ARTEFACT

Data Engineer Writing Test (select two of the three cases-case 1 is affirmatively chosen item)

submission - A public github repository link where you store your scripts.

CASE 1 - Web Scraping System Design & Development

Purpose

Aim of this case is three fold:

- evaluate your coding abilities
- judge your technical experience
- understand how you design a solution

Assessment

You will be scored on :

- coding standard, comments and style
- overall solution design
- appropriate use of source control

Instructions

- Please use hosted MongoDB at Compose (or any database you like) for storing the results of the scraped documents.
- Candidate should put their test results on a public code repository hosted on Github.
- Once the test is completed please share the Github repository URL to hiring team so they can review your work.
- You are building a backend application and no UI is required, input can be provided using a configuration file or command line.

Challenge - News Content Collect and Store

Create a solution that crawls for comments from JD website, cleanses the response, stores in a mongo database (or any database you like) then makes it available to search via an API.

- Write an application to crawl customer comments from one JD product page (https://item.jd.com/1384071.html) using a crawler framework such as [Scrapy] (http://scrapy.org/) or other technologies. You can use a crawl framework of your choice and build the application in Java, Python or Scala.
- The application should cleanse the articles to obtain information relevant to the comment, e.g. content, author, timestamp etc.
- Store the data in a hosted mongo database (or any database you like) for subsequent search and retrieval.
- Write a Restful API that provides access to the content in the database you use. The user should be able to search for comment by keyword.

Bonus point

If you deploy the solution with **Python** language, **Hadoop Mapreduce or Spark** framework.

CASE 2 - WordCount & Median

Purpose

Aim of this test is three fold:

- evaluate your coding abilities
- judge your technical experience
- understand how you design a solution

Assessment

You will be scored on :

- coding standard, comments and style
- overall solution design
- appropriate use of source control

Challenge - WordCount & Median

One of the first problems you'll encounter in data engineering is Word Count, which takes in a text file or set of text files from a directory and outputs the number of occurrences for each word. For example, Word Count on a file containing the following passage:

So call a big meeting,

Get everyone out out,

Make every Who holler,

Make every Who shout shout.

would return:

```
a                1
big              1
call        1
every       2
everyone    1
get              1
holler      1
make        2
meeting          1
out              2
```

```
shout       2
so          1
who         2
```

The first part of the coding challenge is to implement your own version of Word Count that counts all the words from the text files contained in a directory named `wc_input` and outputs the counts (in alphabetical order) to a file named `wc_result.txt`, which is placed in a directory named `wc_output`.

Another common problem is the Running Median - which keeps track of the median for a stream of numbers, updating the median for each new number. The second part of the coding challenge is to implement a running median for the number of words per line of text. Consider each line in a text file as a new stream of words, and find the median number of words per line, up to that point (i.e. the median for that line and all the previous lines). For example, the first line of the passage

So call a big meeting,

Get everyone out out,

Make every Who holler,

Make every Who shout shout.

has 5 words so the running median for the first line is simply 5. Since the second line has 4 words, the running median for the first two lines is the median of {4, 5} = 4.5 (since the median of an even set of numbers is defined as the mean of the middle two elements after sorting). After three lines, the running median would be the median of {4, 4, 5} = 4, and after all four lines the running median is the median of {4, 4, 5, 5} = 4.5. Thus, the correct output for the running median program for the above passage is:

```
5.0
4.5
4.0
4.5
```

We'd like you to implement your own version of this running median that calculates the median number of words per line, for each line of the text files in the `wc_input` directory. If there are multiple files in that directory, the files should be combined into a single stream and processed by your running median program in alphabetical order, so a file named `hello.txt` should be processed before a file named `world.txt`. The resulting running median for each line should then be outputted to a text file named `med_result.txt` in the `wc_output` directory.

You may write your solution in any one of the following programming languages: C, C++, Clojure, Java, Python, Ruby, or Scala - then submit a link to a public Github repo with your source code. In addition to the source code, the top-most directory of your repo must include `wc_input` and `wc_output` directories, and a shell script named `run.sh` that compiles and runs the Word Count and Running Median programs. If your solution requires additional

libraries or dependencies, the shell script should load them first so that your programs can be run on any system just by running `run.sh`.

As a data engineer, it's important that you write clean, well-documented code that scales for large amounts of data. For this reason, it's important to ensure that your solution works well for small and large text files, rather than just the simple examples above.

Bonus point

If you deploy the solution with **Python** language, **Hadoop Mapreduce or Spark** framework.

CASE 3 - SQL

Please write SQL scripts to solve the following problems, this is a rough description, feel free to define your table name and stuff in SQL.

1. Given a payments transactions table - **transanction_timestamp|transaction_id|customer_id|product_id**, return a frequency distribution of the number of payments each customer made. (I.E. 1 transaction — 100 customers, 2 transactions — 50 customers, etc…)
2. Given the same payments table **transanction_timestamp|transaction_id|customer_id|product_id**, return the cumulative distribution. (At least one transaction, at least two transactions, etc…)
3. Given a table of — **friend1_id|friend2_id**. Return the number of mutual friends between two friends.