

HỌC VIỆN KỸ THUẬT QUÂN SỰ

VIỆN CÔNG NGHỆ THÔNG TIN TRUYỀN THÔNG

**BÁO CÁO THỰC TẬP**

**Quản lý trang thiết bị công nghệ thông tin**

**cấp trung, lữ đoàn**

|  |  |  |
| --- | --- | --- |
| **Giáo viên hướng dẫn** | **: 1//** | **- Trần Hữu Phi** |
|  |  |  |
| **Nhóm thực hiện** | **: 3/** | **- Trần Hồng Quân** |
|  | **: 4/** | **- Nguyễn Thế Anh** |
|  | **: 4/** | **- Đỗ Đức Trung** |
|  | **: 3/** | **- Phạm Quang Duẩn** |

**Vĩnh Phúc, tháng 01 năm 2024**

# Mở đầu

Giới thiệu về đề tài thực tập: Trình bày mục tiêu, phạm vi và giới hạn của đề tài thực tập.

Đề tài thực tập: Quản lý trang thiết bị công nghệ thông tin cấp trung, lữ đoàn

Mục tiêu của dự án phần mềm là quản lý trang thiết bị CNTT tại đơn vị cấp trung lữ đoàn. Chức năng chính của phần mềm là cho phép người quản trị hệ thống thông qua việc tạo tài khoản và phân quyền người dùng; cho phép người quản lý hồ sơ và người quản lý chỉ huy các cấp xem, thêm mới, sửa được hồ sơ thông tin, báo cáo. Giúp cho quá trình quản lý quản lý trang thiết bị cũng như việc tiếp cận thông tin hệ thống của người quản lý hồ sơ và người quản lý chỉ huy các cấp được thuận tiện, nhanh chóng và chính xác hơn.

Phạm vi gồm các công việc chính phân tích yêu cầu, thiết kế, lập trình, kiểm thử, triển khai phần mềm quản lý trang thiết bị CNTT tại đơn vị cấp trung lữ đoàn. Dự án là kết quả của việc khảo sát yêu cầu, tìm hiểu các tài liệu liên quan đến các chức năng cần thiết của phần mềm quản lí trang thiết bị CNTT tại đơn vị cấp trung lữ đoàn. Dự án là cơ sở giao tiếp giữa các thành viên trong nhóm, là căn cứ để xây dựng các tài liệu khác: tài liệu thiết kế phần mềm, tài liệu thiết kế CSDL, tài liệu testcase…

Đồng thời, Nghiên cứu, tìm hiểu quy trình phát triển và vận hành phần mềm; Sử dụng thành thạo công cụ quản lý dự án phần mềm; Thực hiện lên kế hoạch, phân công công việc cho các thành viên trong nhóm; Xây dựng kịch bản kiểm thử; Lựa chọn một dự án phần mềm/ ứng dụng để áp dụng lý thuyết và quy trình quản lý, phát triển và vận hành; Triển khai trên máy chủ online.

Giới thiệu về DevOps, CI/CD: Trình bày khái niệm, lợi ích, các nguyên tắc và các công cụ của DevOps, CI/CD.

DevOps là một phương pháp tiếp cận phát triển phần mềm kết hợp phát triển (Dev), vận hành (Ops) và các hoạt động liên quan khác. DevOps tập trung vào việc giảm thiểu khoảng cách giữa các đội phát triển và vận hành, đồng thời tự động hóa các quy trình phát triển và triển khai. CI/CD là một tập hợp các thực tiễn và công cụ giúp tự động hóa quy trình phát triển phần mềm từ khi viết mã đến khi triển khai.

DevOps và CI/CD là hai khái niệm có mối quan hệ chặt chẽ với nhau. DevOps là một phương pháp tiếp cận phát triển phần mềm kết hợp phát triển (Dev) và vận hành (Ops), trong khi CI/CD là một tập hợp các thực tiễn và công cụ giúp tự động hóa quy trình phát triển phần mềm từ khi viết mã đến khi triển khai.

# Chương 1: Tổng quan về quy trình phát triển phần mềm theo hướng DevOps

### I. DEVOPS LÀ GÌ? DEVOPS THÀNH CÔNG NHẤT ĐỊNH PHẢI SỞ HỮU 6 KỸ NĂNG VÀ TỐ CHẤT NÀY

DevOps là gì? DevOps là một văn hóa làm việc đề cao sự hợp tác, kéo hai giai đoạn phát triển (development) và vận hành (operations) xích lại gần nhau hơn. Ngoài ra, DevOps Engineer là một trong những ngành được đánh giá là có mức lương cao so với mức [lương ngành công nghệ thông tin](https://itviec.com/blog/bao-cao-luong-it/?utm_source=blog&utm_medium=internal-link&utm_campaign=salary-report-2022&utm_content=dec2022) năm 2023 ở Việt Nam.

#### 1. DevOps là gì?

Chu trình phát triển phần mềm (Software Development Life Cycle) bao gồm hai giai đoạn chính: phát triển và vận hành. Hai giai đoạn này tương đối tách rời nhau, đặc biệt là ở các công ty có quy mô trung bình trở lên.

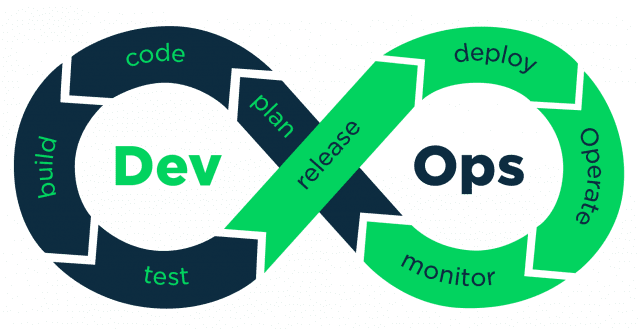
Tuy nhiên, nhằm tối ưu hóa chu trình phát triển phần mềm, giúp sản phẩm IT được release nhanh và thường xuyên hơn, khái niệm DevOps ra đời.

Định nghĩa DevOps là gì?

DevOps là tên gọi mới, là sự kế thừa và phát triển của một quan niệm về phát triển phần mềm đã tồn tại từ khá lâu.

DevOps là sự kết hợp của từ Development (phát triển tính năng sản phẩm) + Operations (vận hành):

* Giai đoạn phát triển (development) bao gồm phần việc của designer, developer, [QA QC](https://itviec.com/blog/qa-la-gi-qc-khac-gi-qa/)…
* Giai đoạn vận hành (operations) có sự tham gia của system engineer, system administrator, operation executive, release engineer, DBA, network engineer, security engineer…



*DevOps là Development (phát triển tính năng sản phẩm) + Operations (vận hành)*

Để cho dễ hình dung, và cũng để trả lời rõ hơn cho câu hỏi “DevOps là gì”, ta cần ngược trở lại lịch sử ngành phần mềm một chút:

* Ở buổi ban sơ của kỷ nguyên máy tính:

Quy trình phát triển phần mềm không hề có sự phân tách rạch ròi giữa hai giai đoạn phát triển (development) và vận hành (operations), nhất là đối với các sản phẩm vừa và nhỏ. Vì là người phát triển sản phẩm, Developer sẽ hiểu rõ về sản phẩm để chọn cách vận hành phù hợp nhất nên anh ta sẽ đảm nhiệm việc develop, đồng thời cũng kiêm luôn việc test, deploy sản phẩm.

* Sau đó, sự bùng nổ về quy mô của các công ty và sản phẩm công nghệ diễn ra:

Từ đó, kéo theo quy mô hệ thống phình ra theo cấp số nhân. Từ một vài server, hệ thống có thể phát triển lên đến hàng chục, hàng trăm, hàng nghìn, hoặc thậm chí hàng triệu server (ví dụ như trường hợp của Google, Facebook).

Yêu cầu chuyên môn hóa trở nên gắt gao, khiến quy trình phát triển phần mềm chia tách thành những giai đoạn riêng biệt. Đây là giai đoạn mà Dev và Ops tách bạch.

Khoảng một thập kỉ trở lại đây, trước nhu cầu phát triển và cải tiến sản phẩm liên tục để đáp ứng thị trường, sự chia tách này lại bộc lộ những nhược điểm rõ rệt.

Ngoài ra, ngành phát triển phần mềm cũng dịch chuyển theo một hướng khác – microservices.

*Microservices: Một sản phẩm lớn được chia tách làm rất nhiều service nhỏ, các service này liên kết với nhau tạo thành một sản phẩm hoàn chỉnh.*

Ví dụ, đối với người dùng, một trang web thương mại điện tử là một sản phẩm hoàn chỉnh. Nhưng trên thực tế, trang web này được gộp lại từ rất nhiều feature như đăng kí, đăng nhập, tìm kiếm.v.v… Mỗi feature này là một service riêng, có thể sử dụng ngôn ngữ lập trình và database riêng.

#### 2. Lợi ích của DevOps

Khi được hỏi “Lợi ích của DevOps là gì?”, anh Minh Tấn chia sẻ, “Cùng với [phương pháp Agile](https://itviec.com/blog/agile-la-gi-scrum-la-gi/?utm_source=blogpost&utm_medium=referral&utm_content=devops-la-gi&utm_campaign=con-duong-su-nghiep-it), DevOps giúp hoàn thiện việc chuyển đổi quy trình phát triển và vận hành phần mềm từ mô hình thác nước (waterfall) sang mô hình phát triển/phát hành liên tục (continuous development/releases)”.

Ngoài ra, những lợi ích chính của DevOps là:

* Tăng cường sự cộng tác chặt chẽ giữa nhóm phát triển (development) và nhóm vận hành (operation), cũng như khả năng làm việc liên chức năng (cross-functional).
* Nâng cao tần suất triển khai (deployment), giúp rút ngắn thời gian phát triển/cải tiến sản phẩm.
* Tận dụng các công cụ tự động hóa, giúp hạn chế rủi ro, giảm tỉ lệ thất bại.
* Thời gian phục hồi sản phẩm nhanh hơn.

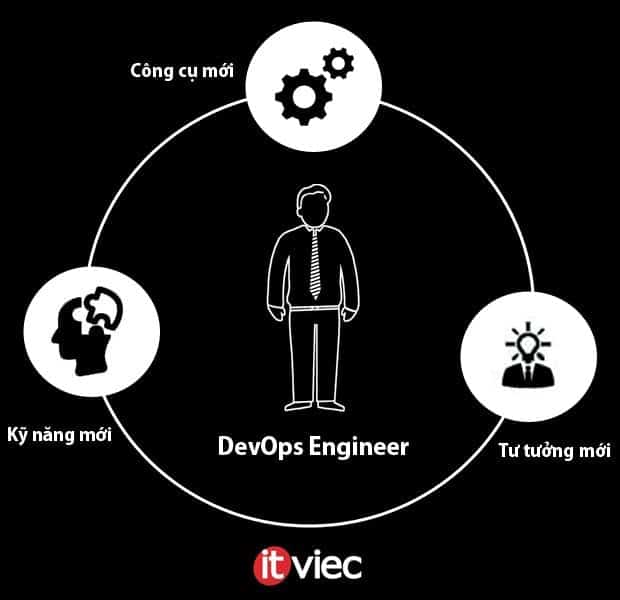
Tất cả đều phục vụ cho mục đích cuối cùng là cải thiện khả năng cung cấp dịch vụ IT một cách nhanh chóng. Từ đó, tăng khả năng cạnh tranh của sản phẩm/doanh nghiệp.

#### 3. Công việc của DevOps Engineer là gì?

Theo anh Đăng Phong, DevOps Engineer là sự kết hợp theo công thức:

*DevOps Engineer = Tư tưởng mới + Công cụ mới + Kỹ năng mới*

Anh diễn giải thêm, nếu ta hiểu được DevOps là gì – Là một văn hóa làm việc mới, một phương thức tiếp cận để thu hẹp khoảng cách giữa quá trình phát triển và vận hành phần mềm thì ta sẽ hiểu được DevOps Engineer là một vị trí nảy sinh do nhu cầu thực tiễn công việc, có thể tạm định nghĩa gồm tư tưởng, công cụ và kĩ năng mới.



#### 4. Vậy công việc của một người làm DevOps là gì?

* Về mặt quy trình, DevOps cần đảm bảo làm thế nào để các bộ phận hợp tác trơn tru thuận lợi hơn
* Về mặt sản phẩm, DevOps cần làm thế nào để các service kết nối và giao tiếp với nhau theo rules hiệu quả, cũng như đảm bảo việc scaling được “êm ái”

***Dựa trên công thức trên, ta có:***

*a. Tư tưởng mới*:

Tư tưởng mới ở đây chính là DevOps Engineer cần cần đặt lợi ích doanh nghiệp, lợi ích sản phẩm lên hàng đầu, đồng thời thấy rằng toàn bộ các team thực chất là cùng một “phe”, cùng chia sẻ lợi ích cũng như rủi ro.

Vậy thì, người làm DevOps Engineer là phải có tư tưởng – mindset đúng đầu tiên.

*Trong hầu hết các công ty/dự án phần mềm, đội ngũ phát triển và vận hành bị chia tách thành nhiều team làm việc tương đối độc lập với nhau: Developer, Tester, Sysadmin.v.v… Cũng từ đó, kiểu tư duy “chúng ta” – “chúng nó” hình thành, tạo nên nhiều xung đột không đáng có, gây ảnh hưởng xấu đến doanh nghiệp/sản phẩm.*

Thay vì đợi team Dev phát triển xong sản phẩm, rồi team Ops mới tham gia vận hành như trước kia. Thì nay, DevOps Engineer nên tham gia ngay từ đầu với đội ngũ phát triển. Nhằm:

* Hiểu sản phẩm hơn, để tối ưu hóa sản phẩm tốt hơn.
* Học ngôn ngữ lập trình mà công ty sử dụng, để nắm được logic code, mài sắc khả năng tư duy, nắm được tiến trình của code chạy như thế nào.v.v…

Khi deploy code, nếu gặp vấn đề ở chỗ nào, DevOps Engineer sẽ có thể chủ động tìm lỗi và fix luôn mà không cần phải chờ developer.

*b. Công cụ mới:*

Nhiệm vụ quan trọng của DevOps Engineer là tự động hóa hệ thống. Cũng nghĩa là, DevOps Engineer cần liên tục tìm hiểu, chọn, và sử dụng các công cụ mới, hoặc thậm chí tự phát triển các automation tool cho công ty. Ví dụ:

* Để hỗ trợ Agile/Scrum thì sử dụng các công cụ như: [JIRA](https://itviec.com/blog/jira-la-gi/), Redmine …
* Để chuẩn bị cho Infrastructure as code, và xây dựng [CI/CD](https://itviec.com/blog/ci-cd-la-gi/) process thì sử dụng công cụ như: Terraform, Ansible, Jenkins, hay Github, Gitlab…

*c. Kỹ năng mới:*

Nhìn chung, công việc chính của DevOps Engineer rất gần với công việc của [Sysadmin](https://itviec.com/it-jobs/system-administrator?utm_source=blogpost&utm_medium=center_post&utm_campaign=linktosite), bao gồm: deploy, optimizing, monitoring, analysis… Điểm khác biệt là:

* DevOps Engineer đòi hỏi nhiều kĩ năng mềm hơn, đồng thời phải biết dùng coding, scripting để automate hệ thống.
* Người làm DevOps Engineer cũng cần tìm hiểu về stack mà sản phẩm công ty đang sử dụng, để có thể cùng review bug, viết những unit test thông thường, và để khi phát triển tiến trình CI/CD thì deploy “êm ái” hơn.

Cho nên, nếu coi DevOps Engineer là Sysadmin “kiểu mới” thì cũng không sai.

Tuy nhiên, Sysadmin “kiểu cũ” rất lười code. Nhưng trong bối cảnh hiện nay, để tiếp tục theo đuổi ngành System thì họ cần phải biết automation.

Cũng có nghĩa, họ cần rèn luyện kỹ năng coding, scripting, và thậm chí học cả những ngôn ngữ lập trình mới theo công nghệ/stack mà nhóm phát triển sản phẩm sử dụng.

#### 5. DevOps cần thành thạo những kỹ năng và tố chất gì?

DevOps cần học nhiều một số ngôn ngữ lập trình cần thiết như: Python, Ruby, Lua Scripting và cả một số tool tùy theo yêu cầu công việc cụ thể.

Dựa trên những chia sẻ của anh Minh Tấn và Đăng Phong, một DevOps muốn thành công cần phải sở hữu những kỹ năng và tố chất sau:

* Kỹ năng lập trình “cứng” là điều bắt buộc

DevOps Engineer thường là vị trí kiêm nhiệm (Developer kiêm nhiệm thêm phần việc operations, hoặc là System Engineer kiêm nhiệm thêm một phần việc của dev, v.v…) chính vì thế một DevOps cần phải có kỹ năng lập trình cứng.

*Ví dụ, Một người là System Engineer kiêm DevOps Engineer. Anh ấy muốn deploy version mới của sản phẩm lên 100 server. Nếu thực hiện việc này thủ công thì sẽ mất rất nhiều thời gian, và không tránh khỏi sai sót.*

*Trong trường hợp deploy thành công 50 con server, còn 50 con thất bại, thì cũng có nghĩa là sản phẩm của mình thất bại. Bởi vì cùng lúc sản phẩm sẽ chạy 2 version khác nhau, mà mình lại không kiểm soát 2 version này được. Muốn khắc phục thì cũng phải có thời gian.*

*Như vậy, để deploy nhanh hơn, hỗ trợ việc back-up, restore, đồng thời giảm thiểu rủi ro, thì với vai trò DevOps Engineer, Anh ấy sẽ viết automated script để ship code tự động lên server.*

Ngôn ngữ lập trình phổ biến cho DevOps Engineer là [Python](https://itviec.com/blog/tai-lieu-lap-trinh-python/), shell script.

Ngoài ra, để Ops, DevOps Engineer cũng cần hiểu sâu, thông thạo về hệ điều hành đang sử dụng (Linux, Docker, v.v…)

* Kỹ năng research tốt

Đặc biệt, người làm DevOps phải có khả năng research tốt để nhanh chóng tìm ra giải pháp, xử lý tình huống.

Một ví dụ vô cùng trực quan, dễ hiểu:

*Nếu triển khai services trên nền tảng on premise. Một ngày “đẹp trời” nào đó, hệ thống gặp vấn đề, Bạn muốn move toàn bộ sản phẩm của mình lên cloud. Tuy nhiên, có rất nhiều cloud, nên chọn dùng cloud nào cho phù hợp?*

*Rõ ràng, trong tình huống này, nếu khả năng research không tốt, không nhanh chóng tìm ra cách để move toàn bộ mọi thứ đang chứa trên on premise lên cloud, thì sản phẩm của mình bị đình trệ rồi.*

*Hoặc, trong DevOps có rất nhiều bài toán hóc búa liên quan đến phần network, I/O, infra system, v.v… Một người cứng về develop nhưng không hiểu sâu về phía Infra thì khi làm DevOps sẽ gặp rất nhiều khó khăn. Người đó buộc phải research về Infra để phục vụ cho công việc.*

* Sự cẩn thận, tỉ mỉ là quan trọng nhất

DevOps Engineer thường sẽ đảm nhiệm những công việc như migrate data cho công ty nên họ cần đề cao sự tỉ mỉ. Khi đó, chỉ cần xảy ra một sai sót nhỏ, ví dụ như sai 1 IP server, thì sẽ gây ảnh hưởng đến toàn hệ thống.

* Luôn đặt lợi ích tập thể lên hàng đầu

Đây là tiêu chí quan trọng nhất, vì DevOps sinh ra là để giải quyết mâu thuẫn.

Tiêu chí này thể hiện qua những việc rất nhỏ nhặt cụ thể. Ví dụ như cách DevOps Engineer suy nghĩ, tổ chức, cấu trúc code/thư mục như thế nào, chia sẻ những best practices, … để mọi người có thể cùng nhau đọc và hiểu code đó, cùng tham gia được với mình.

* Nhìn nhận mọi thứ từ nhiều khía cạnh

Mâu thuẫn giữa nhóm phát triển và vận hành thường nảy sinh từ sự khác biệt về góc nhìn. Vì vậy, DevOps Engineer là người cần nhìn nhận mọi thứ từ nhiều khía cạnh, để khách quan, sáng suốt hơn, biết “thông cảm” hơn.

Cụ thể, khi deploy mà code không chạy, thì DevOps Engineer cần xem xét kĩ: vấn đề nằm ở phía code hay phía môi trường.

*Ví dụ, trường hợp làm với*[*Laravel*](https://itviec.com/blog/framework-la-gi-top-framework-pho-bien-nhat/)*(*[*PHP Framework*](https://itviec.com/blog/framework-php/)*), file config là .env. Sysadmin không có kinh nghiệm thì dễ mắc sai lầm là chỉ lấy phần code đó xuống và chạy và lỗi thì loay hoay và thường nghĩ do code.*

*Trong khi, lẽ ra cần phải hiểu những cấu hình liên quan đến môi trường và cách thức hoạt động của Laravel, và phải tác động vào file .env trước đã.*

* Giao tiếp và hợp tác tốt với các team khác

Trong công việc, mối quan hệ tốt thì cái gì cũng dễ dàng, và ngược lại.

Để xây dựng mối quan hệ tốt, anh Phong nghĩ nên gạt chức danh lead/manager gì gì đó qua một bên, để giao tiếp với họ như là bạn bè bình đẳng.

Bạn tôn trọng họ, thì họ cũng sẽ tôn trọng, dễ dàng chia sẻ với bạn hơn. Mà khi xảy ra chuyện, cần nhờ vả thì họ cũng dễ dàng đồng ý hơn.

Ngoài ra, một DevOps Engineer cũng cần có:

* Kinh nghiệm với system và IT operations, quản lý dữ liệu.
* Nắm vững các tiến trình (CI/CD) và công cụ tự động hóa.
* Khả năng sử dụng nhiều công nghệ và mã nguồn mở, coding/scripting.  
  Liệt kê ra thì dông dài như vậy, nhưng tóm gọn lại, tiêu chí để “chọn” DevOps Engineer của anh Phong là: những sysadmin-coder tài năng, có kinh nghiệm, đồng thời hiểu rõ:
* Hai mục tiêu quan trọng của việc viết phần mềm là “kiếm tiền” và “giao hàng”.
* Tất cả chúng ta – manager, dev, tester, DBA, Sysadmin, v.v… đều cùng một phe, cùng cố gắng để đạt mục tiêu chung: cung cấp phần mềm chất lượng tốt, đáng tin cậy, và đem lại lợi nhuận kinh doanh cho khách hàng.

DevOps trước hết là vấn đề mindset, nên bạn cần phải “đả thông tư tưởng” trước đã. Bạn có thể tìm hiểu “DevOps là gì” từ sách báo, qua các[trao đổi trên diễn đàn](https://www.reddit.com/r/devops/), v.v…

Bạn cũng cần học một số ngôn ngữ lập trình cần thiết cho DevOps như: Python, [Ruby](https://itviec.com/blog/ruby-on-rails-la-gi/), Lua Scripting.

Tiếp đến, bạn có thể lên các trang web tuyển dụng để đọc mô tả công việc của DevOps. Từ đó, bạn sẽ biết thị trường đang cần những kĩ năng gì, xu hướng dùng những tools gì.

#### 6. Quy trình

Các giai đoạn chính của quy trình phát triển phần mềm theo hướng DevOps

Quy trình phát triển phần mềm theo hướng DevOps có thể được chia thành các giai đoạn chính sau:

* Phân tích và thiết kế
* Xây dựng và kiểm thử
* Tích hợp và triển khai

Giai đoạn phân tích và thiết kế

Giai đoạn phân tích và thiết kế bao gồm các hoạt động sau:

* Tìm hiểu nhu cầu của người dùng
* Phân tích các yêu cầu của người dùng
* Thiết kế hệ thống

Trong giai đoạn này, các đội phát triển và vận hành cần làm việc cùng nhau để hiểu rõ nhu cầu của người dùng và thiết kế hệ thống đáp ứng được nhu cầu đó.

Giai đoạn xây dựng và kiểm thử

Giai đoạn xây dựng và kiểm thử bao gồm các hoạt động sau:

* Viết mã
* Tích hợp mã
* Kiểm thử đơn vị
* Kiểm thử tích hợp
* Kiểm thử hệ thống

Trong giai đoạn này, các đội phát triển sẽ viết mã và tích hợp mã của họ với nhau. Các đội kiểm thử sẽ thực hiện các bài kiểm thử để đảm bảo rằng ứng dụng hoạt động đúng như mong đợi.

#### 7. Tài liệu DevOps tham khảo

* Sách Devops – [The Phoenix Project](https://www.amazon.com/Phoenix-Project-DevOps-Helping-Business/dp/0988262592): Cuốn sách này rất nổi tiếng, cá nhân mình đặc biệt thích vì nó giúp mình có mindset đúng về công việc (luôn hướng đến lợi ích tập thể, lợi ích của doanh nghiệp).
* [DevOps Tutorials](https://www.tutorialspoint.com/devops_tutorials.htm): Cung cấp những kiến thức nền tảng cũng như những công cụ phục vụ cho công việc của DevOps.
* [DevOps Việt Nam](https://www.facebook.com/groups/1023642501057106/): Hội nhóm mở trên Facebook, dành riêng cho các DevOps tại Việt Nam. Bạn có thể tìm thấy rất nhiều thông tin từ chia sẻ tài liệu, kinh nghiệm cho đến tuyển dụng DevOps tại đây.
* [What is DevOps](http://radar.oreilly.com/2012/06/what-is-devops.html): Bài viết rất thú vị về sự ra đời cũng như những nguyên lý của DevOps.

### II. CÔNG NGHỆ GIT VÀ GITHUB

#### 1. Quy trình Phát triển phần mềm với Git:

Git Workflow (Luồng làm việc Git):

Git Flow: Sử dụng các nhánh như master, develop, feature branches, release branches, và hotfix branches để quản lý quá trình phát triển.

GitHub Flow: Tập trung vào việc sử dụng nhánh main (hoặc master) để phát triển và triển khai.

#### 2. Công nghệ và Các công cụ hỗ trợ:

Git Clients: Git command line hoặc các giao diện người dùng đồ họa như SourceTree, GitKraken.

Hosting Services: GitHub, GitLab, Bitbucket cung cấp kho lưu trữ và quản lý dự án dựa trên Git.

Continuous Integration (CI) và Continuous Deployment (CD): Sử dụng công cụ như Jenkins, CircleCI, TravisCI để tự động hóa quá trình kiểm thử và triển khai.

#### 3. Quy trình làm việc:

Commit Guidelines: Sử dụng chuẩn commit message để dễ dàng theo dõi lịch sử thay đổi.

Code Review: Sử dụng Pull Requests (PRs) để đề xuất, xem xét và chấp nhận sửa đổi mã nguồn trước khi hợp nhất vào nhánh chính.

Best Practices (Thực hành tốt nhất):

Branch Strategy: Sử dụng nhánh feature riêng cho mỗi tính năng hoặc sửa lỗi, giúp dễ dàng theo dõi và kiểm soát.

Regular Commits: Thực hiện commit thường xuyên và nhất quán để giữ lịch sử rõ ràng và chi tiết.

Testing and Validation: Thực hiện kiểm thử liên tục để đảm bảo chất lượng mã nguồn.

#### 4. Backup và Recovery:

Remote Repositories: Sử dụng kho lưu trữ từ xa (remote repositories) như GitHub để đảm bảo sao lưu và phục hồi dữ liệu.

### III. DOCKER LÀ GÌ? TẠI SAO LẬP TRÌNH VIÊN NÊN BIẾT CÁCH SỬ DỤNG DOCKER?

Thời gian gần đây, Docker đang ngày càng phổ biến trong giới công nghệ và có xu hướng phát triển mạnh mẽ trong tương lai. Các ông lớn công nghệ như [Adobe, Twitter, Netflix,](https://stackshare.io/docker)… hiện cũng đang sử dụng Docker trong quá trình phát triển các sản phẩm của họ.

#### 1. Tại sao lại sử dụng Docker?

Đặt tình huống bạn đang làm một project cùng với team về Machine Learning bằng Python và muốn gửi phần code của mình cho đồng đội trong team để chạy thử. Đồng đội bạn hì hục cài đặt các thư viện cần cho project như scikit-learn, pandas, tensorflow, keras, … và xử lí các xung đột linh tinh xảy ra mất gần một tiếng đồng hồ nhưng code chạy vẫn bị lỗi. À, hóa ra trên máy bạn sử dụng Python 3.7 còn máy đồng đội lại sử dụng Python 3.5 thế mới đau. Thế là đồng đội bạn lại tốn thêm một khoảng thời gian đáng kể để cài đặt Python 3.7 trên máy.

Cuối cùng mất gần cả buổi sáng thì code trên máy đồng đội cũng đã chạy, nhưng kết quả output lại hơi khác biệt so với trên máy bạn, tại sao lại như vậy? Có khá nhiều nguyên nhân nhưng chủ yếu là do sự khác biệt về hệ điều hành, máy bạn chạy Ubuntu nhưng máy đồng đội lại chạy MacOS chẳng hạn.

Sau sự việc trên, bạn và đồng đội tự hỏi liệu có công cụ nào có thể giúp chạy code của người khác trên máy mình một cách nhanh chóng, không gây ra xung đột giữa các dependencies và sử dụng chung một hệ điều hành hay không?  
Câu trả lời đó chính là **Docker**.

#### 2. Sơ lược về lịch sử hình thành

a. Server vật lý (Physical Server)

Nhiều năm về trước, khi chưa có Docker, các project ở công ty hay doanh nghiệp thường được deploy trên các server vật lý theo cấu trúc khá đơn giản:

Mỗi server vật lý thường bao gồm: bộ nhớ, CPU, GPU, kết nối mạng, ổ cứng và hệ điều hành để chạy các ứng dụng.

Các server vật lý thường có chi phí khá cao, rất mạnh mẽ nhưng lại không tận dụng hết được sức mạnh của mình, dù cấu hình phần cứng khá khủng nhưng lại chỉ có thể cài đặt được một hệ điều hành. Điều đó cũng dẫn đến việc hình thành nên các server bằng công nghệ ảo hóa (Virtual Server).

b. Server ảo (Virtual Server)

Mỗi máy chủ vật lý có thể được dùng để chạy nhiều project, mỗi project ta có thể cho chạy trong một máy ảo hay còn được gọi là **Virtual Server**.

Các Virtual Server được tạo ra bằng công nghệ Virtualization (được sử dụng trong các phần mềm như VMware, VirtualBox, Azure VM, …) sẽ chạy trên một máy chủ vật lý, mỗi Server ảo được cài đặt một hệ điều hành riêng (Guest OS) sử dụng các tài nguyên phần cứng được cấp phát từ máy chủ vật lý. Việc quản lý và điều phối các tài nguyên từ Server vật lý sao cho hợp lý được thực hiện bởi lớp Hypervisor như trên hình.

Đến đây, việc tạo ra các môi trường độc lập để tiện cho việc deploy dự án đã giảm bớt một phần chi phí và trở nên khá dễ dàng, nhưng Virtual Server vẫn còn khá nhiều hạn chế. Việc cấp phát cố định các tài nguyên phần cứng (nghĩa là mỗi Virtual Server sẽ có một phần cứng riêng) sẽ gây khó khăn cho việc chia sẻ dữ liệu, tài nguyên giữa các Virtual Server. Điều này thường dẫn đến việc không tận dụng hết tài nguyên từ máy chủ, ví dụ máy ảo của bạn được cấp phát 7 GB RAM nhưng bạn chỉ sử dụng khoảng 4 GB, như vậy là 3 GB còn lại coi như để không, không thể chia sẻ cho các máy ảo khác đang cần.

Ngoài ra, mỗi Server ảo đều chứa một hệ điều hành riêng nên dung lượng của mỗi Server ảo khá nặng, làm mất đi tính di động cần có để chia sẻ các môi trường triển khai ứng dụng giữa những người phát triển với nhau.

Những hạn chế trên cũng dẫn đến sự ra đời của công nghệ tiếp theo – Containerization.

c. Containerization

Công nghệ Containerization sẽ tạo ra trên máy chủ vật lý các máy con gọi là **Container** chứa ứng dụng hoặc project và các thư viện cần thiết để chạy ứng dụng hoặc project đó.

Việc sử dụng Container khắc phục hoàn toàn những bất cập đã nêu ở phía trên của Server vật lý và Server ảo. Container rất nhẹ, thời gian khởi động nhanh và có tính di động cao, việc chia sẻ các môi trường triển khai dự án giữa những người dùng trở nên dễ dàng hơn bao giờ hết.

Những lợi ích trên có được là do các Container sử dụng cơ chế chia sẻ tài nguyên và dùng chung nhân của hệ điều hành ở máy chủ (Host OS), chứ không cần cấp phát cho mình một hệ điều hành riêng (Guest OS) như trên Virtual Server.

Vậy giờ ta muốn sử dụng công nghệ Containerization thì phải sử dụng công cụ nào? Như ở phần đầu mình có đề cập, câu trả lời chính là **Docker.**

#### 3. Docker là gì?

***Docker****là một công cụ mã nguồn mở sử dụng công nghệ containerization cho phép chúng ta tạo ra các môi trường đóng gói (các container) trên máy tính mà không làm ảnh hưởng đến môi trường hiện tại của máy.*

Nghe qua có vẻ Docker khá giống với việc sử dụng các máy ảo, vậy điều gì làm cho Docker ngày càng trở nên phổ biến? Đây là những **nguyên nhân**:

* Các Docker Container chia sẻ nhau chung một nguồn tài nguyên từ máy chủ vật lý, không phải tách biệt như ở các máy ảo, giúp **tối ưu hóa thời gian và không gian** của máy chủ.
* Docker rất **nhẹ và nhanh**, bạn có thể tạo và khởi động các Docker Container chỉ trong vài giây.
* **Cộng đồng sử dụng Docker rất đông đảo** thường xuyên chia sẻ các Docker Image (mình sẽ giải thích Docker Image ở phần sau) trên Docker Hub. Bạn chỉ cần lên đấy tìm các Image cần thiết và pull về để sử dụng.
* Với Docker, rất dễ để liên kết các container với nhau để tạo thành một ứng dụng, làm cho nó **dễ dàng scale, update** các thành phần độc lập với nhau.

Bất kì công cụ nào cũng có những **nhược điểm** đúng không? Theo mình nhược điểm của Docker đó là không thể chạy trực tiếp trên **Windows** và **MacOS**, trên 2 nền tảng hệ điều hành này Docker sẽ tạo ra một máy ảo **Linux** để chạy các Container, điều này có thể ảnh hưởng đến hiệu năng khi sử dụng.

**Đối tượng dùng Docker?**Docker được tạo ra để mang lại lợi ích cho các lập trình viên và đặc biệt là những người quản trị hệ thống, giúp cho lập trình viên tập trung code mà không cần phải lo lắng việc code chạy ngon trên máy mình nhưng lại fail trên máy khác.

#### 4. Tìm hiểu về Docker Engine:

Đây là thành phần cốt lõi của Docker, **Docker Engine** sử dụng cấu trúc Client-Server, được cài đặt trên máy chủ vật lý. Đây là 3 thành phần cơ bản của Docker Engine:

* **Server:** Còn được gọi là **Docker Daemon**, dùng để tạo và quản lí các Images, Container, Networks, …
* **Rest API:** Chỉ thị cho Docker Daemon cần làm những gì.
* **Command Line Interface (CLI):** Dùng để nhập những câu lệnh từ phía client.

#### 5. Kiến trúc của Docker:

a. Docker Client:

Người dùng Docker có thể tương tác với Docker thông qua **Docker** **Client**. Khi các câu lệnh Docker được chạy, **Client** sẽ gửi chúng đến cho **Docker Daemon**để xử lí. **Docker Client** có thể giao tiếp với nhiều **Daemon**.

b. Docker Daemon

Là thành phần core, lắng nghe API request và quản lý các **Docker Object** (Images, Conatainer, Network, Volume). Docker daemon host này cũng có thể giao tiếp được với Docker daemon ở host khác.

c. Docker Registry

Là một nơi chứa các **Docker** **Images**. Người dùng có thể tạo ra các images của mình và tải lên đây hoặc tải về các images được chia sẻ. Nổi tiếng nhất có lẽ là Docker Hub, ngoài ra bạn có thể tạo Registry cho riêng mình.

d. Docker Objects

Đây là những thành phần bạn hay gặp nhất ghi sử dụng Docker.

* **Image:** là các read-only template (chỉ đọc được chứ không thể sửa đổi), dùng để tạo ra các Container thông qua câu lệnh docker run. Môi trường lập trình gồm các ứng dụng, thư viện, hệ điều hành, … sẽ được đóng gói vào trong một Image. Có thể tạo Docker Image bằng cách buid từ Dockerfile hoặc commit từ Docker Container. Docker Image thường được lưu trữ trên Docker Hub hoặc ở registry của người dùng.
* **Container:** khi bạn chạy Image sẽ tạo ra các Container chứa môi trường lập trình đã được đóng gói vào trong Image, bạn có thể tạo ra nhiều Container từ 1 Image. Hiểu theo ngôn ngữ OOP thì Image là 1 class và Container chính là Instance của lớp đó.
* **Network:** cung cấp private network (VLAN) để các Container trên một Host có thể liên lạc được với nhau, hoặc các Container trên nhiều Hosts có thể liên lạc được với nhau (Multi-host Networking).
* **Volume:** dùng để lưu trữ các dữ liệu độc lập với vòng đời của Container và giúp cho các Containers có thể chia sẻ dữ liệu với nhau và với Host.

#### 6. Quy trình làm việc cơ bản với Docker

* Dùng **Docker File** hoặc commit từ **Container** đang sử dụng để tạo ra **Image** chứa môi trường lập trình phù hợp cho project của bạn.
* Gửi **Image** sang cho đồng đội trong team để chạy project bằng cách tạo ra **Container**, việc khởi tạo các **Container** mất khoảng vài giây.
* Bạn có thể upload các **Image** bạn đã tạo ra lên **Docker Hub** để chia sẻ cho cộng đồng.
* Từ **Docker Hub**, bất kì người dùng nào cũng có thể pull **Image** của bạn về để chạy tạo ra các **Container**.

# Chương 2: Phân tích yêu cầu

### 1. Phân tích yêu cầu nghiệp vụ

Các quy nghiệp vụ cần đáp ứng :

Nghiệp vụ “Cập nhật thông tin TTBCNTT”

Nghiệp vụ “Cập nhật thông tin đơn vị”

Nghiệp vụ “Cập nhật thông tin thiết bị cho cá nhân mượn”

Nghiệp vụ “Cập nhật thông tin thiết bị cho đơn vị mượn”

Nghiệp vụ “Tìm kiếm, thống kê”

Nghiệp vụ “Cập nhật thông tin thiết bị cho đơn vị mượn”

### 2. Phân tích yêu cầu của phần mềm

Phần mềm QLTTBCNTT phải đáp ứng được các yêu cầu chung như sau:

Quản trị người dùng theo quyền truy cập: admin sẽ có quyền “Quản trị tài khoản”, các người dùng khác có quyền “Xuất nhập thông tin”. Mọi người dùng có thể thay đổi mật khẩu truy cập vào hệ thống.

* Xử lí quá trình truy cập thông tin người dùng.
* Xác định và theo dõi TTB CNTT: Dự án bao gồm việc xác định và theo dõi tất cả các Trang thiết bị công nghệ thông tin trong tổ chức, bao gồm máy tính cá nhân, máy chủ, thiết bị mạng, phần mềm, và thiết bị lưu trữ.
* Thiết lập hệ thống quản lý TTB CNTT: Phát triển và triển khai hệ thống quản lý TTB CNTT để ghi nhận thông tin về các tài sản, bao gồm thông tin cơ bản, biên chê hiện tại, tình trạng, và thông tin liên quan đến bảo mật.
* Phân loại TTB CNTT: Xác định và phân loại TTB CNTT dựa trên mức độ quan trọng và nhạy cảm. Điều này bao gồm việc gán các nhãn và quy tắc quản lý cụ thể cho từng loại tài sản.
* Theo dõi quá trình cho đơn vị khác hoặc quân nhân khác mượn TTBCNTT
* Bảo trì và cập nhật dữ liệu TTB CNTT: Dự án sẽ bao gồm quy trình bảo trì và cập nhật định kỳ dữ liệu về TTB CNTT để đảm bảo rằng thông tin luôn là chính xác và cập nhật.
* Tạo báo cáo và thông tin liên quan: Phát triển và triển khai các báo cáo và thông tin liên quan về TTB CNTT để hỗ trợ quyết định quản lý và tuân thủ các công văn, chỉ thị của Bộ Quốc phòng về CNTT.

Chương 2 được cụ thể hóa trong Tài liệu Đặc tả yêu cầu người dùng kèm theo.

# Chương 3: Thiết kế hệ thống

Thiết kế tổng quan hệ thống: Xác định kiến trúc hệ thống, các thành phần của hệ thống, mối quan hệ giữa các thành phần.

Thiết kế chi tiết hệ thống: Thiết kế chi tiết các thành phần của hệ thống, bao gồm:

Thiết kế giao diện người dùng

Thiết kế cơ sở dữ liệu

Thiết kế kiến trúc phần mềm

Chương 3 được cụ thể hóa trong Tài liệu Đặc tả phần mềm, tài liệu thiết kế phần mềm, tài liệu thiết kế cơ sở dữ liệu kèm theo.

# Chương 4: Triển khai hệ thống

### 1. Xây dựng mã nguồn:

Xây dựng mã nguồn cho các thành phần của hệ thống, sử dụng các công cụ và kỹ thuật DevOps.

### 2. Tích hợp mã nguồn:

Tích hợp mã nguồn của các nhà phát triển thường xuyên, sử dụng các công cụ và kỹ thuật CI/CD.

Giai đoạn tích hợp và triển khai bao gồm các hoạt động sau:

* Tích hợp liên tục (CI)
* Xây dựng và triển khai liên tục (CD)

Trong giai đoạn này, các đội phát triển và vận hành sẽ làm việc cùng nhau để tự động hóa quy trình tích hợp và triển khai ứng dụng.

CI/CD là một tập hợp các thực tiễn và công cụ giúp tự động hóa quy trình phát triển phần mềm từ khi viết mã đến khi triển khai. CI/CD bao gồm các bước sau:

* Tích hợp mã nguồn (CI): Tự động tích hợp mã nguồn của các nhà phát triển thường xuyên.
* Xây dựng và kiểm thử (CD): Tự động xây dựng và kiểm thử ứng dụng.
* Triển khai (CD): Tự động triển khai ứng dụng lên môi trường sản xuất.

CI/CD là một thành phần quan trọng trong quy trình phát triển phần mềm theo hướng DevOps. CI/CD giúp các tổ chức đạt được nhiều lợi ích, bao gồm:

* Tăng tốc độ phát triển: CI/CD giúp giảm thiểu thời gian cần thiết để tạo ra các bản phát hành mới.
* Tăng chất lượng: CI/CD giúp phát hiện và sửa lỗi sớm hơn, từ đó giảm thiểu rủi ro phát hành phần mềm lỗi.
* Tăng khả năng mở rộng: CI/CD giúp các tổ chức có thể dễ dàng mở rộng quy mô hoạt động phát triển phần mềm.

Các kỹ thuật và công cụ CI/CD

Jenkins là một công cụ tự động hóa mã nguồn phổ biến. Jenkins cung cấp một tập hợp các công cụ và plugin giúp tự động hóa các quy trình CI/CD, bao gồm tích hợp mã nguồn, xây dựng ứng dụng, kiểm thử ứng dụng và triển khai ứng dụng. Jenkins có thể được sử dụng để tự động hóa các quy trình CI/CD cho nhiều loại dự án phần mềm, bao gồm ứng dụng web, ứng dụng di động và ứng dụng máy tính để bàn.

GitHub là một hệ thống quản lý mã nguồn tích hợp CI/CD. GitLab cung cấp một nền tảng tích hợp cho quản lý mã nguồn, phát triển và triển khai ứng dụng. GitLab bao gồm các tính năng CI/CD tích hợp sẵn, giúp các tổ chức có thể dễ dàng bắt đầu với CI/CD.

Docker là một công cụ tạo và quản lý các ứng dụng đóng gói. Docker giúp các tổ chức tạo ra các ứng dụng có thể triển khai được, có thể được vận hành ở bất kỳ môi trường nào. Docker là một công cụ quan trọng trong CI/CD, vì nó giúp các tổ chức tự động hóa quy trình xây dựng và triển khai ứng dụng.

Kubernetes là một hệ thống quản lý container. Kubernetes giúp các tổ chức tự động hóa việc triển khai và quản lý các ứng dụng container. Kubernetes là một công cụ quan trọng trong CI/CD, vì nó giúp các tổ chức triển khai ứng dụng lên môi trường sản xuất một cách nhanh chóng và hiệu quả.

Dưới đây là một số ví dụ cụ thể về cách sử dụng các kỹ thuật và công cụ CI/CD này:

Jenkins có thể được sử dụng để tự động hóa quy trình tích hợp mã nguồn bằng cách sử dụng một plugin như GitLab CI. Plugin này sẽ tự động tích hợp mã nguồn của các nhà phát triển vào một kho lưu trữ GitLab.

GitLab có thể được sử dụng để tự động hóa quy trình xây dựng ứng dụng bằng cách sử dụng một plugin như Gitlab Runner. Plugin này sẽ tự động xây dựng ứng dụng từ mã nguồn đã được tích hợp.

Docker có thể được sử dụng để tự động hóa quy trình triển khai ứng dụng bằng cách sử dụng một plugin như Gitlab Runner. Plugin này sẽ tự động triển khai ứng dụng được đóng gói bằng Docker vào một môi trường sản xuất.

Việc lựa chọn các kỹ thuật và công cụ CI/CD phù hợp sẽ phụ thuộc vào nhu cầu cụ thể của từng tổ chức. Tuy nhiên, các kỹ thuật và công cụ được đề cập ở trên là những lựa chọn phổ biến và hiệu quả.

### 3. Xây dựng và triển khai ứng dụng:

Xây dựng và triển khai ứng dụng lên môi trường sản xuất, sử dụng các công cụ và kỹ thuật DevOps được kể trên.

# Chương 5: Kết quả và đánh giá

### 1. Kết quả thực hiện dự án:

* Yêu cầu về chức năng:
  + Hệ thống đã cung cấp đầy đủ các chức năng theo yêu cầu của người dùng.
  + Hệ thống đáp ứng được các yêu cầu về tính năng mới của người dùng.
* Yêu cầu về hiệu năng:
  + Hệ thống hoạt động ổn định và đáp ứng được các yêu cầu về hiệu năng.
* Yêu cầu về khả năng mở rộng:
  + Hệ thống có thể mở rộng để đáp ứng nhu cầu tăng trưởng của người dùng.

### 2. Kết quả sử dụng các công nghệ và quy trình CI/CD

* Sử dụng công cụ Jenkins để tự động hóa quy trình tích hợp mã nguồn và xây dựng ứng dụng.
* Sử dụng công cụ Docker để tạo các ứng dụng có thể triển khai được.
* Sử dụng công cụ Kubernetes để tự động hóa việc triển khai ứng dụng lên môi trường sản xuất.

### 3. Các khó khăn, thách thức

* Khó khăn về mặt thời gian: mức độ hiểu và làm theo còn hạn chế về các khái niệm và công nghệ sử dụng trong quá trình thực tập
* Khó khăn trong việc phối hợp giữa các đội phát triển và vận hành.
* Khó khăn trong việc lựa chọn các công cụ và kỹ thuật phù hợp.
* Khó khăn trong việc triển khai các thay đổi mới lên môi trường sản xuất.

### 4. Kết luận

Ý kiến đánh giá, đề xuất

* Ưu điểm và nhược điểm của dự án.
* Lời khuyên cho các dự án tương tự.
* Các đề xuất để cải thiện dự án.

Một số lưu ý khi viết báo cáo thực tập

* Báo cáo cần được trình bày rõ ràng, mạch lạc, dễ hiểu.
* Báo cáo cần có đầy đủ các nội dung cần thiết.
* Báo cáo cần được sử dụng các ngôn ngữ chuyên ngành chính xác.
* Báo cáo cần được trình bày theo một bố cục thống nhất.

Dưới đây là một số mẫu báo cáo thực tập theo hướng DevOps mà bạn có thể tham khảo:

* Báo cáo thực tập DevOps tại Google
* Báo cáo thực tập DevOps tại Amazon
* Báo cáo thực tập DevOps tại Microsoft

Bạn có thể sử dụng các mẫu báo cáo này làm cơ sở để viết báo cáo thực tập của mình. Tuy nhiên, bạn cần lưu ý chỉnh sửa các mẫu báo cáo này cho phù hợp với dự án thực tập của mình.