# Báo cáo

**Sinh viên thực hiện**: Trương Công Toàn

**Đề tài** : Tìm hiểu về API Gateway open source WSO2

**Mô tả**: Tìm hiểu về API Gateway open source WSO2 và so sánh với việc tự xây dựng code API Gateway sử dụng Zuul Proxy

[Báo cáo 1](#_Toc86595935)

[**I.** **Tìm hiểu về hệ thống microservices, API Gateway? Tại sao một hệ thống microservices lại cần API Gateway?** 2](#_Toc86595936)

[**1.** **Hệ thống microservices** 2](#_Toc86595937)

[**1.1** **Khái niệm** 2](#_Toc86595938)

[**1.2** **Mô hình microservices** 2](#_Toc86595939)

[**1.3** **Ưu điểm** 3](#_Toc86595940)

[**1.4** **Nhược điểm** 4](#_Toc86595941)

[**1.5** **Khi nào nên sử dụng kiến trúc Microservices?** 4](#_Toc86595942)

[**2.** **API Gateway là gì?** 5](#_Toc86595943)

[**2.1 Vậy API Gateway là gì ?** 6](#_Toc86595944)

[**2.2 Lợi ích của việc sử dụng API Gateway:** 7](#_Toc86595945)

[**2.3 Nhược điểm khi sử dụng API gateway** 8](#_Toc86595946)

[**II.** **Tìm hiểu về API Gateway open source WSO2 và so sánh với việc tự xây dựng code API Gateway sử dụng Zuul Proxy** 9](#_Toc86595947)

[**1.** **API Gateway open source WSO2** 9](#_Toc86595948)

[**1.1** **Tìm hiểu về WSO2 API Manager** 10](#_Toc86595949)

[**2.** **So sánh với việc tự xây dựng code API sử dụng Zuul Proxy** 18](#_Toc86595950)

[**2.1** **Zuul là gì ?** 18](#_Toc86595951)

[**2.2** **Điểm khác biệt giữa xây dựng API sử dụng WSO2 API Manager và xây dựng API với Zuul** 19](#_Toc86595952)

[**III.** **Kết luận** 21](#_Toc86595953)

[**IV. Tài liệu tham khảo** 22](#_Toc86595954)

1. **Tìm hiểu về hệ thống microservices, API Gateway? Tại sao một hệ thống microservices lại cần API Gateway?**
2. **Hệ thống microservices**

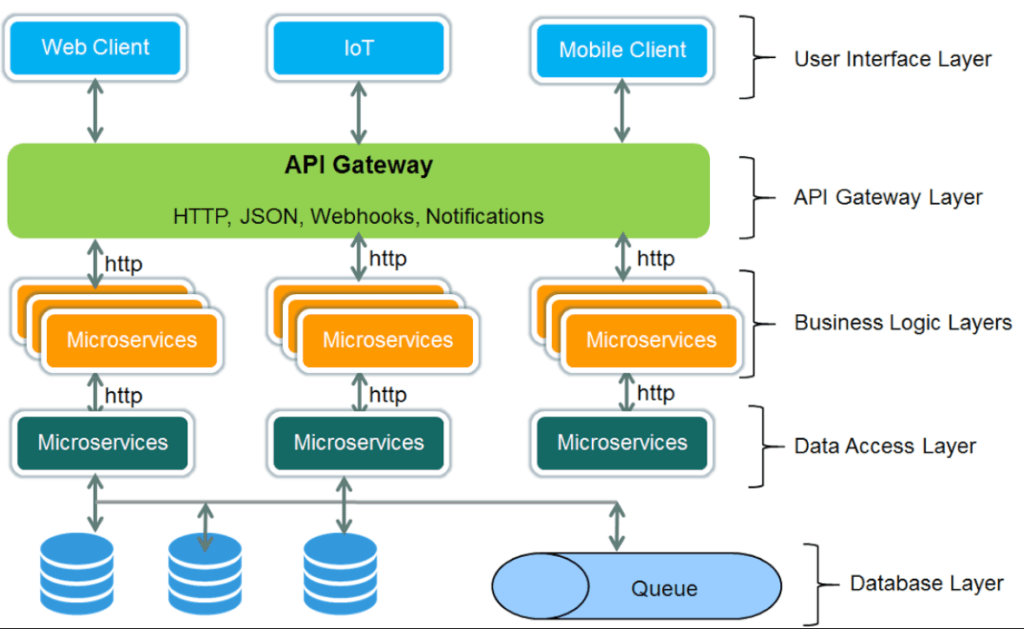
Microservices hiện được quan tâm trong giới phần mềm, công nghệ với nhiều bài viết, blog, thảo luận, truyền thông, hội thảo. Kỳ vọng về khả năng của Microservices đang lên đỉnh giống như một xu hướng thời trang đang lan rộng.

Mỗi microservices là một đơn vị độc lập, cùng nhiều đơn vị khác tạo nên một ứng dụng lớn. Bằng cách chia ứng dụng thành các đơn vị nhỏ, mỗi phần của nó được triển khai độc lập và có thể mở rộng, có thể được viết bởi các nhóm khác nhau bằng các ngôn ngữ lập trình khác nhau và được kiểm tra riêng lẻ.

* 1. **Khái niệm**

Với kiến trúc Microservices mỗi dịch vụ sẽ được chia nhỏ thành nhiều thành phần khác nhau, mỗi thành phần sẽ hoạt động độc lập, được phát triển độc lập và chỉ xử lý các nghiệp vụ chức năng của nó. Mỗi thành phần cũng sẽ không lệ thuộc vào công nghệ phát triển với các thành phần khác.

* 1. **Mô hình microservices**



* 1. **Ưu điểm**

**Một số ưu điểm của hệ thống Microservices có thể kế đến như :**

* Cho phép dễ dàng continuous delivery và deployment các ứng dựng lớn, phức tạp:
* Cải thiện khả năng bảo trì - mỗi service tương đối nhỏ do đó dễ hiểu và thay đổi hơn
* Khả năng testing dễ dàng hơn - các services nhỏ hơn và nhanh hơn để test
* Khả năng triển khai tốt hơn - các services có thể được triển khai độc lập
* Cho phép các services được phát triển bởi những team khác nhau. Mỗi team có thể phát triển, thử nghiệm, triển khai và mở rộng quy mô dịch vụ của mình một cách độc lập với tất cả các team khác.
* Thuận lợi trong việc phát triển lâu dài cho dịch vụ. Vì mỗi thành phần chỉ thực hiện đúng một chức năng, nên việc cải tiến sẽ dễ dàng thực hiện và không gây gián đoạn tới các thành phần khác. Mỗi thành phần sẽ giao tiếp với nhau qua giao diện riêng.
* Giảm thiểu rủi ro: Nếu có lỗi trong một service thì chỉ có service đó bị ảnh hưởng. Các services khác sẽ tiếp tục xử lý các yêu cầu. Trong khi đó, một thành phần hoạt động sai của kiến trúc một khối có thể làm ảnh hưởng toàn bộ hệ thống.
* Dễ dàng thay đổi sử dụng các công nghệ mới: Khi triển khai các services bạn có thể lựa chọn nhiều công nghệ mới. Tương tự khi có thay đổi lớn đối với các services hiện có bạn có thể dễ dàng thay đổi công nghệ.
  1. **Nhược điểm**

**Một số nhược điểm của hệ thống microservices:**

* Các nhà phát triển phải đối phó với sự phức tạp của việc tạo ra một hệ thống phân tán:
* Cần implement việc communication giữa các inter-services
* Handle partial failure là rất phức tạp vì một luồng xử lý cần đi qua nhiều services
* Việc thực hiện các requests trải rộng trên nhiều services khó khăn hơn, điều này cũng đòi hỏi sự phối hợp cẩn thận giữa các teams
* Khó khăn trong việc đảm bảo toàn vẹn CSDL nếu triển khai theo kiến trúc cơ sở dữ liệu phân vùng
* Triển khai và quản lý microservices nếu làm thủ công theo cách đã làm với ứng dụng một khối phức tạp hơn rất nhiều
* Phải xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được hoặc thông điệp gửi đến nhiều đích đến vào các thời điểm khác nhau
  1. **Khi nào nên sử dụng kiến trúc Microservices?**

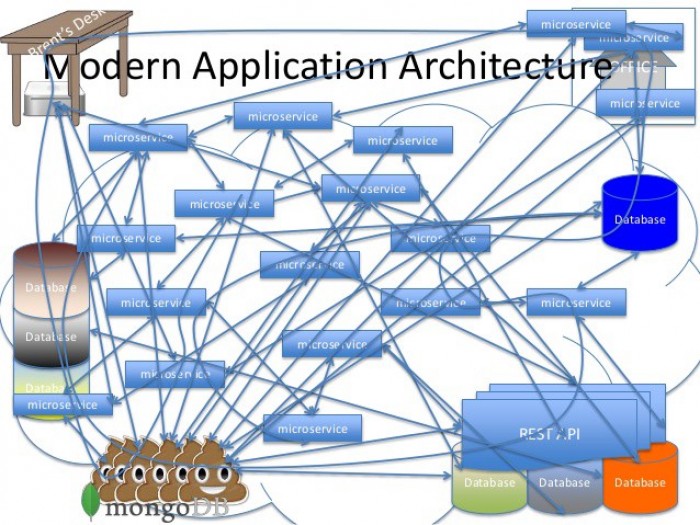
Một thách thức đối với việc sử dụng kiến trúc Microservices là khi nào nên sử dụng nó.

Khi phát triển phiên bản đầu tiên của ứng dụng, chúng ta thường không gặp phải các vấn đề mà Microservices giải quyết. Hơn nữa, sử dụng một kiến trúc phân tán, phức tạp sẽ làm chậm quá trình phát triển.

Vì vậy trừ khi chúng ta có một hệ thống quá phức tạp để quản lý bằng Monolithic Architecture và với hệ thống yêu cầu độ phát triển nhanh thì không nên sử dụng Microservices Architecture.

1. **API Gateway là gì?**

Một hệ thống microservices trung bình sẽ có một vài cho tới hàng trăm services khác nhau, nếu như client giao tiếp trực tiếp với các services này thì sơ đồ giao tiếp giữa client và hệ thống của chúng ta sẽ trông rất rối , có thể tham khảo ví dụ như sau :

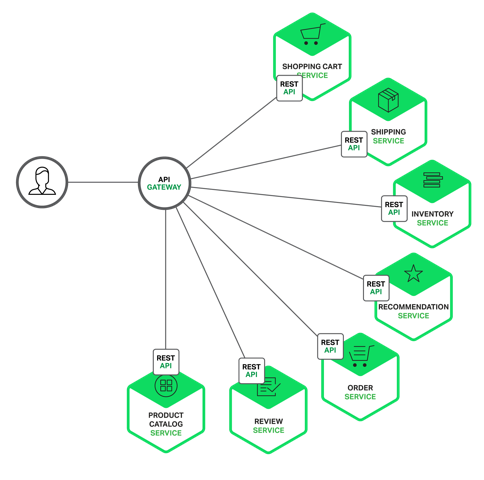


Chính vì sợ rối ren như trên cho nên mới xuất hiện một giải pháp đó chính là API Gateway (tạm dịch là cổng kết nối API) đóng vai trò là một cổng trung gian giữa client và hệ thống microservices đằng sau.

### **2.1 Vậy API Gateway là gì ?**

* API Gateway là phương pháp tiếp cận tối ưu cho kiến trúc microservices. Một cổng kết nối API là một máy chủ truy xuất duy nhất vào hệ thống. Nó cũng tương tự như mẫu thiết kế Facade dựa trên thiết kế hướng đối tượng.
* Cổng kết nối API che giấu đi thông tin kiến trúc hệ thống nội bộ và nó cung cấp các API tùy chỉnh cho mỗi Client. Cổng kết nối API còn có trách nhiệm xác thực, giám sát, cân bằng tải, caching, định hình yêu cầu và quản lí thông tin, xử lí phản hồi tĩnh.
* Cổng kết nối API làm nhiệm vụ định tuyến các yêu cầu, kết hợp và chuyển đổi các giao thức. Tất cả yêu cầu từ Client đều đi qua cổng kết nối API. Sau đó cổng kết nối API định tuyến các yêu cầu này tới microservices phù hợp. Cổng kết nối API Gateway sẽ xử lý một yêu cầu người dùng bằng cách gọi đến một loạt microservices rồi tổng hợp các kết quả. Nó có thể chuyển đổi giữa các giao thức web như HTTP, WebSocket và các giao thức nội bộ không thân thiện với web.

Sơ đồ hệ thống :



* Ngoài nhiệm vụ chính là proxy request thì một hệ thống API Gateway thường sẽ đảm nhận luôn vài vai trò khác như bảo mật API, monitoring, analytics số lượng requests cũng như tình trạng hệ thống phía sau.

### **2.2 Lợi ích của việc sử dụng API Gateway:**

* Che dấu được cấu trúc của hệ thống microservices với bên ngoài:

Clients sẽ tương tác với hệ thống của chúng ta thông qua api gateway chứ không gọi trực tiếp tới một services cụ thể, các endpoints của các services sẽ chỉ được gọi nội bộ, tức là gọi giữa các services với nhau hoặc được gọi từ API gateway, người dùng sẽ gọi các api này thông qua các public endpoints từ API Gateway.

Chính vì vậy cho nên phía client không cần và cũng không thể biết được các services phía backend được phân chia như thế nào, việc refactor code frontend cũng dễ dàng hơn đối với lập trình viên.

* Phần code phía frontend sẽ gọn gàng hơn:

Vì không phải tracking nhiều endpoints, tất cả chỉ việc gọi đến api gateway nên phần code frontend sẽ gọn gàng hơn so với việc phải tracking hàng tá endpoints trên từng services một, nhất là khi mà hệ thống ngày một phình to ra.

* Dễ dàng theo dõi và quản lý traffic.

Đa phần những hệ thống API gateway rộng rãi hiện nay đều sẽ đi kèm tính năng theo dõi , quan sát cũng như quản lý lượng traffic bằng GUI hoặc thông qua những APIs của hệ thống Gateway, VD như với Kong (bản EE)

* Requests caching và cân bằng tải :

API Gateway sẽ kiêm luôn vai trò load balancer của hệ thống, requests sẽ không được gửi trực tiếp đến backend nên sẽ giảm thiểu được rủi ro hệ thống bị quá tải.

* Thêm một lớp bảo mật nữa cho hệ thống:

API gateway giúp ngăn chặn các cuộc tấn công bằng cách thêm một lớp bảo vệ các loại tấn công như ddos, slq injections,...

* Thay thế authentication services:

API gateway thường cung cấp nhiều cơ chế xác thực, chúng ta có thể sử dụng nó để xác thực người dùng luôn, giúp tiết kiệm thời gian và làm hệ thống chúng ta đơn giản hơn. VD một vài cơ chế xác thực hỗ trợ bởi Kong API gateway

**2.3 Nhược điểm khi sử dụng API gateway**

API có một số nhược điểm có thể kể đến như là :

* Tăng thời gian response

Vì phải đi qua server trung gian cho nên việc response sẽ bị trễ hơn so với việc gọi trực tiếp tới hệ thống.

* Thêm tác nhân gây lỗi

Để sử dụng API Gateway thì chúng ta sẽ phải config, rồi chỉnh sửa code, quản lý server gateway, bla bla...Khiến cho chúng ta có thêm việc phải lo, chẳng may gateway có lỗi thì requests sẽ không thể tới được phía server.

* Có thể gây nghẽn cổ chai

Nếu như không được scale hay config hợp lý thì gateway sẽ có thể bị quá tải và làm chậm hệ thống của chúng ta.

* Tốn thêm chi phí

Tiền server, tiền điện, tiền quản lý hệ thống api gateway, với hệ thống lớn cần các tính năng xịn sò thì còn tốn thêm tiền mua bản Enterpise của các api gateway nữa này, tính ra cũng không rẻ chút nào cả.

1. **Tìm hiểu về API Gateway open source WSO2 và so sánh với việc tự xây dựng code API Gateway sử dụng Zuul Proxy**
2. **API Gateway open source WSO2**

WSO2 API Gateway được cung cấp bởi WSO2 EI. WSO2 API Gateway cung cấp runtime và backend component (một API proxy) cho các lệnh gọi API. Nó bảo mật, bảo vệ, quản lý và cân bằng các lời gọi API bằng cách chặn các yêu cầu API và áp dụng các chính sách như điều chỉnh và bảo mật bằng cách sử dụng trình xử lý và quản lý số liệu thống kê API. Sau khi xác thực một cách xử lý (policy), Gateway chuyển các lời gọi dịch vụ Web đến các backend thực tế. Nếu lời gọi dịch vụ là một token request, Gateway sẽ chuyển nó trực tiếp đến Trình quản lý khóa (Key Manager).

WSO2 cung cấp nền tảng phần mềm trung gian mã nguồn mở(open source) hoàn chỉnh duy nhất. Với thiết kế thành phần mang tính cách mạng, nó cũng là dịch vụ nền tảng mã nguồn mở duy nhất dành cho đám mây riêng và công cộng hiện nay. Với nó, việc di chuyển và tích hợp liền mạch giữa các máy chủ, đám mây riêng và đám mây công cộng giờ đây đã trở thành hiện thực.

* 1. **Tìm hiểu về WSO2 API Manager**

WSO2 API Manager là một giải pháp hoàn chỉnh để thiết kế và xuất bản API, tạo và quản lý cộng đồng nhà phát triển và để định tuyến lưu lượng truy cập API. Nó thúc đẩy các thành phần quản trị, bảo mật và quản trị sẵn sàng sản xuất ​​WSO2 Enterprise Service Bus, Máy chủ nhận dạng WSO2 và Đăng ký quản trị WSO2 (WSO2 Identity Server). Ngoài ra, nó tận dụng Trình giám sát hoạt động kinh doanh WSO2 để phân tích dữ liệu lớn, cung cấp cho bạn cái nhìn sâu sắc tức thì về hành vi API.

Là một giải pháp hoàn chỉnh để thiết kế,triển khai và quản lý các API. Nó đi kèm với một API Publisher cho phép thiết kế API của mình và chuyển đổi nó thành một API được quản lý.

Các API Publisher cung cấp một tập hợp các capabilities phong phú như là :

* Design 🡪 Lập kế hoạch và thiết kế một RESTful API
* Implement 🡪 Deploy và Test như một API nguyên mẫu
* Manage 🡪Publish và expose dưới dạng một API RESTful đã được quản lý

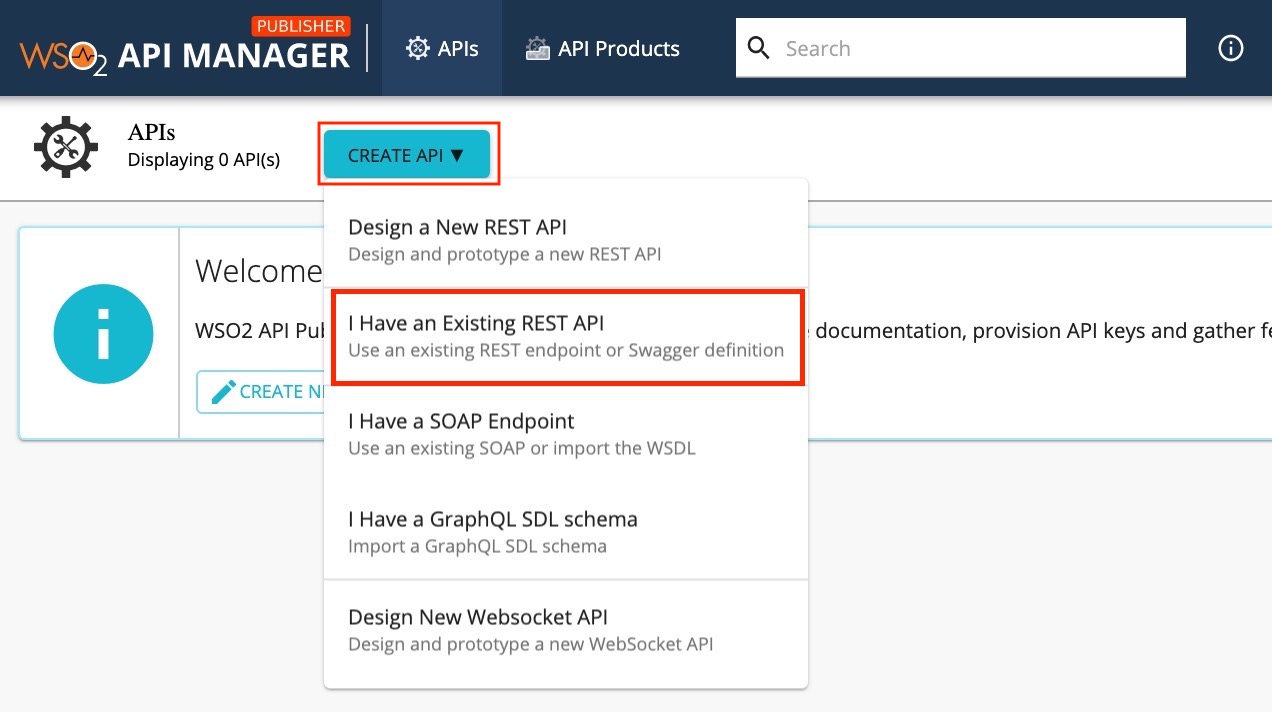
**Tiếp theo chúng ta sẽ tìm hiểu cách thiết kế API với WSO2 API Manager**

**Các bước thực hiện :**

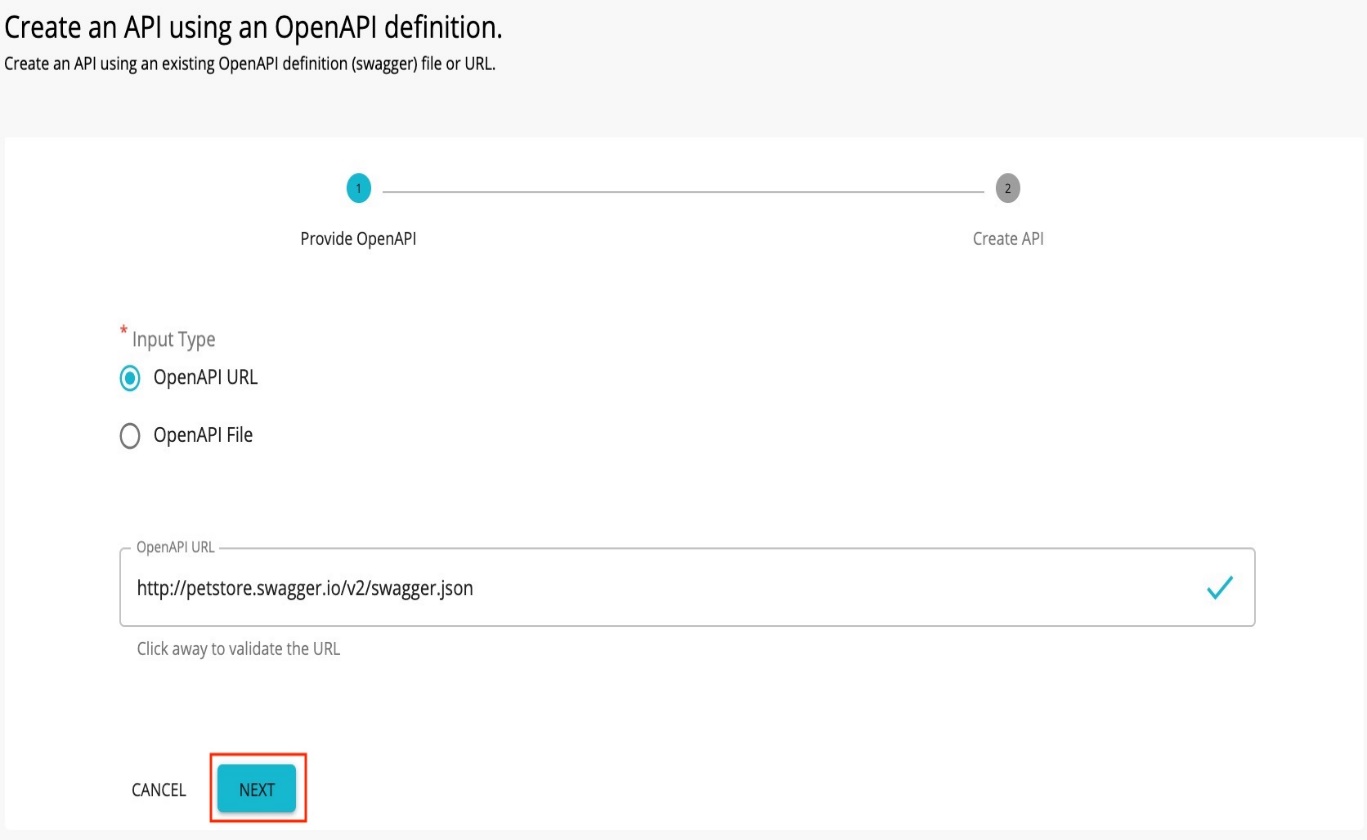
1. Đăng nhập vào WSO2 API Publisher bằng tên người dùng và mật khẩu.

****

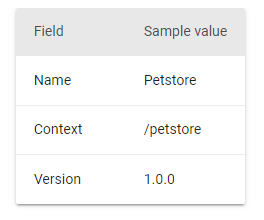
1. Đi tới **CREATE API và chọn**  **I Have an Existing API**

****

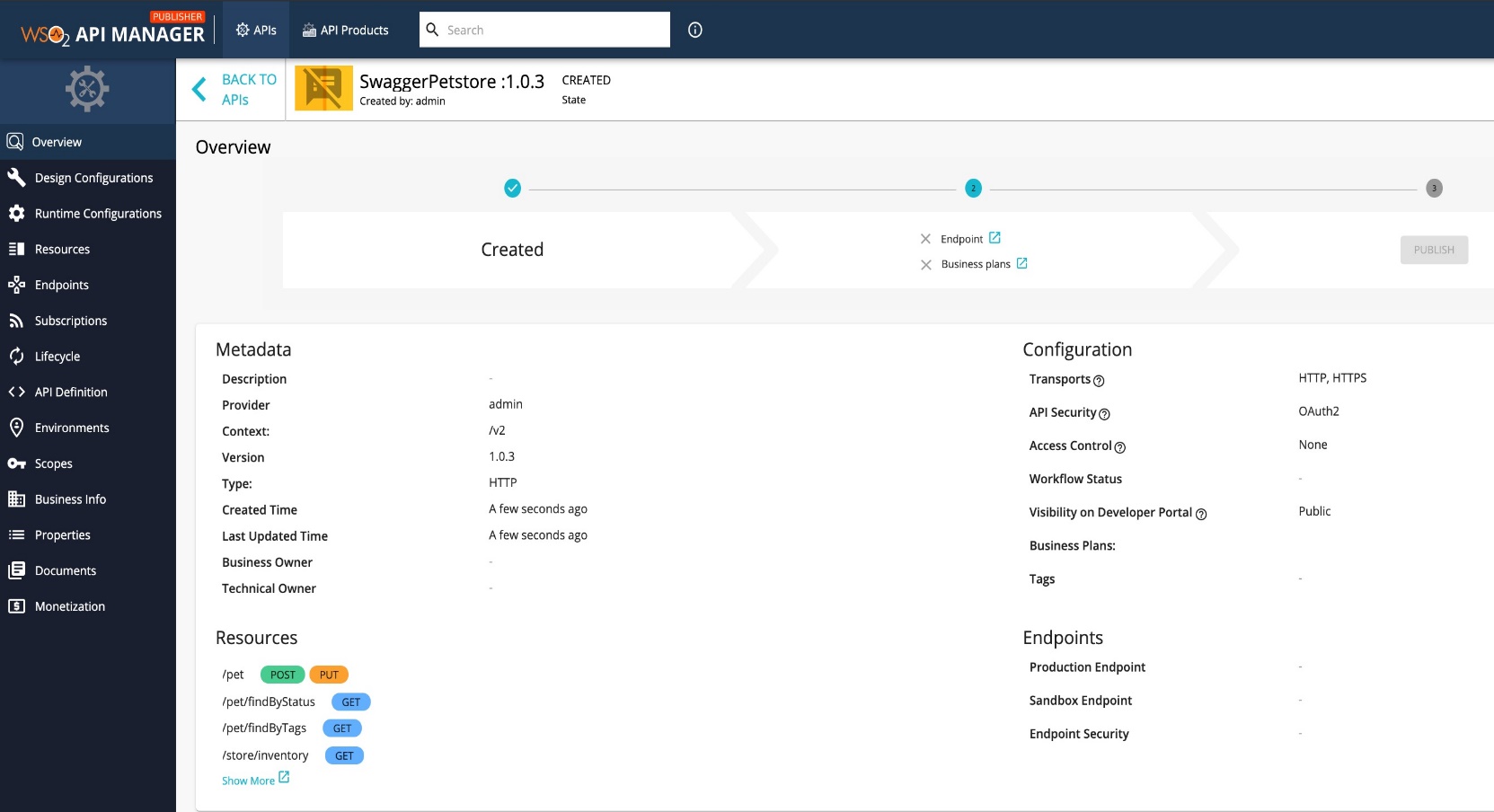
1. Chọn mục **OpenAPI URL và cung cấp URL là** <http://petstore.swagger.io/v2/swagger.json>

****

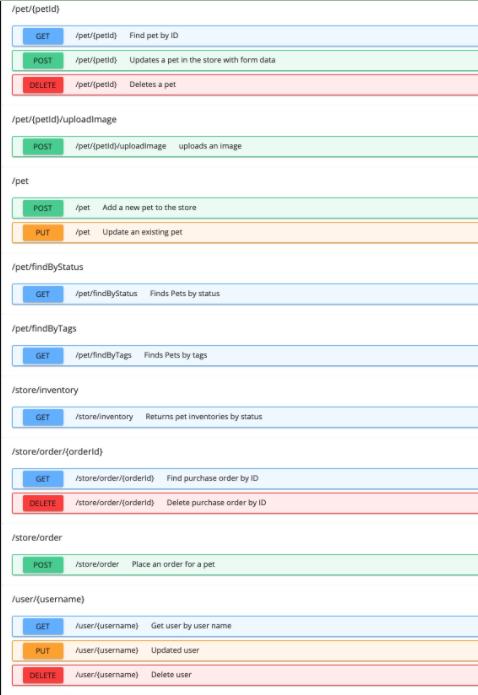
1. Cung cấp các thông tin khởi tạo sau đó chọn **Create**

****

1. Sau đó trang sẽ được chuyển đến trang tổng quan của  **Petstore API** (cái API mà chúng ta đang xây dựng).

****

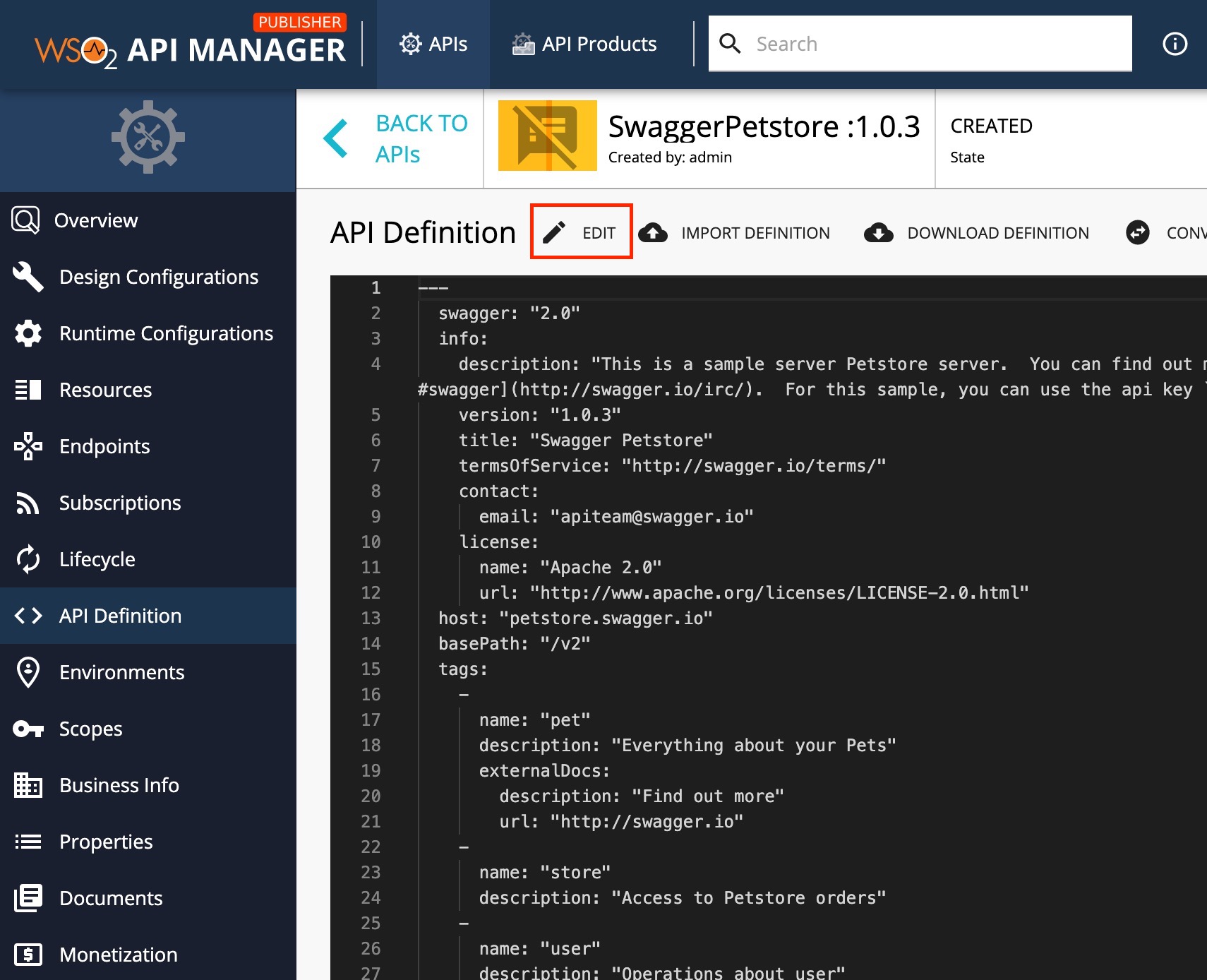
1. Chọn tab **Resource** để xem các Resource đã được tạo

****

**Tạo API từ Open API Definition**

**Các bước thực hiện**

1. Điều hướng đến **API Definition** và click chọn Edit để xóa các tiêu đề bảo mật. Điều này là bắt buộc để gọi API trong developer portal sử dụng Swagger UI.

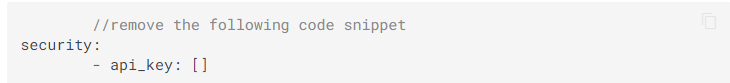
****

1. Xóa các security tag khỏi tài nguyên /pet POST :

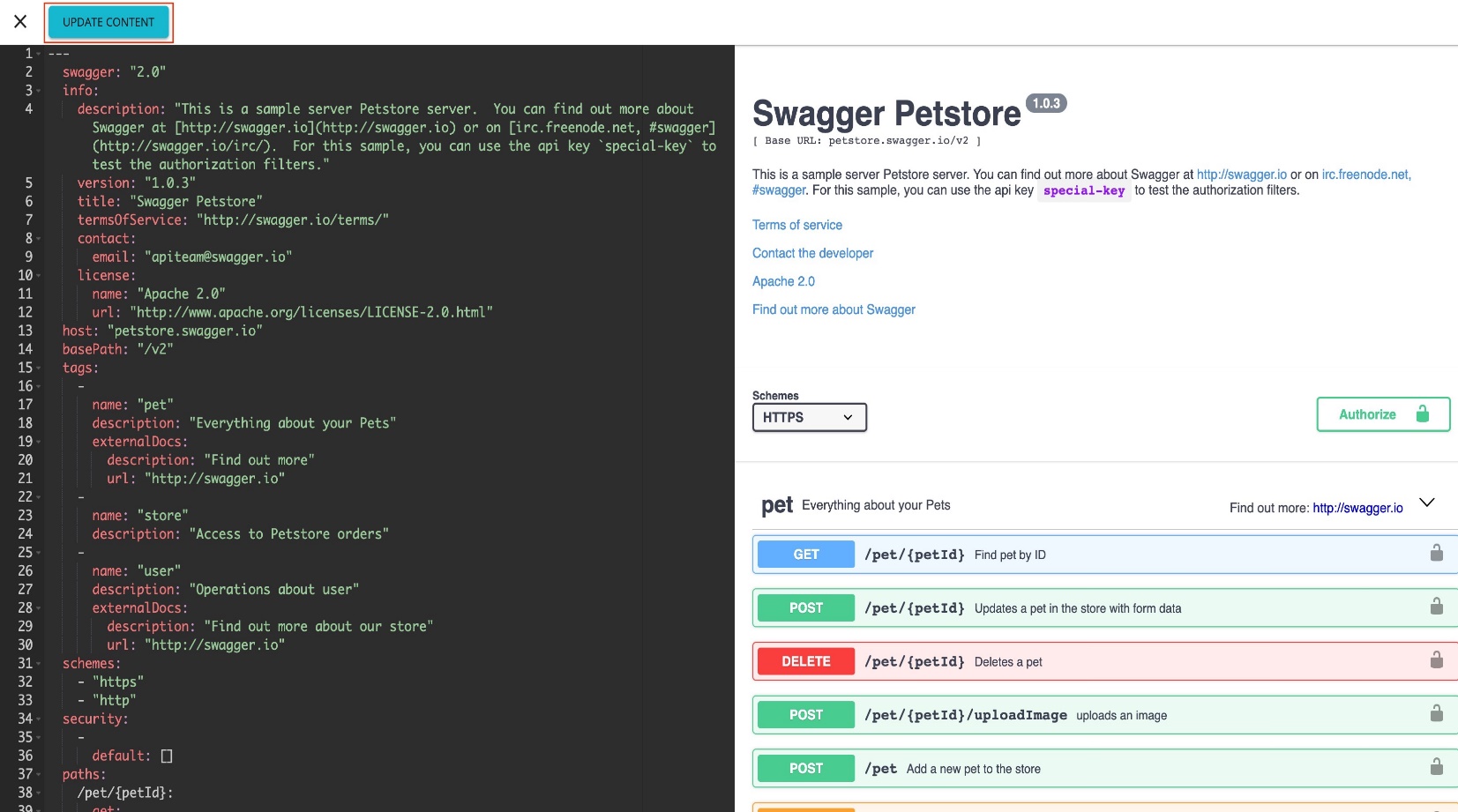
****

1. Xóa security pet/{petId} GET :

**Swagger - Get resource**

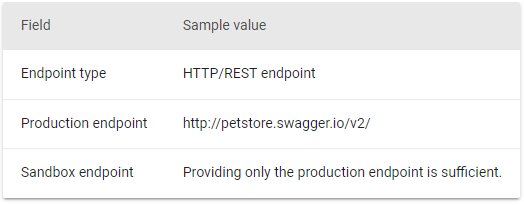
****

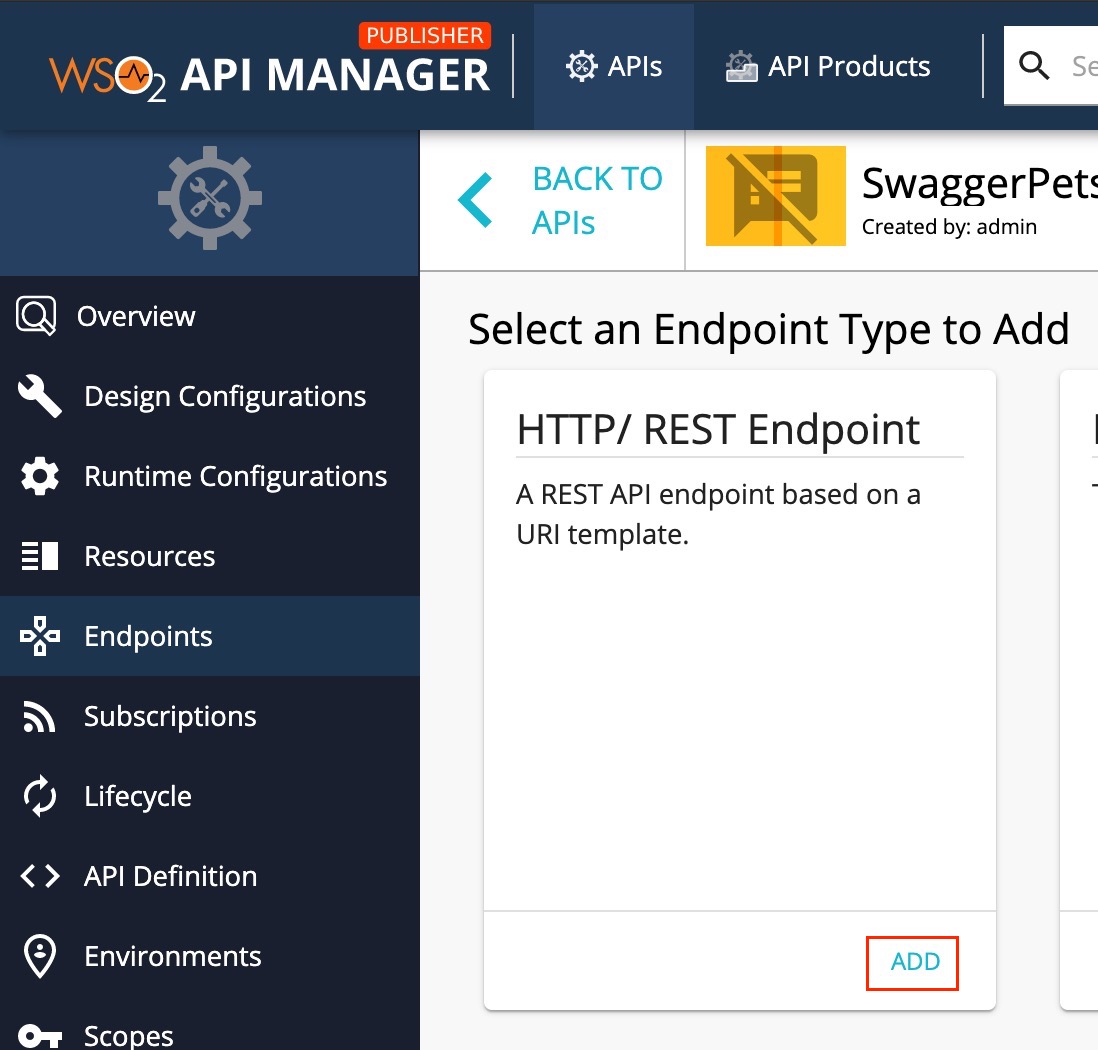
1. Sau khi xóa các security tags , click chọn **Update Contents** để lưu lại các thay đổi

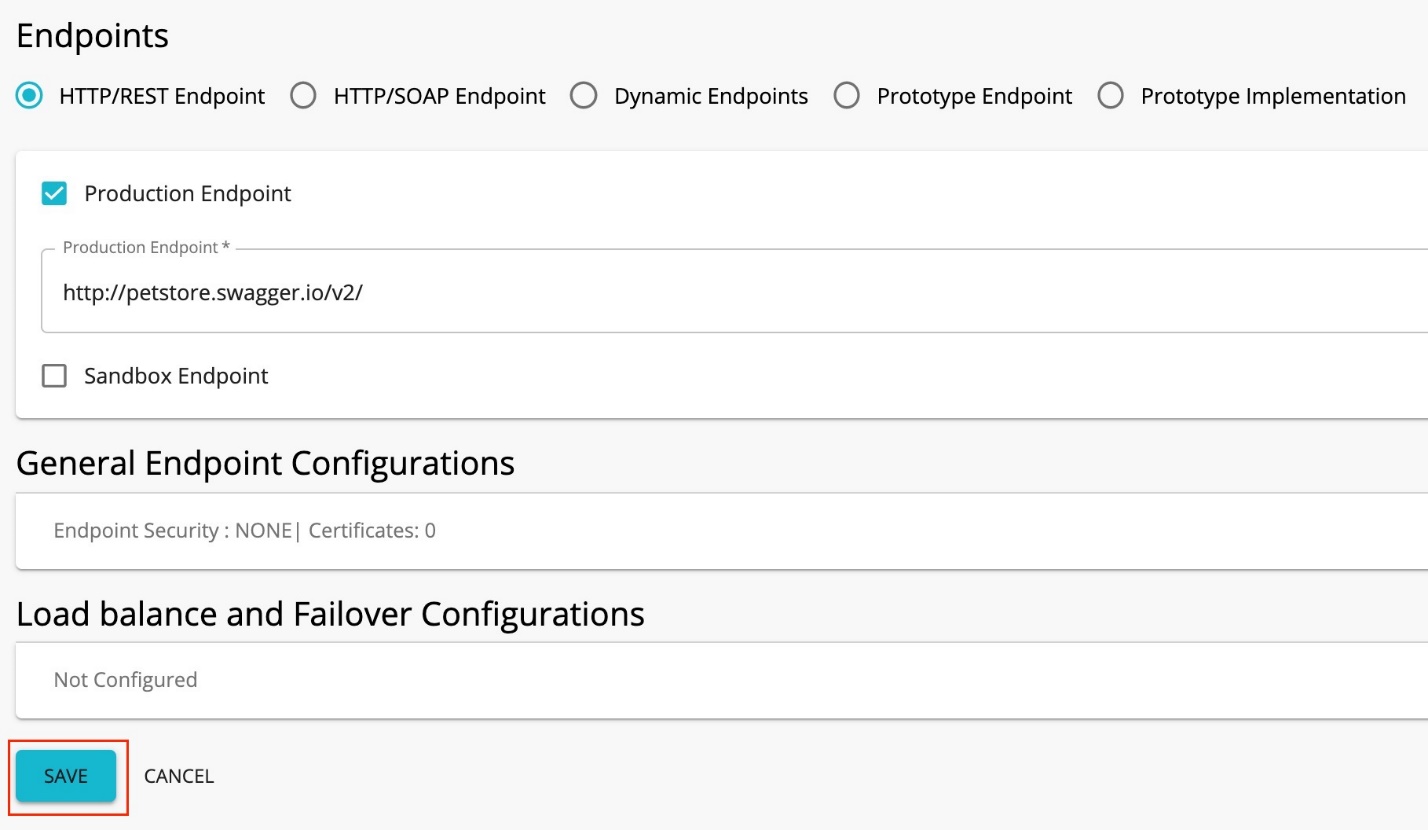
****

* *Thành phần* ***Endpoints***

Điều hướng đến trang **Endpoints** và tiến hành thêm các thông tin :

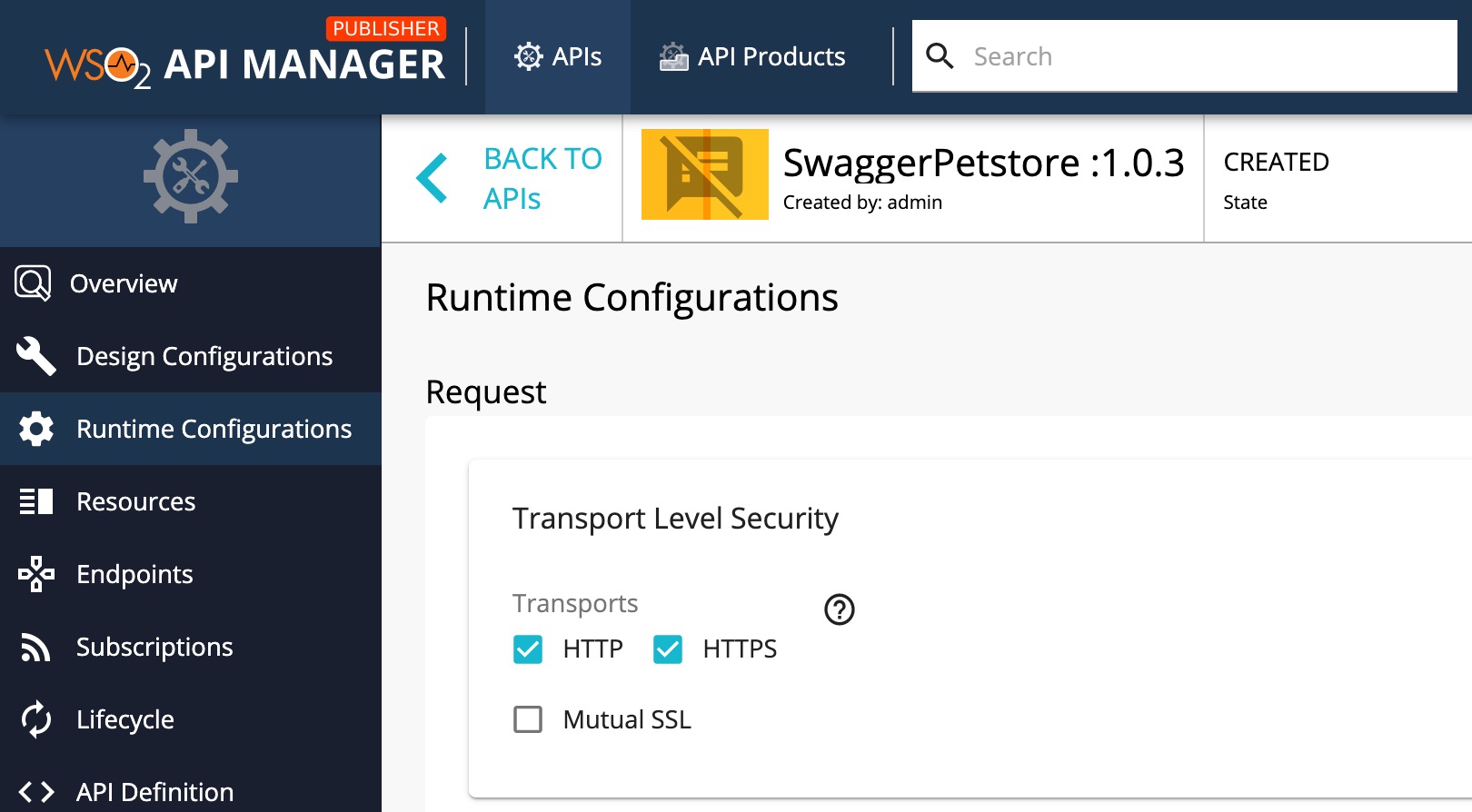






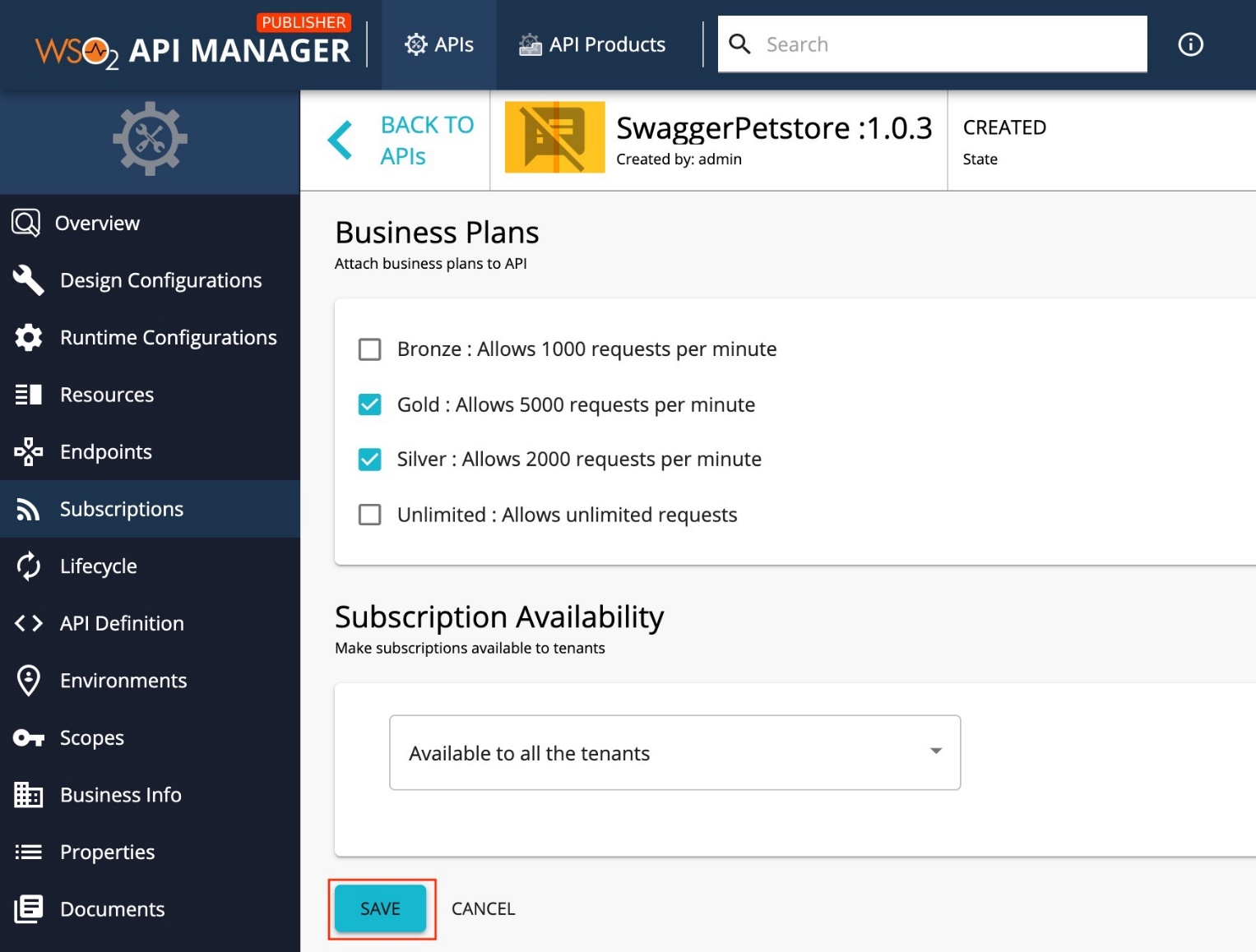
* *Thành phần* ***Runtime Configuration***

Chuyển hướng tới trang **Runtime Configuration**. Bảo mật mức truyền tải (Transport Level Security) xác định giao thức truyền tải, nơi mà API được exposed



* *Thành phần* ***Subscriptions***

Điều hướng đến trang **Subscriptions** sau đó chọn Gold and Silver rồi ấn lưu.



Bây giờ, một API REST từ Định nghĩa API mở đã được tạo và định cấu hình thành công

1. **So sánh với việc tự xây dựng code API sử dụng Zuul Proxy**
   1. **Zuul là gì ?**

Nó là một proxy, gateway và một lớp trung gian giữa user và các service của bạn. Eureka server đã giải quyết vấn đề đặt tên cho từng service thay vì dùng địa chỉ IP của chúng. Tuy nhiên một service vẫn có thể có nhiều instance và chúng sẽ chạy trên các cổng khác nhau. Do đó, nhiệm vụ của Zuul sẽ là:

* Map giữa một prefix path và một service (team-service). Nó sử dụng Eureka server để định tuyến các service được request.
* Giúp cân bằng tải giữa các instance của một service.
* Filter request, thêm xác thực …

Zuul Proxy là: Zuul Proxy là một phần của Zuul, một thư viện về routing và server load balancer của Netflix. Nó sẽ cấu hình service id của các service trong Eureka Server với context path tương ứng của service id đó, expose một URL để các ứng dụng bên ngoài có thể access tới, sau đó dựa vào context path mà các ứng dụng đang request tới để forward request tới đúng ứng dụng bên trong.

* 1. **Điểm khác biệt giữa xây dựng API sử dụng WSO2 API Manager và xây dựng API với Zuul**

Zuul Proxy cung cấp một cổng kỹ thuật cho phép chúng ta có các redundant (micro)services instances ,cân bằng tải, cấu hình tập trung bằng ServerConfig, v.v. Nó rất tốt .

Còn API Manager Gateway cung cấp cho bạn khả năng quản lý bảo mật: người dùng hoặc ứng dụng sẽ sử dụng dịch vụ của bạn (dưới dạng API) sẽ phải đăng ký với nó. Đổi lại, nó cho phép bạn cung cấp bảo mật Oauth2 cho người dùng. Và hơn thế nữa: bạn sẽ có dữ liệu sử dụng và số liệu theo từng người dùng (người dùng cuối hoặc ứng dụng). Lớp API Manager chỉ cần thiết nếu bạn phải quản lý việc sử dụng các dịch vụ của mình.

Do đó có thể không phải tất cả các microservices sẽ được coi là API được quản lý và bảo mật. Chỉ những người dành cho phù thủy mà bạn quan tâm mới có một công cụ quản lý như vậy

WSO2 API Manager là giải pháp cấp doanh nghiệp mã nguồn mở 100% hỗ trợ xuất bản API, quản lý vòng đời, phát triển ứng dụng, kiểm soát truy cập, giới hạn tỷ lệ và phân tích trong một hệ thống tích hợp rất cleanly.

Điểm khác biệt thứ nữa phải kể đến đó là về các xây dựng API gateway. Wso2 xây dựng các API sử dụng API Manager thường từ một Open API Definition cụ thể ở đây là swagger còn đối với việc xây dựng API sử dụng Zuul proxy thì chúng ta cần thêm các dependencies cụ thể là thêm **spring-cloud-starter-netflix-zuul** . Và một điều đáng nói nữa là Zuul hoạt động như một Eureka client. Vì vậy chúng ta có thể đặt tên cho nó, chỉ định port và đường dẫn đến Eureka server như các client trước đó. Cuối cùng là thêm các annotation để thông báo đây là Zuul và Eureka client. Cụ thể là các annotation như sau :



1. **Kết luận**

Qua bài báo cáo này em đã biết thêm được nhiều kiến thức về kiến trúc microservice , kiến thức về API Gateway ,hiểu thêm về cách thức xây dựng API gateway với WSO2 API Manager cũng như xây dựng một API gateway với Zuul. Biết rõ hơn về ưu điểm và nhược điểm của kiến trúc microservices cũng như ưu điểm và nhược điểm của API gateway và tại sao chúng ta nên sử dụng API Gateway trong việc xây dựng hệ thống của mình. Bên cạnh đó trong quá trình tìm hiểu và xây dựng bài báo cáo em cũng hiểu rõ hơn các bước xây dựng API gateway với WSO2 API Gateway opensource cũng như các thành phần có trong nó. Mặt khác em cũng phần nào nắm được các bước để xây dựng API gateway với Zuul. Hiểu được các dependencies cần thêm vào và các annotation cũng như chức năng của chúng.

Trong quá trình xây dựng bản báo cáo em cũng gặp một số vấn đề khó khăn , điển hình như đây là vấn đề khá mới đối với bản thân , tài liệu tìm kiếm phần đa là được viết bằng tiếng Anh và tiếng Nhật , hơn thế nữa kỹ năng xây dựng báo cào cùng với lời văn còn yếu mong nhận được sự góp ý nhiều hơn từ các anh chị đánh giá.

# **IV. Tài liệu tham khảo**

1. Tạo API Gateway với WSO2 Manager: <https://apim.docs.wso2.com/en/3.1.0/learn/design-api/create-api/create-a-rest-api-from-a-swagger-definition/>
2. Xây dựng API Gateway với Zuul

[**https://tubean.github.io/2018/12/microservice-springboot-eureka/**](https://tubean.github.io/2018/12/microservice-springboot-eureka/)

1. Tìm hiểu về mircroservices :

<https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>