

**ĐẠI HỌC NGOẠI NGỮ - TIN HỌC TP. HỒ CHÍ MINH  
KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO ĐỀ TÀI LÝ THUYẾT  
HỌC PHẦN KIỂM ĐỊNH CHẤT LƯỢNG PHẦN MỀM**

**TÍCH HỢP LIÊN TỤC VÀ KIỂM THỬ  
(CONTINUOUS INTEGRATION AND TESTING)**

|  |  |  |
| --- | --- | --- |
| Giảng viên hướng dẫn: (ThS) Cô Bùi Thị Thanh Tú | | |
|  |  | |
| Nhóm 7 | 15DH110369 | Huỳnh Đỗ Huy Tập |
| Các sinh viên thực hiện | 15DH110065 | Nguyễn Hải Duy Đức |
|  | 15DH110043 | Huỳnh Vũ Lê |

# MỤC LỤC

[MỤC LỤC 3](#_Toc529897810)

[1. GIỚI THIỆU ĐỀ TÀI 5](#_Toc529897811)

[2. CƠ SỞ LÝ THUYẾT 6](#_Toc529897812)

[2.1 Tích hợp liên tục 6](#_Toc529897815)

[2.2 Kiểm thử và kiểm thử tự động 8](#_Toc529897816)

[2.3 Chiến lược Kiểm thử tự động 10](#_Toc529897817)

[2.4 Kiểm tra liên tục (Continous Inspection) 12](#_Toc529897818)

[2.5 Build và Phát hành liên tục (Continous Builds and Delivery) 12](#_Toc529897819)

[3. DEMO 14](#_Toc529897820)

[3.1 Nội dung demo 14](#_Toc529897822)

[3.2 Kết quả demo 17](#_Toc529897823)

[4. KẾT LUẬN 19](#_Toc529897824)

[5. THAM KHẢO 20](#_Toc529897825)

# GIỚI THIỆU ĐỀ TÀI

Ứng dụng (application) xuất hiện ngày càng nhiều và đa dạng, giải quyết được nhiều bài toán khác nhau từ đơn giản tới phức tạp, đáp ứng nhu cầu vô hạn của con người.

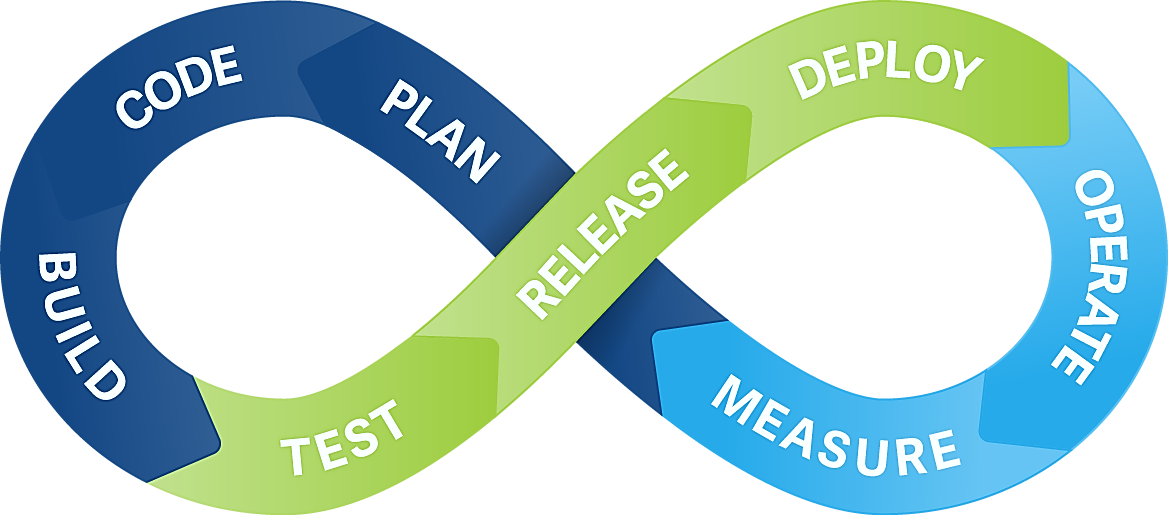
Một ứng dụng tốt bản thân nó đã đáp ứng được một phần nhu cầu của người dùng nhờ khả năng hoạt động chính xác trong một thời gian dài, dễ sử dụng… Các phần mềm không ngừng được sinh ra, cải thiện và phát triển mở rộng, kéo theo đó là hàng loạt công cụ và quy trình để đảm bảo quá trình xây dựng và phát hành từng phiên bản phần mềm luôn được tối ưu nhất, thời gian ngắn nhất, thực hiện với chi phí tối thiểu và giá trị mang lại là tối đa.

Các công cụ và dịch vụ quản lý cấu hình và phiên bản ra đời như git, github, mercury, bitbucket… đã mang lại làn gió mới trong những dự án phát triển sản phẩm phần mềm. Tuy nhiên, dự án càng phát triển, số lượng thành viên tham gia dự án gia tăng thì càng phức tạp trong quá trình “tích hợp” (integration), “kiểm thử” và “build”. Chính vì điều đó, quy trình Agile cùng công nghệ Tích hợp liên tục (Continuous Integration – CI) xuất hiện, như một giải pháp tuyệt vời dẹp bỏ những phiền toái kể trên, đóng góp không nhỏ cho việc quản lý chất lượng của một dự án phần mềm.

# CƠ SỞ LÝ THUYẾT



## Tích hợp liên tục

Quá trình tích hợp phần mềm không phải là một vấn đề mới, và nó không hẳn là vấn đề của các dự án nhỏ lẻ, không có nhiều lệ thuộc vào các hệ thống hay phần mềm khác, nhưng với độ lớn và phức tạp ngày càng gia tăng (ví dụ như chỉ cần một người khác tham gia vào dự án), thì nhu cầu tích hợp và đảm bảo các thành phần (component) hoạt động cùng với nhau ngày càng nhiều. Việc chờ đợi đến khi gần hoàn thành dự án mới tích hợp sẽ dẫn đến hàng loạt các vấn đề về chất lượng phần mềm, khiến chi phí phát triển gia tăng và đình trệ dự án.

Tích hợp liên tục giúp cho việc nhận diện các nguy cơ về tích hợp nhanh hơn, và khiến chúng phát sinh chậm hơn trong quá trình phát triển phần mềm, ví dụ như:

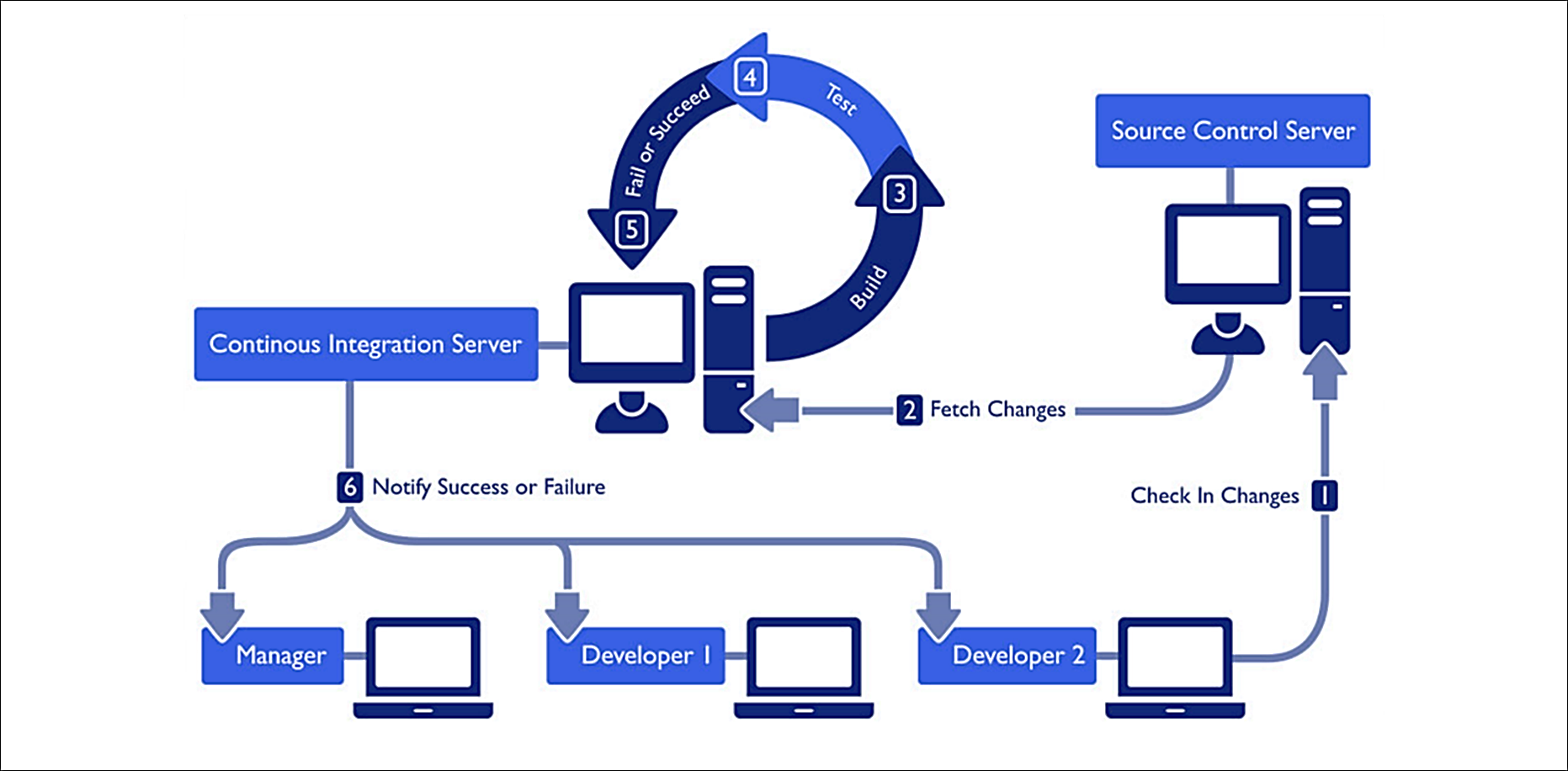
* “Nó chạy trên máy của tôi” – nhưng chưa chắc gì nó chạy trên máy của anh ta hay cô ta.
* Cơ sở dữ liệu phần mềm bất đồng bộ.
* Việc thiết đặt các thông số của dự án không thống nhất.
* Test thiếu.
* …

Trong quá trình tích hợp liên tục:

* Các thành viên phát triển các phần nhỏ của sản phẩm trên máy của họ trước khi Commit code lên Remote Repository, việc này đảm bảo rằng mọi thay đổi của họ không làm gián đoạn quá trình build tích hợp.
* Các thành viên kiểm tra và Commit code của họ lên Remote Repository ít nhất một lần mỗi ngày.
* Build tích hợp diễn ra vài lần trong một ngày trên một hệ thống tách biệt.
* 100% các test phải vượt qua trước khi Build.
* Một bản Build chỉ được Release hành khi nó vượt qua tất cả các tests.
* Sửa các bản Build không thành công là ưu tiên cao nhất.
* Vài thành viên sẽ theo dõi báo cáo sinh ra trong quá trình Test/Build, tìm kiếm các vùng trong dự án có thể cải thiện.

Kịch bản của Tích hợp liên tục trải qua 4 bước cơ bản:

1. Một thành viên Commit code lên Remote Repository, ngay lúc đó, Remote Repository ghi nhận sự thay đổi.
2. Sau khi quá trình Commit hoàn tất, máy chủ CI kiểm tra các thay đổi xảy ra ngay trên Remote Repository, tải về bản sao phiên bản mới nhất từ Repository, thực thi các Script Build, Test, và thu thập Report (việc này chính là “tích hợp” phần mềm).
3. Máy chủ CI sau đó sẽ phản hồi báo cáo, kết quả build thành công hay thất bại dựa trên bản build mới nhất về những thành viên quản lý chất lượng của dự án.
4. Và máy chủ CI lại tiếp tục theo dõi những thay đổi trên Remote Repository.

Quá trình này thực hiện khép kín liên tục.

Hình 1 - Cách tích hợp liên tục hoạt động

## Kiểm thử và kiểm thử tự động

Một phần mềm được xem là “chất lượng” nếu nó thoả mãn các tiêu chí như tính bảo trì, khả năng mở rộng, tính bảo mật, hiệu năng, dễ sử dụng và hoạt động chính xác trong một thời gian dài. Đảm bảo “chất lượng” của sản phẩm phần mềm và của cả quá trình phát triển sản phẩm luôn diễn ra đúng chính là nhiệm vụ của lĩnh vực Kiểm định chất lượng.

Từ việc lên kế hoạch và chiến lược đảm bảo tiêu chuẩn chất lượng, thực hiện và kiểm tra việc thực hiện theo đúng kế hoạch đảm bảo chất lượng đã đề ra, kiểm tra, đo lường, phân tích các số liệu chất lượng thu thập được, từ đó điều chỉnh và thay đổi quy trình cho phù hợp với từng sản phẩm.

Trong đó, việc kiểm thử (testing) đóng vai trò lõi trong suốt các hoạt động đảm bảo chất lượng phần mềm. Quá trình kiểm thử sẽ cố gắng kiểm tra hành vi của phần mềm phù hợp với những đặc tả và yêu cầu đã đặt ra trước đó. Dò tìm khiếm khuyết (defect) và loại bỏ chúng sẽ giúp chúng ta đạt được mục tiêu chất lượng. Khái niệm “kiểm thử” (testing) rất thường được dùng để nói đến việc bảo đảm chứng minh chất lượng của phần mềm.

Với quy trình Agile tích hợp CI, ta có thể chia khâu kiểm thử làm 4 nhóm chính:

1. Tự động (Automated): Unit tests, Component tests.

Unit tests/Component tests đóng vai trò lõi trong kiểm thử sử dụng quy trình Agile, Unit tests sẽ kiểm chứng (verify) chức năng của từng hàm, phương thức hay đối tượng trong khi Component tests sẽ kiểm chứng hành vi của một phần hệ thống như một nhóm các classes cung cấp một vài dịch vụ hay các thành phần bên ngoài hệ thống như cơ sở dữ liệu, hệ thống tập tin, kết nối mạng… Cả hai loại test được thực hiện tự động.

1. Tự động và Thủ công (Automated & Manual): Functional test, System test, ví dụ: Story test, prototypes, Simulations.

Functional test: có thể xem như là những test về nghiệp vụ, xác thực những tính năng, chức năng mà khách hàng mong muốn. Những test này dựa trên những User story thực tế do khách hàng cung cấp, các test chạy ở mức Functional, mỗi test kiểm chứng một điều kiện thoả mãn nghiệp vụ, các test có thể thực hiện tự động hoặc thủ công.

1. Thủ công (Manual): Exploratory testing (chẩn đoán), Scenarios (kịch bản), Usability testing (có thể sử dụng), User Acceptance testing, Alpha/Beta.

Acceptance test, Exploratory test hay Alpha/Beta sẽ kiểm tra việc có hiểu sai hay không yêu cầu của khách hàng, hoặc đội phát triển sản phẩm chưa đáp ứng được mong muốn thực sự của khách hàng. Các test này sẽ bắt chước thực tế của nghiệp vụ, được thực hiện thủ công, và chỉ con người mới làm được.

1. Dùng tools: test các vấn đề về công nghệ, hiệu năng và tải, test bảo mật và “ility” testing (test rỗng).

Kiểm thử trong Tích hợp liên tục/Agile dễ nhận ra nhất chính là kiểm thử tự động (A), kiểm thử tự động và thủ công (B), nó đảm bảo code của toàn bộ dự án luôn build được và luôn chạy đúng (pass tất cả các test case).

Có nhiều lý do để “tự động”:

* Test dài và nhiều.
* Test thủ công dễ bị lỗi.
* Giải phóng con người khỏi việc test, để họ tập trung vào việc khác.
* Test tự động cho ra phản hồi sớm và thường xuyên.
* …

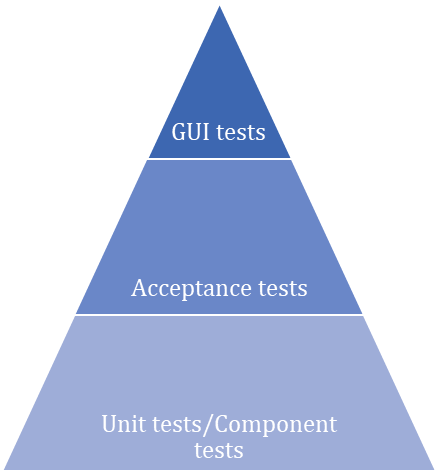
## Chiến lược Kiểm thử tự động

Test tự động trong Agile có thể được chia làm 3 nhóm như hình bên. Nhóm bên dưới chính là nền tảng hỗ trợ cho các nhóm bên trên còn lại.

Tầng thấp nhất chính là tập hợp số lượng lớn các Unit tests/Component tests tự động. Chúng ta cố gắng viết càng nhiều test ở tầng này càng tốt, vì các tests này thường phản hồi kết quả nhanh và thường xuyên hơn.

Tầng giữa chứa đa số là các Functional tests tự động, kiểm chứng rằng mọi thứ chúng ta xây dựng là “đúng”. Các tests ở tầng này có thể gồm các Story test, Acceptance tests, và được diễn dịch sao cho các khách hàng có thể hiểu, chính vì vậy chúng thực hiện nhiều việc hơn những Unit tests, hoàn thành chậm hơn và phản hồi chậm hơn, nhưng vẫn nhanh hơn các thao tác trên giao diện người dùng.

Tầng trên cùng hầu như ít xuất hiện việc tự động hoá. Các tests ở tầng này đều hoàn thành thông qua giao diện đồ hoạ người dùng (GUI), các tests được viết sau khi hoàn thành việc code. Thường thì chi phí để viết các test ở tầng này rất lớn vì hầu như các thành phần giao diện luôn thường xuyên thay đổi, và thao tác trên giao diện luôn chậm hơn các tests ở các tầng dưới, có khi mất đến hàng giờ để hoàn thành. Chúng ta cố gắng giảm thiểu số lượng các tests ở tầng này.

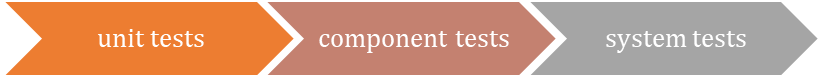


Test thủ công

Hình 2 - Kim tự tháp "tự động"

Tuy nhiên, không phải lúc nào chúng ta cũng có thể tự động hoá:

* Các Usability tests cần người dùng thực sự dùng thử sản phẩm. Việc tự động có thể giúp ích trong việc cài đặt kịch bản thành một chuỗi các thao tác, nhưng để đánh giá kết quả thì chỉ con người mới đủ khả năng để thực hiện.
* Các Exploratory tests có thể thực hiện nhanh hơn bởi các scripts nhưng nó cần người tester dày dạn kinh nghiệm để thiết kế và thực hiện. Kết quả chẩn đoán được dùng để cải thiện việc phát triển trong tương lai, điều này không thể thực hiện được bằng tự động.
* Các tests không bao giờ thất bại.
* Các tests chỉ cần chạy một lần.



Hình 3 - Thứ tự thực hiện các loại tests

Các tests chạy nhanh thì được thực hiện trước!

Việc kiểm thử và công nghệ tích hợp liên tục có thể làm giảm số khiếm khuyết của phần mềm, nhưng sự thật thì, khiếm khuyết vẫn xảy ra. Chính vì vậy, chúng ta có thể viết và chạy các test case sao cho kết quả thu được luôn thất bại, liên tục thực hiện cho tới khi các khiếm khuyết được sửa chữa và test thành công.

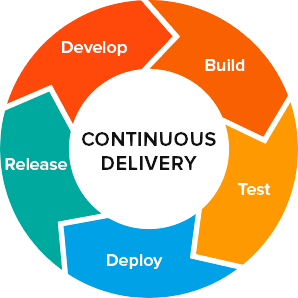
## Kiểm tra liên tục (Continous Inspection)

Bên cạnh việc kiểm thử, chúng ta còn phải kiểm tra (inspection) mã nguồn. Việc này, theo truyền thống, được 2 hoặc nhiều hơn 2 người trực tiếp thực hiện, bằng cách đọc và phân tích code, đặc tả, hay các tài liệu liên quan đến việc phát triển phần mềm và cố gắng loại bỏ các khiếm khuyết về mặt mã nguồn lẫn tài liệu đặc tả liên quan.

Bản thân các công cụ như IDE đều có chức năng kiểm tra và sửa lỗi cú pháp theo thời gian thực nên những người thực hiện vai trò inspector chỉ cần tập trung vào các vấn đề logic và khái niệm.

Ngoài ra, có những công cụ khác hỗ trợ việc kiểm tra và kiểm soát độ phức tạp của mã nguồn dựa trên việc phân tích các số liệu thống kê được như: số lượng classes, methods, số dòng code chưa được comment, lặp code… VD: JavaNCSS, Ndepend, Simian, PMD-CPD…

## Build và Phát hành liên tục (Continous Builds and Delivery)

Build tự động và lặp lại việc Build tự động. Test tự động và lặp lại các Tests tự động. Kiểm tra liên tục và tích hợp Cơ sở dữ liệu liên tục. Một chuỗi các công việc kể trên tạo thành một môi trường tích hợp liên tục hiệu quả, lợi ích của nó là chúng ta có thể phát hành phần mềm (hoạt động được) tại bất kỳ thời điểm nào, trên bất kỳ môi trường nào. Vì nếu không phát hành thì coi như chúng ta chẳng có gì cả!

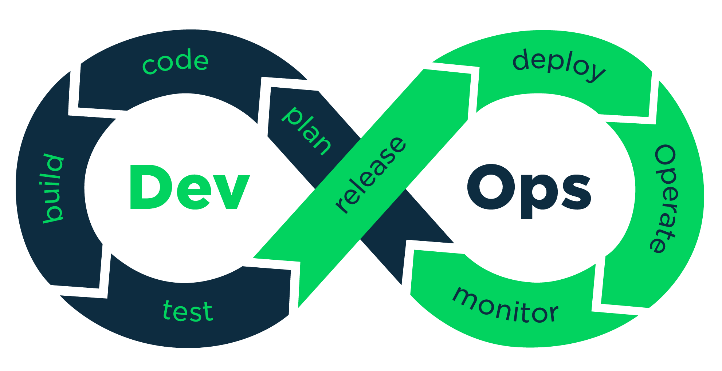
Các bước của một quá trình deploy điển hình:

1. Gán nhãn cho các tài nguyên trong repository.
2. Tạo môi trường sạch.
3. Tạo nhãn cho bản build - được build trực tiếp trên repository và cài đặt lên máy đích.
4. Xác nhận các test chạy thành công.
5. Tạo báo cáo phản hồi
6. Nếu cần, có thể quay về bản build trước đó.

# DEMO



## Nội dung demo

Triển khai một ứng dụng web tĩnh bằng công nghệ Tích hợp liên tục và Phát hành liên tục, trong đó có thực hiện việc test tự động và lấy báo cáo trước khi release thành công một phiên bản ứng dụng.

Môi trường phát triển ứng dụng:

* Ngôn ngữ lập trình: JavaScript
* Framework: React 16
* IDE: VisualStudio Code
* Môi trường thực thi: NodeJS v8.11.2
* Công cụ quản lý gói: npm v6.4.1
* Thư viện và công cụ test: jest, enzyme

Các công cụ hỗ trợ khác:

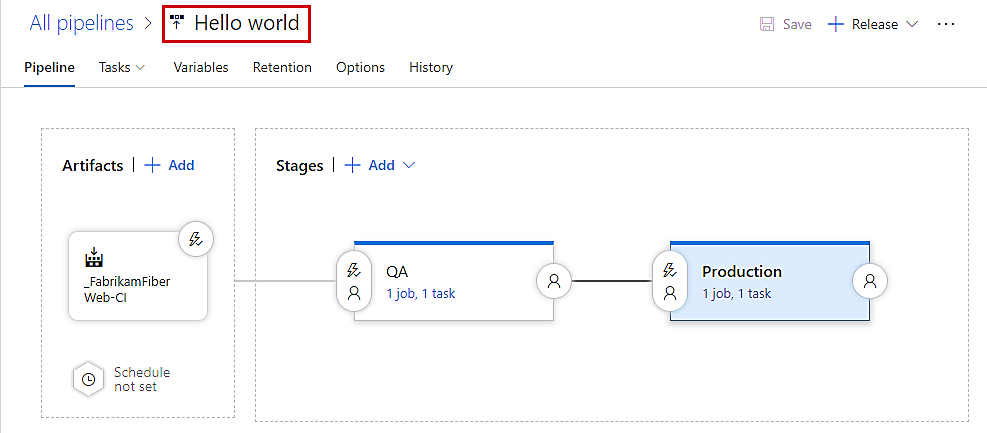
* Công cụ Quản cấu hình và phiên bản: GitHub
* Công cụ Tích hợp liên tục và Phát hành liên tục: MS Azure DevOps
* Sử dụng dịch vụ cơ sở dữ liệu NoSQL và xác thực user: Google FireBase

Mọi thay đổi từ mỗi thành viên tham gia dự án được commit và push lên các nhánh của riêng mình (hoặc lên các nhánh feature khác nhau), có thể tích hợp bằng công cụ quản lý cấu hình và phiên bản, hoặc sử dụng công cụ tích hợp liên tục từ dịch vụ CI.

Azure DevOps là dịch vụ đám mây của Microsoft, cung cấp giải pháp toàn diện từ khởi tạo dự án, quản lý cấu hình, theo dõi và quản lý dự án theo quy trình Agile/Scrum, cho đến tích hợp liên tục, triển khai và phát hành liên tục, đảm bảo chất lượng phần mềm… một cách linh hoạt, trực quan bằng GUI hoặc bằng dòng lệnh.

Về công nghệ CI, Azure DevOps cho phép chúng ta tạo những ống lộ trình (pipeline), mỗi ống có nhiều Job Agent (có thể chạy tuần tự hoặc song song), mỗi Job Agent chứa nhiều task riêng biệt và chạy tuần tự. Tại bất kỳ task trong Job Agent nào, nếu việc thực thi thất bại, thì bản Build xem như thất bại, một bản Build thành công bắt buộc phải passed hết tất cả các tasks. Mã nguồn để tiến hành Build có thể lấy trực tiếp từ công cụ quản lý cấu hình và phiên bản ngay trong DevOps hoặc có thể liên kết đến các dịch vụ bên thứ ba như GitHub, BitBucket… Mỗi ống lộ trình đều có thể theo dõi việc thay đổi theo thời gian thực trên tất cả các nhánh hoặc giới hạn một vài nhánh của repository, ghi nhận log thực thi, các kết quả báo cáo từ các tests…

Về công nghệ CD, Azure DevOps cho phép chúng ta tạo những ống lộ trình (pipeline) với xuất phát điểm là một hoặc nhiều Artifacts (các bản Builds) và kết thúc bằng một chuỗi các giai đoạn - Stages, trong đó giai đoạn Deploy nằm ở bước cuối cùng. Các Stages chủ yếu là Automated Tests (các Functional Tests, System Tests, Acceptance Tests, GUI Tests) tuỳ vào kế hoạch quản lý chất lượng. Tương tự như các ống Build, 1 Stage gồm nhiều Job Agents, một Job Agents có thể có nhiều Tasks.



Hình 4 - Một ống lộ trình Release điển hình

Các ống Build và Release trên Azure DevOps hiện đã cung cấp hàng loạt môi trường, công cụ và phần phụ trợ để có thể triển khai các ứng dụng Web (.Net, Java, NodeJS, Python…) và các ứng dụng di động. Ngoài GUI chúng ta còn có thể tự định nghĩa một ống bằng tập định nghĩa tuần tự YAML.

## Kết quả demo

Với môi trường phát triển ứng dụng như trên, khởi tạo và phát triển dự án theo cấu trúc thư mục như hình bên. Trong đó thư mụcsrc chứa toàn bộ mã nguồn của dự án.

Toàn bộ dự án từ bước khởi tạo ban đầu đều được commit và push trên remote repository nhánh master: <https://github.com/DohutaDA2/team7.git>

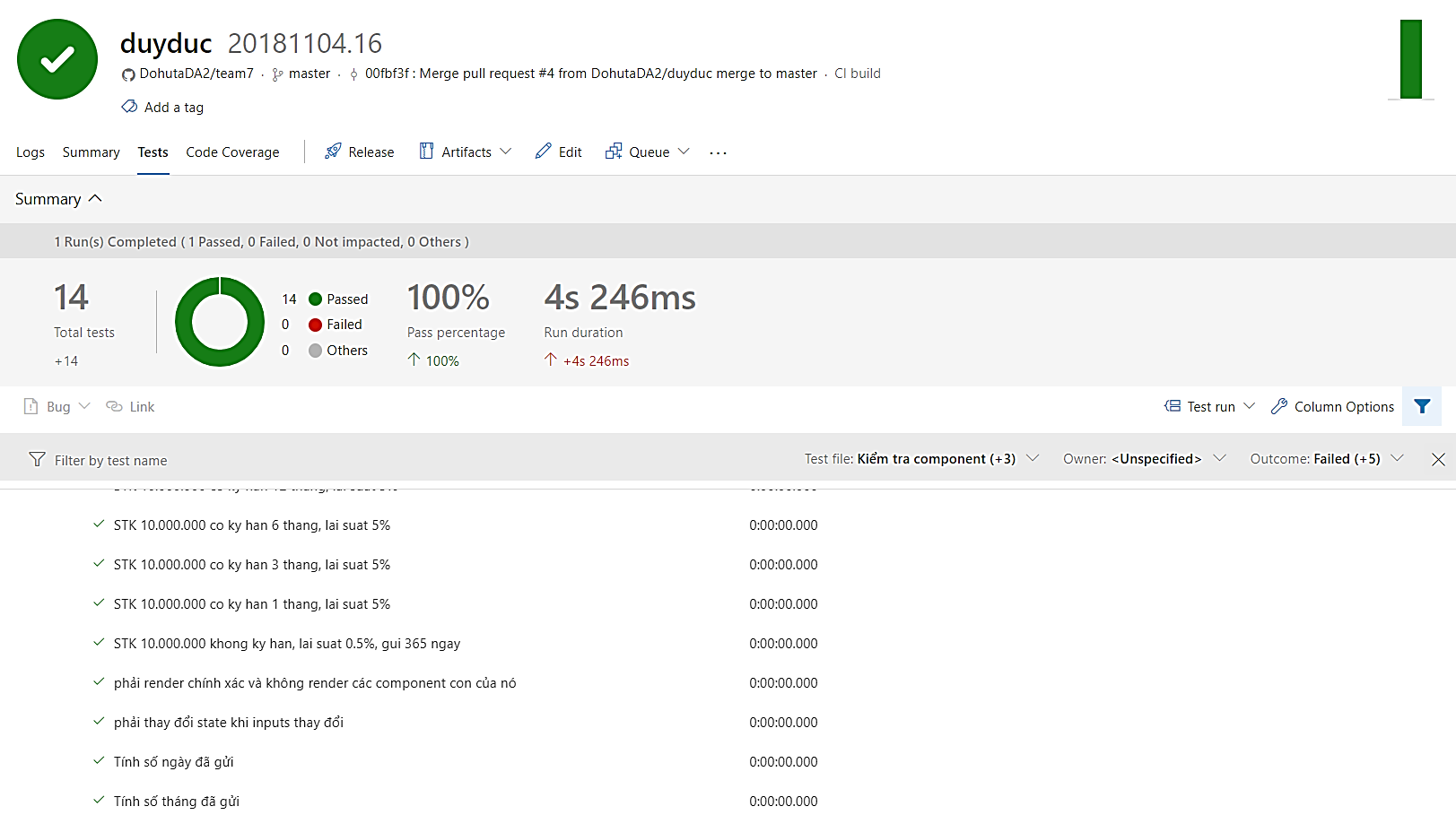
Khi đã thiết lập đầy đủ cả hai ống lộ trình:

* Ống Build: các thông số như repository và nhánh của nó được theo dõi, trigger trên repository - khi nào thì Build, chạy các task liên quan, test task, và giải phóng bản Build.
* Ống Release: stage thu thập bản Build đã giải phóng, stage chạy tests tự động, và stage deploy (thiết lập điểm cuối server mà ứng dụng deploy lên đó).
* Chúng ta đã có được một flow khép kín của công nghệ Tích hợp liên tục, mỗi bản Build và Release được đánh version tự động. Và với bất kỳ thay đổi nào dù là nhỏ nhất thì cũng kích hoạt flow này hoạt động.

Các thông số, báo cáo từ các test tasks mà dịch vụ DevOps giúp chúng ta thu thập và thống kê gồm có: kết quả run các unit tests, kết quả run code coverage tests, trước khi Build thành công và kết quả chạy automated tests trước khi Release thành công. Các kết quả này có thể xem ở dạng tóm tắt tổng thể của từng ống lộ trình hay của từng bản Build/Release riêng biệt trong mỗi ống.

Bản Release được phát hành, deploy và được giữ lại (retain) cho đến khi có một bản Release mới hơn thay thế, các bản Release đều có thể “roll-back”.

Kết hợp với quy trình Agile/Scrum và các công cụ đi kèm, Azure gần như hỗ trợ hoàn hảo cho việc đảm bảo chất lượng và tiến độ của một dự án.



Hình 5 - Báo cáo cụ thể của bản Build số 20181104.16 trên nhánh master, ống build tên duyduc

Ngoài Azure chúng ta còn có sự lựa chọn các dịch vụ CI/CD khác như Kubernetes, Jenkins, GitLabCI, CircleCI…

# KẾT LUẬN

Tích hợp liên tục mang lại những lợi ích không thể phớt lờ. Rất nhiều dự án ngày nay sử dụng công nghệ tích hợp liên tục để triển khai nhờ những đặc tính ưu việt của nó, kết hợp với các công cụ bổ trợ và nền tảng đám mây, đã giúp quy trình đảm bảo chất lượng của một phần mềm diễn ra trơn tru, nhanh và hiệu quả hơn, giúp cho chất lượng của các sản phẩm đầu cuối tốt hơn.

Tuy nhiên, khiếm khuyết của nó không phải là không có. Tích hợp liên tục yêu cầu những thành viên tham gia dự án phải có những kinh nghiệm mới, như sử dụng thành thạo công cụ quản lý cấu hình và phiên bản, hiểu rõ họ cần phải phát triển dự án như thế nào trong môi trường tích hợp liên tục, các testers phải có kinh nghiệm trong việc viết các tests tự động, các nhà quản lý phải hiểu rõ thành viên của mình cần phải biết sử dụng những tools mới với những kiến trúc triển khai phức tạp. Chưa kể hạ tầng để triển khai một dự án tích hợp liên tục cần những phần cứng mạnh mẽ.

# THAM KHẢO

**Introduction to Software Testing**

Nhà xuất bản Cambridge University Press

Tác giả: Paul Ammann, Jeff Offut.

**Agile Testing: A Practical Guide for Testers and Agile Teams**

Nhà xuất bản Addison-Wesley

Tác giả: Lisa Crispin, Janet Gregory

**Continuous Integration: Improving Software Quality and Reducing Risk**

Nhà xuất bản Addison-Wesley

Tác giả: Paul M. Duvall, Steve Matyas và Andrew Glover