ĐẠI HỌC BÁCH KHOA HÀ NỘI

Trường Công Nghệ Thông Tin & Truyền Thông

Lưu trữ và xử lý dữ liệu lớn

Đề tài: Lưu trữ và phân tích dữ liệu việc làm

Giảng viên hướng dẫn: TS. Trần Việt Trung

Nhóm: 17

Sinh viên thực hiện: Nguyễn Đình Khải - 20215400

Lê Tấn Đạt - 20215339

Nguyễn Văn Trường - 20215495

Vũ Phương Thanh - 20210790

Đỗ Xuân Trọng - 20210865

Hà Nội, 12 - 2024

Mục lục

Chương 1

Đặt vấn đề

1.1 Giới thiệu bài toán

Trong thời đại công nghệ 4.0, dữ liệu đã trở thành nguồn tài nguyên quan trọng, đặc biệt là trong lĩnh vực việc làm. Các nền tảng tuyển dụng trực tuyến, mạng xã hội nghề nghiệp và báo cáo thống kê thị trường lao động liên tục tạo ra một lượng lớn dữ liệu liên quan đến mô tả công việc, yêu cầu kỹ năng, mức lương và xu hướng ngành nghề. Những dữ liệu này không chỉ phản ánh sự phát triển của thị trường lao động mà còn cung cấp thông tin giá trị để hỗ trợ các nhà tuyển dụng, ứng viên và các cơ quan quản lý trong việc ra quyết định.

Tuy nhiên, việc khai thác và phân tích dữ liệu việc làm không đơn thuần là một bài toán nhỏ lẻ mà đòi hỏi sự kết hợp giữa các công nghệ xử lý dữ liệu lớn và các phương pháp phân tích hiện đại. Một hệ thống hiệu quả cần có khả năng thu thập, lưu trữ và phân tích hàng triệu bản tin tuyển dụng và hồ sơ ứng viên từ nhiều nguồn khác nhau. Qua đó, bài toán đặt ra là làm thế nào để xây dựng một hệ thống lưu trữ và phân tích dữ liệu việc làm vừa đảm bảo tính chính xác, vừa đạt hiệu năng cao trong bối cảnh dữ liệu ngày càng đa dạng và khổng lồ.

1.2 Phân tích bài toán trong xử lý dữ liệu lớn

Bài toán lưu trữ và phân tích dữ liệu việc làm đối mặt với nhiều thách thức liên quan đến các khía cạnh của xử lý dữ liệu lớn:

- **Khối lượng dữ liệu lớn (Volume)**: Dữ liệu việc làm bao gồm hàng triệu bản tin tuyển dụng, hồ sơ ứng viên, và thông tin liên quan đến xu hướng thị trường lao động. Việc lưu trữ và xử lý khối lượng dữ liệu này đòi hỏi hệ thống có khả năng mở rộng và tối ưu hóa hiệu năng.
- **Tính đa dạng của dữ liệu (Variety)**: Dữ liệu được thu thập từ nhiều nguồn khác nhau như trang tuyển dụng, mạng xã hội nghề nghiệp, và báo cáo thống kê, với định dạng không đồng nhất (văn bản, hình ảnh, bảng số liệu).
- **Tốc độ xử lý dữ liệu (Velocity)**: Để cung cấp thông tin kịp thời, hệ thống cần xử lý dữ liệu gần thời gian thực, đặc biệt trong việc phát hiện xu hướng thị trường và yêu cầu kỹ năng mới.
- **Tính chính xác và giá trị của dữ liệu (Veracity & Value)**: Dữ liệu thô cần được làm sạch, chuẩn hóa và phân tích để đảm bảo tính chính xác và mang lại giá trị thực tiễn.

Việc áp dụng các công nghệ xử lý dữ liệu lớn như Apache Hadoop, Apache Spark, và hệ thống lưu trữ phân tán sẽ giúp giải quyết các thách thức trên. Đồng thời, các phương pháp trực quan hóa dữ liệu sẽ hỗ trợ việc trình bày kết quả phân tích một cách dễ hiểu và hiệu quả.

1.3 Phạm vi và giới hạn của bài tập lớn

1.3.1 Phạm vi của bài tập lớn

- **Thu thập dữ liệu:** Dữ liệu được lấy từ các nguồn tuyển dụng trực tuyến, mạng xã hội nghề nghiệp và các báo cáo thống kê liên quan.
- **Tiền xử lý dữ liệu:** Thực hiện các bước làm sạch, chuẩn hóa và xử lý dữ liệu thô để chuẩn bị cho việc lưu trữ và phân tích.
- **Lưu trữ dữ liệu:** Sử dụng các hệ thống lưu trữ phân tán như

HDFS để quản lý dữ liệu lớn.

- **Phân tích dữ liệu:** Thực hiện các phân tích cơ bản như xác định ngành nghề phổ biến, yêu cầu kỹ năng phổ biến, mức lương trung bình, và xu hướng tuyển dụng theo thời gian.
- **Trực quan hóa dữ liệu:** Xây dựng giao diện trực quan với biểu đồ và bảng số liệu giúp người dùng dễ dàng hiểu được kết quả phân tích.

1.3.2 Giới hạn của bài tập lớn

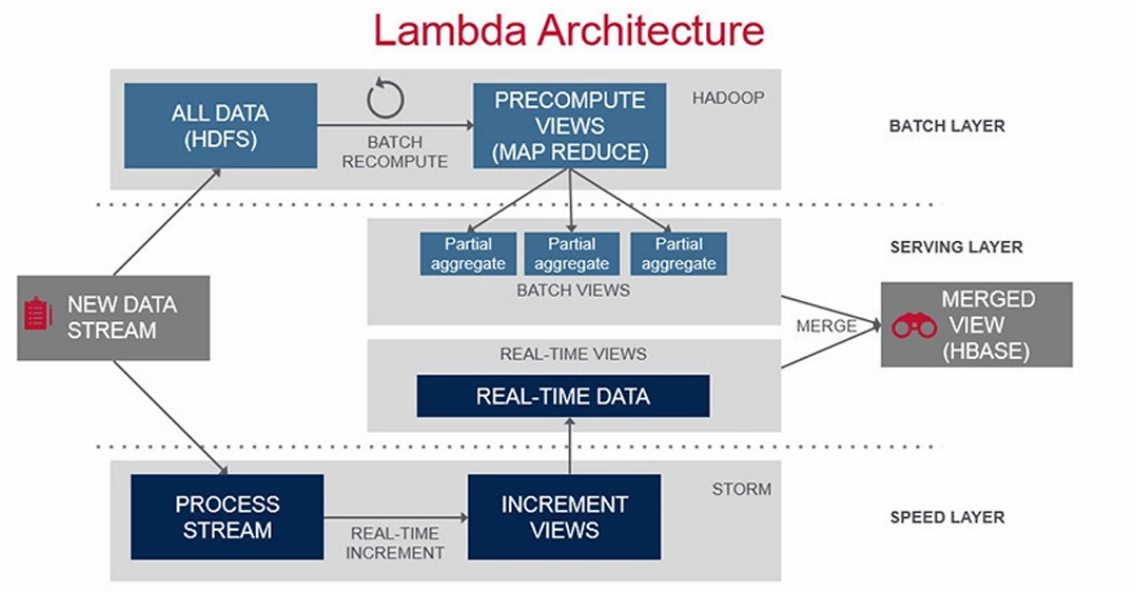
- **Phạm vi dữ liệu:** Chỉ tập trung vào một số ngành nghề hoặc khu vực cụ thể do hạn chế về thời gian và tài nguyên.
- **Độ sâu phân tích:** Các phân tích chỉ dừng lại ở mức độ cơ bản và không đi sâu vào các yếu tố xã hội, kinh tế hoặc chính trị ảnh hưởng đến thị trường lao động.
- **Mô phỏng hệ thống:** Hệ thống được xây dựng nhằm mục đích học thuật, không triển khai thực tế trên quy mô lớn hoặc xử lý dữ liệu theo thời gian thực.
- **Công nghệ sử dụng:** Chỉ sử dụng các công cụ và nền tảng cơ bản như Hadoop, Spark và Python, không tích hợp các công nghệ AI hoặc Machine Learning nâng cao.

Chương 2

Kiến trúc và thiết kế

2.1 Kiến trúc tổng thể

Bài tập lớn sử dụng **Kiến trúc Lambda**. Kiến trúc Lambda là một mô hình kiến trúc được thiết kế để xử lý và phân tích dữ liệu lớn theo thời gian thực và dữ liệu theo lô (batch). Nó được sử dụng rộng rãi trong các hệ thống xử lý dữ liệu lớn, nơi yêu cầu hiệu suất cao và tính linh hoạt trong việc xử lý dữ liệu từ nhiều nguồn khác nhau.



Hình 2.1: Kiến trúc Lambda

2.1.1 Ý tưởng chính

Kiến trúc Lambda chia hệ thống xử lý dữ liệu thành ba lớp chính:

- **Batch Layer (Lớp xử lý theo lô):**
  - Chịu trách nhiệm lưu trữ và xử lý toàn bộ dữ liệu thô.
  - Tạo ra các kết quả xử lý dữ liệu (batch views) mang tính toàn diện và chính xác.
  - Được triển khai trên các công nghệ như Hadoop, Spark hoặc các hệ thống lưu trữ phân tán.
- **Speed Layer (Lớp xử lý nhanh):**
  - Xử lý dữ liệu theo thời gian thực để cung cấp thông tin nhanh chóng, bù đắp cho độ trễ của Batch Layer.
  - Tạo ra các kết quả tạm thời (real-time views) cho dữ liệu mới nhất.
  - Sử dụng các công nghệ như Apache Kafka, Apache Storm, Apache Flink hoặc Spark Streaming.
- **Serving Layer (Lớp phục vụ):**
  - Kết hợp dữ liệu từ Batch Layer và Speed Layer để cung cấp kết quả phân tích cuối cùng.
  - Cung cấp giao diện truy vấn cho người dùng hoặc ứng dụng.
  - Sử dụng các cơ sở dữ liệu như HBase, Cassandra, Elasticsearch hoặc hệ thống lưu trữ khác.

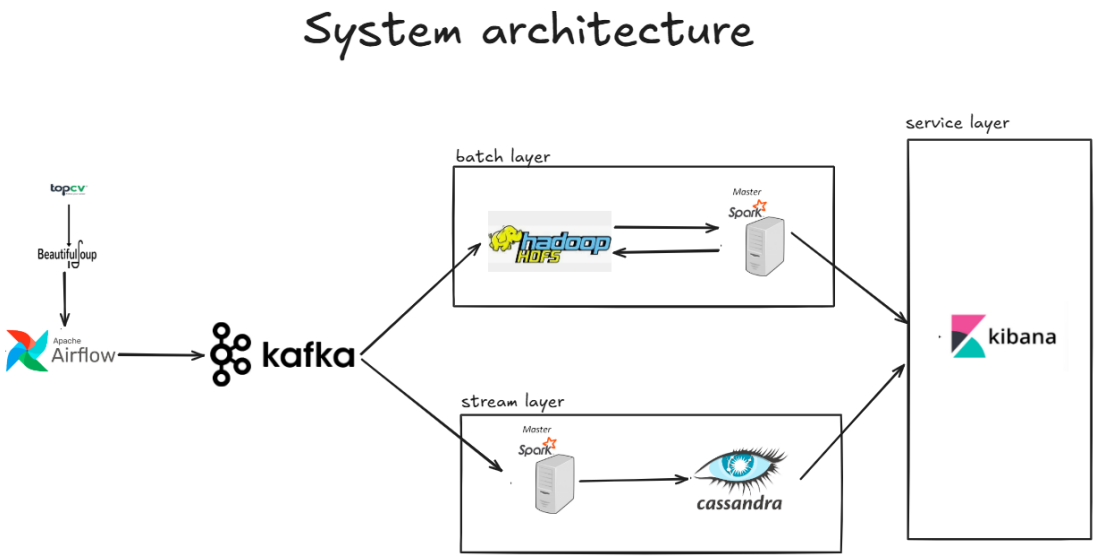
2.1.2 Ưu điểm của kiến trúc Lambda

- **Tính sẵn sàng cao:** Dữ liệu được xử lý đồng thời ở cả hai lớp, đảm bảo tính liên tục ngay cả khi một lớp gặp sự cố.
- **Tính chính xác:** Batch Layer đảm bảo dữ liệu cuối cùng là chính xác và toàn diện.
- **Xử lý theo thời gian thực:** Speed Layer đáp ứng các yêu cầu xử lý nhanh cho dữ liệu mới.
- **Khả năng mở rộng:** Các lớp được triển khai trên các hệ thống phân tán, dễ dàng mở rộng khi dữ liệu tăng lên.

2.1.3 Nhược điểm của kiến trúc Lambda

- **Độ phức tạp cao:** Việc duy trì hai lớp xử lý (batch và speed) đòi hỏi nhiều công sức và tài nguyên.
- **Độ trễ của Batch Layer:** Dữ liệu từ Batch Layer có thể không kịp thời trong một số trường hợp.
- **Tốn kém tài nguyên:** Yêu cầu hạ tầng lớn để hỗ trợ cả hai lớp xử lý.

2.2 Kiến trúc chi tiết



Hình 2.2: Kiến trúc hệ thống

Dữ liệu là thông tin tuyển dụng từ trang web TopCV. Sau quá trình phân tích cú pháp bằng BeautifulSoup ta sẽ thu được dữ liệu được producer gửi tới kafka. Quy trình này là một task được chạy định kì và giám sất bởi Apache Airflow.

2.2.1 Batch Layer

Dữ liệu thô từ kafka được lưu vào HDFS để lưu trữ dài hạn và phục vụ xử lý theo lô. Sau đó Spark cũng được sử dụng để xử lý dữ liệu theo lô từ HDFS, sau đó lưu kết quả trở lại HDFS.

2.2.2 Speed Layer

Spark Streaming nhận dữ liệu theo thời gian thực từ Kafka, dữ liệu sau xử lý được lưu vào Cassandra để phục vụ các ứng dụng yêu cầu truy vấn nhanh.

2.2.3 Serving Layer

Kibana là thành phần chính trong lớp Serving layer, đảm nhận việc lấy dữ liệu đã qua xử lý và lưu trữ để trực quan hóa và hiển thị đến người dùng cuối. Đây là dữ liệu được phục vụ theo cách dễ tiếp cận, thông qua giao diện người dùng trực quan, giúp phân tích và đưa ra quyết định nhanh chóng.

Chương 3

Chi tiết triển khai

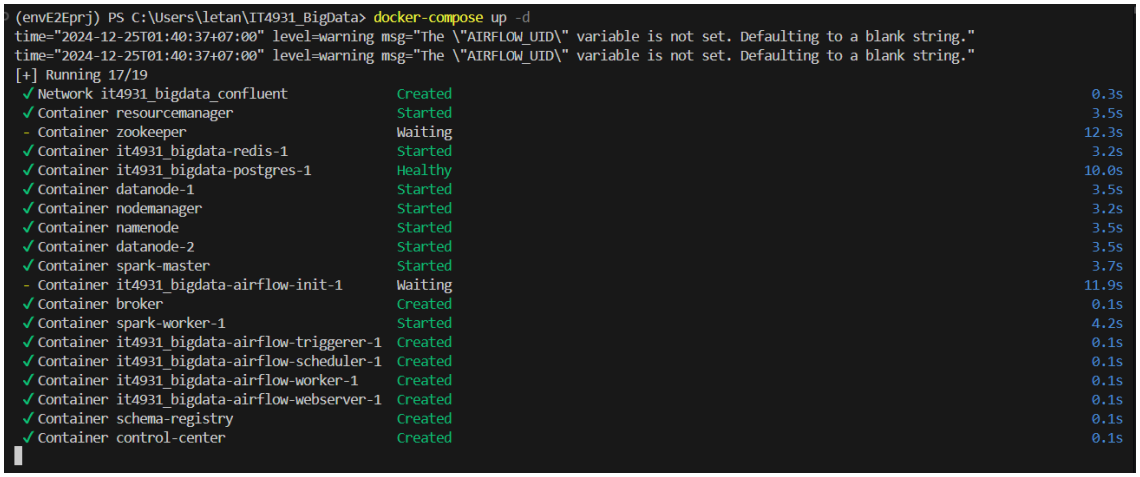
3.1 Docker

Docker được sử dụng để triển khai hệ thống nhằm đảm bảo tính nhất quán, dễ quản lý và triển khai nhanh chóng. Các thành phần tổng hệ thống được đóng gói vào các container độc lập:

- Apache Airflow: Airflow init, Airflow webserver, Airflow Scheduler, Airflow Worker
- Kafka: Zookeeper, Broker, Schema Registry, Control Center
- Spark: Spark master, Spark worker
- HDFS: NameNode, DataNode
- Cassandra
- ...

Tất cả các container trong dự án được kết nối qua mạng **confluent**, cho phép chúng giao tiếp nội bộ mà không cần public các cổng ra ngoài.

Khởi động các dịch vụ chạy lệnh: *docker-compose up -d*

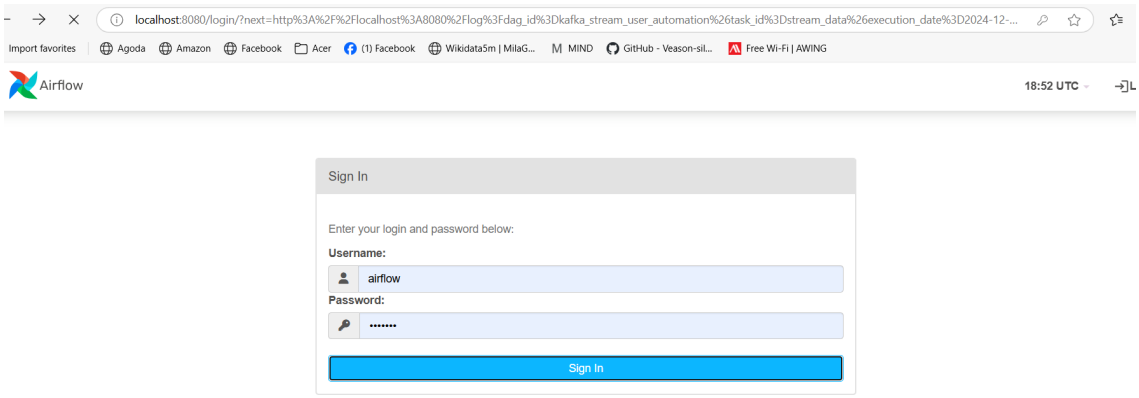


Sau đó, đảm bảo các container hoạt động ổn định, truy cập các web UI:

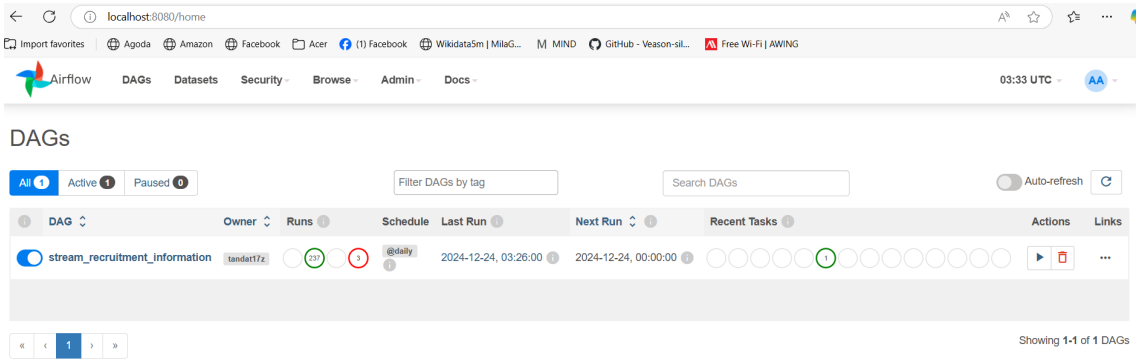
- Airflow Web Server: http://localhost:8080
- Kafka Control Center: http:// localhost:9021
- Spark Master: http://localhost:8083
- HDFS (Hadoop): http://localhost:9870

3.2 Apache Airflow

Apache Airflow là một công cụ mã nguồn mở giúp quản lý và tự động hóa các quy trình công việc phức tạp. Nó cho phép lập kế hoạch, theo dõi và giám sát các tác vụ một cách tự động. Airflow cũng cung cấp giao diện trực quan để giám sát luồng công việc và hỗ trợ retry task tự động nếu xảy ra lỗi.



Hình 3.1: Đăng nhập airflow với username: airflow và password: airflow



Hình 3.2: Giao diện quản lý task của Airflow

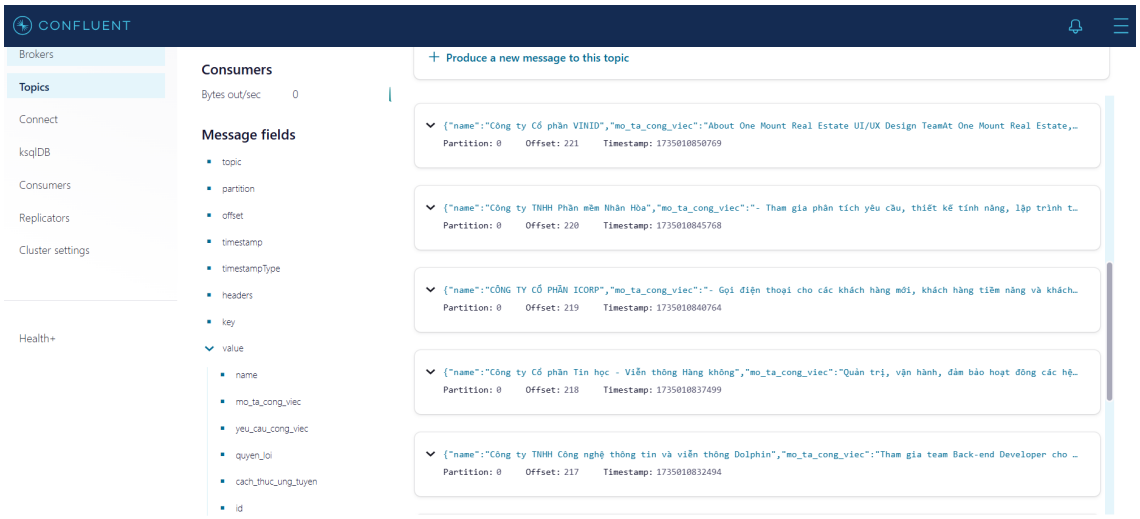
Một DAG có tên *stream_recruitment_information* có nhiệm vụ stream dữ liệu về thông tin tuyển dụng từ các công ty và gửi vào kafka. Tại giao diện Airflow trên, ấn chạy ở Actions. Khi đó, một task *stream_job_data* được tạo thực hiện các công việc sau:

- Lấy danh sách link các trang tuyển dụng
- Lấy danh sách link các job chi tiết.
- Thu thập dữ liệu tuyển dụng sử dụng BeautifulSoup để phân tích cú pháp và trích xuất.
- Chuẩn bị dữ liệu và stream vào kafka.

3.3 Apache Kafka

Apache Kafka là một nền tảng truyền tải dữ liệu phân tán, mạnh mẽ, được thiết kế để xử lý và truyền tải dữ liệu theo thời gian thực. Kafka cho phép xử lý luồng dữ liệu lớn từ nhiều nguồn khác nhau và là công cụ lý tưởng cho các hệ thống yêu cầu khả năng mở rộng và độ trễ thấp.

Là nền tảng truyền tải dữ liệu chính, Kafka chịu trách nhiệm thu thập và phân phối dữ liệu từ nguồn đã crawl ở bước trước đến các thành phần các của hệ thống. Kafka đảm bảo dữ liệu không bị mất nhờ vào khả năng lưu trữ trong các topic. Ngoài ra, hệ thống còn có sự hỗ trợ của Control Center giúp quản lý và giám sát Kafka, cung cấp giao diện trực quan và Schema Registry để quản lý các schema của dữ liệu được gửi qua Kafka để đảm bảo tính đồng nhất và tương thích của dữ liệu.



Hình 3.3: Giao diện quản lý và giám sát kafka của control-center

Trong hệ thống, dữ liệu được producer gửi tới Kafka với 1 topic duy nhất là *recruitment_information*, cấu trúc mỗi message được gửi vào topic là một Json object với các trường như sau:

- id: Là trường làm khóa chính để định danh duy nhất cho mỗi bản ghi
- name: Tên công ty đang tuyển dụng
- mo_ta_cong_viec: Cung cấp thông tin chi tiết về vị trí công việc mà công ty cần tuyển.
- yeu_cau_cong_viec: Liệt kê các tiêu chí cần có của ứng viên (kỹ năng, kinh nghiệm, trình độ, ...)

- quyen_loi: Mô tả những lợi ích mà ứng viên sẽ nhận được khi tham gia vào công ty (lương, thưởng, chế độ phúc lợi, môi trường làm việc, ...)
- cach_thuc_ung_tuyen: Cung cấp thông tin về cách ứng viên có thể nộp đơn ứng tuyển.

3.4 Apache Spark

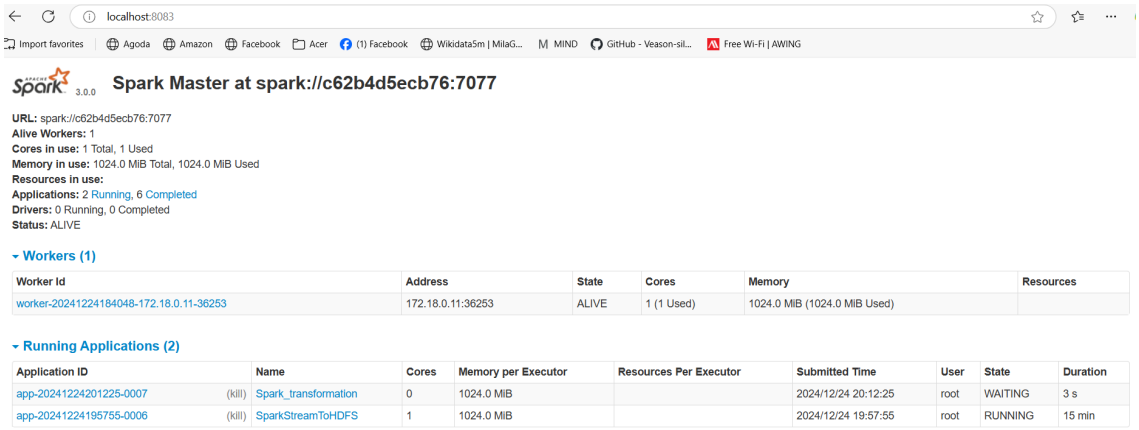
Apache Spark là một nền tảng xử lý dữ liệu phân tán mạnh mẽ, hỗ trợ xử lý dữ liệu lớn theo thời gian thực và theo lô. Spark cung cấp các API cho các tác vụ như phân tích dữ liệu, học máy, và xử lý luồng dữ liệu, với hiệu suất cao nhờ vào việc sử dụng bộ nhớ thay vì lưu trữ đĩa.

Trong hệ thống này, do hạn chế về tài nguyên phần cứng và mục tiêu phục vụ việc chạy thử nghiệm xử lý dữ liệu, Spark được triển khai với:

- 1 spark master: Điều phối các tasks và quản lý phân phối tài nguyên cho spark worker.
- 1 spark worker: Chịu trích nhiệm thực thi các tác vụ.

Hệ thống kết nối trực tiếp Spark với Kafka để đọc dữ liệu theo luồng. Chạy lệnh *docker exec -it spark-master bash* để mở một terminal tương tác trong container spark.

```
(envE2Eprj) PS C:\Users\letan\IT4931_BigData> docker exec -it spark-master bash
bash-5.0#
```

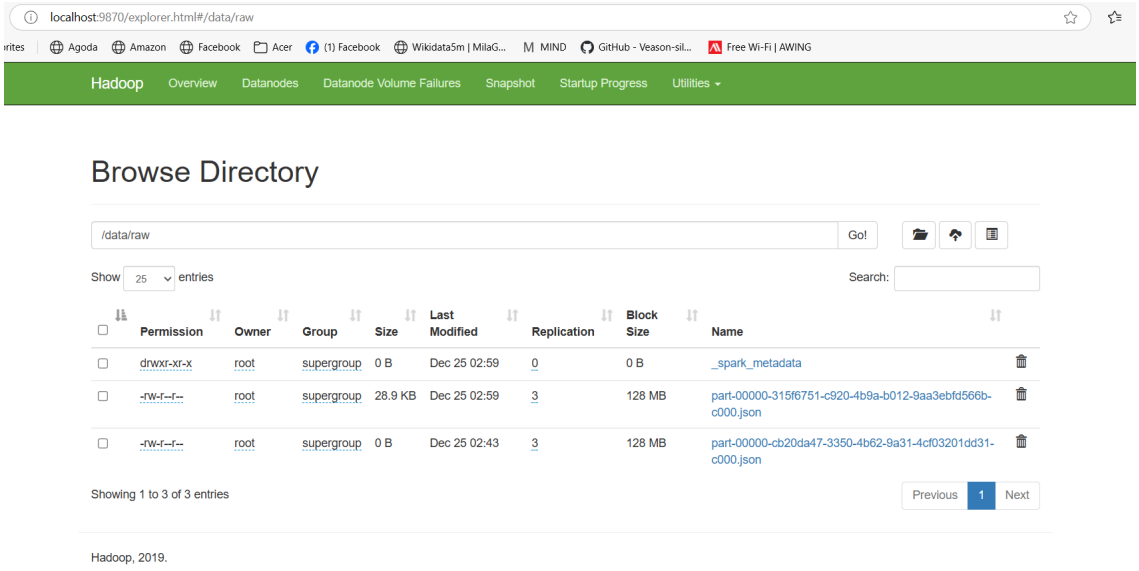


3.4.1 Lưu dữ liệu thô vào HDFS, Batch processing

HDFS (Hadoop Distributed File System) là hệ thống tệp phân tán của Apache Hadoop, được thiết kế để lưu trữ và xử lý dữ liệu lớn. Nó chia nhỏ dữ liệu thành các khối và phân phối chúng trên các máy chủ trong một cụm, đảm bảo tính sẵn sàng và khả năng mở rộng cao. Dữ liệu thô và dữ liệu batch được Hdfs lưu trữ với khả năng phân tán và an toàn cao. Định dạng lưu trữ Json được chọn để tối ưu hóa. Hệ thống triển khai HDFS với 1 Namenode và chỉ 2 Datanode.

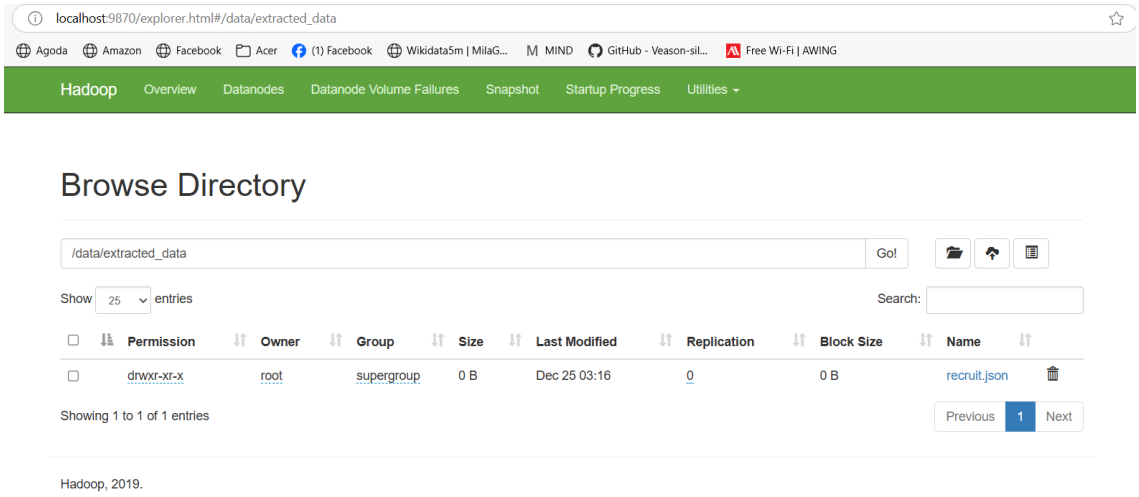
Chạy lệnh sau ở terminal spark-master để gửi ứng dụng Spark tới cluster để thực thi. Thực hiện luồng nhận dữ liệu thô từ kafka và lưu ngay vào HDFS dưới định dạng Json.

```
bash-5.0# spark/bin/spark-submit --master spark://spark-master:7077 --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0
  src/spark_stream_to_hdfs.py
```



Tiếp tục tạo 1 terminal spark-master mới để gửi 1 ứng dụng khác. App này sẽ thực hiện việc xử lý theo lô, đó là đọc dữ liệu thô từ HDFS kia và xử lý rồi lưu lại vào HDFS dữ liệu sau khi được xử lý.

```
(envE2Eprj) PS C:\Users\letan\IT4931_BigData> docker exec -it spark-master bash
bash-5.0# spark/bin/spark-submit --master spark://spark-master:7077 --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0
  src/spark_transformation.py
```



3.4.2 Stream processing

Việc stream dữ liệu từ kafka, biến đổi dữ liệu bằng PySpark và lưu dữ liệu đã xử lý vào cassandra. Khởi động app streaming:

```
bash-5.0# spark/bin/spark-submit --master spark://spark-master:7077 --packages com.datastax.spark:spark-cassandra-connector_2.13:3.0.0,
org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0  src/spark_stream.py
```

Lọc dữ liệu bằng Spark: Dữ liệu cần trích xuất giúp tối ưu hóa khả năng lưu trữ cũng như mang lại tri thức, những góc nhìn có ý nghĩa về dữ liệu hơn đối với người dùng. Từ dữ liệu thô, ta sẽ trích xuất ra thông tin để tạo dataframe với các trường dữ liệu bao gồm:

- company_name: Tên công ty
- frameworks_platforms: Một danh sách tên có frameworks, platforms mà công ty tuyển dụng yêu cầu.
- languages: Danh sách các ngôn ngữ lập trình
- knowledges: Danh sách tên các kiến thức, các kỹ năng mà công ty yêu cầu
- salaries: Danh sách các mức lương mà công ty tuyển dung chi trả.

Các trường thông tin frameworks_platforms, languages, knowledges được trích xuất theo cùng một cách là tìm các xâu trong dữ liệu gốc mà khớp với các xâu được định nghĩa sẵn (gọi là các pattern) tương ứng với mỗi trường. Ví dụ, đối với languages:

```
languages = [' CHAIN ', ' ABAP ', 'Lingo', ' CPL', 'NPL', 'Xtend', ' Flex ', ' Io ', 'Erlang', 'Python', 'MSL', 'SAIL',
 'XPath', 'Lava', ' Clojure', 'Mathematica', 'QPL', 'Oak', 'Objective-C', 'P#', 'css', ' Delphi', ' A+ ', '
 'CoffeeScript', ' SR ', 'ECMAScript', 'Nial', ' Red ', 'Mesa', 'LSL', 'T-SQL', 'E#', ' GAP ', 'Simula', 'L
 'JavaScript', 'PeopleCode', ' UNITY ', ' Blue ', 'Span', 'Lucid', ' BeanShell', ' CSP', 'Scheme', 'Swift',
 ' Scala ', 'Scratch', 'Strand', 'XML', ' SAS', 'PortablE', 'JADE', ' C ', 'Processing', 'Pure', ' BASIC ',
 'ROOP', 'PL/SQL', 'Icon', ' Dart', ' Factor ', 'Java', 'LINC', ' Go ', ' TIE ', ' Cool', 'Kotlin', 'Rust',
 ' Inform ', 'Mary', 'Ruby', 'YQL', 'Pike', ' rc ', ' html ', 'Oz', 'Groovy', 'PowerShell', ' CUDA', 'Hack'
 ' CFEngine', 'C#', 'SPS']
```

Đối với trường salaries thì việc làm sạch dữ liệu sẽ phức tạp hơn. Bởi vì mức lương được biểu diễn dưới nhiều hình thức khác nhau như là 2000$, 20000000VND... Vì vậy hệ thống sẽ đồng nhất lương theo đơn vị triệu VND và thống ke lương theo các khoảng 5 triệu VND. Mức lương trong các đơn vị tuyển dụng sẽ được chia vào các khoảng tương ứng, biểu diễn bằng một mảng các số nguyên là chăn dưới của mỗi khoảng. Ví dụ dưới đây về việc chuyển đổi mức lương:

Mức lương	Mảng quy đổi
8 triệu VNĐ	[5]
26-38 triệu VNĐ	[25,30,35]
2000$	[45]

Apache Cassandra là một cơ sở dữ liệu NoSQL phân tán, được thiết kế để xử lý dữ liệu lớn và yêu cầu tính sẵn sàng cao. Cassandra có khả năng mở rộng ngang và hỗ trợ lưu trữ dữ liệu với độ trễ thấp, rất phù hợp cho các ứng dụng yêu cầu khả năng chịu lỗi, đặc biết là các ứng dụng xử lý dữ liệu thời gian thực.

Trong hệ thống này, Cassandra được sử dụng để phục vụ lưu trữ dữ liệu streaming đã được xử lý, giúp truy xuất nhanh chóng cho các tác vụ phân tích và hiển thị.

```
cassandra@cqlsh> select company_name, framework_platforms  from spark_streams.extracted_recruit;

 company_name                                                | framework_platforms
-------------------------------------------------------------+-----------------------------------------
                                  Công Ty TNHH Luxstay Việt Nam |                                 ['Git']
                                  Công Ty TNHH Giải Pháp Công Nghệ CIT |                                 ['MySQL']
                                  Công ty Cổ phần VINID |                                 ['OSP', 'Figma']
                                  CÔNG TY TNHH MEDIASTEP SOFTWARE VIỆT NAM |                 ['Docker', 'Git', 'Visio']
                                  FPT Smart Cloud (FCI) |                                 ['Git']
                                  Công ty TNHH sản xuất và thương mại Ninh Thanh |                                 ['.NET']
                                  Công ty TNHH sản xuất và thương mại Ninh Thanh |                                 ['.NET']
                                  CÔNG TY TÀI CHÍNH TRÁCH NHIỆM HỮU HẠN MB SHINSEI |                 ['Reactjs', 'Git', 'Redux', 'Bootstrap']
                                  Công ty TNHH Công nghệ thông tin và viễn thông Dolphin |                 ['MySQL', ' Redis', 'Laravel']
                                  CÔNG TY TNHH MEDIASTEP SOFTWARE VIỆT NAM |                 ['Docker', 'Git', 'Visio']
```

Hình 3.4: Kết quả truy vấn từ Cassandra

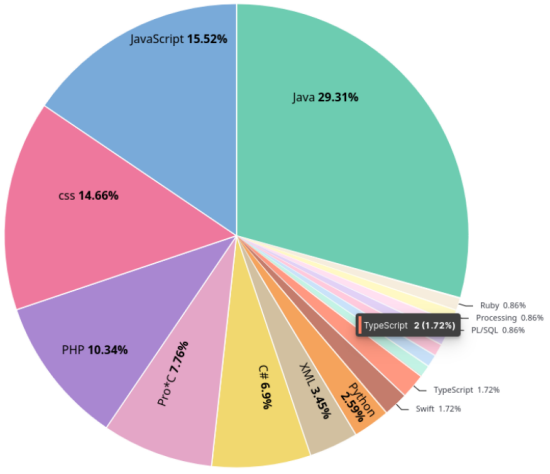
3.5 Kibana

- Là công cụ giao diện người dùng (UI) giúp trực quan hóa dữ liệu và tương tác với Elasticsearch. Kibana cho phép người dùng tạo các bảng điều khiển (dashboards) và báo cáo từ dữ liệu trong Elasticsearch.
- Kibana cho phép người dùng thực hiện tìm kiếm trực tiếp và lọc dữ liệu từ Elasticsearch qua giao diện web dễ sử dụng.

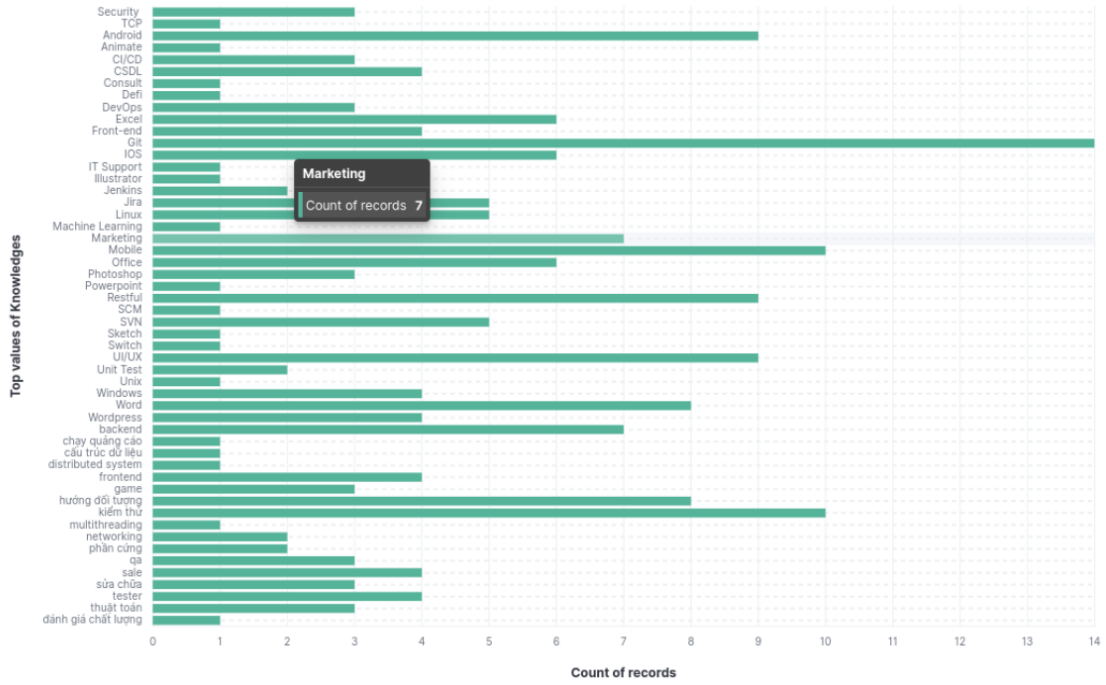
Tạo Dashboard

- Ta có thể tạo visualization trong dashboard. Kibana cho phép kéo thả tiện lợi, nó cho phép chọn các trường thuộc tính trong dữ liệu của mình và đếm số lượng các bộ có cùng giá trị.
- Dữ liệu có thể biểu diễn dạng cột, biểu đồ tròn, cây, vùng,...

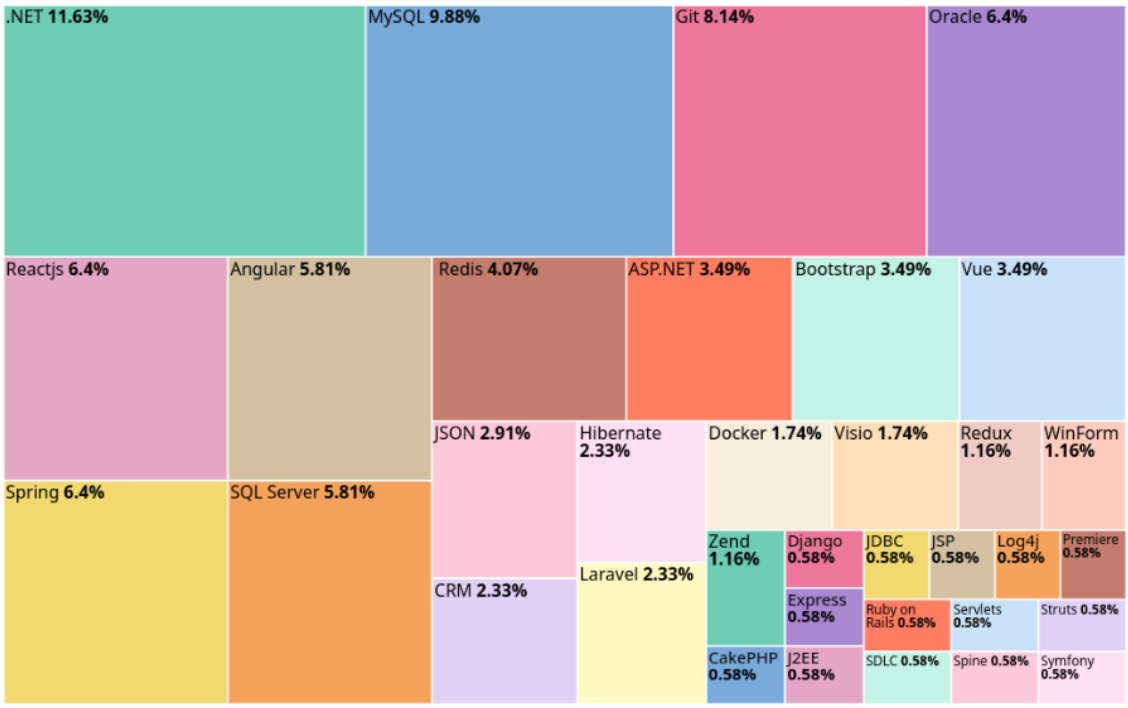
Đây là một số các biểu đồ nhóm vẽ được:



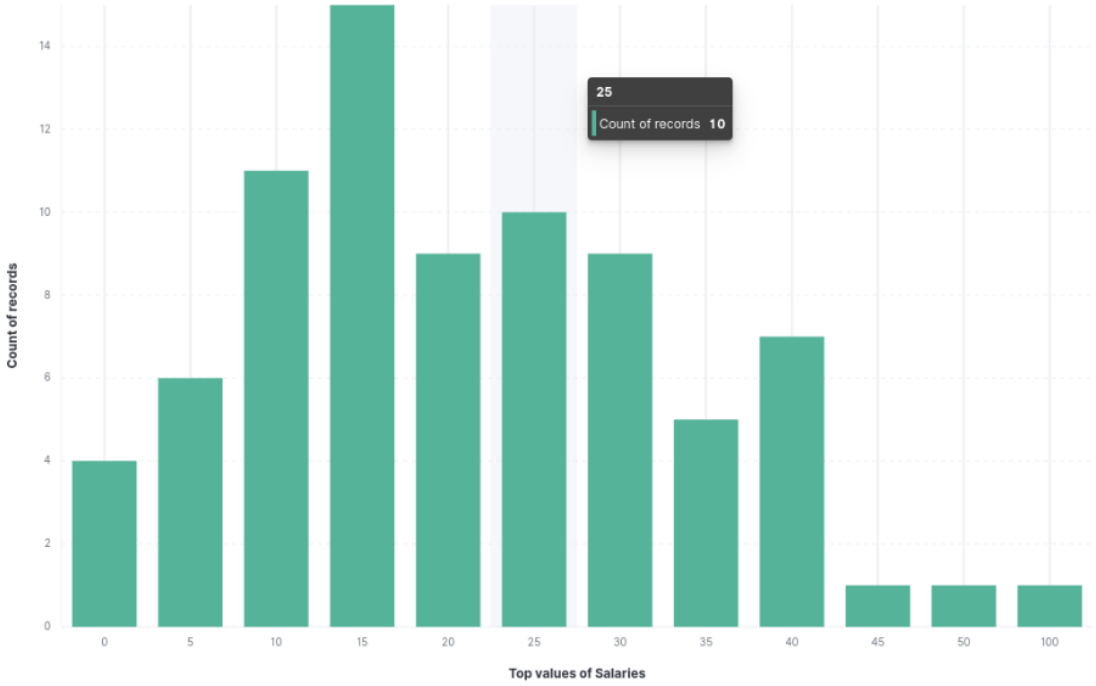
Hình 3.5: Tỉ lệ tuyển dụng dựa trên ngôn ngữ lập trình



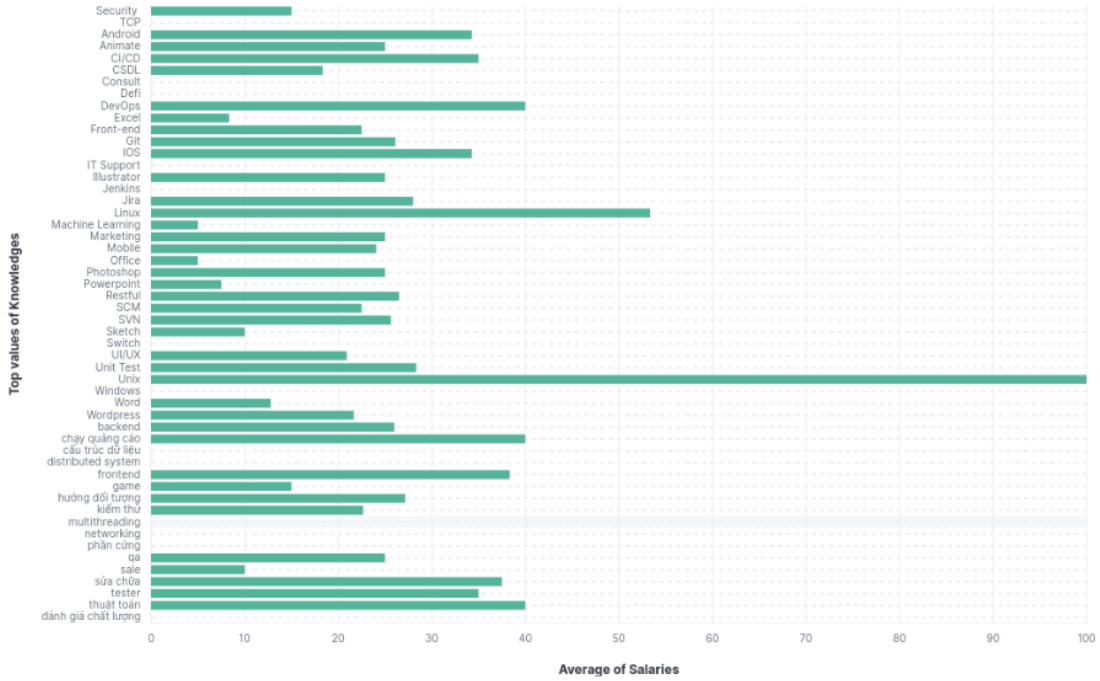
Hình 3.6: Tỉ lệ các vị trí tuyển dụng



Hình 3.7: Tỉ lệ tuyển dụng dựa trên framework



Hình 3.8: Mức lương phổ biến



Hình 3.9: Mức lương theo từng vị trí

Chương 4

Bài học kinh nghiệm

4.1 Kinh nghiệm 1: Xử lý nguồn dữ liệu đa dạng

4.1.1 Vấn đề gặp phải

- Dự án yêu cầu thu thập dữ liệu từ nhiều nguồn và thách thức gặp phải ở sự khác biệt về định dạng dữ liệu. Các trang web thay đổi giao diện, cấu trúc HTML dẫn tới script craling có thể bị lỗi hoặc không lấy đủ dữ liệu.
- Ngoài ra, một số trang web giới hạn số lượng request từ một địa chỉ IP trong khoảng thời gian cụ thể. Hệ thống có thể bị cấm truy cập nếu gửi quá nhiều request liên tục.
- Một số trang có thông tin không đầy đủ (thiếu trường), dữ liệu dư thừa hoặc không cần thiết (do quảng cáo, mã HTML không liên quan) làm cho dữ liệu không đồng nhất.

4.1.2 Các giải pháp đã thử

- Giải pháp để xử lý việc chống spam là thêm khoảng thời gian chờ giữa các lần request, giảm tốc độ gửi request để tránh bị phát hiện hoặc chặn bởi cơ chế chống spam. Đồng thời giúp hệ thống ổn định hơn, tránh làm quá tải.

- Bảo toàn dữ liệu gốc: lưu trữ dữ liệu thô giúp đảm bảo mọi thông tin từ nguồn đều được giữ nguyên, giúp xử lý lại từ dữ liệu này mà không cần crawl lại. Dữ liệu thô được lưu trữ có thể xử lý cho nhiều mục đích khác nhau sau này, nếu cần các yêu cầu phân tích hoặc xử lý thay đổi, có thể tái xử lý mà không cần thu thập lại dữ liệu.
  - Hạn chế của việc lưu dữ liệu thô là tốn lượng lưu trữ vì dữ liệu này sẽ chứa các thông tin dư thừa và không cần thiết. Chất lượng dữ liệu thấp.
  - Tăng độ phức tạp trong quá trình xử lý downstream, tốn nhiều tài nguyên hơn.

4.1.3 Bài học rút ra

- Xây dựng script có khả năng xử lý các thay đổi cấu trúc trang web mà không phải chỉnh sửa toàn bộ mã nguồn, linh hoạt tổng quát hóa hơn.
- Cân bằng giữa tốc độ và chất lượng, cần điều chỉnh tốc độ crawling hợp lý để đảm bảo thu thập đầy đủ dữ liệu mà không bị chặn, gián đoạn.

4.2 Kinh nghiệm 2: Xử lý dữ liệu với Spark

4.2.1 Vấn đề gặp phải

- Bối cảnh: Hệ thống sử dung Spark để xử lý dữ liệu lớn từ HDFS và Kafka. Khi làm việc với dữ liệu streaming, xử lý dữ liệu thời gian thực đòi hỏi phái có một giải pháp có thể nhận và xử lý dữ liệu liên tục.
- Thách thức gặp phải: Hiệu suất xử lý không ổn định, việc quản lý bộ nhớ gặp khó khăn, tăng chi phí và thời gian xử lý.

4.2.2 Các giải pháp đã thử

- Điều chỉnh số lượng executor và cores: Giúp tăng hiệu suất nhưng chưa tối ưu cho khối lượng dữ liệu lớn.

- Xử lý dữ liệu với UDFs: Sử dụng hàm UDF (user defined functions) để thực hiện các phép trích xuất và xử lý dữ liệu từ các cột. Nó tốt khi áp dụng các phép toán phức tạp không có sẵn trong các hàm Spark tích hợp

4.2.3 Bài học rút ra

- Việc xử lý dữ liệu thời gian thực đòi hỏi sự ổn định và hiệu suất cao. Cần đảm bảo rằng các thành phần trong hệ thống (Kafka, Spark, Cassandra) được cấu hình tối ưu để xử lý lượng dữ liệu lớn.
- Cẩn thận với dữ liệu thiếu: Loại bỏ các dòng dữ liệu thiếu trước khi tiếp tục xử lý, tránh lỗi dữ liệu không đầy đủ.

4.3 Kinh nghiệm 3: Stream Processing

4.3.1 Vấn đề gặp phải

- Bối cảnh: Dữ liệu thời gian thực được xử lý qua Kafka và Spark streaming.
- Thách thức gặp phải: Đảm bảo xử lý chính xác (exactly-once), quản lý trạng thái và dữ liệu đến muộn.

4.3.2 Các giải pháp đã thử

- Sử dung checkpointing để quản lý trạng thái
- Sử dụng Kafka offset để đảm bảo exactly-once processing.

4.3.3 Bài học rút ra

- Tầm quan trọng của checkpointing: Giúp hệ thống có thể phục hồi chính xác từ điểm lỗi hoặc tái xử lý các sự kiện bị mất, đảm bảo tính toàn vẹn của dữ liệu
- Cần có chiến lược xử lý các sự kiện đến muộn để đảm bảo tính toàn vẹn cảu dữ liệu. Dữ liệu đến muộn có thể ảnh hưởng tới kết quả cuối cùng, vì vậy việc sử lý chúng một cách hiệu quả là điều cần thiết.

4.4 Kinh nghiệm 4: Data Storage

4.4.1 Vấn đề gặp phải

Dữ liệu từ các nguồn thường không đồng nhất, việc tìm một định dạng lưu trữ chung ban đầu gặp nhiều khó khăn. Một số định dạng phức tạp hoặc khó xử lý làm tăng thời gian phát triển và xử lý.

4.4.2 Các giải pháp đã thử

Sử dụng JSON cho dữ liệu thô ban đầu, vì dễ đọc và dễ kiểm tra. Đồng thời hỗ trợ tốt trên hầu hết các ngôn ngữ lập trình, thư viện và hệ thống xử lý dữ liệu. Hơn nữa, cũng dễ dàng tích hợp giữa các thành phần của hệ thống (ví dụ giữa Kafka, Spark và HDFS).

4.4.3 Bài học rút ra

- JSON là lựa chọn phù hợp cho giai đoạn lưu trữ dữ liệu thô ban đầu, khi dữ liệu chưa được tối ưu nhưng cần đảm bảo tính linh hoạt và dễ sử dụng, giúp đơn giản hóa workflow.
- JSON cung cấp cấu trúc dạng key-value, giúp lưu trữ dữ liệu không đồng nhất từ các nguồn khác nhau mà không càn phải chuẩn hóa ngay lập tức.
- Json cho phép lưu trữ cả dữ liệu có cấu trúc và không có cấu trúc, giúp hệ thống dễ dàng mở rộng khi thêm nguồn dữ liệu mới.

Chương 5

Kết luận và hướng phát triển

5.1 Kết luận

Đề tài "Lưu trữ và phân tích dữ liệu việc làm" đã cung cấp một cái nhìn toàn diện về cách thức thu thập, lưu trữ và phân tích dữ liệu việc làm từ nhiều nguồn khác nhau. Qua quá trình nghiên cứu và triển khai, chúng em đã áp dụng các công nghệ lưu trữ và xử lý dữ liệu lớn như Hadoop, Apache Spark và NoSQL databases để xử lý lượng dữ liệu khổng lồ và thực hiện phân tích các xu hướng việc làm, yêu cầu kỹ năng, và các thông tin thị trường lao động. Việc áp dụng các công cụ này không chỉ giúp cải thiện hiệu quả xử lý mà còn tối ưu hóa việc phân tích dữ liệu theo thời gian thực, giúp người sử dụng có thể đưa ra quyết định chính xác và nhanh chóng.

Bằng cách kết hợp các phương pháp lưu trữ phân tán và phân tích dữ liệu lớn, hệ thống đã có thể cung cấp thông tin chi tiết về thị trường lao động, hỗ trợ các nhà tuyển dụng, ứng viên và các nhà nghiên cứu trong việc hiểu rõ hơn về các xu hướng việc làm hiện tại và tương lai.

5.2 Hướng phát triển

Trong quá trình làm bài tập lớn, mặc dù đã nhận được sự hướng dẫn tận tình của **TS. Trần Việt Trung**, tuy nhiên do kiến thức còn hạn hẹp cũng như kinh nghiệm chưa phong phú, bài tập lớn của chúng em không tránh khỏi những thiếu sót. Do đó, dù hệ thống đã đạt được những mục tiêu ban đầu, vẫn còn nhiều cơ hội để phát triển thêm. Một số hướng phát triển có thể bao gồm:

- **Tăng cường khả năng dự báo (Predictive Analytics):** Sử dụng các mô hình học máy và phân tích dự báo để cung cấp các xu hướng việc làm trong tương lai, giúp các ứng viên và nhà tuyển dụng có cái nhìn rõ hơn về thị trường lao động dài hạn.
- **Mở rộng khả năng tích hợp dữ liệu:** Mở rộng hệ thống để tích hợp thêm các nguồn dữ liệu từ các trang web tuyển dụng, mạng xã hội và các tổ chức giáo dục, nhằm cung cấp cái nhìn toàn diện hơn về thị trường việc làm.
- **Tối ưu hóa hệ thống theo thời gian thực:** Cải thiện lớp Speed Layer để xử lý dữ liệu theo thời gian thực nhanh chóng và chính xác hơn, giúp hệ thống cung cấp thông tin và phân tích kịp thời cho các quyết định nhanh chóng.
- **Phát triển hệ thống gợi ý việc làm:** Xây dựng một hệ thống gợi ý việc làm cho ứng viên dựa trên dữ liệu lịch sử, kỹ năng và sở thích của họ, giúp tối ưu hóa quá trình tìm kiếm việc làm và tuyển dụng.
- **Cải thiện hiệu suất và khả năng mở rộng:** Với sự gia tăng dữ liệu, cần tối ưu hóa hệ thống để đảm bảo khả năng xử lý và lưu trữ hiệu quả, đáp ứng nhu cầu ngày càng cao về dữ liệu lớn và phân tích thời gian thực.

Với các cải tiến này, hi vọng hệ thống sẽ trở thành công cụ mạnh mẽ không chỉ cho việc phân tích thị trường lao động mà còn giúp tối ưu hóa các quyết định trong quá trình tìm kiếm và tuyển dụng việc làm.