

Tìm kiếm bài viết tại đây ...

[**Trang chủ**](https://techhay.vn/)[**Tác giả**](https://techhay.vn/authors)[**Liên hệ**](https://techhay.vn/contact)



**[Kubernetes Practice] - Triển khai nodejs microservice trên Kubernetes - phần 1**

By [**Huỳnh Minh Quân**](https://techhay.vn/author/huynh-minh-quan/)

Published in [**DevOps**](https://techhay.vn/category/devops/)

<https://techhay.vn/kubernetes-practice-trien-khai-nodejs-microservice-tren-kubernetes-phan-1/?fbclid=IwAR3HggvyNmdzWr-BxKPbq8WYUrQduBm3EvDzSvl4cK-QI6ozzPy2C3rxsI0>

phần 2, phần 3

November 08, 2021

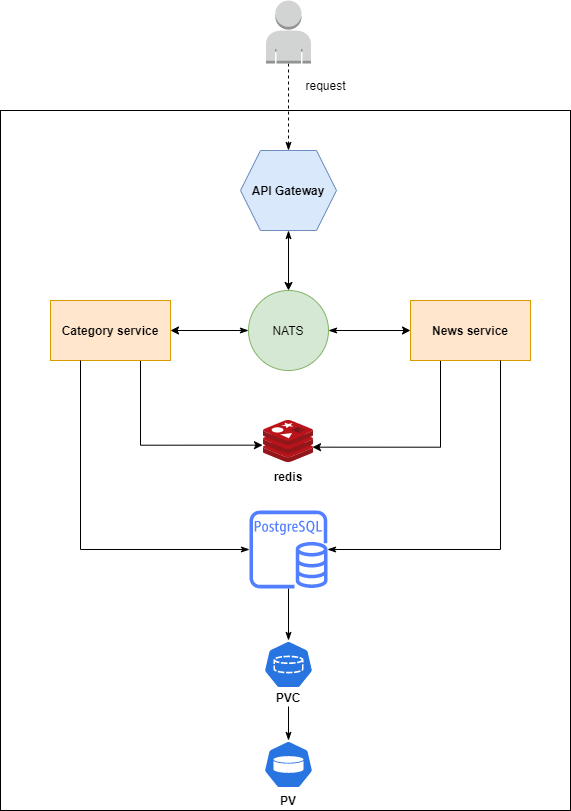
**5 min read**

**Giới thiệu**

Chào mọi người đến với series về kubernetes của mình, trong series này mình sẽ chia sẻ cho các bạn những kinh nghiệm của mình khi triển khai các ứng dụng thực tế trên môi trường kubernetes. Ở phần này mình sẽ nói về cách triển khai nodejs microservice trên môi trường kubernetes, phần một sẽ nói về cách viết file cấu hình từng thành phần cần thiết của microservice, phần hai sẽ nói về cách làm sao để sử dụng gitlab-ci để triển khai ứng dụng một cách tự động.

Về hệ thống, phần backend mình sẽ sử dụng [nodejs molecular](https://moleculer.services/docs/0.14/) framework, đây là một framework dùng để xây dựng một ứng dụng với kiến trúc microservice.

**Kiến trúc của ứng dụng**



API Gateway, sẽ đóng vai trò là ngõ vào cho ứng dụng của ta, nó sẽ tạo một http endpoint và lắng nghe request từ client.

NATS, sẽ đóng vai trò là một transporter, nó sẽ như là một trạm chuyển tiếp để cho các service có thể giao tiếp được với nhau.

Categories service, News service, đây là service thực hiện công việc CRUD resource liên quan.

Redis, dùng để cache kết quả lấy ra từ database và kết quả trả về cho client, giúp giảm số lần thực hiện query vào DB và tăng tốc độ của ứng dụng.

Database thì ta sẽ xài postgres.

Mình đã nói sơ qua về kiến trúc mà ta sẽ triển khai lên trên kubernetes, bây giờ ta sẽ bắt đầu tiến hành viết config cho từng thành phần riêng biệt, bước đầu tiên là ta sẽ build image cho backend của chúng ta.

**Build image**

Các bạn có thể sử dụng image mình đã build sẵn, tên là **080196/microservice**, hoặc nếu các bạn thích tự build image cho chính mình, thì các bạn tải source code từ đây xuống <https://github.com/hoalongnatsu/microservices>.

Sau khi tải được source code, các bạn nhảy vào thư mục microservices/code, và thực hiện build image với tên image sẽ là <docker-hub-username>/microservice, sau đó các bạn thực hiện câu lệnh docker login và push image lên docker hub của mình.

git clone https://github.com/hoalongnatsu/microservices.git && cd microservices/code

docker build . -t 080196/microservice

docker push 080196/microservice

Sau khi build image và push lên docker hub thành công, ta sẽ tiến hành viết file config cho ứng dụng.

**Deploy API Gateway**

Đầu tiên ta sẽ viết file config cho api gateway, để tạo api gateway, ta sẽ dùng Deployment, Deployment là gì các bạn có thể xem [ở đây](https://viblo.asia/p/kubernetes-series-bai-5-deployment-cap-nhat-ung-dung-bang-deployment-RQqKL6q0l7z), tạo file tên là api-gateway-deployment.yaml với config như sau:

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-gateway

labels:

component: api-gateway

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: api-gateway

template:

metadata:

labels:

component: api-gateway

spec:

containers:

- name: api-gateway

image: 080196/microservice

ports:

- name: http

containerPort: 3000

protocol: TCP

livenessProbe:

httpGet:

path: /

port: http

readinessProbe:

httpGet:

path: /

port: http

env:

- name: NODE\_ENV

value: testing

- name: SERVICEDIR

value: dist/services

- name: SERVICES

value: api

- name: PORT

value: "3000"

- name: CACHER

value: redis://redis:6379

Trong image **080196/microservice** chúng ta đã build ở trên, sẽ có 3 service ở trong image đó là api, categories, news. Ta chọn service ta cần chạy bằng cách truyền giá trị của service ta muốn chạy vào biến môi trường có tên là SERVICES, ở file config trên ta chạy api gateway, nên ta truyền vào giá trị là **api**.

Các bạn nhìn vào phần code ở file code/services/api.service.ts, ta sẽ thấy ở chỗ setting cho api gateway ở dòng 15

...

settings: {

port: process.env.PORT || 3001,

...

Với biến môi trường PORT, ta sẽ chọn port mà api gateway lắng nghe, ở trên ta truyền vào giá trị là 3000. Vậy api gateway của ta sẽ chạy và lắng nghe ở port 3000. Biến môi trường CACHER là ta khai báo redis host mà các service của chúng ta sẽ sử dụng. Ta tạo deployment.

$ kubectl apply -f api-gateway-deployment.yaml

deployment.apps/api-gateway created

$ kubectl get deploy

NAME READY UP-TO-DATE AVAILABLE AGE

api-gateway 0/1 1 0 100s

Ta đã tạo được api gateway, nhưng khi bạn get pod, bạn sẽ thấy nó không chạy thành công, mà sẽ bị restart đi restart lại.

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-79688cf6f5-g88f2 0/1 Running 2 93s

Ta logs pod để xem lý do tại sao nó không chạy được mà cứ bị restart.

$ kubectl logs api-gateway-79688cf6f5-g88f2

...

[2021-11-07T14:53:37.449Z] ERROR api-gateway-79688cf6f5-g88f2-28/CACHER: Error: getaddrinfo EAI\_AGAIN redis

at GetAddrInfoReqWrap.onlookup [as oncomplete] (dns.js:60:26) {

errno: 'EAI\_AGAIN',

code: 'EAI\_AGAIN',

syscall: 'getaddrinfo',

hostname: 'redis'

}

Lỗi được hiển thị ở đây là do ta không thể kết nối được tới redis, vì ta chưa tạo redis nào cả, tiếp theo ta sẽ tạo redis.

**Deploy Redis**

Ta tạo một file redis-deployment.yaml với config như sau:

apiVersion: apps/v1

kind: Deployment

metadata:

name: redis

labels:

component: redis

spec:

strategy:

type: Recreate

selector:

matchLabels:

component: redis

template:

metadata:

labels:

component: redis

spec:

containers:

- name: redis

image: redis

ports:

- containerPort: 6379

$ kubectl apply -f redis-deployment.yaml

deployment.apps/redis created

$ kubectl get deploy

NAME READY UP-TO-DATE AVAILABLE AGE

api-gateway 0/1 1 0 16m

redis 1/1 1 1 14s

Vậy là ta đã tạo được redis pod, tiếp theo, nếu muốn connect được tới redis pod này, ta cần phải tạo Service resource cho nó. Tạo file tên là redis-service.yaml với config như sau:

apiVersion: v1

kind: Service

metadata:

name: redis

labels:

component: redis

spec:

selector:

component: redis

ports:

- port: 6379

Ta tạo service:

$ kubectl apply -f redis-service.yaml

service/redis created

Ta Restart api gateway deployment lại để nó có thể kết nối được tới redis.

$ kubectl rollout restart deploy api-gateway

deployment.apps/api-gateway restarted

Bây giờ khi ta get pod ra, ta vẫn thấy nó vẫn chưa chạy được thành công, ta logs nó tiếp để xem tại sao.

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-79688cf6f5-g88f2 0/1 CrashLoopBackOff 8 13m

api-gateway-7f4d5f54f-lzgkd 0/1 Running 2 82s

redis-58c4799ccc-qhv2z 1/1 Running 0 5m41s

$ kubectl logs api-gateway-7f4d5f54f-lzgkd

...

[2021-11-07T15:05:10.388Z] INFO api-gateway-7f4d5f54f-lzgkd-28/CACHER: Redis cacher connected.

Sequelize CLI [Node: 12.13.0, CLI: 6.2.0, ORM: 6.6.5]

Loaded configuration file "migrate/config.js".

Using environment "testing".

ERROR: connect ECONNREFUSED 127.0.0.1:5432

Error: Command failed: sequelize-cli db:migrate

ERROR: connect ECONNREFUSED 127.0.0.1:5432

at ChildProcess.exithandler (child\_process.js:295:12)

at ChildProcess.emit (events.js:210:5)

at maybeClose (internal/child\_process.js:1021:16)

at Process.ChildProcess.\_handle.onexit (internal/child\_process.js:283:5) {

killed: false,

code: 1,

signal: null,

cmd: 'sequelize-cli db:migrate'

}

Sequelize CLI [Node: 12.13.0, CLI: 6.2.0, ORM: 6.6.5]

Loaded configuration file "migrate/config.js".

Using environment "testing".

ERROR: connect ECONNREFUSED 127.0.0.1:5432

...

Kết quả in ra ta thấy được là ta đã kết nối redis thành công, lỗi tiếp theo mà pod api gateway hiển thị là lỗi khi nó migrate database, nó không thể kết nối database được, vì ta chưa tạo database nào cả, bước tiếp theo là ta sẽ tạo database.

**Deploy database**

Để deploy database, ta sẽ không dùng Deployment mà sẽ dùng StatefulSet, về lý do thì các bạn có thể xem [ở đây](https://viblo.asia/p/kubernetes-series-bai-9-statefulsets-deploying-replicated-stateful-applications-07LKXkXp5V4). Ta tạo một file tên là postgres-statefulset.yaml với config như sau:

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: postgres

labels:

component: postgres

spec:

selector:

matchLabels:

component: postgres

serviceName: postgres

template:

metadata:

labels:

component: postgres

spec:

containers:

- name: postgres

image: postgres:11

ports:

- containerPort: 5432

volumeMounts:

- mountPath: /var/lib/postgresql/data

name: postgres-data

env:

- name: POSTGRES\_DB

value: postgres

- name: POSTGRES\_USER

value: postgres

- name: POSTGRES\_PASSWORD

value: postgres

volumeClaimTemplates:

- metadata:

name: postgres-data

spec:

accessModes:

- ReadWriteOnce

storageClassName: hostpath

resources:

requests:

storage: 5Gi

Lưu ý phần storageClassName, này tùy thuộc vào kubernetes cluster của bạn thì bạn sẽ chỉ định trường storageClassName tương ứng nhé, để xem StorageClass thì bạn gõ câu lệnh sau kubectl get sc. Ta tạo statefulset:

$ kubectl apply -f postgres-statefulset.yaml

statefulset.apps/postgres created

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-79688cf6f5-g88f2 0/1 Running 16 32m

api-gateway-7f4d5f54f-lzgkd 0/1 CrashLoopBackOff 11 20m

postgres-0 1/1 Running 0 55s

redis-58c4799ccc-qhv2z 1/1 Running 0 25m

Ta get pod thì ta sẽ thấy postgres pod đã được tạo thành công, tiếp theo muốn kết nối được tới DB, ta cần tạo Service cho nó, tạo file tên là postgres-service.yaml với config như sau:

apiVersion: v1

kind: Service

metadata:

name: postgres

labels:

component: postgres

spec:

selector:

component: postgres

ports:

- port: 5432

Ta tạo service:

$ kubectl apply -f postgres-service.yaml

service/postgres created

Giờ ta sẽ update lại api gateway Deployment để kết nối với DB ta mới tạo, các bạn xem ở trong file code/src/db/connect.ts thì sẽ thấy các biến môi trường mà api gateway dùng để kết nối tới DB.

*import* { Op, Sequelize } *from* "sequelize";

*export* *const* connect = () => *new* Sequelize(

process.env.DB\_NAME,

process.env.DB\_USER,

process.env.DB\_PASSWORD,

{

host: process.env.DB\_HOST,

port: +process.env.DB\_PORT,

operatorsAliases: {

$like: Op.like,

$nlike: Op.notLike,

$eq: Op.eq,

$ne: Op.ne,

$*in*: Op.in,

$nin: Op.notIn,

$gt: Op.gt,

$gte: Op.gte,

$lt: Op.lt,

$lte: Op.lte,

$bet: Op.between,

$contains: Op.contains,

},

dialect: "postgres",

logging: (process.env.ENABLE\_LOG\_QUERY === "true"),

}

);

Ta update lại các biến env của file api-gateway-deployment.yaml, và tạo lại deployment.

...

env:

- name: NODE\_ENV

value: testing

- name: SERVICEDIR

value: dist/services

- name: SERVICES

value: api

- name: PORT

value: "3000"

- name: CACHER

value: redis://redis:6379

- name: DB\_HOST

value: postgres

- name: DB\_PORT

value: "5432"

- name: DB\_NAME

value: postgres

- name: DB\_USER

value: postgres

- name: DB\_PASSWORD

value: postgres

$ kubectl apply -f api-gateway-deployment.yaml

deployment.apps/api-gateway configured

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-544c7f84-6hv7z 1/1 Running 0 2m4s

nats-65687968fc-2drwp 1/1 Running 0 4m23s

postgres-0 1/1 Running 0 31m

redis-58c4799ccc-qhv2z 1/1 Running 0 56m

Bây giờ khi ta get pod, thì ta thấy pod api gateway của ta đã chạy thành công.

**Deploy categories news service**

Tiếp theo ta sẽ deploy 2 service còn lại của micoservice, tạo file tên là categories-news-deployment.yaml với config như sau:

apiVersion: apps/v1

kind: Deployment

metadata:

name: categories-service

labels:

component: categories-service

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: categories-service

template:

metadata:

labels:

component: categories-service

spec:

containers:

- name: categories-service

image: 080196/microservice

env:

- name: NODE\_ENV

value: testing

- name: SERVICEDIR

value: dist/services

- name: SERVICES

value: categories

- name: CACHER

value: redis://redis:6379

- name: DB\_HOST

value: postgres

- name: DB\_PORT

value: "5432"

- name: DB\_NAME

value: postgres

- name: DB\_USER

value: postgres

- name: DB\_PASSWORD

value: postgres

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: news-service

labels:

component: news-service

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: news-service

template:

metadata:

labels:

component: news-service

spec:

containers:

- name: news-service

image: 080196/microservice

env:

- name: NODE\_ENV

value: testing

- name: SERVICEDIR

value: dist/services

- name: SERVICES

value: news

- name: CACHER

value: redis://redis:6379

- name: DB\_HOST

value: postgres

- name: DB\_PORT

value: "5432"

- name: DB\_NAME

value: postgres

- name: DB\_USER

value: postgres

- name: DB\_PASSWORD

value: postgres

$ kubectl apply -f categories-news-deployment.yaml

deployment.apps/categories-service created

deployment.apps/news-service created

Sau khi tạo 2 service này xong, để chúng và api gateway có thể giao tiếp với nhau, ta cần tạo NATS.

**Deploy NATS**

Tạo một file tên là nats-deployment.yaml với config như sau:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nats

labels:

component: nats

spec:

strategy:

type: Recreate

selector:

matchLabels:

component: nats

template:

metadata:

labels:

component: nats

spec:

containers:

- name: nats

image: nats

ports:

- containerPort: 4222

$ kubectl apply -f nats-deployment.yaml

deployment/nats created

Tiếp theo ta tạo service cho NATS, tạo file nats-service.yaml:

apiVersion: v1

kind: Service

metadata:

name: nats

labels:

component: nats

spec:

selector:

component: nats

ports:

- port: 4222

$ kubectl apply -f nats-service.yaml

service/nats created

Bước cuối cùng, ta update lại env của api gateway và categories với news service, và tạo lại chúng. Thêm giá trị này vào env để cập nhật TRANSPORTER cho các service:

...

env:

...

- name: TRANSPORTER

value: nats://nats:4222

Cập nhật lại các deployment.

$ kubectl apply -f kubectl apply -f api-gateway-deployment.yaml

deployment.apps/api-gateway configured

$ kubectl apply -f categories-news-deployment.yaml

deployment.apps/categories-service configured

deployment.apps/news-service configured

Ta get pod và xem logs của api gateway, xem các service đã có thể giao tiếp với nhau được chưa:

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-6cb4c6b657-tlkzq 1/1 Running 0 36s

categories-service-689cdb6c6d-gqtlb 1/1 Running 0 20s

nats-65687968fc-2drwp 1/1 Running 0 20m

news-service-6b85f99987-dcplv 1/1 Running 0 20s

postgres-0 1/1 Running 0 48m

redis-58c4799ccc-qhv2z 1/1 Running 0 72m

$ kubectl logs api-gateway-6cb4c6b657-tlkzq

...

[2021-11-07T16:14:54.181Z] INFO api-gateway-6cb4c6b657-tlkzq-28/REGISTRY: Node 'news-service-6b85f99987-vcjzn-28' connected.

...

[2021-11-07T16:14:57.357Z] INFO api-gateway-6cb4c6b657-tlkzq-28/REGISTRY: Node 'categories-service-689cdb6c6d-gjjjr-29' connected.

...

Bạn sẽ thấy logs là news-service và categories-service đã được kết nối với api-gateway. Vậy là ứng dụng của ta đã chạy thành công, nhưng bạn có để ý thấy những biến env mà ta khai báo thì hơi dài và lập lại ở các file deployment không? Ta có thể dùng ConfigMap để khai báo cấu hình ở chỗ và sử dụng lại cho nhiều nơi, giúp file config của ta gọn hơn.

**Khai báo config chung**

Tạo một file tên là microservice-cm.yaml với config như sau:

apiVersion: v1

kind: ConfigMap

metadata:

name: microservice-cm

labels:

component: microservice-cm

data:

NODE\_ENV: testing

SERVICEDIR: dist/services

TRANSPORTER: nats://nats:4222

CACHER: redis://redis:6379

DB\_NAME: postgres

DB\_HOST: postgres

DB\_USER: postgres

DB\_PASSWORD: postgres

DB\_PORT: "5432"

$ kubectl apply -f microservice-cm.yaml

configmap/microservice-cm created

Ta update lại config của các file deployment như sau, file api-gateway-deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-gateway

labels:

component: api-gateway

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: api-gateway

template:

metadata:

labels:

component: api-gateway

spec:

containers:

- name: api-gateway

image: 080196/microservice

ports:

- name: http

containerPort: 3000

protocol: TCP

livenessProbe:

httpGet:

path: /

port: http

readinessProbe:

httpGet:

path: /

port: http

env:

- name: SERVICES

value: api

- name: PORT

value: "3000"

envFrom:

- configMapRef:

name: microservice-cm

$ kubectl apply -f api-gateway-deployment.yaml

deployment.apps/api-gateway configured

File categories-news-deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: categories-service

labels:

component: categories-service

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: categories-service

template:

metadata:

labels:

component: categories-service

spec:

containers:

- name: categories-service

image: 080196/microservice

env:

- name: SERVICES

value: categories

envFrom:

- configMapRef:

name: microservice-cm

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: news-service

labels:

component: news-service

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: news-service

template:

metadata:

labels:

component: news-service

spec:

containers:

- name: news-service

image: 080196/microservice

env:

- name: SERVICES

value: news

envFrom:

- configMapRef:

name: microservice-cm

$ kubectl apply -f categories-news-deployment.yaml

deployment.apps/categories-service configured

deployment.apps/news-service configured

Ta get pod lại xem mọi thứ vẫn ok không:

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-86b67895fd-cphmv 1/1 Running 0 79s

categories-service-84c74cd87c-zjtd2 1/1 Running 0 53s

nats-65687968fc-2drwp 1/1 Running 0 41m

news-service-69f45b8668-kv9dm 1/1 Running 0 52s

postgres-0 1/1 Running 0 69m

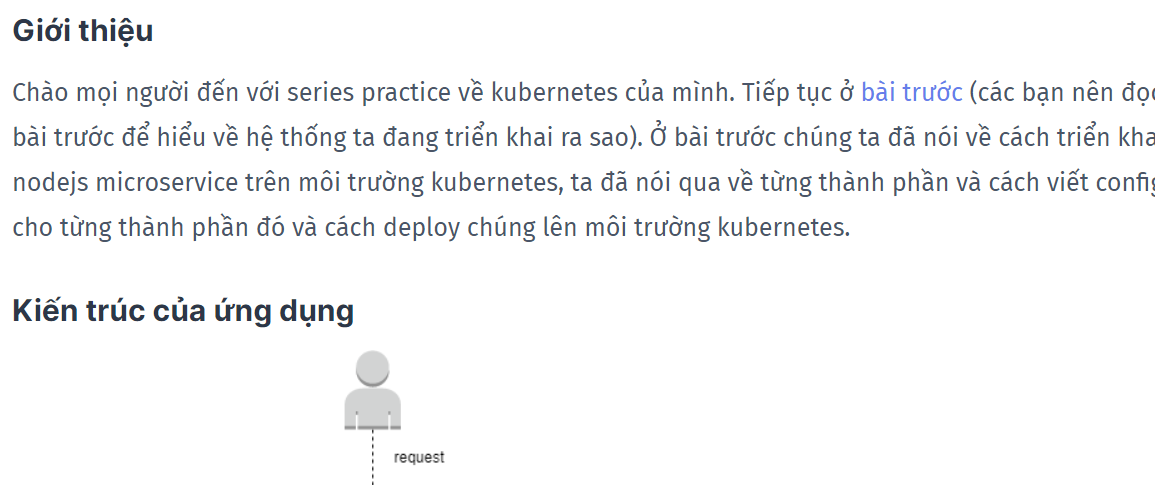
redis-58c4799ccc-qhv2z 1/1 Running 0 93m

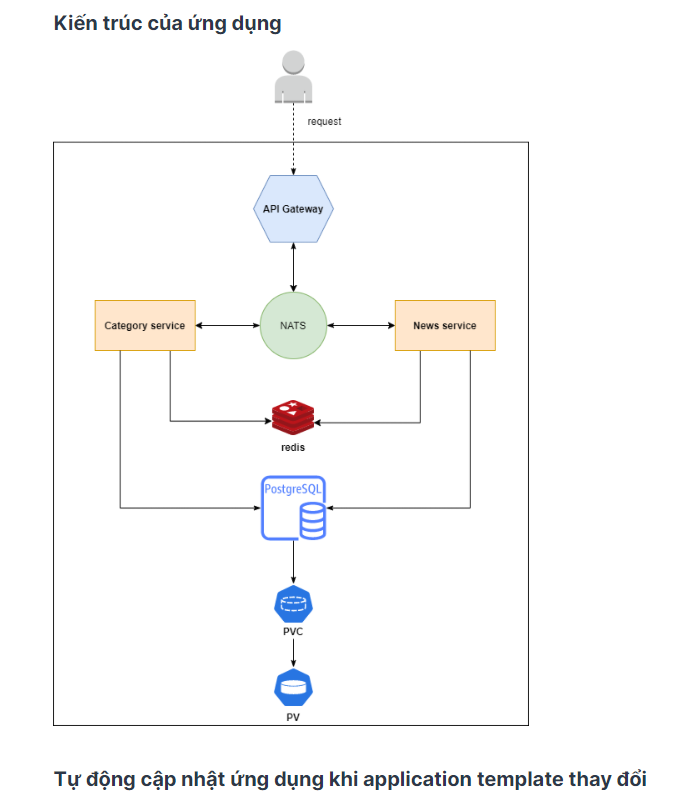
OK, tất cả các thành phần của ứng dụng ta vẫn chạy bình thường. Ta sử dụng ConfigMap để giúp file config của ta nhìn gọn hơn, và ta quản lý tất cả env ở một chỗ, dễ dàng thay đổi và cập nhật lại.

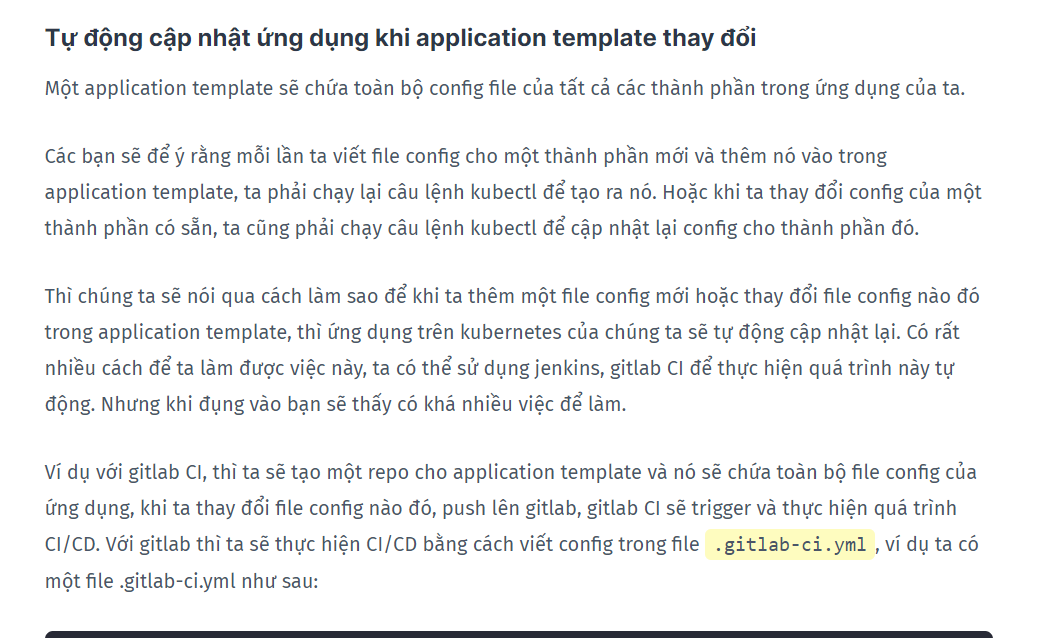
**Kết luận**

Vậy là ta đã triển khai được mô hình microservice lên môi trường kubernetes, như các bạn thấy thì không khó lắm, chỉ cần ta triển khai những thành phần trong đó từ từ, là ta sẽ triển khai được. Phần tiếp theo mình sẽ nói về cách làm sao để tự động được quá trình khi ta cập nhật lại file config nó sẽ tự update lên trên cluster cho chúng ta, không cần ta phải chạy lại câu lệnh kubectl. Và cách dùng gitlab ci để cập nhật lại ứng dụng khi ta sửa và update code trong ứng dụng của ta.

# Phần 2







deploy template:

stage: deploy

tags:

- kala

script:

- rsync -avz -e 'ssh -i ~/.ssh/kala.pem' . ec2-user@$SERVER:~/kubernetes --delete --force

- |

sudo ssh -i ~/.ssh/kala.pem ec2-user@$SERVER << EOF

cd ~/kubernetes

kubectl apply -f . --recursive

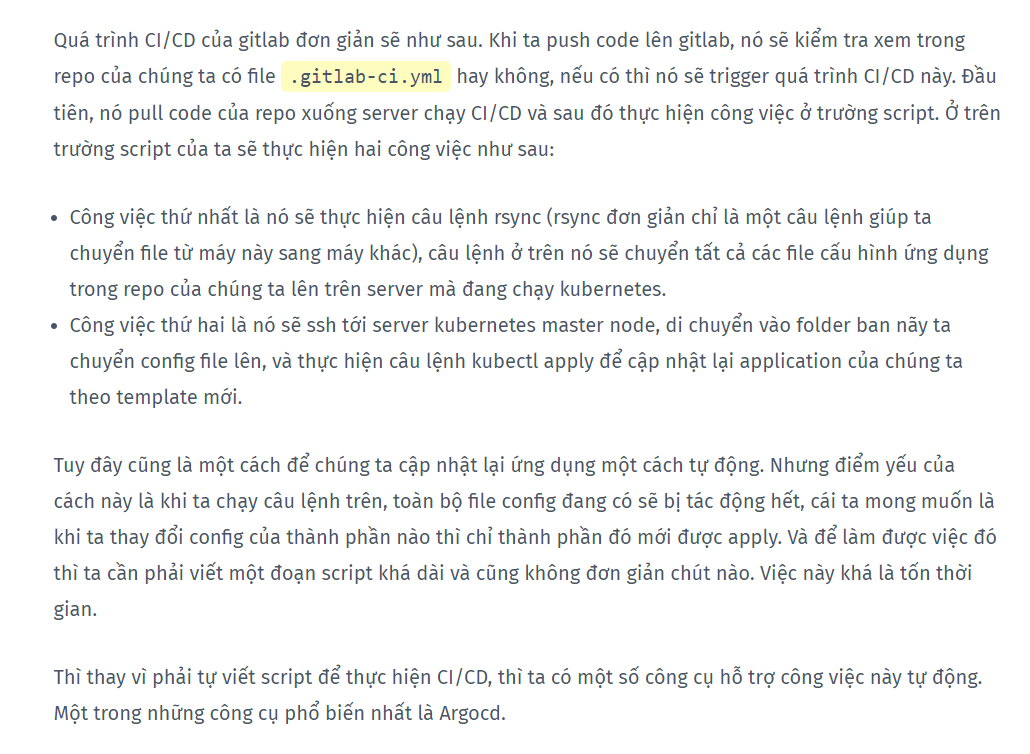
exit

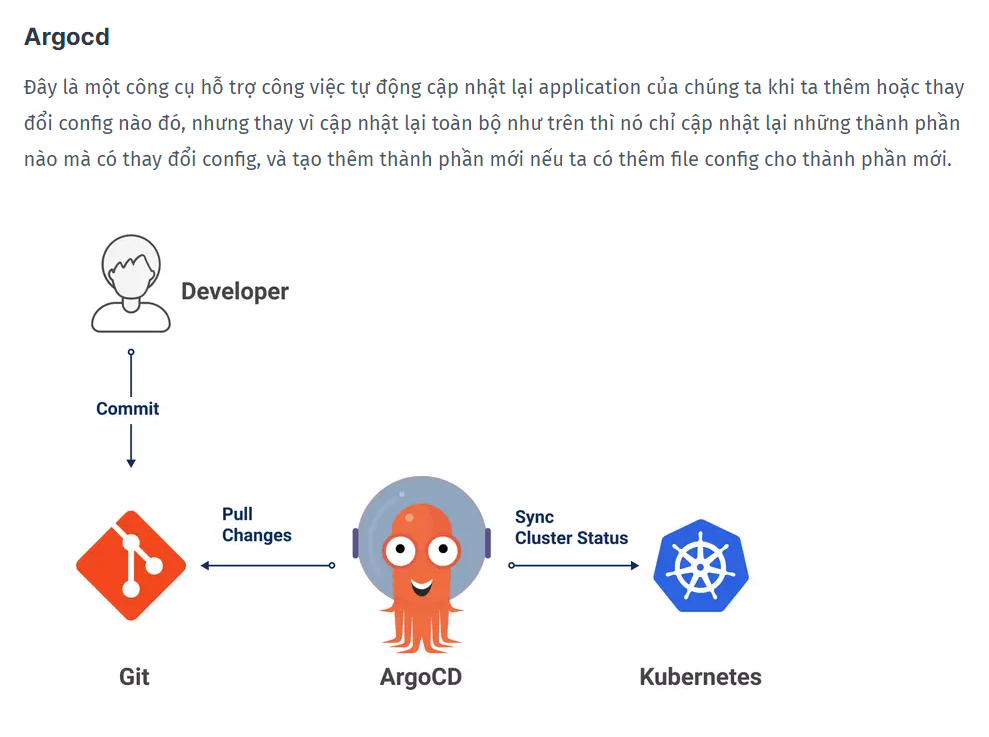
EOF

when: manual

only:

- production





### Cài đặt Argocd

Để cài Argocd, các bạn làm theo các bước sau đây

**1. Install Argo CD**

kubectl create namespace argocd

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

Sau khi chạy câu lệnh xong thì ta chờ một chút để Argo CD install tất cả các Deployment và Service resource của nó.

**2. Install Argo CD CLI**

Đây là cách cho linux:

curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64

chmod +x /usr/local/bin/argocd

Các môi trường khác các bạn có thể xem ở đây <https://argo-cd.readthedocs.io/en/stable/cli_installation/>.

**3. Access The Argo CD API Server**

Ta chạy câu lệnh get pod để kiểm tra toàn bộ Pod của chúng ta đã chạy thành công hay chưa.

$ kubectl get pod -n argocd

argocd-application-controller-0 1/1 Running 0 2m23s

argocd-dex-server-5fc596bcdd-7nvlp 1/1 Running 0 2m23s

argocd-redis-5b6967fdfc-xtns6 1/1 Running 0 2m23s

argocd-repo-server-98598b6c7-chr9k 1/1 Running 0 2m23s

argocd-server-5b4b7b868b-ffs9w 1/1 Running 0 2m23s

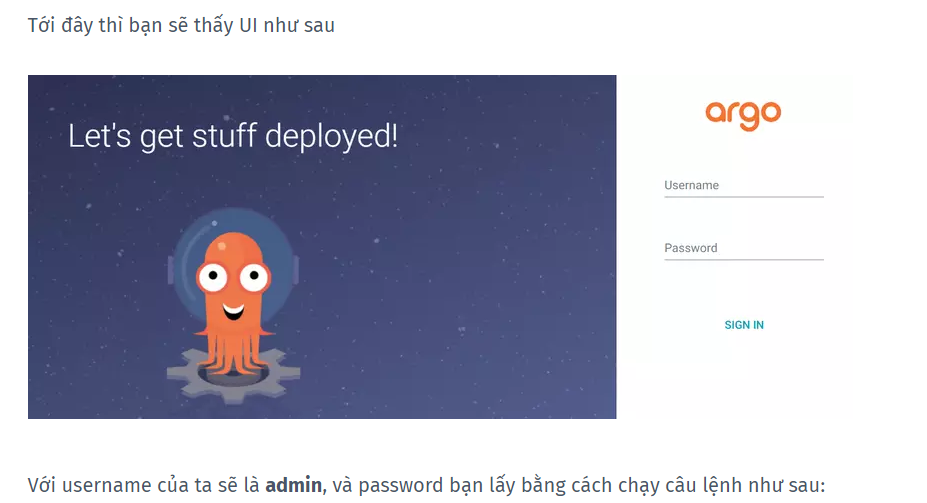
Nếu tất cả đều chạy thành công, để truy cập Argo CD thì ta có thể sử dụng NodePort Service, Ingress. Ở đây để nhanh thì mình dùng Port Forwarding.

$ kubectl port-forward svc/argocd-server -n argocd 8080:443

Forwarding from 127.0.0.1:8080 -> 8080

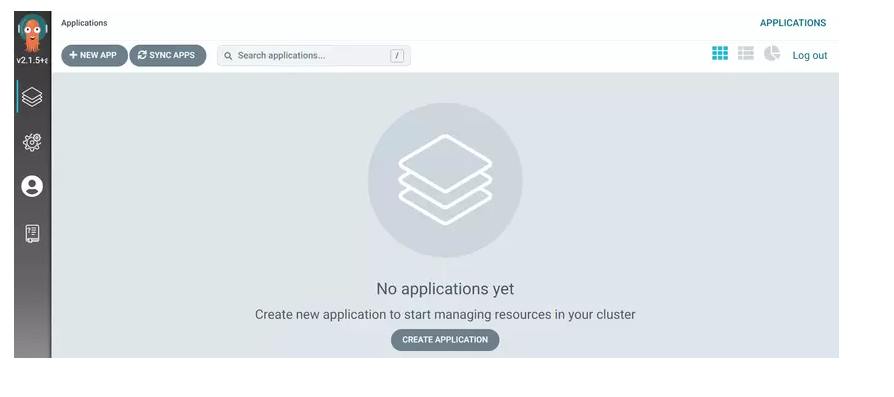
Forwarding from [::1]:8080 -> 8080

Bây giờ thì ta mở web browser và truy cập vào địa chỉ http://localhost:8080. Vì ta chạy localhost nên khi nó báo unsafe, thì ta chỉ việc bấm proceed to localhost (unsafe) là được.



$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d

Sau đó ta lấy password được in ra terminal để đăng nhập. Sau khi login sau bạn sẽ thấy được giao diện.



Nếu bạn muốn update lại password thì ta làm như sau, trước tiên chạy câu lệnh:

$ argocd login <ARGOCD\_SERVER>

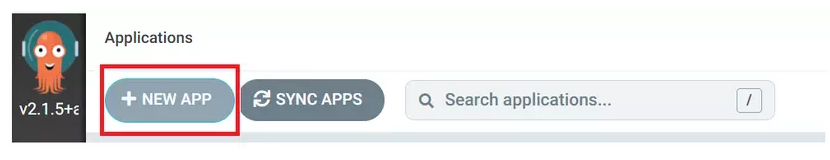
Với ARGOCD\_SERVER là IP của máy ta + port 8080 ở trên. Sau khi login sau thì ta chạy câu lệnh:

$ argocd account update-password

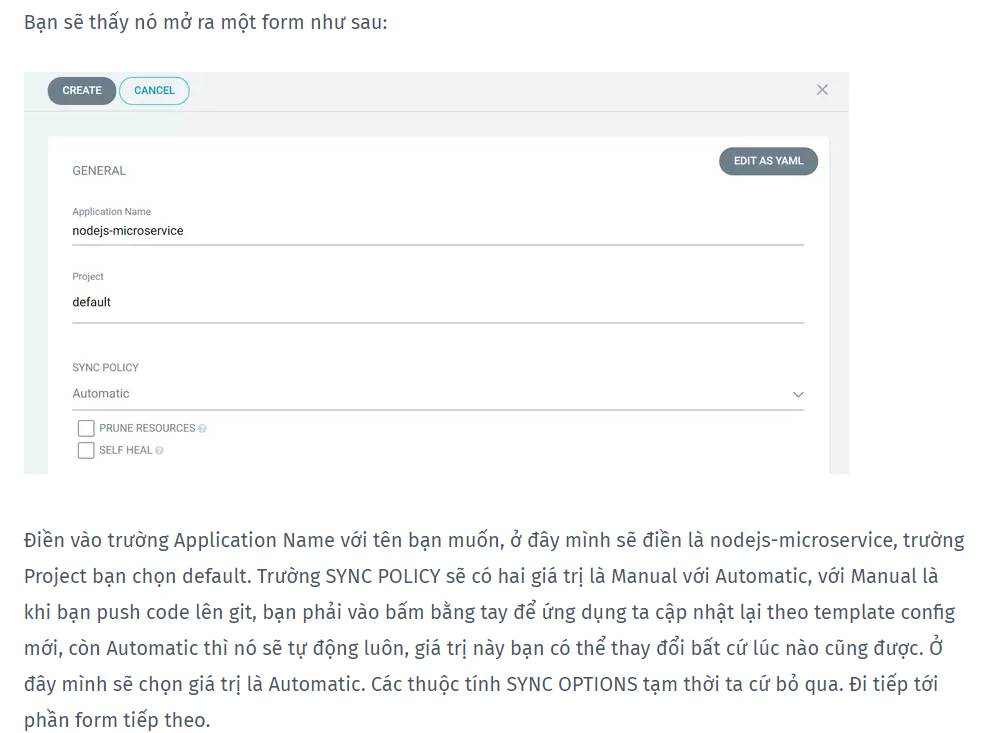
Ok, vậy là ta đã cài đặt xong được Argocd, bước tiếp theo là ta dùng nó để tự động tạo ứng dụng và cập nhật lại ứng dụng theo application template của chúng ta.

### Tạo một App trên Argocd

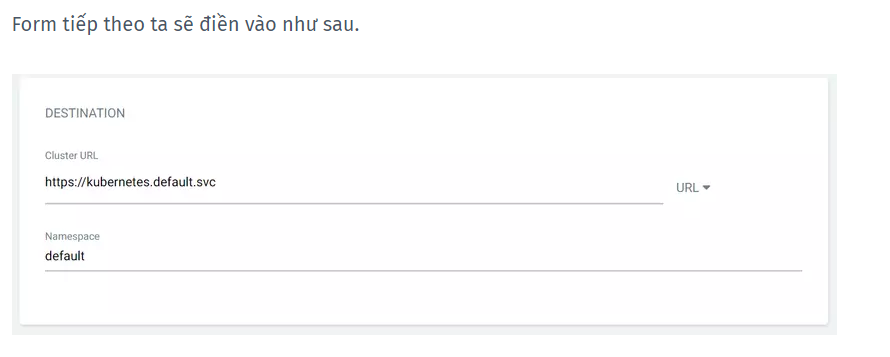
Để kết nối Argocd tới kubernetes cluster và git repo, ta cần tạo một APP trên Argocd. Nhấn vào nút **+ New App** ở trên UI.



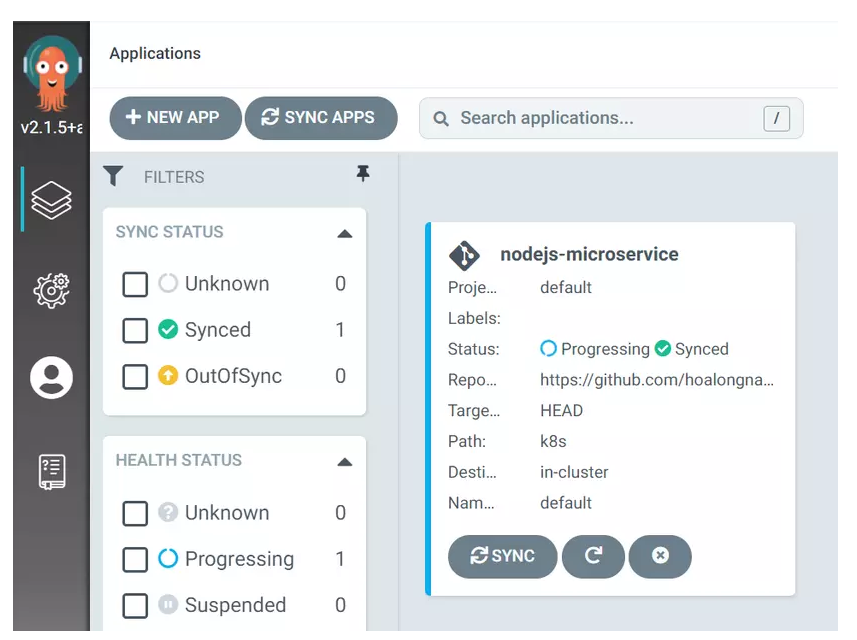
Bạn sẽ thấy nó mở ra một form như sau:







Với Cluster URL bạn điền giá trị https://kubernetes.default.svc, còn namespace bạn điền tên namespace bạn muốn ứng dụng ta deploy tới. Ở đây mình điền là default. Sau khi điền sau hết, bạn nhấn nút create, lúc này Argocd sẽ tạo một APP cho chúng ta và tiến hành deploy ứng dụng lên trên kubernetes cluster của chúng ta.



Sau khi tạo xong bạn sẽ thấy UI như hiện tại, bạn để ý thấy trường status, nếu nó hiển thị giá trị synced có nghĩa là nó đã sync ứng dụng của chúng ta lên trên môi trường kubernetes thành công. Bạn có thể kiểm tra bằng cách gõ câu lệnh get deployment để xem các thành phần trong ứng dụng nodejs microservice của chúng ta.

$ kubectl get deploy

NAME READY UP-TO-DATE AVAILABLE AGE

api-gateway 1/1 1 1 4m

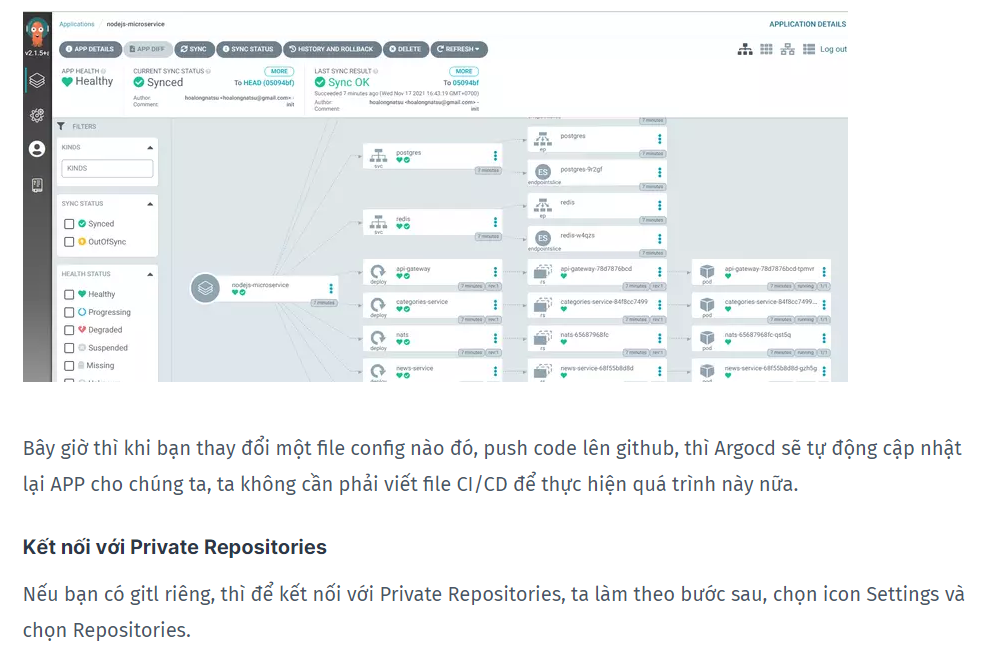
categories-service 1/1 1 1 4m

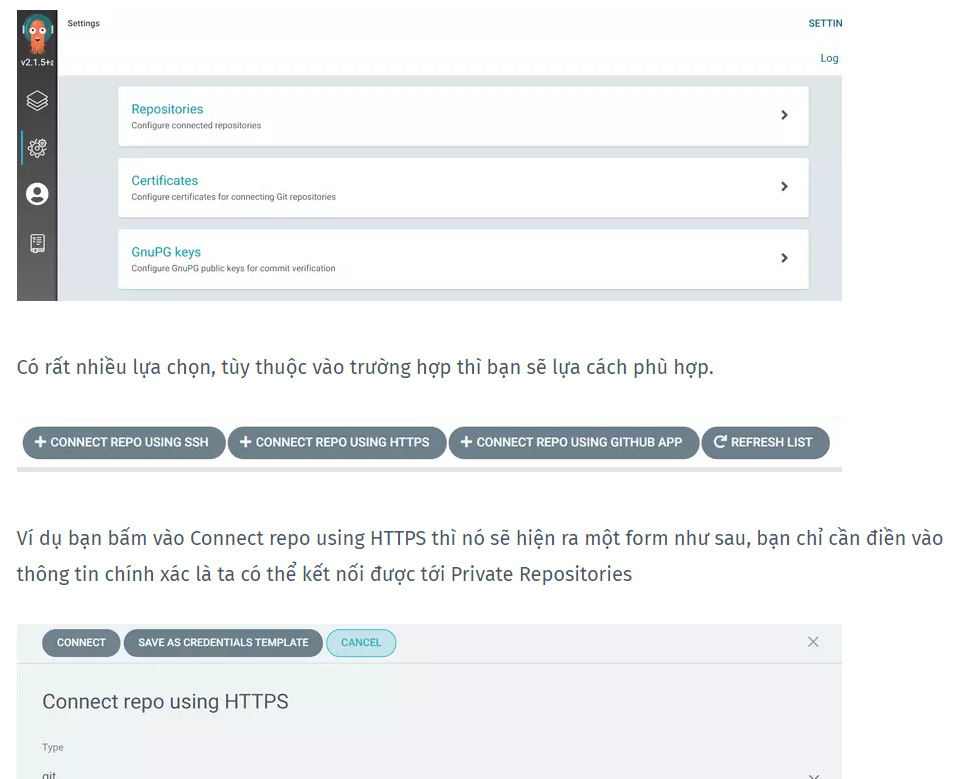
nats 1/1 1 1 4m

news-service 1/1 1 1 4m

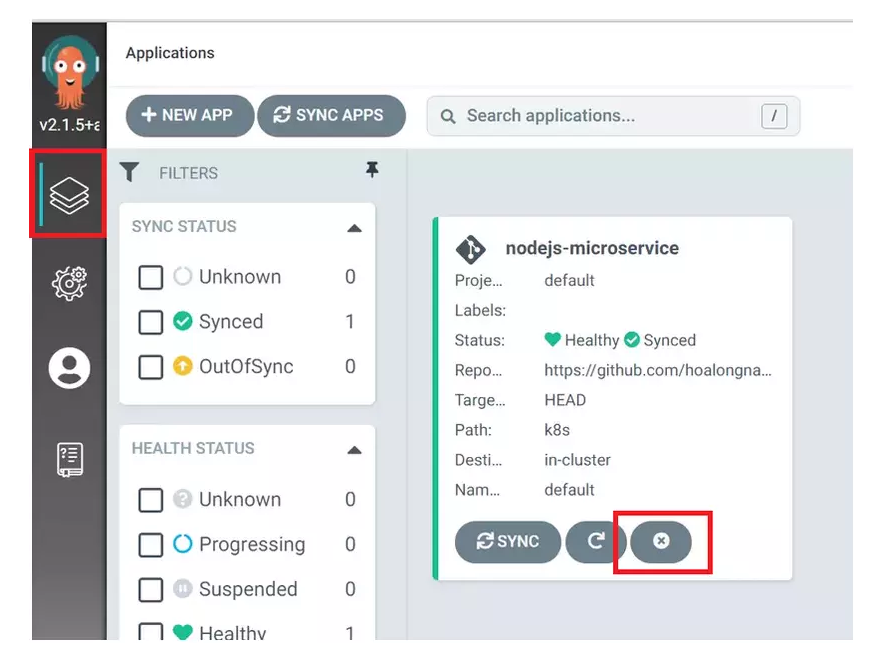
redis 1/1 1 1 4m

Nếu bạn hiển thị ra được tất cả các Deployment thì chúc mừng là ta đã deploy được ứng dụng thành công. Ngoài ra Argocd còn có UI hiển thị các thành phần của chúng ta trên kubernetes kết nối với nhau ra sao và tình trạng hiện của các resource như thế nào, rất hữu ích, bạn nhấn vào APP nodejs-microservice, bạn sẽ thấy UI như sau, rất đẹp và chi tiết.









Bạn chạy câu lệnh get deployment thì sẽ thấy toàn bộ resource của ta đã bị xóa đi.

$ kubectl get deploy

No resources found in default namespace.

Cẩn thận khi xóa APP nhé 😂.

## Kết luận

Vậy là ta đã tìm hiểu xong cơ bản về cách sử dụng Argocd, sử dụng công cụ này sẽ giúp bạn thoải mái hơn trong quá trình cập nhật lại một template config mới cho ứng dụng kubernetes của ta. Ở phần này ta chỉ nói về cách cập nhật template. Phần tiếp theo mình sẽ nói về cách làm sao khi các dev viết code cho chức năng mới xong, dev đẩy lên git repo. Ta sẽ viết CI/CD thế nào để tự động cập nhật lại các chức năng mới cho ứng dụng của chúng ta. Nếu các bạn có thắc mắc hoặc chưa hiểu rõ chỗ nào, các bạn có thể hỏi ở phần comment.

# Phần 3

## [Kubernetes Practice] - Triển khai nodejs microservice trên Kubernetes - phần 3 - Xây dựng CI/CD

By [**Huỳnh Minh Quân**](https://techhay.vn/author/huynh-minh-quan/)

Published in [**DevOps**](https://techhay.vn/category/devops/)

December 03, 2021

**8 min read**

## Giới thiệu

Chào mọi người đến với series practice về kubernetes của mình. Tiếp tục ở bài các phần trước, các bạn nên đọc bài trước để hiểu về hệ thống ta đang triển khai ra sao, [phần 1](https://techhay.vn/kubernetes-practice-trien-khai-nodejs-microservice-tren-kubernetes-phan-1/) nói về cách viết config và cách triển khai microservice lên trên Kubernetes, [phần 2](https://techhay.vn/kubernetes-practice-trien-khai-nodejs-microservice-tren-kubernetes-phan-2/) nói về ta dùng Argocd để tự động cập nhật application khi template config kubernetes của ta thay đổi. Ở bài này, mình sẽ nói về cách khi developer viết code cho chức năng mới của ứng dụng xong và push nó lên trên git repo, thì ta sẽ xây dựng luồng CI/CD thế nào để cập nhật lại ứng dụng với code mới của developer trên môi trường kubernetes.

Ở bài này mình sẽ sử dụng repository là gitlab và chạy CI/CD với gitlab CI. Để làm được bài này thì các bạn cần tạo một repo trên gitlab và đẩy code lên đó nhé, các bạn đặt tên repo là gì cũng được. Các bạn đẩy code nằm ở trong folder code của repo này <https://github.com/hoalongnatsu/microservices.git> lên repo gitlab của các bạn nha. Sau khi xong hết ta bắt tay vào làm.

## Gitlab Runner

Gitlab cung cấp cho ta khá nhiều cách để chạy CI/CD, mình sẽ nói tới phần đơn giản nhất trước, là ta sử dụng Gitlab Runner. Gitlab Runner là một application mà ta sẽ cài ở trên server ta muốn nó chạy CI/CD. Để install gitlab runner, các bạn ssh tới server mà mình muốn chạy CI/CD trên đó, và cài runner như sau, mình sẽ hướng dẫn cài ở trên server linux.

### Install gitlab-runner

**Download the binary for your system**

Chọn OS phù hợp với server của các bạn nhé.

# Linux x86-64

sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64"

# Linux x86

sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-386"

# Linux arm

sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-arm"

# Linux arm64

sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-arm64"

# Linux s390x

sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-s390x"

# Linux ppc64le

sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-ppc64le"

**Give it permissions**

sudo chmod +x /usr/local/bin/gitlab-runner

**Create a GitLab CI user**

sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash

**Install and run as service**

sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner

sudo gitlab-runner start

Nếu bạn gặp lỗi sudo: gitlab-runner: command not found thì mở /etc/sudoers lên và thêm vào ở Defaults secure\_path với dường dẫn /usr/local/bin nữa, còn không thì bạn install gitlab-runner ở dường dẫn /usr/bin cũng được.

Để install trên các môi trường khác thì các bạn xem thêm ở đây <https://docs.gitlab.com/runner/install/>.

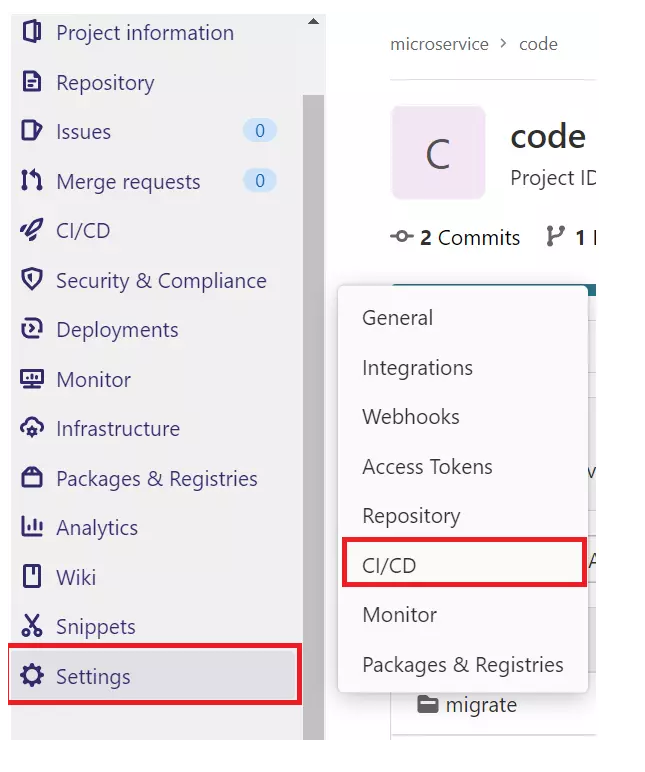
Sau khi install xong thì bạn có thể dùng câu lệnh này để kiểm tra:

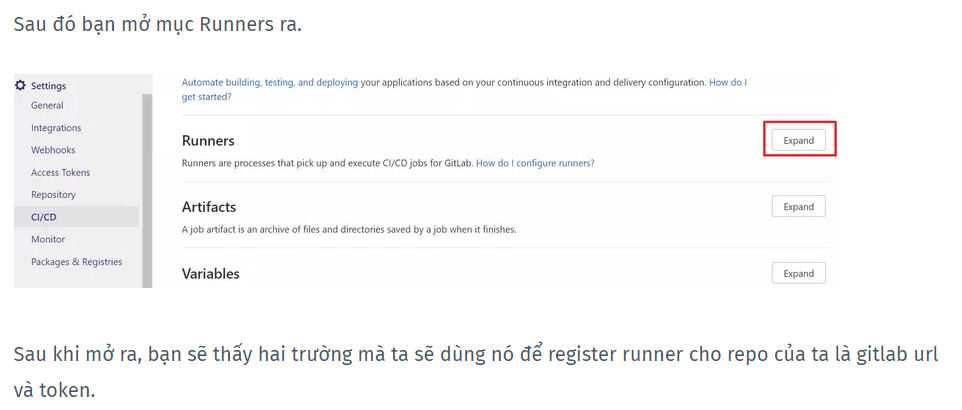
$ sudo gitlab-runner list

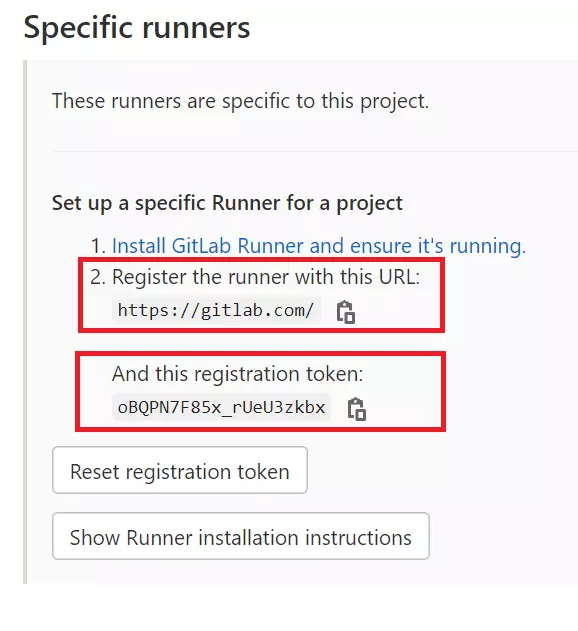
Runtime platform arch=amd64 os=linux pid=2850 revision=de104fcd version=14.5.1

Listing configured runners ConfigFile=/etc/gitlab-runner/config.toml

Câu lệnh này là để ta list toàn bộ runner hiện tại trên server, bây giờ thì tất niên khi ta list ra thì sẽ không có runner nào 😁. Giờ thì ta sẽ đăng ký một con runner để nó chạy CI/CD cho repo của ta ở trên gitlab. Bạn mở gitlab lên, ở chỗ repo của mình chọn mục Settings -> CI/CD.







### Register runner

Giờ thì ta sẽ register một con runner cho repo của ta lên trên server, các bạn làm theo các bước như sau, đầu tiên ta chạy câu lệnh register.

sudo gitlab-runner register

Sau khi run thì nó sẽ bắt bạn nhập gitlab url vào, nhập giá trị url ở phía trên vào. Tiếp đó nó sẽ bắt bạn nhập token, bạn nhập token ở trên vào.

Tiếp theo là nó bắt bạn nhập description, chỗ này thì bạn nhập gì cũng được. Tiếp theo là phần tags, phần này thì quan trọng, lúc chạy CI thì runner sẽ được chạy dựa vào tags, bạn nhập ở đây là microservice nhé.

Bây giờ thì sẽ tới bước ta chọn runner của ta sẽ được thực thi ra sao. Nó sẽ chạy thẳng trên server, hay là mỗi lần chạy nó sẽ tạo một container ra và chạy job CI/CD trên container đó. Chọn shell nếu các bạn muốn runner chạy thẳng trên môi trường server, chọn docker nếu bạn muốn runner chạy trong môi trường docker. Ở đây thì mình sẽ chọn docker, các bạn nên chọn theo mình để tránh bị lỗi nha.

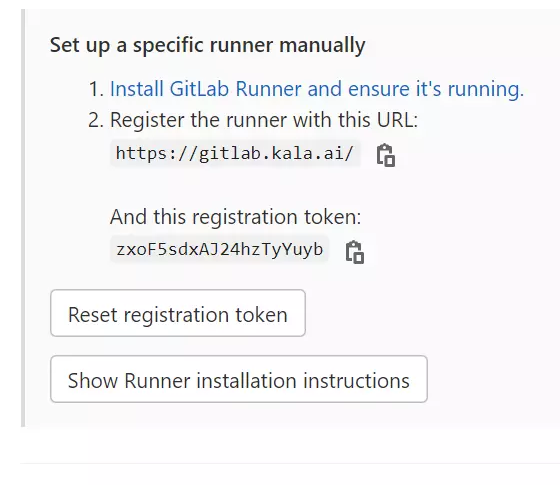
Tiếp theo nó sẽ hỏi default Docker image thì bạn nhập vào là docker:stable, tới đây thì ta đã register runner cho repo của ta thành công, để kiểm tra các runner ta đã đăng ký thì ta chạy câu lệnh list ở trên, giờ thì bạn sẽ thấy được runner mà ta vừa đăng ký.

Runtime platform arch=amd64 os=linux pid=2969 revision=de104fcd version=14.5.1

Listing configured runners ConfigFile=/etc/gitlab-runner/config.toml

microservice Executor=docker Token=zC8B6PZzDouA46mgGry6 URL=https://gitlab.kala.ai/

Kiểm tra trên gitlab bạn sẽ thấy nó hiển thị con runner của ta.



Oke vậy là ta đã xong bước register runner. Và thay vì phải nhập từng bước khá lằng nhằng như trên thì bạn có thể register nhanh một con runner bằng câu lệnh sau:

sudo gitlab-runner register -n \

--url https://gitlab.com/ \

--registration-token REGISTRATION\_TOKEN \

--executor docker \

--description "microservice" \

--docker-image "docker:stable" \

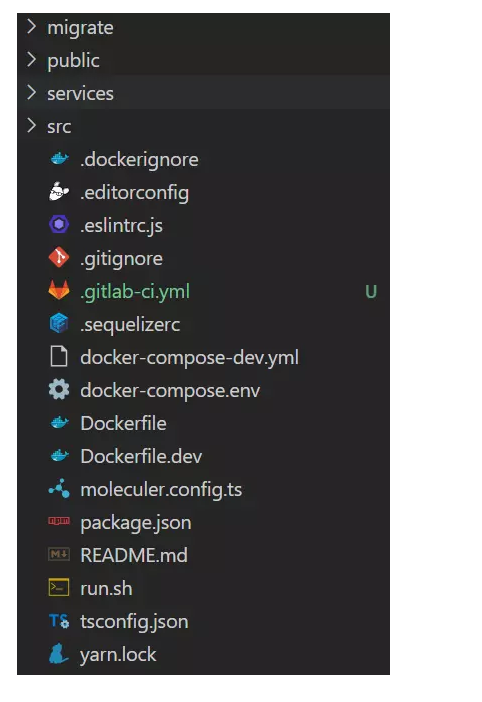
--docker-privileged

Giờ thì ta chỉ cần viết file CI/CD ở trong repo của chúng ta, và mỗi lần ta đẩy code lên thì file CI/CD này sẽ được con runner thực thi. **Lưu ý là vì ta chọn docker nên bắt buộc trên server chạy CI/CD của ta phải có cài docker rồi thì runner mới thực thi được**. Tiếp theo ta sẽ bắt tay vào làm luồng CI.

## Continuous Integration

Đầu tiên là ta sẽ làm bước mà integrate code mới của developer với image mà dùng để chạy ứng dụng của ta trước, sau đó ta mới làm bước deploy. Ở bước integrate này thì ta sẽ viết file CI, công việc của nó sẽ là build image tương ứng với code của branch hiện tại, sau đó sẽ push image đó lên docker registery (nơi ta chứa image).

Gitlab có hỗ trợ cho ta image registry, nên ta sẽ sử dụng nó luôn cho tiện. Để viết CI/CD cho gitlab thì ta sẽ viết trong file .gitlab-ci.yml, ở folder mà ta chứa code của repo, tạo một file .gitlab-ci.yml với config như sau:



stages:

- build

build root image:

stage: build

tags:

- microservice

script:

- docker build . -t $CI\_REGISTRY\_IMAGE:latest -t $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

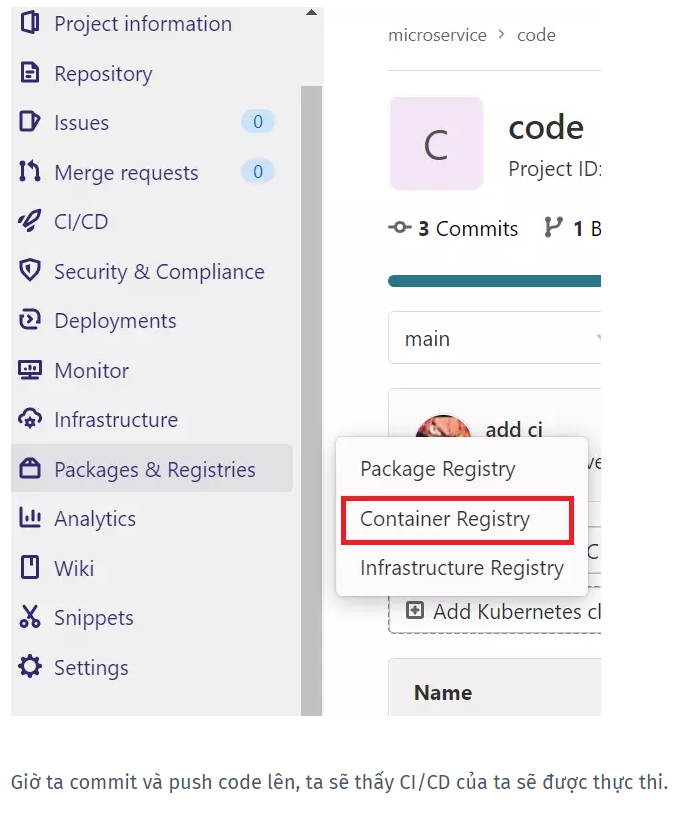
- docker push $CI\_REGISTRY\_IMAGE:latest

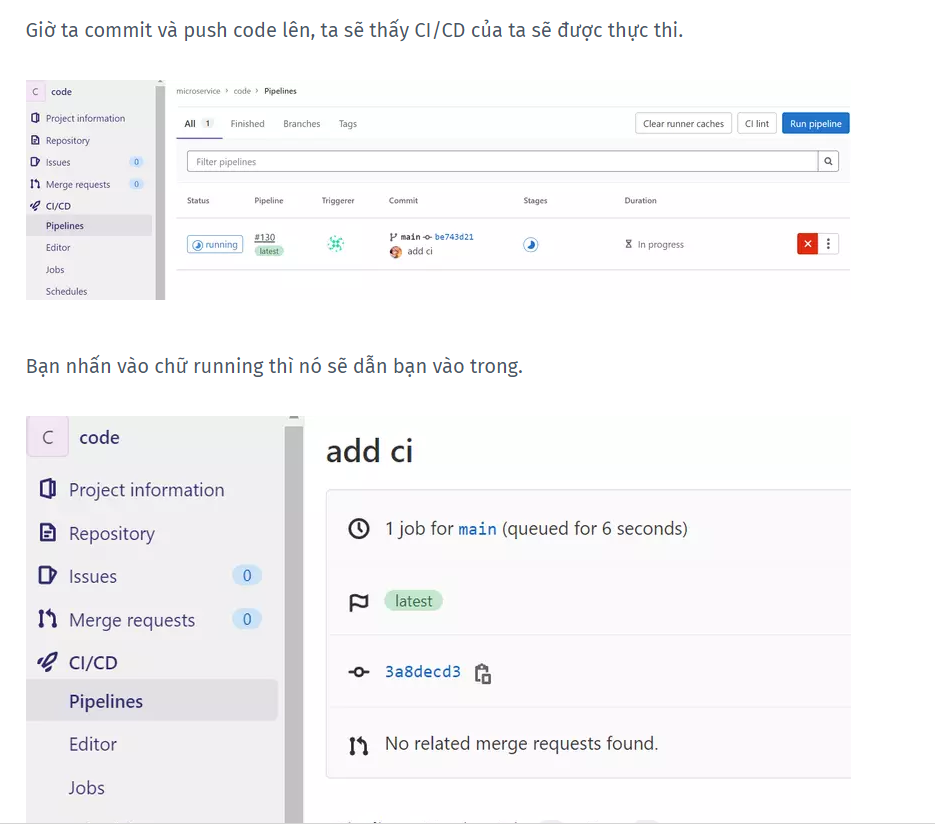
- docker push $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

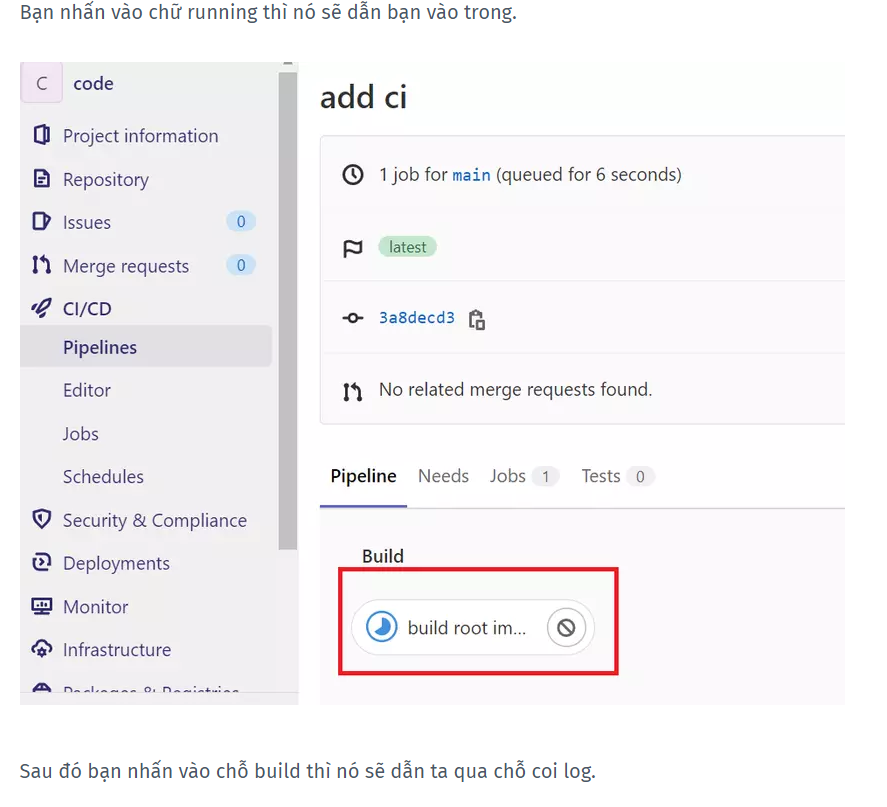
only:

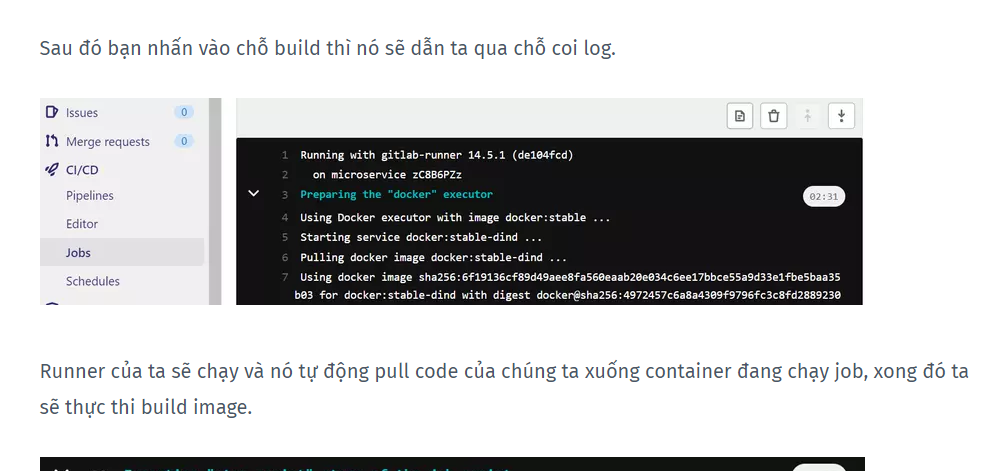
- main

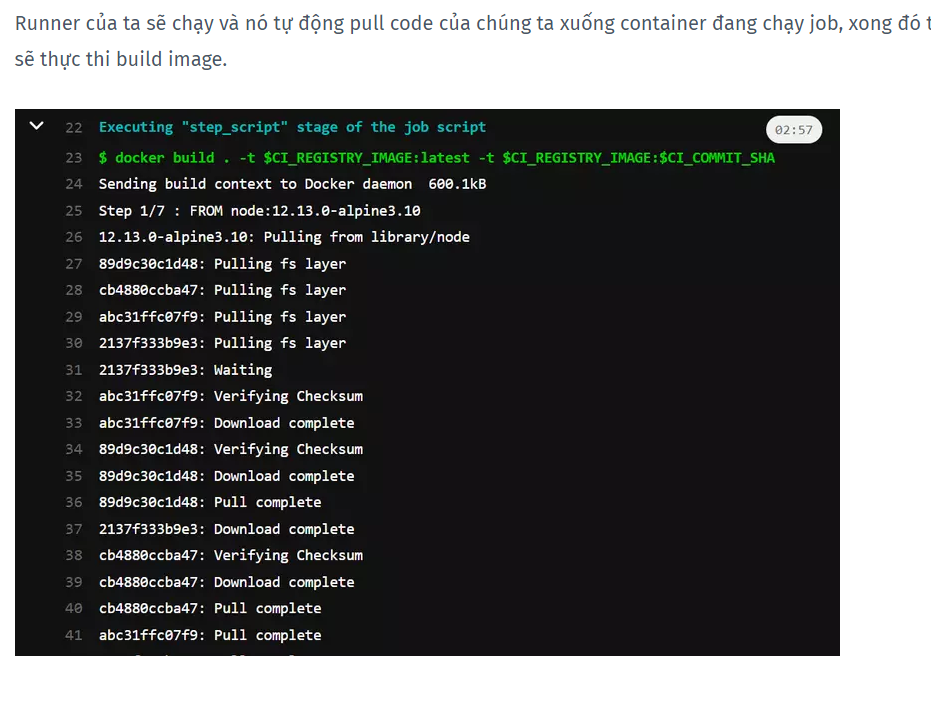
Trường stages là ta sẽ định nghĩa những pipe trong một job của ta, hiện tại thì ta chỉ có một pipe là build thôi. Ở trong file config trên ta sẽ có biến là CI\_REGISTRY\_IMAGE, đây là một trong những biến mặc định của gitlab CI, biến CI\_REGISTRY\_IMAGE sẽ là tên image của chúng ta. Ở trên gitlab repo, bạn bấm sang phần **Package & Registries -> Container Registry**, thì bạn sẽ thấy được tên image của chúng ta. Khi ta push image thì ta sẽ push lên chỗ Container Registry này. Trường script là trường ta sẽ chỉ định những công việc ta sẽ làm, ở file trên đơn giản là ta sẽ build image và push image với code mới nhất lên.

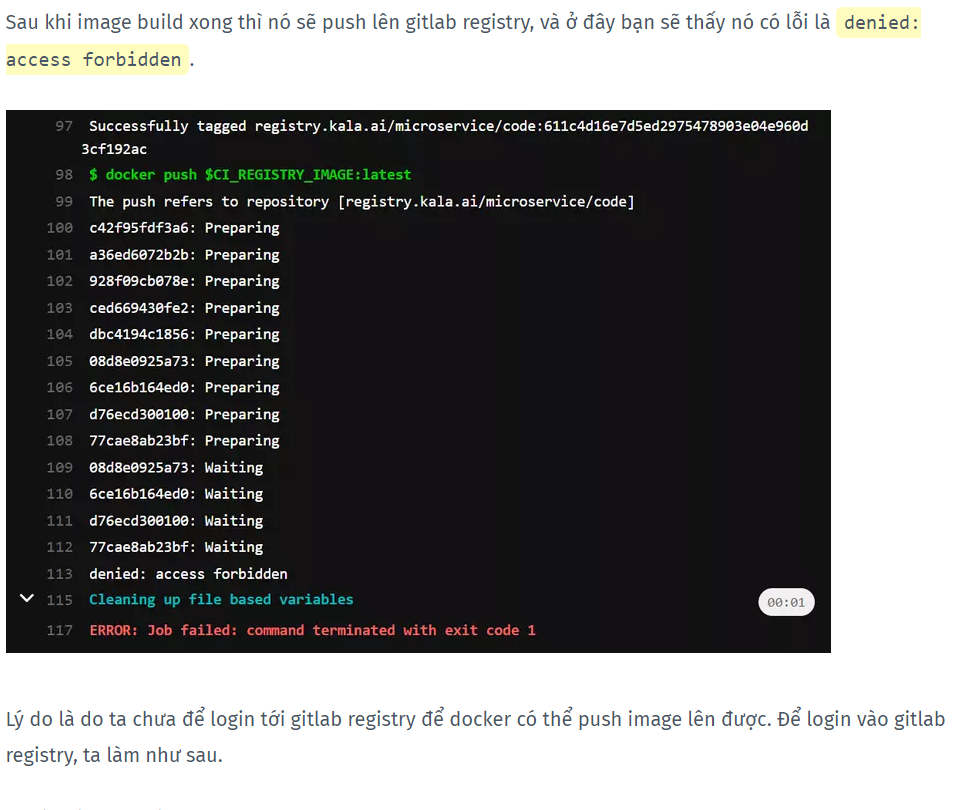


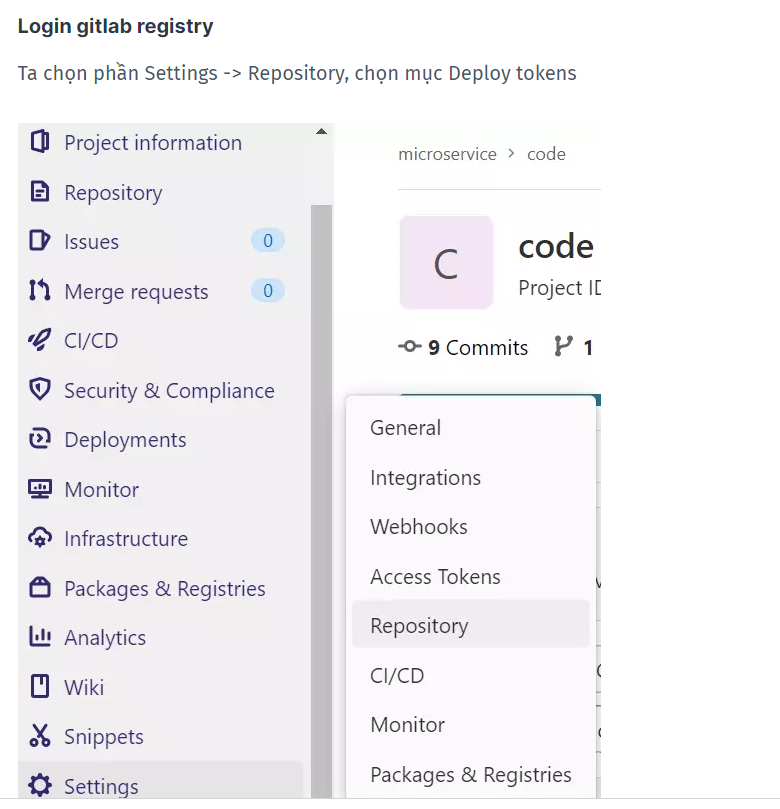


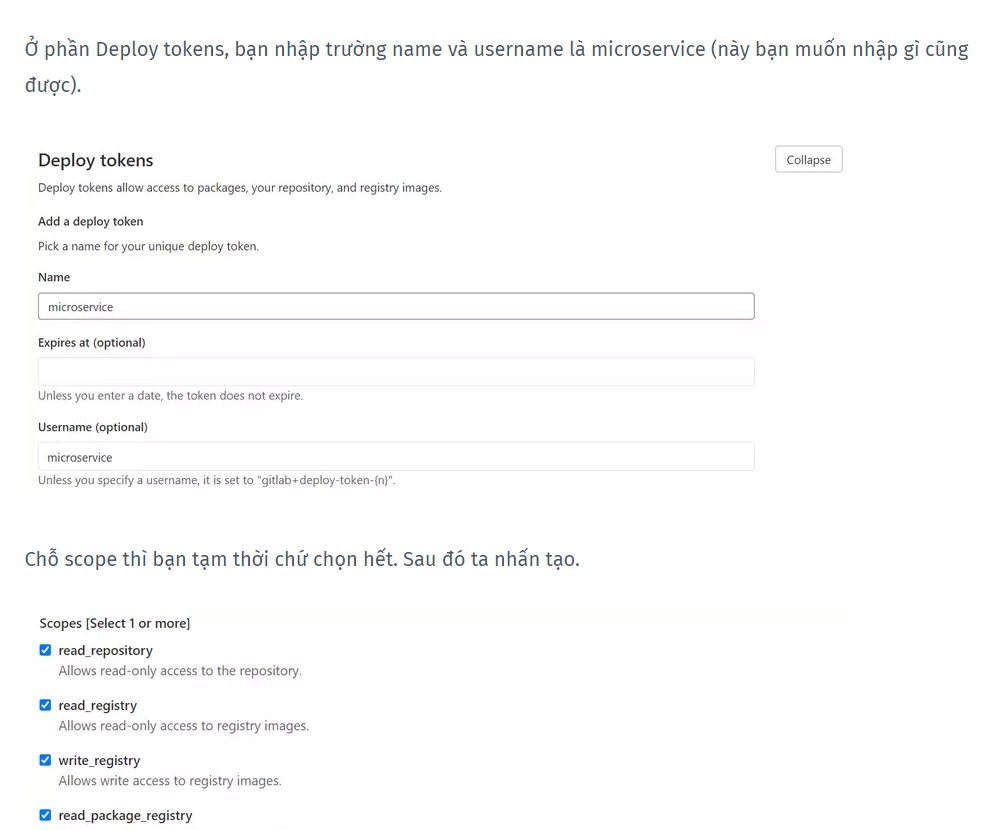


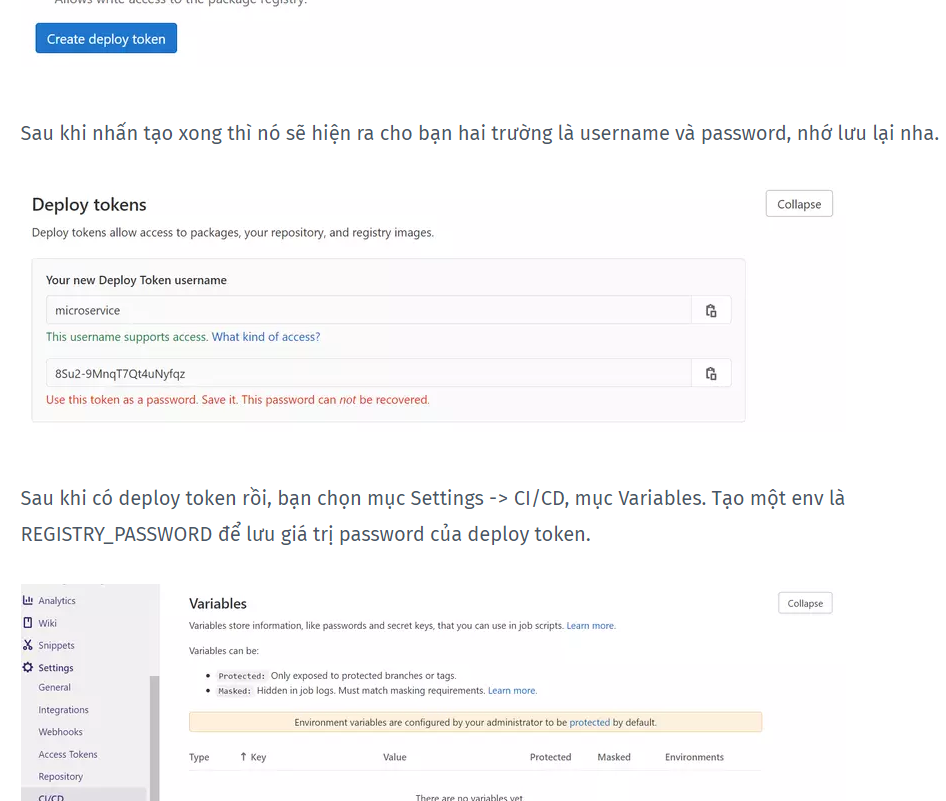


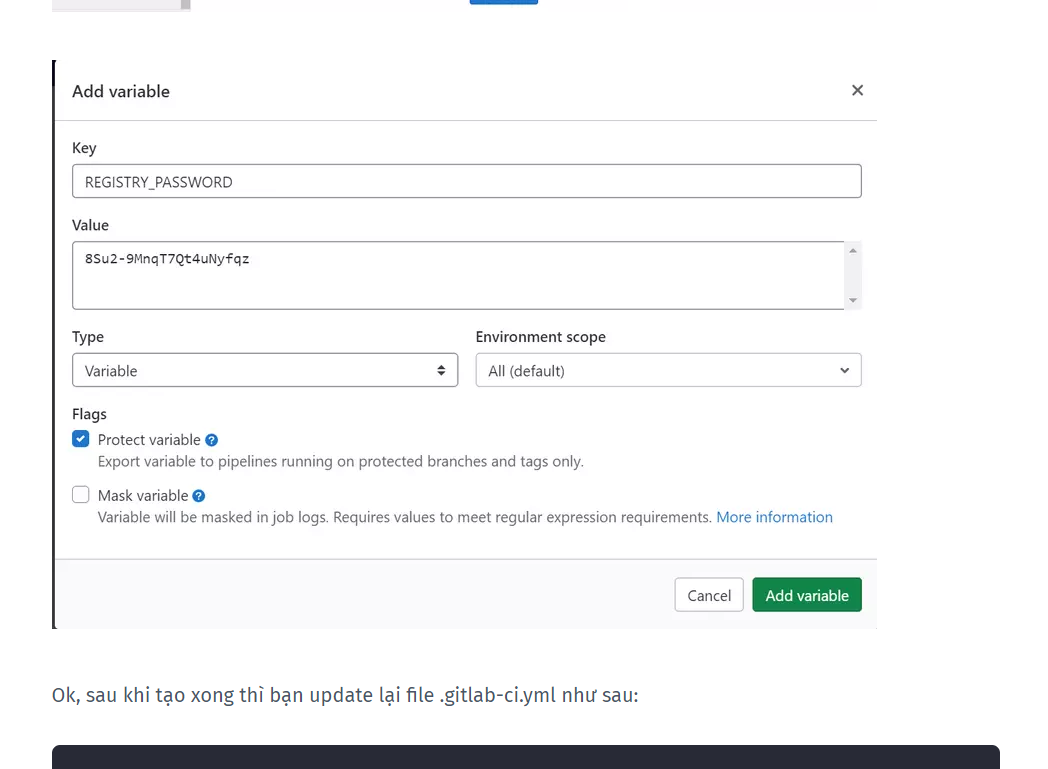












Ok, sau khi tạo xong thì bạn update lại file .gitlab-ci.yml như sau:

stages:

- build

build root image:

stage: build

tags:

- microservice

before\_script:

- echo "$REGISTRY\_PASSWORD" | docker login $CI\_REGISTRY -u microservice --password-stdin

script:

- docker build . -t $CI\_REGISTRY\_IMAGE:latest -t $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

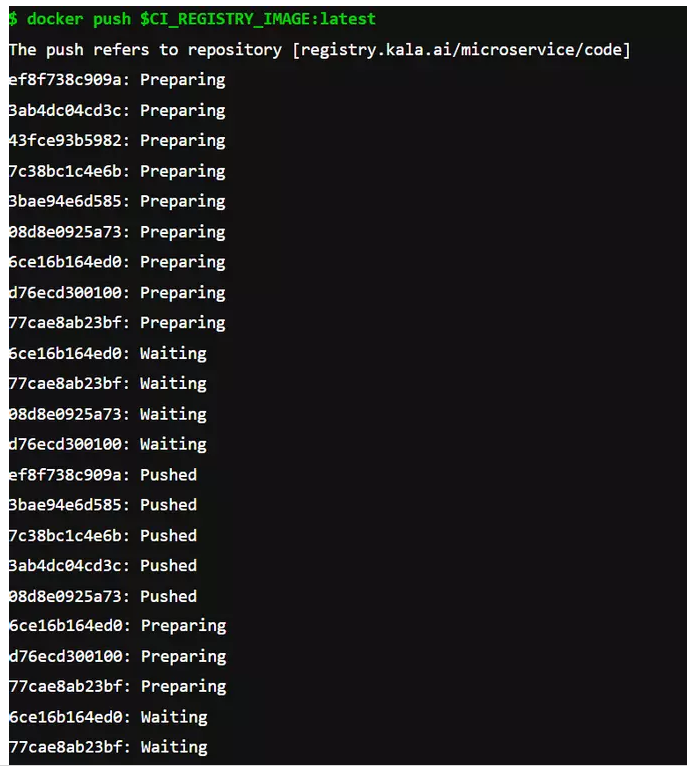
- docker push $CI\_REGISTRY\_IMAGE:latest

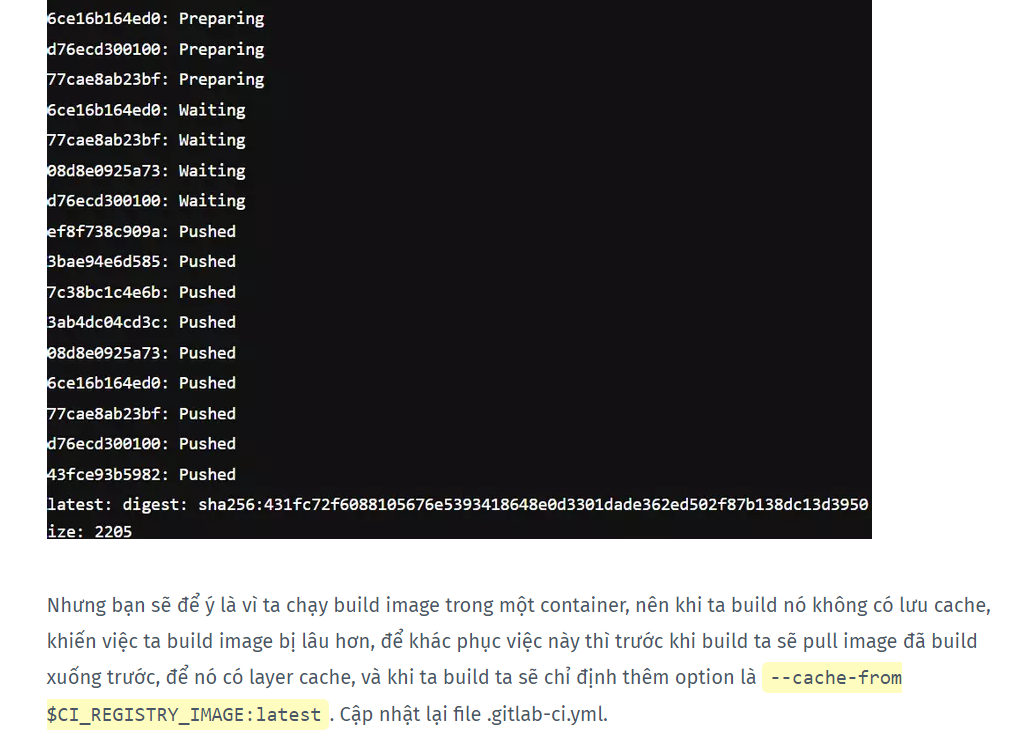
- docker push $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

only:

- main

Ta sẽ thêm trường before\_script để chạy câu lệnh login vào gitlab registry. Biến CI\_REGISTRY là biến mặc định của gitlab CI, biến REGISTRY\_PASSWORD là ta vừa mới tạo. Cập nhật xong thì ta commit code và push nó lên lại. Lúc này bạn sẽ thấy là ta đã build được image và push nó lên thành công.





stages:

- build

build root image:

stage: build

tags:

- microservice

before\_script:

- echo "$REGISTRY\_PASSWORD" | docker login $CI\_REGISTRY -u microservice --password-stdin

script:

- docker pull $CI\_REGISTRY\_IMAGE:latest || true

- docker build . --cache-from $CI\_REGISTRY\_IMAGE:latest -t $CI\_REGISTRY\_IMAGE:latest -t $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

- docker push $CI\_REGISTRY\_IMAGE:latest

- docker push $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

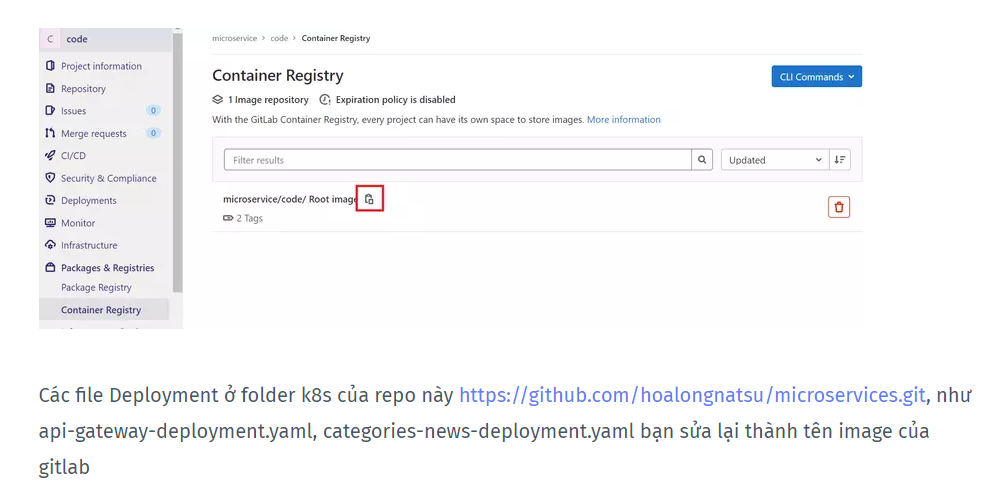
only:

- main

Commit code và push lên, lúc này thì image của ta sẽ được build với tốc độ nhanh hơn.

### Cập nhật lại file config của kubernetes

Tiếp theo ta sẽ cập nhật lại file config Deployment với image mà ta mới build. Để coi tên image, bạn chọn mục copy ở phần Container Registry.



# api-gateway-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-gateway

labels:

component: api-gateway

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: api-gateway

template:

metadata:

labels:

component: api-gateway

spec:

containers:

- name: api-gateway

image: registry.kala.ai/microservice/code

ports:

- name: http

containerPort: 3000

protocol: TCP

livenessProbe:

httpGet:

path: /

port: http

readinessProbe:

httpGet:

path: /

port: http

env:

- name: SERVICES

value: api

- name: PORT

value: "3000"

envFrom:

- configMapRef:

name: microservice-cm

# categories-news-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: categories-service

labels:

component: categories-service

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: categories-service

template:

metadata:

labels:

component: categories-service

spec:

containers:

- name: categories-service

image: registry.kala.ai/microservice/code

env:

- name: SERVICES

value: categories

envFrom:

- configMapRef:

name: microservice-cm

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: news-service

labels:

component: news-service

spec:

revisionHistoryLimit: 1

selector:

matchLabels:

component: news-service

template:

metadata:

labels:

component: news-service

spec:

containers:

- name: news-service

image: registry.kala.ai/microservice/code

env:

- name: SERVICES

value: news

envFrom:

- configMapRef:

name: microservice-cm

Image của mình thì mình sửa lại là registry.kala.ai/microservice/code. Sau đó bạn nếu bạn làm như theo bài trước thì bạn chỉ cần push lên repo chứa template config để Argocd nó tự động cập nhật lại. Còn không thì bạn chạy câu lệnh sau ở folder k8s.

$ kubectl apply -f k8s --recursive

deployment.apps/api-gateway created

deployment.apps/categories-service created

deployment.apps/news-service created

configmap/microservice-cm created

deployment.apps/nats created

service/nats created

service/postgres created

statefulset.apps/postgres created

deployment.apps/redis created

service/redis created

Nhưng khi bạn get pod ra, bạn sẽ thấy mấy Deployment mà ta chỉ định dùng image của gitlab thì nó sẽ bị lỗi là ErrImagePull.

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-84677bf776-jj8x4 0/1 ErrImagePull 0 86s

categories-service-867f848c77-l2c4m 0/1 ImagePullBackOff 0 86s

nats-65687968fc-rbtd4 1/1 Running 0 86s

news-service-686b8557c8-ch2v5 0/1 ImagePullBackOff 0 86s

postgres-0 1/1 Running 0 86s

redis-58c4799ccc-xn9zk 1/1 Running 0 86s

$ kubectl describe pod api-gateway-84677bf776-jj8x4

...

Type Reason Age From Message

---- ------ ---- ---- -------

...

Normal Pulling 61s (x4 over 2m48s) kubelet Pulling image "registry.kala.ai/microservice/code"

Warning Failed 61s (x4 over 2m37s) kubelet Failed to pull image "registry.kala.ai/microservice/code": rpc error: code = Unknown desc = Error response from daemon: pull access denied for registry.kala.ai/microservice/code, repository does not exist or may require 'docker login': denied: requested access to the resource is denied

Warning Failed 61s (x4 over 2m37s) kubelet Error: ErrImagePull

Warning Failed 36s (x6 over 2m37s) kubelet Error: ImagePullBackOff

Normal BackOff 25s (x7 over 2m37s) kubelet Back-off pulling image "registry.kala.ai/microservice/code"

Để kubernetes có thể pull được image từ gitlab, ta cần tạo một Secret với loại là docker-registry rồi chỉ định vào trường **imagePullSecrets** khi khai báo config cho Pod. Ta tạo Secret bằng câu lệnh sau.

$ kubectl create secret docker-registry microservice-registry --docker-server=https://registry.kala.ai --docker-username=microservice --docker-password=8Su2-9MnqT7Qt4uNyfqz

secret/microservice-registry created

Với **—docker-server** là tên gitlab server của bạn, **—docker-username** là username của deploy token ta tạo khi nãy, và **—docker-password** là password của deploy token. Cập nhật lại file Deployment thêm vào trường imagePullSecrets với Secret ta vừa tạo.

# api-gateway-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-gateway

labels:

component: api-gateway

spec:

...

spec:

imagePullSecrets:

- name: microservice-registry

containers:

- name: api-gateway

image: registry.kala.ai/microservice/code

ports:

- name: http

containerPort: 3000

protocol: TCP

...

# categories-news-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: categories-service

labels:

component: categories-service

spec:

...

spec:

imagePullSecrets:

- name: microservice-registry

containers:

- name: categories-service

image: registry.kala.ai/microservice/code

...

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: news-service

labels:

component: news-service

spec:

...

spec:

imagePullSecrets:

- name: microservice-registry

containers:

- name: news-service

...

$ kubectl apply -f k8s --recursive

deployment.apps/api-gateway configured

deployment.apps/categories-service configured

...

statefulset.apps/postgres configured

...

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

api-gateway-6c9f8f68b7-mj6r9 1/1 Running 0 40s

categories-service-84db8d885b-g85md 1/1 Running 0 40s

nats-65687968fc-rbtd4 1/1 Running 0 19m

news-service-76d559db69-vqg59 1/1 Running 0 40s

postgres-0 1/1 Running 0 19m

redis-58c4799ccc-xn9zk 1/1 Running 0 19m

Vậy là ta đã hoàn thành xong bước CI, tiếp theo ta sẽ xem cách làm CD.

## Continuous deployment

Bước này thì ta cũng sẽ có nhiều cách, mình sẽ nói về cách dễ nhất trước là ta sẽ ssh lên kubernetes master và thực hiện câu lệnh kubectl, các cách khác mình sẽ nói ở bài sau. Cập nhật lại file .gitlab-ci.yml như sau:

stages:

- build

- deploy

build root image:

stage: build

tags:

- microservice

script:

- docker build . -t $CI\_REGISTRY\_IMAGE:latest -t $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

- docker push $CI\_REGISTRY\_IMAGE:latest

- docker push $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

only:

- main

deploy stage:

stage: deploy

tags:

- microservice

before\_script:

- mkdir ~/.ssh

- echo -e "${SERVER\_KEY\_PEM//\_/\\n}" > ~/.ssh/key.pem

- apt-get -y update && apt-get -y install openssh-client rsync grsync

- ssh-keyscan -H $SERVER\_IP >> ~/.ssh/known\_hosts

- chmod 400 ~/.ssh/rnd-ecommerce.pem

script:

- |

sudo ssh -i ~/.ssh/key.pem $SERVER\_USER@$SERVER\_IP << EOF

kubectl set image deployment/api-gateway api-gateway=$CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

kubectl set image deployment/news-service news-service=$CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

kubectl set image deployment/categories-service categories-service=$CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_SHA

exit

EOF

only:

- main

Ta sẽ thêm một stages nữa là deploy. Để ssh được tới server thì tùy cách bạn dùng là password hoặc key pem thì sẽ khác nhau. Của mình sẽ dùng cách key pem. Bạn vào phần Settings -> CI/CD -> Variables khi nãy, khai báo thêm các biến là SERVER\_KEY\_PEM, SERVER\_USER, SERVER\_IP tương ứng với server mà đang chạy kubernetes master. Khi ta build image và push lên gitlab xong, job deploy sẽ được chạy, nó sẽ ssh tới kubernetes master và cập nhật lại Deployment với image là image là mới build xong. **Vì ở đây mình làm demo nên mình gộp 3 câu cập nhật Deployment lại trong một repo, còn thực tế các bạn nên làm mỗi repo một Deployment khác nhau nhé**.

Commit và push code lên, lúc này bạn sẽ thấy pipe của ta sẽ có 2 job là build với deploy.

