Vpeljava tabelaričnih tokov v podatkovno arhitekturo

Tjaša Heričko, Saša Brdnik, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
tjasa.hericko@um.si, sasa.brdnik@um.si, muhamed.turkanovic@um.si

Sodobne podatkovne arhitekture se vse bolj usmerjajo k agilnemu modelu ELT, temelječemu na podatkovnih jezerih in koliščih. Ključna prednost takšnega pristopa je uporaba odprtih tabelaričnih formatov, kot so Apache Iceberg, Hudi in Delta Lake, ki temeljijo na odprtih datotečnih formatih, kot so Avro, ORC in Parquet. V prispevku predstavljamo Tableflow – novo rešitev iz ekosistema Confluent, ki omogoča neposredno predstavitev podatkov v Kafka temah kot odprte tabele v formatu Iceberg ali Delta. S tem se podatki, pridobljeni iz virov OLTP, že v fazi zajema in vnosa pretvorijo v format, primeren za poizvedovanje in učinkovitejše shranjevanje neobdelanih podatkov v podatkovno jezero/kolišče. Kafka v tem kontekstu ne služi le pretočni obdelavi, temveč tudi kot mehanizem za zajem in vnos podatkov, skladen s sodobno velepodatkovno arhitekturo. To bistveno zmanjša izgubo konteksta in sheme, ki se pogosto pojavlja pri klasičnih prenosih med operativnimi in analitičnimi sistemi. V prispevku bomo predstavili uporabnost tabelaričnega toka, prikazali praktično uporabo rešitve Tableflow znotraj platforme Confluent Cloud in integracijo s sodobnimi podatkovnimi arhitekturami ter izvedli primerjalno analizo z obstoječimi pristopi materializacije pretočnih podatkov v tabelarno obliko.

Ključne besede:
odprti tabelarični formati,
podatkovni tokovi,
pretočna obdelava podatkov,
Apache Kafka,
Confluent Tableflow.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.2

1 Uvod

V zadnjem desetletju smo priča preobrazbi podatkovnih arhitektur, ki se od tradicionalnega pristopa **ETL** (*angl. Extract-Transform-Load*) vse bolj nagibajo k agilnejšemu modelu **ELT** (*angl. Extract-Load-Transform*). Namesto da bi podatke najprej preoblikovali in jih šele nato naložili v strogo shematizirana relacijska podatkovna skladišča (*angl. relational data warehouses*) po principu ***"Schema-on-Write"***, sodobne podatkovne arhitekture pogosteje temeljijo na stolpčnih (NoSQL) podatkovnih bazah (*angl. columnar databases*) ter **podatkovnih jezerih** (*angl.* ***data lakes***). V primeru slednjih se izvorni podatki v svoji nestrukturirani ali delno strukturirani obliki najprej naložijo v ciljno hrambo, nato pa se po potrebi preoblikujejo za analizo po principu ***"Schema-on-Read"***. To pomeni, da se struktura podatkom določi šele ob njihovem branju, namesto ob njihovem nalaganju v podatkovno hrambo [1,2].

Tako stolpčno kot vrstično orientirana (tj. klasična) podatkovna skladišča, kakor podatkovna jezera, so se v praksi izkazala kot učinkovita pri izpolnjevanju določenih potreb. Klasična podatkovna skladišča so še posebej primerna za obdelavo strukturiranih podatkov in zagotavljajo zanesljivo upravljanje podatkov. Stolpčno orientirana podatkovna skladišča so optimizirana za analitične poizvedbe nad velikimi količinami podatkov, saj omogočajo hitrejše branje in agregiranje podatkov po stolpcih, kar povečuje učinkovitost pri kompleksnih analizah in poročanju. Medtem ko pa so podatkovna jezera učinkovita pri delu z raznovrstnimi podatki in so stroškovno učinkovita. V zadnjih letih se vse bolj uveljavljajo **podatkovna košišča** (*angl.* ***data lakehouses***), ki združujejo ključne prednosti vseh pristopov in hkrati naslavljajo njihove omejitve v enotni rešitvi. Ko govorimo o pristopih, moramo izpostaviti tudi dejstvo, da so vsi omenjeni pristopi podatkovnih skladišč namenjeni analitični obdelavi podatkov (tj. *On-Line Analytical Processing* – ***OLAP***), medtem ko podatkovna košišča dodajajo hkratno podporo transakcijski obdelavi masovne količine podatkov (tj. *On-Line Transactional Processing* – ***OLTP***). Konceptualno gre pri podatkovnih košiščih torej za uvedbo modela podatkovnega skladišča nad podatkovnim jezerom, kjer specializirane komponente omogočajo podporo transakcijskega modela pri delu z velepopodatki in upravljanje ter uveljavljanje sheme podatkov. S tem se ohranjajo zanesljivost, doslednost in zmogljivost navkljub inherentni prilagodljivosti podatkovnih jezer. V nasprotnem primeru namreč ta brez ustreznih mehanizmov pogosto sčasoma postanejo podatkovna močvirja (*angl. data swamps*). Podatkovno košišče omogoča shranjevanje strukturiranih, delno strukturiranih in nestrukturiranih podatkov, podobno kot podatkovno jezero, a hkrati podpira napredno obdelavo in upravljanje podatkov, kot jo poznamo pri podatkovnih skladiščih. Takšen pristop organizacijam omogoča, da znotraj enotne podatkovne infrastrukture učinkovito upravljajo celoten spekter (vele)podatkov – ne glede na njihovo izvorno obliko ali stopnjo strukturiranosti – in zadovoljujejo različne analitične potrebe, tako pri **paketni obdelavi podatkov** (*angl.* ***batch processing***) kot tudi pri **pretočni obdelavi podatkov** (*angl.* ***stream processing***). Takšna arhitektura omogoča večjo prilagodljivost, boljšo razširljivost ter zmanjšuje kompleksnost pri obdelavi podatkov [2,3,4].

Vse večji porast virov podatkov, ki podatke generirajo neprekinjeno, pričakovanj po svežih podatkih, še posebej za umetnointeligenčne (*angl. artificial intelligence* – ***AI***) aplikacije, in potreba po analitičnem vpogledu v podatke ter sprejemanje odločitev v realnem času postavlja v ospredje **podatkovnega inženirstva** (*angl. data engineering*) pretočno obdelavo podatkov, nepogrešljivi del katere najpogosteje predstavlja **Apache Kafka** s svojim ekosistemom. Kafka zagotavlja neprekinjeno obdelavo tokov podatkovnih zapisov, razvrščenih v teme (*angl. topics*), ob nizkih zakasnitvah in visoki prepustnosti. **Realnočasovno** obdelavo podatkovnih tokov – neprekinjeno, sočasno in na nivoju posameznih podatkovnih zapisov (tj. dogodkov) – nato tradicionalno omogočajo komponente Kafkinega ekosistema, kot sta domorodni komponenti **Kafka Streams API** in **ksqlDB** [5]. V tem kontekstu se pri prenosu tokovnih podatkov iz transakcijskih v analitične sisteme pojavlja izziv kompleksnosti procesov ETL/ELT, ki poskrbijo, da so podatki pripravljeni v obliki, ki je uporabna za analitiko ter **strojno učenje** (*angl.* ***machine learning – ML***). Pri tem so podatki izvzeti iz konteksta, v katerem so nastali in bili upravljani, zato med prenosom iz operativnega v analitično okolje pogosto prihaja do izgube konteksta in sheme podatkov. Eden izmed novejših pristopov k naslavljanju tega izziva je uvedba **tabelaričnih tokov** s pomočjo nedavno predstavljene rešitve **Tableflow**, ki je trenutno na voljo v okviru ekosistema **Confluent**. Ta rešitev omogoča neposredno predstavljanje pretočnih podatkov iz Kafka tem kot **odprte tabele** v formatih **Apache Iceberg** ali **Delta Lake**. S

tem so podatki iz operativnih sistemov že ob zajemu in vnosu pripravljeni v formatu, ki je primeren za učinkovito shranjevanje neobdelanih podatkov v podatkovnem jezeru/košišču in za nadaljnjo obdelavo z izbranim **poizvedovalnim/računskim pogonom/mehanizmom** (*angl.* ***query/compute engine***), ki podpira izbrane odprte tabelarične formate. To omogoča bolj prilagodljivo, razširljivo in učinkovito t. i. **brezglavno podatkovno arhitekturo** (*angl.* ***headless data architecture***), kjer so storitve shranjevanja, upravljanja in dostopanja do podatkov formalno ločene od storitev, ki obdelujejo in poizvedujejo po podatkih. Hkrati je takšna arhitektura skladna s principi sodobnega podatkovnega inženirstva *"**Shift Left**"* (tj. premik obdelave in upravljanja podatkov bližje izvornemu podatkovnemu viru) [6,7,8]. V nadaljevanju prispevka se bomo osredotočili na predstavitev rešitve Tableflow in izpostavili prednosti uvedbe tabelaričnih tokov v podatkovno arhitekturo napram tradicionalnemu prenosu pretočnih podatkov med operativnim in analitičnim svetom. Zraven konceptualnega pregleda bomo praktično prikazali uporabo rešitve Tableflow znotraj platforme Confluent Cloud in integracijo s platformo Databricks ter izvedli primerjalno analizo z obstoječimi pristopi materializacije pretočnih podatkov v tabelarno obliko in poizvedovanja po tokovnih podatkih. Izpostavili bomo, kdaj in zakaj pristop z uporabo tabelaričnih tokov predstavlja boljšo alternativo.

2 Podatkovna jezera in košišča

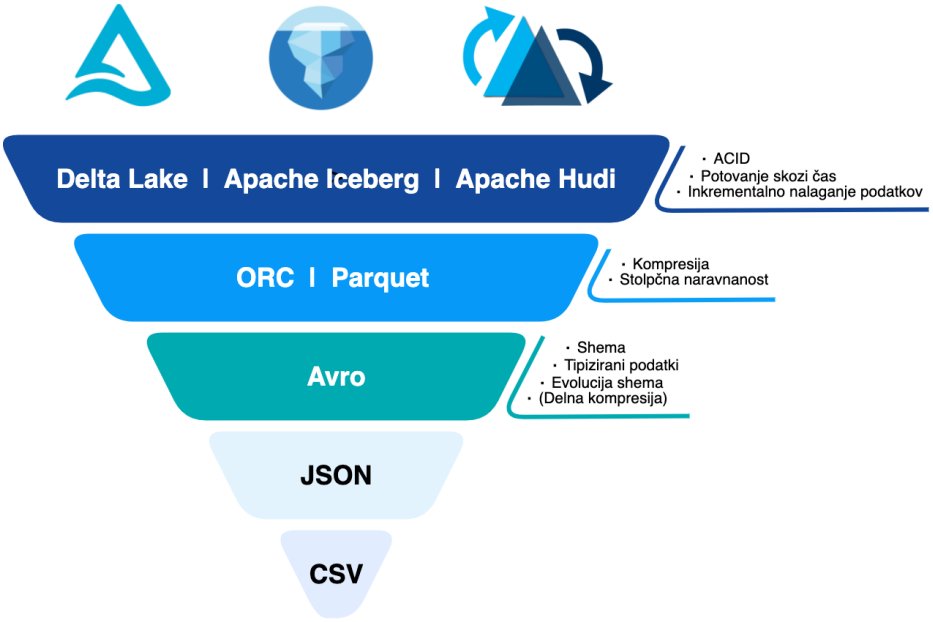
Podatkovna jezera predstavljajo jedro sodobnih podatkovnih arhitektur za shranjevanje podatkov v izvorni, neobdelani obliki, ki v ciljno hrambo vstopajo bodisi kot paketi (*angl.* ***batches***) bodisi kot podatkovni tokovi (*angl. data* ***streams***). V praksi so se izkazala za uporabna predvsem zato, ker podpirajo shranjevanje velike količine podatkov raznovrstnih tipov in različnih velikosti v svoji **izvorni obliki**. Pomemben premik v paradigmi je prehod iz pristopa *"Schema-on-Write"*, kjer je bilo potrebno podatke prilagoditi vnaprej določeni shemi že ob shranjevanju, na pristop *"Schema-on-Read"*, pri katerem se shema uporabi šele ob branju podatkov. Ta pristop omogoča večjo prilagodljivost in enostavnejše delo z nepredvidljivimi ali hitro spreminjajočimi se podatkovnimi strukturami [1,2].

S pomočjo pristopa ELT so podatki iz operativnih sistemov preneseni neposredno v podatkovna jezera, brez predhodne transformacije (T), pri čemer se transformacije izvajajo šele ob branju podatkov. Namesto uporabe zunanjega sistema za izvedbo transformacij podatkov ali izvedbe transformacij pred nalaganjem podatkov, tj. pristop ETL, se transformacije izvajajo znotraj podatkovnega jezera po potrebi. Takšen pristop je botroval t. i. **medaljonski arhitekturi** *(angl.* ***medallion architecture***), kjer imamo znotraj jezera več navideznih con shranjevanja podatkov, tj. bronasta, srebrna in zlata cona, pri čemer podatke v svoji izvorni obliki zmeraj shranjujemo v bronasti coni ter jih nato z enkratno ali večkratno transformacijo (T iz ELT) počistimo, obogatimo in uredimo za srebrno in zlato cono. Prednost takšnega pristopa je izkoriščanje zmogljivosti oblačnih podatkovnih jezer in orodij za obdelavo podatkov, kot so **Amazon S3** (**AWS**), **Google Cloud Storage** (**Google Cloud Platform**) in **Azure Data Lake Storage** (**Microsoft Azure**).

Podatkovna jezera pogosto nimajo urejenega sistema za upravljanje shem, ne podpirajo transakcij, nimajo dobre podpore za analitiko in niso primerna za strukturirane poizvedbe. Omenjeni izzivi so bili v preteklih letih naslovljeni z vpeljavo **podatkovnih košišč** v podatkovne arhitekture, ki v hibridni rešitvi združujejo lastnosti podatkovnih jezer in podatkovnih skladišč. Košišča omogočajo večji nadzor nad upravljanjem podatkov, pri čemer temeljijo na odprtih podatkovnih formatih z neposrednim dostopom preko poizvedb SQL in podporo za enotno obdelavo podatkov za podatkovno analitiko ter strojno učenje. Rešujejo predvsem izziv zastarelih, izgubljenih in neobvladljivih podatkov in pri tem ohranjajo nizke stroške ter interoperabilnost [1,2,9].

Ključno vlogo pri vzpostavitvi podatkovnih košišč igrajo **odprti tabelarični formati** (*angl.* ***open table formats***), ki omogočajo vzpostavitev košišča iz podatkovnega jezera ter tako nadgradijo podatkovno jezero z zmožnostmi podatkovnega skladišča. Najvidnejši predstavniki so **Delta Lake**, **Apache Hudi** in **Apache Iceberg** (Slika 1). Ti temeljijo na **odprtih datotečnih formatih** (*angl.* ***open file formats***), kot so **Apache Avro**, **Apache Parquet** in **Apache ORC**. Odprti tabelarični formati omogočajo ključne funkcionalnosti, kot so na lastnostih ACID temelječe transakcije, možnost potovanja skozi čas (*angl. time travel*) med poizvedovanjem, podpora za evolucijo sheme,

inkrementalno nalaganje podatkov in optimizacija shranjevanju ter branju podatkov [1]. Hkrati omogočajo prilagodljivost glede uporabe različnih **poizvedovalnih/računskih pogonov/mehanizmov** – **Presto**, **Trino**, **Apache Spark**, **Apache Flink** in drugi – za obdelavo podatkov na istem naboru podatkov glede na konkretne primere uporabe. S tem igra uporaba odprtih formatov pomembno vlogo pri demokratizaciji uporabe podatkov, zmanjšuje tveganja omejenosti na določenega ponudnika rešitev in daje večji nadzor organizacijam, kako so podatki shranjeni in uporabljeni [2,3,4].



Slika 1: Odprti formati tabel, temelječi na odprtih formatih datotek, kot osnova za podatkovno kolišče.

Hudi kombinira različne serializacijske tehnike s ciljem podpore izvedbi stolpčnih podatkovnih poizvedb ter hkratne podpore atomarnih posodobitev. Tipična uporaba Hudi je tabela, ki se posodablja s podatki CDC (*angl. Change Data Capture*) iz transakcijske podatkovne baze. Podatki se najprej shranijo v vrstični obliki, nato pa se prepakirajo v stolpčno obliko za izboljšanje učinkovitost poizvedb. Hudi omogoča poizvedovanje po najnovejših podatkih in podpira analitične ter transakcijske posodobitve. Podobno posodobitve v realnem času ter učinkovite poizvedbe omogoča tudi Iceberg, pri čemer ta posebej izstopa s podporo za evolucijo shem in upravljanje tabel pri velikosti podatkov na ravni petabajtov. Delta pa je zasnovan kot dodatna plast na oblaku temelječimi podatkovnimi jezeri, kot je Amazon S3, kjer z uporabo transakcijskega dnevnika omogoča lastnosti ACID, časovno sledenje podatkov ter bistveno hitrejše metapodatkovne operacije pri velikih tabelah [1,10].

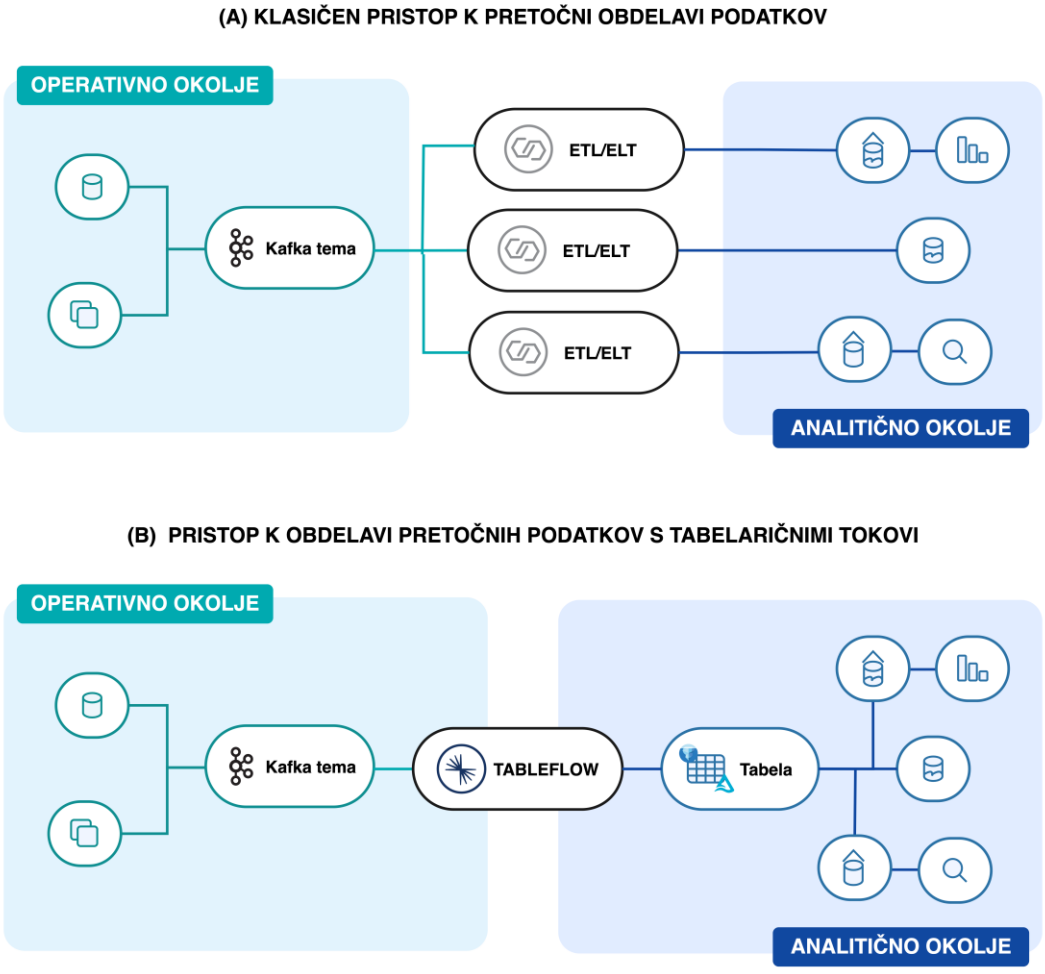
Pomemben trend je vedno tesnejša integracija podatkovnega inženirstva, podatkovne znanosti in umetne inteligence. Napredni sistemi kolišč ter odprti formati omogočajo, da podatkovni znanstveniki in inženirji uporabljajo isti nabor podatkov za analitiko, napovedne modele in AI rešitve brez podvajanja ali premikanja podatkov. Integracija z orodji za strojno učenje, kot sta **TensorFlow** in **PyTorch**, ter platformami za **MLOps**, kot so **MLflow**, **Vertex AI** (Google Cloud), **SageMaker** (AWS) in **Azure Machine Learning**, omogoča celoten življenjski cikel modelov – od zbiranja podatkov, učenja, ovrednotenja, do uvajanja v produkcijo. Ta povezava ustvarja inteligentne podatkovne platforme, kjer so podatki, analitika in AI neločljivo povezani, kar omogoča proaktivno prilagajanje procesov na podlagi napovedi in avtomatiziranega odločanja. Tukaj zaseda posebno mesto prav **Apache Spark** – porazdeljen računski pogon, ki združuje paketno in pretočno obdelavo podatkov, omogoča izvajanje poizvedb SQL (**Spark SQL**), vključuje knjižnice za strojno učenje (**MLlib, PySpark**), grafovno obdelavo (**GraphX**) ter integracijo z različnimi podatkovnimi viri. Zaradi teh lastnosti Spark pogosto deluje kot *"hrbtenica"* podatkovnih platform, kjer povezuje podatkovna jezera, kolišča in analitična orodja.

3 Tabelarični podatkovni tokovi (Tableflow)

Tabelarični podatkovni tokovi rešitve Tableflow predstavljajo nedavno inovacijo v sodobnih podatkovnih arhitekturah, ki omogoča neposredno **preslikavo neprekinjenih tokov dogodkov**, kot so **Kafka teme**, **v odprte tabelarične formate**, kot sta **Iceberg** in **Delta**. Uvedba tabelaričnih tokov stremi k združevanju realnočasovnih podatkovnih tokov z zajemom in vnosom podatkov v ciljno podatkovno hrambo oziroma pripravo podatkov za analitiko. Tako so podatki, ki tečejo v realnem času po Kafkah temah, z uporabo rešitve Tableflow dostopni tudi kot analitične tabele v podatkovnem jezeru/košišču. S tem se omogoča možnost uporabe istih podatkov tako v realnočasovnih aplikacijah kot v analitičnih poizvedbah na dosleden način [6,7].

Klasičen pristop k pretočni obdelavi podatkov tradicionalno vključuje relativno kompleksne procese za prenos podatkov iz operativne v analitične sisteme. Potrebna je vzpostavitev ustrezne infrastrukture (tipično ustrezno konfiguriranje povezovalnikov za Kafko) ter implementacija nizov procesov za ustrezno pripravo podatkov. Ti procesi vključujejo transformacijo dogodkov v tabelarične formate, skrb za sheme, vključno z uporabo registrov shem in upravljanjem evolucije shem, ter neprestano združevanje množice majhnih datotek, ki nastanejo zaradi kontinuiranega toka, s ciljem ohranjanja sprejemljive hitrosti poizvedb. Če se dogodki pretakajo iz procesov CDC, je treba dodatno materializirati in uporabiti te spremembe za pridobivanje relevantnega stanja podatkov. Vse to zahteva veliko inženirskega truda in računskih virov, pri čemer so takšni pristopi nagnjeni k napakam, (le) za to, da se tok podatkov pretvori v obliko, uporabno za hrambo v podatkovnem jezeru in analitiko [7].

Za razliko od klasičnega pristopa k pretočni obdelavi podatkov pristop z uporabo tabelaričnih tokov združuje operativno in analitično okolje, s čimer se odpravlja zgodovinska ločnica med njima, kot je ilustrirano na Sliki 2. Namesto ročnega branja podatkov iz Kafka teme in nalaganja podatkov v podatkovno hrambo, Tableflow poenoti tok in tabelo. Isti podatki so sočasno dostopni kot podatkovni tokovi in kot tabela, brez potrebe po dodatnih procesih ETL. Pri tem funkcionalnosti rešitve Tableflow znotraj ekosistema Confluent v celoti poskrbijo za avtomatizacijo procesov, ki jih je treba pri klasičnem pristopu ročno obvladovati. Tableflow prinaša inovacije v Confluentovi shrambi **Kora**, na osnovi katerih so Kafka teme neposredno materializirane v datotečnem formatu **Parquet**, ki oblikujejo bodisi tabelo **Iceberg** bodisi tabelo **Delta**. Pri tem se sheme ne podvajajo, prav tako jih ni treba ročno preslikovati. Namesto tega rešitev Tableflow uporablja register shem kot vir resnice in poskrbi za skladnost shem ter njihovo evolucijo brez prekinitev delovanja ob morebitnih spremembah. Posledično ni potrebnega ročnega ujemanja polj in tipov, kar v klasičnem pristopu povzroča izzive ob vsaki spremembi v izvorni operativni aplikaciji. Pri pristopu z uporabo tabelaričnih tokov se pravila kakovosti podatkov uveljavljajo v podatkovnem toku pri izvoru, nezdružljivi podatki pa so zavrnjeni zgodaj. Prav tako rešitev Tableflow neprekinjeno v ozadju združuje manjše datoteke, da zagotovi učinkovitost bralnih poizvedb na nastalih tabelah, kar je z uporabo povezovalnikov v sklopu klasičnih pristopov težko doseči v realnem času. Razlika je tudi v dostopu do podatkov, in sicer pri tradicionalnem pristopu so isti podatki razpršeni v dveh sistemih, v Kafki za podatkovne tokove in v podatkovnem jezeru, na primer Hadoop HDFS ali S3, za tabelo, medtem ko pristop s tabelaričnimi tokovi omogoča, da porabniki v operativnem okolju še naprej berejo realnočasovne podatkovne tokove (Kafka API), analitična orodja pa istočasno berejo Iceberg/Delta tabelo preko poizvedb SQL. Pri tem obe predstavitvi temeljita na enem samem, usklajenem naboru podatkov. To je možno zaradi dualnosti tok-tabela, kjer sta tok in tabela dva pogleda na iste podatke, pri čemer rešitev Tableflow to dualnost realizira na ravni hrambe podatkov [7].



Slika 2: Primerjava med (A) klasičnim pristopom k pretočni obdelavi podatkov in (B) pristopom s tabelaričnimi tokovi (povzeto po [7]).

Vpeljava tabelaričnih tokov v podatkovne arhitekture, tj. neposredno preslikovanje Kafka tem v tabele Iceberg/Delta, na področju podatkovnega inženirstva prinaša številne ključne prednosti, ki jih lahko povzamemo v naslednjih vidikih:

- ***Kompleksnost podatkovne arhitekture***. Kafka teme se neposredno predstavijo kot Iceberg/Delta tabele, kar močno poenostavi podatkovno arhitekturo in odpravi potrebo po kompleksnih prilagojenih procesih ETL. Tabelarični tokovi tako omogočajo t. i. pristop *"Zero ETL"*, saj so podatki iz podatkovnih tokov na avtomatiziran način brez dodatnih procesov na voljo kot analitične tabele z minimalno inženirskega truda za organizacije. Pri tem je poskrbljeno za skladnost sheme med podatkovnim tokom in tabelo – vsaka Kafka tema, vključena v tabelarične tokove, mora imeti definirano shemo, ki se nato uporablja tudi pri ustvarjanju tabele. S tem se zagotovi strukturiranost in konsistentnost podatkov v tabeli, avtomatsko pa se podpira tudi evolucija sheme z ohranjanjem združljivosti. Rešitev Tableflow avtomatsko upravlja formatiranje in vzdrževanje tabel. Dogodke v vhodnih formatih sproti pretvarja v stolpčne datoteke Parquet, ki tvorijo podlago tabelam Iceberg/Delta. Pri neprestanem dotoku podatkovnih tokov nastajajo številne majhne datoteke, ki jih Tableflow samodejno združuje in optimizira, briše zastarele datoteke ter tako ohranja hitro branje in učinkovito porabo prostora. Vse to poteka brez podvajanja podatkov, saj Kafka in tabela delita isto osnovo podatkov, tabela pa je običajno dostopna kot *samo-za-branje* (*angl. read-only*), kar zagotavlja konsistentnost in integriteto. Z vsem tem se z vpeljavo tabelaričnih tokov zmanjša inženirski napor, s čimer se razbremenijo ekipe podatkovnih inženirjev, odpravljajo se

nekateri podvojeni koraki obdelave in hrambe podatkov, kar se odraža v manj kompleksni podatkovni arhitekturi napram klasičnim pristopom k pretočni obdelavi podatkov [6,7].

- ***Aktualnost in relevantnost podatkov za analitiko***. Ker se dogodki iz operativnih sistemov v realnem času pretakajo v odprte tabelarične formate, so podatki iz operativnih virov v podatkovnem jezeru/košču neprestano sveži, brez bistvene zakasnitve med dogodkom in njegovo razpoložljivostjo za poizvedovanje. To omogoča hitrejši dostop do svežih podatkov, saj podatki prispejo, so obdelani in na voljo v skoraj realnem času za takojšnje poizvedbe. Prav tako z vpeljavo tabelaričnih tokov zaradi dualnosti tok-tabela ne prihaja do izgube konteksta in sheme podatkov. Pri tem Kafka ne služi le pretočni obdelavi, temveč tudi kot mehanizem za zajem in vnos podatkov. Analitiki in podatkovni znanstveniki lahko tako obravnavajo aktualne in relevantne podatke, kar bistveno zmanjša t. i. *"Time-to-Insight"* in poveča vrednost podatkovne analitike, kakor tudi pospešuje uvajanje umetnointeligenčnih rešitev ter izboljšuje proces podpore odločanju [6,7].
- ***Integracija z obstoječimi orodji in okolji***. Ker sta Iceberg in Delta odprta standarda, Tableflow močno zmanjšuje odvisnost od posameznih orodij in ponudnikov. Iceberg ali Delta imata široko podporo v analitičnih ogrodjih in orodjih, kot so Spark, Trino, Flink, Snowflake in Databricks. S preslikovanjem Kafka tem v Iceberg/Delta se izognemo omejenosti na enega ponudnika rešitev. Podatki so shranjeni v odprti obliki, ki jo lahko bere katerikoli ustrezen pogon, ki je združljiv z odprtimi tabelami. Tabelarični tokovi tako omogočajo, da isti nabor podatkov takoj izkoriščajo različna orodja znotraj različnih okolij, ki lahko dostopajo do tabel brez posebnih prilagoditev [6,7].

4 Tehnološki pregled rešitve Tableflow

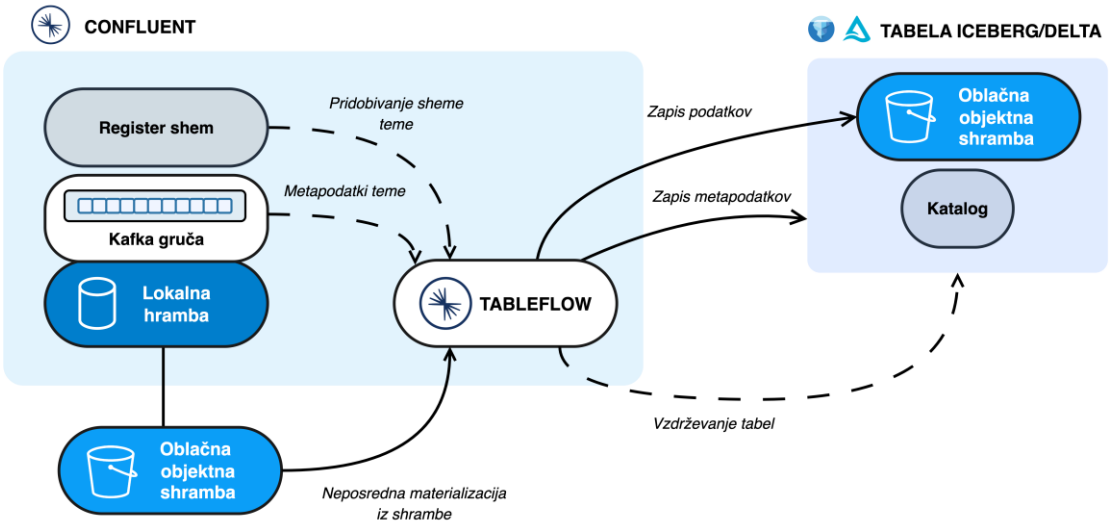
Rešitev Tableflow je storitev znotraj ekosistema Confluent za pretočno podatkovno platformo in je postala splošno dostopna v začetku leta 2025. Gre za popolno vodeno storitev v okviru **Confluent Cloud**, ki nadgrajuje jedro Apache Kafka z novim slojem za materializacijo podatkov. Pri tem Tableflow ni zgolj še en povezovalnik, temveč je globoko integriran v platformo, tako v podatkovno shrambo kot tudi v upravljanje shem in obdelavo tokov. Pri tem Tableflow predstavlja korak v smeri, da podatkovna platforma učinkovito služi tako operativnim aplikacijam kakor analitičnim potrebam na skupnih podatkih v odprtem formatu [6].

V Confluent Cloud konzoli, tj. preko grafičnega uporabniškega vmesnika, ali preko API/CLI lahko podatkovni inženir na izbranih Kafka temah omogoči Tableflow v zgolj nekaj konfiguracijskih korakih. Za vsako takšno temo sistem avtomatsko vzpostavi spremljajočo tabelo, pri čemer podatkovni inženir predhodno izbere željen format, ki se sproti posodablja z novimi dogodki. Tableflow se opira na odprte tabelarične formate, trenutno na Iceberg (splošno dostopen) in Delta (v odprti predogledni fazi). V ozadju Confluent Cloud upravlja infrastrukturo, kar pomeni, da uporabniku ni treba skrbeti za strežnike, shrambo datotek ali usklajevanje – Tableflow deluje kot platformna funkcionalnost. Trenutno je Tableflow na voljo izključno v okolju Confluent Cloud pri določenih ponudnikih storitev v oblaku, na primer AWS, in ni del klasične namestitve Confluent Platform On-Premises, kar poudarja njegovo odvisnost od upravljalnega okolja Confluent in novega shranjevalnega sloja. Kot del ponudbe Confluent se Tableflow obračunava glede na čas uporabe teme in obseg obdelanih podatkov (GB), podobno kot njihove preostale oblačne storitve [6].

Tableflow je zasnovan kot večslojna nadgradnja Kafka infrastrukture. Ključna inovacija je v nadgradnji shranjevalnega podsistema ***Kora Storage Layer***. Kora, ki že sicer skrbi za t. i. *"Tiered Storage"*, tj. premikanje starejših dogodkov Kafka teme na poceni objektno shrambo, je razširjena tako, da lahko iz dogodkov v Kafka temi sproti ustvarja datoteke Parquet. S tem Kafka dogodki postanejo gradniki tabele Iceberg/Delta. Namesto da bi ločen povezovalnik zunaj Kafke bral dogodke in pisal nove kopije v datotečni sistem, Tableflow doseže ***"zero-copy"* materializacijo**. Isti podatki, ki jih Kafka hrani (v Confluent Cloud so Kafka podatki v S3), se predstavijo v obliki datotek Parquet in pripadajočih metapodatkov tabel. Komponenta *Metadata Materializer* v Tableflow se

navezuje na *Confluent Schema Registry*, iz katerega dobi shemo dogodkov, in na podlagi te ustvarja Iceberg metapodatke (manifesti, razvezave) ali transakcijske zapise Delta. Sproti se izvajajo pretvorbe podatkovnih tipov med Kafka shemo in tabelo ter sledi pravilom združljivosti shem (na primer, če se shema nadgradi z novim poljem, se ta sprememba zabeleži v Iceberg/Delta shemi, skladno z nastavitvami kompatibilnosti) [6,7].

Podrobnejša tehnična arhitektura in pretok podatkov s pomočjo Tableflow je prikazan na Sliki 3. Ko dogodki prihajajo v Kafka temo, jih shranjevalni sloj rešitve Tableflow zajeme v obliki mini-datotek odprtega datotečnega formata Parquet na objektni shrambi. Zaradi zagotavljanja nizke zakasnitve se ti zapisi lahko sprva zapišejo kot manjše datoteke. Vzporedno pa procesi v ozadju izvajajo združevanje – manjše datoteke združujejo v večje posnetke datotek, kar izboljšuje poizvedbe. Tableflow s tem rešuje znano omejitev ločenih povezovalnikov, kjer se je bilo treba odločati med hitrostjo prenosa (majhne datoteke, a slabše poizvedbe) in učinkovitostjo poizvedb (velike datoteke, a višja zakasnitev). Z integracijo v shrambo Kafka se lahko optimizira oboje – piše se pogosto za sproten dotok, a v ozadju optimizira za branje. Ko se zapiše nov segment/tabela, *Metadata Materializer* posodobi posnetek stanja Iceberg ali dnevniški zapis Delta, tako da je nov paket podatkov takoj viden analitičnim porabnikom. Kot posledica takšna arhitekturna zasnova povzroča, da obstaja en sam vir podatkov – Kafka posredniki (*angl. broker*) in objektna shramba skupaj hranijo podatke, tabela in Kafka tema pa predstavljata *"dve strani istega kovanca"*. Porabniki lahko torej ali berejo celoten tok prek Kafka API (za minimalno zakasnitev) ali pa dostopajo do posameznih dogodkov oziroma celotne zgodovine preko poizvedb SQL na tabeli (za *ad-hoc* analize) [11].



Slika 3: Tehnična arhitektura rešitve Tableflow v ekosistemu Confluent (povzeto po [11]).

V Confluentoven ekosistemu se Tableflow povezuje s storitvami **kataloga podatkov**, ki vsebuje vgrajen katalog **Iceberg REST** za dostop do tabel. To pomeni, da lahko uporabnik v kateremkoli združljivem pogonu, na primer Spark, Trino, Flink, Presto, ustvari Iceberg tabelo z uporabo URL-ja Confluentovega kataloga in API ključev ter tako poizveduje neposredno po Kafkah podatkih v tabelarni obliki. Hkrati pa podpira integracijo z zunanjimi kataloškimi storitvami, ko so **AWS Glue**, **Snowlake/Apache Polaris** ipd., da se tabela lahko registrira tudi v obstoječe metapodatkovne sisteme organizacije [6,7].

5 Primeri uporabe in integracija s sodobnimi podatkovnimi arhitekturami

Eden od pogostih primerov uporabe je obdelava podatkov o uporabniških interakcijah iz več različnih aplikacij, kot so mobilne aplikacije in spletni brskalniki. Ti dogodki se v realnem času pošiljajo v Kafka temo in so lahko ob podpori Tableflow nemudoma preneseni in dostopni kot analitične tabele v okolju Databricks. Tako lahko analitične ekipe gradijo nadzorne plošče in analize v realnem času brez ločenih postopkov ETL in brez skrbi glede zadrževanja (*angl. retention policy*) podatkov v Kafki. Podoben pristop se uporablja pri aplikacijah strojnega učenja, kjer je cilj ne le obdelava dogodkov, ampak tudi njihovo trajno shranjevanje za kasnejše ponovno učenje modelov. Če bi se pri tem zanašali le na podatke v Kafka temi, bi bili omejeni z nastavljenim zadrževanjem in potencialno nepopolno zgodovino. Tableflow omogoča, da se vsi dogodki zapisujejo v podatkovno jezero v tabelo Delta z neomejeno hrambo, kar daje podatkovnim znanstvenikom popoln dostop do zgodovinskih podatkov ter hkrati vso moč podatkovnih kolišč.

Za bolj poglobljeno ponazoritev si poglejmo primer podjetja, ki uporablja platformo **Databricks** kot svoje podatkovno kolišče in osrednjo platformo za analitiko ter strojno učenje. Databricks združuje lastnosti podatkovnega jezera in skladišča, hkrati pa z **Unity Catalog** nudi centralizirano upravljanje metapodatkov, varnosti in sledljivosti. V takšni arhitekturi Tableflow materializira podatke iz Kafka tem neposredno v tabelo Delta, shranjeno v zunanjem oblaku (npr. Amazon S3) v načinu *Bring-Your-Own-Storage*. Databricks to tabelo registrira kot zunanjo tabelo, kar pomeni, da se tabela Delta piše direktno v zunanje vedro (*angl. bucket*) in je potem na voljo kot zunanja tabela v okolju Databrick [6,7]. To omogoča izvajanje poizvedb SQL s pomočjo **Spark SQL**, integracijo z ML cevovodi prek **PySpark** ter uporabo funkcionalnosti BI.

V številnih primerih je pristop **samo Tableflow** (t. i. *"Zero ETL"*) povsem zadosten – podatki se iz Kafke takoj zapišejo v odprt tabelarični format (tj. Delta), kar omogoča analitikom in podatkovnim znanstvenikom neposreden dostop, pri čemer se prečiščevanje in obogatitev podatkov izvedeta šele znotraj okolja podatkovnega kolišča (tj. ELT in prenos podatkov iz bronaste v srebrno ali zlato cono). Tak pristop je optimalen, ko ni nujne potrebe po takojšnjem odstranjevanju občutljivih podatkov, deduplikaciji ali zapletenih agregacijah pred vnosom v jezero.

Kadar pa so potrebni **kritični postopki že pred materializacijo** (*"Shift Left"*), se v arhitekturo lahko vključi procesor podatkovnih tokov, kot je **Flink** ali **Kafka Streams**. Flink je primernejši za kompleksne transformacije z zahtevno obdelavo dogodkov po času dogodka (*angl. event time*), uporabo naprednih časovnih oken itn. Kafka Streams pa je primernejša izbira, kadar so transformacije tesno vezane na domeno aplikacije, obsegajo manjša stanja in želimo logiko izvajati znotraj obstoječih mikrostoritev brez dodatne infrastrukture. V obeh primerih transformiran tok preide v novo Kafka temo, iz katere Tableflow izvede materializacijo prav tako v novo odprto tabelo.

V praksi se lahko uporabi **dvojna pot**: surova pot, kjer Tableflow neposredno materializira izvorni tok v **bronasto** tabelo, in *curated* pot, kjer se podatki predhodno obdelajo (npr. s Flink ali Kafka Streams) ter nato materializirajo v **srebrno** tabelo. Tak pristop zagotavlja popolno arhivsko sled surovih podatkov, hkrati pa ponuja prečiščene in optimizirane podatke za hitrejše poizvedbe ter zmanjšanje stroškov obdelave v okolju Databricks. Takšna arhitektura omogoča jasno ločitev odgovornosti: **Flink** ali **Kafka Streams za transformacijo, Tableflow za materializacijo, Databricks** skupaj s **Spark za analitiko in strojno učenje**.

Ključna prednost uporabe Tableflow je torej ta, da podatki iz realnočasovnih tokov postanejo neposredno del *lakehouse* ekosistema v obliki odprtih tabel, kot je Delta, kjer so takoj dostopni prek standardnih poizvedb SQL ter brez težav vključeni v obstoječe analitične procese in procese strojnega učenja. Namesto da bi jih morali nekoč brati neposredno iz Kafke – s čimer je bil obseg zgodovinskih podatkov morda omejen zaradi retencijskih politik in so bile kompleksne analize otežene – ali pa graditi lastne cevovode ETL, ki so zahtevali dodatno infrastrukturo, vzdrževanje in uvajali latenco, so ti podatki lahko zdaj neomejeno dostopni, konsistentni in optimizirani za uporabo. Prednost tega, da so sedaj tokovni podatki del podatkovnih kolišč z odprtimi tabelami, je med drugim podpora ACID transakcijam, ki zagotavljajo zanesljivo in predvidljivo stanje podatkov tudi ob sočasnih obdelavah. V klasičnih podatkovnih jezerih (brez odprtih tabel) je bilo to težko zagotoviti, ker si delal z datotekami (Parquet, ORC) brez transakcijskega dnevnika – kar je vodilo do nepopolnih naborov podatkov. Prav tako je prednost kolišč

ta, da omogočajo napredne mehanizme optimizacije poizvedb, kot so združevanje datotek (*angl. file compaction*), indeksiranje in fizično urejanje (*angl. z-ordening, sorting, clustering*) po pogosto uporabljenih ključih, kar bistveno skrajša čas poizvedb in zniža stroške. V jezerih brez te logike je bil dostop do podatkov pogosto drag, ker si moral prebrati ogromno nepotrebnih datotek. Odprti formati z metapodatkovnim slojem vedo, **kje** so relevantni podatki. Košišča hkrati omogočajo t. i. *time travel*, torej vpogled v pretekla stanja tabele in povrnitev na prejšnje verzije, kar podpira revizije, reproducibilnost modelov strojnega učenja in obnovo po napakah. Lahko rečeš: *"Daj mi stanje tabele, kot je bilo včeraj ob 15:00"* ali *"pri verziji commit-a št. 57"*. Prej si moral te scenarije reševati z ročnimi kopijami datotek (verzije map), kar je bilo težko in neučinkovito. Z odprtimi tabelami imaš to vgrajeno. Takšna arhitektura ne združuje le tokovnih in paketnih podatkov v enoten analitični prostor, temveč omogoča tudi enostavno in neposredno učenje modelov na celotni zgodovini dogodkov, kar vodi v hitrejše odločitve, boljše napovedne modele in manj tehničnega dolga.

6 Primerjalna analiza rešitve Tableflow s klasičnimi pristopi

Rešitev **Tableflow** v okolju Confluent Cloud je zasnovana za neposredno materializacijo Kafka tem v odprte tabelarične formate, kot sta Iceberg in Delta, z visoko stopnjo avtomatizacije. Njena naloga je zagotoviti, da so podatki, ki prihajajo iz tokov, sočasno dostopni kot analitične tabele, optimizirane za poizvedbe, brez potrebe po ročnih postopkih ETL. Da bi objektivno ocenili njen doseg in prednosti, jo je smiselno primerjati s pristopi, ki zasledujejo enak cilj – materializacijo pretočnih podatkov v tabelarno obliko.

Kafka Connect Object-Storage Sink predstavlja klasičen pristop materializacije, kjer se Kafka tokovi zapisujejo v podatkovna jezera in objektne shrambe (npr. S3, HDFS, Azure Data Lake/Blob, GCS) kot Parquet/Avro/ORC. Prednost je enostavna konfiguracija in široka podpora ciljnih sistemov, slabost pa, da ponor (*angl. sink*) praviloma ne vzpostavi metapodatkov odprtih tabel (Iceberg/Delta) in ne izvaja samodejne optimizacije datotek, kot so združevanje majhnih datotek (*angl. small files compaction*), gručanje po pogosto uporabljenih stolpcih (*angl. clustering*) ali dodajanje particij. Posledica tega je, da ob neprestanem zapisovanju majhnih paketov podatkov v shrambo nastajajo tisoči majhnih datotek, kar pri analitičnih poizvedbah povzroča veliko število bralnih operacij in podaljša odzivni čas. Na primer, poizvedba v Spark SQL, ki bere tabelo z 10 milijoni zapisov, razpršenih v 200.000 majhnih datotek, se lahko izvaja več minut, medtem ko enaka poizvedba nad optimizirano tabelo, kjer so datoteke združene in ustrezno particionirane (npr. po datumu ali regiji), zaključi v nekaj sekundah. Ker ponor ne upravlja particij, indeksov ali metapodatkov, mora uporabnik samostojno izvajati naknadne postopke optimizacije, pogosto z dodatnimi procesi ETL ali orodji (npr. Spark Jobs). Upravljanje evolucije shem in registracija tabel v analitičnih okoljih tako ostane na strani uporabnika ali pa zahteva integracijo z ločenimi metapodatkovnimi storitvami.

Apache Hudi DeltaStreamer je orodje, ki omogoča pretok podatkov iz Kafke neposredno v Hudi tabele, z osnovno podporo za evolucijo shem in optimizacijo podatkovnih datotek. Je odprtokodna rešitev, ki zahteva samostojno namestitev, konfiguracijo in vzdrževanje infrastrukture. Čeprav ponuja integrirano obdelavo in zapis v format, primeren za analitiko, je upravljanje sistema zahtevnejše in manj primerno za ekipe, ki nimajo izkušenj z administracijo Hadoop/Spark okolij.

Apache Flink v kombinaciji s **SQL** in ustreznim povezovalnikom *sink* za Iceberg ali Delta omogoča, da se podatki iz Kafka tokov neposredno pretvorijo v odprt tabelarični format. Prednost te rešitve je visoka fleksibilnost – uporabnik lahko izvede kompleksne transformacije, filtriranje in obogatitev podatkov že pred zapisom v tabelo. Slabost pa je višja zahtevnost pri postavitvi in upravljanju okolja, potreba po ločeni Flink infrastrukturi in dodatno kodiranje oziroma konfiguracija za upravljanje shem ter optimizacijo datotek.

Tableflow vse zgoraj naštete korake poenostavi z oblačno, v celoti upravljano storitvijo. Integriran je s **Confluent Schema Registry**, kar omogoča avtomatsko upravljanje in evolucijo shem brez prekinitve delovanja.

Optimizacijo datotek (združevanje majhnih datotek, odstranjevanje zastarelih segmentov) izvaja samodejno v ozadju, kar zagotavlja stalno visoko učinkovitost poizvedb. Podpira formate Iceberg in Delta ter omogoča *"Zero ETL"* pristop, kjer je isti nabor podatkov dostopen tako preko Kafka API kot tudi preko poizvedb SQL na materializirani tabeli.

Za boljši pregled nad razlikami med pristopi, ki omogočajo materializacijo podatkovnih tokov v analitične tabele, podajamo primerjalno analizo ključnih lastnosti in zmogljivosti posamezne rešitve v Tabeli 1.

Tabela 1: Primerjalna tabela pristopov materializacije pretočnih podatkov v tabelarno obliko.

Pristop / Lastnost	Kafka Connect Object-Storage Sink (S3/HDFS/ADLS/GCS)	Apache Hudi DeltaStreamer	Flink SQL + Iceberg/Delta Sink	Tableflow
Stopnja avtomatizacije	Srednja – brez avtomatskih metapodatkov	Srednja	Nizka	Visoka (*"Zero ETL"*)
Upravljanje shem	Omejeno / ročno	Osnovno podprto	Ročna/konfiguracijska	Avtomatsko (Schema Registry)
Podpora odprtim tabelam	Ne (le datoteke)	Da – Hudi	Da – Iceberg/Delta	Da – Iceberg/Delta
Optimizacija za poizvedbe	Ne	Delno/ročno	Delno/ročno	Da – samodejno
Kompleksnost	Srednja	Visoka	Visoka	Nizka
Prilagodljivost obdelave	Nizka	Srednja	Zelo visoka	Omejena (razširitev s Flink/Streams)

Čeprav **Kafka Streams** in **ksqlDB** niso neposredni konkurenti Tableflow, igrata pomembno vlogo kot komplementarni orodji v arhitekturah, kjer je potrebna kompleksna transformacija podatkov pred materializacijo. Primer smo predstavili tudi v prejšnjem poglavju. **Kafka Streams** omogoča popolnoma prilagojene procesne tokove na ravni kode, saj v lastni aplikaciji zapišeš logiko, ki prebere podatke iz ene ali več Kafka tem, jih transformira in nato rezultate pošlje v drugo Kafka temo. Od tam naprej lahko to temo preko povezoavlnika *sink* (npr. Kafka Connect S3/HDFS Sink) ali lastne kode zapisujemo v podatkovno jezero. Streams sam po sebi ne materializira podatkov v odprti tabelarični format (Iceberg, Delta) niti ne vzpostavi metapodatkov in optimizacij za poizvedbe, zato je ta korak treba izvesti z dodatnimi orodji ali kodo. **Tableflow** ta zadnji del – "**tema → optimizirana tabela v *odprtem tabelaričnem formatu***" – izvede avtomatsko in v oblaku, brez dodatnih korakov. **ksqlDB** pa ponuja preprost vmesnik SQL za filtriranje in agregacijo podatkov v realnem času ter se pogosto uporablja v scenarijih, kjer želimo hitro definirati transformacije brez razvoja lastne kode. V praksi se torej lahko uporabi kombinacija teh orodij s Tableflow, kjer Streams ali ksqlDB pripravita podatke, Tableflow pa jih nato samodejno materializira v odprtem tabelaričnem formatu.

7 Sklep

V prispevku predstavljamo novo rešitev Tableflow ekosistema Confluent, ki s svojimi inovacijami s tabelaričnim tokom omogoča neposredno predstavitev Kafka tem v odprtih tabelaričnih formatih Iceberg/Delta. Sledenje je ključno za integracijo pretočnih podatkov v sodobne arhitekture, temelječe na podatkovnih jezerih, saj odprti formati omogočajo, da lahko več različnih analiznih pogonov sočasno dostopajo do podatkovne tabele na objektni shrambi. S tem Tableflow omogoča, da podatkovna platforma učinkovito služi tako operativnim aplikacijam kakor analitičnim potrebam na skupnih podatkih v odprtem formatu.

Vpeljava tabelaričnih tokov rešitve Tableflow v podatkovno arhitekturo se odraža v številnih prednostih tako za podatkovne inženirje kot za podatkovne analitike, predvsem v manj kompleksni podatkovni arhitekturi, ki zmanjšuje potrebo po kompleksnih in k napakam nagnjenih procesih ETL, hitrejšem dostopu do aktualnih in relevantnih podatkov, pri čemer zaradi dualnosti tok-tabela ne prihaja do izgube konteksta in sheme podatkov ter enostavnosti integracije z obstoječimi orodji in okolji, pri čemer lahko isti nabor podatkov takoj izkoriščajo različna orodja znotraj različnih okolij, ki lahko dostopajo do podatkov brez posebnih prilagoditev. Podobno kot nas je evolucija sodobnih podatkovnih arhitektur pripeljala od jezer do kolišč – vse z namenom zmanjševanja kompleksnosti ter zbiranja številnih funkcionalnosti in paradigem pod eno streho – tako se tudi tabelarični tokovi ponujajo kot ena izmed takšnih evolucijskih korakov, ki pa se še morajo primerno uveljaviti in dokazati.

Literatura

[1] REIS Joe, HOUSLEY Matt. *Fundamentals of Data Engineering: Plan and Build Robust Data Systems*, O'Reilly Media, 2022.

[2] SERRA James. *Deciphering Data Architectures: Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh*, O'Reilly Media, 2024.

[3] MAZUMDAR Dipankar, GOVINDARAJAN Vinoth. *Engineering Lakehouses with Open Table Formats: Build scalable and efficient lakehouses with Apache Iceberg, Apache Hudi, and Delta Lake*, O'Reilly Media, 2025.

[4] ŠESTAK Martina, TURNAKOVIĆ Muhamed. "Nadgradnja podatkovnih jezer s pomočjo formata za shranjevanje Delta Lake", *OTS 2023 : Sodobne informacijske tehnologije in storitve : zbornik 26. konference*, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 2023, str. 243–254.

[5] ŠESTAK Martina. "Prehod na realno-časovno obdelavo podatkovnih tokov s pomočjo Kafka Streams in KSQL/ksqlDB", *OTS 2022 : Sodobne informacijske tehnologije in storitve : zbornik petindvajsete konference*, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 2022, str. 118–127.

[6] SELWAN Marc. https://www.confluent.io/blog/introducing-tableflow, *Introducing Tableflow*, 2024, obiskano 21. 7. 2025.

[7] Confluent, Inc. https://docs.confluent.io/cloud/current/topics/tableflow, *Tableflow Documentation*, 2025, obiskano 21. 7. 2025.

[8] BELLEMARE Adam. *Shift Left: Unifying Operations and Analytics With Data Products*.

[9] ČUŠ Blaž, GOLEC Darko, STRUGAR Ivan. "Data Lakehouse: Benefits in small and medium enterprises", *Journal of Innovative Business and Management*, letnik 14, številka 2, 2023, str. 1–10.

[10] ARMBRUST Michael, DAS Tathagata, SUN Liwen, YAVUZ Burak, ZHU Shixiong, MURTHY Mukul, TORRES Joseph, VAN HOVELL Herman, IONESCU Adrian, ŁUSZCZAK Alicja, ŚWITAKOWSKI Michał, SZAFAŃSKI Michał, LI Xiao, UESHIN Takuya, MOKHTAR Mostafa, BONCZ Peter, GHODSI Ali, PARANJPYE Sameer, SENSTER Pieter, XIN Reynold, ZAHARIA Matei. "Delta Lake: High-performance ACID table storage over cloud object stores", *Proceedings of the VLDB Endowment*, letnik 13, številka 12, 2020, str. 3411–3424.

[11] BUKTA Victoria, INDRASIRI Kasun. https://www.youtube.com/watch?v=js2vL5TNHB0, *Streaming Meets Governance: Building AI-Ready Tables With Confluent Tableflow and Unity Catalog*, obiskano 21. 7. 2025.