

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



**BÁO CÁO ĐỒ ÁN**

**MÔN: HỆ TÍNH TOÁN PHÂN BỐ**

**ĐỀ TÀI:**

**TRIỂN KHAI HỆ THỐNG**

**MACHINE LEARNING PHÂN BỐ**

**THÔNG QUA SPARK TRÊN GOOGLE CLOUD**

**Giảng viên hướng dẫn: TS. Huỳnh Văn Đặng**

**Lớp: NT533.Q13**

**Sinh viên thực hiện:**

Cáp Hữu Tú – 23521696

Huỳnh Ngọc Ngân Tuyền – 23521753

**Thời gian thực hiện: 29/9/2025 – 29/10/2025**

□□ Tp. Hồ Chí Minh, 10/2025 □□

## **NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

....., ngày ..... tháng ..... năm 2025

## Người nhận xét

(Ký tên và ghi rõ họ tên)

## MỤC LỤC

<b>DANH MỤC HÌNH ẢNH.....</b>	<b>7</b>
<b>DANH MỤC BẢNG.....</b>	<b>8</b>
<b>DANH MỤC BIỂU ĐỒ.....</b>	<b>9</b>
<b>CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN.....</b>	<b>10</b>

<b>CHƯƠNG 2. CÁC CÔNG NGHỆ LIÊN QUAN .....</b>	<b>12</b>
1.1. Mục tiêu của đề tài.....	10
1.2. Tóm tắt nội dung thực hiện .....	10
2.1.1. Khái niệm.....	12
2.1. Công nghệ 1: Apache Spark 3.5.7 .....	12
2.1.2. Đặc điểm .....	12
2.1.3. Khả năng .....	12
2.1.4. Chức năng và nhiệm vụ .....	12
2.1.5. Lý do chọn Apache Spark trong mô hình đồ án .....	13
2.1.5. Lý do chọn Spark phiên bản 3.5.7 .....	13
2.2. Công nghệ 2: Google Cloud Platform (GCP).....	14
2.2.1. Khái niệm.....	14
2.2.2. Đặc điểm .....	14
2.2.3. Khả năng .....	14
2.2.4. Chức năng và nhiệm vụ .....	15
2.2.5. Lý do chọn .....	15
2.3. Công nghệ 3: Google Cloud Storage (GCS).....	16
2.3.2. Đặc điểm .....	17
2.3.3. Khả năng.....	17
2.3.4 Chức năng và nhiệm vụ .....	17
2.3.5. Lý do chọn Google Cloud Storage trong mô hình đồ án .....	18
2.4. Công nghệ 4: GCS Connector (hadoop3-2.2.29).....	18
2.4.1. Khái niệm .....	18

2.4.2. Đặc điểm.....	18
2.4.3. Khả năng.....	18
2.4.4. Chức năng và nhiệm vụ trong đồ án.....	18
 2.5.1. Khái niệm .....	18
2.5.2. Đặc điểm.....	18
2.5.3: Khả năng.....	19
2.5. Công nghệ 5: Apache ZooKeeper 3.7.1.....	18
2.5.4. Chức năng và nhiệm vụ trong đồ án.....	19
2.5.5. Lý do lựa chọn.....	19
 2.6.1. Khái niệm .....	19
2.6. Công nghệ 6: Python 3.8 & pip3 .....	19
2.6.2. Đặc điểm.....	19
2.6.3. Khả năng.....	20
2.6.4. Chức năng và nhiệm vụ trong đồ án.....	20
2.6.5. Lý do lựa chọn.....	20
<b>CHƯƠNG 3. HIỆN THỰC HOÁ .....</b>	<b>21</b>
3.1. Mô hình .....	21
3.2. Các giải thuật đã sử dụng.....	22
 3.2.1. Giải thuật Phân loại (Classification) .....	22
3.2.2. Giải thuật Phân cụm (Clustering) .....	23
3.3. Các bước thực hiện .....	24
3.2.3. Giải thuật mapPartition .....	24
 3.3.1. Train model trên môi trường local .....	25
a) Dự đoán khách hàng rời đi, dataset nhỏ, giải thuật Random Forest.....	26
b) Dự đoán khách hàng rời đi, dataset lớn, giải thuật Random Forest. ....	29
c) Dự đoán khách hàng rời đi, dataset lớn, giải thuật GBT.....	32
d) Phân loại khách hàng theo nhóm, dataset lớn, giải thuật KMeans.....	33
e) Bài toán mở rộng: Phân loại 1 tập dữ liệu hình ảnh lớn theo phương pháp học chuyển giao (Transfer Learning), mô hình học sâu ResNet50. ....	34
3.3.2. Xây dựng hạ tầng mạng.....	39
Tạo VPC .....	39
Tạo Subnets.....	39
Firewalls.....	40

Cloud NAT & Private Google Acces .....	41
3.3.3. Service Accounts và quyền .....	41
3.3.4. Cụm Spark Master (3 nút High Availability) .....	42
3.3.5. Spark Worker Autoscaling Group .....	47
3.3.6. Cụm API + Load Balancer .....	51
3.3.7. Phân tải tác vụ nặng ở API (MapPartition và Scaling API).....	53
3.3.8. Chuẩn bị script để chạy tác vụ huấn luyện từ local.....	55
<b>CHƯƠNG 4. CÁC KỊCH BẢN THỬ NGHIỆM VÀ KẾT QUẢ .....</b>	<b>58</b>
4.1. Kịch bản 1: Kiểm tra hoạt động của cụm Master HA:.....	58
4.2. Kịch bản 2: Kiểm tra tính chịu lỗi (Fault Tolerance) của hệ thống .....	59
4.2.1. Chịu lỗi Worker Node.....	60
4.2.2. Kiểm tra Tính nhất quán sau lỗi .....	61
4.3. Kịch bản 3: Đánh giá hiệu năng và khả năng mở rộng theo chiều ngang (Scalability) đối với cụm Spark worker.....	61
4.3.1. Chạy trên một worker:.....	61
4.3.1. Chạy trên “cụm phân tán”: .....	62
4.3.2. Kết quả.....	62
4.4. Kết quả tổng hợp từ Kịch bản 2 và Kịch bản 3.....	63
4.5. Kịch bản 4: Kiểm thử toàn bộ quy trình (End-to-End) và hiệu năng API Dự đoán.....	63
4.6. Kịch bản 5: Kiểm thử khả năng Autoscaling của cụm API Server, khả năng hoạt động của Load Balancer.....	65
4.7. Kịch bản 7: Kiểm thử hoạt động của IAM .....	69
Phân tích kết quả từ Locust.....	67
5.1. Tổng hợp kết quả .....	71
5.2. Kết luận .....	71
5.3. Ưu điểm.....	72
5.4. Nhược điểm.....	73
<b>CHƯƠNG 5. KẾT LUẬN.....</b>	<b>71</b>

<b>CHƯƠNG 6. TÀI LIỆU THAM KHẢO.....</b>	<b>76</b>
<b>5. CHƯƠNG 7. BẢNG PHÂN CÔNG ĐÁNH GIÁ THÀNH VIÊN .....</b>	<b>77</b>
5.6. Hướng phát triển .....	75
5.7. Kết luận tổng quát .....	75

## **DANH MỤC HÌNH ẢNH**

Hình 3-1. Mô hình giải pháp .....	21
Hình 3-2. Kết quả huấn luyện local.....	28
Hình 3-3. Kết quả huấn luyện local.....	30
Hình 3-4. Bảng so sánh 2 script huấn luyện.....	31
Hình 3-5. Kết quả huấn luyện local.....	33
Hình 3-6. Kết quả huấn luyện local.....	39
Hình 3-7. Tạo VPC.....	39
Hình 3-8. Tạo subnet .....	40
Hình 3-9. Spark worker template .....	50
Hình 3-10. Spark worker group .....	51
Hình 3-11. api Load Balancer .....	53
Hình 4-1. Spark UI .....	58
Hình 4-2. Các status ban đầu trên Spark UI .....	59
Hình 4-3. Máy master 3 up status lên ALIVE.....	59
Hình 4-4. Spark phát hiện worker mất kết nối .....	60
Hình 4-5. Kết quả độ chính xác của mô hình .....	61
Hình 4-6. Kết quả thử nghiệm từ API gọi mô hình.....	61
Hình 4-7. Kết quả chạy trên 1 worker .....	62
Hình 4-8. Kết quả chạy trên cụm phân tán .....	63
Hình 4-9. Web UI API FLask.....	64
Hình 4-10. Kết quả khi up 1 file customer.csv.....	64
Hình 4-11. Kết quả Classify ảnh chó/mèo.....	65
Hình 4-12. Kết quả kiểm thử IAM .....	70

## **DANH MỤC BẢNG**

Bảng 3-1. Bảng thành phần của mô hình .....	22
Bảng 3-2. Firewall rules .....	41
Bảng 3-3. Các gói/dịch vụ cho spark master.....	43
Bảng 4-1. Cấu hình test Locust - stress test trên API server .....	65
Bảng 4-2. Lệnh kiểm thử Locust.....	66
Bảng 7-1. Bảng phân chia công việc .....	77
Bảng 7-2. Bảng timeline công việc .....	77
Bảng 7-3. Các URL .....	77

## **DANH MỤC BIỂU ĐỒ**

Biểu đồ 4-1. Biểu đồ so sánh hiệu năng của mô hình .....	63
Biểu đồ 4-2. CPU Utilization .....	66
Biểu đồ 4-3. Disk I/O .....	66
Biểu đồ 4-4. Network I/O .....	67
Biểu đồ 4-5. Autoscaler .....	67
Biểu đồ 4-6. So sánh thời gian phản hồi trung bình giữa hai lần kiểm thử.....	68
Biểu đồ 4-7. So sánh throughput (req/s) giữa hai lần kiểm thử .....	68
Biểu đồ 4-8. So sánh tỷ lệ lỗi (Failures/s) giữa hai lần kiểm thử.....	69

# CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN

## 1.1. Mục tiêu của đề tài

Xây dựng nền tảng tính toán phân tán: Triển khai và cấu hình thành công một cụm Apache Spark Standalone đa nút trên các máy ảo của Google Compute Engine, cùng với một hệ thống lưu trữ phân tán sử dụng Google Cloud Storage.

Phát triển pipeline machine learning: Xây dựng một quy trình xử lý dữ liệu hoàn chỉnh bằng PySpark, bao gồm các bước từ đọc dữ liệu, tiền xử lý, trích xuất đặc trưng, huấn luyện và đánh giá mô hình machine learning.

Đánh giá và chứng minh các tính chất của hệ thống Phân tán: Thực hiện các kịch bản kiểm thử để đánh giá hiệu năng, khả năng mở rộng (scalability) và tính chịu lỗi (fault tolerance) của hệ thống.

Triển khai Mô hình thành Dịch vụ: Đóng gói mô hình đã huấn luyện và triển khai nó dưới dạng một dịch vụ API đơn giản, có khả năng nhận dữ liệu và trả về kết quả dự đoán.

## 1.2. Tóm tắt nội dung thực hiện

Nghiên cứu lý thuyết: Phân tích kiến trúc, đặc điểm và vai trò của các công nghệ: Spark, Google Cloud, Google Cloud Storage, Google Cloud Connector: hadoop3-2.2.29, Zookeeper 3.7.1, Python3 pip3, Ray cluster và các môi trường chạy Spark trên GCP.

Quá trình triển khai gồm 3 giai đoạn:

- Giai đoạn 1: Phát triển Cục bộ (Local):
  - Thực hiện nghiên cứu và phát triển các kỹ thuật ML trên Spark, bao gồm thử nghiệm với bài toán dự đoán, phân cụm.
  - Xây dựng và gỡ lỗi kịch bản PySpark MLlib hoàn chỉnh cho bài toán dự đoán, phân loại, phân cụm Churn trên máy cá nhân.
  - Phát triển kịch bản Transfer Learning hoàn chỉnh cho bài toán phân loại hình ảnh Chó/Mèo.
- Giai đoạn 2: Xây dựng Hạ tầng Cloud (GCP):
  - Thiết lập môi trường GCP: Tạo Project, cấu hình mạng VPC default, thiết lập các quy tắc Firewall (allow-ssh, allow-internal, allow-web-ui).
  - Triển khai 03 máy ảo Google Compute Engine (1 Master, 2 Worker) với cấu hình e2-standard-4.
- Giai đoạn 3: Triển khai Cụm Spark & Ứng dụng:
  - Cài đặt và cấu hình thủ công Apache Spark trên 03 máy ảo GCE để tạo thành một cụm standalone.

- Di chuyển (migrate) kịch bản PySpark (bài toán Churn) lên cụm Spark, tinh chỉnh để kết nối và đọc/ghi dữ liệu từ Google Cloud Storage (GCS).
- Giai đoạn 4: Kiểm thử và Đánh giá:
  - Thực hiện 03 kịch bản thử nghiệm chuyên sâu để đo lường: khả năng mở rộng (thêm node), khả năng chịu lỗi (mô phỏng tắt worker) và hiệu năng phục vụ thực tế (API load testing).

Tổng kết: Đánh giá kết quả đạt được, chỉ ra các hạn chế và đề xuất hướng phát triển trong tương lai.

## CHƯƠNG 2. CÁC CÔNG NGHỆ LIÊN QUAN

### 2.1. Công nghệ 1: Apache Spark 3.5.7

#### 2.1.1. Khái niệm

Apache Spark là một framework mã nguồn mở, dùng cho điện toán cụm (cluster computing) phân tán, cung cấp giao diện lập trình cho toàn bộ cụm với khả năng chịu lỗi (fault tolerance) và song song hóa dữ liệu (data parallelism) ngầm định. Spark được thiết kế để xử lý dữ liệu quy mô lớn một cách nhanh chóng.

#### 2.1.2. Đặc điểm

Xử lý In-Memory: Đặc điểm nổi bật nhất là khả năng lưu trữ dữ liệu trong bộ nhớ (RAM) của cụm máy tính thay vì trên đĩa cứng, giúp tăng tốc độ xử lý lên gấp nhiều lần so với các framework truyền thống (như MapReduce).

Tính toán lười biếng (Lazy Evaluation): Spark không thực thi ngay lập tức các phép biến đổi (transformation) mà xây dựng một kế hoạch (DAG). Nó chỉ bắt đầu tính toán khi một hành động (action) được gọi (ví dụ: save(), show(), count()).

Tính chịu lỗi (Fault Tolerance): Dựa trên cấu trúc dữ liệu RDD (Resilient Distributed Dataset) hoặc DataFrame, Spark có thể tự động phục hồi dữ liệu bị mất nếu một nút (worker) trong cụm gặp sự cố.

#### 2.1.3. Khả năng

Spark là một hệ sinh thái đa năng, bao gồm:

Spark SQL: Xử lý dữ liệu có cấu trúc bằng cú pháp SQL.

Spark MLlib: Thư viện học máy (Machine Learning) phân tán.

Spark Streaming: Xử lý dữ liệu theo thời gian thực (real-time).

GraphX: Xử lý đồ thị (graph processing).

#### 2.1.4. Chức năng và nhiệm vụ

Trong khuôn khổ đề tài này, Spark (cụ thể là PySpark) đóng vai trò là "bộ não" xử lý trung tâm:

- Huấn luyện ML: Chạy các thuật toán học máy (RandomForest, GBTClassifier, KMeans) trên bộ dữ liệu Telco Churn.
- Tiền xử lý (Featurization): Thực thi pipeline StringIndexer, OneHotEncoder, VectorAssembler trên dữ liệu phân tán.
- Trích xuất đặc trưng ảnh: Sử dụng Pandas UDF để phân tán công việc dùng ResNet50 trích xuất đặc trưng từ hàng ngàn bức ảnh Chó/Mèo.
- Đọc/Ghi dữ liệu: Là công cụ đọc dữ liệu đầu vào (CSV, ảnh) từ Google Cloud Storage (GCS) và ghi mô hình đã huấn luyện ngược trở lại GCS.

## 2.1.5. Lý do chọn Apache Spark trong mô hình đồ án

Apache Spark được chọn vì nó là tiêu chuẩn ngành cho xử lý Big Data. Khả năng xử lý in-memory và thư viện MLlib tích hợp làm cho nó trở thành lựa chọn lý tưởng cho bài toán Machine Learning phân tán, vốn là trọng tâm của đề tài. PySpark API cho phép tận dụng hệ sinh thái Python (quen thuộc với Data Scientist) trong khi vẫn có được sức mạnh của xử lý phân tán.

### 2.1.5. Lý do chọn Spark phiên bản 3.5.7

Việc lựa chọn phiên bản Spark 3.5.7 cho đồ án này là một quyết định kỹ thuật dựa trên 3 yếu tố chính: Tính ổn định, Khả năng tương thích hệ sinh thái, và Hiệu năng cho AI/ML.

Tính ổn định và bảo mật: Phiên bản 3.5.7 là một bản phát hành bảo trì. Điều này có nghĩa là nó đã tích lũy tất cả các bản sửa lỗi (bug fixes) và vá lỗi bảo mật (security patches) được phát hiện kể từ phiên bản 3.5.0. Trong một môi trường đồ án phức tạp, việc sử dụng phiên bản mới nhất và  *ổn định nhất* giúp tránh được các lỗi không mong muốn từ chính framework, cho phép nhóm tập trung vào logic nghiệp vụ (huấn luyện mô hình, xây dựng API). Trong khi đó, một phiên bản .0 mới nhất (như 4.0.0) luôn đi kèm rủi ro. Nó có thể chứa các lỗi mới chưa được phát hiện (undiscovered bugs) có thể làm sụp đổ toàn bộ quy trình xử lý dữ liệu. Trong một dự án phức tạp, ổn định là yếu tố quan trọng hơn.

Khả năng tương thích về hệ sinh thái:

- Tương thích Hadoop 3 & GCS Connector: Spark 3.x (bao gồm 3.5.x) được xây dựng (built) để hoạt động với hệ sinh thái Hadoop 3.x. Điều này là yêu cầu bắt buộc để sử dụng thư viện gcs-connector-hadoop3-2.2.29.jar. Thư viện này là cầu nối duy nhất cho phép Spark đọc/ghi dữ liệu trực tiếp từ Google Cloud Storage (GCS) bằng đường dẫn gs://. Sử dụng Spark 2.x (vốn dùng Hadoop 2.x) sẽ không tương thích.
- Tương thích Python 3 (PySpark): Dòng Spark 3.5 cung cấp hỗ trợ mạnh mẽ và đầy đủ cho các phiên bản Python 3 hiện đại (như 3.9, 3.10, 3.11). Điều này rất quan trọng vì đồ án sử dụng các thư viện AI/ML mới (TensorFlow, Keras) vốn yêu cầu các phiên bản Python mới này.

Xét về hiệu năng và tối ưu hóa cho AI/ML:

- Pandas UDFs & Apache Arrow: Script phân loại ảnh Chó/Mèo phụ thuộc rất nhiều vào @pandas\_udf để chạy mô hình ResNet50 song song trên các lô (batches) ảnh. Spark 3.5 đã tối ưu hóa mạnh mẽ cơ chế này bằng cách sử dụng Apache Arrow, giúp giảm chi phí "giao tiếp" (serialization) giữa Python (chạy ResNet50) và JVM (chạy Spark), mang lại hiệu suất vượt trội.
- API pyspark.ml hoàn thiện: Toàn bộ đồ án sử dụng API học máy dựa trên DataFrame (pyspark.ml), bao gồm cả Pipeline. Dòng Spark 3.x đã hoàn thiện

API này, trong khi các phiên bản cũ hơn (2.x) vẫn còn nhiều tính năng trên API pyspark.mllib (dựa trên RDD) cũ kỵ.

Rủi ro về "Breaking Changes": Việc nâng cấp giữa các phiên bản lớn (ví dụ: từ 3.x lên 4.x) thường chứa "breaking changes". Điều này có nghĩa là các hàm (API) hoặc cấu hình mà nhóm đã sử dụng trong code 3.5.7 có thể bị thay đổi tên hoặc thậm chí bị xóa bỏ trong 4.0. Khi đó, toàn bộ code PySpark sẽ bị lỗi và phải viết lại, tốn rất nhiều thời gian và công sức kiểm thử.

## 2.2. Công nghệ 2: Google Cloud Platform (GCP)

### 2.2.1. Khái niệm

Google Cloud Platform (GCP), thường được gọi là Google Cloud, là một bộ các dịch vụ điện toán đám mây được cung cấp bởi Google. Nền tảng này chạy trên cùng một cơ sở hạ tầng mà Google sử dụng nội bộ cho các sản phẩm dành cho người dùng cuối của mình, chẳng hạn như Google Search, Gmail, và YouTube.

Về bản chất, Google Cloud cung cấp một loạt các dịch vụ đám mây theo dạng module, bao gồm các lĩnh vực chính như tính toán, lưu trữ dữ liệu, phân tích dữ liệu và học máy (machine learning). Trong bối cảnh của đồ án này, chúng ta tập trung vào mô hình Hạ tầng như một Dịch vụ (Infrastructure as a Service - IaaS). Mô hình này cho phép người dùng thuê và quản lý các tài nguyên hạ tầng cơ bản như máy ảo (qua dịch vụ Compute Engine), lưu trữ (qua Cloud Storage), và mạng ảo (qua VPC), từ đó có thể tự xây dựng và triển khai các môi trường ứng dụng tùy chỉnh.

### 2.2.2. Đặc điểm

Quản lý (Managed): Phần lớn các dịch vụ đều được Google quản lý, giúp giảm gánh nặng vận hành.

Thanh toán theo mức dùng (Pay-as-you-go): Chỉ trả tiền cho những tài nguyên (CPU, RAM, lưu trữ) thực sự sử dụng.

Khả năng mở rộng (Scalable): Dễ dàng tăng hoặc giảm tài nguyên (ví dụ: thêm/bớt máy ảo) chỉ bằng vài cú nhấp chuột hoặc dòng lệnh.

An toàn và đáng tin cậy: Thừa hưởng hạ tầng mạng lưới và bảo mật toàn cầu của Google.

### 2.2.3. Khả năng

Cung cấp hàng trăm dịch vụ, trong đó các dịch vụ chính được sử dụng trong đồ án bao gồm:

- Google Compute Engine (GCE): Cung cấp máy ảo (VMs) để chạy Spark Master, Worker, và API Server.
- Google Cloud Storage (GCS): Dịch vụ lưu trữ đối tượng.
- VPC & Firewall: Quản lý mạng và bảo mật.

- Cloud Load Balancing: Phân tải cho cụm API Server.

#### **2.2.4. Chức năng và nhiệm vụ**

- Là nền tảng hạ tầng (Infrastructure-as-a-Service - IaaS) cho toàn bộ đồ án.
- Cung cấp các máy ảo (VMs) để cài đặt và chạy cụm Spark (Master, Worker) và cụm API Server.
- Cung cấp hệ thống mạng (VPC, Subnet) và tường lửa (Firewall Rules) để các thành phần giao tiếp an toàn với nhau.
- Cung cấp Load Balancer để phân phối traffic đến các API server, đảm bảo tính sẵn sàng cao (HA).

#### **2.2.5. Lý do chọn**

Việc lựa chọn Google Cloud làm nền tảng hạ tầng cho đồ án này, thay vì tiếp tục với phương án tự xây dựng OpenStack trên máy cá nhân, được dựa trên các lý do chiến lược sau:

- Tập trung vào Trọng tâm Đồ án: Lý do quan trọng nhất là để trừu tượng hóa sự phức tạp của việc thiết lập và bảo trì lớp IaaS. Quá trình cài đặt OpenStack bằng DevStack đã cho thấy nhiều khó khăn liên quan đến cấu hình mạng, xung đột phần mềm và tính không ổn định, vốn là một dự án phức tạp theo đúng nghĩa của nó. Bằng cách sử dụng một nền tảng IaaS ổn định và được quản lý bởi Google, đồ án có thể tập trung hoàn toàn vào mục tiêu chính: xây dựng, kiểm thử và phân tích hiệu năng của hệ thống tính toán phân bố ứng dụng (pipeline Spark ML) chạy trên hạ tầng đó.
- Độ tin cậy và Ông định Vượt trội: Google Cloud cung cấp một hạ tầng cấp sản xuất, có độ sẵn sàng cao và được quản lý bởi các đội ngũ kỹ sư chuyên nghiệp của Google. Điều này loại bỏ hoàn toàn các rủi ro về lỗi hạ tầng do phần cứng cá nhân, sự bất ổn của mạng WiFi, hay các lỗi cấu hình phức tạp gây ra—những vấn đề đã gặp phải trong phương án trước.
- Hiệu quả Chi phí cho Sinh viên: Google Cloud cung cấp gói dùng thử miễn phí với 300 USD tín dụng cho khách hàng mới, có giá trị trong 90 ngày. Khoản tín dụng này đủ lớn để trang trải toàn bộ chi phí hạ tầng cho quy mô của một đồ án sinh viên, biến nó thành một lựa chọn khả thi về mặt tài chính mà không cần đầu tư vào phần cứng cá nhân.
- Tính thực tiễn và Phù hợp với Ngành công nghiệp: Việc triển khai các hệ thống dữ liệu lớn trên các nền tảng đám mây công cộng (GCP, AWS, Azure) là tiêu chuẩn của ngành công nghiệp hiện nay. Việc tích lũy kinh nghiệm thực tế trên Google Cloud là một kỹ năng có giá trị cao, giúp hồ sơ năng lực của sinh viên trở nên nổi bật và phù hợp hơn với yêu cầu của thị trường lao động.
- Hiệu năng và Khả năng Mở rộng Thực chất: Hạ tầng của Google Cloud cung cấp hiệu năng (mạng tốc độ cao, I/O lưu trữ nhanh) và khả năng mở rộng mà

một hệ thống tự động trên hai laptop không thể đạt được. Điều này cho phép thực hiện các kịch bản kiểm thử hiệu năng một cách có ý nghĩa hơn, thể hiện đúng tiềm năng của một hệ thống phân tán khi được bổ sung thêm tài nguyên.

#### 2.2.6. Lý do vì sao đã học AWS (Amazon Web Services) nhưng chọn GCP?

Hệ sinh thái AI và Machine Learning (ML) vượt trội: Đây gần như là lý do quan trọng nhất cho đồ án (đặc biệt là script ResNet50):

- Google là nơi phát minh ra TensorFlow, thư viện cốt lõi mà nhóm dùng để chạy ResNet50. Do đó, các dịch vụ AI của GCP (như Vertex AI, AI Platform) được tối ưu hóa sâu sắc và tích hợp liền mạch với TensorFlow.
- Google thiết kế TPU (Tensor Processing Units), là các vi mạch chuyên dụng cho việc huấn luyện (training) và suy luận (inference) các mô hình Deep Learning. Đối với các mô hình lớn như ResNet, TPUs cung cấp hiệu năng vượt trội (thường nhanh và rẻ hơn) so với GPU (mà AWS chủ yếu cung cấp).

Dịch vụ dữ liệu lớn và phân tích (Big Data): Google là "cái nôi" của nhiều công nghệ Big Data (như MapReduce, Bigtable). Các dịch vụ dữ liệu của họ phản ánh chuyên môn này:

- Google BigQuery: Thường được coi là dịch vụ kho dữ liệu (Data Warehouse) trên đám mây nhanh nhất và dễ sử dụng nhất.
- Google Dataproc: Dịch vụ Spark và Hadoop được quản lý của GCP. Nó nổi tiếng với khả năng khởi động cụm cực kỳ nhanh (thường dưới 90 giây), giúp tiết kiệm chi phí đáng kể so với AWS EMR (dịch vụ Spark của AWS) vốn mất nhiều thời gian hơn để khởi tạo.
- Google Dataflow: Dịch vụ xử lý dòng (streaming) và lô (batch) dựa trên Apache Beam, rất mạnh mẽ.

Cả Google Cloud (GCP) và AWS đều có các chương trình "miễn phí" hấp dẫn, nhưng chúng hoạt động theo cách hoàn toàn khác nhau:

- GCP cung cấp một gói Free Trial với \$300 tín dụng miễn phí. Số tiền này có giá trị sử dụng trong 90 ngày (3 tháng), có thể dùng \$300 này để thử nghiệm bất kỳ dịch vụ nào của GCP, bao gồm cả các dịch vụ đắt tiền như máy ảo (VM) cấu hình cao, GPU, cụm Kubernetes (GKE), hay BigQuery.
- Mô hình của AWS khác. Họ tập trung vào Free Tier 12 tháng và cấp \$100 cho sinh viên. Người dùng được cho một *danh sách các dịch vụ cụ thể* với giới hạn sử dụng hàng tháng, nếu tiêu hết giới hạn miễn phí và bị tính phí **ngay lập tức**.

### 2.3. Công nghệ 3: Google Cloud Storage (GCS)

### **2.3.1. Khái niệm**

Google Cloud Storage (GCS) là một dịch vụ lưu trữ đối tượng (object storage) trên nền tảng Google Cloud. Nó cho phép lưu trữ và truy xuất bất kỳ lượng dữ liệu nào, bất cứ lúc nào, từ bất kỳ đâu trên thế giới.

### **2.3.2. Đặc điểm**

Linh hoạt vô hạn: Không giới hạn dung lượng lưu trữ.

Độ bền cao: Dữ liệu được sao chép ra nhiều vị trí địa lý để đảm bảo độ bền (99.99%).

Truy cập toàn cầu: Dữ liệu có thể được truy cập từ bất kỳ đâu.

Tách biệt lưu trữ và tính toán: Đây là đặc điểm quan trọng. Dữ liệu (GCS) và tính toán (VMs/Spark) là hai dịch vụ riêng biệt.

### **2.3.3. Khả năng**

Lưu trữ bất kỳ loại file nào. Trong đó án, nó được dùng để lưu:

File dữ liệu thô (.csv).

File ảnh (.jpg, .png).

File script (.py).

Các thư mục chứa model đã huấn luyện (do Spark/MLlib lưu lại).

File log và kết quả (.json).

### **2.3.4 Chức năng và nhiệm vụ**

Đóng vai trò là Data Lake (hồ dữ liệu) và Kho lưu trữ mô hình (Model Registry) trung tâm.

Là nơi chứa dữ liệu đầu vào (telco\_churn.csv, PetImages/\*) cho Spark.

Cung cấp cơ chế xác thực IAM (Identity and Access Management): Service Account (SA) là một dạng tài khoản đặc biệt được dùng bởi ứng dụng hoặc máy ảo (VM instance) để xác thực và truy cập tài nguyên GCP một cách an toàn mà không cần tài khoản người dùng cá nhân.

Là nơi Spark lưu kết quả đầu ra (các mô hình telco\_rf, telco\_gbt, dogcat\_lr\_model).

Là nơi API Server tải mô hình (model.load()) về để thực hiện dự đoán.

Cung cấp Google Cloud Identity-Aware Proxy (IAP) TCP Forwarding (tunneling). Đây là một tính năng của dịch vụ IAP, cho phép người dùng được ủy quyền kết nối an toàn với các tài nguyên Compute Engine (như máy ảo - VM) nằm trong mạng riêng.

(private network) của Google Cloud mà không cần gán địa chỉ IP công cộng (public IP) hoặc sử dụng VPN hay phải SSH dựa vào Key.

### **2.3.5. Lý do chọn Google Cloud Storage trong mô hình đồ án**

GCS được chọn thay vì hệ thống file truyền thống (như HDFS) vì nó tách biệt lưu trữ khỏi tính toán. Cụm Spark có thể đọc dữ liệu từ GCS, thực hiện tính toán, ghi kết quả trả lại GCS, và sau đó tắt đi để tiết kiệm chi phí. Nếu dùng HDFS, dữ liệu sẽ gắn liền với cụm Spark, khiến việc tắt cụm sẽ làm mất dữ liệu.

## **2.4. Công nghệ 4: GCS Connector (hadoop3-2.2.29)**

### **2.4.1. Khái niệm**

GCS Connector là một thư viện Java (.jar) cho phép các ứng dụng dựa trên hệ sinh thái Hadoop (bao gồm cả Apache Spark) có thể truy cập trực tiếp vào Google Cloud Storage (GCS) như một hệ thống file tương thích với Hadoop (HDFS).

### **2.4.2. Đặc điểm**

Hoạt động như một "trình điều khiển" (driver) hoặc "cầu nối".

Phiên bản hadoop3-2.2.29 là phiên bản tương thích với Hadoop 3.x, phù hợp với các phiên bản Spark mới.

### **2.4.3. Khả năng**

Cho phép Spark hiểu và sử dụng đường dẫn có tiền tố gs:// (ví dụ: gs://nt533q13-spark-data/data/...) giống như cách nó hiểu hdfs:// hoặc file:///.

### **2.4.4. Chức năng và nhiệm vụ trong đồ án**

Là thành phần bắt buộc để kết nối Spark với GCS.

Nó được khai báo trong lệnh spark-submit (qua cờ --conf spark.jars=...) để Spark tải thư viện này khi khởi động.

Nó cho phép các lệnh như spark.read.csv("gs://...") và model.save("gs://...") hoạt động thành công.

## **2.5. Công nghệ 5: Apache ZooKeeper 3.7.1**

### **2.5.1. Khái niệm**

Apache ZooKeeper là một dịch vụ điều phối tập trung (centralized coordination service) cho các ứng dụng phân tán. Nó cung cấp một không gian tên (namespace) phân cấp đơn giản và duy trì các thông tin cấu hình, đặt tên, và đồng bộ hóa nhóm.

### **2.5.2. Đặc điểm**

Độ tin cậy cao: ZooKeeper thường chạy ở chế độ "ensemble" (một cụm 3, 5, hoặc 7 nút). Hệ thống vẫn hoạt động miễn là đa số các nút (ví dụ: 2/3 nút) còn sống.

**Đơn giản:** Kiến trúc file-system-like (giống hệ thống file) đơn giản để lưu trữ dữ liệu (gọi là znodes).

**Leader Election:** Có cơ chế bầu cử "lãnh đạo" (leader) mạnh mẽ.

### **2.5.3. Khả năng**

Cung cấp các cơ chế cơ bản cho hệ thống phân tán:

Đồng bộ hóa (Synchronization).

Phát hiện lỗi (Failure Detection).

Bầu cử lãnh đạo (Leader Election).

Quản lý cấu hình (Configuration Management).

### **2.5.4. Chức năng và nhiệm vụ trong đồ án**

**Nhiệm vụ cốt lõi:** Đảm bảo Tính sẵn sàng cao (High Availability - HA) cho Spark Master.

Trong đồ án, 3 máy ảo Spark Master được thiết lập (spark-master-a, b, c).

Cụm ZooKeeper (chạy trên 3 máy chủ) sẽ giám sát 3 Master này.

Nó tổ chức một cuộc bầu cử và chỉ định một Master là ACTIVE (đang hoạt động), 2 Master còn lại ở chế độ STANDBY.

Nếu Master ACTIVE bị lỗi (ví dụ: VM bị crash), ZooKeeper sẽ ngay lập tức phát hiện và bầu một trong hai Master STANDBY lên làm ACTIVE mới. Các Spark Worker sẽ tự động kết nối đến Master mới này và tiếp tục công việc.

### **2.5.5. Lý do lựa chọn**

ZooKeeper được chọn để loại bỏ Điểm lỗi đơn (Single Point of Failure - SPOF) của Spark Master. Trong một hệ thống Spark mặc định (không có HA), nếu máy Master duy nhất bị hỏng, toàn bộ cụm sẽ ngừng hoạt động. Bằng cách sử dụng ZooKeeper, đồ án đảm bảo rằng cụm Spark vẫn vận hành ổn định ngay cả khi một máy Master gặp sự cố.

## **2.6. Công nghệ 6: Python 3 & pip3**

### **2.6.1. Khái niệm**

**Python 3:** Là một ngôn ngữ lập trình thông dịch, cấp cao, đa mục đích, nổi tiếng với cú pháp rõ ràng, dễ đọc.

**pip3:** Là trình quản lý gói (package manager) mặc định cho Python 3. Nó dùng để cài đặt và quản lý các thư viện bên thứ ba.

### **2.6.2. Đặc điểm**

**Hệ sinh thái phong phú:** Có một kho thư viện khổng lồ (PyPI) cho mọi tác vụ.

"Ngôn ngữ của Khoa học Dữ liệu": Là ngôn ngữ thống trị trong lĩnh vực AI và Machine Learning.

Dễ tích hợp: Dễ dàng kết nối các hệ thống khác nhau.

### **2.6.3. Khả năng**

Viết logic xử lý dữ liệu (PySpark).

Xây dựng mô hình Deep Learning (TensorFlow, Keras).

Xây dựng API (Flask, FastAPI).

Thao tác dữ liệu (Pandas, NumPy).

### **2.6.4. Chức năng và nhiệm vụ trong đồ án**

Ngôn ngữ lập trình chính: Toàn bộ logic của các kịch bản spark-submit đều được viết bằng Python (PySpark).

Thư viện ML: Cung cấp các thư viện (tensorflow, keras, Pillow, numpy) được sử dụng trong script phân loại ảnh để tải ResNet50 và xử lý ảnh.

API Server: Ngôn ngữ dùng để xây dựng API server (ví dụ: dùng Flask) nhận request dự đoán.

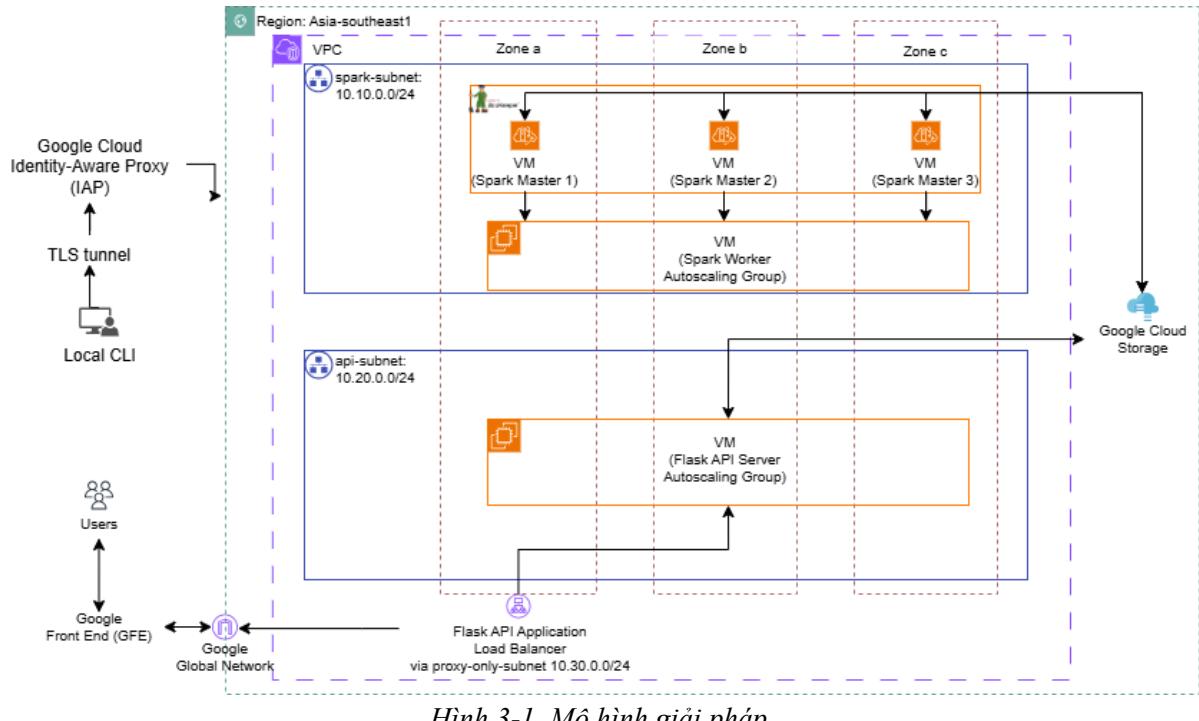
Quản lý thư viện: pip3 được dùng để cài đặt các thư viện cần thiết (ví dụ: pip3 install tensorflow flask) trên các máy ảo (image).

### **2.6.5. Lý do lựa chọn**

Python 3 được chọn vì đây là tiêu chuẩn công nghiệp cho Khoa học Dữ liệu và AI. Giao diện PySpark của Spark rất hoàn thiện, và quan trọng nhất, nó cho phép tích hợp liền mạch với các thư viện AI/ML hàng đầu như TensorFlow/Keras (điều mà Scala/Java làm rất khó khăn), như đã thấy trong script phân loại ảnh.

## CHƯƠNG 3. HIỆN THỰC HOÁ

### 3.1. Mô hình



Hình 3-1. Mô hình giải pháp

Cấp độ	Thành phần	Mô tả / Vai trò	Phạm vi / Cấu hình chính
1. Region / Zone	<b>Region: asia-southeast1</b>	Khu vực triển khai (ví dụ: Singapore)	Gồm 3 zone: asia-southeast1-a, asia-southeast1-b, asia-southeast1-c
	<b>Zone a, b, c</b>	Phân phối VM để đảm bảo HA (High Availability)	Spark Master và Worker, Flask Server nằm trên nhiều zone
2. Network Layer (VPC)	<b>VPC Network</b>	Mạng ảo chính chứa toàn bộ hạ tầng	Phân chia thành nhiều subnet
3. Subnet	<b>spark-subnet (10.10.0.0/24)</b>	Chứa cụm Spark (Master, Worker) và Bastion host	Internal subnet, không public
	<b>api-subnet (10.20.0.0/24)</b>	Chứa cụm Flask API Server autoscaling	Internal subnet, dùng load balancer để public API
	<b>proxy-only-subnet (10.30.0.0/24)</b>	Subnet riêng cho Load Balancer proxy	Dùng bởi Google Cloud Load Balancer
4. Compute Layer (VM Instances)	<b>Spark Master 1 / 2 / 3</b>	3 node master để tạo cụm Spark HA (Zookeeper hoặc standby mode)	Mỗi VM ở một zone khác nhau (1a, 1b, 1c)

	<b>Flask API</b>	Cụm VM chạy Flask API (backend phục vụ user requests)	Autoscaling theo request, dùng Ray để xử lý song song
<b>5. Middleware / Processing Layer</b>	<b>Apache Spark Cluster</b>	Xử lý dữ liệu lớn (big data / ML training)	3 Master + N Worker cấu hình cluster mode
	<b>Ray Cluster</b>	Xử lý song song, phân tán các tác vụ nặng từ Flask	Tích hợp trong cụm Flask API
<b>6. Storage Layer</b>	<b>Google Cloud Storage (GCS)</b>	Lưu trữ model, dữ liệu training, log	Truy cập từ Spark và Flask qua service account
<b>7. Load Balancing / External Access</b>	<b>Google Cloud Load Balancer</b>	Phân phối request đến các Flask API VM	Public IP → proxy-only-subnet → api-subnet
	<b>Google Edge Network &amp; Front End (GFE)</b>	Mạng biên toàn cầu của Google, phục vụ người dùng	Kết nối user → GFE → Load Balancer
<b>8. Access / Admin Layer</b>	<b>Admin Console</b>	Người quản trị điều khiển toàn bộ hạ tầng qua SSH, GCP console	Truy cập qua Bastion Host hoặc IAM roles
<b>9. Security / IAM</b>	<b>Service Account Roles</b>	Cho phép Spark và Flask truy cập GCS, API, logs	Gắn vào VM template hoặc instance group
<b>10. Scaling &amp; Availability</b>	<b>Autoscaling Policies</b>	Tự động thêm/bớt VM khi tải cao	Flask API: theo CPU hoặc request load; Spark Worker: theo job queue
	<b>Zookeeper / Spark Standby Master</b>	Đảm bảo cụm Spark có HA, tránh single point of failure	Zookeeper cluster có thể chạy cùng Spark Master

Bảng 3-1. Bảng thành phần của mô hình

### 3.2. Các giải thuật đã sử dụng

#### 3.2.1. Giải thuật Phân loại (Classification)

Đây là nhóm giải thuật Học có giám sát (Supervised Learning). Mục tiêu là dự đoán một nhãn (label) hoặc một danh mục (category) cụ thể (ví dụ: "Rời đi" / "Ở lại", "Chó" / "Mèo").

- RandomForestClassifier (Rừng Ngẫu nhiên)
  - Ý tưởng: Hoạt động giống như một cuộc "bỏ phiếu dân chủ".

- Cách chạy: Nó xây dựng hàng trăm "cây quyết định" (Decision Trees) một cách độc lập và song song. Mỗi cây sẽ đưa ra một dự đoán riêng (ví dụ: "Chó" hoặc "Mèo").
- Kết quả: Kết quả cuối cùng là dự đoán được nhiều cây "bỏ phiếu" nhất.
- Đặc điểm: Rất ổn định, khó bị overfitting (học vẹt), và chạy tương đối nhanh trên Spark (vì các cây có thể được xây dựng song song).
- GBTClassifier (Cây Tăng cường Gradient)
  - Ý tưởng: Hoạt động giống như một nhóm chuyên gia "sửa lỗi" cho nhau.
  - Cách chạy: Nó cũng xây dựng nhiều cây, nhưng theo kiểu tuần tự.
  - Cây 1 được xây dựng và đưa ra dự đoán (có thể sai nhiều).
  - Cây 2 được xây dựng, nhưng nó tập trung vào việc sửa các lỗi mà Cây 1 đã dự đoán sai.
  - Cây 3 sửa lỗi cho Cây 2, và cứ thế...
  - Đặc điểm: Thường cho độ chính xác cao hơn RandomForest, nhưng huấn luyện chậm hơn (vì phải làm tuần tự) và dễ bị overfitting nếu không tinh chỉnh tham số cẩn thận (maxDepth, stepSize).
- LogisticRegression (Hồi quy Logistic)
  - Ý tưởng: Tìm một đường ranh giới (hoặc mặt phẳng) để phân chia 2 nhóm dữ liệu.
  - Cách chạy: Đây là một mô hình toán học (thống kê) tìm ra một phương trình tuyến tính (ví dụ:  $y = w_1x_1 + w_2x_2 + \dots$ ) để phân tách tốt nhất hai lớp (ví dụ: Chó và Mèo).
  - Đặc điểm: Huấn luyện cực kỳ nhanh. Nó là mô hình "đơn giản", nhưng trở nên rất mạnh mẽ khi được kết hợp với các đặc trưng (features) phức tạp.
  - Đây là lựa chọn tốt để chạy sau ResNet50. ResNet50 thực hiện công việc nặng nhọc là biến ảnh thành vector 2048 chiều; LogisticRegression chỉ làm nhiệm vụ đơn giản là "kẻ một đường" trong không gian 2048 chiều đó để tách Chó/Mèo.

### 3.2.2. Giải thuật Phân cụm (Clustering)

Đây là nhóm giải thuật Học không giám sát (Unsupervised Learning). Giải thuật này không có label để dự đoán. Thay vào đó, yêu cầu mô hình: "Hãy tự tìm các nhóm (clusters) tương đồng trong dữ liệu này."

#### KMeans

- Ý tưởng: Phân chia dữ liệu thành k nhóm, sao cho các điểm trong cùng một nhóm ở gần nhau nhất có thể.

- Cách chạy:
  - Phải chỉ định trước số cụm (ví dụ: k=5).
  - Thuật toán sẽ ngẫu nhiên tạo ra 5 "tâm cụm" (centroids).
  - Nó lặp đi lặp lại 2 bước:
    - Gán mỗi điểm dữ liệu (mỗi khách hàng) vào "tâm cụm" gần nó nhất (dựa trên khoảng cách).
    - Cập nhật lại vị trí "tâm cụm" bằng cách lấy trung bình của tất cả các điểm vừa được gán vào nó.

### 3.2.3. Giải thuật mapPartition

Trong các bài toán học máy, đặc biệt là với dữ liệu phi cấu trúc như hình ảnh, bước trích xuất đặc trưng (featurization) thường rất tốn kém về mặt tính toán.

Lấy ví dụ kịch bản phân loại ảnh Chó/Mèo, chúng ta cần dùng mô hình ResNet50 (một mô hình Deep Learning nặng) để biến đổi từng bức ảnh thành một vector 2048 chiều.

Nếu sử dụng phép biến đổi map() thông thường, Spark sẽ phải:

- Tải mô hình ResNet50 (nặng hàng trăm MB) vào bộ nhớ.
- Xử lý 1 bức ảnh.
- Giải phóng mô hình khỏi bộ nhớ.
- Lặp lại quá trình này cho từng bức ảnh trong hàng triệu bức ảnh.

Điều này dẫn đến chi phí (overhead) khởi tạo tài nguyên là cực kỳ lớn và không hiệu quả. Để giải quyết vấn đề này, Spark cung cấp một phép biến đổi hiệu suất cao là mapPartitions.

mapPartitions là một phép biến đổi (transformation) trong Spark, nó thực thi một hàm tùy chỉnh trên từng phân vùng (partition) của RDD/DataFrame thay vì trên từng hàng. Thay vì chạy hàm 1 triệu lần cho 1 triệu hàng, mapPartitions chỉ chạy hàm đó (ví dụ) 8 lần nếu dữ liệu được chia thành 8 phân vùng.

## 3.3. Các bước thực hiện

Quá trình triển khai đề tài của nhóm tuân thủ phương pháp luận "phát triển cục bộ, triển khai trên đám mây" (local-first development), cho phép gỡ lỗi và kiểm thử logic nhanh chóng trước khi chuyển sang môi trường phân tán quy mô lớn.

Lộ trình được chia thành 4 giai đoạn chính:

- Giai đoạn 1 (Local): Phát triển và kiểm thử các pipeline Machine Learning trên máy cá nhân.

- Giai đoạn 2 (Cloud Infra): Xây dựng hạ tầng mạng và máy ảo trên Google Cloud.
- Giai đoạn 3 (Cloud Deployment): Cài đặt cụm Spark và triển khai pipeline ML lên hạ tầng cloud đã chuẩn bị.
- Giai đoạn 4 (End-to-End Deployment): Triển khai API phục vụ dự đoán dựa vào mô hình đã huấn luyện.

### 3.3.1. Train model trên môi trường local

Bên dưới đây, nhóm báo cáo lại quá trình thử nghiệm model với các trường hợp sau đây:

- Bài toán Classification:
  - Dự đoán khách hàng rời đi, dataset nhỏ, giải thuật Random Forest.
  - Dự đoán khách hàng rời đi, dataset lớn, giải thuật Random Forest.
  - Dự đoán khách hàng rời đi, dataset lớn, giải thuật GBT.
- Bài toán Cluster (hướng đi mở rộng): Phân loại khách hàng theo nhóm, dataset lớn, giải thuật KMeans.
- Bài toán nâng cao: Phân loại 1 tập dữ liệu hình ảnh lớn theo phương pháp học chuyển giao (Transfer Learning), mô hình học sâu ResNet50.

Môi trường thực hiện:

- Thiết bị:
- Phần mềm: Hệ điều hành, Python 3.x, Jupyter Notebook/VS Code, pip install pyspark.

Các bước thực hiện:

- Cài Java và Python:

```
sudo apt install -y openjdk-11-jdk python3 python3-pip python3-venv wget unzip
```

- Cài thư viện Python ML:

```
python3 -m venv venv
source venv/bin/activate
pip install -U pip
pip install pyspark pandas scikit-learn
```

- Tải Apache Spark và giải nén:

```
wget https://downloads.apache.org/spark/spark-3.5.2/spark-3.5.2-bin-hadoop3.tgz
tar -xvzf spark-3.5.2-bin-hadoop3.tgz
```

- Chuyển đến folder mong muốn:

```
sudo mv spark-3.5.2-bin-hadoop3 /opt/spark
```

- Thiết lập biến môi trường:

```
echo 'export SPARK_HOME=/opt/spark' >> ~/.bashrc
echo 'export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin' >> ~/.bashrc
echo 'export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc
source ~/.bashrc
```

- Check lại:

```
spark-shell
```

### a) Dự đoán khách hàng rời đi, dataset nhỏ, giải thuật Random Forest.

Ban đầu, tiến hành train model với script và dataset cơ bản để xem spark có hoạt động đúng hay không. Tạo file dataset mẫu nhỏ và cơ bản:

```
mkdir -p ~/distributed-ml/data
cat > ~/distributed-ml/data/churn.csv << 'EOF'
gender,contract,age,tenure,monthly_charges,churn
Male,Month-to-month,30,10,70.5,1
Female,Two year,45,36,50.2,0
Male,Month-to-month,29,5,75.0,1
Female,One year,34,15,60.3,0
Male,Two year,50,40,55.0,0
EOF
```

Script train model đơn giản và giải thích một số code quan trọng trong script:

```
spark = SparkSession.builder.appName("TelcoChurn").master("local[*]").getOrCreate()
```

- Local: Dùng để yêu cầu Spark chạy ở chế độ local (cục bộ), nghĩa là trên máy đang sử dụng, dùng tất cả các nhân (cores) CPU có sẵn.

```
df = spark.read.option("header", True).option("inferSchema", True).csv("data/telco_churn.csv")
```

- + Sử dụng dataset được lưu trong file .csv, ở đây là data/telco\_churn.csv

```
df = df.dropna(how='any')
```

- + Làm sạch dữ liệu trước khi đưa vào train. Nó sẽ xóa bất kỳ hàng nào có chứa giá trị NULL (rỗng) ở bất kỳ cột nào.

```
df = df.withColumn("label", (df["Churn"] == "Yes").cast("integer"))
```

- + Tạo cột mục tiêu label. Nếu cột Churn là "Yes", label sẽ là 1; ngược lại, label sẽ là 0.

```
categorical_cols = ['gender','Partner','Dependents','Contract','InternetService']
```

```
numeric_cols = ['SeniorCitizen','tenure','MonthlyCharges','TotalCharges']
```

- Đây là phần chuẩn bị những đặc trưng (Feature Engineering). Dùng để biến đổi dữ liệu thô thành dạng vector vì mô hình chỉ có thể hiểu dữ liệu dưới dạng vector 1,0. Trong đó, categorical là dạng chữ và danh mục, numeric là dạng số.
- Ở đây chỉ chọn 5 cột categorical để phân loại và hiển thị dữ liệu, các cột không được chọn sẽ mặc định là được bỏ qua và không được huấn luyện.

```
indexers = [StringIndexer(inputCol=c, outputCol=c+"_idx") for c in categorical_cols]
```

```
encoders = [OneHotEncoder(inputCol=c+"_idx", outputCol=c+"_vec") for c in categorical_cols]
```

- StringIndexer để chuyển đổi các cột chữ thành số (ví dụ male thành 0,0 và female thành 1,0)
- OneHotEncoder lấy các cột chỉ số đó và biến chúng thành vector nhị phân. Điều này giúp mô hình không hiểu nhầm rằng giá trị số lớn hơn là quan trọng hơn.

```
assembler = VectorAssembler(
```

```
    inputCols=numeric_cols + [c+"_vec" for c in categorical_cols],
```

```
    outputCol="features"
```

```
)
```

- Lấy các cột num và vec vừa được tạo ra từ OHE gom lại thành 1 vector duy nhất. Đây mới chính là đầu vào của mô hình Spark ML.

```
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=100)
```

- Khai báo thuật toán đang sử dụng là RandomForest, kèm theo đó là các trường và thuộc tính được chỉ định để train model.

```
pipeline = Pipeline(stages=indexers + encoders + [assembler, rf])
```

- Khai báo pipeline để liệt kê tất cả các bước (Indexers, Encoders, Assembler, và mô hình RF) thành 1 mô hình duy nhất.

```
train, test = df.randomSplit([0.8, 0.2], seed=42)
```

- Model chia data thành 2 phần: train chiếm 80% để huấn luyện và test chiếm 20% để kiểm tra.

```
model = pipeline.fit(train)
```

- Lệnh bắt đầu huấn luyện. Spark sẽ chạy toàn bộ các bước trong pipeline trên dữ liệu train.

```
predictions = model.transform(test)
```

- Sử dụng mô hình đã huấn luyện trên (model) để dự đoán trên tập test.

```
evaluator = BinaryClassificationEvaluator(labelCol="label",
                                         metricName="areaUnderROC")
```

```
auc = evaluator.evaluate(predictions)
```

```
print(f"AUC = {auc:.4f}")
```

- BinaryClassificationEvaluator: Tiến hành tạo 1 công cụ đo lường hiệu suất hoạt động của model vừa train.
- Chọn công cụ AUC. Đây là một chỉ số tốt để đánh giá khả năng phân loại của mô hình (1.0 là hoàn hảo, 0.5 là vô dụng)
- In kết quả ra màn hình

Chạy script:

```
spark-submit train_ml.py
```

Kết quả thu về:

```
AUC = 1.0000
✓ Model saved to models/churn_rf
group12@group12:~/distributed-ml/scripts$
```

Hình 3-2. Kết quả huấn luyện local

- Phân tích kết quả thu về: mô hình đã chạy thành công 100% trên Spark MLlib, và file mô hình đã được lưu lại. Cụ thể: **AUC (Area Under ROC Curve)** là thước đo độ chính xác của mô hình phân loại nhị phân.
  - Giá trị AUC nằm trong khoảng 0 → 1
  - Càng gần 1, mô hình phân biệt tốt giữa hai lớp (ở lại / rời bỏ).
  - AUC = 1.0 nghĩa là mô hình dự đoán hoàn hảo trên tập test.

→ Dựa trên kết quả nhận lại được, nhóm nhận thấy AUC = 1.0 đến từ nguyên nhân do dataset nhỏ nên xảy ra hiện tượng overfitting (quá khớp) vì dữ liệu quá đơn giản hoặc trùng lặp, nên mô hình học “thuộc lòng” dữ liệu.

### b) Dự đoán khách hàng rời đi, dataset lớn, giải thuật Random Forest.

Để đánh giá được hiệu quả hoạt động và tỉ lệ dự đoán của model, nhóm tiến hành nâng cấp script để thu thập kết quả theo nhiều trường dữ liệu hơn và dùng dataset lớn hơn để test model.

Clone dataset từ web về lưu vào file data:

```
wget https://raw.githubusercontent.com/IBM/telco-customer-churn-on-  
icp4d/master/data/Telco-Customer-Churn.csv -O telco_churn.csv
```

Script train model được cải tiến và giải thích một số phần quan trọng:

```
df = df.withColumn(  
    "TotalCharges",  
    when(col("TotalCharges") == "",  
        None).otherwise(col("TotalCharges")).cast("double")  
)
```

- Cải tiến hơn so với script trước vì: Thay vì xóa tất cả các hàng NULL (dropna(how='any')) như code cơ bản, ở script này, lập trình when(col("TotalCharges") == "", None) để nó phát hiện ra TotalCharges có các giá trị là chuỗi rỗng ("") và chủ động đổi chúng thành None (NULL).

```
df = df.na.drop(subset=["TotalCharges"])
```

- Chỉ xóa các hàng bị thiếu giá trị ở đúng cột TotalCharges. Điều này giữ lại được rất nhiều dữ liệu mà code 1 đã vứt bỏ một cách không cần thiết.

```
categorical_cols = [  
    'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',  
    'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',  
    'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',  
    'PaperlessBilling', 'PaymentMethod'  
,  
    numeric_cols = ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']
```

- Feature Engineering nhiều hơn code cơ bản (15 cột dữ liệu chữ và 4 cột số sơ bản).

```
rf = RandomForestClassifier(labelCol="label", featuresCol="features",
                           numTrees=100, maxDepth=8)
```

- Giới hạn độ sâu tối đa của mỗi cây là 8, ngăn overfitting dữ liệu. Giúp mô hình dự đoán chính xác hơn trên tập test.

```
evaluator = BinaryClassificationEvaluator(labelCol="label",
                                         metricName="areaUnderROC")

auc = evaluator.evaluate(predictions)

print(f"AUC = {auc:.4f}")
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

acc_eval = MulticlassClassificationEvaluator(labelCol="label",
                                             predictionCol="prediction", metricName="accuracy")

accuracy = acc_eval.evaluate(predictions)

print(f"⌚ Accuracy = {accuracy:.4f}")
```

- Script này đo lường cả AUC và accuracy: tỷ lệ dự đoán đúng của mô hình.

```
print("📋 Kết quả chi tiết (10 khách hàng đầu tiên):")

predictions.select("customerID", "gender", "Contract", "MonthlyCharges",
                    "Churn", "prediction", "probability").show(10, truncate=False)
```

- Hiển thị chi tiết 10 khách hàng đầu tiên với các trường đã list.

Tiến hành chạy và thu về được kết quả:

```
AUC = 0.8561
25/10/28 19:54:27 WARN DAGScheduler: Broadcasting large task binary with size 2002.2 KiB
⌚ Accuracy = 0.8107
25/10/28 19:54:29 WARN SparkStringutils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
25/10/28 19:54:29 WARN DAGScheduler: Broadcasting large task binary with size 1993.4 KiB
25/10/28 19:54:30 WARN DAGScheduler: Broadcasting large task binary with size 1993.4 KiB
⌚ Lại: 1086 khách hàng
⌚ Rõ: 256 khách hàng
Tổng cộng: 1342 khách hàng
✓ Model saved to /home/group12/distributed-ml/models/telco_rf

📋 Kết quả chi tiết (10 khách hàng đầu tiên):
25/10/28 19:54:53 WARN DAGScheduler: Broadcasting large task binary with size 1996.0 KiB
+-----+-----+-----+-----+-----+
|customerID|gender|Contract      |MonthlyCharges|Churn|prediction|probability
+-----+-----+-----+-----+-----+
|[0004-TLHLJ]Male |Month-to-month|73.9       |Yes |1.0      |[[0.3895611973615546, 0.6104388026384454]|
|[0013-SMEOE]Female|Two year     |109.7      |No  |0.0      |[[0.9307472344754055, 0.06925276552459457]|
|[0015-UOCO3]Female|Month-to-month|48.2       |No  |0.0      |[[0.689565256788864, 0.3104347432111359]|
|[0019-EFAEP]Female|Two year     |101.3      |No  |0.0      |[[0.9085253110613125, 0.09147468893868756]|
|[0023-HGHWL]Male |Month-to-month|25.1       |Yes |1.0      |[[0.2867637378483938, 0.7132362621516062]|
|[0030-FNXPP]Female|Month-to-month|19.85      |No  |0.0      |[[0.8349429516864127, 0.16505704831358725]|
|[0042-RLHYP]Female|Two year     |19.7       |No  |0.0      |[[0.9833801570549238, 0.016619842945076152]|
|[0057-QBUQH]Female|Two year     |25.1       |No  |0.0      |[[0.9516022617248515, 0.04839773827514853]|
|[0078-XZMHT]Male |Two year     |85.15      |No  |0.0      |[[0.9571116739410374, 0.04288832605896256]|
|[0080-EMYVV]Female|One year     |51.45      |No  |0.0      |[[0.8825635358866405, 0.11743646411335953]|
+-----+-----+-----+-----+-----+
only showing top 10 rows
```

Hình 3-3. Kết quả huấn luyện local

Đánh giá kết quả thu được:

- Kết quả 0.8561 (hay 85.61%) là một kết quả tốt, cho thấy mô hình có khả năng phân loại mạnh mẽ.
- Accuracy (Độ chính xác) = 0.8107 (81.07%): Chỉ số này cho biết tỷ lệ dự đoán đúng trên tổng số 1342 khách hàng trong tập dữ liệu

So sánh 2 đoạn script và 2 kết quả trên:

Tính năng	Code đã cải thiện	Code cơ bản	Tại sao code mới tốt hơn?
<b>Xử lý dữ liệu rỗng</b>	dropna(subset=["Total Charges"])	dropna(how='any')	Tốt hơn nhiều. Code 1 vứt bỏ mọi hàng chỉ cần thiếu 1 giá trị (ví dụ: Partner), làm mất dữ liệu. Code 2 chỉ vứt bỏ khi TotalCharges (cột quan trọng) bị thiếu.
<b>Lựa chọn đặc trưng</b>	15 cột categorical (dùng hết)	Chỉ 5 cột categorical (rất ít)	Code 2 sử dụng nhiều thông tin hơn (như Contract, TechSupport), giúp mô hình dự đoán chính xác hơn đáng kể.
<b>Cấu hình mô hình</b>	maxDepth=8	Không maxDepth	có Quan trọng nhất. Code 2 chủ động chống overfitting (học vẹt). Code 1 có rủi ro overfitting cao, tạo ra một mô hình có vẻ tốt nhưng dự đoán kém trong thực tế.
<b>Đánh giá mô hình</b>	AUC và Accuracy	Chỉ AUC	Code 2 cung cấp cái nhìn đa chiều hơn về hiệu suất của mô hình.
<b>Tính hoàn thiện</b>	Có lưu model (save()) Có in kết quả chi tiết (show()) Có thông kê (stay/leave)	Chỉ chạy và in AUC	Code 2 là một kịch bản hoàn chỉnh, sẵn sàng cho công việc thực tế (production). Code 1 chỉ là một thử nghiệm nhanh. Code 2 sẽ không bị crash nếu gặp dữ liệu lạ trong tương lai. Code 1 sẽ bị crash.
<b>Độ ổn định</b>	handleInvalid="keep"	Không có	

Hình 3-4. Bảng so sánh 2 script huấn luyện

Bên cạnh đó, để tối ưu AUC với các dataset lớn, từ đó đánh giá được giải thuật nào là tối ưu hơn, nhóm đã viết script để train model trên bằng giải thuật GBT. GBT cũng để chuẩn bị cho tiến hành bài toán Cluster.

### c) Dự đoán khách hàng rời đi, dataset lớn, giải thuật GBT.

Script train model với giải thuật GBT và giải thích 1 số điểm quan trọng:

```
gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=100,
maxDepth=5, stepSize=0.1)
```

- Khai báo mô hình train bằng thuật toán GBT với các tham số: số lần lặp tại là 100, độ sâu của mỗi cây là 5. Tỷ lệ học (learning rate), kiểm soát mức độ "sửa lỗi" của mỗi cây mới là 0.1.

```
start_time = time.time()
model = pipeline.fit(train)
train_time = time.time() - start_time
```

- start\_time = time.time(): Bắt đầu đo thời gian huấn luyện. train\_time = ...: Tính toán và in ra tổng thời gian huấn luyện.

```
evaluator = BinaryClassificationEvaluator(labelCol="label",
metricName="areaUnderROC")
auc = evaluator.evaluate(predictions)

print(f"\n ✅ Training completed in {train_time:.2f} seconds")
print(f" ✅ AUC = {auc:.4f}")

rdd = predictions.select("prediction", "label").rdd.map(tuple)
metrics = MulticlassMetrics(rdd)

accuracy = predictions.filter(col("label") == col("prediction")).count() / predictions.count()

print(f"\n Accuracy: {accuracy:.2%}")
print(f"Precision (rời đi): {metrics.precision(1):.2f}")
print(f"Recall (rời đi): {metrics.recall(1):.2f}")
print(f"F1-score (rời đi): {metrics.fMeasure(1):.2f}")
```

- Dự đoán AUC và Accuracy như code trước nhưng Accuracy được dự đoán bằng cách tính thủ công: (Số dự đoán đúng) / (Tổng số dự đoán).
- Để có cái nhìn tổng quát hơn về dữ liệu đầu ra, dùng MulticlassMetrics.
- metrics.precision(1): Precision (Độ chính xác) cho lớp "Rời đi" (label 1). Trong số tất cả khách hàng được dự đoán là "Rời đi", có bao nhiêu người thực sự "Rời đi"?
- metrics.recall(1): Recall (Độ nhạy) cho lớp "Rời đi". Trong số tất cả khách hàng thực sự "Rời đi", mô hình đã tìm thấy (dự đoán đúng) được bao nhiêu người?
- Trung bình của 2 tham số trên được tính và ghi vào fMeasure.

Tiến hành chạy và thu về được kết quả:

```

Training Gradient Boosted Tree model...
25/10/28 19:58:27 WARN InstanceBuilder: Failed to load implementation from: dev.ludovic.netlib.blas.JNIBLAS
✓ Training completed in 115.40 seconds
✗ AUC = 0.8326
/opt/spark/python/lib/pyspark.zip/pyspark/sql/context.py:158: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
25/10/28 19:58:33 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
Accuracy: 79.81%

```

Hình 3-5-4. Kết quả huấn luyện local

Đánh giá kết quả thu được:

So sánh giải thuật RF và GBT:

- Đánh giá chung, thông thường, RF sẽ train nhanh hơn GBT, nhưng GBT thường cho độ chính xác cao hơn nếu được tinh chỉnh cẩn thận, đặc biệt là với các dataset mất cân bằng như bài toán Churn.
- Lý do:
  - RF. Nó xây dựng 100 “cây” (hoặc numTrees) hoàn toàn độc lập và song song với nhau. Trong khi đó, GBT xây dựng các cây một cách tuần tự.
  - RF tận dụng được tối đa khả năng xử lý song song của Spark. Giống như giao 100 bài tập cho 100 người làm cùng lúc. Nên trong hệ thống được triển khai trong đồ án này, khi dùng giải thuật RF, Spark có thể giao cho mỗi worker xây dựng một vài cây cùng một lúc. Tuy nhiên, nếu muốn tối ưu về kết quả train với dataset lớn, khuyến khích dùng GBT để xác suất đúng được cải thiện hơn.

→ Sau khi hoàn thành xong bài toán cơ bản: dự đoán khách hàng, nhóm tiếp tục triển khai bài toán phân loại khách hàng theo nhóm.

#### d) Phân loại khách hàng theo nhóm, dataset lớn, giải thuật KMeans.

Script

```
scaler = StandardScaler(inputCol="raw_features", outputCol="features")
```

- Các bước làm sạch, chuyển đổi dữ liệu về dạng vector vẫn như bài toán Classification. Tuy nhiên, bài toán Cluster có thêm bước StandardScaler.
- Vì sao phải có thêm bước này: Thuật toán KMeans hoạt động dựa trên "khoảng cách" (distance) giữa các điểm dữ liệu. Nếu không scale, một cột có giá trị lớn (như TotalCharges từ 0-8000) sẽ "át" hoàn toàn ảnh hưởng của một cột có giá trị nhỏ (như SeniorCitizen từ 0-1). StandardScaler biến đổi tất cả các đặc trưng trong vector để chúng có cùng một thang đo (cụ thể là có trung bình  $\mu=0$  và độ lệch chuẩn  $\sigma=1$ ). Điều này đảm bảo mọi đặc trưng đều có "trọng lượng" đóng góp nhau vào việc tính toán khoảng cách.

```
df_transformed = pipeline.fit(df).transform(df)
```

k = 5

- Chọn chia dữ liệu thành 5 cụm.

```
predictions = model.transform(df_transformed)
```

```
predictions.select("gender", "Contract", "MonthlyCharges", "TotalCharges", "prediction").show(10, truncate=False)
```

- Thêm một cột mới tên là prediction vào DataFrame, chứa ID của cụm (từ 0 đến 4) mà mỗi khách hàng thuộc về.
- Hiển thị 10 khách hàng đầu tiên và họ thuộc về cụm nào.

```
predictions.groupBy("prediction").avg("MonthlyCharges", "TotalCharges").show()
```

- Đây là phần tính toán cốt lõi để tiến hành phân loại dữ liệu theo yêu cầu. Nó tính toán giá trị trung bình của MonthlyCharges và TotalCharges cho từng cụm phục vụ cho việc phân loại theo bản chất của mỗi cụm:

```
evaluator = ClusteringEvaluator()
```

```
silhouette = evaluator.evaluate(predictions)
```

- silhouette = evaluator.evaluate(predictions): Tính toán Silhouette Score. Đây là chỉ số đo lường chất lượng phân cụm. Giá trị càng gần 1.0: Các cụm càng tách biệt rõ ràng và các điểm trong cụm càng gần nhau.

→ Nhằm mở rộng bài toán, nhóm tiến hành triển khai model phân loại 1 tập dữ liệu hình ảnh lớn.

#### e) Bài toán mở rộng: Phân loại 1 tập dữ liệu hình ảnh lớn theo phương pháp học chuyển giao (Transfer Learning), mô hình học sâu ResNet50.

Khái quát về bài toán phân loại hình ảnh lớn: Quyết định xem một ảnh thuộc về nhóm nào trong một tập hợp các nhóm đã định sẵn mà dữ liệu đầu vào là các pixel, một dạng dữ liệu phi cấu trúc mà máy tính rất khó hiểu, sử dụng dữ liệu rất lớn để xử lý trên một máy tính duy nhất trong thời gian hợp lý.

Vì sao chọn phương pháp học chuyển giao Transfer Learning?

- Lý do chính là để tiết kiệm thời gian, tận dụng nguồn dữ liệu không lò, và đạt được độ chính xác cao mà không cần tự mình xây dựng mọi thứ từ đầu.
  - Huấn luyện một mô hình Deep Learning lớn như ResNet50 từ đầu (from scratch) đòi hỏi: Cần nhiều GPU mạnh, chạy liên tục trong nhiều tuần hoặc thậm chí hàng tháng. Chi phí có thể lên tới hàng chục nghìn USD. Bên cạnh đó, mất rất nhiều thời gian để mô hình học được các khái niệm cơ bản như "cạnh", "góc", "màu sắc". Với Transfer Learning, chỉ cần tải mô hình đã được huấn luyện trước mất vài phút và chỉ cần huấn luyện một mô hình LogisticRegression nhỏ cũng chỉ mất vài phút.
  - Các mô hình Deep Learning cần **cực kỳ nhiều dữ liệu** để học. Để dạy một mô hình nhận diện "mắt mèo" từ đầu, chúng ta cần hàng chục nghìn bức ảnh mèo ở mọi góc độ. Đối với bộ dữ liệu Chó/Mèo mà nhóm sử dụng thì chỉ có vài chục nghìn tấm. Con số này là **quá ít** để một mô hình học từ đầu, nó sẽ dẫn đến hiện tượng overfitting.
- Mô hình học sau ResNet50 là gì?
  - Đây được gọi là mô hình nền. Đóng vai trò là một **bộ trích xuất đặc trưng** (feature extractor). Nó *không* được huấn luyện lại. Nó chỉ "nhìn" bức ảnh thô và biến nó thành một vector 2048 chiều (tóm tắt thông minh về nội dung ảnh). Đây không phải là mô hình thực sự được huấn luyện. **Logistic Regression (Mô hình đầu)**: là mô hình *thực sự* được huấn luyện. Nó nhận đầu vào là các vector 2048 chiều đó và học một nhiệm vụ đơn giản: vạch ra ranh giới để phân biệt đâu là vector của "Chó", đâu là vector của "Mèo".
- Script:

```
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

- Dùng thư viện xử lý ảnh Pillow (PL) để bỏ qua những file ảnh bị lỗi nhỏ, nhanh chóng đi thẳng vào phần train.
- "Truncated images" là các file ảnh bị **cắt ngắn, không hoàn chỉnh**. Điều này thường xảy ra khi file bị tải lỗi hoặc bị hỏng, khiến nó thiếu mất phần dữ liệu cuối file.
- Theo mặc định, khi PIL cố gắng mở một file ảnh hỏng như vậy, nó sẽ báo lỗi (IOError) và **dừng toàn bộ chương trình**.
- Bằng cách đặt = True, nghĩa là đang ra lệnh cho PIL: "Nếu gặp ảnh lỗi, đừng dừng lại và crash. Cứ cố gắng tải và xử lý bất cứ phần nào của ảnh mà bạn đọc được."

```
spark.sparkContext.setLogLevel("ERROR")
```

- Khi train trên local với cửa sổ terminal nhỏ, các log và thông báo được in hàng loạt ra màn hình để báo cáo về hoạt động của nó, thêm dòng này để chỉ in ra những thông báo thật sự quan trọng, dễ dàng cho việc theo dõi kết quả khi train model.

```
df = df.withColumn("label", element_at(split(col("image.origin"), "/"), -2))
```

- Tách nhãn (label) từ đường dẫn. Ví dụ: path là .../PetImages/Cat/1.jpg: split(..., "/") tách chuỗi thành mảng: [..., "PetImages", "Cat", "1.jpg"]. element\_at(..., -2) lấy phần tử thứ 2 từ cuối lên, chính là "Cat" (hoặc "Dog").

```
resnet_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
```

- Tải mô hình ResNet50 đã được huấn luyện trước trên bộ dữ liệu ImageNet.

```
def _to_bytes(x):
    if x is None:
        return None
    if isinstance(x, (bytes, bytearray)):
        return bytes(x)
    try:
        return x.tobytes()
    except Exception:
        try:
            return bytes(x)
        except Exception:
            return None
```

- include\_top=False giúp gỡ bỏ lớp top layer của ResNet50. Đây là lớp dùng để phân loại 1000 vật thể của ImageNet. Chúng ta không cần lớp này vì chúng ta chỉ muốn phân loại Chó/Mèo.

```
def _open_image_from_any(content, _origin_path):
    if content is not None:
        b = _to_bytes(content)
        if b:
            try:
                return Image.open(io.BytesIO(b)).convert("RGB").resize((224, 224))
```

```

    except Exception:
        return None
    return None

```

- Thay thế lớp top layer đã gỡ bỏ bằng một lớp gộp trung bình (average pooling). Lớp này lấy đầu ra phức tạp của ResNet và nén nó lại thành một vector phẳng duy nhất. Với ResNet50, vector này có **2048 chiều (dimensions)**.

```
@pandas_udf(ArrayType(FloatType())))
```

- Vì Spark không biết cách chạy mô hình Keras, chúng ta phải dạy nó bằng cách sử dụng **Pandas UDF** (User Defined Function).
- @pandas\_udf**: Thay vì chạy hàm trên từng ảnh sẽ rất chậm, Spark sẽ nhóm các ảnh lại thành một *lô* (*batch*) (dưới dạng pandas.Series).

```

def extract_features_udf(content_series: pd.Series, origin_series: pd.Series) ->
    pd.Series:
    ...
    img = _open_image_from_any(content, origin)
    ...
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    feat = resnet_model.predict(x, verbose=0)

```

- extract\_features\_udf**: Hàm này nhận một lô ảnh và thực hiện:
  - `_open_image_from_any`: Mở dữ liệu nhị phân, chuyển đổi sang RGB, và resize về kích thước 224 x 224 (kích thước tiêu chuẩn đầu vào của ResNet50)
  - `np.expand_dims(x, axis=0)`: Thêm một chiều "batch" vì mô hình Keras luôn mong đợi nhận vào một lô ảnh, kể cả khi chỉ có 1 ảnh.
  - `preprocess_input(x)`: Tiền xử lý ảnh đầu vào theo cách mà ResNet50 đã được huấn luyện.
  - `resnet_model.predict(x)`: Trích xuất đặc trưng, chạy ảnh qua mô hình ResNet50.

```

except Exception as e:
    print("⚠️ Error khi xử lý ảnh:", e)

```

```

out.append([0.0] * 2048)

return pd.Series(out)

```

- **Error Handling:** Nếu một ảnh bị lỗi (ví dụ: file JPEG bị hỏng), hàm sẽ trả về một vector toàn số 0 để pipeline không bị dừng.

```
print("⚡ Bắt đầu huấn luyện mô hình Logistic Regression...")
```

```
indexer = StringIndexer(inputCol="label", outputCol="labelIndex")
```

```
indexer_model = indexer.fit(feature_vector_df)
```

```
data_indexed = indexer_model.transform(feature_vector_df)
```

```
train_df, test_df = data_indexed.randomSplit([0.8, 0.2], seed=42)
```

```
lr      = LogisticRegression(featuresCol="features_vec",      labelCol="labelIndex",
maxIter=50)
```

```
lr_model = lr.fit(train_df)
```

- Huấn luyện mô hình phân loại:
  - StringIndexer chuyển nhãn Cat, Dog thành số.
  - **LogisticRegression:** Huấn luyện một mô hình Hồi quy Logistic: Mô hình này học rất nhanh, vì nó chỉ cần tìm một đường ranh giới siêu phẳng trong không gian 2048 chiều để tách "Chó" ra khỏi "Mèo".
    - Đầu vào (FeaturesCol): Cột features\_vec (vector 2048 chiều từ ResNet50).
    - Đầu ra (LabelCol): Cột labelIndex (0.0 hoặc 1.0).

```
predictions = lr_model.transform(test_df)
```

- Dùng mô hình Logistic Regression đã huấn luyện để dự đoán trên tập test.

```

evaluator = MulticlassClassificationEvaluator(
    labelCol="labelIndex",
    predictionCol="prediction",
    metricName="accuracy"
)

```

```
accuracy = evaluator.evaluate(predictions)
```

- MulticlassClassificationEvaluator: Tính toán độ chính xác (accuracy) bằng cách so sánh cột prediction (dự đoán) và cột labelIndex (thực tế).

```
lr_model.write().overwrite().save(MODEL_DIR)
```

- Lưu kết quả và 3 file:
  - metrics\_df: File JSON chứa độ chính xác (ví dụ: {"accuracy": 0.985}).
  - label\_df: File JSON chứa ánh xạ nhãn (ví dụ: {"index": 0, "label": "Cat"}).
  - lr\_model.write(): Lưu lại toàn bộ mô hình Logistic Regression chứ không phải ResNet50 để có thể tái sử dụng sau này.

- Tiến hành chạy và thu về được kết quả:



```
Độ chính xác mô hình trên tập test: 1.0000
```

Hình 3-5. Kết quả huấn luyện local

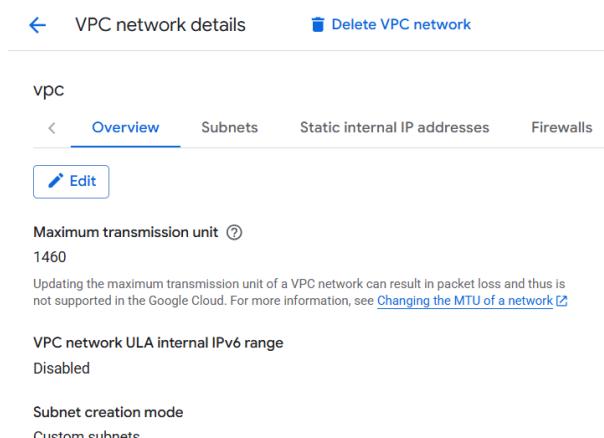
- Đánh giá kết quả thu được:

Do dataset vẫn còn nhỏ so với dataset tối ưu mà model có thể dùng để trên nên AUC = 1.0. Tuy nhiên, nhóm không tiến hành train với dataset lớn hơn vì môi trường thực hiện đang là máy ở với cấu hình CPU và RAM nhỏ nên dataset quá lớn thì quá trình train sẽ diễn ra rất lâu. Nhóm tiến hành chỉnh sửa script và thử nghiệm trên cụm master worker.

### 3.3.2. Xây dựng hạ tầng mạng

#### Tạo VPC

- Name: vpc
- Subnet creation mode: Custom



Hình 3-6. Tạo VPC

#### Tạo Subnets

- Subnet 1: spark-subnet: Subnet cho các

- Region: asia-southeast1-a
- IP range: 10.10.0.0/24
- No external IP
- Private Google Access: On → subnet có thể truy cập các tài nguyên và dịch vụ của Google mà không cần external IP
- Subnet 2: api-subnet
  - Region: asia-southeast1-a
  - IP range: 10.20.0.0/24
  - No external IP
  - Private Google Access: On
- Subnet 3: proxy-only-subnet: cho Load Balancer
  - Region: asia-southeast1-a
  - IP range: 10.20.0.0/24
  - purpose: REGIONAL\_MANAGED\_PROXY
  - Private Google Access: On

**New subnet**

Name \* spark-subnet Lowercase letters, numbers, hyphens allowed

Description

Region \* asia-southeast1 (Singapore)

IP stack type  IPv4 (single-stack)  IPv4 and IPv6 (dual-stack)  IPv6 (single-stack)

Primary IPv4 range

Associate with an internal range Use an internal range to specify the subnet's internal IP address range. The subnetwork can be associated with an entire internal range or only part of the range.

IPv4 range \* 10.10.0.0/24 E.g. 10.0.0.0/24

Hình 3-7. Tạo subnet

Name	Type	Targets	Source	Protocols and ports
vpc-allow-health-check	Ingress firewall rule	Tags: lb-health-check	IPv4 ranges: 35.191.0.0/16, 130.211.0.0/22, 209.85.152.0/22, 209.85.204.0/22	tcp:80, 8080
vpc-allow-web-ui	Ingress firewall rule	Tags: spark	IPv4 ranges: 0.0.0.0/0	tcp:8080, 8081, 9001
vpc-allow-http	Ingress firewall rule	Tags: http-server	IPv4 ranges: 0.0.0.0/0	tcp:80
vpc-allow-https	Ingress firewall rule	Tags: https-server	IPv4 ranges: 0.0.0.0/0	tcp:443

Name	Type	Targets	Source	Protocols and ports
vpc-allow-ssh	Ingress firewall rule	Apply to all	IPv4 ranges: 35.235.240.0/20	tcp:22
vpc-allow-spark-traffic	Ingress firewall rule	Tags: spark, bastion	IPv4 ranges: 10.10.0.0/24	All
vpc-allow-lb-proxy-to-api	Ingress firewall rule	Tags: lb-health-check	IPv4 ranges: 130.211.0.0/22, 35.191.0.0/16, 10.30.0.0/24	tcp:8080
vpc-allow-internal-api-traffic	Ingress firewall rule	Tags: api	IPv4 ranges: 10.20.0.0/24	All
vpc-allow-health-check	Ingress firewall rule	Tags: lb-health-check	IPv4 ranges: 35.191.0.0/16, 130.211.0.0/22, 209.85.152.0/22, 209.85.204.0/22	tcp:80, 8080

Bảng 3-2. Firewall rules

### Cloud NAT & Private Google Access

- Tạo router: vpc-router
  - Network: vpc
  - Region: asia-southeast1
- Tạo NAT:
  - Router: vpc-router
  - auto allocate nat external ips: On
  - nat all subnet ip ranges

Overview	Advertised and learned routes	Best routes
<b>Router overview</b>		
Network	vpc	
Region	asia-southeast1	
Interconnect encryption	Unencrypted	
Cloud Router ASN		
BGP identifier	Automatic	
Tags	-	

### 3.3.3. Service Accounts và quyền

- Tạo Service Account 1 (Spark)
  - Name: spark-sa
  - Display Name: "Spark Service Account"

- Role (Quyền): roles/storage.objectAdmin. Mục đích: Cho phép cụm Spark đọc (dữ liệu) và ghi (model, kết quả) lên Google Cloud Storage.
- Tạo Service Account 2 (API)
  - Name: api-sa
  - Display Name: "API Service Account"
- Role 1 (Quyền 1): roles/storage.objectViewer. Mục đích 1: Cho phép API server đọc các file model đã được huấn luyện từ Google Cloud Storage.
- Role 2 (Quyền 2): roles/compute.viewer. Mục đích 2: Cho phép API server xem (viewer) trạng thái của các tài nguyên Compute Engine (không bắt buộc nhưng hữu ích để theo dõi).
- 

### 3.3.4. Cụm Spark Master (3 nút High Availability)

Tạo VM mẫu, cài đặt các package và cấu hình dịch vụ cần thiết để đóng gói thành image:

Thành phần	Gói / Dịch vụ	Chức năng
<b>Hệ điều hành</b>	ubuntu-minimal 22.04 LTS	Linux nền tảng cho Spark cluster
<b>Java runtime</b>	default-jdk (OpenJDK 11)	Bắt buộc cho Spark, Hadoop, ZooKeeper
<b>Python runtime</b>	python3, pip3  Các thư viện tensorflow-cpu, pyshark, pillow, pyarrow, numpy, pandas, ...	Cho PySpark, TensorFlow/Keras xử lý ML  Hỗ trợ huấn luyện ML
<b>Apache Spark</b>	<b>Spark</b> spark-3.5.1-bin-hadoop3 (tải từ archive.apache.org)	Nền tảng xử lý phân tán
<b>Cài đặt Spark</b>	/opt/spark/	Thư mục chính Spark
<b>Biến môi trường</b>	SPARK_HOME=/opt/spark, PATH=\$PATH:\$SPARK_HOME/ bin:\$SPARK_HOME/sbin	Cấu hình toàn hệ thống
<b>Spark config</b>	/opt/spark/conf/spark-env.sh, /opt/spark/conf/workers	Khai báo master/worker
<b>Spark Master Service</b>	Chạy bởi start-master.sh trong startup script	Dịch vụ Spark Master daemon
<b>Spark REST API</b>	Port 6066, spark.master.rest.enabled=true	Cho phép gửi job từ API Flask

### Google Cloud Storage Connector

<b>Google Cloud Storage Connector</b>	ges-connector-hadoop3-2.2.29.jar	Cho phép Spark đọc / ghi dữ liệu từ Google Cloud Storage (gs://bucket/...) như một hệ thống file Hadoop (HDFS API)
---------------------------------------	----------------------------------	--

	<b>Zookeeper</b>	
<b>ZooKeeper</b>	zookeeper, zookeeperd (APT package)	Quản lý trạng thái Spark HA
<b>Cấu hình chính</b>	/opt/zookeeper/conf/zoo.cfg	Khai báo cluster 3 node
<b>Dữ liệu ZK</b>	/var/lib/zookeeper/	Nơi lưu metadata cluster
<b>File định danh node</b>	/var/lib/zookeeper/myid	Phân biệt từng node (1-3)
<b>Ports nội bộ</b>	2181 (client), 2888 (peer), 3888 (election)	Giao tiếp HA
<b>Service quản lý</b>	systemctl enable zookeeper	Khởi động cùng hệ thống

Bảng 3-3. Các gói/dịch vụ cho spark master

- Spark Standalone và Spark HA Integration: Khai báo cấu hình trong file /opt/spark/conf/spark-env.sh:

```
export SPARK_MASTER_HOST=10.10.0.2
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8080
export SPARK_DAEMON_JAVA_OPTS="\
-Dspark.deploy.recoveryMode=ZOOKEEPER \
-Dspark.deploy.zookeeper.url=10.10.0.2:2181,10.10.0.3:2181,10.10.0.4:2181 \
-Dspark.deploy.zookeeper.dir=/spark"
```

- Khai báo các biến của Spark để tham gia đúng cụm.
- Bật recovery mode = ZOOKEEPER
- Spark Master tự ghi trạng thái job, app, driver vào ZK.
- Khi 1 master chết → standby master tự động trở thành active.
- Từ image trên, tạo cụm Spark master (3 nút HA):
  - Thực hiện lặp qua 3 Vùng (Zone) để tạo 3 máy ảo Spark Master.
  - Tên các máy ảo sẽ được tạo:
    - spark-master-1 (trong zone asia-southeast1-a)

- spark-master-2 (trong zone asia-southeast1-b)
- spark-master-3 (trong zone asia-southeast1-c)
- Cấu hình chung cho mỗi máy ảo:
  - Machine type: e2-standard-4 Subnet: spark-subnet
  - Network: vpc; subnet: spark-subnet
  - External IP: None
  - Service Account: spark-sa
  - Scopes: Bật toàn quyền trên Storage
  - Tags: master, spark
  - Source Image: spark-master-image
  - Startup script:

```
#!/bin/bash

set -e


# 1. Xác định thông tin máy

INTERNAL_IP=$(hostname -I | awk '{print $1}')

ZK_DIR="/var/lib/zookeeper"
ZK_CONF="/etc/zookeeper/conf/zoo.cfg"


sudo mkdir -p $ZK_DIR
sudo chown -R zookeeper:zookeeper $ZK_DIR


# Map IP → myid

case "$INTERNAL_IP" in
  "10.10.0.2") MYID=1 ;;
  "10.10.0.3") MYID=2 ;;
  "10.10.0.4") MYID=3 ;;
  *) MYID=0 ;;
esac
```

```
echo "$MYID" | sudo tee $ZK_DIR/myid >/dev/null  
sudo chown zookeeper:zookeeper $ZK_DIR/myid  
sudo chmod 644 $ZK_DIR/myid
```

### # 2. Cấu hình ZooKeeper zoo.cfg

```
sudo bash -c "cat > $ZK_CONF" <<EOF  
tickTime=2000  
initLimit=10  
syncLimit=5  
dataDir=$ZK_DIR  
clientPort=2181  
maxClientCnxns=100  
  
server.1=10.10.0.2:2888:3888  
server.2=10.10.0.3:2888:3888  
server.3=10.10.0.4:2888:3888  
EOF
```

```
sudo chown zookeeper:zookeeper $ZK_CONF  
sudo chmod 644 $ZK_CONF
```

```
sudo systemctl restart zookeeper || true  
sudo systemctl enable zookeeper
```

### # 3. Cấu hình Spark Master HA

```
SPARK_ENV_FILE="/opt/spark/conf/spark-env.sh"  
  
if ! grep -q "spark.deploy.recoveryMode=ZOOKEEPER" "$SPARK_ENV_FILE"; then  
    cat <<EOF | sudo tee -a "$SPARK_ENV_FILE" >/dev/null
```

```

export SPARK_DAEMON_JAVA_OPTS="-
Dspark.deploy.recoveryMode=ZOOKEEPER \
-Dspark.deploy.zookeeper.url=10.10.0.2:2181,10.10.0.3:2181,10.10.0.4:2181 \
-Dspark.deploy.zookeeper.dir=/spark"

export SPARK_MASTER_HOST=$(hostname -f)
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8080
EOF
fi

# 4. Chuẩn bị thư mục Spark
sudo mkdir -p /var/lib/spark/work
sudo chown -R root:root /var/lib/spark

# 5. Chờ ZooKeeper sẵn sàng
ZK_NODES=("10.10.0.2" "10.10.0.3" "10.10.0.4")
for zk in "${ZK_NODES[@]}"; do
    until nc -z "$zk" 2181 >/dev/null 2>&1; do
        sleep 3
    done
done

# 6. Khởi động Spark Master
/opt/spark/sbin/stop-master.sh || true
sleep 2
/opt/spark/sbin/start-master.sh

```

Script này được nhóm dùng để tự động cấu hình cụm ZooKeeper 3 node và Spark Master chế độ High Availability (HA).

Đầu tiên, script tự nhận diện IP của máy rồi gán ID cho từng node (myid) để ZooKeeper biết node nào trong cụm. Sau đó nó tạo file cấu hình zoo.cfg, khai báo danh

sách 3 node (10.10.0.2–10.10.0.4) và khởi động dịch vụ ZooKeeper. Mỗi node sẽ chạy ZooKeeper để tham gia vào quá trình bầu chọn leader và đồng bộ trạng thái.

Tiếp theo, script chỉnh file môi trường của Spark để bật chế độ HA qua ZooKeeper (spark.deploy.recoveryMode=ZOOKEEPER). Nhờ đó, Spark Master có thể lưu trạng thái hoạt động lên ZooKeeper. Khi leader Master gặp lỗi, ZooKeeper sẽ tự chọn node khác làm leader, giúp cụm hoạt động liên tục.

Cuối cùng, script kiểm tra ZooKeeper trên các node đã sẵn sàng rồi khởi động Spark Master. Khi tất cả node master chạy, chỉ một node giữ vai trò leader, các node còn lại ở standby và tự động chuyển đổi khi có sự cố.

Tóm lại, script này giúp nhóm triển khai cụm Spark có khả năng tự phục hồi, giảm downtime, và đảm bảo độ tin cậy cao cho hệ thống phân tán.

<a href="#">spark-master-1</a>	asia-southeast1-a	10.10.0.2 ( <a href="#">nic0</a> )
<a href="#">spark-master-2</a>	asia-southeast1-b	10.10.0.3 ( <a href="#">nic0</a> )
<a href="#">spark-master-3</a>	asia-southeast1-c	10.10.0.4 ( <a href="#">nic0</a> )

### 3.3.5. Spark Worker Autoscaling Group

Tạo VM mẫu, cài đặt các packages tương tự VM master, không có Zookeeper và không cần cấu hình trước mà các bước cấu hình sẽ thực hiện ở startup script:

```
#!/bin/bash

### Spark Worker Autoscaling Startup Script

set -e

# ===== Metadata-based configs =====

SPARK_HOME="/opt/spark"

SPARK_USER=${SPARK_USER:-$(logname 2>/dev/null || whoami)}

SPARK_MASTER_URL=${SPARK_MASTER_URL:-"spark://10.10.0.2:7077,10.10.0.3:7077,10.10.0.4:7077"}

GCS_BUCKET=${GCS_BUCKET:-"gs://nt533q13-spark-data"}


SPARK_LOG_DIR="/var/log/spark"
SPARK_RUN_DIR="/var/run/spark"
SPARK_WORK_DIR="/var/lib/spark/work"

echo "===== $(date) - Spark Worker startup initiated on $(hostname) ====="
echo "[INFO] SPARK_USER=$SPARK_USER"
```

```

echo "[INFO] SPARK_MASTER_URL=$SPARK_MASTER_URL"
echo "[INFO] GCS_BUCKET=$GCS_BUCKET"

# ===== Ensure required directories =====

for dir in $SPARK_LOG_DIR $SPARK_RUN_DIR $SPARK_WORK_DIR; do
    sudo mkdir -p "$dir"
    sudo chown -R $SPARK_USER:$SPARK_USER "$dir"
    sudo chmod -R 755 "$dir"
done

# ===== spark-env.sh =====

cat <<EOF | sudo tee $SPARK_HOME/conf/spark-env.sh >/dev/null
export SPARK_WORKER_CORES=\$(nproc)
export SPARK_WORKER_MEMORY=\$(free -g | awk '/Mem:/ {printf "%dg", int(\$2*0.8)})')
export SPARK_WORKER_DIR=$SPARK_WORK_DIR
export SPARK_LOG_DIR=$SPARK_LOG_DIR
export SPARK_PID_DIR=$SPARK_RUN_DIR
EOF

# ===== spark-defaults.conf =====

cat <<EOF | sudo tee $SPARK_HOME/conf/spark-defaults.conf >/dev/null
# Enable event logs to GCS
spark.eventLog.enabled      true
spark.eventLog.dir          $GCS_BUCKET/spark-events

# Use GCE Application Default Credentials (no key)
spark.hadoop.google.cloud.auth.service.account.enable   false

```

```

# Optimize GCS connector I/O

spark.hadoop.fs.gs.implicit.dir.repair.enable true
spark.hadoop.fs.gs.outputstream.buffer.size 10485760
EOF


# ===== Restart worker cleanly =====

echo "[INFO] Restarting Spark Worker service..."

$SPARK_HOME/sbin/stop-worker.sh || true
sleep 2

nohup $SPARK_HOME/sbin/start-worker.sh $SPARK_MASTER_URL >
$SPARK_LOG_DIR/start-worker.log 2>&1 &

sleep 5

if pgrep -f "org.apache.spark.deploy.worker.Worker" >/dev/null; then
    echo "[SUCCESS] Spark Worker joined cluster successfully."
else
    echo "[ERROR] Spark Worker failed to start. Check $SPARK_LOG_DIR/start-
worker.log"
fi

echo "===== Startup completed at $(date) ====="

```

Script này giúp tự động cấu hình và khởi động Spark Worker khi máy ảo mới được tạo trong nhóm autoscaling.

Đầu tiên, script xác định các biến môi trường như đường dẫn Spark (/opt/spark), người dùng chạy dịch vụ, địa chỉ cụm Master (spark://10.10.0.2, 10.10.0.3, 10.10.0.4) và bucket GCS dùng để lưu log sự kiện. Sau đó, nó tạo các thư mục cần thiết cho log, PID và dữ liệu làm việc, đồng thời gán quyền cho user Spark.

Tiếp theo, script ghi các cấu hình động vào spark-env.sh, trong đó số CPU và dung lượng RAM Worker được lấy tự động theo tài nguyên thực tế của máy (80% RAM, toàn bộ CPU). File spark-defaults.conf được tạo để bật ghi log sự kiện lên GCS và tối ưu hiệu năng I/O của GCS connector.

Cuối cùng, script dừng Worker cũ (nếu có), khởi động lại tiến trình mới và kiểm tra xem Worker đã gia nhập cụm Spark Master thành công chưa. Nếu Worker hoạt động, hệ thống in thông báo thành công; ngược lại, log lỗi sẽ được lưu lại để kiểm tra.

Nhờ script này, các node Worker trong nhóm autoscaling có thể khởi động nhanh, tự cấu hình và tự động tham gia cụm Spark mà không cần thao tác thủ công, giúp hệ thống mở rộng linh hoạt khi tải tăng.

- Từ đó tạo Instance Template:
  - Name: spark-worker-template
  - Machine type: e2-standard-4
  - Subnet: spark-subnet
  - External IP: None
  - Service Account: spark-sa
  - Source Image: spark-worker-image

Name	Network	Subnetwork	NIC type	Primary internal IP address	Alias IP ranges	IP stack type
nic0	vpc	spark-subnet				IPv4

Name	Image	Interface type	Size (GB)	Device name	Type	Architecture	Encryption
Autogenerated	spark-worker-image-v26	—	30	spark-worker-template	Balanced persistent disk	—	Google-managed

Hình 3-5. Spark worker template

- Tạo Managed Instance Group:
  - Name: spark-worker-group
  - Instance template: spark-worker-template
  - Zone: asia-southeast1-a, b, c

- Min replicas: 1
- Max replicas: 4
- Target CPU utilization: 0.7 (tức 70%)

Autoscaling is turned off. The number of instances in the group won't change automatically. The autoscaling configuration is preserved.

Status	Name	Creation Time	Template	Zone	Per instance config	Internal IP	External IP
<input checked="" type="checkbox"/> Ready	spark-worker-group-0h5k	Oct 27, 2025, 1:53:28 AM UTC+07:00	spark-worker-template-v26 (Regional)	asia-southeast1-c		10.10.0.5 (nic0)	

Hình 3-6. Spark worker group

### 3.3.6. Cụm API + Load Balancer

Tạo VM mẫu, cài đặt các package môi trường java, python3, apache spark, các gói phụ thuộc của python3 để chạy các model (danh sách package đã nêu ở mục 3.3.4. Spark Master), chỉnh sửa /etc/systemd/system/flask.service ExecStart/WorkingDirectory=/opt/app/Flask\_REST\_API rồi đóng gói thành Image.

#### - Tạo Instance Template:

- Name: api-server-template
- Machine type: e2-standard-2
- Subnet: api-subnet
- External IP: None
- Service Account: api-sa
- Tags: api, http-server
- Source Image: api-server-image
- Startup script:

```
#!/bin/bash
set -e # Dừng nếu có lỗi bất thường
```

```

# Đảm bảo systemd đã khởi động xong
sleep 10

# Dọn code cũ
rm -rf /opt/app/

# Clone code mới từ GitHub
git clone https://github.com/hutuswatermelon/NT533Q13-Project /opt/app || true
chmod -R 755 /opt/app

# Restart Flask để load code mới
systemctl restart flask

```

- Tạo Managed Instance Group (MIG):
  - Name: api-server-group
  - Instance template: api-server-template
  - Size: 1 (số lượng instance ban đầu)
  - Zone: asia-southeast1-a, b, c
- Tạo Health Check:
  - Name: api-health-check
  - Protocol: HTTP
  - Port: 8080
  - Request path: /health
  - Check interval: 45s
  - Timeout: 45s
  - Healthy threshold: 2
  - Unhealthy threshold: 2
- Tạo Load Balancer:
  - Loại: HTTP Load Balancer (External Managed)
  - Chọn Internet facing (public)
  - Chọn Single Region Deployment
  - Nhấn Continue
- Tạo Frontend:
  - Protocol: HTTP

- Port: 8080
- Tạo Backend Service:
  - Name: api-backend-service
  - Instance group: api-server-group
  - Zone: asia-southeast1-a
  - Protocol: HTTP
  - Health Check: api-health-check
  - Port name: http
  - Balancing mode: UTILIZATION
  - Các mục khác giữ nguyên

api-load-balancer

Regional external Application Load Balancer

Region  
asia-southeast1

**Frontend**

Protocol ↑	IP:Port	Network Tier	Certificate	SSL Policy	HTTP keepalive timeout ⓘ
HTTP	35.213.143.79:8080	Standard	-		30 seconds

**Host and path rules**

Hosts ↑	Paths	Backend
All unmatched (default)	All unmatched (default)	api-backend-service

**Backend**

Backend services

1. api-backend-service

Endpoint protocol	HTTP
Named port	http
Timeout	60 seconds
IP address selection policy	Only IPv4
Health check	<a href="#">api-health-check</a>
Logging	Disabled
Backend security policy	<a href="#">default-security-policy-for-backend-service-api-backend-service</a>

[▼ Show advanced](#)

Backends

Name ↑	Type	IP stack type	Scope	Healthy	Autoscaling	Balancing mode	Selected ports ⓘ
<a href="#">api-server-group</a>	Instance group	IPv4	asia-southeast1	1 of 1	Off: Target CPU utilization 90%	Max backend utilization: 80%	8080

Hình 3-7. api Load Balancer

### 3.3.7. Phân tải tác vụ nặng ở API (MapPartition và Scaling API)

Trong hệ thống, cơ chế phân tải tác vụ nặng được triển khai thông qua mô hình tính toán song song phân tán của Apache Spark, cho phép xử lý đồng thời nhiều ảnh trong một lần thực thi. Khi người dùng tải tệp .zip lên API thông qua endpoint /batch\_classify\_dogcat, ứng dụng Flask sẽ tạm lưu trữ và tải dữ liệu lên Google Cloud Storage bằng hàm upload\_to\_gcs(), sau đó gửi yêu cầu khởi chạy job tới cụm Spark thông qua REST API của Spark Master:

```

resp = submit_spark_job(
    job_id=job_id,
    zip_uri=input_uri,
    model_path=DOGCAT_MODEL_PATH,
    label_path=DOGCAT_LABEL_PATH,
    output_dir=output_dir
)

```

Tại phía cụm Spark, công việc được chia nhỏ thành nhiều partition tương ứng với các tập con của dữ liệu đầu vào. Mỗi partition được xử lý độc lập bởi một executor trong cluster. Phần xử lý được hiện thực trong script Spark sử dụng hàm mapPartitions(), cho phép khởi tạo mô hình ResNet50 và Logistic Regression một lần duy nhất trên mỗi partition, thay vì khởi tạo lại ở từng bản ghi, qua đó tối ưu hiệu năng và giảm độ trễ khởi tạo mô hình:

```

def process_partition(iterator):
    model = ResNet50(weights="imagenet", include_top=False, pooling="avg")
    for path, content in iterator:
        img = Image.open(io.BytesIO(content)).convert("RGB").resize((224, 224))
        x = preprocess_input(np.expand_dims(np.asarray(img), 0))
        features = model.predict(x)
        yield (path, features.tolist())

```

Việc phân chia dữ liệu và điều phối thực thi được Spark đảm nhiệm hoàn toàn, dựa trên cấu hình tài nguyên được gửi kèm trong payload REST, ví dụ:

```

"spark.executor.instances": "4",
"spark.executor.cores": "2",
"spark.dynamicAllocation.enabled": "true",

```

Điều này giúp hệ thống tự động mở rộng quy mô (horizontal scaling) khi khối lượng dữ liệu tăng, bằng cách khởi tạo thêm executor để xử lý nhiều partition hơn song song. Kết quả suy luận được ghi trực tiếp về GCS theo đường dẫn gs://nt533q13-api-data/jobs/<job\_id>/results/preds.csv, cho phép truy xuất dễ dàng và không phụ thuộc vào node vật lý cụ thể.

Nhờ sự kết hợp giữa cơ chế phân vùng dữ liệu (data partitioning), song song hóa theo executor, và dynamic resource allocation, hệ thống có khả năng xử lý đồng thời

hàng nghìn ảnh với tốc độ cao, duy trì tính ổn định và tận dụng tối đa tài nguyên tính toán trong cụm Spark. Đây là minh chứng cho hiệu quả của kiến trúc phân tán – mở rộng động trong việc triển khai các ứng dụng học máy ở quy mô lớn.

### 3.3.8. Chuẩn bị script để chạy tác vụ huấn luyện từ local

Chuẩn bị script sẵn để có thể thực hiện việc huấn luyện mô hình từ máy tính cá nhân qua gcloud (đã xác thực tài khoản và project id) -> giúp triển khai lệnh spark-submit từ máy tính cá nhân thông qua IAP TCP Forwarding (IAP Tunneling).

```
gcloud compute start-iap-tunnel spark-master-1 8080 --local-host-port=localhost:8080  
--zone=asia-southeast1-a &
```

```
#!/bin/bash

PROJECT_ID="nt533q13-distributed-ml"
ZONE="asia-southeast1-a"
ZK_NODES="10.10.0.2:2181,10.10.0.3:2181,10.10.0.4:2181"

echo "Đang truy vấn master_status từ ZooKeeper..."
RAW_OUTPUT=$(gcloud compute ssh spark-master-1 --project=$PROJECT_ID --zone=$ZONE \
--command "sudo /opt/zookeeper/bin/zkCli.sh -server $ZK_NODES get /spark/master_status" 2>/dev/null)

MASTER_IP=$(echo "$RAW_OUTPUT" | grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}' | tail -n1)

if [ -z "$MASTER_IP" ]; then
    echo "Không xác định được master đang active!"
    echo "Output từ ZooKeeper:"
    echo "$RAW_OUTPUT"
    exit 1
fi
```

```

echo "Master hiện tại: $MASTER_IP"

# Map IP → tên VM

if [[ "$MASTER_IP" == "10.10.0.2" ]]; then
    MASTER_VM="spark-master-1"
elif [[ "$MASTER_IP" == "10.10.0.3" ]]; then
    MASTER_VM="spark-master-2"
elif [[ "$MASTER_IP" == "10.10.0.4" ]]; then
    MASTER_VM="spark-master-3"
else
    echo "IP $MASTER_IP không khớp VM nào đã biết!"
    exit 1
fi

gcloud compute ssh $MASTER_VM --project=$PROJECT_ID --zone=$ZONE \
--command "bash -lc '/opt/spark/bin/spark-submit \
--master spark://10.10.0.2:7077,10.10.0.3:7077,10.10.0.4:7077 \
--deploy-mode client \
--conf spark.deploy.recoveryMode=ZOOKEEPER \
--conf spark.deploy.zookeeper.url=$ZK_NODES \
--conf spark.deploy.zookeeper.dir=/spark \
--conf spark.jars=/opt/spark/jars/gcs-connector-hadoop3-2.2.29.jar \
--conf spark.hadoop.fs.gs.impl=com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem \
--conf spark.hadoop.fs.AbstractFileSystem.gs.impl=com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS \
--conf spark.hadoop.fs.gs.project.id=$PROJECT_ID \
--conf spark.executor.memory=12g \
--conf spark.driver.memory=6g \

```

```
--conf spark.executor.cores=4 \
gs://nt533q13-spark-data/scripts/train_ml_image_bigdata.py"
```

Script này giúp nhóm tự động xác định Spark Master đang hoạt động (leader) và gửi lệnh spark-submit tới đúng node leader trong cụm.

Cụ thể, script kết nối tới ZooKeeper thông qua lệnh zkCli.sh để lấy thông tin /spark/master\_status, sau đó trích xuất địa chỉ IP của Master hiện tại. Dựa vào IP này, script ánh xạ ra tên máy ảo tương ứng (spark-master-1, spark-master-2, hoặc spark-master-3). Nếu không tìm thấy leader, script in log lỗi từ ZooKeeper để kiểm tra.

Khi đã xác định được Master đang active, script dùng gcloud compute ssh để đăng nhập từ xa và chạy lệnh spark-submit. Job huấn luyện được cấu hình chạy ở chế độ client, kết nối đến cả ba master qua danh sách URL, và bật chế độ recovery với ZooKeeper để đảm bảo job không bị gián đoạn nếu master chuyển đổi.

Các tham số spark.executor.memory, spark.driver.memory, và spark.executor.cores được đặt sẵn để tối ưu hiệu năng huấn luyện. Ngoài ra, script cũng tích hợp GCS connector (phiên bản 2.2.29) để đọc/ghi dữ liệu trực tiếp trên Google Cloud Storage mà không cần tải thủ công.

Nhờ cơ chế này, nhóm có thể tự động gửi và quản lý job Spark an toàn trong môi trường HA, đảm bảo tiến trình huấn luyện không bị gián đoạn khi leader master thay đổi.

## CHƯƠNG 4. CÁC KỊCH BẢN THỬ NGHIỆM VÀ KẾT QUẢ

Truy cập vào web UI của Spark master để theo dõi trạng thái và hoạt động của cụm Spark:

The screenshot shows the Apache Spark 3.5.7 Master UI at spark://10.10.0.2:7077. It displays the following information:

- Cluster Statistics:**
  - URL: spark://10.10.0.2:7077
  - Alive Workers: 1
  - Cores in use: 4 Total, 0 Used
  - Memory in use: 12.0 GiB Total, 0.0 B Used
  - Resources in use:
  - Applications: 0 Running, 13 Completed
  - Drivers: 0 Running, 0 Completed
  - Status: ALIVE
- Workers (1)**

Worker Id	Address	State	Cores	Memory	Resources
worker-20251027100817-10.10.0.5-35315	10.10.0.5:35315	ALIVE	4 (0 Used)	12.0 GiB (0.0 B Used)	
- Running Applications (0)**

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
- Completed Applications (13)**

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20251028032049-0012	DogCatFeatureExtraction	4	12.0 GiB		2025/10/28 03:20:49	sa_104726917983945743577	FINISHED	11 min
app-20251027220011-0011	DogCatFeatureExtraction	16	12.0 GiB		2025/10/27 22:00:11	sa_104726917983945743577	FINISHED	1.3 min
app-20251027215238-0010	DogCatFeatureExtraction	8	12.0 GiB		2025/10/27 21:52:38	sa_104726917983945743577	FINISHED	1.7 min
app-20251027215112-0009	DogCatFeatureExtraction	4	12.0 GiB		2025/10/27 21:51:12	sa_104726917983945743577	FINISHED	1.8 min
app-20251027180922-0008	DogCatFeatureExtraction	4	12.0 GiB		2025/10/27 18:09:22	sa_104726917983945743577	FINISHED	3.6 min
app-20251027172332-0007	TelcoChurnPrediction	4	12.0 GiB		2025/10/27 17:23:32	sa_104726917983945743577	FINISHED	2.5 min
...	...	...	...	...	...	...	...	...

Hình 4-1. Spark UI

### 4.1. Kịch bản 1: Kiểm tra hoạt động của cụm Master HA:

Như đã trình bày ở trên, Zookeeper sẽ quản lý trạng thái ALIVE hay STANDBY của Master dựa vào Leader election (mode: leader/follower). Trong cùng 1 thời điểm, chỉ có 1 máy Master ALIVE, còn lại ở chế độ STANDBY:

- 10.10.0.3 ALIVE
- 10.10.0.2 STANDBY
- 10.10.0.4 STANDBY

### Spark Master at spark://10.10.0.3:7077

**URL:** spark://10.10.0.3:7077  
**REST URL:** spark://10.10.0.3:6066 (*cluster mode*)  
**Alive Workers:** 1  
**Cores in use:** 4 Total, 0 Used  
**Memory in use:** 12.0 GiB Total, 0.0 B Used  
**Resources in use:**  
**Applications:** 0 Running, 7 Completed  
**Drivers:** 0 Running, 0 Completed  
**Status:** ALIVE



## Spark Master at spark://10.10.0.2:7077

**URL:** spark://10.10.0.2:7077  
**REST URL:** spark://10.10.0.2:6066 (*cluster mode*)  
**Alive Workers:** 0  
**Cores in use:** 0 Total, 0 Used  
**Memory in use:** 0.0 B Total, 0.0 B Used  
**Resources in use:**  
**Applications:** 0 [Running](#), 0 [Completed](#)  
**Drivers:** 0 Running, 0 Completed  
**Status:** STANDBY



## Spark Master at spark://10.10.0.4:7077

**URL:** spark://10.10.0.4:7077  
**REST URL:** spark://10.10.0.4:6066 (*cluster mode*)  
**Alive Workers:** 0  
**Cores in use:** 0 Total, 0 Used  
**Memory in use:** 0.0 B Total, 0.0 B Used  
**Resources in use:**  
**Applications:** 0 [Running](#), 0 [Completed](#)  
**Drivers:** 0 Running, 0 Completed  
**Status:** STANDBY

Hình 4-2. Các status ban đầu trên Spark UI

Tiến hành dừng máy đang ALIVE (master 1), quan sát spark UI:



## Spark Master at spark://10.10.0.4:7077

**URL:** spark://10.10.0.4:7077  
**REST URL:** spark://10.10.0.4:6066 (*cluster mode*)  
**Alive Workers:** 1  
**Cores in use:** 4 Total, 0 Used  
**Memory in use:** 12.0 GiB Total, 0.0 B Used  
**Resources in use:**  
**Applications:** 0 [Running](#), 0 [Completed](#)  
**Drivers:** 0 Running, 0 Completed  
**Status:** ALIVE

### Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20251027100817-10.10.0.5-35315	10.10.0.5:35315	ALIVE	4 (0 Used)	12.0 GiB (0.0 B Used)	

### Running Applications (0)

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

### Completed Applications (0)

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

Hình 4-3. Máy master 3 up status lên ALIVE

Trạng thái của máy 3 đã chuyển sang ALIVE, chứng tỏ cụm High Availability hoạt động tốt khi có 1 máy đang ALIVE mà bị lỗi.

## 4.2. Kịch bản 2: Kiểm tra tính chịu lỗi (Fault Tolerance) của hệ thống

Kịch bản này mô phỏng điều kiện xấu để chứng minh rằng hệ thống có khả năng phục hồi sau sự cố, một tính chất cốt lõi của hệ thống phân tán.

- Mục tiêu: Chứng minh rằng cụm Spark có thể tự phục hồi và hoàn thành tác vụ ngay cả khi một trong các node worker bị lỗi đột ngột trong quá trình xử lý.
- Các bước thực hiện:
  - Bắt đầu thực thi tác vụ Spark dài trên cụm.
  - Theo dõi tiến trình trên Spark UI.
  - Trong khi tác vụ đang chạy, truy cập vào Google Cloud Console.
  - Chọn một trong các máy ảo đang đóng vai trò là Spark Worker và thực hiện hành động “PAUSE” để mô phỏng sự cố phần cứng hoặc mất kết nối.
  - Tiếp tục quan sát Spark UI.

## Kết quả quan sát

### 4.2.1. Chịu lỗi Worker Node

```

94765736/94765736 5s 0us/step
✖ Đã trích xuất đặc trưng ResNet50.
✖ Bắt đầu huấn luyện mô hình Logistic Regression...
25/10/28 10:29:21 ERROR TaskSchedulerImpl: Lost executor 3 on 10.10.0.10: worker lost: Not receiving heartbeat for 60 seconds
25/10/28 10:29:21 ERROR TaskSchedulerImpl: Lost executor 1 on 10.10.0.9: worker lost: Not receiving heartbeat for 60 seconds

⌚ Độ chính xác mô hình trên tập test: 0.6256
✓ Mô hình đã lưu tại: gs://nt533q13-spark-data/models/dogcat_lr_model

```

**Application: DogCatFeatureExtraction**

ID: app-20251028102554-0001  
 Name: DogCatFeatureExtraction  
 User: caphuutu2005@gmail.com  
 Cores: Unlimited (8 granted)  
 Executor Limit: Unlimited (2 granted)  
 Executor Memory - Default Resource Profile: 12.0 GiB  
 Executor Resources - Default Resource Profile:  
 Submit Date: 2025/10/28 10:25:54  
 State: RUNNING  
[Application Detail UI](#)

**Executor Summary (4)**

ExecutorID	Worker	Cores	Memory	Resource Profile Id	Resources	State	Logs
2	worker-20251028035257-10.10.0.11-40439	4	12288	0		RUNNING	stdout stderr
0	worker-20251027100817-10.10.0.5-35315	4	12288	0		RUNNING	stdout stderr

**Removed Executors (2)**

ExecutorID	Worker	Cores	Memory	Resource Profile Id	Resources	State	Logs
3	worker-20251028035301-10.10.0.10-42457	4	12288	0		LOST	stdout stderr
1	worker-20251028035252-10.10.0.9-46303	4	12288	0		LOST	stdout stderr

app-20251028102554-0001 DogCatFeatureExtraction 8 12.0 GiB 2025/10/28 10:25:54 caphuutu2005@gmail.com FINISHED 12 min

Hình 4-4. Spark phát hiện worker mất kết nối

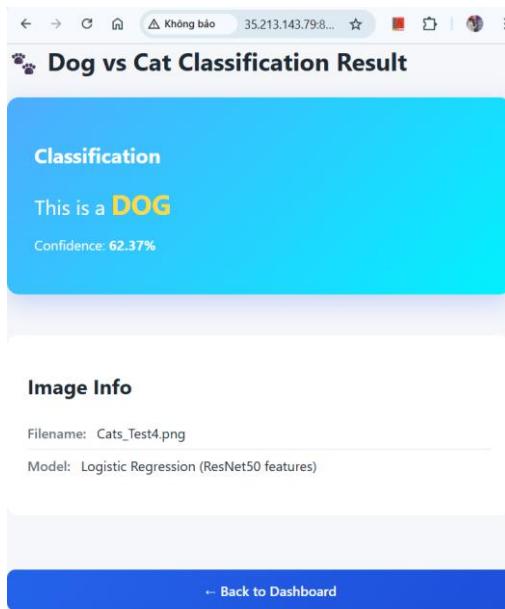
- Các sự kiện trên Spark UI: Các task đang chạy trên worker bị lỗi được đánh dấu là FAILED.
- Spark Master sẽ phát hiện worker đã mất kết nối.
- Các task bị lỗi sẽ được lên lịch lại và thực thi trên các worker còn lại đang hoạt động.
- Kết quả cuối cùng của toàn bộ tác vụ: Tác vụ vẫn phải hoàn thành với trạng thái FINISHED (SUCCEEDED) nhưng tốn thời gian hơn.

Kịch bản này cho thấy rõ cơ chế chịu lỗi của Spark. Mặc dù một phần của cụm bị lỗi, hệ thống không sụp đổ hoàn toàn mà có khả năng tự phục hồi bằng cách phân phối lại công việc nhưng tốn nhiều thời gian hơn.

#### 4.2.2. Kiểm tra Tính nhất quán sau lỗi

Độ chính xác mô hình trên tập test: 0.6256  
Mô hình đã lưu tại: gs://nt533q13-spark-data/models/dogcat\_lr\_model

Hình 4-5. Kết quả độ chính xác của mô hình



Hình 4-6. Kết quả thử nghiệm từ API gọi mô hình

Từ quan sát kết quả, nhóm nhận thấy độ chính xác của mô hình không thay đổi so với độ chính xác của kịch 1, mô hình được gọi từ API hoạt động chính xác. Từ đó chứng minh được tính nhất quán của dữ liệu khi có lỗi xảy ra trong quá trình thực thi huấn luyện.

#### 4.3. Kịch bản 3: Đánh giá hiệu năng và khả năng mở rộng theo chiều ngang (Scalability) đối với cụm Spark worker

Kịch bản so sánh hiệu năng giữa cụm 1 node worker với cụm nhiều node workers (4 nodes).

- Mục tiêu: So sánh trực tiếp thời gian huấn luyện mô hình Machine Learning giữa việc chạy trên một node duy nhất và chạy trên một cụm nhiều node. Qua đó, chứng minh rằng việc thêm tài nguyên tính toán (thêm các node worker) giúp giảm đáng kể thời gian xử lý.

##### 4.3.1. Chạy trên một worker:

- Cấu hình để Spark chỉ sử dụng một worker duy nhất (bằng cách chỉ khởi động một worker).
- Thực thi toàn bộ pipeline huấn luyện mô hình.
- Ghi lại chính xác tổng thời gian thực thi từ lúc bắt đầu đến khi kết thúc qua Spark UI.
- Ghi lại độ chính xác của mô hình.

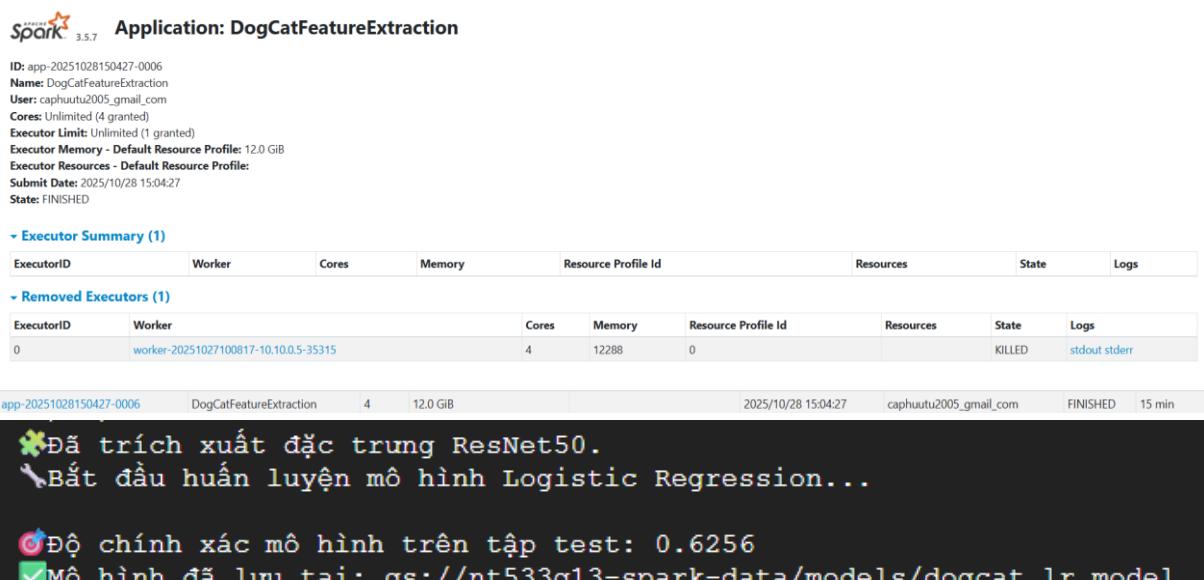
### 4.3.1. Chạy trên “cụm phân tán”:

- Cấu hình để Spark sử dụng tất cả các worker có sẵn (2+ workers).
- Thực thi lại chính xác pipeline huấn luyện mô hình với cùng bộ dữ liệu.
- Ghi lại tổng thời gian thực thi.
- Ghi lại độ chính xác của mô hình.

### 4.3.2. Kết quả

Chạy trên 1 worker:

- Thời gian thực thi + ghi file: 15 phút
- Độ chính xác mô hình: 0.6256



```

Apache Spark 3.5.7
Application: DogCatFeatureExtraction
ID: app-20251028150427-0006
Name: DogCatFeatureExtraction
User: caphuutu2005@gmail.com
Cores: Unlimited (4 granted)
Executor Limit: Unlimited (1 granted)
Executor Memory - Default Resource Profile: 12.0 GiB
Executor Resources - Default Resource Profile:
Submit Date: 2025/10/28 15:04:27
State: FINISHED

- Executor Summary (1)
ExecutorID Worker Cores Memory Resource Profile Id Resources State Logs
- Removed Executors (1)
ExecutorID Worker Cores Memory Resource Profile Id Resources State Logs
0 worker-20251027100817-10.10.0.5-35315 4 12288 0 KILLED stdout stderr

app-20251028150427-0006 DogCatFeatureExtraction 4 12.0 GiB 2025/10/28 15:04:27 caphuutu2005@gmail.com FINISHED 15 min

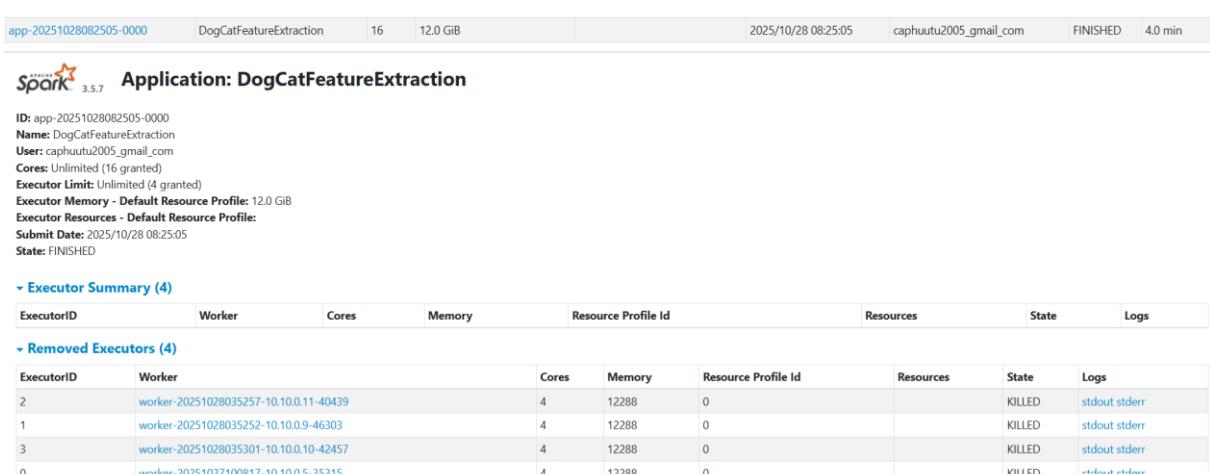
✖ Đã trích xuất đặc trưng ResNet50.
✖ Bắt đầu huấn luyện mô hình Logistic Regression...
⌚ Độ chính xác mô hình trên tập test: 0.6256
✅ Mô hình đã lưu tại: gs://nt533q13-spark-data/models/doqcat_lr.model

```

Hình 4-7. Kết quả chạy trên 1 worker

Chạy trên cụm phân tán:

- Thời gian thực thi + ghi file: 4 phút
- Độ chính xác mô hình: 0.6256



```

Apache Spark 3.5.7
Application: DogCatFeatureExtraction
ID: app-20251028082505-0000
Name: DogCatFeatureExtraction
User: caphuutu2005@gmail.com
Cores: Unlimited (16 granted)
Executor Limit: Unlimited (4 granted)
Executor Memory - Default Resource Profile: 12.0 GiB
Executor Resources - Default Resource Profile:
Submit Date: 2025/10/28 08:25:05
State: FINISHED

- Executor Summary (4)
ExecutorID Worker Cores Memory Resource Profile Id Resources State Logs
- Removed Executors (4)
ExecutorID Worker Cores Memory Resource Profile Id Resources State Logs
2 worker-20251028035257-10.10.0.11-40439 4 12288 0 KILLED stdout stderr
1 worker-20251028035252-10.10.0.9-46303 4 12288 0 KILLED stdout stderr
3 worker-20251028035301-10.10.0.10-42457 4 12288 0 KILLED stdout stderr
0 worker-20251027100817-10.10.0.5-35315 4 12288 0 KILLED stdout stderr

app-20251028082505-0000 DogCatFeatureExtraction 16 12.0 GiB 2025/10/28 08:25:05 caphuutu2005@gmail.com FINISHED 4.0 min

```

```

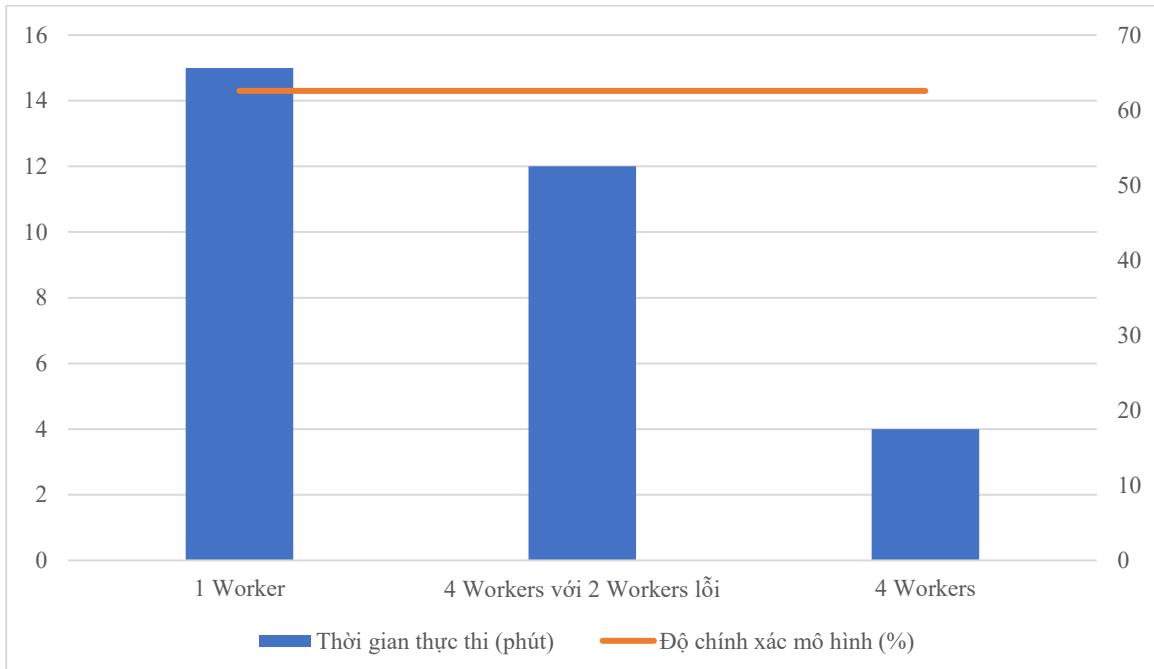
✖ Đã trích xuất đặc trưng ResNet50.
✖ Bắt đầu huấn luyện mô hình Logistic Regression...

✖ Độ chính xác mô hình trên tập test: 0.6256
✓ Mô hình đã lưu tại: gs://nt533q13-spark-data/models/doqcatt lr model

```

Hình 4-8. Kết quả chạy trên cụm phân tán

#### 4.4. Kết quả tổng hợp từ Kịch bản 2 và Kịch bản 3



Biểu đồ 4-1. Biểu đồ so sánh hiệu năng của mô hình từ các kịch bản 2 và kịch bản 3

Thời gian huấn luyện giảm mạnh khi tăng số lượng worker từ 1 → 4 (15 phút → 4 phút), tương đương hiệu năng tăng 3,75 lần.

Khi 2 worker trong cụm 4 bị lỗi, thời gian tăng lên 12 phút nhưng độ chính xác (0.6256) vẫn giữ nguyên.

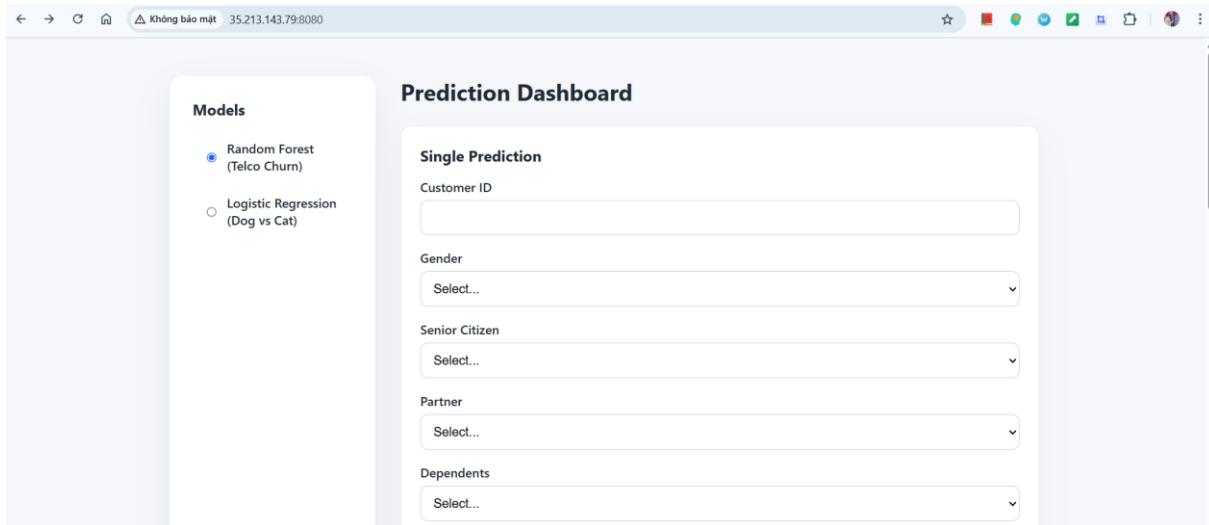
Kết luận: việc phân tán mô hình giúp tăng tốc độ huấn luyện đáng kể, và hệ thống vẫn đảm bảo tính chính xác ổn định ngay cả khi có lỗi nút (tính chịu lỗi của cụm hoạt động tốt).

#### 4.5. Kịch bản 4: Kiểm thử toàn bộ quy trình (End-to-End) và hiệu năng API Dự đoán

Kịch bản này kiểm tra tính thực tiễn của toàn bộ hệ thống, từ việc huấn luyện, triển khai cho đến việc phục vụ các yêu cầu dự đoán dưới tải cao.

- **Mục tiêu:** Đánh giá hiệu năng và khả năng đáp ứng của mô hình đã được triển khai dưới dạng một API khi có nhiều yêu cầu đồng thời.
- **Thiết lập hạ tầng:**
  - Một mô hình đã được huấn luyện từ các kịch bản trước và Google Cloud Storage.

- Một máy ảo riêng biệt trên Google Cloud để chạy API dự đoán (sử dụng Flask).
  - API: Ứng dụng Flask có một endpoint /predict nhận dữ liệu đầu vào và trả về kết quả dự đoán từ mô hình đã tải.
- **Các bước thực hiện:**
- Khởi động dịch vụ API Flask trên máy ảo đã chuẩn bị. Đảm bảo nó đã tải thành công mô hình.
  - Vào trang web UI của ứng dụng, thực hiện gửi các request đến Server.

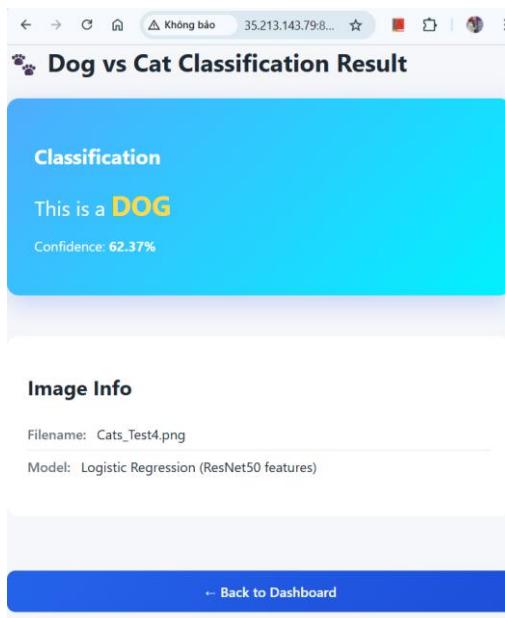


Hình 4-9. Web UI API FLask

Kết quả quan sát được, API cơ bản hoàn thành các tác vụ do yêu cầu đề ra, có trả file kết quả.

Name	Date modified	Type	Size
Churn.csv	24/10/2025 2:14 SA	Microsoft Excel Co...	933 KB
result.csv	24/10/2025 2:27 SA	Microsoft Excel Co...	98 KB
WA_Fn-UseC_-Telco-Customer-Churn.csv	24/10/2025 2:36 SA	Microsoft Excel Co...	998 KB

Hình 4-10. Kết quả khi up 1 file customer.csv



Hình 4-11. Kết quả Classify ảnh chó/mèo

#### 4.6. Kịch bản 5: Kiểm thử khả năng Autoscaling của cụm API Server, khả năng hoạt động của Load Balancer

- **Mục tiêu:** Đánh giá khả năng chịu tải và khả năng tự mở rộng (autoscaling) của hệ thống API dự đoán khi có lượng người dùng tăng cao đột biến.
- Các chỉ số theo dõi gồm:
  - Thời gian phản hồi trung bình (response time).
  - Số lượng request xử lý mỗi giây (throughput),
  - Tỷ lệ lỗi (failures),
  - Mức sử dụng CPU, Disk, Network và hành vi của Autoscaler trên GCP.
- **Thiết lập cấu hình:**

Thành phần	Mô tả
Công cụ	Locust (Python)
Endpoint kiểm thử	/predict/telco
Server	GCP Compute Engine, Flask API (Gunicorn 2 worker), có bật autoscaling
Giới hạn autoscaler	Tối thiểu 3 instance, tối đa 5 instance
Môi trường theo dõi	Cloud Monitoring (CPU, Disk, Network, Autoscaler)

Bảng 4-1. Cấu hình test Locust - stress test trên API server

Kịch bản	Lệnh thực thi	Số user	Tốc độ tạo user	Thời gian test	Mục tiêu

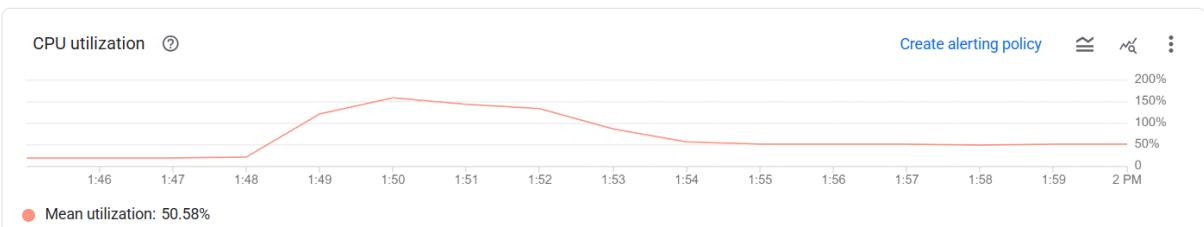
1	locust -u 200 -r 200 20/s 20 -t 1m --csv result	1 phút	Kiểm tra phản hồi cơ bản và tính ổn định của API
2	locust -u 300 -r 300 30/s 30 -t 1m --csv result2	1 phút	Đánh giá khả năng tự động mở rộng (autoscaling) khi tải tăng

Bảng 4-2. Lệnh kiểm thử Locust

Tiến hành quan sát Monitor của API Group:

### Phân tích monitor hiệu năng

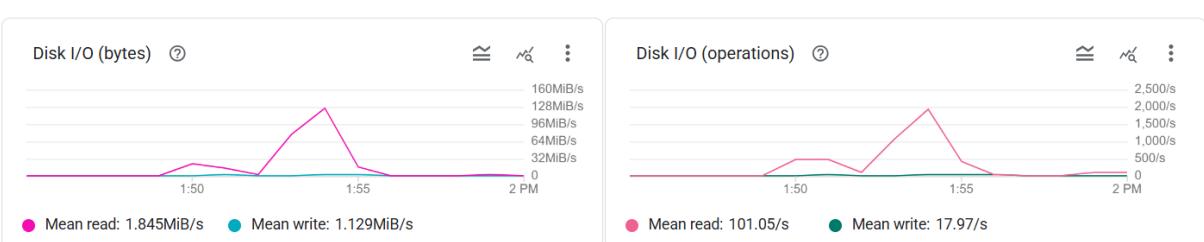
- CPU Utilization (~50%)



Biểu đồ 4-2. CPU Utilization

- Trung bình CPU toàn cụm đạt 50.58%, tuy có peak lên khoảng 70–80% ở giữa test.
- Điều này cho thấy API vẫn còn dư tài CPU, nhưng có thể bị nghẽn I/O hoặc thread lock tại Flask/Gunicorn hoặc network khi concurrent cao.
- Hệ thống chưa quá tải CPU, nhưng có sign của bottleneck ứng dụng (Flask / Spark).

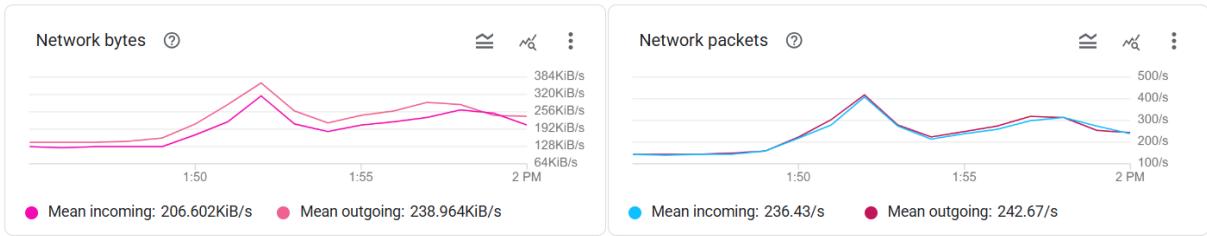
- Disk I/O



Biểu đồ 4-3. Disk I/O

- Mean read: 1.845 MiB/s, Mean write: 1.129 MiB/s, không đáng kể.
- Peak IOPS khoảng 2,000/s chứng tỏ có spike khi nhiều worker đọc/ghi log hoặc cache cùng lúc.
- Disk không phải bottleneck, tuy nhiên có thể gây trễ nếu log hoặc checkpoint Spark ghi ra đĩa cùng lúc.

- Network I/O



*Biểu đồ 4-4. Network I/O*

- Lưu lượng trung bình: ~230 KiB/s, gói tin ~240 pkt/s.
- Gần như cân bằng inbound/outbound, phù hợp với kiểu giao tiếp API REST.
- Network ổn định, không có dấu hiệu nghẽn.

#### - Autoscaler



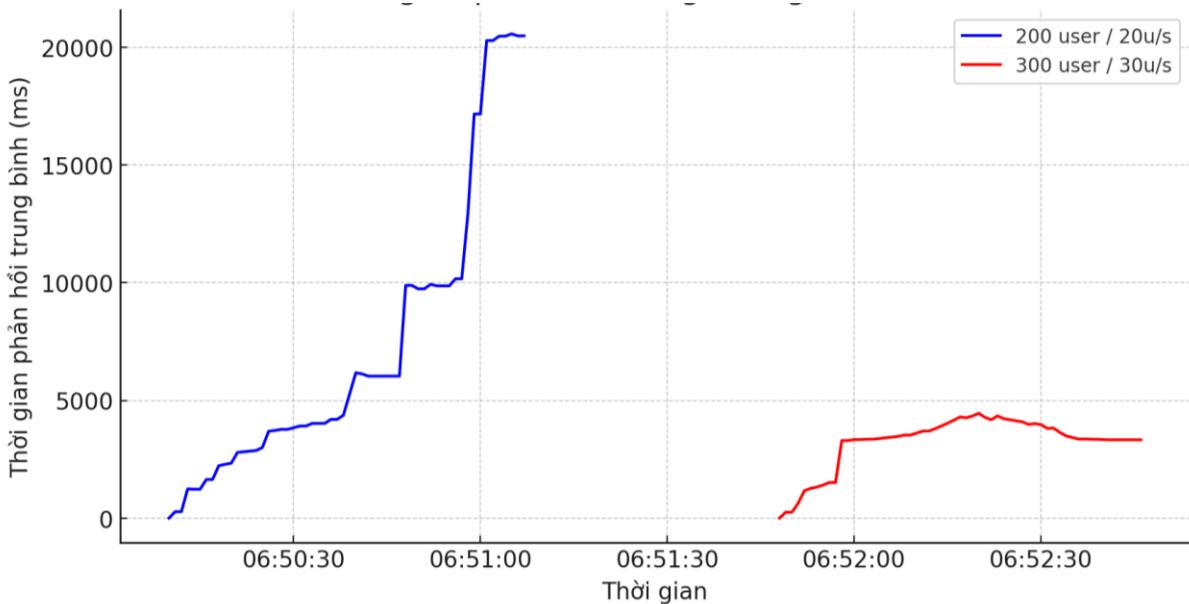
*Biểu đồ 4-5. Autoscaler*

- Tự động scale từ 3 lên 5 instance trong vòng ~2 phút.
- Tổng CPU utilization 250%/capacity 400%: tức là trung bình ~62.5%/instance.
- Autoscaler hoạt động đúng logic mong đợi.
- Tuy nhiên, việc scale hơi chậm hơn peak load, giai đoạn đầu test có thể bị drop hoặc delay UI.
- Dấu hiệu quá tải nhẹ ở lần test thứ 2

UI mất CSS là dấu hiệu server không phục vụ được file tĩnh (static) kịp thời, kết quả locust có lỗi ở giai đoạn cuối khi server quá tải.

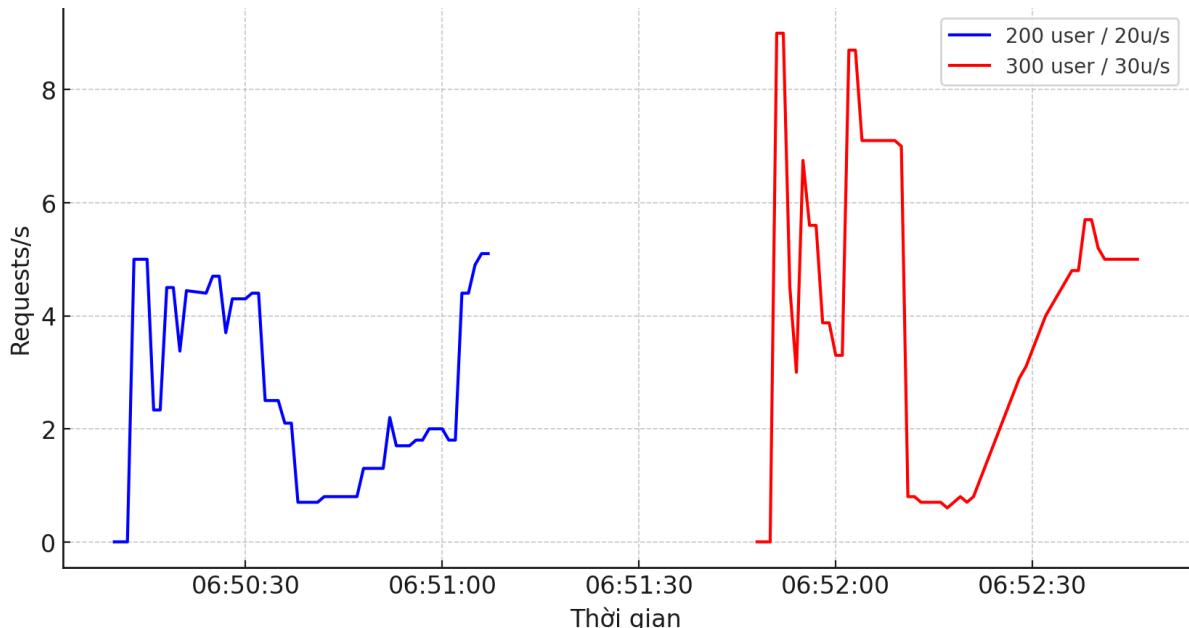
## Phân tích kết quả từ Locust

- Thời gian phản hồi trung bình (Avg Response Time)



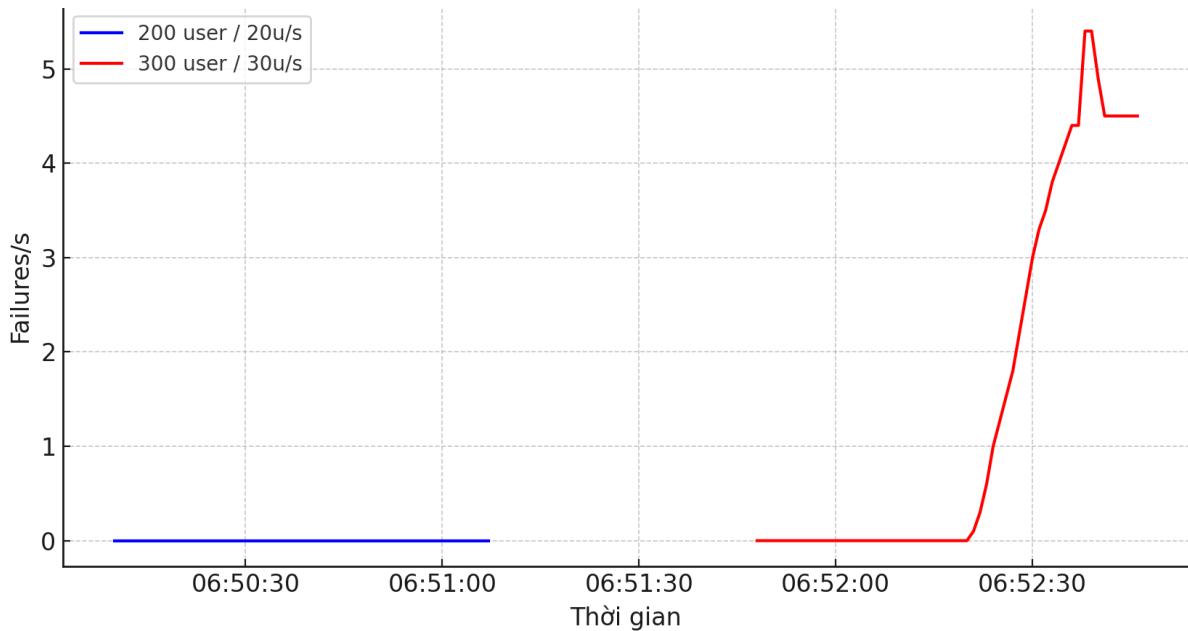
*Biểu đồ 4-6. So sánh thời gian phản hồi trung bình giữa hai lần kiểm thử*

- Ở mức tải 200 user / 20u/s, hệ thống giữ phản hồi ổn định, tuy có xu hướng tăng nhẹ về cuối test (~20.000 ms).
  - Với 300 user / 30u/s, thời gian phản hồi trung bình ban đầu thấp hơn nhưng biến động mạnh, phản ánh autoscaler hoặc hàng đợi xử lý đang điều chỉnh
- Throughput (Requests/s)



*Biểu đồ 4-7. So sánh throughput (req/s) giữa hai lần kiểm thử*

- Cả hai lần test đều dao động trong khoảng 4–9 req/s, nhưng ở lần thứ hai throughput cao hơn và biến động lớn hơn, do autoscaling mở thêm instance trong giai đoạn giữa test.
- Tỷ lệ lỗi (Failures/s)



Biểu đồ 4-8. So sánh tỷ lệ lỗi (Failures/s) giữa hai lần kiểm thử

- Lần 1 không ghi nhận lỗi.
- Lần 2 xuất hiện một số lỗi ( $\approx 4\text{--}5$  failures/s) ở cuối giai đoạn tải cao, trùng với thời điểm giao diện mất CSS — dấu hiệu ứng dụng bị nghẽn khi xử lý đồng thời nhiều request tĩnh và API.

### Kết luận

Trong quá trình kiểm thử tải bằng công cụ Locust, cụm API được triển khai trên GCP đã chứng minh khả năng mở rộng động và duy trì ổn định ở mức tải cao.

Khi tải 200 user/s, hệ thống phản hồi ổn định, thời gian đáp ứng thấp.

Khi tăng lên 300 user/s, autoscaler kích hoạt, mở rộng cụm từ 3 lên 5 instance, CPU đạt 250%/400%.

Lần test 300 user cho thấy cụm autoscale hoạt động, throughput tăng, nhưng hệ thống bắt đầu chạm giới hạn chịu tải ở tầng ứng dụng

### 4.7. Kịch bản 7: Kiểm thử hoạt động của IAM

- **Mục tiêu:** Kiểm thử khả năng bảo mật của IAM do Google Cloud quản lý.
- **Các bước thực hiện:** Chuẩn bị 1 tài khoản không thuộc Project chứa các VMs, tiến hành SSH thông qua IAP tunnel vào 1 máy bất kỳ trong Project.
- **Kết quả quan sát:**

```
C:\Users\CHT>gcloud config set account 23521696@gm.uit.edu.vn
Updated property [core/account].  

C:\Users\CHT>gcloud compute start-iap-tunnel api-server-group-4bms 8080 --local-host-port=localhost:8080 --zone=asia-southeast1-c &
ERROR: (gcloud.compute.start-iap-tunnel) Your current active account [23521696@gm.uit.edu.vn] does not have any valid credentials
Please run:  

$ gcloud auth login  

to obtain new credentials.  

For service account, please activate it first:  

$ gcloud auth activate-service-account ACCOUNT
```

Hình 4-12. Kết quả kiểm thử IAM

- Tài khoản không thuộc Project không thể SSH vào VM, chứng tỏ bảo mật hoạt động tốt.

## CHƯƠNG 5. KẾT LUẬN

### 5.1. Tổng hợp kết quả và đánh giá

Nhóm Khả Năng	Năng Cụ	Kịch Bản Kiểm Thử	Kết Quả Đánh Giá
	Khả Thể		
<b>Tính Sẵn sàng Cao (HA) &amp; Chịu lỗi (Fault Tolerance)</b>	Sẵn sàng Master Node (HA)	Dùng Master ALIVE, kiểm tra Master khác chuyển sang trạng thái ALIVE.	Hoạt động tốt.
<b>Hiệu năng &amp; Khả năng Mở rộng (Scalability)</b>	Chịu lỗi Worker Node Chịu lỗi và Phục hồi liệu	Dùng đột ngột Worker khi tác vụ đang chạy, quan sát task được phân bổ lại. Kiểm tra Tính nhất quán sau và Phục hồi (Xác minh kết quả cuối cùng hoàn toàn chính xác).	Tác vụ hoàn thành (SUCCEEDED), tồn thời gian hơn. Độ chính xác của mô hình vẫn toàn vẹn so với không lỗi. Mô hình được gọi từ API hoạt động tốt.
<b>Tính Năng (Functionality) &amp; Độ tin cậy (Reliability)</b>	Hiệu năng xử lý song	So sánh thời gian huấn luyện mô hình giữa cụm 1 Worker và cụm nhiều Workers.	Thời gian giảm đáng kể (15 phút → 4 phút).
<b>Bảo mật (Security) &amp; Truy cập</b>	Hiệu năng API dưới tải cao	Kiểm tra Autoscaling và Load Balancing của API Server (Sử dụng Locust gửi hàng trăm ConcurrentRequests).	Server phản hồi tốt ở mức 200 users, 20users/s. Đến 300 users và 30 users/s chạm đỉnh overload.
	Quy trình làm việc (End-to- End)	Kiểm thử toàn bộ quy trình từ UI → API Server → Mô hình dự đoán.	API hoàn thành tác vụ, trả về file kết quả. Hoàn thành $\frac{3}{4}$ chức năng API.
	Cơ chế Bảo mật Truy cập (IAP)	Kiểm tra Cơ chế Bảo mật và Truy cập (IAP) (Thử truy cập Spark UI mà không qua IAP hoặc bằng tài khoản không được ủy quyền).	Chỉ có thể truy cập bằng tài khoản Google Cloud có quyền hạn nhất định trong project.

### 5.2. Kết luận

Sau quá trình triển khai và thử nghiệm, nhóm đã xây dựng thành công hệ thống tính toán Machine Learning phân tán trên nền tảng Google Cloud Platform (GCP). Hệ thống được thiết kế dựa trên mô hình cụm Spark Standalone kết hợp Flask API phục vụ

dự đoán, đáp ứng đầy đủ các yêu cầu về hiệu năng, khả năng mở rộng, tính chịu lỗi và bảo mật.

Với kiến trúc nhiều tầng gồm:

- Cụm Spark Master – Worker được triển khai phân vùng trên ba zone trong khu vực asia-southeast1,
- Nhóm Flask API Server Autoscaling Group được phân tải thông qua Load Balancer,
- Cơ chế lưu trữ dữ liệu và mô hình thông qua Google Cloud Storage,
- Hệ thống bảo mật truy cập bằng Google Cloud Identity-Aware Proxy (IAP) và TLS tunnel,
- toàn bộ pipeline từ huấn luyện, lưu trữ, đến phục vụ dự đoán được vận hành tự động và có khả năng mở rộng linh hoạt.

Các kịch bản kiểm thử đã được tiến hành nhằm đánh giá hệ thống:

- Đánh giá hiệu năng và khả năng mở rộng (Scalability):
  - Khi tăng số lượng node worker, thời gian huấn luyện mô hình giảm đáng kể, chứng minh khả năng mở rộng theo chiều ngang của Spark.
  - Độ chính xác mô hình không bị ảnh hưởng, chứng tỏ việc phân tán không làm giảm chất lượng kết quả.
- Kiểm tra tính chịu lỗi (Fault Tolerance):
  - Khi một worker bị dừng đột ngột trong quá trình huấn luyện, Spark Master tự động phát hiện và tái phân bổ tác vụ (rescheduling) cho các worker còn lại.
  - Nhiệm vụ vẫn hoàn tất thành công, minh chứng cho khả năng tự phục hồi và độ tin cậy cao của hệ thống.
- Kiểm thử toàn bộ quy trình (End-to-End) và hiệu năng API dự đoán:
  - Hệ thống Flask API hoạt động ổn định, đáp ứng hàng trăm yêu cầu/giây với độ trễ thấp khi kiểm thử bằng Locust.
  - Khi kết hợp với Load Balancer và Autoscaling Group, hệ thống có khả năng mở rộng tự động theo tải.

Kết quả tổng hợp cho thấy hệ thống đạt được các tiêu chí cốt lõi của một nền tảng tính toán phân tán: Nhanh hơn – Ôn định hơn – Linh hoạt hơn – An toàn hơn.

Có thể áp dụng trực tiếp vào các bài toán huấn luyện và dự đoán mô hình Machine Learning quy mô lớn trên nền tảng đám mây.

### 5.3. Ưu điểm

Sau quá trình triển khai và thử nghiệm, hệ thống đã thể hiện nhiều ưu điểm nổi bật:

- Hiệu năng cao và khả năng mở rộng linh hoạt
  - Cụm Spark Standalone hoạt động ổn định, khi tăng số lượng node worker thì thời gian huấn luyện mô hình giảm đáng kể, chứng minh khả năng mở rộng theo chiều ngang (horizontal scaling).
  - Độ chính xác của mô hình hầu như không thay đổi, cho thấy việc phân tán xử lý không làm suy giảm chất lượng kết quả.
- Tính chịu lỗi và ổn định
  - Khi một worker gặp sự cố, Spark Master tự động phát hiện và tái phân bổ tác vụ cho các node còn lại.
  - Hệ thống duy trì tiến trình và hoàn tất job thành công, thể hiện khả năng tự phục hồi tốt (self-healing).
- Hệ thống API kết hợp với Load Balancer và Autoscaling Group, hệ thống có thể tự động mở rộng khi lưu lượng tăng.
- Kiến trúc phân tầng và bảo mật
  - Cấu trúc nhiều tầng gồm Spark Master–Worker, Flask API, và tầng lưu trữ GCS giúp tách biệt rõ chức năng và dễ mở rộng.
  - Cơ chế bảo mật bằng Google Cloud IAP và TLS tunnel giúp kiểm soát truy cập chặt chẽ và đảm bảo an toàn dữ liệu.
  - Toàn bộ pipeline từ huấn luyện – lưu trữ – dự đoán được tự động hóa trên nền tảng GCP.

#### 5.4. Nhược điểm

Mặc dù hệ thống hoạt động ổn định và đạt được mục tiêu đề ra, vẫn tồn tại một số hạn chế:

Việc xử lý yêu cầu API phân biệt file nhiều hình ảnh kích thước lớn chưa hoạt động ổn định.

Chi phí vận hành cao: Việc sử dụng nhiều VM, Load Balancer và lưu trữ đám mây khiến chi phí tăng nhanh khi mở rộng quy mô.

Quy trình mở rộng và cập nhật chưa hoàn toàn tự động: Managed Instance Group hỗ trợ auto-scaling, nhưng quá trình đóng gói và cập nhật batch image cho các node vẫn chưa chắc chắn, đôi khi service không tự khởi động đúng cách sau khi nhân bản.

Cấu hình hệ thống phức tạp: Việc đồng bộ giữa Spark, ZooKeeper, Flask và IAP yêu cầu cấu hình thủ công, dễ phát sinh lỗi khi thay đổi địa chỉ hoặc phiên bản.

Thiếu cơ chế giám sát và bảo mật nội bộ: Hệ thống chưa tích hợp giám sát tập trung (Prometheus, Grafana) và chưa mã hóa kết nối nội bộ giữa các node.

Phụ thuộc mạnh vào nền tảng Google Cloud: Cấu trúc hiện tại khó di chuyển sang môi trường khác mà không phải cấu hình lại toàn bộ mạng và dịch vụ.

## 5.5. Khó khăn trong quá trình thực hiện

Trong quá trình triển khai, nhóm gặp nhiều khó khăn cả về kỹ thuật lẫn hạ tầng. Một số thách thức tiêu biểu gồm:

- Huấn luyện các model để sử dụng:
  - Do chưa có kiến thức về học máy nên nhóm phải tốn khá nhiều thời gian tìm hiểu và triển khai những script cơ bản.
  - Việc lựa chọn hệ điều hành Ubuntu và dùng máy ảo trên VM Ware để huấn luyện và kiểm thử script đã train model giúp nhóm dễ dàng làm quen với các thao tác trên máy ảo khi triển khai lên môi trường Google Cloud nhưng cũng gặp không ít khó khăn vì cấu hình CPU và RAM nhỏ của máy ảo chỉ có thể ở mức trung bình để không ảnh hưởng đến máy Window đang sử dụng, từ đó dẫn đến không thể dùng bộ test với dung lượng quá lớn.
  - Trong quá trình huấn luyện model và chọn bộ dữ liệu để test, nhóm phải thử nhiều lần với nhiều dataset kích thước khác nhau và nhiều thuật toán, script khác nhau để tối ưu tương đối model khi triển khai lên Google Cloud.
  - Hiện tại, nhóm đang gặp phải lỗi không install được thư viện matplotlib để train model cluster trên máy local. Nhưng vì quá trình kiểm thử trên Google Cloud không sử dụng nên tạm thời nhóm chưa fix lỗi này.
- Cấu hình mạng và truy cập trong môi trường Google Cloud
  - Ban đầu việc thiết lập kết nối SSH, quyền truy cập qua IAP, và việc xác thực tài khoản trên CLI gây nhiều lỗi “Permission denied” hoặc “Timeout”.
  - Việc loại bỏ Bastion host và truy cập trực tiếp qua Google Cloud CLI cũng cần điều chỉnh lại quyền IAM và firewall rules để đảm bảo an toàn nhưng vẫn truy cập được từ local.
- Cấu hình cụm Spark và ZooKeeper cho chế độ HA (High Availability)
  - Việc đồng bộ ba node Spark Master qua ZooKeeper đòi hỏi cấu hình chính xác về hostname, địa chỉ IP nội bộ và phiên bản tương thích.
  - Nhiều lần cụm gặp lỗi Connection refused hoặc Active Master not detected, yêu cầu nhóm phải kiểm tra lại file spark-env.sh, spark-defaults.conf và trạng thái tiến trình ZooKeeper.
- Đóng gói VM thành Image và triển khai Managed Instance Group (MIG)
  - Một số service (như Flask hoặc Spark Worker) không tự khởi động sau khi tạo image do systemd chưa được enable đúng cách.

- Nhóm phải tạo file service riêng, kiểm tra log qua journalctl -u flask.service và chỉnh sửa cấu hình Restart=always để đảm bảo VM tự khởi động ổn định khi MIG mở rộng.
- Triển khai Load Balancer cho API Server
  - Việc tạo Health Check và cấu hình backend service ban đầu gây lỗi 502 do đường dẫn chưa đúng hoặc Flask chưa lắng nghe ở cổng mặc định.
  - Nhóm đã điều chỉnh cấu hình Flask service (--host=0.0.0.0 --port=8080) và cập nhật lại rule của load balancer để hệ thống hoạt động ổn định.
- Lỗi Spark Submit và Stream Timeout khi chạy trên cluster
  - Một số job bị ngắt kết nối do thời gian chờ mạng hoặc driver không nhận diện được master đang active.
  - Sau khi tinh chỉnh lại tham số spark.network.timeout và xác thực địa chỉ của master qua ZooKeeper, hệ thống hoạt động ổn định hơn.

Những khó khăn này giúp nhóm hiểu rõ hơn về cơ chế vận hành của hệ thống phân tán, đặc biệt là cách các thành phần Spark, ZooKeeper và Load Balancer tương tác trong môi trường đám mây.

## 5.6. Hướng phát triển

Tích hợp Prometheus + Grafana để giám sát hiệu năng và trực quan hóa chỉ số huấn luyện theo thời gian thực.

Triển khai cụm Spark và Flask API trên Google Kubernetes Engine (GKE) để tận dụng khả năng tự động co giãn container và quản lý tài nguyên tối ưu.

Bổ sung hỗ trợ GPU cho các tác vụ học sâu (Deep Learning) nhằm tăng tốc xử lý.

Xây dựng dashboard quản trị tập trung, cho phép theo dõi tiến trình, log và trạng thái từng node một cách trực quan.

## 5.7. Kết luận tổng quát

Đồ án đã chứng minh được rằng:

Một hệ thống tính toán phân tán được triển khai đúng kiến trúc có thể tăng tốc độ huấn luyện mô hình, đảm bảo tính sẵn sàng cao, chịu lỗi tốt, và mở rộng linh hoạt theo nhu cầu thực tế.

## CHƯƠNG 6. TÀI LIỆU THAM KHẢO

- [1] M. Zaharia, M. Chowdhury, T. Das, et al., “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,” *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI’12)*, pp. 15–28, 2012.
- [2] Apache Software Foundation, “Apache Spark™: Lightning-fast unified analytics engine,” *Apache Spark Documentation*, 2025. [Online]. Available: <https://spark.apache.org/>
- [3] Google Cloud, “Compute Engine Documentation,” *Google Cloud Platform*, 2025. [Online]. Available: <https://cloud.google.com/compute>
- [4] Google Cloud, “Deploying Distributed Machine Learning on GCP,” *Google Cloud Architecture Center*, 2025. [Online]. Available: <https://cloud.google.com/architecture>
- [5] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: Wait-free coordination for Internet-scale systems,” *USENIX Annual Technical Conference*, pp. 145–158, 2010.
- [6] TensorFlow Authors, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” *arXiv preprint arXiv:1603.04467*, 2016.
- [7] Google Cloud, “Monitoring distributed systems with Prometheus and Grafana on GCP,” *Google Cloud Blog*, 2024. [Online]. Available: <https://cloud.google.com/blog>
- [8] Y. Meng, J. Zhang, and Z. Chen, “A Scalable Machine Learning Platform Based on Spark and Kubernetes,” *IEEE Access*, vol. 9, pp. 14500–14512, 2021.
- [9] S. Ghemawat, H. Gobioff, and S. Leung, “The Google File System,” *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 29–43, 2003.
- [10] X. Meng et al., “MLlib: Machine Learning in Apache Spark,” *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

## CHƯƠNG 7. BẢNG PHÂN CÔNG ĐÁNH GIÁ THÀNH VIÊN

STT	Họ và Tên	MSSV	Công việc thực hiện	Mức độ hoàn thành
1	Cáp Hữu Tú	23521696	Triển khai hệ thống trên Cloud Viết script app API Kiểm thử kịch bản Viết báo cáo	90%
2	Huỳnh Ngọc Ngân Tuyền	23521753	Viết script train ML, Kiểm thử train ML trên hệ thống local Triển khai MIG + Load Balancer Kiểm thử kịch bản Viết báo cáo + làm slide thuyết trình	90%

Bảng 7-1. Bảng phân chia công việc

Công việc	Thời gian thực hiện
Nghiên cứu cơ sở lý thuyết, tìm hiểu công nghệ	29/09 - 05/10
Lên ý tưởng mô hình, triển khai mô hình sơ bộ	06 - 08/10
Triển khai huấn luyện thử bằng Apache Spark trên VM local, triển khai hạ tầng mô hình	08/10 - 12/10
Sửa chữa mô hình	12/10 - 29/10
Triển khai cụm API server	20/10 - 29/10
Thực hiện kịch bản thực nghiệm	26 - 29/10
Hoàn thiện báo cáo	26 - 29/10
Bổ sung báo cáo	29/10 - 02/11

Bảng 7-2. Bảng timeline công việc

Link Slide Thuyết trình	Link Mã nguồn
<a href="https://www.canva.com/design/DAG3EksCmZs/FxjC1FNiOgiU2-4saKzl3A/edit">https://www.canva.com/design/DAG3EksCmZs/FxjC1FNiOgiU2-4saKzl3A/edit</a>	<a href="https://github.com/hutuswatermelon/NT533Q13-Project">https://github.com/hutuswatermelon/NT533Q13-Project</a>

Bảng 7-3. Các URL

HẾT