How to set up an application

To setup an application, we need to have four layers:

1. Frontend
2. Rest API
3. Backend
4. Database

To create a frontend, we can choose many different frameworks. Nowadays, react is one of the most popular frameworks for frontend, as well as used by Glint Solar currently. For the backend, python could be one of the best choices for our application, because the data processing and modelling are based on python. How to choose the database has been answered in the following question. For better maintenance, we can deploy all the three layers in different clusters in the cloud. This solution is robust computing infrastructure, however we need to manage each part ourselves.

As a start up we probably can directly use google APP engine to deploy an application as a container. The following steps are to deploy a web application in the App engine.

1. Open a could shell
2. Clone your application code
3. Install dependency packages for the application
4. Initialize google cloud by running command: gcloud init
5. Deploy the application by running command: gcloud app deploy
6. To view the application: gcloud app browse

We can also consider google cloud run to deploy the application. Google cloud run a docker container containing a webserver. To deploy an application, we need to create a docker image locally to contain the application and upload the docker image to google cloud, and then deploy it by running command: gcloud run deploy --image application_image_file.

How do we store the data? Assume that the data comes to us as the example netcdf.

First we need to decide which database to use to store the data. The wave data is structured data. To store struducted data, we can either use an open source database like postgresql service installed in a cloud cluster, or directly use a cloud based database. For google cloud storages, there are 5 different categories:

1. Bigtable
2. SQL
3. Datastore
4. Storage
5. BigQuery

Bigtable, SQL, BigQuery and Datastore are for structured data. Storage is for unstructured data. It is said that Bigtable is a natural fit for storing time-series data. SQL is much cheaper than Bigtable, but the scalability needs are not too great. Which one is to choose depends on how large our data is.

After we decide the database, then we need to implement a pipeline to ingest netcdf data to our database. If we do not process in real time, we can choose batch ingestion, which typically ingested at specific regular frequencies, and all the data arrives at once or not at all. If we want to have real time ingestion, streaming ingestion could be a better choice.

How do we serve the data? And How do we update the data

We can serve our data using rest API. We can either use BigQuery API or create our own rest API to serve our data. Users access our data through a service endpoint. At the endpoint, we provide methods, such as list, get, delete, insert, update to manage the data. We need to have authentication to define the access capability for different users.

What should the interface between frontend and backend look like?

The communication between frontend and backend are the rest API. The rest API is the interface to provide http requests/responses to serve data. In general, there are two common used methods in http requests, get and post. The rest API has an endpoint that looks like a url. We can add parameters in the url to send data to the backend, and the backend will return the data in the http response. Sometimes, no data will be returned, but we can check the response status. Code 200 for the response status indicates there is no error in the http request.

Answer to bonus question:

If we are able to sort the 120 million lakes according to the area of the polygon and store them in our database, we could serve a faster match by using binary search. The following the pseudo code of binary search. To apply it into lake matching, we need to replace **if target=m then** to **if overlap(target, m) then**. Overlap is a function to calculate if the polygons of the two lakes significantly overlap or not. The time complexities of the new method will be O(m*log(n)), where m is the number of the lakes in the first data source and n is the number of lakes in the second data source.

Binary search

- Get values for list, A1, A2, ….An, n , target
- Set start =1, set end = n
- Set found = NO
- Repeat until ??
  - Set m = middle value between start and end
  - If target = m then
    - Print target found at position m
    - Set found = YES
  - Else if target < Am then end = m-1
    Else start = m+1
- If found = NO then print “Not found”
- End