21521997 – Lê Văn Duy

Kế hoạch chi tiết đề tài: Triển khai CI/CD pipeline toàn diện với GitLab CI, Docker, Kubernetes và ArgoCD cho dự án Golang + React

---

1. Giới thiệu đề tài và mục tiêu

Dự án của tôi tập trung vào việc xây dựng CI/CD pipeline toàn diện với GitLab CI, Docker, và Kubernetes cho một ứng dụng TodoList (bao gồm backend Golang và frontend React). Đặc biệt, ArgoCD sẽ được sử dụng để thực hiện GitOps, đảm bảo quá trình triển khai ứng dụng lên môi trường Kubernetes được tự động và đồng bộ với các thay đổi trong kho mã. Ngoài ra, các loại kiểm thử bảo mật (SAST, SCA, Image Scan, DAST) và kiểm tra hiệu suất sẽ được tích hợp vào pipeline để đảm bảo chất lượng phần mềm.

Mục tiêu của đề tài:

- Tích hợp CI/CD với GitLab CI, Docker, Kubernetes.
- Tự động hóa quá trình kiểm thử bảo mật và hiệu suất.
- Sử dụng GitOps (ArgoCD) để triển khai liên tục lên Kubernetes.
- Đảm bảo quá trình triển khai không gián đoạn và rollback tự động khi có lỗi.

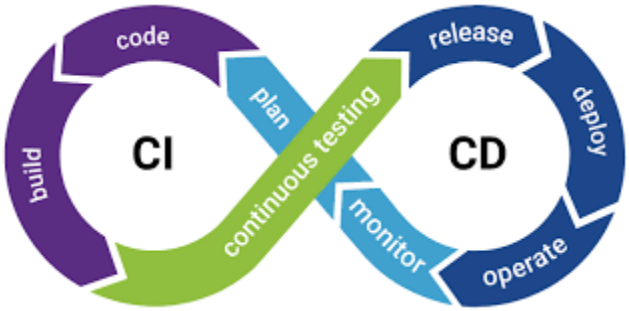
2. Ứng dụng demo

2.1. Mô tả source code

Ứng dụng demo của dự án sẽ là một hệ thống quản lý TodoList với các tính năng CRUD cơ bản (Create, Read, Update, Delete). Ứng dụng này bao gồm:

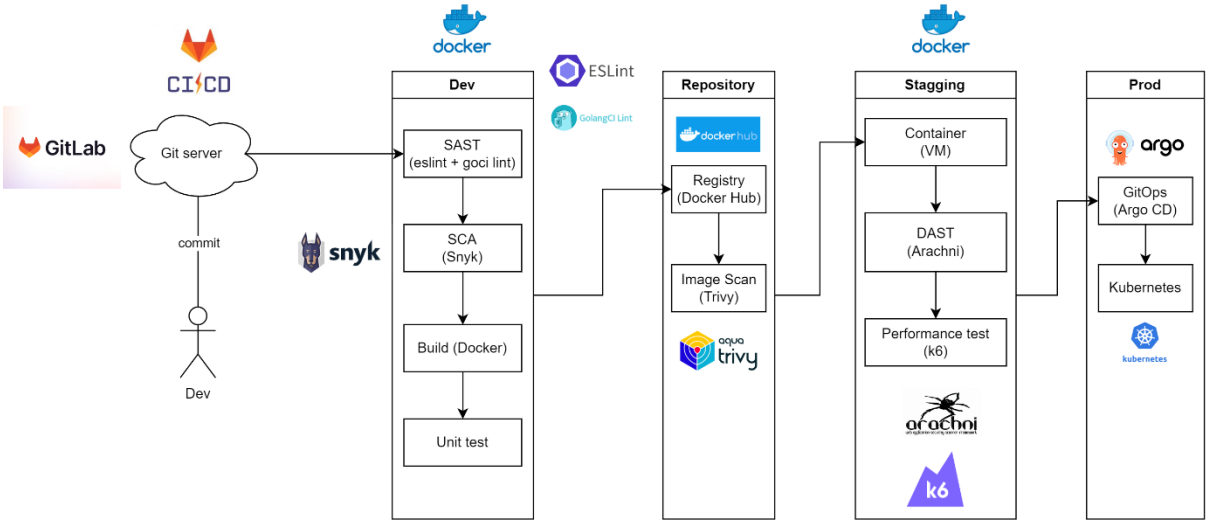
- **Frontend**: Sử dụng ReactJS, Tailwindcss để phát triển giao diện người dùng.
- **Backend**: Được xây dựng bằng Golang, cung cấp các API cần thiết để xử lý các yêu cầu CRUD từ frontend.
- **Database**: Sử dụng MySQL để lưu trữ dữ liệu của ứng dụng.

2.2. Pipeline CI/CD của dự án

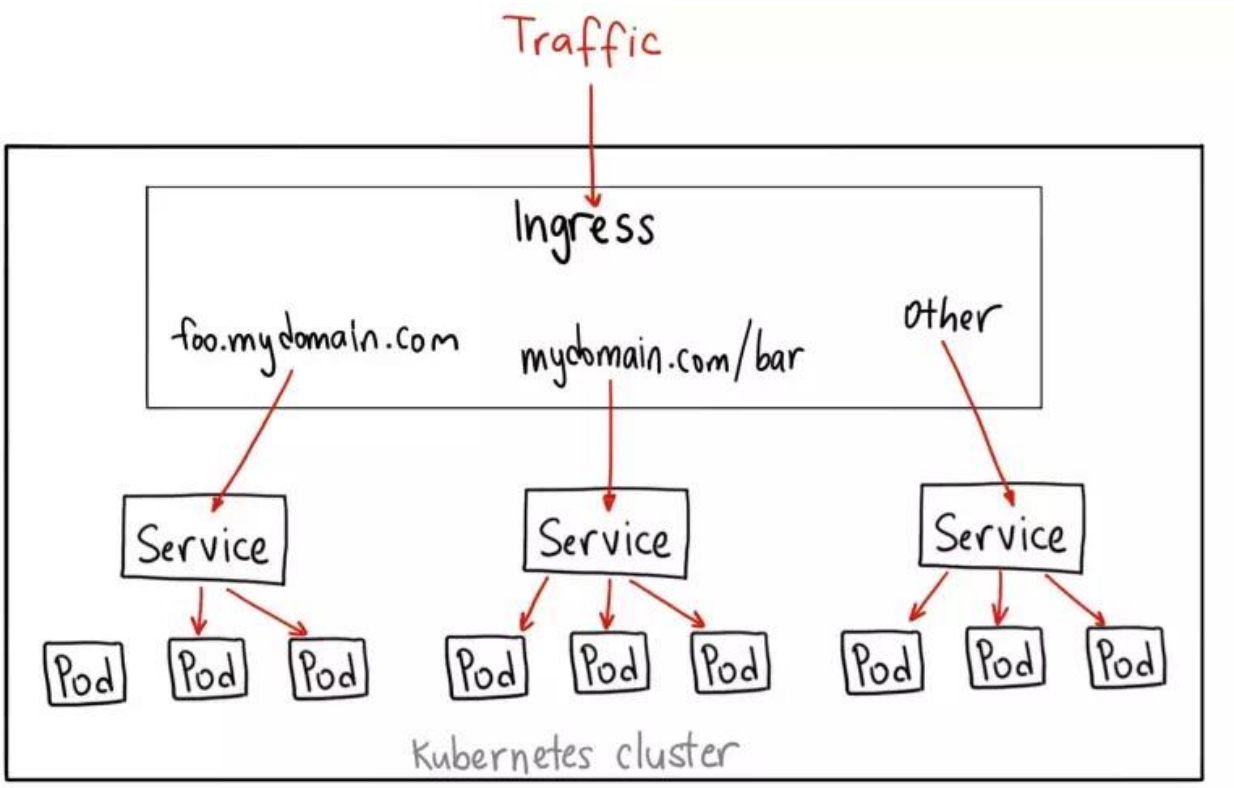


- **Continuous Integration (CI)**: Jenkins sẽ tự động kiểm tra mã nguồn bằng các công cụ như ESLint cho React và gofmt cho Golang. Đồng thời, nó sẽ chạy các unit test cho cả backend và frontend.
- **Continuous Deployment (CD)**: Sau khi mã nguồn được kiểm tra và thông qua, Jenkins sẽ build Docker image cho cả backend và frontend, sau đó triển khai lên môi trường staging và production.
- **Rollback và Zero-downtime Deployment**: Trong trường hợp xảy ra lỗi, Jenkins sẽ tự động rollback và triển khai phiên bản ổn định. Đối với quá trình triển khai không gián đoạn (zero-downtime deployment), Docker Swarm hoặc Docker Compose sẽ được sử dụng, cân nhắc sử dụng K8s.

2.3. Kịch bản sử dụng trong dự án (có thể thay đổi trong lúc thực hiện)



Áp dụng một phần của quy trình DevSecOps



Kịch bản 1: Kiểm thử và tích hợp code (Continuous Integration - CI)

Trigger: Khi có Pull Request (PR) từ nhánh feature vào nhánh develop.

SAST và SCA: GitLab CI sẽ tự động kiểm tra mã nguồn với SAST và SCA để phát hiện các lỗ hổng bảo mật trong mã nguồn và các thành phần phụ thuộc.

Unit Test:

- **Frontend**: Sử dụng Jest hoặc React Testing Library để chạy unit test cho các component React.
- **Backend**: Sử dụng Go test để kiểm tra các chức năng của API Golang.

Build Docker image: Sau khi kiểm thử thành công, GitLab CI sẽ build Docker image cho cả backend và frontend.

Image Scanning: Docker image sẽ được kiểm tra bằng Image Scan để phát hiện các lỗ hổng bảo mật trong các image được tạo.

Kịch bản 2: Triển khai lên môi trường staging (Continuous Deployment - CD)

Trigger: Khi code được **merge vào nhánh develop**.

- **Build Docker image**: Jenkins sẽ tiếp tục build lại Docker image cho cả frontend và backend khi mã nguồn trên nhánh develop thay đổi.
- **Push Docker image lên Docker Hub**: Các image sau khi được build sẽ được đẩy lên một registry như Docker Hub hoặc registry riêng tư, nhằm lưu trữ các phiên bản image để phục vụ quá trình triển khai sau này.
- **Triển khai lên môi trường staging**: GitLab CI sẽ sử dụng Docker Compose để triển khai toàn bộ ứng dụng lên môi trường staging.
- **Smoke Test và** DAST: Sau khi triển khai lên staging, một loạt các bài kiểm tra cơ bản (smoke test) sẽ được thực hiện để đảm bảo rằng các chức năng chính của ứng dụng (CRUD cho TodoList) hoạt động đúng như mong đợi. Sau khi triển khai, GitLab CI sẽ chạy smoke test và DAST để kiểm tra các chức năng chính và lỗ hổng bảo mật khi ứng dụng hoạt động.
- **Notification**: Jenkins sẽ gửi thông báo qua Slack hoặc email về kết quả của quá trình triển khai (thành công hoặc thất bại).

a. **Kịch bản 3: Triển khai lên môi trường production**

Trigger: Khi code được **merge từ nhánh release vào nhánh main**.

- **Backup database**: Trước khi tiến hành triển khai lên môi trường production, Jenkins sẽ thực hiện backup toàn bộ dữ liệu từ MySQL database để đảm bảo không có sự cố mất dữ liệu trong quá trình triển khai.
- **Build và push Docker image**: Jenkins sẽ build lại Docker image từ mã nguồn trên nhánh main và đẩy lên Docker Hub. Điều này giúp giữ cho image triển khai luôn ở phiên bản mới nhất.
- **Triển khai lên production**: Ứng dụng sẽ được triển khai lên Kubernetes môi trường production thông qua ArgoCD.

- **Rollback (trong trường hợp gặp lỗi)**: Nếu sau khi deploy phát hiện lỗi, Jenkins sẽ tự động rollback và khởi động lại phiên bản cũ của Docker image từ registry, đảm bảo ứng dụng vẫn hoạt động bình thường.

b. Kịch bản 4: Xử lý khi ứng dụng gặp sự cố (Failure Handling)

Scenario: Ứng dụng bị lỗi hoặc gặp sự cố sau khi triển khai.

- **Health Check**: Jenkins sẽ thiết lập các endpoint kiểm tra sức khỏe (health check) cho cả frontend và backend. Nếu một trong các dịch vụ gặp sự cố, hệ thống sẽ kích hoạt rollback tự động.
- **Alerting**: Sử dụng hệ thống giám sát để phát hiện lỗi và thông báo qua Slack hoặc email.
- **Rollback**: ArgoCD sẽ tự động rollback nếu hệ thống phát hiện lỗi sau khi triển khai.

c. Kịch bản 5: Triển khai không gián đoạn (Zero-downtime Deployment)

Scenario: Cần triển khai phiên bản mới mà không làm gián đoạn dịch vụ đang hoạt động.

- **Triển khai không gián đoạn**: Docker Swarm hoặc Docker Compose sẽ được sử dụng để đảm bảo rằng khi triển khai phiên bản mới, người dùng không bị ảnh hưởng. Các container cũ sẽ chỉ được tắt sau khi phiên bản mới đã sẵn sàng hoạt động.
- **Rolling Update**: Jenkins sẽ cấu hình để thực hiện rolling update – từng container được cập nhật lần lượt, đảm bảo không có thời gian chết (downtime) trong suốt quá trình deploy.
- **Kiểm thử phiên bản mới**: Sau khi triển khai xong, Jenkins sẽ kiểm tra xem phiên bản mới có hoạt động ổn định không trước khi ngắt phiên bản cũ.

3. Kế hoạch từng tuần (có thể được thay đổi)

- Bắt đầu tính từ ngày nộp proposal là ngày 3/10/2024 tới tuần dự kiến báo cáo là ngày 18/11/2024

3.1. Tuần 1-2: Thiết lập môi trường và nghiên cứu

Tuần 1-2: Thiết lập môi trường và nghiên cứu

- Mục tiêu: Thiết lập các công cụ cần thiết và nghiên cứu về GitLab CI, Docker, Kubernetes, ArgoCD, các công cụ kiểm thử bảo mật và hiệu suất.
- Công việc chi tiết:

- Cài đặt GitLab CI, Docker, Kubernetes (cluster 3 VM) và ArgoCD.
- Thiết lập cấu trúc mã nguồn của ứng dụng TodoList.
- Bắt đầu với cấu hình GitFlow cho quy trình phát triển.

3.2. Tuần 3: Xây dựng CI pipeline

- Mục tiêu: Thiết lập pipeline CI với các kiểm thử ban đầu.
- Công việc chi tiết:
  - Tích hợp SAST (Static Application Security Testing) vào GitLab CI để kiểm tra lỗ hổng bảo mật trong mã nguồn.
  - Cài đặt SCA (Software Composition Analysis) để kiểm tra các thành phần phụ thuộc.
  - Viết unit test cho frontend và backend, tích hợp ESLint và gofmt để kiểm tra chất lượng mã.
  - Cấu hình GitLab CI để tự động chạy kiểm thử khi có Pull Request.

3.3. Tuần 4: Container hóa và kiểm thử Image Scan

- Mục tiêu: Container hóa hệ thống và kiểm thử an toàn của Docker images.
- Công việc chi tiết:
  - Tạo Dockerfile cho backend và frontend.
  - Sử dụng Docker Compose để tạo môi trường container hóa và chạy thử ứng dụng.
  - Tích hợp Image Scanning vào GitLab CI để kiểm tra các lỗ hổng bảo mật trong Docker images.

3.4. Tuần 5: Triển khai Kubernetes môi trường staging

- Mục tiêu: Triển khai ứng dụng lên Kubernetes môi trường staging.
- Công việc chi tiết:
  - Viết manifest cho Kubernetes (deployment, service, configmap) cho backend và frontend.
  - Tích hợp ArgoCD để triển khai GitOps cho Kubernetes.
  - Cấu hình GitLab CI để tự động triển khai ứng dụng lên môi trường staging thông qua ArgoCD.

3.5. Tuần 6: Kiểm thử bảo mật và hiệu suất (DAST, Performance Test)

- Mục tiêu: Tích hợp các loại kiểm thử bảo mật động và hiệu suất.
- Công việc chi tiết:
    - Tích hợp DAST (Dynamic Application Security Testing) vào pipeline để phát hiện các lỗ hổng bảo mật khi ứng dụng hoạt động.
    - Viết và chạy các bài Performance Test để đo lường hiệu suất của ứng dụng.

3.6. Tuần 7: Triển khai production lên Kubernetes

- Mục tiêu: Triển khai lên môi trường production.
- Công việc chi tiết:
    - Backup database trước khi triển khai.
    - Sử dụng ArgoCD để triển khai ứng dụng lên môi trường production Kubernetes.
    - Thiết lập liveness và readiness probes cho các dịch vụ trong Kubernetes để kiểm tra tình trạng ứng dụng.

3.7. Tuần 8: Triển khai không gián đoạn và rollback

- Mục tiêu: Đảm bảo quá trình triển khai không gián đoạn và tích hợp rollback khi có lỗi.
- Công việc chi tiết:
    - Sử dụng chiến lược rolling update trong Kubernetes để triển khai không gián đoạn.
    - Cấu hình GitLab CI và ArgoCD để tự động rollback nếu phát hiện lỗi trong quá trình triển khai.

3.8. Tuần 9: Theo dõi logs và xử lý sự cố

- Mục tiêu: Giám sát và xử lý lỗi.
- Công việc chi tiết:
    - Tích hợp công cụ giám sát log (như ELK Stack hoặc Prometheus + Grafana) để theo dõi trạng thái của ứng dụng.
    - Cấu hình thông báo qua Slack hoặc email khi phát hiện lỗi trong quá trình triển khai hoặc build.

3.9. Tuần 10: Kiểm tra và chuẩn bị demo

- Mục tiêu: Kiểm tra và hoàn thiện toàn bộ hệ thống trước buổi demo.
- Công việc chi tiết:
    - Chạy lại toàn bộ pipeline CI/CD và kiểm tra sự ổn định của ứng dụng.
    - Chuẩn bị tài liệu, báo cáo và kế hoạch demo về các tính năng CI/CD, kiểm thử bảo mật, hiệu suất, và quá trình deploy lên Kubernetes.