Our Cloud Development team has been asked to implement a service for text file processing. The service will receive a file ID via an API call and will have to analyze that file. The analysis of a file consists in calling a specific worker for each type of information that needs to be extracted. Due to the fact that some workers might run for a significant amount of time, the API call to the service must be asynchronous – each time a file is submitted for analysis, the service will create a corresponding task. The caller of the service must have the ability to query the status of each of the submitted tasks.

A processing task has the following structure:

- **Task ID** (string) – to be defined
- **Task Creation Date** (DateTime as ISO 8601 string)
- **File ID** (string) – a UUID in standard notation (eg 123e4567-e89b-12d3-a456-426655440000)
- **Task Status** (enum) – to be defined
- **Task Result** (object)
    - **Referenced File IDs** ([]string) – list of UUIDs contained by the file (each UUID references another file)

Requirements

1. (15p) Use Swagger to define a REST API that will expose the following functionality:
    a. (5p) Send a file to analysis that is specified via the ID
    b. (5p) Get information about an analysis (execution status and results)
    c. (5p) Search for the files that contain a particular UUID
2. (10p) Design the Redis storage to store the distributed processing queue and data in the format given by the API requirements
3. (25p) From Swagger, generate the stub for the server in Golang/Python and implement the first two REST methods, mocking the analysis
4. (30p) Implement the workers
    a. (20p) Implement the logic for distributed execution that retrieves tasks from the processing queue and executes them in parallel
    b. (10p) Extract UUIDs from the file contents and write them to Redis
5. (20p) Extend the API to provide the following feature: given a file ID, check if there's a chain of referenced files that starts with that file and contains a loop

Notes

1. The service should scale to billions of files with sub-second query times
2. Example input data will be provided along with this document

Recommended work time: 4h