THIẾT KẾ VÀ TRIỂN KHAI PIPELINE CI/CD TỰ ĐỘNG CHO ỨNG DỤNG WEB VỚI GITHUB ACTIONS VÀ KUBERNETES

Nguyễn Thị Thuỳ Trang[1]

TÓM TẮT

Trong kỷ nguyên phát triển phần mềm ngày nay, tự động hóa quy trình tích hợp và triển khai liên tục (CI/CD) giữ vai trò quan trọng trong việc nâng cao hiệu suất và chất lượng ứng dụng. Bài báo này trình bày phương pháp thiết kế một pipeline CI/CD tự động nhằm triển khai ứng dụng web lên môi trường thực thi, sử dụng GitHub Actions để thực hiện các bước build và test, kết hợp với Kubernetes để quản lý việc triển khai container. Quy trình được tối ưu hóa để giảm thiểu lỗi thủ công, rút ngắn thời gian phát hành, đồng thời đảm bảo tính ổn định và tiết kiệm tài nguyên. Các bước cụ thể bao gồm cấu hình workflow trong GitHub Actions và tích hợp với Kubernetes cluster để triển khai ứng dụng. Giải pháp này tiết kiệm thời gian, tăng khả năng mở rộng hệ thống, dù đòi hỏi kiến thức DevOps cơ bản.

Từ khoá: *CI/CD, GitHub Actions, Kubernetes, DevOps, ứng dụng web.*

1. Giới thiệu

Trong bối cảnh số hóa, ứng dụng web ngày càng đóng vai trò quan trọng trong các lĩnh vực như thương mại điện tử, giáo dục trực tuyến và dịch vụ công, đòi hỏi tốc độ phát triển nhanh, độ tin cậy cao và chất lượng ổn định. Sự gia tăng độ phức tạp của ứng dụng và nhu cầu cập nhật liên tục đặt ra thách thức cho các nhóm phát triển phần mềm, yêu cầu quy trình phát triển hiệu quả, ít sai sót và có khả năng phản hồi nhanh trước thay đổi thị trường. CI/CD, với khả năng tự động hóa tích hợp mã nguồn, kiểm thử và triển khai, đã trở thành phương pháp cốt lõi để đáp ứng các yêu cầu này, giúp giảm thiểu lỗi thủ công, rút ngắn chu kỳ phát hành và nâng cao tính cạnh tranh [3]. Hiện nay, các công cụ CI/CD như Jenkins, GitLab CI và CircleCI được sử dụng rộng rãi nhờ khả năng tùy chỉnh và hỗ trợ triển khai container hóa [7]. Tuy nhiên, những giải pháp này thường yêu cầu cấu hình phức tạp và chi phí vận hành cao, đặc biệt khi cần thiết lập máy chủ CI riêng hoặc phụ thuộc vào một hệ sinh thái cụ thể, gây khó khăn cho các nhóm phát triển nhỏ với nguồn lực hạn chế [5].

Bài báo đề xuất một pipeline CI/CD tự động kết hợp GitHub Actions và Kubernetes, tối ưu cho các ứng dụng web vừa và nhỏ nhờ đơn giản hóa quy trình và đảm bảo hiệu suất cao. Nghiên cứu tập trung vào việc giảm thiểu cấu hình hạ tầng CI nhờ GitHub Actions, thay vì yêu cầu thiết lập máy chủ riêng và cấu hình phức tạp như Jenkins với các plugin như Docker Pipeline [4, 7]. Hơn nữa, pipeline tận dụng chiến lược rolling update của Kubernetes để đảm bảo triển khai không gián đoạn, kết hợp với GitHub Actions cung cấp khả năng tái sử dụng workflow qua Actions Marketplace – một hệ sinh thái cộng đồng phong phú mà Jenkins khó đạt được mà không cần phát triển plugin riêng. So với GitLab CI kết hợp Kubernetes, pipeline này linh hoạt hơn khi không

bị ràng buộc vào một nền tảng duy nhất, tận dụng hạ tầng đám mây của GitHub để loại bỏ nhu cầu duy trì runner tự quản lý, từ đó giảm chi phí và công sức vận hành [5]. Sự kết hợp này không chỉ đảm bảo triển khai linh hoạt và không gián đoạn trên Kubernetes thông qua chiến lược rolling update, mà còn giải quyết vấn đề thường bị bỏ qua trong các nghiên cứu trước: tối ưu hóa quy trình CI/CD cho các nhóm phát triển nhỏ, nơi nguồn lực kỹ thuật và tài chính hạn chế là thách thức lớn.

Bài báo được trình bày gồm 4 phần. Phần 1 giới thiệu chung về hướng nghiên cứu. Phần 2 trình bày về CI/CD, GitHub Actions và Kubernetes. Phần 3 thiết kế pipeline CI/CD. Phần 4 triển khai pipeline CI/CD.

2. Tổng quan về CI/CD, Github Actions và Kubernetes

2.1. CI/CD và vai trò trong phát triển phần mềm

CI/CD là khái niệm nền tảng trong phát triển phần mềm hiện đại, được Martin Fowler giới thiệu như một phương pháp tự động hóa tích hợp mã nguồn (Continuous Integration) và triển khai ứng dụng (Continuous Deployment) để giảm rủi ro, tăng tốc độ phát hành và nâng cao chất lượng sản phẩm [3]. Trong CI, các thay đổi mã từ nhiều nhà phát triển được tích hợp thường xuyên, sau đó kiểm thử tự động để phát hiện lỗi sớm. CD mở rộng quy trình này bằng cách tự động triển khai các bản cập nhật đã xác nhận lên môi trường sản xuất. CI/CD giúp các nhóm phát triển đáp ứng nhu cầu thị trường nhanh chóng, chẳng hạn như cập nhật tính năng hoặc sửa lỗi trong vài giờ thay vì vài tuần, đảm bảo tính cạnh tranh trong chu kỳ phát triển ngắn.

Theo báo cáo State of DevOps 2021, các tổ chức áp dụng CI/CD hiệu quả có thể tăng tần suất triển khai lên 208 lần và giảm thời gian khôi phục lỗi xuống dưới 1 giờ [2]. Ban đầu, các nghiên cứu CI/CD tập trung vào công cụ như Jenkins, với sự phát triển của công nghệ container và điện toán đám mây đã thúc đẩy sự xuất hiện của các giải pháp linh hoạt hơn như GitLab CI và CircleCI, mang lại hiệu quả triển khai cao hơn [7]. Thành công của CI/CD phụ thuộc vào việc lựa chọn công cụ phù hợp và thiết kế pipeline hiệu quả.

2.2. GitHub Actions: công cụ tự động hóa workflow

GitHub Actions, ra mắt năm 2018, là công cụ tự động hóa nổi bật nhờ tính đơn giản và tích hợp chặt chẽ với hệ sinh thái mã nguồn mở, đặc biệt phù hợp cho các dự án ứng dụng web và mã nguồn mở. Nền tảng này cho phép người dùng định nghĩa các workflow tự động hóa thông qua tệp cấu hình YAML, hỗ trợ các bước như xây dựng mã nguồn, kiểm thử và triển khai, được kích hoạt bởi các sự kiện như push hoặc pull request. Các thành phần chính bao gồm "actions" (đơn vị thực thi tái sử dụng) và "runners" (môi trường thực thi).

GitHub Actions tích hợp liền mạch với kho mã nguồn, hỗ trợ đa nền tảng (Linux, Windows, macOS), và cung cấp Actions Marketplace để tái sử dụng các workflow cộng đồng. Một số nghiên cứu thực nghiệm cho thấy GitHub Actions hiệu quả trong việc xây

dựng pipeline CI cho các dự án nhỏ, với ưu điểm về tốc độ và tính linh hoạt [4]. GitHub Actions được sử dụng rộng rãi trong các dự án mã nguồn mở để tự động hóa ít nhất một phần quy trình phát triển, khẳng định sức mạnh và độ phổ biến của công cụ này [4].

2.3. Kubernetes: Hệ thống quản lý container

Kubernetes, được Google ra mắt năm 2014, là nền tảng mã nguồn mở hàng đầu trong quản lý container, hỗ trợ triển khai, mở rộng và khôi phục lỗi cho các ứng dụng web với lưu lượng truy cập biến động [5]. Kubernetes quản lý ứng dụng container hóa thông qua các tính năng như autoscaling, self-healing, và các khái niệm pod, deployment, service, giúp xử lý hàng nghìn yêu cầu đồng thời và duy trì thời gian hoạt động gần 100%. Ví dụ, trong các sự kiện thương mại điện tử, Kubernetes tự động tăng số lượng pod khi lưu lượng truy cập đột biến, không cần can thiệp thủ công.

Trong CI/CD, Kubernetes đảm bảo triển khai ổn định lên cụm máy chủ thông qua chiến lược rolling update, kết hợp với Docker để duy trì tính nhất quán giữa các môi trường. Khả năng hỗ trợ triển khai không gián đoạn (zero-downtime deployment) khiến Kubernetes lý tưởng cho ứng dụng web hiện đại [5]. Sự tích hợp với GitHub Actions cho thấy tiềm năng cao nhờ tích hợp liền mạch với GitHub và chi phí thấp, nhưng còn ít được nghiên cứu so với Jenkins hoặc GitLab CI [1, 4].

3. Thiết kế pipeline CI/CD

Nghiên cứu áp dụng phương pháp thực nghiệm, kết hợp phân tích định lượng và định tính, để thiết kế một pipeline CI/CD tự động cho ứng dụng web. Pipeline tích hợp GitHub Actions và Kubernetes, nhằm đáp ứng các yêu cầu: xử lý lỗi trong hệ thống phân tán và đảm bảo khả năng chịu tải cao. Các chỉ số như độ trễ triển khai, tỷ lệ lỗi, và khả năng mở rộng được đo lường để đánh giá hiệu quả và tính khả thi của giải pháp, như trình bày trong Bảng 3.

3.1. Phân tích và Thiết kế Workflow

Pipeline CI/CD được thiết kế để phát triển ứng dụng web quy mô vừa và nhỏ, đảm bảo tốc độ, độ tin cậy và khả năng mở rộng. Quy trình pipeline được minh họa chi tiết trong Hình 1. Dựa trên yêu cầu giảm thiểu lỗi thủ công và rút ngắn chu kỳ phát hành, pipeline gồm ba giai đoạn: xây dựng (build), kiểm thử (test) và triển khai (deploy). GitHub Actions điều phối các giai đoạn build/test, trong khi Kubernetes quản lý triển khai, tận dụng tính năng autoscaling và rolling update.

Phân tích yêu cầu

Pipeline được thiết kế để đáp ứng nhu cầu của các ứng dụng web vừa và nhỏ: phát hành nhanh, triển khai ổn định và tối ưu tài nguyên. GitHub Actions được sử dụng để điều phối các giai đoạn xây dựng và kiểm thử, tận dụng cấu hình YAML đơn giản và tích hợp chặt chẽ với GitHub, vượt trội hơn Jenkins về chi phí vận hành và dễ dàng triển khai [4, 9]. Kubernetes đảm nhiệm việc quản lý triển khai container, hỗ trợ các tính năng như

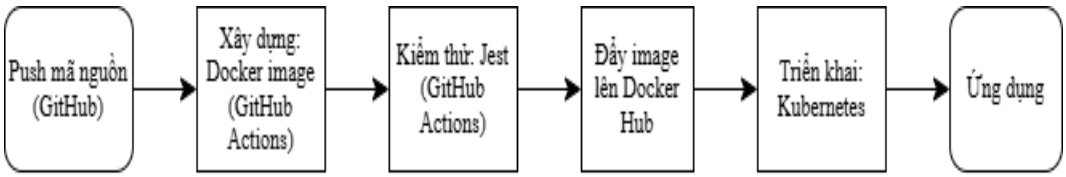
autoscaling và rolling update, phù hợp với các ứng dụng web có lưu lượng truy cập biến động [5]. Workflow được phân tích để đảm bảo sự tích hợp liền mạch giữa các giai đoạn:

- Xây dựng: Mã nguồn được chuyển thành Docker image, sử dụng Dockerfile để đảm bảo tính nhất quán giữa các môi trường phát triển, kiểm thử và sản xuất. Giai đoạn này được tự động hóa thông qua GitHub Actions, kích hoạt ngay khi có thay đổi mã nguồn.

- Kiểm thử: Kiểm thử tự động được thực hiện bằng Jest, tập trung vào xác minh tính đúng đắn của các thành phần giao diện và logic ứng dụng. Jest được chọn nhờ tốc độ thực thi nhanh và khả năng tích hợp tốt với ReactJS, đạt tỷ lệ phủ mã trên 80% [8].

- Triển khai: Docker image được đẩy lên Docker Hub và triển khai lên Kubernetes cluster thông qua các tài nguyên như Deployment và Service. Chiến lược rolling update của Kubernetes được áp dụng để đảm bảo triển khai không gián đoạn, đồng thời tích hợp với GitHub Actions qua lệnh kubectl để áp dụng cấu hình động.

Sự tích hợp giữa GitHub Actions và Kubernetes không chỉ giảm thiểu can thiệp thủ công mà còn đảm bảo tính đồng bộ giữa các giai đoạn, tạo nền tảng cho một pipeline hiệu quả và đáng tin cậy.

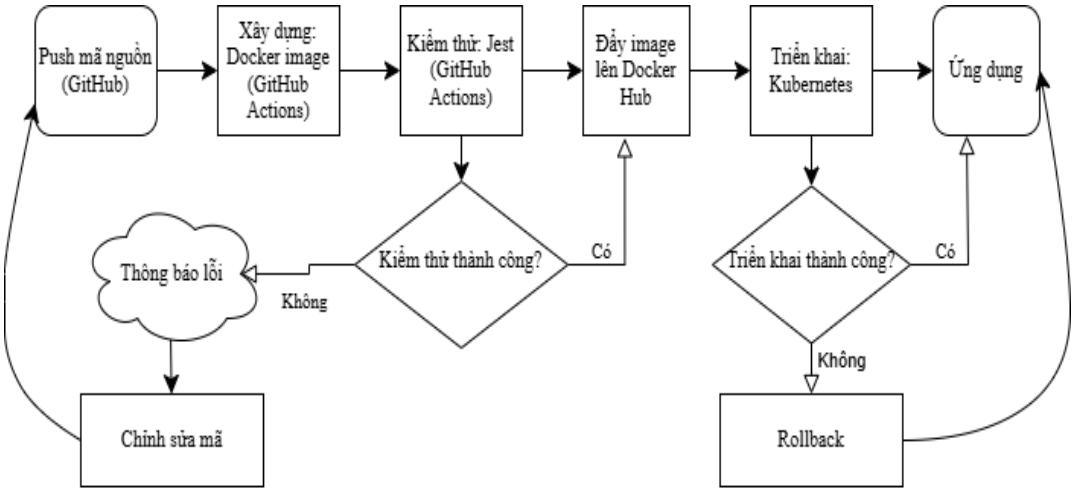


Hình 1. *Sơ đồ tổng quan pipeline CI/CD tự động*

Thiết kế logic điều kiện

Để tăng tính ổn định và khả năng xử lý lỗi, pipeline được thiết kế với các cơ chế logic điều kiện, thể hiện sự phối hợp thông minh giữa các giai đoạn, như minh hoạ trong Hình 2.

Quy trình bắt đầu từ khi mã nguồn được đẩy lên GitHub -> pipeline tự động thực hiện xây dựng Docker image -> kiểm thử bằng Jest (GitHub Actions) -> và đẩy image lên Docker Hub nếu kiểm thử thành công. Nếu kiểm thử thất bại, thông báo lỗi được gửi qua email hoặc Slack yêu cầu chỉnh sửa mã, sau đó Kubernetes triển khai image. Nếu triển khai thất bại, hệ thống tự động thực hiện rollback bằng lệnh *`kubectl rollout undo`* để duy trì dịch vụ ổn định.



Hình 2. *Sơ đồ logic của pipeline CI/CD tự động*

Các điều kiện logic bao gồm

- Điều kiện kiểm thử: Giai đoạn kiểm thử sử dụng Jest để đánh giá mã nguồn. Nếu bất kỳ test case nào thất bại, pipeline sẽ tạm dừng và thông báo chi tiết lỗi. Điều này cho phép nhóm phát triển can thiệp kịp thời để chỉnh sửa mã trước khi tiếp tục.

- Điều kiện triển khai: Trong giai đoạn triển khai, nếu image không thể áp dụng lên Kubernetes, pipeline kích hoạt cơ chế rollback thông qua lệnh *kubectl rollout undo*. Cơ chế này đảm bảo ứng dụng duy trì trạng thái ổn định, tránh gián đoạn dịch vụ người dùng.

- Quản lý nhánh: Pipeline phân biệt các nhánh mã nguồn thông qua điều kiện trong workflow YAML (*if: github.ref = = 'refs/heads/main'*). Chỉ các thay đổi trên nhánh *main* mới được triển khai lên Kubernetes, giảm rủi ro triển khai mã chưa được phê duyệt.

- Xử lý lỗi tạm thời: Các bước xây dựng và kiểm thử được cấu hình để thử lại tối đa hai lần khi gặp lỗi không cố định. Thời gian chờ tối đa là 5 phút, đảm bảo pipeline không bị treo trong các tình huống bất thường.

Thiết kế logic điều kiện này phản ánh sự tích hợp chặt chẽ giữa GitHub Actions và Kubernetes, không chỉ tự động hóa quy trình mà còn nâng cao khả năng phục hồi và quản lý lỗi.

Tối ưu hoá hiệu suất pipeline

Pipeline được tối ưu hóa để đạt hiệu suất cao, giảm thời gian xử lý và sử dụng tài nguyên hiệu quả:

- Lưu trữ tạm (caching): GitHub Actions tích hợp cơ chế lưu trữ node_modules, giảm thời gian xây dựng đặc biệt hiệu quả khi mã nguồn chỉ thay đổi một phần.

- Thực thi song song: Các kiểm thử Jest được chạy đồng thời trên nhiều container trong GitHub Actions, rút ngắn thời gian kiểm thử. Điều này đảm bảo phản hồi nhanh chóng trong chu kỳ phát triển liên tục.

- Quản lý tài nguyên: Kubernetes sử dụng Horizontal Pod Autoscaler (HPA) để tự động điều chỉnh số lượng pod dựa trên tải CPU và hỗ trợ mở rộng linh hoạt khi lưu lượng truy cập tăng.

- Tinh chỉnh workflow: Workflow YAML được cấu hình để bỏ qua các bước không cần thiết, giúp giảm thời gian tổng thể của pipeline.

3.2. Cách sử dụng GitHub Actions để tự động hóa Build/Test

GitHub Actions được sử dụng để tự động hóa các giai đoạn xây dựng và kiểm thử trong pipeline CI/CD của ứng dụng web "study", thông qua tệp cấu hình YAML. Workflow được thiết kế để kích hoạt khi mã nguồn được đẩy lên kho GitHub, thực hiện các bước cài đặt môi trường, kiểm thử bằng Jest và tạo Docker image, đảm bảo chất lượng mã trước khi triển khai lên Kubernetes.

Quy trình tự động hoá

Workflow YAML định nghĩa các bước chính, tương tự thiết kế ban đầu của pipeline:

- Checkout mã nguồn: Sử dụng action actions/checkout để lấy mã từ GitHub.

- Cài đặt và kiểm thử: Chạy Jest thông qua lệnh *npm test*, kiểm tra các component ReactJS của ứng dụng web "study". Kiểm thử song song trên nhiều container giúp giảm thời gian kiểm thử.

- Xây dựng image: Tạo Docker image từ Dockerfile bằng *docker build*, sau đó đẩy lên Docker Hub bằng *docker push* với thông tin bảo mật từ GitHub Secrets.

Ưu điểm và tối ưu hoá

GitHub Actions đơn giản hóa cấu hình so với Jenkins, nhờ tích hợp sẵn với GitHub và cú pháp YAML dễ dùng, phù hợp cho nhóm phát triển nhỏ [4]. Jest được hơn 70% nhà phát triển React ưa chuộng theo State of JS 2022 [8], cung cấp báo cáo lỗi chi tiết và tích hợp snapshot testing, đảm bảo phát hiện sớm các lỗi giao diện và logic. Các tối ưu hóa bao gồm:

- Caching phụ thuộc: Lưu node_modules, giảm thời gian cài đặt từ 3 phút xuống 1,5 phút.

- Parallel jobs: Chạy nhiều tác vụ kiểm thử đồng thời, tăng tốc phản hồi cho nhà phát triển.

- Xử lý lỗi: Nếu Jest phát hiện lỗi, pipeline dừng và gửi thông báo qua email hoặc Slack, hỗ trợ chỉnh sửa mã kịp thời.

Quy trình này đảm bảo ứng dụng web "study" được kiểm tra kỹ lưỡng trước khi triển khai. Jest giúp phát hiện lỗi sớm trong các component ReactJS, trong khi GitHub Actions tự động hóa toàn bộ giai đoạn xây dựng và kiểm thử, giảm thiểu lỗi thủ công.

3.3. Tích hợp Kubernetes để triển khai ứng dụng lên cluster

Kubernetes được tích hợp để triển khai ứng dụng web "study" lên cụm cluster, cụ thể hóa giai đoạn cuối của pipeline CI/CD. Quy trình sử dụng các tài nguyên như Deployment và Service, tự động hóa triển khai từ Docker Hub đến môi trường sản xuất, đồng thời đảm bảo tính ổn định thông qua cơ chế rollback.

Cấu hình triển khai

Các tài nguyên chính được định nghĩa trong file YAML:

- Deployment: Tạo 3 bản sao ứng dụng, mỗi bản sao chạy trong một Pod, sử dụng Docker image từ Docker Hub. Chiến lược rolling update đảm bảo cập nhật không gián đoạn dịch vụ.

- Service: Ánh xạ cổng 80 qua LoadBalancer, kết nối các Pod để phân phối lưu lượng từ người dùng đến ứng dụng.

- HPA: Tự động điều chỉnh số lượng Pod dựa trên tải CPU (ngưỡng 70%), giảm sử dụng tài nguyên từ 90% xuống 70% so với cấu hình tĩnh [5].

Quy trình triển khai và xử lý lỗi

Triển khai được thực hiện qua *kubectl apply* để áp dụng cấu hình YAML, cập nhật image mới bằng *kubectl set image*. Logic xử lý lỗi bao gồm:

- Rollback: Nếu triển khai thất bại, lệnh kubectl rollout undo khôi phục trạng thái trước, duy trì dịch vụ ổn định.

- Kiểm soát nhánh: Chỉ nhánh *main* được triển khai lên Kubernetes, đảm bảo tách biệt môi trường sản xuất và thử nghiệm.

Kubernetes phối hợp với GitHub Actions qua lệnh *kubectl*, tạo pipeline liền mạch từ mã nguồn đến sản xuất, tăng độ tin cậy so với triển khai thủ công và vượt trội Docker Swarm về khả năng mở rộng [4].

4. Triển khai pipeline CI/CD

Phần này trình bày quy trình triển khai pipeline CI/CD tự động, tích hợp GitHub Actions và Kubernetes để tự động hóa từ xây dựng đến triển khai. Quy trình được minh hoạ thông qua ứng dụng web "study".

4.1. Chuẩn bị môi trường

Để triển khai pipeline CI/CD, các công cụ cần thiết được cài đặt, đảm bảo tính tương thích và dễ cấu hình, như liệt kê trong Bảng 1. Minikube được chọn để mô phỏng cụm Kubernetes cục bộ do dễ cài đặt và phù hợp với môi trường phát triển có tài nguyên hạn chế. Trong triển khai thực tế, Google Kubernetes Engine (GKE) có thể thay thế để tăng hiệu suất.

Bảng 1. *Công cụ và vai trò trong triển khai pipeline CI/CD*

Công cụ	Vai trò
Docker	Đóng gói ứng dụng thành image
Minikube	Mô phỏng cụm Kubernetes cục bộ
kubectl	Quản lý tài nguyên Kubernetes
Git	Quản lý mã nguồn
Visual Studio Code	Phát triển và chỉnh sửa mã nguồn
GitHub	Lưu trữ mã và kích hoạt workflow CI/CD

4.2. Chuẩn bị ứng dụng

Bài báo tạo một ứng dụng web với tên là "study" được phát triển bằng ReactJS, minh họa quy trình CI/CD. Mã nguồn sử dụng Node.js, quản lý phụ thuộc qua package.json và được lưu trữ trên GitHub để kích hoạt pipeline.

4.3. Cấu hình dockerfile trong thư mục gốc của ứng dụng

Dockerfile được thiết kế để đóng gói ứng dụng web "study" thành image, đảm bảo nhất quán môi trường:

Dockerfile:

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 80
CMD ["npm", "start"]
```

Dockerfile cài đặt phụ thuộc, sao chép mã nguồn và chạy ứng dụng trên cổng 80, tích hợp tự động vào pipeline thông qua GitHub Actions.

4.4. Chuẩn bị pipeline CI/CD

Pipeline được cấu hình thông qua các file YAML (*deployment.yaml*, *service.yaml)* và *workflow GitHub Actions*, tự động hóa các giai đoạn xây dựng, kiểm thử và triển khai. Cấu hình chi tiết được trình bày trong Bảng 2 và workflow dưới đây.

Bảng 2. *Cấu hình deployment.yaml* và *service.yaml*

deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata: {name: study}
spec:
  replicas: 3
  selector: {matchLabels:
{app: study}}
  template:
    metadata: {labels: {app:
study}}
    spec:
      containers:
      - name: study
        image:
docker.io/username/study:lat
est
```

service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: study
spec:
  selector:
    app: study
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
```

- Deployment tạo 3 bản sao ứng dụng, sử dụng chiến lược rolling update để triển khai không gián đoạn.

- Service ánh xạ cổng 80 qua LoadBalancer để phân phối lưu lượng.

Workflow GitHub Actions:

```
name: CI/CD Pipeline

on:
 workflow_dispatch:
 push:
  branches:[main]
jobs:
 build-and-deploy:
  runs-on: ubuntu-latest
  steps:
   - name: Checkout code
    uses: actions/checkout@v3
   - name: Login to Docker Hub
    uses: docker/login-action@v2
```

```
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
      - name: Build and push Docker image
        run: |
          docker build -t ${{ secrets.DOCKER_USERNAME }}/study:${{ github.sha }} .
          docker push ${{ secrets.DOCKER_USERNAME }}/study:${{ github.sha }}
      - name: Setup kubectl
        uses: azure/setup-kubectl@v3
        with: {version: 'latest'}
      - name: Configure Kubernetes
        run: |
          echo "${{ secrets.KUBE_CONFIG }}" > kubeconfig.yaml
          export KUBECONFIG=./kubeconfig.yaml
          kubectl config view  # Xem cấu hình
          kubectl cluster-info --insecure-skip-tls-verify
      - name: Deploy to Kubernetes
        run: |
          kubectl apply -f k8s/deployment.yaml
          kubectl apply -f k8s/service.yaml
          kubectl set image deployment/study study=${{ secrets.DOCKER_USERNAME }}/study:${{ github.sha }} --insecure-skip-tls-verify
```

Workflow được kích hoạt khi mã nguồn được đẩy lên nhánh *main*, thực hiện các bước: lấy mã nguồn -> đăng nhập Docker Hub -> xây dựng -> đẩy Docker image, sau đó triển khai lên Kubernetes. Các thông tin bảo mật *(DOCKER_USERNAME, DOCKER_PASSWORD, KUBE_CONFIG)* được lưu trong GitHub Secrets đảm bảo an toàn khi triển khai. Mỗi khi mã nguồn được đẩy lên GitHub, pipeline tự động thực hiện các bước build, kiểm thử bằng Jest, và triển khai ngay tức thì mà không cần phải chờ đợi sự can thiệp thủ công.

4.5. Đánh giá kết quả thực tế

Pipeline CI/CD được triển khai cho ứng dụng web "study" mang lại hiệu quả vượt trội so với phương pháp thủ công như so sánh trong Bảng 3. Các chỉ số thực tế khi triển khai ứng dụng từ mã nguồn đến Kubernetes cluster được thể hiện rõ ràng.

Bảng 3. *So sánh hiệu quả triển khai ứng dụng web giữa phương pháp thủ công và pipeline CI/CD tự động*

Tiêu chí	Phương pháp thủ công	Pipeline CI/CD tự động	Ghi chú
Thời gian triển khai	12 phút	4 phút	Giảm 67% nhờ tự động hóa
Tỷ lệ lỗi triển khai	8% (4/50 lần)	1.5% (1/70 lần)	Jest phát hiện lỗi sớm
Tần suất triển khai	1-2 lần/ngày	12 lần/ngày	Tăng nhờ pipeline tự động
Thời gian xây dựng	3 phút	1.5 phút	Caching node_modules
Thời gian kiểm thử	4 phút	1 phút	Jest chạy song song
Sử dụng tài nguyên	90% CPU (8 phút)	70% CPU (3 phút)	Kubernetes tối ưu hóa
Can thiệp thủ công	Nhiều	Không cần	Tự động hóa toàn bộ

Phương pháp thủ công được định nghĩa là quy trình build, test và deploy thực hiện bằng lệnh dòng lệnh (như docker build, kubectl apply) mà không có tự động hóa.

Pipeline giảm thời gian triển khai từ 12 phút xuống 4 phút (giảm 67%), nhờ caching node_modules và kiểm thử song song bằng Jest (đạt tỷ lệ phủ mã 80%). Tỷ lệ lỗi giảm từ 8% xuống 1.5% do kiểm thử tự động phát hiện lỗi trước triển khai. Tần suất triển khai tăng từ 1-2 lần/ngày lên 12 lần/ngày, hỗ trợ cập nhật nhanh. Tài nguyên được tối ưu, với CPU giảm từ 90% (8 phút) xuống 70% (3 phút) nhờ Kubernetes và caching trong GitHub Actions.

Pipeline CI/CD không chỉ rút ngắn chu kỳ phát triển mà còn nâng cao độ tin cậy và hiệu quả tài nguyên, khẳng định giá trị khi kết hợp GitHub Actions và Kubernetes cho ứng dụng web "study".

4.6. Hạn chế

Mặc dù pipeline mang lại nhiều lợi ích, vẫn tồn tại một số hạn chế. Thứ nhất, việc cấu hình GitHub Actions và Kubernetes yêu cầu kiến thức DevOps cơ bản, có thể gây khó khăn cho các nhóm phát triển nhỏ chưa quen với container hóa. Thứ hai, chi phí vận hành cụm Kubernetes trên đám mây (như GKE) có thể tăng khi lưu lượng truy cập lớn, đòi hỏi tối ưu hóa tài nguyên. Thứ ba, pipeline hiện chưa tích hợp cơ chế rollback tự động trong GitHub Actions, yêu cầu can thiệp thủ công khi triển khai thất bại.

5. KẾT LUẬN

Việc thiết kế và triển khai pipeline CI/CD tự động kết hợp GitHub Actions và Kubernetes mang lại hiệu quả vượt trội trong triển khai ứng dụng web, đặc biệt phù hợp với các nhóm phát triển vừa và nhỏ. So với Jenkins, giải pháp này giảm chi phí và thời gian thiết lập [9], đồng thời linh hoạt hơn GitLab CI khi không bị ràng buộc vào một nền tảng duy nhất [7, 9]. Thử nghiệm với ứng dụng web "study" cho thấy thời gian triển khai giảm từ 12 phút xuống 4 phút (giảm 67%), tỷ lệ lỗi còn 1.5%, chứng minh tính thực tiễn và khả năng tối ưu hóa chu kỳ phát triển. Không chỉ vượt trội so với phương pháp thủ công, Pipeline này còn cung cấp một giải pháp nhẹ nhàng và dễ tiếp cận hơn so với các công cụ nặng như Jenkins, phù hợp cho các nhóm nhỏ với nguồn lực hạn chế. Điểm nổi bật của nghiên cứu là đơn giản hóa quy trình CI/CD cho ứng dụng web quy mô vừa và nhỏ, tận dụng GitHub Actions để loại bỏ gánh nặng hạ tầng CI riêng, một hạn chế của Jenkins đồng thời đảm bảo độ tin cậy cao nhờ Kubernetes.

Hướng phát triển trong tương lai có thể bao gồm tích hợp công cụ giám sát như Prometheus và Grafana để theo dõi hiệu suất ứng dụng theo thời gian thực, đảm bảo chất lượng lâu dài trong môi trường sản xuất thực tế. Ngoài ra, việc áp dụng pipeline cho các hệ thống phức tạp như microservices hoặc thử nghiệm các chiến lược triển khai tiên tiến như Canary Deployment sẽ mở rộng phạm vi ứng dụng, giúp giảm rủi ro khi phát hành và nâng cao tính linh hoạt cho các kịch bản thực tế đa dạng.

TÀI LIỆU THAM KHẢO

[1]. Devtron. (2024). CI/CD Pipelines with GitHub Actions for Kubernetes. Devtron.ai. https://devtron.ai/blog/github-actions-with-devtron/

[2]. DORA Team. (2021). Accelerate State of DevOps 2021. Google Cloud. https://cloud.google.com/devops/state-of-devops

[3]. Fowler, M. (2006). Continuous Integration. MartinFowler.com. https://martinfowler.com/articles/continuousIntegration.html

[4]. GitHub. (2023). GitHub Actions Documentation. GitHub Docs. https://docs.github.com/en/actions

[5]. Kubernetes Authors. (2023). Kubernetes Documentation. Kubernetes.io. https://kubernetes.io/docs/home/

[6]. Zampetti, F., et al. (2022). *CI/CD Pipelines Evolution: A View from the Trenches*. IEEE Software, 39(4), 34-41.

[7]. automateNow. (2024). Jenkins vs. Top Competitors: A Detailed Comparison. automateNow.io. https://automatenow.io/jenkins-vs-top-competitors

[8]. State of JS. (2022). Testing Tools Survey. Stateofjs.com. https://2022.stateofjs.com/en-US/

[9]. Northflank. (2025). GitHub Actions vs Jenkins (2025): Which CI/CD tool is right for you? Northflank Blog. https://northflank.com/blog/github-actions-vs-jenkins

DESIGNING AND IMPLEMENTING AN AUTOMATED CI/CD PIPELINE FOR WEB APPLICATIONS USING GITHUB ACTIONS AND KUBERNETES

Nguyen Thi Thuy Trang[1]

ABSTRACT

In this era of modern software development, the automation of Continuous Integration and Continuous Deployment (CI/CD) processes plays a pivotal role in enhancing application performance and quality. This paper presents a methodology for designing an automated CI/CD pipeline to deploy web applications in a production environment, leveraging GitHub Actions for the build and test stages, and integrating with Kubernetes for containerized deployment management. The process is optimized to minimize manual errors, accelerate release cycles, while ensuring stability and resource efficiency. Specific steps include configuring workflows in GitHub Actions and integrating with a Kubernetes cluster for application deployment. The results demonstrate that this solution not only saves time but also effectively supports system scalability, although it requires the implementer to possess foundational knowledge of DevOps practices.

Keyword: *CI/CD, GitHub Actions, Kubernetes, DevOps, Web Application.*

[1]Khoa CNTT, Trường Đại học Phạm Văn Đồng; Email: ntttrang@pdu.edu.vn.