Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Datová integrace nezpracovaných dat z IoT

Ondřej Bednář

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Ondřej Bednář

Informatika

Název práce

Datová integrace nezpracovaných dat z IoT

Název anglicky

Data Integration of Raw IoT Data

Cíle práce

Bakalářské práce je tematicky zaměřena na integraci dat pocházejících ze zařízení IoT. Hlavním cílem je výběr nejvhodnějšího ETL nástroje pro uložení dat ze senzorů používaných v zemědělství do datového skladu s možností budoucího využití např. pro machine learning.
Dílčími cíli jsou:

- charakteristika dostupných řešení,
- vytvoření AHP modelu,
- výběr nejvhodnějších řešení pro následnou implementaci
- implementace těchto řešení a porovnání jejich výkonu.

Metodika

Na základě přehledu odborných informačních zdrojů budou charakterizovány používané ETL nástroje a možnosti přenosu a ukládání dat. Za pomoci vhodně sestaveného rozhodovací modelu AHP (analytical hierarchy process) budou vybrána dvě nejlepší řešení pro následné použití. V praktické části budou vybraná řešení z předchozí části implementována ve vybraném cloudovém prostředí. Implementované nástroje budou následně otestovány za pomoci vhodně navrhnutého výkonnostního testu. Na základě teoretických poznatků a výsledků praktické části budou zpracovány závěry práce.

Doporučený rozsah práce

40 – 50 stran

Klíčová slova

big data, datová integrace, datové jezero, datový sklad, ETL, IoT, operační datové úložiště

Doporučené zdroje informací

GORELIK, Alex. The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science. Sebastopol: O'Reilly Media, 2019. ISBN 9781491931554.

KIMBALL, Ralph a Margy ROSS. The data warehouse toolkit: the definitive guide to dimensional modeling. 3rd ed. Indianapolis: Wiley, c2013. ISBN 9781118530801.

PERROS, Harry. An Introduction to IoT Analytics. London: Taylor & Francis, 2021. ISBN 9780367686314.

SHERMAN, Rick. Business Intelligence Guidebook: From Data Integration to Analytics. Amsterdam: Morgan Kaufmann, 2014. ISBN 9780124114616.

STACKOWIAK, Robert, Art LICHT, Venu MANTHA a Louis NAGODE. Big Data and The Internet of Things: Enterprise Information Architecture for A New Age. Berkeley: Apress, c2015. ISBN 9781484209868.

Předběžný termín obhajoby

2022/23 ZS – PEF

Vedoucí práce

Ing. Jan Masner, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 9. 8. 2021

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5. 10. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 14. 03. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Datová integrace nezpracovaných dat z IoT“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2023 _____

Poděkování

Rád bych poděkoval panu Ing. Janu Masnerovi, Ph.D., za jeho cenné rady, vedení a podporu během psaní této bakalářské práce. Jeho odborné znalosti, nápady a inspirace byly velkým přínosem pro můj akademický rozvoj a práci na této práci. Taktéž bych chtěl poděkovat své mamince za trpělivost, podporu a lásku, kterou mi projevila v průběhu celého studia.

Datová integrace nezpracovaných dat z IoT

Abstrakt

Pokrok ve vývoji hardwaru, softwaru a komunikačních technologií umožnil vznik zařízení obsahujících řadu senzorů připojitelných k internetové síti, čímž se otevřely možnosti k měření a následnému využití různorodých dat poskytujících digitální obraz reálného světa. S růstem počtu a vyspělostí těchto zařízení vyvstává stále větší potřeba získaná objemná data zpracovávat a uchovávat. Technologie zařízení připojených k internetové síti, označovaná jako internet věcí, rozšiřuje současný internet tím, že poskytuje propojení a interakci mezi fyzickým a informačním světem. Kromě vysokého objemu dat internet věcí produkuje data, vyznačující se vysokou kardinalitou ve smyslu unikátnosti hodnot, rychlostí těchto dat z hlediska závislosti počtu příchozích dat na čase, a jejich různou kvalitou. Rychlé a kvalitní zpracování těchto objemných dat umožní další analýzu a využití technik strojového učení.

Tato práce se zaobírá analýzou moderních nástrojů pro datovou integraci dat pocházejících z internetu věcí. V praktické části práce pak implementací těchto nástrojů spolu s hodnocením výkonnostních testů.

Klíčová slova: big data, datová integrace, datové jezero, datový sklad, ETL, internet věcí, operační datové úložiště

Data Integration of Raw IoT Data

Abstract

Rapid advances in hardware, software and communication technologies have enabled the emergence of devices with a multitude of sensors connected to the Internet, providing observations and measurements of diverse real-world data. As the number and maturity of these technologies grows, so will the volume of data processed. Internet-connected device technology, referred to as the Internet of Things (IoT), continues to augment the current Internet by providing connections and interactions between the physical and information worlds. In addition to the high volume of data, the IoT produces data characterised by high cardinality in terms of uniqueness of values, the velocity of this data in terms of number versus time, and its varying quality. Fast and high-quality processing of this data will enable further analysis and possible usage of machine learning techniques. This paper deals with the analysis of modern tools for data integration of data originating from IoT. The practical part of the thesis then includes the implementation of these tools along with the evaluation of performance tests.

Keywords: big data, data integration, data lake, data warehouse, ETL, Internet of Things, operational data store

Obsah

1 Úvod

V tradičních datových prostředích nástroj ETL (Extract, Transform, Load) extrahoval dávky dat ze zdrojového systému obvykle na základě časového rozvrhu. Získaná data transformoval a poté je načetl do úložiště jako datový sklad. Tento model „dávkového ETL“ se využívá již od začátku 90. let minulého století. Většina moderních odvětví však nemůže čekat hodiny nebo dny, než proběhne zpracování dávky dat a naplnění finálního úložiště. Musí reagovat na nová data v reálném čase, co nejblíže okamžiku, ve kterém byla generována.

Integrace dat v reálném čase zvyšuje prostor pro doručení přidané hodnoty a pomáhá získat lepší vhled do rozmanitých zdrojů, které data produkují. V současném prostředí lze komplexní data a informace získaná zařízeními IoT považovat za velký soubor dat shromážděný v reálném čase. Efektivní ukládání těchto velkých objemů dat pomáhá orientovat se v jejich složité struktuře, umožňuje data účinně analyzovat a následně zužitkovat.

Systémy IoT využívá většina průmyslových odvětví. V současné době je v těchto sítích připojeno více než 100 milionů zařízení. Příkladem oborů, ve kterých IoT nalézá uplatnění, jsou energetika, plynárenství, vzduchová technika, zásobování vodou, ale využití je možné i mimo průmyslový sektor. Dobrým příkladem je doprava a skladování, ale i maloobchod a velkoobchod. Společnost Gartner předpovídá, že počet zařízení IoT ve všech průmyslových odvětvích vzroste do roku 2030 na více než osm miliard.

Orientovat se v obrovském množství dat není snadné s ohledem na jejich rozmanitost, rozdílnou strukturu a odlišnost systémů ze kterých pochází. Cílem je dostat data na jedno místo, sjednotit je, vyčistit a dát jim správnou strukturu, tzn. zajistit jejich datovou integraci. Vhodným řešením pro datovou integraci je využití cloudu, který poskytuje efektivní možnosti škálování zpracování a následného uložení dat. Strojové učení pro prediktivní a klasifikační úlohy je nyní integrováno do většiny cloudových IoT platforem, jako je Amazon AWS IoT, Microsoft Azure IoT, Google Cloud IoT nebo IBM Watson IoT Cloud.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je výběr nejvhodnějšího ETL nástroje pro uložení dat ze senzorů používaných v zemědělství do datového skladu s mozností budoucího využití např. pro machine learning.

Dílčími cíli jsou:

- charakteristika dostupných řešení,
- vytvoření AHP modelu,
- výběr nejvhodnějších řešení pro následnou implementaci,
- implementace těchto řešení a porovnání jejich výkonu.

2.2 Metodika

Na základě přehledu odborných informačních zdrojů budou charakterizovány používané ETL nástroje a možnosti přenosu a ukládání dat. Za pomoci vhodně sestaveného rozhodovacího modelu AHP (analytical hierarchy process) budou vybrána dvě nejlepší řešení pro následné použití.

V praktické části budou řešení vybrána z předchozí části implementována ve vybraném cloudovém prostředí. Implementované nástroje budou následně otestovány za pomoci vhodně navrhnutého výkonnostního testu. Na základě teoretických poznatků a výsledků praktické části budou zpracovány závěry práce.

3 Teoretická východiska

3.1 Charakteristika IoT dat

Zařízení internetu věcí dále jen IoT jsou fyzické objekty, které jsou připojeny k internetu a mohou shromažďovat a přenášet data. Tato zařízení mohou zahrnovat jednoduché senzory, měřiče, RFID čipy, ale také složitější monitorovací a řídicí zařízení, či připojená celá vozidla a další stroje. Data pocházející z těchto zařízení jsou uživateli, technikou, stroji denně vytvářena při každodenních činnostech a provozu v řádech miliard bajtů (1, 2). Vlastnosti dat se mohou lišit v závislosti na konkrétní aplikaci a systému IoT (1).

3.1.1 Komplexní sémantika dat

Zařízení zapojená do IoT generují různá nezpracovaná senzorická data, která jsou umístěna na vysoce distribuovaných, heterogenních a zdrojově omezených zařízeních, jsou vzájemně propojena a autonomně komunikují v různých scénářích (1). S ohledem na uvedené jsou pro manipulaci s daty v takto komplexní sémantice zapotřebí spolehlivé integrační techniky.

Se sémantikou dat úzce souvisí pojem datová heterogenita. Data pocházející z IoT se vyznačují svojí různorodostí (datovou heterogenitou). Často pochází z odlišných oblastí reálného světa, vyskytují se například jako fyzikální data, biologická data anebo telemetrická data (1). S rostoucím rozsahem producentů mohou být data z různých zdrojů korelována.

Je evidentní, že různá data z různých zdrojů mohou být různého typu – numerická, textová nebo binární. Organizace dat může být tvořena kombinací strukturovaných dat např. relační záznamy, semistrukturovaných dat např. ve formátu JavaScript Object Notation – JSON a nestrukturovaných dat např. logy. Struktura a forma dat se navíc může v čase měnit. Nejednoznačnost sémantiky a dynamická změna struktury vedou k nejistotě dat a jejich nekvalitě (2).

3.1.2 Masivní měřítko

Obrovské množství inteligentních zařízení připojených k IoT může vytvářet biliony dat v reálném čase, International Data Corporation odhaduje, že v roce 2025, budou IoT zařízení generovat data o objemu 79,4 zettabyte ročně (3). Například jediné zařízení

zapojené do IoT, jako je chytrý měřič nebo bezpečnostní kamera, může denně generovat tisíce datových bodů. Pokud se tyto datové body agregují ve více zařízeních, může objem dat IoT rychle nabýt velmi velkých rozměrů. Taková data potřebují velký úložný prostor a výkonný systém pro jejich zpracování (4).

V těchto rozsáhlých souborech dat mohou být určité datové body nadbytečné (5, 6). Data mohou být duplikovaná například v situacích, kdy jsou senzory hustě rozmístěny ať už náhodně nebo záměrně a snímají stejné veličiny. Vzhledem k překryvu měřených oblastí dochází k duplikaci dat. Tato redundance snižuje výkonnost z důvodu nárůstu výpočetní režie, potřeby nadměrného přenosu a zvyšuje náklady kvůli obsazení zbytečně velkého prostoru v úložišti. Redukce redundantních záznamů může mít vliv na přesnost a spolehlivost dat.

3.1.3 Časová závislost

Stav fyzických objektů je často snímán z hlediska změn měřených hodnot v čase. Pochopení charakteristik v těchto časových řadách je důležité pro efektivní analýzu a interpretaci dat. Časové řady dat se často shromažďují v pravidelných intervalech, například každou minutu, hodinu nebo den. Techniky analýzy časových řad lze použít k identifikaci vzorců a trendů v datech, stejně jako k předpovědi budoucích hodnot a odhalení anomálií (7).

Historická data jsou sice důležitá pro analýzu a vytváření modelů, ale mnohdy je zásadní získat data popisující stav objektů v reálném čase. V takovém řešení jsou kladeny vysoké nároky na spolehlivost integračních řešení a dostupnost databází. Pro využití dat v „real-time“ analýze je žádoucí, aby byla data z IoT pro zpracování předávána rychle a pravidelně. V opačném případě mohou být závěry vyvozené za použití nekvalitních dat nepřesné (7).

3.2 Datová integrace

Zásadou pro automatizované zpracování dat je zajistit důvěryhodnost a kvalitu dat a jejich metadat, aby analytické nástroje a aplikace strojového učení mohly data přesně zpracovávat a interpretovat. Například prostorová data (údaje o trajektorii pohybujících se objektů) jsme schopni v kombinaci s časoprostorovými informacemi využít v systému monitorování silnic a vozidel pro přepravu speciálního (nebezpečného) materiálu (8). Systémy monitorování zdravotního stavu sledují stav pacienta pomocí snímání biologických

údajů, jako je tělesná teplota, krevní tlak a srdeční frekvence (9). Systémy z oblasti enviromentálního prostředí nebo agrikultury pracují s údaji o teplotě vzduchu, humiditě půdy, svítivosti a využívají je například v predikci přítomnosti různých látek (10). Zjištěné informace lze pak využít při zvyšování produkce, předcházení následkům přírodních katastrof nebo ekologickém monitoringu (11). Schopnost kombinovat různé objekty s rozdílnou strukturou modelu či schématu, je tak jedním ze zásadních požadavků na přesnou analýzu dat.

Integrace dat je důležitým aspektem pro získání poznatků a informací z dat IoT. Umožňuje kombinovat a analyzovat data z různých zdrojů konzistentním a smysluplným způsobem (12). V kontextu IoT integrace dat obvykle zahrnuje extrakci dat z různých zařízení a senzorů, jejich transformaci do konzistentního formátu a načtení do cílového systému, jako je datový sklad nebo datové jezero.

Proces integrace dat v systémech IoT se často provádí pomocí specializovaných nástrojů a technologií, jako jsou nástroje ETL, nástroje pro správu dat a nástroje pro správu datových modelů (13). Tyto nástroje a technologie pomáhají zajistit, aby data byla konzistentní, přesná a použitelná. Kvalitně integrovaná data mohou podporovat širokou škálu „data driven“ aplikací jako je monitorování stavu v reálném čase, prediktivní údržba a poskytovat kvalitní základ pro různé druhy analýz jako analýza chování zákazníků.

Datová integrace však nezahrnuje pouze migraci dat tzn. přesun dat z jednoho systému nebo prostředí do jiného. Vzhledem k rozsáhlé struktuře sítí IoT může být velice složité a matoucí v tomto prostředí pracovat, proto dalším požadavkem na datovou integraci je zavedení systému pro uspořádání těchto dat. Jedním z možných řešení může být zavedení ontologie (14), která pomáhá data uspořádat a klasifikovat podle svých vlastností a vztahů a poskytuje vhodný startovací bod pro spojení s konceptem digitálního dvojčete. Termín „digitální dvojče“ poprvé použil Michael Grieves na Michiganské univerzitě v roce 2002, když jej definoval jako *„digitální repliku fyzických aktiv, procesů, lidí, míst, systémů a zařízení, kterou lze využít k různým účelům“*.

Digitální dvojčata mohou být obzvláště užitečná pro integraci dat v kontextu IoT, protože poskytují flexibilní a škálovatelný způsob integrace dat z více zařízení a systémů. Vytvořením digitálního dvojčete systému IoT mohou uživatelé snadno a efektivně kombinovat data z různých zdrojů, včetně senzorů, akčních členů a dalších zařízení, získat tak úplnější a přesnější pohled na chování a výkonnost systému (15). Digitální dvojčata

navíc mohou poskytnout cenné informace o fungování fyzických zařízení a systémů IoT, což uživatelům umožní lépe porozumět jejich chování a identifikovat potenciální problémy.

3.2.1 Výpočetní prostředí

3.2.1.1 Cloud

Cloud je termín pro moderní způsob řešení informačních technologií, kde výpočetní zdroje či aplikace poskytuje dodavatel prostřednictvím internetové sítě. Cloudová řešení umožňují dle aktuální situace zákazníka dynamicky měnit přidělení těchto zdrojů a umožňují tak zákazníkovi pronájem pouze reálně využívaných prostředků.

Cloud se dá rozdělit na několik kategorií dle umístění výpočetního prostředí. Cloudové služby mohou být veřejné, kdy zákazník platí sdílené, zabezpečené prostředí od dodavatelů, jako například AWS (Amazon), Azure (Microsoft), Google Cloud Platform (Google). Dále soukromé, které není sdíleno s veřejností a poskytuje služby interním uživatelům/zákazníkům. Soukromé prostředí je vybudováno a provozováno na „on-premise“ infrastruktuře často s využitím virtualizačního softwaru jako například OpenStack apod. V praxi se často vyskytuje kombinace těchto přístupů spojená v hybridním režimu (16). Tento přístup využívají například výrobní společnosti, které mají jedinečné nastavení síťové infrastruktury nebo instituce, které musí dodržovat právní předpisy a složité regulatorní politiky týkající se ochrany dat. Tyto organizace pak mají k dispozici řešení uskutečněné kombinací infrastruktury „on-premise“ (např. datové centrum v bance) a veřejné cloudové infrastruktury. Služby v těchto infrastrukturách jsou pak provozovány dle individuální potřeby, což umožňuje splnit specifické požadavky v oblasti bezpečnosti, spolehlivosti a ochrany dat.

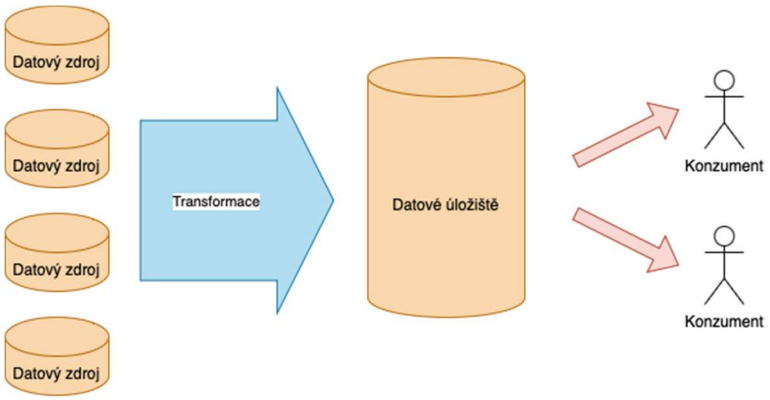
3.2.1.2 Fog

Fog computing rozšiřuje model cloud computingu tím, že výpočty a ukládání dat přenáší z centralizovaného prostředí blíže k síti, kde data vznikají (17). V IoT lze fog computing využít k přenesení části výpočtů a ukládání dat z cloudu do logických „okrajů“ připojených sítí – Edge serverů, což zařízením a senzorům umožní provádět část zpracování a analýzy dat lokálně. To může být užitečné zejména pro aplikace, které vyžadují zpracování v reálném čase nebo mají přísné požadavky na latenci, jako je průmyslová automatizace, chytrá města nebo autonomní vozidla. Výhodou lokálního zpracování je snížení množství dat, která je třeba přenášet do cloudu. Tento výpočetní model může také zvýšit spolehlivost

a bezpečnost systému IoT tím, že poskytuje vrstvu redundance a distribuuje výpočty a ukládání dat mezi více míst a zařízení (18). To může pomoci eliminovat jediný bod selhání systému a lépe jej chránit před kybernetickými hrozbami.

3.2.2 Techniky datové integrace

Obrázek 1 Tradiční architektura ETL



Zdroj: vlastní práce

Tradiční ETL[1] (obr. 1) je soubor operací, které probíhají v rámci architektury datového skladu, obecně je znám jako proces extrakce, transformace a načtení (ETL). Procesy ETL jsou zodpovědné za extrakci dat z různě distribuovaných a často heterogenních datových zdrojů (12). Před jejich přesunem do datového skladu je zapotřebí provést jejich čištění a přizpůsobení tak, aby data odpovídala daným potřebám a pravidlům a zajistit jejich transformaci do odpovídajícího schématu používaném v daném datového skladu (12).

Zrod ETL přišel s nástroji jako Oracle Warehouse Builder, IBM DataStage, Cognos DecisionStream (později IBM Data Manager) a Informatica. Společným faktorem těchto nástrojů je, že transformace se provádí na aplikačních serverech, nikoli ve zdrojovém systému nebo cílové databázi, což vyžaduje, aby data byla extrahována ze zdroje, přenesena a upravena aplikační infrastrukturou (serverem/serverem ETL) a vrácena zpět do datového skladu.

[1] [1] *„**ETL** je zkratka pro **Extract-Transform-Load**, tři databázové funkce, které jsou sloučeny do jednoho nástroje, který vytáhne údaje z jedné databáze a umístí je do jiné databáze. **Extrakt** je proces čtení dat z databáze. **Transform** je proces konverze získaných dat z jejich původní podoby do podoby, ve které musí být, aby se mohly umístit do jiné databáze. **Load** je proces zápisu dat do cílové databáze. ETL se používá k migraci dat z jedné databáze do druhé pro vytvoření datových tržišť a datových skladů a také pro převod databáze z jednoho formátu nebo typu do druhého.“*
Zdroj: https://it-slovnik.cz/pojem/etl/?utm_source=cp&utm_medium=link&utm_campaign=cp

Se stále zvyšujícím se objemem dat a jejich méně strukturovanou podobou se začala uplatňovat obdoba procesu datové integrace zvaná ELT (19), kde je fáze transformace a nahrávání prohozena. Data tak zůstávají na stejné platformě, čímž je možné se vyhnout přesunu velkých objemům dat a tím dlouhé době jejich zpracování. Výhodou nástrojů ELT je, že transformace se provádí v rámci databáze s využitím výkonu podkladového datového skladu a zároveň stále poskytuje funkce nástrojů ETL.

Vzhledem ke stále rostoucímu objemu, rozmanitosti a rychlosti dat bylo rozumné přesunout výpočetní procesy blíže k datům. Platforma Hadoop nabízí rozšiřitelný výpočetní výkon s možností nasazení na běžně dostupném hardwaru a schopnost řešit datově náročné úlohy (20). Vývoj aplikací na platformě Hadoop spočívá převážně v ručním kódování programů za pomoci modelu MapReduce v jazycích Scala, Java nebo Python, což zvyšuje nároky na odbornost datových inženýrů. Některé existující ETL nástroje například Pentaho, vytvořily konektory podporující interakci s touto platformou, čímž se práce s ní zjednodušila.

Převážně však systémy tradičních datových skladů (Data Warehouse) a velkých dat (Big Data) pracují v oddělených prostorech. Jednotná integrace dat napříč Data Warehouse a Big Data je velice obtížná a úspěšná implementace tohoto spojeného systému je výzvou pro většinu organizací. V důsledku toho spojení, mohou také vyvstávat nevýhody. Takové organizace mají vytvořena oddělení s duplicitními schopnostmi, což jim způsobuje aditivní náklady mimo jiné na vyšší úroveň programátorských dovedností potřebných k orientaci v rozmanitých ekosystémech a odsouvá kapacity dále od skutečných potřeb. Navíc bez snadné návaznosti dat mezi Data Warehouse a Big Data a s více verzemi stejných dat je obtížné dosáhnout uživatelem požadované kvality dat.

V systému dávkového zpracování, jako je Hadoop, nás obvykle zajímá počet záznamů, které můžeme zpracovat za sekundu tzv. propustnost, nebo celkový čas potřebný k úspěšnému splnění úlohy na souboru dat určité velikosti. V moderních systémech však narůstá potřeba online zpracování. Příchozí množství dat může být z povahy IoT zařízení neomezené, protože jsou generována postupně v průběhu času. V určitých aplikacích je důležitější doba odezvy služby – tedy doba mezi odesláním požadavku klientem a odpovědí (21). Provoz pouze na platformě Hadoop tak nedokáže zajistit všechny současné požadavky. Dále je třeba zmínit, že s rozvojem rychlosti a kapacity národních a globálních sítí klesá potřeba ukládat množství dat na místních úložištích.

Alternativu k tradičnímu ETL představuje model Streaming ETL, někdy nazývaný ETL v reálném čase. V tomto modelu jsou data přijímána a zpracována hned v momentě zpřístupnění jejich zdrojem (22). Streaming ETL stejně jako jeho tradiční verze ETL zahrnuje extrakci dat, v případě Streaming ETL ve formě datového proudu různých formátů, transformaci datového proudu a jeho nahrání do úložiště (19). Extrahovaná data získaná z externích zdrojů je třeba uložit do paměti pro následnou transformaci v reálném čase. Transformace pak zahrnuje pro ETL typické operace jako filtrování dat, agregaci, spojování datového proudu jiným datovým tokem nebo obohacování proudu s již uloženými daty například v NoSQL databázi. Transformační fáze je značně rozmanitá. Zahrnuje jakýkoli typ transformace pro jakýkoli typ dat a také se v ní spouští řada pravidel a funkcí. Například funkce nahrazení dat během transformace může zahrnovat nahrazení chybících hodnot NULL průměrem měřených hodnot nebo nahrazení kategoriální hodnoty kódem. Pokročilejší transformace může zahrnovat komplexní spojování, agregaci řádků, rozdělení sloupců atd. Transformace dat nemusí probíhat jednorázově, k tomu, aby byla data ve správném formátu, může být zapotřebí více operací. Fáze také naskýtá příležitost k migraci dat z jednoho typu úložiště do jiného. Transformovaná data mohou být vložena do datového skladu nebo datového jezera běžícího na různých platformách. Přítomnost již transformovaných dat v paměti poskytuje příležitost pro analytické aplikace, které provádějí nad transformovanými daty různé operace jako jsou: deskriptivní analýza (23), prediktivní analýza a další metody strojového učení. Základy ETL pipeline tak můžeme využít i pro pokročilejší zpracování dat.

Práce s datovým proudem v reálném čase však čelí různým výzvám, jako výpadky spojení a následná ztráta datových proudů (19), zpoždění při spojování datových proudů (24), měnící se schéma datových proudů, škálovatelnost, složité dotazy na spojení, integrace heterogenních zdrojů dat (25), práce s omezenými zdroji, pře-použitelnost řešení, zpracování datových proudů v distribuovaném prostředí (19), hierarchické shlukování datových proudů , zvýšené nároky na data managment s ohledem na metadata. Další náročné výzvy vznikají při in-memory extrakci strukturovaných informací z nestrukturovaných dat nebo spojení datového proudu a informací uložených v různých databázích v distribuovaném prostředí (26). Použité nástroje by měli být schopny efektivně provádět přístupy k databázi, paralelizovat spojení mezi datovým proudem a úložištěm a následně zpracovávat dotazy v paměti. Efektivně eliminovat úzká hrdla operací, kdy počet příchozích proudů je větší než

rychlost zpracování strojem například za použití in-memory databáze, která eliminuje zdržení operace díky omezení načítání dat z úložiště.

3.2.3 Zpracování datových proudů

Zpracování datových proudů je technologie zpracování velkých objemů dat, která se zaměřuje na zpracování nepřetržitých datových toků v reálném čase. Metoda zpracování datových proudů často zahrnuje provádění četných operací nad příchozí sekvencí dat (datovým proudem), které lze provádět sériově, paralelně nebo kombinovaně (23). Takový pracovní postup se souhrnně nazývá Stream Computing. Zpracování datových proudů v reálném čase je klíčové pro práci s obrovským množstvím živých dat shromážděných z různých senzorů, systémových protokolů apod.

Prostředí Big Dat se v posledních letech přesouvá z čistě dávkového zpracování k zpracování datových proudů. V současné době již na trhu existuje dostatečné množství frameworků poskytující prostředí pro efektivní práci s těmito proudy. Frameworky zavádějí podporu relačních operací na vysoké úrovni, proto lze tyto nástroje pro proudové zpracování dat využít k oblasti ETL. Nástroje podporující tradiční proces ETL poskytují historicky podporu především pro strukturovaná data a jejich dávkové zpracování. Podpora pro pro zpracování semi-strukturovaných a nestrukturovaných dat v rámci Streaming ETL je zaváděna až v posledních letech.

Pro úspěšnou realizací operací s datovým proudem, musí být data v reálném čase zpracovávána sekvenčně a postupně po jednotlivých záznamech nebo v klouzavých časových úsecích, což lze dále využít pro řadu analytických úloh, jako jsou korelace, agregace, filtrování a vzorkování.

3.2.3.1 Rozdíl mezi časem vytvoření časem zpracování

Proudové procesory často potřebují pracovat s časem, zejména pokud se používají pro analytické účely, kde se často používají časová okna jako „průměr za posledních 10 minut“. Mohlo by se zdát, že význam „posledních 10 minut“ by měl být jednoznačný a jasný, ale bohužel tento pojem je překvapivě ošemetný (19). Pokud chceme znát čas kdy k události nebo datovému bodu skutečně došlo nemá smysl se spoléhat na systémové hodiny počítače na kterém datový proud zpracováváme. Čas zpracování procesu nemusí přesně odrážet kdy k událostem skutečně došlo. Data mohou díky nepředvídatelnosti internetového spojení přicházet zpožděná nebo nemusí přijít vůbec. Datové body pak mohou být zpracovávány

v jiném pořadí, než byly generovány, což může vést k chybám nebo nesrovnalostem v datech (19).

Dávkový proces tímto problémem většinou netrpí, ve většině případů je časovou osou zájmu dlouhá historie, nikoli několik minut případně sekund. Mnoho frameworků pro zpracování datových toků používá k určení oken lokální systémové hodiny na zpracovávajícím stroji. Tento přístup má výhodu v tom, že je jednoduchý a rozumný, pokud je prodleva mezi vytvořením události a jejím zpracováním zanedbatelně krátká. V opačném případě se musíme rozhodnout, jak zpožděně doručená data zpracovat.

3.2.3.2 Garance zpracování

V oblasti správy a zpracování dat se pojmy „přesně jednou“, „pouze jednou“ a „alespoň jednou“ vztahují k různým zárukám, které lze při zpracování dat poskytnout (19). Tyto termíny se běžně používají v kontextu distribuovaných systémů a proudových dat, kde mohou být jednotlivé datové body zpracovávány více systémy nebo aplikacemi a mohou být předmětem chyb nebo selhání.

Přesně jednou – je záruka, že datový bod bude zpracován ba právě jednou. To znamená, že datový bod bude zpracován systémem nebo aplikací a že toto zpracování bude úspěšně dokončeno bez chyb nebo selhání. Této záruky může být obtížné dosáhnout v distribuovaných systémech a při streamování dat, protože vyžaduje koordinaci a ošetření chyb v rámci více systémů a aplikací.

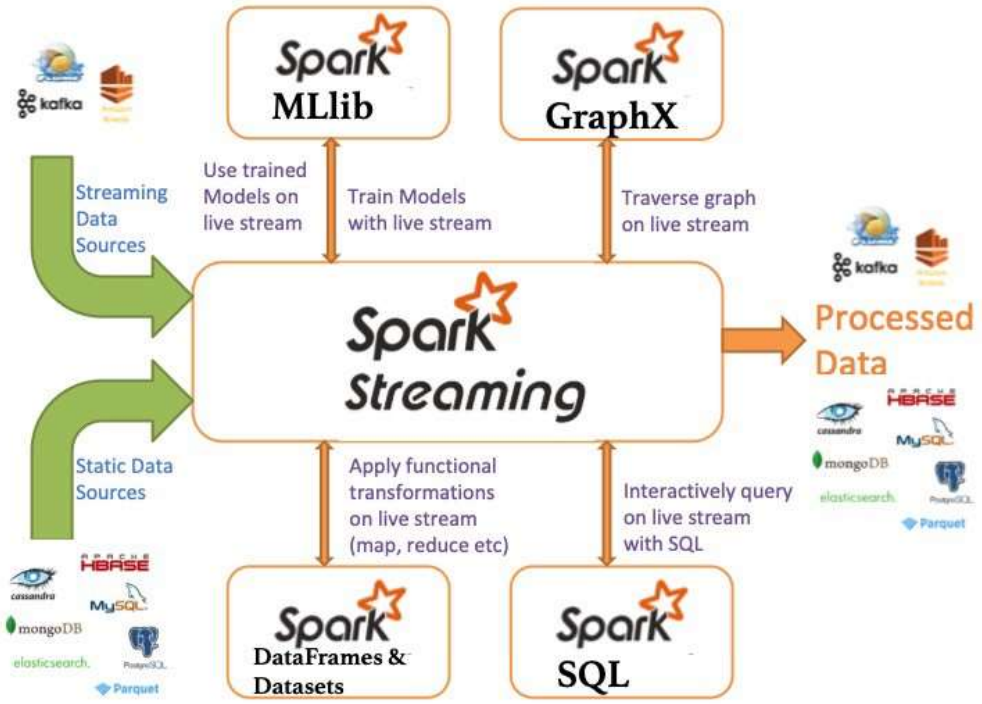
Pouze jednou – je záruka, že datový bod bude zpracován nejvýše jednou. To znamená, že datový bod může být zpracován systémem nebo aplikací, ale pokud je zpracován, bude zpracován pouze jednou a všechny další pokusy o jeho zpracování budou ignorovány. Této záruky lze dosáhnout snáze než záruky přesně jednou, protože nevyžaduje koordinaci a ošetření chyb v různých systémech a aplikacích.

Přinejmenším jednou – je záruka, že datový bod bude zpracován alespoň jednou. To znamená, že datový bod může být systémem nebo aplikací zpracován vícekrát, ale bude zpracován minimálně jednou. Této záruky lze dosáhnout snáze než záruky přesně jednou nebo pouze jednou, protože nevyžaduje koordinaci a zpracování chyb ve více systémech a aplikacích.

3.2.3.3 Nástroje pro zpracování datových toků

Technologie jako Apache Spark a Apache Flink jsou často základem nástrojů ETL jako je Amazon Kinesis, Talend. Tyto technologie poskytují základní „engine“ pro zpracování dat v reálném čase. V architektuře těchto nástrojů existuje několik klíčových rozdílů.

Obrázek 2 Schéma zpracování datových proudů s využitím Spark Streaming

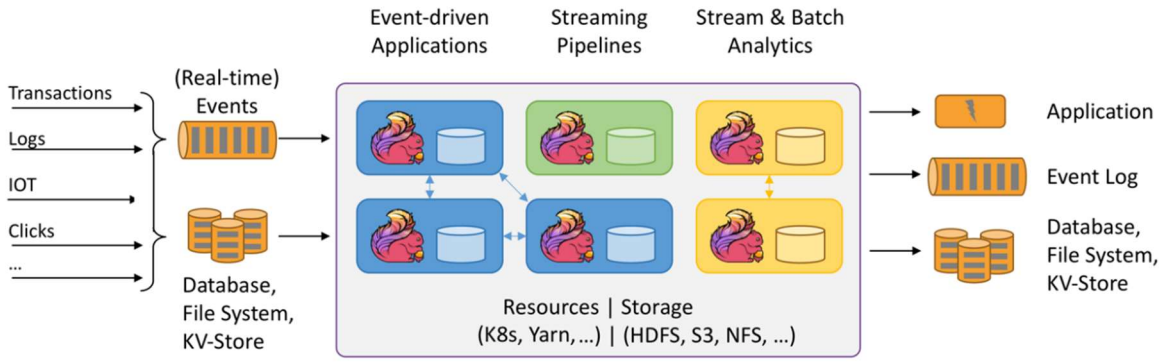


Zdroj: Apache Spark

Apache Spark se řadí mezi nejpopulárnější platformy pro zpracování datového proudu velkých dat (27). Základní datovou strukturou použitou v Apache Spark je Resilient Distributed Dataset (RDD). Jedná se o neměnnou, v rámci Spark clusteru distribuovanou kolekci objektů různého typu. Pro podporu zpracování proudu dat zavádí Spark interní komponentu Spark Streaming (obr. 2), která se vyznačuje škálovatelností, vysokým výkonem a odolností vůči chybám. Streamovací model použitý v Apache Spark příchozí datový proud automaticky paralelizuje a distribuuje data pro zpracování na jednotlivé uzly (worker-nody). Datový proud je v Apache Spark vysokoúrovňovou abstrakcí nazývanou diskrétní proud nebo DStream. Vzhledem ke skutečnosti, že DStream je interně reprezentován jako posloupnost časově ohraničených RDD vytvořených pomocí malých dávek, nelze označit Spark Streaming jako nástroj pro čisté real-time použití (28). Řešení

pomocí malých dávek však může být přijatelné pro řadu případů užití, které nemají tak striktní požadavky na nízkou latenci. DStream je možné vytvořit ze vstupních datových proudů, které pocházejí ze zdrojů, jako jsou Apache Kafka a Amazon Kinesis, nebo použitím vysokoúrovňových operací na jiných DStreamech. Uživatel má možnost uložení datového proudu přímo do souboru, přesměrovat výstup na konzoly nebo vrátit data zpět do Apache Kafky.

Obrázek 3 Schéma zpracování datových proudů s využitím Apache Flink

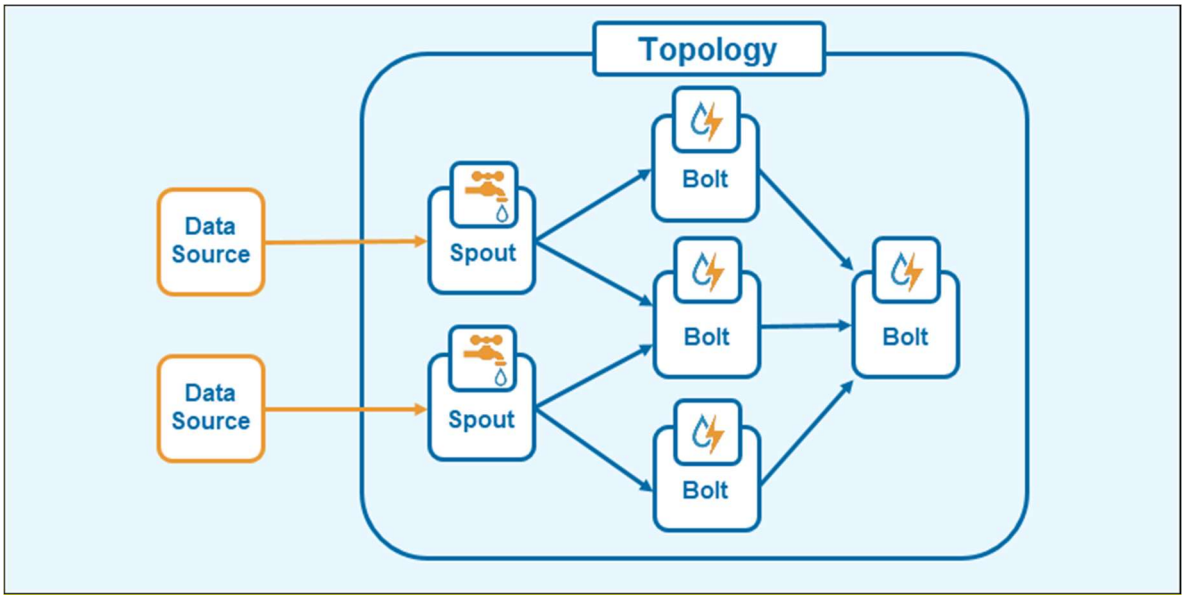


Zdroj: Apache Flink

Apache Flink dříve Stratosphere poskytuje framework pro zpracování datového proudu a analýzu dat v reálném čase (29). Engine se vyznačuje sjednocením práce s datovými proudy a dávkovým zpracováním do jednoho programovacího modelu (obr. 3). Pracuje ve spolupráci s trvalými frontami zpráv, které umožňují zpětné přehrávání datových proudů (jako Apache Kafka nebo Amazon Kinesis). Výpočetní jádro použité ve Apache Flink nerozlišuje mezi zpracováním nejnovějších událostí v reálném čase, průběžnou periodickou agregací dat ve velkých oknech nebo zpracováním velkých objemů historických dat. Různé typy výpočtů jednoduše zahajují své zpracování v různých bodech zachyceného datového proudu a během zpracování si udržují informaci o aktuálním stavu. Flink si tento stav ukládá do externí NoSQL databáze RockDB Tímto řešením odpadá nutnost kombinovat různé systémy pro zpracování datových proudů a dávek. Permanentně uložená informace o aktuálním stavu pak umožňuje automatické obnovení při selhání zpracování. Flink podporuje různá pojetí času (čas události, čas příjmu, čas zpracování) a poskytuje tak programátorům/analytikům potřebnou míru flexibility při definování způsobu korelace událostí.

Apache Storm je volně dostupný a otevřený systém pro distribuované zpracování datových proudů v reálném čase (30). Storm poskytuje jednoduché a efektivní programovací paradigma. Definuje základní stavební komponenty „spout“ a „bolt“ (obr. 4). Spout vystupuje jako zdroj dat, který extrahuje data z externích systémů a periodicky je zasílá dalším připojeným komponentám jako seznam hodnot libovolného typu. Bolt je komponenta, která přijímá data od spoutů nebo jiných boltů a provádí následně programátorem definované výpočty. Programátoři musí implementovat své vlastní spouts a bolts a specifikovat, jak jsou tyto komponenty navzájem propojeny, generují acyklický orientovaný graf datových proudů, který se v prostředí Storm nazývá topologie. Tento klíčový objekt abstrahuje úlohu, kterou lze zpracovat na výpočetním clusteru. Programy Storm jsou robustní, díky své podstatě paralelizované, podporující distribuci příchozích dat a v případě potřeby umožňují opakování pokusu při selhání. Storm, výrazně usnadňuje postup při implementaci paralelních výpočtů v reálném čase.

Obrázek 4 Schéma zpracování datových proudů s využitím Apache Storm



Zdroj: Apache Storm

3.2.3.4 Vývoj ve zpracování datových proudů

Problému s heterogenními zdroji dat a práci s nimi se věnují Meehan a kol., popisují engine pro zpracování datových proudů nazvaný S-Store a jeho roli v systému BigDAWG Polystore (31). S-Store zde funguje jako frontendový procesor, který přijímá vstupní data z různých zdrojů a zpracovává je do podoby, která eliminuje chyby (čištění dat), a převádí tato vstupní data do podoby, kterou lze efektivně vložit do systému BigDAWG. Klíčovým

prvkem je možnost migrace dat z jedné komponenty do druhé komponenty BigDAWG (aktuálně integrované databáze PostgreSQL, SciDB a Accumulo). Výhoda tohoto řešení je ve sjednocení doménově specifických databázových enginů pro danou datovou oblast a z toho plynoucí zisky ve výkonu zpracování oproti univerzálnímu řešení.

Weiping Qu poukazuje na různé požadavky analytických aplikací, jako rozdílnou přípustnou latenci a čerstvost dat (32). Tvrdí, že nedostatečná flexibilita ELT procesů využívající „real-time data“ nebo malé dávky dat může způsobit nadměrnou rezervaci zdrojů pro dotazy s nízkou potřebou čerstvosti, zatímco úlohy s vysokou potřebou čerstvosti mezi tím čekají ve frontě. Proto tvrdí, že úlohy ETL by se měly samy přizpůsobovat měnícím se požadavkům odběratelů nastavením vhodné velikosti dávek podle jejich skutečné potřeby. Definuje model konzistence pro „On-Demand ETL“ a zavádí inkrementální ETL pipeline nazvanou HBelt.

Jang-Ho Choi přichází s distribuovaným frameworkem pro práci s datovými proudy a zároveň optimalizovaným pro použití ve světě IoT (33). Poukazuje, že cloudová integrace nemusí být vhodná pro služby, které vyžadují okamžitou reakci, pokud je server umístěn vzdáleně, nebo pro služby, které zpracovávají citlivá data, jež chtějí uživatelé zpracovávat lokálně. Jeho odlehčený framework pro zpracování datových proudů pro IoT nazvaný DART, umožňuje nový způsob zpracování datového proudů ve spolupráci s plně distribuovanými zařízeními IoT.

3.2.4 Záchyt datového proudu

Jedním z možných požadavků na implementaci vrstvy sběru dat ve Stream ETL je distribuované místo pro prvotní záchyt datových proudů (34). Oblast se obvykle používá jako pracovní prostor, kde lze data čistit, ověřovat a transformovat v rámci přípravy pro načtení do cílového systému. Z mnoha dostupných řešení se pro získávání dat z různých heterogenních zdrojů široce používá Apache Kafka společnosti Confluent. Kafka se pro využití ve Stream ETL hodí díky své schopnosti přijímat rostoucí objemy dat z nestrukturovaných a semistrukturovaných datových zdrojů a pružnosti reagovat na měnící se prostředí (škálovatelnost, registr schémat). V současné době se Kafka široce využívá v různých organizacích včetně ING, Bosch, Disney, Twitter, Walmart atd.

Apache Kafka je open source distribuovaný systém událostí zaznamenaných ve formě logu (35). Tvořený je clusterem několika brokerů schopných ukládat data přiřazená k daným

tématům. Systém může být implementován v cloudovém nebo na „on-premise“ prostředí. Zdroje dat v Kafce označení jako producenti, zapisují data do určitého tématu (topic) rozděleného do oddílů (partitions). Jeden z hlavních benefitů při použití Kafky je její horizontální škálovatelnost. Ta je dosažena možnosti konfigurace počtu oddílů. Konfigurace se provádí v době vytváření tématu, změna počtu oddílu je však možná i po nasazení (repartitioning).

Ztrátě dat v důsledku selhání oddílu lze zabránit použitím replikačního faktoru. Tento faktor závisí na počtu očekávaných selhání brokerů a na počtu odběratelů v daném tématu. Pokud je pravděpodobnost selhání brokerů vyšší, lze nastavit vyšší replikační faktor a podobně lze nastavit vyšší replikační faktor, pokud je v tématu více odběratelů. Kafka má možnost uchovávat zprávy v tématu navždy (infinity topic), proto neexistuje možnost ztráty příchozích dat ani možnost výskytu datové nekonzistence. V „on-premise“ prostředí neexistuje nekonečné místo na disku, proto pokud nemá dojít k vyčerpání disku nebo k nadměrné finanční náročnosti na uložení dat v cloudu, lze nastavit politiku uchovávání, která automaticky odstraní zprávy starší než určité časové období. Další možnost retence dat je pomocí tzv. „Compacted Topic“. Za pomoci této funkce lze v Kafce starší zprávy na základě shodného klíče automaticky odstranit. Jakmile přijde nová zpráva obsahující daný klíč, Kafka prohlédá oddíl a odstraní neaktuální hodnoty s daným klíčem. Oddíl tak udržuje pouze nejaktuálnější hodnoty jednotlivých klíčů. Kafka navíc umožňuje ke každé zprávě připojit časovou značku díky funkci „LogAppendTime“, a obohatit tak zprávu o informaci dostupnou pro následné zpracování odběratelem.

3.2.5 Datová úložiště

Pro uložení dat z IoT přichází v úvahu několik možností. Volba vhodného datového úložiště záleží primárně na daném případu užití. Mezí další faktory výběru lze zařadit například omezení daná regulátorem.

3.2.5.1.1 Datové jezero Data Lake

Datové jezero je i v oblasti velkých dat poměrně novým konceptem. Stejně tak jeho využití, architektura, implementace a použití. Přestože je koncept datového jezera celkem jednoduchý, jeho integrace do systémů není zdaleka tak rozšířená, jako použití datových skladů. Základní rozdíl mezi jezerem a datových skladem je ten, že všechna data vytvořená zdrojovým systémem jsou v jezeře uložena v jediné datové struktuře (36). Data jsou v jezeře

uložena ve své primární často nestrukturované podobě. Před uložením dat není nutné žádné komplexní zpracování nebo jejich transformace. Data uložená v datovém jezeře mohou sloužit například pro potřeby analýzy týmu datových analytiků/scientistů, kteří jsou z dat schopni vytvořit nové dodatečné informace. Jezero může také sloužit pro potřebu obnovy dat v jiném datovém úložišti po jejich neočekávané ztrátě (disaster recovery).

3.2.5.1.2 Datový sklad Data Warehouse

Datový sklad je datové úložiště, které provádí agregaci a shromažďuje a seskupuje data z různých zdrojů do jednoho centrálního objektu (37). Data z datového skladu lze načítat a analyzovat za účelem vytváření sestav nebo vztahů mezi jednotlivými datovými sadami. Koncept datových skladů se poprvé objevil v 80. letech 20. století, kdy výzkumníci společnosti IBM Paul Murphy a Barry Devlin vyvinuli první podnikový datový sklad (38).

Koncem minulého století se dostal do popředí zejména názorový proud na architekturu datového skladu zformovaný okolo Raplha Kimballa, který razil tezi, že *„Datový sklad není nic jiného než sjednocení datových tržišť“*. Tržiště si zde můžeme představit jako jednoduchou formu datového skladu, které obsahuje pouze určenou podmnožinu dat se zaměřením pouze na jeden subjekt či doménu. Spojením těchto tržišť pak dostáváme datový sklad. Univerzální vzor pro výrobu datového skladu je vytvořen za pomocí metod dimenzionálního modelování. Mezi slabé stránky tohoto přístupu lze zařadit především výkonnostní problémy spojené s dimenzionální strukturou dat. Transformace dat do této struktury je časově náročná. SQL dotazy datových analytiků často nereflektující tuto strukturu, což má za důsledek snížený výkon SQL enginu.

V současné době existují nové specializované architektury řešení s možností využití pro data z IoT. Datový sklad časových řad například poskytuje možnost vykazovat a analyzovat data v pravidelných intervalech ze zdrojů, jako jsou senzory, zařízení a další objekty IoT. Řešení postavené například na platformě Cloudera (39) umožňuje dotazování v reálném čase, včetně pokročilých analytických úloh, jako je statistické modelování a strojové učení. Vše s optimalizací pro časové řady.

3.3 Přehled dostupných ETL nástrojů

Nástroje ETL mohou být kategorizovány podle různých kritérií, jako je typ licenční politiky, typ služby, nebo platforma, na které se specializují (41, 43). Dále lze brát v úvahu typ dat, pro které je nástroj optimalizován, jako jsou například data z IoT nebo časové řady.

Tyto nástroje mohou být rozmanité a spadat do více kategorií. Například, komerční ETL „stand-alone“ nástroj může zahrnovat funkce pro práci s daty z IoT a časovými řadami. Je nutné zdůraznit, že mnoho nástrojů ETL je velmi přizpůsobitelných a lze je upravit pro specifické potřeby. Z tohoto důvodu není jednoduché učinit obecné závěry o schopnostech všech nástrojů ETL v dané kategorii, ale kategorizace může být užitečná pro přehled.

3.3.1 Z hlediska licenční politiky

Komerční nástroje ETL vyvíjejí a prodávají komerční dodavatelé. Nástroje obvykle nabízejí širokou škálu funkcí např. použití grafických uživatelských rozhraní (GUI) pro architekturu ETL, podporu většiny relačních i nerelačních databází, rozsáhlou dokumentaci a další pokročilé funkce. Nevýhoda komerčních řešení spočívá v jejich cenové náročnosti. Další faktor jejich používání je potřeba „know-how“, které může vyžadovat specializované školení.

Existuje několik komerčních nástrojů ETL, které lze využít pro práci s daty IoT. Mezi populární nástroje lze uvést:

Informatica Intelligent Cloud Services – tato cloudová platforma pro integraci dat umožňuje připojit se k různým zařízením IoT. Informatica navíc nabízí specifické konektory IoT pro populární platformy IoT, jako je AWS IoT, Azure IoT a další, které podporují snadné připojení a umožňují následně shromáždit data a analyzovat je v reálném čase.

Talend: Talend – nástroj pro integraci dat, který poskytuje širokou škálu konektorů a předpřipravených funkcí pro integraci dat ze zdrojů dat IoT, jako jsou MQTT, AMQP a Kafka.

SAP Data Services – nástroj pro integraci dat, který poskytuje podporu pro data IoT prostřednictvím konektoru „Internet of Things“, který umožňuje shromažďovat a zpracovávat data IoT z různých zdrojů.

IBM InfoSphere Information Server – platforma pro integraci dat, která poskytuje podporu pro data IoT prostřednictvím konektoru „Internet of Things“, který umožňuje shromažďovat, zpracovávat a analyzovat data IoT.

Microsoft SQL Server Integration Services – v rámci této platformy je k dispozici nástroj pro integraci dat, který podporuje data IoT prostřednictvím konektorů pro Azure IoT

Hub a Event Hub, které umožňují shromažďovat, zpracovávat a analyzovat data internetu věcí.

Open-source nástroje ETL jsou vyvíjeny a udržovány komunitou dobrovolníků nebo jsou vyvíjeny v akademickém prostředí. Někteří poskytovatelé komerčních řešení nabízí vedle vlastních produktů i open-source variantu. Použití těchto nástrojů je obvykle zdarma, ale oproti komerčním řešením nabízejí pouze základní škálu funkcí. Výraznou výhodou open-source řešení je, že organizace mohou přistupovat ke zdrojovému kódu a studovat vnitřní strukturu nástroje a dále rozšiřovat jeho možnosti podle svých specifických potřeb.

3.3.2 ETL v rámci platformy

Nástroje ETL mohou být samostatným softwarem, který se instaluje a spouští v místním prostředí, nebo mohou být součástí větší služby či platformy. Příkladem nástroje ETL, který je součástí větší služby, je cloudová platforma pro integraci dat. Tyto platformy nabízejí řadu funkcí datové integrace a ETL jako službu (45) a uživatelé k nim mohou přistupovat prostřednictvím webového prohlížeče nebo rozhraní API. Specifickou výhodou nástrojů ETL založených na cloudu je efektivita. Cloudová technologie poskytuje přijatelnou latenci, vysokou dostupnost a pružnost, takže výpočetní zdroje se škálují podle aktuálních požadavků na zpracování dat. Při využití služeb stejného poskytovatele pro datovou integraci a datové úložiště lze díky tomu, že všechny datové procesy probíhají v rámci stejné sdílené infrastruktury datovou pipeline značně optimalizovat. Nevýhodou cloudových nástrojů ETL je, že často pracují pouze v prostředí daného cloudového poskytovatele. To znamená, že nepodporují data uložená u jiných cloudových poskytovatelů nebo v lokálních datových centrech.

Azure Data Factory je plně spravovaná služba datové integrace, která umožňuje vytvářet, plánovat a orchestrovat datové pipeline pro přesun a transformaci dat. Dokáže přijímat data z nejrůznějších datových zdrojů, včetně služeb Azure, lokálních databází a souborových systémů. Obsahuje také rozhraní drag-and-drop, které usnadňuje vytváření, konfiguraci a správu datových pipeline, a také výkonnou sadu vývojářských nástrojů a rozhraní API, které umožňují vytvářet vlastní řešení datové integrace.

Kinesis a Azure Analytic Stream jsou dvě vedoucí platformy pro streamování dat v rámci cloudového prostředí (zdroj gartner), které nabízí společnosti Amazon a Microsoft a jsou vhodné pro využití v kontextu IoT. Ačkoli jsou obě platformy navrženy tak, aby

zpracovávaly datové toky a umožňovaly organizacím provádět analýzu a zpracování dat v reálném čase, existují mezi nimi některé klíčové rozdíly.

Amazon Kinesis je založen na architektuře „shared-nothing“, kde je každý datový proud zpracováván nezávisle samostatnou službou. Azure Analytic Stream je založen na architektuře shared-everything, kde je více streamů zpracováváno společně sdílenou službou.

Google Dataflow je cloudová služba pro zpracování dat, která uživatelům umožňuje vyvíjet a provádět dávkové i proudové operace. Dataflow poskytuje jednotný programovací model založený Apache Beam, který podporuje více programovacích jazyků včetně jazyků Java, Python a Go. Beam poskytuje sadu SDK, která vývojářům umožňuje vytvářet datové pipeline pomocí preferovaného jazyka. Model zpracování Dataflow je založen na konceptu transformací dat, které jsou vyjádřeny jako řada kroků v rámci pipeline. Tyto transformace mohou být buď dávkové, nebo proudové, v závislosti na potřebách uživatele. Streamovací model Dataflow využívá koncept časových oken ke sdružování datových prvků ke zpracování.

Dalším příkladem nástrojů ETL, které jsou součástí rozsáhlejší služby, jsou nástroje postavené na platformách pro Business Intelligence (BI). Mnoho platforem BI zahrnuje funkce ETL jako součást své nabídky a umožňuje uživatelům extrahovat, transformovat a načítat data z různých zdrojů do datového skladu nebo jiného cílového systému. Příklady platforem BI s funkcemi ETL jsou reportovací nástroje Tableau, Power BI a Qlik.

3.3.3 Vlastní řešení ETL

Společnosti s kapacitou pro vývoj softwaru mohou vytvářet vlastní ETL řešení pomocí obecných programovacích jazyků a dostupných open-source frameworků pro distribuované zpracování. Hlavní výhodou tohoto přístupu je flexibilita při vytváření řešení, které je tak možné přizpůsobit prioritám a pracovním postupům organizace. Jako jazyky pro tvorbu procesů ETL se prosazují především Python, Java a Scala. Pro tvorbu vlastního ETL řešení se nabízí využití spojení vlastního kódu s plánovačem procesů jako Apache Airflow, či využití knihoven vybavených funkcionalitou ETL jako Pygrametl, Petl, Scriptella (40).

Největší nevýhodou tohoto přístupu jsou nároky na vlastní zdroje potřebné k vytvoření vlastního nástroje, včetně zabezpečení dalších požadavků spojených s vývojem softwaru jako testování, údržba a odstraňování chyb. Další nároky jsou kladeny na výrobu kvalitní

dokumentace a průběžné proškolování a podporu nových uživatelů, kteří chtějí nástroj užívat.

3.3.4 Specializované nástroje

Existuje několik nástrojů ETL, které jsou speciálně navrženy pro práci s daty IoT. Tyto nástroje jsou optimalizovány pro extrakci dat z velkého počtu zařízení a senzorů IoT, transformaci dat do formátu vhodného pro analýzu a reporting a načtení dat do cílového systému, jako je datové jezero nebo datový sklad. Mezi příklady nástrojů ETL specializovaných na data IoT lze zařadit např:

Flux – open-source nástroj ETL vyvinutý společností InfluxData, který je speciálně navržen pro práci s časovými řadami dat ze zařízení IoT.

StreamSets – komerční nástroj ETL, který obsahuje řadu funkcí pro práci s daty internetu věcí, včetně možnosti přijímat data z nejrůznějších zdrojů a podpory zpracování dat v reálném čase.

Apache Nifi – open-source nástroj ETL, který obsahuje řadu funkcí pro práci s daty IoT, včetně možnosti zpracovávat data v reálném čase a podpory široké škály datových zdrojů a formátů.

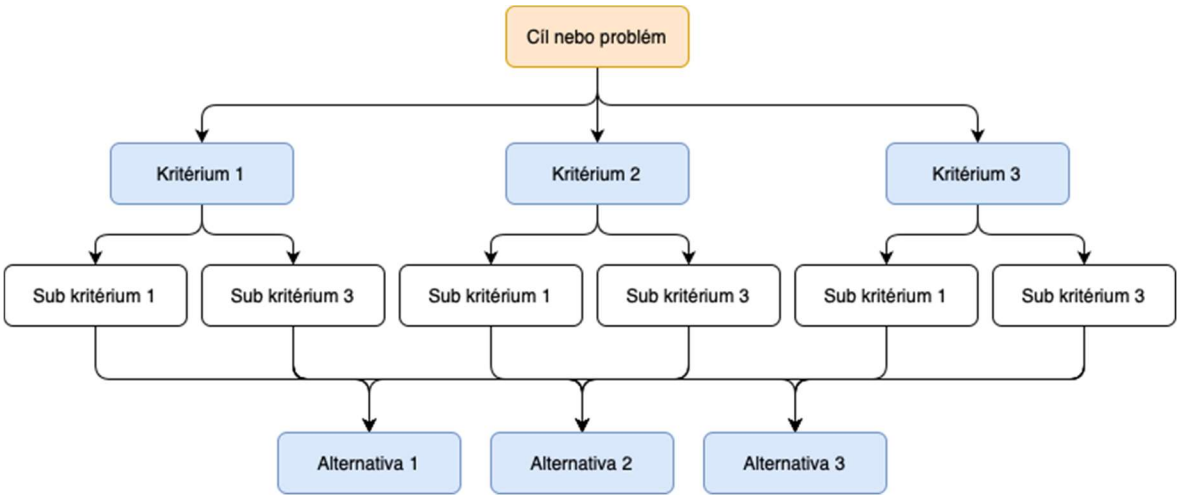
3.4 Metodika výběru ETL nástroje

3.4.1 Analytický hierarchický proces

Analytický hierarchický proces (AHP) je metoda pro podporu a analýzu složitých rozhodnutí s využitím poznatků z matematiky a psychologie (42). AHP poskytuje racionální rámec potřebné pro rozhodnutí tím, že kvantifikuje poměr mezi kritérii a alternativními možnostmi a vyjadřuje vztah těchto prvků k celkovému cíli. Problém je modelován jako hierarchie obsahující několik na sobě závislých částí (obr. 5).

V kontextu výběru nástroje ETL lze AHP použít k vyhodnocení různých možností na základě souboru kritérií, jako je výkon, škálovatelnost, konektivita a náklady. Je však zásadní kritéria, stejně jako váhy použité v procesu hodnocení velmi pečlivě definovat. V modelu AHP musí být zajištěn přesný obraz konkrétních požadavků pro potřeby daného případu použití.

Obrázek 5 Čtyř úrovňová hierarchie AHP



Zdroj: vlastní práce

Metody párového porovnání poskytují základ pro stanovení vah jednotlivých kritérií a jejich dílčí částí. Váhy umožňují kvantifikovat důležitost každého kritéria a seřadit je tak od nejdůležitějšího po nejméně důležité. Aby tyto váhy odrážely skutečnou důležitost kritérií, musí poskytovat informaci o tom, o kolik je jedno kritérium důležitější (méně důležité) než jiné. Váhy dle zvolené srovnávací metody mohou znázorňovat různé vlastnosti, proto je interpretace vah kritérií u jednotlivých metod odlišná. K porovnání kritérií ve dvojicích, a to nejen z hlediska toho, které kritérium je důležitější než druhé, ale také z hlediska toho, o kolik je důležitější, je zapotřebí definovat stupnici, která bude toho hodnocení odrážet.

V Saatyho metodě párových porovnání znázorňují poměry důležitost mezi jednotlivými kritérii. Pro tento účel navrhl Saaty devítibodovou škálu pro určení důležitosti různých prvků. Vyjádření těchto intenzit je podpořeno doplňujícími slovními popisy, které umožňují rozhodovateli snadněji a přesněji vyjádřit své preference. Stupnice v základu obvykle nabývá pouze lichých hodnot, pokud je potřeba kompromis mezi dvěma stupni je možné využít také sudé hodnoty.

3.5 Obdobná řešení

Odborná literatura se zaměřuje ve větší míře na různé aspekty datové integrace než na výběr konkrétních ETL nástrojů, přesto existuje několik textů, které se podobnou problematikou zabývají. Většina autorů však srovnává jednotlivé nástroje nebo platformy na základě vlastnoručně určených kritérií. Pro srovnání různých řešení datové integrace by bylo

vhodnější použít unifikované řešení. Pro porovnání DI architektury potažmo ETL nástrojů lze využít TPC Decision Support Benchmark (TPC-DI), což je sada srovnávacích testů vyvinutá organizací Transaction Processing Performance Council (TPC) (44). Hodí se pro hodnocení výkonu systémů, které provádějí složité dotazy na podporu rozhodování a úlohy datové integrace s využitím velkých datových sad. Bohužel TPC doposud nezveřejnilo pro tento benchmark žádné výsledky.

Společnost Gartner každoročně hodnotí postavení relativních konkurentů na trhu v oblasti datové integrace (46). Výzkum zahrnuje dodavatele, kteří nabízejí samostatný softwarový produkt umožňující vytvoření a implementaci moderní datové infrastruktury pro různé případy užití. Ve výzkumu nejsou zahrnuty nástroje, které jsou poskytovány jako části platforem s jiným primárním zaměřením, než je datová integrace. Výsledek výzkumu z roku 2022 provedený na základě srovnání 21 poskytovatelů zobrazuje graf 1 v podobě Magic Quadrant.

Graf 1 Gartner Magic Quadrant pro nástroje datové integrace

Zdroj: Gartner (2022)

V knize (47) se autor Jarrett Goldfedder zaměřuje na problematiku vytváření efektivních týmů datové integrace. V kapitole Choosing ETL tool popisuje, jakým způsobem strategicky plánovat migraci dat a integraci, aby se předešlo problémům na

poslední chvíli a jak vytvářet správná řešení pro budoucí integrační projekty. Jednou z klíčových částí je výběr ETL nástrojů pro úspěšnou integraci dat. Autor se věnuje možnostem, které jsou k dispozici pro výběr správných nástrojů a probírá několik nástrojů ETL, které jsou v současné době na trhu a které budou nejlépe vyhovovat integračním potřebám. Autor zdůrazňuje, že výběr vhodného nástroje ETL je zásadní a měl by zohledňovat typ organizace, typ použití, stávající úložiště dat, konektivitu, zpracování v reálném čase, možnosti integrace dat, podporu vizualizace a cenu.

Cílem článku (48) je studium stávajících nástrojů ETL a jejich funkcí a za druhé identifikace zásadních funkcí nástroje ETL pro daný podnik nebo organizaci. V článku se autor zaměřuje na kategorizaci nástrojů ETL a dostupné nástroje pro jednotlivé typy organizací. Autor knihy tvrdí, že mnoho nástrojů pro extrakci, transformaci a načítání dat může spadat do více než jedné třídy vzhledem ke svým funkcím a technikám implementace. Výběr vhodného nástroje ETL je podle autora zásadní pro každou organizaci, ale vybraný nástroj ETL by neměl být více komplexní, než je skutečně potřeba. Pokud organizace nepotřebuje aktualizace v reálném čase a nepotřebuje zpracovávat proudové datové sady, může si vybrat jakýkoli jednoduchý nástroj, který může vyhovovat jejím požadavkům. Pokud organizace pracuje s velkým množstvím dat nebo s proudovými daty, může si nástroj ETL vytvořit na základě technologie open-source. Autor též konstatuje, že existuje mnoho zásadních kritérií pro proces výběru nástrojů ETL, jako je typ organizace, typ použití, stávající úložiště dat, konektivita, zpracování v reálném čase, možnosti integrace dat, podpora vizualizace a cena. Na trhu je sice k dispozici celá řada nástrojů ETL, avšak stále existuje prostor pro zlepšení, zejména v oblasti open-source technologií pro malé podniky, které by poskytovaly více funkcí v jednom nástroji s přijatelnou cenou.

Diplomová práce (49) se zabývá integrací strukturovaných a nestrukturovaných dat pro využití v Business Intelligence. Autor v praktické části práce využívá data z bankovního prostředí, pro které navrhuje vhodný datový model v 3NF a posléze architekturu jednotlivých vrstev pro datovou integraci dat ve skoro reálném čase. Zvolený případ užití se věnuje především zpřístupnění dat pro použití v rámci reportovacích nástrojů. V další části práce se autor věnuje srovnání ETL nástrojů Oracle Data Integrator a Pentaho Data Integration z hlediska rychlosti přenosu testovacích dat mezi jednotlivými vrstvami, kde nástroj společnosti Oracle dosahuje řádově lepších výsledků. V poslední části práce hodnotí a posléze srovnává reportovací nástroje na základě výsledků testu.

Výběru vhodné IoT cloud platformy pro realizaci komplexního řešení postaveném na technologii IoT se věnuje bakalářská práce autora Martina Miltona (50). Autor definuje kritéria sestavená na základě průzkumu relevantní literatury bez zaměření na konkrétní případ užití. Jako alternativy pro výběr zahrnuje do hodnocení produkty od společností Microsoft, Google a Amazon, které posléze vyhodnocuje. Výběr IoT platformy se z určité části prolíná s výběrem vhodného ETL nástroje obsahuje však části, které nejsou pro výběr ETL nástroje zásadní jako např. zabezpečení. Vzhledem k volbě obecných kritériích bez určitého zaměření nemusí být zvolený nástroj tím nejvhodnějším pro konkrétní případ užití.

4 Vlastní práce

Kapitola se zabývá výběrem ETL nástroje fiktivním zemědělským podnikem. Kritéria modelu AHP byla zvolena s ohledem na popsaný případ užití. Následně byly vybrány vhodné alternativy, které byly na základě zvolených kritérií ohodnoceny.

Praktická část se věnuje datové integraci IoT systému sázecích strojů s možností přímého připojení k internetu. Práce vychází z předpokladů, že daný zemědělský podnik chce v rámci cloudového prostředí integrovat data v reálném čase. Vzhledem k tomu, že na sadbu mají zemědělci pouze jeden pokus ročně, je požadavek na spolehlivou integraci dat velmi důležitý. Z důvodu možnosti včasného odhalení případných problémů a optimalizaci secího procesu chce podnik data zpracovat co nejblíže času jejich vytvoření.

Podnik provozuje moderní secí stroj, který je tvořen soustavou více samostatných secích zařízení a meteorologickou stanici. Tato zařízení jsou vybavena řadou senzorů, které se používají k monitorování a řízení různých aspektů procesu sázení. Secí zařízení obsahuje:

- Senzory osiva – zjišťují přítomnost a vzdálenost semen při jejich sázení a lze je použít k úpravě rychlosti sázení nebo vzdálenosti, aby se zajistilo, že rostliny budou rovnoměrně rozmístěny a sadba bude mít správnou hustotu.
- Hloubkové senzory – zaznamenávají hloubku sázecího nástroje při jeho vstupu do půdy a lze je použít k zajištění správné hloubky zasetí semen.
- Pneumatické senzory – zajišťující provoz a kontrolu pohonu sázecího systému.

Další využitelná data produkuje samotný sázecí stroj pomocí senzorů GPS, kde tyto data lze je použít ke kontrole způsobu sázení a zajištění správného umístění budoucích rostlin. Kromě těchto datových bodů mohou secí stroje shromažďovat i další informace, například o rychlosti a směru jízdy secího stroje, provozních podmínkách a výkonu různých systémů stroje. Více o zdrojových datech v části 4.2.1 Model zdrojových dat.

V práci byl uvažován secí stroj vybavený 32 jednotkami secího zařízení. Každá z těchto jednotek snímá data ze šesti senzorů na frekvenci 5 Hz. Stroj samotný zasílá data ze tří senzorů o stejné frekvenci. Meteorologická stanice zasílá data ze dvou senzoru o frekvenci 2 Hz. Celkový datový tok se rovná 979 záznamům za sekundu. Data jsou do cloudového prostředí zasílána skrze IoT bránu. Jako komunikační protokol byl vybrán MQTT, který se často využívá v systémech IoT a odpovídá danému případu užití.

Samotný ETL proces se skládá z několika dílčích kroků:

1. Čtení příchozích datových proudů a extrahování měřených hodnot ze surových dat;
2. Vyčištění datových proudů od nulových hodnot;
3. Rozčlenění spojeného datového proudu na rovnoměrná okna dle pevného intervalu jedné minuty;
4. Vypočtení agregací pro jednotlivé parametry v rámci jednotlivých oken;
5. Spojení datového proudu pocházejícího z jednotek s datovým proudem ze stroje s datovým proudem meteostanice (tento krok zajišťuje sjednocení datových proudů a umožňuje jejich další zpracování jako jednoho celku);
6. Uložení vypočtených hodnot do datového úložiště pro následnou analýzu.

4.1 Vlastnosti nástroje ETL

Vzhledem k poměrně velkému počtu dodavatelů nástrojů ETL, rozsahu možných případu užití a různých schopností ETL nástrojů je velmi obtížné až nemožné vybrat nástroj, který by univerzálně pokrýval všechny potřeby (47). Například některé nástroje mohou být lepší při extrakci dat z relačních databází, zatímco jiné mohou být lepší při extrakci dat z plochých souborů, webových služeb nebo zařízení IoT. Rozdílné přístupy se objevují i ve fázi transformace dat. Část nástrojů ETL umí pracovat pouze v dávkovém způsobu zpracování. Jiné nástroje jsou určené přímo pro zpracování datového proudu. Rozdíly mezi nástroji ETL se projevují i z hlediska nákladů. Některé nástroje mohou mít jednorázový licenční poplatek, zatímco jiné mohou účtovat měsíční nebo roční předplatné. Při hodnocení nástroje ETL lze mít na paměti níže uvedené obecné aspekty.

4.1.1 Vývojové prostředí

Obecně lze nástroje ETL dle přístupu k vývoji rozdělit do dvou kategorií vývojového paradigmatu na nástroje „no-code“ a nástroje založené na zdrojovém kódu.

Nástroje ETL založené na kódu jsou tradiční nástroje ETL, které vyžadují, aby uživatelé psali a spravovali vlastní kód pro všechny fáze ETL procesu. To může vyžadovat specializované programátorské dovednosti a odborné znalosti a pro netechnické uživatele může být používání nástroje obtížnější.

Naproti tomu nástroje ETL „no-code“ nevyžadují, aby uživatelé psali nebo spravovali kód. Místo toho poskytují vizuální rozhraní nebo grafické uživatelské rozhraní (GUI), s předem připravenými funkcemi a komponentami, které uživatelům umožňuje navrhovat a spravovat proces ETL bez zdlouhavého psaní kódu. To může usnadnit používání nástroje netechnickým uživatelům a může snížit čas a úsilí potřebné k vývoji a údržbě procesů ETL.

Volba mezi nástroji ETL založenými na kódu a bez kódu závisí na konkrétních potřebách a cílech organizace. Nástroje ETL založené na kódu mohou být lepší volbou pro organizace, které mají silný technický tým a potřebují flexibilitu a výkon poskytnutý využitím vlastního kódu, zatímco nástroje ETL bez kódu mohou být lepší volbou pro organizace, které chtějí zpřístupnit proces ETL netechnickým uživatelům.

4.1.2 Architektura

Při výběru vhodného nástroje s ohledem na architekturu je třeba vzít v úvahu aspekty, jako je podpora paralelního zpracování, masivního multiprocesingu (MPP) a vyvažování zátěže (load balancing). Rovněž je třeba vzít v úvahu podporu pro víceuživatelskou správu procesů ETL běžících na více strojích a podporu pro vytváření a sdílení datového modelu.

Další aspekt je způsob zpracování dat, jak bylo popsáno výše. Stream ETL může být lepší volbou pro aplikace, které vyžadují schopnost zpracovávat a analyzovat data v reálném čase, zatímco micro-batch ETL může být lepší volbou pro aplikace, které vyžadují schopnost zpracovávat větší objemy dat a provádět složitější transformace.

Do úvahy je nutné vzít i spolehlivost daného nástroje. Pokud aplikace potřebuje pokračovat v práci i v případě výpadku jednoho stroje (nebo několika strojů, sítě či celého datového centra), lze pro zpracování použít více strojů, které poskytnou potřebnou redundanci, když jeden stroj selže, může jeho zátěž převzít jiný.

4.1.3 Funkčnost

Hlavní funkčnost se zaměřuje na to, zda je nástroj orientován na čištění dat, nebo na transformaci dat, případně zda provádí obojí. Lze tak získat jasnou představu o tom, jaký nástroj vybrat v závislosti na povaze dat, která mají být do nástroje vložena.

Důležitým aspektem funkčnosti je také podpora nástroje ETL připojit se k různým zdrojům ať už pro získání požadovaných dat nebo schopnosti načíst transformovaná data do cílového systému. Tato konektivita je důležitou vlastností nástroje ETL, protože umožňuje

uživatelům pracovat s širokou škálou datových zdrojů, včetně databází, plochých souborů a dalších typů datových úložišť. Pro využití v oblasti IoT je často nutné integrovat ETL nástroje s dalšími technologiemi a platformami používanými v systémech IoT, jako jsou platformy IoT, datová jezera a výpočetní enginy pro zpracování datových toků, brány pro příjem IoT dat. Schopnost integrace s dalšími technologiemi je důležitá pro zajištění bezproblémového komplexního řešení a dále je možné tímto způsobem zmírnit riziko pro „vendor lock-in".

Dalším klíčovým aspektem je také podpora metadat. ETL nástroje jsou také zodpovědné za využití metadat pro mapování zdrojových objektů na cílová schémata. Proto je velmi důležitý výběr nástroje, který odpovídá metadatové strategii organizace.

4.1.4 Výkon

Důležitá je schopnost nástroje škálovat podle toho, jak rostou nároky na výpočetní zdroje. Systémy IoT mohou generovat velké objemy dat a jejichž množství může rychle růst s tím, jak se do sítě přidávají další zařízení. Vybraný nástroj ETL pro IoT musí být škálovatelný a dostatečně flexibilní, aby zvládl měnící se potřeby systému IoT v čase.

Škálovatelnost lze zlepšit pomocí různých přístupů a technik, jako je rozdělení dat do oddílů (partitioning) a současné využití paralelismu nebo federace dat. Upořádání dat do oddílů na menší, lépe organizované části pomáhá rozdělit pracovní zátěž ETL mezi více výpočetních jednotek, jako jsou CPU, GPU nebo cloudové instance, a zvýšit tak výkon a propustnost procesu ETL. Federace dat může využívat distribuované souborové systémy, databáze nebo datová jezera k ukládání a zpracování dat škálovatelným způsobem.

Podle některých odhadů se očekává, že počet IoT zařízení vzroste z přibližně 20 miliard v roce 2019 na více než 75 miliard v roce 2025, což představuje roční míru růstu 30 %. To podtrhuje důležitost výběru škálovatelného, vysoce výkonného nástroje ETL, který může růst spolu s uživatelem.

4.1.5 Použitelnost

Použitelnost je jedním z důležitých faktorů každého nástroje. Proto je třeba vzít v úvahu, že nástroj by měl být snadno použitelný, srozumitelný a uživatelé by měli být schopni si na práci s daným nástrojem rychle zvyknout. V tomto ohledu je důležité, aby nástroj měl dobře navržené rozhraní. Různorodý tým uživatelů, může mít různé jazykové

preference. Podpora více jazyků může organizacím také usnadnit používání nástroje. Nástroje by měly být schopny rozdělit proces na malé stavební bloky a umožnit tak uživateli vytvořit uživatelsky definované funkce využitelné v různých místech datového toku.

4.2 Vytvoření modelu AHP

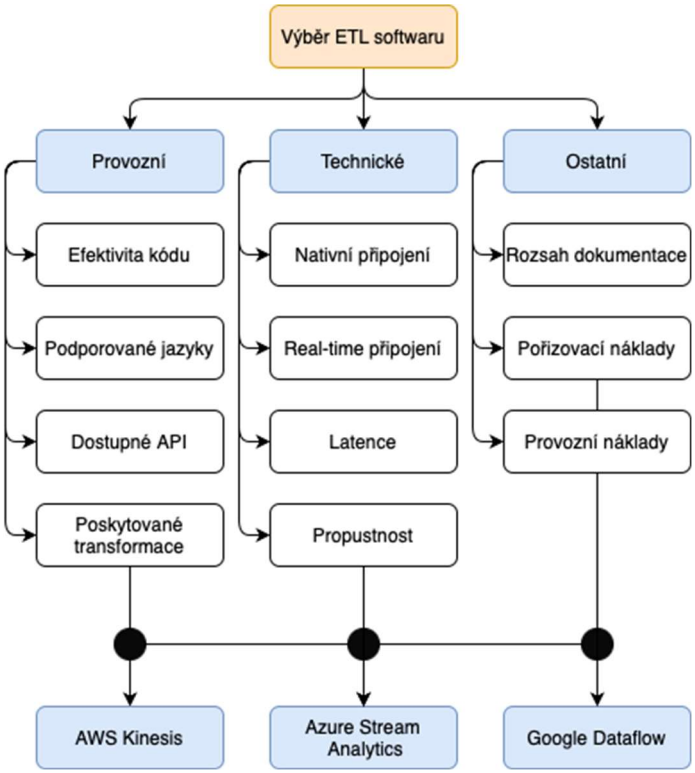
4.2.1 Formulace cíle

Cílem modelu AHP je na základě stanovených kritérií vybrat optimální ETL nástroj pro výše popsaný případ užití.

4.2.2 Vybraná kritéria a jejich váhy

Pro určení souboru kritérií byla zvolena čtyř úrovňová hierarchie (obr. 6). Poté byla použita metoda shora dolů. Bylo vytvořeno pět základních skupin kritérií, které byly následně doplněny dílčími kritérii. Kritéria byla zvolena v závislosti na dáném případu užití s využitím poznatků získaných v teoretické části práce.

Obrázek 6 Výběr ETL software



Zdroj: vlastní práce

I. Provozní kritéria

- Efektivita kódu – počet potřebných znaků v kódu pro základní transformace jako čištění, spojování, normalizaci, agregaci a filtrování dat. Výsledek byl sestaven na základě vlastního testování. Pseudokód je uveden v Příloze A.
- Podporované jazyky – počet podporovaných dotazovacích či skriptovacích jazyků. Sestaven byl na základě rešerše dostupných oficiálních zdrojů poskytovatele.
- Dostupné API – počet dostupných funkcí skrze rozhraní REST API s možností navrhovat, publikovat a spravovat ETL nástroj. Sestavený byl na základě rešerše dostupných oficiálních zdrojů poskytovatele.
- Poskytované transformace – počet možných komplexních transformací dat pomocí rozhraní drag and drop, SQL či skriptovacích jazyků. Sestavený byl na základě rešerše dostupných oficiálních zdrojů poskytovatele.

II. Technická kritéria

- Nativní připojení – počet možných nativních připojení k databázovým systémům, které nástroj podporuje. Sestaveno bylo na základě rešerše dostupných oficiálních zdrojů poskytovatele.
- Real-time připojení – zahrnuje počet možných připojení k systémům typu „fronty zpráv“, které nástroj podporuje. Sestaveno bylo na základě rešerše dostupných oficiálních zdrojů poskytovatele.
- Latence – průměrná latence nástroje. Výsledek byl sestaven na základě vlastního testování a rešerše dostupných zdrojů. Testováno s různým objemem dat.
- Propustnost – průměrná propustnost nástroje. Výsledek sestavený na základě vlastního testování a rešerše dostupných zdrojů. Testováno se stejným počtem výpočetních zdrojů pro každý nástroj.

III. Ostatní kritéria

- Rozsah dokumentace – velikost obsahu dokumentace a šířka poskytovaných formátů. Sestaven byl na základě rešerše dostupných oficiálních zdrojů poskytovatele.

- Pořizovací náklady – náklady v dolarech na implementaci nástroje vzhledem k zamýšlenému případu užití. Sestaveny byly na základě rešerše dostupných oficiálních zdrojů poskytovatele.
- Provozní náklady – průměrné měsíční náklady v dolarech na provoz nástroje vzhledem k zamýšlenému případu užití. Sestaveny byly na základě rešerše dostupných oficiálních zdrojů poskytovatele.

Váhy byly vytvořeny s ohledem na potřeby architektury ETL v cloudovém prostředí za pomoci Saatyho stupnice (tab. 1). Detailní hodnotící tabulky (Příloha B). Jako nejdůležitější skupina byla vybrána provozní a technická skupina kritérií z důvodu, že se jedná o hlavní stavební kameny ETL řešení. Jako nejméně důležitá skupina kritérií z hlediska vah byla zvolena skupina ostatní, která byla použita pro doplnění rozhodování.

V provozní skupině bylo jako nejdůležitější určeno kritérium poskytované transformace, protože tato funkcionalita definuje základní použití nástroje. Mírně nižší hodnocení obdržely kritéria podporované jazyky a dostupné API díky jejich podobnému dopadu na využitelnost nástroje. Nejmenší váhou bylo ohodnoceno kritérium efektivita kódu. Rozdíly mezi váhami v této skupině nejsou příliš vysoké, všechna kritéria jsou pro výběr nástroje poměrně důležitá.

V technické skupině bylo preferováno kritérium real-time připojení. Toto kritérium je podstatné pro připojení k různým datovým proudům a dodává prostor pro rozšíření využití nástroje i v dalších případech užití. V rámci skupiny bylo pak mírněji hodnoceno kritérium nativní připojení. Výkonnostní metriky jako latence a propustnost byly v rámci skupiny ohodnoceny stejně nejnižší váhou. Rozdíly mezi těmito metrikami nejsou pro konečný výběr nástroje v zamýšleném případu užití příliš důležité.

Nakonec ve skupině ostatní byly shodně hodnoceny kritéria pořizovacích a provozních nákladů. Rozsah dokumentace byl v této skupině vyhodnocen jako nejméně důležité kritérium, z důvodu, že rozsah dokumentace není pro daný případ užití zásadně omezující.

Před výpočtem byla provedena kontrola konzistence, kde byla ověřena validita vytvořené matice. Výsledné normované váhy byly vypočteny za pomoci metody geometrického průměru.

Tabulka 1 Vypočtená váha kritérií

Skupina kritérií	Váha skupin kritérií	Název kritéria	Váha ve skupině	Váha normovaná
Provozní	0,4	Efektivita kódu	0,122	0,049
		Podporované jazyky	0,227	0,091
		Dostupné API	0,227	0,091
		Poskytované transformace	0,423	0,169
Technická	0,4	Nativní připojení	0,249	0,100
		Real-time připojení	0,558	0,223
		Latence	0,096	0,039
		Propustnost	0,096	0,039
Ostatní	0,2	Rozsah dokumentace	0,143	0,029
		Pořizovací náklady	0,429	0,086
		Provozní náklady	0,429	0,086

Zdroj: vlastní práce

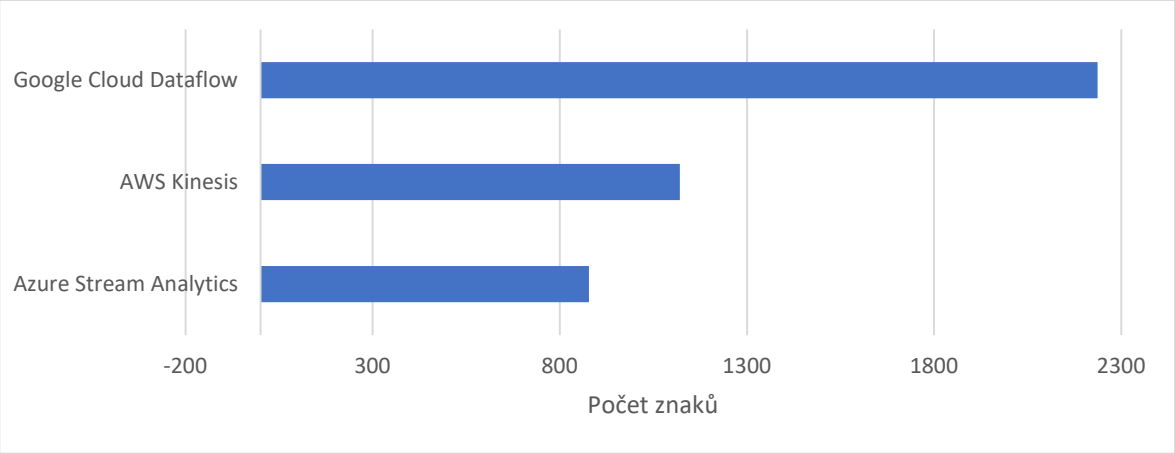
4.2.3 Vybrané alternativy a jejich ohodnocení

Jako alternativy byly do AHP modelu zahrnuty nástroje Azure Stream Analytics, AWS Kinesis a Google Dataflow. Všechny tři nástroje pracují v rámci cloudového prostředí a jsou integrovány do širší platformy poskytovatele. Nabízí potřebnou škálovatelnost a schopnost zpracovávat velké objemy dat v reálném čase. Každý nástroj poskytuje širokou škálu konektorů, které umožňují integraci s různými zdroji a proudy dat. Cenové politiky těchto platforem jsou založeny na využití výpočetních zdrojů a mohou být nákladově efektivnější než provoz vlastní infrastruktury. Každý nástroj podporuje více programovacích jazyků, což poskytuje flexibilitu z hlediska vývoje. Všechny nástroje nabízí řadu vestavěných transformací a algoritmů, které mohou usnadnit vytváření a nasazování potrubí ETL.

Efektivita kódu – z hlediska potřebných znaků pro vytvoření výsledného kódu se jeví nejvíce efektivní nástroje pro zpracování datových toků založené na jazyku SQL, jako jsou Azure Stream Analytics a AWS Kinesis Analytics. Nástroje umožňují stručnější kód, protože používají známou syntaxi SQL, ale mohou být omezenější z hlediska typů operací, které lze provádět. Nástroje, které využívají externí frameworky nebo programovací modely pro zpracování datových toků, jako jsou Apache Flink a Apache Beam, vyžadují pro

konstrukci kódu pro práci s datovým proudem více znaků. Je to dáno díky jejich flexibilitě, která je vykoupena potřebou explicitnější definice transformací dat. Jako nejméně efektivní byl označen Google Dataflow, který Apache Beam využívá a potřebuje více znaků pro zápis požadované funkce než ostatní dva nástroje (graf 2).

Graf 2 Počet potřebných znaků



Zdroj: vlastní práce

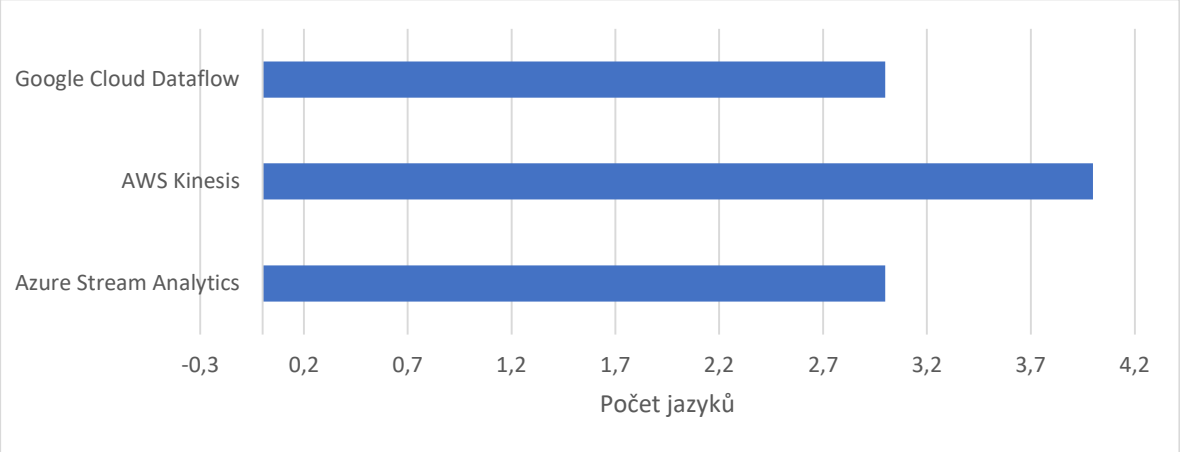
Podporované jazyky – AWS Kinesis nabízí dva různé způsoby zpracování streamovaných dat. Jednak aplikace založené na Kinesis SQL, které používají standardní syntaxi SQL k definování transformací a analýz nad datovým proudem. Díky tomu je snazší začít se zpracováním dat v reálném čase, pro méně technicky nadané uživatele. Naproti tomu lze využít integraci Kinesis s Apache Flink, kde je možné využít pokročilejší programovací model pro definování složité logiky zpracování dat nad proudovými daty. Flink také nabízí podporu pokročilejších funkcí, jako jsou stavové operace, vlastní operátory a modely strojového učení. Pro psaní vlastního kódu pak Flink podporuje několik programovacích jazyků (graf 3).

Azure Stream Analytics pro zpracování streamovaných dat v reálném čase využívá vlastní dotazovací jazyk podobný jazyku SQL Stream Analytics Query Language (SAQL). Jazyk poskytuje řadu funkcí a operátorů pro zpracování dat, podporuje vytváření oken, spojování, sjednocování a různé výstupní formáty a integruje se s dalšími službami Azure. Nástroj Stream Analytics dále také podporuje uživatelsky definované funkce v JavaScriptu nebo C#.

Služba Google Dataflow poskytuje podporu dotazovacích jazyků podobných jazyku SQL, ale nenabízí vestavěný dotazovací jazyk SQL jako ostatní dvě služby. Místo toho

Google Dataflow podporuje Apache Beam, což je jednotný programovací model, který podporuje více jazyků včetně jazyků Java, Python a Go. Ačkoli tedy Dataflow nenabízí vestavěný dotazovací jazyk SQL, poskytuje prostřednictvím Beam SQL rozhraní API podobné SQL, které vývojářům umožňují využít jejich znalosti jazyka SQL a psát dotazy nad streamovanými a dávkovými daty.

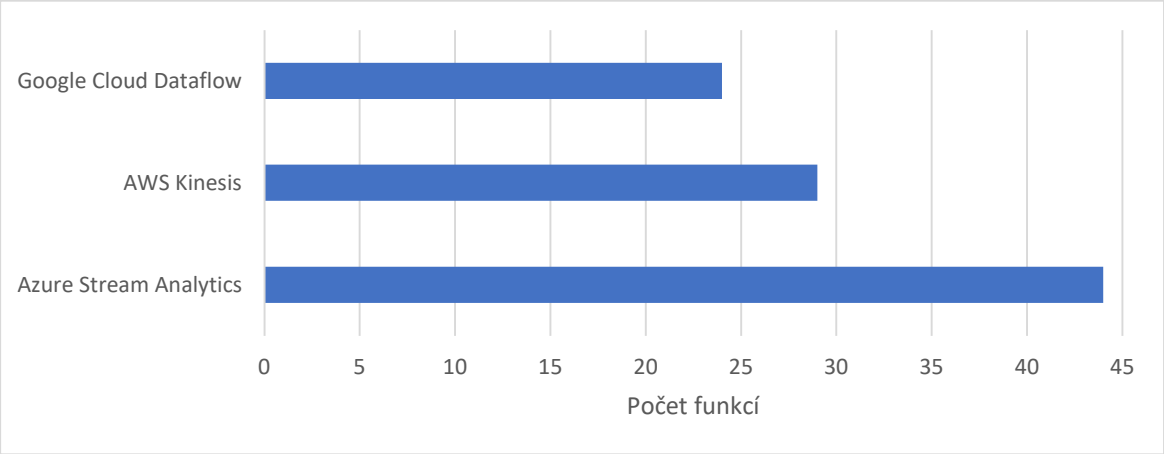
Graf 3 Dostupné programovací jazyky



Zdroj: vlastní práce

Dostupné API – Azure Stream Analytics poskytuje rozhraní REST API pro správu zdrojů Stream Analytics, včetně možnosti vytvářet, spouštět, zastavovat a mazat úlohy. Kromě toho jsou k dispozici SDK pro .NET, Javu, Node.js a Python, které poskytují abstrakci vyšší úrovně nad rozhraním REST API. Nástroj AWS Kinesis dává k dispozici sadu SDK pro několik programovacích jazyků včetně jazyků Java, .NET, Python a Node.js. Tyto SDK umožňují vývojářům vytvářet aplikace Kinesis a programově komunikovat se službou Kinesis. Kromě toho Kinesis poskytuje rozhraní REST API pro správu zdrojů Kinesis, včetně možnosti vytvářet, spouštět, zastavovat a odstraňovat datové toky. Google v nástroji Dataflow umožňuje za pomoci rozhraní REST API správu úloh a prostředků nástroje Dataflow, včetně možnosti vytvářet, spouštět, zastavovat a odstraňovat úlohy. Kromě toho jsou k dispozici klientské knihovny pro jazyky Java, Python a Go, které poskytují abstrakci vyšší úrovně nad rozhraním REST API. Každý dodavatel poskytuje podobnou sadu možností API, včetně rozhraní REST API i SDK specifických pro daný jazyk (graf 4).

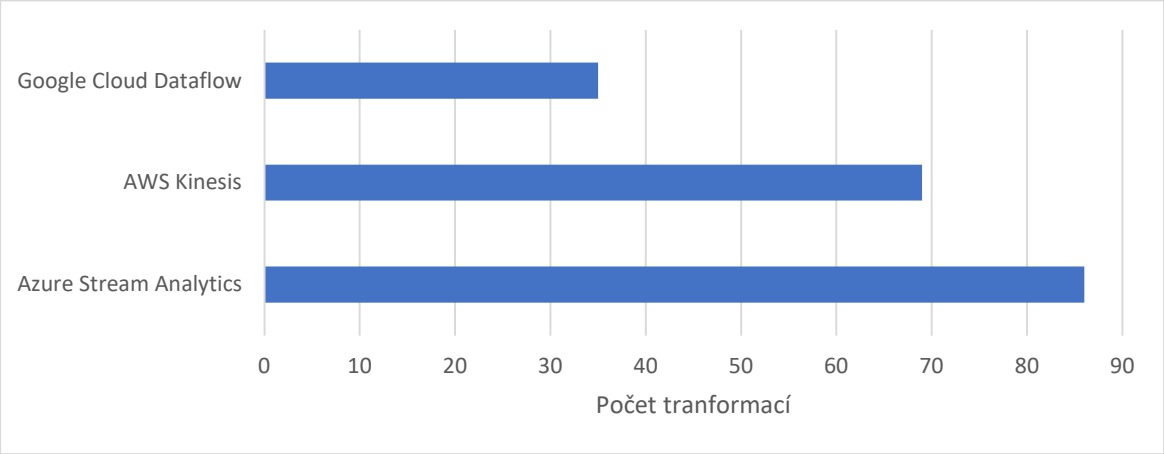
Graf 4 Dostupných funkce API



Zdroj: vlastní práce

Poskytované transformace – celkově lze říct, že všichni tři dodavatelé nabízejí rozsáhlou sadu vestavěných transformací a funkcí pro manipulaci s daty (graf 5), přičemž mezi nimi existují určité rozdíly. Azure Stream Analytics poskytuje jedinečnou sadu prostorových funkcí a funkcí strojového učení, zatímco AWS Kinesis podporuje vlastní transformace pomocí funkcí Lambda. Google Dataflow nabízí rozsáhlou sadu funkcí pro manipulaci s daty.

Graf 5 Dostupné build-in transformace



Zdroj: vlastní práce

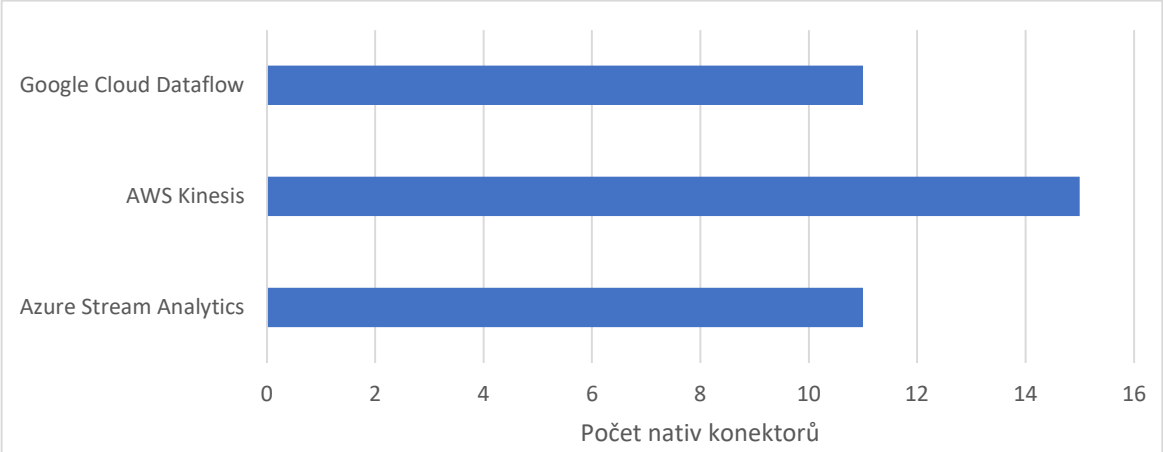
Nativní připojení, Real-time připojení – služby se zaměřují především na integraci s vlastními datovými zdroji a službami (graf 6, 7). Nicméně nabízejí konektory a API, které umožňují integraci s externími zdroji dat, včetně řešení třetích stran, jako je Apache Kafka.

Tyto konektory usnadňují používání těchto služeb pro ETL úlohy v reálném čase, a to jak pro zdroje dat na vlastních platformách, tak i pro ty, které jsou umístěny mimo platformy.

Azure Stream Analytics nabízí konektory pro integraci s externími zdroji převážně v rámci vlastní platformy, jako jsou například Azure Event Hubs, Azure Blob Storage a další. AWS Kinesis nabízí řešení i pro nástroje třetích stran, jako jsou Apache Kafka, Salesforce a další, a také umožňuje přístup k externím zdrojům pomocí AWS Lambda a AWS Fargate. Google Dataflow má konektory pro řešení třetích stran, jako jsou Apache Kafka a Cassandra, a také nabízí možnost integrace s externími zdroji dat pomocí Google Cloud Storage.

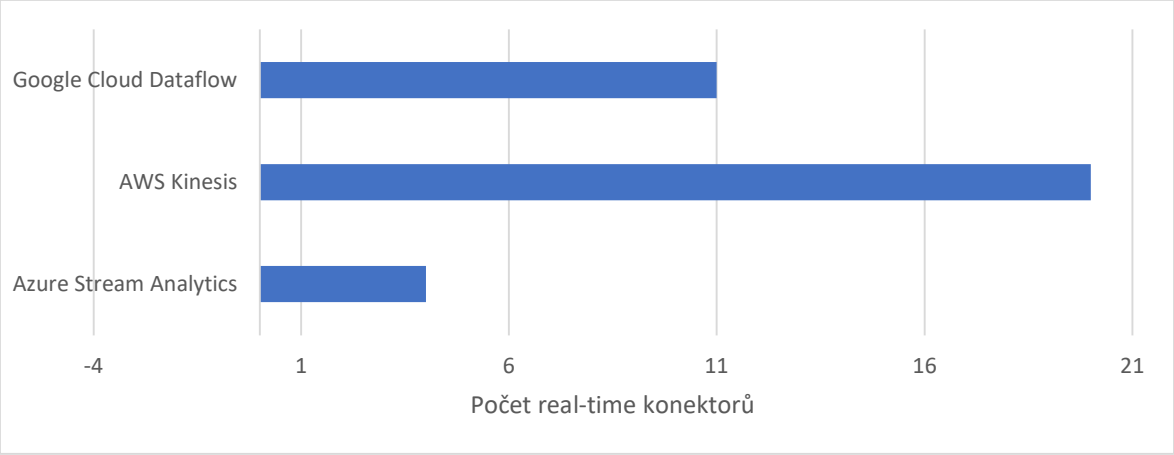
Vzhledem k tomu, že tyto služby se zaměřují především na své vlastní datové zdroje a služby, může být integrace s externími zdroji dat a službami složitá a časově náročná. Nicméně konektory a API těchto služeb usnadňují tuto integraci a umožňují využití těchto nástrojů pro úlohy ETL.

Graf 6 Dostupné konektory pro nativní připojení



Zdroj: vlastní práce

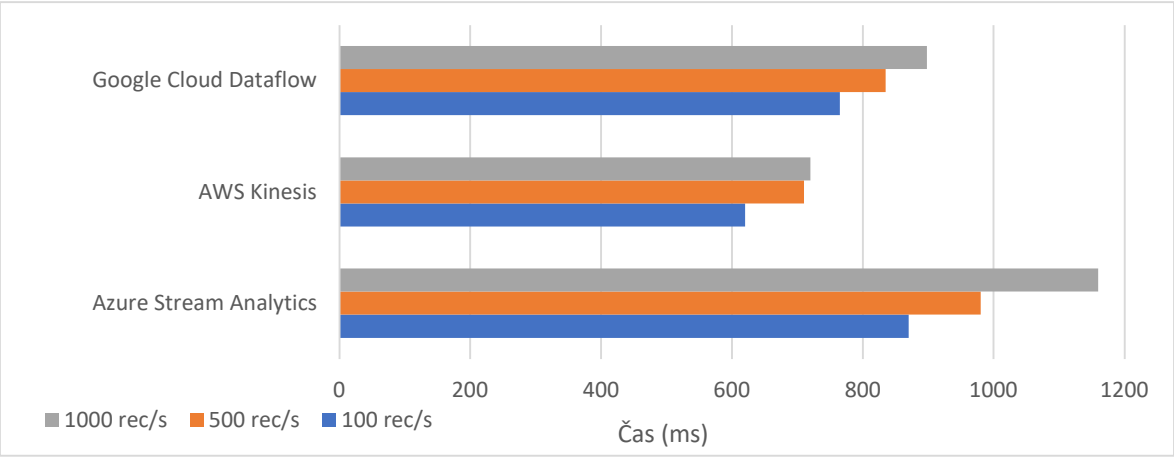
Graf 7 Dostupné konektory pro real-time připojení



Zdroj: vlastní práce

Latence – Testování bylo provedeno se zátěží 100, 1000 a 3000 záznamů za sekundu. Test spočíval jednoduché v transformaci dat pocházejícího z jediného datového proudu s velikostí jednoho záznamu 200 B. V rámci testování AWS Kinesis byl průchod měřen všemi zapojenými komponentami. Výsledky testování v milisekundách byly odečteny pomocí vestavěného monitoringu příslušného nástroje (graf 8).

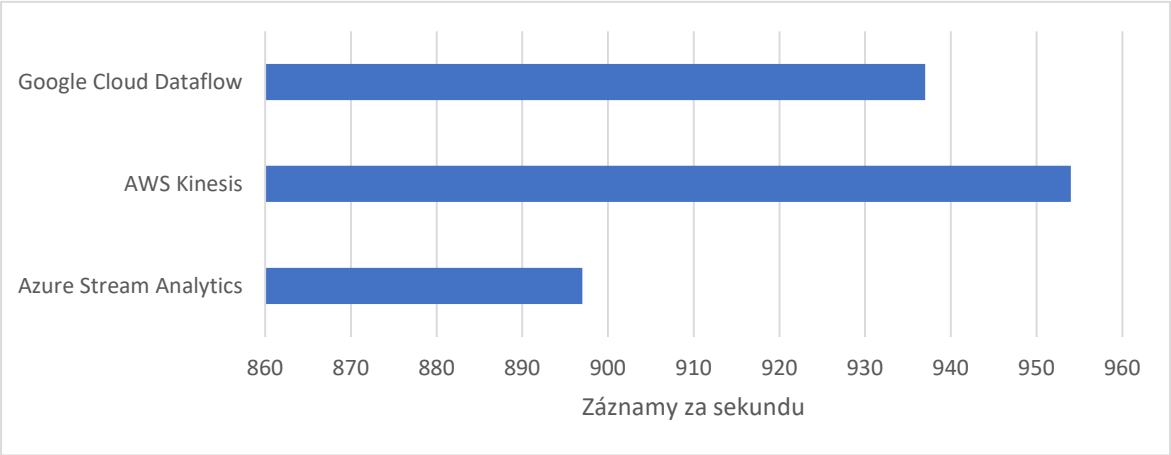
Graf 8 Průměrná latence nástroje



Zdroj: vlastní práce

Propustnost – V testu byla měřena maximální propustnost datových záznamů na jednu účtovatelnou jednotku nástroje. V případě nástroje Dataflow bylo počítáno s jedním pracovním uzlem s alokací 1 vCPU a 4 GB RAM. Výsledky byly odečteny pomocí vestavěného monitoringu příslušného nástroje (graf 9).

Graf 9 Průměrná propustnost nástroje



Zdroj: vlastní práce

Rozsah dokumentace – Služba Azure Stream Analytics v rámci dokumentace pokrývá širokou škálu témat souvisejících s používáním služby, včetně úvodních příruček, výukových materiálů a referenčních materiálů. Dokumentace je přehledná a snadno se v ní orientuje a obsahuje podrobná vysvětlení a příklady ke každé funkci. Rozsah dokumentace je však ve srovnání se službami Azure Stream Analytics a AWS Kinesis relativně menší.

AWS Kinesis poskytuje rozsáhlou dokumentaci, která pokrývá širokou škálu témat souvisejících se službou. Dokumentace je dobře organizovaná a snadno se v ní orientuje a obsahuje podrobná vysvětlení a příklady pro každou funkci. Může však být poněkud zahlcující vzhledem k obrovskému objemu dostupných informací.

Google Dataflow nabízí neméně obsáhlou knihovnu dokumentace, která pokrývá širokou škálu témat souvisejících se službou, včetně průvodců pro začátek, výukových programů a referenčních materiálů. Dokumentace je přehledná a snadno se v ní orientuje a obsahuje podrobná vysvětlení a příklady ke každé funkci.

Celkově ačkoli mohou existovat určité rozdíly v rozsahu dokumentace, všichni tři dodavatelé poskytují komplexní a dobře organizovanou dokumentaci, která uživatelům pomůže začít a co nejlépe využívat jejich služby.

Pořizovací náklady – všechny z nástrojů fungují na cenovém modelu „pay-as-you-go“, kde nejsou vyžadovány žádné počáteční náklady. Uživatelé tak platí pouze za zdroje, které jsou využívány během běhu úlohy. Uživatelé mohou úlohu kdykoli zastavit, aby se vyhnuli dalším poplatkům.

Za zmínku stojí, že každá služba nabízí bezplatnou úroveň, která uživatelům umožňuje experimentovat a testovat službu bez jakýchkoli poplatků. Bezplatná úroveň má obvykle omezení týkající se množství dat, která lze zpracovávat, počtu dostupných prostředků pro zpracování. Může být dobrým způsobem, jak se službou začít a posoudit její vhodnost pro konkrétní případ použití.

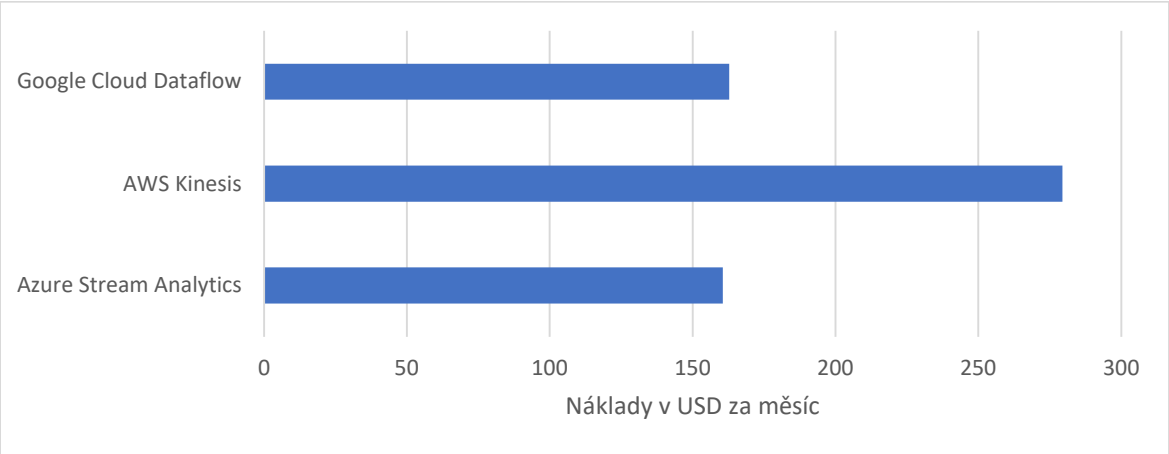
Provozní náklady – Azure Stream Analytics používá koncept Streaming Units (SU) k popisu úrovně prostředků pro zpracování přidělených úloze Stream Analytics. Jednotky SU představují kombinaci kapacity CPU, paměti a I/O a používají se k určení ceny úloh služby Stream Analytics. Podle dokumentace Azure Stream Analytics může jedna jednotka SU zpracovat až 1 MB/s vstupních dat nebo až 1 000 vstupních událostí za sekundu.

AWS Kinesis využívá podobný koncept Kinesis Processing Units (KPU), který zde znázorňují základní jednotku škálovatelnosti pro zpracování dat. KPU je pevná jednotka výpočetní kapacity, která dokáže zpracovat určité množství datového proudu. Každý KPU může zpracovat až 1 MB/s vstupních dat a až 2 MB/s výstupních dat nebo až 1000 záznamů za sekundu, podle toho, které hranice je dosaženo dříve.

Google Dataflow nabízí jiný přístup k alokování výpočetních zdrojů. Umožňuje zvolit počet tzv. Worker Units, které reprezentují počet virtuálních strojů přidělené úloze Dataflow. Dále umožňuje nastavit počet přidělených výpočetních zdrojů pro dané „workery“ podle požadavků zákazníka. Množství výpočetních prostředků přidělených úloze je tedy určeno počtem jednotek Worker Units x počet přidělených výpočetních zdrojů. Podle dokumentace Google Cloud může typický „worker“ v Dataflow zpracovat až 1 000 vstupních záznamů za sekundu na jedno jádro.

Všichni poskytovatelé umožňují použít vlastní nástroj pro výpočet odhadu provozních nákladů. Při jejich tvorbě bylo uvažováno s datovým tokem 1 600 záznamů za sekundu o velikosti 200 B s celkovým datovým objemem 801,99 GiB za měsíc. Pro nástroje Stream Analytics a Kinesis byla pro daný datový tok uvažována spotřeba 2 SU, respektive 2 KPU. Dataflow byl uvažován 2x worker node s 1vCPU a 4 GB RAM. Náklady se mohou v jednotlivých regionech lišit, proto byl vybrán u všech nástrojů shodně region Franfurt. Cenově nejpříznivější z hlediska měsíčních nákladů se jeví nástroj Azure Stream Analytics a Google Dataflow (graf 10). Je vhodné zmínit, že se jedná o ceníkové ceny a všichni z dodavatelů poskytují na své produkty individuální slevy při využití více nástrojů v platformě.

Graf 10 Měsíční provozní náklady



Zdroj: vlastní práce

4.3 Praktické otestování vybraných řešení

4.3.1 Model dat

V sekci modelu zdrojových dat je důležité, jak zdrojová data (tab. 2), přijímaná ze zařízení, slouží jako základ pro celý proces transformace dat. Ukázka raw dat v JSON formátu poskytuje lepší představu o struktuře a obsahu těchto informací. V ukázce surových dat je patrné, jak jednotlivé prvky, například atributy a hodnoty, jsou uspořádány a jak mohou být později využity pro další zpracování. Model cílové databáze (tab. 3) představuje konečnou podobu dat, připravenou pro efektivní uložení a snadný přístup.

Tabulka 2 Přehled zdrojových objektů

Entita	**MQTT Topic**	**Poznámka**
planter	planter/location	Geolokace stroje
planter	planter/crop	Kategorie dané plodiny
planter	planter/speed	Rychlost stroje
unit	unit/<id>/planting_depth	Hloubka setí dané jednotky
unit	unit/<id>/soil_moisture	Vlhkost půdy
unit	unit/<id>/seed_count	Počet spotřebovaných semen
unit	unit/<id>/air_pressure	Tlak vzduchu v jednotce
unit	unit/<id>/seed_pressure	Tlak v dopravníku osiva
unit	unit/<id>/fertilizer_pressure	Tlak v zásobníku hnojiva
meteo	meteo/moisture	Vlhkost vzduchu
meteo	meteo/temperature	Teplota vzduchu

Zdroj: vlastní práce

Ukázka surových dat ve formátu JSON:

```
{
  "id":unit_001,
  "name":"Unit Depth Sensor",
  "payload":{
    "data":{
      "metric":"Depth",
      "value":23,
    }
  }
}
```

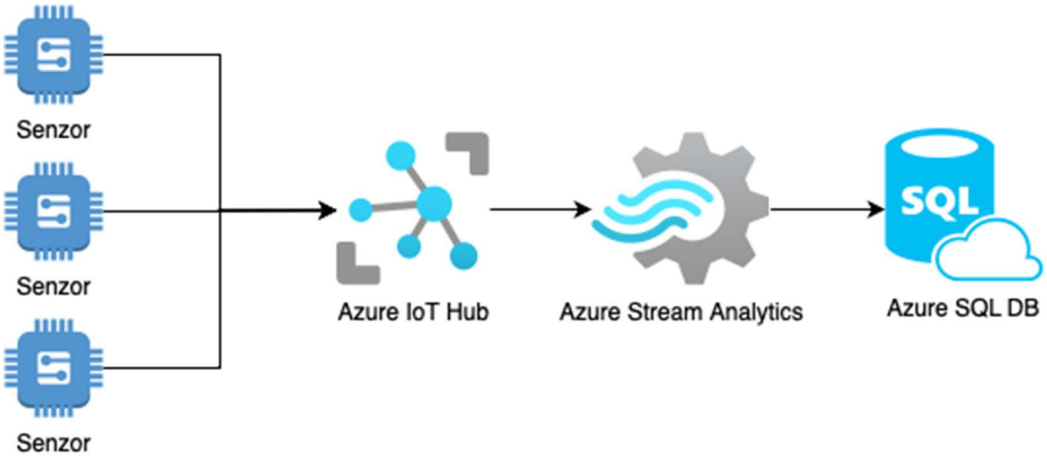
Tabulka 3 Model datového úložiště

Atribut	**Datový typ**	**Popis**
unit_id	Integer	Identifikátor jednotky
planter_id	Integer	Identifikátor secího stroje
crop_id	Integer	Kategorie osiva
max_light_intensity	Float	Maximální intezita osvětlení
avg_planter_speed	Float	Průmerná rychlost secího stroje
avg_planting_depth	Float	Průměrná hloubka setí
sum_seed_count	Integer	Suma použitého osiva
max_fertilizer_pressure	Float	Maximální tlak v zásobníku hnojiva
max_seed_pressure	Float	Maximální tlak v systému setí
avg_air_temperature	Float	Průměrná teplota okolního vzduchu
latitude	String	Zeměpisná šířka
longitude	String	Zeměpisná délka

Zdroj: vlastní práce

4.3.2 Popis implementace A

Obrázek 7 Architektura implementace A



Zdroj: vlastní práce

Samotné zpracování datového proudu v Azure Stream Analytics se odehrává ve spojení s nástroji Azure IoT Hub a Azure SQL Database (obr. 7). Senzory a zařízení jsou připojeny k platformě Azure IoT Hub, který umožňuje přenos dat v reálném čase a sledování jejich výkonu. Data jsou v IoT Hub ukládána v tzv. „device twin“, což je virtuální zastoupení každého připojeného zařízení. Azure Stream Analytics pak umožňuje zpracování datového proudu pomocí standardního SQL jazyka a s jeho pomocí můžeme filtrovat, agregovat, spojovat a transformovat data z IoT Hub. V případě potřeby složitějších konstrukcí je možné pomocí zvláštních klíčových slov, jako jsou CROSS APPLY nebo OUTER APPLY využít User Defined Functions (UDF), které umožňují přizpůsobit zpracování datového proudu. UDF lze vytvořit v jazyce JavaScript nebo C#.

Abychom mohli začít se zpracováním dat, musíme nejprve připojit vstupní proud dat k Azure Stream Analytics. Vstupní proud dat může být jakýkoliv proud dat vytvořený pomocí služeb Azure. V rámci konfigurace vstupního proudu dat musíme specifikovat formát dat, který vstupní proud využívá. Dále můžeme konfigurovat zpoždění proudů dat, počet paralelních instancí, velikost bufferu atd.

Jakmile jsme úspěšně připojili vstupní proud dat k Azure Stream Analytics, můžeme přistoupit k samotnému zpracování dat. Zpracování datového proudu je v Azure Stream Analytics realizováno pomocí tzv. „jobs“. Každý job obsahuje definici vstupního a výstupního proudu dat, stejně jako dotaz napsaný v jazyce SQL, který řídí zpracování dat.

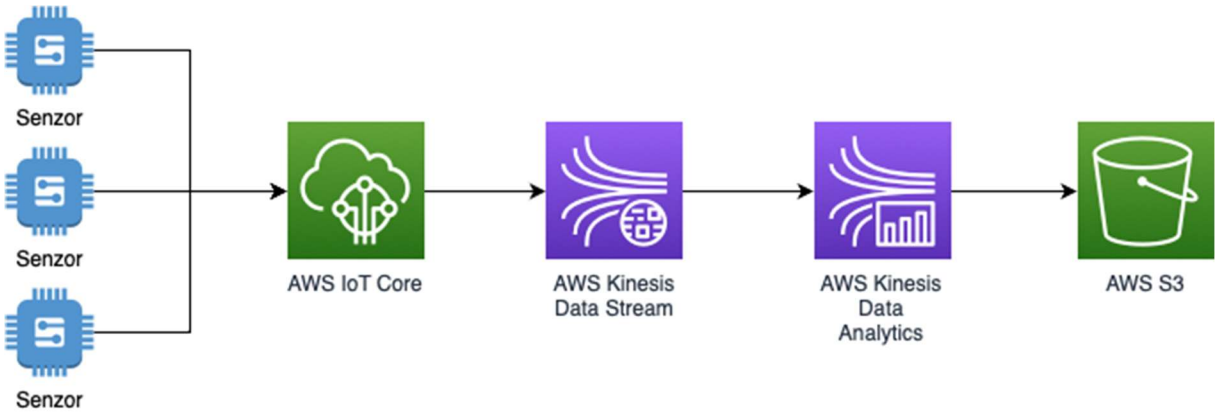
Výstupní proud dat může být směrován na řadu cílů, včetně Azure SQL Database a Azure Blob Storage. Směrování dat do cílové datového úložiště probíhá pomocí tzv. „output adapterů“. Tyto adaptéry představují definice, které specifikují cílové místo, kam mají být data odeslána. Výstupní proud dat můžeme dále filtrovat, upravovat a obohacovat pomocí dalších operací v rámci samotného jobu. Výsledný kód pro „job“ obsahující logiku zpracování datového proudu dle výše uvedeného pseudokódu je k vidění v Příloze F.

V rámci Azure Stream Analytics lze nastavit různá pravidla a upozornění, které se spustí v případě výskytu určitých událostí v datovém proudu. Lze například vytvořit pravidlo, které upozorní, pokud se v datovém proudu vyskytne určitá kombinace hodnot nebo pokud přichází data v neobvyklém časovém rozmezí.

Závěrem lze říci, že Azure Stream Analytics poskytuje robustní nástroj pro zpracování a analýzu streamovaných dat z IoT Hub. Pomocí jazyka SQL a vlastních funkcí je možné rychle a efektivně zpracovat velké množství dat a vytvořit užitečné výstupy pro další aplikace. Díky integraci s Azure IoT Hub a Azure SQL Database lze snadno propojit celou data pipeline od sběru dat přes zpracování až po ukládání výsledků do relačních databází. Azure Stream Analytics tak představuje zajímavou alternativu k nástroji AWS Kinesis.

4.3.3 Popis implementace B

Obrázek 8 Architektura implementace B



Zdroj: vlastní práce

Zařízení a senzory jsou připojeny k AWS cloud platformě pomocí služby AWS IoT Core, která umožňuje přenos dat v reálném čase a sledování jejich výkonu. Služba IoT Core přeposílá všechny příchozí zprávy ve formátu MQTT od jednotlivých zařízení dále do nástroje AWS Kinesis k dalšímu zpracování (obr. 8). Samotný AWS Kinesis se skládá ze tří

základních komponent: Data Stream, Data Analytics a Firehose. Data Stream automaticky zajišťuje potřebnou infrastrukturu pro zpracování datového proudu a umožňuje směrování datového proudu do dalších aplikací v rámci nástroje Kinesis. Hlavní logika zpracování dat je poté provedena pomocí vytvořené aplikace v Data Analytics. Komponenta také umožnuje zpracovaná data vystavit do úložiště Amazon S3. Poslední součást Kinesis Firehose je vhodná v případě potřeby využití složitějšího datového úložiště nebo úložiště poskytované pomocí třetích stran pro výstup datového proudu. Díky této architektuře lze snadno spravovat velké množství dat a poskytnout rychlí přístup k datům.

Práce se samotným nástrojem AWS Kinesis probíhá z webového grafického rozhraní – AWS Console, stejně jako pro jiné nástroje z platformy AWS. AWS dále podporuje správu cloudových nástrojů skrze CLI nebo SDK. Jako první pro zpracování datového proudu je nutné vytvořit nový „data stream“ v komponentě Data Stream. Jeho vytvoření spočívá v jednoduchém nastavení požadovaného jména datového proudu pro pozdější identifikaci v rámci AWS a zvolení požadovaného módu kapacity. Kapacitní režim určuje způsob, jakým jsou poskytovány výpočetní zdroje a s tím spojené účtování poplatků. V rámci aplikace Data Streams může být vybrán jeden z režimů kapacity, a to režim výpočetních zdrojů tzv. „na vyžádání“ nebo režim Provisioned, kde je kapacita čtení a zápisu omezena na určitou pevnou hodnotu. Vzhledem k tomu, že služba Azure Stream Analytics umožňuje pouze druhý způsob alokování výpočetních zdrojů, byl pro srovnání vybrán mód Provisioned.

Po vytvoření „data stream“ je nutné vytvořit ve službě IoT Core pravidlo vynucující přeposílání přijatých zpráv do Kinesis. IoT Core umožňuje základní filtrování zpráv vhodných pro přenesení za pomoci vlastního SQL jazyka. Po nastavení je možné se přesvědčit o správnosti konfigurace pomocí vestavěného monitoringu v nástroji Data Stream.

Dalším krokem je vytvoření samotné logiky pro zpracování datového proudu. Děje se tak v komponentě Data Analytics. Komponenta umožňuje vytvoření aplikace na bázi Apache Flink nebo tzv. SQL aplikaci, kde je možné zpracovávat a analyzovat proudová data pomocí standardního jazyka SQL. Metoda pomocí SQL je i přesto, že je stále dostupná, označena jako zastaralá, navíc podporuje pouze jeden vstupní datový proud, proto byl vybrán typ aplikace na základě Apache Flink. Pro samotnou tvorbu kódu aplikace je možné využít nástroj Apache Zeppelin, který zpřístupní webové IDE napojené na AWS

infrastrukturu. Další možností je lokální vývoj a pozdější nahrání vytvořené aplikace pomocí CLI nebo webového úložiště Amazon S3. Z důvodu preference vlastního IDE byl vybrán postup pomocí nahrání zdrojového kódu z lokálního prostředí.

Samotné vytvoření nové aplikace v Data Analytics je stejně přímočaré jako v případě předchozí komponenty. Zapotřebí je zvolit jméno vytvářené aplikace a použitou verzi Apache Flink. Po vytvoření základní kostry je možné přejít k vývoji samotného kódu pracujícího s Apache Flink. Výsledný kód obsahující logiku zpracování datového proudu a uložení zpracovaných dat (Příloha G), byl po zapouzdření v .jar souboru nahrán do úložiště S3.

Poslední krok při přípravě aplikace v nástroji Data Analytics spočívá v úpravě konfigurace aplikace. V této fázi se spáruje zdrojový kód z úložiště s danou aplikací a nastaví se důležité parametry ovlivňující chování aplikace, především proměnné běhového prostředí sloužící pro výběr konkrétního požadovaného datového proudu vstupujícího do aplikace a také cílové objekty pro zápis dat. Vedle toho je také možné konfigurovat parametry jako například škálování úlohy nebo paralelizace úloh v rámci KPU. Škálování úlohy umožňuje přidělení více KPU, což zvyšuje výpočetní výkon aplikace a urychluje její běh. Paralelizace úloh v rámci KPU zase zajišťuje efektivnější využití výpočetních zdrojů jednoho KPU.

Nástroj Data Analytics mimo výše uvedené umožňuje nastavit vhodné logování a monitoring úlohy, což umožňuje sledovat stav aplikace během jejího běhu a identifikovat případné problémy. Kromě toho, nástroj Data Analytics poskytuje možnost zabezpečit odstávky aplikace, což je užitečné při výpadku KPU nebo jiného problému v prostředí, který by mohl ovlivnit běh aplikace. Tímto způsobem se aplikace dokáže obnovit a pokračovat v běhu po odstranění problému. Správná konfigurace aplikace v nástroji Data Analytics je klíčová pro úspěšný běh aplikace a efektivní využití výpočetních zdrojů v rámci služby AWS Kinesis.

4.3.4 Performance testy

Pro vytvoření výkonnostního testu byl zvolen nástroj Apache JMeter, který umožňuje provádět různé druhy výkonových a zátěžových testů na webových aplikacích, databázích, aplikačních serverech a dalších systémech. V kontextu testování AWS Kinesis a Azure Stream Analytics lze JMeter použít pro vytváření zátěžových testů a simulaci reálných podmínek pro zpracování datových proudů v obou platformách. Tento nástroj umožňuje

vytvářet scénáře a testovací sady, které umožní měřit výkon a spolehlivost obou nástrojů a poskytnout tak cenné informace pro optimalizaci a vylepšení datových procesů.

Apache JMeter byl zvolen z důvodu jednoduchosti použití, rozšiřitelnosti a podpory různých druhů protokolů, včetně MQTT a dalších. Další velkou výhodou je možnost monitorovat výsledné hodnoty testů v reálném čase pomocí grafů a tabulek.

Pro implementaci testování v JMeteru byl nejprve vytvořen testovací plán s komponentami pro simulaci zátěže Thread Group, MQTT Connect a MQTT Pub Sampler. V konfiguraci Thread Groupu byl nastaven počet a druh simulovaných zařízení dle popsaného případu užití a modelu zdrojových dat podkapitola (4.3.1). Čas trvání testu byl nastaven na dobu 10 minut. V komponentě MQTT Connect bylo třeba vytvořit spojení s danou platformou na základě předem vytvořených certifikátů a přístupových klíčů. Následovala konfigurace MQTT Pub Sampleru pro každé simulované zařízení. Sampler umožňuje odesílat zprávy v MQTT formátu do platformy AWS IoT Core nebo Azure IoT Hub. V komponentě bylo nezbytné nastavit vlastní MQTT zprávu, která obsahuje generovaná testovací data a směrování do příslušného MQTT topicu. V poslední řade bylo třeba nastavit další parametry, jako je QoS úroveň.

V dalších komponentách, jako jsou Assertion a View Results Tree, bylo nastaveno ověřování chování nástroje Jmeter. Pro ověření doby odezvy byla vytvořena další aplikace v rámci platformy, která simuluje přístup k datům od odběratele. Tento test byl zaveden s cílem zkontrolovat, jak rychle se systém dokáže přizpůsobit novým datům. Aplikace měla za úkol sledovat stav nových souborů v cílovém úložišti a v případě doručení nových dat, tyto data vyzvednout a porovnat čas jejich vytvoření ve zdrojovém zařízení s aktuálním časem. V poslední řadě byly na základě výstupu vyhotoveny grafy zobrazující chování systému z hlediska doby odezvy a spotřeby výpočetních zdrojů.

5 Výsledky a diskuse

5.1 Vyhodnocení AHP

Na základě hodnoty indexu konzistence CI menší než 0,1 byla rozhodovací matice označena jako dostatečně konzistentní. Avšak, je důležité poznamenat, že AHP by mělo být použito jako nástroj pro podporu rozhodování, nikoli jako konečné řešení. Efektivita AHP závisí na kvalitě dat a na tom, jak dobře hodnotící osoba stanoví preference a váhy. AHP může být úspěšně použito pro řešení mnoha problémů, ale je třeba si uvědomit, že jeho výstupy jsou závislé na vstupních datech a na způsobu jejich interpretace. Dle vytvořeného modelu se jako nejvhodnější nástroje pro implementaci jeví AWS Kinesis a Azure Stream Analytics (tab. 4).

Tabulka 4 Výsledné body AHP

Kritérium	**Váha**	**Sub-kritérium**	**Alternativa**					
			ASA	**AWSK**	**GCD**	**ASA**	**AWSK**	**GCD**
			vi			**norm.**		
Provozní	0,4	Efektivita kódu	0,637	0,258	0,105	0,255	0,103	0,042
		Podporované jazyky	0,500	0,250	0,250	0,200	0,100	0,100
		Dostupné API	0,600	0,200	0,200	0,240	0,080	0,080
		Poskytované transformace	0,637	0,258	0,105	0,255	0,103	0,042
Technická	0,4	Nativní připojení	0,200	0,600	0,200	0,080	0,240	0,080
		Real-time připojení	0,088	0,669	0,243	0,035	0,268	0,097
		Latence	0,135	0,822	0,333	0,054	0,329	0,133
		Propustnost	0,333	0,333	0,333	0,133	0,133	0,133
Ostatní	0,2	Rozsah dokumentace	0,200	0,400	0,400	0,040	0,080	0,080
		Pořizovací náklady	0,333	0,333	0,333	0,067	0,067	0,067
		Provozní náklady	0,455	0,091	0,455	0,091	0,018	0,091
					Suma	1,450	1,521	0,945

Zdroj: vlastní práce

5.2 Vyhodnocení implementace

Oba nástroje, Azure Stream Analystic a Amazon Kinesis, poskytují uživatelům mnoho možností, jak implementovat a spravovat své aplikace. Uživatelé mohou volit mezi různými způsoby práce, ať už preferují čistě příkazovou řádku nebo komplexní webové rozhraní.

Nicméně, prostředí poskytované Amazonem může na první pohled působit méně přehledně, zejména kvůli velkému množství různých nástrojů, které Amazon nabízí. Samotný nástroj Kinesis je navíc rozdělen do několika různých komponent, což může být matoucí a ztížit správné využívání nástroje. Uživatelé si nemohou být zcela jistí, jak optimálně využít jednotlivé komponenty a jak je vzájemně propojit, aby dosáhli těch nejlepších výsledků.

Je třeba poznamenat, že funkcionalita v jednotlivých komponentách se částečně překrývá, což může dále komplikovat uživatelské rozhodování o tom, jak nejlépe využít nástroj Kinesis. Přesto se jedná o velmi výkonný nástroj, který umožňuje uživatelům zpracovávat a analyzovat velká množství dat v reálném čase.

V této práci bylo zkoumáno zpracování dat v reálném čase v cloudu, s důrazem na watermarky. Bylo zjištěno, že Azure nenabízí takovou funkčnost jako Kinesis, který umožňuje vkládání watermarků přímo do datového proudu. Navíc je v Azure omezena možnost nastavení time processingu pouze na vstupu do Azure Stream Analytics (ASA), zatímco v Kinesis lze změnit druh time processingu přímo v kódu.

V Azure není možné měnit partitování datového proudu z IoT hubu, pouze zvolit počáteční nastavení počtu partition. Byl zaznamenán případ, kdy byly dvě zařízení přiřazena do stejné partition, přestože byla k dispozici volná partition. Takovéto chování může mít dopad na výkon celého systému díky „sešikmení“ rozložení zátěže na výpočetní uzly.

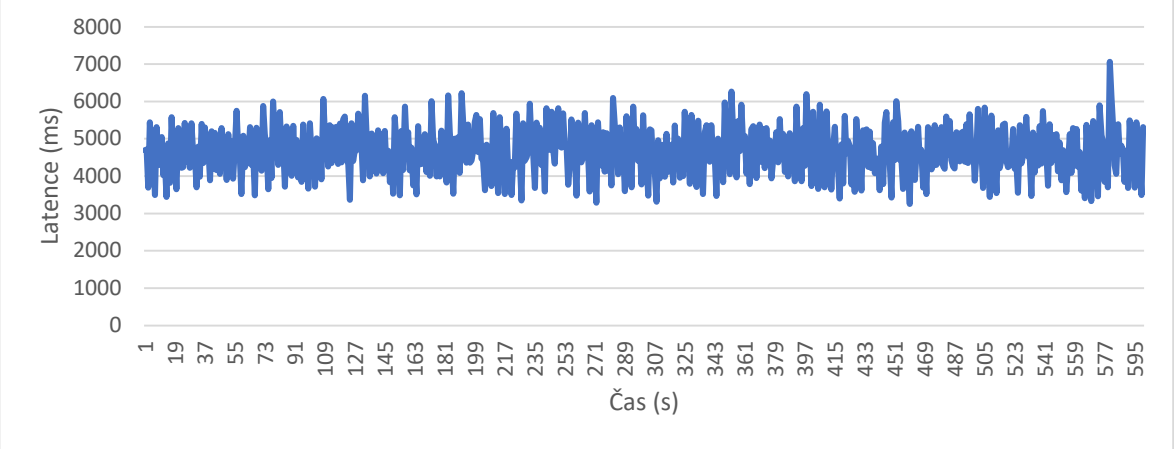
Z hlediska debbugingu je Azure výhodnější, díky monitorovacím a logovacím službám s menší latencí a přehlednějším grafům v ASA. Monitorovací systém na platformě Amazon trpí značným zpožděním v zobrazení metrik a grafů a vývojář je často nucen čekat.

Celkově lze říci, že Azure je vhodnější pro jednodušší projekty, zatímco Kinesis může být výhodnější pro složitější projekty, díky větší funkčnosti a přehlednosti. SQL jazyk je vhodný pro jednoduchou datovou analýzu, zatímco Flink, který používá jazyk Java, vyžaduje pokročilé znalosti programování.

5.3 Vyhodnocení testování

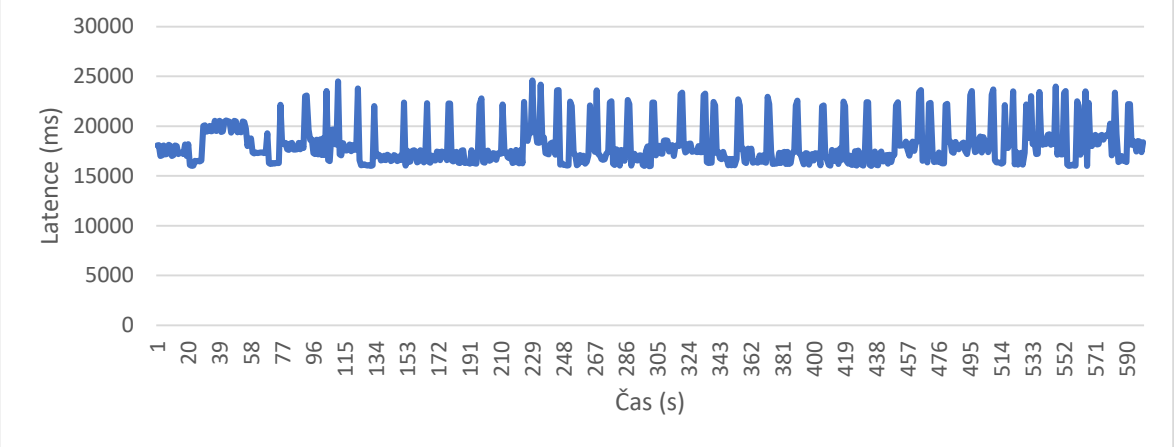
Během testování bylo zjištěno, že obě implementace, Kinesis i Azure, dokázaly úspěšně zpracovat datový tok o rychlosti 975 IoT záznamů za sekundu po dobu 10 minut. Avšak, z analýzy dat vyplynulo, že průměrná hodnota odezvy v Kinesis byla 4652,3 ms, se směrodatnou odchylkou 653,7 ms (graf 11), zatímco Azure vykazoval průměr 18062 ms se směrodatnou odchylkou 2035,5 ms (graf 12). Z průběhu latence v nástroji ASA lze vyvodit, že nástroj data do cílového úložiště ukládá data po menších dávkách nebo cílové úložiště umožňuje přístup k datům až od určité dávky, proto vidíme v průběhu výrazné periodické špičky, kterými druhý nástroj netrpí.

Graf 11 Výsledek nástroje AWS Kinesis



Zdroj: vlastní práce

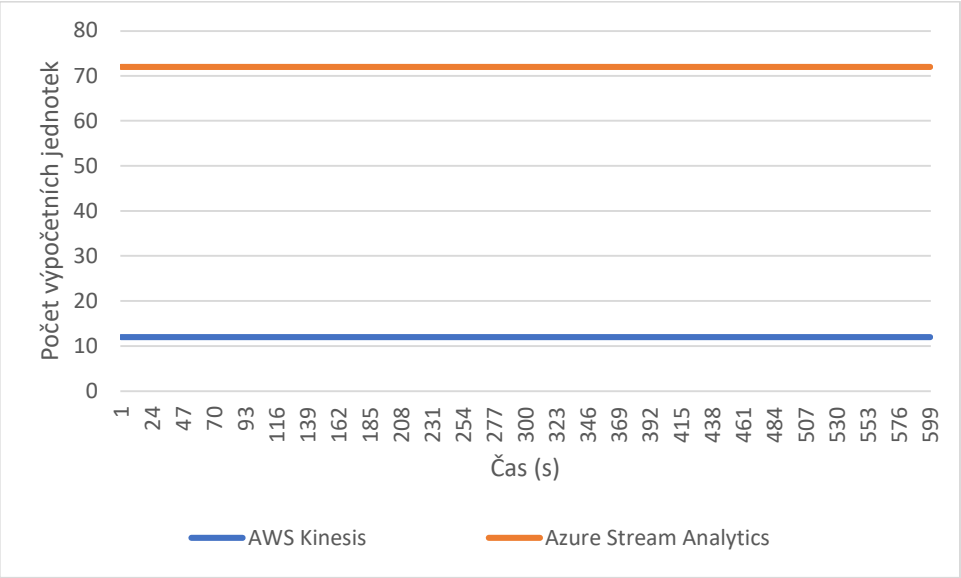
Graf 12 Výsledek nástroje Azure Stream Analytics



Zdroj: vlastní práce

Nástroj Azure navíc vyžadoval větší paralelizaci úlohy pro eliminaci problémového „backpressure“. Tento jev nastává v případě, kdy se do nástroje dostává vyšší množství záznamů, než je schopen zpracovat. Škálování výpočetních jednotek bylo zvoleno dynamické zvolené nástrojem, avšak po celou dobu trvání testu zůstával počet výpočetních jednotek u obou poskytovatelů konstantní (graf 13).

Graf 13 Počet využitých výpočetních jednotek v testu



Zdroj: vlastní práce

Z těchto výsledků lze vyvodit, že oba nástroje jsou schopny zpracovat datové toky v reálném čase, avšak Kinesis má znatelně nižší latenci, což může být výhodné pro náročnější projekty, které vyžadují rychlou reakci na příchozí data. Na druhé straně, pro jednodušší projekty s menším datovým tokem, může být Azure vhodnější díky nižší ceně a jednoduššímu použití.

Celkově lze tedy říci, že oba nástroje se ukázaly jako funkční a spolehlivé, i když s rozdílnou úrovní latence.

6 Závěr

Vhodný nástroj ETL a vzor architektury umožňují efektivní rozvoj použití dat v zemědělství. Výběr správných nástrojů a vzoru architektury je zásadní, protože jakmile je jednou architektura zavedena, je velmi obtížné a nákladné ji změnit. V této bakalářské práci uplatněné implementace s nástroji a vzory architektury jsou přizpůsobitelné i pro další projekty ETL s podobným účelem. U procesů výběru ETL je žádoucí brát v úvahu i další témata, jako je ověřitelnost dat, virtualizace, jednotkové testy a jejich hodnocení.

Tato bakalářská práce se zaměřila na výběr nejvhodnějšího ETL nástroje pro uložení dat ze senzorů používaných v zemědělství do datového skladu s možností budoucího využití např. pro machine learning. Nejprve byly charakterizovány dostupné ETL nástroje a možnosti přenosu a ukládání dat z odborných zdrojů. Následně byl sestaven AHP model, který umožnil vybrat nejlepší řešení pro následnou implementaci.

Z AHP modelu vyplynulo, že nejvhodnějšími řešeními jsou Azure Stream Analytics a Amazon Kinesis. Tyto nástroje byly poté implementovány v cloudovém prostředí a následně otestovány vhodně navrhnutým výkonnostním testem.

Z analýzy dat vyplynulo, že oba nástroje dokázaly úspěšně zpracovat datový tok o rychlosti 975 IoT záznamů za sekundu po dobu 10 minut. Nicméně, průměrná hodnota odezvy v Kinesis se jeví jako výrazně nižší než v řešení Azure. Z toho lze vyvodit, že oba nástroje jsou schopny zpracovat datové toky v reálném čase, avšak Kinesis má nižší latenci, což může být výhodné pro náročnější projekty, které vyžadují rychlejší odezvu na data.

Závěrem lze konstatovat, že Kinesis byl vybrán jako nejvhodnější řešení pro implementaci z důvodu pokročilé funkčnosti a možností využití v budoucích aplikacích, i když náklady na jeho provoz jsou o něco vyšší než u nástroje Azure Stream Analytics. Tato bakalářská práce ukázala, že výběr ETL nástroje pro uložení dat ze senzorů je důležitým krokem při tvorbě datového skladu a má zásadní vliv na kvalitu dat a efektivitu následného využití.

7 Seznam použitých zdrojů

1. VERMESAN, O a P FRIESS. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. B.m.: River Publishers, 2013. River Publishers series in communications. ISBN 9788792982735.
2. VERMESAN, O a P FRIESS. *Internet of Things Applications - From Research and Innovation to Market Deployment*. B.m.: River Publishers, 2014. ISBN 9781000797367.
3. HOJLO, Jeffrey. *Future of Industry Ecosystems: Shared Data and Insights* [online]. 2021. Dostupné z: https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/.
4. UR REHMAN, Muhammad Habib, Ibrar YAQOOB, Khaled SALAH, Muhammad IMRAN, Prem Prakash JAYARAMAN a Charith PERERA. The Role of Big Data Analytics in Industrial Internet of Things. *Future Gener. Comput. Syst.* [online]. 2019, **99**(C), 247–259. ISSN 0167-739X. Dostupné z: doi:10.1016/j.future.2019.04.020
5. VERMA, Neetu a Dinesh SINGH. Data Redundancy Implications in Wireless Sensor Networks. *Procedia Computer Science* [online]. 2018, **132**, 1210–1217. ISSN 1877-0509. Dostupné z: doi:https://doi.org/10.1016/j.procs.2018.05.036
6. XIE, Sai a Zhe CHEN. *Anomaly Detection and Redundancy Elimination of Big Sensor Data in Internet of Things* [online]. B.m.: arXiv. 2017. Dostupné z: doi:10.48550/ARXIV.1703.03225
7. PERROS, H G. *An Introduction to IoT Analytics* [online]. B.m.: CRC Press, 2021. Chapman & Hall/CRC Data Science Series. ISBN 9781000337822. Dostupné z: https://books.google.cz/books?id=1N0mEAAAQBAJ
8. KAMILARIS, Andreas a Frank O OSTERMANN. Geospatial Analysis and the Internet of Things. *ISPRS International Journal of Geo-Information* [online]. 2018, **7**(7). ISSN 2220-9964. Dostupné z: doi:10.3390/ijgi7070269
9. BALAKRISHNA SIVADI AND THIRUMARAN, M. and Solanki Vijender Kumar. IoT Sensor Data Integration in Healthcare using Semantics and Machine Learning Approaches. In: Vijender Kumar and Kumar Raghvendra and Ahad Md. Atiqur Rahman BALAS VALENTINA E. AND SOLANKI, ed. *A Handbook of Internet of Things in Biomedical and Cyber Physical System* [online]. Cham: Springer International Publishing, 2020, s. 275–300. ISBN 978-3-030-23983-1. Dostupné z: doi:10.1007/978-3-030-23983-1_11
10. CUKJATI, Jernej, Domen MONGUS, Krista Rizman ŽALIK a Borut ŽALIK. IoT and Satellite Sensor Data Integration for Assessment of Environmental Variables: A Case Study on NO2. *Sensors* [online]. 2022, **22**(15). ISSN 1424-8220. Dostupné z: doi:10.3390/s22155660
11. BREWSTER, Christopher, Ioanna ROUSSAKI, Nikos KALATZIS, Kevin DOOLIN a Keith ELLIS. IoT in Agriculture: Designing a Europe-Wide Large-Scale Pilot. *IEEE Communications Magazine* [online]. 2017, **55**(9), 26–33. Dostupné z: doi:10.1109/MCOM.2017.1600528
12. KIMBALL, R a J CASERTA. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data* [online]. B.m.: Wiley, 2011. ISBN 9781118079683. Dostupné z: https://books.google.cz/books?id=TCLfzU2ilVkC
13. SOUIBGUI, Manel, Faten ATIGUI, Saloua ZAMMALI, Samira CHERFI a Sadok ben YAHIA. Data quality in ETL process: A preliminary study. *Procedia Computer*

Science [online]. 2019, **159**, 676–687. ISSN 1877-0509. Dostupné z: doi:https://doi.org/10.1016/j.procs.2019.09.223
14. HACHEM, Sara, Thiago TEIXEIRA a Valérie ISSARNY. Ontologies for the Internet of Things [online]. 2011. Dostupné z: doi:10.1145/2093190.2093193
15. MINERVA, R, G M LEE a N CRESPI. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of the IEEE* [online]. 2020, **108**(10), 1785–1824. ISSN 1558-2256. Dostupné z: doi:10.1109/JPROC.2020.2998530
16. CISCO SYSTEMS INC. Global Hybrid Cloud Trends Report. Cisco: Networking, Cloud, and Cybersecurity Solutions. *Online* [online]. 2022. Dostupné z: https://www.cisco.com/c/en/us/solutions/hybrid-cloud/2022-trends.html.
17. IORGA, Michaela, Larry FELDMAN, Robert BARTON, Michael MARTIN, Nedim GOREN a Charif MAHMOUDI. *Fog Computing Conceptual Model* [online]. B.m.: Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD. 2018. Dostupné z: doi:https://doi.org/10.6028/NIST.SP.500-325
18. KHOLOD, Ivan, Maria EFIMOVA, Andrey RUKAVITSYN a Shorov ANDREY. Time Series Distributed Analysis in IoT with ETL and Data Mining Technologies. In: Olga GALININA, Sergey ANDREEV, Sergey BALANDIN a Yevgeni KOUCHERYAVY, ed. *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Cham: Springer International Publishing, 2017, s. 97–108. ISBN 978-3-319-67380-6.
19. KLEPPMANN, M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. B.m.: O'Reilly Media, 2017. ISBN 9781491903117.
20. WHITE, Tom. *Hadoop: The Definitive Guide*. 4th vyd. B.m.: O'Reilly Media, Inc., 2015. ISBN 1491901632.
21. AKIDAU, Tyler, Robert BRADSHAW, Craig CHAMBERS, Slava CHERNYAK, Rafael J FERNÁNDEZ-MOCTEZUMA, Reuven LAX, Sam MCVEETY, Daniel MILLS, Frances PERRY, Eric SCHMIDT a Sam WHITTLE. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, out-of-Order Data Processing. *Proc. VLDB Endow.* [online]. 2015, **8**(12), 1792–1803. ISSN 2150-8097. Dostupné z: doi:10.14778/2824032.2824076
22. KAMBURUGAMUVE, Supun, Geoffrey FOX, Judy QIU a David LEAKE. *Survey of Distributed Stream Processing for Large Stream Sources* [online]. 2014. Dostupné z: doi:10.13140/RG.2.1.2938.7927
23. ELLIS, B. *Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data* [online]. B.m.: Wiley, 2014. ISBN 9781118838020. Dostupné z: https://books.google.cz/books?id=DFnOAwAAQBAJ
24. LIAN, X a L CHEN. Similarity Join Processing on Uncertain Data Streams. *IEEE Transactions on Knowledge and Data Engineering* [online]. 2011, **23**(11), 1718–1734. ISSN 1558-2191. Dostupné z: doi:10.1109/TKDE.2010.208
25. LI, X a Y MAO. Real-Time data ETL framework for big real-time data analysis. In: *2015 IEEE International Conference on Information and Automation* [online]. 2015, s. 1289–1294. Dostupné z: doi:10.1109/ICInfA.2015.7279485
26. JEON, Y -H., K -H. LEE a H -J. KIM. Distributed Join Processing Between Streaming and Stored Big Data Under the Micro-Batch Model. *IEEE Access*

[online]. 2019, **7**, 34583–34598. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2019.2904730
27. ZAHARIA, Matei, Mosharaf CHOWDHURY, Tathagata DAS, Ankur DAVE, Justin MA, Murphy MCCAULEY, Michael J FRANKLIN, Scott SHENKER a Ion STOICA. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. USA: USENIX Association, 2012, s. 2. NSDI'12.
28. ZAHARIA, Matei, Tathagata DAS, Haoyuan LI, Scott SHENKER a Ion STOICA. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. In: *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*. USA: USENIX Association, 2012, s. 10. HotCloud'12.
29. ALEXANDROV, Alexander, Rico BERGMANN, Stephan EWEN, Johann-Christoph FREYTAG, Fabian HUESKE, Arvid HEISE, Odej KAO, Marcus LEICH, Ulf LESER, Volker MARKL, Felix NAUMANN, Mathias PETERS, Astrid RHEINLÄNDER, Matthias J SAX, Sebastian SCHELTER, Mareike HÖGER, Kostas TZOUMAS a Daniel WARNEKE. The Stratosphere platform for big data analytics. *The VLDB Journal* [online]. 2014, **23**(6), 939–964. ISSN 0949-877X. Dostupné z: doi:10.1007/s00778-014-0357-y
30. ALLEN, S T, M JANKOWSKI a P PATHIRANA. *Storm Applied: Strategies for real-time event processing* [online]. B.m.: Manning, 2015. ISBN 9781617291890. Dostupné z: https://books.google.cz/books?id=r6K-oAEACAAJ
31. MEEHAN, John, Stan ZDONIK, Shaobo TIAN, Yulong TIAN, Nesime TATBUL, Adam DZIEDZIC a Aaron ELMORE. *Integrating real-time and batch processing in a polystore* [online]. 2016. Dostupné z: doi:10.1109/HPEC.2016.7761585
32. QU, Weiping. *On-Demand ETL for Real-Time Analytics* [online]. B.m., 2021. b.n. Dostupné z: https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/6252
33. CHOI, Jang-Ho, Junyong PARK, Hwin Dol PARK a Ok-gee MIN. DART: Fast and Efficient Distributed Stream Processing Framework for Internet of Things. *ETRI Journal* [online]. 2017, **39**(2), 202–212. ISSN 1225-6463. Dostupné z: doi:https://doi.org/10.4218/etrij.17.2816.0109
34. ILLA, P K a N PADHI. Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics. *IEEE Access* [online]. 2018, **6**, 55162–55170. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2018.2872799
35. KREPS, Jay, Neha NARKHEDE, Jun RAO a OTHERS. Kafka: A distributed messaging system for log processing. In: *Proceedings of the NetDB*. 2011, s. 1–7.
36. GORELIK, A. *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science* [online]. B.m.: O'Reilly Media, 2019. ISBN 9781491931554. Dostupné z: https://books.google.cz/books?id=6SuGjgEACAAJ
37. ROSS, M a R KIMBALL. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* [online]. B.m.: Wiley, 2013. ISBN 9781118732281. Dostupné z: https://books.google.cz/books?id=4rFXzk8wAB8C
38. DEVLIN, Barry A a Paul T MURPHY. An Architecture for a Business and Information System. *IBM Syst. J.* 1988, **27**, 60–80.
39. CLOUDERA INC. *Time Series Data Warehouse: A reference architecture utilizing a modern data warehouse, based on Cloudera 6*. prosinec 2019
40. BISWAS, Neepa, Anamitra SARKAR a Kartick Chandra MONDAL. Empirical Analysis of Programmable ETL Tools. In: Jyotsna Kumar MANDAL, Somnath MUKHOPADHYAY, Paramartha DUTTA a Kousik DASGUPTA, ed.

Computational Intelligence, Communications, and Business Analytics. Singapore: Springer Singapore, 2019, s. 267–277. ISBN 978-981-13-8581-0.
41. SREEMATHY, J, R BRINDHA, M Selva NAGALAKSHMI, N SUVEKHA, N Karthick RAGUL a M PRAVEENNANDHA. Overview of ETL Tools and Talend-Data Integration. In: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)* [online]. 2021, s. 1650–1654. ISBN 2575-7288. Dostupné z: doi:10.1109/ICACCS51430.2021.9441984
42. SAATY, Thomas. Decision making with the Analytic Hierarchy Process. *Int. J. Services Sciences Int. J. Services Sciences* [online]. 2008, **1**, 83–98. Dostupné z: doi:10.1504/IJSSCI.2008.017590
43. MUKHERJEE, R a P KAR. A Comparative Review of Data Warehousing ETL Tools with New Trends and Industry Insight. In: *2017 IEEE 7th International Advance Computing Conference (IACC)* [online]. 2017, s. 943–948. ISBN 2473-3571. Dostupné z: doi:10.1109/IACC.2017.0192
44. POESS, Meikel, Tilmann RABL, Hans-Arno JACOBSEN a Brian CAUFIELD. TPC-DI: The First Industry Benchmark for Data Integration. *Proc. VLDB Endow.* [online]. 2014, **7**(13), 1367–1378. ISSN 2150-8097. Dostupné z: doi:10.14778/2733004.2733009
45. DIOUF, P S, A BOLY a S NDIAYE. Variety of data in the ETL processes in the cloud: State of the art. In: *2018 IEEE International Conference on Innovative Research and Development (ICIRD)* [online]. 2018, s. 1–5. Dostupné z: doi:10.1109/ICIRD.2018.8376308
46. ZAIDI, Ehtisham, Sharat MENON, Robert THANARAJ a Nina SHOWELL. *Gartner Magic Quadrant for Data Integration Tools* [online]. srpen 2022. Dostupné z: https://www.gartner.com/en/documents/4017726
47. GOLDFEDDER, Jarrett. Choosing an ETL Tool. In: *Building a Data Integration Team: Skills, Requirements, and Solutions for Designing Integrations* [online]. Berkeley, CA: Apress, 2020, s. 75–101. ISBN 978-1-4842-5653-4. Dostupné z: doi:10.1007/978-1-4842-5653-4_5
48. PATEL, Monika a Dhiren PATEL. Progressive Growth of ETL Tools: A Literature Review of Past to Equip Future. In: [online]. 2020, s. 389–398. ISBN 978-981-15-6013-2. Dostupné z: doi:10.1007/978-981-15-6014-9_45
49. ZIMA, Michal. *Výběr ETL nástroje pro cloudové řešení BI [online]* [online]. B.m., 2015. b.n. Dostupné z: https://theses.cz/id/fzwmjt/
50. MILTON, Martin. *Porovnání vybraných platforem pro realizaci IoT řešení [online]* [online]. Dostupné z: https://theses.cz/id/dkau2f/

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam grafů

8.4 Seznam použitých zkratek

AHP	Analytic hierarchy process
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ASA	Azure Stream Analytics
AWS	Amazon Web Services
AWSK	Amazon Web Services Kinesis
BI	Business Intelligence
CI	Continuous Integration
CLI	Command Line Interface
CPU	Central Processing Unit
DART	Dart programming language
ETL	Extract-Transform-Load
GB	gigabyte
GCD	Google Cloud Dataflow
GPU	Graphics Processing Unit
GUI	Graphic User Interface
IBM	International Business Machines Corporation

IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
KPU	Kinesis Processing Units
MPP	Massively Parallel Processing
MQTT	Message Queuing Telemetry Transport
NoSQL	non-SQL nebo non-relational
QoS	Quality of Service
RAM	Random Access Memory
RDD	Resilient Distributed Dataset
RFID	Radio Frequency Identification
SAQL	Stream Analytics Query Language
SDK	software development kit
SQL	Structured Query Language
SU	Streaming Units
TPC	Transaction Processing Performance Council
TPC-DI	Transaction Processing Performance Council-Data Integration
vCPU	virtual Central Processing Unit

Přílohy

Příloha A Pseudokód stanovení efektivity nástroje

Příloha B Detailní hodnotící tabulky AHP

Příloha C Test efektivity nástroje Azure Stream Analytics

Příloha D Test efektivity nástroje AWS Kinesis

Příloha E Test efektivity nástroje Google Dataflow

Příloha F Zátěžový test Azure Stream Analytics

Příloha G Zátěžový test AWS Kinesis

Příloha A Pseudokód stanovení efektivity nástroje

1. Připoj se k tématu MQTT a spusť záchyt proudu dat.
2. Filtruj zprávy na základě hodnoty tlaku vzduchového čerpadla.
    a. U každé zprávy zkontroluj hodnotu tlaku vzduchového čerpadla.
    b. Pokud je hodnota vyšší než 50 zprávu ponech, v opačném případě ji vyřaď.
3. Vyčisti zprávy odstraněním pole časové značky a zaokrouhlením hodnoty vlhkosti.
    a. Pro každou zprávu odstraň pole časového razítka.
    b. Zaokrouhli hodnotu vlhkosti na dvě desetinná místa.
    c. Vrať vyčištěnou zprávu.
4. Agreguj zprávy výpočtem průměrné teploty, vlhkosti a tlaku vzduchového čerpadla.
    a. Pro každou dávku zpráv vypočítej průměrnou teplotu, vlhkost a tlak vzduchového čerpadla.
    b. Vrať agregovaná data.
5. Spoj agregované údaje o teplotě s datovým zdrojem podrobných údajů o zařízení na základě hodnoty device_id.
    a. Pro každou zprávu v datech o teplotě ji spoj s odpovídajícími podrobnostmi o zařízení pomocí hodnoty device_id.
    b. Vrať spojená data.
6. Spojená data zapiš do datového skladu.
7. Ukonči datový tok a uzavři spojení.

Zdroj: vlastní práce

Příloha B Detailní hodnotící tabulky AHP

Tabulka 5 Párové porovnání hlavních kritérií

Kritérium	Provozní	Technická	Ostatní	v_i'	v_i
Provozní	1,00	1,00	2,00	1,26	0,4
Technická	1,00	1,00	2,00	1,26	0,4
Ostatní	0,50	0,50	1,00	0,63	0,2

Zdroj: vlastní práce

Tabulka 6 Párové porovnání provozní skupiny kritérií

Kritérium	Efektivita kódu	Podporované jazyky	Dostupné API	Poskytované transformace	v_i'	v_i
Efektivita kódu	1,00	0,50	0,50	0,33	0,54	0,12
Podporované jazyky	2,00	1,00	1,00	0,50	1,00	0,23
Dostupné API	2,00	1,00	1,00	0,50	1,00	0,23
Poskytované transformace	3,00	2,00	2,00	1,00	1,86	0,42

Zdroj: vlastní práce

Tabulka 7 Párové porovnání technické skupiny kritérií

Kritérium	Nativní připojení	Real-time připojení	Latence	Propustnost	v_i'	v_i
Nativní připojení	1,00	0,33	3,00	3,00	1,32	0,25
Real-time připojení	3,00	1,00	5,00	5,00	2,94	0,56
Latence	0,33	0,20	1,00	1,00	0,51	0,10
Propustnost	0,33	0,20	1,00	1,00	0,51	0,10

Zdroj: vlastní práce

Tabulka 8 Párové porovnání obecné skupiny kritérií

Kritérium	Dokumentace	Pořizovací nákl.	Provozní nákl.	v_i'	v_i
Dokumentace	1,00	0,33	0,33	0,48	0,14
Pořizovací nákl.	3,00	1,00	1,00	1,44	0,43
Provozní nákl.	3,00	1,00	1,00	1,44	0,43

Zdroj: vlastní práce

Tabulka 9 Párové porovnání efektivita kódu

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	3	5	2,47	0,637
AWSK	1/3	1	3	1,00	0,258
GCD	1/5	1/3	1	0,41	0,105

Zdroj: vlastní práce

Tabulka 10 Párové porovnání podporované jazyky

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	2	2	1,59	0,5
AWSK	1/2	1	1	0,79	0,25
GCD	1/2	1	1	0,79	0,25

Zdroj: vlastní práce

Tabulka 11 Párové porovnání dostupné API

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	3	3	2,08	0,6
AWSK	1/3	1	1	0,69	0,2
GCD	1/3	1	1	0,69	0,2

Zdroj: vlastní práce

Tabulka 12 Párové porovnání poskytované transformace

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	3	5	2,47	0,637
AWSK	1/3	1	3	1,00	0,258
GCD	1/5	1/3	1	0,41	0,105

Zdroj: vlastní práce

Tabulka 13 Párové porovnání nativní připojení

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	1/3	1	0,69	0,200
AWSK	3	1	3	2,08	0,600
GCD	1	1/3	1	0,69	0,200

Zdroj: vlastní práce

Tabulka 14 Párové porovnání real-time připojení

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	1/7	1/3	0,36	0,088
AWSK	7	1	3	2,76	0,669
GCD	3	1/3	1	1,00	0,243

Zdroj: vlastní práce

Tabulka 15 Párové porovnání latence

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	1/5	1/3	0,41	0,135
AWSK	5	1	3	2,47	0,822
GCD	3	1/3	1	1,00	0,333

Zdroj: vlastní práce

Tabulka 16 Párové porovnání propustnost

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	1	1	1,00	0,333
AWSK	1	1	1	1,00	0,333
GCD	1	1	1	1,00	0,333

Zdroj: vlastní práce

Tabulka 17 Párové porovnání rozsah dokumentace

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	1/2	1/2	0,63	0,200
AWSK	2	1	1	1,26	0,400
GCD	2	1	1	1,26	0,400

Zdroj: vlastní práce

Tabulka 18 Párové porovnání pořizovací náklady

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	1	1	1,00	0,333
AWSK	1	1	1	1,00	0,333
GCD	1	1	1	1,00	0,333

Zdroj: vlastní práce

Tabulka 19 Párové porovnání provozní náklady

Alternativa	ASA	AWSK	GCD	vi'	vi
ASA	1	5	1	1,71	0,455
AWSK	1/5	1	1/5	0,34	0,091
GCD	1	5	1	1,71	0,455

Zdroj: vlastní práce

Příloha C Test efektivity nástroje Azure Stream Analytics

```
WITH Input AS (
    SELECT *
    FROM MQTTInput
)
, Filtered AS (
    SELECT *
    FROM Input
    WHERE air_pump_pressure > 50
)
, Cleaned AS (
    SELECT
        device_id,
        ROUND(humidity, 2) as humidity,
        temperature,
        air_pump_pressure
    FROM Filtered
)
, Aggregated AS (
    SELECT
        device_id,
        AVG(temperature) AS avg_temperature,
        AVG(humidity) AS avg_humidity,
        AVG(air_pump_pressure) AS avg_air_pump_pressure,
        System.Timestamp() AS WindowEnd
    FROM Cleaned
    GROUP BY
        device_id,
        TumblingWindow(minute, 1)
)
, Joined AS (
    SELECT
        Aggregated.device_id,
        Aggregated.avg_temperature,
        Aggregated.avg_humidity,
        Aggregated.avg_air_pump_pressure,
        DeviceDetails.*
    FROM Aggregated
    JOIN DeviceDetails ON Aggregated.device_id = DeviceDetails.device_id
)
SELECT *
INTO DataWarehouse
FROM Joined
```

Zdroj: vlastní práce

```
CREATE STREAM filtered_stream (
    air_pump_pressure INTEGER,
    temperature DOUBLE,
    humidity DOUBLE
);
CREATE PUMP "stream_pump" AS INSERT INTO filtered_stream
SELECT
    air_pump_pressure,
    temperature,
    ROUND(humidity, 2)
FROM
    source_stream
WHERE
    air_pump_pressure > 50;
CREATE STREAM aggregated_stream (
    temperature DOUBLE,
    humidity DOUBLE,
    air_pump_pressure DOUBLE
);
CREATE PUMP "aggregated_pump" AS INSERT INTO aggregated_stream
SELECT
    AVG(temperature) AS temperature,
    AVG(ROUND(humidity, 2)) AS humidity,
    AVG(air_pump_pressure) AS air_pump_pressure
FROM
    filtered_stream
WINDOW TUMBLING (SIZE 1 MINUTE);
CREATE STREAM joined_stream (
    device_id VARCHAR(50),
    temperature DOUBLE,
    humidity DOUBLE,
    air_pump_pressure DOUBLE
);
CREATE PUMP "join_pump" AS INSERT INTO joined_stream
SELECT
    device_details.device_id,
    aggregated_stream.temperature,
    aggregated_stream.humidity,
    aggregated_stream.air_pump_pressure
FROM
    aggregated_stream
JOIN
    device_details
ON
    device_details.device_id = aggregated_stream.device_id;
CREATE STREAM sink_stream (
    device_id VARCHAR(50),
```

```
    temperature DOUBLE,
    humidity DOUBLE,
    air_pump_pressure DOUBLE
);
CREATE PUMP "sink_pump" AS INSERT INTO sink_stream
SELECT
    device_id,
    temperature,
    humidity,
    air_pump_pressure
FROM
    joined_stream;
```

Zdroj: vlastní práce

```
class FilterMessagesFn(beam.DoFn):
    def process(self, message):
        if message['air_pump_pressure'] > 50:
            yield message
class CleanMessagesFn(beam.DoFn):
    def process(self, message):
        del message['timestamp']
        message['humidity'] = round(message['humidity'], 2)
        yield message
class AggregateMessagesFn(beam.DoFn):
    def process(self, messages):
        temp_sum, hum_sum, pressure_sum, count = 0, 0, 0, 0
        for message in messages:
            temp_sum += message['temperature']
            hum_sum += message['humidity']
            pressure_sum += message['air_pump_pressure']
            count += 1
        avg_temp = temp_sum / count
        avg_hum = hum_sum / count
        avg_pressure = pressure_sum / count
        yield {'average_temperature': avg_temp, 'average_humidity': avg_hum,
'average_pressure': avg_pressure}
class JoinMessagesFn(beam.DoFn):
    def process(self, element, device_details):
        device_id = element['device_id']
        if device_id in device_details:
            element.update(device_details[device_id])
            yield element
def run():
    with beam.Pipeline() as p:
        messages = (p | beam.io.ReadFromPubSub('<subscription-id>')
                      | beam.ParDo(FilterMessagesFn())
                      | beam.ParDo(CleanMessagesFn())
                      | beam.WindowInto(window.FixedWindows(30))
                      | beam.CombineGlobally(AggregateMessagesFn()).without_defaults())
        device_details = (p | 'Read Device Details' >> beam.io.ReadFromText('<details-file>')
                            | 'Parse Device Details' >> beam.Map(lambda x: (x.split(',')[0],
{'device_name': x.split(',')[1]})))
        joined_data = (messages | 'Join Messages and Device Details' >>
beam.ParDo(JoinMessagesFn(), beam.pvalue.AsDict(device_details)))
        joined_data | 'Write to Data Warehouse' >> beam.io.WriteToBigQuery(
                      '<project-id>:<dataset>.<table>',
schema='average_temperature:FLOAT,average_humidity:FLOAT,average_pressure:FLO
AT,device_id:STRING,device_name:STRING',
                      create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,
                      write_disposition=beam.io.BigQueryDisposition.WRITE_APPEND)
```

Zdroj: vlastní práce

Příloha F Zátěžový test Azure Stream Analytics

```
WITH
PlanStream AS (
   SELECT  id,
         AVG(GetArrayElement(data, 0).[value]) as avg_data,
         MAX(i.stamp) as stamp
   FROM inputStream i TIMESTAMP BY stamp
   WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
   AND GetArrayElement(data,0).[key] = 'planting_depth'
   GROUP BY id, TumblingWindow(second, 1)
),
FerStream AS (
   SELECT  id,
         AVG(GetArrayElement(data, 0).[value]) as avg_data,
         MAX(i.stamp) as stamp
   FROM inputStream i TIMESTAMP BY stamp
   WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
   AND GetArrayElement(data,0).[key] = 'fertilizer_pressure'
   GROUP BY id, TumblingWindow(second, 1)
),
SeedStream AS (
   SELECT  id,
         AVG(GetArrayElement(data, 0).[value]) as avg_data,
         MAX(i.stamp) as stamp
   FROM inputStream i TIMESTAMP BY stamp
   WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
   AND GetArrayElement(data,0).[key] = 'seed_count'
   GROUP BY id, TumblingWindow(second, 1)
),
SoilMoisStream AS (
   SELECT  id,
         AVG(GetArrayElement(data, 0).[value]) as avg_data,
         MAX(i.stamp) as stamp
   FROM inputStream i TIMESTAMP BY stamp
   WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
   AND GetArrayElement(data,0).[key] = 'soil_moisture'
   GROUP BY id, TumblingWindow(second, 1)
),
SoilTempStream AS (
   SELECT  id,
         AVG(GetArrayElement(data, 0).[value]) as avg_data,
         MAX(i.stamp) as stamp
   FROM inputStream i TIMESTAMP BY stamp
   WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
   AND GetArrayElement(data,0).[key] = 'soil_temperature'
   GROUP BY id, TumblingWindow(second, 1)
),
SeedPresStream AS (
```

```
    SELECT  id,
        AVG(GetArrayElement(data, 0).[value]) as avg_data,
        MAX(i.stamp) as stamp
    FROM inputStream i TIMESTAMP BY stamp
    WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
    AND GetArrayElement(data,0).[key] = 'seed_pressure'
    GROUP BY id, TumblingWindow(second, 1)
),
AirPressStream AS (
    SELECT  id,
        AVG(GetArrayElement(data, 0).[value]) as avg_data,
        MAX(i.stamp) as stamp
    FROM inputStream i TIMESTAMP BY stamp
    WHERE DATEDIFF(millisecond, i.stamp, System.Timestamp()) > -2000
    AND GetArrayElement(data,0).[key] = 'air_pressure'
    GROUP BY id, TumblingWindow(second, 1)
),
outStream AS (
SELECT t1.id, t2.id as id2, t3.id as id3,
    t1.stamp as stamp1, t1.stamp as stamp2, t2.stamp as stamp3
FROM PlanStream t1
LEFT OUTER JOIN FerStream t2
ON DATEDIFF(second,t1,t2) BETWEEN -1 AND 1
AND t1.id = t2.id
LEFT OUTER JOIN SeedStream t3
ON DATEDIFF(second,t1,t3) BETWEEN -1 AND 1
AND t1.id = t3.id
LEFT OUTER JOIN AirPressStream t4
ON DATEDIFF(second,t1,t4) BETWEEN -1 AND 1
AND t1.id = t4.id
LEFT OUTER JOIN SoilMoisStream t5
ON DATEDIFF(second,t1,t5) BETWEEN -1 AND 1
AND t1.id = t5.id
LEFT OUTER JOIN SoilTempStream t6
ON DATEDIFF(second,t1,t6) BETWEEN -1 AND 1
AND t1.id = t6.id
LEFT OUTER JOIN SeedPresStream t7
ON DATEDIFF(second,t1,t7) BETWEEN -1 AND 1
AND t1.id = t7.id
)

SELECT *
INTO blobout
FROM outStream PARTITION BY id
```

Zdroj: vlastní práce

```
public class KinesisS3SinkJob {
   private static final Logger log = LogManager.getLogger(KinesisS3SinkJob.class);

   private static DataStream<String> getKinesisStream(String streamName,
                                       StreamExecutionEnvironment streamEnv,
                                       ParameterTool parameter){
      Properties inputProperties = new Properties();
      inputProperties.setProperty(ConsumerConfigConstants.SHARD_GETRECORDS_IN
TERVAL_MILLIS, "100");
      inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION,
parameter.get("region"));
      inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITI
ON,
            parameter.get("stream_init_position"));
      if
(parameter.get("stream_init_position").equalsIgnoreCase(StreamPosition.AT_TIMESTAM
P.name())) {
         inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_TIME
STAMP,
               parameter.get("stream_initial_timestamp"));
      }
      return streamEnv.addSource(new FlinkKinesisConsumer<>(streamName,
            new SimpleStringSchema(),
            inputProperties));
   }

   private static KeyedStream<DeviceMsg, String>
getUnitObjStream(DataStream<String> stream, ParameterTool parameter) {
      return stream
            .map(new MapFunction<String,DeviceMsg>() {
               @Override
               public DeviceMsg map(String rec) throws Exception {
                  ObjectMapper mapper = new ObjectMapper();
                  JsonNode recNode = mapper.readTree(rec);
                  if (recNode == null) {
                     throw new Exception("Exception in reading incoming record");
                  }
                  List<Tuple2<String, Double>> data = new ArrayList<>();
                  data.add(new Tuple2<>(recNode.get("data")
                        .findValuesAsText("key").get(0),
                        Double.valueOf(recNode.get("data")
                              .findValuesAsText("value").get(0))));

                  try {
                     return new DeviceMsg(
                           recNode.get("id").textValue(),
```

```
                        data,
                        recNode.get("desc").textValue(),
                        Long.parseLong(recNode.get("timestamp").textValue())
                    );
                } catch (Exception e){
                    log.info("Error in parsing the input records to Event POJO");
                    return null;
                }
            }})
        .assignTimestampsAndWatermarks(new TimestampsAndWatermarksSupplier())
        .keyBy((new KeySelector<DeviceMsg, String>() {
            @Override
            public String getKey(DeviceMsg msg) {
                return msg.getCode();
            }
        }))
        .window(TumblingEventTimeWindows.of(Time.milliseconds(
                Long.parseLong(parameter.get("aggregate_window_size")))))
        .aggregate(new AverageAggregate())
        .assignTimestampsAndWatermarks(new
TimestampsAndWatermarksAggregateSupplier())
        .keyBy((new KeySelector<DeviceMsg, String>() {
            @Override
            public String getKey(DeviceMsg msg) {
                return msg.getCode();
            }
        }));
    }

    private static StreamingFileSink<String> createS3Sink(String path, ParameterTool
parameter) {

        return StreamingFileSink
                .forRowFormat(new Path(path)
                        , new SimpleStringEncoder<String>("UTF-8"))
                .withBucketAssigner(new DateTimeBucketAssigner<>("yyyy-MM-dd--HH"))
                .withBucketCheckInterval(250)
                .withRollingPolicy(
                        DefaultRollingPolicy.builder()
                                .withRolloverInterval(Duration.ofMillis(Long.parseLong(
                                        parameter.get("rollover_interval"))))
                                .withInactivityInterval(Duration.ofSeconds(60))
                                .build())
                .build();
    }

    public static void main(String[] args) throws Exception {
```

```
        final StreamExecutionEnvironment streamEnv =
StreamExecutionEnvironment.getExecutionEnvironment();

        ParameterTool parameter;
           if (streamEnv instanceof LocalStreamEnvironment) {
              parameter = ParameterTool.fromArgs(args);
           } else {
              Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
              Properties flinkProperties = applicationProperties.get("FlinkAppProperties");
              if (flinkProperties == null) {
                 throw new RuntimeException("Unable to load properties from Group ID
FlinkAppProperties.");
              }
              parameter = ParameterToolUtils.fromApplicationProperties(flinkProperties);
           }

        DataStream<String> unitAirPressStrStream =
              getKinesisStream("unit-air-press-stream", streamEnv, parameter);
        DataStream<String> unitFerPressStrStream =
              getKinesisStream("unit-fer-press-stream", streamEnv, parameter);
        DataStream<String> unitMoisStrStream =
              getKinesisStream("unit-mois-stream", streamEnv, parameter);
        DataStream<String> unitTempStrStream =
              getKinesisStream("unit-temp-stream", streamEnv, parameter);
        DataStream<String> unitPlanDepthStrStream =
              getKinesisStream("unit-plan-depth-stream", streamEnv, parameter);
        DataStream<String> unitSeedPresStrStream =
              getKinesisStream("unit-seed-pres-stream", streamEnv, parameter);
        DataStream<String> unitSeedPressStrStream =
              getKinesisStream("unit-seed-press-stream", streamEnv, parameter);

        KeyedStream<DeviceMsg, String> unitAirPressObjStream =
              getUnitObjStream(unitAirPressStrStream, parameter);
        KeyedStream<DeviceMsg, String> unitFerPressObjStream =
              getUnitObjStream(unitFerPressStrStream, parameter);
        KeyedStream<DeviceMsg, String> unitMoisPressObjStream =
              getUnitObjStream(unitMoisStrStream, parameter);
        KeyedStream<DeviceMsg, String> unitTempObjStream =
              getUnitObjStream(unitTempStrStream, parameter);
        KeyedStream<DeviceMsg, String> unitPlanDepthObjStream =
              getUnitObjStream(unitPlanDepthStrStream, parameter);
        KeyedStream<DeviceMsg, String> unitSeedPresObjStream =
              getUnitObjStream(unitSeedPresStrStream, parameter);
        KeyedStream<DeviceMsg, String> unitSeedPressObjStream =
              getUnitObjStream(unitSeedPressStrStream, parameter);

        DataStream<String> joinedStream = unitAirPressObjStream
                 .join(unitFerPressObjStream)
```

```
                .where(unitAirPressObjStream.getKeySelector())
                .equalTo(unitFerPressObjStream.getKeySelector())
                .window(TumblingEventTimeWindows.of(Time.milliseconds(Long.parseLon
g(parameter.get("joined_window_size")))))
                .apply((JoinFunction<DeviceMsg, DeviceMsg, String>) (deviceMsg,
deviceMsg2) -> {
                    List<Tuple2<String,Double>> data = new ArrayList<>();
                    data.addAll(deviceMsg.getData());
                    data.addAll(deviceMsg2.getData());
                    DeviceMsg msg = new DeviceMsg(
                            deviceMsg.getCode(),
                            data,
                            deviceMsg.getDesc(),
                            Math.max(deviceMsg.getTimestamp(),
                                    deviceMsg2.getTimestamp())
                    );
                    return msg.toString();
                }).join(unitMoisPressObjStream)
                .where(unitAirPressObjStream.getKeySelector())
                .equalTo(unitMoisPressObjStream.getKeySelector())
                .window(TumblingEventTimeWindows.of(Time.milliseconds(Long.parseLong(p
arameter.get("joined_window_size")))))
                .apply((JoinFunction<DeviceMsg, DeviceMsg, String>) (deviceMsg,
deviceMsg2) -> {
                    List<Tuple2<String,Double>> data = new ArrayList<>();
                    data.addAll(deviceMsg.getData());
                    data.addAll(deviceMsg2.getData());
                    DeviceMsg msg = new DeviceMsg(
                            deviceMsg.getCode(),
                            data,
                            deviceMsg.getDesc(),
                            Math.max(deviceMsg.getTimestamp(),
                                    deviceMsg2.getTimestamp())
                    );
                    return msg.toString();
                }).join(unitTempObjStream)
                .where(unitAirPressObjStream.getKeySelector())
                .equalTo(unitTempObjStream.getKeySelector())
                .window(TumblingEventTimeWindows.of(Time.milliseconds(Long.parseLong(p
arameter.get("joined_window_size")))))
                .apply((JoinFunction<DeviceMsg, DeviceMsg, String>) (deviceMsg,
deviceMsg2) -> {
                    List<Tuple2<String,Double>> data = new ArrayList<>();
                    data.addAll(deviceMsg.getData());
                    data.addAll(deviceMsg2.getData());
                    DeviceMsg msg = new DeviceMsg(
                            deviceMsg.getCode(),
                            data,
                            deviceMsg.getDesc(),
```

```
                    Math.max(deviceMsg.getTimestamp(),
                        deviceMsg2.getTimestamp())
                );
                return msg.toString();
            }).join(unitPlanDepthObjStream)
            .where(unitAirPressObjStream.getKeySelector())
            .equalTo(unitPlanDepthObjStream.getKeySelector())
            .window(TumblingEventTimeWindows.of(Time.milliseconds(Long.parseLong(p
arameter.get("joined_window_size")))))
            .apply((JoinFunction<DeviceMsg, DeviceMsg, String>) (deviceMsg,
deviceMsg2) -> {
                List<Tuple2<String,Double>> data = new ArrayList<>();
                data.addAll(deviceMsg.getData());
                data.addAll(deviceMsg2.getData());
                DeviceMsg msg = new DeviceMsg(
                    deviceMsg.getCode(),
                    data,
                    deviceMsg.getDesc(),
                    Math.max(deviceMsg.getTimestamp(),
                        deviceMsg2.getTimestamp())
                );
                return msg.toString();
            }).join(unitSeedPresObjStream)
            .where(unitAirPressObjStream.getKeySelector())
            .equalTo(unitSeedPresObjStream.getKeySelector())
            .window(TumblingEventTimeWindows.of(Time.milliseconds(Long.parseLong(p
arameter.get("joined_window_size")))))
            .apply((JoinFunction<DeviceMsg, DeviceMsg, String>) (deviceMsg,
deviceMsg2) -> {
                List<Tuple2<String,Double>> data = new ArrayList<>();
                data.addAll(deviceMsg.getData());
                data.addAll(deviceMsg2.getData());
                DeviceMsg msg = new DeviceMsg(
                    deviceMsg.getCode(),
                    data,
                    deviceMsg.getDesc(),
                    Math.max(deviceMsg.getTimestamp(),
                        deviceMsg2.getTimestamp())
                );
                return msg.toString();
            }).join(unitSeedPressObjStream)
            .where(unitAirPressObjStream.getKeySelector())
            .equalTo(unitSeedPressObjStream.getKeySelector())
            .window(TumblingEventTimeWindows.of(Time.milliseconds(Long.parseLong(p
arameter.get("joined_window_size")))))
            .apply((JoinFunction<DeviceMsg, DeviceMsg, String>) (deviceMsg,
deviceMsg2) -> {
                List<Tuple2<String,Double>> data = new ArrayList<>();
                data.addAll(deviceMsg.getData());
```

```
                data.addAll(deviceMsg2.getData());
                DeviceMsg msg = new DeviceMsg(
                        deviceMsg.getCode(),
                        data,
                        deviceMsg.getDesc(),
                        Math.max(deviceMsg.getTimestamp(),
                                deviceMsg2.getTimestamp())
                );
                return msg.toString();
            });


        joinedStream.addSink(createS3Sink(parameter.get("s3_output_path"), parameter));

        streamEnv.execute("EDU Flink Streaming Kinesis to S3");

    }
}
```

Zdroj: vlastní práce