Tiki Products Recommendation System

# Working Backwards

* Users want to discover product they haven’t yet seen that they might enjoy
* Their own behavior (ratings, purchases, views) are probably the best predictors
* As before, availability and partition-tolerance are important. Consistency not so much

# HBase our first choice

* But any NoSQL approach would do these days

# How do movie recommendations get into HBase?

* We need to do machine learning
  + **Spark MLlib**
    - **Spark ALS** can actually generate individual movie recommendations for you
  + **Flink** could also be alternative
* Timeliness requirements need to be thought out
  + Real-time ML is a tall order – do you really need recommendation based on the rating you just left?
  + That kind of would be nice

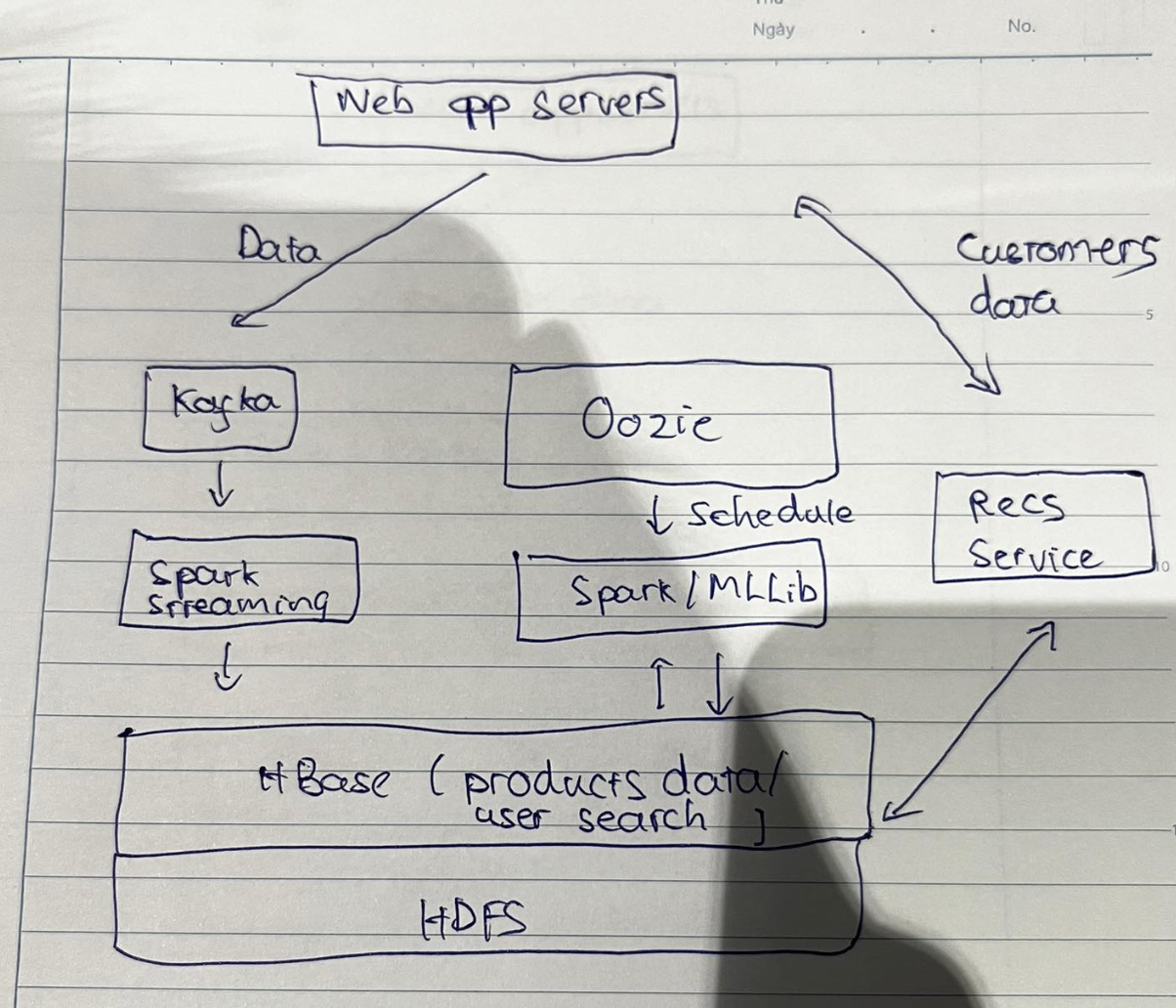
# Creative thinking

* Pre-computing recommendations for every user
  + Isn’t timely
  + Wastes resources
* Item-based collaborative filtering
  + Store movies similar to other movies (these relationships don’t change quickly)
  + At runtime, recommend movies similar to ones you’ve liked (based on real-time behavior data)
* So, we need something that can quickly look up movies similar to ones you’ve liked at scale
  + Could reside within web app, but probably want your own service for this
* We also need to quickly get at your past ratings/ views/ etc.

# Ok Then.

* So, we’ll have some web service to create recommendations on demand
* It’ll talk to a fast NoSQL data store with movie similarities data
* And it also needs your past rating/ purchases/ etc.
* Movie similarities (which are expensive) can be updated infrequently, based on log data with views/ ratings/ etc.

# Architecture

­­

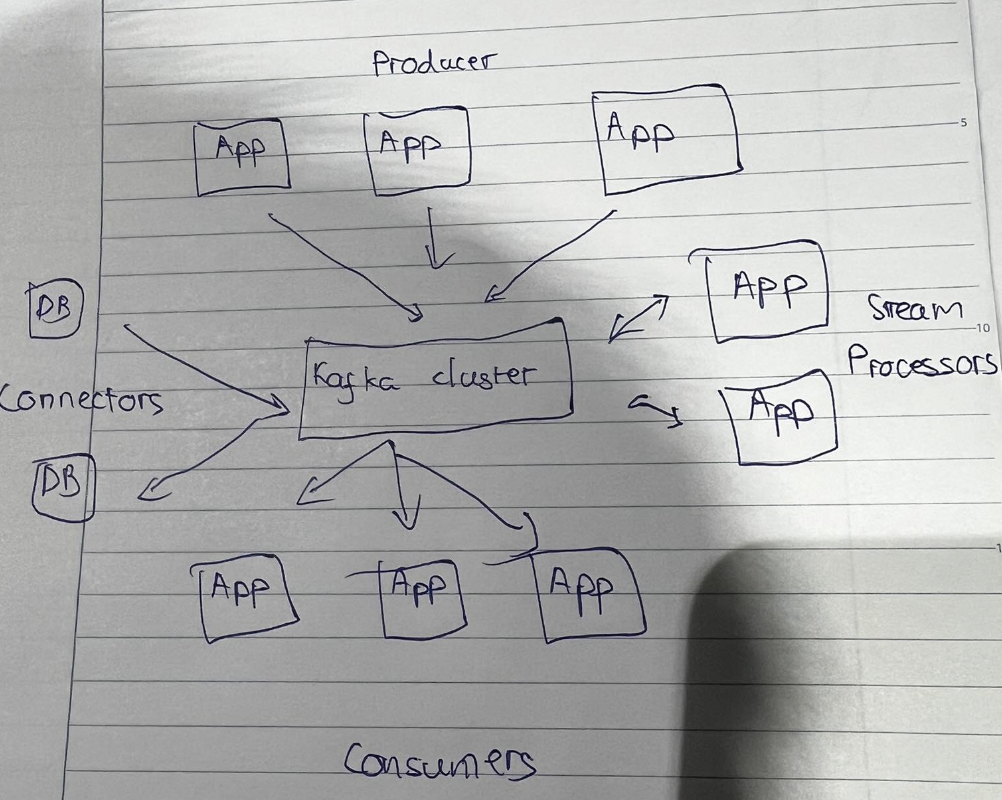
# Scenario

* Data ML: 3 features

# Kafka

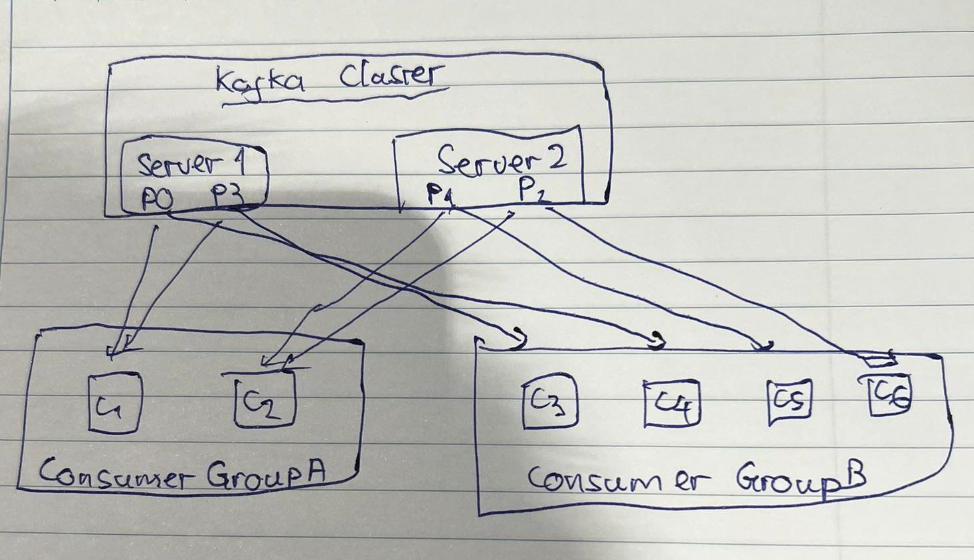
* Kafka is general-purpose publish/subscribe messaging system
* Kafka servers store all incoming messages from *publishers* for some period of time, and *publishes* them to a stream of data called a *topic*
* Kafka *consumers* subscribe to one or more topics, and receive data as it’s published
* A stream/topic can have many different consumers, all with their own position in the stream maintained
* It’s not just for Hadoop

# Kafka Architecture



# How Kafka scales

* Kafka itself may be distributed among many processes on many servers
  + Will distribute the storage of stream data as well
* Consumers may also be distributed
  + Consumers of the same group will have messages distributed amongst them
  + Consumers of different groups will get their own copy of each message



# Visualize

* Products
* Categories