==========

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?

Our tracking system can have two parts(real-time model, pre-generated recommendation) to enable recommendation service. First, for each user, we create a pre-generated recommendation list. This list is generated by user preference. Once the user login, we will load

the list before he/she search something. One shortcoming is the user must have a long history and the data should be fresh. Otherwise, the pre-generated list may be not accurate Second, deploying a real-time model on the backend. Once the pre-generated list doesn't work(it can be detected by another model to calculate the similarity between top recommendation choices and user's choice), the situation will be found by the detection model and activate a real-time recommendation model, then return a new recommendation list to the user. The real-time model will be updated daily or weekly and is trained by fresh historical data. This is how recommendation system works.

Besides, we also keep collecting user's activities, browsing history, location history, search history and third-party 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?

       Redis: cache system for quick read/write
       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:
           Redis: quick read/write
           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:
           Redis: once the node fails without backup, data will be loss.
           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, age, gender … …
owner: owner_id, owner_name … …
search_history: userId, RV_id, search_time, user_location, departure_date, return_date, order_id … …

5. What is your process?

ETL pipeline:
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

Recommendation:
1 loading pre-generated recommendation list after a user login
2 a detection model to track the user's choice, once it has a low similarity between top recommendation options and user's choice, tracking system will activate a real-time recommendation model
3 the real-time model will generate a new recommendation list for user's choice
4 tracking system will keep tracking all activities and send to Kafka to update real-time recommendation model

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?

To do a recommendation, we have to make a model. Once our recommendation model is done and works well for a long time, but one day a user doesn't click any content in recommendation list or clicks contents that have a low recommendation probability from the model. It means our training set is outdated or incomplete, so our data is not fresh. We should get more recent activities from users and re-train a model. I have some solutions.

First, If our users don't login for a long time, we should fill up this time gap. I found our user can be login by Google account, Apple account and Facebook account. I recommend users login in these ways, then we could find their social activities from these platforms. Then we have more fresh data. If we don't want to buy it, we could create a web crawler to keep fetching fresh data from our users' main page. I used Scrapy, a distributed web crawler, to fetch data from Verizon community weekly. It works good and easy to deploy a web crawling service.

Second, we could recommend users fill in their social media information or login by more types of social media account. Once we have a web crawler and have more platforms to track users' social behavior. The more we know about our users, the more fresh is a recommendation list from our model.

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)