Phần 1: Kiến trúc đơn khối - Kiến trúc phân tán

1. Monolithic Architecture
2. Ưu điểm:

* Design: thiết kế đơn giản chỉ 1 code base duy nhất với nhiều function phụ thuộc lẫn nhau
* Development: khi một ứng dụng được xây dựng với một code base, việc phát triển (develop) sẽ dễ dàng hơn.
* Deployment: chỉ thực thi (executable) 1 file hoặc 1 directory nên việc triển khai (deployment) dễ dàng hơn.
* Debugging: tất cả code ở cùng 1 nơi nên việc the dõi yêu cầu (request) và tìm sự cố (issue) dễ dàng hơn
* Testing: Có thể test toàn bộ ứng dụng một cách dễ dàng và tìm kiếm lỗi trong môi trường đồng nhất.
* Performance - Hiệu suất cao trong các ứng dụng ít phức tạp: Các ứng dụng ít phức tạp có thể chạy hiệu quả với kiến trúc monolithic do giảm thiểu giao tiếp mạng và độ trễ

1. Nhược điểm

* Slower development speed: một ứng dụng lớn, nguyên khối khiến quá trình phát triển trở nên phức tạp và chậm hơn.
* Scalability: không thể mở rộng các thành phần riêng lẻ.
* Reliability: nếu có lỗi trong bất kỳ mô-đun nào, lỗi đó có thể ảnh hưởng đến tính khả dụng của toàn bộ ứng dụng.
* Barrier to technology adoption (Rào cản đối với việc áp dụng công nghệ): bất kỳ thay đổi nào trong khuôn khổ hoặc ngôn ngữ đều ảnh hưởng đến toàn bộ ứng dụng, khiến việc thay đổi thường tốn kém và mất thời gian.
* Lack of flexibility (Thiếu tính linh hoạt): bị hạn chế bởi các công nghệ đã được sử dụng trong khối nguyên khối.
* Deployment: một thay đổi nhỏ đối với một ứng dụng nguyên khối đòi hỏi

phải triển khai lại toàn bộ hệ thống.

1. Distributed Architecture
2. Ưu điểm

* Khả năng chịu lỗi và tin cậy: nếu một dịch vụ bị lỗi, các dịch vụ khác có thể tiếp tục thực hiện các yêu cầu dịch vụ.
* Khả năng thích ứng: Chức năng của ứng dụng được chia thành các đơn vị phần mềm được triển khai riêng biệt, dễ dàng xác định vị trí và áp dụng thay đổi hơn, phạm vi thử nghiệm được giảm xuống chỉ còn dịch vụ bị ảnh hưởng và rủi ro triển khai giảm đáng kể vì thường chỉ triển khai dịch vụ bị ảnh hưởng.
* Có khả năng mở rộng: vì chứa một tập hợp các máy độc lập nên có thể mở rộng theo chiều ngang dễ dàng.
* Độ trễ thấp: do có nhiều máy chủ và có khả năng phản hồi đến gần người dùng hơn để giải quyết truy vấn, do đó mất rất ít thời gian để giải quyết truy vấn của người dùng.

1. Khuyết điểm

* Tăng đáng kể độ phức tạp của kiến trúc: quản lý nhiều chương trình nhỏ hơn rõ ràng phức tạp hơn so với một ứng dụng monolithic.
* Yêu cầu phải deploy tự động.
* Giảm tổng thể về hiệu suất: kiến trúc microservices thường dẫn đến mức tiêu tốn tài nguyên cao hơn so với kiến trúc monolithic có cùng quy mô.
* Các microservice đòi hỏi nhiều bộ nhớ hơn, tốn nhiều chu kỳ CPU hơn và cần băng thông mạng lớn hơn.
* Quản lý các giao dịch phân tán, tính nhất quán cuối cùng, quản lý quy trìnhlàm việc, xử lý lỗi, đồng bộ hóa dữ liệu
* Khó khắc phục sự cố và gỡ lỗi: xác định và giải quyết các vấn đề trong kiến trúc microservices cũng phức tạp tương tự như việc tổng hợp các file theo dõi và log. Khi một yêu cầu thất bại đi qua nhiều microservice được lưu trữ trong các môi trườnchạy riêng biệt, việc xác định chính xác vị trí và nguyên nhân của sự cố có thể rất khó khăn.
* Chi phí cao:

• Tăng tổng dung lượng bộ nhớ

• Tài nguyên bị trùng lặp khi sử dụng nhiều container hoặc máy ảo.

• Tiêu tốn băng thông khi gọi các dịch vụ web RESTful …

Phần 2. Architecture Patterns

*1. Layered Architecture*

* ***Ưu điểm:*** tách biệt logic giữa các tầng
* ***Khuyết điểm:*** kiến trúc phân lớp vẫn có một yếu điểm dễ thấy là logic phải được truyền theo thức tự các tầng, bất kể tầng đó có xử lý dữ liệu hay không. Điều này gây ra sự lãng phí tài nguyên.
* **Trường hợp nên dùng kiến trúc phân lớp:** Nếu dự án có những hạn chế nhất định về thời gian và ngân sách, kiến trúc phân lớp là một sự lựa chọn thích hợp vì các lý do chính sau:

• Thuộc loại kiến trúc monolithic nên không phức tạp

• Dựa trên mức độ phổ biến của kiến trúc phân lớp, hầu hết developer điều quen thuộc, việc vận hành, phát triển dự án trở nên dễ dàng hơn.

• Kiến trúc phân lớp thuộc nhóm kiến trúc phân vùng kỹ thuật (Technical Partitioning) nên việc sử dụng kiến trúc này phù hợp với nhiều tổ chức khác nhau do các nhóm được chia thành nhóm chuyên trách kỹ thuật như nhóm FE, nhóm BE, nhóm database, ...

*2.Client – Server Architecture*

a. Ưu điểm

* Quản lý tập trung trên một Server, kiến trúc này đơn giản hóa việc bảo trì, cập nhật và quản lý bảo mật. Người quản trị có thể giám sát và quản lý dữ liệu, áp dụng các bản cập nhật và thực thi chính sách bảo mật hiệu quả từ một vị trí duy nhất.
* Khả năng mở rộng: khi số lượng Client tăng lên, có thể thêm Server hoặc mở rộng dung lượng Servẻ hiện có mà không làm thay đổi đáng kể kiến trúc hệ thống tổng thể.
* Tối ưu hóa tài nguyên: mô hình này cho phép phân bổ tài nguyên được tối ưu hóa. Server được thiết kế để xử lý chuyên sâu và lưu trữ dữ liệu lớn, trong khi Client được tối ưu hóa cho các tương tác và yêu cầu của người dùng. Sự tách biệt này đảm bảo sử dụng hiệu quả tài nguyên hệ thống.
* Độ tin cậy và tính khả dụng: với cơ sở hạ tầng máy chủ mạnh, hệ thống Client-Server có thể đảm bảo độ tin cậy và tính khả dụng cao.

b.Nhược điểm

* Kiểm soát tập trung có thể dẫn đến khả năng xảy ra lỗi cao hơn. Khi nhiều Client gửi yêu cầu đồng thời đến Server, Server có thể quá tải và làm chậm hiệu suất. Điều này cũng có thể dẫn đến lỗi Server, khiến toàn bộ hệ thống ngừng hoạt động.
* Server mạnh hơn Client, nghĩa là Server đắt hơn. Server cũng yêu cầu người có kiến thức về mạng và cơ sở hạ tầng để quản lý hệ thống.
* Kiến trúc Client-Server dễ bị tấn công Từ chối dịch vụ (DoS) vì số lượng Server thường nhỏ hơn số lượng Client.
* Các gói dữ liệu có thể bị thay đổi trong quá trình truyền, dẫn đến khả năng mất thông tin hữu ích.

1. *Pipeline Architecture*

* ***Ưu*** điểm: dễ dàng thay đổi/bảo trì/dùng lại từng filter của hệ thống, phù hợp với nhiều hoạt động nghiệp vụ, dễ dàng nâng cấp bằng cách thêm filter mới.
* Khuyết điểm: 2 filter kề nhau cần tuân thủ định dạng dữ liệu chung
* Tình huống **nên dùng**: trong các ứng dụng xử lý dữ liệu mà dữ liệu nhập cần được xử lý bởi nhiều công đoạn khác nhau và có tính độc lập cao trước khi tạo ra kết quả cuối cùng.

1. *Event-Driven Architecture – EDA*
2. Ưu điểm

* Tính linh hoạt cao: Các dịch vụ không phụ thuộc chặt chẽ vào nhau, giúp hệ thống dễ mở rộng và bảo trì.
* Khả năng mở rộng tốt: Do xử lý sự kiện bất đồng bộ nên dễ dàng mở rộng theo chiều ngang (scale-out).
* Tốc độ phản hồi nhanh: Các sự kiện được xử lý ngay khi phát sinh, không cần chờ phản hồi từ các dịch vụ khác.
* Giảm tải cho hệ thống: Chỉ xử lý khi có sự kiện, tránh việc kiểm tra liên tục (polling).
* Tính **đàn hồi cao**: Hệ thống có thể tiếp tục hoạt động ngay cả khi một số dịch vụ bị lỗi, miễn là hàng đợi sự kiện (event queue) vẫn hoạt động.

1. Nhược điểm

* **Độ phức tạp cao**: Khó theo dõi và gỡ lỗi do hệ thống phi tập trung.
* Khó kiểm soát thứ tự xử lý: Do các sự kiện được xử lý bất đồng bộ, có thể xảy ra tình trạng xử lý không đúng thứ tự mong muốn.
* Cần cơ chế quản lý sự kiện tốt: Cần sử dụng message broker (Kafka, RabbitMQ) để đảm bảo tin nhắn không bị mất hoặc trùng lặp.
* Độ trễ có thể tăng: Nếu hàng đợi quá tải hoặc gặp sự cố, các sự kiện có thể bị xử lý chậm hơn mong đợi.

1. *Service-Oriented Architecture – SOA*
2. Ưu điểm

* **Tái sử dụng dịch vụ cao**: Các dịch vụ có thể được tái sử dụng trong nhiều ứng dụng khác nhau.
* **Khả năng tích hợp tốt**: Hỗ trợ giao tiếp với các hệ thống cũ (legacy systems) thông qua các giao thức tiêu chuẩn như SOAP, REST.
* **Dễ bảo trì và phát triển**: Các dịch vụ có thể được phát triển và triển khai độc lập.
* **Quản lý bảo mật tốt hơn**: Do sử dụng các giao thức như WS-Security để đảm bảo an toàn dữ liệu.

1. Nhược điểm

* **Hiệu suất thấp hơn do có quá nhiều giao tiếp qua mạng**: Các dịch vụ liên tục trao đổi dữ liệu với nhau, gây tốn băng thông.
* **Cấu trúc phức tạp**: Cần một **Enterprise Service Bus (ESB)** để quản lý luồng dữ liệu giữa các dịch vụ, làm tăng độ phức tạp.
* **Khó mở rộng**: Mô hình này chủ yếu tập trung vào việc tích hợp hơn là tối ưu khả năng mở rộng.
* **Độ trễ cao**: Việc xử lý dữ liệu thông qua ESB có thể gây chậm trễ.

1. *Microservices Architecture*
2. Ưu điểm

* **Dễ mở rộng (scalability)**: Có thể mở rộng từng service riêng lẻ thay vì toàn bộ hệ thống.
* **Triển khai nhanh và linh hoạt**: Các nhóm có thể phát triển, triển khai và cập nhật từng microservice mà không ảnh hưởng đến hệ thống chung.
* **Công nghệ đa dạng**: Có thể sử dụng nhiều công nghệ khác nhau cho từng service, giúp tối ưu hiệu suất.
* **Dễ bảo trì và nâng cấp**: Do các dịch vụ nhỏ gọn, độc lập, giúp giảm thiểu ảnh hưởng khi thay đổi hoặc sửa lỗi.
* **Tăng độ ổn định**: Nếu một service bị lỗi, hệ thống tổng thể vẫn có thể tiếp tục hoạt động.

1. Nhược điểm

* **Tăng độ phức tạp**: Việc quản lý nhiều service nhỏ thay vì một hệ thống lớn yêu cầu công cụ và kỹ năng quản lý tốt hơn.
* **Khó khăn trong giao tiếp giữa các service**: Cần sử dụng API Gateway hoặc message broker để quản lý giao tiếp giữa các service
* **Vấn đề đồng bộ dữ liệu**: Vì các microservice hoạt động độc lập, cần có giải pháp để đồng bộ dữ liệu giữa chúng
* **Cần DevOps mạnh**: Việc triển khai, giám sát và quản lý microservices yêu cầu các công cụ CI/CD, containerization (Docker, Kubernetes)
* **Chi phí vận hành cao**: Mỗi service có thể yêu cầu một database riêng, nhiều instance chạy song song, làm tăng chi phí hạ tầng.