

# **BÁO CÁO BÀI TẬP LỚN CUỐI KỲ CHƯƠNG TRÌNH VDT 2024 LĨNH VỰC CLOUD – GIAI ĐOẠN 1**

**Sinh viên: Lê Minh Việt Anh – Đại học Bách khoa Hà Nội**

## 0 Môi trường thực hành:

Máy ảo Ubuntu desktop 24.04 được tạo trên Virtual Box (4 CPUs, 4 GB RAM, 80 GB ổ cứng) để triển khai Kubernetes.

Máy ảo Ubuntu server 22.04 được tạo trên Virtual Box (1 CPU, 2 GB RAM, 15 GB ổ cứng) để triển khai HAProxy.

## 1 Triển khai Kubernetes

**Yêu cầu** Triển khai được Kubernetes qua công cụ minikube trên 1 node, hoặc qua công cụ kubeadm hoặc kubespray lên 1 master node VM + 1 worker node VM

**Output** Tài liệu cài đặt  
Log của các lệnh kiểm tra hệ thống: **kubectl get nodes –o wide**

### 1.1 Tài liệu cài đặt Kubernetes qua công cụ minikube trên 1 node

Cần cài đặt: Docker, Kubernetes, minikube

#### 1.1.1 Cài đặt Docker

Bước 1: Cài đặt các package cần thiết

```
sudo apt update  
sudo apt install apt-transport-https curl
```

Bước 2: Thêm khóa công khai chính thức của Docker lên máy Ubuntu

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o  
/etc/apt/keyrings/docker.gpg
```

Bước 3: Cài đặt kho lưu trữ của Docker để cài đặt và cập nhật các phiên bản ổn định của Docker

```
echo "deb [arch=$(dpkg --print-architecture) signed -  
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(  
/etc/os-release && echo "$VERSION_CODENAME") stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null
```

Bước 4: Làm mới kho lưu trữ

```
sudo apt update
```

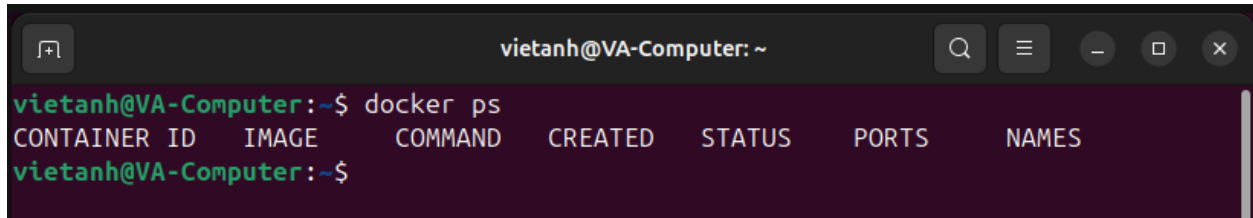
Bước 5: Cài đặt Docker Engine

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin  
docker-compose-plugin
```

Bước 6: Cho phép người dùng không có quyền root sử dụng Docker

```
sudo usermod -aG docker vietanh (có thể thay bằng username khác)
```

Bước 7: Gõ lệnh docker ps để kiểm tra



```
viétanh@VA-Computer: ~  
viétanh@VA-Computer:~$ docker ps  
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES  
viétanh@VA-Computer:~$
```

### 1.1.2 Cài đặt kubecel

Bước 1: Cài đặt file nhị phân kubectl bằng curl

```
curl -LO https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
```

Bước 2: Tải file checksum kubectl

```
curl -LO https://dl.k8s.io/release/$(curl -L -s  
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256
```

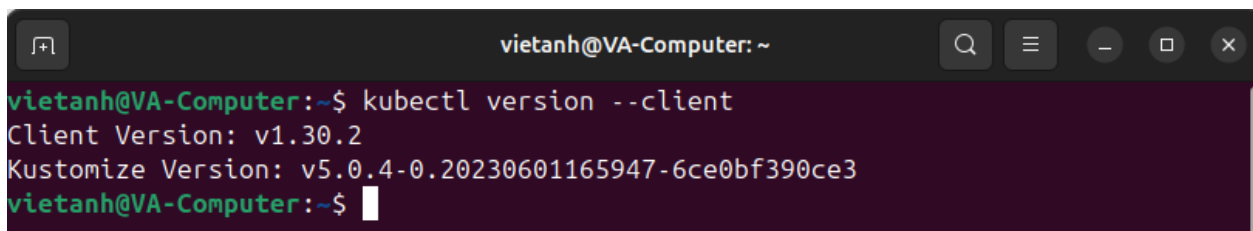
xác thực file nhị phân đã tải bằng file checksum, kết quả hiển thị OK

```
echo "$(cat kubectl.sha256) kubectl" | sha256sum --check
```

Bước 3: Cài đặt kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Bước 4: Kiểm tra cài đặt kubectl thành công



```
viétanh@VA-Computer:~$ kubectl version --client  
Client Version: v1.30.2  
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3  
viétanh@VA-Computer:~$
```

### 1.1.3 Cài đặt minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-  
amd64  
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-  
amd64
```

Chạy minikube

```
viethanh@VA-Computer: ~  
viethanh@VA-Computer:~$ minikube start  
🌟 minikube v1.33.1 on Ubuntu 24.04 (vbox/amd64)  
🌟 Using the docker driver based on existing profile  
👍 Starting "minikube" primary control-plane node in "minikube" cluster  
📦 Pulling base image v0.0.44 ...  
🔄 Restarting existing docker container for "minikube" ...  
📦 Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...  
🔍 Verifying Kubernetes components...  
   ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5  
🌟 Enabled addons: storage-provisioner, default-storageclass  
📦 Done! kubectl is now configured to use "minikube" cluster and "default" name  
space by default  
viethanh@VA-Computer:~$
```

Log các lệnh kiểm tra hệ thống

```
viethanh@VA-Computer: ~  
viethanh@VA-Computer:~$ kubectl get nodes -o wide  
NAME          STATUS    ROLES          AGE    VERSION    INTERNAL-IP    EXTERNAL-IP  
OS-IMAGE      KERNEL-VERSION CONTAINER-RUNTIME  
minikube      Ready     control-plane   28h    v1.30.0    192.168.49.2   <none>  
Ubuntu 22.04.4 LTS 6.8.0-35-generic docker://26.1.1  
viethanh@VA-Computer:~$
```

Ta thấy minikube đã được triển khai trong máy ảo, với địa chỉ IP giao tiếp với máy ảo là 192.168.49.2/24

## 2 K8S Helm Charts

**Yêu cầu 1** Cài đặt ArgoCD lên Kubernetes Cluster, expose được ArgoCD qua NodePort

**Output 1** File manifests sử dụng để triển khai ArgoCD lên K8S Cluster  
Ảnh chụp màn hình giao diện hệ thống ArgoCD khi truy cập qua trình duyệt

### 2.1 Cài đặt ArgoCD

Bước 1: Tạo namespace argocd

```
kubectl create namespace argocd
```

Bước 2: Tải argo trên namespace argocd

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

Sử dụng file manifest có sẵn trên: <https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml>

## 2.2 Expose ArgoCD qua NodePort

Bước 1: Đổi Service Type của argocd-server từ ClusterIP thành NodePort để có thể truy cập từ bên ngoài cluster

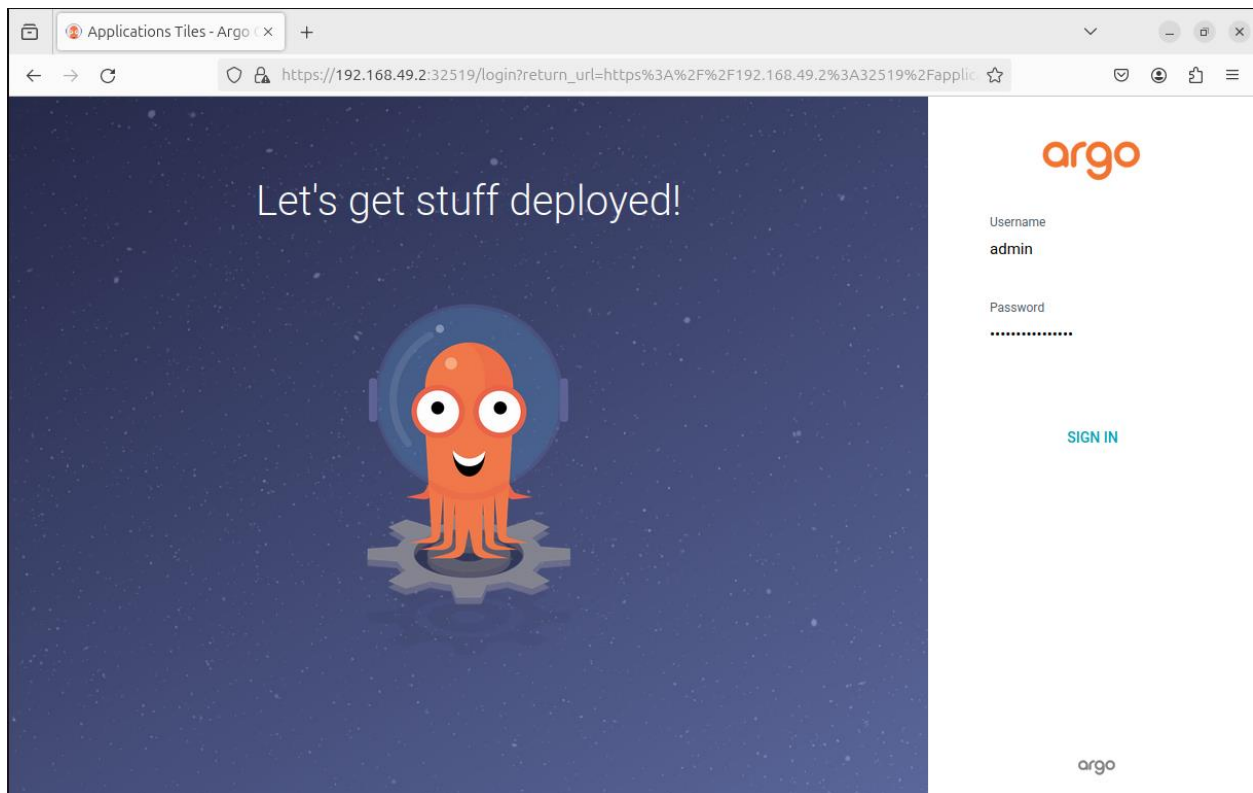
```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'
```

Bước 2: Lấy URL để truy cập giao diện ArgoCD từ trình duyệt

```
viethanh@VA-Computer: ~$ minikube service argocd-server -n argocd
```

NAMESPACE	NAME	TARGET PORT	URL
argocd	argocd-server	http/80	http://192.168.49.2:30099
		https/443	http://192.168.49.2:32519

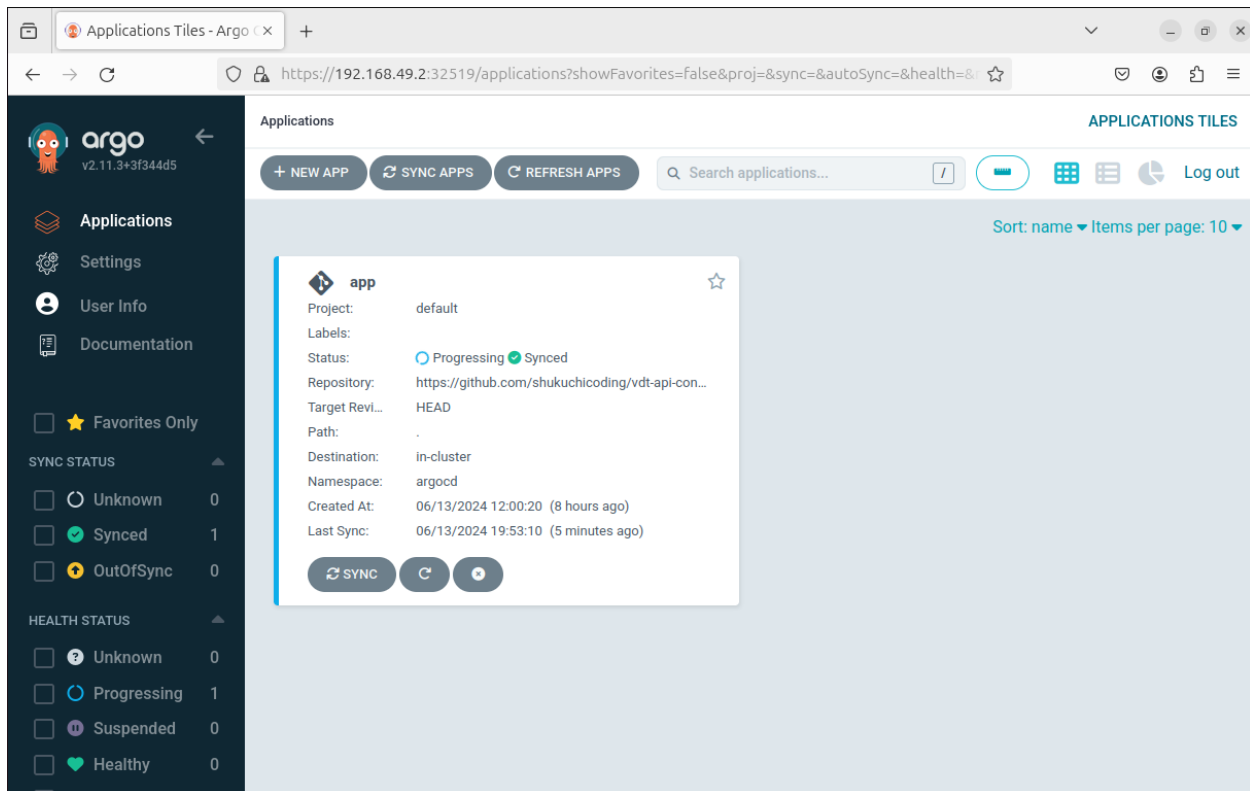
```
[argocd argocd-server http/80  
https/443 http://192.168.49.2:30099  
http://192.168.49.2:32519]  
viethanh@VA-Computer: ~$
```



### Bước 3: Đăng nhập vào ArgoCD

Mặc định, username là admin, để lấy mật khẩu, ta gõ lệnh

```
viethanh@VA-Computer: ~  
viethanh@VA-Computer:~$ kubectl -n argocd get secret argocd-initial-admin-secret  
-o jsonpath='{.data.password}' | base64 -d  
v7yewFa2PxIvuVeIv  
viethanh@VA-Computer:~$
```



Giao diện hệ thống đã xuất hiện

**Yêu cầu 2** Viết 2 helm chart cho web deployment và api deployment, để vào 1 folder riêng trong repo web và repo api  
Tạo 2 repo config cho web và api, trong các repo này chứa các file values.yaml là các cấu hình cần thiết để chạy web và api trên K8S bằng helm chart  
Sử dụng tính năng multisource của ArgoCD để triển khai service web và service api lên K8S Cluster, expose các service dưới dạng NodePort

**Output 2** Các helm chart sử dụng để triển khai web và api lên k8s cluster  
Các file values.yaml trong 2 config repo của web service và api service

manifest của ArgoCD Application  
Ảnh chụp giao diện hệ thống ArgoCD trên trình duyệt  
Ảnh chụp giao diện WEB URL, API URL

Helm chart:

Web: [vdt2024\\_web/web-config at master · shukuchicoding/vdt2024\\_web \(github.com\)](https://github.com/shukuchicoding/vdt2024_web/blob/master/web-config)

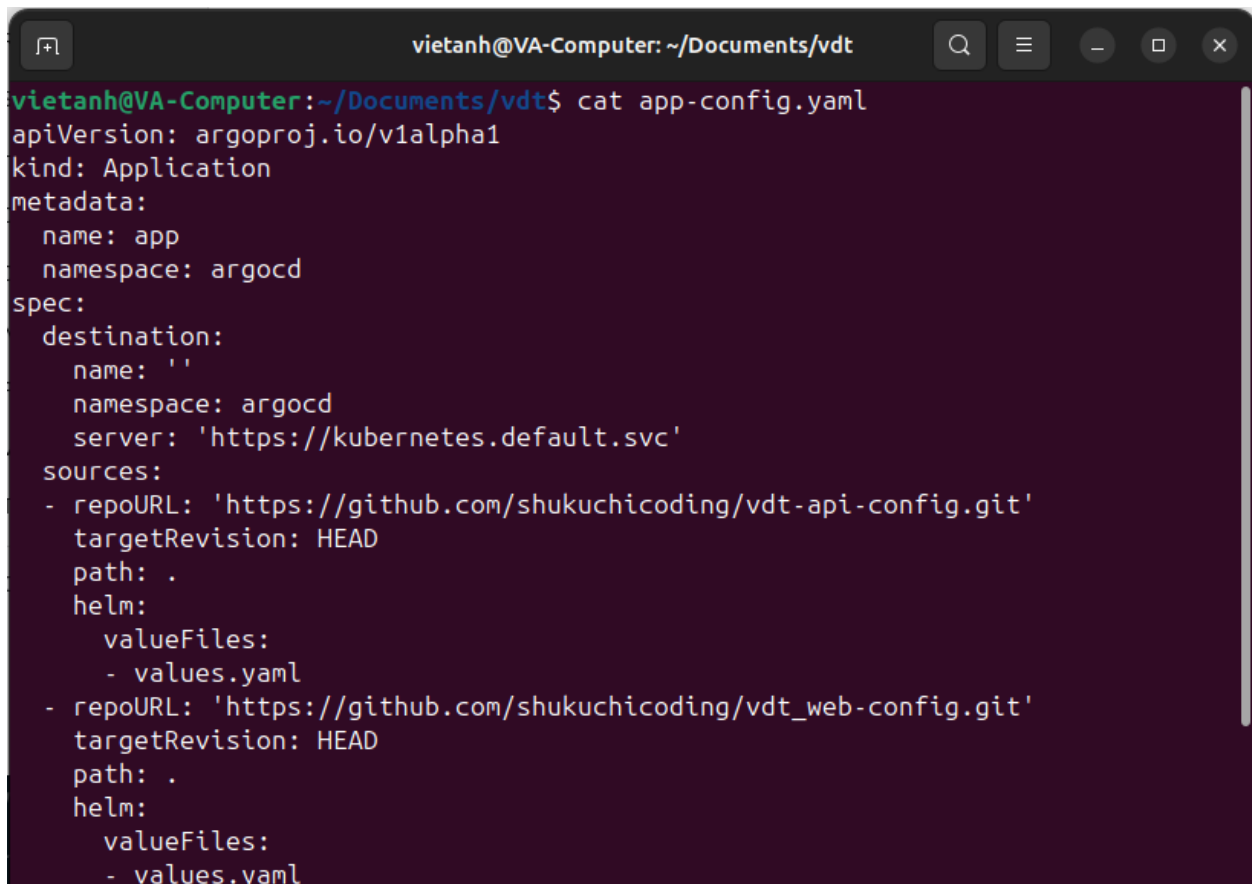
API: [vdt2024\\_api/api-config at master · shukuchicoding/vdt2024\\_api \(github.com\)](https://github.com/shukuchicoding/vdt2024_api/blob/master/api-config)

Config repo:

Web: [shukuchicoding/vdt\\_web-config \(github.com\)](https://github.com/shukuchicoding/vdt_web-config)

API: [shukuchicoding/vdt-api-config \(github.com\)](https://github.com/shukuchicoding/vdt-api-config)

Manifest của Application:

A terminal window titled 'vietanh@VA-Computer: ~/Documents/vdt' displays the command 'cat app-config.yaml' and its output. The output is a YAML configuration for an ArgoCD Application. It specifies the API version as 'argoproj.io/v1alpha1', the kind as 'Application', and the namespace as 'argocd'. The application is named 'app'. It defines two sources: one for 'vdt-api-config' and another for 'vdt\_web-config', both pointing to repositories on GitHub. Each source is configured with its repository URL, target revision (HEAD), path, and a Helm chart with a 'values.yaml' file.

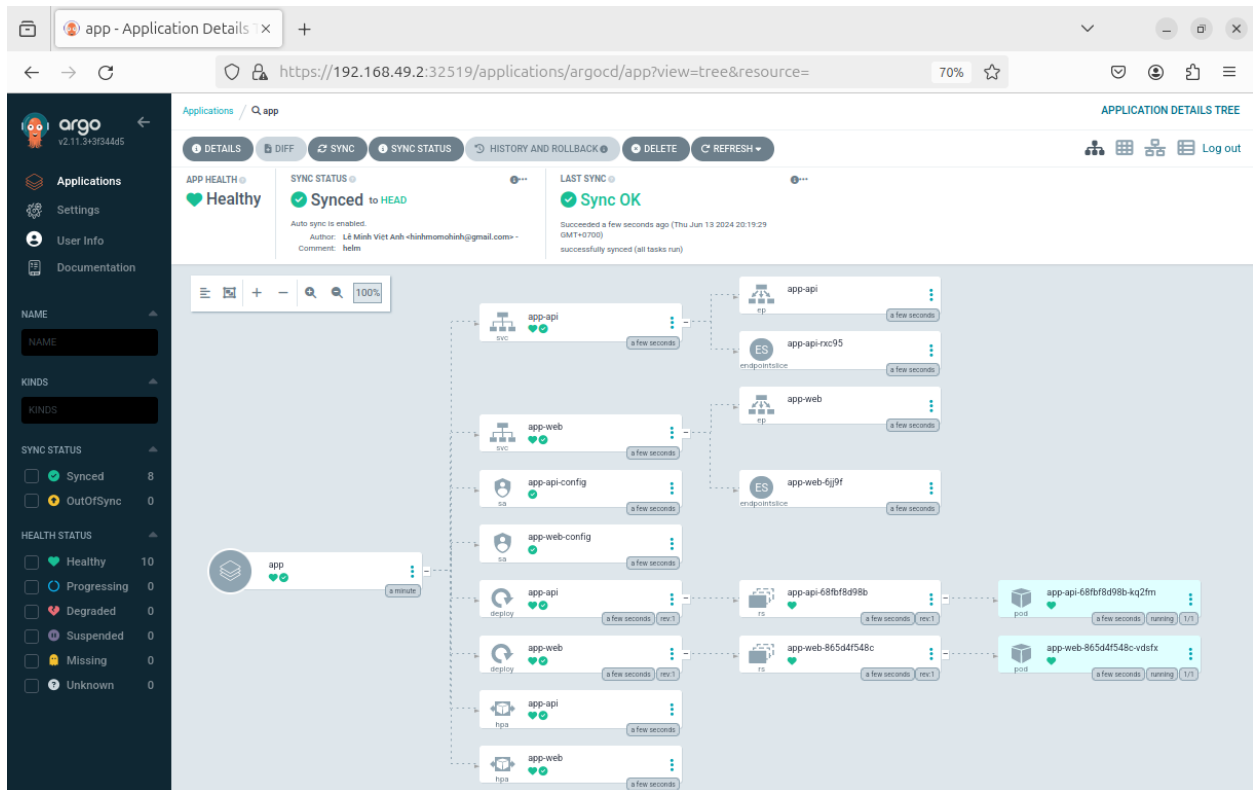
```
vietanh@VA-Computer:~/Documents/vdt$ cat app-config.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: argocd
    server: 'https://kubernetes.default.svc'
  sources:
  - repoURL: 'https://github.com/shukuchicoding/vdt-api-config.git'
    targetRevision: HEAD
    path: .
    helm:
      valueFiles:
      - values.yaml
  - repoURL: 'https://github.com/shukuchicoding/vdt_web-config.git'
    targetRevision: HEAD
    path: .
    helm:
      valueFiles:
      - values.yaml
```

```
project: default
syncPolicy:
  automated:
    prune: true
    selfHeal: true
vietanh@VA-Computer:~/Documents/vdt$
```

Chạy lệnh sau

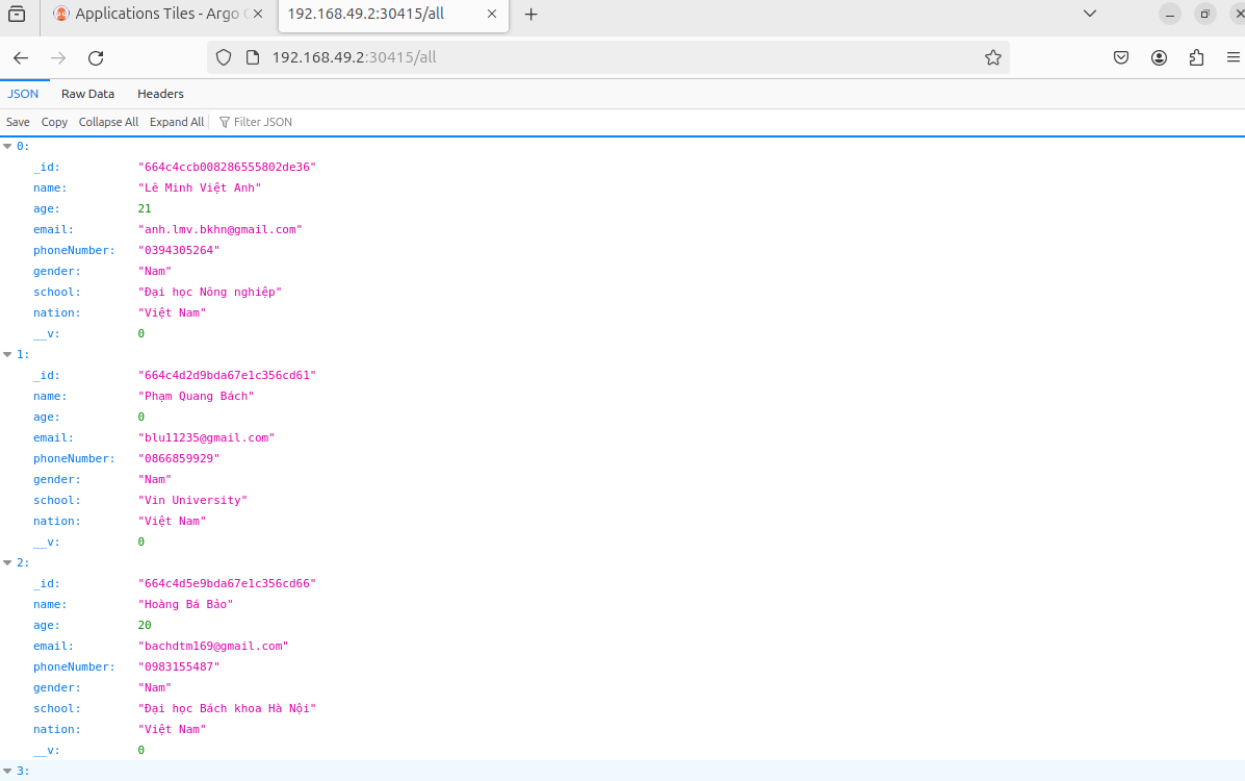
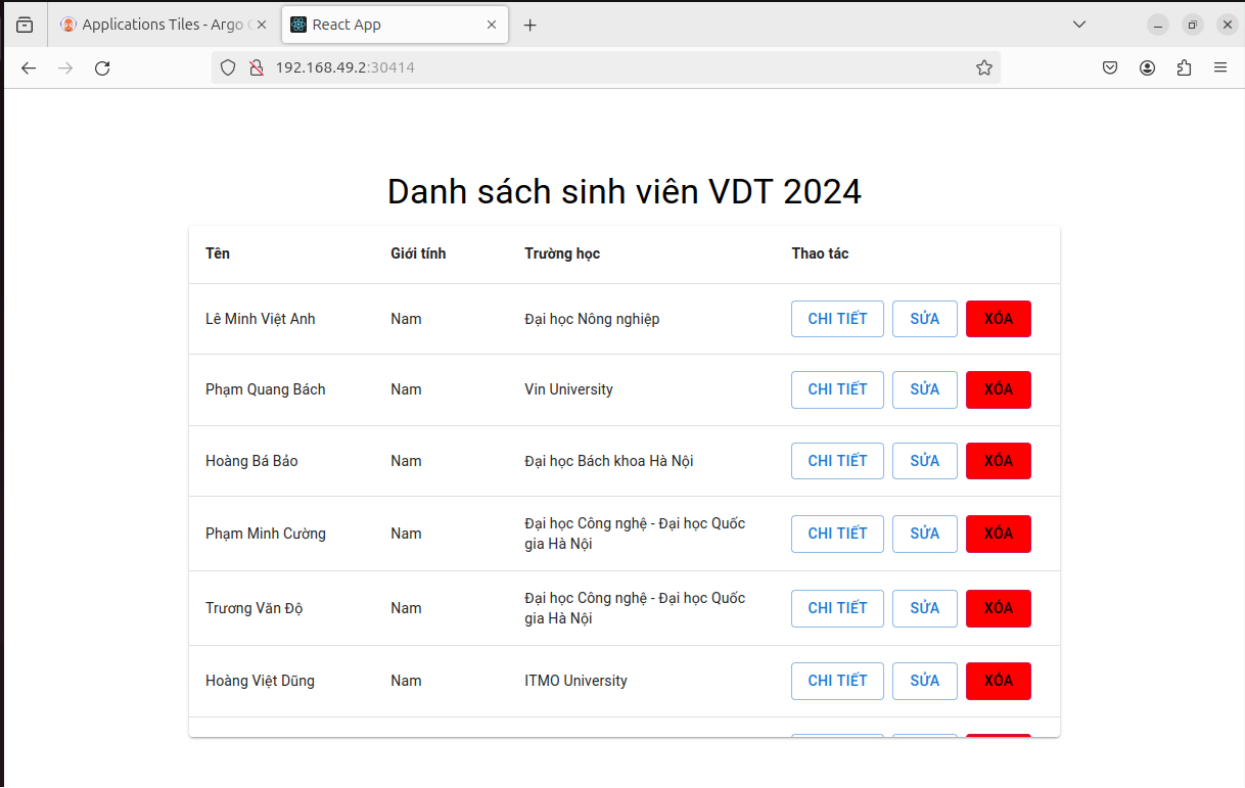
```
kubectl apply -f app-config.yaml -n argocd
```

sẽ hiển thị trên màn hình như sau



Truy cập WEB URL và API URL có kết quả như sau:





## 3 Continuous Delivery

File setup công cụ CI/CD

Api: [vdt2024\\_api/.github/workflows/node.js.yml](https://github.com/shukuchicoding/vdt2024_api/.github/workflows/node.js.yml) at master · shukuchicoding/vdt2024\_api

Web: [vdt2024\\_web/.github/workflows/cicd.yml](https://github.com/shukuchicoding/vdt2024_web/.github/workflows/cicd.yml) at master · shukuchicoding/vdt2024\_web

## 4 Monitoring

## 5 Logging

## 6 Security

### 6.1 Dựng HAProxy Loadbalancer

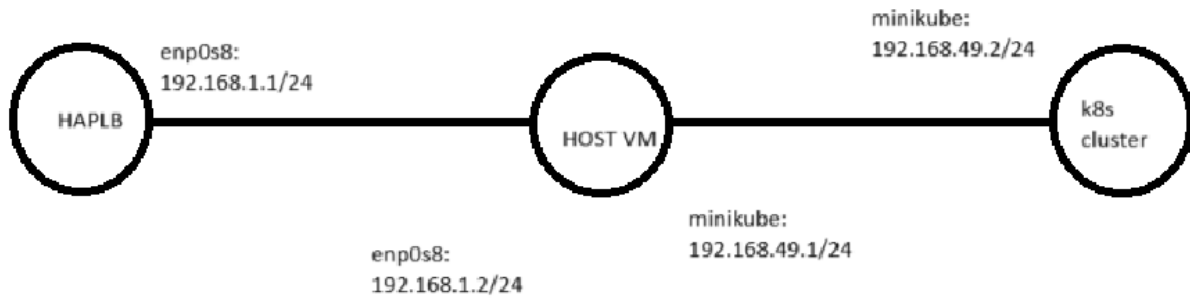
**Yêu cầu 1** Dựng HAProxy Loadbalancer trên 1 VM riêng với mode TCP, mở 2 port web\_port và api\_port trên LB trỏ đến 2 NodePort của Web Deployment và API Deployment của K8S cluster

Sử dụng 1 trong 2 giải pháp Ingress, hoặc HAProxy sidecar container cho các deployment, đảm bảo truy cập đến các port đều sử dụng https

**Output 1** File cấu hình HAProxy Loadbalancer cho webport và apiport  
File cấu hình ingress hoặc file cấu hình deployment sau khi thêm HAProxy Sidecar container vào Deployment  
Kết quả truy cập vào web port và api port từ trình duyệt thông qua https hoặc dùng curl

Ở phần này, ta sử dụng một máy ảo Ubuntu server 22.04

Thiết lập sơ đồ mạng như sau:



Thực hiện các bước sau trên HAProxyLoadbalancer:

Bước 1: Cài đặt haproxy

```
sudo apt install haproxy -y
```

Bước 2: Sử dụng nano chỉnh sửa file haproxy.cfg

```
sudo nano /etc/haproxy/haproxy.cfg
```

Bước 3: Chỉnh sửa file cấu hình

```
GNU nano 6.2 /etc/haproxy/haproxy.cfg
global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
    stats timeout 30s
    user haproxy
    group haproxy
    daemon
    maxconn 15
    # Default SSL material locations
    ca-base /etc/ssl/certs
    crt-base /etc/ssl/private

    # See: https://ssl-config.mozilla.org/#server=haproxy&server-version=2.0.3&config=intermediate
    ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-EC-
    ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_P
    ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
    log global
    mode http
    option httplog
    option dontlognull
    timeout connect 10000
    timeout client 50000
    timeout server 50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http

frontend web_frontend
    bind *:80
    mode tcp
    option tcplog
    default_backend web_backend

frontend api_frontend
    bind *:8080
    mode tcp
    option tcplog
    default_backend api_backend

backend web_backend
    balance roundrobin
    mode tcp
    option tcp-check
    server web_server 192.168.1.2:30414 check

backend api_backend
    balance roundrobin
    mode tcp
    option tcp-check
    server api_server 192.168.1.2:30415 check
```

## 6.2 Xác thực và phân quyền

**Yêu cầu 2** Đảm bảo một số URL của api service khi truy cập phải có xác thực thông qua 1 trong số các phương thức cookie, basic auth, token auth, nếu không sẽ trả về HTTP response code 403

Thực hiện phân quyền cho hai loại người dùng trên API:

- Nếu người dùng có role là user thì truy cập vào GET request trả về 200, còn truy cập vào POST/DELETE thì trả về 403
- Nếu người dùng có role là admin thì truy cập vào GET request

trả về 200, còn truy cập vào POST/DELETE thì trả về 2xx

**Output 2** File trình bày giải pháp sử dụng để authen/authorization cho các service

Kết quả HTTP response khi curl hoặc dùng postman gọi vào các URL khi truyền thêm thông tin xác thực và khi không truyền thông tin xác thực

Kết quả HTTP response khi curl hoặc dùng postman vào các URL với các method GET/POST/DELETE khi lần lượt dùng thông tin xác thực của các user có role là admin và user

Giải pháp sử dụng: JWT và JSONWEBTOKEN

Bước 1: Cài đặt express-jwt và jsonwebtoken

```
npm install express-jwt jsonwebtoken
```

Bước 2: Viết thêm các hàm login, generateToken, authenticateToken, authorizeRole phục vụ cho xác thực danh tính và xác thực quyền

```
const generateToken = (user) => {
  return jwt.sign({ id: user._id, role: user.role }, "123456789", { expiresIn: '12h' });
};

const authenticateToken = (req, res, next) => {
  const token = req.cookies.token;
  console.log("Token: ", token);
  if (!token) {
    return res.status(403).json({ message: 'No token provided' });
  }
  jwt.verify(token, "123456789", (err, user) => {
    if (err) {
      return res.status(403).json({ message: 'Failed to authenticate token' });
    }
    req.user = user;
    next();
  });
};

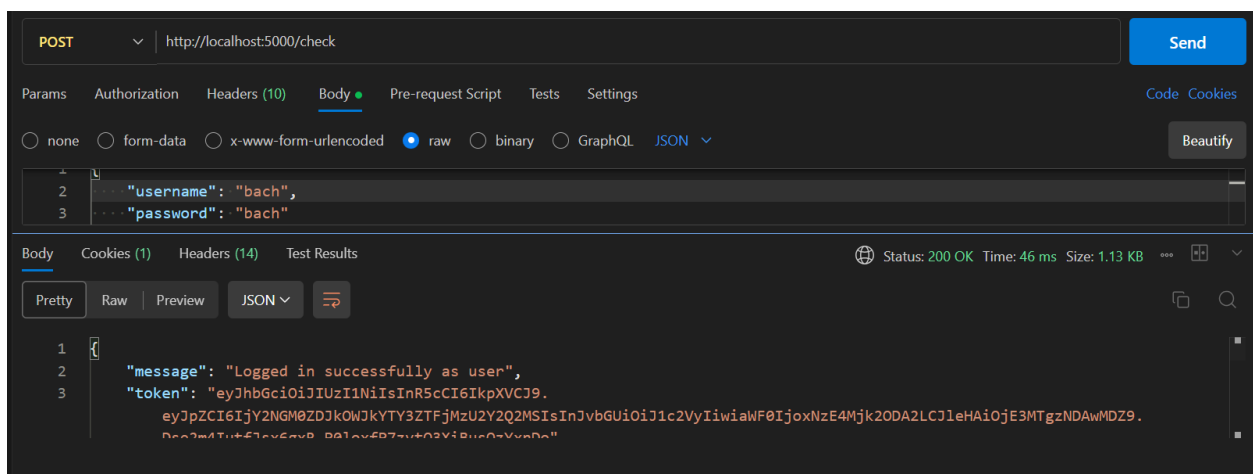
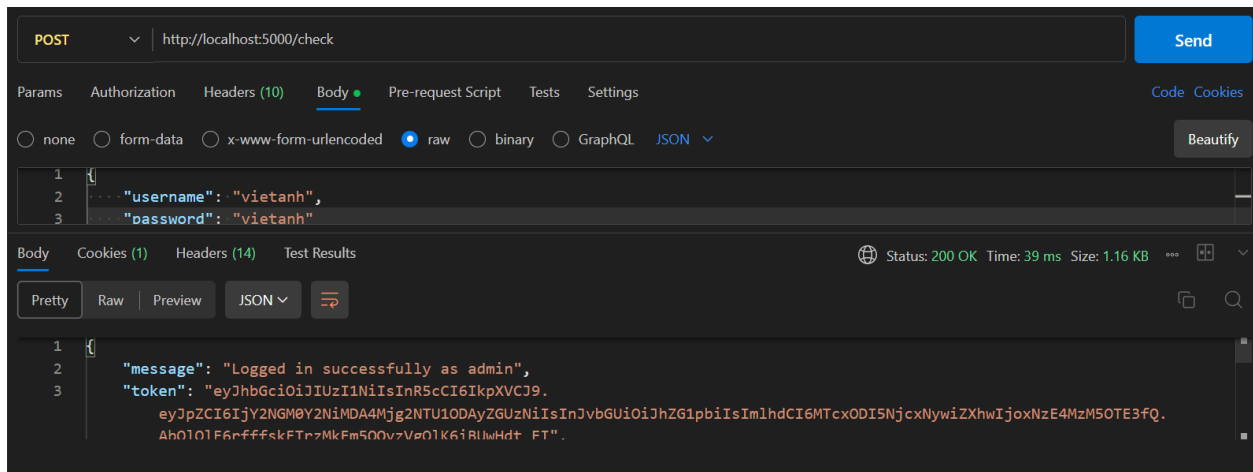
const authorizeRoles = (...roles) => {
  return (req, res, next) => {
    console.log("Roles: ", req.user.role);
    if (!roles.includes(req.user.role)) {
```

```

        return res.status(403).json({ message: 'Access denied', role:
req.user.role, requiredRoles: roles});
    }
    next();
};
};
//dang nhap
app.post("/check", async (req, res) => {
    try {
        const { username, password } = req.body;
        const user = await student.findOne({ username });
        if (!user || user.username === 'none' || user.password === 'none') {
            return res.status(401).json({ message: 'Invalid username' });
        }
        if (!compare(password, user.password)) {
            return res.status(401).json({ message: 'Invalid password' });
        }
        console.log("touch");
        const token = generateToken(user);
        res.cookie('token', token, { httpOnly: true });
        res.status(200).json({ message: `Logged in successfully as ${user.role}`,
token, user });
    } catch (error) {
        logger.error({
            message: error.message,
            path: req.originalUrl,
            method: req.method,
            responseCode: 500
        });
        res.status(500).json({ message: error.message });
    }
});
});

```

Kết quả chạy postman:



## 6.3 Rate limit

**Yêu cầu 3** Viết 2 helm chart cho web deployment và api deployment, để vào 1 folder riêng trong repo web và repo api

Tạo 2 repo config cho web và api, trong các repo này chứa các file values.yaml là các cấu hình cần thiết để chạy web và api trên K8S bằng helm chart

Sử dụng tính năng multisource của ArgoCD để triển khai service web và service api lên K8S Cluster, expose các service dưới dạng NodePort

**Output 3** Các helm chart sử dụng để triển khai web và api lên k8s cluster

Các file values.yaml trong 2 config repo của web service và api service

manifest của ArgoCD Application

Ảnh chụp giao diện hệ thống ArgoCD trên trình duyệt

Ảnh chụp giao diện WEB URL, API URL

Sử dụng express-rate-limit để giới hạn số lần truy cập endpoint của API, giải pháp này đơn giản, dễ sử dụng trong các dự án node js.

### Bước 1: Cài đặt express-rate-limit

```
npm install express-rate-limit
```

Bước 2: Trong index.js của API, thêm các đoạn mã sau:

```
const { rateLimit } = require("express-rate-limit");

const apilimiter = rateLimit({
  windowMs: 60 * 1000, // Thời gian cửa sổ (trong mili giây)
  max: 10, // Số lượng yêu cầu tối đa trong cửa sổ thời gian
  message: "Bạn đã gửi quá nhiều yêu cầu, vui lòng thử lại sau 1 phút",
  statusCode: 409, // Mã HTTP Response trả về khi vượt quá giới hạn
});
app.use(apilimiter);
```

### Bước 3: Kiểm tra kết quả bằng Postman



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE					
				All Logs ▾	Clear
>	GET	http://localhost:5000/all	200	166 ms	
>	GET	http://localhost:5000/all	200	134 ms	
>	GET	http://localhost:5000/all	200	126 ms	
>	GET	http://localhost:5000/all	200	99 ms	
>	GET	http://localhost:5000/all	200	182 ms	
>	GET	http://localhost:5000/all	200	129 ms	
>	GET	http://localhost:5000/all	200	129 ms	
>	GET	http://localhost:5000/all	200	122 ms	
>	GET	http://localhost:5000/all	200	115 ms	
>	GET	http://localhost:5000/all	200	125 ms	
>	GET	http://localhost:5000/all	409	4 ms	
>	GET	http://localhost:5000/all	409	3 ms	
>	GET	http://localhost:5000/all	409	3 ms	
>	GET	http://localhost:5000/all	409	6 ms	