**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

🙞🙜🙞🙜🕮🕮🕮🙞🙜🙞🙜

**NGÔ HẢI ĐĂNG**

**TRẦN GIA HUY**

**ĐỒ ÁN CHUYÊN NGÀNH**

**NGHIÊN CỨU VÀ TRIỂN KHAI MÔ HÌNH MÁY HỌC ĐỂ PHÁT HIỆN NHỮNG MẪU LẬP TRÌNH KÉM AN TOÀN TRONG QUY TRÌNH PHÁT TRIỂN VÀ VẬN HÀNH PHẦN MỀM**

**(DEVSECOPS)**

**RESEARCHING AND DEPLOYING A MACHINE LEARNING MODEL TO DETECT INSECURE PROGRAMMING PATTERNS IN THE SOFTWARE DEVELOPMENT AND OPERATION PROCESS****(DEVSECOPS)**

**KỸ SƯ NGÀNH MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG DỮ LIỆU**

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**NGÔ HẢI ĐĂNG - 20521157**

**TRẦN GIA HUY - 20520552**

**ĐỒ ÁN CHUYÊN NGÀNH**

**NGHIÊN CỨU VÀ TRIỂN KHAI MÔ HÌNH MÁY HỌC ĐỂ PHÁT HIỆN NHỮNG MẪU LẬP TRÌNH KÉM AN TOÀN TRONG QUY TRÌNH PHÁT TRIỂN VÀ VẬN HÀNH PHẦN MỀM**

**(DEVSECOPS)**

**RESEARCHING AND DEPLOYING A MACHINE LEARNING MODEL TO DETECT INSECURE PROGRAMMING PATTERNS IN THE SOFTWARE DEVELOPMENT AND OPERATION PROCESS****(DEVSECOPS)**

**KỸ SƯ NGÀNH MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG DỮ LIỆU**

**GIẢNG VIÊN HƯỚNG DẪN**

**THS. ĐẶNG LÊ BẢO CHƯƠNG**

**LỜI CẢM ƠN**

Xin bày tỏ lòng biết ơn chân thành đến ThS. Đặng Lê Bảo Chương - Giảng viên Khoa Mạng máy tính và Truyền thông Trường Đại học Công nghệ Thông tin đã định hướng đề tài, động viên, giúp đỡ nhóm tác giả bằng tất cả sự tận tâm và nhiệt huyết trong suốt thời gian nghiên cứu, hoàn thành đề tài này.

Cuối cùng, nhóm tác giả xin bày tỏ lòng biết ơn đối với gia đình và bạn bè đã luôn giúp đỡ, động viên nhóm tác giả hoàn thành đề tài.

Do những điều kiện chủ quan và khách quan chắc chắn khóa luận không thể tránh khỏi những thiếu sót, nhóm tác giả rất mong nhận được những ý kiến đóng góp của Thầy Cô và các bạn.

Trân trọng cảm ơn.

TP. Hồ Chí Minh, ngày … tháng … năm …

Nhóm tác giả

**MỤC LỤC**

[CHƯƠNG 1. GIỚI THIỆU 7](#_Toc156677356)

[1.1. Lý do chọn đề tài 7](#_Toc156677357)

[1.2. Mục tiêu nghiên cứu 7](#_Toc156677358)

[1.3. Đối tượng nghiên cứu 7](#_Toc156677359)

[1.4. Phạm vi nghiên cứu 8](#_Toc156677360)

[1.5. Những đóng góp của đồ án 8](#_Toc156677361)

[1.6. Cấu trúc của đồ án 8](#_Toc156677362)

[Chương 1: Giới Thiệu 8](#_Toc156677363)

[Chương 2: Cơ sở lý thuyết 9](#_Toc156677364)

[Chương 3: Phân tích và thiết kế mô hình 9](#_Toc156677365)

[Chương 4: Triển khai và đánh giá kết quả 9](#_Toc156677366)

[Chương 5: Kết luận và hướng phát triển 9](#_Toc156677367)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 10](#_Toc156677368)

[2.1. Mô hình CI/CD 10](#_Toc156677369)

[2.1.1. Continuous Integration – CI 10](#_Toc156677370)

[2.1.1.1. Tổng quan 10](#_Toc156677371)

[2.1.1.2. Github 12](#_Toc156677372)

[2.1.1.3. Jenkins 12](#_Toc156677373)

[2.1.2. Continuous Delivery - CD/CDE 13](#_Toc156677374)

[2.1.3. Continuous Deployment - CD 14](#_Toc156677375)

[2.2. Mô hình DevOps 15](#_Toc156677376)

[2.2.1 Nguồn gốc 15](#_Toc156677377)

[2.2.2. Đặc điểm 16](#_Toc156677378)

[2.2.2. Triển khai 17](#_Toc156677379)

[2.3. Mô hình DevSecOps 19](#_Toc156677380)

[2.3.1. Nguồn gốc 19](#_Toc156677381)

[2.3.2. Đặc điểm 20](#_Toc156677382)

[2.3.3. Triển khai 22](#_Toc156677383)

[2.4. Mô hình Microservices 24](#_Toc156677384)

[2.4.1. Tổng quan 24](#_Toc156677385)

[2.4.2. Container hóa 25](#_Toc156677386)

[2.4.3. Snyk 27](#_Toc156677387)

[2.5. Trí tuệ nhân tạo 28](#_Toc156677388)

[2.5.1. Học máy 29](#_Toc156677389)

[2.5.2. Học sâu 31](#_Toc156677390)

[CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ VÀ THIẾT KẾ MÔ HÌNH 33](#_Toc156677391)

[3.1. Mô hình tổng quan 33](#_Toc156677392)

[3.2. Mô hình chi tiết 34](#_Toc156677393)

[3.3. Phân tích và thiết kế Jenkins Pipeline 36](#_Toc156677394)

[CHƯƠNG 4. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 37](#_Toc156677395)

[4.1. Triển khai mô hình 37](#_Toc156677396)

[4.2. Cấu hình các ứng dụng 39](#_Toc156677397)

[4.3. Lỗ hổng XSS 42](#_Toc156677398)

[4.4. Kết quả thử nghiệm 42](#_Toc156677399)

[4.4.1. Triển khai ứng dụng 43](#_Toc156677400)

[4.4.2. Đánh giá mô hình đã triển khai 43](#_Toc156677401)

[4.4.2.1 Ưu điểm của mô hình đã triển khai 43](#_Toc156677402)

[4.4.2.2. Nhược điểm của mô hình đã triển khai 44](#_Toc156677403)

[CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 44](#_Toc156677404)

[5.1. Kết luận 44](#_Toc156677405)

[5.2. Hướng phát triển 44](#_Toc156677406)

[TÀI LIỆU THAM KHẢO 46](#_Toc156677407)

[PHỤ LỤC 47](#_Toc156677408)

[Mã nguồn chứa lỗ hổng bảo mật 47](#_Toc156677409)

## CHƯƠNG 1. GIỚI THIỆU

### Lý do chọn đề tài

Trước hết, an toàn phần mềm đã trở thành một quan tâm hàng đầu với sự phát triển nhanh chóng của các ứng dụng và hệ thống phần mềm ngày nay. Sự xuất hiện của các lỗ hổng an toàn có thể dẫn đến hậu quả nghiêm trọng như mất thông tin cá nhân, thất thoát dữ liệu quan trọng, hoặc thậm chí là mất kiểm soát toàn bộ hệ thống. Do đó, nghiên cứu về cách phát hiện mẫu lập trình kém an toàn là bước quan trọng để tạo ra các hệ thống phần mềm đáng tin cậy.

Thứ hai, bảo mật thông tin ngày càng trở nên phức tạp và đòi hỏi sự chú ý đặc biệt đối với các lập trình viên và nhà phát triển phần mềm. Đối mặt với những mối đe dọa ngày càng tinh vi, việc phát triển mô hình máy học có khả năng phát hiện mẫu lập trình có thể tạo ra lỗ hổng bảo mật là một cách hiệu quả để nâng cao mức độ an toàn cho các ứng dụng và hệ thống.

Thứ ba, mô hình máy học có thể đóng vai trò quan trọng trong việc tăng cường hiệu suất trong quy trình phát triển phần mềm. Việc tự động hóa phát hiện lỗi lập trình có thể giảm thiểu thời gian và nguồn lực cần thiết cho các bước kiểm thử thủ công, giúp tăng cường năng suất và giảm rủi ro của việc triển khai phần mềm có vấn đề an toàn.

### Mục tiêu nghiên cứu

* Nghiên cứu quy trình phát triển phần mềm có tính tích hợp liên tục (Continuous Integration - CI), triển khai liên tục (Continuous Deployment

- CD) dựa trên mô hình DevOps và các nguy cơ gây mất an toàn thông tin.

* Xây dựng quy trình đảm bảo an toàn thông tin và tích hợp các công cụ kiểm thử tự động để kiểm tra tính bảo mật của ứng dụng.
* Triển khai và đánh giá quy trình đảm bảo an toàn thông tin cho một dự án phát triển ứng dụng.

### Đối tượng nghiên cứu

* Mô hình DevOps và các nguy cơ gây mất an toàn thông tin.
* Các công cụ hỗ trợ cho quá trình tích hợp liên tục và triển khai liên tục.
* Các công cụ học máy tích hợp đảm bảo an toàn thông tin trong quá trình phát triển và vận hành ứng dụng.

### Phạm vi nghiên cứu

Nghiên cứu và tìm hiểu tham khảo về quy trình phát triển và vận hành phần mềm DevOps. Từ đó, triển khai mô hình nâng cao tính bảo mật cho DevOps bằng việc ứng dụng giải pháp mã nguồn mở và các công cụ tích hợp học máy để kiểm thử bảo mật nhằm phát hiện các mẫu lập trình kém trong ứng dụng.

### Những đóng góp của đồ án

Các quy trình phát triển phần mềm đã triển khai theo hướng DevOps trong thực tế thì vẫn có khả năng mở rộng và tích hợp mô hình DevSecOps để giảm thiểu rủi ro bảo mật. Mô hình có thể ứng dụng cho các doanh nghiệp vừa và nhỏ trong hoạt động phát triển và vận hành phần mềm. Ngày càng xuất hiện nhiều rủi ro gây mất an toàn thông tin và vấn đề bảo mật ứng dụng đang cấp thiết hơn bao giờ hết. Khi sự cố bảo mật xảy ra, tính tin cậy của sản phẩm phần mềm trực tiếp bị ảnh hưởng và tốn rất nhiều nguồn lực để khắc phục sự cố.

Do đó, mô hình đảm bảo an toàn thông tin trong quy trình phát triển và vận hành phần mềm (DevSecOps) được nhóm tác giả nghiên cứu nhằm giải quyết vấn đề thực tế này.

### Cấu trúc của đồ án

Bao gồm 5 chương như sau:

### Chương 1: Giới Thiệu

Trình bày lý do chọn đề tài, mục tiêu nghiên cứu, đối tượng nghiên cứu, phạm vi nghiên cứu, những đóng góp của đồ án.

### Chương 2: Cơ sở lý thuyết

Trình bày về cơ sở lý thuyết các nội dung liên quan đến khái niệm, công cụ, mô hình trong quá trình thực hiện. Các kiến thức được trình bày trong chương này là cơ sở để thiết kế và triển khai mô hình hệ thống DevSecOps. Ngoài ra còn mô tả về học máy được tích hợp vào công cụ kiểm thử bảo mật để phát hiện mẫu lập trình kém an toàn trong quy trình phát triển và vận hành ứng dụng.

### Chương 3: Phân tích và thiết kế mô hình

Từ cơ sở lý thuyết trên, trong chương này sẽ phân tích và thiết kế mô hình. Bao gồm các mô hình hệ thống, các bước thực hiện CI/CD trong Jenkins Pipeline và ứng dụng bảo mật.

### Chương 4: Triển khai và đánh giá kết quả

Chi tiết về các thành phần trong hệ thống, các chức năng trong mô hình và liệt kê bước cần thiết để triển khai mô hình DevSecOps.

Đánh giá ưu nhược điểm của mô hình trong kịch bản thử nghiệm.

### Chương 5: Kết luận và hướng phát triển

Tổng kết quá trình thực hiện đồ án và đề xuất các hướng phát triển đề tài trong tương lai.

## CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

### Mô hình CI/CD

### Continuous Integration – CI

### Tổng quan

Tích hợp liên tục (Continuous Integration - CI) là một phương pháp phát triển phần mềm DevOps trong đó các nhà phát triển thường xuyên hợp nhất các thay đổi mã của họ vào kho lưu trữ trung tâm, sau đó chạy các bản dựng và thử nghiệm tự động. Tích hợp liên tục thường đề cập đến giai đoạn xây dựng hoặc tích hợp của quy trình phát hành phần mềm và bao gồm cả thành phần tự động hóa (ví dụ: CI hoặc build service) và thành phần văn hóa (ví dụ: học cách tích hợp thường xuyên). Mục tiêu chính của tích hợp liên tục là tìm và giải quyết lỗi nhanh hơn, cải thiện chất lượng phần mềm và giảm thời gian xác thực và phát hành bản cập nhật phần mềm mới.

Trước đây, các nhà phát triển trong một nhóm có thể làm việc độc lập trong một thời gian dài và chỉ hợp nhất các thay đổi của họ vào nhánh chính sau khi công việc của họ hoàn thành. Điều này khiến cho việc hợp nhất các thay đổi mã trở nên khó khăn và tốn thời gian, đồng thời còn dẫn đến lỗi tích tụ lâu ngày mà không được sửa chữa. Những yếu tố này khiến việc cung cấp thông tin cập nhật cho khách hàng một cách nhanh chóng trở nên khó khăn hơn. Với sự tích hợp liên tục, các nhà phát triển thường xuyên cam kết sử dụng kho lưu trữ dùng chung bằng hệ thống kiểm soát phiên bản như Git. Trước mỗi lần cam kết, nhà phát triển có thể chọn chạy thử nghiệm đơn vị cục bộ trên mã của họ dưới dạng lớp xác minh bổ sung trước khi tích hợp. Dịch vụ tích hợp liên tục tự động xây dựng và chạy thử nghiệm đơn vị đối với các thay đổi mã mới để phát hiện ngay mọi lỗi.

A diagram of a process

Description automatically generated

Hình 2.1.1: Mô hình CI

### Github

GitHub là một dịch vụ lưu trữ mã nguồn và quản lý phiên bản phân tán (distributed version control system) phổ biến. Nó cho phép các nhà phát triển phát triển phần mềm cộng tác, quản lý và theo dõi sự thay đổi trong mã nguồn của dự án.

GitHub cung cấp một nền tảng trực quan và dễ sử dụng để lưu trữ các dự án phần mềm, quản lý phiên bản và phối hợp công việc giữa các thành viên trong một nhóm phát triển. Được phát triển dựa trên hệ thống quản lý phiên bản Git, GitHub cung cấp nhiều tính năng hữu ích như hệ thống theo dõi lỗi (issue tracking), wiki, công cụ hỗ trợ liên quan đến việc kiểm tra mã (code review), tích hợp liên tục (continuous integration) và nhiều hơn nữa.

### Jenkins

Jenkins có mã nguồn mở Jenkins là một máy chủ tích hợp liên tục (CI) mã nguồn mở và được phát triển bằng Java.. Nó quản lý và kiểm soát một số giai đoạn của quy trình phân phối phần mềm, bao gồm xây dựng, tài liệu, kiểm tra tự động, đóng gói và phân tích mã tĩnh. Jenkins là một công cụ DevOps rất phổ biến được hàng nghìn nhóm phát triển sử dụng.

A screenshot of a computer

Description automatically generated

Hình 2.1.2: Quy trình làm việc Jenkins

Quá trình tự động hóa của Jenkins thường được kích hoạt bằng cách thay đổi mã trong các kho lưu trữ như GitHub, Bitbucket và GitLab, đồng thời tích hợp với các công cụ xây dựng như Maven và Gradle. Jenkins hỗ trợ việc sử dụng các công nghệ vùng chứa như Docker và Kubernetes để thử nghiệm và đóng gói các bản phát hành phần mềm, nhưng đây không phải là giải pháp gốc Kubernetes hay giải pháp CI gốc vùng chứa.

### Continuous Delivery - CD/CDE

Phân phối liên tục (Continuous Delivery - CD/CDE) là bước tiếp theo sau khi hoàn tất thành công quá trình tích hợp liên tục. Quá trình này sẽ triển khai ứng dụng tự động trong một hoặc nhiều môi trường staging.

CI thực hiện đóng gói ứng dụng và lưu trữ chúng trong package management (ví dụ như Nexus, Azure Artifacts,…). Gói ứng dụng này sẽ được triển khai trong quá trình phân phối liên tục. Điều này đảm bảo rằng ứng dụng sẽ được xác thực trong các môi trường khác nhau đều có cùng chất lượng như nhau.

Mối liên kết giữa tích hợp liên tục và phân phối liên tục trong môi trường tích hợp rất phổ biến. Trong khi CI sẽ thực hiện kiểm thử đơn vị (unit test) trong quá trình phát triển phần mềm thì CD sẽ kiểm tra toàn bộ ứng dụng bao gồm giao diện người dùng, chức năng và tất cả các phụ thuộc của nó.

Điểm quan trọng trong quy trình phân phối liên tục là việc triển khai xuống môi trường sản xuất được kích hoạt theo cách thủ công bởi những người được cấp quyền phê duyệt.

A diagram of a process

Description automatically generatedHình 2.1.3: Mô hình phân phối liên tục

### Continuous Deployment - CD

Triển khai liên tục (Continuous Deployment - CD) là một chiến lược phát triển và phát hành phần mềm trong đó mọi thay đổi mã đều được tự động thực hiện thông qua một quy trình kiểm tra và kiểm tra trước khi được đưa vào sản xuất. Việc triển khai liên tục sẽ loại bỏ các biện pháp bảo vệ của con người đối với mã chưa được chứng minh trong phần mềm trực tiếp. Nó chỉ nên được kết hợp khi nhóm phát triển và nhà phát triển tuân thủ nghiêm ngặt các phương pháp phát triển sẵn sàng sản xuất và thử nghiệm kỹ lưỡng cũng như khi họ áp dụng các số liệu và giám sát thời gian thực phức tạp trong sản xuất để phát hiện bất kỳ vấn đề nào với các bản phát hành phần mềm mới. Mục đích của việc triển khai liên tục là để tăng tốc độ phát triển và giảm thời gian để các thay đổi đến tay người dùng cuối. Để áp dụng triển khai liên tục, hệ thống tích hợp liên tục (CI) thường được sử dụng để tự động xây dựng và kiểm tra các thay đổi đối với cơ sở mã. Khi các thay đổi đã vượt qua tất cả các thử nghiệm, chúng sẽ tự động được triển khai vào môi trường sản xuất. Điều này có thể đạt được bằng cách sử dụng quy trình triển khai, tự động hóa quá trình xây dựng, thử nghiệm và triển khai các bản cập nhật. Quy trình phân phối liên tục bao gồm nhiều bước khác nhau, chẳng hạn như xây dựng mã, chạy thử nghiệm tự động, triển khai lên môi trường chạy thử, thử nghiệm trong các môi trường đó và triển khai vào sản xuất. Việc triển khai liên tục chủ yếu dựa vào tự động hóa để đảm bảo rằng các thay đổi được triển khai một cách nhất quán và đáng tin cậy. Các nhóm phát triển có thể nhanh chóng phát hiện và khắc phục sự cố, giảm nguy cơ phát sinh lỗi và cuối cùng mang lại giá trị cho người dùng cuối với tốc độ nhanh hơn bằng cách liên tục A diagram of a process

Description automatically generatedtriển khai các bản cập nhật mã.

Hình 2.1.4: Mô hình triển khai liên tục

Tích hợp, phân phối và triển khai liên tục được gọi chung là phát triển phần mềm liên tục và chúng được liên kết với các phương pháp Agile và DevOps. Phân phối và triển khai liên tục bắt nguồn từ sự tích hợp liên tục, một phương pháp phát triển, xây dựng và thử nghiệm mã mới một cách nhanh chóng bằng tự động hóa để chỉ những mã được biết là tốt mới trở thành một phần của sản phẩm phần mềm. Triển khai liên tục không giống như phân phối liên tục, mặc dù hai thuật ngữ này thường bị nhầm lẫn.

### 2.2. Mô hình DevOps

### 2.2.1 Nguồn gốc

Có nhiều sự tích li kỳ về sự ra đời của DevOps nhưng thực tế thì nó không được tạo mới hoàn toàn mà thay vào đó, những hạt giống của DevOps đã được gieo trồng từ lâu và đã được nuôi dưỡng bởi các chuyên gia CNTT có tư duy tiến bộ trong một số lĩnh vực. Hai tiền thân chính của DevOps là:

Quản lý hệ thống doanh nghiệp (ESM): Nhiều người tham gia vào định nghĩa ban đầu về DevOps là quản trị viên hệ thống (system administrators). Các chuyên gia này đã đưa các phương pháp hay nhất của ESM đến DevOps, bao gồm quản lý cấu hình, giám sát hệ thống, cung cấp tự động và phương pháp chuỗi công cụ.

Agile development : Một người quan sát đã note “DevOps có thể được hiểu là sự phát triển vượt bậc của Agile - phát triển phần mềm agile quy định sự hợp tác chặt chẽ của khách hàng, quản lý sản phẩm, nhà phát triển và (đôi khi) QA để lấp đầy khoảng trống và nhanh chóng lặp lại để hướng tới một sản phẩm tốt hơn ... Từ quan điểm này, DevOps chỉ đơn giản là mở rộng các nguyên tắc Agile vượt ra ngoài ranh giới của mã cho toàn bộ dịch vụ được cung cấp”.

Trong những năm qua, nhiều người đã làm việc để áp dụng các phương pháp Agile vào các hoạt động vận hành nhằm mục đích xóa bỏ bức tường giữa hai bộ phận phát triển và vận hành. Bằng cách chia sẻ kỹ năng, ý tưởng, vấn đề, quy trình, công cụ, mục tiêu và kinh nghiệm, phong trào đã hình thành nên DevOps. Với cách tiếp cận DevOps, vai trò của nhà phát triển bao gồm lập trình viên, người kiểm thử, QA và chuyên gia vận hành. Mỗi cá nhân đều có các hiểu biết cơ bản về lĩnh vực của các thành viên khác. Tất cả đều phát triển phần mềm và đưa phần mềm đó đến với người dùng một cách nhanh chóng và an toàn.

A diagram of software development

Description automatically generated

Hình 2.2.1: Quy trình DevOps

### 2.2.2. Đặc điểm

Trước đây thì bộ phận Dev và Operations làm việc độc lập với từng công việc cụ thể khác nhau. Bộ phận Dev có mục đích chính là xây dựng function đúng với yêu cầu đưa đến cho khách hàng nhanh chóng. Đối với bộ phận Operations mục đích chính là tạo ra một hệ thống ổn định.

Devops chính là sự kết hợp hài hòa giữa Dev và Operations với những mục tiêu chung như:

* Fast time - to market (TTM): Code nhanh, triển khai nhanh.
* Few production failures: Đảm bảo tính ổn định từ đầu.
* Immediate recovery from failures: Nếu có xảy ra lỗi thì 1 team Devops sẽ có thể nhanh chóng tìm ra nguyên nhân và khắc phục lỗi ngay lập tức.

Những điều này đòi hỏi cả Dev và operations phải làm việc cùng nhau, thống nhất trong mục tiêu, dùng tools để kiểm tra tốc độ, bảo đảm sự ổn định của sản phẩm và mang lại sản phẩm hoàn hảo cho khách hàng.

So sánh với mô hình làm việc truyền thống Traditional Silos, ta có thể thấy:

Với quy trình truyền thống, bộ phận dev sẽ có trách nhiệm viết code để đưa sang cho QA test. QA khi phát hiện ra lỗi sẽ đưa cho bên dev fix, rồi chuyển lại cho QA để test. Quá trình này sẽ lặp đi lặp lại cho đến khi sản phẩm không còn lỗi, khi đó sẽ được chuyển giao cho bộ phận operation để phát triển code. Khi này, QA sẽ test lại lần nữa. Lúc này nếu có phát hiện ra code thì nguyên nhân có thể đến từ code của Dev viết sai hoặc do operation đã deploy sai. Với mô hình làm việc truyền thống, cách làm của Dev và Operation đều mang hơi hướng cá nhân, không giúp đỡ hỗ trợ lẫn nhau, điều này sẽ khiến cho quá trình tìm kiếm nguyên nhân gây ra lỗi sẽ lâu hơn.

Với mô hình Devops, Dev QA và operations là một team thống nhất cùng làm một sản phẩm, ở đó họ sẽ cùng nhau linh hoạt sử dụng các tools để tạo ra một hệ thống automation từ khi build code, test cho đến khi deploy. Khi phát hiện ra lỗi họ sẽ nhanh chóng tìm ra bug và fix nó lại, đồng thời deploy code mới lên.

### 2.2.2. Triển khai

Các công ty sử dụng một bộ công cụ và phương pháp tiếp cận đa dạng để xây dựng quy trình DevOps độc đáo và hiệu quả, được tùy chỉnh cho nhu cầu của tổ chức của họ. Các bước phổ biến bao gồm thiết lập công cụ CI/CD, tìm nguồn cung ứng môi trường kiểm soát, thiết lập máy chủ bản dựng, thiết lập các công cụ tự động hóa bản dựng để thử nghiệm và triển khai vào sản xuất.

Bước 1: Thiết lập công cụ CI/CD:

* + - * + Đối với các công ty mới bắt đầu xây dựng quy trình DevOps, việc đầu tiên trong kinh doanh là chọn một công cụ CI/CD. Mỗi công ty có những nhu cầu và yêu cầu khác nhau, vì vậy không phải mọi công cụ đều phù hợp với mọi tình huống. Mặc dù có rất nhiều công cụ CI/CD nhưng Jenkins là một trong những công cụ được sử dụng phổ biến nhất. Jenkins có hàng trăm plugin và công cụ do cộng đồng đóng góp nên có thể dễ dàng tùy chỉnh để hoạt động tốt cho nhiều ứng dụng khác nhau.

Bước 2: Tìm nguồn môi trường kiểm soát:

* + - * + Các công ty làm việc với các nhóm phát triển lớn cần một nơi dành riêng để lưu trữ và chia sẻ mã luôn thay đổi, tránh xung đột khi hợp nhất và dễ dàng tạo các phiên bản khác nhau của ứng dụng hoặc phần mềm. Các công cụ quản lý kiểm soát nguồn như Git cho phép cộng tác hiệu quả với các thành viên trong nhóm từ mọi nơi trên thế giới, lưu trữ mã của mỗi nhà phát triển trong một kho lưu trữ chung riêng biệt. BitBucket và GitLab là hai công cụ quản lý kiểm soát nguồn phổ biến khác.

Bước 3. Thiết lập Build Server:

* + - * + Thiết lập máy chủ xây dựng, còn được gọi là máy chủ tích hợp liên tục (CI) là bước quan trọng tiếp theo trước khi dự án của bạn có thể đi xuống phần còn lại của quy trình. Máy chủ xây dựng là môi trường tập trung, ổn định và đáng tin cậy để xây dựng các dự án phát triển phân tán. Máy chủ xây dựng truy xuất mã tích hợp từ kho lưu trữ mã nguồn, hoạt động như một điểm tích hợp cho tất cả các nhà phát triển và cung cấp một môi trường không bị nhiễm độc để đảm bảo rằng mã thực sự hoạt động. Giống như các công cụ khác được đề cập ở trên, các công ty có thể chọn từ nhiều máy chủ xây dựng, mỗi máy chủ có các tính năng khác nhau. Jenkins là một trong những giải pháp phổ biến nhất để tạo bản dựng. Các giải pháp khác bao gồm TeamCity, Travis-CI và go.cd.

Bước 4. Thiết lập hoặc xây dựng các công cụ tự động hóa để kiểm tra:

* + - * + Sau khi mã được định cấu hình trên máy chủ xây dựng, đã đến lúc kiểm tra nó! Trong giai đoạn thử nghiệm, các nhà phát triển chạy thử nghiệm tự động để đảm bảo rằng chỉ có mã không có lỗi mới tiếp tục được đưa đến giai đoạn triển khai. Một số thử nghiệm tự động được thực hiện ở giai đoạn thử nghiệm bao gồm thử nghiệm đơn vị, chức năng, tích hợp và hồi quy. Hầu hết các thử nghiệm đều được chạy qua CI và chạy lần lượt từng thử nghiệm. Để bắt đầu chạy thử nghiệm, TestComplete là một lựa chọn tốt. Nó có plugin Jenkins cho phép bạn chạy thử nghiệm trong dự án Đường ống Jenkins với các tính năng bổ sung như lưu trữ lịch sử thử nghiệm và cho phép bạn xem kết quả trực tiếp từ Jenkins.

Bước 5. Triển khai vào sản xuất:

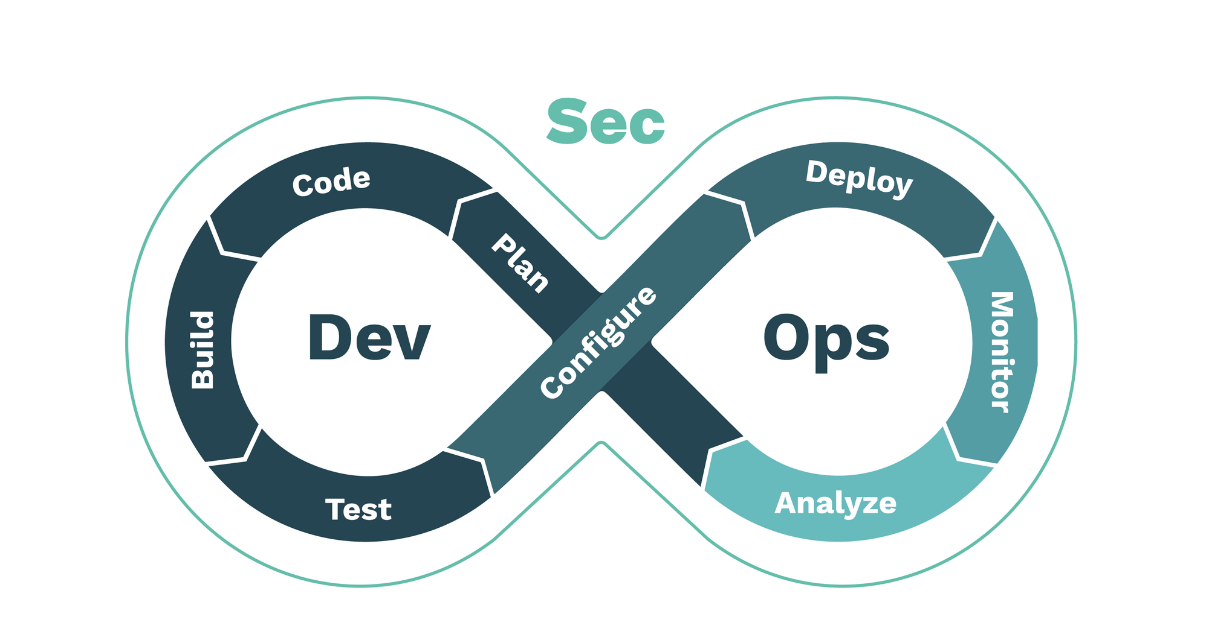
* + - * + Giai đoạn cuối cùng trong quy trình của bạn là giai đoạn triển khai, nơi phần mềm của bạn sẵn sàng được đưa vào sản xuất. Cách dễ nhất để triển khai mã là định cấu hình máy chủ xây dựng của bạn, chẳng hạn như Jenkins, để chạy tập lệnh nhằm triển khai ứng dụng của bạn. Điều này có thể được thiết lập để chạy tự động hoặc bạn có thể thực hiện thủ công. Chỉ nên sử dụng triển khai tự động nếu bạn hoàn toàn tin tưởng rằng mã xấu sẽ không được đưa vào sản xuất. Bạn có thể liên kết phần này với các bản dựng thử nghiệm của mình để tập lệnh chỉ chạy nếu tất cả các thử nghiệm đã vượt qua.

### 2.3. Mô hình DevSecOps

### 2.3.1. Nguồn gốc

Quay trở lại thời xa xưa, DevSecOps bắt đầu vào năm 1976 và điều đó bắt đầu bằng một bài báo viết mô tả 11 thuộc tính của chất lượng. Rất nhiều trong số những khái niệm này đã được suy nghĩ rất tiến bộ. Bây giờ, bạn không thấy nhiều biện pháp bảo mật vào những năm 70 và cũng không có nhiều tin tặc hoặc trường hợp lạm dụng phần mềm không giống như ngày nay khi bảo mật phải được tích hợp vào phần mềm.

Thế hệ DevSecOps ban đầu tập trung nhiều hơn vào các công cụ và là một hoạt động thực hành được xếp lớp giữa phát triển và vận hành. Mục tiêu là tích hợp các biện pháp bảo mật vào vòng đời phát triển hệ thống nhưng không làm chậm quá trình phát triển. Trong thời gian này, các nhà phát triển mang gánh nặng và kỳ vọng phải đọc và áp dụng nhiều chính sách đồng thời giảm thiểu các vấn đề từ hàng chục công cụ bảo mật khác nhau. Trong khi đó, các chính sách vẫn được giữ bên ngoài các công cụ và quy trình phát triển điển hình. Vấn đề với cách tiếp cận này là gì? Nó hạn chế đáng kể khả năng mở rộng và không thực tế về lâu dài. AppSec là tĩnh và không thể theo kịp nhu cầu liên tục về chất lượng phát hành hoặc thay đổi nhanh chóng.



Hình 2.2.1.: Mô hình DevSecOps

### 2.3.2. Đặc điểm

DevSecOps là một phương pháp tích hợp bảo mật vào vòng đời phát triển phần mềm (SDLC) trong môi trường Agile bao gồm các đặc điểm sau:

Hợp tác: Các nhóm phát triển, bảo mật và vận hành làm việc cùng nhau để xây dựng phần mềm bảo mật ngay từ đầu.

Tự động hóa: Kiểm tra và xác thực bảo mật được tự động hóa và tích hợp vào SDLC.

Tích hợp và phân phối liên tục: Mã được tích hợp và kiểm tra liên tục, cho phép phát hiện sớm và khắc phục các vấn đề bảo mật.

Văn hóa bảo mật: Bảo mật được coi là trách nhiệm chung giữa tất cả các nhóm, thay vì được giữ trong nhóm bảo mật.

Cách tiếp cận dịch chuyển sang trái: Kiểm tra và xác thực bảo mật được thực hiện càng sớm càng tốt trong SDLC để giảm nguy cơ về các lỗ hổng bảo mật.

Cách tiếp cận dựa trên rủi ro: Trọng tâm là xác định và giảm thiểu rủi ro cao nhất cho tổ chức, thay vì tuân thủ một phương pháp bảo mật chung cho tất cả.

Phương pháp linh hoạt: Áp dụng các phương pháp linh hoạt, chẳng hạn như Scrum hoặc Kanban, để cho phép phát hành thường xuyên và phản ứng nhanh với các mối đe dọa bảo mật.

Tuân thủ: Đảm bảo tuân thủ các tiêu chuẩn và quy định bảo mật, chẳng hạn như ISO 27001 hoặc PCI DSS.

Học hỏi liên tục: Liên tục theo dõi và học hỏi từ các sự cố bảo mật, cải thiện các biện pháp bảo mật theo thời gian.

Sự phát triển của DevSecOps đã được thể hiện rõ ràng trong các lĩnh vực tuân thủ và có ý thức bảo mật. Ví dụ, nó có lượng người theo dõi ngày càng tăng trong Bộ Quốc phòng Hoa Kỳ quan tâm đến an ninh. Các dự án như Platform One đang nêu ví dụ về cách thực hành DevSecOps có thể bảo vệ công nghệ đám mây và nguồn mở trong các nhiệm vụ chính phủ có ý thức bảo mật nhất. DevSecOps có tỷ lệ thâm nhập từ 20% đến 50% trong ngành, theo Gartner's Hype Cycle for Agile and DevOps, 2020. Đại dịch đã đóng vai trò như chất xúc tác cho DevSecOps khi các tổ chức chuyển hoạt động phát triển ứng dụng sang đám mây.

Ngay cả khi bạn coi DevSecOps như một bước khác trong hành trình DevOps của mình, bạn vẫn có thể mong đợi những thay đổi đối với chuỗi công cụ, vai trò trong DevOps và nhóm bảo mật cũng như cách các nhóm của bạn tương tác. Hơn 60% số người trả lời Khảo sát DevSecOps toàn cầu năm 2021 của GitLab báo cáo rằng họ có các vai trò và trách nhiệm mới nhờ DevOps, vì vậy hãy chuẩn bị trước cho nhân viên của bạn và hạn chế tối đa sự ngạc nhiên. DevSecOps mang đến một nền văn hóa giống như cách DevOps làm. Thúc đẩy văn hóa DevSecOps là đặt vấn đề bảo mật lên hàng đầu và coi đó là công việc của mọi người. Các tổ chức DevSecOps cần phải vượt xa chương trình đào tạo bảo mật trực tuyến bắt buộc trên toàn công ty bằng các cuộc đối thoại có sẵn và đưa tính bảo mật vào các quy trình kinh doanh và phát triển.

### 2.3.3. Triển khai

**DevSecOps ở giai đoạn xây dựng:**

Xây dựng nền tảng hình ảnh đáng tin cậy: Để bắt đầu, chúng ta cần tin tưởng vào sổ đăng ký hình ảnh đang được sử dụng. Chúng tôi đã gặp một số hình ảnh cơ sở dữ liệu rất kỳ lạ từ nhiều nguồn khác nhau và có thể rất mơ hồ. Những điều này có thể dẫn đến khả năng bị tin tặc chiếm đoạt - những lỗi tốn kém và tốn thời gian để khắc phục sau đó. Tóm lại: hãy hết sức lưu ý về những gì bạn sử dụng làm hình ảnh cơ sở. Và hình ảnh cơ bản nên tối thiểu nhất có thể. Càng có nhiều yếu tố trong hình ảnh thì càng có nhiều lỗ hổng tiềm ẩn cần được khai thác.

Hình ảnh cần được cập nhật thường xuyên: Việc này chủ yếu phải được tự động hóa, đôi khi phụ thuộc vào công việc thủ công. Ngoài ra, hãy nhớ kiểm tra các phần phụ thuộc và hành động theo chúng. Đó là điều có thể bị bỏ qua, vì với những thay đổi về hình ảnh, có thể cần phải điều chỉnh điều gì đó ở cấp độ ứng dụng. Nhưng đây chính là lúc môi trường phi sản xuất trở nên hữu ích, để kiểm tra và khắc phục sự cố, cho phép bạn dễ dàng đưa các bản cập nhật vào môi trường sản xuất của mình.

Giữ hình ảnh sạch sẽ - theo cách tự động: Dành thời gian để xóa mọi phần mềm không cần thiết và có khả năng bị khai thác khỏi hình ảnh của bạn, bao gồm trình quản lý gói, công cụ mạng, shell, trình biên dịch và trình gỡ lỗi. Đây là những chỗ đứng lý tưởng để tin tặc tìm ra những lỗ hổng tiềm ẩn. Chúng tôi thực sự khuyên bạn nên sử dụng công cụ SCA để tự động hóa tác vụ này. Các công cụ SCA thực hiện quét tự động cơ sở mã của ứng dụng và các thành phần có liên quan như vùng chứa và sổ đăng ký, xác định tất cả các thành phần nguồn mở, dữ liệu tuân thủ giấy phép của chúng và mọi lỗ hổng bảo mật. Cùng với việc cung cấp khả năng hiển thị về việc sử dụng nguồn mở, một số công cụ SCA còn giúp khắc phục các lỗ hổng nguồn mở thông qua việc ưu tiên hóa và tự động khắc phục. Và sử dụng chức năng quét tự động để xem qua sổ đăng ký hình ảnh của bạn và phát hiện các lỗ hổng tiềm ẩn và đã biết. Các công cụ SAST được sử dụng trong giai đoạn xây dựng để quét chủ động với quyền truy cập bên trong nhằm thực hiện phân tích tĩnh và kiểm tra mã nguồn ứng dụng hoặc mã byte để phát hiện các phần tử không an toàn. Họ có thể xác định chính xác các vấn đề (nhưng dễ đưa ra kết quả sai). Chúng cũng dành riêng cho từng ngôn ngữ và không thể xác định các lỗ hổng trong thời gian chạy vì quá trình quét SAST được thực hiện mà không cần chạy ứng dụng.

Hạn chế khả năng hiển thị: Thông tin nhạy cảm và bí mật cũng như cách truy cập và sử dụng thông tin đó là yếu tố quan trọng cần cân nhắc khi thực hiện các hoạt động chặt chẽ. Không lưu trữ các tệp hoặc mã không được mã hóa, không sử dụng chúng làm biến môi trường và đảm bảo bạn có kiến ​​thức đầy đủ về vị trí và cách thức mọi thông tin bí mật đang được sử dụng cũng như sử dụng ý thức chung để hạn chế khả năng hiển thị.

**DevSecOps ở giai đoạn triển khai:**

Luôn thông báo về việc triển khai: Nếu có sự cố, vấn đề đó cần được theo dõi bởi cá nhân đã gây ra sự cố đó. Việc này có thể được thực hiện bằng Slack, email hoặc sms, nhưng cần phải có một đường dây liên lạc rõ ràng giữa những người chịu trách nhiệm về việc đó. Và nếu bạn cảm thấy việc triển khai có rủi ro, hãy thận trọng khi cần thiết và chặn việc triển khai. Giải thích lý do và thông báo cho nhà phát triển cách khắc phục.

Tự động quét lỗ hổng: Trong giai đoạn triển khai, tính năng quét DAST được sử dụng để thăm dò một ứng dụng trực tiếp nhằm mô phỏng hành động của tin tặc trong đời thực một cách an toàn. Các sản phẩm hiện đại có thể tìm thấy các lỗ hổng trong thời gian chạy và thường xác nhận chúng để loại bỏ các kết quả dương tính giả, trong khi chúng không có quyền truy cập vào nguồn ứng dụng, chúng độc lập với ngôn ngữ và có thể kiểm tra toàn bộ môi trường ứng dụng.

**DevSecOps ở giai đoạn vận hành:**

Khi quá trình xây dựng và triển khai hoàn tất và ứng dụng đang chạy, điều quan trọng là phải theo dõi chặt chẽ các ứng dụng đang vận hành. Đó là nơi tất cả những quyết định được đưa ra trước đó trở thành hiện thực.

Sử dụng chức năng quét động: Theo kinh nghiệm của chúng tôi, đây là giai đoạn mà tự động hóa có thể được áp dụng cho nhiều bước để giúp mọi thứ có thể quản lý được. Sử dụng và triển khai chức năng quét động để phát hiện bất kỳ lỗ hổng nào và các vùng chứa đang chạy nhằm xây dựng quy trình cũng như hành động nhanh chóng khi hoạt động độc hại bị gắn cờ.

Sử dụng danh sách trắng: Điều này cho phép bạn xây dựng danh sách các quy trình được phê duyệt đồng thời chặn các hành động không mong muốn như leo thang đặc quyền, khai thác tiền điện tử, các quy trình độc hại và có thể khai thác. Đây là điều có thể yêu cầu một số công việc bổ sung hoặc phần mềm của bên thứ ba trừ khi bạn có kiến ​​thức tốt về các kỹ thuật và công cụ chống hack. Và hãy đảm bảo rằng bạn đang xây dựng các quy trình và tự động hóa cách xử lý khi phát hiện thấy các mối đe dọa - đây là điều bắt buộc để hạn chế thiệt hại và tác hại tiềm ẩn đối với công ty mà bạn đang xử lý những mối đe dọa này.

### Mô hình Microservices

### Tổng quan

Microservices là một kiểu kiến trúc trong đó các ứng dụng phần mềm lớn, phức tạp bao gồm một hoặc nhiều dịch vụ (hay còn gọi là microservice). Các microservice có thể được triển khai độc lập với nhau. Mỗi dịch vụ này chỉ tập trung vào việc hoàn thành một nhiệm vụ và thực hiện tốt nhiệm vụ đó.

Kiến trúc microservices dẫn đến số lượng các ứng dụng nhỏ được triển khai thường xuyên hơn và nhiều hơn. DevOps cho phép triển khai thường xuyên hơn và mở rộng quy mô để xử lý số lượng ngày càng tăng các bản phát hành microservice mới. Nếu không có DevOps, mỗi nhóm microservice phải tạo cơ sở hạ tầng và dịch vụ DevOps của riêng chúng, dẫn đến chi phí phát triển cao hơn. Điều đó cũng có nghĩa là các mức chất lượng, bảo mật và tính khả dụng của các dịch vụ giữa các nhóm microservice không nhất quán.

Kiến trúc microservices mang lại những lợi ích sau:

* Cho phép phân phối và triển khai liên tục các ứng dụng lớn, phức tạp.
* Các dịch vụ nhỏ và dễ bảo trì.
* Có thể triển khai độc lập các dịch vụ.
* Các dịch vụ có thể mở rộng một cách độc lập.
* Kiến trúc microservices cho phép các nhóm tự chủ.
* Nó cho phép dễ dàng thử nghiệm và áp dụng các công nghệ mới.
* Một dịch vụ bị lỗi sẽ không ảnh hưởng đến các dịch vụ còn lại.

### Container hóa

Container hóa là một quy trình triển khai phần mềm với khả năng đóng gói mã của ứng dụng cùng tất cả các tệp và thư viện cần thiết để chạy trên bất kỳ cơ sở hạ tầng nào. Theo cách truyền thống, để chạy bất kỳ ứng dụng nào trên máy tính của bạn, bạn sẽ phải cài đặt phiên bản phù hợp với hệ điều hành trên máy tính của bạn. Ví dụ: bạn sẽ cần cài đặt phiên bản Windows của một gói phần mềm trên thiết bị chạy Windows. Tuy nhiên, với container hóa, bạn có thể tạo một gói phần mềm duy nhất, hay còn gọi là [bộ chứa](https://aws.amazon.com/containers), chạy trên tất cả các loại thiết bị và hệ điều hành.

Docker được sử dụng phổ biến trong ngành công nghiệp phần mềm vì những tính năng tuyệt vời mà nó mang lại.

* Một số thuật ngữ được dùng trong Docker:
  + Image: Đây là một khuôn mẫu chứa các tập tin cần thiết để tạo một container.
  + Container: Đây là một instance của image. Nó có thể được triển khai, chạy và xóa bỏ thông qua remote client.
  + Docker Engine: Đây là một công nghệ container hóa mã nguồn mở giúp xây dựng và đóng gói ứng dụng. Docker Engine hoạt động như một ứng dụng sử dụng mô hình máy khách - máy chủ16.
  + Docker Hub: Đây là kho lưu trữ các docker image. Nó cung cấp tài nguyên một cách tập trung cho việc tìm kiếm image, phân phối và quản lý các thay đổi của image.
  + Docker Machine: Đây là công cụ giúp cài đặt Docker Engine trên máy tính người dùng, các dịch vụ điện toán đám mây, trung tâm dữ liệu.
  + Dockerfile: Đây là tập tin tài liệu văn bản chứa các hướng dẫn để Docker có thể build image một cách tự động.
* Các thành phần của Docker:
  + Docker Daemon: Đây là một tiến trình chạy ngầm trên máy chủ để quản lý các container.
  + Docker Client: Đây là một công cụ cung cấp giao diện dòng lệnh (command line) để người dùng tương tác với daemon. Docker Client kiểm soát hầu hết các luồng công việc (workflow) của Docker.
  + Docker Engine: Đây là một công nghệ container hóa mã nguồn mở giúp xây dựng và đóng gói ứng dụng. Docker Engine hoạt động như một ứng dụng sử dụng mô hình máy khách - máy chủ16.
  + Docker Hub: Đây là kho lưu trữ các docker image. Nó cung cấp tài nguyên một cách tập trung cho việc tìm kiếm image, phân phối và quản lý các thay đổi của image.
  + Docker Machine: Đây là công cụ giúp cài đặt Docker Engine trên máy tính người dùng, các dịch vụ điện toán đám mây, trung tâm dữ liệu.
  + Docker Compose: Đây là công cụ dùng để định nghĩa và chạy nhiều container cho Docker Application.
  + Docker Swarm: Đây là công cụ native clustering cho Docker, cho phép gom một số Docker host lại với nhau thành dạng cụm (cluster).

A computer screen with text and images

Description automatically generated with medium confidenceHình 2.4: Kiến trúc Docker

### 2.4.3. Snyk

Snyk là công cụ quét rủi ro lỗ hổng bảo mật tại thời điểm xây dựng CLI hỗ trợ Ruby và các ngôn ngữ khác với nhiều mặc định an toàn. Liên tục và tự động tìm kiếm, sửa chữa và giám sát các lỗ hổng trong các phần phụ thuộc nguồn mở trong suốt quá trình phát triển của bạn. Bảo mật trên quy mô lớn yêu cầu các nhà phát triển ứng dụng phải là bước đầu tiên trong quy trình bảo mật để kiểm tra lỗ hổng trang web. Bảo mật ứng dụng nguồn mở Snyk giúp các nhà phát triển phát triển nhanh chóng và luôn an toàn. Bảo mật tất cả các thành phần của ứng dụng gốc trên nền tảng đám mây hiện đại trong một công cụ quét mã nguồn mở Snyk duy nhất. Nền tảng bảo mật ứng dụng đám mây của Snyk được xây dựng nhằm mục đích giúp các nhà phát triển nguồn mở dễ dàng sử dụng nhằm phát triển an toàn và không có rủi ro ở quy mô và tốc độ.

Giải pháp bảo mật ứng dụng web Snyk đang giúp các nhà phát triển sử dụng các phần phụ thuộc nguồn mở và luôn được bảo mật. Snyk là phần mềm bảo mật ứng dụng đám mây miễn phí dành cho nguồn mở. Snyk tự động tìm, sửa, giám sát và ngăn chặn các lỗ hổng trong các ứng dụng Ruby, Node.js, Java, Python và Scala của bạn. Snyk giám sát và theo dõi các lỗ hổng trong hơn 800.000 phần mềm nguồn mở và giúp bảo vệ hơn 25.000 ứng dụng trực tuyến. 83% người dùng ứng dụng quét lỗ hổng ứng dụng web Snyk đã tìm thấy rủi ro và lỗ hổng trong ứng dụng trực tuyến của họ và các lỗ hổng mới được tiết lộ thường xuyên, khiến ứng dụng của bạn gặp rủi ro.

Để cài đặt công cụ CLI tiện ích Snyk mã nguồn mở, bạn sẽ cần cài đặt các phụ thuộc và điều kiện tiên quyết sau:

* + - * + Một dự án có ngôn ngữ được hỗ trợ Snyk, ví dụ: Dự án Ruby
        + Code sử dụng các gói nguồn mở
        + Dự án được triển khai trên hệ thống quản lý mã nguồn được hỗ trợ, ví dụ: GitHub
        + Tạo tài khoản Snyk bằng cách truy cập trang Snyk

Một số danh sách tính năng tuyệt vời của Snyk CLI và công cụ tại thời điểm xây dựng để tìm và sửa các lỗ hổng đã biết trong các phần phụ thuộc nguồn mở là:

* + - * + Tìm lỗ hổng bằng cách chạy thử nghiệm snyk trên một dự án trong quy trình CI.
        + Khắc phục các lỗ hổng bằng cách sử dụng trình hướng dẫn snyk và bảo vệ snyk.
        + Trình hướng dẫn Snyk hướng dẫn bạn cách tìm và sửa các lỗ hổng đã biết trong một dự án.
        + Trình giám sát cảnh báo snyk ghi lại trạng thái phụ thuộc và mọi lỗ hổng trên snyk.
        + Ngăn chặn các phần phụ thuộc dễ bị tấn công mới bằng cách chạy thử nghiệm snyk trong quy trình CI khi các phần phụ thuộc Node.js hoặc Ruby dễ bị tấn công được thêm vào.

### 2.5. Trí tuệ nhân tạo

Khái niệm trí tuệ nhân tạo, học máy và học sâu có một mối quan hệ mật thiết với nhau và thường bị nhầm lẫn là một, thực tế bản thân chúng đều ám chỉ những khái niệm riêng biệt. Để giúp cho việc hình dung cụ thế hơn, có thể xem học sâu là một tập hợp con của học máy, trí tuệ nhân tạo là một tập hợp lớn hơn của học máy và học sâu.

A diagram of machine learning

Description automatically generated

Hình 2.5: Mô hình tương quan giữa trí tuệ nhân tạo, học máy và học sâu

### 2.5.1. Học máy

Học máy là lĩnh vực của trí tuệ nhân tạo, thuật ngữ học máy đề cập đến một tập hợp các phương pháp cung cấp cho máy tính “khả năng học mà không cần được lập trình một cách rõ ràng”. Học máy chủ yếu tập trung vào việc sử dụng dữ liệu và thuật toán để bắt chước cách mà con người học, từ đó dần dần cải thiện độ chính xác của mình. Dựa trên phương pháp tiếp cận, có thể chia học máy thành bốn nhóm cơ bản là học có giám sát (Supervised Learning), học không giám sát (Unsupervised Learning), học bán giám sát (Semi–supervised Learning) và học tăng cường (Reinforcement Learning). Sự khác biệt giữa các phương pháp tiếp cận học máy này nằm ở các thuật toán được sử dụng, dữ liệu đầu vào và loại vấn đề mà chúng giải quyết.

Các phương pháp được mô tả như sau:

**Học có giám sát:** là thuật toán dự đoán đầu ra (còn được gọi là outcome) của một dữ liệu mới (new, input) dựa trên các cặp (input, outcome) đã biết từ trước. Cặp dữ liệu này được gọi là dữ liệu và nhãn (data, label). Học có giám sát được chia làm hai nhóm nhỏ là phân lớp (Classification) và hồi quy (Regression).

***•*** Phân lớp: được sử dụng khi đầu ra mong muốn là một tập hữu hạn và rời rạc. Một ví dụ cụ thể trong bài toán phát hiện hành động con người, số lượng các lớp (class) có thể là: nhị phân (tức là chỉ cần phân loại tất cả các hành vi con người thành có hành động hoặc không có hành động) và đa lớp (tức là ngoài việc phân loại hành động bao gồm có hay không xảy ra hành động thì sau đó các hành động xảy ra còn phải được phân loại cụ thể như đứng lên, chạy, đi, ngồi xuống, …).

***•*** Hồi quy: được sử dụng khi đầu ra mong muốn là một dải giá trị liên tục. Ví dụ trong bài toán tính giá trị xác suất, giả sử bài toán cần giải quyết là điểm đáng tin cậy của người dùng trong việc cho vay tín dụng. Điểm tin cậy đại diện cho khả năng người đó có thể chi trả cho khoản vay của họ và nằm trong khoảng từ 0 đến 1. Điều đó thể hiện xác suất một người sẽ trả các khoản vay của họ.

**Học không giám sát:** là thuật toán dựa vào cấu trúc của dữ liệu đầu vào để khai phá những quy luật ẩn bên trong các tập dữ liệu đầu vào. Các quy luật như phân

nhóm hay giảm số chiều dữ liệu nhằm cải thiện khả năng tính toán. Thuật toán này không có dữ liệu đầu ra hay nhãn mà chỉ biết được dữ liệu đầu vào. Học không giám sát được chia làm hai nhóm nhỏ là phân nhóm (Clustering) và kết hợp (Association).

*•* Phân nhóm: được sử dụng khi muốn phân toàn bộ dữ liệu bất kỳ thành các nhóm nhỏ dựa trên sự kiện liên quan giữa các dữ liệu trong mỗi nhóm, chẳng hạn như phân nhóm khách hàng dựa trên hành vi mua hàng.

*•* Kết hợp: được sử dụng khi muốn khai phá các quy luật của cấu trúc dữ liệu, chẳng hạn như người mua món hàng A cũng có xu hướng mua món hàng B.

**Học bán giám sát:** là thuật toán kết hợp giữa học có giám sát và không giám sát. Thuật toán này cho phép dữ liệu đầu vào là hỗn hợp giữa các dữ liệu được gắn nhãn và không gắn nhãn. Áp dụng học bán giám sát sẽ giúp tiết kiệm thời gian và chi phí cho việc gắn nhãn dữ liệu.

**Học tăng cường:** là thuật toán giúp cho một hệ thống tự động xác định hành vi dựa trên ngữ cảnh để đạt được mục đích cao nhất. Học tăng cường thường được ứng dụng trong lý thuyết trò chơi (Game Theory), trong đó, học tăng cường cần xác định nước đi tiếp theo để đạt được điểm số cao nhất qua mỗi lần chơi.

### 2.5.2. Học sâu

Như đã đề cập, học sâu là một nhánh nhỏ của lĩnh vực học máy và sự khác biệt giữa chúng được thể hiện rõ ràng ở cách học. Cụ thể, học sâu tự động hoá phần lớn quy trình trích xuất thuộc tính và cho phép sử dụng các tập dữ liệu lớn hơn rất nhiều so với học máy. Học sâu có thể nhận đầu vào là các tập dữ liệu phi cấu trúc ở dạng thô như văn bản, hình ảnh, … và tự động xác định tập hợp các thuộc tính giúp phân biệt các dữ liệu với nhau. Chính vì lý do đó mà học sâu chủ yếu được ứng dụng trong lĩnh vực thị giác máy tính, xử lý ngôn ngữ tự nhiên và nhận dạng giọng nói.

A diagram of a network

Description automatically generated**Mạng neural:** là một mô hình được lấy cảm hứng từ cấu trúc và chức năng của não bộ con người, là kết nối giữa các nơ-ron thần kinh trong sinh học. Trong hình bên dưới, mỗi ô tròn đại diện cho một neural (còn được gọi là một nút), mỗi neural có một giá trị riêng.

2.5.1 – Mô hình học sâu

Giá trị của neural được tính toán thông qua giá trị của các neural trước đó và cũng là đầu vào cho các neural phía sau. Nhìn chung, mỗi neural có thể nhận một hoặc nhiều giá trị làm giá trị đầu vào và thực hiện các phép tính toán dựa trên giá trị đó, và kết quả của phép tính là giá trị của neural. Giá trị của các neural đầu tiên (còn được gọi là input layer) được lấy từ đầu vào mà không cần phải thông qua bước tính toán, trong khi đó giá trị của neural cuối cùng (còn được gọi là output layer) sẽ là kết quả cuối cùng của toàn bộ quá trình tính toán.

## CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ VÀ THIẾT KẾ MÔ HÌNH

### Mô hình tổng quan

Trên cơ sở lý thuyết và các kiến thức đã tìm hiểu về mô hình DevSecOps và để triển khai mô hình đảm bảo được an toàn bảo mật thông tin cho một quy trình phát triển và vận hành ứng dụng. Mô hình DevSecOps dựa trên các tính chất của quy trình DevOps, nhóm tác giả đã tích hợp các công cụ kiểm thử bảo mật và công cụ sử dụng học máy để phân tích, phát hiện các nguy cơ ảnh hưởng đến bảo mật trên ứng dụng khi triển khai.

A diagram of a security system

Description automatically generatedMô hình tổng quan được biểu diễn trong hình 3.1. bao gồm nhiều thành phần nhằm thực hiện quy trình CI/CD và bổ sung khả năng phát hiện các nguy cơ bảo mật.

Hình 3.1: Mô hình tổng quan

Mô hình bao gồm:

* + - * + Code repository: lưu trữ mã nguồn của ứng dụng web, quản lý phiên bản, tiếp nhận các sự thay đổi trong mã nguồn và trong kịch bản tự động hóa.
        + CI platform: để thực hiện quá trình kiểm thử bảo mật, đóng gói ứng dụng và triển khai ứng dụng trên môi trường phù hợp một cách tự động.
        + Repository of container images: lưu trữ container images, quản lý phiên bản và đảm bảo tính riêng tư.
        + Dashboard: hiển thị các cảnh báo bảo mật với sự thay đổi trong mã nguồn.

Mô hình mô tả một quy trình DevSecOps bao gồm 6 bước:

1. Người phát triển ứng dụng thực hiện những thay đổi trong mã nguồn và tải lên *Code repository* (commit).
2. Thực hiện *Automated security testing* ngay khi nhận ra sự thay đổi, bao gồm nhiều phương thức kiểm thử bảo mật.
3. Chuyển kết quả kiểm thử bảo mật sang *Dashboard* để hiển thị và cho phép người phát triển nhận thức về các nguy cơ bảo mật.
4. Đóng gói phiên bản mới của ứng dụng dưới dạng container image nếu đạt điều kiện bảo mật.
5. Đẩy container image tới *Repository of container images* để lưu trữ và quản lý theo tag.
6. Người quản lý xem xét các cảnh báo bảo mật và đồng thời triển khai ứng dụng xuống môi trường production.

Mô hình tổng quan có thể triển khai cho các quy trình phát triển phần mềm khác nhau tùy thuộc đặc tính của ngôn ngữ lập trình cũng như những công nghệ được lựa chọn để triển khai. Mô hình tổng quát được ứng dụng để phát hiện các yếu tố bảo mật này.

### Mô hình chi tiết

Mô hình này hướng đến việc phát hiện mẫu lập trình kém trong quy trình phát triển và vận hành ứng dụng.

A screenshot of a computer screen

Description automatically generated

Hình 3.2: Mô hình chi tiết Mô hình bao gồm các thành phần:

* Github:
  + Phân chia nhánh, quản lý mã nguồn của ứng dụng.
  + Lưu trữ jenkinsfile và yaml để cấu hình hệ thống.
  + Thông báo sự thay đổi trong mã nguồn đến *Jenkins* qua webhook.
* Jenkins:
  + Nhận webhook từ *Github* để thực thi pipeline.
  + Thực thi *Jenkins pipeline*.
* SonarQube server:
  + Tiếp nhận các yêu cầu quét mã nguồn từ *Jenkins*.
  + Lưu trữ các báo cáo kết quả quét bằng SonarQube.
  + Xác định mã nguồn có thỏa điều kiện bảo mật hay không.
* Snyk:
  + Tiếp nhận các yêu cầu quét bảo mật web từ *Jenkins*.
  + Quét các lỗ hổng bảo mật.
  + Sử dụng: SonarQube, Snyk.

### Phân tích và thiết kế Jenkins Pipeline

Jenkins pipeline được khởi tạo từ Jenkinsfile bao gồm các bước cần thiết để thực hiện CI/CD và tích hợp phát hiện các vấn đề bảo mật bằng các công cụ kiểm thử bảo mật tự động.

A diagram of a software development process

Description automatically generated

Hình 3.3: Thiết kế Jenkins pipeline

1. A github webhook
   * Developer đẩy code lên Github khiến Jenkins kích hoạt pipeline.
   * Pipeline thực thi theo quy trình đã định sẵn trong Jenkinsfile.
2. Run Sonar and Snyk
   * Sử dụng SonarQube và SonarScanner.
   * Kết nối SonarScanner đến SonarQube server để yêu cầu server tiến hành quét các lỗ hổng bảo mật có trong mã nguồn. Kiểm tra các yêu cầu về tính năng của ứng dụng.
   * Thực hiện các test case tự động bằng python.
   * Tự động dừng pipeline khi ứng dụng không vượt qua test case.
3. Security gate
   * Kiểm tra điều kiện bảo mật của ứng dụng.
   * Tự động dừng pipeline khi phát hiện lỗi.
4. Create production images
   * Tương tự như bước *Create staging images*.
   * Thay đổi nhãn (tag) của staging image để triển khai xuống môi trường production.
5. Deploy to production
   * Thực hiện sau khi thỏa điều kiện ở security gate.
   * Triển khai ứng dụng web xuống môi trường production.

## CHƯƠNG 4. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

### Triển khai mô hình

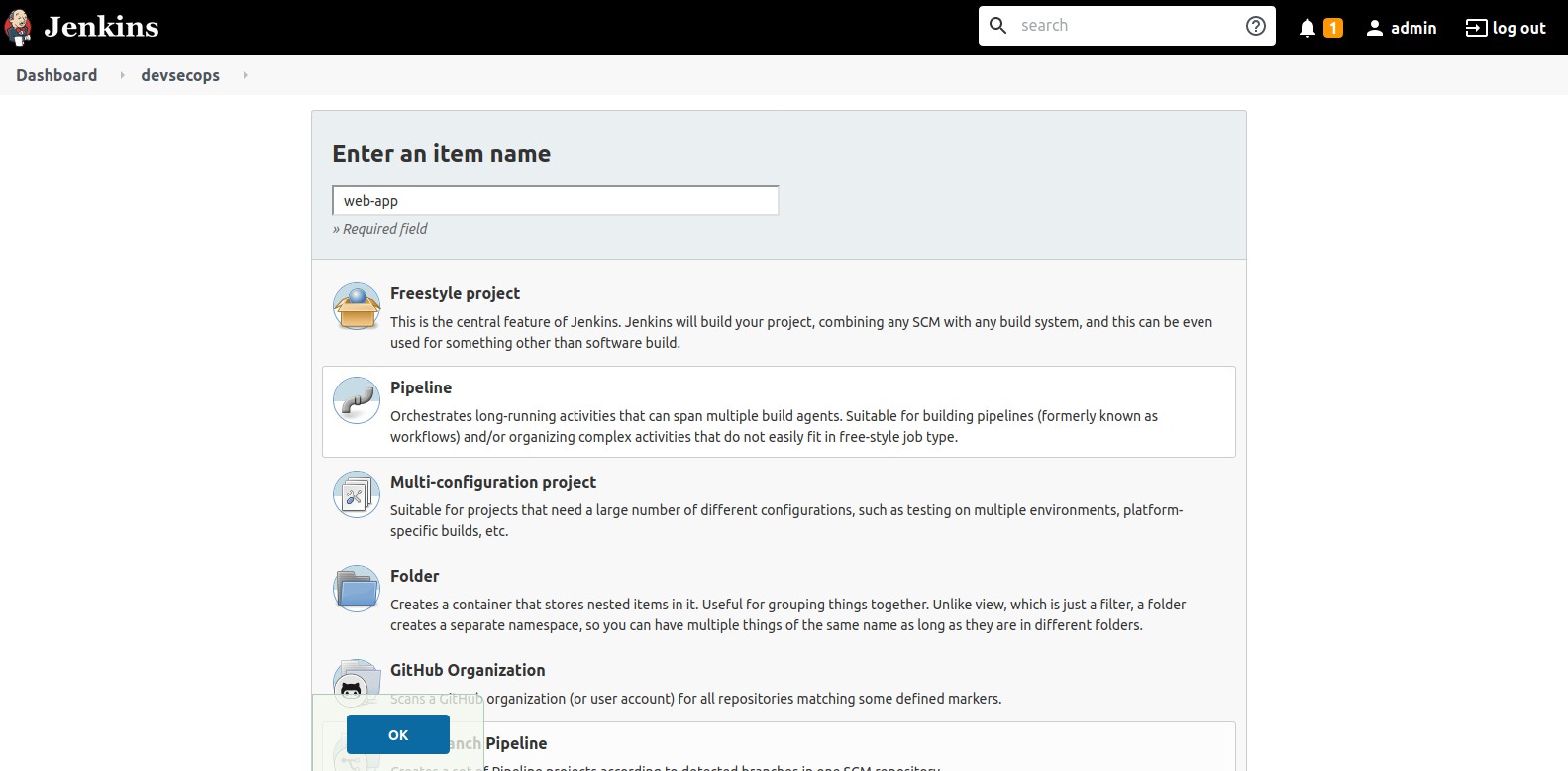
Người quản lý dự án yêu cầu triển khai một trang web giới thiệu sản phẩm xuống môi trường sản phẩm và phải đảm bảo an toàn thông tin, phát hiện các lỗ hổng bảo mật web. Đội ngũ DevOps nhận được yêu cầu, tiến hành thực hiện các bước trong quy trình phát triển và vận hành phần mềm để triển khai ứng dụng web đó. Quy trình được mô tả trong hình 4.1.

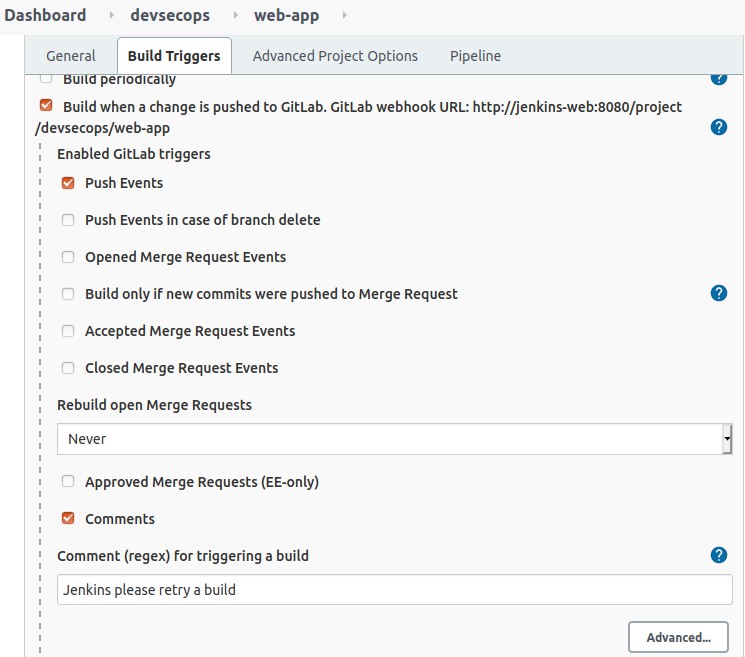
A diagram of a process

Description automatically generated

Hình 4. 1 Mô hình triển khai ứng dụng

Toàn bộ quy trình được triển khai cụ thể qua các bước sau:

***Bước 1:*** Tạo mới một dự án Jenkins. Tạo Jenkins pipeline.

Hình 4.2: Tạo mới Jenkins pipeline Cấu hình Jenkins pipeline.

Hình 4.3: Cấu hình Jenkins pipeline

***Bước 2:*** Tạo webhook trên Github.

Webhook này sẽ được thực thi khi có sự thay đổi trên nhánh *main*

***Bước 3:*** Đẩy mã nguồn của trang web lên Github.

Đây là mã nguồn có lỗ hổng bảo mật và nguy cơ gây mất an toàn thông tin. Thực hiện *git push* đến repository mà kỹ sư DevOps đã tạo ở trên. Sau khi đẩy mã nguồn thành công, quá trình tích hợp và triển khai được diễn ra tự động.

***Bước 4:*** Thực thi Jenkins pipeline tự động.

Quá trình Jenkins pipeline được bắt đầu ngay khi có sự thay đổi mã nguồn. Quá trình CI/CD và quét bảo mật được thực thi qua từng bước. Tiến hành vào dự án trên Jenkins để kiểm tra.

### Cấu hình các ứng dụng

Để các thành phần trong mô hình có thể liên kết với nhau cần thực hiện một số bước cấu hình:

***Bước 1:*** Thực hiện tại SonarQube.

* + - Lấy token của tài khoản admin/admin.
    - Tạo webhook đến Jenkins khi có kết quả quét.

**2**

Hình 4.2.1: Cấu hình webhook trên SonarQube

A screenshot of a webbook

Description automatically generated***Bước 2:*** Thực hiện tại Github.

**1**

**1**

* + - * + Lấy token tài khoản.
        + A screenshot of a computer

          Description automatically generatedTạo mới repository của dự án trên Github để lưu trữ mã nguồn.

**2**

**3**

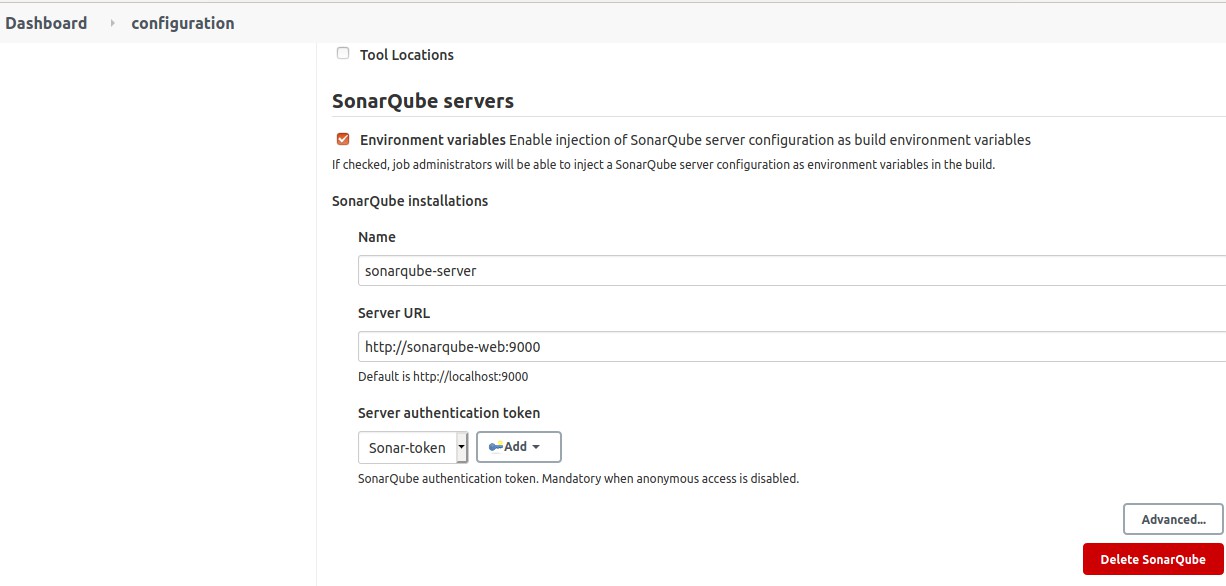
**1**

Hình 4.2.2: Tạo mới repository trên Github.

* + - * + Sau khi đã tạo repository, kỹ sư DevOps thông báo cho Developer về repository vừa tạo. Developer được yêu cầu lưu trữ mã nguồn và quản lý tại repository này.

***Bước 3:*** Thực hiện tại Jenkins.

* + - * + Cài đặt các plugin cần thiết bao gồm: Github, SonarQube Scanner, Snyk và các plugin mặc định của Jenkins.
        + Cấu hình SonarQube Scanner plugin.



**3**

**2**

**1**

Hình 4.2.3: Cấu hình SonarQube trên Jenkins

* + - * + A screenshot of a computer

          Description automatically generatedCấu hình Github plugin.

**4**

**2**

**1**

**3**

Hình 4.2.4: Cấu hình Github trên Jenkins

### 4.3. Lỗ hổng XSS

Cross Site Scripting (XSS) là một trong những tấn công phổ biến và dễ bị tấn công nhất mà tất cả các Tester có kinh nghiệm đều biết đến. Nó được coi là một trong những tấn công nguy hiểm nhất đối với các ứng dụng web và có thể mang lại những hậu quả nghiêm trọng. Giới thiệu về tấn công XSS Tấn công XSS là một đoạn mã độc, để khái thác một lỗ hổng XSS, hacker sẽ chèn mã độc thông qua các đoạn script để thực thi chúng ở phía Client. Thông thường, các cuộc tấn công XSS được sử dụng để vượt qua truy cập và mạo danh người dùng.

Mục đích chính của cuộc tấn công này là ăn cắp dữ liệu nhận dạng của người dùng như: cookies, session tokens và các thông tin khác. Trong hầu hết các trường hợp, cuộc tấn công này đang được sử dụng để ăn cắp cookie của người khác. Như chúng ta biết, cookie giúp chúng tôi đăng nhập tự động. Do đó với cookie bị đánh cắp, chúng tôi có thể đăng nhập bằng các thông tin nhận dạng khác. Và đây là một trong những lý do, tại sao cuộc tấn công này được coi là một trong những cuộc tấn công nguy hiểm nhất.

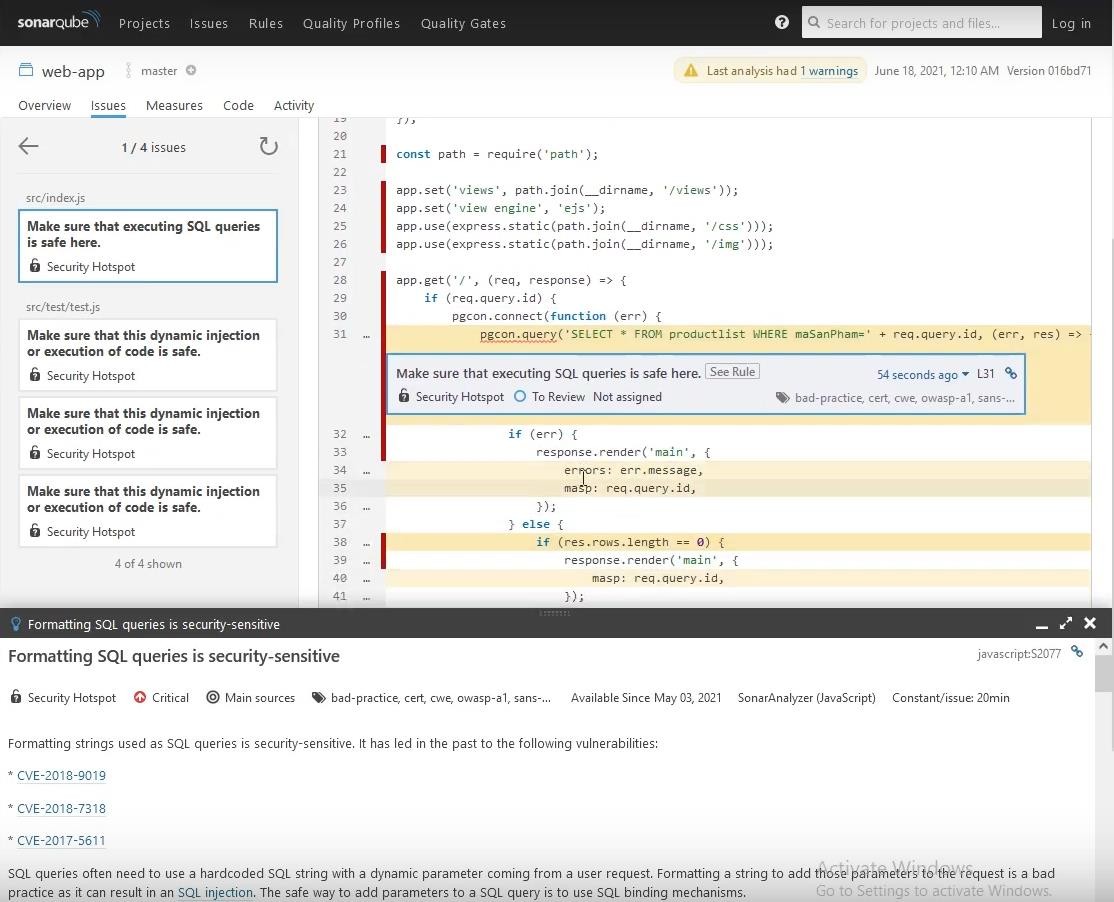
Để mô phỏng lỗ hổng Reflected XSS, toàn bộ dữ liệu mà người dùng nhập vào ô tìm kiếm sẽ trở thành tham số để thực hiện truy vấn tìm kiếm sản phẩm. Truy vấn này thực thi ở dạng URL. Sau đó, kết quả phản hồi từ server sẽ bao gồm dữ liệu người dùng nhập vào kèm theo kết quả tìm kiếm. Trình duyệt người dùng hiển thị toàn bộ kết quả trả về mà không có sự phân biệt về loại dữ liệu. Điều này được minh họa thông qua câu lệnh sau:

*response.render(‘main’, {masp: req.query.id})*

Với câu lệnh này server sẽ render thành HTML để trả về cho máy khách. Tham số *masp* là toàn bộ dữ liệu người tấn công nhập vào một đoạn mã JavaScript, ví dụ như:dùng nhập vào. Vì vậy, nếu kẻ

*<script>alert(‘XSS’)</script>*

### 4.4. Kết quả thử nghiệm

SonarQube đã phát hiện SQLi trong mã nguồn web, mô tả lỗi có thể xảy ra

hiện XSS.

### 4.4.1. Triển khai ứng dụng

A screenshot of a computer

Description automatically generatedToàn bộ quá trình triển khai ứng dụng web được thực hiện bởi Jenkins. Các bước đã chạy trên Jenkins được thể hiện ở hình 4.21.

Hình 4.21: Kết quả Jenkins pipeline

Pipeline sẽ tự động dừng nếu phát hiện lỗi bảo mật hoặc trang web không đạt yêu cầu về chức năng. Khi đó, phiên bản ứng dụng sẽ không được triển khai xuống môi trường production.

### 4.4.2. Đánh giá mô hình đã triển khai

### 4.4.2.1 Ưu điểm của mô hình đã triển khai

Mô hình đáp ứng yêu cầu của quy trình phát triển phần mềm nhanh và tự động hóa trong hầu hết các bước:

* + - * Triển khai ứng dụng xuống môi trường sản phẩm một cách nhanh nhất.

Quá trình triển khai không làm gián đoạn dịch vụ.

* + - * Phát hiện các lỗi bảo mật và nguy cơ gây mất an toàn thông tin trên ứng dụng.

### 4.4.2.2. Nhược điểm của mô hình đã triển khai

Các thành phần trong mô hình tồn tại những nhược điểm sau:

* + - * Các công cụ quét bảo mật dựa trên luật (rule based), hạn chế khả năng xác định các lỗi bảo mật mới hoặc lỗi có nhiều cách thể hiện.
      * Số lượng cảnh báo nhầm lẫn (false positive) trên các công cụ ảnh hưởng đến kết quả đánh giá tự động. Cần sự can thiệp của con người để xác định sai sót trong quá trình phát hiện lỗi.
      * Công cụ bảo mật phân tích động tốn nhiều thời gian để quét các lỗ hổng. Cần thu hẹp phạm vi quét của công cụ cho một lỗi cụ thể.

## CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### Kết luận

Qua quá trình nghiên cứu và triển khai mô hình đảm bảo an toàn thông tin trong quy trình phát triển và vận hành phần mềm, nhóm tác giả đã đạt được những kết quả như sau:

* + - Nắm được kiến thức tổng quan về quy trình phát triển phần mềm, sử dụng các công nghệ tự động hóa để thực hiện quá trình CI/CD .
    - Sử dụng các công cụ kiểm thử bảo mật và tích hợp vào quá trình CI/CD để đánh giá bảo mật tự động cho ứng dụng web.
    - Tận dụng được các ưu điểm của Kubernetes trong việc điều phối và vận hành hệ thống.
    - Triển khai mô hình cho một dự án phát triển web cụ thể và đánh giá ưu nhược điểm của mô hình.

### Hướng phát triển

Từ những kết quả đã đạt được, nhóm tác giả đã đề ra các hướng phát triển để nâng cao hiệu suất của mô hình.

* + - Bổ sung các công cụ bảo mật kiểm tra tính an toàn trong các file cấu hình hệ thống.
    - Chạy đồng thời nhiều kiểm thử cùng lúc để tăng tốc độ của mô hình.
    - Thêm nhiều chế độ quét để phù hợp với các dự án khác nhau.
    - Thay đổi điều kiện quét theo cách linh hoạt hơn. Điều kiện quét có thể điều chỉnh trên trang quản lý bảo mật tập trung để phù hợp với mục đích quét.

## TÀI LIỆU THAM KHẢO

1. “IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries,” *IEEE Std 610*. pp. 1–217, 1991, doi: [10.1109/IEEESTD.1991.106963](https://doi.org/10.1109/IEEESTD.1991.106963).
2. A. Stellman and J. Greene, *Applied software project management*. “ O’Reilly Media, Inc.,” 2005.
3. B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” *J. Syst. Softw.*, vol. 123, pp. 176–189, 2017.
4. M. Shahin, M. A. Babar, and L. Zhu, “Continuous Integration, Delivery and

Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: [10.1109/ACCESS.2017.2685629](https://doi.org/10.1109/ACCESS.2017.2685629).

1. P. Nielsen, T. Winkler, and J. Nørbjerg, *Closing the IT Development- Operations Gap: The DevOps Knowledge Sharing Framework*. 2017.
2. V. Lenarduzzi, A. Sillitti, and D. Taibi, “Analyzing Forty Years of Software Maintenance Models,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 146–148, doi: [10.1109/ICSE-C.2017.122](https://doi.org/10.1109/ICSE-C.2017.122).

## PHỤ LỤC

### Mã nguồn chứa lỗ hổng bảo mật

https://github.com/nhdang002/uitdev/