Data Dropzone Challenge

Sivasundar Karthikeyan | Data Science Graduate

Secure and Scalable RESTful Service using Python – GCP

A web service using Python’s Flask module which allows the users to perform RESTful operations securely via OpenAPI based Swagger UI. The service is deployed on Google Cloud Platform’s App Engine which can be scaled conveniently.

Objectives

- Requirements:
  - To design and implement a RESTful service which allows users to send their data via POST request
  - To transform the data, to store the original and transformed data in appropriate data storage
  - To design a visualizable web service and document the RESTful service using Swagger UI
  - To use a cloud service which allows to scale the RESTful service and data storage at ease
  - To secure the API endpoints
  - To find approaches for testing, integrating and deploying the service conveniently
- Feature(s):
  - To include “nice-to-have” functionalities in addition to the above

Deliverables

- An all-in-one web service
  - which allows users to securely connect with the service
  - which allows users to perform CRUD operations
  - which scales both ways at ease as and when required
- A UI using Swagger which allows users to perform CRUD operations conveniently

RESTful Service Implementation

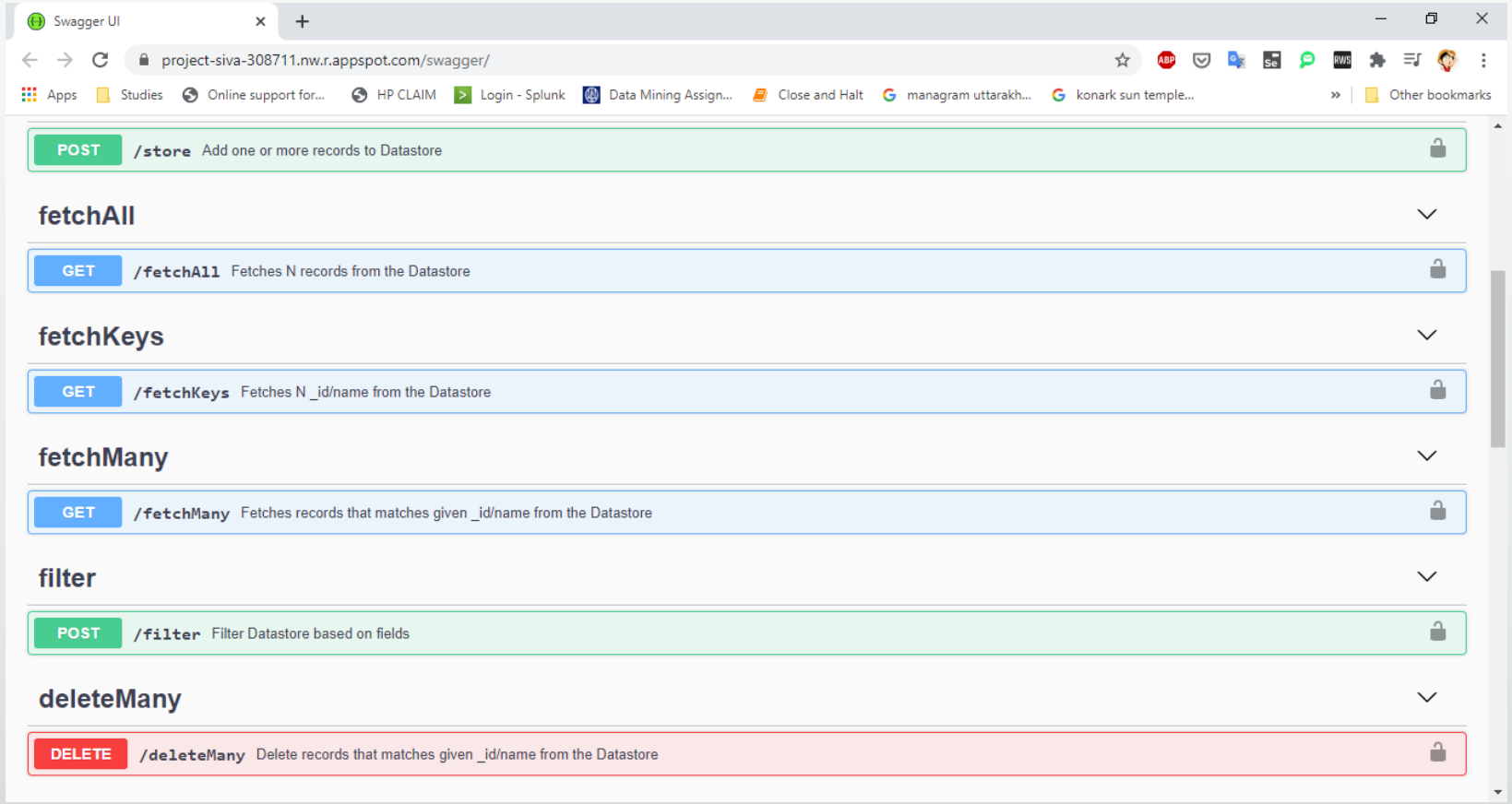
- Service has 6 endpoints related to CRUD operations
  - \store [POST]
    - Add one or more records to Datastore
  - \fetchAll [GET]
    - Fetches N records from the Datastore
  - \fetchKeys [GET]
    - Fetches N _id/name from the Datastore
  - \fetchMany [GET]
    - Fetches records that matches given _id/name from the Datastore
  - \filter [POST]
    - Filter Datastore based on fields
  - \delete [DELETE]
    - Delete records that matches given _id/name from the Datastore
- \store was the only required endpoint and remaining are “nice-to-have” features

Data Storage

- There are many options available on GCP each with their own perks
  - Big Table
    - For NoSQL like MongoDB with better performance
  - Storage
    - For file storage like AWS S3
  - SQL
    - For RDBMS data like Redshift
  - Spanner
    - For distributed and scalable RDBMS data
  - Datastore
    - For NoSQL with some SQL support

- Possible approaches
  - We could push the JSON data in a file, store it in Cloud Storage and store the file path in Cloud SQL
  - We could store the data as a key-value pair in Big Table or Datastore which allows us to access the data without parsing files
- Implemented approach
  - I decided to use Datastore so that we can have the original data alongside the transformed data for access ease

Swagger UI - Documented UI for API Requests



Scalability

- Google App Engine
  - A powerful service provided for web and mobile applications
  - Provides automatic, basic and manual scaling
  - I chose automatic scaling with a minimum of 1 instance and a maximum of 5
  - When the first instance reaches 65% of the CPU utilization, it triggers for starting the next instance automatically
  - The triggered instances will shutdown based on traffic and CPU utilization like how they started
- Datastore
  - Big Table is one of the USPs for GCP and Datastore extends the Big Table
  - This allows us to exploit Big Table features along with simple SQL features
  - Datastore offers high performance due to its distributed storage environment

Security

- We could implement various security measures to secure the API endpoints
- Possible approaches
  - Basic authentication with username and password
  - JWT access and refresh tokens with Bearer or OAuth
  - Restricting IP addresses or devices IDs
  - Using SSL certifications
- Implemented approach
  - I implemented Basic authentication (Kindly use the credentials from app.yaml for Swagger UI)
  - This can be combined with token-based approaches in future
  - SSL certification is provided for App Engines by default

Testing Frameworks

- We could use various frameworks to test our API
- The most used frameworks are
  - PyTest
  - Robot Framework
  - Selenium
  - Cucumber
  - Windmill
- I am familiar Selenium and PyTest
- I am very good at using BS4 which can be used to simulate the requests

Continuous Integration and Continuous Deployment

- I am comfortable with the versioning systems which is basic for CI/CD
- Everything is simple on GCP and CI/CD is not an exception
- Cloud Build can be used for configuring a CI/CD with a few clicks
- Once the git repository and Cloud Build is setup, Cloud Trigger monitors the changes in the git repository
- Any changes in the repository will trigger Cloud Build to pull the updated version from git repository and deploy automatically
- App Engine allows us to run multiple versions in multiple instances and divert the traffic eventually to the newer version
- App Engine allows us to test a new feature with a small user group from real-time before deploying globally

Appendix

- Google Cloud
  - https://cloud.google.com/docs
- Swagger UI
  - https://project-siva-308711.nw.r.appspot.com/
  - https://project-siva-308711.nw.r.appspot.com/swagger/
  - https://swagger.io/solutions/api-documentation/
  - https://petstore.swagger.io/
- Python
  - https://flask.palletsprojects.com/en/1.1.x/
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/
  - https://docs.pytest.org/en/stable/
  - https://www.selenium.dev/documentation/en/
  - https://robotframework.org/