Podatki v utripu srca: podatkovna platforma za zajem biosenzoričnih podatkov v realnem času

Urška Nered, Jure Jeraj

Result d.o.o., Ljubljana, Slovenija
urska.nered@result.si, jure.jeraj@result.si

Pametne naprave s senzorji, ki beležijo naše gibalne, spalne in druge navade, so potencialno obsežen vir podatkov – še posebej, če jih lahko izvlečemo in združimo s podatki iz drugih virov. V prispevku predstavljamo podatkovno platformo, ki je namenjena prav temu – zajemu, shranjevanju in obogatitvi biosenzoričnih podatkov v realnem času. Za izhodiščne komponente smo izbrali Apache Kafka kot komunikacijsko vodilo, Apache Druid za shranjevanje in ksqlDB za transformacije podatkov. Taka platforma predstavlja osnovo in izhodišče za končni produkcijski sistem, saj že temelji na ustreznih tehnologijah za delo s pretočnimi podatki v realnem času.

Ključne besede:
podatkovna platforma
biosenzorični podatki
pretočni podatki
Apache Kafka
Apache Druid
ksqlDB

Univerzitetna založba
Univerze v Mariboru

ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.19

1 Uvod

Večina si nas življenje deli z vsaj eno pametno napravo, ki zbira podatke o naših dejavnostih, spalnih navadah, srčnem utripu in drugih metrikah, ki so posredno ali neposredno povezane tudi z našim zdravjem. Ker so naprave vedno z nami, je zajem takih podatkov enostaven tudi dolgoročno in v velikih količinah, s čimer predstavlja velik potencial za spremljanje ne le naših športnih dosežkov ampak tudi zdravstvenega stanja v daljšem obdobju, zaznavanje sprememb, spremljanje pacienta v času okrevanja. Če veliko takih podatkov naprave že beležijo in hranijo, zakaj jih ne bi izkoristili?

Za izkoriščanje takih podatkov najprej naletimo na problem zaprtih podatkov, tipično vezanih na proizvajalca naprave ali aplikacije, ki beleži meritve. Problem ni nepomemben, ni pa nepremostljiv. Podatke torej z nekaj dela imamo. Kaj pa zdaj? Lahko jih shranjujemo, a sami po sebi brez pametne uporabe niso vredni veliko.

Za resnično obogatitev in dodano vrednost iz podatkov, bi si jih želeli zajemati, kombinirati s podatki iz drugih virov, jih imeti možnost ponovno uporabiti za drugačen namen, vse to pa početi sproti v realnem času. Tukaj nastopi podatkovna platforma kot enotna točka, kamor se lahko stekajo podatki iz vseh naših pametnih naprav skupaj s podatki iz drugih poljubnih virov. Omogoča nam njihovo shranjevanje, procesiranje, integracijo, obogatitev za sprotno uporabo, hkrati pa spremljanje trendov v daljšem časovnem obdobju. Ob primernih varnostnih mehanizmih je z dovoljenjem posameznika to lahko osnova za zbiranje podatkov za namene znanstvenih raziskav, treniranje modelov umetne inteligence za diagnostične namene, zgodnje zaznavanje sprememb v zdravstvenem stanju itd. V nadaljevanju bo prikazan zametek takšne podatkovne platforme. Njeni ključni sestavni deli so Apache Kafka za zajem podatkov s senzorjev v realnem času, ksqlDB za njihovo sprotno procesiranje in Apache Druid za shranjevanje za takojšnjo ali kasnejšo uporabo.

2 Na kratko o podatkih

Podatkovna platforma, ki jo predstavljamo v tem članku, je namenjena zbiranju in obdelavi podatkov, zbranih s senzorji pametnih naprav – pametnih telefonov, ur, majic, tehtnic … Pametne naprave, ki nas spremljajo večino časa, o nas vejo veliko: koliko in kako se gibamo, kako spimo, koliko smo pod stresom itd., vendar so podatki, ki jih zberejo, težko dosegljivi izven naprave same ali mobilnih aplikacij proizvajalca. Kljub nekaterim standardom, ki obstajajo, pa je implementacija med proizvajalci različna.

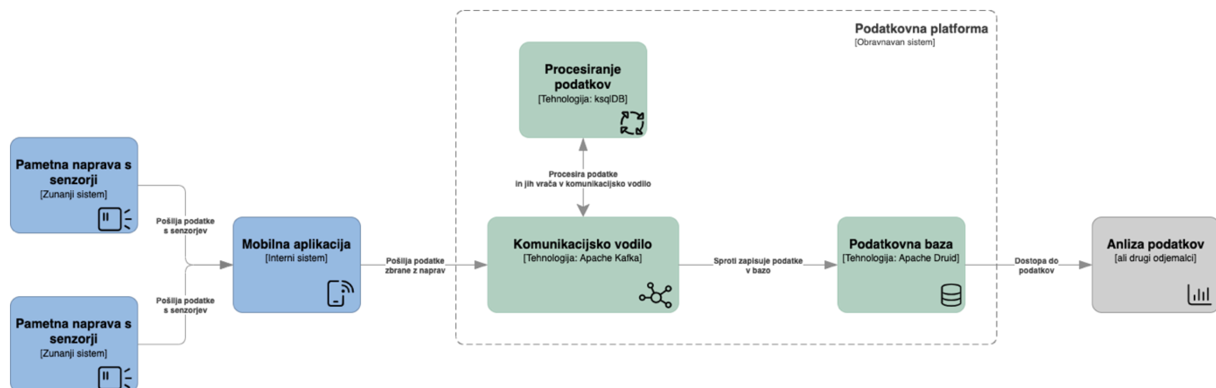
Za pridobivanje podatkov iz različnih pametnih naprav smo razvili svojo mobilno aplikacijo z zalednim delom, ki pa bo tema za kak drug članek. V našem primeru je dovolj, da vemo, da lahko sistem podatke na platformo pošilja v realnem času, če je pametna naprava dovolj blizu telefona, da sta lahko povezana preko Bluetooth povezave (v nasprotnem primeru naprava podatke shrani in pošlje, ko se spet povežeta). Podatki so opremljeni s tipom naprave, ki jih je zbrala, časom meritve, tipom meritve (koraki, srčni utrip, teža itd.) in seveda izmerjeno vrednostjo v dogovorjeni enoti.

3 Arhitektura podatkovne platforme

Za osnovno podatkovno platformo za zajem in uporabo biosenzoričnih podatkov potrebujemo le nekaj komponent, ki jih kasneje lahko nadgrajujemo. O komponentah platforme govorimo od sprejema podatkov, ki jih pošilja zaledni del mobilne aplikacije naprej. Sama mobilna aplikacija skupaj z zalednim delom, pa tudi pametne naprave, ki zbirajo podatke, so z vidika platforme zunanje naprave oz. zunanji vir podatkov, kot je prikazano na sliki 1.

Ker gre za podatke s senzorjev, pričakujemo veliko število majhnih sporočil v kratkih časovnih intervalih. Kadar je pametna naprava povezana s telefonom, mu bo meritve pošiljala sproti, telefon pa jih bo sproti posredoval

naprej. Zato bi želeli celo platformo prilagoditi zajemanju podatkov v realnem času. Kot osrednjo komponento naše platforme izberemo komunikacijsko vodilo, katerega glavna funkcija je sprejem podatkov iz zunanjih virov, v našem primeru mobilne aplikacije, in distribucija podatkov ostalim komponentam platforme. Podatke seveda želimo nekje hraniti, zato na komunikacijsko vodilo priključimo še podatkovno bazo, ki bo prejete podatke sproti zapisovala (shranjevala). Ker bi radi podatke sproti obdelali in obogatili že na nivoju sporočila, izberemo še komponento za procesiranje pretočnih podatkov v realnem času, ki bo brala vhodne podatke s komunikacijskega vodila, jih obogatila in ločeno zapisovala nazaj v komunikacijsko vodilo, da bodo na voljo za shranjevanje v bazo in vsem ostalim potencialnim odjemalcem. Za našo platformo smo kot komunikacijsko vodilo izbrali **Apache Kafka** [1], za podatkovno bazo **Apache Druid** [2], podatke pa bomo obdelovali s **ksqlDB** [3].



Slika 1: Konceptualna arhitektura predstavljene podatkovne platforme.

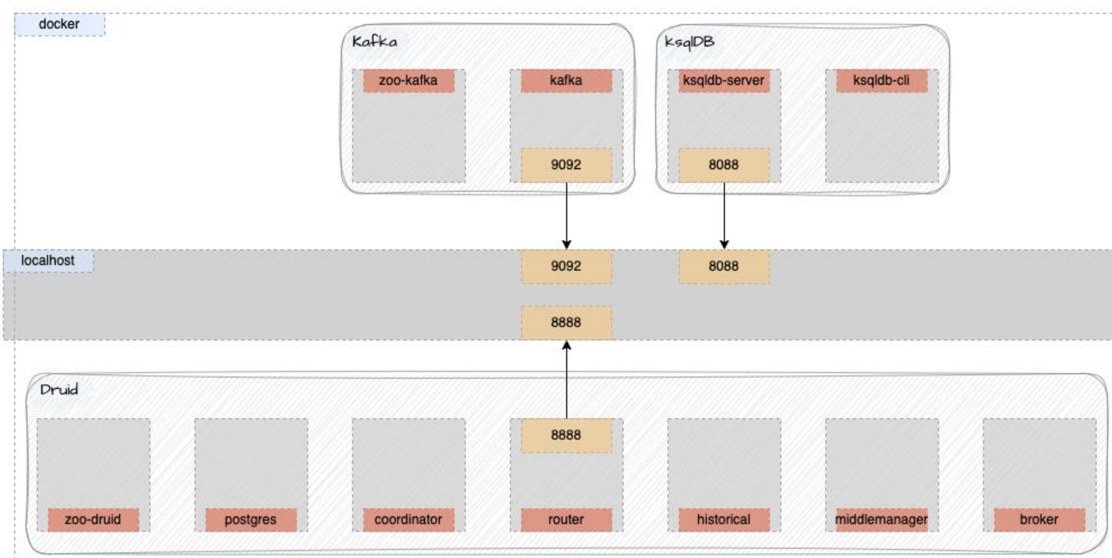
3.1 Podrobna arhitektura

Vse komponente podatkovne platforme so za naš primer nameščene kot Docker vsebniki, ki jih poženemo z eno definirano docker-compose datoteko. Slika 2 prikazuje vsebnike, ki jih moramo zagnati za delovanje vseh delov platforme. Vsaki komponenti pripada več kot en vsebnik, saj različni deli skrbijo za različne naloge, na tak način pa jih lahko tudi neodvisno skaliramo glede na naše potrebe. Za Kafko in Druid potrebujemo še ZooKeeper [4], ki skrbi za povezovanje in komunikacijo med njunimi posameznimi vmesniki[1].

Na sliki 2 so prikazana vrata (ang. *port*), preko katerih je posamezen vsebnik dostopen. Zaradi preglednosti so prikazana le tista, ki so najbolj pomembna in jih bomo uporabljali v našem primeru.

V produkcijskem okolju bi postavitev seveda izgledala drugače. Še vedno je možna postavitev vseh komponent z vsebniki, čeprav se za izbrane komponente načeloma priporoča neposredna namestitev na strežnik. Bi pa v nasprotju z našo platformo, kjer vse komponente zaženemo na enem strežniku oz. osebnem računalniku, za produkcijsko rešitev seveda potrebovali gručo strežnikov in orodje za orkestracijo vsebnikov (Docker Swarm, Kubernetes ipd.).

[1] https://www.fluentd.org/
https://grafana.com/oss/loki/
https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
CVE – izvorno "Common Vulnerabilities and Exposures", https://www.cve.org/
https://www.defectdojo.org/
seznam sestavnih delov – izvorno "Software Bill of Materials"



Slika 2: Podrobna arhitektura predstavljene platforme s prikazom vseh uporabljenih Docker vmesnikov.

3.2 Komunikacijsko vodilo – Apache Kafka

Komunikacijsko vodilo ima v podatkovni platformi pomembno funkcijo. Njegova glavna naloga je distribucija podatkov med komponentami platforme. Komponente bi si podatke lahko sicer izmenjevale tudi neposredno, vendar se na ta način stvari lahko hitro zapletejo. Komponente med sabo komunicirajo z različnimi protokoli, nekatere morda zahtevajo razvoj lastne rešitve za povezavo z drugimi komponentami itd. Z naraščanjem števila komponent kompleksnost takega sistema hitro narašča, komponente pa so, kar se tiče komunikacije, močno odvisne druga od druge. V primeru, da ena od para ne deluje, mora druga čakati, preden ji lahko pošlje podatke, menjava ene komponente z novo pa lahko zahteva veliko uskladitev v celotnem sistemu.

Za poenostavitev sistema in manjšo soodvisnost komponent je smiselno uvesti novo, ki skrbi za prenos podatkov med ostalimi – komunikacijsko vodilo. Pojem je sicer splošen in lahko pomeni marsikaj, odvisno od potreb našega sistema – komponente za integracijo, asinhroni sistemi za posredovanje sporočil, razni drugi posredniki in vmesniki itd., v našem primeru pa govorimo predvsem o sistemih za pretok podatkov v realnem času. V ta namen smo kot komunikacijsko vodilo v naši platformi izbrali orodje Apache Kafka [1].

Ključni nalogi Kafke sta prenos podatkov in zmanjšanje odvisnosti med ostalimi komponentami v sistemu, saj si podatke izmenjujejo preko Kafke, ne več neposredno. Kafka sporočila prenaša v binarni obliki, s čimer zagotavlja večje hitrosti prenosa in omogoča prenos sporočil različnih formatov. Za serializacijo in deserializacijo sporočil skrbijo ustvarjalci in odjemalci sporočil. Sporočilo je sestavljeno iz neobveznega ključa in pripadajoče vrednosti oz. vsebine sporočila. Sistem, ki ustvari sporočilo, ga zapiše v določeno temo (ang. *topic*). Vsaka vrsta sporočil tipično pripada svoji temi, podobno kot zapis v tabeli v relacijski bazi. Ni pa taka delitev obvezna, sporočila lahko npr. v teme razporedimo glede na izvorni sistem, eno sporočilo lahko zapišemo v več kot eno temo ipd. Vsaka tema je razdeljena na particije, sporočila z enakim ključem pa so vedno pripisana isti particiji. Znotraj ene particije je vrstni red sporočil zagotovljen, vsako sporočilo ima svoj identifikator, ki je vedno naraščajoč. Sporočila so shranjena za določen čas (osnovna nastavitev je 7 dni), v tem času pa jih lahko prebere poljubno število odjemalcev. Odjemalec sporočil na podlagi identifikatorja zadnjega prebranega sporočila v vsaki particiji ve, katerih sporočil še ni prebral. V primeru težav ob ponovnem zagonu nadaljuje z branjem od zadnjega prebranega sporočila dalje.

3.2.1 Kafka v naši platformi

Za našo podatkovno platformo z biosenzoričnimi podatki imamo za vsak tip naprave, ki lahko pošilja podatke s svojih senzorjev, narejeno ločeno temo, katere ime je enako imenu, ki ga določimo napravi. V primeru dodajanja novih naprav se nove teme ustvarijo samodejno (taka nastavitev za produkcijski sistem sicer ni priporočljiva, saj običajno želimo imeti več nadzora).

V produkcijskem okolju bi za zagotavljanje dostopnosti sistema za Kafko uporabili vsaj tri strežnike. V našem primeru imamo postavljen le en "strežnik" (vsebnik), vseeno pa za pravilno delovanje potrebujemo tudi ZooKeeper. Kafka je z lokalnega okolja dostopna na vratih 9092 (Slika).

Mobilna aplikacija podatke s senzorjev pošilja v formatu JSON, vsako sporočilo pa vključuje tri vrednosti:

- Type (*String*): tip meritve, npr. "HEART_RATE" (srčni utrip); hkrati tudi ključ za sporočilo,
- Value (*Float*): izmerjena vrednost v dogovorjeni enoti, za srčni utrip so to npr. udarci na minuto oz. BPM,
- DateTime (*String*): datum in čas meritve (UTC).

Primeri sporočil so prikazani na sliki 3.

```
kcat -C -b localhost:9092 \
    -t GarminVivoactive3 \
     -o beginning -e
{"type":"HEART_RATE","value":161.0,"dateTime":"2022-10-23T07:49:03"}
{"type":"CADENCE","value":83.0,"dateTime":"2022-10-23T07:49:03"}
{"type":"HEART_RATE","value":162.0,"dateTime":"2022-10-23T07:49:04"}
{"type":"CADENCE","value":83.0,"dateTime":"2022-10-23T07:49:04"}
{"type":"HEART_RATE","value":164.0,"dateTime":"2022-10-23T07:49:08"}
{"type":"CADENCE","value":82.0,"dateTime":"2022-10-23T07:49:08"}
{"type":"HEART_RATE","value":167.0,"dateTime":"2022-10-23T07:49:11"}
{"type":"CADENCE","value":83.0,"dateTime":"2022-10-23T07:49:11"}
{"type":"HEART_RATE","value":170.0,"dateTime":"2022-10-23T07:49:12"}
{"type":"CADENCE","value":83.0,"dateTime":"2022-10-23T07:49:12"}
{"type":"HEART_RATE","value":173.0,"dateTime":"2022-10-23T07:49:14"}
{"type":"CADENCE","value":83.0,"dateTime":"2022-10-23T07:49:14"}
```

Slika 3: Primer sporočil, ki jih na platformo pošilja mobilna aplikacija za zajem podatkov s pametnih naprav.

3.3 Shranjevanje podatkov in poizvedbe – Apache Druid

Apache Druid je precej nova baza, njen razvoj se je začel leta 2011, odprtokodna pa je od leta 2012 (Apache licenca od leta 2015). Danes jo uporablja kar nekaj velikih podjetij, med drugimi Airbnb, eBay, Netflix, Paypal, Slack in Reddit. Z nje zajemajo ogromne količine podatkov v realnem času (Netflix npr. v Druid shrani do 2 TB podatkov vsako uro, eBay pa zajema podatke s hitrostjo več kot 100.000 dogodkov na sekundo) [7].

Naš primer je seveda veliko manjši, za trenutno stanje bi zadostovala katerakoli podatkovna baza. Je pa smiselno že v tej fazi predvideti, kakšne podatke pričakujemo, kakšna rast je mogoča in na kakšen način jih želimo shranjevati ter nato do njih dostopati, in izbrati podatkovno bazo, ki je prilagojena taki uporabi. V našem primeru gre za meritve senzorjev, kjer pričakujemo veliko število majhnih sporočil, podatke pa bi radi zajemali v realnem času. Še ena tipična lastnost podatkov s senzorjev (in pravzaprav vseh podatkov v realnem času) je, da so časovno urejeni in je čas nastanka podatka ena od ključnih informacij. S tega vidika izbira bolj klasične relacijske baze ni najboljša, podatki bolj ustrezajo kakšni od časovno vrstičnih (ang. *timeseries*) baz. Druid je v prvi vrsti namenjen delu s pretočnimi podatki in v svoji arhitekturi združuje principe podatkovnih skladišč, časovno vrstičnih baz in sistemov za pregledovanje log datotek, tako da poleg hitrega shranjevanja pretočnih podatkov omogoča tudi dober odziv na analitične poizvedbe [8]. Velja pa omeniti, da Druid ni dobra izbira za sisteme, kjer potrebujemo hitre posodobitve obstoječih podatkov in v primerih, ko želimo v poizvedbah združevati več zelo velikih tabel dejstev (ang. *fact table*) [8].

Druid sestavlja šest vrst internih komponent, ki skrbijo za različne naloge, vsako pa lahko horizontalno skaliramo glede na svoje potrebe (slika 2):

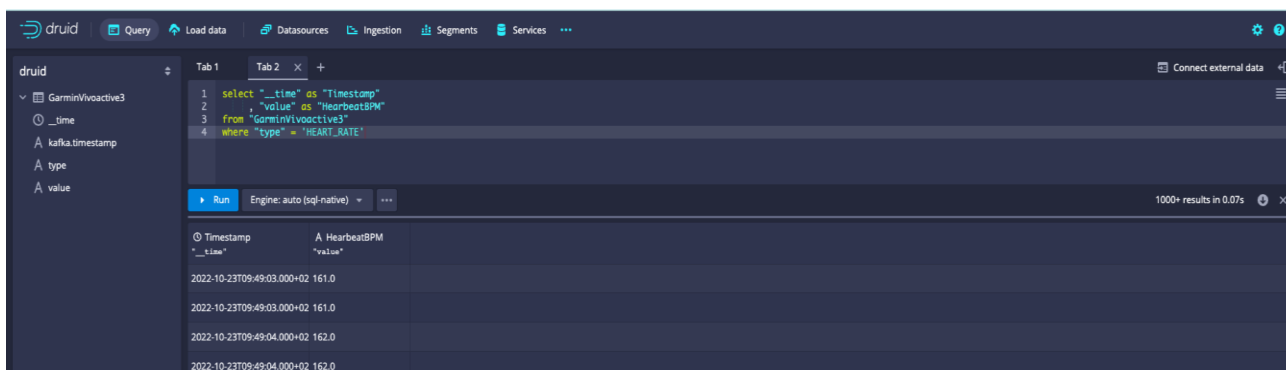
- *Coordinator* skrbi za dostopnost podatkov v sistemu,
- *Overlord* dodeljuje naloge, povezane z zajemom podatkov,
- *Broker* sprejema in streže poizvedbe,
- *Router* ni obvezen, skrbi pa za preusmeritev zahtev zgornjim trem komponentam,
- *Historical* hrani podatke za poizvedbe,
- *MiddleManager* zajema vhodne podatke.

Poleg vsega tega za delovanje Druida potrebujemo še relacijsko bazo, običajno PostgreSQL, kamor se shranjujejo metapodatki, ZooKeeper, ki skrbi za koordinacijo strežnikov in komponent, za prenos podatkov v ozadju in ponovno vzpostavitev sistema, v primeru popolne odpovedi pa potrebujemo tudi zunanje shranjevanje podatkov, tipično S3, HDFS ali omrežno povezan datotečni sistem.

3.3.1 Druid v naši platformi

V našem primeru uporabljamo vseh šest Druidovih komponent, tudi *Router*, so pa vse postavljene kot Docker vmesniki (slika 2). Poleg osnovnih Druidovih komponent imamo še ločen ZooKeeper (ne delita si ga s Kafko) in PosgreSQL za shranjevanje metapodatkov. Za zunanjo hrambo podatkov uporabljamo kar lokalni datotečni sistem, saj v tej fazi podatkov še ni zelo veliko. Za zajem podatkov s Kafke moramo pri postavitvi Druida definirati ustrezno razširitev, ki to omogoča ("*druid-kafka-indexing-service*"). Ta bo nato avtomatično vključena, s čimer bosta omogočena povezava s Kafko in neposredni zajem podatkov.

Druidov uporabniški vmesnik (slika 2) je dosegljiv preko vrat 8888, od koder *Router* skrbi za preusmeritev naših zahtev ustreznim komponentam. Preko uporabniškega vmesnika lahko urejamo zajem podatkov, kjer v našem primeru definiramo na Kafki temo, ki jo želimo brati, določimo stolpec, ki bo primarno uporabljen za particioniranje po času, določimo morebitne filtre za podatke, ki jih ne želimo zajemati (npr. neveljavne vrednosti na senzorju), in ostale nastavitve zajema. Ko nastavitve objavimo, se zažene zajemanje podatkov. Podatki so takoj po začetku zajema že na voljo za poizvedbe (slika 4). Naprej jih lahko obdelujemo v jeziku SQL, jih po želji filtriramo, agregiramo itd.



Slika 4: Prikaz Druidovega uporabniškega vmesnika, kjer lahko pregledujemo podatke.

Ker je Druid v osnovi namenjen delu s podatki v časovnih vrstah, ima veliko funkcionalnosti vezanih na časovne vrednosti. Tako lahko npr. Nastavimo, v katerem časovnem pasu smo, kar bo časovne podatke izpisovalo v našem lokalnem času, čeprav so shranjeni v UTC. Za optimizacijo poizvedb se lahko tudi odločimo, da so nam dovolj približni rezultati agregacij, kar še dodatno pohitri obdelavo podatkov.

3.4 Procesiranje podatkov – ksqlDB

Manj zahtevno procesiranje podatkov bi sicer lahko naredili kar v Druidu in izpustili ločeno komponento, vendar smo se zaradi ločitve nalog in lažje potencialne menjave komponent v prihodnosti odločili, da shranjevanje in procesiranje ločimo.

Zaradi enostavnosti uporabe smo izbrali ksqlDB [2], ki je del Kafka ekosistema. Podoben namen ima tudi Kafka Streams [9], je pa uporaba ksqlDB veliko enostavnejša, saj se uporablja kot ločena komponenta, transformacije podatkov pa lahko delamo v jeziku SQL, kar omogoča široko uporabo. Po drugi strani je Kafka Streams knjižnica, ki jo lahko uporabimo v lastni aplikaciji, transformacije pa so zato manj dostopne za analitike. Druga velika prednost ksqlDB so integrirani vmesniki za povezovanje najbolj razširjenih zunanjih podatkovnih sistemov s Kafko, za katere bi sicer uporabili komponento Kafka Connect [10].

Komponenta ksqlDB omogoča sprotno procesiranje pretočnih podatkov s Kafke, pri čemer lahko podatke filtriramo, agregiramo in združujemo s podatki iz drugih Kafka tem. Končni rezultat procesiranja so pretočni podatki v ločeni temi, za agregirane podatke pa nam je na voljo prikaz v tabeli, kjer se vrednosti sproti posodabljajo glede na nove prejete podatke. V ozadju obeh je določena Kafka tema, kamor ksqlDB pošilja podatke oz. spremembe, zunanji sistemi pa jo lahko berejo kot vsako drugo temo. Poleg takih "potisnih" poizvedb (ang. *push query*), kjer so rezultati posredovani naprej kot nov tok pretočnih podatkov, so nam na voljo tudi poizvedbe v bolj klasični obliki, kjer pošljemo poizvedbo in dobimo enkraten odgovor (ang. *pull query*). Take vrste poizvedb imajo določene omejitve, ne podpirajo npr. neposrednih agregacij, saj se rezultat le-teh načeloma ves čas spreminja s prihodom novih podatkov. V takem primeru moramo v ozadju ustvariti tabelo kot potisno poizvedbo in agregacije pripraviti že v tabeli, nato pa jih s poizvedbo le prebrati.

Za delovanje ksqlDB potrebujemo eno komponento – strežnik, opcijsko pa lahko dodamo tudi CLI, s katerim lahko preko ukazne vrstice upravljamo z vsebino (slika 2). Kadar interaktivne poizvedbe in transformacije niso potrebne ali pa celo niso zaželene, lahko to komponento izpustimo in ob zagonu vmesnika podamo datoteko z definiranimi tabelami in tokovi podatkov, ki jih želimo ustvariti.

S svojimi funkcionalnostmi daje ksqlDB uporabniško izkušnjo podatkovne baze, ki omogoča povezovanje z drugimi sistemi, procesiranje podatkov in poizvedbe na "tabelah".

3.4.1 ksqlDB v naši platformi

V naši platformi ksqlDB uporabljamo za sprotno filtriranje in obogatitev podatkov, ki prihajajo z merilnih naprav. Ker vsi podatki, ki jih izmeri določena naprava, pridejo v isto Kafka temo (Slika 3), jih s ksqlDB ločimo glede na tip meritve. V primeru je prikazano, kako meritve srčnega utripa ločimo od ostalih v svojo temo "heart_rate", vrednosti preimenujemo ("value" preimenujemo v "measured_bpm"), poleg tega pa glede na izmerjene vrednosti dodamo še podatek o coni srčnega utripa ("hr_zone"), na podlagi katere lahko analiziramo napor med dejavnostjo (slika 5).

```
ksql> select * from heart_rate limit 20;
+-------------------------------------+-------------------------------------+-------------------------------------+
|DATETIME                             |MEASURED_BPM                         |HR_ZONE                              |
+-------------------------------------+-------------------------------------+-------------------------------------+
|2022-10-23T07:49:03                  |161.0                                |ZONE 2                               |
|2022-10-23T07:49:04                  |162.0                                |ZONE 2                               |
|2022-10-23T07:49:08                  |164.0                                |ZONE 3                               |
|2022-10-23T07:49:11                  |167.0                                |ZONE 3                               |
|2022-10-23T07:49:12                  |170.0                                |ZONE 3                               |
|2022-10-23T07:49:14                  |173.0                                |ZONE 3                               |
|2022-10-23T07:49:15                  |174.0                                |ZONE 3                               |
|2022-10-23T07:49:17                  |177.0                                |ZONE 4                               |
|2022-10-23T07:49:21                  |180.0                                |ZONE 4                               |
|2022-10-23T07:49:23                  |181.0                                |ZONE 4                               |
|2022-10-23T07:49:31                  |183.0                                |ZONE 4                               |
|2022-10-23T07:49:36                  |184.0                                |ZONE 4                               |
|2022-10-23T07:49:43                  |184.0                                |ZONE 4                               |
|2022-10-23T07:49:50                  |185.0                                |ZONE 4                               |
|2022-10-23T07:49:51                  |185.0                                |ZONE 4                               |
|2022-10-23T07:49:58                  |185.0                                |ZONE 4                               |
|2022-10-23T07:50:06                  |185.0                                |ZONE 4                               |
|2022-10-23T07:50:07                  |186.0                                |ZONE 5                               |
|2022-10-23T07:50:14                  |185.0                                |ZONE 4                               |
|2022-10-23T07:50:21                  |186.0                                |ZONE 5                               |
Limit Reached
Query terminated
ksql>
```

Slika 5: Primer poizvedbe s ksqlDB.

Če se vrnemo na Kafko, lahko vidimo, da je zdaj na voljo tudi tema »heart_rate«, iz katere lahko beremo procesirane podatke (slika 6).

```
kcat -C -b localhost:9092 \
    -t heart_rate \
     -o beginning -e

{"DATETIME":"2022-10-23T07:49:03","MEASURED_BPM":161.0,"HR_ZONE":"ZONE 2"}
{"DATETIME":"2022-10-23T07:49:04","MEASURED_BPM":162.0,"HR_ZONE":"ZONE 2"}
{"DATETIME":"2022-10-23T07:49:08","MEASURED_BPM":164.0,"HR_ZONE":"ZONE 3"}
{"DATETIME":"2022-10-23T07:49:11","MEASURED_BPM":167.0,"HR_ZONE":"ZONE 3"}
{"DATETIME":"2022-10-23T07:49:12","MEASURED_BPM":170.0,"HR_ZONE":"ZONE 3"}
{"DATETIME":"2022-10-23T07:49:14","MEASURED_BPM":173.0,"HR_ZONE":"ZONE 3"}
{"DATETIME":"2022-10-23T07:49:15","MEASURED_BPM":174.0,"HR_ZONE":"ZONE 3"}
{"DATETIME":"2022-10-23T07:49:17","MEASURED_BPM":177.0,"HR_ZONE":"ZONE 4"}
{"DATETIME":"2022-10-23T07:49:21","MEASURED_BPM":180.0,"HR_ZONE":"ZONE 4"}
{"DATETIME":"2022-10-23T07:49:23","MEASURED_BPM":181.0,"HR_ZONE":"ZONE 4"}
{"DATETIME":"2022-10-23T07:49:31","MEASURED_BPM":183.0,"HR_ZONE":"ZONE 4"}
```

Slika 6: Podatki, ki smo jih obdelali s ksqlDB so na voljo v novi Kafka temi "heart_rate".

Poleg surovih podatkov lahko zdaj v Druid zajemamo tudi že procesirane podatke o srčem utripu in na njih delamo dodatne analize.

4 Zaključek

V prispevku smo na kratko predstavili osnutek podatkovne platforme za zajem in obdelavo biosenzoričnih podatkov v realnem času. Platforma podatke prejema iz različnih pametnih naprav (pametna ura, merilec srčnega utripa ...). Enotna vstopna točka za podatke je Kafka, ki deluje kot osrednji živčni sistem platforme. Podatke lahko nato "v letu" prečistimo, transformiramo in poenotimo s ksqlDB. Končne (pa tudi surove in neobdelane) podatke na koncu shranimo v podatkovno bazo Apache Druid, ki je optimizirana za hitro transakcijsko zapisovanje in že v osnovi kompatibilna s Kafko, po drugi strani pa je prilagojena za hitro branje in odziv na poizvedbe tudi na velikih količinah podatkov.

S predstavljeno rešitvijo lahko precej hitro pridemo do zametka platforme, ki v vseh fazah že uporablja ene od vodilnih gradnikov na tem področju. To je lahko osnova in model za izgradnjo prave produkcijske podatkovne platforme za delo s podatki v realnem času.

Literatura

[1] https://kafka.apache.org/, Apache Kafka, obiskano 17. 7. 2023.

[2] https://ksqldb.io/, ksqlDB, obiskano 17. 7. 2023.

[3] https://druid.apache.org/, Apache Druid, obiskano 17. 7. 2023.

[4] https://zookeeper.apache.org/, Apache ZooKeeper, obiskano 18. 7. 2023.

[5] https://developer.confluent.io/learn/kraft/, KRaft: Apache Kafka without ZooKeeper, obiskano 18. 7. 2023.

[6] https://cwiki.apache.org/confluence/display/KAFKA/KIP-833%3A+Mark+KRaft+as+Production+Ready, KIP-833: Mark KRaft as Production Ready, obiskano 18. 7. 2023.

[7] https://druid.apache.org/druid-powered.html, Powered by Apache Druid, obiskano 19. 7. 2023.

[8] https://druid.apache.org/docs/latest/design/index.html, Introduction to Apache Druid, obiskano 19. 7. 2023.

[9] https://kafka.apache.org/documentation/streams/, Kafka Streams, obiskano 24. 7. 2023.

[10] https://docs.confluent.io/platform/current/connect/index.html, Kafka Connect, obiskano 24. 7. 2023.