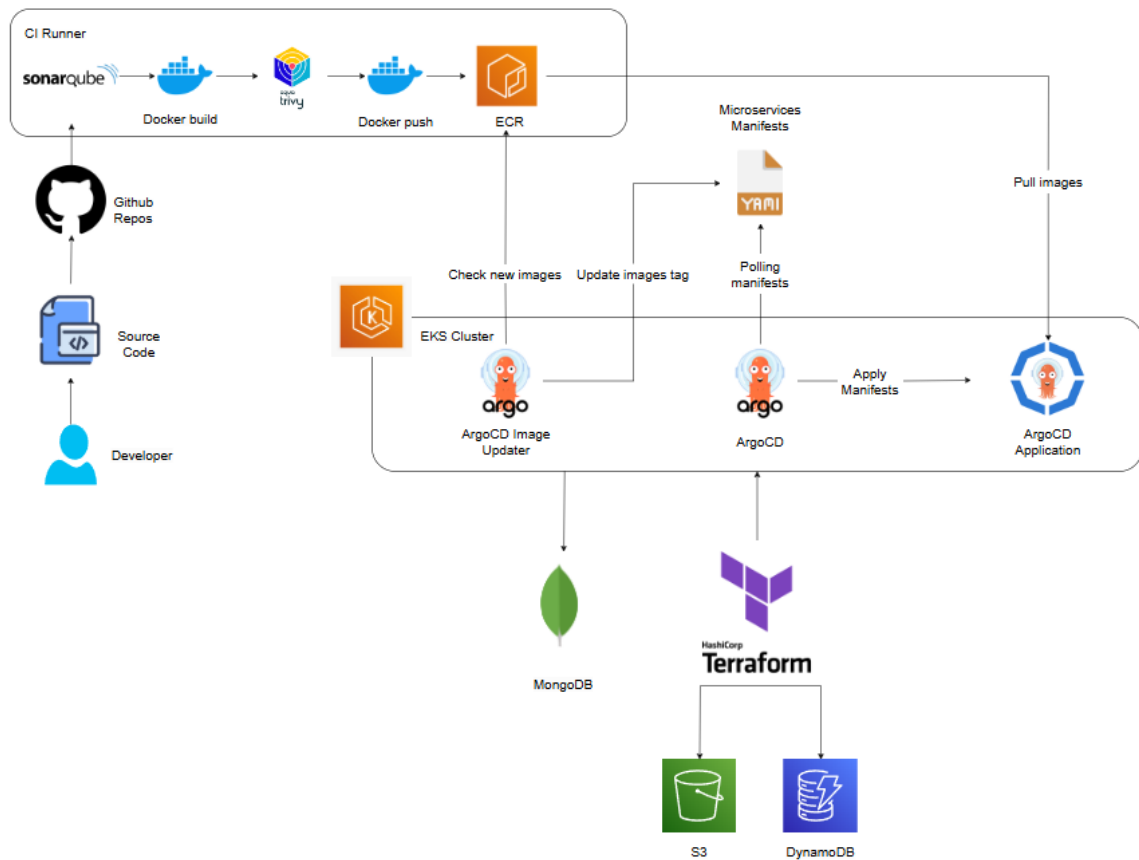


I. Mô hình triển khai

Pipeline được thiết kế với các giai đoạn sau:



Hình 1 Mô hình triển khai

- Phân tích mã nguồn với SonarQube giúp đánh giá và cải thiện chất lượng phần mềm bằng cách phát hiện lỗi lập trình, lỗ hổng bảo mật và các vấn đề liên quan đến hiệu suất mã.
- Xây dựng Docker hình ảnh và sử dụng Trivy để quét hình ảnh nhằm phát hiện lỗ hổng trong các lớp, thư viện hoặc tệp cấu hình.
- Sau khi quét bảo mật thành công, hình ảnh được push lên **Amazon ECR**, sẵn sàng cho bước triển khai.
- Các tệp cấu hình deployment YAML sẽ được cập nhật và đẩy lên Git. **Argo CD** sẽ tự động cập nhật Docker Image mới và đồng bộ cấu hình từ Git lên cụm **Amazon EKS**, đảm bảo ứng dụng được triển khai đúng phiên bản và đúng cấu hình một cách tự động
- Cấu hình cụm EKS và các tài nguyên đám mây được quản lý dưới dạng mã với **Terraform** nhằm kiểm soát phiên bản và tự động hóa.

II. Triển Khai Hệ Thống

1. Thiết lập môi trường và các công cụ CI/CD

a. Dockerfile

```
1 FROM node AS build
2
3 WORKDIR /app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 FROM node:20-alpine
12
13 WORKDIR /app
14
15 COPY --from=build /app /app
```

Hình 2 Dockerfile

Dockerfile được thiết kế theo kiến trúc multi-stage build, giúp tối ưu hóa kích thước image cuối cùng và tăng hiệu quả triển khai ứng dụng Node.js. Tập này tách biệt hai giai đoạn chính: build và runtime, đảm bảo môi trường chạy thực tế nhẹ và bảo mật hơn.

- Giai đoạn 1: Build

FROM node AS build

Sử dụng image node mặc định (latest) để tạo môi trường build và đặt tên cho stage là build.

WORKDIR /app

Thiết lập thư mục làm việc là app.

COPY package*.json ./

Sao chép các tệp package.json và package-lock.json vào container.

RUN npm install

Cài đặt các dependencies được định nghĩa trong package.json.

COPY ..

Sao chép mã nguồn vào container.

- Giai đoạn 2: Runtime

Sử dụng image node:20-alpine là phiên bản nhỏ gọn hơn.

FROM node:20-alpine AS production

Thiết lập thư mục làm việc app.

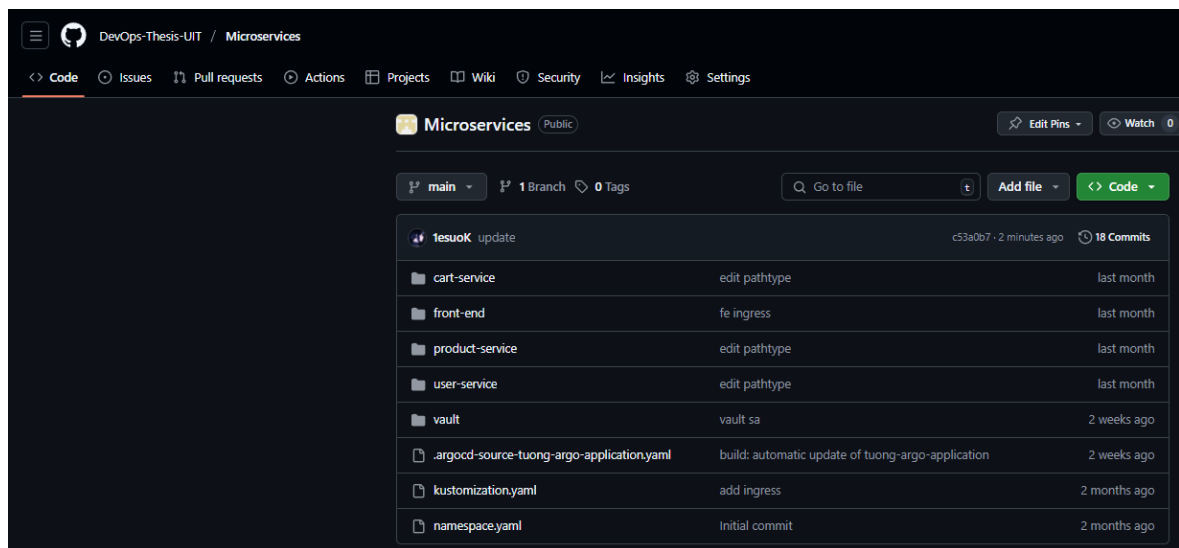
WORKDIR /app

Sao chép toàn bộ nội dung từ giai đoạn build sang giai đoạn runtime.

COPY --from=build /app /app

b. K8s Manifest

Các tệp manifest dành cho hướng triển khai GitHub Actions được khai báo ở đường dẫn <https://github.com/DevOps-Thesis-UIT/Microservices.git>



Hình 3 Kubernetes Manifests Github

Các tệp manifest dành cho hướng triển khai GitLab CI được khai báo ở đường dẫn <https://gitlab.com/nt505/manifest.git>

NT505 / Manifest

master manifest +

manifest

Find file Edit Code

Update file kustomization.yml
nhutlinh231 authored 11 hours ago

3fb1f7db History

Name	Last commit	Last update
cart-service	Update file deployment.yml	11 hours ago
front-end	Edit deployment.yml	14 hours ago
product-service	Update 4 files	3 days ago
user-service	Update 4 files	3 days ago
vault	Update file kustomization.yml	11 hours ago
.argocd-source-nt505.yaml	build: automatic update of nt505	14 hours ago
kustomization.yml	Update file kustomization.yml	11 hours ago
namespace.yml	feat: init project	5 months ago

Hình 4 Kubernetes Manifests Gitlab

c. Khởi tạo Runner

Tạo một EC2 Instance để self-host Github Actions Runner.

Instances (1) Info

Last updated less than a minute ago Connect

Find Instance by attribute or tag (case-sensitive)

All states

Name = Runner X Clear filters

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input type="checkbox"/>	Runner	i-075189192c9b0a2ea	Running	t2.medium	2/2 checks passed	View alarms	us-east-1b

Hình 5 Runner Server

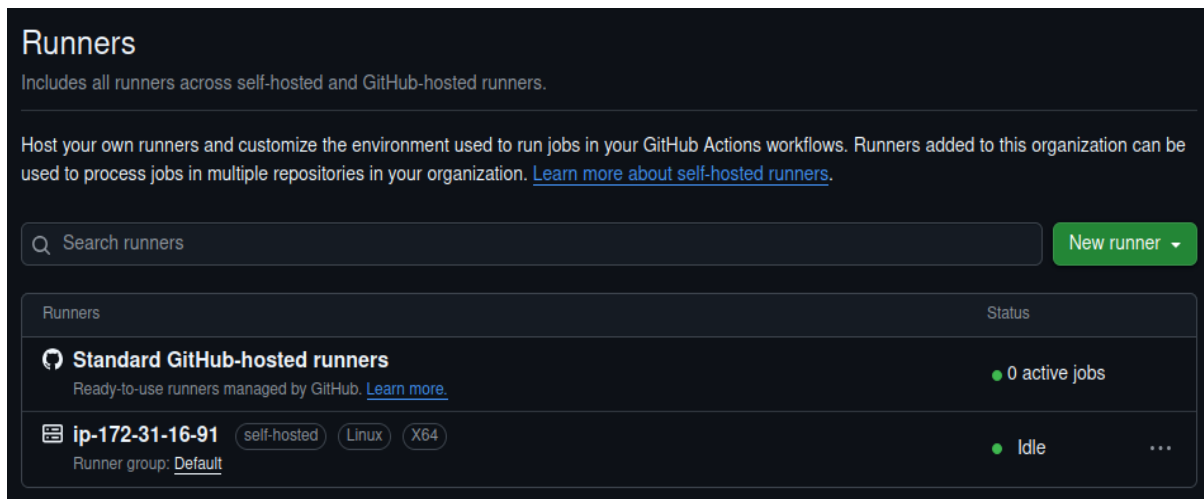
Chạy các lệnh sau để tải runner trên môi trường Linux.

```
$ mkdir actions-runner && cd actions-runner
$ curl -o actions-runner-linux-x64-2.324.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.324.0/actions-runner-
linux-x64-2.324.0.tar.gz
$ tar xzf ./actions-runner-linux-x64-2.324.0.tar.gz
```

Sau đó thiết lập cấu hình và thiết lập runner chạy như một service.

```
$ ./config.sh --url https://github.com/DevOps-Thesis-UIT --token {your_token}
$ sudo ./svc.sh install
$ sudo ./svc.sh start
```

Cài đặt thành công runner.

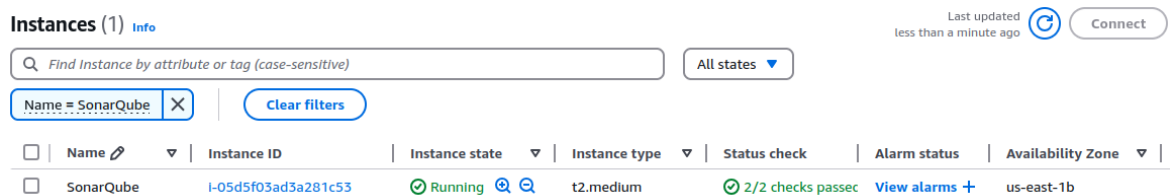


Hình 6 Kết quả cài đặt runner

Sau khi cài đặt thành công runner, chạy tệp script bao gồm các lệnh cài đặt Docker, OpenSCAP CLI, Ansible, Python và thư viện beautifulsoup cho runner để chạy workflow.

d. Khởi tạo SonarQube

Tạo một EC2 Instance để host SonarQube



Hình 7 SonarQube Server

Nhóm cài đặt SonarQube bằng cách sử dụng một tệp script nhóm tự viết dựa trên hướng dẫn của SonarQube. Tệp Script bao gồm việc cài đặt Docker, Docker Compose, khởi tạo và chạy SonarQube sử dụng tệp docker-compose gồm SonarQube và PostgreSQL container

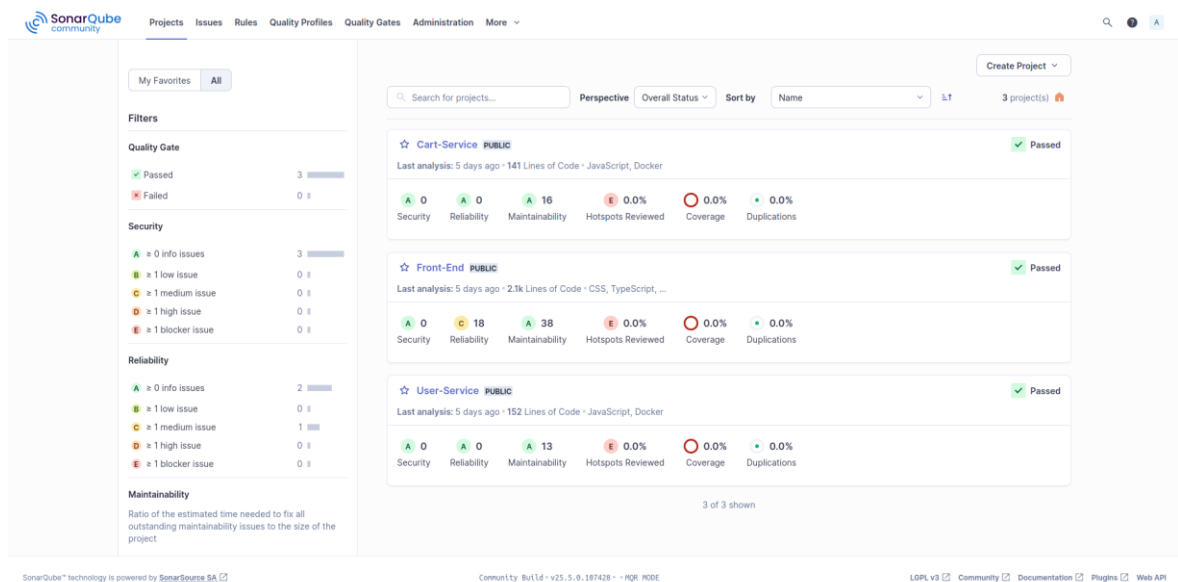
```

1  version: "3"
2
3  services:
4    sonarqube:
5      image: sonarqube:community
6      depends_on:
7        - db
8      environment:
9        SONAR_SEARCH_JAVA_OPTS: "-Xms1G -Xmx1G"
10       SONAR_WEB_JAVA_OPTS: "-Xmx1G -Xms256m"
11       SONAR_SCANNER_OPTS: "-Xmx1G -Xms1G"
12       SONAR_CE_JAVA_OPTS: "-Xmx1G -Xms1G"
13       SONAR_JDBC_URL: jdbc:postgresql://db:5432/sonar
14       SONAR_JDBC_USERNAME: sonar
15       SONAR_JDBC_PASSWORD: sonar
16     volumes:
17       - sonarqube_data:/opt/sonarqube/data
18       - sonarqube_extensions:/opt/sonarqube/extensions
19       - sonarqube_logs:/opt/sonarqube/logs
20     ports:
21       - "9000:9000"
22
23   db:
24     image: postgres:12
25     environment:
26       POSTGRES_USER: sonar
27       POSTGRES_PASSWORD: sonar
28     volumes:
29       - postgresql:/var/lib/postgresql
30       - postgresql_data:/var/lib/postgresql/data
31
32   volumes:
33     sonarqube_data:
34     sonarqube_extensions:
35     sonarqube_logs:
36     postgresql:
37     postgresql_data:

```

Hình 8 Docker-compose của Sonar

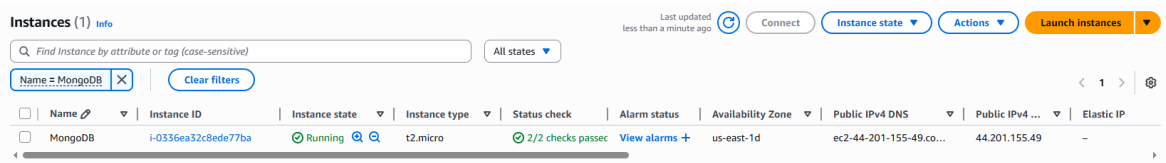
Giao diện của SonarQube khi cài đặt thành công



Hình 9 Giao diện SonarQube

e. Khởi tạo MongoDB Server

Tạo EC2 Instance cho MongoDB Server trên AWS



Hình 10 MongoDB Server

Sau đó, sử dụng một tệp script cài đặt MongoDB 8.0 trên Ubuntu bằng cách thêm kho lưu trữ chính thức của MongoDB, cài đặt gói mongodb-org, sau đó khởi động và kích hoạt dịch vụ MongoDB.

Cài đặt thành công MongoDB

```
ubuntu@ip-172-31-92-86:~$ sudo systemctl status mongod
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset:
   Active: active (running) since Tue 2025-05-27 13:16:07 UTC; 2min 4s ago
     Docs: https://docs.mongodb.org/manual
    Main PID: 1963 (mongod)
      Memory: 182.3M
         CPU: 1.426s
    CGroup: /system.slice/mongod.service
            └─1963 /usr/bin/mongod --config /etc/mongod.conf

May 27 13:16:07 ip-172-31-92-86 systemd[1]: Started MongoDB Database Server.
May 27 13:16:08 ip-172-31-92-86 mongod[1963]: {"t":{"$date":"2025-05-27T13:16:0
lines 1-12/12 (END)
```

Hình 11 Trạng thái MongoDB Server

Chỉnh sửa cấu hình mongo tại /etc/mongod.conf để bật tính năng “authorization” và thay đổi “network interfaces”

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /var/lib/mongodb
#   engine:
#   wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:
security:
  authorization: enabled

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options:

#auditLog:
```

Hình 12 Tập cấu hình MongoDB

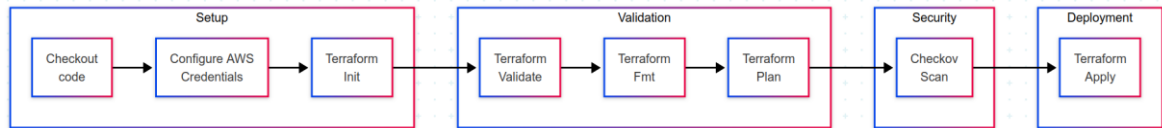
Tạo user bằng các lệnh

```
$ mongosh
test> use admin
admin> db.createUser({
...     user: "{username}",
...     pwd: "{password}",
...     roles: [ { role: "root", db: "admin" } ]
... })
```

Sau đó import data từ tệp products.json

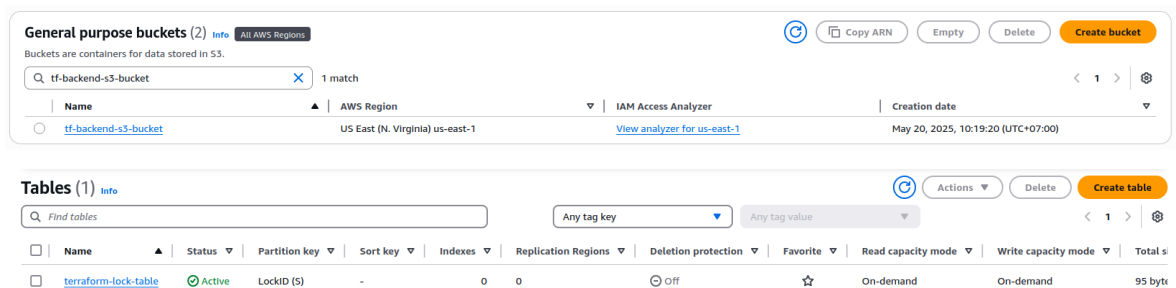

```
mongoimport --uri "mongodb://{username}:{password}@{host
ip}:27017/product_service?authSource=admin" --collection products --tệp
products.json --jsonArray
```

f. Tự động hoá triển khai EKS bằng Terraform



Hình 13 Mô hình triển khai EKS bằng Terraform

Tạo S3 bucket và DynamoDB table để làm backend cho terraform



Hình 14 S3 Bucket và DynamoDB Table cho Terraform

Sử dụng một custom module để khởi tạo EKS kết hợp với VPC Official Module của AWS trên Terraform Registry

2. Triển khai GitHub Actions

a. Tạo workflow tích hợp OpenSCAP



Hình 15 Job Build & Push Docker Image

Job này thực hiện phân tích chất lượng mã nguồn, quét bảo mật image và xây dựng docker image để đẩy lên ECR Registry. Job gồm bước chính:

- + SonarQube: Sử dụng SonarQube để phân tích chất lượng mã nguồn.
- + Cấu hình AWS Credentials: Thiết lập thông tin xác thực AWS và đăng nhập vào AWS ECR.
- + Build docker image: thực hiện xây dựng docker image.
- + Trivy Scan: Quét các lỗ hổng trong image đã xây dựng bằng công cụ Trivy
- + Push docker image: Đẩy docker image đã xây dựng lên AWS ECR.

b. Triển khai liên tục với GitOps

- **Argo CD Configuration & Nginx Ingress Controller**

Cài đặt Argo CD lên cụm kubernetes bằng các lệnh

```
kubectl create namespace Argo CD
```

```
kubectl apply -n Argo CD -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Expose Argo CD bằng Load Balancer

```
kubectl patch svc Argo CD-server -n Argo CD -p '{"spec": {"type": "LoadBalancer"}}'
```

Để deploy tạo một tệp application.yaml trỏ tới Repos lưu trữ Kubernetes Manifests

```

ArgoCD > application.yaml > {} metadata > namespace
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: tuong-argo-application
5    namespace: argocd
6  spec:
7    project: default
8    source:
9      repoURL: 'https://github.com/DevOps-Thesis-UIT/Microservices.git'
10     path: .
11     targetRevision: HEAD
12   destination:
13     namespace: microservices
14     server: 'https://kubernetes.default.svc'
15
16   syncPolicy:
17     syncOptions:
18       - CreateNamespace=true
19     automated:
20       selfHeal: true
21       prune: true

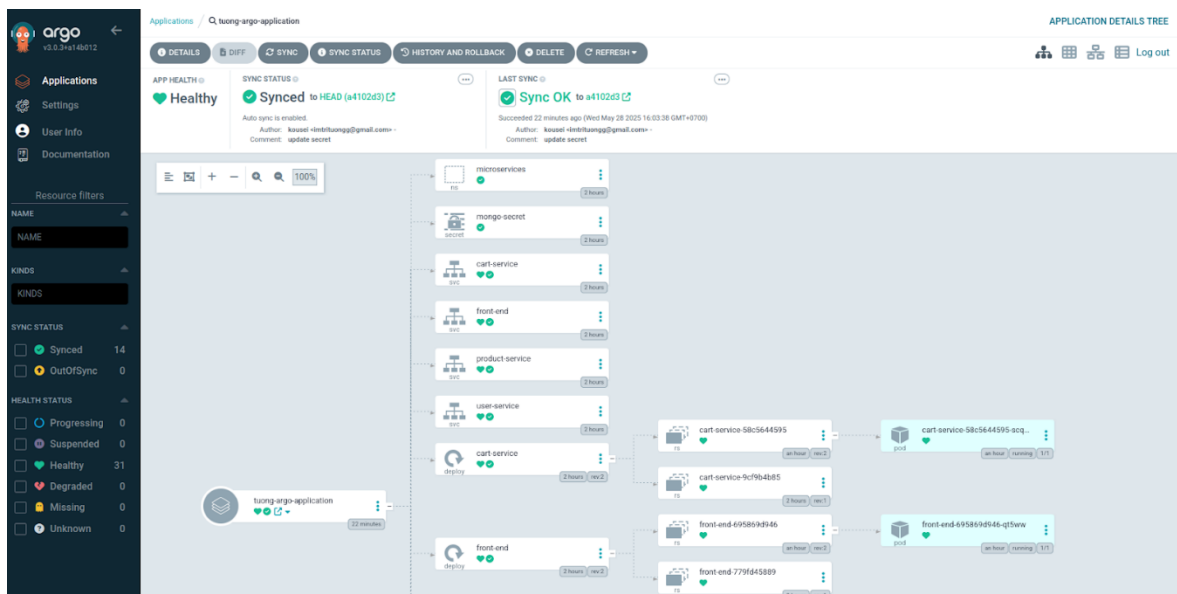
```

Hình 16 Argo CD application.yaml

Khi apply deployment chỉ cần chạy đơn giản lệnh

kubectl apply -f application.yaml

Argo CD sẽ tự động triển khai theo những gì được cấu hình trong Manifests



Hình 17 Argo CD tự động triển khai

Cài đặt Nginx Ingress Controller

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

```
helm install ingress-nginx ingress-nginx/ingress-nginx \
--namespace ingress-nginx --create-namespace
```

Kiểm tra cài đặt ingress-nginx

```
kousei .../KLTN/DevOps-Thesis/ArgoCD 15:03 kubectl get pods -n ingress-nginx
NAME                                     READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-5b498b5b49-szlr 1/1     Running   0           52m

kousei .../KLTN/DevOps-Thesis/ArgoCD 15:04 kubectl get svc -n ingress-nginx
NAME                                     TYPE           CLUSTER-IP      EXTERNAL-IP
ingress-nginx-controller                LoadBalancer   192.168.226.221  ae8a9803941f449fa
ingress-nginx-controller-admission     ClusterIP       192.168.142.135  <none>
```

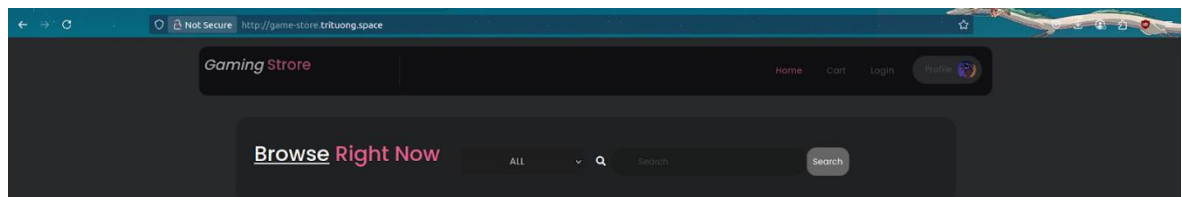
Hình 18 Kiểm tra cài đặt nginx

Sau đó cấu hình External IP của ingress-nginx-controller với domain

CNAME	game-store	0	ae8a9803941f449faba04d4ba24639dc-2063105622.us-east-1.elb.amazonaws.com
-------	------------	---	---

Hình 19 Cấu hình DNS với ingress-nginx-controller

Ứng dụng triển khai thành công (Không SSL/TLS)



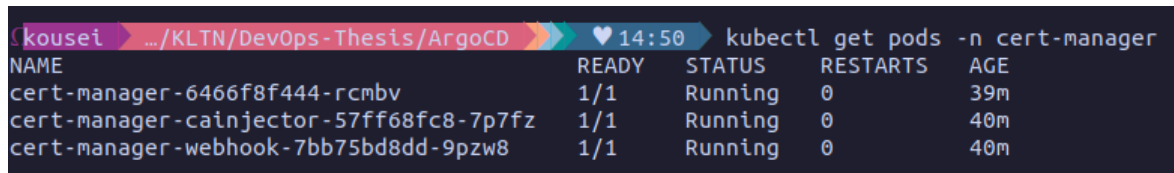
Hình 20 Ứng dụng chưa có SSL/TLS

- Cert-manager

Cài đặt Cert-Manager bằng lệnh

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
helm install cert-manager jetstack/cert-manager \
--namespace cert-manager --create-namespace \
--version v1.12.0 \
--set installCRDs=true
```

Cert-Manager chạy thành công



kousei .../KLTN/DevOps-Thesis/ArgoCD 14:50 kubectl get pods -n cert-manager				
NAME	READY	STATUS	RESTARTS	AGE
cert-manager-6466f8f444-rcmbv	1/1	Running	0	39m
cert-manager-cainjector-57ff68fc8-7p7fz	1/1	Running	0	40m
cert-manager-webhook-7bb75bd8dd-9pzw8	1/1	Running	0	40m

Hình 21 Trạng thái Cert-Manager

Tạo một ClusterIssuer để nhận certificates từ Let's Encrypt, sau đó apply vào cụm.

```
# cluster-issuer.yaml

apiVersion: cert-manager.io/v1

kind: ClusterIssuer

metadata:

  name: letsencrypt-prod

spec:

  acme:

    server: https://acme-v02.api.letsencrypt.org/directory

    email: your-email@example.com

    privateKeySecretRef:

      name: letsencrypt-prod

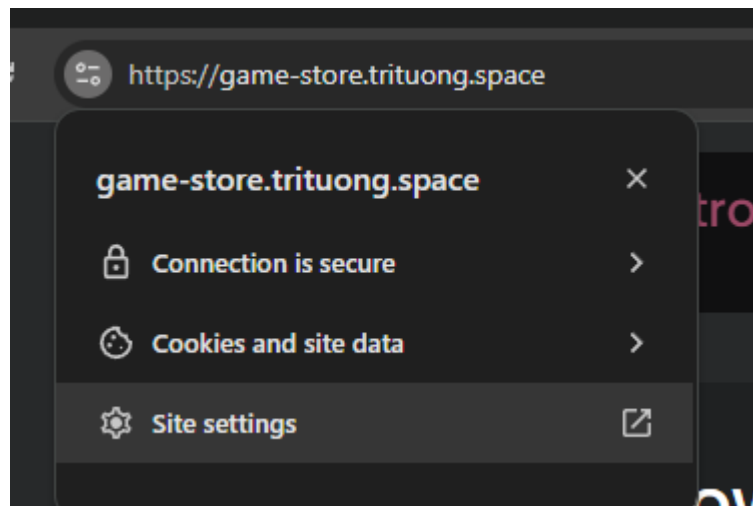
    solvers:

      - http01:

          ingress:

            class: nginx
```

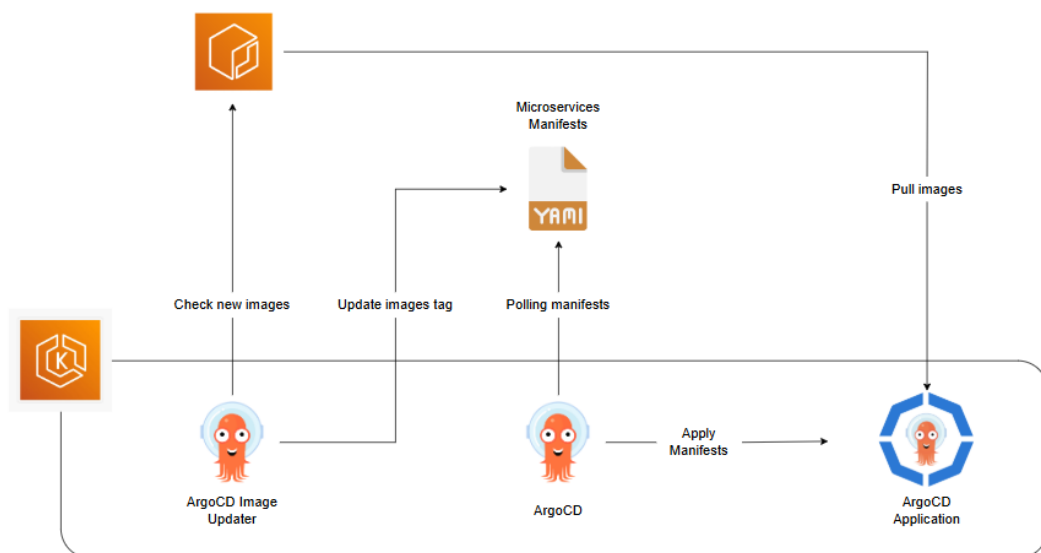
Truy cập web kiểm tra



Hình 22 Cert của ứng dụng

- Argo CD Image Updater

Cơ chế hoạt động của Argo CD Image Updater

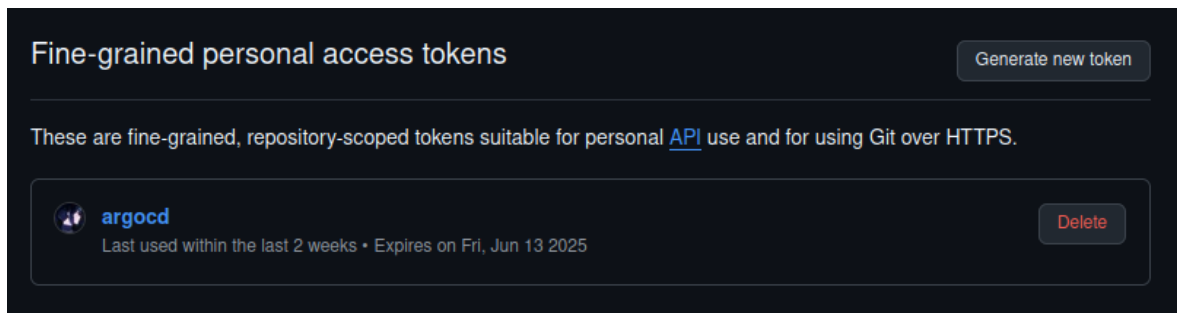


Hình 23 Cơ chế hoạt động Argo CD Image Updater

Cài đặt Image Updater lên cụm kubernetes

```
kubectl apply -n Argo CD -f https://raw.githubusercontent.com/argoproj-labs/Argo-CD-image-updater/stable/manifests/install.yaml
```

Truy cập Github để tạo một Credentials cho Argo CD quyền chỉnh sửa repos



Hình 24 Access Tokens cho Argo CD

Dùng Credentials vừa tạo để tạo một secret trong cụm kubernetes

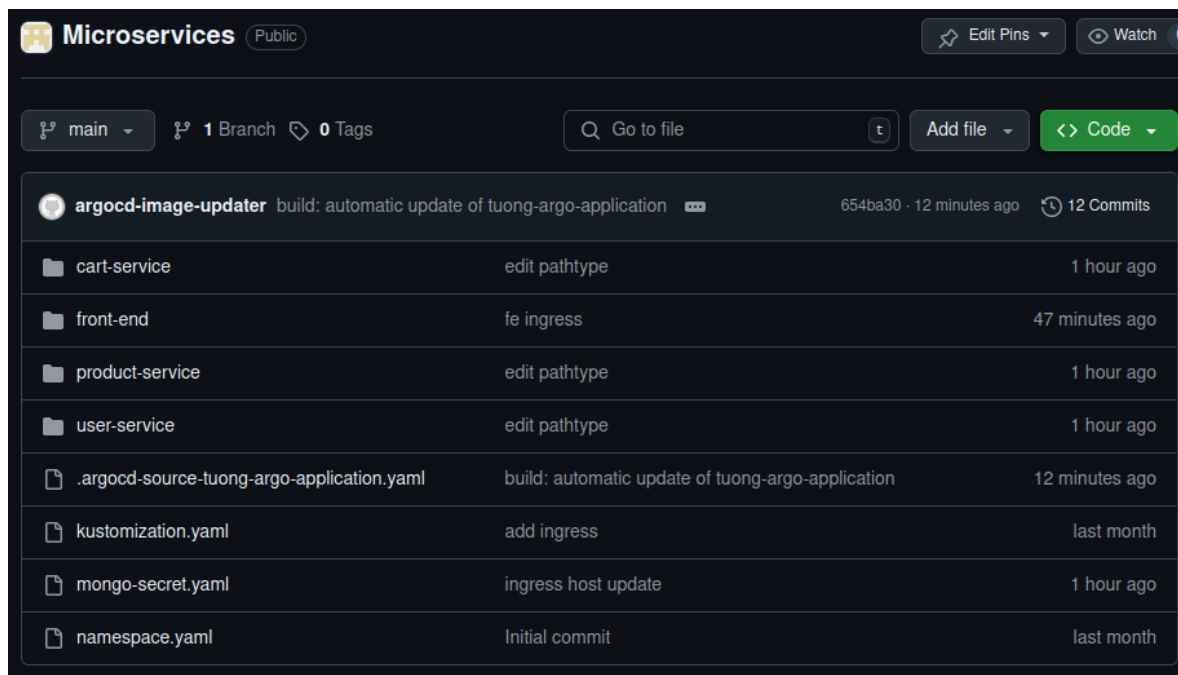
```
kubectl -n Argo CD create secret generic git-creds \
--from-literal=username=<user_name> \
--from-literal=password=<access_token>
```

Update annotations của Argo CD application gồm các repos và update strategy của từng repos

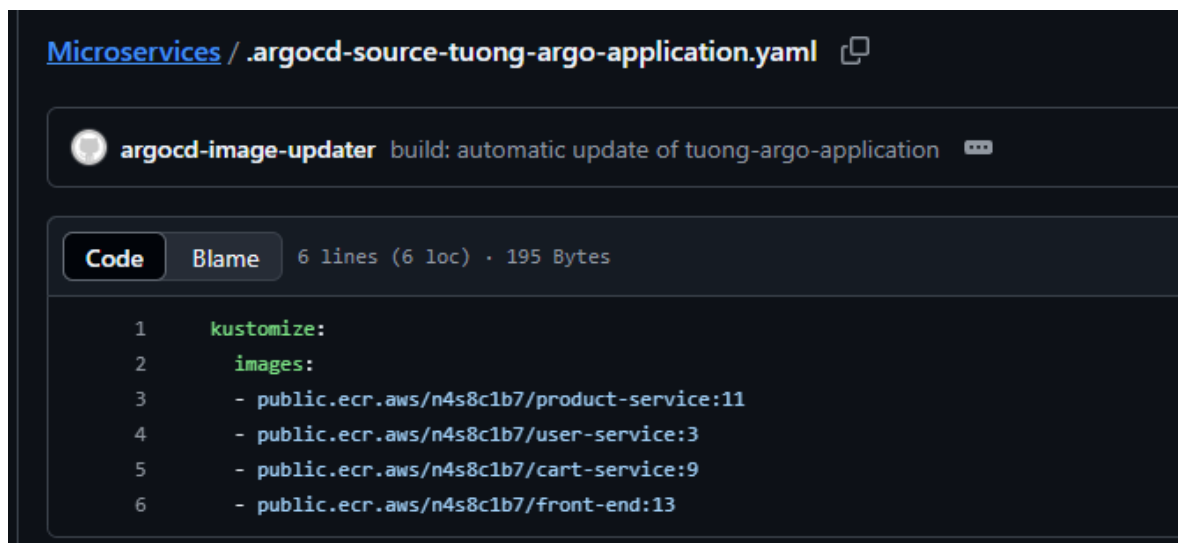
```
ArgoCD > application.yaml > {} spec
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: tuong-argo-application
5    namespace: argocd
6    annotations:
7      argocd-image-updater.argoproj.io/image-list: |
8        product=public.ecr.aws/n4s8clb7/product-service,
9        user=public.ecr.aws/n4s8clb7/user-service,
10       cart=public.ecr.aws/n4s8clb7/cart-service,
11       frontend=public.ecr.aws/n4s8clb7/front-end
12
13      argocd-image-updater.argoproj.io/product.update-strategy: newest-build
14      argocd-image-updater.argoproj.io/user.update-strategy: newest-build
15      argocd-image-updater.argoproj.io/cart-service.update-strategy: newest-build
16      argocd-image-updater.argoproj.io/frontend.update-strategy: newest-build
17
18      argocd-image-updater.argoproj.io/write-back-method: git:secret:argocd/git-creds
19  spec:
20    project: default
21    source:
22      repoURL: 'https://github.com/DevOps-Thesis-UIT/Microservices.git'
23      path: .
24      targetRevision: HEAD
25    destination:
26      namespace: microservices
27      server: 'https://kubernetes.default.svc'
28
29    syncPolicy:
30      syncOptions:
31        - CreateNamespace=true
32      automated:
33        selfHeal: true
34        prune: true
```

Hình 25 Cập nhật annotations cho application.yml

Khi cấu hình thành công Argo CD-image-updater sẽ tự động cập nhật image với tag mới nhất tại tệp .Argo CD-source-tuong-argo-application.yaml



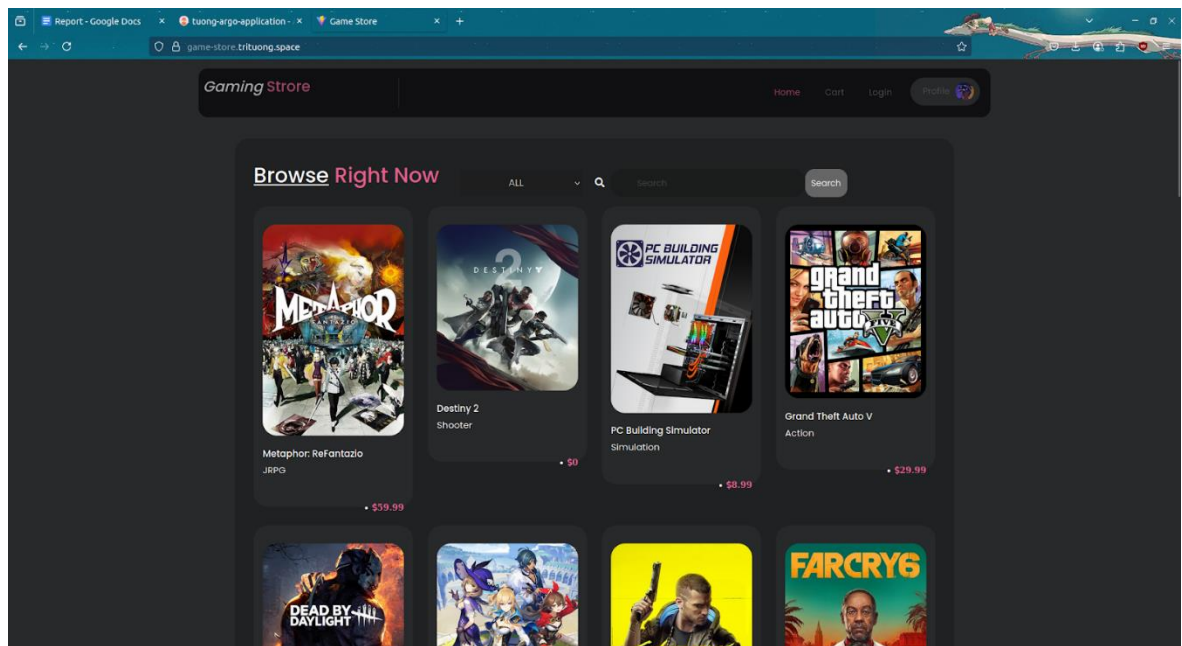
Hình 26 Argo CD git commit



Hình 27 Nội dung Argo CD cập nhật

- Kết quả

Triển khai thành công ứng dụng trên nền tảng kubernetes sử dụng Argo CD cùng với thiết lập SSL/TLS và Image Updater để tự động cập nhật image mới.



Hình 28 Ứng dụng đã triển khai thành công