Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Proudy událostí a nástroje na jejich zpracování

Diplomová práce

Autor: Bc. Jiří Pipek
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Karel Malý, Ph.D.

Hradec Králové duben 2024

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 8.4.2024 Jiří Pipek

Poděkování:

Děkuji vedoucímu diplomové práce panu Ing. Karlu Malému, Ph.D. za metodické vedení práce.

Anotace

Diplomová práce poskytuje ucelený pohled na proudy událostí, zkoumá technologie a platformy pro jejich uchování i zpracování a identifikuje typické problémy s návrhy na jejich řešení. Klade důraz na platformy Apache Kafka, Azure Event Hubs, Google Cloud Pub/Sub a Amazon Kinesis Data Streams pro uchování událostí a Kafka Streams, Apache Flink, a Apache Spark pro zpracování. Práce obsahuje teoretickou část s přehledem základních konceptů a srovnáním technologií, a praktickou část, kde jsou pomocí vybraných scénářů testovány klíčové funkcionality jako obohacení, filtrace, agregace a transformace proudů dat. Zkoumané nástroje jsou porovnány na základě latence, propustnosti, využití zdrojů a náročnosti vývoje. Výsledky slouží pro základní orientaci v oblasti proudového zpracování událostí, přičemž poukazují na nutnost dalšího testování pro hlubší pochopení vhodnosti technologií pro specifické použití.

Annotation

Title: Event streams and event stream processing software

The diploma thesis provides a comprehensive view of event streams, explores technologies and platforms for storing and processing them, and identifies typical problems with suggestions for their solution. It emphasizes the Apache Kafka, Azure Event Hubs, Google Cloud Pub/Sub, and Amazon Kinesis Data Streams platforms for event storage and Kafka Streams, Apache Flink, and Apache Spark for processing. The thesis includes a theoretical part with an overview of basic concepts and a comparison of technologies, and a practical part where key functionalities such as enrichment, filtering, aggregation, and data transformation are tested using selected scenarios. The examined tools are compared on the basis of latency, throughput, resource utilization and development effort. The results serve as a basic orientation in the field of stream-based event processing, while pointing to the need for further testing to gain a deeper understanding of the suitability of the technologies for specific applications.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Seznam zkratek

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CPU	Central Processing Unit
CQRS	Command Query Responsibility Segregation
HDFS	Hadoop Distributed File System
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MLlib	Machine Learning Library
MQTT	Message Queuing Telemetry Transport
PHP	Hypertext Preprocessor
Protobuf	Protocol Buffers Documentation
RAM	Random Access Memory
UUID	Universally Unique Identifier
VoIP	Voice over Internet Protocol
XML	Extensible Markup Language

1 Úvod

Dnešní doba je poměrně turbulentní a přináší mnohokrát problémy, na které je třeba rychle reagovat. Může se jednat jak o běžné události ve světě, tak také o potřebu rychlých reakcí v rámci různých sektorů podnikání, IT svět nevyjímaje. Řešením těchto problémů se stávají čím dál častěji proudy událostí, které hrají klíčovou roli právě v umožnění okamžitých reakcí, především v možnosti zpracování dat v reálném čase. Zejména tak slouží k analýze velkých objemů dat, pomocí nichž organizacím umožňují rychle reagovat na změny a získávat cenné poznatky pro podnikový přínos.

Díky obrovskému nárůstu popularity této oblasti není však jednoduché se vyznat v existujících nástrojích, které slouží pro uchování a zpracování těchto událostí. Jako všechny technologie, tak i tyto mají svá specifika a je potřeba provést detailní výzkum toho, co umožňují a jakým způsobem se dají využít. Nároky organizací se totiž často liší: některé vyžadují uchování dat ve svých datacentrech, jiné vyžadují kompletně celou správu realizovat v cloudových úložištích třetích stran.

Po výběru vhodné platformy pro uchování dat, do které začnou proudit miliony událostí, je třeba tato data začít systematickým způsobem zpracovávat. I v této oblasti dnes existuje poměrně velká škála možností, ve které na počátku není zcela snadné se zorientovat. Společnosti tak často dělají chyby ve výběru a utrácí výrazné množství peněz za vývoj aplikací, které poté nejsou pro jejich případy použití vhodné.

Následně se může dostavit neporozumění návrhovým vzorům, které se pro asynchronní komunikaci a proudy událostí používají. Vývojáři se k těmto architekturám často staví nedůvěřivě, nedrží se jich a často naráží na řadu problémů, které tato oblast pod povrchem skrývá.

Práce si tedy klade za cíl tuto problematiku zmapovat a poskytnout ucelený pohled, podle kterého bude možné se případným problémům vyhnout.

2 Cíl práce

Diplomová práce si klade za cíl definovat problematiku proudů událostí, představit výběr platforem a technologií pro jejich uchování i zpracování. Zároveň se pokusí nastínit i konkrétní problémy, které při práci s nimi mohou vznikat a navrhnout jejich ošetření.

V rámci teoretické části bude nejprve vysvětlena základní terminologie a koncepty spojené s proudy událostí, včetně definic a vlastností těchto proudů. Následovat bude detailní představení vybraných platforem pro uchování událostí a popis rozdílných přístupů ke zpracování proudů událostí. V této části budou také teoreticky zkoumány hlavní technologie pro praktické porovnání. Speciální pozornost bude věnována identifikaci a řešení problémů, popřípadě chybových stavů, které mohou při práci s událostmi vznikat.

V praktické části práce bude provedena analýza klíčových funkcí, které jednotlivé nástroje na zpracování událostí nabízejí. Z nich poté vzniknou vhodné testovací scénáře, pomocí kterých bude možné nástroje porovnat. Jednotlivé scénáře budou následně implementovány ve všech třech zkoumaných nástrojích s cílem naměřit a porovnat vybrané klíčové metriky pro každou z analyzovaných technologií. Tím bude možné posoudit jejich vhodnost pro různé typy scénářů v praxi. Mezi takové metriky budou patřit především požadavky na hardware, výkonnost jednotlivých technologií u různých typů operací, ale třeba také náročnost na vývoj.

Celkově by měl výstup práce přispět k lepšímu porozumění a využití těchto technologií v praxi.

3 Proudy událostí

Kapitola je soustředěna na obecnou definici proudů událostí a jejich vlastností, pojednává také o konceptech, které se v posledních letech objevují ve veřejném prostoru soustředěném kolem informačních technologií, zejména okolo zpracování dat v reálném čase a také datových integrací [1]. Dále se dotýká problematiky platforem a technologií, které jsou pro uchování proudů událostí vhodné.

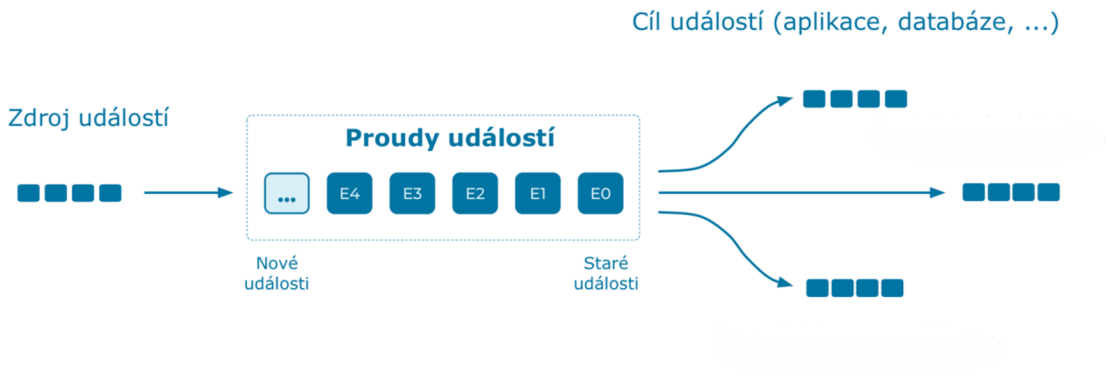
3.1 Událost

V kontextu této kapitoly je třeba nejprve získat jasný vhled do pojmu události, neboť události tvoří základní stavební kámen, na němž stojí problematika proudů událostí. Událost je v obecném smyslu termínem označujícím konkrétní děj, situaci nebo jev, který nastává v určitém čase a místě. Jedná se o moment, kdy dochází k nějaké změně, akci nebo reakci a může být pozorován a zaznamenán. Události se vyskytují v různých aspektech lidského života a v různých oborech, díky tomu jsou klíčovým prvkem pro porozumění děje v reálném světě. V kontextu softwarových systémů lze události chápat jako jevy, které nastávají uvnitř samotného systému. Tyto události v sobě často nesou významné informace o dění, které ovlivňuje průběh podnikání nebo způsob, jakým uživatelé interagují s konkrétním procesem.

Události mohou mít různou formu i význam a často se vyskytují v různých kontextech. Může jít o událost, jako je odeslání emailu odběrateli, aktualizace stavu inventáře v distribučním centru nebo dokončení platby za produkty či služby. Nicméně klíčovým aspektem událostí je, že každá z nich by měla spustit jednu či více akcí či procesů jako reakci na ně [6].

3.2 Definice a vlastnosti proudů událostí

Proudy událostí, často označované také jako streamování dat, představují kontinuální a neustálý tok informací, přičemž každá jednotlivá zpráva obsahuje data týkající se konkrétní události nebo změny v systému. Klíčovým aspektem proudů událostí je ve většině případů jeho okamžitá a reálná analýza, která probíhá bezprostředně po příjmu dat do systému [2]. Způsob zpracování dat a následná akce může být ovlivněna povahou samotných dat či charakterem událostí.



Obrázek 1 Datový tok proudů událostí. Podle: [3]

Hlavní vlastností a charakteristikou proudu událostí je, že není omezen časovým rámcem, což znamená, že nemá přesný začátek ani konec. Každá jednotlivá událost je zpracovávána v okamžiku jejího vzniku a následně řízena podle určených pravidel. Události mohou spouštět specifické chování aplikací a být využívány pro firemní vizualizace v reálném čase [4]. Stejně tak mohou být dále integrovány do jiných systémů pro jejich pozdější zpracování ve větších dávkách [4].

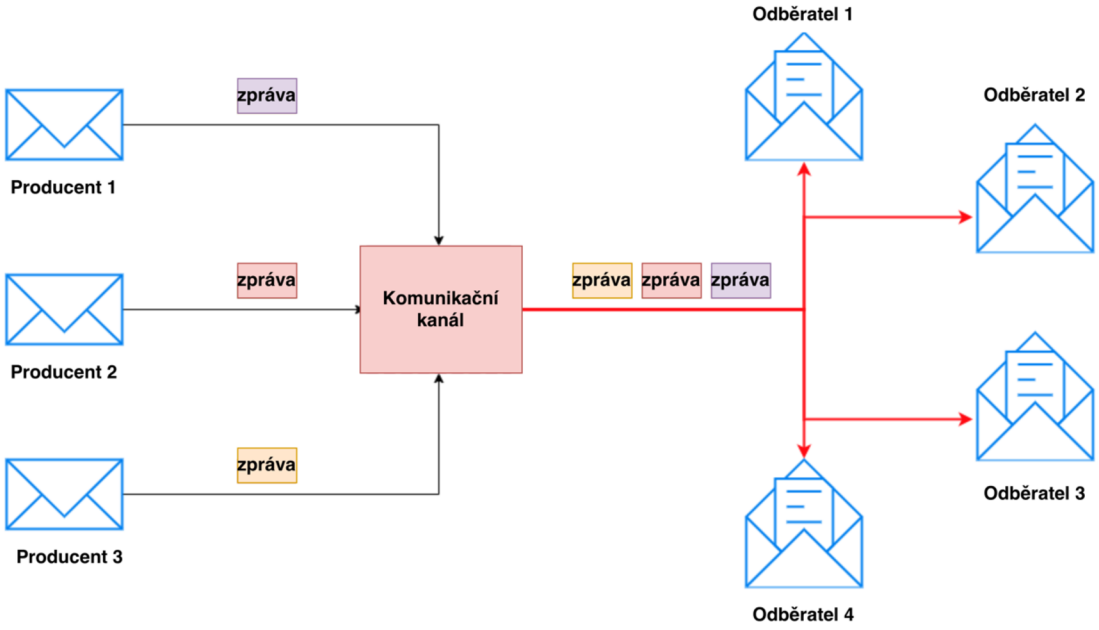
Příkladem, který ilustruje tok proudů událostí, mohou být bankovní transakce. Při každém pohybu na účtu je generována nová událost obsahující informace, jako je například čas transakce, bankovní účet příjemce, částka a mnohé další. Jelikož banka má obvykle stovky různých bankovních účtů a může provádět tisíce transakcí denně, generuje tak kontinuální proud dat. Způsob a četnost, jakým organizace analyzují tato data a reagují na ně, závisí na specifických potřebách i cílech dané organizace v rámci její obchodní činnosti [5]. Konkrétně u transakcí se může jednat o odhalování podvodů v reálném čase.

3.3 Platformy pro proudy událostí

Platformy pro proudy událostí (Event Streaming Platforms, dále jen ESP) jsou klíčovými technologiemi, které společnostem umožňují efektivní přijímání a uchovávání gigabajtů událostí za sekundu z různých zdrojů. Shromážděná data jsou následně k dispozici v téměř reálném čase pro ostatní aplikace, které mohou reagovat na události v okamžiku jejich vzniku [7].

Všechny platformy fungují na principu *publish-subscribe*, kde při komunikaci mezi komponentami jedna z nich (producent) zprávy odesílá a jiné (odběratelé)

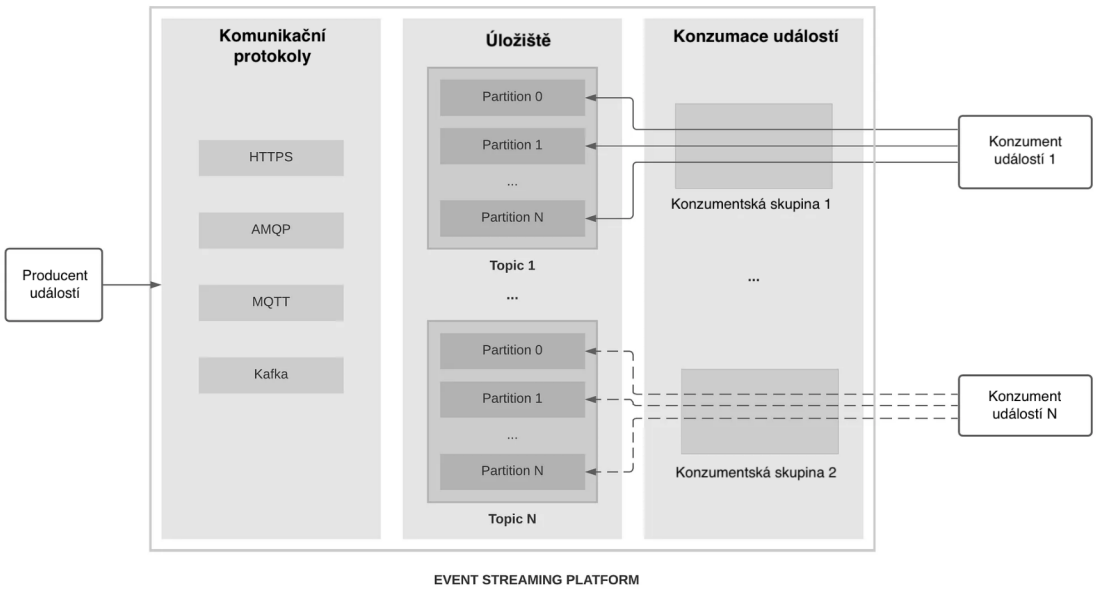
se přihlašují k odběru těchto zpráv. K této výměně se obvykle používá prostředník, v tomto případě komunikační kanál, který zprávy uchovává [11].



Obrázek 2 Publish-subscribe model přenosu událostí. Podle: [12]

Architektura těchto platforem, zobrazena na Obrázek 3 níže, se skládá ze třech hlavních částí, které budou v následujících kapitolách podrobně popsány. Detailní vysvětlení je nezbytné, jelikož v mnoha případech se ESP nesprávně zaměňují za tradiční *messaging systems*[1], což může vést velmi jednoduše k chybám v návrzích aplikací, a pak i k samotným implementačním chybám.

[1] *Messaging systems* umožňují aplikacím a systémům výměnu zpráv pomocí front, ze kterých jsou ve většině případů přečtené zprávy odebírány. Příkladem takových systémů je například RabitMQ, nebo ActiveMQ [12].



Obrázek 3 High-level architektura platforem pro proudy událostí. Podle: [14]

3.3.1 Komunikační protokoly

Obecně komunikační protokoly představují sadu formálních pravidel, která popisují postup přenosu nebo výměny dat, zejména v rámci sítě. Standardizovaný komunikační protokol je takový, který byl zaveden jako norma. Mezi příklady těchto standardizovaných protokolů patří bezdrátová síť (WiFi), internetový protokol (IP) a hypertextový přenosový protokol (HTTP) [15].

V kontextu ESP je tato vrstva zodpovědná za příjem událostí od zdrojových systémů s velmi vysokou propustností. Data jsou často přijímána právě prostřednictvím několika transportních protokolů, jako jsou HTTPS, AMQP, MQTT nebo Kafka, čímž je umožněna obsluha široké škály zdrojů. Flexibilita v podpoře různých transportních protokolů zajišťuje, že systém může efektivně komunikovat s mnoha zařízeními a aplikacemi, což je nezbytné v heterogenních síťových prostředích a při implementaci komplexních IoT řešení.

3.3.2 Formát událostí

Události v ESP jsou základním stavebním prvkem, který umožňuje softwaru reagovat na různé akce nebo změny ve stavu. Tyto události jsou typicky strukturované a skládají se z několika klíčových částí [16]:

- **Klíč události:** Slouží k identifikaci a segregaci zpráv v rámci datového proudu. Klíč může být také použit pro určení, do které části v rámci úložiště bude událost zařazena.
- **Hodnota události:** Je hlavním obsahem události, která může obsahovat data ve formátech jako je obyčejný text, JSON, XML, nebo v různých binárních formátech. Hodnota je schopna reprezentovat širokou škálu dat, od jednoduchých událostí po složité strukturované objekty.
- **Časové razítko:** Označuje čas, kdy byla událost vytvořena nebo zpracována, což je důležité pro řazení událostí a zajištění správného pořadí v procesu zpracování událostí.
- **Metadata:** Zahrnují dodatečné informace, jako jsou například hlavičky, které mohou nést informace potřebné pro zpracování událostí. Těmi mohou být přídavné identifikátory, informace o zdroji události a jiné.
- **Pořadové číslo:** Jedná se o jedinečný identifikátor události v rámci úložiště, který umožňuje sledovat, které události byly již zpracovány. Pořadové číslo je klíčové pro správu stavu v distribuovaných systémech proudového zpracování.

Hlavičky	"original_client_id": 234222, "tracking_uuid": "AF13alkj21hl9fasl882\242fjhaiuhc"
Metadata	"offset": 23, "partition": 0, "topic": "Shopping.Cart.v2", "timestamp": 1660501516
Klíč	3125123
Hodnota	"cart_id": 3125123, "item_map": [(521,1), (923,3)], "shipping": "express"

Obrázek 4 Příklad formátu události. Podle: [16]

Serializace událostí

ESP jsou obvykle serializačně agnostické a přijímají jakákoli serializovaná data od textu čitelného pro člověka až po surové bajty. Existují však obecně přijímané strukturované datové formáty, které mohou práci s daty ve velké míře ulehčovat. Jedná se o formáty JSON, Avro a Protobuf, což jsou nejčastěji používané formáty v rámci proudového zpracování a jsou také nejvíce podporovány různými programovacími jazyky a knihovnami [17].

Formáty mají následující vlastnosti:

- **JSON:** JSON je textový formát založený na JavaScriptové syntaxi a je hojně používaný pro výměnu dat mezi webovým klientem a serverem. Jeho struktura je snadno čitelná pro lidi, což usnadňuje ladění a vývoj. Zároveň je podporován širokou škálou programovacích jazyků. Mezi další z výhod patří to, že nevyžaduje předem definované schéma, což umožňuje práci s dynamickými a rychle měnícími se strukturami dat [18].
- **Avro:** Avro je binární formát serializace vyvinutý v rámci projektu Apache, který podporuje schématickou serializaci dat. Používá schéma pro definování struktury dat, což zajišťuje kompatibilitu verzí a jednodušší

evoluci datových struktur. Je také účinnější v ukládání dat než textové formáty, což šetří místo a zvyšuje propustnost při přenosu dat [19].

- **Protobuf:** Protobuf, binární formát vyvinutý a podporovaný společností Google, byl navržen pro použití s širokou škálou programovacích jazyků a nabízí kompaktní reprezentaci dat, která efektivně šetří místo na disku a kapacitu síťového přenosu. Oproti textově orientovanému formátu JSON, Protobuf prioritizuje efektivitu úložiště a síťové propustnosti, čehož dosahuje pomocí svého binárního formátu. Dále se Protobuf vyznačuje silnou typovou kontrolou, což umožňuje přesné definování a vynucení datového schématu. Toto schéma poskytuje jasný a přesný popis struktury dat, což usnadňuje komunikaci mezi různými komponentami systému a zajišťuje konzistenci dat při jejich zápisu a čtení [20].

Při výběru formátu pro ESP je důležité zvážit specifické požadavky projektu, jako jsou požadavky na efektivitu přenosu dat, kompatibilitu mezi různými systémy, a snadnost práce s daty. Každý z těchto formátů má své silné a slabé stránky a je vhodný pro různé scénáře použití v rámci proudového zpracování dat.

3.3.3 Ukládání dat

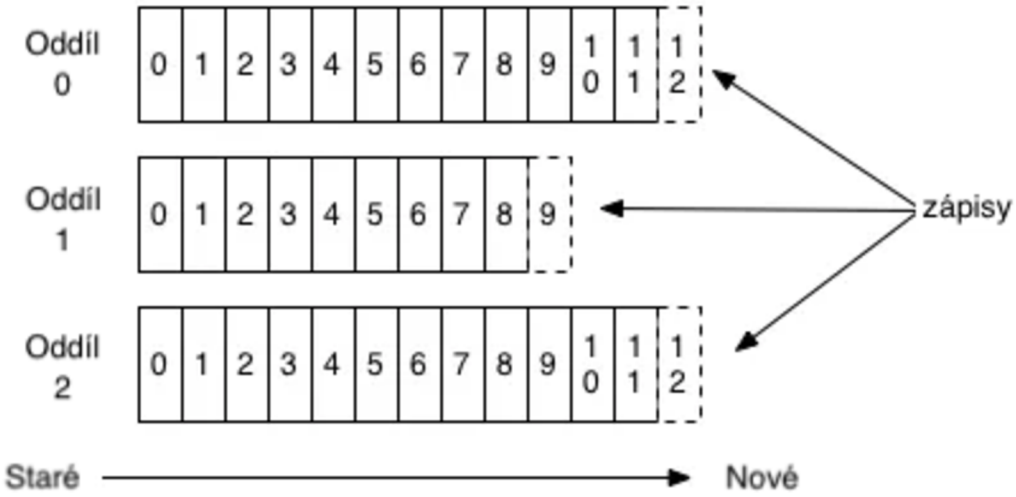
Ukládání dat v ESP je kritickou součástí, díky které je zajištěna vysoká dostupnost, odolnost a efektivita zpracování velkých datových proudů. Tato kapitola se zabývá klíčovými koncepty jako jsou *topics* (témata), *partitions* (oddíly) a replikace dat, které společně formují základní strukturu pro ukládání a správu dat v těchto platformách.

Témata a oddíly

V obecnosti jsou témata datovým tokem, do kterého jsou publikovány události se stejným byznysovým významem a jsou rozděleny na několik oddílů. Umožňují tak segregaci dat na základě obsahu nebo zdroje, což usnadňuje jejich organizaci a zpřístupnění pro specifické aplikace nebo služby.

Oddíly představují logický výsek nebo část záznamů daného tématu. Jak bylo již v kapitole 0 zmíněno, tak každá událost v oddílu má pořadové číslo, tzv. *offset,* což je pozice, na které je v oddílu událost uložena. Díky rozdělení na oddíly je umožněno

klientům provádět paralelní zpracování, což ve výsledku zvyšuje propustnost, a tak i efektivitu systémů [6].

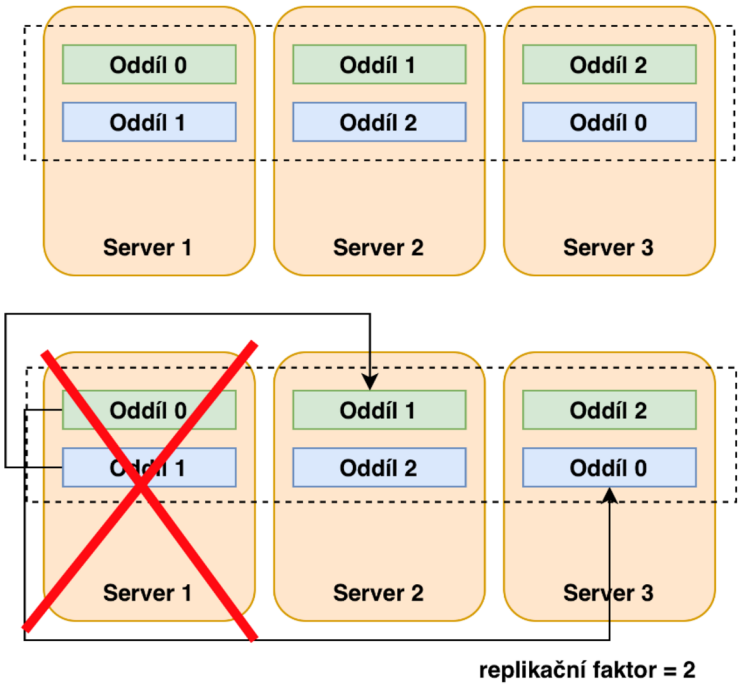


Obrázek 5 Architektura témat a oddílů. Podle: [21]

Replikace dat

Replikace dat je proces, při kterém jsou kopie oddílů distribuovány mezi různé servery nebo lokality. Tím se zvyšuje odolnost proti chybám, jelikož v případě selhání jednoho serveru může být datový tok obnoven z jiné kopie [21]. Přístup k tomu, jak jsou data replikována, se může lišit v závislosti na ESP, nicméně všechny známé distribuce touto možností disponují, jelikož je u distribuovaných systémů důležitá vysoká dostupnost.

U většiny ESP je pomocí replikačního faktoru definováno, na kolik serverů se mají data replikovat. Z obrázku níže (Obrázek 6) je zřejmé, jak se jednotlivé oddíly replikují vždy na jiný server. V případě výpadku *Serveru 1* je zajištěno, že data se nacházejí ještě na *Serveru 2* a *Serveru 3.*



Obrázek 6 Replikace dat v ESP. Podle: [22]

Retence dat

Retence dat je v ESP zásadním prvkem pro správu životního cyklu událostí. Tato retence odkazuje na dobu, po kterou jsou události uchovávány v systému před jejich automatickým odstraněním nebo archivací. Je důležitá pro regulaci kapacity úložiště a zajištění, že data jsou dostupná pro pozdější zpracování, analýzy, nebo dokonce pro účely obnovy po chybě [21]. V rámci platformy může být retence nastavena na úrovni jednotlivých témat, což umožňuje kontrolu nad uchováváním dat v závislosti na jejich významu. Retenční politiky mohou být konfigurovány buď na základě časového intervalu (například uchování událostí po dobu 7 dní) nebo na základě velikosti dat v tématu (například uchování posledních 100 GB dat), přičemž starší data jsou poté automaticky odstraňována. Efektivní nastavení retence je klíčové pro optimalizaci výkonu a správu zdrojů v ESP, přičemž musí být vyváženo mezi potřebami úložiště, přístupností dat a regulačními požadavky.

3.3.4 Konzumace událostí

Konzumace událostí v ESP je proces, při kterém aplikace (konzumenti) přijímají a zpracovávají data z témat. Konkrétně konzumenti přistupují k datům v jednotlivých oddílech témat. Mimo jiné mohou konzumovat data i z více oddílů najednou, což umožňuje flexibilitu v závislosti na potřebách aplikace. Přístup k datům je typicky sekvenční, což znamená, že konzument postupně čte data v pořadí, v jakém byla do oddílu zapsána [8].

Konzumentské skupiny

Pro efektivní a škálovatelnou konzumaci událostí se používají konzumentské skupiny, což je sada konzumentů, kteří společně konzumují data z jednoho tématu. Distribuce oddílů mezi konzumenty ve skupině umožňuje paralelní zpracování a zvyšuje celkovou propustnost. Každý oddíl je tak v daném čase přiřazen pouze jednomu konzumentovi v rámci skupiny, což minimalizuje problémy s duplicitním zpracováním dat [21]. To je možné vidět na Obrázek 3, kde je tento princip znázorněn.

Ukládání pořadových čísel konzumentů

Pořadové číslo konzumenta neboli *consumer offset,* představuje pozici konzumenta v oddílu, a proto je klíčové pro jeho správu. Umožňuje totiž pokračování v konzumaci od poslední zpracované události i v případě restartu nebo výpadku. Pořadová čísla jsou obvykle ukládána v centrálním úložišti nebo v rámci samotné ESP, o což se starají samotní konzumenti, a je tedy na nich, v jakých intervalech a s jakou četností jejich pozici ukládají.

3.3.5 Přehled komerčních a open-source platforem

Po obecném představení principů a významu ESP je v této části představen přehled konkrétních platforem dostupných na trhu. Tyto platformy se od sebe v mnoha aspektech odlišují, ale jsou postaveny na podobných principech a základech. Je tedy dobré se s platformami seznámit, jelikož různé typy implementací mohou být vhodné pro různé podnikové potřeby a scénáře. V následujících odstavcích tak budou podrobně rozebrány komerční, ale i open-source řešení a jejich rozdílné charakteristiky.

Apache Kafka

Apache Kafka je distribuovaná streamovací platforma umožňující ukládání velkého množství zpráv a nabízející vysokou dostupnost, propustnost a škálovatelnost [8]. Kafka klastr[2] se skládá z několika serverů (brokerů[3]), které se v případě správného nastavení při výpadku dokáží zastoupit, a hodí se proto pro řadu kritických případů, jako je třeba *messaging*, ukládání logů, zpracování proudů událostí, ukládání metrik a mnoho dalších [9].

Kromě samotných brokerů, na kterých Kafka uchovává data, tak lze do platformy připojit i další komponenty, které poskytují lepší integraci s okolními systémy, případně pomáhají se zpracováním přenášených dat. Originální distribuce Kafky přináší ještě nástroje jako jsou Kafka Connect a Kafka Streams [27].

Kafka Connect je open-source framework pro integraci dat s Apache Kafka. Poskytuje standardizované rozhraní pro vývoj konektorů, které umožňují přenos dat z různých zdrojů a do různých cílů skrze Kafku. Mezi nejznámější existující konektory dnes patří například JDBC konektor, Elasticsearch konektor, Amazon S3 konektor a mnoho dalších.

Kafka Streams je klientská knihovna pro vytváření aplikací a mikroslužeb, které pracují nad daty v Kafce, konkrétně nad nimi provádějí různé transformace, mapování, filtrace, obohacení a mnoho dalších operací [28].

Mimo originální distribuci Kafky existují ještě společnosti, které vyvinuly další nástroje sloužící pro lepší práci s Kafka platformou. Nejznámější a nejvyužívanější z těchto společností je Confluent, který v jejich platformě přináší ještě komponenty jako jsou Schema Registry, ksqlDB, Control Center a mnoho dalších [29]. Nutno podotknout, že ne všechny jejich nástroje jsou open-source a je tedy potřeba pro jejich použití licence.

Kafka klastr lze provozovat několika způsoby, včetně vlastního klastru nebo cloudové služby. V případě provozu vlastního klastru je nutné nainstalovat a nakonfigurovat Kafka brokery, Apache ZooKeeper a případně další komponenty

---

[2] V kontextu výpočetní techniky a informačních systémů je klastr (cluster) soubor vzájemně propojených počítačů, které pracují společně tak, aby byly vnímány jako jeden systém [22].
[3] Kafka broker je jeden ze serverů, který je součástí Kafka klastr, má unikátní identifikátor [9].

z ekosystému. Toto řešení poskytuje maximální kontrolu, ale vyžaduje také značné znalosti a údržbu.

Platformu lze však také provozovat v cloudu, a jelikož je platforma open-source, byla využita několika společnostmi, které nabízejí Kafku jako službu. Tento způsob nevyžaduje tolik odborných znalostí, avšak na druhou stranu přináší s sebou vyšší náklady na její provoz. Mezi největší poskytovatele cloudových řešení se řadí společnosti, jako jsou Confluent, Amazon, Redhat a Cloudera [26].

Azure Event Hubs

Azure Event Hubs je moderní ESP poskytovaná v rámci ekosystému Microsoft Azure. Jedná se o distribuovanou, plně spravovanou službu, která umožňuje přijímat, ukládat a zpracovávat obrovské množství událostí v reálném čase s nízkou latencí. Jelikož se jedná o produkt společnosti Microsoft, je tak možné ji bezproblémově integrovat s dalšími jejími nástroji, jako jsou Stream Analytics, Power BI a Event Grid. Obrovskou výhodou této technologie je, že poskytuje víceprotokolový engine, který nativně podporuje protokoly AMQP, Apache Kafka a HTTPS. To umožňuje širší použití, jelikož pro producenty a konzumenty není vyžadovaná specifická klientská knihovna pro danou platformu, ale mohou použít různé knihovny v závislosti na preferovaném protokolu [30].

Microsoft poskytuje Azure Event Hubs ve svém cloudovém řešení jako kompletně spravovatelnou *platformu jako službu*[4] s minimálními nároky na konfiguraci nebo správu.

Platforma disponuje všemi uvedenými obecnými vlastnostmi ESP, umožňuje však navíc dělit témata do jmenných prostorů, kterým lze nastavovat rozdílné konfigurace a úrovně zabezpečení v závislosti na povaze dat. Není zde ale možné nastavovat libovolně retenční dobu témat, jelikož závisí na prémiovosti služby, která je zákazníkem používána. Standardní verze platformy nabízí maximální retenční dobu 7 dní, prémiová a dedikovaná poté retenci až 90 dní [31].

---

[4] Platforma jako služba (Platform-as-a-Service, PaaS) je cloudový model, který poskytuje infrastrukturu a nástroje pro vývoj, testování a nasazení softwaru a aplikací. Správu. Poskytuje abstraktní vrstvu, která umožňuje vývojářům zaměřit se na tvorbu samotných aplikací.

Google Cloud Pub/Sub

Google Cloud Pub/Sub je plně spravovatelná, škálovatelná a globálně distribuovatelná *messaging service*. Stejně jako předchozí zmíněné platformy, umožňuje asynchronní komunikaci mezi aplikacemi s nízkou latencí a je užitečná pro analýzu a integraci velkého množství dat [33]. Kombinuje však možnosti horizontální škálovatelnosti podobné platformě Apache Kafka s možnostmi využití tradičních *messaging* přístupů, které nabízí technologie, jako jsou Apache ActiveMQ [34] nebo RabbitMQ [35]. Další rozdílnou vlastností od ostatních platforem, zmíněných v diplomové práci, je přístup k přijímání zpráv. Google Cloud Pub/Sub platforma pronajímá jednotlivé zprávy klientům-konzumentům, a poté sleduje, zda je daná zpráva úspěšně zpracována. Tento přístup maximalizuje paralelismus aplikací a pomáhá zajistit nezávislost produkující a konzumující strany. Díky vlastnímu přístupu k paralelismu tak platforma nedělí témata do oddílů tak, jak tomu je u předchozích zmíněných [36].

Platforma nabízí bohatou a různorodou sadu nástrojů a služeb, které lze použít k vytváření a provozování aplikací řízených událostmi. Poskytuje klientské knihovny v programovacích jazycích Python, Java, Go, C#, C++, JavaScript a Ruby, dále je možné platformu skvěle integrovat i s ostatními nástroji od Google, jako jsou například Cloud Functions [38], Cloud Storage [39], nebo dokonce Gmail [40]. V rámci integrací s produkty třetích stran je možné jej používat s nástroji Splunk, Datadog, Striim nebo Informatica [37].

Jak již název napovídá, tak služba je provozována v cloudu a je založena na modelu *pay-as-you-go*, což znamená, že uživatelé platí pouze za zdroje, které spotřebují. Sazby za odesílání, přijímání a ukládání zpráv se pak mohou lišit v závislosti na několika různých faktorech, jako například region, kvalita poskytované služby nebo velikost samotné zprávy.

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams je jednou ze čtyř částí nabízených v ekosystému Amazon Kinesis v rámci jedné z nejznámějších cloudových platforem Amazon Web Services (AWS). Jako ostatní platformy dokáže shromažďovat data v obrovském měřítku a zpřístupňovat je pro analýzu v reálném čase [41].

Architektura Kinesis Data Streams je velmi podobná ostatním ESP, nicméně i tady je třeba zmínit několik odlišností. Platforma nepoužívá témata, ale pouze oddíly, které však disponují určitými kapacitními omezeními v rychlosti čtení a zápisu. Každý oddíl podporuje až 5 transakcí za sekundu pro čtení (až do maximální celkové rychlosti čtení dat 2 MB za sekundu) a až 1 000 záznamů za sekundu pro zápis (až do maximální celkové rychlosti zápisu dat 1 MB za sekundu) [42]. Události v platformě mohou mít velikost maximálně 1 MB a maximální retenční doba dat je omezena na 365 dní, přičemž je možné ji získat pouze za poplatek.

Platforma je snadno integrovatelná s ostatními AWS produkty, jako jsou Amazon Connect, Amazon EventBridge, AWS Lambda a mnoho dalších. Dále také nabízí integraci přes svoje AWS SDK API, které je dostupné pro většinu známých programovacích jazyků. Mezi nejhlavnější patří C++, Java, JavaScript, Go, Python, .NET a PHP [43]. Dále je také integrovatelná s nástroji třetích stran, jako jsou Apache Flink, Apache Spark, Fluentd, Debezium, Kafka, Oracle GoldenGate, atd [43].

Amazon Kinesis Data Streams je nabízen v AWS cloudu a stejně jako předchozí platforma je založena na modelu *pay-as-you-go*. Kinesis Data Streams má dva kapacitní režimy: v prvním jsou zdroje na vyžádání a v druhém jsou rezervovány dopředu. V režimu kapacity na vyžádání je zpoplatněn každý GB zapsaných a přečtených dat z datových toků. V režimu rezervované kapacity je určen počet oddílů potřebných pro aplikaci na základě jejího počtu požadavků na zápis a čtení. Oddíl je pak jednotka kapacity, která poskytuje 1 MB/s zápisu a 2 MB/s čtení po celou dobu [45].

4 Zpracování proudů událostí

Zpracování proudů událostí je nedílnou součástí při práci s daty v reálném čase, jelikož po přenesení dat do platforem zmíněných v předchozí kapitole je potřeba data zpracovat, to znamená je obohatit, vyfiltrovat, nebo jinak upravit. Využívá se v široké škále aplikací, od monitorování webového provozu až po analýzu finančních transakcí, správu IoT zařízení a zpracování sociálních médií. Jedná se tak o techniku správy dat, která zahrnuje nepřetržitý příjem nových dat, nad kterými jsou tyto operace v reálném čase prováděny. Po zpracování jsou pak většinou data předána zpět aplikaci, datovému úložišti, nebo dalšímu nástroji pro zpracování dat. K těmto účelům se většinou používají další nástroje a knihovny, které tento typ práce nad daty umožňují. Mimo jiné existuje dnes již řada architektur, které nabízejí pravidla, jak se k takovým datům chovat a jakým způsobem je zpracovávat. Mezi další velká témata patří ošetření chybových stavů, jelikož při kontinuálním prouděním dat může být problematické se s chybami správně vypořádat. Tato kapitola si tak klade za cíl všechny zmíněné části blíže popsat a představit možnosti, kterými je možné se při vývoji vydat.

4.1 Událostmi řízené architektury

Událostmi řízené architektury (Event-driven Architectures, dále jen EDA) si pomocí sady různých návrhových vzorů kladou za cíl umožnit společnostem detekovat důležitá data pro jejich byznys, na která jsou pak schopné téměř v reálném čase reagovat. Tyto architektury tak častokrát nahrazují klasickou architekturu *požadavek/odpověď*, kdy služby musely čekat na odpověď, než přešly ke zpracování dalšího požadavku. Událostmi řízené architektury jsou tak často spojovány s asynchronní komunikací, kdy odesílatel a příjemce nemusí vzájemně čekat na přesun k dalšímu úkolu, ale tyto úkoly zpracovávají nezávisle na sobě. Příkladem mohou být dvě následující situace: telefonní hovor reprezentující synchronní komunikaci, namísto toho textová zpráva, která představuje komunikaci asynchronní. Při tomto typu komunikace je zpráva odeslána, aniž odesílatel ví, zda ji někdo čte, a tak nečeká na okamžitou odpověď.

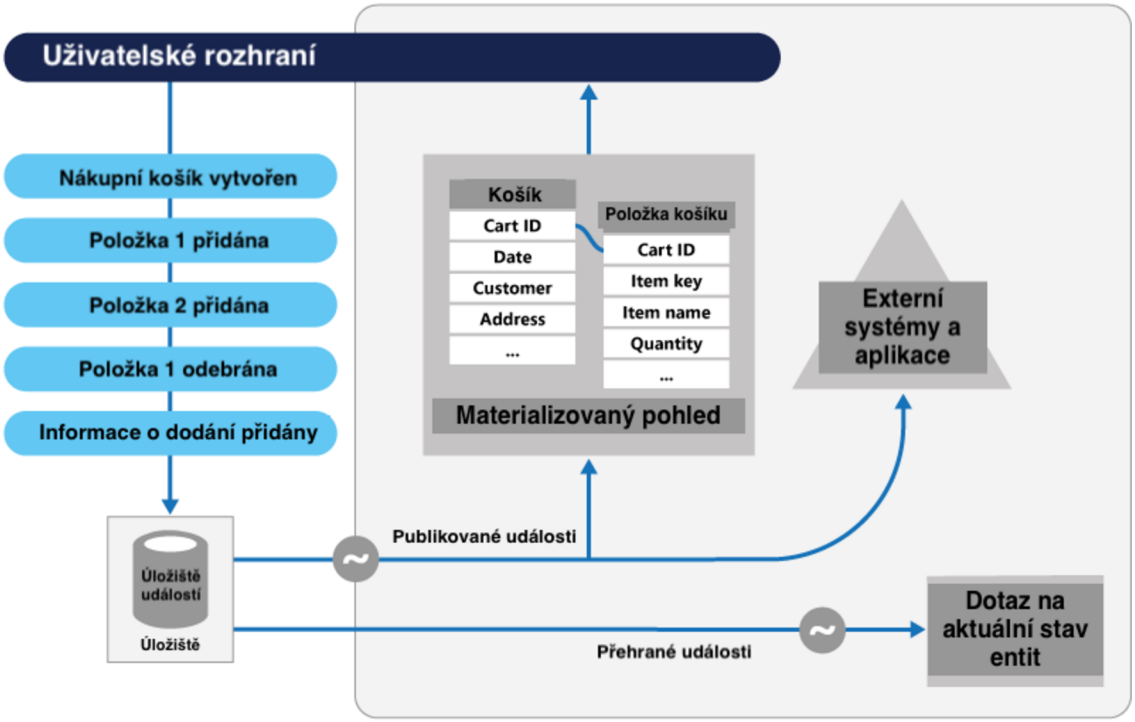
Ve zkratce tak tyto architektury nabízejí způsob budování IT systémů, které umožňují tok informací mezi aplikacemi, mikroslužbami a připojenými zařízeními v reálném čase podle toho, jak se v podniku objevují události [46].

4.1.1 Návrhové vzory

Pro funkční implementaci EDA v distribuovaném prostředí je nezbytné definovat několik nejznámějších návrhových vzorů, které nabízejí strukturovaný přístup k řešení problémů a pomáhají zajistit, aby byl systém robustní a efektivní. Dále budou zmíněny tři nejznámější vzory, nicméně jich dnes existuje mnohem víc. Rozhodnutí, jaké vzory pro systém použít, závisí zejména na povaze společnosti, aplikací a dat.

Získávání událostí (Event Sourcing)

Tradiční metody ukládání stavu systému obvykle ukládají pouze aktuální stav dat. Tento vzor naproti tomu zahrnuje ukládání všech změn, které způsobují změnu stavu, namísto stavu samotného. Události uložené ve vhodné platformě tak představují neměnné fakty o věcech, které se staly [47]. Tyto události jsou uloženy v pořadí, v jakém byly vytvořeny, a lze se na ně dotazovat, nebo je použít k odvození aktuálního stavu systému v libovolném časovém okamžiku.



Obrázek 7 Návrhový vzor: Získávání událostí. Podle: [48]

Vzor tak nabízí několik výhod [47]:

- **Historický stav:** Díky známosti všech předchozích stavů je možné obnovit systém v kterémkoliv okamžiku.
- **Auditovatelnost:** Historie umožňuje zjistit, kdo změny v systému realizoval.
- **Přehrání událostí:** Protože jsou všechny události uloženy, lze je přehrát a obnovit tak požadovaný stav. To je užitečné při testování, ladění nebo dokonce při přechodu na jinou stavovou strukturu.
- **Časový dotaz:** Je možné se ptát na stav systému v libovolném časovém okamžiku. Například: „Jaký byl stav nákupního košíku uživatele X k datu Y?“.

Mezi nevýhody patří [47]:

- **Změny struktury událostí:** V průběhu času se může struktura událostí měnit, nicméně zpracování těchto změn pak může být náročné, protože události jsou neměnné.
- **Rychlost obnovení stavu:** Pokud je událostí mnoho, může být obnovení stavu časově náročné.

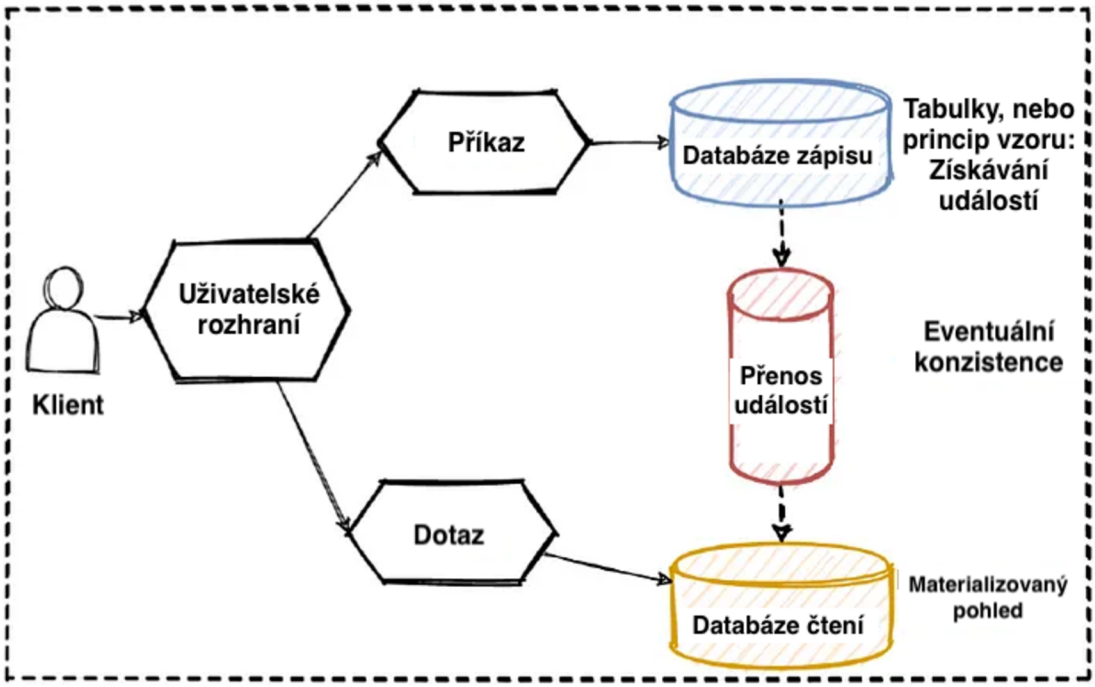
- **Komplexnost:** Implementace podle tohoto návrhového vzoru je poměrně složitá a komplexní, a proto je potřeba uvažovat, zda je u systému uchovávání kompletní historii nezbytné.

Segregace odpovědnosti příkazového dotazu (CQRS)

Návrhový vzor v systému odděluje operace čtení od zápisu, což umožňuje optimalizovat každou operaci zvlášť a zpracovávat ji různými modely nebo dokonce různými službami, čímž se zvyšuje výkonnost, škálovatelnost a bezpečnost systému [48]. V tradičním modelu CRUD (Create, Read, Update, Delete) se pro operace čtení i zápisu používá stejný datový model. V mnoha systémech, zejména složitých, však mohou být požadavky a výkonnostní charakteristiky pro čtení a zápis velmi odlišné.

CQRS to řeší vytvořením dvou samostatných modelů [50]:

- **Příkazový model:** Tento model zpracovává všechny operace zápisu. Je zodpovědný za udržování konzistence stavu systému a zahrnuje obchodní logiku pro ověřování a zpracování příkazů.
- **Model dotazů:** Tento model zpracovává všechny operace čtení. Je optimalizován pro dotazování, často s denormalizovaným datovým modelem, který je strukturován tak, aby podporoval specifické vzory čtení systému.



Obrázek 8 Návrhový vzor: CQRS. Podle: [51]

Vzor tak nabízí několik výhod [50]:

- **Optimalizace výkonu:** Díky rozdělení čtení a zápisu je možné obě strany optimalizovat nezávisle, což vede k nárůstu výkonu pro obě operace.
- **Škálovatelnost:** Každý model lze škálovat nezávisle na jeho vytížení. Pokud má systém například vysokou zátěž při čtení, ale nízkou při zápisu, lze modelu dotazů přidělit více prostředků.
- **Flexibilita:** Různé modely lze navrhnout tak, aby co nejlépe vyhovovaly potřebám různých provozů. Například model příkazů může být vysoce normalizovaný, aby se zabránilo redundanci dat, zatímco model dotazů může být denormalizovaný, aby podporoval efektivní dotazování.
- **Zabezpečení:** Oddělením příkazů a dotazů lze snáze použít různá pravidla zabezpečení pro čtení a zápis dat.

Mezi nevýhody patří [50]:

- **Nárůst komplexity:** Zavedení vzoru zvyšuje složitost systému. Vyžaduje údržbu dvou samostatných modelů, což může vést k většímu množství kódu a potenciálnímu výskytu chyb.

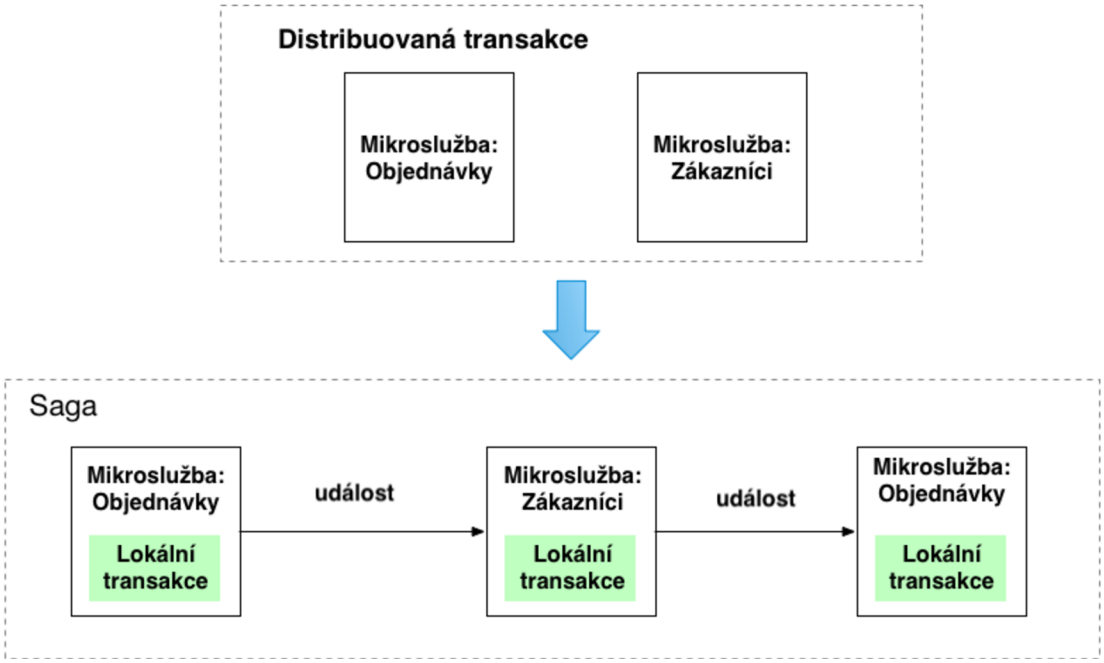
- **Synchronizace dat:** Zajištění synchronizace příkazového a dotazovacího modelu může být náročné, zejména v systémech s vysokou úrovní souběžnosti.
- **Eventuální konzistence:** V mnoha implementacích návrhového vzoru jsou model příkazů a model dotazů aktualizovány asynchronně, což vede ke stavu eventuální konzistence. To znamená, že může dojít ke zpoždění, než se změna provedená příkazem projeví v modelu dotazu.

Vzor ságy (Saga Pattern)

Tento návrhový vzor je vhodný pro řízení transakcí ve složitých distribuovaných systémech, jelikož zatímco v jednoduché, monolitické aplikaci je možné spravovat transakce napříč různými tabulkami nebo databázemi s cílem udržet konzistenci dat, v distribuovaném prostředí, kde jsou komponenty rozloženy napříč mnoha počítači, se správa transakcí stává komplexnější.

Sága přistupuje k tomuto problému tak, že rozkládá velkou transakci na několik menších, samostatných transakcí, kde se každá z těchto menších transakcí týká pouze dat uvnitř jedné specifické služby. Po dokončení každé z těchto lokálních transakcí se generuje událost, která spouští následující transakci v rámci ságy.

Pokud všechny tyto lokální transakce proběhnou úspěšně, je celá sága považována za úspěšně dokončenou. Nicméně, pokud některá z lokálních transakcí selže, sága zahájí řadu kompenzačních transakcí, jejichž účelem je vrátit systém do stavu před započetím ságy. Tím jsou anulovány všechny předchozí účinky transakcí, které již byly provedeny [52]. Tímto způsobem sága pomáhá udržovat konzistenci a stabilitu v distribuovaných systémech.



Obrázek 9 Návrhový vzor: Vzor ságy. Podle: [53]

Vzor tak nabízí několik výhod [52]:

- **Udržování konzistence dat:** Poskytuje mechanismus pro udržování konzistence dat ve více službách v distribuovaném systému řízeném událostmi.
- **Správa při selhání:** Pomocí kompenzačních transakcí zvládne sága účinně řešit selhání a výjimky, čímž pomáhá zajistit celkovou konzistenci systému.
- **Volné propojení služeb:** Každá služba může fungovat a vyvíjet se nezávisle, což zvyšuje flexibilitu a škálovatelnost systému.

Mezi nevýhody pak patří následující [52]:

- **Eventuální konzistence:** Vzor ságy vede ke konzistentnímu systému, nicméně v určité části procesu tomu tak být nemusí. To může být pro určitou povahu operací obtížně zvládnutelný koncept.
- **Návrh kompenzačních transakcí:** Návrh kompenzačních transakcí může být složitý. Musí efektivně zrušit změny předchozích transakcí a zároveň se vypořádat s možností, že některé z těchto změn již mohly být zpracovány jinými službami.

4.2 Nástroje pro zpracování proudů událostí

Jak již bylo psáno v úvodu kapitoly, nástroje pro zpracování proudů událostí jsou klíčové komponenty v architekturách zaměřených na data v reálném čase a událostmi řízené aplikace. Nástroje pomáhají podnikům zachytávat, zpracovávat, analyzovat data a reagovat na ně v reálném čase. Mechanismus zpracování událostí ve většině případů organizuje data ze vstupního zdroje do malých dávek, které následně v logické části aplikace zpracovává. V této části se definují transformace, které se poté aplikují na příchozí data. Ta se následně posílají do výstupního proudu, který je napojen například na databázi, nebo další platformu pro proudy událostí.

Aplikace pro zpracování událostí se typicky skládá ze čtyř částí:

- **Zdroj dat:** Zdrojem, odkud aplikace čte data, může být Kafka a jiné platformy, IoT senzory, databáze, nebo obyčejné soubory ze souborového systému.
- **Výstupní proudy:** Stejně jako u zdroje dat se jedná o platformy, kam jsou zpracované proudy zapisovány. Nástroje častokrát umí zapisovat do více zdrojů současně.
- **Procesní logika:** Zde nastává samotná práce s daty a provádění operací nad nimi.
- **Správa stavů:** Umožňuje udržovat informace o stavu zpracování, které mohou být využity například při havárii aplikace, nebo při potřebě provádět nad daty agregační funkce.

Nástrojů v dnešní době vzniká opravdu mnoho, práce se však zaměřuje pouze na tři vybrané, které mají otevřený kód, jsou tedy použitelné zdarma a v hodnocení statistiky Gartner [54] jsou mimo jiné na nejvyšších příčkách.

4.2.1 Kafka Streams

Kafka Streams představuje dynamickou a odolnou Scala/Java knihovnu pro práci s proudy událostí, která nabízí nástroje pro vytváření efektivních, odolných a škálovatelných aplikací pro zpracování kontinuálních datových proudů v reálném čase. Umožňuje tak vývojářům vytvářet pokročilé streamovací úlohy, analýzy na datech v reálném čase, transformace datových proudů a vytváření systémů založených na událostech. Kafka Streams tedy poskytuje širokou škálu funkcionalit pro komplexní a efektivní zpracování a analýzu datových toků [55].

Architektura

Kafka Streams aplikace využívá architekturu založenou na procesorové topologii, což je klíčový koncept pro definici logiky zpracování datových proudů. Její topologie se skládá z uzlů, známých jako procesory proudů, které jsou propojené pomocí hran (datových proudů) nebo sdílených úložišť stavů [58]. V rámci této topologie se rozlišují dva speciální typy procesorů:

- **Sink procesor:** Tento typ procesoru posílá zpracované záznamy z předchozích procesorů do konkrétního Kafka tématu, ale již nepředává data žádným dalším procesorům.
- **Source procesor:** Naopak Source procesor přijímá data přímo z jednoho nebo více Kafka témat a tyto záznamy pak předává následujícím procesorům v topologii.

Aplikace pro zpracování proudů může definovat jednu nebo více takových topologií, přičemž běžně se definuje jedna specifická topologie. Vývojáři mají možnost definovat topologie buď pomocí nízkoúrovňového procesorového API nebo prostřednictvím Kafka Streams DSL (Domain-Specific Language), který je postaven nad procesorovým API [56].

Příklad DSL API je možné vidět v následující ukázce kódu:

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, Long> numbers = builder.stream(
    "numbers-input-topic",
    Consumed.with(
    Serdes.String(),
    Serdes.Long()
    );
KStream<String, Long> onlyPositives = stream.filter((key, value) -> value >
0);
KStream<String, Long> onlyPositives = stream.filter(
    new Predicate<String, Long>() {
    @Override
    public boolean test(String key, Long value) {
        return value > 0;
    }
    });
```

Zdrojový kód 1 Kafka Streams DSL API. Podle: [56]

Příklad Processor API vypadá následovně:

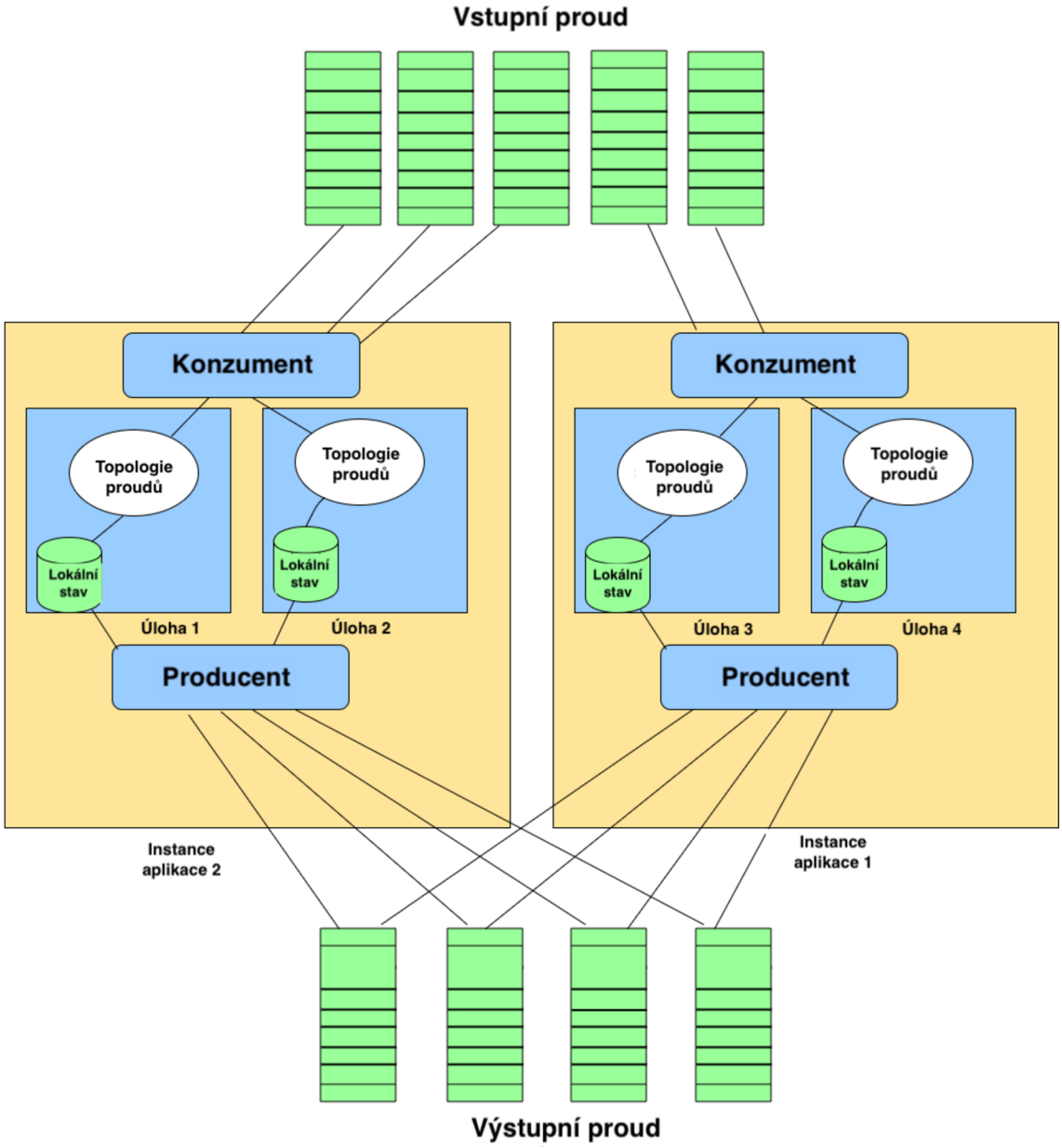
```
public class WordCountProcessor implements Processor<String, String,
String, String> {
    private KeyValueStore<String, Integer> kvStore;
    @Override
    public void init(final ProcessorContext<String, String> context) {
       . . .
    }
    @Override
    public void process(final Record<String, String> record) {
        final String[] words =
record.value().toLowerCase(Locale.getDefault()).split("\\W+");

        for (final String word : words) {
            final Integer oldValue = kvStore.get(word);

            if (oldValue == null) {
                kvStore.put(word, 1);
            } else {
                kvStore.put(word, oldValue + 1);
            }}}
```

Zdrojový kód 2 Kafka Streams Processor API. Zdroj: [57]

Topologie procesoru je tedy logickou abstrakcí pro kód zpracování datových proudů. Tato logická topologie je v době běhu aplikace instanciována a replikována pro paralelní zpracování dat, což umožňuje efektivní a škálovatelné zpracování proudů dat v reálném čase [55].



Obrázek 10 Architektura Kafka Streams aplikace. Podle: [59]

Knihovna pro tvorbu topologií také poskytuje sadu operací pro zpracování proudů událostí, které zahrnují [56]:

- **Filtrování:** Vybrání zpráv na základě určitých kritérií.

- **Transformace:** Změna datových záznamů na nový formát nebo hodnotu.
- **Agregace:** Souhrnné operace jako součty, průměry, minima/maxima.
- **Spojování (Joining):** Kombinace dat z více proudů podle klíčů nebo časových oken.
- **Okenní operace (Windowing):** Seskupení dat do časových oken pro realizaci časově omezených operací.
- **Skupinové operace (Grouping):** Seskupení zpráv podle klíčů pro následné agregace.
- **Stavové operace:** Uchování stavu mezi zpracováním jednotlivých záznamů, což umožňuje složitější zpracování jako je například sledování sekvencí událostí nebo rozpoznávání vzorů.
- **Interaktivní dotazy:** Přístup k stavu aplikace v reálném čase z vnějších aplikací.

Použití

Kafka Streams je efektivní nástroj pro zpracování velkých objemů datových proudů v reálném čase, ideální pro aplikace vyžadující rychlou odezvu a nízkou latenci. Je široce využívána v různých oborech od finančního sektoru, přes obchodování na burze, monitorování zásob, až po zdravotnické monitorování v reálném čase. Dále se uplatňuje v oblasti mikroslužeb, analýz a správy událostí [55]. Díky možnosti interaktivních dotazů se skvěle hodí také k implementaci návrhového vzoru CQRS.

Výhody a nevýhody

Hlavní výhody Kafka Streams zahrnují její těsnou integraci s Kafka platformou, což poskytuje efektivní zpracování datových proudů s vysokou propustností a nízkou latencí. Díky škálovatelnosti a odolnosti vůči chybám je tak ideální pro vývoj robustních, distribuovaných aplikací pro zpracování proudů dat. Díky podpoře jak stavových, tak nestavových operací umožňuje komplexní zpracování dat, včetně okenních operací a spojování proudů. Její programovací model je flexibilní a přístupný, což usnadňuje implementaci pokročilých analytických a transformačních úloh. Na rozdíl od systémů vyžadujících zřízení dedikovaného

klastru pro zpracování proudů, které může přinést významnou režii a složitost, Kafka Streams umožňuje integraci přímo do stávajících Java aplikací [55].

Na druhou stranu, Kafka Streams může být náročnější na zdroje v porovnání s jinými streamovacími nástroji, zejména v situacích, kde je požadováno zpracování velkých objemů dat nebo složitých dotazů. Její implementace vyžaduje solidní porozumění ekosystému Kafka, což může být pro některé vývojáře bariérou. Navíc je Kafka Streams primárně zaměřena na práci s daty uvnitř ekosystému Kafka, což může omezovat její využití v architekturách, které vyžadují integraci s různými datovými zdroji a systémy.

4.2.2 Apache Flink

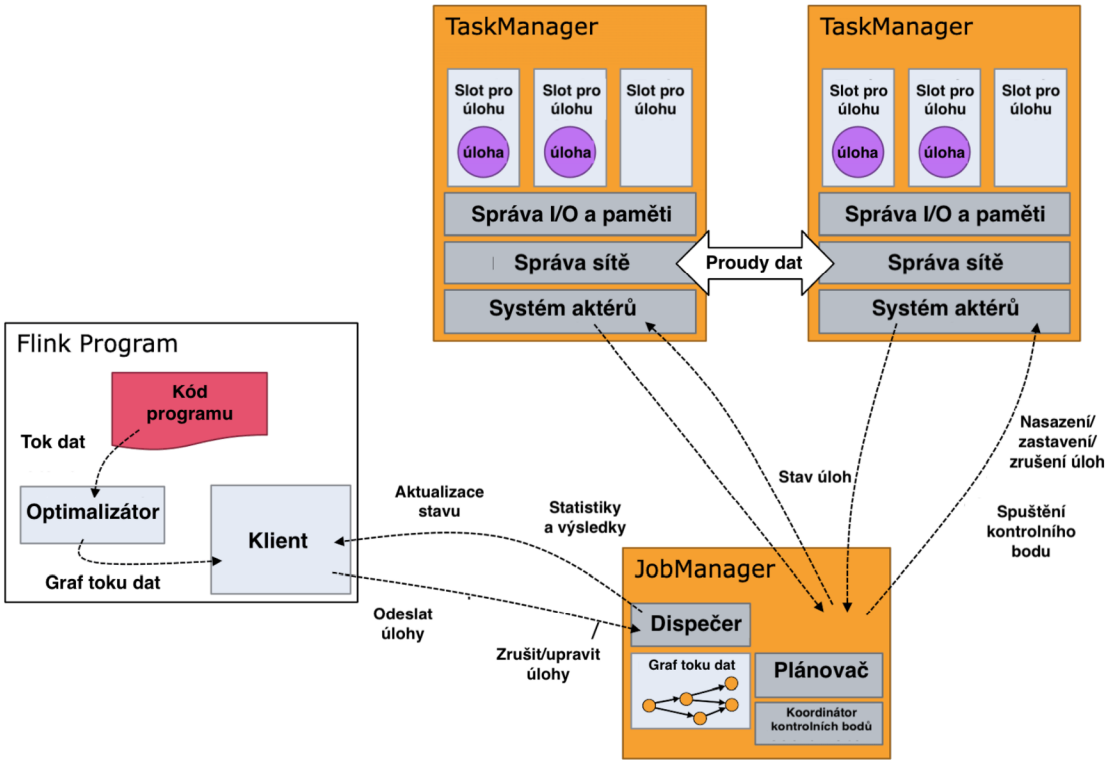
Apache Flink představuje open-source distribuovaný Java/Scala framework určený pro stavové zpracování datových sad s neomezeným (datové proudy) i omezeným (dávky) objemem dat. Aplikace pro zpracování datových proudů jsou koncipovány tak, aby běžely nepřetržitě s minimálními výpadky a zpracovávaly data v reálném čase, jakmile jsou nahrávána. Hlavními vlastnostmi Apache Flink jsou nízká latence zpracování, vysoká dostupnost a horizontální škálovatelnost. Mezi další významné rysy frameworku patří pokročilé řízení stavů, práce s časovými známkami událostí a se sofistikovaným zpracováním dat mimo pořadí a pozdními událostmi. Apache Flink byl vyvinut primárně s ohledem na zpracování datových proudů a nabízí sjednocené programové rozhraní pro zpracování datových proudů i dávek. Tímto způsobem poskytuje uživatelům flexibilitu a konzistenci při práci s různými typy datových úloh [60].

Apache Flink je implementací Kappa architektury, což je přístup, kde veškerá data jsou zpracovávána jako proudy v reálném čase. Klíčovým prvkem této architektury je jediný streamovací procesor, který zpracovává jak proudová, tak dávková data. Dávková data jsou v Kappa architektuře považována za speciální případ proudových dat. Ve srovnání s tradiční Lambda architekturou, která používá oddělené procesory pro dávkové a proudové zpracování dat a vyžaduje oddělené kódové základy, Kappa architektura tento problém řeší tím, že má pouze jeden pohled, a to reálný čas. Tím se eliminuje potřeba slučování různých kódových

základen a zjednodušuje se celý proces. Tato vlastnost činí Flink vhodným pro aplikace, kde je potřeba rychlé a efektivní zpracování dat v reálném čase [63].

Architektura

Zatímco Kafka Streams je knihovna, která běží jako součást aplikace, Flink je samostatný engine pro zpracování datových toků a je nasazován nezávisle. Flink aplikace je spouštěna ve Flink klastru, který se skládá z hlavního uzlu (JobManager), který plánuje úlohy, a několika vedlejších uzlů (TaskManager), na kterých samotné úlohy běží. Klastr je integrován se všemi běžnými správci klastrových zdrojů, jako jsou Hadoop YARN a Kubernetes, ale lze jej také nastavit tak, aby běžel jako samostatný klastr nebo pro neprodukční a testovací prostředí i jako knihovna [62].



Obrázek 11 Architektura Apache Flink klastru. Podle: [62]

Podobně, jako Kafka Streams, tak také Flink poskytuje vývojářům různé úrovně abstrakce pro vývoj aplikací. Nejnižší úroveň abstrakce umožňuje stavové a časově závislé zpracování proudů, nabízí konzistentní, odolný stav a podporuje sofistikované výpočty se zpětnými voláními založenými na událostech a zpracování

času. Pro mnohé aplikace, které nepotřebují tuto nízkoúrovňovou abstrakci, je k dispozici DataStream API, dostupné z programovacích jazyků Java, Scala a Python. Ty poskytují běžné stavební bloky pro zpracování proudů, jako jsou transformace, spojení, agregace a časová okna. Nejvyšší abstrakcí je však podpora SQL, kde Flink umožňuje uživatelům využívat standardní SQL dotazy pro interakci s proudovými a dávkovými daty [64].

Příklad DataStream API je znázorněn na následujícím kódu:

```
DataStream<Transaction> transactions = env
.addSource(new TransactionSource())
    .name("transactions");

DataStream<Alert> alerts = transactions.keyBy(Transaction::getAccountId)
    .process(new FraudDetector()).name("fraud-detector");

alerts.addSink(new AlertSink()).name("send-alerts");

env.execute("Fraud Detection");
```

Zdrojový kód 3 Apache Flink DataStream API. Podle: [65]

Příklad definici Flink úlohy pomocí SQL pak může vypadat následovně:

```
INSERT INTO
    crd_transactions_large_eur
SELECT
    *
FROM
    crd_transactions
WHERE
    transactionCurrency = 'EUR'
    AND transactionAmount > 5000;
```

Zdrojový kód 4 Apache Flink SQL API. Zdroj: [autor]

Použití

Apache Flink je používán k vytváření různých typů aplikací pro zpracování datových proudů i dávek díky svému širokému spektru funkcí. Mezi běžné typy aplikací, které využívají Apache Flink, patří [61]:

- **Aplikace řízené událostmi:** Tyto aplikace zpracovávají události z jednoho nebo více datových proudů a provádějí výpočty, aktualizaci stavu nebo externí akce na základě těchto událostí. Důležitou vlastností je schopnost provádět stavové zpracování, což umožňuje implementovat logiku, která závisí na historii přijatých událostí. Tímto způsobem mohou aplikace provádět pokročilé transformace a rozhodnutí na základě kontextu. Mezi typické aplikace řízené událostmi patří detekce podvodů, detekce anomálií, varování založená na pravidlech, monitorování obchodních procesů a mnoho dalších.
- **Aplikace pro datovou analytiku:** Tyto aplikace slouží k extrakci informací a poznatků z dat. Namísto tradičního dotazování se nad omezenými datovými sadami, kde se dotazy buď opakovaně spouštějí nebo se aktualizují v souladu s novými daty, Flink provádí analýzu kontinuálním způsobem.
- **Aplikace pro datové toky:** Tento druh aplikací se specializuje na transformaci a obohacení dat, která se přesouvají z jednoho úložiště dat do druhého. Tradičně se proces ETL (Extract-Transform-Load) provádí periodicky, v dávkách. S Apache Flink může proces probíhat nepřetržitě, což umožňuje rychlý přenos dat s nízkou latencí na jejich konečné místo určení.

Výhody a nevýhody

Mezi hlavní výhody lze u Flinku zařadit jeho robustní architekturu s širokým spektrem funkcí, která ho činí vhodným pro komplexní zpracování datových proudů. Jeho schopnost sjednoceně zpracovávat omezené a neomezené proudy dat umožňuje integrovat dávkové a proudové zpracování do jednoho systému. Flink nabízí sofistikovanou správu stavu, podporuje ukládání kontrolních bodů pro případ havárie klastru nebo aplikace a umožňuje zpracování událostí s garantovanou konzistencí *přesně jednou*. Dále poskytuje několik úrovní API, která vývojářům poskytují flexibilitu potřebnou pro řešení běžných i velmi specializovaných případů použití při zpracování proudů dat.

Na druhou stranu, složitost architektury Flink může být náročná na učení a provoz, což představuje výzvy i pro zkušené odborníky. Vývojáři a operátoři se

často potýkají se složitostmi, jako jsou vlastní vodoznaky, serializace, evoluce typů atd. Flink může představovat obtíže při nasazování úloh a operacích v klastru, včetně ladění výkonu s ohledem na výběr hardwaru a charakteristiky úlohy.
Organizace, které používají Flink, tak často potřebují dva týmy expertů, jeden se věnuje vývoji úloh a druhý se věnuje udržování operačního stavu klastru. Z tohoto důvodu je Flink používán především ve velkých organizacích s komplexními a pokročilými potřebami zpracování datových proudů [62].

4.2.3 Apache Spark Streaming

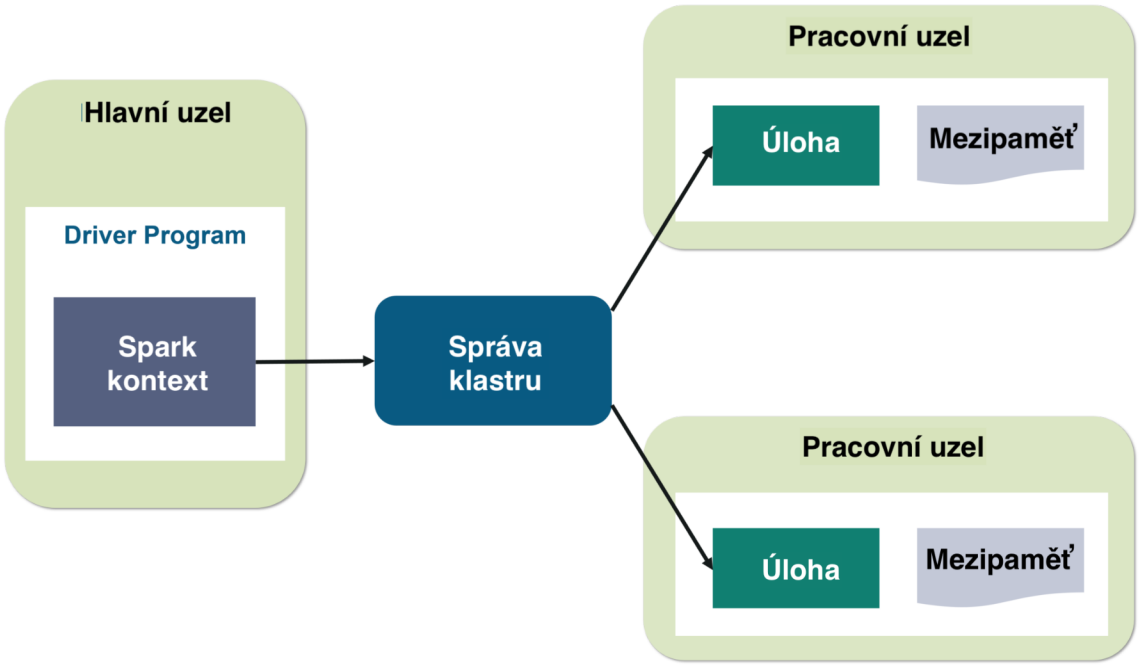
Apache Spark je open-source distribuovaný systém určený pro zpracování velkých dat. Jeho hlavní předností je využití ukládání dat v paměti a možnost manuální optimalizace vykonávání dotazů, což může pomoci při potřebě rychlých analytických dotazů na data jakéhokoli rozsahu. Spark poskytuje API pro jazyky Java, Scala, Python a R, čímž podporuje opětovné použití kódu pro různé typy úloh, jako je dávkové zpracování, interaktivní dotazy, analýza dat v reálném čase, strojové učení a zpracování grafů [66].

Apache Spark pro práci s proudy událostí používá Apache Spark Streaming, což je speciálně navržená komponenta pro zpracování datových proudů v reálném čase. Komponenta využívá koncept, kde jsou datové proudy rozdělené do mikrodávek. Ty umožňují systému zpracovávat proudová data v krátkých intervalech, což poskytuje iluzi kontinuálního proudu. Díky tomu mohou organizace zpracovávat a analyzovat data okamžitě, jakmile jsou dostupná, což je klíčové pro aplikace vyžadující rychlé rozhodování nebo okamžitou reakci. Spark Streaming podporuje širokou škálu zdrojů dat, včetně Kafka, Flume a Amazon Kinesis a umožňuje export zpracovaných dat do různých systémů, jako jsou souborové systémy, databáze nebo živé dashboardy [67].

Architektura

Podobně jako architektura Apache Flink, tak i Apache Spark je založen na master/slave módu, který je optimalizován pro distribuované zpracování velkých datových sad. V této architektuře hlavní uzel, známý také jako *Driver Program*, plní roli řídícího centra. Tento uzel se stará o plánování úloh, distribuci

práce mezi pracovní uzly a správu celkového stavu aplikace. Hlavní uzel interpretuje uživatelský kód a rozkládá jej na jednotlivé úlohy, které mají být provedeny. Na druhé straně jsou pracovní uzly odpovědné za provedení přidělených úloh. Každý pracovní uzel zpracovává část dat a vrací výsledky zpět do hlavního uzlu. Tato decentralizovaná architektura umožňuje efektivní zpracování rozsáhlých datových sad tím, že rozkládá výpočetní zátěž a datovou analýzu na více uzlů v klastru. Výsledkem je vysoký výkon a škálovatelnost, což je klíčové pro zpracování velkých dat v reálném čase a pro komplexní analytické úlohy [68].



Obrázek 12 Architektura Apache Spark klastru. Podle: [68]

Apache Spark Streaming přistupuje ke zpracování datových proudů unikátním způsobem, rozděluje totiž proudová data na velmi malé mikrodávky a nezpracovává tak jednotlivé záznamy samostatně. Toto rozdělení dat do milisekundových mikrodávek umožňuje paralelní příjem a ukládání dat do paměti pracovních uzlů Sparku. Data jsou pak zpracovávána výkonným enginem Sparku, který dokáže zpracovat tyto mikrodávky během milisekund a výsledky předávat do dalších systémů. Narozdíl od tradičních modelů s nepřetržitým zpracováním, v Spark Streaming jsou úlohy dynamicky přiřazovány pracovním uzlům na základě umístění

dat a dostupných zdrojů. Tento přístup vede k lepší distribuci zátěže a rychlejšímu zotavení z chyb.

V rámci Spark Streaming architektury se každá mikro dávka zpracovává jako „Resilient Distributed Dataset“ (RDD), což je základní abstrakce Sparku pro odolné distribuované datasety. Díky RDD mohou být proudová data zpracována pomocí libovolné knihovny nebo kódu Sparku, což dává vývojářům velkou flexibilitu a možnosti integrace s širokou škálou funkcí a nástrojů dostupných v Sparku [69].

Obrázek 13 Proces zpracování událostí u Spark Streaming. Podle: [67]

Jak již bylo zmíněno v úvodu, tak Spark poskytuje širokou škálu API, pomocí kterých je možné aplikace vytvářet. Kromě knihoven pro jazyky Python, R, Javu a Scalu nabízí také přístup pomocí SQL, stejně jako Flink, a navíc ještě poskytuje knihovnu MLlib pro strojové učení.

Příklad Java API pro Spark vypadá následovně:

```
JavaInputDStream<ConsumerRecord<String, String>> messages =
    KafkaUtils.createDirectStream(
    streamingContext,
    LocationStrategies.PreferConsistent(),
    ConsumerStrategies.<String, String> Subscribe(topics, kafkaParams));
JavaPairDStream<String, String> results = messages
  .mapToPair(
      record -> new Tuple2<>(record.key(), record.value())
  );
JavaDStream<String> lines = results
  .map(
      tuple2 -> tuple2._2()
  );
```

Zdrojový kód 5 Spark Streaming Java API. Zdroj: [70]

Příklad Spark SQL vypadá takto:

```
spark.sql(""" SELECT  country, city, zipcode, state FROM ZIPCODES
          WHERE state in ('PR','AZ','FL') order by state """)
     .show(10)
```

Zdrojový kód 6 Spark SQL API. Zdroj: [71]

Použití

Apache Spark, především tedy jeho komponenta Spark Streaming, se stala klíčovým nástrojem pro řadu předních společností, jako jsou Zendesk, Uber, Netflix a Pinterest, využívajících jeho schopností pro analýzu dat v reálném čase. Uber například používá Spark Streaming k analýze dat z mobilních aplikací, kde vytvářejí strukturovaná data z neuspořádaných událostí prostřednictvím ETL procesu s využitím Kafky, Spark Streaming a HDFS. Podobně Pinterest využívá Spark Streaming pro zpracování dat z Kafky a získává okamžitý přehled o interakcích s piny na celém světě, což mu pomáhá vylepšovat doporučení uživatelům v reálném čase. Netflix pak využívá Spark Streaming pro vývoj doporučovacího systému pro své filmy a seriály, který pracuje s miliardy událostí denně.

Význam Spark Streaming spočívá ve schopnosti společností zpracovávat a získávat hodnotu z rostoucího objemu dat v reálném čase. Od online transakcí po

sociální sítě a senzorová data, existuje potřeba neustálého monitorování a okamžité reakce na data. Spark Streaming umožňuje paralelní zpracování dat v klastru a výsledky pak poskytuje do dalších systémů jako Cassandra, HBase nebo Kafka. Tento systém je založen na modelu s nepřetržitými operátory, kde každý uzel zpracovává data a předává je dalším operátorům v pipeline, což umožňuje efektivní zpracování
od příjmu dat až po jejich uložení a analýzu [69].

Výhody a nevýhody

Jednou z hlavních výhod je rychlost zpracování, protože díky své in-memory technologii umožňuje rychlé zpracování datových sad. Dále je Spark známý svou snadností použití, poskytující intuitivní API ve více programovacích jazycích, jako jsou Scala, Java, Python a R. Spark také umožňuje zpracovávat různé typy úloh od dávkového zpracování až po streamování a strojové učení.

Na druhé straně skrývá i několik úskalí. První z nich je absence automatického optimalizačního procesu, což může vyžadovat dodatečné úsilí od vývojářů pro optimalizaci výkonu. Dále také trpí problémy s řízením souborového systému, což může být problematické při práci s velkým množstvím malých souborů. Dalším omezením je menší množství algoritmů ve srovnání s jinými systémy strojového učení. Navíc Spark nemusí být ideální pro některé aplikace pracující s proudy událostí a jejich okénkovými operacemi [72].

4.3 Problém ošetření chybových stavů

Při zpracování proudů událostí může nastat situace, kdy danou událost nebude z nějakého důvodu možné zpracovat. Důvody, které se v tomto případě dějí, se většinou týkají problému deserializace události, kdy událost není v požadovaném formátu. Také nedostupný výstup, do kterého se snaží aplikace zapsat, může být problémem. Dále mohou být důvodem problémy s integrací na externí systémy, síťové problémy, duplikace událostí atd. Příčinou přerušení proudů událostí může být mnoho, proto si tato kapitola klade za cíl popsat základní principy, jak se v takových situacích chovat.

Ošetření duplikací zpráv

Ošetření duplikací zpráv v proudových aplikacích je klíčovým aspektem pro zajištění integrity a přesnosti dat. Duplikace mohou vznikat z různých důvodů, jako jsou síťové chyby, chyby ve zpracování, nebo jako výsledek opakovaných pokusů o doručení zprávy. Pro ošetření duplikací zpráv se využívají následující mechanismy [73]:

- **Idempotence:** Idempotentní zpracování znamená, že opakované zpracování té samé zprávy nezmění stav systému více než jednou. To je zajištěno tak, že systém rozpozná již zpracovanou zprávu a ignoruje všechny její další instance.
- **Deduplikace na základě unikátních identifikátorů:** Každá zpráva může mít přiřazen unikátní identifikátor (např. UUID nebo kombinace atributů, které ji jednoznačně identifikují). Systém pak může sledovat, které identifikátory již byly zpracovány, a duplikáty tak efektivně odstranit.
- **Sledování pořadových čísel:** Systémy jako Kafka ukládají zprávy do logu s garantovaným pořadím. Konzumenti mohou sledovat poslední pořadové číslo, které zpracovali, a tím se vyhnout zpracování duplicitních zpráv.
- **Transakční zpracování:** Některé technologie umožňují použít transakční zpracování pro zajištění, aby výsledky byly aplikovány pouze jednou, i když dojde k opakovanému zpracování zpráv.

Mechanismy znovuzaslání zpráv

Mechanismy znovuzaslání zpráv jsou důležitou součástí systémů, zvláště v případech, kdy dochází k dočasným chybám v síti nebo v komunikačních systémech. Zajišťují tak, že zprávy nejsou ztraceny a že jsou doručeny i v případě výskytu problémů. Mezi hlavní mechanismy znovuzaslání zpráv patří [74]:

- **Automatické znovuzaslání:** Jedná se o funkci, kde systém automaticky opakuje pokus o zaslání zprávy, pokud nedošlo k úspěšnému doručení. Tento mechanismus je často doplněn o časový limit nebo maximální počet pokusů o znovuzaslání. Dalším krokem může být exponenciální navyšování intervalů mezi pokusy znovuzaslání, což pomáhá snížit zátěž na systém a síť.

Automatické zaslání může být implementováno přímo v aplikaci, kde se opakují pokusy během zpracování. Tento způsob však blokuje průchodnost dalších zpráv. Další možnou implementací je zpětné zaslání zprávy do zdrojového systému, ze kterého se po nějaké době zpráva znovu přečte a zkusí zpracovat. Tento způsob tak neblokuje průchod zpráv, nicméně není vhodný pro případy, kdy je potřeba zachovat pořadí zpráv.

- **Dead-Letter queue (DLQ):** Pokud se zprávy ani po několika pokusech nepodaří zpracovat, používají se tzn. DLQ, což jsou fronty, či témata, do kterých se ukládají zprávy pro pozdější analýzu, nebo manuální zásah operátorů.

Přeskočení zpráv

Do zpracovávaného proudu událostí se běžně může nedopatřením dostat zpráva, která není validní, nebo ji nelze deserializovat. To může v aplikaci způsobovat výjimky a běh aplikace se tak zastaví. U zpracování tohoto typu dat to však není žádoucí a je třeba se s tím nějak vypořádat. Tento mechanismus se stará o to, aby systém takové zprávy přeskočil a případný záznam o chybě zapsal vhodně do logu.

Zastavení zpracování

Tento způsob ošetření chyb není tak běžný, nicméně se může stát, že aplikace zpracovává data takové povahy, že při chybě není možné pokračovat ve zpracování. V tomto případě by aplikace měla zpracování zastavit a vyvolat chybu, která upozorní pomocí vhodných monitorovacích nástrojů operátory, aby chybu odstranili.

5 Analýza a návrh testových kritérií

Jak již bylo v úvodu práce zmíněno, cílem je porovnat představené technologie pro zpracování proudů událostí a nabídnout tak ucelený pohled pro jejich využití v různých situacích. To však nelze bez předchozího vytyčení jasně stanovených požadavků a metrik, které se mají u nástrojů porovnávat a sledovat. Tato kapitola si tedy klade za cíl analyzovat a navrhnout, jakým způsobem bude testování prováděno a na co bude zaměřeno, včetně přesné definice testovacích scénářů.

5.1 Popis problému

Pro porovnání byly vybrány nástroje Kafka Streams, Apache Flink a Apache Spark Streaming, přičemž každý z nástrojů přistupuje ke zpracování proudů událostí odlišným způsobem, a proto je třeba najít společná kritéria, podle kterých bude možné ověřit vhodnost pro daný testovací scénář.

Při popisu testovacích metrik a kritérií je důležité rozlišovat mezi kvantitativními a kvalitativními metrikami. Kvantitativní metriky jsou měřitelné a lze je vyjádřit číselně, příkladem může být latence, propustnost, nebo využití zdrojů. Tyto metriky při stejných testovacích podmínkách poskytují objektivní a měřitelné hodnoty, které umožňují přímé srovnání mezi různými nástroji.

Naopak kvalitativní metriky jsou subjektivnější a nejsou snadno měřitelné pomocí konkrétních číselných hodnot. Může to být například hodnocení funkcionality, dostupnost a kvalita dokumentace, nebo třeba náročnost na vývoj. Tyto aspekty jsou hodnoceny na základě uživatelských zkušeností a specifických potřeb pro daný testovací scénář.

Obě kategorie metrik jsou však klíčové pro komplexní hodnocení a porovnání těchto nástrojů. Kvantitativní metriky poskytují jasná a měřitelná data pro objektivní porovnání, zatímco kvalitativní metriky umožňují hlubší porozumění uživatelské zkušenosti a celkové vhodnosti nástroje pro konkrétní použití. Je důležité, aby byl při porovnávání nástrojů vzat v úvahu správný poměr obou typů metrik, aby byl zajištěn komplexní a vyvážený pohled.

5.2 Stanovení metrik

Jak bylo popsáno výše, pro komplexní porovnání je potřeba stanovit jak kvantitativní, tak kvalitativní metriky. U kvantitativních metrik poté bude stanovena vhodně zvolená stupnice, pomocí které bude možné hodnotit.

5.2.1 Kvantitativní metriky

Kvantitativní metriky poskytují objektivní měření výkonnosti a efektivity testovaných systémů. Níže jsou detailně popsány vybrané metriky:

Latence událostí

Tato metrika měří dobu potřebnou pro přenos události ze zdrojového kanálu do cílového. V kontextu zpracování proudů událostí je vždy nižší latence považována za lepší, neboť znamená rychlejší doručení událostí. V aplikacích pracujících v reálném čase, jako jsou VoIP, hry nebo obchodování na burze, je minimální latence klíčová pro uživatelskou spokojenost a systémovou reaktivitu.

- **Výpočet:** Latence se měří jako časový interval mezi okamžikem, kdy byla událost vygenerována (odeslána) ze zdroje, a okamžikem, kdy byla přijata na cílovém bodě. To se obvykle provádí zaznamenáním časových razítek na obou koncích a jejich následným porovnáním.
- **Jednotka:** Latence se typicky vyjadřuje v milisekundách (ms) nebo sekundách (s), v závislosti na požadavcích aplikace.

Propustnost událostí

Metrika hodnotí schopnost systému zpracovávat nebo přenášet maximální množství událostí v daném časovém úseku. Vyšší propustnost je žádoucí, jelikož indikuje schopnost systému efektivně zvládat větší objemy dat bez ztráty výkonu. To je obzvláště důležité v aplikacích vyžadujících vysokou míru datové komunikace, jako jsou datová centra, streamovací služby a distribuované databáze.

- **Výpočet:** Propustnost se měří jako počet událostí zpracovaných za jednotku času. Může být spočítána zaznamenáním počtu událostí přenesených nebo zpracovaných v systému a následným vydělením časového úseku, ve kterém tyto události proběhly.

- **Jednotka:** Propustnost se vyjadřuje v událostech za sekundu.

Zatížení CPU serverů

Měří procentuální využití výpočetní kapacity procesoru serveru během zpracování požadavků. Nižší využití CPU je preferováno, protože naznačuje, že server pracuje efektivně a má rezervy pro zpracování dalších úloh bez zvýšení odezvy nebo snížení výkonu.

- **Výpočet:** Zatížení CPU se měří pomocí nástrojů pro monitorování výkonu, které zaznamenávají využití procesoru během testování. Hodnota je obvykle uváděna jako procento maximální kapacity procesoru používané během zpracování požadavků.
- **Jednotka:** Zatížení CPU se vyjadřuje v procentech (%).

Zatížení paměti serverů

Odkazuje na procento využití operační paměti serveru. Stejně jako u CPU, nižší využití paměti je lepší, jelikož ukazuje, že systém efektivně spravuje své zdroje a může zvládat další aplikace nebo služby bez nutnosti rozšiřování hardwarových zdrojů.

- **Výpočet:** Podobně jako zatížení CPU, i zatížení paměti se měří monitorovacími nástroji, které sledují, kolik operační paměti je využíváno v průběhu testu. Výsledek je vyjádřen jako procento celkové dostupné paměti využívané aplikací nebo procesem.
- **Jednotka:** Zatížení paměti se také vyjadřuje v procentech (%).

5.2.2 Kvalitativní metriky

Mezi kvalitativní metriky, které lze porovnávat v rámci jednotlivých testovacích scénářů, byla vybrány následující:

Náročnost na vývoj

Tato metrika hodnotí, jak snadné nebo obtížné je v dané technologii vyvíjet. Zohledňuje aspekty jako jsou počáteční nastavení, složitost rozhraní, dostupné abstrakce a podpora pro vývojové nástroje.

Dostupnost a kvalita dokumentace

Metrika hodnotí dostupnost, úplnost a srozumitelnost dokumentace a vzdělávacích materiálů pro každý nástroj. Zahrnuje i přístupnost komunitní podpory a příkladů kódu.

5.2.3 Stupnice hodnocení

Při výběru stupnice pro hodnocení kvalitativních metrik je důležité navrhnout systém, který umožňuje jednoznačné a konzistentní hodnocení. Dle [75] [76] bude pro účely hodnocení náročnosti na vývoj a kvality dokumentace použita pětibodová stupnici, kde bude každý stupeň představovat jasně definovaný popis úrovně. Při hodnocení kvalitativních metrik je důležité zajistit, aby byla stupnice používána konzistentně, což znamená, že každý bod stupnice musí mít jasně definovaná kritéria, která umožní provádět objektivní a konzistentní posouzení.

V níže uvedených tabulkách jsou prezentovány pětibodové stupnice, které byly pro účely hodnocení v diplomové práci navrženy:

Náročnost na vývoj		
Hodnocení	**Bodové skóre**	**Popis**
Vynikající	5	Vývoj je velmi snadný a intuitivní, minimální požadavky na znalosti a úsilí.
Dobré	4	Nastavení a vývoj jsou jednoduché, intuitivní a s nízkými požadavky na znalosti a úsilí.
Střední	3	Průměrná náročnost na nastavení a vývoj, vyžaduje běžnou úroveň znalostí a úsilí.
Špatné	2	Vývoj je obtížný s častými problémy, vyžadující hlubší porozumění technologii.
Velmi špatné	1	Vývoj je extrémně obtížný, vyžaduje značné úsilí a specifické znalosti.

Tabulka 1 Stupnice hodnocení pro náročnost na vývoj. Zdroj: [autor]

Dostupnost a kvalita dokumentace		
Hodnocení	**Bodové skóre**	**Popis**
Vynikající	5	Dokumentace je snadno dostupná, komplexní, aktuální a velmi dobře strukturovaná.
Dobré	4	Dokumentace je snadno dostupná, dobře strukturovaná, úplná a aktualizovaná.
Střední	3	Dokumentace je dostupná, ale může obsahovat mezery nebo není plně aktualizovaná.
Špatné	2	Dokumentace je obtížně dostupná, částečně neúplná nebo obsahuje nepřesnosti.
Velmi špatné	1	Dokumentace je velmi obtížně dostupná nebo zcela chybí, je neúplná nebo nesrozumitelná.

Tabulka 2 Stupnice hodnocení pro dostupnost a kvalitu dokumentace. Zdroj: [autor]

5.2.4 Omezení rozsahu testování

V rámci této diplomové práce je důležité identifikovat a definovat omezení, která budou mít vliv na rozsah testování nástrojů, jelikož cílem je zajistit, že rozsah práce zůstane realistický a zvládnutelný, přičemž bude poskytnut jasný a konzistentní základ pro porovnání vybraných technologií. Zde jsou specifikována klíčová omezení testovacího procesu:

- **Škálovatelnost a vysoká dostupnost:** Testy se budou zaměřovat na menší datové sady a budou prováděny v prostředí s omezeným počtem uzlů. Rozsáhlé škálování, důraz na vysokou dostupnost a hodnocení výkonnosti při zpracování velkých datových objemů nebudou součástí této práce.
- **Bezpečnostní aspekty:** Hlavním cílem testů je evaluace výkonnostních a funkčních vlastností nástrojů. Bezpečnostní prvky, jako je odolnost vůči útokům nebo implementace bezpečnostních protokolů, nebudou v rámci této práce posuzovány.
- **Detailní funkcionality:** Testování se soustředí na klíčové vlastnosti a základní funkcionalitu nástrojů. Rozsáhlé prozkoumávání pokročilých nebo

specifických funkcí, které nejsou přímo relevantní pro základní porovnání, nebude zahrnuto.

- **Integrace se třetími stranami:** Testování se nebude věnovat integraci testovaných nástrojů s externími aplikacemi nebo službami třetích stran. Bude zaměřeno čistě na vnitřní funkcionalitu a výkon bez zkoumání interoperability, která by mohla být relevantní v širším nasazení a integraci do složitějších systémových architektur.
- **Ekonomické faktory:** Finanční aspekty související s nasazením a provozem testovaných nástrojů, včetně analýzy nákladů, nebudou v této práci hodnoceny.

Tato omezení jsou zásadní pro interpretaci výsledků a jejich objektivní hodnocení. Je klíčové tyto hranice jasně respektovat během celého testovacího procesu, aby bylo zajištěno, že závěry a doporučení budou relevantní.

5.3 Návrh testovacích scénářů

Návrh testovacích scénářů je v kontextu diplomové práce klíčovým prvkem, který si klade za cíl podat podrobný přehled o tom, jakým způsobem budou jednotlivé testy realizovány. To zahrnuje výběr podpůrných nástrojů, definici testovacích dat, specifikaci testovacího prostředí a samotný návrh jednotlivých testovacích případů.

5.3.1 Výběr podpůrných nástrojů

V rámci komplexních testů je klíčové definovat nástroje, které budou sloužit jako základ pro testovací prostředí a procesy. Pro proudy událostí je třeba zvolit některou z platforem z kapitoly 3.3, dále je nutné použít nástroj pro generování dat a několik monitorovacích nástrojů, které poskytnou sadu požadovaných metrik.

Pro účely práce byly vybrány tyto nástroje:

- **Platforma pro proudy událostí:** Pro účely práce bude použita Apache Kafka, jelikož se jedná o open-source řešení a zároveň je v této oblasti jednou z nejpoužívanějších a nejznámějších technologií [82], z tohoto důvodů se její použití jeví jako nejvhodnější.

- **Generování dat:** Pro simulaci reálných datových proudů bude využit již zmiňovaný Kafka Connect, který nabízí open-source Datagen konektor [77]. Tento konektor poskytuje možnost generování realistických datových proudů, které mohou napodobovat široké spektrum aplikací a scénářů použití. Konkrétně umožňuje definovat schémata a konfigurace pro simulaci různých typů událostí, jako jsou logy, transakce nebo události ze sociálních sítí, a to s různými frekvencemi a objemy. S jeho využitím lze snadno nastavit a spustit produkci zpráv do Kafka *topics*, což umožňuje efektivní testování zpracování a přenosu dat v reálném čase.
- **Sběr metrik:** Pro sběr metrik o propustnosti a latenci událostí bude použit nástroj Kafka CLI (rozhraní příkazového řádku) [78], který poskytuje informace o časech, kdy byla událost zapsána do vstupních a výstupních Kafka *topics*. Tato data následně poslouží k výpočtu obou zmiňovaných metrik. Pro sběr systémových metrik o zátěži CPU a vytížení paměti budou použity nástroje Prometheus [79], Grafana [80] a NodeExporter [81], které umožní získání procentuální vytíženosti za určité časové období pro porovnání.
- **Analýza dat:** Pro datovou analýzu o propustnosti a latenci budou implementovány vlastní Python skripty, které z nasbíraných dat vypočítají vhodné metriky, jako jsou průměrné, minimální a maximální hodnoty, medián a percentily, pomocí nichž bude možné jednotlivé technologie porovnat.

5.3.2 Datové objekty

Pro testování byly vybrány dva datové objekty, které nabízí v základu Datagen konektor. Tyto objekty simulují realistické pracovní prostředí s informacemi o zaměstnancích a jejich umístěním v rámci organizace. Datové modely objektu vypadají následovně:

Atribut	Typ	Popis
employee_id	int	Jednoznačný identifikátor zaměstnance.
first_name	string	Křestní jméno zaměstnance.
last_name	string	Příjmení zaměstnance.
age	int	Věk zaměstnance.
ssn	string	Sociální pojištění číslo zaměstnance.
hourly_rate	int	Hodinová sazba zaměstnance.
gender	string	Pohlaví zaměstnance.
email	string	E-mailová adresa zaměstnance.

Tabulka 3 Datový model – Zaměstnanec. Podle: [77]

Atribut	Typ	Popis
employee_id	int	Jednoznačný identifikátor zaměstnance.
lab	string	Označení laboratoře nebo pracoviště.
department_id	int	Identifikátor oddělení.
arrival_date	int	Datum příchodu zaměstnance, vyjádřený jako časové razítko.

Tabulka 4 Datový model – Umístění zaměstnance. Podle: [77]

Propojení mezi těmito dvěma modely je založeno na atributu *employee_id*, což umožňuje simulovat a testovat spojovací operace, které jsou pro zpracování proudů událostí nedílnou součástí.

5.3.3 Popis testů

V rámci testovacích scénářů budou pokryty čtyři hlavní aspekty zpracování datových proudů: filtrace, agregace, obohacení a transformace dat [83]. Scénáře budou následně implementovány v již zmiňovaných nástrojích vybraných pro porovnání.

Jednotlivé testy budou vypadat následovně:

- **Test filtrace dat:** Během tohoto scénáře dojde k použití filtrů na proud událostí s cílem selektivně extrahovat záznamy dle specifických atributů, v tomto případě pohlaví (gender) a hodinové sazby (hourly_rate). Tento test

bude hodnotit schopnost efektivně selektovat relevantní data z proudu událostí.

- **Test agregace dat:** Agregační test se zaměří na kumulaci údajů o počtu zaměstnanců v jednotlivých laboratořích. Tento proces bude demonstrací schopnosti agregovat data v reálném čase a poskytovat souhrnné statistické informace.
- **Test obohacení dat:** Test obohacení spočívá ve vytvoření nového datového objektu, který integrací informací z obou původních modelů na základě identifikátoru zaměstnance (employee_id) poskytne komplexnější pohled na dostupná data. Výstupem bude objekt obsahující jméno a příjmení zaměstnance, a název laboratoře, kde se aktuálně zaměstnanec nachází. Tímto testem se ověří schopnost nástroje provádět spojení datových proudů.
- **Test transformace dat:** Transformační test bude provádět převod hodinových sazeb na měsíční platy s přihlédnutím k pracovní době 160 hodin měsíčně. Test si klade za cíl posoudit flexibilitu a efektivitu nástroje v oblasti transformace datových atributů.

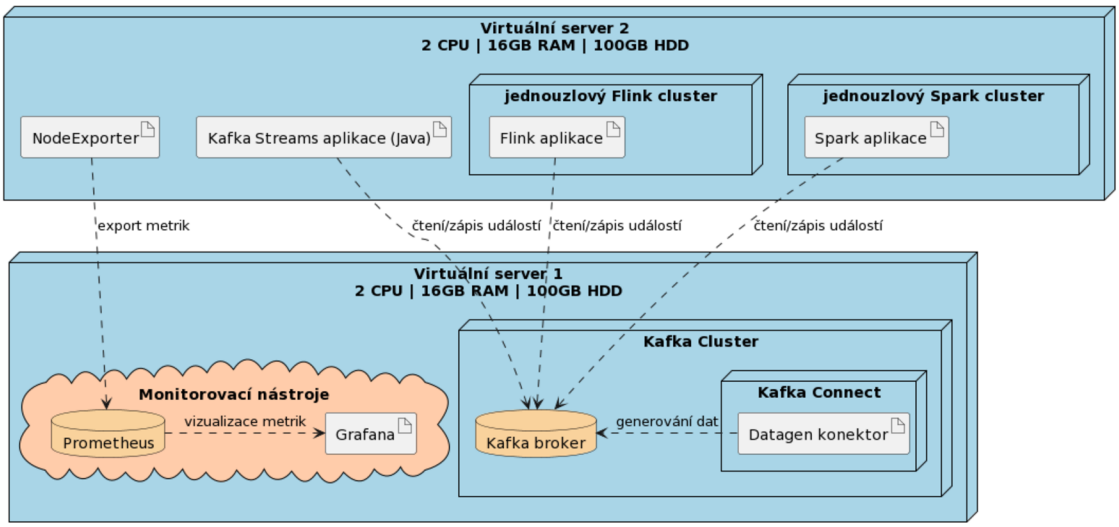
Výsledky těchto testů poskytnou detailní ucelený pohled na výkonnostní a funkcionální charakteristiky zkoumaných nástrojů, což umožní následné objektivní srovnání a hodnocení.

5.3.4 Návrh testovacího prostředí

V rámci návrhu testovacího prostředí diplomové práce je navrženo vytvoření dvou virtuálních serverů, kde každý bude vybavený 2 vCPU, 16 GB RAM a 125 GB pevným diskem. Tato konfigurace byla zvolena s cílem zajistit dostatečné výpočetní a paměťové kapacity pro efektivní běh testovaných aplikací a zároveň poskytnout standardizované prostředí pro všechny testy.

Na prvním serveru bude umístěn Apache Kafka klastr společně s monitorovacími nástroji Prometheus a Grafana, které budou shromažďovat a vizualizovat metriky výkonu. Tento server bude rovněž obsahovat Kafka Connect s Datagen konektorem pro generování testovacích datových proudů.

Druhý server bude sloužit pro spuštění jednotlivých nástrojů pro proudové zpracování: Apache Flink, Apache Spark a Kafka Streams, kde každý z nich bude testován samostatně, aby bylo zajištěno, že výsledky nebudou zkresleny současným během ostatních procesů. Kromě těchto aplikací je na serveru rovněž nasazen NodeExporter, který umožňuje sběr a export hardwarových metrik do systému Prometheus.



Obrázek 14 Diagram nasazení testovacího prostředí. Zdroj: [autor]

Celé testovací prostředí bude implementováno na domácím serveru, na kterém budou podle potřeby dynamicky vytvářeny virtuální servery s výše specifikovanými parametry. Tento přístup umožní efektivní využití dostupných zdrojů a zajistí, že každý testovací scénář bude probíhat v izolovaném a kontrolovaném prostředí, čímž se minimalizují potenciální vnější vlivy a zajistí se vysoká míra reprodukovatelnosti testů.

Výběr konkrétních verzí softwarových komponent a detailní nastavení jednotlivých klastrů a aplikací bude provedeno v následující fázi práce, konkrétně v části věnované implementaci.

6 Implementace a měření

V kapitole o implementaci a měření bude v první části popsáno, jakým způsobem testovací prostředí vznikalo, jakým způsobem byly jednotlivé komponenty nakonfigurovány a následně jak byly naimplementovány testovací scénáře v jednotlivých nástrojích. Druhá část kapitoly bude věnována interpretaci a evaluaci výsledků z měření včetně výběru nástroje, který si vedl v rámci testování nejlépe.

6.1 Tvorba testovacího prostředí

Pro vytvoření celého testovacího prostředí bylo využito několika nástrojů, které umožňují automatickou tvorbu virtuálních strojů a následné nasazení potřebných komponent. V následujících podkapitolách budou jednotlivé části popsány.

6.1.1 Tvorba virtuálních strojů

V procesu tvorby virtuálních serverů byl využit hlavní fyzický server s procesorem Intel Core i5-6300HQ 2.30GHz, který poskytuje solidní výkon pro virtualizaci a současně běh několika virtuálních strojů. Tento server běží na a operačním systému Ubuntu 20.04.6 LTS, je vybaven 40 GB operační pamětí RAM a 500 GB SSD úložištěm, což umožňuje efektivní a rychlou práci virtuálních strojů. Pro automatizaci vytváření a konfigurace virtuálních strojů byly použity nástroje VirtualBox a Vagrant, které společně poskytují flexibilní a spolehlivé řešení pro správu virtuálních prostředí.

VirtualBox [84] je open-source software pro virtualizaci, který umožňuje spouštět více operačních systémů nezávisle na hlavním systému. Nabízí rozsáhlé možnosti konfigurace virtuálních strojů, včetně nastavení velikosti paměti RAM, počtu CPU, velikosti a typu úložiště, síťových adaptérů a dalších. Díky uživatelsky přívětivému rozhraní a podpoře široké škály operačních systémů je VirtualBox vhodný pro vývoj, testování a demonstrační účely.

Vagrant [85] je nástroj pro automatizaci vytváření a správy virtuálních strojů, který umožňuje jednoduše definovat a konfigurovat virtuální prostředí pomocí jednoduchého textového souboru známého jako *Vagrantfile*. Tento soubor obsahuje

definice, jak by měl být virtuální stroj nastaven, včetně operačního systému, počtu alokovaných zdrojů (CPU, paměť), síťových nastavení a dalších. Vagrant umožňuje vývojářům a testerům rychle vytvářet a mazat izolovaná a reprodukovatelná prostředí, což zjednodušuje proces vývoje a testování.

```
Vagrant.configure("2") do |config|

servers = [
  {
    :hostname => virtual_server_1,
    :box => "generic/alma8",
    :ram => 16384,
    :cpu => 2,
    :ip => virtual_server_1_ip,
  },
  {
    :hostname => virtual_server_2,
    :box => "generic/alma8",
    :ram => 16384,
    :cpu => 2,
    :ip => virtual_server_2_ip,
  },
]
. . .
  servers.each do |machine|
    config.vm.define machine[:hostname] do |node|
        node.vm.box = machine[:box]
        node.vm.hostname = machine[:hostname]
        node.vm.network "private_network", ip: machine[:ip]
        node.vm.provider "virtualbox" do |domain|
          domain.memory = machine[:ram]
          domain.cpus = machine[:cpu]
          domain.name = machine[:hostname]
        end

        node.vm.provision "shell", inline: script
        node.vm.provision "shell", inline: scriptRhel

    end
  end
end
```

Zdrojový kód 7 Příklad konfigurace strojů ve Vagrantfile. Zdroj: [autor]

Pro virtuální stroje byl jako operační systém zvolen AlmaLinux [86]. Tato volba je strategická z toho důvodu, jelikož AlmaLinux poskytuje binární kompatibilitu s Red Hat Enterprise Linux (RHEL), což znamená, že všechny aplikace a konfigurace, které jsou kompatibilní s RHEL, mohou být bez problémů nasazeny i na AlmaLinux. Toto je zvláště cenné pro organizace a vývojáře, kteří hledají stabilní, bezpečné a bezplatné serverové prostředí, které je v souladu s ekosystémem RHEL.

6.1.2 Konfigurace komponent

Pro zajištění validního srovnání výkonnosti mezi nástroji je nezbytné, aby byla konfigurace jak procesorových, tak paměťových zdrojů pro každý nástroj sjednocena. Každý z těchto nástrojů byl konfigurován tak, aby měl k dispozici 8 GB operační paměti a 3 procesorová jádra. Pro Apache Flink se toto nastavení procesorů realizuje přes *taskmanager.numberOfTaskSlots: 3*, zatímco paměť je specifikována pomocí *taskmanager.memory.process.size: 8g*. V případě Apache Spark se pro přidělení procesorových jader používá *spark.executor.cores: 3* a pro paměť *spark.executor.memory: 8g*. Kafka Streams řídí paralelismus skrze *NUM_STREAM_THREADS_CONFIG: 3* a implicitně spravuje paměť JVM, ale je důležité zajištění dostatečného množství heap prostoru nastavením *-Xmx8g -Xms8g*. Tato konfigurace zajišťuje, že každý nástroj operuje v podobných podmínkách, což umožňuje přesné a férové srovnání jejich výkonnosti v testovacím prostředí.

6.1.3 Automatizované nasazování komponent

Pro automatizované nasazování komponent byl využit nástroj Ansible, který umožňuje definovat, jak by mělo být prostředí nakonfigurováno a nasazeno prostřednictvím tzv. *playbooků*, což jsou soubory ve formátu YAML popisující úlohy, které se mají na cílových serverech provést [87].

Pro každou komponentu byl v testovacím prostředí vytvořen vlastní Ansible *playbook*, který definoval potřebné kroky pro její instalaci a konfiguraci v souladu se schématem nasazení v kapitole 5.3.4.

```yaml
- hosts: spark_cluster
  vars:
    spark_version: "3.5.1"
    spark_url: "https://archive.apache.org/dist/spark/spark-3.5.1/spark-
3.5.1-bin-hadoop3.tgz"
  tasks:
    - name: Download Apache Spark
      get_url:
        url: "{{ spark_url }}"
        dest: "/tmp/spark-{{ spark_version }}-bin-hadoop3.tgz"
        timeout: 100

    - name: Unpack Apache Spark
      unarchive:
        src: "/tmp/spark-{{ spark_version }}-bin-hadoop3.tgz"
        dest: "/opt"
        remote_src: yes

    - name: Copy custom spark-env.sh
      copy:
        src: "spark-env.sh"
        dest: "/opt/spark-{{ spark_version }}-bin-hadoop3/conf/spark-
env.sh"
        owner: root
        group: root
        mode: '0644'
      become: yes

    - name: Start Spark master and worker
      shell: "/opt/spark-{{ spark_version }}-bin-hadoop3/sbin/start-all.sh"
      async: 10
      poll: 0

    - name: Verify Spark is running
      uri:
        url: "http://localhost:8080"
        method: GET
      register: result
      until: result.status == 200
      retries: 5
      delay: 10
```

Zdrojový kód 8 Ukázka Ansible playbooku pro instalaci Apache Spark. Zdroj: [autor]

6.1.4 Tvorba Kafka témat a generování dat

Témata v Kafka klastru byly vytvořeny automatizovaně v rámci tvorby celého Kafka pomocí Ansible *playbooků*, zatímco pro nasazení a konfiguraci Datagen

konektoru pro generování dat bylo využito REST API, které Kafka Connect poskytuje. Tento přístup umožňuje efektivní a dynamické nasazení konektorů bez nutnosti přímého zásahu do běžícího systému. Díky REST API lze konektory snadno přidávat, konfigurovat nebo odebírat, což poskytuje velkou flexibilitu při automatizaci testovacího procesu.

```
curl --location --request PUT
'http://virtualserver1:8083/connectors/EMPLOYEE-CONNECTOR/config' \
--header 'Content-Type: application/json' \
--data '{
    "connector.class":
"io.confluent.kafka.connect.datagen.DatagenConnector",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "false",
    "kafka.topic": "EMPLOYEES",
    "quickstart": "payroll_employee",
    "max.interval": 5,
    "iterations": 10000000,
    "tasks.max": "1"
  }'
```

Zdrojový kód 9 Konfigurace Datagen Connectoru. Zdroj: [autor]

6.1.5 Shrnutí

Testovací prostředí bylo vytvořeno pomocí automatizačních nástrojů s cílem dosáhnout vyšší přenositelnosti a snadnějšího vytváření na různých strojích a v různých konfiguracích. Tento přístup nejenže zajišťuje konzistenci a spolehlivost nasazení, ale také umožňuje provádět testování na větších klastrech. Díky automatizaci lze rychle a efektivně nasadit testovací prostředí na libovolný počet cílových strojů, což umožňuje testování aplikací v reálném prostředí a simulaci různých pracovních zátěží a scénářů použití.

6.2 Implementace testovacích scénářů

Tato část se zaměřuje na popis implementace vybraných testovacích scénářů z kapitoly 5.3.3 kontextu nástrojů Kafka Streams, Apache Flink a Apache Spark. Jejím cílem je ukázat, jakým způsobem lze v těchto nástrojích realizovat operace pro filtrace, agregace, obohacování a transformace dat. Konkrétní zdrojové kódy jsou součástí přílohy 9.1.

6.2.1 Filtrace dat

Filtrace dat je proces selektivního vybírání záznamů nebo událostí z velkého objemu dat na základě definovaných kritérií nebo podmínek. Cílem filtrace je snížit objem dat, která jsou zpracovávána, a soustředit se pouze na relevantní události pro daný účel nebo aplikaci.

V rámci Kafka Streams se provádí pomocí operace *filter*, která umožňuje selektivně zpracovávat záznamy na základě definovaných kritérií. Při implementaci filtrace je možné například vybírat záznamy dle specifických atributů objektů nebo hodnot klíčů.

```
KStream<String, Employee> employee = builder.stream(
        INPUT_TOPIC, Consumed.with(
                Serdes.String(), employeeSerde));

KStream<String, Employee> filteredEmployee = employee.filter(
        (key, value) -> "female".equals(
                value.gender) && value.hourlyRate > 15);

filteredEmployee.to(
        OUTPUT_TOPIC, Produced.with(
                Serdes.String(), employeeSerde));
```

Zdrojový kód 10 Ukázka filtrace dat v Kafka Streams. Zdroj: [autor]

Implementace filtrace ve Sparku a Flinku je pak velmi podobná, přičemž hlavní rozdíl spočívá v tom, jakým způsobem pracují s datovými proudy. Spark Streaming pracuje s mikrodávkami, což znamená, že data jsou zpracovávána v krátkých časových intervalech, zatímco Flink, stejně jako Kafka Streams pracuje s událostmi nebo záznamy v reálném čase bez konkrétního časového intervalu.

6.2.2 Agregace dat

Agregace dat obecně zahrnuje sběr a zpracování dat do souhrnných informací na základě určitých kritérií nebo operací. Operace umožňuje vytvářet skupiny dat na základě společného klíče a aplikovat na ně různé agregační funkce, jako je suma, průměr, minimum, maximum nebo počet záznamů ve skupině.

Implementace agregace v Kafka Streams se často provádí pomocí operací jako *groupByKey* a *aggregate*, které umožňují seskupení záznamů podle klíče a aplikaci agregační funkce na hodnoty ve skupině.

V Apache Spark je přístup k agregaci podobný, ale syntaxe a některé operace mohou být odlišné, což je vidět na ukázce zdrojového kódu níže.

```
Dataset<EmployeeLocation> employeeLocationDataset = spark
        .readStream()
        .format("kafka")
        .option("kafka.bootstrap.servers", config.getString(BROKERS))
        .option("subscribe", INPUT_TOPIC)
        .option("startingOffsets", "earliest")
        .load()
        .selectExpr("CAST(value AS STRING) as message")
        .select(functions.from_json(functions.col("message"),
schema).as("json"))
        .select("json.*")
        .as(Encoders.bean(EmployeeLocation.class));

StreamingQuery query = employeeLocationDataset
        .select("*")
        .groupBy("lab")
        .count()
        .selectExpr("lab as key", "CAST(count AS STRING) as value")
        .writeStream()
        .format("kafka")
        .option("checkpointLocation",
config.getString(CHECKPOINT_LOCATION))
        .option("kafka.bootstrap.servers", config.getString(BROKERS))
        .option("topic", OUTPUT_TOPIC)
        .outputMode("update").
        trigger(Trigger.ProcessingTime("0 seconds"))
        .start();
```

Zdrojový kód 11 Ukázka agregace dat ve Sparku. Zdroj: [autor]

6.2.3 Obohacení dat

Obohacení dat je proces integrování informací z různých zdrojů nebo datových proudů s cílem vytvořit komplexnější a užitečnější pohled na data. Tento proces může zahrnovat spojování dat pomocí společného klíče nebo identifikátoru a přidávání dalších atributů nebo informací, které obohacují původní data.

V kontextu Kafka Streams je obohacení dat často prováděno pomocí operace *leftJoin*, která umožňuje spojit dva datové proudy na základě společného klíče a obohatit jednu sadu dat pomocí informací z druhé sady.

```
KStream<String, Employee> employee = builder.stream(
INPUT_TOPIC, Consumed.with(
        Serdes.String(), employeeSerde));

KStream<String, EmployeeLocation> employeeLocation = builder.stream(
INPUT_LOCATION_TOPIC, Consumed.with(
        Serdes.String(), employeeLocationSerde));

KTable<String, Employee> employeeTable = employee
        .groupByKey(Grouped.with(Serdes.String(), employeeSerde))
        .reduce((oldValue, newValue) -> newValue);

KStream<String, EnrichedEmployee> enrichedEmployeeStream = employeeLocation
        .selectKey((employeeId, employeeLocationData) -> employeeId)
        .leftJoin(employeeTable, (employeeLocationData, employeeData) -> {
        if (employeeData != null) {
        EnrichedEmployee enriched = new EnrichedEmployee();
        enriched.employeeId = employeeData.employeeId;
        enriched.firstName = employeeData.firstName;
        enriched.lastName = employeeData.lastName;
        enriched.lab = employeeLocationData.lab;
        return enriched;
        } else {
        return null;
        }
        });

enrichedEmployeeStream.to(
OUTPUT_TOPIC, Produced.with(
        Serdes.String(), enrichedEmployeeSerde));
```

Zdrojový kód 12 Ukázka obohacení dat v Kafka Streams. Zdroj: [autor]

Apache Spark neumožňuje přímo transformovat průběžný datový proud do tabulky, jak je tomu u Kafka Streams, ale je nutné pro podobné obohacení dat pracovat s Dataset nebo DataFrame API a explicitně řídit logiku *joinu* a správu stavu, což může být méně intuitivní a vyžaduje to složitější správu závislostí a stavů. Spark sice poskytuje rozhraní pro *stream-stream joiny*, ale proces integrace a obohacování dat je zde méně pružný ve srovnání s Kafka Streams.

6.2.4 Transformace dat

V kontextu proudů událostí zahrnuje transformace dat aplikaci funkcí nebo operací na vstupní data s cílem vytvořit nová data, která odpovídají požadovanému formátu nebo struktuře.

Implementace transformace dat v Kafka Streams často zahrnuje použití operací jako *map*, *flatMap* nebo *transform*, které umožňují aplikovat uživatelsky definované funkce na jednotlivé záznamy.

Apache Spark a Apache Flink umožňují podobné funkcionality, rozdíl je opět pouze v syntaxi. Následující ukázka zobrazuje implementaci transformace v Apache Flink.

```
DataStream<Tuple2<String, EmployeeWithSalary>> transformedDataStream =
employeeDataStream.map(employeeTuple -> {
  EmployeeWithSalary transformedEmployee = new EmployeeWithSalary();
  double monthlySalary = employeeTuple.f1.hourly_rate *
WORK_HOURS_PER_MONTH;
  transformedEmployee.employeeId = employeeTuple.f1.employee_id;
  transformedEmployee.firstName = employeeTuple.f1.first_name;
  transformedEmployee.lastName = employeeTuple.f1.last_name;
  transformedEmployee.age = employeeTuple.f1.age;
  transformedEmployee.ssn = employeeTuple.f1.ssn;
  transformedEmployee.email = employeeTuple.f1.email;
  transformedEmployee.gender = employeeTuple.f1.gender;
  transformedEmployee.salary = monthlySalary;

  return new Tuple2<>(employeeTuple.f0, transformedEmployee);
}).returns(new TypeHint<>() {});
```

Zdrojový kód 13 Ukázka transformace dat v Apache Flink. Zdroj: [autor]

6.3 Sběr metrik

Pro efektivní měření výkonu zkoumaných technologií bylo klíčové pečlivě naplánovat sběr a analýzu relevantních metrik. Celý proces sběru dat byl strategicky rozdělen do dvou fází, aby byla zajištěna komplexnost a přesnost získaných informací. V první fázi byla pozornost soustředěna na shromažďování časových razítek událostí z Kafka témat, což umožnilo detailní sledování latence a propustnosti událostí v reálném čase. Druhá fáze se věnovala monitorování využití hardwarových zdrojů na virtuálním serveru, na kterém byly testy prováděny, s využitím systému Prometheus.

Pro obě fáze sběru dat byly implementovány specializované Python skripty. Tyto skripty byly navrženy tak, aby automatizovaně extrahovaly a transformovaly získaná data do strukturované formy Excelu ve formátu *xlsx*.

6.3.1 Sběr časů událostí

Pro získání časových razítek událostí byly využity Python skripty obsahující vhodnou knihovnu pro práci s Kafka platformou. Tyto skripty následně načítaly záznamy ze vstupních a výstupních Kafka témat, extrahovaly časová razítka generování a přijetí události a ukládaly je do výše zmíněného formátu *xlsx* pro další analýzu. Tento přístup umožnil opakovaně počítat latenci událostí a propustnost pro tisíce událostí bez větší námahy.

```
timestamps_input = []
timestamps_output = []
try:
    while True:
        msg = consumer.poll(timeout=1.0)
        if msg is None:
            break
        if msg.error():
            if msg.error().code() == KafkaException._PARTITION_EOF:
                continue
            else:
                print(msg.error())
                break
        if msg.topic() == 'EMPLOYEES:

            timestamps_input.append(msg.timestamp()[1])
        elif msg.topic() == 'EMPLOYEES_TRANSFORMED':
            timestamps_output.append(msg.timestamp()[1])
except KeyboardInterrupt:
    pass
finally:
    consumer.close()

df_input = pd.DataFrame(timestamps_input, columns=['Input Topic
Timestamp'])
df_output = pd.DataFrame(timestamps_output, columns=['Output Topic
Timestamp'])

df = pd.concat([df_input, df_output], axis=1)
df.to_excel('kafka_timestamps.xlsx', index=False)
```

Zdrojový kód 14 Sběr časových razítek pomocí Pythonu. Zdroj: [autor]

Měření latence bylo prováděno na datových tocích v reálném čase, zatímco propustnost byla hodnocena na základě zpracování předem připravené sady 50 000 událostí v Kafka tématu.

6.3.2 Sběr hardwarových metrik

Další část sběru metrik spočívala v extrakci informací o využití hardwarových zdrojů z monitorovacího nástroje Prometheus. Zde bylo využito REST API, které tyto metriky poskytovalo do souboru, z něhož opět Python skripty extrahovaly data do Excelu.

```
curl -G -fsS \
--data-urlencode 'query=avg((node_memory_MemTotal_bytes -
node_memory_MemAvailable_bytes) / node_memory_MemTotal_bytes) * 100' \
--data-urlencode 'start=1711565903' \
--data-urlencode 'end=1711485491' \
--data-urlencode 'step=1s' 'http://virtualserver1:9090/api/v1/query_range'
> input.json

curl -G -fsS
--data-urlencode 'query=(sum by (instance)
(irate(node_cpu_seconds_total{mode!="idle"}[5s])) / on(instance) group_left
sum by (instance) (irate(node_cpu_seconds_total[5s]))) * 100' \
--data-urlencode 'start=1711324800' \
--data-urlencode 'end=1711325098' \
--data-urlencode 'step=5s' 'http://virtualserver1:9090/api/v1/query_range'
> input.json
```

Zdrojový kód 15 Ukázka sběru hardwarových metrik. Zdroj: [autor]

```
import json
import pandas as pd

def process_data(input_file, output_file):
    with open(input_file, 'r') as file:
        data = json.load(file)

    values = data['data']['result'][0]['values']

    timestamps = [item[0] for item in values]
    percentages = [item[1] for item in values]

    df = pd.DataFrame({
        'Timestamp': timestamps,
        'Usage (%)': percentages
    })

    df.to_excel(output_file, index=False)
    print(f'Data successfully exported to {output_file}')

input_file = 'input.json'
output_file = 'output.xlsx'

process_data(input_file, output_file)
```

Zdrojový kód 16 Extrakce metrik do Excelu pomocí Pythonu. Zdroj: [autor]

6.3.3 Souhrn metrik

Ze všech získaných výsledků vyplynul komplexní Excel soubor, který slouží jako agregovaná databáze všech získaných informací. Soubor je rozdělen do více listů, přičemž každý list je věnován záznamům z jednotlivých testů specifického nástroje.

V rámci dokumentu byly pomocí příslušných funkcí vypočítány souhrnné statistiky pro každý test, zahrnující širokou škálu metrik, které poskytují komplexní pohled na výkon a efektivitu zkoumaných nástrojů. K těmto metrikám patří:

- **Průměrná latence:** Doba potřebná pro přenos události ze vstupního do výstupního tématu, což umožňuje hodnotit latenci zpracování v reálném čase.
- **Maximální a minimální latence:** Extremní hodnoty latence, poskytující vhled na nejlepší a nejhorší výkon nástroje.

- **Medián a percentilové hodnoty latence (95., 99., 99,5. percentil):** Statistiky ukazující distribuci latence, což pomáhá identifikovat variabilitu ve výkonu.
- **Propustnost:** Počet zpráv zpracovaných za sekundu, indikující schopnost systému efektivně zpracovávat vysoký objem dat.
- **Průměrné, minimální a maximální využití CPU:** Metriky v procentech, odhalující, jak intenzivně systém využívá procesorovou kapacitu.
- **Průměrné, minimální a maximální využití paměti:** Ukazatele v procentech, poskytující přehled o efektivitě využití paměťových zdrojů.
- **Počet záznamů:** Celkový počet událostí zpracovaných během testu, umožňující hodnotit rozsah datové sady.

6.4 Zhodnocení nástrojů

Tato kapitola poskytuje ucelený pohled na výkon a efektivitu zkoumaných nástrojů pro zpracování dat tak, jak bylo definováno výše v práci. Je zásadní pro porovnání jednotlivých nástrojů v různých aspektech výkonu, využití zdrojů a celkové efektivity při zpracování dat.

V podkapitolách níže je uvedený souhrnný pohled na výsledky testování, které jsou součástí přílohy 9.2.

6.4.1 Test filtrace dat

Metrika	Apache Flink	Apache Spark	Kafka Streams
Latence	5 ms	272 ms	63 ms
Propustnost	4545 událostí/s	4545 událostí/s	16667 událostí/s
Využití CPU	27,2 %	62,7 %	16,2 %
Využití paměti	12,5 %	13,9 %	11,2 %
Náročnost na vývoj	4 body	5 bodů	4 body
Dostupnost dokumentace	3 body	3 body	4 body

Tabulka 5 Souhrn výsledků testu filtrace. Zdroj: [autor]

Výsledky v Tabulka 5 naznačují, že Kafka Streams je vysoce efektivní ve filtraci velkých objemů dat s nízkou latencí a využitím zdrojů, což jej činí vhodnou volbou pro aplikace, které vyžadují vysokou propustnost a nízkou latenci. Apache Flink, přestože má menší propustnost než Kafka Streams, v testu měření latence exceloval a poskytuje tak solidní výkon s dobrým využitím zdrojů. Naproti tomu Apache Spark, přestože může být vhodný pro určité scénáře díky jeho flexibilitě a rozsáhlému ekosystému, se zdá být pro tento případ užití méně efektivní ve srovnání s Kafka Streams a Flinkem.

6.4.2 Test agregace dat

Metrika	Apache Flink	Apache Spark	Kafka Streams
Latence	---	---	---
Propustnost	7143 událostí/s	7143 událostí/s	1515 událostí/s
Využití CPU	2,5 %	18,9 %	6,3 %
Využití paměti	9,6 %	10,2 %	10,7 %
Náročnost na vývoj	3 body	3 bodů	4 body
Dostupnost dokumentace	2 body	3 body	5 body

Tabulka 6 Souhrn výsledků testu agregace. Zdroj: [autor]

Výsledky testu z Tabulka 6 ukazují, že Apache Flink a Apache Spark poskytují poměrně dobrou propustnost, Flink se navíc vyznačuje nízkým využitím CPU a paměti, což z něj dělá efektivní volbu pro dlouhodobé operace. Apache Spark, ač srovnatelný ve výkonu s Flinkem, má o něco vyšší nároky na zdroje, což může vyžadovat robustnější hardwarové konfigurace. Kafka Streams, i když má nižší propustnost v tomto konkrétním testu, se vyznačuje dobrým využitím zdrojů a vynikající dostupností dokumentace, což usnadňuje jeho vývoj.

Důvodem absence měření latence v tomto testu je povaha agregace dat, která zahrnuje procesy nezávislé na okamžitém přenosu jednotlivých zpráv, a proto je méně relevantní metrikou pro hodnocení výkonu v kontextu agregace.

6.4.3 Test transformace dat

Metrika	Apache Flink	Apache Spark	Kafka Streams
Latence	8 ms	267 ms	55 ms
Propustnost	2941 událostí/s	943 událostí/s	12500 událostí/s
Využití CPU	33,6 %	61,1 %	16,3 %
Využití paměti	12,4 %	13,9 %	11,1 %
Náročnost na vývoj	4 body	4 bodů	5 body
Dostupnost dokumentace	2 body	4 body	5 body

Tabulka 7 Souhrn výsledků testu transformace. Zdroj: [autor]

Podobně jako u filtrace dat, která má podobnou povahu, tak i v testu transformace dat Kafka Streams vykazuje výrazně lepší propustnost a nízkou latenci ve srovnání s ostatními nástroji. Nicméně nejnižší latence dosahoval opět Apache Flink, což ho řadí jako silného konkurenta pro transformační úlohy. Naopak Apache Spark se v tomto testu transformace objevuje s nižší propustností a vyšší latencí, což naznačuje jeho pomalejší reakci na požadavky transformace dat ve srovnání s ostatními nástroji. V rámci náročnosti vývoje tu stejně, jako ve všech předchozích testech exceloval Kafka Streams, který má výborně popsané jednotlivé funkcionality a existuje k němu i několik neoficiálních zdrojů.

6.4.4 Test obohacení dat

Metrika	Apache Flink	Apache Spark	Kafka Streams
Latence	3208 ms	42827 ms	1311 ms
Propustnost	5000 událostí/s	2174 událostí/s	5000 událostí/s
Využití CPU	13,1 %	52,6 %	11,2 %
Využití paměti	12,5 %	15,1 %	11,1 %
Náročnost na vývoj	2 body	3 bodů	4 body
Dostupnost dokumentace	2 body	2 body	4 body

Tabulka 8 Souhrn výsledků testu obohacení. Zdroj: [autor]

Z Tabulka 8 je jasně vidět, že v rámci testu obohacení Apache Spark nepředvedl tak dobré výsledky jako jeho konkurenti, má výrazně vyšší zpoždění, nižší propustnost a větší využití hardwarových zdrojů. Kafka Streams a Apache Flink naopak dosáhly srovnatelně vysoké propustnosti a oba měly nízké využití CPU, což naznačuje jejich vhodnost pro jejich použití při obohacování dat. Nicméně u nástroje Apache Flink nebylo úplně snadné funkcionalitu implementovat, a proto se zde řadí na poslední místo.

6.4.5 Shrnutí

Ze všech předchozích výsledků je patrné, že Kafka Streams exceluje v kategoriích týkajících se efektivity využití zdrojů a propustnosti, což částečně reflektuje jeho specificky zaměřenou integraci s Kafka platformou. Nástroj byl navržen speciálně pro práci s ní, a proto tak ukazuje vysokou efektivitu pro konkrétní scénáře, které jsou v Kafce široce využívány.

Na druhou stranu, Apache Flink se vyznačuje mimořádně nízkou latencí, což potvrzuje tvrzení dostupných zdrojů [60][61], že toto nástroj opravdu nabízí. Flink byl navržen pro aplikace s nízkou latencí a vysokou propustností, což jej činí ideální volbou pro scénáře vyžadující rychlé zpracování dat v reálném čase. Tato charakteristika je důležitá pro aplikace, kde je zásadní minimalizovat zpoždění mezi přijetím dat a jejich zpracováním.

Apache Spark, ačkoli je efektivní v některých aspektech, jako je propustnost v rámci agregace, občas trpí vyšší latencí a využitím CPU, což naznačuje, že může být méně vhodný pro operace vyžadující vysokou reaktivitu nebo efektivní zpracování v reálném čase. Spark je tradičně silný v dávkovém zpracování, kde může efektivně zpracovávat velké objemy dat, ale v testech s nízkou latencí a vysokou propustností na datech v reálném čase nemusí vždy excelovat.

Pokud by byly brány v potaz pouze výsledky provedených testů, jako nejvhodnějším nástrojem by se jevil Kafka Streams, který má celkově nejčastěji nejlepší naměřené hodnoty. V tomto kontextu je ale třeba zdůraznit, že srovnání probíhalo v omezených podmínkách a poskytuje pouze zjednodušený náhled na vlastnosti a výkonnostní charakteristiky těchto nástrojů na základních příkladech. Je proto vhodné vybírat technologii podle konkrétního případu použití a nároků na platformu. Kafka Streams sice poskytuje jednoduchý a intuitivní přístup, nicméně je omezen na ekosystém Kafka platformy, zatímco nástroje Apache Flink a Apache Spark jsou rozsáhlejšími a poskytují širší škálu funkcí a možností integrací s okolím, avšak jejich použití vyžaduje složitější konfiguraci a správu.

7 Závěr

Cílem této diplomové práce bylo poskytnout komplexní pohled na proudy událostí, představit technologie a platformy sloužící k uchování a zpracování proudů událostí, identifikovat potenciální problémy a popsat řešení pro jejich překonání. Zvláštní pozornost byla věnována platformám Apache Kafka, Azure Event Hubs, Google Cloud Pub/Sub a Amazon Kinesis Data Streams pro uchování událostí, a technologiím Kafka Streams, Apache Flink a Apache Spark pro jejich zpracování.

V teoretické části byla představena základní terminologie a koncepty spojené s proudy událostí, včetně jejich definic a charakteristických vlastností. Bylo provedeno důkladné představení a srovnání vybraných platforem pro uchování a zpracování událostí, spolu s popisem rozdílných přístupů, které tyto platformy nabízejí. Speciální pozornost byla věnována potenciálním problémům a chybovým stavům, které mohou v praxi nastat, a byly navrženy metody pro jejich řešení.

V praktické části práce byla provedena podrobná analýza funkcí, které jednotlivé nástroje na zpracování událostí poskytují. Na základě této analýzy byly vytvořeny a implementovány testovací scénáře, které demonstrovaly základní funkcionality, jako jsou obohacení, filtrace, agregace a transformace dat v nástrojích Kafka Streams, Apache Flink a Apache Spark. Nástroje pak byly v rámci jednotlivých scénářů pomocí klíčových výkonnostních metrik porovnány. Testování zahrnovalo měření latence, propustnosti, využití hardwarových zdrojů, ale také meření náročnosti na vývoj. Kromě samotných testů práce přináší i pohled na to, jakým způsobem takové metriky sbírat a vyhodnocovat.

Výsledky tohoto základního srovnání poskytují cenné poznatky o silných a slabých stránkách zkoumaných nástrojů a platforem. Je důležité podotknout, že v praxi by testování mělo být rozšířeno o další aspekty, jako je odolnost vůči chybám, škálovatelnost, možnosti zabezpečení, vysoká dostupnost a náklady na provoz, aby bylo možné získat úplnější obraz o vhodnosti těchto technologií pro konkrétní aplikace.

Tato práce tedy slouží spíše jako základní stavební kámen pro ty, kteří stojí na počátku vývoje aplikací založených na proudovém zpracování událostí. Poskytuje

základní porovnání a přehled, který může sloužit jako východisko pro hlubší analýzu a rozhodovací proces při výběru nejvhodnějších technologií a platforem pro specifické podnikové potřeby.

Závěrem lze říci, že proudy událostí a technologie pro jejich zpracování představují klíčové prvky moderních IT architektur, umožňující efektivní a flexibilní zpracování velkých objemů dat v reálném čase. Správný výběr nástrojů a hluboké porozumění zmíněným principům jsou zásadní pro úspěšnou implementaci řešení, která jsou schopná rychle reagovat na dynamické podnikové a technologické výzvy.

8 Seznam použité literatury

[1] HELLER, Martin, 2022. What is streaming data? Event stream processing explained. online. In: InfoWorld. Dostupné z: https://www.infoworld.com/article/3646589/what-is-streaming-data-event-stream-processing-explained.html. [citováno 2023-09-21].

[2] Anon., 2020. What is Event Streaming? Webové sídlo. Dostupné z: https://tanzu.vmware.com/event-streaming. [citováno 2023-09-21].

[3] Anon., [s. a.]. Event Stream. online. In: Confluent. Dostupné z: https://developer.confluent.io/patterns/event-stream/event-stream/. [citováno 2023-09-21].

[4] DANUSHKA, Dunith, 2021. Making Sense of Unbounded Data. In: Tributary Data. 2021-02-01. Dostupné z: Tributary Data, https://medium.com/event-driven-utopia/making-sense-of-unbounded-data-4abdfa0edad2. [citováno 2023-09-21].

[5] Anon., [s. a.]. Fraud Prevention Reinvented with Data Streaming & Machine Learning. online. In: Confluent. Dostupné z: https://www.confluent.io/blog/fraud-prevention/. [citováno 2023-09-21].

[6] Anon., [s. a.]. What is Event Streaming? A Deep Dive. Webové sídlo. Dostupné z: https://ably.com/topic/event-streaming. [citováno 2023-09-21].

[7] POVZNER, Anna; Prince MAHAJAN; Jason GUSTAFSON; Jun RAO; Ismael JUMA et al., 2023. Kora: A Cloud-Native Event Streaming Platform for Kafka. online. Proceedings of the VLDB Endowment, roč. 16, č. 12, s. 3822–3834. Dostupné z: https://doi.org/10.14778/3611540.3611567.

[8] NARKHEDE, Neha; Gwen SHAPIRA a Todd PALINO, 2017. Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale. 1st vyd. O'Reilly Media, Inc. ISBN 978-1-4919-3616-0.

[9] Anon., [s. a.]. Apache Kafka. online. In: Apache Kafka. Dostupné z: https://kafka.apache.org/documentation/#uses. [citováno 2023-09-21].

[10] Anon., 2022. Kafka Brokers | Learn Apache Kafka with Conduktor. online. In: Kafka Brokers | Learn Apache Kafka with Conduktor. Dostupné z: https://www.conduktor.io/kafkakafka-brokers. [citováno 2023-09-21].

[11] Anon., [s. a.]. Publisher Subscriber (Pub-Sub) Design Pattern. Webové sídlo. Dostupné z: https://www.enjoyalgorithms.com/blog/publisher-subscriber-pattern. [citováno 2023-09-27].

[12] Anon., [s. a.]. What Is Pub/Sub? Publish/Subscribe Messaging Explained – BMC Software | Blogs. Webové sídlo. Dostupné z: https://www.bmc.com/blogs/pub-sub-publish-subscribe/. [citováno 2023-09-27].

[13] Anon., [s. a.]. Kafka vs. Traditional Messaging Queues: A Comprehensive Comparison | by Remis Haroon | Medium. Webové sídlo. Dostupné z: https://medium.com/@remisharoon/kafka-vs-traditional-messaging-queues-a-comprehensive-comparison-1687645f93a6. [citováno 2024-01-20].

[14] DANUSHKA, Dunith, 2021. Anatomy of an Event Streaming Platform — Part 1. In: Tributary Data. 2021-04-08. Dostupné z: Tributary Data, https://medium.com/event-driven-utopia/anatomy-of-an-event-streaming-platform-part-1-dc58eb9b2412. [citováno 2024-01-20].

[15] Anon., 2022. Standardised Communications Protocols | ARDC. online. In: https://ardc.edu.au/. Dostupné z: https://ardc.edu.au/resource/standardised-communications-protocols/. [citováno 2024-01-20].

[16] Anon., [s. a.]. Event Design and Event Streams Best Practices. Webové sídlo. Dostupné z: https://developer.confluent.io/courses/event-design/best-practices/. [citováno 2024-01-20].

[17] Anon., [s. a.]. Event Serializer. online. In: Confluent. Dostupné z: https://developer.confluent.io/patterns/event/event-serializer/. [citováno 2024-01-20].

[18] Anon., [s. a.]. JSON. Webové sídlo. Dostupné z: https://www.json.org/json-en.html. [citováno 2024-01-20].

[19] Anon., [s. a.]. Apache AvroTM 1.11.1 Documentation. online. In: Apache Avro. Dostupné z: https://avro.apache.org/docs/1.11.1/. [citováno 2024-01-20].

[20] Anon., [s. a.]. Protocol Buffers. Webové sídlo. Dostupné z: https://protobuf.dev/. [citováno 2024-01-20].

[21] Anon., [s. a.]. Kafka Topics Explained. In: Coding Harbour. Dostupné z: Coding Harbour, https://codingharbour.com/apache-kafka/the-introduction-to-kafka-topics-and-partitions/. [citováno 2024-01-20].

[22] Anon., [s. a.]. Operating system requirements | CDP Private Cloud. Webové sídlo. Dostupné z: https://docs.cloudera.com/cdp-private-cloud-base/7.1.8/kafka-configuring/topics/kafka-config-os-requirements.html. [citováno 2024-01-21].

[23] Anon., [s. a.]. What is a cluster? – Definition from TechTarget. Webové sídlo. Dostupné z: https://www.techtarget.com/whatis/definition/cluster. [citováno 2024-01-21].

[24] KUMAR, Narayan, 2020. Kafka Architecture & Internal. In: Medium. 2020-10-20. Dostupné z: Medium, https://mail-narayank.medium.com/kafka-architecture-internal-d0b3334d1df. [citováno 2023-09-27].

[25] FRANKLIN, Jerry, 2022. Apache Kafka Architecture: What You Need to Know. online. In: Upsolver. Dostupné z: https://www.upsolver.com/blog/apache-kafka-architecture-what-you-need-to-know. [citováno 2023-09-27].

[26] WAEHNER, Kai, 2021. Comparison of Open Source Apache Kafka vs Vendors including Confluent, Cloudera, Red Hat, Amazon MSK. In: Kai Waehner. 2021-04-20. Dostupné z: Kai Waehner, https://www.kai-waehner.de/blog/2021/04/20/comparison-open-source-apache-kafka-vs-confluent-cloudera-red-hat-amazon-msk-cloud/. [citováno 2023-10-03].

[27] Anon., [s. a.]. Ecosystem - Apache Kafka - Apache Software Foundation. Webové sídlo. Dostupné z: https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem. [citováno 2023-10-03].

[28] Anon., [s. a.]. Apache Kafka. online. In: Apache Kafka. Dostupné z: https://kafka.apache.org/documentation/streams/. [citováno 2023-10-03].

[29] Anon., [s. a.]. Confluent Platform Overview | Confluent Documentation. Webové sídlo. Dostupné z: https://docs.confluent.io/platform/current/platform.html. [citováno 2023-10-03].

[30] SPELLURU, 2023. Azure Event Hubs – data streaming platform with Kafka support - Azure Event Hubs. Webové sídlo. Dostupné z: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about. [citováno 2023-10-03].

[31] ALAM, Izhar, 2023. Azure Event Hub | Event Hub Architecture | Working. online. In: Cloud Training Program. Dostupné z: https://k21academy.com/microsoft-azure/data-engineer/azure-event-hubs/. [citováno 2023-10-11].

[32] Anon., [s. a.]. What is PaaS? Platform as a Service | Microsoft Azure. Webové sídlo. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas. [citováno 2023-10-11].

[33] Anon., [s. a.]. What is Pub/Sub? | Cloud Pub/Sub Documentation | Google Cloud. Webové sídlo. Dostupné z: https://cloud.google.com/pubsub/docs/overview. [citováno 2023-10-11].

[34] Anon., [s. a.]. ActiveMQ. Webové sídlo. Dostupné z: https://activemq.apache.org/. [citováno 2023-10-11].

[35] Anon., [s. a.]. RabbitMQ: One broker to queue them all | RabbitMQ. Webové sídlo. Dostupné z: https://www.rabbitmq.com/. [citováno 2023-10-11].

[36] MAITI, Kamal, 2023. Deep Dive into GCP Pub/Sub: Unraveling the Architecture and Benefits. In: Medium. 2023-05-15. Dostupné z: Medium, https://medium.com/@kamal.maiti/gcp-pub-sub-architecture-understanding-199e1f7913b7. [citováno 2023-10-11].

[37] Anon., [s. a.]. Pub/Sub for Application & Data Integration. online. In: Google Cloud. Dostupné z: https://cloud.google.com/pubsub. [citováno 2023-10-11].

[38] Anon., [s. a.]. Cloud Functions | Google Cloud. Webové sídlo. Dostupné z: https://cloud.google.com/functions?hl=cs. [citováno 2024-01-21].

[39] Anon., [s. a.]. Cloud Storage | Google Cloud. Webové sídlo. Dostupné z: https://cloud.google.com/storage?hl=en. [citováno 2024-01-21].

[40] Anon., [s. a.]. Push Notifications | Gmail. online. In: Google for Developers. Dostupné z: https://developers.google.com/gmail/api/guides/push. [citováno 2024-01-21].

[41] MAKOTA, Tarik; Brian MAGUIRE; Danny GAGNE a Rajeev CHAKRABARTI, 2021. Scalable Data Streaming with Amazon Kinesis: Design and secure highly available, cost-effective data streaming applications with Amazon Kinesis. 1st edition. Packt Publishing. ISBN 978-1-80056-540-1.

[42] Amazon Kinesis Data Streams Terminology and Concepts. AWS [online]. 2023 [cit. 2023-10-11]. Dostupné z: https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html

[43] Anon., [s. a.]. Data Streams Integration. Webové sídlo. Dostupné z: https://aws.amazon.com/kinesis/data-streams/integrations/. [citováno 2023-12-06].

[44] Anon., [s. a.]. Developer Tools - SDKs and Programming Toolkits for Building on AWS. Webové sídlo. Dostupné z: https://aws.amazon.com/developer/tools/. [citováno 2023-12-06].

[45] Anon., [s. a.]. Managed Streaming Data Service | Amazon Kinesis Data Streams Pricing | Amazon Web Services. Webové sídlo. Dostupné z: https://aws.amazon.com/kinesis/data-streams/pricing/. [citováno 2023-12-06].

[46] Anon., [s. a.]. The Complete Guide to Event-Driven Architecture | Solace. Webové sídlo. Dostupné z: https://solace.com/what-is-event-driven-architecture/. [citováno 2023-12-06].

[47] Anon., [s. a.]. Microservices Pattern: Pattern: Event sourcing. online. In: microservices.io. Dostupné z: http://microservices.io/patterns/data/event-sourcing.html. [citováno 2023-12-06].

[48] MARTINEKUAN, [s. a.]. Event Sourcing pattern - Azure Architecture Center. Webové sídlo. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing. [citováno 2023-12-06].

[49] MARTINEKUAN, [s. a.]. CQRS pattern - Azure Architecture Center. Webové sídlo. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs. [citováno 2023-12-06].

[50] Anon., [s. a.]. CQRS pattern - AWS Prescriptive Guidance. Webové sídlo. Dostupné z: https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html. [citováno 2023-12-06].

[51] OZKAYA, Mehmet, 2023. CQRS Design Pattern in Microservices Architectures. In: Design Microservices Architecture with Patterns & Principles. 2023-04-09. Dostupné z: Design Microservices Architecture with Patterns & Principles, https://medium.com/design-microservices-architecture-with-patterns/cqrs-design-pattern-in-microservices-architectures-5d41e359768c. [citováno 2023-12-06].

[52] MARTINEKUAN, [s. a.]. Saga pattern - Azure Design Patterns. Webové sídlo. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga. [citováno 2023-12-06].

[53] Anon., [s. a.]. Pattern: Saga. Webové sídlo. Dostupné z: https://microservices.io/patterns/data/saga.html. [citováno 2023-12-06].

[54] INC, Gartner, [s. a.]. Best Event Stream Processing Reviews 2023 | Gartner Peer Insights. online. In: Gartner. Dostupné z: https://www.gartner.com/market/event-stream-processing. [citováno 2023-12-06].

[55] SEYMOUR, Mitch, 2021. Mastering Kafka Streams and ksqlDB: Building Real-Time Data Systems by Example. 1st edition. Beijing China ; Boston [MA]: O'Reilly Media. ISBN 978-1-4920-6249-3.

[56] Anon., [s. a.]. Apache Kafka. online. In: Apache Kafka. Dostupné z: https://kafka.apache.org/10/documentation/streams/developer-guide/dsl-api. [citováno 2023-12-06].

[57] Anon., [s. a.]. Kafka Streams Processor API for Confluent Platform | Confluent Documentation. Webové sídlo. Dostupné z: https://docs.confluent.io/platform/current/streams/developer-guide/processor-api.html. [citováno 2023-12-06].

[58] Anon., [s. a.]. What is Kafka Streams: A Comprehensive Guide 101 | Hevo. Dostupné z: https://hevodata.com/learn/kafka-streams/. [citováno 2023-12-06].

[59] Anon., [s. a.]. Kafka Stream architecture - Building Data Streaming Applications with Apache Kafka [Book]. Webové sídlo. ISBN 9781787283985. Dostupné z: https://www.oreilly.com/library/view/building-data-streaming/9781787283985/a3fef279-d67a-4db6-b74e-e9bc34383612.xhtml. [citováno 2023-12-06].

[60] Anon., [s. a.]. What is Apache Flink? - Apache Flink Explained - AWS. online. In: Amazon Web Services, Inc. Dostupné z: https://aws.amazon.com/what-is/apache-flink/. [citováno 2023-12-06].

[61] Anon., [s. a.]. Use Cases. Webové sídlo. Dostupné z: https://flink.apache.org/what-is-flink/use-cases/. [citováno 2023-12-06].

[62] Anon., [s. a.]. Apache Flink: An Introduction. online. In: Confluent. Dostupné z: https://www.confluent.io/learn/apache-flink/. [2023-12-06].

[63] Anon., [s. a.]. Apache Flink - Architecture. Webové sídlo. Dostupné z: https://www.tutorialspoint.com/apache_flink/apache_flink_architecture.htm. [citováno 2023-12-06].

[64] Anon., [s. a.]. Overview. Webové sídlo. Dostupné z: https://nightlies.apache.org/flink/flink-docs-release-1.18/docs/concepts/overview/. [citováno 2023-12-07].

[65] Anon., [s. a.]. Fraud Detection with the DataStream API. Webové sídlo. Dostupné z: https://nightlies.apache.org/flink/flink-docs-release-1.18/docs/try-flink/datastream/. [citováno 2023-12-07].

[66] Anon., [s. a.]. What is Spark? - Introduction to Apache Spark and Analytics - AWS. online. In: Amazon Web Services, Inc. Dostupné z: https://aws.amazon.com/what-is/apache-spark/. [citováno 2023-12-07].

[67] Anon., [s. a.]. Spark Streaming - Spark 3.5.0 Documentation. Webové sídlo. Dostupné z: https://spark.apache.org/docs/latest/streaming-programming-guide.html. [citováno 2023-12-07].

[68] SAYS, S. P. Mohanty, 2018. Apache Spark Architecture | Distributed System Architecture Explained. In: Edureka. 2018-09-28. Dostupné z: Edureka, https://www.edureka.co/blog/spark-architecture/. [citováno 2023-12-07].

[69] Anon., [s. a.]. A Beginners Guide to Spark Streaming Architecture with Example. Webové sídlo. Dostupné z: https://www.projectpro.io/article/spark-streaming-example/540. [citováno 2023-12-08].

[70] CHANDRAKANT, Kumar, 2019. Building a Data Pipeline with Kafka, Spark Streaming and Cassandra | Baeldung. Webové sídlo. Dostupné z: https://www.baeldung.com/kafka-spark-data-pipeline. [citováno 2023-12-08].

[71] NAVEEN (NNK), 2022. Spark SQL Explained with Examples. online. In: Spark By {Examples}. Dostupné z: https://sparkbyexamples.com/spark/spark-sql-explained/. [citováno 2023-12-08].

[72] Anon., [s. a.]. What are the Advantages & Disadvantages of Apache Spark? Webové sídlo. Dostupné z: https://www.knowledgehut.com/blog/big-data/apache-spark-advantages-disadvantages. [citováno 2023-12-08].

[73] s GOLDER, Rob, 2023. Kafka Deduplication Patterns (1 of 2). In: Lydtech Consulting. 2023-10-22. Dostupné z: Lydtech Consulting, https://medium.com/lydtech-consulting/kafka-deduplication-patterns-1-of-2-ef0371a3331b. [citováno 2023-12-08].

[74] Anon., [s. a.]. Error Handling Patterns in Kafka. online. In: Confluent. Dostupné z: https://www.confluent.io/blog/error-handling-patterns-in-kafka/. [citováno 2023-12-08].

[75] HUBBARD, Douglas W., 2014. How to Measure Anything: Finding the Value of Intangibles in Business. 3rd edition. Hoboken, New Jersey: Wiley. ISBN 978-1-118-53927-9.

[76] BHAT, Adi, 2018. Rating Scale: Definition, Survey Question Types & Examples. In: QuestionPro. 2018-08-14. Dostupné z: QuestionPro, https://www.questionpro.com/blog/rating-scale/. [citováno 2024-02-26].

[77] Anon., [s. a.]. Datagen Source Connector for Confluent Platform | Confluent Documentation. Webové sídlo. Dostupné z: https://docs.confluent.io/kafka-connectors/datagen/current/overview.html. [citováno 2024-02-26].

[78] Anon., [s. a.]. Kafka Command Line Interface (CLI) Tools | Confluent Documentation. Webové sídlo. Dostupné

z: https://docs.confluent.io/kafka/operations-tools/kafka-tools.html. [citováno 2024-02-26].

[79] Anon., [s. a.]. Prometheus - Monitoring system & time series database. Webové sídlo. Dostupné z: https://prometheus.io/. [citováno 2024-02-26].

[80] PROMETHEUS, [s. a.]. Grafana | Prometheus. Webové sídlo. Dostupné z: https://prometheus.io/docs/visualization/grafana/. [citováno 2024-02-26].

[81] PROMETHEUS, [s. a.]. Monitoring Linux host metrics with the Node Exporter | Prometheus. Webové sídlo. Dostupné z: https://prometheus.io/docs/guides/node-exporter/. [citováno 2024-02-26].

[82] BOSCH, Richard, 2023. What is an Enterprise Event Streaming Platform? In: Axual. 2023-04-05. Dostupné z: Axual, https://axual.com/what-is-an-enterprise-event-streaming-platform/. [citováno 2024-03-03].

[83] WILKES, Steve, 2017. Making the Most of Apache Kafka – Data Processing and Preparation for Kafka. In: Striim. 2017-08-31. Dostupné z: Striim, https://www.striim.com/blog/making-apache-kafka-processing-preparation-kafka/. [citováno 2024-03-03].

[84] Anon., [s. a.]. Oracle VM VirtualBox. Webové sídlo. Dostupné z: https://www.virtualbox.org/. [citováno 2024-03-07].

[85] Anon., [s. a.]. Vagrant by HashiCorp. online. In: Vagrant by HashiCorp. Dostupné z: https://www.vagrantup.com/. [citováno 2024-03-07].

[86] Anon., [s. a.]. AlmaLinux OS - Forever-Free Enterprise-Grade Operating System. online. In: AlmaLinux OS. Dostupné z: https://almalinux.org/. [citováno 2024-03-07].

[87] Anon., [s. a.]. Ansible is Simple IT Automation. Webové sídlo. Dostupné z: https://www.ansible.com/. [citováno 2024-03-07].

9 Přílohy

9.1 Příloha 1 – Zdrojové kódy testů

https://github.com/jirkapipek/flink-spark-kstreams-comparsion-dp/tree/main/test-cases

9.2 Příloha 2 – Komplexní výsledky testů

https://github.com/jirkapipek/flink-spark-kstreams-comparsion-dp/blob/main/vysledky_testu.xlsx

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. Jiří Pipek**

Studium: I2200404

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Proudy událostí a nástroje na jejich zpracování**

Název diplomové práce AJ: Event streams and event stream processing software

Cíl, metody, literatura, předpoklady:

Cílem diplomové práce je popsat některé z existujících nástrojů pro zpracování proudů událostí. Pro vybrané nástroje navrhnout a provést testy k jejich vzájemnému porovnání z hlediska výkonnosti, nároků na HW a náročnosti na vývoj.

Osnova:

Úvod
Teoretická část
Analýza a návrh
Praktická část
Závěr
Literatura

SEYMOUR, Mitch. Mastering Kafka Streams and ksqlDB. ISBN 9781492062493.

GERARD, Maas. Stream Processing with Apache Spark: Mastering Structured Streaming and Spark Streaming. ISBN 9781491944240.

FABIAN, Hueske. Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications. ISBN 9781491974292.

KLEPPMANN, Martin. Making Sense of Stream Processing: The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms. 978-1-491-94010-5.

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Ing. Karel Malý, Ph.D.

Datum zadání závěrečné práce: 26.1.2021