**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP TỐT NGHIỆP**

**ĐỀ TÀI: TÌM HIỀU VỀ MICROSERVICES**

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: PHẠM VĂN TỊNH

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá :K57

Tp. Hồ Chí Minh, năm 2020

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP TỐT NGHIỆP**

**ĐỀ TÀI: TÌM HIỀU VỀ MICROSERVICES**

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: PHẠM VĂN TỊNH

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá :K57

Tp. Hồ Chí Minh, năm 2020

|  |  |
| --- | --- |
| TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI  PHÂN HIỆU TẠI TP. HỒ CHÍ MINH  **BỘ MÔN CÔNG NGHỆ THÔNG TIN**  --------------------------------------- | CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM  Độc lập – Tự do - Hạnh phúc  --------------------------------------------------- |

**NHIỆM VỤ THỰC TẬP TỐT NGHIỆP**

Họ và tên sinh viên: PHẠM VĂN TỊNH

MSSV: 5751071043

Chuyên ngành: Công nghệ Thông tin

Lớp: CQ.57.CNTT

1. Tên đề tài thực tập tốt nghiệp:

Tìm hiều về Microservices.

1. Nhiệm vụ thực tập tốt nghiệp:
   1. Tổng quan về đơn vị thực tập.
   2. Tìm hiều về Microservices.
   3. Nghiên cứu về hệ thống xác thực và phân quyền.
2. Ngày bắt đầu thực tập tốt nghiệp: ngày 10 tháng 02 năm 2020
3. Ngày hoàn thành báo cáo thực tập tốt nghiệp: ngày 30 tháng 07 năm 2020
4. Họ tên giáo viên hướng dẫn: Trần Phong Nhã
5. Tên công ty thực tập: Bệnh viện Đại Học Y Dược TPHCM

Địa chỉ: 215 Hồng Bàng, Phường 11, Quận 5, TPHCM

|  |  |
| --- | --- |
| **Trưởng bộ môn** | ***Tp. Hồ Chí Minh, ngày …. Tháng …..năm* …….**  **Giáo viên hướng dẫn**  **Trần Phong Nhã** |
|  | |

**LỜI CẢM ƠN**

Size 13

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN (size 15, bold)**

*(Cách 1 tab, Time newRoman, 20)*

Size 13

|  |
| --- |
| ***Tp. Hồ Chí Minh, ngày ….… tháng ….… năm ….…***  **Giáo viên hướng dẫn**  **Lê Nhật Tùng** |

**MỤC LỤC**

CHƯƠNG 1: TỔNG QUAN ………………………………………..…….………… 1

1.1. Tổng quan về nợ công…………………………………………………..……....... 1

1.1.1. Nợ công Việt Nam …………………………………………………………..… 2

**DANH MỤC CHỮ VIẾT TẮT**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Mô tả** | **Ý nghĩa** | **Ghi chú** |
|  | **GPS** |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ (size 15, bold)**

*(Cách 1 tab, Time newRoman, 20, mỗi nội dung trình bày bắt đầu từ 1 trang mới)*

Bảng 1.1: (size 13)…………………………………………………………………........

Bảng 1.2: ..........................................................................................................................

Sơ đồ 1.1:………………………………………………………………………………..

Hình 1.1: ………………………………………………………………………………..

**Ghi chú:**

* + Xếp sau trang Mục lục
  + Chữ số thứ nhất chỉ tên chương
  + Chữ số thứ hai chỉ thứ tự bảng biểu, sơ đồ, hình,…trong mỗi chương
  + Ở cuối mỗi bảng biểu, sơ đồ, hình,…trong mỗi chương phải có ghi chú, giải thích, nêu rõ nguồn trích hoặc sao chụp,…
  1. **CHƯƠNG I: MỞ ĐẦU**

1. **Tổng quan về Bệnh Viện Đại Học Y Dược**

Với 25 năm hình thành và phát triển, Bệnh viện Đại Học Y Dược TPHCM là địa chỉ chăm sóc sức khỏe uy tín của hàng triệu người bệnh.

Bệnh viện có 3 cơ sở, trụ sở chính tại 215 Hồng Bàng, Phường 11, Quận 5, TPHCM. Mỗi năm, Bệnh viện tiếp nhận trung bình hơn 2 triệu lượt người khám ngoại trú (khoảng 7.000 người khám/ngày), điều trị nội trú 55.000 người, phẫu thuật khoảng 30.000 trường hợp.

Là một Bệnh viện của trường đại học (Đại Học Y Dược TPHCM) cùng tầm nhìn đạt chuẩn quốc tế, bệnh viện phấn đấu nâng cao chất lượng chuyên môn và nâng cao chất lượng dịch vụ, đáp ứng mong đợi của người dân trong và ngoài nước. Bệnh viện Đại Học Y Dược TPHCM mong muốn mỗi người dân khi đến khám chữa bệnh luôn nhận được chất lượng dịch vụ y tế tốt nhất.

1. **Thông tin liên hệ**

- Trụ sở chính**:** 215 Hồng Bàng, Phường 11, Quận 5, TPHCM

- Điện thoại liên hệ: +84 (28) 3855 4269

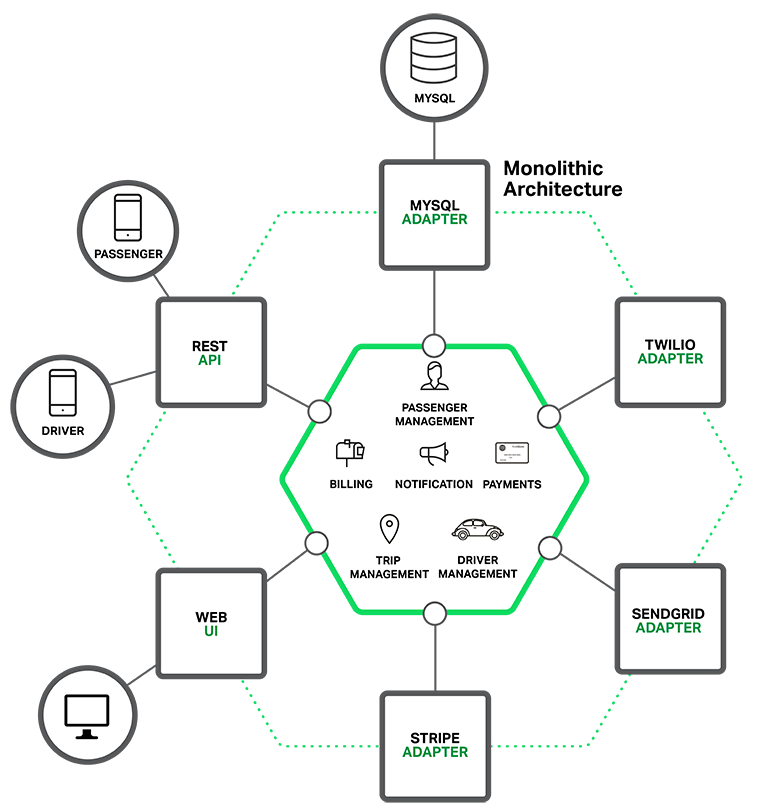
- Fax: +84 (28) 3950 6126

- Trang web: [www.bvdaihoc.com.vn](http://www.bvdaihoc.com.vn)

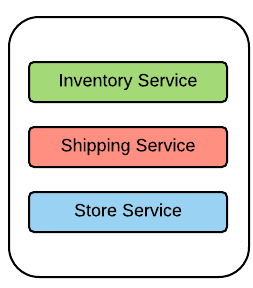
1. **Kết quả thu được**
   1. **CHƯƠNG II: MICROSERVICES**
2. **Giới thiệu**

Theo cách đơn giản thì Microservices thì đúng như cái tên của nó nghĩa là việc chia nhỏ ứng dụng lớn thành các ứng dụng nhỏ, mỗi ứng dụng đảm nhiệm một chức năng riêng biệt được kết nối với nhau một cách hài hòa để đáp ứng yêu cầu nghiệp vụ của sản phẩm. Trước tiên hãy xem lý do tại sao nên xem xét sử dụng Microservices.

* 1. ***Kiến trúc một khối (Monolithic Architecture)***



Một dịch vụ gọi taxi qua di động, với logic kinh doanh được thể hiện bởi các khối dịch vụ, đối tượng cho từng vùng nghiệp vụ (domain objects) và các sự kiện (events: khách đặt xe, hủy xe, xe nhận khách,.. ) Xung quanh lõi và bộ chuyển đổi (adapter) ví dụ như kết nối vào cơ sở dũ liệu, gửi nhận thông điệp, web service,…

Trong nhiều trường hợp, người ta có thể xây dựng các services độc lập nhưng chúng lại được triển khai chung. Mặc dù có cấu trúc module hóa hợp lý, nhưng ứng dụng kiểu này sẽ đóng gói và cài đặt thành một khối. Mã chạy cụ thể tùy thuộc vào ngôn ngữ lập trình hay thư viện framework. 

Tương tự như ứng dụng monolithic, những services này to và phức tạp lên theo thời gian vì thường xuyên thêm các tính năng. Và thế là những ứng dụng này lại trở thành một mớ các services monolithic, cũng không còn khác mấy so với kiến trúc một khối thông thường. Hình trên thể hiện một ứng dụng gồm nhiều services. Những services này được triển khai cùng một lúc vào 1 ứng dụng lớn. Dù bên trong gồm các services thì đây vẫn là một ứng dụng monolithic. Một số tính chất của kiến trúc một khối:

* + Được thiết kế, phát triển và triển khai theo một khối duy nhất.
  + Ứng dụng monolithic phức tạp và to gây khó khăn trong việc bảo trì, nâng cấp và thêm tính năng mới.
  + Khó áp dụng phát triển theo kiểu Agile.
  + Phải triển khai lại hệ thống dù chỉ cập nhật hay nâng cấp một phần.
  + Mở rộng: phải mở rộng cả khối ứng dụng, gặp khó khăn nếu có các yêu cầu về tài nguyên khác nhau (ví dụ một service yêu cầu thêm tốc độ xử lý, service khác lại yêu cầu nhiều dung lượng bộ nhớ).
  + Độ tin cậy: ứng dụng monolithic phải sử dụng chung công nghệ nên khó thay đổi và áp dụng công nghệ mới.

Những tính chất giới hạn trên của kiến trúc Monolithic dẫn tới sự phát triển của kiến trúc Microservices.

* 1. ***Kiến trúc dịch vụ nhỏ (Microservices Architecture)***

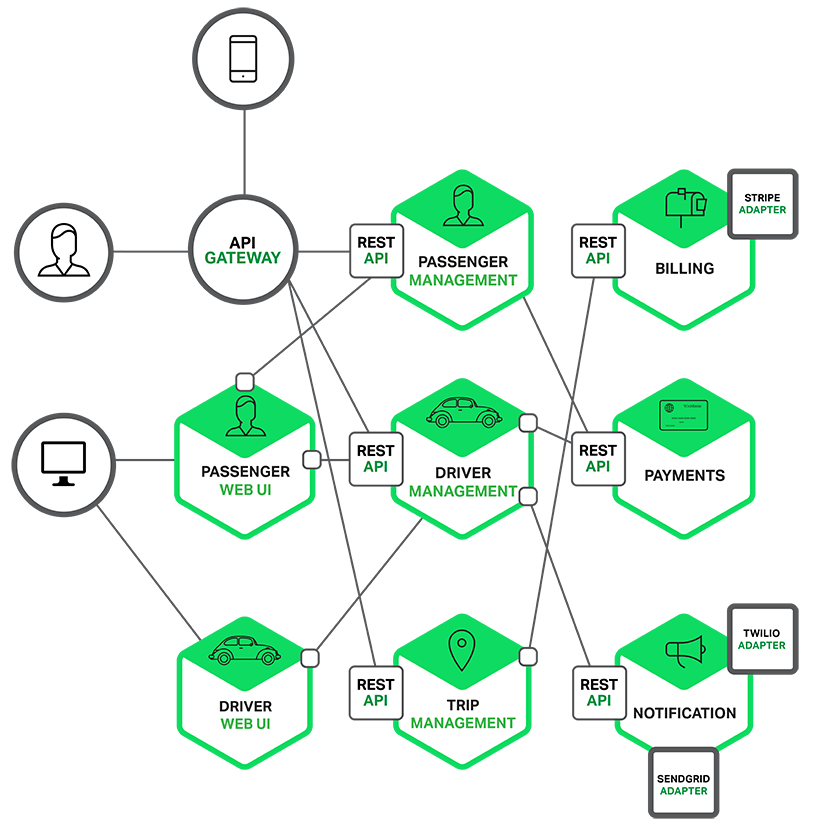
Nền tảng của kiến trúc Microservices là xây dựng một ứng dụng mà ứng dụng này là tổng hợp nhiều service nhỏ và độc lập có thể chạy riêng biệt, phát triển và triển khai độc lập.

Ta có thể giải quyết các vấn đề của ứng dụng một khối bằng kiến trúc Microservices (nhiều dịch vụ nhỏ). Ý tưởng chia nhỏ ứng dụng lớn ra thành các dịch vụ nhỏ kết nối với nhau.

Mỗi dịch vụ nhỏ thực hiện một tập các chức năng chuyên biệt như quản lý đơn hàng, quản lý khách hàng. Mỗi dịch vụ là một ứng dụng nhỏ có kiến trúc đa diện lõi là business logic kết nối ra các adapter khác nhau. Một số dịch vụ nhỏ lộ ra giao tiếp lập trình API cho dịch vụ nhỏ khác hay ứng dụng client gọi tới. Khi vận hành, mỗi dịch vụ nhỏ được chạy trong máy ảo hoặc Docker Container.

Mỗi vùng chức năng giờ được thực thi bỏi một dịch vụ nhỏ. Ứng dụng web cũng có thể chia nhỏ hơn chuyên cho từng đối tượng người dùng (hành khách, tài xế). Thiết kế giao diện cho từng đối tượng người dùng giúp tối ưu trải nghiệp tốt hơn, tốc độ nhanh hơn, dễ dàng tương thích hơn trong khi chức năng tối giản hơn.

Một số khái niệm về Microservices nói về quá trình chia tách ứng dụng monolithic thành nhóm các services độc lập. Tuy nhiên, Microservices không đơn giản là chỉ về chia tách các services sẵn có trong monolithic.



Điều quan trọng chính là nhìn vào các tính năng trong một ứng dụng monolithic, ta có thể nhận biết, xác định yêu cầu và khả năng cần thiết để đáp ứng một nghiệp vụ. Sau đó từng nghiệp vụ này sẽ được xây dựng thành các services nhỏ, độc lập. Những services này có thể sử dụng các nền tảng công nghệ khác nhau và phục vụ một mục đích cụ thể và có giới hạn.

1. **Ưu điểm và nhược điểm của Microservices**
   1. ***Ưu điểm***

Cho phép dễ dàng phát triển và triển khai các ứng dụng lớn, phức tạp:

Cải thiện khả năng bảo trì – mỗi service tương đối nhỏ do đó dể hiểu và thay đổi hơn.

Khả năng kiểm tra/kiểm thử dễ dàng hơn. – các services nhỏ hơn và nhanh kiểm thử hơn.

Khả năng triển khai tốt hơn – các services có thể được triển khai độc lập.

Cho phép các services được phát triển, thử nghiệm, triển khai và mở rộng quy mô dịch vụ của mình một các độc lập với tất cả các nhóm phát triển khác.

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 khiến kiến trúc một khối monolithic có thể làm ảnh hưởng toàn bộ hệ thống.

Dễ dàng thay đổi sử dụng các công nghệ mới: Khi triển khai các services nhà phát triển có thể lựa chọn nhiều công nghệ mới. Tương tự khi có thay đổi lớn với các services hiện có và có thể dễ dàng thay đổi công nghệ.

* 1. ***Nhược điểm***

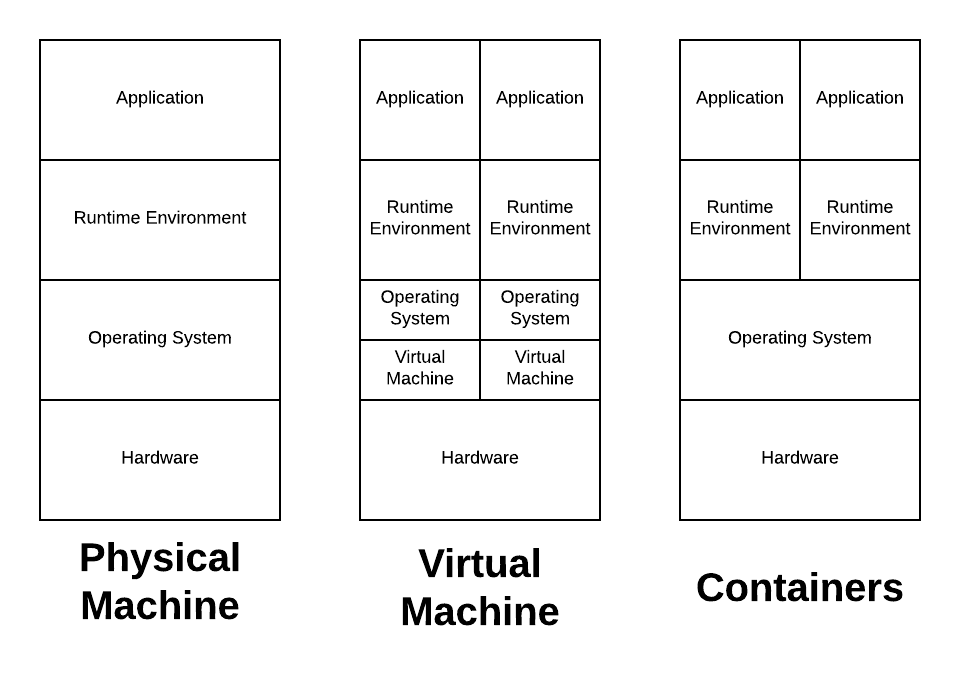
Các nhà phát triển phải đối phó với sự phức tạp của việc tạo ra một hệ thống phân tán:

* + Cần triển khai thực hiện các giao tiếp giữa các services.
  + Thiết kế và xử lý một luồng xử lý rất phức tạp. Vì một luồng xử lý có thể đi qua nhiều services.
  + Việc thực hiện các yêu cầu trải rộng trên nhiều services khó khan hơn, điều này đòi hỏi sự phối hợp cẩn thận giữa các nhóm phát triển.
  + Khó khan trong việc đảm bảo toàn vẹn CSDL nếu triển khai theo kiến trúc cơ sở dữ liệu phân vùng.

Phải xử lý sự cố khi kết nối chậm, lỗi thông điệp không gửi được hoặc thông điệp gửi đến nhiều đích đến vào các thời điểm khác nhau.

Triển khai và quản lý các Microservices nếu làm thủ công theo cách đã làm với những ứng dụng một khối khó khan hơn rất nhiều.

1. **Docker với Microservices**
   1. ***Sự phát triển của công nghệ trong hỗ trợ xây dựng cơ sở hạ tầng CNTT***

Bên cạnh sự phát triển của các kiến trúc phần mềm, một số công nghệ mới xuất hiện như Docker và Kubernetes để hỗ trợ cơ sở hạ tầng phần mềm và đảm bảo quản lý hiệu quả các sản phẩm và dịch vụ có thể mở rộng. Nhà phát triển các công nghệ mới trên đã phát triển từ việc sử dụng Hardware Virtualization để chưa các Container. 

Hình ảnh đầu tiên cho thấy một máy chủ vật lý hoặc máy chủ phần cứng. Thông thường, khi nhà phát triển xây dựng các ứng dụng, họ sử dụng các tài nguyên được cung cấp bởi hệ điều hành máy chủ và một mẫu có sẵn để triển khai ứng dụng. Nhưng nếu muốn mở rộng quy mô ứng dụng thì sao? Tại một số thời điểm , họ cần thêm một máy chủ phần cứng mới và số lượng tiếp tục tăng, thì chi phí là một điều đáng lo ngại.

Điều này đã dẫn đến sự phát triển ảo hóa phần cứng để tối ưu hóa các cơ sở hạ tầng CNTT thông qua một thứ mà chúng ta gọi là Máy ảo (Virtual Machine). Trong sơ đồ thứ hai, máy ảo có nhiều hệ điều hành khách trên một máy ảo vật lý (hệ điều hành máy chủ). Điều này cho phép nhà phát --triển chạy nhiều ứng dụng mà không cần đặt nhiều máy vật lý. Hệ điều hành máy chủ có thể đảm bảo rằng có sự phân phối tài nguyên có hệ thống và cân bằng tải giữa các Virtual Machine khác nhau chạy trên nó.

Mặc dù Virtual Machine làm cho phần mềm dễ truy cập hơn để duy trì và giảm đáng kể chi phí, vẫn có thể tối ưu hóa nhiều hơn. Chẳng hạn, không phải tất cả các ứng dụng sẽ hoạt động như mong đợi trong môi trường Hệ điều hành khách. Ngoài ra, Hệ điều hành khách sẽ yêu cầu rất nhiều tài nguyên thậm chí để chạy các tiến trình đơn giản.

Những vấn đề trên dẫn đến sự đổi mới tiếp theo: container hóa. Không giống như các máy ảo có hệ điều hành cụ thể, các container là ứng dụng cụ thể, làm cho chúng nhẹ hơn rất nhiều. Hơn nữa, Virtual Machine có thể chạy nhiều tiến trình trong khi một container chạy như một tiến trình đơn lẻ:

* + Có thể chạy nhiều Container trên một máy vật lý hoặc thậm chí có thể chạy nó trên máy ảo. Trong cả hai trường hợp, container giải quyết các vấn đề liên quan đến ứng dụng của bạn.
  + Containerization không cạnh tranh với Virtualization, mà là một yếu tố bổ sung thêm để tối ưu hóa hơn nữa cơ sở hạ tần CNTT.
  1. ***Docker***

Docker là nền tảng Container ảo hóa phần mềm hàng đầu thế giới. Nó gói gọn ứng dụng Microservices vào Docker Container có thể được duy trì và triển khai một cách độc lập. Mỗi trong số container này sẽ chịu trách nhiệm cho một chức năng kinh doanh cụ thể (services).

Với Docker, nhà phát triển có thể làm ứng dụng của mình độc lập với môi trường máy chủ. Với kiến trúc Microservices, giờ đây nhà phát triển có thể gói gọn từng service trong các container Docker. Docker container là các môi trường cô lập tài nguyên nhẹ, thông qua đó nhà phát triển có thể xây dựng, bảo trì, vận chuyển và triển khai ứng dụng.

Ưu điểm:

* + Docker là một phần mềm phổ biến với sự hỗ trợ công động mã nguồn mở và cộng đồng Microservices.
  + Docker rất nhẹ khi so sánh với các máy ảo làm cho nó hiệu quả về chi phí và tài nguyên.
  + Docker cung cấp tính đồng nhất trên các môi trường phát triển và sản xuất làm cho nó phù hợp để xây dựng các ứng dụng gốc trên đám mây.
  + Docker cung cấp các cơ sở để tích hợp và triển khai liên tục.

Vì thế, Docker là sự tối ưu hóa rất cần thiết cho Microservices.

1. **Một số điều cần lưu ý khi thiết kế Microservices**

Trong quá trình thiết kế, ta nên xác định và giới hạn các services theo chức năng và nghiệp vụ thực tế.

Đảm bảo Microservices có thể phát triển và triển khai độc lập.

Mục tiêu của thiết kế là phạm vi của Microservices phục vụ một nghiệp vụ chứ không chỉ đơn giản làm các dịch vụ nhỏ hơn. Kích thước hợp lý của một service là kích thước đầy đủ để đáp ứng yêu cầu của một chức năng của hệ thống.

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, các nhà phát triển thường không gặp 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. Đây là một vấn đề lớn đối với start-up vì họ cần phát triển nhanh mô hình kinh doanh cùng ứng dụng đi kèm.

Vì vây, trừ khi có một hệ thống quá phức tạp để quản lý bằng Monolithic Architecture, hoặc xác định tương lai của ứng dụng trở nên như vậy. Thì kiến trúc Monolic vẫn đủ tốt.

* 1. **CHƯƠNG III: SINGLE SIGN-ON**

1. **Giới thiệu**



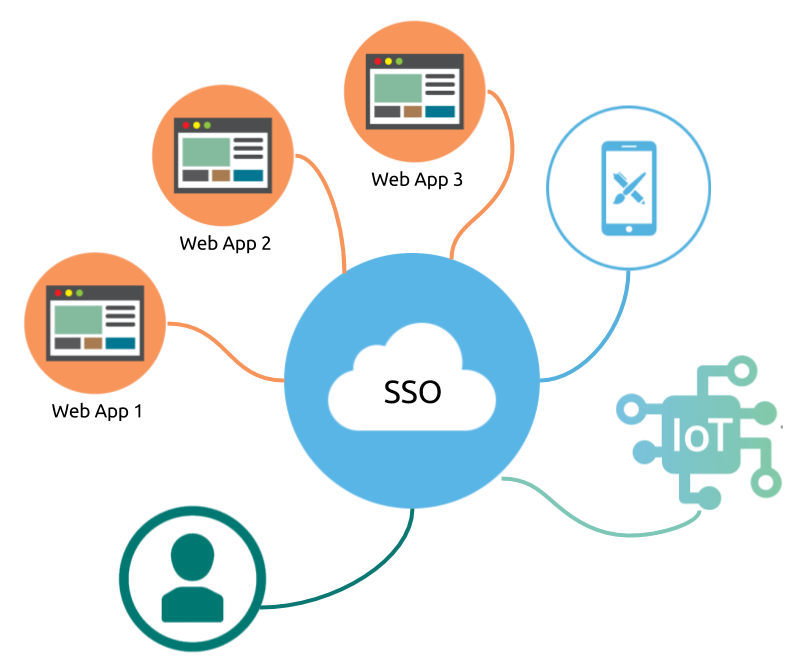
Đăng nhập một lần (Single Sign-On) SSO là dịch vụ xác thực phiên đăng nhập và người dùng cho phép người dùng cuối nhập một bộ thông tin đăng nhập (có thể gồm tên và mật khẩu) để có quyền truy cập vào ứng dụng.

Trong dịch vụ web SSO cơ bản, module agent trên máy chủ ứng dụng sẽ truy xuất thông tin xác thực cho từng người dùng từ máy chủ SSO chuyên dụng, đồng thời xác thực chéo người dùng qua kho lưu trữ người dùng dưới dạng thư mục LDAP (Lightweight Directory Protocol). Dịch vụ xác thực người dùng cuối cho tất cả các ứng dụng mà người dùng đã được cấp quyền và loại bỏ lời nhắc nhập mật khẩu tiếp theo cho các ứng dụng riêng lẻ trong cùng một phiên.

1. **Cách thức hoạt động**
   1. ***Cơ sở lý thuyết***

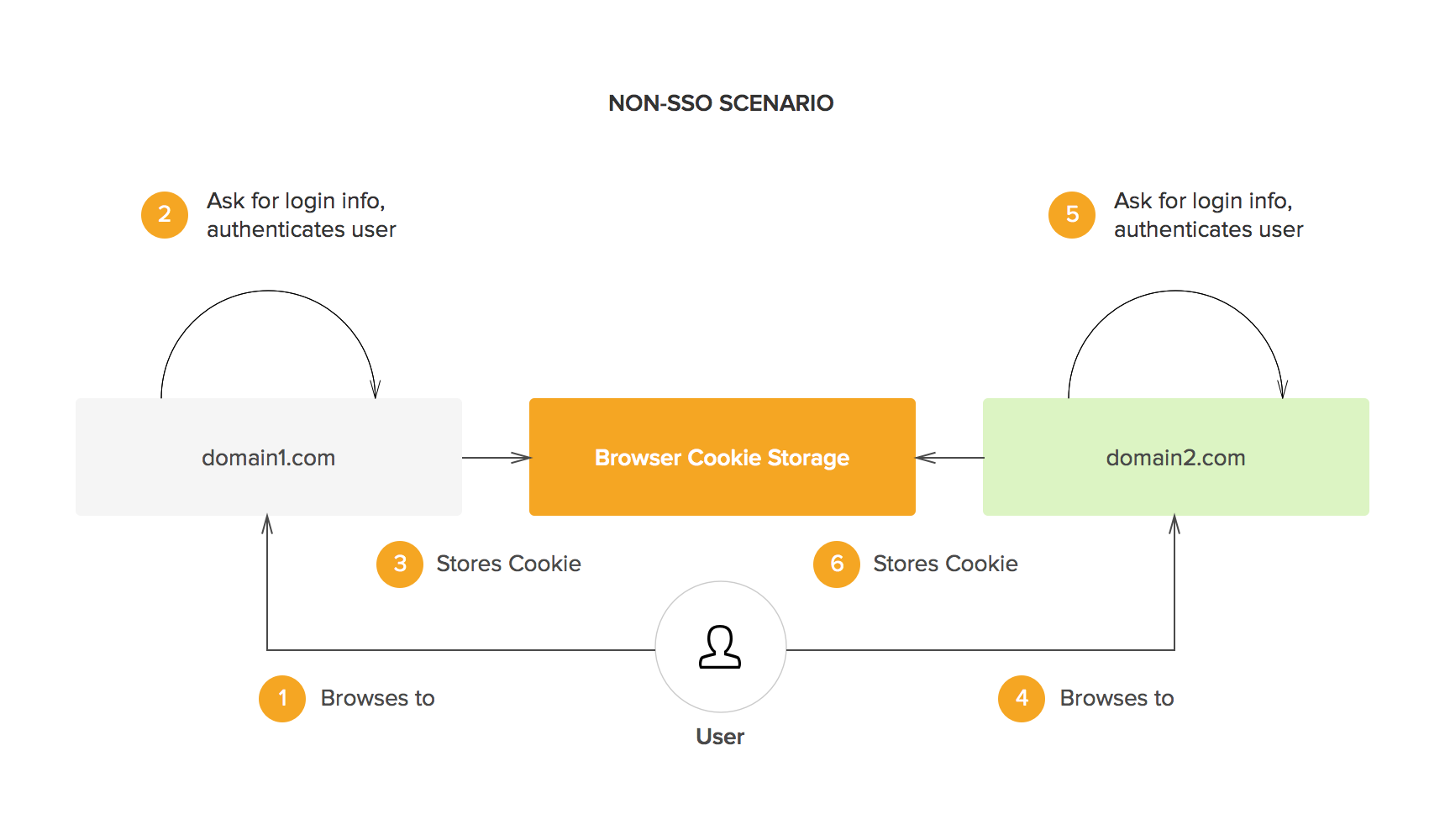
Single SignOn là một hệ thống quản lý danh tính liên kết hay liên kết danh tính. OAuth là nền tảng framework cho phép các dịch vụ bên thứ ba sử dụng thông tin tài khoản của người dùng cuối, chẳng hạn như Facebook, Google,…mà không để lộ mật khẩu của người dùng.

OAuth hoạt động như một trung gian đại diện cho người dùng cuối thông qua một mã Token truy cập, mã này cho phép thông tin tài khoản cụ thể được chia sẻ. Khi người dùng cố gắng truy cập một ứng dụng từ nhà cung cấp dịch vụ, nhà cung cấp dịch vụ sẽ gửi yêu cầu đến bên thứ 3 để xác thực. Nhà cung cấp dịch vụ sau đó sẽ xác minh, xác thực và cho phép người dùng đăng nhập.

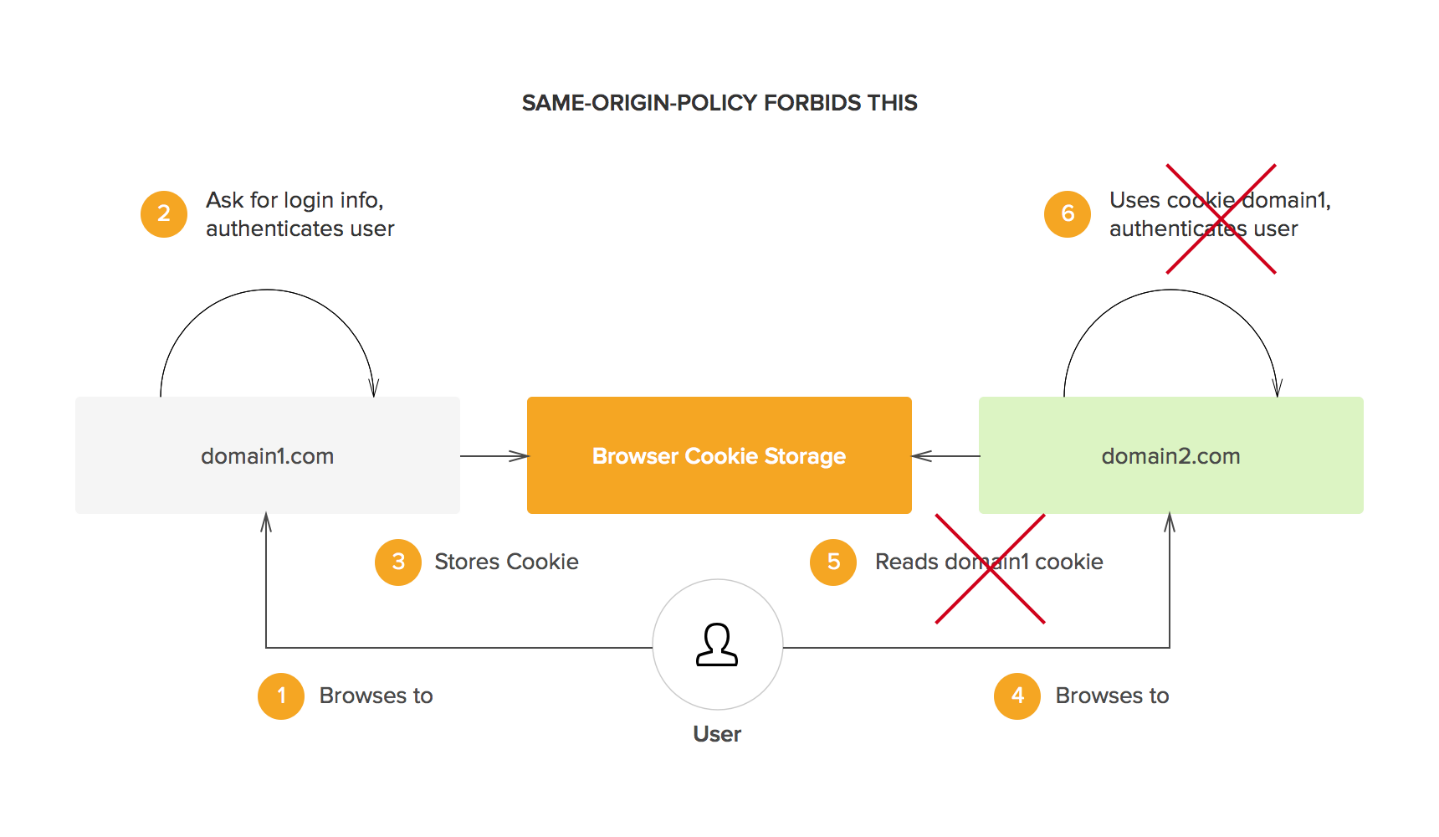


* 1. ***Cách thức hoạt động***

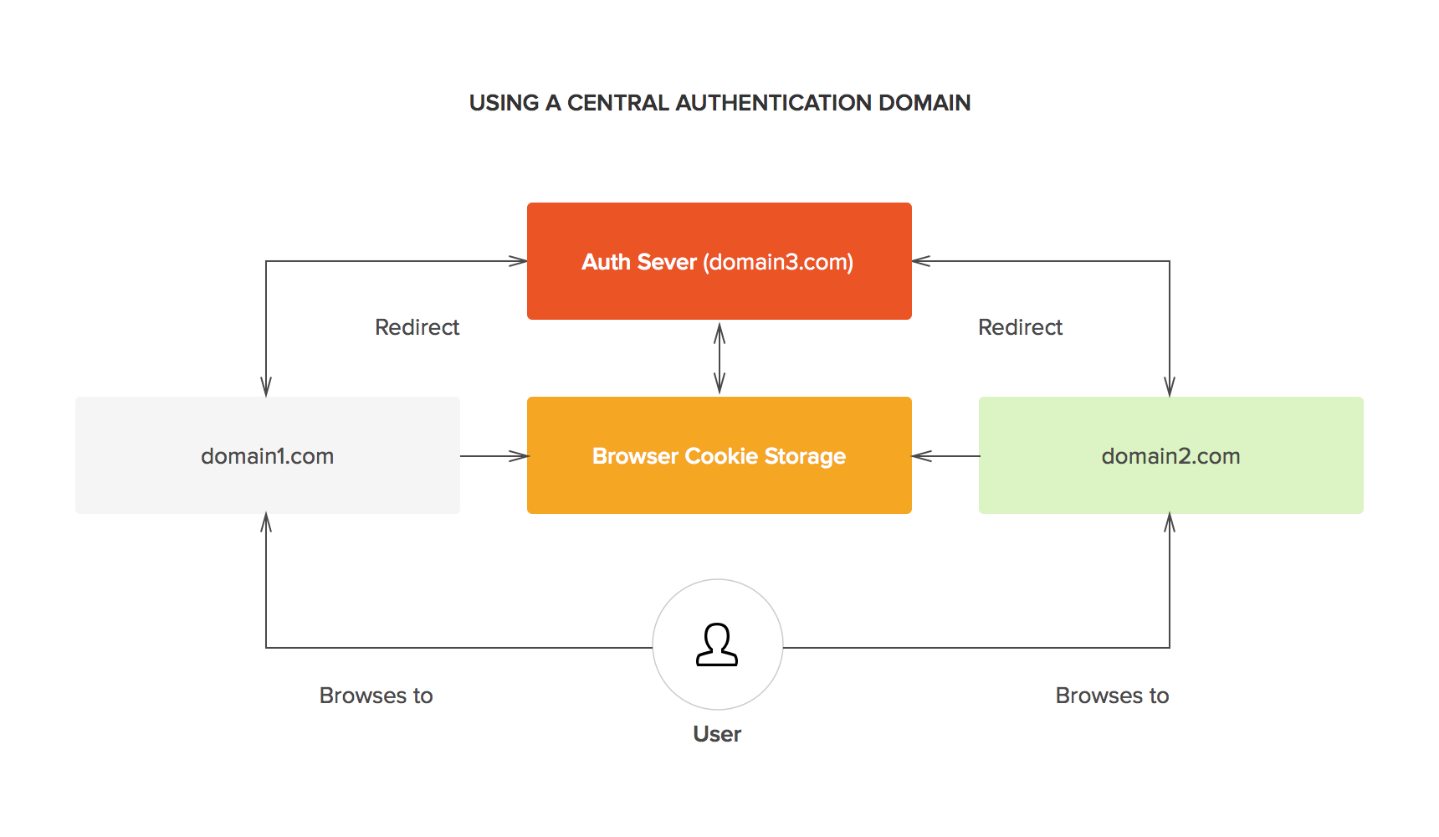
Các nhà phát triển web sẽ đối mặt với một vấn đề: phát triển một ứng dụng tại domain X và bây giờ muốn phát triển một ứng dụng mới tại domain Y sử dụng các thông tin đăng nhập giống với domain X. Nhưng thực tế, các nhà phát triển mong muốn nhiều hơn thế: nếu người dùng đã đăng nhập tại domain X thì họ cũng sẽ tự động tại domain Y. Đây là thứ SSO (Single-Sign On) giải quyết.



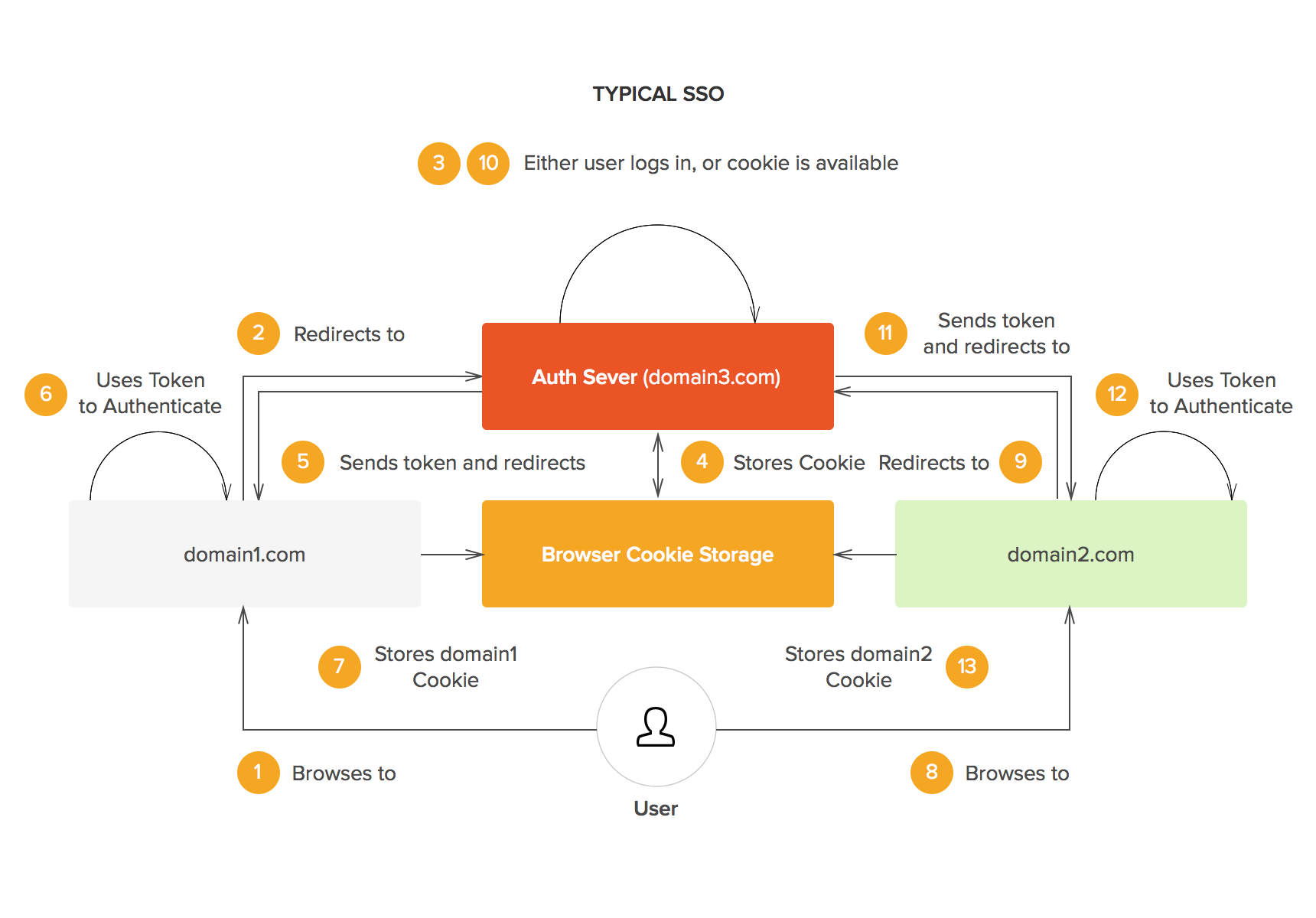
Đầu tiên, giải pháp cho kịch bản trên là chia sẻ thông tin session giữa các domain. Tuy nhiên, vì lý do bảo mật, các trình duyệt buộc phải tuân theo chính sách *same origin policy.* Nội dung cơ bản của chính sách này là **chỉ những người tạo ra mới có quyền truy cập** các cookie (hya bất kỳ dữ liệu cục bộ nào). Nói cách khác, domain X không thể truy cập các cookie từ domain Y và ngược lại. Đây là vấn đề mà SSO giải quyết: chia sẻ thông tin session trên nhiều domain khác nhau.



Các giao thức SSO chia sẻ thông tin session theo nhiều cách khác nhau, nhưng những thứ cơ bản thì giống nhau đó là: domain trung tâm để thực hiện xác thực (authentication) và sau đó session được chia sẻ với các domain khác theo nhiều cách. Ví dụ: domain trung tâm có thể tạo ra một Json Web Token (JWT) đã được đăng ký. Token này có thể được truyền tới client và được sử dụng để xác thực người dùng cho domain hiện tại cũng như bất kỳ domain nào khác. Token có thể truyền tới domain gốc bằng cách điều hướng và chứa tất cả các thông tin cần thiết để xác minh người dùng cho domain đang xác thực. Khi một token được đăng ký thì nó không thể bị chỉnh sửa bởi bất kỳ client nào.



Bất cứ khi nào người dùng tới một domain yêu cầu xác thực, họ sẽ được chuyển đến domain xác thực (authentication domain). Nếu người dùng đăng nhập tại domain xác thực, người dùng sẽ ngay lập tực được chuyển hướng trở lại domain gốc với token để xác thực các request tiếp theo.



1. **Rủi ro bảo mật**

Mặc dù đăng nhập một lần là tính năng rất tiện lợi đối với người dùng, nhưng hiện tại tiềm ẩn nhiều rủi ro cho bảo mật doanh nghiệp. Kẻ tấn công khi dành quyền kiểm soát thông tin đăng nhập SSO của người dùng sẽ có quyền truy cập vào mọi ứng dụng mà người dùng có thể truy cập, dẫn đến gia tăng mức độ thiệt hại tiềm tàng. Để tránh các truy cập độc hại, điều cần thiết là toàn bộ các yếu tố triển khai SSO cần phải được kết hợp với quản trị danh tính. Các tổ chức cũng có thể sử dụng xác thực hai yếu tố (2FA) hoặc xác thực đa yếu tố (MFA) kết hợp với SSO để cải thiện bảo mật.

1. **Ứng dụng của SSO**

Google, Linkedln, Facebook,… điều cung cấp các dịch vụ SSO phổ biến cho phép người dùng cuối đăn nhập vào ứng dụng của bên thứ ba trực tiếp từ tài khoản social. Mặc dù phương thức này là một tiện ích cực kỳ tiện lợi, nó lại có thể gây ra rủi ro bảo mật nghiêm trọng.

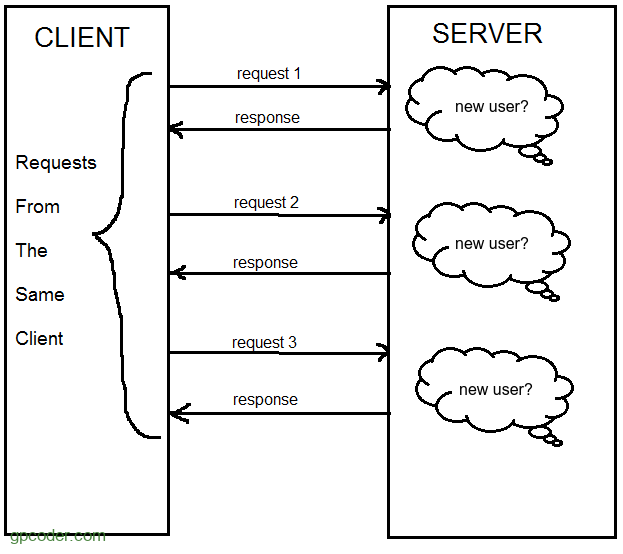
* 1. **CHƯƠNG IV: XÁC THỰC VÀ PHÂN QUYỀN**

1. **Xác thực (Authentication)**

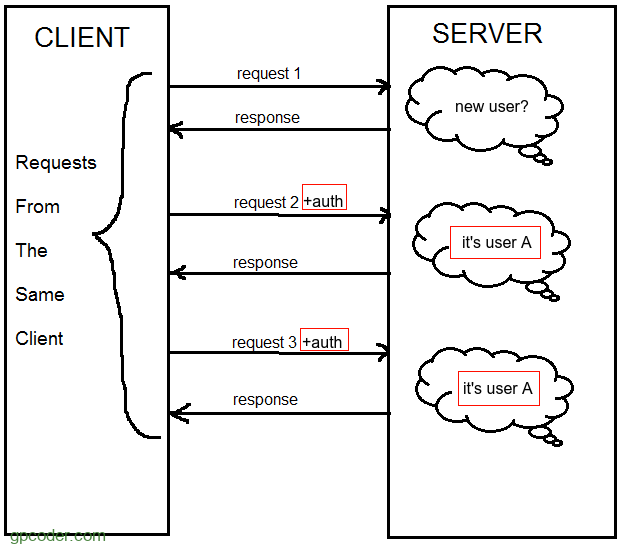
Xác thực/ Định danh (authentication) là hành động nhằm thiết lập hoặc chứng thực một cái gì đó (hoặc một người nào đó) đáng tin cậy, có nghĩa là những lời khai báo do người dùng đó đưa ra hoặc về vật đó là sự thật.

Như đã biết, RESTful web service sử dụng HTTP protocol như là một phương tiện giao tiếp và HTTP request là stateless protocol. Tức là server không lưu giữ bất kì thông tin nào của client, server xử lý các request một cách độc lập, không phụ thuộc vào trạng thái hay kết quả của request trước.

Như hình bên dưới, do server không lưu giữ bất kỳ thông tin nào của request trước. Nên mỗi request gửi lên server đều phải chứng thực lại, mặc dù request của cùng một user thực hiện.

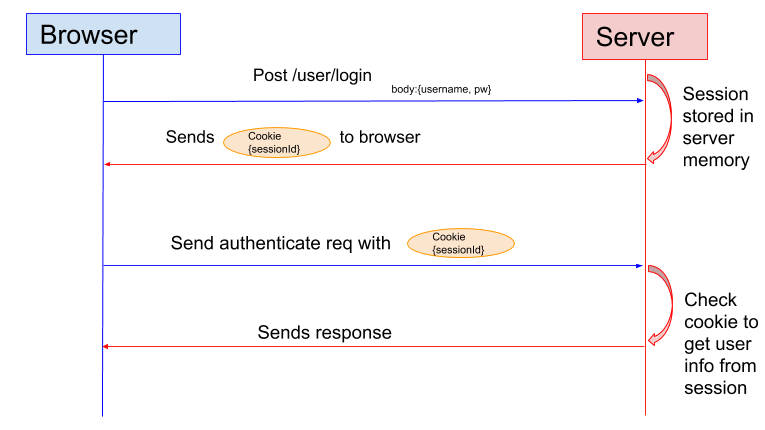


Một trong những cách giải quyết vấn đề này là mỗi request gửi lên đều gửi kèm theo thông tin đã chứng thực trước đó.

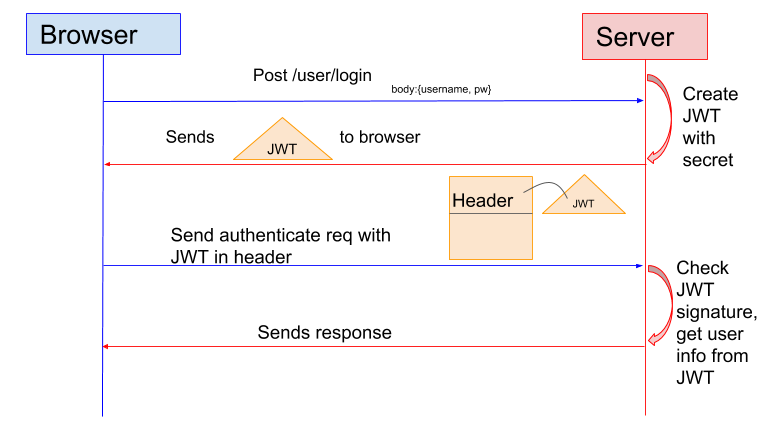


Các cơ chế lưu giữ đăng nhập người dùng:

* + Basic Authentication
    - Là loại đơn giản và cơ bản nhất hiện có. Nó chỉ yêu cầu tên người dùng và mật khẩu để kiểm tra ủy quyền của một người nào đó.
    - Tên người dùng và mật khẩu được gửi dưới dạng giá trị tiêu đề trong Authorization Header. Trong khi sử dụng xác thực cơ bản, chúng ta thêm từ Basic trước khi nhập tên và mật khẩu.
    - Các giá trị tên người dùng và mật khẩu này được mã hóa bằng Base64 nếu không server sẽ không nhận ra nó.
  + Session-based Authentication: server sẽ tạo một session cho người dùng sau khi người dùng đăng nhập. Session ID sẽ được lưu ở cookie trong trình duyệt của người dùng. Trong khi người dùng vẫn còn đăng nhập, cookie sẽ gửi tiếp cùng với những request tiếp theo. Server có thể so sánh các session ID lưu trữ ở cookie với session được lưu trong bộ nhớ để xác minh thông tin người dùng và phản hồi với trạng thái tương ứng.



* + Token-based Authentication: một số trang website sử dụng JSON WEB TOKEN (JWT) thay thế cho việc xác thực bằng session. Trong Token Based Authentication, server tạo JWT một cách bí mật và gửi JWT tới client. Client lưu JWT và thêm JWT và header với mọi request. Server sau đó sẽ xác thực JWT với mọi request từ client và trả về response.



Phần lớn các trang web hiện nay đều sử dụng JWT cho việc xác thực do khả năng mở rộng cho các thiết bị di động.

1. **Phân quyền (Authorization)**

Authorization xảy ra sau khi hệ thống đã được xác thực (authentication) thành công, cuối cùng cho phép người dùng toàn quyền truy cập các tài nguyên hệ thống như thông tin, file, cơ sở dữ liệu,…Nói cách đơn giản, authorization xác định khả năng của người dùng truy cập vào hệ thống và ở mức độ nào. Khi danh tính của bạn được hệ thống xác minh sau khi xác thực thành công, người dùng sẽ được phép truy cập vào tài nguyên của hệ thống.



1. **Sự khác nhau Authentication và Authorization**

Truy cập vào một hệ thống được bảo vệ bởi cả Authentication và Authorization. Mọi nỗ lực truy cập hệ thống có thể được xác thực bằng cách nhập thông tin xác thực, nhưng chỉ có thể được chấp nhận sau khi ủy quyền thành công. Nếu nỗ lục được xác thực nhưng không được phép, hệ thống sẽ từ chối quyền truy cập vào hệ thống.

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| Authentication xác nhận danh tính của người dùng để cấp quyền truy cập vào hệ thống. | Authorization xác định xem người dùng có được phép truy cập tài nguyên không. |
| Đây là quá trình xác nhận thông tin đăng nhập để có quyền truy cập của người dùng. | Đó là quá trình xác minh xem có cho phép truy cập hay không. |
| Nó quyết định liệu người dùng có phải là những gì họ tuyên bố hay không. | Xác định những gì người dùng có thể và không thể truy cập. |
| Authentication thường yêu cầu tên người dùng và mật khẩu. | Các yếu tố xác thực cần thiết để authorization có thể khác nhau, tùy thuộc vào mức độ bảo mật. |
| Authentication là bước đầu tiên của Authorization. | Authorization được thực hiện sau khi Authentication thành công. |

* 1. **CHƯƠNG V: MICROSERVICES AUTHENTICATION VÀ AUTHORIZATION**

1. **Vấn đề**

Login cho việc authentication và authorization logic cần phải được implement lặp lại ở các services khác nhau.

Service chỉ nên tập trung vào business logic của nó.

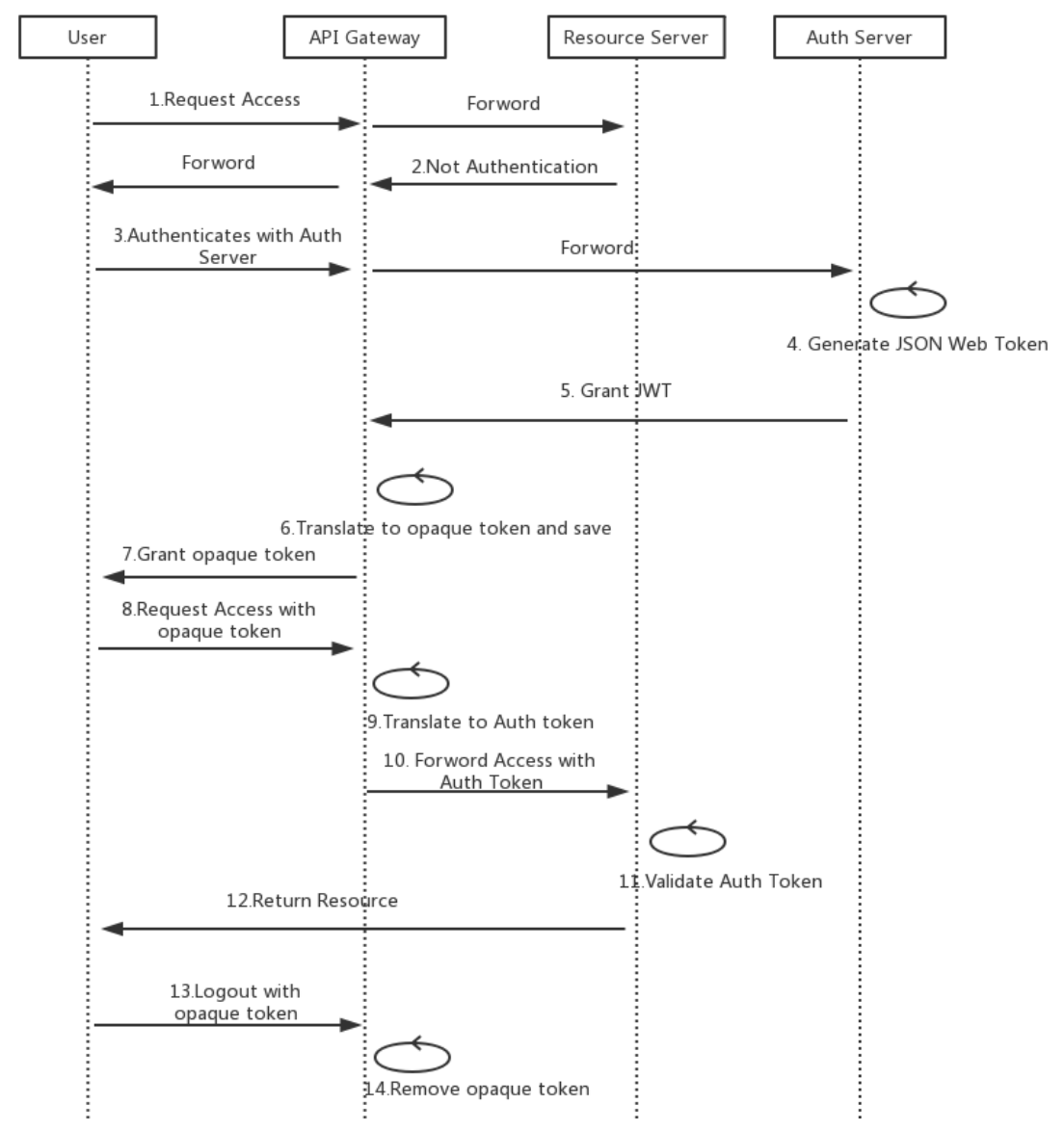
Việc sử dụng session ảnh hưởng đến việc mở rộng theo chiều ngang của hệ thống.

Việc authentication và authorization trong kiến trúc Microservices phức tạp hơn nhiều khi mà không chỉ có việc kết nối giữa các clients và server mà còn liên quan đến kết nối giữa các services với nhau.

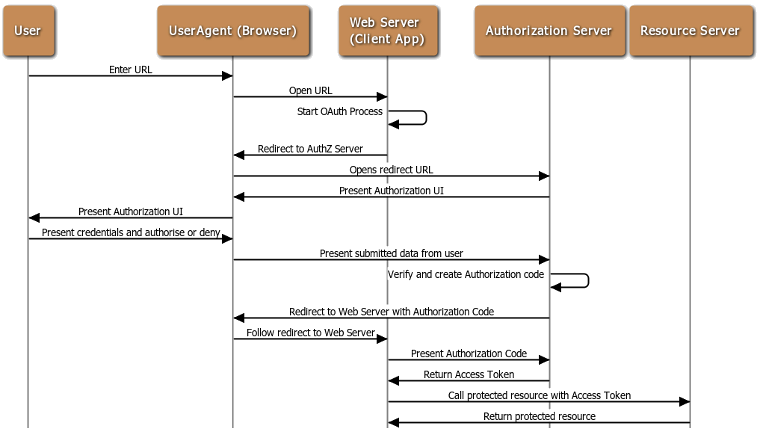
1. **Giải pháp**
   1. ***Distributed Session Management (quản lý phiên)***
   * Sticky session: tất cả các request của một user cụ thể sẽ được gửi đến cùng một server đã xử lý request đầu tiên của user đó.
   * Session replication: mỗi instance sẽ lưu toàn bộ session và sẽ được đồng bộ với nhau.
   * Centralized sessin storage: tính available và scalable cao nhưng cần phải được bảo vệ từ những nguồn không tin cậy hoặc cần phải được kết nối bảo mật.
   1. ***Client token***

Thay vì lưu session trên server thì với cách này clients sẽ giữ một token được cung cấp bởi server và dùng nó cho việc authentication và authorization. Thường thì token sẽ tạo ra theo chuẩn JWT.

* 1. ***Client token với API Gateway***
  + Giải pháp này sẽ thêm 1 serveice gọi API Gateway đứng giữa clients và các services.
  + API Gateway có thể làm việc với Authentication server để lấy JWT khi user đăng nhập và biên đổi thành opaque token (dạng token mà client không thể có thông tin dịch ngược) của riêng nó rồi gửi về cho client
  + Client phải gắn kèm opaque token mỗi khi request đến API Gateway -. API Gateway sẽ truy xuất loại token đã lưu, dịch ngược ra JWT và gửi nó đến các services tương ứng.
  + Tất cả các requests từ clients sẽ được gửi đến API Gateway thay vì đến server.



* 1. ***Third-party application access***
  + API Token: hệ thống sẽ cung cấp cho user cách thức để sinh ra 1 token theo nhu cầu. Với token này user có thể đưa vào các ứng dụng 3rd-party để yêu cầu dữ liệu từ hệ thống.
  + Giảm thiểu việc tiết lộ password.
  + Cho phép khả năng thay đổi quyền truy cập bất cứ lúc nào.



* 1. ***Mutual Authentication (Authentication giữa các services trong hệ thống Microservices)***

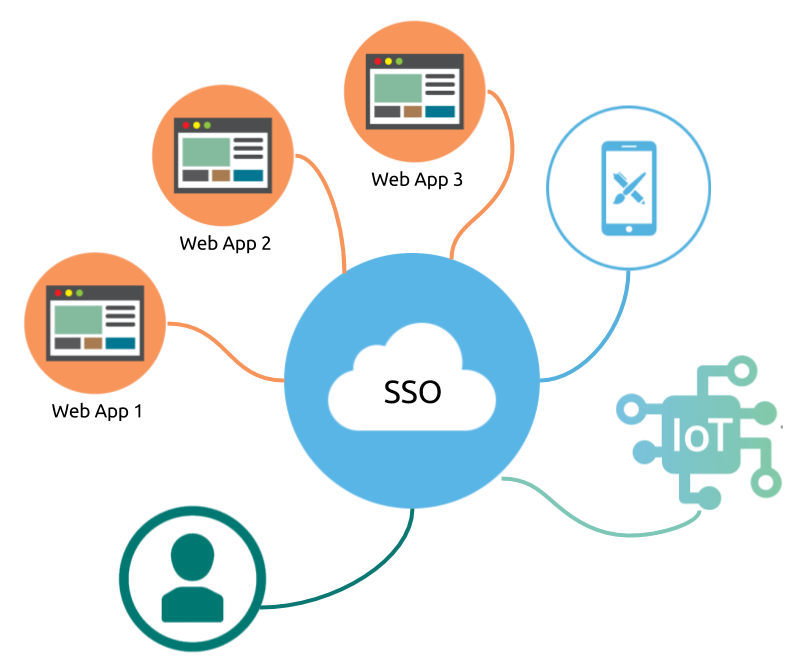
Nên phát triển ứng dụng SSL và TLS để tránh bị tấn công và đánh cắp thông tin.

Cần có 1 certificate cho mỗi microservice, các microservices sẽ được authenticated với certificates của các Microservices khác.

Trong trường hợp các Microservice intances thường xuyên bị thay đổi, ta có thể tạo ra một private certificate center để phục vụ cho việc quản lý các centificate cho các Microservices như việc cung cấp, thay đổi hoặc thu hồi.

* 1. ***Single Sign-On***

Cho phép người dùng sử dụng một bộ thông tin đăng nhập (tên – mật khẩu) để truy cập vào nhiều services.



**TÀI LIỆU THAM KHẢO**

1. T.S Lê Phan Thị Diệu Thảo, Nguyễn Thảo Phương, *Nợ công và những vấn đề cần bàn thêm*, Nhà xuất bản Kinh Tế.
2. Boulding, K.E. (1995), *Economics analysis*, Hamish Hamilton, London.
3. <Http://dantri.com.vn/c76/s76-402058/nhung-moc-chinh-cua-khung-hoang-no-chau-au.htm>, “Những mốc chính của khủng hoảng nợ châu Âu”

**PHỤ LỤC**

Phụ lục 1: hướng dẫn cài đặt

Phụ lục 2: hướng dẫn sử dụng