**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**======\*\*\*======**

**Diagram

Description automatically generated with low confidence**

**BÁO CÁO THỰC NGHIỆM**

**HỌC PHẦN: MỘT SỐ CÔNG NGHỆ PHÁT TRIỂN PHẦN MỀM**

**XÂY DỰNG HỆ THỐNG IOT**

**VỚI CÔNG NGHỆ IOT CỦA MICROSOFT**

**NHÓM 19**

|  |  |
| --- | --- |
|  | *Hà Nội: 2024* |
|  |

|  |  |
| --- | --- |
| **CBHD:** | ***TS. Hà Mạnh Đào*** |
| **Sinh viên:** | Lê Công Nghĩa – 2021604236  (Nhóm trưởng) |
|  | Nguyễn Như Công – 2021603571  Tiến Minh Quang – 2021604955 |

# **LỜI CẢM ƠN**

Lời đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành tới TS.Hà Mạnh Đào. Trong suốt quá trình học tập và thực hiện đề tài một số công nghệ phát triển phần mềm, chúng em đã nhận được sự hỗ trợ, hướng dẫn tận tình từ thầy. Sự chỉ bảo chi tiết và những chia sẻ thực tế của thầy không chỉ giúp chúng em nâng cao kiến thức mà còn phát triển kỹ năng làm việc nhóm và giải quyết vấn đề. Chính phương pháp dạy học và định hướng của thầy đã giúp chúng em phát huy khả năng và khám phá những tiềm năng của bản thân.

Các buổi thực hành không chỉ là cơ hội để chúng em củng cố kiến thức mà còn là hành trang quý báu giúp chúng em tự tin khi bước vào môi trường làm việc thực tế. Để hoàn thành đề tài này, chúng em đã cùng nhau thảo luận, nghiên cứu, áp dụng những kiến thức từ môn học, các tài liệu tham khảo và cả những kinh nghiệm cá nhân.

Chúng em rất mong nhận được ý kiến đóng góp từ thầy cô và bạn đọc để có thể hoàn thiện đề tài hơn nữa. Một lần nữa, chúng em xin bày tỏ lòng biết ơn sâu sắc tới thầy vì những hướng dẫn quý báu trong suốt quá trình thực hiện đề tài.

*Một lần nữa, chúng em xin chân thành cảm ơn!*

# ***Nhóm sinh viên thực hiện.***

# **MỤC LỤC**

[DANH MỤC HÌNH ẢNH 1](#_Toc183174902)

[MỞ ĐẦU 2](#_Toc183174904)

[1. Lý do chọn chủ đề/đề tài 2](#_Toc183174905)

[2. Mục tiêu nghiên cứu 2](#_Toc183174906)

[3. Đối tượng và phạm vi nghiên cứu 3](#_Toc183174907)

[4. Ý nghĩa 3](#_Toc183174908)

[5. Cấu trúc của báo cáo 3](#_Toc183174909)

[CHƯƠNG 1.](#_Toc183174910) **[ỨNG DỤNG CỦA DDD TRONG PHÁT TRIỂN PHẦN MỀM VÀ KIẾN TRÚC MICROSERVICES](#_Toc183174910)** [5](#_Toc183174910)

[1.1. Tổng quan về Domain-Driven Design (DDD) 5](#_Toc183174911)

*[1.1.1. Định nghĩa](#_Toc183174913)* [5](#_Toc183174913)

*[1.1.2. Lịch sử phát triển](#_Toc183174913)* [6](#_Toc183174913)

[1.2. Các khái niệm cốt lõi trong DDD 6](#_Toc183174912)

*[1.2.1. Domain và Subdomain](#_Toc183174913)* [6](#_Toc183174913)

*[1.2.2. Business Logic và Business Rules](#_Toc183174914)* [7](#_Toc183174914)

*[1.2.3. Entity](#_Toc183174914)* [8](#_Toc183174914)

*[1.2.4. Value Object](#_Toc183174914)* [8](#_Toc183174914)

*[1.2.5. Aggregate](#_Toc183174914)* [10](#_Toc183174914)

*[1.2.6. Repository](#_Toc183174914)* [11](#_Toc183174914)

*[1.2.7. Service](#_Toc183174914)* [12](#_Toc183174914)

*[1.2.8. Bounded Context](#_Toc183174914)* [12](#_Toc183174914)

*[1.2.9. Ubiquitous Language](#_Toc183174914)* [12](#_Toc183174914)

[1.3. Các kiến trúc cốt lõi trong DDD 1](#_Toc183174912)3

*[1.3.1. Lớp ứng dụng (Application Layer)](#_Toc183174914)* [13](#_Toc183174914)

*[1.3.2. Lớp Domain (Domain Layer)](#_Toc183174914)* [13](#_Toc183174914)

*[1.3.3. Lớp hạ tầng (Infrastructure Layer)](#_Toc183174914)* [14](#_Toc183174914)

*[1.3.4. Lớp giao diện người dùng (User Interface Layer)](#_Toc183174914)* [1](#_Toc183174914)4

[1.4. Lợi ích và thách thức khi áp dụng DDD 1](#_Toc183174912)5

*[1.4.1. Lợi ích](#_Toc183174914)* [15](#_Toc183174914)

*[1.4.2. Thách thức](#_Toc183174914)* [15](#_Toc183174914)

[1.5. Triển khai dự án thực tế 1](#_Toc183174912)6

*[1.5.1. Mô tả cấu trúc dự án](#_Toc183174914)* [16](#_Toc183174914)

*[1.5.2. Thiết kế DDD của chương trình](#_Toc183174914)* [18](#_Toc183174914)

*[1.5.3. Cài đặt dự án](#_Toc183174914)* [20](#_Toc183174914)

*[1.5.4. Triển khai với Docker Swarm](#_Toc183174914)* [26](#_Toc183174914)

[CHƯƠNG 2.](#_Toc183174910) **[PHÁT HIỆN XE THEO THỜI GIAN THỰC BẰNG PYTHON VÀ OPENCV](#_Toc183174910)** [3](#_Toc183174910)1

[2.1. Giới thiệu và mô tả bài toán 3](#_Toc183174912)1

[2.2. Phân tích cấu trúc chương trình 3](#_Toc183174912)2

[2.3. Các hàm API được sử dụng 3](#_Toc183174912)6

[2.4. Cài đặt và triển khai 39](#_Toc183174912)

[2](#_Toc183174913)*[.4.1. Cài đặt môi trường](#_Toc183174913)* [39](#_Toc183174913)

[2](#_Toc183174913)*[.4.2. Triển khai dự án](#_Toc183174913)* [4](#_Toc183174913)3

[2.5. Kết quả và đánh giá 4](#_Toc183174912)7

[CHƯƠNG 3. XÂY DỰNG HỆ THỐNG IOT VỚI CÔNG NGHỆ IOT CỦA MICROSOFT 4](#_Toc183174910)9

[3.1. Tổng quan về IoT 49](#_Toc183174911)

[3](#_Toc183174913)*[.1.1. Khái niệm IoT](#_Toc183174913)* [49](#_Toc183174913)

[3](#_Toc183174913)*[.1.2. Cơ sở kỹ thuật của IoT](#_Toc183174913)* [5](#_Toc183174913)1

[3](#_Toc183174913)*[.1.3. Các ứng dụng của IoT trong cuộc sống](#_Toc183174913)* [5](#_Toc183174913)2

[3.2. Tổng quan về dịch vụ IoT trong Microsoft Azure 5](#_Toc183174912)4

[3](#_Toc183174913)*[.2.1. Azure IoT Hub](#_Toc183174913)* [5](#_Toc183174913)5

[3](#_Toc183174914)*[.2.2. Azure IoT Central](#_Toc183174914)* [5](#_Toc183174914)8

[3.3. Thiết kế hệ thống 6](#_Toc183174912)0

[3](#_Toc183174913)*[.3.1.](#_Toc183174913)* [Mô hình kiến trúc tổng thể 6](#_Toc183174913)0

[3](#_Toc183174913)*[.3.2.](#_Toc183174913)* [Liên kết giữa các thành phần 6](#_Toc183174913)1

[3](#_Toc183174913)*[.3.3.](#_Toc183174913)* [Lựa chọn và cấu hình thiết bị 6](#_Toc183174913)2

[3](#_Toc183174913)*[.3.4.](#_Toc183174913)* [Xây dựng các module xử lý dữ liệu trên đám mây 6](#_Toc183174913)3

[3](#_Toc183174913)*[.3.5.](#_Toc183174913)* [Lưu đồ hoạt động (Flow Diagram) 6](#_Toc183174913)3

[3.4. Cài đặt và cấu hình hệ thống 6](#_Toc183174912)4

[3.4.1 Cài đặt 6](#_Toc183174937)4

[3.4.2 Cấu hình Azure 6](#_Toc183174938)8

[3.5. Kết quả và ứng dụng thực tiễn 7](#_Toc183174912)3

[3.5.1 Kết quả 7](#_Toc183174937)3

[3.5.2 Ứng dụng thực tiễn 7](#_Toc183174938)3

[KẾT LUẬN VÀ KIẾN NGHỊ 7](#_Toc183174946)4

[TÀI LIỆU THAM KHẢO 7](#_Toc183174946)6

# 

# **DANH MỤC HÌNH ẢNH**

[Hình 1 Domain-Driven Design](#_Toc183189402) 5

[Hình 2 Cuốn sách Domain-Driven Design 6](#_Toc183189402)

[Hình 3 Domain và Subdomain 7](#_Toc183189402)

[Hình 4 Internet of Things 4](#_Toc183189402)9

[Hình 5. Ứng dụng cảu IoT trong tủ lạnh 5](#_Toc183189403)0

[Hình 6. Một số loại cảm biến hay gặp 5](#_Toc183189404)2

[Hình 7. Tổng quan về ứng dụng của IoT 5](#_Toc183189405)2

[Hình 8. Theo dõi lộ trình đi của xe chở hàng 53](#_Toc183189406)

[Hình 9. Theo dõi tình trạng sinh trưởng của cây trồng 5](#_Toc183189407)3

[Hình 10. Ví dụ về nhà thông minh 54](#_Toc183189408)

[Hình 11. Azure IoT Hub 55](#_Toc183189409)

[Hình 12. Kiến trúc Azure IoT Hub 56](#_Toc183189410)

[Hình 13. Azure IoT Central 5](#_Toc183189411)8

[Hình 14. Kiến trúc Azure IoT Central 5](#_Toc183189412)9

[Hình 15. Kết nối phần cứng ESP8266 và DHT11 6](#_Toc183189413)1

[Hình 16. Cảm biến DHT11 6](#_Toc183189414)2

[Hình 17. Mạch ESP8266 6](#_Toc183189415)2

[Hình 18. Lưu đồ hoạt động 6](#_Toc183189415)3

**MỞ ĐẦU**

## 1. Lý do chọn chủ đề/đề tài

Trong thời đại công nghiệp 4.0, Internet of Things (IoT) đã và đang trở thành một xu hướng công nghệ nổi bật, mang lại sự đổi mới vượt bậc trong nhiều lĩnh vực như sản xuất, y tế, giao thông, và đời sống hàng ngày. Microsoft là một trong những nhà cung cấp giải pháp IoT hàng đầu, cung cấp nền tảng mạnh mẽ và linh hoạt để xây dựng các hệ thống IoT thông minh. Tuy nhiên, việc ứng dụng và triển khai các giải pháp IoT của Microsoft tại Việt Nam vẫn còn hạn chế do rào cản về kiến thức, kỹ thuật và chi phí. Vì vậy, việc nghiên cứu và xây dựng một hệ thống IoT với công nghệ của Microsoft không chỉ giúp nâng cao năng lực triển khai IoT mà còn mở ra cơ hội áp dụng thực tiễn trong các doanh nghiệp Việt Nam.

## 2. Mục tiêu nghiên cứu

- Thiết kế và triển khai một hệ thống IoT tích hợp, bao gồm các thành phần như thiết bị cảm biến , module kết nối, và nền tảng lưu trữ, xử lý dữ liệu trên đám mây (Azure).

- Minh họa cách thức kết nối và quản lý dữ liệu giữa các thiết bị IoT và dịch vụ đám mây của Microsoft thông qua Azure IoT Hub và Azure IoT Central.

- Nghiên cứu và áp dụng các công cụ và dịch vụ của Microsoft IoT như Azure IoT Hub, Azure Digital Twins, và Power BI để tối ưu hóa quá trình thu thập, xử lý, và phân tích dữ liệu.

- Thử nghiệm hệ thống trong một số tình huống thực tế như giám sát môi trường (nhiệt độ, độ ẩm) sử dụng cảm biến DHT11 và module ESP8266.

- Cung cấp tài liệu hướng dẫn và giải pháp kỹ thuật nhằm hỗ trợ doanh nghiệp Việt Nam trong việc ứng dụng công nghệ IoT của Microsoft vào sản xuất và quản lý.

- Hỗ trợ cộng đồng nghiên cứu và phát triển IoT trong nước với các kết quả và bài học kinh nghiệm từ nghiên cứu này.

Cuối cùng là không chỉ tạo ra một hệ thống mẫu có tính khả thi mà còn thúc đẩy việc ứng dụng rộng rãi công nghệ IoT trong các lĩnh vực khác nhau tại Việt Nam, qua đó đóng góp vào quá trình chuyển đổi số quốc gia.

## 3. Đối tượng và phạm vi nghiên cứu

- **Đối tượng nghiên cứu**: Các công nghệ IoT của Microsoft, bao gồm nền tảng Azure IoT Hub, Azure IoT Central, và các công cụ phân tích dữ liệu như Azure Machine Learning và Power BI. Ứng dụng IoT trong giám sát nhiệt độ và độ ẩm của môi trường theo thời gian thực.

- **Phạm vi nghiên cứu**: Đề tài tập trung vào việc nghiên cứu lý thuyết về IoT và công nghệ của Microsoft. Xây dựng và thử nghiệm hệ thống IoT trên nền tảng Azure trong quy mô nhỏ, bao gồm cảm biến nhiệt độ DHT11 và module ESP8266. Đề xuất các mô hình triển khai phù hợp với điều kiện thực tế của các doanh nghiệp vừa và nhỏ tại Việt Nam.

## 4. Ý nghĩa

* **Ý nghĩa khoa học**: Góp phần làm phong phú thêm tài liệu nghiên cứu về IoT và các giải pháp công nghệ của Microsoft. Minh họa rõ ràng quy trình triển khai và tích hợp các dịch vụ IoT của Microsoft trong môi trường thực tế. Đề xuất các giải pháp kỹ thuật và mô hình vận hành có giá trị tham khảo cho cộng đồng nghiên cứu và phát triển IoT.
* **Ý nghĩa thực tiễn**: Cung cấp một giải pháp mẫu và các tài liệu hướng dẫn chi tiết, giúp doanh nghiệp Việt Nam dễ dàng tiếp cận và ứng dụng công nghệ IoT của Microsoft. Nâng cao hiệu quả quản lý và vận hành trong các lĩnh vực sản xuất, logistics, và nông nghiệp thông minh thông qua việc áp dụng IoT. Thúc đẩy chuyển đổi số tại Việt Nam, đặc biệt trong bối cảnh các doanh nghiệp đang tìm kiếm các giải pháp công nghệ hiện đại để tối ưu hóa chi phí và nâng cao năng lực cạnh tranh.

## 5. Cấu trúc của báo cáo

Ngoài phần Mở đầu và Kết luận, báo cáo sẽ gồm 4 chương chính.

**Chương 1. Ứng dụng của DDD trong phát triển phần mềm và kiến trúc Microservice**

Chương này giới thiệu về Domain-Driven Design (DDD). Kết hợp với kiến trúc microservices, DDD hỗ trợ tổ chức các dịch vụ độc lập, phản ánh chính xác quy trình nghiệp vụ và dễ dàng mở rộng, bảo trì. Chương cũng phân tích lợi ích, thách thức và cách áp dụng DDD hiệu quả trong các dự án thực tế.

**Chương 2. Phát hiện xe theo thời gian thực sử dụng Python và OpenCV**

Chương này giới thiệu và phân tích dự án phát hiện xe theo thời gian thực, một ứng dụng sử dụng ngôn ngữ lập trình Python và thư viện OpenCV. Với sự hỗ trợ của kỹ thuật thị giác máy tính hiện đại và bộ phân loại Haar Cascade, chương trình có khả năng nhận dạng và theo dõi các phương tiện giao thông trong video trực tiếp hoặc từ tệp video.

Trình bày chi tiết từ cách cấu trúc chương trình, các hàm API được sử dụng, cho đến quy trình cài đặt và triển khai. Đồng thời, chương cũng phân tích kết quả chương trình, cũng như đánh giá tiềm năng ứng dụng thực tiễn.

**Chương 3. Xây dựng hệ thống IoT với công nghệ IoT của Microsoft**

Chương này sẽ đề cập một cách tổng quan nhất về Internet of Things (IoT) các khái niệm, thuộc tính, cơ sở kỹ thuật và các ứng dụng của IoT trong cuộc sống hiện đại ngày nay cũng như các công nghệ IoT của Microsoft.

Trình bày thiết kế tổng thể của hệ thống IoT, cũng như cài đặt và triển khai hệ thống: Mô hình kiến trúc tổng thể, mô tả cách các thành phần liên kết với nhau, lựa chọn và cấu hình thiết bị cảm biến, gateway, xây dựng các module xử lý dữ liệu trên đám mây.

# **CHƯƠNG 1. ỨNG DỤNG CỦA DDD TRONG PHÁT TRIỂN PHẦN MỀM VÀ KIẾN TRÚC MICROSERVICES**

## Tổng quan về Domain-Driven Design (DDD)

### ***Định nghĩa***

### IMG_256

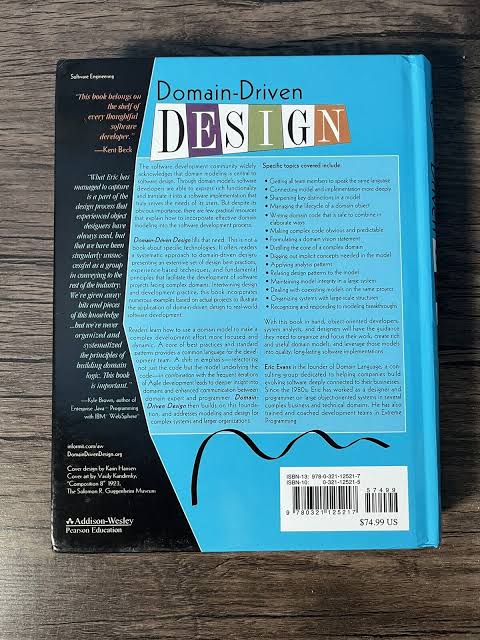
Hình 1 Domain-Driven Design

Domain Driven Design (DDD) là một phương pháp phát triển phần mềm nhằm xây dựng các hệ thống phức tạp thông qua việc hiểu rõ và mô hình hóa domain (lĩnh vực nghiệp vụ) mà phần mềm hướng tới. DDD không chỉ tập trung vào việc mã hóa phần mềm mà còn nhấn mạnh tầm quan trọng của việc thiết kế các đối tượng (entities) và hành vi (behaviors) phản ánh chính xác các quy trình nghiệp vụ trong thế giới thực. DDD hướng đến việc xây dựng các hệ thống bằng cách chia nhỏ các vấn đề lớn thành các subdomains, mỗi subdomain sẽ có các mô hình riêng biệt và chuyên sâu. Điều này giúp các đội ngũ phát triển dễ dàng quản lý các yêu cầu nghiệp vụ phức tạp và thay đổi một cách linh hoạt mà không làm gián đoạn toàn bộ hệ thống. Mỗi phần của hệ thống sẽ sử dụng một Ubiquitous Language (ngôn ngữ phổ quát) để giao tiếp giữa các thành viên trong nhóm phát triển, giúp mọi người có thể hiểu và mô tả vấn đề một cách rõ ràng và chính xác.

### ***1.1.2. Lịch sử phát triển***

Domain-Driven Design ra đời từ cuốn sách "Domain-Driven Design: Tackling Complexity in the Heart of Software" của Eric Evans xuất bản năm 2003.

Trước khi DDD xuất hiện, thiết kế phần mềm thường chú trọng vào khía cạnh kỹ thuật, nhưng chưa thực sự giải quyết tốt các vấn đề phức tạp về nghiệp vụ. DDD đã mở ra cách tiếp cận mới, giúp các nhóm phát triển phần mềm xây dựng hệ thống có khả năng phản ánh chính xác các khái niệm, quy trình thực tế trong lĩnh vực của hệ thống đó.



Hình 2 Cuốn sách Domain-Driven Design

## Các khái niệm cốt lõi trong DDD

### ***Domain và Subdomain***

Domain là toàn bộ phạm vi vấn đề mà phần mềm đang giải quyết. Đây có thể là các lĩnh vực như quản lý bán hàng, tài chính, y tế, hoặc giáo dục. Mỗi domain sẽ có các quy trình nghiệp vụ và yêu cầu riêng biệt mà phần mềm cần phải thực thi.

Subdomains là các phân nhánh của domain lớn, mỗi subdomain tập trung vào một phần nhỏ của nghiệp vụ mà có thể hoạt động độc lập với các phần còn lại của hệ thống. DDD khuyến khích chia hệ thống thành các subdomains để dễ dàng quản lý và phát triển.

Ví dụ: Trong một hệ thống quản lý ngân hàng, domain chính có thể là 'quản lý tài khoản ngân hàng', trong khi các subdomains sẽ là 'quản lý giao dịch', 'quản lý vay mượn', 'quản lý thẻ tín dụng'. Mỗi subdomain này sẽ có các mô hình và quy trình nghiệp vụ riêng biệt nhưng liên kết với nhau để tạo thành một hệ thống hoàn chỉnh.

### IMG_256

Hình 3 Domain và Subdomain

### ***Business Logic và Business Rules***

Business Logic là tập hợp các quy trình và hành vi giúp hệ thống thực hiện các nghiệp vụ theo yêu cầu của khách hàng. Nó có thể bao gồm các phép toán, quy trình kiểm tra dữ liệu, hay việc tính toán các giá trị trong hệ thống. Business logic chính là yếu tố quan trọng trong việc quyết định cách thức phần mềm xử lý và ra quyết định dựa trên các dữ liệu đầu vào từ người dùng hoặc hệ thống bên ngoài.

Business Rules là các quy định và luật lệ mô tả cách thức xử lý các tình huống trong nghiệp vụ, giúp hệ thống tuân thủ các chính sách và quy trình của tổ chức. Các quy tắc này sẽ có ảnh hưởng trực tiếp đến business logic, quyết định cách mà dữ liệu sẽ được xử lý trong hệ thống.

Ví dụ: Trong một hệ thống thương mại điện tử, business rule có thể là 'Một người chỉ có thể đặt tối đa 10 sản phẩm trong cùng một đơn hàng.' Business logic sẽ là 'Khi một người thêm sản phẩm vào giỏ hàng, hệ thống sẽ kiểm tra xem số lượng có vượt quá giới hạn này hay không.

### ***Entity (Thực thể)***

Entity là các đối tượng có danh tính duy nhất trong hệ thống, được nhận diện qua một thuộc tính nhất định. Entity có thể thay đổi trạng thái theo thời gian nhưng vẫn giữ được danh tính ban đầu.

Ví dụ: Trong một hệ thống quản lý sinh viên, mỗi sinh viên là một Entity, với mã sinh viên là duy nhất.

class SinhVien {  
 private String maSinhVien; // Danh tính duy nhất  
 private String ten;  
 private int tuoi;  
 public SinhVien(String maSinhVien, String ten, int tuoi) {  
 this.maSinhVien = maSinhVien;  
 this.ten = ten;  
 this.tuoi = tuoi;  
 }  
 // Getter và Setter  
 public String getMaSinhVien() { return maSinhVien; }  
 public String getTen() { return ten; }  
 public int getTuoi() { return tuoi; }  
 public void setTuoi(int tuoi) { this.tuoi = tuoi; }  
}

### ***Value Object (Đối tượng giá trị)***

Value Object là các đối tượng không có danh tính riêng mà chỉ được nhận diện qua giá trị của các thuộc tính của chúng. Chúng thường bất biến và có thể thay thế bằng các đối tượng khác có cùng giá trị.

Ví dụ: Một địa chỉ giao hàng có thể là một Value Object. Nếu địa chỉ thay đổi, chúng ta tạo một đối tượng mới thay vì thay đổi giá trị bên trong.

class DiaChi {  
 private String soNha;  
 private String duong;  
 private String thanhPho;  
 private String maBuuDien;  
 public DiaChi(String soNha, String duong, String thanhPho, String maBuuDien) {  
 this.soNha = soNha;  
 this.duong = duong;  
 this.thanhPho = thanhPho;  
 this.maBuuDien = maBuuDien;  
 }  
 // Getter  
 public String getDiaChiDayDu() {  
 return soNha + " " + duong + ", " + thanhPho + ", " + maBuuDien;  
 }  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) return true;  
 if (obj == null || getClass() != obj.getClass()) return false;  
 DiaChi diaChi = (DiaChi) obj;  
 return soNha.equals(diaChi.soNha) && duong.equals(diaChi.duong) &&   
 thanhPho.equals(diaChi.thanhPho) && maBuuDien.equals(diaChi.maBuuDien);  
 }  
 @Override  
 public int hashCode() {  
 return Objects.hash(soNha, duong, thanhPho, maBuuDien);  
 }  
}

### ***Aggregate (Tập hợp)***

Aggregate là một nhóm các Entity và Value Object có mối liên kết mật thiết, hoạt động như một đơn vị nhất quán. Aggregate có một Entity đóng vai trò "root" để quản lý các thành phần bên trong và đảm bảo tính toàn vẹn của dữ liệu trong tập hợp.

Ví dụ: Trong hệ thống quản lý đơn hàng, một đơn hàng có thể là một Aggregate, trong đó "Đơn hàng" là Aggregate Root, chứa các sản phẩm (Entity).

class SanPham {  
 private String maSanPham;  
 private String tenSanPham;  
 private double gia;  
 public SanPham(String maSanPham, String tenSanPham, double gia) {  
 this.maSanPham = maSanPham;  
 this.tenSanPham = tenSanPham;  
 this.gia = gia;  
 }  
 public String getMaSanPham() { return maSanPham; }  
 public double getGia() { return gia; }  
}  
class DonHang {  
 private String maDonHang;  
 private List<SanPham> sanPhams;  
  
 public DonHang(String maDonHang) {  
 this.maDonHang = maDonHang;  
 this.sanPhams = new ArrayList<>();  
 }  
 public void themSanPham(SanPham sanPham) {  
 this.sanPhams.add(sanPham);  
 }  
 public double tinhTongGiaTri() {  
 return sanPhams.stream().mapToDouble(SanPham::getGia).sum();  
 }  
}

### ***Repository (Kho lưu trữ)***

Repository là lớp trung gian giữa tầng dữ liệu và mô hình nghiệp vụ, chịu trách nhiệm truy xuất và lưu trữ các Aggregate. Repository giúp đơn giản hóa việc thao tác dữ liệu bằng cách cung cấp một giao diện dễ sử dụng.

Ví dụ: Repository cho `DonHang`.  
interface DonHangRepository {  
 void luuDonHang(DonHang donHang);  
 DonHang timDonHangTheoMa(String maDonHang);  
}  
class DonHangRepositoryImpl implements DonHangRepository {  
 private Map<String, DonHang> database = new HashMap<>();  
 @Override  
 public void luuDonHang(DonHang donHang) {  
 database.put(donHang.getMaDonHang(), donHang);  
 }  
 @Override  
 public DonHang timDonHangTheoMa(String maDonHang) {  
 return database.get(maDonHang);  
 }  
}

### ***Service (Dịch vụ)***

Service là lớp để thực hiện các hành động nghiệp vụ phức tạp mà không thuộc về Entity hay Value Object nào. Service không lưu trữ trạng thái và được dùng để triển khai các quy trình nghiệp vụ.

Ví dụ: Một `ThanhToanService` để xử lý thanh toán.

class ThanhToanService {  
 public boolean xuLyThanhToan(DonHang donHang, double soTienThanhToan) {  
 double tongGiaTri = donHang.tinhTongGiaTri();  
 return soTienThanhToan >= tongGiaTri;  
 }  
}

### ***Bounded Context (Ngữ cảnh giới hạn)***

Bounded Context xác định phạm vi mà các khái niệm và mô hình có cùng ý nghĩa. Điều này rất quan trọng trong các hệ thống lớn, giúp tránh xung đột khi các phần khác nhau của hệ thống có thể có những khái niệm trùng tên nhưng ý nghĩa khác nhau.

Ví dụ: Trong hệ thống bán hàng, "Sản phẩm" trong ngữ cảnh "Kho hàng" có thể chỉ mang ý nghĩa vật lý (số lượng, vị trí), còn "Sản phẩm" trong ngữ cảnh "Kế toán" mang ý nghĩa tài chính (giá trị, doanh thu).

### ***Ubiquitous Language (Ngôn ngữ chung)***

Ubiquitous Language là ngôn ngữ nhất quán để mô tả các khái niệm trong hệ thống, được sử dụng chung giữa đội phát triển và chuyên gia nghiệp vụ. Ngôn ngữ này xuất hiện không chỉ trong tài liệu mà còn trong mã nguồn để đảm bảo sự đồng nhất.

Ví dụ: Nếu trong hệ thống quản lý bán hàng, "Đơn hàng" và "Sản phẩm" là những khái niệm quan trọng, thì chúng ta sẽ sử dụng chúng đúng nghĩa ở cả mã nguồn và trong các cuộc thảo luận với các bên liên quan.

## Các thành phần kiến trúc trong DDD

### ***Lớp ứng dụng (Application Layer)***

- Chức năng chính: Đây là lớp chứa logic điều phối và kiểm soát quá trình xử lý của hệ thống, không chứa các quy tắc nghiệp vụ phức tạp. Lớp ứng dụng sử dụng các dịch vụ từ lớp Domain để thực hiện các công việc cần thiết, điều phối thao tác của hệ thống và cung cấp dữ liệu đến UI để hiển thị.  
- Đặc điểm:  
 + Không chứa các quy tắc nghiệp vụ phức tạp; thay vào đó, nó sử dụng các dịch vụ từ lớp Domain để thực hiện các công việc cần thiết.  
 + Có thể bao gồm các quy trình ngắn hạn như tạo lệnh (command), tạo sự kiện hoặc thực thi các thao tác đơn giản trong hệ thống.  
- Ví dụ: Trong một ứng dụng quản lý đơn hàng, lớp ứng dụng sẽ điều phối luồng công việc khi tạo đơn hàng mới, như lấy thông tin sản phẩm từ kho, kiểm tra tính hợp lệ, và cuối cùng lưu đơn hàng thông qua lớp domain mà không thực hiện các thao tác nghiệp vụ cụ thể.

### ***Lớp Domain (Domain Layer)***

- Chức năng chính: Đây là lớp cốt lõi của hệ thống, nơi chứa toàn bộ logic nghiệp vụ và mô hình của domain. Lớp này phản ánh quy trình, quy tắc nghiệp vụ thực tế của lĩnh vực.  
- Đặc điểm:  
 + Chứa các Entity, Value Object, Aggregates, và các Service nghiệp vụ.  
 + Các quy tắc nghiệp vụ được thực thi tại đây, đảm bảo tính chính xác và toàn vẹn của dữ liệu cũng như quy trình.  
 + Độc lập với các lớp khác, không phụ thuộc vào lớp hạ tầng hay lớp giao diện người dùng.  
- Các thành phần chính: Entity, Value Object, Aggregate, Domain Service.

### ***Lớp hạ tầng (Infrastructure Layer)***

- Chức năng chính: Cung cấp các dịch vụ hỗ trợ cho lớp Domain và lớp Application, kết nối hệ thống với các công nghệ bên ngoài như cơ sở dữ liệu, dịch vụ email, và hệ thống lưu trữ.  
- Đặc điểm:  
 + Chứa các thành phần để truy cập dữ liệu và quản lý kết nối, như các Repository, Unit of Work, và các dịch vụ bên ngoài khác.  
 + Hỗ trợ việc thực thi các thao tác nghiệp vụ trong domain qua việc lưu trữ dữ liệu, quản lý cơ sở dữ liệu và các dịch vụ khác.  
 + Sử dụng các mô hình, như Repository Pattern, để tách biệt logic nghiệp vụ và việc lưu trữ dữ liệu, giúp lớp Domain không phụ thuộc vào chi tiết của lớp hạ tầng.  
- Ví dụ: Trong một hệ thống quản lý nhân viên, lớp hạ tầng chứa các repository để truy vấn và cập nhật dữ liệu trong cơ sở dữ liệu, dịch vụ gửi email thông báo, và các thành phần truyền thông tin từ lớp domain ra bên ngoài.

### ***Lớp giao diện người dùng (User Interface Layer)***

- Chức năng chính: Hiển thị dữ liệu và tương tác với người dùng cuối. Đây là lớp giao tiếp trực tiếp với người dùng để thu thập đầu vào và cung cấp đầu ra.  
- Đặc điểm:  
 + Bao gồm các thành phần giao diện như màn hình, trang web, hoặc API cho các ứng dụng di động.  
 + Sử dụng dữ liệu từ lớp Application để hiển thị thông tin và nhận thao tác người dùng để gửi lại cho Application Layer.  
 + Không chứa các logic nghiệp vụ ngoài một số thao tác hiển thị đơn giản.  
- Ví dụ: Trong một ứng dụng đặt hàng, lớp giao diện người dùng sẽ bao gồm các trang web hiển thị sản phẩm, giỏ hàng, và quy trình thanh toán. Các dữ liệu cần thiết được lấy từ lớp ứng dụng và cập nhật khi người dùng thêm hoặc xoá sản phẩm trong giỏ hàng.

## Lợi ích và thách thức khi áp dụng DDD

### ***Lợi ích***

DDD mang lại nhiều lợi ích rõ ràng trong việc xây dựng các hệ thống phần mềm phức tạp. Một số lợi ích chính bao gồm:

- Giảm sự phức tạp: DDD giúp chia nhỏ các vấn đề phức tạp thành các subdomains, mỗi subdomain sẽ chỉ tập trung vào một phần nhỏ của nghiệp vụ, giúp giảm thiểu sự phức tạp khi phát triển và bảo trì hệ thống.  
- Tăng tính mở rộng: Các subdomains có thể được phát triển và bảo trì độc lập, giúp hệ thống có thể mở rộng một cách linh hoạt mà không gây gián đoạn đến các phần khác của hệ thống.  
- Cải thiện khả năng bảo trì: Việc tổ chức các business logic theo subdomains giúp cho mã nguồn dễ dàng được bảo trì. Khi có thay đổi nghiệp vụ, các thay đổi chỉ cần áp dụng trong subdomain tương ứng mà không ảnh hưởng đến toàn bộ hệ thống.  
- Cải thiện giao tiếp: Sử dụng một ngôn ngữ phổ quát giúp tất cả các bên liên quan có thể giao tiếp và hiểu các yêu cầu nghiệp vụ một cách rõ ràng. Điều này giúp giảm thiểu sự hiểu nhầm và thúc đẩy sự hợp tác giữa các nhóm phát triển và khách hàng.

### ***Thách thức***

- DDD đòi hỏi các nhà phát triển và chuyên gia nghiệp vụ phải hiểu rõ về lĩnh vực mà phần mềm phục vụ. Việc này cần thời gian và sự hợp tác chặt chẽ giữa các bên để xây dựng một Ubiquitous Language (ngôn ngữ phổ quát) thống nhất.

## - Chi phí và thời gian phát triển ban đầu cao, khó áp dụng trong các đội ngũ không đồng nhất.

- Chia nhỏ domain thành các subdomain và Bounded Context có thể làm tăng số lượng thành phần cần quản lý. Điều này đặc biệt khó khăn trong các dự án lớn hoặc khi thiếu công cụ quản lý phù hợp.

- Các nhà phát triển thiếu kinh nghiệm có thể gặp khó khăn trong việc triển khai các khái niệm này.

## - Khi áp dụng DDD vào các hệ thống hiện tại (legacy systems), có thể phát sinh các xung đột về kiến trúc, đặc biệt nếu hệ thống cũ không được xây dựng dựa trên các nguyên tắc tương thích với DDD.

## Triển khai dự án thực tế

### ***Mô tả cấu trúc dự án***

### Dự án được triển khai theo kiến trúc microservices, sử dụng Spring Boot và Spring Cloud cho từng microservice độc lập. Các thành phần chính được tổ chức như sau:

#### ***Các microservice chính***

**- Springboot-microservices**: Chứa các dịch vụ chính được phát triển với Spring Boot. Mỗi dịch vụ có thể bao gồm logic nghiệp vụ riêng biệt, như quản lý sản phẩm hoặc đánh giá.

**- Spring-kafka-microservices**: Tích hợp với Kafka để xử lý các tin nhắn và luồng sự kiện trong hệ thống. Cấu trúc bao gồm:

- S**pringboot-rabbitmq-microservices**: Tích hợp với RabbitMQ để xử lý các tin nhắn giữa các dịch vụ.

#### ***Các dịch vụ hỗ trợ***

**- Api gateway.png**: Hình ảnh mô tả cấu trúc của API Gateway, chịu trách nhiệm quản lý các yêu cầu từ client và phân phối chúng tới các dịch vụ tương ứng.

**- Communication between microservices.png**: Hình ảnh minh họa cách các microservices giao tiếp với nhau..

**- Spring Service registry interface.png**: Hình ảnh về giao diện đăng ký dịch vụ, giúp các microservices có thể tìm và kết nối với nhau thông qua một trung tâm.

**-** C**ircuit breaker.png**: Hình ảnh minh họa cơ chế “circuit breaker” để bảo vệ hệ thống khỏi các lỗi liên dịch vụ.

**- Differences between monolithic and microservices.png:** Hình ảnh so sánh sự khác biệt giữa kiến trúc monolithic và microservices.

#### ***Cấu trúc thư mục trong mỗi microservice***

Trong từng microservice, cấu trúc thư mục được tổ chức nhất quán để dễ dàng bảo trì và phát triển:

- controller: Chứa các lớp Controller, chịu trách nhiệm nhận các yêu cầu từ client, điều phối các dịch vụ nghiệp vụ thích hợp và trả về kết quả. Controller đóng vai trò là điểm truy cập chính cho người dùng vào mỗi microservice.

- service: Chứa các lớp Service thực hiện logic nghiệp vụ chính. Đây là nơi xử lý các quy trình kinh doanh cốt lõi của từng microservice. Các lớp này tập trung vào xử lý nghiệp vụ mà không phụ thuộc vào cách lưu trữ dữ liệu.

- repository: Chứa các lớp Repository để truy xuất và lưu trữ dữ liệu, thường tương tác với cơ sở dữ liệu (SQL, NoSQL, v.v.). Các repository giúp tách biệt logic nghiệp vụ với tầng lưu trữ, giúp tăng khả năng kiểm thử và tái sử dụng.

- entity: Chứa các lớp Entity, đại diện cho các đối tượng chính trong domain của microservice. Các entity phản ánh cấu trúc dữ liệu lưu trữ và được xử lý trong hệ thống, đồng thời đảm bảo tính nhất quán của dữ liệu.

- config: Chứa các cấu hình riêng của từng microservice, bao gồm cấu hình Kafka, kết nối cơ sở dữ liệu, và các cài đặt đặc thù khác.

- resources: Bao gồm các tệp cấu hình như application.yml và bootstrap.yml. Các tệp này chứa các thông tin về môi trường chạy, kết nối cơ sở dữ liệu, và endpoint của các dịch vụ khác.

#### ***Các tệp cấu hình chung***

- application.yml và bootstrap.yml: Các tệp này chứa thông tin về kết nối cơ sở dữ liệu, cấu hình Kafka, thông tin về endpoint của các dịch vụ khác, cũng như các cài đặt bảo mật và caching.

- jvmcc-config-server: Tệp cấu hình trung tâm trên Config Server cung cấp cấu hình cho tất cả các dịch vụ, đảm bảo các dịch vụ luôn đồng bộ và cập nhật.

### ***Thiết kế DDD của chương trình***

### Dựa trên cấu trúc của hệ thống, các thành phần theo mô hình DDD được triển khai như sau:

#### ***Bounded Context***

Trong kiến trúc microservices, mỗi dịch vụ là một Bounded Context riêng biệt, đại diện cho một phần nghiệp vụ cụ thể với logic và mô hình domain riêng:

- Product Service đại diện cho Bounded Context "Product", với các nghiệp vụ quản lý sản phẩm như tạo, sửa, xóa và truy xuất thông tin sản phẩm.

- Review Service đại diện cho Bounded Context "Review", với nghiệp vụ quản lý đánh giá sản phẩm, bao gồm việc thêm và lấy đánh giá từ khách hàng.

#### ***Entity***

- Các Entity là các đối tượng có danh tính duy nhất, đóng vai trò là các thực thể chính trong từng Bounded Context:

- Trong Product Service, Product là một thực thể chính với các thuộc tính như productId, name, price, và description.

- Trong Review Service, Review là thực thể chính chứa thông tin như reviewId, content, rating, và productId.

- Mỗi entity này duy trì danh tính riêng biệt và có khả năng thay đổi trạng thái trong quá trình hoạt động.

#### ***Value Object***

- Value Object là các đối tượng không có danh tính riêng, thường đại diện cho các đặc điểm hoặc thông tin bổ sung của một entity.

- Address là một ví dụ về Value Object nếu có dịch vụ liên quan đến khách hàng. Address có thể bao gồm các thuộc tính như street, city, và postalCode.

#### ***Aggregate và Aggregate Root***

Một Aggregate là nhóm các entity và value object có liên kết chặt chẽ, được quản lý bởi một Aggregate Root:

- Trong Product Service, Product là Aggregate Root quản lý các entity liên quan đến sản phẩm như Category hay Supplier.

- Trong Review Service, Review là một aggregate với Review là Aggregate Root, tập hợp các thông tin đánh giá liên quan đến một sản phẩm.

#### ***Repository***

Repository đóng vai trò là tầng trung gian giữa domain và cơ sở dữ liệu, hỗ trợ lưu trữ và truy vấn các Aggregate:

- ProductRepository trong Product Service chịu trách nhiệm lưu trữ và truy vấn các sản phẩm theo productId.

- ReviewRepository trong Review Service quản lý lưu trữ và tìm kiếm các đánh giá theo reviewId hoặc productId.

#### ***Service***

Service chứa các logic nghiệp vụ không thuộc về một entity hoặc value object cụ thể nào, giúp thực hiện các quy trình nghiệp vụ phức tạp:

- ProductService có thể chứa các phương thức để tính toán giá khuyến mãi hoặc xử lý các điều kiện liên quan đến kho hàng.

- ReviewService có thể xử lý các thao tác liên quan đến việc tổng hợp đánh giá hoặc tính toán điểm trung bình xếp hạng.

#### **Application Layer**

Lớp ứng dụng điều phối quy trình xử lý của hệ thống, đóng vai trò trung gian giữa client và domain layer. Các Controller trong mỗi microservice đóng vai trò là lớp ứng dụng, nhận yêu cầu từ client, sau đó điều phối các phương thức nghiệp vụ thích hợp.

#### **Infrastructure Layer**

Lớp hạ tầng hỗ trợ các dịch vụ cần thiết như kết nối cơ sở dữ liệu, giao tiếp với các microservices khác, và quản lý bảo mật.

- Config Server cung cấp cấu hình tập trung, đảm bảo các dịch vụ đồng bộ.

- Zipkin Server hỗ trợ giám sát hiệu suất, giúp theo dõi các yêu cầu qua hệ thống.

- LDAP và Keycloak đảm bảo quản lý người dùng và bảo mật truy cập cho hệ thống.

### ***Cài đặt dự án***

#### ***Chạy spring-kafka-microservice project***

- Bước 1: Clone dự án từ github: <https://github.com/srinathsai/Building-SpringBoot-MicroServices>

- Bước 2: Thêm file docker-compose-kafka.yml vào thư mục cùng cấp với folder spring-kafka-microservice có nội dung như dưới đây

|  |
| --- |
| version: "3"  services:  zookeeper:  image: confluentinc/cp-zookeeper:7.3.1  container\_name: zookeeper  environment:  ZOOKEEPER\_CLIENT\_PORT: 2181  ZOOKEEPER\_TICK\_TIME: 2000  ports:  - "2181:2181"  networks:  - kafka-network  volumes:  - zookeeper-data:/var/lib/zookeeper/data  - zookeeper-logs:/var/lib/zookeeper/log  kafka:  image: confluentinc/cp-kafka:7.3.1  container\_name: kafka  environment:  KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://localhost:9092  KAFKA\_BROKER\_ID: 1  KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1  KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  depends\_on:  - zookeeper  ports:  - "9092:9092"  networks:  - kafka-network  volumes:  - kafka-data:/var/lib/kafka/data  networks:  kafka-network:  driver: bridge  volumes:  kafka-data:  zookeeper-data:  zookeeper-logs: |

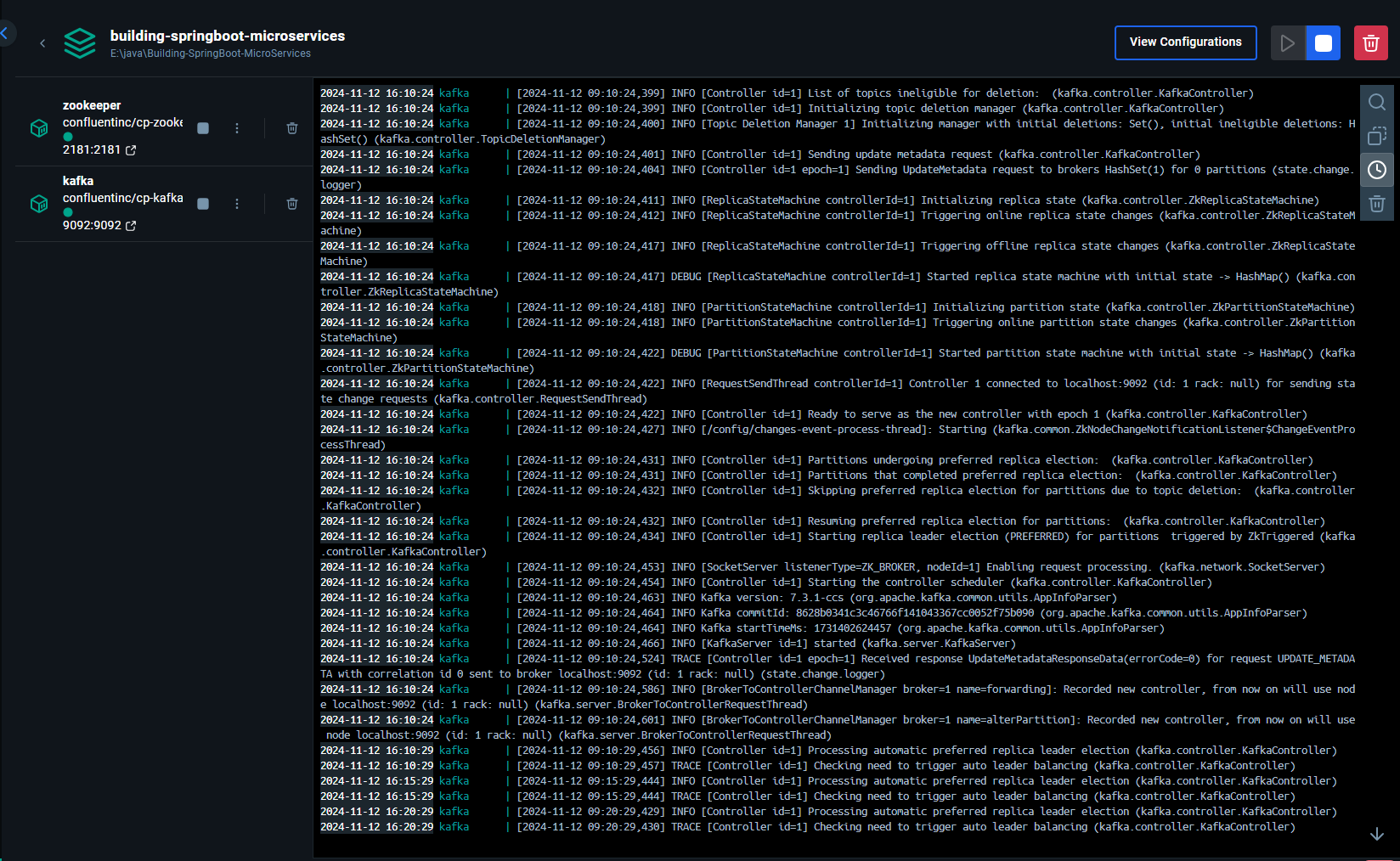
- Bước 3: Cài đặt docker nếu chưa có: <https://www.docker.com/get-started>

- Bước 4: Mở docker compose và chạy câu lệnh sau: docker compose -f docker-compose-kafka.yml up -d

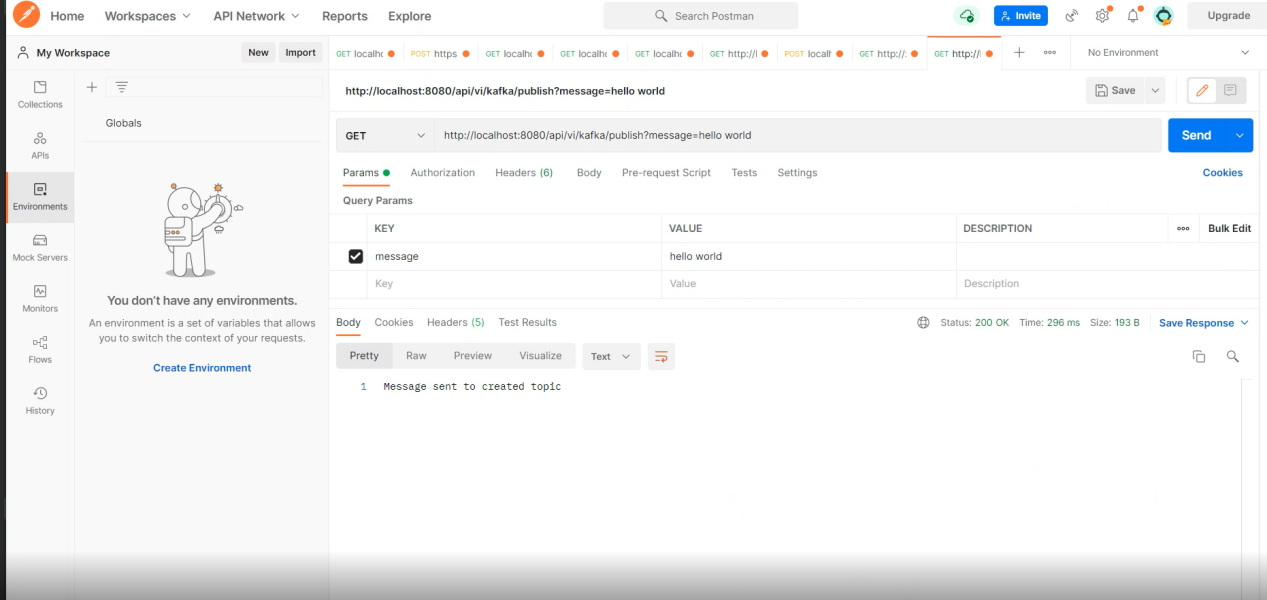
- Bước 5: Đợi quá trình build hoàn tất, các microservice sẽ sẵn sàng để kiểm tra và sử dụng thông qua API Gateway hoặc các công cụ như Postman. Để dừng các container, nhấn Ctrl + C hoặc chạy câu lệnh: docker compose -f docker-compose-kafka.yml down

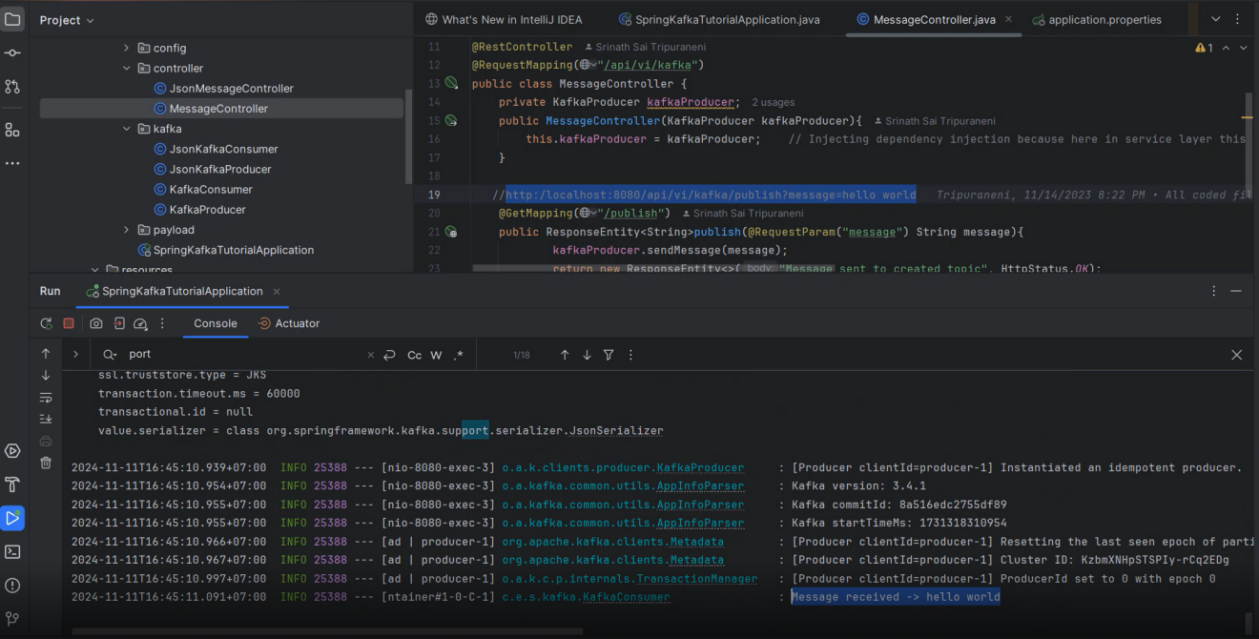
- Bước 6: Kiểm tra

Sau khi hoàn chạy thành công các service bật docker-desktop lên để kiểm tra:



Các service đã được chạy thành công





#### ***Chạy springboot-rabbitmq-microservice project***

- Bước 1: Clone dự án từ github: <https://github.com/srinathsai/Building-SpringBoot-MicroServices>

- Bước 2: Thêm file docker-compose-rebbitmq.yml vào thư mục cùng cấp với folder spring-kafka-microservice có nội dung như dưới đây

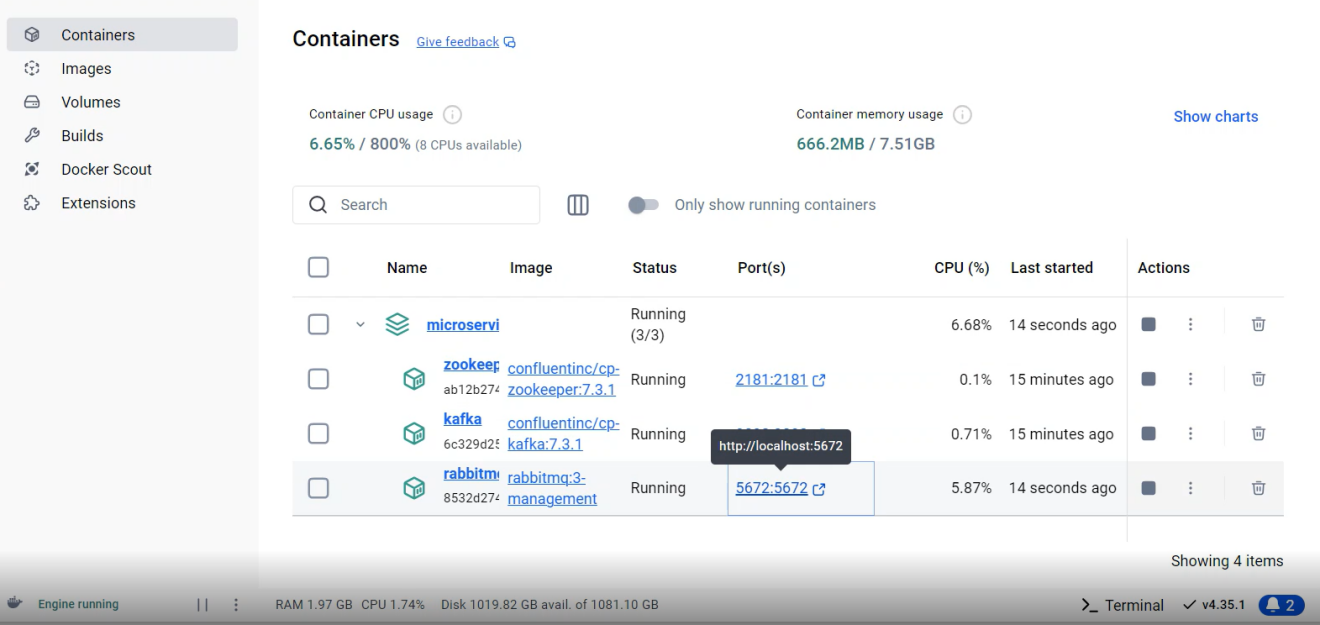
|  |
| --- |
| services:  rabbitmq:  image: rabbitmq:3-management  container\_name: 'rabbitmq'  ports:  - 5672:5672  volumes:  - ~/.docker-conf/rabbitmq/data/:/var/lib/rabbitmq/  - ~/.docker-conf/rabbitmq/log/:/var/log/rabbitmq  networks:  - rabbitmq\_go\_net  networks:  rabbitmq\_go\_net:  driver: bridge |

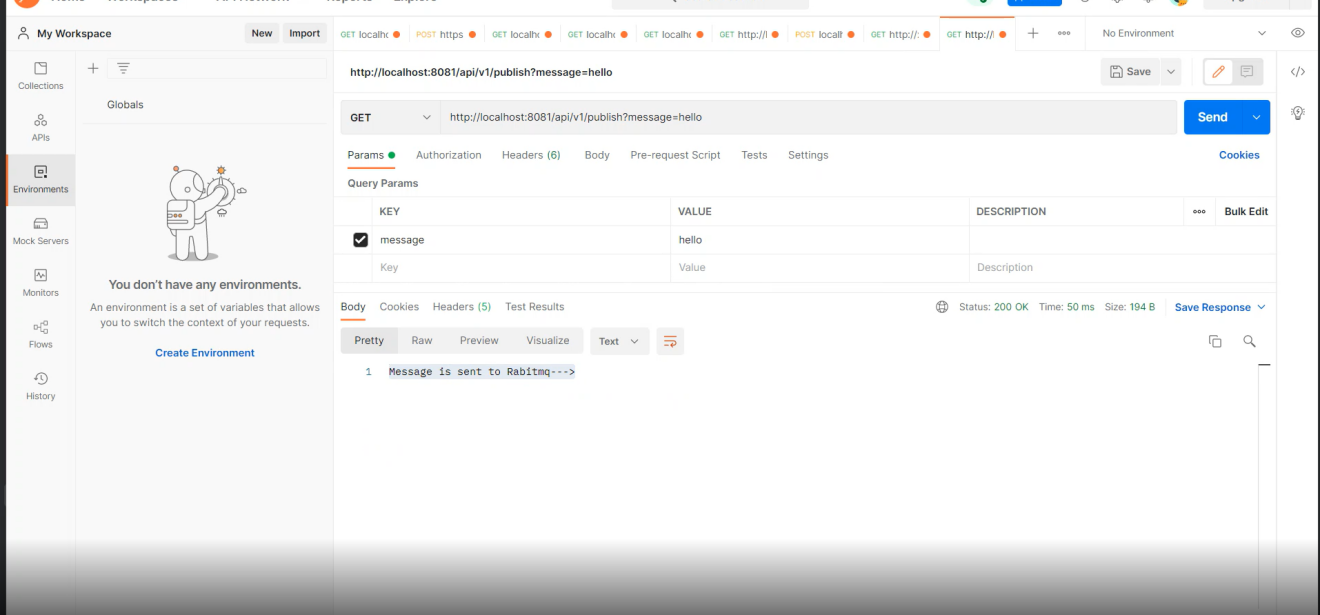
- Bước 3: Cài đặt docker nếu chưa có: <https://www.docker.com/get-started>

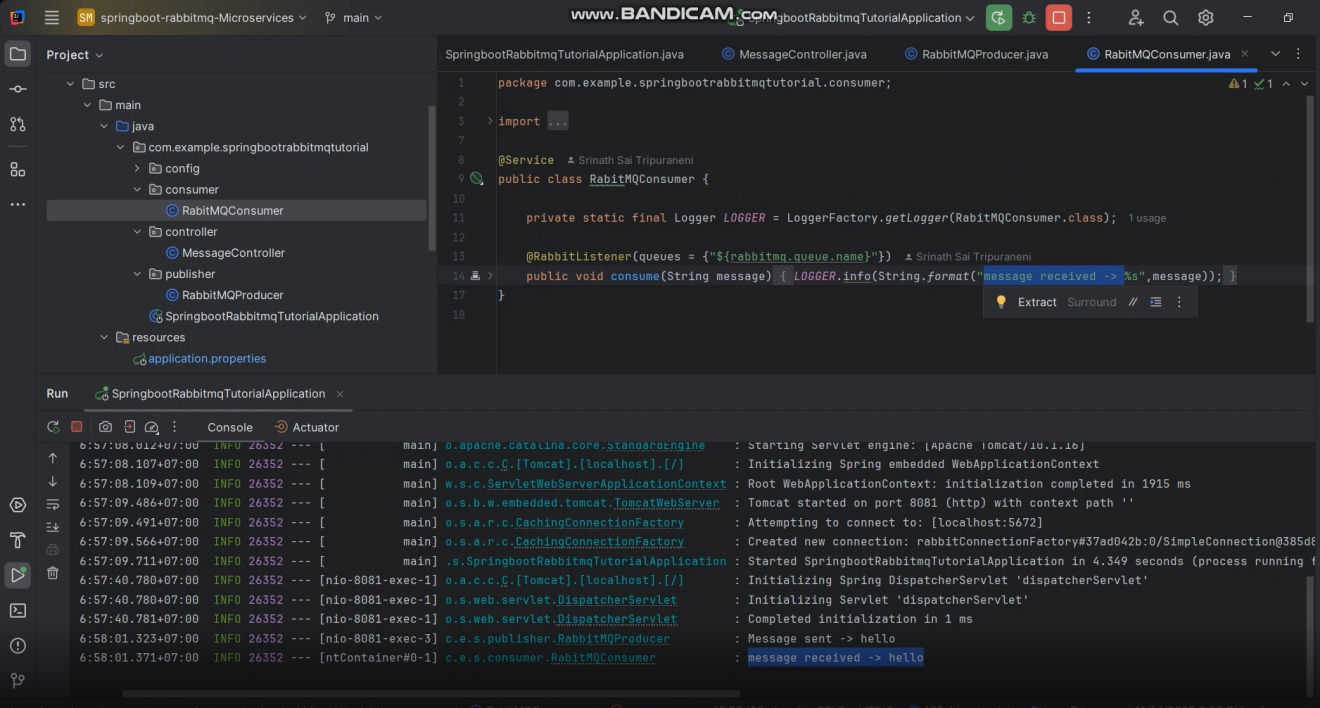
- Bước 4: Mở docker compose và chạy câu lệnh: docker compose -f docker-compose-rabbitmq.yml up -d

- Bước 5: Đợi quá trình build hoàn tất, các microservice sẽ sẵn sàng để kiểm tra và sử dụng thông qua API Gateway hoặc các công cụ như Postman. Để dừng các container, nhấn Ctrl + C hoặc chạy câu lệnh: docker compose -f docker-compose-kafka.yml down

- Bước 6: Kiểm tra







### ***Triển khai với Docker Swarm***

#### ***Giới thiệu Docker Swarm***

## - Docker Swarm là một công cụ quản lý cụm (cluster) được tích hợp trực tiếp vào Docker, cho phép triển khai và quản lý nhiều container Docker trên một hoặc nhiều máy chủ. Docker Swarm giúp tổ chức các container thành các dịch vụ phân tán và chịu trách nhiệm cân bằng tải, tự động phục hồi và quản lý các tài nguyên hiệu quả.

- Trong dự án này, chúng ta sẽ triển khai Docker Swarm để quản lý các microservices trong kiến trúc Docker. Docker Swarm sẽ giúp việc mở rộng và quản lý các dịch vụ trở nên đơn giản và hiệu quả hơn.

#### ***Lợi ích của việc triển khai với Docker Swarm***

## - Quản lý phân tán: Docker Swarm giúp bạn quản lý một cụm các máy chủ Docker một cách dễ dàng và linh hoạt.

- Cân bằng tải tự động: Docker Swarm tự động phân phối các yêu cầu đến các dịch vụ tương ứng và đảm bảo rằng tất cả các dịch vụ đều có thể truy cập được từ các client.

- Khả năng phục hồi cao: Khi một container hoặc node gặp sự cố, Docker Swarm sẽ tự động khởi động lại các container để duy trì sự ổn định của hệ thống.

- Dễ dang mở rộng: Docker Swarm cho phép bạn mở rộng các dịch vụ bằng cách thêm máy chủ vào cụm Docker một cách dễ dàng mà không làm gián đoạn dịch vụ.

#### ***Triển khai dự án với Docker Swarm***

Dự án triển khai các microservices sử dụng kiến trúc Docker, kết hợp với Docker Swarm để quản lý và điều phối các dịch vụ container trên các máy chủ khác nhau. Dự án bao gồm các dịch vụ chính sau:

## - Springboot-microservices: Chứa các dịch vụ chính được phát triển với Spring Boot. Mỗi dịch vụ có thể bao gồm logic nghiệp vụ riêng biệt, như quản lý sản phẩm hoặc đánh giá.

## - Spring-kafka-Microservices: Tích hợp với Kafka để xử lý các tin nhắn và luồng sự kiện trong hệ thống.

## - Springboot-rabbitmq-Microservices: Tích hợp với RabbitMQ để xử lý các tin nhắn giữa các dịch vụ.

## - Bước 1: Cài đặt docker và docker swarm - Trước khi triển khai dự án, đảm bảo Docker đã được cài đặt trên tất cả các máy chủ tham gia vào cụm Docker Swarm. Sau đó, khởi tạo Swarm bằng câu lệnh: docker swarm init

## - Bước 2: Cấu hình Docker Compose cho Docker Swarm - Bạn cần đảm bảo rằng trong tệp docker-compose.yml, bạn chỉ định các dịch vụ dưới dạng replicas để Docker Swarm có thể quản lý các bản sao của container.

|  |
| --- |
| version: "3.8"  services:  zookeeper:  image: confluentinc/cp-zookeeper:7.3.1  container\_name: zookeeper  environment:  ZOOKEEPER\_CLIENT\_PORT: 2181  ZOOKEEPER\_TICK\_TIME: 2000  ports:  - "2181:2181"  networks:  - kafka-network  volumes:  - zookeeper-data:/var/lib/zookeeper/data  - zookeeper-logs:/var/lib/zookeeper/log  deploy:  replicas: 1  restart\_policy:  condition: on-failure  kafka:  image: confluentinc/cp-kafka:7.3.1  container\_name: kafka  environment:  KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092  KAFKA\_BROKER\_ID: 1  KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1  KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  ports:  - "9092:9092"  networks:  - kafka-network  volumes:  - kafka-data:/var/lib/kafka/data  depends\_on:  - zookeeper  deploy:  replicas: 1  restart\_policy:  condition: on-failure  rabbitmq:  image: rabbitmq:3-management  container\_name: rabbitmq  ports:  - "5672:5672"  - "15672:15672" # Management console port  volumes:  - rabbitmq-data:/var/lib/rabbitmq  - rabbitmq-logs:/var/log/rabbitmq  networks:  - rabbitmq\_go\_net  deploy:  replicas: 1  restart\_policy:  condition: on-failure  networks:  kafka-network:  driver: overlay  rabbitmq\_go\_net:  driver: overlay  volumes:  kafka-data:  zookeeper-data:  zookeeper-logs:  rabbitmq-data:  rabbitmq-logs: |

## - Bước 3: Khởi động docker stack, chạy câu lệnh sau để khởi động toàn bộ các dịch vụ đã được định nghĩa trong docker-sompose.yml: docker stack deploy -c docker-compose.yml jvmcc-stack.

## - Bước 4: Kiểm tra trạng thái triển khai bằng cách sử dụng câu lệnh: docker stack ps jvmcc-stack.

## 

# **CHƯƠNG 2. PHÁT HIỆN XE THEO THỜI GIAN THỰC BẰNG PYTHON VÀ OPENCV**

## Giới thiệu và mô tả bài toán

Dự án "Phát Hiện Xe Theo Thời Gian Thực" là một ứng dụng sử dụng ngôn ngữ lập trình Python và thư viện OpenCV, sử dụng kỹ thuật thị giác máy tính hiện đại để nhận dạng các xe trong video trực tiếp theo thời gian thực thông qua camera hoặc video đã được ghi sẵn, với khả năng xử lý và phân tích hình ảnh một cách nhanh chóng.

Cốt lõi của dự án được xây dựng dựa trên bộ phân loại Haar Cascade, một công cụ mạnh mẽ trong lĩnh vực nhận dạng đối tượng. Haar Cascade cho phép nhận diện xe trong video bằng cách xác định các đặc điểm đặc trưng của xe (như hình dáng và các chi tiết nhận dạng). Khi phát hiện được xe, ứng dụng sẽ vẽ các khung nhận dạng quanh xe để dễ dàng theo dõi và phân biệt các phương tiện trong khung hình. Việc này không chỉ giúp người dùng theo dõi hành trình của xe trong thời gian thực mà còn giúp tạo ra các bản ghi có thể phân tích được sau này.

Dự án phát hiện xe theo thời gian thực hiện nay được nhiều công ty và tổ chức sử dụng rộng rãi trong nhiều lĩnh vực quan trọng. Ứng dụng của công nghệ này rất đa dạng, bao gồm quản lý giao thông, giám sát hành trình xe và cảm biến hỗ trợ xe đỗ …

Để triển khai dự án này, người dùng cần cài đặt Python phiên bản từ 3.9 trở lên, thư viện OpenCV và một môi trường IDE tương thích như PyCharm, Anaconda, Visual Studio Code ...

Dự án đã được đính kèm theo mã nguồn, cho phép người dùng tải về, cài đặt và chạy ngay. Đây là một cách tuyệt vời để hiểu sâu hơn về cách hoạt động của hệ thống theo dõi và phát hiện xe. Dự án không chỉ cung cấp kiến thức nền tảng về thị giác máy tính và xử lý hình ảnh mà còn giúp người dùng thực hành và cải thiện kỹ năng lập trình Python, đặc biệt trong lĩnh vực AI và thị giác máy tính.

## Bạn có thể tìm hiểu chi tiết hơn về dự án và tải nguồn mã tại liên kết sau: https://itsourcecode.com/free-projects/python-projects/real-time-car-detection-opencv-python-with-source-code

## Phân tích cấu trúc chương trình

## Chương trình này được viết bằng ngôn ngữ Python và sử dụng thư viện OpenCV để phát hiện và nhận dạng xe trong một video. Cấu trúc của chương trình có thể được phân tích chi tiết như sau:

## **Import các thư viện cần thiết**

|  |
| --- |
| import numpy as np  Import cv2 |

## - *numpy*: Đây là thư viện được dùng phổ biến cho các thao tác tính toán và xử lý ma trận, mặc dù trong mã này numpy chưa được sử dụng trực tiếp.

## - *cv2*: Thư viện OpenCV giúp xử lý hình ảnh và video trong Python

## ***Định nghĩa đường dẫn đến bộ phân loại Haar Cascade***

|  |
| --- |
| cascade\_src = 'cars.xml' |

## - *cascade\_src:* Đây là đường dẫn tới file .xml chứa bộ phân loại Haar Cascade được huấn luyện sẵn để nhận diện xe. File này chứa các đặc trưng nhận diện xe dưới dạng ma trận phân loại Haar.

## ***Khai báo mã nguồn video***

|  |
| --- |
| video = 'data/video1.avi' |

## ***Định nghĩa hàm detectCars***

|  |
| --- |
| def detectCars(filename):  rectangles = []  cascade = cv2.CascadeClassifier(cascade\_src)  vc = cv2.VideoCapture(filename) |

## - *Hàm detectCars*: Nhận vào tên file video làm tham số (filename), sau đó sẽ xử lý từng khung hình để phát hiện và vẽ khung xe.

## - *rectangles*: Một danh sách trống, nhưng hiện tại chưa được sử dụng trong chương trình.

## *- cascade*: Tạo một bộ phân loại Haar Cascade từ file cascade\_src bằng cách sử dụng cv2.CascadeClassifier(cascade\_src). Bộ phân loại này sẽ được dùng để phát hiện xe.

## - *vc*: Đối tượng cv2.VideoCapture để mở file video.

## ***Kiểm tra việc mở video***: Kiểm tra xem video có được mở thành công hay không. Nếu thành công, chương trình sẽ đọc khung hình đầu tiên (vc.read()), đồng thời trả về True cho rval. Nếu thất bại, rval được đặt thành False

|  |
| --- |
| if vc.isOpened():  rval, frame = vc.read()  else:  rval = False |

## ***Vòng lặp xử lý khung hình***

|  |
| --- |
| while rval:  rval, frame = vc.read()  frameHeight, frameWidth, fdepth = frame.shape |

## *- while rval*: Vòng lặp này sẽ tiếp tục chạy khi rval là True, tức là còn khung hình mới.

## - *vc.read()*: Đọc từng khung hình từ video và lưu vào frame. rval được cập nhật sau mỗi lần đọc, nên vòng lặp sẽ tự động kết thúc khi không còn khung hình để đọc.

## - *frame.shape*: Lấy kích thước khung hình, bao gồm chiều cao (frameHeight), chiều rộng (frameWidth), và độ sâu màu (fdepth).

## ***Thay đổi kích thước khung hình và chuyển sang thang độ xám***

|  |
| --- |
| frame = cv2.resize(frame, (600, 400))  gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) |

## - *cv2.resize*: Khung hình được thay đổi kích thước về 600x400 để giảm tải xử lý.

## - *cv2.cvtColor*: Chuyển khung hình sang thang độ xám để tối ưu tốc độ và độ chính xác của bộ phân loại, vì việc phát hiện chỉ dựa trên cường độ ánh sáng mà không cần màu sắc.

## ***Phát hiện xe***

|  |
| --- |
| cars = cascade.detectMultiScale(gray, 1.3, 3) |

## - *cascade.detectMultiScale*: Đây là phương thức phát hiện đối tượng của OpenCV và trả về một danh sách các toạ độ hình chữ nhật bao quanh các xe phát hiện được. Trong đó:

+ *gray*: Là khung hình đầu vào, đã chuyển sang thang độ xám.

+ *1.3*: Hệ số scale factor để điều chỉnh kích thước ảnh ở mỗi bước của quá trình phát hiện (giá trị càng lớn, phát hiện càng nhanh nhưng có thể bỏ sót).

+ *3*: Số lượng vùng lân cận tối thiểu để xác nhận đó là một đối tượng (giá trị càng lớn, phát hiện càng chính xác nhưng dễ bỏ sót).

1. ***Vẽ khung chữ nhật quanh các xe được phát hiện***

|  |
| --- |
| for (x, y, w, h) in cars:  cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2) |

*- Vòng lặp for*: Lặp qua từng đối tượng xe phát hiện được, với mỗi đối tượng có các toạ độ (x, y) và kích thước (w, h).

- *cv2.rectangle*: Vẽ một khung chữ nhật màu đỏ (0, 0, 255) xung quanh đối tượng phát hiện được. Độ dày khung hình chữ nhật là 2.

1. ***Hiển thị kết quả và xử lý thoát chương trình***

|  |
| --- |
| cv2.imshow("Result", frame)  if cv2.waitKey(33) == ord('q'):  break |

- *cv2.imshow*: Hiển thị khung hình hiện tại với tên cửa sổ là "Result".

- *cv2.waitKey(33)*: Đợi 33ms giữa các khung hình, giúp hiển thị khung hình với tốc độ khoảng 30fps (1000ms / 33 ≈ 30). Nếu phím 'q' được nhấn, vòng lặp sẽ dừng

1. ***Giải phóng tài nguyên***

|  |
| --- |
| vc.release() |

1. ***Gọi hàm detectCars***

|  |
| --- |
| detectCars(video) |

## Các hàm API được sử dụng

Chương trình trên sử dụng một số hàm API từ thư viện OpenCV (cv2), mỗi hàm có một mục đích cụ thể trong việc xử lý hình ảnh và video. Dưới đây là phân tích chi tiết về các hàm API này:

### ***cv2.CascadeClassifier(cascade\_src)***

- *Mô tả*: Tạo một bộ phân loại Haar Cascade để phát hiện đối tượng trong ảnh.

- *Tham số truyền vào*: cascade\_src là đường dẫn tới file .xml chứa dữ liệu của bộ phân loại Haar Cascade (trong trường hợp này là cars.xml).

- *Công dụng*: Bộ phân loại này được dùng để nhận diện đối tượng (xe) trong hình ảnh hoặc video bằng cách phân tích các đặc trưng nhận dạng đã được huấn luyện sẵn.

- *Kết quả*: Trả về một đối tượng CascadeClassifier dùng để phát hiện đối tượng trong ảnh.

### ***cv2.VideoCapture(filename)***

- *Mô tả*: Mở một file video hoặc một thiết bị camera để lấy nguồn dữ liệu đầu vào cho xử lý video.

- *Tham số truyền vào*: filename là đường dẫn tới file video hoặc ID của camera (ví dụ: 0 cho camera mặc định).

- *Công dụng*: Tạo đối tượng VideoCapture cho phép đọc từng khung hình của video để xử lý.

- *Kết quả*: Trả về đối tượng VideoCapture. Bạn có thể sử dụng đối tượng này để đọc các khung hình liên tiếp từ video.

### ***vc.isOpened()***

- *Mô tả*: Kiểm tra xem video hoặc camera đã được mở thành công hay chưa.

- *Công dụng*: Tránh lỗi khi mở video hoặc camera; nếu không mở được, hàm sẽ trả về False.

- *Kết quả*: Trả về True nếu video hoặc camera đã được mở thành công, False nếu ngược lại.

### ***vc.read()***

- *Mô tả*: Đọc một khung hình từ video hoặc camera

- *Công dụng*: Dùng để truy xuất từng khung hình liên tiếp từ video cho các bước xử lý sau.

- *Kết quả*: Trả về hai giá trị

+ rval: True nếu đọc thành công khung hình, False nếu không còn khung hình nào để đọc.

+ frame: Khung hình hiện tại (dưới dạng ma trận màu BGR).

### ***frame.shape***

- *Mô tả*: Lấy kích thước của khung hình hiện tại

- *Công dụng*: Lấy thông tin về chiều cao (frameHeight), chiều rộng (frameWidth), và độ sâu màu (fdepth) của khung hình

- *Kết quả*: Trả về một tuple chứa chiều cao, chiều rộng và độ sâu màu của khung hình

### ***cv2.resize(frame, (600, 400))***

- *Mô tả*: Thay đổi kích thước của khung hình

- *Tham số truyền vào*:

+ frame: Khung hình cần thay đổi kích thước.

+ (600, 400): Kích thước mong muốn (rộng 600 và cao 400).

- *Công dụng*: Giảm kích thước khung hình để tăng tốc độ xử lý.

- *Kết quả*: Trả về khung hình đã được thay đổi kích thước.

### ***cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)***

- *Mô tả*: Chuyển khung hình từ không gian màu BGR sang thang độ xám.

- *Tham số truyền vào*:

+ frame: Khung hình đầu vào ở định dạng BGR (màu).

+ cv2.COLOR\_BGR2GRAY: Mã chuyển đổi từ không gian màu BGR sang grayscale.

- *Công dụng*: Giảm độ phức tạp của hình ảnh, giúp tăng hiệu quả và tốc độ của bộ phân loại Haar Cascade.

- *Kết quả*: Trả về khung hình ở thang độ xám.

### ***cascade.detectMultiScale(gray, 1.3, 3)***

- *Mô tả*: Phát hiện các đối tượng trong khung hình bằng bộ phân loại Haar Cascade.

- *Tham số truyền vào*:

+ gray: Khung hình đầu vào ở thang độ xám.

+1.3: Hệ số scale factor, làm giảm kích thước của khung hình ở mỗi bước.

+3: Số lượng vùng lân cận cần thiết để xác nhận đối tượng.

- *Công dụng*: Dùng để phát hiện các đối tượng (xe) trong khung hình dựa trên bộ phân loại Haar Cascade.

- *Kết quả*: Trả về danh sách các toạ độ hình chữ nhật bao quanh các đối tượng phát hiện được trong khung hình.

### ***cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)***

- *Mô tả*: Vẽ hình chữ nhật trên khung hình

- *Tham số truyền vào*:

+ frame: Khung hình cần vẽ hình chữ nhật.

+ (x, y): Tọa độ góc trên bên trái của hình chữ nhật.

+ (x + w, y + h): Tọa độ góc dưới bên phải của hình chữ nhật.

+ (0, 0, 255): Màu sắc của đường viền hình chữ nhật (đỏ).

+ 2: Độ dày của đường viền.

- *Công dụng*: Đánh dấu vùng chứa đối tượng phát hiện (xe) trên khung hình.

- *Kết quả*: Thêm một khung chữ nhật vào vị trí của đối tượng phát hiện được.

### ***cv2.imshow("Result", frame)***

- *Mô tả*: Hiển thị khung hình trên một cửa sổ mới.

- *Tham số truyền vào*:

+ "Result": Tên của cửa sổ hiển thị.

+ frame: Khung hình cần hiển thị.

- *Công dụng*: Giúp quan sát quá trình phát hiện đối tượng trên khung hình

- *Kết quả*: Tạo một cửa sổ và hiển thị khung hình hiện tại với các khung nhận dạng đã vẽ.

### ***cv2.waitKey(33)***

- *Mô tả*: Đợi một khoảng thời gian (tính bằng ms) trước khi chuyển sang khung hình tiếp theo.

- *Tham số truyền vào*: 33 là thời gian chờ (33ms) giữa mỗi khung hình (khoảng 30 fps).

- *Công dụng*: Cho phép người dùng nhấn phím để thoát khỏi vòng lặp hoặc để tạo hiệu ứng hiển thị video ở tốc độ gần với tốc độ thực.

- *Kết quả*: Nếu người dùng nhấn phím q (ASCII của 'q'), chương trình sẽ thoát khỏi vòng lặp.

### ***vc.release()***

- *Mô tả*: Giải phóng đối tượng VideoCapture khi không còn cần thiết.

- *Công dụng*: Đóng video hoặc giải phóng tài nguyên camera, tránh rò rỉ tài nguyên.

- *Kết quả*: Đối tượng VideoCapture được đóng và giải phóng.

## Cài đặt và triển khai

### ***Cài đặt môi trường***

## Trước tiên, chúng ta cần phải thiết lập một môi trường IDE phù hợp, chẳng hạn như PyCharm, Anaconda, hoặc Visual Studio Code ... Tuy nhiên, PyCharm vẫn được khuyến nghị cài đặt để mang lại sự tiện lợi và dễ dàng sử dụng nhất. Để cài đặt PyCharm chúng ta cần thực hiện theo các bước dưới đây:

## - *Bước 1*: Truy cập vào đường dẫn sau và chọn Download để tải file cài đặt: <https://www.jetbrains.com/pycharm/download/?section=windows>

## 

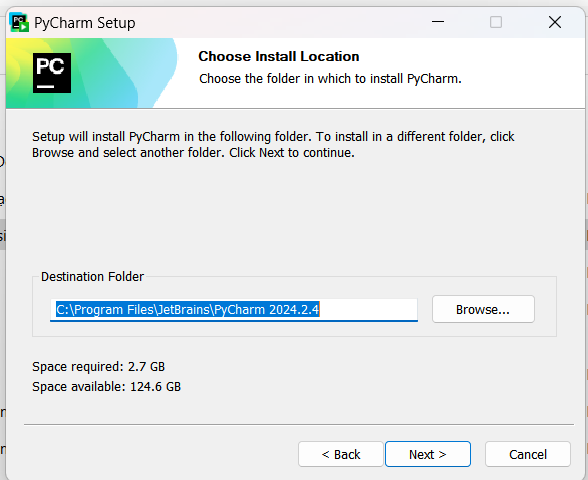
- *Bước 2*: Chạy file pycharm-professional-2024.2.4 đã tải về.

## 

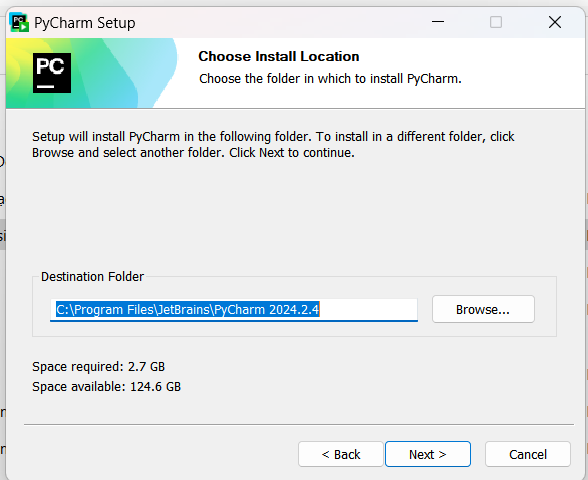
- *Bước 3*: Chọn Next.

## 

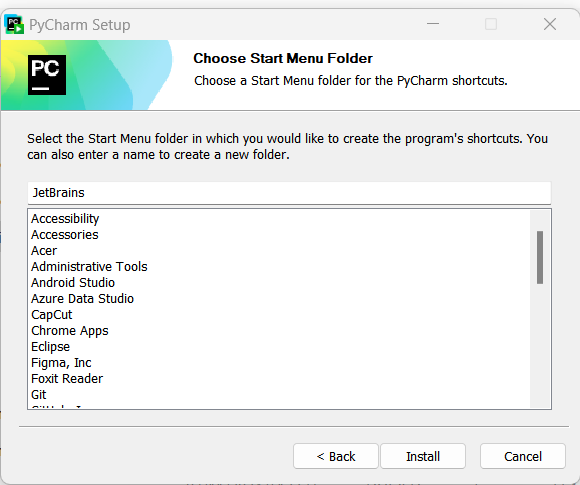
- *Bước 4*: Tại đây bạn có thể thay đổi nơi lưu trữ cài đặt chương trình bằng cách chọn Browse. Nếu không muốn thay đổi gì nữa thì ta chọn Next.



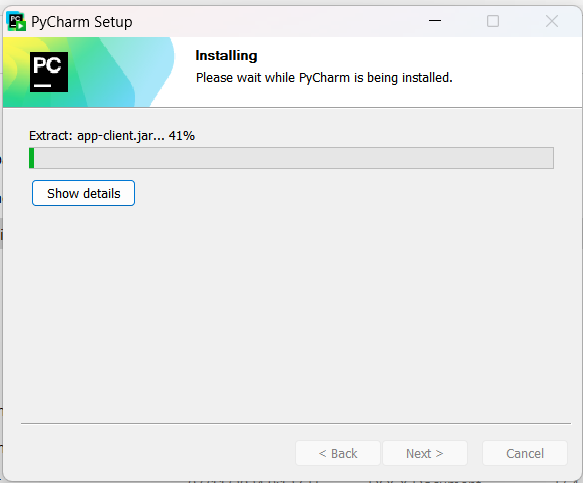
- *Bước 5*: Tiếp theo ta có thể chọn một số tuỳ chọn như create desktop shortcut, add “open folder as project” … Sau đó ta tiếp tục chọn Next.



- *Bước 6*: Ở đây ta giữ nguyên mặc định là JetBrains và chọn Install.



- *Bước 7*: Đợi chương trình được cài đặt.

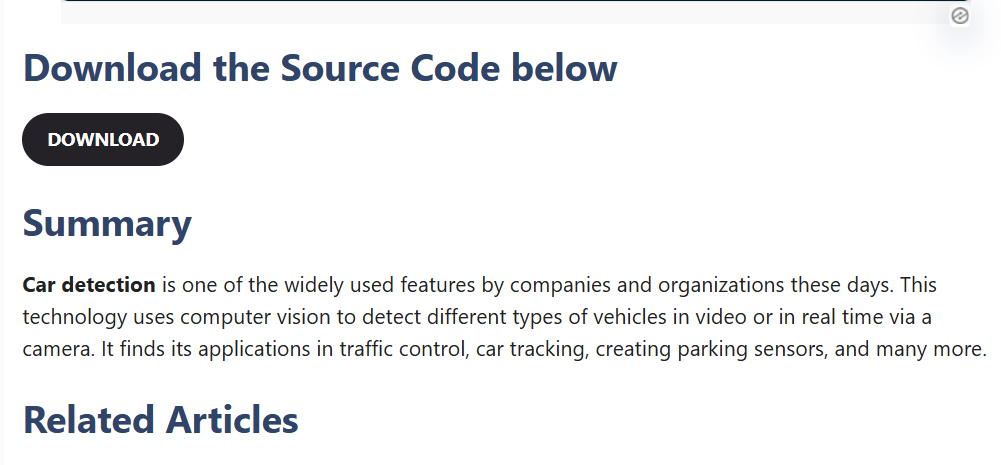


- *Bước 8*: Sau khi chương trình được cài đặt xong, ta chọn Finish. PyCharm đã được cài đặt thành công.

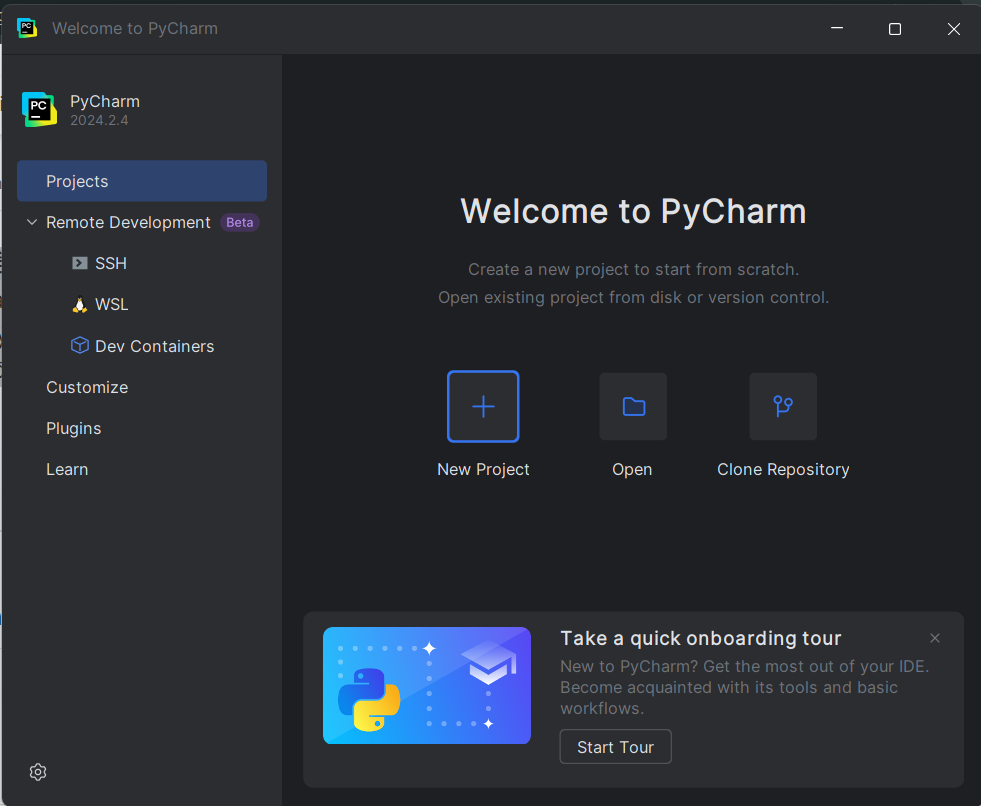
## 

### ***Triển khai dự án***

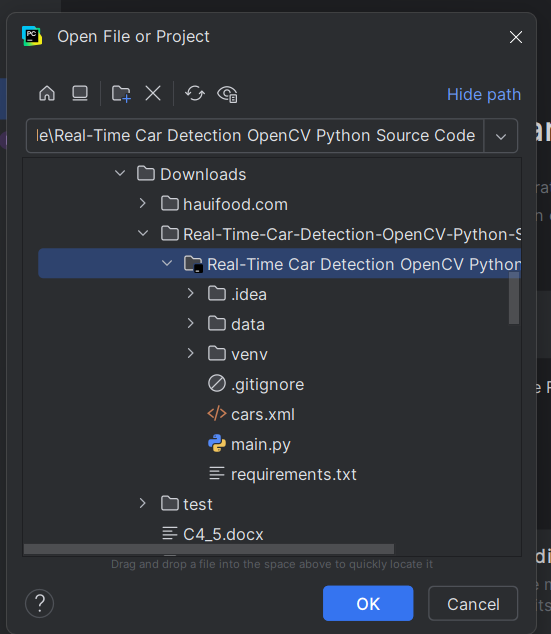
- *Bước 1*: Truy cập đường dẫn sau và tải mã nguồn của dự án về máy: <https://itsourcecode.com/free-projects/python-projects/real-time-car-detection-opencv-python-with-source-code>



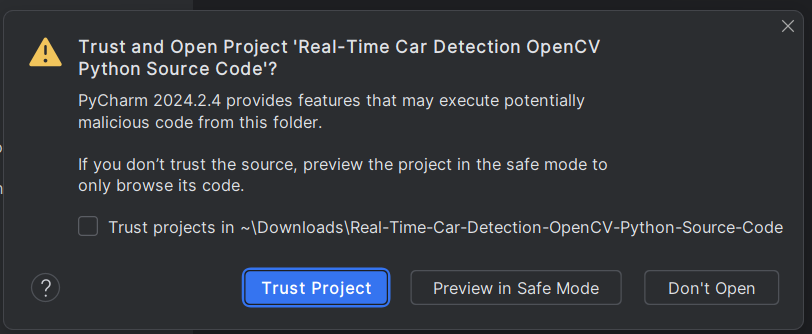
- *Bước 2*: Mở PyCharm 2024.2.4 đã cài đặt lên.



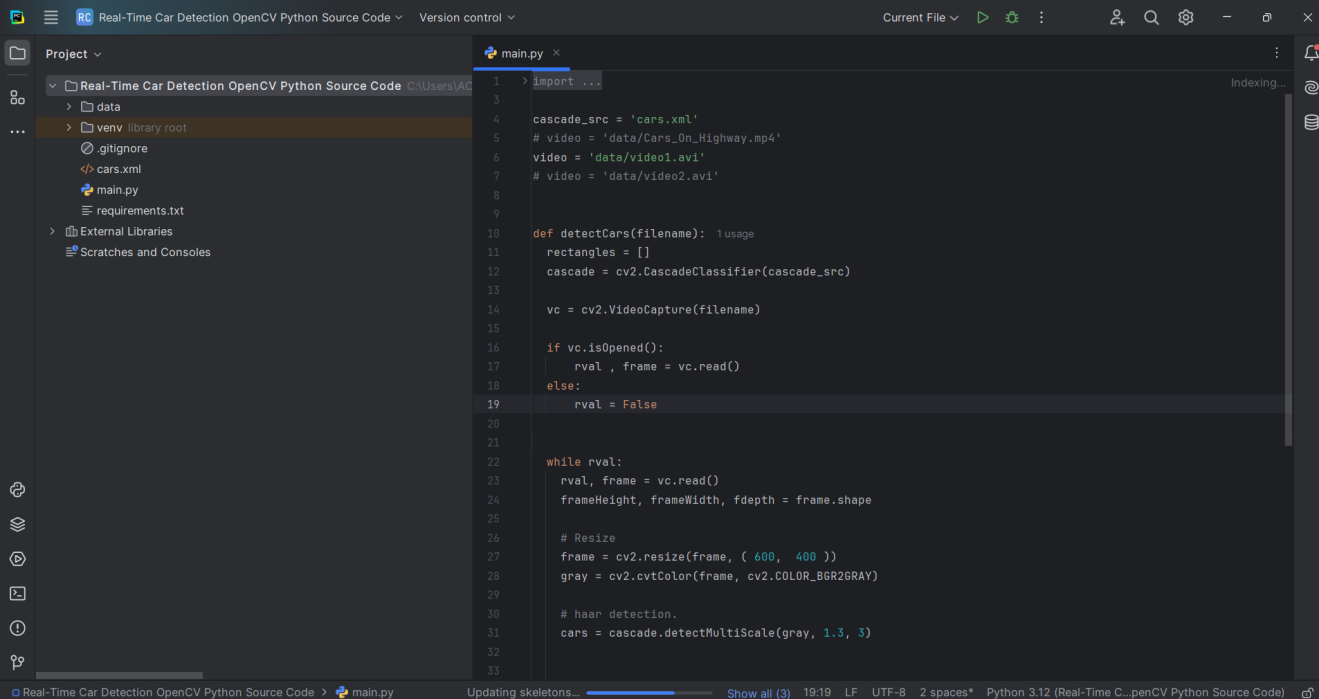
- *Bước 3*: Chọn Open và select tới dự án đã được tải về trước đó và chọn OK.



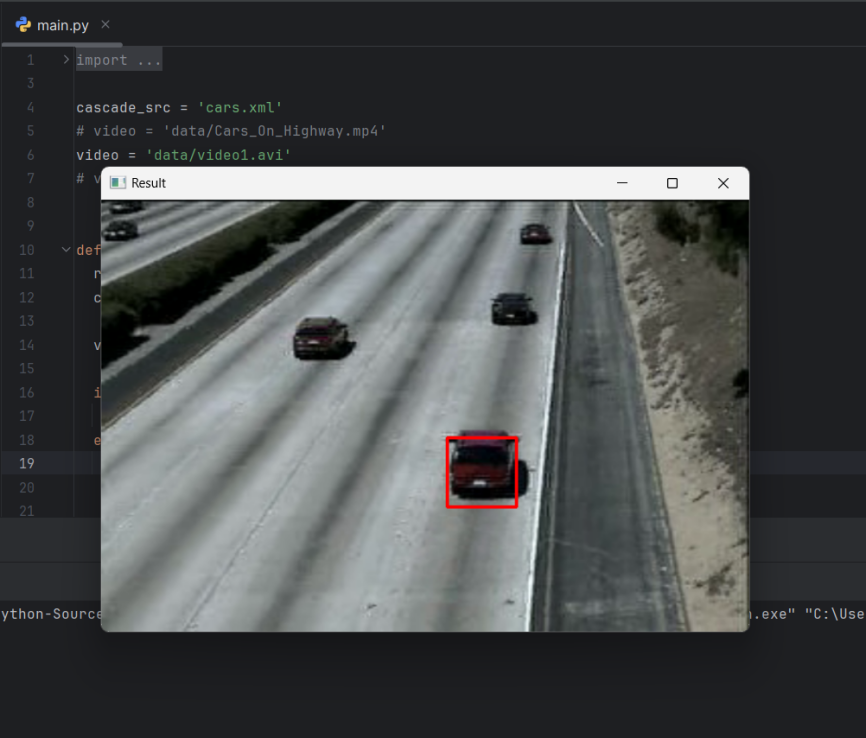
- *Bước 4*: Phần mềm sẽ hiện lên màn hình xác nhận và chọn Trust Project.



- *Bước 5*: Select tới file main.py, mở nó lên và click vào biểu tượng hình tam giác ở phía trên thanh taskbar hoặc chuột phải -> Run “main”.



- *Bước 6*: Sau khi file được chạy lên thành công và đây là kết quả của chương trình.



## Kết quả và đánh giá

## Chương trình trên có mục đích phát hiện xe từ một video hoặc camera theo thời gian thực, sử dụng thư viện OpenCV và bộ phân loại Haar Cascade. Dưới đây là phân tích chi tiết kết quả của chương trình sau khi nó chạy:

### ***Kết quả phát hiện xe***

### - Chương trình sẽ duyệt qua từng khung hình (frame) trong video và áp dụng bộ phân loại Haar Cascade để phát hiện các đối tượng xe.

- Mỗi khi bộ phân loại xác định được một chiếc xe, chương trình sẽ ghi nhận vị trí của xe bằng cách tạo một khung chữ nhật (bounding box) bao quanh xe trên khung hình.

- Khung chữ nhật này có màu đỏ (0, 0, 255) và có độ dày viền là 2 pixel. Việc sử dụng khung hình giúp người dùng dễ dàng xác định vị trí của xe trong khung hình.

### ***Hiển thị khung hình có chứa các đối tượng được phát hiện***

### - Khung hình với các đối tượng được phát hiện sẽ được hiển thị trong một cửa sổ tên là "Result".

### - Cửa sổ này sẽ hiển thị liên tục các khung hình từ video theo thời gian thực, tạo ra một chuỗi hình ảnh động mà người dùng có thể quan sát.

### - Người dùng có thể theo dõi vị trí của các xe di chuyển qua các khung hình nhờ các khung nhận dạng được vẽ.

### ***Tốc độ hiển thị và độ chính xác***

### - Chương trình sử dụng cv2.waitKey(33) để đặt tốc độ khung hình ở mức 30 fps (frames per second), gần với tốc độ thực của một video thông thường. Điều này giúp video phát hiện xe hiển thị mượt mà và gần với thời gian thực.

### - Tuy nhiên, tốc độ và độ chính xác của phát hiện phụ thuộc vào nhiều yếu tố:

+ *Chất lượng của video*: Độ phân giải cao có thể cải thiện độ chính xác nhưng có thể làm giảm tốc độ.

+ *Thông số bộ phân loại*: Các thông số như scaleFactor (1.3) và minNeighbors (3) ảnh hưởng đến số lượng xe được phát hiện.

+ *Hạn chế của Haar Cascade*: Bộ phân loại Haar Cascade có thể không phát hiện xe ở các góc độ phức tạp hoặc khi ánh sáng yếu.

### ***Điều khiển dừng chương trình***

### - Chương trình sẽ tiếp tục phát hiện và hiển thị các khung hình cho đến khi người dùng nhấn phím 'q'.

### - Khi phím 'q' được nhấn, vòng lặp dừng lại, chương trình kết thúc và video sẽ đóng lại.

### - Giải phóng tài nguyên sau khi kết thúc

### - Khi người dùng nhấn phím 'q' để thoát, chương trình sẽ giải phóng đối tượng VideoCapture, đóng kết nối với video hoặc camera, và giải phóng các tài nguyên khác liên quan đến OpenCV.

### - Điều này đảm bảo rằng không có tài nguyên nào bị rò rỉ và chương trình kết thúc một cách gọn gàng.

### ***Ứng dụng tiền năng của chương trình***

Chương trình này là một ví dụ cơ bản nhưng có thể áp dụng cho nhiều ứng dụng khác nhau trong thực tế:

**- *Quản lý giao thông***: Phát hiện và theo dõi lưu lượng xe để phân tích hoặc cảnh báo về tắc nghẽn giao thông.

**- *Giám sát an ninh***: Theo dõi phương tiện trong các khu vực yêu cầu bảo mật cao, như bãi đỗ xe hoặc đường vào các cơ quan.

## ****-** *Hệ thống hỗ trợ đỗ xe***: Giúp nhận diện các phương tiện trong bãi đỗ xe để điều phối và giám sát vị trí đỗ.

# **CHƯƠNG 3. XÂY DỰNG HỆ THỐNG IOT VỚI CÔNG NGHỆ IOT CỦA MICROSOFT**

## Tổng quan về IoT

### ***3.1.1. Khái niệm IoT***

Internet of Things (IoT) là thuật ngữ dùng để chỉ các đối tượng có thể được nhận biết cũng như sự tồn tại của chúng trong một kiến trúc mang tính kết nối. Đây là một viễn cảnh trong đó mọi vật, mọi con vật hoặc con người được cung cấp các định danh và khả năng tự động truyền tải dữ liệu qua một mạng lưới mà không cần sự tương tác giữa con người-với-con người hoặc con người-với-máy tính. IoT tiến hoá từ sự hội tụ của các công nghệ không dây, hệ thống vi cơ điện tử (MEMS) và Internet. Cụm từ này được đưa ra bởi Kevin Ashton vào năm 1999. Ông là một nhà khoa học đã sáng lập ra Trung tâm Auto-ID ở đại học MIT [7].

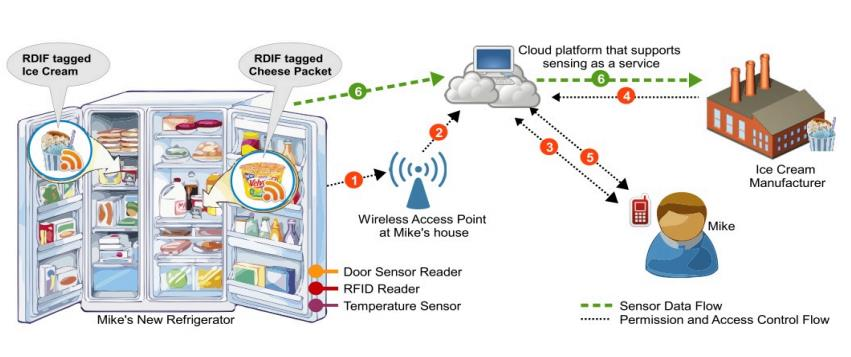


Hình 4 Internet of Things

Thuật ngữ IoT hay Internet vạn vật đề cập đến mạng lưới tập hợp các thiết bị thông minh và công nghệ tạo điều kiện thuận lợi cho hoạt động giao tiếp giữa thiết bị và đám mây cũng như giữa các thiết bị với nhau. Nhờ sự ra đời của chip máy tính giá rẻ và công nghệ viễn thông băng thông cao, ngày nay, chúng ta có hàng tỷ thiết bị được kết nối với internet. Điều này nghĩa là các thiết bị hàng ngày như bàn chải đánh răng, máy hút bụi, ô tô và máy móc có thể sử dụng cảm biến để thu thập dữ liệu và phản hồi lại người dùng một cách thông minh.

Internet vạn vật tích hợp “vạn vật” với Internet mỗi ngày. Các kỹ sư máy tính đã và đang thêm các cảm biến và bộ xử lý vào các vật dụng hàng ngày kể từ những năm 90. Tuy nhiên, tiến độ ban đầu rất chậm vì các con chip còn to và cồng kềnh. Loại chip máy tính công suất thấp gọi là thẻ tag RFID, lần đầu tiên được sử dụng để theo dõi các thiết bị đắt đỏ. Khi kích cỡ của các thiết bị điện toán dần nhỏ lại, những con chip này cũng trở nên nhỏ hơn, nhanh hơn và thông minh hơn theo thời gian.

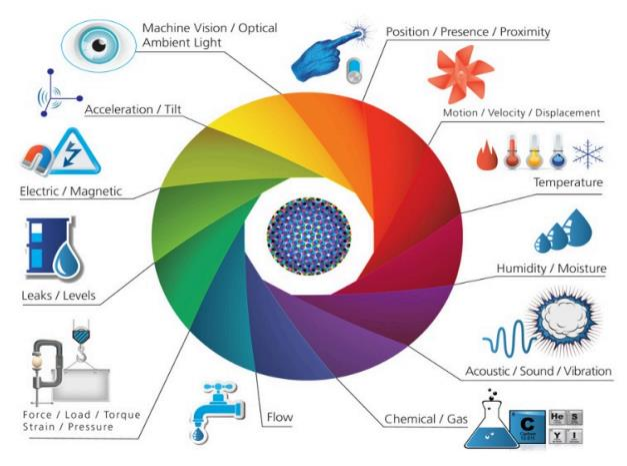
Một ví dụ điển hình cho IoT là tủ lạnh thông minh, nó có thể là một chiếc tủ lạnh bình thường nhưng có gắn thêm các cảm biến bên trong giúp kiểm tra được số lượng các loại thực phẩm có trong tủ lạnh, cảm biến nhiệt độ, cảm biến phát hiện mở cửa,...và các thông tin này được đưa lên internet. Với một danh mục thực phẩm được thiết lập trước bởi người dùng, khi mà một trong các loại thực phẩm đó sắp hết thì nó sẽ thông báo ngay cho chủ nhân nó biết rằng cần phải bổ sung gấp, thậm chí nếu các loại sản phẩm được gắn mã ID thì nó sẽ tự động trực tiếp gửi thông báo cần nhập hàng đến siêu thị và nhân viên siêu thị sẽ gửi loại thực phẩm đó đến tận nhà.



Hình 5 Ứng dụng của IoT trong tủ lạnh

### ***3.1.2. Cơ sở kỹ thuật của IoT***

* Giao thức chính: Trong IoT, các thiết bị phải giao tiếp được với nhau (D2D). Dữ liệu sau đó phải được thu thập và gửi tới máy chủ (D2S). Máy chủ cũng có để chia sẻ dữ liệu với nhau (S2S), có thể cung cấp lại cho các thiết bị, để phân tích các chương trình, hoặc cho người dùng. Các giao thức có thể dùng trong IoT là: Message Queue Telemetry Transport (MQTT), Extensible Messaging và Presence Protocol (XMPP), Data Distribution Service (DDS), Advanced Message Queue Protocol (AMQP).
* Năng lực truyền thông (Communication Capabilities): Địa chỉ IP được coi là yếu tố quan trọng trong IoT, khi mà mỗi thiết bị được gán một địa chỉ IP riêng biệt. Do đó khả năng cấp phát địa chỉ IP sẽ quyết định đến tương lai của IoT, hiện nay sự ra đời và gia tăng mạnh mẽ của IPv6 trong không gian địa chỉ là một yếu tố quan trọng trong phát triển Internet of Things.
* Công suất thiết bị (Device Power): Các tiêu chí hình thức chính của thiết bị khi triển khai một ứng dụng IoT là phải giá thành thấp, mỏng, nhẹ... do đó nó đòi hỏi thiết bị phải tiêu tốn một công suất cực nhỏ (Ultra Low Power) và giao thức truyền thông không dây gọn nhẹ, đơn giản, đòi hỏi ít năng lượng (Low Energy Wireless Technologies) như Zigbee, BLE (Bluetooth low energy), ANT/ANT+, NIKE+ …
* Công nghệ cảm biến (Sensor Technology): Trong IoT, cảm biến đóng vai trò then chốt, nó đo đạt cảm nhận giá trị từ môi trường xung quanh rồi gửi đến bộ vi xử lý sau đó được gửi lên mạng. Chúng ta có thể bắt gặp một số loại cảm biến về cảnh báo cháy rừng, cảnh báo động đất, cảm biến nhiệt độ, độ ẩm,… Hiện nay với sự phát triển nhanh của các công nghệ 4.0 đòi hỏi các cảm biến phải tiêu hao một lượng năng lượng cực kỳ thấp, có độ chính xác cao và thời gian đáp ứng phải nhanh và cũng đòi hỏi giá cảm biến cũng phải thấp.



Hình 6 Một số loại cảm biến hay gặp

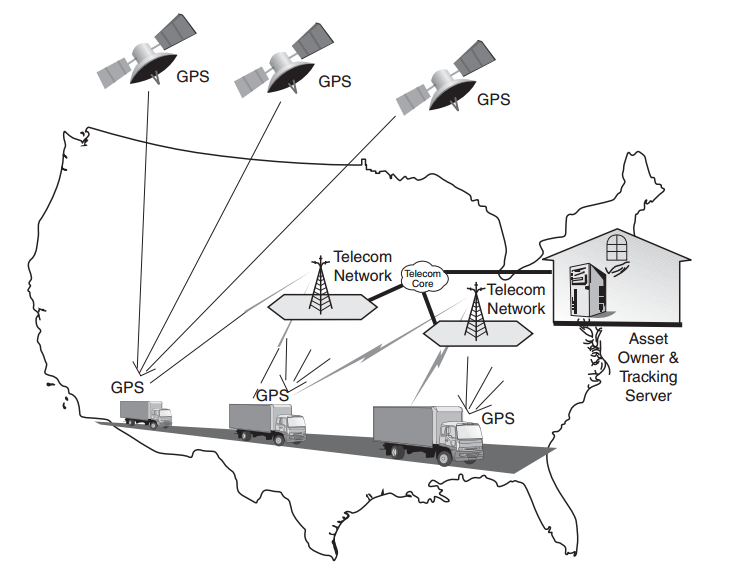
### ***3.1.3. Các ứng dụng của IoT trong cuộc sống***

Với những hiệu quả thông minh rất thiết thực mà IoT đem đến cho con người, IoT đã và đang được tích hợp trên khắp mọi thứ, mọi nơi xung quanh thế giới mà con người đang sống. Từ chiếc vòng đeo tay, những đồ gia dụng trong nhà, những mãnh vườn đang ươm hạt giống, cho đến những sinh vật sống như động vật hay con người...đều có sử dụng giải pháp IoT.



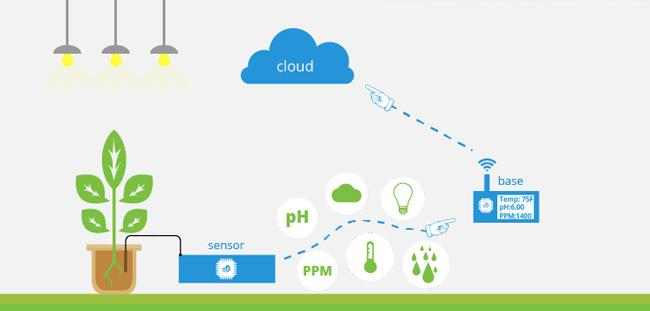
Hình 7 Tổng quan về ứng dụng của IoT

* Ứng dụng trong lĩnh vực vận tải: Ứng dụng điển hình nhất trong lĩnh vực này là gắn chíp lấy tọa độ GPS lên xe chở hàng, nhằm kiểm soát lộ trình, tốc độ, thời gian đi đến của các xe chở hàng. Ứng dụng này giúp quản lý tốt khâu vận chuyển, có những xử lý kịp thời khi xe đi không đúng lộ trình hoạt bị hỏng hóc trên những lộ trình mà ở đó mạng di động không phủ sóng tới được.



Hình 8 Theo dõi lộ trình đi của xe chở hàng

* Ứng dụng trong lĩnh vực sản xuất nông nghiệp: Quá trình sinh trưởng và phát triển của cây trồng phải trải qua nhiều giai đoạn. Ở mỗi giai đoạn cần có sự chăm sóc khác nhau về chất dinh dưỡng cũng như chế độ tưới tiêu phù hợp. Nhờ vào ứng dụng khoa học kỹ thuật, sử dụng cảm biến để lấy thông số nhiệt độ, độ ẩm, độ pH của đất trồng, cùng với bảng dữ liệu về quy trình sinh trưởng của loại cây đó, hệ thống sẽ tự động tưới tiêu bón lót cho cây trồng phù hợp với từng giai đoạn phát triển của cây trồng.



Hình 9 Theo dõi tình trạng sinh trưởng của cây trồng

* Ứng dụng trong nhà thông minh: Vài năm trở lại đây, khi thế giới đang dần tiến vào kỷ nguyên Internet of Things, kết nối mọi vật qua Internet, nhà thông minh trở thành một xu hướng công nghệ tất yếu, là tiêu chuẩn của nhà ở hiện đại. Trong căn hộ thông minh, tất cả các thiết bị từ rèm cửa, điều hoà, dàn âm thanh, hệ thống ánh sáng, hệ thống an ninh, thiết bị nhà tắm... được kết nối với nhau và hoạt động hoàn toàn tự động theo kịch bản lập trình sẵn, đáp ứng đúng nhu cầu sử dụng của khách hàng.



Hình 10 Ví dụ về nhà thông minh

## Tổng quan về dịch vụ IoT trong Microsoft Azure

Microsoft Azure, thường được gọi là Azure, là một nền tảng điện toán đám mây (cloud computing) được phát triển bởi Microsoft. Ra mắt lần đầu tiên vào năm 2010, Azure hiện là một trong những nền tảng đám mây hàng đầu trên thế giới, cung cấp hơn 200 dịch vụ cho phép người dùng phát triển, triển khai, và quản lý ứng dụng trên một mạng lưới toàn cầu.

Azure hỗ trợ nhiều mô hình triển khai, bao gồm IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), và SaaS (Software-as-a-Service), đồng thời cung cấp tích hợp mạnh mẽ với các công nghệ nguồn mở, hệ sinh thái Microsoft, và các giải pháp bên thứ ba.

Microsoft Azure IoT bao gồm Azure IoT Hub: kết nối và quản lý hàng triệu thiết bị IoT và Azure IoT Central: nền tảng SaaS để triển khai và quản lý các giải pháp IoT.

### ***3.2.1. Azure IoT Hub***

### IMG_256

Hình 11 Azure IoT Hub

- Azure IoT Hub là một dịch vụ thuộc nền tảng đám mây Microsoft Azure, được thiết kế để quản lý, giám sát và giao tiếp với hàng triệu thiết bị IoT (Internet of Things). Với khả năng cung cấp kết nối an toàn, giao tiếp hai chiều và tích hợp mạnh mẽ với các dịch vụ Azure khác, IoT Hub giúp doanh nghiệp dễ dàng triển khai các giải pháp IoT từ giai đoạn ý tưởng đến triển khai thực tế.

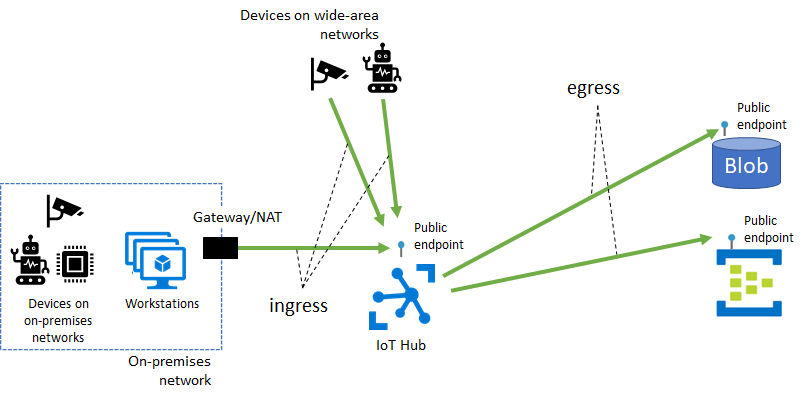
- Azure IoT Hub đóng vai trò là trung tâm kết nối và điều phối trong hệ sinh thái IoT:

* Kết nối các thiết bị IoT: IoT Hub cho phép thiết bị gửi dữ liệu cảm biến lên cloud và nhận lệnh điều khiển từ xa từ cloud.
* Quản lý thiết bị: Quản lý trạng thái, cấu hình, và cập nhật firmware từ xa cho hàng triệu thiết bị.
* Xử lý dữ liệu: Thu thập và định tuyến dữ liệu đến các công cụ phân tích, xử lý hoặc lưu trữ trên đám mây Azure.
* Bảo mật hệ thống IoT: IoT Hub tích hợp nhiều lớp bảo mật từ thiết bị đến cloud, bảo vệ dữ liệu và danh tính thiết bị khỏi các cuộc tấn công mạng.

- Một số đặc điểm nổi bật của Azure IoT Hub:

* Azure IoT Hub hỗ trợ giao tiếp cả hai chiều giữa thiết bị và nền tảng cloud: Các thiết bị gửi dữ liệu đến cloud Uplink có thể gửi dữ liệu liên tục hoặc theo yêu cầu đến IoT Hub để phân tích hoặc lưu trữ. Cloud (Downlink) có thể gửi lệnh điều khiển như bật/tắt thiết bị, thay đổi cấu hình, hoặc thực hiện cập nhật firmware.
* Azure IoT Hub cung cấp các công cụ mạnh mẽ để quản lý một lượng lớn thiết bị IoT: Lưu trữ trạng thái của từng thiết bị, hỗ trợ đồng bộ cấu hình giữa thiết bị và cloud, cập nhật Firmware từ xa dễ dàng mà không cần truy cập trực tiếp vào thiết bị, ghi nhận và phân tích trạng thái hoạt động của thiết bị (logs), giúp phát hiện và xử lý lỗi kịp thời.
* IoT Hub cung cấp tính năng định tuyến dữ liệu, cho phép bạn chuyển tiếp dữ liệu từ thiết bị đến các dịch vụ Azure hoặc ứng dụng tùy chỉnh.
* IoT Hub được xây dựng với lớp bảo mật nhiều cấp độ: Xác thực thiết bị (SAS), mã hóa dữ liệu, quản lý quyền truy cập.
* Hỗ trợ đa giao thức và kết hợp mạnh mẽ với nhiều dịch vụ trong hệ sinh thái Azure như: Azure Functions, Azure Machine Learning, Power BI …

- Kiến trúc của Azure IoT Hub:



Hình 12 Kiến trúc Azure IoT Hub

IoT Hub hoạt động như một cầu nối giữa thiết bị và nền tảng đám mây Azure, bao gồm:

* Thiết bị IoT (IoT Devices): Các cảm biến hoặc thiết bị thông minh thu thập dữ liệu từ môi trường thực.
* IoT Gateway: Làm nhiệm vụ trung gian, kết nối thiết bị không hỗ trợ internet với IoT Hub.
* Azure IoT Hub: Xử lý dữ liệu gửi đến từ các thiết bị và định tuyến dữ liệu đến các dịch vụ Azure như Azure Blob Storage, Azure Stream Analytics, hoặc Azure Functions.
* Ứng dụng của doanh nghiệp/người dùng: Các ứng dụng dùng để giám sát, phân tích, và đưa ra hành động dựa trên dữ liệu thu thập.

- Azure IoT Hub cũng như nhiều dịch vụ IoT khác mang lại nhiều lợi ích trong các lĩnh vực như: Nông nghiệp thông minh, quản lý nhà thông minh, sản xuất nông nghiệp hay quản lý chuỗi cung ứng …

- Azure IoT Hub có nhiều đặc điểm nổi trội là thế nhưng cũng có nhiều nhược điểm như:

* Chi phí có thể trở nên đắt đỏ khi số lượng thiết bị kết nối trà truyền tải dữ liệu tăng cao.
* Yêu cầu kiến thức chuyên sâu về hệ sinh thái Azure và IoT để thiết lập và tích hợp một cách hiệu quả. Việc cấu hình bảo mật và giao thức có thể phức tạp đối với người mới bắt đầu.
* Phụ thuộc vào kết nối internet: Azure IoT Hub yêu cầu kết nối internet liên tục để hoạt động, dẫn đến nguy cơ gián đoạn nếu hạ tầng mạng không ổn định.
* Mặc dù hỗ trợ đa nền tảng, Azure IoT Hub hoạt động hiệu quả nhất khi tích hợp với các dịch vụ của Azure. Điều này có thể là hạn chế nếu doanh nghiệp sử dụng hệ sinh thái đám mây khác như AWS hoặc Google Cloud.
* Rủi ro vendor lock-in: Sử dụng Azure IoT Hub có thể dẫn đến phụ thuộc vào hệ sinh thái Microsoft, gây khó khăn nếu muốn chuyển đổi sang nền tảng khác trong tương lai.

### ***3.2.2. Azure IoT Central***

### IMG_256

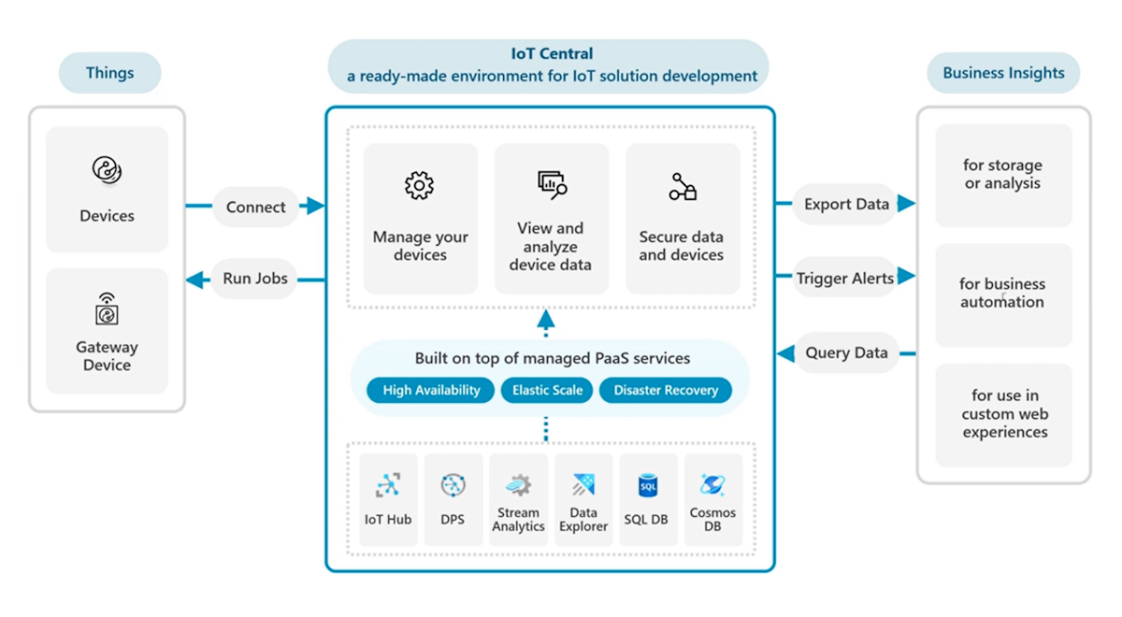
Hình 13 Azure IoT Central

- Azure IoT Central là một nền tảng quản lý IoT dạng SaaS (Software-as-a-Service) của Microsoft, được thiết kế để đơn giản hóa việc xây dựng, triển khai và quản lý các giải pháp IoT quy mô lớn. Đây là một công cụ mạnh mẽ dành cho doanh nghiệp muốn nhanh chóng triển khai ứng dụng IoT mà không cần phải xây dựng hệ thống từ đầu.

- Các đặc điểm chính của Azure IoT Central:

* Dễ dàng triển khai: Azure IoT Central được thiết kế để giảm thiểu sự phức tạp khi triển khai IoT với giao diện người dùng trực quan, các template đã được tích hợp sẵn giúp triển khai nhanh các ứng dụng.
* Azure IoT Central cung cấp các công cụ mạnh mẽ để quản lý thiết bị với dashboard tùy chỉnh hiển thị dữ liệu qua biểu đồ, bảng, hoặc bản đồ.
* Hỗ trợ các giao thức chuẩn như MQTT, HTTPS, AMQP, cho phép theo dõi trạng thái thiết bị, hỗ trợ cập nhật firmware, thay đổi cấu hình mà không cần truy cập trực tiếp.
* Azure IoT Central dễ dàng tích hợp với các dịch vụ Azure khác để mở rộng khả năng: Azure Time Series Insights, Azure Machine Learning, Microsoft Power BI …
* Mô hình SaaS tối ưu hóa chi phí: Thanh toán theo sử dụng, khách hàng chỉ cần trả phí dựa trên số lượng thiết bị và lượng dữ liệu sử dụng, giúp tối ưu chi phí cho cả doanh nghiệp nhỏ lẫn các tập đoàn lớn. Không cần quản lý cơ sở hạ tầng, Azure IoT Central tự động xử lý các yếu tố hạ tầng như máy chủ, cơ sở dữ liệu, và bảo mật, giúp giảm bớt gánh nặng quản trị.

- Các thành phần chính của Azure IoT Central:



Hình 14 Kiến trúc Azure IoT Central

* Lớp thiết bị IoT (IoT Devices Layer): Lớp này bao gồm các thiết bị IoT như các cảm biến, bộ truyền động, thiết bị thông minh gửi dữ liệu về Azure IoT Central dùng các giao thức phổ biến là MQTT, AMQP, HTTPS.
* Azure IoT Hub: Đây là lớp trung gian đóng vai trò cổng giao tiếp giữa thiết bị IoT và Azure IoT Central, nhận và xử lý dữ liệu từ thiết bị, quản lý kết nối hai chiều và mô tả hiển thị các trạng thái của thiết bị (Device Twin) - metadata, cấu hình, trạng thái kết nối.
* Lớp dữ liệu (Data Processing Layer): Lớp này quản lý trực tiếp các sự kiện, thông báo được gửi đến người dùng qua email, SMS. Sử dụng các dịch vụ lưu trữ như Azure Blob Storage, Azure Data Lake để lưu trữ dữ liệu từ thiết bị và xử lý dữ liệu bằng Azure Stream Analytics hoặc Azure Functions trước khi chuyển đến dashboard.
* Lớp ứng dụng (Application Layer): Cung cấp một giao diện quản lý trực quan trên nền web để cấu hình, quản lý và giám sát thiết bị với dashboard tùy chỉnh. Hiển thị dữ liệu thời gian thực từ thiết bị với biểu đồ, bảng số liệu dễ hiểu. Hỗ trợ API RESTful để kết nối với các ứng dụng khác như CRM, ERP, hoặc hệ thống phân tích dữ liệu.

## Thiết kế hệ thống

## *3.3.1. Mô hình kiến trúc tổng thể*

1. Lớp thiết bị (Device Layer):

* Thiết bị IoT: ESP8266 là vi điều khiển với khả năng kết nối Wi-Fi, được tích hợp cảm biến DHT11 để thu thập dữ liệu nhiệt độ và độ ẩm.
* Nhiệm vụ: Thu thập dữ liệu từ cảm biến và truyền đến Gateway hoặc trực tiếp lên đám mây.

1. Lớp Gateway (Gateway Layer):

* Vai trò: Nếu cần quản lý nhiều thiết bị, gateway sẽ xử lý tập trung dữ liệu trước khi truyền lên đám mây. Trong bài này, ESP8266 có thể truyền trực tiếp dữ liệu lên Azure IoT Central thông qua MQTT.
* Lựa chọn Gateway: Không cần gateway riêng biệt trong thiết kế này.

1. **Lớp đám mây (Cloud Layer)**:

* Azure IoT Central: Cung cấp nền tảng quản lý thiết bị, xử lý và lưu trữ dữ liệu từ các thiết bị IoT. Dữ liệu được truyền qua giao thức MQTT, HTTPS hoặc AMQP.
* Các module đám mây:
* Ingestion: Tiếp nhận dữ liệu từ thiết bị
* Storage: Lưu trữ dữ liệu trên Azure Blob Storage hoặc Azure Table Storage. Processing: Phân tích dữ liệu với Azure Functions hoặc Azure Stream Analytics.
* Visualization: Trình bày dữ liệu trên giao diện Azure IoT Central.
* Hệ thống ghi nhận và hiển thị các đánh giá công khai để hỗ trợ người dùng khác.

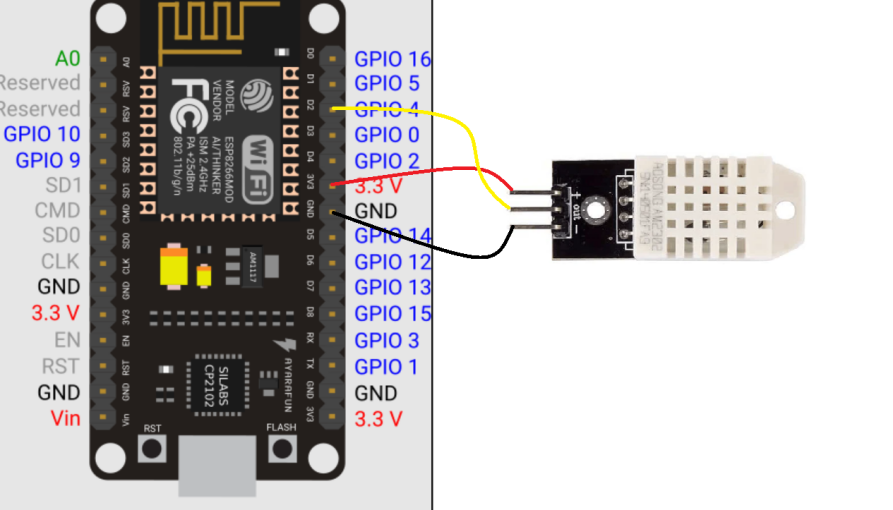
1. **Lớp ứng dụng (Application Layer)**: Azure IoT Central hỗ trợ dashboard hiển thị nhiệt độ và độ ẩm thời gian thực. Người dùng có thể theo dõi dữ liệu, cài đặt cảnh báo, hoặc tải báo cáo.

## *3.3.2. Liên kết giữa các thành phần*

- Kết nối giữa thiết bị ESP8266 và cảm biến DHT11:

* Cách thức giao tiếp: Cảm biến DHT11 giao tiếp với ESP8266 qua giao tiếp 1-wire (Single-Wire Protocol). Cảm biến gửi dữ liệu nhiệt độ và độ ẩm dạng kỹ thuật số qua chân DATA.

Kết nối phần cứng: DATA: Chân dữ liệu của DHT11 được nối vào một GPIO trên ESP8266 (ví dụ: D4 trên NodeMCU). VCC: Cấp nguồn cho cảm biến (3.3V hoặc 5V từ ESP8266). GND: Nối đất chung với ESP8266.



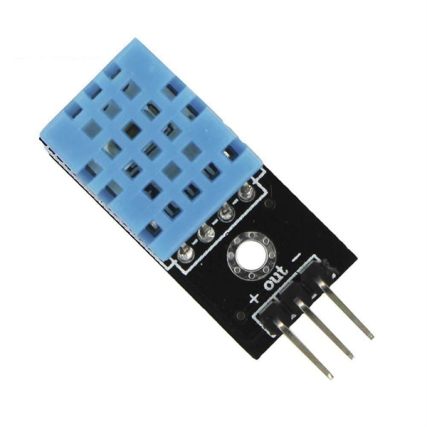
Hình 15 Kết nối phần cứng giữa ESP8266 và DHT11

- Kết nối giữa ESP8266 và Azure IoT Central:

* Kết nối qua mạng Wi-Fi: ESP8266 kết nối Wi-Fi thông qua SSID và mật khẩu do người dùng cấu hình. ESP8266 đóng vai trò client gửi dữ liệu đến Azure IoT Central qua giao thức MQTT.
* Giao thức truyền dữ liệu: MQTT (Message Queuing Telemetry Transport). Dữ liệu từ ESP8266 được truyền dưới dạng JSON (key-value) thông qua MQTT Publish. Azure IoT Hub đóng vai trò là MQTT Broker.

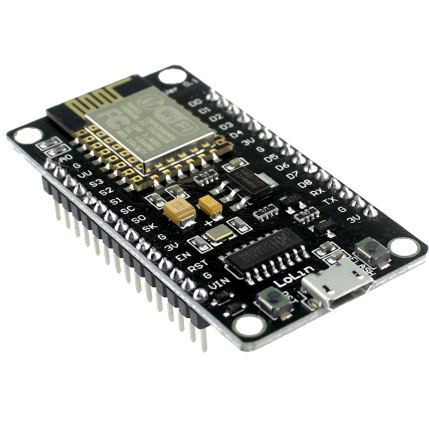
### ***3.3.3. Lựa chọn và cấu hình thiết bị***

- Cảm biến DHT11/DHT22



Hình 16 Cảm biến DHT11

- ESP8266 (NodeMCU hoặc tương tự) với các cấu hình: kết nối Wi-Fi, sử dụng thư viện MQTT (ví dụ: PubSubClient) để gửi dữ liệu và gửi dữ liệu định kỳ hoặc khi có thay đổi.



Hình 17 Mạch ESP8266

- Azure IoT Central: Tạo một IoT Application trên Azure IoT Central, IoT Device Template để định nghĩa cấu trúc dữ liệu nhiệt độ và độ ẩm và cấu hình các thông tin kết nối MQTT (Device Connection String) trong ESP8266.

### ***3.3.4. Xây dựng các module xử lý dữ liệu trên đám mây***

- Module nhận dữ liệu: Sử dụng Azure IoT Hub (phần của Azure IoT Central) để nhận dữ liệu từ thiết bị.

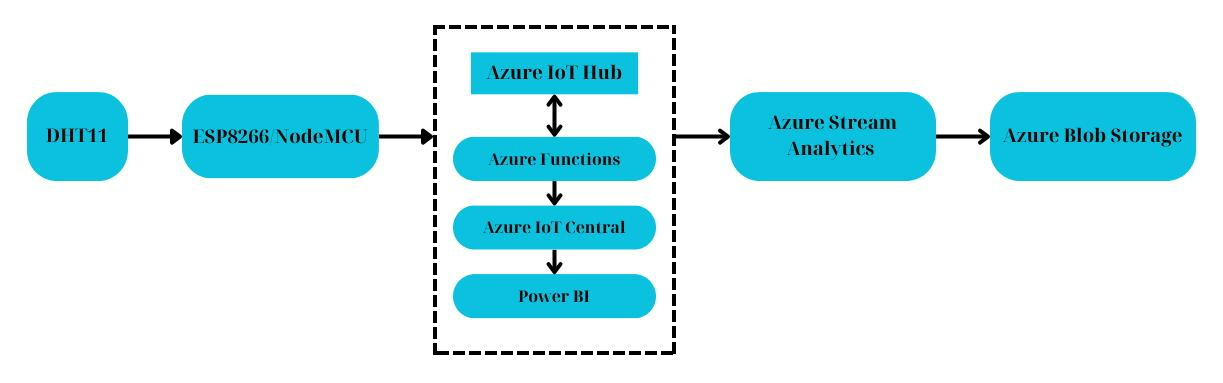
- Module xử lý:

* Azure Stream Analytics: Phân tích luồng dữ liệu để phát hiện bất thường (ví dụ: nhiệt độ vượt ngưỡng).
* Azure Functions: Thực hiện xử lý tùy chỉnh, gửi cảnh báo qua email hoặc ứng dụng khác.

- Module lưu trữ: Dữ liệu thô được lưu trữ trên Azure Blob Storage. Dữ liệu đã xử lý được lưu trong Azure Cosmos DB hoặc SQL Database.

- Module hiển thị: Sử dụng các widget trong Azure IoT Central để hiển thị dữ liệu.

### ***3.3.5. Lưu đồ hoạt động (Flow Diagram)***



Hình 18 Lưu đồ hoạt động

- Thiết bl IoT (ESP8266, DHT11): Cảm biến đo nhiệt độ và độ ẩm gửi dữ liệu (ở dạng JSON) lên Azure IoT Hub.

- Azure IoT Hub: Nhận dữ liệu từ thiết bị IoT, xác thực và chuyển tiếp dữ liệu đến các dịch vụ xử lý (Stream Analytics hoặc Event Hub).

- Azure Stream Analytics: Dữ liệu từ IoT Hub được đưa vào Azure Stream Analytics để phân tích, ví dụ: kiểm tra xem nhiệt độ có vượt ngưỡng cho phép không.

- Azure Functions: Nếu có sự kiện bất thường (như nhiệt độ quá cao), Azure Functions có thể kích hoạt hành động tự động, ví dụ: gửi cảnh báo qua email hoặc SMS.

- Lưu trữ dữ liệu: Dữ liệu đã được xử lý có thể được lưu trữ trong Azure Blob Storage hoặc Azure Cosmos DB để truy xuất sau này.

- Azure IoT Central: Dữ liệu từ thiết bị được hiển thị trên Azure IoT Central, nơi bạn có thể theo dõi nhiệt độ và độ ẩm trong thời gian thực thông qua các dashboard trực quan.

- Power BI: Để phân tích dữ liệu lâu dài và tạo báo cáo chi tiết, bạn có thể sử dụng Power BI để kết nối với cơ sở dữ liệu và tạo các biểu đồ phân tích.

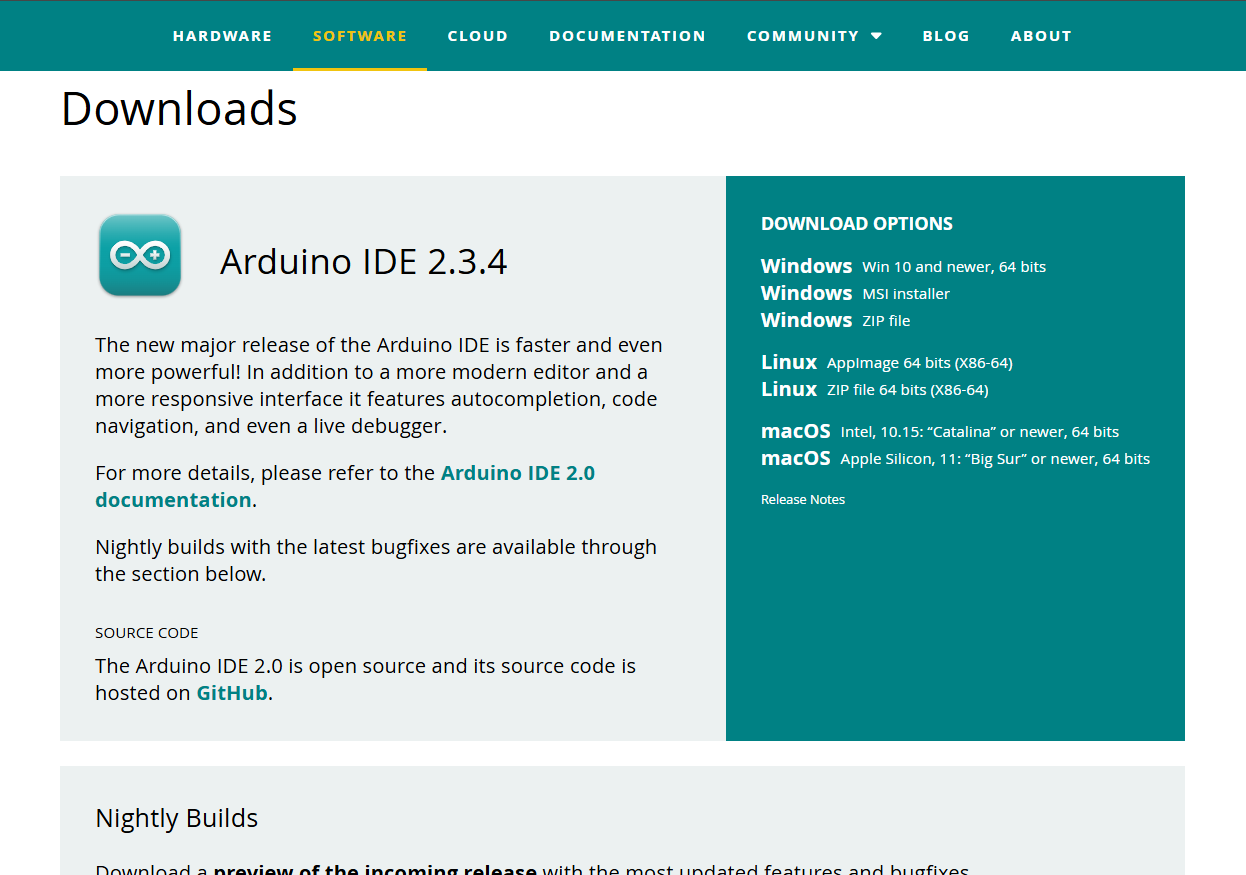
## Cài đặt và cấu hình hệ thống

## *3.4.1. Cài đặt*

***3.4.1.1. Cài đặt Arduino IDE***

* Bước 1: Truy cập trang web của Arduino và chọn Window MSI installer:

<https://www.arduino.cc/en/software>



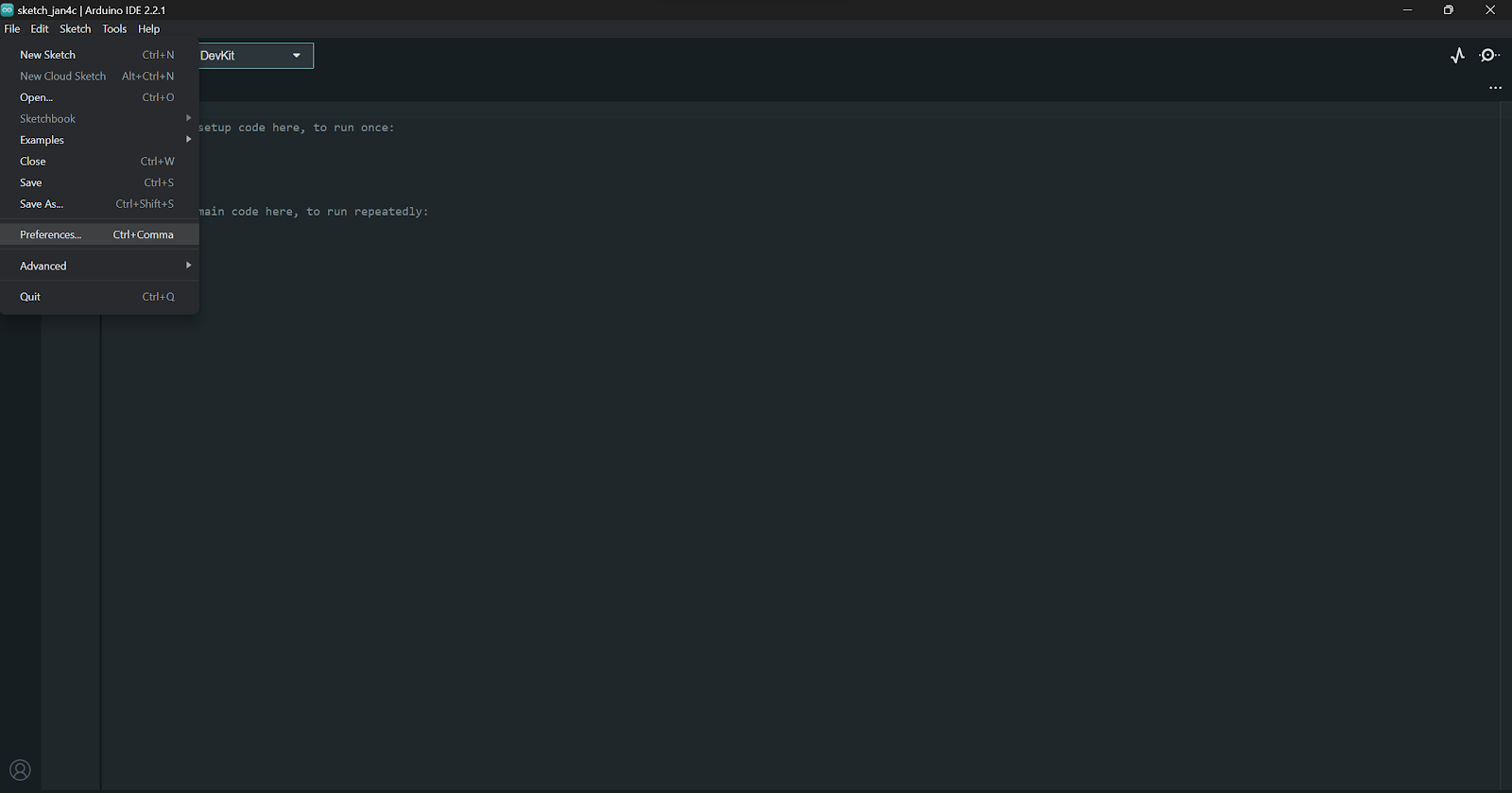
- Bước 2: Chạy file cài đặt vừa tải về



* Bước 3: Đợi chương trình cài đặt xong.

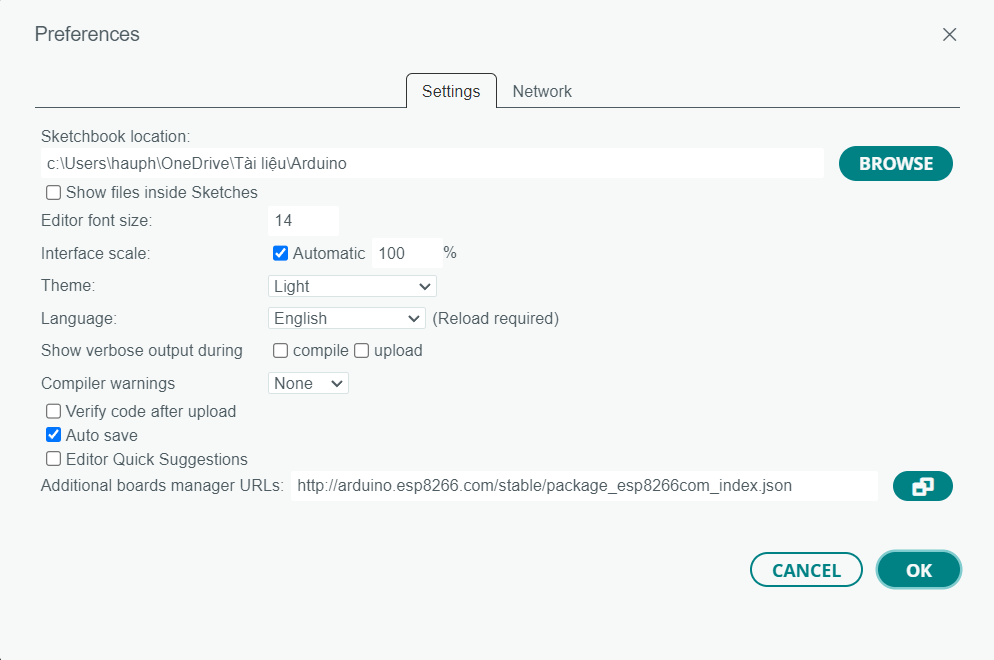
***3.4.1.2. Cài đặt các thư viện***

- Bước 1: Mở Arduino IDE và chọn “file” -> “Preferences”

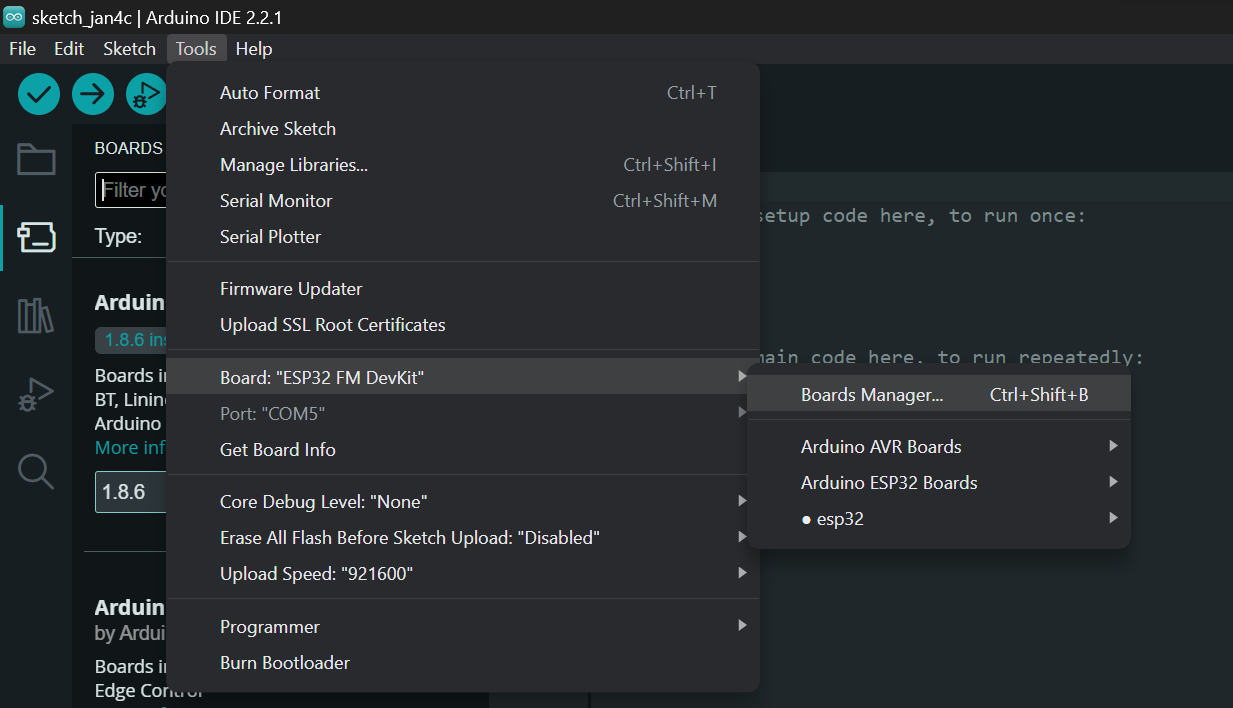


- Bước 2: Trong ô "Additional Boards Manager URLs," thêm đường dẫn

<http://arduino.esp8266.com/stable/package_esp8266com_index.json>



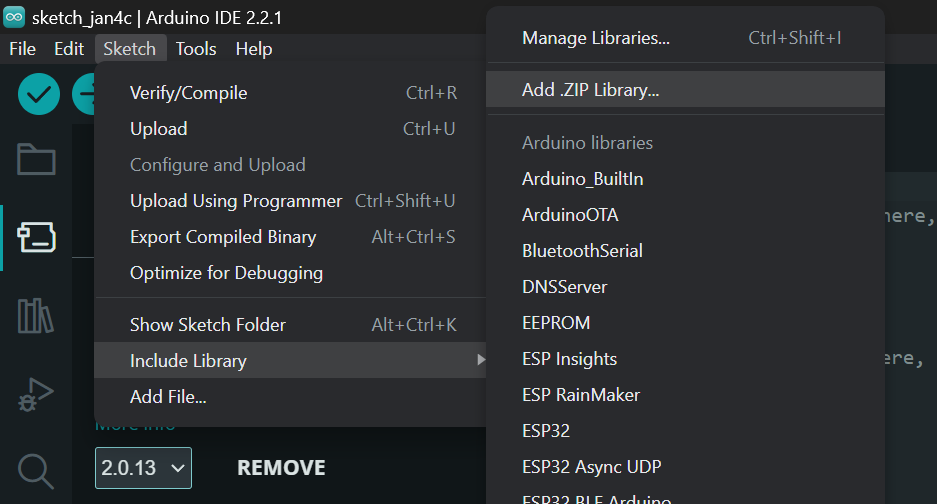
- Bước 3: Đóng hộp thoại và truy cập "Tools" -> "Board" -> "Boards Manager."



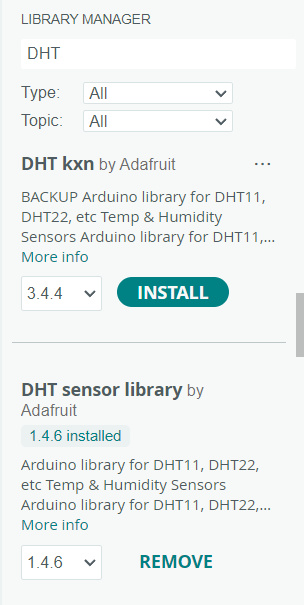
- Bước 4: Tìm “esp8266” và cài đặt “esp8266” bởi Espressif Systems.



- Bước 5: Trong Arduino IDE truy cập “Sketch” -> “Include Library” -> “Manage Libraries”.



- Bước 6: Tìm kiếm DHT và cài đặt “DHT sensor library by Adafruit”



## *3.4.2. Cấu hình Azure*

## - Bước 1: Tạo một tài khoản Microsoft Azure

## <https://azure.microsoft.com/en-us/products/iot-hub>

## IMG_256

## - Bước 2: Vào All services và chọn Internet of Things

## 

## - Bước 3: Chọn IoT Central Applications và tạo 1 project mới

## IMG_256

## - Bước 4: Chọn Review + Create

## 

## - Bước 5: Truy cập vào đường dẫn được cung cấp ở IoT Central Application URL, ví dụ ở đây sẽ là: <https://iotdht11.azureiotcentral.com>

## 

## - Bước 6: Vào Device templates và chọn IoT device -> Next: Customize

## 

## - Bước 7: Sau khi tạo xong template mới ta chọn Custom model

## 

## - Bước 8: Tại đây ta sẽ chọn Add capability để thiết lập dữ liệu cho device như dưới đây

## 

## - Bước 9: Chọn New để tạo một device mới

## 

## - Bước 10: Vào device đã tạo chọn connect, tại đây ta sẽ lấy được các thông tin để kết nối với cloud

## 

## - Bước 11: Mở dự án với Arduino IDE và thay SCOPE\_ID, DEVICE\_ID, DEVICE\_KEY tương ứng với ID scope, Device ID và Primary key được cung cấp ở trên

## IMG_256

## - Bước 12: Nạp code vào mạch ESP8266

## IMG_256

## Kết quả và ứng dụng thực tiễn

## *3.5.1. Kết quả*

**-** Thu thập dữ liệu chính xác: Hệ thống sử dụng cảm biến DHT11 để đo nhiệt độ và độ ẩm môi trường với độ chính xác phù hợp cho các ứng dụng giám sát cơ bản.

- ESP8266 hoạt động như một thiết bị IoT, truyền dữ liệu đo đạc đến Azure IoT Hub qua giao thức MQTT hoặc HTTP.

- Azure IoT Hub nhận và lưu trữ dữ liệu, tích hợp với các dịch vụ như Azure Stream Analytics hoặc Power BI để hiển thị trực quan và phân tích dữ liệu.

- Cảnh báo thời gian thực: Thiết lập hệ thống gửi thông báo hoặc cảnh báo qua email hoặc SMS nếu nhiệt độ và độ ẩm vượt ngưỡng cho phép.

## *3.5.2**. Ứng dụng thực tiễn*

- Trong Nông nghiệp:

+ Hệ thống giám sát và duy trì nhiệt độ, độ ẩm tối ưu để đảm bảo cây trồng phát triển tốt nhất.

+ Kho lưu trữ nông sản: Đảm bảo điều kiện môi trường phù hợp để bảo quản nông sản, giảm tổn thất do nấm mốc hoặc hư hỏng.

- Trong Công nghiệp:

+Quản lý kho bãi: Giám sát nhiệt độ, độ ẩm trong kho chứa nguyên vật liệu hoặc sản phẩm để tránh hư hỏng.

+ Phòng máy chủ (server room): Duy trì môi trường ổn định cho các thiết bị CNTT, tránh hiện tượng quá nhiệt hoặc độ ẩm cao gây hỏng hóc.

- Trong Đời sống Hàng ngày: Tích hợp vào các hệ thống nhà thông minh.

**KẾT LUẬN VÀ KIẾN NGHỊ**

**Kết luận**

Dự án phát triển hệ thống giám sát nhiệt độ và độ ẩm sử dụng NodeMCU ESP8266 với Azure IoT Hub đã mang lại nhiều kết quả tích cực và đáng chú ý. Qua quá trình triển khai, nhóm nghiên cứu đã đạt được những thành tựu sau:

- Triển khai thành công: Hệ thống đã được triển khai một cách thành công, cho phép ESP8266 đọc dữ liệu từ cảm biến DHT11 và truyền thông tin lên Azure IoT Hub, nền tảng lưu trữ và hiển thị dữ liệu trực tuyến.

- Giám sát nhiệt độ và độ ẩm chính xác: Hệ thống đã chứng minh khả năng giám sát nhiệt độ và độ ẩm môi trường một cách chính xác, cung cấp thông tin chi tiết và quan trọng về điều kiện môi trường.

- Kết nối được Azure IoT Hub: Đã thành công trong việc kết nối và gửi dữ liệu lên Azure IoT Hub, mở ra khả năng theo dõi và quản lý dữ liệu từ xa thông qua giao diện web.

- Kiểm thử và đánh giá hệ thống: Hệ thống đã trải qua quá trình kiểm thử mô phỏng và thực tế, đảm bảo rằng nó hoạt động ổn định và đáng tin cậy trong nhiều điều kiện môi trường khác nhau.

Trong quá trình thực hiện dự án đã mang lại cho nhóm nghiên cứu những kiến thức sâu rộng và kinh nghiệm quý báu:

- Lập trình ESP8266: Nắm vững kỹ năng lập trình ESP8266 và sử dụng Azure IoT Hub để phát triển ứng dụng IoT.

- Giao thức Azure IoT Hub: Hiểu rõ về cách sử dụng Azure IoT Hub và các API của nó để lưu trữ và hiển thị dữ liệu một cách hiệu quả.

- Kiểm thử và đánh giá hệ thống: Phát triển kỹ năng kiểm thử hệ thống để đảm bảo tính chính xác và đáng tin cậy của dữ liệu.

- Quản lý dự án: Thu được kinh nghiệm quản lý dự án, từ việc lên kế hoạch đến triển khai và giám sát tiến độ.

**Kiến nghị**

Để dự án được triển khai hiệu quả hơn trong tương lai, cần lưu ý một số vấn đề sau:

- Độ chính xác của cảm biến: Cảm biến DHT11 có thể mang lại dữ liệu không chính xác trong môi trường đặc biệt khắc nghiệt, và việc nâng cấp cảm biến là một hướng phát triển tiếp theo.

- Phụ thuộc vào kết nối Internet: Tính phụ thuộc chặt chẽ vào kết nối Internet có thể tạo ra vấn đề trong trường hợp mất kết nối, và việc xem xét các giao thức như MQTT có thể giúp giảm độ trễ.

# **TÀI LIỆU THAM KHẢO**

[1].Tài liệu học tập môn học Một số công nghệ phát triển phần mềm.

[2]."Learning OpenCV 4 Computer Vision with Python" của Joseph Howse, Joe Minichino, Alexey Spizhevoy, 2020.

[3]."Computer Vision: Algorithms and Applications" của Richard Szeliski, 2020.

[4]."Python for Data Analysis" của Wes McKinney, 2017.

[5]."Real-Time Car Detection using Python and OpenCV" Itsourcecode.