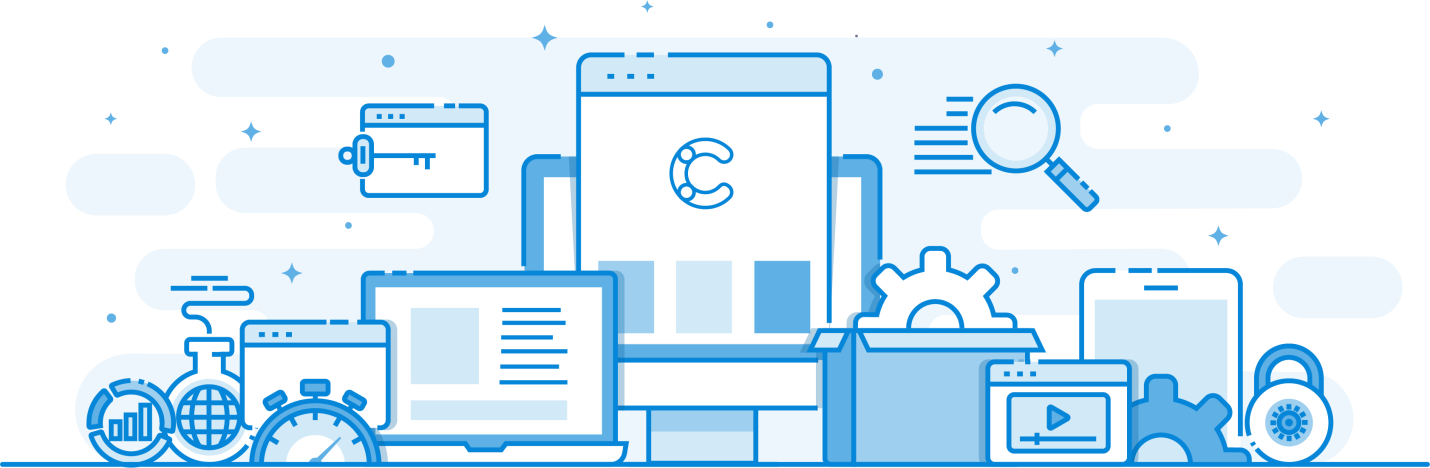
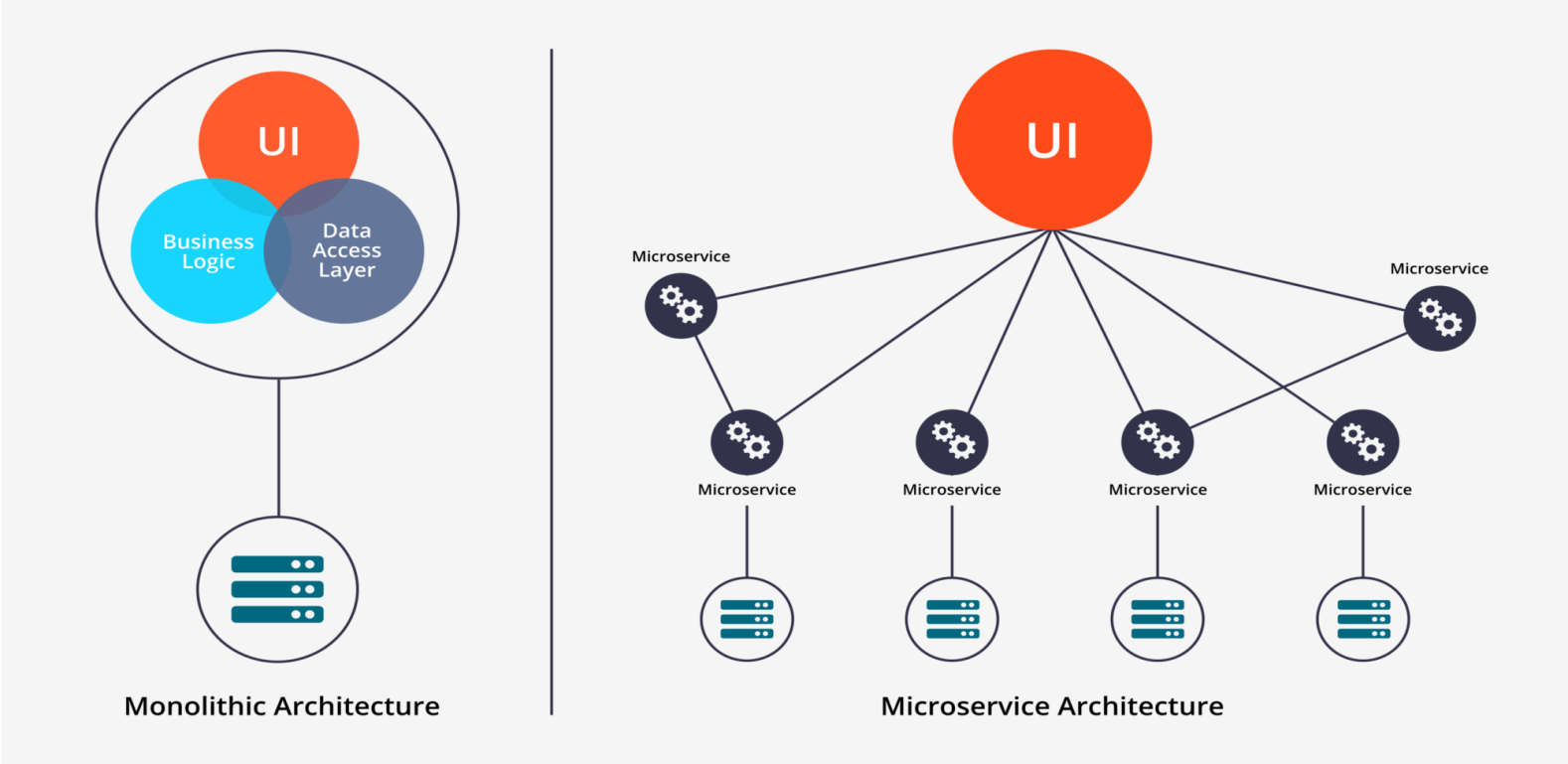
1. Microservices là gì ?

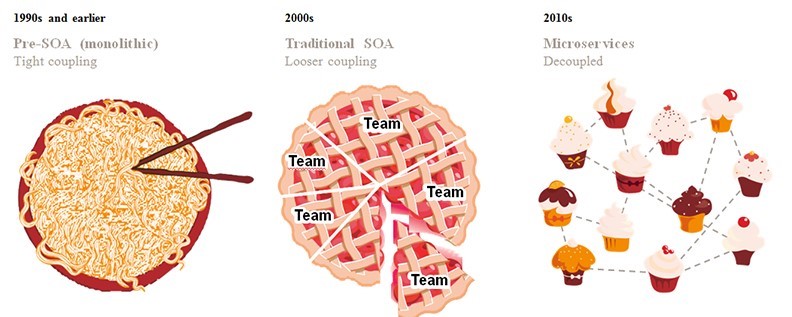


* Microservices là một kỹ thuật phát triển phần mềm - một biến thể của kiến ​​trúc SOA (service – oriented architecture)
* Nền tảng kiến trúc microservices là xây dựng một ứng dụng tổng hợp của nhiều services nhỏ và độc lập có thể chạy riêng biệt, phát triển và triển khai độc lập, liên tục.

1. Kiến trúc của Microsevices. Ưu & nhược điểm
   1. Kiến trúc ứng dụng



* Ý tưởng quan trọng chính là nhìn vào các tính năng trong một ứng dụng monolithic, ta có thể nhận biết, xác định các yêu cầu và khả năng cần thiết để đáp ứng một nghiệp vụ. Sau đó từng năng lực nghiệp vụ này sẽ được xây dựng thành những service 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 & nhược điểm
     1. Ưu điểm



* Microservice giúp giảm thiểu quá trình phức tạp hóa, rối rắm hóa trong các hệ thống lớn.
* Microservice cho phép mỗi service phát triển, đóng gói, triển khai và có thể mở rộng quy mô một cách dễ dàng, độc lập với nhau.
* Mỗi microservice cho phép lựa chọn công nghệ thích hợp, tiết kiệm chi phí phát triển, đóng gói & triển khai.
  + 1. Khuyết điểm
* Nhược điểm đầu tiên của Microservice chính từ tên gọi của nó. Microservices khuyến khích chia nhỏ gọn các services -> khó khăn trong việc chia nhỏ, chia như thế nào để khi chia quá nhiều sẽ dẫn đến mảnh mún, vụn vặt, khó kiểm soát
* Nhược điểm thứ hai của Microservice đến từ đặc tính phân tán
* Thứ ba, đôi khi testing một service trong kiến trúc microservice phải chạy tất cả services khác trong chuỗi mắt xích mà service đó phụ thuộc.
* Cuối cùng, việc triển khai microservices phức tạp rất nhiều nếu làm thủ công theo cách đã làm với ứng dụng Monolithic.

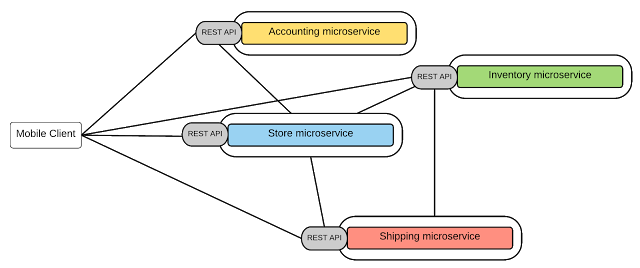
1. Kích cỡ, phạm vi & tính năng

* Số dòng code/ kích cỡ của một đội lập trình là chỉ số tồi
* "Micro" là một từ khóa dễ gây nhầm lẫn. Một số lập trình viên nghĩ rằng họ nên tạo ra services nhỏ hết mức.
* Trong SOA, services thường trở thành các cục monolithic với nhiều hàm, chức năng khác hỗ trợ.

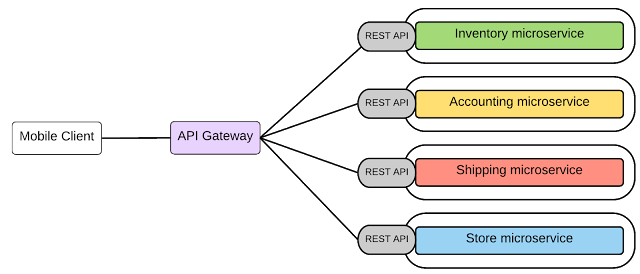
Một Số Chỉ Dẫn Khi Thiết Kế Microservices

* Single Responsibility Principle
* Trong quá trình thiết kế, xác định và giới hạn các services theo chức năng nghiệp vụ thực tế (theo [Domain Driven Design](http://martinfowler.com/bliki/BoundedContext.html))
* 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 services nhỏ hơn.

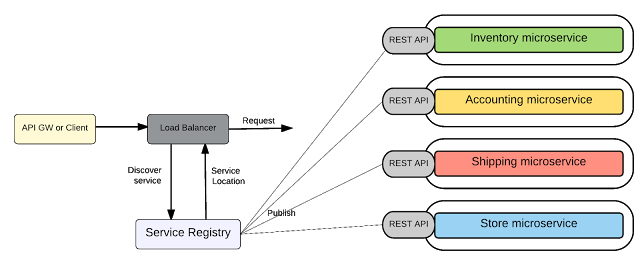
1. Kiểu mẫu thiết kế phổ biến
   1. Point-to-point – Kết nối trực tiếp giữa các services



* Mô hình này đơn giản và hoạt động ổn với ứng dụng microservices tương đối nhỏ nhưng khi số lượng services tăng lên, việc giao tiếp trở nên cực phức tạp.
* Tổng hợp một số vấn đề của kiểu giao tiếp trực tiếp:
* Những yêu cầu như xác thực người dùng, điều tiết, giám sát,...phải được xây dựng tại tất cả microservices.
* Việc trên dẫn đến lập các tính năng chung, mỗi microservices có thể trở nên phức tạp.
* Không có cách quản lý, kiểm soát giao tiếp giữa các services
  1. API – Gateway



* API Gateway sử dụng một cổng truyền tin gọn nhẹ như một điểm vào chính cho client và triển khai những chức năng chung mà không liên quan đến nghiệp vụ đặc thù ở cấp Gateway này.
* API Gateway cho phép bạn sử dụng một API được quản lý qua REST/HTTP.
* API Gateway cung cấp các lợi thế dưới đây:
* Cung cấp một lớp trừu tượng hóa các microservices.
* Định tuyến và chuyển đổi tin nhắn gọn nhẹ ở cấp gateway.
* Một điểm tập trung cho các chức năng chung không mang tính nghiệp vụ kinh doanh như bảo mật, giám sát và điều tiết.
* Microservices trở nên càng gọn nhẹ vì các chức năng chung không mang tính nghiệp vụ đều chuyển sang Gateway.



* Điều hướng tới microservices tương ứng với mỗi request từ Client
* Load balancer giúp phân phối lưu lượng truy cập ứng dụng
* Circuit breaker giúp ngăn ngừa chuỗi thất bại liên tục của các services
* Security and user management (Using JWT or OAuth2)
* Rate limiting giúp giới hạn tốc độ số lương yêu cầu HTTP từ client trong một khoảng thời gian nhất định
* Generate UI cơ bản cho microservices
* Generate tài liệu các APIs với Swagger Service

1. Nghiệp vụ trong spec không rõ ràng, đầy đủ khiến cho developer mất nhiều thời gian hỏi, đôi khi hiểu nhầm spec -> sửa code.
2. Không có tầm nhìn bao quát tới chi tiết dẫn đến việc thiết kế model, sử dụng object, function không được tối ưu.
3. Khách hàng thường không biết tech, họ chỉ cần làm sản phẩm đúng ý họ. Trong khi đó, dev lại cần nghiệp vụ -> thiếu tương tác trực tiếp với khách hàng. DDD đề cao sự tương tác trực tiếp, thấu hiểu nghiệp vụ từ khách hàng tới dev. Tuy nhiên, như đề cập trên, khách hàng họ không biết tech, mà dev lại dùng chủ yếu ngôn ngữ tech và ngược lại. Vậy làm thế nào để khách hàng có thể truyền đạt hết, rõ nghiệp vụ cho dev và dev nói chuyện với khách hàng một cách dễ hiểu để tương tác giữa hai bên được tốt nhât ? DDD giúp ta giải quyết các vấn đề này bằng việc tập chung vào nghiệp vụ. DDD cung cấp cho chúng ta cấu trúc phát triển dự án, những công nghệ giúp cho việc thiết kế, phát triển dự án tập chung vào nghiệp vụ đối với các dự án có nghiệp vụ phức tạp. Qua quá trình tìm hiểu, mình thấy một số concept quan trọng:
   1. Domain: Ở đây có thể hiểu đơn giản là miền nghiệp vụ bao gồm tất cả kiến thức, tầm ảnh hưởng, các khái niệm trong lĩnh vực liên quan tới nghiệp vụ. Việc hiểu rõ về domain sẽ giúp chúng ta xây dựng hệ thống tốt hơn, đáp ứng đúng yêu cầu. Tuy nhiên, làm sao để hiểu được domain ?
   2. Ubiquitous language: Để trả lời câu hỏi trên, chúng ta có "Ngôn ngữ chung". Nó là tập hợp của các từ ngữ thông thường trong lĩnh vực nghiệp vụ. Tuy nhiên được diễn đạt một cách dễ hiểu, gần gũi để cả dev và customer hiểu và nói chuyện được với nhau bằng ngôn ngữ này. NOTE: không dùng ngôn ngữ chuyên môn trong tech.
   3. Entity: Đại diện cho một đối tượng trừu tượng trong nghiệp vụ. Có thể hiểu đơn giản như Model với các đặc tính để nhận biết (attribute).
   4. Repository: Chắc hẳn chúng ta không còn xa lạ với cụm từ này. Áp dụng repository design pattern trong DDD đem lại hiệu quả vô cùng lớn.
   5. Domain service: Chính là các phương thức nơi cài đặt logic nghiệp vụ là cách để thực thi actions, operations and activities.