Arhitektura podatkovne platforme za podporo naprednih BI rešitev

Jani Kajzovar, Klemen Drev, Uroš Trstenjak, Aljaž Mislovič, Miha Pavlinek

Databox Inc., Ptuj, Slovenija
jani.kajzovar@databox.com, klemen.drev@databox.com, uros.trstenjak@databox.com, aljaz@databox.com, miha.pavlinek@databox.com

Predstavljamo modularno arhitekturo podatkovnega jezera, ki omogoča izvajanje analitičnih poizvedb neposredno na Parquet datotekah, shranjenih v objektni shrambi (npr. S3) z uporabo vgrajenega SQL-pogona DuckDB. Sistem vsebuje zajem in pretvorbo surovih podatkov iz heterogenih virov v stolpčni format Parquet, njihovo shranjevanje v objektni shrambi ter dinamično poizvedovanje znotraj mikroservisov. Ključni prispevek je neposredna integracija rezultatov poizvedb v poslovnoanalitična orodja brez potrebe po klasičnih podatkovnih bazah, kar omogoča hiter dostop do podatkov in horizontalno skalabilnost. Rešitev je univerzalna in primerna za različne domene, vključno z IoT, telemetrijo in spletno analitiko.

Ključne besede:
podatkovno jezero,
objektna shramba,
Parquet,
DuckDB,
mikroservisna arhitektura,
poslovna analitika.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.8

1 Uvod

V času, ko količina razpoložljivih podatkov eksponentno narašča, podjetja iščejo arhitekturne pristope, ki omogočajo hitrejši dostop do podatkov, zmanjšanje stroškov infrastrukture in večjo prilagodljivost pri analitiki. Klasična podatkovna skladišča s centralizirano obdelavo pogosto ne zadoščajo več – ne le zaradi omejitev pri skalabilnosti, temveč tudi zaradi:

- visoke kompleksnosti upravljanja gruče strežnikov in vzdrževanja,
- težav pri horizontalnem razširjanju (elastičnosti),
- togosti sheme, ki otežuje evolucijo in migracije,
- slabše zmogljivosti zaradi tesne povezanosti med shranjevanjem in obdelovanjem.

Pojmi, kot so "data lake" in "in-process analytics", odražajo premik k bolj modularnim, porazdeljenim rešitvam, ki podatke obdelujejo tam, kjer so shranjeni – in samo takrat, ko je to potrebno.

Oblikovanje arhitektur, kjer lahko aplikacije izvajajo poizvedbe neposredno na podatkih v objektni shrambi, se je izkazalo kot obetaven pristop za številne primere rabe. Pojav orodij, kot sta DuckDB in Apache Arrow, je omogočil razvoj lahkih analitičnih rešitev, ki se ne zanašajo več na centralne baze, temveč na vgrajene SQL pogone, ki tečejo v kontekstu aplikacijskih servisov. Tak pristop je še posebej primeren za mikroservisne arhitekture, kjer posamezni deli sistema potrebujejo lasten vpogled v podatke, brez globalne koordinacije ali skupne podatkovne plasti.

Takšno arhitekturo smo v podjetju Databox zasnovali za potrebe produkta »Datasets«, ki uporabnikom omogoča poglobljeno analitiko na osnovi surovih podatkov, ki so združeni v podatkovne nabore (ang. datasets).

2 Zahteve in načela zasnove

Motiv za vpeljavo nove platforme izhaja iz produktne zahteve za rešitev "Datasets", kjer uporabniki na preprost način iz heterogenih virov pridobivajo surove podatke, ki so organizirani v podatkovne nabore (datasets). Ti so uporabnikom prikazani v obliki tabel, nad katerimi je omogočeno filtriranje, sortiranje, skrivanje/premikanje stolpcev, dodajanje novih kalkuliranih stolpcev ipd. Te modificirane nabore je kasneje mogoče združevati ter nad njimi pripravljati poslovne metrike, ki so namenjene vizualizaciji ali nadaljnji analizi.
Za potrebe jasnejše strukture smo podatkovne nabore razdelili na tri logične komponente: izvorne nabore (ang. source datasets), ki predstavljajo originalne, neobdelane podatke; modificirane nabore (modified datasets), kjer poteka sled modifikacij in transformacij; ter združene nabore (ang. merged datasets), ki združujejo vse izvorne ter druge združene nabore v nov nabor za poizvedovanje.

Druge pomembnejše zahteve:

- vsak nabor ima lastno podatkovno shemo z možnostjo prilagajanja,
- promptno odzivanje ob nalaganju izvornih naborov, modifikacijah, združevanju in pripravi metrik,
- podpora transformacijam ter sprotnim kalkulacijam na osnovi formul,
- v naborih je mogoče shranjevati večje količine podatkov (več milijonov vrstic),
- nizki obratovalni stroški.

Skladno s podanimi zahtevami, smo arhitekturo za podatkovno platformo zasnovali okoli štirih ključnih načel, ki omogočajo visoko učinkovitost, skalabilnost in nizke operativne stroške:

- **Ločenost shrambe in izvajanja** (ang. separation of storage and compute) Skladiščenje podatkov in izvajanje analitične logike sta v sistemu strogo ločena. Vsi podatki so shranjeni v enotnem formatu znotraj objektne shrambe, medtem ko se analitične poizvedbe izvajajo po potrebi v aplikacijskih servisih, z ločenim poizvedbenim pogonom. Ta zasnova omogoča neodvisno skaliranje

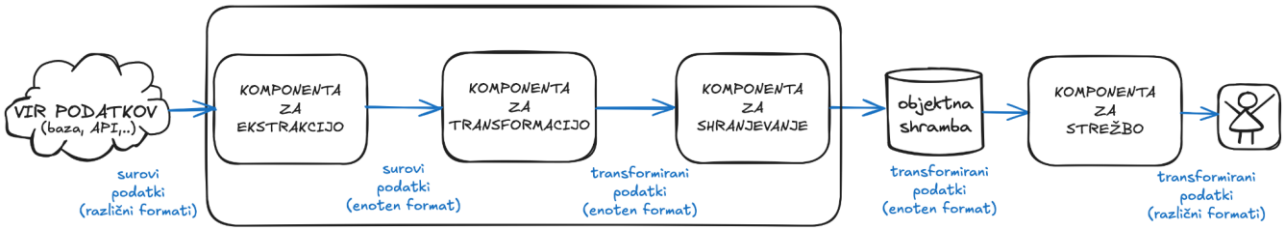
posameznih slojev in preprečuje tesno povezanost med podatkovno in obdelovalno komponento, kot je to pogosto pri klasičnih podatkovnih bazah.

- **Modularnost in zamenljivost komponent:** Vsi gradniki sistema – od zajema podatkov, transformacij, shranjevanja, pa do poizvedb – so zasnovani kot neodvisne, zamenljive enote. Tako lahko posamezne komponente zamenjamo ali nadgradimo brez večjih posegov v preostanek sistema. To omogoča hitrejše iteracije, testiranje alternativnih pristopov (npr. zamenjava objektne shambe z drugo tehnologijo) in lažje prilagajanje spremembam v oblačni infrastrukturi.
- **Stateless pristop v mikroservisnem okolju:** Obdelava podatkov poteka znotraj aplikacijskih mikroservisov, ki so po zasnovi "stateless". Vsaka poizvedba se izvede neodvisno, z uporabo začasnega konteksta. To pomeni, da mikroservisi ne hranijo stanja med zahtevki, kar omogoča enostavno razporejanje, samodejno skaliranje in odpornost na napake.
- **Horizontalna skalabilnost z minimalnim upravljanjem:** Ker je vsak mikroservis popolnoma neodvisna enota, z zmožnostjo analitične poizvedb, lahko sistem enostavno skaliramo horizontalno. To pomeni, da lahko avtomatsko prilagajamo število izvajalnih enot glede na obremenitev, na primer število zahtevkov ali dolžino vrste v sistemu za razpošiljanje sporočil. Ta pristop omogoča dinamično prilagajanje zmogljivosti – na primer ob povečanem številu zahtev za vizualizacijo, izvoze podatkov ali periodične transformacije – brez vpliva na druge dele sistema. Vsak izvajalna enota obdeluje svojo nalogo z izoliranim kontekstom, kar omogoča skoraj linearno skaliranje brez potrebe po deljenem stanju.

3 Arhitektura

3.1. Visokonivojska predstavitev

Za lažje razumevanje zasnove predstavljamo visokonivojski diagram arhitekture za pripravo izvornih naborov (ang. source datasets), ki predstavljajo originalne, neobdelane podatke. Diagram prikazuje ključne komponente sistema in pretok podatkov med njimi. Fokus je na logičnem zaporedju faz: od uvoza surovih podatkov do njihove transformacije, shranjevanja in produktne uporabe



Slika 1: Visokonivojska predstavitev arhitekture.

3.2. Tok podatkov

Arhitektura sistema sledi pretočnemu modelu, kjer so vse faze – od vnosa podatkov do strežbe rezultatov – ločene, modularne in sledljive. Le te smo v nadaljevanju podrobneje opisali:

Vnos podatkov iz zunanjih virov

Podatke v sistem dostavlja interni ETL mehanizem, ki skrbi za zajem iz zunanjih API-jev, relacijskih baz in drugih virov. Ta komponenta za ekstrakcijo podatkov predstavlja vhodno točko v podatkovno jezero. Pomembno je, da sistem vse podatke prevzame in jih pretvori v enoten stolpčen format in s tem skrije podrobnosti posameznega vira podatkov. To omogoča enoten podatkovni model in takojšnjo integracijo z nadaljnjimi komponentami.

Transformacija in tipizacija

Prejete podatke sistem najprej obdela z nizom transformacij, ki vključujejo:

- pretvorbo vrednosti (npr. normalizacija enot),
- tipizacijo stolpcev glede na predpisano shemo (npr. int64, timestamp, decimal),
- odstranjevanje ali preimenovanje stolpcev za boljšo razumljivost,

Transformacije se izvajajo znotraj komponente za transformacijo. Sistem ob tem spremlja spremembe sheme – če se vhodna shema razlikuje od pričakovane, se aktivira mehanizem za validacijo, ki zazna neskladja in po potrebi sproži opozorilo ali ustavi obdelavo. Ta pristop omogoča kontrolirano prilagajanje na evolucijo podatkov skozi čas.

Shranjevanje in verzioniranje v objektni shrambi

Pripravljen izvorni nabor podatkov v poenotenem formatu in normalizirani vsebini nato shranimo v objektno shrambo. Dodatno ta komponenta skrbi za verzioniranje. Verzioniranje omogoča obnovljivost preteklih stanj, podporo za "time-travel" razhroščevanje in enostavno testiranje novih transformacij brez izgube zgodovine. Poleg izvornih (source) naborov podatkov sistem podpira tudi shranjevanje modificirane (modified) in združene (merged) različice, ki služijo kot predelana ali kombinirana nadgradnja izvornih podatkov. Ostale različice imajo ločen ETL tok podatkov, ki pa kot vhod uporabijo predhodno shranjen nabor podatkov iz objektne shrambe.

Strežba rezultatov

Shranjene nabore podatkov iz objektne shrambe ponujamo preko različnih funkcionalnosti produktne rešitve "Datasets". Rezultati poizvedb so uporabnikom dostopni preko različnih kanalov:

- Vpogled v podatke: kot JSON odgovor prek REST API-ja,
- Izvoz podatkov: kot CSV ali Parquet export v namenske S3 exports mape,
- Vizualizacija: kot interaktivni prikaz v BI orodju, kjer API služi kot query backend.

Servis samodejno optimizira poizvedbe glede na kontekst (npr. paginacija, agregacije, filtri) in zagotavlja odzivne čase, primerno za interaktivno raziskovanje podatkov.

4 Ključne tehnologije

Izbira ključnih tehnologij, ki sestavljajo hrbtenico podatkovne arhitekture, vključujejo stolpčni format Parquet ter analitični pogon DuckDB. Poseben poudarek je na učinkovitem shranjevanju in obdelavi podatkov v podatkovnem jezeru, kjer te tehnologije omogočajo visoko zmogljivost tudi pri obdelavi večjih količin podatkov. Opisane so prednosti za analitiko ter praktične možnosti neposrednega branja, pisanja in procesiranja podatkovnih nizov.

4.1. Uporaba stolpčnega formata Parquet

Pri zasnovi arhitekture podatkovnega jezera smo se odločili za uporabo odprtokodnega stolpčnega formata Apache Parquet kot osnovnega formata za shranjevanje podatkov. Parquet je bil razvit posebej za potrebe analitičnih sistemov, kjer je branje podatkov pogosto omejeno na posamezne stolpce ali podmnožice tabel, ne pa

celotne vrstice. V takih primerih stolpčni formati nudijo bistveno boljšo učinkovitost glede hitrosti in porabe virov v primerjavi z vrstičnimi formati (npr. CSV, JSON).

Struktura in shema

Parquet temelji na strogo definirani shemi, ki opisuje strukturo podatkov v datoteki (stolpce, podatkovne tipe, gnezdenje, ponovitve). Ta shema je shranjena skupaj s podatki v datoteki Parquet, kar omogoča podatkovne vire z vgrajeno shemo in poenostavi izvajanje poizvedb brez zunanjih metapodatkov. Shema je definirana v formatu Apache Thrift, kar omogoča interoperabilnost s številnimi orodji in jeziki (npr. Spark, Pandas, DuckDB, Hive) [1]. Uporaba statične sheme pomeni tudi, da lahko poizvedbeni pogoni, kot je DuckDB, že pred začetkom branja podatkov optimizirajo poizvedbe - katere stolpce bodo brali in kako bodo interpretirali podatkovne tipe (npr. INT64, FLOAT, BOOLEAN, TIMESTAMP).

Kompresija in kodiranje

Parquet vključuje napredne možnosti kompresiranja in kodiranja, kot so:

- Stiskanje na ravni stolpcev (ang. Column-level compression): Vsak stolpec se stisne neodvisno, kar omogoča boljše razmerje kompresije, saj imajo podatki znotraj stolpcev pogosto podoben tip in porazdelitev.
- Kodirne sheme (ang. Encoding schemes): Uporablja učinkovite sheme, kot so "run-length encoding", "dictionary encoding" in "bit-packing", kar zmanjša velikost brez izgub.
  Podpora več algoritmom: Parquet omogoča uporabo različnih kompresijskih algoritmov (npr. Snappy, Gzip, Brotli, ZSTD), odvisno od primera uporabe in kompromisa med hitrostjo in kompresijo [1].

Kompresija v Parquetu ni le orodje za zmanjševanje velikosti, temveč ima neposreden vpliv na hitrost poizvedb – manjši podatkovni bloki pomenijo manj vhodno/izhodnih operacij in hitrejše prenose preko omrežja (kar je še posebej pomembno pri obdelavi podatkov, shranjenih v objektni shrambi).

Prednosti za analitiko v oblaku

Uporaba Parquet formata prinaša več ključnih prednosti za obdelavo podatkov v porazdeljenih in oblačnih okoljih:

- Selektivno branje (ang. predicate pushdown): Parquet omogoča, da poizvedbeni pogoni preberejo samo relevantne dele podatkov – tako po stolpcih kot po vrsticah (z uporabo statistik v metapodatkih) [2].
- Kompaktnost: Zaradi učinkovite kompresije so Parquet datoteke pogosto 5–10× manjše od enakovrednih CSV ali JSON datotek.
- Optimizacija prenosa iz objektne shrambe: Manjša količina podatkov pomeni nižje stroške prenosa iz shrambe ter manjšo obremenitev omrežja in procesorja.
- Podpora za evolucijo sheme: Parquet podpira določeno stopnjo fleksibilnosti pri spreminjanju sheme (npr. dodajanje/odstranjevanje stolpcev), kar olajša razvoj v hitro spreminjajočem se podatkovnem okolju.

4.2. DuckDB kot vgrajen SQL pogon za obdelavo Parquet podatkov

DuckDB je sodoben, stolpčno zasnovan relacijski pogon, optimiziran za analitične poizvedbe in zasnovan za "in-process" uporabo. Deluje kot vgrajena komponenta znotraj aplikacijskega procesa, brez potrebe po zunanjem strežniku, omrežnih povezavah ali kompleksni infrastrukturi. Zaradi tega je posebej primeren za mikroservisne in oblačne arhitekture, kjer želimo analitične zmožnosti vključiti neposredno v aplikacijsko logiko.

Neposredno branje in pisanje Parquet datotek

DuckDB podpira nativno branje in zapisovanje Parquet datotek – tako lokalno kot prek objektne shrambe, kot je Amazon S3 [3]. Poizvedbe se lahko izvajajo neposredno nad datotekami v oblaku brez vnaprejšnjega uvoza, kar omogoča query-on-read arhitekturo brez replikacije podatkov.
Primer branja Parquet iz S3:

```
SELECT *
FROM read_parquet('s3://my-bucket/data/events.parquet')
WHERE event_type = 'click';
```

Primer zapisovanja Parquet nazaj na S3:

```
COPY (
  SELECT * FROM session_events WHERE duration > 30
)
TO 's3://my-bucket/filtered/long_sessions.parquet'
(FORMAT PARQUET, COMPRESSION ZSTD);
```

Obdelava podatkov, večjih od pomnilnika (ang. spill to disk)

Čeprav DuckDB deluje "in-process" in primarno v pomnilniku, vključuje mehanizem za obdelavo naborov podatkov, ki presegajo razpoložljiv RAM. S pomočjo funkcionalnosti "spill to disk" DuckDB samodejno shranjuje vmesne rezultate na disk, kadar zmanjka pomnilnika med izvajanjem poizvedbe. To omogoča robustno obdelavo velikih Parquet datotek brez ročnega upravljanja z delitvijo podatkov ali ustvarjanjem podatkovnih delov (chunking) [4].

Privzeto se začasne datoteke ustvarjajo v sistemski začasni mapi, vendar lahko pot in velikost diskovnih medpomnilnikov prilagodimo z uporabo konfiguracijskih nastavitev DuckDB.

Visoka zmogljivost zaradi optimizirane izvedbe

DuckDB izvaja poizvedbe s t.i. vektoriziranim izvajanjem (ang. vectorized execution), kjer se podatki obdelujejo v blokih (vektorjih) – običajno 1024 vrstic naenkrat. To omogoča boljši izkoristek procesorskega predpomnilnika in SIMD ukazov v primerjavi z pristopi branja vrstico po vrstico (ang. row-at-a-time). Poleg tega izvaja številne optimizacije, kot so:

- odlaganje branja vrednosti do trenutka, ko so res potrebne (ang. late materialization) [6],
- filtriranje vrstic že med branjem iz Parquet datotek (ang. predicate pushdown),
- uporaba metapodatkov za optimizacijo načrta poizvedbe (ang. statistics-aware query planning),
- branje zgolj potrebnih stolpcev (ang. column pruning).

Kot rezultat DuckDB pogosto dosega hitrosti primerljive ali boljše od distribuiranih rešitev (npr. Spark, Presto) pri manjših do srednje velikih naborov podatkov, brez potrebe po grozdu strežnikov ali zapletenem usklajevanju.

Podpora za standardni SQL in napredne poizvedbe

DuckDB podpira obsežen podnabor SQL standarda: CTE-je (WITH izrazi), okenske funkcije (window functions), agregacije, GROUP BY ROLLUP, PIVOT, date/time funkcije, pod-poizvedbe, UNION, INTERSECT, EXCEPT, JOIN vseh vrst in še več [7]. To pomeni, da lahko kompleksne analitične poizvedbe, ki bi sicer zahtevale več korakov ali zunanja orodja, izvedemo v enem koraku - neposredno nad Parquet datotekami.
To zmanjša potrebo po dodatni obdelavi v aplikacijski kodi in omogoča bolj deklarativen slog razvoja.

Zaradi kombinacije lastnosti – "in-process" izvedbe, podpore za Parquet, neposredne integracije z objektno shrambo ter možnosti obdelave večjih naborov podatkov brez izpada – je DuckDB izjemno primeren za sodobne podatkovne arhitekture, kjer je potrebna hitra, fleksibilna in nizkocenovna analitika brez tradicionalnih omejitev centraliziranih podatkovnih skladišč, opisanih v uvodu.

5 Implementacija

Implementacija sistema temelji na načelih vgradljivosti, pretočnosti in odprtosti. Tehnologiji iz prejšnjega poglavja – DuckDB in Parquet – sta v središču zasnove, ki smo jo kombinirali z objektno shrambo kot trajnim slojem. Skupaj tvorijo osnovo pretočnega in modularnega podatkovnega sistema, prilagojenega sodobnim analitičnim potrebam.

5.1. Ključne komponente

Arhitekturo sestavlja pet ključnih komponent, ki skupaj tvorijo skalabilen, modularen in nizkocenoven podatkovni sistem, zasnovan za analitične delovne obremenitve v oblaku.

Parquet datoteke
Prej opisan Parquet je naš privzeti podatkovni format zaradi svoje stolpčne zasnove, učinkovite stiskanja, podpore za kompleksne tipe in možnost selektivnega branja. Vsaka datoteka vsebuje tudi samo-pojasnjevalno shemo, kar omogoča dinamično analizo brez zunanjih metapodatkov.

Amazon S3 objektna shramba
Amazon S3 predstavlja trajno plast za shranjevanje vseh naborov podatkov. Podatki so zapisani v Parquet formatu in organizirani po naborih, verzijah in vlogah (izvorni, predelani, združeni). Objektna shramba omogoča nizke stroške, visoko razpoložljivost in neposreden dostop prek HTTP(S), kar jo naredi idealno za podatkovna jezera.

DuckDB
DuckDB deluje kot vgrajen SQL pogon (embedded engine) znotraj vsakega mikroservisa. Poizvedbe se izvajajo neposredno nad Parquet datotekami na S3, brez potrebe po predhodnem nalaganju podatkov ali vzdrževanju ločene baze. Zaradi podpore za "spill to disk" in vektoriziranega izvajanja DuckDB omogoča zmogljivo obdelavo tudi večjih naborov podatkov v omejenih kontekstih. Kombinacija teh treh tehnologij tvori jedro naše aplikacije.

Dataset Engine / API servis
"Dataset Engine" je aplikacijski sloj, ki povezuje vse ostale komponente. Njegove naloge vključujejo pridobivanje podatkov iz zunanjih virov, pretvorbo v Parquet, transformiranje in normalizacijo podatkov, zapisovanje v S3, izvajanje poizvedb prek DuckDB ter vračanje rezultatov uporabnikom ali BI orodjem preko API vmesnikov. Ta komponenta predstavlja centralni zaledni servis naše rešitve "Datasets".

Kubernetes

Kubernetes služi kot temeljna infrastruktura za orkestracijo in izvajanje sistema. Vsi mikroservisi, vključno z instancami, ki uporabljajo DuckDB, tečejo kot podi (osnovne enote za izvajanje enega ali več povezanih kontejnerjev) znotraj Kubernetes okolja. To omogoča:

- avtomatsko skaliranje števila izvajalcev glede na obremenitev (npr. velikost čakalne vrste zahtevkov),
- dinamično upravljanje virov (CPU, pomnilnik, disk za začasne datoteke),
- visoko razpoložljivost z mehanizmi za ponovni zagon neuspešnih komponent.

S tem Kubernetes ni zgolj infrastrukturna plast, ampak aktivno podpira elastičnost, robustnost in učinkovitost celotne platforme.

5.2. Organizacija S3 prostora

Za učinkovito, pregledno in razširljivo upravljanje s podatki uporabljamo dosledno strukturirano organizacijo objektne shrambe v Amazon S3. S tem omogočamo enostavno upravljanje različic, arhiviranje, izvoze in obdelavo naborov podatkov na podlagi njihovega tipa in izvora.

Strukturirane mape

Organizacija S3 prostora temelji na jasni razdelitvi glede na tip nabora podatkov. Vsaka vrsta nabora ima svojo lastno korensko mapo, kar omogoča enostavno ločevanje, upravljanje in iskanje po posameznih tipih podatkov:

- sources/ – izvorni nabori podatkov, kot jih posreduje ETL sistem opisan v poglavju Arhitektura,
- modified/ – modificirani nabori podatkov so transformirane in obogatene različice izvornih podatkov (npr. z dodatnimi izračunanimi stolpci, pretvorbo podatkovnih tipov, filtriranjem),
- merged/ – rezultat združevanja dveh ali več naborov podatkov tvori združen nabor,
- exports/ – začasne datoteke, ustvarjene za zahtevke uporabniških izvozov keteregakoli nabora podatkov (npr. v format CSV ali Parquet).

Primer tipične poti do datoteke:

```
s3://databox-dev-datasets/sources/{datasetId}/v{timestamp}_{uuid}.parquet
s3://databox-dev-datasets/sources/{datasetId}/v{timestamp}_{uuid}.json
```

Verzioniranje in identifikacija

Vsak nabor podatkov je verzioniran z uporabo časovnega žiga v{timestamp} (npr. v202407210845), kar omogoča sledenje zgodovini in ponovljivost obdelav. Poleg verzije vsak zapis vsebuje tudi UUID, ki služi kot globalni identifikator posamezne datoteke. S tem lahko isti nabor podatkov obstaja v več različicah, vsaka z lastno vsebino in enolično identifikacijo.

Dodatno se za potrebe optimizacije shema vsake Parquet datoteke hrani tudi ločeno v .json obliki pod istim imenom, kar omogoča samostojen dostop do metapodatkov brez potrebe po branju vsebine parquet datoteke.

Arhiviranje in življenjski cikel

Za optimizacijo stroškov shranjevanja uporabljamo S3 "lifecycle policies", ki samodejno arhivirajo stare verzije. Po npr. enem mesecu se stare verzije datotek izbrišejo. Operativno s tem zagotavljamo:

- nadzorovano porabo prostora,
- samodejno čiščenje starih različic,
- ločevanje aktivnih in arhiviranih podatkov brez ročnega posega.

5.3. Mikroservisna integracija DuckDB

Ena izmed osrednjih značilnosti arhitekture je uporaba DuckDB kot vgrajenega (embedded) SQL pogona znotraj posameznih mikroservisov. Ta integracija omogoča, da se podatkovna poizvedba izvede neposredno na S3 v okviru aplikacije, brez potrebe po vmesnih bazah ali ločenih odložiščih podatkov.

Za branje podatkov z objektne shrambe (S3) mikroservisi dinamično nastavijo S3 parametre, ki vključujejo dostopne ključe, regijo in ciljno končno točko (endpoint). Po tem je mogoče brati Parquet datoteke neposredno prek "read_parquet" ali "parquet_scan".

Začasne tabele in podpora za začasne datoteke (ang. spill to disk)

DuckDB omogoča ustvarjanje začasnih tabel (CREATE TEMP TABLE), ki se pogosto uporabljajo za pripravo vmesnih rezultatov, združevanje podatkov iz več virov ali zaporedno izvajanje več transformacij.

```
CREATE TEMP TABLE filtered AS (
  SELECT * FROM parquet_scan('s3://...') WHERE active = true
);
```

V primeru, ko količina podatkov preseže razpoložljiv pomnilnik, DuckDB samodejno aktivira mehanizem "spill to disk", pri čemer vmesne rezultate shranjuje v začasno mapo.

Konfigurabilnost in parametri za nadzor porabe virov

Arhitektura je zasnovana tako, da omogoča enostavno prilagajanje različnim operativnim okoljem in obremenitvam, hkrati pa podpira postopno širitev funkcionalnosti z minimalnim posegom v obstoječi sistem. Ta lastnost je ključna za dolgoročno vzdržnost in prilagodljivost platforme.

Vsak mikroservis, ki izvaja DuckDB poizvedbe, ima možnost dinamične konfiguracije virov, kar omogoča fino uravnavanje delovanja glede na zahteve okolja ali naloge. Tipične nastavitve vključujejo:

```
SET memory_limit='4GB';             -- maksimalna količina RAM-a
SET threads=4;                      -- število sočasnih niti za izvajanje
SET max_temp_directory_size='20GB'; -- meja za uporabo diska
SET temp_directory='/tmp/duckdb';   -- pot do začasnih datotek
```

V kombinaciji z možnostjo konfiguracije DuckDB parametrov na nivoju aplikacije ali okolja (npr. prek .env ali Kubernetes ConfigMap) to omogoča centralizirano upravljanje zmogljivosti brez sprememb v kodi.

Podpora za različna okolja

Celoten sistem podpira več okolij (npr. produkcija, razvoj, testiranje) z uporabo okoljsko pogojenih konfiguracij. Parametri, kot so dostopni ključi, končne točke podatkovne shrambe in poti do izhodnih map, se določijo prek okoljskih spremenljivk, kar omogoča:

- izolacijo med okolji,
- lokalni razvoj brez povezave do oblaka (npr. v testnih okoljih namesto S3 uporabljamo LocalStack),
- konsistentno izvajanje testov z uporabo nadzorovanih virov.

6 Zaključek

Z razvojem lastne arhitekture za obdelavo naborov podatkov neposredno na objektni shrambi smo v praksi potrdili, da je možno zgraditi zmogljiv analitični sistem tudi brez uporabe tradicionalnih podatkovnih skladišč ali kompleksnih distribuiranih rešitev. Ključni element uspeha je bila odločitev za arhitekturo, ki združuje: stolpčno shranjevanje (Parquet), vgrajeno obdelavo (DuckDB) in poenostavljeno operativno plast v Kubernetes okolju.

Med razvojem in uvedbo smo se osredotočili na tri kritične cilje:

operativno učinkovitost (minimalna kompleksnost upravljanja),
odpornost na spremembe shem (razvoj sistema za zaznavanje in prilagoditev),
hitro odzivnost v aplikacijah, kjer poizvedbe pogosto izhajajo iz konteksta uporabnika.

Arhitektura se je izkazala kot stabilna tudi pri večjih količinah podatkov, zlasti zaradi možnosti obdelave, ki presega pomnilnik (spill to disk), in zaradi sposobnosti dinamičnega skaliranja prek Kubernetes. Hkrati nam modularna zasnova omogoča postopno širitev sistema brez potrebe po nadgradnji celotne infrastrukture.

Izkušnje, pridobljene pri gradnji tega sistema, kažejo, da lahko tudi relativno majhna in osredotočena rešitev, če je pravilno zasnovana, pokrije večino sodobnih analitičnih potreb – od verzioniranja, transformacij, dinamičnih poizvedb pa vse do integracije z uporabniškimi aplikacijami.

Literatura

[1] Apache Parquet Format Specification. GitHub, https://github.com/apache/parquet-format, obiskano 24.7.2025

[2] Apache Arrow documentation. https://arrow.apache.org/docs/python/parquet.html#filters-predicate-pushdown, obiskano 24.7.2025

[3] DuckDB httpfs extension. https://duckdb.org/docs/stable/core_extensions/httpfs/overview, obiskano 24.7.2025

[4] DuckDB Memory tuning guide. https://duckdb.org/docs/stable/guides/performance/how_to_tune_workloads.html#larger-than-memory-workloads-out-of-core-processing, obiskano 24.7.2025

[5] DuckDB Execution format. https://duckdb.org/docs/stable/internals/vector.html, obiskano 24.7.2025

[6] DuckDB blog. https://motherduck.com/blog/announcing-duckdb-13-on-motherduck-cdw/, obiskano 24.7.2025

[7] DuckDB SQL documentation. https://duckdb.org/docs/stable/sql/introduction, obiskano 24.7.2025