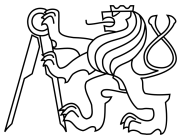
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Analýza bezpečnostních rizik aplikací z logů v reálném čase
Student:	Bc. Vojtěch Krákora
Vedoucí:	Pavel Pivoňka, GWCPM
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Nastudujete problém zjišťování bezpečnostních rizik aplikací spojených s kompromitací dat z logů nakonfigurovaných pomocí frameworku (například Log4j [2]), zejména pomocí metod strojového učení [4].
Vyberte potřebná data, pomocí vhodných metod pro zpracování dat v reálném čase, tak, aby bylo možno sledovat projevy bezpečnostních rizik pomocí prezentačního nástroje, jakým je např. Google Charts [3].
Analyzujte data pomocí asociačních pravidel a implementujte framework v jazyce Java, který umožní využití extrahovaných znalostí v systémech třetích stran [5] podporujících tzv. SIEM (např. LogRhythm, ArcSight, QRadar atd.).
Demonstrujte funkčnost frameworku za použití provozních (eventuálně referenčních) dat systému CETIN-NIP postaveného na platformě Unify [1].

Seznam odborné literatury

[1] [http://www.physter.com/unify/]
[2] [http://logging.apache.org/log4j/2.x/]
[3] [http://logging.apache.org/log4j/2.x/]
[4] [http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html]
[5] [https://cs.wikipedia.org/wiki/SIEM]

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 20. října 2016

České vysoké učení technické v Praze

Fakulta informačních technologií

Department of Katedra teoretické informatiky

Diplomová práce

Analýza bezpečnostních rizik aplikací z logů v reálném čase

Bc. Vojtěch Krákora

Vedoucí práce: Pavel Pivoňka, GWCPM

9. května 2017

Poděkování

Děkuji svému vedoucímu Pavlovi Pivoňkovi, GWCPM za odborné vedení, pomoc a rady při zpracování této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Prague dne 9. května 2017 .....

Odkaz na tuto práci

Krákora, Vojtěch. *Analýza bezpečnostních rizik aplikací z logů v reálném čase.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Diplomová práce se zaměřuje na detekci bezpečnostních rizik v reálném čase. Využívám nástroj MS Azure Machine Learning, kam jsou odesílána Javou předzpracovaná data. Jsou použity algoritmy pro shlukování a detekci anomálie. Podstatou práce je detekce rizik z logu integrační platformy v reálném čase. Vytvořené řešení tyto údaje ukládá do NoSQL databáze a pomocí Google Charts prezentuje výsledky.

Klíčová slova dolování z textu, bezpečnostní rizika, shlukování, anomálie

Abstract

The diploma thesis focuses on detecting security risks in real time. The preprocessed data are being sent to MS Azure Machine Learning, where are algorithms for clustering and anomaly detection used. The essence of the thesis is the detection of risks from the integration platform log in real time. The created solution stores these data in the NoSQL database and presents the results using Google Charts..

Keywords text mining, security risks, clustering, anomaly

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době rozvoje informačních technologií se lze častěji setkat s problémem řešení bezpečnostních rizik. Jako jeden z vývojářů integrační platformy Unify jsem se rozhodl na tento problém reagovat touto prací.

Unify integruje velké množství systémů z vnitřní sítě zákazníka, ale i napříč internetem. V množství požadavků až 7 za vteřinu se mohou různá rizika snadno ztratit. Dnes se lze setkat s tím, že logy Unify jsou hlídány jednoduchými skripty a aplikací, jenž podle přesně určených vzorců reaguje. Tyto způsoby nemusí zachytit veškerá rizika a často jsou příliš pomalá. Proto jedním z cílů práce je prověřit, jaká bezpečnostní rizika se mohou vyskytovat na integrační platformě.

První kapitola obsahuje teoretické informace. Seznamuje nás s různými bezpečnostními riziky a známými způsoby jejich detekce. Dále přináší informace o integrační platformě Unify, na které jsou prováděny experimenty a testování. Kapitola také ukazuje informace o strojovém učení a jeho metodách. Je zde vysvětleno, proč jsem se rozhodl namísto těžení asociačních pravidel použít shlukování a detekci anomálií.

Druhá kapitola přináší návrhy, které vedou k cílové implementaci. Je zde představena architektura celé aplikace, včetně využití NoSQL databáze. Protože tato aplikace by měla rozšířit již funkční Unify, je zde uveden bezpečný způsob přístupu k provozu platformy. Dále je uvedena informace o tom, jak mohou aplikace třetích stran využívat zpracovaná data.

Třetí kapitola popisuje realizaci. Vysvětluje nutná nastavení pro aplikační server, kolekce v databázi a také vytvoření modelu na MS Azure ML.

Čtvrtá kapitola se zabývá experimenty ve shlukování a detekci anomálií. Hledá ideální nastavení pro specifická data z Unify. Vytěžené znalosti jsou pak použity nad reálnými daty. Z těchto podkapitol následně vyplynuly nedostatky a návrhy do budoucna.

Kapitola 1

Úvod do problematiky

1.1 Kybernetická bezpečnost

Pro dnešní dobu je běžné využívání informačních technologií prakticky kdekoliv. Jde o stavební kámen mnoha podniků. Informačním systémem rozumíme kombinaci softwaru, hardwaru, infrastruktury a trénovaného personálu [1]. Tam, kde se informační systémy vyskytují pomáhají práci zjednodušovat, tvořit, či se na ní jinak podílet. Systémy je třeba udržovat v provozu a co nejvíce se vyhnout všem možným rizikům, které mohou narušit obchodní proces.

Obecně lze na zabezpečení informačního systému nahlížet z mnoha různých úhlů. Zabezpečit lze síť pomocí certifikovaného přístupu, správně zabezpečené bezdrátové sítě. Do operačních systémů se nainstalují antivirové programy. Samotná zařízení se fyzicky uzamknou a znemožní se přístup nepovolaným osobám a podobně.

Povinnost zabezpečovat informační systémy přikládá zákon o kybernetické bezpečnosti 181/2014 Sb. Tento zákon stanovuje povinnosti a práva orgánů veřejné moci v oblasti kybernetické bezpečnosti [2]. Mimo jiné také nařizuje orgánům veřejné moci, povinnost zajišťovat kybernetickou bezpečnost.

Kromě zákonů jsou k dispozici normy. Jednou z takových norem je ISO 27001. Tato norma stanovuje požadavky, které je nutné dodržovat pro kybernetickou bezpečnost. Tyto požadavky jsou stanoveny jak na samotné informační systémy, tak i na zaměstnance, informační procesy, či strategie firmy [3].

Lze říci, že zajištění kybernetické bezpečnosti a následná detekce jednotlivých bezpečnostních rizik je velmi specifická činnost. Každá určitá doména má svá rizika a jiné druhy způsobů jejich detekce.

V práci řeším kybernetickou bezpečnost pro konkrétní část informačního systému. Podstatná tedy jsou bezpečností rizika na integrační platformě Unify. Definice integrační platformy a její popis je v sekci 1.2.

1.1.1 Bezpečnostní rizika

Jak již bylo uvedeno, pro tuto práci jsou podstatná bezpečností rizika na konkrétní integrační platformě. Cílem je hledat pouze ta, která vznikají na straně programu. Zabezpečení přístupů fyzicky k serverům Unify a podobně nebudou probírána.

1.1.1.1 Zero day útoky

Zero day útoky lze volně přeložit jako útoky nultého dne. Jde o taková bezpečnostní rizika, která využívají zranitelností nějakého systému, nebo jeho části, která nebyla zveřejněna [4].

Integrační platforma je složena z mnoha modulů, které tvoří její celek. S rostoucím množstvím softwaru a použitých knihoven jsou šance na skrytou chybu velmi vysoké.

1.1.1.2 Odepření služby

Útoky, jenž se snaží o znemožnění užívání nějaké služby jejími uživateli, označujeme jako DoS. Zkratka DoS pochází z anglického názvu *Denial of service*, přeloženo jako odepření služby. Poddruhem DoS útoku je DDoS (z anglického *distributed denial of service*). Při DDoS je použito velké množství různých zařízení, které požadovanou službu zahltí požadavky. Cílem je, aby služba nedokázala požadavky odbavovat a došlo tím k tomu, aby nové požadavky odmítala [5].

Ukázka rozdílu mezi DoS a DDoS je na obrázku 1.1. V části s DoS je znázorněn útočník, který se snaží znepřístupnit službu. U DDoS útočník využije nakažené PC (obecně to mohou být i mobilní telefony, tablety, či jiná zařízení s přístupem do sítě) a následně jejich pomocí zahlcuje cílovou službu.

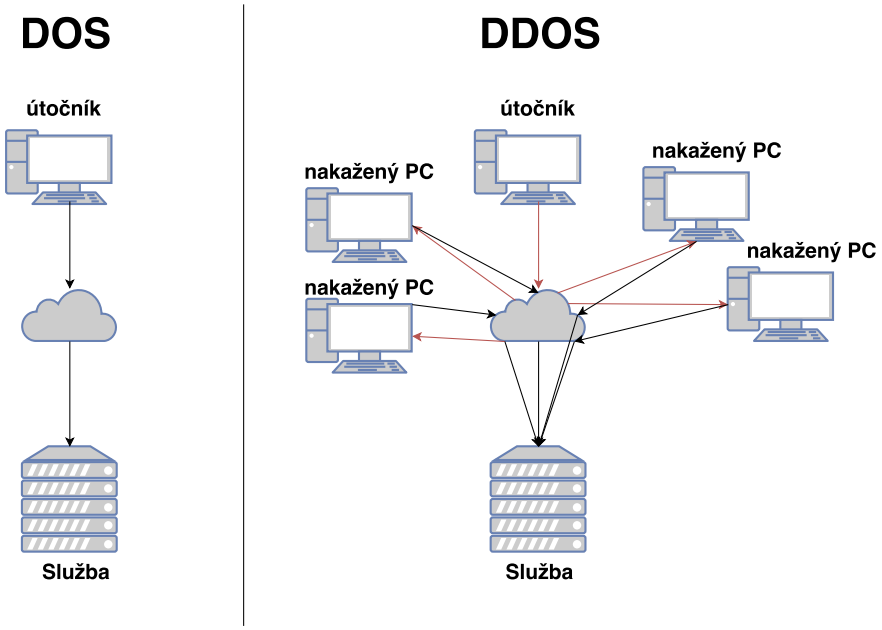
Důsledky útoku se následně dotýkají mnoha uživatelů nedostupně služby. Proto je nutné nejen detekovat velké množství požadavků na vlastní služby, ale také například nedostupnost partnera, na jehož službu jsem připojen. Při včasné detekci je možné zjistit, že je partner pod DDoS útokem dříve, než to zjistí on.

Jeden DDoS útok se odehrál na služby O2 v roce 2016. Jeho důsledky následně omezil i provoz integrační platformy, pro kterou je tato práce tvořena.

1.1.1.3 Nevalidní dotazy

Riziko, které je třeba hlídat, ale není nutně vytvořené za účelem poškodit někoho nebo něco. Integrační platforma musí zpracovávat různé požadavky od konzumentů a ty zpravidla přeposílat poskytovatelům služeb. Mohou nastat situace, kdy konzument posílá nevalidní požadavek.

V případě, že požadavky jsou v XML formě, lze definovat přesné znění zprávy. V případě, že je její validita narušena dochází k nekompatibilnímu



Obrázek 1.1: Grafické porovnání DoS a DDoS útoku

dotazu na rozhraní poskytovatele. To může vyústit v nevyřízené a odmítnuté požadavky. Z této situace mohou vzniknout nechtěné útoky odepření služeb. V případě, že požadavek nebude zamítnut, možným důsledkem může být zisk nevalidních dat nebo dokonce i dat, která se ke konzumentovi neměla dostat.

Tyto popřípadě i jiné neočekávané situace je třeba včas odhalit a řešit.

1.1.1.4 Ostatní

Mnoho bezpečnostních rizik může být neznámých. Nelze vše zaškatulkovat do obecných kategorií. Rizikem mohou být i služby, které po autorizaci nabízí různé možnosti získání dat, například pomocí výběru z databáze. Z těchto důvodů je vhodné nezaměřovat se na konkrétní kategorie rizik, ale snažit se rozpoznávat rizika jakožto celek.

1.1.2 Zjišťování bezpečnostních rizik

Rozpoznávat bezpečnostní rizika je při jejich různorodosti složitý úkol. Základem je dodržovat bezpečnostní pravidla například při manipulaci s hesly. Na platformě, kde se lze setkat se stovkami služeb se bude vyskytovat velké množství hesel. Správa těchto hesel není snadným úkolem. Přesto by se nemělo stát, že i třeba na testovacím prostředí aplikace se budou hesla zasílat e-mailovou komunikací. Hesla budou totožná jako uživatelská jména nebo dokonce zabezpečení nebude existovat.

V následujících kapitolách jsou popsány různé metody, jak na softwarové úrovni detekovat co nejvíce možných bezpečnostních rizik.

1.1.2.1 Posouzení kódu

Posouzení kódu (anglicky code review) je metoda kontroly kvality kódu. Programátor, který napsal nějaký kus komponenty, dá svou část programu na kontrolu druhé osobě. Není přímo nutné, aby kontrolující osoba byla nadřízený. Kontrolující programátor hlídá kvalitu kódu a zároveň prověřuje jeho funkčnost. Tato metoda vede ke zkvalitnění dodávaného softwaru, ale je finančně i časově náročná [6]. Lze ovšem říci, že finanční i časová náročnost jsou diskutabilní. Lze se setkat s tím, že kód, který neprošel posouzením druhou osobou nebude fungovat a charakteristika chyby neumožní její snadnou detekci. Pokud by se tato chyba odhalila pomocí posouzení, mohl být čas naopak ušetřen. Další důvod pro ušetření času jsou nutné byrokratické procesy, které jsou nutné pro opravení chyby (například vytvoření opravné dávky, manuálu a kontaktování release managementu). To už hodně záleží na procesech, které jsou při vývoji používány.

Zvýšením kontroly se i zvyšuje šance na detekci bezpečnostního rizika. Jako velmi efektivní metodu ji uvádí zdroj [7].

1.1.2.2 Analýza z pohledu uživatele

Ve snaze celý proces detekce bezpečnostních rizik zautomatizovat se používá i metoda založená na pohledu uživatele aplikace [7]. Aplikace má své uživatele, proto je možné vytvářet scénáře, kdy se uživatel pokouší najít v aplikaci nějaký nedostatek. Takový nedostatek může následně produkovat bezpečnostní riziko. Automatizace těchto scénářů je zpravidla snadno proveditelná a často patří do klasických testovacích scénářů.

Testují se například SQL injection, přístup do neoprávněných míst a další.

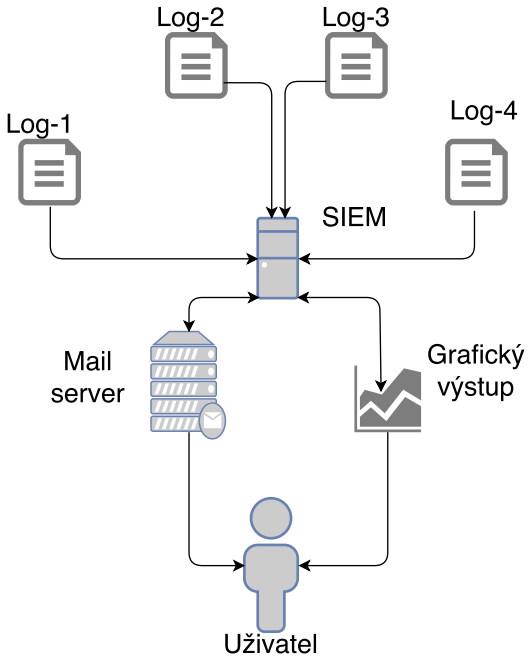
1.1.3 Analýza na straně serveru

Předchozí způsoby se snažili zabránit výskytu rizik. Pro zachování kybernetické bezpečnosti je vhodné sledovat i aktuální situaci v aplikaci. Vzhledem k rozsahu moderních aplikací je takřka vyloučené, že bychom dokázali rizikům předejít. Sledování aktuálního stavu a provozu můžeme být schopni detekovat podezřelé činnosti. Na základě toho, pak dokážeme zjistit bezpečnostní riziko.

I zde lze využit automatizaci. Například ve zdroji [8] je uveden systém na rozpoznání vzorů zpráv. Koncept založený na znalostním inženýrství může pomoci detekovat i využití zranitelnosti nultého dne (1.1.1.1) [9].

Podobný princip využívají i systémy SIEM (security information and event management). SIEM se zabývá bezpečnostním managementem. Jeho myšlenka je taková, že rozsáhlé aplikace mají své zdroje informací (logy) na různých místech, ale je zapotřebí k nim přistupovat z jednoho místa. Z tohoto místa

se provádí následná analýza kompletně všech zdrojů a vyhodnocují se bezpečnostní rizika [10]. Pomocí tohoto vyhodnocení pak může specializovaný personál reagovat na vzniklou situaci. Grafické znázornění příkladu SIEM je na obrázku 1.2.



Obrázek 1.2: Grafické znázornění SIEM

1.2 Integrační platforma Unify

Jedním z cílů práce je detekovat bezpečnostní rizika na integrační platformě Unify [11]. Integrace je proces, který propojuje nesourodé systémy, tak aby umožnil jejich snadnou komunikaci [12].

1.2.1 Význam Unify

Jak je uvedeno ve zdroji [11], Unify pracuje s webovými službami, zprávami, transformací a směrováním dat nebo také přenosem souborů. Její význam koordinace interakcí mezi různými podnikovými systémy.

Rozsáhlé systémy se skládají z odlišných komponent. Komponenty mohou být například databázové, webové nebo souborové. Aby se zajistila komunikace mezi nimi, je vytvářena integrační platforma. Základem Unify je sběrnice ESB (enterprise service bus). Tato sběrnice vystavuje různé interfacy a je schopna volat veškeré nutné komponenty. Jednotlivé komponenty pak nekomunikují mezi sebou napřímo, ale skrz ESB.

Podobný přístup je i pro partnery mimo domovskou organizaci, kde je Unify provozováno. Každý takový partner je připojen na B2B sběrnici (business to business). Ta stejně jako ESB vystavuje interfacy a následně volá rozhraní ESB. Význam B2B je takový, že se zpravidla vyskytuje v takzvané demilitarizované zóně. Z tohoto důvodu je nutné zde dbát na vyšší bezpečnost provozu.

Mimo zmíněných sběrnic je Unify složeno i z dalších komponent. Například notifikační engine slouží k rozesílání mailových nebo po připojení na SMS konektor, i SMS zpráv. Komponenta ETL (extract transform load) zpravidla transformuje data z jednoho uložiště do druhého. SFE (secure file exchange) má za úkol přenos souborů. Tento přenos se hlavně využívá mezi demilitarizovanou zónou a vnitřní sítí.

1.2.2 Technologie

Unify je postaveno na otevřeném softwaru. Hlavní použité technologie jsou:

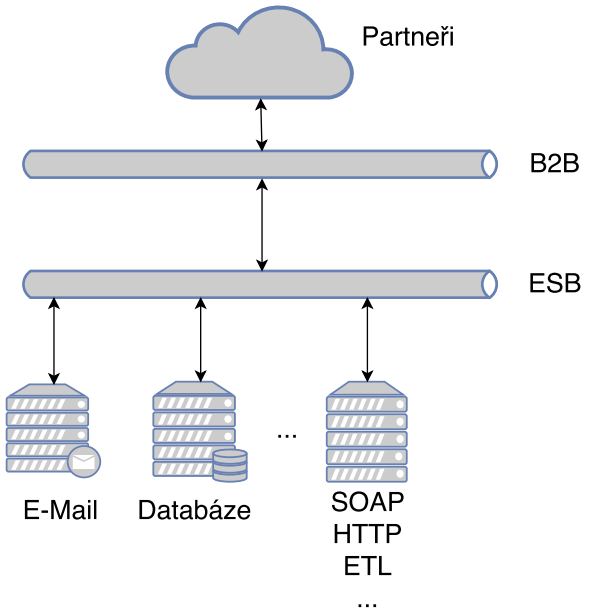
- **JBoss** - aplikační server s podporou Java EE 6 [13]
- **Switchyard** - vývojářský framework, který pomocí Apache Camel umožňuje snazší vývoj aplikací postupy architektury orientované na služby
- **Apache Camel** - otevřený software jakožto aplikační rámec pro integraci
- **Databáze** - pro účel přeposílání zpráv a journalování zpráv se využívají databáze
- **Nginx SW Load balancer** - webový servere, který distribuuje zprávy mezi dva, či více totožných serverů za účelem rozložení zátěže

1.2.3 Architektura Unify

Architektura již byla z části popsána v sekci 1.2.1. Skládá se tedy ze dvou sběrnic, kde jedna slouží pro komunikaci vnitřních systémů a druhá pro komunikaci se systémy partnerů. Dále Unify obsahuje různé komponenty pro prací s daty nebo jejich přenos, popřípadě rozesílání emailů a SMS zpráv. Architekturu přibližuje obrázek 1.3.

Mimo sběrnic a komponent Unify obsahuje i uživatelské rozhraní, sloužící pro konfiguraci.

Z použitých technologí v kapitole 1.2.2 plyne, že hlavním stavebním kamenem pro implementaci je Java EE 6.



Obrázek 1.3: Enterprise Service Bus

1.2.4 Komunikace na Unify

Jak bylo zmíněno, na Unify je komunikace jak pomocí XML zpráv, JSON zpráv, tak například i přenosem souborů. Největší provoz je zpravidla na sběrnici ESB. Požadavky na ESB lze rozdělit na tři druhy:

1.2.4.1 Synchronní zprávy

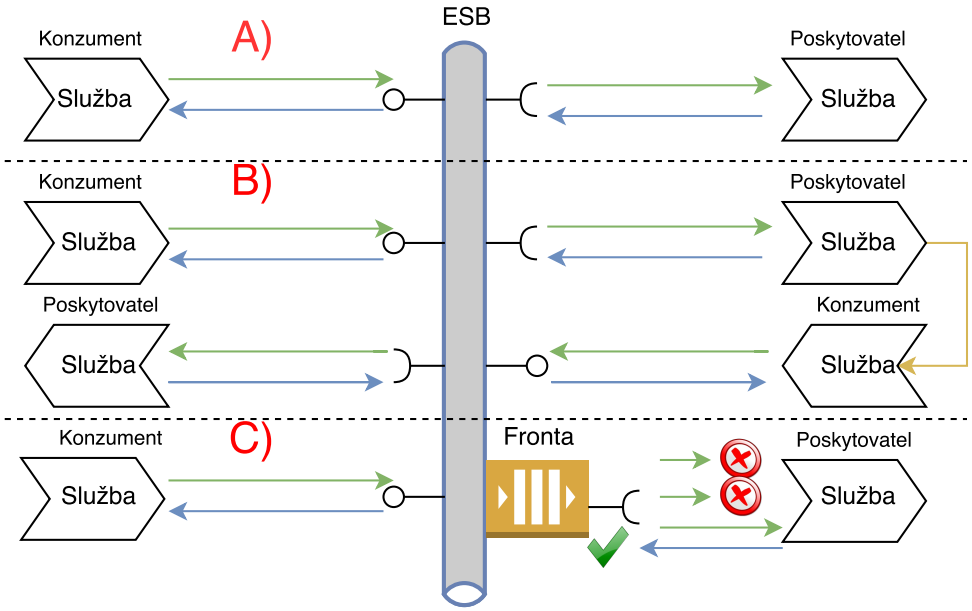
Konzument odešle požadavek, na který okamžitě dostane odpověď o tom, zdali se ho podařilo doručit. Doručení nezajišťuje i zpracování požadavku cílovým systémem. Byť u časově nenáročných požadavků je i to možné. Využití je zpravidla na dotazující se služby. Kdy synchronně na zprávu dostaneme odpověď s informací na kterou jsme se tázali. V obrázku 1.4 část „A)“.

1.2.4.2 Asynchronní zprávy

Asynchronní zprávy se na Unify dají rozdělit na dvě synchronní. První je dotaz konzumenta služby, na který obdrží okamžitou odpověď o přijetí požadavku. Jakmile je požadavek cílovým systémem zpracován, je odeslána zpět konzumentovi synchronní zpráva, která informuje i o zpracování požadavku. Běžné využití je pro takové služby, které požadavek zpracovávají delší dobu. V obrázku 1.4 část „B)“.

1.2.4.3 Zprávy s frontou

Je-li nutné mít jistotu odeslání požadavku (zpravidla například nové objednávky), pošle se dotaz na službu s frontou. Služba si požadavek uloží do fronty a odpoví zpět konzumentovi informací o přijetí. Následně požadavek odešle cílovému systému. Jakmile od systému obdrží kladnou synchronní odpověď o zpracování, vyjme požadavek z fronty a více ho nevolá. V opačném případě zkusí požadavek poslat po nějaké době znova. V obrázku 1.4 část „C)“.



Obrázek 1.4: Běžné druhy komunikace na ESB, Unify - A) synchronní, B) asynchronní, C) zprávy s frontou

1.3 Strojové učení

Strojové učení patří jako podkapitola do znalostního inženýrství. Jde o algoritmy a metody, které pomáhají stroji, aby se učil. Učením je myšleno to, kdy je stroj schopen na základě zkušenosti se rozhodovat. Toto rozhodování dokáže vyvodit automaticky bez pomoci člověka [14].

Strojové učení je využíváno například ve spamových filtrech, v algoritmech rozpoznávajících obličej nebo jako automatické třídění článků [15].

Základem jsou tři techniky [16]:

- učení s učitelem - algoritmus má na vstupu učící množinu dat a validační množinu,

- učení bez učitele - algoritmus získá na vstupu pouze neoznačená vstupní data,
- zpětnovazebné učení - algoritmus se učí pomocí zpětné vazby.

Zmíněné techniky se snaží implementovat jeden ze základních úkolů strojového učení:

- klasifikace - rozdělení dat do konkrétních skupin,
- regrese - nalezení správných hodnot pro vstupní data,
- shlukování - rozdělení dat do skupin vzhledem k jejich vzájemné podobnosti.

1.3.1 Těžení textu

V logách integrační platformy je veškerá komunikace v textovém formátu. Kromě zpráv ve formátu XML zde jsou i zprávy ve formátu JSON, popřípadě výpisy chyb, pokud nějaká nastala. Protože v práci řeším analýzu takových logů, tak následující kapitola přiblíží těžení znalostí z textu.

Těžení textu má velký význam, neboť 80 % informací uložených v počítačích jsou textové formy [17].

Těžení textu je metoda, která těží informace z textu [18]. Přesto, že text mining spadá do kategorie data miningových metod, rozdíl je v tom, že nepracuje s číselnými a většinou ani nominálními hodnotami. Data mining dokáže detekovat skryté informace ze vstupních dat. Text mining informace nemá většinou ve svých datech nijak skryté. Při zpracování textu jde o automatizaci procesu tak, jak by ho zvládl člověk, počítačem. [18].

Dle zdroje [19] lze problematiku těžení textu rozdělit následovně:

- **Získání informací**

  Systémy získání informací identifikují v kolekci souborů takové, jaké jsou vhodné pro vstupní požadavek. Jde například o problematiku vyhledávacích nástrojů. Princip je takový, že hledáme-li konkrétní dotaz, jsou vybrány z kolekce všech souborů jen ty, která se dotazu týkají. To se rozhoduje například slovy použitými v dotazu a jejich synonymy.

- **Zpracování přirozeného jazyka**

  Zpracování přirozeného jazyka je jeden z nejtěžších problémů text miningu. V této disciplíně se řeší převod textu do mluveného slova, rozpoznání řeči a podobně. Princip je naučit stroje rozumět přirozenému jazyku. To pomáhá například při anotaci souborů.

- **Těžení dat**

  Těžení dat je proces, který hledá skryté vzory v datech. Při použití s text miningem je využité pro prezentaci různých výsledků koncovému uživateli.

- **Extrakce informací**

  Extrakce informací je proces, kde jsou vytažena data ze vstupu a vložena do logických struktur [20].

- **Transformace textových dokumentů**

  Aby bylo možné jednotlivé textové dokumenty klasifikovat nebo shlukovat je třeba je převést na číselný vektor. V této části kapitoly se budu zabývat možnostmi takového převodu.

  TF-IDF

  TF-IDF je zkratka anglického názvu *Term Frequency Inverse Document Frequency*. Překlad je snazší, pokud je rozdělen na Term Frequency, což je v překladu četnost slova a Inverse document frequency, jenž znamená: převrácena četnost dokumentu [21].

  Četnost slova vyjádříme následovně: máme slovo w a množinu dokumentů D, skládající se z dokumentů $d_1, d_2 \dots d_3 \in D$. Potom četnost slova $TF(w, d_x)$ vyjadřuje kolikrát se slovo w vyskytlo v dokumentu d_x.

  Převrácená četnost dokumentu vystihuje jak podstatné slovo w je. Značíme ji jako

  $$IDF(w, D) = \log \frac{|D|}{|D_w \subseteq D|},$$

  kde $|D|$ vyjadřuje velikost množiny všech dokumentů a D_w množina všech dokumentů, ve kterých se slovo w. $|D_w|$ pak značí velikost takové množiny [21].

  Pro slovo w v dokumentu d vypočteme TF-IDF následovně:

  $$TF - IDF(w, d, D) = TF(w, d) * IDF(w, D).$$

  Na základě uvedeného vzorce jsme schopni reprezentovat textový dokument pomocí vektoru. Vektor takového dokumentu bude vždy nezáporný a bez úprav bude mít tolik dimenzí, kolik má jednoznačných slov v sobě.

  Snížení dimenzionality

  Při práci s textovými dokumenty je třeba snížit jejich dimenzionalitu. Jak je uvedeno v sekci 1.3.1 bez úprav dimenzionality TF-IDF pomůže vytvořit vektor o velikosti rovné počtu unikátních slov. Do takového seznamu slov by ovšem v ten moment mohly zapadat slova stejného významu, například jinak skloňována. Dále bude-li pro jednotlivá slova

v dokumentu oddělovač mezera, mohou vzniknout jako dvě unikátní slova například „slovo“ a „slovo,“.

Před snížením dimenzionality je vhodné zbavit se veškeré interpunkce. Kromě interpunkce je vhodné i text převést kompletně na velká nebo malá písmena.

Pro snížení dimenzionality se používá odstranění takzvaných stop-slov. Jde o slova taková, která nemají velký význam. Příklad pro to mohou být předložky a spojky. Pro mnoho jazyků jsou k dispozici slovníky s takovými slovy. Dle konkrétní charakteristiky textového dokumentu může být vhodné nadefinovat si svá stop-slova.

Dalším nástrojem pro redukci počtu dimenzí je stemitizace. Cílem stemitizace je redukce slov tím, že jsou převedena na kořen slova [22]. Je třeba si uvědomit, že tímto krokem je možné ztratit původní význam slova a je tedy třeba promyslet, jestli na konkrétních datech stemitizaci využít.

1.4 Těžení Asociačních pravidel

Jednou z prvních myšlenek jak dosáhnout cíle bylo využití metody těžení asociačních pravidel.

1.4.1 Definice

Cílem těžení asociačních pravidel je najít zajímavé korelace či časté vzorce v množině dat uložených v relační databázi nebo jiných uložištích [23]. Jako ukázka praktického využití si lze představit využití v obchodních řetězcích. Obsah nákupu, který si zákazník zakoupil, je uložen v rámci jedné transakce. Algoritmus vyhledá zajímavé vzorce a pravidla mezi položkami v transakcích. Výsledkem je možnost predikovat co si zákazník zakoupí. Jako příklad uvedeme zákazníka, který si koupil housku a salát. Jako predikce další položky lze očekávat hamburgerové maso. Tato znalost pak lze využít k lepšímu přemístění zboží v regálech či k osobnějším nabídkám zboží v reklamních tiskovinách.

Těžení asociačních pravidel lze dle zdroje [24] popsat následovně: Nechť $I = I_1, I_2, \ldots, I_m$ je množina binárních atributů, kterou budeme nazývat položka. Nechť T je databáze transakcí. Každá transakce t je reprezentována binárním vektorem, kde platí, že $t[k] = 1$ pokud t zakoupila položku I_k. V opačném případě $t[k] = 0$. Nechť X je množina některých položek z I. Říkáme, že transakce t splňuje X pokud platí pro všechny položky I_k v X pravidlo $t[k] = 1$.

Asociačním pravidlem rozumíme implikaci $X \implies I_j$, kde X je množina některých položek v množině I a $I_j \in I$ ale zároveň $I_j \notin X$.

Pravidlo $A \implies B$ platí s podporou S v transakci T, kde $S(A \implies B) = P(A \cup B)$. Pravidlo $A \implies B$ má v transakci T spolehlivost C, kde $C(A \implies B) = P(B|A)$.

Cílem těžení je zjistit vztah mezi různými položkami tak, že přítomnost některých položek v transakci implikuje přítomnost jiné položky.

1.4.2 Myšlenka pro využití

Myšlenka využití při analýze logu spočívala v tom, že by každá jednotlivá zpráva byla počítána jako transakce. Jednotlivá slova (po předzpracování) by tvořila položky. Pak by bylo možné predikovat, že výskyt některých termů bude znamenat například to, že dojde k odmítnutí nevalidního požadavku. Tyto získané informace by ale neměli nikterak velkou hodnotu. O odmítnutí požadavku se v platformě dozvíme zpravidla v rámci milisekund v synchronní odpovědi.

Velmi pravděpodobně bezpečnostní rizika budou přibývat. Tato možnost by šla využít pro již existující nebo alespoň několikrát se vyskytující případy.

Z těchto důvodů jsem tuto myšlenku zavrhl s tím, že bude lepší vyzkoušet takové metody, které budou mít šanci rozpoznat špatný požadavek, i když ho uvidí poprvé. Tyto metody jsou rozepsány v následujících sekcích.

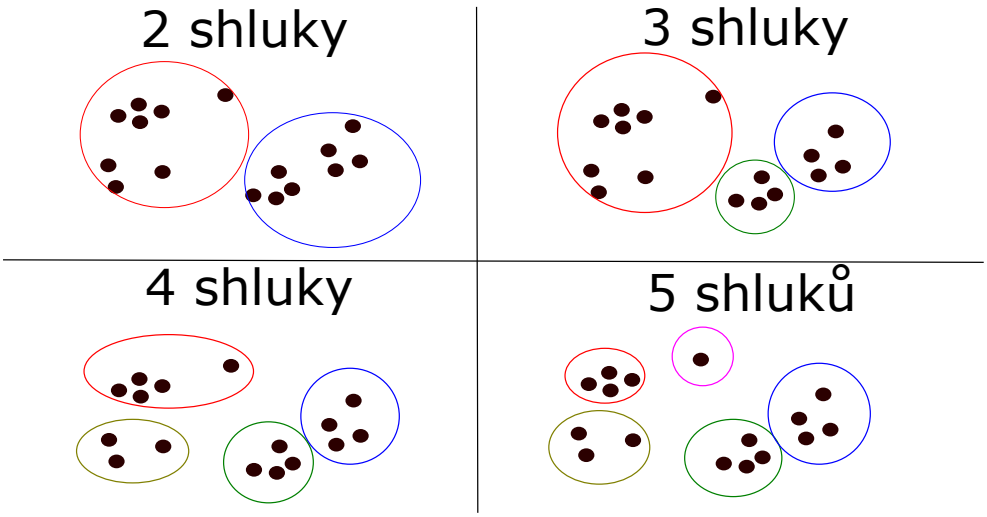
1.5 Shluková analýza

Jako jednu z možností vyřešení detekce bezpečnostních rizik na základě dat z logů jsem si vybral shlukovou analýzu (dle anglického názvu také nazýván clustering). Clustering využívá znalosti ze vstupních dat k tomu, aby dokázal vstupní požadavky rozdělit do shluků. Všechny informace jsou do těchto shluků přiřazovány na základě podobnosti. Podobnost lze definovat podle charakteristiky dat a požadovaném účelu shlukování. Jednotlivé shluky pak tvoří užitečné a logické skupinky .

Všechny objekty ve shluku si jsou navzájem podobné a zároveň se nepodobají objektům v jiném shluku [25]. Na obrázku 1.5 je ukázka několika příkladů, kdy je na stejná data použita shluková analýza s jiným počtem shluků.

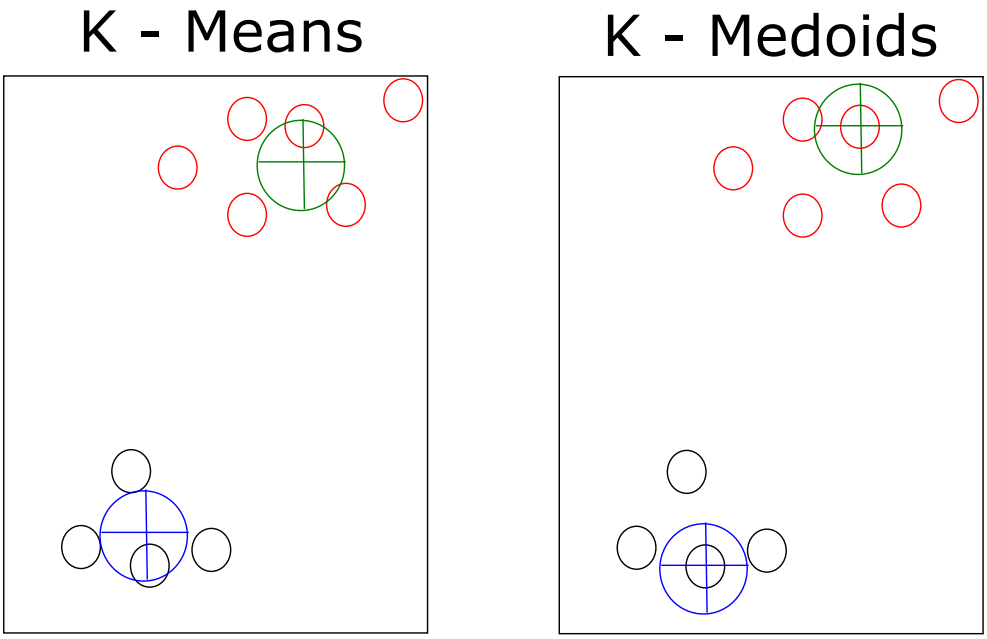
1.5.1 K-means

K-means a K-medoids jsou metody shlukové analýzy. Tyto metody jsou založené na principu centroidů. Centroidem je vyjádřen pomyslný střed každého shluku. V případě K-means jde o střed shluku nezávisle na tom, je-li tento bod i objektem vstupních dat, nebo ne. K-medoids jako střed shluku určí vždy nejvhodnějšího zástupce ze vstupních dat. Graficky je tento rozdíl zobrazen na obrázku 1.6. Z obrázku vyplývá, že shluk v K-means může mít střed mimo data, kdežto K-medoids má možnost středem shluku určit pouze nejvhodnější objekt ze vstupních dat.



Obrázek 1.5: Ukázka shlukové analýzy s různým počtem shluků

Tyto algoritmu patří do kategorie bez učitele. Algoritmy mají na vstupu jednu množinu dat. Na této množině se naučí data třídit do jednotlivých shluků dle podobnosti.



Obrázek 1.6: Ukázka shluku a jeho centroidu pomocí metody K-means a K-medoids

Postup algoritmu pro K-means je nejdříve vybrat K bodů jako centroidy.

Tyto centroidy lze vybrat náhodně nebo prvních N. Volbou prvních centroidů se zabývá zdroj [26]. Po vybrání středů se všechny data přiřadí k nejvhodnějšímu shluku. Výběr takového shluku často ovlivňuje například vzdálenost bodu k centroidu. Následuje přepočet nových středů. Přiřazení do shluků a výpočet nových centroidů se opakuje, dokud shluky mění. Po té považujeme algoritmus za hotový a data rozdělená do skupin.

Algoritmus s náhodným výběrem středů:

1. náhodně vyber $k > 0$ středů,
2. vypočti vzdálenost každého bodu ke všem středům,
3. přiřaď bod ke shluku k jehož středu má nejmenší vzdálenost,
4. vypočti nové středy shluků,
5. vypočti nové vzdálenosti bodů ke všem středům,
6. pokud se žádná data nezměnila skonči, jinak pokračuj bodem 3.

Jedním z kroků algoritmu je přiřadit vstupním objektům správný shluk. K tomuto důvodu je nutné definovat funkci pro měření vzdáleností mezi jednotlivými objekty [25]. Požadavky na takovou funkci jsou, aby byla jednoduchá, protože je velmi často volaná. Pokud data jsou v eukleidovském prostoru používají se například následující funkce na měření vzdálenosti:

1.5.1.1 Eukleidova vzdálenost

Eukleidova vzdálenost určuje vzdálenost mezi dvěma body v eukleidově prostoru. Výpočet euiklediovy vzdálenosti mezi body $p = (p_1, p_2, \ldots, p_n)$ a $q = (q_1, q_2, \ldots, q_n)$ je

$$d_e(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

1.5.2 Cosinova vzdálenost

Cosinova vzdálenost se vypočítá následovně [27]: Máme vektor $A = [A_1, A_2, \ldots, A_n]$ a vektor $B = [B_1, B_2, \ldots, B_n]$. Cosinovu vzdálenost vypočteme jako

$$cos(A, B) = \frac{A * B}{||A|| * ||B||},$$

kde $||A||$ vyjadřuje velikost vektoru A. Stejně tak velikost vektoru B je označena $||B||$.

1.5.2.1 Manhattanská vzdálenost

Dalším příkladem měření vzdáleností bodů $p = (p_1, p_2, \ldots, p_n)$ a $q = (q_1, q_2, \ldots, q_n)$ v eukleidově prostoru je manhattanská vzdálenost. Její výpočet je dle vzorce:

$$d_m(p, q) = \sum_{i=1}^{n} |p_i - q_i|.$$

1.5.3 Důvod využití

Informace, které integrační platformou prochází jsou velmi různorodé. Požadavky je možné rozdělovat do různých skupin jako například:

- skupiny dle druhu služby (SOAP, REST, databázová ...),
- skupiny dle služeb (služby na notifikace, na objednávky ...),
- skupiny dle toho, zdali je zpráva požadavek nebo odpověď.

Jako jedno z možných rozdělení je možné na požadavky v pořádku, podezřelé požadavky a chybné požadavky. Cílem je najít takovou konfiguraci, která by podobné rozdělení dokázala najít. Tedy rozeznat požadavky, které jsou v pořádku (běžná komunikace na integrační platformě.) od těch, jenž jsou podezřelé (ne příliš častý vzor požadavku, ...), či zaručeně chybné (chyba protokolu, validace, ...).

Vhodné rozdělení může být i pouze na podezřelé zprávy a zprávy korektní.

1.6 Detekce anomálie

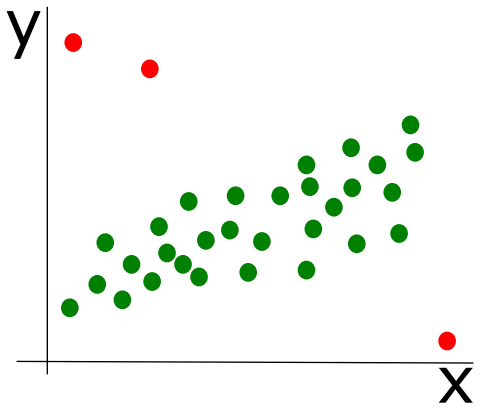
Většina požadavků, která projde skrz integrační platformu jsou v pořádku. Chyba či podezřelá zpráva se vyskytují zpravidla minimálně. Proto jsem jako další metodu zvolil detekci anomálie. Detekce anomálie je proces, při kterém se vyhledávají taková data, která se od ostatních výrazněji liší. Detekování odlišností se využívá právě při zajišťování bezpečnosti [28]. Příklad anomálie je na obrázku 1.7, kde jsou červeně zvýrazněné anomálie.

Princip detekce anomálií je definovat běžná data / chování. Následně veškerá data / chování, která těmto požadavků neodpovídají označit za anomálie 1.7.

1.6.1 Metody detekce anomálie

Detekce anomálií lze rozdělit na tři základní druhy [29]:

- statistické rozdělení
- metody založené na vzdálenosti



Obrázek 1.7: Ukázka anomálií v datech

- metody založené na hustotě

Statistické metody očekávají, že data mají průběh nějakého statistického rozdělení. Tomu ja následně přizpůsoben přístup zjišťování anomálií. Zpravidla proto tyto metody nejsou v praxi využívané [29].

U metod založených na vzdálenosti se vypočítá vzdálenost konkrétního data a jeho sousedů. Je-li nalezena vzdálenost větší než předem daný práh, je cílový prvek chápán jako anomálie.

Metody založené na hustotě vypočítávají takzvaný LOF (Local Outlier Factor). LOF je hodnota, která u každého prvku určuje míru, jak moc je anomálií [30].

Základ výpočtu LOF je výpočet vzdálenosti ke $k > 0$ nejbližším sousedům. Tato vzdálenost je použita pro odhad hustoty. Následné porovnání hustoty elementu s hustotami svých k sousedů rozhoduje o tom, bude-li označen za anomálii.

Definice LOF dle [30] je popsána pomocí několika dílčích definicích:

1.6.1.1 K-vzdálenost

Nechť pro libovolné celé číslo k, kde $k > 0$ je určena *k-vzdálenost* objektu p $(k - dist(p))$ jako vzdálenost $d(p, o)$, mezi objektem p a objektem $o \in D$, kde platí:

- pro nejméně k objektů $o' \in D \backslash p$ platí, že $d(p, o') \leq d(p, o)$ a zároveň
- pro alespoň $k - 1$ objektů $o' \in D \backslash p$ platí, že $d(p, o') < d(p, o)$.

Množinu k nejbližších sousedů definujeme jako $N_k(p)$

1.6.1.2 Dosažitelná vzdálenost

Nechť k je přirozené číslo. Dosažitelná vzdálenost objektu p s ohledem na objekt o je definována následovně:

$$reach - dist_k(p, o) = max(k - distance(o), d(p, o)).$$

1.6.1.3 Hustota dosažitelnosti

Hustota dosažitelnosti objektu p je definována jako

$$lrd_{N_k}(p) = \frac{1}{\frac{\sum_{o \in N_k(p)} reach - dist_k(p,o}{|N_k(p)|}}.$$

Hustota dosažitelnosti lze popsat jako inverze průměru dosažitelné vzdálenosti k nejbližších sousedů p.

1.6.1.4 Činitel anomálie objektu p (LOF)

LOF objektu p je definován jako

$$LOF_{N_k}(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_{N_k}(o)}{lrd_{N_k}(p)}}{|N_k(p)|}.$$

1.6.2 Analýza hlavních komponent

Analýza hlavních komponent, častěji označována jako PCA z anglického překladu *Principal Component Analysis*. PCA provede analýzu vstupních dat. Tyto data se skládají z mnoha proměnných, které na sobě mohou být závislé. Účelem je vybrat z těchto proměnných důležitou informaci a tu reprezentovat jako nové proměnné zvané hlavní komponenty [31]. Tyto hlavní komponenty jsou lineární kombinací originálních hodnot.

Kapitola 2

Návrh řešení

V této kapitole se zabývám principy, technologiemi a algoritmy, které jsem se rozhodl použít k tomu abych, splnil cíle této práce. Tedy vytvoření aplikace, jenž umožní sledovat bezpečností rizika v reálném čase.

Návrh řešení plyne z toho, že hlavním stavebním kamenem je integrační platforma Unify. Tato platforma využívá již některé metody a standarty.

2.1 Architektura aplikace

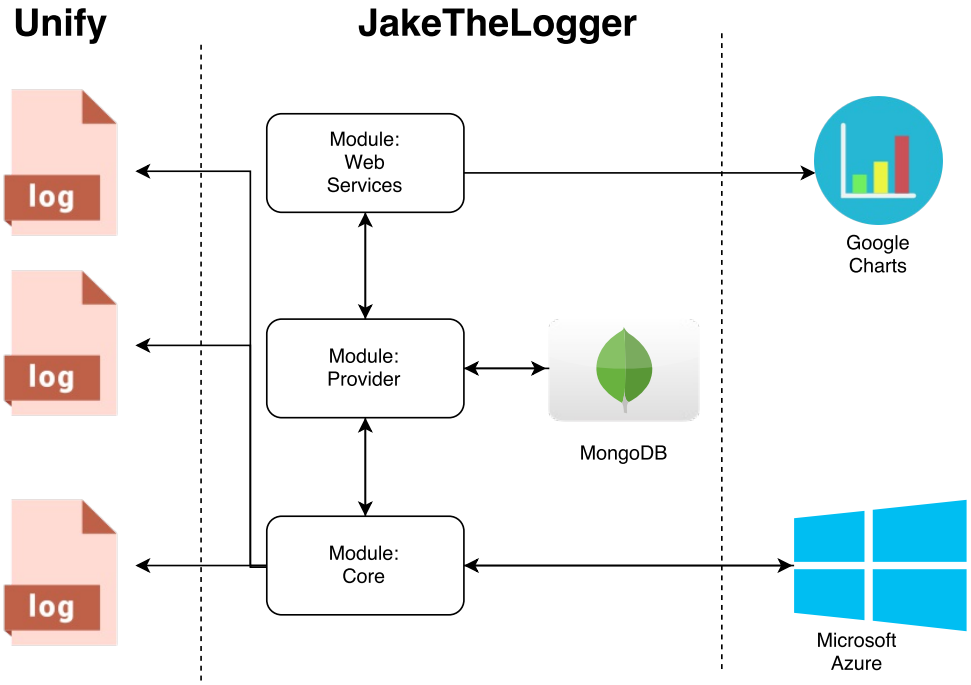
Požadavkem na aplikaci je, aby zprávy zpracovávala a predikovala v reálném časem. Proto je princip položen na čtení požadavků proudících přes platformu. Dále jsou předzpracovány a odeslány do cloudového řešení Microsoft Azure (více v sekci 2.2). Po přijetí odpovědi je výsledek uložen do databáze. Pro vizualizaci dat slouží REST API [32], které vypisuje předem definované informace v ideálním formátu pro zobrazení v Google Charts [33].

Pro snazší představu o architektuře aplikace poslouží obrázek 2.1, na kterém je vidět high level desing [34].

Základem celé aplikace je neohrozit stávající integrační platformu. Proto jsem se rozhodl, že informace o proběhlé komunikaci získám pomocí čtení logovacích souborů. Čtením přírůstků k jednotlivým auditovým logům získám chtěné požadavky a pro Unify to nepředstavuje žádnou zátěž.

Dalším stavebním kamenem je použitý aplikační server Jboss (více v kapitole 2.3). Na serveru je celá aplikace. Dochází zde k předzpracování zpráv, jejich odeslání do Microsoft Azure (2.2) a také k ukládání do DB.

Jak jsem již zmiňoval provoz z platformy je po zpracování odesílán do MS Azure, zde jsou definovány jednotlivé algoritmy, jejichž výsledky jsou vráceny zpět do aplikačního serveru. Azure jsem se rozhodl používat, protože umožňuje rozložení výkonu na server Microsoftu a protože využití služeb v cloudu se stává stále oblíbenějším. Díky spolupráci s Microsoftem je možné i získat, popřípadě zakoupit, instanci Azure do vlastní sítě. Po diskuzi s Yuriy Zaytsev



Obrázek 2.1: high level desing aplikace s pracovním názvem JakeTheLogger

z týmu Microsoftu jsem získal i potvrzení, že podobné řešení postavené na jboss se v MS Azure nevyskytuje (k březnu 2017).

Získané výsledky jsou zpracovány a uloženy do NoSQL databáze MongoDB (více v kapitole 2.9).

Aby výsledky nebyly jen hodnoty uložené v databázi, je použité REST API, přes které lze výsledky sdílet. API je navržené tak, aby v případě použití Google Charts nevznikly žádné potíže.

Google Charts se u cílového zákazníka používají již nyní například na zobrazení stavu objednávek. Proto se jejich použití jeví jako další logický krok. Nicméně, není problém stejné API použít pro svoji libovolnou aplikaci, která hodnoty použije buď pro zobrazování přehledů, nebo jako jeden z dalších vstupů například do různých systémů SIEM.

2.2 Microsoft Azure

Na integrační platformě Unify [11] je předpokládaný maximální provoz 20 požadavků za vteřinu. Vzhledem k takto silnému provozu bude potřeba i přiměřeně velký výpočetní výkon.

Spolupráce se společností Miscrosof [35] mi umožnila jako řešení vyzkoušet její cloudové služby Microsfot Azure [36]. Dostupné na stránkách `https://studio.azureml.net`.

Microsoft Azure je sada integrovaných cloudových služeb. Azure nabízí cloudová řešení pro mnoho činností. Motivací k použití této služby k detekci bezpečnostních rizik je nástroj Microsoft Azure Machine Learning Studio [37].

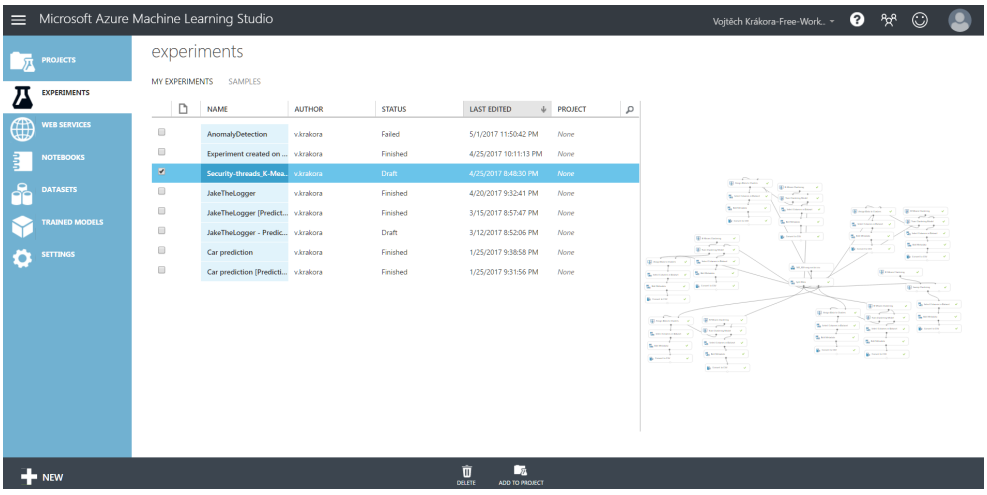
Microsoft Azure Machine Learning Studio je plně cloudová služba, která umožňuje vytváření prediktivních modelů pro strojové učení [37]. Výhodou studia je to, že veškerý výkon je rozprostřen vevnitř cloudu. Díky grafickému rozhraní je snadné vytvořit učící model, který je následně převeden do modelu prediktivního.

Aby měl prediktivní model smysl, je třeba mu poskytovat nějaká data, u kterých je predikce využita. K tomu se využívají webové služby. Prediktivní model se vystaví na specifické URL adrese. Zde je pak očekáván na vstupu konkrétní formát JSONu a služba vrací předem definovanou odpoveď se správnými parametry.

Výhodou využití cloudu je přenesení výpočetní zátěže mimo společnost. Naopak rizikem je problém s konektivitou, který může vytvořit výpadek služby a nebude tedy možné po tuto dobu predikovat rizika. Jednou z možností, jak řešit takové riziko je nechat přenést instanci MS Azure do své sítě.

2.2.1 Prostředí Microsoft Azure Machine Learning Studio

Azure Machine Learning Studio (dále jako Azure nebo MS Azure) usnadňuje vytváření algoritmu pro strojové učení. Úvodní obrazovka grafického rozhraní je zobrazena na obrázku 2.2.



Obrázek 2.2: Grafické rozhraní Microsoft Azure Machine Learning Studia

Základem veškerých experimentů jsou vstupní data. Ty lze do Azure uložit jako vlastní dataset. Data mohou být předzpracována, ale není to podmínkou. Azure poskytuje řadu nástrojů pro přípravu dat a stejně tak i možnost vytvoření vlastních komponent napsaných v jazyce R nebo Python.

Samotný experiment se skládá z jednotlivých komponent. Ty jsou tedy buď již definované nebo vlastní. Experimenty jsou děleny na trénovací experimenty a na prediktivní. Trénovací experiment je takový, který naučí celý experiment predikovat data. Nachází se v něm zpravidla trénovací množina (může být rozdělena dále i na validační část), vybrané komponenty představující algoritmy strojového učení a prvky zobrazující výsledky. Mezi těmito hlavními částmi se mohou vyskytovat další komponenty například pro předzpracování dat, rozdělení dat, validaci, stažení a podobně. Pokud jsme s výsledky trénovacího experimentu spokojení může být automaticky vygenerován experiment prediktivní. Ten je připraven na vstup získat data a vrátit výsledky z naučeného algoritmu. Takový experiment je možné vystavit jako webovou službu.

Další záložky obsahují poznámky nebo například vystavené webové služby.

2.3 JBoss

Základem integrační platformy Unify je aplikační server JBoss AS 7 [11]. Je potřeba, aby cílová aplikace byla kompatibilní, proto se i její vývoj bude soustředit na nasazení na JBoss.

Jboss AS je aplikační server pro Javu EE[38]. Aplikační server je mezivrstva mezi operačním systémem a aplikacemi. Pomáhá zjednodušovat vývoj aplikací. Umožňuje přístup na souborový systém, ale také transakční zpracování požadavků nebo ukládání dat do databáze. Je využíván takzvanými enterprise aplikacemi [39].

2.4 Logování Unify

Abychom nenarušil novou komponentou provoz platformy, budou veškeré zprávy, které přes ni budou vedeny čteny z logu. Platforma Unify loguje veškerý průběh do souborových logů. Protože celá integrace je založena na Javě, je využit logovací framework Log4j [40]. Každý požadavek, který na platformu přijde je zalogován do audit.logu. Zpráva je vždy na jedné řádce a zároveň na jedné řádce je povolena pouze jedna zpráva.

```
REQ;20170117;16:03:01;WorkRequestMgmtService;
cancelOrderedService;esb1-1484665381312-5369
```

Kód 2.1: Část řádky v auditovém logu Unify

Jedna řádka auditového logu obsahuje:

- informaci o čase zápisu,
- závažnost z pohledu javy,
- název třídy, který uložil řádku do logu,
- název vlákna, který zapisovanou část zpracovával,

- informaci zda je o požadavek (REQ) nebo odpověď (RES), či chyby (ERR),
- datum přijetí požadavku,
- čas přijetí požadavku,
- název logující komponenty,
- název operace,
- jednoznačný identifikátor požadavku,
- obsah požadavku.

Jednotlivé části logu jsou od sebe odděleny středníkem.

Vzhledem k tomuto principu jsem se rozhodl přistoupit k tomu, že se vybrané logovací soubory budou kontinuálně číst, kde se bude ke každému řádku přistupovat jako k samostatné zprávě, která bude následně zpracována dál.

Díky této volbě nebude nutné nijak zasahovat do integrační platformy Unify a minimalizuje se tím riziko, jakéhokoliv nebezpečí ze strany této aplikace.

Unify využívá pro některé služby logování do Oracle databáze. Ale vzhledem k tomu, že jde pouze o několik málo určených služeb, připojení na databázi by tak kromě případných komplikací ani nepřineslo žádaný účinek. Veškerá komunikace ukládaná do databáze je stejně duplikovaná i v logovacích souborech.

2.5 Předzpracování dat

Pro snazší práci s informacemi z logů jsem se rozhodl pro normalizaci jednotlivých požadavků. Normalizace je jeden z požadavků při zpracovávání textu [41].

V kapitole 2.4 jsou vidět informace, které se kromě samotné zprávy logují. V každé zprávě se objevuje takzvaná integrační hlavička. V té jsou základní údaje, jako čas odeslání, jednoznačné identifikátory, zdrojové a cílové systémy. Položka jako je časové razítko bude zpravidla pro každý požadavek jiná, stejně na tom budou jednoznačné identifikátory. Z tohoto důvodu jsem se rozhodl zvolit jejich nahrazení.

Při normalizaci dat jsem se podobně jako ve zdroji [42] rozhodl použít následovně:

- nahrazení všech čísel - pomocí speciálního symbolu „#“ nahradím všechny výskyty čísel,

- velikost písmem - všechna písmena jsou z velkých znaků převedena na znaky malé,

- nahrazení speciálních znaků -veškeré znaky jako jsou čárka, tečka . . . jsou nahrazeny mezerou,

- odstranění XML tagů - rozhodl jsem zpracovávat jen obsah zpráv bez XML tagů.

Nahrazení čísel se mi jeví jako logický krok. V jednotlivých požadavcích jsou čísla výsledky různých měření na sítí nebo právě časové razítko. Pro další využití považuji za podstatné vědět, že se v daném místě vyskytovalo číslo, než že to bylo nějaké konkrétní číslo.

Převod písmen na malá zajistí, aby slova, lišící se právě jen ve velikosti nějakých písmen byla vyhodnocena jako stejná.

Veškerá komunikace na platformě je převedena do XML (není-li již od počátku vedena v XML). Protože téměř u všech zpráv stejného druhu se používají ty samé XML tagy, nebudou pro další zpracování podstatné a budou zcela odstraněny. Algoritmus bude dále pracovat jen s reálným a normalizovaným obsahem zprávy.

Pro ilustraci normalizace poslouží následující ilustrační příklad:

```
<WebService>
  <header>
    <timestamp>2016-08-09T11:11:11</timestamp>
  </header>
  <body>
    <firstElement>Hello world!</firstElement>
    <secondElement>0</secondElement>
  </body>
</WebService>
```

Kód 2.2: Ilustrační příklad XML požadavku

ten by normalizaci vypadal následovně:

```
# # #T# # # hello world #
```

Kód 2.3: Ilustrační příklad po normalizaci

Čištění dat probíhá na straně aplikace. Do MS Azure tedy chodí už pouze korektní vstupní data. Vyčištění se stará o to, že nekorektní požadavky nejsou zpracovány. Je to pojistka proto, kdyby nějaká zpráva byla více řádková (například nechtěnou změnou logování). V takovém případě se nepodaří pomocí regulárního výrazu najít jednoznačný identifikátor platformy a „požadavek“ je ignorován.

2.6 Vytvoření vektoru

V sekci 1.3.1 jsem navrhl, jak textová data převést do vektoru. Tím je zaručené, že budou-li se data přenášet přes internet do Microsoft Azure, budou anonymizována. Z vektoru nedokážeme zpětně zprávu vyčíst. Jediným vodítkem, které vektor spojuje zpět s původní zprávou je jednoznačný identifikátor *platform-id*.

2.6.1 Forma vektoru

Pokud by nedošlo k žádné změně, vektor by byl tak velký, kolik unikátních slov by se v rámci zpráv našlo. Takový vektor by byl velmi obsáhlý. Mnoho dat by nemělo ani smysl, neboť slova, která se vyskytují ve všech zprávách nebo například pouze v jedné nemají příliš velký informační smysl. Z tohoto důvodu jsem vytvořil dva konfigurační parametry, které stanovují horní a spodní procentní hranici. Tyto hranice vymezují pouze ty slova, jejichž procentuální výskyt ve zprávách je právě v intervalu vytyčeném hranicemi.

Při jednom z experimentů jsem pro horní hranici 85 %, spodní hranici 10 % a velikost vektoru $|v| = 10$ získal slova uvedené v tabulce 2.1. Tato slova budou následně použita v algoritmu TF-IDF. Výsledek bude tvořit vektor o velikost rovnému počtu slov v tabulce 2.1.

Tabulka 2.1: Výběr slov a hodnoty vektoru pro zprávu err1-1490865735867-8212

četnost výskytu v %	**slovo**	**hodnota ve vektoru**
79	esb#	0.0
36	##	0.0
43	res	0.0
16	err	0.0114536341484269
16	org	0.0687218048905616
19	zis	0.0
18	threads	0.0107174901755745
82	thread	0.0
16	err#	0.0114536341484269
18	default	0.0107174901755745

2.7 Konstrukce shlukování

Ve studiu Azure machine learning je možné vytvářet své projekty, do projektů umístit své experimenty a ty následně vystavit jako webovou službu.

Základem úspěšného experimentu je vytvořit učící model. To je takový model, pro který máme zvolený cílový algoritmus a na připravených datech

ho naučíme aby dokázal v našem případě co nejlépe rozdělovat zprávy do clusterů.

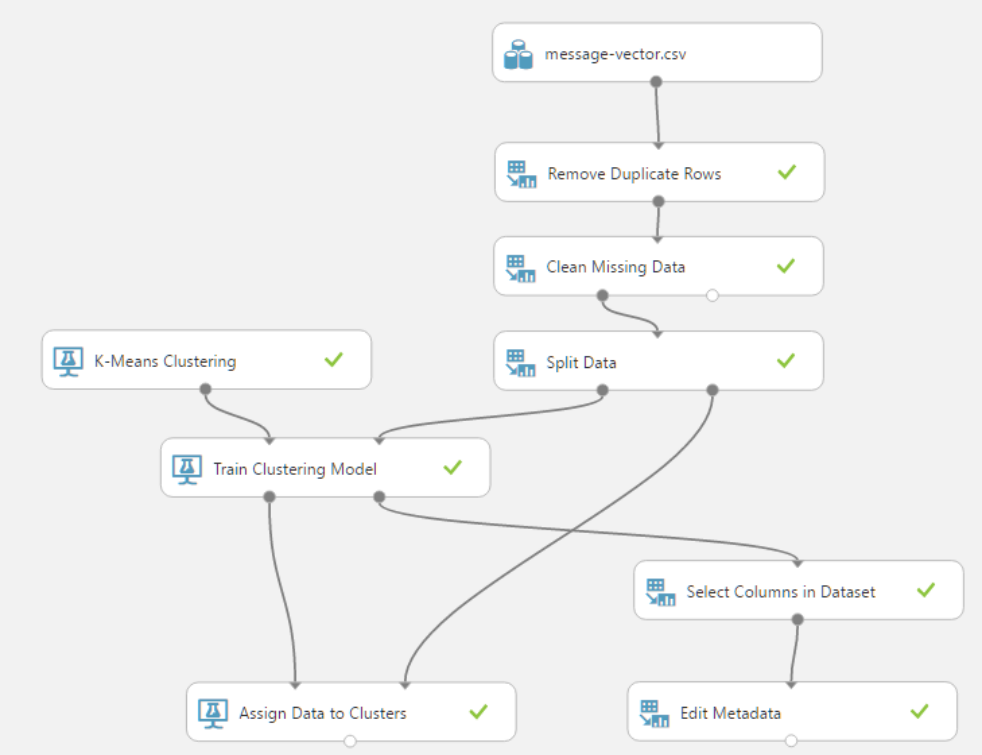
2.7.0.1 Předzpracování

Veškeré předzpracování a čištění dat probíhá v mojí aplikaci tak, jak je uvedeno v kapitole 2.5. I přes to jsem se rozhodl umístit základní předzpracování dat i trénovacího modelu.

Na obrázku 2.3 jsou dva moduly předzpracování:

- **Remove Duplicate Rows** - odstraní duplicitní řádky,
- **Clean Missing Data** - odstraní nekompletní řádky.

Modul **Split Data** slouží k náhodnému rozdělení vstupních dat. Zde to má smysl takový, že 80 % požadavků je zpracováno učícím algoritmem. Zbylých 20 % je použito pro otestování naučeného algoritmu.



Obrázek 2.3: Clustering k-means v prostředí MS AZURE ML Studio

2.7.0.2 Zpracování

Pro zpracování jsem zvolil K-Means shlukování. Jeho ideální nastavení je řešením kapitoly 4. Z modulu **Train Clustering Model** je možné získat naučený model a výsledek přiřazení vstupních dat. Pro lepší zobrazení výsledku přiřazení vstupních dat jsem zvolil pro finální zobrazení pouze sloupec s identifikátorem zprávy a informaci o clusteru, do kterého zpráva spadá. Modul **Edit Metadata** umožňuje přejmenovávat sloupce a měnit jejich datové typy. Také je dobrým nástrojem pro vizualizaci dat. Naučený model je propojen s modulem **Assign Data to Clusters**. Do tohoto modulu je zavedeno i zbývajících 20 % vstupních dat. Výsledný algoritmus je na nich otestován.

Na obrázku 2.3 je vidět celý vytvořený trénovací experiment.

2.8 Konstrukce detekce anomálie

Druhou možností, kterou jsem vyzkoušel je detekce anomálie. Jedním z předpokladů je, že korektní požadavky se od podezřelých budou lišit dostatečně na to, aby bylo možné je detekovat jako anomálie.

Princip předzpracování dat v Azure ML studiu je stejný, jako při konstrukci modelu pro shlukování. Řádky s chybějícími hodnotami a duplikované pro trénování nebudeme používat.

Kromě výše uvedené předzpracující části i zde je část učící a část vyhodnocovací. Hlavním rozdílem je, že na vstupu jsou dva datasety. Jedna množina dat je složena pouze z korektních požadavků. Druhá množina jsou data korektní i nekorektní v poměru 87:13. Algoritmus je naučen pouze na nepodezřelých vzorcích. Mixovaná data jsou pak použita při zkoušení naučeného modelu ve **Score Model**.

Trénovací model je vidět na obrázku 2.4.

Ideální nastavení i vhodný algoritmus jsou předmětem kapitoly 4.

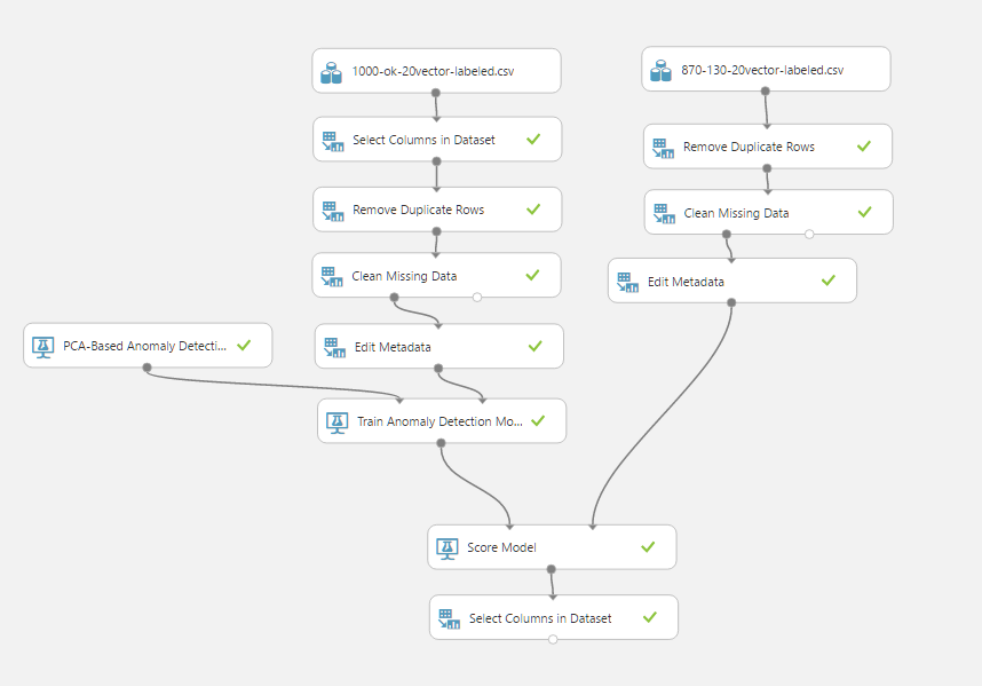
2.9 Ukládání dat

Pro potřeby aplikace je nutné v co nejrychlejším možném čase ukládat jednotlivé requesty. Při ohromném provoze, který se na integrační platformě vyskytuje, to je nutná podmínka pro to, aby bylo možné v reálném čase, jednotlivé požadavky zpracovávat.

Rozhodl jsem se za tímto účelem využít MongoDB [43], protože se očekává, že bude třeba ukládat v mimořádném případě až 30 záznamů za sekundu. Zpravidla bude docházet k více zápisům než čtením.

2.9.1 NoSQL

MongoDB patří do takzvaných NoSQL databází [43]. NoSQL v angličtině znamená „Not Only SQL“ [44], v překladu „Nikoliv pouze SQL“. Jde o skupinu



Obrázek 2.4: Detekce anomálií v prostředí MS AZURE ML Studio

nerelačních databází. Takové databáze nejsou primárně postavené na principu tabulek a zpravidla nepoužívají SQL pro práci s daty [44].

2.9.2 O MongoDB

MongoDB je licencovaná pod GNU AGPL v3.0 [45] licencí. Data jsou ukládána ve formátu BSON. BSON je binárně zakódovaná JSON [46].

V MongoDB se vytvářejí kolekce, každá kolekce obsahuje soubory. Soubory mají parametry [43]. Soubory a jejich parametry lze v čase libovolně měnit nebo přidávat. Což je výhoda, pokud zjistíme, že aktuální návrh není finální, vyhneme se problémům s migrací do nového schématu.

V rámci souboru je možné definovat čítač, který se využije k tomu, aby automaticky generoval jednoznačný identifikátor k souborům nebo lze využít parametr souboru **_id**. Ten vygeneruje jednoznačnou identifikaci, ze které jsme schopni například získat i čas vložení dokumentu do kolekce.

2.10 Prezentace dat

Vzhledem k tomu, že by aplikace měla být schopna určovat bezpečnostní rizika, je třeba nějakým způsobem prezentovat její výstupy. Monitoring na aplikaci Unify je momentálně postaven na tom, že konkrétní specializovaní lidé

kontrolují logy a v případě výskytu chyb, varování nebo jiné netypické události zjišťují co bylo příčinou.

Rozhodl jsem se tedy, že nejlepší bude grafické znázornění. Kromě údajů o tom, že byl zaznamenán požadavek, který je podezřelý budu grafy využívat i k prezentaci základního monitoringu. Jelikož se bude veškerá komunikace ukládat, bude vhodné prezentovat například i kolik požadavků na jednotlivé komponentě proběhlo za poslední hodinu a podobně.

Společnost Cetin [47], ve které bude aplikace testována a jenž je uživatelem integrační platformy používá pro různá grafická znázornění grafy od Google Charts[33].

Tyto grafy jsou napsané v jazyce Javascript. Je tedy možné jejich umístění například na intranetové stránky, kde se vysoce postavení lidé společnosti vyznají lépe než v jednotlivých monitorovacích aplikacích.

Na tomto základě jsem se rozhodl vytvořit REST API [32], jenž budou Google Charts schopny snadno konzumovat a v případné jiné aplikace, které by stály o podobná data budou schopny se jim přizpůsobit.

2.11 Využití dat systémy 3. stran

Do budoucna je potřeba počítat s rozšířením monitoringu a je proto vhodné aplikaci připravit tak, aby její výsledky mohly být využity v aplikacích 3. stran.

Lze předpokládat, že k monitorování bezpečnosti provozu budou použity systémy SIEM (Security Information and Event Management) [48]. SIEM funguje na principu, kdy zpracovává co nejvíce údajů, na jejichž základě pak rozeznává neočekávané situace a rizika [49].

Tím, že jsem se rozhodl data ukládat tak, jak uvádím v kapitole 2.9, bude libovolný SIEM po připojení do DB schopen získat jak originální zprávu, tak její normalizovanou verzi popřípadě i výsledek vyhodnocení mé aplikace.

Dále je možnost napojit SIEM i na REST API obdobně jako Google Charts v kapitole 2.10.

Tyto možnosti se vztahují i na jiné různé aplikace. Vznikne tím systém, který bude moct být využíván vývojáři. Díky tomu nebudou muset řešit problematiku detekce bezpečnostního rizika a bude jim stačit pouze aplikovat svůj problém a následně se připojit do databáze nebo využít webové rozhraní.

Kapitola 3

Realizace

3.1 Nutné přípravy pro JBoss

Aby aplikace mohla fungovat na aplikačním serveru, bylo potřeba provést některé konfigurační změny. Před každým nasazením je potřeba provést následující konfigurace.

3.1.1 Připravení modulů

V aplikaci využívám různé Java knihovny, abych k nim měl přístup i v aplikačním serveru, je nutné do něj přidat speciální modul.

Jboss umožňuje snadné přidání modulů. Veškeré moduly jsou umístěny v *wildfly/jboss-eap-7.0/modules/system/layers*. Zde jsem vytvořil svůj modul s konkrétními java knihovnami:

- commons-codec-1.10.jar,
- json-simple-1.1.1.jar,
- mongo-java-driver-3.4.2.jar.

Moduly se pak musí zaregistrovat v konfiguračním souboru *layers.conf*. Následně je možné, aby deploymenty uvnitř aplikačního serveru s tímto modulem využívali uvedené knihovny.

3.1.2 Port offset

Dále bylo nutné pro JBoss nastavit portový posun. Protože na serveru z pravidla běží více aplikací, je běžný problém v kolizi portů. Z tohoto důvodu jsem zvolil posun o 10000. Webové služby na tomto aplikačním serveru tedy místo portu 8080 běží na portu 18080 a podobně.

3.1.3 Zapnutí CORS

CORS (Cross-origin resource sharing), neboli *sdílené zdroje odjinud*, umožňuje odesílání odpovědí na požadavky z jiné domény [50]. V aplikaci je to potřebné pro REST API, kterého se následně dotazuje Google charts.

Corse se v Jboss povoluje v konfiguračním souboru pro standalone aplikaci *standalone.xml* pro doménovou *domain.xml*.

3.2 Vytvoření modelu na Azure

Již bylo uvedeno, jak jsem navrhl vytvoření trénovacího algoritmu v Azure ML studiu. Po natrénování algoritmu je vhodné z něj udělat prediktivní model. Tedy takový, který se pokouší správně detekovat výstup na základě nového vstupu. V Azure ML je toto možné udělat z trénovacího modelu na několik kliknutí. Vznikne model, který se liší od trénovacího tím, že vstup přijímá z webové služby a stejně tak na webovou službu odesílá i výstup. Je možné i v prediktivním modelu spustit na vstupu uložený dataset místo webové služby.

3.2.1 Clustering v Azure

Na obrázku 3.1 je k dispozici náhled na prediktivní model shlukování. Modul nesoucí název **JakeTheLogger** je natrénovaný model z kapitoly 2.7. Ten je přiveden na vstup modulu, který na základě natrénovaného algoritmu přiřadím datům (druhý vstup) shluky, do kterých spadají. Výsledek je odeslán pomocí webové služby.

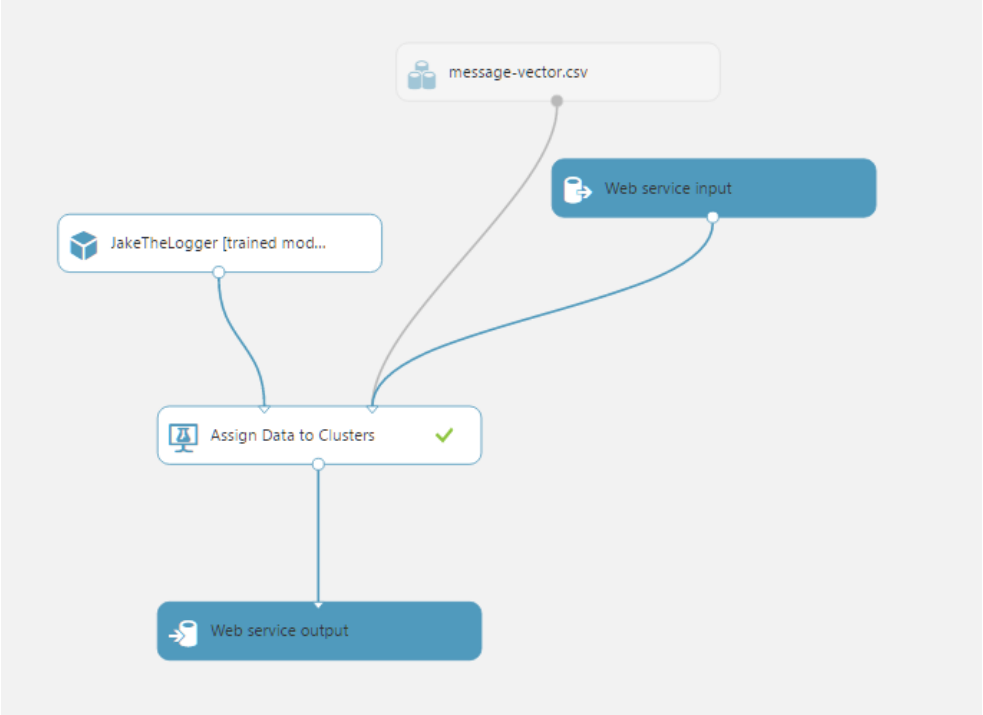
3.2.2 Detekce anomálií v Azure

Detekce anomálií má v případě prediktivního modelu velmi podobný model jako shlukování. Na obrázku 3.2 je **AnomalyService**, což je název trénovacího algoritmu z kapitoly 2.8. Vstupem je opět webová služba. Méně zvýrazněný vstup je dataset, který by se použil, pokud bychom na vstupu nechtěli dostávat data z webové služby.

3.2.3 Webová služba

Po vytvoření webové služby získáme takzvaný „API key“. Tento řetězec bude sloužit pro přihlášení se do Azure, při dotazování se na konkrétní službu.

Také je možné službu otestovat. Otevře se nám okno s očekávanými políčky (obr. 3.3). Po vyplnění políček se zobrazí odpověď z prediktivního modelu. Tímto způsobem můžeme otestovat funkčnost nebo pár vzorků. Jiná použití by byla velmi časově a zdrojově nevýhodná.



Obrázek 3.1: Prediktivní model shlukování v Azures

3.3 Databáze

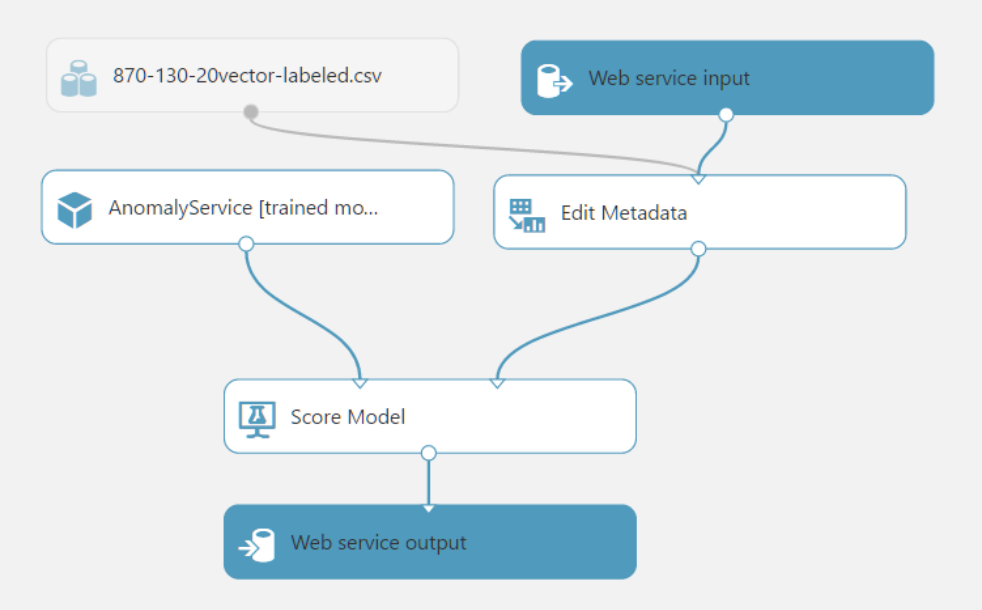
Jak již bylo uvedeno, pro ukládání dat byla zvolena MongoDB. Základ pro běh aplikace jsou tyto kolekce:

- **messages** - veškerá komunikace,
- **terms** - slova pro použití v TF-IDF algoritmu,
- **used-messages** - zprávy, ze kterých byly vypočteny slova v kolekci terms (podstatné pro výpočet TF-IDF).

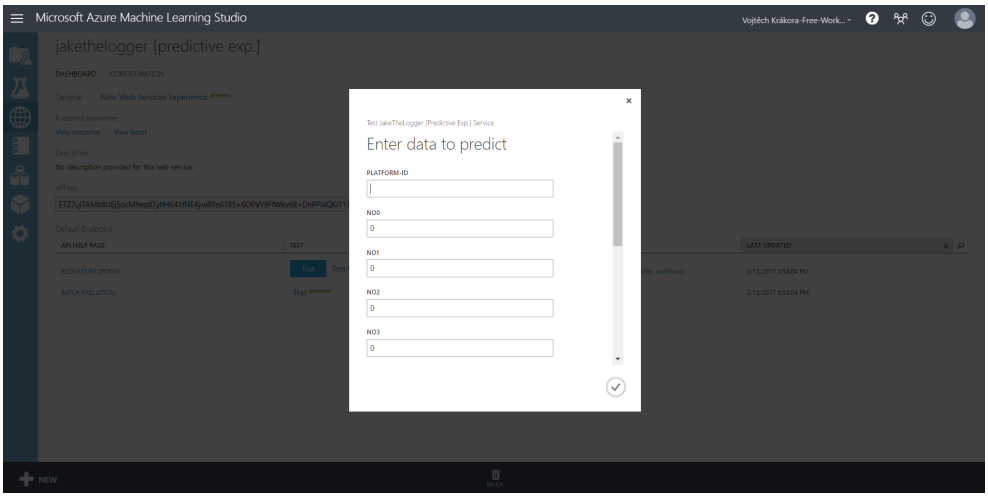
3.3.1 Kolekce terms

V práci využívám databázi k ukládání všech slov, ze kterých se tvoří vektor, jenž reprezentuje konkrétní požadavek (více v kapitole 2.6). Tím není potřeba je mít v paměti a při případném výpadku je znova vypočítávat.

V kolekci *terms* ukládám soubory jejichž struktura je automatický identifikátor, slovo pro konstrukci vektoru, časové razítko přidání dokumentu do kolekce a hodnota v procentech vyjadřující v kolika souborech kolekce used-messages se slovo vyskytuje.



Obrázek 3.2: Prediktivní model v detekci anomálií v Azure



Obrázek 3.3: Zobrazené okno pro otestování prediktivního modelu jako webové služby

3.3.2 Kolekce messages a used-messages

Další kolekcí je kolekce *messages*, která má stejnou strukturu jako *used-messages*. V té jsou uložené veškeré požadavky, které byly přečteny z logů integrační platformy. Protože ještě před uložením do kolekce dochází v Azure k vyhodnocení, je zpráva uložené i s informací, která určuje zdali je požadavek vyhodnocen

jako bezpečností riziko nebo není.

Struktura každého souboru je:

- **__id** - automaticky generovaný identifikátor,
- **timestamp** - čas uložení souboru,
- **original-message** - původní požadavek tak, jak byl převzat z logu integrační platformy,
- **normalized-message** - požadavek ve znormalizované podobě,
- **platform-id** - jednoznačný identifikátor v rámci integrační platformy,
- **assignment** - informace od Azure s výsledkem přiřazení kategorie.

Význam kolekce *used-messages* je pro výpočet slov do kolekce *terms* a pro následný výpočet TF-IDF algoritmu.

3.3.3 Práce s MongoDB v Javě

V implementaci jsem vytvořil třídu *MongoClientService* (aby bylo možné třídy využívat i v jiných modulech, musí se taková třída skládat z interfacu a jeho implementace, v textu se budu bavit o celku implementace a interfacu dohromady například jako o třídě MongoClientService). Tato třída umožňuje distribuci konkrétní databáze napříč celou aplikací.

V jednotlivých modulech si vyvoláme instanci konkrétní databáze a nad tou jsme schopni pracovat. Ovladače pro MongoDB nám umožňují jak data vkládat, tak je číst.

3.4 Uložení dat

V sekci 3.3 byla představena databáze, která bude využívána. V programu dochází k ukládání dat v následujících krocích.

3.4.1 Použité zprávy

Pokud program detekuje, že kolekce terms je prázdná, zpravidla se jedná o první start a je třeba vypočítat vhodné slova pro výpočet vektoru. Protože výpočet vektoru je přes již zmiňovaný algoritmus TF-IDF, je nutné mít zachované jak slova ze kterých vektor vypočteme, tak zprávy, ze kterých jsme vypočetli tato slova.

Proto při prvním spuštění se k zpráv, kde $k > 0$, uloží do kolekce used-messages.

3.4.2 Vektorová slova

Dále přichází zápis do kolekce terms. Po uložení všech zpráv v kolekci used-messages zjistíma v kolika dokumentech se nachází (vyjádřeno procenty). Na tento výpočet stačí použít převrácenou hodnotu IDF „zbavenou“ logaritmu:

$$\frac{1}{e^{idf(docs,term)}},$$

kde *idf* je funkce popsaná v sekci 1.3.1, *docs* je množina všech dokumentů (v tomto případě dokumenty z kolekce used-messages) a *term* je konkrétní slovo z kolekce terms, pro které děláme tento výpočet.

3.4.3 Zprávy

Všechny zprávy poté, co znají svůj shluk nebo zdali jsou anomálií jsou uloženy do kolekce messages.

3.4.4 Proces ukládání dat

Kolekce messages je plněna teprve po té, co se vrátí odpověď z MS Azure. Celá zpráva je ve třídě *LogListener* ukládána do formátu *AuditLogMessage*. Ta kopíruje svým obsahem právě kolekci messages. *AuditLogMessage* je vlastníkem metody *toDBObject*, která obsah třídy vrátí ve vhodném formátu pro uložení do MongoDB.

Proces uložení zpráv je na zobrazen na obrázku 3.4. Nejdříve jsou data přečtena z cílové platformy. Dalším krokem je výpočet vektoru. Vektor je odeslán do Azure ML a je přijat výsledek. Nakonec je zpráva i s výsledkem uložena do kolekce messages.

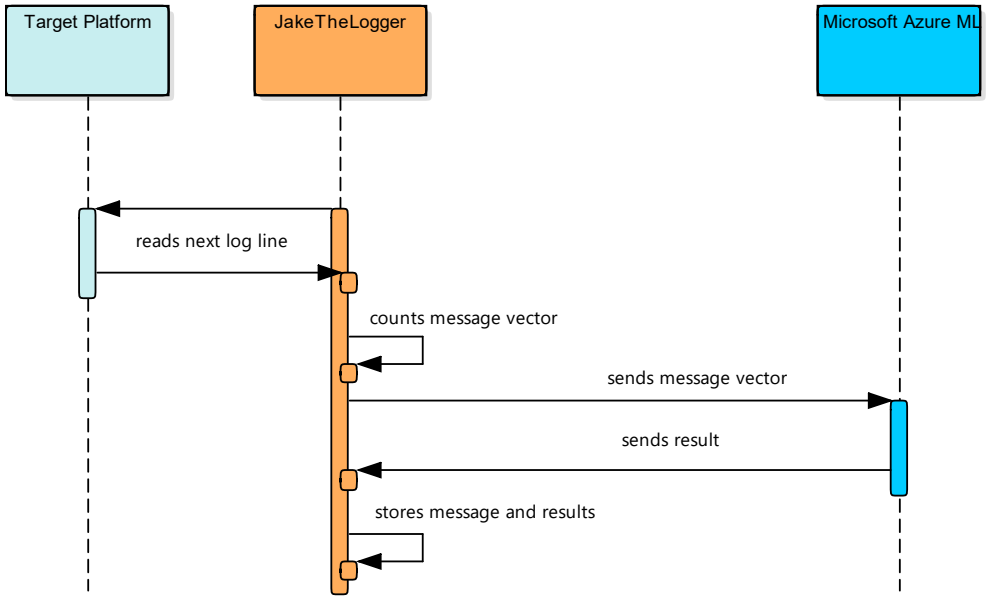
3.5 Čtení dat z logů

Při návrhu zisku jednotlivých požadavků z integrační platformy jsem vycházel z toho, že nová aplikace musí minimálně, či spíše vůbec nezatěžovat Unify [11]. Vzhledem k tomu, že přes integraci proudí veškerý provoz, je sama o sobě dosti vytížená a v případě, že by touto aplikací byl způsoben výpadek došlo by k silnému ztížení veškerých business procesů, což si nelze dovolit.

Unify veškeré požadavky ukládá do logovacích souborů. Některé, převážně rizikové, služby se zároveň ukládají do Oracle databáze. Ale vzhledem k tomu, že nejde o všechny dostupné služby rozhodl jsem se toho nevyužít.

Princip získání dat proudících přes integrační platformu je založen na čtení jednotlivých logovacích souborů. Jako vhodný nástroj jsem vybral Java třídu *Tailer* z dostupné knihovny org.apache.commons.io [51].

Třída *Tailer*, po implementaci listeneru, se chová stejně jako linuxový příkaz tail [52]. Průběžně kontroluje čtený soubor a každou nově zapsanou řádku zpracovává.



Obrázek 3.4: Sekvenční diagram ukazující proces získání a uložení zpráv

Tímto řešením získáváme data z integrační platformy, aniž bychom jí zatěžovali.

Jednou z věcí, které bylo potřeba vyřešit byla situace, kdy třída Tailer během čekání na nový přírůstek logovacího souboru zcela zablokovala program. Situaci jsem vyřešil tím, že procesu, který čte z logu integrační platformy jsem pomocí *ExecutorService* [53] umožnil běžet na pozadí aplikace. Tím jsem neblokoval řízení programu.

3.6 Odeslání dat do Azure

Protože jsou data odesílána do cloudu, předzpracováváme je lokálně. Přímo do Microsoft Azure odesíláme už jen identifikátor zprávy a vypočtený vektor. To vede k tomu, že veškerá data jsou anonymizována.

Po přečtení zprávy z auditového logu Unify je zpráva předzpracována (2.5) a následně je z ní vytvořen vektor (2.6).

Celý proces předzpracování zpráv probíhá v implementované třídě *LogListener*. Po získání dat, jako textového řetězce, jsou uložena do struktury třídy *AuditLogMessage*.

Následuje proces vytvoření vektoru, který je odesílán do MS Azure. Vektor se vytváří dle pravidel uvedených v sekci 2.6 včetně všech procesů předzpracování. Metody pro výpočet TF a IDF jsou implementované ve třídě *WeightsCounterService*.

Vektor samotný je reprezentován jako seznam *Double* čísel.

Pro odeslání dat bylo třeba vytvořit třídu *AzureWebService*. Vytvoření vhodného požadavku na Azure je podmíněno přihlášením se do služby. Proto do hlavičky je přidána *Basic Access authentication* (jednoduché ověření přístupu).

Na požadavek ihned dostaneme synchronní odpověď s výsledkem. Výsledek je zpracován a přiložen k datům načteným z logu.

Ukázka příkladu pro vektor o velikosti 3 je vidět v kódu 3.1.

```
{
        "Inputs": {
                "input1": {
                        "ColumnNames": [
                                "platform-id",
                                "No0",
                                "No1",
                                "No2"
                        ],
                "Values": [
                        [
                                "esb1-123456",
                                "0.012",
                                "0.512",
                                "0.003"
                        ]
                ]
        }
},
        "GlobalParameters": {}
}
```

Kód 3.1: Požadavek na webovou službu Azure ML

3.7 Napojení na Google Charts

Google vystavuje dokumentaci k produktu Google Charts na adrese `https://developers.google.com/chart/`. Samotné grafy jsou generovány pomocí javascriptu. Z knihovny grafů lze vybrat nepřeberné množství různorodých grafů. Obrázek 3.5 ukazuje dva takové grafy.

3.7.1 Rest API

Aby grafy byly generovány s daty z databáze bylo vytvořeno REST API. Toto API je postavené na míru Google Charts. Ovšem zpravidla není problém postavit další aplikaci tak, aby dokázala přijímat data z REST stejně, popřípadě je dokázala samostatně transformovat.

Za tímto účelem vznikl kompletně celý modul aplikace. Tento modul má dovětek *web-services*. Je složen ze dvou tříd:

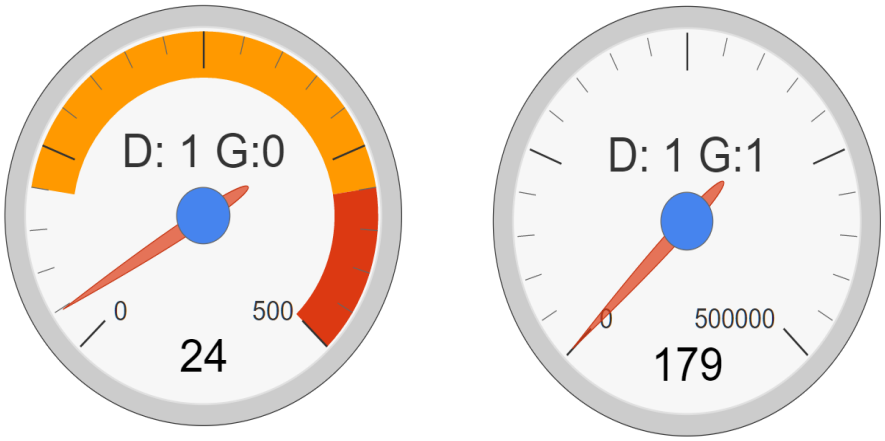
- DataProviderService

- ChartsProviderService

DataProviderService vystavuje webové služby na konkrétních URL adresách. Tím také poskytuje na jednotlivé GET požadavky na zmíněné URL odpovědi s daty z databáze. Jako příklad URL je `http://localhost:18080/jake/rest/data/messages/1/1`. Adresu lze rozdělit na:

- server: *localhost* - to platí pouze pokud jsou skripty s Google Charts spouštěny ze stejného serveru, jinak je nutné doplnit správnou adresu,
- port: *18080* - běžný port pro http je 8080, ale v kapitole 3.1.2 je popsáno, že byl proveden posun portů,
- adresa vystaveného REST API: */jake/rest/data* - na této adrese aplikační server poskytuje REST,
- volaná kolekce: */messages*,
- počet dnů: první číslo za určením kolekce oznamuje počet dnů zpětně od času zavolání chceme filtrovat,
- skupina: druhé číslo určuje skupinu požadavků, pro tento příklad 0 - podezřelé, 1 - korektní.

Data z databáze, ve správné struktuře, vybírá třída ChartsProvderService.
Grafy z Google Charts očekávají konkrétní odpověď v předem určeném formátu JSON.



Počet podezřelých zpráv 31.3.2017 mezi 5:00 - 5:59.

Počet korektních zpráv 31.3.2017 mezi 5:00 - 5:59.

Obrázek 3.5: Počet podezřelých a korektních zpráv prezentován nástrojem Google Charts

Kapitola 4

Analýza a vyhodnocení dat

V této kapitole jsou popsány experimenty, které zkoumali hlavně ideální nastavení algoritmů. Protože v návrhu jsou algoritmy pro shlukování a pro detekci anomálie, je potřeba vybrat pouze jeden, který bude použit.

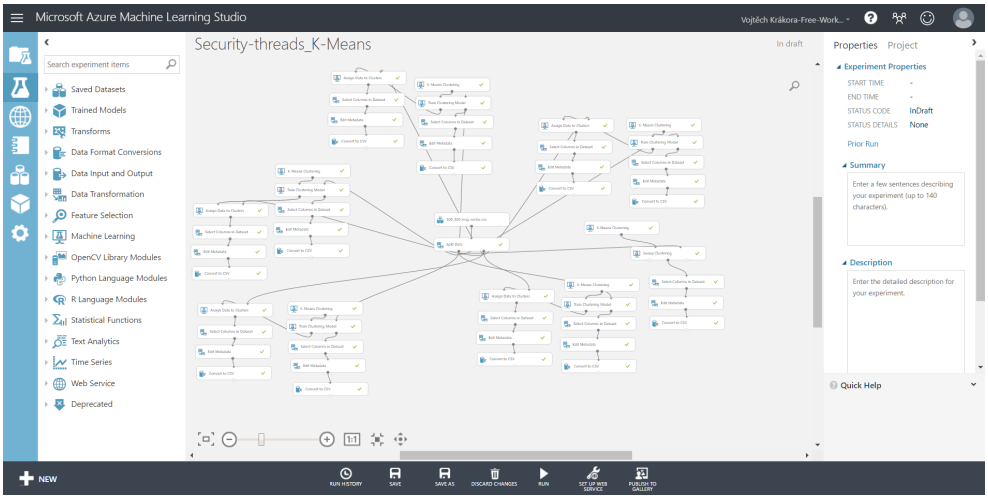
Experimenty byly provedeny v prostředí Microsoft Azure ML. Vstupní data na všech experimentech jsou reálná data z testovacího prostředí Unify ve společnosti CETIN. Data byla převedena na vektory pomocí uvedených metod a přes již implementovanou aplikaci pracovně nazývanou JakeTheLogger. Pro jejich snazší export byla vytvořena nová kolekce, která se rovnala obrazu výstupu. Tato kolekce následně byla exportována do formátu CSV a nahrána do Microsoft Azure ML jako dataset.

Pro experimenty byly použity metody trénovacích modulu popsaných v kapitolách 2.7 a 2.8 s tím rozdílem, že například byly testovány dvě metody najednou. Pro ilustraci poslouží obrázek 4.1. Na obrázku je vidět, že se různé experimenty na shlukování prováděli na stejných vstupních datech.

4.1 Analýza K-Means

V této části je popsán celkový proces vytváření nejlepšího možného nastavení pro shlukovací k-means algoritmus. Jako základ jsem vybral trénovací data. U těchto dat jsem se rozhodl určit 1000 trénovacích vzorků. Protože cílem je detekovat bezpečnostní rizika, a to jak ve formě špatných requestů tak i například podvržených zpráv, vybral z testovacích dat 500 vzorků, které dopadli chybou a 500 vzorků, které proběhli v pořádku. První množina slouží jako reprezentanti skupiny zpráv, které chceme rozpoznávat. Druhá množina 500 prvků je reprezentantem klasických zpráv, které by měli přes integrační platformu proudit.

Pro poměr 50:50 jsem se rozhodl, aby byla dostatečně velká pravděpodobnost, že shluky v k-means najdou správná místa. Pokud bych zvolil poměr, ve kterém by se podezřelé zprávy vyskytovali v malém množství nebo dokonce vůbec, algoritmus k-means by mohl pomocí shluků například rozpoznávat jed-



Obrázek 4.1: Experimentování se shlukováním v Azure ML

notlivé služby nebo najít zcela jiné vzory v datech. Takové vzory by se nemuseli hodit pro analýzu bezpečnostních rizik.

Data jsem rozdělil náhodně poměrem 80:20 na trénovací a testovací. Testovací budou použita k odhadu správnosti shlukování. Vzhledem k tomu, že jsem jednoznačný identifikátor podezřelých zpráv označil, je možné na první pohled dokázat odlišit takovou zprávu od zprávy korektní.

Následující sekce se zabývají jednotlivými možnostmi, jak ovlivnit běh algoritmu k-means. Pro všechny sekce byla použita stejná vstupní data.

4.1.1 Počet shluků

K-means shlukuje data do k shluků vhledem k podobnosti jednotlivých dat. V experimentu bezpečnostních rizik lze očekávat, že by rozdělení mohlo být na dvě skupiny:

- data v pořádku,
- podezřelá data.

I přes tuto myšlenku byl proveden experiment na různá množství shluků. Konkrétně na 2, 3, 4, 10 a 20. Všechny experimenty byly spuštěny současně na totožných vstupních datech. Hodnoty ostatních konfiguračních parametrů (samozřejmě mimo počtu shluků) byly zvoleny následující:

- **výběr prvotních středových bodů:** náhodný,
- **metrika:** eukleidova vzdálenost,
- **počet běhů:** 100.

4.1.1.1 2 shluky

Pro volby dvou shluků vznikly dva, na počet zpráv, téměř shodné shluky. Protože poměr vstupních dat byl 50:50 jde o očekávaný výsledek. Při kontrole konečných přiřazení shluků jsem narazil na zprávu, která by měla být označená jako podezřelá, ale bylo vyhodnocena jako zpráva korektní. V celé množině šlo o jedinou zprávu. Tato zpráva oznamovala, že došlo ke špatnému zadání hesla k jedné ze služeb. Jiná chyba nebyla nalezena. Lze tedy říci, že dochází k rozdělení na podezřelé zprávy a zprávy korektní.

4.1.1.2 3 shluky

Vizualizace výsledků přiřazení clusterů pro $k = 3$ nám ukazuje, že tři vzniklé shluky jsou o velikostech 35, 409 a 355 vzorků. Zde je rozdělení shluků takové, že jeden obsahuje pouze zprávy podezřelé. Další shluk obsahuje pouze zprávy v pořádku a stejně jako v případě dvou shluků i podezřelou zprávu o špatném zadání hesla. Poslední, nejmenší shluk je kombinací. Po detailnějším rozboru vyplynulo, že zprávy v tomto shluku jsou oproti ostatním zprávám kratší. Normalizovaná zpráva se skládá zpravidla z několika slov. Výsledkem rozdělení trénovací množiny byly shluky podezřelých zpráv, korektních zpráv a velmi krátkých zpráv.

4.1.1.3 4 shluky

Čtyři shluky ukázaly pro jejich velikost podobný výsledek jako když $k = 3$. Největší cluster obsahuje zprávy většinu korektních zpráv a již zmiňované krátké chybové zprávy. Druhý shluk obsahuje pouze chyby. Charakteristikou jde spíše o kratší zprávy převážně způsobené komponentou notifikačního enginu. Dále je přítomen shluk, jehož obsahem jsou jen zprávy chybové. Převážně jde o zprávy, které jsou delší a nesou například informaci o tom proč chyba nastala, nikoli pouze upozornění, že nastala. Poslední, obsahově nejmenší, shluk obsahuje pouze několik málo korektních požadavků.

4.1.1.4 10 shluků

Experiment s deseti shluky ukazuje na trend, kdy se k-means přizpůsobuje jednotlivým službám. U dvou shluků došlo ke kolizi korektních a podezřelých požadavků. Oba tyto clustery shlukovali převážně kratší zprávy. U těch je větší pravděpodobnost, že jejich vektor bude převážně nulový. Ostatní shluky obsahují buď chyby nebo správné požadavky, jen se přizpůsobují ke konkrétním službám. Je zde možné například rozpoznávat shluk, který obsahuje chyby hlásící špatný KMS překlad a stejně tak shluk s chybami špatného přihlášení.

4.1.1.5 20 shluků

K-means s $k = 20$ potvrzuje vývoj, který se udál již při $k = 10$. Pouze jeden shluk je ze smíšených korektních zpráv a z podezřelých. Ten obsahuje pouze jeden záznam z podezřelých. Jednotlivé clustery více rozpoznávají jednotlivé služby a chyby. U chyb, jejichž druhů je ve vstupních dat méně než služeb se to projevuje více.

4.1.1.6 Shrnutí

Z celého vývoje dat, která byla použita na testování, plyne, že nejlepší je nechat dva shluky. Tím dokážeme rozlišit podezřelé zprávy od korektních. To zcela naplňuje požadavky detekce bezpečnostních rizik. Další možností je vytvořit shluků více a pokusit se i detekovat různé druhy chyb. Při této volbě bude vyžadováno více starostí s organizováním, které clustery patří do jaké skupiny. Při velkém počtu shluků by mohlo docházet k přeučení, kdy by sice byly rozpoznávány konkrétní chyby a konkrétní služby, ovšem s dalším nasazením nové služby by byly potíže.

4.1.2 Metrika

Pro porovnání různých způsobů měření vzdálenosti jsem zvolil na MS Azure ML eucleidovu vzdálenost (1.5.1.1) a cosinovu vzdálenost (1.5.2). Protože celá analýza ideálního počtu shluků byla založena na eucleidově vzdálenosti, další měření stačí provést už pouze na cosinově vzdálenosti.

Nastavení k-means je následovné:

- **výběr prvotních středových bodů:** náhodný,
- **metrika:** cosinova vzdálenost,
- **počet běhů:** 100.

Při zkoumání výsledků se objevil jev, kdy některé zprávy nebyly přiřazené do žádného clusteru. Po hlubší analýze jsem zjistil, že jde o zprávy, které pochází z B2B brány. Tyto požadavky po normalizaci mají velikost nula. To vede k tomu, že jejich vektor je nulový. S takovým vektorem má cosinova vzdálenost problém. Nelze jej vypočítat, tak zpráva není označena žádným shlukem.

Kromě těchto nulových dat dochází k falešně pozitivnímu označení některých výrazně menších zpráv jako bezpečnostní rizika. Konkrétně pro případ, kde $k = 3$ došlo rozdělení dat žádného shluku (prázdné zprávy), podezřelé zprávy, korektní a další skupinu korektních. V této skupině dominovali zprávy z REST rozhraních. Ty se od ostatních požadavků liší, že nejsou v XML zápisu ale v JSON.

Při náhledu, jak byly zprávy rozděleny do clusterů, při 4 shlucích lze opět pozorovat, že podezřelé zprávy jsou všechny z testovacích dat zahrnuty v jednom shluku. Ostatní shluky jsou složeny už pouze ze zpráv z běžné komunikace. Jejich rozdělení je pak závislé na charakteru zprávy.

Deset shluků také vykazuje dokonalé rozdělení testovacích na rizika a ostatní zprávy. Obě skupiny jsou rozloženy do stejného počtu shluků.

V případě $k = 20$ jsou data opět rozdělena. Ovšem na rozdíl od testovaných $k < 20$ se zde objevují clustery, které obsahují například jen jednu zprávu.

Z výsledků analýzy, kdy porovnáváme metriku euikleidovu a cosinovu je pozorovatelná lepší přesnost u cosinovy délky. Problém zde nastává se zprávami, které mají nulový vektor. Tyto zprávy je třeba ošetřit nebo nahradit normalizační funkci tak, aby nulový vektor negenerovala.

4.1.3 Inicializace

Vliv na běh algoritmu má i počáteční nastavení středů jednotlivých shluků. MS Azure ve svém modulu pro k-means nabízí možnosti [54]:

- **prvních N**: $N \in \mathbb{N}$ prvních vzorků z dat je určeno jako střed shluku,
- **náhodné**: jsou vybrány náhodné body ze vstupních vzorků jako středy,
- **k-means++**: algoritmus definovaný Davidem Arthurem a Sergeiem Vassilvitskii [55],
- **k-means++Fast**: optimalizovaná verze K-Means++,
- **rovnoměrně**: středy jsou rozmístěné ve stejné vzdálenosti v prostoru,
- **dle popisku sloupce**: algoritmus, který na základě popisu sloupce dat rozhoduje o tom, jak budou středy umístěny.

Z experimentu jsem se rozhodl vyřadit K-Means++Fast, jenž je optimalizovanou verzí K-Means++ a inicializaci dle popisku sloupce, neboť jsem nedohledal oficiální dokumentaci k metodě.

Pro experiment jsem zvolil následující nastavení:

- **výběr prvotních středových bodů:** výběr pomocí komponenty Sweep Clustering (hledá nejlepší nastavení shlukování [56]),
- **výběr prvotních středových bodů:** předmětem experimentu,
- **metrika:** eukleidova i cosinova vzdálenost,
- **počet běhů:** 100.

Výsledky experimentu ukazuje tabulka 4.1. V tabulce se vyskytují počty zpráv, které byly v jednotlivých metodách označeny falešně pozitivní (dobrá zpráva označená jako podezřelá) a falešně negativní (podezřelá zpráva označená jako korektní). U všech běhů pro cosinovu metriku bylo vždy 20 zpráv neoznačeno žádným číslem shluku. Jde o nulové vektory, které cosinova vzdálenost nedokáže vypočítat.

Z tabulky (4.1) se jeví nejlepší metoda prvních N. Porovnáme-li výsledky této tabulky s obrázkem 4.2 a 4.3, který ukazuje graf závislosti počtu shluků na konkrétní metodě inicializace (vypočítané pomocí metody sweep clustering) je vidět, že metoda prvních vytváří velký počet shluků. Maximální počet byl stanoven na 30. Až na této hranici se prvních N zastavila. Vzniklo tím mnoho méně obsazených shluků. Jednotlivé zprávy byly rozděleny korektně na podezřelé a nepodezřelé. Velký počet shluků pak vedl i k většímu počtu rozeznaných chyb. Problém by mohl nastat s novou, v trénovacích datech neexistující, chybou. Proto i přes skvělý výsledek neshledávám tuto metodu ideální.

Ve všech ostatních metodách inicializace se při měření euklidovou metrikou objevila jedna stejná podezřelá zpráva, která byla označena jako nepodezřelá. Jde o chybu, která se svého druhu v trénovacích datech objevuje jediná. Celkově to ukazuje na slabost při zjišťování nových chyb.

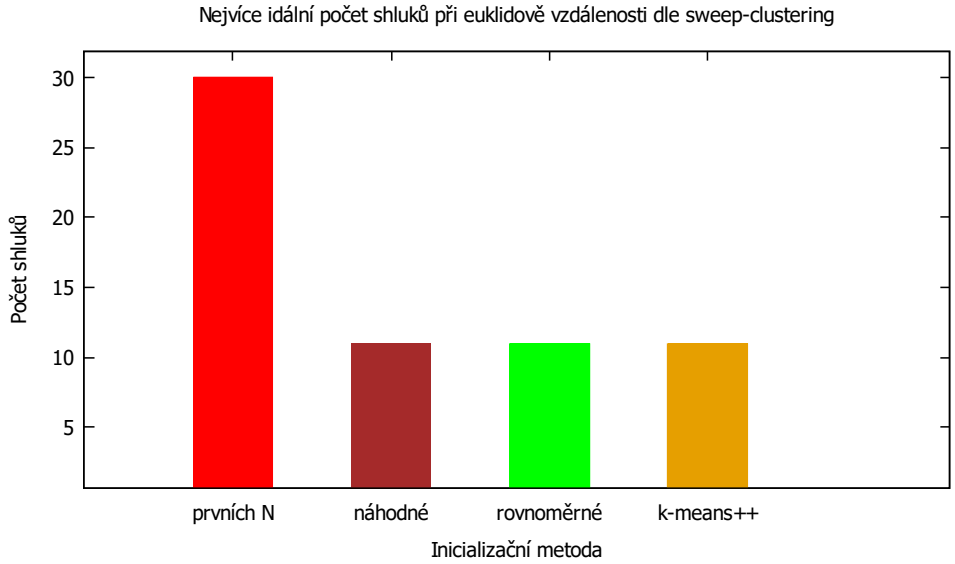
Cosinova metrika si vedla značně lépe, než euklidova. Ve vstupních datech měla jen jeden případ špatného určení. Šlo o případ, kdy korektní zpráva byla označená za chybnou. Při bližším zkoumání jsem zjistil, že jde zprávu, která hlásí nenalezení dokumentu v uložišti. Ve skutečnosti algoritmus našel a označil jako chybu zprávu, která nesla informaci o chybě. Z pohledu administrátora platformy může jít o zajímavou informaci, neboť dotazuje-li se někdo na neexistující soubor jde také o bezpečnostní riziko.

Tabulka 4.1: Zobrazení chyb při shlukování 800 vzorků, při rovnoměrném rozdělení korektních a podezřelých zpráv

	Prvních N		Náhodné		K-Means++		Rovnoměrně	
	FP	FN	FP	FN	FP	FN	FP	FN
Eukleidova metrika	0	0	0	1	0	1	0	1
Cosinova metrika	0	0	0	0	1	0	0	0

4.2 Analýza detekce anomálie

Při detekci anomálie jsem kromě experimentů s ideálním nastavením vytvořil i experiment mezi dvěma metodami. Jedna je detekce anomálie založená na principu PCA, jejíž princip je přiblížen v kapitole 1.6.2. Druhou metodou je One-Class Support Vector Machine, dále bude označována zkratkou SVM. Jde o metodu, která rozděluje n-dimenzionální data lineárně do dvou tříd [57].



Obrázek 4.2: Počty clusterů, které jako ideální vyhodnotila metoda sweep clustering při euklidově vzdálenosti

4.2.1 Popis vstupních dat

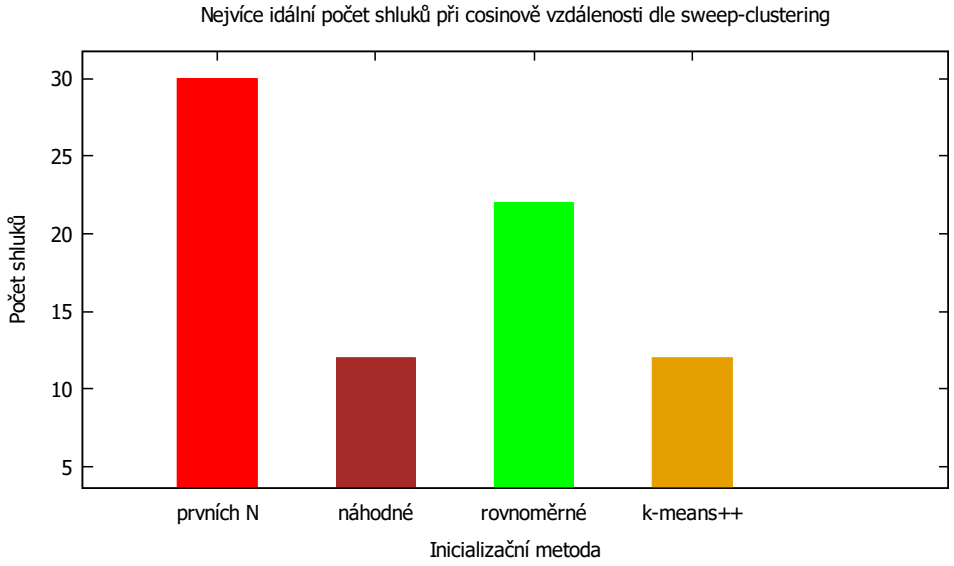
Rozdíl v přístupu mezi shlukováním a detekce anomálie je v trénovacích datech. Při detekování anomálií byla jako testovací data použita pouze data, která byla manuálně určena jako nezávadná. Dále byla připravena i množina dat testovacích. Testovací data obsahují v poměru 87:13 korektní zprávy a podezřelé.

Velikost obou množin je 1000 požadavků. Vstupy byly vybrány náhodným výběrem z téměř 50000 požadavků velké množiny na ESB. Zprávy, které integrační platformou v pořádku prošly byly hodnoceny jako korektní. Zprávy, které vyvolávali chyby byly označené jako podezřelé. Nevýhodou tohoto přístupu může být přehlédnutí rizikové zprávy, která se běžným pohledem může jevit, že je v pořádku.

4.2.2 Ohodnocení klasifikace

Pro dosažení nejlepšího možného výsledku používám MS Azure komponentu nazvanou „Tune Model Hyperparameters.“ Jde o komponentu, která se snaží optimalizovat parametry modelu, tak aby našla ideální nastavení [36]. Na základě toho je nutné být schopen rozeznat, jestli předchozí ohodnocení parametrů je horší než stávající. K tomu poslouží tyto možnosti nastavení [58]:

- **přesnost:** lze vyjádřit vzorcem $\frac{tp+tn}{tp+tn+fp+fn}$,



Obrázek 4.3: Počty clusterů, které jako ideální vyhodnotila metoda sweep clustering při cosinově vzdálenosti

- **správnost:** lze vyjádřit vzorcem $\frac{tp}{tp+fp}$,
- **podíl:** lze vyjádřit vzorcem $\frac{tp}{tp+fn}$,
- **F-míra:** lze vyjádřit vzorcem $2 * \frac{Přesnost * Podíl}{Přesnost + Podíl}$.

Proměnné ve vzorcích jsou zkratky:

- tp - true positive (korektní zprávy označené jako korektní),
- tn - true negative (podezřelé zprávy označené jako podezřelé),
- fp - false positive (podezřelé zprávy označené jako korektní),
- fn - false negative (korektní zprávy označené jako podezřelé).

Experiment s ohodnocením klasifikace na základě přesnosti je zobrazen v tabulce 4.2. Lze pozorovat, že ani při jedné z použitých metod nedošlo k označení podezřelé zprávy ve zprávu korektní. Z hlediska bezpečnosti je lepší prověřovat správné zprávy, než ignorovat podezřelé. Jediný rozdíl je v tom, že SVM hodnotí méně korektních zpráv jako podezřelé. Při bližším pohledu na takové zprávy jde převážně o kratší požadavky.

Po změně hodnotícího vzorce na *správnost* se výsledná data příliš nezměnila. V tabulce 4.3 je vidět, že hodnoty pro SVM zůstali totožné, jako v případě

Tabulka 4.2: Výsledky při ohodnocení pomocí metody *přesnost*

	PCA		SVM	
	P-M: korektní	P-M: podezřelé	P-M: korektní	P-M: podezřelé
Korektní	525	165	656	34
Podezřelé	0	102	0	102

Tabulka 4.3: Výsledky při ohodnocení pomocí metody *správnost*

	PCA		SVM	
	P-M: korektní	P-M: podezřelé	P-M: korektní	P-M: podezřelé
Korektní	511	179	656	34
Podezřelé	0	102	0	102

použití Přesnosti. U PCA došlo k lehkému zhoršení. Toto zhoršení je rovno 14 korektním zprávám, které byly označené jako podezřelé.

Změna hodnocení na Podíl pro výsledek příliš změnu neznamenal (hlavně v porovnání s hodnocením *správnost*). Tabulka 4.4 ukazuje, že jediná změna oproti hodnocení Správnost je o jednu zprávu víc, která byla chybně označena jako podezřelá v případě použití PCA. SVM si stejně jako v předchozích dvou experimentech drží stejné hodnoty.

Tabulka 4.4: Výsledky při ohodnocení pomocí metody *podíl*

	PCA		SVM	
	P-M: korektní	P-M: podezřelé	P-M: korektní	P-M: podezřelé
Korektní	510	180	656	34
Podezřelé	0	102	0	102

Použití *F-míry* vedlo k totožným výsledkům jako experiment s *přesností*. Více ji znázorněno v tabulce 4.5

Tabulka 4.5: Výsledky při ohodnocení pomocí metody *F-míra*

	PCA		SVM	
	P-M: korektní	P-M: podezřelé	P-M: korektní	P-M: podezřelé
Korektní	525	165	656	34
Podezřelé	0	102	0	102

Při porovnání výsledků klasifikace *F-míry* a *přesnosti* došlo ke zcela totožným výsledkům. Obě tyto metody byly z výše uvedených i nejvýše úspěšné z pohledu PCA. SVM nezaznamenalo žádnou změnu na hodnocení klasifikace.

4.3 Shrnutí experimentů

Pro výsledné nasazení jsem se rozhodl využít shlukování. Vzhledem k výsledkům z analytické části se úspěšně vedlo rozdělit data na shluky korektních a podezřelých zpráv.

Určení vhodnější metriky je složité. Rozhodl jsem se využít euklidovu vzdálenost. Důvodem je, že nebude se stávající implementací vznikat ta část zpráv, které není přiřazen žádný shluk.

Inicializaci prvních centroidů pro shluky jsem se rozhodl zvolit metodou k-means++. K tomuto rozhodnutí mě vede konkrétní příklad „korektní“ zprávy, která byla nositelem informace o chybě a byla označena za podezřelou. Toto odhalení se ovšem povedlo při měření vzdáleností pomocí cosinova vzorce. Vzhledem k tomu, že inicializace pro euklidovu vzdálenost měla podobné výsledky, bylo to impulzem pro výběr metody k-means++.

4.4 Nasazení

Nasazení proběhlo na testovací ESB Unify. Pro natrénování modelu na MS Azure ML jsem z logů vzal veškeré korektní požadavky ze dne 5.5.2017 od 10 do 18 hodin. Získal jsem odhadem 14000 požadavků. To samé množství chyb jsem získal z logů za celý rok 2017. Dohromady jsem měl k učení algoritmu přes 28000 požadavků, v poměru 50:50 (korektní:chybné požadavky).

Vektor o velikosti 10 používal následující slova:

Tabulka 4.6: Výběr slov pro tvorbu vektoru při nasazení

četnost výskytu v %	**slovo**
45	exception
47	esb#
23	notificationmessageservice
11	without
12	#/mailhammer/mailhammerservice
14	cancelorderedservice
7	#t#
11	notificationalertjmsservice
27	java
14	apache

Naučený shlukovací algoritmus algoritmus byl konfigurován na dva shluky. Za dobu provozu jednoho dne nebyl nalezen žádný podezřelý požadavek, který by byl označen jako korektní.

Korektní požadavky označené jako podezřelé se v řádu jednotek požadavků vyskytly. Všechny tyto požadavky byly oznámení o chybě. Ovšem cílové systémy je odeslali jako korektní zprávu se statusem 200 (OK). Lze tedy tyto

zprávy označit za bezpečnostní rizika. Pokud by se v logu hledalo na základě statusu zpráv, což je dost běžné, nebyly by tyto rizika odhaleny. Tím se ukazuje význam této aplikace.

4.5 Nedostatky a budoucnost práce

Při experimentech jsem zjistil jeden nedostatek, kvůli kterému jsem zavrhl jinak úspěšnou cosinovu metriku. Problém je, že po normalizaci se u krátkých zpráv může stát, že jejich vektor bude nulový. S tím ovšem nepočítá vzorec pro cosinovu metriku.

Oprava takového nedostatku může například spočívat v tom, že k hodnotám celého výsledného vektoru přičteme konstantu $k \neq 0$. Další možností je úprava normalizační funkce například tak, aby brala v potaz i názvy XML elementů. To ovšem samo o sobě stále nemusí zamezit výskytu nulových vektorů.

Další vhodnou úpravou by bylo detekovat požadavek z logu jako celek, ale nikoliv na základě toho, že se celý vyskytuje na jedné řádce. Z provozu se zjistilo, že některé chyby vypisují své hlášky i se znaky pro označení konce řádky. Stávající algoritmus tak pracuje pouze s první řádkou a ostatní jsou ignorovány. To může zapříčinit, že přicházíme o užitečnou informaci, která by k detekci mohla pomoci. Jednou z možností by mohlo být využití regulárního výrazu.

Práce by šla rozšířit o uživatelské rozhraní. Různé konfigurace, které se musí provádět by se pak mohly dělat jednodušeji. Příkladem takových úprav může být přidání do čtení logů aplikaci, jejíž logy nejsou strukturované tak, jak jsme z Unify zvyklí.

Protože zejména pro shlukování je dobré mít učící data rovnoměrně rozdělená, doporučoval bych jednotlivá detekovaná rizika v databázi uchovávat právě za účelem přeučení algoritmu. Přeučení bych vzhledem k časové nenáročnosti (řádově minuty) doporučoval denně. Minimálně je nutné po každém novém přidání operace nebo služby.

Kapitola 5

Závěr

V rámci této práce jsem vytvořil nástroj, který dokáže detekovat bezpečnostní rizika na integrační platformě Unify. Proces získání dat z Unify je vyřešen tak, aby celou platformu neohrožoval.

Při předzpracování jednotlivých zpráv jsem zjistil, že lepší by bylo použit pro normalizaci metodu, která nebude produkovat nulový vektor. Ovšem i navržená implementace funguje v pořádku, jen při použití cosinovy vzdálenost jsou některé zprávy ve stavu, kdy u nich nelze detekovat, zdali jsou nebo ne rizikem.

Protože je využívaná cloudová služba MS Azure ML, jsou data předzpracovaná před odesláním. V MS Azure ML studiu jsem vytvořil experimenty se shlukováním a detekcí anomálií. Po experimentální části jsem se rozhodl pro shlukování a jeho prediktivní model jsem vystavil jako webovou službu.

Zpracované informace z cloudu se pak ukládají společně s původní zprávou do MongoDB. Díky databázi a REST API je možné k datům přistupovat i systémy třetích stran.

Při testováních na datech z testovacího prostředí jsem ověřil funkčnost a schopnost detekovat i rizika, která vypadala jako běžné zprávy.

Průběžné, či konečné výsledky lze prezentovat pomocí nástroje Google Charts, jehož javascriptové rozhraní jsem pro prezentaci využil.

Literatura

[1] Online. Available from: http://www.businessdictionary.com/definition/information-system.html

[2] Available from: https://www.nbu.cz/cs/pravni-predpisy/1091-zakon-o-kyberneticke-bezpecnosti-a-o-zmene-souvisejicich-zakonu-zakon-o-kyberneticke-bezpecnosti/

[3] ISO 27001. Available from: http://www.rac.cz/rac/homepage.nsf/CZ/BS7799

[4] Bilge, L.; Dumitras, T. Investigating Zero-Day Attacks. Online, 08 2013.

[5] ZETTER, K. Online, 01 2016. Available from: https://www.wired.com/2016/01/hacker-lexicon-what-are-dos-and-ddos-attacks/

[6] Software, S. 11 Best Practices for Peer Code Review. Online, 2011. Available from: http://support.smartbear.com/support/media/resources/cc/11_Best_Practices_for_Peer_Code_Review.pdf

[7] Petukhov, A.; Kozlov, D. Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing. *Computing Systems Lab, Department of Computer Science, Moscow State University*, 2008. Available from: https://www.owasp.org/images/3/3e/OWASP-AppSecEU08-Petukhov.pdf

[8] Corona, I.; Giacinto, G. Detection of Server-side Web Attacks. *Department of Electrical and Electronic Engineering, University of Cagliari, Italy*, 2010.

[9] Ahn, S.-H.; Kim, N.-U.; et al. Big Data Analysis System Concept for Detecting Unknown Attacks. *Department of Electrical and Computer Engineering, Sungkyunkwan University, College of Information and Communication Engineering, Sungkyunkwan University*, 2014.

[10] Rouse, M. security information and event management (SIEM). Online. Available from: http://searchsecurity.techtarget.com/definition/security-information-and-event-management-SIEM

[11] *Unify integration platform*. Available from: https://www.physter.com/unify/

[12] Satzger, G. System Integration in Information Technology - An Intermediation Rather Than a Procurement Task? *University of Augsburg*, 1996. Available from: http://www.fim-rc.de/Paperbibliothek/Veroeffentlicht/027/wi-27.pdf

[13] Available from: http://www.oracle.com/technetwork/java/javaee/tech/javaee6technologies-1955512.html

[14] Schapire, R. Online, 2008, cOS 511: Theoretical Machine Learning. Available from: https://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf

[15] Smola, A.; Vishwanathan, S. *Introduction to Machine Learning*. pub l ished by the press synd icate o f the un ivers ity o f cambr idge, 2008, ISBN ISBN 0 521 82583 0.

[16] Munoz, A. Machine Learning and Optimization. Online. Available from: https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf

[17] Paul, S. K.; Agrawal, M.; et al. An Information Retrieval(IR) Techniques for text Mining on web for Unstructured data. *International Journal of Advanced Research in Computer Science and Software Engineering*, volume 4, 02 2014, ISSN 2277 128X. Available from: https://www.ijarcsse.com/docs/papers/Special_Issue/icadet2014/Lord_18.pdf

[18] Witten, I. H. Text mining. *Computer Science, University of Waikato, Hamilton, New Zealand*, 0. Available from: http://www.cos.ufrj.br/~jano/LinkedDocuments/_papers/aula13/04-IHW-Textmining.pdf

[19] Ghosh, M. S.; Roy, M. S.; et al. A tutorial review on Text Mining Algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, volume 1, 06 2012, ISSN 2278 – 1021. Available from: https://pdfs.semanticscholar.org/5fc6/b674cde1f39847b8783349af200eb68c9d48.pdf

[20] Sankar, K.; Babu, D. G. N. K. S. A Study of Text Mining For Web Information Retrieval System From Textual Databases. *International Journal of Emerging Technology and Advanced Engineering*, volume 3, 12 2013, ISSN 2250-2459.

[21] Ramos, J. Using TF-IDF to Determine Word Relevance in Document Queries. *Department of Computer Science, Rutgers University*, 0. Available from: https://pdfs.semanticscholar.org/b3bf/6373ff41a115197cb5b30e57830c16130c2c.pdf

[22] Vijayarani, D. S.; Ilamathi, M. J.; et al. Preprocessing Techniques for Text Mining - An Overview. *International Journal of Computer Science & Communication Networks*, volume 5, 2016, ISSN 2249-5789. Available from: http://www.ijcscn.com/Documents/Volumes/vol5issue1/ijcscn2015050102.pdf

[23] Zhao, Q.; Bhowmick, S. S. Association Rule Mining: A Survey. Online. Available from: http://www.lsi.upc.edu/~bejar/amlt/material_art/assrules20zhao03association.pdf

[24] Agrawal, R.; Imielinski, T.; et al. Mining Association Rules between Sets of Items in Large Databases. Online. Available from: http://www.almaden.ibm.com/cs/quest/papers/sigmod93.pdf

[25] Tan, P.-N.; Steinbach, M.; et al. *Introduction to Data Mining*. Addison-Wesley, first edition, 2005, ISBN 978-0321321367. Available from: http://www-users.cs.umn.edu/~kumar/dmbook/index.php

[26] Bhusare, B. B.; Bansode, S. M. Centroids Initialization for K-Means Clustering using Improved Pillar Algorithm. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, volume 3, 04 2014. Available from: https://pdfs.semanticscholar.org/b278/74148c4af4d3eedc64909b0b738e5b1c73cf.pdf

[27] E. Garcia, P. Online, 2015. Available from: http://www.minerazzi.com/tutorials/cosine-similarity-tutorial.pdf

[28] Deepthi, A. S.; Rao, D. K. Anomaly Detection Using Principal Component Analysis. *IJCST*, volume 5, 10 2014, ISSN 0976-8491. Available from: http://www.ijcst.com/vol54/1/28-Adathakula-Sree-Deepthi.pdf

[29] Anand, P. R.; Kumar, T. K. PCA Based Anomaly Detection. *International Journal of Research in Advent Technology*, volume 02, 2014, ISSN 2321-9637. Available from: http://www.ijrat.org/downloads/feb-2014/paper20id-222014114.pdf

[30] Breunig, M. M.; Kriegel, H.-P.; et al. LOF: Identifying Density-Based Local Outliers. *Proc. ACM SIGMOD 2000 Int. Conf. On Management of Data, Dalles, TX*, 2000. Available from: http://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf

[31] Abdi, H.; Williams, L. J. 2010. Available from: http://www.utdallas.edu/~herve/abdi-awPCA2010.pdf

[32] Rodriguez, A. RESTful Web services: The basics. Online, 06 2008. Available from: https://www.ibm.com/developerworks/webservices/library/ws-restful/

[33] *Google Charts*. Available from: https://developers.google.com/chart/

[34] Johnson, C. H. High Level Design Distributed Network Traffic Controller. 02 2005. Available from: https://people.ok.ubc.ca/rlawrenc/research/Students/CJ_05_Design.pdf

[35] Available from: https://www.microsoft.com/cs-cz/

[36] Microsoft, . *Microsoft Azure*. Available from: https://azure.microsoft.com/cs-cz/

[37] *Microsoft Azure Machine Learning Studio*. Available from: https://studio.azureml.net/

[38] *DEPLOY ANYWHERE WITH RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM*. Available from: https://www.redhat.com/cms/managed-files/mi-deploy-anywhere-jboss-eap-datasheet-inc0405103lw-201605-en.pdf

[39] JBoss: Aplikační server. Online, 02 2008. Available from: https://www.root.cz/clanky/jboss-aplikacni-server/

[40] *Apache Log4j 2*. Available from: https://logging.apache.org/log4j/2.x/

[41] Sproat, R.; Bedrick, S. CS506/606: Txt Nrmlztn. 2011. Available from: http://www.csee.ogi.edu/~sproatr/Courses/TextNorm/

[42] Li, W. Automatic Log Analysis using Machine Learning. 11 2013. Available from: http://uu.diva-portal.org/smash/get/diva2:667650/FULLTEXT01.pdf

[43] *MongoDB*. Available from: www.mongodb.com

[44] Moniruzzaman, A. B. M.; Hossain, S. A. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application*, volume 6, 04 2013. Available from: https://arxiv.org/ftp/arxiv/papers/1307/1307.0191.pdf

[45] Available from: https://www.mongodb.com/community/licensing

[46] Available from: http://bsonspec.org/

[47] *CETIN*. Available from: https://www.cetin.cz/

[48] Constantine, C. SIEM and Log Management - Everything you need to know but were afraid to ask, Part 1. 2014. Available from: https://www.alienvault.com/blogs/security-essentials/everything-you-wanted-to-know-about-siem-and-log-management-but-were-afraid

[49] Work?, H. D. S. Written by Colton Bachman. 2016. Available from: https://www.integritysrc.com/blog/313-how-does-siem-work

[50] Cross-Origin Resource Sharing. 2014. Available from: https://www.w3.org/TR/cors/

[51] *Tailer class*. Available from: https://commons.apache.org/proper/commons-io/javadocs/api-2.4/org/apache/commons/io/input/Tailer.html

[52] *Tail*. Available from: https://www.gnu.org/software/coreutils/manual/html_node/tail-invocation.html

[53] *Interface ExecutorService*. Available from: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html

[54] Microsoft. K-Means Clustering. Online, 09 2016. Available from: https://msdn.microsoft.com/en-us/library/azure/dn905944.aspx

[55] Arthur, D.; Vassilvitskii, S. k-means++: The Advantages of Careful Seeding. *Stanford*, 2006. Available from: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

[56] Azure, M. Online, 08 2015. Available from: https://msdn.microsoft.com/en-us/library/azure/mt484327.aspx

[57] Sunil, R. Understanding Support Vector Machine algorithm from examples (along with code). Online, 06 2015. Available from: https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/

[58] Powers, D. M. W. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *School of Informatics and Engineering Flinders University of South Australia PO Box 2100, Adelaide 5001, South Australia*, 2007. Available from: http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf

Příloha A

Acronyms

API Application programming interface

B2B Business to business

CORS Cross origin resource sharing

CSV Comma separated valies

DDoS Distributed denial of service

DoS Denial of service

ESB Enterprise service bus

ETL Extract, transform, load

GUI Graphical user interface

HLD High level desing

IDF Inverse document frequency

JSON Javascipt object notation

KMS Key management system

LOF Local outlier factor

ML Machine learning

MS Microsoft

NoSQL Not only structured query language

PC Personal computer

PCA Principal component analysis

REST Representational state transfer

SIEM Security information and event management

SMS Short message service

SQL Structured query language

SVM Support vector machines

TF Term frequency

URL Uniform resource locator

XML Extensible markup language

Příloha B

Instalace lokálního prostředí

Tato kapitola popisuje jak nainstalovat prostředí aplikace JakeTheLogger lokálně.

B.1 Očekávané podmínky

1. **Operační systém:** instalace je připravena pro instalaci na operační systémy založené na Linuxu.

2. **Aplikační server:** je vyžadováno použití přiloženého aplikačního serveru, který je již nakonfigurován.

B.2 Instalace databáze

1. Nainstalujeme MongoDB 3.4.2.

2. V MongoDB vytvoříme databázi „jake-the-logger-test“ příkazem *use jake-the-logger-test*.

3. Vytvoříme kolekci příkazem *db.createCollection('configurations')*.

4. Do kolekce configurations vložíme základní konfiguraci:

```
db.getCollection('configurations').insert(
        {
                "component": "esb",
                "azure-url": "<adresa na experiment v MS
                    Azure ML",
                "azure-api-key":"<api key>",
                "log-file":"<input file>"
        })
```

Kód B.1: Základní konfigurace

B.3 Spuštění aplikačního serveru

1. Aplikační server rozbalíme.
2. Deploymenty aplikace přesuneme do složky v aplikačním serveru: *wildfly/jboss-eap-7.0/standalone/deployments*.
3. Spustíme server příkazem *./wildfly/jboss-eap-7.0/bin/standalone.sh -c standalone-full.xml* .

Příloha C

Návod na experiment

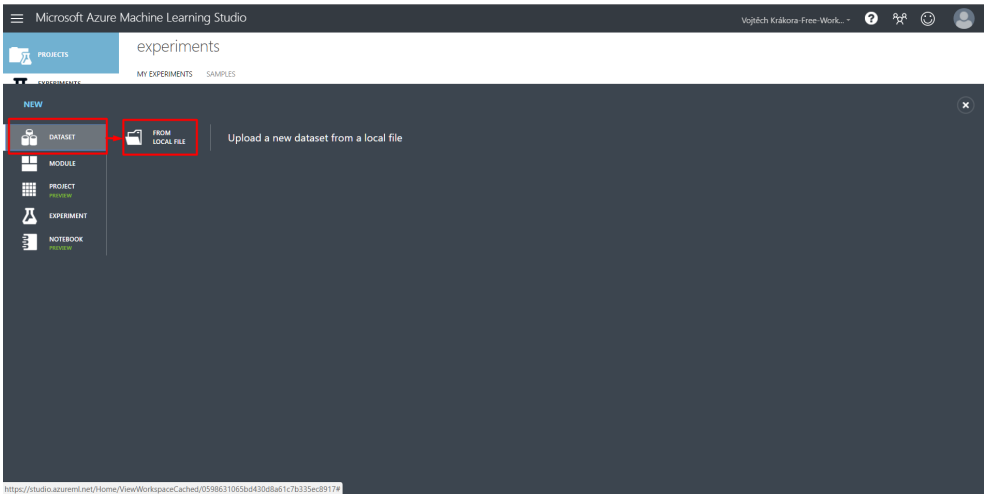
V této kapitole je popsán postup vytvoření modelu v MS Azure ML.

C.1 Nový dataset

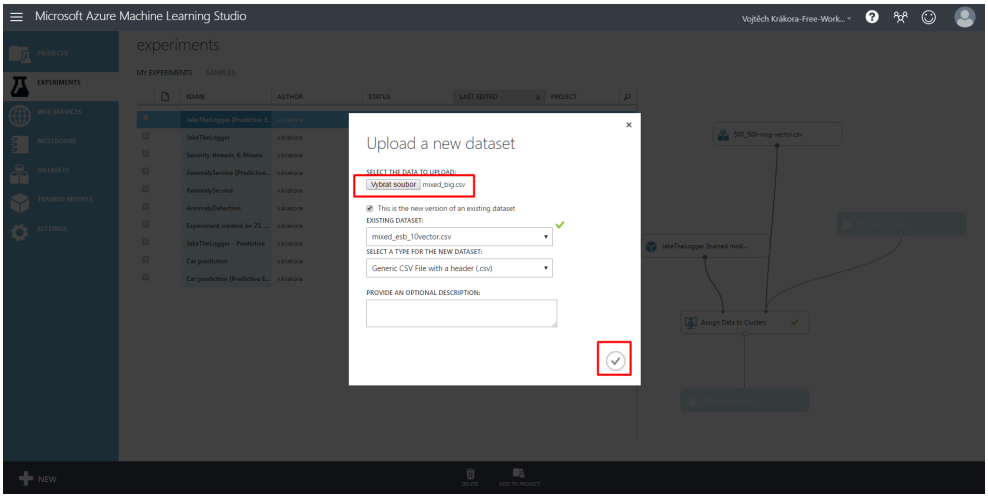
Po registraci a přihlášení se na stránce `https://studio.azureml.net/` vložíme nová data, ze kterých se model bude trénovat. Lze využít přiložená data na disku. Jsou uložená v *data/experimental_input_data/mixed_big.csv*.

1. Postup na obrázku C.1

2. Postup na obrázku C.2



Obrázek C.1: Vytvoření datasetu v MS Azure ML



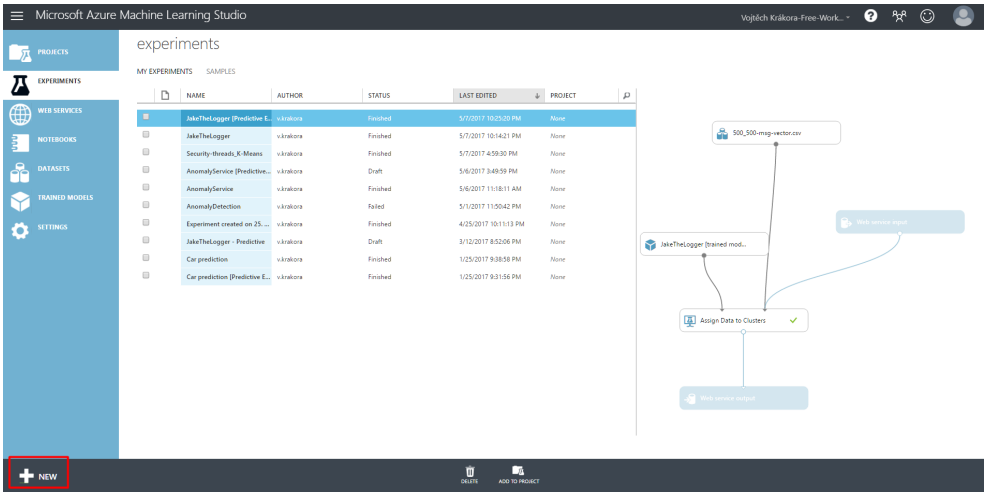
Obrázek C.2: Konfigurace datasetu v MS Azure ML

C.2 Nový experiment

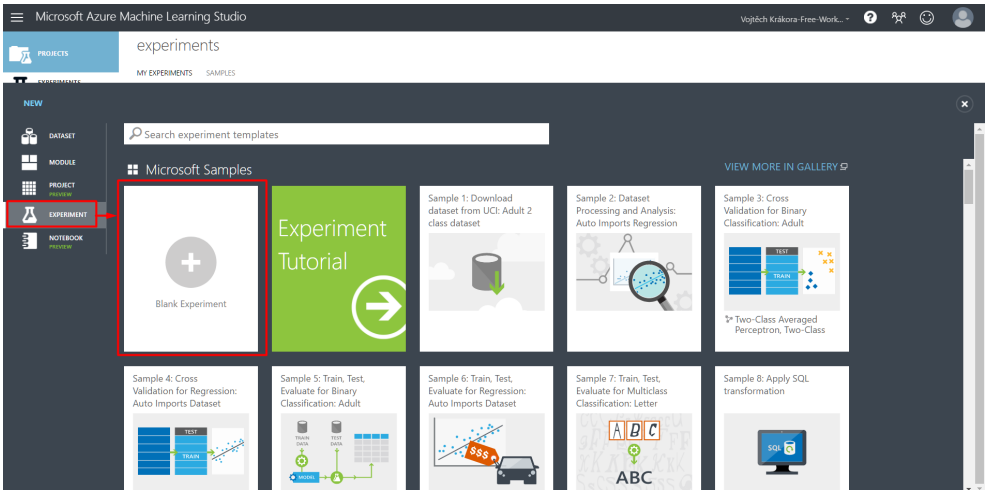
Vytvoříme nový experiment, která vytrénujeme na vložený dataset.

1. Postup na obrázku C.3.
2. Postup na obrázku C.4.
3. Postup na obrázku C.5.
4. Konfigurace **Split Data** je výchozí mimo:
   a) **Fraction of rows in the first output dataset:** 0,8.
5. Konfigurace **K-Means Clustering** je výchozí mimo:
   a) **Initialization:** K-Means++,
   b) **Metric:** Cosine,
   c) **Iterations:** 2500.
6. Konfigurace **Train Clustering Model** je výchozí mimo:
   a) v **Launch column selector** je vybrán sloupec *platform-id*.
7. Konfigurace obou **Select Columns in Dataset** je výchozí mimo:
   a) v **Launch column selector** jsou vybrány sloupce *Assignments,platform-id*.
8. Konfigurace **Assign Data to Clusters** je výchozí.

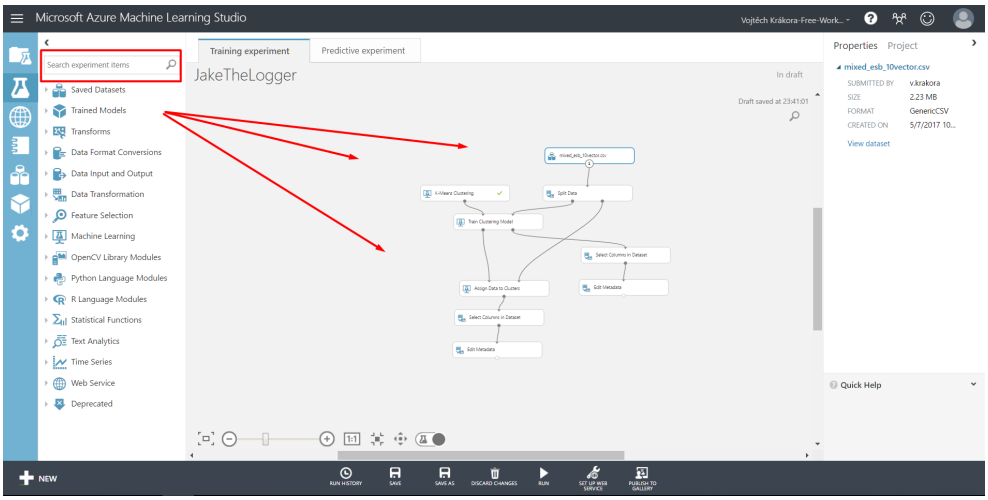
9. Konfigurace obou **Edit Metadata** je výchozí mimo:
   a) v **Launch column selector** jsou vybrány sloupce *Assignments,platform-id*.
10. Postup na obrázku C.6.
11. Postup na obrázku C.7.
12. Postup na obrázku C.8.
13. Postup na obrázku C.9.
14. Postup na obrázku C.10.



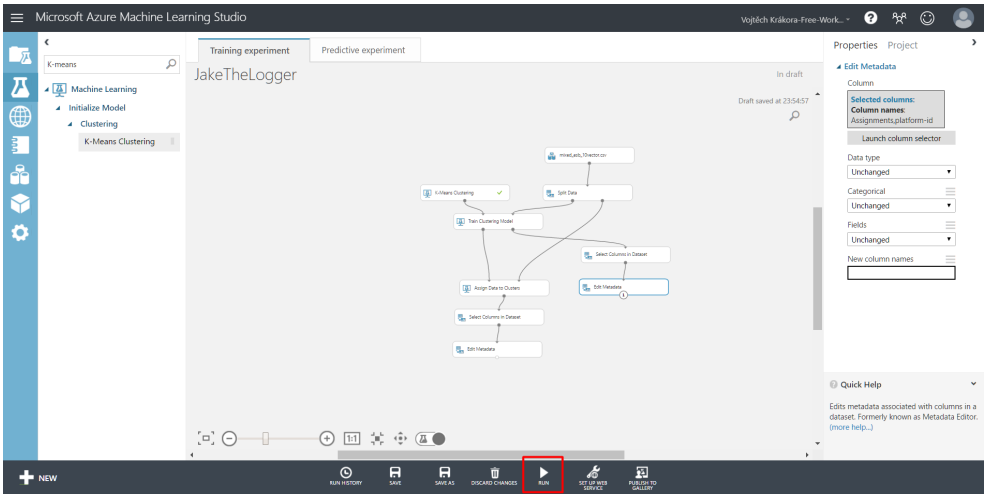
Obrázek C.3: Vytvoření nového experimentu v MS Azure ML



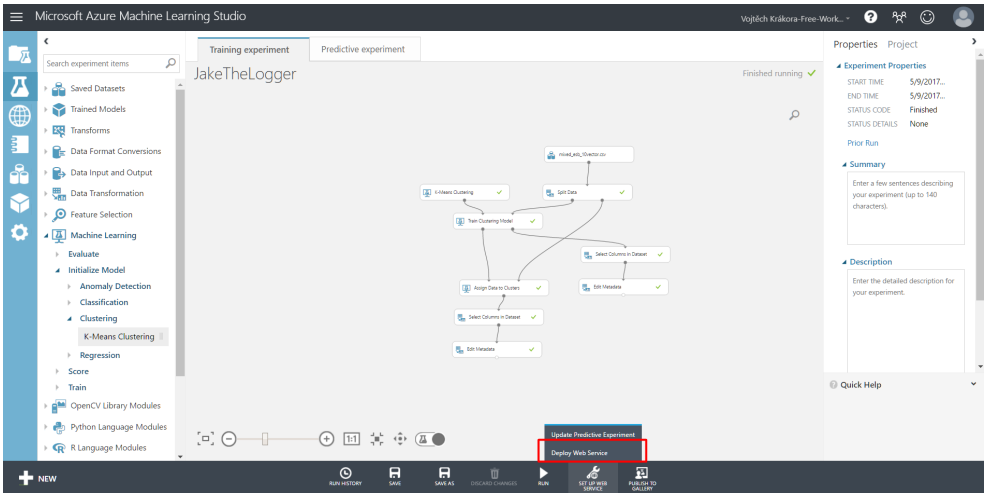
Obrázek C.4: Vytvoření nového experimentu v MS Azure ML II



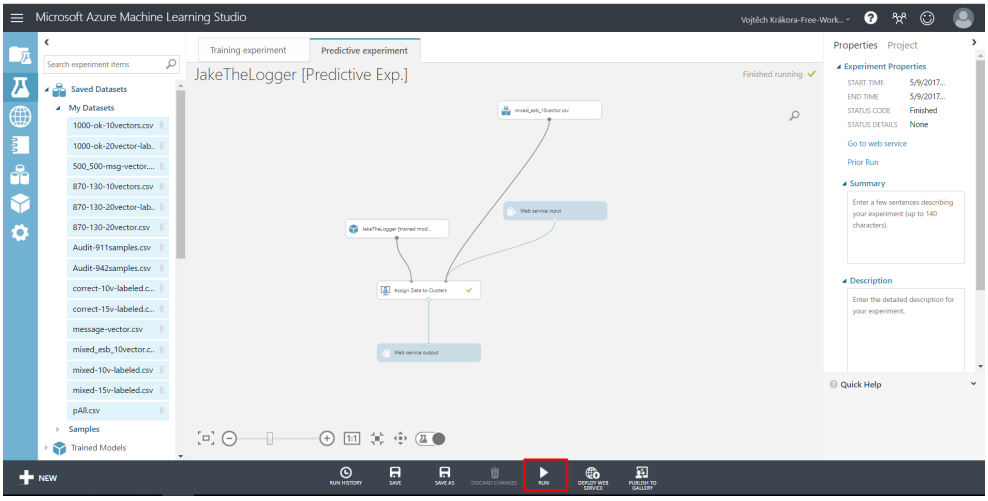
Obrázek C.5: Komponenty experimentu v MS Azure ML



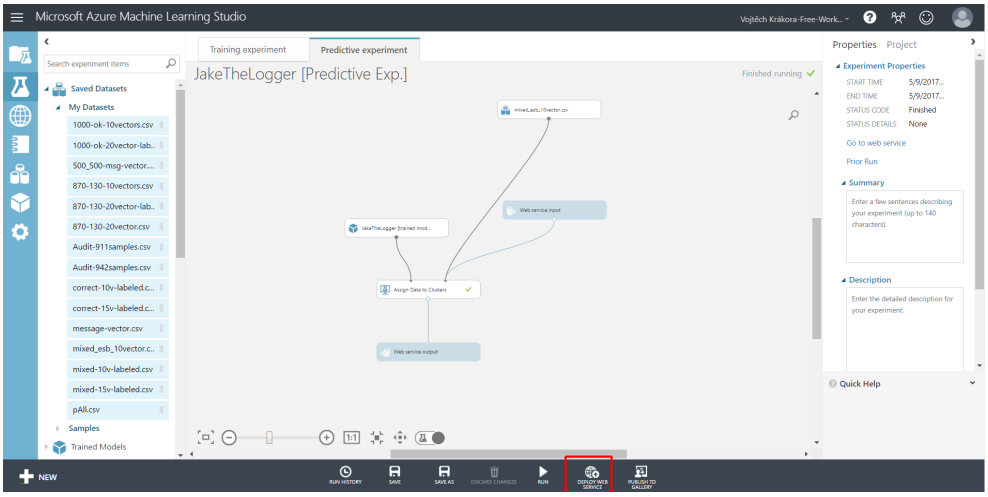
Obrázek C.6: Spuštění experimentu v MS Azure ML



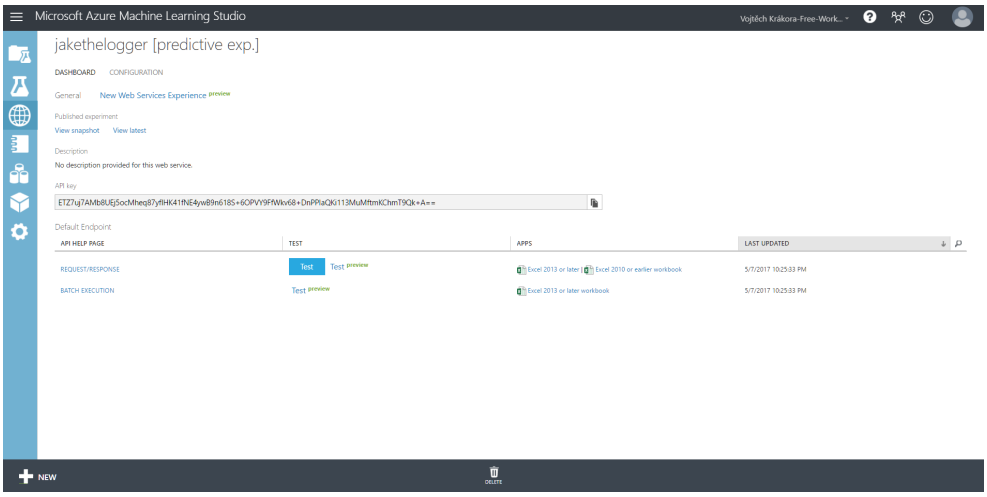
Obrázek C.7: Vytvoření webové služby v MS Azure ML



Obrázek C.8: Spuštění webové služby v MS Azure ML



Obrázek C.9: Deploy webové služby v MS Azure ML



Obrázek C.10: Informace k webové službě v MS Azure ML

Příloha D

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `data` ..... experimental data
- `deployments` ..... jar and war files ready for deployment
- `src` ..... sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
  - `app` ..... the directory of source codes of the application
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
- `wildfly-jake-the-logger-SDK.zip` . application server with all modules