## Authors

- Johann Guepjop

## Eng Manager(s)/Tech Lead(s)/PM(s)

- Clifford Fajardo
- Sachin Karisiddappa

## Reviewers

| Name | Review Status |
|---|---|
| Jinqing Xu | |
| Kevin Reber | |
| Clifford Fajardo | |
| Sachin Karisiddappa | |

# Mini RFC - Flask/Remix Benchmarking

The purpose of this document is to record the journey taken to conduct benchmarking on a flask and remix app in order to compare their performance.

## Basic Setup

- Create a simple flask (version 1) and remix app with different endpoints for different payload sizes. The current sizes range from small, medium, to large, with respect to payloads similar to those use within LinkedIn.
- Each payload endpoint can simply return the payload size directly.

# Add Database

In order to more accurately simulate an internal flask/remix app we will be adding a database to retrieve the different payloads from. Postgresql used for this project

## Creating a postgres database

- Install postgres using homebrew: `brew install postgresql`
- Start postgres: `brew services start postgresql`
- Get into your postgres db using: psql postgres

## Creating a new db and user

- Now that we are in the postgres db, create a new db:
  CREATE DATABASE benchmark_db;
- Create a new user and give them access to the db:
  CREATE USER usr WITH PASSWORD 'password';
  GRANT ALL PRIVILEGES ON DATABASE benchmark_db TO usr;
  \l; (to confirm that the db was created)
  \q; (to quit postgres session)

## Setup database

- Install needed packages:
  pip install psycopg2-binary
- Create a db initialization script called **init.py** (this can be done in either flask or remix app). Although this could be done in a terminal we will be using some script to ensure that we can easily recreate/modify the table if needed.
- This table (**payload**) will store the small, medium, and large payloads that we currently have:

```
import json
from lib2to3.pgen2.pgen import DFAState
import os
import psycopg2
from largePayload import getLargePayload
from mediumPayload import getMediumPayload

from smallPayload import getSmallPayload



conn = psycopg2.connect(
       host="localhost",
```

```
        database="benchmark_db",
        user=os.environ['DB_USERNAME'],
        password=os.environ['DB_PASSWORD'])

# Open a cursor to perform database operations
cur = conn.cursor()

# Execute a command: this creates a new table
cur.execute('DROP TABLE IF EXISTS payload;')
cur.execute('CREATE TABLE payload (id serial PRIMARY KEY,'
                                   'title varchar (150) NOT NULL,'
                                   'data json NOT NULL);'
                                   )

# Insert data into the table
cur.execute("""
            INSERT INTO payload (title, data)
            VALUES (%s, %s);
            """,
            ('small', json.dumps(getSmallPayload())))

# Insert data into the table
cur.execute("""
            INSERT INTO payload (title, data)
            VALUES (%s, %s);
            """,
            ('medium', json.dumps(getMediumPayload())))

# Insert data into the table
cur.execute("""
            INSERT INTO payload (title, data)
            VALUES (%s, %s);
            """,
            ('large', json.dumps(getLargePayload())))

conn.commit()

cur.close()
conn.close()
```

- In your terminal run:
  `export DB_USERNAME="sammy"`

export DB_PASSWORD="password"
python init_db.py

- Open your postgres session again to confirm that the table was created:
  psql postgres
  \c benchmark_db
  SELECT * FROM payload; (output might be large due to payload sizes)
  \q;
- Ensure that the table has the right owner. Open another session as admin:
  psql postgres
  \c benchmark_db
  \dt
  (if table payload has the wrong owner)
  ALTER DATABASE benchmark_db OWNER TO usr;
  ALTER TABLE payload OWNER TO usr;

# Get db data:

## Flask app:

- Modify your flask app inorder to create a connection to the db and to retrieve the payloads from the table created earlier (changes mad in file webapp.py):

```
# Importing flask module in the project is mandatory
# An object of Flask class is our WSGI application.
from flask import Flask, render_template
from largePayload import *
from mediumPayload import *
from smallPayload import *
import os
import psycopg2
# Flask constructor takes the name of
# current module (__name__) as argument.

app = Flask(__name__)

def get_db_connection():
    conn = psycopg2.connect(host='localhost',
```

```
                            database='flask_db',
                            user=os.environ['DB_USERNAME'],
                            password=os.environ['DB_PASSWORD'])
    return conn
# The route() function of the Flask class is a decorator,
# which tells the application which URL should call
# the associated function.
@app.route('/')
# '/' URL is bound with hello_world() function.
def hello_world():
    return 'Hello World'


@app.route('/small-json-payload')
def small_payload():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
                SELECT data FROM payload
                Where title='small';""")
    payload = cur.fetchall()
    cur.close()
    conn.close()
    return payload.__str__()


@app.route('/medium-json-payload')
def medium_payload():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
                SELECT data FROM payload
                Where title='medium';""")
    payload = cur.fetchall()
    cur.close()
```

```
    conn.close()
    return payload.__str__()

@app.route('/large-json-payload')
def large_payload():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("""
                SELECT data FROM payload
                Where title='large';""")
    payload = cur.fetchall()
    cur.close()
    conn.close()
    return payload.__str__()
# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the application
    # on the local development server.
    app.run(debug=True)
```

- To run the flask app, run the following:
  export DB_USERNAME="sammy"
  export DB_PASSWORD="password"
  export FLASK_APP=webapp
  Flask run

## Remix app:

- Create a client that will connect with the postgres db in a separate file called database.server.ts:

```
const { Client } = require("pg");
const client = new Client({
 user: process.env.PGUSER,
 host: process.env.PGHOST,
 database: process.env.PGDATABASE,
 password: process.env.PGPASSWORD,
```

```
  port: process.env.PGPORT,
});
client.connect();
export function getDB() {
 return client;
}
```

- In each of your routes' loader use this client to retrieve payload data:

```
import { json } from "@remix-run/node";
import { getDB } from "~/models/database.server";

export const loader = async () => {
 const client = getDB();
 const res = await client.query(
   "SELECT data FROM payload Where title = 'small'"
 );
 return json({
   data: await res,
 });
};
```

# Run benchmarks

Different tools can be used to run benchmarks on both flask and remix app. For the purpose of simplicity, due to time constraints, we will be avoiding graphing tools for these benchmarks. The two different tools here will be **autocannon**, and **oha** (feel free to use other http/node benchmarking tools)

- Run this command for each endpoint for the remix and flask apps:
  Oha: oha -z 10s -c 100 --http-version 1.1 http://localhost:3000/large-json-payload
  Autocannon:
  autocannon -c 100 -d 30 -p 10 http://127.0.0.1:5000/medium-json-payload
- Benchmark results can be seen in the repo linked below

Note
Following the discussion on this thread:
https://github.com/remix-run/remix/discussions/3835, it is our understanding that Remix

servers, with default settings, have additional tools enabled that impact its performance. Due to time constraints, we will not be modifying/disabling these tools right now. Future changes will be made when possible to get a full picture of the performance stats for a Remix server.

# Reference

- Github repo: https://github.com/guepjo/remix-python-benchmark/tree/main