# 2. laboratorijska vježba

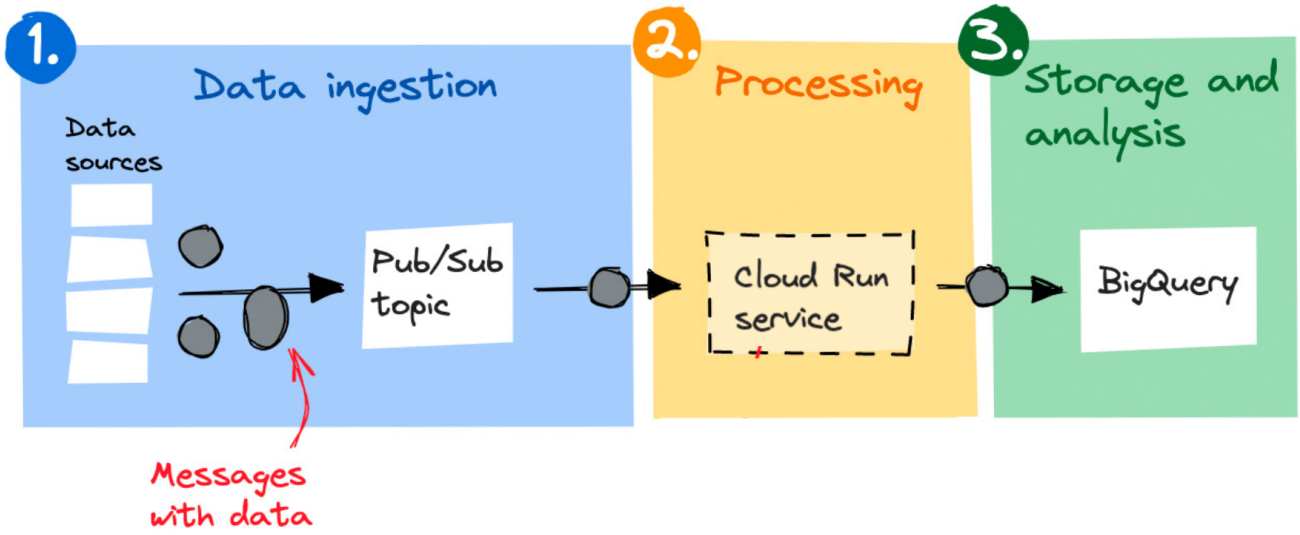


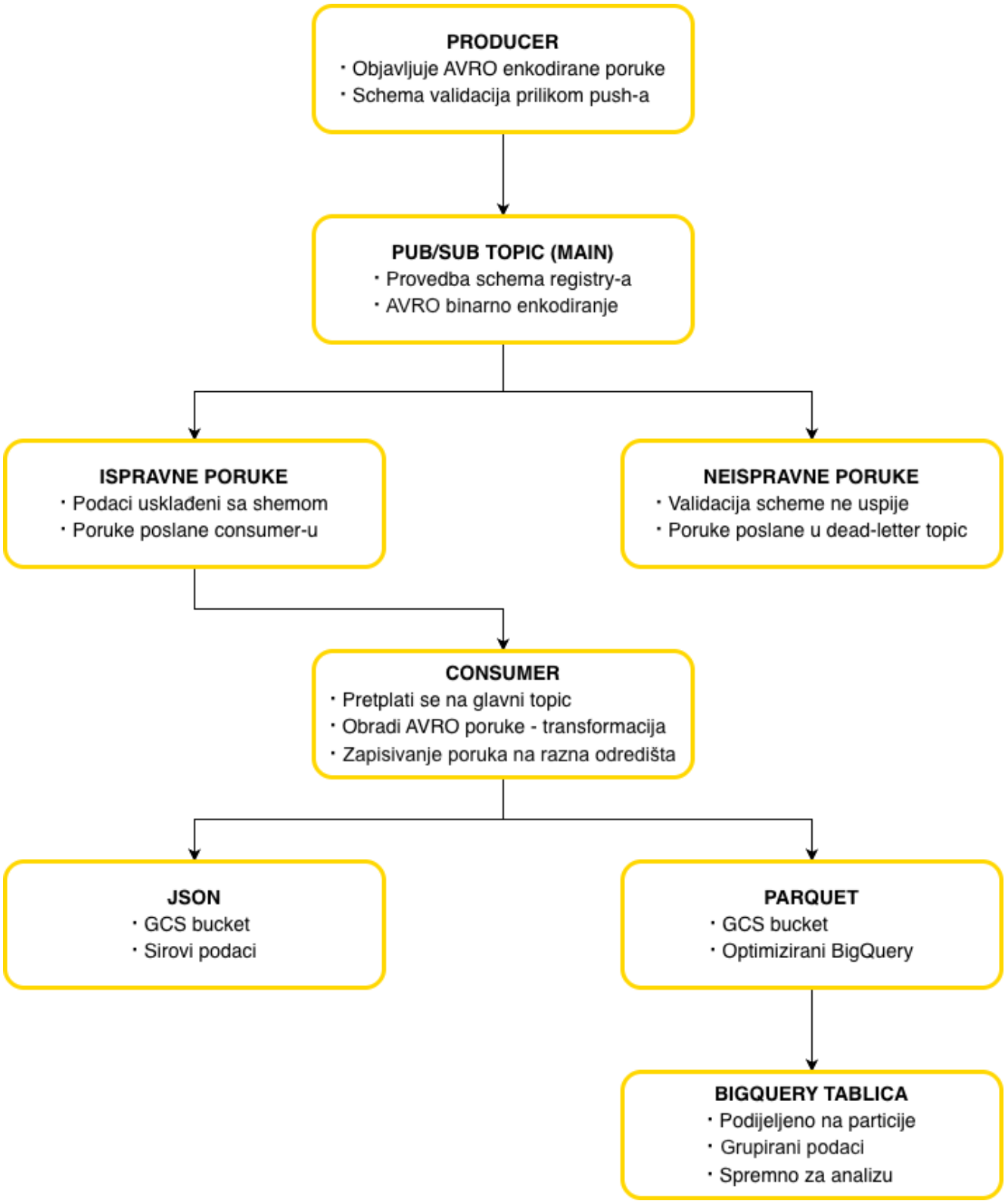
## Uvod

U prethodnoj vježbi implementirali smo producer i consumer aplikacije koje komuniciraju putem Pub/Sub-a i deployane su na Cloud Run. U ovoj vježbi nadograđujemo taj sustav dodavanjem novih komponenti koje omogućuju pohranu, validaciju, sinkronizaciju i automatizaciju podataka u sklopu data streaming pipelinea.

**Pregled arhitekture**

Arhitektura dijela pipeline-a u ovoj vježbi prikazana je na slici 1:






*Slika 1. Arhitektura Data Ingestion Pipeline-a: Python Producer → Pub/Sub (with Schema) → Python Consumer → GCS (Parquet/JSON) → BigQuery, Dead-letter Topic*

**Cilj ove laboratorijske vježbe je:**

- definirati i primijeniti schema registry nad Pub/Sub topicom,
- implementirati odredišta za pohranu podataka u Cloud Storage (GCS) i BigQuery,
- postaviti CI/CD pipeline koji automatski deploya promjene koda,
- osigurati validaciju poruka i testirati dead-letter topic mehanizam.

> **Napomena**: Neke usluge Google Cloud Platforma (GCP) mogu biti blokirane na Eduroam mreži. Ako naiđete na probleme s mrežom ili pristupom, preporuča se prebacivanje na alternativnu internetsku vezu (npr. mobilni podaci ili drugi Wi-Fi).

# GCP tehnologije i koncepti

## Schema Registry

Schema Registry je komponenta unutar Pub/Sub-a koja omogućuje definiranje, verzioniranje i validaciju strukture poruka. GCP Schema Registry može raditi s Avro ili Protocol Buffer formatom, dok neki drugi schema registriji često podržavaju i druge formate (JSON, CSV, i sl.). Svaka poruka koja se šalje na Pub/Sub topic može biti automatski provjerena prema definiranoj schemi, čime se osigurava dosljedan format podataka i smanjuje mogućnost grešaka kod consumer-a. Svrha u pipeline-u:

- Osigurava da sve poruke imaju dosljednu strukturu, što sprječava greške kod consumer-a.
- Omogućuje kontrolirane promjene schema kroz verzioniranje, što znači da nove verzije poruka mogu postojati bez prekidanja postojećih consumers-a.
- Povezan je s dead-letter topic-om, tako da se nevaljane poruke automatski preusmjeravaju na obradu pogrešaka.

Kako se koristi:

1. Kreira se schema na GCP-u.
2. Producer šalje poruke; Pub/Sub provjerava jesu li u skladu sa schema.
3. Nevaljane poruke idu u dead-letter topic, a valjane dalje u namjenjeni topic.

O Avro i Protocol Buffers (Protobuf) formatu možete pronaći više na linkovima:

- Apache Avro – What is Avro?: Big Data File Format Guide
- Protocol Buffers – Overview (official)

---

## Dead-letter Topic

Dead-letter Topic je poseban Pub/Sub topic u koji se šalju poruke koje nisu prošle validaciju ili su doživjele grešku pri obradi.

Svrha u pipeline-u:

- Spriječava gubitak podataka.
- Omogućuje naknadnu analizu pogrešaka i debugiranje.
- Pomaže u testiranju i demonstraciji kako pipeline reagira na nevalidne poruke.

Kako se koristi:

- Prilikom kreiranja Pub/Sub topic-a ili schema-e definira se dead-letter topic.
- Consumer ili Pub/Sub automatski preusmjerava neispravne poruke.

**Kreiranje scheme i dead-letter topica:**

**Kreiranje scheme:**

- Otvorite Google Cloud Console -> Pub/Sub -> Schemas -> Create Schema.
- Unesite ime scheme i odaberite format (AVRO).
- Definirajte polja i tipove podataka prema poruci koju šalje produceru skladu s očekivanim podacima. Primjer AVRO schema:

```
{
 "type" : "record",
 "name" : "Avro",
 "fields" : [
   {
     "name" : "StringField",
     "type" : "string"
   },
   {
     "name" : "FloatField",
     "type" : "float"
   },
   {
     "name" : "BooleanField",
     "type" : "boolean"
   }
 ]
}
```

- Ili pomocu komande preko gcloud CLI:

```
gcloud pubsub schemas create <ime scheme> \
    --type=avro \
    --definition='schema.avsc'
```

**Kreiranje i povezivanje dead-letter topic-a**:

- Otvorite Google Cloud Console - Pub/Sub -> Topics -> Create Topic te kreirajte deadletter topic i subscription na taj topic
- Ili pomoću naredbi:

```
gcloud pubsub topics create <ime_deadletter_topica>

gcloud pubsub subscriptions create <ime_deadletter_sub> \
    --topic=<ime_deadletter_topic> \
    --project=<project_id>
```

- Povežite dead-letter topic s glavnim topic-om (koji ste kreirali u prošloj laboratriijskoj vježbi, možete kreirati i novi):
  - Otvorite topic na koji će producer slati poruke
  - Kliknite Edit -> sekcija Dead-letter topic.

    - Odaberite ranije kreirani dead-letter topic
    - Postavite max delivery attempts; broj pokušaja dostave poruke prije nego se prebaci u dead-letter topic (preporuka: 5) i spremite promjene
    - Ili pomoću naredbe:

```
gcloud pubsub subscriptions update <topic_subscription_ime> \
--dead-letter-
topic=projects/<project_id>/topics/<ime_deadletter_topic> \
--max-delivery-attempts=5
```

    - Zamijenite imena u <> da odgovaraju imenima vaših resursa

**Povezivanje schema s topic-om**:

- U istom topic-u odaberite Schema Settings -> Attach existing schema.
- Odaberi željenu schemu i način validacije:
    - *Validate messages against schema* - sve poruke moraju biti u skladu sa schemom - to mi želimo
    - *Allow messages that do not match schema* - opcionalno, manje strogo.
- Ili pomoću naredbe:

```
gcloud pubsub topics update <ime_topic> \
    --schema=<ime_schema> \
    --message-schema-options=REJECT_INVALID
```

Kada producer šalje poruku:

- Ako je validna -> ide na regularni topic i consumer je prima.
- Ako nije validna -> automatski ide u dead-letter topic

---

## Cloud Storage (GCS)

**Google Cloud Storage** je objektno orijentirani servis za pohranu datoteka u oblaku. Omogućuje sigurno, skalabilno i trajno spremanje bilo koje vrste podataka. Podaci se organiziraju u bucket-e (spremnike), koji se mogu strukturirati pomoću foldera i povezati s drugim Google Cloud servisima kao što su Pub/Sub, Dataflow, BigQuery i Vertex AI.

Svrha u pipeline-u

- U sklopu vašeg podatkovnog pipeline-a, GCS ima ključnu ulogu kao sloj za trajno i organizirano pohranjivanje obrađenih podataka. Njegove glavne funkcije uključuju:
    - Trajnu pohranu poruka u Parquet formatu, što omogućuje efikasno čitanje, kompresiju i analizu podataka.
    - Particioniranje podataka po vremenskim atributima (year/month/day/hour), čime se olakšava filtriranje i izvođenje vremenski ograničenih upita u BigQuery-u.

    - Služi kao izvor podataka za BigQuery - podaci se mogu čitati direktno iz GCS-a (kroz external table) ili se mogu učitati u BigQuery dataset za trajnu analizu.

Kako se koristi:

- Kreira se bucket (npr. gcs-pipeline-data).
- Consumer sprema poruke u Parquet datoteke u bucketu, particionirane po vremenu.
- Podaci se mogu dalje učitati u BigQuery ili koristiti direktno iz GCS-a.

**Kreiranje bucket-a**

- Otvorite Google Cloud Console -> Cloud Storage -> Create Bucket, unesite ime bucket-a, region i storage class Standard
- Kreiranje bucket-a pomoću CLI-a

```
gcloud storage buckets create gs://gcs-reddit-data \
    --location=<regija_projekta> \
    --default-storage-class=STANDARD
```

- gs://gcs-reddit-data - URI bucket-a.
- --location - region gdje će se podaci fizički pohraniti.
- --default-storage-class - klasa pohrane (Standard, Nearline, Coldline… ).

**Povezivanje bucket-a s pipeline-om**:

- Consumer u pipeline-u sprema obrađene poruke iz Pub/Sub-a u Parquet format.
- Preporučena struktura za particioniranje po vremenu: gs://gcs-reddit-data/year=YYYY/month=MM/day=DD/hour=HH/part-XXXXX.parquet
    - year, month, day, hour - direktoriji za particioniranje po vremenu.
    - part-XXXXX.parquet - Parquet fajl s porukama obrađenim od consumer-a.
    - Ovo omogućuje jednostavno filtriranje podataka po vremenu u BigQuery-u.
- Primjer: gs://gcs-pipeline-data/topic/year=2025/month=10/day=23/hour=15/part-0001.parquet
- Ovaj dio ćete definirati u kodu, odnosno consumer-u

**Parquet Format**

Parquet je format pohrane podataka razvijen za potrebe analitičkih sustava i velikih količina podataka. Za razliku od tradicionalnih redno orijentiranih formata (npr. CSV), Parquet pohranjuje podatke po stupcima, što donosi brojne prednosti:

- Manja potrošnja prostora; Parquet koristi naprednu kompresiju i enkodiranje po stupcima.
- Brže čitanje podataka - pri izvođenju upita, čitaju se samo stupci koji su potrebni.
- Bolja integracija s analitičkim alatima - podržan je u BigQuery-u, Spark-u, Pandas-u i mnogim drugim alatima.
- Samostalna schema; svaka Parquet datoteka sadrži vlastitu definiciju strukture podataka (nazivi stupaca, tipovi podataka itd.), što olakšava interoperabilnost i validaciju.

U kontekstu pipeline-a, Parquet omogućuje da se podaci iz Pub/Sub-a trajno pohrane na učinkovit način, bez gubitka strukture, i da se kasnije brzo analiziraju u BigQuery-u ili drugim alatima za analitiku.

> **Napomena**: Preporučuje se odvojiti bucket za raw/topic podatke kako bi pipeline bio pregledan i lako održiv.

---

## BigQuery

**BigQuery** je fully managed serverless alat za pohranu i analizu podataka (data warehouse) unutar Google Cloud platforme. Omogućuje rad s ogromnim količinama podataka pomoću SQL-a, bez potrebe za upravljanjem infrastrukturom. BigQuery koristi distribuiranu arhitekturu koja omogućuje paralelnu obradu podataka i automatsko skaliranje, što znači da se upiti izvršavaju brzo, bez obzira na veličinu podataka.

Svrha u pipeline-u:

- BigQuery služi kao analitički sloj sustava, mjesto gdje završavaju obrađene poruke spremne za analizu i izvještavanje. Njegove glavne funkcije uključuju:
    1. Analitičko skladište podataka (Data Warehouse)
    - Centralizirano pohranjuje podatke prikupljene iz različitih izvora (Pub/Sub, GCS, API-i).
    2. Brza analiza podataka
    - Omogućuje izvođenje kompleksnih SQL upita nad terabajtima podataka u sekundama, što je ključno za praćenje tokova podataka i donošenje odluka u stvarnom vremenu.
    3. Integracija s GCS-om
    - Podaci spremljeni u GCS-u (Parquet format) mogu se: a. Čitati direktno kroz External Table (bez kopiranja) b. Fizički učitati u BigQuery tablicu kroz Load Job
    4. Particioniranje i klasteriranje
    - Optimizira performanse upita i smanjuje troškove, jer se upiti izvršavaju samo nad relevantnim dijelovima podataka (npr. po datumu).

> **Napomena**: U ovoj laboratorijskoj vježbi podaci spremljeni u GCS-u (Parquet format) će se učitati u BigQuery tablicu za trajnu pohranu i analizu. Nećemo koristiti external table, već fizičko učitavanje podataka putem load job-a kako bi svi podaci bili dostupni unutar BigQuery-a.

**Kreiranje BigQuery dataset-a i tablice**

Dataset je logička grupa tablica unutar BigQuery-a (slično kao baza u tradicionalnim bazama podataka).

Otvorite BigQuery Console → Create Dataset → unesite naziv dataset-a (npr. `reddit_pipeline`) → odaberite region → Create Dataset.

- Unutar dataset-a kreirajte tablicu (npr. reddit_messages) koja odgovara strukturi podataka s Reddita (id, title..).

- Ili pomoću CLI:

```
bq --location=<REGION> mk --dataset <PROJECT_ID>:<DATASET_NAME>

bq mk --table <PROJECT_ID>:<DATASET_NAME>.<TABLE_NAME>
```

> **Napomena** Definicija kolona tablice može se također definirati direktno u kodu ako se učitavanje radi programski.

**Load Job (učitavanje podataka u BigQuery)**

Odredište omogućuje fizičko učitavanje podataka iz GCS bucket-a u BigQuery tablicu, što je preporučeni način za trajnu pohranu i analizu. U ovoj laboratorijskoj vježbi odredište se definira **u consumer kodu**, koristeći Python BigQuery klijent (`google.cloud.bigquery`) Consumer sprema poruke u GCS u Parquet formatu, particionirane po vremenskim atributima (year/month/day/hour), a BigQuery Load Job učitava cijele Parquet fajlove, a ne pojedinačne redove, što je učinkovitije i skalabilnije za streaming/batch pipeline.

Primjer korištenja:

```
from google.cloud import bigquery

bq_client = bigquery.Client()
table_id = "<PROJECT_ID>.<DATASET_NAME>.<TABLE_NAME>"

gcs_uri =
"gs://<BUCKET_NAME>/parquet_reddit/year=2025/month=11/day=20/hour=15/part-
123456.parquet"

job_config = bigquery.LoadJobConfig(
    source_format=bigquery.SourceFormat.PARQUET,
    write_disposition=bigquery.WriteDisposition.WRITE_APPEND,
    schema_update_options=[
        bigquery.SchemaUpdateOption.ALLOW_FIELD_ADDITION,
        bigquery.SchemaUpdateOption.ALLOW_FIELD_RELAXATION,
    ],
)


load_job = bq_client.load_table_from_uri(gcs_uri, table_id,
job_config=job_config)
load_job.result()

print(f"Loaded Parquet from GCS to BigQuery: {gcs_uri} -> {table_id}")
```

- Podaci se automatski učitavaju nakon obrade.
- Omogućuje transformaciju i validaciju prije učitavanja.
- Ručno kreiranje dataset-a i tablice je opcionalno

Nakon ovog koraka, podaci su trajno pohranjeni u BigQuery tablici i spremni za SQL upite, izvještavanje ili integraciju s BI alatima.

**External Table (opcionalno)**

External Table omogućuje da BigQuery čita podatke direktno iz GCS-a, bez fizičkog učitavanja.
U BigQuery Console-u:

- Otvori dataset → Create Table
- Source: Google Cloud Storage
- URI: `gs://gcs-pipeline-data/year=*/month=*/day=*/hour=*/part-*.parquet`
- File format: Parquet
- Destination: External Table -> naziv tablice npr. reddit_messages_external
- Kliknite Create Table

Pomoću CLI:

```
bq mkdef \
  --source_format=PARQUET \
  "gs://gcs-data-data/year=*/month=*/day=*/hour=*/part-*.parquet" >
table_def.json

bq mk \
  --external_table_definition=table_def.json \
  reddit_pipeline.reddit_messages_external
```

> **Napomena**: External Table odmah reflektira nove podatke u GCS-u, ali ne pohranjuje ih trajno u BigQuery.

---

## CI/CD (Continuous Integration / Continuous Deployment)

CI/CD je praksa u software development koja omogućuje automatizaciju builda, testiranja i deploya aplikacija. Ova praksa osigurava stabilnost koda, brže iteracije i smanjuje rizik od pogrešaka u produkciji.

### Continuous Integration (CI)

CI predstavlja stalno spajanje i testiranje koda:

- Svaka promjena u repozitoriju automatski se builda i testira.
- Greške se odmah detektiraju, prije nego što dođu u produkciju.
- Omogućuje održavanje kvalitetnog i stabilnog koda.

### Continuous Deployment (CD)

CD označava stalno automatsko deployanje:

- Nakon uspješnog CI procesa, nova verzija aplikacije automatski se deploya na odabranu platformu (npr. Cloud Run).
- Omogućuje brze iteracije i stalno osvježavanje produkcijske verzije aplikacije.

### Prednosti CI/CD-a

- Brže i sigurnije uvođenje promjena.
- Automatsko testiranje i validacija koda.
- Manje ručnih grešaka u deployu.
- Transparentnost i praćenje verzija.

### CI/CD u laboratorijskoj vježbi

U ovoj vježbi koristimo GitHub Actions za implementaciju CI/CD pipeline-a. Pipeline:

- Automatski builda i deploya producer i consumer aplikacije na Cloud Run.
- Osigurava da se najnovija verzija koda koristi za odredišta u GCS i BigQuery.
- Omogućuje testiranje promjena prije nego što postanu dio produkcijskog pipeline-a.

### Preduvjeti

1. GitHub račun i repozitorij za projekt.
2. Organizacija koda (primjer): /producer Dockerfile main.py /consumer Dockerfile main.py
3. Aktiviran Cloud Run i projekt na GCP (iz prošle laboratorijske vježbe).
4. Service Account s rolama za Cloud Run, GCS i BigQuery (koristite servisni račun koji ste generirali u prošloj laboratorijskoj vježbi).

- ako nema potrebne role za izvršavanje svih akcija potrebno ih je dodati

### Kreiranje GitHub Actions workflow-a

- Generirajte folder .github/workflows u root-u vašeg projekta.
- Dodajte ci-cd.yaml
- Primjer workflowa kojeg će biti potrebno nadopuniti i korigirati da odgovara vašoj implementaciji:

```
name: Consumer & Producer Pipeline

on:
  <action>:
    branches: [...]  # Na neku akciju (push na granu, pull request...) u
ovim granama pokreće se workflow


jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      # 1. Checkout repository
      # Radi checkout koda iz repozitorija
      - uses: actions/checkout@v3


      # 2. Authenticate to Google Cloud
      - name: Authenticate to GCP
        # Prijava u Google Cloud koristeći secrets ili service account
        uses: google-github-actions/auth@v1
        with:
          credentials_json: ${{ secrets.GCP_SA_KEY }}

      # 3. Set up Cloud SDK
```

```
      - name: Set up Cloud SDK
        uses: google-github-actions/setup-gcloud@v1
        # Instalira GCP SDK ako već nije instaliran

      # 4. Configure Docker
      - name: Configure Docker for Artifact Registry
        # Konfigurira Docker za push i pull iz Artifact Registry
        run: |
          # gcloud auth configure-docker ...

      # 5. Extract environment variables
      - name: Extract environment variables
        # Čitanje potrebnih varijabli iz .env datoteka ili sličnog izvora
        id: extract_env
        run: |
          # PROJECT_ID = ...
          # OTHER_ENV_VARS = ...
          # npr.:
          PROJECT_ID=$(grep 'PROJECT_ID:' ./lab-2/lab2-rjesenje/.env.yaml
| cut -d ' ' -f2 | tr -d '"')
          echo "project_id=$PROJECT_ID" >> $GITHUB_OUTPUT


      # ----- Consumer -----
      - name: Build Consumer Docker image
        # Docker build za Consumer
        run: |
          # docker build -t <consumer-image> .

      - name: Push Consumer Docker image
        # Push Docker image-a u Artifact Registry
        run: |
          # docker push <consumer-image>

      - name: Deploy Consumer to Cloud Run
        # Deploy Consumer servisa na Cloud Run
        run: |
          # gcloud run deploy <consumer-service> --image <consumer-image>
--env-vars-file <env-file>

      # ----- Producer -----
      - name: Build Producer Docker image
        # Docker build za Producer
        run: |
          # docker build -t <producer-image> .

      - name: Push Producer Docker image
        # Push Producer image-a u Artifact Registry
        run: |
          # docker push <producer-image>

      - name: Deploy Producer Job to Cloud Run
        # Deploy Producer Job-a
        run: |
```

```
          # gcloud run jobs deploy <producer-job> --image <producer-image>
--env-vars-file <env-file>

      - name: Execute Producer Job
        # Pokretanje Producer Job-a
        run: |
          # gcloud run jobs execute <producer-job>
```

**GitHub Secrets**

U svrhu CI/CD, svi osjetljivi podaci koji se koriste za deploy i izvršavanje aplikacija ne smiju biti javno dostupni u repozitoriju. To uključuje:

- ID GCP projekta
- JSON ključ Service Account-a s rolama za Cloud Run, GCS i BigQuery
- Ostale osjetljive varijable koje aplikacija koristi

Kako bi se osigurala sigurnost i automatski deploy, ove vrijednosti se pohranjuju u **GitHub Secrets**. GitHub Secrets omogućuju:

- **Sigurno pohranjivanje tajni** bez izlaganja u verzioniranom kodu.
- **Automatsko korištenje u workflow-u** kroz CI/CD pipeline, npr. za autentikaciju u Google Cloud-u ili konfiguriranje environment varijabli.
- **Centralizirano upravljanje** vrijednostima, što olakšava izmjene i rotaciju ključeva.

**Potrebno je postaviti sljedeće GitHub Secrets:**

- `GCP_PROJECT` → ID GCP projekta
- `GCP_SA_KEY` → JSON ključ Service Account-a s permissionima za Cloud Run, GCS i BigQuery

**Kako postaviti GitHub Secrets:**

1. Otvorite repozitorij na GitHub-u.
2. Idite na **Settings → Secrets → Actions**.
3. Kliknite **New repository secret**.
4. Unesite ime (npr. `GCP_PROJECT`) i vrijednost (ID projekta ili JSON key).
5. Spremite secret.
6. Ponavljajte postupak za sve potrebne varijable.

> **Važno:** Ni u kojem slučaju se vrijednosti iz Secrets ne smiju dodavati direktno u repozitorij ili verzionirani kod. CI/CD workflow automatski učitava te vrijednosti i koristi ih tijekom builda, push-a i deploy procesa.

**Korištenje `.env.yaml` i tajnih vrijednosti u laboratorijskoj vježbi**

- `.env.yaml` služi za pohranu **konstantnih ili konfiguracijskih vrijednosti** koje su potrebne za rad aplikacije ili CI/CD pipeline-a (npr. URI bucket-a, ID tablice, defaultni parametri).
- **Osjetljive vrijednosti (tajni ključevi, ID projekata, API ključevi)** ne smiju biti pohranjene u `.env.yaml` ako se datoteka verzionira.

- Sve tajne vrijednosti koje aplikacija koristi prilikom rada (npr. pristup GCS-u, BigQuery-u, Pub/Sub-u) pohranjuju se **u GCP Secret Manager-u** kao što smo to radili u prethodnoj laboratorijskoj vježbi.
- `.env.yaml` može sadržavati reference na ove tajne (npr. ime sekreta ili ID), ali ne i stvarne vrijednosti.
- Za potrebe automatiziranog builda, deploya i konfiguracije pipeline-a, workflow koristi **GitHub Secrets** kao što je navedeno u prethodnom poglavlju
- CI/CD workflow uvozi iz `.env.yaml` samo **vrijednosti koje su potrebne za build ili deploy**, dok sve osjetljive informacije dolaze iz GitHub Secrets.

**Pylint**

Pylint je alat za statičku analizu Python koda koji provjerava sintaksu, stil i moguće runtime greške. Njegova svrha je osigurati dosljedan i kvalitetan kod, što smanjuje rizik od grešaka prilikom deploya. Trebat ćete ga koristiti unutar CI/CD pipeline-u te u provjeri koda lokalno.

**Kako se koristi u VSCode-u:**

- Instalirajte pylint: pip install pylint
- Pokrenite pylint u terminalu, unutar file koji želite ispitati, naredbom **pylint .**
- Kod s greškama i upozorenjima bit će ispisan direktno u terminalu.

**Kako se koristi u CI/CD skripti:**

```
- name: Install dependencies
  run: pip install pylint

- name: Run pylint
  run: pylint lab-2/lab2-rjesenje/consumer/
```

- Ako pylint detektira kritične greške, workflow fail-a, što sprječava deploy dok se greške ne isprave.

**EditorConfig**

EditorConfig je standard koji definira dosljedno formatiranje koda (indentacija, završetak linija, spacing, utf-8 encoding) unutar tima. Pomaže da svi članovi tima pišu kod u istom stilu, bez obzira na IDE ili editor.

**Kako se koristi u VSCode-u:**

- Instalirajte EditorConfig plugin za VSCode
- Dodajte .editorconfig datoteku u root projekta s pravilima formatiranja:

```
root = true

[*]
indent_style = space
indent_size = 4
end_of_line = lf
charset = utf-8
```

```
trim_trailing_whitespace = true
insert_final_newline = true
```

- VSCode automatski primjenjuje ove postavke na otvorene datoteke.

**Kako se koristi u CI/CD skripti:**

- Dodajte job koji provjerava formatiranje prema EditorConfig standardu, npr. pomoću editorconfig-checker:

```
- name: Install EditorConfig
  run: pip install editorconfig
- name: Check EditorConfig
  run: editorconfig-check .
```

- Ako alat pronađe greške u formatiranju, job će failati i build neće ići dalje

# 1. Zadatak

Cilj ovog zadatka je postaviti CI/CD pipeline za automatski build i deploy producer-a i consumer-a na Cloud Run pomoću GitHub Actions, s jasnom separacijom Continuous Integration (CI) i Continuous Deployment (CD) faza prema *"best practice"* principima.

## Opis rješenja

Implementirajte dvije odvojene GitHub Actions workflow datoteke koje će:

1. CI faza - Provjeriti kvalitetu koda prije nego što se promjena spoji u glavnu granu
2. CD faza - Deployati aplikaciju tek nakon što su promjene pregledane i odobrene

## Koraci implementacije

### 1. Postavljanje GitHub Secrets

U GitHub repozitoriju postavite sljedeće tajne (secrets):

- GCP_PROJECT - ID vašeg GCP projekta
- GCP_SA_KEY - JSON ključ Service Account-a s rolama:
  - Cloud Run Developer
  - Storage Admin
  - BigQuery Data Editor

### 2. Kreiranje CI workflow-a (.github/workflows/ci.yaml)

Workflow se treba pokrenuti na pull request prema glavnoj grani (main) i sadržavati:

- Checkout repozitorija
- Setup Google Cloud SDK (samo za autentikaciju ako je potrebno)
- Lint provjera pomoću pylint-a

- Provjera EditorConfig standarda
- Build test - izgradnja Docker image-ova (samo lokalno, bez push-a)
- Unit testovi (ako postoje)

```
name: Continuous Integration (CI)
on:
  pull_request:
    branches: [ main, master ]
```

**3. Kreiranje CD workflow-a (.github/workflows/cd.yaml)**

Workflow se treba pokrenuti na push na glavnu granu (main) i sadržavati:

- Checkout repozitorija
- Setup Google Cloud SDK s autentikacijom
- Build i push Docker image-ova za producer i consumer u Container Registry
- Deploy consumer kod kao Cloud Run Service
- Deploy producer kod kao Cloud Run Job
- Execute producer job

```
name: Continuous Deployment (CD)
on:
  push:
    branches: [ main, master ]
```

**4. Zaštita glavne grane (GitHub Branch Protection Rules)**

Postavite pravila za glavnu granu:

- Require a pull request before merging - obavezan PR prije merge-a
- Require status checks to pass before merging - obavezno prolazak CI provjera
- Include required CI jobs - označite pylint i editorconfig-check kao required checks
- Require at least 1 approval - minimalno 1 review prije merge-a
- (Preporučeno) Do not allow bypassing the above settings - blokiraj za admine
- (Opcionalno) Include administrators - primjeni pravila i na admine

**5. Testiranje implementacije**

1. Kreirajte feature granu i napravite promjenu u kodu
2. Otvorite pull request prema main grani - trebao bi se pokrenuti CI workflow
3. Provjerite prolazi li CI ispravno
4. Zatražite review i dobijte approval
5. Merge-ajte PR u main granu - trebao bi se pokrenuti CD workflow
6. Provjerite je li deploy uspješno obavljen na Cloud Run-u

**Očekivani rezultat**

- CI faza se pokreće automatski na svaki pull request
- Deploy se događa samo kada se promjena spoji u main granu (nakon odobrenog PR-a)
- Direktan push na main je onemogućen ili zahtijeva override
- Sve promjene su pregledane i testirane prije deploya u produkciju
- Pipeline slijedi profesionalne standarde razdvajanja CI i CD

### Napomene

- Ova struktura omogućuje ranu detekciju problema u CI fazi
- Samo provjeren i odobren kod dolazi do produkcije
- Jasna separacija odgovornosti između razvoja i deploya
- Pobošljavanje kvalitete koda kroz obavezne code reviews
- Standardna praksa u modernim DevOps timovima

## 2. Zadatak

Tema ovog zadatka je nadogradnja prethodno razvijenog Reddit pipeline-a dodavanjem validacije poruka, trajne pohrane i izlaznih modula za analitiku. Cilj je da sve poruke proizvedene od strane producera budu validirane prema definiranoj schemi, a potom pohranjene u GCS i BigQuery, dok neispravne poruke odlaze u dead-letter topic.

Library-je koji se preporučuju:

- os - za dohvat konfiguracijskih vrijednosti iz .env.yaml
- json
- io - za rad s binarnim podacima AVRO-a
- threading - za paralelno pokretanje consumer-a
- datetime - za timestamp i particioniranje podataka
- pandas - za konverziju podataka u DataFrame, Parquet i BigQuery učitavanje
- fastavro – parse_schema, schemaless_reader za dekodiranje AVRO poruka
- google.cloud.pubsub_v1 – SubscriberClient za primanje poruka
- google.cloud.storage – Client za pohranu u GCS
- google.cloud.bigquery – Client i LoadJobConfig za pohranu u BigQuery
- flask
- logging – za praćenje procesa i grešaka

> **Napomena**: Možete koristiti i druge library-je, ovo je samo ideja. Struktura koda i funkcije navedene u nastavku služe za pomoć, naravno, možete koristiti što god želite i kako želite u svojoj implementaciji sve dok je zadatak u potpunosti riješen u skladu s definicijom.

### Kreiranje i povezivanje Pub/Sub resursa

- Kreirajte AVRO schemu koja odgovara strukturi podataka iz Reddit API-a.
- Kreirajte glavni Pub/Sub topic za primanje poruka od producera (ili koristi postojeći)
- Kreirajte dead-letter topic za neispravne poruke.
- Povežite glavni topic s AVRO schemom i postavi:
    - REJECT_INVALID opcija
    - dead-letter topic kao destinaciju za nevalidne poruke

## Nadogradnja producera

Dohvatite top 10 postova s Reddit-a kao u prethodnoj vježbi, provedite kodiranje JSON u AVRO i pošaljite poruku na Pub/Sub. Testirajte funkcionalnost dead-lettering-a, dodajte ručno **neispravnu poruku** koja ne zadovoljava schema (npr. nedostaje obavezno polje) Tajne dohvaćajte iz Secret Managera (kao u 1. laboratorijskoj vježbi), a konfiguracijske vrijednosti poput PROJECT_ID, TOPIC_ID iz .env.yaml file-a.

Pub/Sub ne podržava direktno JSON validaciju prema schemi, stoga je potrebno poruke enkodirati u AVRO format.

Ideja je:

- Producer dohvaća podatke (u JSON formatu) s Reddita.
- Svaku JSON poruku kodiramo u AVRO binarni format prema definiranoj schemi.
- Ako poruka ne zadovoljava schema, šalje se u dead-letter topic (kao JSON).

Primjer dijelova koda i potrebnih library-ja:

```
import json
import io
from google.cloud import pubsub_v1, secretmanager
from fastavro import schemaless_writer, parse_schema
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Definicija AVRO scheme
reddit_schema_dict = {...}
# Parsiranje
parsed_schema = parse_schema(reddit_schema_dict)
# Enkodiranje JSON -> AVRO
def encode_avro(post_dict):
    bytes_writer = io.BytesIO()
    schemaless_writer(bytes_writer, parsed_schema, post_dict)
    return bytes_writer.getvalue()

# JSON
post_json = {
    "id": "abc123",
    "title": "Example Reddit post",
  ...
}
# Kodiranje
avro_bytes = encode_avro(post_json)
...
```

Nadogradnja funkcije koja šalje poruke na Pub/Sub:

- Kodirajte svaku poruku u AVRO binarni format.
- Ako kodiranje ili publish ne uspije, pošaljite poruku u dead-letter topic.

- Podsjetnik na Pub/Sub funkcije: ...

```
    from google.cloud import pubsub_v1
    publisher = pubsub_v1.PublisherClient()
    topic_path = publisher.topic_path("PROJECT_ID", "TOPIC_ID")
    future = publisher.publish(topic_path, data=avro_bytes)
    message_id = future.result()
    print(f"Published message ID: {message_id}")
...
```

> **Napomena**: Nakon publish-a poruke na Pub/Sub postavite sleep(3) ili više kako nebi došlo do TimeoutError-a na Cloud Run-u

### Nadogradnja consumera

Cilj je osigurati da sve poruke s producera budu validirane i pohranjene, dok neispravne završavaju u dead-letter topicu.

- Primanje poruka s Pub/Sub-a
    - Kreirajte ili koristite postojeću Subscriber instancu za vaš Pub/Sub subscription.
    - Definirajte callback funkciju koja će se pozivati za svaku poruku primljenu s topic-a.
- Dekodiranje AVRO poruka
    - Unutar callback funkcije, dohvatite binarni sadržaj poruke (message.data).
    - Koristite schemaless_reader i prethodno parsiranu AVRO schemu da dekodirate poruku u Python dictionary.
- Provjera grešaka i nack
    - Ako AVRO schema ne prođe, poruka se automatski šalje u dead-letter topic (zbog opcije REJECT_INVALID na topicu).
    - message.nack() pozivate samo ako dođe do runtime greške unutar callback funkcije, npr. problem pri pohrani u GCS ili BigQuery.
- Pohrana poruka u GCS i BigQuery
    - Pohrana u GCS u JSON formatu
    - Pohrana u GCS u Parquet formatu: konvertirajte poruku u Pandas DataFrame, spremite u Parquet format i organizirajte folder strukturu po godini, mjesecu, danu i satu, neka je return iz vaše funkcije s ovom funkcionalnošću path do GCS Bucketa kako biste mogli dohvatiti Parquet file-ove za učitavanje u BigQuery
    - Pohrana u BigQuery: dohvatite odgovarajuće Parquet file-ove i pošaljite ih u BigQuery tablicu koristeći LoadJobConfig

Tajne dohvaćajte iz Secret Managera (kao u 1. laboratorijskoj vježbi), a konfiguracijske vrijednosti poput PROJECT_ID, TOPIC_ID iz .env.yaml file-a

### Dockerizacija

- Dockerizirajte producer i consumer aplikacije putem CI/CD datoteke.
- Provjerite da su environment varijable i GCP credentials pravilno postavljeni kako bi aplikacije mogle pristupiti Pub/Sub-u, GCS-u i BigQuery-u.