

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

**BÀI TẬP LỚN**

MÔN: LƯU TRỮ VÀ XỬ LÝ DỮ LIỆU LỚN

**Phân tích và xử lý dữ liệu tuyển dụng Việt Nam**

Nhóm : 5  
Mã lớp học : 144943  
Giáo viên hướng dẫn : TS. Trần Việt Trung  
Danh sách sinh viên thực hiện:

STT	Họ tên	Mã sinh viên	Email	Lớp
1	Phạm Huy Hoàng	20204653	hoang.ph204653	IT1-04
2	Tổng Việt Dũng	20204534	dung.tv204534	IT1-06
3	Mai Vũ Duy	20204646	duy.mv204646	IT1-04
4	Hồ Sỹ Vinh	20204862	vinh.hs204862	IT2-04
5	Nguyễn Hữu Ý	20200679	y.nh200679	IT1-05

*Hà Nội, tháng 12 năm 2023*

## *Mục lục*

1	Giới thiệu tổng quan đề tài	3
2	Khái quát công nghệ sử dụng	4
2.1	Apache Spark	4
2.2	Docker	6
2.2.1	Luồng phát triển phần mềm với Docker:	6
2.2.2	Kiến trúc Docker	7
2.3	Kafka	8
2.3.1	Kafka streaming	8
2.3.2	Kafka overview	8
2.4	Kubernetes	9
2.4.1	Giới thiệu	9
2.4.2	Các khái niệm cơ bản của Kubernetes	10
2.4.3	Kiến trúc của cụm Kubernetes	10
3	Thiết kế hệ thống	12
3.1	Pipeline của hệ thống	12
3.2	Phần crawl dữ liệu	13
3.3	Phần triển khai Docker	13
3.4	Một số công việc xử lý trong Spark job	14
3.5	Phần lưu dữ liệu và xử lý dữ liệu	15
4	Kết quả	18
5	Một số trải nghiệm và khó khăn gặp phải	20
5.1	Trải nghiệm	20
5.2	Khó khăn	20
6	Kết luận	22

## 1 Giới thiệu tổng quan đề tài

Trong môi trường kinh doanh ngày nay, các trang web tuyển dụng đóng vai trò quan trọng trong việc cung cấp nguồn thông tin về nhu cầu, đặc điểm, yêu cầu của các vị trí việc làm đến với người tìm việc. Đề tài sẽ tập trung vào việc thu thập và phân tích dữ liệu từ các trang tin tuyển dụng lớn như Careerbuilder, Careerlink,... thông qua các phương pháp thống kê và học máy nhằm có cái nhìn tổng quan về xu hướng tuyển dụng, ngành nghề có nhu cầu cao, các kỹ năng hay chứng chỉ được đánh giá cao bởi nhà tuyển dụng. Tuy nhiên để đáp ứng lượng dữ liệu vô cùng lớn, được tạo ra liên tục hiện nay đòi hỏi cần có một hệ thống lưu trữ và xử lý dữ liệu hiệu quả, phân tán, có khả năng mở rộng linh hoạt. Mục tiêu chính của đề tài sẽ là sử dụng các công nghệ xử lý dữ liệu lớn phổ biến hiện nay Kubernetes, Spark, Kafka nhằm xây dựng một hệ thống xử lý dữ liệu tuyển dụng mạnh mẽ nhằm trích xuất các giá trị từ lượng dữ liệu khổng lồ một cách nhanh chóng. Ban đầu nhóm chúng em dự định triển khai hệ thống trên cụm Kubernetes 1 node tuy nhiên trong quá trình cài đặt nhóm gặp khó khăn trong việc kết nối giữa các pod nên hệ thống sẽ chỉ được triển khai trên Docker.

## 2 Khái quát công nghệ sử dụng

### 2.1 Apache Spark

Apache Spark là một công cụ tính toán hợp nhất và tập hợp các thư viện mã nguồn mở được thiết kế để xử lý và phân tích dữ liệu song song trên các cụm máy tính phân tán. Nó được phát triển để đáp ứng nhu cầu ngày càng tăng về xử lý dữ liệu lớn và phức tạp trong các ứng dụng như xử lý dữ liệu thời gian thực, đồ thị, máy học, và trí tuệ nhân tạo.

Spark bao gồm các thư viện cho các tác vụ đa dạng, từ SQL đến xử lý dữ liệu thời gian thực (streaming) và học máy, đồng thời chạy mọi nơi từ laptop đến cụm hàng nghìn máy chủ. Spark được sử dụng để dễ dàng mở rộng quy mô sang xử lý dữ liệu lớn hoặc cực kỳ lớn.

Ưu điểm của Apache Spark

- **Khả năng tương thích rất tốt:** Apache spark là phần mềm mã nguồn mở miễn phí, bởi vậy nó tương thích với mọi hệ điều hành. Đồng thời phần mềm này tích hợp với mọi định dạng tệp và các nguồn dữ liệu do Hadoop hỗ trợ.
- **Khả năng xử lý dữ liệu nhanh, quy mô lớn:** Apache spark có thể xử lý cùng lúc nhiều dữ liệu trên nhiều máy tính khác nhau dựa trên thời gian thực.
- **Hỗ trợ đa ngôn ngữ:** Spark hỗ trợ Java, Python, R, Scala.
- **Điều chỉnh độ trễ linh hoạt:** Phần mềm có thể tạo sẵn các sparkcontext giúp tăng, giảm độ trễ thực thi công việc (theo giây). Người dùng có thể thực hiện cùng lúc nhiều tính toán mà không tạo sự chênh lệch quá lớn.

Kiến trúc của Apache Spark

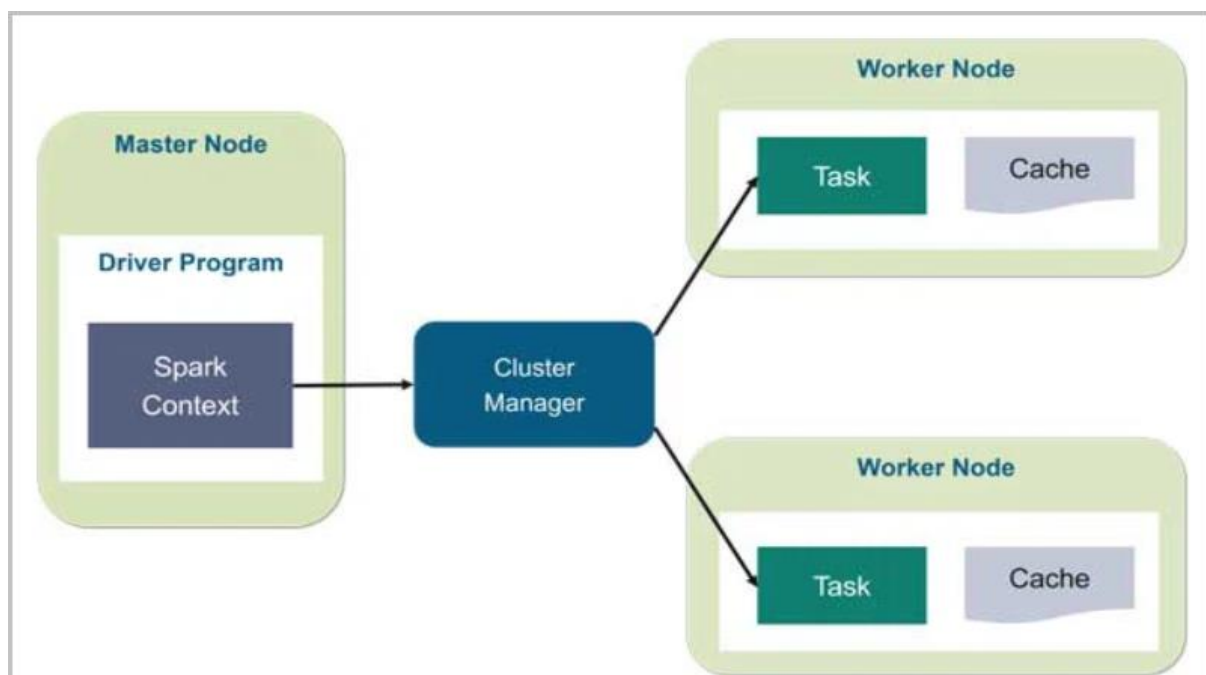
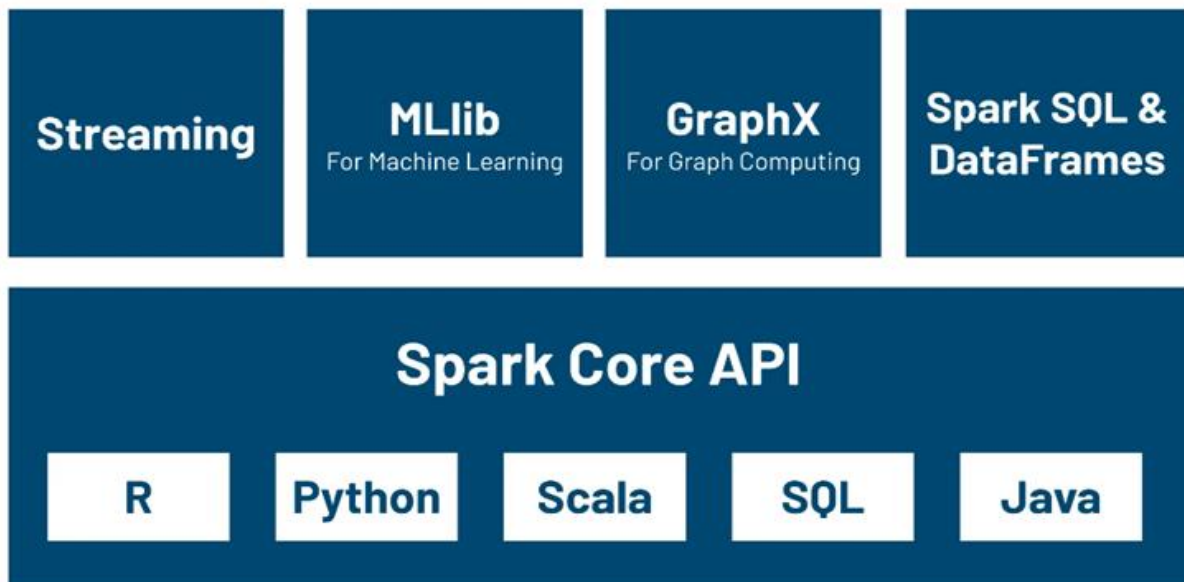


Figure 1 Kiến trúc Apache Spark

Apache Spark được cấu thành từ hai phần chính là trình thực thi (executors) và trình điều khiển (driver). Trong đó, trình điều khiển có nhiệm vụ giúp chuyển đổi mã của người dùng thành các tác vụ (tasks), sau đó phân phối chúng trên các nút xử lý (worker nodes). Trình thực thi sẽ thực hiện các tác vụ này và chạy trên các nút xử lý mà trình điều khiển giao cho nó.

Các thành phần của Apache Spark:



*Figure 2 Thành phần của Apache Spark*

- **Spark Core** là thành phần cốt lõi của Apache Spark, đóng vai trò quan trọng trong việc xử lý dữ liệu phân tán (DDP). Nó chịu trách nhiệm về quản lý bộ nhớ, lập lịch tác vụ, và khôi phục lỗi, làm nền tảng cho các thành phần khác của Spark.
- **Spark SQL**, một thành phần khác, mang đến khả năng truy vấn dữ liệu bằng ngôn ngữ SQL trên các tập dữ liệu lớn. Điều này không chỉ giúp tích hợp với cơ sở dữ liệu quan hệ như MySQL, Oracle, hoặc PostgreSQL mà còn tận dụng tính năng phân tán của Spark để xử lý dữ liệu hiệu quả. Spark SQL không chỉ giới hạn ở việc truy vấn dữ liệu mà còn kết hợp tốt với các công cụ ETL và Spark Streaming.
- **Spark Streaming** là một module cho phép xử lý dữ liệu trực tiếp trong thời gian thực (real-time). Spark Streaming cung cấp một cơ chế xử lý dữ liệu liên tục (stream processing) bằng cách chia nhỏ dữ liệu thành một chuỗi các microbatch nhỏ hơn và xử lý chúng thông qua API Apache Spark và nó còn tích hợp với các module khác của Apache Spark như Spark SQL và MLlib.
- **Spark MLlib (Machine Learning Library)** Spark MLlib là một thư viện học máy phân tán trên Spark, sử dụng kiến trúc phân tán bộ nhớ. Spark MLlib đã được chuyên trang công nghệ benchmark đánh giá là có tốc độ nhanh hơn gấp 9 lần so với phiên bản chạy trên Hadoop.
- **GraphX** là một nền tảng xử lý đồ thị trên Spark, cung cấp các API để thực hiện các tính toán trên đồ thị bằng cách sử dụng Pregel API.

## 2.2 Docker

Trước kia, ở công đoạn triển khai phần mềm trên phần cứng, nhà phát triển phải đảm các thư viện mà chương trình cần được tải trên hệ điều hành của máy tính đó. Công đoạn này thường mất nhiều thời gian và tiềm ẩn nhiều rủi ro. Một trong số rủi ro là có thể hệ điều hành đó không hỗ trợ thư viện cần hoặc thư viện của chương trình này không tương thích với thư viện của chương trình kia. Đó là một trong những vấn đề đặt ra yêu cầu phải sử dụng Docker. Docker là một nền tảng phục vụ việc phát triển, đóng gói, chạy ứng dụng. Docker giúp các nhà phát triển không phải bận tâm về cấu hình phần cứng để chạy chương trình nữa để tập trung vào phát triển phần mềm. Cụ thể Docker cung cấp cho nhà phát triển khả năng đóng gói chương trình và những thư viện cần thành một container, container này có thể chạy trên bất kỳ hệ điều hành nào có cài đặt Docker và sử dụng hiệu quả tài nguyên của máy chủ hơn là hệ điều hành ảo (vd: VMWare).

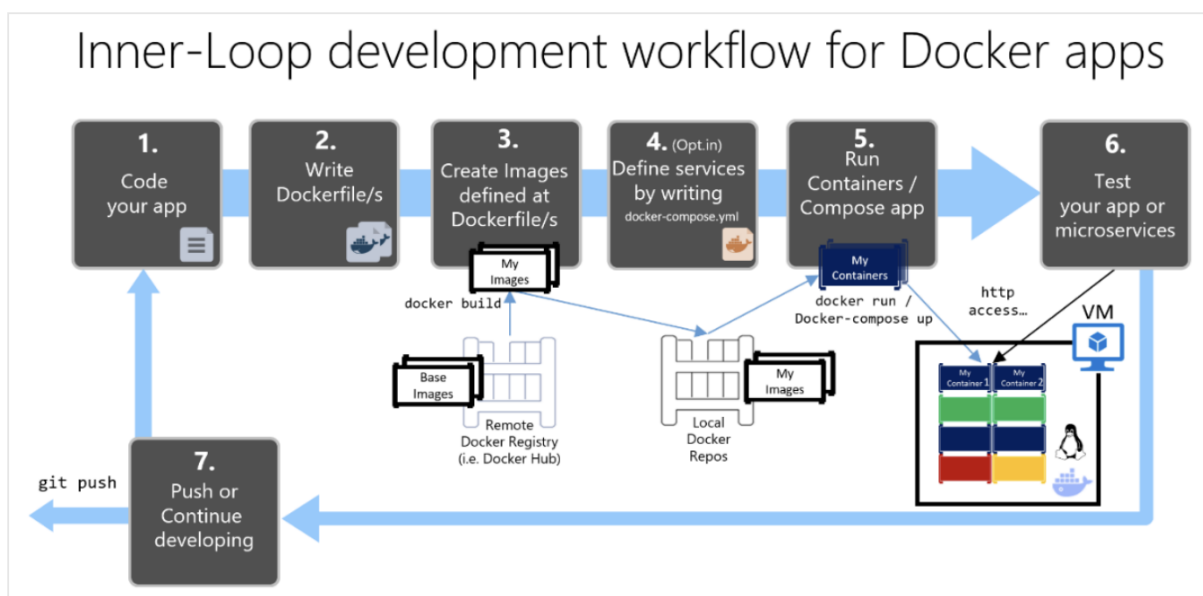


Figure 5-1. Step-by-step workflow for developing Docker containerized apps

### 2.2.1 Luồng phát triển phần mềm với Docker:

Sau khi phát triển phần mềm(bước 1), người lập trình sẽ viết một file Dockerfile. Dockerfile là một file chứa các chỉ dẫn giúp docker đóng gói chương trình thành một file **Image**. Nhờ vào file này, người lập trình có thể lựa chọn hệ điều hành muốn sử dụng trong Image, đưa mã nguồn chương trình vào Image, tại các thư viện và thiết lập biến môi trường cần thiết. Ở bước 3, lập trình viên build file Image từ file Dockerfile ở bước 2, kết quả bước này là file Image có thể triển khai thành các container trên bất kỳ phần cứng nào có cài đặt docker. Trước khi triển khai, ở bước 4, lập trình viên sẽ viết một file Docker Compose để định nghĩa service. Một service gồm nhiều chương trình, các chương trình có thể tương tác với nhau hoặc với chương trình bên ngoài nên cần file Compose này để cấu hình mạng, các chương trình sẽ chạy. Sau khi viết file Compose, lập trình viên có thể triển khai trên phần cứng (bước 5) và test service (bước 6). Cuối cùng, nếu cần sửa đổi mã nguồn, người lập trình có thể làm và thực hiện lại các bước từ 2 tới 6.

## 2.2.2 Kiến trúc Docker

Docker sử dụng kiến trúc client-server. **Docker client** sẽ gửi yêu cầu đến **docker daemon**. Docker daemon chạy trên docker server sẽ chịu trách nhiệm xây dựng Docker Image, triển khai các container đó từ Docker Image hoặc triển khai các cụm container từ Docker Compose

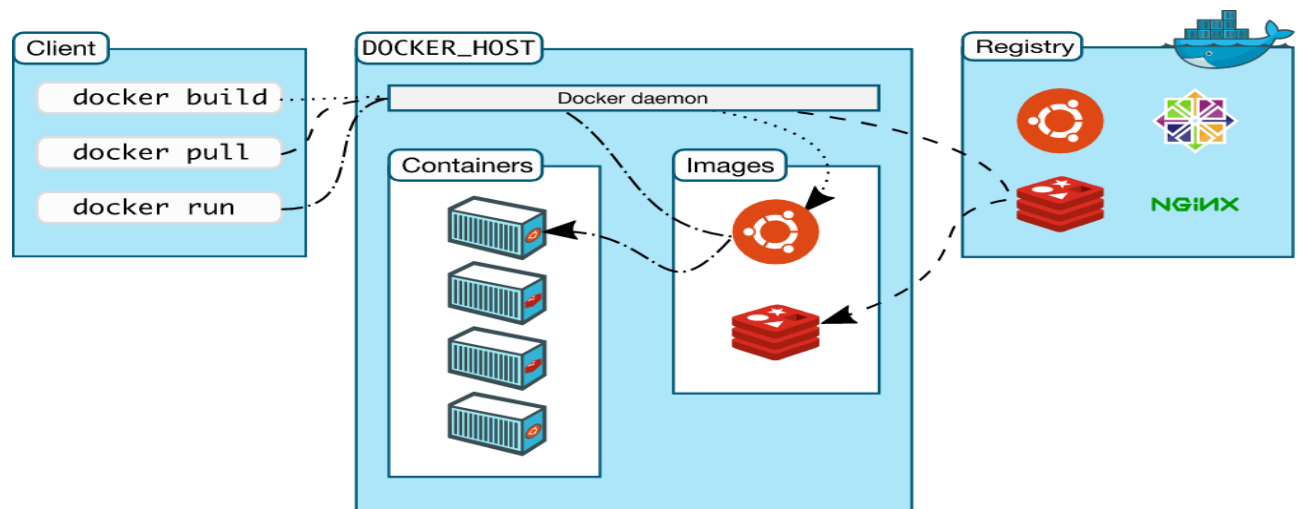


Figure 3 Kiến trúc của Docker

**Docker Desktop:** Docker Desktop là một ứng dụng dễ cài đặt, cho phép xây dựng và chia sẻ các ứng dụng và vi dịch vụ được chứa trong container. Docker Desktop bao gồm daemon Docker (dockerd), Docker client, Docker Compose, Docker Content Trust, Kubernetes và Credential Helper.

**Docker registries :** Lưu trữ các Docker Image, Docker Hub là 1 Docker registries công khai mà ai cũng có thể sử dụng, theo mặc định thì Docker được cấu hình để tìm kiếm image trên Docker Hub.

**Docker objects :** Khi sử dụng Docker, bạn đang tạo và sử dụng images, container , network, và các đối tượng khác.

- **Images :** là một khuôn mẫu để tạo một container. Thường thì image sẽ dựa trên 1 image có sẵn với những tùy chỉnh thêm. Bạn cũng có thể tự build một image riêng cho mình hoặc sử dụng những Image được chia sẻ từ cộng đồng Docker Hub. Một image sẽ được build dựa trên những chỉ dẫn của Dockerfile.
- **Container :** là một instance của một image. Người dùng có thể create, start, stop, move or delete container dựa trên Docker API hoặc Docker CLI.

## 2.3 Kafka

Apache Kafka là một hệ thống phân tán message – distributed messaging system. Nó được phát triển theo mô hình public/subscribe, bên public dữ liệu được gọi là producer và bên subscribe dữ liệu theo các topic sẽ được gọi là consumer

### 2.3.1 Kafka streaming

Kafka là một kho dữ liệu phân tán được tối ưu hóa cho việc xử lý dữ liệu trong thời gian thực. Nếu là một platform streaming trực tuyến sẽ cần phải xử lý một cách liên tục, tuần tự tăng dần với độ trễ thấp và thông lượng cao. Kafka cung cấp cho người dùng 3 chức năng chính như sau:

1. Publish và subscribe các stream của record
2. Lưu trữ các stream của record theo thứ tự các record được tạo, với cơ chế chống chịu lỗi (dữ liệu không bị mất mát)
3. Khả năng xử lý stream của record theo thời gian thực

### 2.3.2 Kafka overview

- **Kafka producer:** Kafka lưu và phân loại message theo từng topic, sử dụng producer để publish message vào các topic, dữ liệu sẽ được gửi đến partition thuộc về topic của nó.
- **Kafka consumer:** Kafka sử dụng consumer để subscribe vào các topic, nhiều consumer có thể subscribe cùng 1 topic.
- **Kafka topic:** Dữ liệu truyền trong kafka theo từng topic, dữ liệu có những đặc điểm khác nhau thì sẽ tạo ra những topic khác nhau tùy thuộc vào bài toán.
- **Kafka partition:** Một topic có thể phân ra thành 1 hay nhiều partition, các partition được phân tán ra các node, giúp cho consumer có thể đọc song song nhiều partition của 1 topic cùng 1 lúc.

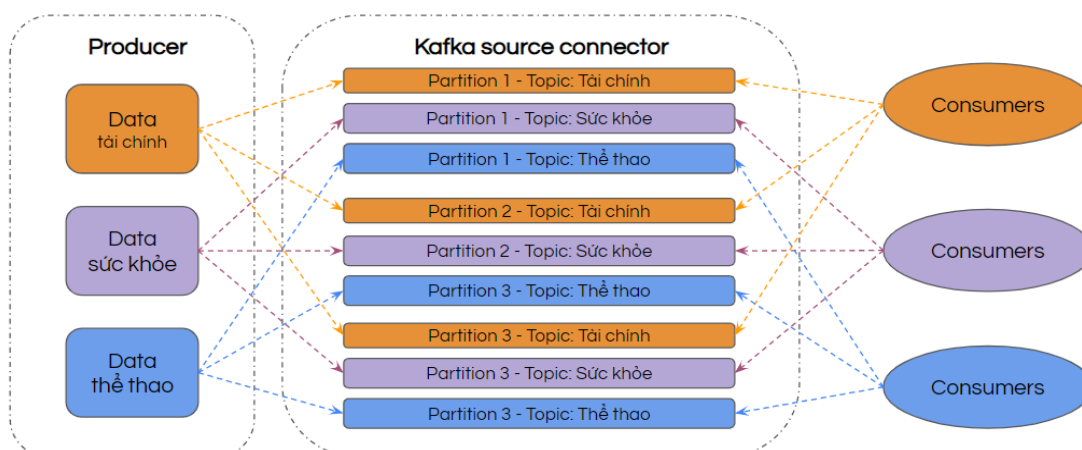


Figure 4 Thành phần của Kafka



## 2.4 Kubernetes

### 2.4.1 Giới thiệu

Kubernetes (k8s) là hệ thống mã nguồn mở mạnh mẽ được dùng để quản lý các ứng dụng đóng gói và các service, giúp thuận lợi trong việc cấu hình và tự động hóa việc triển khai ứng dụng. K8s được phát triển bởi Google và được sử dụng để quản lý số lượng container lên đến hàng tỷ.

Chức năng:

- **Tự phục hồi (Self-healing)** : Kubernetes khởi động lại các containers bị lỗi, thay thế các container, xóa các container không phản hồi lại cấu hình health check do người dùng xác định và không cho các client biết đến chúng cho đến khi chúng sẵn sàng hoạt động.
- **Cân bằng tải (Load balancing)** : Nếu lượng traffic truy cập đến một container cao, Kubernetes có thể cân bằng tải và phân phối lưu lượng mạng (network traffic) để việc triển khai được ổn định.
- **Điều chỉnh bộ nhớ (Storage orchestration)** : Kubernetes cho phép tự động mount một hệ thống lưu trữ mà bạn chọn, như local storages, public cloud providers, v.v.
- **Đóng gói tự động** : Người dùng cung cấp cho Kubernetes một cluster gồm các node mà nó có thể sử dụng để chạy các tác vụ được đóng gói (containerized task). Người dùng cho Kubernetes biết mỗi container cần bao nhiêu CPU và bộ nhớ (RAM). Kubernetes có thể điều phối các container đến các node để tận dụng tốt nhất các resource.
- **Tự động rollouts và rollback** : Người dùng có thể mô tả trạng thái mong muốn cho các container được triển khai dùng Kubernetes và nó có thể thay đổi trạng thái thực tế sang trạng thái mong muốn với tần suất được kiểm soát.

### 2.4.2 Các khái niệm cơ bản của Kubernetes

- **Pod**: Là khái niệm cơ bản và quan trọng nhất trên K8s. Bản thân Pod có thể chứa 1 hoặc nhiều container. Pod là nơi ứng dụng được chạy trong đó. Pod là các tiến trình nằm trên worker node. Bản thân Pod có tài nguyên riêng về file system, cpu, ram, volumes, địa chỉ network, ...

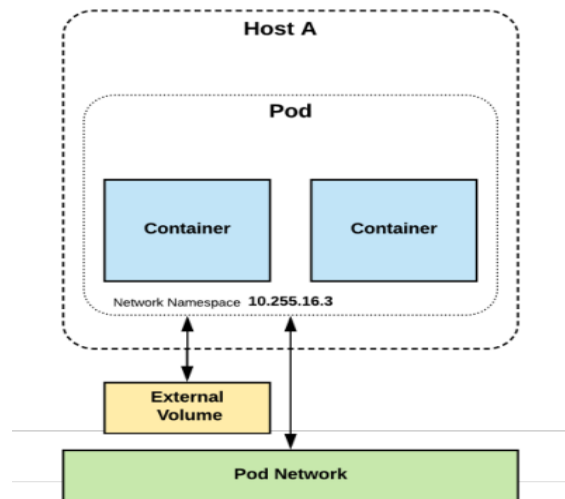


Figure 5 Kiến trúc thành phần Pod trong Kubernetes

- **Service**: Là phần network của K8s giúp các Pod gọi nhau ổn định hơn hoặc để Load balancing giữa nhiều bản sao của Pod và có thể dùng để dẫn traffic từ người dùng vào ứng dụng (Pod) giúp người dùng có thể sử dụng được ứng dụng

### 2.4.3 Kiến trúc của cụm Kubernetes

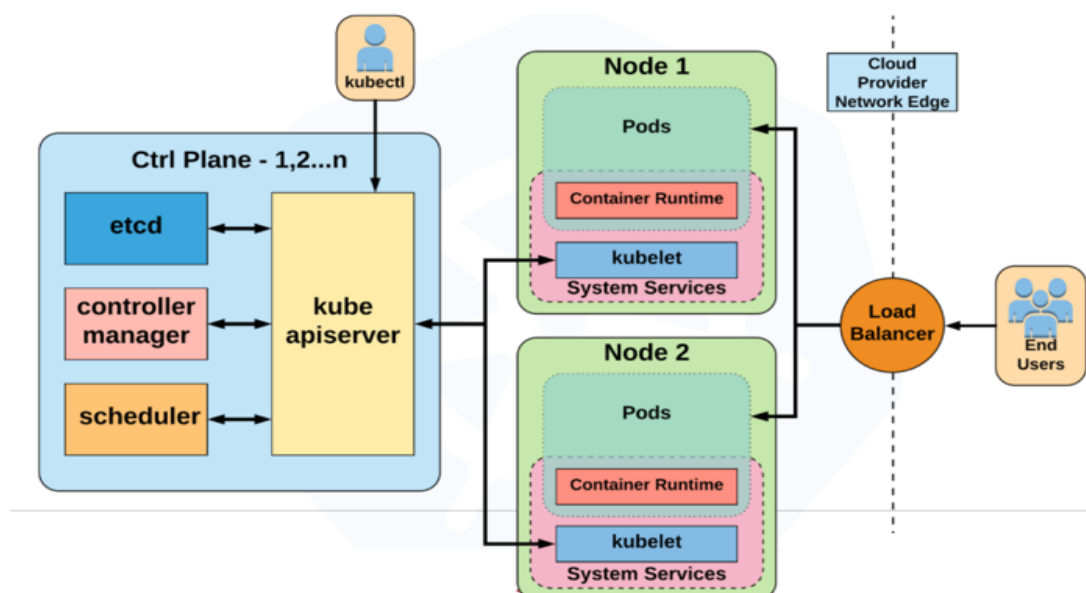


Figure 6 Kiến trúc cụm Kubernetes

Kiến trúc Kubernetes cơ bản tồn tại ở hai phần: control plane và các node. Mỗi node có thể là máy vật lý hoặc máy ảo và là môi trường Linux của riêng nó. Mỗi node cũng chạy các pods, bao gồm các container.

- **Control plane** : điều khiển quản lý và điều phối toàn bộ cụm Kubernetes. Nó bao gồm một số thành phần làm việc cùng nhau để duy trì trạng thái mong muốn của hệ thống. Các thành phần của control plane:
  - **kube-apiserver** : Đóng vai trò là điểm quản lý trung tâm và hiển thị API Kubernetes cho máy khách để liên lạc theo cụm.
  - **etcd** : Kho lưu key-value phân tán lưu trữ dữ liệu và trạng thái cấu hình của cụm
  - **kube-controller-manager** : Bao gồm các bộ điều khiển khác nhau chịu trách nhiệm duy trì trạng thái cụm mong muốn, xử lý các tác vụ như mở rộng quy mô, sao chép và quản lý nút
  - **kube-scheduler** : Xác định các nút nào sẽ được chỉ định nhóm dựa trên tính khả dụng của tài nguyên và các chính sách lập lịch khác
- **Node** : Mỗi node của cụm k8s sẽ có 1 thành phần Node nhằm duy trì các pod đang chạy và cung cấp runtime environment Kubernetes. Các thành phần của Node bao gồm :
  - **kubelet**: Đóng vai trò là tác nhân nút chịu trách nhiệm quản lý vòng đời của mọi pod trên máy chủ của nó.
  - **kube-proxy**: Quản lý các quy tắc mạng trên mỗi nút. Thực hiện chuyển tiếp kết nối hoặc cân bằng tải cho dịch vụ cụm Kubernetes.
  - **Container runtime**: thành phần cơ bản hỗ trợ Kubernetes chạy các container một cách hiệu quả. Nó chịu trách nhiệm quản lý việc thực thi và vòng đời của các container trong môi trường Kubernetes.

### 3 Thiết kế hệ thống

#### 3.1 Pipeline của hệ thống

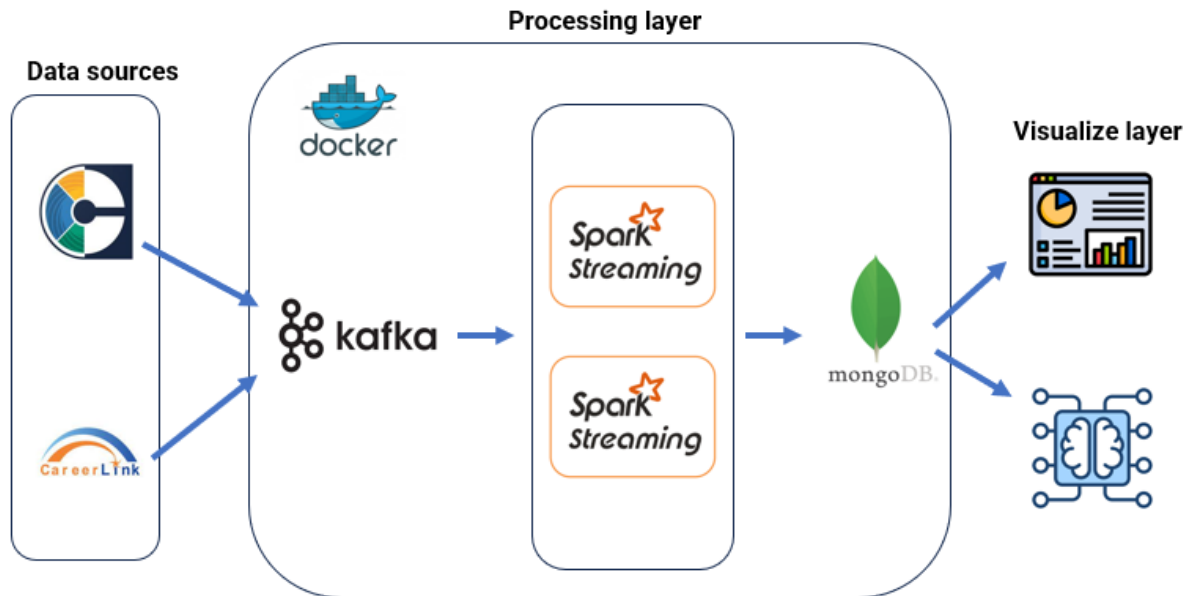


Figure 7 Pipeline của hệ thống

Data sẽ được crawl từ hai nguồn Careerlink và Careerbuilder sử dụng Scrapy chạy. Scrapy sẽ đóng vai trò là Kafka producer, đổ dữ liệu vào Kafka broker với 2 topic như trên. Cụm Spark được xây dựng bao gồm 1 master node và 2 worker node. Spark jobs sau khi được gửi lên cụm sẽ đóng vai trò là Kafka consumer đọc dữ liệu từ Kafka, thực hiện các bước xử lý và làm sạch và lưu trữ vào MongoDB. Từ MongoDB có thể thực hiện việc visualize dữ liệu sử dụng Streamlit. Cụm Spark, Kafka và MongoDB sẽ được triển khai dưới dạng container trong môi trường Docker.

Các thành phần của hệ thống:

- **Kafka** : 2 broker được quản lý bởi 2 zookeeper nhằm mô phỏng môi trường cụm Kafka nhiều nút.
- **Spark**: Cụm Spark bao gồm 1 master và 2 workers
- **Mongodb**: để đơn giản nhóm chúng em sẽ cho data sau khi xử lý lưu trữ vào 1 container mongodb duy nhất.

Dự định ban đầu nhóm chúng em muốn triển khai hệ thống trên cụm Kubernetes của Docker tuy nhiên chúng em gặp khó khăn trong việc kết nối giữa các pod nên sẽ chỉ sử dụng Docker cho hệ thống.

### 3.2 Phần crawl dữ liệu

Sử dụng 2 spider Scrapy crawl dữ liệu từ 2 trang web Careerlink và Careerbuilder. Spider sẽ được khởi động sau khi submit job lên cụm Spark.

### 3.3 Phần triển khai Docker

STT	Tên container	Image	Ports	Ghi chú
1	spark-master-1	bitnami/spark3:5	4040:4040 6060:6060 7077:7077 8080:8080	. Cổng 8080: spark web UI
2	spark-worker-1-1	bitnami/spark:3.5	8081:8081	
3	spark-worker-2-2	bitnami/spark:3.5	8082:8082	
4	zookeeper-1	bitnami/zookeeper:3.5	22181:2181	
5	zookeeper-2	bitnami/zookeeper:3.5	32181:2181	
6	kafka-1	bitnami/kafka:3.5	29092:29092	
7	kafka-2	bitnami/kafka:3.5	39092:39092	
8	mymongoddb	mongo	27017:27017	

1. **spark-master-1**: container đóng vai trò Spark master quản lý các container Spark worker.
2. **spark-worker-1** và **spark-worker-2** : 2 container đóng vai trò worker trong cụm Spark.
3. **zookeeper-1** và **zookeeper-2**: 2 container đóng vai trò quản lý các container Kafka.
4. **kafka-1** và **kafka-2**: 2 container đóng vai trò các Kafka broker.
5. **mymongoddb**: container đóng vai trò lưu trữ dữ liệu từ Spark

### 3.4 Một số công việc xử lý trong Spark job

#### 1. Lấy ra số năm kinh nghiệm

Từ trường “job\_experience\_required” tạo ra 2 trường mới là số năm kinh nghiệm yêu cầu tối thiểu và số năm kinh nghiệm yêu cầu tối đa. Nếu không có thông tin thì 2 trường sẽ này để trống. Ví dụ trường “job\_experience\_required” có giá trị là “2-7 năm” sẽ sinh ra 2 trường mới là “job-yoe-min” có giá trị là 2 và trường “job-yoe-max” có giá trị là 7.

#### 2. Lấy ra mức lương tối thiểu, tối đa

Từ trường “salary” tạo ra 2 trường mới là mức lương min và mức lương max. Nếu không có thông tin thì 2 trường này để trống. Đơn vị lương là “VND”. Ví dụ trường “salary” có giá trị là “Lương 8 Tr – 13 Tr VND” sẽ sinh ra 2 trường mới là “salary\_min” : 8000000 và “salary\_max” : 13000000

#### 3. Điều chỉnh lại giá trị trường “job\_address”

VD:

- “Cầu Giấy, Hà Nội” -> “HaNoi”
- “Hồ Chí Minh” -> “HoChiMinh”

#### 4. Điều chỉnh lại giá trị trường “industry”

VD:

- “Bán hàng/ kinh doanh” -> “Advertising and marketing”
- “Bán lẻ/ bán sỉ” -> “Retail and wholesale”

#### 5. Điều chỉnh lại giá trị trường “employment\_type”

VD:

- “Bán thời gian” -> “part-time”
- “Nhân viên” -> “full-time”

#### 6. Thay đổi định dạng

Chuyển định dạng trường “job\_listed” ( ngày đăng tin ) và “deadline” ( ngày hết hạn ) về định dạng date.

### 3.5 Phần lưu dữ liệu và xử lý dữ liệu

- **Khởi động các containers:**


































Name	Image	Status	CPU (%)	Port(s)	Last sta	Actions
 <a href="#">vietnam-jobs-analysis</a>		Running (8/8)	4.59%		3 minute	 
 <a href="#">spark-worker-2-1</a> f1c49cbcd354 	<a href="#">docker.io/bitnami/spark:3</a>	Running	0.15%	<a href="#">8082:8082</a> 	3 minute	 
 <a href="#">spark-worker-1-1</a> 5d394185e28d 	<a href="#">docker.io/bitnami/spark:3</a>	Running	0.22%	<a href="#">8081:8081</a> 	3 minute	 
 <a href="#">kafka-2</a> ce5403c53dd5 	<a href="#">bitnami/kafka:3.5</a>	Running	1.32%	<a href="#">39092:39092</a> 	3 minute	 
 <a href="#">kafka-1</a> db7540e28bff 	<a href="#">bitnami/kafka:3.5</a>	Running	1.59%	<a href="#">29092:29092</a> 	3 minute	 
 <a href="#">spark-master-1</a> 434514f87fcd 	<a href="#">docker.io/bitnami/spark:3</a>	Running	0.19%	<a href="#">4040:4040</a>  <a href="#">Show all ports (4)</a>	3 minute	 
 <a href="#">mymongodb</a> mymongo		Running	0.70%	<a href="#">27017:27017</a> 	3 minute	 

Figure 8 Các container đang chạy

- **Truy cập localhost:8080:** để xem trạng thái của cụm Spark. Cụm spark sau khi được khởi tạo sẽ bao gồm 1 master node và 2 worker node :

 **Spark Master at spark://434514f87fcd:7077**

URL: spark://434514f87fcd:7077  
Alive Workers: 2  
Cores in use: 8 Total, 0 Used  
Memory in use: 8.0 GiB Total, 0.0 B Used  
Resources in use:  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

▼ Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20240104065030-172.20.0.9-45123	172.20.0.9:45123	ALIVE	4 (0 Used)	4.0 GiB (0.0 B Used)
worker-20240104065031-172.20.0.8-37657	172.20.0.8:37657	ALIVE	4 (0 Used)	4.0 GiB (0.0 B Used)

▼ Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State
----------------	------	-------	---------------------	------------------------	----------------	------	-------

▼ Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State
----------------	------	-------	---------------------	------------------------	----------------	------	-------

Figure 9 Spark web UI

- **Copy các file Spark thực thi vào 1 worker node**

```
docker cp vietnam-jobs-analysis/spark/Careerbuilder vietnam-jobs-analysis-spark-worker-1-1:opt/bitnami/spark
```

```
docker cp vietnam-jobs-analysis/spark/Careerlink vietnam-jobs-analysis-spark-worker-2-1:opt/bitnami/spark
```

- **Tiến hành submit job lên cụm Spark**

VD: submit job xử lý data careerbuilder

```
docker exec -it vietnam-jobs-analysis-spark-worker-1-1 /bin/bash spark-submit --master spark://spark-master:7077 --conf spark.jars.packages=org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0,org.apache.kafka:kafka-clients:3.5.0,org.mongodb.spark:mongo-spark-connector_2.12:3.0.2 --conf spark.jars.ivy=/tmp/binami/pkg/cache --num-executors 2 --driver-memory 512m --executor-memory 512m --executor-cores 2 Careerbuilder/CareerbuilderMain.py
```

Sau khi submit job thành công spark web UI sẽ hiển thị job đang chạy:

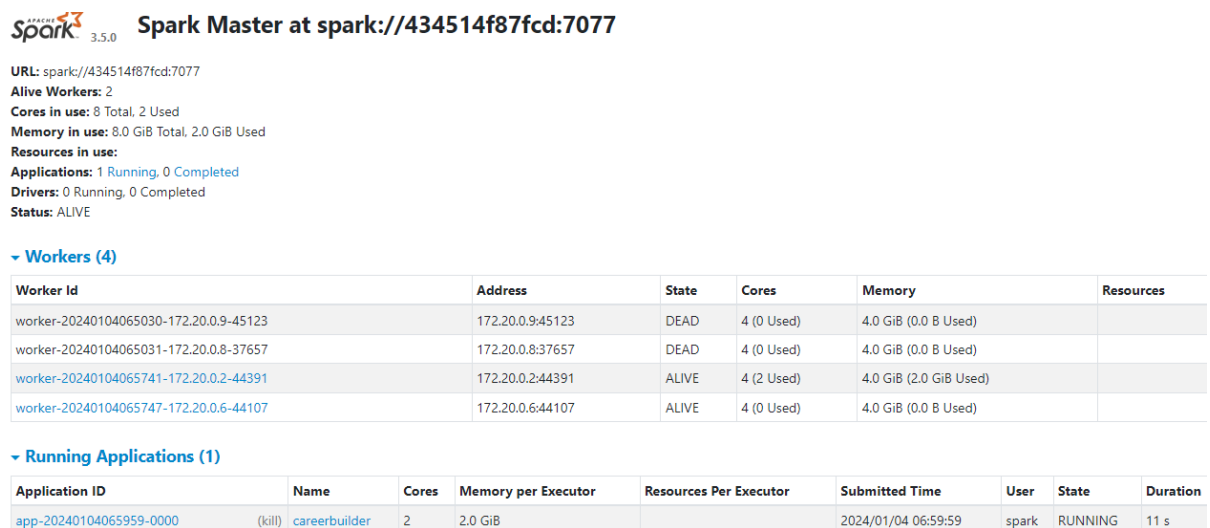


Figure 10 Spark web UI sau khi submit job



## - Chạy scrapy crawl dữ liệu

Dữ liệu kafka nhận được từ scrapy với topic careerbuilder

```
C:\Users\Admin>docker exec -it kafka-1 kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic careerbuilder
--from-beginning

{"job_id": "job-item-358F5623", "job_title": "CVC Quản Trị Rủi Ro Sản Phẩm - Hà Nội", "job_listed": "04-01-2024", "job_d
eadline": "Hạn nộp: 02-02-2024", "salary": "Lương: Cạnh tranh", "company_name": "Ngân Hàng TMCP Việt Nam Thịnh Vượng - V
PBANK", "job_address": "Hà Nội", "job_experience_required": "2 - 3
Năm", "employment_type": "Nhân viên chính thức", "job_function": "Nhân viên", "industries": "Ngân hàng, Kế toán / Kiế
m toán, Tài chính / Đầu tư", "welfare": "Chế độ bảo hiểm, Du lịch, Đồng phục, Chế độ thưởng, Chăm sóc sức khỏe, Đào tạo,
Tăng lương, Nghỉ phép năm", "job_description": "Receiving, reviewing and submitting product proposals to the Pro
duct Committee or the delegated approval authority, Developing, reviewing and revising (if necessary) VPBank's Pr
oduct Development, Approval and Management Policy & Process, Product Committee Bylaws to ensure compliance with internal
regulations, operational needs and international best practices., Supporting Business units and Supporting units
in implementing the Product Development, Approval and Management Policy & Process, Product Committee Bylaws,
Coordinating the implementation of periodic or adhoc product review programs as per BOM's requirements, Preparoing q
uarterly reports on Product Committee performance, Performing other ad hoc tasks assigned by the CRO or the Produ
ct Committee's Chairman, Preparing monthly reports on the risk profile of bankwide products (Risk trigger reports
), Supporting the coordination with related units to periodically develop and review product risk assessment meth
odology and product risk indicators", "job_requirement": "- University degree or higher level in Economics, Financ
e, Banking, Risk Management or other related major., - 2-3 experience working in the fields of Finance, Banking, R
isk Management. Prefer candidates who are working in Operational Risk/Credit Risk Department or Product Development Depa
rtment at other banks. - Willingness to learn and work with international requirements and standards. - Good
```

Figure 11 Dữ liệu bên trong Kafka sau khi crawl

## - Dữ liệu đã được xử lý Mongoddb nhận được

```
{
  _id: ObjectId('658be874cf682c2379a37c85'),
  job_id: '358F47B2',
  job_title: 'Nhân Viên Thu Mua',
  job_listed: ISODate('2023-12-27T00:00:00.000Z'),
  job_deadline: ISODate('2024-01-27T00:00:00.000Z'),
  salary: 'Lương: 11 Tr - 15 Tr VND',
  company_name: 'Atad Steel Structure Corporation',
  job_address: 'HoChiMinh',
  job_experience_required: '2 - 3 Năm',
  employment_type: 'full-time',
  welfare: 'Chế độ bảo hiểm, Du lịch, Đồng phục, Chế độ thưởng, Chăm sóc sức khỏe, Đào tạo, Nghỉ phép năm',
  job_description: '',
  job_requirement: '',
  salary_min: '11000000',
  salary_max: '15000000',
  job_yoe_min: '2',
  job_yoe_max: '3',
  job_level: 'Nhân viên',
  industry: 'Manufacturing',
  ingested_at: ISODate('2023-12-27T00:00:00.000Z'),
  updated_at: ISODate('2023-12-27T00:00:00.000Z')
}
```

Figure 12 Dữ liệu sau khi xử lý bên trong MongoDB

## 4 Kết quả

Các biểu đồ dữ liệu biểu diễn trên Streamlit lấy từ dữ liệu thu được từ MongoDB

### Job Distribution Across Industries

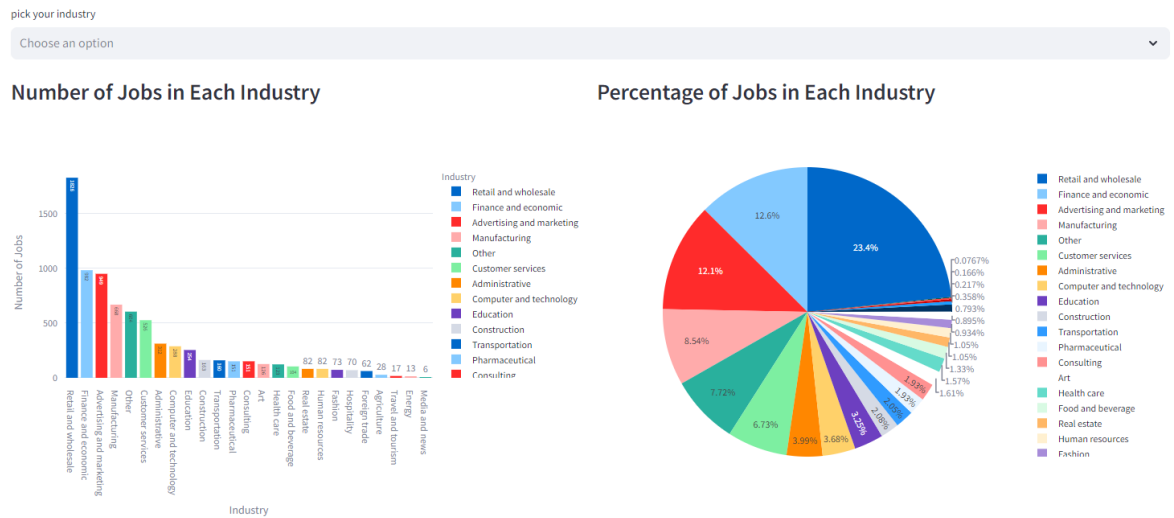


Figure 12 Biểu đồ thống kê về các ngành công nghiệp từ dữ liệu tuyển dụng



Figure 13 Biểu đồ phân bố về mức lương

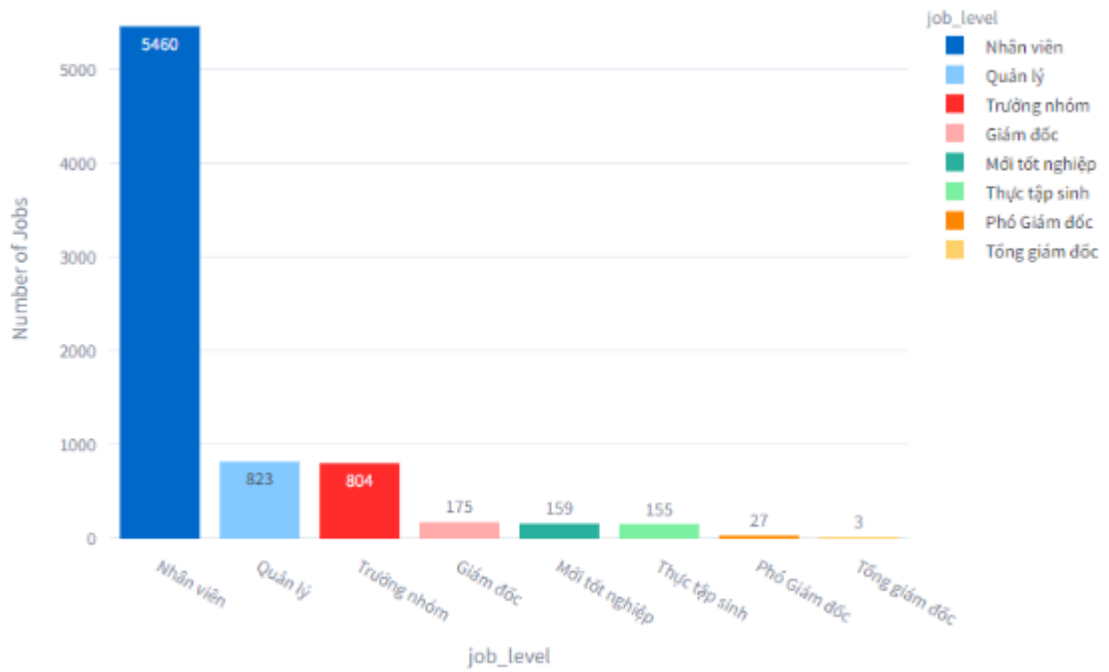


Figure 14 Biểu đồ thống kê số lượng job theo cấp bậc

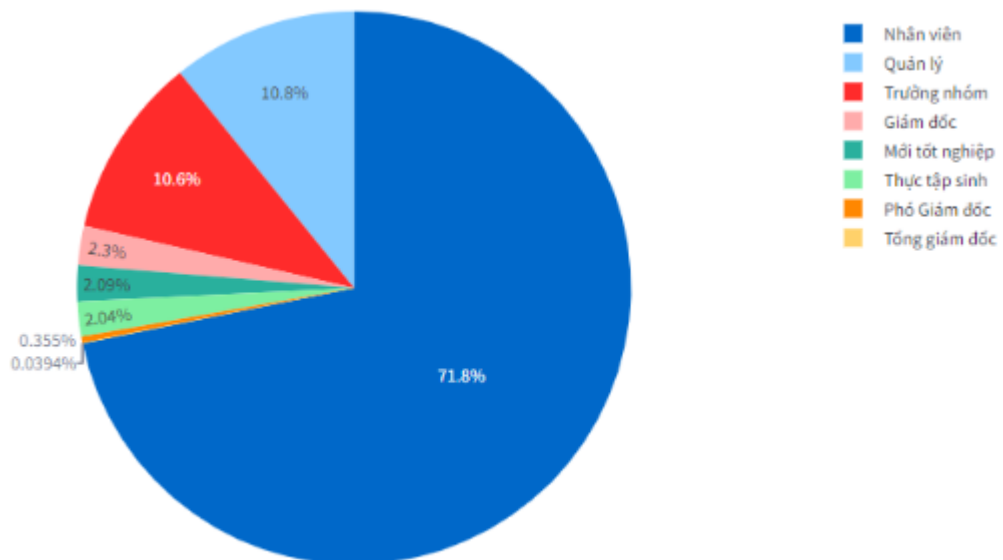


Figure 15 Biểu đồ thống kê về cấp bậc

## 5 Một số trải nghiệm và khó khăn gặp phải








### 5.1 Trải nghiệm

- Lập trình đa luồng khi thu thập dữ liệu
- Triển khai cụm Spark mô phỏng môi trường phân tán

### 5.2 Khó khăn

Khi triển khai hệ thống lên cụm Kubernetes nhóm em gặp một số khó khăn:

- Xây dựng cụm Spark trên K8s sử dụng Helm và khởi tạo Kafka pod bằng cách triển khai các file zookeeper.yaml, kafka.yaml (<https://github.com/Noxshine/vietnam-jobs-analysis>)

Name	Image	Status
 <a href="#">k8s_POD_kafka-broker-c678646b4-zppg9_default_95d4098fbde4d59f7e</a>	<a href="#">registry.k8s.io/pause:3.9</a>	Running
 <a href="#">k8s_kafka-broker_kafka-broker-c678646b4-zppg9_default_7d5cd45d33d0</a>	<a href="#">sha256:a692873757c06a</a>	Running
 <a href="#">k8s_POD_spark-release-worker-0_default_1c1479ef-8315-667710e7fcf7</a>	<a href="#">registry.k8s.io/pause:3.9</a>	Running
 <a href="#">k8s_POD_spark-release-master-0_default_784cf21b-8edd-0e767a567c74</a>	<a href="#">registry.k8s.io/pause:3.9</a>	Running
 <a href="#">k8s_spark-worker_spark-release-worker-0_default_1c14790796703b3725</a>	<a href="#">sha256:f265dec0108355c</a>	Running
 <a href="#">k8s_spark-master_spark-release-master-0_default_784cf2f2fe9364bdbf</a>	<a href="#">sha256:f265dec0108355c</a>	Running
 <a href="#">k8s_POD_spark-release-worker-1_default_83e4cbe4-3fd1-</a>	<a href="#">registry.k8s.io/pause:3.9</a>	Running

Các service sau khi khởi tạo các pod:


```
(venv) PS C:\Users\Admin\Desktop\vietnam-jobs-analysis> kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kafka-service	ClusterIP	10.104.146.219	<none>	9092/TCP	11m
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	13m
spark-release-headless	ClusterIP	None	<none>	<none>	9m36s
spark-release-master-svc	ClusterIP	10.102.50.102	<none>	7077/TCP, 80/TCP	9m36s
zookeeper-service	NodePort	10.110.7.124	<none>	2181:30181/TCP	12m

Các pod sau khi khởi tạo:

```
(venv) PS C:\Users\Admin\Desktop\vietnam-jobs-analysis> kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
kafka-broker-c678646b4-zppg9       1/1     Running   0           13m
spark-release-master-0              1/1     Running   0           11m
spark-release-worker-0              1/1     Running   0           11m
spark-release-worker-1              1/1     Running   0           10m
zookeeper-7d46888797-gw4dr         1/1     Running   0           14m
```

Cụm Spark đã được dựng thành công:

 **Spark Master at spark://10.1.1.253:7077**

URL: spark://10.1.1.253:7077

Alive Workers: 2

Cores in use: 8 Total, 0 Used

Memory in use: 16.5 GiB Total, 0.0 B Used

Resources in use:

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20240104164008-10.1.1.254-7079	10.1.1.254:7079	ALIVE	4 (0 Used)	8.2 GiB (0.0 B Used)
worker-20240104164009-10.1.1.255-7079	10.1.1.255:7079	ALIVE	4 (0 Used)	8.2 GiB (0.0 B Used)

Khi submit Spark job gặp lỗi :

Spark job không thể khởi tạo KafkaClient do không thể nhận biết được Kafka broker đang hoạt động.

```
24/01/04 17:18:07 INFO NetworkClient: [AdminClient clientId=adminclient-1] Node -1 disconnected.
24/01/04 17:18:07 WARN NetworkClient: [AdminClient clientId=adminclient-1] Connection to node -1 (kafka-broker/127.0.0.1:9092)
could not be established. Broker may not be available.
24/01/04 17:18:08 INFO NetworkClient: [AdminClient clientId=adminclient-1] Node -1 disconnected.
24/01/04 17:18:08 WARN NetworkClient: [AdminClient clientId=adminclient-1] Connection to node -1 (kafka-broker/127.0.0.1:9092)
could not be established. Broker may not be available.
```

## 6 Kết luận và hướng phát triển

### 6.1 Kết luận

- Sau quá trình tìm hiểu, nghiên cứu và thực hiện đồ án, nhóm em đã hoàn thành được các mục tiêu đề ra bao gồm việc tìm hiểu các kiến thức về công nghệ được học và áp dụng vào xây dựng hệ thống phân tích dữ liệu thời gian thực dữ liệu tuyển dụng.
- Đồ án đã tạo điều kiện cho nhóm tìm hiểu và thực hành các công nghệ được sử dụng hiện nay, mang tính thực tiễn cao, giúp chúng em có thêm kiến thức về môn học và có thể áp dụng thực tiễn.
- Tuy nhiên dự án không thành công trong việc triển khai trên môi trường Kubernetes.

### 6.2 Hướng phát triển

- Cố gắng xây dựng hệ thống sử dụng Kubernetes trên 1 node, sau đó phát triển thành nhiều node với lượng data lớn hơn và áp dụng các kỹ thuật xử lý và phân tích phức tạp hơn như sử dụng Học máy,...
- Tiếp tục trau dồi kiến thức, phát triển bản thân để nâng cao trình độ, kinh nghiệm để có thể xây dựng và đưa ra những giải pháp, sản phẩm tốt hơn trong tương lai.