

**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN
THÔNG**



**Tìm hiểu về các công cụ xử lý, tích hợp
dữ liệu luồng liên tục**

Nhóm sinh viên thực hiện: 20183571 - Nguyễn Hữu Kiệt
20183574 - Nguyễn Đình Lâm
20183566 - Phạm Minh Khôi
20183568 - Đỗ Lương Kiên

Hà Nội, tháng 05 năm 2022

I. Giới thiệu

Hiện tại dữ liệu lớn đang trở thành một xu hướng mới và cũng dần trở thành một lĩnh vực nghiên cứu chính của thế giới. Với tốc độ tăng trưởng nhanh chóng của dữ liệu dẫn đến nhu cầu phân tích và xử lý thời gian thực cũng được các công ty công nghệ hàng đầu như Google, IBM, Amazon ... chú trọng và phát triển các công cụ Streaming của riêng họ.

Streaming là Công nghệ truyền dữ liệu luồng liên tục. Nếu như trước đây khi xem 1 video ta cần download toàn bộ video đó về thì với streaming chia video thành nhiều phần nên ta chỉ cần loading trước 1 lượng dữ liệu nhỏ. Hiện nay streaming đang dần trở thành một phần quan trọng khi thế giới đang chuyển sang sử dụng điện toán đám mây với một lượng dữ liệu khổng lồ cùng với yêu cầu phải xử lý thời gian thực như:

- Giám sát trang web, giám sát mạng.
- Phát hiện gian lận, tấn công mạng.
- Số lượng click chuột trên web.
- Internet of Things.

II. Các công cụ Streaming

2.1 Kafka

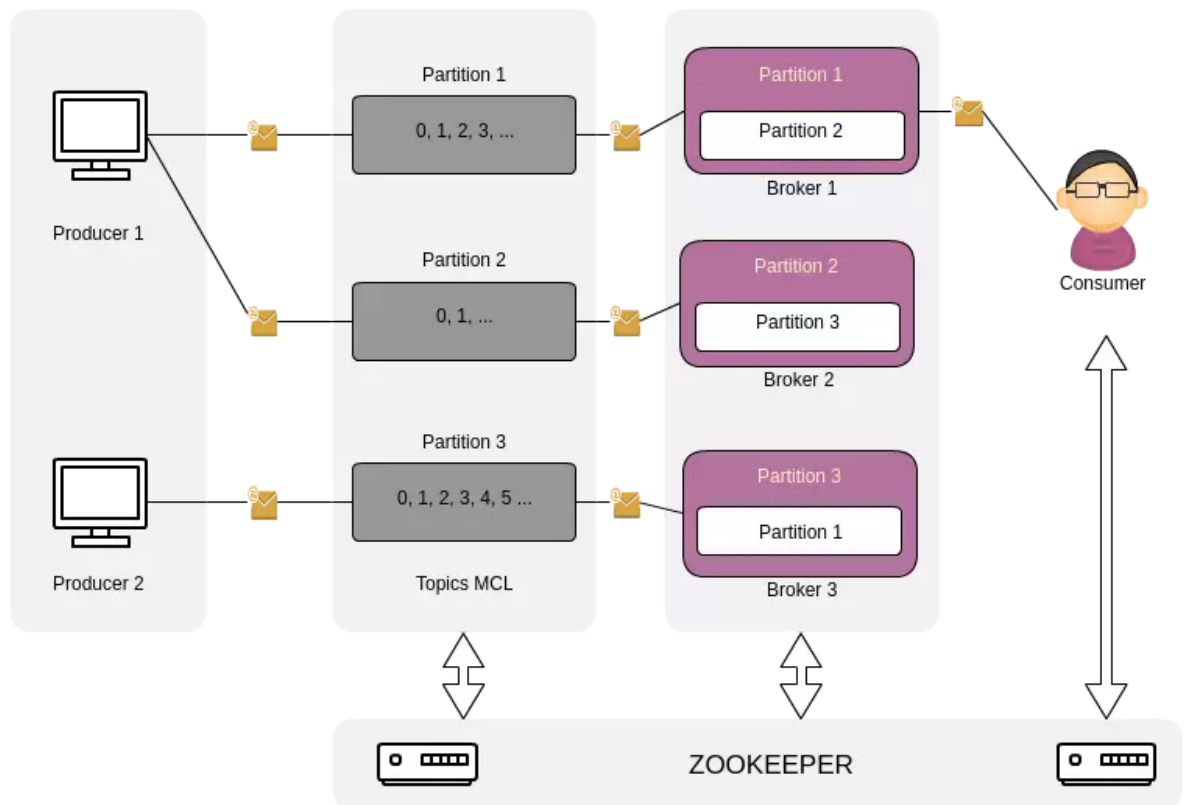
Apache Kafka là hệ thống truyền thông điệp phân tán, độ tin cậy cao, dễ dàng mở rộng và có thông lượng cao. Kafka cung cấp cơ chế offset để lấy thông điệp một cách linh hoạt, cho phép các ứng dụng xử lý lại dữ liệu nếu việc xử lý trước đó bị lỗi. Apache Kafka có những đặc điểm như :

- Tốc độ nhanh: một máy đơn Kafka có thể xử lý số lượng dữ liệu đến hàng trăm megabyte trong một giây.
- Khả năng mở rộng: khi Kafka chạy trên một cụm, luồng dữ liệu sẽ được phân chia và được vận chuyển tới các nút trong cụm, do đó cho phép trung chuyển các dữ liệu mà có khối lượng lớn hơn nhiều so với sức chứa của một máy đơn.

- Độ tin cậy: Dữ liệu vào hàng đợi sẽ được lưu trữ trên ổ đĩa và được sao chép tới các nút khác trong cụm để ngăn ngừa việc mất dữ liệu.

Kafka sử dụng mô hình truyền thông public-subscribe, bên public dữ liệu được gọi là producer còn subscribe nhận dữ liệu theo topic được gọi là consumer. Các thành phần chính của Kafka như sau:

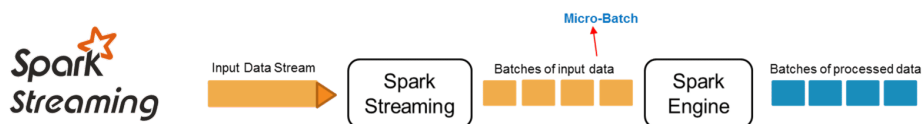
- Topic: Topic tương tự như một table trong CSDL SQL và là nơi chứa các message. Dữ liệu trong kafka theo topic, khi muốn truyền dữ liệu khác nhau hay truyền cho các ứng dụng khác nhau ta sẽ tạo ra các topic
- Partition: một topic được phân chia thành nhiều partition (là nơi chứa dữ liệu cho các topic). Các dữ liệu được lưu theo một thứ tự bất biến (offset) . Một partition sẽ có tối thiểu 1 replica và số lượng replica luôn nhỏ hơn số lượng broker.
- Broker: kafka chạy trên một cụm nhiều máy (node) thì các máy đó được gọi là broker. Broker là nơi lưu trữ các partition, một broker có thể lưu trữ nhiều partition
- Producer: là nơi đẩy dữ liệu từ người dùng vào các partition của topic. Tùy vào việc có chỉ định rõ ghi vào phân vùng nào không thì producer sẽ gửi phân vùng của topic đó hoặc phân bổ đều vào các partition
- Consumer: sẽ đọc dữ liệu từ broker. Kafka là hệ thống sử dụng mô hình truyền thông public-subscribe nên mỗi một topic có thể đc xử lý bởi nhiều consumer khác nhau, miễn là consumer đẩy subscribe topic đấy.



2.2 Spark Streaming

Spark Streaming là một công cụ tích hợp trực tiếp đi cùng nền tảng của Spark và cũng là một trong những công cụ truyền dữ liệu thời gian thực phổ biến nhất. Nhờ sự đóng góp của cộng đồng mã nguồn mở Spark, công cụ đã có được sự cải thiện đáng kể về hiệu suất và giảm độ trễ trong các phiên bản gần đây. Spark Streaming có hỗ trợ đa dạng các ngôn ngữ lập trình như java, python và scala cùng các thành phần khác trong nền tảng Apache Spark.

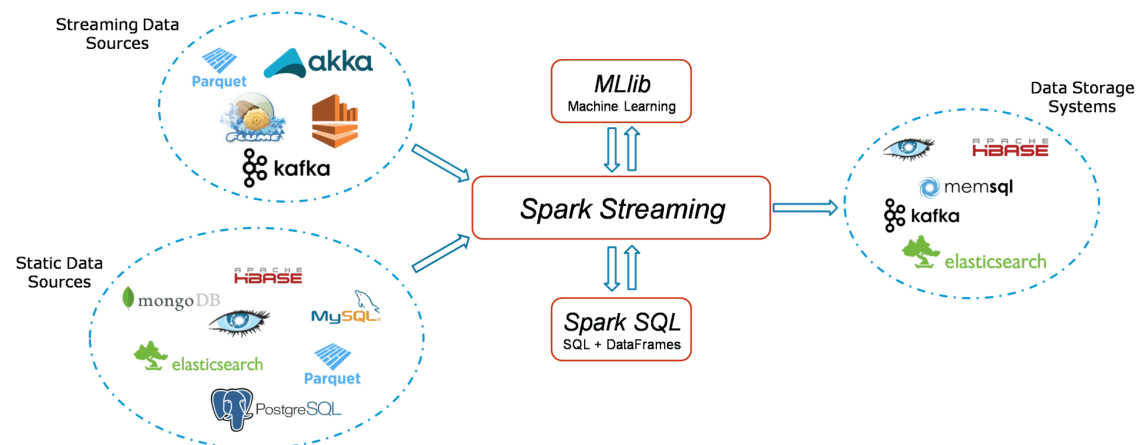
Spark Streaming chia luồng dữ liệu thành các lô X giây được gọi là các dòng, bản chất là một chuỗi các RDD. Spark xử lý các RDD đó bằng các Api spark và kết quả được trả về theo từng đợt.



Một hệ thống workflow của spark streaming bao gồm 4 giai đoạn:





1. Dữ liệu đẩy vào Spark Streaming từ các nguồn realtime streaming như Kafka, AWS, .. hay static như Hbase, Mongo DB, MySql, ...

2. Từ Spark Streaming Dữ liệu có thể đưa vào MLlib để áp dụng mô hình học máy.
3. Hoặc dữ liệu cũng có thể đưa vào Spark SQL để truy vấn.
4. Cuối cùng sau khi thao tác với dữ liệu nó sẽ được lưu vào database hoặc file system



2.3 Apache Flink

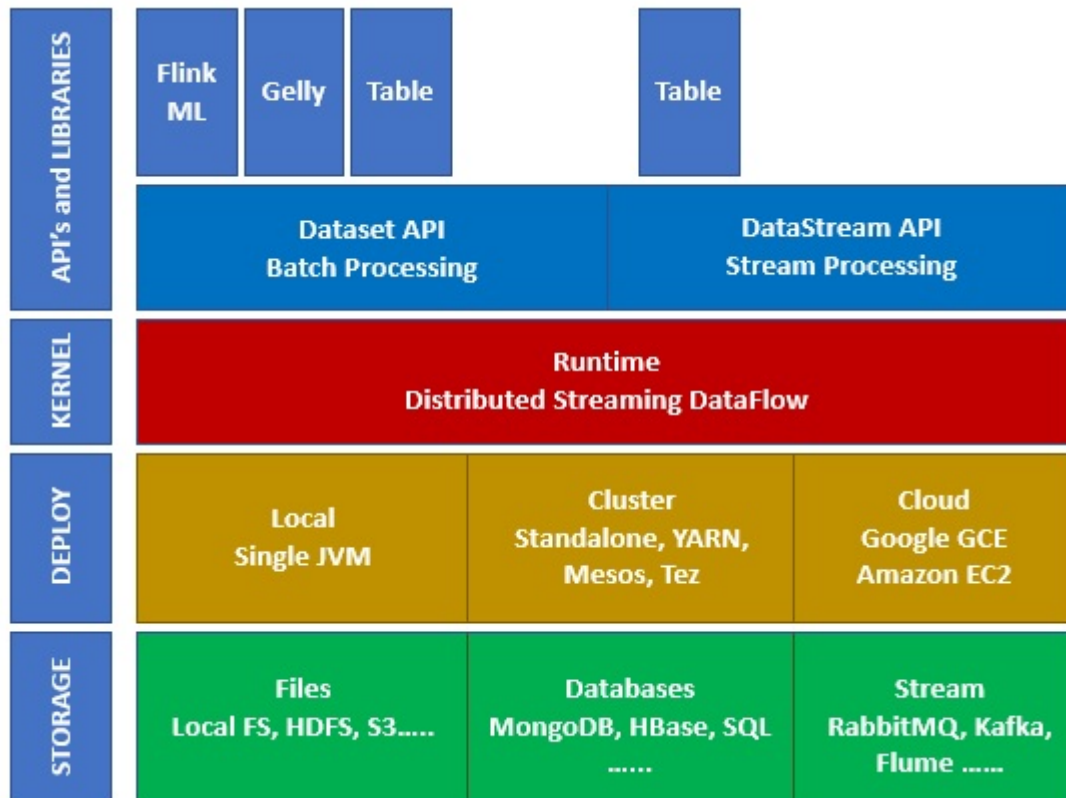
Apache Flink là một công cụ thuộc thế hệ thứ 4 của Apache.

			
✓ Batch	✓ Batch ✓ Interactive	✓ Batch ✓ Interactive ✓ Near-Real Time Streaming ✓ Iterative processing	✓ Hybrid (Streaming +Batch) ✓ Interactive ✓ Real-Time Streaming ✓ Native Iterative processing
MapReduce	Direct Acyclic Graphs (DAG) Dataflows	RDD: Resilient Distributed Datasets	Cyclic Dataflows
1 st Generation (1G)	2 nd Generation (2G)	3 rd Generation (3G)	4 th Generation (4G)

Nó có thể được coi là sự lai tạo giữa Storm và Spark. Nó khắc phục một số vấn đề như giảm thiểu độ trễ và khả năng chịu lỗi dữ liệu

của Spark; bên cạnh đó nó cũng triển khai Apache Beam cho phép xử lý dữ liệu thời gian thực khi nó có thể nhận dữ liệu đầu vào là dạng luồng.

Hệ sinh thái Apache Flink bao gồm 4 lớp:



- Store: Flink có thể lựa chọn nhiều hệ thống dữ liệu để làm đầu vào.
- Deploy: Lớp thứ 2 dùng để quản lý tài nguyên và triển khai chúng ở các chế độ như: Local mode, Cluster Standalone, Cloud
- Kernel: Lớp runtime dùng để xử lý phân tán, có khả năng chịu lỗi, độ ổn định cao.
- APIs & Libraries: Lớp trên cùng và là lớp quan trọng nhất của Apache Flink, có các Dataset APIs đảm nhiệm việc xử lý theo lô, và Datastream APIs phụ trách xử lý theo luồng. Ngoài ra còn có các thư viện phụ trợ như Flink ML cho học máy, Gelly cho xử lý đồ thị và Tables cho truy vấn dữ liệu.

III. Demo

Chạy ứng dụng wordcount trên pipeline cả 3 tool: kafka, spark streaming và flink

3.1 kafka-producer tạo ra các messages ngẫu nhiên gửi word ngẫu nhiên từ one tới ten lên queue

```
10:24:14.791 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (a486c6fc-e5d1-45fd-97b9-4a62e43ce4d2, three) to topic example @ 165356065482.
10:24:14.957 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (2ee79319-2918-4fdf-b8fb-970b768ee77d, five) to topic example @ 1653560654953.
10:24:15.064 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (b47b5acd-3570-4e85-9706-a21317d39dc0, one) to topic example @ 1653560655057.
10:24:15.168 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (55f1ec5d-99c9-4763-8a43-d06ff8a20205, three) to topic example @ 1653560655166.
10:24:15.314 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (4c8b381b-74fd-4f15-ab0a-fc4437e696d3, eight) to topic example @ 1653560655301.
10:24:15.416 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (dddbb024-255b-4db1-97b7-07b08beee752, nine) to topic example @ 1653560655415.
10:24:15.518 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (da8cb971-a1f0-4a01-b090-03fce6a5c517, one) to topic example @ 1653560655517.
10:24:15.655 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (ea4bd69c-e0cb-471b-bec3-6f28f6541a71, eight) to topic example @ 1653560655641.
10:24:15.761 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (e7f2943a-5401-45a9-84f3-3c74ccf73134, seven) to topic example @ 1653560655758.
10:24:15.868 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (2110a17c-4fac-4e99-a66e-cfbbdbdbde3df, five) to topic example @ 1653560655863.
10:24:15.970 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (2d593493-1846-4281-b297-11e5081b42e0, ten) to topic example @ 1653560655969.
10:24:16.079 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (e03c1c97-8bf7-47af-9ea0-2db518c64a07, one) to topic example @ 1653560656072.
10:24:16.181 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (ec7f9e25-a509-476c-88bd-595719d1825a, one) to topic example @ 1653560656180.
10:24:16.285 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (60398393-9aa8-4524-917b-26019bb4f1a2, three) to topic example @ 1653560656282.
10:24:16.387 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (1c92f1ce-3bbd-4dbd-abb1-2ab4a5eb0752, seven) to topic example @ 1653560656386.
10:24:16.494 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (cadf7d71-2c69-4991-8215-3230cd5837cb, five) to topic example @ 1653560656489.
10:24:16.598 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (9f0330be-cf09-4838-8159-b0d64f36d9b6, eight) to topic example @ 1653560656597.
10:24:16.702 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (2592fe51-8809-4b88-a608-8292ecb8dd34, one) to topic example @ 1653560656700.
10:24:16.804 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (d7aabcb4-cf7d-4317-bf3a-cbdb85a1f851, nine) to topic example @ 1653560656802.
10:24:16.907 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (0720714c-aefb-45e6-b5fc-d74f2360e6e9, nine) to topic example @ 1653560656904.
10:24:17.012 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (25dd7af3-f9a1-4f2a-a6af-1dfe865581c8, two) to topic example @ 1653560657008.
10:24:17.114 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (a2644f4e-1a00-4666-80ea-162fcb35e05, four) to topic example @ 1653560657112.
10:24:17.219 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (e1565080-8085-4bf2-8cfd-aaaf5b108656, one) to topic example @ 1653560657215.
10:24:17.325 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (31db67c5-f4b1-4fe5-9a09-f22935f69cca, nine) to topic example @ 1653560657322.
10:24:17.429 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (18291cb0-8200-4b41-af2f-339440052f32, seven) to topic example @ 1653560657425.
10:24:17.533 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (1bd9cc72-13ce-4b03-b5d6-a9aa59c24a4b, five) to topic example @ 1653560657530.
10:24:17.635 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (5840f511-4b37-431f-adb6-dba2986aec3f, four) to topic example @ 1653560657633.
10:24:17.738 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (a18f31b9-09a9-4f08-ba2e-aeca6ab80db9, five) to topic example @ 1653560657736.
10:24:17.843 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (a43b2e87-9185-4cdd-ab3b-60cbdf03b45f, two) to topic example @ 1653560657840.
10:24:17.949 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (2eab5446-38a6-48b3-b225-06178cd96cc0, seven) to topic example @ 1653560657947.
10:24:18.052 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (7abc08b8-9529-4b16-ac16-91d91f72864d, one) to topic example @ 1653560658050.
10:24:18.155 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (2b646fb5-268e-47b0-b519-05379447609d, two) to topic example @ 1653560658153.
10:24:18.256 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (c0605071-e2b3-4b39-b4cb-9b7530367d02, eight) to topic example @ 1653560658255.
10:24:18.358 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (b739248c-6558-4b65-9e2d-9107572fc5f1, ten) to topic example @ 1653560658357.
10:24:18.461 [main] INFO org.davidcampos.kafka.producer.KafkaProducerExample - Sent (a3603f6-f600-4bc9-83aa-f565c6ddffe9, eight) to topic example @ 1653560658459.
```

3.2 spark_consumer xử lý dữ liệu trong trong mỗi batch 5s và tính word count

```
(two,8)
(one,4)
(nine,6)
(six,1)
(three,3)
(five,2)
(four,6)
(seven,8)
(eight,2)
(ten,8)
```

```
Time: 1653560865000 ms
```

```
(two,4)
(one,5)
(nine,6)
(six,6)
(three,3)
(five,5)
(four,4)
(seven,4)
(eight,6)
(ten,5)
```

```
Time: 1653560870000 ms
```

```
(two,9)
(one,6)
(nine,2)
(six,3)
(three,4)
(five,5)
(four,3)
(seven,4)
(eight,7)
(ten,5)
```

3.3 flink_consumer lấy dữ liệu liên tục từ queue của kafka và thực hiện wordcount


```
2> (five,1666)
1> (eight,1711)
2> (five,1667)
1> (eight,1712)
2> (nine,1686)
1> (three,1729)
1> (three,1730)
1> (ten,1688)
1> (eight,1713)
1> (seven,1697)
2> (two,1650)
2> (two,1651)
2> (nine,1687)
1> (three,1731)
2> (four,1695)
2> (four,1696)
2> (one,1688)
2> (five,1668)
1> (seven,1698)
2> (one,1689)
2> (one,1690)
2> (one,1691)
1> (seven,1699)
1> (three,1732)
2> (six,1731)
1> (eight,1714)
2> (two,1652)
1> (seven,1700)
2> (one,1692)
1> (seven,1701)
2> (four,1697)
2> (nine,1688)
2> (nine,1689)
2> (one,1693)
2> (one,1694)
2> (nine,1690)
2> (nine,1691)
1> (seven,1702)
1> (ten,1689)
```

=> kafka chỉ là hàng đợi chứ ko xử lý gì cả. viết 1 file java để tạo message gửi vào queue. Spark đọc queue cũng liên tục nhưng 5s mới xử lý (5s là cấu hình interval time có thể thay đổi được) còn flink thì lấy và xử lý liên tục. Các kết quả từ logs và print sẽ lưu vào logs của docker.