Coronavirus Playback

Replay the development of Coronavirus with real time simulation

Professor: Mrudula Mukadam

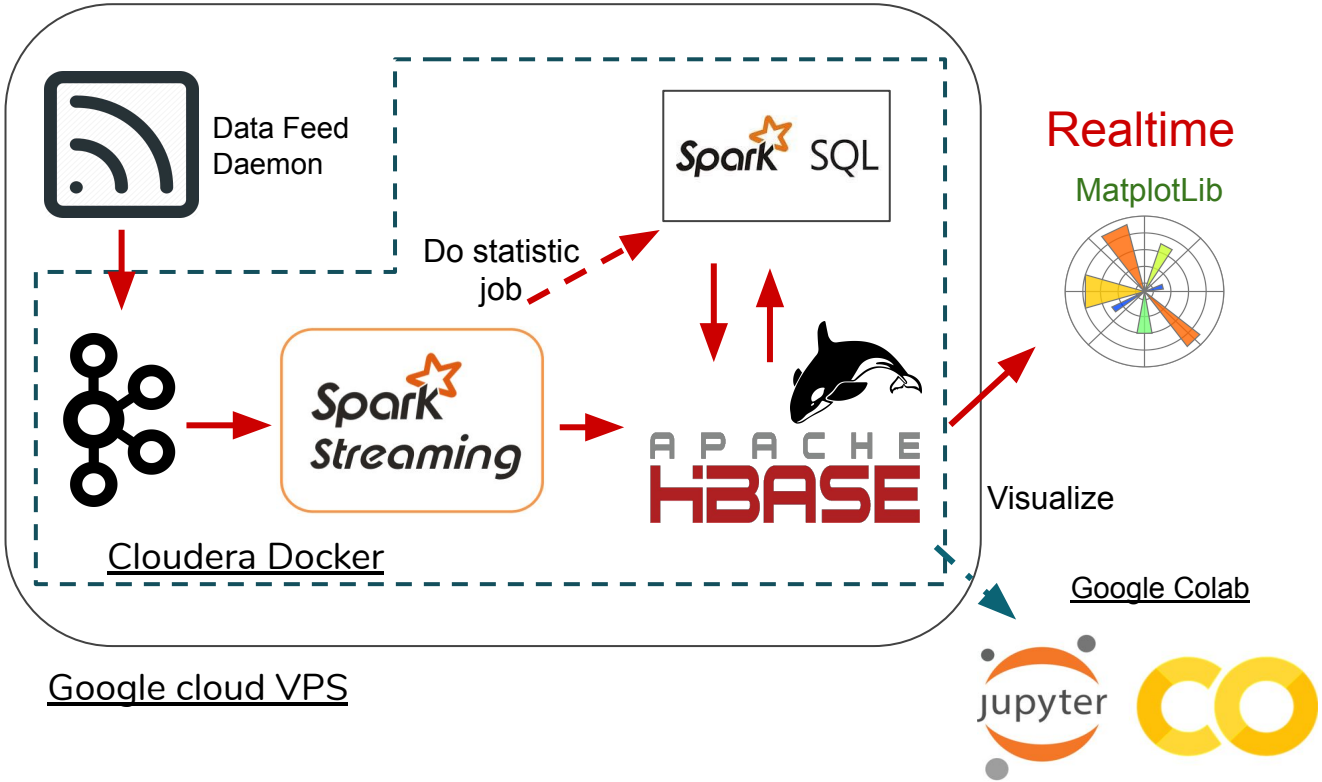
Team:

Le Chi Nhan, Ngo
Tien Khanh, Nguyen

Agenda

- Overview
- Data Feed use python and kafka lib
- Kafka
- Spark Streaming
- HBase
- Spark SQL
- Visualize -
  - Colab - Google
  - Realtime chart with MatplotLib and python

Overview


Data Feed
Daemon
Spark SQL
Do statistic
job
Realtime
MatplotLib
Spark
Streaming
APACHE
HBASE
Visualize
Cloudera Docker
Google Colab
Google cloud VPS
jupyter
CO

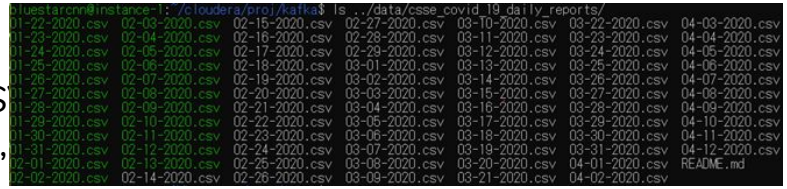
Dataset

1. https://github.com/CSSEGISandData/COVID-19

	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered	Latitude	Longitude
2	Hubei	China	2020-03-15T18:20:18	67794	3085	54288	30.9756	112.2707
3		Italy	2020-03-14T20:13:16	24747	1809	2335	41.8719	12.5674
4		Iran	2020-03-15T18:20:18	13938	724	4590	32.4279	53.688
5		Korea, South	2020-03-15T18:20:18	8162	75	510	35.9078	127.7669
6		Spain	2020-03-15T18:20:18	7798	289	517	40.4637	-3.7492
7		Germany	2020-03-15T18:20:18	5795	11	46	51.1657	10.4515
8	France	France	2020-03-15T18:20:18	4499	91	12	46.2276	2.2137
9		Switzerland	2020-03-15T18:20:18	2200	14	4	46.8182	8.2275

Data specs:

- 01/22 to current, about 85 daily CS
- Total ~75k records in ~80 csv files ,



Challengings:

- Uncertain columns set / pattern -> Optional column picking
- Various time format -> Optional parsing
- Mistaken date -> Apply date of data file

Data Feed cron job

To simulate the realtime processing we get data file from oldest to newest.

A **bash** + **python** when run read **csv** and feed to **kafka**

Interval: Every 2 seconds

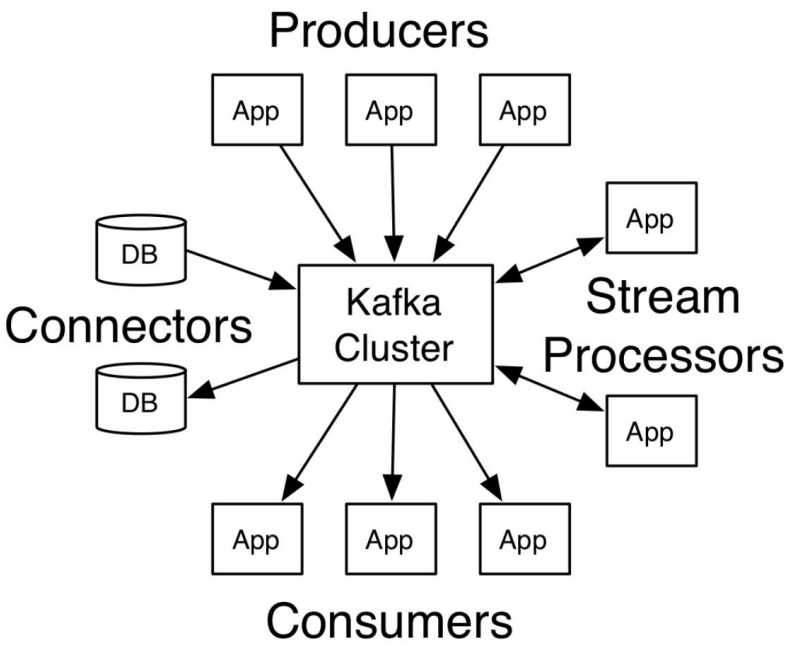
Input: dataset CSV

Periodically publish data into **Kafka**

Kafka - Why Kafka?

- A distributed streaming platform
- Publish and subscribe to streams of records
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.

Supported APIs



Kafka - Consumer implementation (java)

- Kafka Streaming
- Spark Streaming
- HBase

```java
@Log4j
public class KConsumer {

    public static void startConsumer() throws InterruptedException {
        JavaSparkContext sc = SparkConfig.getSparkContext();
        Map<String, String> kafkaParams = KafkaConfig.generateKafkaParams();
        Set<String> topicName = Collections.singleton(KafkaConfig.TOPIC_NAME);
        Configuration hadoopConf = sc.hadoopConfiguration();
        HBaseRepository repo = HBaseRepository.getInstance();

        try (JavaStreamingContext streamingContext = new JavaStreamingContext(sc, new Duration(5000))) {
            JavaPairInputDStream<String, String> kafkaSparkPairInputDStream = KafkaUtils.createDirectStream(
                    streamingContext, String.class,
                    String.class, StringDecoder.class, StringDecoder.class, kafkaParams, topicName);

            JavaDStream<CoronaRecord> recoredRDDs = kafkaSparkPairInputDStream
                    .map(RecordParser::parse).filter(r -> r != null);

            recoredRDDs.foreachRDD(rdd -> {
                if (!rdd.isEmpty()) {
                    repo.save(hadoopConf, rdd);

                    // Refresh the visualization
                    log.info("============= Refreshing the visualization =========");
                    CoronaAnalysisApp.init();
                    CoronaAnalysisApp.generateTotalCasesPilot();
                }
            });

            streamingContext.start();
            streamingContext.awaitTermination();
        }
    }
}
```

HBase tables - corona_cases

```
ROW                                   COLUMN+CELL
 Australia|20200123                   column=cc:confirmedCases, timestamp=1587003705186, value=\x00\x00\x00\x00
 Australia|20200123                   column=cc:country, timestamp=1587003705186, value=Australia
 Australia|20200123                   column=cc:county, timestamp=1587003705186, value=
 Australia|20200123                   column=cc:date, timestamp=1587003705186, value=2020/01/23
 Australia|20200123                   column=cc:deathCases, timestamp=1587003705186, value=\x00\x00\x00\x00
 Australia|20200123                   column=cc:recoveredCases, timestamp=1587003705186, value=\x00\x00\x00\x00
 Australia|20200123                   column=cc:state, timestamp=1587003705186, value=
```

- Storage Corona cases data
- **Key**: country|state|date
- **Column Family**: cc
- **Column**: country, state, date, deathCases, recoveredCases, confirmedCases

HBase tables - pilot

- Storage Visualization data
- **Key**: yyyyMMdd
- **Column Family**: country
- **Column**: country value
- **Value**: confirmedCases

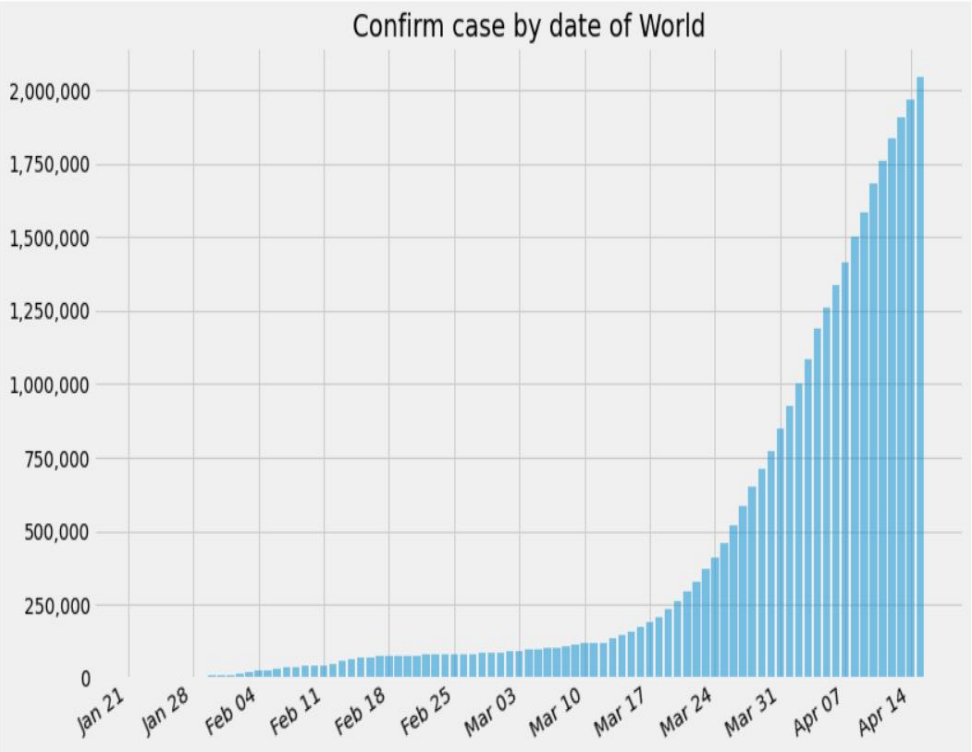
```
ROW                   COLUMN+CELL
 20200122             column=country:CN, timestamp=1587009922682, value=547
 20200122             column=country:JP, timestamp=1587009922682, value=2
 20200122             column=country:KR, timestamp=1587009922682, value=1
 20200122             column=country:US, timestamp=1587009922682, value=1
 20200123             column=country:CN, timestamp=1587009922682, value=639
 20200123             column=country:JP, timestamp=1587009922682, value=1
 20200123             column=country:KR, timestamp=1587009922682, value=1
 20200123             column=country:US, timestamp=1587009922682, value=1
 20200124             column=country:CN, timestamp=1587009922682, value=916
 20200124             column=country:FR, timestamp=1587009922682, value=2
 20200124             column=country:JP, timestamp=1587009922682, value=2
 20200124             column=country:KR, timestamp=1587009922682, value=2
 20200124             column=country:US, timestamp=1587009922682, value=2
```

SparkSQL - Performs analysis

1. Scan records in **HBase**
2. Create **SparkSession**
3. Create DataFrame from the **SparkSession** & **HBase** records
4. Query records in DataFrame with **SparkSQL**
5. Display in console/ save data to **HBase** table

Visualize - Colab

1. Bar chart of confirm case of the World



Visualize Realtime - MatplotLab

- Repeat get data from **Hbase** using **happybase**
- Column extract with **panda** and **numpy**
- Render using `FuncAnimation`

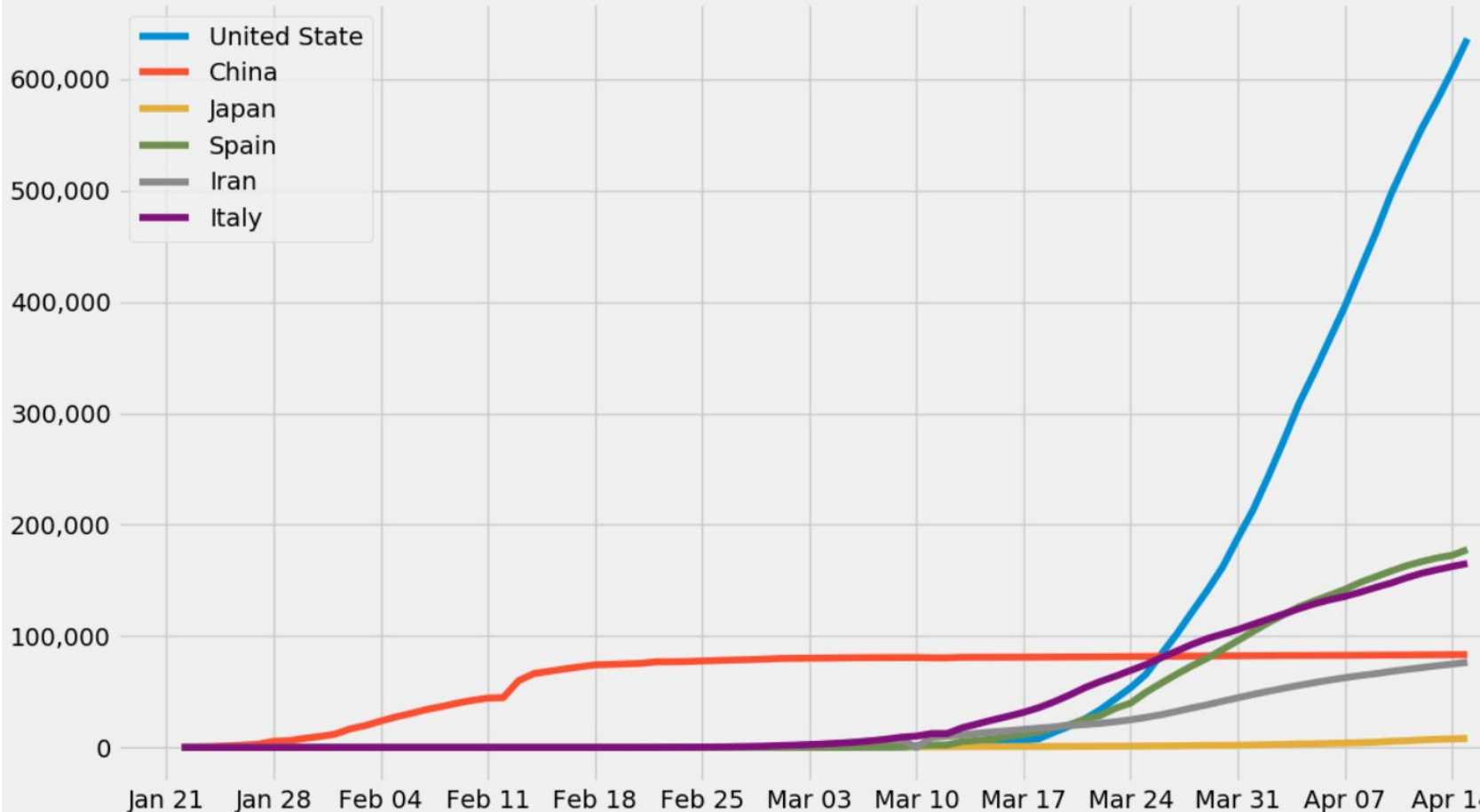
```
df = pd.DataFrame(table.scan(), columns=['date','countries'])
df2 = pd.concat(
    [
        df.drop(['countries'], axis=1),
        df.countries.apply(pd.Series)
    ], axis=1)

dates = pd.to_datetime(df2['date'].str.decode("utf-8"))
debug(dates)
# print('Last', dates[dates.index[-1]])
if isZeroLast(df2)==True:
    return

plt.cla()
setAxisTick(ax)

for (index, row) in metas.iterrows():
    colName = row['col']
    # print(row['col'], row['id'])

    if colName in df2:
        # print(colName)
        cases = df2[colName].str.decode("utf-8").fillna("0").apply(lambda x: int(x))
        debug(cases)
        ax.plot(dates, cases , label=row['title'] )
```

Use happybase to fetch data

```
def connect(tname):
    CONNECTION = happybase.Connection('104.154.17.226', 9090)
    CONNECTION.open()
    # print(CONNECTION.tables())

    _table = CONNECTION.table(tname)
    print(_table)
    return _table


table = connect('pilot')

metas = pd.DataFrame({'col':[
                            b'country:US',
                            b'country:CN',
                            b'country:JP',
                            b'country:ES',
                            b'country:IR',
                            b'country:IT'
                        ],
                    'title':['United State', 'China', 'Japan', 'Spain', 'Iran', 'Italy'] } )
```

Demo checklist - Realtime

- **Kafka** server ON ?
- Topic created ?
- **Streaming** app started ?
- **Hbase** truncated ?
- **Plot** started ?
- ./sendallfile
- Go go go!


United State
China
Japan
Spain
Iran
Italy
600,000
500,000
400,000
300,000
200,000
100,000
0
Jan 21
Jan 28
Feb 04
Feb 11
Feb 18
Feb 25
Mar 03
Mar 10
Mar 17
Mar 24
Mar 31
Apr 07

Sources list

- /proj
  - /kafka
    - **sendkafka.py** -> receive input from stdin and send to kafka
    - **sendallfile.sh**, **sendfile.sh**: -> scan folder and send csv as input std
    - **./start-kafka.sh**, **./recreate-topic.sh**,..
  - /plot
    - **plot.py** -> realtime plot
    - **plothisto.py** -> plot bar chart
  - /cs532
    - /config: **HBaseConfig**.java | **KafkaConfig**.java | **SparkConfig**.java
    - /kafka
      - **KConsumer**.java
    - /hbase
      - **HBaseRepository**.java
    - /model
      - **CoronaRecord**.java