.1.3 Node.js

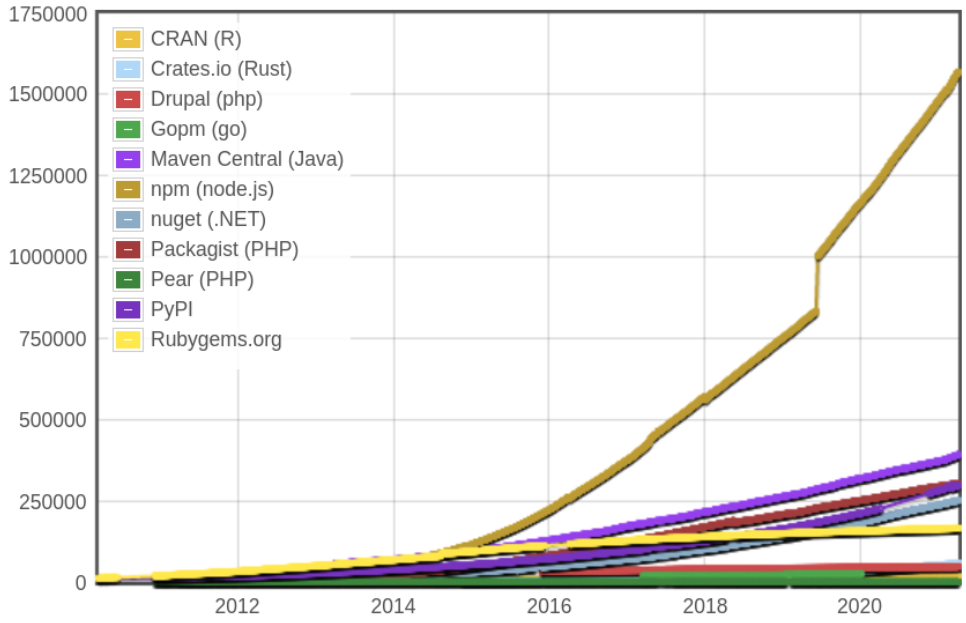
Node.js je vysoce výkonné, událostmi řízené prostředí pro JavaScript. Základem Node.js je JavaScriptový **interpret V8** od společnosti Google. Nad ním je tenká vrstva kódu v jazyce C++ poskytující potřebné zázemí jako je event-loop vyhodnocující příchozí události, obsluhu Input/Output bufferů a tak dále. Avšak Node.js je navržen tak, aby vývojáře odstínil od nízkoúrovňových problémů. Node.js je čistý JavaScript běžící v jednom vlákně. Programy pro Node.js využívají **asynchronní operace** pro minimalizaci režie procesoru a maximalizaci výkonu [23].

Node.js se typicky využívá pro psaní backendových služeb, které mohou vystavit REST API, GraphQL API či komunikovat s dalšími službami nebo frontendovými aplikacemi jiným způsobem. Jak již bylo zmíněno výše Node.js lze dobře kombinovat s TypeScriptem a taktéž ho lze využít v AWS Lambda funkcích. Je to tedy ideální technologie pro řešení problému této práce.

4.1.4 NPM

NPM nebo také **Node Package Manager** je správce balíků pro JavaScript. Pro Node.js je výchozím správcem balíků a instaluje se společně s Node.js. NPM se stará o proces instalace, odstranění, upgrade a hlídání kompatibility používaných balíků v rámci projektu. Pod balíky si lze představit například knihovny, ale i celé frameworky.

NPM se skládá z klientského programu, který se používá v příkazové řádce a online databáze věřejných balíků které jsou zdarma. NPM nabízí už více než 1 550 000 balíků, což z něho dělá suverénně největšího správce balíků [24].



Obrázek 4.1: Přehled správců balíků z pohledu počtu dostupných balíků (převzato z [24])

4.2 Použité technologie pro shlukovou analýzu

Středobod vznikající služby se bude točit okolo shlukové analýzy. V této sekci bude popsána knihovna, která umožňuje používat algoritmy **shlukové analýzy**. Dále bude představen návrh pro **vizualizaci shluků**. Poslední část se zaměří na to, jak získávat pro jednotlivé adresy časy **západů a východů slunce** v jednotlivých dnech v roce - tyto časy se využijí jako dimenze pro realizaci prototypu.

4.2.1 Knihovna pro shlukovou analýzu

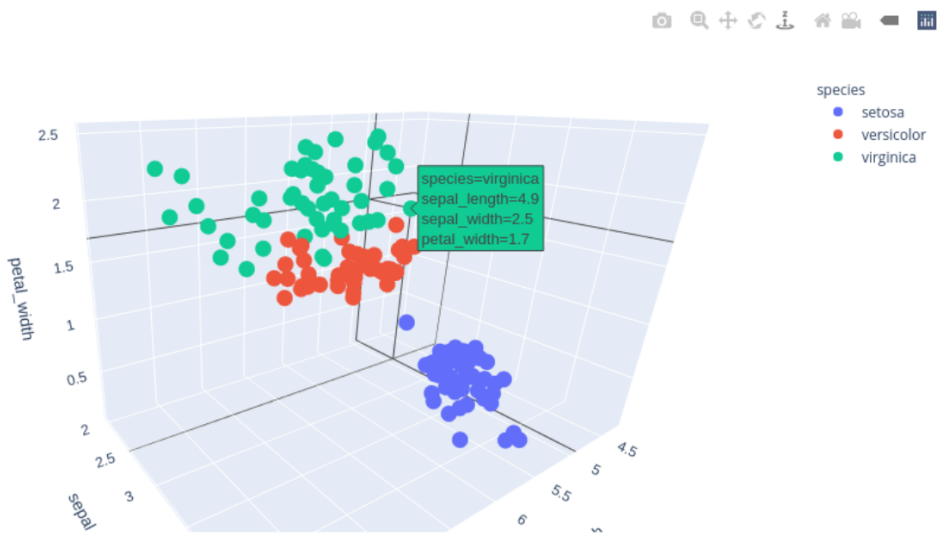
Dle analýzy existuje více metod pro tvorbu shlukové analýzy. Mezi ty často skloňované patří metoda založená na rozdělování s hlavním představitelem **k-means** a metoda založená na hustotě s představitelem **DBSCAN**. Moje myšlenka je pokusit se udělat shlukovou analýzu těmito dvěma způsoby.

Z čehož vychází návrh knihovny. Byla vybrána knihovna z NPM s názvem: **density-clustering** [25]. Tato knihovna dokáže řešit oba zmíněné přístupy - k-means a DBSCAN a má týdenní stažení větší než 100 000. V rámci NPM lze nalézt více podobných knihoven, nicméně tato byla mezi jedněmi z nejvíce stahovanými a také dokáže používat více metod shlukové analýzy, není tak třeba integrovat více knihoven.

Další alternativou by mohlo být využití shlukové analýzy jako služby například na AWS, zde existuje služba na strojové učení, která nabízí mimo jiné použití k-means, ale už nenabízí DBSCAN [26].

4.2.2 Vizualizace shluků

Důležitým faktorem při ladění parametrů shlukové analýzy je vizualizace jednotlivých shluků. V rámci NPM existuje velká řada knihoven, které dokáží vygenerovat obrázek shluků. Nicméně po chvíli pátrání jsem našel ještě něco lepšího - online nástroj, který umí vygenerovat **interaktivní 3D graf**. Je tedy možné libovolně přibližovat, otáčet, naklánět prostor a detailně pozorovat jednotlivé objekty a shluky - tento nástroj se jmenuje Plotly [27].



Obrázek 4.2: Ukázka interaktivního prosředí v rámci plotly.com (vytvořeno jako snímek obrazovky z [27])

Skrze knihovnu z NPM s názvem **plotly** [28] lze iniciovat vytvoření interaktivního grafu. Knihovna na pozadí provolá plotly.com, vytvoří tam graf a jako výsledek poskytne URL, na které lze graf otevřít a procházet. Proces lze ještě vylepšit pomocí NPM knihovny **opn** [29], která umí otevřít URL ve výchozím webovém prohlížeči.

Nutno podotknout, že si je nutné vytvořit pro používání interaktivních grafů na `plotly.com` účet. Nicméně v základní verzi je zdarma generování 100 grafů denně.

4.2.3 Získání informací o západu/východu slunce

Jeden z navrhovaných bodů je ten, že jako dimenze pro shlukovou analýzu bude zvolen **západ a východ slunce**. Když bude známý datum, čas a poloha logu z chytré domácnosti, bude pak možné odvodit nové dimenze, které budou zohledňovat rozdíl (například sekund) od západu, respektive východu slunce. Tyto informace jsou totiž deterministické a dají se získat i historicky.

Jeden ze způsobů, jak tyto informace získat je pomocí API z `https://sunrise-sunset.org/api`. Nicméně ještě přímočařejší přístup je skrze NPM knihovnu **sunrise-sunset-js** [30], která poskytuje ty stejné informace - tedy to, že na základě polohy a času poskytne informace o tom, v kolik hodin byl v daný den západ a východ slunce.

Do dimenzí je ukládán rozdíl v sekundách. Rozdíl je počítán pomocí knihovny **moment.js** [31].

4.3 Použité technologie a úložiště v rámci AWS

Přehledový popis jednotlivých AWS serverless komponent byl již učiněn v rámci 3.2.3. Nicméně v této části bude popsáno jaké **služby** a v jakých kombinacích byly použity pro návrh implementace prototypu.

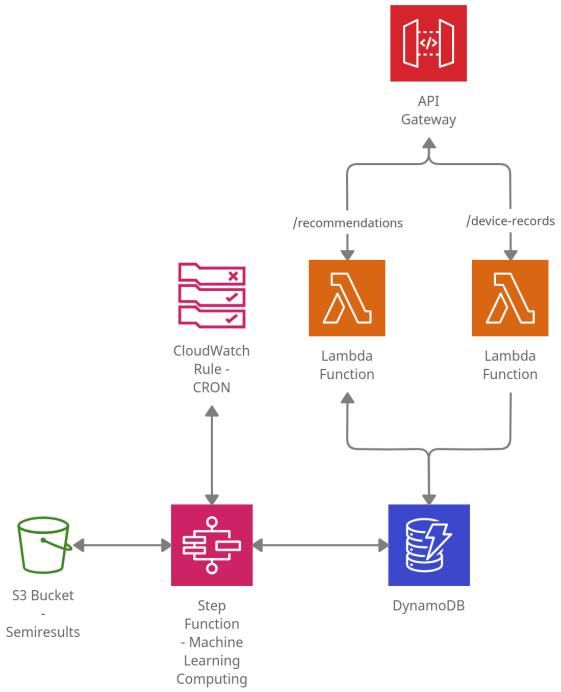
Bude zde taktéž rozveden a diskutován **návrh AWS architektury** jako celku. Společně s prezentací výběru služeb bude popsáno jaké specifika jednotlivé služby mají.

4.3.1 Návrh AWS architektury

Navržená AWS architektura se dá rozdělit do několika hlavních částí. Středobod bude počítání doporučení na základě algoritmů strojového učení, konkrétně shlukové analýzy. Myšlenka je taková, že bude výpočet rozdělen do dílčích malých částí (jako získávání dat, výpočet pomocí k-means...), z nichž půjde některé i paralelizovat. Každá takováto dílčí část bude jedna **Lambda funkce**. Celý proces orchestrace jednotlivých Lambda funkcí zastřeší **Step Functions**. Tím bude vyřešena tvorba doporučení.

Další částí bude poskytnutí aplikačního rozhraní a backendové služby pro příjem IoT senzorických dat a následného poskytnutí doporučení. Pro tento účel bude použita **API Gateway** v kombinaci s **Lambda Funkcemi**.

Co se týče databáze, bude využito **DynamoDB**. V této databázi budou uloženy jak vstupní senzorická data, tak i výstupní doporučení.



Obrázek 4.3: Návrh AWS architektury, přičemž unitř Step Function se skrývá sada AWS Lambda funkcí, pro kroky výpočtu doporučení na základě shlukové analýzy

Níže budou rozvedeny detailnější popisy a argumenty pro volby jednotlivých služeb, případně budou diskutovány alternativy.

4.3.2 Výpočet doporučení

Vize je rozdělit proces shlukové analýzy do dílčích částí. Má to několik benefitů, mezi které patří: možnost některé dílčí části paralelizovat, nastavovat jim různé výpočetní parametry (např CPU), možnost udělat některé části obecné (a tedy znovupoužitelné, například pro jiný typ doporučovacího systému). Jelikož se logika rozbije do malých jasně definovaných částí může to také zvýšit čitelnost kódu. Dále bude velmi jednoduché implementovat rozšíření - například využít další typ shlukové analýzy.

4.3.2.1 Step functions

O orchestraci procesu shlukové analýzy se postará AWS Step Functions. Step Functions je **orchestrační serverless služba**, která umožní kombinovat AWS Lambda funkce a další AWS služby. Prostřednictvím grafické konzole Step functions lze vidět posloupnost série volání funkcí jako řadu kroků řízených událostmi. Step Functions je založena na stavových automatech a úlohách (Task). Task je stav, který představuje jednu pracovní jednotku, kterou provádí AWS služba - typicky Lambda funkce [32].

Základní princip je takový, že jedna Lambda funkce při ukončení automaticky vygeneruje událost, která aktivuje další Lambda funkci, která je v pořadí a její vstup je výstup předchozí funkce.

Step functions nabízí spoustu zajímavých funkcionalit jako je opakování vykonání stavu (Lambda funkce) v případě selhání, větvení (podmínkové stavy), paralelizace stavů (Parallel operace), dynamické paralelizování (Map operace - pro dynamické pole provede paralelní vykonání stejné stavu s různými parametry).

4.3.2.2 Lambda funkce

Jednotlivý krok (stav) bude Lambda funkce. AWS Lambda je **výpočetní platforma** založená na událostech. Jedná se o výpočetní službu, která spouští kód v reakci na události a automaticky spravuje výpočetní prostředky požadované tímto kódem. Lambda funkce podporuje jazyky jako Node.js, Python, Java, Go, Ruby či C# (prostřednictvím .NET Core). Na rozdíl od Amazon EC2 (virtuální server), jehož cena je účtována od hodiny je AWS Lambda účtována zaokrouhlováním nahoru na nejbližší milisekundu[3].

Tato skutečnost se zde velmi hodí - jelikož bude žádoucí spustit proces pro tvorbu doporučení s periodou například jednou za dva týdny a proces výpočtu bude trvat řádově jednotky/desítky minut. To by znamenalo v případě virtuálního stroje (EC2), že by stroj nebyl drtivou většinu času využit.

Dále je u Lambda funkcí dobrá vlastnost **automatické škálovatelnosti**. Konkrétně jedna funkce umí automaticky škálovat do 3000[4] souběžně vykonávaných operací (každé datacentrum má tento limit jiný, v této práci použité datacentrum v Irsku má právě tuto hodnotu) [33]. V kódu a infrastruktuře tak není nutné řešit nějaké věci ohledně této problematiky, Lambda funkce to udělá za nás.

Lambda funkci lze nastavit mimojiné **runtime** (např Node.js), **velikost CPU paměti** v MB (např 128MB), **timeout** pro vykonání (např 10s),

[3]Platební model je trochu složitější, detailnější popis platebního modelu AWS serverless technologií lze načíst například zde: `https://techbeacon.com/enterprise-it/economics-serverless-computing-real-world-test`

[4]Toto je výchozí hodnota, která jde na vyžádání zvýšit, jak lze pozorovat zde: `https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html`

proměnné prostředí (environment variables) a samotný **zdrojový kód** (který lze nahrát dokonce přímo v TypeScriptu např. prostřednictvím AWS CDK).

4.3.2.3 Simple Storage Service (Amazon S3)

Drobná nevýhoda použití Step functions je ta, že pro poskytnutí vstup/výstupních dat je omezen limit velikosti, konkrétně na 256KB [34]. Což při navrhovaném řešení nebude stačit - data shlukové analýzy budou dosahovat větších objemů.

Nicméně řešení není složité. Stačí si jednotlivé **mezivýsledky ukládat na nějaké úložiště** a mezi jednotlivými funkcemi si posílat referenci na úložiště, ve kterém je mezivýsledek uložený. Nejjednodušší je využít Simple Storage Service (Amazon S3).

Amazon Simple Storage Service (Amazon S3) je služba pro ukládání objektů, která nabízí špičkovou škálovatelnost, dostupnost, zabezpečení, výkon a retenci dat. Na S3 lze vytvořit tzv. **Bucket** a v Bucketu může být několik objektů (například objektů typu json) uložené pod **unikátním klíčem**. Dokonce lze vytvořit **adresáře** (folders), do kterých lze vkládat objekty nebo další adresáře - princip je velmi podobný jako adresářová struktura z operačních systémů.

Do S3 z kódu lze přistupovat pohodlně prostřednictvím **AWS SDK**. Integrace a používání je tak velmi jednoduché.

Poslední bod (který se vykoná vždy, bez ohledu na úspěch) Step functions bude Lambda funkce, která po sobě smaže veškeré mezivýsledky.

4.3.2.4 CloudWatch Rule

Pro účely periodické inicializace procesu tvorby doporučení v rámci AWS je vhodné použít službu CloudWatch Rule.

CloudWatch Rule dokáže vytvořit pravidla, která automaticky spustí v automatizovaném plánu události pomocí **CRON** definice nebo tzv. **Rate výrazů**, např. *rate(5 minutes)* pro definici opakování každých 5 minut. Všechny naplánované události používají časové pásmo UTC a minimální přesnost plánů je 1 minuta [35].

CloudWatch Rule tedy bude periodicky generovat události, které inicializují Step Functions proces pro tvorbu doporučení.

4.3.3 Backendová služba

Z pohledu AWS architektury bude pro účely tvorby backendového REST-API využita Amazon API Gateway v kombinaci s Lambda funkcemi.

4.3.3.1 API Gateway

Jako brána pro HTTP requesty bude využita služba Amazon API Gateway. Amazon API Gateway je plně spravovaná služba, která vývojářům usnadňuje vytváření, publikování, údržbu, monitorování a zabezpečení API v jakémkoli měřítku. Rozhraní API fungují jako „přední dveře“ aplikací pro přístup k datům či byznys logice z **Lambda funkcí**. Pomocí API Gateway lze vytvářet RESTful API či WebSocket API, které umožňují obousměrnou komunikaci v reálném čase.

API Gateway zajišťuje všechny úkoly spojené s přijímáním a zpracováním až stovek tisíc souběžných requestů na API, včetně správy provozu, podpory CORS, autorizace, řízení přístupu, monitorování a správy verzí API. API Gateway nemá žádné minimální poplatky ani náklady na spuštění. Platí se za volání API a za množství přenesených dat [36].

4.3.3.2 Lambda funkce

Jako výpočetní výkon na zpracování requestů pro backendovou službu bude sloužit Lambda funkce. Obrovská výhoda této služby na tento případ užití je automatická škálovatelnost, platba pouze za skutečně využitý čas a jednoduchost použití (jak bylo popsáno v 4.3.2.2).

Nicméně každá mince má dvě strany a to platí i pro tuto službu, která také obsahuje stinné stránky. Konkrétně se jedná o problém zvaný **cold start**. Jelikož Lambda funkce funguje na vyžádání, tak AWS před poskytnutím výpočetního výkonu funkce musí spustit instanci s tímto prostředím (a stáhnout do ní dané zdrojové kódy). Životnost instance je v případě neaktivity (například zpracování dalšího requestu) jen dočasná - zhruba 10 minut. Po této době je instance vypnuta a je znovu zapnuta až při zpracování další události. Odbavení této další události bude výrazně **pomalejší**, jelikož se musí opět zapnout instance - tomuto pomalejšímu startu se říká *cold start*.

Nelze přesně říci jak dlouho bude cold start trvat. Záleží na více faktorech - na runtime prostředí, velikosti přidělené paměti funkce, velikosti zdrojového kódu a dalších aspektech. Více o této problematice, včetně časových analýz na základě výše zmíněných atributů lze nalázt zde [37]. Nicméně vlastní konkrétní test mi u Lambda funkce (Node.js, 128MB CPU), která při nastartované instanci (warm start) odbaví request zhruba za 350ms, trval při cold startu zhruba 2s.

Tento aspekt tak nemusí být pro některé případy užití dostatečný. Avšak existují způsoby jak to řešit - například periodicky posílat requesty na API, aby se nevypnula instance, na což dokonce existuje už předchystané řešení [38].

Každopádně pro účel navrhované služby s kterou by mohla potenciálně interagovat jiná backendová služba nepředstavuje cold start větší problém.

4.3.4 Databáze

Jako databázová služba v rámci AWS bude využita služba **Amazon DynamoDB**. Alternativou by mohla být Amazon Aurora.

Amazon DynamoDB je plně spravovaná databázová key-value NoSQL databáze, která poskytuje rychlý a předvídatelný výkon. DynamoDB umožňuje snížit administrativní zátěž provozu a škálování distribuované databáze, takže není třeba řešit správu hardwaru, konfigurací, replikací nebo škálování. DynamoDB také nabízí šifrování, což eliminuje provozní zátěž a složitost ochrany citlivých dat.

V DynamoDB existují základní komponenty v podobě **tabulek**, **záznamů** a **atributů**. Tabulka je kolekce záznamů a každý záznam je kolekcí atributů. DynamoDB používá primární klíče k jedinečné identifikaci každé položky v tabulce a sekundární indexy, které poskytují větší flexibilitu při dotazování. DynamoDB dále nabízí použití streamů, které lze použít k zachycení událostí po úpravě dat v tabulkách [39].

DynamoDB podporuje dva různé druhy primárních klíčů. První je jednoduchý primární klíč, který je složený z jednoho atributu a je známý pod názvem **partition key**. DynamoDB používá partition key hodnotu jako vstup do interní hashovací funkce. Výstup z hashovací funkce určuje oddíl (interní fyzické úložiště DynamoDB), ve kterém je položka uložena.

Druhá alternativa je typ klíče, označovaný jako složený primární klíč a skládá se ze dvou atributů - **partition key** a **sort key**. Partition key funguje stejně, tedy DynamoDB ho používá jako vstup do interní hashovací funkce, jejímž výstupem je oddíl, ve kterém je položka uložena. Všechny položky se stejnou partition key hodnotou jsou uloženy společně, v seřazeném pořadí podle sort key hodnoty [40].

K této službě se z Node.js lze připojit opět prostřednictvím AWS SDK a k tvorbě tabulek lze využít AWS CDK. Při tvorbě tabulek je nutné definovat primární klíče a názvy tabulek. Zbytek atributů není třeba definovat. Do atributu lze uložit i objekt a to i různého tvaru, podobně jako například v MongoDB.

Tato databáze má ale i své nevýhody. Nepodporuje například, jako relační databáze, nativní spojování tabulek (join). Dále není možné (na databázové úrovni) agregovat.

Pro účel navrhované služby, která bude přijímat data od IoT zařízení a ukládat vypočítané doporučení bude tato služba vhodná (není třeba tvořit agregace či dělat složité spojování tabulek relačního typu). Dále budou v databázi uloženy informace o domech (jejich polohy).

4.4 Infrastruktura jako kód

Po rozmyšlení návrhu AWS architektury nastává otázka, jak tuto architekturu v AWS vytvořit a případně upravovat. Cest je několik, základní možnost

je otevřít si **webovou aplikaci** Amazon Web Services a pomocí UI nadefinovat architekturu. Tento proces je ale pomalý, náchylný na chyby a těžko znovupoužitelný.

Alternativní cesta je nainstalovat si do počítače **AWS Command Line Interface** a pomocí příkazová řádky zadávat příkazy pro vytvoření infrastruktury. Ani tento postup ale není zcela ideální, sice si lze nachystat příkazy, které se dají znovu použít, ale už například nelze vidět ucelený obraz o architektuře.

Nejnovějším trendem v této problematice je tzv. tvorba **infrastruktury jako kódu**. To tedy znamená, že do nějakého jazyka, který je čitelný pro člověka i stroj zadá uživatel definici infrastruktury. Výhodou je primárně dobrá rozšiřitelnost a znovupoužitelnost.

Tohoto přístupu lze v AWS dosáhnout pomocí **CloudFormation** nebo pomocí nástroje **Terraform**. Což je definice infrastruktury v souboru typu *yaml*. Avšak ani to není zcela triviální a čitelný zápis. Existuje další alternativa, respektive nadstavba nad CloudFormation a to sice nástroj s názvem **AWS CDK**, která dokáže infrastrukturu definovat na úrovni několika programovacích jazyků - například i TypeScriptu a tedy ještě více programátorům zjednodušit práci s infrastrukturou.

4.4.1 AWS CDK

AWS CDK je framework pro definování cloudové infrastruktury v kódu, který je postaven na AWS CloudFormation. Pomocí AWS CDK lze definovat cloudové zdroje ve známém programovacím jazyce. AWS CDK podporuje **TypeScript**, JavaScript, Python, Javu a C# (.Net) [41].

Z definovaného kódu pomocí jazyka jako TypeScript se automaticky vytvoří *yaml/json* soubor, který slouží jako vstup do služby **CloudFormation**, která automaticky vytvoří či změní infrastrukturu a její propojení podle zadané definice.

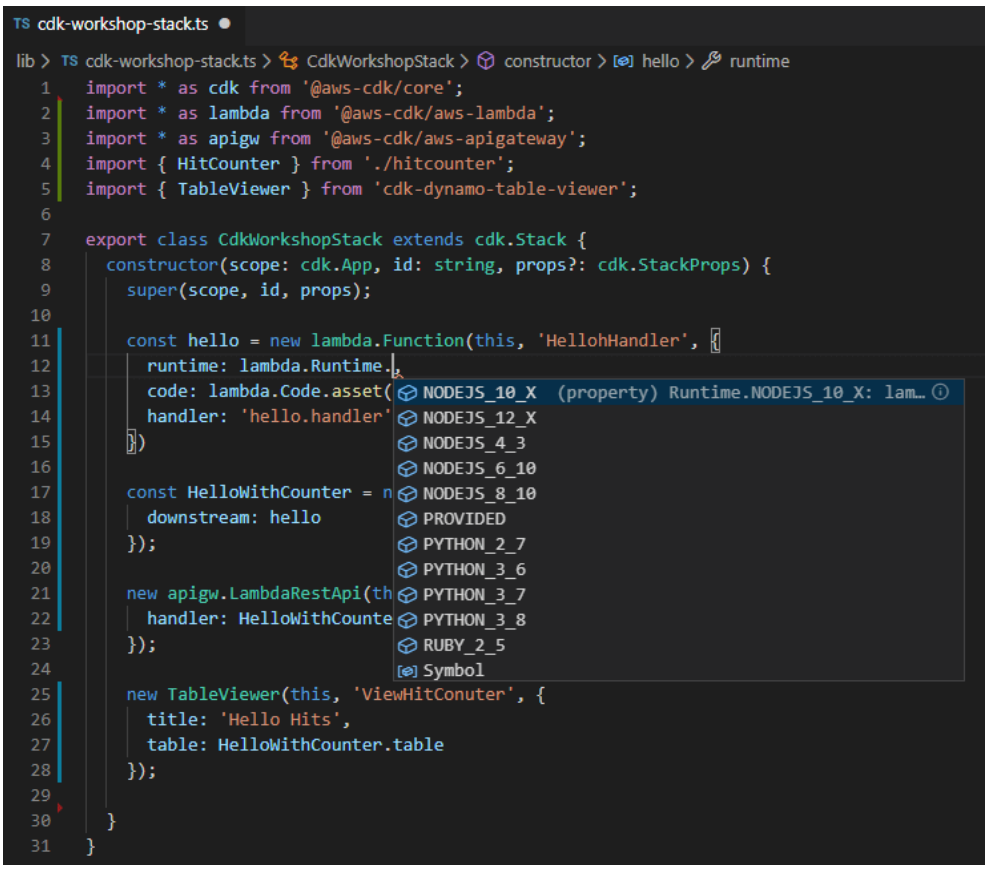
Lze tak vytvářet vysoce spolehlivé, znovupoužitelné a nákladově efektivní aplikace v cloudu bez obav z manuálního vytváření a konfigurace AWS infrastruktury.

Síla AWS CDK není pouze znovupoužitelnost a čitelnost, AWS CDK totiž nabízí mnoho zajímavých vylepšení, které dokáží zefektivnit práci uživatelů. Z knihovny AWS CDK lze v programovacím jazyce importovat mnoho tzv. **konstruktů** (construct), které v sobě mohou obsahovat už více integrovaných služeb. Další konstrukty si může uživatel vytvořit sám a využít je třeba v dalším projektu.

Konstrukty se skládají do tzv. **zásobníku** (stack) a celý takovýto zásobník, pak lze nasadit. Nicméně i do zásobníku lze přijímat vstupní atributy a udělat ho tak obecný a znovupoužitelný. Typická znovupoužitelnost je na úrovni **více vývojových prostředí** (development, stage, produkce).

Obecně CDK velmi pomáhá s integrací jednotlivých komponent, konkrétně v TypeScriptu je knihovna CDK velmi dobře typově pokrytá a zdokumento-

vaná. Uživatel tak jasně ví, co může u jaké služby nastavit a také to jaké služby do sebe může integrovat.



Obrázek 4.4: Ukázka AWS CDK, na které lze vidět jak vypadá vytváření zásobníku, importování konstruktů, napovídání parametrů či integrace jednotlivých služeb (převzato z [41])

Prostřednictvím AWS CDK není nutné definovat pouze architekturu, ale také prostřednictvím tohoto frameworku **aktualizovat zdrojový kód** - například v Lambda funkcích. CloudFormation automaticky detekuje změnu a aktualizuje pouze ty služby, které je nutné aktualizovat.

Nicméně i zde je potřeba zmínit stinné stránky. Proces nasazení trvá jednotky minut (záleží na počtu služeb). To se nemusí zdát mnoho, ale pokud vývojář cloudové architektury nemá jasnou představu o tom, co chce, ale potřebuje iterativně zkoušet různé řešení, tak není tato služba z tohoto pohledu ideální. Mezi další stinné stránky lze zařadit to, že ne vždy je každá úprava v kódu infrastruktury automaticky aplikovatelná - typicky u mazání služby může dojít k chybě. Je pak někdy nutné ručně smazat nějakou konkrétní

službu, v krajním případě smazat celý stack a vytvořit ho znovu.

V rámci realizovatelného prototypu bude framework AWS CDK dobrý pomocník, který dokáže definovat navrženou architekturu, včetně AWS Step Functions a databáze. Taktéž pomocí CDK bude snadno možné aktualizovat zdrojové kódy jednotlivých Lambda funkcí.

Kapitola 5

Návrh a implementace prototypu

V této kapitole bude přiblížen návrh a implementace konkrétního prototypu, který bude splňovat zadání, bude vycházet z definice problému, analýzy a návrhu technologií.

V první části této kapitoly bude popsán proces **výpočtu doporučení**, který bude opřen o algoritmy shlukové analýzy. V této sekci bude také popsána tvorba **scénářů**, které dokáží více konkretizovat výsledky shlukové analýzy. Poslední část této kapitoly bude věnována tvorbě **backendové služby**, která dokáže přijímat IoT senzorická data a poskytovat doporučení.

5.1 Výpočet doporučení

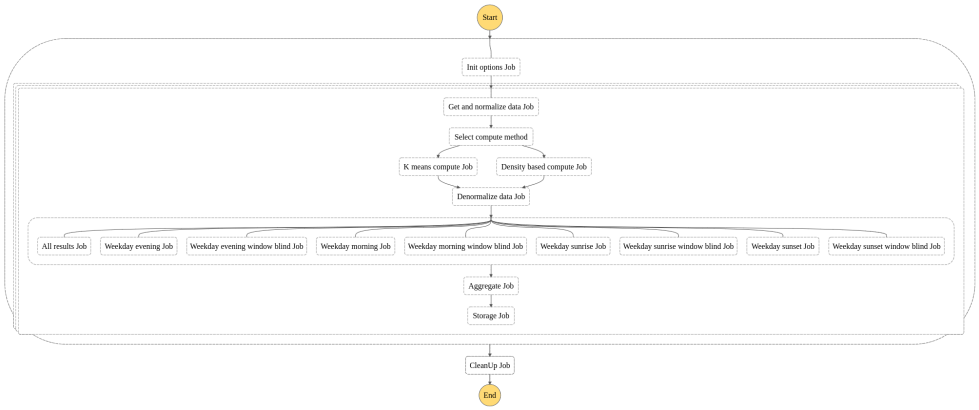
Výpočet obecného doporučení vychází z procesu **získávání znalostí z databází**, tedy z toho, že tento proces je rozdělen do několika kroků, které na sobě vzájemně závisí - výstup jednoho kroku je vstupem do kroku následujícího.

Zjednodušená idea je ta, že záznamy jednotlivých IoT zařízení budou sloužit jako vstup do **shlukové analýzy** - vybrané atributy budou sloužit jako dimenze. Tam kde vznikne shluk bude možné pozorovat události, které spolu souvisí - například se dějí v **podobný čas** v rámci dne. Z tohoto shluku budou vybrány zástupci každého unikátního zařízení a jejich typické dimenze tedy atributy, kterých dosahují. Sada takovýchto transformovaných shluků bude sloužit jako podklad pro **doporučení** - doporučení toho, aby si uživatel takovou operaci zautomatizoval, protože ji dělá opakovaně manuálně. Takovýto proces se bude aplikovat **odděleně na jednotlivé domy**.

V této části budou postupně odkryty jednotlivé kroky navrženého procesu. Konkrétně z pohledu návrhu a z pohledu implementačních detailů. Jak bylo zmíněno v návrhu architektury, jednotlivé kroky procesu tvorby doporučení

budou nasazeny jako samostatné **Lambda funkce** a budou orchestrovaný pomocí Step Functions.

Pro lepší orientaci nad celým procesem je dobré nahlédnout na obrázek 5.1, který zobrazuje jednotlivé kroky, které budou postupně popisovány. Kroky nebudou popisovány přesně od zhora dolů, ale spíše tak jak dávají myšlenkově za sebou smysl.



Obrázek 5.1: Celý proces tvorby doporučení (vytvořeno pomocí AWS Step functions diagramu)

5.1.1 Získání a normalizace dat

Jak již bylo zmíněno v kapitole s definicí problému, jsou k dispozici historická data z MongoDB, konkrétně ze dvou kolekcí - **DeviceRecords**, která obsahuje jednotlivé logy IoT zařízení a druhá kolekce **Homes**, která obsahuje informace o domech a jejich polohách (ve skutečnosti jsou to kvůli anonymizaci údajů polohy obce podle PSČ, ale taková přesnost je pro tuto práci dostatečná).

Jelikož jsem dospěl v rámci analýzy a návrhu technologií k volbě databáze v podobě AWS DynamoDB je nezbytné nejdříve zmigrovat tyto historické data do **DynamoDB** (později budou data přijímaná přes API). Před migrací je nutné rozmyslet jaké atributy a v jakých formátech budou data ukládána. Další části rozebírají návrh a realizaci **odvozených atributů** a **výběru atributů obecně**, následně pak i jejich **normalizaci**. Poslední část je věnována **komentáři** k této sekci.

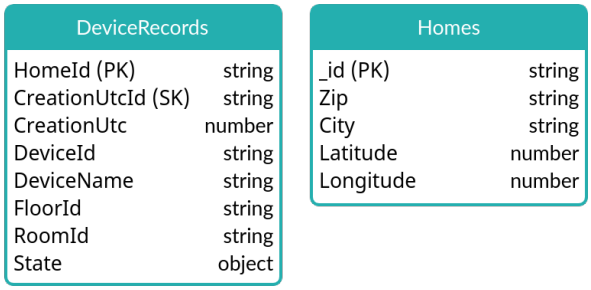
5.1.1.1 Ukládání senzorických IoT dat v DynamoDB

Nejprve je nutné vytvořit DynamoDB tabulky, pro ukládání **senzorických dat** (DeviceRecords) a **domů** (Homes). Jak již bylo zmíněno v návrhu technologií u DynamoDB není nutné při tvorbě tabulek definovat jednotlivé atributy a jejich typy. Avšak je nutné definovat primární klíč.

Jedna alternativa jak definovat unikátní primární klíč by byla ta, že by se přepoužil unikátní atribut id z MongoDB kolekce. Nicméně lze zvolit i chytřejší primární klíč, tak aby **rychlost čtecích operací** nad tabulkou dosahovala lepších výsledků. V plánu je tvořit doporučení pro jednotlivé domy odděleně. Odděleně se tedy i k datům bude přistupovat a bude dobré když data od jednoho domu budou **fyzicky uloženy u sebe**. Toho lze dosáhnout tak, že budou mít data stejnou hodnotu *partition key*.

Dále bude chtěné mít data **seřazená podle času** vykonání. Bude tedy výhodné, když druhá část primárního klíče bude tvořena pomocí *sort key*. V sort key bude uložen číselný timestamp času. Nicméně kdyby byly vykonány dvě události ve stejném domě a ve stejný čas, tak by primární klíč nebyl unikátní. Proto bude sort key složenina typu číselný timestamp a unikátní id (převzaté z původní MongoDB kolekce).

Tabulka pro uložení domů, bude kopírovat vstupní MongoDB kolekci a bude přepoužit převzatý primární klíč (id).



Obrázek 5.2: Schéma DyanomDB databáze pro ukládání záznamů zařízení a domů

5.1.1.2 Odvozené atributy

Kromě atributů, které jsou k dispozici, lze vytvořit i atributy nové, takové které se odpovodí od stávajících atributů.

První takový atribut bude jen jiný pohled na čas. Mám totiž k dispozici atribut CreationUtc, ve kterém je timestamp, tedy zakódovaný datum a čas. Nicméně pokud budu chtít hledat události, které uživatel dělá ve stejný čas (například vytahuje žaluzie každý den v 8:00), tak je potřeba dívat se jen na tu část timestampu, která udává **čas** (nikoliv datum). A aby se s tímto atributem dobře pracovalo, tak je dobré převést si ho na unifikovanou jednotku. Zvolil jsem **transformaci na sekundy**.

Podobný problém chci řešit pro datum. Z datum lze odvodit řadu metadat, které mohou být postupně pridávány jako nové atributy. V současné době mi po krátké analýze dával smysl atribut s názvem **typ dne**, který může nabývat hodnot **všední den**, **víkend** nebo **sváteční den**.

Další dva nové atributy jsou **čas východu** a **západu slunce**. Konkrétně tyto atributy udávají rozdíl východu, respektive západu slunce od času události (CreationUtc) v ten den kdy se událost stala. Tyto atributy zjistím tak, že se přes HomeId podívám do tabulky s domy, z které si vezmu polohu - **Latitude** a **Longitude** v kombinaci s **datumem** (z CreationUtc) a s těmito parametry zavolám knihovnu připravenou z návrhu technologií, která mi dané časy západu a východu slunce vrátí. Podobně jako u prvního časového atributu převedu rozdíl do sekund.

5.1.1.3 Výběr atributů

V datech jsou k dispozici atributy *FloorId* a *RoomId*, která udávají v jakém patře, respektive místnosti bylo zařízení, které logovalo událost umístněné. Avšak v systému jsou tyto dva parametry nepovinné, a tak v datech u většiny záznamů tyto atributy nejsou vyplněné. V tuto chvíli jsem je tak neuvažoval. Dalším dostupným atributem je *DeviceName*, to je ale spíše informativní atribut pro uživatele. Pro stroj je vhodnější atribut *DeviceId*, který jasně definuje zařízení.

Celkově jsou tak použity tyto atributy:

- **CreationUtc** - čas v rámci dne
- **DayType** - typ dne (všední den, víkend nebo svátek)
- **DeviceId** - identifikátor zařízení
- **State** - stav ve kterém bylo zařízení
- **SunriseDiffTime** - rozdíl sekund od východu slunce
- **SunsetDiffTime** - rozdíl sekund od západu slunce

Tyto atributy tak tvoří dimenze pro shlukovou analýzu. Nicméně než budou použity pro samotnou shlukovou analýzu, je nejdříve nezbytné tyto dimenze normalizovat.

5.1.1.4 Normalizace atributů

Normalizace je proces, kdy se snažím všechny dimenze dostat do datového typu čísla v rozmezí od nuly do jedničky. Některé knihovny to umí řešit automatizovaně, zde jsem si to naprogramoval sám.

CreationUtc, SunriseDiffTime, SunsetDiffTime
Tyto atributy už mám definované jako číselnou hodnotu. Nyní pouze provedu její **normalizaci mezi 0 a 1**. Toho docílím tak, že dané číslo vydělím maximálním počtem vteřin v rámci dne.

DeviceId, DayType
Toto jsou tzv. **kategorické proměnné**. Udělám to tak, že si všechny unikátní dostupné hodnoty vložím do pole a u každého prvku pole použiju jeho **index** (tedy například DeviceId *5c503b194f89eb0001094fa5* převedu na číslo *42*). Následně dané čísla převedu opět do rozsahu 0 až 1. Toho docílím vydělením *délky pole - 1*. Nicméně zde je dobré si uvědomit, že ty čísla, která jsou blízko u sebe mají bližší také vzdálenost, která se následně bude počítat ve shlukové analýze. Abych minimalizoval tento problém, tak před vložením kategorických proměnných do pole provedu náhodné seřazení prvků. Problém sice zůstává, ale jelikož se proces bude pouštět vícekrát, tak se tím alespoň zmírní dopat stejných výsledků.

State
Trochu složitější situace je u stavu, jelikož každý typ (DeviceType) má jiné atributy pro definici stavu. Kupříkladu obyčejné binární světlo nabývá hodnoty: zapnuto (true) a vypnoto (false), ale existují třeba i světla s intenzitou, ty nabývají hodnot 0 až 100. Dále existují i zařízení, které pro definici stavů potřebují **více složek**, například barevné světlo - složky Red, Green, Blue (každá složka číselná hodnota od 0 do 255).

Obecný postup u jednoprvkových stavů je jednoduchý, převedu je na **číslo** (pokud už číslo nejsou) a pak dané číslo **vydělím maximálním číslem**, kterého může tento atribut nabývat.

U atributů složených z více částí jsem zvolil postup následující. Jednotlivé složky převedu na čísla (pokud je třeba) a následně z nich vytvořím jedno číslo přes **bitové posuny** jednotlivých složek (po 8 bitech). Následně provedu vydělení maximálním číslem.

5.1.1.5 Přidání vah jednotlivým atributům

Ukázalo se jako výhodné mít možnost **nastavovat váhy** jednotlivým atributům (dimenzím). To znamená, že čím vyšší váhu atributu nastavím, tím víc bude tento atribut **prioritnější**, tedy že se budou shlukovat objekty více na základě této dimenze.

To se ukázalo jako extrémně výhodné u času. Tím, že se zvýšila váha tohoto atributu, tak bylo možné hledat shluky, které obsahovaly objekty (události), které se staly v podobný čas (např. rozdíly pouze v řádu minut), což byla žádaná vlastnost.

Dále je zapojení váhy výhodné v tom ohledu, že když některé dimenzi nastavím váhu na **nulu**, tak nebude tato dimenze brána v potaz - bude pro konkrétní instanci běhu shlukové analýzi **ignorována**. To se hodí například pro situaci, kdy v jedné instanci hledám shluky primárně podle času a v jiné instanci podle rozdílu od západu slunce.

Váhu realizuji jednoduše tak, že normalizované čísla (od 0 do 1) vynásobím váhovou hodnotou. Například pokud chci dát jedné dimenzi váhu 10, tak budu mít na této dimenzi čísla v rozmezí od 0 do 10.

5.1.1.6 Komentář

Všechny výše zmíněné operace v rámci tohoto kroku, tedy selekci, odvození nových atributů, transformaci, normalizaci a přidávání vah dělám za běhu.

Alternativou by bylo některé kroky dělat už **dopředu** a ukládat si je do databáze, například si to dovedu představit u odvození nových atributů, ale jelikož se jedná o operace, které jsou výpočetně rychlé, tak nebyla potřeba k tomuto kroku sáhnout, nicméně je dobré to zmínit jako možnou budoucí **optimalizaci**.

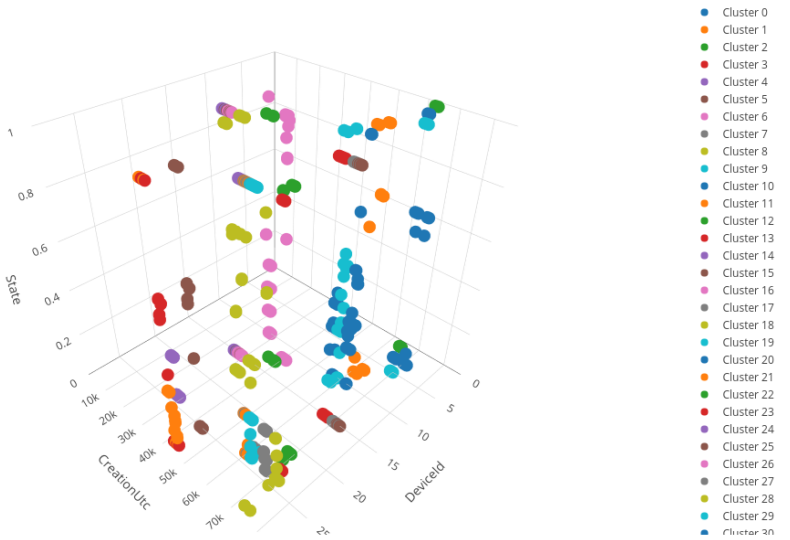
5.1.2 Shluková analýza

Připravené datové matice s jednotlivými dimenzemi slouží jako vstup do knihovny, která provádí shlukovou analýzu. Konkrétně je využito metody **k-means** a **DBSCAN**. Vize je použít obě metody a dále pracovat s výsledky obou z nich (bude popsáno dále jak).

Kromě datové matice existují další parametry, které slouží jako vstup do metod shlukové analýzy. Jak bylo popsáno v analýze u k-means je to parametr **k**, a u DBSCAN jsou to parametry **radius sousedství** (neighborhood radius) a **počet bodů v sousedství** (points in neighborhood). Způsobů jak nastavit tyto parametry je více. Například pro k-means existuje tzv. elbow metoda [9]. Alternativou je experimentálně zkoušet zadávat různé prvky a sledovat kvalitu dat. V této práci jsem zvolil druhou zmíněnou metodu v kombinaci s nastavováním vah.

Výsledný tvar obou metod je stejný. Je to pole shluků, kde každý shluk obsahuje několik položek, které jsou definovány pomocí referencí do původní datové matice.

Takto připravený výsledek šlo použít pro vizualizaci na připraveném nástroji. Nicméně bohužel šlo najednou prostřednictvím tohoto nástroje vizualizovat maximálně 3 dimenze. Způsob na pozorování výsledků tak je buď vybrat 3 nejzajímavější dimenze, případně využít metodu jako PCA nebo t-SNE [42]. Ovšem mnohdy není jasné jak vybrat ty nejzajímavější 3 dimenze. V této práci je navrženo 6 dimenzí, z čehož jsou 3 časové. Z těchto časových bude v jednom běhu pouštěna vždy jen jedna. Čili najednou budou použity 4 dimenze. Typ dne má jen 3 stavy (všední den, víkend, svátek), a tak je tato dimenze určena jako nejméně zajímavá. Ty zbylé 3 budou použity na zobrazení 3D grafu. Při rozšíření dimenzí by dávalo smysl využít pokročilejších metod jako PCA nebo t-SNE.



Obrázek 5.3: Ukázka shlukové analýzy provedena metodou k-means

5.1.3 Čištění a denormalizace dat

V tomto kroku budou provedeny dva úkoly **čištění dat** na základě chybové funkce a **denormalizace dat**, tedy převedení normalizované datové matice zpět do původních hodnot.

5.1.3.1 Čištění dat

Výsledek shlukové analýzy je v tomto kroku poslán do **chybové funkce**, která má za cíl z výsledků odebrat některé nevyhovující shluky a objekty v rámci shluků.

Ve výsledcích se někdy objevují shluky, které mají objekty hodně rozptýlené po prostoru. Když se člověk podívá, co objekty v rámci shluku reprezentují, tak to v řešení této úlohy nedává úplný smysl - například v kontextu dimenze času se mi do jednoho shluku spojí události vytvořené v 7:00 a v 9:30, což je velký rozptyl na relevantní nalezení vzoru. Jedna z cest jak donutit spojovat shluky do bližších vzdáleností je upravit vstupní parametry do shlukových metod (k; radius sousedství, počet bodů v sousedství) či zvýšit váhy jednotlivých dimenzí. Nicméně tento způsob není univerzální, protože se pak snižuje počet objektů v rámci shluku, což také není úplně žádoucí. Chce to tedy najít zlatou střední cestu, nicméně i u té zůstane část shluků více rozptýlených a tedy nerelevantních. Otázkou zbývá, jak definovat rozptýlený shluk a co s ním dělat.

Rozptýlení lze dobře deterministicky měřit na základě **variačního koeficientu**. Zjednodušeně řečeno pokud je velké množství objektů v rámci

shluku vzdáleno od středu shluku, tak je tento koeficient vyšší [43]. Pro shluk, který má tento variační koeficient vyšší, než stanovený **práh** tak je určen jako nevyhovující. Nastavení tohotu prahu je konfigurovatelná hodnota, která je momentálně nastavena na hodnotu *0.25*. Toto číslo bylo zvoleno experimentováním a je snadno měnitelné.

Další jednoduché označení nevyhovujících shluků je v podobě **nízkého počtu objektů** ve shluku. Nízký počet nemusí znamenat nutně něco špatného, obzvláště v případě, pokud má shluk nízký variační koeficient. Avšak pokud chceme tvořit uživateli chytré domácnosti doporučení na základě toho, že něco dělá opakovaně, tak je dobré definovat alespoň nějaký práh pro počet objektů ve shluku. V současné době je nastaven tento práh na *10*.

Nevyhovující shluky aktuálně zcela filtruji z původních výsledků. Alternativní metodou by mohlo být nefiltrovat tyto shluky, ale pokusit se je nějakým způsobem rozdělit a ty malé sloučit, například opětovným spuštěním metody.

Uvnitř shluků se tak vyskytuje řada objektů. Objekty značí jednotlivé události, které byly zalogované zařízením (v nějaký čas, s nějakým stavem atd.). V takovémto shluku se dle očekávání objevuje jedno konkrétní zařízení (identifikované podle DeviceId) vícekrát v kombinaci s jinými zařízeními, které se také objevují několikrát - například kolem času 17:00 je vytvořen shluk, který má 55 objektů a obsahuje *zařízení žaluzie s id 1 a počtem opakování 30* a *zařízení světlo s id 2 a počtem opakování 25*. To je zcela validní a dobrý výsledek, který může sloužit jako podklad pro doporučení, avšak někdy se do takového správného shluku připojí i jiné zařízení, které vzniklo spíše náhodou, než, že se dá vyhodnotit jako opakující vzorec.

Nad vyhovujícími shluky je tak provedena druhá část čištění, která má za cíl identifikovat takovéto nevyhovující objekty uvnitř shluků. Jednoduše je stanoven práh pro minimální výskyt zařízení (identifikováno podle DeviceId) uvnitř shluku. Pokud je výskyt nižší jak práh, tak je toto zařízení odebráno ze shluku.

5.1.3.2 Denormalizace dat

V tomto kroku chci objekty ve shlucích převést z normalizované podobny do té **denormalizované**, tedy mít ve shluku objekty v podobě, jak vypadaly před aplikací normalizace.

Nejprve jsem měl vytvořenou denormalizační funkci, která fungovala na principu jako ta normalizační (dělení váhou dimenze, násobení maximální možnou hodnotou, případně bitové posuny u vícesložkových stavů). Avšak zde se ukázal problém v podobě práce s desetinnými čísly. Při normalizaci, kdy se počítala hodnota mezi 0 a 1 musela být hodnota mnohdy zaokrouhlena na maximální přesnost JavaScriptu. Tedy mohlo se stát, že normalizované číslo bylo na základě **zaokrouhlování** denormalizováno na jiné číslo než původní.

Vyřešil jsem to tak, že jsem si v rámci lokální paměti uložil **hash** objektů v podobě před normalizací a když jsem objekty chtěl opět denormalizovat, tak

jsem jednoduše použil přidělený index do hashovacího objektu, který vrátil denormalizovanou hodnotu.

5.1.4 Agregace uvnitř shluků

V tomto kroku existují vyhovující shluky, které jsou denormalizované. V každém shluku je několik objektů, které reprezentují několik unikátních zařízení. Například existuje shluk, který má 55 objektů a obsahuje *zařízení žaluzie s id 1 a počtem opakování 30* a *zařízení světlo s id 2 a počtem opakování 25*. Cílem tohoto kroku je provést agregaci podle unikátního identifikátoru zařízení - tedy zde podle *DeviceId*.

Výsledkem výše zmíněného příkladu by tak bylo doporučení, které by obsahovalo dvě zařízení: *žaluzie s id 1* a *světlo s id 2*. Nyní je třeba definovat jakým stylem se doplní k jednotlivým zařízením hodnoty atributů (čas, stav, atd.). Je tedy třeba u každé dimenze definovat jak vytvořit z pole hodnot jednu výslednou hodnotu.

U **kategorické dimenze** (DayType neboli typ dne) je řešení vcelku přímočaré v podobě **modusu** neboli nejčastěji vyskytující se hodnotě [44]. Tedy pokud je například na vstupu dimenze 53x použit typ dne *všední den* a 2x *víkend*, tak je vybrána hodnota *všední den*.

U dimenzí reprezentující **čas** (CreationUtc, SunriseDiffTime, SunsetDiffTime), mi dával smysl použít spíše **aritmetický průměr** jednotlivých hodnot. V kombinaci s tím jsem využil **zaokrouhlování** k nejbližším deseti minutám.

Nejtěžší volba byla u dimenze reprezentující **stav** (State). Představme si například zařízení žaluzie, které lze vytahovat a toto zařízení u toho nabývá stavu v rozsahu *0-100*. Ve shluku se objeví *9x* vytáhnutí do úrovně *100* a *1x* do úrovně *80*. Aritmetický průměr by tak vypočítal hodnotu *98* a modus hodnotu *100*. Momentálně byl využit **modus** a toto nastavení je opět jednoduše konfigurovatelné. Pokud by byl stav vícesložkový, tak by byl proveden modus na každou dílčí složku.

U každého takto agregovaného shluku byly ještě poznačeny informace o tom jak byl tento shluk původně **velký** (ClusterSize), kolik má unikátních **zařízení** (DeviceCount) a kolikrát byl v průměru **počet opakování** na unikátní zařízení (RepeatPerDevice = ClusterSize / DeviceCount). Tyto atributy budou dále sloužit pro **prioritizaci výsledků**.

Takto agregované shluky lze tedy považovat za **doporučení**. Níže lze vidět reálnou ukázku doporučení, které vzniklo z metody DBSCAN. Ukázka obsahuje informace o **čase**, **typu dne** a o **zařízeních**, kterých se to týká - tedy zde zařízení *žaluzie*, které se dá do polohy *posun (Shift)* a *náklon (Tilt)* do hodnoty *100*. To znamená, že žaluzie zatemní okno. V doporučení je dále zařízení definující *světlo*, které se dá do úrovně *intenzity 100*, to značí plně rozsvícené světlo. Výsledek je tak při zamyšlení relevantní - uživatel často večer manuálně zatemňoval okno a rozvícel světlo, tak mu to doporučme zautomatizovat.

U výsledků jsou dále i jiné atributy, které budou sloužit pro reprezentaci, prioritizaci a filtrování. Tyto aspekty budou částečně variabilní a budou probíhat na úrovni dotazů přes REST API. O této problematice bude proveden detailnější popis později.

```
{
    "homeId": "5c98b3115b4a330001521a27",
    "type": "time",
    "computeMethod": "densityBased",
    "clusterSize": 81,
    "deviceCount": 2,
    "repeatPerDevice": 40.5,
    "time": "19:30:00.000+00:00",
    "dayType": "weekday",
    "devices": [
        {
            "DeviceId": "5c503be04f89eb0001094faa",
            "DeviceName": "Zaluzie 6",
            "DeviceType": "windowBlind",
            "State": {
                "_t": "WindowBlindState",
                "Position": {
                    "Shift": 100,
                    "Tilt": 100
                }
            }
        },
        {
            "DeviceId": "5c4f5ddd494bdc0001173eb2",
            "DeviceName": "Jidelna",
            "DeviceType": "dimmableLight",
            "State": {
                "_t": "DimmableLightState",
                "Intensity": 100
            }
        }
    ]
}
```

Zdrojový kód 5.1: Ukázka vytvořeného doporučení na základě shlukové (DBSCAN) analýzy

5.1.5 Rozdělení výsledků - tvorba scénářů

Takovýchto doporučení, která byla popsána výše může pro jeden dům vzniknout celá řada. Při pohledu na data jsem ovšem nalezl nějaké opakující se vzorce, které se prolínaly napříč několika chytrými domácnostmi.

Mezi konkrétní poznatky patří:

- lidé vykazují opakující se aktivity **ráno** a ve **večerních hodinách** - pravděpodobně před odchodem, respektive příchodem z práce

- lidé vykazují opakující se aktivity ve **všední dny**; víkendy a svátky jsou více dynamické a nepředvídatelné - o víkendech a svátcích lidé pravděpodobně více opouštějí svůj domov

- spoustu událostí se děje v čase blízkém **východu** a **západu slunce** - lidé pravděpodobně reagují na změnu intenzity osvětlení

- velmi časté zařízení, které nemají lidé zautomatizované a používají ho tedy manuálně, ale zároveň s opakujícím se vzorcem, je zařízení **žaluzie**

- naopak zařízení, které lidé téměř manuálně nepoužívají je **regulace teploty** - pravděpodobně si teplotu lidé rychle vyladí a zautomatizují a pak nemají velkou potřebu dělat manuální regulace

Z nabitých poznatků vznikl návrh scénářů, které rozdělí výsledky do více skupin. Výsledky mohou být i ve více scénářích zároveň.

Návrh scénářů vypadá následovně:

- **všechny výsledky** - tento scénář jen zpropaguje všechny příchozí výsledky (to je vhodné pro budoucí zpětné rozšíření)

- **večerní všední dny** - redukce na výsledky s časy 15:00 - 23:00

- **večerní všední dny** s alespoň jedním zařízením typu **žaluzie**

- **ranní všední dny** - redukce na výsledky s časy 04:00 - 12:00

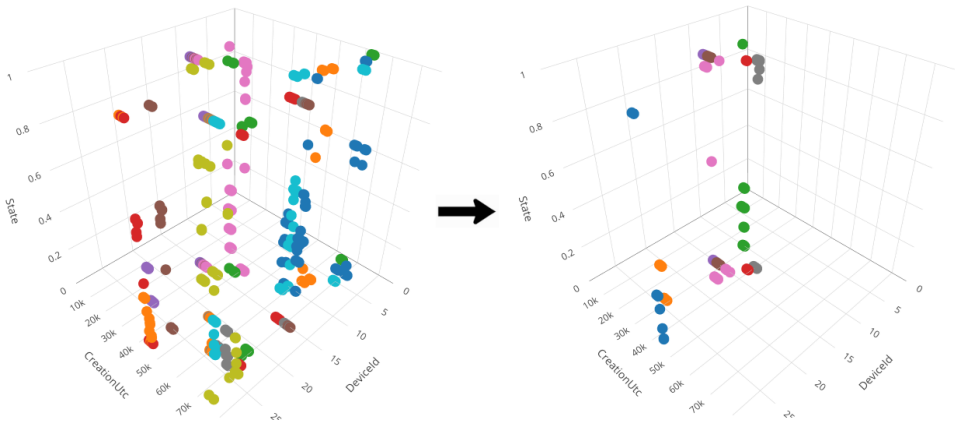
- **ranní všední dny** s alespoň jedním zařízením typu **žaluzie**

- **období kolem východu slunce** - výsledky s časy plus/mínus 1 hodina od východu slunce

- **období kolem východu slunce** s alespoň jedním zařízením typu **žaluzie**

- **období kolem západu slunce** - výsledky s časy plus/mínus 1 hodina od západu slunce

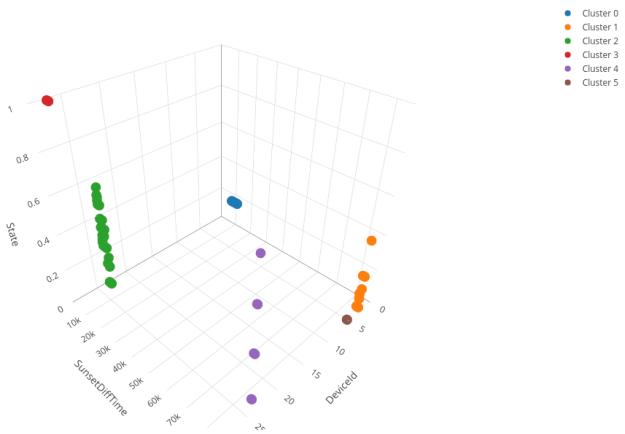
- **období kolem západu slunce** s alespoň jedním zařízením typu **žaluzie**



Obrázek 5.4: Ukázka scénáře, který z objektů vybere ty, které jsou ve všední dny a v ranních hodinách

Obrázek 5.5 ukazuje scénář pro výběr událostí, které se uskutečnily **blízko západu slunce**, tedy s rozdílem plus nebo mínus jedna hodina od této události - mínusová hodnota na časové ose je zobrazena jako odečet od maximální hodnoty, proto jsou objekty na grafu vidět jak na začátku časové osy, tak i na konci.

U scénářů s východem a západem slunce mažu originální (všechny) výsledky, protože samy o sobě nedávají smysl - zde je nutné pozorovat jen události, které se dějí blízko východu a západu slunce.

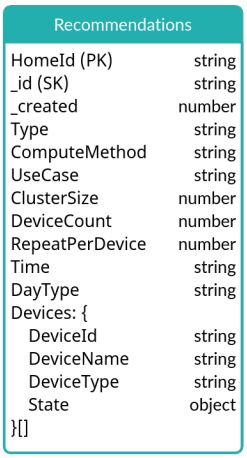


Obrázek 5.5: Ukázka scénáře, který vybírá jen události blízko západu slunce

5.1.6 Uložení doporučení

Doporučení, jehož struktura byla prezentována v rámci 5.1, je v tomto kroku uložené do DynamoDB databáze. Toto doporučení už jen bylo obohaceno o informaci, z kterého **scénáře** (useCase) pochází a **kdy bylo uloženo** (created).

Návrh primárního klíče opět reflektuje myšlenku, že je dobré, aby byly **doporučení pro jeden dům fyzicky u sebe** z důvodu rychlejšího čtení. Toho je docíleno přes použití identifikátoru domu (HomeId) jako *partition key*. *Sort key* je zde vygenerované id, tím je zaručena unikátnost primárního klíče.



Obrázek 5.6: Schéma DynamoDB databáze pro ukládání doporučení

5.1.7 Kompletace procesu

K takto připraveným krokům na začátek přibude ještě jeden krok, který samotný proces **inicializuje** a jeden krok, ve kterém se **vyčistí** mezivýsledky jednotlivých kroků.

5.1.7.1 Inicializace

Každá inicializace procesu bude mít svoji **konfiguraci**, ve které je možné definovat všechny výše zmíněné parametry - jako *váhy dimenzí*, *parametry shlukové analýzy* (k, radius sousedství, počet bodů v sousedství), *parametry filtrace shluků* (maximální variační koeficient, minimální velikost shluku, minimální opakování zařízení), *výběr metody pro shlukovou analýzu* (k-means, DBSCAN), *seznam scénářů*, které budou použity (lze jich použít více najednou) a *identifikátor domu*, pro který vzniknou doporučení.

Dále je do nastavení přidáno, jaký *typ doporučení* z konfigurace vznikne (čas, východ slunce, západ slunce) - to je jen pomocný příznak pro budoucí filtrování. Ke konfiguraci je dále možné nastavit *filtry typů zařízení*, tedy to, že chci doporučení hledat jen nad určitou podmnožinou zařízení.

Další nastavitelné hodnoty jsou **počet záznamů**, nad kterými bude vznikat shluková analýza a jaká bude **frekvence spouštění**.

Před samotným spuštěním tvorby nových doporučení probíhá **mazání starých doporučení**. Ty už totiž přestávají být relevantní, protože se vytvoří nové na základě novějších dat.

5.1.7.2 Čištění mezivýsledků

Jak již bylo zmíněno dříve, menší nevýhoda Step Functions je tak, že velikost vstup/výstupních dat je **omezena**, tudíž nelze pro můj případ užití použít to, že výstup jedné Lambda funkce bude vstupem do funkce druhé. Avšak řešení není složité, ukládám si jednotlivé mezivýsledky na službě **AWS S3** a v rámci vstup/výstupních operací uvnitř Step Functions si předávám referenci na mezivýsledek.

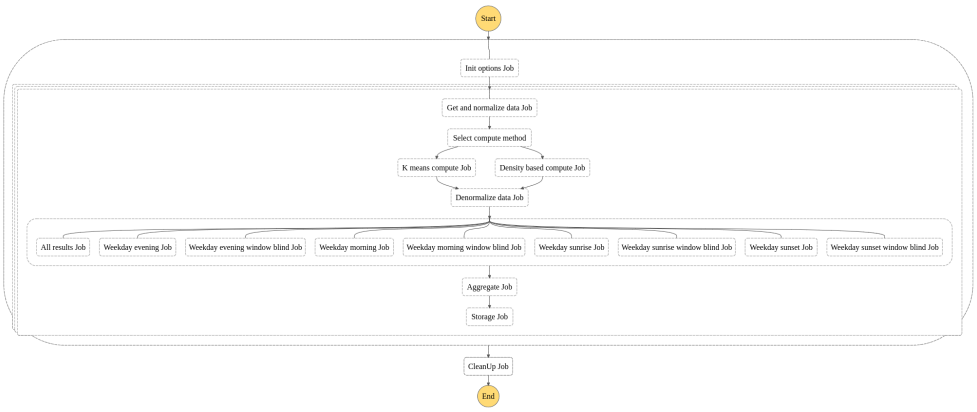
Na konci všech běhů pro tvorbu doporučení je spuštěn ještě jeden krok, který veškeré mezivýsledky z S3 smaže. Tím se tedy provede úklid a nastává **konec výpočtu**.

5.1.7.3 Propojení kroků

Každý krok je nahrán do samostatné Lambda Funkce a Step Functions se stará o orchestraci. CloudWatch rule vyšle událost, která nastartuje první krok v Step Functions, tímto krokem je inicializace. V inicializaci se tedy vytvoří veškeré konfigurace. Vznikne pole konfigurací.

Všechny tyto konfigurace jsou **paralelně spuštěny** a každá zahájí vlastní instanci procesu. Nejprve je spuštěna Lambda funkce, ve které probíhá *získání a normalizace* dat. Dále **probíhá tok** Step Functions kroků **na základě zvolené** výpočetní metody shlukové analýzy. Tok se po té spojí a provede se *denormalizace* dat. Pro všechny nadefinované scénáře je potom **paralelně** provedeno samotné vykonaní scénářů. Pokračuje se *agregací* dat, *uložením samotných doporučení* a závěr patří *čištění mezivýsledků*.

Celý proces Step Functions, včetně jednotlivých paralelních větví lze sledovat na **interaktivním diagramu**, který je automaticky vygenerován v rámci webové aplikace AWS. Tento diagram se postupně začne rozsvěcet zeleně jak v něm putuje tok kroků. U každého kroku lze vidět jaký byl vstup a jaký byl výstup Lambda funkce. Dále jsou zde vidět další logy, které vzniknou uvnitř jednotlivých funkcí. Také zde můžeme nalézt časy jednotlivých kroků, to nám může dát dobrý obrázek toho, kde jsou pomalé místa (a lze na to reagovat např. vertikálním škálováním).



Obrázek 5.7: Celý proces tvorby doporučení (vytvořeno pomocí AWS Step functions diagramu)

V procesu tvorby doporučení tak existuje velké množství **paralelizací**, **automatického škálování** výpočetního výkonu a v případě pádu některého kroku je tento krok automaticky **zopakován**. To vše bez nutnosti nastavení serveru.

Proces je nastaven tak, aby byl dobře **rozšířitelný** - další metody shlukové analýzy, scénáře či dimenze je velice jednoduché přidat či změnit. Další benefit návrhu je v tom, že jsou jednotlivé kroky **izolované**, snadno by tak šlo například úplně změnit přístup k výpočtu shlukové analýzy (třeba přes jinou službu nebo jiný programovací jazyk).

Co lze z určitého pohledu považovat za nevýhodu návrhu je přesný popis dimenzí a atributů. Pokročilejší verze systému by mohla dokázat přijímat libovolné objekty, automaticky si z nich vytvořit dimenze, normalizovat je, shlukovat je a vytvořit doporučení. Nicméně v současném řešení modularizace do jednotlivých kroků alespoň přispěla k lepší **znovupoužitelnosti** - některé kroky jsou obecné a ty zbylé nejsou rozsáhlé, takže by se v nich případně jednoduše prováděly specifické změny vhodné pro jiný systém.

5.2 Přidání aplikačního rozhraní

Po dokončené části systému, která se starala o výpočet doporučení zbývá zrealizovat aplikační rozhraní a backendovou službu, která bude mít za cíl **přijímat nová IoT data** od zařízení chytrého domu a **poskytovat vypočítaná doporučení** na základě variabilních filtračních parametrů.

Na aplikační rozhraní byla použita služba *Amazon API Gateway* a jako výpočetní výkon na zpracování requestů je využita *Lambda funkce*. Architektura je taková, že do každého koncového bodu aplikačního rozhraní je připojena jedna Lambda funkce. Přesně tak, jak je to na na úvodní stránce

AWS serverless [19]. Alternativní přístup pro stavbu REST-API by mohl být využít například knihovnu express [45], která by se pak celá zabalila do jedné Lambda funkce.

V rámci tvorby API je provedena **autorizace**, **validace vstupních parametrů** a samotné vykonání requestu. API je dále nasazené v cloudovém prostředí na **nastavené doméně**.

5.2.1 Autorizace a validace requestů

Pro autorizaci bylo využito vygenerování přístupového tokenu (access token), který integrovaná služba musí použít v hlavičce dotazu v atributu *authorization: Bearer <access token>*. Pokud by byl token nesprávný, tak služba vrátí **401 Unauthorized**. Pokročilejší přístup k této problematice se dá řešit například přes AWS službu Cognito, která umí zajišťovat celý proces autentizace a autorizace (včetně například expirace a obnovení tokenu).

Navrhované API bude v jednom koncovém bodě v requestu přijímat data v těle requestu. Druhý koncový bod bude sloužit pro získávání doporučení. Zde bude možné zadat několik volitelných parametrů a některé povinné přes query parametry. Oba tyto koncové body tak pro větší robustnost systému potřebují **validaci vstupních parametrů**. Toho lze docílit například přes validační knihovnu joi [46]. V této knihovně se nadefinuje schéma jednotlivých parametrů (včetně povinnosti a volitelnosti parametrů) a následně se toto nadefinované schéma porovná se vstupními parametry. Pokud by knihovna našla chybějící povinný parametr či libovolný parametr v jiném datovém typu (či jiném oboru hodnot), tak vyhodí chybu, která se ze služby zpropaguje HTTP odpovědí **422 - Unprocessable Entity**.

5.2.2 Příjem nových IoT dat

Koncový bod, respektive Lambda funkce na zpracování nových IoT dat je vcelku jednoduchá. Tento koncový bod je pojmenovaný **/device-records** a HTTP metoda je typu **POST**. Tento koncový bod přijímá pole IoT logů. Je tedy možné přijmout více záznamů najednou, což může být výhodné při hromadném posílání dat.

V rámci zpracování requestu probíhá autorizace, validace a případné uložení do připravené DynamoDB tabulky. V případě úspěchu je uživateli vrácen HTTP kód **200 - OK** spolu s odpovědí o počtu uložených záznamů.

5.2.3 Poskytování doporučení

Tento bod je poslední částí implementace. Před samotnou implementací je nezbytné dobře promyslet, co se od služby **očekává**, jak má být používaná a jak ji navrhnout tak, aby se dobře rozšiřovala. Dále je potřeba zrekapitulovat jaké doporučení a s jakými parametry jsou v databázi uložené.

Původní myšlenka, která vzešla ze zadání byla taková, že se tvoří služba, která uživatelům chytré domácnosti doporučí **zautomatizovat události**, které dělají v domácnosti opakovaně manuálně. To je scénář, který je nutný podporovat na prvním místě. V tomto scénáři je dobré uživateli doporučovat spíše méně, ale **hodně relevantních doporučení**, abychom ho příliš nezahltili. Avšak v průběhu implementace jsem problematiku konzultovat mimojiné s technikem Home Qest, který pomáhá lidem s instalací a údržbou systému - pomáhá tak uživatelům nastavovat jejich chytré domy, včetně pomoci s automatizací událostí. Tento člověk mě navedl na jeho potřebu a to sice, že by ocenil klidně **větší množství** (a klidně duplicit) takovýchto doporučení, které by si hromadně stáhl a použil to jako podklad, který by ručně zkonzultoval s danými uživateli chytré domácnosti a to co by si uživatelé vybrali, to by jim nastavil.

Dalším faktem je, že doporučení vznikají ze stejných vstupních dat **více metodami** shlukové analýzy (k-means a DBSCAN), to ale tedy znamená to, že mohou být některé výsledky **duplicitní**. U výsledků si poznačuji mimojiné jakou *metodou shlukové analýzy bylo doporučení vytvořeno* a do jakého *scénáře* je doporučení zařazeno 5.1.5.

Z výše zmíněných poznatků mi tak vzešla idea udělat dotazování nad daty více **variabilní**, pokud o to uživatel projeví zájem. Jinými slovy, ve výchozím nastavení udělat API jednoduché, bez nutnosti zadávání mnoha parametrů, ale pokud bude konzument API chtít může si regulovat, co chce získávat.

Vstupní parametry tedy jsou:

- **Strategie (strategy)** - tento parametr specifikuje strategii dotazování v kontextu scénáře 5.1.5. Na výběr je buď *normal* (výchozí), *windowBlind* nebo *all*. Strategie windowBlind je nejvíce specifická - pokrývá všechny scénáře, které obsahují **vytahování žaluzií** (ráno, večer, při východu a západu slunce). Strategie normal vrací doporučení z **raná, večera a z období západu a východu slunce**. Poslední strategie - all vrací **všechny** doporučení.

- **Filtrace duplicit (filterDuplication)**, výchozí hodnota je *true* - jak již zaznělo, na základě více metod shlukové analýzy (či na základě kombinace více scénářů) může docházet k duplicitám. Jedna možnost by byla vybrat buď výsledky jedné nebo druhé metody, ale pokusil jsem se navrhnout sofistikovanější řešení. Pokud je tento parametr nastaven na hodnotu *true*, tak je provedena **filtrace** jako: doporučení se seřadí podle nejvíce opakování ku počtu zařízení (*repeatPerDevice*), postupně se bere doporučení; z tohoto doporučení se poznačí všechna zařízení, která už byla použita; pokud v právě zpracovaném doporučení existuje alespoň jedno zařízení, které ještě nebylo použito, tak toto doporučení také použiji ve výsledcích.

- **Identifikátor domu (homeId)** - toto je jediný povinný parametr. Doporučení se vyfiltrují podle ID chytrého domu.

Lze diskutovat jestli je to dokonalé řešení. Pravděpodobně ne nutně, ale otázka jestli ideální existuje. V případě navržené filtrace duplicit může docházet k tomu, že je zahozeno doporučení, které obsahuje zařízení například v čase 15:00, protože už bylo použito doporučení se stejným zařízením v jiném čase, např v 10:00. Nicméně pokud by si to uživatel zautomatizoval, tak už by se toto zařízení nelogovalo a v příští iteraci tvorby doporučení by dostaly prostor jiné doporučení s tímto zařízením.

Alternativně by mohlo jít to, že by se procházela doporučení hodina po hodině a daný filtr duplicit by se prováděl na této hodinové úrovni. Avšak zde zůstává otázka jestli by to bylo lepší, protože když by například existovalo jedno zařízení, které uživatel používá velmi často manuálně (například světlo na záchodě), tak by mohlo být ve výsledcích mnohokrát a zastínilo by tak doporučení od jiných zařízení.

Každopádně konzument API má stále možnost využít *strategii all* a *nevyužít filtraci duplikace (nastavit na false)* a získat tak surová, nefiltrovaná data a případnou **filtraci si provést na své straně** podle libovolné potřeby.

5.2.3.1 Nastavení domény

Pokud se nasadí API Gateway (s Lambda funkcemi) do AWS, tak poskytne AWS **zdarma automaticky vygenerovanou URL** běžící na protokolu HTTPS (včetně certifikátu). Příklad takovéto URL je `https://ua3kwyqsj7.execute-api.eu-west-1.amazonaws.com`. To je samo o sobě skvělé, protože není nutné platit za certifikát a registraci domény pokud chceme provozovat službu komunikující přes HTTPS.

Na druhou stranu, zmíněná URL není příliš reprezentativní a slouží spíše k rychlým prototypům. Každopádně AWS nabízí jednoduchou cestu k správě vlastních domén a problematice DNS obecně [47]. Připravenou doménu lze poté snadno propojit s API Gateway a provozovat potom službu na vhodnější URL, což jsem pomocí AWS CDK také udělal. Služba, která vznikla v rámci této práce je dostupná na URL: `https://home-recommender.qestapp.net`.

Kapitola 6

Testování a zhodnocení provozu

Tato kapitola bude patřit testování a zhodnocení. Testování systému probíhalo na několika úrovních. V průběhu vývoje vznikaly **jednotkové a integrační testy**, které pokryly některé důležité části služby a zvýšily tak kvalitu kódu. Dále byl proveden jednoduchý **zátěžový test**, který prověřil robustnost a škálovatelnost služby. V další části bude popsáno testování nebo spíše uživatelská zpětná vazba. Závěr bude věnován zhodnocení provozu a to konkrétně po stránce finanční.

6.1 Jednotkové a integrační testy

V průběhu vývoje vznikaly dílčí funkcionality jako různé funkce na zpracování dat, práce s časovými parametry či práce s datovými maticemi. Pokusil jsem se některé z těchto dílčích komponent pokrýt takzvanými **jednotkovými testy**, které ověřily jejich správnost.

Dále jsem vytvořil **integrační testy**, které měly za cíl otestovat integraci více dílčích komponent. Takové testy typicky provolaly API a validovaly správné chování backendové služby na základě různých vstupních parametrů.

Jak jednotkové tak integrační testy byly vytvořeny pomocí knihovny jest [48], která nabízí velmi jednoduché a intuitivní použití, ale také pokročilejší funkcionality jako je snaphotování výsledků [49].

6.2 Ladění tvorby doporučení

Do této kapitoly by se dalo zařadit i ladění tvorby doporučení, tedy procesu získávání znalostí z databáze. Není to sice klasický testovací scénář, kde je nějaký vstup a po té porovnávám očekávaný výstup, ale to u těchto problémů není ani tak docela možné ani praktické a to proto, že v tvorbě doporučení je řada náhodných prvků a tak nejsou výsledky deterministické. Dále jsou

takovéto algoritmy potřeba ladit a pozorovat lidským okem, čili sledovat jestli by mi jako reálnému uživateli dávaly takové výsledky neboli doporučení smysl.

Samozřejmě jsem měl proces získávání doporučení možnost ladit přímo v AWS, nicméně tady se projevila stinná stránka navrženého řešení, které obsahuje AWS Step Functions a nasazování přes AWS CDK a to sice, že pro aktualizaci kódu či nějaké služby trvá znovunasazení do AWS přes CDK jednotky minut (na mém stroji při všech službách zhruba 3 minuty), což je pro rychlé ladění nepraktické.

Udělal jsem tedy to, že jsem si proces získávání doporučení, který je v AWS definován pomocí Step Functions duplicitně definoval i do lokálního skriptu, napsaného v TypeScriptu. Může se to zdát jako velká duplicita, ale ve výsledku je to je pár funkcí, které se provolávají za sebou a tam kde je v Step Functions paralelismus jsem udělal cyklus přes metodu map.

Dále se doporučení ukládají do DynamoDB databáze, ve které se rychle zorientovat nad výsledky není úplně snadné, a tak jsem si napsal jednoduchou funkci, která místo do DynamoDB ukládala do souborů.

Výše zmíněné faktory mi pomohly rychle ladit a vylepšovat proces získávání doporučení. Po vyladění se tento proces nahrál do AWS a tam je orchestrován klasicky pomocí Step Functions pro reálné využití.

6.3 Vyhýbání se proprietárnímu uzamčení

Tento aspekt taktéž přispěl k tomu, že moje řešení není proprietárně uzamčené (**vendor lock-in**) na AWS serverless technologiích. Tedy kdybych se rozhodl, že nechci proces získávání doporučení realizovat pomocí *Lambda funkcí* a *Step Functions*, tak by nebyl velký problém tento připravený skript použít pro běh doporučení například na AWS virtuálních strojích (EC2) či na **jiném cloudovém poskytovateli** nebo třeba na vlastních strojích.

Když jsem si tento fakt uvědomil, tak jsem ale našel v kódu jednu technologii, která byla spjata s AWS serverless a to sice *Amazon S3*, kterou jsem používal na ukládání mezivýsledků, proto jsem dal toto ukládání mezivýsledků jako nastavitelnou vlastnost, kterou lze vypnout.

Poslední služba z AWS, která je použita je *DynamoDB databáze*, ale do této databáze se dá připojovat i kupříkladu z virtuálních nebo vlastních strojů a v případě potřeby by tato databázová vrstva šla také velmi rychle vyměnit (jsou to pouze 3 tabulky).

6.4 Zátěžové testování

Alespoň velmi základním způsobem jsem chtěl otestovat to, jestli dokáže služba **škálovat a ustát větší nárazovou zátěž**, která by mohla například nastat, kdyby se do služby nahrávaly logy IoT zařízení najednou, například

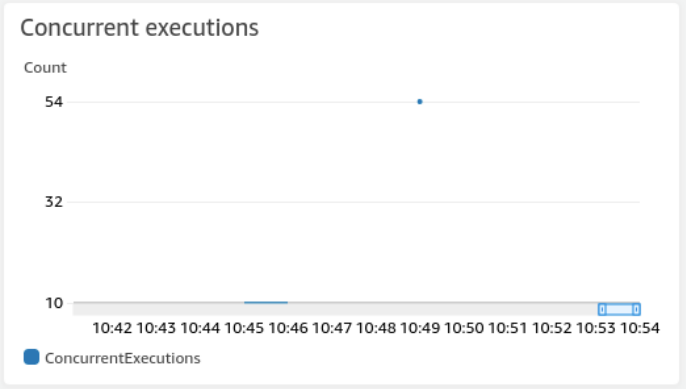
jednou za den. Napsal jsem si tak jednoduchý skript, který provolával koncový bod API pro příjem IoT dat.

Ve skriptu lze regulovat:

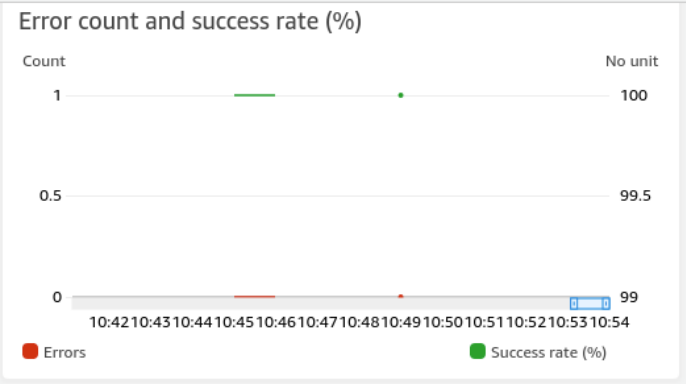
- požadovaný počet přenesených záznamů
- počet záznamů v jednom requestu
- maximální počet paralelních requestů

První test, vykonaný kolem 10:46, měl za úkol přenést 10 000 záznamů, v jednom requestu použít 100 záznamů a paralelně provádět 10 volání.

Druhý test, vykonaný kolem 10:49, měl za úkol přenést 100 000 záznamů, v jednom requestu použít 100 záznamů a paralelně provádět až 100 volání.

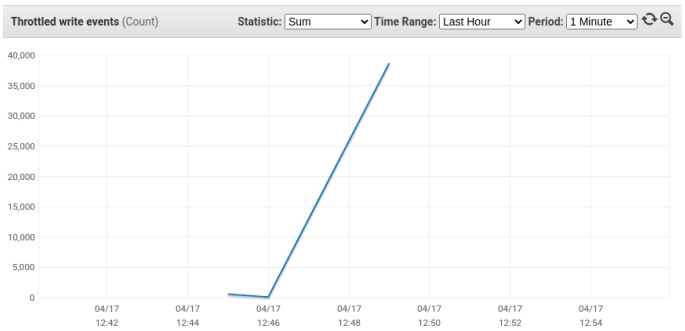


Obrázek 6.1: Zátěžový test - škálování Lambda funkcí (vytvořeno pomocí AWS Cloudwatch)

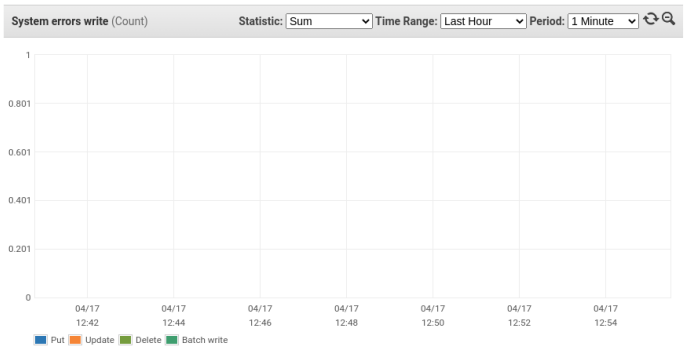


Obrázek 6.2: Zátěžový test - počet chybně zpracovaných requestů (vytvořeno pomocí AWS Cloudwatch)

Obrázek 6.1 nám ukazuje vývoj škálování, v prvním testu paralelně probíhalo 10 requestů zároveň. Ve druhém testu jsem se pokoušel provést 100 paralelních requestů, nicméně pravděpodobně můj lokální počítač nezvládal provést takové množství requestů zároveň, a tak na obrázku můžeme pozorovat 54 konkurentně vykonávaných requestů. **Ani jeden request nebyl odmítnut** (throttling) ani neskončil s chybou, to dokazuje obrázek 6.2.

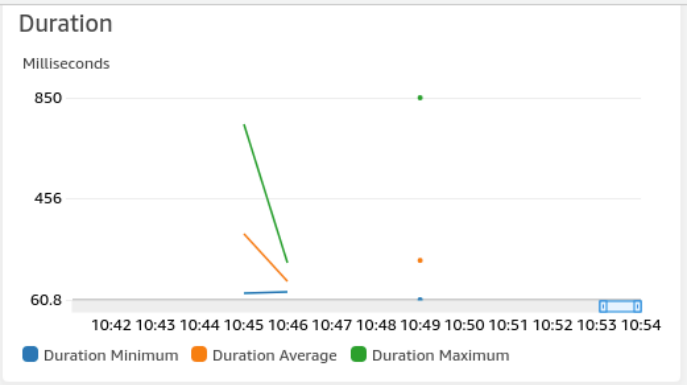


Obrázek 6.3: Zátěžový test - DynamoDB throttling (vytvořeno pomocí AWS Cloudwatch)



Obrázek 6.4: Zátěžový test - DynamoDB počet chybných zápisů (vytvořeno pomocí AWS Cloudwatch)

Avšak situace ohledně throttlingu už je horší na úrovni databáze, což můžeme pozorovat na obrázku 6.3, zde je možné vidět, že nemalé množství zapisovaných záznamů je odmítnuto. Nicméně throttling neznamená automaticky chybu. Odmítnuté zápisy se AWS SDK automaticky pokusí provést (i opakovaně) později [50], což se skutečně povedlo a není tedy zaznamenán **žádný zápis, který by skončil s chybou** zápisu, jak ilustruje 6.4.



Obrázek 6.5: Zátěžový test - délka provádění requestů (vytvořeno pomocí AWS Cloudwatch)

Poslední obrázek 6.5 nám dává náhled na to, jak dlouho trvalo vykonávání requestu. Průměr je tak někde kolem 250 ms a maxima dosahují hranice 850 ms - to je způsobené primárně již zmíněným *cold startem*.

6.5 Uživatelská zpětná vazba výsledků

V systému bylo potřeba nějakým způsobem ověřit jestli jsou **vytvořená doporučení validní**, tedy jestli dávají smysl. V první etapě jsem se pokoušel takové rozhodnutí učinit **sám**, což dokázalo zachytit nejvíce nerelevantní výsledky, ale takový subjektivní pocit samozřejmě nestačí.

V druhé etapě jsem tak vyrazil na konzultaci **za samotnými tvůrci Home Qest**, kteří se v problematice chytrých domů pohybují už několik let a už dokáží částečně rozpoznat, co si lidé obvykle v chytrém domě automatizují. Z konzultace vzešlo to, že je tam řada výsledků zajímavých, ale je jich mnoho - tam vznikla myšlenka *scénářů*, které jsem později implementoval (5.1.5). To tedy pomohlo více zacílit relevanci výsledku do obvyklých scénářů. Dále jsem po této zpětné vazbě upravil svoji *chybovou funkci*, kde jsem kromě počtu objektů ve shluku začal zohledňovat také variační koeficient.

Poslední etapa zpětné vazby proběhla od **technika Home Qest**, jehož práce spočívá mimo jiné v tom, že lidem pomáhá nastavit jejich chytré domy (včetně pomoci automatizace událostí), takže takový člověk by mohl mít nejlepší povědomí o tom co je relevantní. Jeho pohledem jsme se tak dívali na **doporučení jednotlivých domů**, z kterých projevil velké nadšení a potvrdil, že celý koncept určitě dává smysl. Ukázka těchto výsledků je v příloze B.

Dále ale jak již bylo zmíněno, projevil technik zájem o to, aby mohl z jeho pohledu vidět *více doporučení* (klidně duplicitních), což by se mu hodilo pro jeho práci, kdy lidem pomáhá nastavovat automatické události. To by tak byl

pro něho podklad, z kterého by mohl ručně vybírat. Na základě této zpětné vazby byl vytvořen variabilnější přístup k doporučením přes API (5.2.3), tak aby bylo možné získat všechna data.

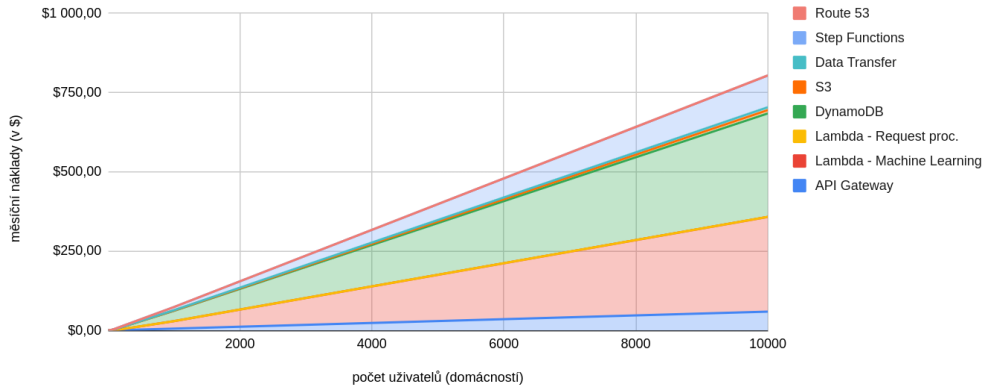
6.6 Cena provozu systému

Posledním aspektem, který je dobré zhodnotit, je cena provozu AWS služeb. Jedna z výhod AWS serverless je ta, že lze dobře **předvídat jaké budou náklady na jednoho uživatele** (zde na jeden chytrý dům) a jak náklady porostou s přibývajícím počtem uživatelů.

Lze toho docílit pomocí online nástroje od AWS s názvem pricing calculator [51]. Do tohoto nástroje lze zadat služby, které daná architektura obsahuje. K jednotlivým službám je nutné zadat predikovanou zátěž (počet requestů, uložených dat a tak dále).

Tyto hodnoty jsem nastavil pro jednu domácnost na měsíc následovně: domácnost vyprodukuje *1500 manuálních logů*, proces tvorby doporučení budu spouštět *4x do měsíce*, pro *shlukovou analýzu* bude využito posledních *4000 hodnot* a pro výsledná doporučení bude probíhat 100 requestů. Z těchto hodnot jsem experimentálně zjistil odvozené hodnoty jako *počet ms výpočetního výkonu*, počet *přenesených dat*, počet *uložených dat* a tak dále. Nastavené hodnoty **lze snadno měnit** a velmi rychle získat novou predikci nákladů.

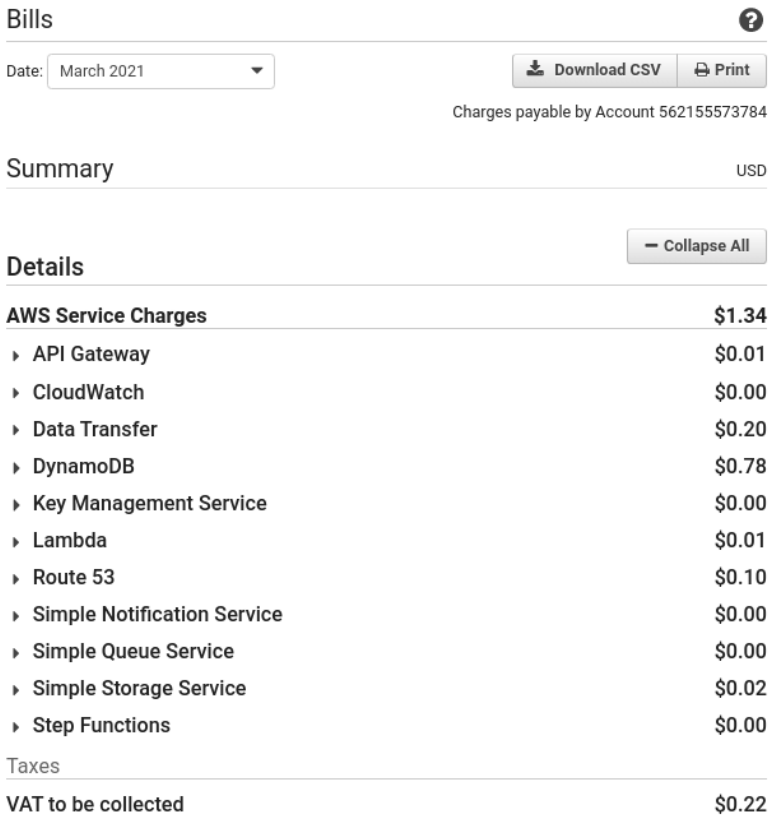
Graf 6.6 zobrazuje predikci nákladů v závislosti na počtu uživatelů (chytrých domácností). V grafu lze taktéž vidět cenový rozpad na jednotlivé služby.



Obrázek 6.6: Predikce nákladů v závislosti na počtu uživatelů

Pokud bychom tedy uvažovali, že budeme podporovat pro začátek 20 chytrých domácností, tak by se měsíční náklady měly pohybovat kolem jednoho dolaru.

V AWS existuje služba Bills and Cost Management [52], která dokáže monitorovat **ceny jednotlivých služeb**. V měsíci březnu byla v rámci AWS simulována aktivita podobná produkčnímu chování pro 20 chytrých domů. Tato měsíční aktivita, jak ilisturuje obrázek 6.7, byla naúčtována za **1.34$**, což potvrzuje predikovanou cenu.



Obrázek 6.7: Přehled provozu služby za měsíc březen (vytvořeno pomocí AWS Bills and Cost Management)

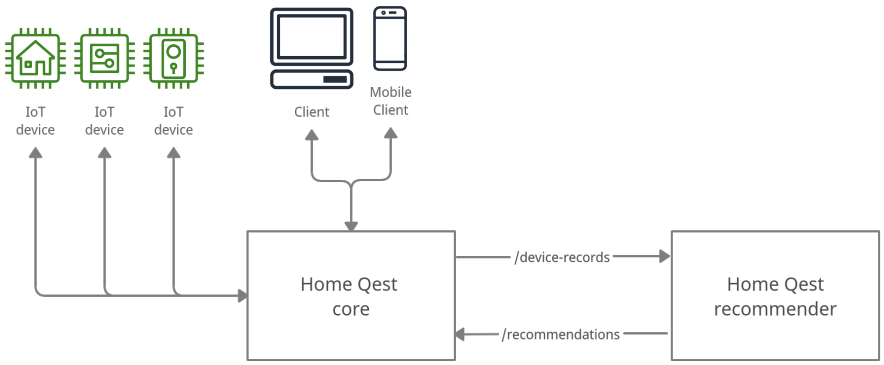
Kapitola 7

Možnosti integrace a budoucího rozšíření

Cílem této kapitoly je popsat možnosti dalšího rozvoje a návazností na projekt. První část této kapitoly popíše návrh možné **integrace** služby. Další části budou věnovány zamyšlením nad případným rozvojem této služby z pohledu **vylepšování algoritmu** a **správy dat**.

7.1 Integrace

Možností jak integrovat nově vzniknutou službu je určitě více, nicméně pokusil jsem se navrhnout to nejvíce pravděpodobné.



Obrázek 7.1: Návrh integrace do Home Qest core

Obrázek 7.1 ilustruje současné (zjednodušené) fungování *Home Qest core* a případnou integraci nově vzniknuté služby *Home Qest recommender*. Se systémem interagují **IoT zařízení**, které dokáží do systému posílat logy

o svých událostech (device records). Uživatelé v podobě mobilní či webové aplikace dokáží jednotlivá zařízení taktéž ovládat (například vytáhnout žaluzie).

Myšlenka integrace je taková, že by *Home Qest core* posílal logy o událostech IoT zařízení do služby *Home Qest recommender*. Varianta je, že by tyto logy posílal buď rovnou, jak chodí od zařízení a nebo by je poslal jednou za větší časový úsek, například v noci. Druhá varianta má výhodu, že není nutné vykonávat request do *Home Qest recommender* za běhu, během zpracování svého requestu.

Pro **doporučení** by si sahal *Home Qest core* prostřednictvím připraveného koncového bodu do služby *Home Qest recommender*. Perioda dotazování by ideálně chtěla synchronizovat s periodou přepočítávání doporučení na straně *Home Qest recommender*. Na straně služby *Home Qest recommender* by ještě mohl být filtr v duchu, dej mi doporučení vytvořené po datumu, aby se vyhnulo duplicitnímu čtení. Nicméně teď se stará doporučení mažou (už nejsou relevantní po novém výpočtu doporučení), a tak pokud by byla doba přepočtu na obou službách stejná nebylo by to ani potřeba. *Home Qest core* by pak doporučení uživateli zpropagoval do mobilní či webové aplikace, například v podobě **notifikace**.

Pokud by chtěl nově vzniknutou službu využívat **technik** (5.2.3), tak by mohl provolávat rovnou službu *Home Qest recommender* a jelikož je to technický člověk, tak by pro něho mohlo být dostatečné dostávat doporučení ve formátu JSON získaného rouvnou z API. Alternativně by mohla vzniknout jednoduchá webová aplikace, která by mu výsledky lépe formátovala.

7.2 Vylepšování algoritmu pro tvorbu doporučení

Oblast vylepšování algortimu pro tvorbu doporučení je téměř nevyčerpatelná. Nejjednodušší vylepšování je na úrovni **ladění parametrů** shlukové analýzy (např k pro k-means) či ladění vah dimenzí. Dále by šla **vylepšovat chybová funkce**, jak jsem již zmínil, filtruji nevyhovující shluky a objekty, alternativně by je šlo zkusit ještě zachránit například opětovným spuštěním procesu shlukové analýzy.

Dále by šlo přidat **novou metodu shlukové analýzy**, přidat **nový scénář** či **nové strategie** na filtrování nad doporučením. Všechny zmíněné body by šly dobře přidávat do existujícího systému, je na to dobře připravený.

Hezké by bylo udělat úplně **obecný doporučovací systém**. Systém by tak jen přijímal libovolné pole dat a výsledkem by byla sada doporučení. Automaticky by se tak provedla analýza vstupních parametrů, normalizace dat, přidávání vah, tvorba datové matice, shluková analýza a reprezentace dat. Pokud by se rozhodlo jít touto cestou, šla by řada dílčích částí z této práce použít.

Další rozšíření by mohlo být v podobě zapojení **zpětné vazby uživatele** na již vytvořená doporučení. Tato oblast nabízí více cest, jak to řešit. Dejme

tomu, že by systém navrhl doporučení s nějakým zařízením, např. *světlo na toaletě* v nějakém čase, např v *7:00*. Uživatel by toto doporučení na zautomatizování nepotvrdil. Systém by tak mohl na nějakou dobu (nebo na vždy) doporučení s tímto zařízením (*světlo na toaletě*) nedoporučovat. Nedoporučovat by se dalo buď jen v hodinách blízkých *7:00* či v žádném čase. Co kdyby doporučování ještě obsahovalo více zařízení? Zakázat doporučení, která obsahují přesně tyto zařízení nebo stačí část z nich? Lze vidět, že ač bylo toto téma nastíněno jen velmi povrchově hned otevřelo řadu otázek.

Home Qest také nově umí pracovat s **počasím**, tedy dokáže nastavovat podmínkové akce na základě počasí. Další rozšíření by tak mohlo být podobně jako východ/západ slunce přidat jiné dimenze související s počasím, například déšť, oblačnost a tak dále.

7.3 Správa dat a konfigurace

Co současné řešení přímo nepodporuje je **správa dat domů** (DB tabulka Homes), kde jsou uložené informace o domech, jako je poloha, která slouží pro generování dimenzí související s východem a západem slunce. Pokud by tak do současného řešení chtěl někdo zapojit nový dům, musel by ho přidat do *DynamoDB databáze*. Lepší varianta by tak byla podporat správu přes API a to včetně editace a mazání.

Dále by mohla existovat podpora pro **správu konfigurací souvisejících s doporučením**. Jako je nastavení dimenzí, jejich vah, parametrů shlukové analýzy (k pro k-means), různé prahy chybové funkce, počet záznamů, na kterých se provádí datová analýza a tak dále. V současné době jsou tyto nastavení definovaná v jednom konfiguračním souboru ve zdrojovém kódu. Lepší by ale bylo kdyby byla alespoň v databázi nebo ještě lépe editovatelná přes API.

Další rozšíření by mohlo přijít v podobě **webové aplikace na správu dat a konfigurací**. Přes tuto aplikaci by tak například mohli jít spravovat výše zmíněné informace o domech a konfiguracích. Dále by zde mohli být v nějaké formě zobrazené výsledné doporučení a různé statistiky nad daty. Samozřejmě může přijít argument, že je to zbytečná práce, protože k datům a statistikám je už přístup přes databázi či přes statistiky Amazon Web Services, ale tyto technologie už jsou trochu specifické a například těžko uchopitelné pro netechnické lidi, ale i netechnický člověk, by mohl na tomto projektu najít uplatnění. Kdyby měl například možnost dobře pozorovat přehled všech doporučení, mohl by v datech nacházet nové podněty na *scénáře* či *strategie dotazování*.

Závěr

Cílem této diplomové práce bylo vytvoření nezávislé služby pro systémy domácí automatizace, která dokáže na základě IoT senzorických dat doporučovat nastavení chytré domácnosti. U navržené architektrury měl být kladen důraz na stabilní škálovatelnou architekturu pomocí AWS serverless technologií.

Podařilo se navrhnout a implementovat **funkční prototyp v plném rozsahu zadání**. Samotná tvorba doporučení se opřela o algoritmy shlukové analýzy, konkrétně byly použity metody *k-means* a *DBSCAN*. Dále bylo navrženo několik scénářů a strategií pro variabilitu dotazování nad vytvořenými doporučeními.

Celý proces tvorby doporučení je postaven na službách *AWS serverless*, které kompletně pomáhají orchestrovat periodické předzpracování vstupních dat, shlukovou analýzu, čištění a ukládání výsledků do databáze. Proces je rozdělen do malých samostatných kroků, které jsou separátně nasazené pomocí *Amazon Lambda funkcí*. Jednotlivé kroky tak lze odděleně škálovat či jim nastavovat ideální výpočetní výkon. Tyto kroky jsou na víc zcela izolované, což přináší možnost například konkrétní krok, jako shluková analýza, nahradit za *Lambda funkci* napsanou v jiném programovacím jazyce. Benefitem současného řešení je také to, že v sobě zahrnuje velké množství paralelních výpočtů.

Služba dále poskytuje aplikační rozhraní, které podporuje příjem nových IoT dat a poskytování samotných doporučení. Pro tuto část jsou taktéž využity *AWS serverless* technologie, které poskytují stabilní škálovatelnou architekturu. Stabilita a škálovatelnost byla otestována pomocí zátěžového testu, ve kterém se ukázalo, že služba dokáže bez problému škálovat do několika desítek paralelních zpracování najednou.

Celý projekt se podařilo nasadit do *Amazon Web Services*. K procesu nasazení byl využit relativně nový trend v podobě *infrastruktury jako kódu*, konkrétně bylo využito frameworku *AWS CDK*. Benefitem je to, že je cloudová architektura definovaná pomocí stejného jazyka, jako je napsána samotná

služba. Tato infrastruktura je tak snadno rozšiřitelná a znovupoužitelná.

Jako testovací data sloužila anonymizovaná data od systému *Home Qest*, který provozuje systém pro chytré domácnosti s asi dvaceti aktivními chytrými domy. Průběh a výsledky této práce byly prezentovány a konzultovány s tvůrci *Home Qest* a s technikem tohoto systému. Celý koncept byl přijat velmi kladně a zároveň konzultace přispěly k řadě podnětům na vylepšení, z nichž byla většina zapracována a některá alespoň nastíněna v rámci budoucího rozšíření. Mezi takovéto rozšíření patří například *uživatelská zpětná vazba*, která by ovlivňovala budoucí tvorbu doporučení.

Dále byl vypočítán odhad na kompletní provoz této služby v *AWS*. Z tohoto odhadu vyšlo to, že celkové provozování služby pro daný počet domů (nízké desítky domů) by činil pouhé nízké jednotky dolarů. V této cenové analýze byla také demonstrována predikce toho jak by se zvyšovaly náklady v závislosti na počtu chytrých domácností.

Službu se dále podařilo nasadit na lidsky čitelnou doménu a opatřit jí autorizačním zabezpečením. Služba je tak připravena k integraci, která je v této práci také navržena. Pevně věřím, že k ní dojde a výsledek této práce tak bude použit.

Literatura

[1] D. Gann, J. B.; Venables, T.: *Digital Futures: Making Homes Smarter*. Chartered Institute of Housing, 1999, ISBN 978-1900396141.

[2] Number of Smart Homes forecast worldwide until 2025. *Statista [online]*, 2021, [cit. 2021-03-26]. Dostupné z: `https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-worldwide`

[3] Piatetsky-Shapiro, G.: *Knowledge Discovery in Real Databases*. AI Magazine, 11(4), 1990.

[4] Usama M. Fayyad, P. S., Gregory Piatetsky-Shapiro; Uthurusamy, R.: *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996, ISBN 9780262560979.

[5] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.: The KDD process for extracting useful knowledge from volumes of data. *Commun. ACM*, ročník 39, 1996.

[6] Jiawei Han, J. P., Micheline Kamber: *Data Mining: Concepts and Technique*. Morgan Kaufmann Publishers Inc., 2011, ISBN 978-0-12-381479-1.

[7] Rui Xu, D. C. W.: *Clustering*. Wiley-IEEE Press, 2009, ISBN 978-0-470-27680-8.

[8] Gangwar, P.: What is cluster analysis. *Slideshare [online]*, 2016, [cit. 2021-03-27]. Dostupné z: `https://www.slideshare.net/Prabhatgangwar/what-is-cluster-analysis`

[9] Dangeti, P.: Statistics for Machine Learning by Pratap Dangeti. *O'Reilly Media [online]*, 2017, [cit. 2021-04-10]. Dostupné z: `https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/c71ea970-0f3c-4973-8d3a-b09a7a6553c1.xhtml`

[10] Yobero, C.: K-Means Clustering Tutorial. *RStudio [online]*, 2018, [cit. 2021-03-27]. Dostupné z: https://rpubs.com/cyobero/k-means

[11] Alboukadel: Types of Clustering Methods: Overview and Quick Start R Code. *Datanovia [online]*, 2020, [cit. 2021-03-27]. Dostupné z: https://www.datanovia.com/en/blog/types-of-clustering-methods-overview-and-quick-start-r-code

[12] Mell, P.; Grance, T.: The NIST Definition of Cloud Computing. 2011-09-28 2011, doi:https://doi.org/10.6028/NIST.SP.800-145.

[13] Ahead in the Clouds. *Newsinsights [online]*, 2019, [cit. 2021-03-29]. Dostupné z: https://www.hanetf.com/article/190/ahead-in-the-clouds-skyy

[14] Problémy legislativy cloud computingu. *Systemonline [online]*, 2017, [cit. 2021-04-01]. Dostupné z: https://www.systemonline.cz/virtualizace/problemy-legislativy-cloud-computingu-z.htm

[15] Reno, N.: Cloud Market Ends 2020 on a High while Microsoft Continues to Gain Ground on Amazon. *Synergy Research Group [online]*, 2021, [cit. 2021-04-02]. Dostupné z: https://www.srgresearch.com/articles/cloud-market-ends-2020-high-while-microsoft-continues-gain-ground-amazon

[16] Richter, F.: Amazon Leads 130 dollars-Billion Cloud Market. *Statista [online]*, 2021, [cit. 2021-04-01]. Dostupné z: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers

[17] Global Infrastructure. *Amazon Web Services [online]*, 2021, [cit. 2021-04-02]. Dostupné z: https://aws.amazon.com/about-aws/global-infrastructure

[18] Cloud computing with AWS. *Amazon Web Services [online]*, 2021, [cit. 2021-04-02]. Dostupné z: https://aws.amazon.com/what-is-aws

[19] Serverless on AWS. *Amazon Web Services [online]*, 2021, [cit. 2021-04-02]. Dostupné z: https://aws.amazon.com/serverless

[20] Edelhoff, R.: Serverless services on AWS. *Itemis AG [online]*, 2021, [cit. 2021-04-03]. Dostupné z: https://blogs.itemis.com/en/serverless-services-on-aws

[21] About JavaScript. *MDN [online]*, 2021, [cit. 2021-04-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[22] What is TypeScript? *Microsoft [online]*, 2021, [cit. 2021-04-04]. Dostupné z: https://www.typescriptlang.org

[23] Nguyen, D.: *Node.js Okamžitě*. Brno: Computer Press, 2016, ISBN 978-80-251-4820-4.

[24] DeBill, E.: Module Counts. 2021, [cit. 2021-04-04]. Dostupné z: http://www.modulecounts.com/

[25] Density Based Clustering for JavaScript. *npm [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://www.npmjs.com/package/density-clustering

[26] K-Means Algorithm. *Amazon Web Services [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://docs.aws.amazon.com/sagemaker/latest/dg/k-means.html

[27] 3D Scatter Plots. *Plotly [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://plotly.com/python/3d-scatter-plots/

[28] Plotly Node API. *npm [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://www.npmjs.com/package/plotly

[29] Open. *npm [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://www.npmjs.com/package/opn

[30] Sunrise-Sunset-Js. *npm [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://www.npmjs.com/package/sunrise-sunset-js

[31] Moment.js. *npm [online]*, 2021, [cit. 2021-04-05]. Dostupné z: https://www.npmjs.com/package/moment

[32] What is AWS Step Functions? *Amazon Web Services [online]*, 2021, [cit. 2021-04-07]. Dostupné z: https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html

[33] AWS Lambda function scaling. *Amazon Web Services [online]*, 2021, [cit. 2021-04-07]. Dostupné z: https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html

[34] AWS Step Functions increases payload size to 256KB. *Amazon Web Services [online]*, 2021, [cit. 2021-04-07]. Dostupné z: https://aws.amazon.com/about-aws/whats-new/2020/09/aws-step-functions-increases-payload-size-to-256kb

[35] Schedule Expressions for Rules. *Amazon Web Services [online]*, 2021, [cit. 2021-04-08]. Dostupné z: https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.html

[36] Amazon API Gateway. *Amazon Web Services [online]*, 2021, [cit. 2021-04-08]. Dostupné z: https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.html

[37] Cold Starts in AWS Lambda. *Mikhail Shilkov [online]*, 2021, [cit. 2021-04-08]. Dostupné z: https://mikhail.io/serverless/coldstarts/aws/

[38] Neves, G.: Keeping Functions Warm - How To Fix AWS Lambda Cold Start Issues. *Serverless [online]*, 2021, [cit. 2021-04-08]. Dostupné z: https://www.serverless.com/blog/keep-your-lambdas-warm

[39] What Is Amazon DynamoDB? *Amazon Web Services [online]*, 2021, [cit. 2021-04-09]. Dostupné z: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[40] Core Components of Amazon DynamoDB. *Amazon Web Services [online]*, 2021, [cit. 2021-04-09]. Dostupné z: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

[41] What is the AWS CDK? *Amazon Web Services [online]*, 2021, [cit. 2021-04-10]. Dostupné z: https://docs.aws.amazon.com/cdk/latest/guide/home.html

[42] van der Maaten, L.: t-SNE. [cit. 2021-04-11]. Dostupné z: https://lvdmaaten.github.io/tsne

[43] Frost, J.: Coefficient of Variation in Statistics. 2021, [cit. 2021-04-11]. Dostupné z: https://statisticsbyjim.com/basics/coefficient-variation

[44] Modus. *Nová média, s. r. o. [online]*, [cit. 2021-04-11]. Dostupné z: https://matematika.cz/modus

[45] Fast, unopinionated, minimalist web framework for node. *npm [online]*, 2021, [cit. 2021-04-15]. Dostupné z: https://www.npmjs.com/package/express

[46] The most powerful schema description language and data validator for JavaScript. *npm [online]*, 2021, [cit. 2021-04-15]. Dostupné z: https://www.npmjs.com/package/joi

[47] Amazon Route 53 Documentation. *Amazon Web Services [online]*, 2021, [cit. 2021-04-17]. Dostupné z: https://docs.aws.amazon.com/route53

[48] Delightful JavaScript Testing. *npm [online]*, 2021, [cit. 2021-04-17]. Dostupné z: https://www.npmjs.com/package/jest

[49] Bekkhus, S.: Snapshot Testing. *Facebook, Inc. [online]*, 2021, [cit. 2021-04-17]. Dostupné z: `https://jestjs.io/docs/snapshot-testing`

[50] Error Handling with DynamoDB. *Amazon Web Services [online]*, 2021, [cit. 2021-04-17]. Dostupné z: `https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.Errors.html`

[51] AWS Pricing Calculator. *Amazon Web Services [online]*, 2021, [cit. 2021-04-17]. Dostupné z: `https://calculator.aws/`

[52] What is AWS Billing and Cost Management? *Amazon Web Services [online]*, 2021, [cit. 2021-04-17]. Dostupné z: `https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-what-is.html`

Příloha **A**

Seznam použitých zkratek

AWS Amazon Web Services

IoT Internet of Things

ID Identity Document

API Aplication Interface

DB Database

KDD Knowledge Discovery in Databases

DBSCAN Density-based spatial clustering of applications with noise

IaaS Infrastructure as a Service

PaaS Platform as a Service

CPU Central Processing Unit

GPU Graphics Processing Unit

RAM Random Access Memory

HTTP Hypertext Transfer Protocol

CRM Customer relationship management

REST Representational State Transfer

SOAP Simple Object Access Protocol

SDK Software Development Kit

FaaS Function as a service

IDE Integrated Development Enviroment

NPM Node Package Manager

URL Uniform Resource Locator

MB Megabyte

CDK Cloud Development Kit

CI/CD Continuous Integration/Continuous Delivery

DNS Domain Name System

JSON JavaScript Object Notation

Příloha B

Ukázka výsledných doporučení

V této příloze budou demonstrovány konkrétní reálné výsledky, které byly získány přes API. Byl proveden request na URL `https://home-recommender.qestapp.net/recommendations?homeId=5c98b3115b4a330001521a27`, z které byl vrácen výsledek v podobě pole doporučení[5].

První doporučení (řádek 2 - 28) nabízí vytahování žaluzií ve všední dny, v čase 7:30. Druhé (řádek 29 - 61) doporučuje rozsvítit dvojici světel v 20:00 všedního dne. Poslední doporučení (řádek 62 - 102) nabízí rozsvícení trojice světel při západu slunce každý všední den.


```
1  [
2      {
3          "useCase": "weekdayMorningAll",
4          "dayType": "weekday",
5          "devices": [
6              {
7                  "DeviceType": "windowBlind",
8                  "DeviceId": "5c503be04f89eb0001094faa",
9                  "State": {
10                     "_t": "WindowBlindState",
11                     "Position": {
12                         "Shift": 100,
13                         "Tilt": 0
14                     }
15                 },
16                 "DeviceName": "Zaluzie 6"
17             }
18         ],
19         "deviceCount": 1,
20         "homeId": "5c98b3115b4a330001521a27",
21         "computeMethod": "densityBased",
22         "repeatPerDevice": 70,
23         "_created": "2021-03-19T15:16:06.263Z",
24         "_id": "0fc39fcf-bcf6-46ec-86b0-210b49452b24",
```


[5] Není to kompletní odpověd, ale pouze výběr z výsledků.

```
        "time": "07:30:00.000+00:00",
        "clusterSize": 70,
        "type": "time"
    },
    {
        "useCase": "weekdayEveningAll",
        "dayType": "weekday",
        "devices": [
            {
                "DeviceType": "dimmableLight",
                "DeviceId": "5c4f5d82494bdc0001173eb1",
                "State": {
                    "_t": "DimmableLightState",
                    "Intensity": 100
                },
                "DeviceName": "Nad gaucom"
            },
            {
                "DeviceType": "binaryLight",
                "DeviceId": "5c4f62c4494bdc0001173ebe",
                "State": {
                    "_t": "BinaryLightState",
                    "TurnedOn": true
                },
                "DeviceName": "Hlavne svetlo"
            }
        ],
        "deviceCount": 2,
        "homeId": "5c98b3115b4a330001521a27",
        "computeMethod": "densityBased",
        "repeatPerDevice": 40,
        "_created": "2021-03-19T15:16:05.779Z",
        "_id": "2098eddb-1216-46ee-8086-356afb598a40",
        "time": "20:00:00.000+00:00",
        "clusterSize": 80,
        "type": "time"
    },
    {
        "useCase": "weekdaySunsetAll",
        "dayType": "weekday",
        "devices": [
            {
                "DeviceType": "binaryLight",
                "DeviceId": "5c4f62c4494bdc0001173ebe",
                "State": {
                    "_t": "BinaryLightState",
                    "TurnedOn": true
                },
                "DeviceName": "Hlavne svetlo"
            },
            {
                "DeviceType": "binaryLight",
                "DeviceId": "5c4f6097494bdc0001173eb9",
                "State": {
```

```
                    "_t": "BinaryLightState",
                    "TurnedOn": true
                },
                "DeviceName": "Za krbom"
            },
            {
                "DeviceType": "binaryLight",
                "DeviceId": "5c4f628e494bdc0001173ebd",
                "State": {
                    "_t": "BinaryLightState",
                    "TurnedOn": true
                },
                "DeviceName": "Svetlo"
            }
        ],
        "deviceCount": 3,
        "homeId": "5c98b3115b4a330001521a27",
        "computeMethod": "densityBased",
        "repeatPerDevice": 16,
        "_created": "2021-03-19T15:19:55.303Z",
        "_id": "6d11ac73-57b4-4bd4-974a-e86154109271",
        "clusterSize": 48,
        "type": "sunset"
    }
]
```

Zdrojový kód B.1: Ukázka (vybraných) doporučení získaných přes API

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS