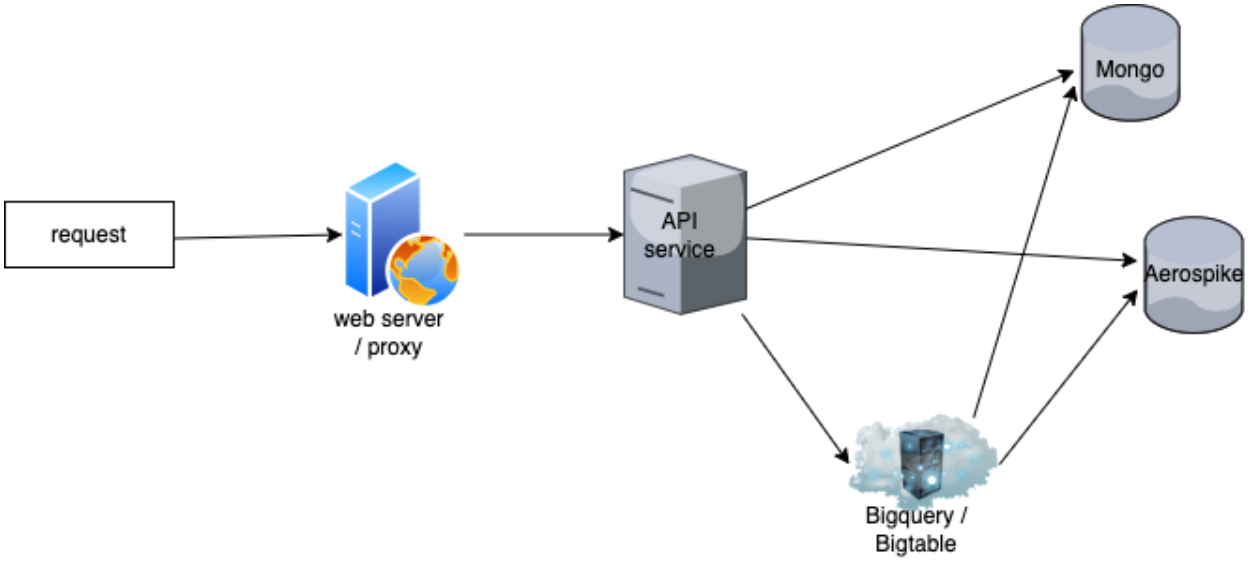
Create a user profile API service (POC)



Overview

Basically this API service utilizes user profile in aerospike, segment metadata in mongoDB, and it uses user mapping to get user's other associated ids and their own information.

It uses REST architecture, which is the most popular approach to build API services. It's built with flask, and other necessary flask plugins.

Because this POC is only for demo purposes, the API is hosted in a development server. The server is able to connect to mongoDB, aerospike, as well as Bigquery tables in google cloud.

```
 * Serving Flask app 'test' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:7000/ (Press CTRL+C to quit)
```

Input and Output

An HTTP request is sent to the service endpoint, and the service responds with result in json format. In the request, user id is included as a parameter in the url.

HTTP request:

curl http://127.0.0.1:7000?id=AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE

HTTP response:


```
{
  "AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE":{
    "attr_1":"value_1",
    "attr_2":"value_2",
    "attr_3":"value_3",
    "attr_4":"value_4",
    "attr_5":"value_5"
  }
}
```


Web server / Proxy

request would go through web server (and / or) proxy before reaching API service, for the purpose of firewall, security, authorization, etc.

MongoDB

Segment metadata is stored in mongoDB, including segment name, type, size, etc.

each mongo query takes about 20ms, but with tens/hundreds of segments per user, the total query time is too long for an API call. So we load all the metadata to memory when the API application initiates, save them in a dictionary, so that future requests only need to read from the dictionary. This dictionary is refreshed periodically to get most updated metadata.

Aerospike

The actual user profile data for each user is stored in aerospike. Querying aerospike takes tens of ms per user, which is sufficient in an API call. In production the aerospike cluster will have multiple nodes to ensure availability and scalibility.

User types in aerospike:
id_type_1 `"111111"`
id_type_2 `"222222"`
id_type_3 `"333333"`
id_type_4 `"444444"`

Bigquery

To query mapping of different user ids, the API service reads bigquery table. Once a user id is found, its other associated ids are retrieved. Then we query those other ids in aerospike to find their associated segments as well.

BQ table read speed is 20s per million records, which is apparently way too slow for an API call. (because BQ is not a key value store, we have to run query to find a user in the dataset) So we can load the latest partition into memory when the API application initiates, and save it in a dictionary. Future request only need to read from the dictionary. But with this method, we aren't able to use the full power of user mapping, because one partition only contains a small number of users. Also loading too much data in the API server's memory is not ideal when the volume is large. A better solution is mentioned in "Potential improvement" section.

Potential improvement

There are numerous ways to improve this POC's performance, scalability. E.g, ensure the servers are in the same geographical region in google cloud. The latency between the servers can be reduced if they are in the same region.

There are multiple tools (fully managed or self managed) in google cloud to deploy API, including cloud run, API gateway, app engine, virtual instances. Google cloud provides good security, easy authorization, easy to scale features.

Querying data from bigquery takes time in the scale of seconds or minutes. Instead we can use Bigtable which has low latency and high scalability. It's fully managed service, no need for manual configuration.

If the volume of requests is huge, this application can be scaled up by adding more servers to host API service. Load balancer will be added to handle incoming requests and distribute them evenly among the API servers.