12 Yếu tố tạo nên app xịn xò

* Sau một thời gian lập trình nhận ra một điều: Build một ứng dụng là một chuyện khá khó.
* Thế nhưng, khi ứng dụng bắt đầu có người sử dụng(lên production), ta sẽ phải gặp nhiều vấn đề còn … khó hơn nữa:  
  + Làm sao thêm tính năng, sửa lỗi mà không ảnh hưởng đến ứng dụng đang chạy  
  + Làm sao để những tính năng mới, bug fix có thể được re\_lease nhanh chóng đến tay người dùng  
  + Khi lượng người dùng tăng gấp 5 lần, 10 lần, làm sao để có thể nhanh chóng scale hệ thống  
  + Làm sao để mấy bạn developer mới gia nhập có thể dễ dàng chạy ứng dụng ở local, test và push code.
* Đây là những vấn đề làm đau đầu nhiều team, vì nó đòi hỏi không chỉ kiến thức lập trình, mà còn là kiến thức về system archietecture, operation, qui trình…
* Do vậy trong bài này mình sẽ chia sẻ về Twelve-Factor App. Đây là 12 yếu tố cần thiết để xây dựng một ứng dụng xịn xò, ổn định, dễ mở rộng, dễ deploy.  
  **Twelve-Factor App này có gì hay ho**
* Đây là những kinh nghiệm được tổng hợp từ các developer đã tham gia phát triển về deploy, vận hành vài trăm app trên nền tảng Heroku. Bạn có thể xem 12 yếu tố này là cách để build 1 web app dễ mở rộng, dễ deploy, dễ tiếng cận cho developer mới gia nhập luôn.
* Cá nhân mình thấy những kinh nghiệm này khá hữu ích, nó giải quyết được khá nhiều vấn đề chúng ta thực sự gặp phải khi xậy dựng, vận hành một ứng dụng web.

12 yếu tố này bao gồm   
1. Codebase: Một codebase nằm trong source control, deploy nhiều nơi

2. Dependencies: Dependencies như library/framework/extension phải được ghi rõ ràng, tách biệt từng app

3. Config: Lưu trữ thiết lập vào biến môi trường (environment variable)  
4. Backing services(dịch vụ hỗ trợ): Xem các service đi kèm (database, API, …) như là resource của app.

5. Build, release, run: Tách riêng quá trình release, build và run.

6. Processes(quy trình): App nên chạy dưới dạng stateless processes

7. Port binding(ràng buộc cổng): Một service có thể được access thông qua 1 port cố định

8. Concurrency(Đồng thời): Một app nên được chia tách thành nhiều process nhỏ để tăng concurrency

9. Disposability(Dùng một lần): Process của web app nên sống nhanh, chết vội, để có thể dễ dàng chạy.kill process nhanh chóng.

10. Dev/prod parity (tương đồng): Các môi trường dev/staging/production nên giống nhau hết sức có thể

11. Logs: Logs nên được viết dưới dạng stream ở stduot

12. Admin Processes: Một số task dạng admin (tạo database, fixx dữ liệu) nên được chạy trong cùng môi trường với app đang chạy

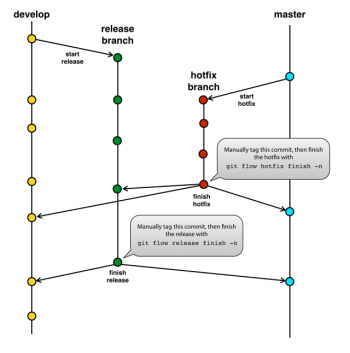
- Nếu các bạn đọc qua thấy không hiểu cũng không sao. Mình sẽ giải thích kĩ hơn và đưa 1 số ví dụ cụ thể cho từng yêu tố nhé.

1. Codebase: Một source code trong source control, có nhiều deploy

- Mỗi app nên chỉ có 1 source code (codebase) duy nhất, nằm trong source control như Git/SVN.

- Mỗi khi code được build và chạy trên một môi trường nào đó, ta gọi đó là bản deploy.

- Code chạy trên môi trường Production gọi là Production de-ploy, chạy trên Staging gọi là Staging deploy. Deploy trên stag-ing có thể có nhiều commit hơn, nằm ở branch khác Production. Những 2 code này đều nằm trong 1 source control.



* Nhờ vậy, khi developer mới gia nhập, ta có thể dễ dàng lấy và đọc source code từ source control.
* Gần đây, khi kiến trúc microservice đang thịnh hành, ta có thể dùng monorepo – 1 repo chứa toàn bộ source code của các service. Hoặc có thể coi mỗi service là 1 app, lưu source code của service đó vào 1 repo riêng.

1. Dependencies: dependencies rõ ràng, tách biệt.  
   - Cho các bạn chưa biết, dependencies tức là các package/thư viện/tool của bên thứ 3 mà app của bạn cần có để chạy được.   
   - Một app cần phải thiết lập rõ ràng những dependency mà nó sử dụng, không lệ thuộc vào các dependency có sẵn của hệ thống. Các dependencies này phải tách biệt trong từng app.  
   - Giả sử thế này, bạn đang viết ứng dụng nhận dạng JAV idol, sử dụng thư viện IdolRec ver 2.1. Thằng bạn của bạn cũng biết ứng dụng nhận diện sử dụng ver 1.3:

+ Cách thức đúng ở đây là: app sẽ ghi rõ mình dùng thư viện IdolRec + version. Thư viện này sẽ bảo vào thư mục app, chạy trong app.

+ Nếu ứng dụng phụ thuộc vào dependencies ngầm trong máy, sẽ dễ dẫn đền tình trạng “It works on my machine” 🡺 Chạy được trên máy dev, nhưng lên Production thì tèo…

+ Nếu không ghi rõ, tách biệt dependencies, khi 2 ứng dụng chạy trên 1 máy sẽ bị conflict vì 2 thư viện khác nhau. Hoặc có thể… Không chạy được vì máy kia chưa xài IdolRec

* Hiện tại, đa phần các ngôn ngữ lập trình đều có cách package manager hỗ trợ chuyện này (NodeJS thì lưu dependencies vào gemfile, C# thì lưu vào Web.config hoặc App.config…)
* Nhờ vậy, khi ta deploy ứng dụng lên server, hoặc developer mới gia nhập pull code về, ta chỉ cần chạy npm install hoặc bun-ble install để tải thư viện về, là ứng dụng có thể chạy bình thường, không cần cài dependencies ngầm.

1. Config: Lưu trữ thiết lập vào biến môi trường

* Như đã nói ở phần 1, một source code có thể được deploy ở nhiều môi trường khác nhau. Mỗi môi trường sẽ có các thiết lập khác nhau như:

+ Connection String tới Database (App staging sễ kết nối tới DB Staging, App Production sẽ kết nói với DB production)  
+ API Key, Token để được gọi tới các service khác.  
+ Hostname, URL của ứng dụng, …

* Cách ngu nhất là hard code toàn bộ các thiết lập này, sau đó sửa lại code lại mỗi lần deploy. Một số cách khác ổn hơn thì lưu các thiết lập này vào 1 file riêng, mỗi môi trường sẽ có 1 file:

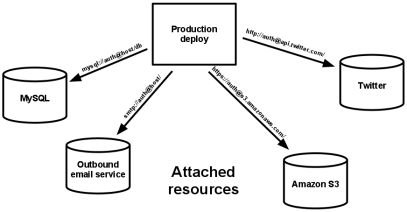
+ Bên C#, ta sẽ có Web.config, web.staging.config chứa các thiết lập Staging, web.production.config chứa các thiết lập Production

+ Bên NodeJS, ta có thể lưu vào file .env, tách thành .env.staging, .env.production

* Cá nhân mình thấy cách lưu file này cũng đã tạm ổn. Tuy nhiên, ta cần phải đảm bảo chỉ một số ít người mới có quyền xem các thiết lập Production, hoặc dễ quên lưu nhầm file vào source control.
* Theo twelve factor app, toàn bộ các thiết lập này nền được bỏ vào biến môi trường. Các biến này có thể dễ dàng được thay đổi lúc deploy.

1. Backing servies: Tách rời các service đi kèm (database, API, …), xem như là re-source của app

* Như mình đã nói ở phần 3, để có 1 ứng dụng chạy được, ngoài trừ server chạy app, ta phải có các server đi kèm như DB server đi kèm như Database Server (SQL, MongoDB), Cache Server (redis memcache), message queue.
* Các service này nên được tách rời, deploy tách riêng với app. Khi ta muốn nâng cấp cache server, hoặc muốn SQL Server trên cloud thay cho server nội bộ để dễ mở rộng hơn; ta đều có thể dễ dàng nâng cấp mà không cần đụng tới app hiện tại.



* Ngoài ra, như đã nói ở điều 3. ta cũng nên lưu kết nối tới các server này vào file config/biến môi trường, khi thay đổi db server, message queue chỉ cần thay đổi các biến này và restart app là xong, không cần phải sửa source code.

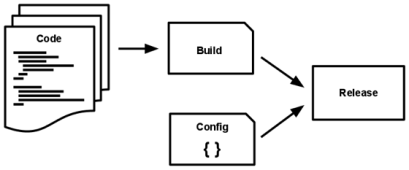
1. Build, Release, Run -> Tách rời các môi trường build, release và chạy app

* Để biển source code thành 1 ứng dụng chạy được, ta thường phải đi qua 3 bước sau:

+ Build: Lấy code từ source control, tải các thư viện (dependencies) cần thiết. Thực hiện build để code thành bi-nary(.jar, .dll) có thể chạy được. (Với một số ngôn ngữ như PHP, NodeJS, Python thì không cần build, nhưng vẫn cần tải thư viện)

+ Release: Kết hợp source code đã được build, cùng với các thiết lập môi trường (config trong phần 3) để gom thành 1 cục có thể chạy được. Chuyển cục này vào server production để có thể chạy.

+ Run: Chạy cục vừa chuyển (bằng lệnh node server.js, python, index.py) trên server production.



* Tại sao phải tách rời 3 bước: