Prehod na realno-časovno obdelavo podatkovnih tokov s pomočjo Kafka Streams in KSQL/ksqlDB

Martina Šestak

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
martina.sestak@um.si

Sinopsis V sodobnih arhitekturah IKT rešitev se pogosto srečamo z obdelavo dogodkov, ki jih proizvajajo poslovni procesi podjetij. Potreba po takojšnji obdelavi posameznih dogodkov oz. podatkovnih tokov (angl. stream), ki jih le-ti tvorijo, namesto paketov (angl. batch) je zadnjih nekaj let v ospredje postavila rešitve za pretočno obdelavo podatkov. Ena izmed pogosto uporabljanih rešitev za pretočno obdelavo podatkov je platforma Apache Kafka, ki ponuja širok nabor orodij in rešitev za realno-časovno obdelavo podatkov. V prispevku bomo podrobneje predstavili platformo Kafka iz vidika realno-časovne pretočne obdelave podatkov. Predstavili bomo razlike med rešitvama Kafka Streams in KSQL/ksqlDB ter njuno umeščenost v ekosistem Kafka oz. v ekosistem IKT rešitev, ki uporabljajo Kafko za dogodkovno orientirano obdelavo podatkov. Izpostavili bomo ključne prednosti Kafke pred klasičnimi sporočilnimi sistemi, kot je npr. RabbitMQ. Pri tem bomo poskusili biti objektivni ter v razpravo vključili tudi morebitne konkurenčne sisteme, kot je Apache Flink s svojo nadgradnjo Flink SQL. S tehničnega vidika pa se bomo predvsem osredotočili na KSQL oz. ksqlDB. Opisali bomo podobnosti in razlike s klasičnim poizvedovanjem po podatkovnih bazah. Dotaknili se bomo tudi scenarijev povezovanja tabel s tokovi ter kompleksnejših primerov in izzivov uporabe KSQL/ksqlDB pri realno-časovni obdelavi podatkov.

Ključne besede:
podatkovni toki
realno-časovna obdelava podatkovnih tokov
pretočna obdelava
Kafka
Kafka Streams
KSQL
ksqlDB

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.10

1 Uvod

Dogodkovno orientirana miselnost procesiranja sveta okrog nas postavlja v ospredje dogodke (angl. event) kot atomarno osnovo obdelave podatkov v modernih IKT rešitvah. Dogodke lahko tako metaforično predstavimo kot posamične atome vode, ki skupaj tvorijo potok, ki ima svoj izvir in po navadi tudi ponor, pri čemer lahko trenuten način obdelave teh dogodkov enačimo z naključnim zajemom vode v dlan in požirkom med sprehodom po gozdu. Platforma Kafka se je že vrsto let nazaj izkazala kot odlična osnova za pretočno obdelavo podatkov, pri čemer lahko le to, s svojo osnovno gručo Kafka, metaforično predstavimo kot vodno strugo.

V trenutnih IKT rešitvah in s tem povezanih arhitekturah IT/IS se že opaža pogostejša uporaba Kafke, vendar z osredotočanjem na razvoj Kafka proizvajalcev (angl. producer) in potrošnikov (angl. consumer) ter uporabo Kafke kot sporočilnega sistema. Prihajamo pa vedno bolj do situacij, kjer želimo dogodke neprekinjeno in v celoti spremljati in ne zgolj sporadično. Na primeru našega potoka, si zahtevo lahko predstavimo kot nenadno zastrupitev vode, kjer je potrebno neprekinjeno in detajlno spremljati celoten tok na določeni točki in iskati morebitne škodljive primesi. Za ta namen je nujno potrebno začeti Kafko uporabljati kot platformo za realno-časovno obdelavo podatkovnih tokov, in sicer z uporabo komponent, kot sta Kafka Streams in/ali KSQL oz. ksqlDB. Prva komponenta ponavadi zahteva višji nivo programerskih veščin, vendar omogoča izvedbo bolj kompleksnih analiz nad podatkovnimi toki. Slednja komponenta pa je bolj prijazna do končnega uporabnika, saj omogoča izvedbo povpraševanj s pomočjo sintakse, ki temelji na poizvedovalnem jeziku SQL in pri uporabniku spodbuja občutek, kot da je v interakciji s klasično podatkovno bazo.

V nadaljevanju bomo predstavili osnovne lastnosti pretočne obdelave podatkov in platforme Kafka. Podrobno bomo predstavili komponenti Kafka Streams in KSQL/ksqlDB ter razlike v njuni uporabi, ki lahko vplivajo na izbiro ene izmed teh dveh rešitev za realno-časovno obdelavo podatkovnih tokov. Slednji komponenti ekosistema Kafka bomo predstavili tudi na praktičnem primeru, s pomočjo katerega bomo potem argumentirali izzive in možnosti njune uporabe.

2 Pretočna obdelava podatkov

Izbira načina obdelave domenskih podatkov direktno vpliva na zmogljivost in možnosti analiz, ki jih lahko izvajamo nad zbranimi podatki. Podjetja, ki želijo imeti vpogled v podatke takoj ko le-tisti pridejo v sistem ker jim hitra analiza omogoča, da iz podatkov izvlečejo največjo vrednost za njihovo poslovanje, se pogosto odločajo za pretočno obdelavo podatkov. Obstajajo pa še vedno primeri "tradicionalnega" pristopa k obdelavi podatkov, kjer se podatki v sistemu analizirajo v množicah, ki jim rečemo "paketi" (angl. batches).

Pretočna obdelava podatkov temelji na tehnologiji "pretakanja" (angl. streaming) podatkov. V tem primeru, sistem, ki omogoča pretakanje podatkov, vsebuje stroj za obdelavo podatkov, ki je načrtovan z neskončnimi nabori podatkov v mislih [1]. Pri paketni obdelavi je stroj za povpraševanje (angl. query engine) naravnan tako, da periodično izvede povpraševanja nad končnim naborom podatkov, ki so v sistemu zabeleženi od časa zadnjega povpraševanja, medtem ko ta stroj neprekinjeno izvaja povpraševanja nad neskončnimi podatkovnimi toki pri pretočni obdelavi.

Kot največja prednost pretočne obdelave podatke se v literaturi izpostavlja možnost obdelave podatkov z zelo nizko zakasnitvijo (angl. latency) [2]. Ravno je zakasnitev najbolj relevantno merilo za določanje zmogljivosti sistema za realno-časovno obdelavo podatkov. Namreč, ponudniki rešitev za pretočno obdelavo podatkov pogosto trdijo, da omogočajo realno-časovno obdelavo podatkov, vendar temu v praksi ni vedno tako. Glede na vpliv zakasnitve na IKT sistem znotraj katerega se izvaja, razlikujemo tri pristopa za realno-časovno obdelavo podatkov [3]:

- Trda realno-časovno obdelava podatkov (angl. hard real-time) – zakasnitev se meri v milisekundah in sistem ima nulto toleranco na zakasnitev, saj so posledice lahko katastrofalne (npr. zavorni sistem v avtomobilu),
- Mehka realno-časovna obdelava podatkov (angl. soft real-time) – zakasnitev se meri v sekundah in nima katastrofalnega učinka na sistem, vendar vpliva na učinkovitost sistema (angl. sistem za trgovanje na borzah), in
- Obdelava podatkov skoraj v realnem času (angl. near real-time) – zakasnitev se meri v minutah in ne povzroča večje škode na delovanje sistema (npr. sistemi za poročanja).

V naslednjem poglavju bomo predstavili platformo Kafka ter analizirali možnosti realno-časovne obdelave podatkov, ki jih ta platforma ponuja.

3 Platforma Kafka

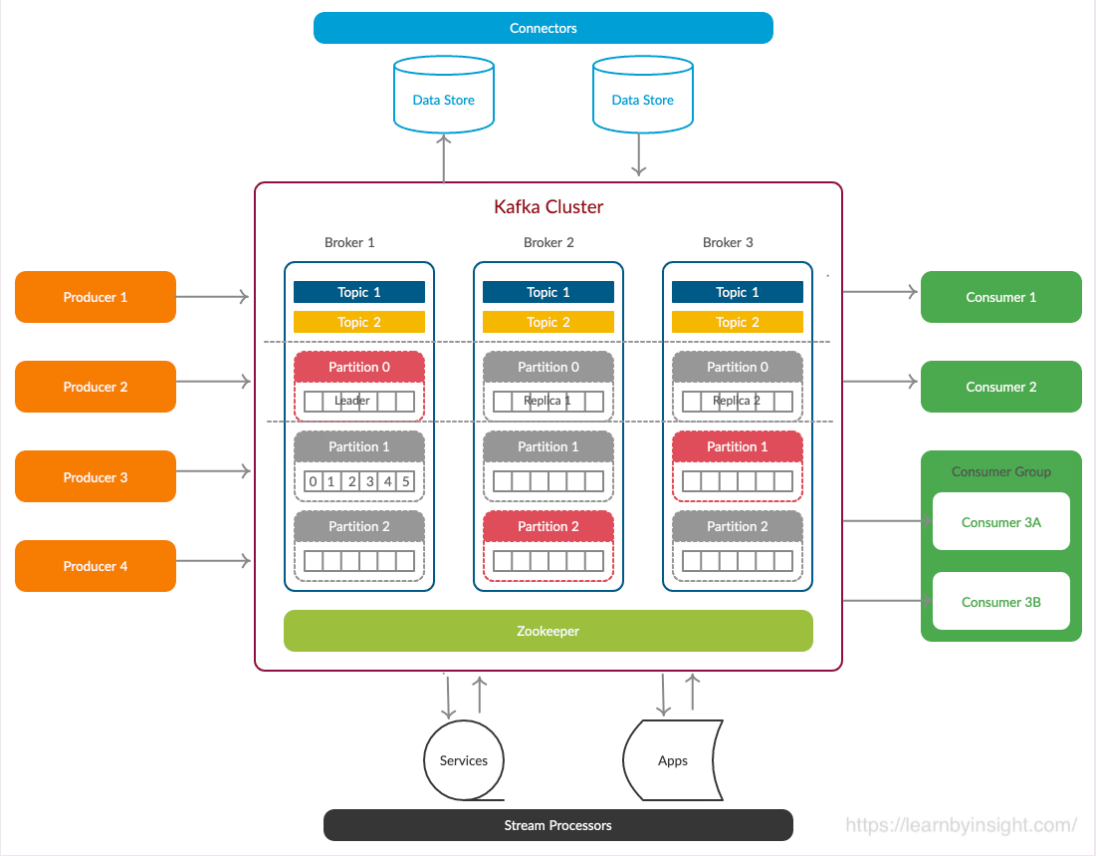
3.1 Osnovni koncepti in arhitektura

V literaturi se pogosto srečamo z definicijo platforme Kafka kot sporočilnega sistema, ki je odgovoren za prenos podatkov iz ene aplikacije v drugo. Danes se to sporočanje s pomočjo Kafke dosti krat izvaja v porazdeljenem okolju, vendar Kafka še zmeraj zagotavlja realno-časovno obdelavo podatkov ob nizkih zakasnitvah in visoki prepustnosti. V nadaljevanju bomo na kratko predstavili arhitekturo platforme Kafka, ki je tudi prikazana na Sliki 1. Sporočila so v Kafki razvrščena v *teme* (angl. topic), ki predstavljajo tok (angl. stream) podatkovnih zapisov, ki so kategorizirane v skup. Vsak tok podatkov je neomejen in kontinuiran pretok podatkov v realnem času. Teme definirajo razvijalci preden ko začnejo pošiljati sporočila v sistem. Najbližja analogija teme je tabela v podatkovnih bazah. Vsaka Kafka tema je razdeljena v *particije* (angl. partition), ki predstavljajo dejansko enoto za shranjevanje podatkovnih zapisov v obliki dnevniške datoteke (angl. log file), medtem ko tema predstavlja le logični koncept v ekosistemu Kafka. Teme so shranjene na strežnikih, ki se imenujejo *posredniki* (angl. brokers) in tvorijo Kafka gručo s katero upravlja Zookeeper in znotraj katere se podatki shranjujejo v porazdeljeni obliki (particije na različnih posrednikih).

Primarna vloga tem v Kafki je zmanjšanje odvisnosti med *proizvajalci* (angl. producers) in *potrošniki* (angl. consumers) [5]. To pomeni, da s tem lahko zagotavljamo, da npr. počasni potrošniki ne vplivajo na proizvajalce, dodajanje potrošnikov ali napake potrošnikov ne vplivata na ekosistem, ali da se potrošniki lahko razširijo brez vpliva na ekosistem. V ozadju, Kafka sporočilni sistem sledi vzorcu *objava-naročnina* (angl. publish-subscribe), pri čemer je ena aplikacija *proizvajalec*, ki pošilja sporočila v določeno Kafka temo. Z druge strani, ko se naroči na to isto Kafka temo, druga aplikacija, ki se ji reče *potrošnik,* sproti sledi prispelim zapisom v to Kafka temo in je v realnem času o tem obveščena. Proizvajalci in potrošniki so instance programske kode, ki jih razvijamo po želji glede na poslovno logiko. Kafka je s stališča proizvajalcev in potrošnikov agnostična, kar pomeni, da je neodvisna od platforme in programskega jezika v katerem se le-ti implementirajo. Za namene prebiranja novih sporočil, potrošniki hranijo zadnji odmik (angl. offset) na podlagi katerega, posredniki vedo, katera nova sporočila potrošnik še ni prebral oz. obdelal. Potrošniki lahko tudi tvorijo potrošniške skupine (angl. consumer groups), ki predstavljajo eno končno aplikacijo/sistem. Ko jih preberejo, potrošniki procesirajo prispele podatkovne zapise kot pare ključ-vrednost (angl. key-value pairs), kar pomeni, da je za namene obdelave potrebno pretvoriti vsaki podatkovni zapis iz polja bajtov (angl. byte array) v par ključ-vrednost.

Funkcionalnosti platforme Kafka danes na trgu ponuja več ponudnikov. Pri tem je večina ponujenih rešitev nastala na podlagi implementacije odprtokodne platforme Apache Kafka in se majhne razlike izmed ponudnikov pojavljajo predvsem pri implementaciji določenih funkcionalnosti platforme zaradi različnih namenov (pretakanje, analiza, infrastruktura ali nekaj tretjega). Razen Apache Kafka, najbolj znane variante različnih ponudnikov platforme Kafka so Confluent Kafka, Cloudera Hortonworks Kafka, Red Hat Kafka in Amazon MSK. V tem

prispevku bomo praktične primere implementirali s pomočjo platforme *Confluent Kafka*, ki se osredotoča na pretakanje dogodkov in vključuje dodatne možnosti za shrambo in obdelavo podatkov.

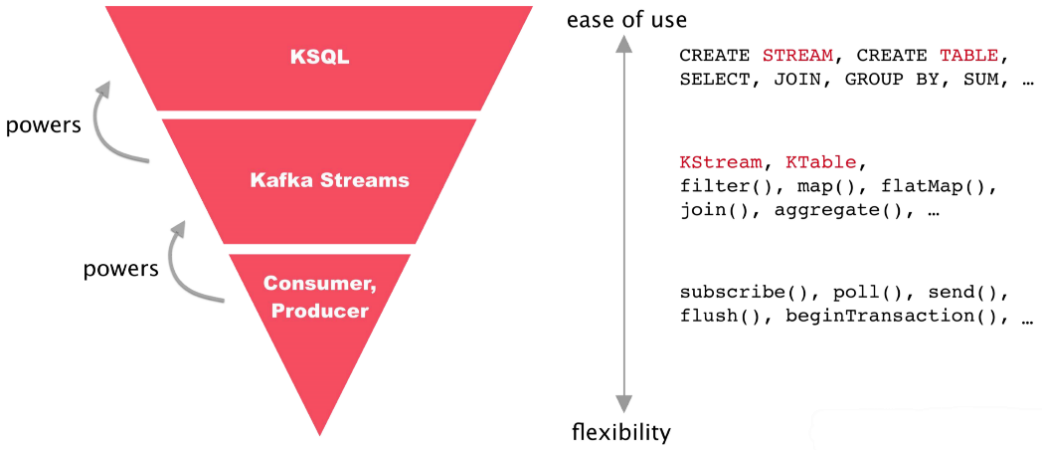


Slika 1: Arhitektura platforme Kafka.
Vir: [4].

V osnovi, platforma Kafka vključuje pet osrednjih programskih vmesnikov, ki omogočajo enostavnejšo upravljanje in uporabo Kafka ekosistema. Obdelava podatkov v realnem času, in sicer neprekinjeno, sočasno in na način, da se obdeluje vsak posamezen podatkovni zapis (dogodek), se izvaja s pomočjo *Kafka Streams API* ali s pomočjo *KSQL/ksqlDB*. Pri tem realno-časovna obdelava običajno vključuje branje neprekinjeno prihajajočih podatkovnih zapisov določenega toka/teme, izvajanje nekaterih analiz ali preoblikovanja podatkov in nato zapisovanje rezultatov tega v drug ali nov tok/temo.

Za izvedbo realno-časovne obdelave s platformo Kafka lahko uberemo več načinov:

- Klasičen način – vključuje razvoj potrošnika, ki podatke obdela in jih nato s pomočjo proizvajalca pošiljamo naprej v nov tok. Takšen način je zelo zahteven, saj je za namene stanovitne obdelave potrebno veliko namenskega procesiranja.
- Način kjer uporabimo polnopravni okvir za obdelavo realno-časovnih tokov (angl. full-fledged stream processing framework), kot so Spark Streaming, Flink, Storm itd.
- Domorodni način za Kafko - vključuje uporabo Kafka Streams API ali KSQL.



Slika 2: Primerjava med različnimi možnostmi platforme Kafka za realno-časovno obdelavo podatkov. Vir: [6].

V odvisnosti od poslovnih potreb naše aplikacije lahko uporabimo bodisi preproste Kafka programske elemente, kot so proizvajalci in potrošniki, ali pa preidemo na uporabo Streams API ali KSQL, ki nam na različnih nivojih abstrkacije pomagajo pri realno-časovni obdelavi podatkov (Slika 2). V nadaljevanju bomo prikazali domorodni način za realno-časovno obdelavo, in sicer z uporabo Kafka Streams API-ja ter KSQL/ksqlDB.

3.2 Kafka Streams

Kafka Streams je osrednja komponenta, ki omogoča realno-časovno obdelavo pretakajočih se podatkov. Gre se za odjemalsko knjižnico za gradnjo aplikacij in mikrostoritev, kjer so vhodni in izhodni podatki shranjeni v Kafka posrednikih. Takšne programske nadgradnje omogočajo višji nivo abstrakcije ter transformacijo in obogatenje podatkov med pretakanjem. Apache Kafka Streams se kot knjižnica izvaja v sklopu aplikacije in ne znotraj posrednikov.

Med ostalim, Apache Kafka Streams omogoča [7]:

- Procesiranje vsakega posameznega podatkovnega zapisa med pretakanjem z milisekundno zakasnitvijo,
- Grupiranje toka po ključu,
- Združevanje več tokov,
- Neprekinjeno transformacijo,
- Nestanovitno (angl. stateless) in stanovitno (angl. stateful) obdelavo glede na to, ali se nov podatkovni tok procesira neodvisno od prejšnjih tokov ali ne,
- Filtriranje, mapiranje, združevanja, agregacije podatkov,
- Časovno obdelavo (angl. windowing).

Znotraj Kafka Streams aplikacije za obdelavo se vsak korak obdelave izvaja po določenem vrstnem redu, ki sestavlja *topologijo* procesorja. Topologija procesorja definira logiko obdelave podatkov, ki jo mora izvesti aplikacija za obdelavo tokov (angl. streams application). V topologiji procesorja pretoka obstaja vozlišče, ki ga imenujemo procesor toka, ki predstavlja korak obdelave za pretvorbo podatkov v tokove s sprejemanjem enega vhodnega podatkovnega zapisa od svojih procesorjev v topologiji navzgor. Prav tako naknadno izdela enega ali več izhodnih zapisov za svoje nadaljnje procesorje.

Kafka Streams API ponuja podporo obdelave tokov kakor tabel, in sicer s pomočjo svojih osnovnih abstrakcij *KStream* in *KTable*. KStream je objekt, ki se uporablja za abstrakcijo toka (teme), s katerim lahko izvajamo določene

operacije, kot sta filtriranje in mapiranje (nestanovitni operaciji). Uporabljamo ga, kadar na dogodke v toku gledamo kot neodvisna nespremenljiva dejstva, ki se konstantno posodabljajo (npr. beleženje dogodkov kot so gol, podaja, kot na tekmi). Dogodki se konstantno dogajajo, pri čemer nov dogodek ne prepiše prejšnjega. KTable je abstrakcija toka dnevnika sprememb, kjer vsak podatkovni zapis predstavlja posodobitev. Natančneje, vrednost v podatkovnem zapisu se razlaga kot posodobitev "UPDATE" zadnje vrednosti za isti ključ zapisa, če ustrezen ključ še ne obstaja, se posodobitev šteje kot vnos "INSERT". V primerjavi s tokom dogodkov tabela predstavlja stanje sveta v določenem trenutku, običajno najbolj sveže stanje (npr. statistika nogometne tekme). Tabela je pogled toka dogodkov in ta pogled se nenehno posodablja, ko je zajet nov dogodek. Nad KTable objektom izvajamo določene operacije, kot sta združevanje (angl. join) in agregiranje (stanovitni operaciji), pri čemer ustvarjamo t. i. obogatene tokove (angl. enriched streams).

3.3 KSQL/ksqlDB

KSQL je zgrajen na Kafka Streams in predstavlja robusten okvir za obdelavo tokov, ki so del Kafke. Ponuja nam poizvedbeni sloj za izdelavo aplikacij za pretakanje dogodkov na Kafkine teme. Medtem ko Kafka Streams omogoča pisanje nekaterih zapletenih topologij in za to zahteva nekaj bistvenega znanja programiranja, želi KSQL to zapletenost abstrahirati tako, da nam zagotovi vsem znano semantiko SQL, pri čemer ne smemo pozabiti, da ne govorimo o paketnem SQL-u, temveč pretočnem SQL-u, tj. izvedbi povpraševanj med pretakanjem podatkov. Podobno kot Streams API, KSQL omogoča višji nivo abstrakcije ter transformacijo in obogatenje podatkov med pretakanjem. Kot naslednik KSQL-a je nastal ksqlDB in se danes ta dva koncepta oz. rešitvi pogosto uporabljata izmenično.

KSQL oz. ksqlDB lahko uporabljamo za številne operacije nad tokovi, kot so [8]:

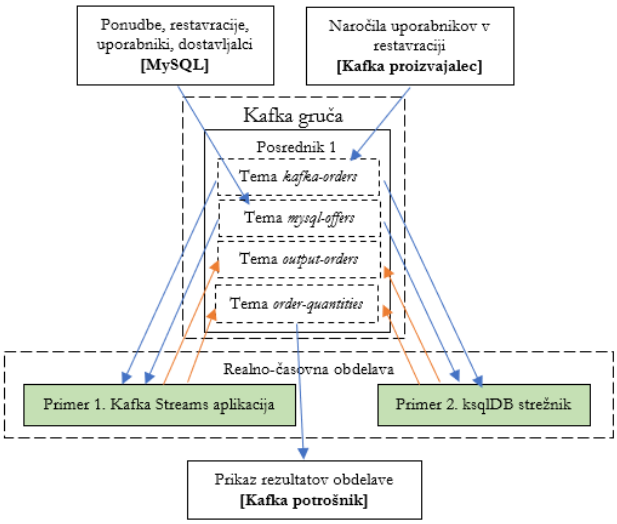
- Raziskovanje podatkov shranjenih v Kafka temi,
- Pretočni ETL (angl. Extract-Transform-Load),
- Zaznavanje anomalij,
- Realno-časovno spremljanje dogodkov.

V razliko od Streams API, ki se izvaja v sklopu naše aplikacije, se ksqlDB izvaja kot podatkovna baza, ki se paralelno namesti ob Kafka gruči in je z njo v tesni povezavi. Ker se ksqlDB samodejno izvaja na platformi Apache Kafka, je potrebna torej prvo namestitev Kafke, ki je konfigurirana za uporabo ksqlDB.

4 Realno-časovna obdelava podatkovnih tokov na praktičnem primeru

4.1 Opis primera

V praktičnem primeru smo vzpostavili arhitekturo s pomočjo platforme Kafka, ki omogoča neprekinjeno branje naročil v restavracijah, ki jo preko spletne strani lahko oddajo uporabniki (Slika 3). Naročila kot podatkovne toke vsakih deset sekund v temo *kafka-orders* pošilja Kafka proizvajalec implementiran v programskem jeziku Java, medtem ko se s pomočjo knjižnice Kafka Connect v teme s prefiksom *mysql-* (npr. *mysql-offers*) uvezejo domenski podatki o ponudbah, uporabnikih, restavracijah, dostavljalcih, ki so shranjeni v bazi MySQL. Naslednji korak je združena obdelava domenskih podatkov in podatkovnih tokov ter prikaz rezultatov. Za prvo varianto obdelave smo implementirali zasebno Kafka Streams aplikacijo, pri drugi pa je za ta namen izvedeno nekaj povpraševanj nad ksqlDB bazo s pomočjo sintakse KSQL. Za prikaz rezultatov smo razvili preprostega Kafka potrošnika.



Slika 3: Arhitektura praktičnega primera za realno-časovno obdelavo naročil v restavracijah.
Vir: lasten.

4.2 Primer uporabe Kafka Streams

Kot rečeno v predhodnem poglavju je za realno-časovno obdelavo s pomočjo Kafka Streams potrebno implementirati aplikacijo, ki jo zaženemo v sistemu. Znotraj aplikacije implementirati Streams procesorja, ki bere podatkovne toke iz ustreznih Kafka tem, jih shrani kot objekte KStream/KTable odvisno od potrebnih analiz, procesira podatke in rezultate ponovno shrani v ciljno Kafka temo. Pogoj za uspešno delovanje je, da so Kafka teme, ki se uporabljajo, predhodno ustvarjene na posrednikih. V našem primeru Streams procesor izvaja topologijo, ki je sestavljena iz dveh podtopologij znotraj katerih se izvajajo določeni koraki obdelave. Izsek programske kode za vsako podtopologijo je prikazan na Sliki 4. Prva podtopologija prebere podatkovne zapise iz teme *kafka-orders*, jih shrani kot objekt KStream (ker želimo zgodovinsko beležiti vsa prispela naročila), pri čemer bo ključ tega objekta identifikator uporabnika (tipa *Integer*), zabeležena vrednost pa predstavlja številko naročila (tipa *Order*). V naslednjem koraku topologije procesorsko vozlišče grupira posamezna naročila po uporabnikih, jih potem prešteje in v izhodno temo *output-orders* shrani število oddanih naročil za posamezne uporabnike. V drugi podtopologiji pa se seštevek količine naročenih izdelkov v vsakem naročilu shrani v temo *order-quantities*.

```
KStream<Integer, Order> inputStream = streamsBuilder.stream(props.getProperty("input.topic.name"), Consumed.with(Serdes.Integer(), serdes));

inputStream.map((k,v)->new KeyValue<>(v.getUser_id().intValue(),v.getOrder_no()))
        .groupByKey(Grouped.with(Serdes.Integer(), Serdes.String())).count().toStream().mapValues(value -> value.toString())
        .to( s: "output-orders", Produced.with(Serdes.Integer(), Serdes.String()));
inputStream
    .map((k,v)->new KeyValue<>(v.getOrder_no(),v.getItems())) KStream<String, Map<String, Integer>>
    .flatMapValues(value -> Arrays.asList(value.get("quantity"))) KStream<String, Integer>
    .groupByKey(Grouped.with(Serdes.String(),Serdes.Integer())) KGroupedStream<String, Integer>
    .reduce(Integer::sum) KTable<String, Integer>
    .toStream() KStream<String, Integer>
    .mapValues(v -> v.toString()) KStream<String, String>
    .to( s: "order-quantities", Produced.with(Serdes.String(), Serdes.String()));
```

Slika 4: Izsek programske kode Kafka Streams aplikacije.
Vir: lasten.

4.3 Primer uporabe KSQL/ksqlDB

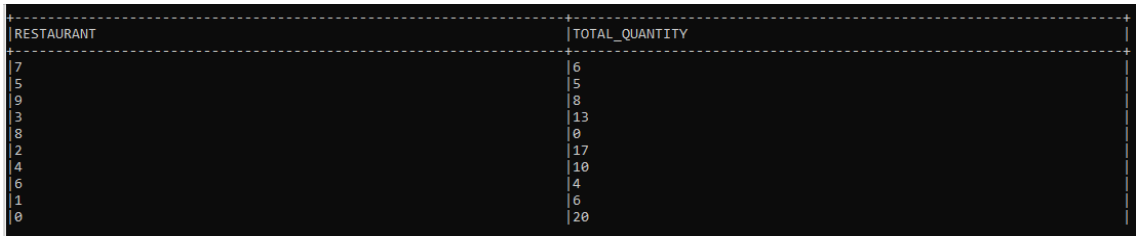
Pri KSQL/ksqlDB načinu obdelave ne potrebujemo programsko implementirati aplikacije, ampak se samo povežemo na ksqlDB strežnik. Podobno kot pri klasičnih podatkovnih bazah je tukaj prvi korak pripraviti objekte v katere bomo brali zapise iz podatkovnih tokov. Ustvarimo lahko podatkovni tok (z ukazom CREATE STREAM) ali tabelo (z ukazom CREATE TABLE). Obadva objekta sta analogija objektom KStream oz. KTable pri uporabi

Kafks Streams in za vsakega obstaja nabor primerov, ko je njegova uporaba najbolj smiselna. Sintaksa KSQL je zelo podobna sintaksi SQL-a, pri čemer je edino pomembno upoštevati podatkovno strukturo zapisov v izhodni Kafka temi, da bi jih pravilno prebrali. Kot primer lahko ustvarimo podatkovni tok za branje vseh naročil iz teme *kafka-orders* s pomočjo ukaza kjer vir podatkov opredelimo s ključno besedo *WITH*:

create stream kafka_sorders (order_no varchar, date bigint, rest_id integer, user_id integer, courier_id integer, item_id integer, quantity integer) with (kafka_topic='kafka-orders', key_format='kafka', value_format='avro');

Ko smo ustvarili podatkovni tok *kafka_sorders*, iz njega lahko izpišemo trenutne zapise s pomočjo ukaza SELECT, vendar se bo tisti izpis neprekinjeno posodabljal z vsako novo vrstico oz. objektom, ki ga preberemo iz Kafka teme. Kot naslednji korak lahko podatke shranjene v toku *kafka_sorders* ponovno združimo z domenskimi podatki o restavracijah (za katere smo predhodno tudi ustvarili podatkovni tok *kafka_srestaurants*, ki bere zapise iz teme *mysql-restaurants*) in v realnem času računamo seštevek količine naročenih izdelkov za posamezno restavracijo.

Za ta namen ustvarimo ksqlDB tabelo, kam bomo zabeležili edino zadnje stanje naročenih količin za posamezno restavracijo, pri čemer bo posamezna vrstica v tabeli rezultat združevanja tokov *kafka_sorders* in *kafka_srestaurants* po stolpcu *rest_id* v določenem časovnem okvirju (zadnja ura, mesec, ipd.). Kot rezultat obdelave se potem izpiše trenutni seštevek naročenih količin v posamezni restavraciji (Slika 5), vendar se tudi ta vpogled spreminja v realnem času, glede na to prispela naročila z novimi količinami izdelkov.



RESTAURANT	TOTAL_QUANTITY
7	6
5	5
9	8
3	13
8	0
2	17
4	10
6	4
1	6
0	20

Slika 5: Prikaz obdelave naročenih količin v posameznih restavracijah s pomočjo KSQL/ksqlDB. Vir: lasten.

5 Diskusija

V primerjavi s klasičnimi sporočilnimi sistemi, kot je RabbitMQ, je Kafka veliko bolj učinkovita pri uporabi sistemskih resursov. Kafka temelji na t. i. *povleci* (angl. pull) modelu sporočanja, pri katerem posredniki pošljejo potrošnikom nova sporočila šele na njihovo zahtevo in je zaradi tega možen scenarij, da določeni potrošnik v sistemu ne razpolaga z najnovejšimi podatki. Z uporabo *povleci* modela in razvrščanja sporočil (angl. message ordering) Kafka lahko doseže boljšo prepustnost v sistemu, saj je manjša verjetnost pojave zastojev pri prejemanju sporočil. V razliko od *povleci* modela pri Kafki, RabbitMQ in večina podobnih sporočilnih sistemov temeljijo na t. i. *potisni* (angl. push) modelu sporočanja, pri katerem posredniki obvestijo naročene potrošnike o novem sporočilu, takoj ko ga proizvajalec pošlje v sistem, s čimer se lahko doseže večja hitrost pri pošiljanju sporočil. Kafka je zaradi podpore za atomarnost pošiljanja in branja vsakega sporočila priporočena rešitev za podjetja, kjer se že uporabljajo relacijske podatkovne baze in je pomembno zagotavljati konsistentnost podatkov.

S pomočjo komponente Kafka Streams, platforma Kafka omogoča razvoj mikrostoritev, s čimer se poveča prilagodljivost sistema za različne scenarije in analize podatkov, ki jih uporabniki potrebujejo. Namen te komponente je kombinirati možnost realno-časovne obdelave podatkovnih tokov v rangu milisekund čez izvajanje klasičnih Java/Scala aplikacij znotraj Kafka gruče. Kot največji izziv uporabe Kafka Streams, uporabniki poudarjajo potrebo po dobrem (celo naprednem) poznavanju programiranja zaradi razvoja Streams aplikacije. Uporaba knjižnice Kafka Streams za programerje pomeni, da lahko obdelavo podatkovnih tokov implementirajo v obliki programske kode, ki se izvaja znotraj že obstoječe aplikacije in se izvaja neposredno znotraj Kafka gruče, kar

poenostavlja interakcijo v sistemu. Kar se tiče ponujenih možnosti pri obdelavi pa Kafka Streams podpira nestanovitno in stanovitno obdelavo podatkovnih tokov, kar ustvarja možnost za napredne analize in agregacije nad podatki. Omogoča tudi izvedbo časovne obdelave podatkovnih tokov, kar je danes zelo pomembno za množična podjetja, pri katerih je časovna komponenta dogodkov, ki jih beležijo, podlaga za poslovne odločitve (npr. v domeni Industrije 4.0/5.0).

KSQL oz. ksqlDB sta nastali z namenom izboljšanja uporabniške izkušnje pri obdelavi podatkovnih tokov, ki pridejo v sistem. Sintaksa KSQL/ksqlDB temelji na klasični sintaksi jezika SQL z nekaj prilagoditev, ki so potrebne zaradi pretočne obdelave podatkov. Čeprav je veliko manj zahtevna za zagon v Kafka gruči, komponenta KSQL/ksqlDB zahteva določanje sheme podatkovnega toka preden začne brati zapise iz toka. Slednja lastnost spominja na zmanjšano prilagodljivost pri spremembah v shemi podatkovnih zapisov, ki že vrsto let predstavlja eno izmed glavnih pomanjkljivosti relacijskih podatkovnih baz. Namreč, v današnjih poslovnih okoljih lahko prihaja do spremembe v strukturi podatkov, ki jih je potrebno obdelati in shraniti v sistemu. Vsaka takšna sprememba v shemi podatkovnega toka zahteva ponovno ustvarjanje novih podatkovnih struktur v ksqlDB, saj drugače lahko pride do neujemanja starih in novih zapisov ter posledično napak pri izvajanju povpraševanj.

Platforma Kafka je danes postala sopomenka za pretakanje podatkov in realno-časovno obdelavo podatkovnih tokov, vendar ne gre za univerzalno tehnološko rešitev, ki je najbolj primerna za uporabo v vseh scenarijih, saj tudi Kafka ima svoje slabosti in ne podpira vsega, kar je danes podjetjem potrebno. Na trgu tehnologij za obdelavo podatkovnih tokov se uporabljajo tudi rešitve, kot so Flink (podpira tudi paketno obdelavo), Storm (izjemno hiter za preproste scenarije) ter Spark Streaming (podpira lambda arhitekturo in zagotavja odpornost na izpade). V primerjavi z naštetimi rešitvami sta komponenti Kafka Streams in KSQL/ksqlDB nekaj novejše, bolj vezane za uporabo Kafke v sistemu in bolj primerne za enostavnejše scenarije, vendar kažejo prednosti Kafke v obdelavi podatkovnih tokov in omogočajo razvoj mikrostoritev za obdelavo podatkovnih tokov v obstoječih IKT sistemih.

6 Zaključek

V prispevku smo predstavili razliko med pretočno in paketno obdelavo podatkov, ki danes postaja vedno bolj očitna in pomembna pri izbiri ustreznih tehnoloških rešitev. Predstavili smo platformo Kafka za pretočno obdelavo podatkov ter osnovne komponente arhitekture platforme. Osredotočili smo se na proces obdelave podatkov v realnem času, ki ga ta platforma zagotavlja čez več načinov. Za namen preverjanja uporabnosti v realnem okolju smo izbrali obdelavo podatkov s pomočjo dveh rešitev, in sicer knjižnice Kafka Streams ter KSQL oz. ksqlDB. Razliko v pristopu pri obdelavi ter kompleksnost obdelave s posamezno rešitvijo smo prikazali tudi na praktičnem primeru realno-časovne obdelave naročil v restavracijah. Prikazali smo, da, čeprav obedve rešitvi uporabljata podobne podatkovne strukture pri obdelavi podatkov (podatkovni toki ali tabele), možnost obdelave z uporabo sintakse, ki temelji na poizvedovalnem jeziku SQL, pri rešitvi KSQL/ksqlDB bistveno olajša celoten proces za končnega uporabnika, vendar je nabor možnih analiz bolj omejen kot pri Kafka Streams. V prihodnosti načrtujemo izvedbo celovite primerjave učinkovitosti obeh rešitev za realno-časovno obdelavo ter vzpostavljanje ustreznih meril za primerjavo različnih rešitev na trgu, ki se trenutno uporabljajo za ta namen.

Literatura

[1] AKIDAU, Tyler, CHERNYAK, Slava, LAX, Reuven, Streaming systems: the what, where, when, and how of large-scale data processing, O'Reilly Media, Inc., 2018.

[2] FRIEDMAN, Ellen, TZOUMAS, Kostas, Introduction to Apache Flink: stream processing for real time and beyond, O'Reilly Media, Inc., 2016.

[3] What is the difference between real-time and streaming analytics?, https://www.bizdata.com.au/blogpost.php?p=real-time-vs-streaming-analytics, obiskano 1.8.2022.

[4] Kafka Clusters Architecture 101: A Comprehensive Guide, https://hevodata.com/learn/kafka-clusters/, obiskano 27. 7. 2022.

[5] ESTRADA, Raul, Apache Kafka 1.0 Cookbook: Over 100 practical recipes on using distributed enterprise messaging to handle real-time data, Packt Publishing Ltd., 2017.

[6] KSQL and Kafka Streams, https://docs.confluent.io/5.4.2/ksql/docs/concepts/ksql-and-kafka-streams.html, obiskano 27. 7. 2022.

[7] Kafka Streams – Core concepts, https://kafka.apache.org/32/documentation/streams/core-concepts, obiskano 2. 8. 2022.

[8] Introducing KSQL: Streaming SQL for Apache Kafka, https://www.confluent.io/blog/ksql-streaming-sql-for-apache-kafka/, obiskano 2. 8. 2022.