172.31.16.110

Exercise : 1

1.1) At Twitter, the ETL pipelines were difficult to build and maintain. Also, they introduced latency - with nightly batch jobs, business intelligence was being conducted on day-old data. As the pace of business quickened, organizations wanted fresher data more frequently, which stressed the ETL pipelines even more.

1.2) The article mentions counting tweet impressions as an example where the lambda architecture would be appropriate at Twitter. This requires real-time updates as users interact with tweets right now, but also historic counts dating back to when a tweet was first posted.

1.3) Two limitations of the lambda architecture that Twitter found were: (1) complexity - everything needs to be implemented twice, for both batch and real-time platforms, with separate teams often maintaining them, and (2) unclear semantics - aggregate values can fluctuate unpredictably, for example if real-time data is dropped and then reappears later from logs.

1.4) The Kappa architecture proposes that everything is a stream - batch processing is just streaming through historic data. The idea is to use a single stream processing engine rather than separate batch and real-time platforms.

1.5) Apache Beam explicitly recognizes the difference between event time (when an event occurred) and processing time (when the event is observed by the system). The concept of watermarks relates the two to make statements about the completeness of observed data.

Exercise : 2

2.a) The Kappa architecture does away with the batch processing layer and relies entirely on stream processing. It only reprocesses historical data when application code changes, rather than periodically. This differs from the Lambda architecture which has separate batch and real-time processing layers that need to be merged.

2.b) Pure streaming systems like Storm provide very low latency but high per-item processing cost. Micro-batch systems like Spark Streaming trade some latency for increased throughput by buffering data into small batches before processing.

2.c) In Storm's data processing pipeline, data is ingested by spouts and then passed between bolts which do processing, filtering, writing to storage etc. The nodes are connected in a directed graph that represents the data flow. Storm

provides different stream groupings like shuffle grouping and field grouping to control this flow.

2.d) Spark Streaming takes the Spark batch processing engine and enables it to work on real-time data by breaking the stream into small "micro-batches" and processing each batch as a Spark job. This provides sub-second latency by limiting the batch size. The batches are represented as Spark RDDs and can be processed using the normal Spark API.

Exercise : 3

a)put.py :

```
import json
from kafka import KafkaProducer

student = {
   "id": "A20528281",
   "name": "Trinadh_Rayala",
   "eye_color": "Brown"
}


producer = KafkaProducer(bootstrap_servers='172.31.16.110:9092')

student_json = json.dumps(student)
producer.send('students', key=b'student1', value=student_json.encode('utf-8'))

producer.flush()
producer.close()
```

b)get.py

```
import json
from kafka import KafkaConsumer
consumer = KafkaConsumer(
   'students',
    bootstrap_servers='172.31.16.110:9092',
    auto_offset_reset='earliest',
    group_id='student-group'
)
```

```
for message in consumer:
    student = json.loads(message.value)

    print(f"Student Id: {student['id']}")
    print(f"Name: {student['name']}")
    print(f"Eye Color: {student['eye_color']}")

    print()

consumer.close()
```

```
hadoop@ip-172-31-16-110 ~]$ python /home/hadoop/get.py
ey=MYID, Value='A20528281'
ey=MYNAME, Value='Trinadh_Rayala'
ey=MYEYECOLOR, Value='Brown'
```

Exercise : 4

Given Input :

Hi my name is trinadh Rayala, I have taken the

OUTPUT:

```
-------------------------------------------
Time: 2023-11-25 20:12:20
-------------------------------------------
('name', 1)
('is', 1)
('trinadh', 1)
('rayala,', 1)
('i', 1)
('have', 1)
('hi', 1)
('my', 1)
('taken', 1)
('the', 1)
...
```
