Công ty cổ phần VCCorp

BÁO CÁO TUẦN 1

Tìm hiểu về Microfrontend, 12 Factor và Clean Architecture

Tác giả: Ngô Minh Đức

Người hướng dẫn: Anh Ngô Văn Vĩ

Hà Hội, 6/2025

Tóm tắt nội dung

Báo cáo này trình bày những nội dung quan trọng về khái niệm, đặc điểm, ưu điểm, nhược điểm, các tối ưu của các mô hình Microfrontend, Clean Architecture và phương pháp 12Factor.

MỤC LỤC

Chương 1: Microfrontend

Đây là một kiến trúc web, cho phép chúng ta chia nhỏ dự án thành các thành phần độc lập để thực hiện những chức năng khác nhau. Kiến trúc này được lấy ý tưởng từ microservices của backend nhưng tập trung chủ yếu ở frontend. Mỗi thành phần độc lập có thể là 1 màn hình, 1 package, 1 component hay 1 function do các nhóm khác nhau quản lý, phát triển, kiểm thử và triển khai trong cùng một lúc.

Hiện nay, các công ty phải phát triển các ứng dụng web lớn và phân tán. Điều đó đòi hỏi sự phân chia ứng dụng lớn thành các phần nhỏ và phân công cho các nhóm riêng biệt. Ví dụ: một trang web phục vụ cho mục đích thương mại có nhiều thành phần như giỏ hàng, sản phẩm, thanh toán, tài khoản, thông báo, dịch vụ,… Thay vì phải viết chung các vào một codebase thì kiến trúc này sẽ chia nhỏ để mỗi nhóm phụ trách 1 phần riêng và làm việc song song, không ảnh hưởng đến toàn bộ hệ thống.

Kiến trúc Microfrontend mang lại nhiều lợi ích cho cả lập trình viên và cả người dùng. Thứ nhất, kiến trúc này cho phép các nhóm dev làm việc song song trên các thành phần riêng biệt của ứng dụng, phát triển và kiểm thử bằng những công nghệ khác nhau. Điều này giúp tăng hiệu quả công việc, giảm thời gian xây dựng, dễ dàng trong việc kiểm thử và tái sử dụng lại khi cần thiết. Thứ hai, kiến trúc có dạng modun tách rời nên việc thêm hoặc thay đổi frontend mới sẽ không ảnh hưởng đến các thành phần còn lại, giúp việc mở rộng hiểu quả hơn. Cuối cùng, kiến trúc giúp tăng trải nghiệm người dùng bằng việc giảm thời gian tải bằng cách cung cấp những thành phần thực sự cần thiết với các hành vi của người dùng.

Tuy nhiên, kiến trúc này còn có nhiều nhược điểm, đặc biệt là đối với các dự án nhỏ, đơn giản, không quá nhiều chức năng riêng biệt, team nhỏ, ít thành viên. Điều này khiến cho cấu trúc tổng thể trở nên phức tạp dẫn đến việc khó quản lí, khó giao tiếp giữa các thành phần nếu không có quy định chung ban đầu của leader. Giao

diện có thể không nhất quán nếu sử dụng nhiều công nghệ khác nhau và không tuân thủ hê thống thiết kế chung.

Tóm lại, Microfrontend là một cấu trúc web mạng lại nhiều tiện lợi cho ứng dựng lớn, cần phân chia thành nhiều thành phần và giao cho các nhóm phát triển khác nhau. Tuy nhiên, khi sử dụng cần cân nhắc về các vấn đề tích hợp, đồng bộ các công nghệ đúng các, có các quy định chung, style riêng cho hệ thống của mình.

Chương 2: 12 Factors

12Factor hay còn được gọi là "12 nguyên tắc phát triển ứng dụng" là một phương pháp dùng để xây dựng các ứng dụng một cách nhanh chóng, an toàn và dễ dàng trong việc triển khai, bảo trì, mở rộng hệ thống. Với việc các web, app hiện này ngày càng lớn, phức tạp thì phương pháp này được hình thành để mạng lại những hiệu quả tốt nhất cho dev đặc biệt đối với môi trường cloud native, microservices, DevOps, CI/CD và hệ thống phân tán. Sau đây là những yếu tố quan trọng của phương pháp này:

STT	Tên yếu tố	Đặc điểm	Minh họa
1	Codebase	- Một mã gốc được theo dõi bởi hệ thống quản lý phiên bản như Git, Mecurial,… và nhiều lần triển khai. - Các bản lưu của các phiên bản được gọi là kho mã (repository). - Một code base là một repo riêng lẻ hoặc nhóm các repo riêng lẻ chia sẻ cùng một commit nguồn. - Mỗi hệ thống phân tán có thể có nhiều codebase mà mỗi codebase đều phải tuân theo 12factor.	Triển khai một ứng dụng thương mại điện tử có nhiều chức năng như giỏ hàng, quản lí người dùng, thành toán,.. thì cần tổ chức một repo Git duy nhất chưa toàn bộ mã nguồn cho ứng dụng và phân chia thành nhiều môi trường triển khai khác nhau như: - dev.shop (môi trường dev) - staging.shop (môi trường test) - myshop (production)

		- Việc nhiều ứng dụng chia sẻ cùng một mã sẽ vi phạm luật và phải xem xét lại các nhóm mã chia sẻ thành các thư viện thông qua dependence manager. - Mã gốc của ứng dụng chỉ có một nhưng có thể có nhiều triển khai của ứng dụng đó bằng việc các dev sẽ có một bản lưu của một ứng dụng đang chạy trên môi trường phát triển cá nhân. Ví dụ: dev có nhiều commit nhưng chưa trienr khai vào hệ thống thử (staging) và hê thống có nhiều commit mã chưa chưa triển khai vào hệ thống sản xuất (production) nhưng cả hai lại đều chia sẻ chung một codebase.	
2	Dependences	- 12Factor không dựa vào các tồn tại ngầm của các gói toàn hệ thống mà khai báo tất cả dependences chính xác và đầy đủ thông qua dependence declaration, sử dụng dependenve isolation trong quá trình thực thi không cho ngầm định rò rỉ từ hệ thống xung quanh.	- Trong NodeJS cần cài các denpendencies đầy đủ trong file package.json thủ công hoặc lệnh “npm install”. - Trong Java đặc biệt là Maven thì thường phải cài các dependency về groupId, artifacID, version bằng thủ công hoặc lệnh “mvn clean install”

		- Dependence specification được triển khai đầy đủ và rõ rang để áp dụng cho cả quá trình sản xuất và phát triển. - Việc khai báo đầu đủ denpendence giúp đơn giản hóa việc thiết lập ứng dụng. Dev mới có thể kiểm tra codebase trên máy tính cá nhân và thiết lập các điều kiện cần thiết để chạy mã bằng build command.	
3	Config	- Config là mọi thứ có khả năng thay đổi giữa các lần deploy gồm xử lí database, Memcached, thông tin Xác thực từ các dịch vụ bên ngoài và giá trị triển khai như trên máy chủ. - Khi xây dựng ứng dụng cần đảm bảo việc tách biệt config ra khỏi code vì giữa các lần triển khai thì config sẽ thay đổi nhiều còn code thì không. - 12factor thường lưu trữ config trong các biến môi trường env. Các biến này có thể dễ dàng thay đổi mà không thay đổi code và cũng là một	Trong NodeJS thì để thiết lập config cần cài gói dotenv bằng lệnh “npm install dotenv”. Sau đó tạo file .env rồi thiết lập cấu hình các PORT, DB_USER, DB_PASSWORD, DB_SERVER, DB_PORT, DB_NAME,… Trong file chính để thiết lập server thì khai báo các lệnh như: “require(‘dotenv’).config

		tiêu chuẩn không phụ thuộc vào hệ điều hành.	

4	Backing Services	- Là bất kì dịch vụ nào mà ứng dụng sử dụng qua mạng như kho dữ liệu MySQL, MongoDB,…, hệ thống nhắn tin như RabbitMQ, dịch vụ SMTP cho email gửi đi như Postifix và các hệ thống trữ đệm như Memcached.
		- Các dịch vụ về cơ sở dữ liệu thường được quản lí bởi admin. Ngoài quản lí cục bộ, ứng dụng có thể có các dịch vụ do bên thứ ba cũng cấp và quản lý như SMTP, Amazon S3 hay Google Maps,..
		- Không phân biệt giữa các dịch vụ và bên thứ ba. Đây đều là tài nguyên được đính kèm, truy cập thông qua URL hoặc trình định vị, xác thực được lưu trữ trong cấu hình.
		- Mỗi dịch vụ là một tài nguyên. Tài nguyên có thể gắn vào và tách ra theo ý muốn của dev.
			Trong NodeJS để kết nối với csdl SQL Server còn có cách lệnh:
			const config ={
			user:
			process.env.DB_USER,
			password:
			process.env.DB_PASSWORD,
			server:
			process.env.DB_SERVER,
			port:
			parseInt(process.env.DB_PORT, 10),
			database:
			process.env.DB_NAME,
			options:{
			encrypt: true,
			trustServerCertificate:
			true
			}
			}
			let pool;
			const connectToDatabase =
			async() => {
			}
			connectToDatabase();

5	Build, Release, Run	- Codebase được chuyển thành deploy thông qua các 3 giai đoạn. Thứ nhất, giai đoạn xây dựng là một chuyển từ code repo thành gói	Trong NodeJS, giai đoạn Build gồm biên dịch TypeScript sang JS để có mã nguồn của các src dist, .. Giái đoạn Release sẽ tạo bản release bằng cách gắn config

		thực thi (build) và lấy các phụ thuộc của nhà cung cấp và biên dịch các tệp. Thứ hai, phát hành là lấy bản dựng dựng được từ xây dựng và hợp nó với cấu hình hiện tại, chứa cả bản dựng và bản cấu hình, sẵn sàng thực hiện trong môi trường thực thi. Thứ ba, giai đoạn chạy là chạy ứng dụng trong môi trường thực thi bằng cách khởi chạy một số quy trình nhất định. - 12Factor sử dụng nghiêm ngặt, tách biệt giữa các giai đoạn trên. - Mỗi công cụ triển khai thường cung cấp các công cụ quản lí phát hành. - Mỗi bản phát hành luôn có ID bản phát hành duy nhất, không thể thay đổi khi được tạo. - Giai đoạn chạy nên được giữ ở càng ít moving part càng tốt và có thể diễn ra tự động bằng khởi động lại máy hoặc khởi động lại bởi process manager.	vào bản build(src, dist,..) thông qua biến môi trường (env vars): export PORT = 3000,… Giai đoạn Run thì chỉ cần chạy các lệnh node file.js
6	Processes	- Ứng dụng được thực thi trong môi trường thực thi dưới dạng một hoặc nhiều quy trình.	Trong NodeJS, khi xây dựng web có các thành phần như Web server (Express), Email worker, Cron job, Session người dùng thì không nên

		- Với 12Factor, quy tình không có trạng thái và không chia sẻ gì cả. Bất kì dữ liệu nào cần duy trì phải được lưu trữ trong Backing services có trạng thái và thường là database. - Hệ thống tệp của quy trình như một bộ đệm giao dịch ngắn gọn. Các tệp lưu trong bộ nhớ đệm có thể không được sử dụng cho các yêu cầu công việc trong tương lai vì nhiều yêu cầu trong tương lai sẽ được phục vụ bởi một quy trình khác. - Các trình đóng gói tài sản như Django-assetpackager hay Jammit sẽ được cấu hình và biên dịch ngay trong giai đoạn build. - Hệ thống web cần lưu trữ phiên (session) người dùng trong bộ nhớ của quy trình ứng dụng.	chỉ lưu các session ở RAM mà nên cấu hình stateless session với Redis rồi xử lí hàng đợi email tách biệt với web, chạy đọc lập dưới dạng tiến trình riêng biệt, sau đó đồng bộ dữ liệu sau một khoảng thời gian xác đinh.
7	Port Binding	- 12Factor không dựa vào việc tiêm thời gian chạy của máy chủ web vào môi trường thực thi để tạo dịch vụ hướng đến web. Ứng dụng web xuất HTTP dưới dạng dịch vụ bằng cách liên kết với một cổng và lắng nghe các	Trong NodeJS, ứng dụng tự mở cổng bằng các lệnh như: const PORT = process.env.PORT || 3000; và dùng trong cloud,…

		yêu cầu đến trên cổng đó. - Trong quá trình triển khai, một lớp định tuyến xử lý các yêu cầu định tuyến từ tên máy chủ công khai đến các quy trình web có cổng ràng buộc - HTTP không là dịch vụ duy nhất có thể được xuất bằng liên kết cổng. Bất kỳ loại phần mềm máy chủ nào cũng có thể chạy thông qua một quy trình liên kết với một cổng và chờ các yêu cầu đến. - Một ứng dụng có thể trở thành dịch vụ hỗ trợ cho ứng dụng khác bằng cách cung cấp URL tới ứng dụng hỗ trợ dưới dạng xử lý tài nguyên trong cấu hình cho ứng dụng sử dụng bằng phương pháp liên kết các cổng.	
8	Concurrency	- Là mở rộng quy mô thông qua mô hình nhân bản các tiến trình, chia nhỏ tiến trình để mở rộng cho chiều ngang thay vì chiều dọc. - Các quy trình lấy tín hiệu mạnh từ mô hình quy trình Unix để chạy các service daemons . Nhà phát triển có thể	Trong NodeJS sẽ có nhiều cấu trúc khác nhau như web.js, worker.js,… khác nhau. Trong đó web.js dùng để xử lý các HTTP bằng các lệnh require, app.get(),… Còn worker.js dùng để xử lý background jobs với các lệnh hiển thị các thành phần ra giao diện người dùng.

		thiết kế ứng dụng của họ để xử lý nhiều khối lượng công việc khác nhau bằng cách chỉ định từng loại công việc cho một loại quy trình. - Ứng dụng cũng phải có khả năng mở rộng nhiều quy trình chạy trên nhiều máy vật lý. Vì vậy, quy trình này có bản chất share-nothing và phân vùng theo chiều ngang. - Các quy trình ứng dụng không bao giờ demon hóa hoặc ghi các tệp PID và dựa vào trình quản lí quy trình của hệ điều hành để quản lý luồng đầu ra , phản hồi các quy trình bị sập và xử lý các lần khởi động lại và tắt máy do người dùng khởi tạo	
9	Disposability	- Là việc tối đa hóa sự mạnh mẽ với khởi động nhanh và tắt máy nhẹ nhàng. - Các quy trình có thể dùng một lần và có thể được bắt đầu hoặc dừng lại bất cứ lúc nào tạo điều kiện cho việc mở rộng đàn hồi nhanh chóng, triển khai nhanh chóng các thay đổi về mã hoặc cấu hình và	Trong NodeJS có thể tắt sạch sẽ khi bị dừng bởi Docker, Heroku với lệnh server.close() khi khi app nhận ‘SIGTERM’

		tính mạnh mẽ của các lần triển khai sản xuất. - Các quy trình nên giảm thiểu thời gian khởi động (mất vài giây từ khi lệnh khởi chạy được thực thi cho đến khi quy trình được thiết lập) cung cấp tính linh hoạt hơn cho quy trình phát hành và mở rộng quy mô. - Các tiến trình tắt một cách nhẹ nhàng khi chúng nhận được tín hiệu SIGTERM từ trình quản lý tiến trình. Mô hình này ngầm hiểu rằng các yêu cầu HTTP ngắn hoặc trong trường hợp thăm dò dài, máy khách sẽ phải cố gắng kết nối lại liền mạch khi mất kết nối. - Các quy trình cũng phải mạnh mẽ chống lại tình trạng sudden death, nên sử dụng một backend xếp hàng mạnh mẽ, chẳng hạn như Beanstalkd, trả lại các công việc cho hàng đợi khi máy khách ngắt kết nối hoặc hết thời gian chờ. 12Factor được thiết kế để xử lý các lần kết thúc bất ngờ, không bình thường.	

10	Dev/Prod Parity	- Là giữ cho quá trình phát triển, dàn dựng và sản xuất giống nhau nhất có thể - 12factor được thiết kế để triển khai liên tục bằng cách giữ khoảng cách giữa phát triển và sản xuất nhỏ bằng cách thu hẹp khoảng cách thời gian, khoảng cách nhân sự và khoảng cách giữa các công cụ. - Backing services là một lĩnh vực mà tính tương đương giữa dev/prod rất quan trọng, Nhiều ngôn ngữ cung cấp các thư viện giúp đơn giản hóa việc truy cập vào, bao gồm các bộ điều hợp cho các loại dịch vụ khác nhau. - 12factor chống lại sự thôi thúc sử dụng các backing service khác nhau giữa phát triển và sản xuất. Các dịch vụ sao lưu hiện đại như Memcached, PostgreSQL và RabbitMQ không khó cài đặt và chạy nhờ các hệ thống đóng gói hiện đại. Ngoài ra, các công cụ cung cấp khai báo như Chef và Puppet kết hợp với các môi trường ảo nhẹ	Trong NodeJS, dùng file .env như trong production và chạy trong Docker như production, đồng thời đồng bộ lệnh run app: 'npm run start' và cấu hình env đồng bộ, dùng thêm Docker để đồng bộ môi trường bằng các file .yml chưa cấu hình của các thành phần để chạy cả local dev và production staging.

		như Docker và Vagrant cho phép các nhà phát triển chạy các môi trường cục bộ gần giống với môi trường sản xuất và chi phí thấp hơn.	
11	Logs	- Là việc xử lý nhật ký như luồng sự kiện. - 12factor không quan tâm đến việc định tuyến hoặc lưu trữ luồng đầu ra mà mỗi process đang chạy sẽ ghi luồng sự kiện của nó để nhà phát triển xem luồng này ở phía trước của thiết bị đầu cuối và quan sát hành vi của ứng dụng trong quá trình phát triển cục bộ. - Trong quá trình triển khai, mỗi luồng của quy trình sẽ được môi trường thực thi nắm bắt, đối chiếu với tất cả các luồng khác từ ứng dụng và định tuyến đến một hoặc nhiều đích cuối cùng để xem và lưu trữ lâu dài và được quản lý hoàn toàn bởi môi trường thực thi như định tuyến nhật ký bằng Open-source log routers (Logplex,…). - Luồng sự kiện cho một ứng dụng có thể được định tuyến đến một tệp	Trong NodeJS, một số logging chuẩn như: console.log() hay console.error()sẽ in ra stdout/stderr, và nếu app chạy trong Docker/Kubernetes, hệ thống sẽ tự thu thập.

		hoặc được theo dõi qua tail thời gian thực trong một thiết bị đầu cuối và được gửi đến một hệ thống lập chỉ mục và phân tích nhật ký,	
12	Admin Processes	- Là chạy các tác vụ quản lý như các quy trình một lần. - Dev thường muốn thực hiện các tác vụ quản trị hoặc bảo trì một lần cho ứng dụng như: chạy di chuyển cơ sở dữ liệu; chạy một bảng điều khiển để chạy mã tùy ý hoặc kiểm tra các mô hình của ứng dụng so với cơ sở dữ liệu trực tiếp; chạy các tập lệnh một lần được cam kết vào kho lưu trữ của ứng dụng. - Các quy trình quản trị đều chạy trên bản phát hành , sử dụng cùng codebase, config. Mã quản trị phải được gửi cùng với mã ứng dụng để tránh các sự cố đồng bộ hóa. - Các kỹ thuật cô lập phụ thuộc giống nhau nên được sử dụng trên tất cả các loại quy trình. - 12factor ưu tiên mạnh mẽ các ngôn ngữ cung cấp shell REPL ngay khi cài đặt và giúp dễ dàng chạy các tập lệnh	Trong NodeJS, để reset mật khẩu của một người dùng, ta tạo một file scripts/reset-password.js với nội dung: kết nối lại database thông qua module db.js, sử dụng thư viện 'bcrypt' để băm mật khẩu mới, sau đó thực hiện câu lệnh SQL cập nhật mật khẩu theo email được truyền vào dòng lệnh. Script này sử dụng các biến môi trường từ .env như DATABASE_URL,… để đảm bảo nhất quán với app chính. Để chạy, ta chỉ cần gõ lệnh node scripts/reset-password.js user@example.com hoặc thêm sẵn lệnh trong package.json để gọi bằng npm run. Cách tiếp cận này giúp tách biệt các tác vụ quản trị khỏi luồng xử lý chính, dễ kiểm soát, dễ triển khai trong môi trường production (Docker, CI/CD hoặc SSH vào server).

		một lần. Dev gọi các quy trình quản trị một lần bằng lệnh shell trực tiếp bên trong thư mục kiểm tra của ứng dụng và có thể sử dụng ssh hoặc cơ chế thực thi lệnh từ xa khác do môi trường thực thi của quá trình triển khai đó cung cấp.	

Tóm lại phương pháp này là một mô hình lý tưởng cho các ứng dụng cloud-native với các ưu điểm vượt trội như dễ deploy, rollback, scale ngang, bảo trì và quản lý cấu hình. Tuy nhiên, phương pháp này là không phù hợp với tất cả các trường hợp phát triển hệ thống. Phương pháp này thiên về web app, ít phù hợp với desktop. Game và IoT, ít hướng vào bảo mật sâu hơn như xác thực danh tính, mã hóa thông tin,.. khó áp dụng để áp dụng vào các việc tối ưu các hệ thống cũ, phức tạp và phải cẩn thận khi sử dung, không nên áp dụng phương pháp này một cách máy móc.

Chương 3: Clean Architecture

Đây là một mô hình thiết kế phần mềm bằng cách chia ứng dụng thành nhiều lớp theo một vòng tròn đồng tâm với nguyên tắc mọi sự phụ thuộc đều hướng vào phía trung tâm vòng tròn. Điều đó đồng nghĩa với việc các lớp ở bên trong không biết bất cứ điều gì về các lớp bên ngoài. Mục tiên của mô hình này là tạo ra một hệ thống nhiều ưu điểm như: mạch lac, linh hoạt, dễ bảo trì, kiểm thử, mở rộng và độc lập với framework, cơ sở dữ liệu và giao diện người dùng.

Kiến trúc của Clean Architecture chia thành 4 layers chính gồm: Entity, Use cases, Interface Adapters và Framework and Drivers:

- Entity: là layer quan trọng nhất dùng để mô tả các Business Logic,là nơi dev thực hiện giải quyết các vấn đề khi xây dựng app. Layer này không có framework và có thể chạy mà không cần emulator, giúp dễ dàng test, maintain và develop phần Bussiness Logic.
- Use-case: là tập hợp các tule liên quan trực tiếp tới ứng dụng cục bộ.

- Interface Adapter: là tập hợp các adapter phục vụ tương tác với các công nghệ.
- Framework and Drivers: là tập hợp các công cụ về cơ sở dữ liệu và các framework.

Ngoài những ưu điểm vượt trội như trên thì mô hình này cũng tồn tại những hạn chế như: cồng kềnh, phức tạp và tốn chi phí phát triển bản đầu, khó áp dụng do over-engineering và không thể sử dụng tùy ý các frameword do luật dependency inversion. Vì vậy, mô hình nên được áp dụng trong các trường hợp dev có kinh nghiệm với các kiến trúc sâu, quản lí code.