FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Indikátor sentimentu trhu
Student:	David Lebl
Vedoucí:	Ing. Stanislav Kuznetsov
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je návrh, implementace a nasazení škálovatelné aplikace analyzující textový sentiment ze sociálních medií či zpravodajů na téma kryptoměn.
Indikátor sentimentu trhu (IST)
1. Seznamte se s problematikou vývoje aplikací v kategorii Big Data, real-time analýzou textového sentimentu, způsoby vystavení a zabezpečení přístupu k datům, a nasazení aplikace v cloudové službě.
2. Proveďte analýzu potřeb sentiment indikátoru pro koncové uživatele a navrhněte vhodné řešení.
3. Navrhněte a implementujte aplikaci založenou na architektuře microservices.
4. Řešení dokumentujte a nasaďte do produkčního prostředí.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 17. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Indikátor sentimentu trhu

David Lebl

Katedra softwarového inženýrství

Vedoucí práce: Ing. Stanislav Kuznetsov

15. května 2019

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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2019 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Lebl, David. *Indikátor sentimentu trhu*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Cíl této práce je návrh, implementace a nasazení aplikace analyzující sentiment příspěvků ze sociální sítě Twitter na téma kryptoměn a obchodování. Teoretická část práce se zabývá rešerší big data architektury pro real-time analýzu streamů. V analytické a implementační části je řešena volba zvolených technologií pro danou problematiku, implementace dílčích aplikací architektury microservices a způsob nasazení aplikace do cloudového prostředí Kubernetes. Výsledkem práce je monitorovací systém určený jako služba pro retail investory.

Klíčová slova analýza sentiment, kryptoměny, stream processing, big data, microservices, Docker, Kubernetes, cloud

Abstract

The goal of this work is to design, implement and deploy an application that analyzes the sentiment of Twitter posts focusing on cryptocurrency and trading. The theoretical part of the thesis deals with research of big data architecture for real-time stream analysis. The analytical and implementation part deals with the choice of selected technologies, implementation of partial applications of the microservices architecture and the way of deploying the application to the Kubernetes cloud environment. The result of the work is a monitoring system intended as a service for retail investors.

Keywords sentiment analysis, cryptocurrencis, stream processing, big data, microservices, Docker, Kubernetes, cloud

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

V roce 2017 vzrostl velký zájem v investování do kryptoměn, především díky snadné dostupnosti a velkému mediálnímu ohlasu. Rozruch tak vznikl i na sociálních sítích, kde lidé sdílí své názory, investice, emoce a novinky, čímž vytváří veřejný sentiment pro daný trh. Velcí investoři (Market Maker) jsou schopni využít těchto dat, a tím přizpůsobit cenu pro jejich prospěch. Z těchto důvodů se dá předpokládat funkčnost tržního sentimentu jako indikátoru ceny, a proto by bylo zajímavé nabídnout aplikaci vyhodnocující sentiment jako produkt pro retail investory.

Výsledek práce bude nabízen jako produkt pro retail investory, který jim pomůže lépe zanalyzovat investice a pomůže k lepším ziskům. Vedlejším přínosem práce pak bude navržená architektura microservices jakožto způsob real-time analyzování sentimentu z frekventovaných sociálních medií.

Téma jsem si zvolil z osobního zájmu o kryptoměny a obchodování s nimi na burze. Protože dosavadní existující produkty nebyly dostatečně sofistikované a důvěryhodné, rozhodl jsem se pro tento projekt se zaměřením na studovaný obor softwarového inženýrství. Práce vychází a částečně navazuje na softwarový týmový projekt BI-SP1 a BI-SP2, kde jsem se s podobným tématem seznámil a pracoval na něm.

V práci se zabývám analýzou, návrhem, implementací a nasazením aplikace postavené na architektuře microservisis a big data stream analytics, s důrazem zaměřeným na modularitu, flexibilitu a škálovatelnost architektury. Naopak práce neřeší jak přistupovat k analýze textového sentimentu, ani jej nebude v hlubší míře implementovat.

Jako první si v práci analyzujeme systémové požadavky, možnosti architekturu pro real-time analýzu a způsoby vyhodnocení textového sentimentu. Poté

zdůvodníme použité technologie, implementujeme je a vyhodnotíme jejich přínos. Nakonec si ukážeme možnosti nasazení v cloudové platformě.

Kapitola 1

Cíl práce

Cílem práce je návrh, implementace a nasazení škálovatelné aplikace analyzující textový sentiment ze sociálních medií či zpravodajů na téma kryptoměn – Indikátor sentimentu trhu (IST).

Cílem teoretické části práce je seznámení se s problematikou vývoje aplikací v kategorii big data, real-time analýzou textového sentimentu, způsoby vystavení a zabezpečení přístupu k datům.

Cílem analytické části je provést analýzu potřeb sentiment indikátoru pro koncové uživatele a navrhnout vhodné řešení architektury.

Cílem praktické/implementační části práce je návrh a implementace aplikace založené na architektuře microservices. Výsledné řešení bude nasazeno do cloudového produkčního prostředí a zdokumentováno.

Kapitola 2

Rešerše

2.1 Zavedení pojmů

2.1.1 Microservices

Mikroservices, neboli Microservices Architektura je způsob vývoje jednotlivých aplikací jakožto struktury malých servises postavených okolo business domény, kde jednotlivé aplikace jsou navzájem na sobě nezávisle a každá z nich by měla představovat jeden business model (např. product-service, payment-service). [1, 2] Tato architektura na rozdíl od tradiční monolitické přináší výhody v flexibilitě, spolehlivosti, škálovatelnosti, „zero downtime deploymentu“ a komplexnosti systému.

2.1.2 Big Data

Big Data je buzzword, který nemá konkrétní definici, ale lze ho chápat jako metodu přístupu k velkému množství strukturovaným a nestrukturovaným datům jinak, než s použitím standardních zavedených nástrojů (např. porovnání NoSQL a SQL relačních databází.) [3]

2.1.2.1 NoSQL

NoSQL (Not-only-SQL) je označení pro databáze, jiné než tradiční relační (RDBMS), které jsou navrženy pro big data. Z většiny se jedná o distribuované systémy umožňující horizontálně škálovat.

Typy NoSQL databázi lze v základu rozdělit podle datového modelu na:

- key-value,
- document-base,

- column-base,
- graph-base.

2.1.2.2 CAP teorém

CAP teorém je pomůcka pro návrh distribuovaných systémů upozorňující na kompromisy při volbě vlastností distribuovaného datového systému. Dojde-li k výpadku části systému lze zaručit pouze konzistence, nebo dostupnost. [4]

Dle [5, 6], distribuované systémy mohou garantovat podporu nejvýše dvou ze tří následujících „CAP“ vlastností:

C – Consistency Konzistence garantuje, že každý uzel v clusteru při dotazování vrací stejné a nejaktuálnější výsledky/záznamy.

A - Availability Dostupnost garantuje, že na každý požadavek čtení a zápisu je odpovězeno. Každý bežící uzel musí být schopný odpovědět v rozumném čase. (Určuje dostupnost a latenci systému)

P – Partition Tolarance Určuje jestli systém bude fungovat v případě výpadku části systému a bude schopen obnovit danou část.

Ve skutečnosti v *distribuovaných systémech* se jedná pouze o 1 ze 2 voleb, a to mezi A a C, kde vlastnost P je bráno jako povinnost distribuovaného systému:

CP = C Distribuovaný systém je vždy konzistentní, ale v případě výpadku nemusí být vždy dostupný, garantování konzistence je preferováno oproti udržení okamžité dostupnosti/latence. Konzistence je dosažena aktualizováním uzlů před tím než je dovoleno čtení.

AP = A Distribuovaný systém je vždy dostupný, ale v případě výpadku nějakého z uzlů nejsou zaručena konzistentní/aktuální data. Dostupnost je dosažena replikováním dat napříč uzly.

CA Systém který je vždy konzistentní a dostupný. Často jako single DB uzel, což ale není distribuovaný systém a tedy nespadá do CAP teorému.

2.1.3 Container

Cointainer (kontejner), je software který zabaluje kód/aplikaci se všemi závislostmi, tak aby daná aplikace mohla běžet rychle a spolehlivě v kterémkoliv výpočetním prostředí. [7] Ve zkratce se jedná o odlehčenou verzi virtualizace, která přímo využívá jádro OS a tím je mnohem rychlejší jak plnohodnotná virtualizace celého OS. To pak přináší výhody v rychlosti, velikosti prostoru na disku a image, a schopnosti virtualizace většího množství aplikací.

2.2 Požadavky na architekturu

Aplikace vyhodnocující sentiment, zejména ze sociálních medií typu Twitter, musí stíhat zpracovávat velké množství zpráv v rozsahu vetším než je schopný zpracovat běžný server, ve špičkách lze předpokládat až tisíce vyfiltrovaných tweetů za minutu. Dále textový sentiment musí být zpracovaný nejlépe v reálném čase, tedy jako „near“ real-time stream do několika sekund. Z těchto důvodů nelze použít tradiční řešení jednoho výpočetního serveru a musí být použity technologie „big data“ podporující horizontální škálování.

Architektura aplikace musí být také odolná proti výpadku streamu a extrémním zátěžovým špičkám. Tyto problémy se obvykle řeší distribuovanou frontou, load-balancningem a mnoha dalšími cloud-base metodami.

2.3 Řešení architektury pro real-time analýzu

Problematika real-time analýzy streamů spadá do řešení cloudové architektury, které je mnohdy nabízeno jako kompletní platforma od cloudových poskytovatelů jako jsou Google, IBM, Amazon a další. Existuje ale i velká řada open-source nástrojů, které dílčí části dané problematiky implementují a jsou běžně používány i v některých komerčních cloudových řešeních.

2.3.1 Open-source nástroje pro real-time analýzu

Processing engines, ML, AI

- Apache Storm
- Apache Flink
- Apache Spark – Spark Streaming, Spark MLLib, Spark ML
- TensorFlow

Message brokers

- Kafka
- RabbitMQ

Databáze

- Cassandra
- ElasticSearch

A další open source a enterprise nástroje podle [8, 9, 10].

2.4 Existující řešení

Podíváme-li se po kompletním hotovém řešení analyzující sentiment kryptoměn z twitteru a dalších sociálních medií (Reddit, Facebook), najdeme několik veřejně přístupných i placených služeb. Většina těchto služeb je ale velmi primitivní, příliš obecná, a nebo také nedokážeme ověřit její úspěšnost a budoucí spolehlivost.

Seznam kompletních řešení:

1. Social Market Analytics (SMA) [11, 12]
2. iSentium [13]
3. Fear and Greed Index [14]
4. CoinGossip.io [15]

2.4.1 Existující řešení pro vyhodnocování textového sentimentu

Pro vyhodnocování samotného textového sentimentu lze použít jednoduchý slovník s ohodnocenými slovy, a nebo také je možno se obrátit na open-source knihovny využívající pokročilých metod strojového učení (ML) a umělé inteligence (AI).

Metody a implementace vyhodnocující textový sentiment:

1. Ohodnocený slovník slov a emotikonů z open datasetu.
2. Stanford CoreNPL [16].
3. Twitter TensorFlow-CNN [17].
4. a další [18].

2.5 Přístup k datům a zabezpečení

Application Programming Interface (API) je rozhraní nabízející například sadu příkazu, funkcionalit, tříd a protokolů knihoven, který slouží programátorům pro vytváření softwaru nebo integrace s externími systémy. Web API je pak způsob integrace s webovými systémy, většinou pomocí standardizovaného HTTP protokolu, sloužící k dotazování nad datovými zdroji.

Pro Web API – integraci s externími systémy existují tři nejčastěji používané základní technologie: REST, WebSokets a SSE, které definují způsob webové komunikace mezi aplikačním klientem a serverem.

2.5.0.1 REST

Representational State Transfer (REST) je architektonický návrh, pro webové systémy, který definuje jak webové standardy, typu HTTP a URI, by měly být použity. [19]

Hlavní myšlenkou RESTfull architektury je bezstavová komunikace mezi oddělenými nezávislými systémy typu client–server. Bezstavová komunikace je docílena jednoznačným identifikováním dotazovaného zdroje pomocí URI, kde pro základní způsob dotazování jsou použity metody: GET, PUT, POST a DELETE.

REST architektura je primárně navržena pro synchronní komunikaci (request–response) přes HTTP protokol a běžně používaná jako standardní technologie pro návrh API a integraci systémů.

2.5.0.2 WebSokets

WebSokets je komunikační protokol, součástí HTML5 specifikace, který podporuje obousměrnou komunikaci (full-duplex) v rámci jedno TCP spojení mezi klientem a serverem. [20]

Obousměrná komunikace a použití TCP protokolu (na HTTP portu 80/443) umožňuje real-time přenos dat mezi klientem / webovým prohlížečem a serverem, a to bez omezení na firewall povolující HTTP. Komunikace je vytvořena pomocí otevřeného spojení, které vznikne po „handshake“ požadavku od klienta i serveru, což vede k nízké latenci spojení s minimálními výpočetním náklady na režii.

2.5.0.3 Server-sent events

Server-sent events (SSE), nebo také EventSource API, je webový standard součástí HTML5 specifikace, který umožňuje ze serveru streamovat textová data ke klientovi přes klasický HTTP protokol. [21]

Princip této technologie spočítá v otevřeném nepřerušeném „long-live“ HTTP spojení, které umožňuje automatickém odebírání data bez nutnosti „pullingu“ způsobem request-response, čímž je dosaženo vysoké efektivity a nízké latence přípojení.

Rozdíl a nevýhodou oproti technologii WebSokets je v jednosměrné komunikaci (mono-directional) což znamená, že v rámci jednoho spojení je možné

posílat data například pouze ze serveru ke klientovi. Naopak SSE lépe podporuje základní webové komunikační funkce díky použití klasického HTTP protokol.

Tato technologie je pak využívána pro „server push“ ve webovém prohlížeči a pro „real-time data streaming“. Mezi konkrétní use case například patří publikování tržních cen, sociálních příspěvků a chatu, herního skóre a další.

2.5.1 Zabezpečení

Všechny zmíněné technologie (REST, WebSokets, SSE) podporují šifrovanou komunikaci. Pro zabezpečení API je docíleno pomocí autentizaci a autorizaci.

OAuth2 je open standart navrhnuty pro HTTP komunikaci a běžně používaný pro garantování přístupu uživateli bez nutnosti sdílení hesla. Princip této technologie spočívá ve vytvoření „access tokenu“ autorizačním serverem, který je poskytnut klientovi pro přístup při dotazovaní na zabezpečený server.

Kapitola 3

Analýza a návrh

Tato kapitola se zabývá analýzou vhodného řešení pro real-time vyhodnocování sentimentu ze sociální sítě Twitter. Kromě způsobů samotného vyhodnocování sentimentu je v kapitole probrána volba vhodných technologii pro dílčí problematiku a také návrh samotné architektury aplikace.

3.1 Analýza požadavků

Aplikace je určená pro retail/neprofesionální investory, jakožto alternativní a doplňující způsob analyzování budoucího vývoje ceny na základě sociálního/tržního sentimentu.

Funkční požadavky

1. Vyhodnocování textového sentimentu.
2. Real-time analýza sentimentu jednotlivých kryptoměn z sociální sítě Twitter.
3. Zobrazení a přístup k analyzovaným datům.
4. Monitoring nasazených aplikací a streamů.

Nefunkční požadavky

Nefunkční požadavky z většiny vychází ze zadání práce a rešerše architektury pro real-time analýzu 2.3, s požadavky na vysokou náročnost výpočetního systému a možnost rozšířitelnosti v navazující práci. Dílčími nefunkčními požadavky jsou:

1. Architektura microservices
2. Škálovatelnost, modularita, flexibilita
3. Dokumentace k deploymentu

3.2 Zdroje sentimentu

3.2.1 Twitter API

Twitter API umožňuje odebírání příspěvků pomocí Streaming HTTP protokolu (Server-Sent-Event SSE) pro posílání dat ke klientovi přes otevřené připojení, což zaručuje nízkou latenci například oproti jednotlivému dotazování ze strany klienta. Pro vytvoření připojení je potřeba nejprve zaslat autorizovaný požadavek (OAuth) a konfiguraci pro požadovaný stream.

Pro odebírání příspěvku je tedu nutná registrace a výběr služby, na které je potom vygenerován API klíč/token pro odebírání, prohledávání a sdílení příspěvku.

Twitter API aktuálně nabízí tři druhy služeb pro odebírání a vyhledávání příspěvků:

Standard (free) Bezplatná služba nabízející odebírání real-time příspěvků podle klíčových slov, vybraných uživatelů a lokace s omezením na 400 klíčových slov a 5 000 uživatelů (API: statuses/filter). Dále také umožnuje prohledávat 7 dní staré tweety.

Premium Částečně placená služba rozšiřující Standart API o možnost kompletního prohledávání Tweetů do minulosti.

Enterprise Placená služba nabízející kompletní vyhledávání a filtrování tweet podle veškerých operací (např. klíčová slova, emoji, totožné fráze, *symboly*, …). (API: PowerTrack)

V této práci volíme pouze základní službu Standard, která pro většinu účelů bude stačt. Pro filtrování tweetů si navrhneme vlastní aplikaci, která bude odfiltrovávat tweety především od botů a podvodných uživatelů a nabídne stejné možnosti jako Enterprise verze.

Avšak služba Standard nenabízí sledování symbolů (cashtag: `$`, `$BTC`), které jsou pro sledování kryptoměn relativně důležité, tedy v ideálním případě by se měla zvolit varianta Enterprise s PowerTrack API.

Více v kompletním přehledu operátorů pro Standardní [22] a Prémiové [23] funkce.

Model

Streamované příspěvky z Twitter API jsou ve formátu JSON a v případe odběru/streamu přichází vždy jako jeden příspěvek. Kromě textu a metadat příspěvku najdeme v JSON objektu také kompletní informace o uživateli, díky kterým je možno jednoduše odfiltrovat podvodné a nevýznamné uživatele.

Samotný JSON objekt nenese přímou informaci o typu příspěvku (např. příspěvek, komentář, …), ale na základě použitých a nepoužitých atributů, podle Twitter dokumentace [24], lze příspěvky rozdělit na:

Tweet Obyčejný příspěvek obsahující text, uživatele, zmíněné uživatele, media, odkazy, a další metadata…

Retweet Sdílený příspěvek (Tweet) obsahující původní sdílený Tweet.

Quote (citace) stejně jako Retweet, ale obsahuje navíc nový komentář od sdílejícího uživatele.

Replay Odpověd (Tweet), pouze s odkazem na odpovídající Tweet.

3.3 Sentiment indikátor

Přínosem pro uživatele jakož to retail traider by měla být možnost analyzovat získaná data podobně jako při obchodování na burze, kde traider kromě aktuální ceny má také k dispozici historický vývoj ceny (OHCL), množství provedených obchodů (volume) a další…

V případě tržního sentimentu je možné získat podobná data a zobrazovat uživateli víc než jen hodnotu aktuálního vyhodnoceného sentimentu, ale také vývoj hodnoty sentimentu s porovnání s cenou, množství zanalyzovaných příspěvků, sdílení a komentářů a aktuální trend pro dané kryptoměny.

Sentiment tedy lze rozdělit do následujících kategorii měření:

1. aktivita uživatelů a trendující měny,
2. textový sentiment uživatelů a zpravodajců,
3. novinky a události,
4. trading signály.

V této práci se především budeme zabývat prvními třech kategoriemi pro indikování tržního sentimentu příslušné kryptoměny. Trading signály nejsou natolik důležité a může být o něco těžší je analyzovat než ostatní kategorie.

3.3.1 Aktivita – Volume

Aktivita uživatelů Twittru, i frekventovanost publikací zpravodajských kanálů, v sentimentu indikuje zájem o danou kryptoměnu, jak v pozitivním i negativním slova smyslu. Máme-li trendující kryptoměnu, lze předpokládat i zvýšený zájem a počet obchodu na burze, což v podstatě reprezentuje předpověď budoucího „volume“ na burze.

Aplikace analyzující aktivitu by měla umět zpracovat veškeré množství odebíraných zpráv, tak aby byly rychle a nenáročně vyhodnoceny. Aktivitu (volume sentimentu) lze uživateli jednoduše zobrazovat jako poměr součtů přijatých tweetu vůči retweetům, quote a odpovědím. Příslušnou kryptoměnu z kontextu zprávy rychle identifikujeme pomocí hashtagu, symbolu či případně i názvu měny v textu a frekventovaností identifikovaných měn pak dostáváme aktuální trend. Případné sofistikovanější řešení pro vyhodnocování kontextu zprávy, např. Stanford CoreNPL, by bylo použito až po odfiltrování nezajímavých uživatelů. Tímto řešením jednoduchého monitoringu výrazně snížíme nároky na výkon a příliš tím neztratíme přesnost vyhodnocovaného sentimentu.

Aktivitu dále lze kategorizovat podle typu uživatele, opět s relativně rychlým způsobem identifikování (user-enrichment-service):

Běžný uživatel je možné vyfiltrovat podle informace o počtu celkových příspěvků, sledovaných uživatelů, a také v meta-datech tweetu lze nalézt user-agenta typu iOS, Android a Web.

Zpravodajská media ideálně identifikujeme pomocí vlastní ručně vedené databáze věrohodných zpravodajů.

Traideři většinu používají v příspěvku symboly. Dále by se daly identifikovat podle dialektu použitých slov. (Tato kategorie není zas tolik důležitá, a z většiny bude stačit i malý vzorek tweetu s použitými symboly pro identifikaci.)

Boti a podvodní uživatelé lze poznat a filtrovat podle velkého množství příspěvků, počtu sledování a sledujících, a také zejména podle slov typu: „Airdrop, giveawy, free drop, …“. A opět podle metadat user-agenta, které v tomto případě z většiny nebude iOS, Android a Web, ale název registrované aplikace přes Twitter API.

Další užitečný údaje o aktivitě je poloha zveřejnění a reakce na tweet.

3.3.2 Vyhodnocování textového sentimentu

Vyhodnocování textového sentimentu s použitím sofistikovanějšího systému ML či AI je výpočetně náročnější, a proto je potřeba pro toto řešení nejprve

vyfiltrovat nezajímavé retweety a tweety od botů a nevýznamných uživatelů (odhadem bude odfiltrováno až 1/1000 zpráv).

Pro vyhodnocování textového sentimentu z Twitteru proto bude použito více aplikací s odlišným způsobem vyhodnocování. Tímto dosáhneme snadněji udržitelnou real-time analýzu, v případě složitějších a náročnějších metod vyhodnocování, ale také zvýšíme přesnost vyhodnocovaného sentimentu a uživateli poskytneme větší množství dat pro analýzu. Různé typů vyhodnocování sentimentu se pak také hodí pro rozdílné kategorie uživatelů (zpravodajství, traideři, ...).

Pro demonstraci si tak navrhneme tři aplikace vyhodnocující sentiment:

Slovník a Emoji je výkonnostně nejrychlejší aplikace, která pouze rozparsuje slova a podle definovaného ohodnoceného slovníku spočítá průměrný sentiment nalezených slov. Datasety s ohodnocenými slovy pro různé jazyky můžeme nalézt v „open datasetech“ jako například Kaggel (datasets: Emotions Sensor Data Set, Emoji sentiment).

Pro emotikony (emoji) platí v podstatě to samé jako pro ohodnocená slova, ale také dává významný smysl zobrazovat uživateli nejpoužívanější (trandující) emotikony v rámci „aktivity monitoringu“.

TensorFlow-CNN *Twitter sentiment classification by Daniele Grattarola* je konkrétní open-source implementace ML framaworku TensorFlow CNN „convolutional neural network“, zaměřující se na naučení ML modelu a vyhodnocování sentimentu tweetů. [17] Implementace je demonstrována na datasetu obsahující 1 578 627 ohodnocených tweetů a s přesnosti kolem 75 %. [25]

Avšak z vlastního předchozího testování, v týmovém projektu BI-SP2, se jedná o velmi pomalé řešení, alespoň v případě nepodporovanosti grafické karty v běžném cloudové prostředí (pro srovnání, aplikace stíhat zpracovat cca. 2tweety/sekundu v prostředí VM a uvnitř Docker containeru, 4xCPU, 2GB RAM).

Výhody pak tohoto řešení, na rozdíl od jednoduchého ohodnoceného slovníku, je přesnější vyhodnocování přirozeného jazyka, kde naučený model si poradí i s věcmi jako dvojitý zápor a podobné…

Nevýhodou pak tohoto konkretního řešení kromě rychlosti je vyhodnocování sentimentu ve správném kontextu zprávy, která obsahuje více elementů/podmětů. To například můžeme vidět na větě „I really hate Bitcoin and like Ethereum“, kde toto řešení nebude vyhodnoceno jako negativní a pozitivní pro oba podměty zvlášť, ale daný výsledek by pravděpodobně vznikl jako neutrální. [25]

Stanford CoreNPL je rozšířená open-source knihovna, napsaná v jazyce Java, poskytující nástroje pro analýzu přirozeného jazyka (konkrétně: angličtina, čínština, francouzština, němčina, španělština a arabština). Hlavní zaměření knihovny je především na analýzu struktury textu, rozpoznání entit (osob, společností, lokace,…), rozklad slovních druhů, tokenizace slov, zjednodušení vět, parsování vět, ale také i indikování sentimentu v rámci kontextu rozparsovaných vět. [26]

Výhodou použití této aplikace je identifikování kontextu tweetů, což v našem případě je identifikování konkrétní kryptoměny (v případě kryptoměn tato funkce není natolik významná jako u běžných akcii/komodit, kde je o něco těžší rozpoznat příslušný kontext než na bázi přímého názvu kryptoměny). Další výhodou oproti zmíněného TensorFlow řešení je určení podmětu / hlavního kontextu věty a indikování sentimentu v rámci rozparsovaných vět. Například věta „I really hate Bitcoin and like Ethereum“ bude rozparsována na dvě věty kde sentiment pro entitu Bitcoin bude negativní a Ethereum pozitivní. Aplikace je určená i pro běžné cloudové využití, nabízí případné REST API a poskytuje relativně rychlé zpracování textu.

3.4 Návrh architektury aplikace

V souladu se zadáním této bakalářské práce bude architektura aplikací na bázi microservices. Aplikace/systém tedy bude sestavena z několika menších aplikací s integrací pomocí message broker a REST API. Jednotlivé aplikace pak budou nasazeny na cloudové platformě a budou podporovat kontejnerizaci. V rámci aplikací bude také existovat více databází určené pro různé účely analýzy.

Jinou možnosti návrhu architektury pro „real-time stream analytics“, by bylo použít *výpočetní cluster* typu Hadoop. Například vhodnou volbou by byl Apache Spark (nebo Apache Storm), který nabízí knihovny pro Stream Processing (Spark Streaming) i pro způsob vyhodnocování sentimentu pomocí ML – Machine Learnignu (MLib), s podporou pro integrační nástroje jako je Apache Kafka a NoSQL databáze typu Cassandra a další. Takové řešení by bylo pravděpodobně rychlejší a lépe optimalizované, především pro Stream Processing, ale pro ML a celkový vývoj aplikace velmi specifické s ne příliš rozšířenými vývojovými nástroji, což by mělo následky na komplikovanost vývoje, modularitu a flexibilitu aplikace.

3.4.1 Microservices

Architektura microservices, popsaná v kapitole 2.1.1, kromě samotných navržených aplikaci potřebuje nástroje pro koordinaci a komunikace mezi jednotli-

vými aplikacemi, jakožto kompromis oproti monolitickému řešení. Mezi běžné vzory microservices pro webové aplikace patří například:

- service registry/discovery
- config server
- routing, API gatway
- load balancing a circuit breakers
- security service
- distributed messaging – message broker

Tyto nástroje jsou v některých případech součástí deployment platformy např. Kubernetes, Pivotal Cloud Foundry a další. V případě některých požadavků na systém či nativního běhu aplikaci na vlastním serveru je potřeba některé nástroje a aplikace dodat. Volba pak těchto nástrojů pro microservicis architekturu bude částečně záležet na deployment platformě a dalších použitých technologii. Více informací o deplyomentu v sekci 3.4.5.

3.4.2 Aplikace a technologie

Jádro každé aplikace, která je součástí microservices, by mělo být „cloud-ready“ a podporovat webové, integrační a deployment funkce pro rychlý a ucelený vývoj aplikací.

Aplikace v základu lze rozdělit do dvou kategorii podle použitých technologii, způsobu integrace a nasazení:

Web Services označíme jako běžné webové aplikace zajišťující komunikaci a integraci s *vnějšími* systémy a uživateli. V této kategorii budou aplikace zajišťující např. API, security, UI, a další. Použité technologie pro tuto kategorii aplikací, budou patřit mezi standardnější a běžně používané, pro zajištění snadné integrace a bezpečnosti. Takovéto aplikace lze také nasadit i do běžnějších cloudových prostředí, vzhledem k menším nárokům na výpočetní prostředky, škálovatelnost a flexibilitu.

Stream Processing (Data Flow) označíme aplikace určené pro real-time analýzu streamů v rámci integrace s *vnitřními* big data distribuovanými systémy. Zvolené technologie budou navrženy pro splnění vysokých požadavků na škálovatelnost, garantování nepřetržitého běhu systému a s tím spojenou flexibilitu a modularitu pro rozšiřující práci.

V obou kategoriích se bude v základu jednat o webové aplikace, a to z důvodu integrace do cloudových platform, konfigurace a monitoringu. Zvolený *bootstrapping* hlavních microservice aplikací proto bude napsán v jazyce Java (možno i Kotlin a Groovy) s použitím hlavního frameworku **Spring Boot** a **Spring Cloud**. Takovýto přístup zajistí ucelený, lépe doladěný a udržovatelný vývoj projektu i pro případnou navazující práci.

3.4.3 Message broker

V jednoduchém návrhu architektury microservices, aplikace jsou integrovány a navzájem komunikují pomocí REST či SOAP rozhraní využívající standardní webový HTTP protokol. REST (REpresentational State Transfer) je aktuálně primárně používám jako standardní technologie pro API a request/response komunikací mezi backend a frontend aplikacemi (client server), navržené primárně pro synchronní komunikaci. Řešení pro škálovatelnost aplikací používá Load Balancing s metodou round-robin, kde data z aplikace jsou posílány s pomocí service discovery na instanci přijímající aplikace s rovnoměrnou zátěži na dané instance. Toto řešení avšak nezvládá zátěžové špičky, kdy může při synchronní komunikaci dojít k vyčerpání vláken z poolu či zřetězenému timeoutu. Z těchto důvodů a rychlosti REST HTTP protokolu (oproti TCP), běžně používaný návrh integrace aplikací není ideální způsob architektury pro real-time analýzu streamů.

Pro komunikaci mezi aplikacemi bude použit návrhový integrační vzor Message Broker. Zjednodušeně se jedná o message middleware implementující distribuovanou frontu nabízející integraci mezi aplikacemi systémem producer–consumer pro zajištění neblokujícího publikování dat s nízkou latencí a odolností vůči ztrátě dat / transakčnímu zpracování.

Takový návrh také nabízí mnohem lepší flexibilitu a modularitu, kde přidělené (i nově přidělené) aplikace pouze publikují data na určitý kanál (topic) a zpracovávající aplikace zvolený kanál odebírají, i v rámci duplikace zprávy pro více aplikací.

Kafka vs. RabbitMQ

RabbitMQ a Apache Kafka patří mezi nejznámější open-source implementace Message brooker nabízející velkou řadu funkcionalit. RabbitMQ podporuje velkou řadu programovacích jazyků (Java, Spring, .NET, PHP, Python, Ruby, JavaScript, Go, Elixir, Objective-C, Swift) a také nabízí větší volbu messaging protokolů jako AMQP, STOMP, MQTT, ale i standardní webové protokoly HTTP a WebSokets. Kafka, na rozdíl od RabbitMQ, pouze primárně poskytuje Java clienta, ale aktuálně i existují neoficiální open-source clienti podporující většinu jazyků kromě JavaScriptu, čímž se eliminuje použití na frontendu / webovém prohlížeči.

Kafka je primárně navržena pro velký průtok objemu dat se zajištěním durability, rychlosti a škálovatelnosti. Messages jsou ukládány do flat file uložište (log soubor) se zpětným ověřením zapisu (reply on demand), kde pak consumer si sám data přečte pomocí offsetu bez nutnosti jakékoliv administrativní správy ze strany Kafky. Čímž Kafka dosahuje obrovského průtoku dat kolem 3M messages/s pro consumera a 100k messages/s pro publishera. RabbitMQ je spíše běžnější message broker, s podporou okamžité odpovědi na request / reply a zaměřením spíše na routing a bezpečnost, kde maximální zátěž publikování dat je kolem 20k messages/s. [27]

Use-case pak těchto dvou technologii právě spočívá v podporovanosti integrace a výkonosti. Kafka má především použití pro Website Activity Tracking, Metrics, Log Aggregation, Stream Processing, Event Sourcing a Commit logs. V praxi se Kafka často používá s technologiemi Apache Spark, Apache Flink, Apache Beam, Google Cloud Data Flow a Spring Cloud Data Flow. RabbitMQ se pak spíše používá pro integraci aplikací (microservices) vyžadující složitější message routing a podporovanosti širší škály technologii. Aktuální verze Kafka podporuje stream odběru a transakční zpracování, čímž získává stejné výhody jako RabbitMQ. [28]

Zvolený message brokeru pro tuto aplikaci bude **Apache Kafka** z důvodů vyšších požadavků na průtok dat (především pro případnou navazující práci), postačujícího jednoduchého routingu, historie streamů, a dalších předpokladů pro Stream Processing. Kafka navíc pro svůj běh ještě potřebuje externí aplikaci Zookeper (jedná se o highly-available synchronous distributed storage system) pro správu clustrů, routingu a synchronizace consumerů.

3.4.4 Stream Processing – Data Flow

Data Flow nebo-li stream processing je real-time návrh distribuované architektury, která flexibilním způsobem integruje jednotlivé aplikace (microservices) přes message middleware / message broker. Aplikace pak mohou být integrovány podobným způsobem jako „unix pipelines“.

V rámci framevorku Spring Boot a Spring Cloud existuje i DataFlow

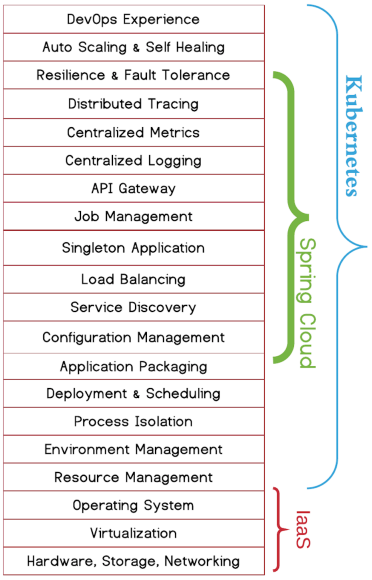
3.4.5 Deployment platforma

Spring Cloud

Open-source framework Spring od Pivotal je nejpopulárnější technologie pro vývoj webových aplikací v jazyce Java a nabízí právě i clouduové řešení Spring Cloud, které dodává většinu nástrojů pro správu microservices.

Srovnání cloudové funkcionalit těchto dvou platform je znázorněno na obrázku

3.1.



Obrázek 3.1: Požadavky na distribuované systémy / microservices [29]

Hlavní cílená platforma v této práce pro setream analýzu bude **Kubernetes**. A to z důvodu funkce Auto Scalingu a Self Healingu (zero-downtime-deploymentu), pro zaručení nepřetržité analýzy streamu v případě zvýšené zátěže, výpadku a upgrade verze. Tuto funkcionalitu řeší také i Pivotal Cloud Foundry, ale nepodporuje Docker kontejnerizaci, což může být základní nedostatek v případě návazné práce.

3.5 Návrh aplikací

3.5.1 Seznam Data Flow aplikací

Stream Processing aplikace v rámci Spring Cloud Data Flow (SCDF) se dělí do tří základních typů: source, processor, sink. V rámci projektu SCDF jsou tyto aplikace pro základní analýzu poskytovány, ovšem je potřeba si vytvořit upravené nebo vlastní aplikace pro doplnění funkcionalit.

tweet-filter aplikace typu – processor – filtrující příchozí tweety podle jednoduchých parametrů podel konfigurace. Aplikace slouží například k odfiltrování podvodných uživatelů, filtrování a routování zvoleného jazyka pro nadcházející zpracování, snížení toku dat a náročnosti na systém

tweet-enrichment aplikace typu – processor – transformující/obohacující tweety o hlavní kryptoměnu nacházející se v kontextu zprávy.

activity-counter aplikace typu – sink – ukládající sociální aktivitu (volume sentimentu) do perzistentní databáze.

tweet-sentiment-sink aplikace typu – sink – ukládající tweety a vyhodnocený sentiment do zvolené databáze.

tensorflow-sentiment aplikace typu – processor – vyhodnocující textový sentiment příchozích tweetů pomocí metody ML z kapitoly 3.3.2 – TensorFlow-CNN.

word-emoji-sentiment aplikace typu – processor – vyhodnocující textový sentiment pomocí a metody ohodnoceného slovníku z kapitoli 3.3.2 – Slovník a Emoji.

npl-sentiment aplikace typu – processor – vyhodnocující textový sentiment pomocí knihovny Stanford CoreNPL z kapitoly 3.3.2 – Stanford CoreNPL, a navíc obohacující sentiment o hlavní kryptoměnu z kontextu tweetu.

3.5.2 Seznam Web Services

Mimo aplikací pro Stream Processing analýzu je potřeba navrhnout bezpečnou službu poskytující vyhodnocená data uživateli, a s tím spojené doplňující informace a konfiguraci.

sentiment-api je aplikace nabízející API pro dotazovaní nad vyhodnocenými daty, sloužící především pro integraci do trading systémů. Data budou poskytována přes webové REST službu a také pomocí HTTP Server-Sent-Events pro streaming aktuálního sentimentu.

coin-market-service je aplikace registrující a sledující pohyb ceny vybraných kryptoměn za účelem srovnání vývoje sentimentu oproti ceně pro nadefinované kryptoměny.

3.5.3 DTO

Streamovaná data mezi Data Flow aplikacemu budou posílána přes message broker především ve formátu JSON. Pro jednoduší práci s daty (filtrování, transformace, enrichment a ukládání), příchozí/odchozí stream bude při deserializaci/serializaci mapován na/z DTO (Data transfer object). V jazyce Java se například pro serializaci a deserializaci používá Jackson knihovna, která umožňuje jednoduše namapovat JSON (nebo XML) na POJO (Plain Old Java Object) pouze pomocí anotací, nebo případně použitím složitějších getterů a stetterů. Výchozí model DTO bude přesně mapován na Twitter objekt, viz oficiální dokumentace [30].

3.6 Databáze

Aplikace analyzující streamy ze sociální sítě Twitter produkuje velké množství jednoduchých záznamů, které je potřeba perzistentně uložit a zpřístupnit k analýze. Za účelem real-time analýzy a ukládání velkého obejmu dat je potřeba zvolit vhodné databázové řešení, které si kromě rychlého zápisu poradí také s vyhledáváním indexovaných a agregovaných dat. Klasické relační (RDBMS) SQL (Structured Query Language) databáze na toto řešení není vhodná především kvůli agregaci a rychlosti čtení/vyhledávání dat, a také ani není potřeba z důvodu jednoduché datové struktury/modelu.

V Big Data technologii se používají NoSQL (Not-only-SQL) database, které si s těmito problémy umí poradit pomocí horizontálního škálování a opačnému přístupu agregace a duplicitě zapisovaných dat, více v teorii 2.1.2.1. Mezi nejefektivnější key-value NoSQL databáze patří například Cassandra, která umí škálovat jak čtení tak i zápis, s ohledem na CAP teorém (z kapitoly 2.1.2.2). Při volbě databáze, pro zaznamenávání vyhodnoceného sentimentu, je preferovanou volbou CAP teorému dodržet AP, za cílem vždy zapsat data a splnit high availability systému.

Vyhodnocená data sentimentu, jak ze sociální sítě Twitter i běžných zpravodajských medii, má jednoduchou strukturu záznamu (measurement) s timestampem, který ve většině případem bude analyzován a agregován na časové ose. Takovéto *time series data* si pro analýzu vyžadují náročné agregace nad velkým množstvím dat vybraného časového úseku, které pro rychlé vyhledávání je potřeba downsamplovat. Například máme-li frekventovaná data zaznamenávající vývoj ceny či sentimentu každou jednu minutu, a chceme získat vývoj ceny za jednotlivé měsíce v posledních pěti letech pro vybraný trh, museli bychom projít a spočítat žádanou operaci nad 2,6M záznamů pro každý požadovaný trh. Což se pak dále komplikuje při volbě různých časových období, filtrace, volby matematické funkce, …, a tedy za účelem komplexní analýzy není možné takováto data jednoduše mezi-ukládat (cache).

V případě použití database Cassanda by bylo potřeba dopředu navrhnout datový model agregovaných a downsamlovaných dat, což by vyžadovalo vytvořit aplikační logiku nad Casandrou pro doplnění funkcionalit. Pro tuto problematiku pak existují specifické time series databases (TSDB) používáné především pro analýzu a monitoring systému, senzorů, events log, a také pro zaznamenávání vývoje ceny na burze.

Podle testovaného měření (benchmarku) [31, 32], TSDB (konkrétně InfluxDB) pro daný use case překonávají Cassandru v:

- 4,5x rychlejším zápisu dat,

- 2,1x menším prostoru dat na disku,
- 45x rychlejší odpovědi na dotaz (vyhledávání).

3.6.1 Time Series Database

TSDB (Time Series Database) jsou databáze optimalizovány pro práci a analýzu nad daty obsahující timestamp s jednoduchou strukturou záznamu, kde detailní vyhledávané záznamy mohou být rychle souhrnně agregovány i nad velkým časovým úsekem.

Mezi starší oblíbené opensource TSDB patří například Graphite, umožnující rychlý zápis velkého objemu záznamů, škálovatelnost a udržovatelnost velkého datasetu napříč delším časovým úsekem. Avšak nevýhodou této database je velmi jednoduchá, omezující single-dimenzinální struktura záznamu, a také zvýšené nároky na úložný prostor v závislosti na počtu omezených indexů.

Pokročilejší a novodobější opensource TSDB, podporující multi-dimenzionální metriky oproti Graphite, jsou InfluxDB a Prometheus, oba napsené v jazyce Go, mírně lišící se odlišným use casem. Prometheus se spíše používá pro monitoring a alerting systémů, především kvůli přístupu ke škálovatelnosti jakož to data-pulling ze strany servru. Naproti tomu InfluxDB, škáluje pomocí clusterování (pouze v Enterprise verzi) a nabízí rozšířený datový model o druhou vrstvu indexace a doprovázejíc informace, čímž nabízí i možnost jakožto nástroj pro loggingu. Z CAP teorému Prometheus zajišťuje pouze AP, vzhledem k architektuře nepodporující konzistenci, InfluxDB pak hybridně CP a AP.

Zvolený typ databáze pro vyhodnocený sentiment bude právě TSDB. A to z důvodu jednoduchého vývoje aplikace oproti jiným zmíněným řešením, a poskytovaným nástrojům pro rychlou efektivní práci s time series data. Konkrétní TSDB pak bude použit InfluxDB vyznačující se vysokou výkoností, možností clustrování a bohatším datovým modelem.

3.6.2 InfluxDB a ElasticSearch

Dalším možným opensource databázovým nástrojem pro analyzování vyhodnoceného sentimentu je ElasticSearch (ES), napsaný v Javě, jakožto distribuovaný systém primárně určený jako indexační full-text vyhledávací engine. Díky volné datové struktuře (podobné NoSQL dokumentu) ukládaných dat ve formátu JSON a pokročilé indexaci se jedná o velmi komplexní a univerzální systém pro analyzovaní, například loggingu.

Výhoda ElasticSerche spočívá především v snadné konfiguraci horizontálního škálování, distribuci a parallizovatelnosti indexovaných dat, HTTP REST a Java API, doprovázejícím nástrojů Logstash (integrace) a Kibana (dashbo-

ard), a také již zmíněná volnost datového modelu, který vylmi urychluje vývoj a poskytuje široké možnosti near-real-time analýzy.

Nevýhodou pak tohoto univerzálního nástroje je rychlost, především pro dotazovaní nad agregovanými daty a dotazování nad větším objemem dat. Podle testování a srovnání s TSDB [33], ES je několikanásobně pomalejší a vyžadující si složitější optimalizaci v rámci JVM. Oproti tomu TSDB – InfluxDB je defaultně optimalizován pro time-series-data, společně s jednoduchým modelem indexace překonávajíc ES v:

- 6,1x vetší rychlosti zápisu,
- 2,5x méně datového úložiště,
- 8,2x rychlejší dotazování (vyhledávání).

V případě srovnání ElasticSearch a Cassandri, oproti InfluxDB [32, 33, 31], ES dosahuje o něco lepší výkonosti oproti Cassadre v rámci time-series-data use-case a indexovaného modelu. Ale vzhledem k durabilitě dat podle testování Jepsen [34] a samotné dokumentace [35], ES může ztrácet data a tedy by neměl být použit jako primární perzistentní databáze.

V řešení této práce budou použity obě databáze, kde **InfluxDB** bude použita jako rychlá základní TSDB pro real-time analýzu a persistenci dat sentimentu za účelem rychlého a efektivního zobrazení dat uživateli. **ElasticSearch** pak bude použit jako prémiový nástroj pro pokročilejší analýzu sentiment, z efektivních a konzistentních důvodů, určená pouze pro menší skupinu uživatelů. Další či rozšířenou možností je použít ES jako batch nástroj pro specifické vyhledávání a poté cache-ování dotazovaných dat do InfluxDB.

Kapitola 4

Implementace

Tato kapitola popisuje použití hlavních zvolených technologii, implementaci dílčích části aplikace a jejich konfigurace pro lokální development.

4.1 Souhrn použité technologie

Architektura microservices a stream processingu je v základu postavená na Spring Boot (Java) aplikacích, které spolu navzájem komunikují přes message broker Kafka, a jsou nasazeny v Docker kontejneru pomocí Spring Cloud Data Flow (SCDF) na lokální či cloudové platformě Kubernetes. Aplikace a průtok dat (streamů) mezi aplikacemi je monitorován v rámci JVM pomocí Micrometer.io, ukládán do InfluxDB a zobrazen pro monitorování v dashboardu Grafana. Vyhodnocená data jsou také zobrazeny v dashbordu Grafana a vystaveny přes RESTové API pomocí Spring Web MVC. Více v následujícím popisu rozděleného podle typu technologií.

4.1.1 Aplikace

Java je objektově orientovaný programovací jazyk, který je kompilovaný do byte code pro spuštění uvnitř Java virtual machine (JVM).

Java je práci použitá jako programovací jazyk pro vývoj webových servcies a stream processing aplikací.

Spring Boot je webový framework vycházející ze základního Spring Frameworku a nabízí autokonfiguraci aplikace podle připojených knihoven a vytvořených „Bean“, také nabízí vestavěný Tomcat server a spuštění aplikace jako JAR (ne WAR), umožnující rychlý začátek vývoje vhodný pro microservices aplikace.

Spring Boot je v práci použit jako základní jádro každé microservice aplikace zajišťující integraci přes message-broker, monitoring, aplikační stav a health endpoint, persistenci dat. Dále je Spring Boot používám i pro běžnou webovou aplikaci nabízející REST API.

4.1.2 Deployment

SCDF Spring Cloud Data Flow – nástroj pro integraci a deployment Spring Boot microservice aplikací do cloudového prostředí Cloud Foundry a Kubernetes, určený pro real-time stream analýzu a task/batch processing.

SCDF slouží v práci pro real-time analýzu sentimentu, s použitím hotových, upravených a nově vytvořených stream aplikací modulárně integrovaných mezi sebou.

Docker Kontejner platforma, v práci určená pro deployment všech potřebných aplikací a zapouzdření aplikací pro běh na Kubernetes platformě.

Kubernetes Deployment platforma pro správu a automatický deployment, škálování a konfiguraci Docker containerů napříč clustery. Použitý jako možná deployment platforma zajišťující automatickou škálovatelnost a zero-downtime-deployment.

4.1.3 Monitoring

Micrometer Monitorovací nástroj pro jednoduchý měření výkonosti JVM aplikací, v základu předkonfigurován pro měření: caches, class loader, RAM a garbage collection, využití CPU, thread pools. Podporující integrace s monitoring systémy jako Influx/Telegraf, Prometheus, Google Stackdriver, Netflix Atlas, AppOptics, …

V práci je použit pro monitoring Stream Processing aplikací a kontroly správného chodu a vytíženosti.

InfluxDB Velmi efektivní a rychlá TSDB (databáze) v práci použitá pro persistenci a analýzu dat vyhodnoceného sentimentu.

ElasticSearch Analytický distribuovaný nástroj primárně určený pro vyhledávání v textu a indexaci. Použitý jako sekundární databáze/nástroj pro pokročilejší analýzu a filtrování vyhodnoceného sentimentu.

Grafana Open source platforma / UI dashboard pro analýzu a monitoring nad time series data, s podporou celé řady „data source“ systémů (Graphite, InfluxDB, Prometheus, Elasticsearch, …)

4.1.4 Vývojové nástroje

Maven Apache Maven je nástroj pro automatické build-ování, řízení a správu dependencí/knihoven, především používán v jazyce Java. Každá aplikace/projekt obsahuje root POM (Project Object Model) XML soubor, který definuje název a verzi projektu, použité dependence knihoven, a jejich konkrétní verze, plugin a build lifcykly (např: .jar package Spring-Boot aplikace, pakage Docker image), a další.

V této práci je Maven použit pro import knihoven, build a spouštění testů jednotlivých aplikací/modulů i celého projektu. V případné navazující prácác pak Maven půjde využít pro CI/CD všech aplikací.

Git Verzovací systém pro správu zdrojových kódů, release verzí a dokumentace k projektu. Zvolený server pro tuto práci je GitLab, který kromě verzování kodů a wiki dokumentace, je také použit pro registraci Docker image, a v navazující verzi může být použit i pro CI/CD.

IntelliJ IDEA Primárně Java IDE (integrated development environmen), od JetBrains, je nejpokročilejší nástroj pro psaní kódů a obecně celkový vývoj aplikace. Tento nástroj nabízí velmi kvalitním coding asistenci (chytré doplňovaní, navigování v kódu, náhled dokumentace, inspekce, oprava efektivity kódu, language injection). A dále také nabízí build nástroje, debugování, runtime hot-swapping, logging, version control – git, test pokrytí, databázové nástroje, aplikační server, integraci s Docker. IDE také podporuje široké škály technologií a frameworků ve velmi detailní kvalitě, čímž rapidně urychluje vývoj projektu.

4.2 Spring Boot

Podle návrhu architektury aplikací 3.4.2, pro zjednodušení vývoje, především integrace a deploymentu, každá aplikace bude „cloud-ready“. Zvolený bootstraping aplikace je za pomocí frameworku Spring Boot a inicializátoru projektu https://start.spring.iostart.spring.io s Web, Cloud a dalšími dependencemi. Architektura microservices se pak skládá z jednotlivých „run-alone“ JAR webových aplikací využívající Spring MVC/WebFlux s vloženým Tomcat/Netty aplikačním serverem.

4.3 Stream Processing

Stream Processing aplikace, je klasická webová aplikace inicializovaná pomocí klasického Spring Boot bootstrapingu start.spring.io, s využitím dependence Spring Cloud Stream, který již nabízí před-zhotovený model pro snadnou a standardizovanou integraci s message broker jako Kafka nebo RabbitMQ.

Nabízeny jsou tři druhy základního rozhraní pro binding: Source (output), Processor(input a output), Sink(input). Případně pro sofistikovanější práci se streamy a Kafkou, je potřeba přidat kromě dependence `Kafka` také dependenci `Kafka Streams`, což například umožňuje práci s Kafka streamy podobně jako Java 8 stream, (`KStream, KTable, GlobalKTable`).

4.3.1 Implementace aplikací

Pro nakonfigurování zvoleného stream bindingu stačí pouze přidat nad třídu anotaci `@EnableBinding` see zvoleným rozhraním, například: `@EnableBinding( Processor.class)`. Specifikování metody pro přijímání a odesílání streamu je pomocí `@StreamListener(Processor.INPUT)` a `@SendTo(Processor.OUTPUT)`. Konfigurace konkrétního topicu pro binding destinaci se definuje v samostatném konfiguračním souboru.

Na ukázce kódu 4.1 je znázorněn princip implementace aplikace a zvolené rozhraní (interface) pro „data binding“ typu „processor“, které filtruje příchozí tweety z destinace `input` a vyfiltrované tweety posílá do destinace `output`. Konkrétní „topic“ (odebíraný kanál) zvolené destinace je pak konfigurován v rámci definice streamu pomocí SCDF, nebo samostatně v parametru či v konfiguraci aplikace, jako na ukázce 4.7 a 4.6. Definované rozhraní `@Filter` nabízí podobné funkce jako rozhraní `@Transformer`, pouze rozšířené o `discard-channel` do kterého lze nakonfigurovat odfiltrované nežádoucí tweety, třeba pro analýzu aktivity botů.

4.3.2 Twitter DTO

Přijímané message v Spring Cloud Stream jsou defaultně nastavené na JSON formát a v základu poskytují nakonfigurovaný MessageConverter s Jackson ObjectMapper pro serializaci a deserializaci POJO. V aplikaci filtrující tweety či ukládající vyhodnocený sentiment je mapování na POJO vytvořeno z důvodu jednoduší práce s daty při filtrování či konventu do DTO measurementu. Ukázka kódu 4.2.

4.3.3 Rozšíření SCDF aplikace

SCDF nabízí hotové aplikace mezi které patří zapouzdřené TensorFlow API pro vyhodnocování textového sentimentu. Avšak aplikace má velmi zjednodušený formát zpracovaného sentimentu a tweetu, které pro dostatečnou analýzu je potřeba rozšířit o celý přijímaný Twitter JSON, detailnější hodnotu vyhodnoceného sentimentu a verze naučeného modelu pro vyhodnocování sentimentu.

Na ukázce kódu 4.3 a 4.4 je znázorněno rozšíření hotové Spring aplikace přidané pomocí Maven dependence. V aplikaci je vytvořena nová nadstavba

Spring Java-base konfigurace definující beanu pro `TensorflowOutputConverter` injectnutou uvnitr základního Spring TensorFlow API – `TensorflowCommonProcessorConfiguration.class`. V konfigurační třídy se taky nachází konfigurace properties (`@ConfigurationProperties` POJO), která slouží pro konfiguraci aplikace při deploymentu.

4.3.4 Stream Pipelines - SCDF

Pro definování streamů je v práci použit Spring Cloud Data Flow (SCDF), jakožto deployment nástroj pro integraci s vytvořenými Spring Cloud Stream aplikacemi určenými pro analýzu sentimentu.

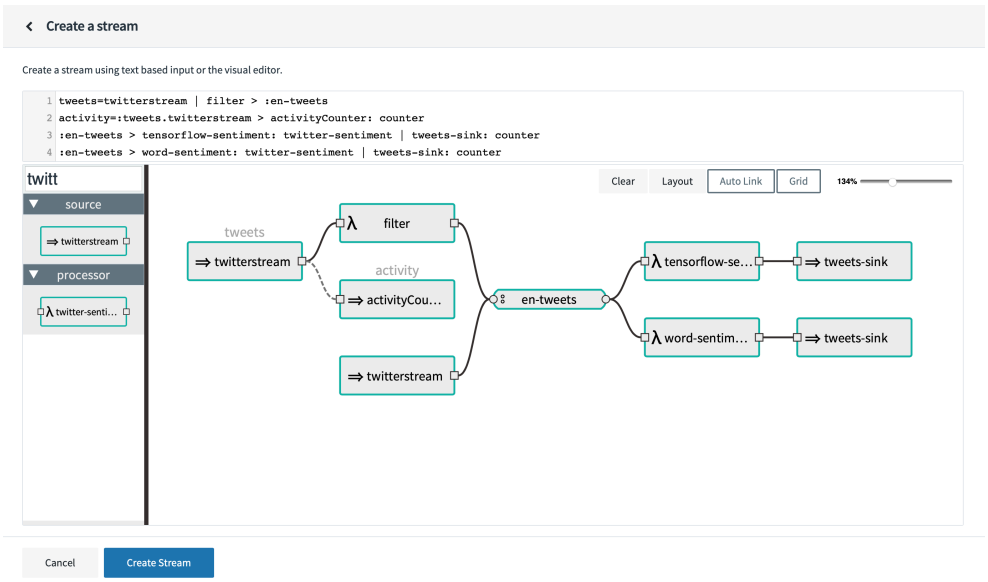
V SCDF se vytváří jednoduché streamy pomocí unixových „pipeline“, například spojením příkazů: `ls -l | grep key | less`. V případě microservices se jedná o jednotlivé aplikace spojené prostřednictvím message middleware, konkrétně použitý Apache Kafka. Aplikace standartě mají jeden input a output, ale je možné vytvořit aplikaci přijímající/odesílající z více destinací, nebo výstup aplikace přesměrovat do společné jedné destinaci `tap`.

Streamy lze pak vytvářet pomocí konzolové aplikace – `Stream Application DSL`, ukázka 4.5, nebo z webového prohlížeče dataflow-servru se znázorněným dataflow diagramem, viz. obrázek 4.1. Aplikační konfugurace (properties), lze doplnit součásti definice streamu (Stream DSL). Ale vzhledem z k použití hesel a tokenů, z bezpečnostních důvodu je vhodnější kompletní konfiguraci všech aplikací streamu nahrát až při deploymentu.

4.3.5 Konfigurace, profily a lokální development

Stream Processing aplikace v rámci SCDF lze spouštět a integrovat s ostatními aplikacemi přímo z IDE bez nutnosti deploymentu do Kubernetes runtime nebo Docker kontejnerizace. Klasická konfigurace v rámci Spring Boot aplikace se nachází pod cesotu `resources/` v souboru `application.properties`, nebo případně YAML formát `application.yml`. Tuto konfiguraci pro integraci pipeline streamů a dalších konfigurací jednotlivých aplikací, konfiguruje SCDF pomocí parametru při spouštěni aplikace. V případě použitá Kubernetes platformy, SCDF automaticky nakonfiguruje CofigMap Kubernetes „podu“.

V případě vývoje aplikace na lokálním stroji, například z IDE, můžeme nastavit profil pro vývojové prostředí v souboru `application-dev.properties` a z něj nakonfigurovat integraci přes message-broker Kafka, stejně jako na ukázce výpisu kódu 4.7. To samé by dalo udělat pomocí parametru při součtení aplikace, ukázka z výpisu kódu 4.6.



Obrázek 4.1: Dataflow-server web UI a stream diagram.

```
@Slf4j
@EnableBinding(Processor.class)
@EnableConfigurationProperties(TweetFilterProperties.class)
public class TweetFilter {

    @Autowired
    private TweetFilterProperties properties;

    @Filter(inputChannel = Processor.INPUT, outputChannel =
    ↪  Processor.OUTPUT)
    public boolean filterTweets(TweetSimple tweet) {
        return filter(tweet);
    }

    public boolean filter(TweetSimple tweet) {
        //return ...
    }

}
```

Výpis kódu 4.1: Binding aplikace s message-middleware pro aplikaci filtrující tweety

```
@JsonIgnoreProperties(ignoreUnknown = true)
@JsonNaming(PropertyNamingStrategy.SnakeCaseStrategy.class)
@Data
public class TweetSimple implements Serializable {

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "EEE
    ↪  MMM dd HH:mm:ss ZZZZZ yyyy", locale = "en")
    private Date createdAt;
    private String text;
    private TwitterUserSimple user;
    //...
    private List<String> hashtags;
    private List<String> symbols;
    private List<String> urls;

    @JsonProperty("entities")
    private void unpackEntities(Map<String, List<Map<String,
    ↪  Object>>> entities) {
        hashtags = entities.get("hashtags").stream()
                .map(s -> ((String) s.get("text")))
                .collect(Collectors.toList());
        symbols = entities.get("symbols").stream()
                .map(s -> ((String) s.get("text")).toUpperCase())
                .collect(Collectors.toList());
        urls = entities.get("urls").stream()
                .map(s -> ((String) s.get("expanded_url")))
                .collect(Collectors.toList());
    }

    public enum TweetType {
        TWEET, RETWEET, QUOTE, REPLY
    }
    public TweetType getTweetType() {
        if(retweetedStatus != null)
            return TweetType.RETWEET;
        if(isQuoteStatus)
            return TweetType.QUOTE;
        if(inReplyToStatusId != null)
            return TweetType.REPLY;

        return TweetType.TWEET;
    }
}
```

Výpis kódu 4.2: Serializace a deserializace Twitter JSON na Java DTO

```
@SpringBootApplication
@Import(TwSentimentProcessorConfiguration.class)
public class TwitterSentimentProcessorApplication {
   public static void main(String[] args) {
      SpringAppli-
      ↪ cation.run(TwitterSentimentProcessorApplication.class,
      ↪ args);
   }
}

@EnableConfigurationProperties({TwSentimentProcessorProperties.class})
@Import(TwitterSentimentProcessorConfiguration.class)
@Slf4j
public class TwSentimentProcessorConfiguration {

   @Autowired
   private TwSentimentProcessorProperties properties;

   @Bean @Primary
   public TensorflowOutputConverter
   ↪ twTensorflowOutputConverter() {
      log.info("Loading custom
      ↪ TwSentimentTensorflowOutputConverter");
      return new TwSentimentOutputConverter(properties);
   }
   //...
}
```

Výpis kódu 4.3: Rozšíření původní aplikace SCDF o vlastní konfiguraci výstupu

```
public class TwSentimentOutputConverter implements
↪  TensorflowOutputConverter<String> {

   private ObjectMapper objectMapper = new ObjectMapper();
   private final TwSentimentProcessorProperties properties;

   @Autowired
   public
   ↪  TwSentimentOutputConverter(TwSentimentProcessorProperties
   ↪  properties) {
      this.properties = properties;
   }

   @Override
   public String convert(Map<String, Tensor<?>> resultTensors,
   ↪  Map<String, Object> processorContext) {
      Tensor tensor =
      ↪  resultTensors.entrySet().iterator().next().getValue();
      //...
      Map<String, Object> outputJsonMap = new
      ↪  HashMap<>(inputJsonMap);
      outputJsonMap.put("sentiment", sentimentString);
      outputJsonMap.put("sentiment_value", resultMatrix[0][1]);
      outputJsonMap.put("sentiment_version",
      ↪  properties.getVersion());
      outputJsonMap.put("sentiment_type", properties.getType());
      //...
      return objectMapper.writeValueAsString(outputJsonMap);
   }
}
```

Výpis kódu 4.4: Rozšíření původní aplikace SCDF – TensorflowOutputConverter

```
tweets=twitterstream | filter > :en-tweets
activity=:tweets.twitterstream > activityCounter: counter
:en-tweets > tensorflow-sentiment: twitter-sentiment | tweets-sink: counter
:en-tweets > word-sentiment: twitter-sentiment | tweets-sink: counter
twitterstream > :en-tweets
```

Výpis kódu 4.5: Vytváření/definice streamů – Stream Application DSL

```
java -Dserver.port=8081
↪  -Dspring.cloud.stream.bindings.input.destination="en-tweets"
↪  -Dspring.cloud.stream.bindings.output.destination="tw-
↪  sentiment" -jar
↪  myapp.jar
```

Výpis kódu 4.6: Spuštění samostatné aplikace s konfigurací message bindingu

```
server.port: 8081
# Kafka topic input/output configuration
spring.cloud.stream:
  kafka.binder.brokers: http://{IP_ADDRESS_KAFKA}:9092
  bindings:
    input:
       destination: en-tweets
       group: myapp-instance-1
    output.destination: tw-sentiment
```

Výpis kódu 4.7: Ukázka development profilu konfiguračního souboru pro Stream Processing aplikaci.

Kapitola 5

Deployment

Stream Processing aplikace jsou nasazeny pomocí Spring Cloud Data Flow (SCDF), který je použit pro deployment na lokálním vývojovém stroji a Kubernetes clusteru. Za účelem otestování produkčního nasazení je vytvořena konfigurace „podů“ pro předem připravený Google Kubernetes Engine cluster. Protože jde o dražší placenou službu je aplikace nasazena na serveru pouze v lokální konfiguraci. Případná navazující práce by tak mohla řešit vlastní konfiguraci Kubernetes clustru na klasických virtuálních strojích.

5.1 Local

Pro jednoduché nastavení lokálního deploymentu je vytvořen `docker-compose.yml`, který umožňuje spustit jedním příkazem `docker-compose up` všechny potřebné doprovázející services jako Docker kontejner. Mezi hlavní services patří:

- kafka
- zookeeper
- dataflow-server
- skipper-server
- influxdb
- grafana
    - elasticsearch
    - logstash

* kibana

Lokální deployment i v případe Dockeru aplikací SCDF se liší ve vytváření a nasazení stream aplikací, kde aplikace jsou spuštěny všechny společně uvnitř jednoho `skipper-server`. Takovýto deployment pak nenabízí výhody distribuovaných systémů a horizontální škálovaní, a je vhodný pouze pro testování a vývoj. V případě plného nasazení je potřeba zvolit Kubernetes, Cloud Foundry či jinou podporovanou platformu.

5.2 Docker Image

Ke spuštění aplikace uvnitř Docker kontejneru, nebo v Kubernetes platformě, je potřeba nakonfigurovat a vytvořit Docekr image pro každou aplikaci. Manuálně lze vytvořit image aplikace v konfiguračním souboru `Dockerfile`, viz ukázka výpisu kódu 5.1, kde je specifikován image operačního systému a samotná aplikace s potřebnými programy (jako Java OpenJDK) spolu s příkazem pro spuštění aplikace uvnitř kontejneru.

Další a v případě microservices praktičtější možností je vytvořit automatizovaný plugin v Maven (např. io.fabric8), který vygeneruje potřebný `Dockerfile` pro aktuální verzi aplikace, a také rovnou umožní build a push image do container registry (např. DockerHub nebo GitLab Registry).

```
FROM springci/spring-boot-jdk11-ci-image:master
COPY target/tw-sentiment-processor-0.0.2-SNAPSHOT.jar
↪  /target/tw-sentiment-processor-0.0.2-SNAPSHOT.jar
VOLUME ["/tmp"]
ENTRYPOINT ["java", "-jar",
↪  "/target/tw-sentiment-processor-0.0.2-SNAPSHOT.jar"]
```

Výpis kódu 5.1: Dockerfile konfigurace

5.3 Kubernetes

Spouštění aplikací na Kubernetes platformě je komplikovanější něž klasický Docker image. Každá aplikace musí mít minimálně dva konfigurační soubory, a to pro globální konfiguraci instancí aplikace `configmap.yaml`, a pro deployment aplikace `deployment.yaml`. Spuštění a vytvoření „podů“ se provede příkazem: `kubectl create -f ./`.

Vzhledem k již vytvořenému `docker-compose.yaml` z Docker lokálního developmentu, je možné vygenerovat příslušné konfigurační soubory i Kubernetes pomocí nástroje „Kompose“, a po vygenerování případně upravit.

Tyto konfigurace jsou vytvořeny pouze pro doprovázející aplikace typu Services, jelikož aplikace typu Stream Processing jsou nasazeny pomocí SCDF, který příslušnou konfiguraci vytvoří automaticky.

5.4 Zero-down-time-deployment

SCDF používá Skipper server a Spring Cloud Deployer pro deployment data pipelines / aplikací do cloudového prostředí Kubernetes či Cloud Foundry (i Local, YARN, Mesos). SCDF se stará o konfiguraci přidělených prostředků a množství bežících instancí, o registrovaní verzí aplikací, upgrade a rollback aplikací/streamů a dalších funkcionalit. Konfigugurace zero-downtime-deploymentu je tak automaticky zajištěna v rámci Spring Cloud aplikací a funkcionality SCDF.

V případě deploymentu aplikace typu Service mimo SCDF, musí být nakonfigurovat „YAML Deployment descriptor“ pro každou aplikaci zvlášť. V ukázce výpisu kódu 5.2, je určen počet instancí/replik a strategii „rolling update“ s maximálním počtem možných nedostupných instancí, určených pro update, tak aby vždy existoval dostatečný počet instancí pro běh nepřetržitý aplikace.

Kubernetes pro rolling udpate potřebuje znát aplikační stav, aby mohl označit „pod“ (instanci aplikace) jako připravený ke spuštění. Spring Boot nabízí nástroj Actuator, který stačí přidat jako Maven dependenci, a tím vytvoří API endpoint: `/actuator/health`, na kterém nabízí informaci o stavu aplikace. Konfigurace endpointu v „podu“ pak bude vypadat jako na ukázce výpisu kódu 5.3. Pro deployment v rámci SCDF je tato konfigurace automaticky zajištěna.

```
apiVersion: apps/v1
kind: Deployment
spec:
 replicas: 3
 strategy:
   rollingUpdate:
     maxSurge: 0
     maxUnavailable: 1
   type: RollingUpdate
```

Výpis kódu 5.2: Konfigurace deployment descriptoru pro zajištení rolling update.

5.5 Dokumentace

Dokumentace k nasazení jednotlivých aplikací typu Services a Stream Processing na konkrétní deployment platformu je uvedena společně se zdrojovými kódy v příloze práce a verzovacím systému Git (GitLab).

```
spec:
 containers:
 - name: my-app
   image: myimage:1.0
   readinessProbe:
     httpGet:
       path: /actuator/health
       port: 8080
```

Výpis kódu 5.3: Konfigurace „Kubernetes Deployment“ a nastavení „endpoint“ pro zjištění stavu aplikace.

Kapitola 6

Testování a monitoring

6.1 Testování

Součástí každé aplikace je také testování, za účelem minimalizování chyb výsledné aplikace.

6.1.1 Unit a integrační testy

Pro testování se dají použít jednoduché Unit testy, které pro běh nepotřebují spuštění samotné aplikace. Ale také lze vytvořit integrační testy v rámci Spring contextu a doprovázející autokonfigurace a konfigurace aplikace.

Na ukázce výpisu kódu 6.1 je znázorněn JUnit a integrační test Stream Processing aplikace filtrující tweety na základě konfigurace v `src/test/resources/application.properties` s využitím `spring-boot-starter-test` pro načtení contextu aplikace.

6.1.2 Systémové integrační testy

Dalším možným způsobem testování by byly integrační testy napříč aplikacemi přes vybraný message-broker, což by vyžadovalo nakonfigurování DevOps CI/CD, nebo vytvoření aplikace spouštějící stream pipeline pro SCDF testovací server. Takové řešení je nad rámec práce a může být doplněno v navazující práci. Ale i tak můžeme uživatelsky otestovat integraci aplikací při deploymentu v lokálním prostředí, kde za pomocí defaultní konfigurace Spring Boot Actuator lze monitorovat spuštěné streamy. Případné integrační a aplikační chyby kromě logu můžeme vidět právě i v monitorovacím nástroji, kde chyby v rámci streamu jsou posílány do error-channel příslušného message brokeru.

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment =
↪  SpringBootTest.WebEnvironment.RANDOM_PORT)
public class TweetFilterApplicationTests {

   private ObjectMapper mapper;

   @Autowired
   private TweetFilter tweetFilter;

   @Before
   public void init() {
      mapper = new ObjectMapper();
   }

   @Test
   public void filterTest() throws IOException {
      TweetSimple tweetTrue = mapper.readValue(new
      ↪  File("src/test/resources/tweets-input1.json"),
      ↪  TweetSimple.class);
      TweetSimple tweetFalse = mapper.readValue(new
      ↪  File("src/test/resources/tweets-input2.json"),
      ↪  TweetSimple.class);

      assertThat(tweetFilter.filterTweets(tweetTrue));
      assertThat(!tweetFilter.filterTweets(tweetFalse));
   }
   //...
}
```

Výpis kódu 6.1: Testování filtrace a deserializace příchozí zprávy.

6.2 Dashboard Grafana

Pro analýzu a zobrazení dat uživateli byl zvolen Grafana dashboard, který je předkonfigurovaný a spuštěný z Docker image s doprovázejícími pluginy pro napojení na databaze InfluxDB a ElasticSearch, a dalšími pluginy pro rozšiřující grafy.

Za účelem analýzy byly v Grafaně vytvořeny dva dashboardy: Aktivita a Sentiment. V pohledu „Aktivita“ je zobrazen vývoj o aktivitě na sociální síti, tedy počet tweetů, retweetů, …, a to samé také pro nejčastější použité hashtagy, které zachycují aktuální trend. V pohledu Sentiment se naopak nachází vývoj sentimentu, rozdělen podle `tagu` na pozitivní, negativní a neutrální, dále zjednodušený pohled nad vyfiltrovanou aktivitou, a také přehled posledních a nejvýznamnějších tweetu ve zvoleném časovém úseku s možným překliknutím na webové stránky Twitteru. Nad oběma pohledy je možné filtrovat data podle tagů a také rychle agregovat data do časových intervalů, podobných jako na burze (5m, 1h, 4h, …), automaticky i manuálně.

Data vyhodnoceného sentimentu lze získat z InfluxDB do Grafany pomocí SQL syntaxe, například ukázka výpisu kódu 6.2, ale také i interaktivně z Grafana UI. Grafana dále také nabízí upozornění (alert), které je například nastaveno v rámci monitoringu systému, v případě prudkého poklesu aktivity.

```
SELECT count("value") FROM "twitter_sentiment" WHERE
↪  ("tweet_type" =~ /^(QUOTE|REPLY|RETWEET|TWEET)$/) AND time >=
↪  now() - 3h GROUP BY time(5m), "sentiment" fill(null);SELECT
↪  moving_average(median("value"), 10) FROM "twitter_sentiment"
↪  WHERE time >= now() - 3h GROUP BY time(5m) fill(null)
```

Výpis kódu 6.2: Grafana InfluxDB query

6.2.1 Monitoring

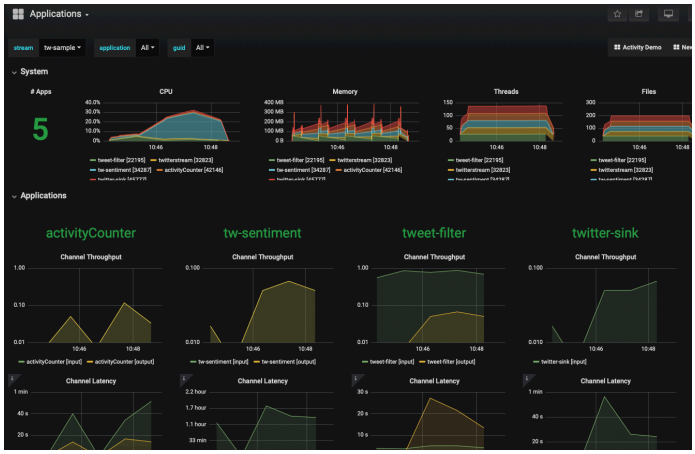
Monitorování aplikací je možné na úrovni Kubernetes Docker kontejnerů, nebo podrobněji v rámci JVM pomocí `micrometer.io`. Pro integrování Micrometer se Spring Boot aplikací stačit pouze přidat Maven dependenci a nakonfigurovat příslušnou monitorovací databázi, viz ukázka výpisu kódu 6.3 automatické konfigurace vygenerované z SCDF.

Micrometer v základu pro Spring Boot aplikaci umožňuje sledovat vytíženost procesoru, JVM paměti, ale také i sledování průtoku a latence pro konkrétní „channel“ message-brokeru (Kafka), včetně globálního „error channelu“ určeného pro alerting. Do Grafana dashboaru, viz obrázek 6.2, je pak importován samotný dashbord poskytnutý od SCDF určený pro monitoring streamů s vytvořenou konfigurací pro zvolené databáze (InfluxDB).

Obrázek 6.1: Grafana dashboard – sentiment pohled

```
app.activityCounter.management.metrics.export.influx.db =
↪ myinfluxdb
app.activityCounter.management.metrics.export.influx.enabled =
↪ true
app.activityCounter.management.metrics.export.influx.uri =
↪ http://influxdb:8086
```

Výpis kódu 6.3: Konfigurace monitoringu pro aplikaci ActivityCounter v rámci SCDF



Obrázek 6.2: Grafana a SCDF – monitoring streamů

Závěr

Cílem práce bylo navrhnout, implementovat a nasadit škálovatelnou architekturu aplikací vyhodnocující textový sentiment ze sociálních medií či zpravodajů na téma kryptoměn. Implementované řešení je založeno na architektuře microservices a nasazeno v cloudovém řešení splňující všechny požadované vlastnosti dané architektury, čímž zadání bylo naplněno.

V teoretické části práce jsme se seznámili s architekturou microservices, technologiemi pro big data a real-time analýzy streamů. Na základě těchto znalostí byly v práci zvoleny vhodné technologie, vyzkoušeno jejich použití, a po jejich detailním nastudování byly v práci implementovány.

Zvolená architektura splňuje požadavky pro horizontálního škálovaní a modularitu se snadnou rozšířitelností pro případnou navazující práci. Navazující práce by tak mohla využít vytvořené architektury a vytvořených aplikací pro vybudování sofistikovanějšího a přesnějšího systému vyhodnocování sentimentu z více typů zdrojů. Dalším zaměření navazující práce by bylo nakonfigurovat plnohodnotný Kubernetes cluster a pro vývoj použít DevOps CI/CD.

Vysledná práce, včetně zdrojových kódu a docker images, je zdokumentována na GitLabu a nahrána na přiloženém USB flash disku.

Bibliografie

1. JAMES LEWIS, Martin Fowler. *Microservices a definition of this new architectural term* [online]. 2014 [cit. 2019-04-18]. Dostupné z: `https://martinfowler.com/articles/microservices.html`.
2. KAPPAGANTULA, Sahiti. *What Is Microservices? An Introduction to Microservice Architecture* [online]. 2018 [cit. 2019-04-17]. Dostupné z: `https://dzone.com/articles/what-is-microservices-an-introduction-to-microserv`.
3. GATTERMAYER, Josef. Co jsou to Big Data. In: *BI-BIG Big Data* [online]. ČVUT Fakulta informačních technologií, 2018, s. 16–18 [cit. 2019-04-18].
4. MEHRA, Akhil. *Understanding the CAP Theorem* [online]. 2019 [cit. 2019-04-18]. Dostupné z: `https://dzone.com/articles/understanding-the-cap-theorem`.
5. MESSINGER, Lior. *Better explaining the CAP Theorem* [online]. 2013 [cit. 2019-04-18]. Dostupné z: `https://dzone.com/articles/better-explaining-cap-theorem`.
6. GATTERMAYER, Josef. Co jsou to Big Data. In: *BI-BIG Big Data* [online]. ČVUT Fakulta informačních technologií, 2018, s. 41–56 [cit. 2019-04-18].
7. DOCKER INC. *What is a Container?* [online]. © 2019 [cit. 2019-04-18]. Dostupné z: `https://www.docker.com/resources/what-container`.
8. IMPETUS TECHNOLOGIES, Inc. Streamanalytix. *The Visual Big Data Analytics Platform for Stream Processing and Machine Learning* [online]. 2017 [cit. 2019-04-18]. Dostupné z: `https://www.streamanalytix.com/wp-content/uploads/2017/12/StreamAnalytix-3.1-2.pdf`.

9. WÄHNER, Kai. *Comparison of Streaming Analytics Frameworks* [online]. 2016 [cit. 2019-04-18]. Dostupné z: https://dzone.com/articles/comparison-of-streaming-analytics-frameworks.
10. FEDAK, Vladimir. *8 Open Source Big Data Tools to use in 2018* [online]. 2018 [cit. 2019-04-18]. Dostupné z: https://towardsdatascience.com/8-open-source-big-data-tools-to-use-in-2018-e35cab47ca1d.
11. SOCIAL MARKET ANALYTICS, Inc. *Social Market Analytics (SMA)* [online]. 2011 [cit. 2019-04-17]. Dostupné z: https://www.socialmarketanalytics.com.
12. GROUP, CME. *Equity Index* [online]. © 2019 [cit. 2019-04-17]. Dostupné z: https://activetrader.cmegroup.com/Products/EquityIndex?s=BTC.
13. ISENTIUM, LLC. *iSentium* [online] [cit. 2019-04-17]. Dostupné z: http://isentium.com.
14. ALTERNATIVE.ME. *Fear and Greed Index* [online]. © 2019 [cit. 2019-04-17]. Dostupné z: https://alternative.me/crypto/fear-and-greed-index/.
15. LIN, Jin. *CoinGossip.io* [online]. © 2019 [cit. 2019-04-17]. Dostupné z: https://coingossip.io.
16. NLP, Stanford. *Stanford CoreNLP* [online]. 2014 [cit. 2019-04-18]. Dostupné z: https://stanfordnlp.github.io/CoreNLP/index.html.
17. GRATTAROLA, Daniele. *Twitter sentiment classification by Daniele Grattarola* [online]. 2016 [cit. 2019-04-18]. Dostupné z: https://github.com/danielegrattarola/twitter-sentiment-cnn.
18. XIA, Meng Xuan. *Stanford CoreNLP* [online]. 2017 [cit. 2019-04-18]. Dostupné z: https://github.com/xiamx/awesome-sentiment-analysis.
19. TILKOV, Stefan. *A Brief Introduction to REST* [online]. 2010 [cit. 2019-05-01]. Dostupné z: https://www.infoq.com/minibooks/emag-03-2010-rest.
20. REES, Eric van. *Introduction to WebSockets* [online]. 2018 [cit. 2019-05-01]. Dostupné z: https://www.scaledrone.com/blog/introduction-to-websockets/.
21. REES, Eric van. *Getting to know Server-Sent Events (SSE)* [online]. 2010 [cit. 2019-05-01]. Dostupné z: https://www.scaledrone.com/blog/getting-to-know-server-sent-events-sse/.
22. TWITTER, Inc. *Building standard queries* [online]. © 2019 [cit. 2019-05-08]. Dostupné z: https://developer.twitter.com/en/docs/tweets/rules-and-filtering/overview/standard-operators.

23. TWITTER, Inc. *Premium operators* [online]. © 2019 [cit. 2019-05-08]. Dostupné z: https://developer.twitter.com/en/docs/tweets/rules-and-filtering/overview/premium-operators.

24. TWITTER, Inc. *Tweet objects* [online]. © 2019 [cit. 2019-05-08]. Dostupné z: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.

25. NAJI, Links. *Twitter Sentiment Analysis Training Corpus (Dataset)* [online]. 2012 [cit. 2019-04-18]. Dostupné z: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/.

26. MANNING, Christopher D.; SURDEANU, Mihai; BAUER, John; FINKEL, Jenny; BETHARD, Steven J.; MCCLOSKY, David. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Association for Computational Linguistics (ACL) System Demonstrations* [online]. 2014, s. 55–60 [cit. 2019-04-28]. Dostupné z: http://www.aclweb.org/anthology/P/P14/P14-5010.

27. TRUDEAU, Yves. *Exploring Message Brokers: RabbitMQ, Kafka, ActiveMQ, and Kestrel* [online]. 2014 [cit. 2019-04-18]. Dostupné z: Exploring%20Message%20Brokers:%20RabbitMQ,%20Kafka,%20ActiveMQ,%20and%20Kestrel.

28. HUMPHREY, PIETER. *Understanding When to use RabbitMQ or Apache Kafka* [online]. 2017 [cit. 2019-04-18]. Dostupné z: https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka.

29. IBRYAM, Bilgin. *Spring Cloud for Microservices Compared to Kubernetes* [online]. 2016 [cit. 2019-05-08]. Dostupné z: https://developers.redhat.com/blog/2016/12/09/spring-cloud-for-microservices-compared-to-kubernetes/.

30. TWITTER, Inc. *Introduction to Tweet JSON* [online]. © 2019 [cit. 2019-05-08]. Dostupné z: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json.

31. CHURILO, Chris. *InfluxDB Tops Cassandra in Time Series Data & Metrics Benchmark* [online]. 2018 [cit. 2019-04-18]. Dostupné z: https://www.influxdata.com/blog/influxdb-vs-cassandra-time-series/.

32. INFLUXDATA. *influxdb-comparisons* [online]. 2016 [cit. 2019-04-18]. Dostupné z: https://github.com/influxdata/influxdb-comparisons.

33. CHURILO, Chris. *InfluxDB vs. Elasticsearch for Time Series Data & Metrics Benchmark* [online]. 2018 [cit. 2019-05-08]. Dostupné z: https://www.influxdata.com/blog/influxdb-markedly-elasticsearch-in-time-series-data-metrics-benchmark/.

34. STRIPE. *Jepsen: Elasticsearch 1.5.0* [online]. 2015 [cit. 2019-05-08]. Dostupné z: https://aphyr.com/posts/323-call-me-maybe-elasticsearch-1-5-0.

35. B.V., Elasticsearch. *Elasticsearch Resiliency Status* [online]. © 2019 [cit. 2019-05-08]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/resiliency/current/index.html#_jepsen_test_failures_status_ongoing.

Příloha A

Seznam použitých zkratek

AI Artificial intelligence

API Application programming interface

DTO Data transfer object

ES ElasticSearch

GUI Graphical user interface

IDE Integrated Development Environmen

JDK Java Development Kit

JSON JavaScript Object Notation

JVM Java Virtual Machine

ML Machine learning

NoSQL Not only SQL

RDBMS Relational Database Management System

REST Representational State Transfer

OHCL Open-high-close-low chart

OS Operační systém

SQL Structured Query Language

TSDB Time Series Database

XML Extensible markup language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `dataflow_kubect`
  - `README.md` ..... dokumentace k nasazení na Kubernetes
  - `kubernetes` ..... adresář obsahující konfiguraci pro spuštění
- `dataflow_local`
  - `README.md` ..... dokumentace k nasazení na local
  - `docker-compose.yaml` ..... soubor pro spuštění
- `scdf_apps` ..... adresář obsahující zdrojové kódy aplikace
  - `pom.xml` ..... Maven root module pro build všech aplikací
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX