



CSE521 Big Data Analytics **Project**

Submitted to faculty: Prof. Amit Ganatra
Date of Submission: 30th April, 2022

Name	Roll Number
Krina Khakhariya	AU1940208
Arti Singhal	AU1940181
Krunal Savaj	AU1940271

Location-Based-Recommendation of Restaurants System

TABLE OF CONTENT:

ABSTRACT

KEY TERMS

1. INTRODUCTION

YELP DATASET

2. BACKGROUND

A. Apache Spark

B. Elasticsearch

C. Apache Kafka

D. Kibana

E. Apache Zookeeper

F. MLlib Machine learning library

G. Spark Streaming

3. STATIC RECOMMENDATION SYSTEM

4. STREAMING RECOMMENDATION SYSTEM

5. RESULTS

6. FUTURE WORK

7. CONCLUSION

8. REFERENCES

ABSTRACT

Technology has become an integral part of our lives which has made our life easier and more convenient. Now humans have everything at their fingertips for example which place to go on vacation, restaurants to live in, places to eat, and so on. But as they have not visited some places earlier, what becomes important for them is to review to know more about the place, its ambiance, and services. So, ratings and reviews are key for the users to make such kinds of decisions. But what is not right is the availability of a lot of reviews on the internet and it might end up users using the wrong service or product just because they rely on a person of opposite taste. So the main motivation is to come up with a system that users can use for recommendations depending on their needs.

KEY TERMS USED

Static Recommendation System, Streaming Recommendation System, Apache Kafka, Apache Zookeeper, Apache Spark, Elasticsearch, Kibana

1. INTRODUCTION

Reviews and ratings have become a part and parcel of one's life as it helps the user to make a decision and helps them choose from many available options. They have enhanced the chances of buying the desired product for an example of choosing a movie to watch, eating at a restaurant, buying clothes online or other accessories but as everything has its boon and bane so likewise sometimes the reviews can mislead the user and make the, choose the wrong one as every human being has different tastes, choices, and preferences. So to overcome this problem, we have to make a system that helps users in choosing the right one when finding a restaurant according to their needs.

This system has two recommendation systems which first uses Static data and other uses the real-time streaming data. Both of these systems are elaborated on in a later section of the paper.

As in the restaurant business, there is huge competition, the restaurants try to improve their quality to impress and satisfy their customers. We processed tips data on daily basis by using streaming and recommended top restaurants to the user.

Tools like Apache Zookeeper and Apache Kafka are used to stream the static data. Inverted indexes have been made in Elasticsearch and visualized using Kibana. We will discuss them in the latter part of the paper.

Our systems were experimented on the Yelp dataset. The results and future works will be discussed.

Java, Scala, and Python are the languages used for completing the project.

YELP DATASET:

An academic dataset challenge hosted by Yelp has been used as our dataset which comprises reviews, tips, check-in, user information, and business in CSV format.

Review data has User_Id , Review_Id,Stars,Date,Text etc.

Business data has City, State, Latitude, Longitude, Business Id, Name, Stars, etc.

Tip data has Text, User_Id, Business_Id.

The data comprises various business reviews and we have stuck to the Restaurants domain. For the convenience of analyzing, we have converted the JSON format to CSV files using the Python script.

Business

```
{
  "business_id":"encrypted business id",
  "name":"business name",
  "neighborhood":"hood name",
  "address":"full address",
  "city":"city",
  "state":"state -- if applicable --",
  "postal code":"postal code",
  "latitude":latitude,
  "longitude":longitude,
  "stars":star rating, rounded to half-stars,
  "review_count":number of reviews,
  "is_open":0/1 (closed/open),
  "attributes":["an array of strings: each array element is an attribute"],
  "categories":["an array of strings of business categories"],
  "hours":["an array of strings of business hours"],
  "type": "business"
}
```

Figure 1: Business Dataset

Review

```
{
  "review_id": "encrypted review id",
  "user_id": "encrypted user id",
  "business_id": "encrypted business id",
  "stars": star rating, rounded to half-stars,
  "date": "date formatted like 2009-12-19",
  "text": "review text",
  "useful": number of useful votes received,
  "funny": number of funny votes received,
  "cool": number of cool review votes received,
  "type": "review"
}
```

Figure 2: Review Dataset

User

```
{
  "user_id": "encrypted user id",
  "name": "first name",
  "review_count": number of reviews,
  "yelping_since": date formatted like "2009-12-19",
  "friends": ["an array of encrypted ids of friends"],
  "useful": "number of useful votes sent by the user",
  "funny": "number of funny votes sent by the user",
  "cool": "number of cool votes sent by the user",
  "fans": "number of fans the user has",
  "elite": ["an array of years the user was elite"],
  "average_stars": floating point average like 4.31,
  "compliment_hot": number of hot compliments received by the user,
  "compliment_more": number of more compliments received by the user,
  "compliment_profile": number of profile compliments received by the user,
  "compliment_cute": number of cute compliments received by the user,
  "compliment_list": number of list compliments received by the user,
  "compliment_note": number of note compliments received by the user,
  "compliment_plain": number of plain compliments received by the user,
  "compliment_cool": number of cool compliments received by the user,
  "compliment_funny": number of funny compliments received by the user,
  "compliment_writer": number of writer compliments received by the user,
  "compliment_photos": number of photo compliments received by the user,
  "type": "user"
}
```

Figure 3: User Dataset

Tips

```
{
  "text": "text of the tip",
  "date": "date formatted like 2009-12-19",
  "likes": compliment count,
  "business_id": "encrypted business id",
  "user_id": "encrypted user id",
  "type": "tip"
}
```

Figure 4: Tips Dataset

Check-in

```
{
  "time": ["an array of check ins with the format day-hour:number of check ins from hour to hour+1"],
  "business_id": "encrypted business id",
  "type": "checkin"
}
```

Figure 5: Check-in Dataset

2. BACKGROUND

A. Apache-Spark [1]

The Apache Spark application programming interface is based on a read-only multiset of data objects dispersed over a cluster of workstations, a data structure known as the resilient distributed dataset (RDD). This is kept in a fault-tolerant manner. It was created in reaction to limitations in the MapReduce cluster computing paradigm, which mandates distributed programs to follow a specific linear data flow structure: MapReduce programs read data from the disc, map a function across it, reduce the map's results, and then after it will save the reduction results to disc. RDDs serve as a working set for distributed programs that will provide a limited kind of distributed shared memory.

The availability of RDDs makes it easier to design algorithms that are repeated often, which visit their dataset several times in a loop, as well as interactive data analysis, which involves repetitive database-style data querying. Such applications' latency could be lowered by several orders of magnitude. The training algorithms for machine learning systems, which provided the initial impetus for the development of Apache Spark, are an example of iterative algorithms.

A cluster manager and a distributed storage system are required for Apache Spark. Spark supports standalone, Hadoop YARN, and Apache Mesos for cluster management. For distributed storage, Spark can interface with a wide variety, including Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, Amazon S3, MapR File System (MapRFS), Kudu, or a custom solution can be implemented.

B. Elasticsearch [2]

Elasticsearch is a Lucene-based search engine. It's a full-text search engine with a distributed, multitenant capability, an HTTP web interface, and schema-free JSON documents. Elasticsearch is a Java-based search engine that is open source and licensed under the Apache License. Official clients are available in a variety of languages, including Java, .NET (C#), Python, Groovy, and others.

The most popular enterprise search engine is Elasticsearch, which is based on Lucene and is followed by Apache Solr, which is also based on Lucene. Elasticsearch is being developed with Logstash, a data gathering and log parsing engine, and Kibana, an analytics and visualization platform. The three components are intended to be used as part of an integrated solution known as the 'ELK stack.'

C. Apache Kafka [3]

Apache Kafka is a Scala and Java-based open-source stream processing technology developed by the Apache Software Foundation. The goal of the project is to provide a single, high-throughput, low-latency platform for real-time data flows. Its storage layer is simply a 'Massively scalable

pub/sub message queue architected as a distributed transaction log,' which makes it extremely useful for processing streaming data in enterprise systems. Kafka also provides Kafka Streams, a Java stream processing library, and connects to external systems via Kafka Connect.

D. Kibana [4]

Kibana is an open-source Elasticsearch data visualization plugin. On top of the content indexed on an Elasticsearch cluster, it gives visualization features. On top of massive amounts of data, users may construct bar, line, and scatter plots, as well as pie charts and maps. Elasticsearch, Logstash, and Kibana (also known as Elastic stack) can be purchased as a product or as a service. Kibana accesses the data for visualizations such as dashboards, while Logstash sends an input stream to Elastic for storage and search.

E. Apache Zookeeper [5]

ZooKeeper is a centralized service that manages configuration information, names objects, provides distributed synchronization, and manages groups. Distributed systems use all of these types of services in one way or another. There is a lot of work that goes into correcting the bugs and race situations that are unavoidable each time they are introduced. Because these functions are complex to deploy, applications often scrimp on them at first, making them brittle in the face of change and difficult to manage. Different implementations of these services, even when done correctly, result in management complexity when the apps are launched.

F. MLlib Machine Learning Library [1]

Spark MLlib is a distributed machine learning framework built on top of Spark Core that is up to nine times faster than Apache Mahout's disk-based implementations and scales well as Vowpal Wabbit, thanks in part to the distributed memory-based Spark design. Many popular machine learning and statistical algorithms have been built and delivered with MLlib, making large-scale machine learning pipelines easier to manage, including:

1. Classification and regression:
Linear regression, decision trees, support vector machines, logistic regression, naive Bayes classification
2. Summary statistics, correlations, stratified sampling, hypothesis testing, random data generation
3. Dimensionality reduction techniques such as singular value decomposition (SVD), and principal component analysis (PCA)
4. Collaborative filtering techniques including alternating least squares (ALS)
5. Feature extraction and transformation functions

6. Optimization algorithms such as stochastic gradient descent, and limited-memoryBFGS (L-BFGS).
7. Cluster analysis methods including k-means, and Latent Dirichlet Allocation (LDA)

G. Spark Streaming [1]

To execute streaming analytics, Spark Streaming makes use of Spark Core's quick scheduling functionality. It ingests data in mini-batches and then applies RDD transformations to those mini-batches. This design allows the same application code that was created for batch analytics to be utilized for streaming analytics, making lambda architecture easier to implement. However, this ease comes at the cost of a latency equivalent to the duration of the mini-batch. Storm and the streaming component of Flink are two other streaming data engines that handle events one by one rather than in mini-batches. Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets are all supported by Spark Streaming.

3. STATIC RECOMMENDATION SYSTEM

In this system, we have planned to recommend to the user the list of top 5 restaurants based on their location and their preferences in food choices. For better results, we have planned to build the model only if the user has rated at least one of the nearby restaurants. The following steps to follow :

Step 1: Depending on the location of the user, the list of restaurants that are nearby is obtained.

Step 2: There is filtration in the list of restaurants based on the cuisine choice.

Step 3: If the user has already rated the restaurants which are in the list in step 2 then we can build an ALS model using the current user and all users' recent ratings on the list of the restaurants obtained in Step 2.

Step 4: Predicted the user rating on the list of restaurants obtained.

Step 5: Recommendations of the top 5 restaurants are shown.

Step 6: If the user has not rated any of the restaurants, then we have recommended the top 5 restaurants which are nearby.

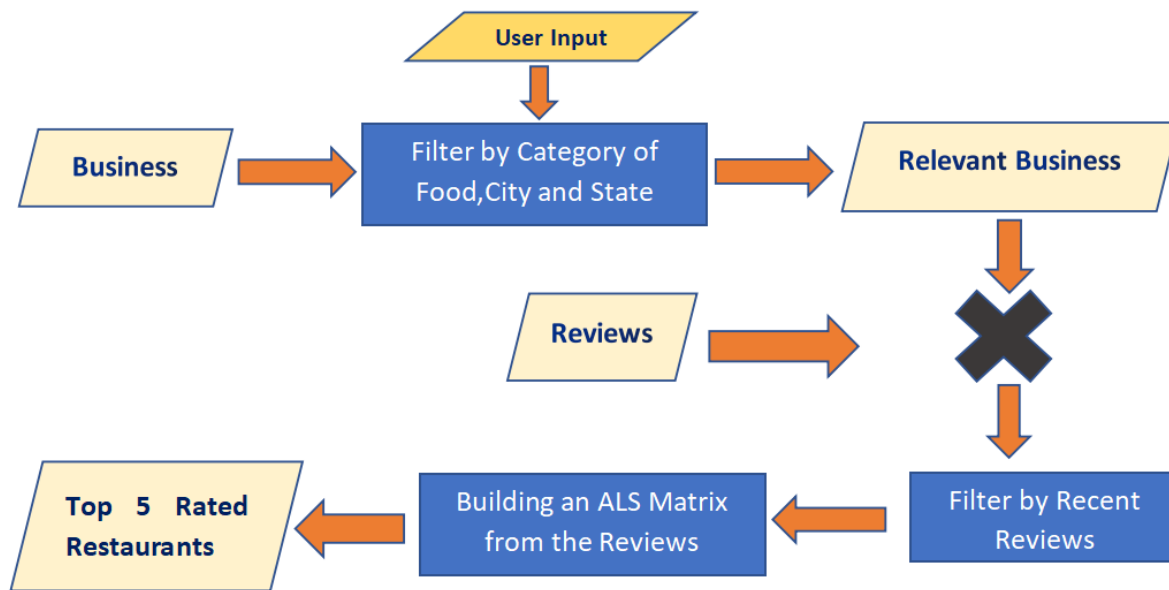


Figure 6: Static Recommendation System flow chart

The system is built on Apache Spark using Scala and its results are discussed in later sections.

4. STREAMING RECOMMENDATION SYSTEM

Due to heavy competition in restaurants, the main focus should be on negative reviews to get better. For example, a restaurant that gets bad reviews on their service should try to improve for the better. We want to recommend the user depending on the reviews given to the restaurant on the same day which helps them to choose better for that particular day. For our project, we did not get the real-time streaming data to work on so we came up with the python program using Kafka and Zookeeper servers which streams the customer reviews of the restaurant nearby the user. Following are the steps :

Step 1: By using the Python program the data was streamed and matched the user's input with the category attribute of the streamed data.

Step 2: Ask the user to provide the details of the cuisine they want to eat. Example: Noodles

Step 3: The restaurant was clustered within the 10 miles of the user location.

Step 4: Then sorted 5 Restaurants based on the user input and the location.

Step 5: The data was sent to Elastic search and visualized it using Kibana

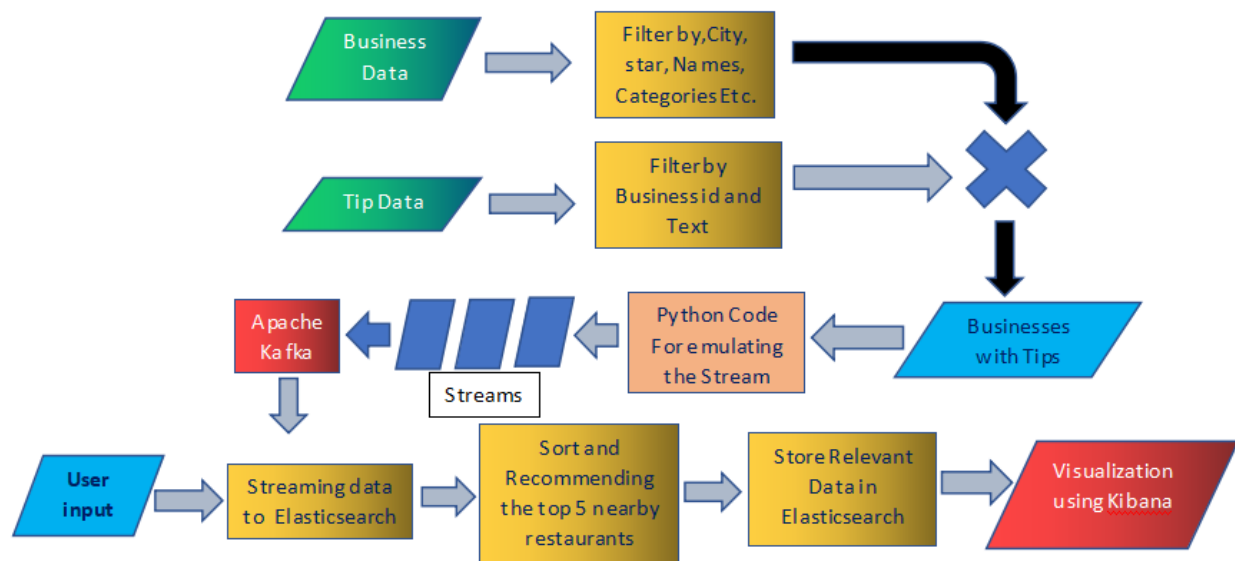


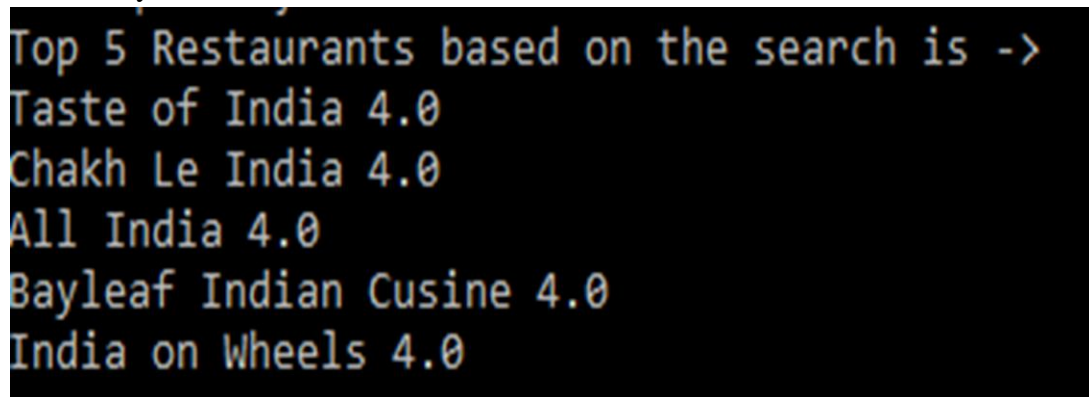
Figure 7: Streaming Recommendation System flow chart

The system is built using Apache Spark using Python, Apache Kafka, Apache Zookeeper Servers, Elasticsearch, and Kibana.

5. RESULTS

The result of the Static Recommendation System is displayed here which consists of the top five restaurants obtained using the ALS model. The restaurant along with the predicted rating is shown below.

Results of the Streaming Recommendation System are also shown in which we can see that output has changed every time the data is streamed, which means that the user is being recommended differently for each new data stream which says that daily reviews on a particular restaurant change and the user will be recommended the best restaurants based on that day review.



```
Top 5 Restaurants based on the search is ->
Taste of India 4.0
Chakh Le India 4.0
All India 4.0
Bayleaf Indian Cusine 4.0
India on Wheels 4.0
```

Figure 8: Static Model Output

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1232,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "yelprow",
        "_type" : "resturant",
        "_id" : "AVv1z1OKlr2Rc5Oa8fH7",
        "_score" : 1.0,
        "_source" : {
          "city" : "Las Vegas",
          "categories" : "[ 'Tapas/Small Plates': 'Tapas Bars': 'Restaurants' ]",
          "full_address" : "3900 Paradise Rd Eastside Las Vegas: NV 89169",
          "longitude" : "-115.1544802",
          "text" : "The bacon wrapped dates are my favorites... Lobster escargot is awesome too",
          "Business_Id" : "8buIrlzBCO70EcAQSZko7w",
          "latitude" : "36.1188384",
          "name" : "Firefly",
          "review_count" : "1357",
          "stars" : "4.5",
          "state" : "NV"
        }
      },
      {
        "_index" : "yelprow",
        "_type" : "resturant",
        "_id" : "AVv1z1OKlr2Rc5Oa8fH8",
        "_score" : 1.0,
        "_source" : {
          "city" : "Las Vegas",
          "categories" : "[ 'Bakeries': 'Cuban': 'Food': 'Bars': 'Nightlife': 'Restaurants' ]",
          "full_address" : "Shalimar Hotel/ Parquin & Restaurant 1401 Las Vegas Blvd S Downtown Las Vegas: NV 89104",
          "longitude" : "-115.149618",
          "text" : "Amazing !!! Great service Mojitos were wonderful Definitely will be back!",
          "Business_Id" : "cJTgJTzEWg5ErWnvaHsUBA",
          "latitude" : "36.155323",
          "name" : "Florida Cafe Cuban Bar & Grill",
          "review_count" : "265",
          "stars" : "3.5",
          "state" : "NV"
        }
      }
    ]
  }
}
```

Figure 9: Streaming Model Output 1

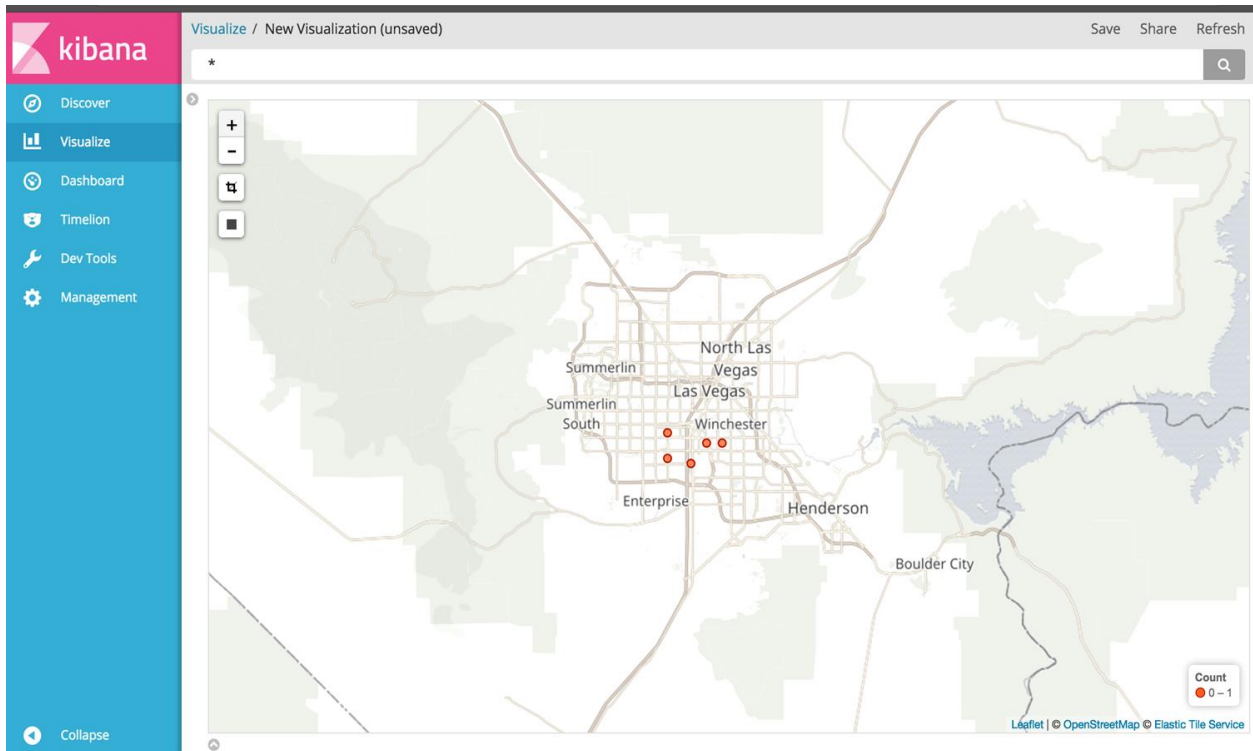


Figure 10: Streaming Model Output 2

6. FUTURE WORK

In the streaming model, we could not get enough data every day, so we experimented by streaming data to Kafka. We have worked only on trending reviews of the restaurants and not the overall rating present. In future analysis, we can perform sentiment analysis on the data of reviews so as to get better results.

7. CONCLUSION

We tried to solve the problem user faced who used to get misled due to a lot of reviews and difference in the choice that exists between humans, so we proposed a new approach in which it recommends user better restaurant based on analyzing the daily reviews which helped the user to choose better on the same day on the basis on the same day reviews. Also, for the future work, the work can be extended by implementing sentimental analysis we can have much better results.

8. REFERENCES

1. https://en.wikipedia.org/wiki/Apache_Spark
2. <https://www.elastic.co/products/elasticsearch>
3. https://en.wikipedia.org/wiki/Apache_Kafka
4. <https://www.elastic.co/products/kibana>
5. <https://zookeeper.apache.org/>