

Soluții de implementare a unei aplicații REST API în Google Cloud Platform

1. Descriere tema

Aplicația REST API este realizată în Python și Flask și a fost creată pentru a prelua date despre cărți dintr-o bază de date SQLite3: <https://github.com/cpatrickalves/simple-flask-api>

O aplicație REST API expune funcționalitățile unei aplicații sau a unei baze de date printr-un set de endpoint-uri (adrese URL). Aceste URL-uri sunt accesate de clienți, cum ar fi aplicații web, mobile sau alte servicii web, pentru a efectua operații CRUD (Create, Read, Update, Delete) sau alte acțiuni, folosind cereri HTTP. Principiile REST și utilizarea metodelor HTTP (GET pentru citire, POST pentru creare, PUT/PATCH pentru actualizare și DELETE pentru ștergere) permit manipularea resurselor într-un mod standardizat și intuitiv.

Vom prezenta în continuare câteva soluții de implementare a acestei aplicații în platforma Google Cloud.

2. Descrierea modelului de date utilizat

Baza noastră de date "books.db" conține informații despre cărți. Datele sunt stocate și gestionate utilizând SQLite - un sistem de gestionare a bazelor de date relaționale, fără server și bazat pe fișiere.

id	published	author	title	first_sentence
Filter	Filter	Filter	Filter	Filter
1	2014	Ann Leckie	Ancillary Justice	The body lay naked and facedown, a ...
2	2013	John Scalzi	Redshirts	From the top of the large boulder he ...
3	2012	Jo Walton	Among Others	The Phurnacite factory in Abercwmbo...
4	2011	Connie Willis	Blackout, All Clear (Vol. 2 - Blackout)	By noon Michael and Merope still ...
5	2010	Paolo Bacigalupi	The Windup Girl	"No! I don't want the mangosteen."
6	2010	China Mieville	The City & The City	I could not see the street or much of ...
7	2009	Neil Gaiman	The Graveyard Book	Nobody Owens, known to his friends ...
8	2008	Michael Chabon	The Yiddish Policemen's Union	Nine months Landsman's been floppi...
9	2007	Vernor Vinge	Rainbows End	The first bit of dumb luck came ...
10	2006	Robert Charles Wilson	Spin	One night in October when he was te...

3. Alegerea soluțiilor pentru stocarea datelor, argumentând decizia

Pentru stocarea datelor, am ales să folosim Google Cloud Storage și BigQuery.

- Google Cloud Storage va fi folosit pentru stocarea fișierului books.csv provenit din baza de date SQLite inițială. GCS oferă scalabilitate, durabilitate și disponibilitate ridicată pentru datele noastre, asigurând că acestea sunt accesibile și securizate
- Pentru încărcarea fișierului books.csv și interogarea datelor, am ales BigQuery datorită capacității sale de a gestiona volume mari de date structurate într-un mod eficient și scalabil. BigQuery permite interogări rapide asupra datelor noastre, permițându-ne să obținem informații valoroase din seturile noastre de date.

Google Cloud Storage și BigQuery sunt integrate în ecosistemul Google Cloud Platform, și ne oferă o experiență de dezvoltare și gestionare simplificată, alături de scalabilitate și performanță.

4. Implementarea modelului de date utilizat

Vom folosi App Engine, BigQuery și Google Cloud Storage:

1. utilizatorul trimite o solicitare HTTP către aplicația Flask.
2. app.py preia solicitarea și, dacă este necesar, efectuează interogări către BigQuery.
3. app.py utilizează requirements.txt pentru a asigura că toate bibliotecile necesare sunt disponibile pentru procesarea solicitării.
4. app.yaml joacă un rol în configurarea modului în care aplicația este desfășurată și gestionată pe App Engine, influențând modul în care solicitările sunt tratate.

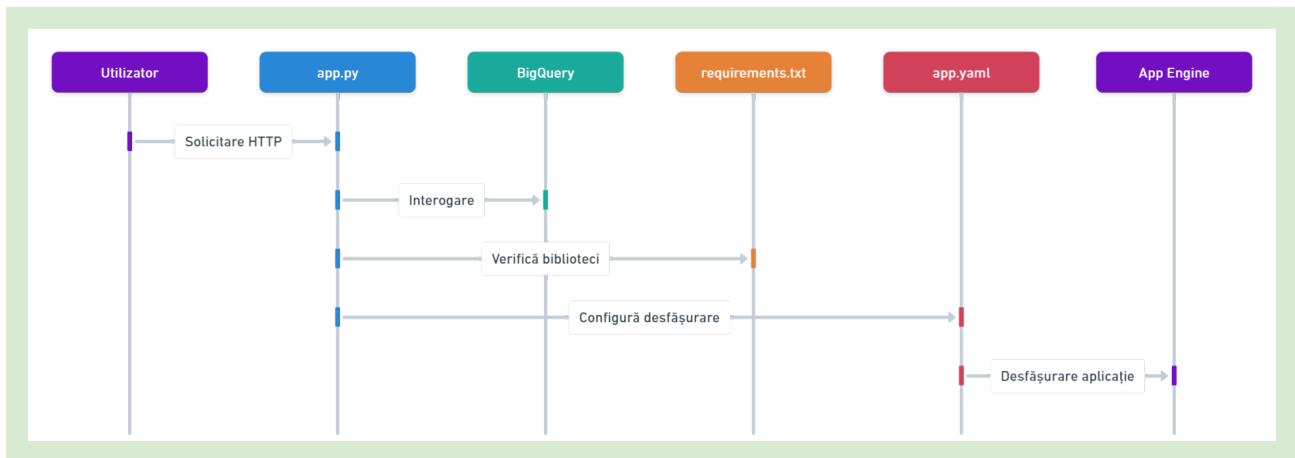


Fig: 1. Diagrama ilustrează interacțiunea utilizatorilor cu aplicația Flask și cum componentele acesteia interacționează între ele.

Activam App Engine Admin API, API-ul BigQuery și adăugăm roluri contului de serviciu App Engine: BigQuery Admin, BigQuery Data Viewer, BigQuery Job User.

Facem fork în github la proiectul <https://github.com/cpatrickalves/simple-flask-api>

În Cloud Shell clonăm branch-ul de pe github (acolo unde facem modificările) și facem deploy.

```
daci_draghia@cloudshell:~ (proiectcc-419616)$ git clone https://github.com/ddaci/simple-flask-api-projcc.git
Cloning into 'simple-flask-api-projcc'...
remote: Enumerating objects: 68, done.
remote: Counting objects: 100% (34/34), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 68 (delta 12), reused 27 (delta 8), pack-reused 34
Receiving objects: 100% (68/68), 23.16 KiB | 3.31 MiB/s, done.
Resolving deltas: 100% (24/24), done.
```

```
daci_draghia@cloudshell:~ (proiectcc-419616)$ ls
README-cloudshell.txt  simple-flask-api-projcc
daci_draghia@cloudshell:~ (proiectcc-419616)$ cd simple-flask-api-projcc
```

```
daci_draghia@cloudshell:~/simple-flask-api-projcc (proiectcc-419616)$ ls
app.py  db  LICENSE  Pipfile  Pipfile.lock  Procfile  README.md  requirements.txt  runtime.txt
```

```
daci_draghia@cloudshell:~/simple-flask-api-projcc (proiectcc-419616)$ git checkout projcc
Branch 'projcc' set up to track remote branch 'projcc' from 'origin'.
Switched to a new branch 'projcc'
```

```
daci_draghia@cloudshell:~/simple-flask-api-projcc (proiectcc-419616)$ gcloud app deploy
Services to deploy:
```

```
descriptor:    [/home/daci_draghia/simple-flask-api-projcc/app.yaml]
source:       [/home/daci_draghia/simple-flask-api-projcc]
target project: [proiectcc-419616]
target service: [default]
```

```
target version:      [20240409t084441]
target url:          [https://proiectcc-419616.ew.r.appspot.com]
target service account: [proiectcc-419616@appspot.gserviceaccount.com]
```

Do you want to continue (Y/n)? y

```
Beginning deployment of service [default]...
Created .gcloudignore file. See `gcloud topic gcloudignore` for details.
Uploading 9 files to Google Cloud Storage
11%
22%
33%
44%
56%
67%
78%
89%
100%
100%
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://proiectcc-419616.ew.r.appspot.com]
```

You can stream logs from the command line by running:
\$ gcloud app logs tail -s default

To view your application in the web browser run:
\$ gcloud app browse

```
daci_draghia@cloudshell:~/simple-flask-api-projcc (proiectcc-419616)$ gcloud app browse
Did not detect your browser. Go to this link to view your app:
https://proiectcc-419616.ew.r.appspot.com
```

App.yaml

```
runtime: custom
env: flex

env_variables:
  GOOGLE_CLOUD_PROJECT: "proiectccrefacut"

manual_scaling:
  instances: 1

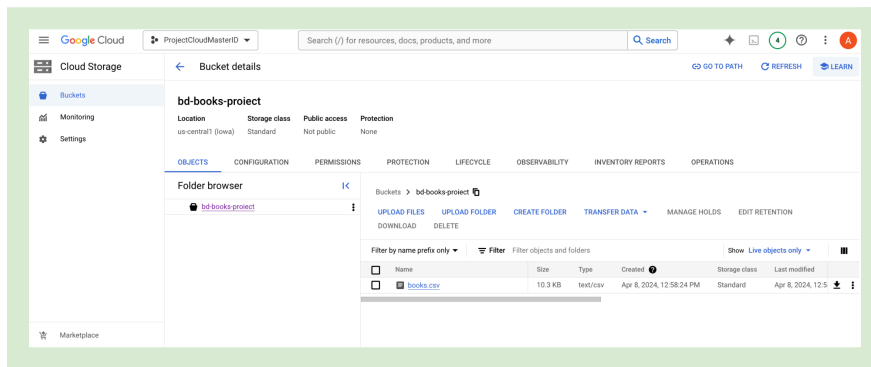
resources:
  cpu: 1
  memory_gb: 0.5
  disk_size_gb: 10
```

Requirements.txt

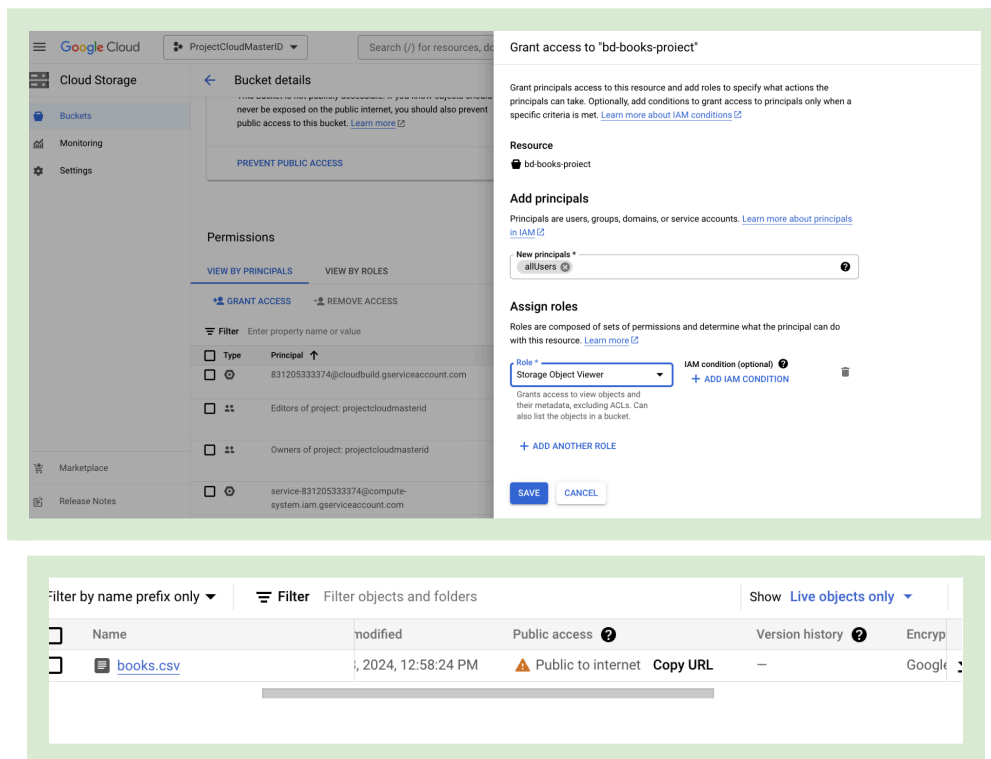
```
rClick==7.0
Flask==1.1.1
unicorn==20.0.4
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
google-cloud-bigquery==2.30.1
```

Google Cloud Storage

Incarcam books.db in Google Cloud Storage - cream un bucket care sa contina baza de date in format csv.



Permitem accesul public la bucket și creăm un URL accesibil public pentru baza de date.



Big Query

Am creat un set de date via Big Query. Apoi am creat un tabel in cadrul setului de date si am încărcat fișierul de date (books.csv) din bucket-ul de stocare în noul tabel creat.

File format
CSV

☐ Source Data Partitioning

Destination

Project *
projectcloudmasterid

Dataset *
books

Table *
books

Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors, dashes, and spaces are allowed.

Table type
Native table

Schema

☒ Auto detect

Schema will be automatically generated.

Partition and cluster settings

Partitioning
No partitioning

Clustering order
Clustering order determines the sort order of the data. Clustering can be used on both partitioned and non-partitioned tables.

Tags

Tags help you manage and enforce policies on your resources. Tags consist of a unique tag key and a set of tag values. [Learn more](#)

SELECT SCOPE

Advanced options

Facem dataset-ul public:

Grant access to "books"

Grant principals access to this resource and add roles to specify what actions the principals can take. Optionally, add conditions to grant access to principals only when a specific criteria is met. [Learn more about IAM conditions](#)

Resource

books

Add principals

Principals are users, groups, domains, or service accounts. [Learn more about principals in IAM](#)

New principals *

allUsers

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

Role *

BigQuery Data Viewer

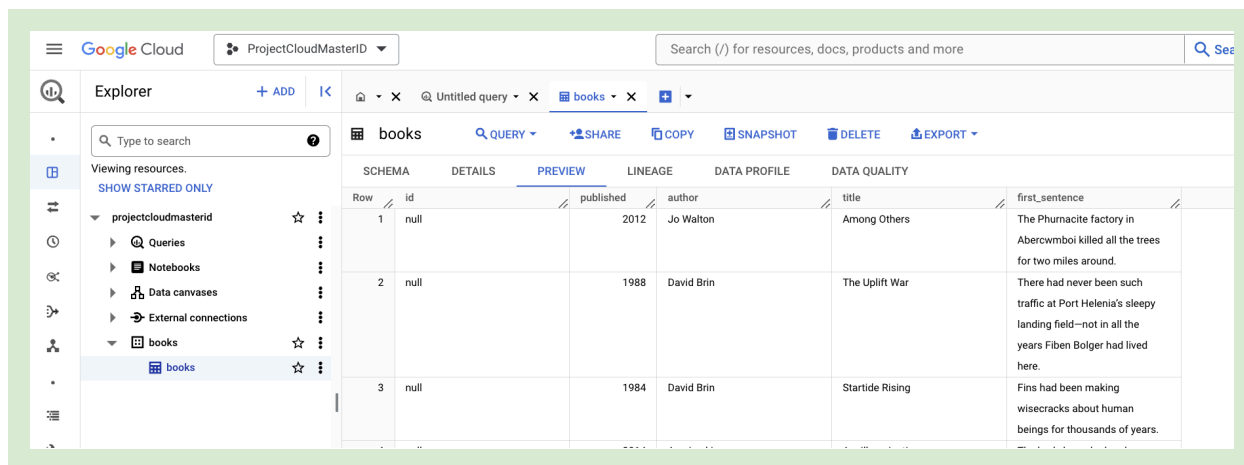
Access to view datasets and all of their contents

+ ADD ANOTHER ROLE

SAVE

CANCEL

5. Testarea unei operații de extragere a datelor din soluția de stocare aleasă utilizând un limbaj de programare.
Așa arată datele cu care lucrăm:



Row	id	published	author	title	first_sentence
1	null	2012	Jo Walton	Among Others	The Phurnacite factory in Abercwmboi killed all the trees for two miles around.
2	null	1988	David Brin	The Uplift War	There had never been such traffic at Port Helenia's sleepy landing field—not in all the years Fiben Bolger had lived here.
3	null	1984	David Brin	Startide Rising	Fins had been making wisecracks about human beings for thousands of years.

La punctul 6 prezentăm pe larg cum se extrag datele din soluția de stocare.

6. Descrierea codului sursă

Am modificat codul sursa conform celor de mai jos pentru a putea efectua următoarele funcționalități:

1. Afișarea tuturor cărților din BigQuery
<https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/bigquery-data>
2. Afișarea cărților grupate după un anumit autor
<https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/books/by-author?author=David%20Brin>
3. Afișarea cărților grupate după an
https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/books/by-year?published_year=2005
4. Adaugarea unei carti - aici este URL-ul către care se trimite o cerere POST (utilizând web.postman.co)
<https://projectcloudmasterid.ew.r.appspot.com/api/v2/resources/books>

#Se importă modulele necesare (Flask, request, jsonify, os și bigquery)

```
import os
from flask import Flask, request, jsonify
from google.cloud import bigquery
```

```
app = Flask(__name__)
```

Rută pentru pagina de start

The process of mapping URLs to functions is called routing.

```
@app.route('/', methods=['GET'])
```

```
def home():
```

```
    return "<h1>Distant Reading Archive</h1><p>This is a prototype API</p>"
```

Ruta 1 pentru obținerea tuturor cărților din BigQuery

```
@app.route('/api/v2/resources/bigquery-data', methods=['GET'])
```

```
def get_bigquery_data():
```

```
    client = bigquery.Client()
```

```
    query = "SELECT * FROM `proiectccrefacut.booksr.booksr`"
```

```
    query_job = client.query(query)
```

```
    results = query_job.result()
```

```
    rows = [dict(row) for row in results]
```

```
    return jsonify(rows)
```

Ruta 2 pentru obținerea cărților din BigQuery grupate după autor

```
@app.route('/api/v2/resources/books/by-author', methods=['GET'])
```

```
def get_books_by_author():
```

```
    author = request.args.get('author')
```

```
    if not author:
```

```
        return "<p>Parametrul 'author' trebuie specificat în URL. Exemplu: /api/v2/resources/books/by-author?author=David Brin</p>", 400
```

```
    client = bigquery.Client()
```

```
    query = f"SELECT * FROM `proiectccrefacut.booksr.booksr` WHERE author = '{author}'"
```

```
    query_job = client.query(query)
```

```
    results = query_job.result()
```

```
    rows = [dict(row) for row in results]
```

```
    if not rows:
```

```
        return jsonify({'message': f'Nu s-au găsit cărți scrise de autorul {author}.'}), 404
```

```
    return jsonify(rows)
```

Ruta 3 pentru obținerea cărților din BigQuery grupate după an

```
@app.route('/api/v2/resources/books/by-year', methods=['GET'])
```

```
def get_books_by_year():
```

```
    published_year = request.args.get('published_year')
```

```
    if not published_year:
```

```
        return "<p>Parametrul 'published_year' trebuie specificat în URL. Exemplu: /api/v2/resources/books/by-year?published_year=2005</p>", 400
```

```
    client = bigquery.Client()
```

```
    query = ""
```

```
        SELECT * FROM `proiectccrefacut.booksr.booksr`
```

```
        WHERE CAST(published AS STRING) LIKE @published_year
```

```
    ""
```

```
    job_config = bigquery.QueryJobConfig(
```

```
        query_parameters=[
```

```
            bigquery.ScalarQueryParameter("published_year", "STRING", published_year)
```

```
        ]
```

```
    )
```

```
    query_job = client.query(query, job_config=job_config)
```

```
    results = query_job.result()
```

```
    rows = [dict(row) for row in results]
```

```
    if not rows:
```

```
        return jsonify({'message': f'Nu s-au găsit cărți publicate în anul {published_year}.'}), 404
```

```
    return jsonify(rows)
```

Ruta 4 pentru adaugarea unei cărți

```
# Funcția add_book() permite adăugarea unei noi cărți în baza de date BigQuery. Aceasta primește datele cărții sub formă de JSON într-o cerere POST și le
adaugă în BigQuery. Dacă adăugarea reușește, returnează un mesaj de succes; altfel, returnează erorile întâmpinate
@app.route('/api/v2/resources/books', methods=['POST'])
def add_book():
    if not request.is_json:
        return "<p>The content isn't of type JSON</p>"

    content = request.get_json()
    title = content.get('title')
    author = content.get('author')
    published = content.get('published')
    first_sentence = content.get('first_sentence')

    if not all([title, author, published, first_sentence]):
        return jsonify({"error": "Missing required fields"}), 400

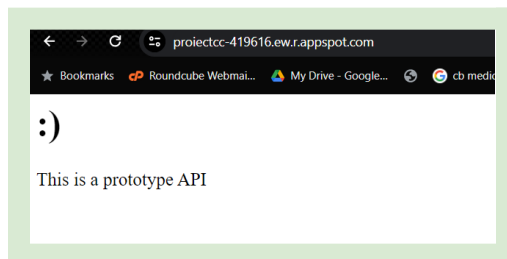
    client = bigquery.Client()
    table_id = "proiectccrefacut.booksr.booksr"
    rows_to_insert = [{"title": title, "author": author, "published": published, "first_sentence": first_sentence}]
    errors = client.insert_rows_json(table_id, rows_to_insert)

    if not errors:
        return jsonify({"message": "Book added successfully"}), 201
    else:
        return jsonify({"errors": errors}), 400

@app.errorhandler(404)
def page_not_found(e):
    return "<h1>404</h1><p>The resource could not be found</p>", 404

if __name__ == "__main__":
    port = int(os.environ.get('PORT', 8080))
    app.run(host='0.0.0.0', port=port)
```

Ruta de home: <https://proiectcc-419616.ew.r.appspot.com/>



Ruta care arată toate cărțile: <https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/bigquery-data>


```
projectcc-419616.ew.r.appspot.com/api/v2/resources/bigquery-data
pretty-print
{
  "author": "Jo Walton",
  "first_sentence": "The Phurnacite factory in Abercwmboi killed all the trees for two miles around.",
  "id": "null",
  "published": 2012,
  "title": "Among Others"
},
{
  "author": "David Brin",
  "first_sentence": "There had never been such traffic at Port Helenia's sleepy landing field-not in all the years Fiben Bolger had lived here.",
  "id": "null",
  "published": 1988,
  "title": "The Uplift War"
},
}
```

Ruta pentru afișarea cărților grupate după un anumit autor

<https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/books/by-author?author=David%20Brin>

```
projectcloudmasterid.ew.r.appspot.com/api/v2/resources/books/by-author?author=David%20Brin
pretty-print
{
  "author": "David Brin",
  "first_sentence": "There had never been such traffic at Port Helenia's sleepy landing field-not in all the years Fiben Bolger had lived here.",
  "id": "null",
  "published": 1988,
  "title": "The Uplift War"
},
{
  "author": "David Brin",
  "first_sentence": "Fins had been making wisecracks about human beings for thousands of years.",
  "id": "null",
  "published": 1984,
  "title": "Startide Rising"
},
{
  "author": "David Brin",
  "first_sentence": "There had never been such traffic at Port Helenia's sleepy landing field-not in all the years Fiben Bolger had lived here.",
  "id": null,
  "published": 1988,
  "title": "The Uplift War"
},
{
  "author": "David Brin",
  "first_sentence": "There had never been such traffic at Port Helenia's sleepy landing field-not in all the years Fiben Bolger had lived here.",
  "id": "null",
  "published": 1988,
  "title": "The Uplift War"
},
}
```

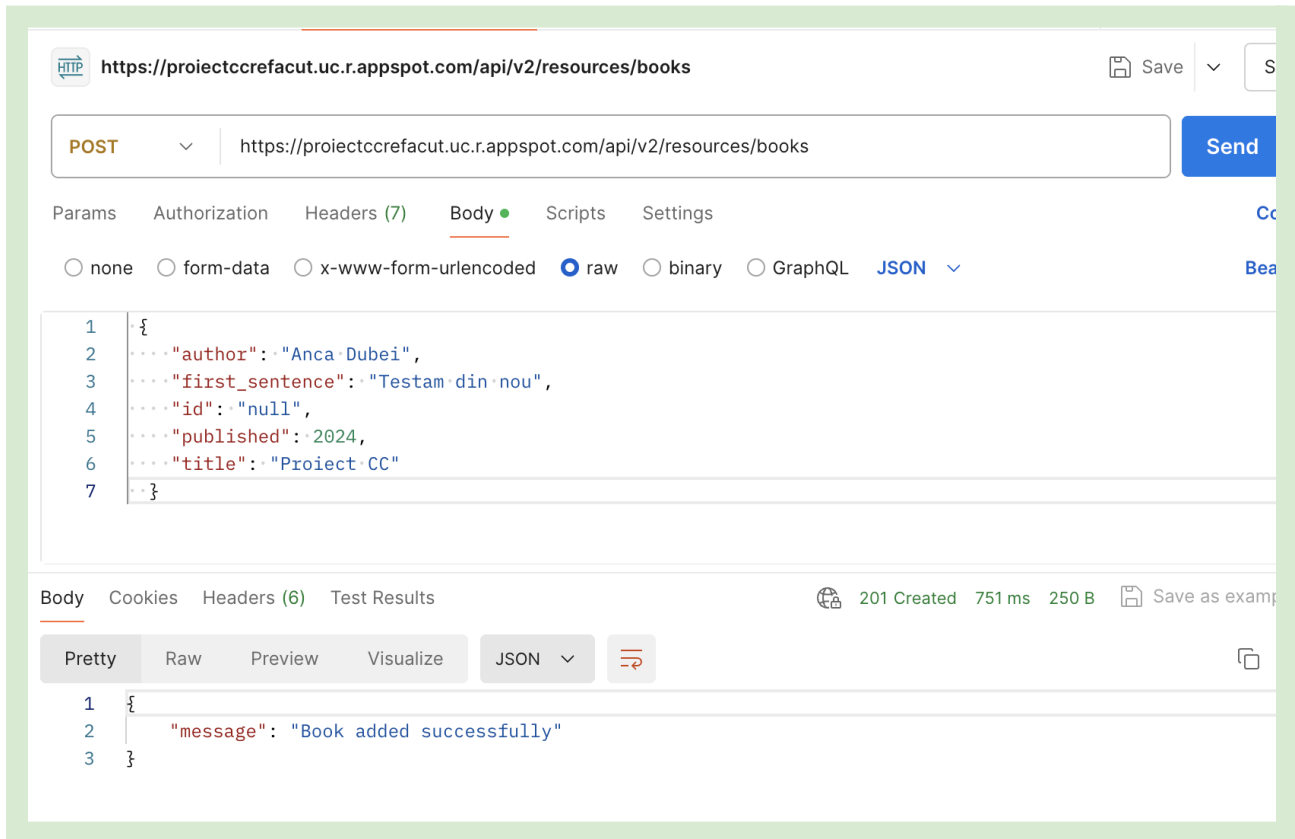
Ruta pentru afișarea cărților grupate după an

https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/books/by-year?published_year=2005

```
projectccrefacut.uc.r.appspot.com/api/v2/resources/books/by-author?author=Anca%20Dubei
pretty-print
{
  "author": "Anca Dubei",
  "first_sentence": "Testam din nou",
  "id": null,
  "published": 2024,
  "title": "Proiect CC"
}
```

Ruta pentru adăugarea unei cărți <https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/books>

Cu ajutorul web.postman.co, am trimis o cerere POST către ruta <https://proiectccrefacut.uc.r.appspot.com/api/v2/resources/books> și observăm în imaginea de mai jos că aceasta a fost procesată cu succes.



Putem observa că această carte a fost adăugată în `proiectccrefacut.booksr.booksr`



7. Alegerea soluțiilor pentru calcul, argumentând decizia

Pentru a implementa și a rula aplicația noastră REST API într-un mediu scalabil și performant, am ales să utilizăm Google Cloud Platform (GCP) datorită suitei sale de servicii flexibile și scalabile. Alegerea soluțiilor pentru calcul a fost influențată de nevoile noastre de a gestiona și procesa datele într-un mod eficient, de a asigura disponibilitatea și scalabilitatea aplicației și de a automatiza procesele de desfășurare și gestionare.

AppEngine

App Engine oferă un mediu gestionat care se ocupă de scalabilitatea automată, gestionarea resurselor și securitatea aplicației noastre, permițându-ne să ne concentrăm pe dezvoltarea aplicației fără a ne preocupa de administrarea infrastructurii. Prin configurarea mediului runtime Python 3.9 și specificând configurările necesare în fișierul app.yaml, am putut desfășura rapid și ușor aplicația noastră.

BigQuery

Pentru gestionarea și interogarea eficientă a datelor noastre structurate, am ales să utilizăm BigQuery, serviciul de analiză de date a GCP. BigQuery ne permite să interogăm și să analizăm seturi mari de date în timp real, oferind performanță și scalabilitate fără a fi nevoie să ne preocupăm de administrarea infrastructurii subiacente.

8. Modul de automatizare a trecerii în producție

Automatizarea trecerii în producție a codului sursă se referă la procesul de implementare automatizată a modificărilor aduse codului într-un mediu de producție, fără a fi necesară intervenția manuală repetitivă a dezvoltatorilor. Acest proces poate fi realizat folosind un sistem de integrare continuă și livrare continuă (CI/CD).

Setarea unui flux CI (Continuous Integration) /CD (Continuous Deployment) cu Cloud Build

Folosim un serviciu de CI/CD Cloud Build, pentru a automatiza procesul de construire, testare și desfășurare a aplicației. Configurăm acest flux pentru a monitoriza repository-ului de cod, astfel încât să declanșeze automat construirea și desfășurarea aplicației atunci când sunt detectate modificări.

Automatizarea trecerii în producție folosind Cloud Run, Docker și Cloud build (serverless)

Cloud Run este o platformă de calcul complet gestionată oferită de Google Cloud, care scalează automat containerele fără stare. Este proiectată pentru a permite dezvoltatorilor să ruleze aplicațiile lor containerizate într-un mediu serverless, unde Google Cloud se ocupă de infrastructură, scalare și gestionarea containerelor.

Facturarea se bazează pe resursele efectiv consumate de aplicație. Plătim doar pentru CPU, memorie și rețea utilizate în timpul gestionării cererilor.

Cloud Run este proiectat pentru **aplicații fără stare**, ceea ce înseamnă că fiecare cerere ar trebui să fie independentă și să nu se bazeze pe starea stocată în instanța containerului.

Implementarea pe Cloud Run este simplă și poate fi realizată cu o singură comandă, aplicațiile se pun în funcțiune rapid.

Utilizări comune pentru Cloud Run:

Microservicii: Rularea serviciilor individuale care gestionează sarcini specifice, permițând arhitecturi de aplicații modulare și scalabile.

APIs: Găzduirea API-urilor care pot scala automat în funcție de traficul de intrare.

Aplicații web: Implementarea aplicațiilor web care trebuie să scaleze în funcție de variațiile de trafic.

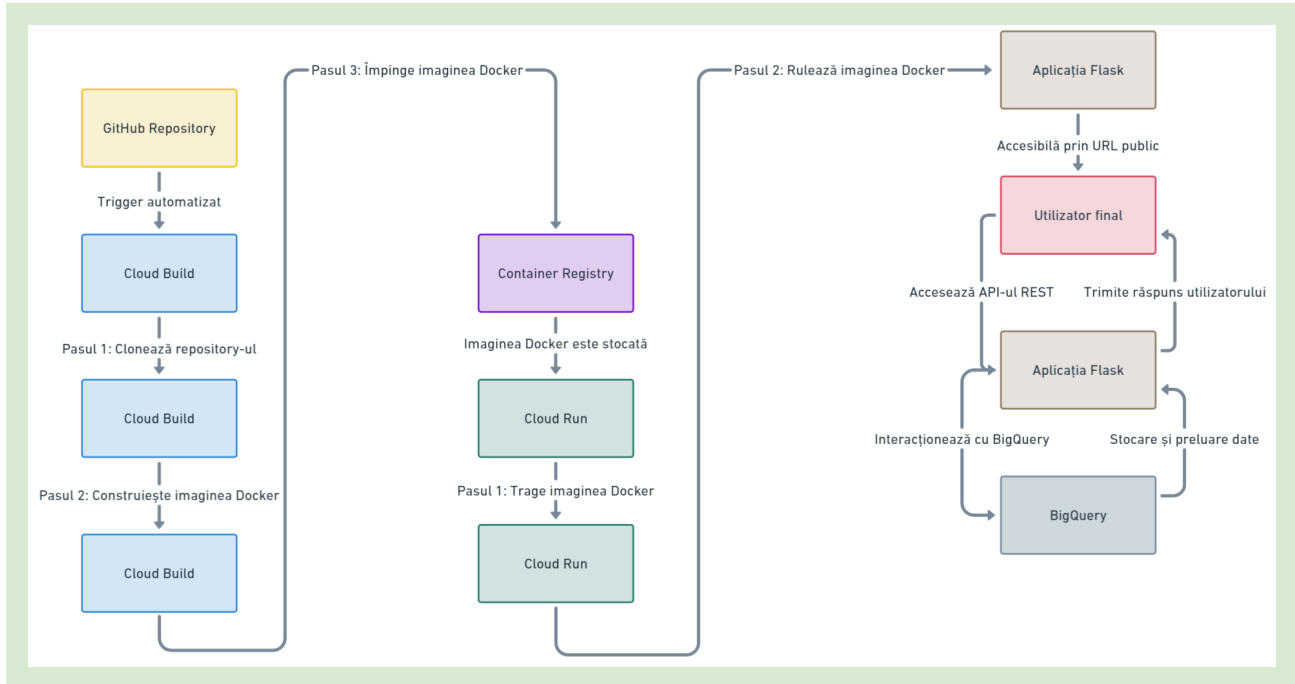


Fig: 2. Diagrama ilustrează automatizarea trecerii în producție folosind Cloud Run, Docker și Cloud build

Implementare

Am creat un nou proiect în Google Cloud Platform.

Activăm următoarele API-uri:

- Cloud Run API
- Cloud Build API
- Container Registry API
- BigQuery API

Baza de date este implementată în BigQuery, la fel cum am arătat la început.

Google Cloud flaskapidockercloudrunbigquery Search (/) for resources, docs, products, and more Search

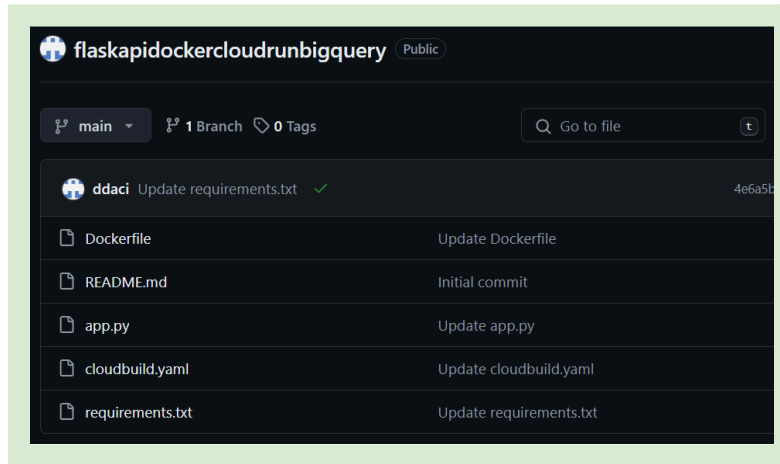
Explorer + ADD UNTITLED QUERY carti QUERY SHARE COPY SNAPSHOT DELETE EXPORT

Viewing resources. [SHOW STARRED ONLY](#)

- flaskapidockercloudrunbigquery
- Queries
- Notebooks
- Data canvases
- External connections
- carti

Row	id	published	author	title	first_sentence
1		2012	Jo Walton	Among Others	The Plumach factory in Abercwmboi killed all the trees for two miles around.
2		1988	David Brin	The Uplift War	There had never been such traffic at Port Helenia's sleepy landing field—not in all the years Fiben Bolger had lived here.
3		1994	David Brin	Startide Rising	Fins had been making

Codul sursă al aplicației se găsește pe Github și este structurat astfel:



Containerizăm aplicația(app.py) cu Docker - Dockerfile specifică modul în care aplicația este containerizată.

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Create a virtual environment
RUN python -m venv venv

# Activate the virtual environment and install dependencies
RUN /app/venv/bin/pip install --no-cache-dir -r requirements.txt

# Expose port 8080 to the outside world
EXPOSE 8080

# Set the environment variable to tell Flask to run the app
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_ENV=production

# Set the entrypoint to run gunicorn server using the virtual environment
CMD ["/app/venv/bin/gunicorn", "--bind", "0.0.0.0:8080", "app:app"]
```

Creăm un fișier de configurare Cloud Build (cloudbuild.yaml) pentru a automatiza procesul de construire și trimitere a imaginii Docker în Google Container Registry (GCR).

Deployment - se utilizează imaginea Docker din GCR pentru a desfășura aplicația pe Cloud Run.

cloudbuild.yaml

```
options:
  logging: CLOUD_LOGGING_ONLY

steps:
- name: 'gcr.io/cloud-builders/gcloud'
```

```

entrypoint: 'bash'
args:
- '-c'
- |
  # Descărcăm cheia JSON din GCS în directorul de lucru al build-ului
  gsutil cp gs://cheie/flaskapidockercloudrunbigquery-88d7334afd05.json /workspace/flaskapidockercloudrunbigquery-88d7334afd05.json

- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/flaskapidockercloudrunbigquery/flask-app', '.']
  env:
- 'GOOGLE_APPLICATION_CREDENTIALS=/workspace/flaskapidockercloudrunbigquery-88d7334afd05.json'

- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/flaskapidockercloudrunbigquery/flask-app']
  env:
- 'GOOGLE_APPLICATION_CREDENTIALS=/workspace/flaskapidockercloudrunbigquery-88d7334afd05.json'

- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: 'gcloud'
  args: ['run', 'deploy', 'flask-app', '--image', 'gcr.io/flaskapidockercloudrunbigquery/flask-app', '--platform', 'managed', '--region', 'asia-east1', '--allow-unauthenticated']
  env:
- 'GOOGLE_APPLICATION_CREDENTIALS=/workspace/flaskapidockercloudrunbigquery-88d7334afd05.json'

images:
- 'gcr.io/flaskapidockercloudrunbigquery/flask-app'

artifacts:
  objects:
    location: 'gs://cheie'
    paths: ['flaskapidockercloudrunbigquery-88d7334afd05.json']

```

requirements.txt

```

Click==8.0.4
itsdangerous==2.0.1
Jinja2==3.0.3
MarkupSafe==2.0.1
google-cloud-bigquery==2.30.1
Flask==2.0.1
gunicorn==20.1.0
Werkzeug==2.0.1

```

app.py

```

from flask import Flask, request, jsonify
from google.cloud import bigquery

# Init app
app = Flask(__name__)

@app.route('/', methods=['GET'])
def home():
    return """
    <h1>Use the following routes (add them to the url adress above)</h1>
    <p>all books /api/v2/resources/bigquery-data</p>
    <p>/api/v2/resources/books/by-author?author=David Brin</p>
    <p>/api/v2/resources/books/by-year?published_year=2005</p>
    <p>/api/v2/resources/books</p>
    """

# Endpoint for fetching data from BigQuery
@app.route('/api/v2/resources/bigquery-data', methods=['GET'])
def get_bigquery_data():
    client = bigquery.Client()
    query = """
    SELECT * FROM `flaskapidockercloudrunbigquery.carti.carti`
    """

```

```

LIMIT 10
"""

query_job = client.query(query)
results = query_job.result()
rows = [dict(row) for row in results]
return jsonify(rows)

# Endpoint for fetching books by author
@app.route('/api/v2/resources/books/by-author', methods=['GET'])
def get_books_by_author():
    query_parameters = request.args
    author = query_parameters.get('author')
    if not author:
        return "<p>Parametrul 'author' trebuie specificat în URL. Exemplu: /api/v2/resources/books/by-author?author=David Brin</p>", 400

    client = bigquery.Client()
    query = f"""
        SELECT * FROM `flaskapidockercloudrunbigquery.carti.carti`
        WHERE author = '{author}'
    """

    query_job = client.query(query)
    results = query_job.result()
    rows = [dict(row) for row in results]
    if not rows:
        return jsonify({'message': f'Nu s-au găsit cărți scrise de autorul {author}.'}), 404
    return jsonify(rows)

# Endpoint for fetching books by year
@app.route('/api/v2/resources/books/by-year', methods=['GET'])
def get_books_by_year():
    query_parameters = request.args
    published_year = query_parameters.get('published_year')
    if not published_year:
        return "<p>Parametrul 'published_year' trebuie specificat în URL. Exemplu: /api/v2/resources/books/by-year?published_year=2005</p>", 400

    client = bigquery.Client()
    query = """
        SELECT * FROM `flaskapidockercloudrunbigquery.carti.carti`
        WHERE CAST(published AS STRING) LIKE @published_year
    """

    job_config = bigquery.QueryJobConfig(
        query_parameters=[
            bigquery.ScalarQueryParameter("published_year", "STRING", published_year)
        ]
    )
    query_job = client.query(query, job_config=job_config)
    results = query_job.result()
    rows = [dict(row) for row in results]
    if not rows:
        return jsonify({'message': f'Nu s-au găsit cărți publicate în anul {published_year}.'}), 404
    return jsonify(rows)

# Error handler for 404
@app.errorhandler(404)
def page_not_found(e):
    return "<h1>404</h1><p>The resource could not be found</p>", 404

# Endpoint for adding a book
@app.route('/api/v2/resources/books', methods=['POST'])
def add_book():
    if not request.is_json:
        return "<p>The content isn't of type JSON</p>", 400

    content = request.get_json()
    title = content.get('title')
    author = content.get('author')
    published = content.get('published')
    first_sentence = content.get('first_sentence')

```

```

if not all([title, author, published, first_sentence]):
    return jsonify({"error": "Missing required fields"}), 400

client = bigquery.Client()
table_id = "flaskapidockercloudrunbigquery.carti.carti"
rows_to_insert = [
    {
        "title": title,
        "author": author,
        "published": published,
        "first_sentence": first_sentence
    }
]

errors = client.insert_rows_json(table_id, rows_to_insert)
if not errors:
    return jsonify({"message": "Book added successfully"}), 201
else:
    return jsonify({"errors": errors}), 400

# A method that runs the application server.
if __name__ == "__main__":
    app.run(debug=True, threaded=True, port=8080)

```

Configurăm un trigger în Cloud Build pentru a lansa build-uri automat la fiecare commit efectuat în proiectul de pe Github.

[←](#)
[Edit trigger](#)
[■ DISABLE](#)
[🗑 DELETE](#)

Source: [ddaci/flaskapidockercloudrunbigquery](#) [View triggered builds](#)

Name *

Must be unique within the project's region

Region *

Description

Tags

Event

Repository event that invokes trigger

☒ Push to a branch
☐ Push new tag
☐ Pull request
Not available for Cloud Source Repositories

Or in response to

☐ Manual invocation
☐ Pub/Sub message
☐ Webhook event

Source

Repository generation

☒ 1st gen
☐ 2nd gen

Repository *

Select the repository to watch for events and clone when the trigger is invoked

Location

☒ Repository
ddaci/flaskapidockercloudrunbigquery (GitHub App)

☐ Inline
Write inline YAML

Cloud Build configuration file location *

Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

Advanced

Substitution variables

Substitutions allow re-use of a cloudbuild.yaml file with different variable values. Use bash string manipulation to combine variables and bindings to access arbitrary data in the JSON payload of the webhook. [Learn more](#)

Approval

☐ Require approval before build executes

Build logs

☒ Send build logs to GitHub

Build logs will be visible to any GitHub user with read access to this repository.

Service account

If no custom service account is provided, the default Cloud Build Service Account will be used. We strongly recommend using a custom service account with limited permissions needed for this build's execution
[LEARN MORE](#)

Vom utiliza un cont de serviciu cu permisiuni limitate pentru build-uri, îmbunătățind astfel securitatea și controlul asupra resurselor din Google Cloud Platform:

- Se creează un cont de serviciu personalizat cu permisiuni limitate în Google Cloud Console - îl numim contserv
- Se generează și se descarcă cheia JSON pentru acest cont de serviciu personalizat.
- Fișierul cloudbuild.yaml se actualizează pentru a specifica contul de serviciu personalizat.
- Cheia JSON se încarcă într-un bucket Google Cloud Storage și se configurează pasul pentru a descărca cheia în timpul build-ului.
- Trigger-ul Cloud Build se actualizează pentru a folosi noua configurație.

Service accounts for project "flaskapidockercloudrunbigquery"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. [Learn more about service accounts](#)

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. [Learn more about service account organization policies](#)

Filter	Enter property name or value								
	Email	Status	Name	Description	Key ID	Key creation date	OAuth 2 Client ID		Actions
<input type="checkbox"/>	contserv@flaskapidockercloudrunbigquery.iam.gserviceaccount.com	Enabled	contserv		88d7334af05d65ea4dd7d4ad7559029d3c876fe	May 19, 2024	117106048001966935258		
<input type="checkbox"/>	679938535850-compute@developer.gserviceaccount.com	Enabled	Default compute service account		No keys		110371814684820405953		

Contul de serviciu Cloud Build (contserv@flaskapidockercloudrunbigquery.iam.gserviceaccount.com) trebuie să poată acționa în numele contului de serviciu Compute Engine (679938535850-compute@developer.gserviceaccount.com) pentru a executa build-uri și implementări. Pentru ca acest lucru să fie posibil, contul de serviciu Cloud Build trebuie să aibă permisiunea Service Account User asupra contului de serviciu Compute Engine.

Edit access to "Default compute service account"

Principal ?	Resource
contserv@flaskapidockercloudrunbigquery.iam.gserviceaccount.com	Default compute service account

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

Role
Service Account User

IAM condition (optional) ?
[+ ADD IAM CONDITION](#)

Run operations as the service account.

[+ ADD ANOTHER ROLE](#)

SAVE

CANCEL

Roluri și Permiuni pentru contul de serviciu 679938535850-compute@developer.gserviceaccount.com

1. Artifact Registry Writer: Pentru a putea împinge imagini Docker în Artifact Registry, astfel încât acestea să fie disponibile pentru implementare pe Cloud Run sau alte servicii.
2. Cloud Build Editor: Pentru a gestiona procesele de build și a automatiza construirea și implementarea aplicațiilor.
3. Editor: Pentru a avea permisiuni generale de gestionare a resurselor proiectului, necesare pentru diverse operațiuni.
4. Service Account User: Pentru a permite contului de serviciu să acționeze în numele altor conturi de serviciu, esențial pentru interacțiunile dintre Cloud Build și Compute Engine.
5. Storage Admin: Pentru a gestiona resursele de stocare necesare în timpul proceselor de build și implementare.

Roluri și Permiuni pentru contul de serviciu conserv@flaskapidockercloudrunbigquery.iam.gserviceaccount.com

1. Artifact Registry Writer: Pentru a putea împinge imagini Docker în Artifact Registry, astfel încât acestea să fie disponibile pentru implementare pe Cloud Run sau alte servicii.
2. Cloud Build Editor: Pentru a gestiona procesele de build și a automatiza construirea și implementarea aplicațiilor.
3. Cloud Build Service Account: Pentru a asigura că contul de serviciu are permisiunile necesare pentru a efectua build-uri în Cloud Build.
4. Cloud Run Admin: Pentru a permite gestionarea și implementarea aplicațiilor pe Cloud Run.
5. Service Account User: Pentru a permite contului de serviciu să acționeze în numele altor conturi de serviciu, esențial pentru interacțiunile dintre diverse servicii și resurse.

6. Storage Admin: Pentru a gestiona resursele de stocare necesare în timpul proceselor de build și implementare.
7. Viewer: Pentru a permite vizualizarea resurselor din proiect fără a putea modifica aceste resurse, util pentru monitorizare și diagnosticare.

Permissions for project "flaskapidockercloudrunbigquery"

These permissions affect this project and all of its resources. [Learn more](#)

[VIEW BY PRINCIPALS](#) [VIEW BY ROLES](#)

[+ GRANT ACCESS](#) [- REMOVE ACCESS](#)

Filter Enter property name or value

<input type="checkbox"/> Type	Principal	Name	Role	Security insights	
<input type="checkbox"/>	679938535850-compute@developer.gserviceaccount.com	Default compute service account	Artifact Registry Writer Cloud Build Editor Editor Service Account User Storage Admin		
<input type="checkbox"/>	contserv@flaskapidockercloudrunbigquery.iam.gserviceaccount.com	contserv	Artifact Registry Writer Cloud Build Editor Cloud Build Service Account Cloud Run Admin Service Account User Storage Admin Viewer		
<input type="checkbox"/>	daci.draghia@gmail.com	daciana draghia	Owner		

Cloud build funcționează fără erori.

flaskapidockercloudrunbigquery

Build history [STOP STREAMING BUILDS](#)

Region: asia-east1

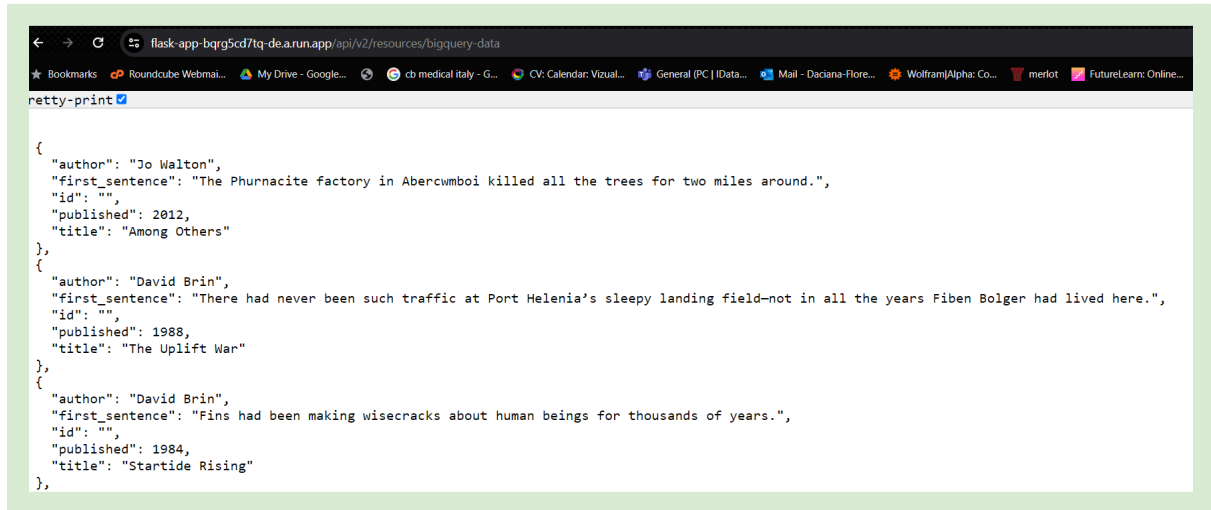
Filter Enter property name or value

<input type="checkbox"/> Status	Build	Source	Ref	Commit	Trigger Name	Created	Duration	Security Insights	
<input type="checkbox"/>	17d7a7bb	ddaci/flaskapidockercloudrunbigquery	main	4e6a5b4	flask-app-deploy	5/21/24, 12:19 AM	2 min 12 sec	VIEW	
<input type="checkbox"/>	08af3fee	ddaci/flaskapidockercloudrunbigquery	main	f125e6f	flask-app-deploy	5/21/24, 12:16 AM	27 sec	-	
<input type="checkbox"/>	0e7b8fc2	ddaci/flaskapidockercloudrunbigquery	main	5534707	flask-app-deploy	5/21/24, 12:13 AM	29 sec	-	
<input type="checkbox"/>	83198f9c	ddaci/flaskapidockercloudrunbigquery	main	3f5502b	flask-app-deploy	5/21/24, 12:11 AM	24 sec	-	
<input type="checkbox"/>	7efcbf2c	ddaci/flaskapidockercloudrunbigquery	main	76323e8	flask-app-deploy	5/21/24, 12:08 AM	25 sec	-	
<input type="checkbox"/>	4001d488	ddaci/flaskapidockercloudrunbigquery	main	a3ad6da	flask-app-deploy	5/21/24, 12:07 AM	25 sec	-	
<input type="checkbox"/>	e0f7f791	ddaci/flaskapidockercloudrunbigquery	main	366f351	flask-app-deploy	5/20/24, 11:51 PM	5 min 58 sec	-	
<input type="checkbox"/>	4a2cd4bb	ddaci/flaskapidockercloudrunbigquery	main	9d8db41	flask-app-deploy	5/20/24, 11:42 PM	5 min 54 sec	-	
<input type="checkbox"/>	10f1d4f0	ddaci/flaskapidockercloudrunbigquery	main	26d3534	flask-app-deploy	5/19/24, 10:16 PM	2 min 1 sec	-	
<input type="checkbox"/>	d24fe138	ddaci/flaskapidockercloudrunbigquery	main	cc76625	flask-app-deploy	5/19/24, 10:14 PM	-	-	
<input type="checkbox"/>	6dc483cc	ddaci/flaskapidockercloudrunbigquery	main	977146e	flask-app-deploy	5/19/24, 9:40 PM	-	-	
<input type="checkbox"/>	a0d69539	ddaci/flaskapidockercloudrunbigquery	main	381d244	flask-app-deploy	5/19/24, 9:31 PM	-	-	
<input type="checkbox"/>	611902bf	ddaci/flaskapidockercloudrunbigquery	main	2915811	flask-app-deploy	5/19/24, 9:11 PM	-	-	
<input type="checkbox"/>	9703ee9f	ddaci/flaskapidockercloudrunbigquery	main	590292c	flask-app-deploy	5/19/24, 8:51 PM	43 sec	-	
<input type="checkbox"/>	a6b57b99	ddaci/flaskapidockercloudrunbigquery	main	d3e0673	flask-app-deploy	5/19/24, 7:19 PM	43 sec	-	
<input type="checkbox"/>	84bdc100	ddaci/flaskapidockercloudrunbigquery	main	a6f7c35	flask-app-deploy	5/19/24, 6:40 PM	49 sec	-	
<input type="checkbox"/>	88637c69	ddaci/flaskapidockercloudrunbigquery	main	f4addee	flask-app-deploy	5/19/24, 6:11 PM	43 sec	-	

Aceasta este aplicația construită:

<https://flask-app-bqrg5cd7tq-de.a.run.app/>

<https://flask-app-bqrg5cd7tq-de.a.run.app/api/v2/resources/bigquery-data>



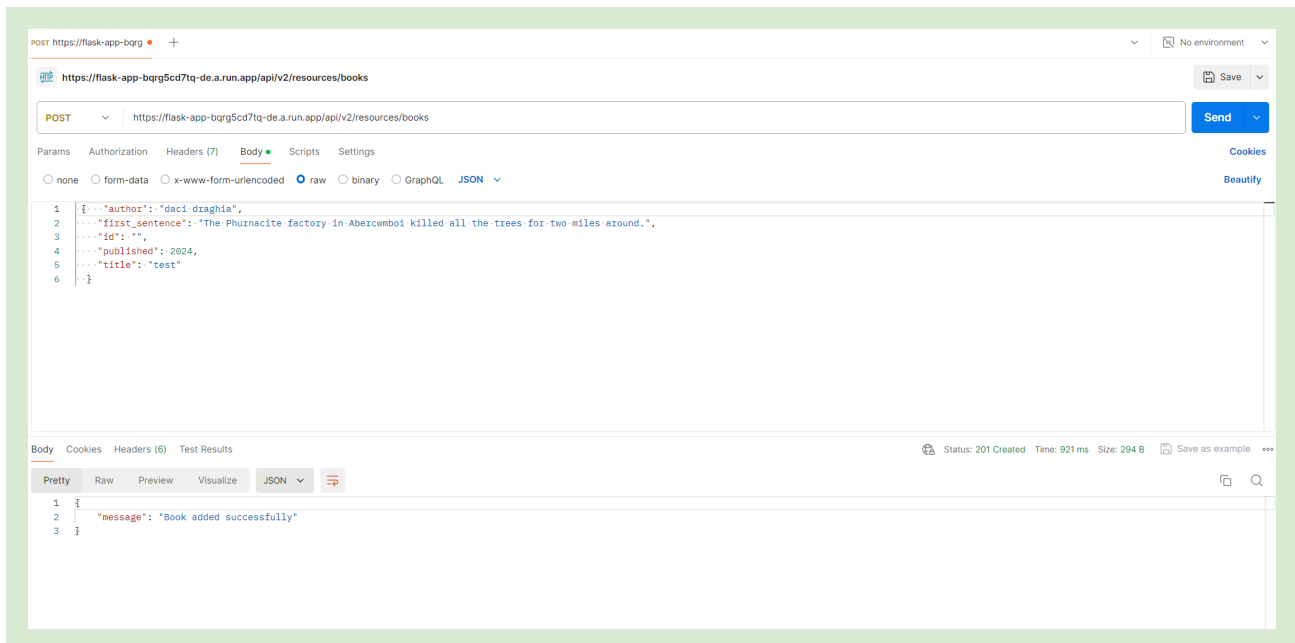
```
netty-print

{
  "author": "Jo Walton",
  "first_sentence": "The Phurnacite factory in Abercwmboi killed all the trees for two miles around.",
  "id": "",
  "published": 2012,
  "title": "Among Others"
},
{
  "author": "David Brin",
  "first_sentence": "There had never been such traffic at Port Helenia's sleepy landing field-not in all the years Fiben Bolger had lived here.",
  "id": "",
  "published": 1988,
  "title": "The Uplift War"
},
{
  "author": "David Brin",
  "first_sentence": "Fins had been making wisecracks about human beings for thousands of years.",
  "id": "",
  "published": 1984,
  "title": "Startide Rising"
},
}
```

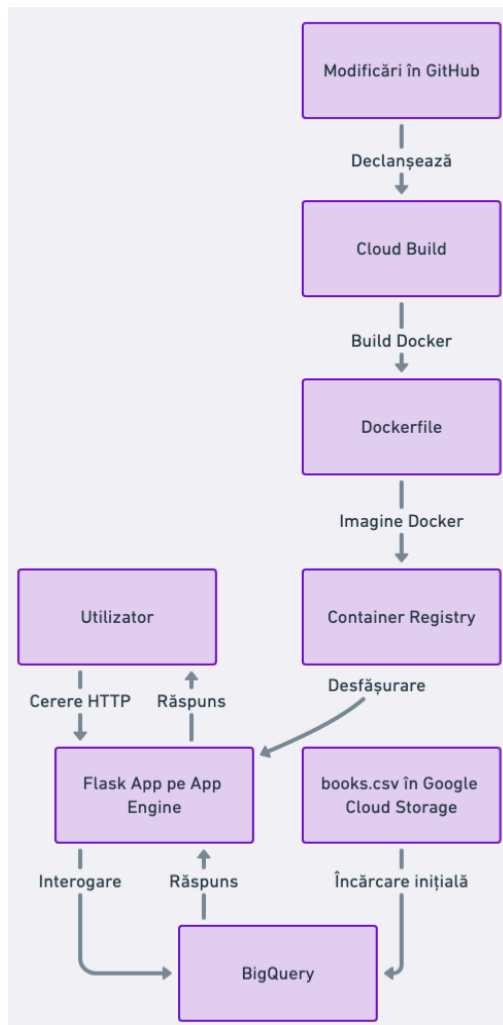
<https://flask-app-bqrg5cd7tq-de.a.run.app/api/v2/resources/books/by-author?author=David%20Brin>

https://flask-app-bqrg5cd7tq-de.a.run.app/api/v2/resources/books/by-year?published_year=2005

Funcționează și ruta de adăugare.



9. Diagrama



- ❖ Utilizatorul trimite o cerere HTTP către aplicația Flask
- ❖ Aplicația Flask rulează pe App Engine
- ❖ Flask preia cererea și decide ce operațiuni trebuie efectuate
- ❖ Dacă sunt necesare date din BigQuery, Flask face o interogare către BigQuery
- ❖ Datele sunt returnate de BigQuery către Flask
- ❖ Flask trimite răspunsul utilizatorului
- ❖ Pentru încărcarea inițială a datelor, fișierul books.csv este încărcat în Google Cloud Storage
- ❖ Datele din books.csv sunt încărcate din Google Cloud Storage în BigQuery
- ❖ Procesul de CI/CD
 - Modificările sunt împinse în repository-ul GitHub
 - Cloud Build detectează modificările și rulează build-ul
 - Cloud Build folosește Dockerfile pentru a construi o imagine Docker
 - Imaginea Docker este încărcată în Container Registry
 - Cloud Build desfășoară imaginea Docker pe App Engine

Fig: 3. Diagrama ilustrează interacțiunea utilizatorilor cu aplicația și procesul de CI/CD

10. Evaluarea maturității din perspectiva arhitecturii native cloud

Pentru a evalua maturitatea aplicației din perspectiva arhitecturii native cloud, vom utiliza cele trei axe principale : axa serviciilor, axa proiectării aplicațiilor și axa automatizării. Aceste axe ajută la înțelegerea gradului de maturitate al aplicației în ceea ce privește adaptabilitatea, scalabilitatea și automatizarea în contextul unui mediu cloud.

Axa serviciilor

Axa serviciilor se referă la modul în care aplicația utilizează serviciile oferite de platformele cloud și cât de bine este integrată cu acestea.

Aplicația noastră folosește :

- **Google App Engine** : Aplicația folosește Google App Engine pentru a rula și gestiona serviciul web. App Engine oferă scalabilitate automată, gestionare a resurselor și securitate.
- **Google Cloud Storage (GCS)**: Folosirea GCS pentru stocarea fișierului booksr.csv din baza de date SQLite inițială demonstrează utilizarea unui serviciu de stocare scalabil și durabil.
- **BigQuery** : Integrarea cu BigQuery pentru gestionarea și interogarea datelor demonstrează capacitatea aplicației de a lucra cu volume mari de date și de a efectua analize complexe într-un mod eficient și scalabil.

Concluzie : Aplicația utilizează eficient serviciile oferite de Google Cloud, ceea ce îi conferă un grad ridicat de maturitate pe axa serviciilor.

Axa proiectării aplicației

Axa proiectării aplicațiilor se referă la modul în care aplicația este structurată și construită pentru a fi eficientă, scalabilă și ușor de întreținut.

- **Separarea microserviciilor** : Arhitectura aplicației este modularizată, cu funcționalități distincte gestionate de endpoint-uri separate. De exemplu, există endpoint-uri distincte pentru afișarea cărților din BigQuery, pentru cărțile grupate după autor sau an și pentru adăugarea de cărți noi. Această separare permite scalabilitate și dezvoltare independentă a modulelor.
- **Utilizarea unor servicii bazate pe REST API**: Aplicația este construită pe un model REST API, ceea ce permite comunicarea între componente și interacțiunea cu alte servicii sau clienți externi. Acest design permite o integrare ușoară și flexibilitate în gestionarea și extinderea funcționalităților.

Concluzie: Aplicația are o arhitectură bine proiectată și modulară, ceea ce îi conferă un grad ridicat de maturitate pe axa proiectării aplicațiilor.

Axa automatizării

Axa automatizării se referă la capacitatea aplicației de a automatiza procesele de build, testare și desfășurare pentru a îmbunătăți eficiența și consistența livrării.

Automatizarea procesului de desfășurare :

- **CI/CD cu Cloud Build** : Utilizarea Cloud Build pentru a automatiza procesul de construire și desfășurare a aplicației este un indicator cheie al maturității. Configurația cloudbuild.yaml definește pașii necesari pentru a construi imaginea Docker, a o încărca în Container Registry și a o desfășura pe App Engine. Fluxul CI/CD monitorizează automat repository-ul de cod și declanșează construirea și desfășurarea automată a aplicației atunci când sunt detectate modificări. Acest lucru reduce eroarea umană și timpul necesar pentru desfășurarea actualizărilor.
- **Automatizarea scalării și gestionării resurselor** : Aplicația beneficiază de scalabilitatea automată oferită de App Engine, care gestionează automat resursele în funcție de volumul de trafic. Astfel, nu este necesară intervenția manuală pentru a gestiona infrastructura sau pentru a face față creșterii bruște a cererilor.
- **Webhook-uri GitHub**: Configurarea webhook-urilor pentru a declanșa build-uri automate la fiecare push în repository-ul GitHub asigură că orice modificare a codului este integrată și desfășurată fără intervenție manuală.

Concluzie : Aplicația utilizează un sistem robust de CI/CD pentru a automatiza procesele de build și deploy, conferindu-i un grad ridicat de maturitate pe axa automatizării.

În general, arhitectura aplicației este bine adaptată pentru mediul cloud, beneficiind de servicii gestionate, o proiectare modulară și automatizarea proceselor cheie. Utilizarea GCP oferă un mediu scalabil, performant și ușor de administrat pentru dezvoltarea și desfășurarea aplicației.

11. Considerente din punct de vedere SLA

Service Level Agreement (SLA) reprezintă un angajament formal (contract) între un furnizor de servicii și un client privind așteptările clientului ca și consecințele dacă acestea nu sunt îndeplinite.

În vederea realizării SLA-ului, se au în vedere (lista nu este exhaustivă) :

- descrierea serviciului
- indicatorii de serviciu (SLI)
- obiective (SLO)
- responsabilități
- proceduri de monitorizare și de raportare
- proceduri de gestionare incidente
- penalități
- compensații

Indicatori pe care îi putem lua în considerare pentru aplicația noastră pentru SLA

- **Timpul de răspuns al aplicației (API response time)** - timpul necesar pentru a răspunde la cereri HTTP (GET, POST, etc)
- **Rata de eroare** : procentajul cererilor care se termină cu un cod de eroare HTTP (ex: eroare 404 pentru resurse inexistente)
- **Disponibilitate** : procentajul de timp în care serviciul este disponibil și răspunde la cereri
- **Rata de succes a încărcării în BD** : procentajul cererilor POST către endpoint-ul `/api/v2/resources/books` salvate cu succes în baza de date

Pentru a măsura aceste metrice, sunt folosite instrumente de monitorizare și observabilitate cum ar fi **Prometheus**, **Grafana**, ELK Stack, sau soluții comerciale precum New Relic sau Datadog.

Obiective (SLO)

Ca exemplu, niște obiective realiste pentru indicatorii de mai sus ar fi :

- Timpul de răspuns : 95% din cererile către API vor primi un răspuns în mai puțin de 200ms
- Rata de eroare: Mai puțin de 1% din cererile către API vor rezulta într-un cod de eroare HTTP
- Disponibilitate : Serviciul va fi disponibil 99.9% din timp
- Rata de succes a încărcării în BD : 99% din cererile POST către endpoint-ul `/api/v2/resources/books` vor fi salvate cu succes în baza de date