

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**TRÌNH TẤN ĐẠT
TẠ ĐỨC BẢO**

**KHÓA LUẬN TỐT NGHIỆP
PHÁT TRIỂN HỆ THỐNG PHÂN TÁN MICROSERVICES CHO
ỨNG DỤNG CHIA SẺ VIDEO**
**DEVELOPMENT OF A MICROSERVICES DISTRIBUTED SYSTEM
FOR VIDEO SHARING APPLICATION**

CỬ NHÂN NGÀNH MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

TP. HỒ CHÍ MINH, 2025

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**



**TRỊNH TÂN ĐẠT - 21520714
TẠ ĐỨC BẢO – 21520623**

**KHÓA LUẬN TỐT NGHIỆP
PHÁT TRIỂN HỆ THỐNG PHÂN TÁN MICROSERVICES CHO
ỨNG DỤNG CHIA SẺ VIDEO
DEVELOPMENT OF A MICROSERVICES DISTRIBUTED SYSTEM
FOR VIDEO SHARING APPLICATION**

CỬ NHÂN NGÀNH MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

GIẢNG VIÊN HƯỚNG DẪN

Ths. Lê Anh Tuấn

TP. HỒ CHÍ MINH, 2025

DANH SÁCH HỘI ĐỒNG BẢO VỆ KHÓA LUẬN

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số ngày của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

1. – Chủ tịch.
2. – Thư ký.
3. – Ủy viên.
4. – Ủy viên.

LỜI CẢM ƠN

Em Trịnh Tân Đạt và Tạ Đức Bảo xin gửi lời cảm ơn chân thành đến thầy Lê Anh Tuấn vì sự hướng dẫn và đóng góp nhiệt tình của thầy trong xuyên suốt quá trình thực hiện khóa luận tốt nghiệp của chúng em.

Cảm ơn thầy đã luôn sẵn lòng dành thời gian cho em, từ việc hướng dẫn chung đến các buổi hướng dẫn cá nhân. Những ý kiến, góp ý và nhận xét của thầy đã giúp chúng em hiểu rõ hơn về yêu cầu của khóa luận này.

Bên cạnh đó, chúng em cũng xin gửi lời cảm ơn đến trường Đại học Công Nghệ Thông Tin – Đại Học Quốc Gia TPHCM đã tạo điều kiện để chúng em có cơ hội mở rộng kiến thức, hiểu biết hơn với môi trường thực tế để thực hiện khóa luận này, giúp bản thân tích lũy được nhiều kinh nghiệm để giải quyết các vấn đề gặp phải trong xuyên suốt quá trình thực hiện khóa luận tốt nghiệp.

Đặc biệt hơn hết, em xin gửi lời cảm ơn chân thành đến khoa Mạng máy tính và Truyền thông dữ liệu đã tạo điều kiện, kiến thức làm cơ sở để chúng em hoàn thành khóa luận này.

Bên cạnh những kiến thức đã tích lũy được áp dụng vào khóa luận này, có thể chúng em vẫn còn nhiều thiếu sót mong được thầy cô góp ý và chia sẻ để chúng em kịp thời sửa chữa và rút kinh nghiệm trong những hành trình sau này.

Cuối cùng, em xin được phép chúc quý thầy cô nói chung và thầy Lê Anh Tuấn nói riêng lời chúc mạnh khỏe và mong quý thầy cô sẽ tiếp tục thành công trong sự nghiệp giảng dạy của mình.

Trân trọng,

Trịnh Tân Đạt (21520714)

Tạ Đức Bảo (21520623)

MỤC LỤC

| | |
|--|-----|
| MỤC LỤC | v |
| DANH MỤC HÌNH VẼ | xi |
| DANH MỤC BẢNG | xiv |
| DANH MỤC TỪ VIẾT TẮT | xv |
| Chương 1. TỔNG QUAN ĐỀ TÀI | 1 |
| 1.1. Đặt vấn đề..... | 1 |
| 1.2. Các nghiên cứu liên quan | 1 |
| 1.3. Mục tiêu nghiên cứu..... | 3 |
| 1.4. Đối tượng và phạm vi nghiên cứu..... | 3 |
| 1.4.1. Đối tượng nghiên cứu..... | 3 |
| 1.4.2. Phạm vi nghiên cứu | 4 |
| 1.5. Phương pháp nghiên cứu..... | 4 |
| 1.6. Cấu trúc khóa luận..... | 5 |
| Chương 2. CƠ SỞ LÝ THUYẾT | 6 |
| 2.1. Kiến trúc Microservices | 6 |
| 2.2. Tổng quan về ứng dụng..... | 6 |
| 2.3. Redis Caching phân tán..... | 7 |
| 2.3.1. Giới thiệu | 7 |
| 2.3.2. Kiến trúc hoạt động Redis | 7 |
| 2.3.3. Khái niệm về Caching phân tán..... | 8 |
| 2.3.3.1. Caching trong hệ thống phân tán | 8 |
| 2.3.3.2. Chiến lược xóa cache..... | 9 |
| 2.4. Apache Kafka..... | 9 |

| | | |
|-----------|---|----|
| 2.4.1. | Giới thiệu | 9 |
| 2.4.2. | Kiến trúc của Kafka..... | 10 |
| 2.4.3. | Cơ chế hoạt động của Kafka..... | 11 |
| 2.4.4. | Khái niệm về thông điệp phân tán..... | 12 |
| 2.4.4.1. | Thông điệp trong hệ thống phân tán..... | 12 |
| 2.5. | Hệ thống quản lý log ELK | 12 |
| 2.5.1. | Định nghĩa và tổng quan..... | 12 |
| 2.5.1.1. | Elasticsearch | 12 |
| 2.5.1.2. | Logstash | 12 |
| 2.5.1.3. | Kibana..... | 13 |
| 2.5.1.4. | Tổng quan | 13 |
| 2.6. | Hệ thống AutoScaling | 14 |
| 2.6.1. | Định nghĩa | 14 |
| 2.6.1.1. | Autoscaling | 14 |
| 2.6.1.2. | Horizontal Pod Autoscaling..... | 14 |
| 2.6.1.3. | KEDA | 15 |
| Chương 3. | PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG | 16 |
| 3.1. | Kiến trúc ứng dụng..... | 16 |
| 3.2. | Sơ đồ Use Cases | 17 |
| 3.3. | Kiến trúc phần mềm | 18 |
| 3.3.1. | View..... | 18 |
| 3.3.2. | Controller..... | 19 |
| 3.3.3. | Model..... | 19 |
| 3.4. | Database Schema..... | 20 |

| | | |
|-----------|--|----|
| 3.5. | Sơ đồ hoạt động qua các giai đoạn triển khai | 21 |
| 3.5.1. | Giai đoạn Dev | 21 |
| 3.5.2. | Giai đoạn Test..... | 22 |
| 3.5.3. | Giai đoạn Production | 24 |
| 3.6. | Thiết kế hệ thống triển khai | 25 |
| 3.6.1. | Giai đoạn Dev | 25 |
| 3.6.2. | Giai đoạn Test..... | 25 |
| 3.6.3. | Giai đoạn Production | 25 |
| 3.7. | Tài nguyên triển khai ứng dụng | 26 |
| 3.8. | Các tài nguyên triển khai khác | 27 |
| Chương 4. | HIỆN THỰC HỆ THỐNG | 28 |
| 4.1. | Tính năng phát trực tiếp | 28 |
| 4.1.1. | Cấu hình Nginx..... | 28 |
| 4.1.2. | Thử nghiệm Live Stream trên ứng dụng | 30 |
| 4.2. | Triển khai Redis cho ứng dụng | 31 |
| 4.2.1. | Triển khai Redis cho User-Service..... | 31 |
| 4.2.1.1. | Caching thông tin người dùng | 31 |
| 4.2.1.2. | Cấu hình Redis trong User-Service | 31 |
| 4.2.1.3. | Cấu trúc cache..... | 33 |
| 4.2.1.4. | Hết hạn và xóa cache | 34 |
| 4.2.1.5. | Kết luận..... | 34 |
| 4.2.2. | Tối ưu hóa Redis..... | 34 |
| 4.2.2.1. | Vấn đề gấp phải | 34 |
| 4.2.2.2. | Redis cluster..... | 35 |

| | |
|--|----|
| 4.2.2.3. Lợi ích của Redis Cluster trong hệ thống phân tán | 35 |
| 4.2.3. Tính đồng bộ và quản lý dữ liệu cache..... | 36 |
| 4.2.3.1. Đồng bộ cache với các cơ sở dữ liệu | 36 |
| 4.2.3.2. Kết luận..... | 37 |
| 4.2.4. Triển khai hệ thống Redis lên K8S..... | 37 |
| 4.3. Triển khai giải pháp xử lý dữ liệu thời gian thực..... | 39 |
| 4.3.1. Kafka trong Predict Toxic Comment | 39 |
| 4.3.1.1. Cấu trúc hệ thống Predict Toxic Comment | 39 |
| 4.3.1.2. Triển khai Kafka cho Predict Toxic Comment..... | 40 |
| 4.3.2. Kafka trong Video Recommendation..... | 41 |
| 4.3.2.1. Cấu trúc hệ thống để xuất video | 41 |
| 4.4. Triển khai mô hình dự đoán bình luận tiêu cực | 42 |
| 4.4.1. Giới thiệu | 42 |
| 4.4.1.1. Mục tiêu | 42 |
| 4.4.1.2. Giới thiệu về BERT | 42 |
| 4.4.2. Tổng quan | 43 |
| 4.4.2.1. Dữ liệu thu thập | 43 |
| 4.4.2.2. Các bước thực hiện | 44 |
| 4.4.3. Phân tích mật mã | 44 |
| 4.4.3.1. Khai báo thư viện..... | 44 |
| 4.4.3.2. Thu thập dữ liệu..... | 45 |
| 4.4.3.3. Kết hợp dữ liệu | 49 |
| 4.4.3.4. Xây dựng mô hình và token..... | 50 |
| 4.4.3.5. Chuẩn bị dữ liệu để huấn luyện | 53 |

| | | |
|----------|--|----|
| 4.4.3.6. | Huấn luyện mô hình..... | 55 |
| 4.4.3.7. | Kiểm tra và thử nghiệm | 58 |
| 4.5. | Triển khai mô hình AI cho hệ thống gọi ý video | 60 |
| 4.5.1. | Giới thiệu | 60 |
| 4.5.1.1. | Mục tiêu và yêu cầu của hệ thống gọi ý | 60 |
| 4.5.1.2. | Giới thiệu về sphương pháp sử dụng AI trong gọi ý video .. | 60 |
| 4.5.2. | Dữ liệu sử dụng cho việc huấn luyện | 60 |
| 4.5.2.1. | Giới thiệu về Youtube-8m dataset | 61 |
| 4.5.2.2. | Chuẩn bị dữ liệu..... | 61 |
| 4.5.3. | Mô hình AI cho hệ thống gọi ý video | 63 |
| 4.5.3.1. | Mô hình và kiến trúc mạng nơ-ron | 63 |
| 4.5.4. | Triển khai hệ thống và môi trường huấn luyện | 64 |
| 4.5.4.1. | Môi trường huấn luyện | 64 |
| 4.5.4.2. | Các thông số huấn luyện..... | 65 |
| 4.5.4.3. | Tài nguyên huấn luyện..... | 66 |
| 4.5.5. | Cách tiếp cận và tối ưu hóa huấn luyện..... | 66 |
| 4.5.5.1. | Huấn luyện mô hình..... | 66 |
| 4.5.6. | Kết luận..... | 82 |
| 4.5.6.1. | Ưu điểm của mô hình | 82 |
| 4.5.6.2. | Nhược điểm của mô hình..... | 83 |
| 4.5.6.3. | Định hướng phát triển | 83 |
| 4.6. | Triển khai hệ thống quản lý log cho ứng dụng | 84 |
| 4.6.1. | Cấu hình Logstash | 84 |
| 4.6.1.1. | Cấu hình chung | 85 |

| | |
|---|------------|
| 4.6.1.2. Cấu hình User Services..... | 85 |
| 4.6.1.3. Cấu hình Video Service | 86 |
| 4.6.1.4. Cấu hình Comment Service | 86 |
| 4.7. Triển khai hệ thống Autoscale cho K8S | 87 |
| 4.8. Đặc tả giao diện tính năng ứng dụng..... | 89 |
| 4.9. Hiện thực triển khai hệ thống | 93 |
| 4.9.1. CI/CD pipeline..... | 93 |
| 4.9.2. Triển khai lên Server | 94 |
| 4.9.3. Autoscale | 95 |
| 4.9.4. Hệ thống quản lý Log ELK | 95 |
| 4.9.4.1. Log của các Service | 95 |
| 4.9.4.2. Log của hệ thống | 97 |
| 4.9.5. Quản lý Cluster..... | 98 |
| Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN | 99 |
| 5.1. Kết luận | 99 |
| 5.1.1. Ứng dụng CI/CD | 99 |
| 5.1.2. Tích hợp AI..... | 99 |
| 5.1.3. Khả năng mở rộng và quản lý hạ tầng..... | 99 |
| 5.2. Hướng phát triển | 100 |
| 5.2.1. Cải thiện mô hình AI để xuất video..... | 100 |
| 5.2.2. Tìm hiểu và viết bài báo khoa học..... | 100 |
| 5.2.3. Tìm nhà đầu tư..... | 100 |
| 5.2.4. Cải thiện hệ thống phân phối video | 100 |
| TÀI LIỆU THAM KHẢO | 101 |

DANH MỤC HÌNH VẼ

| | |
|--|----|
| Hình 2.1: Mô hình ứng dụng Redis..... | 7 |
| Hình 2.2: Apache Kafka..... | 9 |
| Hình 2.3: Cấu trúc Kafka đơn giản | 10 |
| Hình 2.4: Cơ chế hoạt động của Kafka | 11 |
| Hình 2.5: Thông điệp phân tán..... | 12 |
| Hình 2.6: Mô hình hoạt động ELK | 13 |
| Hình 2.7: Cách hoạt động của Horizontal Pod Autoscale..... | 14 |
| Hình 3.1: Kiến trúc ứng dụng | 16 |
| Hình 3.2: Sơ đồ Use Cases..... | 17 |
| Hình 3.3: Kiến trúc phần mềm | 18 |
| Hình 3.4: Database Schema | 20 |
| Hình 3.5: Giai đoạn Dev | 21 |
| Hình 3.6: Giai đoạn Test | 22 |
| Hình 3.7: Giai đoạn Production | 24 |
| Hình 3.8: Mô hình triển khai đối với ứng dụng | 26 |
| Hình 3.9: Mô hình triển khai 1 số tài nguyên khác | 27 |
| Hình 4.1: Cấu hình OBS Studio cho Live Streaming | 29 |
| Hình 4.2: Lấy Stream Key từ ứng dụng | 30 |
| Hình 4.3: Cấu hình Streaming trên OBS Studio | 30 |
| Hình 4.4: Kiểm tra tính năng Streaming thử nghiệm..... | 30 |
| Hình 4.5: Kiểm tra thư mục HLS | 31 |
| Hình 4.6: Mô hình Redis Cluster | 35 |
| Hình 4.7: Mô hình đồng bộ cơ sở dữ liệu | 36 |
| Hình 4.8: StatefulSet Redis | 38 |
| Hình 4.9: Pod Redis | 38 |
| Hình 4.10: Cấu trúc Kafka trong việc xử lý bình luận..... | 40 |
| Hình 4.11: Phân loại bình luận..... | 41 |
| Hình 4.12: Cấu trúc hệ thống đề xuất Video..... | 41 |

| | |
|--|----|
| Hình 4.13: Tải và cài đặt thư viện | 45 |
| Hình 4.14: Đầu nhãn dữ liệu | 45 |
| Hình 4.15: Mô tả tập dữ liệu | 46 |
| Hình 4.16: Biểu đồ thống kê dữ liệu | 46 |
| Hình 4.17: Số liệu thống kê các nhãn | 47 |
| Hình 4.18: Biểu đồ trực quan 2 tập dữ liệu..... | 48 |
| Hình 4.19: WordCloud cho bình luận độc hại | 48 |
| Hình 4.20: WordCloud cho bình luận sạch..... | 48 |
| Hình 4.21: Các tập dữ liệu từ dataframe | 49 |
| Hình 4.22: Kiểm tra Dataframe..... | 49 |
| Hình 4.23: Khởi tạo Token | 51 |
| Hình 4.24: Khởi tạo mô hình huấn luyện..... | 52 |
| Hình 4.25: Phần cứng và thông tin mô hình BERT | 53 |
| Hình 4.26: Kết quả xử lý dữ liệu..... | 54 |
| Hình 4.27: Batch từ tập dữ liệu huấn luyện | 55 |
| Hình 4.28: Giá trị huấn luyện mô hình | 56 |
| Hình 4.29: Kết quả đánh giá mô hình | 57 |
| Hình 4.30: Lưu mô hình..... | 58 |
| Hình 4.31: Thủ nghiệm trên 1 bình luận..... | 58 |
| Hình 4.32: Bình luận sạch..... | 59 |
| Hình 4.33: Kết quả sau xử lý AI | 59 |
| Hình 4.34: Bình luận độc hại | 59 |
| Hình 4.35: Kết quả sau xử lý AI | 59 |
| Hình 4.36: Bình luận kí tự bất kì..... | 60 |
| Hình 4.37: Kết quả khi tải dataset về | 68 |
| Hình 4.38: Kết quả sau khi tiền xử lý dữ liệu | 71 |
| Hình 4.39: Dữ liệu chuẩn bị cho mô hình | 73 |
| Hình 4.40: Kết quả sau khi huấn luyện | 75 |
| Hình 4.41: Phân tích độ chính xác | 76 |

| | |
|---|----|
| Hình 4.42: Phân tích sự mất mát dữ liệu..... | 77 |
| Hình 4.43: Đánh Giá Hiệu Suất Mô Hình Trên Tập Validation | 79 |
| Hình 4.44: Phân tích phân phối nhãn trong tập huấn luyện | 80 |
| Hình 4.45: Đường Cong Precision-Recall | 81 |
| Hình 4.46: Tóm tắt kết quả huấn luyện mô hình | 82 |
| Hình 4.47: Kiểm tra Metrics Server | 87 |
| Hình 4.48: Kiểm tra KEDA | 88 |
| Hình 4.49: Giao diện chính của ứng dụng | 89 |
| Hình 4.50: Màn hình xem video | 90 |
| Hình 4.51: Màn hình hồ sơ người dùng | 90 |
| Hình 4.52: Màn hình tải lên video | 91 |
| Hình 4.53: Màn hình thay đổi thông tin cá nhân | 91 |
| Hình 4.54: Màn hình phát trực tiếp | 92 |
| Hình 4.55: Màn hình xem phát trực tiếp | 92 |
| Hình 4.56: CI pipeline | 93 |
| Hình 4.57: SonarQube quét code | 93 |
| Hình 4.58: CD pipeline | 93 |
| Hình 4.59: Thông báo CI/CD từ Slack..... | 94 |
| Hình 4.60: Các pod chính của ứng dụng | 94 |
| Hình 4.61: Các Horizontal Pod Autoscaler..... | 95 |
| Hình 4.62: Log người dùng đăng nhập và đăng ký..... | 95 |
| Hình 4.63: Log đăng tải video mới | 96 |
| Hình 4.64: Log xem video | 96 |
| Hình 4.65: Log ELK của Comment Service | 96 |
| Hình 4.66: Log được gửi từ filebeat..... | 97 |
| Hình 4.67: Trường thông tin “message” | 97 |
| Hình 4.68: Dashboard | 97 |
| Hình 4.69: Quản lý Cluster | 98 |

DANH MỤC BẢNG

| | |
|------------------------------------|---|
| Bảng 1.1: Cấu trúc khóa luận | 5 |
|------------------------------------|---|

DANH MỤC TỪ VIẾT TẮT

| STT | Ký hiệu chữ viết tắt | Chữ viết đầy đủ |
|------------|-----------------------------|---|
| 1 | CI | Continuos Integration |
| 2 | CD | Continuos Deployment |
| 3 | K8S | Kubernetes |
| 4 | UI | User Interface |
| 5 | AI | Artificial Intelligence |
| 6 | ELK | Elasticsearch, Logstash & Kibana |
| 7 | API | Application Programming Interface |
| 8 | RTMP | Real-Time Messaging Protocol |
| 9 | HLS | HTTP Live Streaming |
| 10 | CORS | Cross-Origin Resource Sharing |
| 11 | LRU | Least Recently Used |
| 12 | LFU | Least Frequently Used |
| 13 | TTL | Time To Live |
| 14 | BERT | Bidirectional Encoder Representations from Transformers |
| 15 | UNICEF | United Nations Children's Fund |
| 16 | MDC | Mapped Diagnostic Context |
| 17 | HPA | Horizontal Pod AutoScaling |
| 18 | KEDA | Kubernetes Event-Driven Autoscaler |

Chương 1. TỔNG QUAN ĐỀ TÀI

1.1. Đặt vấn đề

Trong kỷ nguyên số hóa và xu hướng cá nhân hóa các dịch vụ trên Internet, nhu cầu chia sẻ video cá nhân dần đã trở nên phổ biến. Tuy nhiên, để đáp ứng được khối lượng người dùng đông đảo và yêu cầu cao về hiệu suất, hệ thống cần được thiết kế và phát triển dựa trên kiến trúc Microservices.

Kiến trúc Microservices không chỉ đem lại tính linh hoạt trong quá trình phát triển và triển khai, mà còn giúp tối ưu hóa hiệu suất và khả năng mở rộng. Song song đó, sự kết hợp với các kỹ thuật DevOps, AI, Java Springboot và công cụ như CI/CD sẽ giúp tự động hóa quy trình phát triển, đổi mới nhanh chóng và giảm thiểu nguy cơ khi thay đổi.

Đề tài “Phát triển hệ thống phân tán Microservices cho ứng dụng chia sẻ video” được thực hiện nhằm tạo ra một nền tảng chia sẻ video độc lập, linh hoạt và mang lại giá trị thực tiễn cho người sử dụng.

1.2. Các nghiên cứu liên quan

Các nghiên cứu trước đây đã chỉ ra rằng việc áp dụng kiến trúc Microservices cũng như các kỹ thuật DevOps trong phát triển hệ thống phân tán mang lại nhiều lợi ích về tính mở rộng và độ bền:

- "Reimagining video infrastructure to empower YouTube": Bài viết này trên blog chính thức của YouTube cung cấp cái nhìn sâu sắc về cách YouTube đã tái cấu trúc hạ tầng video để đáp ứng nhu cầu ngày càng tăng của người dùng. Bài viết nhấn mạnh việc triển khai các công nghệ mới để cải thiện hiệu suất và trải nghiệm người dùng [1].

Bài viết tập trung vào việc mô tả cách YouTube đã cải tiến hạ tầng video để: Tăng hiệu suất - Giảm độ trễ và cải thiện tốc độ tải video. Cá nhân hóa nội dung - Sử dụng công nghệ AI và học máy để đề xuất video phù hợp với sở thích người dùng.

Tối ưu hóa chi phí - Triển khai các giải pháp hiệu quả trong quản lý lưu trữ và phân phối dữ liệu.

Các công nghệ nổi bật được nêu trong bài báo bao gồm “Dynamic Adaptive Streaming over HTTP” được Youtube sử dụng để cải thiện trải nghiệm người dùng, đặc biệt trong các môi trường mạng có băng thông không ổn định. YouTube sử dụng “VP9” để tối ưu hóa việc nén và truyền tải video, từ đó giảm băng thông cần thiết và “AV1” được nhắc đến trong bối cảnh là công nghệ tiềm năng mà YouTube đang thử nghiệm để nâng cao hiệu quả. Sau đó, “Quick UDP Internet Connections” được giới thiệu như một phần quan trọng của hạ tầng YouTube để tăng tốc độ truyền tải và cải thiện chất lượng trải nghiệm. Cuối cùng, “AI và Machine Learning” được nhấn mạnh trong việc cá nhân hóa trải nghiệm người dùng, tối ưu hóa việc đề xuất video và quản lý nội dung

- “A Netflix Special — Product Case Study (Tech Focused)”: Bài viết trên phân tích cách Netflix tận dụng công nghệ để xây dựng sản phẩm vượt trội, tập trung vào ba cấp độ: hạ tầng, nền tảng và trải nghiệm người dùng [2]

Bài viết mô tả tổng quan về hạ tầng và công nghệ Netflix sử dụng như sau:

Tổng quan hạ tầng của Netflix: Netflix xây dựng kiến trúc hạ tầng dựa trên cloud computing và tập trung vào ba cấp độ chính: Về hạ tầng Netflix sử dụng Amazon Web Services (AWS) để vận hành và lưu trữ dữ liệu, đảm bảo tính linh hoạt và khả năng mở rộng. Về nền tảng Netflix tích hợp các công cụ và framework để phát triển dịch vụ, đảm bảo tính nhất quán và hiệu suất. Trải nghiệm người dùng. Bên cạnh đó Netflix tận dụng các thuật toán AI và Machine Learning để cá nhân hóa nội dung và tối ưu hóa giao diện người dùng.

Công nghệ chính: Netflix đã chuyển đổi từ các trung tâm dữ liệu truyền thống sang hoàn toàn sử dụng Amazon Web Service. Trong đó, “Cloud Computing” được Netflix xử lý lưu lượng lớn trong các khung giờ cao điểm đi kèm với khả năng mở rộng và tính sẵn sàng cao. Netflix sử dụng kiến trúc “microservices” để chia nhỏ hệ thống thành nhiều dịch vụ độc lập như quản lý tài khoản, phát video, hay đề xuất nội

dung. Bên cạnh đó, Netflix đã phát triển Content Delivery Network riêng, giúp giảm tải cho các nhà cung cấp dịch vụ internet và cải thiện chất lượng phát video. Cuối cùng, Netflix thu thập dữ liệu người dùng (lịch sử xem, thời gian dừng, nội dung ưa thích) để cá nhân hóa trải nghiệm bằng cách sử dụng các thuật toán AI giúp đề xuất nội dung chính xác hơn.

1.3. Mục tiêu nghiên cứu

Đề tài "Phát triển hệ thống phân tán Microservices cho ứng dụng chia sẻ video" xây dựng một ứng chia sẻ video thông qua nghiên cứu, áp dụng các nguyên lý của kiến trúc microservice và các kỹ thuật DevOps trong việc phát triển và triển khai. Bên cạnh đó, ứng dụng sẽ được tích hợp trí tuệ nhân tạo qua việc xây dựng xây dựng các mô hình học sâu nhằm tăng thêm tính thực tiễn của ứng dụng và tăng trải nghiệm người dùng.

Đề tài không chỉ giúp nâng cao kỹ thuật mà còn phát triển các kỹ năng trong môi trường thực tế như làm việc nhóm và quản lý dự án, hướng tới mục tiêu hoàn thiện một sản phẩm ứng dụng trong môi trường thực tiễn.

Cuối cùng, ứng dụng chia sẻ video được xây dựng để hướng tới một sản phẩm miễn phí, dễ dàng sử dụng đối với những doanh nghiệp nhỏ, các lớp học cần chia sẻ và đăng tải những video phù hợp với nhu cầu.

1.4. Đối tượng và phạm vi nghiên cứu

1.4.1. Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là các nguyên tắc và phương pháp thiết kế hệ thống Microservices trên ứng dụng Java Springboot, bao gồm:

- Kiến trúc Microservices.
- Các kỹ thuật và công cụ DevOps như CI/CD, Kubernetes.
- Các kỹ thuật tối ưu hóa hiệu năng như Redis và Kafka.
- Ứng dụng các mô hình học sâu trong phát triển tính năng AI cho hệ thống.

1.4.2. Phạm vi nghiên cứu

Phạm vi nghiên cứu sẽ đánh giá sự hiệu quả của hệ thống phân tán lên ứng dụng qua các tính năng như chia sẻ video, người dùng, bình luận, khả năng tối ưu hóa trong thời gian thực và khả năng đưa ứng dụng vào môi trường thực tế.

1.5. Phương pháp nghiên cứu

Đề tài "Phát triển hệ thống phân tán Microservices cho ứng dụng chia sẻ video", được áp dụng các phương pháp nghiên cứu sau:

- Tìm hiểu lý thuyết và nền tảng kiến thức liên quan: Thu thập các tài liệu khoa học, bài báo, tài liệu hướng dẫn và các nghiên cứu trước đây liên quan đến kiến trúc Microservices, DevOps, AI, và Java Spring Boot.
- Phân tích thực trạng các hệ thống tương tự: Nghiên cứu các hệ thống chia sẻ video hiện tại (YouTube) để rút ra những ưu và nhược điểm, từ đó đưa ra hướng phát triển phù hợp.
- Nghiên cứu và phát triển mô hình học sâu: Xây dựng các mô hình AI để tích hợp các tính năng như gợi ý video, phát hiện nội dung bình luận không phù hợp
- Kiểm thử mô hình: Sử dụng bộ dữ liệu mẫu để huấn luyện và đánh giá hiệu quả của mô hình trước khi tích hợp vào hệ thống.
- Phân tích kết quả: Đánh giá hệ thống qua các chỉ số hiệu suất tính ổn định và trải nghiệm người dùng.
- Cải tiến: Tối ưu hóa hệ thống dựa trên kết quả thử nghiệm

1.6. Cấu trúc khóa luận

Khóa luận của nhóm chúng em đề tài **PHÁT TRIỂN HỆ THỐNG PHÂN TÁN MICROSERVICES CHO ỨNG DỤNG CHIA SẺ VIDEO** được trình bày bao gồm 5 chương với mô tả như bảng sau:

| Chương | Mô tả |
|--|--|
| Chương 1: Tổng quan đề tài | Giới thiệu tổng quan về đề tài, mô tả yêu cầu trong việc xây dựng ứng dụng |
| Chương 2: Cơ sở lý thuyết | Trình bày lý thuyết về các công cụ được dùng trong ứng dụng |
| Chương 3: Phân tích và thiết kế hệ thống | Mô tả hệ thống ứng dụng và kiến trúc triển khai ứng dụng |
| Chương 4: Hiện thực hệ thống | Cách triển khai các cấu trúc cần thiết để chạy ứng dụng |
| Chương 5: Kết luận và hướng phát triển | Đưa ra cái nhìn tổng quát và định hướng tương lai |

Bảng 1.1: Cấu trúc khóa luận

Chương 2. CƠ SỞ LÝ THUYẾT

2.1. Kiến trúc Microservices

Microservice là một kiến trúc phần mềm trong đó các ứng dụng được cấu thành từ nhiều dịch vụ nhỏ, độc lập, và có thể triển khai riêng lẻ. Mỗi dịch vụ được xây dựng để thực hiện một chức năng cụ thể và giao tiếp với các dịch vụ khác thông qua các giao thức nhẹ như HTTP hoặc message queue. Kiến trúc này mang lại nhiều lợi ích, bao gồm tính linh hoạt, khả năng mở rộng và bảo trì dễ dàng.

Ứng dụng Video Sharing được triển khai thành 3 Microservice khác nhau nhằm thuận tiện cho việc triển khai, quản lý tài nguyên và tăng tính bảo mật của ứng dụng.

2.2. Tổng quan về ứng dụng

Ứng dụng Video Sharing sử dụng kiến trúc microservice để chia nhỏ các chức năng của hệ thống thành các dịch vụ độc lập. Các dịch vụ này được container hóa bằng Docker và triển khai trên Kubernetes. Hệ thống bao gồm các thành phần chính như:

- Frontend (ReactJS): Giao diện người dùng động, phản hồi nhanh, bắt mắt và vô cùng thuận tiện với người dùng.
- Backend (Java Spring Boot): Xử lý logic nghiệp vụ, tương tác với cơ sở dữ liệu, thân thiện, dễ sử dụng và thao tác phản hồi tốt.
- Backend (Python Flask): Xử dụng Python để xử lý các mô hình AI và giao tiếp với các service khác thông qua Kafka
- Database (MongoDB): Lưu trữ dữ liệu người dùng, video và bình luận.
- Authentication (Spring Security): Quản lý xác thực và phân quyền, đảm bảo tính bảo mật và an toàn cho ứng dụng.
- CI/CD (GitHub Actions): Tự động hóa xây dựng, kiểm thử và triển khai tạo nên sự thuận tiện trong việc cập nhật, triển khai.

- Code Quality (SonarCloud): Đảm bảo chất lượng mã nguồn, tránh sai sót trong việc lập trình.
- Security Scanning (Trivy): Kiểm tra lỗ hổng bảo mật trong container.

Kiến trúc này đảm bảo hệ thống có khả năng mở rộng, linh hoạt, và dễ bảo trì, đồng thời cung cấp trải nghiệm người dùng tốt và bảo mật cao. Tạo nên 1 ứng dụng có khả năng phát triển vô cùng tốt trong ngành công nghệ thông tin.

2.3. Redis Caching phân tán

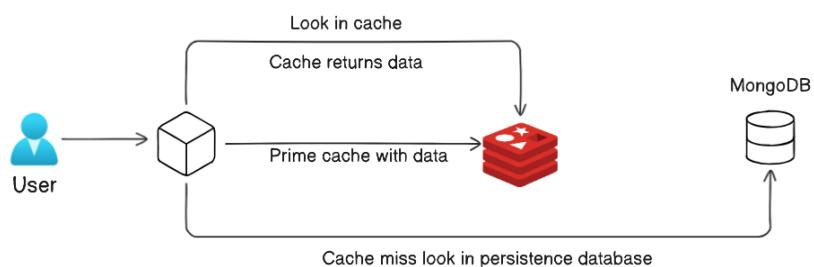
2.3.1. Giới thiệu

Redis là một cơ sở dữ liệu hoạt động hoàn toàn trên bộ nhớ RAM, điều này giúp tăng tốc độ truy xuất dữ liệu do không phải chờ đợi quá trình đọc ghi từ ổ cứng như truyền thống. Điều này giúp mang lại khả năng xử lý nhanh chóng và hiệu quả cho các tác vụ cần truy xuất dữ liệu đồng thời.

Là một cơ sở dữ liệu NoSQL, Redis mang lại sự linh hoạt cho thiết kế cấu trúc dữ liệu, phù hợp với các nhu cầu khác nhau của ứng dụng. Redis có thể sử dụng cho những tác vụ đơn giản như theo dõi số lượng người truy cập hoặc những tác vụ lớn như lưu trữ thông tin chi tiết về người dùng.

Trong nhiều hệ thống, Redis thường được sử dụng làm cache layer để lưu trữ các kết quả truy vấn phổ biến hoặc các dữ liệu được sử dụng thường xuyên, giúp giảm tải cho cơ sở dữ liệu chính và tăng tốc độ phản hồi cho người dùng.

2.3.2. Kiến trúc hoạt động Redis



Hình 2.1: Mô hình ứng dụng Redis

Kiến trúc sử dụng Redis thường bao gồm 2 thành phần chính: cache (Redis) và cơ sở dữ liệu chính. Quy trình hoạt động gồm 2 bước chính: Kiểm tra dữ liệu trong Redis cache và truy vấn cơ sở dữ liệu.

- **Bước 1:** Kiểm tra cache

Khi một yêu cầu truy vấn dữ liệu được gửi từ ứng dụng, hệ thống sẽ kiểm tra Redis cache để xác định xem dữ liệu đã có sẵn trong bộ nhớ đệm hay chưa.

Nếu dữ liệu có sẵn trong cache, hệ thống sẽ truy xuất dữ liệu từ Redis và trả kết quả ngay lập tức. Thời gian truy xuất trong cache thường rất nhanh do Redis hoạt động trên RAM, điều này giúp giảm đáng kể độ trễ so với việc truy vấn vào cơ sở dữ liệu chính.

- **Bước 2:** Truy vấn cơ sở dữ liệu chính

Nếu dữ liệu không có trong cache “cache miss”. Lúc đó, hệ thống sẽ thực hiện một truy vấn tới cơ sở dữ liệu để lấy dữ liệu. Sau khi dữ liệu được truy xuất từ cơ sở dữ liệu chính, hệ thống sẽ:

- Trả dữ liệu này về cho ứng dụng.
- Lưu dữ liệu vừa truy xuất được từ cơ sở dữ liệu vào Redis để phục vụ cho các yêu cầu tiếp theo.

2.3.3. Khái niệm về Caching phân tán

2.3.3.1. Caching trong hệ thống phân tán

Caching phân tán trong hệ thống microservice hoạt động theo cơ chế cache-aside. Quy trình gồm:

- Cache hit: Khi một yêu cầu dữ liệu được gửi từ ứng dụng, hệ thống sẽ kiểm tra cache trước. Nếu dữ liệu tồn tại trong cache, nó sẽ được trả về ngay lập tức, giúp giảm đáng kể thời gian truy xuất.
- Cache miss: Nếu dữ liệu không tồn tại trong cache, hệ thống sẽ truy vấn từ cơ sở dữ liệu chính và lưu lại dữ liệu trong cache cho các lần yêu cầu tiếp theo.

Điều này đảm bảo rằng dữ liệu phổ biến được lưu trữ trong cache, giúp giảm thiểu số lần truy cập tới cơ sở dữ liệu và cải thiện tốc độ phản hồi cho người dùng.

2.3.3.2. Chiến lược xóa cache

Việc quản lý dung lượng và thời gian sống của dữ liệu trong cache là một thách thức quan trọng. Redis hỗ trợ nhiều chiến lược xóa cache để đảm bảo rằng dữ liệu cũ hoặc ít sử dụng sẽ được loại bỏ, nhường chỗ cho dữ liệu mới:

- Time To Live (TTL): Redis tự động xóa các khóa sau một thời gian được thiết lập trước. TTL đảm bảo dữ liệu cache luôn tươi mới và không trở nên lỗi thời.

2.4. Apache Kafka

2.4.1. Giới thiệu

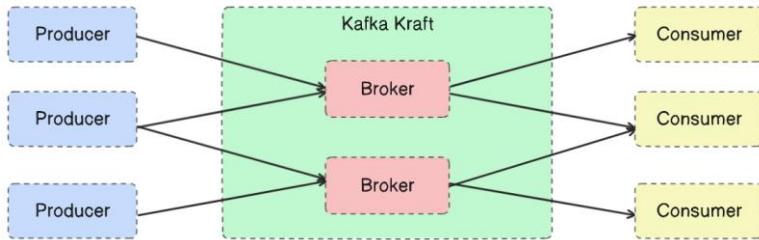


Hình 2.2: Apache Kafka

Apache Kafka là một nền tảng truyền thông điệp phân tán mạnh mẽ, được thiết kế để xử lý khối lượng lớn dữ liệu theo thời gian thực với khả năng mở rộng cao. Được phát triển ban đầu bởi LinkedIn và sau đó chuyển sang Apache Software Foundation.

Trong kiến trúc microservice, các dịch vụ thường cần trao đổi thông tin dưới dạng thông điệp hoặc dữ liệu. Kafka đóng vai trò là lớp trung gian, đảm bảo rằng các thông điệp được truyền đi một cách đáng tin cậy giữa các dịch vụ. Điều này giúp tách biệt sự phụ thuộc giữa các dịch vụ và đảm bảo hệ thống có khả năng xử lý lưu lượng lớn mà không bị gián đoạn. Với khả năng chịu tải cao, Kafka có thể xử lý hàng triệu thông điệp mỗi giây, vì vậy nó đã trở thành lựa chọn lý tưởng cho các hệ thống microservice hiện đại.

2.4.2. Kiến trúc của Kafka



Hình 2.3: Cấu trúc Kafka đơn giản

Kafka được xây dựng trên một kiến trúc phân tán với các thành phần chính như sau:

Broker: là thành phần trung gian của Kafka, chịu trách nhiệm lưu trữ, quản lý và phân phối thông điệp. Trong hệ thống, một cụm Kafka có thể bao gồm nhiều broker, đảm bảo tính chịu lỗi và khả năng mở rộng. Mỗi broker sẽ chịu trách nhiệm quản lý một phần thông điệp và tương tác với các broker khác để đồng bộ hóa dữ liệu và duy trì tính nhất quán.

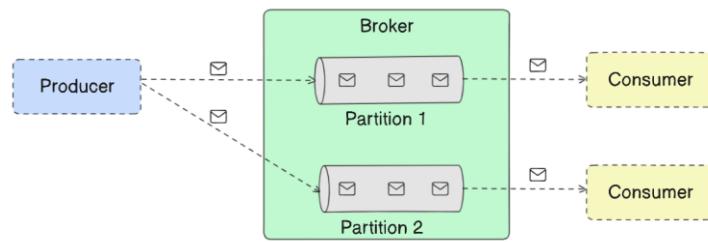
Topic: là nơi các thông điệp được tổ chức và phân loại. Mỗi topic có thể được chia thành nhiều partition để tăng khả năng xử lý đồng thời. Topic cho phép các bên gửi (producer) ghi thông điệp và bên nhận (consumer) đọc thông điệp. Tạo nên một mô hình xuất-nhập (pub-sub) mạnh mẽ cho việc xử lý dữ liệu theo thời gian thực.

Partition: Mỗi topic trong Kafka được chia thành nhiều partition để tối ưu hóa hiệu suất và khả năng mở rộng. Partition cho phép lưu trữ và quản lý dữ liệu theo dạng song song, tăng cường khả năng đọc/ghi thông điệp đồng thời từ nhiều nguồn. Partition đảm bảo rằng dữ liệu có thể được xử lý theo trình tự và dễ dàng quản lý sự phân phối của dữ liệu trong cụm Kafka.

Producer: chịu trách nhiệm gửi thông điệp đến các topic cụ thể trong Kafka. Mỗi producer có thể cấu hình để lựa chọn partition nào sẽ lưu trữ thông điệp, dựa trên các giá trị khóa hoặc chính sách phân phối ngẫu nhiên. Producer là nơi ghi dữ liệu vào hệ thống Kafka và đảm bảo rằng thông điệp được gửi đến đúng topic và partition.

Consumer: là thành phần nhận và xử lý các thông điệp từ Kafka. Các consumer được tổ chức thành nhóm (consumer group), mỗi nhóm có thể đọc từ một hoặc nhiều topic. Các consumer trong cùng một nhóm sẽ chia sẻ công việc và đảm bảo rằng mỗi partition chỉ được một consumer xử lý tại một thời điểm, giúp tối ưu hóa khả năng xử lý đồng thời.

2.4.3. Cơ chế hoạt động của Kafka



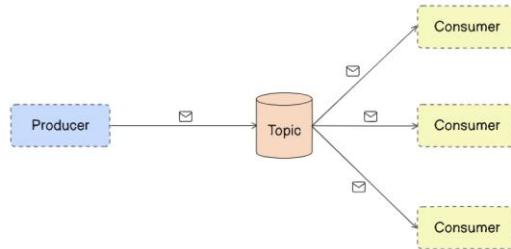
Hình 2.4: Cơ chế hoạt động của Kafka

Ghi/đọc thông điệp: Cơ chế ghi thông điệp của Kafka dựa trên các producer, nơi các producer ghi thông điệp gửi dữ liệu đến các partition của topic. Kafka cho phép producer ghi thông điệp một cách nhanh chóng bằng cách vào việc tổ chức dữ liệu tuần tự trên đĩa, giúp giảm thiểu thời gian I/O. Các thông điệp sau khi được ghi vào partition sẽ được lưu giữ trong một khoảng thời gian xác định, đảm bảo khả năng truy xuất dữ liệu trong tương lai.

Về phía consumer, Kafka sử dụng mô hình pull, các consumer chủ động truy xuất thông điệp từ Kafka thay vì nhận thông điệp từ hệ thống. Điều này giúp consumer có thể kiểm soát tốc độ đọc dữ liệu và tránh tình trạng quá tải.

2.4.4. Khái niệm về thông điệp phân tán

2.4.4.1. Thông điệp trong hệ thống phân tán



Hình 2.5: Thông điệp phân tán

Trong các hệ thống phân tán, thông điệp là cách thức mà các thành phần trong hệ thống giao tiếp với nhau. Một thông điệp có thể được định nghĩa là một đơn vị dữ liệu, chứa đựng thông tin cần thiết để truyền tải từ một dịch vụ này đến một dịch vụ khác trong một hệ thống lớn. Thông điệp bao gồm phần payload (nội dung chính của thông điệp) và các metadata liên quan như thời gian gửi, producer, và các thông tin điều khiển khác.

2.5. Hệ thống quản lý log ELK

2.5.1. Định nghĩa và tổng quan

2.5.1.1. Elasticsearch

Elasticsearch là một công cụ phân tích và tìm kiếm RESTful phân tán, database có thể mở rộng và cơ sở dữ liệu vector. Nó được thiết kế để giải quyết vô số trường hợp sử dụng. Nền tảng của Elastic Stack tập trung lưu trữ dữ liệu và cung cấp khả năng tìm kiếm cực nhanh, mức độ liên quan cao và khả năng phân tích mạnh mẽ, có thể mở rộng. [3]

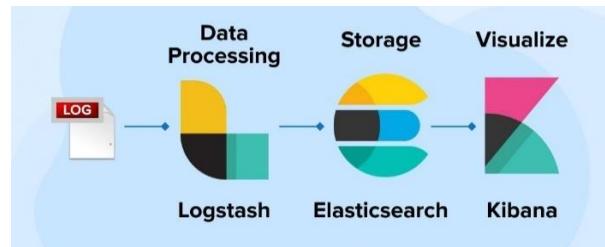
2.5.1.2. Logstash

Logstash là một đường dẫn phía máy chủ để xử lý dữ liệu. Nhiệm vụ là để nhập dữ liệu từ nhiều nguồn, sau đó chuyển đổi và gửi dữ liệu đó đến hệ thống lưu trữ của máy chủ. [4]

2.5.1.3. Kibana

Kibana cung cấp khả năng trực quan hóa bên trong nội dung được lập chỉ mục trên cụm Elasticsearch. Người dùng có thể tạo các biểu đồ thanh, đường và phân tán hoặc biểu đồ hình tròn và bản đồ trên khối lượng dữ liệu lớn. [5]

2.5.1.4. Tổng quan



Hình 2.6: Mô hình hoạt động ELK

Elasticsearch, Logstash và Kibana kết hợp với nhau tạo thành công cụ cung cấp một giải pháp tích hợp, mạnh mẽ để quản lý khối lượng dữ liệu lớn, cung cấp thông tin chi tiết theo thời gian thực và bộ phân tích toàn diện.

Cách hoạt động của ELK trong ứng dụng Microservices được chúng em xây dựng như sau: Với mỗi services sẽ được gắn thêm các log tại những hoạt động tác động đến ứng dụng của người dùng như thêm bình luận, đăng tải video, đăng ký, đăng nhập người dùng. Logstash sẽ nhận các log trên để gửi đến hệ thống Elasticsearch để lưu trữ và hiển thị để người quản lý theo dõi qua Kibana, bên cạnh đó Kibana sẽ đóng vai trò ảo hóa các dữ liệu trên để đưa ra những thông số, biểu đồ nhằm trực quan hóa dữ liệu. Ngoài ra chúng em sẽ tận dụng filebeat để quản lý server chạy ELK nhằm tăng cường bảo mật và quản lý cluster dễ dàng hơn.

2.6. Hệ thống AutoScaling

2.6.1. Định nghĩa

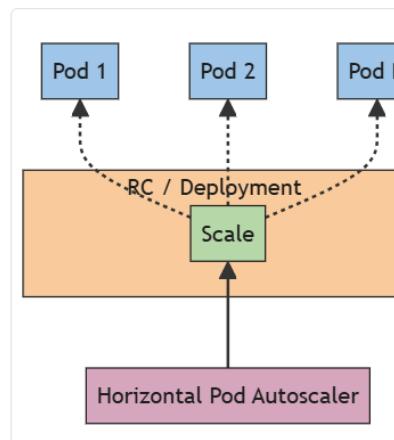
2.6.1.1. Autoscaling

Autoscaling là một phương pháp được sử dụng trong điện toán đám mây để tự động điều chỉnh lượng tài nguyên tính toán trong cụm máy chủ, thường được đo bằng số lượng máy chủ đang hoạt động và tự động dựa trên tải trong cụm máy chủ. [6]

2.6.1.2. Horizontal Pod Autoscaling

Trong Kubernetes, HorizontalPodAutoscaler tự động cập nhật tài nguyên khối lượng công việc (chẳng hạn như Triển khai hoặc StatefulSet), nhằm mục đích tự động điều chỉnh quy mô khối lượng công việc để phù hợp với nhu cầu.

Horizontal Scaling là phản ứng với tải tăng lên sẽ triển khai nhiều Pod hơn. Khác với Vertical Scaling, Horizontal Scaling chỉ định nhiều tài nguyên hơn (bộ nhớ hoặc CPU) cho các Pod đang chạy cho khối lượng công việc. [7]



Hình 2.7: Cách hoạt động của Horizontal Pod Autoscalle

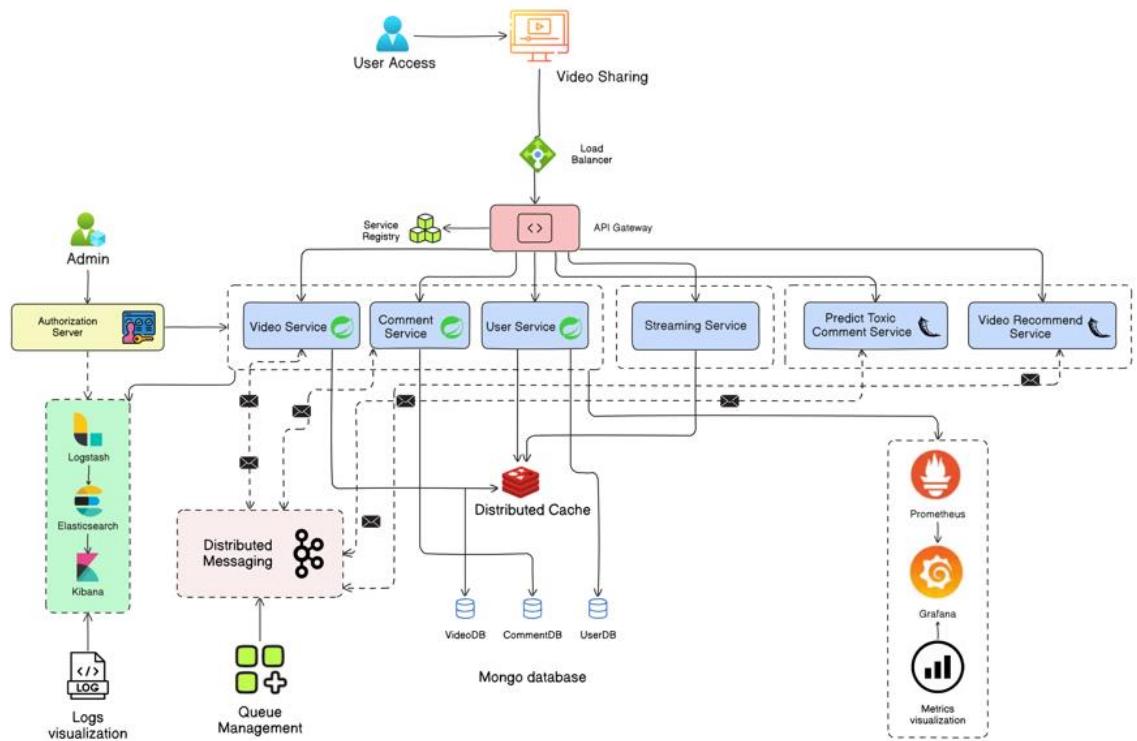
2.6.1.3. KEDA

KEDA là một công cụ chia tỷ lệ tự động theo sự kiện dựa trên Kubernetes. Với KEDA, có thể điều chỉnh quy mô của bất kỳ vùng chứa nào trong Kubernetes dựa trên số lượng sự kiện cần được xử lý.

Bên cạnh đó, KEDA là một thành phần nhẹ và đơn mục đích có thể được thêm vào bất kỳ cụm Kubernetes nào. KEDA hoạt động cùng với các thành phần Kubernetes tiêu chuẩn như Horizontal Pod Autoscaler và có thể mở rộng chức năng mà không cần ghi đè hoặc sao chép. Với KEDA, có thể ánh xạ rõ ràng các ứng dụng bạn muốn sử dụng quy mô theo sự kiện, trong khi các ứng dụng khác vẫn tiếp tục hoạt động. Điều này làm cho KEDA trở thành một lựa chọn linh hoạt và an toàn để chạy cùng với bất kỳ ứng dụng hoặc khung Kubernetes nào khác. [8]

Chương 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1. Kiến trúc ứng dụng

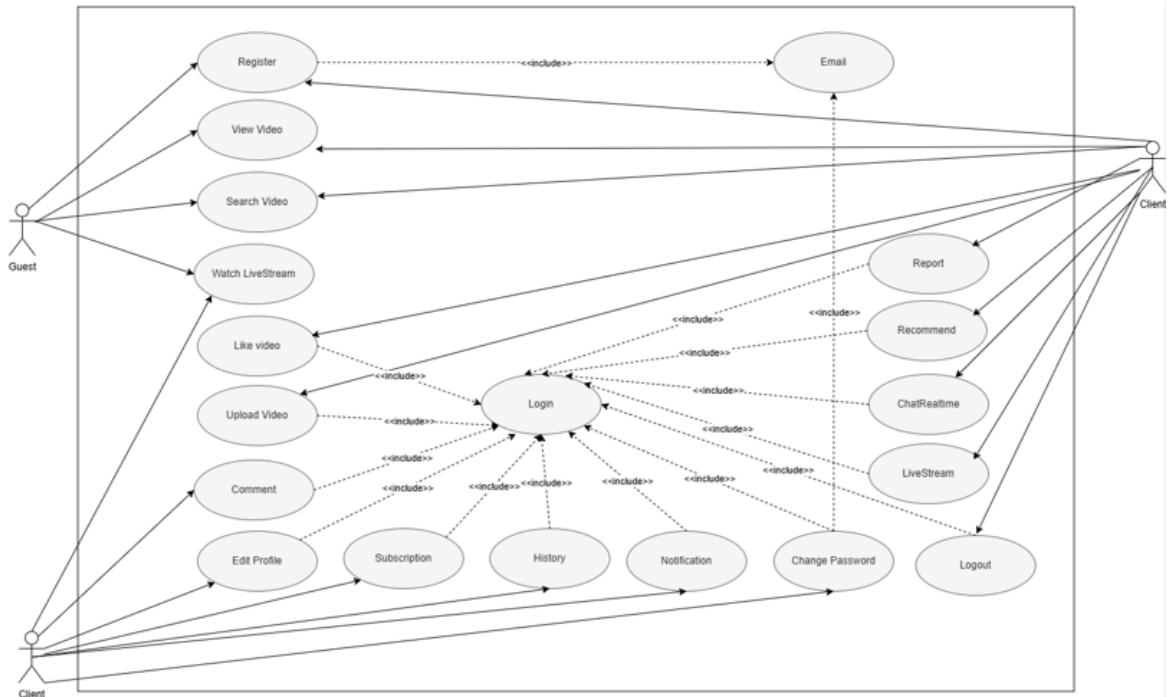


Hình 3.1: Kiến trúc ứng dụng

- Thiết kế kiến trúc microservice gồm các dịch vụ Video Service, Comment Service, User Service, Streaming Service, và các AI service (gợi ý video, phát hiện bình luận tiêu cực).
- Thiết kế cơ sở dữ liệu và phân quyền, xác định cách các dịch vụ sẽ tương tác với nhau, bao gồm việc sử dụng Redis để lưu trữ bộ nhớ đệm và Kafka để xử lý thông điệp giữa các dịch vụ.
- Xác định mô hình AI cho từng dịch vụ: gợi ý video dựa trên hành vi người dùng và phát hiện bình luận tiêu cực bằng phân tích ngữ nghĩa
- ELK là nơi lưu trữ các log hoạt động của Microservices và log của server hoạt động Kubernetes

- Prometheus, Grafana được sử dụng để kiểm soát cluster về tài nguyên phần cứng hoạt động
- Người dùng sẽ truy cập ứng dụng thông qua tên miền frontend và API sẽ làm nhiệm vụ gọi và đưa yêu cầu tới các Service để xử lý các tác vụ từ người dùng

3.2. Sơ đồ Use Cases



Hình 3.2: Sơ đồ Use Cases

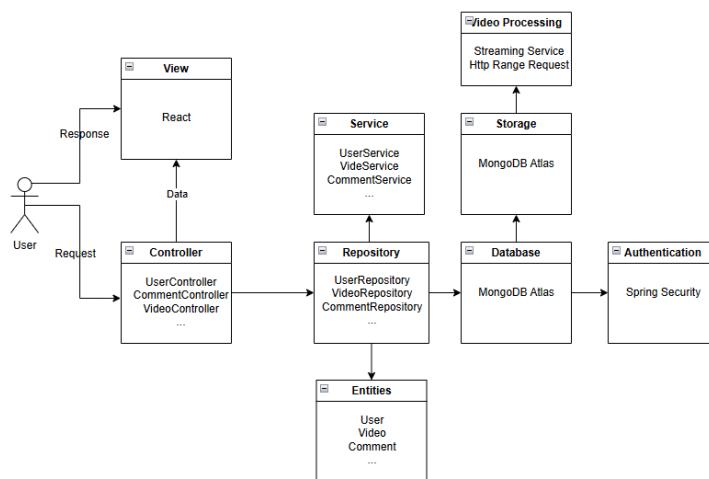
Sơ đồ Use Case của hệ thống Video Sharing mô tả các chức năng chính mà người dùng có thể thực hiện bao gồm các vai trò: Guest (Khách) là người dùng chưa đăng nhập vào hệ thống và Client là người dùng đã đăng nhập vào hệ thống.

- Register (Đăng ký): Cho phép người dùng khách đăng ký tạo tài khoản mới thông qua việc xác thực email người dùng
- Login (Đăng nhập): Cho phép người dùng đăng nhập vào hệ thống thông qua thông tin tài khoản đã đăng ký và đăng nhập là tiền đề để thực hiện các chức năng khác như "Edit Profile", "Change Password", "Logout".

- Các tính năng của Video: Bao gồm có view, search, like, comment và upload cho phép người dùng Client thao tác với các tính năng trên Video của ứng dụng.
- Cùng với đó, còn có 1 số tính năng người dùng như Edit Profile, Change Password, Logout giúp cho người dùng thuận tiện hơn trong việc sử dụng.

Tổng quan về các quan hệ: Các use case "View Video", "Search Video" có thể được thực hiện bởi cả Guest và Client. Các use case còn lại như "Like Video", "Upload Video", "Comment", "Edit Profile", "Change Password", và "Logout" yêu cầu người dùng phải đăng nhập (Client). Use case "Register" bao gồm bước "Verify Email" để xác minh tài khoản người dùng mới.

3.3. Kiến trúc phần mềm



Hình 3.3: Kiến trúc phần mềm

Ứng dụng Video Sharing được thiết kế dựa trên kiến trúc microservice và mô hình MVC (Model-View-Controller) kết hợp với các công nghệ hiện đại như Docker, Kubernetes, ReactJS, Spring Security, và MongoDB Atlas.

3.3.1. View

Thành phần giao diện người dùng (UI) được xây dựng bằng ReactJS. Nó chịu trách nhiệm hiển thị dữ liệu và nhận các tương tác từ người dùng. View là nơi nhận các yêu cầu từ người dùng và hiển thị phản hồi từ hệ thống và gửi các yêu cầu tới Controller để xử lý.

3.3.2. Controller

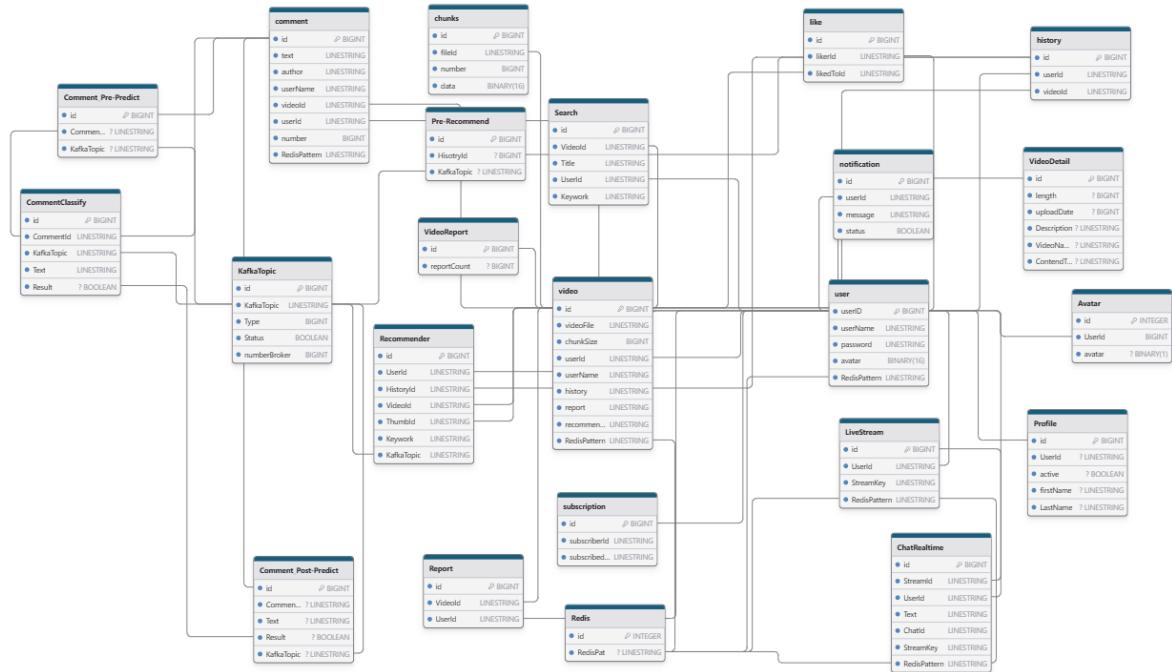
Controller xử lý các yêu cầu từ View. Các Controller chính bao gồm UserController, CommentController, và VideoController đóng vai trò điều phối luồng dữ liệu giữa View và Service. Gọi các dịch vụ tương ứng để xử lý nghiệp vụ và trả về kết quả cho View.

3.3.3. Model

Phần Model bao gồm các thành phần xử lý như sau:

- Service chứa các logic nghiệp vụ của ứng dụng. Các Service chính bao gồm UserService, VideoService, và CommentService thực hiện các nghiệp vụ cụ thể như quản lý người dùng, quản lý video và quản lý bình luận.
- Repository chịu trách nhiệm giao tiếp với cơ sở dữ liệu. Các Repository chính bao gồm UserRepository, VideoRepository, và CommentRepository thực hiện các thao tác CRUD (Create, Read, Update, Delete) với cơ sở dữ liệu MongoDB.
- Entities đại diện cho các đối tượng dữ liệu chính trong hệ thống, bao gồm User, Video, và Comment định nghĩa cấu trúc dữ liệu và mối quan hệ giữa các đối tượng.
- Databases của đề tài sử dụng MongoDB Atlas để quản lý cơ sở dữ liệu NoSQL, hỗ trợ lưu trữ và truy xuất dữ liệu linh hoạt. MongoDB cung cấp khả năng lưu trữ dữ liệu người dùng, video, và bình luận. Cung cấp khả năng mở rộng và bảo mật cao.
- Sử dụng Spring Security để quản lý xác thực và phân quyền người dùng nhằm bảo vệ các endpoint của hệ thống, đảm bảo chỉ người dùng hợp lệ mới có thể truy cập và thực hiện các thao tác.
- Video Processing bao gồm các dịch vụ xử lý video như streaming service và xử lý các yêu cầu HTTP Range Request. Đảm bảo việc phát lại video mượt mà và hiệu quả, hỗ trợ các tính năng như phát lại từng phần của video.

3.4. Database Schema



Hình 3.4: Database Schema

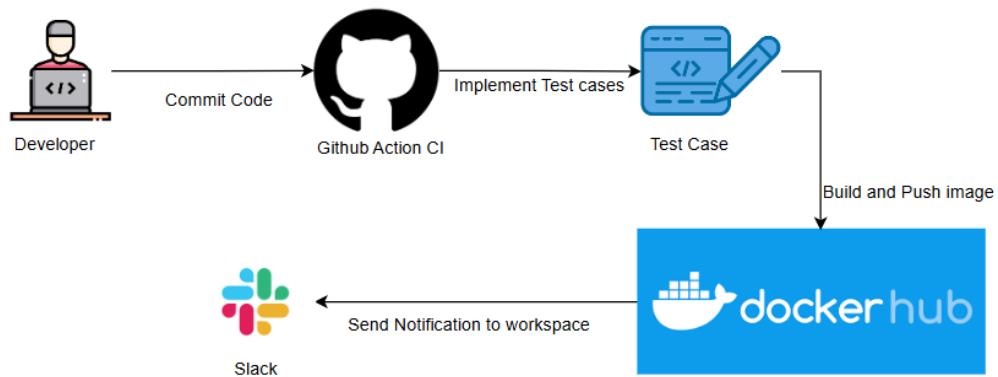
Sơ đồ này thể hiện mối quan hệ giữa các bảng trong cơ sở dữ liệu của một ứng dụng chia sẻ video. Các bảng này kết nối với nhau thông qua các khoá chính và khoá ngoại, thể hiện các mối quan hệ như người dùng, video, bình luận, lượt thích, và thông tin tệp. Mỗi bảng có chức năng và vai trò riêng trong việc quản lý dữ liệu của ứng dụng.

- Bảng user lưu trữ thông tin người dùng, bao gồm tên, mật khẩu, trạng thái hoạt động, tên họ, và avatar.
- Bảng video lưu trữ thông tin về video như tên tệp, độ dài, kích thước, ngày tải lên, mô tả, và loại nội dung.
- Bảng comment lưu trữ các bình luận của người dùng trên các video, bao gồm nội dung bình luận, tác giả, và liên kết đến video tương ứng.
- Bảng like quản lý các lượt thích của người dùng trên các video hoặc bình luận.

- Bảng subscription quản lý thông tin về việc người dùng đăng ký kênh của những người dùng khác.
- Bảng fs.chunks quản lý các phần của video được chia nhỏ để lưu trữ và truyền tải hiệu quả. Các mối quan hệ giữa các bảng được thiết lập qua các khoá ngoại, đảm bảo tính toàn vẹn và nhất quán của dữ liệu.

3.5. Sơ đồ hoạt động qua các giai đoạn triển khai

3.5.1. Giai đoạn Dev



Hình 3.5: Giai đoạn Dev

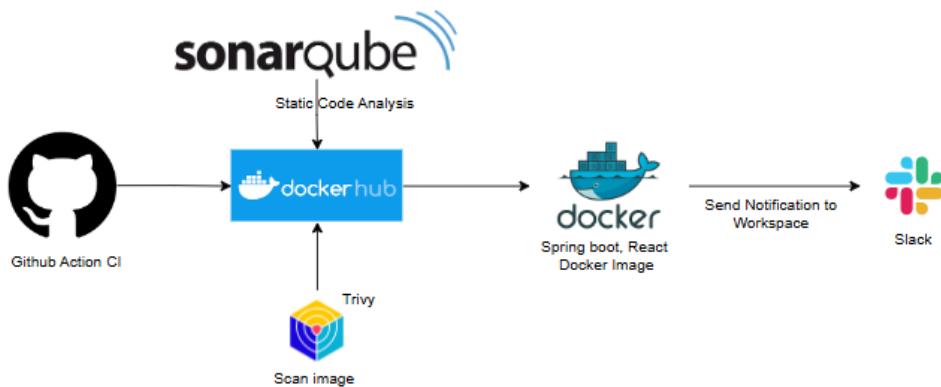
Mô tả quy trình

Giai đoạn Phát triển (Dev) trong quy trình CI/CD bao gồm các bước chính từ việc phát triển mã nguồn, kiểm thử tự động, xây dựng và đẩy image Docker lên Docker Hub, và gửi thông báo đến Slack. Quy trình chi tiết như sau:

- Phát triển mã nguồn: Developer sẽ thực hiện công việc phát triển tính năng mới hoặc sửa lỗi trong mã nguồn của ứng dụng trên môi trường làm việc cá nhân.
- Commit mã nguồn: Khi hoàn thành việc phát triển developer sẽ commit mã nguồn mới lên kho lưu trữ mã nguồn trên GitHub. Đây là bước đầu tiên để bắt đầu quy trình tích hợp liên tục (CI).
- GitHub Actions: Sau khi mã nguồn được commit, GitHub Actions sẽ tự động được kích hoạt. GitHub Actions là một dịch vụ CI/CD của GitHub cho phép tự động hóa quy trình xây dựng, kiểm thử và triển khai ứng dụng.

- GitHub Actions sẽ chạy các pipeline được định nghĩa trong tệp cấu hình (workflow file). Trong pipeline này, các bước thực hiện bao gồm việc kiểm thử và xây dựng image Docker.
- Kiểm thử tự động: Trong pipeline, bước đầu tiên là thực hiện các test case. Các test case được viết sẵn sẽ kiểm tra mã nguồn để đảm bảo rằng không có lỗi phát sinh và các tính năng mới hoạt động đúng như mong đợi.
- Nếu bất kỳ test case nào không thành công, pipeline sẽ dừng lại và thông báo lỗi sẽ được gửi đến nhà phát triển.
- Xây dựng và đẩy image Docker: Nếu tất cả các test case đều thành công, bước tiếp theo là xây dựng image Docker của ứng dụng.
- Image Docker sau đó sẽ được đẩy lên Docker Hub. Docker Hub là một kho lưu trữ công cộng cho phép lưu trữ và phân phối các image Docker.
- Gửi thông báo đến Slack: Sau khi hoàn thành quá trình xây dựng và đẩy image Docker, một thông báo sẽ được gửi đến workspace trên Slack để thông báo cho nhóm phát triển về trạng thái của pipeline.
- Thông báo này bao gồm thông tin về việc kiểm thử và xây dựng image Docker có thành công hay không, giúp nhóm phát triển theo dõi quá trình CI/CD một cách dễ dàng.

3.5.2. Giai đoạn Test



Hình 3.6: Giai đoạn Test

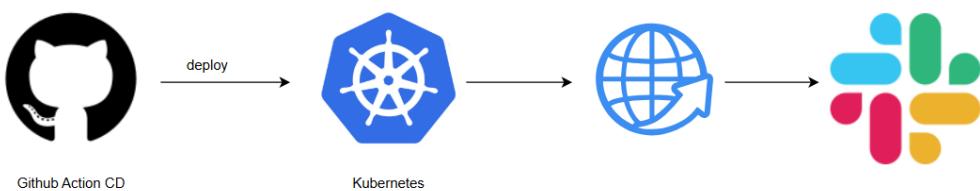
Mô tả quy trình:

Giai đoạn Kiểm thử (Test) trong quy trình CI/CD tập trung vào việc đảm bảo chất lượng mã nguồn và bảo mật cho ứng dụng thông qua kiểm tra tĩnh mã nguồn và quét lỗ hổng bảo mật. Quy trình chi tiết bao gồm các bước sau:

- GitHub Actions CI: Khi mã nguồn được commit lên kho lưu trữ GitHub, GitHub Actions sẽ tự động kích hoạt pipeline CI/CD.
- Pipeline này bao gồm các bước kiểm tra tĩnh mã nguồn, xây dựng image Docker và quét bảo mật image Docker.
- Phân tích mã nguồn tĩnh với SonarQube: Phân tích mã nguồn tĩnh: SonarQube sẽ thực hiện phân tích tĩnh mã nguồn để kiểm tra các lỗi mã, code smells, và các vấn đề về bảo mật tiềm ẩn. SonarQube cung cấp một báo cáo chi tiết về chất lượng mã nguồn, giúp nhà phát triển phát hiện và sửa chữa các vấn đề trước khi triển khai.
- Kết quả: Báo cáo phân tích mã nguồn được lưu trữ và hiển thị trong giao diện của SonarQube. Nếu có lỗi nghiêm trọng, pipeline có thể bị dừng lại để nhà phát triển khắc phục.
- Xây dựng và đẩy image Docker lên Docker Hub: Xây dựng image Docker: Nếu mã nguồn vượt qua kiểm tra tĩnh, GitHub Actions sẽ tiến hành xây dựng image Docker cho ứng dụng.
- Đẩy image lên Docker Hub: Sau khi xây dựng thành công, image Docker sẽ được đẩy lên Docker Hub, nơi lưu trữ và quản lý các image Docker.
- Quét bảo mật image Docker với Trivy: Quét lỗ hổng bảo mật, Trivy sẽ quét image Docker trên Docker Hub để phát hiện các lỗ hổng bảo mật. Trivy kiểm tra các lỗ hổng trong hệ điều hành và các thành phần phụ thuộc của ứng dụng.
- Kết quả: Báo cáo quét bảo mật sẽ chỉ ra các lỗ hổng phát hiện được, phân loại chúng theo mức độ nghiêm trọng. Nếu có lỗ hổng nghiêm trọng, pipeline sẽ bị dừng để đảm bảo an toàn trước khi triển khai.

- Thông báo kết quả: Sau khi hoàn thành tất cả các bước kiểm tra và quét bảo mật, một thông báo sẽ được gửi đến workspace trên Slack để thông báo cho nhóm phát triển về trạng thái của pipeline.
- Nội dung thông báo: Thông báo bao gồm thông tin về kết quả phân tích mã nguồn, kết quả quét bảo mật và trạng thái xây dựng image Docker. Nhóm phát triển sẽ nhận được thông báo kịp thời để có hành động phù hợp nếu có lỗi hoặc lỗi hỏng phát hiện.

3.5.3. Giai đoạn Production



Hình 3.7: Giai đoạn Production

Mô tả quy trình:

Giai đoạn Production là bước cuối cùng trong quy trình CI/CD, nơi ứng dụng được triển khai lên môi trường sản xuất để phục vụ người dùng thực tế. Quy trình chi tiết bao gồm các bước sau:

Triển khai ứng dụng lên Kubernetes:

- GitHub Actions CD: Khi mã nguồn đã sẵn sàng cho sản xuất, GitHub Actions sẽ kích hoạt pipeline CD để triển khai ứng dụng lên GKE. Pipeline này sẽ kéo các tệp manifest của Kubernetes từ kho lưu trữ GitHub và sử dụng chúng để triển khai ứng dụng lên cluster GKE.

Cấu hình domain:

- Mua domain từ bên thứ ba: Một domain sẽ được mua từ nhà cung cấp domain bên thứ ba (ví dụ: GoDaddy, Namecheap, Google Domains). Domain này sẽ được sử dụng để truy cập ứng dụng sản xuất.

- Cấu hình DNS: Sau khi mua domain, cấu hình DNS sẽ được thiết lập để trỏ domain tới địa chỉ IP của Node Port trong cluster. Điều này cho phép người dùng truy cập ứng dụng thông qua domain dễ nhớ thay vì sử dụng địa chỉ IP.

3.6. Thiết kế hệ thống triển khai

3.6.1. Giai đoạn Dev

Build:

- Sử dụng Node.js và npm để cài đặt các gói cần thiết.
- Gọi lệnh npm run build để tạo ra phiên bản tối ưu của ứng dụng.
- Gọi lệnh mvn clean package để tạo ra file jar cho backend

Test:

- Chạy các bài test tự động cho từng thành phần bằng Jest và JUnit (cho React và Spring Boot).
- Kiểm tra tích hợp bằng cách triển khai các service trong môi trường thử nghiệm.

3.6.2. Giai đoạn Test

Trivy:

- Sử dụng Trivy để quét image.

SonarQube:

- Kiểm tra source code và phân tích lỗ hổng bằng Sonarcloud.
- Nếu không phát hiện lỗi thì sẽ có được image Docker an toàn và thông báo tới workspace Slack.

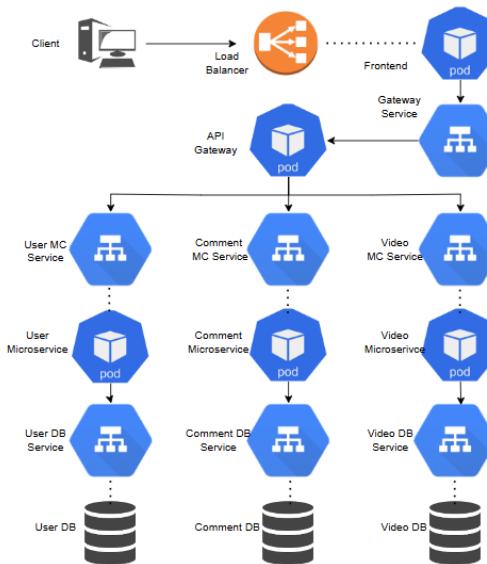
3.6.3. Giai đoạn Production

Kubernetes:

- Tạo cluster Kubernetes trên server Ubuntu bằng kubeadm.

- Những deployments trên kubernetes sẽ được triển khai lại một cách tự động thông qua CD.
- Mua 1 domain từ bên thứ 3, sau đó trỏ ip của ứng dụng vào domain này.

3.7. Tài nguyên triển khai ứng dụng



Hình 3.8: Mô hình triển khai đối với ứng dụng

Mô hình tổng quan biểu hiện khái quát các pod và kết nối sẽ sử dụng trong việc triển khai ứng dụng qua các container được đẩy lên Kubernetes.

Mô hình tài nguyên triển khai đầy đủ sẽ bao gồm 8 pod. Trong đó, 2 pod sử dụng Load Balancer hoặc NodePort là API-Gateway và Front-end (để người dùng có thể truy cập qua trình duyệt), 3 pod sử dụng NodePort (tương ứng với 3 service của ứng dụng) và 3 pod sử dụng ClusterIP (tương ứng với 3 database của 3 service).

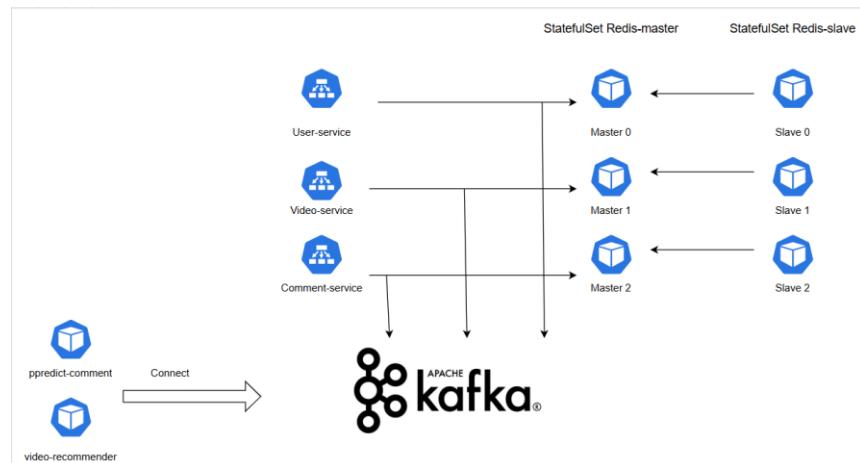
Phân tích mô hình triển khai trên Kubernetes:

- IP của front-end sẽ được công bố và tên miền sẽ có bản ghi “A” trỏ vào IP Load Balancer của pod front-end và hiển thị cho người dùng sử dụng và tương tác với giao diện của ứng dụng Video Sharing.
- Ứng dụng tại front-end sẽ kết nối với IP của API-Gateway để gửi yêu cầu hiển thị dữ liệu, giao diện của các service tương ứng (User Service, Comment

Service, Video Service). Tại API-Gateway sẽ gửi yêu cầu thông qua các service bằng các NodePort tương ứng với từng service (8081 – User Service, 8082 – Comment Service, 8083 – Video Service).

- Tại các service sẽ tiến hành kết nối với các pod database sử dụng ClusterIP để lấy dữ liệu tiến hành xử lý và hiển thị qua giao diện cho người dùng.

3.8. Các tài nguyên triển khai khác



Hình 3.9: Mô hình triển khai 1 số tài nguyên khác

Phân tích mô hình:

- 3 redis master và 3 redis slave sẽ được triển khai dưới dạng 2 StatefulSet tạo thành 6 pod trên server và 3 Microservices sẽ kết nối với 3 pod master để lưu cache.
- 2 mô hình huấn luyện sẽ được triển khai dưới 2 pod và kết nối Kafka KRaft được triển khai trên server để xử lý dữ liệu phân tán

Chương 4. HIỆN THỰC HỆ THỐNG

4.1. Tính năng phát trực tiếp

4.1.1. Cấu hình Nginx

Cấu hình trong http

```
server {
    listen 13000;

    location / {
        root   html;
        index  index.html index.htm;
    }

    location /hls {
        types {
            application/vnd.apple.mpegurl m3u8;
            video/mp2t ts;
        }
        alias /tmp/hls;
        add_header Cache-Control no-cache;

        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type,
Accept';
    }

    location /stat {
        rtmp_stat all;
        rtmp_stat_stylesheet /stat.xsl;
    }

    location /stat.xsl {
        root /usr/share/nginx/html;
    }

    location /stat_json {
        rtmp_stat all;
        rtmp_stat_stylesheet /stat_json.xsl;
    }
    location /stat_json.xsl {
        root /usr/share/nginx/html;
    }
}
```

Thêm mục Server sau vào cuối cấu hình bên trong http{} bao gồm có location của root và định dạng HLS (Chuyển Video từ dạng RTMP sang HLS).

Server sẽ thực hiện phát Live Streaming tại port 13000

Định dạng file HLS dưới dạng m3u8 và thêm alias là thư mục tương ứng lưu video Streaming đó (Phải đảm bảo thư mục tồn tại)

Mục stat để hiển thị các thông tin các streamkey đang livestream để gọi các API tới ứng dụng.

Các phần add_header nhằm tránh các lỗi CORS xuất hiện khi thêm vào code JavaScript của ứng dụng.

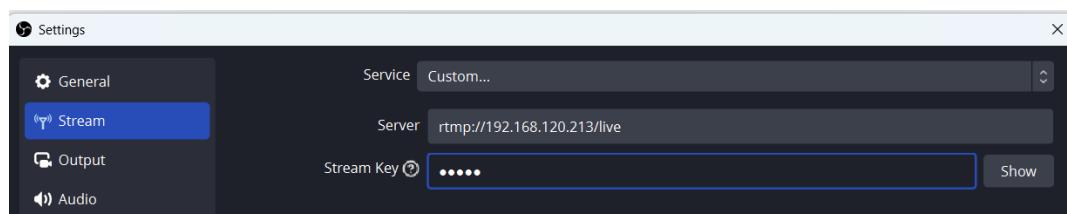
Cấu hình RTMP

```
rtmp {  
    server {  
        listen 1935;  
        chunk_size 4096;  
  
        application live {  
            live on;  
            record off;  
  
            hls on;  
            hls_path /tmp/hls;  
            hls_fragment 2s;  
            hls_playlist_length 10s;  
        }  
    }  
}
```

Thêm cấu hình RTMP module sau vào cuối file config để mở tính năng Live Streaming, xác định thư mục HLS và các cấu hình trình phát liên quan.

Cài đặt OBS Studio

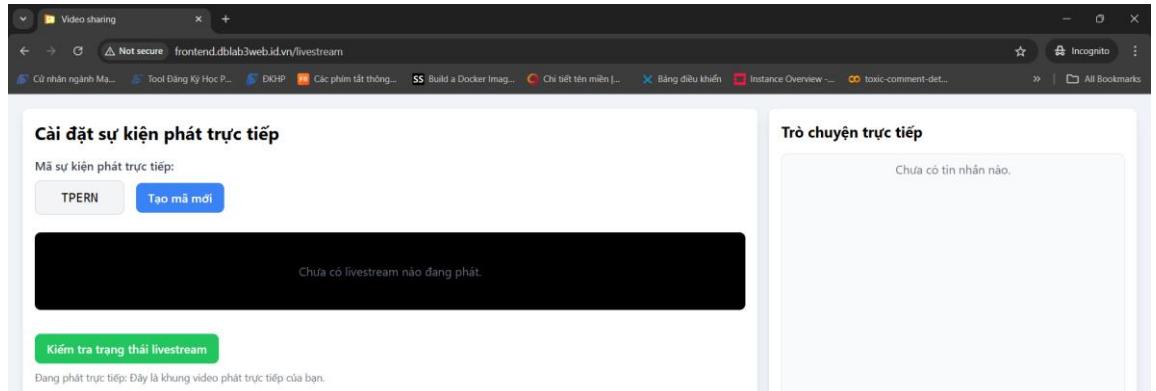
Vào Settings -> Chọn Stream -> Tại Server chọn Custom -> Điền thông tin Server



Hình 4.1: Cấu hình OBS Studio cho Live Streaming

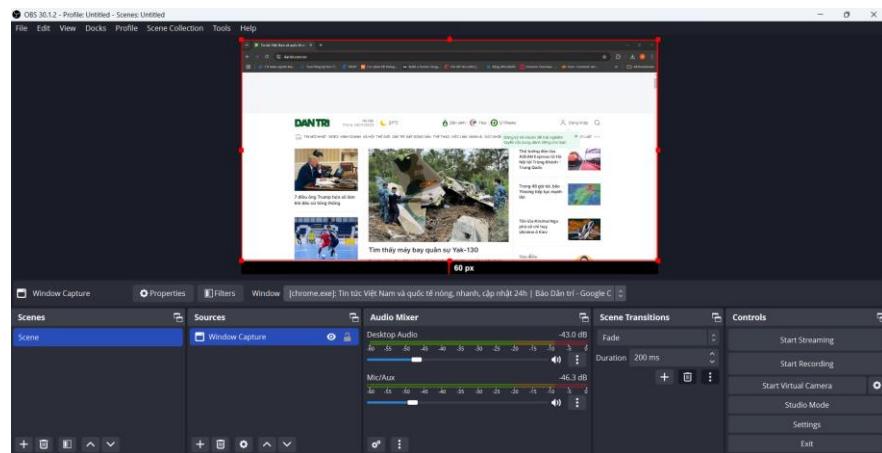
Stream Key sẽ được lấy tại tính năng phát trực tiếp của ứng dụng

4.1.2. Thử nghiệm Live Stream trên ứng dụng

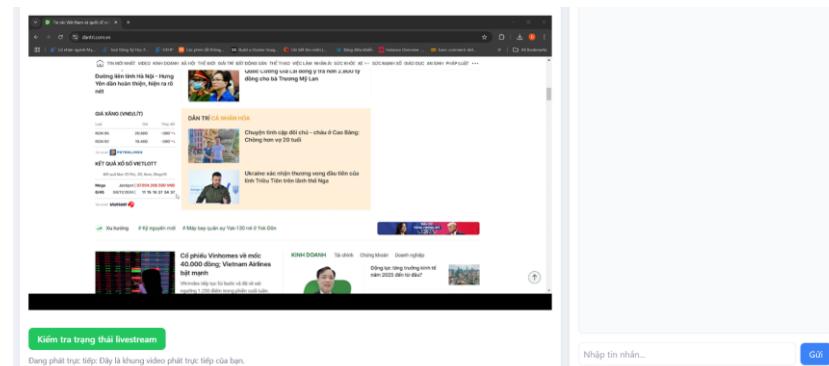


Hình 4.2: Lấy Stream Key từ ứng dụng

Thực hiện nhập Stream Key qua Key được cung cấp tại tính năng Live Stream của ứng dụng. Sau đó cấu hình tại OBS Studio và bắt đầu Streaming.



Hình 4.3: Cấu hình Streaming trên OBS Studio



Hình 4.4: Kiểm tra tính năng Streaming thử nghiệm

Kiểm tra thư mục HLS trên Server để thấy được các file chạy phục vụ cho việc Streaming

```
root@cluster-2:/home/ubuntu# ls /tmp/hls  
TPERN-24.ts  TPERN-25.ts  TPERN-26.ts  TPERN-27.ts  TPERN-28.ts  TPERN-29.ts  TPERN.m3u8
```

Hình 4.5: Kiểm tra thư mục HLS

4.2. Triển khai Redis cho ứng dụng

4.2.1. Triển khai Redis cho User-Service

4.2.1.1. Caching thông tin người dùng

Việc caching thông tin người dùng giúp giảm thiểu số lượng truy vấn tới cơ sở dữ liệu. Cụ thể, Redis sẽ lưu trữ thông tin người dùng bao gồm hồ sơ cá nhân (profile), avatar, và cài đặt cá nhân. Khi có yêu cầu truy xuất thông tin người dùng, hệ thống sẽ kiểm tra trong Redis cache trước. Nếu dữ liệu đã có sẵn trong cache, quá trình truy xuất sẽ diễn ra ngay tại Redis mà không cần truy vấn đến cơ sở dữ liệu nhằm giúp giảm thời gian phản hồi và tiết kiệm tài nguyên.

Luồng hoạt động của quá trình caching thông tin người dùng:

- **Bước 3:** Khi có yêu cầu lấy thông tin người dùng, User-Service sẽ kiểm tra dữ liệu trong Redis thông qua key đại diện cho ID của người dùng.
- **Bước 4:** Nếu thông tin có trong cache, hệ thống sẽ trả về dữ liệu từ Redis mà không cần truy xuất cơ sở dữ liệu chính.
- **Bước 5:** Nếu không tìm thấy dữ liệu trong cache, User-Service sẽ truy vấn MongoDB để lấy thông tin và sau đó lưu lại dữ liệu trong Redis để phục vụ các yêu cầu sau.

4.2.1.2. Cấu hình Redis trong User-Service

Thiết lập Redis Connection Factory:

```

@Bean
public RedisConnectionFactory redisConnectionFactory() {
    RedisStandaloneConfiguration redisConfig = new RedisStandaloneConfiguration();
    redisConfig.setHostName(redisHost);
    redisConfig.setPort(redisPort);
    redisConfig.setPassword(redisPassword);
    return new LettuceConnectionFactory(redisConfig);
}

```

Phương thức `redisConnectionFactory()` thiết lập kết nối đến Redis server, sử dụng `RedisStandaloneConfiguration` để xác định thông tin kết nối bao gồm `redisHost`, `redisPort`, và `redisPassword`. Việc sử dụng `LettuceConnectionFactory` giúp đảm bảo kết nối Redis được quản lý hiệu quả với khả năng chịu tải cao và tối ưu hóa hiệu suất truy cập.

RedisTemplate:

```

@Bean
public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
connectionFactory) {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(connectionFactory);
    template.setKeySerializer(new StringRedisSerializer());
    template.setValueSerializer(new GenericJackson2JsonRedisSerializer());
    return template;
}

```

Phương thức `redisTemplate()` cấu hình một `RedisTemplate`, được sử dụng để tương tác với Redis theo cách thuận tiện hơn. Ở đây, `StringRedisSerializer` được dùng để chuyển đổi các key thành chuỗi, giúp truy vấn Redis dễ dàng hơn, trong khi `GenericJackson2JsonRedisSerializer` được sử dụng để chuyển đổi các giá trị thành JSON. Điều này đảm bảo dữ liệu được lưu trữ dưới dạng JSON, thuận tiện cho việc quản lý và truy xuất thông tin người dùng.

Sử dụng Lettuce:

Lettuce là một client non-blocking được sử dụng để giao tiếp với Redis, hỗ trợ nhiều tính năng như tự động tái kết nối, xử lý tối ưu kết nối mạng và hỗ trợ tốt cho các ứng dụng Spring Boot.

Bằng cách cấu hình Redis trong User-Service, ứng dụng sẽ có thể lưu trữ và truy xuất dữ liệu từ cache một cách dễ dàng, tăng cường hiệu suất và giảm độ trễ khi phản hồi các yêu cầu từ phía người dùng.

4.2.1.3. Cấu trúc cache

Cấu trúc cache trong Redis cho User-Service được thiết kế để dễ dàng truy xuất thông tin. Trong ví dụ dưới đây, Redis lưu trữ thông tin người dùng dưới dạng HashMap, trong đó USER_CACHE là key chính đại diện cho danh sách người dùng, và mỗi mục trong HashMap tương ứng với ID của từng người dùng. Cấu trúc này cho phép truy xuất từng người dùng theo ID với thời gian tối thiểu:

```
private static final String USER_CACHE = "USER_CACHE";

@Cacheable(value = USER_CACHE, key = "#id")
@GetMapping("/listUserById/{id}")
public ResponseEntity listUserById(@PathVariable("id") String id) {
    try {
        AuthUser cachedUser = (AuthUser) redisTemplate.opsForHash().get(USER_CACHE, id);
        if (cachedUser != null) {
            return ResponseEntity.ok(cachedUser);
        }

        AuthUser user = userRepository.findById(id)
            .orElseThrow(() -> new Exception("User not found"));

        redisTemplate.opsForHash().put(USER_CACHE, id, user);
        return ResponseEntity.ok(user);
    } catch (Exception e) {
        return ResponseEntity.internalServerError().body(e.getMessage());
    }
}
```

- Redis Hash: USER_CACHE là key chính chứa thông tin tất cả người dùng dưới dạng một HashMap.
- Key của Redis: Mỗi người dùng sẽ có một key riêng là ID của họ.
- Truy xuất từ Redis: Khi có yêu cầu lấy thông tin người dùng, hệ thống sẽ kiểm tra trong USER_CACHE. Nếu không tìm thấy, dữ liệu sẽ được truy vấn từ cơ sở dữ liệu và lưu lại vào Redis.

4.2.1.4. Hết hạn và xóa cache

Việc quản lý thời gian tồn tại (TTL) của cache là rất quan trọng để đảm bảo rằng dữ liệu trong Redis luôn cập nhật. Trong quá trình triển khai Redis cho User-Service, có thể cấu hình thời gian hết hạn cho các key trong Redis. Điều này giúp hệ thống đảm bảo rằng dữ liệu cũ sẽ tự động bị xóa sau một khoảng thời gian định sẵn, đồng thời tránh tình trạng quá tải bộ nhớ.

- TTL (Time to Live): Cấu hình thời gian tồn tại của mỗi key trong Redis. Khi thời gian tồn tại của key hết hạn, Redis sẽ tự động xóa key đó. Điều này giúp tránh lưu trữ các dữ liệu cũ và không cần thiết trong cache.

4.2.1.5. Kết luận

Việc triển khai Redis cho User-Service giúp tối ưu hóa quá trình truy xuất thông tin người dùng, cải thiện hiệu suất và giảm thiểu tải cho cơ sở dữ liệu. Bằng cách sử dụng cấu trúc cache phân tán cùng với chiến lược hết hạn và xóa cache linh hoạt, Redis đảm bảo rằng dữ liệu được lưu trữ tạm thời luôn cập nhật và sẵn sàng cho các yêu cầu truy xuất. Redis không chỉ mang lại tốc độ truy xuất cao mà còn giúp hệ thống hoạt động ổn định và hiệu quả hơn trong môi trường phân tán.

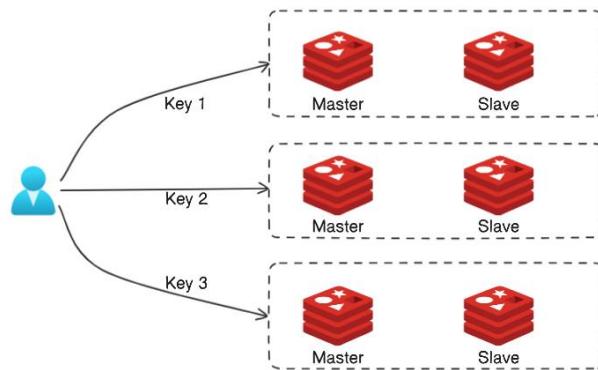
4.2.2. Tối ưu hóa Redis

4.2.2.1. Vấn đề gấp phái

Redis là một giải pháp caching phân tán mạnh mẽ và phổ biến, cung cấp tốc độ truy xuất dữ liệu vượt trội nhờ lưu trữ trực tiếp trên RAM. Tuy nhiên, trong các hệ thống lớn với khối lượng dữ liệu tăng cao, một máy chủ Redis đơn lẻ có thể gặp giới hạn về dung lượng bộ nhớ và xử lý. Để khắc phục vấn đề này, Redis Cluster đã được phát triển nhằm mở rộng khả năng lưu trữ và xử lý thông qua việc phân phối dữ liệu trên nhiều máy chủ khác nhau. Redis Cluster không chỉ cải thiện khả năng mở rộng (scalability), mà còn đảm bảo tính sẵn sàng cao (high availability) cho hệ thống.

4.2.2.2. Redis cluster

Redis Cluster là một cơ chế phân phối dữ liệu, trong đó dữ liệu được chia thành nhiều phần và được lưu trữ trên các node khác nhau. Mỗi node trong Redis Cluster có thể hoạt động độc lập nhưng vẫn liên kết với nhau thành một cụm (cluster), từ đó cho phép dữ liệu được lưu trữ và truy xuất một cách tối ưu. Redis Cluster hỗ trợ khả năng mở rộng ngang (horizontal scalability), có thể dễ dàng thêm hoặc bớt các node mà không ảnh hưởng đến hoạt động của hệ thống.



Hình 4.6: Mô hình Redis Cluster

Redis Cluster sử dụng một phương pháp phân chia dữ liệu gọi là "hash slot". Hệ thống sẽ chia toàn bộ không gian dữ liệu thành 16,384 slot khác nhau. Mỗi key trong Redis sẽ được ánh xạ đến một trong các slot này thông qua hàm băm, và mỗi node trong cụm sẽ quản lý một nhóm các slot. Điều này giúp Redis phân phối dữ liệu đồng đều giữa các node trong cluster và đảm bảo rằng không có node nào bị quá tải về dữ liệu.

Ngoài ra, Redis Cluster còn hỗ trợ khả năng dự phòng (replication). Mỗi node trong cluster có thể có một hoặc nhiều bản sao (replica) để đảm bảo tính sẵn sàng cao trong trường hợp node chính gặp sự cố.

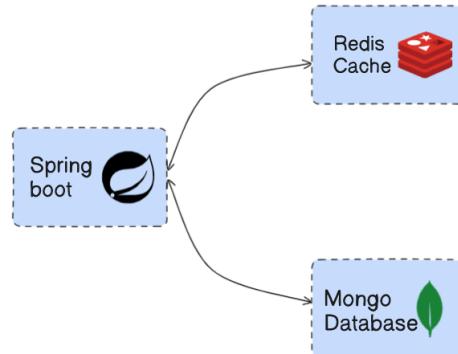
4.2.2.3. Lợi ích của Redis Cluster trong hệ thống phân tán

Redis Cluster mang lại nhiều lợi ích đáng kể, đặc biệt trong các hệ thống microservice phân tán có quy mô lớn:

- Mở rộng quy mô: Redis Cluster cho phép mở rộng hệ thống một cách dễ dàng bằng cách thêm các node mới khi cần thiết. Điều này giúp hệ thống có khả năng xử lý tốt hơn khi khối lượng dữ liệu tăng lên, tránh tình trạng nghẽn cỗ chai.
- Phân phối dữ liệu: Dữ liệu được phân phối đều giữa các node, từ đó giảm thiểu tình trạng quá tải cho một node cụ thể. Điều này cũng giúp tăng tốc độ truy xuất và tối ưu hóa sử dụng tài nguyên hệ thống.
- Tính sẵn sàng cao: Với cơ chế replication, Redis Cluster đảm bảo rằng dữ liệu luôn có sẵn ngay cả khi một node gặp sự cố. Bản sao của dữ liệu trên các node dự phòng sẽ ngay lập tức được sử dụng để thay thế, giúp hệ thống hoạt động liên tục mà không bị gián đoạn.
- Khả năng chịu lỗi: Redis Cluster có khả năng tự động chuyển đổi (failover) khi phát hiện lỗi ở một node. Khi một node chính gặp sự cố, một node dự phòng sẽ tự động thay thế để đảm bảo hoạt động liên tục của hệ thống.

4.2.3. Tính đồng bộ và quản lý dữ liệu cache

4.2.3.1. Đồng bộ cache với các cơ sở dữ liệu



Hình 4.7: Mô hình đồng bộ cơ sở dữ liệu

Đồng bộ hóa dữ liệu giữa Redis và cơ sở dữ liệu là một trong những vấn đề lớn với hệ thống ché cache. Dữ liệu trong cache có thể không đồng bộ với cơ sở dữ liệu nếu không có cơ chế quản lý phù hợp.

1. TTL (Time-to-live) và Expiration

Để đảm bảo tính đồng bộ của dữ liệu, việc sử dụng cơ chế TTL hoặc expiration (hết hạn) cho các key trong Redis giúp xóa hoặc làm mới cache sau một khoảng thời gian nhất định. Khi cache hết hạn, dữ liệu mới sẽ được lấy từ cơ sở dữ liệu và lưu lại trong cache.

```
redisTemplate.expire("VIDEO_CACHE", 30, TimeUnit.MINUTES);
```

4.2.3.2. Kết luận

Việc duy trì tính đồng bộ và quản lý dữ liệu trong Redis là yếu tố quan trọng giúp tối ưu hiệu suất và độ tin cậy của hệ thống. Đồng bộ hóa cache với cơ sở dữ liệu và bảo mật Redis bằng các phương pháp như sử dụng mật khẩu, SSL/TLS, firewall, và giới hạn số lượng kết nối là những bước thiết yếu để bảo vệ dữ liệu và duy trì hiệu suất của hệ thống. Những giải pháp này sẽ giúp đảm bảo hệ thống Redis hoạt động hiệu quả và an toàn trong môi trường phân tán.

4.2.4. Triển khai hệ thống Redis lên K8S

Hệ thống được triển khai bao gồm 3 Redis Master kết nối với 3 Redis Slave tương ứng với việc lưu cache của 3 Microservice.

Tạo ConfigMap

Thực hiện tạo file ConfigMap cấu hình cho tất cả Redis Master và Slave

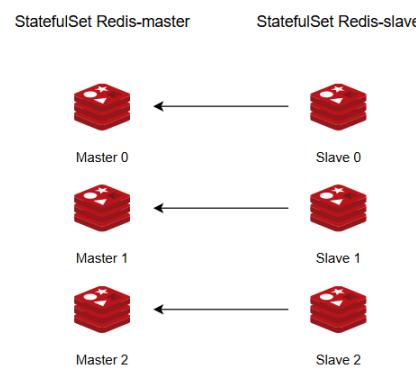
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: redis-config
  namespace: redis
data:
  redis.conf: |
    port 6379
    bind 0.0.0.0
    dir /data
    appendonly yes
    requirepass *****
```

ConfigMap được cấu hình triển khai trên namespace redis nhằm đảm bảo các Master và Slave sẽ hoạt động dưới port 6379. Redis sẽ lắng nghe trên mọi địa chỉ IP

nhằm việc kết nối từ các pod Microservice tới và sẽ yêu cầu nhập password để được kết nối.

Triển khai StatefulSet cho Redis Master và Redis Slave

Các Statefulset được triển khai như mô hình sau:



Hình 4.8: StatefulSet Redis

Hệ thống bao gồm 2 StatefulSet cho Redis Master và Redis Slave, mỗi StatefulSet chứa 3 pod và các pod Slave chỉ connect với pod Master tương ứng.

Kiểm tra các pod được triển khai trên namespace redis

| NAME | READY | STATUS | RESTARTS | AGE |
|--------------------|-------|---------|----------|-------|
| pod/redis-master-0 | 1/1 | Running | 0 | 6d23h |
| pod/redis-master-1 | 1/1 | Running | 0 | 6d23h |
| pod/redis-master-2 | 1/1 | Running | 0 | 6d23h |
| pod/redis-slave-0 | 1/1 | Running | 0 | 6d23h |
| pod/redis-slave-1 | 1/1 | Running | 0 | 6d23h |
| pod/redis-slave-2 | 1/1 | Running | 0 | 6d23h |

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|----------------------|-----------|--------------|-------------|----------------|-------|
| service/redis-master | NodePort | 10.101.84.76 | <none> | 6379:30393/TCP | 6d23h |
| service/redis-slave | ClusterIP | None | <none> | 6379/TCP | 6d23h |

| NAME | READY | AGE |
|-------------------------------|-------|-------|
| statefulset.apps/redis-master | 3/3 | 6d23h |
| statefulset.apps/redis-slave | 3/3 | 6d23h |

Hình 4.9: Pod Redis

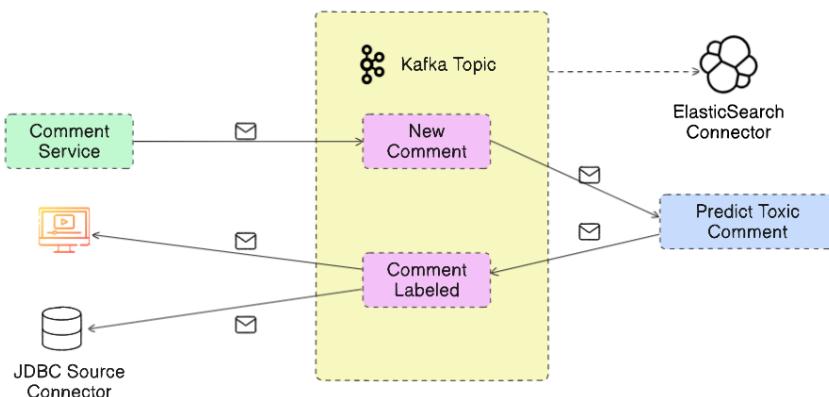
4.3. Triển khai giải pháp xử lý dữ liệu thời gian thực

4.3.1. Kafka trong Predict Toxic Comment

Hệ thống Predict Toxic Comment được thiết kế dưới dạng microservice, trong đó các dịch vụ độc lập hoạt động và tương tác với nhau thông qua các thông điệp do Kafka quản lý. Cấu trúc cơ bản của hệ thống bao gồm các thành phần sau:

- **Comment Service:** Đây là dịch vụ chịu trách nhiệm nhận các bình luận từ người dùng thông qua API và lưu trữ chúng. Sau đó, Comment Service sẽ gửi các bình luận này vào Kafka thông qua một topic có tên là new-comments để chờ phân tích.
- **Kafka Broker:** kafka đóng vai trò là trung gian truyền thông điệp giữa các dịch vụ. Khi một bình luận mới được đăng, Comment Service sẽ gửi dữ liệu bình luận lên kafka, nơi dữ liệu này được ghi vào topic đã chỉ định. Một consumer sẽ được triển khai để liên tục theo dõi và lấy dữ liệu từ topic new-comments.
- **Predict Toxic Comment Service:** Đây là dịch vụ được triển khai để phân tích nội dung của các bình luận và xác định liệu bình luận đó có phải là độc hại (toxic) hay không. Predict Toxic Comment Service lắng nghe dữ liệu từ Kafka và sử dụng mô hình AI đã được huấn luyện để dự đoán kết quả. Khi dịch vụ này nhận được một bình luận từ Kafka, nó sẽ xử lý thông tin và trả về kết quả là "toxic" hoặc "non-toxic".
- **Lưu trữ kết quả:** Sau khi Predict Toxic Comment Service hoàn thành việc phân tích, kết quả sẽ được trả lại cho hệ thống để lưu trữ hoặc thực hiện các hành động tiếp theo, chẳng hạn như cảnh báo người dùng, chặn bình luận hoặc thông báo cho quản trị viên về nội dung độc hại.

4.3.1.2. Triển khai Kafka cho Predict Toxic Comment



Hình 4.10: Cấu trúc Kafka trong việc xử lý bình luận

Trong hệ thống Predict Toxic Comment, Kafka được triển khai với một Kafka broker để quản lý việc truyền tải thông điệp giữa các dịch vụ. Comment Service sẽ gửi các bình luận mới vào topic, nơi dữ liệu này được lưu trữ và chờ xử lý. Predict Toxic Comment Service lắng nghe topic này, lấy các bình luận và sử dụng mô hình AI để dự đoán mức độ độc hại của nội dung. Quá trình này diễn ra liên tục, giúp hệ thống xử lý bình luận thời gian thực một cách hiệu quả và nhanh chóng.

Gửi bình luận đến Kafka: Sau khi bình luận được lưu, phương thức sẽ chuyển bình luận thành chuỗi JSON thông qua thư viện Gson. Chuỗi JSON này sau đó được gửi tới Kafka thông qua đối tượng kafkaTemplate.send(TOPIC, commentJson). Đây là cơ chế giúp gửi bình luận đến một Kafka topic để Predict Toxic Comment Service, có thể lắng nghe và xử lý thông điệp từ topic này.

Phân loại bình luận (Predict Toxic Comment): Sau khi lấy được nội dung bình luận comment_text, hàm sẽ gọi tới chức năng predict_toxicity() – một mô hình AI đã được huấn luyện từ trước – để phân tích và dự đoán liệu bình luận có mang tính chất độc hại (toxic) hay không. Kết quả sẽ được phân loại thành "Toxic" hoặc "Non-toxic", và hiển thị thông báo qua print.

```

Received comment: Mediocre from user userid1 on video video1
1/1 ━━━━━━ 0s 267ms/step
Comment ID 6715b6dbac6fe5186f575b4d is classified as Toxic
Received comment: Hello everybody from user userid1 on video video1
1/1 ━━━━━━ 0s 63ms/step
Comment ID 6715b6f2ac6fe5186f575b4e is classified as Non-toxic
Received comment: Disaster from user userid1 on video video1
1/1 ━━━━━━ 0s 46ms/step
Comment ID 6715b825ac6fe5186f575b4f is classified as Toxic
Received comment: Lost from user userid1 on video video1
1/1 ━━━━━━ 0s 38ms/step
Comment ID 6715b838ac6fe5186f575b50 is classified as Toxic
Received comment: Messy from user userid1 on video video1
1/1 ━━━━━━ 0s 63ms/step
Comment ID 6715b843ac6fe5186f575b51 is classified as Toxic
Received comment: Stubborn from user userid1 on video video1
1/1 ━━━━━━ 0s 38ms/step
Comment ID 6715b84aac6fe5186f575b52 is classified as Toxic

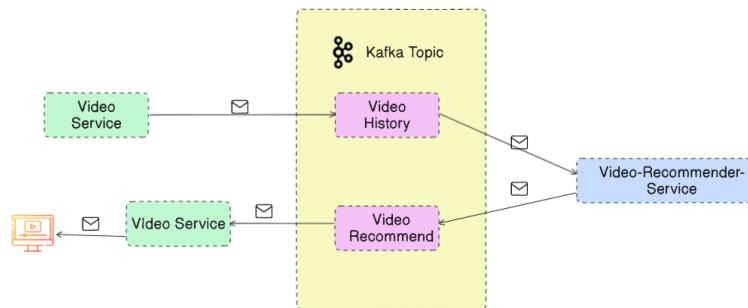
```

Hình 4.11: Phân loại bình luận

Phản hồi kết quả: Kết quả phân loại sẽ được gửi lại qua topic để cho biết bình luận có tính chất độc hại hay không, cùng với thông tin chi tiết về bình luận, người dùng, và video mà nó thuộc về.

4.3.2. Kafka trong Video Recommendation

4.3.2.1. Cấu trúc hệ thống đề xuất video



Hình 4.12: Cấu trúc hệ thống đề xuất Video

Với mỗi lần xem video (trừ lần đầu tạo tài khoản), VideoService sẽ gửi một topic với thông điệp là video đã xem và số lần xem tương ứng, khi đó bên consumer sẽ nhận được và xử lý thông qua mô hình AI và sẽ produce lại topic (chứa kết quả trả về là các đề xuất cho người dùng này), sau khi nhận được topic này thì sẽ so sánh với các dữ liệu trong cơ sở dữ liệu và hiện thị lên giao diện người dùng.

4.4. Triển khai mô hình dự đoán bình luận tiêu cực

4.4.1. Giới thiệu

4.4.1.1. Mục tiêu

Ngày nay, mạng xã hội cũng như các ứng dụng như Youtube, Facebook đang là nơi sinh hoạt của tất cả mọi người. Ở đó, những bình luận tiêu cực với ý muốn tấn công người khác luôn là một vấn đề đáng lo ngại vì những bình luận xấu sẽ ảnh hưởng rất nhiều tới cảm xúc cũng như tinh thần của người khác. Với nạn nhân là các em ở lứa tuổi học sinh, sinh viên thì hậu quả ám ảnh khó lường. Số liệu khảo sát của UNICEF từng cho thấy 21% thanh thiếu niên tham gia khảo sát là nạn nhân của bắt nạt trên MXH tại Việt Nam. Trong những trường hợp nghiêm trọng, “bắt nạt trực tuyến” bắt nguồn từ những bình luận tiêu cực còn đưa nạn nhân tìm đến con đường tự tử. [9]

Do đó, nhóm chúng em quyết định xây dựng mô hình dự đoán “Toxic Comment” này để ngăn chặn những bình luận tiêu cực và tác hại của nó trên ứng dụng của chúng em. Trong đó, chúng em sử dụng mô hình BERT để triển khai. [10]

4.4.1.2. Giới thiệu về BERT

Mô hình ngôn ngữ BERT là một khung học máy nguồn mở để xử lý ngôn ngữ tự nhiên (NLP). BERT được thiết kế để giúp máy tính hiểu được ý nghĩa của ngôn ngữ mơ hồ trong văn bản bằng cách sử dụng văn bản xung quanh để thiết lập ngữ cảnh. Khung BERT đã được huấn luyện trước bằng cách sử dụng văn bản từ Wikipedia và có thể được tinh chỉnh bằng các bộ dữ liệu câu hỏi và câu trả lời.

BERT, viết tắt của Bidirectional Encoder Representations from Transformers, dựa trên máy biến áp, một mô hình deep learning trong đó mọi phần tử đầu ra được kết nối với mọi phần tử đầu vào và trọng số giữa chúng được tính toán linh hoạt dựa trên kết nối của chúng. [11]

4.4.2. Tổng quan

4.4.2.1. Dữ liệu thu thập

Nguồn thu thập

Dữ liệu được sử dụng trong mô hình huấn luyện là dữ liệu phân loại “Toxic Comment” của Jigsaw. Dữ liệu được tham khảo từ Kaggle “Toxic Comment Classification Challenge” [12].

Trong đó, tập dữ liệu được sử dụng trong các bài báo như

- “Evaluating The Effectiveness of Capsule Neural Network in Toxic Comment Classification using Pre-trained BERT Embeddings” của IEEE vào năm 2023 để trình bày nghiên cứu về tiềm năng của CapsNet trong việc phân loại bình luận độc hại bằng cách tận dụng mô hình BERT. [13]
- "Convolutional Neural Networks for Toxic Comment Classification": Bài báo so sánh hiệu quả của mạng nơ-ron tích chập (CNN) với các phương pháp truyền thống như túi từ (Bag-of-Words) kết hợp với các thuật toán phân loại văn bản phỏ biến. [14]
- “JIGSAW MULTILINGUAL TOXIC COMMENT CLASSIFICATION”: Một nghiên cứu khác triển khai mô hình LSTM để phân loại dựa trên độ dài bình luận, tần suất từ ngữ độc hại, và phân tích cảm xúc, giúp xác định mối liên hệ giữa độ dài bình luận và mức độ độc hại [15]
- “Is preprocessing of text really worth your time for toxic comment classification?”: Tập trung vào vai trò của tiền xử lý dữ liệu trong việc cải thiện độ chính xác của mô hình phân loại bình luận trực tuyến, bài viết thử nghiệm trên tập dữ liệu Jigsaw để đánh giá [16]

Đặc điểm dữ liệu

Ngôn ngữ: Tiếng Anh

Có 8 cột chính

- Id: ID của các bình luận
 - Comment_text: Các bình luận dưới dạng chữ
 - Toxic: Nhãn độc hại (gồm 2 giá trị 0 và 1)
 - Severe_toxic: Cực kì độc hại (gồm 2 giá trị 0 và 1)
 - Obscene: Tục tĩu (gồm 2 giá trị 0 và 1)
 - Threat: Đe dọa (gồm 2 giá trị 0 và 1)
 - Insult: Sỉ nhục (gồm 2 giá trị 0 và 1)
- Tổng cộng 159752 dữ liệu

4.4.2.2. Các bước thực hiện

Mô hình huấn luyện được thực hiện tổng quan qua các bước sau:

- **Bước 1:** Cài đặt và khai báo thư viện cần thiết để triển khai mô hình
- **Bước 2:** Xử lý tiền dữ liệu
- **Bước 3:** Kết hợp dữ liệu
- **Bước 4:** Xây dựng mô hình và Token
- **Bước 5:** Chuẩn bị dữ liệu cho huấn luyện
- **Bước 6:** Huấn luyện mô hình
- **Bước 7:** Thử nghiệm

4.4.3. Phân tích mã

4.4.3.1. Khai báo thư viện

Tải và cài đặt thư viện

Thực hiện tải và cài đặt các thư viện cần thiết trong suốt quá trình huấn luyện mô hình. Bao gồm các thư viện như torch, transformer, numpy, pandas, matplotlib, seaborn, wordCloud, pylab.

```

1. Install and Import libraries

① # Install torch and transformers
!pip install torch
!pip install transformers

# Import libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from pylab import rcParams

import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Avoid warnings
import warnings
warnings.filterwarnings('ignore')

```

Hình 4.13: Tải và cài đặt thư viện

4.4.3.2. Thu thập dữ liệu

Khai báo các tập dữ liệu tương ứng

Sử dụng hàm `read_csv` để đọc các file excel từ thư viện pandas (pd) và đưa nó vào các biến tương ứng để thao tác (df)

Tập dữ liệu `train.csv.zip` là tập dữ liệu được tham khảo từ Kaggle bao gồm các bình luận có chứa nhãn nhị phân tương ứng về các trường hợp đánh giá toxic của bình luận đó.

Thực hiện in ra các đầu nhãn dữ liệu để kiểm tra dữ liệu

| | id | comment_text | toxic | \ |
|---|------------------|---|-------|---|
| 0 | 0000997932d777bf | Explanation\nwhy the edits made under my usern... | 0 | |
| 1 | 000103f0d9cfb60f | D'aww! He matches this background colour I'm s... | 0 | |
| 2 | 000113f07ec002fd | Hey man, I'm really not trying to edit war. It... | 0 | |
| 3 | 0001b41b1c6bb37e | "\nMore\nI can't make any real suggestions on ... | 0 | |
| 4 | 0001d958c54c6e35 | You, sir, are my hero. Any chance you remember... | 0 | |
| | | severe_toxic obscene threat insult identity_hate | | |
| 0 | 0 | 0 0 0 0 0 | 0 | |
| 1 | 0 | 0 0 0 0 0 | 0 | |
| 2 | 0 | 0 0 0 0 0 | 0 | |
| 3 | 0 | 0 0 0 0 0 | 0 | |
| 4 | 0 | 0 0 0 0 0 | 0 | |

Hình 4.14: Đầu nhãn dữ liệu

Mô tả dữ liệu

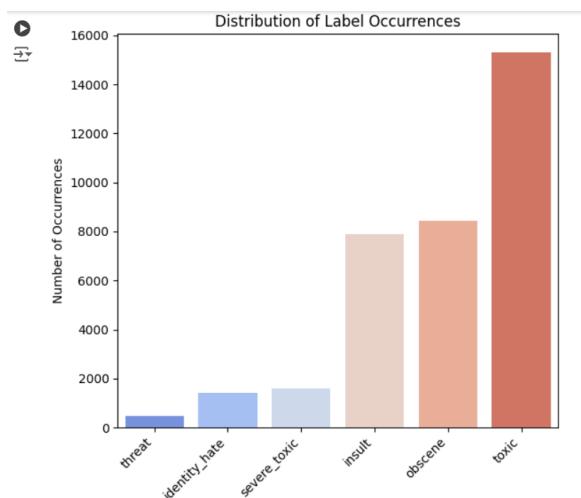
Sử dụng hàm `describe()` để mô tả dữ liệu nhằm hiểu rõ hơn về dữ liệu

| | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|
| count | 159571.000000 | 159571.000000 | 159571.000000 | 159571.000000 | 159571.000000 | 159571.000000 |
| mean | 0.095844 | 0.009996 | 0.052948 | 0.002996 | 0.049364 | 0.008805 |
| std | 0.294379 | 0.099477 | 0.223931 | 0.054650 | 0.216627 | 0.093420 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 75% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

Hình 4.15: Mô tả tập dữ liệu

Tạo biểu đồ thống kê dữ liệu

Tạo biểu đồ với trục x là các nhãn về độc hại và trục y là số lượng đếm được của mỗi nhãn. Biểu đồ thu được:



Hình 4.16: Biểu đồ thống kê dữ liệu

Thông qua biểu đồ có thể thấy được nhãn “toxic” xuất hiện nhiều nhất. Xếp sau đó là 2 nhãn “obscence” và “insult” và ít xuất hiện nhất là nhãn “threat” với chưa tới 2000 dữ liệu.

Hiển thị thống kê qua các giá trị cụ thể

Sử dụng hàm agg để tính các giá trị của các nhãn. Bao gồm giá trị thống kê tổng (sum), trung bình (mean), độ lệch chuẩn (std), giá trị nhỏ nhất (min) và lớn nhất (max) cho từng nhãn. Sau đó, sắp xếp tăng dần theo tổng (ascending dựa theo biến sum).

| | sum | mean | std | min | max |
|---------------|---------|----------|----------|-----|-----|
| threat | 478.0 | 0.002996 | 0.054650 | 0.0 | 1.0 |
| identity_hate | 1405.0 | 0.008805 | 0.093420 | 0.0 | 1.0 |
| severe_toxic | 1595.0 | 0.009996 | 0.099477 | 0.0 | 1.0 |
| insult | 7877.0 | 0.049364 | 0.216627 | 0.0 | 1.0 |
| obscene | 8449.0 | 0.052948 | 0.223931 | 0.0 | 1.0 |
| toxic | 15294.0 | 0.095844 | 0.294379 | 0.0 | 1.0 |

Hình 4.17: Số liệu thống kê các nhãn

Số liệu “sum” được thống kê khớp với biểu đồ được thể hiện ở trên với số lượng nhãn “toxic” được xuất hiện nhiều nhất (15294) và nhãn “threat” được xuất hiện ít nhất (478). Bên cạnh đó các giá trị của các nhãn đều là giá trị nhị phân nên có min bằng 0 và max bằng 1.

Phân chia dữ liệu thành 2 loại

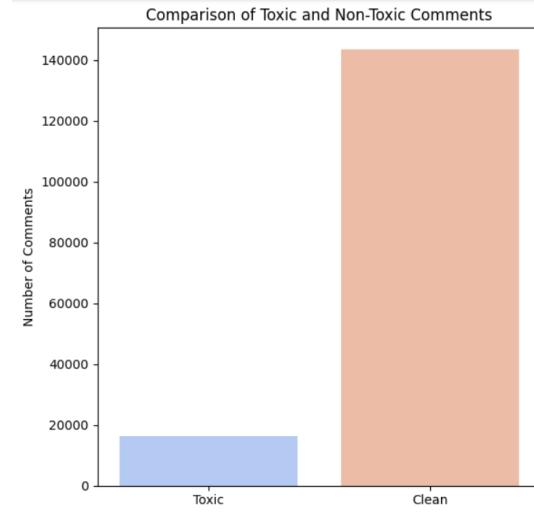
```
# Toxic comment sub data
toxic_comments_training = df[df[column_labels].sum(axis=1) > 0]

# Clean comment sub data
clean_comments_training = df[df[column_labels].sum(axis=1) == 0]
```

Phân chia dữ liệu thành 2 tập với 1 tập bao gồm các dữ liệu “độc hại” (Là các dữ liệu có ít nhất 1 cột có giá trị bằng 1), 1 tập gồm các bình luận “sạch” (Tất cả nhãn dữ liệu bằng 0). Kết quả thu được bao gồm tổng cộng 16225 dữ liệu bình luận độc hại và 143346 dữ liệu bình luận sạch.

Tạo biểu đồ trực quan hóa 2 tập dữ liệu

Sau khi chia thành 2 tập dữ liệu sạch và độc hại sẽ thực hiện tạo biểu đồ nhằm trực quan hóa sự chênh lệch của 2 tập dữ liệu trên.



Hình 4.18: Biểu đồ trực quan 2 tập dữ liệu

Tạo 2 WordCloud về dữ liệu bình luận sạch và bình luận độc hại

Tạo 2 WordCloud nhằm trực quan hóa dữ liệu cho biết các từ ngữ nào được xuất hiện nhiều nhất trong bình luận sạch và bình luận độc hại. Các bình luận độc hại được biểu diễn dưới màu tối và ngược lại đối với các bình luận sạch.



Hình 4.19: WordCloud cho bình luận độc hại



Hình 4.20: WordCloud cho bình luận sạch

4.4.3.3. Kết hợp dữ liệu

Tạo Dataframe từ dữ liệu kết hợp

```
# Create a Dataframe from combining
clean_sampled = clean_comments_training.sample(n=16225, random_state=42)
dataframe = pd.concat([toxic_comments_training, clean_sampled], ignore_index=True)
dataframe = df.sample(frac=1, random_state=42).reset_index(drop=True)
```

Lấy ngẫu nhiên 1 mẫu gồm 16225 hàng (bằng với số lượng tập dữ liệu bình luận độc hại) từ tập dữ liệu sạch và đảm bảo kết quả mẫu tái lập được với random_state = 42.

Thực hiện kết hợp mẫu dữ liệu sạch thu được và toàn bộ dữ liệu độc hại. Đặt lại mục của Dataframe mới với ignore_index=True

Cuối cùng, xóa trộn dữ liệu và đảm bảo kết quả tái lập được

```
(16225, 8)
(16225, 8)
(159571, 8)
```

Hình 4.21: Các tập dữ liệu từ dataframe

Sau đó, kiểm tra lại tập dữ liệu Dataframe nhằm đảm bảo dữ liệu vẫn còn đầy đủ sau khi kết hợp

```
# Checking Data
print(dataframe.info)

<bound method DataFrame.info of
 0    7ca72b5b9c688e9e  Geez, are you forgetful! We've already discuss...
 1    c03f72fd87bf54f  carloca RFA \n\nThanks for your support on my ...
 2    9e5b0e8fc1ff2e84  "\n\n Birthday \n\nNo worries, it's what I do ...
 3    5332799e706665a6  Pseudoscience category? \n\nI'm assuming that ...
 4    dfa7df0fb4366680  (and if such phrase exists, it would be provided...
 ...
159566  811ed72c51830f42  REDIRECT Talk:John Loveday (experimental physi...
159567  2acc7c7d0386401f  Back it up. Post the line here with the refere...
159568  c1f95b89050a9e4  I won't stop that. Sometimes Germanic equals G...
159569  32eb8decfe1d66f0  "\n\n British Bands? \n\nI think you've mista...
159570  8c6c5e4228fb6ba8  You are WRONG. \n\nJustin Thompson is mentioned...
```

| | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|--------|-------|--------------|---------|--------|--------|---------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... |
| 159566 | 0 | 0 | 0 | 0 | 0 | 0 |
| 159567 | 0 | 0 | 0 | 0 | 0 | 0 |
| 159568 | 1 | 0 | 0 | 0 | 0 | 0 |
| 159569 | 0 | 0 | 0 | 0 | 0 | 0 |
| 159570 | 0 | 0 | 0 | 0 | 0 | 0 |

[159571 rows x 8 columns]

Hình 4.22: Kiểm tra Dataframe

Chia dữ liệu thành các tập huấn luyện và tập kiểm tra

```

train_data, test_data = train_test_split(dataframe, test_size=0.25, random_state=42)

train_texts = train_data['comment_text']
test_texts = test_data['comment_text']
train_labels = train_data.iloc[:, 2:]
test_labels = test_data.iloc[:, 2:]

# Split the test set into validation set
val_data, test_data = train_test_split(test_data, test_size=0.5, random_state=42)

val_texts = val_data['comment_text']
test_texts = test_data['comment_text']
val_labels = val_data.iloc[:, 2:]
test_labels = test_data.iloc[:, 2:]

```

Chia Dataframe thành 2 tập (train_data dùng để huấn luyện mô hình và test_data dùng để kiểm tra mô hình sau huấn luyện)

Trong đó, 25% dữ liệu được đưa vào tập kiểm tra (test_size=0.25) và 75% dữ liệu được đưa vào tập huấn luyện

Sau đó, thực hiện tách phần văn bản chữ và nhãn nhị phân từ tập dữ liệu huấn luyện và dữ liệu kiểm tra (Các nhãn bắt đầu từ cột thứ 2 nên chọn iloc từ 2).

Chia tiếp tập kiểm tra thành tập kiểm tra và tập xác thực với tỉ lệ 50% cho 2 tập dữ liệu và tách nhãn tương tự như 2 tập dữ liệu trên.

Như vậy sẽ bao gồm 3 tập dữ liệu với tỉ lệ 75% cho tập dữ liệu huấn luyện, 12,5% cho tập dữ liệu kiểm tra và 12,5% cho tập dữ liệu xác thực.

4.4.3.4. Xây dựng mô hình và token

Mã hóa và chuyển đổi dữ liệu đầu vào

```

def tokenize_and_encode(tokenizer, comments, labels, max_length=128):
    encoded_comments = list(map(lambda comment: tokenizer.encode_plus(
        comment,
        add_special_tokens=True,
        max_length=max_length,
        pad_to_max_length=True,
        return_attention_mask=True,
        return_tensors='pt'
    ), comments))

```

Thực hiện mã hóa và chuyển đổi dữ liệu đầu vào thành tensor PyTorch để sử dụng cho việc huấn luyện mô hình.

Hàm “encoded_comments” để mã hóa từng bình luận với

- “add_special_tokens=True”: Thêm các token đặc biệt (như [CLS], [SEP]) để phù hợp với mô hình transformer.
- “max_length=max_length”: Độ dài tối đa của chuỗi mã hóa (các token vượt quá max_length sẽ bị loại bỏ).
- “pad_to_max_length=True”: Điền padding ([PAD]) để tất cả các chuỗi có cùng độ dài max_length.
- “return_attention_mask=True”: Trả về attention mask (mask đánh dấu các token thực sự hay là padding).
- “return_tensors='pt'”: Trả về tensor PyTorch.

Sau đó, thực hiện tách input_ids (Mã hóa token của các bình luận) và attention_masks (Mask nhị phân để mô hình biết vị trí nào là token thực sự (1) và padding (0)) từ kết quả mã hóa.

Cuối cùng thực hiện chuyển nhãn labels thành tensors và tra về các tensors cần thiết bao gồm tensors chưa token của các bình luận, tensors chưa attention_masks và tensor chưa nhãn các bình luận.

Khởi tạo Tokenizer

“BertTokenizer”: Lớp token hóa dành riêng cho mô hình BERT. Kết quả thu được bao gồm các file từ việc khởi tạo



Hình 4.23: Khởi tạo Token

Khởi tạo mô hình sử dụng để huấn luyện dữ liệu

```

model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
                                                       num_labels=6)

model.safetensors: 100% 440M/440M [00:02<00:00, 238MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint
You should probably TRAIN this model on a down-stream task to be able to use it for predict

```

Hình 4.24: Khởi tạo mô hình huấn luyện

- “BertForSequenceClassification”: Một lớp được định nghĩa trong thư viện transformers của Hugging Face, tùy chỉnh BERT để sử dụng cho bài toán phân loại.

- Tham số bert-base-uncased là mô hình BERT cơ bản:

Base: Gồm 12 layers, mỗi layer có 768 hidden units, và 12 attention heads (tổng cộng 110 triệu tham số).

Uncased: Mô hình đã được huấn luyện với dữ liệu không phân biệt chữ hoa/thường.

- Tham số num_labels=6:

num_labels: Số lượng lớp đầu ra của mô hình, ứng với số nhãn mà mô hình phải dự đoán.

- Hoạt động bên trong:

BERT Encoder: Lớp chính của BERT xử lý chuỗi đầu vào và tạo ra biểu diễn ngữ nghĩa dưới dạng vector.

Classification Head: Một lớp dense (fully connected) phía trên biểu diễn BERT tương ứng với số nhãn cần dự đoán (num_labels). Sử dụng biểu diễn từ token đặc biệt [CLS] để dự đoán lớp.

Xác định thiết bị sử dụng huấn luyện

Nếu hệ thống có “GPU” (cuda) thì sẽ sử dụng GPU để huấn luyện mô hình vì nó nhanh hơn “CPU” khá nhiều và tối ưu hóa được thời gian chạy mô hình

Cuối cùng, in ra loại phần cứng sử dụng và in ra mô hình sử dụng để huấn luyện

```

  Using device: cuda
  BertForSequenceClassification(
    bert: BertModel(
      embeddings: BertEmbeddings(
        word_embeddings: Embedding(30522, 768, padding_idx=0)
        position_embeddings: Embedding(512, 768)
        token_type_embeddings: Embedding(2, 768)
        LayerNorm: LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        dropout: Dropout(p=0.1, inplace=False)
      )
      encoder: BertEncoder(
        layer: ModuleList(
          (0-11): 12 × BertLayer(
            attention: BertAttention(
              self: BertSdpSelfAttention(
                query: Linear(in_features=768, out_features=768, bias=True)
                key: Linear(in_features=768, out_features=768, bias=True)
                value: Linear(in_features=768, out_features=768, bias=True)
                dropout: Dropout(p=0.1, inplace=False)
              )
              output: BertSelfOutput(
                dense: Linear(in_features=768, out_features=768, bias=True)
                LayerNorm: LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                dropout: Dropout(p=0.1, inplace=False)
              )
            )
            intermediate: BertIntermediate(
              dense: Linear(in_features=768, out_features=3072, bias=True)
              intermediate_act_fn: GELUActivation()
            )
            output: BertOutput(
              dense: Linear(in_features=3072, out_features=768, bias=True)
              LayerNorm: LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              dropout: Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      pooler: BertPooler(
        dense: Linear(in_features=768, out_features=768, bias=True)
        activation: Tanh()
      )
    )
    dropout: Dropout(p=0.1, inplace=False)
    classifier: Linear(in_features=768, out_features=6, bias=True)
  )
)

```

Hình 4.25: Phân cứng và thông tin mô hình BERT

4.4.3.5. Chuẩn bị dữ liệu để huấn luyện

Xử lý dữ liệu

```

# Function process data
def process_dataset(tokenizer, texts, labels):
    input_ids, attention_masks, labels_tensor = tokenize_and_encode(
        tokenizer,
        texts,
        labels.values
    )
    return input_ids, attention_masks, labels_tensor

train_data = process_dataset(tokenizer, train_texts, train_labels)
test_data = process_dataset(tokenizer, test_texts, test_labels)
val_data = process_dataset(tokenizer, val_texts, val_labels)

```

Hàm “process-dataset” sử dụng tokenizer (BERT) để mã hóa văn bản trả về:

- input_ids: Mảng các ID tương ứng với từng token.
- attention_masks: Mặt nạ chú ý để bỏ qua padding.
- labels_tensor: Nhãn dạng tensor.

Xử lý dữ liệu cho cả 3 tập huấn luyện, kiểm tra và xác thực đã phân chia trước đó, sau đó thực hiện phân tách dữ liệu của từng tập và kiểm tra kết quả.

Kết quả thu được tập huấn luyện có 119678 chuỗi với Input Ids và Attention Mask có 128 token và các nhãn có 6 token.

```
Training Comments: (119678,)  
Input Ids      : torch.Size([119678, 128])  
Attention Mask : torch.Size([119678, 128])  
Labels        : torch.Size([119678, 6])
```

Hình 4.26: Kết quả xử lý dữ liệu

Tạo DataLoader cho các tập dữ liệu.

```
batch_size = 32  
  
# Create DataLoader for the training set  
train_loader = DataLoader(  
    dataset=TensorDataset(input_ids, attention_masks, labels),  
    batch_size=batch_size,  
    shuffle=True  
)  
  
# Create DataLoader for the testing set  
test_loader = DataLoader(  
    dataset=TensorDataset(test_input_ids, test_attention_masks, test_labels),  
    batch_size=batch_size,  
    shuffle=False  
)  
  
# Create DataLoader for the validation set  
val_loader = DataLoader(  
    dataset=TensorDataset(val_input_ids, val_attention_masks, val_labels),  
    batch_size=batch_size,  
    shuffle=False  
)
```

DataLoader: Một lớp trong PyTorch hỗ trợ nạp dữ liệu theo lô (batch) trong quá trình huấn luyện, kiểm thử và xác thực. Với công dụng có thể chia dữ liệu thành các lô nhỏ để xử lý hiệu quả hơn. Hỗ trợ shuffle (xáo trộn) để tránh hiện tượng học tuần tự (trong quá trình huấn luyện). Tích hợp dễ dàng vào vòng lặp huấn luyện.

“batch_size=32”: Kích thước của mỗi lô dữ liệu (batch). Ở đây, mỗi batch sẽ chứa 32 mẫu dữ liệu.

“TensorDataset”: Tạo một tập dữ liệu từ các tensor (input_ids, attention_masks, labels). Với input_ids là Tensor chứa token ID, attention_masks là Tensor chứa mặt nạ chú ý, labels là tensor chứa các nhãn,

Thực hiện xáo trộn dữ liệu để tránh hiện tượng tuần tự (chỉ áp dụng cho tập huấn luyện). Kiểm tra batch từ tập huấn luyện ta thu được kết quả sau

```
Batch Size : 32
Each Input ids shape : torch.Size([32, 128])
Input ids :
tensor([
 101, 2024, 2017, 2000, 2425, 2033, 2025, 2000, 11582, 2111,
 2013, 10514, 24128, 1996, 3606, 1029, 2339, 2024, 2017, 2107,
 1044, 22571, 10085, 28884, 1029, 2040, 102, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Corresponding Decoded text:
[CLS] are you to tell me not to warn people from supressing the truth? why are ?
Corresponding Attention Mask :
tensor([
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Corresponding Label: tensor([0., 0., 0., 0., 0., 0., 0.])
```

Hình 4.27: Batch từ tập dữ liệu huấn luyện

4.4.3.6. Huấn luyện mô hình

Cài đặt bộ tối ưu hóa

AdamW: Đây là một biến thể của thuật toán tối ưu Adam, trong đó "W" chỉ việc điều chỉnh trọng số (weight decay). AdamW đã được chứng minh là mang lại hiệu suất tốt hơn trong các mô hình học sâu như BERT.

“model.parameters()”: Truyền tất cả các tham số của mô hình vào optimizer để chúng có thể được tối ưu trong quá trình huấn luyện.

“lr=5e-5”: Đặt tỷ lệ học (learning rate) cho optimizer. Ở đây, 5e-5 là 0.00005, một tỷ lệ học phổ biến khi huấn luyện các mô hình BERT.

Huấn luyện mô hình

Hàm “train-epoch” thực hiện huấn luyện cho 1 epoch bao gồm tính toán và cập nhật tham số của mô hình. Quy trình thực hiện của hàm bao gồm có:

- Đưa mô hình vào chế độ huấn luyện (model.train)
- Với mỗi batch dữ liệu trong train_loader, thực hiện: Tính loss. Cập nhật tham số mô hình qua optimizer.step().

Hàm “validate_epoch” để đánh giá mô hình trên tập xác thực 1 epoch. Quy trình thực hiện bao gồm:

- Đưa mô hình vào chế độ đánh giá (model.eval()).
- Với mỗi batch trong val_loader, tính toán loss mà không cần tính toán gradient (để tiết kiệm bộ nhớ).
- Trả về tổng loss của tập xác thực.

Hàm “train_model” để huấn luyện mô hình trong num_epochs vòng lặp. Quy trình thực hiện bao gồm:

- Gọi train_epoch và validate_epoch cho mỗi epoch.
- Tính loss trung bình cho cả tập huấn luyện và xác thực.
- In ra kết quả cho mỗi epoch.

Sau 3 epoch, kết quả output đánh giá loss giảm dần qua các epoch

```
Epoch 1/3, Training Loss: 0.0509, Validation Loss: 0.0428
Epoch 2/3, Training Loss: 0.0363, Validation Loss: 0.0407
Epoch 3/3, Training Loss: 0.0299, Validation Loss: 0.0439
```

Hình 4.28: Giá trị huấn luyện mô hình

Dánh giá mô hình

```
#Evaluate function
def evaluate_model(model, test_loader, device):
    true_labels, predicted_labels = get_predictions(model, test_loader, device)

    accuracy, precision, recall = calculate_metrics(true_labels, predicted_labels)

    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')

evaluate_model(model, test_loader, device)
```

Hàm “calculate_metrics” để tính toán các chỉ số đánh giá như Accuracy, Precision, và Recall từ các nhãn thực tế và nhãn dự đoán

- “accuracy_score”: Tính độ chính xác (accuracy) bằng cách so sánh số lượng nhãn đúng với tổng số nhãn.

- “precision_score”: Tính precision (độ chính xác) cho bài toán phân loại đa nhãn, sử dụng average='micro' để tính toàn bộ bài toán.
- “recall_score”: Tính recall (độ nhạy) cho bài toán phân loại đa nhãn, cũng với average='micro'.

Hàm “get_predictions” thu thập các nhãn thực tế và nhãn dự đoán từ mô hình sau khi chạy dự đoán trên tập kiểm tra.

- “model.eval()”: Chuyển mô hình sang chế độ đánh giá.
- “torch.sigmoid(outputs.logits)": Áp dụng hàm kích hoạt sigmoid lên logits đầu ra của mô hình để có được xác suất cho mỗi nhãn.
- “(predicted_probs > 0.5).astype(int)": Chuyển xác suất thành nhãn dự đoán bằng cách áp dụng ngưỡng 0.5.

Hàm “evaluate_model” đánh giá mô hình bằng cách tính toán các chỉ số performance như accuracy, precision và recall sau khi dự đoán trên tập kiểm tra.

- Gọi get_predictions để lấy các nhãn thực tế và nhãn dự đoán.
- Tính toán các chỉ số đánh giá bằng cách gọi calculate_metrics.
- In ra kết quả của các chỉ số.

Kết quả thu được bao gồm tỉ lệ chính xác (Accuracy) xấp xỉ 92%, khả năng tránh dự đoán sai (Precision) là 78,39% và khả năng tìm ra các nhãn dương tính (Recall) là 75,79%

| | |
|--|-------------------|
| | Accuracy: 0.9199 |
| | Precision: 0.7839 |
| | Recall: 0.7579 |

Hình 4.29: Kết quả đánh giá mô hình

Lưu mô hình

Lưu mô hình vào thư mục Model_Training để sử dụng vào ứng dụng

```

#Save model
save_dir="Model_Training"
model.save_pretrained(save_dir)
tokenizer.save_pretrained(save_dir)

('Model_Training/tokenizer_config.json',
 'Model_Training/special_tokens_map.json',
 'Model_Training/vocab.txt',
 'Model_Training/added_tokens.json')

```

Hình 4.30: Lưu mô hình

4.4.3.7. Kiểm tra và thử nghiệm

Thực hiện tạo hàm và đưa các bình luận bất kì vào kiểm tra kết quả đánh giá của mô hình. Từ đó đưa vào sử dụng trong ứng dụng để xem kết quả

```

# Define function for prediction
def predict_user_input_v2(input_text, model, tokenizer, device):
    labels_list = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']

    # Tokenize input
    encodings = tokenizer(input_text, padding=True, truncation=True, return_tensors="pt")

    # Prepare input data for the model
    input_ids = encodings['input_ids'].to(device)
    attention_mask = encodings['attention_mask'].to(device)

    # Set model to evaluation mode and make prediction
    model.eval()
    with torch.no_grad():
        logits = model(input_ids, attention_mask=attention_mask).logits
        predictions = torch.sigmoid(logits).cpu().numpy()

    # Convert predictions to binary
    predicted_labels = (predictions > 0.5).astype(int)

    # Map predictions to labels
    result = dict(zip(labels_list, predicted_labels[0]))
    return result

# Test with sample input
input_text = 'Are you stupid!'
result = predict_user_input_v2(input_text, model, tokenizer, device)
print(result)
{'toxic': 1, 'severe_toxic': 0, 'obscene': 1, 'threat': 0, 'insult': 1, 'identity_hate': 0}

```

Hình 4.31: Thử nghiệm trên 1 bình luận

Đưa vào ứng dụng

Truy cập vào ứng dụng và bình luận theo 2 dạng sạch và độc hại để xem cách xử lý của AI sau khi được áp dụng. Đưa bình luận sạch và kiểm tra kết quả

6 lượt xem | Ngày 12, 12 năm 2024

Hi, myname is Bao



Bảo Tạ

What is that?

0 0 Phản hồi

Hình 4.32: Bình luận sạch

6 lượt xem | Ngày 12, 12 năm 2024

Viết bình luận ...



Bảo Tạ

What is that?

0 0 Phản hồi



Bảo Tạ

Hi, myname is Bao

0 0 Phản hồi

Hình 4.33: Kết quả sau xử lý AI

Đưa vào bình luận độc hại và kiểm tra kết quả

Fuck you, what a suck video



Bảo Tạ

What is that?

0 0 Phản hồi



Bảo Tạ

Hi, myname is Bao

0 0 Phản hồi

Hình 4.34: Bình luận độc hại



Bảo Tạ

Hi, myname is Bao

0 0 Phản hồi



Bảo Tạ

**** you, what a **** video

0 0 Phản hồi

Hình 4.35: Kết quả sau xử lý AI

Đưa vào bình luận ký tự bất kì không có ý nghĩa



Hình 4.36: Bình luận kí tự bất kì

Như vậy, mô hình trên sẽ tiến hành nhận diện các từ ngữ độc hại và che nó đi nhằm tránh các người dùng thấy và nâng cao được trải nghiệm sử dụng ứng dụng,

4.5. Triển khai mô hình AI cho hệ thống gợi ý video

4.5.1. Giới thiệu

4.5.1.1. Mục tiêu và yêu cầu của hệ thống gợi ý

Hệ thống gợi ý video trong ứng dụng có nhiệm vụ cung cấp các nội dung phù hợp với sở thích và hành vi của người sử dụng. Mục tiêu chính bao gồm:

- Tăng trải nghiệm người dùng thông qua các gợi ý cá nhân hóa.
- Cải thiện tỷ lệ tương tác như lượt xem, lượt thích và thời gian xem.
- Giảm thiểu thời gian tìm kiếm video phù hợp.

4.5.1.2. Giới thiệu về phương pháp sử dụng AI trong gợi ý video

Phương pháp sử dụng AI dựa trên mô hình học máy (Machine Learning) và mạng nơ-ron (Neural Network) nhằm học từ dữ liệu tương tác của người dùng. Hệ thống gợi ý video sử dụng kỹ thuật học sâu (Deep Learning) để phân tích các đặc trưng của video và hành vi người dùng, từ đó dự đoán các video mà người dùng sẽ quan tâm.

4.5.2. Dữ liệu sử dụng cho việc huấn luyện

Mô hình AI gợi ý video được xây dựng dựa trên bộ dữ liệu lớn để đảm bảo khả năng gợi ý chính xác và phù hợp với sở thích người dùng.

Trang chủ: <https://research.google.com/youtube8m/>

4.5.2.1. Giới thiệu về Youtube-8m dataset

Bộ dữ liệu YouTube-8M là bộ dữ liệu lớn và phổ biến nhất được sử dụng cho ứng dụng về hệ thống gợi ý video. Bộ dữ liệu này được phát triển và công bố bởi Google để cung cấp cho cộng đồng nghiên cứu một nguồn tài nguyên phong phú về video. Được xây dựng từ hàng triệu video YouTube với nội dung đa dạng, bộ dữ liệu YouTube-8M chứa hơn 6 triệu video đã được chú thích với hơn 3,700 nhãn phân loại, bao gồm các thể loại như âm nhạc, thể thao, thời trang, học thuật, và giải trí.

Bộ dữ liệu này được thiết kế dưới dạng các tập tin .tfrecord, một định dạng nén dành riêng cho TensorFlow để xử lý các tập dữ liệu lớn. Mỗi video trong tập dữ liệu bao gồm các thông tin đặc trưng như:

- id: Định danh duy nhất của video.
- labels: Các nhãn thể loại mà video thuộc về.
- mean_audio: Đặc trưng âm thanh trung bình của video.
- mean_rgb: Đặc trưng hình ảnh trung bình của video.

Đây là một bộ dữ liệu đã qua xử lý, trong đó các video không chứa hình ảnh trực tiếp mà là các đặc trưng trích xuất từ khung hình và âm thanh. Điều này cho phép mô hình có thể phân tích và học từ các đặc điểm chính của video mà không cần tải xuống từng video đầy đủ, giúp giảm thiểu nhu cầu về không gian lưu trữ và tài nguyên tính toán.

4.5.2.2. Chuẩn bị dữ liệu

Quá trình chuẩn bị dữ liệu là bước quan trọng nhằm đảm bảo rằng bộ dữ liệu đầu vào có chất lượng tốt, nhất quán, và phù hợp với yêu cầu của mô hình thông qua các bước sau:

Tải dữ liệu:

- Vì YouTube-8M là một bộ dữ liệu lớn nên chỉ chọn một phần nhỏ gồm 1/10 số lượng tệp .tfrecord cho các giai đoạn thử nghiệm ban đầu để tiết kiệm tài

nguyên và thời gian xử lý. Quá trình tải dữ liệu từ Jupyter và giải nén được thực hiện bằng Jupyter nhằm tận dụng tài nguyên tính toán sẵn có.

Tiền xử lý dữ liệu:

- Chuyển đổi định dạng: Tập dữ liệu YouTube-8M ở định dạng .tfrecord, nên phải sử dụng thư viện TensorFlow để đọc và chuyển đổi các tệp này thành các tensors, giúp dễ dàng huấn luyện và đánh giá mô hình.
- Xử lý đặc trưng: Với mỗi video, các đặc trưng mean_rgb và mean_audio được sử dụng để trích xuất thông tin về hình ảnh và âm thanh của video. Các đặc trưng này cần được chuẩn hóa để đảm bảo rằng các giá trị đầu vào của mô hình nằm trong cùng một phạm vi, giúp mô hình dễ dàng học các mối quan hệ giữa dữ liệu.
- Xử lý nhãn (labels): Các nhãn trong bộ dữ liệu thường chứa nhiều thể loại cho mỗi video. Nên phải chuyển đổi nhãn thành dạng one-hot encoding để mô hình có thể dự đoán nhiều thể loại cho mỗi video. Ngoài ra, còn phải lọc những nhãn ít phổ biến để đảm bảo rằng các nhãn chính được đại diện tốt trong tập huấn luyện.

Tách dữ liệu huấn luyện và kiểm tra:

- Để đánh giá hiệu suất của mô hình, chia tập dữ liệu thành hai phần, gồm tập huấn luyện (training set) chiếm 80% và tập kiểm tra (test set) chiếm 20%. Quá trình này đảm bảo rằng mô hình có thể học từ tập huấn luyện và được đánh giá khách quan trên tập kiểm tra.

Lưu trữ và quản lý dữ liệu: Do kích thước lớn, bộ dữ liệu được lưu trữ trong bộ nhớ đệm của Jupyter trong suốt quá trình huấn luyện. Ngoài ra, cũng thực hiện các bước lưu trữ dữ liệu đã tiền xử lý dưới dạng .tfrecord mới nhằm tiết kiệm thời gian khi chạy lại chương trình.

4.5.3. Mô hình AI cho hệ thống gợi ý video

4.5.3.1. Mô hình và kiến trúc mạng nơ-ron

Hệ thống gợi ý video sử dụng mô hình học sâu (Deep Learning) dựa vào kỹ thuật CNN (Convolutional Neural Network) để trích xuất đặc trưng từ dữ liệu video và kết hợp với thông tin lịch sử người dùng để đưa ra các dự đoán về thể loại video mà người dùng có khả năng quan tâm.

Mục tiêu mô hình

Input:

- Lịch sử các thể loại video mà người dùng đã xem. Ví dụ: thể loại action, drama, comedy và số lần xem tương ứng.
- Thông tin về các video trong hệ thống (dữ liệu đầu vào dưới dạng vector hoặc embedding).

Output:

- Dự đoán danh sách top N thể loại video mà người dùng có khả năng xem tiếp theo, dựa trên lịch sử và xu hướng xem hiện tại.

Quy trình xử lý mô hình

Input Data:

- Dữ liệu lịch sử xem của người dùng được biểu diễn dưới dạng vector đặc trưng (feature vector). Ví dụ:
 - [action: 5, comedy: 3, drama: 2]
 - Biểu diễn này giúp mô hình hiểu được tần suất và mối quan tâm của người dùng đối với từng thể loại.

CNN Layer:

- Sử dụng mạng nơ-ron (CNN) để học và trích xuất các đặc trưng từ dữ liệu video. CNN hoạt động hiệu quả trong việc phát hiện các mẫu (patterns) và mối quan hệ giữa các thể loại video dựa trên lịch sử người dùng.

Lý do chọn CNN:

- CNN giúp mô hình học được sự tương quan giữa các thể loại video thông qua phép tích chập.
- Mô hình hóa mối quan hệ giữa thể loại video bằng cách học các đặc trưng ẩn trong dữ liệu đầu vào.

Fully Connected Layers:

- Sau khi các đặc trưng được trích xuất thông qua các lớp tích chập, chúng được đưa qua các lớp Dense (Fully Connected) để tổng hợp thông tin. Các lớp này giúp mô hình học được các biểu diễn mạnh mẽ từ các đặc trưng video và lịch sử người dùng.
- Một lớp Dropout có thể được sử dụng để tránh tình trạng overfitting trong quá trình huấn luyện.

Output Layer:

- Lớp đầu ra là một vector xác suất, trong đó mỗi phần tử biểu diễn khả năng của từng thể loại video sẽ được người dùng quan tâm.
- Vector đầu ra sẽ được sắp xếp và chọn ra Top N thể loại video có giá trị cao nhất để đề xuất cho người dùng.

4.5.4. Triển khai hệ thống và môi trường huấn luyện

4.5.4.1. Môi trường huấn luyện

Môi trường huấn luyện mô hình AI được triển khai trên nền tảng Jupyter Notebook. Jupyter là một công cụ phát triển mạnh mẽ và trực quan, đặc biệt phù hợp cho các dự án nghiên cứu và học máy nhờ khả năng tổ chức mã nguồn, chú thích và biểu đồ trực quan. Trong suốt quá trình huấn luyện, Jupyter Notebook cho phép giám sát tiến trình huấn luyện, điều chỉnh các thông số, và kiểm tra kết quả trong thời gian thực mà không cần chuyển đổi môi trường làm việc.

- Server: <https://jupyter.trinhdat.id.vn/>
- Password: *****

- Ngôn ngữ lập trình: Python 3.
- Thư viện chính: TensorFlow và Keras được sử dụng cho việc xây dựng và huấn luyện mô hình học sâu, với Pandas và NumPy hỗ trợ xử lý dữ liệu, và Matplotlib dùng để trực quan hóa kết quả.

4.5.4.2. Các thông số huấn luyện

Các thông số huấn luyện đóng vai trò quan trọng trong việc điều chỉnh mô hình để đạt hiệu suất cao nhất. Các thông số chính được thiết lập như sau:

- Tốc độ học (learning rate): Tốc độ học là 0.001. Đây là tốc độ điều chỉnh trọng số sau mỗi lần cập nhật. Giá trị này được chọn để đảm bảo rằng mô hình không điều chỉnh quá mạnh (có thể gây mất ổn định) nhưng vẫn đủ để đạt được tối ưu toàn cục sau một số epoch nhất định.
- Hàm mất mát (loss function): Đối với bài toán phân loại đa nhãn, hàm Binary Crossentropy được sử dụng làm hàm mất mát. Hàm này giúp đánh giá độ lệch giữa kết quả dự đoán và nhãn thực tế, từ đó tối ưu hóa mô hình.
- Hàm kích hoạt: Hàm kích hoạt ReLU được sử dụng trong các lớp ẩn để thêm tính phi tuyến vào mô hình. Trong lớp đầu ra, hàm kích hoạt sigmoid được sử dụng để dự đoán xác suất cho mỗi nhãn, cho phép mô hình đưa ra dự đoán đa nhãn.
- Số lớp và số lượng nút: Mô hình có cấu trúc gồm nhiều lớp mạng nơ-ron bao gồm các lớp Dense (fully connected) và Dropout để giảm thiểu tình trạng overfitting. Cụ thể:
 - Lớp đầu vào: Nhận các đặc trưng mean_rgb và mean_audio.
 - Lớp đầu ra: Có số lượng nút tương ứng với số nhãn phân loại và sử dụng hàm kích hoạt sigmoid.

- Các tham số tối ưu hóa khác: Bộ tối ưu hóa Adam được chọn với các tham số mặc định trong TensorFlow, vì đây là bộ tối ưu hóa phổ biến và hiệu quả cho các bài toán học sâu.

4.5.4.3. Tài nguyên huấn luyện

Tài nguyên phần mềm:

- TensorFlow: Phiên bản TensorFlow tối thiểu là 2.x để đảm bảo các chức năng liên quan đến Keras và khả năng xử lý GPU được hỗ trợ đầy đủ.
- Python và thư viện hỗ trợ: Python 3.x được sử dụng cùng với các thư viện hỗ trợ như NumPy, Pandas, và Matplotlib để xử lý dữ liệu, trực quan hóa và phân tích kết quả huấn luyện.
- Jupyter Notebook: Jupyter cung cấp môi trường làm việc thuận tiện với khả năng chia sẻ và giám sát quá trình huấn luyện trong thời gian thực.

Việc triển khai môi trường huấn luyện này không chỉ đảm bảo rằng quá trình huấn luyện diễn ra suôn sẻ mà còn giúp giám sát và tối ưu hóa các thông số, từ đó đảm bảo hiệu suất và chất lượng của mô hình gợi ý video.

4.5.5. Cách tiếp cận và tối ưu hóa huấn luyện

4.5.5.1. Huấn luyện mô hình

Quá trình huấn luyện mô hình gợi ý video sử dụng YouTube-8M dataset, trong đó mỗi bước bao gồm việc điều chỉnh trọng số của các lớp trong mạng nơ-ron để giảm thiểu hàm mất mát (loss function). Dữ liệu đầu vào bao gồm hai đặc trưng chính: mean_rgb và mean_audio, là các vector biểu diễn nội dung hình ảnh và âm thanh của video. Mô hình xử lý các đặc trưng này thông qua nhiều lớp mạng nơ-ron để học cách dự đoán thể loại video.

- Chuẩn bị đầu vào và mục tiêu (target): Dữ liệu được chia thành các tập huấn luyện và tập kiểm tra với tỷ lệ 80-20. Điều này đảm bảo rằng mô hình được đánh giá trên dữ liệu không tham gia vào quá trình huấn luyện.

- Cấu trúc mạng nơ-ron: Mô hình bao gồm các lớp Dense và Dropout xen kẽ, sử dụng hàm kích hoạt ReLU cho các lớp ẩn để tăng khả năng học các đặc trưng phi tuyến, và hàm kích hoạt sigmoid ở lớp đầu ra để dự đoán xác suất cho mỗi nhãn thể loại.
- Thuật toán tối ưu hóa: Bộ tối ưu hóa Adam được sử dụng để cập nhật trọng số của mô hình. Adam là thuật toán phổ biến cho bài toán học sâu nhờ khả năng tự điều chỉnh tốc độ học cho mỗi tham số, giúp đạt được kết quả tối ưu nhanh chóng.
- Huấn luyện qua từng epoch: Mô hình trải qua nhiều epoch, mỗi epoch bao gồm một lần lặp qua toàn bộ dữ liệu huấn luyện. Sau mỗi epoch, các trọng số của mô hình được điều chỉnh dựa trên độ lỗi so với đầu ra mong muốn. Việc huấn luyện dừng lại khi mô hình đạt độ chính xác mong muốn hoặc khi số lượng epoch tối đa đạt tới.

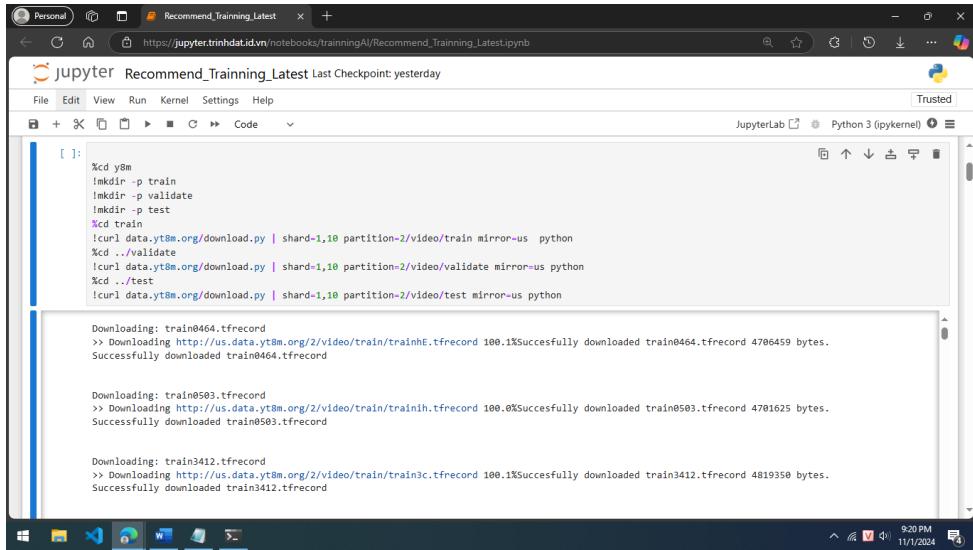
Tải dữ liệu Youtube-8m:

```
%cd y8m
!mkdir -p train
!mkdir -p validate
!mkdir -p test
%cd train
!curl data.yt8m.org/download.py | shard=1,10 partition=2/video/train mirror=us python
%cd ../validate
!curl data.yt8m.org/download.py | shard=1,10 partition=2/video/validate mirror=us python
%cd ../test
!curl data.yt8m.org/download.py | shard=1,10 partition=2/video/test mirror=us python
```

Việc chuẩn bị dữ liệu bằng cách phân chia thành các tập huấn luyện, kiểm tra, và đánh giá đóng vai trò quan trọng trong quá trình huấn luyện mô hình học sâu. Các mục tiêu chính bao gồm:

- Tổ chức cấu trúc dữ liệu: Chuẩn bị các thư mục train, validate, và test để lưu trữ các dữ liệu huấn luyện, kiểm tra, và đánh giá tương ứng. Điều này giúp cho quá trình huấn luyện dễ dàng điều hướng đến đúng dữ liệu và giảm thiểu nhầm lẫn khi triển khai.

- Tải dữ liệu tự động từ nguồn YouTube-8M: Tận dụng tập lệnh download.py từ kho dữ liệu của YouTube-8M, đoạn mã trên cho phép tải dữ liệu từ nguồn chính thức với phân phối hợp lý giữa các tập dữ liệu. Sử dụng cú pháp curl để tải tập lệnh về, và sau đó sử dụng Python để thực thi các câu lệnh nhằm tự động tải và lưu trữ các video mẫu trong từng phân vùng.
- Đảm bảo tính đa dạng và đại diện của dữ liệu: Với các chỉ số shard=1,10, dữ liệu sẽ được phân phối đều qua các tập, giúp mô hình có thể học tốt hơn trên các mẫu đa dạng. Phân vùng dữ liệu hợp lý là một trong những phương pháp giúp đánh giá công bằng hiệu suất của mô hình và tránh tình trạng overfitting khi chỉ huấn luyện trên một tập dữ liệu duy nhất.



The screenshot shows a Jupyter Notebook window titled "Recommend_Training_Latest". The terminal tab (labeled "1") displays the following command sequence:

```

!curl data.yt8m.org/download.py | shard=1,10 partition=2/video/train mirror=us python
!curl data.yt8m.org/download.py | shard=1,10 partition=2/video/validate mirror=us python
!curl data.yt8m.org/download.py | shard=1,10 partition=2/video/test mirror=us python

```

Below the command, the terminal output shows three parallel downloads of TFRecord files:

- Downloading: train0464.tfrecord
 >> Downloading http://us.data.yt8m.org/2/video/train/trainhE.tfrecord 100.0%Successfully downloaded train0464.tfrecord 4706459 bytes.
 Successfully downloaded train0464.tfrecord
- Downloading: train0503.tfrecord
 >> Downloading http://us.data.yt8m.org/2/video/train/trainih.tfrecord 100.0%Successfully downloaded train0503.tfrecord 4701625 bytes.
 Successfully downloaded train0503.tfrecord
- Downloading: train3412.tfrecord
 >> Downloading http://us.data.yt8m.org/2/video/train/train3c.tfrecord 100.1%Successfully downloaded train3412.tfrecord 4819350 bytes.
 Successfully downloaded train3412.tfrecord

Hình 4.37: Kết quả khi tải dataset về

Sau khi thực hiện các lệnh trên, dữ liệu từ kho YouTube-8M sẽ được phân bổ vào ba thư mục tương ứng với các tập:

- Thư mục train: Chứa dữ liệu huấn luyện với các video được lấy mẫu từ phân vùng train của kho dữ liệu. Đây là tập dữ liệu chính dùng để tối ưu hóa các trọng số của mô hình và để mô hình học được các đặc trưng từ dữ liệu.
- Thư mục validate: Chứa dữ liệu kiểm tra, được sử dụng trong quá trình huấn luyện để theo dõi độ chính xác của mô hình và điều chỉnh các tham số nhằm

tránh overfitting. Tập này giúp xác định mức độ hiệu quả của mô hình trên dữ liệu không được huấn luyện trực tiếp.

- Thư mục test: Chứa dữ liệu đánh giá, là tập dữ liệu cuối cùng được sử dụng để kiểm tra hiệu suất của mô hình sau khi đã hoàn thành huấn luyện. Kết quả trên tập dữ liệu này phản ánh khả năng tổng quát hóa của mô hình trên các dữ liệu hoàn toàn mới.

Cài đặt các thư viện cần thiết và chuẩn bị môi trường

Cài đặt thư viện TensorFlow và TensorFlow Datasets:

- TensorFlow là một thư viện mã nguồn mở mạnh mẽ, hỗ trợ việc xây dựng và huấn luyện các mô hình học sâu trên các loại dữ liệu lớn, chẳng hạn như video và hình ảnh. TensorFlow cung cấp các công cụ và API để sử dụng để huấn luyện, đánh giá, và tối ưu hóa mô hình.
- TensorFlow Datasets là một thư viện bổ sung, hỗ trợ truy cập và quản lý các tập dữ liệu lớn và tiêu chuẩn. Việc sử dụng thư viện này giúp giảm bớt thời gian và công sức cần thiết để tải, xử lý, và quản lý dữ liệu một cách thủ công.

Kiểm tra tương thích và thiết lập môi trường huấn luyện:

- Các thư viện TensorFlow và TensorFlow Datasets yêu cầu hệ sinh thái Python và các thư viện phụ trợ đi kèm, do đó việc cài đặt đảm bảo môi trường huấn luyện đã sẵn sàng và đầy đủ chức năng.
- Đảm bảo khả năng tương thích giữa các phiên bản của TensorFlow với hệ điều hành và các phiên bản của thư viện khác cũng là một yêu cầu quan trọng trong việc chuẩn bị môi trường.

Tiền xử lý dữ liệu

```
def load_dataset(tfrecord_dir, temp):
    files = tf.io.gfile.glob(os.path.join(tfrecord_dir, "*tfrecord"))
    raw_dataset = tf.data.TFRecordDataset(files)
    print("Files found:", files)

    feature_description = {
        'id': tf.io.FixedLenFeature([], tf.string),
        'labels': tf.io.VarLenFeature(tf.int64),
        'mean_rgb': tf.io.FixedLenFeature([1024], tf.float32),
        'mean_audio': tf.io.FixedLenFeature([128], tf.float32),
    }

    def _parse_function(proto):
        return tf.io.parse_single_example(proto, feature_description)

    parsed_dataset = raw_dataset.map(_parse_function)
    # print(parsed_dataset)
    print(parsed_dataset)
    return parsed_dataset
```

Mục tiêu của phần này là tạo các tập dữ liệu huấn luyện và kiểm tra bằng cách đọc và giải mã các tập tin TFRecord chứa các video mẫu từ tập dữ liệu YouTube-8M. Các bước tiền xử lý này bao gồm việc tải dữ liệu từ tập tin, xác định cấu trúc của các đặc trưng, và chuẩn hóa dữ liệu, nhằm đảm bảo mô hình nhận được dữ liệu ở định dạng và quy chuẩn phù hợp.

Đọc và chuẩn hóa dữ liệu từ TFRecord:

- Thư viện `tf.data.TFRecordDataset` được sử dụng để đọc các tập tin TFRecord từ thư mục chỉ định. Đây là định dạng tiêu chuẩn và hiệu quả cho lưu trữ và truyền tải các tập dữ liệu lớn.
- Sau khi tải, mỗi tập tin chứa một tập hợp các video, mỗi video là một mẫu dữ liệu với các đặc trưng cụ thể cần được giải mã.

Định nghĩa và phân tích đặc trưng:

- Để giải mã dữ liệu từ các tập tin TFRecord, cấu trúc của các đặc trưng cần được định nghĩa thông qua `feature_description`. Cấu trúc này bao gồm:
 - `mean_rgb`: Đặc trưng trung bình về màu sắc RGB của video, dưới dạng mảng 1024 giá trị (`tf.float32`).

- mean_audio: Đặc trưng trung bình về âm thanh của video, dưới dạng mảng 128 giá trị (tf.float32).

Tạo các tập dữ liệu huấn luyện và kiểm tra:

- Hàm load_dataset giúp tải và phân tích các tập tin TFRecord trong thư mục chỉ định, đồng thời in ra các tập tin đã được tìm thấy để đảm bảo tính toàn vẹn của dữ liệu đầu vào.
- Hàm này trả về các tập dữ liệu đã được giải mã và chuẩn hóa, sẵn sàng cho quá trình huấn luyện và kiểm tra mô hình.

```

feature_description = {
    'id': tf.io.FixedLenFeature([], tf.string),
    'labels': tf.io.VarLenFeature(tf.int64),
    'mean_rgb': tf.io.FixedLenFeature([1024], tf.float32),
    'mean_audio': tf.io.FixedLenFeature([128], tf.float32),
}

def _parse_function(proto):
    return tf.io.parse_single_example(proto, feature_description)

# print(temp)
# print(tfrecord_dir)
parsed_dataset = raw_dataset.map(_parse_function)
print(parsed_dataset)
return parsed_dataset

```

Files found: ['/content/y8m/train/train0276.tfrecord', '/content/y8m/train/train0895.tfrecord', '/content/y8m/train/train0616.tfrecord', '/content/y8m/train/train3749.tfrecord', '/content/y8m/train/train2128.tfrecord', '/content/y8m/train/train2198.tfrecord', '/content/y8m/train/train3601.tfrecord', '/content/y8m/train/train2327.tfrecord', '/content/y8m/train/train1558.tfrecord', '/content/y8m/train/train2149.tfrecord', '/content/y8m/train/train0427.tfrecord', '/content/y8m/train/train1642.tfrecord', '/content/y8m/train/train3063.tfrecord', '/content/y8m/train/train0843.tfrecord', '/content/y8m/train/train696.tfrecord', '/content/y8m/train/train0593.tfrecord', '/content/y8m/train/train1420.tfrecord', '/content/y8m/train/train0777.tfrecord', '/content/y8m/train/train2772.tfrecord', '/content/y8m/train/train0897.tfrecord', '/content/y8m/train/train0043.tfrecord', '/content/y8m/train/train1186.tfrecord', '/content/y8m/train/train3541.tfrecord', '/content/y8m/train/train1117.tfrecord', '/content/y8m/train/train3747.tfrecord', '/content/y8m/train/train0668.tfrecord', '/content/y8m/train/train3652.tfrecord', '/content/y8m/train/train0356.tfrecord', '/content/y8m/train/train0552.tfrecord', '/content/y8m/train/train2401.tfrecord', '/content/y8m/train/train0746.tfrecord', '/content/y8m/train/train3313.tfrecord', '/content/y8m/train/train3519.tfrecord', '/content/y8m/train/train0656.tfrecord']

Hình 4.38: Kết quả sau khi tiền xử lý dữ liệu

Kết quả của phần này là hai tập dữ liệu chính thức: tập dữ liệu huấn luyện (train_dataset) và tập dữ liệu kiểm tra (val_dataset), đã được giải mã và chuyển đổi thành định dạng TensorFlow. Cụ thể:

- Tập dữ liệu huấn luyện (train_dataset): Chứa các video mẫu từ tập huấn luyện, đã được giải mã thành các đặc trưng id, labels, mean_rgb, và mean_audio, giúp cho mô hình học các mẫu đặc trưng của video từ góc độ hình ảnh và âm thanh.

- Tập dữ liệu kiểm tra (val_dataset): Chứa các video mẫu từ tập kiểm tra với cùng cấu trúc đặc trưng, cho phép đánh giá hiệu suất của mô hình trong suốt quá trình huấn luyện.

Chuẩn bị dữ liệu đầu vào cho mô hình

```

def prepare_data(dataset):
    def extract_features_labels(sample):
        features = tf.concat([sample['mean_rgb'], sample['mean_audio']], axis=-1)
        labels = tf.sparse.to_dense(sample['labels']) # Chuyển đổi nhãn từ sparse sang
dense
        labels = tf.reduce_sum(tf.one_hot(labels, depth=4716), axis=0) # Đảm bảo nhãn
là multi-hot encoding
        return features, labels

    dataset = dataset.map(extract_features_labels)
    dataset = dataset.batch(64).prefetch(tf.data.AUTOTUNE) # Chia batch và prefetch để
tăng tốc độ huấn luyện
    return dataset

train_data = prepare_data(train_dataset)
val_data = prepare_data(val_dataset)

```

Kết hợp và chuẩn hóa đặc trưng video:

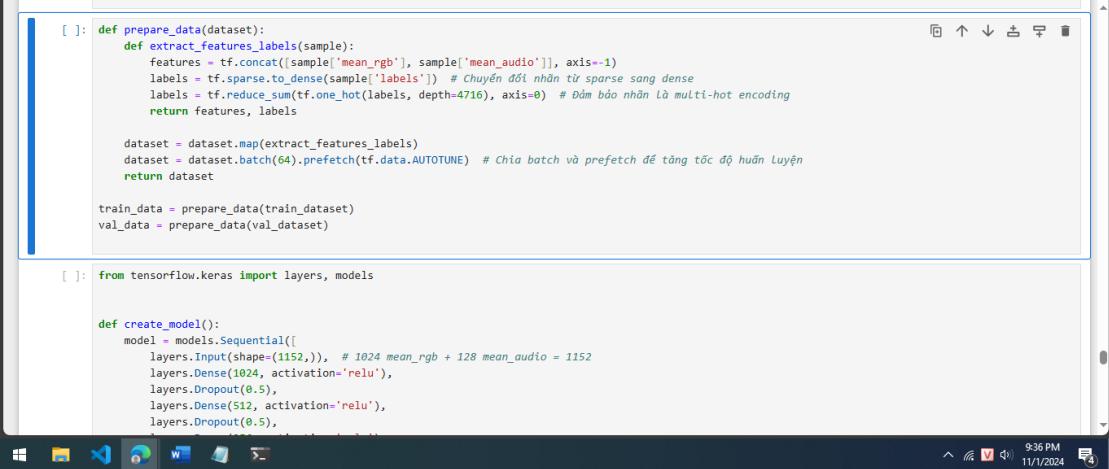
- Mỗi video được biểu diễn qua hai loại đặc trưng chính: mean_rgb (màu sắc trung bình) và mean_audio (âm thanh trung bình), được trích xuất từ các khung hình và đoạn âm thanh của video. Phương pháp tf.concat giúp kết hợp hai loại đặc trưng này thành một vector đặc trưng duy nhất cho mỗi video.
- Việc kết hợp đặc trưng này cung cấp cho mô hình thông tin toàn diện hơn về cả nội dung hình ảnh và âm thanh của video, hỗ trợ việc học các mẫu phức tạp trong dữ liệu.

Chuyển đổi nhãn thành multi-hot encoding:

- Nhãn của mỗi video trong tập dữ liệu YouTube-8M được biểu diễn ở dạng sparse, thể hiện qua chỉ số của các lớp. Các nhãn này cần được chuyển đổi sang định dạng multi-hot encoding để mô hình hiểu được các nhãn của video nào đó thuộc nhiều lớp cùng lúc.

Tối ưu hóa quy trình xử lý dữ liệu với batching và prefetching:

- Để tăng tốc quá trình huấn luyện, dữ liệu được chia thành các batch có kích thước 64 mẫu thông qua phương thức .batch(64). Điều này giúp mô hình xử lý nhiều mẫu cùng lúc, giảm thiểu thời gian và chi phí tính toán.



```
[ ]: def prepare_data(dataset):
    def extract_features_labels(sample):
        features = tf.concat([sample['mean_rgb'], sample['mean_audio']], axis=-1)
        labels = tf.sparse.to_dense(sample['labels']) # Chuyển đổi nhãn từ sparse sang dense
        labels = tf.reduce_sum(tf.one_hot(labels, depth=4716), axis=0) # Đảm bảo nhãn là multi-hot encoding
        return features, labels

    dataset = dataset.map(extract_features_labels)
    dataset = dataset.batch(64).prefetch(tf.data.AUTOTUNE) # Chia batch và prefetch để tăng tốc độ huấn luyện
    return dataset

train_data = prepare_data(train_dataset)
val_data = prepare_data(val_dataset)

[ ]: from tensorflow.keras import layers, models

def create_model():
    model = models.Sequential([
        layers.Input(shape=(1152,)), # 1024 mean_rgb + 128 mean_audio = 1152
        layers.Dense(1024, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),
        ...
    ])

```

The screenshot shows a Jupyter Notebook cell containing Python code for preparing a dataset and creating a neural network model. The code defines a `prepare_data` function that takes a dataset and returns features and labels. It uses TensorFlow operations like `tf.concat`, `tf.sparse.to_dense`, and `tf.reduce_sum` to process the data. The main dataset is mapped through this function, then batched and prefetched. The notebook also imports `layers` and `models` from `tensorflow.keras` and defines a `create_model` function that creates a sequential model with several layers, including input, dense, and dropout layers.

Hình 4.39: Dữ liệu chuẩn bị cho mô hình

Kết quả của quy trình này là hai tập dữ liệu:

- Tập dữ liệu huấn luyện (`train_data`): Đã qua tiền xử lý, chuẩn hóa đặc trưng, chuyển đổi nhãn thành multi-hot encoding, chia thành các batch 64 và tối ưu hóa bằng prefetching. Tập dữ liệu này được sử dụng để huấn luyện mô hình gọi ý video.
- Tập dữ liệu kiểm tra (`val_data`): Đã qua tiền xử lý tương tự như tập huấn luyện, phục vụ cho việc đánh giá hiệu suất của mô hình trong quá trình huấn luyện.

Huấn luyện mô hình CNN cho mô hình gợi ý video

```
def create_cnn_model():
    model = models.Sequential([
        layers.Input(shape=(1152, 1)),
        layers.Conv1D(64, kernel_size=3, activation='relu'),
        layers.MaxPooling1D(pool_size=2),
        layers.Conv1D(128, kernel_size=3, activation='relu'),
        layers.MaxPooling1D(pool_size=2),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5)
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(4716, activation='sigmoid')
    ])

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

model = create_cnn_model()
model.summary()
history = model.fit(train_data, validation_data=val_data, epochs=5)
```

Xây dựng kiến trúc CNN:

- Kiến trúc mô hình được thiết kế theo cấu trúc chuỗi tầng với nhiều lớp tích chập (convolutional layers) và lớp gộp (pooling layers), cho phép trích xuất các đặc trưng quan trọng từ dữ liệu video.
- Đầu vào của mô hình là vector đặc trưng dài 1152, được kết hợp từ mean_rgb và mean_audio, và định dạng đầu vào là (1152, 1). Số chiều này đảm bảo mô hình tiếp nhận đầy đủ thông tin về cả hình ảnh và âm thanh từ video.
- Tầng Flatten được sử dụng để chuyển đổi từ dữ liệu 1D thành dạng vector, cho phép kết nối với các tầng đầu ra.
- Lớp Dense với 512 và 256 đơn vị cùng hàm kích hoạt relu giúp tăng cường khả năng học sâu của mô hình. Thêm Dropout (với tỷ lệ 0.5) vào các tầng này để tránh hiện tượng overfitting và tăng khả năng tổng quát hóa của mô hình.
- Lớp cuối cùng là Dense với 4716 đơn vị và hàm kích hoạt sigmoid, cho phép mô hình dự đoán nhiều nhãn (multi-label) với đầu ra giữa 0 và 1 cho mỗi nhãn.

Biên dịch và cấu hình hàm tối ưu hóa:

- Mô hình được biên dịch với bộ tối ưu hóa adam, đây là một trong những thuật toán tối ưu hóa phổ biến và hiệu quả cho học sâu, đặc biệt cho mạng CNN.

Huấn luyện mô hình:

- Trong mỗi epoch, mô hình sẽ học từ dữ liệu đầu vào, điều chỉnh các trọng số để giảm thiểu sai số dự đoán, và đánh giá kết quả trên tập dữ liệu kiểm tra. Việc này giúp mô hình học dần các mẫu dữ liệu và điều chỉnh cho phù hợp.

| Layer (type) | Output Shape | Param # |
|--------------------------------|------------------|----------|
| conv1d (Conv1D) | (None, 1150, 64) | 256 |
| max_pooling1d (MaxPooling1D) | (None, 575, 64) | 0 |
| conv1d_1 (Conv1D) | (None, 573, 128) | 24704 |
| max_pooling1d_1 (MaxPooling1D) | (None, 286, 128) | 0 |
| flatten (Flatten) | (None, 36608) | 0 |
| dense_4 (Dense) | (None, 512) | 18743808 |
| dropout_2 (Dropout) | (None, 512) | 0 |
| dense_5 (Dense) | (None, 256) | 131328 |
| dropout_3 (Dropout) | (None, 256) | 0 |
| dense_6 (Dense) | (None, 4716) | 1212012 |

Total params: 20112108 (76.72 MB)
Trainable params: 20112108 (76.72 MB)

Hình 4.40: Kết quả sau khi huấn luyện

Kết quả của quá trình này bao gồm:

- Kiến trúc mô hình CNN được xây dựng hoàn chỉnh: Mô hình với các tầng tích chập, gộp, và các tầng kết nối dày đã sẵn sàng để nhận dữ liệu và dự đoán các thể loại video dựa trên đặc trưng của dữ liệu đầu vào. Việc thêm các tầng Dropout giúp giảm thiểu overfitting và tăng khả năng tổng quát của mô hình.
- Biên dịch và huấn luyện: Mô hình CNN sau khi biên dịch đã được huấn luyện thành công trên dữ liệu. Sau mỗi epoch, mô hình cải thiện độ chính xác và giảm lỗi, đồng thời kết quả được theo dõi qua tập kiểm tra. Phân tích kết quả huấn luyện và hiệu suất mô hình

```

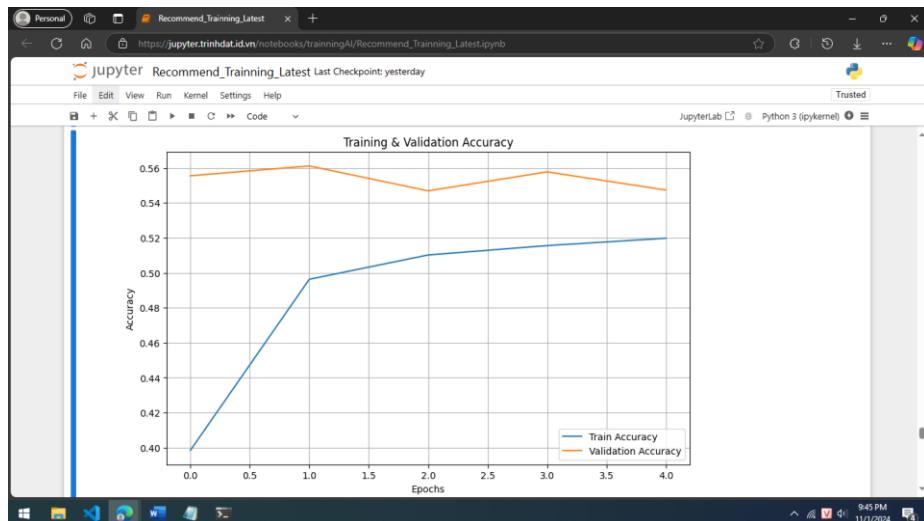
def plot_accuracy(history):
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training & Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_accuracy(history)

```

Phân tích độ chính xác (Accuracy):

- Mục đích của biểu đồ Training & Validation Accuracy là giúp quan sát sự thay đổi trong độ chính xác của mô hình qua các epoch. Độ chính xác đo lường mức độ mô hình dự đoán đúng nhãn mục tiêu, và khi biểu đồ cho thấy đường Validation Accuracy gần giống với Train Accuracy, điều này chỉ ra mô hình đang học hiệu quả mà không bị quá khớp (overfitting).



Hình 4.41: Phân tích độ chính xác

Biểu đồ Training & Validation Accuracy:

- Kết quả biểu đồ cho thấy sự tiến bộ của mô hình qua từng epoch, minh chứng cho việc học các đặc trưng từ dữ liệu huấn luyện và khả năng tổng quát hóa trên dữ liệu kiểm tra.

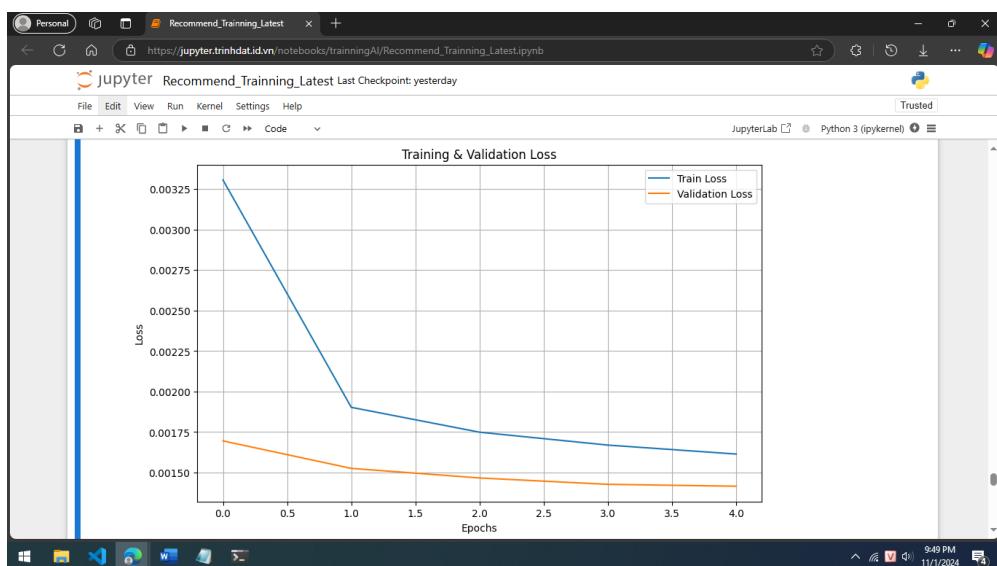
- Đường Validation Accuracy dao động ổn định và gần với Train Accuracy cho thấy mô hình không bị overfitting và có khả năng dự đoán chính xác trên dữ liệu mới.

```
def plot_loss(history):
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training & Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_loss(history)
```

Phân tích hàm mất mát (Loss):

- Biểu đồ Training & Validation Loss cung cấp cái nhìn chi tiết về sự thay đổi của hàm mất mát qua các epoch, cho thấy mức độ lỗi của mô hình trong quá trình dự đoán.
- Mục đích là quan sát đường hàm mất mát giảm dần, biểu thị việc mô hình đang học và điều chỉnh tham số để giảm thiểu sai số. Nếu Train Loss giảm nhưng Validation Loss tăng, điều này có thể là dấu hiệu của overfitting, khi đó, các điều chỉnh như tăng Dropout hoặc giảm số lượng epoch sẽ được cân nhắc.



Hình 4.42: Phân tích sự mất mát dữ liệu

Biểu đồ Training & Validation Loss:

- Hàm mất mát giảm dần qua các epoch là dấu hiệu cho thấy mô hình đang học hiệu quả, giảm dần sai số trong quá trình tối ưu hóa. Nếu Validation Loss dao động trong một phạm vi chấp nhận được, điều này minh chứng cho việc mô hình có khả năng tổng quát hóa tốt trên tập kiểm tra.

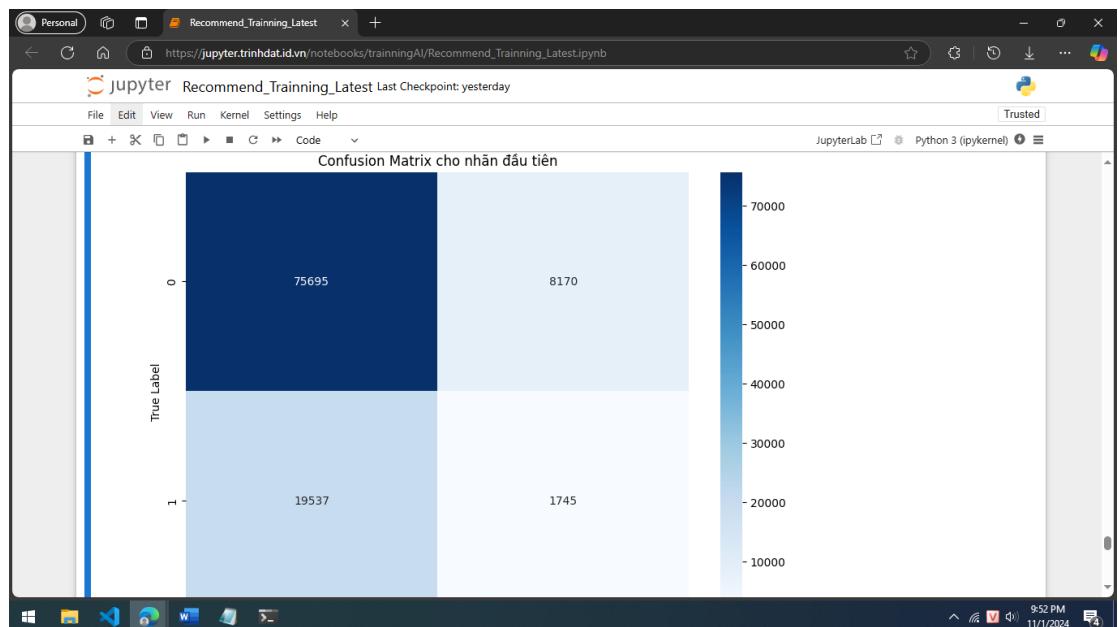
```
y_pred = model.predict(val_data)
y_true = np.concatenate([y for x, y in val_data], axis=0)

cm = confusion_matrix(y_true[:, 0], y_pred[:, 0].round())

plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix cho nhãn đầu tiên')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Đánh giá hiệu suất phân loại:

- Việc xây dựng ma trận nhầm lẫn cung cấp một cái nhìn trực quan về số lần mô hình dự đoán đúng và sai cho từng nhãn cụ thể. Từ đó, có thể xác định rõ ràng nhãn mà mô hình thường xuyên dự đoán sai.
- Đối với bài toán này, chỉ số đầu tiên trong mỗi bộ nhãn được sử dụng để đánh giá, giúp đơn giản hóa và dễ dàng nhận định hiệu suất của mô hình.



Hình 4.43: Đánh Giá Hiệu Suất Mô Hình Trên Tập Validation

Hiệu suất phân loại:

- Các số liệu trên đường chéo chính của ma trận thể hiện các lần mô hình dự đoán đúng nhãn cho nhãn đầu tiên. Nếu các giá trị trên đường chéo chính cao, điều này chứng minh khả năng phân loại chính xác của mô hình với nhãn này.

Nhận định cải tiến:

- Dựa vào Confusion Matrix, nếu số lượng dự đoán sai cho một nhãn cụ thể cao, điều này gợi ý rằng cần tăng cường dữ liệu hoặc tinh chỉnh đặc trưng để cải thiện khả năng nhận diện của mô hình.

```
label_distribution = np.concatenate([y for x, y in train_data], axis=0).sum(axis=0)

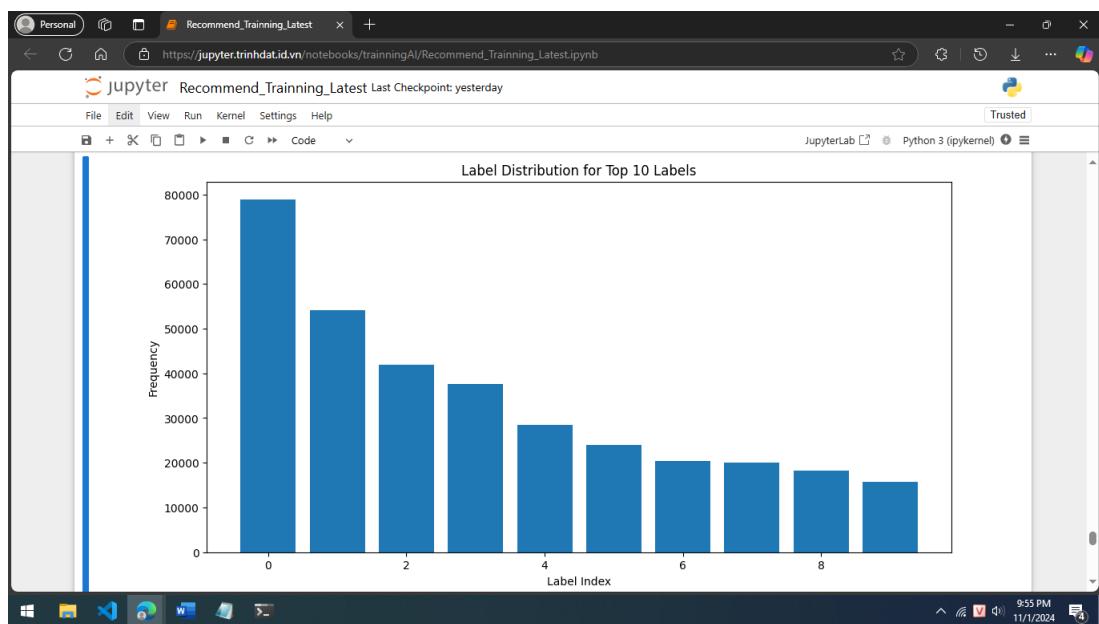
# Vẽ biểu đồ phân phối nhãn
plt.figure(figsize=(12, 6))
plt.bar(range(10), label_distribution[:10]) # Hiển thị 10 nhãn đầu tiên
plt.title('Label Distribution for Top 10 Labels')
plt.xlabel('Label Index')
plt.ylabel('Frequency')
plt.show()
```

Hiệu rõ phân phối nhãn:

- Việc đếm số lần xuất hiện của từng nhãn trong tập huấn luyện cho phép xác định rõ các thể loại video nào có nhiều dữ liệu hơn và thể loại nào ít hơn.

Đánh giá tính cân bằng của tập dữ liệu:

- Tập dữ liệu không cân bằng có thể dẫn đến hiện tượng mô hình bị thiên lệch, nghĩa là nó sẽ dễ dàng phân loại chính xác các nhãn có số lượng dữ liệu lớn trong khi gặp khó khăn với các nhãn có số lượng nhỏ hơn.



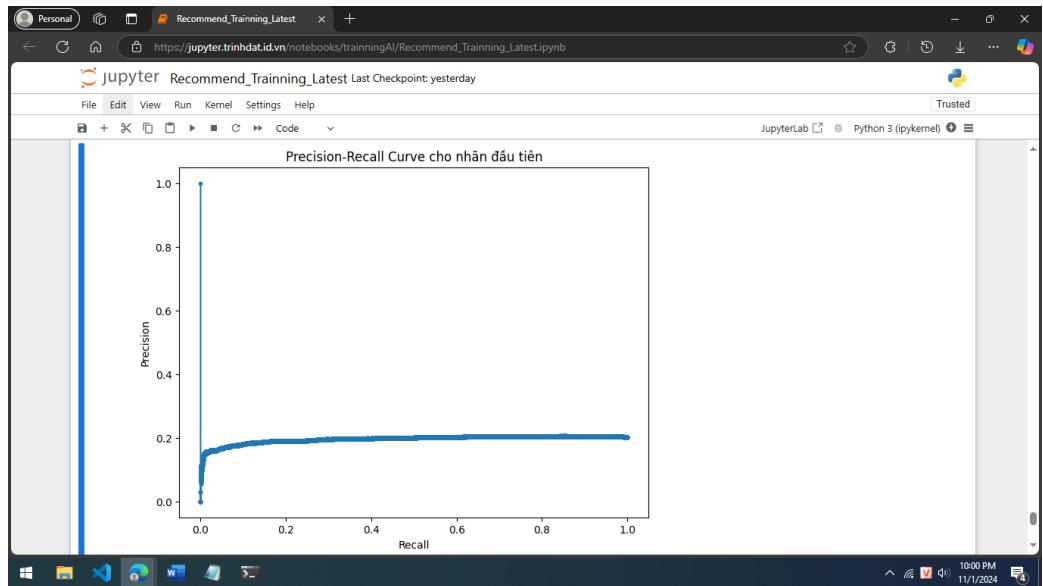
Hình 4.44: Phân tích phân phối nhãn trong tập huấn luyện

Hiện trạng phân phối nhãn:

- Biểu đồ cột cho thấy số lần xuất hiện của từng nhãn, từ đó có thể xác định nhanh chóng các nhãn có số lượng lớn hoặc nhỏ. Những nhãn có tần suất cao cho thấy rằng mô hình sẽ có đủ dữ liệu để học hỏi, trong khi các nhãn có tần suất thấp có thể cần được tăng cường dữ liệu hoặc điều chỉnh cách thức huấn luyện.

Đánh giá hiệu suất phân loại:

- Đường cong Precision-Recall giúp đánh giá khả năng phân loại của mô hình bằng cách cung cấp cái nhìn sâu sắc về mối quan hệ giữa độ chính xác (precision) và độ nhạy (recall) khi thay đổi ngưỡng phân loại.



Hình 4.45: Đường Cong Precision-Recall

Biểu đồ Precision-Recall:

- Biểu đồ cho thấy sự thay đổi của độ chính xác và độ nhạy khi thay đổi ngưỡng phân loại. Một đường cong gần đến góc trên bên trái của biểu đồ cho thấy mô hình có hiệu suất tốt, với độ chính xác và độ nhạy cao.

```
training_summary = {
    'Epoch': list(range(1, len(history.history['loss']) + 1)),
    'Training Loss': history.history['loss'],
    'Validation Loss': history.history['val_loss'],
    'Training Accuracy': history.history['accuracy'],
    'Validation Accuracy': history.history['val_accuracy']
}

df_summary = pd.DataFrame(training_summary)
```

Tổng hợp thông tin huấn luyện:

- Việc thu thập thông tin từ quá trình huấn luyện, bao gồm độ mất mát (loss) và độ chính xác (accuracy) cho cả tập huấn luyện và tập kiểm tra (validation), giúp theo dõi và đánh giá hiệu suất của mô hình trong suốt quá trình huấn luyện.

Dễ dàng phân tích và trực quan hóa:

- Sử dụng bảng dữ liệu (DataFrame) để tổ chức và trình bày thông tin một cách rõ ràng. Bảng dữ liệu cho phép dễ dàng so sánh các chỉ số quan trọng và phát hiện các xu hướng trong quá trình huấn luyện.

```

Personal https://jupyter.trinhdat.id.vn/notebooks/trainingAI/Recommend_Training_Latest.ipynb
jupyter Recommend_Training_Latest Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
+ - Trusted
JupyterLab Python 3 (ipykernel)
df_summary = pd.DataFrame(training_summary)

# Hiển thị bảng
print(df_summary)

Epoch Training Loss Validation Loss Training Accuracy \
0 1 0.003306 0.001695 0.398427
1 2 0.001903 0.001526 0.496280 |
2 3 0.001749 0.001467 0.510183
3 4 0.001670 0.001428 0.515545
4 5 0.001614 0.001416 0.519749

Validation Accuracy
0 0.555498
1 0.56129
2 0.546863
3 0.557724
4 0.547319

[ ]: from sklearn.metrics import precision_score

# Tính precision cho mỗi nhãn
y_pred_binary = (y_pred > 0.5).astype(int)
precision = 1 - 1 / (1 + np.sum(np.abs(y_true - y_pred_binary)))
precision
9:58 PM 11/1/2024

```

Hình 4.46: Tóm tắt kết quả huấn luyện mô hình

Bảng tóm tắt:

- Bảng tóm tắt cho thấy rõ ràng các chỉ số quan trọng về hiệu suất của mô hình qua từng epoch. Thông tin này bao gồm cả độ mỉm mỉm và độ chính xác trên cả tập huấn luyện và tập kiểm tra, cho phép đánh giá khả năng tổng quát của mô hình.

Đánh giá sự phù hợp của mô hình:

- Qua bảng tóm tắt, có thể xác định được điểm nào mà mô hình bắt đầu overfit (độ chính xác trên tập huấn luyện tăng trong khi độ chính xác trên tập kiểm tra giảm) hoặc xác định số lượng epoch tối ưu cho việc huấn luyện mô hình.

4.5.6. Kết luận

4.5.6.1. Ưu điểm của mô hình

Tích hợp linh hoạt với hệ thống microservice:

- Mô hình AI được triển khai dưới dạng Flask kết hợp với Kafka để giao tiếp giữa các microservice, đảm bảo hệ thống hoạt động độc lập, linh hoạt và dễ dàng hoạt động

Hiệu suất dự đoán nhanh:

- CNN đảm bảo hệ thống dự đoán nhanh, phù hợp với hệ thống gợi ý thời gian thực

Khả năng mở rộng:

- Có thể dễ dàng mở rộng mô hình để học từ các nguồn dữ liệu phong phú hơn như metadata video, thông tin người dùng, hoặc các tương tác khác (like, comment, share).

4.5.6.2. Nhược điểm của mô hình

Phụ thuộc vào dữ liệu lịch sử người dùng:

- Mô hình hoạt động dựa trên lịch sử xem video của người dùng, vì vậy với người dùng mới (chưa có dữ liệu lịch sử), hệ thống sẽ không đưa ra đề xuất.

Chưa kết hợp đầy đủ các tín hiệu khác:

- Mô hình hiện tại mới sử dụng lịch sử xem thể loại video. Các yếu tố như độ dài video, thời gian xem trung bình, hay các tương tác khác như like, comment, chia sẻ vẫn chưa được đưa vào mô hình.

4.5.6.3. Định hướng phát triển

Tích hợp nhiều nguồn dữ liệu đầu vào:

- Mở rộng dữ liệu huấn luyện gồm các thông số (thời gian xem, số lượt thích, bình luận,...)

Xây dựng hệ thống tự huấn luyện lại mô hình

- Kết hợp Kafka để tự động thu thập dữ liệu mới từ người dùng và tự động huấn luyện lại mô hình nhằm phù hợp với xu hướng thay đổi của người dùng

4.6. Triển khai hệ thống quản lý log cho ứng dụng

4.6.1. Cấu hình Logstash

Sau khi thực hiện tải ELK (Elasticsearch, Logstash & Kibana) vào server môi trường Linux, thực hiện chỉnh sửa file cấu hình input và output của Logstash để thêm các port tương ứng cho các service gửi log vào và xây dựng định dạng file log để thu thập và đọc dữ liệu trên kibana.

Cấu hình file input của Logstash

Tại file input trên, Logstash sẽ lắng nghe 3 port 5050, 5060, 5070 tương ứng với nơi các microservices sẽ gửi log về và lắng nghe port 5044 là nơi để filebeat thao tác trên hệ thống.

Cấu hình file output của Logstash

```
output {
  if ([type] == "userservice") {
    elasticsearch {
      hosts => ["http://192.168.2.45:9200"]
      index => "springboot-user-logs-%{+YYYY.MM.dd}"
    }
  }
  else if ([type] == "commentservice") {
    elasticsearch {
      hosts => ["http://192.168.2.45:9200"]
      index => "springboot-comment-logs-%{+YYYY.MM.dd}"
    }
  }
  else if ([type] == "videoservice") {
    elasticsearch {
      hosts => ["http://192.168.2.45:9200"]
      index => "springboot-video-logs-%{+YYYY.MM.dd}"
    }
  }
}
```

Thêm các trường type để phân lọc dữ liệu log đầu vào, log của service nào sẽ tương ứng với nhãn type ấy và xuất ra file log có tên của services đó. Bên cạnh đó là câu điều kiện để phân loại trường pipeline cho filebeat và thay đổi pipeline tương ứng.

4.6.1.1. Cấu hình chung

Thêm thư viện tại file pom.xml cho tất cả các services để chạy logstash

```
<dependency>
    <groupId>net.logstash.logback</groupId>
    <artifactId>logstash-logback-encoder</artifactId>
    <version>7.0.1</version>
</dependency>
```

Sau đó cấu hình cho các logging để đảm bảo các log sẽ chạy theo cấu hình của file logback-spring.xml tại application.properties

Cuối cùng, thêm file logback-spring.xml là nơi cấu hình cho các log được gửi thẳng đến Logstash của server. Các thông tin cần thiết bao gồm ip của server và port mà service đó sẽ sử dụng.

```
<appender name="LOGSTASH"
class="net.logstash.logback.appenders.LogstashTcpSocketAppender">
    <destination>192.168.120.213:port</destination>
    <encoder class="net.logstash.logback.encoder.LogstashEncoder">
    </encoder>
</appender>
```

4.6.1.2. Cấu hình User Services

Toàn bộ log dưới đây được thêm vào file UserController.java

Đăng ký tài khoản mới

```
MDC.put("type", "userservice");
MDC.put("action", "register");
logger.info("UserID: " + userFromDb.getId());
```

Sử dụng MDC để thêm các nhãn cho log được gửi đi, ở đây nhãn “type” đóng vai trò là phân loại của dữ liệu giữa các service trong trường hợp là “userservice” và “action” là thao tác của người dùng là đăng ký tài khoản “register”.

Sau đó sử dụng biến logger từ “LoggerFactory” để tạo 1 log bao gồm thông tin là ID của user mới đăng ký.

Đăng nhập tài khoản

Tương tự như đăng ký tài khoản, nhãn “action” sẽ là thao tác đăng nhập “login” và lấy thông tin ID của người dùng đã đăng nhập để gửi log về Logstash.

Cập nhật trang cá nhân

Thay đổi nhãn “action” thành “update-profile” để gửi log báo cáo người dùng đã cập nhật trang cá nhân

Thay đổi mật khẩu

Lúc này nhãn “action” sẽ là “change-password” để gửi log báo cáo người dùng đã thay đổi mật khẩu

4.6.1.3. Cấu hình Video Service

Toàn bộ log dưới đây được thêm tại file VideoService.java

Đăng tải video mới

```
MDC.put("type", "videoservice");
MDC.put("action", "upload");
MDC.put("videoid", videoID.toString());
logger.info("User " + userID.toString() + " upload a new video");
```

Thêm log tại hàm addVideo tương tự như User Service. Trường “type” lúc này sẽ đổi hoàn toàn qua Video Service với nhãn “action” cho biết hành động đăng tải video sẽ là “upload” và sẽ lấy 2 dữ liệu ID của người dùng và video để biết được người dùng nào đã upload video đó.

Chạy video

Thêm log tại hàm getVideoWithStream với “action” là “play-video” cùng với ID của Video để cho biết Video nào đang được chạy.

4.6.1.4. Cấu hình Comment Service

Tất cả log dưới đây được thực hiện tại file CommentController.java

Người dùng bình luận

```

MDC.put("type", "commentservice");
MDC.put("action", "upload");
MDC.put("userID", comment.getUserId());
MDC.put("videoID", comment.getVideoId());
logger.info("CommentId: " + savedComment.getId());

```

Tại hàm uploadComment thêm các log với nhãn “type” sẽ là commentservice, “action” sẽ là upload báo hiệu người dùng đăng tải bình luận và các trường thông tin cho biết người dùng (UserId) sẽ đăng tải bình luận (CommentId) ở video nào (VideoId).

4.7. Triển khai hệ thống Autoscale cho K8S

Horizontal Pod Autoscaling

Cài đặt Metrics Server

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Cài đặt Metrics Server qua lệnh trên và kiểm tra Metrics Server hoạt động bằng cách tìm kiếm các top pod qua kubectl

| NAME | CPU(cores) | MEMORY(bytes) |
|--|------------|---------------|
| api-gateway-8c5fbbb76-xhgkh | 2m | 162Mi |
| comment-mongo-deployment-7764f8868-fr677 | 10m | 109Mi |
| comment-service-57c594b849-9jcgx | 2m | 144Mi |
| react-frontend-6cdfb66b74-8mndk | 0m | 5Mi |
| redis-master-0 | 2m | 3Mi |
| redis-master-1 | 3m | 3Mi |
| redis-master-2 | 2m | 3Mi |
| redis-slave-0 | 3m | 3Mi |
| redis-slave-1 | 2m | 3Mi |
| redis-slave-2 | 3m | 3Mi |
| user-mongo-deployment-6d6d74947-w46fz | 10m | 94Mi |
| user-service-67f5ccb4f5-r26wq | 2m | 166Mi |
| video-mongo-deployment-6df6dfc86c-bvzdx | 10m | 107Mi |
| video-service-97ff4dbb7-vktb6 | 2m | 174Mi |

Hình 4.47: Kiểm tra Metrics Server

Xây dựng Autoscaling cho Frontend

```
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: react-frontend
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: react-frontend
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 80
```

Sử dụng Horizontal Pod Autoscaler để xây dựng autoscaling cho Frontend với số replica nhỏ nhất là 1 và nhiều nhất là 5, số replica sẽ tự động tăng lên khi pod sử dụng quá 80% của CPU.

Thực hiện tương tự cho 1 số pod quan trọng hoạt động nhiều CPU như API, Video Service, Comment Service và User Service

Statefulset Autoscaling

Để triển khai Autoscaling cho các Statefulset (Redis) thì cần phải cài đặt KEDA để sử dụng tính năng ScaledObject

Cài đặt KEDA

Thực hiện cài đặt KEDA qua helm và kiểm tra namespace keda đảm bảo KEDA đã hoạt động

```
root@cluster-2:/home/ubuntu# kubectl get all -n keda
NAME                                         READY   STATUS    RESTARTS   AGE
pod/keda-admission-webhooks-79f9b7b679-2fx6w   1/1     Running   0          14d
pod/keda-operator-77995499d6-kmzx7            1/1     Running   1 (14d ago) 14d
pod/keda-operator-metrics-apiserver-7858df7987-zm7x8  1/1     Running   0          14d

NAME                           TYPE      CLUSTER-IP        EXTERNAL-IP   PORT(S)      AGE
service/keda-admission-webhooks   ClusterIP   10.98.53.230   <none>       443/TCP      14d
service/keda-operator             ClusterIP   10.101.5.26    <none>       9666/TCP     14d
service/keda-operator-metrics-apiserver   ClusterIP   10.107.176.59   <none>       443/TCP,8080/TCP 14d

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/keda-admission-webhooks   1/1     1           1           14d
deployment.apps/keda-operator             1/1     1           1           14d
deployment.apps/keda-operator-metrics-apiserver  1/1     1           1           14d

NAME                               DESIRED  CURRENT   READY   AGE
replicaset.apps/keda-admission-webhooks   1        1         1        14d
replicaset.apps/keda-operator-77995499d6   1        1         1        14d
replicaset.apps/keda-operator-metrics-apiserver  1        1         1        14d
```

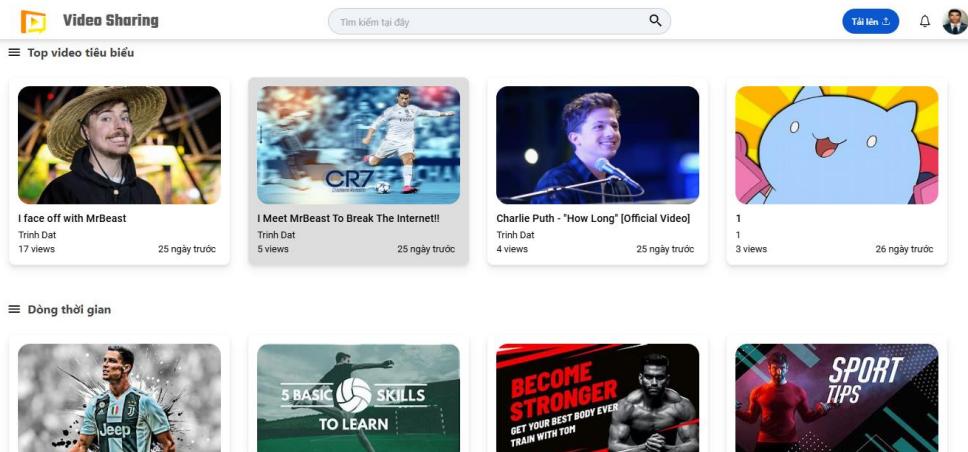
Hình 4.48: Kiểm tra KEDA

Triển khai Autoscaling cho Redis master bằng ScaledObject

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: redis-master-autoscaler
  namespace: default
spec:
  scaleTargetRef:
    name: redis-master
    kind: StatefulSet
  minReplicaCount: 3
  maxReplicaCount: 9
  triggers:
    - type: cpu
      metadata:
        type: Utilization
        value: "70"
```

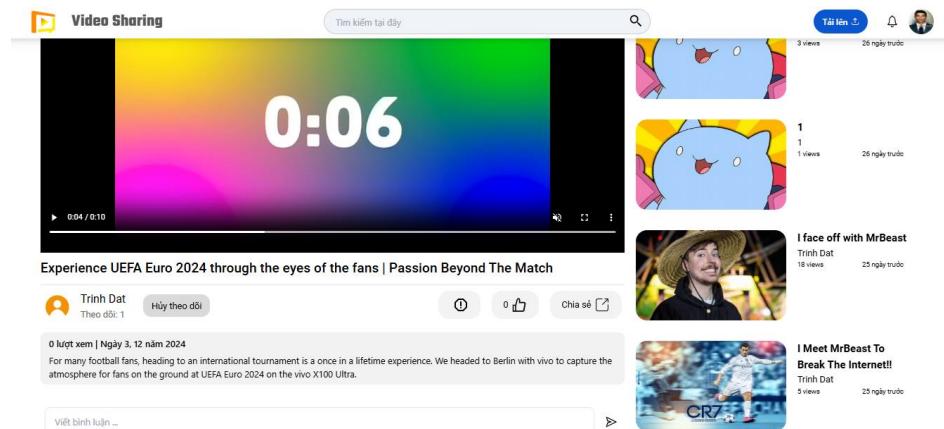
Sử dụng api của KEDA để cài đặt ScaledObject cho Statefulset redis-master với giới hạn của 3 pod master được tăng lên là 9 khi mức sử dụng của pod bất kì vượt quá 70% CPU. Thực hiện tương tự với Statefulset redis-slave

4.8. Đặc tả giao diện tính năng ứng dụng



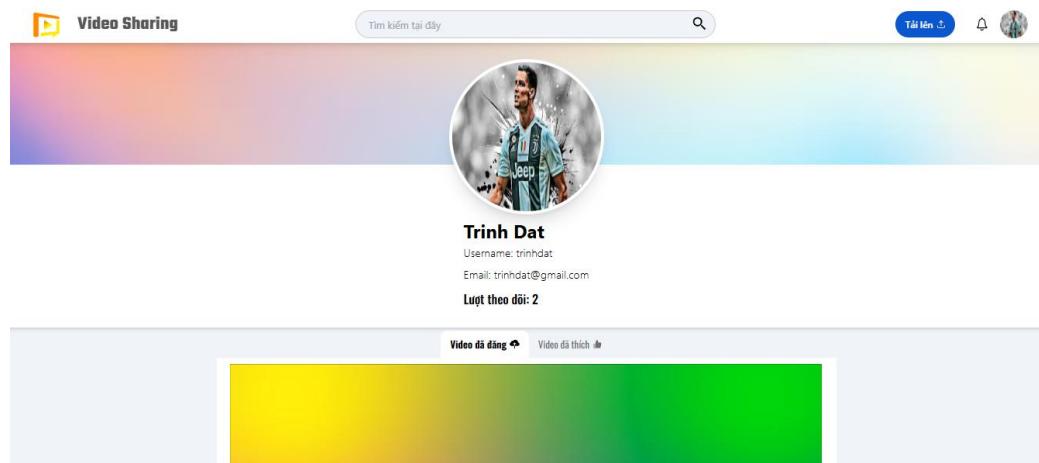
Hình 4.49: Giao diện chính của ứng dụng

Thanh Điều Hướng cung cấp logo dẫn về trang chủ và các liên kết đến Trang Chủ, Tải Lên Video, Lịch Sử Xem, Phát LiveStream ,Hồ Sơ Cá Nhân, và Đăng Xuất/Đăng Nhập tùy trạng thái người dùng. Danh Sách Video hiển thị hình thu nhỏ, tiêu đề, người đăng, số lượt xem, và cho phép sắp xếp theo mới nhất, phổ biến nhất, hoặc gợi ý cá nhân hóa. Thanh Tìm Kiếm hỗ trợ tìm kiếm nhanh theo từ khóa, tiêu đề và gợi ý tự động khi nhập liệu.



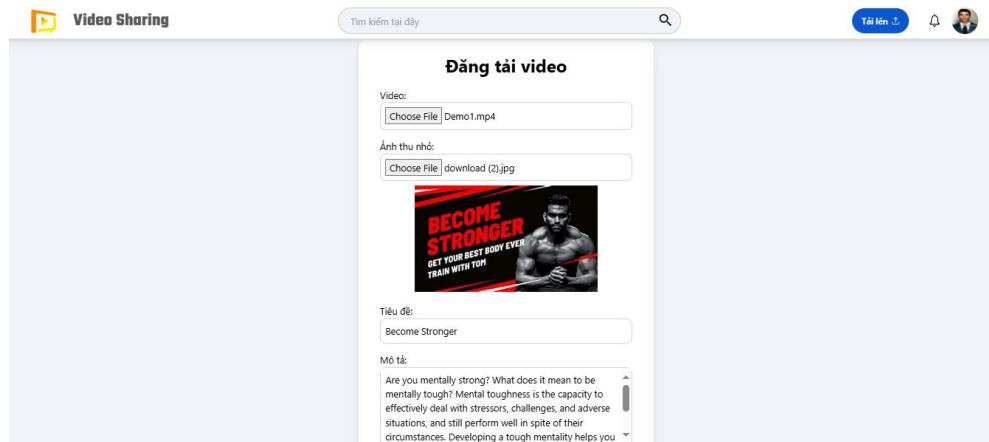
Hình 4.50: Màn hình xem video

Trình phát video hiển thị nội dung cùng các nút điều khiển cơ bản, tiêu đề, mô tả, thông tin người đăng, và ngày tải lên. Người dùng có thể thích, không thích, xem hoặc thêm bình luận, và truy cập danh sách video liên quan được đề xuất.



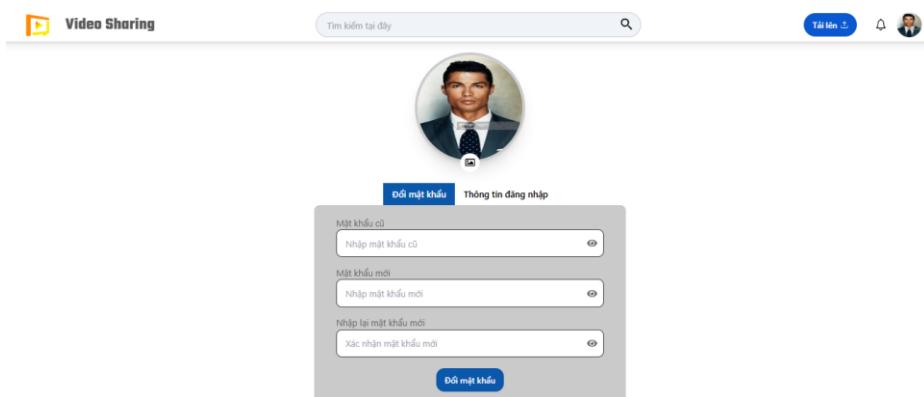
Hình 4.51: Màn hình hồ sơ người dùng

Trang thông tin cá nhân hiển thị tên, email, ảnh đại diện, danh sách video đã tải lên với lượt xem, thích, bình luận, và cho phép chỉnh sửa thông tin cá nhân. Người dùng có thể thay đổi mật khẩu và quản lý các tùy chọn bảo mật tài khoản.



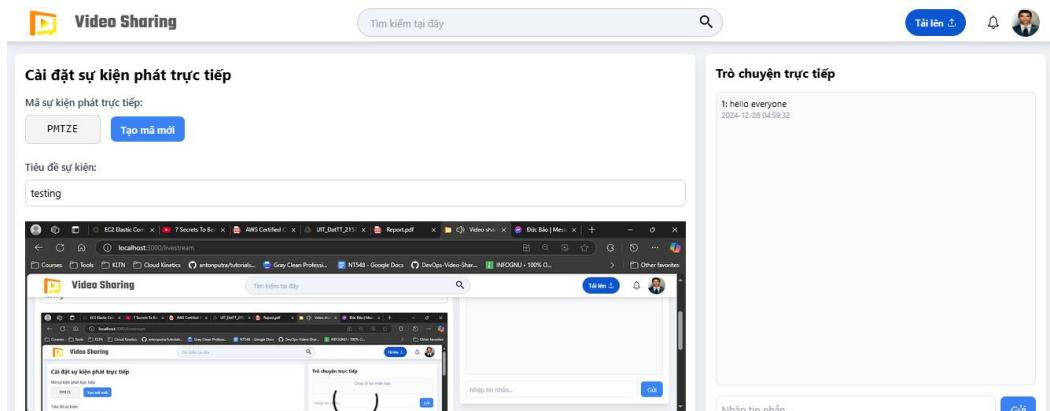
Hình 4.52: Màn hình tải lên video

Màn hình tải lên video gồm các trường nhập tiêu đề, mô tả, chọn tệp video và nút tải lên. Người dùng có thể xem trước video để kiểm tra chất lượng và nội dung trước khi tải lên.



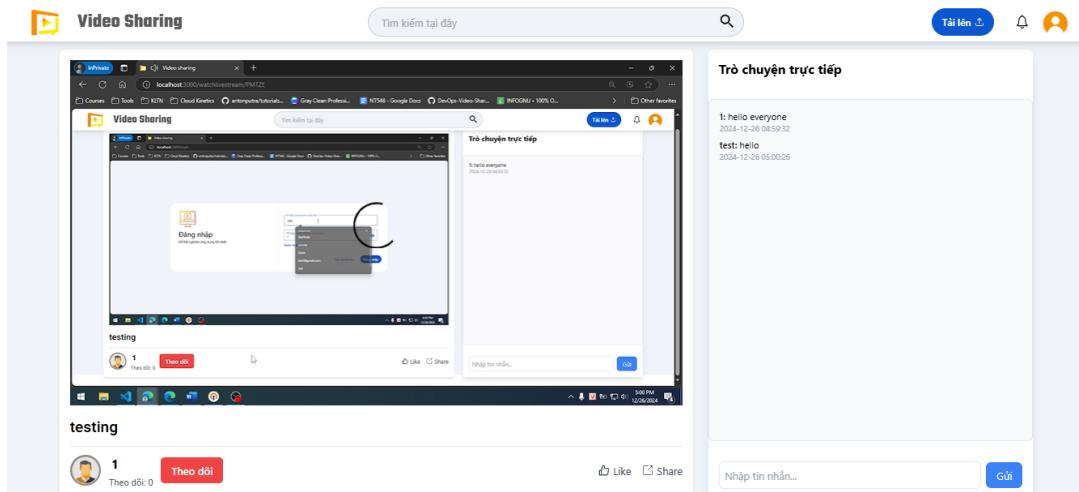
Hình 4.53: Màn hình thay đổi thông tin cá nhân

Màn hình thay đổi thông tin cá nhân bao gồm các trường nhập liệu cho tên, email, tên đăng nhập, và ảnh đại diện, cùng nút lưu thay đổi. Người dùng nhận thông báo thành công hoặc thất bại sau khi lưu.



Hình 4.54: Màn hình phát trực tiếp

Màn hình phát trực tiếp gồm Key (cho phép phát trực tiếp từ phần mềm thứ ba như OBS,...) và một thanh trò chuyện trực tiếp với người xem.



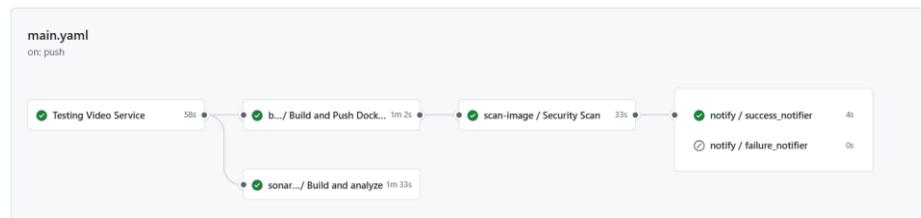
Hình 4.55: Màn hình xem phát trực tiếp

Màn hình xem phát trực tiếp gồm 1 khung hình xem và một cửa sổ trò chuyện trực tiếp.

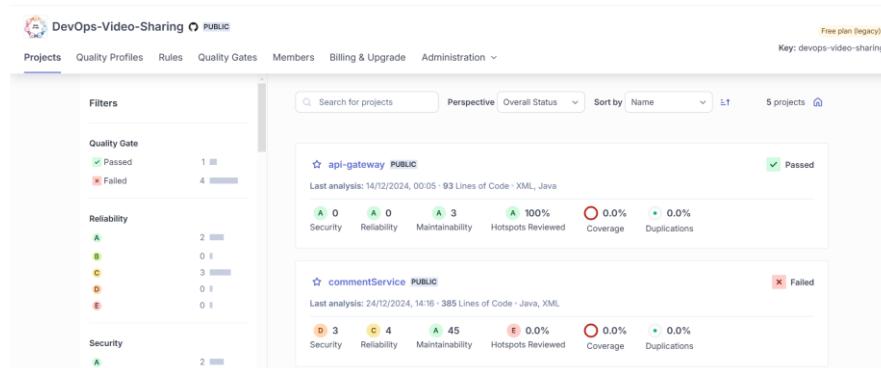
4.9. Hiện thực triển khai hệ thống

4.9.1. CI/CD pipeline

Thực hiện cập nhật và push code để theo dõi tiến trình hoạt động của pipeline

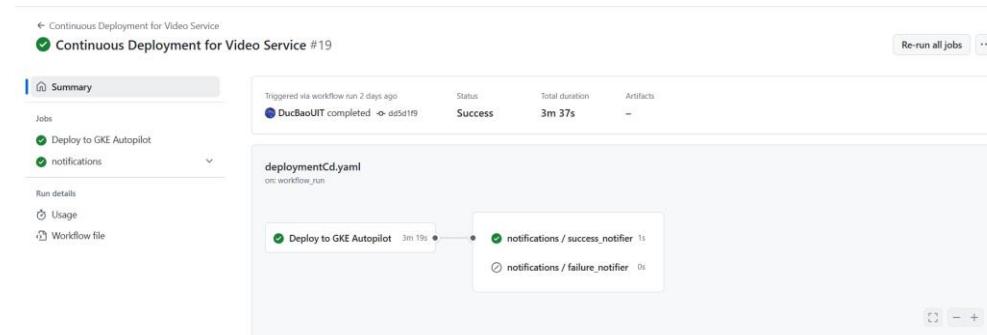


Hình 4.56: CI pipeline



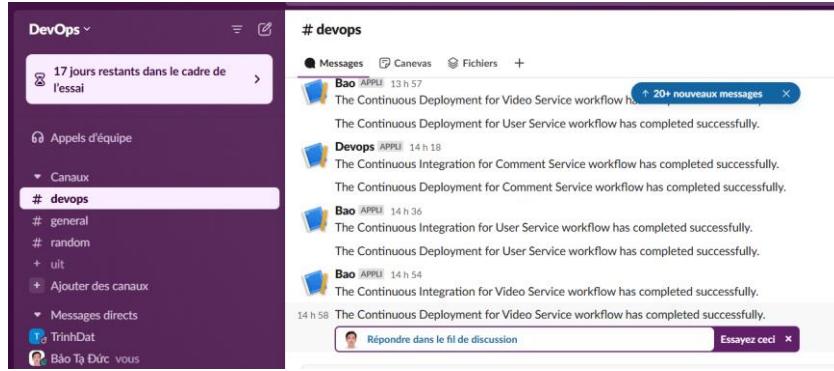
Hình 4.57: SonarQube quét code

CI Pipeline được hoàn thành tuần tự qua 4 giai đoạn: Kiểm tra code, xây dựng các image để chuẩn bị triển khai và quét image bằng sonarcloud, thực hiện scan docker image vừa build bằng Trivy và cuối cùng thông báo qua Slack.



Hình 4.58: CD pipeline

Sau khi CI pipeline hoàn thành, CD sẽ được thực hiện ngay sau đó để triển khai microservices lên server



Hình 4.59: Thông báo CI/CD từ Slack

CI/CD Pipeline sẽ được thực hiện tại 3 service (User service, Video service và Comment service) cùng API Gateway và Frontend tương tự như trên

4.9.2. Triển khai lên Server

| NAME | READY | STATUS | RESTARTS | AGE |
|---|-------|---------|----------|-------|
| api-gateway-649fbf8dc9-mlcw6 | 1/1 | Running | 0 | 2d10h |
| comment-mongo-deployment-5fcc68fd97-4nb2w | 1/1 | Running | 0 | 2d10h |
| comment-service-687f5f5d77-s9sc9 | 1/1 | Running | 0 | 2d9h |
| react-frontend-58d665c8f6-j4fl5 | 1/1 | Running | 0 | 2d10h |
| user-mongo-deployment-65d654d487-j8jds | 1/1 | Running | 0 | 2d10h |
| user-service-54c8d6bf8f-t9bwk | 1/1 | Running | 0 | 2d9h |
| video-mongo-deployment-77744fb98b-tm69f | 1/1 | Running | 0 | 2d10h |
| video-service-7b7d8f774-l74kf | 1/1 | Running | 0 | 2d9h |

Hình 4.60: Các pod chính của ứng dụng

Sau khi hoàn thành CI/CD pipeline, sẽ bao gồm 8 pod của ứng dụng được triển khai lên K8S để chạy ứng dụng với 2 service NodePort là API và Frontend để có thể kết nối dễ dàng bởi người dùng.

Bên cạnh đó sẽ bao gồm có 6 pod Redis (3 pod Master và 3 pod Slave) để cho các microservices kết nối vào, 2 pod sử dụng cho mô hình huấn luyện (Dự đoán bình luận tiêu cực và đề xuất video) và 3 pod KEDA sử dụng cho việc autoscale các StatefulSet. Ngoài ra, còn các pod khác sử dụng để quản lý cluster (Prometheus và Grafana)

4.9.3. Autoscale

| NAME | REFERENCE | TARGETS | MINPODS | MAXPODS | REPLICAS | AGE |
|----------------------------------|----------------------------|-------------|---------|---------|----------|-----|
| api-gateway | Deployment/api-gateway | cpu: 0%/80% | 1 | 5 | 1 | 62d |
| comment-service | Deployment/comment-service | cpu: 1%/80% | 1 | 5 | 1 | 62d |
| keda-hpa-redis-master-autoscaler | StatefulSet/redis-master | cpu: 1%/70% | 3 | 6 | 3 | 62d |
| keda-hpa-redis-slave-autoscaler | StatefulSet/redis-slave | cpu: 1%/70% | 3 | 6 | 3 | 62d |
| react-frontend | Deployment/react-frontend | cpu: 0%/80% | 1 | 5 | 1 | 39d |
| user-service | Deployment/user-service | cpu: 1%/80% | 1 | 5 | 1 | 62d |
| video-service | Deployment/video-service | cpu: 0%/80% | 1 | 5 | 1 | 62d |

Hình 4.61: Các Horizontal Pod Autoscaler

Các pod Microservice của ứng dụng sẽ tự động scale khi số CPU sử dụng vượt quá 80% và sẽ chỉ tăng được tối đa 5 pod đối với mỗi Deployments. Đối với các StatefulSet của redis master và redis slave sẽ tự động scales pod khi vượt quá 70% CPU cho trước và tối đa được phép tăng lên 6 pod cho mỗi StatefulSet.

4.9.4. Hệ thống quản lý Log ELK

4.9.4.1. Log của các Service

User Service

Thực hiện tạo tài khoản và đăng nhập trên ứng dụng để theo dõi log tại server Kibana

| Time ↓ | Document |
|------------------------------|---|
| > Oct 8, 2024 @ 23:23:39.560 | @timestamp: Oct 8, 2024 @ 23:23:39.560 @version: 1 action: login host: 192.168.120.213 level: INFO level_value: 20,000 logger_name: com.programming.userService.controller.UserController message: UserID: 67055c71af42f31743268331 port: 24,773 thread_name: http-nio-8081-exec-3 type: userservice _id: TmvxbJIBvK4yCfMagXaU _index: springboot-user-logs-2024.10.08 _score: - _type: _doc |
| > Oct 8, 2024 @ 23:23:13.843 | @timestamp: Oct 8, 2024 @ 23:23:13.843 @version: 1 action: register host: 192.168.120.213 level: INFO level_value: 20,000 logger_name: com.programming.userService.controller.UserController message: UserID: 67055c71af42f31743268331 port: 24,773 thread_name: http-nio-8081-exec-2 type: userservice _id: TGvxbJIBvK4yCfMAHXYh _index: springboot-user-logs-2024.10.08 _score: - _type: _doc |

Hình 4.62: Log người dùng đăng nhập và đăng ký

Tại log thu được từ Kibana, có thể thấy được các trường thông tin cơ bản về thời gian, phiên bản và cả IP của host, bên cạnh đó là trường type tương ứng với “userservice” như đã mô tả.

2 log trên chứa 2 nhãn “action” khác nhau, 1 log “register” và 1 log “login” và 2 log đều chứa UserID thực hiện

Video Service

| Time ↓ | Document |
|------------------------------|---|
| > Oct 9, 2024 @ 16:52:58.035 | @timestamp: Oct 9, 2024 @ 16:52:58.035 @version: 1 action: upload host: 192.168.120.11 level: INFO level_value: 20,000 logger_name: com.programming.videoService.service.VideoService message: User 67055c71af42f31743268331 upload a new video port: 49,044 thread_name: http-nio-8083-exec-2 type: videoservice videoId: 6706527abc1177082693b072 videoId.keyword: 6706527abc1177082693b072 _id: 6GuycJIBvK4yCfMAYEc _index: springboot-video-logs-2024.10.09 _score: - _type: _doc |

Hình 4.63: Log đăng tải video mới

| | |
|------------------------------|--|
| > Oct 9, 2024 @ 17:08:39.673 | @timestamp: Oct 9, 2024 @ 17:08:39.673 @version: 1 action: play-video host: 192.168.120.131 level: INFO level_value: 20,000 logger_name: com.programming.videoService.service.VideoService message: VideoId: 67065620c25d6e4adeeba030 port: 56,269 thread_name: http-nio-8083-exec-3 type: videoservice _id: QGvAcJIBvK4yCfMAi4J3 _index: springboot-video-logs-2024.10.09 _score: - _type: _doc |
|------------------------------|--|

Hình 4.64: Log xem video

Tương tự như User Service, log của Video Service cũng sẽ có các trường thông tin tương ứng. Đối với đăng tải video mới sẽ có thêm thông tin UserID để biết được người dùng đăng tải video đó.

Comment Service

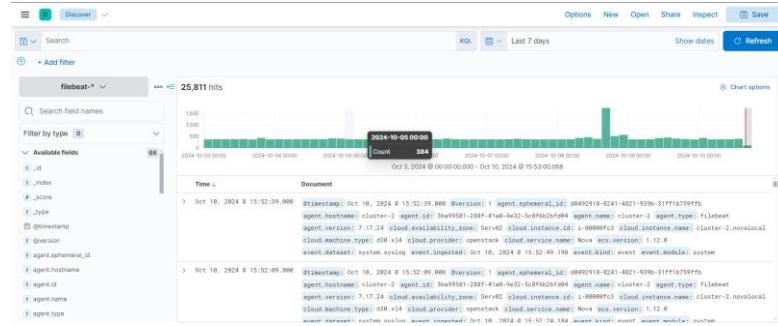
Thực hiện bình luận vào Video bất kì và kiểm tra log từ hệ thống ELK

| Time ↓ | Document |
|-------------------------------|--|
| > Dec 26, 2024 @ 17:18:46.526 | @timestamp: Dec 26, 2024 @ 17:18:46.526 @version: 1 action: upload host: 192.168.120.213 level: INFO level_value: 20,000 logger_name: com.programming.commentService.controller.CommentController message: CommentId: 2f77fd75-f439-4481-9658-431021b49a47 port: 21,436 thread_name: http-nio-8082-exec-2 type: commentservice userId: 676a6b181a0e383ccb17c095 videoID: 676a6b39f30006610cc5fdc1 _id: YH55ApQBvK4yCfMA1q14 _index: springboot-comment-logs-2024.12.26 _score: - _type: _doc |

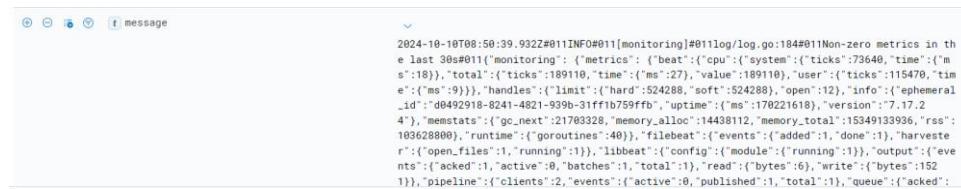
Hình 4.65: Log ELK của Comment Service

Đối với Comment Service, log thu được sẽ chứa trường thông tin UserID và VideoID để kiểm soát được bình luận của User nào trên Video tương ứng.

4.9.4.2. Log của hệ thống

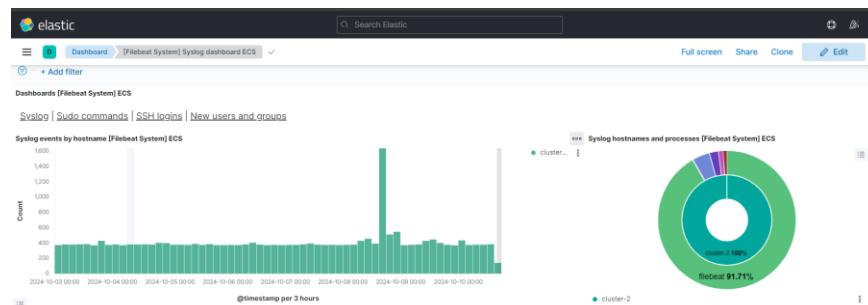


Hình 4.66: Log được gửi từ filebeat



Hình 4.67: Trường thông tin “message”

Log từ filebeat được xây dựng để quản lý hệ thống sẽ bao gồm các trường thông tin về server. Nó cung cấp những thông tin từ cloud về máy ảo như “cloud_provider”, “cloud_machine_type” giúp người dùng dễ dàng quản lý và theo dõi máy ảo qua log đồng thời filebeat còn cung cấp các message cho người dùng phân tích về quá trình quản lý hệ thống.



Hình 4.68: Dashboard

Dashboard trên cho biết được số lượng các log được gửi ra từ các thành phần khác nhau nhằm trực quan hóa dữ liệu cho người dùng. Bên cạnh đó, có thể theo dõi

sự xâm nhập vào server qua mục “Sudo commands” và “SSH logins” làm tăng được tính bảo mật của server.

4.9.5. Quản lý Cluster



Hình 4.69: Quản lý Cluster

Prometheus và Grafana sẽ đóng vai trò trong việc quản lý hiệu suất của các node. Dashboard sẽ là nơi hiển thị các thông tin bao gồm: Tốc độ mạng, dữ liệu sử dụng, CPU sử dụng, lưu lượng các tệp hệ thống và 1 số thông tin khác giúp người quản lý dễ dàng giám sát hoạt động và điều phối cluster.

Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Trong quá trình phát triển ứng dụng chia sẻ video dựa trên kiến trúc microservice, đã áp dụng các công nghệ mới để xây dựng một hệ thống mạnh mẽ, hiệu quả và dễ dàng mở rộng. Những điểm nổi bật của dự án bao gồm:

5.1.1. Ứng dụng CI/CD

Việc triển khai quy trình CI/CD đã giúp tối ưu hóa quy trình phát triển và triển khai ứng dụng. Bằng cách sử dụng công cụ GitHub Actions để tự động hóa quy trình làm việc và triển khai. Điều này giúp giảm thiểu lỗi, tăng tốc độ phát triển và đảm bảo tính ổn định của hệ thống.

5.1.2. Tích hợp AI

Hệ thống tích hợp các mô hình AI để cung cấp các tính năng thông minh, bao gồm:

Dự đoán bình luận độc hại:

Huấn luyện một mô hình giúp nhận biết các bình luận có mục đích tiêu cực.

Gợi ý video:

Huấn luyện một mô hình AI để xuất video giúp tăng trải nghiệm giữa người dùng và ứng dụng.

5.1.3. Khả năng mở rộng và quản lý hạ tầng

Hệ thống được thiết kế để đáp ứng các nhu cầu về hiệu năng và khả năng mở rộng bằng cách triển khai trên Kubernetes. Redis được sử dụng để cải thiện tốc độ truy cập dữ liệu thông qua caching và sử dụng Kafka để xử lý luồng dữ liệu lớn giữa các dịch vụ.

5.2. Hướng phát triển

5.2.1. Cải thiện mô hình AI để xuất video

Nâng cấp mô hình AI để có thể gợi ý video dựa trên hành vi tương tác của người dùng, đồng thời tích hợp tính năng tự động phân loại video thông qua phân tích hình ảnh. Điều này sẽ giúp nâng cao tính cá nhân hóa và cải thiện trải nghiệm người dùng.

5.2.2. Tìm hiểu và viết bài báo khoa học

Dự kiến trong tháng 6, sẽ hoàn thiện một bài báo khoa học nhằm chia sẻ kinh nghiệm và thành tựu trong việc ứng dụng AI vào kiến trúc microservice.

5.2.3. Tìm nhà đầu tư

Nhằm khắc phục hạn chế về phần cứng và mở rộng ứng dụng, chúng em có thể sẽ tìm kiếm một nhà đầu tư, đặc biệt là các trung tâm giáo dục. Họ có thể sử dụng nền tảng này của chúng em để tải lên video dạy nhằm phục vụ nhu cầu giảng dạy nội bộ một cách hiệu quả.

5.2.4. Cải thiện hệ thống phân phối video

Tích hợp hệ thống CDN để phân phối nội dung video tới người dùng cuối với tốc độ nhanh hơn và độ trễ thấp hơn. Giúp người dùng ở khu vực địa lý xa máy chủ chính vẫn có thể truy cập nội dung nhanh chóng và ổn định.

Nén video để giảm kích thước tệp video mà không ảnh hưởng đến chất lượng hình ảnh. Giúp giảm băng thông truyền tải hữu ích cho người dùng có kết nối internet yếu.

TÀI LIỆU THAM KHẢO

- [1] T. Y. TEAM, "Reimagining video infrastructure to empower YouTube," 21 06 2021. [Online]. Available: <https://blog.youtube/inside-youtube/new-era-video-infrastructure/>. [Accessed 31 12 2024].
- [2] J. Pradeeba, "A Netflix Special — Product Case Study (Tech Focused)," 22 07 2024. [Online]. Available: <https://medium.com/@pjega92/a-netflix-special-product-strategy-case-study-tech-focused-ec291c7b5597>. [Accessed 31 12 2024].
- [3] Elastic, "The Elastic Suite," Elastic, [Online]. Available: <https://www.elastic.co/fr/elasticsearch>.
- [4] Elastic, "Logstash," Elastic, [Online]. Available: <https://www.elastic.co/fr/logstash>.
- [5] Wikipedia, "Kibana," 10 02 2024. [Online]. Available: <https://en.wikipedia.org/wiki/Kibana>.
- [6] Wikipedia, "Autoscaling," 5 7 2024. [Online]. Available: <https://en.wikipedia.org/wiki/Autoscaling>.
- [7] Kubernetes, "Horizontal Pod Autoscaling," 03 10 2024. [Online]. Available: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>.
- [8] KEDA, "Kubernetes Event-driven Autoscaling," KEDA, [Online]. Available: <https://keda.sh/>.
- [9] B. đ. t. P. L. t. p. H. C. Minh, "Bình luận đèn - những cái thòng lọng trên mạng xã hội," 28 12 2020. [Online]. Available: <https://plo.vn/binh-luong-den-nhung-cai-thong-long-tren-mang-xa-hoi>.

luan-den-nhung-cai-thong-long-tren-mang-xa-hoi-post655103.html.
[Accessed 14 12 2024].

- [10] GeeksforGeeks, "Toxic Comment Classification using BERT," GeeksforGeeks, 24 05 2024. [Online]. Available: <https://www.geeksforgeeks.org/toxic-comment-classification-using-bert/>. [Accessed 14 12 2024].
- [11] C. Hashemi-pour, "BERT language model," 2014. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model>. [Accessed 14 12 2024].
- [12] Jigsaw, "Toxic Comment Classification Challenge," Kaggle, 13 5 2018. [Online]. Available: <https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/overview>. [Accessed 14 12 2024].
- [13] IEEE, "Evaluating the Effectiveness of Capsule Neural Network in Toxic Comment Classification Using Pre-Trained BERT Embeddings," 31 10 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10322429>. [Accessed 14 12 2024].
- [14] S. K. T. A. G. V. V. P. P. Spiros V. Georgakopoulos, "Convolutional Neural Networks for Toxic Comment," 2018.
- [15] G. R. R. A. K. E. 1Shanmughapriya M, JIGSAW MULTILINGUAL TOXIC COMMENT, IJCRT, 2022.
- [16] F. Mohammad, "Is preprocessing of text really worth your time for toxic comment," p. 7, 2018.

- [17] T. c. n. hàng, "Xây dựng nền tảng tín dụng ngân hàng khu vực dựa trên Microservices," Tạp chí ngân hàng, 28 07 2021. [Online]. Available: <https://tapchinganhang.gov.vn/xay-dung-nen-tang-tin-dung-ngan-hang-khu-vuc-dua-tren-microservices-11712.html>. [Accessed 12 2024].
- [18] P. S. G. Nupura Torvekar, "Microservices and It's Applications : An Overview," *International Journal of Computer Sciences and Engineering*, p. 7, 2019.
- [19] J. B. H. O. Mali Senapathi, "DevOps Capabilities, Practices, and Challenges: Insights from a Case," p. 11, 2018.