Causaly Data Engineering Assignment

Welcome to the Causaly data engineering assignment. This assignment is fairly open-ended and is given to all prospective candidates applying for a role in data engineering across all levels of seniority. The only two technologies that we assume are Python 3.x and Docker. We will incorporate your background and the seniority of the role to which you are applying when evaluating your submission. The objective of the assignment is to allow a candidate to showcase their knowledge of basic concepts in data engineering, to build a working solution to a problem, and to provide a clear way for us to reproduce this solution. Beyond this - it is up to the candidate to showcase anything extra that they want to impress us with.

Inputs

In this assignment you are provided with two files:

- **SpaceLifeSciences.xml**: This comprises a set of meeting abstracts for scientific meetings in the Space Life Sciences domain that were published between 1990 - 2010. It has been downloaded using public access from PubMed.gov. PubMed is an online resource housing millions of citations for biomedical literature.
- **Nlmmeetingabstractset_120101.dtd**: This is the document type definition file for the above XML file

Instructions

The objective of this assignment is to implement a simple data processing pipeline, store the result in a data store, and expose two API queries on these data. This should be done within a single virtual machine environment. This task will allow the candidate to demonstrate various architectural choices involved in designing an end-to-end system while at the same time providing the basis for an extended technical discussion on this system during a technical interview.

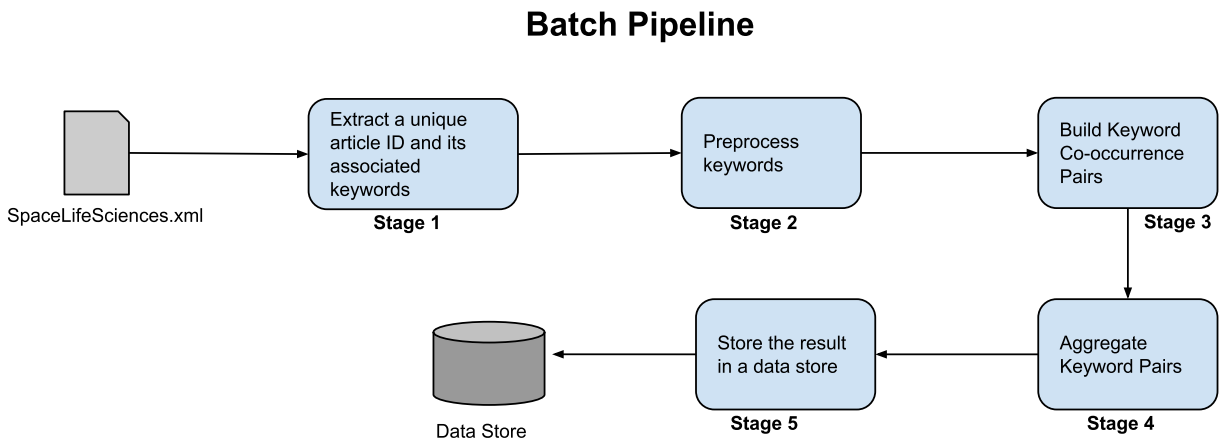
Concretely, the specifications of the VM environment are as follows:

VM Characteristics

The VM should be built using Docker and the OS should be some variant of Unix.

Batch Data Pipeline

The design of the data pipeline should be as follows:



Each blue pipeline stage should be implemented as a single Python 3.x function (you should not need more than a few lines for each of these functions). You can use any tool or approach for pipeline orchestration that you are familiar with and would like to showcase for us.

- **Stage 1**: Use the DTD file provided to identify or construct a meaningful ID to reference each specific meeting abstract and to extract the keywords associated with it as a list
- **Stage 2**: Perform any necessary pre-processing on the keywords
- **Stage 3**: Build up a list of all the pairs of keywords that are found (there will be duplicates if the same pair of keywords co-occur in different meeting abstracts)
- **Stage 4**: Aggregate this list into a meaningful data structure so that for any keyword, you can find all the keywords that co-occur with that keyword, and that you can also keep track of which abstract ID's each keyword pair came from
- **Stage 5**: Write this data structure to a data store

To understand the logic of co-occurring keywords, suppose there are only two abstracts in the original file with ID's 1 and 2. Suppose abstract 1 has keywords `anaemia`, `space`, `blood` and abstract 2 has

keywords `anaemia`, `blood`, `flight`. The first abstract will generate the following pairs of co-occurring keywords: `anaemia-space`, `anaemia-blood`, `space-blood`. The second abstract will generate the following pairs of co-occurring keywords: `anaemia-blood`, `anaemia-flight`, `flight-blood`. Given the keyword `anaemia`, the most frequently co-occurring keyword from this set of two abstracts is `blood` as it co-occurs with `anaemia` in both abstracts and no other keyword occurs with `anaemia` twice.

Data Store

Use any technology that you are familiar with and would like to showcase. This data store should start up when the VM starts up.

Rest API Endpoints

Using any method of exposing a rest API you prefer, you should then expose two API endpoints:

> *getMostCooccurringKeywords()*
> This should take a single string such as “anaemia” and it should return a list of strings (to handle ties) of the keywords that co-occur most often with the string provided.
>
> *addMeetingAbstract()*
> This should take in a single string containing one `<MeetingAbstract>` XML element as defined in the DTD file, which describes a single meeting abstract. The endpoint should process this entry, reusing as much of the batch pipeline functions you defined earlier as possible, and update the data store accordingly. Finally, it should return a status with an optional error message in case of failure.

For the above two cases, please handle any edge cases in any manner you deem appropriate.

Deliverable

Please create a **private** git repository online (you may use a provider like GitHub or BitBucket) and upon contacting us that you are ready, we will provide instructions for sharing this with us (do not share with anyone else). Your repository should be complete and self contained. That is, please remember to include the input files provided and any other files (code,Dockerfiles etc...) that are needed.

Evaluation Criteria

In evaluating each assignment we will be looking at the following aspects of your submission:

- **Reproducibility**: Is it clear how to set up the environment, execute the data pipeline and query the API endpoints?
- **Presentation/Polish**: Is the code quality good? How is the repo organised? Are the instructions well articulated?
- **Architecture**: Do the architectural/technology decisions make sense? What benefits do they provide? Are they integrated effectively with each other?
- **Logic**: Does the solution solve the actual specifications? Are any obvious end cases handled meaningfully?

Please note that we are NOT looking for a production-ready environment i.e. we will not be looking at any other aspects of your setup such as security vulnerabilities, separate containers for separate services, scalability to inputs that do not fit into memory, best practices for production environments etc...