

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công Nghệ Thông Tin và Truyền Thông



BÁO CÁO BÀI TẬP LỚN

IT6120 – AN TOÀN THÔNG TIN CHO IOT

Tìm hiểu mô hình Publish/Subscribe trong Apache Kafka
và ứng dụng xử lý dòng dữ liệu video theo thời gian thực

Giảng Viên: PGS.TS. Trần Quang Đức

Nhóm sinh viên thực hiện

Lê Văn Đồng

MSHV: CB202913M

Lê Viết Hoàng

MSHV: 20173128

Hà Nội, 2021

MỤC LỤC

BẢNG CHÚ GIẢI THUẬT NGỮ	4
DANH MỤC HÌNH VẼ VÀ ĐỒ THỊ	5
CHƯƠNG 1. CƠ SỞ LÝ THUYẾT	6
1.1. Hệ thống gửi/nhận thông điệp (message)	6
1.1.1. Cơ chế Message Queue	6
1.1.2. Cơ chế Publish/Subscribe Messaging	8
1.1.3. Đánh giá hệ thống gửi/nhận thông điệp	9
1.2. Apache Kafka.....	10
1.2.1. Kiến trúc thành phần Apache Kafka	11
1.2.1.1. Cấu trúc Apache Kafka.....	11
1.2.1.2. Partition	12
1.2.1.3. Cấu trúc dữ liệu log	14
1.2.1.4. Lưu trữ và xử lý dữ liệu	15
1.2.1.5. Consumer và Consumer Group	16
1.2.1.6. Vai trò của Zookeeper.....	18
1.2.2. Một số ứng dụng của Kafka	18
CHƯƠNG 2. ỨNG DỤNG KAFKA TRONG XỬ LÝ DÒNG DỮ LIỆU VIDEO THEO THỜI GIAN THỰC	21
2.1. Tích hợp kafka trong hệ thống xử lý dòng video theo thời gian thực	21
2.2. Triển khai và cài đặt thành phần hệ thống	23
2.2.1. Cài đặt máy chủ phân tích	23
2.2.2. Cài đặt máy chủ HLS	25
2.3. Kết quả đạt được	28
TÀI LIỆU THAM KHẢO	34

BẢNG CHÚ GIẢI THUẬT NGỮ

TCP	Transmission Control Protocol
RAM	Random Access Memory
AI	Artificial intelligence
ACL	Access Control List
IoT	Internet of Things

DANH MỤC HÌNH VẼ VÀ ĐỒ THỊ

Hình 1.1. Cơ chế Message Queue	7
Hình 1.2. Cơ chế Publish/Subscribe Messaging	8
Hình 1.3. Mô hình cấu trúc kafka đơn giản	11
Hình 1.4. Cấu trúc dữ liệu log trong kafka	15
Hình 1.5. Sơ đồ minh họa kết nối trong hệ thống thương mại điện tử	20
Hình 1.6. Ứng dụng Kafka trong xây dựng các hệ thống lớn.....	20
Hình 2.1. Kiến trúc hệ thống phân tích dòng video theo kiến trúc phân tán, với phần mềm quản lý tập trung.....	21
Hình 2.2. Tích hợp kafka trong hệ thống xử lý dòng video theo thời gian thực	22
Hình 2.3. Thông tin tài khoản xác thực dịch vụ.....	28
Hình 2.4. Danh sách các topic lắng nghe sự kiện của Kafka.....	29
Hình 2.5. Giao diện tạo mới một topic.....	30
Hình 2.6. Giao diện quản lý các camera	31
Hình 2.7. Xem dòng video trực tiếp từ camera.....	31
Hình 2.8. Các thông điệp từ kafka thông báo có người đi vào vùng cấm	32

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

Nội dung chính của phần này sẽ giới thiệu cơ chế Publish/Subscribe và ứng dụng của nó trong hệ thống Kafka – nền tảng xử lý dòng dữ liệu (stream) theo thời gian thực.

1.1. Hệ thống gửi/nhận thông điệp (message)

1.1.1. Cơ chế Message Queue

Với sự bùng nổ lớn cả về số lượng và sự đa dạng của các ứng dụng với các thao tác thu thập hoặc xử lý dữ liệu, việc triển khai các connections truyền dữ liệu từ nơi nó được tạo ra đến nhiều nơi khác nhau để xử lý được coi như một phần trong quá trình phát triển ứng dụng. **Messaging** là một công nghệ quan trọng dùng để kết nối dữ liệu trong môi trường này.

Messaging thực hiện việc truyền các bản ghi dữ liệu (records) từ hệ thống này sang hệ thống khác thông qua một hệ thống trung gian. Ngược lại với các kết nối trực tiếp, nơi người gửi biết được người nhận và kết nối trực tiếp đến từng người nhận, các giải pháp messaging sẽ tách rời việc gửi dữ liệu khỏi việc xử lý dữ liệu. Người gửi không cần biết người nhận nào sẽ thấy dữ liệu đó hoặc khi nào họ sẽ thấy dữ liệu đó.

Các giải pháp messaging đơn giản hóa việc phát triển ứng dụng bằng cách cung cấp cho nhà thiết kế hoặc người lập trình các thành phần hệ thống đã được chuẩn hóa, có thể tái sử dụng để xử lý tốt luồng dữ liệu. Từ đó, người lập trình có thể tập trung vào phân lỗi logic trong ứng dụng của họ. Ngoài định tuyến dữ liệu, messaging systems cũng có thể cung cấp các tính năng quan trọng như khả năng chịu lỗi, ghi nhật ký (logging), cho phép xử lý phân tán để cải thiện khả năng quản lý và độ tin cậy.

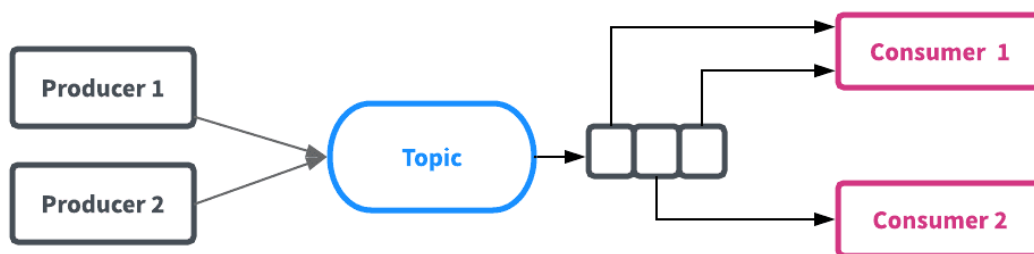
Messaging là một thuật ngữ bao gồm một số mô hình khác nhau dựa trên cách dữ liệu được chuyển từ người gửi sang người nhận. Hai mô hình message chính hiện

nay là **Message Queue** và **Publish / Subscribe Messaging** hay còn được gọi là cơ chế pub/sub messaging.

❖ Về Message Queue

Message Queue được thiết kế cho các tình huống như **tasklist** hoặc **workqueue**, trong đó nó nhận được message đến và đảm bảo rằng mỗi message sẽ được gửi cho một **topic** hay **channel** và xử lý bởi chính xác một **consumer**.

Ví dụ: Kịch bản Message Queue có thể được minh họa qua việc phát hành phiếu lương. Trong đó mọi phiếu lương đều phải được phát hành một lần và chỉ một lần duy nhất, mà việc xử lý ở máy kiểm tra nào không quan trọng.



Hình 1.1. Cơ chế Message Queue

Message Queue có thể nâng cao khả năng xử lý thông điệp lên cao bằng cách thêm nhiều consumers cho mỗi topic, nhưng chỉ duy nhất một consumer sẽ nhận được mỗi message ở topic này. Consumer nào nhận được message nào được xác định bằng cách implementation Message Queue. Để đảm bảo rằng một message chỉ được xử lý bởi một consumer, mỗi message sẽ bị xóa khỏi queue sau khi nó đã được một consumer nhận và xử lý.

Message Queue hỗ trợ các trường hợp (use-case) mà đảm bảo việc mỗi message được xử lý chính xác một và chỉ một lần duy nhất (exactly one), nhưng không cần thiết

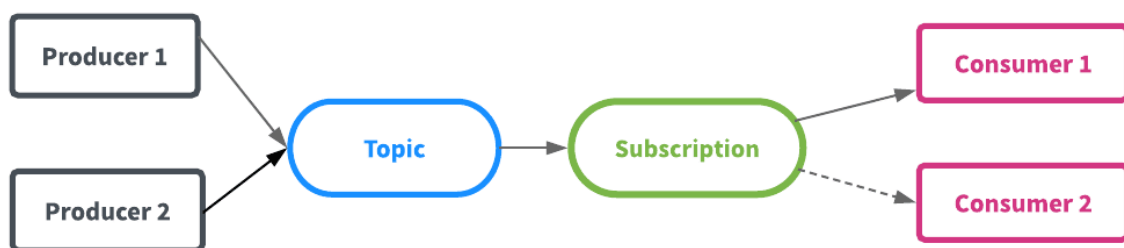
phải xử lý message theo thứ tự. Trong trường hợp lỗi từ mạng hoặc từ consumer, Message Queue sẽ thử gửi lại message (không nhất thiết phải cho cùng một consumer) và kết quả là message có thể được xử lý không theo thứ tự.

Một số đặc trưng của Message Queue

- Exactly One
- Scale processing (sẽ không đảm bảo thứ tự)
- Nếu đảm bảo thứ tự (sẽ không xử lý song song nên sẽ không scale processing)

1.1.2. Cơ chế Publish/Subscribe Messaging

Giống như Message Queue, cơ chế **Publish/Subscribe Messaging** cũng cho phép truyền message từ nơi gửi đến nơi sử dụng. Tuy nhiên, ngược với Message Queue, Publish/Subscribe Messaging cho phép nhiều consumer nhận từng message trong một topic. Hơn nữa, message pub-sub đảm bảo rằng mỗi consumer nhận được message trong một topic theo thứ tự chính xác mà message system nhận được. Các hệ thống Publish-Subscribe Messaging hỗ trợ các trường hợp yêu cầu nhiều consumers nhận từng message và các message đó được nhận theo thứ tự của mỗi consumer.



Hình 1.2. Cơ chế Publish/Subscribe Messaging

Ví dụ, một dịch vụ stock sticker có thể được sử dụng bởi một số lượng lớn người dùng và ứng dụng, tất cả người dùng đều muốn nhận toàn bộ message về topic họ

chọn. Điều quan trọng đối với những người dùng này là họ nhận được tin nhắn theo thứ tự mà việc nhìn thấy giá cao - giá thấp cho một cổ phiếu rất khác so với việc nhìn thấy giá thấp - giá cao.

Các trường hợp sử dụng pub-sub thường được liên kết với các ứng dụng trạng thái, cụ thể nó sẽ quan tâm đến thứ tự của các tin nhắn nhận được vì thứ tự của các tin nhắn xác định trạng thái của ứng dụng và do đó sẽ ảnh hưởng đến tính chính xác của bất kỳ logic xử lý nào mà ứng dụng có dùng đến.

Một số đặc trưng của Pub-sub messaging

- Multi-Subscriber
- Đảm bảo thứ tự các thông điệp

1.1.3. Đánh giá hệ thống gửi/nhận thông điệp

❖ Một số sản phẩm tiêu biểu

Cả hai mô hình Message Queue và Pub-sub messaging đều cần thiết trong các ứng dụng hiện đại, trong đó có một số công nghệ cho phép hỗ trợ cả hai cơ chế trên. Các công nghệ như **Apache ActiveMQ**, **Amazon SQS**, **IBM Websphere MQ**, **RabbitMQ** được phát triển chủ yếu cho các trường hợp sử dụng message queue, trong khi các hệ thống như **Amazon SNS**, **Apache Kafka**, **RocketMQ** và **Google Cloud Pub / Sub** được thiết kế chủ yếu cho các trường hợp sử dụng pub-sub. Ngoài ra còn có các giải pháp như **Apache Pulsar** cung cấp hỗ trợ cho cả xếp hàng tin nhắn và nhắn tin pub-sub.

Hiện nay, một số nền tảng công nghệ về messaging như RabbitMQ, Apache Kafka hay RocketMQ đều bổ sung thêm các tính năng có thể phục vụ của cả hai loại mô hình, tuy nhiên, về bản chất thì mỗi công nghệ chỉ có thể lựa chọn 1 trong 2 mô hình

để triển khai. Do đó, dù đảm bảo đầy đủ các chức năng nhưng sẽ luôn có những điểm riêng biệt của mỗi nền tảng công nghệ vừa nêu.

❖ **Đánh giá ưu điểm và nhược điểm**

- Điểm mạnh của Message Queue là nó cho phép phân chia việc xử lý dữ liệu trên nhiều consumers. Từ đó, có thể mở rộng khả năng xử lý dữ liệu.
- Điểm yếu của Message Queue là nó không cho phép multi-subscriber - nghĩa là chỉ có một subscriber duy nhất được xử lý một message sau đó message đó sẽ biến mất.
- Trong khi đó, điểm mạnh của Pub-sub messaging là cho phép broadcast dữ liệu đến các nơi xử lý dữ liệu.
- Nhưng ngược lại với Message Queue, điểm yếu của Pub-sub messaging là không có cách nào để scale việc xử lý dữ liệu bởi vì mỗi một tin nhắn đều được truyền đến mọi subscriber.

Từ đó, ta thấy rằng việc lựa chọn Message Queue hay Pub-sub messaging còn tùy vào mục đích sử dụng để tận dụng ưu điểm cũng như hạn chế nhược điểm của mỗi hệ thống truyền thông điệp. Tuy vậy, Kafka là một giải pháp công nghệ mới cho phép tận dụng các lợi thế của các hệ thống truyền thông điệp truyền thống.

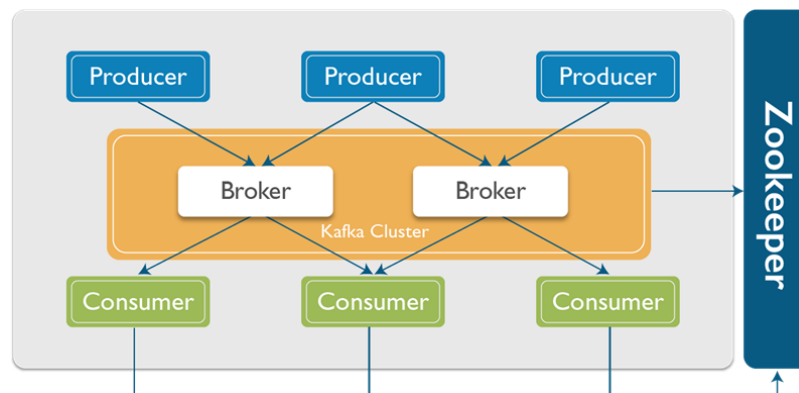
1.2. Apache Kafka

Kafka là một nền tảng truyền dòng dữ liệu (stream) phân tán nằm trong dự án mã nguồn mở (opensource). Dự án Kafka ban đầu được phát triển bởi LinkedIn và sau đó trở thành dự án Apache mã nguồn mở vào năm 2011. Kafka được viết bằng ngôn ngữ Scala và Java. Mục tiêu chính của Kafka là cung cấp một nền tảng có độ trễ thấp và thông lượng cao, cho phép xử lý các nguồn dữ liệu theo thời gian thực.

1.2.1. Kiến trúc thành phần Apache Kafka

1.2.1.1. Cấu trúc Apache Kafka

Kafka được xây dựng dựa trên mô hình publish/subscribe, tương tự như bất kỳ hệ thống message nào khác. Các ứng dụng (đóng vai trò là producer) gửi các messages (records) tới một node kafka (broker) và nói rằng những messages này sẽ được xử lý bởi các ứng dụng khác gọi là consumers. Các messages này sẽ được lưu trữ tại một nơi gọi là topic và sau đó consumer có thể subscribe tới topic đó và lắng nghe những messages này. Messages có thể là bất cứ thông tin gì như giá trị cảm biến, hành động người dùng, ... Topic có thể được xem như là tên của một danh mục mà các messages sẽ được lưu trữ và được đẩy vào.



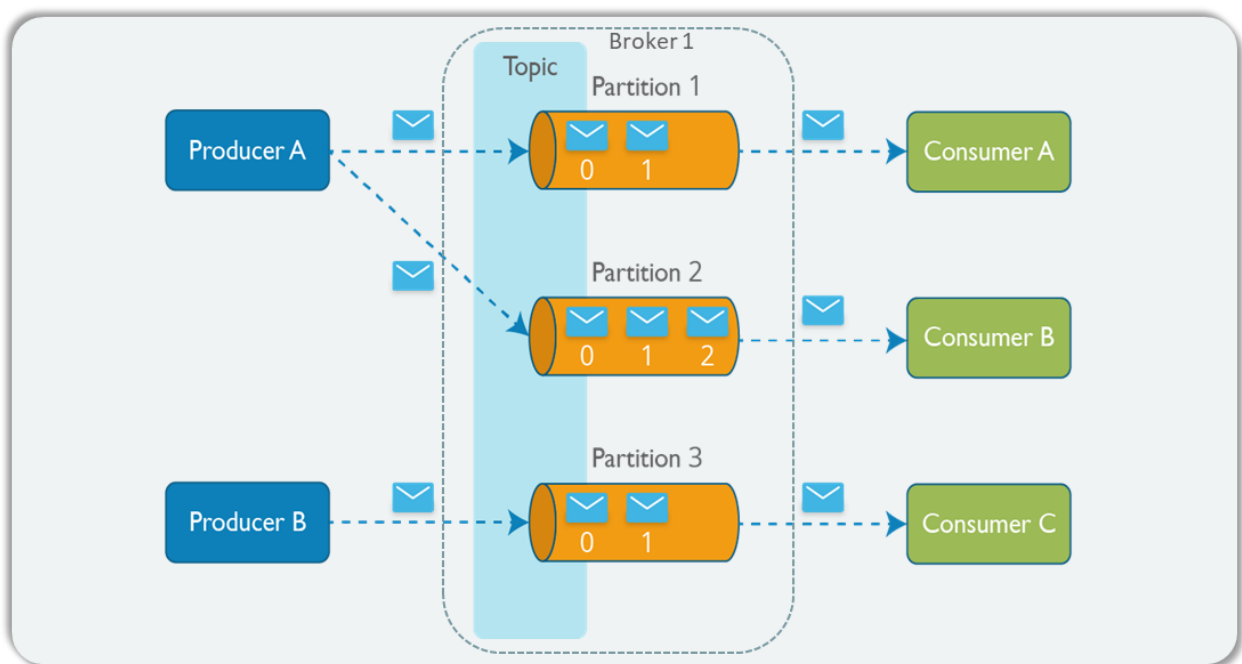
Hình 1.3. Mô hình cấu trúc kafka đơn giản

Ta có thể thấy, cấu trúc của kafka bao gồm các thành phần chính sau:

- **Producer:** Một producer có thể là bất kỳ ứng dụng nào có chức năng publish message vào một topic.
- **Messages:** Messages đơn thuần là byte array và developer có thể sử dụng chúng để lưu bất kỳ object với bất kỳ format nào - thông thường là String, JSON và Avro.

- **Topic:** Một topic là một category hoặc feed name nơi mà record được publish.
- **Partitions:** Các topic được chia nhỏ vào các đoạn khác nhau, các đoạn này được gọi là partition.
- **Consumer:** Một consumer có thể là bất kì ứng dụng nào có chức năng subscribe vào một topic và tiêu thụ các tin nhắn.
- **Broker:** Kafka cluster là một tập (set) các máy chủ, mỗi một set này được gọi là 1 broker.
- **Zookeeper:** được dùng để quản lý và bố trí các broker.

Cấu trúc chi tiết của kafka



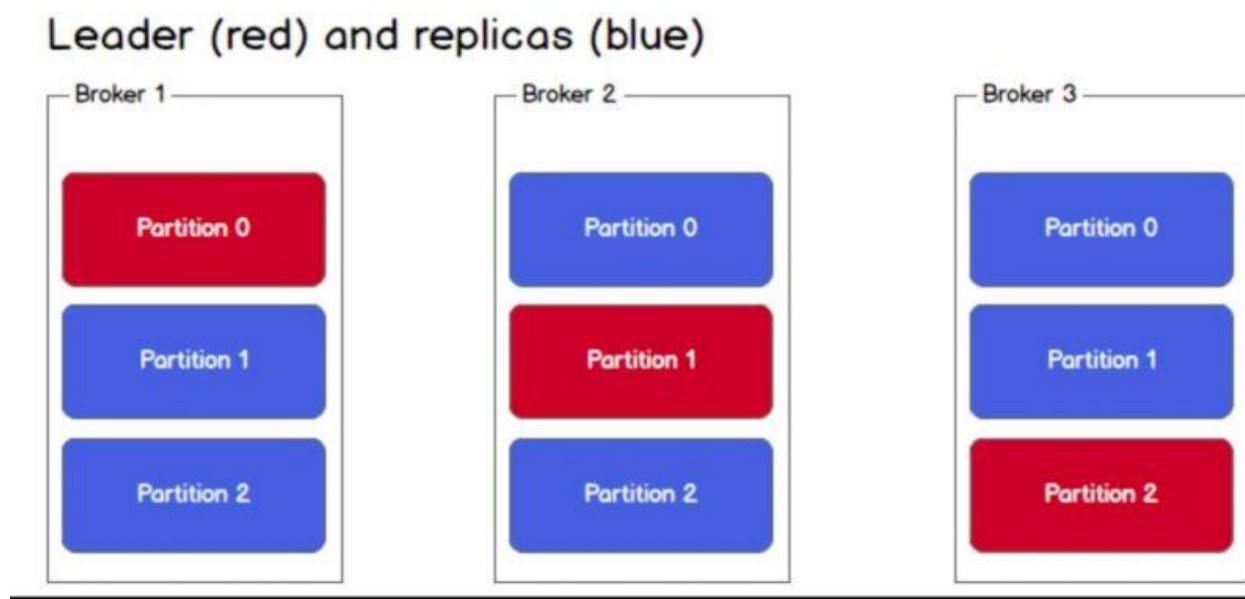
1.2.1.2. Partition

Topics trong kafka có thể có kích cỡ rất lớn, như vậy không nên lưu trữ tất cả dữ liệu của một topic trên một node, dữ liệu cần được phân chia ra thành nhiều partition sẽ giúp bảo toàn dữ liệu cũng như xử lý dữ liệu dễ dàng hơn. Partitions cho phép chúng

ta thực hiện subscribe song song tới một topic cụ thể bằng cách phân chia dữ liệu trong một topic cụ thể ra cho nhiều broker khác nhau (kafka node), mỗi partition có thể được đặt trên một máy riêng biệt – cho phép nhiều consumer đọc dữ liệu từ một topic diễn ra một cách song song.

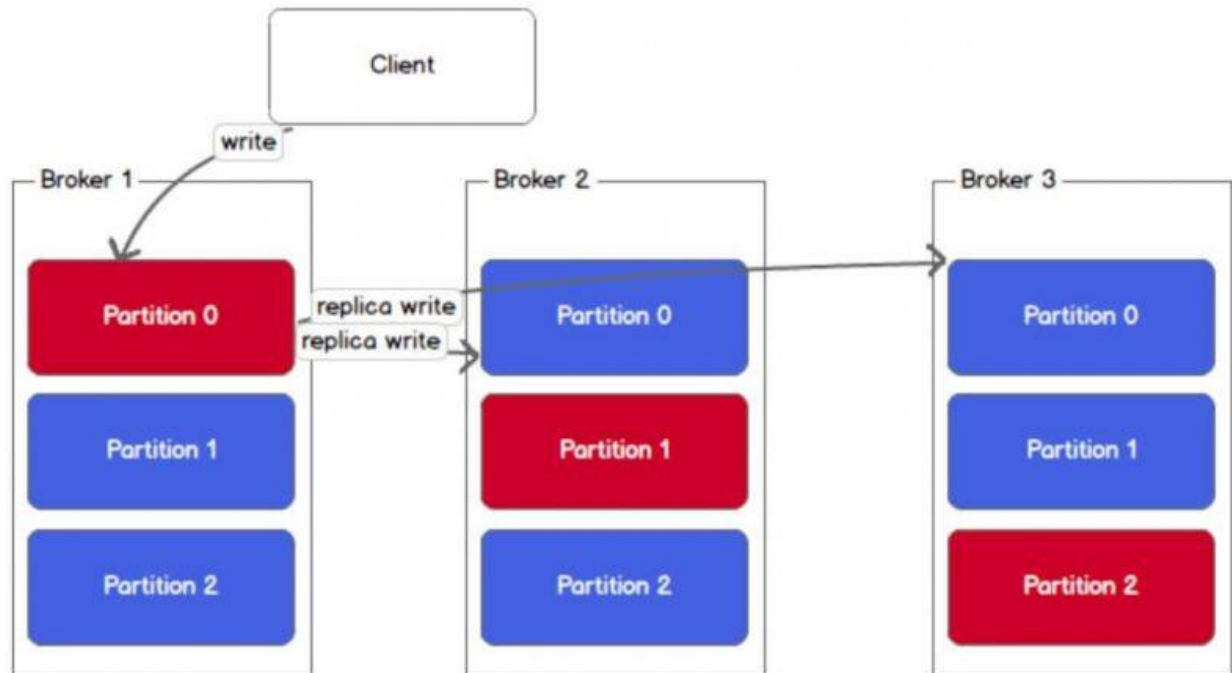
Để tăng tính khả dụng (availability) của partition, mỗi partition cũng có giá trị replicas của riêng nó. Để dễ hiểu hơn về kafka, mình sẽ trình bày bằng ví dụ với 3 node/broker.

Bây giờ, một topic sẽ được chia ra thành 3 partitions và mỗi broker sẽ có một bản copy của partition. Trong số những bản copy partition này, sẽ có một bản copy được bầu chọn làm leader, trong khi những bản copy khác chỉ thực hiện đồng bộ dữ liệu với partition leader.



Tất cả các hành động ghi và đọc tới một topic đều phải đi qua partition leader tương ứng và leader sẽ phối hợp để cập nhật dữ liệu mới tới các replica partition khác. Nếu leader bị hỏng, một trong các replica partition sẽ đảm nhận vai trò là một leader mới.

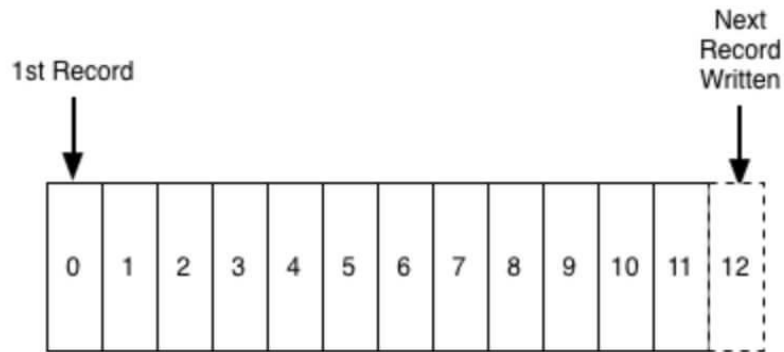
Leader (red) and replicas (blue)



Để một producer/consumer ghi/đọc message từ một partition, chắc chắn chúng cần phải biết leader. Trong Kafka, những thông tin như vậy gọi là metadata, và chúng được lưu trữ trong một dịch vụ gọi là Zookeeper.

1.2.1.3. Cấu trúc dữ liệu log

Sự thay đổi chính dẫn tới khả năng mở rộng và tăng hiệu suất đáng kể của kafka chính là log. Thường thì các lập trình viên khi mới tiếp cận kafka cảm thấy khá rối khi lần đầu tiên nghe đến “log“, bởi vì ta thường hiểu “log” chính là thuật ngữ được sử dụng trong log ứng dụng. Tuy nhiên, log ở đây chính là cấu trúc dữ liệu log. Log là một cấu trúc dữ liệu có thứ tự nhất quán mà chỉ hỗ trợ dạng nối thêm (append). Ta không thể chỉnh sửa hay xóa các records từ nó. Nó được đọc từ trái sang phải và được đảm bảo thứ tự các item.



Hình 1.4. Cấu trúc dữ liệu log trong kafka

Một nguồn dữ liệu sẽ ghi message vào log và một hoặc nhiều consumer khác sẽ đọc message từ log tại thời điểm họ lựa chọn. Mỗi entry trong log được định danh bởi một con số gọi là offset, hay nói một cách dễ hiểu hơn, offset giống như chỉ số tuần tự trong một array vậy. Vì chuỗi/offset chỉ có thể được duy trì trên từng node/broker cụ thể và không thể được duy trì đối với toàn bộ cluster, do đó Kafka chỉ đảm bảo sắp xếp thứ tự dữ liệu cho mỗi partition.

1.2.1.4. Lưu trữ và xử lý dữ liệu

Kafka lưu trữ tất cả message vào ổ đĩa (disk), không hề lưu trên RAM và được sắp xếp có thứ tự trong cấu trúc log cho phép kafka tận dụng tối đa khả năng đọc và ghi lên disk một cách tuần tự. Nó là một cách lựa chọn khá phổ biến để lưu trữ dữ liệu trên ổ đĩa mà vẫn có thể sử dụng tối đa hiệu năng, dựa trên một số lý do dưới đây:

- Kafka có một giao thức mà nhóm các message lại với nhau. Điều này cho phép các request network nhóm các message lại với nhau, giúp giảm thiểu chi phí sử dụng tài nguyên mạng, server, gom các message lại thành một cục và consumer sẽ tìm nạp một khối message cùng một lúc – do đó sẽ giảm tải disk cho hệ điều hành.

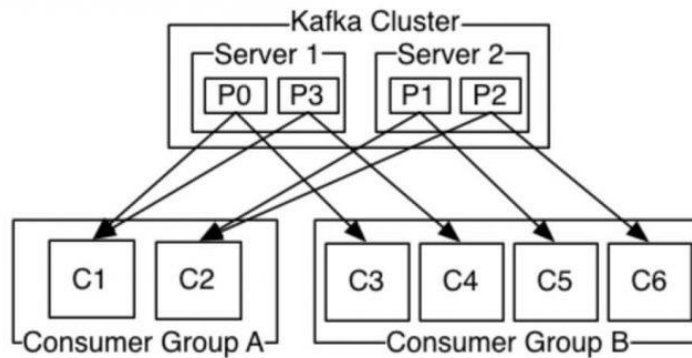
- Kafka phụ thuộc khá nhiều vào pagecache của hệ điều hành cho việc lưu trữ dữ liệu, sử dụng RAM trên máy một cách hiệu quả.
- Kafka lưu trữ các messages dưới định dạng nhị phân xuyên suốt quá trình (producer > broker > consumer), làm cho nó có thể tận dụng tối ưu hóa khả năng zero-copy. Nghĩa là khi hệ điều hành copy dữ liệu từ pagecache trực tiếp sang socket, hoàn toàn bỏ qua ứng dụng trung gian là kafka.
- Đọc/ghi dữ liệu tuyến tính trên disk nhanh. Vấn đề làm cho disk chậm hiện nay thường là do quá trình tìm kiếm trên disk nhiều lần. Kafka đọc và ghi trên disk tuyến tính, do đó nó có thể tận dụng tối đa hóa hiệu suất trên disk.

1.2.1.5. Consumer và Consumer Group

Consumer đọc các messages từ bất kỳ partition nào, cho phép mở rộng lượng thông điệp được sử dụng tương tự như cách các producer cung cấp message. Consumer cũng được tổ chức thành các consumer groups cho một topic cụ thể – mỗi consumer bên trong group đọc message từ một partition duy nhất, để tránh việc có 2 consumer cùng xử lý đọc cùng một message 2 lần và toàn bộ group xử lý tất cả các message từ toàn bộ topic.

- Nếu số consumer > số partition, khi đó một số consumer sẽ ở chế độ rảnh rỗi bởi vì chúng không có partition nào để xử lý.
- Nếu số partition > số consumer, khi đó consumer sẽ nhận các message từ nhiều partition.
- Nếu bạn có số consumer = số partition, mỗi consumer sẽ đọc message theo thứ tự từ 1 partition.

Để dễ hiểu hơn, ta có thể minh họa qua hình ảnh dưới đây



Trong đó, Server 1 giữ partition 0 và 3 và server 2 giữ các partition 1 và 2. Ta có 2 consumer groups là A và B. Group A có 2 consumer và group B có 4 consumer. Consumer group A có 2 consumer, vậy nên mỗi consumer sẽ đọc message từ 2 partition. Trong consumer group B, số lượng consumer bằng số partition nên mỗi consumer sẽ đọc message từ 1 partition.

Kafka tuân theo các quy tắc được cung cấp bởi broker và consumer. Nghĩa là kafka không theo dõi các record được đọc bởi consumer và do đó không biết gì về hành vi của consumer. Việc giữ lại các messages trong một khoảng thời gian được cấu hình trước và nó tùy thuộc vào consumer, để điều chỉnh thời gian sao cho phù hợp. Bản thân consumer sẽ thăm dò xem Kafka có message nào mới hay không và cho Kafka biết những record nào chúng muốn đọc. Điều này cho phép chúng tăng/giảm offset mà consumer muốn, do đó nó có thể đọc lại các message đã được đọc rồi và tái xử lý các sự kiện trong trường hợp gặp sự cố.

Ví dụ: nếu Kafka được cấu hình để giữ các messages tồn tại trong một ngày và consumer bị down lâu hơn 1 ngày, khi đó consumer sẽ mất message. Tuy nhiên, nếu consumer chỉ bị down trong khoảng 1 giờ đồng hồ, khi đó nó hoàn toàn có thể bắt đầu đọc lại message từ offset mới nhất.

1.2.1.6. Vai trò của Zookeeper

Zookeeper đóng vai trò là nơi lưu trữ dữ liệu phân tán dạng key-value. Nó được tối ưu hóa cho các tác vụ đọc nhanh nhưng ghi chậm. Kafka sử dụng Zookeeper để thực hiện việc bầu chọn leader của Kafka broker và topic partition. Zookeeper cũng được thiết kế có khả năng chịu lỗi cao, do đó Kafka phụ thuộc khá nhiều vào Zookeeper. Nó cũng được sử dụng để lưu trữ tất cả metadata như là:

- Offset cho mỗi partition của consumer group
- ACL (Access control list) – được sử dụng cho việc giới hạn truy cập/ủy quyền
- Quota của consumer/producer – số lượng message tối đa mỗi giây
- Partition Leader và trạng thái của chúng

Producer và consumer không tương tác trực tiếp với Zookeeper để biết leader của partition hay những metadata khác, thay vào đó chúng sẽ truy vấn metadata tới Kafka broker – sau đó Kafka sẽ tương tác với Zookeeper và gửi phản hồi metadata về lại cho chúng.

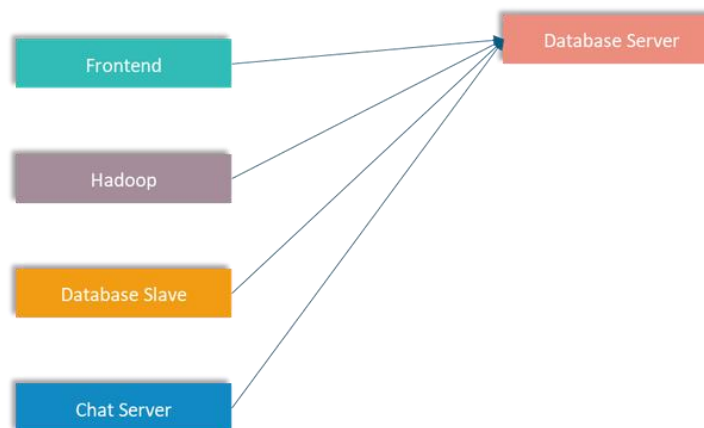
1.2.2. Một số ứng dụng của Kafka

❖ Xây dựng các hệ thống thương mại điện tử

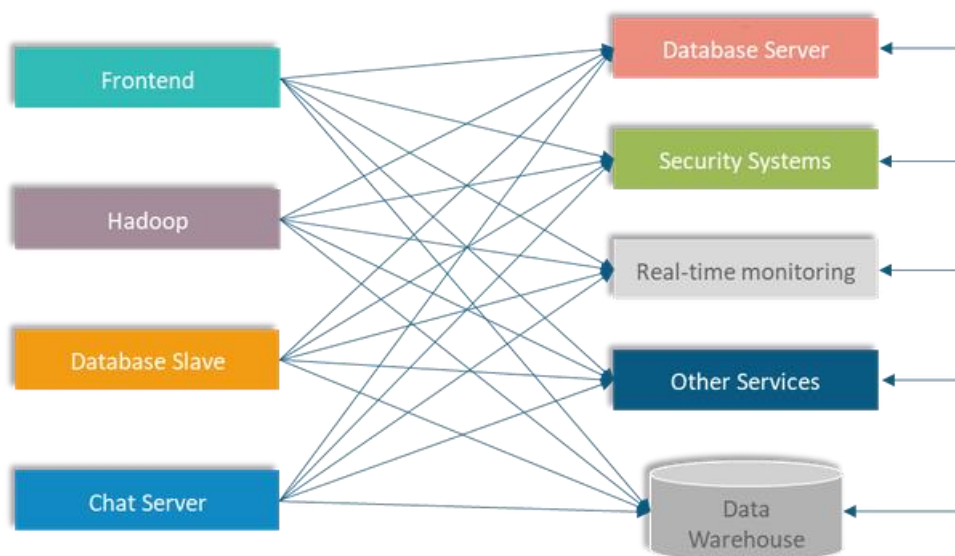
Kafka là dự án mã nguồn mở, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao và là hệ thống nhắn tin nhanh. Vì tính đáng tin cậy của nó, kafka đang dần được thay thế cho hệ thống gửi nhận thông điệp truyền thống. Nó được sử dụng cho các hệ thống nhắn tin thông thường trong các ngữ cảnh khác nhau. Đây là hệ quả khi khả năng mở rộng ngang và chuyên giao dữ liệu đáng tin cậy là những yêu cầu quan trọng nhất. Một vài use case cho kafka:

- Website Activity Monitoring - theo dõi hoạt động của website
- Stream Processing - xử lý dòng dữ liệu
- Log Aggregation - tổng hợp log
- Metrics Collection - thu thập dữ liệu

Ví dụ, trong một hệ thống thương mại điện tử có nhiều máy chủ thực hiện các công việc khác nhau, tất cả các máy chủ này đều muốn giao tiếp với database server, vì vậy chúng ta sẽ có nhiều data pipeline kết nối các server khác đến database server này, hình ảnh minh họa như sau:

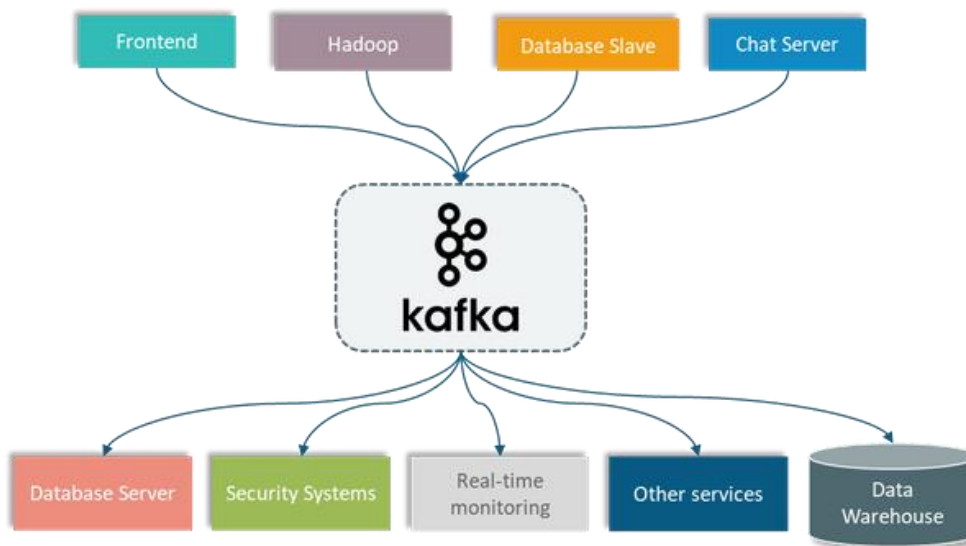


Nhưng trong thực tế, hệ thống thương mại điện tử sẽ còn phải kết nối đến một vài server khác nữa, kết nối vô cùng phức tạp. Như ta thấy ở ảnh dưới, data pipeline đang trở nên phức tạp theo sự gia tăng của số lượng hệ thống.



Hình 1.5. Sơ đồ minh họa kết nối trong hệ thống thương mại điện tử

Để giải quyết vấn đề này thì kafka ra đời. Kafka tách rời các data pipeline giữa các hệ thống để làm cho việc communicate giữa các hệ thống trở nên đơn giản hơn và dễ quản lý hơn.

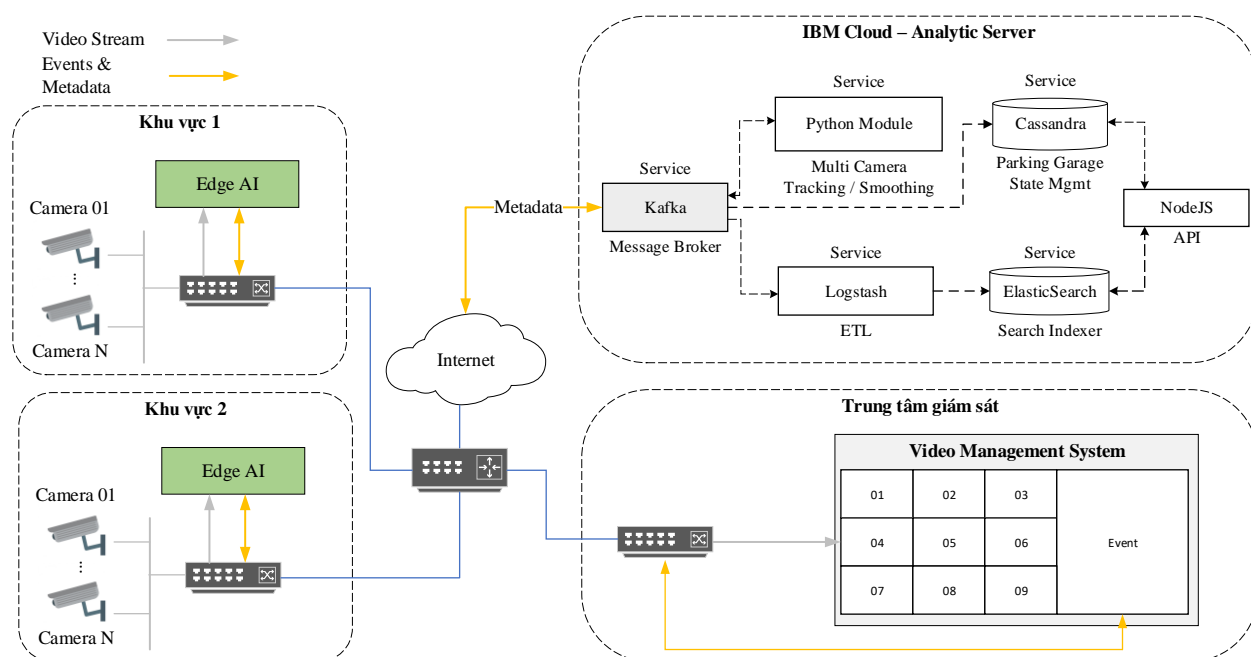


Hình 1.6. Ứng dụng Kafka trong xây dựng các hệ thống lớn

CHƯƠNG 2. ỨNG DỤNG KAFKA TRONG XỬ LÝ DÒNG DỮ LIỆU VIDEO THEO THỜI GIAN THỰC

2.1. Tích hợp kafka trong hệ thống xử lý dòng video theo thời gian thực

Hình vẽ dưới đây mô tả cấu trúc một hệ thống phân tích dòng video theo thời gian thực, ứng dụng nhiều công nghệ hiện đại. Cụ thể là ứng dụng hệ thống mạng Internet vạn vật IoT, các công nghệ điện toán đám mây (Cloud Computing), công nghệ trí tuệ nhân tạo (AI), ...

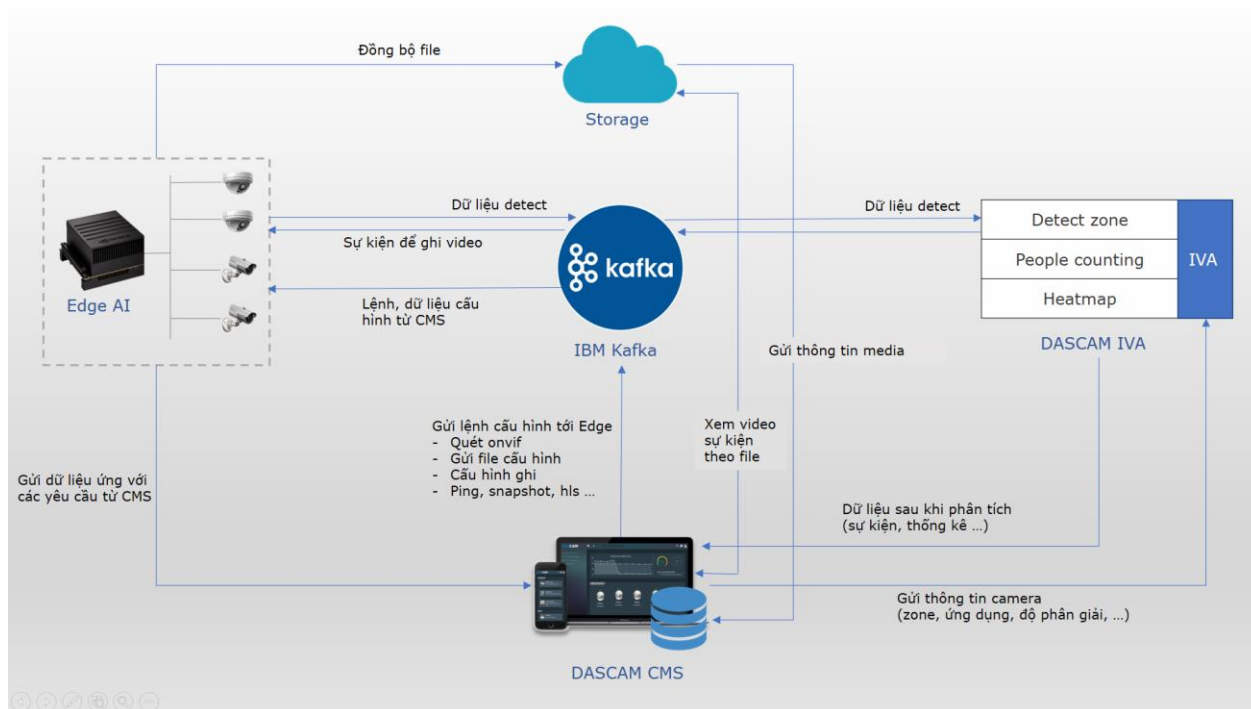


Hình 2.1. Kiến trúc hệ thống phân tích dòng video theo kiến trúc phân tán, với phần mềm quản lý tập trung

Trong đó, các dòng video sẽ được tiền xử lý bởi các thiết bị Edge-AI, bản chất là các thiết bị nhúng, được tích hợp phân cứng chuyên dụng chạy các thuật toán học sâu, nhằm kết xuất những thông tin quan trọng dưới dạng metadata theo thời gian thực. Dữ liệu metadata này là dữ liệu dạng text, nên khó khai thác thông tin nếu không biết

rõ ngữ cảnh thực tế của bài toán. Hơn nữa, việc che giấu và làm mờ thông tin trên dữ liệu text cũng đơn giản hơn nhiều so với dòng video. Dữ liệu này sẽ liên tục được gửi về máy chủ được triển khai trên hạ tầng đám mây, hoặc máy chủ cục bộ để phân tích, trích xuất những thông tin có ý nghĩa dưới dạng cảnh báo, gửi về cho hệ thống quản trị tập trung VMS, cho phép người dùng giám sát linh hoạt.

Dưới đây là cấu trúc chi tiết hệ thống ứng dụng Kafka để xử lý dòng video cho phép phát hiện người đi vào vùng cấm.



Hình 2.2. Tích hợp kafka trong hệ thống xử lý dòng video theo thời gian thực

2.2. Triển khai và cài đặt thành phần hệ thống

2.2.1. Cài đặt máy chủ phân tích

Máy chủ phân tích (analysis server) được xây dựng bằng framework Flask, sẽ tạo một dịch vụ điều khiển các hoạt động của ứng dụng AI tại cổng 5003, dựa vào các tương tác này, người dùng tương tác với hệ thống, các chức năng mặc định của API bao gồm:

- create: Khởi tạo ứng dụng AI cho camera
- show: Hiện thị các thông tin ứng dụng đang chạy, gồm tên cam. tên ứng dụng, dung lượng RAM và CPU tương ứng
- show-cam: Hiện thị các thông tình ứng dụng ứng với cam đầu vào
- update: Cập nhật cấu hình các ứng dụng
- restart: Khởi động lại các ứng dụng đang có trong hệ thống
- restart-cam: Khởi động lại các ứng dụng trên 1 cam
- fix: sửa lỗi các ứng dụng đang gặp
- stop: dừng hoạt động của camera
- stop-cam: dừng hoạt động của ứng dụng trên cam

Server nhận yêu cầu từ client gồm tên ứng dụng và các thông số cấu hình. Sau đó sẽ chạy các ứng theo yêu cầu của client.

❖ Cài đặt Python3

```
sudo apt update  
  
sudo apt install software-properties-common  
  
sudo add-apt-repository ppa:deadsnakes/ppa
```

```
sudo apt install python3.7
```

❖ Cài đặt pip3

```
sudo apt update  
sudo apt install python3-pip
```

❖ Cài đặt một số thư viện

```
pip3 install flask Flask-RESTful kafka-python scikit-image  
filterpy
```

❖ Cài đặt ứng dụng

```
git clone  
ssh://git@bkcs.dynu.net:2222/dungtt/server_analysis_scam.git  
cd server_analysis_scam  
sudo su  
export FLASK_APP=api2.py  
export FLASK_ENV=development  
export FLASK_DEBUG=0  
flask run --host 0.0.0.0 --port 5004 --no-debugger >/dev/null 2>&1  
&  
sudo python3 api.py >/dev/null 2>&1 &
```


2.2.2. Cài đặt máy chủ HLS

Máy chủ HLS dùng để quản lý các thiết bị camera và nhận các sự kiện cảnh báo từ máy chủ phân tích. Đồng thời đây cũng là media server cho phép upload và download các file m3u8 phục vụ việc xem video thời gian thực từ các camera.

Server được xây dựng bằng framework Laravel, triển khai trên web server apache2, sử dụng CSDL MySQL và redis server để lưu trữ các file segment trực tiếp trong RAM đáp ứng tốc độ xử lý cao.

❖ Cài đặt redis server

```
sudo apt update  
sudo apt install redis-server
```

❖ Cài đặt apache và cập nhật tường lửa (firewall)

```
sudo apt update  
sudo apt install apache2  
sudo ufw allow in "Apache Full"
```

❖ Cài đặt MySQL

```
sudo apt update  
sudo apt install mysql-server  
sudo mysql_secure_installation
```

❖ Cài đặt PHP7.4

```
sudo apt-get update

sudo apt -y install software-properties-common

sudo add-apt-repository ppa:ondrej/php

sudo apt-get update

sudo apt-get install php7.4 -y

sudo apt-get install php7.4-
{bcmath,bz2,intl,gd,mbstring,mysql,zip,fpm,curl,xml,mbstring} -y
```

❖ Cấu hình apache2

```
sudo nano /etc/apache2/sites-available/ 000-default.conf
```

Thêm các dòng cấu hình sau:

```
<VirtualHost *:80>

    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html/kafka-iva-laravel/public

    <Directory /var/www/html/kafka-iva-laravel/>

        Options FollowSymLinks

        AllowOverride All

        Order allow,deny
```

```
        allow from all

    </Directory>

</VirtualHost>
```

Khởi động lại máy chủ dịch vụ

```
sudo a2enmod rewrite

sudo service apache2 restart
```

Cài đặt gói phần mềm

```
git clone ssh://git@bkcs.dynu.net:2222/dungtt/kafka-iva-
laravel.git

cd kafka-iva-laravel

sudo cp .env.example .env

sudo composer install


sudo chown -R $USER:www-data storage

sudo chown -R $USER:www-data bootstrap/cache

sudo chown -R $USER:www-data public

chmod -R 775 storage

chmod -R 775 bootstrap/cache
```

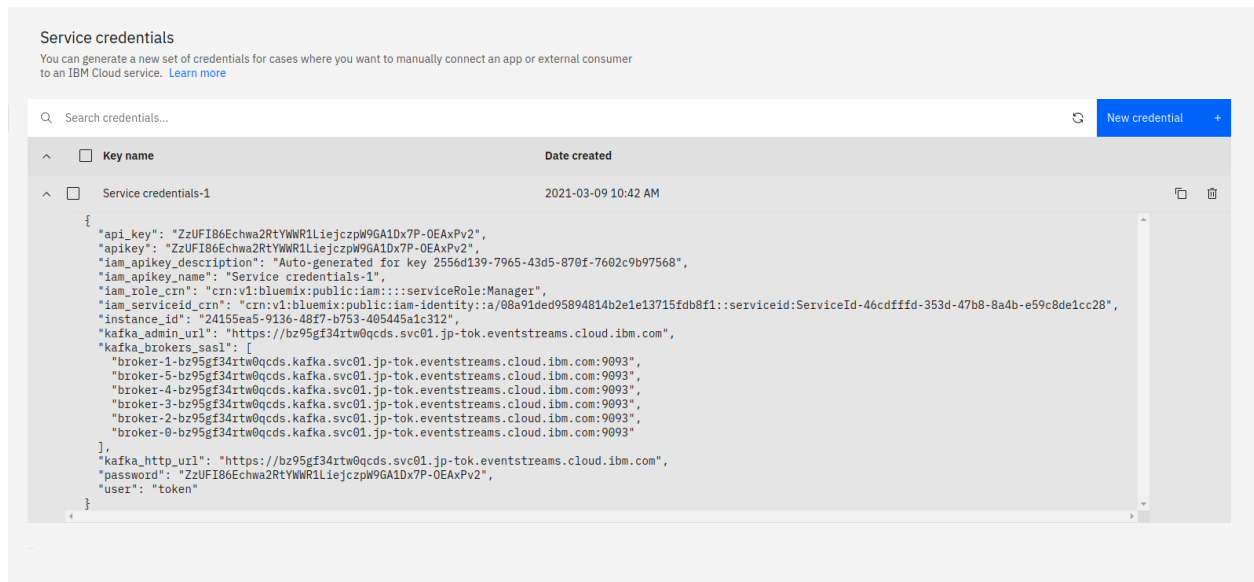
```
chmod -R 775 bootstrap/cache/
```

```
chmod -R 775 bootstrap/public/
```

Ngoài việc tự cấu hình máy chủ Kafka cục bộ, ta có thể sử dụng một số dịch vụ trên nền tảng đám mây được cung cấp sẵn như IBM Kafka. Để sử dụng dịch vụ, ta cần đăng ký tài khoản và trả phí cho lượng tài nguyên sử dụng. Tham khảo tại: [\[https://www.ibm.com/cloud/learn/apache-kafka\]](https://www.ibm.com/cloud/learn/apache-kafka)

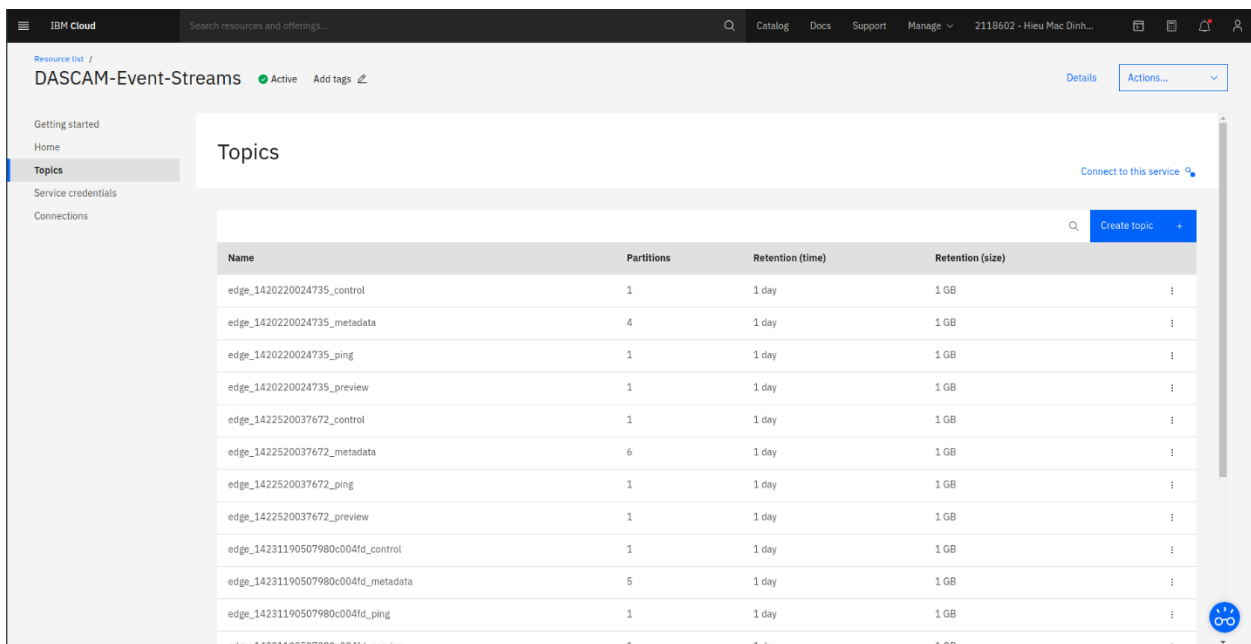
2.3. Kết quả đạt được

Ta có thông tin về tên đăng nhập (username), mật khẩu (password) và địa chỉ máy chủ dùng để cho máy khách (client) xác thực.



Hình 2.3. Thông tin tài khoản xác thực dịch vụ

Danh sách các topic lắng nghe sự kiện của Kafka



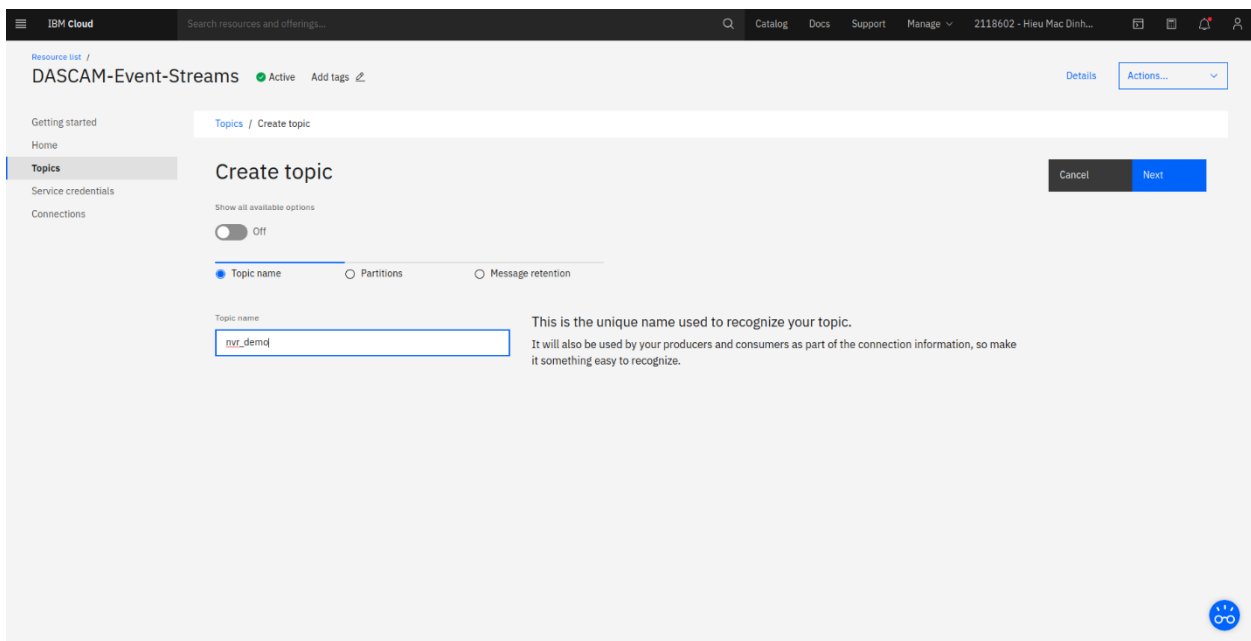
The screenshot shows the IBM Cloud console for the 'DASCAM-Event-Streams' service. The 'Topics' tab is selected, displaying a table of existing topics. The table has columns for Name, Partitions, Retention (time), and Retention (size). A 'Create topic' button is visible in the top right corner of the table area.

Name	Partitions	Retention (time)	Retention (size)
edge_1420220024735_control	1	1 day	1 GB
edge_1420220024735_metadata	4	1 day	1 GB
edge_1420220024735_ping	1	1 day	1 GB
edge_1420220024735_preview	1	1 day	1 GB
edge_1422520037672_control	1	1 day	1 GB
edge_1422520037672_metadata	6	1 day	1 GB
edge_1422520037672_ping	1	1 day	1 GB
edge_1422520037672_preview	1	1 day	1 GB
edge_14231190507980c004fd_control	1	1 day	1 GB
edge_14231190507980c004fd_metadata	5	1 day	1 GB
edge_14231190507980c004fd_ping	1	1 day	1 GB
edge_14231190507980c004fd_preview	1	1 day	1 GB

Hình 2.4. Danh sách các topic lắng nghe sự kiện của Kafka

Để tạo mới topic, ta làm như sau:

- Nhập tên của topic

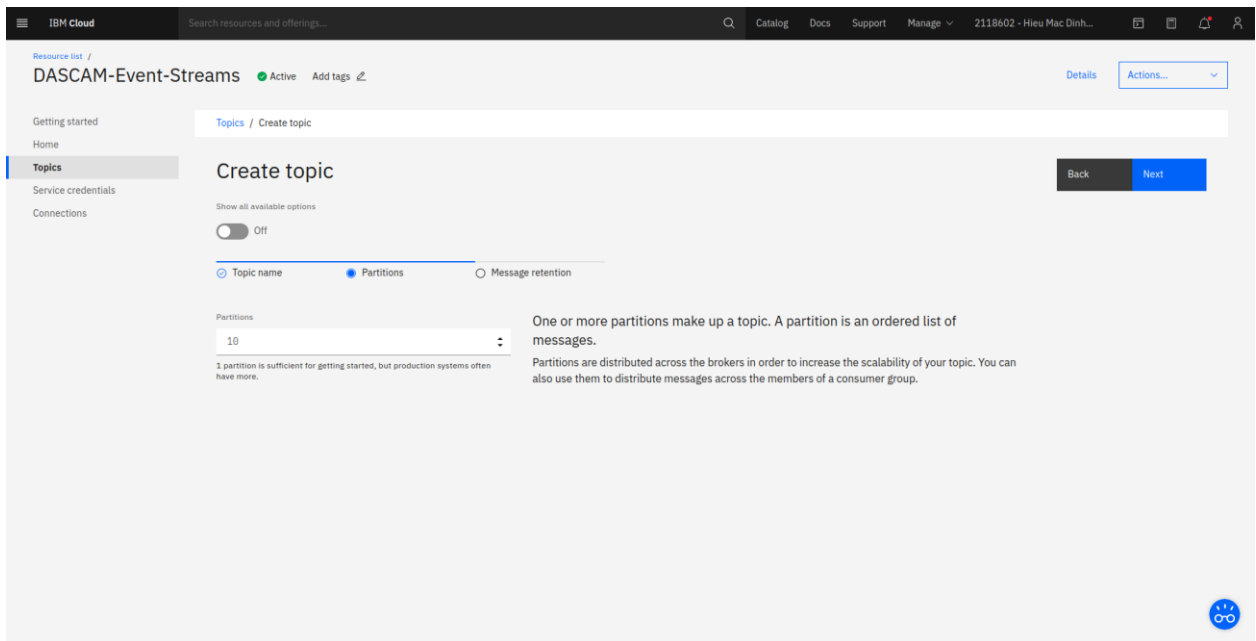


The screenshot shows the 'Create topic' form in the IBM Cloud console. The form has a 'Topic name' field with the value 'nvr_demo' entered. There are radio buttons for 'Partitions' and 'Message retention', and a 'Show all available options' toggle. A 'Next' button is visible at the bottom right.

Topic name:

This is the unique name used to recognize your topic. It will also be used by your producers and consumers as part of the connection information, so make it something easy to recognize.

- Nhập số lượng partition của topic



IBM Cloud

Search resources and offerings...

Resource list / DASCAM-Event-Streams Active Add tags

Getting started

Home

Topics

Service credentials

Connections

Topics / Create topic

Create topic

Show all available options

☐ Off

☒ Topic name ☒ Partitions ☐ Message retention

Partitions

10

1 partition is sufficient for getting started, but production systems often have more.

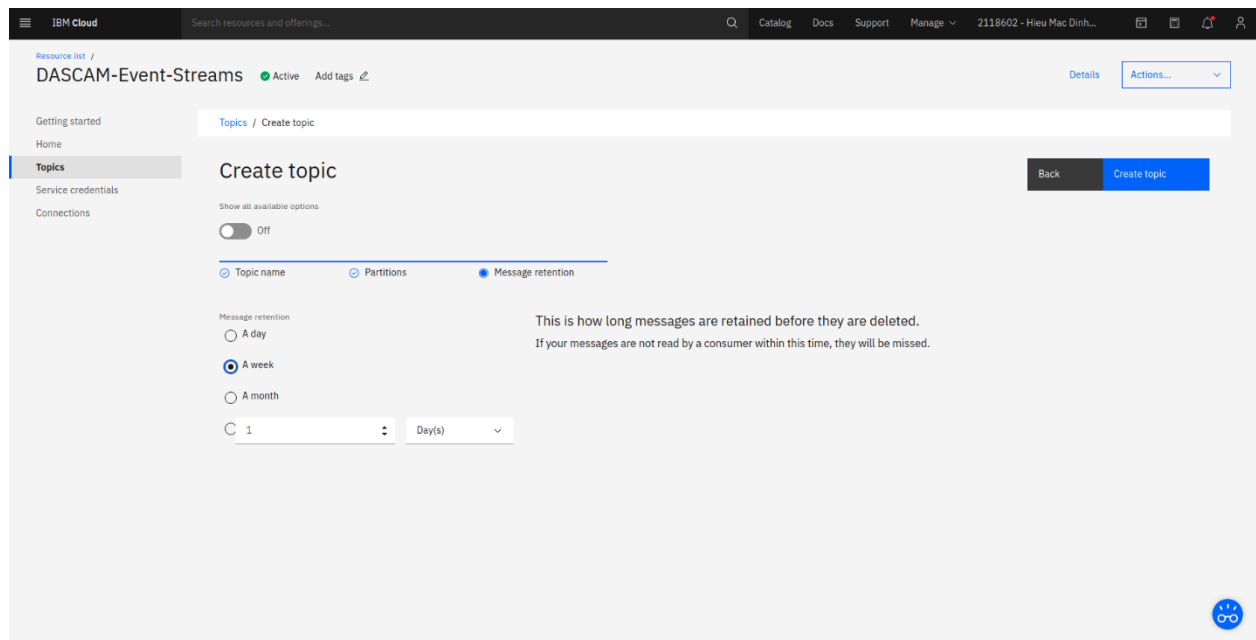
One or more partitions make up a topic. A partition is an ordered list of messages.

Partitions are distributed across the brokers in order to increase the scalability of your topic. You can also use them to distribute messages across the members of a consumer group.

Back Next

Hình 2.5. Giao diện tạo mới một topic

Thời gian các sự kiện được lưu trên hệ thống



IBM Cloud

Search resources and offerings...

Resource list / DASCAM-Event-Streams Active Add tags

Getting started

Home

Topics

Service credentials

Connections

Topics / Create topic

Create topic

Show all available options

☐ Off

☒ Topic name ☒ Partitions ☒ Message retention

Message retention

☐ A day

☒ A week

☐ A month

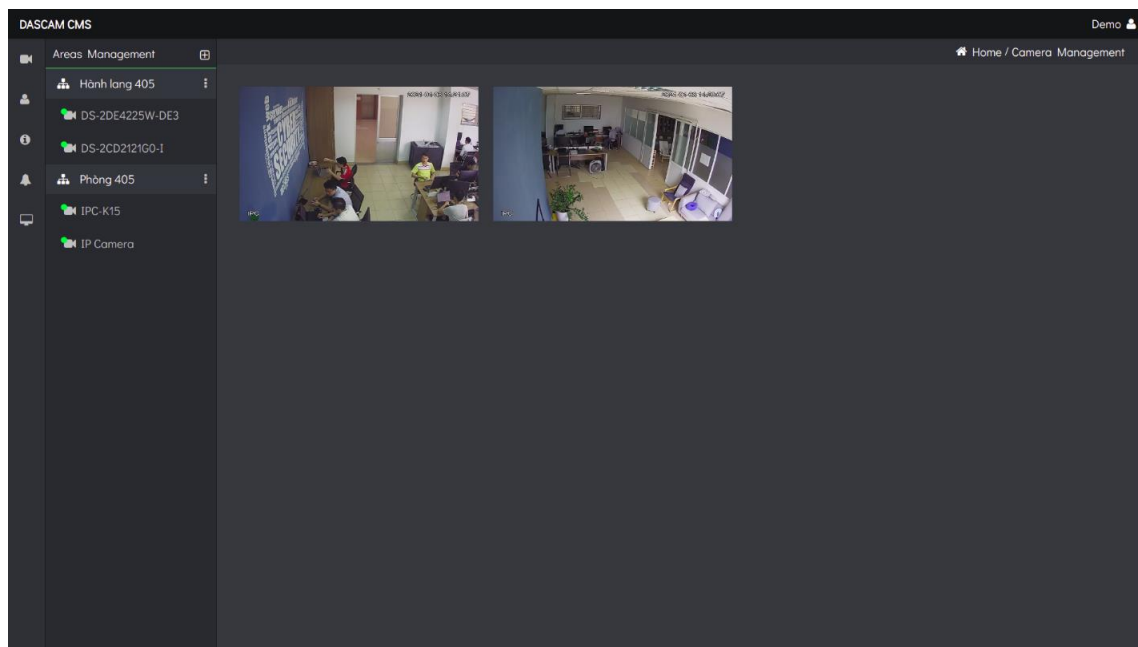
Day(s)

This is how long messages are retained before they are deleted.

If your messages are not read by a consumer within this time, they will be missed.

Back Create topic

Giao diện quản lý các camera



Hình 2.6. Giao diện quản lý các camera



Hình 2.7. Xem dòng video trực tiếp từ camera

Thông tin các **bouding box** sau khi detect người được gửi lên kafka

```
python3 consumer.py
[{"version": "4.0", "id": "69532", "@timestamp": "2021-04-08T14:18:50.022Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.83, "x2": 559.466, "y2": 1182.05, "label": "head"}, {"x1": 3911, "y1": 528.466, "x2": 379.975, "y2": 578.393, "label": "head"}, {"x1": 3739, "y1": 1179.05, "x2": 267.315, "y2": 1209.89, "label": "head"}, {"x1": 3468, "y1": 1027.89, "x2": 546.6, "y2": 1243.96, "label": "person"}, {"x1": 3886, "y1": 1140.1, "x2": 255.675, "y2": 1239.06, "label": "person"}, {"x1": 3868, "y1": 417.868, "x2": 516.761, "y2": 499.899, "label": "head"}, {"x1": 3914, "y1": 1016.05, "x2": 401.522, "y2": 1066.08, "label": "head"}, {"x1": 3912, "y1": 445.014, "x2": 367.125, "y2": 645.962, "label": "person"}, {"x1": 3873, "y1": 315.054, "x2": 524.607, "y2": 600.251, "label": "person"}]}, {"version": "4.0", "id": "69533", "@timestamp": "2021-04-08T14:18:50.052Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.95, "x2": 559.407, "y2": 1182.15, "label": "head"}, {"x1": 3911, "y1": 528.432, "x2": 380.307, "y2": 578.23, "label": "head"}, {"x1": 3739, "y1": 1179.05, "x2": 267.259, "y2": 1209.86, "label": "head"}, {"x1": 3468, "y1": 1028.18, "x2": 546.484, "y2": 1244.23, "label": "person"}, {"x1": 3886, "y1": 1139.87, "x2": 255.822, "y2": 1238.82, "label": "person"}, {"x1": 3868, "y1": 418.383, "x2": 516.629, "y2": 500.335, "label": "head"}, {"x1": 3914, "y1": 1015.77, "x2": 400.445, "y2": 1066.12, "label": "head"}, {"x1": 3912, "y1": 444.947, "x2": 367.381, "y2": 645.823, "label": "person"}, {"x1": 3873, "y1": 315.578, "x2": 524.295, "y2": 600.906, "label": "person"}]}, {"version": "4.0", "id": "69534", "@timestamp": "2021-04-08T14:18:50.156Z", "sensorId": "camera-2", "objects": [{"x1": 3868, "y1": 420.165, "x2": 515.526, "y2": 500.631, "label": "head"}, {"x1": 3913, "y1": 1116.41, "x2": 559.389, "y2": 1182.33, "label": "head"}, {"x1": 3739, "y1": 1179.1, "x2": 267.409, "y2": 1209.84, "label": "head"}, {"x1": 3911, "y1": 527.309, "x2": 379.989, "y2": 577.186, "label": "head"}, {"x1": 3914, "y1": 1017.54, "x2": 403.111, "y2": 1063.73, "label": "head"}, {"x1": 3468, "y1": 1026.95, "x2": 545.986, "y2": 1243.79, "label": "person"}, {"x1": 3912, "y1": 444.254, "x2": 368.226, "y2": 644.861, "label": "person"}, {"x1": 3886, "y1": 1140.09, "x2": 256.339, "y2": 1238.91, "label": "person"}, {"x1": 3873, "y1": 315.578, "x2": 524.295, "y2": 600.906, "label": "person"}]}, {"version": "4.0", "id": "69535", "@timestamp": "2021-04-08T14:18:50.189Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.36, "x2": 559.038, "y2": 1182.35, "label": "head"}, {"x1": 3911, "y1": 527.268, "x2": 380.041, "y2": 577.132, "label": "head"}, {"x1": 3739, "y1": 1179.07, "x2": 267.405, "y2": 1209.81, "label": "head"}, {"x1": 3468, "y1": 1027.19, "x2": 545.837, "y2": 1244.13, "label": "person"}, {"x1": 3886, "y1": 1139.77, "x2": 255.628, "y2": 1238.58, "label": "person"}, {"x1": 3868, "y1": 420.556, "x2": 515.568, "y2": 500.869, "label": "head"}, {"x1": 3914, "y1": 1017.8, "x2": 402.436, "y2": 1063.81, "label": "head"}, {"x1": 3912, "y1": 444.441, "x2": 368.131, "y2": 645.002, "label": "person"}, {"x1": 3873, "y1": 315.578, "x2": 524.295, "y2": 600.906, "label": "person"}]}, {"version": "4.0", "id": "69536", "@timestamp": "2021-04-08T14:18:50.237Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.24, "x2": 558.854, "y2": 1182.31, "label": "head"}, {"x1": 3911, "y1": 527.218, "x2": 380.095, "y2": 577.063, "label": "head"}, {"x1": 3739, "y1": 1179.01, "x2": 267.408, "y2": 1209.73, "label": "head"}, {"x1": 3468, "y1": 1027.33, "x2": 545.766, "y2": 1244.37, "label": "person"}, {"x1": 3886, "y1": 1139.39, "x2": 255.507, "y2": 1238.18, "label": "person"}, {"x1": 3868, "y1": 420.873, "x2": 515.441, "y2": 501.028, "label": "head"}, {"x1": 3914, "y1": 1017.97, "x2": 402.15, "y2": 1063.7, "label": "head"}, {"x1": 3912, "y1": 444.5, "x2": 368.153, "y2": 645.008, "label": "person"}, {"x1": 3873, "y1": 315.578, "x2": 524.295, "y2": 600.906, "label": "person"}]}, {"version": "4.0", "id": "69537", "@timestamp": "2021-04-08T14:18:50.290Z", "sensorId": "camera-2", "objects": [{"x1": 3868, "y1": 421.95, "x2": 514.148, "y2": 503.145, "label": "head"}, {"x1": 3913, "y1": 1116.58, "x2": 559.464, "y2": 1182.25, "label": "head"}, {"x1": 3739, "y1": 1179.03, "x2": 267.691, "y2": 1209.74, "label": "head"}, {"x1": 3911, "y1": 526.503, "x2": 379.488, "y2": 576.969, "label": "head"}, {"x1": 3914, "y1": 1019.88, "x2": 404.393, "y2": 1062.15, "label": "head"}, {"x1": 3468, "y1": 1028.82, "x2": 547.001, "y2": 1244.56, "label": "person"}, {"x1": 3912, "y1": 444.384, "x2": 369.125, "y2": 644.637, "label": "person"}, {"x1": 3873, "y1": 315.528, "x2": 525.561, "y2": 597.812, "label": "person"}, {"x1": 3886, "y1": 1139.49, "x2": 256.704, "y2": 1238.42, "label": "person"}, {"x1": 3921, "y1": 925.839, "x2": 377.638, "y2": 1103.72, "label": "person"}]}, {"version": "4.0", "id": "69538", "@timestamp": "2021-04-08T14:18:50.323Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.64, "x2": 559.338, "y2": 1182.29, "label": "head"}, {"x1": 3911, "y1": 526.509, "x2": 379.606, "y2": 577.012, "label": "head"}, {"x1": 3739, "y1": 1179.01, "x2": 267.65, "y2": 1209.72, "label": "head"}, {"x1": 3468, "y1": 1028.84, "x2": 546.915, "y2": 1244.49, "label": "person"}, {"x1": 3886, "y1": 1139.47, "x2": 256.482, "y2": 1238.4, "label": "person"}, {"x1": 3868, "y1": 422.233, "x2": 513.869, "y2": 503.498, "label": "head"}, {"x1": 3914, "y1": 1020.11, "x2": 404.592, "y2": 1062.12, "label": "head"}, {"x1": 3912, "y1": 444.832, "x2": 368.811, "y2": 645.058, "label": "person"}, {"x1": 3873, "y1": 315.528, "x2": 525.561, "y2": 597.812, "label": "person"}, {"x1": 3921, "y1": 925.843, "x2": 377.989, "y2": 1103.62, "label": "person"}, {"x1": 3873, "y1": 316.313, "x2": 525.514, "y2": 598.419, "label": "person"}]}, {"version": "4.0", "id": "69539", "@timestamp": "2021-04-08T14:18:50.371Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.7, "x2": 559.193, "y2": 1182.33, "label": "head"}, {"x1": 3911, "y1": 526.467, "x2": 379.649, "y2": 577.021, "label": "head"}, {"x1": 3739, "y1": 1179.07, "x2": 267.591, "y2": 1209.71, "label": "head"}, {"x1": 3468, "y1": 1028.85, "x2": 546.859, "y2": 1244.41, "label": "person"}, {"x1": 3886, "y1": 1139.42, "x2": 256.421, "y2": 1238.36, "label": "person"}, {"x1": 3868, "y1": 422.615, "x2": 513.861, "y2": 503.939, "label": "head"}, {"x1": 3914, "y1": 1020.35, "x2": 404.526, "y2": 1062.15, "label": "head"}, {"x1": 3912, "y1": 445.146, "x2": 368.557, "y2": 645.336, "label": "person"}, {"x1": 3873, "y1": 315.528, "x2": 525.561, "y2": 597.812, "label": "person"}, {"x1": 3921, "y1": 925.881, "x2": 377.861, "y2": 1103.47, "label": "person"}, {"x1": 3873, "y1": 316.484, "x2": 525.637, "y2": 598.312, "label": "person"}]}, {"version": "4.0", "id": "69540", "@timestamp": "2021-04-08T14:18:50.425Z", "sensorId": "camera-2", "objects": [{"x1": 3913, "y1": 1116.21, "x2": 558.116, "y2": 1181.98, "label": "head"}, {"x1": 3739, "y1": 1178.87, "x2": 267.598, "y2": 1209.73, "label": "head"}, {"x1": 3914, "y1": 1019.71, "x2": 398.49, "y2": 1062.75, "label": "head"}, {"x1": 3868, "y1": 423.665, "x2": 514.409, "y2": 504.28, "label": "head"}, {"x1": 3911, "y1": 526.479, "x2": 379.198, "y2": 577.044, "label": "head"}, {"x1": 3468, "y1": 1029.546, "x2": 546.781, "y2": 1244.66, "label": "person"}, {"x1": 3912, "y1": 445.639, "x2": 368.876, "y2": 646.04, "label": "person"}, {"x1": 3886, "y1": 1139.96, "x2": 256.385, "y2": 1239.11, "label": "person"}, {"x1": 3873, "y1": 315.634, "x2": 527.242, "y2": 595.459, "label": "person"}, {"x1": 3921, "y1": 926.402, "x2": 376.136, "y2": 1102.51, "label": "person"}]}
```

Server nhận các message từ kafka xử lý người đi vào vùng cấm, sau đó thông báo cho máy khách.

DASCAM CMS

Notification Management

Cameras IPC-K15

From 04/08/2021 To 04/09/2021 Apply

time occurred	Event	Location	Model
2021-04-08 14:17:03	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 14:09:03	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 13:15:35	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 13:12:24	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 13:11:47	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 13:07:55	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 13:05:36	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 13:03:12	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:58:40	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:57:52	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:56:52	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:55:19	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:51:38	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:49:28	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15
2021-04-08 12:48:58	Detected person has entered the restricted zone	khu vực ghế sofa	IPC-K15

IPC

Detail

Video: [watch video](#)

Location: khu vực ghế sofa

Camera Information

Device Name: IPC-K15

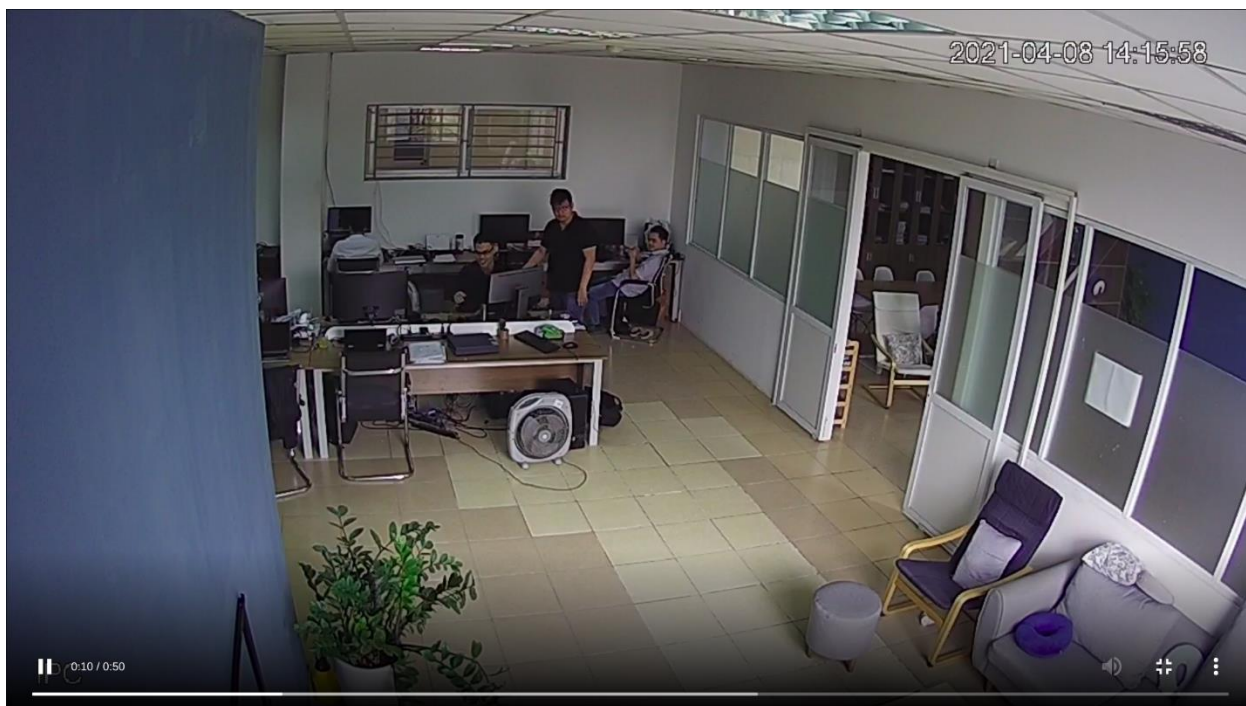
Manufacturer: Dahua

Firmware Version: 2.400.0000.16.R, Build Date 2017-08-31

IP Address: 192.168.200.48

Hình 2.8. Các thông điệp từ kafka thông báo có người đi vào vùng cấm

Video ghi lại sự kiện người đi vào vùng cấm



TÀI LIỆU THAM KHẢO

- [1] Bài giảng Xử lý ảnh, Trần Quang Đức, Đại học Bách Khoa Hà Nội, 2020.
- [2] Bài giảng Hệ phân tán, Trần Hải Anh, Đại học Bách Khoa Hà Nội, 2020.
- [3] <https://kafka.apache.org/intro>
- [4] <https://streaml.io/resources/tutorials/concepts/messaging-and-queuing>
- [5] <https://www.ibm.com/cloud/learn/apache-kafka>