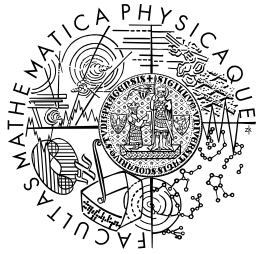


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Ondřej Kaštovský

Predikce délky trvání datového profilování

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: doc. RNDr. Jan Kofroň, Ph.D.

Studijní program: Informatika (B0613A140006)

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chtěl bych mnohokrát poděkovat doc. RNDr. Janu Kofroňovi, Ph.D. za cenné rady, shovívavost a ochotu s blížícím se odevzdáním práce. Dále děkuji Mgr. Lukáši Kolkovi za předané znalosti a profesní zkušenosti a možnost na tématu této práce pracovat. V neposlední řadě bych chtěl poděkovat rodině, přítelkyni a kamarádům za podporu během studia a při psaní závěrečných stran této práce. Děkuji Vám všem.

Název práce: Predikce délky trvání datového profilování

Autor: Ondřej Kaštovský

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: doc. RNDr. Jan Kofroň, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Kvalitní data dnes hrají zásadní roli ve strategickém plánování a rozhodovacích procesech firem. Schopnost předpovídat délku trvání úloh spojených s jejich zpracováním a analýzou je klíčová pro efektivní využití zdrojů a optimalizaci pracovních procesů. Cílem práce je rozšířit funkcionalitu platformy pro správu dat *Ataccama ONE* společnosti *Ataccama* o novou mikroservisu umožňující předpovídat délku trvání úloh datového profilování. V řešení se zabýváme identifikací klíčových vlastností dat, které délku trvání úloh ovlivňují, a využitím těchto poznatků ke tvorbě prototypu modelu strojového učení, jenž umožní délky trvání úloh predikovat. Součástí řešení je také detekce a zpracování úloh v reálném čase a příprava na budoucí integraci řešení do platformy. Důraz je kladen na kvalitu implementace a rozšiřitelnost o možnost predikce dalších typů úloh.

Klíčová slova: datový management datové profilování predikce

Title: Prediction of data-profiling duration

Author: Ondřej Kaštovský

Department: Department of Distributed and Dependable Systems

Supervisor: doc. RNDr. Jan Kofroň, Ph.D., Department of Distributed and Dependable Systems

Abstract: Today, data quality plays a vital role in strategic planning and corporate decision-making processes. The ability to predict the duration of tasks related to data processing and analysis is crucial for efficient use of resources and optimization of work processes. The goal of this work is to extend the functionality of *Ataccama ONE*, a data management platform of *Ataccama*, with a new microservice that allows predicting the duration of data profiling jobs. Our solution involves identifying the key data characteristics that affect the duration of these jobs and using these insights to prototype a machine learning model to predict job durations. An important part of the solution is also to detect and process newly executed jobs in the platform in real-time and prepare the microservices for future integration into the platform. Emphasis is then placed on the quality of the implementation and the extensibility of the solution to predict other types of jobs.

Keywords: data management data profiling prediction

Obsah

Úvod

S rychle rostoucím objemem dat se v posledních letech stává jejich kvalitní správa a analýza klíčovým faktorem pro rozhodovací procesy firem ve většině odvětví. V tomto datově nabitém prostředí je nezbytné data nejen efektivně spravovat a analyzovat, ale také optimalizovat procesy, které se jich dotýkají[1]. Jedním z aspektů této optimalizace je schopnost předpovědět délku trvání úloh spojených s jejich analýzou, například datovým profilováním, jehož výsledek je zásadní pro pochopení kvality, struktury a obsahu datových sad, a proto hraje důležitou roli ve zlepšování efektivity datových procesů. Schopnost odhadovat délku trvání úloh spojených s profilováním dat umožňuje lepší plánování a využití zdrojů, což je v dnešním světě velkých dat naprosto neocenitelné.

Cíl práce

Cílem této práce je vytvoření prototypu modelu strojového učení pro predikce délek trvání úloh datového profilování na základě identifikace vlastností dat databázových tabulek, které délku trvání těchto úloh ovlivňují, a následná integrace vytvořeného modelu do platformy Ataccama ONE (ONE)[2] s důrazem na kvalitu implementace a její možné rozšíření pro predikci délek trvání dalších podporovaných úloh platformy.

Struktura práce

První kapitola je věnována procesu datového profilování a jeho využití v platformě Ataccama ONE. Ve druhé kapitole je podrobně rozebrán analyzovaný problém, který se zabývá predikcí délek trvání úloh datového profilování v této platformě, a jsou popsány možnosti a cíle jeho řešení. Také je zde popsán modelový příklad využití predikcí v kontextu chytrého plánování úloh v platformě. Třetí kapitola se zaměřuje na přípravu dat, měření výsledků a jejich následnou analýzu potřebnou k vytvoření predikčního modelu. Čtvrtá kapitola pojednává o technických řešeních, zahrnuje požadavky na implementaci, popis a zdůvodnění

volby použitých technologií. Pátá kapitola se zabývá komunikací s platformou Ataccama ONE a integrací výsledné aplikace do této platformy. Šestá kapitola je věnována implementaci aplikace, její architektuře, popisu jednotlivých komponent a testování. Sedmá kapitola obsahuje měření přesnosti modelů a analýzu výsledků. Závěrečná kapitola shrnuje celou práci, hodnotí naplnění stanovených cílů a nastiňuje možnosti dalšího vývoje.

Kapitola 1

Datové profilování, Ataccama ONE

V úvodní kapitole si definujeme proces *datového profilování* a popíšeme některé další základní pojmy z oblasti *datového managementu*[3]. Dále si představíme platformu *Ataccama ONE* a ukážeme si, jaké možnosti nám v oblasti *datového profilování* poskytuje.

1.1 Datové profilování

Datové profilování je proces zkoumání, analýzy a porozumění dat dostupných v datovém zdroji. Cílem tohoto procesu je získat statistiky a detailní informace o kvalitě, struktuře a anomáliích dat a předat je ve srozumitelné podobě koncovým uživatelům, jimiž jsou zpravidla analytické týmy firem. Získané statistiky mohou firmám pomoci identifikovat potenciální příležitosti pro zlepšení kvality a správy jejich dat a díky tomu omezit chybná obchodní rozhodnutí nad daty nekvalitními[4]. Během procesu datového profilování se na úrovni sloupců, řádků, tabulek či celých datových zdrojů analyzují různé vlastnosti dat, například:

- Numerické statistiky – minima, maxima, průměry, součty, mediány, standardní odchylky a rozptyly pro numerické sloupce.
- Řetězcové statistiky – analýza masky, analýza vzorů, minimální, průměrná a maximální délka hodnot.
- Kvalita dat – počet unikátních, odlišných[1], duplicitních a prázdných[2] hodnot.
- Frekvence výskytu na základě frekvenční analýzy.

[1] Distinct
[2] Null

- Klasifikace dat – na základě detekčních pravidel [5] se sloupcům přiřazují značky charakterizující jejich obsah (například email, adresa, křestní jméno atd.), které usnadňují následnou orientaci v metadatech.

1.2 Datový katalog

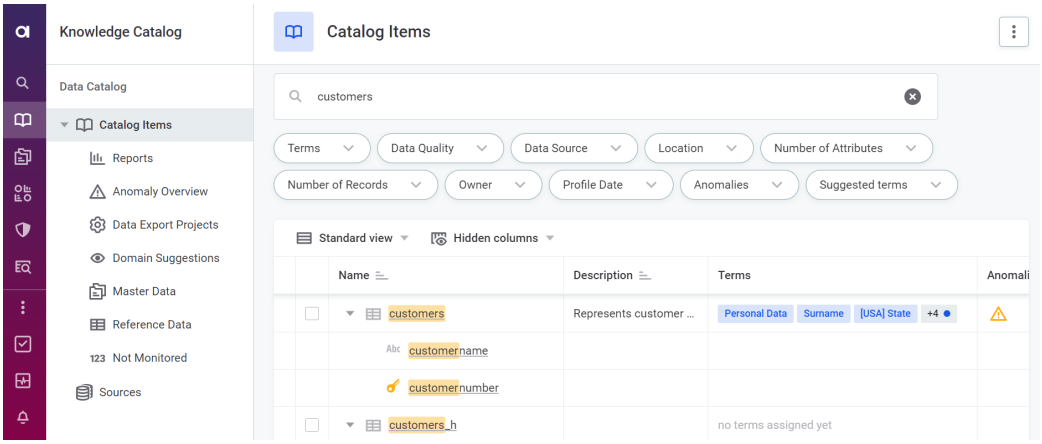
Datový katalog je systém sloužící k organizaci a správě metadat o připojených datových zdrojích. Takovými metadaty mohou být například informace o tom, jak jsou data uložena, kdo je jejich vlasníkem nebo jaký je jejich obsah. Do *datového katalogu* jsou kromě metadat často ukládány také výsledky *datového profilování.*

1.3 Ataccama One

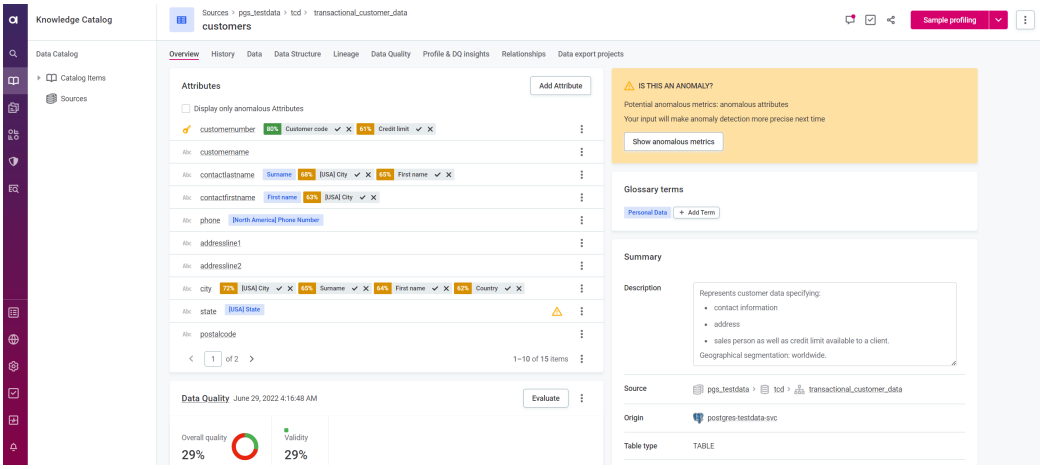
Ataccama ONE je platforma pro správu dat vyvíjená společností *Ataccama[6]* zabývající se různými disciplínami *datového managementu* jako je *Data Governance*[7], *Master Data Management*[8] a *Data Quality*[9]. Součástí nástrojů *Data Quality* je také *datové profilování.*

1.3.1 Knowledge Catalog

Knowledge Catalog[10] je sekce *ONE*, která obsahuje *datový katalog* (*Obrázek 1.1*), a umožňuje uživatelům mimo jiné pouštět *úlohy datového profilování* a prohlížet jejich výsledky. Jednotlivé položky v *datovém katalogu* se nazývají *Catalog Item*[11]. *Catalog Item* (*Obrázek 1.2*) je abstrakce nad tabulkami (databázovými nebo souborovými) obsahující skutečná data rozdělená do atributů, na které můžeme pohlížet jako na tabulkové sloupce. Zároveň však také může abstrahovat i jiné entity jako reporty z nástrojů pro vizualizaci dat, případně datové formáty netabulkových datových zdrojů. V této práci se budeme zabývat pouze tabulkovými zdroji, a to z toho důvodu, že se v kontextu *datového profilování Ataccama* primárně zaměřuje právě na ně. Dále tedy budeme pro lepší srozumitelnost textu tyto položky označovat jako *tabulky.*



Obrázek 1.1 Datový katalog[12] v ONE



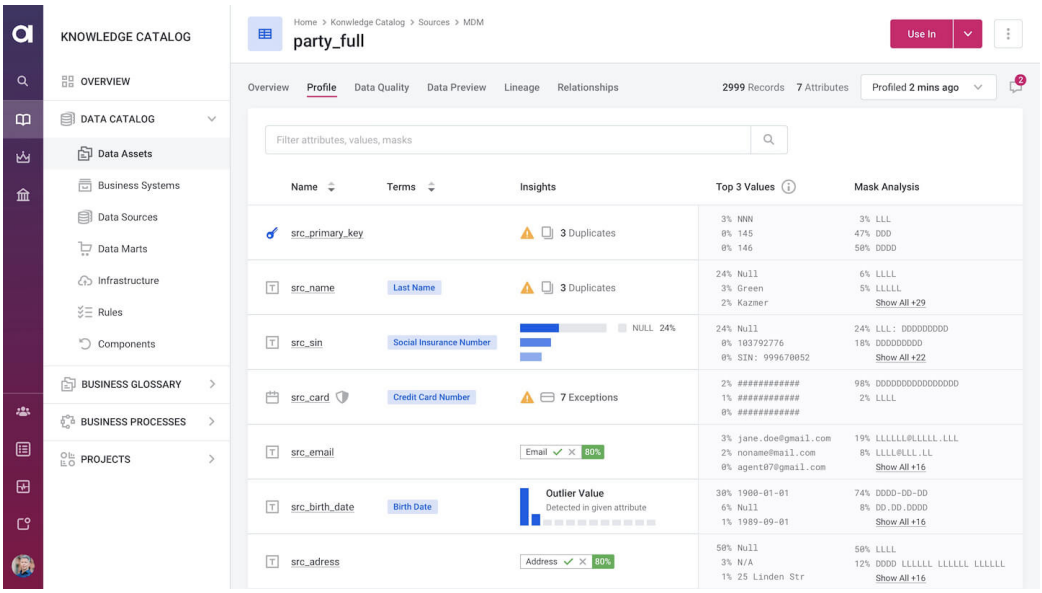
Obrázek 1.2 *Catalog Item*[11] `customers` v ONE. Při prohlížení můžeme mimo jiné procházet jeho data, sledovat kvalitu a klasifikaci dat a prohlížet výsledky úloh datového profilování.

1.3.2 Datové profilování v Ataccama ONE

ONE nám umožňuje spustit *úlohu datového profilování* jak pro celou *tabulku*[3], tak také pouze pro její vzorek [4]. Velikost a způsob získání vzorku závisí na implementaci konkrétní databázové technologie, například databázový systém *PostgreSQL*[13] využívá funkce `TABLESAMPLE`[14] a v případě ONE vzorkuje na

[3] Full Profiling
[4] Sample Profiling

deset tisíc záznamů. Profilování vzorku je proto vhodné pro získání reprezentativního přehledu hlavních charakteristik analyzovaných dat v řádu několika sekund. Profilování celé tabulky pak zahrnuje podrobnou komplexní analýzu veškerých aspektů dat (*Obrázek 1.3*) a je časově i zdrojově výrazně náročnější. V praxi se pak velmi často používá kombinace obou způsobů. Algoritmus *úloh datového profilování* v ONE se skládá ze dvou fází. První fáze přečte data z příslušného datového zdroje, nazveme ji proto *fáze čtení*. Druhá fáze nad přečtenými daty vypočte statistiky zmíněné v sekci 1.1, nazveme ji proto *fáze výpočtu*.



Obrázek 1.3 Výsledky úlohy datového profilování v platformě Ataccama ONE

Kapitola 2

Analýza problému

Než přejdeme k samotné definici problému, představme si důležité pojmy, které budeme při jeho popisu používat.

Metadata Management Module (MMM) je centrální backend služba *ONE*[15]. Spravuje metadata všech ostatních modulů *ONE* včetně datového katalogu.

Data Processing Engine (DPE) je horizontálně škálovatelná výpočetní jednotka *ONE*, sloužící pro úlohy zpracování dat jako je například *datové profilování*. Jako jediná se připojuje na data zákazníků a uživatelů *ONE*.

Data Processing Module (DPM) je modul spravující v *ONE* všechny spuštěné úlohy a jejich metadata. Tato metadata kromě základních infomací jako je jméno, typ a status úlohy obsahují také standardní[1] a chybový[2] výstup poskytující důležité informace o běhu jednotlivých úloh. Jedna instance *DPM* může spravovat více instancí *DPE*.

ONE API je rozhraní umožňující se pomocí dotazovacího jazyka *GraphQL*[16] dotazovat na data v ONE a manipulovat s nimi. Pomocí *ONE API* je tedy možné programově získávat výsledky jednotlivých *úloh datového profilování*, čemuž se podrobněji věnujeme v kapitole 5.

2.1 Popis problému

Uživatel pomocí uživatelského rozhraní v *MMM* spustí *úlohu datového profilování*. Výpočet se začne provádět na dostupné instanci *DPE*. *DPM* ani uživatel nevědí, jak dlouho úloha poběží. Uživatel není schopen předvídat vliv jeho akcí na

[1] STDOUT
[2] STDERR

vytížení jednotlivých *DPE* ani čas, kdy bude úloha dokončena. *DPM* není schopen chytrého plánování úloh pro ideální vytížení a vysokou propustnost výkonných jednotek *DPE*, protože před spuštěním *úlohy datového profilování* nemá o předpokládané délce jejího trvání, která se může pohybovat v řádech nižších jednotek sekund až vyšších jednotek hodin, žádné informace. Délky trvání *úloh datového profilování* jsou závislé na vlastnostech dat profilovaných tabulek analyzovaných v sekci 3.2. Tyto vlastnosti jsou ovšem v platformě, konkrétně v *MMM*, dostupné až po dokončení celé úlohy.

2.2 Možnosti a cíl řešení

Cílem řešení je využít délky trvání již dokončených *úloh datového profilování* a podobností mezi tabulkami, které byly těmito úlohami profilovány, k predikci délky trvání budoucích úloh. Podobnostmi tabulek rozumíme podobné vlastnosti dat tabulek získaných z výsledků jejich profilování (například podobný počet záznamů, sloupců, prázdných záznamů atd.). Vlastnosti dat tabulek, které ovlivňují délku trvání *úloh datového profilování*, musíme nejprve identifikovat a následně je vhodně využít k sestrojení prototypu modelu strojového učení, který bude na jejich základě schopen délky trvání úloh predikovat. V rámci této práce se primárně zaměříme na vytvoření jednoho predikčního modelu na každý typ úlohy, případně na její jasně definovanou podúlohu. Předpokládáme, že pro vytvoření funkčního prototypu bude jeden model pro každý typ (pod)úlohy dostatečný. Alternativně bychom se mohli pokoušet o vytvoření více modelů, například pro každou tabulku nebo skupinu příbuzných tabulek zvlášť (například dedikovaný model pro tabulky se stejným počtem sloupců). Tyto alternativní přístupy zvážíme v rámci budoucích rozšíření řešení, pokud na základě analýzy a výkonnosti prototypu vyhodnotíme, že by mohly přesnost predikcí výrazně zlepšit.

Každá tabulka má v ONE svůj unikátní identifikátor, který zůstává neměnný, dokud tabulka existuje. Libovolné změny jejího obsahu jako jsou například přidávání nebo mazání záznamů tedy tento identifikátor zachovávají. To, jestli libovolná tabulka byla již profilována, můžeme vyčíst buď z uživatelského rozhraní ONE nebo získat programově pomocí *ONE API*. Při spuštění *úlohy datového profilování* na libovolné tabulce mohou nastat následující situace:

Tabulka nebyla nikdy profilována – *ONE* momentálně neumožňuje o datech tabulky získat před jejím prvním naprofilováním žádné informace, což zahrnuje i celkový počet záznamů. Důvodem je to, že informace o celkovém počtu záznamů je součástí výsledků jednotlivých úloh datového profilování a je možné ji získat až po dokončení příslušné úlohy a zveřejnění jejích výsledků v datovém katalogu. Předpokládáme, že počet záznamů tabulky bude mít na délku profilování nejzásadnější vliv. Před prvním spuštěním

úlohy na každé tabulce tedy není možné určit, zda-li má tabulka jeden tisíc nebo sto milionů záznamů a nemá tak smysl se v tomto případě o predikci délky trvání pokoušet. Pokud by v budoucnu došlo v platformě *ONE* k přidání možnosti zjistit údaj o počtu záznamů (a případně i sloupců a jejich datových typů) před nebo při spuštění úlohy, můžeme řešení o tuto možnost rozšířit.

Celá tabulka byla alespoň jednou profilována – V tomto případě máme dvě možnosti, jak k predikci přistoupit. Můžeme buď využít natrénovaného modelu na všech zpracovaných úlohách a jako vstupní údaje pro predikci mu předat vlastnosti dat získané z předchozích profilování tabulky, nebo délku trvání úlohy odhadnout čistě na základě předchozích výsledků úloh na stejné tabulce. Ve druhém případě pak můžeme například pozorovat trend změny počtu záznamů mezi výsledky jednotlivých úloh, nebo použít průměr délek jejich trvání. Toto rozhodnutí závisí na pozorováních v sekci 3.2.

Tabulka byla profilována pouze na vzorku – Tento případ je pro nás zdaleka nejzajímavější. Z výsledků profilování vzorku tabulky, které zahrnují také počet záznamů celé tabulky, můžeme odhadnout vlastnosti dat celé tabulky. U statistik profilování popisujících počtové vlastnosti sloupců (počet odlišných hodnot, počet prázdných hodnot atd.) můžeme využít přímého vynásobení hodnoty konkrétní statistiky z profilování vzorku poměrem celkového počtu záznamů tabulky k počtu záznamů vzorku. Tento způsob také prozatím zvolíme, v budoucích rozšířeních pak můžeme zvažovat možné přesnější dopočitání těchto statistik pomocí frekvencí výskytů jednotlivých hodnot apod. U ostatních „nepočtových“ statistik můžeme jejich hodnoty použít přímo.

Na základě odhadu vlastností dat celé tabulky následně můžeme pomocí natrénovaného modelu predikovat délku trvání *úlohy datového profilování* na celé tabulce. Na tento případ užití bychom se chtěli v řešení primárně zaměřit, protože profilování vzorku tabulky zabere zpravidla několik sekund a predikce délky trvání úlohy na celé tabulce, která může trvat několik hodin, by nám umožnila realizovat chytré plánování popsané v sekci 2.2.1.

2.2.1 Modelový příklad využití predikcí v platformě Atacama ONE

Zákazník potřebuje naprofilovat sto tabulek o několika milionech záznamů během jednoho týdne, přičemž profilování by mělo probíhat v nočních hodinách od 22:00 do 6:00, aby se minimalizoval dopad na běžný pracovní provoz. Na základě historických dat a vlastností každé tabulky *ONE* využije predikční model

k odhadu délky trvání jednotlivých *úloh datového profilování* pro každou tabulku. Následně DPM inteligentně naplánuje úlohy tak, aby využíval výpočetní výkon dostupných *DPE* co nejefektivněji, zatímco zohlední časová omezení zákazníka. Fungování tohoto procesu můžeme rozdělit do tří fází:

Fáze analýzy a plánování – Systém *ONE* nejprve během několika jednotek minut provede profilování vzorku každé tabulky, což mu umožní provést odhad vlastností celé tabulky.

Fáze predikce – Na základě odhadů vlastností celé tabulky s využitím predikčního modelu systém *ONE* odhadne celkový čas potřebný pro profilování každé tabulky a rozvrhne úlohy tak, aby vyhověl požadavkům zákazníka a zároveň optimalizoval využití výpočetních zdrojů *DPE*.

Fáze realizace – Profilování se provádí podle naplánovaného rozvrhu. Systém neustále monitoruje průběh a případně upravuje plán v reálném čase na základě aktuálních výsledků a dostupnosti zdrojů. Délky trvání jednotlivých úloh jsou společně s jejich výsledky ukládány a používány pro predikce délek trvání dalších úloh. Na tuto část řešení se ovšem práce nezaměřuje.

Pokud by si zákazník přál tento modelový příklad provádět periodicky, například jednou za měsíc, můžeme v první fázi využít již předtrénovaného modelu z dřívějších běhů a vyhnout se tak nutnosti nejprve tabulky profilovat na jejich vzorku. Oba přístupy mají své výhody, pokud zvolíme vždy způsob profilování vzorku, získáme přesný údaj o počtu záznamů tabulky. Při větším počtu profilovaných tabulek však může i profilování vzorku trvat netriviální časový úsek. V tomto případě tedy může být lepší využít, na úkor znalosti aktuálního počtu záznamů, předtrénovaného modelu, který na vstupu dostane odhad celkového počtu záznamů, jenž budeme dopočítávat z celkových počtů záznamů historických výsledků profilování na každé tabulce a to tak, že zprůměrujeme rozdíly příbytků a úbytků mezi jednotlivými výsledky a tento průměr přičteme k výsledku poslednímu.

Kapitola 3

Predikční model

V této kapitole se budeme věnovat vytváření predikčního modelu od přípravy dat pro určení klíčových metrik až po jejich analýzu.

3.1 Příprava dat

Abychom mohli určit metriky ovlivňující délku trvání *úloh datového profilování*, museli jsme nejdříve připravit tabulky s různými vlastnostmi a následně je naprofilovat. Pro vytváření tabulek jsme využili databázový systém *PostgreSQL* a *PL/pgSQL*[17] – procedurální jazyk pro psaní *SQL* určený pro databázový systém *PostgreSQL*. Zkoumanými proměnnými pro nás byly:

- počet záznamů (řádků) tabulky
- počet sloupců tabulky
- datový typ
  - `STRING`
  - `INTEGER`
  - `LONG`
  - `FLOAT`
  - `BOOLEAN`
  - `DATE`
  - `DATETIME`
- délka a velikost jednotlivých hodnot
- počet prázdných (*Null*) hodnot

- počet odlišných (*Distinct*) hodnot
- rozdělení a uspořádání dat ve sloupcích

Prvním a zásadním krokem bylo provést měření délek trvání většího množství *úloh datového profilování* na tabulkách charakterizovaných různými vlastnostmi dat. Tento proces byl nezbytný k identifikaci vlastností dat, které mají vliv na délku trvání úloh. Rozhodli jsme se pohlížet na algoritmus *úloh datového profilování* jako na „černou skříňku“, a proto se při identifikaci metrik, které zahrneme do našeho predikčního modelu, chceme opřít výhradně o data z těchto měření.

3.1.1 Generování velkých tabulek v PL/pgSQL

Pro generování tabulek s velkým množstvím záznamů (desítky milionů) jsme museli najít efektivní způsob, jelikož jsme narazili na omezení běžných *SQL* funkcí, které byly příliš pomalé. Využili jsme funkce `generate_series`[18], která je v jazyce *PL/pgSQL* navržena pro rychlé generování sekvencí čísel nebo časových intervalů. Klasické metody vkládání dat pomocí cyklů nebo masivních `INSERT` příkazů jsou pro velká data neefektivní, protože každý příkaz vyžaduje samostatné zpracování a komunikaci s databázovým serverem. Funkce `generate_series` minimalizuje tyto nároky tím, že vytváří potřebná data na straně serveru a umožňuje jejich rychlé vložení. Pomocí generování určitého poměru unikátních čísel jsme pak mohli využít hešovacích funkcí[19] pro generování daného počtu navzájem odlišných řetězců (*Výpis kódu 3.1.1*). Celkem jsme takto vygenerovali přes 300 tabulek pokrývajících kombinace zkoumaných proměnných.

```
CREATE TABLE unique_strings AS
SELECT
  substring(md5(i::text || 'salt1'), 1, 10) AS unique_string1,
  substring(md5(i::text || 'salt2'), 1, 10) AS unique_string2
FROM generate_series(1, 100) AS s(i);
```

Výpis kódu 3.1 *Generování tabulky se dvěma sloupci s unikátními záznamy v jazyce* PL/pgSQL

3.2 Měření a výsledky

Úlohy byly spouštěny na lokální instanci ONE v prostředí *Docker* kontejneru, které dostatečně dobře simulovalo produkční běh. Databáze obsahující profilované tabulky však nebyly umístěny lokálně, ale hostovány na vzdáleném serveru, aby se dosáhlo realistického scénáře při běžném nasazení. Časy obou fází úloh datového profilování byly extrahovány ze standardních výstupů běhů úloh, které jsou

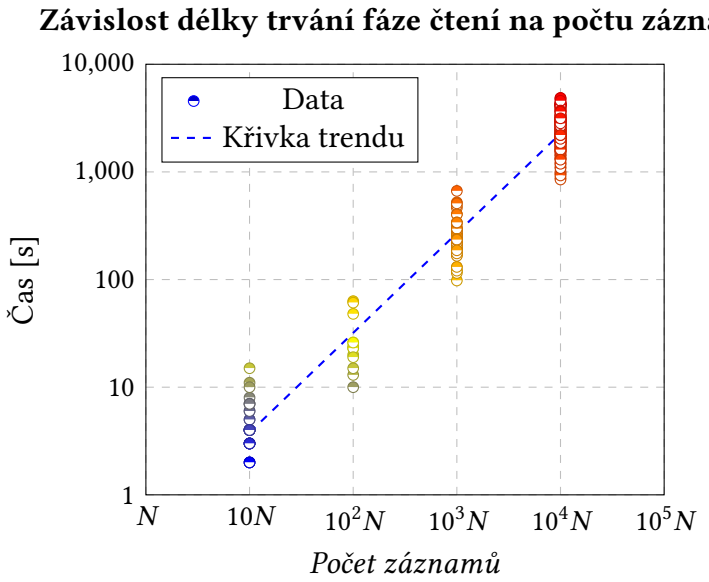
ukládány do objektového úložiště MinIO[20], které je součástí ONE. Protože *Minio* poskytuje *Python* sadu nástrojů pro vývoj[21], jejíž součástí je i klient pro programový přístup k cílovému úložišti, pro automatizované získávání a parsování časů ze standardních výstupů jsme využili skript napsaný právě v jazyce *Python*. Vzhledem k tomu, že úlohy datového profilování na tabulkách o několika milionech záznamů mohou trvat až několik hodin, zvolili jsme velikosti a kombinace vlastností vytvářených tabulek tak, abychom dokázali pokrýt co nejvíce různých konfigurací při rozumném výsledném čase. Celkem jsme spustili okolo devíti set úloh datového profilování s celkovou délkou jejich trvání přes tři sta hodin. Při měření se nám potvrdilo, že bude nutné oddělit predikční modely pro *fázi čtení* a *fázi výpočtu*, jelikož délka jejich trvání závisí na jiných vlastnostech dat tabulek. Tyto vlastnosti a závěry měření si zanalyzujeme v následujících dvou sekcích, v přiložených grafech ovšem nebudeme z důvodu citlivosti dat úvádět přesné počty záznamů tabulek, zachováme ovšem jejich poměr tak, aby z nich byly závislosti jasně patrné.

3.2.1 Fáze čtení

Délka trvání fáze čtení je závislá na rychlosti internetového připojení (*Graf* 3.5) a celkové velikosti dat tabulky, kterou lze vyjádřit pomocí počtu záznamů (*Graf 3.1*), počtu sloupců (*Graf 3.2 a 3.3*) a délce jednotlivých hodnot v řetězcové reprezentaci (*Graf* 3.4). Z měření vyplývá, že doba čtení tabulek se sloupci datových typů různých od typu `STRING`, se rovná době čtení tabulek se sloupci datového typu `STRING`, pokud mají obě tabulky hodnoty o stejné délce v řetězcové reprezentaci. To znamená, že datový typ sloupce je pro čtení irelevantní, důležitá je průměrná délka hodnot každého sloupce v řetězcové reprezentaci. Například tabulka s pěticifernými číselnými hodnotami typů `LONG`, `INTEGER` a `FLOAT` má podobnou délku trvání *fáze čtení* jako tabulka s hodnotami typu `STRING` o pěti znacích. Pro ostatní datové typy to platí obdobně, sloupce s hodnotami typu `BOOLEAN` trvají stejně dlouho jako sloupce s jednoznakovými hodnotami typu `STRING`. Tabulky se sloupci s datumy ve formátu `YYYY-MM-DD` trvají stejně dlouho jako tabulky se sloupci s osmiznakovými hodnotami typu `STRING`. Otestovali jsme také další vlastnosti dat jako počet odlišných hodnot, prázdných hodnot, uspořádanost a rozložení dat. Tyto vlastnosti však na délku trvání *fáze čtení* nemají zásadní vliv (*Tabulka 3.1*).

Při testování různých rychlostí internetového připojení jsme sice občasně naměřili hodnoty, které signalizovaly lineární vztah mezi rychlostí připojení a délkou trvání *fáze čtení*, nemůžeme se na ně ovšem nijak spolehnout, jelikož při kolísání stability připojení se naměřené délky trvání *fáze čtení* stejné tabulky lišily až o vyšší desítky procent. Důležitou roli hraje také aktuální vytížení databáze, ve které se data k profilování nachází. Toto vytížení však nemáme žádnou možnost

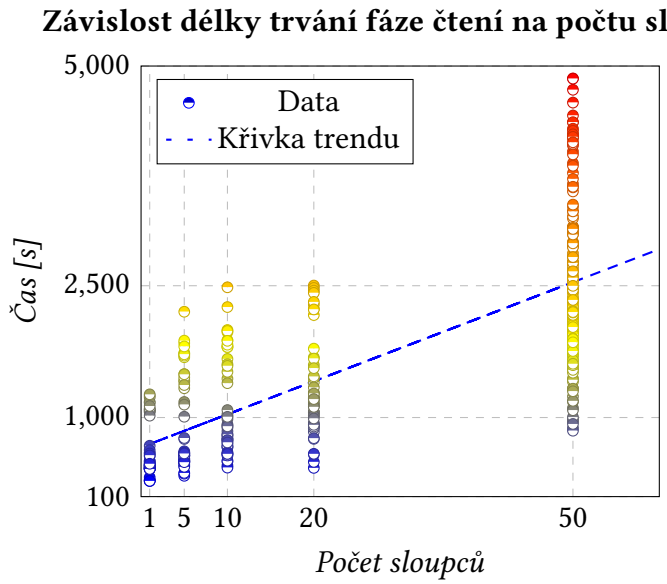
měřit, a proto ho nebudeme v rámci řešení zvažovat.



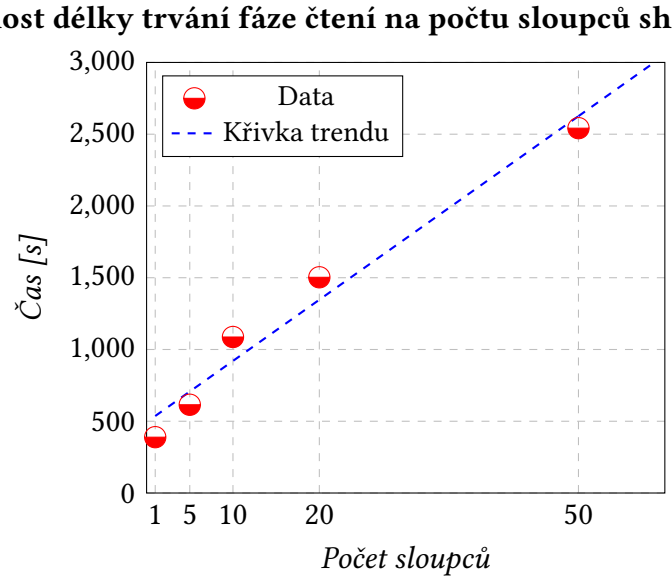
Obrázek 3.1 Graf znázorňuje závislost délky trvání fáze čtení na počtu záznamů v tabulkách s padesáti sloupci bez ohledu na další vlastnosti dat. Na obou osách je pro lepší znázornění použito logaritmické měřítko. Data ukazují, že délka trvání fáze čtení vykazuje lineární trend v závislosti na počtu záznamů.

Vlastnost tabulky	Délka trvání fáze čtení [s]
Všechny hodnoty prázdné	1902
Všechny hodnoty unikátní	2088
Všechny hodnoty stejné	2026
Hodnoty seřazeny sestupně	2107
Hodnoty seřazeny vzestupně	2200

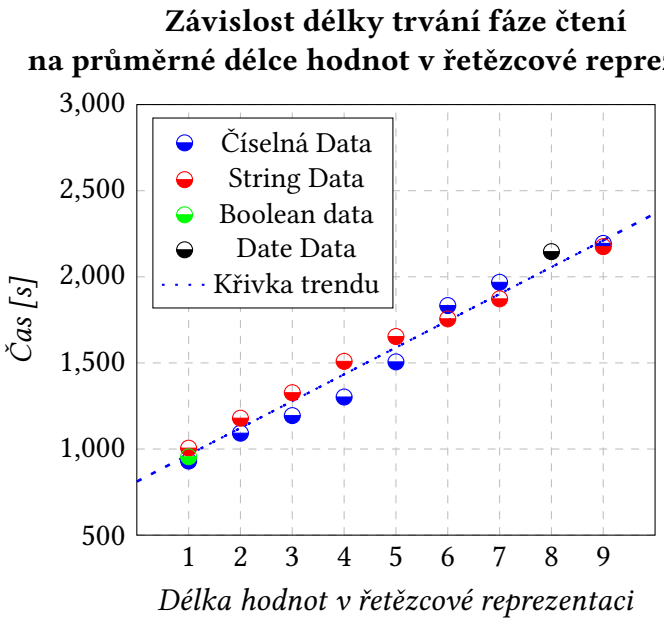
Tabulka 3.1 Tabulka znázorňuje délky trvání fáze čtení pro vlastnosti dat tabulek, které nemají na délku trvání významný vliv. Měření byla prováděna na tabulkách o stejném počtu záznamů a sloupců při stejné rychlosti internetového připojení



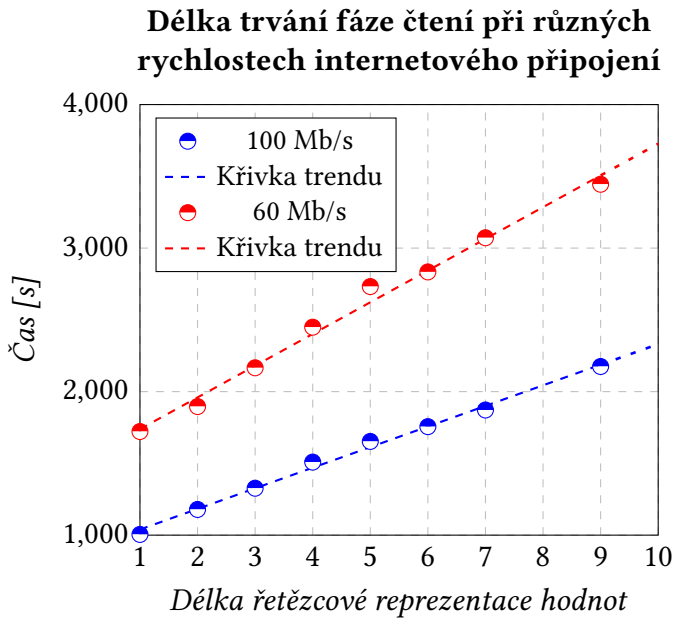
Obrázek 3.2 Graf znázorňuje závislost délky trvání fáze čtení na počtu sloupců v tabulkách se stejným počtem záznamů bez ohledu na další vlastnosti dat. Data ukazují, že délka trvání fáze čtení vykazuje mírně rostoucí trend vůči počtu sloupců.



Obrázek 3.3 Graf znázorňuje závislost průměrné délky trvání fáze čtení na počtu sloupců tabulek se shodnými vlastnostmi. Data ukazují, že pokud mají tabulky shodné vlastnosti, vztah mezi délkou trvání fáze čtení a počtem sloupců vykazuje lineární trend.



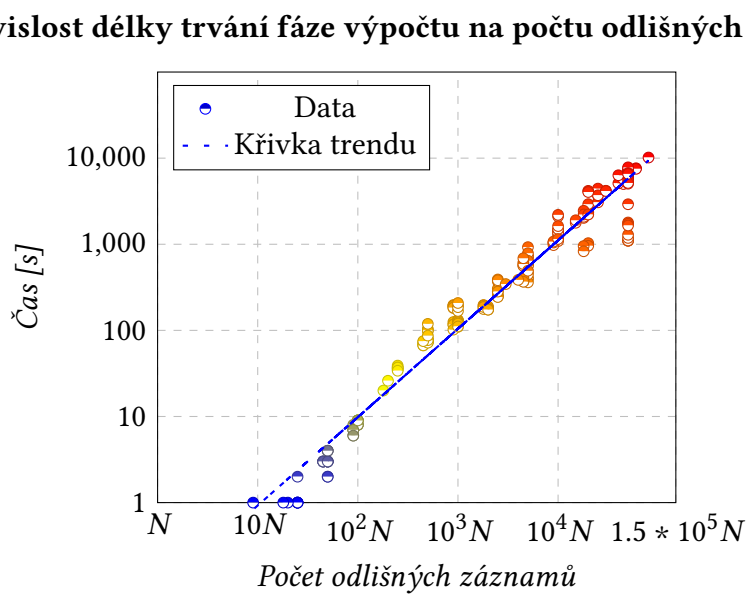
Obrázek 3.4 Graf znázorňuje závislost délky trvání fáze čtení na průměrné délce hodnot v řetězcové reprezentaci v tabulkách s padesáti sloupci bez ohledu na další vlastnosti dat. Na obou osách je pro lepší znázornění použito logaritmické měřítko. Data ukazují, že délka trvání fáze čtení je lineárně závislá na průměrné délce jejích hodnot v řetězcové reprezentaci nezávisle na datovém typu.



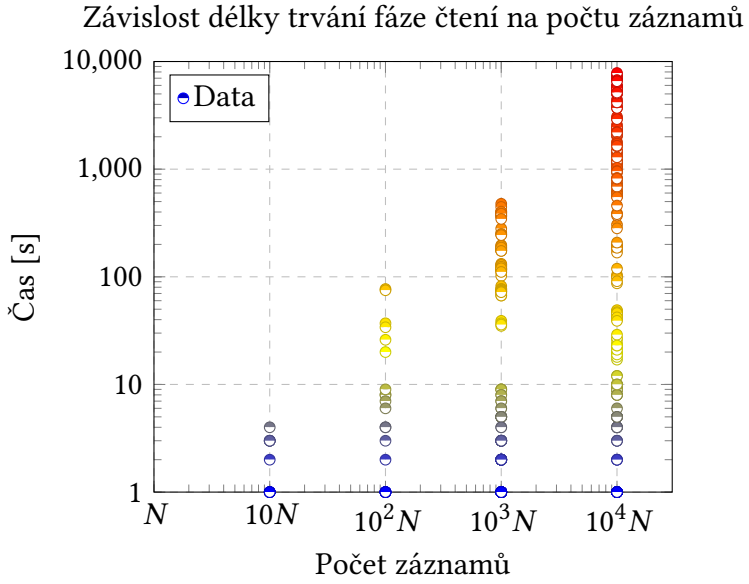
Obrázek 3.5 Graf porovnává délku trvání fáze čtení při dvou různých rychlostech internetového připojení. K porovnání je využita závislost mezi délkou hodnot v řetězcové reprezentaci a délkou trvání fáze čtení na tabulkách se stejným počtem záznamů a padesáti sloupci.

3.2.2 Fáze výpočtu

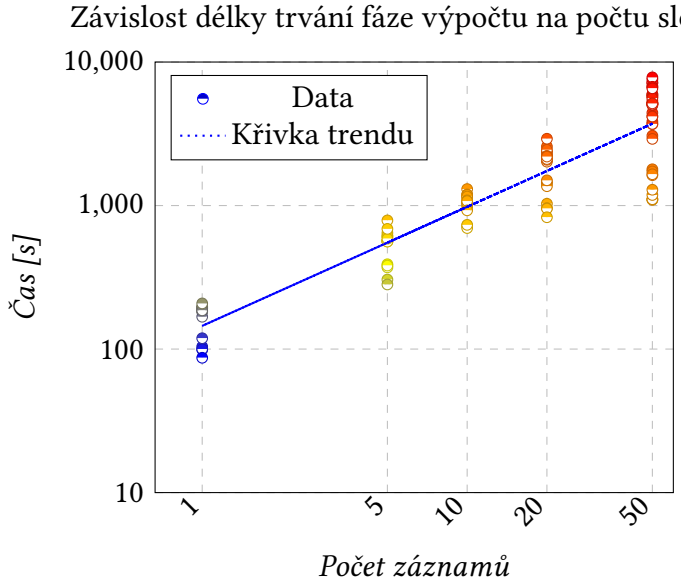
Délka trvání fáze výpočtu je závislá na počtu odlišných hodnot (*Graf 3.6*) a počtu sloupců tabulky (*Graf 3.8*). Na rozdíl od *fáze čtení* není rozhodující celkový počet záznamů, protože pro tabulky o různém počtu záznamů se stejnými hodnotami je délka trvání *fáze výpočtu* prakticky nulová a celkový počet záznamů tabulky tak nemá na délku trvání rozhodující vliv (*Graf 3.7*). Otestovali jsme opět i další vlastnosti, ale žádné na délku trvání fáze výpočtu nemají zásadní vliv.



Obrázek 3.6 Graf zobrazuje závislost délky trvání fáze výpočtu na počtu odlišných záznamů napříč všemi sloupci tabulky. Data ukazují, že počet odlišných záznamů má na délku trvání fáze výpočtu zásadní vliv a je vykazuje vzhledem k délce trvání lineární trend.



Obrázek 3.7 Graf zobrazuje závislost délky fáze čtení na počtu záznamů tabulky



Obrázek 3.8 Graf zobrazuje závislost délky fáze výpočtu na počtu sloupců tabulky

Vlastnost tabulky	Délka trvání fáze čtení [s]
Všechny hodnoty prázdné	0
Všechny hodnoty stejné	0
Hodnoty seřazeny sestupně	5053
Hodnoty seřazeny vzestupně	5149
Všechny hodnoty unikátní délky 8	960
Všechny hodnoty unikátní délky 16	1024

Tabulka 3.2 Tabulka znázorňuje délky trvání fáze výpočtu pro vlastnosti dat tabulek, které nemají na délku trvání významný vliv. Každá skupina oddělená vodorovnou čarou byla měřena na tabulkách o stejném počtu záznamů a sloupcu při stejné rychlosti internetového připojení

3.3 Volba algoritmu

Na základě analýzy z předchozí sekce jsme se pro obě fáze úloh datového profilování rozhodli pro predikce použít regresi. Regresi volíme, protože je vhodná

pro odhadování hodnot závislé proměnné, na základě vztahů mezi nezávislými proměnnými. Regresní modely jde také snadno přizbůsobit a rozšířit o další proměnné, což může být důležité v případě, kdy by bylo třeba model upravit na základě nových poznatků nebo změn v rámci chování algoritmu *úloh datového profilování*.

Pro *fázi čtení* budou k predikci délky jejího trvání použity jako nezávislé proměnné modelu počet záznamů, počet sloupců a průměrná délka hodnot v řetězcové reprezentaci. Pro *fázi výpočtu* budou k predikci délky jejího trvání použity jako nezávislé proměnné počet odlišných hodnot a počet sloupců.

Také jsme vypozorovali, že délku trvání jednotlivých *úloh datového profilování* ovlivňují i vnější vlivy jako rychlost a stabilita internetového připojení, vytížení jednotlivých DPE a vytížení a druh datového zdroje. Z tohoto důvodu nechceme vytvořit jeden univerzální model, který by byl již předtrénovaný pro každého zákazníka, protože by se vnější podmínky, za kterých byl model trénován, mohly výrazně lišit od vnějších podmínek zákazníka a predikce by tak mohly být výrazněji chybné. Model bude tedy dedikovaný pro každého zákazníka a bude se trénovat pouze na jeho výsledcích *úloh datového profilování*, protože pro tvorbu prototypu můžeme předpokládat, že jeden konkrétní zákazník má své datové zdroje umístěny na jednom místě a vnější podmínky budou pro každou nově spuštěnou úlohu velmi podobné. V budoucích rozšířeních můžeme zvážit vytvoření dedikovaných modelů pro každý datový zdroj zvlášť.

Přestože jsou vlastnosti ovlivňující délku trvání *úloh datového profilování* shodné pro profilování celé tabulky i pro profilování jejího vzorku, rozhodli jsme se z důvodu praktického použití pro oba typy vytvořit také dedikovaný model, protože nechceme trénovat model určený pro predikce délky trvání úloh na celých tabulkách pouze na výsledcích jejich vzorků. Přestože profilování vzorku tabulky trvá zpravidla pár jednotek sekund, může se nám schopnost jej predikovat hodit v případě pouštění většího počtu *úloh datového profilování* na vzorcích tabulek.

Kapitola 4

Analýza technických řešení

V této kapitole si rozebereme požadavky na implementaci a použité technologie.

4.1 Požadavky na implementaci

Prvním důležitým požadavkem je, abychom při implementaci splňovali všechny *Ataccama* standardy používané ve vývoji ONE. Vzhledem k tomu, že ONE používá mikroservisní architekturu a ze stávajících mikroservis nedává smysl žádnou rozšiřovat[1], nabízí se přímočará možnost řešení implementovat jako mikroservisu. Tento přístup přináší několik zásadních výhod, jako je možnost separátně vydávat nové verze bez závislosti na ostatních modulech. Vzhledem k očekávaným vyšším nárokům na paměť, protože budeme zaznamenávat až desetitisíce úloh, nám mikroservisní řešení umožní také lepší kontrolu nad zdroji, jelikož mikroservisa bude provozována v dedikovaném kontejneru.

Dále jsme požadavky rozdělili na funkční a mimofunkční a v pořadí jim přiřadili identifikátory ve formátu **P-ID**. Požadavek na implementaci řešení jako mikroservisu označme **P-0**.

4.1.1 Funkční požadavky

P-1 Mikroservisa automaticky detekuje a zpracovává nové úlohy spuštěné v ONE.

P-2 Mikroservisa podporuje asynchronní zpracování více detekovaných úloh najednou.

[1] princip *Separation of Concerns*

P-3 Mikroservisa umožňuje import metadat již dokončených úloh, které nebyly zaznamenány během jejího dočasného výpadku.

P-4 Mikroservisa poskytuje REST API, které pro daný identifikátor tabulky vrátí odhad délky trvání jeho profilování na základě predikčního modelu.

P-5 Mikroservisa umožňuje dynamické znovunatrénování predikčního modelu na základě nově zpracovaných úloh.

P-6 Architektura mikroservisy je navržena tak, aby bylo snadné přidání podpory pro predikci dalších typů úloh.

4.1.2 Mimofunkční požadavky

P-7 Mikroservisa používá existující *ONE API* bez možnosti jej modifikovat.

P-8 Predikční model by měl dosáhnout přesnosti, která je obecně spolehlivá, přičemž nečasté výraznější odchylky jsou možné.

P-9 Mikroservisa efektivně řeší správu metadat úloh, aby byly zachovány rozumné paměťové nároky a relevance dat tak, že pravidelně odstraňuje metadata příliš starých úloh.

Můžeme si všimnout, že požadavek **P-8** je definován volněji a požaduje *obecně spolehlivou* přesnost. Důvodem této formulace jsou výsledky měření z předchozí kapitoly. Na délku trvání úloh datového profilování má vliv spousta vnějších vlivů jako například aktuální vytížení datového zdroje, rychlost a stabilita internetového připojení, dostupnost a vytížení jednotlivých DPE. Tyto vlivy nemůžeme ovlivnit ani měřit a naměřené výsledky ukázaly, že se délky běhu úloh mohou výrazněji lišit i při datovém profilování stejné tabulky za rozdílných vnějších podmínek. Z tohoto důvodu nevyžadujeme, aby přesnost predikčního modelu byla extrémně vysoká, dosažení řádové přesnosti, která nám v budoucnu umožní implementovat chytré plánování popsané v sekci 2.2.1, je dostatečné.

4.2 Použité technologie

Pro implementaci řešení bylo nutné zvolit jazyk postavený na Java Virtual Machine, a to z důvodu údržby a budoucí rozšiřitelnosti řešení stávajícími vývojáři používajícími primárně jazyk *Java*[22]. Pro vývoj mikroservisy tedy připadaly v úvahu jazyky *Java* a *Kotlin*[23]. Jako primární jazyk implementace pak byl zvolen jazyk *Java*, a to převážně díky daleko větším zkušenostem autora s tímto jazykem. V některých částech implementace a při psaní testů byl ovšem díky pokročilejším

syntaktickým možnostem využit také jazyk *Kotlin*. Jazyky *Java* a *Kotlin* jsou interoperabilní, protože jsou oba překládány do *Java bytecode*, který je následně interpretován pomocí Java Virtual Machine. Jako nástroj pro sestavení a správu projektu byl použit nástroj *Gradle*[24]. Jeho základem je soubor `build.gradle` napsaný v jazyce *Groovy*[25], který definuje základní informace o projektu, závislosti na knihovnách třetích stran, případně jejich konfigurace. Při sestavení aplikace zpracovává pouze ty soubory, které byly změněny od předchozího sestavení, což značně zvyšuje jeho výkon a rychlost[2]. Jako alternativu nástroje pro sestavování aplikace jsme zvažovali nástroj Apache Maven[26], avšak jeho nevýhodami oproti nástroji *Gradle* jsou nižší uživatelská přívětivost (převážně kvůli nutnosti konfigurace projektu v jazyce *XML*), horší výkon a správa závislostí.[27] V následujících sekcích si popíšeme technologie použité ve vývoji mikroservisy. Technologie využité při jejím testování popisujeme společně se strukturou testů v sekci 6.3

4.2.1 Spring framework a Spring Boot

Spring[28] je moderní open-source framework určený pro vývoj enterprise aplikací v jazyce *Java*. *Spring Boot*[29] je framework určený pro vývoj mikroservisních aplikací (splnění požadavku **P-0**) v jazyce *Java* postavený na základech *Spring* frameworku obohaceném o jednodušší možnosti konfigurace a knihovny třetích stran ve formě startovacích balíčků poskytujících nástroje pro rychlejší a jednodušší vývoj, například:

`spring-boot-starter-web` – podpora pro tvorbu RESTful služeb včetně vestavěného Tomcat[30] serveru

`spring-boot-starter-data-jpa` – podpora práce s relačními databázemi pomocí Java Persistance a využití objektově-relačního mapování pomocí technologie *Hibernate*[31]

`spring-boot-starter-test` – podpora pro testování Spring Boot aplikací

`org:testcontainers` – podpora pro kontejnerové testy pomocí technologie *Docker*[32]

Další zásadní výhodou *Spring* frameworku je jeho implementace principu *Inversion of Control* v jeho specializované formě zvané *dependency injection*[33], jejíž principem je schopnost objektů určovat si vlastní závislosti, a to pouze přes parametry konstruktoru, tovární metody, případně *settery*. Výhodami použití tohoto principu jsou snížení vazby mezi jednotlivými komponentami, modulárnost a

[2]Tento proces se nazývá *inkrementální sestavování*

možnost izolovaně testovat jednotlivé části kódu. Použití jiného frameworku jsme nezvažovali, jelikož *Spring Boot* plně vyhovuje našim požadavkům, je zdaleka nejpopulárnějším frameworkem pro vývoj mikroservisních aplikací v jazyce Java a používá se v rámci vývoje ONE.

4.2.2 Project Lombok

Project lombok [34] je knihovna v jazyce *Java*, která za použití *Java* anotací umožňuje generování rutinního (anglicky *boilerplate*) kódu jako jsou například konstruktory, *gettery*, *settery*, metody `equals`, `toString` a další. Při definici datové třídy můžeme například využít anotaci `@Data`, která automaticky pro třídu vygeneruje konstruktor pro všechny `final` atributy, *gettery*, *settery* pro všechny `non-final` atributy a metody `toString`, `equals` a `hashCode`. Další hojně využívanou anotací je pak `@Builder`, která pro třídu implementuje návrhový vzor *Builder* a umožní nám tak postupné vystavování objektů.

4.2.3 GraphQL klient

Pro komunikaci s *ONE API* je třeba zvolit vhodnou knihovnu s GraphQL klientem. Zvažovali jsme tři alternativy, *GraphQL Java*[35], *DGS Framework* a *Apollo Kotlin*[36].

GraphQL Java je základní knihovna pro práci s *GraphQL* v jazyce Java. Je navržena pro nízkoúrovňové operace, což uživatelům poskytuje vysokou míru flexibility a kontrolu nad zpracováním dat. Neposkytuje však některé pokročilejší funkce jako například automatické mapování GraphQL operací do tříd, což vede ke zvýšené složitosti a délce kódu.

DGS Framework je framework navržený primárně pro tvorbu *GraphQL* serverových aplikací s použitím frameworku *Spring Boot*. Jeho integrace s frameworkem *Spring Boot* zjednodušuje nastavení a zvyšuje produktivitu vývoje díky přehledné konfiguraci a podpoře *Java* anotací. Nicméně, není primárně určen pro tvorbu *GraphQL* klientů a neposkytuje některé funkcionality jako *Apollo Kotlin*.

Apollo Kotlin[36] je GraphQL klient umožňující generování Java modelů přímo z GraphQL operací. Jeho výhodou je, že převádí řetězcovou reprezentaci GraphQL operací do Kotlin případně Java tříd, umožňuje s jednotlivými operacemi pracovat jako se standardními objekty jazyka a zaručuje tak typovou kontrolu. Stejně tak automaticky mapuje do objektové reprezentace i odpovědi, tudíž se uživatel nemusí starat o jejich parsování. Dalšími výhodami jsou také široká podpora pro testování pomocí mockování odpovědí a tzv. *Data builders*[37], které umožňují jednoduchým způsobem manuálně vytvářet předdefinované odpovědi GraphQL operací. Apollo Kotlin také automaticky integruje do *Gradle* tasky umožňující přegenerování Java modelů, stahování GraphQL schémat pomocí introspekce a

další. Pro reaktivní programování *Apollo Kotlin* doporučuje rozšíření *RxJava* [38] umožňující provádět asynchronní operace *GrapQL* a přijímat výsledky těchto operací jako tzv. *observables*. Díky této funkcionalitě nám *Apollo Kotlin* umožňuje splnit požadavek **P-2**. Po zvážení všech těchto faktorů jsme se rozhodli pro *Apollo Kotlin* kvůli jeho uživatelské přívětivosti, integraci s moderními vývojovými postupy a výborné podpoře pro tvorbu a konfiguraci *GraphQL* klientů.

4.2.4 Knihovna pro predikční model

Volba knihovny pro predikční model je vázaná na volbu jazyka *Java*. Vzhledem k tomu, že *Java* není úplně běžně používaný jazyk pro tyto typy úloh, bylo nalezení vhodné knihovny relativně obtížné. Představme si dvě knihovny, které poskytují širokou škálu regresních algoritmů a jsou vhodné pro použití v rámci naší mikroservisy.

Weka[39] je open-source software programů strojového učení napsaný v jazyce *Java*, který poskytuje *Java* API pro použití jednotlivých komponent v kódu. Obsahuje velmi širokou škálu algoritmů zaměřených na prediktivní modelování, je ovšem dostupná pouze pod licencí *GNU General Public License*[40] a není tak vhodná pro komerční použití[41]. *Tribuo*[42] je knihovna pro vytváření modelů strojového učení napsaná v jazyce *Java* vyvíjená týmem Oracle Labs[43] pod licencí *Apache 2.0*.[44]. Poskytuje širokou škálu regresních algoritmů pro lineární i nelineární regresi včetně integrace nativních knihoven jako *TensorFlow* nebo *XGBoost*. Vzhledem k jejímu aktivnímu vývoji, použití moderních přístupů jazyka a široké škále podporovaných algoritmů jsme pro implementaci predikčního modelu zvolili právě tuto knihovnu.

4.2.5 Docker a Docker Compose

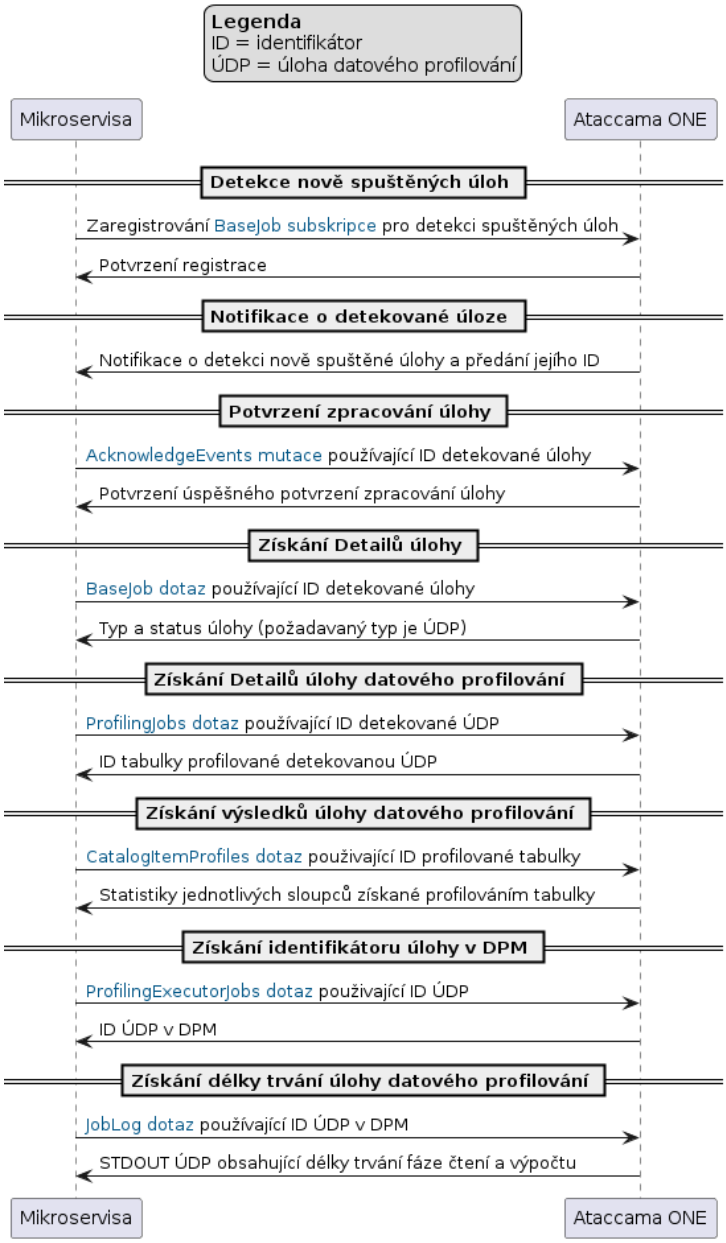
Docker[32] je platforma pro vytváření, nasazování a správu izolovaných aplikací v dedikovaných kontejnerech. Kontejnery umožňují balení aplikace společně s jejím prostředím a závislostmi do samostatného balíčku, který je konzistentní napříč různými vývojovými a produkčními prostředími. *Docker Compose*[45] je nástroj pro definování a správu vícekontejnerových aplikací. Umožňuje pomocí jediného souboru `docker-compose.yml` definovat celou aplikaci včetně nastavení jejích parametrů a konfigurací potřebných pro její spuštění. Mikroservisa využívá *Docker Compose* pro správu kontejnerů pro aplikaci *Spring Boot* a databázi *PostgreSQL*. Soubor `docker-compose.yml` je hlavní konfigurační soubor pro definici těchto kontejnerů a jejich správu. Podrobnosti o konfiguraci a spouštění aplikace jsou popsány v přiloženém `README.md`.

Kapitola 5

Komunikace s platformou Ataccama ONE

K detekci nově spuštěných úloh v ONE a získávání jejich metadat potřebných pro trénink predikčního modelu je nutné využít *ONE API* – rozhraní umožňující se pomocí *GraphQL* dotazovat a manipulovat s daty v ONE. Vzhledem k tomu, že *GraphQL* schéma metadata modelu je velmi rozsáhlé (obsahuje více než deset tisíc typů), bylo nezbytné nejdříve schéma nastudovat a navrhnout posloupnost *GraphQL* operací umožňující při splnění požadavku **P-7** tato metadata získat. Vzhledem k rozsáhlosti schématu se budeme věnovat pouze jeho částem potřebným v kontextu řešeného problému a jednotlivé *GraphQL* operace si včetně jejich parametrů[1] a odpovědí popíšeme v následujících sekcích této kapitoly. Sekvenční diagram v jazyce *UML* (*Obrázek 5.1*) popisuje posloupnost volání jednotlivých operací a tok dat mezi mikroservisou a platformou ONE.

[1]Vykřičník u datového typu parametru znamená, že hodnota nemůže být *null*



Obrázek 5.1 Sekvenční diagram popisující detekci nově spuštěných úloh v reálném čase a následnou posloupnost *GraphQL operací* pro získání jejich metadat potřebných pro trénink predikčního modelu.

```
subscription BaseJob ($subscriptionId: GID!, $ackLimit: Int!, $
    chunkSize)  {
    _externalEvents(
    subscriptionId: $subscriptionId,
    ackLimit: $ackLimit,
    chunkSize: $chunkSize
    entityType: "baseJob"
  ) {
    events {
        id
        entityId
    }
  }
}
```

Výpis kódu 5.1 *BaseJob subskripce*

5.1 MMM API

MMM API je část *ONE API* pro komunikaci s MMM umožňující dotazovat se na medatata a výsledky profilování jednotlivých tabulek. *GraphQL* operace jsou v případě *dotazů*[46] a *mutací*[47] posílány na endpoint `<ONE_platform_URL>/graphql`, který odpovídá umístění *HTTP* serveru MMM. V případě *GraphQL subskripcí*[48] jsou přesměrovány na endpoint `<ONE_platform_URL>/subscripti`

5.1.1 BaseJob subskripce

Subskripce je navržena tak, aby detekovala jakoukoli změnu stavu libovolného typu úlohy, jenž je potomkem entity `baseJob` a umožňovala tak v reálném čase reagovat na příchozí události a zpracovávat je (*Výpis kódu 5.1*). Entita `baseJob` je rodičovská entita reprezentující libovolný typ úlohy, které ONE umožňuje. V případě úloh datového profilování se potomek entity `baseJob` nazývá `profilingJob`, mohli bychom tedy detekovat pouze tyto úlohy, ovšem z důvodu rozšiřitelnosti o další typy úloh v budoucnu volíme obecnou entitu `baseJob`.

Subskripce očekává čtyři parametry (*Tabulka 5.1*). Odpovědí jsou události (`events`), které obsahují kromě vlastních identifikátorů také identifikátory entit (`entityId`), na které se událost vztahuje, v našem případě se jedná o identifkátory detekovaných úloh. Identifikátor poslední přijaté události využijeme při popisu potvrzování jejich zpracování v sekci 5.1.2 a identifikátor detekované úlohy při dotazování na její typ a status v sekci 5.1.3

```
mutation AcknowledgeEvents($id: GID!, $lastEventId: Long!){
    _acknowledgeExternalEvents(subscriptionId: $id, lastEventId: $
        lastEventId)
}
```

Výpis kódu 5.2 *AcknowledgeEvents mutace*

Parametr	Typ	Popis
subscriptionId	GID!	Unikátní identifikátor subskripce.
ackLimit	Int	Maximální počet událostí, které lze přijmout bez potvrzení serveru.
chunkSize	Int	Maximální počet událostí, které lze poslat v jedné odpovědi.
entityType	String	Typ entity, na kterou se subscripce vztahuje, v tomto případě `baseJob`.

Tabulka 5.1 Parametry *BaseJob subskripce*

5.1.2 AcknowledgeEvents mutace

Zpracování událostí je nutné serveru potvrdit *mutací* (*Výpis kódu 5.2*) alespoň jednou za počet přijetí událostí specifikovaný v parametru `ackLimit` *BaseJob* subskripce (5.1.1). *Mutace* očekává dva parametry (*Tabulka 5.2*) a v odpovědi se dozvíme, zda-li bylo potvrzení úspěšné.

Parametr	Typ	Popis
id	GID!	Unikátní identifikátor subskripce, přijetí jejíž událostí potvrdit.
lastEventId	Long!	Identifikátor události, jejíž zpracování, včetně všech událostí jí předcházejících, potvrdit.

Tabulka 5.2 Parametry *AcknowledgeEvents mutace*

```
query BaseJob($jobId: GID!) {
    baseJob(gid: $jobId) {
        gid
        publishedVersion {
            status
            type
        }
    }
}
```

Výpis kódu 5.3 *BaseJob* dotaz

5.1.3 BaseJob dotaz

Poté, co získáme identifikátor úlohy, musíme se dotázat na její status a typ (*Výpis kódu 5.3*). V případě úloh datového profilování nás zajímají úlohy typu `PROFILING` a abychom mohli začít výsledek úloh zpracovávat, musí být dokončené, neboli mít status `FINISHED`. Dotaz očekává jediný parametr (*Tabulka* 5.3), a to identifikátor úlohy získaný již z *BaseJob subskripce* (5.1.1).

Parametr	Typ	Popis
jobId	GID!	Unikátní identifikátor úlohy.

Tabulka 5.3 Parametry *BaseJob dotazu*

5.1.4 ProfilingJobs

Jakmile jsme pomocí *BaseJob* dotazu (5.1.3) detekovali novou dokončenou úlohu datového profilování, můžeme se dotázat jakou tabulku profilovala. Dotaz *ProfilingJobs* (*Výpis kódu 5.4*) očekává tři parametry (*Tabulka* 5.4) a používáme ho nejenom pro získání identifikátoru tabulky, který se v odpovědi nachází pod položkou `linkedEntityId`, ale také pro získávání všech identifikátorů úloh datového profilování v časovém intervalu, kdy byla mikroservisa nedostupná. Pro oba případy využíváme jiný filtr, v prvním případě do odpovědi filtrujeme pouze jediný `profilingJob`, jehož identifikátor je shodný s identifikátorem `baseJob` získaným v *BaseJob dotazu* (*Výpis kódu 5.1.3*). Ve druhém případě do odpovědi filtrujeme všechny úlohy `profilingJob`, které byly dokončeny po zadané časové značce.

```
query ProfilingJobs($filter: [Filter!]!, $size: Int, $skip:Int) {
  profilingJobs(
    filter: $filter
    versionSelector: { publishedVersion: true }
    orderBy: { property: "finishedAt", direction: DESC }
    size:$size
    skip:$skip
  ) {
    edges {
      node {
        gid
        publishedVersion {
          linkedEntityId
        }
      }
    }
  }
}
```

Výpis kódu 5.4 *ProfilingJob* dotaz

Parametr	Typ	Popis
filter	[Filter!]!	Filtr, který do odpovědi vyfiltruje pouze entity splňující jeho definici.
size	Int	Počet entit `profilingJob` v odpovědi.
skip	Int	Počet entit `profilingJob`, které v odpovědi přeskočit.

Tabulka 5.4 Parametry *ProfilingJobs dotazu*

5.1.5 CatalogItemProfiles dotaz

Entita `catalogItemProfile` obsahuje výsledky a statistiky úloh datového profilování. Přestože hned z `baseJob` dotazu máme k dispozici identifikátor úlohy datového profilování, ve schématu neexistuje žádná vazba mezi entitami typu `profilingJob` a `catalogItemProfile`. Z tohoto důvodu *CatalogItemProfile dotaz* (Výpis kódu 5.5, 5.6, 5.7) používá dříve získaný identifikátor tabulky a to tak, že do odpovědí vyfitruje pouze entity `catalogItemProfile` na tabulce, které se úloha týkala. Jednotlivé entity `catalogItemProfile` řadíme podle času jejich profilování, abychom je mohli správně namapovat na odpovídající entity typu `profilingJob`. Předpokládáme, že pokud jsou spuštěny dvě úlohy datového profilování na stejné tabulce ve stejný čas, tak přestože mohou být při para-

lelním běhu dokončeny v opačném pořadí, než byly spuštěny, jejich výsledky budou velmi pravděpodobně stejné, protože během krátkého časového úseku nemohlo dojít k modifikaci tabulky. Absence vazby mezi entitami `profilingJob` a `catalogItemProfile` řešení poměrně komplikuje a jedná se tak o vhodné místo pro rozšíření ONE, které by umožnilo lepší práci s *ONE API* v budoucnu. Každá entita `catalogItemProfile` obsahuje entity typu `attributeProfile`, které obsahují výsledky profilování jednotlivých sloupců tabulky. Ty získáme dynamicky pomocí poddotazu `getProfileData`, který jako parametr přijímá názvy konkrétních statistik, například `distinctCount`, který v odpovědi obsahuje počet odlišných hodnot v konkrétním sloupci. V našem případě se v odpovědi nacházejí všechny dostupné statistiky.

Parametr	Typ	Popis
filter	[Filter!]!	Filtr, který do odpovědi vyfiltruje pouze entity splňující jeho definici.
size	Int	Počet entit `CatalogItemProfile` v odpovědi.
skip	Int	Počet entit `CatalogItemProfile`, které v odpovědi přeskočit.

Tabulka 5.5 Parametry `CatalogItemProfile dotazu`

Parametry `size` a `skip` využijeme při získávání dat o historických jobech, které jsme z důvodu dočasného výpadku nezaznamenali.

5.1.6 ProfilingExecutorJobs dotaz

Entita `profilingExecutorJob` reprezentuje úlohu spuštěnou v DPM. Obaluje entitu `ProfilingJob`, pomocí jejíhož identifikátoru, získaného v *BaseJoB dotazu* ze sekce 5.1.3, do odpovědi vyfiltrujeme jediný `profilingExecutorJob` (*Výpis kódu 5.8*), jehož identifikátor použijeme v sekci 5.2.1 pro získání standardního výstupu úlohy, ze které získáme časy obou fází příslušné úlohy datového profilování.

5.2 DPM API

DPM API je část ONE API pro komunikaci s DPM. GraphQL dotazy jsou v případě *dotazů* a *mutací* posílány na endpoint `dpm-<ONE_platform_URL>/graphql`, který odpovídá umístění HTTP serveru Data Processing Modulu.

```
query catalogItemProfiles($filter: [Filter!]!, $size: Int, $skip: Int)
    {
  catalogItemProfiles(
    versionSelector: { publishedVersion: true }
    orderBy: { property: "profiledAt", direction: DESC }
    filter: $filter
    size: $size
    skip: $skip
  ) {
    edges {
      node {
        gid
        getProfileData(properties: [], anomalousOnly: false, size: 1)
            {
          ...history
        }
        publishedVersion {
          profileType
          profiledAt
          attributeProfiles {
            edges {
              node {
                publishedVersion {
                  attribute {
                    publishedVersion {
                      dataType
                    }
                  }
                }
                getProfileData(properties: [], anomalousOnly: false,
                    size: 1) {
                  ...history
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Výpis kódu 5.5 *CatalogItemProfiles dotaz*

```
fragment dataHistory on DataHistory {
  values {
    ...value
  }
}
```

Výpis kódu 5.6 *dataHistory fragment*

```
fragment valueHistory on ValueHistory {
    name
    ... on LongValueHistory {
        integerValues {
            value
        }
    }
    ... on DoubleValueHistory {
        doubleValues {
            value
        }
    }
    ... on DoubleValueScoreHistory {
        doubleValues {
            value
        }
    }
    ... on StringValueHistory {
        stringValues {
            value
        }
    }
    ... on BooleanValueHistory {
        booleanValues {
            value
        }
    }
}
```

Výpis kódu 5.7 *valueHistory fragment*

```
query ProfilingExecutorJobs($filter: [Filter!]!) {
  profilingExecutorJobs(
    filter: $filter
    versionSelector: { publishedVersion: true }
  ) {
    edges {
      node {
        gid
        publishedVersion {
          job {
            gid
          }
        }
      }
    }
  }
}
```

Výpis kódu 5.8 *ProfilingExecutorJobs dotaz*

```
query JobLog($id: String!) {
    getJobLog(id: $id, logType: STD_OUTPUT) {
        edges
        __typename
    }
}
```

Výpis kódu 5.9 *JobLog* dotaz

5.2.1 JobLog

Vzhledem k pozorováním učiněným v sekci 3.2 potřebujeme určit délku trvání fáze čtení a výpočtu úlohy datového profilování. Jediným možným způsobem je tyto údaje získat ze standardního výstupu běhu úlohy, jehož serializovaná kopie je uložena v řetězcové reprezentaci tak, že jeden řádek výstupu odpovídá jedné hraně (`edges`) v odpovědi.

Parametr	Typ	Popis
id	String!	Identifikátor `profilingExecutorJob` získaný z MMM API.

Tabulka 5.6 Parametry *JobLog dotazu*

Odpovědí jsou jednotlivé řádky standardního výstupu úlohy (*Výpis kódu 5.10*),

```
1  27.03.2024 13:26:10 [INFO]     Starting runtime...
2  27.03.2024 13:26:10 [INFO]     Running runtime...
3  27.03.2024 13:26:10 [DEBUG]    Starting runnables...
4  27.03.2024 13:26:10 [DEBUG]   [in] Reading...
5  27.03.2024 13:56:13 [DEBUG]   [in] Reading done
6  27.03.2024 13:56:13 [DEBUG]   [in] Batch finishing...
7  27.03.2024 13:56:13 [DEBUG]   [in] Computing main...
8  27.03.2024 14:26:13 [DEBUG]   [in] Computing done
9  27.03.2024 14:26:13 [DEBUG]   [in] Done
10 27.03.2024 14:26:13 [DEBUG]    Writing result...
11 27.03.2024 14:26:13 [INFO]     Stopping runtime...
12 27.03.2024 14:26:13 [INFO]     Finished!
```

Výpis kódu 5.10 *Fragment standardního výstupu úlohy datového profilování. Časový úsek mezi řádky 4 a 5 udává délku fáze čtení, mezi řádky 7 a 8 délku fáze výpočtu.*

ze kterého získáme požadované údaje.

Kapitola 6

Implementace

Implementace byla verzována pomocí technologie *Git*[49] na firemní instanci služby *GitLab*[50]. V průběhu bylo nutné z důvodu interních změn technologií v rámci společnosti *Ataccama* přejít na novější verze jazyka *Java* a frameworku *Spring Boot*, konkrétně pak z *Java 11* na *Java 17* a *SpringBoot 5* na *SpringBoot 6*. Výsledná aplikace je kontejnerizována pomocí technologie *Docker* a její spuštění je realizováno prostřednictvím příkazu
`docker compose up`. Návod na spuštění, konfiguraci, spouštění testů a další je podrobněji popsán v přiloženém `README.md`.

Vzhledem k tomu, že je aplikace určená výhradně jako integrovatelná mikroservisa do *ONE*, její provoz je závislý na dostupnosti aktivní instance *ONE*, jejíž *URL* adresa je konfigurována v souboru `docker-compose.yml`. Pokud aplikace nenalezne běžící webovou instanci *ONE*, standardní spuštění aplikace se bude neúspěšně pokoušet připojit k serveru *MMM* a nedokáže detekovat nově spuštěné úlohy. *REST API* pro dotazování se na délku trvání úloh zůstává funkční. Avšak podle závěrů analýzy uvedených v sekci 3.2 mikroservisa při prvním spuštění neobsahuje žádná uložená metadata úloh ani serializované modely.

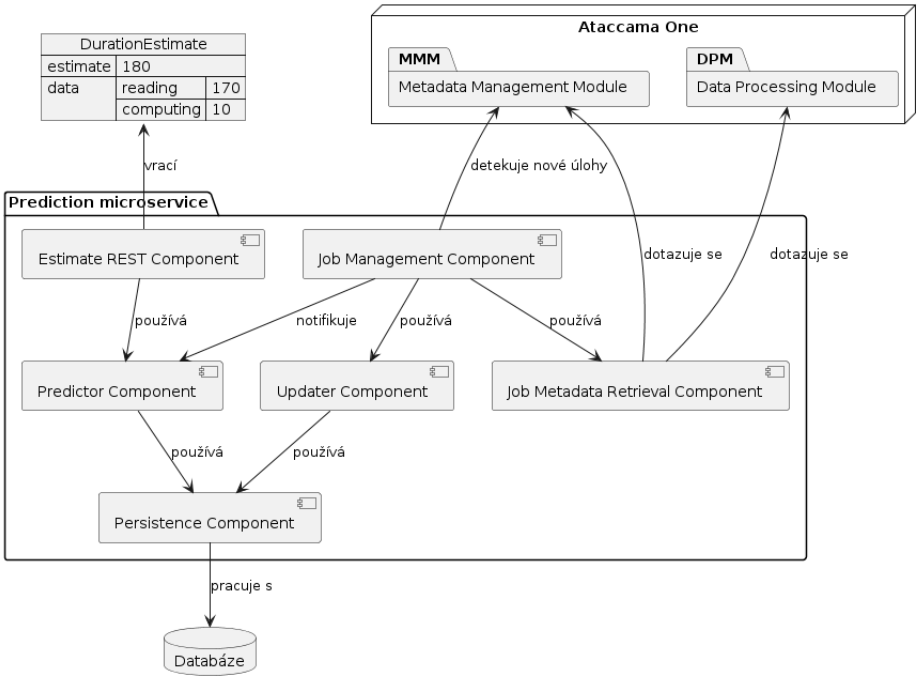
Architektura je navržena tak, aby byla snadno rozšiřitelná, a umožňuje tak jednoduché přidání podpory pro další typy úloh (splnění požadavku**P-6**). *BaseJob subskripce* a *BaseJob dotazy* popsané v kapitole 5 jsou navrženy tak, aby detekovaly a zpracovávaly libovolné úlohy spuštěné *one* a všechny komponenty využívají dostatečnou míru abstrakce pro implementaci tříd pro další typy úloh.

6.1 ApolloClient a Kotlin

Před zahájením implementace bylo nezbytné ověřit, zda je možné pomocí knihovny *Apollo Kotlin* a její třídy `ApolloClient` úspěšně komunikovat pomocí *ONE API* s ONE. Inicializace třídy `ApolloClient` v jazyce *Java* byla kompli-

kovaná, především kvůli problémům s autentizací pomocí autorizačních *HTTP* hlaviček. Všechny pokusy o komunikaci s *ONE API* selhaly a vrátily *HTTP* stavový kód 401 signalizující neautorizovaný přístup. Rozhodli jsme se proto klienta implementovat v jazyce Kotlin, jehož syntaxe nám umožnila lépe namapovat hodnotu autorizační hlavičky a úspěšně posílat autorizované požadavky. Mikroservisa využívá dva klienty, zvlášť pro komunikaci s *MMM* a *DPM*. Klient pro komunikaci s MMM navíc kvůli nutnosti nepřetržitého spojení pro detekci nově příchozích úloh pomocí *GraphQL subskripce* (splnění požadavku **P-1**) využívá zotavovací mechanismus, který umožňuje automatické obnovení spojení při neočekáváném výpadku.

6.2 Architektura

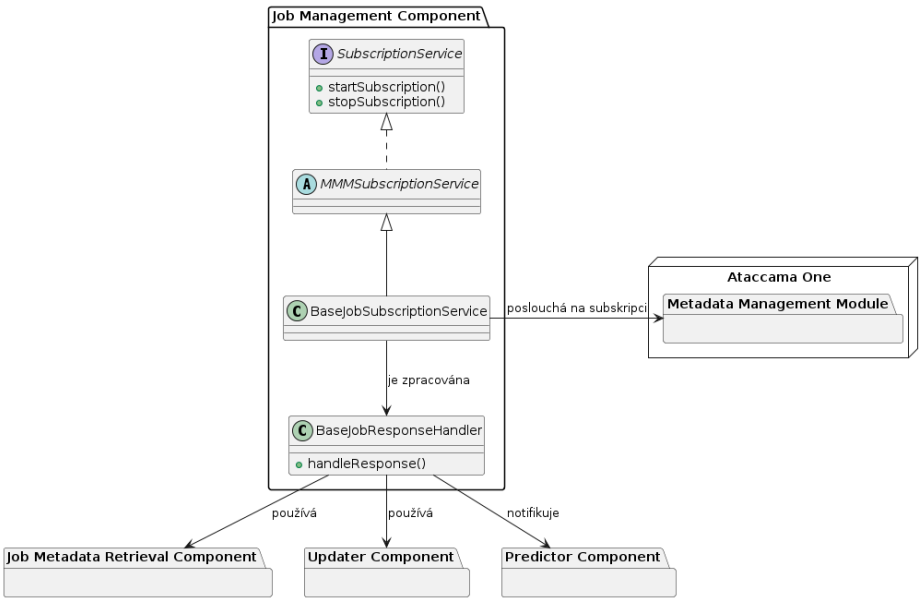


Obrázek 6.1 Diagram komponent

Architektura mikroservisy používá komponentovou strukturu a umožňuje tak lépe zvládnout složitost systému tím, že rozkládá celkovou funkcionalitu na menší, samostatně fungující a snadno spravovatelné části. Každá komponenta je navržena tak, aby plnila specifické funkce v rámci celkového systému a byla relativně nezávislá na ostatních komponentách. Pro grafické znázornění závislostí použijeme *UML*[51] diagramy. Obrázek 6.1 znázorňuje diagram komponent a jejich

integraci s platformou ONE. Jednotlivé komponenty si popíšeme v následujících sekcích.

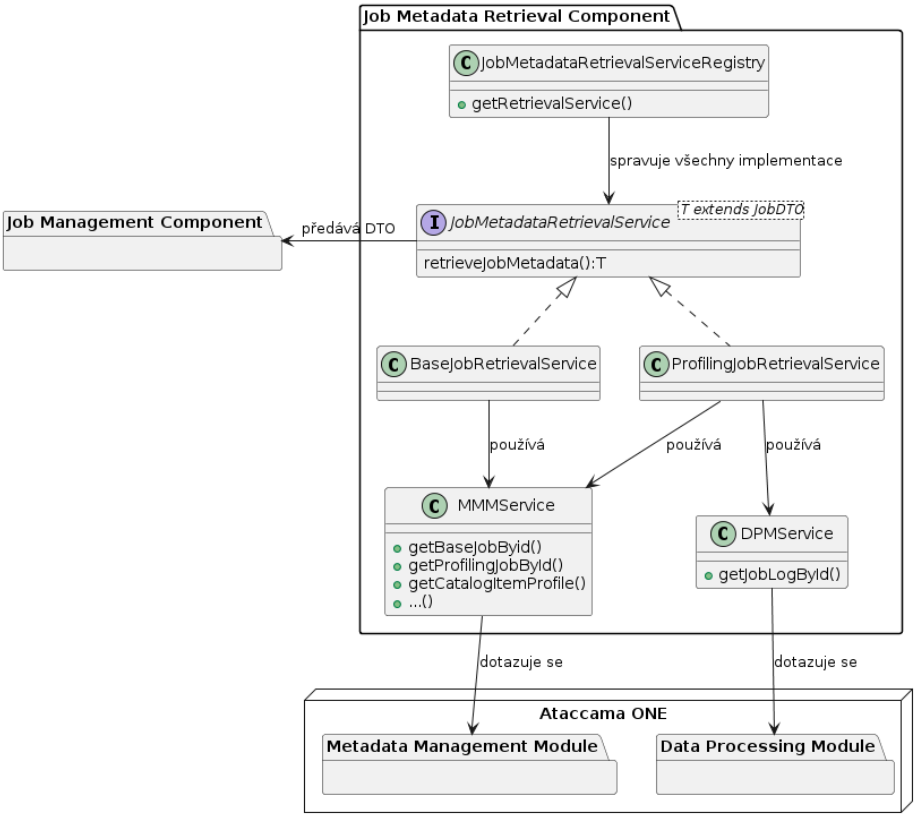
6.2.1 Job Management komponenta



Obrázek 6.2 Diagram hlavních třídy a metod JobManagement komponenty

Job Management komponenta (*Obrázek 6.2*) je centrální komponenta mikroservisy řídící správu příchozích úloh z *MMM* pomocí *GraphQL subskripce*. Pomocí knihovny *RXJava* zmíněné v sekci 4.2.3 asynchronně zpracovává detekované úlohy (splnění požadavku **P-2**) a komunikuje s *JobMetadataRetrieval* komponentou, od které získává metadata jednotlivých úloh, které následně předává komponentě *Updater*. Také notifikuje *JobMetadataRetrieval* komponentu o typu zpracovávané úlohy.

6.2.2 Job Metadata Retrieval komponenta

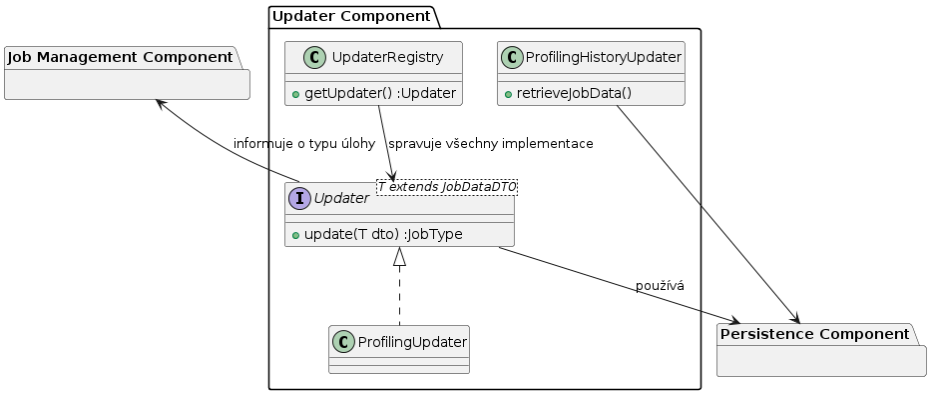


Obrázek 6.3 Diagram hlavních tříd JobMetadataRetrieval komponenty

Job Metadata Retrieval je komponenta sloužící pro získávání metadat detekovaných úloh konkrétního typu (*Obrázek 6.3*). Pomocí *ONE API* komunikuje s *MMM* a *DPM* (popsáno v kapitole 5), obdržené odpovědi zpracovává a následně mapuje do objektů pro přenos dat (*DTO*[1]). Tyto objekty pak předává *JobManagement* komponentě, která je dále zpracovává.

[1]Data Transfer Object

6.2.3 Updater komponenta



Obrázek 6.4 Diagram hlavních tříd Updater komponenty

Updater komponenta slouží jako prostředník pro komunikaci mezi *Job Management* a *Persistence* komponentami (*Obrázek 6.4*). Od komponenty *Job Management* je jí předán *DTO* a ona rozhodne, jakým způsobem se bude zpracovávat a dále předávat komponentě Persistence. Třída `ProfilingHistoryUpdater` je zodpovědná za import metadat úloh spuštěných během jejího výpadku a splňuje tak požadavek **P-3**.

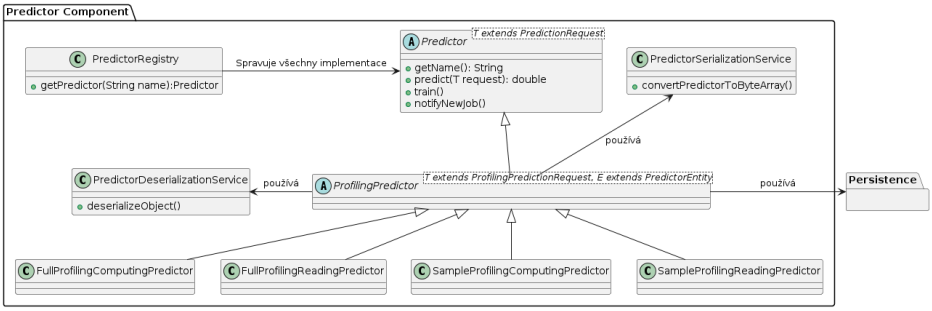
6.2.4 Persistence komponenta

Persistence komponenta (*Obrázek 6.5*) je komponenta sloužící pro komunikaci s databází. Její úlohou je uchovávat metadata úloh a serializované predikční modely. Pro přímý přístup k databázi využívá rozhraní zvaná *CrudRepositories*. Tato specializovaná rozhraní poskytované *Spring Boot frameworkem* umožňují provádět základní operace pro vytváření, čtení, aktualizaci a mazání dat (CRUD[2]) bez nutnosti psát detailní databázové dotazy pomocí jazyka *SQL*. Jednotlivé tabulky jsou definovány jako entity používající objektově relační mapování skrze technologii *Hibernate ORM [31]*. Tento přístup umožňuje abstrakci databázových operací a integraci datových modelů přímo do kódu v jazyce *Java*.

Využití generik v jazyce *Java* v kontextu *CrudRepositories* a celých databázových servis zvyšuje flexibilitu a rozšiřitelnost systému a umožňuje jednoduché přidávání podpory pro další typy úloh a jejich serializované predikční modely.

Komponenta pomocí `@Scheduled`[52] plánované metody maže v konfigurovatelném intervalu metadata příliš starých úloh (splnění požadavku **P-9**).

[2]Create, Read, Update, Delete

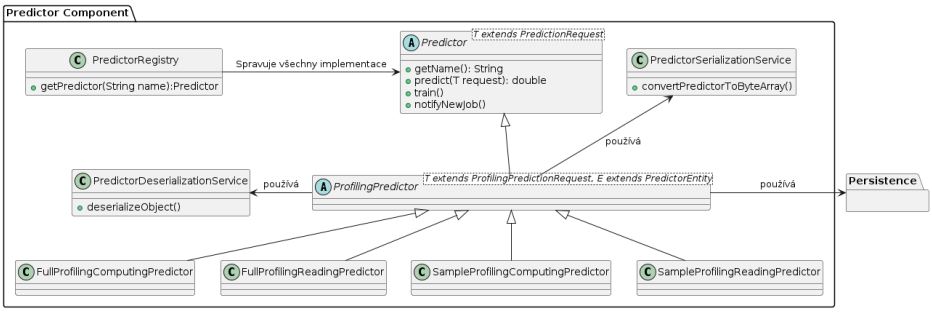


Obrázek 6.5 Diagram hlavních tříd Persistence komponenty

Databáze

Mikroservisa používá databázový systém *PostgreSQL* a definuje tabulky pro každý typ podporované úlohy a predikčního modelu. Systém *PostgreSQL* byl zvolen, protože *ONE* v produkci využívá databázový systém *Amazon Aurora PostgreSQL* [53], jehož vlastnosti jsou s *PostgreSQL* téměř shodné a syntaxe jazyka *SQL* stejná, což nám zaručuje kompatibilitu při nasazení mikroservisy do produkce.

6.2.5 Predictor komponenta



Obrázek 6.6 Diagram hlavních tříd Predictor komponenty

Predictor komponenta (*Obrázek 6.6*) slouží pro správu predikčních modelů. Jejím úkolem je modely inicializovat a při detekci nově příchozích úloh znovutrénovat (splnění požadavku **P-5**). Také komunikuje s Persistence komponentou, která je zodpovědná za ukládádání jejich serializovaných podob. Zároveň také komponenta komunikuje s EstimateAPI, které vrací odhad délky trvání požadované úlohy. Komponenta umožňuje jednoduché přidávání prediktorů pro další typy úloh, případně změnu stávajících prediktorů. Implementace není závislá na knihovně *Tribuo*, tudíž i případná změna knihovny je jednoduchá. Komponenta

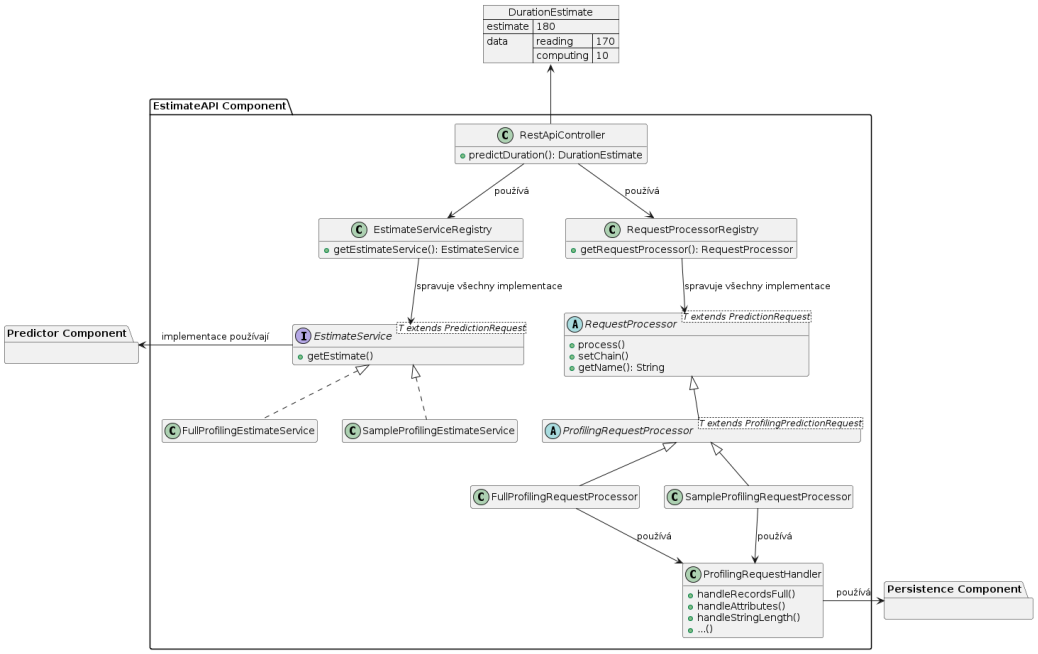
je notifikována JobManagement komponentou o typu nově zpracované úlohy, na jehož základě se příslušné třídy `Predictor` nastaví jako neaktuální a jsou při následující trénovací iteraci znovunatrénovány. Trénování je spouštěno automaticky pomocí `@Scheduled` metody v pravidelném intervalu konfigurovatelném v souboru `application.properties`. Neaktuální modely se trénují jeden po druhém, což proces činí šetrnější na paměť.

SQLDataSource

Během implementace jsme narazili na problém s používáním *Tribuo* třídy *SQLDataSource*[54]. Při opakovaném čtení metadat uložených z databáze docházelo při použití konceptu *try-with resources* i explicitním voláním metody `close()` k neuzavírání databázového spojení a pro každé další čtení bylo vytvořeno spojení nové. To po několika zpracovaných úlohách vedlo k vyhození výjimky signalizující překročení maximálního počtu možných databázových spojení. Při zkoumání kódu knihovny jsme došli k závěru, že třída *SQLDataSource* nevyužívá *recyklaci spojení*[3], ale vytváří separátní *connection pool* pro každé další čtení. Metoda `close()`[55] pak neuzavírá konkrétní spojení, ale pouze instance třídy `java.sql.Statement`. O navazování spojení se v rámci třídy *SQLDataSource* stará třída *SQLDBconfig*[56], konkrétně její metoda `getConnection`. Z tohoto důvodu jsme se rozhodli problém vyřešit vytvořením vlastní implementace této třídy nazvané *OverrideSQLDBConfig*, jež je přímým potomkem třídy *SQLDBConfig* a spravuje jediné spojení, které otevírá a zavírá za použití návrhového vzoru *odložená inicializace*[4].

[3] connection pooling
[4] Lazy Initialization

6.2.6 EstimateAPI komponenta



Obrázek 6.7 Diagram hlavních tříd EstimateApi komponenty

EstimateApi je komponenta poskytující REST API (splnění požadavku **P-4**) pro komunikaci s uživatelem (*Obrázek 6.7*). REST API poskytuje jediný endpoint `POST /predict`. V těle tohoto HTTP požadavku se předávají parametry pro predikční model.

Požadavek `POST /estimate` očekává tělo s parametry popsanými v *Tabulce 6.1.* Kontroler využívá knihovnu *Jackson*[57], která umožňuje automatickou serializaci a deserializaci *POJO*[5] do formátu `JSON` a obráceně. V případě těla požadavku je obsah automaticky deserializován do objektu třídy příslušného požadavku (konkrétní potomek třídy `PredictionRequest`) s jednoduchou možností rozšíření o požadavky pro další typy úloh. V případě úloh datového profilování jsou povinnými parametry typ profilování a identifikátor tabulky. Nespecifikované parametry jsou v rámci zpracování požadavku dopočítány z metadat předchozích dokončených úloh datového profilování na stejné tabulce. Zpracování požadavku využívá návrhový vzor *Chain of Responsibility* a jsou za něj zodpovědní konkrétní potomci třídy `RequestProcessor`.

[5]Plain old Java object

Parametr	Typ	Povinný	Popis
type	enum [fullProfiling, sampleProfiling]	ANO	Typ úlohy k predikci.
catalogItemId	UUID	ANO	Identifikátor tabulky.
records	Long	NE	Počet záznamů tabulky.
attributes	Integer	NE	Počet sloupců.
distinctTotal	Long	NE	Počet odlišných hodnot.
stringTotalLength	Double	NE	Celková délka řetězcové reprezentace hodnot.

Tabulka 6.1 Parametry těla HTTP požadavku

```
{
  "estimate": 2024 ,
  "data": {
    "readingTime": 1212,
    "computingTime": 812
  }
}
```

Výpis kódu 6.1 *Formát odpovědi ve formátu JSON*

Formát odpovědi

Formát odpovědi ve formátu `JSON` definuje třída `DurationEstimate` využívající návrhový vzor *Composite* umožňující rozdělit délku trvání úlohy na více podúloh a na základě času jednotlivých podúloh spočítat délku trvání úlohy celé. V případě úloh datového profilování je úloha rozdělena na *fázi čtení* a *fázi výpočtu* a odpovědí je `JSON` obsahující celkový odhad délky trvání úlohy vypočtený jako součet odhadovaných délek trvání *fáze čttení* a *fáze výpočtu* (*Výpis kódu 6.1*)

6.3 Testování

Mikroservisa je pokryta sadou automatizovaných testů, které zajišťují komplexní ověření funkčnosti jednotlivých komponent. Vývojové prostředí využívá

speciální *Spring* profil nazvaný „test“. Tento profil je klíčový pro spouštění integračních testů, během kterých se aplikační databázový systém *PostgreSQL* nahrazuje vestavěnou databází *H2*[58]. Databáze *H2* je běžně používanou technologií v testování *Java* aplikací, protože se jedná databázi v paměti, která nevyžaduje externí instalaci nebo konfiguraci, je snadno integrovatelná do testovacího kódu a díky jejím vlastnostem umožňuje výrazně testy zrychlit, protože eliminuje latenci spojenou s komunikací na databázový server. Testy jsou automatizovaně spouštěny prostřednictvím `docker compose up` při každém sestavení aplikace, která není spuštěna pokud některý z testů selže. Testy lze také spoušťet odděleně, například z příkazové řádky pomocí příkazu `./gradlew test`. Podrobnější informace se nachází v přiloženém `README.md`

6.3.1 Unit testy

Unit testy se zaměřují na ověření funkcionalit jednotlivých komponent mikroservisy bez nutnosti spouštět *Spring* aplikační kontext. Vyžívají frameworky jako *JUnit*[59] a *Mockito*[60] pro simulaci závislostí a interakcí mezi třídami.

6.3.2 Integrační testy

Integrační testy ověřují funkčnost systému jako celku a jsou klíčové pro zajištění, že všechny komponenty aplikace spolupracují podle očekávání. Tento typ testování je realizován s využitím *Spring* aplikačního kontextu, který spravuje vytvořené Java Beans. K zavedení testovacího prostředí a jeho konfigurace využíváme speciální *Spring* anotace sloužící pro testování konkrétních funkcionalit aplikace jako například `@SpringBootTest`, `DataJpaTest`, `@MockMvcTest` a zároveň využíváme mockování *Java Beans* pomocí anotace `@MockBean`.

Kapitola 7

Měření přesnosti modelů a výsledky

Na základě analýzy ze sekce 3.2 jsme se rozhodli otestovat tři regresní modely strojového učení z knihovny *Tribuo*. Pro evaluaci modelů jsme použili data z měření popsaném v sekci 3.2. Tato data obsahují metadata zhruba devíti set spuštěných úloh. Pro rigoróznější evaluaci a výběr modelů by zde bylo vhodnější použít metadata více úloh a ještě rozšířit rozmanitost profilovaných tabulek, příprava dat a jejich profilování by však vyžadovalo další netriviální časové a zdrojové nároky, a proto jsme se rozhodli, že pro přípravu prototypu modelu postačí data z původních měření.

LinearSGDTrainer[61]

CARTRegressionTrainer[62]

XGBoostRegressionTrainer[63]

7.1 Statistické metriky

Při vyhodnocování predikčních modelů v knihovně *Tribuo* máme k dispozici tři statistické metriky: *Střední absolutní chyba (MAE)*, *Odmocnina střední kvadratické chyby (RMSE)* a *Koeficient Determinace (R^2)*.

7.1.1 Odmocnina střední kvadratické chyby

RMSE popisuje velikost chyb mezi predikcemi regresního modelu a hodnotami pozorovanými v datech. Jedná se v zásadě o standardní odchylku chyb predikcí

modelu na daném datasetu. Je definována jako

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

kde:

- n je počet pozorování v datové sadě
- y_i je skutečná hodnota i-tého datového údaje
- $\hat{y}_i$ je modelem predikovaná hodnota i-tého datového údaje

Čím nižší hodnota *RMSE* je, tím lepší a přesnější model je, v případě dokonalého modelu pak $RMSE = 0$. *RMSE* díky svému kvadratickému charakteru (můžeme si všimnout, že ve vzorci umocňujeme rozdíl skutečné a predikované hodnoty) klade vyšší váhu větším chybám, což je užitečné v případě, kdy i nečasté velké nepřesnosti mohou vést k zásadnějším problémům než častější menší nepřesnosti.

7.1.2 Střední absolutní chyba

MAE měří průměrnou velikost chyb predikcí bez ohledu na jejich směr. Je definována jako:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

kde n, y_i, a $\hat{y}_i$ mají stejný význam jako v RMSE. *MAE* narozdíl od *RMSE* zachovává jednotku a nízká hodnota *MAE* v kontextu modelu naznačuje, že chyby mezi odhadovanými a skutečnými hodnotami jsou průměrně malé, což svědčí o dobré přesnosti predikcí.

V případě predikcí délky trvání úloh datového profilování v sekundách pak tato hodnota říká, o kolik sekund se v průměru predikce liší oproti skutečné hodnotě.

7.1.3 Koeficient Determinace

R^2, někdy označovaný jako koeficient úplnosti, je statistická míra, která určuje procento variability v závislé proměnné, která může být vysvětlena nezávislými proměnnými modelu. R^2 je definován jako:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

kde:

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ je součet čtverců chyb *tzv. reziduí* mezi predikovanými a skutečnými hodnotami.
- $\sum_{i=1}^{n}(y_i - \bar{y})^2$ je celkový součet čtverců rozdílů mezi pozorovanými datovými údaji a pozorovaným průměrem
- $\bar{y}$ je průměrná hodnota skutečných hodnot.
- n je počet pozorování v datové sadě.

Hodnota blízká 1 značí, že model dobře vysvětluje variabilitu odpovědi dat. V kontextu predikce *úloh datového profilování*. Vysoká hodnota R^2 blížící se k 1 značí, že model přesně predikuje dobu trvání úloh, což umožňuje spolehlivé plánování a efektivní využití zdrojů.

7.2 Výběr modelu

Vzhledem zaměření řešení na schopnost chytrého plánování úloh se při výběru konkrétního modelu budeme rozhodovat primárně podle hodnoty metriky *MAE*. V požadavku **P-8** zmiňujeme, že nečasté výraznější odchylky jsou možné, což činí *MAE* přínosnější, protože na rozdíl od *RMSE* nepřikládá velkou váhu výraznějším chybám a poskytuje nám přímou hodnotu průměrné chyby ve stejných jednotkách jako je měřená veličina. Pro porovnání výkonnosti testovaných modelů použijeme *k-fold křížovou validací* pomocí *Tribuo* třídy `CrossValidation`. Jako k zvolíme 10. Na naměřených datech otestujeme vybrané modely pro různé hodnoty základních hyperparametrů a na základě výsledků vybereme model nejvhodnější.

7.2.1 Fáze výpočtu

Pro fázi výpočtu jsou nezávislými proměnnými modelu počet sloupců a počet odlišných záznamů. Závislou proměnnou je pak délka trvání této fáze v sekundách. Z tabulek 7.1, 7.2 a 7.3 vidíme, že modely *CARTRegressionTrainer XGBoostRegressionTrainer* se chovají velmi podobně, *XGBoostRegressionTrainer* při použití osmnácti stromů je ovšem lehce výkonější. *MAE* o hodnotě 110 ukazuje, že průměrná chyba predikcí jsou necelé dvě minuty, což je v kontextu řešeného problému hodnota, která na dobré úrovni umožní realizaci chytrého plánování. Hodnota $R^2 = 0,823$ pak říká, že model efektivně vysvětluje variabilitu cílové proměnné a je tedy poměrně spolehlivý. V mikroservise tedy tento model použijeme.

LearningRate[64]	Průměrná hodnota *MAE*	Průměrná hodnota *RMSE*	Průměrná hodnota R^2
0,01	316,198	1063,850	-0,053
0,05	1737,477	5958,560	-92,388
0,1	256.639	5958.5697	0,241

Tabulka 7.1 Výkonnost LinearSGDTrainer pro fázi výpočtu s použitím AdaGrad optimiser

Maximální hloubka stromu	Průměrná hodnota *MAE*	Průměrná hodnota *RMSE*	Průměrná hodnota R^2
5	116,543	454,500	0,819
10	113,136	454,164	0,819
20	113,136	454,164	0,819

Tabulka 7.2 Výkonnost CARTRegressionTrainer pro fázi výpočtu

Počet stromů	Průměrná hodnota *MAE*	Průměrná hodnota *RMSE*	Průměrná hodnota R^2
5	145,084	519,676	0,080
10	112,813	457,013	0,823
18	110,886	451,0509	0,823
20	110,959	451,21	0,822
30	111,111	451,565	0,821
50	111,1994	451,908	0,8208

Tabulka 7.3 Výkonnost XGBoostRegressionTrainer pro fázi výpočtu

7.2.2 Fáze čtení

Pro fázi čtení jsou nezávislými proměnnými modelu počet záznamů, počet sloupců a průměrná délka hodnot sloupců v řetězcové reprezentaci. Závislou

proměnnou je pak délka trvání této fáze v sekundách. Z tabulek 7.4, 7.5 a 7.6 vidíme, že modely *CARTRegressionTrainer XGBoostRegressionTrainer* se opět chovají velmi podobně, ovšem *XGBoostRegressionTrainer* je při použití 10 stromů opět lehce výkonnější. Hodnoty *MAE* a R^2 jsou sice nižší než při evaluaci *fáze predikce*, což si vysvětlujeme tím, že dataset obsahoval data naměřená při různých vnějších vlivech, které mají na fázi čtení větší vliv než na fázi výpočtu. Nejvýkonnější model by ovšem opět měl být pro realizaci chytrého plánování dostatečný.

LearningRate	**Průměrná hodnota *MAE***	**Průměrná hodnota *RMSE***	**Průměrná hodnota R^2**
0,01	574.514	949.976	0.164
0.05	616.066	998,048	0,018
0.1	625,188	1010,12	-0,007

Tabulka 7.4 Výkonnost LinearSGDTrainer pro fázi čtení s použitím AdaGrad optimiser

Maximální hloubka stromu	**Průměrná hodnota *MAE***	**Průměrná hodnota *RMSE***	**Průměrná hodnota R^2**
5	282,119	485,824	0,802
10	286,771	496,307	0,793
20	286,771	496,307	0,793

Tabulka 7.5 Výkonnost CARTRegressionTrainer pro fázi čtení

Počet stromů	Průměrná hodnota *MAE*	Průměrná hodnota *RMSE*	Průměrná hodnota R^2
5	281,708	538,479	0,760
10	277,235	488,772	0,799
15	283,195	492,013	0,796
20	285,461	494,103	0,795
30	286,611	495,718	0,794
50	287,046	496,275	0,793

Tabulka 7.6 Výkonnost XGBoostRegressionTrainer pro fázi čtení

7.3 Modelový příklad v praxi

Abychom ověřili praktické využití mikroservisy v praxi, rozhodli jsme se ve zmenšeném měřítku provést experiment odpovídající modelovému příkladu využití predikcí ze sekce 2.2.1. Nejprve jsme spustili mikroservisu a následně čtyřicet *úloh datového profilování* na celých náhodných tabulkách s realistickými daty. Během jejich postupného dokončování se znovutrénovaly oba predikční modely pro *fázi čtení* a *fázi výpočtu*. Poté jsme spustili profilování vzorku dvanácti dosud nenaprofilovaných tabulek a přes REST API jsme získali predikované délky trvání *úloh datového profilování* na celých tabulkách (*Výpis kódu 7.1*). Následně jsme spustili *úlohy datového profilování* na celých tabulkách a porovnali skutečné délky trvání s odhadovanými (*Tabulka 7.7*). Přestože byl původní model natrénován na relativně malém množství dat, ukázalo se, že volba vstupních proměnných pro oba modely je dobrá, protože přesnost predikcí je řádově přesná a zcela jistě by nám umožnila realizovat chytré plánování. Výraznější odchylky v predikci délky trvání *fáze výpočtu* u Tabulek 3, 4 a 5 přisuzujeme nízkému počtu trénovacích dat. Vzhledem k tomu, že při integraci mikroservisy do platformy se budou oba zmíněné modely pro každého zákazníka trénovat od začátku, experiment prověřuje efektivnost predikcí hned ze startu používání a můžeme říct, že úspěšně ověřil, že již po několika málo zaregistrovaných a zpracovaných úlohách lze délky trvání budoucích úloh obstojně predikovat.

```
curl -X POST -H "Content-Type: application/json" \
    -d '{"type":"fullProfiling","catalogItemId":"<
        IDENTIFIKÁTOR_TABULKY>}'\
     http://localhost:8080/predict
```

Výpis kódu 7.1 *Dotaz na délku trvání úlohy*

Tabulka	**Odhadovaná délka trvání fáze čtení [s]**	**Skutečná délka trvání fáze čtení [s]**	**Odhadovaná délka trvání fáze výpočtu [s]**	**Skutečná délka trvání fáze výpočtu [s]**
Tabulka1	484	545	4369	4534
Tabulka2	31	57	213	365
Tabulka3	1291	1075	7848	5981
Tabulka4	151	140	860	250
Tabulka5	388	312	192	864
Tabulka6	7	1	5	5
Tabulka7	6	2	5	0
Tabulka8	8	1	3	6
Tabulka9	21	14	5	2
Tabulka10	8	6	45	1
Tabulka11	484	549	4369	4567
Tabulka12	965	757	951	981

Tabulka 7.7 Modelový příklad v praxi

Závěr

V této bakalářské práci jsme se věnovali vývoji a implementaci predikčního modelu, jehož cílem je odhadovat délky trvání úloh datového profilování v platformě Ataccama ONE s primárním zaměřením na využití těchto predikcí pro možnost chytrého plánování. Během práce jsme prokázali, že určité vlastnosti dat tabulek ovlivňují délku trvání těchto úloh. Tyto vlastnosti jsme pomocí měření a analýzy výsledků většího množství úloh identifikovali a využili k sestrojení funkčního modelu strojového učení, jehož přesnost a využitelnost pro realizaci chytrého plánování jsme otestovali na reálných datech. Výsledkem práce je nová mikroservisa integrovatelná do platformy Ataccama ONE splňující všechny předem stanovené požadavky. Implementované řešení podporuje detekci a zpracování nově spuštěných úloh v reálném čase a připravuje půdu pro jednoduchou budoucí integraci do platformy. Důraz byl kladen především na kvalitu implementace a snadnou rozšiřitelnost funkcionality řešení pro predikce dalších typů úloh. Výslednou přesnost vytvořeného prototypu predikčního modelu můžeme považovat za obecně spolehlivou a použitelnou pro chytré plánování úloh a naše řešení pokládáme za dobrý základ pro další vývoj v rámci platformy.

Seznam použité literatury

[1] *Big data: The next frontier for innovation, competition, and productivity | McKinsey*. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/big-data-the-next-frontier-for-innovation. [navštíveno 22. 4. 2024].

[2] *Ataccama ONE Platform Overview | Ataccama*. URL: https://www.ataccama.com/platform. [navštíveno 8. 4. 2024].

[3] *Definition of Data Management (DM) - Gartner Information Technology Glossary*. URL: https://www.gartner.com/en/information-technology/glossary/dmi-data-management-and-integration. [navštíveno 22. 4. 2024].

[4] *The age of analytics: Competing in a data-driven world | McKinsey*. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-age-of-analytics-competing-in-a-data-driven-world. [navštíveno 22. 4. 2024].

[5] *Detection and DQ Evaluation Rules :: Ataccama ONE*. URL: https://docs.ataccama.com/one/latest/data-quality/rules.html#more-about-detection-rules. [navštíveno 22. 4. 2024].

[6] *AI-Powered Data Management | Ataccama*. URL: https://www.ataccama.com. [navštíveno 8. 4. 2024].

[7] *Definition of Data Governance - IT Glossary | Gartner*. URL: https://www.gartner.com/en/information-technology/glossary/data-governance. [navštíveno 22. 4. 2024].

[8] *Definition of Master Data Management (MDM) - IT Glossary | Gartner*. URL: https://www.gartner.com/en/information-technology/glossary/master-data-management-mdm. [navštíveno 22. 4. 2024].

[9] *Definition of Data Quality Tools - Gartner Information Technology Glossary*. URL: https://www.gartner.com/en/information-technology/glossary/data-quality-tools. [navštíveno 22. 4. 2024].

[10] *Get Started with Catalog and Glossary :: Ataccama ONE*. URL: https://docs.ataccama.com/one/latest/getting-started/get-started-with-catalog-and-glossary.html. [navštíveno 8. 4. 2024].

[11] *Get Started with Catalog and Glossary :: Ataccama ONE*. URL: https://docs.ataccama.com/one/latest/getting-started/_images/get-started-with-catalog-and-glossary-customers.png. [navštíveno 8. 4. 2024].

[12] *Get Started with Catalog and Glossary :: Ataccama ONE*. URL: https://docs.ataccama.com/one/latest/getting-started/_images/get-started-with-catalog-and-glossary-catalog-items.png. [navštíveno 8. 4. 2024].

[13] *PostgreSQL: The world's most advanced open source database*. URL: PostgreSQL:%20The%20world's%20most%20advanced%20open%20source%20database. [navštíveno 8. 4. 2024].

[14] *TABLESAMPLE Implementation - PostgreSQL wiki*. URL: https://wiki.postgresql.org/wiki/TABLESAMPLE_Implementation. [navštíveno 22. 4. 2024].

[15] *Metadata Model Overview :: Ataccama ONE*. URL: https://docs.ataccama.com/one/latest/metadata-model/metadata-model-overview.html. [navštíveno 8. 4. 2024].

[16] *GraphQL | A query language for your API*. URL: https://graphql.org/. [navštíveno 7. 5. 2024].

[17] *PostgreSQL: Documentation: 16: Chapter 43. PL/pgSQL — SQL Procedural Language*. URL: https://www.postgresql.org/docs/current/plpgsql.html. [navštíveno 29. 4. 2024].

[18] *PostgreSQL: Documentation: 16: 9.25. Set Returning Functions*. URL: https://www.postgresql.org/docs/current/functions-srf.html#FUNCTIONS-SRF. [navštíveno 29. 4. 2024].

[19] *PostgreSQL: Documentation: 16: F.28. pgcrypto — cryptographic functions*. URL: https://www.postgresql.org/docs/16/pgcrypto.html#PGCRYPTO-PASSWORD-HASHING-FUNCS. [navštíveno 29. 4. 2024].

[20] *MinIO | S3 & Kubernetes Native Object Storage for AI*. URL: https://min.io/. [navštíveno 22. 4. 2024].

[21] *Python Quickstart Guide — MinIO Object Storage for Linux*. URL: https://min.io/docs/minio/linux/developers/python/minio-py.html. [navštíveno 8. 5. 2024].

[22] *Java | Oracle*. URL: https://www.java.com/en/. [navštíveno 8. 5. 2024].

[23] *Kotlin Programming Language*. URL: https://kotlinlang.org/. [navštíveno 8. 5. 2024].

[24] *Gradle Build Tool*. URL: https://gradle.org/. [navštíveno 21. 4. 2024].

[25] *The Apache Groovy programming language*. URL: https://groovy-lang.org/. [navštíveno 8. 5. 2024].

[26] *Maven – Welcome to Apache Maven*. URL: https://maven.apache.org/. [navštíveno 21. 4. 2024].

[27] *Gradle | Gradle vs Maven Comparison*. URL: https://gradle.org/maven-vs-gradle/. [navštíveno 21. 4. 2024].

[28] *Spring Framework*. URL: https://spring.io/projects/spring-framework. [navštíveno 20. 4. 2024].

[29] *Spring Boot*. URL: https://spring.io/projects/spring-boot. [navštíveno 8. 4. 2024].

[30] Apache Tomcat Project. *Apache Tomcat® - Welcome!* URL: https://tomcat.apache.org/. [navštíveno 8. 4. 2024].

[31] *Your relational data. Objectively. - Hibernate ORM*. URL: https://hibernate.org/orm/. [navštíveno 8. 4. 2024].

[32] *Docker: Accelerated Container Application Development*. URL: https://www.docker.com/#build. [navštíveno 28. 4. 2024].

[33] *Dependency Injection :: Spring Framework*. URL: https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html. [navštíveno 20. 4. 2024].

[34] The Project Lombok Authors. *Project Lombok*. URL: https://projectlombok.org. [navštíveno 8. 4. 2024].

[35] *Hello from GraphQL Java | GraphQL Java*. URL: https://www.graphql-java.com/documentation/getting-started. [navštíveno 8. 4. 2024].

[36] *Introduction to Apollo Kotlin - Apollo GraphQL Docs*. URL: https://www.apollographql.com/docs/kotlin. [navštíveno 8. 4. 2024].

[37] *Data builders (experimental) - Apollo GraphQL Docs*. URL: https://www.apollographql.com/docs/kotlin/testing/data-builders/. [navštíveno 21. 4. 2024].

[38] *RxJava support - Apollo GraphQL Docs*. URL: https://www.apollographql.com/docs/kotlin/advanced/rxjava. [navštíveno 27. 4. 2024].

[39] *Weka Wiki*. URL: https://waikato.github.io/weka-wiki/. [navštíveno 28. 4. 2024].

[40] *The GNU General Public License v3.0 - GNU Project - Free Software Foundation.* URL: https://www.gnu.org/licenses/gpl-3.0.html. [navštíveno 8. 5. 2024].

[41] *Commercial applications - Weka Wiki.* URL: https://waikato.github.io/weka-wiki/faqs/commercial_applications/. [navštíveno 28. 4. 2024].

[42] *Machine Learning in Java - Tribuo: Machine Learning in Java.* URL: https://tribuo.org/. [navštíveno 8. 4. 2024].

[43] *Oracle Labs | Home.* URL: https://labs.oracle.com/pls/apex/r/labs/labs/intro. [navštíveno 8. 4. 2024].

[44] *Apache License, Version 2.0.* URL: https://www.apache.org/licenses/LICENSE-2.0. [navštíveno 8. 5. 2024].

[45] *Docker Compose overview | Docker Docs.* URL: https://docs.docker.com/compose/. [navštíveno 29. 4. 2024].

[46] *GraphQL.* URL: https://spec.graphql.org/October2021/%5C#sec-Query. [navštíveno 21. 4. 2024].

[47] *GraphQL.* URL: https://spec.graphql.org/October2021/%5C#sec-Mutation. [navštíveno 21. 4. 2024].

[48] *GraphQL.* URL: https://spec.graphql.org/October2021/%5C#sec-Subscription. [navštíveno 21. 4. 2024].

[49] *Git.* URL: https://git-scm.com/. [navštíveno 8. 5. 2024].

[50] *The most-comprehensive AI-powered DevSecOps platform | GitLab.* URL: https://about.gitlab.com/. [navštíveno 8. 5. 2024].

[51] *Welcome To UML Web Site!* URL: https://www.uml.org/. [navštíveno 8. 5. 2024].

[52] *Scheduled (Spring Framework 6.1.6 API).* URL: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html. [navštíveno 8. 5. 2024].

[53] *Working with Amazon Aurora PostgreSQL - Amazon Aurora.* URL: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.AuroraPostgreSQL.html. [navštíveno 27. 4. 2024].

[54] *SQLDataSource (Tribuo 4.3.1 API).* URL: https://tribuo.org/learn/4.3/javadoc/org/tribuo/data/sql/SQLDataSource.html. [navštíveno 27. 4. 2024].

[55] *tribuo/Data/src/main/java/org/tribuo/data/sql/SQLDataSource.java*. URL: https://github.com/oracle/tribuo/blob/3ba47e6e4f45e488d4eccd8d3884a9b4c9a9b918/Data/src/main/java/org/tribuo/data/sql/SQLDataSource.java#L105-L115. [navštíveno 27. 4. 2024].

[56] *SQLDBConfig (Tribuo 4.3.1 API)*. URL: https://tribuo.org/learn/4.3/javadoc/org/tribuo/data/sql/SQLDBConfig.html. [navštíveno 27. 4. 2024].

[57] *FasterXML/jackson: Main Portal page for the Jackson project*. URL: https://github.com/FasterXML/jackson. [navštíveno 8. 5. 2024].

[58] *H2 Database Engine*. URL: https://www.h2database.com/html/main.html. [navštíveno 7. 5. 2024].

[59] *JUnit 5*. URL: https://junit.org/junit5/. [navštíveno 8. 5. 2024].

[60] *Tasty mocking framework for unit tests in Java*. URL: https://site.mockito.org/. [navštíveno 8. 5. 2024].

[61] *LinearSGDTrainer (Tribuo 4.3.1 API)*. URL: https://tribuo.org/learn/4.3/javadoc/org/tribuo/regression/sgd/linear/LinearSGDTrainer.html. [navštíveno 29. 4. 2024].

[62] *CARTRegressionTrainer (Tribuo 4.3.1 API)*. URL: https://tribuo.org/learn/4.3/javadoc/org/tribuo/regression/rtree/CARTRegressionTrainer.html. [navštíveno 29. 4. 2024].

[63] *XGBoostRegressionTrainer (Tribuo 4.3.1 API)*. URL: https://tribuo.org/learn/4.3/javadoc/org/tribuo/regression/xgboost/XGBoostRegressionTrainer.html. [navštíveno 29. 4. 2024].

[64] *Learning rate in Regression models | by ahmad mousavi | Medium*. URL: https://medium.com/@sam683/learning-rate-in-regression-models-ff704126a3ae. [navštíveno 8. 5. 2024].

Seznam tabulek

Seznam obrázků

Seznam použitých zkratek

DPE Data Processing Engine 10

DPM Data Processing Module 10

MAE Střední absolutní chyba 50

MMM Metadata Management Module 10

ONE Ataccama ONE 4

R^2 Koeficient Determinace 50

RMSE Odmocnina střední kvadratické chyby 50

Příloha A

Přehled elektronických příloh

- `src` ..... zdrojové kódy mikroservisy.
  - `main`
    - `graphql` ..... soubory GraphQL operací a schémat
    - `java` ..... zdrojové kódy v jazyce Java
      - `com.ataccama.prediction` ..... společný Java balík
        - `constants` ..... konstanty
        - `dto` ..... objekty pro přenos dat
        - `estimateapi` ..... kód EstimateApi komponenty
        - `jobmanagement` ..... kód JobManagement komponenty
        - `persistence` ..... kód Persistence komponenty
        - `predictor` ..... kód Predictor komponenty
        - `apollo` ..... Apollo Kotlin konfigurace
        - `retrieval` ..... kód JobMetadataRetrieval komponenty
        - `updater` ..... kód Updater komponenty
    - `kotlin – zdrojové kódy v jazyce Kotlin`
      - `com.ataccama.prediction` ..... společný Java balík
        - `apollo` ..... Apollo Kotlin konfigurace
    - `resources` ..... konfigurační soubory
  - `tests` ..... zdrojové kódy testů
- `build.gradle`
- `docker-compose.yml`
- `Dockerfile`
- `gradlew`
- `gradlew.bat`
- `settings.gradle`
- `README.md` ..... návod na spuštění a konfiguraci projektu