Data Engineer Exercises

Instructions: Provide your responses to **at least one** of the following exercises **of your choosing**. If you would like to be considered for a senior-level role, consider doing two or more exercises. Feel free to use references or open-source code from the Internet as long as you give credit to them appropriately. Do not post or share these exact questions with anyone else, or post them publicly online. Share your responses directly with the Mox recruiter only. Be prepared to answer additional questions about the topics of the exercise(s) you chose. Use common sense and explain any assumptions you make if you're unsure about something.

Exercise 1: Data structures and performance

You are provided a word dictionary dataset with terms and definitions. There's currently 10 million elements in this dictionary, which is small enough to comfortably fit just a single copy of it all into local memory of your dev instance. The following questions assume using a native implementation using data structures in your language of choice without using external database systems or libraries such as Redis, Postgres, SQLite, etc.

a) Imagine that you want to be able to look up a given term verbatim and retrieve the definition, or throw an error if the term doesn't exist. What native data structure would you use to store this dataset and what's the Big O notation for retrieving an entry from it?

b) Imagine instead that you want to be able to look up a given entry and return the next term in the dictionary that follows that entry. The provided entry may or may not exist in the dictionary. For example, if provided the entry "orangd" which does not exist in the dictionary, you want to return the term and definition for "orange". What data structure would you use to store this dataset and what's the Big O notation for the above request? Provide code snippets to help explain how you would use the data structure to achieve this functionality.

c) You are now providing a service to display this dataset for a frontend dictionary app. It needs to have the ability to allow people to flip to an arbitrary letter, e.g. "G", and retrieve a pageful of entries (page size varies depending on the client's individual settings) starting from that letter, as well as flip back and forth to the next and previous pages on the UI. What data structure would you use to store this dataset for this service, and are there any changes you would need to make to the contents of this data to meet this requirement? Provide code snippets to help explain how this would work.

d) Imagine that you need to do all of the above but this time your dataset will be saved to a database of your choice (please specify which you would prefer to use and why). Describe what settings and configurations you would apply to your db/tables to get the best performance for use cases a/b/c. Compare and contrast each of your above native implementation's performance (both Big O and real world) to that using a database. Finally, draw a conclusion of all the pros and cons of a native vs database solution, including performance and other factors you consider important for such a service, and outline how you would architect such an application to handle any potential issues that you think it may encounter in production.

Exercise 2: Spark ETL

Consider the dataset provided here: https://www.kaggle.com/smid80/coronavirus-covid19tweets-early-april. Complete the following using Spark. Package your code in a zip such that 1) the dataset is downloaded on run-time instead of included in the zip. Feel free to use this test API key: {"username":"testexerciseuser","key":"1ab7db4952d8bca38e7357e5d4dbdd35"} or create your own. 2) The code runs easily with a single command on any machine without needing the user to install a specific version of Spark or whatever packages required.

1) Create a derived dataset with the following information.

   tweet_date: the date partition from the file name **user_id**: the user_id from the source dataset
   num_tweets: the number of tweets from the given user on this day **hashtags**: an array of hashtags found in all the text fields for that user on that day. For example, if I tweeted "Hello #world" and then "I'm a #data engineer" on one day this field should look like: ["#world", "#data"] **tweet_sources:** for this user-day combination, a map of which source program they used, and the number of tweets from that source. For example, if I sent two messages with my Android phone, and three using Tweetdeck, this field should look like: {"Twitter for Android": 2, "Tweetdeck": 3} **screen_name**: the screen_name associated with this user_id. If the user changed their screen name throughout the day, then display the screen_name associated with their final tweet of the day.

2) Create another derived dataset with the following information. For each tweet where is_quote = TRUE, provide:

   reply_date: the date partition from the file name of the reply tweet **reply_user_id**: the user_id of the replier **original_user_id**: the user_id of the original user **reply_delay**: if the original user's tweet exists in the dataset, calculate the difference between the created_at of the original user's tweet and that of the replier's tweet **tweet_number**: for the user who replied, the tweet number of this reply in chronological order. For example, if I sent four tweets today (not necessarily replies themselves) before this reply tweet, this would be tweet_number = 5.

3) Create a derived dataset with the following information. Consider whether you would use the source dataset to create this or the #1 derived dataset and explain why.

   user_id: the user_id from the source dataset old_screen_name: the previous screen name of the user new_screen_name: the new screen name of the user change_date: the date the user changed their screen name

Exercise 3: Realtime stream processor

Obtain realtime weather data from the API here: https://openweathermap.org/current . Feel free to create a new account or use our test API key: 86360c8475357ebb01df5334aa34a6ed

Create a realtime streaming application that ingests the input data from the source above. You are welcome and encouraged to use a stream processing framework such as Flink or Spark Streaming. Include documentation on how to run your application. Your application should accept a config file to adjust its behavior. An sample config file may look like:

locations=HongKong,Singapore,Tokyo,Seoul,London,Paris,NewYork
app_key=86360c8475357ebb01df5334aa34a6ed
frequency_sec=15 #fetch data every 15 seconds, can be 5, 10, 15, 20, 30 aggregate_period_sec=60 #aggregate data every minute

Your job should:

1) Ingest the raw data
2) Emit the following into the console standard output for each location, once a minute:

   location: the input location specified in the config datetime: YYYY-mm-ddTHH:mm:ssZ in UTC for the start of the aggregated period avg_temperature: the average of the “temp” field over the aggregated period temperature_diff: the final recorded “temp” minus the first recorded “temp” in the aggregated period description: the most frequent description over the course of the aggregated period. If there’s a tie, pick the one that occurred first. e.g. windy,cloudy,rainy,rainy,cloudy,sunny -> cloudy

   This should appear as a single message with all locations in a list rather than as separate messages.

3) Additionally, emit a second data source that ingests data from #2 and provides the following metric:

   datetime: YYYY-mm-ddTHH:mm:ssZ in UTC for the start of the aggregated period
   avg_global_temperature: the average of the “avg_temperature” for all locations temperature_change: the avg_global_temperature from the current minute minus the avg_global_temperature from the previous aggregated period

Exercise 4: ETL Workflow Management

Imagine that you implemented an workflow management solution for your company and have the following use case that you need to fulfill. Describe in detail how you would architect a solution that can solve the following situations within the system of your choice (Airflow, Luigi, Nifi, etc.)

You have a source bucket in AWS S3 / Azure Blob / GCP Cloud Storage that gets populated with new data every couple of minutes in a directory structure like so, where the subfolder and file prefix corresponds to the latest timestamp of the data in that file:

source/customer_new_customer_event/2021-09-01/07_00_file_123.csv
source/customer_new_customer_event/2021-09-01/07_15_file_124.csv
source/customer_new_customer_event/2021-09-01/07_27_file_125.csv
source/customer_new_customer_event/2021-09-01/07_50_file_126.csv
source/customer_new_customer_event/2021-09-01/08_12_file_127.csv
source/payment_successful_payment_event/2021-09-05/11_30_file_abc.csv
source/payment_successful_payment_event/2021-09-05/11_55_file_def.csv
source/payment_successful_payment_event/2021-09-05/12_17_file_ghi.csv

Your job needs to copy these files over to a target folder in the bucket as a single gzipped file, such that there is 1 file per hour with the most up to date data possible, and cannot be more than 30 minutes late:

For example, the first 4 entries in the source folder above should all exist in one file by 8:20:
target/customer_new_customer_event/2021-09-01/07.csv.gz
And if I check this file at time 07:58, the contents of the file 07_27_file_125.csv should already be included in the combined file 07.csv.gz

Requirements:

a) The number of events and size of files are unknown and may change each day, and your job needs to be able to scale up and down as needed while still meeting the 30 minute timeliness requirement.
b) 99% of the time, the data will arrive in your source folder on time, but occasionally you will get late-arriving, out of order data added up to 1 hour after the expected timestamp, e.g. 11_55_file_def.csv may arrive at 12:50 instead, after the file 12_07_file_ghi.csv has already appeared. You need to be able to detect such late arriving files and add them to your target folder as needed.
c) You have other jobs that depend on the output of this job. You want those jobs to be able to run as soon as the data they need is available, e.g. a job that depends on customer_new_customer_event does not need to wait for payment_successful_payment_event to complete before running. Note that the names of events are not known to you at the start of the run and new ones may appear at any time.
d) Occasionally, the system populating data in the source folder will fail, and will stop sending any data for up to 2 hours for all events, but will eventually resume where it left off. Other times, a particular event won't have any data for a while because there's no new events happening and that's perfectly OK but may warrant a notification to the team. You need the job to be able to handle both situations accordingly.

Write code snippets with documentation and describe what sort of technology, infrastructure, or other configuration/features/design elements that you would need to be able to meet these requirements. If you don't have any solutions for one or more subparts, that's OK. Just write down any ideas that you have that may get you closer and/or what sort of changes you would request from other systems to resolve the issue.