==========

Part 1 - system design

==========

Outdoorsy is the most trusted RV rental and outdoor experiences marketplace on the planet. We have grown from a lofty white-board idea in 2015, to over $525M in GMV and offices worldwide in the US, Canada, Australia, Europe and the UK. We've mobilized the 56+ million idle RVs and camper vans around the world to ensure everyone has the access, choice, and opportunity to safely spend more time outside.

Over 856,000 vacation nights were booked on Outdoorsy in 2019. 93% of reviewed Outdoorsy
bookings receive 5-star ratings. With over 600,000 families and adventure seekers—and 50,000
RVs, travel trailers, and campervans to choose from—that's a lot of sunset selfies taken!

As a RV rental company, we want to build a website for our potential customers and customers to view our **listings**, **plans** and **RV models**. As usual, there is a tracking mechanism behind the website to **track user's activities.**

As a data engineer, you will own the data work from data collection, processing and serve the analytical needs.

1. Design and implement the tracking event to serve the need for the product development. One of the top initiatives is to improve search recommendations for our web and mobile products. In order to enable such a data science solution,
    1. What tracking system are you going to use? Why?

This tracking system I think needs to collect customers' search history, browsing history and location history from the web and mobile products.

To build a stable tracking system, we must guarantee all tracking events have been collected.

The collection system must have four properties, high throughput, highly scalable, high availability, and persistent storage. Kafka cluster has a good performance in these four areas. Its high throughput makes the database has less pressure when the data volume grows dramatically. If one node fails in the cluster, the leader election mechanism will be triggered immediately to keep the high availability. Compared to Flume, Kafka can store data in memory or local disk according to a data retention policy. Zero-copy also is a key to make Kafka has high performance.

In the persistence layer, I recommend Druid if raw data is only used once. Druid supports ad-hoc search and historical analysis. For ad-hoc search, once we ingest new data into Druid, we can query it immediately. For a historical analysis, if we don't need raw data, we can send a request to Druid, then Druid will do a pre-aggregation. For example, 100GB raw data will be 20MB aggregated data.

2. Design and implement the data processing process

    1. Which platform are you going to use?
       AWS

    2. What technologies are you going to use?

       Kafka: message bus for data collection;
       SparkSQL: large-scale dataset processing;
       Spark Streaming/Structured Streaming: real-time data processing;
       Druid: ad-hoc search and historical analysis in the persistence layer

    3. Pros and Cons?
       Pros:Kafka: high throughput, high availability
       Pros:
            Kafka: high throughput, high availability
            Spark: a good performance to process large-scale dataset
            Druid: a good performance for ad-hoc search and historical analysis
       Cons:
            Kafka: it doesn’t have its own monitor, a little bit hard to do a maintenance
            Spark: Consume huge amount of memory in JVM. 1GB data will consume 5GB memory. Tungsten is going to solve this issue.
            Druid: doesn’t support join operation.
    4. What is your data model?

       user: userId, userName, email, phone, home;
       RV model: RV_id, owner, RV Name, model type, rental_status,
       activities: userId, timestamp, tracking_event, latitude, longitude

5. What is your process?

1 Backend Engineers add a log collector and send log events to Kafka by Producer APIs
2 Setup Kafka cluster, like topic design, persistence strategy, backup
3 Submit a Kafka indexing task to the real-time node of Druid, then Druid will pull new added log events and provide ad-hoc search. If needed, using spark streaming/structure streaming to do some complex calculations before ingesting into Druid.
4 Load historical data into Spark to do some aggregations in the early morning day by day.
5 Monitor status of real-time tasks and write a script to bring services back once it fails.
6 Schedule ETL jobs on the Airflow, setup data auto-backfill strategies to keep data high availability.
7 Design APIs which return target data from various databases. This is optional.
8 Routine maintenances to meet constantly changing requirements

3. Considerations

1. This exercise is not expected for a candidate to spend more than one hour to complete, so please help limit the scope by sharing your thoughts and ideas on product specific usage.

2. Some possible implementation includes the kafka tracking, data staging, data processing with python, java and/or other big data technologies, such as spark, hive, pig, flink, etc

4. Open end questions

1. How do you address the scalability, stability and performance issue in your process?

Stability:
Real-time tasks: monitor the process id status once the task has been created. If the process cannot be found, the task will be brought up automatically once we add auto-bringup service.

Offline large-scale task: If the task works day by day, script scans which day's data is still missing in databases at the beginning, then start to backfill missing data after its current ETL job is done.

Scalability:

If we are using a Spark cluster in AWS EMR, sometimes data is huge, sometimes it is small. To reduce budget, adding an auto-scale policy is necessary. Once the total usage of memory is over 80%, we can consider to add one or more data node(depends on how the policy designs). Once the total usage of memory is lower than 30%, we need to reduce the cluster size.

Performance:

For a distributed system, scaling up/scaling out is a fast way to improve the performance, but it costs our budget pool. Another way to improve the overall performance is to tune up calculation algorithms, cache reused data, setup the correct number of partitions and JVM GC strategy tuning

2. How do you handle data freshness?
   determine by user search frequency. Most of users would like to search 7 days data.

How to handle the data freshness depends on our requirements. If our data scientists need to process it immediately, we take this type of data as real-time. If the data is large but our customers or data scientists don't need it immediately, we can schedule an hourly or daily task. Besides, we also need to consider the processing time. If it cannot be ready based on the requirement due to the data size after we do a lot of optimizations, we can start to consider sacrificing a little bit of freshness.

5. Expectations of output

1. Describe your thoughts and answers for each question in Word or editor of your choice

2. Please send them back to Chi-Yi (ckuan@outdoorsy.com)