

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

LƯU MINH THIỆN – 21521460

NGUYỄN THÀNH ĐĂNG – 21520683

QUÁCH THỊ HOÀI PHƯƠNG – 21520409

TRẦN KHÔI NGUYÊN – 21522397



ĐỒ ÁN MÔN HỌC

**TRIỂN KHAI MÔ HÌNH DEVSECOPS CHO HỆ THỐNG CI/CD CÓ ĐỘ
SẴN SÀNG CAO TRONG MÔI TRƯỜNG MICROSERVICES**

MÔN: CÔNG NGHỆ DEVOPS VÀ ỨNG DỤNG

LỚP: NT548.P11

GIẢNG VIÊN HƯỚNG DẪN

LÊ ANH TUẤN

TP. HỒ CHÍ MINH, 2024

MỤC LỤC

Chương 1. GIỚI THIỆU	13
1.1. Lý do chọn đề tài.....	13
1.2. Phạm vi nghiên cứu.....	14
1.3. Tóm tắt đồ án	14
Chương 2. CƠ SỞ LÝ THUYẾT.....	16
2.1. Tổng quan về DevOps.....	16
2.1.1. DevOps là gì?.....	16
2.1.2. Lịch sử phát triển của DevOps	16
2.1.3. Mục tiêu và lợi ích của DevOps.....	18
2.1.4. Các nguyên tắc chính của DevOps.....	19
2.1.5. Giới thiệu về DevSecOps.....	20
2.2. Kiến trúc microservices.....	21
2.3. CI/CD và tự động hóa quy trình.....	22
2.3.1. Khái niệm CI/CD	22
2.3.2. Vai trò của CI/CD trong DevOps	24
2.4. Công cụ và công nghệ liên quan.....	25
2.4.1. GitHub (Version Control Systems)	25
2.4.2. Quản lý hạ tầng và cấu hình	26
2.4.2.1. Terraform.....	26
2.4.2.2. Ansible.....	27
2.4.3. Các công cụ cho CI/CD	28
2.4.3.1. FluxCD	28
2.4.3.2. Github Action.....	29

2.4.4.	Quản lý container và orchestration.....	30
2.4.4.1.	Docker.....	30
2.4.4.2.	Kubernetes	31
2.4.5.	Quản lý lưu trữ.....	33
2.4.5.1.	Harbor	33
2.4.5.2.	MinIO.....	34
2.4.6.	Công cụ bảo mật.....	35
2.4.6.1.	Checkov.....	35
2.4.6.2.	Snyk	36
2.4.6.3.	SonarQube	37
2.4.6.4.	Trivy	39
2.4.6.5.	Vault	39
2.4.7.	Công cụ giám sát	42
2.4.7.1.	Prometheus.....	42
2.4.7.2.	Loki	44
2.4.7.3.	Grafana	46
2.4.8.	AWS (Amazon Web Service)	47
2.4.8.1.	Amazon VPC.....	47
2.4.8.2.	Amazon EC2	49
2.4.8.1.1.	EC2 Auto Scaling.....	50
2.4.8.1.2.	Elastic Load Balancing.....	51
2.4.8.3.	Amazon EKS	53
Chương 3.	PHÂN TÍCH THIẾT KẾ.....	55
3.1.	Mô hình mạng triển khai.....	55

3.2.	Quy trình triển khai hạ tầng	57
3.3.	Mô hình triển khai hệ thống	58
3.4.	Ứng dụng microservices	62
3.5.	Quản lý tên miền	66
3.6.	Quản lý mã nguồn	67
Chương 4. HIỆN THỰC ĐỀ TÀI.....		69
4.1.	Triển khai cơ sở hạ tầng	69
4.2.	Cấu hình cơ sở hạ tầng	74
4.3.	Cấu hình các máy chủ	75
4.3.1.	Cấu hình HA Proxy	75
4.3.2.	Nhóm công cụ lưu trữ	77
4.3.2.1.	Cấu hình MinIO	77
4.3.2.2.	Cấu hình Harbor	79
4.3.3.	Nhóm công cụ bảo mật	81
4.3.3.1.	Cấu hình Snyk	81
4.3.3.2.	Cấu hình SonarQube	83
4.3.3.3.	Cấu hình Trivy	85
4.3.3.4.	Cấu hình Vault	88
4.3.4.	Nhóm công cụ giám sát	90
4.3.4.1.	Cấu hình Prometheus	90
4.3.4.2.	Cấu hình Loki	92
4.3.4.3.	Cấu hình Grafana	96
4.4.	Triển khai quy trình CI/CD	100
4.4.1.	Quy trình CI	100

4.4.2. Quy trình CD.....	107
Chương 5. KIỂM THỬ	114
5.1. Kiểm thử triển khai hạ tầng.....	114
5.2. Kiểm thử CI/CD.....	114
5.3. Kiểm thử tính sẵn sàng.....	114
Chương 6. KẾT LUẬN & HƯỚNG PHÁT TRIỂN.....	116
6.1. Kết luận.....	116
6.2. Hướng phát triển.....	116

DANH MỤC HÌNH VẼ

Hình 1. DevOps	16
Hình 2. Lịch sử phát triển của DevOps	17
Hình 3. DevSecOps	20
Hình 4. Kiến trúc Microservice.....	21
Hình 5. CI/CD pipeline	23
Hình 6. CD pipeline	23
Hình 7. Luồng hoạt động của GitHub.....	25
Hình 8. Terraform	26
Hình 9. Ansible.....	27
Hình 10. Quy trình hoạt động của FluxCD	28
Hình 11. GitHub Actions.....	29
Hình 12. Kiến trúc Docker	30
Hình 13. Kiến trúc Kubernetes.....	31
Hình 14. Harbor	33
Hình 15. MinIO	34
Hình 16. Checkov	35
Hình 17. Snyk.....	36
Hình 18. SonarQube.....	37
Hình 19. Trivy	39
Hình 20. Vault	39
Hình 21. Mô hình của Vault	40
Hình 22. Hoạt động của Vault	41
Hình 23. Prometheus.....	42
Hình 24. Prometheus model	42
Hình 25. Grafana Loki	44
Hình 26. Grafana Loki model	45
Hình 27. Grafana.....	46

Hình 28. Kiến trúc Amazon VPC	47
Hình 29. Amazon EC2.....	49
Hình 30. EC2 Auto Scaling.....	50
Hình 31. Mô hình Auto Scaling Group.....	51
Hình 32. Elastic Load Balancing	51
Hình 33. Kiến trúc Amazon EKS	53
Hình 34. Mô hình mạng triển khai.....	55
Hình 35. Mô hình triển khai hạ tầng.....	57
Hình 36. Mô hình triển khai hệ thống	58
Hình 37. Ứng dụng microservice	62
Hình 38. Danh sách các API của ứng dụng.....	63
Hình 39. Màn hình xác thực người dùng	64
Hình 40. Màn hình đăng nhập	64
Hình 41. Đăng ký	65
Hình 42. Màn hình quản lý sinh viên.....	65
Hình 43. Màn hình quản lý giảng viên	66
Hình 44. Màn hình quản lý lớp học	66
Hình 45. Quản lý tên miền.....	67
Hình 46. Quản lý mã nguồn	68
Hình 47. Triển khai cơ sở hạ tầng.....	70
Hình 48. GitHub Actions được trigger khi có commit mới lên source Terraform 71	
Hình 49. Workflow cho pipeline triển khai.....	71
Hình 50. Quét bảo mật với Checkov	71
Hình 51. Kết quả sau khi scan với Checkov.....	72
Hình 52. Thông tin chi tiết của tệp SARIF	72
Hình 53. Kiểm tra thông tin các EC2.....	72
Hình 54. Kiểm tra thông tin VPC	73
Hình 55. Kiểm tra thông tin Auto Scaling Group.....	73

Hình 56. Kiểm tra thông tin EKS	73
Hình 57. Inventory file	74
Hình 58. Cấu hình cơ sở hạ tầng.....	75
Hình 59. Cấu hình HA Proxy: Frontend.....	75
Hình 60. Cấu hình HA Proxy: Backend	76
Hình 61. MinIO: Giao diện đăng nhập	77
Hình 62. Thông tin các bucket trên MinIO	78
Hình 63. Cấu hình tạo access key cho MinIO	78
Hình 64. Harbor: Giao diện đăng nhập.....	79
Hình 65. Thông tin các project trên Harbor	79
Hình 66. Project production trên Harbor.....	80
Hình 67. Project staging trên Harbor	80
Hình 68. Thông tin các image đã được đánh tag trong một repository	80
Hình 69. Snyk Dashboard	81
Hình 70. Tạo Snyk API Token	82
Hình 71. Xác định loại dự án	82
Hình 72. Phân biệt để scan nhiều loại dự án khác nhau	83
Hình 73. Tích hợp SonarQube	84
Hình 74. Cấu hình scan SonarQube.....	84
Hình 75. Tích hợp Trivy trong Harbor	85
Hình 76. Giao diện các biểu đồ sau khi được scan bởi Trivy	85
Hình 77. Cấu hình lên lịch scan toàn bộ image	86
Hình 78. Thông tin các lỗ hổng được scan bởi Trivy	86
Hình 79. Cấu hình webhook trên Harbor	86
Hình 80. Chọn sự kiện sẽ trigger webhook	87
Hình 81. Cấu hình Bash Script cho Postee	87
Hình 82. Nội dung message được gửi đến Telegram chứa thông tin scan	88
Hình 83. Nội dung của file .csv chứa thông tin chi tiết kết quả scan bởi Trivy	
	88

Hình 84. Quản lý thông tin lưu trữ trên Vault.....	89
Hình 85. Các biến secret được lưu trữ trên Vault	89
Hình 86. Vault agent inject secret vào các pod thông qua annotations.....	90
Hình 87. Prometheus Helm repository	90
Hình 88. HelmRelease để triển khai Prometheus.....	91
Hình 89. Ingress cho Prometheus.....	91
Hình 90. Kiểm tra thông tin các resource của Prometheus	92
Hình 91. Kiểm tra kết nối đến các Node Exporter	92
Hình 92. Loki Helm repository.....	93
Hình 93. HelmRelease để triển khai Loki.....	93
Hình 94. HelmRelease để triển khai Promtail.....	93
Hình 95. Cấu hình value cho Loki	94
Hình 96. Cấu hình value cho Promtail	95
Hình 97. Ingress cho Loki	95
Hình 98. Kiểm tra thông tin các resource của Loki và Promtail	96
Hình 99. Kiểm tra trạng thái sẵn sàng của Loki.....	96
Hình 100. Bật cấu hình Grafana.....	96
Hình 101. Ingress cho Grafana	97
Hình 102. Truy cập vào Grafana web UI	97
Hình 103. Grafana kết nối đến các data source	98
Hình 104. Dashboard cho Prometheus	98
Hình 105. Dashboard cho Loki.....	99
Hình 106. Cấu hình alert cho Grafana	99
Hình 107. Nội dung alert message từ Grafana	100
Hình 108. Quy trình CI	100
Hình 109. Quản lý secrets của Github Actions	100
Hình 110. Cấu hình workflow call	101
Hình 111. Gọi lại workflow template	101
Hình 112. Đặt tên tag cho các image.....	102

Hình 113.	Luồng CI của GitHub Actions	102
Hình 114.	Xác định loại dự án (determine-project-type)	103
Hình 115.	Quét lỗ hổng bảo mật bằng Snyk	103
Hình 116.	Xem kết quả trên giao diện Snyk sau khi scan.....	104
Hình 117.	Quét chất lượng mã nguồn bằng SonarQube	104
Hình 118.	Xem kết quả trên giao diện SonarQube sau khi scan.....	105
Hình 119.	Build và push image lên Harbor.....	105
Hình 120.	Thông tin image sau khi được push lên Harbor	106
Hình 121.	Image được scan bởi Trivy sau khi được push lên.....	106
Hình 122.	Thông báo kết quả scan bởi Trivy được gửi đến Telegram.....	106
Hình 123.	Cấu trúc thư mục manifests/base	107
Hình 124.	Cấu trúc thư mục overlays.....	108
Hình 125.	Cách áp dụng patch từ các file.....	108
Hình 126.	Luồng CD hoạt động với FluxCD	109
Hình 127.	FluxCD: Định nghĩa ImageRepository	109
Hình 128.	FluxCD: Định nghĩa ImagePolicy.....	110
Hình 129.	FluxCD: Định nghĩa ImageUpdateAutomation	110
Hình 130.	FluxCD push commit thay đổi image tag trên repository	111
Hình 131.	Image tag được cập nhật lại dựa vào policy	112
Hình 132.	Kiểm tra trạng thái ứng dụng trên môi trường staging và production	112
Hình 133.	Kiểm tra thông tin Ingress của môi trường staging và production	112
Hình 134.	Truy cập vào ứng dụng với endpoint tương ứng của từng môi trường	113

DANH MỤC BẢNG

Bảng 1.	Sự khác biệt giữa DevSecOps và DevOps.....	21
Bảng 2.	Bảng phân chia nhiệm vụ	118

LỜI CẢM ƠN

Nhóm chúng em xin gửi lời cảm ơn sâu sắc và chân thành đến **thầy Lê Anh Tuấn**. Trong suốt quá trình thực hiện đồ án, thầy đã tận tình hướng dẫn, hỗ trợ chúng em với sự tận tâm và nhiệt huyết.

Những kiến thức từ môn học **Công nghệ DevOps và Ứng dụng**, cùng với sự định hướng sâu sắc của thầy, đã giúp chúng em xây dựng nền tảng kiến thức vững chắc và phát triển kỹ năng áp dụng vào thực tiễn. Sự chỉ dẫn của thầy không chỉ dừng lại ở chuyên môn mà còn khích lệ chúng em đổi mới và vượt qua những thử thách trong quá trình thực hiện dự án.

Chúng em thật sự trân trọng và biết ơn những đóng góp quý báu của thầy trong suốt hành trình này. Chúng em hy vọng có thể tiếp tục phát triển những kiến thức và kỹ năng mà thầy đã truyền đạt, góp phần cho sự thành công trong công việc tương lai.

Trân trọng,

Nhóm 3

Chương 1. GIỚI THIỆU

1.1. Lý do chọn đề tài

Trong kỷ nguyên số hóa hiện nay, các doanh nghiệp đang phải đổi mới với áp lực ngày càng tăng trong việc phát triển và triển khai phần mềm nhanh chóng, đáng tin cậy và an toàn. Việc áp dụng phương pháp DevOps truyền thống với các quy trình thủ công không còn đáp ứng được yêu cầu về tốc độ và chất lượng. Đặc biệt, các thách thức về bảo mật, quản lý cấu hình và giám sát hệ thống đang trở nên phức tạp hơn trong môi trường đám mây.

DevOps đã trở thành một phương pháp tiếp cận phổ biến, tuy nhiên, việc tích hợp bảo mật ngay từ đầu trong chu trình phát triển phần mềm - hay còn gọi là DevSecOps - ngày càng được chú trọng. Điều này đặc biệt quan trọng với hệ thống CI/CD trong môi trường microservices, vốn yêu cầu độ sẵn sàng cao và khả năng bảo mật mạnh mẽ. Mô hình DevSecOps ra đời nhằm tích hợp bảo mật vào quy trình DevOps, biến bảo mật trở thành trách nhiệm chung từ giai đoạn phát triển đến triển khai.

Đề tài tập trung áp dụng DevSecOps để xây dựng hệ thống CI/CD có độ sẵn sàng cao, tích hợp bảo mật xuyên suốt quy trình phát triển và triển khai nhằm đảm bảo hiệu quả vận hành và an toàn thông tin. Đề tài được chọn xuất phát từ nhu cầu cấp thiết trong việc xây dựng một giải pháp DevOps tự động hóa toàn diện, giải quyết các vấn đề then chốt như:

- Tối ưu hóa quy trình phát triển và triển khai (CI/CD) để giảm thiểu thời gian đưa sản phẩm ra thị trường.
- Nâng cao chất lượng mã nguồn thông qua kiểm tra tự động.
- Đảm bảo tính bảo mật trong quá trình phát triển và vận hành.
- Đảm bảo tính sẵn sàng cho hệ thống.
- Tự động hóa quy trình giám sát và cảnh báo sớm các vấn đề hệ thống.
- Ứng dụng được các công cụ DevOps phổ biến và mạnh mẽ để tối ưu khả năng hoạt động của hệ thống.
- Quản lý hiệu quả tài nguyên đám mây và chi phí vận hành.

1.2. Phạm vi nghiên cứu

Đề tài tập trung nghiên cứu và triển khai các nội dung sau:

- Tích hợp bảo mật trong DevOps: Áp dụng các nguyên tắc DevSecOps để đưa bảo mật vào mọi giai đoạn của chu trình phát triển phần mềm.
- Xây dựng hệ thống CI/CD: Thiết kế và triển khai quy trình tích hợp và triển khai liên tục với các công cụ hiện đại, đảm bảo tự động hóa và khả năng phản hồi nhanh.
- Môi trường microservices: Tập trung vào việc triển khai các giải pháp bảo mật và tự động hóa trong môi trường microservices, nơi mỗi dịch vụ hoạt động độc lập nhưng phối hợp chặt chẽ với nhau.
- Công cụ và công nghệ bảo mật: Lựa chọn và triển khai các công cụ DevSecOps như SAST, SCA, DAST, IAST, và các công cụ bảo mật nguồn mở, đồng thời tích hợp chúng vào hệ thống CI/CD.
- Độ sẵn sàng cao: Nghiên cứu và áp dụng các giải pháp tăng cường độ sẵn sàng, đảm bảo hệ thống hoạt động liên tục, ổn định ngay cả trong điều kiện tải cao hoặc có sự cố bất ngờ.
- Đánh giá và tối ưu hóa: Thực hiện thử nghiệm, đánh giá hiệu quả của mô hình về bảo mật, hiệu suất và khả năng mở rộng, từ đó đề xuất các cải tiến phù hợp.

1.3. Tóm tắt đồ án

Đồ án xây dựng một hệ thống DevOps tự động hóa toàn diện với ba luồng chính:

- Luồng quản lý cấu hình và bảo mật:
 - Kỹ sư DevOps commit infrastructure code lên GitHub.
 - Checkov thực hiện security scanning trên cấu hình.
 - HashiCorp Vault quản lý và phân phối các thông tin nhạy cảm.
 - Các policy được áp dụng để đảm bảo compliance và security.
- Luồng phát triển và đảm bảo chất lượng:
 - Quy trình bắt đầu từ việc developer commit code lên GitHub repository.

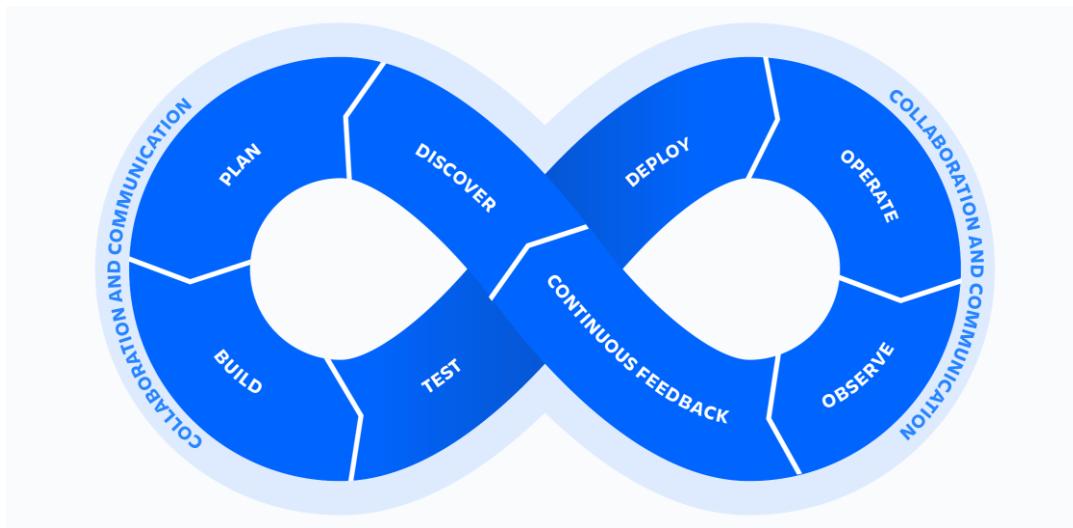
- Pipeline CI tự động kích hoạt.
 - Snyk thực hiện quét và phân tích lỗ hổng bảo mật trong dependencies.
 - Tự động hóa kiểm tra chất lượng mã nguồn thông qua SonarQube .
 - Kết quả phân tích được phản hồi về GitHub.
 - Image được build và đẩy lên Harbor Registry sau khi pass tất cả kiểm tra.
 - Trivy thực hiện quét bảo mật và gửi thông tin kết quả đến Telegram.
 - Flux CD controller theo dõi các thay đổi và tự động cập nhật môi trường Kubernetes.
- Luồng vận hành và giám sát:
- Ứng dụng được triển khai tự động lên Amazon EKS.
 - Prometheus thu thập metrics từ các service và node.
 - Grafana visualize dữ liệu và metric theo thời gian thực.
 - Grafana Loki tập trung logs từ các ứng dụng.
 - Hệ thống cảnh báo qua Telegram được kích hoạt khi phát hiện bất thường.
 - MinIO lưu trữ lịch sử logs phục vụ phân tích sau này.

Kết quả của đồ án là một hệ thống DevSecOps tự động hóa end-to-end, giúp tối ưu hóa quy trình phát triển, triển khai và vận hành, đồng thời đảm bảo tính bảo mật, khả năng mở rộng và dễ dàng quản lý. Hệ thống này có thể được áp dụng cho các dự án phát triển phần mềm có quy mô từ nhỏ đến lớn, đáp ứng các yêu cầu về tốc độ, chất lượng và bảo mật trong môi trường phát triển hiện đại.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về DevOps

2.1.1. DevOps là gì?



Hình 1. DevOps

DevOps là sự kết hợp của hai từ "Development" (Phát triển) và "Operations" (Vận hành), một phương pháp làm việc nhằm tăng cường sự hợp tác giữa các nhóm phát triển phần mềm và nhóm vận hành hệ thống [1]. DevOps hướng tới việc:

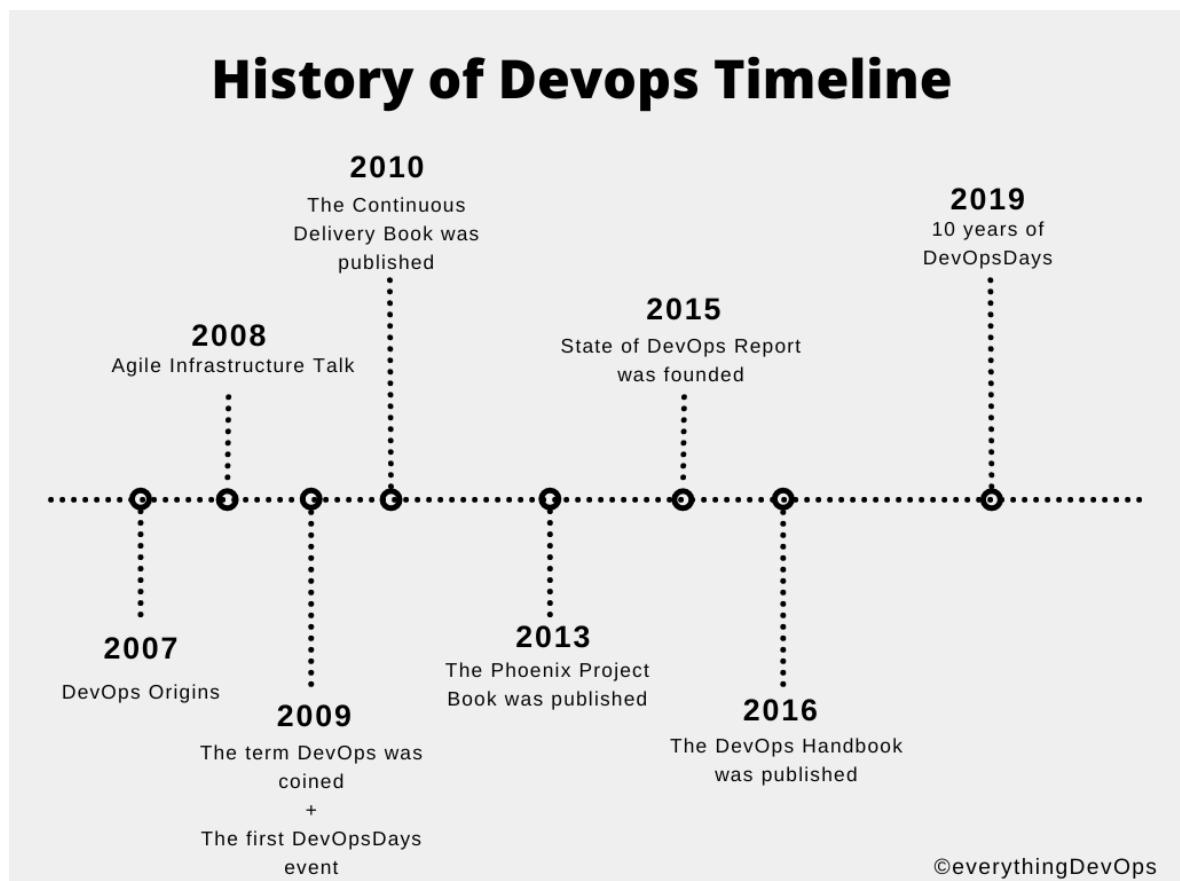
- Tự động hóa và tối ưu hóa quy trình phát triển, kiểm thử, triển khai và giám sát phần mềm.
- Cải thiện hiệu suất bằng cách giảm thời gian chuyển giao sản phẩm từ giai đoạn phát triển sang triển khai thực tế.
- Tăng chất lượng bằng việc tích hợp liên tục (CI) và triển khai liên tục (CD).

DevOps không chỉ là công nghệ mà còn là một văn hóa và triết lý làm việc.

2.1.2. Lịch sử phát triển của DevOps

Khi phương pháp phát triển Agile ngày càng phổ biến, một thuật ngữ mới đã xuất hiện để mô tả các hoạt động và nguyên tắc được sử dụng để hợp lý hóa quy trình phát triển và triển khai: **DevOps**.

Để chia sẻ về lịch sử của DevOps, dưới đây là lịch sử chi tiết theo dòng thời gian của DevOps, liệt kê tất cả các thời điểm quan trọng và tóm tắt những đóng góp của những người có ảnh hưởng chính từ năm 2007 đến năm 2019.



Hình 2. Lịch sử phát triển của DevOps

2007: Patrick Debois nhận ra vấn đề về sự không đồng bộ giữa team phát triển (Dev) và vận hành (Ops) khi làm việc trong một dự án di chuyển trung tâm dữ liệu.

2008: Tại Hội nghị Agile, Andrew Shafer và Patrick Debois gặp nhau và thảo luận về "Cơ sở hạ tầng Agile" - đây được xem như điểm khởi đầu của DevOps.

2009:

- John Allspaw và Paul Hammond trình bày về việc hợp tác Dev và Ops tại Flickr
- Patrick Debois tổ chức hội nghị DevOpsDays đầu tiên tại Ghent, Bỉ, nơi thuật ngữ "DevOps" được ra đời

2010: Jez Humble và David Farley xuất bản sách "Continuous Delivery", giới thiệu các nguyên tắc và thực hành kỹ thuật cho việc phân phối phần mềm nhanh chóng.

2013: Gene Kim, Kevin Behr và George Spafford xuất bản "The Phoenix Project", áp dụng nguyên tắc sản xuất tinh gọn vào phát triển phần mềm.

2015: DORA (DevOps Research and Assessment) được thành lập bởi Nicole Forsgren, Gene Kim và Jez Humble, tạo ra các nghiên cứu quan trọng về DevOps.

2016: "The DevOps Handbook" được xuất bản bởi Gene Kim, Jez Humble, Patrick Debois và John Willis, cung cấp hướng dẫn thực tế về triển khai DevOps.

2019: Kỷ niệm 10 năm DevOpsDays với hơn 60 sự kiện được tổ chức tại 21 quốc gia. Bridget Kromhout tiếp quản vai trò lãnh đạo DevOpsDays từ Patrick Debois [2].

2.1.3. Mục tiêu và lợi ích của DevOps

DevOps có thể tạo ra một môi trường hợp tác nơi các lập trình viên và các nhà điều hành làm việc với nhau và cùng hướng tới mục đích chung. Một cột mốc quan trọng trong quá trình này là việc thực hiện việc tích hợp và triển khai liên tục (continuous integration and continuous delivery – CI / CD). Điều này cho phép các team đưa phần mềm ra thị trường nhanh hơn, ít lỗi hơn.

Các lợi ích quan trọng của DevOps:

- **Dự đoán:** Các bản release mới có tỷ lệ thất bại thấp hơn.
- **Bảo trì:** Phục hồi các bản release một cách dễ dàng khi có vấn đề.
- **Khôi phục:** Khôi phục các phiên bản trước khi cần thiết.
- **Chất lượng cao hơn:** Kết hợp các vấn đề về cơ sở hạ tầng giúp cải thiện chất lượng phát triển ứng dụng.
- **Thời gian:** Thời gian được giảm lên đến 50%.
- **Giảm rủi ro:** Kết hợp bảo mật vào vòng đời phần mềm giúp giảm rủi ro.
- **Tiết kiệm chi phí:** Tiết kiệm chi phí hơn các phương pháp khác.
- **Khả năng phục hồi:** Hệ thống ổn định, an toàn hơn và các thay đổi có thể kiểm tra được.

- **Chia nhỏ codebase:** DevOps dựa trên phương pháp lập trình nhanh, hỗ trợ chia nhỏ các codebase thành các phần nhỏ, dễ quản lý hơn.

2.1.4. Các nguyên tắc chính của DevOps

DevOps là một triết lý phát triển phần mềm tập trung vào sự hợp tác, giao tiếp và chia sẻ giữa các team để đảm bảo việc phát triển và triển khai phần mềm chất lượng một cách kịp thời.

Nguyên tắc đầu tiên là *phát hành tăng dần* (**Incremental Releases**), khuyến khích việc phát hành code thường xuyên theo từng phần nhỏ thay vì đợi tích tụ lâu dài. Điều này giúp phát hiện và sửa lỗi nhanh chóng, tránh tình trạng "địa ngục hợp nhất" code.

Tự động hóa (Automation) là nguyên tắc quan trọng tiếp theo, nhằm tự động hóa tối đa các quy trình từ kiểm thử code đến quản lý cơ sở hạ tầng (IaC), giúp các developer tập trung vào viết code và phát triển tính năng mới.

DevOps Pipeline bao gồm 4 giai đoạn chính: Phát triển (Develop), Xây dựng (Build), Kiểm thử (Test) và Triển khai (Deploy). Pipeline này được hỗ trợ bởi các chiến lược CI/CD (Tích hợp liên tục/Phân phối liên tục).

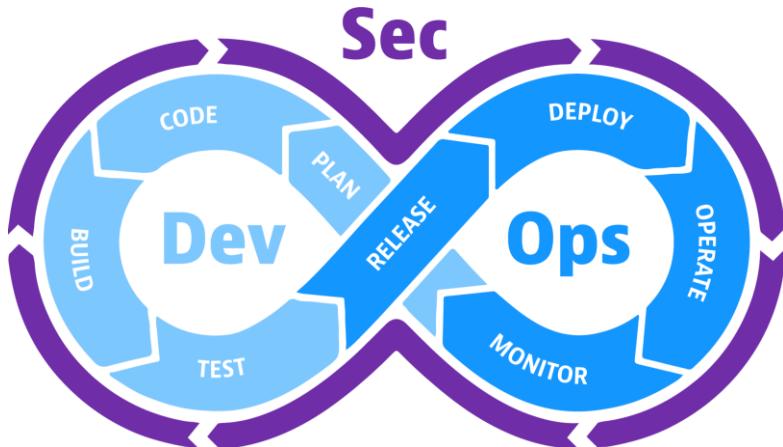
Giám sát liên tục (Continuous Monitoring) đảm bảo ứng dụng hoạt động ổn định bằng cách theo dõi logs, hiệu suất hệ thống và cơ sở hạ tầng. Khi phát hiện vấn đề, team có thể nhanh chóng khôi phục và khắc phục.

Chia sẻ phản hồi (Feedback Sharing) là yếu tố quan trọng giúp cải thiện chất lượng sản phẩm. Team DevOps thu thập phản hồi từ nhiều nguồn khác nhau như người dùng cuối, các bên liên quan và công cụ giám sát.

Kiểm soát phiên bản (Version Control) đóng vai trò trung tâm trong quy trình DevOps, giúp quản lý code một cách có tổ chức và cho phép nhiều developer làm việc song song thông qua hệ thống nhánh (branch).

Cuối cùng, *sự hợp tác (Collaboration)* giữa team Dev và Ops là nền tảng của DevOps, đòi hỏi sự tin tưởng, giao tiếp và chia sẻ trách nhiệm để cùng hướng tới mục tiêu cung cấp phần mềm chất lượng cao [3].

2.1.5. Giới thiệu về DevSecOps



Hình 3. DevSecOps

DevSecOps, viết tắt của Development Security Operations, đề cập đến khái niệm biến bảo mật phần mềm thành một phần cốt lõi của toàn bộ quy trình phát triển phần mềm. Theo truyền thống, các đánh giá bảo mật được thêm vào riêng biệt và thường là sau khi phần mềm được phát triển và tích hợp đầy đủ. Các nhà phát triển sẽ viết mã và các nhóm CNTT sẽ triển khai mà không tính đến bảo mật. Chỉ sau khi phần mềm được viết và đưa vào môi trường sản xuất, các kỹ sư bảo mật mới kiểm tra các lỗ hổng tiềm ẩn trong mã.

Cách tiếp cận này đối với bảo mật phần mềm nhanh chóng trở nên kém hiệu quả và thậm chí nguy hiểm – đặc biệt là trong môi trường đám mây, nơi tốc độ triển khai được tăng tốc đáng kể. Nếu phát hiện ra sự cố bảo mật, sẽ cần phải thực hiện nhiệm vụ tẻ nhạt là rút mã đã được viết và triển khai. Điều này cũng có nghĩa là các vấn đề nằm ngoài tầm kiểm soát và chỉ được phát hiện sau khi phần mềm đã được đưa vào sản xuất. Một quy trình như vậy khét tiếng là khiến các tổ chức phải đổi mặt với rủi ro và mối đe dọa bảo mật.

Định nghĩa DevSecOps

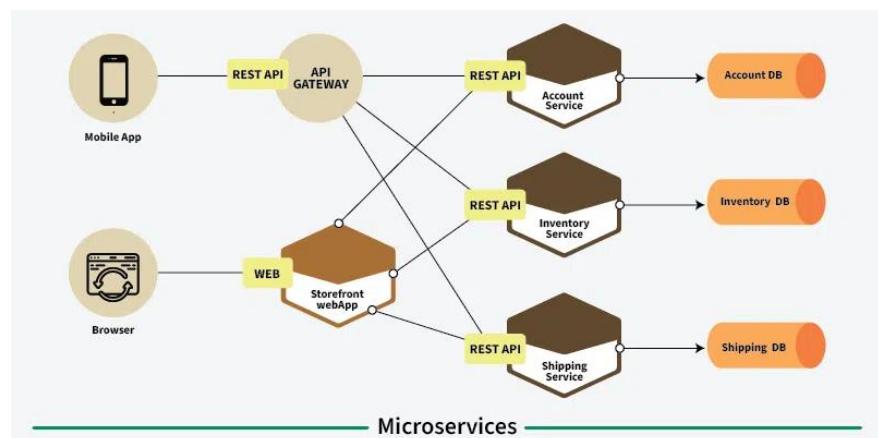
Phương pháp DevSecOps giải quyết các vấn đề bảo mật ngay lập tức bằng cách tích hợp bảo mật vào tất cả các giai đoạn phân phối phần mềm. DevSecOps cũng được gọi là bảo mật 'shift left', có nghĩa là chuyển bảo mật đến thời điểm sớm nhất có thể trong quá trình phát triển. Điều này đảm bảo rằng các nhà phát triển nghĩ về bảo mật khi họ viết mã, rằng phần mềm được kiểm tra các vấn đề bảo

mật trước khi triển khai và rằng các nhóm CNTT có kế hoạch để giải quyết các vấn đề bảo mật nhanh chóng nếu chúng xuất hiện sau khi triển khai.

Bảng 1. Sự khác biệt giữa DevSecOps và DevOps [4]

Tiêu chí	DevOps	DevSecOps
Văn hóa làm việc	Tập trung hợp tác giữa đội phát triển (Dev) và đội vận hành (Ops)	Thêm vào đó là trách nhiệm của đội bảo mật (Sec) trong toàn bộ quy trình
Xử lý bảo mật	Bảo mật thường được thực hiện ở cuối quá trình phát triển phần mềm	Tích hợp bảo mật từ sớm trong quy trình phát triển và triển khai (CI/CD)
Công cụ bảo mật	Kết hợp một số phương pháp bảo mật truyền thống	Sử dụng các công cụ bảo mật hiện đại, phù hợp với quy trình CI/CD
Hiệu quả	Có thể bị chậm trễ vì bảo mật xử lý sau, dễ tạo ra lỗi và rủi ro lớn	Giảm nguy cơ bảo mật và chi phí xử lý lỗi bằng cách bảo mật ngay từ đầu
Tự động hóa	Tự động hóa quá trình phát triển, nhưng bảo mật xử lý chủ yếu bằng tay	Tự động hóa cả bảo mật, giúp tiết kiệm thời gian và tăng hiệu quả

2.2. Kiến trúc microservices



Hình 4. Kiến trúc Microservice

Microservices là một phương pháp phát triển phần mềm, trong đó một ứng dụng lớn được chia thành các dịch vụ nhỏ, độc lập với nhau. Mỗi dịch vụ thực hiện một chức năng cụ thể và giao tiếp với các dịch vụ khác thông qua các APIs [5].

Đặc điểm của Microservices:

- Độc lập và tách biệt: Các microservices có thể được phát triển, triển khai và mở rộng độc lập mà không ảnh hưởng đến các phần khác của hệ thống.
- Định hướng theo chức năng: Mỗi microservice tập trung vào một chức năng cụ thể.
- Công nghệ đa dạng: Các microservices có thể được phát triển bằng nhiều ngôn ngữ lập trình và công nghệ khác nhau.

Kiến trúc Microservices so với kiến trúc Monolithic:

- Monolithic: Là mô hình kiến trúc truyền thống, trong đó toàn bộ ứng dụng được phát triển và triển khai như một khối duy nhất. Điều này có thể gây ra nhiều khó khăn trong việc bảo trì, mở rộng và triển khai.
- Microservices: Chia nhỏ ứng dụng thành các dịch vụ nhỏ, độc lập. Mỗi dịch vụ có thể được phát triển, triển khai và mở rộng một cách độc lập.

Lợi ích của Microservices:

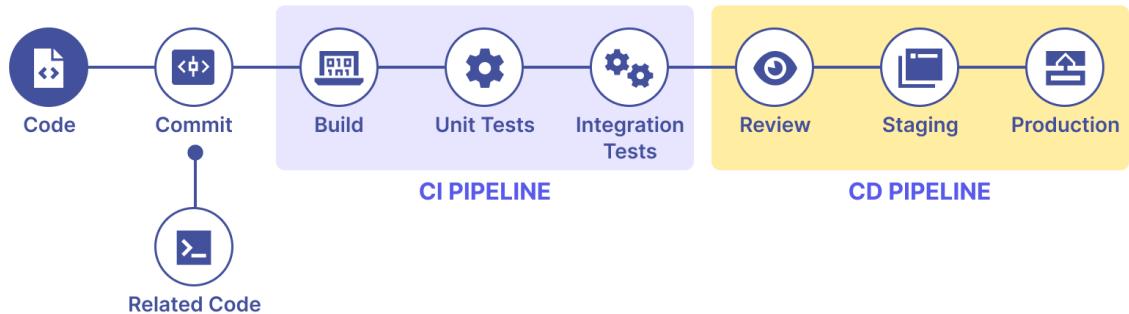
- Do các dịch vụ được tách biệt, việc bảo trì và mở rộng có thể được thực hiện dễ dàng hơn.
- Các dịch vụ có thể được triển khai độc lập, giúp dễ dàng thực hiện triển khai liên tục.
- Lỗi ở một dịch vụ không làm sụp đổ toàn bộ hệ thống, giúp hệ thống có khả năng chịu lỗi cao hơn.
- Mỗi đội phát triển có thể chọn công nghệ phù hợp nhất cho dịch vụ mà họ phát triển.

2.3. CI/CD và tự động hóa quy trình

2.3.1. Khái niệm CI/CD

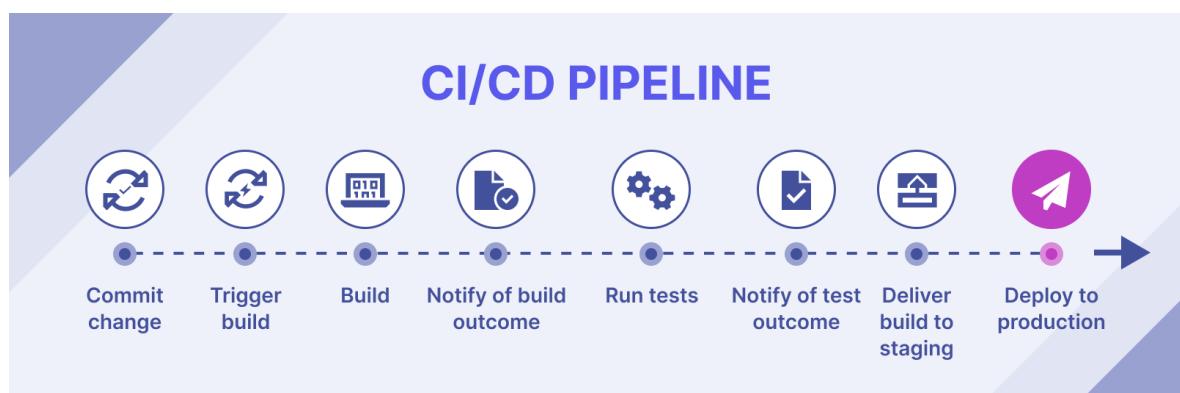
CI/CD Pipelines (hay còn được gọi là Tích hợp liên tục/Phát hành liên tục Pipelines) là một tập hợp các công cụ và quy trình tự động hóa việc xây dựng,

kiểm tra và triển khai mã. Các đường ống này thường được sử dụng trong phát triển phần mềm để đảm bảo rằng mã được xây dựng và triển khai một cách nhất quán và đáng tin cậy [6].



Hình 5. CI/CD pipeline

CI (Continuous Integration) là quá trình liên tục kiểm tra và tích hợp code mới vào code chính của một dự án. Mục tiêu của CI là đảm bảo rằng bất kỳ thay đổi nào trong code (chẳng hạn như sửa lỗi hoặc thêm tính năng) đều được kiểm tra ngay lập tức để phát hiện lỗi trước khi nó gây ảnh hưởng xấu đến hệ thống. CI hoạt động như một hệ thống kiểm tra tự động. Mỗi khi thay đổi code và commit lên hệ thống, các bài kiểm tra tự động sẽ được chạy để kiểm tra xem code có hoạt động đúng và không gây lỗi hay không.



Hình 6. CD pipeline

CD là quá trình đưa code từ nơi developer viết lên một nơi mà người dùng có thể sử dụng được, như website hoặc ứng dụng di động. Có 2 loại CD chính:

- **Continuous Delivery** (Phân phối liên tục): Đảm bảo rằng code luôn sẵn sàng để được triển khai bất cứ lúc nào. Sau khi code đã vượt qua các kiểm

tra CI, nó được đặt ở trạng thái sẵn sàng để triển khai. Tuy nhiên, việc triển khai có thể được thực hiện thủ công.

- **Continuous Deployment** (Triển khai liên tục): Tự động triển khai code ngay khi nó vượt qua tất cả các kiểm tra. Không cần sự can thiệp thủ công, hệ thống sẽ tự động đưa code vào môi trường production.

2.3.2. Vai trò của CI/CD trong DevOps

Tăng tốc độ phát triển và triển khai phần mềm: Việc tự động hóa quá trình kiểm tra và triển khai giúp các nhóm DevOps nhanh chóng đưa code vào môi trường production mà không làm gián đoạn hệ thống. Cho phép doanh nghiệp đưa các tính năng mới và bản vá lỗi ra thị trường nhanh chóng, đáp ứng nhu cầu của người dùng kịp thời.

Giảm thiểu lỗi và rủi ro trong quá trình phát triển: CI/CD giúp giảm thiểu lỗi bằng cách tự động kiểm tra code sau mỗi lần commit. Nếu có vấn đề xuất hiện, nó sẽ được phát hiện sớm và sửa chữa trước khi triển khai lên production.

Tăng cường sự hợp tác trong nhóm: Tất cả mọi người, từ developer đến tester, đều có thể thấy tiến độ của các bản build, giúp việc phối hợp tốt hơn nhằm đảm bảo phần mềm đạt chất lượng cao nhất.

Đảm bảo tính nhất quán: CI/CD giúp duy trì tính nhất quán khi phát hành phần mềm bằng cách đảm bảo rằng tất cả các bản build đều tuân thủ quy trình kiểm tra chuẩn đã được thiết lập sẵn. Điều này giúp chất lượng phần mềm luôn ổn định.

Hỗ trợ mở rộng và duy trì hệ thống phức tạp: Trong các hệ thống phần mềm lớn, việc mở rộng và duy trì hệ thống có thể trở nên rất phức tạp. CI/CD đóng vai trò quan trọng trong việc giảm tải và tự động hóa các quy trình này, đảm bảo sự ổn định và khả năng mở rộng của hệ thống.

Tối ưu hóa chi phí và tài nguyên: CI/CD giúp tự động hóa các quy trình phát triển, kiểm thử và triển khai, từ đó tiết kiệm tài nguyên, thời gian từ đó tiết kiệm chi phí nhân lực và giảm lỗi phát sinh từ thao tác thủ công.

2.4. Công cụ và công nghệ liên quan

2.4.1. GitHub (Version Control Systems)

Git là một hệ thống VCS (Version Control System) dùng để quản lý và kiểm tra các phiên bản source code khác nhau trong quá trình phát triển.

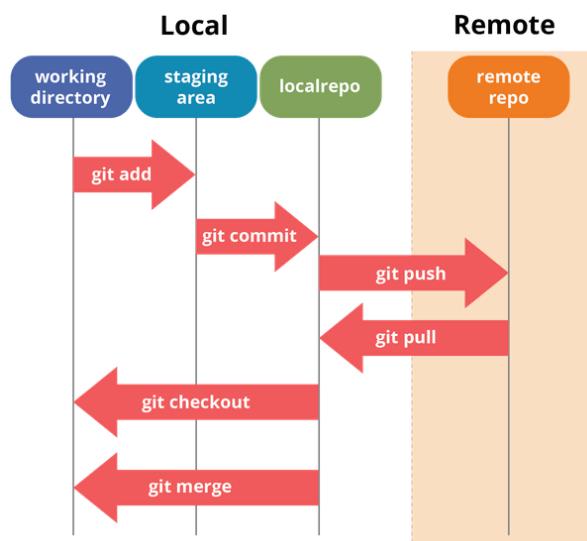
Trên Git, có thể lưu trạng thái của file khi có nhu cầu dưới dạng lịch sử cập nhật. Vì thế, có thể đưa file đã chỉnh sửa một lần về trạng thái cũ hay có thể hiển thị sự khác biệt ở nơi chỉnh sửa.

Thêm nữa, khi định ghi đè (overwrite) lên file mới nhất đã chỉnh sửa của người khác bằng file đã chỉnh sửa dựa trên file cũ, thì khi đăng (upload) lên server sẽ hiện ra cảnh cáo. Vì thế, sẽ không xảy ra thất bại về việc đã ghi đè lên nội dung chỉnh sửa của người khác mà không hề hay biết.

Mỗi tập tin trong Git được quản lý dựa trên ba trạng thái: committed, modified, và staged.

- Committed: dữ liệu đã được lưu trữ một cách an toàn trong cơ sở dữ liệu.
- Modified: đã thay đổi tập tin nhưng chưa commit vào cơ sở dữ liệu.
- Staged: đã đánh dấu sẽ commit phiên bản hiện tại của một tập tin đã chỉnh sửa trong lần commit sắp tới.

Điều này tạo ra ba phần riêng biệt của một dự án sử dụng Git: thư mục Git, thư mục làm việc, và khu vực tổ chức (staging area).



Hình 7. Luồng hoạt động của GitHub

Thư mục Git là nơi Git lưu trữ các "siêu dữ kiện" (metadata) và cơ sở dữ liệu cho dự án. Đây là phần quan trọng nhất của Git, nó là phần được sao lưu về khi người dùng tạo một bản sao (clone) của một kho chứa từ một máy tính khác.

Thư mục làm việc là bản sao một phiên bản của dự án. Những tập tin này được kéo về (pulled) từ cơ sở dữ liệu được nén lại trong thư mục Git và lưu trên ổ cứng cho người dùng sử dụng hoặc chỉnh sửa.

Khu vực khán đài là một tập tin đơn giản được chứa trong thư mục Git, nó chứa thông tin về những gì sẽ được commit trong lần commit sắp tới. Nó còn được biết đến với cái tên "chỉ mục" (index), nhưng khu vực tổ chức (staging area) đang dần được coi là tên tiêu chuẩn.

Tiến trình công việc (workflow) cơ bản của Git:

- Thay đổi các tập tin trong thư mục làm việc.
- Tổ chức các tập tin, tạo mới ảnh của các tập tin đó vào khu vực tổ chức.
- Commit, ảnh của các tập tin trong khu vực tổ chức sẽ được lưu trữ vĩnh viễn vào thư mục Git [7].

Nếu một phiên bản nào đó của một tập tin ở trong thư mục Git, nó được coi là đã commit. Nếu như nó đã được sửa và thêm vào khu vực tổ chức, nghĩa là nó đã được staged. Và nếu nó được thay đổi từ khi checkout nhưng chưa được staged, nó được coi là đã thay đổi.

2.4.2. Quản lý hạ tầng và cấu hình

2.4.2.1. Terraform



Hình 8. Terraform

Terraform là một công cụ mã nguồn mở được phát triển bởi HashiCorp, giúp tự động hóa việc quản lý cơ sở hạ tầng dưới dạng mã. Terraform cho phép các nhà phát triển và quản trị hệ thống định nghĩa hạ tầng của mình bằng các file cấu hình có thể đọc được và dễ dàng quản lý.

Đặc điểm và lợi ích của Terraform:

- Terraform có thể tương tác với nhiều nhà cung cấp điện toán đám mây khác nhau như AWS, Azure, Google Cloud, và nhiều hệ thống cơ sở hạ tầng khác.
- Các module trong Terraform cho phép tái sử dụng và chia sẻ cấu hình giữa các dự án và nhóm.
- Terraform cho phép tự động hóa các công việc như tạo, cập nhật, và xóa các tài nguyên cơ sở hạ tầng.
- Terraform ghi nhật ký tất cả các thay đổi được thực hiện và cung cấp cơ chế xác nhận trước khi triển khai thay đổi.

Một vài thành phần chính của Terraform: Providers (cung cấp API để tương tác với các dịch vụ đám mây hoặc hệ thống), Modules (tập hợp các cấu hình được tái sử dụng), Resources (các phần tử cơ bản được quản lý, như máy ảo, mạng), Data Sources (lấy thông tin từ bên ngoài để sử dụng trong cấu hình), State (tệp lưu trữ trạng thái cơ sở hạ tầng).

2.4.2.2. Ansible



Hình 9. Ansible

Ansible là một công cụ tự động hóa mã nguồn mở, được thiết kế để tự động hóa việc quản lý cấu hình, triển khai ứng dụng, và các tác vụ phức tạp khác. Ansible nổi bật nhờ tính đơn giản, không yêu cầu cài đặt các agents trên các máy đích, và khả năng mở rộng mạnh mẽ. Ansible sử dụng ngôn ngữ YAML để định nghĩa các kịch bản (playbooks) dễ đọc, giúp các quản trị viên hệ thống và nhà phát triển dễ dàng viết và duy trì các kịch bản tự động hóa.

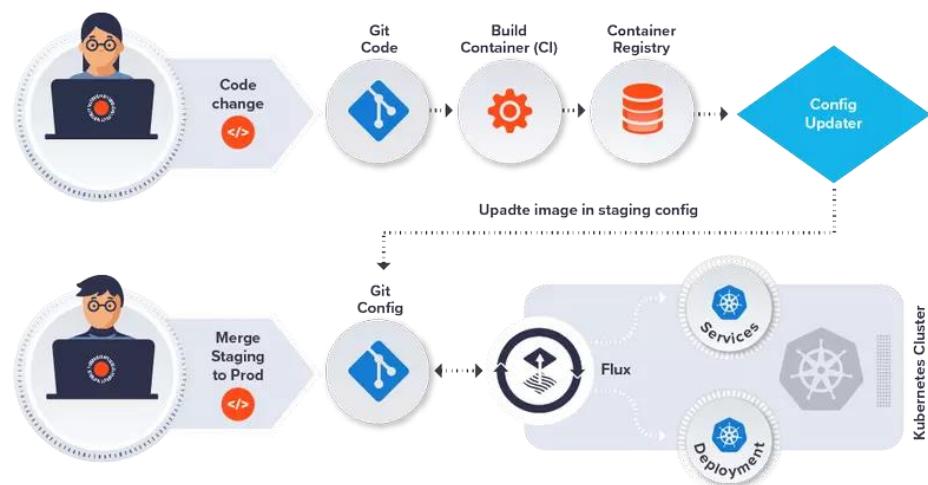
Các tính năng chính:

- Quản lý cấu hình: Ansible giúp tự động hóa việc cài đặt và cấu hình phần mềm trên các máy chủ, đảm bảo rằng tất cả các máy chủ đều có cấu hình nhất quán.
- Triển khai ứng dụng: Ansible có thể tự động hóa quá trình triển khai ứng dụng, từ việc chuẩn bị môi trường, cài đặt phần mềm, đến cấu hình và khởi động ứng dụng.
- Quản lý hạ tầng: Ansible có thể quản lý và điều phối các tài nguyên hạ tầng, chẳng hạn như tạo và cấu hình máy ảo, thiết lập mạng và lưu trữ.

2.4.3. Các công cụ cho CI/CD

2.4.3.1. FluxCD

FluxCD là một công cụ mã nguồn mở dùng để triển khai GitOps, cho phép tự động hóa quy trình triển khai ứng dụng và quản lý cấu hình hệ thống dựa trên Git. FluxCD là một phần quan trọng của hệ thống GitOps và cung cấp các tính năng mạnh mẽ để đảm bảo tính nhất quán và an toàn trong quá trình triển khai [8].



Hình 10. Quy trình hoạt động của FluxCD

FluxCD hoạt động bằng cách đọc cấu hình từ kho lưu trữ Git của và tự động áp dụng các thay đổi đó lên hệ thống hoặc cụm Kubernetes. Quá trình này diễn ra như sau:

1. Flux đọc cấu hình từ kho lưu trữ Git.
2. Flux so sánh cấu hình với tài nguyên hiện tại trong hệ thống.
3. Flux cập nhật hoặc triển khai lại các tài nguyên để đảm bảo tính nhất quán.

Một số thành phần quan trọng trong FluxCD:

- Kho lưu trữ Git: Nơi lưu trữ mã nguồn cấu hình và tài nguyên hệ thống.
- Flux Controller: Quá trình quản lý triển khai và quản lý tài nguyên hệ thống dựa trên cấu hình Git.
- Các custom resource definitions (CRDs): FluxCD sử dụng CRDs để định nghĩa cách triển khai ứng dụng và quản lý cấu hình. Chúng ta đã tìm hiểu về GitOps và FluxCD, từ định nghĩa cơ bản đến cách cài đặt và sử dụng. GitOps cung cấp một cách hiệu quả để tự động hóa quy trình triển khai và quản lý hệ thống, đồng thời đảm bảo tính nhất quán và dễ quản lý. FluxCD là một công cụ mạnh mẽ để thực hiện GitOps trong các dự án.

2.4.3.2. Github Action



Hình 11. GitHub Actions

GitHub Actions là 1 nền tảng miễn phí do GitHub cung cấp để giúp tự động hóa quá trình CI/CD, cho phép người dùng định nghĩa các workflow tự động hóa các hoạt động trong phát triển phần mềm.

Mỗi workflow trong GitHub Actions là một tập hợp các hành động (actions) được định nghĩa trong file YAML. Các actions này có thể là các lệnh cụ thể như: build ứng dụng, test, triển khai ứng dụng. Có thể sử dụng các actions được cung cấp sẵn bởi GitHub, các actions mà cộng đồng lập trình viên tạo ra hoặc tạo các hành động tùy chỉnh để phù hợp với nhu cầu của dự án của mình.

Các tính năng nổi bật của GitHub Actions:

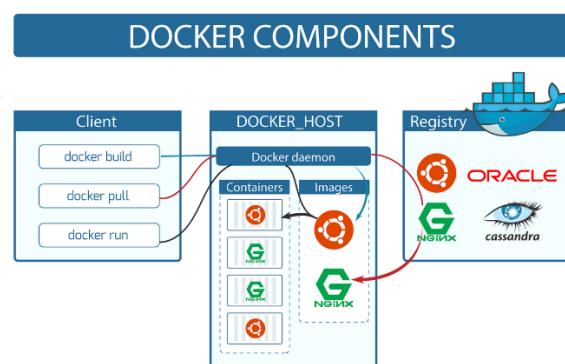
- **Tích hợp sẵn với GitHub:** Vì GitHub Actions được phát triển bởi GitHub nên nó được tích hợp sẵn với nền tảng GitHub. Điều này có nghĩa là có thể

sử dụng các tính năng của GitHub Actions trực tiếp từ trang web của GitHub, và không cần phải cài đặt hoặc cấu hình bất kỳ phần mềm nào khác.

- **Hỗ trợ chạy trên nhiều hệ điều hành:** GitHub Actions có thể chạy trên nhiều hệ điều hành khác nhau: Windows, macOS, Ubuntu... Điều này cho phép kiểm tra ứng dụng của mình trên nhiều nền tảng khác nhau và đảm bảo rằng nó hoạt động đúng cách trên mọi nền tảng.
- **Hỗ trợ chạy trên các môi trường ảo:** Cho phép thực hiện trong các môi trường ảo khác nhau, giúp chúng ta kiểm tra ứng dụng của mình trên nhiều môi trường đầy một cách dễ dàng. Ví dụ, khi phát triển ứng dụng sử dụng Nodejs, có thể định nghĩa các phiên bản Nodejs khác nhau để đảm bảo rằng ứng dụng hoạt động đúng trên các version khác nhau.
- **Tích hợp các công cụ bên thứ ba:** Có thể tích hợp các công cụ bên thứ ba vào quá trình CI/CD của mình. Ví dụ, có thể sử dụng GitHub Actions để triển khai ứng dụng của mình lên AWS hoặc Azure.
- **Truy cập các thông tin về quá trình CI/CD:** có thể truy cập các thông tin về quá trình CI/CD, bao gồm các thông tin về phiên bản mã nguồn, kết quả kiểm tra, và các lỗi phát hiện được. Điều này giúp quản lý và phát triển mã nguồn của mình một cách hiệu quả hơn.

2.4.4. Quản lý container và orchestration

2.4.4.1. Docker



Hình 12. Kiến trúc Docker

Docker là một công cụ nền tảng mã nguồn mở cho phép đóng gói, triển khai và chạy các ứng dụng trong các container. Một container là một gói phần mềm chứa tất cả những gì cần thiết để ứng dụng có thể chạy một cách nhất quán trên mọi môi trường, từ môi trường phát triển đến môi trường sản xuất [9].

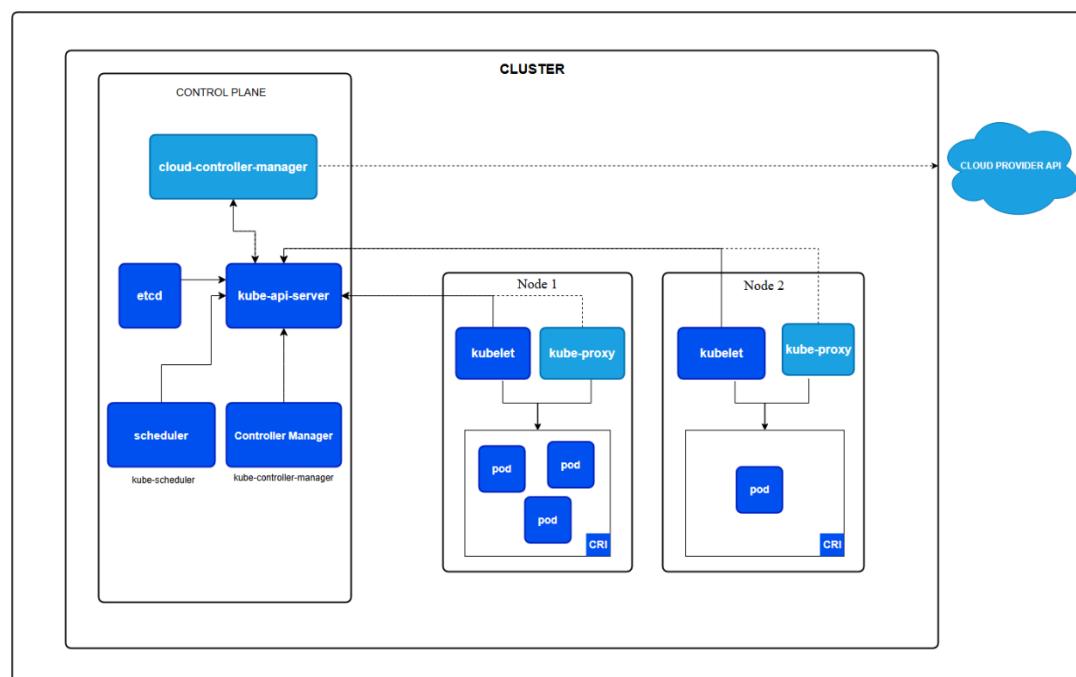
Tính năng nổi bật:

- Đóng gói ứng dụng và các phụ thuộc thành một container duy nhất, giảm thiểu các vấn đề về tương thích môi trường.
- Dễ dàng triển khai và di chuyển container giữa các hệ thống khác nhau.
- Hỗ trợ quản lý phiên bản và cấu hình, giúp việc cập nhật và duy trì ứng dụng trở nên dễ dàng hơn.

Lợi ích:

- Giảm thiểu xung đột môi trường giữa đội phát triển và vận hành.
- Tăng tốc độ triển khai nhờ việc sử dụng các image container nhẹ và nhất quán.
- Dễ dàng mở rộng quy mô hệ thống nhờ khả năng triển khai nhanh chóng các container bổ sung.

2.4.4.2. Kubernetes



Hình 13. Kiến trúc Kubernetes

Kubernetes, thường được gọi tắt là K8s, là một nền tảng mã nguồn mở hàng đầu dùng để quản lý các ứng dụng container hóa và dịch vụ liên quan. Nền tảng này được thiết kế để hỗ trợ việc triển khai, mở rộng và quản lý các ứng dụng container trên quy mô lớn một cách dễ dàng, hiệu quả. Kubernetes là giải pháp không thể thiếu trong hệ sinh thái microservices, nơi mà tính linh hoạt, độ tin cậy, và khả năng mở rộng là những yếu tố cốt lõi.

Tính năng nổi bật:

- Kubernetes tự động hóa quy trình triển khai và quản lý container, bao gồm việc phân phối tài nguyên, giám sát trạng thái container, và tự động khởi động lại các container gặp sự cố. Điều này giúp giảm thiểu công sức của các quản trị viên hệ thống.
- Declarative Configuration: Kubernetes cho phép người dùng định nghĩa trạng thái mong muốn của hệ thống thông qua tệp cấu hình YAML hoặc JSON. Công cụ này sẽ tự động điều chỉnh hệ thống để đảm bảo trạng thái thực tế luôn phù hợp với trạng thái đã khai báo.
- Orchestration: Quản lý các container trên nhiều máy chủ vật lý hoặc ảo, giúp đảm bảo tính nhất quán và hiệu quả.
- Self-healing: Phát hiện và thay thế các container lỗi hoặc ngừng hoạt động mà không cần sự can thiệp của con người.
- Scaling: Hỗ trợ mở rộng hoặc thu nhỏ số lượng container dựa trên tài nguyên hoặc nhu cầu thực tế.

Lợi ích:

- Đảm bảo tính sẵn sàng cao của ứng dụng nhờ khả năng tự động phục hồi.
- Tăng tính linh hoạt trong triển khai ứng dụng, cho phép sử dụng tài nguyên hiệu quả hơn.
- Dễ dàng tích hợp với các công cụ DevOps khác để xây dựng quy trình tự động hóa.

2.4.5. Quản lý lưu trữ

2.4.5.1. Harbor



Hình 14. Harbor

Harbor là một giải pháp mã nguồn mở dùng để quản lý và lưu trữ Docker image, hoạt động như một private registry. Harbor cung cấp tính năng bảo mật mạnh mẽ, bao gồm quét lỗ hổng, xác thực người dùng và quản lý quyền truy cập. Ngoài ra, giao diện người dùng trực quan và khả năng tích hợp với quy trình CI/CD giúp tự động hóa việc phát triển và triển khai ứng dụng. Lợi ích cho doanh nghiệp bao gồm tăng cường bảo mật hình ảnh, quản lý phiên bản dễ dàng, tối ưu hóa quy trình phát triển và khả năng phân phối hình ảnh, từ đó nâng cao hiệu quả và giảm thiểu rủi ro trong quá trình phát triển ứng dụng.

Tính năng nổi bật:

- Quản lý image: Lưu trữ và tổ chức các container image trong kho lưu trữ.
- Bảo mật: Tích hợp quét lỗ hổng bảo mật với các công cụ như Trivy.
- Hỗ trợ đa nền tảng: Làm việc với Docker và Kubernetes.
- Điều khiển truy cập (RBAC): Quản lý quyền truy cập dựa trên vai trò.

Lợi ích:

- Duy trì sự an toàn của container image trước khi triển khai.
- Giảm thiểu rủi ro bảo mật thông qua tính năng quét lỗ hổng.
- Quản lý kho lưu trữ container một cách tập trung.

2.4.5.2. MinIO



Hình 15. MinIO

MinIO là một nền tảng lưu trữ đối tượng (object storage) mã nguồn mở, được thiết kế để cung cấp hiệu suất cao, khả năng mở rộng và tương thích với giao thức S3 của Amazon Web Services (AWS). MinIO thường được sử dụng cho các ứng dụng cần lưu trữ lượng lớn dữ liệu phi cấu trúc, như dữ liệu ứng dụng, tệp đa phương tiện, sao lưu, hoặc dữ liệu phân tích.

Đặc điểm nổi bật của MinIO:

- Tương thích với giao thức S3: MinIO hoàn toàn tương thích với giao thức S3 của AWS, cho phép các ứng dụng và công cụ dựa trên S3 hoạt động trực tiếp với MinIO mà không cần thay đổi mã.
- Hiệu suất cao: MinIO được tối ưu hóa để cung cấp tốc độ truyền tải dữ liệu nhanh chóng, phù hợp cho các ứng dụng yêu cầu hiệu suất cao.
- Khả năng mở rộng: Hỗ trợ cả triển khai cục bộ và phân tán, cho phép dễ dàng mở rộng từ một máy chủ nhỏ đến một cụm lớn lưu trữ hàng petabyte dữ liệu.
- Mã nguồn mở: MinIO là một dự án mã nguồn mở, được cộng đồng và doanh nghiệp hỗ trợ mạnh mẽ.
- Tích hợp tốt: MinIO có thể tích hợp với các hệ thống như Kubernetes, Hadoop, Spark, và các công cụ DevOps khác.
- Bảo mật cao: Hỗ trợ mã hóa dữ liệu, quản lý khóa (KMS), kiểm soát truy cập (IAM), và các tiêu chuẩn bảo mật phổ biến.

2.4.6. Công cụ bảo mật

2.4.6.1. Checkov



Hình 16. Checkov

Checkov là một công cụ phân tích mã tĩnh quét các lỗ hổng bảo mật. Công cụ này ban đầu được phát triển bởi Bridgecrew.io, nhưng hiện thuộc sở hữu của Prisma Cloud. Với Checkov, có thể dễ dàng tìm thấy các lỗ hổng trước khi mã triển khai cơ sở hạ tầng. Đối với tất cả các công cụ mà Checkov hỗ trợ, công cụ này có một bộ chính sách tích hợp mà mã sẽ được kiểm tra theo, được xác định là hoặc được coi là các biện pháp thực hành tốt nhất. Các chính sách tùy chỉnh cũng có thể được viết bằng Python hoặc YAML.

Checkov hỗ trợ các công cụ IaC sau:

- **Terraform (dành cho AWS, GCP, Azure và OCI)** – các chính sách bao gồm nhiều khía cạnh khác nhau, bao gồm mã hóa, kiểm soát truy cập và các biện pháp bảo mật chung, áp dụng cho nhiều nhà cung cấp và dịch vụ đám mây
- **CloudFormation** – tập trung vào cấu hình tài nguyên dành riêng cho AWS, đảm bảo thiết lập dịch vụ, tiêu chuẩn mã hóa và kiểm soát truy cập an toàn và hiệu quả
- **Azure Resource Manager (ARM)** – nhắm mục tiêu vào các tài nguyên cụ thể của Azure, tập trung vào bảo mật, cấu hình mạng và tuân thủ các biện pháp thực hành tốt nhất của Azure
- **Helm Chart** – nhấn mạnh các giá trị mặc định an toàn, thông số kỹ thuật của vùng chứa và cấu hình quản lý tài nguyên trong định nghĩa biểu đồ Helm.
- **Kubernetes** – các chính sách bao gồm nhiều biện pháp thực hành tốt nhất, bao gồm đảm bảo các container không chạy dưới dạng root, đặt giới

hạn CPU phù hợp và nhiều khía cạnh quan trọng khác về bảo mật và hiệu suất

- **Docker** - Chính sách đảm bảo các biện pháp thực hành tốt nhất trong cấu hình Dockerfile, chẳng hạn như tránh người dùng có đặc quyền, sử dụng hình ảnh cơ sở chính thức và giảm thiểu bề mặt tấn công của vùng chứa

2.4.6.2. Snyk



Hình 17. Snyk

SNYK là các công cụ quét bảo mật/lỗ hổng thời gian/lỗ hổng thời gian 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 phụ thuộc nguồn mở trong suốt quá trình phát triển. Bảo mật ở quy mô 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 và giữ an toàn. Bảo mật tất cả các thành phần của các ứng dụng gốc đám mây hiện đại trong một công cụ quét nguồn mở SNYK duy nhất. Nền tảng bảo mật ứng dụng Cloud SNYK là mục đích được xây dựng để dễ dàng sử dụng bởi các nhà phát triển nguồn mở để phát triển bảo mật 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 phụ thuộc nguồn mở và giữ an toàn. SNYK là phần mềm bảo mật ứng dụng đám mây miễn phí cho nguồn mở. SNYK tự động tìm, sửa chữa, giám sát và ngăn ngừa các lỗ hổng trong các ứng dụng Ruby, Node.js, Java, Python và Scala. SNYK theo dõi 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 dễ bị tổn thương ứng dụng web SNYK tìm thấy rủi ro và lỗ hổng trong các ứng dụng trực tuyến của họ và các lỗ hổng mới được tiết lộ thường xuyên, khiến ứng dụng gặp rủi ro.

Yêu cầu hệ thống

Để cài đặt Công cụ CLI tiện ích SNYK nguồn mở, cần cài đặt các phụ thuộc và điều kiện tiên quyết sau: Điều kiện tiên quyết:

- Một dự án với ngôn ngữ được hỗ trợ SNYK, ví dụ như Ruby
- Dự án mã sử dụng các gói nguồn mở
- Dự án được triển khai trên hệ thống quản lý mã nguồn được hỗ trợ, ví dụ như GitHub
- Tạo một tài khoản SNYK bằng cách truy cập trang web SNYK

Đặc trưng

Một số danh sách tuyệt vời các tính năng của SNYK CLI và công cụ thời gian xây dựng để tìm và khắc phục các lỗ hổng đã biết trong các phụ thuộc nguồn mở là:

- Tìm các lỗ hổng bằng cách chạy thử nghiệm SNYK trên một dự án trong quy trình CI.
- Khắc phục lỗ hổng bằng cách sử dụng Wizard Snyk và SNYK Protect.
- Snyk Wizard tìm kiếm và sửa chữa các lỗ hổng đã biết trong một dự án.
- Cảnh báo SNYK Monitor ghi lại trạng thái phụ thuộc và bất kỳ lỗ hổng nào trên SNYK.
- Ngăn chặn các phụ thuộc mới dễ bị tổn thương bằng cách chạy thử nghiệm SNYK trong quy trình CI.

2.4.6.3. SonarQube



Hình 18. SonarQube

Được phát triển bởi 10 năm trước bởi SonarSource, SonarQube là một platform mã nguồn mở giúp nhà phát triển có thể kiểm tra chất lượng code của dự án, được viết bằng java nhưng nó hỗ trợ nhiều ngôn ngữ khác nhau: PHP, Ruby, Java (bao gồm cả Android), C#, JavaScript, TypeScript, C/C++, Kotlin, Go, COBOL, PL/SQL, PL/I, ABAP, VB.NET, VB6, Python, RPG, Flex, Objective-C, Swift, CSS, HTML, và XML và hỗ trợ các database để lưu trữ kết quả: MySql, Postgresql.

Nguồn gốc hình thành SonarQube:

Tài liệu về SonarQube ban đầu được phát triển bởi Sonarsource - Một công ty chuyên về công nghệ phần mềm và giám sát chất lượng mã nguồn. Nền tảng này đã thay đổi qua nhiều phiên bản cải tiến kể từ khi ra mắt và trở thành một trong những công cụ giám sát mã nguồn hàng đầu trên thị trường.

Đội ngũ tài năng tại công ty Sonarsource liên tục nghiên cứu và cung cấp cho cộng đồng phát triển phần mềm một công cụ mạnh mẽ. Người dùng có thể khai thác rất nhiều tính năng hiệu quả từ quá trình nghiên cứu SonarQube để quản lý chất lượng mã nguồn.

Lợi ích nhận được khi sử dụng SonarQube

Sử dụng SonarQube mang lại nhiều lợi ích quan trọng đối với quá trình phát triển phần mềm. Đầu tiên, công cụ giúp người dùng xác định và loại bỏ lỗi từ mã nguồn. Từ đó cải thiện tính năng và độ ổn định của ứng dụng.

Chương trình SonarQube cũng phân tích mã nguồn để phát hiện các điểm yếu, đưa ra báo cáo chi tiết để việc nâng cao bảo mật. Bên cạnh đó, việc theo dõi và đánh giá chất lượng mã nguồn thường xuyên giúp tối ưu hóa quy trình phát triển chương trình. Đây là cách giúp nhà phát triển phần mềm giảm thiểu công việc sửa lỗi ở giai đoạn sau.

Với SonarQube, nhóm phát triển có thể tiết kiệm thời gian, tối ưu hóa mã nguồn và tạo ra sản phẩm phần mềm chất lượng cao hơn. Những tiện ích mà SonarQube có thể thực hiện tốt hơn code convention chính là:

- Phát hiện bug
- Phát hiện code smell, duplicate
- Tính toán technical debt
- So sánh chất lượng code so với các lần kiểm tra trước
- Tính toán độ bao phủ của Unit test (Unit-test coverage)

2.4.6.4. Trivy



Hình 19. Trivy

Trivy là một trình quét lỗ hổng mã nguồn mở, đơn giản và toàn diện cho các container và các hiện vật khác. Trivy được phát triển vào năm 2019 bởi Aqua Security . Nó phát hiện các lỗ hổng của các gói hệ điều hành và cả các phụ thuộc của ứng dụng. Trước khi đẩy vào số đăng ký container hoặc triển khai ứng dụng, có thể dễ dàng quét hình ảnh container cục bộ và các hiện vật khác. Do đó, điều này giúp tự tin rằng mọi thứ đều ổn với ứng dụng của mình mà không cần cấu hình cẩn thận hơn để sử dụng như các trình quét khác.

Các tính năng của Trivy Scanner: Trivy có những tính năng sau:

- Cài đặt dễ dàng – apt, yum, apk, Bundler, Composer, pipenv, Poetry, v.v.
- Độ chính xác cao.
- Phát hiện lỗ hổng toàn diện.
- DevSecOps – Phù hợp với CI như Jenkins, GitHub Actions, Travis CI, GitLab CI, v.v.
- Hỗ trợ nhiều định dạng – Bao gồm container image, hệ thống tệp cục bộ, kho lưu trữ git từ xa.

2.4.6.5. Vault



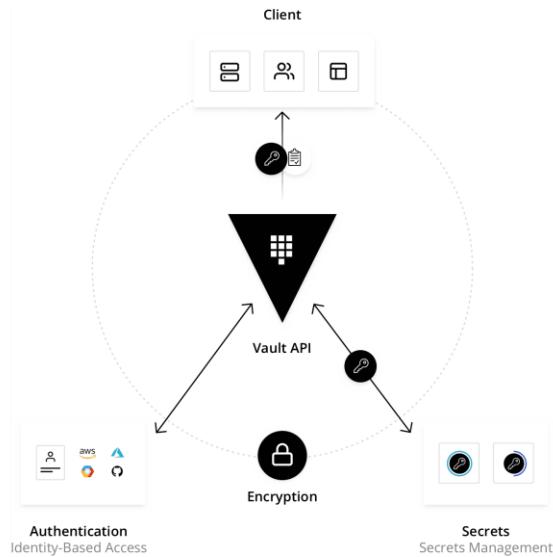
Hình 20. Vault

HashiCorp Vault là hệ thống quản lý bí mật và mã hóa dựa trên danh tính. Hệ thống này cung cấp các dịch vụ mã hóa được kiểm soát bằng các phương pháp xác thực và ủy quyền để đảm bảo quyền truy cập an toàn, có thể kiểm tra và hạn chế vào bí mật .

Bí mật là bất kỳ thứ gì muốn kiểm soát chặt chẽ quyền truy cập, chẳng hạn như mã thông báo, khóa API, mật khẩu, khóa mã hóa hoặc chứng chỉ. Vault cung cấp giao diện thống nhất cho bất kỳ bí mật nào, đồng thời cung cấp quyền kiểm soát truy cập chặt chẽ và ghi lại nhật ký kiểm tra chi tiết.

Khóa API cho các dịch vụ bên ngoài, thông tin xác thực cho giao tiếp kiến trúc hướng dịch vụ, v.v. Có thể khó hiểu ai đang truy cập vào bí mật nào, đặc biệt là vì điều này có thể tùy thuộc vào nền tảng. Việc thêm vào việc lăn khóa, lưu trữ an toàn và nhật ký kiểm tra chi tiết gần như là không thể nếu không có giải pháp tùy chỉnh. Đây chính là lúc Vault vào cuộc.

Vault xác thực và cấp quyền cho khách hàng (người dùng, máy, ứng dụng) trước khi cấp cho họ quyền truy cập vào thông tin bí mật hoặc dữ liệu nhạy cảm được lưu trữ.



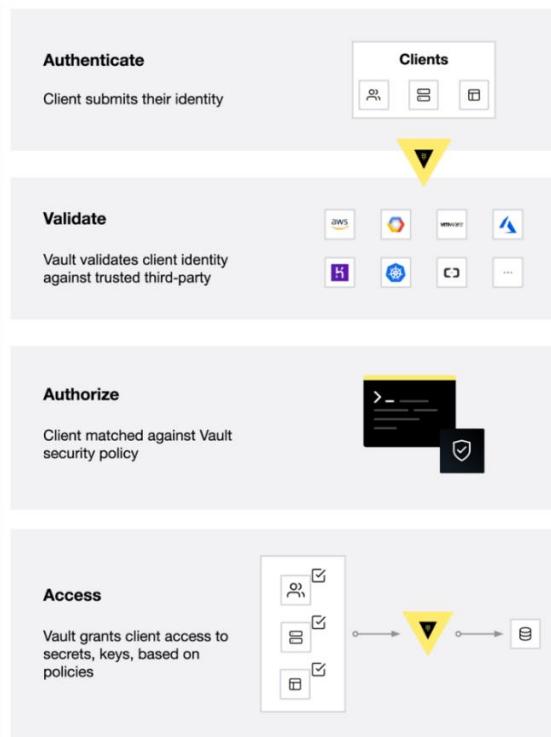
Hình 21. Mô hình của Vault

Hoạt động của Vault

Vault chủ yếu hoạt động với các mã thông báo và một mã thông báo được liên kết với chính sách của khách hàng. Mỗi chính sách dựa trên đường dẫn và các quy tắc chính sách hạn chế các hành động và khả năng truy cập vào các đường dẫn cho mỗi khách hàng. Với Vault, có thể tạo mã thông báo theo cách thủ công và chỉ định chúng cho khách hàng của mình hoặc khách hàng có thể đăng nhập và lấy mã thông báo.

Quy trình làm việc cốt lõi của Vault bao gồm bốn giai đoạn:

- **Xác thực:** Xác thực trong Vault là quá trình mà khách hàng cung cấp thông tin mà Vault sử dụng để xác định xem họ có phải là người mà họ nói hay không. Sau khi khách hàng được xác thực bằng phương pháp xác thực, một mã thông báo sẽ được tạo và liên kết với chính sách.
 - **Kiểm tra tính hợp lệ:** Vault xác thực ứng dụng khách với các nguồn đáng tin cậy của bên thứ ba, chẳng hạn như Github, LDAP, AppRole, v.v.
 - **Authorize :** Một máy khách được khớp với chính sách bảo mật Vault. Chính sách này là một tập hợp các quy tắc xác định điểm cuối API nào mà máy khách có quyền truy cập bằng mã thông báo Vault của mình. Chính sách cung cấp một cách khai báo để cấp hoặc cấm quyền truy cập vào một số đường dẫn và hoạt động nhất định trong Vault.
 - **Truy cập :** Vault cấp quyền truy cập vào các bí mật, khóa và khả năng mã hóa bằng cách phát hành mã thông báo dựa trên các chính sách liên quan đến danh tính của khách hàng. Sau đó, khách hàng có thể sử dụng mã thông báo Vault của mình cho các hoạt động tương lai [10].



Hình 22. Hoạt động của Vault

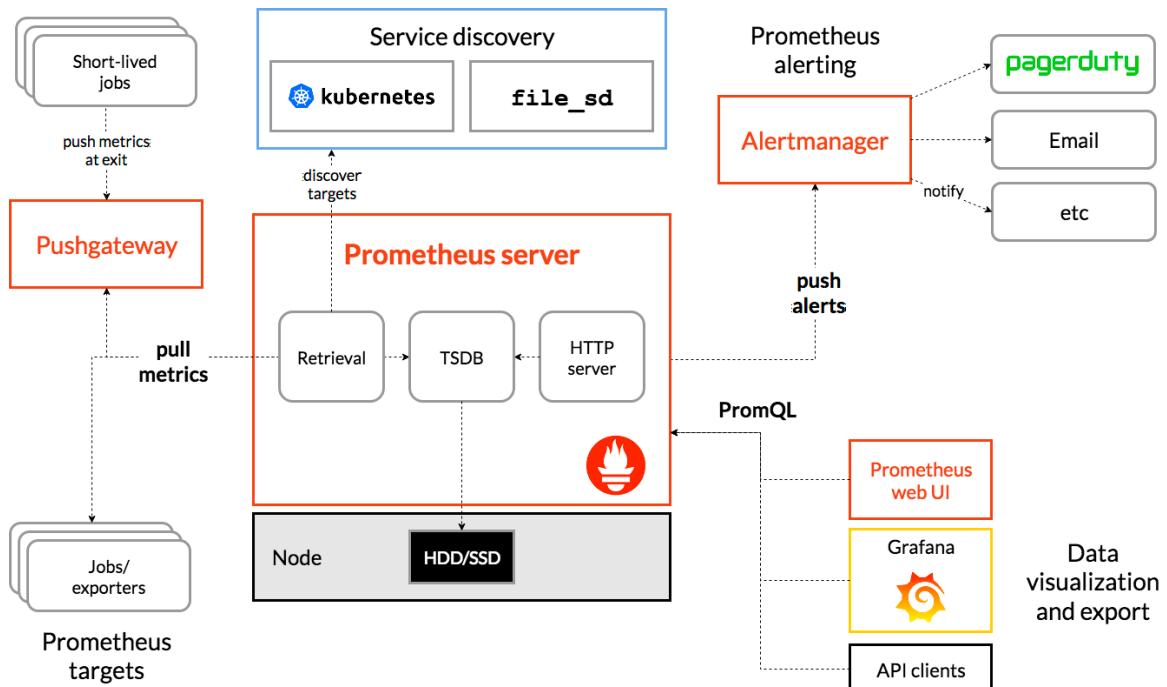
2.4.7. Công cụ giám sát

2.4.7.1. Prometheus



Hình 23. *Prometheus*

Prometheus là một hệ thống giám sát và cảnh báo mã nguồn mở được phát triển bởi SoundCloud. Prometheus giúp thu thập, lưu trữ, và phân tích các số liệu từ hệ thống ứng dụng [11].



Hình 24. *Prometheus model*

Các thành phần chính trong mô hình hoạt động của Prometheus bao gồm:

Prometheus Server:

Thành phần trung tâm có nhiệm vụ thu thập và lưu trữ dữ liệu từ các endpoint của ứng dụng, sau đó lưu trữ chúng trong cơ sở dữ liệu theo kiểu time-series. Server cũng xử lý các truy vấn từ người dùng thông qua PromQL.

- **Retrieval:**
 - Component chịu trách nhiệm pull metrics từ targets.

- Thực hiện scraping theo interval được cấu hình.
- Xử lý service discovery để biết targets cần scrape.
- TSDB (Time Series Database):
 - Lưu trữ time series data.
 - Tối ưu hóa cho dữ liệu dạng metrics theo thời gian.
 - Được lưu trên Node's HDD/SSD.
- HTTP Server:
 - Expose Prometheus API.
 - Cho phép query metrics qua PromQL.
 - Phục vụ giao diện web UI.

Exporters: Các công cụ này thu thập dữ liệu từ các ứng dụng và dịch vụ, sau đó cung cấp chúng theo định dạng mà Prometheus có thể đọc được. Có nhiều loại Exporters, chẳng hạn như Node Exporter để thu thập số liệu của hệ điều hành, hay các Exporters riêng cho MySQL, Kafka...

- Là các nguồn metrics cần được thu thập.
- Chạy như các jobs hoặc exporters để expose metrics.
- Có thể là các ứng dụng, services, infrastructure components.

Push Gateway: Được sử dụng để thu thập các số liệu từ những dịch vụ có vòng đời ngắn (short-lived jobs). Các ứng dụng gửi dữ liệu trực tiếp tới Push Gateway, sau đó Prometheus sẽ thu thập dữ liệu từ đây. Chức năng chính của Push Gateway:

- Là gateway trung gian để nhận metrics từ short-lived jobs, trong đó short-lived jobs là:
 - Các jobs chạy trong thời gian ngắn.
 - Push metrics tới Pushgateway khi kết thúc.
 - Phù hợp với các batch jobs, cron jobs.
 - Lưu trữ tạm thời metrics.
 - Cho phép Prometheus server pull metrics từ đó.

Alertmanager: Thành phần quản lý và xử lý các cảnh báo do Prometheus gửi tới. Alertmanager có thể gửi cảnh báo đến email, Telegram, Slack, PagerDuty và nhiều kênh thông báo khác.

- Xử lý và quản lý cảnh báo.
- Nhận alerts từ Prometheus server.
- Có thể gom nhóm, lọc và route alerts.

Prometheus Query Language (PromQL): Là ngôn ngữ truy vấn mạnh mẽ, cho phép người dùng trích xuất và phân tích các số liệu đã thu thập.

2.4.7.2. Loki

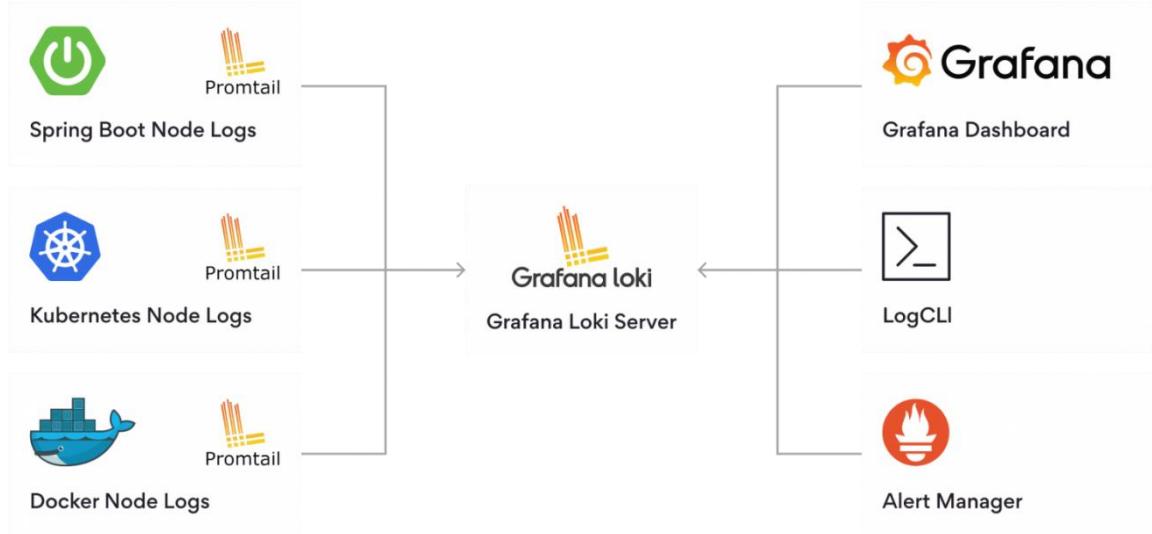


Hình 25. *Grafana Loki*

Grafana Loki là một hệ thống quản lý và tập hợp log được phát triển bởi Grafana Labs, tập trung vào việc xử lý và truy vấn log với hiệu quả cao và chi phí thấp. Loki được thiết kế để tích hợp mượt mà với Prometheus và Grafana nhằm cung cấp một giải pháp giám sát và phân tích log tối ưu cho các hệ thống phân tán và môi trường cloud-native.

Một vài đặc điểm chính của Grafana Loki:

- Là một hệ thống log aggregation được thiết kế bởi Grafana Labs.
- Lấy cảm hứng từ Prometheus, nhưng tập trung vào logs thay vì metrics.
- Được thiết kế để hiệu quả về chi phí và dễ vận hành.
- Sử dụng cùng labeling system như Prometheus giúp đồng bộ dữ liệu logs với metrics một cách dễ dàng.



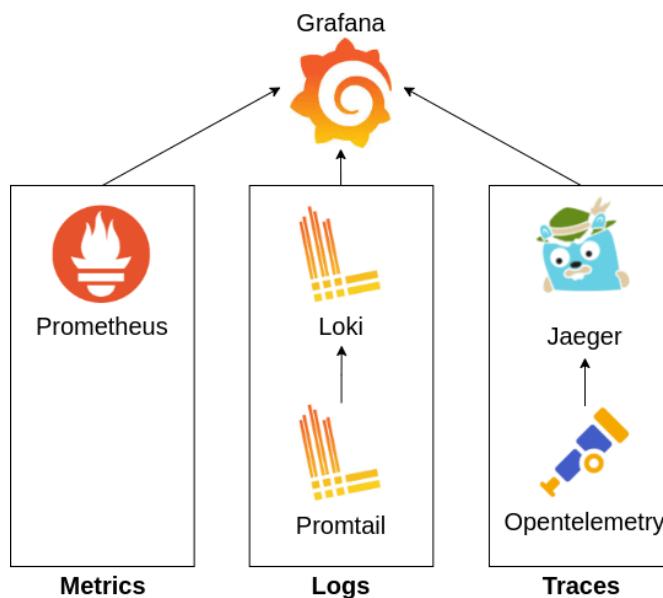
Hình 26. *Grafana Loki model*

Các thành phần chính của Grafana Loki:

- **Grafana Loki Server:** Đây là thành phần cốt lõi của hệ thống Loki với các nhiệm vụ:
 - Nhận và lưu trữ logs từ Promtail.
 - Lập chỉ mục (index) logs dựa trên nhãn.
 - Xử lý các truy vấn từ Grafana.
 - Nén và loại bỏ các logs trùng lặp.
 - Quản lý các chính sách lưu trữ logs.
- **Log Sources:**
 - App logs: Logs từ các ứng dụng chạy trên hệ thống.
 - Kubernetes Node Logs: Logs từ các nodes trong Kubernetes cluster chứa logs hệ thống, logs container, và sự kiện của Kubernetes.
 - Docker Node Logs: Logs từ Docker containers ghi lại đầu ra của container (stdout/stderr) và logs của Docker daemon.
- **Promtail:** Promtail là agent thu thập logs, được triển khai trên mỗi node cần thu thập dữ liệu. Nó có các chức năng chính như sau:
 - Khám phá các mục tiêu thu thập logs (service discovery).
 - Gắn nhãn cho các log streams.

- Đẩy logs đến Loki server để xử lý.
- Theo dõi log files và tích hợp với các dịch vụ phát hiện.
- **LogCLI:** Công cụ dòng lệnh cho phép truy vấn Loki, hữu ích cho quá trình gỡ lỗi và tự động hóa, đồng thời có thể tích hợp với các script.
- **Alert Manager:** Xử lý các cảnh báo dựa trên mẫu logs, tích hợp với các cảnh báo trong Grafana, và quản lý các thông báo qua các kênh khác nhau. Mặc dù Loki không có hệ thống cảnh báo tích hợp như Prometheus, nhưng có thể sử dụng các công cụ bổ trợ như Prometheus hoặc AlertManager để thiết lập cảnh báo dựa trên logs. Cơ chế cảnh báo này được triển khai dựa trên việc phát hiện các mẫu logs cụ thể hoặc các sự kiện đặc biệt [12].

2.4.7.3. Grafana



Hình 27. Grafana

Grafana là một công cụ trực quan hóa dữ liệu mã nguồn mở, cho phép người dùng giám sát và phân tích dữ liệu time-series từ các nguồn khác nhau, như Prometheus, Loki, Elasticsearch... Cho phép query, visualize, alert và hiểu về metrics bất kể chúng được lưu trữ ở đâu.

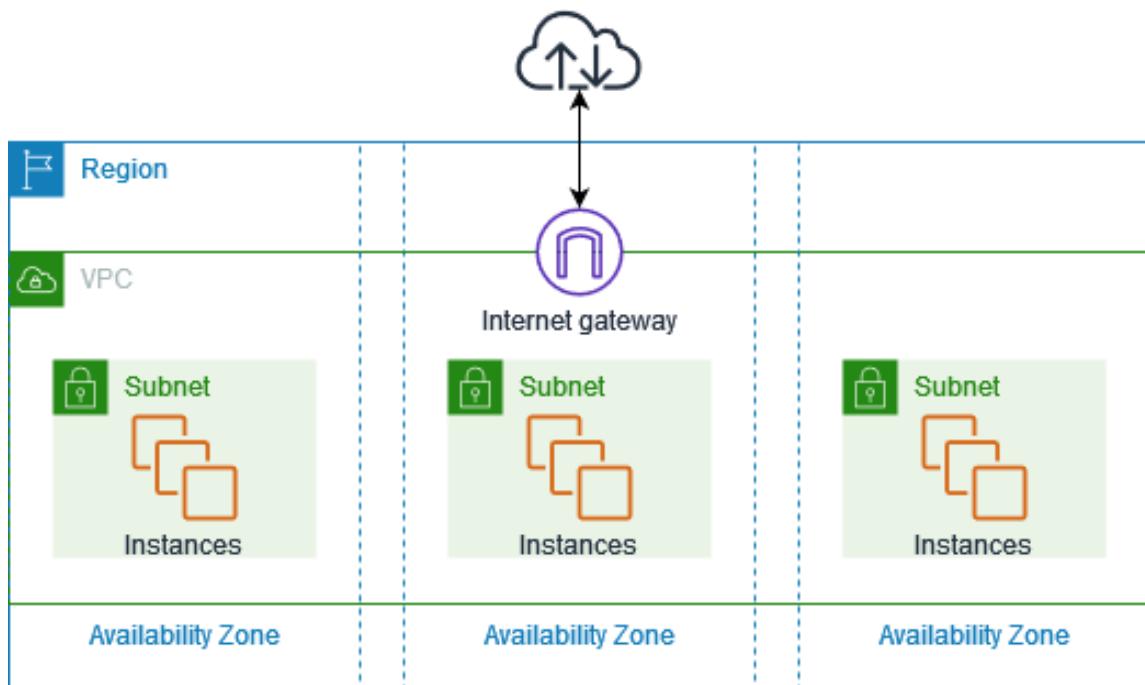
Grafana đặc biệt hữu ích trong việc hiển thị dữ liệu logs, metrics, và cung cấp một nền tảng tập trung để tạo các bảng điều khiển (dashboard) giúp giám sát hệ thống và ứng dụng.

Các thành phần chính của Grafana:

- **Connections:** Tích hợp với nhiều nguồn dữ liệu qua các plugin tích hợp sẵn hoặc từ cộng đồng để kết nối và lấy dữ liệu từ các hệ thống khác như Prometheus, Loki, Elasticsearch...
- **Dashboards:** Giao diện trực quan hóa dữ liệu.
- **Alerting:** Cho phép tạo các quy tắc cảnh báo dựa trên dữ liệu được lấy từ các nguồn tích hợp.
- **Annotations:** Là các ghi chú mà người dùng có thể thêm vào biểu đồ để đánh dấu các sự kiện quan trọng hoặc các thay đổi trong hệ thống.
- **Administration:** Quản lý thông tin xác thực người dùng và phân quyền.

2.4.8. AWS (Amazon Web Service)

2.4.8.1. Amazon VPC



Hình 28. Kiến trúc Amazon VPC

Amazon Virtual Private Cloud (VPC) là một dịch vụ cho phép bạn khởi chạy các tài nguyên AWS trong một mạng ảo logic mà bạn tự định nghĩa. Mạng ảo này tương tự như mạng truyền thống trong trung tâm dữ liệu của riêng bạn, nhưng được vận hành trên cơ sở hạ tầng linh hoạt và dễ mở rộng của AWS [13].

Đặc điểm nổi bật của Amazon VPC:

- Mạng ảo logic: VPC tạo một mạng riêng biệt cho các tài nguyên AWS của bạn, cung cấp khả năng kiểm soát toàn diện đối với môi trường mạng, bao gồm cấu hình địa chỉ IP, subnet, route table và gateway.
- Tích hợp với các dịch vụ AWS: Amazon VPC tích hợp sâu với các dịch vụ AWS khác như Amazon EC2, Elastic Load Balancing, và Amazon RDS, cung cấp một nền tảng toàn diện để xây dựng các ứng dụng và hệ thống linh hoạt.
- Cách ly tài nguyên: Các tài nguyên trong một VPC được cô lập hoàn toàn với các tài nguyên trong các VPC khác, đảm bảo bảo mật và tính riêng tư.
- Kết nối mạng linh hoạt: Bạn có thể kết nối VPC với internet thông qua Internet Gateway, kết nối với các mạng tại chỗ (on-premises) bằng VPN Gateway hoặc AWS Direct Connect, và kết nối các VPC với nhau bằng VPC Peering.

Kiến trúc VPC cơ bản:

- Subnet:
 - Mỗi VPC được chia thành các subnet, thường phân bố ở nhiều Availability Zones (AZ) để tăng tính sẵn sàng.
 - Các subnet có thể được định nghĩa là Public Subnet (có kết nối internet) hoặc Private Subnet (chỉ sử dụng nội bộ).
- Internet Gateway (IGW): Là thành phần cho phép các tài nguyên trong VPC kết nối với internet. Public Subnet thường sử dụng IGW để cung cấp truy cập internet.
- Route Tables: Định tuyến lưu lượng giữa các subnet, Internet Gateway, hoặc VPN Gateway, cho phép kiểm soát đường đi của dữ liệu.
- Elastic IP (EIP): Là địa chỉ IP cố định giúp EC2 hoặc các dịch vụ khác trong VPC có thể truy cập từ bên ngoài.

Lợi ích của Amazon VPC:

- Bảo mật cao: Tăng cường bảo mật bằng cách sử dụng Network ACL và Security Groups để kiểm soát lưu lượng vào và ra.
- Tùy chỉnh linh hoạt: Dễ dàng điều chỉnh các thành phần mạng để phù hợp với nhu cầu cụ thể của ứng dụng.
- Khả năng mở rộng: Hỗ trợ khôi phục công việc lớn với tính sẵn sàng và khả năng mở rộng cao.

2.4.8.2. Amazon EC2



Hình 29. Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) là một cơ sở hạ tầng điện toán đám mây được cung cấp bởi **Amazon Web Services (AWS)** giúp cung cấp tài nguyên máy tính ảo hoá theo yêu cầu.

Amazon EC2 cung cấp các ứng dụng máy tính ảo hoá có thể mở rộng về khả năng xử lý cùng các thành phần phần cứng ảo như bộ nhớ máy tính (ram), vi xử lý, linh hoạt trong việc lựa chọn các phân vùng lưu trữ dữ liệu ở các nền tảng khác nhau và sự an toàn trong quản lý dịch vụ bởi kiến trúc ảo hoá đám mây mạnh mẽ của AWS.

Amazon EC2 sẽ cung cấp một hoặc máy chủ ảo có thể kết hợp với nhau để dễ dàng triển khai ứng dụng nhanh nhất và đảm bảo tính sẵn sàng cao nhất. Thậm chí về mặt thanh toán, dễ dàng biết được các mức chi phí cần thanh toán dựa trên thông tin tài nguyên sử dụng.

Amazon EC2 Instance là một cloud server. Với một tài khoản có thể tạo và sử dụng nhiều Amazon EC2 Instance. Các Amazon EC2 Instance được chạy trên cùng một server vật lý và chia sẻ memory, CPU, ổ cứng...

Tuy nhiên do tính chất của cloud service nên mỗi một Instance được hoạt động giống như một server riêng lẻ.

Các đặc tính của Amazon EC2:

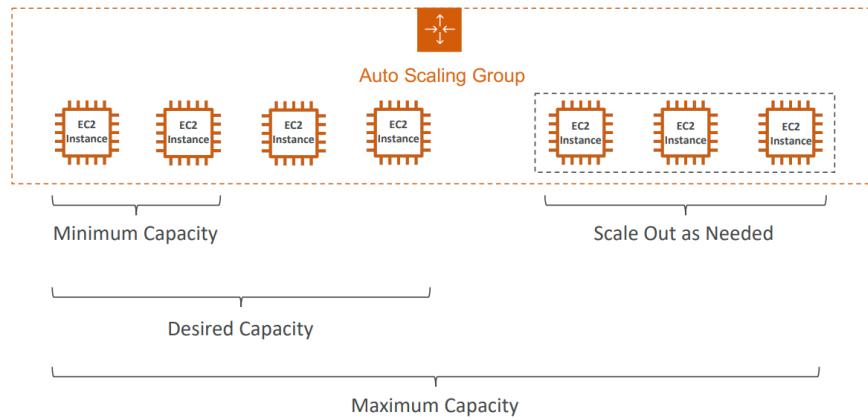
- **Scaling:**
 - Scaling Up/Down: Tăng/Giảm capacity (RAM, CPU...) của Instance.
 - Scaling In/Out: Tăng/Giảm số lượng Instance.
- **Security:**
 - Có thể thiết lập rank IP Private dành riêng cho EC2.
 - Sử dụng Security Group và Network ACLS để control inbound/outbound.
 - Dedicated Instance: Tạo EC2 trên 1 hardware physical dành riêng cho 1 khách hàng duy nhất.
- **Cost:**
 - On-Demand Instance: Tính theo giờ, đáp ứng nhu cầu dùng trong thời gian ngắn. Dùng bao nhiêu, trả bấy nhiêu.
 - Reserved Instance: Cho phép trả trước 1 lượng Server cho 1 hoặc 3 năm. Chi phí chỉ bằng 75% so với On-Demand.

2.4.8.1.1. EC2 Auto Scaling



Hình 30. EC2 Auto Scaling

Trong thực tế, tải đặt lên một ứng dụng hoặc website có thể thay đổi rất chóng vánh. Ví dụ với một website đăng ký học phần của các trường Đại học, các trang đăng ký của một website thường chịu tải gần như bằng 0 vào giai đoạn giữa hoặc cuối kì học, nhưng tải sẽ tăng đột biến trong vài ngày khi sinh viên ồ ạt vào đăng ký học. Thường thì đây là vấn đề khó giải quyết dưới on-premise (đây có lẽ cũng là lý do mà câu chuyện đăng ký học vẫn tiếp diễn ở nhiều trường từ năm này qua năm khác).



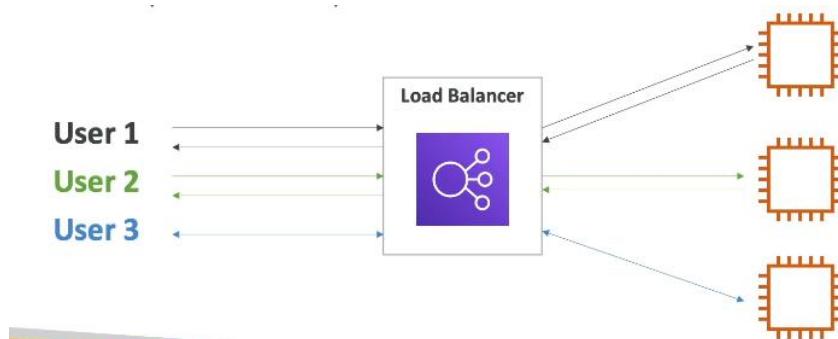
Hình 31. Mô hình Auto Scaling Group

Đối với Auto Scaling Group trên AWS, dịch vụ này được xây dựng nhằm cung cấp các tính năng chính như sau:

- Scale out/scale in (tăng/giảm) số lượng instances EC2 phù hợp với sự thay đổi của tải đặt lên hệ thống.
- Cho phép người dùng có thể kiểm soát số lượng instances tối thiểu và tối đa, nhằm tránh các lỗi không xác định gây ảnh hưởng hệ thống hoặc làm gia tăng chi phí.
- Tự động đăng ký mới các instances với các bộ cân bằng tải cũng như thay thế các Instances gặp sự cố (VD: bị terminated hoặc gặp lỗi).

Thông thường, quá trình scale instances mới sẽ không diễn ra ngay lập tức mà sẽ mất một khoảng thời gian (thường là vài phút) do cần khởi tạo các EC2 instances mới, do đó cần lưu ý để thiết kế các chiến lược scale sao cho tối ưu với nhu cầu của hệ thống.

2.4.8.1.2. Elastic Load Balancing



Hình 32. Elastic Load Balancing

Elastic Load Balancing (ELB) là dịch vụ được quản lý bởi AWS, dùng để tự động phân phối lưu lượng truy cập đến nhiều tài nguyên như Amazon EC2 instances, containers, hoặc địa chỉ IP trong một hoặc nhiều Availability Zones (AZs) trong cùng một khu vực. ELB được thiết kế nhằm tăng khả năng chịu lỗi, cải thiện hiệu năng ứng dụng và đảm bảo tính sẵn sàng cao của hệ thống.

Chức năng và lợi ích của ELB:

- Phân phối lưu lượng (Load Distribution): ELB tự động chia lưu lượng đến các tài nguyên nằm phía sau (back-end) như EC2 instances, giúp giảm tải và ngăn ngừa tình trạng quá tải trên bất kỳ tài nguyên cụ thể nào.
- Cung cấp một điểm truy cập duy nhất (Single Access Point): ELB hoạt động như một điểm truy cập duy nhất thông qua DNS, giúp đơn giản hóa việc truy cập và cải thiện trải nghiệm người dùng.
- Tích hợp kiểm tra sức khỏe (Health Checks): ELB liên tục kiểm tra trạng thái của các tài nguyên phía sau. Nếu một tài nguyên không hoạt động, ELB sẽ ngừng chuyển hướng lưu lượng đến tài nguyên đó, giúp đảm bảo tính ổn định của hệ thống.
- Hỗ trợ SSL Termination: ELB cung cấp khả năng mã hóa HTTPS, giảm tải cho các tài nguyên phía sau bằng cách thực hiện mã hóa và giải mã SSL/TLS.
- Khả năng chịu lỗi và tính sẵn sàng cao (High Availability): ELB phân phối lưu lượng qua nhiều AZs, đảm bảo hệ thống vẫn hoạt động ngay cả khi một AZ gặp sự cố.
- Tự động mở rộng (Auto Scaling): ELB tự động điều chỉnh khả năng xử lý lưu lượng tăng đột biến, hỗ trợ ứng dụng ở mọi quy mô.

Các Loại Elastic Load Balancer: AWS hiện cung cấp ba loại Load Balancer chính:

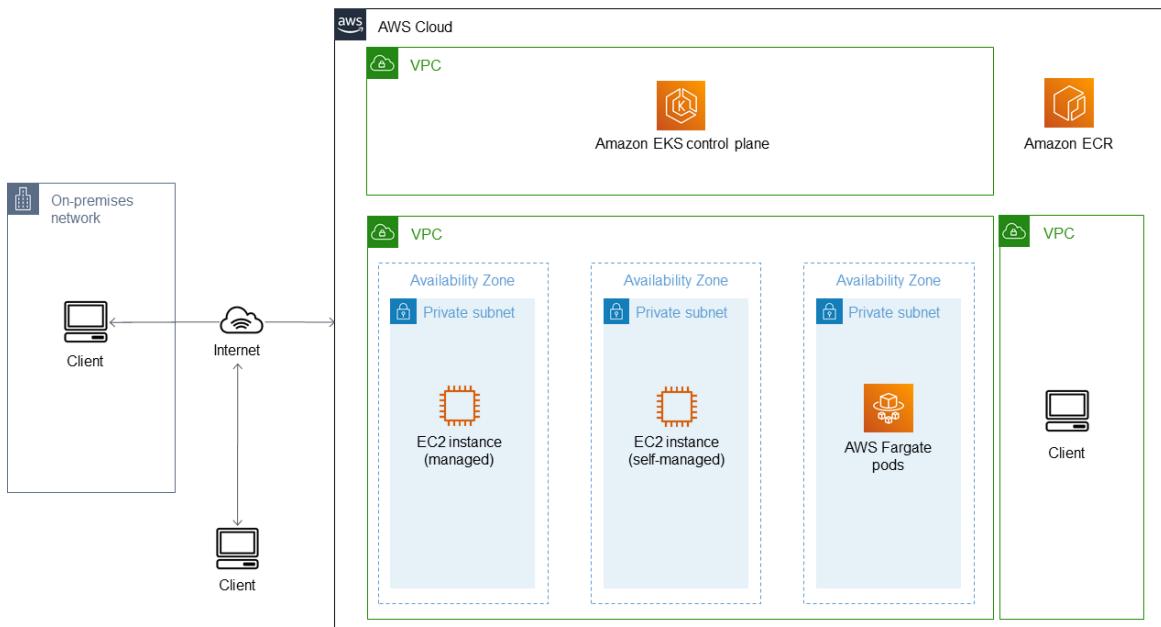
- Application Load Balancer (ALB)
 - Hoạt động ở tầng 7 (HTTP/HTTPS), thích hợp cho các ứng dụng web.
 - Hỗ trợ định tuyến dựa trên nội dung, như URL hoặc Host header.

- Network Load Balancer (NLB)
 - Hoạt động ở tầng 4 (TCP/UDP), phù hợp cho các ứng dụng yêu cầu hiệu suất cao, như trò chơi hoặc hệ thống có lưu lượng lớn.
 - Có độ trễ thấp và khả năng xử lý hàng triệu yêu cầu mỗi giây.
- Classic Load Balancer (CLB)
 - Là loại Load Balancer thế hệ cũ, hỗ trợ cả tầng 4 và tầng 7.
 - Đang dần được thay thế bởi ALB và NLB để tận dụng công nghệ mới và tính năng nâng cao.

Tại sao nên sử dụng ELB?

- Dịch vụ được quản lý: AWS đảm bảo ELB luôn sẵn sàng, xử lý bảo trì và cập nhật tự động.
- Khả năng tích hợp: ELB tích hợp chặt chẽ với các dịch vụ khác như Auto Scaling, Route 53, và CloudWatch.
- Hiệu quả chi phí: Mặc dù có thể tự triển khai Load Balancer, ELB giúp tiết kiệm công sức quản lý và đảm bảo hiệu suất ổn định [14].

2.4.8.3. Amazon EKS



Hình 33. Kiến trúc Amazon EKS

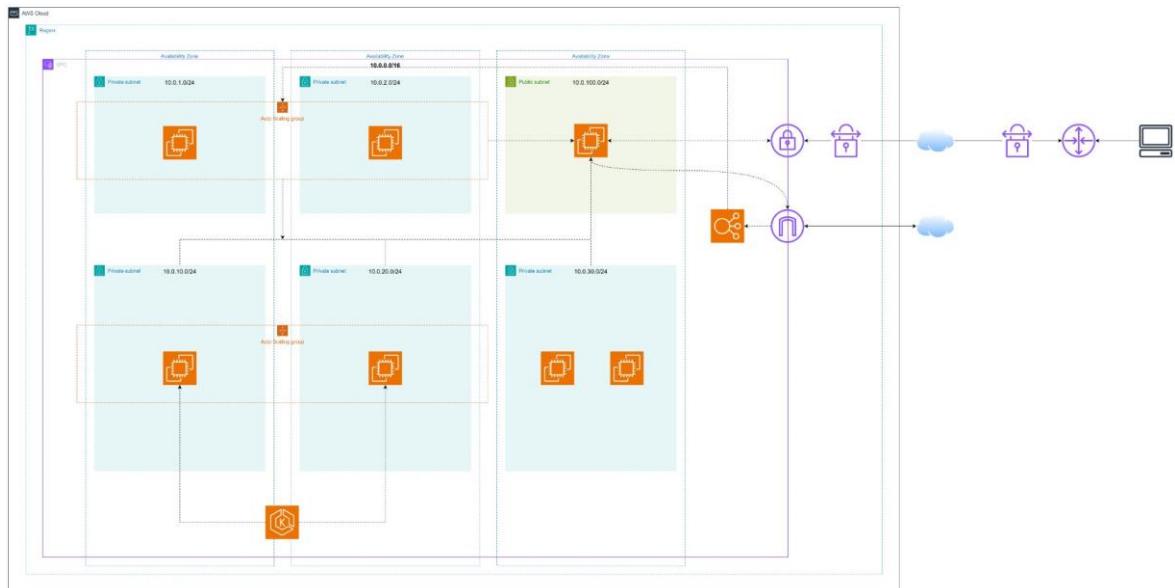
Amazon Elastic Kubernetes Service (EKS) là một dịch vụ quản lý Kubernetes được AWS cung cấp nhằm đơn giản hóa quá trình triển khai, quản lý và mở rộng các cụm Kubernetes trên AWS. Với Amazon EKS, bạn không cần tự vận hành hay duy trì cơ sở hạ tầng của Kubernetes control plane. AWS đảm nhận các nhiệm vụ đó, cung cấp một control plane hoàn toàn được quản lý, có tính khả dụng cao và an toàn [15].

Lợi ích của EKS: Có một số lợi ích khi sử dụng EKS cho các ứng dụng chứa trong container:

- Dịch vụ Kubernetes được quản lý hoàn toàn: Với EKS, AWS quản lý việc triển khai, mở rộng quy mô và vận hành cụm Kubernetes. Điều này loại bỏ chi phí vận hành quản lý cụm Kubernetes, cho phép tập trung vào việc chạy các ứng dụng của mình.
- Tính khả dụng cao và khả năng mở rộng: EKS được thiết kế để có tính khả dụng cao và có thể mở rộng. Nó chạy các phiên bản mặt phẳng điều khiển Kubernetes trên nhiều vùng khả dụng, đảm bảo khả năng phục hồi và giảm nguy cơ thời gian chết. EKS cũng cho phép mở rộng ứng dụng theo chiều ngang và chiều dọc khi cần.
- Bảo mật: EKS được thiết kế với mục đích bảo mật. Nó cung cấp một số tính năng bảo mật, bao gồm cài đặt VPC, vai trò IAM cho tài khoản dịch vụ Kubernetes và tích hợp AWS Key Management Service (KMS) để mã hóa bí mật.
- Dễ sử dụng: EKS dễ sử dụng, ngay cả khi mới làm quen với Kubernetes. Nó cung cấp giao diện đơn giản để triển khai và quản lý các ứng dụng được chứa trong container của và tích hợp liền mạch với các dịch vụ AWS khác.
- Hiệu quả về chi phí: EKS là giải pháp tiết kiệm chi phí để chạy các ứng dụng chứa trong container. Chỉ trả tiền cho các tài nguyên sử dụng và không có chi phí trả trước hoặc cam kết dài hạn.

Chương 3. PHÂN TÍCH THIẾT KẾ

3.1. Mô hình mạng triển khai



Hình 34. Mô hình mạng triển khai

Kiến trúc mạng này thể hiện một thiết kế multi-AZ (Availability Zone) trên AWS Cloud, tập trung vào high availability và security. Hệ thống được triển khai trên 3 AZ khác nhau, mỗi AZ có các private subnet riêng biệt để cách ly và bảo vệ các resources.

Network layout được thiết kế theo mô hình hub-spoke, cho phép quản lý traffic một cách hiệu quả thông qua việc sử dụng các load balancer và NAT gateway. Security được áp dụng theo nhiều lớp với security groups và network ACLs, đảm bảo chỉ những traffic được phép mới có thể đi vào hệ thống. Các resources được phân bố đều trên các AZ để đảm bảo khả năng chịu lỗi và duy trì độ sẵn sàng của dịch vụ ngay cả khi một AZ gặp sự cố.

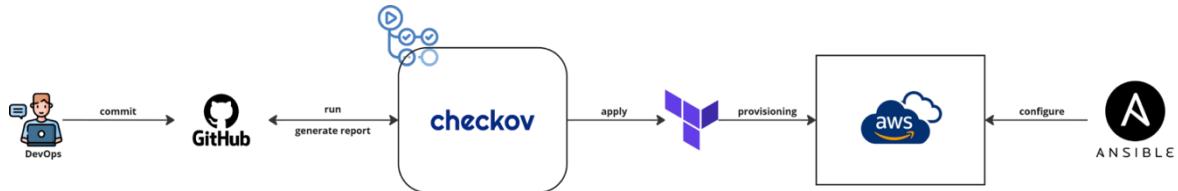
Thiết kế này cũng tạo điều kiện thuận lợi cho việc scale horizontally khi cần thiết, đồng thời duy trì được security mạnh mẽ thông qua việc cách ly network và áp dụng các cấu hình security phù hợp.

Phân tích cụ thể mô hình: Mô hình triển khai một hạ tầng AWS hoàn chỉnh với VPC được chia thành nhiều vùng mạng khác nhau (public và private subnets) nhằm đảm bảo tính bảo mật và khả năng mở rộng. Các thanh phần chính:

- VPC và Network layout:
 - Thiết lập VPC với public và private subnets.
 - Public subnets cho các dịch vụ cần kết nối internet.
 - Private subnets cho các ứng dụng và dữ liệu nội bộ.
 - Tối ưu cho high availability với multiple AZs.
- Bastion Host (Gateway):
 - Đóng vai trò như jump server.
 - Cài đặt OpenVPN server cho phép truy cập an toàn vào các tài nguyên trong private network.
 - Được đặt trong public subnet.
- Storage và Security Servers:
 - Các máy chủ phục vụ lưu trữ và bảo mật (Harbor, MinIO, SonarQube, Vault).
 - Được triển khai trong private subnets để tăng tính bảo mật.
 - Tách biệt chức năng để dễ quản lý và bảo trì.
- Load Balancing Layer:
 - Kết hợp hai loại load balancer:
 - Network Load Balancer (NLB) cho Layer 4:
 - Xử lý cả HTTP (80) và HTTPS (443) traffic.
 - Health check liên tục để đảm bảo service hoạt động.
 - Triển khai ở public subnet để nhận traffic từ internet.
 - HAProxy trong Auto Scaling Group:
 - Triển khai trên private subnet.
 - Tự động scale dựa trên CPU usage (ngưỡng 80%).
 - Được đặt sau NLB để phân phối traffic.
 - Đảm bảo high availability qua multiple AZs.
 - Tự động scale theo nhu cầu tải.
- Kubernetes Cluster (EKS):
 - Cung cấp nền tảng container orchestration.

- Triển khai trong private subnets.
- Hỗ trợ auto-scaling cho worker nodes.

3.2. Quy trình triển khai hạ tầng



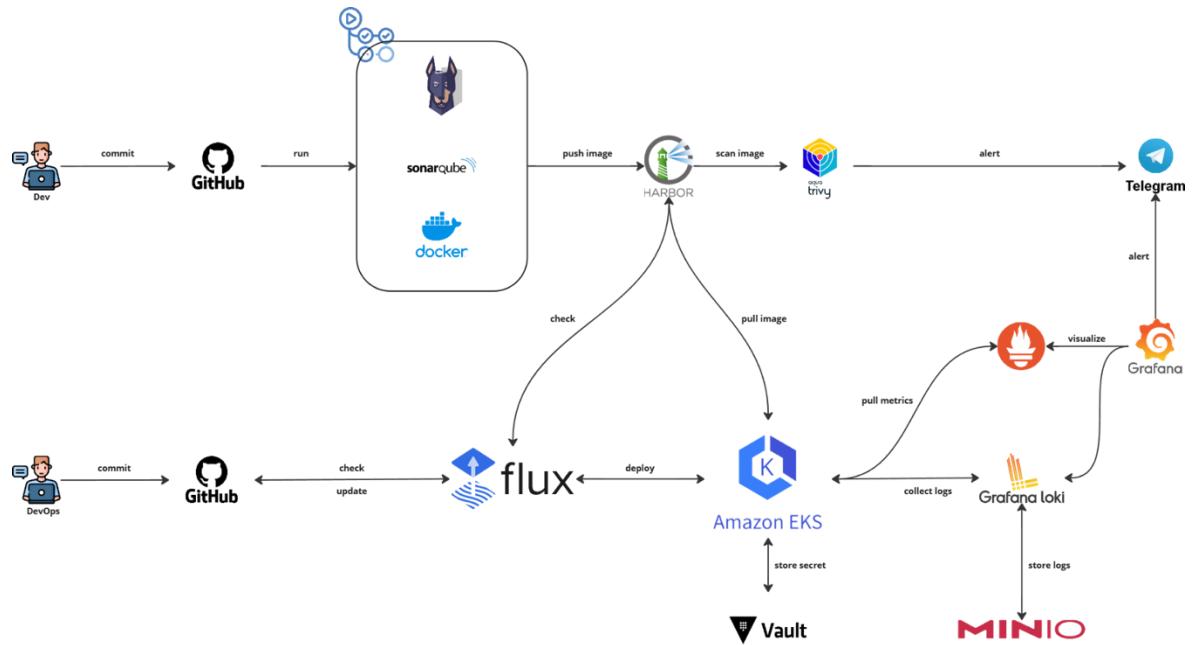
Hình 35. Mô hình triển khai hạ tầng

Đây là quy trình tự động hóa quản lý cơ sở hạ tầng theo phương pháp Infrastructure as Code. Quy trình được thiết kế để tự động hóa toàn bộ việc triển khai và cấu hình hạ tầng, từ khâu phát triển mã nguồn cho đến khi triển khai thực tế, với trọng tâm là đảm bảo tính bảo mật và nhất quán của hệ thống.

Phân tích luồng hoạt động:

- Phát triển và quản lý mã nguồn:
 - Kỹ sư DevOps viết và đẩy mã cấu hình hạ tầng lên GitHub.
 - Mã nguồn được quản lý có hệ thống và theo dõi thay đổi.
- Kiểm tra bảo mật:
 - Checkov thực hiện quét và phân tích mã nguồn.
 - Tạo báo cáo về các lỗ hổng bảo mật tiềm ẩn.
 - Đảm bảo tuân thủ các quy định về bảo mật.
- Triển khai hạ tầng:
 - Terraform nhận mã đã được kiểm tra.
 - Tự động triển khai cơ sở hạ tầng trên AWS.
 - Đảm bảo hạ tầng được tạo theo đúng cấu hình.
- Cấu hình hệ thống:
 - Ansible thực hiện cài đặt và cấu hình tự động.
 - Đảm bảo các máy chủ và dịch vụ hoạt động đúng.
 - Duy trì tính nhất quán trong toàn bộ hệ thống.

3.3. Mô hình triển khai hệ thống



Hình 36. Mô hình triển khai hệ thống

Quy trình CI/CD với tích hợp DevSecOps được xây dựng để tự động hóa các giai đoạn phát triển phần mềm, từ việc tích hợp mã nguồn (Continuous Integration) đến triển khai (Continuous Deployment), đồng thời đảm bảo yếu tố bảo mật xuyên suốt vòng đời phát triển. Đây là mô hình hiện đại được thiết kế nhằm tối ưu hóa tốc độ phát triển, giảm thiểu lỗi thủ công, và tăng tính bảo mật.

Trong quy trình này, lập trình viên chỉ cần tập trung phát triển mã nguồn, mọi bước từ kiểm tra, đóng gói, đến triển khai đều được tự động hóa. Luồng hoạt động được thiết kế xoay quanh một pipeline CI/CD, sử dụng GitHub để quản lý mã nguồn, tự động hóa việc xây dựng container image, kiểm tra bảo mật trước khi lưu trữ, và triển khai ứng dụng trên môi trường Kubernetes. Đồng thời, các hệ thống giám sát và phân tích log được tích hợp để đảm bảo mọi hoạt động vận hành được kiểm soát chặt chẽ.

Từ góc độ bảo mật, quy trình này không chỉ phát hiện các lỗ hổng sớm mà còn liên tục theo dõi, lưu trữ log, và gửi cảnh báo cho đội ngũ phát triển khi có sự cố phát sinh, đảm bảo ứng dụng luôn đạt tiêu chuẩn an toàn trước khi đến tay người dùng cuối.

Luồng hoạt động chi tiết:

- Phần CI (Continuous Integration):

- Developer đẩy mã nguồn lên GitHub:
 - Lập trình viên thực hiện các thay đổi trong mã nguồn và đẩy (commit) mã lên repository trong GitHub.
 - Việc commit sẽ kích hoạt pipeline CI được định nghĩa thông qua GitHub Actions.
- Quét bảo mật với Snyk:
 - Pipeline bắt đầu với bước quét bảo mật mã nguồn bằng Snyk để phát hiện các lỗ hổng bảo mật và các vấn đề phụ thuộc trong mã nguồn.
 - Nếu phát hiện lỗ hổng nghiêm trọng, pipeline sẽ dừng lại và thông báo lỗi để lập trình viên sửa chữa. Nếu không, pipeline sẽ chuyển sang bước tiếp theo.
- Phân tích chất lượng mã với SonarQube:
 - Mã nguồn được phân tích bằng SonarQube để kiểm tra các vấn đề liên quan đến chất lượng mã (code quality) và các lỗi tiềm ẩn.
 - Báo cáo từ SonarQube cung cấp thông tin về mức độ phức tạp, khả năng bảo trì, và những lỗ hổng an ninh tiềm ẩn trong mã nguồn.
 - Nếu đạt yêu cầu chất lượng, pipeline tiếp tục sang bước build container.
- Build và push container image:
 - GitHub Actions sử dụng Docker để build container image từ mã nguồn.
 - Sau khi build thành công, image sẽ được push lên Harbor để lưu trữ và quản lý container images.
- Tự động quét bảo mật container image với Trivy:
 - Sau khi image được lưu trữ trong Harbor, Trivy (tích hợp trong Harbor) sẽ tự động quét image để phát hiện các lỗ hổng bảo mật.

- Kết quả quét được xuất ra file CSV, bao gồm thông tin chi tiết về các lỗ hổng.
 - Sau khi Trivy hoàn tất việc quét, pipeline sẽ tự động gửi thông báo đến nhóm phát triển qua Telegram.
- **Phần CD (Continuous Deployment):**
- Flux kiểm tra và theo dõi Harbor và GitHub chứa manifest triển khai lên Kubernetes:
 - Theo dõi Harbor:
 - Flux được cấu hình để theo dõi tag của container image trong Harbor.
 - Flux trích xuất thông tin timestamp từ tag của image. Nếu phát hiện có tag mới được push lên Harbor (dựa trên timestamp), Flux sẽ coi đây là một phiên bản mới của ứng dụng.
 - Theo dõi repository GitHub chứa deployment manifest:
 - Đồng thời, Flux cũng theo dõi repository chứa file manifest Kubernetes (định nghĩa deployment, service, configmap...).
 - Bất kỳ thay đổi nào trong repository này cũng sẽ được Flux tự động xử lý để cập nhật ứng dụng.
 - Flux cập nhật deployment manifest:
 - Khi phát hiện có image với tag mới trên Harbor, Flux tự động cập nhật deployment manifest:
 - Trong phần cấu hình image, tag cũ sẽ được thay thế bằng tag mới vừa phát hiện từ Harbor.
 - Việc thay đổi tag này được thực hiện dựa trên image policy đã được định nghĩa.
 - Tự động triển khai ứng dụng trên EKS:
 - Flux apply thay đổi mới lên cụm EKS:

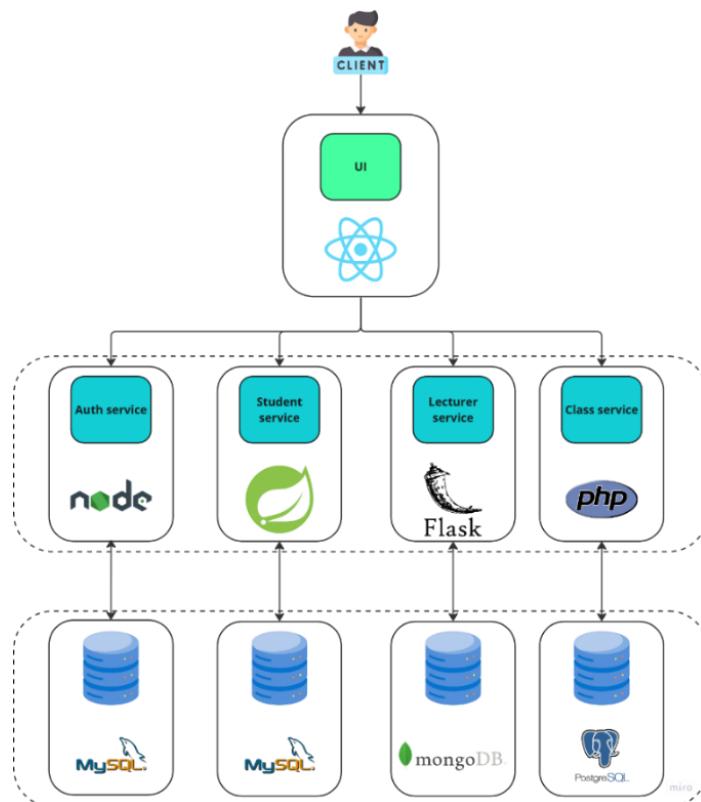
- Sau khi sửa đổi deployment manifest, Flux thực hiện apply file manifest lên cụm Kubernetes.
- Image mới sẽ được pull từ Harbor và triển khai lên cụm EKS.
- EKS thực hiện triển khai:
 - EKS, cụm Kubernetes trên AWS, tiếp nhận các thay đổi từ Flux và thực hiện deploy ứng dụng dựa trên file manifest mới.
 - Quá trình triển khai đảm bảo các tài nguyên ứng dụng (pod, service, config...) được cập nhật và chạy đúng với phiên bản mới nhất của container image.

- **Phần giám sát và log:**

- Giám sát với Grafana, Prometheus và Loki:
 - Dashboard cho Prometheus: Grafana được tích hợp với Prometheus để hiển thị dữ liệu metric từ hệ thống (CPU, RAM, Disk) và cluster Kubernetes. Các dashboard được cấu hình sẵn hoặc tạo mới để theo dõi các chỉ số trên.
 - Dashboard cho Loki:
 - Grafana tích hợp với Loki để hiển thị log từ ứng dụng và hệ thống.
 - Các dashboard log giúp đội ngũ DevOps dễ dàng lọc, tìm kiếm, và phân tích các lỗi hoặc sự cố.
- Quản lý log với Loki và MinIO:
 - Thu thập log với Loki:
 - Logs từ ứng dụng và container trong Kubernetes được thu thập bởi Loki thông qua Promtail.
 - Logs được lưu trữ trong Loki để phục vụ cho việc giám sát và phân tích.
 - Lưu trữ log lâu dài với MinIO: MinIO được tích hợp làm nơi lưu trữ log lâu dài để đảm bảo khả năng truy xuất khi cần.
- Cấu hình Alert Rules trên Grafana:

- Các alert rules được cấu hình trực tiếp trên Grafana để giám sát các chỉ số từ Prometheus.
- Grafana sử dụng Alerting UI để tạo và quản lý các rule:
 - Conditions: Xác định điều kiện cảnh báo.
 - Threshold: Ngưỡng kích hoạt cảnh báo.
 - Notification channels: Kết nối với Telegram để gửi thông báo.
- Khi đạt tới điều kiện và ngưỡng threshold đã cấu hình, Grafana sẽ gửi alert đến Telegram để cảnh báo.

3.4. Ứng dụng microservices



Hình 37. Ứng dụng microservice

Các tính năng đã triển khai:

- **Auth service:**
 - Ngôn ngữ: JavaScript (NodeJs, ExpressJs)
 - Cơ sở dữ liệu: MySQL

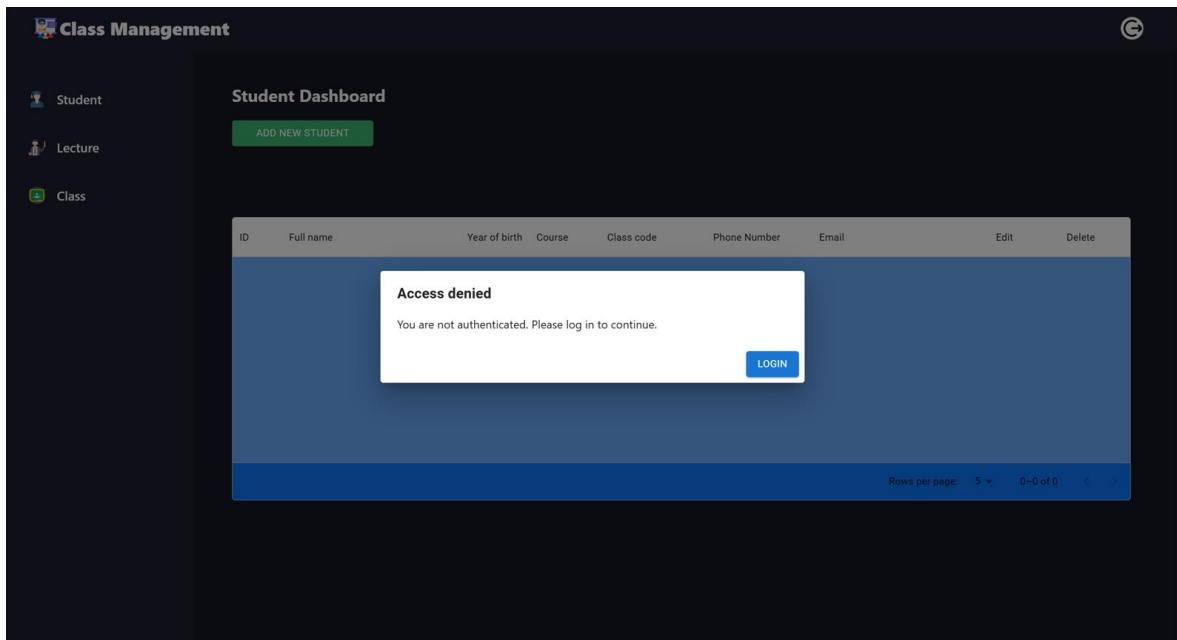
- Chức năng: Xử lý các chức năng liên quan đến xác thực người dùng, bao gồm đăng nhập, đăng ký, và quản lý phiên làm việc của người dùng.
- **Student service:**
 - Ngôn ngữ: Java (Spring Boot framework)
 - Cơ sở dữ liệu: MySQL
 - Chức năng: Quản lý thông tin sinh viên, bao gồm lưu trữ, truy xuất, cập nhật và xóa thông tin sinh viên.
- **Lecturer service:**
 - Ngôn ngữ: Python (Flask)
 - Cơ sở dữ liệu: MongoDB
 - Chức năng: Quản lý thông tin giảng viên, bao gồm lưu trữ, truy xuất, cập nhật và xóa thông tin giảng viên.
- **Class service:**
 - Ngôn ngữ: PHP
 - Cơ sở dữ liệu: PostgreSQL
 - Chức năng: Quản lý thông tin lớp học, bao gồm lưu trữ, truy xuất, cập nhật và xóa thông tin lớp học.

Danh sách các API đã triển khai:

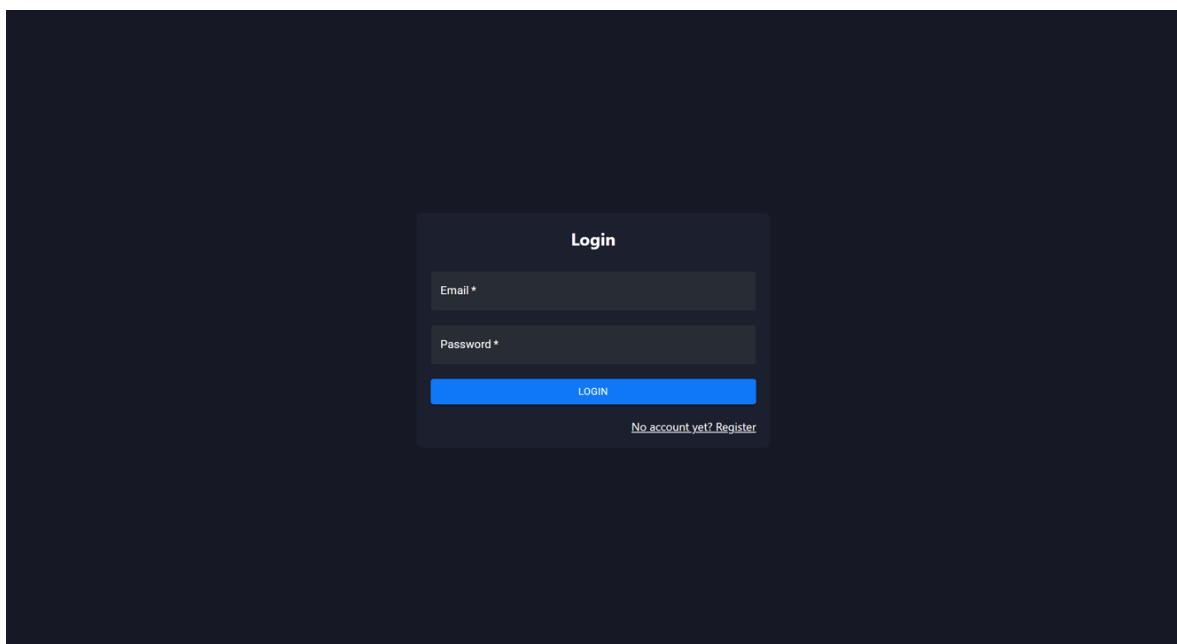
Service	Base URL	Trạng thái	URL(s)
Auth service	https://auth-api.th1enlm02.live	GET	https://auth-api.th1enlm02.live
		POST	https://auth-api.th1enlm02.live/login
		POST	https://auth-api.th1enlm02.live/register
		GET	https://auth-api.th1enlm02.live/logout
Student service	https://student-api.th1enlm02.live	GET	https://student-api.th1enlm02.live/student/getAll
		POST	https://student-api.th1enlm02.live/student/add
		PUT	https://student-api.th1enlm02.live/student/{id}
		DELETE	https://student-api.th1enlm02.live/student/{id}
Lecturer service	https://lecturer-api.th1enlm02.live	GET	https://lecturer-api.th1enlm02.live/lecturer/getAll
		POST	https://lecturer-api.th1enlm02.live/lecturer/add
		PUT	https://lecturer-api.th1enlm02.live/lecturer/{id}
		DELETE	https://lecturer-api.th1enlm02.live/lecturer/{id}
Class service	https://class-api.th1enlm02.live	GET	https://class-api.th1enlm02.live/class/getAll
		POST	https://class-api.th1enlm02.live/class/add
		PUT	https://class-api.th1enlm02.live/class/{classCode}
		DELETE	https://class-api.th1enlm02.live/class/{classCode}

Hình 38. Danh sách các API của ứng dụng

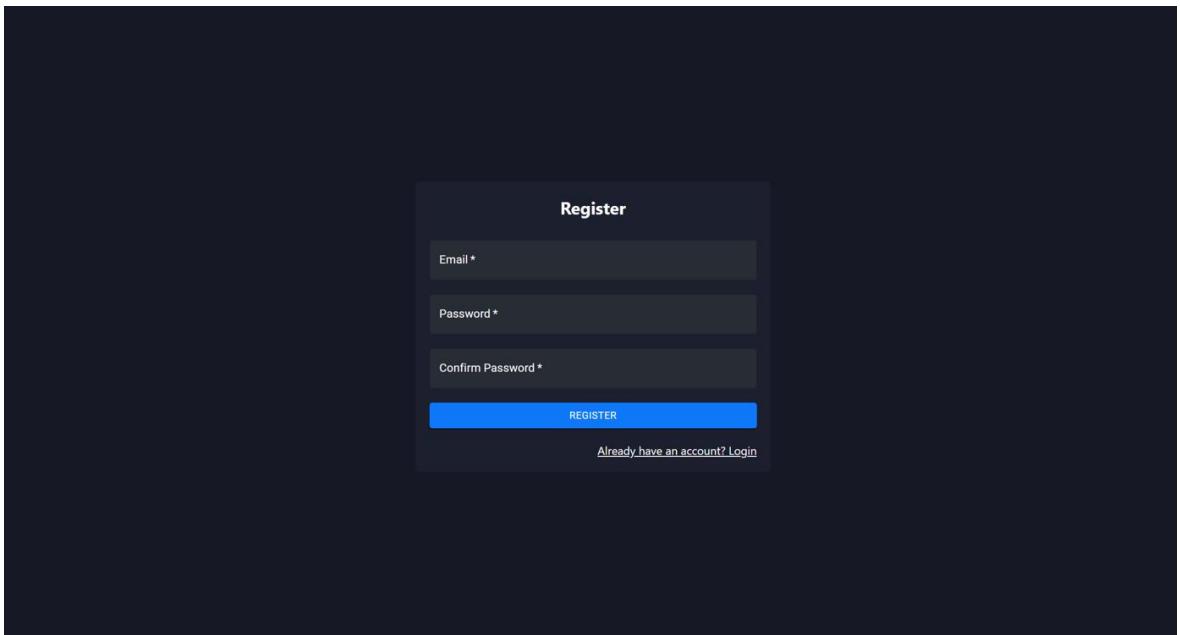
Màn hình ứng dụng:



Hình 39. Màn hình xác thực người dùng



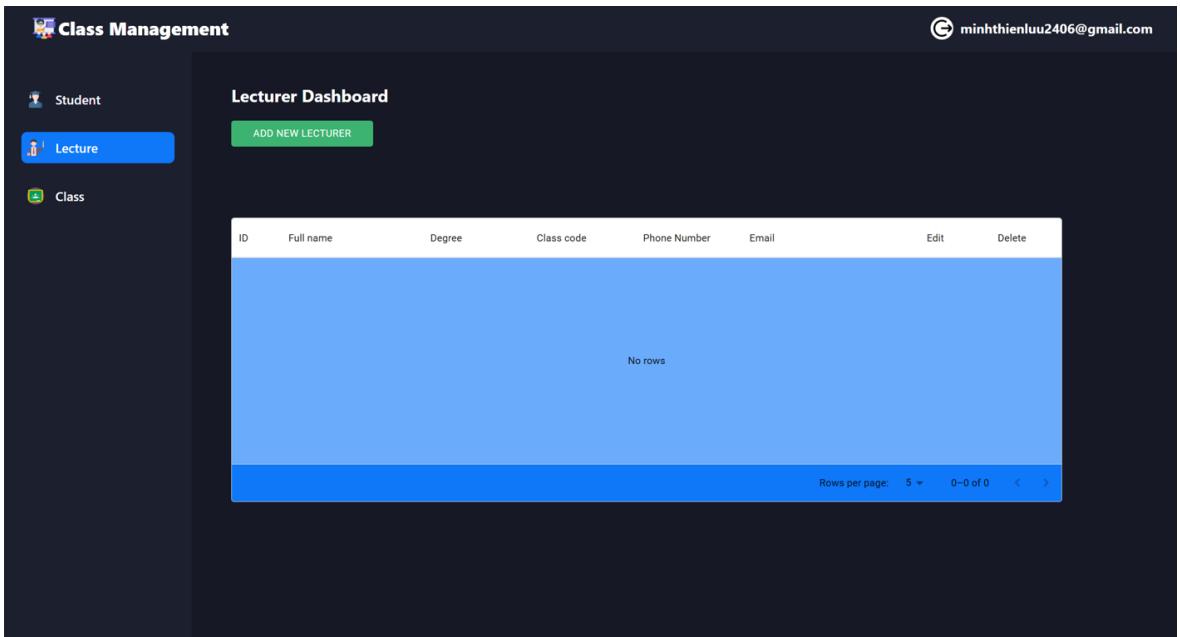
Hình 40. Màn hình đăng nhập



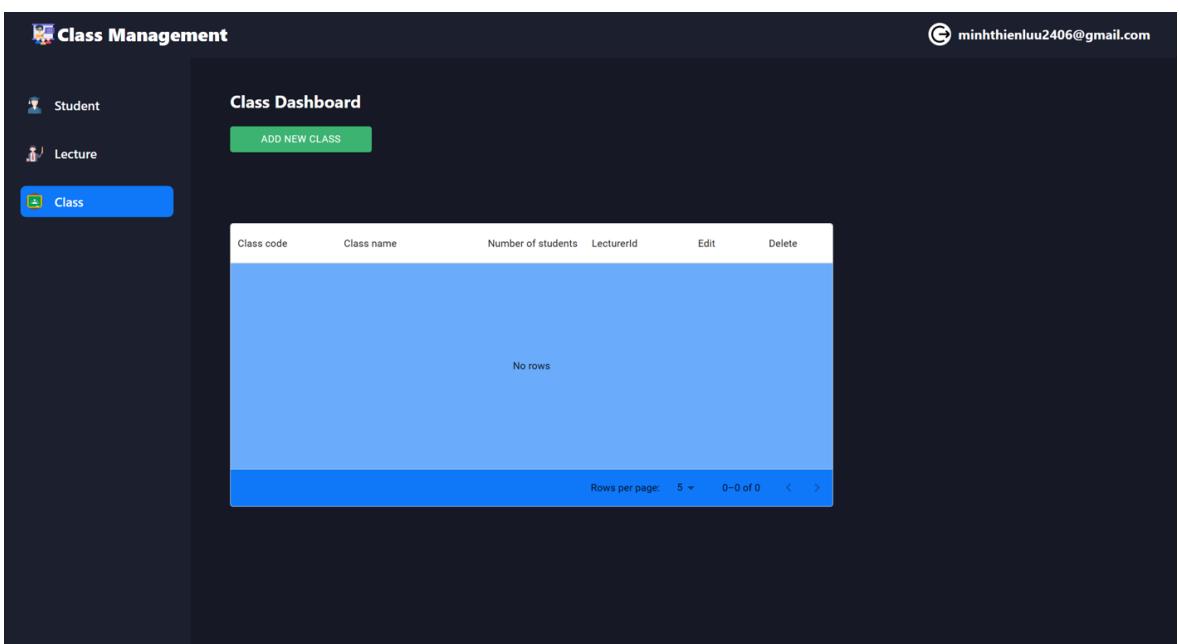
Hình 41. Đăng ký

A screenshot of the "Student Dashboard" in the "Class Management" application. The dashboard has a sidebar with "Student" (selected), "Lecture", and "Class" options. The main area shows a table with columns: ID, Full name, Year of birth, Course, Class code, Phone Number, Email, Edit, and Delete. A message "No rows" is displayed in the table area. At the bottom right, there are pagination controls: "Rows per page: 5", "0-0 of 0", and navigation arrows.

Hình 42. Màn hình quản lý sinh viên



Hình 43. Màn hình quản lý giảng viên



Hình 44. Màn hình quản lý lớp học

3.5. Quản lý tên miền

Cloudflare là một nền tảng dịch vụ đám mây nổi tiếng, cung cấp các giải pháp bảo mật, hiệu suất và quản lý hạ tầng mạng cho các ứng dụng và website. Với mạng lưới máy chủ toàn cầu, Cloudflare giúp bảo vệ và tăng tốc các trang web, API, và

dịch vụ trực tuyến bằng cách lọc lưu lượng truy cập độc hại và tối ưu hóa tốc độ tải trang.

Quản lý DNS của tên miền đã đăng ký:

- Cloudflare được sử dụng để quản lý DNS cho các tên miền của hệ thống.
- Các bản ghi DNS (A, CNAME) được cấu hình trên Cloudflare để trỏ tới các dịch vụ như:
 - Load balancer của scaling group.
 - Các server bên trong hệ thống.
 - Các API hoặc ứng dụng frontend/backend.

Type	Name	Content	Proxy status	TTL	Actions
CNAME	application	th1enlm02.live	DNS only	Auto	Edit
CNAME	auth-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	class-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	grafana	th1enlm02.live	DNS only	Auto	Edit
CNAME	harbor	th1enlm02.live	DNS only	Auto	Edit
CNAME	lecturer-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	loki	th1enlm02.live	DNS only	Auto	Edit
CNAME	minio	th1enlm02.live	DNS only	Auto	Edit
CNAME	prometheus	th1enlm02.live	DNS only	Auto	Edit
CNAME	sonarqube	th1enlm02.live	DNS only	Auto	Edit
CNAME	stag-application	th1enlm02.live	DNS only	Auto	Edit
CNAME	stag-auth-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	stag-class-api	th1enlm02.live	DNS only	Auto	Edit
CNAME	stag-lecturer-api	th1enlm02.live	DNS only	Auto	Edit

Hình 45. Quản lý tên miền

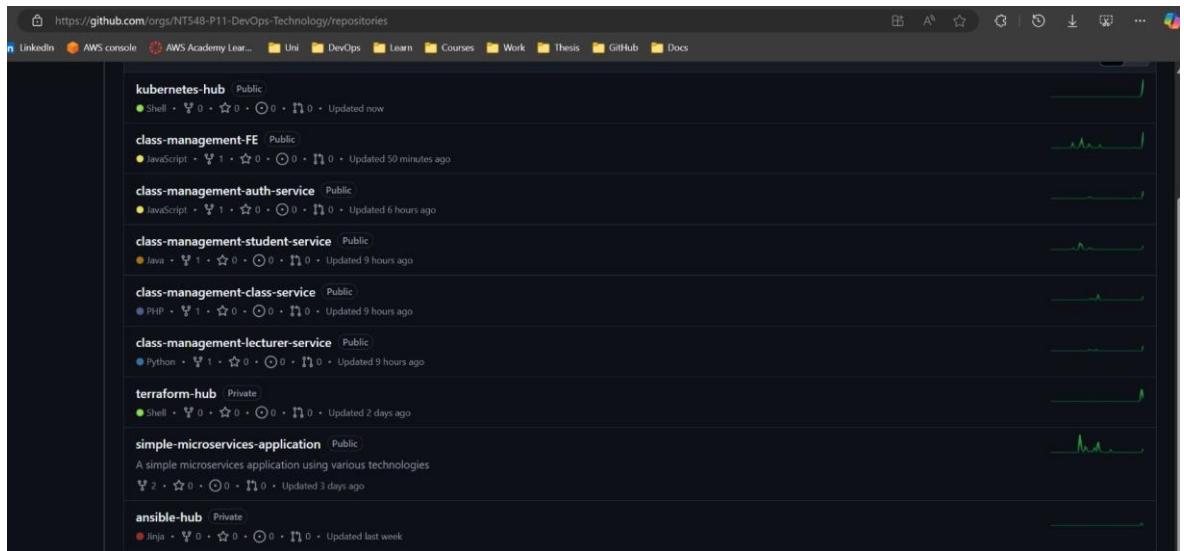
3.6. Quản lý mã nguồn

GitHub cung cấp một nền tảng quản lý mã nguồn mạnh mẽ, hỗ trợ quy trình phát triển phần mềm hiện đại và cộng tác hiệu quả giữa các thành viên trong nhóm.

Lưu trữ và tổ chức repository:

- Mã nguồn của các ứng dụng (microservices) được tổ chức thành các repository riêng biệt trong một GitHub Organization để dễ quản lý.

- Repository chứa mã nguồn, tài liệu, và các file cấu hình CI/CD (như GitHub Actions).
- Mã nguồn cho cơ sở hạ tầng (Ansible, Terraform).
- Kubernetes manifest.



Hình 46. Quản lý mã nguồn

Chương 4. HIỆN THỰC ĐỀ TÀI

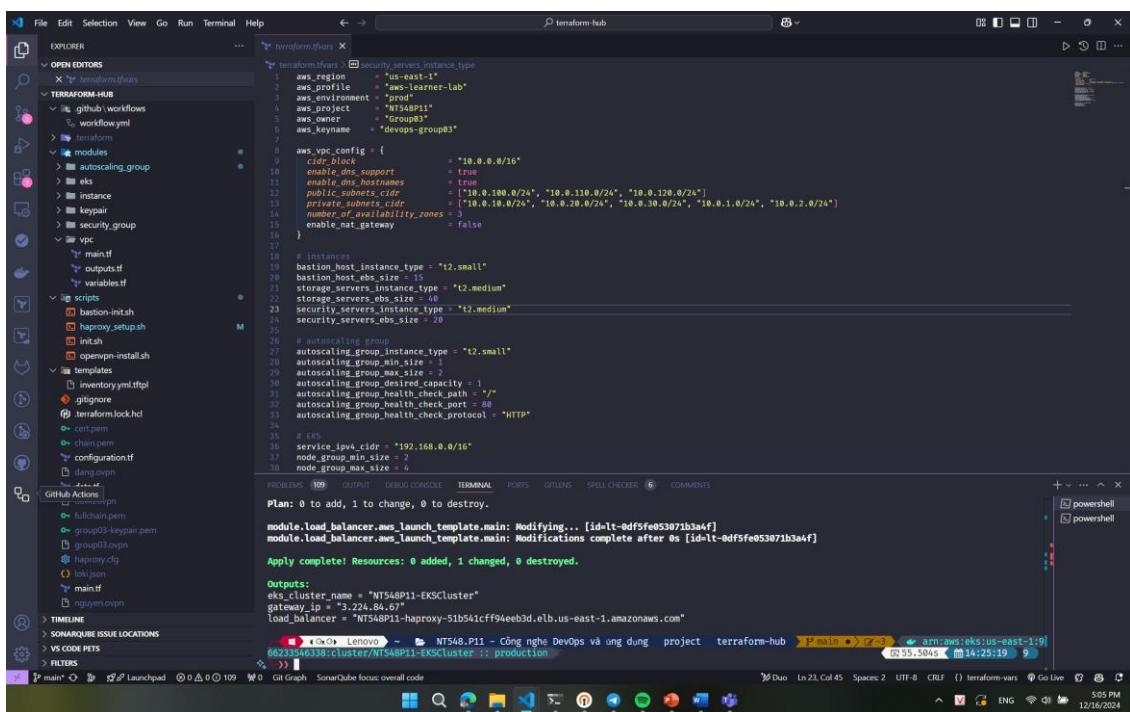
4.1. Triển khai cơ sở hạ tầng

Terraform là một công cụ tự động hóa mã nguồn mở giúp triển khai, quản lý và duy trì cơ sở hạ tầng dưới dạng mã. Sử dụng Terraform định nghĩa các resource sẽ triển khai cho hệ thống, sau đó apply để triển khai toàn bộ cơ sở hạ tầng. Triển khai các tài nguyên cơ bản trên AWS như VPC, EC2 instances, Auto Scaling, và EKS (Elastic Kubernetes Service).

Cụ thể thông tin resouce được triển khai như sau:

- Cấu hình VPC:
 - CIDR Block: Mạng con cho toàn bộ VPC sử dụng CIDR 10.0.0.0/16.
 - Public Subnets: Ba subnet công cộng được định nghĩa với CIDR 10.0.100.0/24, 10.0.110.0/24, và 10.0.120.0/24.
 - Private Subnets: Năm subnet riêng được định nghĩa với các CIDR khác nhau cho mỗi vùng Availability Zone (AZ), bao gồm 10.0.10.0/24, 10.0.20.0/24, 10.0.30.0/24, 10.0.1.0/24, và 10.0.2.0/24.
- Các loại instance EC2:
 - Bastion Host: Một instance EC2 loại t2.small được sử dụng làm bastion host để truy cập an toàn vào các tài nguyên trong VPC thông qua SSH.
 - Storage Servers: Các server lưu trữ được triển khai với loại instance t2.medium và dung lượng EBS là 40GB.
 - Security Servers: Các server bảo mật cũng sử dụng loại instance t2.medium và dung lượng EBS là 20GB.
- Auto Scaling Group:
 - Instance Type: Các instance sử dụng loại t2.small.
 - Minimum Size: Tối thiểu 1 instance EC2.
 - Maximum Size: Tối đa 2 instance EC2.
 - Desired Capacity: Số lượng instance mong muốn là 1.

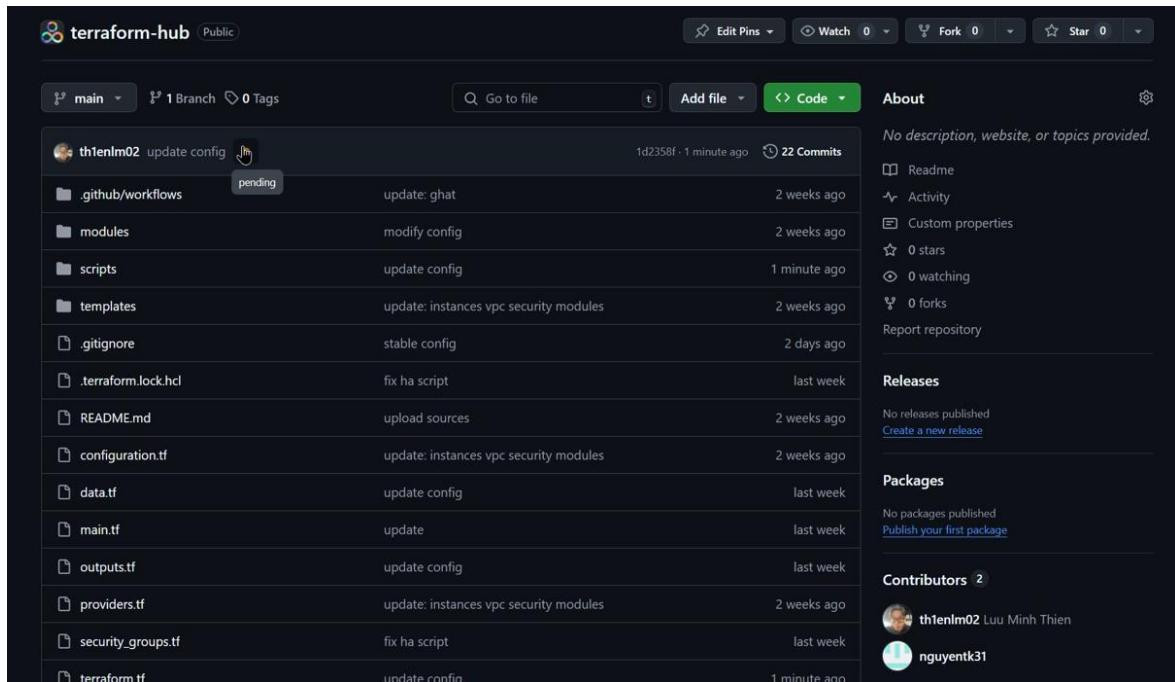
- Health Check: Các instance trong Auto Scaling Group sẽ được kiểm tra sức khỏe qua HTTP trên port 80. Nếu một instance không đáp ứng yêu cầu sức khỏe, Auto Scaling sẽ thay thế nó.
- EKS:
 - Cấu hình dịch vụ EKS sử dụng CIDR 192.168.0.0/16.
 - Cấu hình cho node group của EKS với các thông số:
 - Minimum Size: 2 node.
 - Maximum Size: 4 node.
 - Desired Size: 4 node.



```

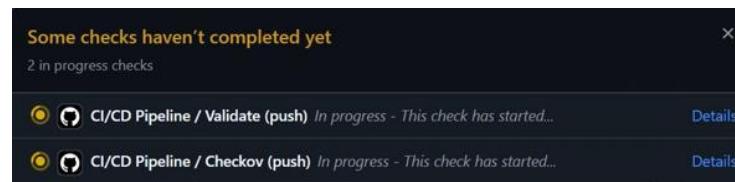
File Edt Selection View Go Run Terminal Help
... terraform-hub ...
EXPLORER
  OPEN EDITORS
    terraform-hub
  TERRAFORM-HUB
    .github/workflows
      workflow.yml
    terraform
      modules
        autoscaling_group
        eks
        instance
        keypair
        security_group
      vpc
        main.tf
        outputs.tf
        variables.tf
      scripts
        bastion-init.sh
        haproxy-setup.sh
        initd.sh
        openvpn-install.sh
      templates
        inventory.yml.tpl
        .gitignore
        .terrafom.lock.hcl
        .cert.pem
        .chain.pem
        configuration.tf
        dang.openvpn
      GitHub Actions
        fullchain.pem
        group03-keypair.pem
        group03.openvpn
        haproxy.cfg
        loki.json
        main.tf
        ngnx.openvpn
      TIMELINE
      SONARQUBE ISSUE LOCATIONS
      VS CODE PETS
      RULES
PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER COMMENTS
Plan: 0 to add, 1 to change, 0 to destroy.
module.load_balancer.aws_launch_template.main: Modifying... [id=lt-0df5fe053071b3a4f]
module.load_balancer.aws_launch_template.main: Modifications complete after 0s [id=lt-0df5fe053071b3a4f]
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
Outputs:
eks_cluster_name = "NT548P11-EKSCluster"
gateway_ip = "3.224.84.67"
load_balancer = "NT548P11-haproxy-51b541cff94eeb3d.elb.us-east-1.amazonaws.com"
  
```

Hình 47. Triển khai cơ sở hạ tầng

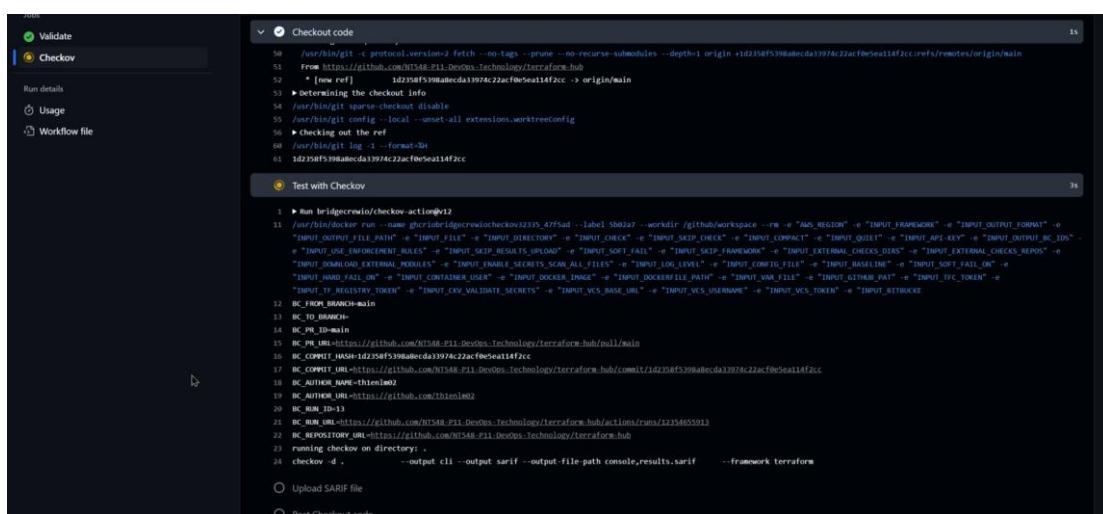


Hình 48. GitHub Actions được trigger khi có commit mới lên source Terraform
GitHub Actions sẽ chạy các job định nghĩa trong workflow bao gồm:

- Kiểm tra cú pháp của cấu hình mã Terraform.
- Quét bảo mật với Checkov để phát hiện các lỗ hổng.



Hình 49. Workflow cho pipeline triển khai



Hình 50. Quét bảo mật với Checkov

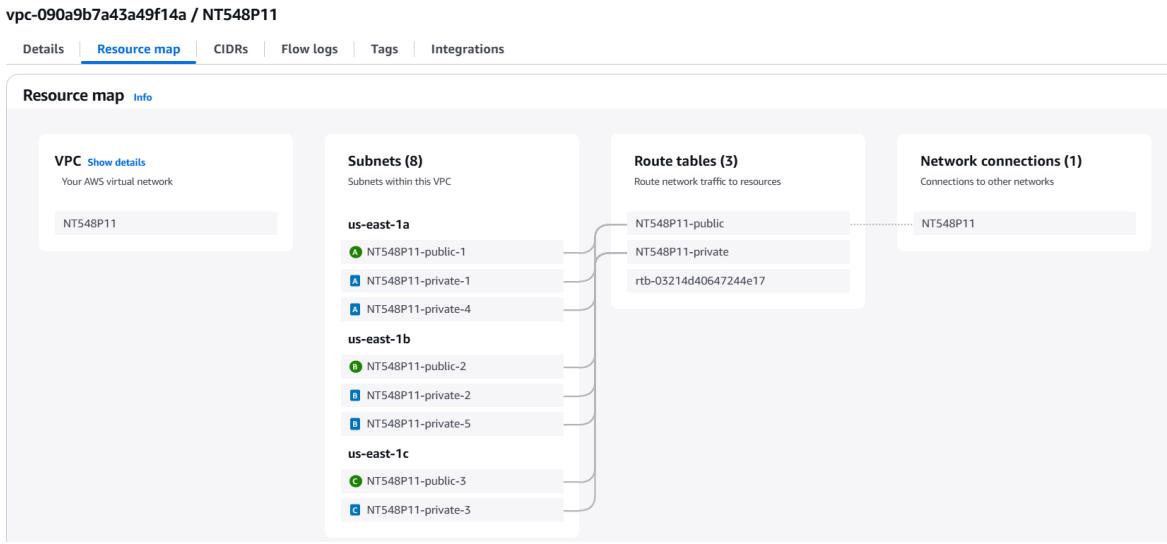
Hình 51. Kết quả sau khi scan với Checkov

Hình 52. Thông tin chi tiết của tệp SARIF

Các resource sau khi được triển khai thành công:

Instances (8) Info							
Find Instance by attribute or tag (case-sensitive)		Instance ID	Instance state	Instance ...	Status check	Alarm status	Availability
Name							Public IPv4 DNS
NT548P11-storage-servers	i-03f41274ef12480b1	Running	t2.medium	2/2 checks passed	View alarms +	us-east-1c	-
NT548P11-security-servers	i-0cbd7be5544ffed8a	Running	t2.medium	2/2 checks passed	View alarms +	us-east-1c	-
NT548P11-haproxy	i-06b14dff3c4877b2	Running	t2.small	2/2 checks passed	View alarms +	us-east-1a	-
NT548P11-gateway	i-0677c0b27f75a7992	Running	t2.small	2/2 checks passed	View alarms +	us-east-1a	ec2-3-224-84-67.comp...
	i-0846ff9d81db0c8	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1b	-
	i-0861ada9471362e5b	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1c	-
	i-0c8fd5707093d303	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1a	-
	i-03e4d5856101ed364	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1b	-

Hình 53. Kiểm tra thông tin các EC2



Hình 54. Kiểm tra thông tin VPC

The screenshot shows the AWS Auto Scaling groups page. It displays two Auto Scaling groups: 'eks-NT548P11-EKSNodegroup' (with 4 instances) and 'NT548P11-haproxy' (with 1 instance). The page includes filters for Name, Launch template/configuration, Instances, Status, Desired..., Min, Max, and Availability Zones.

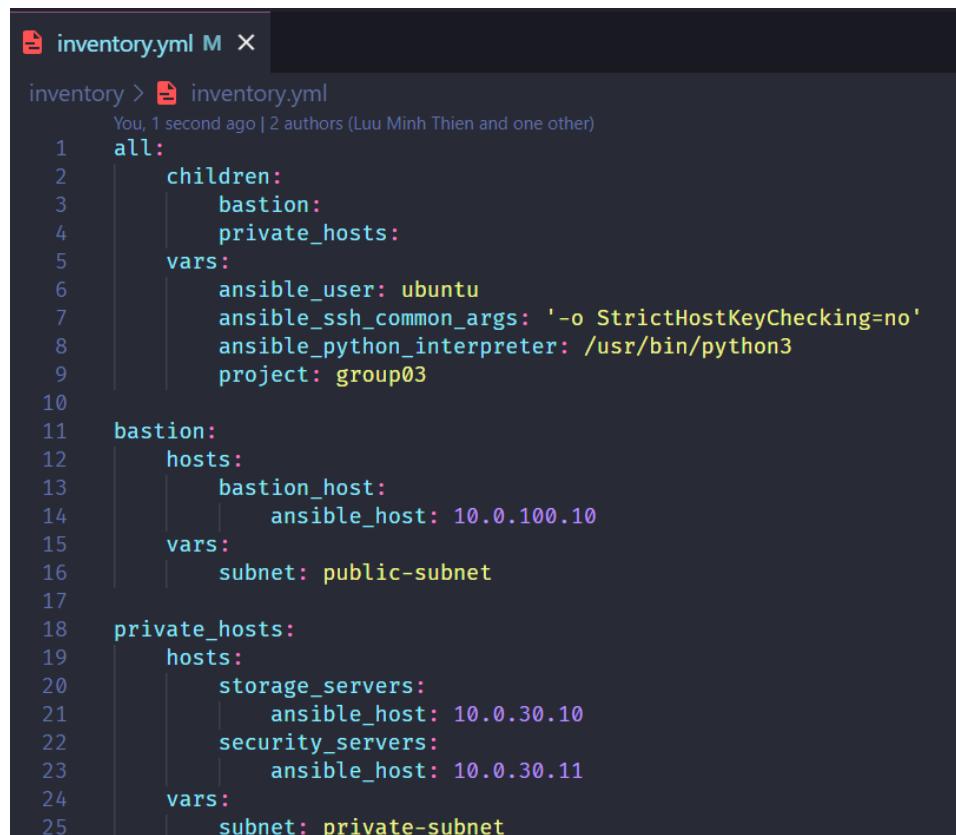
Hình 55. Kiểm tra thông tin Auto Scaling Group

The screenshot shows the AWS EKS Cluster page for 'NT548P11-EKSCluster'. It includes sections for Cluster info (Status: Active, Kubernetes version: 1.29, Support period: Standard support until March 23, 2025, Provider: EKS), Nodes (4), and Node groups (1). The Nodes table lists four t3.medium instances, each created 21 minutes ago and marked as Ready. The Node groups table shows one group named 'NT548P11-EKSNodegroup' with a desired size of 4, AMI release version 1.29.10-20241213, and an active status.

Hình 56. Kiểm tra thông tin EKS

4.2. Cấu hình cơ sở hạ tầng

Ansible là một công cụ quản lý cấu hình và tự động hóa triển khai mã nguồn mở, giúp tự động hóa các tác vụ như cấu hình máy chủ, triển khai ứng dụng, và quản lý cơ sở hạ tầng. Cấu hình inventory đóng vai trò quan trọng trong Ansible, định nghĩa các nhóm máy chủ và các biến liên quan.



```
inventory > inventory.yml
You, 1 second ago | 2 authors (Luu Minh Thien and one other)
1   all:
2     children:
3       bastion:
4       private_hosts:
5     vars:
6       ansible_user: ubuntu
7       ansible_ssh_common_args: '-o StrictHostKeyChecking=no'
8       ansible_python_interpreter: /usr/bin/python3
9       project: group03
10
11  bastion:
12    hosts:
13      bastion_host:
14        ansible_host: 10.0.100.10
15    vars:
16      subnet: public-subnet
17
18  private_hosts:
19    hosts:
20      storage_servers:
21        ansible_host: 10.0.30.10
22      security_servers:
23        ansible_host: 10.0.30.11
24    vars:
25      subnet: private-subnet
```

Hình 57. Inventory file

Tiến hành cấu hình cho các instance được tạo, trong đó chứa các role tương ứng cho từng host và việc cấu hình sẽ được chạy tự động bằng Ansible. Role là một cách tổ chức các playbook và tài nguyên trong Ansible, giúp quản lý và tái sử dụng cấu hình một cách hiệu quả. Một role chia nhỏ các tác vụ phức tạp thành các thành phần nhỏ, dễ quản lý và tái sử dụng.

```

inventory > inventory.yml
inventory > inventory.yml
You, 47 others, 917 authors (Luu Minh Thien and one other)
1 all:
2   children:
3     bastion:
4       hosts:
5         private_hosts:
6           vars:
7             ansible_user: ubuntu
8             ansible_ssh_common_args: '-o StrictHostKeyChecking=no'
9             ansible_python_interpreter: /usr/bin/python3
10            project: group1
11
12      bastion:
13        hosts:
14          bastion_host:
15            ansible_host: 10.0.100.10
16
17      private_hosts:
18        hosts:
19          storage_servers:
20            ansible_host: 10.0.30.10
21          security_servers:
22            ansible_host: 10.0.30.11
23
24      vars:
25        subnet: private-subnet
26

```

PLAY [Ping all hosts] ****

TASK [Ping all hosts] ****

ok: [bastion_host]

ok: [storage_servers]

ok: [security_servers]

PLAY RECAP ****

bastion_host : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

security_servers : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

storage_servers : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

Hình 58. Cấu hình cơ sở hạ tầng

4.3. Cấu hình các máy chủ

4.3.1. Cấu hình HA Proxy

HAProxy được cấu hình làm reverse proxy và load balancer, cung cấp khả năng định tuyến các yêu cầu HTTP/HTTPS đến các backend cụ thể dựa trên tiêu chí ACL (Access Control List). Nó cũng đảm bảo tính bảo mật bằng cách hỗ trợ giao thức HTTPS và các chính sách CORS.

Frontend:

```

frontend https-in
  bind :80
  bind :443 ssl crt /etc/haproxy/certs

  mode http
  acl is_cors_preflight method OPTIONS
  acl is_app_domain hdr(host) -i application.thienlm02.live
  acl is_stag_domain hdr(host) -i stag-application.thienlm02.live
  http-request set-header Access-Control-Allow-Origin "https://application.thienlm02.live" if is_app_domain
  http-request set-header Access-Control-Allow-Origin "https://stag-application.thienlm02.live" if is_stag_domain
  http-request set-header X-Forwarded-Proto https
  http-response set-header Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS"
  http-response set-header Access-Control-Allow-Headers "Content-Type, Authorization, X-Theme, Client-Id, Client-Secret, fineract-platform-tenantid"
  http-response set-header Access-Control-Max-Age "3600" if is_cors_preflight

  acl ACL_harbor hdr(host) -i harbor.thienlm02.live www.harbor.thienlm02.live
  use_backend backend_harbor if ACL_harbor

  acl ACL_minio hdr(host) -i minio.thienlm02.live www.minio.thienlm02.live
  use_backend backend_minio if ACL_minio

  acl ACL_minio_api hdr(host) -i minio-api.thienlm02.live www.minio-api.thienlm02.live
  use_backend backend_minio_api if ACL_minio_api

  acl ACL_vault hdr(host) -i vault.thienlm02.live www.vault.thienlm02.live
  use_backend backend_vault if ACL_vault

  acl ACL_sonarqube hdr(host) -i sonarqube.thienlm02.live www.sonarqube.thienlm02.live
  use_backend backend_sonarqube if ACL_sonarqube

  default_backend eks_cluster
  redirect scheme https code 301 if /{ ssl_fc }

```

Hình 59. Cấu hình HA Proxy: Frontend

Frontend nhận các yêu cầu HTTP/HTTPS và định tuyến chúng đến các backend phù hợp:

- Binding: Lắng nghe trên cổng 80 (HTTP) và 443 (HTTPS) với chứng chỉ SSL tại /etc/haproxy/certs.
- CORS Support:
 - Sử dụng các ACL để kiểm tra tên miền và thêm header CORS (Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers).
 - Xử lý yêu cầu OPTIONS (preflight) với header Access-Control-Max-Age.
- Redirect: Tự động chuyển HTTP sang HTTPS nếu yêu cầu không sử dụng SSL.
- ACLs và backend:
 - Định tuyến các tên miền cụ thể đến backend tương ứng như harbor, minio, vault, và sonarqube.
 - Sử dụng backend mặc định eks_cluster nếu không khớp ACL.

Backend:

```
backend eks_cluster
    server eks_lb a4a904cae36f8485eae23fefef4fdf640-2b239e9dadd2bcf0.elb.us-east-1.amazonaws.com:80 check

backend backend_harbor
    server harbor 10.0.30.10:80 check

backend backend_minio
    server minio 10.0.30.10:9001 check

backend backend_minio_api
    server minio_api 10.0.30.10:9000 check

backend backend_vault
    server vault 10.0.30.11:8200 check

backend backend_sonarqube
    server sonarqube 10.0.30.11:9000 check
```

Hình 60. Cấu hình HA Proxy: Backend

Mỗi backend đại diện cho một dịch vụ hoặc ứng dụng:

- eks_cluster: Định tuyến đến ELB của AWS với kiểm tra trạng thái (check).
- backend_harbor: Dịch vụ Harbor tại 10.0.30.10:80.

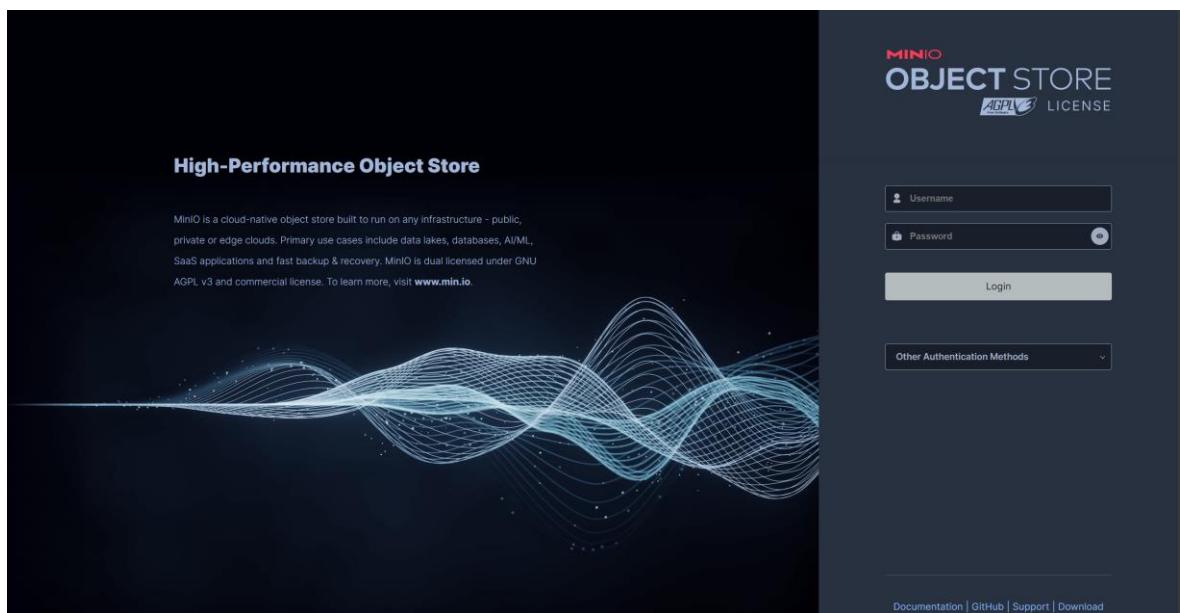
- backend_minio và backend_minio_api: MinIO quản lý tại cổng 9001 và API tại cổng 9000.
- backend_vault: Dịch vụ Vault tại 10.0.30.11:8200.
- backend_sonarqube: Dịch vụ SonarQube tại 10.0.30.11:9000.

4.3.2. Nhóm công cụ lưu trữ

4.3.2.1. Cấu hình MinIO

Trong cấu hình hiện tại, MinIO được triển khai với hai dịch vụ:

- MinIO web UI: Giao diện quản trị trên cổng 9001.
- MinIO API: Dịch vụ API trên cổng 9000.



Hình 61. MinIO: Giao diện đăng nhập

MinIO đóng vai trò lưu trữ các phần sau:

- Lưu trữ cấu hình cho HAProxy:
 - MinIO được sử dụng làm nơi lưu trữ các tệp cấu hình quan trọng của HAProxy như:
 - Tệp cấu hình chính của HAProxy (haproxy.cfg)
 - Các khóa SSL và chứng chỉ cho HTTPS
 - Khi một EC2 instance mới trong Auto Scaling Group được tạo, nó sẽ tự động tải về các tệp cấu hình từ MinIO để đảm bảo cấu hình nhất quán.

- Lưu trữ log cho Loki:

- MinIO hoạt động như một storage backend cho Loki, nơi lưu trữ log hệ thống hoặc log ứng dụng được thu thập.
- Điều này giúp đảm bảo log được lưu trữ lâu dài và có thể truy xuất một cách linh hoạt khi cần.

The screenshot shows the MinIO Object Browser interface. On the left, there's a sidebar with navigation links for User (Object Browser, Access Keys, Documentation), Administrator (Buckets, Policies, Identity, Monitoring, Events, Configuration, License), and a license key. The main area is titled 'Object Browser' with a search bar labeled 'Filter Buckets'. It displays a table with three columns: Name, Objects, and Size. The 'haproxy-config' bucket has 2 objects (8.8 kB) and the 'loki' bucket has 140 objects (2.2 MB). Both buckets have 'R/W' access.

Hình 62. Thông tin các bucket trên MinIO

Cấu hình tạo access key cho MinIO dùng để xác thực khi sử dụng MinIO client kết nối đến.

The screenshot shows the 'Create Access Key' page in the MinIO console. It includes a 'Create Access Key' button, a 'Learn more about Access Keys' link, and sections for 'Assign Custom Credentials' and 'Assign Access Policies'. The main form fields are: Access Key (pMo5IVFMWkI9rNEhKxyy), Secret Key (redacted), Restrict beyond user policy (OFF), Expiry (December 31, 2024 00:00), Name (group03), Description (group03), and Comments (group03). There are also 'Clear' and 'Create' buttons at the bottom.

Hình 63. Cấu hình tạo access key cho MinIO

4.3.2.2. Cấu hình Harbor

Harbor được sử dụng để quản lý và lưu trữ các images của từng microservice. Mỗi ứng dụng được tổ chức thành các projects riêng biệt trong Harbor, phân tách rõ ràng theo môi trường (production, staging).



Hình 64. Harbor: Giao diện đăng nhập

A screenshot of the Harbor 'Projects' page. The left sidebar shows navigation links for 'Projects', 'Logs', 'Administration' (with sub-links for 'Users', 'Robot Accounts', 'Registries', 'Replications', 'Distributions', 'Labels', 'Project Quotas', 'Interrogation Services', 'Clean Up', 'Job Service Dashboard', and 'Configuration'). The main content area displays project statistics: 2 Private, 0 Public, and 2 Total projects; 9 Private, 0 Public, and 9 Total repositories; and a quota used of 1.32 GB. Below this is a table listing projects: 'production' (Private, Project Admin, Project, 5 repos, created 12/16/24, 2:32 PM) and 'staging' (Private, Project Admin, Project, 4 repos, created 12/16/24, 2:32 PM). A bottom footer shows page size and item count information.

Hình 65. Thông tin các project trên Harbor

- Production: Chứa các image phục vụ triển khai ứng dụng ở môi trường production.

The screenshot shows the Harbor interface for the 'production' project. At the top, it displays 'Access Level: Private' and 'Quota used: 1.12GiB of unlimited'. Below this is a navigation bar with tabs: Summary, Repositories (which is selected), Members, Labels, Scanner, P2P Preheat, Policy, Robot Accounts, Webhooks, Logs, and Configuration. A search bar and filter options are also present. The main content area shows a table of repositories:

Name	Artifacts	Pulls	Last Modified Time
production/class-management-lecturer-service	1	3	12/17/24, 2:13 AM
production/class-management-class-service	1	3	12/17/24, 2:13 AM
production/class-management-student-service	1	3	12/17/24, 2:13 AM
production/class-management-fe	1	4	12/17/24, 2:13 AM
production/class-management-auth-service	1	3	12/17/24, 2:02 AM

At the bottom right, there are page size and item count controls.

Hình 66. Project production trên Harbor

- Staging: Chứa các image phục vụ triển khai ứng dụng ở môi trường staging.

The screenshot shows the Harbor interface for the 'staging' project. It has the same layout as the 'production' project, with 'Access Level: Private' and 'Quota used: 1.23GiB of unlimited'. The 'Repositories' tab is selected. The main content area shows a table of repositories:

Name	Artifacts	Pulls	Last Modified Time
staging/class-management-fe	2	7	12/17/24, 2:29 AM
staging/class-management-lecturer-service	2	5	12/17/24, 2:13 AM
staging/class-management-class-service	2	5	12/17/24, 2:13 AM
staging/class-management-auth-service	1	4	12/17/24, 2:13 AM
staging/class-management-student-service	2	5	12/17/24, 2:01 AM

At the bottom right, there are page size and item count controls.

Hình 67. Project staging trên Harbor

Harbor đóng vai trò trung tâm trong việc quản lý image của các microservices. Với cách tổ chức theo projects (production, staging) và repositories (services), hệ thống đảm bảo tính rõ ràng và dễ quản lý trong việc triển khai các ứng dụng.

The screenshot shows the Harbor interface for the 'class-management-fe' repository under the 'production' project. The 'Artifacts' tab is selected. It shows two tagged images:

Artifacts	Tags	Signed	Size	Vulnerabilities	SBOM	Labels	Push Time	Pull Time
sha256:dbbbd4f1	latest, 3e... (2)	☒	158.95MiB	H 22 Total - 22 Fixable	No SBOM		12/17/24, 2:29 AM	12/17/24, 2:30 AM
sha256:ef14b163	b9f6c7c-1..	☒	158.94MiB	H 22 Total - 22 Fixable	No SBOM		12/17/24, 1:39 AM	12/17/24, 2:13 AM

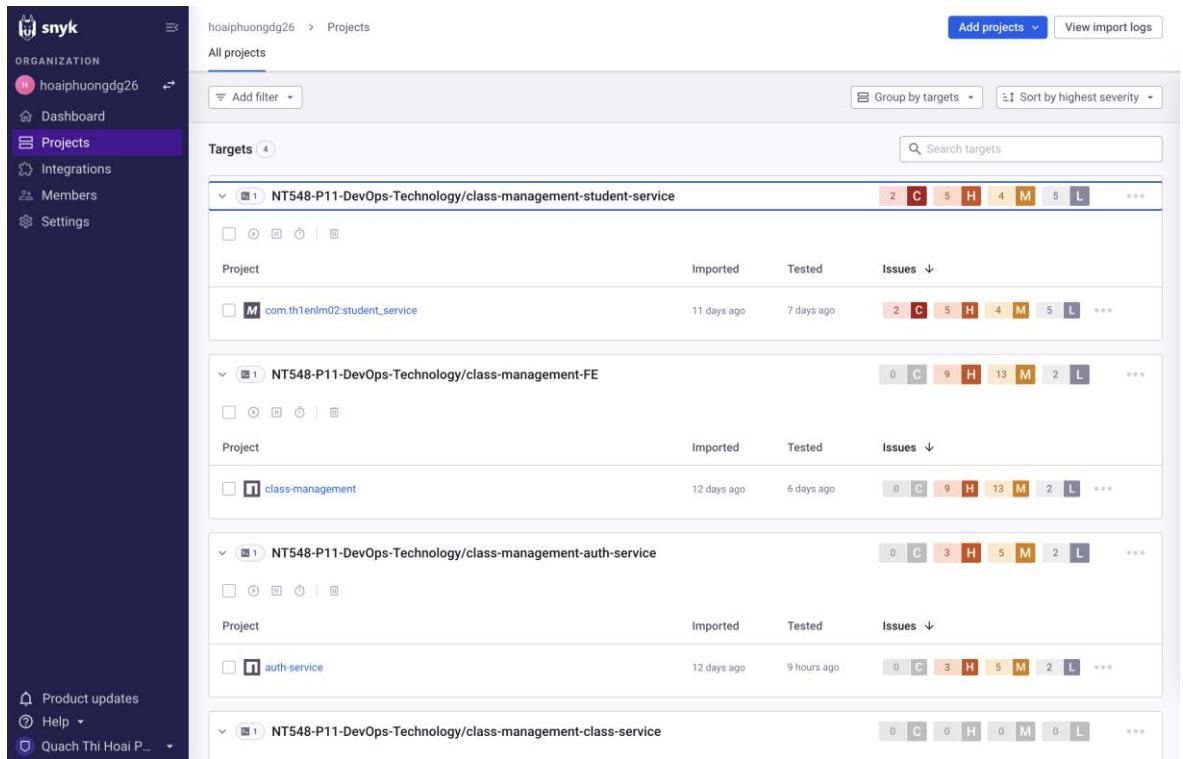
At the bottom right, there are page size and item count controls.

Hình 68. Thông tin các image đã được đánh tag trong một repository

4.3.3. Nhóm công cụ bảo mật

4.3.3.1. Cấu hình Snyk

Snyk là công cụ bảo mật giúp quét và phát hiện các lỗ hổng bảo mật trong code và dependencies.

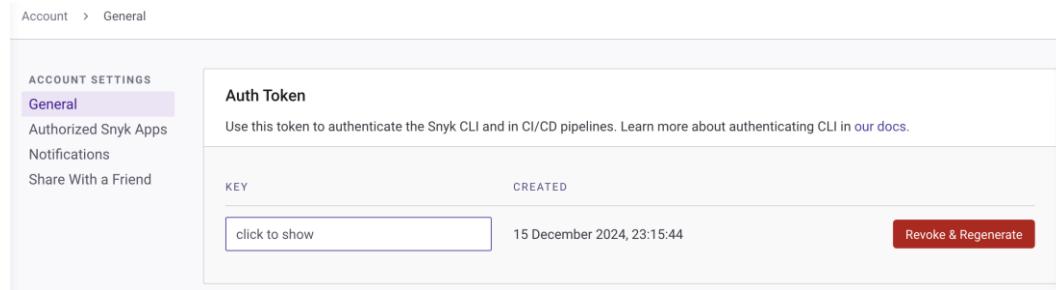


Hình 69. Snyk Dashboard

Trên màn hình dashboard của project "NT548-P11-DevOps-Technology" với các microservices (student-service, FE (Frontend), auth-service, class-service) hiển thị các chỉ số quan trọng như:

- C/H/M/L: Mức độ nghiêm trọng của lỗ hổng (Critical/High/Medium/Low).
- Imported: Thời điểm code được import vào Snyk.
- Tested: Lần quét gần nhất.

Để tích hợp Snyk với GitHub Actions cần tạo Snyk API Token và thêm token này vào GitHub Secrets.



Hình 70. Tạo Snyk API Token

Đối với việc tích hợp Snyk vào Github Action của các service với các ngôn ngữ khác nhau, cần xác định loại project để cài đặt đúng môi trường và dependencies tương ứng. Ở đây dựa vào các file đặc trưng của từng dự án.

```

jobs:
  determine-project-type:
    runs-on: ubuntu-latest
    outputs:
      project_type: ${{ steps.set-type.outputs.project_type }}
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Determine project type
        id: set-type
        run: |
          if [ -f "package.json" ]; then
            echo "project_type=node" >> $GITHUB_OUTPUT
          elif [ -f "requirements.txt" ]; then
            echo "project_type=python" >> $GITHUB_OUTPUT
          elif [ -f "composer.json" ]; then
            echo "project_type=php" >> $GITHUB_OUTPUT
          elif [ -f "pom.xml" ] || [ -f "build.gradle" ] || [ -f "build.gradle.kts" ]; then
            echo "project_type=java" >> $GITHUB_OUTPUT
          else
            echo "project_type=unknown" >> $GITHUB_OUTPUT
          fi

```

Hình 71. Xác định loại dự án

Các bước cơ bản để scan:

- Checkout code.
- Setup môi trường tương ứng (Node/Python/PHP/Java).
- Cài đặt dependencies.
- Cài đặt/cấu hình công cụ scan (Snyk).
- Chạy scan với các tham số phù hợp.
- Upload kết quả scan.

Đặc điểm chung:

- Cần token xác thực.

- Scan all projects trong repo.
- Set severity threshold (mức độ nghiêm trọng).
- Monitor mode để theo dõi liên tục.
- Error handling để workflow không fail khi có lỗi.

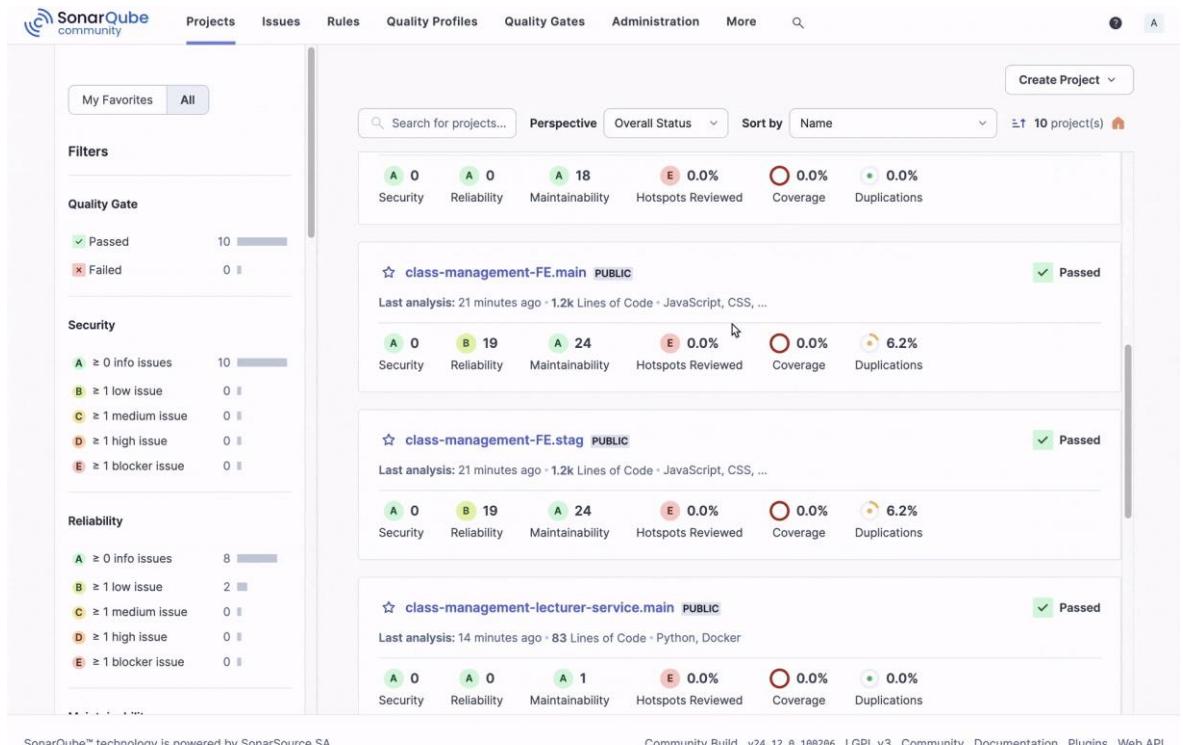
> <input checked="" type="checkbox"/> Pull snyk/snyk:node	4s
> <input checked="" type="checkbox"/> Pull snyk/snyk:python	2s
> <input checked="" type="checkbox"/> Pull snyk/snyk:php	3s
> <input checked="" type="checkbox"/> Checkout repo	0s
> <input checked="" type="checkbox"/> Setup Node.js	0s
<input type="checkbox"/> Setup Python	0s
<input type="checkbox"/> Setup PHP	0s
<input type="checkbox"/> Setup Java	0s
<input type="checkbox"/> Set up Java environment	0s
<input type="checkbox"/> Install Snyk CLI	0s
<input type="checkbox"/> Set Maven Wrapper Permissions	0s
> <input checked="" type="checkbox"/> Install dependencies (Node.js)	21s
<input type="checkbox"/> Install dependencies (Python)	0s
<input type="checkbox"/> Install dependencies (PHP)	0s
<input type="checkbox"/> Install dependencies (Maven)	0s
> <input checked="" type="checkbox"/> Run Snyk to check for vulnerabilities (Node.js)	8s
<input type="checkbox"/> Run Snyk to check for vulnerabilities (Python)	0s
<input type="checkbox"/> Run Snyk to check for vulnerabilities (PHP)	0s
<input type="checkbox"/> Run Snyk to check for vulnerabilities (Java)	0s
> <input checked="" type="checkbox"/> Post Setup Node.js	0s
> <input checked="" type="checkbox"/> Post Checkout repo	0s

Hình 72. Phân biệt để scan nhiều loại dự án khác nhau

Kết quả có thể thấy, đối với mỗi loại dự án, quy trình sẽ tự động chọn những step phù hợp, và bỏ qua những step không cần thiết.

4.3.3.2. Cấu hình SonarQube

SonarQube là công cụ phân tích chất lượng code tự động, phát hiện bugs, code smells, và security vulnerabilities.



Hình 73. Tích hợp SonarQube

Setup các thành phần token, url cần thiết để gọi tới SonarQube server.

```
sonar_scan:
  needs: security_scan
  runs-on: ubuntu-latest
  steps:
    - name: Checkout repo
      uses: actions/checkout@v3
      with:
        ref: ${{ inputs.branch || github.event.inputs.branch || github.ref_name }}
        fetch-depth: 0
        submodules: recursive

    - name: Set up SonarQube Scanner
      uses: sonarsource/sonarqube-scan-action@master
      env:
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}
      with:
        projectBaseDir: .
        args: >
          -Dsonar.projectKey=${{ github.event.repository.name }}.${{ github.ref_name }}
          -Dsonar.sources=.
          -Dsonar.exclusions=**/*.java,**/node_modules/**,**/*.json
          -Dsonar.language=js,ts
          -Dsonar.sourceEncoding=UTF-8
```

Hình 74. Cấu hình scan SonarQube

4.3.3.3. Cấu hình Trivy

Trivy là một công cụ quét lỗ hổng bảo mật tích hợp trực tiếp trong Harbor. Trivy được sử dụng để kiểm tra bảo mật cho các image trước khi triển khai.

The screenshot shows the Harbor interface under the 'Interrogation Services' section. On the left sidebar, 'Interrogation Services' is selected. The main area displays the 'Image Scanners' table with one entry for 'Trivy'. The 'Scanners' tab is active. The 'Trivy' row shows the following details:

- Name:** Default
- Endpoint:** http://trivy-adapter:8080
- Health:** Healthy
- Enabled:** Enabled
- Authorization:** None
- Vulnerability:** Supported
- SBOM:** Supported
- Description:** The Trivy scanner adapter

Below the table, there are sections for 'Scanner', 'Properties', and 'Capabilities' which list various MIME types and properties. A note at the bottom states: 'The default scanner has been installed. To install other scanners refer to the documentation.'

Hình 75. Tích hợp Trivy trong Harbor

The screenshot shows the Harbor interface under the 'Security Hub' section. The main area displays the following information:

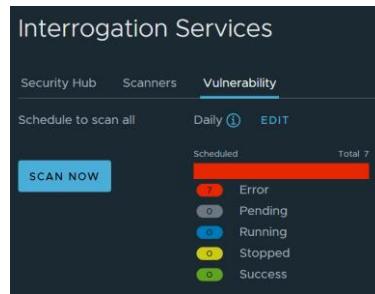
- 10 artifact(s), 10 scanned, 0 not scanned**
- Total Vulnerabilities:** 3008 total with 257 fixable
 - Critical: 10
 - High: 345
 - Medium: 1359
 - Low: 1290
 - n/a: 4
 - None: 0
- Top 5 Most Dangerous Artifacts:**

REPOSITORY NAME	DIGEST	VULNERABILITIES
Q.production/cl...	sha256:0fae3884	45%
Q.production/cl...	sha256:a4e6b575	38%
Q.staging/class...	sha256:d15e2e2d	38%
Q.production/cl...	sha256:13cb7ea2	32%
Q.staging/class...	sha256:6b2ff4a7	32%
- Top 5 Most Dangerous CVEs:**

CVE ID	SEVERITY	CVSS3	PACKAGE
Q,CVE-2023-45...	Critical	9.8	zliblg@11.2.13.d...
Q,CVE-2023-68...	Critical	9.8	libaom3@3.6.0-1...
Q,CVE-2023-45...	Critical	9.8	zliblg-dev@11.2...
Q,CVE-2017-9117	Low	9.8	libtiffxx6@4.5.0-...
Q,CVE-2017-9117	Low	9.8	libtiff-dev@4.5.0...

Hình 76. Giao diện các biểu đồ sau khi được scan bởi Trivy

Cấu hình lên lịch scan toàn bộ image mỗi ngày một lần.



Hình 77. Cấu hình lên lịch scan toàn bộ image

SCAN VULNERABILITY		Vulnerability	Severity	CVSS3	Package	Current version	Fixed in version	Listed in CVE Allowlist
>	CVE-2024-45590	High	ghsa: 7.5 rvd: 7.5 redhat: 7.5	body-parser	1.20.2		1.20.3	No
>	CVE-2024-4068	High	ghsa: 7.5 rvd: 7.5 redhat: 7.5	braces	3.0.2		3.0.3	No
>	CVE-2024-21538	High	ghsa: 7.5 rvd: 7.5 redhat: 4.4	cross-spawn	7.0.3		7.0.5, 6.0.6	No
>	CVE-2024-45296	High	ghsa: 7.5 rvd: 7.5 redhat: 5.3	path-to-regexp	0.1.7		1.9.0, 0.1.10, 8.0.0, 3.3.0, 6.3.0	No
>	CVE-2024-39338	High	ghsa: 7.5 rvd: 7.5 redhat: 7.5	axios	1.6.8		1.7.4	No
>	CVE-2024-21536	High	ghsa: 7.5 rvd: 7.5 redhat: 7.5	http-proxy-middleware	2.0.6		2.0.7, 3.0.3	No
>	CVE-2021-3803	High	ghsa: 7.5 rvd: 7.5 redhat: 7.5	nth-check	1.0.2		2.0.1	No
>	CVE-2024-47068	High	ghsa: 6.4 rvd: 6.1 redhat: 6.4	rollup	2.79.1		3.29.5, 4.22.4, 2.79.2	No
>	CVE-2024-29180	High	ghsa: 7.4 rvd: 7.4 redhat: 7.4	webpack-dev-middleware	5.3.3		7.1.0, 6.1.2, 5.3.4	No
>	CVE-2024-37890	High	ghsa: 7.5 rvd: 5.9 redhat: 7.5	ws	7.5.9		5.2.4, 6.2.3, 7.5.10, 8.17.1	No

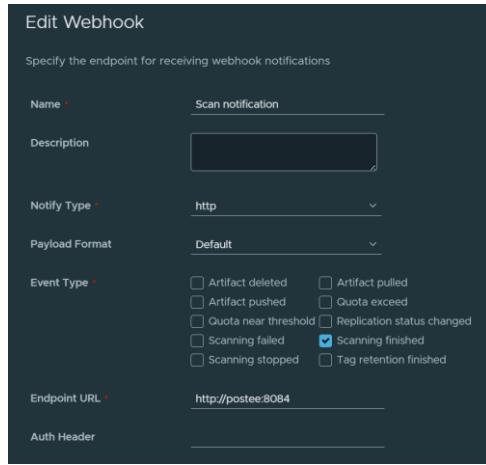
Hình 78. Thông tin các lỗ hổng được scan bởi Trivy

Tích hợp thông báo qua Telegram: Cấu hình thông báo qua Telegram mỗi khi có image mới được push lên Harbor và được scan thành công bởi Trivy. Mục đích để thông tin đến dev sớm nhận ra các lỗ hổng bảo mật thông qua file .csv mà sửa lỗi, cập nhật phiên bản kịp thời.

- Tích hợp webhook: Sử dụng tính năng webhook của Harbor để phát hiện sự kiện quét bảo mật (Scanning finished). Trong đó, sử dụng một tool là Postee, nó sẽ là tool trung gian và nhận đầu vào là thông tin scan và sẽ xử lý sau đó gửi message đến Telegram.

Summary	Repositories	Members	Labels	Scanner	P2P Preheat	Policy	Robot Accounts	Webhooks	Logs	Configuration
Webhooks										
+ NEW WEBHOOK ACTION ▾										
Name	Enabled	Notify Type	Payload Format	Endpoint URL	Event types	Created	Description			
Scan notification	Enabled	http	Default	http://postee:8084	Scanning finished	11/6/24, 2:01 PM		Page size	15	1 - 1 of 1 items

Hình 79. Cấu hình webhook trên Harbor



Hình 80. Chọn sự kiện sẽ trigger webhook

- Ở Postee, cấu hình Bash Script để xử lý và gửi nội dung scan nhận được, trong đó chú ý tới biến \$POSTEE_EVENT là biến nhận được từ output của template và từ đây sẽ trích xuất thông tin để gọi API của Harbor và Telegram để gửi message và file .csv đến Telegram:

```

actions:
- name: stdout
  type: stdout
  enable: true

- name: telegram
  type: exec
  enable: true
  exec-script: |
    #!/bin/sh
    TELEGRAM_BOT_TOKEN=[REDACTED]
    TELEGRAM_CHAT_ID=[REDACTED]
    TIME="10"
    TELEGRAM_API_URL="https://api.telegram.org/bot$TELEGRAM_BOT_TOKEN/sendDocument"
    HARBOR_URL=[REDACTED]
    HARBOR_AUTH=[REDACTED]

    FULL_TEXT=$POSTEE_EVENT
    TEXT=$(echo $FULL_TEXT | jq -r '.message')
    PROJECT_NAME=$(echo $FULL_TEXT | jq -r '.project')
    REPO_NAME=$(echo $FULL_TEXT | jq -r '.repo')
    TAG_NAME=$(echo $FULL_TEXT | jq -r '.tag')

    PROJECT_ID=$(curl -X 'GET' \
      "$HARBOR_URL/api/v2.0/projects/$PROJECT_NAME" \
      -H 'accept: application/json' \
      -H 'X-Is-Resource-Name: false' \
      -H "authorization: Basic $HARBOR_AUTH" | jq -r '.project_id')
    echo "Project ID: $PROJECT_ID"

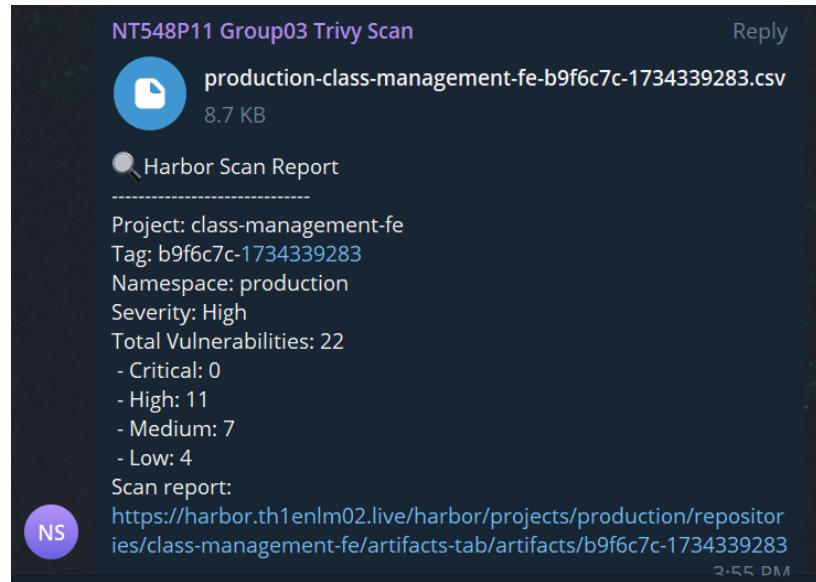
    EXECUTION_ID=$(curl -X 'POST' \
      "$HARBOR_URL/api/v2.0/export/cve" \
      -H 'accept: application/json' \
      -H 'X-Scan-Data-Type: application/vnd.security.vulnerability.report; version=1.1' \
      -H "authorization: Basic $HARBOR_AUTH" \
      -H "Content-Type: application/json" \
      -d '{
        "projects": [
          "$PROJECT_ID"
        ],
        "repositories": """$REPO_NAME""",
        "tags": """$TAG_NAME"""
      }' | jq -r '.id')
    echo "Execution ID: $EXECUTION_ID"

    FILE_PRESENT="false"
    while [ "$FILE_PRESENT" != "true" ]; do

```

Hình 81. Cấu hình Bash Script cho Postee

- Kết quả được gửi từ bot của Telegram nhận được từ Postee trong đó chứa nội dung như đã định dạng kèm file csv tương ứng:



Hình 82. Nội dung message được gửi đến Telegram chứa thông tin scan

Preview 'production-class-management-fe-b9f6c7c-1734339283.csv'									
Repository	Artifact Digest	CVE	Package	Current Version	Fixed in version	Severity	CWE Ids	Additional Data	Scanner
production/class-manag	sha256:b930a834;	CVE-2024-39338	axios	1.6.8	1.7.4	High	CWE-918	{"CVSS": {"nvd": "Trivy"}	
production/class-manag	sha256:b930a834;	CVE-2024-45590	body-parser	1.20.2	1.20.3	High	CWE-405	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-4068	braces	3.0.2	3.0.3	High	CWE-1050,CWE-	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-47764	cookie	0.5.0	0.7.0	Low	CWE-74	{"CVSS": {"redha": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-21538	cross-spawn	7.0.3	7.0.5, 6.0.6	High	CWE-1333	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-33883	ejs	3.1.9	3.1.10	Medium	CWE-693	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-43796	express	4.18.3	4.20.0, 5.0.0	Low	CWE-79	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-29041	express	4.18.3	4.19.2, 5.0.0-beta.3	Medium	CWE-1286,CWE-	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-21536	http-proxy-middleware	2.0.6	2.0.7, 3.0.3	High	CWE-400	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-4067	micromatch	4.0.5	4.0.8	Medium	CWE-1333	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-55565	nanooid	3.3.7	5.0.9, 3.3.8	Medium	CWE-835	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2021-3803	nth-check	1.0.2	2.0.1	High	CWE-1333	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-45296	path-to-regexp	0.1.7	1.9.0, 0.1.10, 8.0.0, 3.3.0, 6.3.0	High	CWE-1333	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-52798	path-to-regexp	0.1.7	0.1.12	Medium	CWE-1333	{"CVSS": {"redha": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2023-44270	postcss	7.0.39	8.4.31	Medium	CWE-74	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-47068	rollup	2.79.1	3.29.5, 4.22.4, 2.79.2	High	CWE-79	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-43799	send	0.18.0	0.19.0	Low	CWE-79	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-43800	serve-static	1.15.0	1.16.0, 2.1.0	Low	CWE-79	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-43788	webpack	5.90.3	5.94.0	Medium	CWE-79	{"CVSS": {"nvd": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-29180	webpack-dev-m	5.3.3	7.1.0, 6.1.2, 5.3.4	High	CWE-22	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-37890	ws	7.5.9	5.2.4, 6.2.3, 7.5.10, 8.17.1	High	CWE-476	{"CVSS": {"ghsa": "Trivy"}}	
production/class-manag	sha256:b930a834;	CVE-2024-37890	ws	8.16.0	5.2.4, 6.2.3, 7.5.10, 8.17.1	High	CWE-476	{"CVSS": {"ghsa": "Trivy"}}	

Hình 83. Nội dung của file .csv chứa thông tin chi tiết kết quả scan bởi Trivy

4.3.3.4. Cấu hình Vault

Vault đóng vai trò là nơi lưu trữ secret như thông tin xác thực cơ sở dữ liệu, API URL và các thông tin nhạy cảm khác.

Định nghĩa các policy để giới hạn quyền truy cập secret.

Tạo role tương ứng với mỗi nhóm dịch vụ hoặc môi trường (staging, production).

Secrets / nt548p11 / production

nt548p11 version 2

Secrets Configuration

production/ Search Create secret +

auth/

class/

lecturer/

student/

1-4 of 4 < 1 >

Hình 84. Quản lý thông tin lưu trữ trên Vault

Secrets / nt548p11 / production / auth / service

production/auth/service

Key	Value	Actions
AUTH_SERVICE_DATABASE	*****	
AUTH_SERVICE_HOST	*****	
AUTH_SERVICE_PASSWORD	*****	
AUTH_SERVICE_PORT	*****	
AUTH_SERVICE_USER	*****	
BASE_URL_FE	*****	

Hình 85. Các biến secret được lưu trữ trên Vault

Qua annotation trong Kubernetes manifest, Vault Agent được cấu hình để tự động lấy các secret từ Vault, sau đó inject chúng vào container dưới dạng tệp hoặc biến môi trường.

```

manifests > overlays > staging > vault-patches.yml
Nguyen.tk, 2 weeks ago | 2 authors (Nguyen.tk and one other)
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: auth-deployment
5  spec:
6    template:
7      metadata:
8        annotations:
9          vault.hashicorp.com/agent-inject: "true"
10         vault.hashicorp.com/role: "readonly"
11         vault.hashicorp.com/template-static-secret-render-interval: "10s"
12         vault.hashicorp.com/agent-inject-secret-auth: "nt548p11/data/staging/auth/service"
13         vault.hashicorp.com/agent-inject-template-auth: |
14           {{- with secret "nt548p11/data/staging/auth/service" -}}
15             {{- range $key, $value := .Data.data -}}
16               export {{ $key }}="{{ $value }}" \n
17             {{- end }}
18           {{- end }}
19     spec:
20       serviceAccountName: vault-sa
21       containers:
22         - name: class-management-auth-service-container
23           command: ["/bin/sh", "-c"]
24           args: ["source /vault/secrets/auth && node server.js"]
25

```

Hình 86. Vault agent inject secret vào các pod thông qua annotations.

Vault đóng vai trò trung tâm trong việc bảo mật và quản lý thông tin nhạy cảm, giúp hệ thống trở nên an toàn và dễ quản lý hơn. Với sự tích hợp chặt chẽ cùng Kubernetes, Vault không chỉ bảo mật mà còn hỗ trợ tự động hóa trong môi trường triển khai phức tạp.

4.3.4. Nhóm công cụ giám sát

4.3.4.1. Cấu hình Prometheus

Prometheus được cấu hình làm công cụ giám sát chính trong hệ thống. Công cụ này thu thập và lưu trữ các chỉ số từ các dịch vụ, node, hoặc cluster Kubernetes, cung cấp cơ sở dữ liệu time-series cho việc theo dõi và cảnh báo hiệu suất hệ thống.

Prometheus được triển khai bằng cách sử dụng FluxCD kết hợp với Helm:

```

helm > prometheus > repository.yml
dangnguyen03, 3 weeks ago | 1 author (dangnguyen03)
1  apiVersion: source.toolkit.fluxcd.io/v1
2  kind: HelmRepository
3  metadata:
4    name: prometheus-repo
5  spec:
6    interval: 10m
7    url: https://prometheus-community.github.io/helm-charts

```

Hình 87. Prometheus Helm repository

Cấu hình HelmRelease để triển khai Prometheus với các giá trị ghi đè (overrides) tùy chỉnh.

```
helm > prometheus > release.yaml
dangnguyen03, 3 weeks ago | 1 author (dangnguyen03)
1 apiVersion: helm.toolkit.fluxcd.io/v2
2 kind: HelmRelease
3 metadata:
4   name: prometheus
5   namespace: prometheus
6 spec:
7   interval: 10m
8   releaseName: prometheus
9   chart:
10     spec:
11       chart: kube-prometheus-stack
12       version: '65.0.0'
13       sourceRef:
14         kind: HelmRepository
15         name: prometheus-repo
16     valuesFrom:
17       - kind: ConfigMap
18         name: prometheus-values
19         valuesKey: values.yml
```

Hình 88. HelmRelease để triển khai Prometheus

Để có thể sử dụng domain truy cập vào Prometheus trên cụm EKS, tiến hành cấu hình Ingress cho Prometheus, trong đó định nghĩa rule khi truy cập vào domain đang mapping cho Prometheus trên cụm thì sẽ trả vào service có tên là prometheus-operated với port là 9090, nó là một service do Prometheus Operator tạo ra để quản lý và duy trì các pod của Prometheus.

```
helm > prometheus > ingress.yaml
Nguyentk, 2 weeks ago | 3 authors (You and others)
1 apiVersion: networking.k8s.io/v1
2 kind: Ingress
3 metadata:
4   name: prometheus-ingress
5   labels:
6     name: prometheus-ingress
7   annotations:
8     ingress.kubernetes.io/auth-type: "basic"
9     ingress.kubernetes.io/auth-secret: "prometheus-basic-auth"
10  spec:
11    ingressClassName: nginx
12    rules:
13      - host: prometheus.thienlm02.live
14        http:
15          paths:
16            - pathType: Prefix
17              path: "/"
18              backend:
19                service:
20                  name: prometheus-operated
21                  port:
22                    number: 9090
```

Hình 89. Ingress cho Prometheus

Kiểm tra thông tin và trạng thái các resource của Prometheus được tạo trên cụm EKS sau khi triển khai.

```
> kubectl -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP           NODE      NOMINATED NODE   READINESS G
ATES
alertmanager-prometheus-stack-kube-prom-alertmanager-0   2/2    Running   26 (9m52s ago)  5d18h  10.42.0.204  ip-192-168-110-10  <none>        <none>
prometheus-prometheus-stack-kube-prom-prometheus-0       2/2    Running   24 (9m26s ago)  5d18h  10.42.2.8   ip-192-168-110-12  <none>        <none>
prometheus-stack-kube-prom-operator-69c57f884b-pw4lf      1/1    Running   14 (9m23s ago)  5d18h  10.42.1.126  ip-192-168-110-11  <none>        <none>
prometheus-stack-kube-state-metrics-6f44fb6dc6-mcjk2      1/1    Running   15 (9m52s ago)  5d18h  10.42.0.202  ip-192-168-110-10  <none>        <none>
prometheus-stack-prometheus-node-exporter-tbdc9          1/1    Running   12 (9m23s ago)  5d18h  192.168.110.11 ip-192-168-110-11  <none>        <none>
prometheus-stack-prometheus-node-exporter-vrfqb          1/1    Running   12 (9m26s ago)  5d18h  192.168.110.12 ip-192-168-110-12  <none>        <none>
prometheus-stack-prometheus-node-exporter-vznm1          1/1    Running   15 (9m52s ago)  5d18h  192.168.110.10 ip-192-168-110-10  <none>        <none>
> kubectl get svc -o wide
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE   SELECTOR
alertmanager-operated   ClusterIP  None         <none>        9093/TCP,9094/TCP,9094/UDP  5d18h  app.kubernetes.io/name=alertmanager
prometheus-operated    ClusterIP  None         <none>        9090/TCP               5d18h  app.kubernetes.io/name=prometheus
prometheus-stack-kube-prom-alertmanager   ClusterIP  10.43.14.4  <none>        9093/TCP,8080/TCP          5d18h  alertmanager-prometheus-stack-kube-prom-alertmanager
prometheus-stack-kube-prom-operator   ClusterIP  10.43.72.73 <none>        443/TCP                  5d18h  app=kube-prometheus-stack-kube-prom-operator,release=prometheus-stack
prometheus-stack-kube-prom-prometheus   ClusterIP  10.43.38.235 <none>        9090/TCP,8080/TCP          5d18h  app.kubernetes.io/name=prometheus-stack-kube-prom-prometheus
prometheus-stack-kube-state-metrics   ClusterIP  10.43.146.127 <none>        8080/TCP                 5d18h  app.kubernetes.io/instance=prometheus-stack-k,app.kubernetes.io/name=kube-state-metrics
prometheus-stack-prometheus-node-exporter   ClusterIP  10.43.42.72  <none>        9100/TCP                 5d18h  app.kubernetes.io/instance=prometheus-stack-k,app.kubernetes.io/name=prometheus-node-exporter
> kubectl get ing -o wide
NAME          CLASS      HOSTS          ADDRESS          PORTS   AGE
prometheus-ingress  traefik   prometheus-cluster.thienlm02.live  192.168.110.10,192.168.110.11,192.168.110.12  80      5d12h
```

Hình 90. Kiểm tra thông tin các resource của Prometheus

Truy cập vào URL đã được cấu hình để kiểm tra trạng thái kết nối đến các Node Exporter.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.42.0.204:9093/metrics	UP	container="alertmanager", endpoint="http-web", instance="10.42.0.204:9093", job="prometheus-stack-kube-prom-alertmanager", namespace="monitoring", pod="alertmanager-prometheus-stack-kube-prom-alertmanager-0", service="prometheus-stack-kube-prom-alertmanager"	5.77s ago	4.124ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.42.0.204:8080/metrics	UP	container="config-reloader", endpoint="reloader-web", instance="10.42.0.204:8080", job="prometheus-stack-kube-prom-alertmanager", namespace="monitoring", pod="alertmanager-prometheus-stack-kube-prom-alertmanager-0", service="prometheus-stack-kube-prom-alertmanager"	19.895s ago	2.733ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://192.168.110.10:6443/metrics	UP	endpoint="https", instance="192.168.110.10:6443", job="spinerver", namespace="default", service="kubernetes"	24.862s ago	137.118ms	

Hình 91. Kiểm tra kết nối đến các Node Exporter

4.3.4.2. Cấu hình Loki

Triển khai Loki được thực hiện tương tự như Prometheus, sử dụng FluxCD và Helm. Bên cạnh đó, để thu thập log từ các container trong Kubernetes cluster, Promtail được triển khai cùng với Loki.

```

helm > loki-promtail > repository.yml
dangnguyen03, 3 weeks ago | 1 author (dangnguyen03)
1 apiVersion: source.toolkit.fluxcd.io/v1
2 kind: HelmRepository
3 metadata:
4   name: grafana-repo
5   namespace: loki
6 spec:
7   interval: 10m
8   url: https://grafana.github.io/helm-charts
9

```

Hình 92. Loki Helm repository

Cấu hình HelmRelease để triển khai Loki.

```

helm > loki-promtail > loki > release.yml
dangnguyen03, 3 weeks ago | 1 author (dangnguyen03)
1 apiVersion: helm.toolkit.fluxcd.io/v2
2 kind: HelmRelease
3 metadata:
4   name: loki
5   namespace: loki
6 spec:
7   interval: 10m
8   releaseName: loki
9   chart:
10     spec:
11       chart: loki
12       version: '6.16.0'
13       sourceRef:
14         kind: HelmRepository
15         name: grafana-repo
16   valuesFrom:
17     - kind: ConfigMap
18       name: loki-values
19       valuesKey: values.yml

```

Hình 93. HelmRelease để triển khai Loki

Cấu hình triển khai Promtail để gửi log về Loki.

```

helm > loki-promtail > promtail > release.yml
dangnguyen03, 3 weeks ago | 1 author (dangnguyen03)
1 apiVersion: helm.toolkit.fluxcd.io/v2
2 kind: HelmRelease
3 metadata:
4   name: promtail
5   namespace: loki
6 spec:
7   interval: 10m
8   releaseName: promtail
9   chart:
10     spec:
11       chart: promtail
12       sourceRef:
13         kind: HelmRepository
14         name: grafana-repo
15   valuesFrom:
16     - kind: ConfigMap
17       name: promtail-values
18       valuesKey: values.yml
19

```

Hình 94. HelmRelease để triển khai Promtail

Cấu hình thông tin value sẽ được sử dụng để ghi đè cấu hình mặc định khi triển khai Loki thông qua Helm, nhằm đáp ứng các yêu cầu về lưu trữ, truy vấn, và quản lý log.

- schemaConfig: Cấu hình schema cho dữ liệu log:
 - store: Loại lưu trữ chính được sử dụng (ở đây là TSDB).
 - object_store: Định nghĩa hệ thống lưu trữ đối tượng (ở đây là S3).
 - index: Tiền tố và chu kỳ làm mới cho chỉ mục.
- storage:
 - type: Kiểu lưu trữ (S3).
 - s3: Kết nối đến MinIO, với thông tin chứng thực và endpoint tùy chỉnh.

```
helm > loki-promtail > loki > values.yaml
dangnguyen03, 2 weeks ago | 3 authors (Nguyen.tk and others)
1 ---
2 loki:
3   auth_enabled: false
4   commonConfig:
5     replication_factor: 1
6     path_prefix: /tmp/loki
7   schemaConfig:
8     configs:
9       - from: 2024-04-01
10      store: tsdb
11      object_store: s3
12      schema: v13
13      index:
14        prefix: loki_index_
15        period: 24h
16   storage:
17     type: s3
18     bucketNames:
19       chunks: "loki"
20       ruler: "loki"
21       admin: "admin"
22     s3:
23       s3: "s3://1LYmzSeaQxPCNj1Soooy:NrvapaDsVLxeDEV5R72CBvUT9aMV6hfs5IHj0ncn@10.0.30.10:9000/loki"
24       insecure: true
25       s3ForcePathStyle: true
26   ingestor:
27     chunk_encoding: snappy
28   tracing:
29     enabled: true
30   querier:
31     # Default is 4, if you have enough memory and CPU you can increase, reduce if OOMing
32     max_concurrent: 2
```

Hình 95. Cấu hình value cho Loki

Cấu hình Promtail để gửi log đến Loki thông qua endpoint, chỉ định địa chỉ API Gateway của Loki, nơi Promtail sẽ đẩy dữ liệu log. Sau khi triển khai, Promtail sẽ hoạt động như một agent để đảm bảo dữ liệu log được đồng bộ hóa và phân tích dễ dàng.

```

helm > loki-promtail > promtail > values.yml
      You, 3 weeks ago | 1 author (You)
  1   config:
  2     logLevel: info
  3     clients:
  4       - url: http://loki-gateway/loki/api/v1/push
  5

```

Hình 96. Cấu hình value cho Promtail

Để có thể sử dụng URL truy cập vào Loki trên cụm EKS, tiến hành cấu hình Ingress cho Loki, trong đó định nghĩa rule khi truy cập vào domain đang mapping cho Loki trên cụm thì sẽ trả vào service có tên là loki-gateway với port là 80, service này cung cấp một API HTTP để nhận dữ liệu log từ các client như Promtail.

```

helm > loki-promtail > loki > ingress.yml
      Nguyen.tk, 2 weeks ago | 3 authors (You and others)
  1   apiVersion: networking.k8s.io/v1
  2   kind: Ingress
  3   metadata:
  4     name: loki-ingress
  5     labels:
  6       name: loki-ingress
  7     annotations:
  8       ingress.kubernetes.io/auth-type: "basic"
  9       ingress.kubernetes.io/auth-secret: "loki-basic-auth"
 10   spec:
 11     ingressClassName: nginx
 12     rules:
 13       - host: loki.thienlm02.live
 14         http:
 15           paths:
 16             - pathType: Prefix
 17               path: "/"
 18               backend:
 19                 service:
 20                   name: loki-gateway
 21                   port:
 22                     number: 80
 23

```

Hình 97. Ingress cho Loki

Kiểm tra thông tin và trạng thái các resource của Loki và Promtail được tạo trên cụm EKS sau khi triển khai.

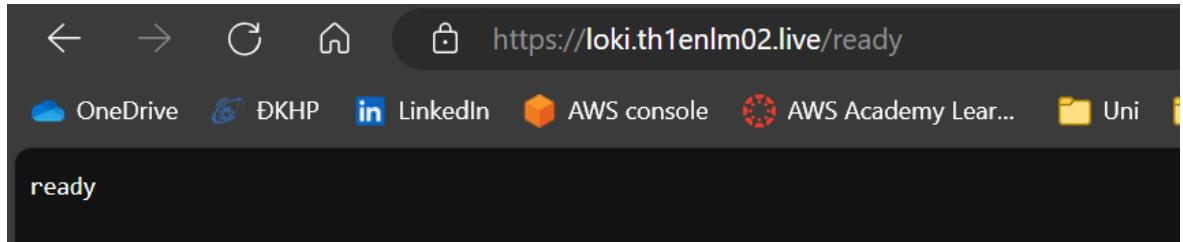
```

> kubectl -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP           NODE      NOMINATED NODE   READINESS GATES
loki-0         2/2    Running   10 (6m11s ago) 2d1h  10.42.0.219  ip-192-168-110-10  <none>    <none>
loki-1         2/2    Running   10 (5m52s ago) 2d1h  10.42.2.14   ip-192-168-110-12  <none>    <none>
loki-2         2/2    Running   10 (5m51s ago) 2d1h  10.42.1.145  ip-192-168-110-11  <none>    <none>
loki-canary-h94cb 1/1    Running   5 (5m52s ago) 2d1h  10.42.2.15   ip-192-168-110-12  <none>    <none>
loki-canary-hdkx8 1/1    Running   5 (6m11s ago) 2d1h  10.42.0.216  ip-192-168-110-10  <none>    <none>
loki-canary-jqvk  1/1    Running   5 (5m51s ago) 2d1h  10.42.1.142  ip-192-168-110-11  <none>    <none>
loki-gateway-8d764c899-wrqss 1/1    Running   10 (5m43s ago) 2d1h  10.42.2.12   ip-192-168-110-12  <none>    <none>
loki-minio-0       1/1    Running   5 (5m51s ago) 2d1h  10.42.1.141  ip-192-168-110-11  <none>    <none>
loki-chunks-cache-0 2/2    Running   5 (5m51s ago) 2d1h  10.42.2.16   ip-192-168-110-12  <none>    <none>
promtail-57qf2    1/1    Running   5 (5m51s ago) 2d1h  10.42.1.146  ip-192-168-110-11  <none>    <none>
promtail-d94xq    1/1    Running   5 (6m11s ago) 2d1h  10.42.0.212  ip-192-168-110-10  <none>    <none>
promtail-f1j8c    1/1    Running   5 (5m52s ago) 2d1h  10.42.2.10   ip-192-168-110-12  <none>    <none>
> kubectl get svc -o wide
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE   SELECTOR
loki             ClusterIP  10.43.157.71 <none>        3100/TCP,9095/TCP 2d1h  app.kubernetes.io/component=single-binary,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-canary      ClusterIP  10.43.131.12 <none>        3500/TCP   2d1h  app.kubernetes.io/component=canary,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-chunks-cache ClusterIP None          <none>        11211/TCP,9150/TCP 2d1h  app.kubernetes.io/component=memcached-chunks-cache,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-gateway     ClusterIP  10.43.113.206 <none>       80/TCP     2d1h  app.kubernetes.io/component=gateway,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-headless    ClusterIP None          <none>        3100/TCP   2d1h  app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
loki-memberlist  ClusterIP None          <none>        7946/TCP   2d1h  app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki,app.kubernetes.io/part-of=memberlist
loki-minio       ClusterIP  10.43.192.199 <none>       9000/TCP   2d1h  app-minio,release=loki
loki-minio-console ClusterIP 10.43.250.152 <none>       9001/TCP   2d1h  app-minio,release=loki
loki-minio-svc   ClusterIP None          <none>        9000/TCP   2d1h  app-minio,release=loki
loki-results-cache ClusterIP None          <none>       11211/TCP,9150/TCP 2d1h  app.kubernetes.io/component=memcached-results-cache,app.kubernetes.io/instance=loki,app.kubernetes.io/name=loki
> kubectl get ing -o wide
NAME          CLASS      HOSTS          ADDRESS          PORTS   AGE
loki-ingress  traefik   loki-cluster.thienlm02.live  192.168.110.10,192.168.110.11,192.168.110.12  80      4d23h

```

Hình 98. Kiểm tra thông tin các resource của Loki và Promtail

Truy cập vào URL sau để kiểm tra trạng thái sẵn sàng của Loki.



Hình 99. Kiểm tra trạng thái sẵn sàng của Loki

4.3.4.3. Cấu hình Grafana

Grafana được triển khai thông qua cấu hình value của Prometheus bằng cách bật tùy chọn Grafana.

```

helm > prometheus > values.yaml
dangnguyen03, 2 weeks ago | 2 authors (dangnguyen03 and one other)
1  > grafana:
2  |   enabled: true

```

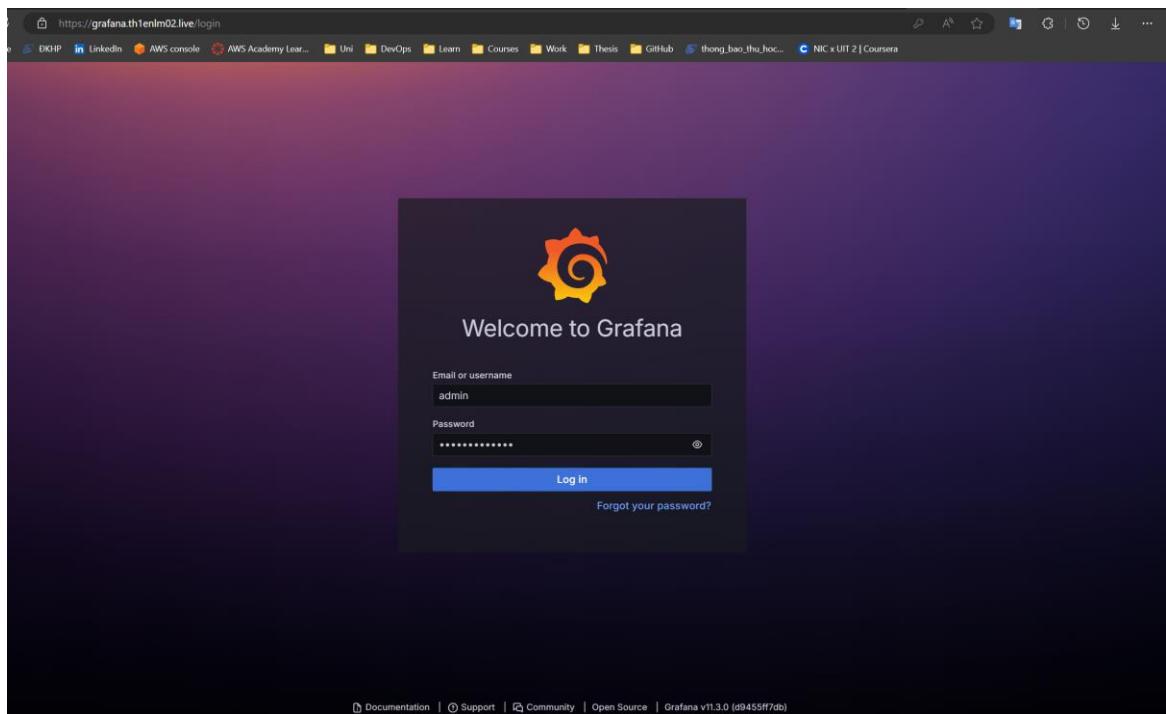
Hình 100. Bật cấu hình Grafana

Cấu hình Ingress cho Grafana để có thể truy cập thông qua URL.

```
helm > grafana > ingress.yaml
dangnguyen03, 2 weeks ago | 1 author (dangnguyen03)
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: grafana-ingress
5    labels:
6      name: grafana-ingress
7    namespace: prometheus
8    annotations:
9      ingress.kubernetes.io/auth-type: "basic"
10   spec:
11     ingressClassName: nginx
12     rules:
13       - host: grafana.th1enlm02.live
14         http:
15           paths:
16             - pathType: Prefix
17               path: "/"
18               backend:
19                 service:
20                   name: prometheus-grafana
21                   port:
22                     number: 80
```

Hình 101. Ingress cho Grafana

Truy cập vào Grafana web UI.



Hình 102. Truy cập vào Grafana web UI

Kết nối đến các data source để lấy dữ liệu từ các nguồn như Prometheus, Loki.

The screenshot shows the Grafana interface for managing data sources. At the top right is a blue button labeled '+ Add new data source'. Below it is a search bar and a dropdown menu set to 'Sort by A-Z'. The main area lists five data sources:

- influxdb**: InfluxDB | http://localhost:8086. Buttons: Build a dashboard, Explore.
- Loki Cluster Logs**: Loki | https://loki-cluster.th1enlm02.live/. Buttons: Build a dashboard, Explore.
- Loki Server Logs**: Loki | https://loki.th1enlm02.live/. Buttons: Build a dashboard, Explore.
- Prometheus Cluster Metrics**: Prometheus | https://prometheus-cluster.th1enlm02.live/. Buttons: Build a dashboard, Explore.
- Prometheus Server Metrics**: Prometheus | https://prometheus.th1enlm02.live/. Buttons: Build a dashboard, Explore.

Hình 103. Grafana kết nối đến các data source

Sau khi kết nối thành công đến các data source, thêm các dashboard cần thiết, có thể tùy chỉnh thông tin và các giá trị để hiển thị giao diện theo ý muốn.

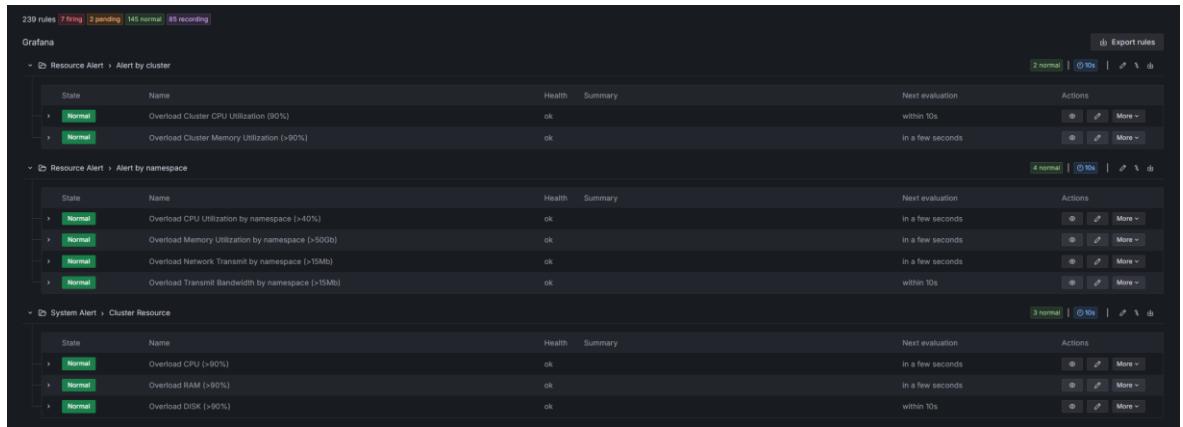


Hình 104. Dashboard cho Prometheus



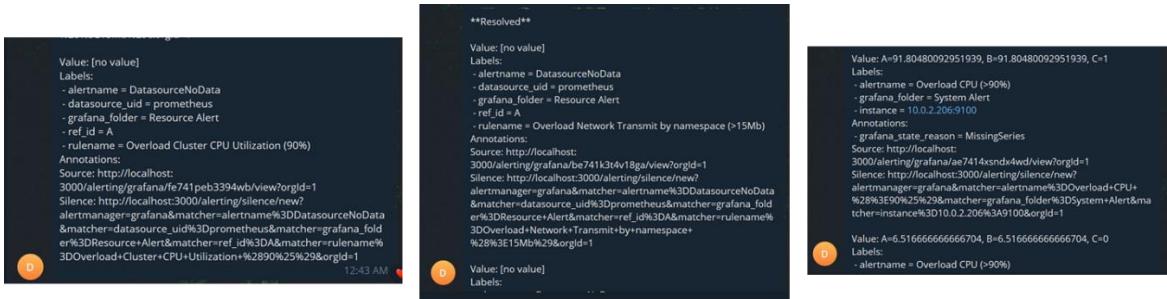
Hình 105. Dashboard cho Loki

Grafana cung cấp tính năng cảnh báo (alerting) mạnh mẽ giúp giám sát dữ liệu theo thời gian thực và gửi thông báo khi các điều kiện xác định được đáp ứng. Cấu hình alert cho Grafana để gửi thông báo đến Telegram khi đạt đến các ngưỡng RAM, CPU, Disk quy định thu thập được từ Prometheus.



Hình 106. Cấu hình alert cho Grafana

Nội dung alert message tương ứng với từng phạm vi: Cluster, Namespace, Resource.



Hình 107. Nội dung alert message từ Grafana

4.4. Triển khai quy trình CI/CD

4.4.1. Quy trình CI

Cấu hình GitHub Actions:



Hình 108. Quy trình CI

GitHub Actions sẽ đi qua 4 luồng chính bao gồm determine-project-type, security_scan, sonar_scan và build_and_push.

Name	Visibility	Last updated
HARBOR_PASSWORD	Public repositories	2 weeks ago
HARBOR_REGISTRY	Public repositories	2 weeks ago
HARBOR_USERNAME	Public repositories	2 weeks ago
SNYK_TOKEN	Public repositories	2 weeks ago
SONAR_HOST_URL	Public repositories	2 weeks ago
SONAR_TOKEN	Public repositories	2 weeks ago

Hình 109. Quản lý secrets của Github Actions

Để có thể tái sử dụng lại workflow cho tất cả các service, sử dụng workflow tại service “FE” làm template workflow, vì thế cấu hình workflow call tại service này.

```
 20      - stag
 21  workflow_call:
 22    inputs:
 23      branch:
 24        description: 'Branch to run workflow on'
 25        type: string
 26        required: false
 27        default: ''
 28    secrets:
 29      SONAR_TOKEN:
 30        required: true
 31      SONAR_HOST_URL:
 32        required: true
 33      HARBOR_USERNAME:
 34        required: true
 35      HARBOR_PASSWORD:
 36        required: true
 37      HARBOR_REGISTRY:
 38        required: true
 39      SNYK_TOKEN:
 40        required: true
```

Hình 110. Cấu hình workflow call

Khi đó, với các service khác, chỉ cần gọi lại workflow template thì các quy trình tương tự sẽ được gọi.

```
name: CI Pipeline
on:
  push:
    branches:
      - main
      - stag
  pull_request:
    branches:
      - main
      - stag
  workflow_dispatch:
    inputs:
      branch:
        description: 'Branch to run workflow on'
        required: true
        default: 'stag'
        type: choice
        options:
          - main
          - stag

jobs:
  call-reusable-workflow:
    uses: NT548-P11-DevOps-Technology/class-management-FE/.github/workflows/main.yml@main
    with:
      branch: ${{ github.event.inputs.branch || github.ref_name }}
      secrets: inherit
```

Hình 111. Gọi lại workflow template

Trong quá trình build và push image lên Harbor, cấu hình các biến môi trường để lưu mã commit, timestamp để lưu tag của các image theo đúng định dạng mong muốn.

```

- name: Set environment variables
  id: set-env
  run: |
    echo "REPO_NAME=${GITHUB_REPOSITORY##*/}" >> $GITHUB_ENV
    if [[ "$GITHUB_REF" == "refs/heads/main" ]]; then
      echo "ENVIRONMENT=production" >> $GITHUB_ENV
    elif [[ "${GITHUB_REF}" == "refs/heads/stag" ]]; then
      echo "ENVIRONMENT=staging" >> $GITHUB_ENV
    fi
    echo "COMMIT_SHA=$(git rev-parse --short HEAD)" >> $GITHUB_ENV

- name: Build and Push Docker Images
  run: |
    SERVICE_NAME="${REPO_NAME,,}"
    IMAGE_NAME="${{ secrets.HARBOR_REGISTRY }}/${ENVIRONMENT}/${SERVICE_NAME}"
    TIMESTAMP=$(date +%s)

    # Build Docker image
    docker build -t "$IMAGE_NAME:$COMMIT_SHA-$TIMESTAMP" -t "$IMAGE_NAME:latest" .

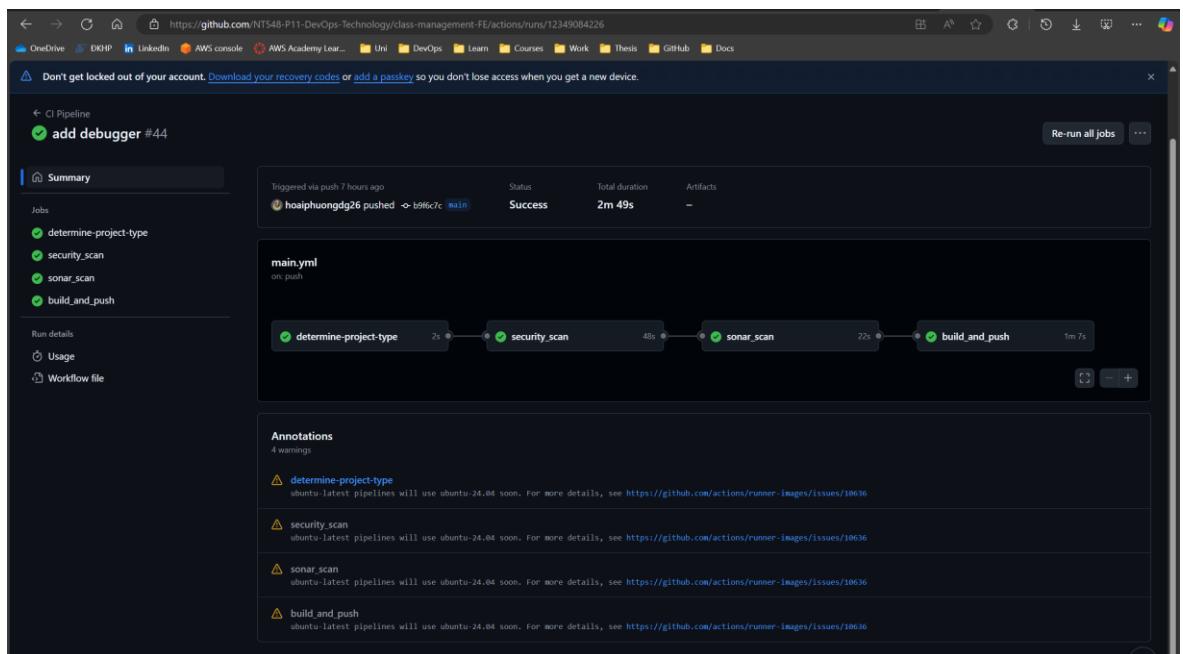
    # Push Docker image
    docker push "$IMAGE_NAME:$COMMIT_SHA-$TIMESTAMP"
    docker push "$IMAGE_NAME:latest"

```

Hình 112. Đặt tên tag cho các image

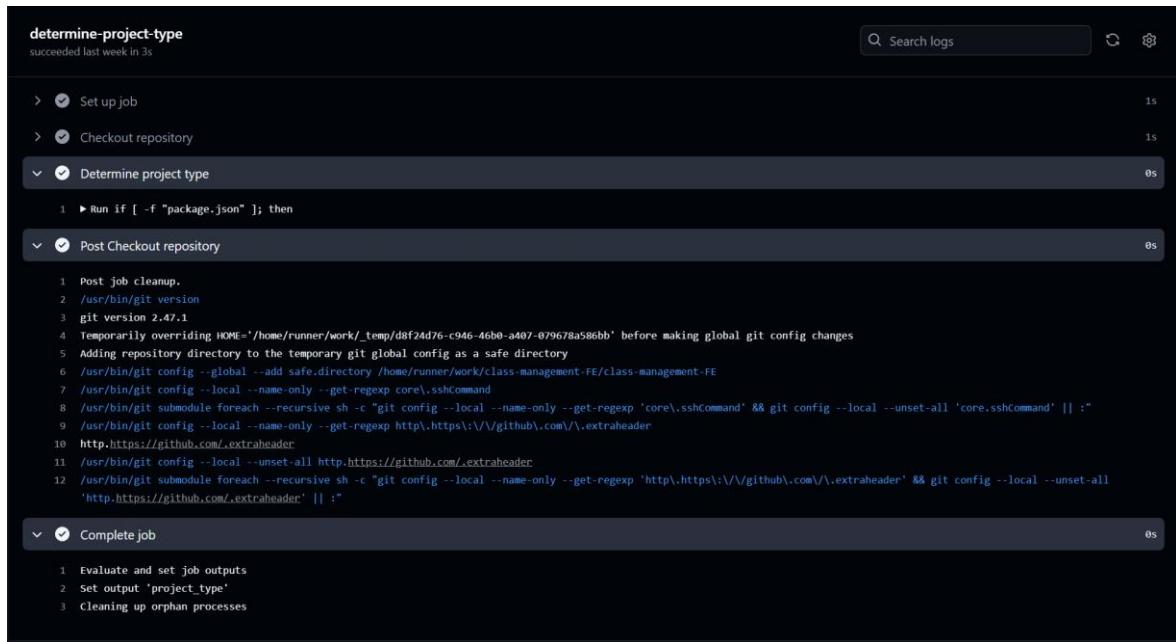
Mô tả chi tiết luồng CI:

Bước 1. Luồng CI sẽ được trigger mỗi khi có commit mới được push lên.



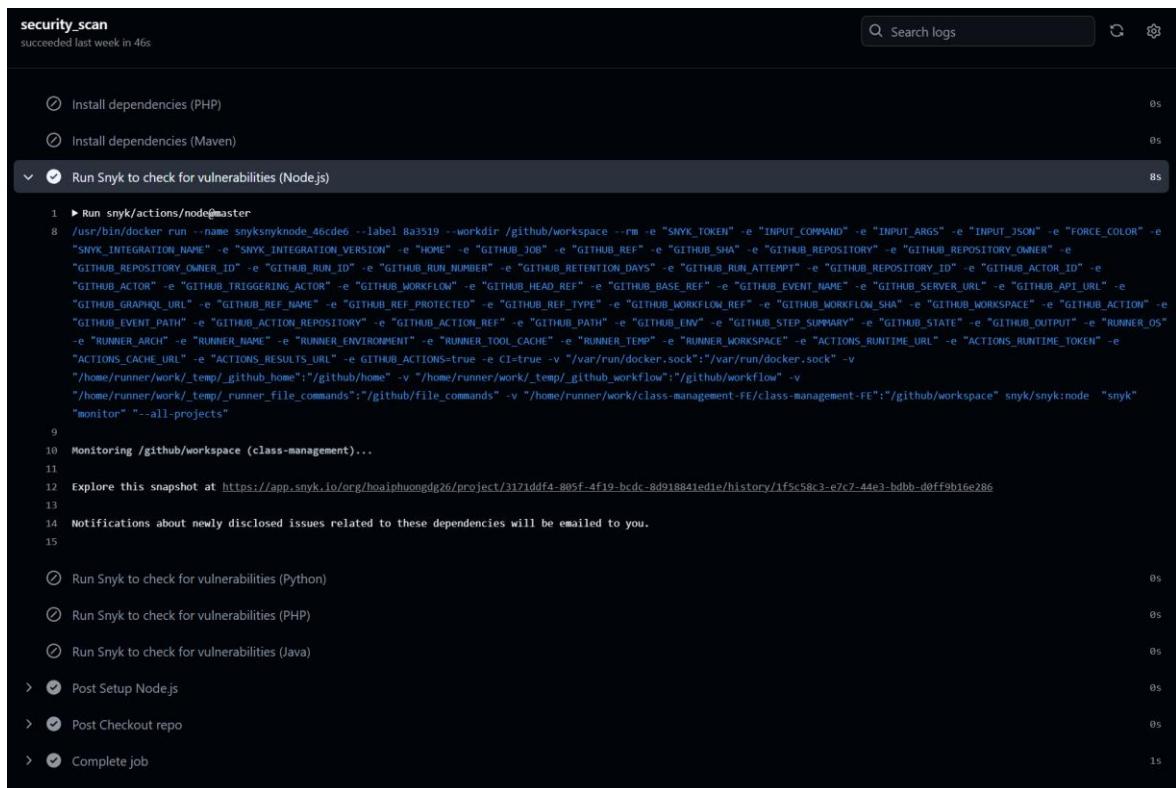
Hình 113. Luồng CI của GitHub Actions

Bước 2. Xác định loại dự án (determine-project-type) dựa trên các tệp cấu hình có trong repository. Pipeline sẽ kiểm tra sự hiện diện của các tệp đặc trưng cho từng loại dự án (Node.js, Python, PHP, Java). Sau khi xác định loại dự án, kết quả sẽ được lưu vào output của job để sử dụng trong các stage tiếp theo.



Hình 114. Xác định loại dự án (determine-project-type)

Bước 3. Sử dụng Snyk được sử dụng để quét lỗ hổng bảo mật trong mã nguồn và các phụ thuộc của dự án. Mục đích giúp phát hiện sớm các vấn đề bảo mật trong mã nguồn và thư viện phụ thuộc.



Hình 115. Quét lỗ hổng bảo mật bằng Snyk

The screenshot shows the Snyk interface with the following details:

- Organization:** hoaiphuongdg26
- Projects:** NT548-P11-DevOps-Technology/class-management-student-service, NT548-P11-DevOps-Technology/class-management-FE, NT548-P11-DevOps-Technology/class-management-auth-service, NT548-P11-DevOps-Technology/class-management-class-service.
- Targets:** com.th1enim02:student_service, class-management, auth-service.
- Issues:** C, H, M, L severity levels.
- Logs:** Add projects, View import logs, Add filter, Group by targets, Sort by highest severity.

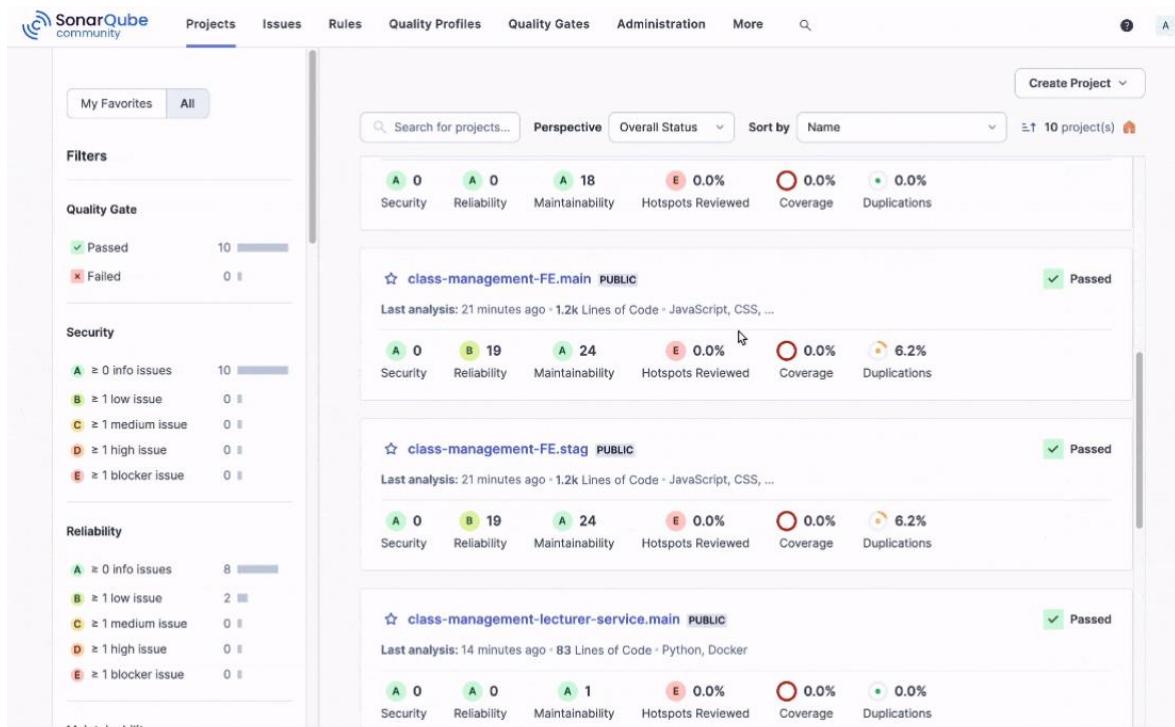
Hình 116. Xem kết quả trên giao diện Snyk sau khi scan

Bước 4. SonarQube được sử dụng trong stage này để quét chất lượng mã nguồn. Công cụ này sẽ phân tích mã nguồn của dự án, kiểm tra các vấn đề như lỗi, cảnh báo, mã lặp lại, và tuân thủ các quy tắc lập trình.

The screenshot shows the SonarQube log terminal with the following details:

- sonar.scan** succeeded last week in 24s
- Set up job**, **Checkout repo**, **Set up SonarQube Scanner**
- Logs:** Search logs, Cache restored successfully, Cache restored from key: sonar-scanner-cli-6.2.1.4610-Linux-X64, Cache size: >50 MB (52600491 B), /usr/bin/tar -xf /home/runner/work/_temp/881927ad-8fb4-4a02-a256-9b30395cf526/cache.tzst -P -C /home/runner/work/class-management-FE/class-management-FE --use-compress-program unzstd, Run echo "\$GITHUB_TEMP/sonar-scanner-cli-6.2.1.4610-Linux-X64" >> \$GITHUB_PATH, Run \$GITHUB_ACTION_PATH/scripts/run-sonar-scanner-cli.sh -Dsonar.projectKey=class-management-FE.main -Dsonar.sources=*.java,**/*node_modules/**, **/*.json -Dsonar.language=js,ts -Dsonar.sourceEncoding=UTF-8 + sonar-scanner -Dsonar.projectBaseDir=.. -Dsonar.projectKey=class-management-FE.main -Dsonar.sources=.. -Dsonar.exclusions=**/*.java,**/*node_modules/**, **/*.json -Dsonar.language=js,ts -Dsonar.sourceEncoding=UTF-8

Hình 117. Quét chất lượng mã nguồn bằng SonarQube



Hình 118. Xem kết quả trên giao diện SonarQube sau khi scan

Bước 5. Mã nguồn sẽ được đóng gói (build), sau đó đẩy (push) image lên registry (Harbor). Đây là bước quan trọng để triển khai ứng dụng vào môi trường production hoặc staging.

The screenshot shows a terminal or CI pipeline log for a job named 'build_and_push' which succeeded last week in 57s. The log details the steps taken:

- Checkout repository (0s)
- Set up Docker Buildx (2s)
- Set up Docker (1s)
- Login to Harbor (0s)
- Set environment variables (0s)
- Build and Push Docker Images (48s)**

Below this, the log shows the Docker build process:

```

1  # Run SERVICE_NAME="${REPO_NAME,,}"
2  #0 building with "default" instance using docker driver
3
4  #1 [internal] load build definition from Dockerfile
5  #1 transferring dockerfile: 1678 done
6  #1 DONE 0.0s
7
8  #2 [internal] load metadata for docker.io/library/node:20-alpine
9  #2 DONE 0.0s
10
11 #3 [internal] load .dockerignore
12 #3 transferring context: 7kB done
13 #3 DONE 0.0s
14
15 #4 [1/5] FROM docker.io/library/node:20-alpine
16 #4 DONE 0.0s
17
18 #5 [internal] load build context
19 #5 transferring context: 1.40MB 0.0s done
20 #5 DONE 0.0s
21
22 #6 [2/5] WORKDIR /app-frontend
23

```

Hình 119. Build và push image lên Harbor

Bước 6. Image sau khi được push lên Harbor đã được đánh tag dựa trên mã commit và timestamp tại thời điểm build.

The screenshot shows the Harbor Artifacts page for the project 'class-management-fe'. It lists two artifacts: 'sha256:b930a834' and 'sha256:cd407548'. Both artifacts have the tag 'latest'. The first artifact has a size of 158.94MB and 22 fixable vulnerabilities. The second artifact also has a size of 158.94MB and 22 fixable vulnerabilities. The 'PULL TIME' and 'PUSH TIME' for both are 12/16/24, 3:55 PM. A modal window is open over the table, showing the same information: 'latest' tag, 158.94MB size, 22 fixable vulnerabilities, and the same push/pull time.

Hình 120. Thông tin image sau khi được push lên Harbor

Bước 7. Đồng thời image cũng được scan bởi Trivy và hiển thị thông số sau khi scan hoàn tất.

The screenshot shows the Harbor Artifacts page for the project 'class-management-fe'. It lists the same two artifacts as before. A modal window titled 'Vulnerability Severity: High' is displayed, stating there are 22 total fixable vulnerabilities. Below the modal, a chart shows the distribution of vulnerabilities by severity: Critical (0), High (11), Medium (10), Low (1), and None (0). The modal also includes details about the scan: 'Scanned by: Trivy@v0.51.2', 'Duration: 8 sec', and 'Scan completed time: 12/16/24, 3:55 PM'.

Hình 121. Image được scan bởi Trivy sau khi được push lên

Bước 8. Thông báo được gửi đến Telegram thông qua webhook, trong đó chứa file .csv bao gồm thông tin chi tiết nội dung của image được scan.

The screenshot shows a Telegram message from 'NT548P11 Group03 Trivy Scan'. The message contains a 'Trivy Scan Report' attachment and a link to a 'Scan report'. The report table shows the following data:

Repository	Artifact Digest	CVE	Package	Current Version	Fixed in version	Severity	CWE Ids	Additional Data	Scanner
production/class-management-fe	b9f6c7c-1734339283.csv		axis	1.64	1.74	High	CWE-918	{CVSS: "med"}	Trivy
			body-parser	1.20.2	1.20.3	High	CWE-405	{CVSS: "med"}	Trivy
			braces	3.0.2	3.0.3	High	CWE-1050,CWE-1051	{CVSS: "high"}	Trivy
			cookie	0.50	0.70	Low	CWE-74	{CVSS: "low"}	Trivy
			cross-spawn	7.0.3	7.0.5, 8.0.6	High	CWE-1333	{CVSS: "high"}	Trivy
			ejs	4.1.19	4.1.20	Medium	CWE-493	{CVSS: "high"}	Trivy
			express	4.18.3	4.20.0, 5.0.0	Low	CWE-79	{CVSS: "med"}	Trivy
			express	4.18.3	4.19.2, 5.0.0-beta.3	Medium	CWE-126,CWE-1333	{CVSS: "high"}	Trivy
			http-proxy-middleware	2.0.6	2.0.7, 3.0.3	High	CWE-400	{CVSS: "med"}	Trivy
			micromatch	4.0.5	4.0.8	Medium	CWE-1133	{CVSS: "high"}	Trivy
			nanoid	3.3.7	5.0.8, 3.3.8	Medium	CWE-843	{CVSS: "high"}	Trivy
			nth-check	1.0.2	2.0.1	High	CWE-1333	{CVSS: "med"}	Trivy
			path-to-regexp	0.1.7	1.9.0, 1.10, 8.0.0, 3.3.0, 6.3.0	High	CWE-1333	{CVSS: "high"}	Trivy
			path-to-regexp	0.1.7	0.1.12	Medium	CWE-1333	{CVSS: "med"}	Trivy
			postcss	7.0.39	8.4.31	Medium	CWE-74	{CVSS: "med"}	Trivy
			rollup	2.79.1	3.29.5, 4.22.4, 2.79.2	High	CWE-79	{CVSS: "med"}	Trivy
			send	0.18.0	0.19.0	Low	CWE-79	{CVSS: "med"}	Trivy
			serve-static	1.15.0	1.16.0, 2.1.0	Low	CWE-79	{CVSS: "med"}	Trivy
			webpack	5.90.3	5.94.0	Medium	CWE-79	{CVSS: "med"}	Trivy
			webpack-dev-m	5.3.3	7.1.0, 6.12, 5.3.4	High	CWE-22	{CVSS: "high"}	Trivy
			ws	7.5.9	5.2.4, 6.2.3, 7.5.10, 6.17.1	High	CWE-476	{CVSS: "high"}	Trivy
			ws	8.16.0	5.2.4, 6.2.3, 7.5.10, 6.17.1	High	CWE-476	{CVSS: "high"}	Trivy

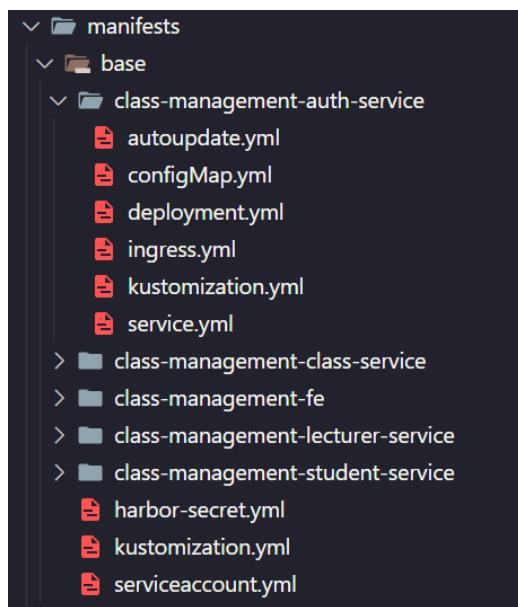
Hình 122. Thông báo kết quả scan bởi Trivy được gửi đến Telegram

Bằng cách tích hợp các công cụ bảo mật như Snyk, SonarQube vào quy trình CI, giúp không chỉ nâng cao chất lượng phần mềm mà còn giảm thiểu rủi ro và tối ưu hóa thời gian triển khai, góp phần tạo ra các ứng dụng an toàn và đáng tin cậy hơn trong môi trường sản xuất.

4.4.2. Quy trình CD

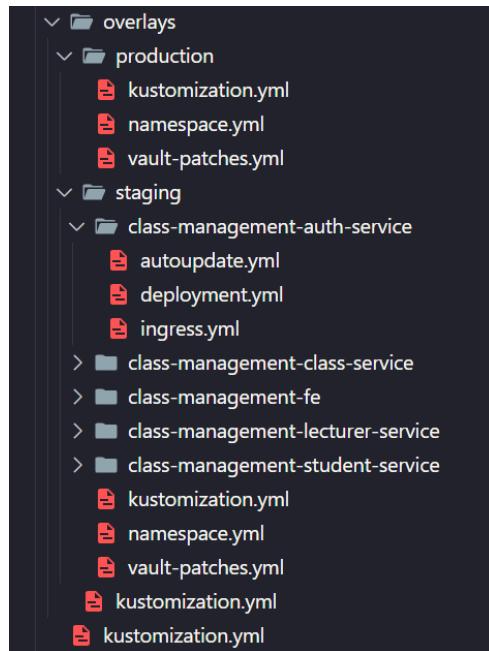
Cấu trúc thư mục cho manifest: Hệ thống được tổ chức theo mô hình Kustomize với 2 thư mục chính:

- manifests/base:
 - Đây là thư mục chứa các manifest gốc.
 - Mỗi service có một thư mục riêng chứa các cấu hình cơ bản.
 - Các cấu hình này sẽ được dùng làm base cho tất cả môi trường.



Hình 123. Cấu trúc thư mục manifests/base

- overlays:
 - Chứa các cấu hình riêng theo môi trường (staging, production).
 - Mỗi môi trường có thư mục riêng.
 - Trong mỗi môi trường, mỗi service có thư mục riêng chứa các file patch.
 - Các file patch này sẽ ghi đè/bổ sung lên cấu hình gốc từ base.



Hình 124. Cấu trúc thư mục *overlays*

```

manifests > overlays > staging > kustomization.yml
You, 2 weeks ago | 1 author (You)
1 resources:
2   - ../../base
3   - namespace.yml
4 namespace: staging
5 patches:
6   - path: ./class-management-fe/ingress.yml
7   - path: ./class-management-fe/configMap.yml
8   - path: ./class-management-fe/deployment.yml
9   - path: ./class-management-fe/autoupdate.yml
10  - path: ./class-management-auth-service/ingress.yml
11  - path: ./class-management-auth-service/deployment.yml
12  - path: ./class-management-auth-service/autoupdate.yml
13  - path: ./class-management-student-service/ingress.yml
14  - path: ./class-management-student-service/deployment.yml
15  - path: ./class-management-student-service/autoupdate.yml
16  - path: ./class-management-lecturer-service/ingress.yml
17  - path: ./class-management-lecturer-service/deployment.yml
18  - path: ./class-management-lecturer-service/autoupdate.yml
19  - path: ./class-management-class-service/ingress.yml
20  - path: ./class-management-class-service/deployment.yml
21  - path: ./class-management-class-service/autoupdate.yml
22  - path: ./vault-patches.yml

```

Hình 125. Cách áp dụng patch từ các file

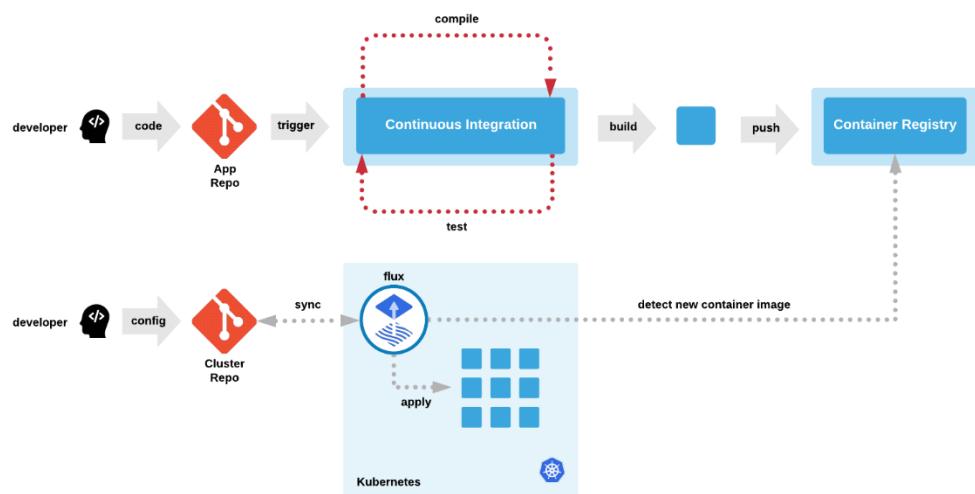
Nguyên tắc hoạt động: Khi triển khai, Kustomize sẽ:

- Lấy cấu hình gốc từ base.
- Áp dụng các patch tương ứng với môi trường.
- Tạo ra manifest cuối cùng để apply vào cluster.

Ưu điểm của cách tổ chức này:

- Tách biệt rõ ràng giữa cấu hình cơ bản và cấu hình môi trường.
- Dễ dàng quản lý sự khác biệt giữa các môi trường.
- Giảm thiểu việc lặp lại code.
- Dễ dàng thêm môi trường mới.

Luồng CD hoạt động với FluxCD:



Hình 126. Luồng CD hoạt động với FluxCD

Quy trình CD sử dụng FluxCD giúp tự động hóa việc triển khai ứng dụng từ Harbor tới EKS. Dưới đây là một số thành phần và quy trình triển khai cụ thể:

- ImageRepository: Định nghĩa nguồn image cần sử dụng từ Harbor, với thông tin về repository và khoảng thời gian quét.

```
You, 2 weeks ago | 1 author (You)
1  apiVersion: image.toolkit.fluxcd.io/v1beta2
2  kind: ImageRepository
3  metadata:
4    name: frontend-repo
5  spec:
6    image: harbor.thienlm02.live/production/class-management-fe
7    interval: 20s
8    provider: generic
9    secretRef:
10   name: harbor-secret
```

Hình 127. FluxCD: Định nghĩa ImageRepository

- interval quy định thời gian giữa các lần kiểm tra để phát hiện image mới.
- secretRef chứa thông tin xác thực cho Harbor.

- ImagePolicy: Định nghĩa cách thức xử lý các image tag theo quy tắc và biểu thức chính quy.

```

12  ---
13  apiVersion: image.toolkit.fluxcd.io/v1beta1
14  kind: ImagePolicy
15  metadata:
16    name: frontend-policy
17  spec:
18    imageRepositoryRef:
19      name: frontend-repo
20    filterTags:
21      pattern: '^a-fA-F0-9]+-(?P<ts>.*$)'
22      extract: '$ts'
23    policy:
24      numerical:
25        order: asc
26

```

Hình 128. FluxCD: Định nghĩa ImagePolicy

- Quy tắc filterTags dùng để chọn lọc các thẻ với cú pháp đặc biệt và trích xuất thông tin về thời gian từ thẻ đó.
- ‘policy’ xác định cách sắp xếp các thẻ hình ảnh theo thứ tự tăng dần hoặc giảm dần.
- Mục đích để xác định được image mới nhất được push thông qua policy và cập nhật cấu hình của deployment.
- ImageUpdateAutomation: Định nghĩa việc tự động cập nhật các image vào manifest trong GitHub.

```

27  ---
28  apiVersion: image.toolkit.fluxcd.io/v1beta1
29  kind: ImageUpdateAutomation
30  metadata:
31    name: frontend-img
32  spec:
33    interval: 30m
34    sourceRef:
35      kind: GitRepository
36      name: flux-system
37      namespace: flux-system
38    git:
39      checkout:
40        ref:
41          branch: main
42      commit:
43        author:
44          email: fluxcdbot@users.noreply.github.com
45          name: Flux
46          messageTemplate: 'Release Production'
47        push:
48          branch: main
49      update:
50        path: ./manifests/base/class-management-fe
51        strategy: Setters

```

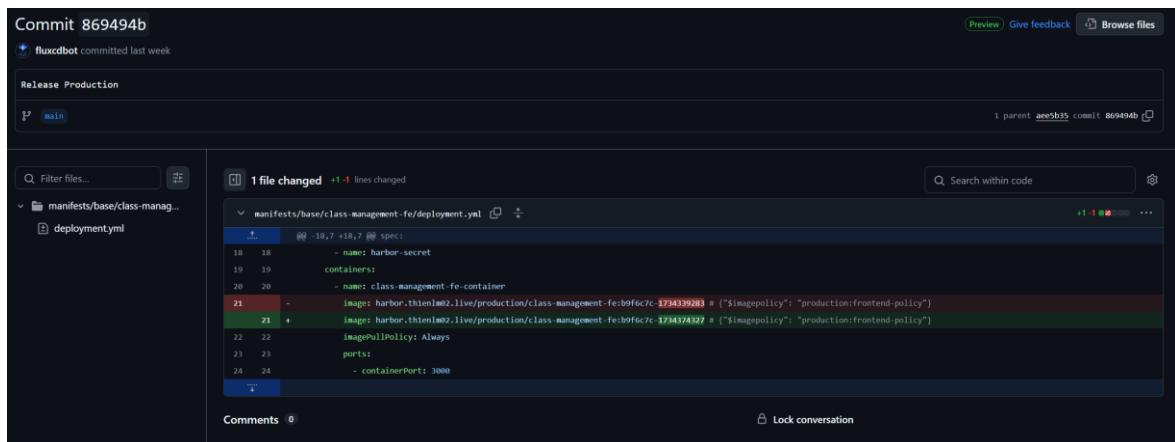
Hình 129. FluxCD: Định nghĩa ImageUpdateAutomation

- Sử dụng GitHub để kiểm tra và push commit lên chi nhánh chính để cập nhật lại thông tin image tag mới nhất dựa trên policy đã được cấu hình.
- ‘update’ chỉ định các tệp manifest cần cập nhật, với chiến lược sử dụng Setters để cập nhật các giá trị cần thiết.

Quy trình CD này cho phép tự động hóa việc triển khai các bản phát hành mới của ứng dụng, từ việc lấy image mới từ Harbor, cập nhật tag, đến việc tự động cập nhật manifest trong GitHub và apply thay đổi vào các môi trường.

Mô tả chi tiết luồng CD:

Bước 1. FluxCD sẽ liên tục theo dõi repository của GitHub và Harbor để kiểm tra cập nhật mới nhất. Trường hợp khi có image mới được push lên Harbor, FluxCD sẽ thực hiện thay đổi thông tin image tag trên repository được cấu hình và apply lại thông tin cấu hình mới nhất lên cụm EKS.



Hình 130. FluxCD push commit thay đổi image tag trên repository

Bước 2. Dựa vào timestamp được trích xuất từ image tag, luồng CD sẽ xác định được thông tin image mới nhất và cập nhật lại thông tin image tại repo chứa manifests dựa trên policy đã được cấu hình.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  labels:
    app: class-management-fe
spec:
  replicas: 1
  selector:
    matchLabels:
      app: class-management-fe
  template:
    metadata:
      labels:
        app: class-management-fe
  spec:
    imagePullSecrets:
      - name: harbor-secret
    containers:
      - name: class-management-fe-container
        image: harbor.thienlm02.live/production/class-management-fe:b9f6c7-174339283 # ["$imagePolicy": "production:frontend-policy"]
        imagePullPolicy: Always
        ports:
          - containerPort: 3000
        env:
          - name: REACT_APP_BASE_URL_AUTH
            valueFrom:
              configMapKeyRef:
                name: base-url-configmap
                key: REACT_APP_BASE_URL_AUTH
          - name: REACT_APP_BASE_URL_STUDENT
            valueFrom:
              configMapKeyRef:
                name: base-url-configmap
                key: REACT_APP_BASE_URL_STUDENT
          - name: REACT_APP_BASE_URL_LECTURER
            valueFrom:
              configMapKeyRef:
                name: base-url-configmap
                key: REACT_APP_BASE_URL_LECTURER

```

Hình 131. Image tag được cập nhật lại dựa vào policy

Bước 3. Kiểm tra thông tin các service đã được triển khai trên cụm EKS. Ứng dụng được triển khai trên 2 môi trường staging và production tương ứng với 2 namespace cùng tên.

```

❯ kubectl get pods --all-namespaces -o wide -n production
NAME                               READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
auth-deployment-5545947d8d-9jtgk   2/2    Running   0          62m    10.0.2.230  ip-10-0-2-206.ec2.internal <none>  <none>
auth-mysql-deployment-7d8c47699c-mwgl9 2/2    Running   0          62m    10.0.2.173  ip-10-0-2-146.ec2.internal <none>  <none>
class-deployment-d65b6567b-z2r9k   2/2    Running   0          62m    10.0.2.74   ip-10-0-2-206.ec2.internal <none>  <none>
class-postgresql-deployment-65fb48b7-qq8sg 2/2    Running   0          62m    10.0.2.52   ip-10-0-2-146.ec2.internal <none>  <none>
frontend-deployment-588bc4b49c-6qgg7  1/1    Running   0          62m    10.0.2.238  ip-10-0-2-206.ec2.internal <none>  <none>
lecturer-deployment-5464fc6759-vlsdp  2/2    Running   0          62m    10.0.10.160  ip-10-0-10-203.ec2.internal <none>  <none>
lecturer-mongodb-deployment-684fd6b84-hc28l 2/2    Running   6 (2m2s ago) 62m    10.0.30.218  ip-10-0-30-189.ec2.internal <none>  <none>
student-deployment-5bb6fff857-64n47   2/2    Running   0          62m    10.0.2.72   ip-10-0-2-206.ec2.internal <none>  <none>
student-mysql-deployment-5f67dc985f-vjlkld  2/2    Running   0          62m    10.0.30.215 ip-10-0-30-189.ec2.internal <none>  <none>
>
❯ kubectl get pods --all-namespaces -o wide -n staging
NAME                               READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
auth-deployment-c8489fd77-5cjf7   2/2    Running   0          8m12s  10.0.2.49   ip-10-0-2-206.ec2.internal <none>  <none>
auth-mysql-deployment-64b58d8f69-lvbq7 2/2    Running   0          8m12s  10.0.2.219  ip-10-0-2-206.ec2.internal <none>  <none>
class-deployment-65ccc89db6-858cf  2/2    Running   0          8m12s  10.0.10.244 ip-10-0-10-203.ec2.internal <none>  <none>
class-postgresql-deployment-7c4496d6cf-khqgq 2/2    Running   0          8m12s  10.0.10.193  ip-10-0-10-203.ec2.internal <none>  <none>
frontend-deployment-79d7c76cf-wgr6d  1/1    Running   0          8m12s  10.0.30.192 ip-10-0-30-189.ec2.internal <none>  <none>
lecturer-deployment-64dbbd94-glbkg  2/2    Running   0          95s    10.0.2.177  ip-10-0-2-206.ec2.internal <none>  <none>
lecturer-mongodb-deployment-7676548464-897jz 2/2    Running   6 (2m10s ago) 8m12s  10.0.30.139 ip-10-0-30-189.ec2.internal <none>  <none>
student-deployment-776dd9b5-vdkv2  2/2    Running   1 (7m50s ago) 8m12s  10.0.2.45   ip-10-0-2-206.ec2.internal <none>  <none>
student-mysql-deployment-54bc86b8f5-f46s9  2/2    Running   0          8m12s  10.0.30.72   ip-10-0-30-189.ec2.internal <none>  <none>

```

Hình 132. Kiểm tra trạng thái ứng dụng trên môi trường staging và production

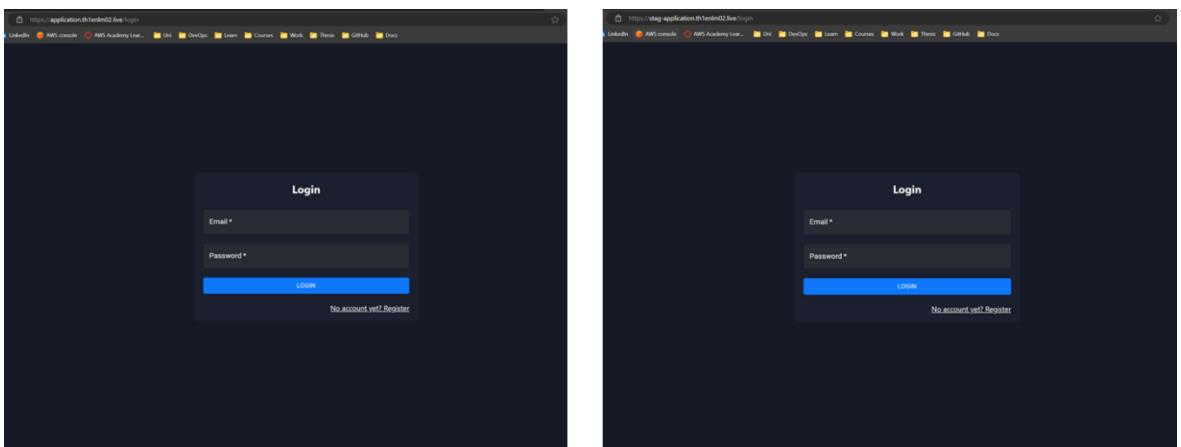
```

❯ kubectl get svc --all-namespaces -o yaml
NAME           CLASS  HOSTS          ADDRESS                                     PORTS   AGE
auth-service-ingress  nginx  auth-api.thienlm02.live  a4a904cae36f8485eae23fe...  80      7h2m
class-service-ingress  nginx  class-api.thienlm02.live a4a904cae36f8485eae23fe...  80      7h2m
frontend-ingress   nginx  application.thienlm02.live a4a904cae36f8485eae23fe...  80      7h2m
lecturer-service-ingress  nginx  lecturer-api.thienlm02.live a4a904cae36f8485eae23fe...  80      7h2m
student-service-ingress  nginx  student-api.thienlm02.live a4a904cae36f8485eae23fe...  80      7h2m
>
❯ kubectl get svc --all-namespaces -o yaml
NAME           CLASS  HOSTS          ADDRESS                                     PORTS   AGE
auth-service-ingress  nginx  stag-auth-api.thienlm02.live a4a904cae36f8485eae23fe...  80      11m
class-service-ingress  nginx  stag-class-api.thienlm02.live a4a904cae36f8485eae23fe...  80      11m
frontend-ingress   nginx  stag-application.thienlm02.live a4a904cae36f8485eae23fe...  80      11m
lecturer-service-ingress  nginx  stag-lecturer-api.thienlm02.live a4a904cae36f8485eae23fe...  80      11m
student-service-ingress  nginx  stag-student-api.thienlm02.live a4a904cae36f8485eae23fe...  80      11m

```

Hình 133. Kiểm tra thông tin Ingress của môi trường staging và production

Bước 4. Truy cập vào ứng dụng với endpoint tương ứng của từng môi trường để đảm bảo ứng dụng được triển khai thành công.



Hình 134. Truy cập vào ứng dụng với endpoint tương ứng của từng môi trường

Chương 5. KIỂM THỬ

5.1. Kiểm thử triển khai hạ tầng

Video demo sẽ trình bày quá trình triển khai cơ sở hạ tầng từ đầu đến cuối. Bắt đầu bằng việc commit mã Terraform lên repository, quy trình được kiểm tra bảo mật bởi Checkov. Sau đó, thực hiện lệnh terraform apply để triển khai cơ sở hạ tầng, với toàn bộ quá trình được giám sát trên giao diện Terraform Cloud. Cuối cùng, kiểm tra trực tiếp các tài nguyên đã được tạo trên AWS Console để xác nhận kết quả triển khai.

Link video demo:

https://drive.google.com/file/d/1IaN9VDx1kSyUh5nQLP101omfxoVQdnBK/view?usp=drive_link

5.2. Kiểm thử CI/CD

Video sẽ minh họa quy trình kiểm thử pipeline CI/CD. Quy trình bắt đầu khi mã nguồn mới được commit lên hai nhánh ‘stag’ và ‘prod’, tương ứng với hai môi trường. Theo dõi các job trong GitHub Actions được thực hiện tuần tự. Image sau khi được push lên Harbor được tự động scan bảo mật bởi Trivy, với thông báo kết quả scan được gửi đến Telegram. Trong phần CD, FluxCD cập nhật tag image mới trong repo manifest và tự động áp dụng manifest, triển khai phiên bản mới trên Kubernetes. Cuối video, kiểm tra endpoint của cả hai môi trường để đảm bảo ứng dụng hoạt động bình thường.

Link video demo:

https://drive.google.com/file/d/1axQf2kU06HsuZSouREWKh0fYIHQ-x42/view?usp=drive_link

5.3. Kiểm thử tính sẵn sàng

Video sẽ kiểm tra khả năng duy trì dịch vụ khi gặp sự cố. Quy trình bắt đầu bằng việc tắt dịch vụ HA Proxy trong scaling group. Hệ thống sẽ tự động tạo một EC2 mới và khởi tạo lại HA Proxy. Trong khi đó, kiểm tra cho thấy ứng dụng không

khả dụng. Sau khi EC2 và HA Proxy mới sẵn sàng, dịch vụ được phục hồi và ứng dụng hoạt động trở lại bình thường, chứng minh tính sẵn sàng của hệ thống.

Link video demo:

[https://drive.google.com/file/d/1jvSz1XJGTFYiskR2BXti0DEDAZwLLq_z/view
?usp=drive_link](https://drive.google.com/file/d/1jvSz1XJGTFYiskR2BXti0DEDAZwLLq_z/view?usp=drive_link)

Chương 6. KẾT LUẬN & HƯỚNG PHÁT TRIỂN

6.1. Kết luận

Sau quá trình nghiên cứu và triển khai, đề tài "**Triển khai mô hình DevSecOps cho hệ thống CI/CD có độ sẵn sàng cao trong môi trường microservices**" đã hoàn thành các mục tiêu quan trọng cả về lý thuyết lẫn thực tiễn.

Về mặt kỹ thuật: Nhóm đã xây dựng thành công một quy trình CI/CD tích hợp chặt chẽ các yếu tố bảo mật ngay từ giai đoạn phát triển. Điều này đảm bảo rằng mã nguồn, các dịch vụ triển khai, và cơ sở hạ tầng đều tuân thủ các tiêu chuẩn bảo mật cao nhất, giảm thiểu nguy cơ sự cố trong quá trình vận hành. Hệ thống được thiết kế với khả năng giám sát, cảnh báo và tự động phản ứng nhanh chóng, hỗ trợ duy trì tính liên tục và độ ổn định cho dịch vụ.

Về mặt ứng dụng thực tế: Mô hình đã chứng minh khả năng rút ngắn thời gian phát hành sản phẩm, nâng cao chất lượng và độ tin cậy của ứng dụng thông qua tự động hóa các quy trình kiểm tra chất lượng và bảo mật. Đồng thời, mô hình cũng góp phần tối ưu hóa chi phí vận hành nhờ việc quản lý tài nguyên hiệu quả và giảm thiểu can thiệp thủ công.

Những kết quả đạt được không chỉ phản ánh tiềm năng áp dụng mô hình DevSecOps trong các hệ thống hiện đại mà còn mở ra cơ hội cải tiến hơn nữa về mặt quy mô, hiệu suất, và an ninh, hướng tới xây dựng các giải pháp công nghệ bền vững và tối ưu trong tương lai.

6.2. Hướng phát triển

Tăng cường kiểm thử và bảo mật động:

- Tích hợp công cụ kiểm thử bảo mật: Sử dụng OWASP ZAP (Zed Attack Proxy) để thực hiện kiểm tra bảo mật tự động cho ứng dụng, đảm bảo phát hiện sớm các lỗ hổng bảo mật trong môi trường thực.
- Triển khai DAST (Dynamic Application Security Testing): Xây dựng quy trình kiểm thử bảo mật động trong pipeline CI/CD nhằm phát hiện và khắc phục lỗ hổng từ giai đoạn phát triển đến triển khai.

- Thiết lập các test case tự động hóa để kiểm tra khả năng chịu đựng tấn công của hệ thống, đặc biệt ở các API endpoints quan trọng.
- Phân tích và báo cáo lỗ hổng bảo mật thời gian thực: Kết hợp công cụ giám sát và báo cáo tập trung để cung cấp thông tin chi tiết về các lỗ hổng ngay khi phát hiện.
- Triển khai các giải pháp tự động hóa kiểm tra độ an toàn của API, ngăn chặn các tấn công phổ biến như injection hay cross-site scripting.

Quy trình phê duyệt và kiểm soát triển khai:

- Xây dựng quy trình approval tự động, liên kết chặt chẽ giữa các môi trường Dev, Staging, và Production, đảm bảo rằng mọi thay đổi đều được xem xét kỹ lưỡng trước khi triển khai.
- Tích hợp công cụ theo dõi và quản lý các yêu cầu phê duyệt, giúp kiểm soát tiến độ và trạng thái từng yêu cầu.
- Kết nối notification system để gửi thông báo đến các bên liên quan khi có yêu cầu phê duyệt hoặc triển khai.
- Đảm bảo rằng mọi yêu cầu đều được kiểm tra tuân thủ chính sách và đáp ứng các tiêu chuẩn bảo mật trước khi được phê duyệt.
- Xây dựng giao diện trực quan để theo dõi toàn bộ quy trình approval, từ yêu cầu đến triển khai.
- Thiết lập quy trình tự động rollback khi phát hiện vấn đề sau triển khai, giúp hạn chế rủi ro và khôi phục nhanh chóng.

PHỤ LỤC

Bảng 2. Bảng phân chia nhiệm vụ

Tên thành viên	Nhiệm vụ	Tỉ lệ hoàn thành
Lưu Minh Thiện	<ul style="list-style-type: none"> - Thiết kế cơ sở hạ tầng - Viết script Terraform, Ansible, Kubernetes manifest - Chuẩn bị mã nguồn ứng dụng - Cấu hình HA Proxy, MinIO, Harbor, Trivy - Triển khai luồng CD - Chuẩn bị slide thuyết trình - Viết báo cáo 	100%
Quách Thị Hoài Phương	<ul style="list-style-type: none"> - Thiết kế cơ sở hạ tầng - Viết script Terraform, Ansible - Cấu hình Snyk, SonarQube - Triển khai luồng CI - Chuẩn bị slide thuyết trình - Viết báo cáo 	100%
Nguyễn Thành Đăng	<ul style="list-style-type: none"> - Thiết kế cơ sở hạ tầng - Viết script Ansible - Cấu hình HA Proxy, OpenVPN - Cấu hình Prometheus, Loki, Grafana - Viết báo cáo 	100%
Trần Khôi Nguyên	<ul style="list-style-type: none"> - Thiết kế cơ sở hạ tầng - Viết script Terraform, Kubernetes manifest - Cấu hình Vault, MinIO - Triển khai luồng CD - Viết báo cáo 	100%

Link slides thuyết trình:

https://docs.google.com/presentation/d/1S1iT2GfMUCmaJuSW8AcMN6UCKiucYNzq/edit?usp=drive_link&ouid=116387112076285976163&rtpof=true&sd=true

Link GitHub:

<https://github.com/NT548-P11-DevOps-Technology>

TÀI LIỆU THAM KHẢO

- [1] “What Is DevOps?” Atlassian. <https://www.atlassian.com/devops> (accessed Dec. 15, 2024)
- [2] “The History of DevOps” EverythingDevOps. <https://everythingdevops.dev/a-brief-history-of-devops-and-its-impact-on-software-development> (accessed Dec. 15, 2024)
- [3] “What Do DevOps Principles Include?” phoenixNAP. <https://phoenixnap.com/blog/devops-principles> (accessed Dec. 15, 2024)
- [4] “Difference Between DevSecOps and DevOps” GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-devops-and-devsecops> (accessed Dec. 15, 2024)
- [5] “What are Microservices?” GeeksforGeeks. <https://www.geeksforgeeks.org/microservices> (accessed Dec. 15, 2024)
- [6] “CI/CD Pipelines là gì ?” <https://devops.vinahost.vn/CI-CD/Overview-CI-CD-Pipelines> (accessed Dec. 15, 2024)
- [7] “A Simple Introduction to Git” <https://telestreak.com/tech/git-tutorial/a-simple-introduction-to-git> (accessed Dec. 15, 2024) [Section: How does Git work?]
- [8] “What is Flux CD?” CNCF <https://www.cncf.io/blog/2023/09/15/what-is-flux-cd> (accessed Dec. 15, 2024)
- [9] “Getting started with Docker” Medium. <https://medium.com/@muhasheikh8/getting-started-with-docker-146a1f936cbe> (accessed Dec. 15, 2024) [Section: What is Docker?]
- [10] “Hashicorp Vault” Gems of Coding. <https://gemsofcoding.com/Hashicorp-Vault>

-
- [11] “**What is Prometheus**” Last9. <https://last9.io/blog/what-is-prometheus> (accessed Dec. 15, 2024)
- [12] “**Log Aggregation using Grafana Loki Stack**” Persistent. <https://www.persistent.com/blogs/log-aggregation-using-grafana-loki-stack> (accessed Dec. 15, 2024) [Section: Grafana Loki Solution Architecture]
- [13] “**What is Amazon VPC?**” AWS. <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html> (accessed Dec. 15, 2024)
- [14] “**Elastic Load Balancing**” Medium. <https://medium.com/analytics-vidhya/elastic-load-balancing-fb6a9f851c53> (accessed Dec. 15, 2024)
- [15] “**Complete Guide to AWS EKS: Architecture, Pricing, and Tips for Success**” Spot. <https://spot.io/resources/aws-eks/complete-guide-aws-eks-architecture-pricing-tips> (accessed Dec. 15, 2024) [Section: What Is AWS EKS (Elastic Kubernetes Service)?]