DEVOPS

Created	@January 14, 2025 9:35 PM
Reviewed	☐

▼ MỤC TIÊU

▼ GIAI ĐOẠN 1: On-promise

- Set up hệ thống CI/CD, Log & Monitoring , Deploy dự án CMS trên môi trường Promise (tự setup)
- CI/CD: Sử dụng gitlab CI / Jenkins
- Docker register: Docker hub(public)/ harbor(private)
- Monitoring: Triển khai hệ thống giám soát toàn diện với PNG Stack (Prometheus, Exporter, Grafana)
- K8s

▼ GIAI ĐOẠN 2: CLOUD

- Triển khai các bước trên trên môi trường cloud bao gồm:
  - Thuê máy chủ
  - Mua domain
  - SSL/TLS

▼ JD INTERN

- Linux
- CI/CD Tool
- Docker
- IaC
- microservices
- K8s, heml
- git

▼ Lộ trình học devops

1. ~~LINUX~~
2. ~~GIT~~
3. ~~CICD~~
4. ~~Docker~~
5. Jenkins
6. ~~Monitors~~
7. ~~Kubernetes~~

▼ 8. Các core

▼ Networking

- SSH, cấu hình Secure cho SSH
- NFS, cấu hình Secure cho NFS

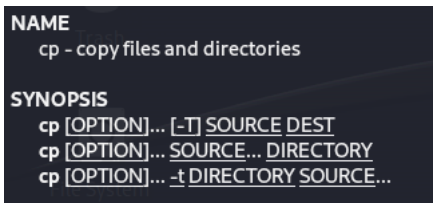
- OS

        - Cách hoạt động của OS
        - socket, threading, v.v
        - Các core concept
    - 9. Cloud
- Phỏng vấn Devops

  https://devopsedu.vn/cac-cau-hoi-phong-van-devsecops-hay-gap/
- DEVOPS FOR FRESHER
    - Linux
        - Common command
            - **pwd**: Hiển thị tên của địa chỉ đang làm việc hiện tại
            - **man**: hiển thị cách sử dụng 1 lệnh
            - **cd**: truy cập vào một địa chỉ
                - lệnh cd : default là /home/
                - cd .. : lùi lại 1 thư
            - **ls** : hiện thị nội dung địa chỉ hiện tại chứa

              ! Có thể ẩn tệp bằng cách thêm . trước tên của tệp đó

              ⇒ ls có thể xem cả tệp bị ẩn bằng lệnh: ls -a
                - ls -h : hiện thị thông tin chi tiết
                - ls -ha : kết hợp 2
            - **clear** : xóa màn
            - **mkdir** : tạo thư mục (không thể tạo thư mục mới trùng tên
                - mkdir -p : tạo thư mục cha/con
                    - VD : mkdir -p /CTF/Bai1
            - **rmdir** : xóa thư mục(thư mục rỗng)
        - WORKING WITH FILE
            - Ở trong linux, các file có thể trùng tên với nhau
            - **file** : xem định dạng của một
            - **touch** : tạo 1 file, có thể tạo nh file phân cách nhau bởi dấu cách
            - **rm** : xóa file
                - rm -i : hỏi trước khi xác nhận
                - rm -r : xóa thư mục
                - rm -rf : xóa thư mục mà k hỏi (áp dụng với 1 số file)
            - **cp** : sa0ochép



    - cd -r : coppy 1 thư muc
- mv : di chuyển(có thể dùng để đổi tên file nhanh)

```
NAME
    mv - move (rename) files

SYNOPSIS
    mv [OPTION]... [-T] SOURCE DEST
    mv [OPTION]... SOURCE... DIRECTORY
    mv [OPTION]... -t DIRECTORY SOURCE...
```

    - mv file1.txt file 2 .txt (ví dụ)
    - Đổi tên thu mục

▼ WORKING WITH FILE CONTENT

- head : hiện thị nội dung 10 dòng đầu của 1 file
    - head -5 : hiện thị nd 5 dòng đầu, có thể thay đổi theo ý muốn
- tail : Hiện thị 10 dòng cuối của file
    - **tail -n _sodong_** : in ra n dòng cuối cùng
    - **tail -f _tenfile_** : xem trực tiếp dữ liệu realtime

```
NAME
    tail - output the last part of files

SYNOPSIS
    tail [OPTION]... [FILE]...

DESCRIPTION
    Print the last 10 lines of each FILE to standard output.  With more than one FILE, precede each with a header giving the file name.

    With no FILE, or when FILE is -, read standard input.

    Mandatory arguments to long options are mandatory for short options too.
```

- cat : xem hết tất cả thông tin của file
    -

```
NAME
    cat - concatenate files and print on the standard output

SYNOPSIS
    cat [OPTION]... [FILE]...

DESCRIPTION
    Concatenate FILE(s) to standard output.

    With no FILE, or when FILE is -, read standard input.
```

    - Có thể đọc nhiều file 1 lúc

```
└─$ cat 1.txt 2.txt 3.txt
Anh
Yeu
Em
```

    - Có thể xuất thôn tin đọc ra 1 file

```
┌──(kali㉿kali)-[~/Desktop/test]
└─$ cat 1.txt 2.txt 3.txt >4.txt
```

- Tạo và viết 1 file (dùng ctrl + d để thoát ra)

```
josh@ubuntu20:~/Documents$ cat > file1.txt
I am having a good day.
```

⇒ Sẽ ghi đè lên nếu file trùng

- Tạo một file mới có nd như file cũ

```
┌──(kali㉿kali)-[~/Desktop/test]
└─$ cat 4.txt
thay doi hay ghi de
```

- less : xem nội dung file, có thể cuộn, tìm kiếm, qua trang , v.v(lợi hại hơn more)
  -

```
SYNOPSIS
     less -?
     less --help
     less -V
     less --version
     less [-[+]aABcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX~]
          [-b space] [-h lines] [-j line] [-k keyfile]
          [-{oO}logfile] [-p pattern] [-P prompt] [-t tag]
          [-T tagsfile] [-x tab,...] [-y lines] [-z lines]
          [-# shift] [+[+]cmd] [--] filename...
     (See the OPTIONS section for alternate option syntax with long option names.)
```

▼ FILE CONSTRUCT

- / : thư mục nguồn
- uptime : hiện thị thời gian hệ điều hành đã chạy

```
NAME
     uptime - Tell how long the system has been running.

SYNOPSIS
     uptime [options]

DESCRIPTION
     uptime gives a one line display of the following information.  The current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

     This is the same information contained in the header line displayed by w(1).

     System load averages is the average number of processes that are either in a runnable or uninterruptable state.  A process in a runnable state is either using the CPU or waiting to use the CPU.  A process in uninterruptable
     state is waiting for some I/O access, eg waiting for disk.  The averages are taken over the three time intervals.  Load averages are not normalized for the number of CPUs in a system, so a load average of 1 means a single
     CPU system is loaded all the time while on a 4 CPU system it means it was idle 75% of the time.
```

- frehtop
- ps -A : báo cáo ảnh chụp nhanh của các quy trình hiện tại
- df : hiện thị dung lượng lưu
- fdisk : hiện thị dung lượng lưu trữ
- lsblk : hiện thị danh sách các khối thiết bị
- **htop : hiển thị đầy đủ nhất**

▼ NETWORKING

- **ifconfig**: hiển thị cấu hình mạng
- **ip** : cũng như trên mà có nh tính năng

▼ PACKAGE MANAGE

- sudo apt update : cập nhập phần mềm(tải bản cập nhập mới nhất về chứ chưa nâng cấp)
- sudo apt list — upgradable : xem danh sách có thể update
- sudo apt upgrade : nâng cấp phần mềm
- sudo apt remove : gỡ phần mềm
- sudo apt search + (tên app): tra cứu app

▼ TEXT EDITOR

- nano : là 1 text editor
- vim : 1 text editor khác, nhma thao tác với editor này bằng lệnh
  - Có 2 chế độ:
    - Insert mode : i
    - Command mode
  - **:wq** - ghi và thoát
  - **:q!** : thoát mà không ghi
  - **esc** : thoát khỏi chế độ ...
  - **dd**: xóa 1 dòng
  - **u** : giống CTRL + Z
  - **yy**: copy
  - **p**: past
  - **/** : Tìm kiếm từ khóa
    - n : tìm kiếm từ khóa match kế tiếp
    - N: tìm kiếm từ khóa match đằng trước

▼ **CÁC CÂU LỆNH PHỤC VỤ DỰ ÁN**

- `free`: Hiển thị thông tin bộ nhớ RAM (dung lượng sử dụng và còn trống).
  - Tham số hay dùng: `h` (hiển thị theo đơn vị dễ đọc, như MB/GB).
  - Ví dụ: `free -h`
- `df`: Hiển thị thông tin dung lượng ổ đĩa (đã dùng và còn trống).
  - Ví dụ: `df -h` (hiển thị dưới dạng đơn vị dễ đọc).
- `htop`: Hiển thị danh sách các tiến trình đang chạy và tài nguyên hệ thống (CPU, RAM).
  - Nhấn `q` để thoát.
- `ps`: Hiển thị các tiến trình đang chạy.
  - `ps aux`: Hiển thị tất cả các tiến trình với thông tin chi tiết.
  - **ps - ef** :
    - `e`: Hiển thị tất cả các tiến trình trên hệ thống (equivalent to `-all`).
    - `f`: Hiển thị thông tin chi tiết (full-format listing) về các tiến trình.
- `hostnamectl`: Hiển thị hoặc thay đổi thông tin máy (tên máy, hệ điều hành, kernel, v.v.).
  - Ví dụ:
    - Xem thông tin: `hostnamectl`

        - Đổi hostname: `sudo hostnamectl set-hostname <tên_mới>`
- `reboot`: Khởi động lại hệ thống.
- `netstat`: Hiển thị thông tin kết nối mạng, cổng đang sử dụng, giao thức.
    - Tham số thường dùng:
        - `tuln`: Liệt kê các cổng đang lắng nghe (listening).
        - Ví dụ: `netstat -tuln`
- `traceroute <tên_host hoặc IP>`: Theo dõi đường đi của gói tin từ máy đến đích (qua các router trung gian).
    - Ví dụ: `traceroute google.com`
    - Nếu chưa cài, bạn có thể cài bằng lệnh:

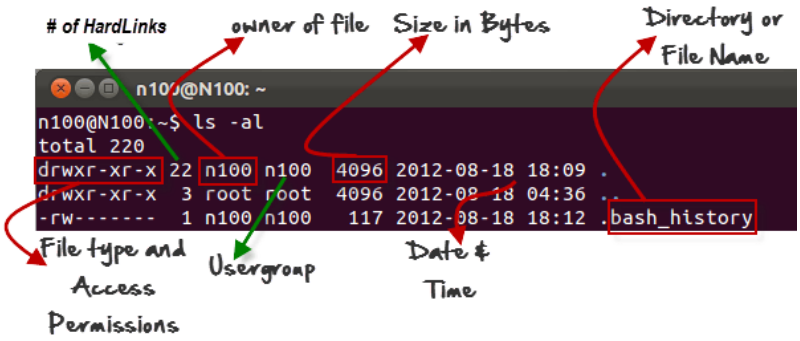
      ```
      bash
      Copy code
      sudo apt install traceroute
      ```
- `telnet <IP hoặc domain> <cổng>`: Kiểm tra kết nối đến một cổng cụ thể trên máy đích.
    - Ví dụ: `telnet 192.168.1.1 22`
- **ps aux | grep <tên tiến trình>** : grep ra tiến trình
- **kill - 9 <ID>** : kill tiến trình

▼ QUẢN TRỊ TRONG LINUX

▼ Người dùng và nhóm

- Tạo user mới
    - !NOTE: Khi tạo user mới, linux tự động tạo group trùng tên với user đó
    - **useradd**: Tạo user mới
    - **adduser**: Tạo user đồng thời tạo thêm 1 số thông tin phụ (thư mục, các thông tin user
- **deluser** : xóa user
- Group
    - **groupadd** _tenGr_ : tạo group
    - **delgroup** -tenGr-: xóa group
    - g
        - -a : user join thêm vào group mới- dữ nguyên các group cũ
    - **deluser -tenUser- -tenGr-**: xóa user khỏi group
    - **groups -tenUser-** : liệt kê các group của user
    - **cat /etc/group**: Liệt kê các group

▼ Quyền sở hữu



- d : directiory (thư mục)
- - : file
- **chwon -user-:group -tenThuMuc-**: set quyền sở hữu cho user - group cho thư mục / file
- **chown -R** : Set lên toàn bộ thư mục/ file con của nó

▼ Quyền tác động

- chmod -quyền- -file/thưMuc-" đặt quyền cho file/ thư mục
  - ugo : user, group, other

```
root@lab-server:~# chmod g=rwx datas/
root@lab-server:~# ls -l
total 8
drwxrwxr-x 2 root devops2 4096 Jan  7 12:00 datas
drwx------ 3 root root    4096 Jan  4 12:28 snap
root@lab-server:~# chmod u=rwx,g=rx,o=- datas/
root@lab-server:~# ls -l
total 8
drwxr-x--- 2 root devops2 4096 Jan  7 12:00 datas
drwx------ 3 root root    4096 Jan  4 12:28 snap
```

  - Để có thể tác động lên 1 file trong linux, thì nhóm người dùng tối thiểu phải có quyền thực thi
- Phân quyền bằng số
  - r =4, w = 2, x = 1

```
root@lab-server:~# r=4,w=2,x=1 ⇒ 4+2+1=7^C
root@lab-server:~# chmod 750 datas/
root@lab-server:~# ls -l
total 8
drwxr-x--- 2 root devops2 4096 Jan  7 12:00 datas
drwx------ 3 root root    4096 Jan  4 12:28 snap
root@lab-server:~# x=1,w=2
```

▼ **TƯ DUY TRIỂN KHAI DỰ ÁN**

- Devops không quan trọng code, không quan trọng setup infrac, DEVOPS BIẾT TRIỂN KHAI CÀNG NHIỀU DỰ ÁN CÀNG TỐT

▼ Tư duy để triển khai dự án

- Chú ý:
  - Nên tạo mỗi dự án một user riêng, Không sử dụng user root để triển khai mọi dự án ⇒ Đảm báo tính security

▼ Cần nhớ

1. Dự án nào thì có cộng cụ tương ứng

   !LƯU Ý: Version của công cụ phải ≥ version dự án yêu cầu

2. Chỉ cần quan tâm tới file cấu hình trong dự án
3. Triển khai cần đúng 2 bước, 1 là Build, 2 là Run
4. Mỗi dự án sẽ có 1 thư mục và user riêng - cùng tên với dự án

▼ **Thực hành triển khai dự án**

1. Copy file zip dự án vào server

```
PS D:\projects> scp todolist.zip manhnv@192.168.1.110:/home/manhnv
manhnv@192.168.1.110's password:
todolist.zip
```

2. Cài đặt unzip, Giải nén
3. Tạo thư mục dự án, user dự án, đặt quyền

```
root@lab-server:/home/manhnv# mkdir /projects
root@lab-server:/home/manhnv# mv todolist /projects/
root@lab-server:/home/manhnv# ls -l /projects/
total 4
drwxr-xr-x 5 root root 4096 Jan  7 21:53 todolist
root@lab-server:/home/manhnv# adduser todolist
Adding user `todolist' ...
Adding new group `todolist' (1001) ...
Adding new user `todolist' (1001) with group `todolist' ...
Creating home directory `/home/todolist' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for todolist
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
root@lab-server:/home/manhnv# chown -R todolist:todolist
```

```
root@lab-server:/home/manhnv# chmod -R 750 /projects/todolist/
```

4. Research để cài đặt các công cụ dự án

   ⇒ Lên mạng tìm kiếm cách triển khai dự án "tên công nghệ", và làm theo để xác định các công cụ cần cài

5. Build dự án

   Quan trọng là file cấu hình của dự án là gì, từ đó config,build , v.v

   - **Mỗi dự án khi build sẽ ra 1 file/thư mục, nhiệm vụ là chạy được đống đó**

   ▼ Frontend

   - Package.json, vueconfig.js, v.v

   ▼ Triển khai với nginx

   1. **Cài npm, yarn**
   2. **Yarn install, yarn dev để kiểm tra ở local**
   3. Yarn build để build
   4. **Cài đặt nginx**

      4.1 apt install nginx

      4.2 một số file trong nginx

      - file config của nginx, có thể dùng để config port

```
root@lab-server:/etc/nginx# vi sites-available/default
```

      - file config khác

```
root@lab-server:/etc/nginx# vi /etc/nginx/conf.d/todolist.conf
```

      4.3 Cấu hình

      - Cấu hình file conf ui

```
server {
  listen 8081;
  root /projects/ui/dist/;
  index index.html;
  try_files $uri $uri/ /index.html;
}
```

- Cấu hình file /etc/nginx/sites-vailables/default cho mở ở port 9000

```
#
server {
        listen 9000 default_server;
        listen [::]:9000 default_server;
```

- **Thêm user ui vào group www-data (user của nginx, fix case lỗi 500)**

▼ Triển khai với service systemd

▼ Triển khai với pm2

▼ Backend

▼ JAVA SPRINGBOOT

-

6. Chạy dự án
   - Dùng nginx

▼ VMWARE

▼ Phân biệt Bridge và Nat

- Bridge: các thiết bị cùng mạng có (điện thoại, lap top, v.v) có thể kết nối với máy ảo
- Nat: Chỉ có máy chỉnh mới có thể kết nối với máy ảo

! NOTE:

- PC: Dùng Bridge
- Laptop: Dùng NAT

⇒ Vì Laptop di chuyển nhiều ⇒ gateway thay đổi, có thể máy ảo sẽ k có internet

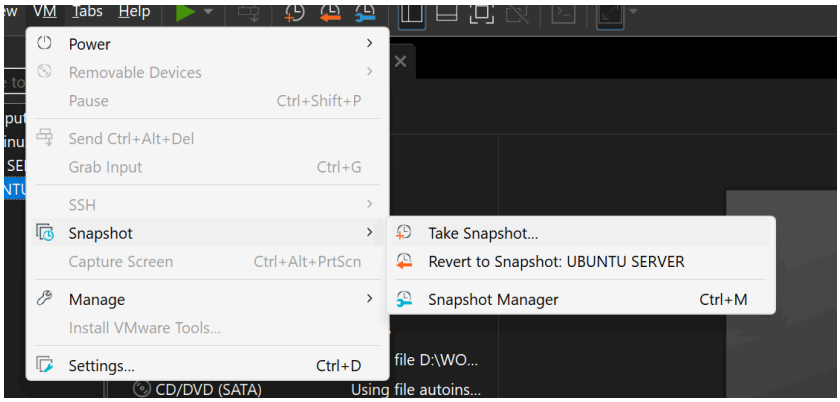
▼ Cấu hình IP tĩnh cho server (mạng NAT)

```
network:
  ethernets:
    ens33:
      dhcp4: no
      addresses: [192.168.213.99/24]
      gateway4: 192.168.213.2
      nameservers:
          addresses: [8.8.8.8, 8.8.4.4]
  version: 2
~
```

- **netplan apply**: apply cấu hình

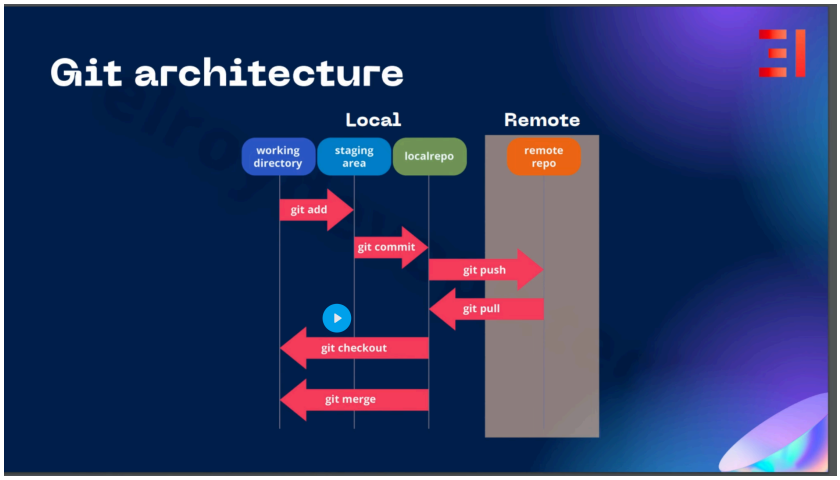
▼ Snapshot

- Dùng để Rollback máy ảo, và tạo ra máy ảo base ⇒ để lần sau tạo các máy ảo khác 1 cách nhanh chóng



▼ GIT

- Git architecture



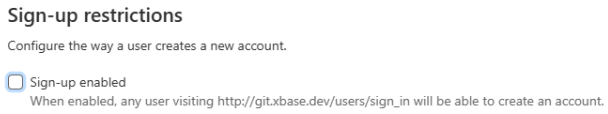
▼ Cài đặt gitlab

1. Dựng 1 gitlab server trên ubuntu
2. Tải gitlab server
   - https://packages.gitlab.com/gitlab/gitlab-ce?page=250
3. Add host
   - Add host ở gitlab server
     - DNS phân giải domain về ip gitlab server

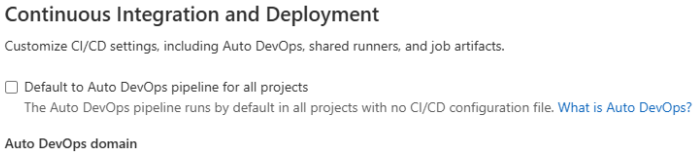
```
root@gitlabserver:/home/zues# vi /etc/hosts
root@gitlabserver:/home/zues# vi /etc/gitlab/gitlab.rb
```

     - **gitlab-ctl reconfigure:** Start your Gitlab instance
   - Add host ở máy window phân giải domain về ip của gitlab server
   - Credential
     - username : root
     - password init: **vi /etc/gitlab/initial_root_passwordv**
4. Confiig

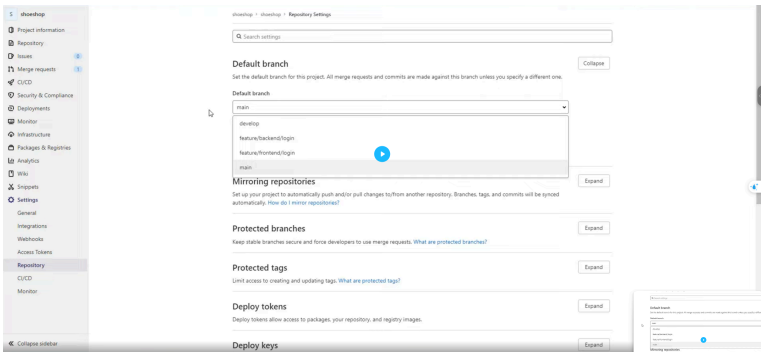
- Disable tự tạo mới user



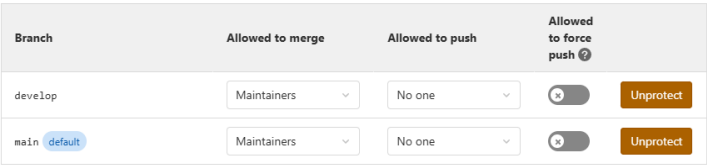
- Disable auto devops



- Đổi MK admin
- Set up default branch



- Setup cho dev không có quyền merge code trực tiếp vào nhánh main, dev, cũng như k cho push code trực tiếp lên 2 nhánh này

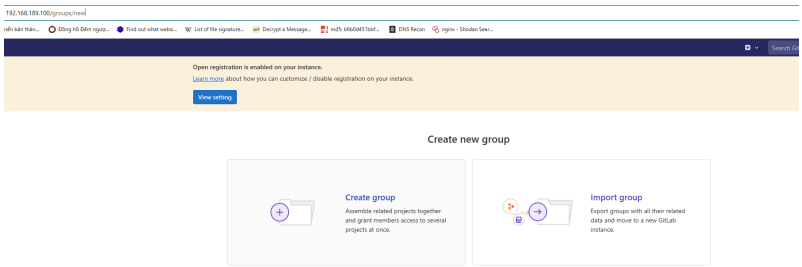


5. Tổng quan
    - Admin ：Để quản lý user, group

▼ Git thông dụng

- !NOTE: với mỗi dự án, sẽ có nhiều project (frontend cũng nhiều prj backend (microservice))

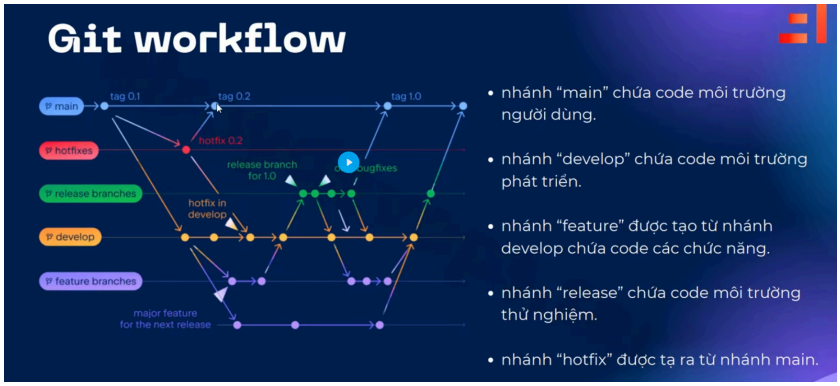
  ⇒ Vậy nên với mỗi dự án, sẽ tạo 1 group để chứa
- Tạo group:



- Tạo prj
- Đẩy prj lên gitlab

- **TUÂN THỦ GIT CONVENTION**: https://viblo.asia/p/lam-the-nao-de-viet-conventional-commits-cho-de-su-dung-07LKXbb2lV4

▼ GIT WORKFLOW



- tag: dùng để đánh dấu 1 phiên bản được release của dự án

▼ CI/CD

⇒ Giải quyết bài toán tự động hóa quy trình setup, triển khai dự án

▼ Các bước tiến hành

1. Cài đặt các công cụ tự động hóa
   - gitllab-runner: https://www.linuxtechi.com/how-to-install-gitlab-runner-on-ubuntu/
     - Link 2: https://technoupdate.medium.com/how-to-configure-gitlab-runner-on-ubuntu-20-04-c26e2f16fd24

```
root@ubuntuserver:/home/zues# gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=4072 revision=a1fca00e version=17.8.1
Running in system-mode.

Created missing unique system ID                    system_id=s_34f3cc3783bb
Enter the GitLab instance URL (for example, https://gitlab.com/):
http://192.168.189.100/
Enter the registration token:
GR1348941xP9vU-xAtoUDzk2Zh79c
Enter a description for the runner:
[ubuntuserver]: lab-server
```

   - file /etc/gitlab-runner/config.toml ⇒ file config

```
concurrent = 4
check_interval = 0
connection_max_age = "15m0s"
shutdown_timeout = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "lab-server"
  url = "http://192.168.189.100/"
  id = 1
  token = "LCEB78v28TzWSBJnad8y"
  token_obtained_at = 2025-01-18T03:07:31Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "shell"
  [runners.custom_build_dir]
  [runners.cache]
    MaxUploadedArchiveSize = 0
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
~
```

   - concurent : số project mà runner này có thể chạy cùng lúc
   - Lệnh dùng để start service runner

```
nohup gitlab-runner run --working-directory /home/gitlab-runner/ --config /etc/gitlab-runner/config.toml --service gitlab-runner --user gitlab-runner 2>&1 &
```

▼ **nohup 2>&1 &**

1. `nohup`:
   - **Mục đích**: Chạy một tiến trình mà không bị dừng (kết thúc) khi bạn thoát khỏi terminal (ngắt kết nối).
   - **Hoạt động**:
     - Thông thường, khi bạn đóng terminal, tín hiệu `SIGHUP` (hangup) sẽ được gửi đến tất cả các tiến trình con, khiến chúng dừng lại.
     - `nohup` ngăn tín hiệu `SIGHUP` này để tiến trình tiếp tục chạy.

- **Mặc định đầu ra**: Nếu bạn không chỉ định file đầu ra, `nohup` sẽ ghi kết quả của lệnh vào file `nohup.out` trong thư mục hiện tại.

---

2. `2>&1`:

- **Mục đích**: Kết hợp đầu ra lỗi (stderr) và đầu ra thường (stdout) thành một luồng duy nhất.
- **Hoạt động**:
  - `2` đại diện cho `stderr` (dòng lỗi).
  - `1` đại diện cho `stdout` (dòng kết quả).
  - `2>&1` có nghĩa là tất cả các lỗi (stderr) sẽ được chuyển hướng đến nơi mà đầu ra (stdout) đang gửi tới.
- Kết quả là cả lỗi và kết quả đều được ghi chung vào file hoặc hiển thị chung trên màn hình.

---

3. `&`:

- **Mục đích**: Chạy tiến trình ở chế độ nền (background).
- **Hoạt động**:
  - Khi thêm `&` vào cuối lệnh, shell sẽ khởi chạy tiến trình này dưới dạng nền.
  - Điều này cho phép bạn tiếp tục sử dụng terminal mà không bị chặn bởi tiến trình đang chạy.

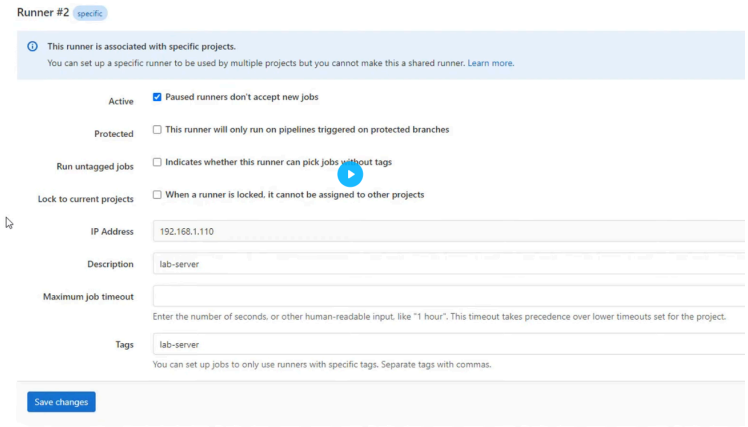
▼

▼ ps -ef

Cấu trúc của lệnh:

- `ps`: Là viết tắt của "Process Status", được dùng để hiển thị thông tin về các tiến trình đang chạy.
- `e`: Hiển thị tất cả các tiến trình trên hệ thống (bao gồm cả các tiến trình không phải của người dùng hiện tại).
- `f`: Hiển thị thông tin đầy đủ (full format) về các tiến trình, bao gồm nhiều chi tiết hơn so với định dạng mặc định.

▼ Setting runner



1. Active

- **Mô tả**:
  - Cho phép hoặc tạm dừng runner.

- **Chi tiết**:
  - Nếu checkbox này được tích, runner sẽ được kích hoạt và có thể nhận job.
  - Nếu không tích, runner sẽ bị "paused" và không nhận thêm job mới.

---

2. Paused runners don’t accept new jobs

- **Mô tả**:
  - Khi một runner bị "paused", nó sẽ không nhận thêm job mới nào từ GitLab.
- **Ứng dụng**:
  - Dùng để tạm dừng runner trong trường hợp bảo trì hoặc không muốn nhận job mới.

---

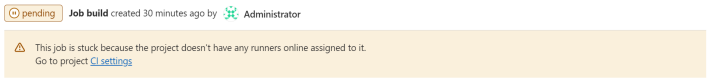
3. Protected

- **Mô tả**:
  - Khi tùy chọn này được bật, runner chỉ chạy job từ các branch hoặc tag được đánh dấu là "protected".
- **Ứng dụng**:
  - Hữu ích để bảo vệ các pipeline nhạy cảm trên branch chính (như `main` hoặc `master`).

---

4. Run untagged jobs

- **Mô tả**:
  - Cho phép runner nhận và chạy các job không có tag.
- **Chi tiết**:
  - Nếu checkbox này không được tích, runner chỉ chạy job với tag cụ thể khớp với cấu hình của nó.
  - Thực tế sẽ không chọn, mà sẽ gắn tag
- Ví dụ: khi không check vào tich này, nếu không chỉ định runner thông qua tag thì sẽ báo lỗi stuck



---

5. Lock to current projects

- **Mô tả**:
  - Khi tùy chọn này được bật, runner chỉ được gắn với các project hiện tại.
- **Chi tiết**:
  - Runner không thể được gán cho các project khác ngoài project hiện tại.

---

6. IP Address

- **Mô tả**:
  - Địa chỉ IP của server nơi runner đang hoạt động.
- **Chi tiết**:
  - Thông tin này giúp nhận biết máy chủ nào đang vận hành runner.

---

7. Description

- **Mô tả**:
  - Mô tả ngắn gọn về runner.
- **Chi tiết**:
  - Đây là một trường tùy chọn để giúp phân biệt các runner khác nhau, đặc biệt khi có nhiều runner được cấu hình.

---

8. Maximum job timeout

- **Mô tả**:
  - Đặt thời gian tối đa mà runner có thể chạy một job trước khi bị hủy (timeout).
- **Chi tiết**:
  - Có thể nhập thời gian bằng giây hoặc dạng dễ đọc (như "1 hour").
- **Ứng dụng**:
  - Hữu ích để ngăn chặn các job chạy quá lâu và tiêu tốn tài nguyên.

---

9. Tags

- **Mô tả**:
  - Danh sách các tag mà runner hỗ trợ, cách nhau bởi dấu phẩy.
- **Chi tiết**:
  - Các job trong GitLab pipeline sẽ được gửi đến runner khớp với tag được chỉ định.
- **Ứng dụng**:
  - Dùng để gán runner cho các công việc cụ thể, như `build`, `test`, `deploy`, v.v.

2. Viết file cấu hình công việc

▼ Các đặc điểm

- Pineline sẽ được chạy mỗi khi prj có 1 file thay đổi
- Tên của job đặt trùng tên của stage
- Github runner sẽ mặc định kéo code trên server về, và đứng ở tại thư mục đó
- Tuy nhiên: mỗi job, runner sẽ thực hiện xóa code cũ đi và thực hiện kéo code mới
- Để đáp ứng đúng tính chất mỗi project sẽ được chạy bởi 1 user ⇒ ta cấu hình cho gitlab-runner có thể sử dụng lệnh sudo

  ▼ Thực hiện

  - MỞ visudo

```
gitlab-runner ALL=(ALL) NOPASSWD: /bin/cp*
gitlab-runner ALL=(ALL) NOPASSWD: /bin/chown*
gitlab-runner ALL=(ALL) NOPASSWD: /bin/su shoeshop*
```

▼ gitlab-ci.yml convention

1. Cấu trúc cơ bản

Tổ chức file theo thứ tự logic và giữ mọi thứ đơn giản, rõ ràng:

```
# 1. Danh sách các stages (giai đoạn của pipeline)
stages:
  - build
```

```
  - test
  - deploy

# 2. Định nghĩa các job theo stage
build:
  stage: build
  script:
    - echo "Building the project"
  artifacts:
    paths:
      - build/

test:
  stage: test
  script:
    - echo "Running tests"
  dependencies:
    - build

deploy:
  stage: deploy
  script:
    - echo "Deploying the application"
  only:
    - main
```

2. Quy tắc chung

1. **Tối giản và ngắn gọn:**
   - Sử dụng tên job ngắn và mô tả chính xác.
   - Không lặp lại thông tin không cần thiết.
   - Tách logic phức tạp ra file script riêng (nếu cần).
2. **Đặt tên stage rõ ràng:**
   - Sử dụng tên giai đoạn như `build`, `test`, `deploy` hoặc cụ thể hơn nếu cần (`lint`, `integration-tests`).
3. **Tách biệt job theo giai đoạn:**
   - Mỗi job chỉ làm một nhiệm vụ rõ ràng.
4. **Sử dụng biến môi trường:**
   - Thay vì viết giá trị cố định, hãy sử dụng biến để dễ cấu hình và bảo mật:

     ```
     variables:
       ENVIRONMENT: production
     ```
5. **Định nghĩa artifacts:**
   - Lưu trữ kết quả (artifacts) để dùng cho các job tiếp theo:

```
artifacts:
  paths:
    - build/
```

6. **Xử lý lỗi an toàn:**
   - Luôn kiểm tra lỗi trong các lệnh:

```
script:
  - some-command || echo "Command failed"
```

---

3. Quy tắc viết script

1. **Không viết quá nhiều lệnh trong `script`:**
   - Tách các lệnh dài ra file shell hoặc script:

```
script:
  - ./scripts/build.sh
```

2. **Luôn log trạng thái:**
   - Ghi rõ các bước trong log để dễ dàng debug:

```
script:
  - echo "Step 1: Initializing"
  - echo "Step 2: Building..."
```

3. **Sử dụng các công cụ kiểm tra mã tự động:**
   - Lint và kiểm tra trước khi chạy các bước khác:

```
lint:
  stage: test
  script:
    - npm run lint
```

---

4. Quản lý trigger và condition

1. **Chạy job chỉ trên một nhánh nhất định:**

```
only:
```

```
  - main
```

2. **Loại trừ job trên một nhánh hoặc tag cụ thể:**

```
except:
  - tags
  - feature/*
```

3. **Sử dụng điều kiện để kiểm soát job:**

```
rules:
  - if: '$CI_COMMIT_BRANCH == "main"'
```

---

5. Tái sử dụng cấu hình với anchors

Giảm lặp lại bằng cách sử dụng anchors và aliases trong YAML:

```
.default-job: &default-job
  tags:
    - docker
  retry: 2
  timeout: 15m

build:
  <<: *default-job
  stage: build
  script:
    - echo "Building project"

test:
  <<: *default-job
  stage: test
  script:
    - echo "Running tests"
```

---

6. Sử dụng caching

Dùng caching để tăng tốc độ build và test:

```
cache:
  paths:
    - node_modules/
```

```
  - .cache/
```

7. Đặt thời gian timeout cho job

Tránh job chạy mãi không dừng bằng cách thiết lập thời gian tối đa:

```
timeout: 20m
```

8. Sử dụng include để chia nhỏ file

Khi file `gitlab-ci.yml` lớn, chia thành nhiều file nhỏ hơn:

```
include:
  - local: .gitlab-ci/test.yml
  - remote: 'https://example.com/deploy.yml'
```

9. Quản lý bảo mật

1. **Ẩn các thông tin nhạy cảm:**
   - Sử dụng `CI/CD Variables` để lưu trữ thông tin như API keys, mật khẩu.
2. **Không lưu mật khẩu trực tiếp trong file:**

   ```
   script:
     - echo $SECRET_KEY
   ```

10. Kiểm tra file trước khi commit

Trước khi đẩy file `gitlab-ci.yml`, kiểm tra tính hợp lệ:

```
gitlab-ci-lint
```

▼ Viết file .gitlab-ci.yml ở cùng cấp cao nhất của prj

```yaml
stages:
  - build
  - deploy
  - checklog

build:
  stage: build
  script:
    - whoami
    - pwd
    - ls
  tags:
    - ubuntu-server
```

- Giải thích
  - Chạy lệnh whoami, pwd, ls, chỉ định runner: ubuntu-server

```yaml
stages:
    - build
    - deploy
    - checklog

build:
    stage: build
    variables:
        GIT_STRATEGY: clone
    script:
        - mvn install -DskipTests=true
    tags:
        - lab-server

deploy:
    stage: deploy
    variables:
        GIT_STRATEGY: none
    script:
        - java -jar target/shoe-ShopingCart-0.0.1-SNAPSHOT.
    tags:
```

! NOTE:

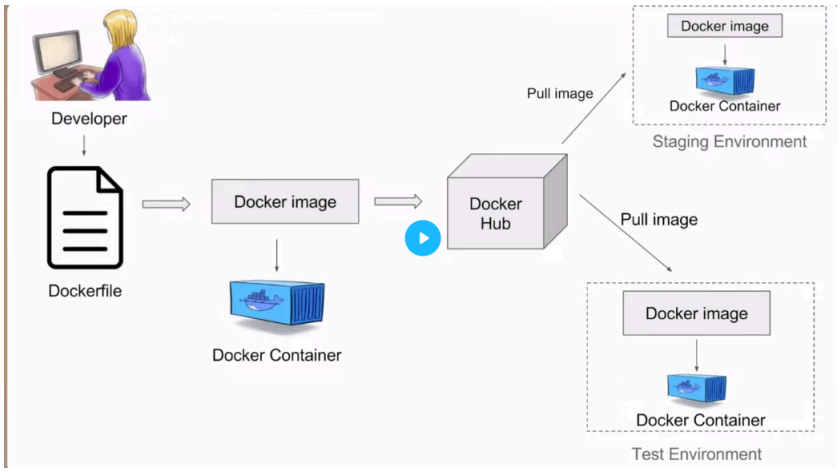
- GIT_STRATEGY: clone ⇒ tiến hành clone khi chạy
- GIT_STRATEGY: none ⇒ Không tiến hành clone khi chạy

▼ CD (Continous Delivery - deploy thủ công)

- SET up để chạy manual
- Phân quyền tác động user được phép run

▼ DOCKER

- Docker architecture



▼ **DOCKER LÀ GÌ?**

- Docker là một open platform, cung cấp cho người sử dụng công cụ và service để có thể **đóng gói ứng dụng** của mình , giúp chạy trên các môi trường khác nhau một cách đơn giản nhất

  ⇒ Nôm na là docker sẽ lo hết vụ build environment, khi join dự án mới, dev không cần phải set up môi trường lại rườm rà như: Cài nodejs, mysql, mongodb, v.v, mà chỉ cần chạy docker là đủ

- **GỒM 2 thành phần chính**
  - **Docker Engine**: Dùng để tạo Docker Image và chạy Docker Container
  - **Docker Hub**: Lưu trữ các Docker Image
- **CÁC KHÁI NIỆM QUAN TRỌNG KHÁC:**
  - **Docker Machine**: Tạo ra các Docker Engine trên máy chủ
  - **Docker Compose**: Chạy ứng dụng thông qua việc định nghĩa cấu hình Docker Container bằng tệp cấu hình, Giúp quản lí multil-container- khởi động nhiều container 1 lúc
    - **Cấu hình bằng file docker-compose.yml**
  - **Docker Image:** Một dạng tập hợp các tệp ứng dụng, tạo bởi Docker Engine
  - **Docker Container**: Dạng Runtime của Docker Image, dùng để tạo môi trường chạy ứng dụng

▼ **DOCKER IMAGE**

Docker Image là gì? (kungfutech.edu.vn)

▼ **DOCKER CONTAINER**

Docker Containers là gì? (kungfutech.edu.vn)

▼ **DOCKER COMPOSE**

Docker Compose là gì? (kungfutech.edu.vn)

▼ DOCKER COMMANDS

Sử Dụng `docker exec`

1. **Xác Định Container Đang Chạy**

   Đầu tiên, bạn cần biết tên hoặc ID của container đang chạy. Sử dụng lệnh sau để liệt kê tất cả các container đang chạy:

```
docker ps
```

Lệnh này sẽ hiển thị danh sách các container cùng với tên hoặc ID của chúng.

2. **Chạy Lệnh Trong Container**

   Sau khi bạn đã xác định tên hoặc ID của container, bạn có thể sử dụng lệnh `docker exec` để chạy lệnh trong container đó. Ví dụ:

   ```
   docker exec -it <container_name_or_id> <command>
   ```

   Thay thế `<container_name_or_id>` với tên hoặc ID của container và `<command>` với lệnh bạn muốn thực thi.

   Ví dụ, để cài đặt một gói mới hoặc chạy một lệnh PHP trong container của bạn, bạn có thể sử dụng:

   ```
   docker exec -it my-php-container apt-get install -y <package_name>
   ```

   Hoặc để kiểm tra các extension PHP:

   ```
   docker exec -it my-php-container php -m | grep zip
   ```

Cập Nhật Tệp `docker-compose.yml` (Tùy Chọn)

Nếu bạn cần thêm lệnh vào quá trình khởi động container và bạn sử dụng Docker Compose, bạn có thể sử dụng tùy chọn `command` trong tệp `docker-compose.yml`. Ví dụ:

```
version: '3.8'

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:80"
    volumes:
      - .:/var/www/html
    command: ["sh", "-c", "php -S 0.0.0.0:80 -t /var/www/html"]
```

- `command` thay thế lệnh mặc định để chạy container với lệnh tùy chỉnh.

Thay Đổi Dockerfile (Nếu Cần)

Nếu các lệnh bạn muốn thêm phải được thực hiện mỗi lần container khởi động và cần được tích hợp vào Dockerfile, bạn sẽ cần phải chỉnh sửa Dockerfile và xây dựng lại hình ảnh Docker. Nhưng nếu các lệnh chỉ cần chạy tạm thời hoặc không thường xuyên, sử dụng `docker exec` là cách thuận tiện hơn.

Sử dụng `docker exec` là một cách nhanh chóng để thực hiện các thao tác cần thiết mà không cần phải xây dựng lại toàn bộ hình ảnh Docker mỗi lần thay đổi.

▼ CÀI ĐẶT VÀ SỬ DỤNG DOCKER

▼ 1. Tải và cài đặt dokcer

- Tạo file .sh với nội dung

```
#!/bin/bash

sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrin
gs/docker-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.co
m/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install -y docker-ce
sudo systemctl start docker
sudo systemctl enable docker
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(una
me -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker --version
docker-compose --version
```

- bash ./docker_install.sh

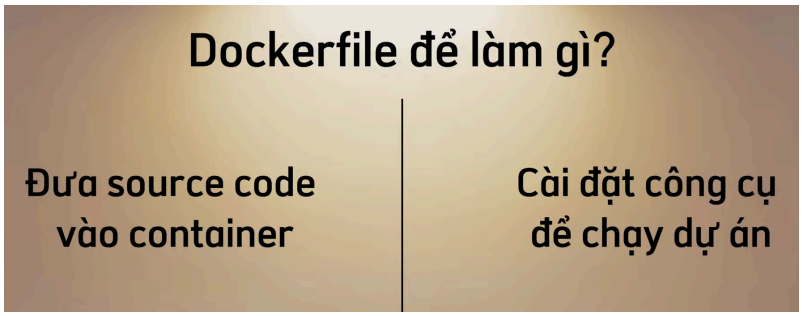
▼ 2. Cài đặt container (ubuntu)

- **docker pull ubuntu:20.04**: tải image ubuntu về docker
- **docker run —name ubuntu -it ubuntu:20.04**: run 1 container từ image ubuntu

  ⇒ Chỉ sử dụng 1 lần khi start lại container lần đầu tiên
- **docker ps -a : hiển thị các container** (kể cả container đang không chạy)
- **docker start _ID_:** run lại 1 container đã tắt
- **docker exec -it _container_ _lenhThucThi_** : truy cập vào container
- **docker run --name nginx -dp 9999:80 -it nginx:** Lệnh sẽ chạy 1 container gán tên là nginx, map cổng 9999 trên host với cổng 80 trên container, tạo dựa trên image có tên là nginx(nếu k có trong máy thì sẽ pull từ docker hub)
- **docker stop _id_** : dừng container
- **docker rm _id_: xóa container**
  - **docker rm -f _id_**: buộc xóa container
  - **docker rm -f $(docker ps -a):** xóa toàn bộ các container hiện có
- **docker rmi _tenImage+Tag_** : xóa image

▼ DOCKERIZE

▼ Docker file là gì, để làm gì



▼ Docker file commands

- FROM
- WORKDIR
- COPY
- RUN
- ENV
- EXPOSE
- CMD
- ENTRYPOINT

▼ **Tư duy để viết Dockerfile tối ưu**

- **non root user**: Không sử dụng user root để triển khai dự án, mà sử dụng user khác
- **base image**: chọn base image phù hợp
  - Đề xuất base đc build từ alpine ⇒ nhẹ gọn,
  - học cách research các docker image phù hợp(chọn base từ các nguồn offical, verifed, Sponsored_
  - Chọn image đúng version dự án
- **multipe stage:** giảm kích thước docker image cuối cùng khi được build ra

  !NOTE: image sẽ có kích thước dựa vào stage cuối cùng

▼ DOCKER REGISTER SERVER

- Là nơi lưu trữ các docker image

▼ Các cách triển khai registry server

▼ dockerhub

- Dùng dockerbub để lưu trữ

1. Đăng nhập : **docker login**
2. docker image name convention:

```
docker image = domain/project/repo:tag
```

```
docker tag xbase/ui:v1 zzzueszz/ui:v1
```

⇒ project là tên tk docker hub

3. Đẩy docker image lên repo: **docker push**

▼ Self-certified

- Tạo 1 private registry riêng biệt, tự xác thực
- ⇒ khá giống doanh nghiệp, miễn phí, tuy nhiên không thực tế

▼ 1. Cài đặt docker private registry

```
# apt get update
# apt-get install openssl
# mkdir -p /tools/registry/ && cd /tools/registry
# mkdir data certs
# openssl req -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key -subj "/CN=192.168.1.100" -addext "subjectAltName = DNS:192.168.1.100,IP:192.168.1.100" -x509 -days 365 -out certs/domain.crt
# vi docker-compose.yml
```

- file docker-compose.yml

```
version: '3'
services:
  registry:
    image: registry:2
    restart: always
    container_name: registry-server
    ports:
      - "5000:5000"
    volumes:
      - ./data:/var/lib/registry
      - ./certs:/certs
    environment:
      REGISTRY_HTTP_TLS_CERTIFICATE: ./certs/domain.crt
      REGISTRY_HTTP_TLS_KEY: ./certs/domain.key
```

- 2. docker-compose up -d
- 3. basic authen for registry server: https://learning-ocean.com/tutorials/docker/docker-docker-registry-basic-authentication/

4. fix lỗi: **tls: failed to verify certificate: x509: certificate signed by unknown authority** **(PHẢI XÁC NHÂN CHỨNG CHỈ DO CHÚNG TA TỰ KÝ)**

```
root@gitlab-server:/tools/regsitry# mkdir -p /etc/docker/certs.d/192.168.1.100:5000/
root@gitlab-server:/tools/regsitry# cp certs/domain.crt /etc/docker/certs.d/192.168.1.100:5000/ca.crt
root@gitlab-server:/tools/regsitry# systemctl restart docker
```

▼ **harbor: mua domain, mua vps, tự dựng 1 server registry riêng ⇒ sát với thực tế, áp dụng cách này**

▼ CÁC THÀNH PHẦN KHÁC CỦA DOCKER

▼ Docker volumn

- Thư mục chung liên kết giữa container và server
- ⇒ có tác dụng lưu được các dữ liệu trong docker ra bên ngoài dù có stop docker
- dùng tag -v hoặc “volumn” trong docker compose

▼ Docker network

- Tạo môi trường để các container có thể giao tiếp với nhau, container với bên ngoài cũng như giữa các cụm docker container

▼ TRIỂN KHAI DỰ ÁN LÊN EC2, CI/CD

▼ Tool để tương tác với aws

- Termius

▼ EC2

- Là dịch vụ cho thuê máy ảo trên đám mây, các vấn đề về setup, security đã được AWS lo liệu hết

▼ 1. TẠO INSTANCE

- Follow hướng dẫn tạo 1 instace ubuntu t2.micro
- Có 30gb miễn phí tổng cộng ⇒ cấp cho instance 15gb, còn lại để dành lưu các dữ liệu khác (VD: ảnh)
- **IP PUBLIC mỗi lần stop, start instance sẽ đổi 1 lần**
- SỬ DỤNG XONG THÌ TẮT INSTANCE ĐỂ TIẾT KIỆM (NÊN TERMINAL)

▼ 2. CONNECT MÁY ẢO BẰNG TERMINUS

- Add key
- Tạo host
- Kết nối tới máy ảo, username = ubuntu(default)

▼ JENKINS

- Là máy chủ opensource cung cấp plugin triển khai tự động hóa
- BLOG SETUP JENKINS: https://elroydevops.tech/jenkins-cicd-pipeline-du-an-thuc-te-sieu-chi-tiet/

▼ Cài đặt và set up reverse proxy

- Script cài đặt (chỉ hỗ trợ java 17,21)

```
#!/bin/bash

apt install openjdk-17-jdk -y
java --version
wget -p -O - https://pkg.jenkins.io/debian/jenkins.io.key | apt-key add -
sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 5BA31D57EF5975CA
apt-get update
apt install jenkins -y
systemctl start jenkins
ufw allow 8080
```

- reverse proxy = nginx để truy cập nginx thông qua port 80

  server {

  Listen 80;

  server_name jenkins|.elroydevops.tech;

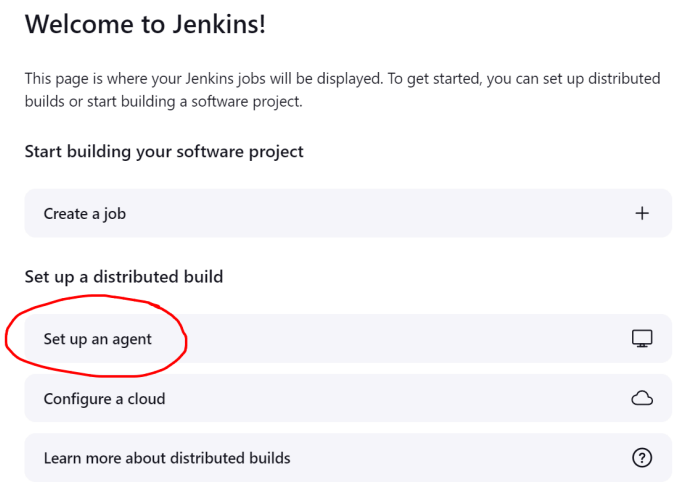
  location / {
  proxy_pass http:// jenkins.eLlroydevops.tech: 8080;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection keep-alive;
  proxy_set_header Host $host;
  proxy_cache_bypass $http_upgrade;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

```
proxy_set_header X-Forwarded-Proto $scheme;
}
```

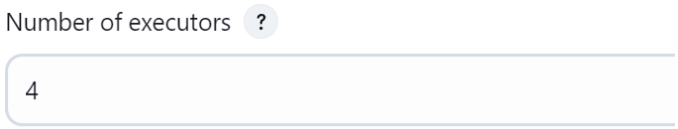
▼ CI/CD (Continous Deployment)

▼ **1. Kết nối Jenkin agent trên server cần triển khai dự án đến jenkin server**

- Thay vì ssh (kết nối ngang cấp) với server cần triển khai dự án ⇒ Rủi ro làm sập hàng loạt server
- ⇒ Sử dụng Jenkin Agent (Quản lý trạng thái server, log, securty)
- Bất cứ server nào muốn triển khai đều cần cài jenkin version
  - Java version trên server triển khai phải trùng với Java trên Jenkin server
- Set up agent



- Số pineline đc phép chạy trong 1 server



▼ 2. Kết nối jenkins server đến gitlab

▼ CI/CD (Continous Delivery)

1. Khởi động dịch vụ jenkin agent cùng server

▼ CI/CD Parameter

- https://devopsedu.vn/courses/devops-for-freshers/lesson/bai-31-jenkins-ci-cd-parameter/

▼ Ứng dụng khác của Jenkins

- Checklogs:https://devopsedu.vn/courses/devops-for-freshers/lesson/bai-32-ung-dung-khac-cua-jenkins/

▼ MONITORING

- Giúp luôn giám sát theo dõi hệ thống, nhằm chủ động trong việc bảo trì, khắc phục, nâng cấp hệ thống trước những nguy cơ chưa lường trước
- Giúp việc rà soát lỗi, khắc phục hệ thống trong thời gian ngắn nhất
- Với những hệ thống 100 nghìn, triệu user thì việc khắc phục hệ thống nhanh nhất có thể là điều hết sức quan trọng
- Giúp giám sát hệ thống 1 cách tự động
- Các Tool: Zabbix, Grafana, ELK stack

▼ UP WEBSITE LÊN HTTPS

Hướng Dẫn Cấu Hình Reverse Proxy Nginx & SSL với Certbot

1. Giới Thiệu

Reverse Proxy là một kỹ thuật giúp chuyển tiếp request từ client đến server backend (ví dụ: Spring Boot chạy trên cổng 8080).
Nginx có thể hoạt động như một reverse proxy và hỗ trợ SSL với Let's Encrypt thông qua Certbot.

---

2. Cài Đặt Nginx

Trên Ubuntu/Debian, cài đặt Nginx bằng lệnh:

```
sudo apt update
sudo apt install nginx -y
```

Kiểm tra trạng thái Nginx:

```
sudo systemctl status nginx
```

Nếu Nginx chưa chạy, khởi động lại:

```
sudo systemctl start nginx
```

---

3. Cấu Hình Reverse Proxy với Nginx

Tạo file cấu hình cho ứng dụng (ví dụ: Spring Boot chạy trên cổng 8080):

```
sudo nano /etc/nginx/sites-available/springboot
```

Thêm nội dung sau:

```
server {
    listen 80;
    server_name hocdethanhcong.io.vn;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name hocdethanhcong.io.vn;

    ssl_certificate /etc/letsencrypt/live/hocdethanhcong.io.vn/fullchain.pem;
```

```
    ssl_certificate_key /etc/letsencrypt/live/hocdethanhcong.io.vn/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Lưu lại và tạo symbolic link để kích hoạt site:

```
sudo ln -s /etc/nginx/sites-available/springboot /etc/nginx/sites-enabled/
```

Kiểm tra cấu hình và restart Nginx:

```
sudo nginx -t
sudo systemctl restart nginx
```

---

4. Cấp Chứng Chỉ SSL với Certbot (Let's Encrypt)

Cài đặt Certbot:

```
sudo apt install certbot python3-certbot-nginx -y
```

Chạy lệnh cấp SSL:

```
sudo certbot --nginx -d hocdethanhcong.io.vn --email your-email@example.com
```

Chọn `2` để tự động redirect HTTP → HTTPS.

Kiểm tra chứng chỉ SSL:

```
sudo certbot certificates
```

---

5. Cấu Hình Tự Động Gia Hạn SSL

Certbot tự động gia hạn chứng chỉ SSL, nhưng bạn có thể kiểm tra bằng tay:

```
sudo certbot renew --dry-run
```

Nếu thành công, thêm cronjob tự động gia hạn:

```
sudo crontab -e
```

Thêm dòng sau:

```
0 3 * * * certbot renew --quiet
```

---

6. Kiểm Tra Hoạt Động

6.1 Kiểm tra trạng thái Nginx:

```
sudo systemctl status nginx
```

6.2 Kiểm tra cổng 80, 443 có mở không:

```
sudo netstat -tulpn | grep -E '80|443'
```

6.3 Kiểm tra DNS tên miền:

```
dig +short hocdethanhcong.io.vn
```

Nếu kết quả không phải IP của server, cập nhật DNS.

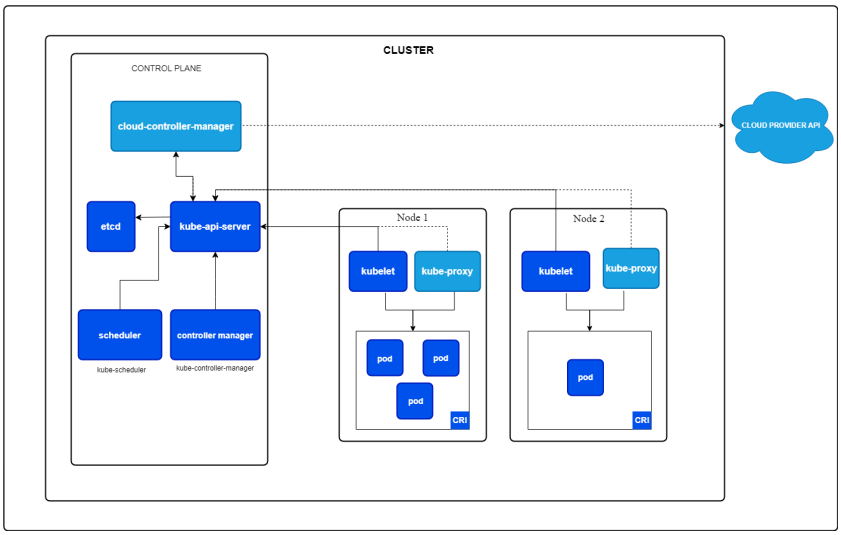
---

7. Tóm Tắt Quy Trình

1. **Cài đặt Nginx** (`sudo apt install nginx`)
2. **Cấu hình Reverse Proxy** (`proxy_pass http://127.0.0.1:8080;`)
3. **Cài đặt Certbot & cấp SSL** (`sudo certbot --nginx`)
4. **Cấu hình redirect HTTP → HTTPS** (`return 301 https://$host$request_uri;`)
5. **Tự động gia hạn SSL** (`certbot renew --dry-run` + cronjob)
6. **Kiểm tra hoạt động** (`sudo systemctl status nginx` + test trình duyệt)

Sau khi làm theo hướng dẫn, website của bạn sẽ hoạt động với HTTPS và Reverse Proxy Nginx! 🚀

- ▼ KUBERNETES
  - ▼ MỞ ĐẦU K8S
    - ▼ Kubernetes là gì, để làm gì?
      - Kubernetes dùng để quản lý cụm container, giúp tự động hồi phục, tự động cấu hình
    - ▼ Kubernetes dùng khi nào?
      - Dự án lớn, chạy đường dài
      - Có nhu cầu Scaling về tương lai
      - Dự án Microservice phức tạp
      - Dự án cần triển khai đa môi trường
      - Dự án cần khả năng tự hồi phục, chạm tới người zero-downtime (self-healing)
    - ▼ Khi nào không nên dùng
      - Dự án nhỏ, chạy đường ngắn
      - Chỉ cần triển khai môi trường trong thời gian ngắn
    - ▼ Hạ tầng Kubernetes



- CONTROL PLANE: Ban quản trị
  - **clould-controller-manager** : đóng vai trò như giám đốc trong 1 công ty, đưa ra các quyết định về cluster (chỉ có ở môi trường Cloud)
  - **kube-api-server**: Đóng vai trò như 1 thư ký, là 1 chuẩn giao tiếp giữa Control Plane và các Node
  - **etcd**: Là cơ sở dữ liệu lưu trữ dữ liệu của các Cluster
  - **scheduler**: Đóng vai trò là một giám đốc nhân sự, có tác dụng quyết định chọn Pod nào để làm việc cho Node nào
  - **controller manager**: Đóng vai trò là 1 giám sát viên, giám sát hoạt động của các Node
- Node: khu vực làm việc
  - **kubelet**: Đóng vai trò như trưởng phòng, nhận các yêu cầu từ CONTROL PLANE(BAN QUẢN TRỊ), và phân việc cho các Pod
  - **pod**: Đóng vai trò như nhân viên, làm các việc mà kubelet ủy thác
    - Pod là đơn vị nhỏ nhất
    - 1 Node có thể chứa 1 hoặc nhiều Pod
    - Pod chứa các container
  - **kube-proxy**: Đóng vai trò như nhân viên giao hàng, giúp các Node có thể giao tiếp ra bên ngoài

journalctl -u k3s --no-pager | tail -n 50

▼ CÀI ĐẶT CỤM K8S

▼ Triển khai On-premise

DOCS: https://devopsedu.vn/courses/khoa-hoc-kubenetes-thuc-te/lesson/bai-5-trien-khai-kubernetes-cluster-tren-on-premise-2/?page_tab=overview

Cấu hình tài nguyên

Hostname	OS	IP	RAM (tối thiểu)	CPU (tối thiểu)
k8s-master-1	Ubuntu 22.04	192.168.1.111	3G	2
k8s-master-2	Ubuntu 22.04	192.168.1.112	3G	2
k8s-master-3	Ubuntu 22.04	192.168.1.113	3G	2

Thực hiện trên cả 3 servers

Thêm hosts

```
# vi /etc/hosts/
```

Nội dung cấu hình

```
192.168.1.111 k8s-master-1
192.168.1.112 k8s-master-2
192.168.1.113 k8s-master-3
```

Cập nhật và nâng cấp hệ thống

```
# sudo apt update -y && sudo apt upgrade -y
```

Tạo user devops và chuyển sang user devops

```
# adduser devops
# su devops
# cd /home/devops
```

Tắt swap

▼ **SWAP LÀ GÌ? VÌ SAO PHẢI TẮT**

1. Swap là gì?Swap là một vùng bộ nhớ ảo trên ổ đĩa được sử dụng khi RAM đầy. Khi hệ thống hết RAM, Linux sẽ chuyển một phần dữ liệu từ RAM sang swap để giải phóng không gian. Tuy nhiên, vì ổ đĩa chậm hơn RAM rất nhiều, sử dụng swap có thể làm giảm hiệu suất hệ thống.

---

2. Tại sao phải tắt Swap khi cài Kubernetes (K8s)?

Kubernetes yêu cầu tắt **swap** vì các lý do sau:

🔷 1. Kubernetes Scheduler cần quản lý tài nguyên chính xác

- Kubernetes sử dụng **cgroups** để quản lý tài nguyên CPU, RAM.

- Nếu swap được bật, hệ thống có thể đẩy bớt dữ liệu từ RAM xuống ổ đĩa, làm sai lệch thông tin mà Kubernetes sử dụng để quyết định nơi chạy Pod.
- K8s cần biết chính xác lượng RAM thực tế để tránh overcommit tài nguyên.

🔷 2. Performance (Hiệu suất)

- Swap làm giảm hiệu suất của Kubernetes vì đọc/ghi từ ổ đĩa chậm hơn RAM.
- Trong môi trường container, hiệu suất là ưu tiên hàng đầu.

🔷 3. kubelet sẽ không khởi động nếu Swap bật

- Kể từ Kubernetes v1.8, nếu swap không được tắt, `kubelet` sẽ báo lỗi và không khởi động được.

Ví dụ lỗi:

```
fail: [k8s-master-1] (item=kubelet) => {"changed": false, "msg": "kubelet failed to start: Unit kubelet.servi
ce has failed
```

```
# sudo swapoff -a
# sudo sed -i '/swap.img/s/^/#/' /etc/fstab
```

Cấu hình module kernel

```
# echo -e "overlaynbr_netfilter" | sudo tee /etc/modules-load.d/containerd.conf > /dev/null
```

Tải module kernel

```
# sudo modprobe overlay
# sudo modprobe br_netfilter
```

Cấu hình hệ thống mạng

```
# echo "net.bridge.bridge-nf-call-ip6tables = 1" | sudo tee -a /etc/sysctl.d/kubernetes.conf
# echo "net.bridge.bridge-nf-call-iptables = 1" | sudo tee -a /etc/sysctl.d/kubernetes.conf
# echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.d/kubernetes.conf
```

Áp dụng cấu hình sysctl

```
# sudo sysctl --system
```

Cài đặt các gói cần thiết và thêm kho Docker

```
# sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates
# sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o /etc/apt/truste
d.gpg.d/docker.gpg
# sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -c
s) stable"
```

Cài đặt containerd

```
# sudo apt update -y
# sudo apt install -y containerd.io
```

Cấu hình containerd

```
# containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
# sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
```

Khởi động containerd

```
# sudo systemctl restart containerd
# sudo systemctl enable containerd
```

Thêm kho lưu trữ Kubernetes

```
# echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list
# curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Cài đặt các gói Kubernetes

```
# sudo apt update -y
# sudo apt install -y kubelet kubeadm kubectl
# sudo apt-mark hold kubelet kubeadm kubectl
```

Xem tiếp bài giảng để cài đặt K8s cluster phù hợp (xác định mô hình cụm cần cài đặt và reset cụm nếu cài không như mong muốn)

Khối lệnh reset cụm khi đã khởi tạo cụm

```
# sudo kubeadm reset -f
# sudo rm -rf /var/lib/etcd
# sudo rm -rf /etc/kubernetes/manifests/*
```

Mô hình đầu tiên: 1 master 2 worker

Thực hiện trên server k8s-master-1

```
# sudo kubeadm init
# mkdir -p $HOME/.kube
# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
# kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

Thực hiện trên server k8s-master-2 và k8s master-3

```
# sudo kubeadm join 192.168.1.111:6443 --token your_token --discovery-token-ca-cert-hash your_sha
```

Mô hình thứ hai: 3 master (worker) – xem video bài giảng để hiểu

Thực hiện trên server k8s-master-1

```
# sudo kubeadm init --control-plane-endpoint "192.168.1.111:6443" --upload-certs
# mkdir -p $HOME/.kube
# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
# kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

Thực hiện trên server k8s-master-2 và k8s-master-3

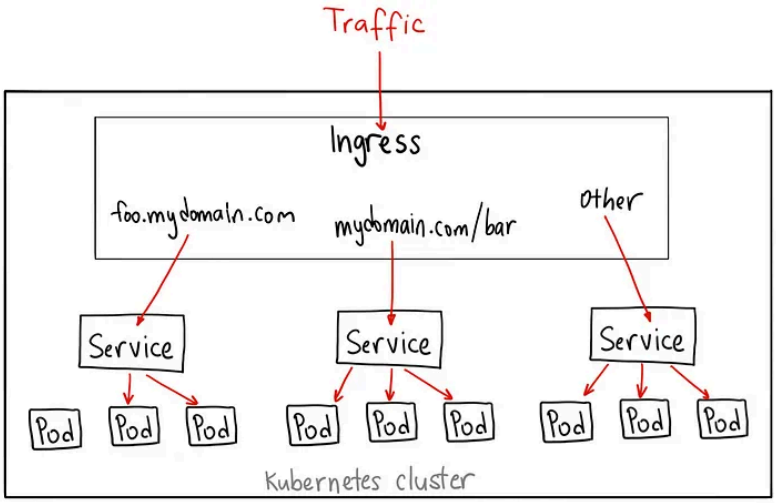
```
# sudo kubeadm join 192.168.1.111:6443 --token your_token --discovery-token-ca-cert-hash your_sha --control-plane --certificate-key your_cert
# mkdir -p $HOME/.kube
# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

▼ TRIỂN KHAI DỰ ÁN

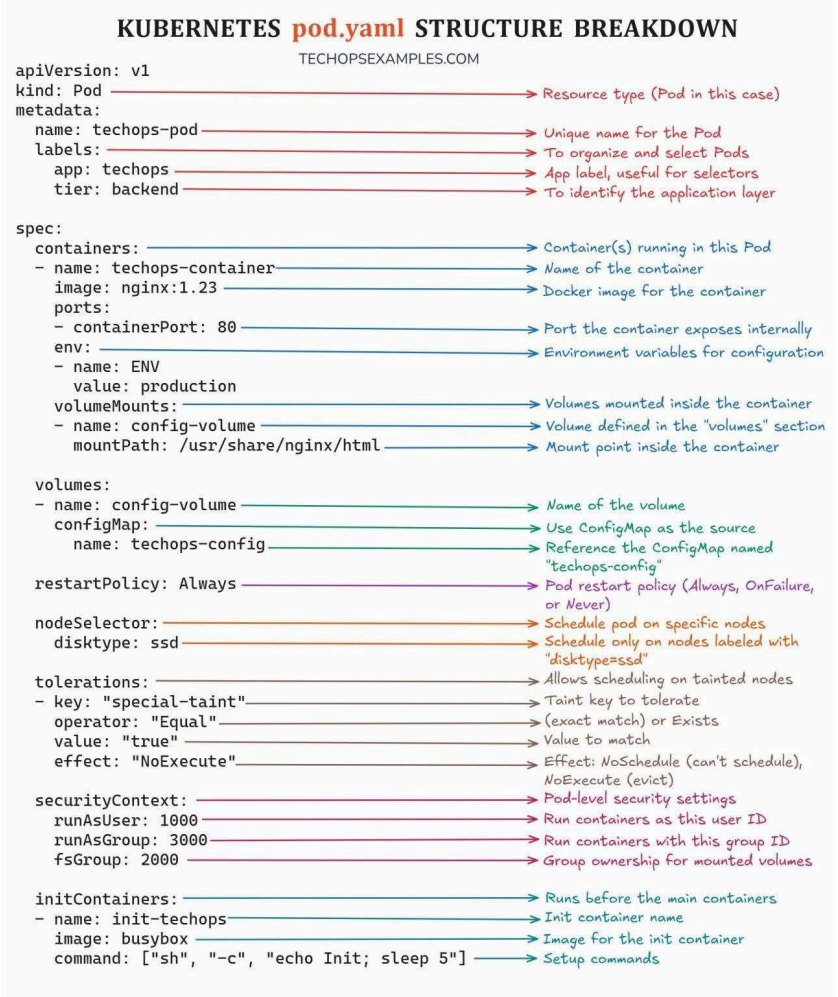
▼ Mô hình triển khai dự án

DOC: https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

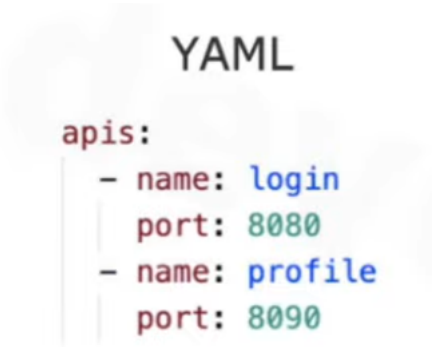
1. Cluster IP
2. NodePort
3. LoadBlancer
4. **Ingress (Cách tối ưu nhất)**



▼ File cấu hình



- Sử dụng file YAML để cấu hình



▼ Các thành phần chính

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: bee
spec:
  containers:
    - name: nginx
      image: nginx
      tolerations:
        - key: spray
          value: mortein
          effect: NoSchedule
```

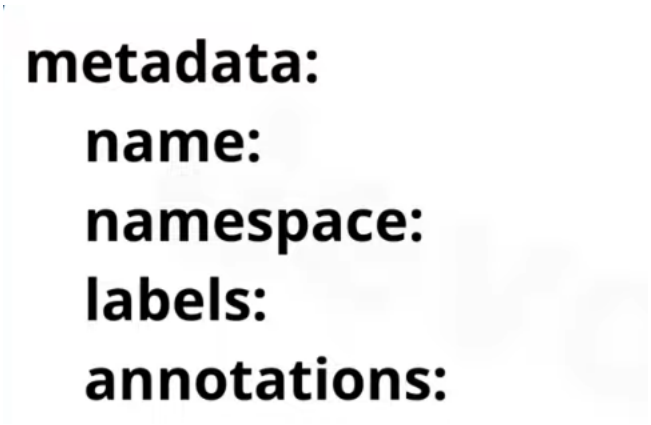
- **apiVersion**

> **apiVersion: v1**
> **apiVersion: apps/v1**
> **apiVersion: batch/v1**
> **apiVersion: networking.k8s.io/v1**
> **apiVersion: policy/v1**
> **apiVersion: rbac.authorization.k8s.io/v1**
> **apiVersion: autoscaling/v1**
> **apiVersion: storage.k8s.io/v1**

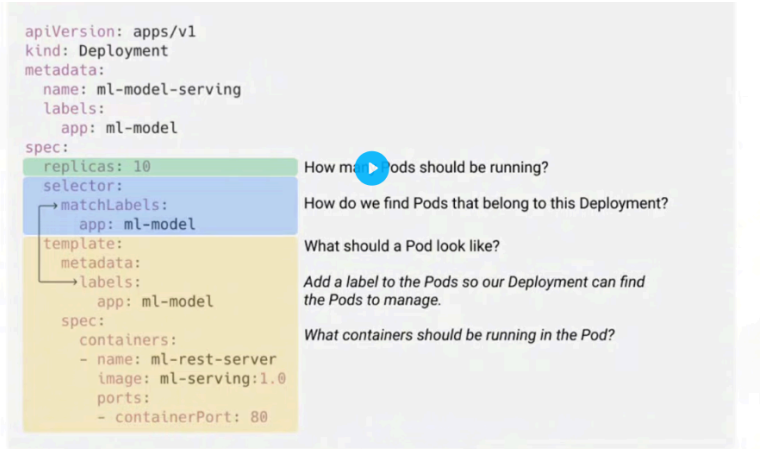
- **kind:** **sử dụng để chỉ định loại tài nguyên Kubernetes**

> **kind: Pod**
> **kind: Deployment**
> **kind: StatefulSet**
> **kind: DaemonSet**
> **kind: Service**
> **kind: Ingress**
> **kind: ConfigMap**
> **kind: Secret**
> **kind: PersistentVolume**
> **kind: PersistentVolumeClaim**

- **metadata: chứa các thông tin như tên tài nguyên, nhãn (labels) và namespace?**

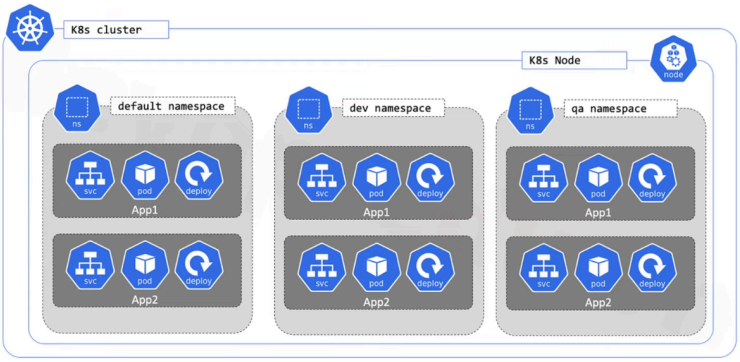


- spec: Sử dụng để mô tả cụ thể tài nguyên đó



▼ Namespace

- Là cách tổ chức và phân tách các tài nguyên trong một cụm Kubernetes để quản lý tốt hơn
- Chia nhỏ cụm thành các không gian làm việc nhỏ hơn ⇒ giúp dễ dàng quản lý và vận hành
- Ví dụ



▼ Các lệnh tương tác với Namespace

- **kubectl get pod —namespace tenNamespace** (Mặc định không có tag namepsace thì chỉ định namespace default) : liệt kê các pod trong 1 name space
- kubectl get ns : lấy danh sách các namepsace
- **kubectl create ns *tenNamespace:* *Tạo namespace mới***
- **kubectl delete ns *tenNamespace*: *Xóa namespace***

! Tuy nhiên trong thực tế, devops sẽ cấu hình lên các file yaml để tự động tạo namespace để dễ dàng cho việc triển khai, back up

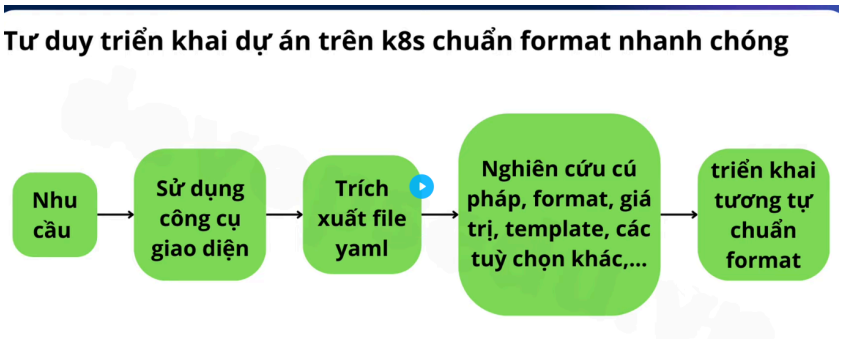
- **kubectl apply -f tenfile**: apply cấu hình file yaml

- ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: project-1
```

-

▼ Chuẩn phương pháp triển khai dư án Kubernetes hiệu quả



▼ Rancher K8s

- Rancher là gì?

Rancher là một công cụ giúp triển khai, quản lý và giám sát nhiều cụm Kubernetes trên các môi trường khác nhau, bao gồm cả On-premise và trên các nhà cung cấp dịch vụ đám mây như AWS, Azure, Google Cloud,...

- Rancher làm được gì?

- **Quản lý nhiều cụm Kuberntes.**
- **Phân quyền mạnh mẽ (based RBAC Kuberntes).**
- **Hỗ trợ giám sát cụm Kubernetes.**
- **Bảo mật tốt.**

- Có 2 cách cài đặt

  ▼ **Thông qua docker**

1. Tạo rancher-server với cấu hình như sau (thêm 1 ổ đĩa rời - sát với triển khai thực tế)

3	k8s-master-3	192.168.1.113	k8s-master-3	4GB	2	20GB	
4	rancher-server	192.168.1.114	rancher-server	2GB	1	20GB	rancher.devopseku.vn
						20GB	

2. Gắn disk vào server

```
# sudo mkfs.ext4 -m 0 /dev/sdb
# mkdir /data
# echo "/dev/sdb  /data  ext4  defaults  0  0" | sudo tee -a /etc/fstab
# mount -a
# sudo df -h
```

3. Cài đặt docker, docker-compose
4. Check Version rancher, cài Rancher bằng Docker
   - Check Version rancher tương thích với K8S không: https://www.suse.com/suse-rancher/support-matrix/all-supported-versions/rancher-v2-9-2/
   -

```
version: '3'
services:
  rancher-server:
    image: rancher/rancher:v2.9.2
    container_name: rancher-server
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /data/rancher/data:/var/lib/rancher
    privileged: true
```

5. Lấy MK init

   `# docker logs rancher-server 2>&1 | grep "Bootstrap Password:"`

6. Kết nối cụm K8S tới Rancher
   - Chọn option: **Generic**
   - Cluster Name : ghi cụ thể cho tường mình
7. Command chạy trên server k8s-master-1

   curl --insecure -sfL https://192.168.189.114/v3/import/dvm99wz7qvrpdk6p2vbvzmwvtcd6pqck6b98glzfm4nppzrdvgklbk_m-tmg4j49v.yaml | kubectl apply -f -

▼ Thông qua Kubernetes

▼ **Lỗi khi kết nối rancher với k8s**

- calico (pod network phải hoạt động, nếu không phải tải về)
- **Nếu vẫn lỗi, ktra log trên pod cluster-agent, trong case của tôi là do không đủ dung lượng để chạy container cluster-agent nên không thể kết nối tới rancher**

1. Get all Pod
2. Check pod calico existing?

3. Check log pod calico
4. describe pod cluster agent xem thông tin
    - **kubectl describe pod -tenPod- -n cattle-system**

▼ Pod

- Pod là gì?
    - Pod là đơn vị nhỏ nhất trong K8S, có thể chứa 1 container (Thường dùng) hoặc nhiều container (case khi các container phụ thuộc và cần share resource với nhau)
    - Các container trong 1 pod có thể giao tiếp với nhau bằng localhost, share 2 dạng resource với nhau
        - Networking
        - Storage

  ⇒ Pod giống như một wrapper - môi trường để chạy container, lúc này k8s không trực tiếp quản lý container mà quản lý thông qua các pod

- Cách sử dụng Pod
    - **DOC: https://viblo.asia/p/kubernetes-series-bai-2-kubernetes-pod-thanh-phan-de-chay-container-YWOZr3QElQ0**

▼ Deployment

DOC: https://viblo.asia/p/kubernetes-series-bai-5-deployment-cap-nhat-ung-dung-bang-deployment-RQqKL6q0l7z

▼ deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    workload.user.cattle.io/workloadselector: apps.deployment-car-serv-car-serv-deployment
  name: car-serv-deployment
  namespace: car-serv
spec:
  replicas: 2
  revisionHistoryLimit: 11
  selector:
    matchLabels:
      workload.user.cattle.io/workloadselector: apps.deployment-car-serv-car-serv-deployment
  template:
    metadata:
      labels:
        workload.user.cattle.io/workloadselector: apps.deployment-car-serv-car-serv-deployment
      namespace: car-serv
    spec:
      containers:
        - image: elroydevops/car-serv
          imagePullPolicy: Always
          name: car-serv
          ports:
            - containerPort: 80
              name: tcp
              protocol: TCP
```

▼ Command line

Cập nhật trực tiếp số lượng replicas

```
# kubectl scale deployment <ten-deployment> --replicas=<so-replicas>
```

Xem chi tiết cụ thể về một Deployment

```
# kubectl describe deployment -n <namespace>
```

Xem cấu hình YAML của một Deployment

```
# kubectl get deployment <ten-deployment> -o yaml
```

Cập nhật Deployment bằng cách thay đổi hình ảnh container

```
# kubectl set image deployment/<ten-deployment> <ten-container>=<ten-image>:<tag-moi>
```

Rollback Deployment về phiên bản trước

```
# kubectl rollout undo deployment <ten-deployment>
```

Kiểm tra lịch sử các phiên bản của Deployment

```
# kubectl rollout history deployment <ten-deployment>
```

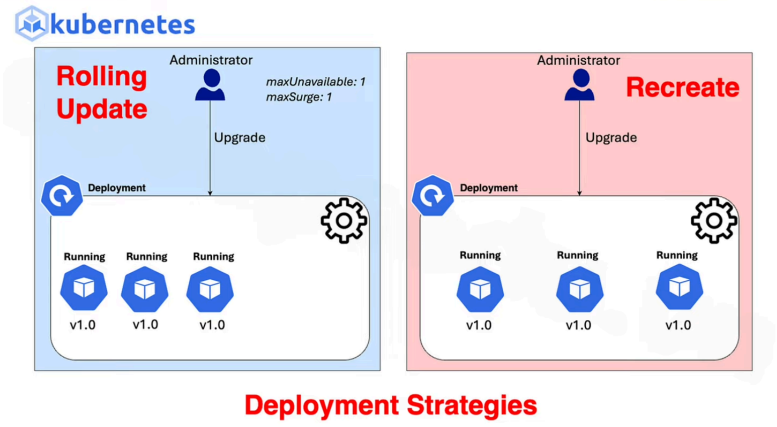
Liệt kê các Pod được tạo bởi một Deployment cụ thể

```
# kubectl get pods -l app=<ten-deployment> -n <namespace>
```

Cập nhật biến môi trường cho các container trong Deployment

```
# kubectl set env deployment/<ten-deployment> <key>=<value>
```

▼ Chiến lược triển khai Deployment



▼ Rolling Update (Cập nhập dần dần)

```
kind: Deployment
metadata:
```

```
  labels:
    app: car-serv-deployment
  name: car-serv-deployment
  namespace: car-serv
spec:
  replicas: 2
  strategy:
    rollingUpdate:
      maxSurge: 1 #25%
      maxUnavailable: 1 #25%
    type: RollingUpdate
  revisionHistoryLimit: 11
  selector:
    matchLabels:
      app: car-serv-deployment
  template:
    metadata:
      labels:
        app: car-serv-deployment
      namespace: car-serv
    spec:
      containers:
        - image: elroydevops/car-serv
          imagePullPolicy: Always
          name: car-serv
          ports:
            - containerPort: 80
              name: tcp
              protocol: TCP
```

▼ Rolling update

Khi cập nhật một `Deployment` trong Kubernetes, `maxSurge` và `maxUnavailable` quyết định cách các Pod cũ được thay thế bằng Pod mới mà không làm hệ thống gián đoạn.

🔷 1. maxSurge là gì?

🔷 **maxSurge** = Số lượng **Pod mới** có thể được tạo ra **vượt quá** số lượng `replicas`.

- Nghĩa là trong quá trình cập nhật, Kubernetes **có thể thêm bao nhiêu Pod mới** trước khi xóa bớt Pod cũ.
- Giá trị có thể là **số nguyên** hoặc **phần trăm** (%) tổng số `replicas`.

🔷 Ví dụ về maxSurge

Giả sử `replicas: 4` và `maxSurge: 1`

👉 Khi cập nhật, Kubernetes **có thể tạo thêm 1 Pod mới trước khi xóa Pod cũ**, tức là **tổng cộng có 5 Pod chạy tạm thời**.

Nếu dùng tỷ lệ phần trăm:

🔷 `maxSurge: 25%` với `replicas: 4`

→ 25% của 4 = **1 Pod mới có thể được tạo thêm**.

🔷 2. maxUnavailable là gì?

🔷 **maxUnavailable** = Số lượng **Pod cũ** có thể bị xóa ngay lập tức trong quá trình cập nhật.

- Nghĩa là Kubernetes có thể gỡ bỏ **bao nhiêu Pod** trước khi tạo Pod mới.
- Giá trị có thể là **số nguyên** hoặc **phần trăm** (%) tổng số `replicas`.

🔷 Ví dụ về maxUnavailable

Giả sử `replicas: 4` và `maxUnavailable: 2`

👉 Kubernetes **có thể tắt ngay 2 Pod cũ** trước khi tạo Pod mới.

Nếu dùng tỷ lệ phần trăm:

🔷 `maxUnavailable: 50%` với `replicas: 4`

→ 50% của 4 = **2 Pod cũ có thể bị xóa ngay lập tức**.

📌 3. maxSurge và maxUnavailable kết hợp như thế nào?

Khi cập nhật, Kubernetes sử dụng cả hai giá trị này để **tạo và xóa Pod một cách an toàn**.

🔷 Ví dụ cụ thể

```
yaml
CopyEdit
spec:
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

Cách hoạt động:

1️⃣ Kubernetes **tạo thêm 1 Pod mới**, tổng số Pod lúc này là **5** (4 + 1).

2️⃣ Sau khi Pod mới chạy ổn định, **xóa đi 1 Pod cũ**, trở lại 4 Pod.

3️⃣ Tiếp tục lặp lại quá trình trên cho đến khi tất cả Pod cũ được thay thế.

🚀 **Kết quả**:

- Luôn có ít nhất **3 Pod hoạt động** (vì `maxUnavailable: 1`, nghĩa là chỉ 1 Pod cũ bị xóa cùng lúc).
- Sẽ không có thời điểm nào mà toàn bộ dịch vụ bị gián đoạn.

Khi nào `maxSurge` và `maxUnavailable` bằng nhau?

👉 Khi bạn muốn tốc độ cập nhật ổn định, không tăng hoặc giảm số lượng Pod tổng thể.

Ví dụ:

```
yaml
CopyEdit
replicas: 4
strategy:
  type: RollingUpdate
  rollingUpdate:
```

```
    maxSurge: 1
    maxUnavailable: 1
```

Cách hoạt động:

1️⃣ **Tạo 1 Pod mới** (tổng cộng 5 Pod).

2️⃣ Sau khi Pod mới chạy ổn định, **xóa 1 Pod cũ** (trở lại 4 Pod).

3️⃣ Lặp lại quá trình này cho đến khi tất cả Pod cũ được thay thế.

📌 **Kết quả:**

- Luôn có ít nhất **3 Pod chạy**.
- Số lượng Pod không thay đổi nhiều trong quá trình cập nhật.

🎯 Khi nào `maxSurge` lớn hơn `maxUnavailable`?

👉 Khi bạn muốn **tăng độ sẵn sàng**, đảm bảo không có downtime.

Ví dụ:

```
yaml
CopyEdit
replicas: 4
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 2
    maxUnavailable: 1
```

Cách hoạt động:

1️⃣ **Tạo 2 Pod mới** (tổng cộng 6 Pod).

2️⃣ Sau khi các Pod mới chạy ổn định, **xóa 1 Pod cũ**.

3️⃣ Lặp lại quá trình này cho đến khi tất cả Pod cũ được thay thế.

📌 **Kết quả:**

- Luôn có **ít nhất 4 Pod chạy** trong suốt quá trình cập nhật.
- **Không có downtime**, nhưng sẽ tốn thêm tài nguyên.

🎯 Khi nào `maxUnavailable` lớn hơn `maxSurge`?

👉 Khi bạn muốn **cập nhật nhanh hơn** mà không cần thêm tài nguyên.

Ví dụ:

```
yaml
CopyEdit
replicas: 4
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
```

```
    maxUnavailable: 2
```

Cách hoạt động:

1️⃣ **Xóa 2 Pod cũ ngay lập tức** (chỉ còn 2 Pod chạy).

2️⃣ **Tạo 1 Pod mới** (tổng cộng 3 Pod).

3️⃣ Khi Pod mới chạy ổn định, **tiếp tục tạo thêm 1 Pod mới** (trở lại 4 Pod).

4️⃣ Lặp lại quá trình này cho đến khi hoàn tất cập nhật.

📌 **Kết quả:**

- Sẽ có lúc chỉ còn **2 Pod chạy**, có thể gây gián đoạn nhẹ.
- **Cập nhật nhanh hơn**, nhưng có rủi ro downtime.

🔥 Kết luận: Chọn giá trị nào phù hợp?

Mục tiêu	maxSurge	maxUnavailable
Cập nhật an toàn, không downtime	**Cao hơn**	Thấp
Cập nhật nhanh, chấp nhận gián đoạn	Thấp	**Cao hơn**
Giữ số lượng Pod ổn định	Bằng nhau	Bằng nhau

👉 Nếu hệ thống cần luôn hoạt động, nên chọn `maxSurge > maxUnavailable`.

👉 Nếu cần cập nhật nhanh và có thể chịu gián đoạn, chọn `maxUnavailable > maxSurge`.

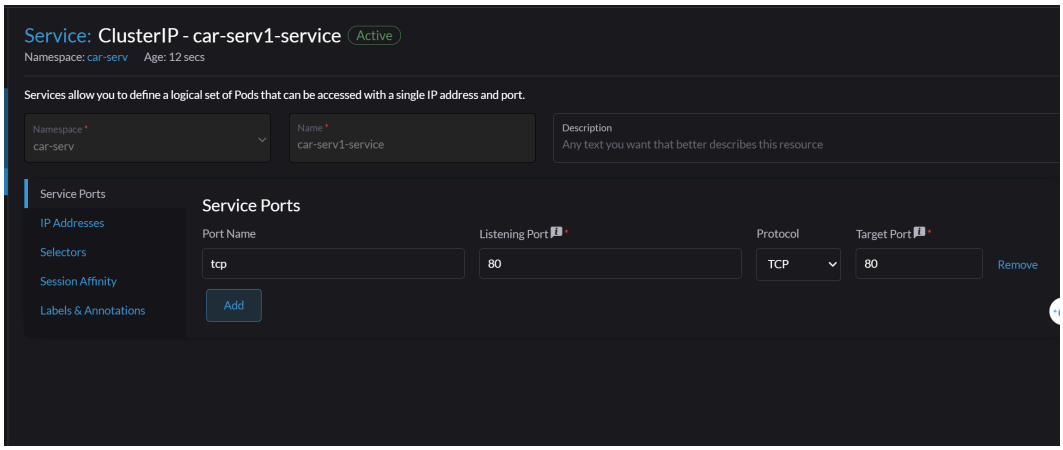
▼ Recreate (Cập nhập 1 lần- Xóa tất cả Pod cũ trước khi tạo Pod mới.)

▼ Service

▼ **ClusterIP**

NOTE: Với service này, ta chỉ có thể truy cập các port từ bên trong cụm k8s, không thể truy cập trực tiếp từ

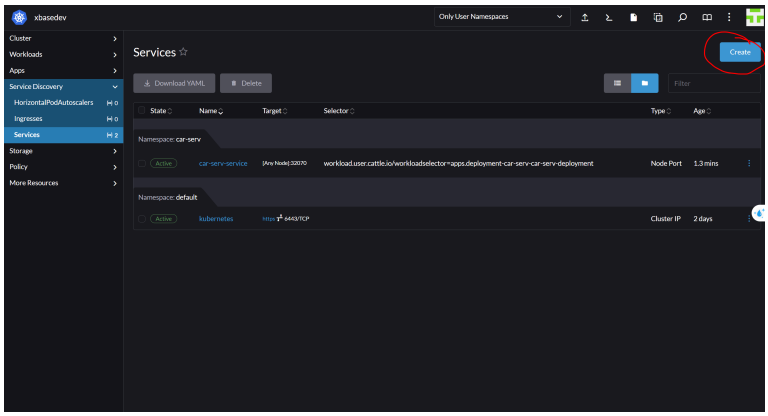
1. Tạo service cluster IP

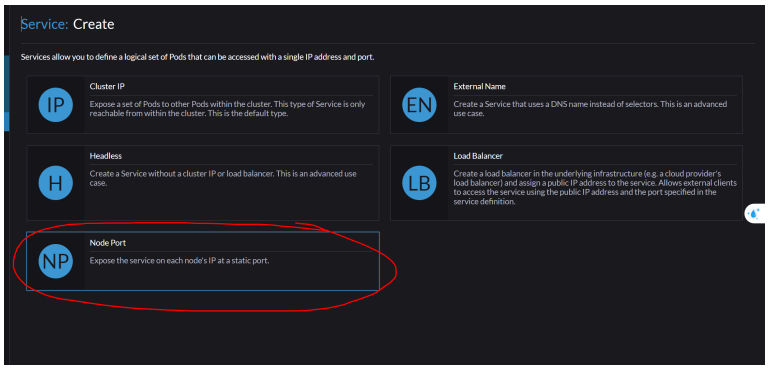


2. Chọn selector

▼ **NodePort**

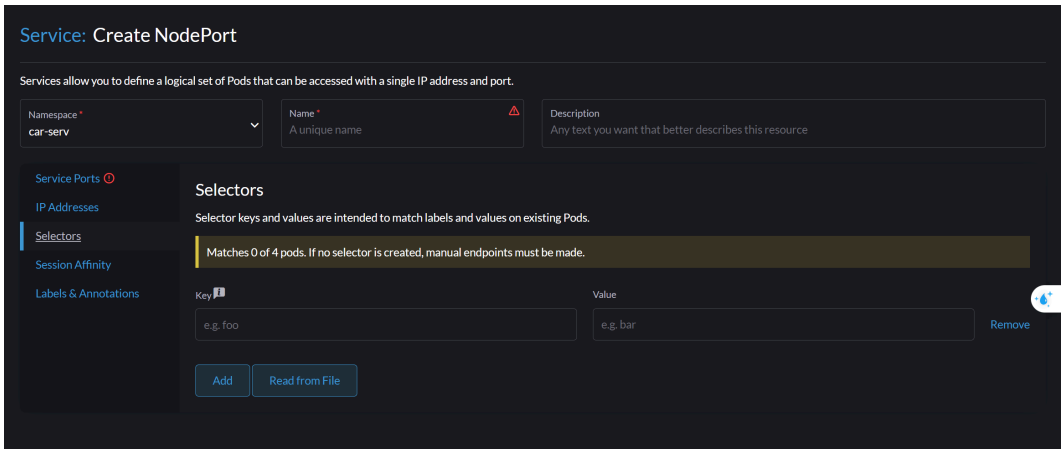
1. Tạo mới NodePort





2. điền tt

3. chọn selector (trùng với selector cấu hình trong deployment)



- ▼ LoadBalancer
- ▼ ExternalName

▼ Ingress

https://viblo.asia/p/k8s-basic-kubernetes-ingress-pgjLNb39L32

- ▼ Cài đặt ingress-nginx bằng helm

```
# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
# helm repo update
# helm pull ingress-nginx/ingress-nginx
# tar -xzf ingress-nginx-4.11.3.tgz
# vi ingress-nginx/values.yaml
>> Sửa type: LoadBalancing ⇒ type: NodePort
>> Sửa nodePort http: "" ⇒ http: "30080"
>> Sửa nodePort https: "" ⇒ https: "30443"
# kubectl create ns ingress-nginx
# helm -n ingress-nginx install ingress-nginx -f ingress-nginx/values.yaml ingress-nginx
```

- ▼ Cấu hình nginx Load Balancer

```
upstream my_servers {
    server 192.168.1.111:30080;
    server 192.168.1.112:30080;
    server 192.168.1.113:30080;
}

server {
    listen 80;

    location / {
        proxy_pass http://my_servers;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

- ▼ Cấu hình ingress dự án car-serv

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: car-serv-ingress
  namespace: car-serv
spec:
  ingressClassName: nginx
  rules:
    - host: car-serv-onpre.devopsedu.vn
      http:
        paths:
          - backend:
```

```
            service:
              name: car-serv1-service
              port:
                number: 80
          path: /
          pathType: Prefix
```

▼ Helm

- Là trình quản lý gói và công cụ cho môi trường K8s

▼ Cài đặt Helm

```
 wget https://get.helm.sh/helm-v3.16.2-linux-amd64.tar.gz
# tar xvf helm-v3.16.2-linux-amd64.tar.gz
# sudo mv linux-amd64/helm /usr/bin/
# helm version
```

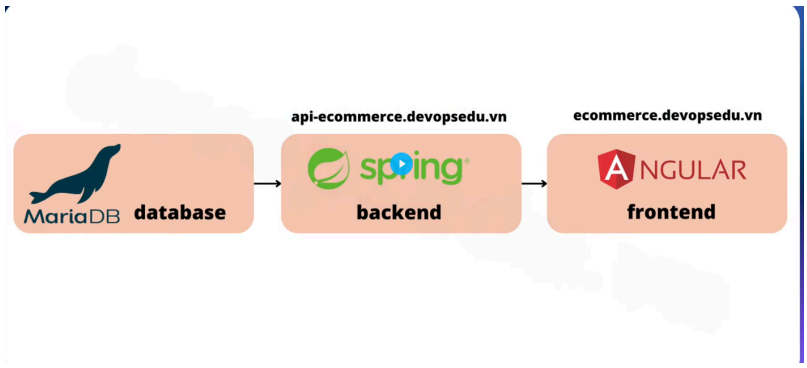
▼ Template file .yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: car-serv
  name: car-serv-deployment
  namespace: car-serv
spec:
  replicas: 2
  revisionHistoryLimit: 11
  selector:
    matchLabels:
      app: car-serv
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: car-serv
      namespace: car-serv
    spec:
      containers:
        - image: elroydevops/car-serv
          imagePullPolicy: Always
          name: car-serv
          ports:
            - containerPort: 80
              name: tcp
              protocol: TCP
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: car-serv-service
  namespace: car-serv
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: tcp
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: car-serv
  sessionAffinity: None
  type: ClusterIP
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: car-serv-ingress
  namespace: car-serv
spec:
  ingressClassName: nginx
  rules:
    - host: car-serv-onpre.devopsedu.vn
      http:
        paths:
          - backend:
              service:
                name: car-serv-service
                port:
                  number: 80
            path: /
            pathType: Prefix
```

▼ **TRIỂN KHAI DỰ ÁN FULLSTACK**

▼ 1. Mô hình dự án



▼ 2. Cài đặt Server DB, MariaDb

3. Di chuyển file vào server

▼ 4. Triển khai dự án Frontend

5. Triển khai MariaDB

6. Triển khai dự án Backend

▼ ConfigMap

- Vấn đề: Mỗi khi dev chỉnh sửa file config, v.v ... thì phải build lại docker file ⇒ tốn rất nhiều tg
  - ⇒ ConfigMap ra đời để giải quyết vấn đề này, lúc này ta chỉ cần config ở K8S, và chúng sẽ đc map vào trong container

▼ ConfigMap.yml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ecommerce-backend-application-properties-configmap
  namespace: ecommerce
data:
  application.properties: |
    spring.datasource.url=jdbc:mysql://192.168.189.115:3306/full-stack-ecommerce
    spring.datasource.username=ecommerceapp
    spring.datasource.password=StrongPa55WorD
    spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
    spring.datasource.sql-script-encoding=UTF-8

    spring.jpa.properties.hibernate.globally_quoted_i/dentifiers=true
    spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
    spring.jpa.hibernate.ddl-auto=none
    spring.jpa.show-sql=true
    spring.jpa.properties.hibernate.format_sql=true

    spring.data.rest.base-path=/api
    spring.data.rest.detection-strategy=ANNOTATED

    allowed.origins=http://ecommerce.devopsedu.vn

    okta.oauth2.client-id=0oab0IzwjoN1Rjsar5d7
    okta.oauth2.issuer=https://dev-82108115.okta.com/oauth2/default
```

▼ **Sử dụng configMap để config cho dự án SpringBoot**

▼ 1. DockerFile

build stage

FROM maven:3.8.3-openjdk-17 as build

WORKDIR ./src
COPY . .

RUN mvn install -DskipTests=true

run stage

FROM openjdk:17-alpine

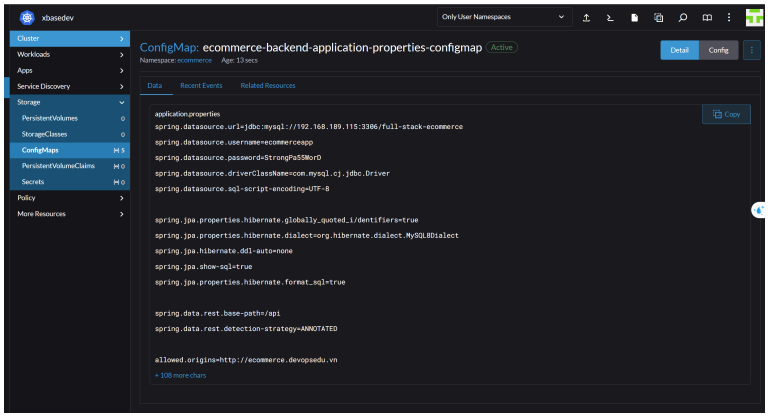
RUN unlink /etc/localtime;ln -s /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
COPY --from=build src/target/spring-boot-ecommerce-0.0.1-SNAPSHOT.jar /run/spring-boot-ecommerce-0.0.1-SNAPSHOT.jar

EXPOSE 8080

ENTRYPOINT java -jar /run/spring-boot-ecommerce-0.0.1-SNAPSHOT.jar **--spring.config.location=/run/src/main/resources/application.properties**

2. Thêm ConfigMap (file yml như trên

   ⇒ Lúc này ConfigMap sẽ được lưu trong Storage



▼ 3. Mount ConfigMap và trong Container

```
spec:
  containers:
    - image: zzzueszz/ecommerce-backend:v2
      imagePullPolicy: Always
      name: ecommerce-backend
      ports:
        - containerPort: 8080
          name: tcp
          protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
        - mountPath: /run/src/main/resources/application.properties
          name: ecommerce-backend-application-properties-config-volume
          subPath: application.properties
  volumes:
    - configMap:
        defaultMode: 420
        name: ecommerce-backend-application-properties-configmap
      name: ecommerce-backend-application-properties-config-volume
  dnsPolicy: ClusterFirst
  restartPolicy: Always
```

- volumes: Tạo 1 volumn để lưu configMap
- volumnMounts: Mount configMap vào trong container

4. Sau khi cấu hình sau,ta có thể chỉnh sửa data file application.properties tùy ý, và redeploy deployment để apply cấu hình

▼ Sử dụng Secret

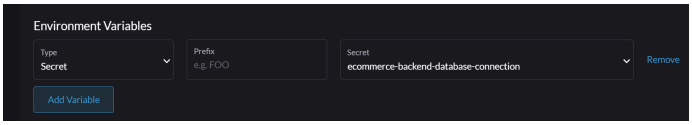
- Vấn đề: ConfigMap không đảm bảo vấn đề bảo mật, các dữ liệu nhạy cảm có thể bị lộ
- ⇒ Secret giải quyết vấn đề này

▼ Secret File

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: ecommerce-backend-database-connection
  namespace: ecommerce
type: Opaque
stringData:
  MARIADB_HOST: "192.168.189.115" #chú ý thay IP của bạn
  MARIADB_DB: "full-stack-ecommerce"
  MARIADB_PORT: '3306'
  MARIADB_USERNAME: "ecommerceapp"
  MARIADB_PASSWORD: "StrongPa55WorD"
```

▼ Config deployment.yml để apply secret

- Thêm Secret thủ công bằng giao diện
  1. Vào phần edit của deployment, map tới secret

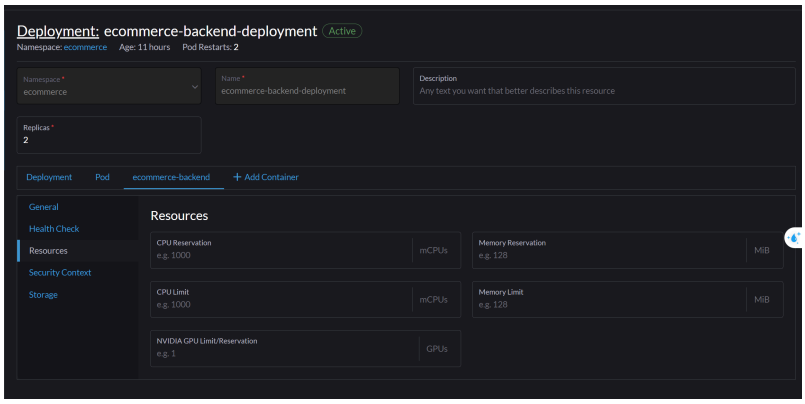


▼ Request and Limit

Thuộc tính	Request (yêu cầu)	Limit (giới hạn)
Mục đích	Đảm bảo container có tài nguyên tối thiểu để chạy	Giới hạn lượng tài nguyên tối đa container có thể sử dụng
Scheduling	Dùng để quyết định nếu Node có thể chạy container lên đó	Không ảnh hưởng đến việc lập lịch triển khai container
CPU	Container có thể sử dụng nhiều hơn request nếu có tài nguyên thừa	CPU sẽ bị "Throttled" nếu vượt quá ngưỡng.
Memory	Container có thể sử dụng nhiều hơn request nếu có tài nguyên thừa	Container sẽ bị kill (status restart) nếu vượt quá ngưỡng

⇒ Trong tùy dự án mà sẽ có chỉ số riêng để config autoscaling

- Config trên Rancher

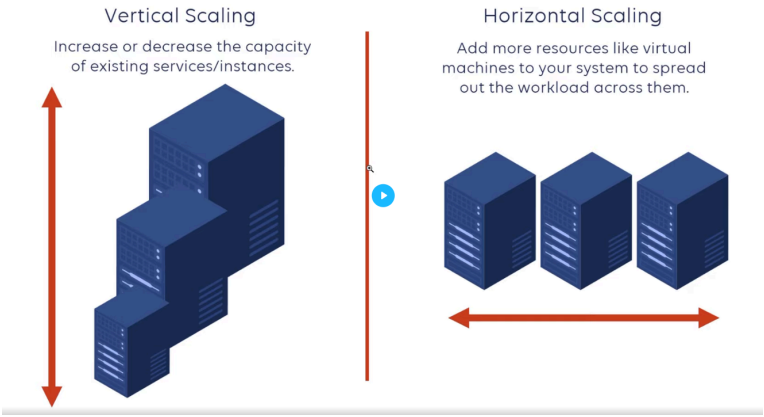


- Config file yml

```
image: zzzueszz/ecommerce-backend:v3
imagePullPolicy: Always
name: ecommerce-backend
ports:
  - containerPort: 8080
    name: tcp
    protocol: TCP
resources:
  requests:
    cpu: 500m
    memory: 512Mi
  limits:
    cpu: 1000m
    memory: 1028Mi
```

▼ AutoScaling

- **Bài toán:** Giả sử bình thường hệ thống chịu tải 10k concurrent user, nhưng vào 1 thời điểm, phải chịu tải lên tới 100k concurrent user ⇒ Lúc này phải scale-up hệ thống để đáp ứng tải trọng này ⇒ **Sử dụng AutoScaling**
- AutoScaling có 2 loại



  - **Vertical Scaling** : Đến ngưỡng, hệ thống sẽ tăng sức chứa - vd: memory, cpu để chịu tải

    ⇒ Khi đã tăng sức chứa, tới khi muốn scale-down xuống dễ gặp lỗi
  - **Horizontal Scaling**: Đến ngưỡng, hệ thống sẽ tạo ra nhiều virtual-machine - các replicas để phân chia tải trọng

▼ **HAP (Horizontal Pod Autoscaling)**

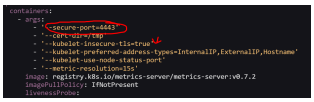
DOC:https://viblo.asia/p/hpa-la-gi-auto-scaling-pod-bang-hpa-va-keda-ByEZkerA5Q0

▼ Để có thể giám sát được thông tin CPU, MEMORY ⇒ Cài đặt thêm metrics server

```
# helm repo add  metric-server https://kubernetes-sigs.github.io/metrics-server/
# helm pull metric-server/metrics-server
# tar -xvf metrics-server-*
# helm install metric-server metrics-server -n kube-system
```

▼ **FIX LỖI TLS cho Metrics Server**

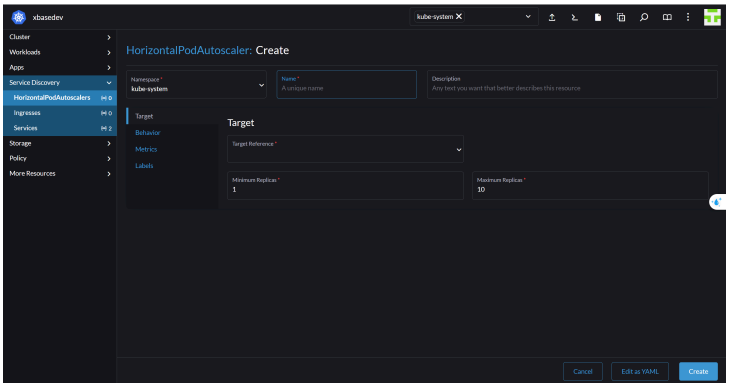
- Vào file yml deployment của metric server, chỉnh tất cả port sang 4443, và thêm flag như sau



▼ Command sử dụng cho metrics server

- **kubectl top nodes** : liệt kê tài nguyên sd của các node
- **kubectl top po -n ecommerce**: leietj kê tài nguyên sử dụng của các pod trong namespace

▼ Thêm tài nguyên HPA bằng rancher



▼ file yml

Cấu hình HPA

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: ecommerce-backend-autoscaling
  namespace: ecommerce
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: deployment
    name: ecommerce-backend-deployment
  minReplicas: 2
  maxReplicas: 4
  targetCPUUtilizationPercentage: 50
```

▼ Một số kiến thức về Rancher

Doc: https://devopsedu.vn/courses/khoa-hoc-kubenetes-thuc-te/lesson/bai-30-su-dung-rancher-2/

▼ RBAC

▼ NETWORK POLICY K8S

▼ TRIỂN KHAI CÔNG CỤ

▼ Storage

https://viblo.asia/p/k8s-basic-kubernetes-storage-qPoL7X6mVvk

▼ PV và PVC

https://viblo.asia/p/k8s-basic-kubernetes-storage-qPoL7X6mVvk

▼ Khái niệm

1. PersistentVolume (PV) - Ổ đĩa vật lý

🔷 **PersistentVolume (PV)** là một tài nguyên lưu trữ trong cụm Kubernetes. Nó đại diện cho một phần dung lượng lưu trữ vật lý có thể được sử dụng bởi các ứng dụng trong Pod.

🔷 PV có thể được cấp phát thủ công hoặc động (tự động tạo khi cần).

📌 **Ví dụ:**

Giả sử bạn có một ổ đĩa 100GB trên máy chủ, bạn có thể định nghĩa nó như một **PersistentVolume** trong Kubernetes để các ứng dụng có thể sử dụng.

📜 **Ví dụ YAML:**

```
yaml
CopyEdit
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data"
```

- `capacity: 10Gi` → Dung lượng lưu trữ là 10GB.
- `accessModes: ReadWriteOnce` → Chỉ một Pod có thể ghi, nhưng nhiều Pod có thể đọc.
- `persistentVolumeReclaimPolicy: Retain` → Khi PV không còn được sử dụng, nó vẫn giữ dữ liệu (không bị xóa).
- `hostPath: "/mnt/data"` → PV sử dụng thư mục `/mnt/data` trên node để lưu trữ dữ liệu.

---

2. PersistentVolumeClaim (PVC) - Yêu cầu sử dụng PV

🔷 **PersistentVolumeClaim (PVC)** là một yêu cầu từ một Pod để sử dụng một phần dung lượng của PersistentVolume.

🔷 Một ứng dụng trong Kubernetes không trực tiếp sử dụng PV mà sẽ tạo một **PVC** để yêu cầu lưu trữ. Nếu có PV phù hợp, Kubernetes sẽ liên kết PVC với PV đó.

📜 **Ví dụ YAML:**

```
yaml
CopyEdit
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
```

```
    requests:
      storage: 5Gi
```

- **Ứng dụng chỉ cần yêu cầu 5GB** (`storage: 5Gi`), Kubernetes sẽ tìm một PV có đủ dung lượng (10GB trong ví dụ trên) để liên kết.

💡 **Tóm tắt:**

- **PV**: Nhà kho chứa hàng (ổ đĩa thật).
- **PVC**: Phiếu yêu cầu thuê kho chứa hàng.
- **Pod**: Nhận được một phần của kho (khi PVC được cấp phát).

---

3. StorageClass - Cấp phát PV động

🔷 **StorageClass (SC)** là cách Kubernetes tạo **PV tự động** thay vì bạn phải khai báo PV thủ công.

🔷 Khi một PVC được tạo mà không có PV nào phù hợp, **StorageClass** sẽ giúp tự động tạo PV.

📜 **Ví dụ YAML:**

```
yaml
CopyEdit
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: my-storageclass
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
```

- `provisioner: kubernetes.io/aws-ebs` → Kubernetes sẽ tạo PV sử dụng **EBS (Elastic Block Store) của AWS**.
- `type: gp2` → Sử dụng loại lưu trữ `gp2` của AWS.

📜 **PVC sử dụng StorageClass:**

```
yaml
CopyEdit
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: my-storageclass
```

- Khi PVC này được tạo, **StorageClass** `my-storageclass` sẽ tự động cấp phát một PV phù hợp.

🎯 Tóm tắt đơn giản

Thành phần	Chức năng
PersistentVolume (PV)	Lưu trữ dữ liệu thật, do admin tạo hoặc tự động cấp phát.
PersistentVolumeClaim (PVC)	Yêu cầu sử dụng dung lượng lưu trữ từ PV.
StorageClass (SC)	Tự động tạo PV khi PVC không tìm thấy PV phù hợp.

📌 **Tương tự với thế giới thực:**

- **PV** = Ổ cứng hoặc kho chứa hàng thật.
- **PVC** = Đơn yêu cầu thuê kho.
- **StorageClass** = Hệ thống quản lý kho, tự tạo kho khi có yêu cầu.

▼ Setup

▼ 1. Cài NFS server trên database server

```
# sudo apt install nfs-server -y
# sudo mkdir /data
# sudo chown -R nobody:nogroup /data
# sudo chmod -R 777 /data
# sudo vi /etc/exports
/data *(rw,sync,no_subtree_check)
# sudo exportfs -rav
# sudo systemctl restart nfs-server
```

- **Mở file cấu hình NFS** `/etc/exports` và khai báo thư mục cần chia sẻ.
- **Chạy** `exportfs -rav` để áp dụng cấu hình.
- **Khởi động lại dịch vụ** `nfs-server` để chắc chắn thay đổi có hiệu lực.

2. Cài đặt nfs-common trên các node

`sudo apt install nfs-common -y`

3. Cấu hình PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /data
    server: 192.168.189.115 #chú ý thay đổi địa chỉ IP tương ứng
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs-storage
```

4. Cấu hình PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
```

```
  namespace: ecommerce
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  storageClassName: nfs-storage
```

5. Cấu hình pod mount dữ liệu vào thư mục /data của NFS server

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-nginx
  namespace: ecommerce
spec:
  containers:
    - image: nginx
      name: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: nfs-storage
  volumes:
    - name: nfs-storage
      persistentVolumeClaim:
        claimName: nfs-pvc
```

▼ StatefulSet và setup MariaDB/ Redis

▼ Lý do chọn type StatefulSet để deploy DB

🔥 Vì sao cần triển khai Database bằng StatefulSet trong Kubernetes?

Trong Kubernetes, **StatefulSet** là lựa chọn phù hợp nhất để triển khai **cơ sở dữ liệu quan trọng** (MySQL, PostgreSQL, MongoDB, Cassandra, etc.). Lý do chính là vì cơ sở dữ liệu cần **tính nhất quán, ổn định và lưu trữ dữ liệu lâu dài**, điều mà **Deployment không đảm bảo**.

Hãy phân tích chi tiết tại sao **StatefulSet** là lựa chọn tốt nhất cho database!

🚀 1. Tính nhất quán của dữ liệu (Data Consistency)

🔷 Database **không thể mất dữ liệu** mỗi khi Pod bị xóa hoặc restart.

🔷 **StatefulSet đảm bảo rằng mỗi Pod có PersistentVolume riêng**, giúp giữ dữ liệu ngay cả khi Pod bị xóa hoặc di chuyển.

📌 **Ví dụ:**

- Nếu bạn chạy PostgreSQL bằng **Deployment**, khi Pod chết và một Pod mới được tạo, dữ liệu sẽ bị mất nếu không có lưu trữ cố định.
- **StatefulSet gán mỗi Pod với một Persistent Volume riêng**, đảm bảo dữ liệu không bị mất khi Pod bị khởi động lại.

🆔 2. Tên Pod cố định (Stable Network Identity)

🔷 **StatefulSet tạo các Pod với tên cố định**, ví dụ `db-0`, `db-1`, `db-2`.

🔷 Điều này giúp database biết **chính xác** từng node là ai, tránh trường hợp bị thay đổi địa chỉ IP mỗi lần khởi động.

📌 **Ví dụ:**

- Với MongoDB **Replica Set**, node **primary** và **secondary** cần nhận diện nhau bằng **tên cố định** (ví dụ: `mongo-0`, `mongo-1`).
- Nếu dùng **Deployment**, Pod có thể khởi động lại với một tên ngẫu nhiên, làm MongoDB mất kết nối.
- **StatefulSet giữ nguyên tên**, đảm bảo hệ thống database hoạt động ổn định.

📂 3. Lưu trữ dữ liệu lâu dài (Persistent Storage per Pod)

🔷 **StatefulSet tạo PersistentVolume riêng biệt cho từng Pod** (PVC không bị thay đổi ngay cả khi Pod bị xóa).

🔷 **Deployment không đảm bảo điều này**, trừ khi bạn tự cấu hình thủ công.

📜 **Ví dụ StatefulSet với PVC riêng:**

```
yaml
CopyEdit
volumeClaimTemplates:
  - metadata:
      name: db-storage
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 10Gi
```

- Mỗi Pod (`db-0`, `db-1`, `db-2`) sẽ có một PVC riêng biệt (`db-storage-db-0`, `db-storage-db-1`), giúp bảo vệ dữ liệu.
- Nếu Pod `db-0` bị restart, nó vẫn gắn với PVC `db-storage-db-0`, dữ liệu không bị mất.

🔄 4. Khởi động / Dừng Pod theo thứ tự xác định

🔷 Một số database như **MongoDB, MySQL Cluster, Elasticsearch** cần khởi động **theo thứ tự** để đảm bảo hoạt động đúng.

🔷 **StatefulSet đảm bảo Pod được khởi động và tắt theo thứ tự (`db-0` → `db-1` → `db-2`).**

📌 **Ví dụ:**

- Nếu bạn triển khai **MongoDB Replica Set**, Pod **primary** phải chạy trước các **secondary**.
- **StatefulSet đảm bảo thứ tự khởi động này**, giúp cluster không bị lỗi.

📡 5. Khả năng mở rộng linh hoạt mà không mất dữ liệu (Scaling & Data Persistence)

🔷 Khi mở rộng số lượng Pod trong StatefulSet, Kubernetes sẽ **tạo Pod mới với tên cố định (`db-3`, `db-4`...) và gán PVC riêng**.

🔷 **Deployment không làm được điều này**, vì nó chỉ tạo Pod ngẫu nhiên mà không có PVC riêng.

📌 **Ví dụ thực tế:**

- Nếu bạn mở rộng MongoDB từ 3 node lên 5 node (`db-3`, `db-4`), các node mới sẽ tự động tham gia Replica Set mà không mất dữ liệu.

🔥 6. Tích hợp tốt với Headless Service (Truy cập trực tiếp vào từng DB node)

🔷 **StatefulSet thường được kết hợp với Headless Service (`ClusterIP: None`)** để cho phép các Pod database có thể tìm thấy nhau.

🔷 Điều này giúp **database cluster hoạt động đúng mà không bị thay đổi endpoint** mỗi khi Pod restart.

📜 **Ví dụ Headless Service cho PostgreSQL:**

```
yaml
CopyEdit
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  clusterIP: None
  selector:
    app: postgres
```

- Khi sử dụng, các Pod database có thể giao tiếp với nhau bằng DNS cố định:
  - `postgres-0.postgres.default.svc.cluster.local`
  - `postgres-1.postgres.default.svc.cluster.local`

✅ **Kết quả:** Database không bị lỗi khi một Pod bị restart hoặc thay đổi vị trí trong cluster.

⚡ Tóm tắt: Vì sao StatefulSet quan trọng cho Database?

Lý do	Deployment	StatefulSet
Giữ dữ liệu khi restart	❌ Không (mất dữ liệu nếu không có PVC riêng)	✅ Có (mỗi Pod có PVC riêng)
Tên Pod cố định	❌ Không (tên thay đổi ngẫu nhiên)	✅ Có (ví dụ: `db-0`, `db-1`, `db-2`)
Thứ tự khởi động / tắt Pod	❌ Không (có thể chạy bất kỳ Pod nào trước)	✅ Có (luôn theo thứ tự `0 → 1 → 2`)
Khả năng scaling mà không mất dữ liệu	❌ Không đảm bảo	✅ Có (mỗi replica có PVC riêng)
Hỗ trợ Database Cluster (MySQL, MongoDB, PostgreSQL, etc.)	❌ Không phù hợp	✅ Tối ưu cho Database

✅ **Kết luận: StatefulSet là lựa chọn tốt nhất** khi triển khai cơ sở dữ liệu trong Kubernetes, vì nó đảm bảo **tính nhất quán, dữ liệu không mất, địa chỉ Pod ổn định và khả năng mở rộng**.

▼ Triển khai MariaDB

NOTE: Làm sao để setup các pod trong deployment có thể gọi tới DB mà không cần thông qua service

1. Chỉnh sửa file cấu hình NFS (db-server)

```
# sudo vi /etc/exports
/data *(rw,sync,no_subtree_check,no_root_squash)
# sudo exportfs -rav
# sudo systemctl restart nfs-server
```

2. Cấu hình StatefulSet MariaDB

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mariadb
  namespace: ecommerce
spec:
  serviceName: mariadb-service
  replicas: 1
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      securityContext:
        fsGroup: 65534
      containers:
      - name: mariadb
        image: mariadb:latest
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "devopseduvn"
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mariadb-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mariadb-storage
        persistentVolumeClaim: # Dùng chung volumn cho các pod
        # Dùng volumeClaimTemplates => cấp phát volumn độc lập cho pod
          claimName: nfs-pvc
```

3. Cấu hình NodePort MariaDB để test từ bên ngoài

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mariadb-service
  namespace: ecommerce
spec:
  selector:
    app: mariadb
  type: NodePort
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 31306
```

▼ Triển khai Redis

▼ MONITORING / LOG/ BACKUP

▼ DaemonSet

- Có tác dụng quản lý pod, sinh ra với mục tiêu đảm báo sẽ có Pod được triển khai toàn bộ trên các node thuộc cluster
- Khi 1 worker join và cluster ⇒ sẽ tự động có 1 agent triển khai trên node đó
- Khi 1 worker out cluster, agent sẽ được thu hồi bởi trình collected garbage

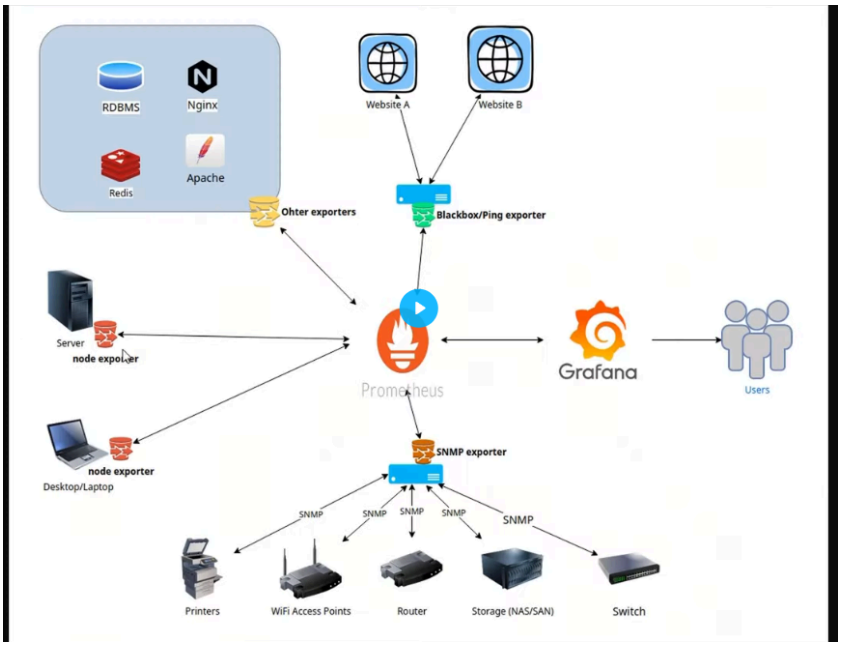
▼ Tain và Tolerance

- Là cơ chế giúp kiểm soát được việc pod được scheduled trên các node

⇒ **Dùng để ngăn không cho Pod chạy trên một số Node, trừ khi Pod có `toleration` tương ứng.**

▼ **Triển khai Kube prometheus stack**

DOC:https://devopsedu.vn/courses/khoa-hoc-kubenetes-thuc-te/lesson/bai-39-trien-khai-kube-prometheus-stack-2/



- Tư duy xây dựng hệ thống
  - CODE FIRST: Triển khai công cụ = heml ⇒ để tối đa phần mặc định, những thứ cần sửa thì config trong file values.yml

▼ Tiến hành triển khai

1. Tạo thư mục lưu dữ liệu

```
# sudo mkdir /data/monitoring
# sudo chown -R nobody:nogroup /data/
# sudo chmod -R 777 /data
```

2. Tạo Namespace
3. Tạo PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: monitoring-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /data/monitoring
    server: 192.168.189.115 #Chú ý thay đổi địa chỉ IP tương ứng của bạn
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs-storage
```

4. Cài đặt kube prometheus stack

```
# kubectl create ns monitoring
# helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
# helm repo update
# helm upgrade --install devopseduvn prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.accessModes[0]=ReadWriteOnce \
  --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.resources.requests.storage=10Gi \
  --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.storageClassName=nfs-storage
```

5. Tạo ingress Grafana
6. Tạo ingress Prometheus

▼ Dashboard Grafana

▼ FIX LOI

MK: prom-operator, TK: Admin

```
kubectl create clusterrolebinding calico \
  --clusterrole=cluster-admin \
  --serviceaccount=kube-system:calico-kube-controllers
```

4️⃣ Xóa pod Calico để restart lại

```
sh
CopyEdit
kubectl delete pods -n kube-system -l k8s-app=calico-node
```

5️⃣ Kiểm tra lại node `k8s-master-3`

Chạy lệnh:

```
sh
CopyEdit
kubectl describe node k8s-master-3
```

Nếu có taint `node-role.kubernetes.io/control-plane:NoSchedule`, thì cần remove:

```
sh
CopyEdit
kubectl taint nodes k8s-master-3 node-role.kubernetes.io/control-plane:NoSchedule-
```

- ▼ Công cụ Uptime kuma
- ▼ Backup & Restore sử dụng **Velero**
  - ▼ Lỗi khi kết nối velero với minio
    - Check lỗi: velero backup-location get -o yaml
    - Xóa namespace velero và làm lại : kubectl delete -n velero
  - ▼ Tìm hiểu công cụ Velero
    - Là công cụ phát triển bởi VMWare, giúp sao lưu (backup), phát triển (migrate) và hồi phục (restore) cụm cluster k8s
    - Cách hoạt động:
      - sử dụng object storage (aws s3, gg cloude, hoặc 1 bucket)để lưu trữ dữ liệu
    -
  - ▼ Cài đặt Velero

    Cài đặt Velero client (Thực hiện trên server k8s-master-1 hoặc kubectl shell Rancher)

```
# wget https://github.com/vmware-tanzu/velero/releases/download/v1.15.0/velero-v1.15.0-linux-amd64.tar.gz
# tar -xvf velero-v1.15.0-linux-amd64.tar.gz
# sudo mv velero-v1.15.0-linux-amd64/velero /usr/local/bin
```

NOTE: Tác dung của /usr/local/bin:

- ▼ Tác dụng

Tác dụng của `/usr/local/bin` trong Linux & Unix

Thư mục `/usr/local/bin` là một trong những đường dẫn quan trọng trong hệ thống Linux & Unix, thường được dùng để lưu trữ các tệp thực thi (binaries) do người dùng cài đặt thủ công hoặc từ các nguồn bên ngoài hệ thống quản lý gói chính thức.

---

1. `/usr/local/bin` dùng để làm gì?

- **Chứa các chương trình thực thi (binaries) do người dùng cài đặt**→ Nếu bạn biên dịch hoặc cài đặt phần mềm thủ công từ mã nguồn, các tệp thực thi thường được đặt ở đây.
- **Ưu tiên cao hơn so với** `/usr/bin` → Hệ thống thường tìm kiếm tệp thực thi trong `/usr/local/bin` trước `/usr/bin`, giúp bạn chạy phiên bản tùy chỉnh của một chương trình mà không ghi đè bản gốc.
- **Không bị ảnh hưởng khi cập nhật hệ thống**→ Khi bạn cập nhật OS hoặc package system, thư mục `/usr/local/bin` thường **không bị ghi đè**, giúp bảo vệ các chương trình tùy chỉnh của bạn.

- **Hỗ trợ quản lý phần mềm cục bộ** → Nếu bạn muốn cài đặt phần mềm chỉ dùng riêng cho hệ thống của mình mà không ảnh hưởng đến hệ thống chung, bạn có thể đặt nó vào đây thay vì `/usr/bin` hoặc `/bin`.

---

2. `/usr/local/bin` khác gì với `/usr/bin`, `/bin`?

Thư mục	Chứa gì?	Ưu tiên sử dụng
`/usr/local/bin`	Binary cài thủ công bởi người dùng, không thuộc hệ thống quản lý gói.	Ưu tiên để cài đặt chương trình tùy chỉnh.
`/usr/bin`	Binary từ hệ thống quản lý gói (`apt`, `yum`, `dnf`, `pacman`).	Chứa hầu hết các lệnh mặc định của hệ thống.
`/bin`	Binary cơ bản, cần thiết để hệ thống khởi động.	Chứa các lệnh quan trọng như `ls`, `cat`, `cp`.

📌 **Lưu ý:**

- `/usr/local/bin` thường được ưu tiên trong `$PATH`, nên nếu có hai chương trình cùng tên trong `/usr/local/bin` và `/usr/bin`, thì hệ thống sẽ chạy chương trình trong `/usr/local/bin` trước.
- Nếu bạn muốn chạy một chương trình cụ thể từ `/usr/bin`, có thể dùng đường dẫn tuyệt đối, ví dụ:

```
bash
CopyEdit
/usr/bin/python3
```

---

3. Khi nào nên dùng `/usr/local/bin`?

✅ Khi **cài đặt phần mềm từ mã nguồn** (không dùng trình quản lý gói như `apt`, `yum`, `dnf`).

✅ Khi **tạo script tùy chỉnh** và muốn nó có thể chạy ở bất kỳ đâu trong hệ thống mà không cần gọi đường dẫn đầy đủ.

✅ Khi **muốn ghi đè một chương trình hệ thống mà không làm ảnh hưởng đến bản gốc**.

Ví dụ: Nếu bạn có một script `my_script.sh`, bạn có thể đặt nó vào `/usr/local/bin` để chạy từ bất cứ đâu:

```
bash
CopyEdit
sudo mv my_script.sh /usr/local/bin/my_script
chmod +x /usr/local/bin/my_script
```

Bây giờ, bạn có thể chạy `my_script` từ bất kỳ thư mục nào.

---

4. Cách kiểm tra và quản lý `/usr/local/bin`

- **Kiểm tra thư mục `/usr/local/bin` có trong `$PATH` không:**

```
bash
CopyEdit
echo $PATH
```

- **Liệt kê các tệp trong `/usr/local/bin`:**

```
bash
CopyEdit
ls -l /usr/local/bin
```

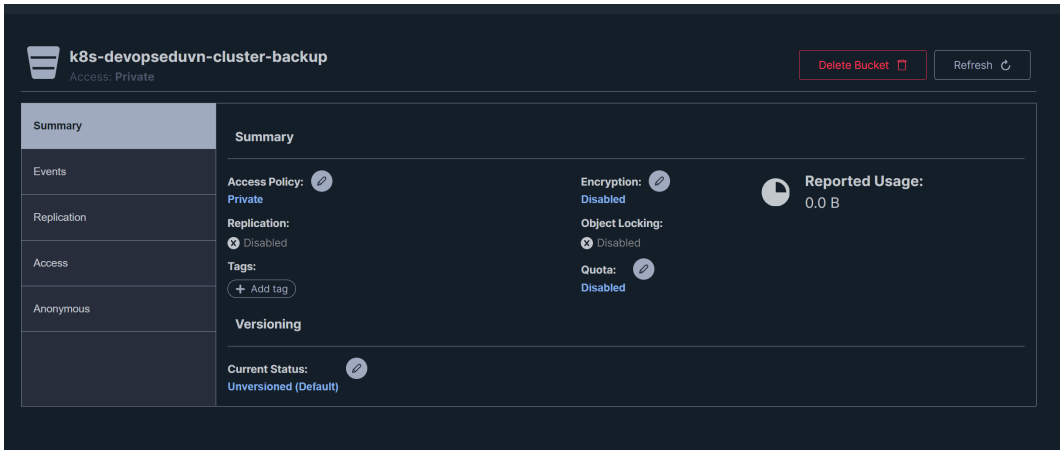
- **Xóa một binary trong** `/usr/local/bin` **:**

```
bash
CopyEdit
sudo rm /usr/local/bin/tên_file
```

▼ Cài đặt Bucket (Minio) bằng docker-compose

```
version: '3'
services:
  minio:
    image: minio/minio
    container_name: minio
    ports:
      - "9000:9000"
      - "9001:9001"
    volumes:
      - ./storage:/data
    environment:
      MINIO_ROOT_USER: devopseduvn
      MINIO_ROOT_PASSWORD: devopseduvn
    command: server --console-address ":9001" /data
```

▼ Tạo bucket



Cấu hình các biến MiniO (thực hiện trên server k8s-master-1 hoặc kubectl shell Rancher)

```
# export MINIO_URL="http://192.168.189.115:9000"
# export MINIO_ACCESS_KEY_ID="RfY4sTuZI6F7zvDZuhMm"
# export velero backup-location describe default
="xWbWfuS0j4YZTkpm8kOX4JNZmwGE1VhPQ0OwQKUH"
# export MINIO_BUCKET="k8s-devopseduvn-cluster-backup"
```

Cài đặt Velero với MinIO làm backend lưu trữ các bản sao lưu của Kubernetes (thực hiện trên server k8s-master-1 hoặc kubectl shell Rancher)

```
velero install \
  --provider aws \
  --bucket $MINIO_BUCKET \
  --secret-file <(echo -e "[default]\naws_access_key_id=$MINIO_ACCESS_KEY_ID\naws_secret_access_key
=$MINIO_SECRET_KEY_ID") \
  --use-node-agent \
  --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=$MINIO_URL \
  --plugins velero/velero-plugin-for-aws:v1.5.0 \
  --namespace velero
```

Backup tài nguyên của Namespace cụ thể (thực hiện trên server k8s-master-1 hoặc kubectl shell Rancher)

```
velero backup create ecommerce-v1 --include-namespaces ecommerce
```

Restore bản backup cụ thể (thực hiện trên server k8s-master-1 hoặc kubectl shell Rancher)

```
# velero restore create ecommerce-v1 --from-backup ecommerce-v1 --include-namespaces ecommerce
```

Đặt lịch tự động backup – ví dụ backup hàng ngày (thực hiện trên server k8s-master-1 hoặc kubectl shell Rancher)

```
# velero schedule create daily-cluster-backup --schedule="0 0 * * *" --include-namespaces '*'
```

- Mở rộng phân vùng ổ cứng
  - Sử dụng hết dung lượng ổ cứng

    Nếu bạn muốn sử dụng toàn bộ dung lượng 23GB của `sda3`, bạn có thể mở rộng Logical Volume `ubuntu--vg-ubuntu--lv` bằng các bước sau:

    ### Bước 1: Kiểm tra dung lượng trống trong Volume Group

    bash

    Copy

    ```
    sudo vgdisplay ubuntu-vg
    ```

    - Ghi lại giá trị `Free PE / Size`.

    ### Bước 2: Mở rộng Logical Volume

    bash

    Copy

    ```
    sudo lvextend -l +100%FREE /dev/ubuntu-vg/ubuntu-lv
    ```

    - Lệnh này sẽ sử dụng toàn bộ dung lượng còn trống trong Volume Group để mở rộng Logical Volume.

    ### Bước 3: Mở rộng filesystem

    Sau khi mở rộng Logical Volume, bạn cần mở rộng filesystem để sử dụng dung lượng mới:

    - Nếu bạn sử dụng `ext4`:

      bash

Copy

```
sudo resize2fs /dev/ubuntu-vg/ubuntu-lv
```

- Nếu bạn sử dụng `xfs`:

  bash

  Copy

```
sudo xfs_growfs /
```

Bước 4: Kiểm tra lại dung lượng

bash

Copy

```
df -h
```

- Bây giờ, Logical Volume `ubuntu--vg-ubuntu--lv` sẽ sử dụng toàn bộ dung lượng có sẵn trong Volume Group.

▼ CRON JOB

🔍 Cron Job trong Linux là gì?

Cron Job là một **tác vụ tự động chạy theo lịch trình** trong hệ thống Linux. Nó giúp bạn **chạy lệnh hoặc script định kỳ** mà không cần can thiệp thủ công.

- **Crontab (cron table)**: File chứa danh sách các cron jobs.
- **Cron daemon (`crond`)**: Dịch vụ nền giúp thực thi cron jobs.
- **Cú pháp crontab**: `* * * * command_to_run`
  - `* * * *` là lịch trình chạy (biểu thức thời gian).
  - `command_to_run` là lệnh sẽ được thực thi.

🕒 Giải thích biểu thức thời gian

`0 0 * * *`

```
*    *    *    *    *
|    |    |    |    |
|    |    |    |    +---- Ngày trong tuần (0 - Chủ Nhật, 6 - Thứ Bảy)
|    |    |    +---------- Tháng (1-12)
|    |    +--------------- Ngày trong tháng (1-31)
|    +-------------------- Giờ (0-23)
+------------------------- Phút (0-59)
```

Biểu thức cron gồm 5 phần:

👉 `0 0 * * *` nghĩa là:

- **Phút = 0** (bắt đầu từ phút 0).
- **Giờ = 0** (tức là 12:00 AM, nửa đêm).
- **Ngày trong tháng = `*`** (mỗi ngày trong tháng).
- **Tháng = `*`** (mỗi tháng trong năm).
- **Ngày trong tuần = `*`** (mỗi ngày trong tuần).

✅ **Kết luận: Lệnh này sẽ chạy vào lúc 00:00 mỗi ngày**.

Cách kiểm tra & quản lý cron job

1️⃣ **Xem danh sách các cron job trên hệ thống:**

```
crontab -l
```

2️⃣ **Chỉnh sửa cron job:**

```
crontab -e
```

Sau đó thêm dòng:

```
0 0 * * * /usr/local/bin/backup_script.sh
```

📌 **Lưu ý:** Nếu sử dụng `crontab -e`, cron job sẽ chạy với user hiện tại.

3️⃣ **Kiểm tra logs của cron job:**

```
cat /var/log/syslog | grep CRON
```

▼ HIGH AVAILABILITYS

▼ MONGODB

▼ **1. Cài đặt mongodb**

Thực hiện trên cả 3 servers

Cài đặt MongoDB

Cài đặt các công cụ cần thiết

```
# apt install -y net-tools telnet traceroute
```

Cài đặt MongoDB version 7.0.x cho Ubuntu 20.04

```
# Cập nhật danh sách package và cài đặt các gói cần thiết
sudo apt update && sudo apt install -y software-properties-common apt-transport-https ca-certificates

# Thêm key GPG của MongoDB
curl -fsSL https://pgp.mongodb.com/server-7.0.asc | sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg --dearmor

# Thêm repository MongoDB đúng cho Ubuntu 20.04 (Focal Fossa)
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list

# Cập nhật danh sách package
```

```
sudo apt update

# Cài đặt MongoDB
sudo apt install -y mongodb-org

# Kích hoạt và khởi động dịch vụ MongoDB
sudo systemctl enable mongod
sudo systemctl start mongod

# Kiểm tra trạng thái dịch vụ MongoDB
sudo systemctl status mongod
```

Mở file cấu hình để chỉnh sửa thiết lập mongodb thành cluster

```
# vi /etc/mongod.conf
```

Tìm và sửa 2 phần này trong file (xem video bài giảng để rõ hơn)

```
 bindIp: 0.0.0.0

replication:
  replSetName: "mongodb-devopseduvn"
```

Khởi động lại mongodb để áp dụng cấu hình mới

```
# systemctl restart mongod
```

▼ **2. Khởi tạo cụm mongodb**

Thực hiện trên server 1

Khởi tạo cụm mongodb

Truy cập môi trường mongodb

```
# mongosh
```

Khởi tạo cụm

```
test> rs.initiate( {
      _id : "mongodb-devopseduvn",
      members: [
        { _id: 0, host: "k8s-master-1:27017" },
        { _id: 1, host: "k8s-master-2:27017" },
        { _id: 2, host: "k8s-master-3:27017" }
      ]
   })

mongodb-cluster-devopseduvn [direct: secondary] test> rs.status()
mongodb-cluster-devopseduvn [direct: primary] test> rs.isMaster()
```

Thêm thử dữ liệu test tính HA

```
mongodb-cluster-devopseduvn [direct: primary] test> use devopseduvndb
mongodb-cluster-devopseduvn [direct: primary] devopseduvndb> db.courses.insertOne({ "name": ["DevO
ps for freshers", "pipeline DevSecOps", "HA tools"] })
```

Cấu hình priority cho 3 server (để đảm bảo đồng bộ thứ tự ưu tiên với cấu hình VIP)

```
mongodb-devopseduvn [direct: primary] test> cfg = rs.conf()
mongodb-devopseduvn [direct: primary] test> cfg.members[0].priority = 100
mongodb-devopseduvn [direct: primary] test> cfg.members[1].priority = 90
mongodb-devopseduvn [direct: primary] test> cfg.members[2].priority = 80
mongodb-devopseduvn [direct: primary] test> rs.reconfig(cfg)
```

▼ 3. Kiểm tra tính đồng bộ dữ liệu trên server 2 và server 3

Thực hiện trên server 2 hoặc server 3

Kiểm tra tính đồng bộ dữ liệu

Truy cập môi trường mongodb

```
# mongosh
```

Kiểm tra dữ liệu

```
mongodb-cluster-devopseduvn [direct: secondary] test> show dbs
mongodb-cluster-devopseduvn [direct: secondary] test> use devopseduvndb
mongodb-cluster-devopseduvn [direct: secondary] devopseduvndb> show collections
```

Kiểm tra tính cluster replica không thể thêm dữ liệu

```
mongodb-cluster-devopseduvn [direct: secondary] devopseduvndb> db.accounts.insertOne({ "username":
["elroydevops", "manhnv"] })
```

▼ 4. Kiểm tra tính HA

Thực hiện trên server 1

Kiểm tra tính HA

Tắt/khởi động lại server giả định server bị lỗi

```
# reboot
```

Thực hiện trên server 2 hoặc server 3

Kiểm tra tính HA

Kiểm tra xem mongodb server 2 hoặc 3 xem được đề xuất làm primary không tương tự như video bài giảng

▼ 5. Cấu hình VIP (Virtual IP)

▼ Giải thích

Cấu hình VIP (Virtual IP)

- **Không có VIP**:
  - Ứng dụng phải biết **tất cả IP của các node DB** (ví dụ: 3 node).

    - Khi **Primary bị sập**, ứng dụng **phải tự động chuyển sang node mới**.
    - Điều này **tăng độ phức tạp** và có thể gây downtime nếu không xử lý tốt.
- **Có VIP**:
    - VIP **là một IP ảo** được cấu hình để trỏ đến node Primary hoặc một node khả dụng.
    - Nếu Primary bị sập, **VIP sẽ tự động chuyển sang node mới**.
    - Ứng dụng chỉ cần **kết nối đến VIP**, không cần thay đổi cấu hình khi có failover.

🔷 Lưu ý quan trọng

- VIP chỉ **giúp điều phối kết nối**, không thay thế **cơ chế failover** của MongoDB.
- Để VIP hoạt động, cần sử dụng **Keepalived**, **HAProxy**, hoặc **Load Balancer** để quản lý.
- Trong MongoDB Replica Set, nếu dùng VIP, nó nên **trỏ đến mongos hoặc một load balancer**, **không nên trỏ trực tiếp vào primary** vì MongoDB tự động quản lý failover.

Bạn có thể dùng **Keepalived + VIP** để đảm bảo **VIP luôn trỏ tới Primary**. 🚀

Cài đặt keepalived

```
# apt install -y keepalived
```

Thực hiện trên server 1

Cấu hình keepalived

Thêm file cấu hình

```
# vi /etc/keepalived/keepalived.conf
```

Nội dung sau (xem bài giảng để hiểu chi tiết)

```
vrrp_instance VI_1 {
   state MASTER
   interface ens33
   virtual_router_id 51
   priority 100
   advert_int 1
   authentication {
      auth_type PASS
      auth_pass password
   }
   virtual_ipaddress {
      192.168.213.14
   }
}
```

Thực hiện trên server 2

Cấu hình keepalived

Thêm file cấu hình

```
# vi /etc/keepalived/keepalived.conf
```

Nội dung sau (xem bài giảng để hiểu chi tiết)

```
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass password
    }
    virtual_ipaddress {
        192.168.213.14
    }
}
```

Thực hiện trên server 3

Cấu hình keepalived

Thêm file cấu hình

```
# vi /etc/keepalived/keepalived.conf
```

Nội dung sau (xem bài giảng để hiểu chi tiết)

```
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 80
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass password
    }
    virtual_ipaddress {
        192.168.213.14
    }
}
```

Thực hiện trên cả 3 servers

Cấu hình VIP (virtual IP)

Khởi động lại keepalived

```
# systemctl restart keepalived
```

▼ MARIADB

DOCS:https://viblo.asia/p/cai-dat-ha-galera-cluster-mariadb-su-dung-maxscale-tren-centos-6-ZDEvLRaEeJb

▼ 1. Cài đặt MariaDB

Cài đặt các công cụ cần thiết

```
# apt install -y net-tools telnet traceroute
```

Cài đặt mariadb server và mariadb client

```
# apt install -y mariadb-server mariadb-client
```

Thực hiện trên server 1

Cấu hình mariadb galera cluster

Thêm file cấu hình galera

```
# vi /etc/mysql/conf.d/galera.cnf
```

Nội dung như sau

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Cluster Configuration
wsrep_cluster_name="mariadb-cluster-devopseduvn"
wsrep_cluster_address="gcomm://192.168.189.111,192.168.189.112,192.168.189.113"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Node Configuration
wsrep_node_address="192.168.189.111"
wsrep_node_name="server-1"
```

Thực hiện trên server 2

Cấu hình mariadb galera cluster

Thêm file cấu hình galera

```
# vi /etc/mysql/conf.d/galera.cnf
```

Nội dung như sau

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
```

```
# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Cluster Configuration
wsrep_cluster_name="mariadb-cluster-devopseduvn"
wsrep_cluster_address="gcomm://192.168.189.111,192.168.189.112,192.168.189.113"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Node Configuration
wsrep_node_address="192.168.189.112"
wsrep_node_name="server-2"
```

Thực hiện trên server 3

Cấu hình mariadb galera cluster

Thêm file cấu hình galera

```
# vi /etc/mysql/conf.d/galera.cnf
```

Nội dung như sau

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Cluster Configuration
wsrep_cluster_name="mariadb-cluster-devopseduvn"
wsrep_cluster_address="gcomm://192.168.189.111,192.168.189.112,192.168.189.113"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Node Configuration
wsrep_node_address="192.168.189.113"
wsrep_node_name="server-3"
```

▼ 2. Cấu hình Mariadb Galera cluster và kiểm tra đồng bộ dữ liệu

▼ **LỖI NẾU LỠ GALERA_NEW_CLUSTER Ở NHIỀU NODE**

⇒ Lúc này, các node sẽ không join vô cùng 1 cluster ⇒ cách khắc phục là stop service mariadb trên các node, và tạo galera_new_cluster trên 1 node duy nhất

Thực hiện trên server 3

Cấu hình mariadb galera cluster

Thêm file cấu hình galera

```
# vi /etc/mysql/conf.d/galera.cnf
```

Nội dung như sau

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Cluster Configuration
wsrep_cluster_name="mariadb-cluster-devopseduvn"
wsrep_cluster_address="gcomm://192.168.213.11,192.168.213.12,192.168.213.13"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Node Configuration
wsrep_node_address="192.168.213.13"
wsrep_node_name="server-3"
```

Thực hiện trên server 1

Cấu hình mariadb galera cluster

Tắt mariadb server

```
# systemctl stop mariadb
```

Khởi tạo cụm mariadb galera cluster

```
# galera_new_cluster
```

Kiểm tra cụm mariadb galera cluster (thấy số lượng 1)

```
# mysql -u root -e "SHOW STATUS LIKE 'wsrep_cluster_size'"
```

Thực hiện trên server 2 và server 3

Cấu hình mariadb galera cluster

Khởi động lại mariadb server để áp dụng cấu hình mới join vào cụm

```
# systemctl restart mariadb
```

Thực hiện trên server 1

Kiểm tra tính đồng bộ dữ liệu

Truy cập vào môi trường mariadb

```
# mysql
```

Thêm dữ liệu

```
MariaDB [(None)]> create database devopseduvndb;
MariaDB [(None)]> use devopseduvndb;
MariaDB [devopseduvndb]> create table series (Name VARCHAR(50) PRIMARY KEY, Lever VARCHAR(25), R
elease_date DATE);
MariaDB [devopseduvndb]> insert into series (Name, Lever, Release_date) VALUES ('DevOps for Freshers',
'Basic/Freshers', '2024-01-15'), ('Xây dựng quy trình pipeline DevSecOps', 'Thực tế chuyên sâu', '2024-06-1
6'), ('HA tools', 'Advanced/Thực tế', '2024-08-18');
```

Thực hiện trên server 2 hoặc server 3

Kiểm tra tính đồng bộ dữ liệu

Truy cập vào môi trường mariadb

```
# mysql
```

Kiểm tra dữ liệu

```
MariaDB [(None)]> use devopseduvndb;
MariaDB [devopseduvndb]> show databases;
MariaDB [devopseduvndb]> show tables;
MariaDB [devopseduvndb]> select * from series;
```

▼ 3. Cài đặt MAXSCALE

▼ **Max scale là gì và tại sao phải cần?**

▼ Max scale là gì?

MaxScale là gì?

MaxScale là **một proxy layer dành cho MariaDB**, được phát triển bởi MariaDB Corporation. Nó hoạt động như một **load balancer**, **query router** và **công cụ bảo mật** giữa ứng dụng và database.

MaxScale giúp quản lý truy vấn SQL đến Galera Cluster hoặc Replica Set MariaDB một cách thông minh, giảm tải cho database và đảm bảo hệ thống hoạt động ổn định.

▼ Tại sao cần?

Tại sao cần MaxScale trên Galera Cluster MariaDB?

Galera Cluster cho phép **multi-master replication**, nghĩa là bất kỳ node nào cũng có thể đọc và ghi. Tuy nhiên, có một số vấn đề thực tế khi triển khai:

1. **Cần Load Balancer để tối ưu truy vấn**
   - Nếu ứng dụng gửi truy vấn ngẫu nhiên đến các node, có thể gây mất cân bằng tải.
   - MaxScale giúp điều hướng truy vấn một cách thông minh:

    - **Đọc (SELECT)** có thể được phân tán giữa nhiều node.
    - **Ghi (INSERT, UPDATE, DELETE)** có thể được gửi đến một node cụ thể để giảm xung đột.
  2. **Tránh Read/Write Conflict**
    - Nếu bạn cho phép **ghi trên nhiều node**, có thể xảy ra transaction rollback do xung đột.
    - MaxScale có thể đảm bảo **chỉ gửi lệnh ghi đến một node**, trong khi vẫn phân tán lệnh đọc ra nhiều node khác.
  3. **Tự động Failover**
    - Khi một node bị lỗi, MaxScale có thể tự động **định tuyến truy vấn sang các node còn lại**, giúp hệ thống không bị gián đoạn.
  4. **Connection Pooling & Query Routing**
    - Giúp giảm tải kết nối vào database bằng cách duy trì một **pool kết nối**.
    - Có thể **định tuyến truy vấn theo loại hoặc người dùng**, ví dụ:
      - User A chỉ được gửi đến Node 1.
      - Các truy vấn `SELECT` chỉ được gửi đến node có latency thấp nhất.
  5. **Bảo mật**
    - Chặn SQL injection ở mức proxy.
    - Kiểm soát truy cập dựa trên IP hoặc User.

```
wget https://downloads.mariadb.com/MaxScale/2.5.0/ubuntu/dists/focal/main/binary-amd64/maxscale-2.5.0-1.ubuntu.focal.x86_64.deb
```

And then install MaxScale

```
dpkg -i maxscale-2.5.0-1.ubuntu.focal.x86_64.deb
```

▼ 4. Cấu hình lại Max scale và mariadb

▼ Cấu hình file /etc/maxscale.cnf

CHỈNH TRÊN SERVER 1,2,3 (COPY I CHANG)

- Định nghĩa Server

```
# Server definitions
#
# Set the address of the server to the network
# address of a MariaDB server.
#

[server1]
type=server
address=192.168.189.111
port=3306
protocol=MariaDBBackend

[server2]
type=server
address=192.168.189.112
port=3306
protocol=MariaDBBackend

[server3]
type=server
address=0.0.0.0
port=3306
protocol=MariaDBBackend
# Monitor for the servers
```

- Đặt tk mk cho user maxscale

```
[MariaDB-Monitor]
type=monitor
module=mariadbmon
servers=server1
user=maxscale
password=devopseduvn
monitor_interval=2000
```

```
[Read-Only-Service]
type=service
router=readconnroute
servers=server1
user=maxscale
password=devopseduvn
router_options=slave
```

- Thêm server 2, 3 vào Read-Write-Service

```
[Read-Write-Service]
type=service
router=readwritesplit
servers=server1,server2,server3
user=maxscale
password=devopseduvn
```

- Đảm bảo firewall mở port 4006, 4008

▼ Bind cho mọi IP có thể truy cập port 3306

CHỈNH TRÊN 3 SERVER

- Chỉnh file /etc/mysql/mariadb.conf.d/50-server.cnf

▼ Tạo user, và test HA

Thực hiện trên server 1

Tạo user maxscale trong mariadb

Truy cập vào môi trường mariadb

```
# mysql
```

Các câu lệnh tạo user

```
MariaDB [(None)]> CREATE USER 'maxscale'@'%' IDENTIFIED BY 'devopseduvn';
MariaDB [(None)]> GRANT REPLICATION CLIENT ON *.* TO 'maxscale'@'%';
MariaDB [(None)]> GRANT REPLICATION SLAVE ADMIN ON *.* TO 'maxscale'@'%';
MariaDB [(None)]> GRANT REPLICA MONITOR ON *.* TO 'maxscale'@'%';
MariaDB [(None)]> FLUSH PRIVILEGES;
```

▼ POSTGRESQL

DOC: https://docs.percona.com/postgresql/14/solutions/high-availability.html#architecture-layout

▼ 1. CÀI ĐẶT PERCONA CHO POSTGRESQL

Thực hiện trên cả 3 servers

Cài đặt percona cho postgresql

```
# wget https://repo.percona.com/apt/percona-release_latest.$(lsb_release -sc)_all.deb
# dpkg -i percona-release_latest.$(lsb_release -sc)_all.deb
# apt update -y
# percona-release setup ppg14
# apt install -y percona-ppg-server-14
# systemctl status postgresql
```

Cài partroni, etcd, pgBackRest

```
# apt install -y python3-pip python3-dev binutils
# apt install percona-patroni etcd etcd-server etcd-client percona-pgbackrest -y
# systemctl stop {etcd,patroni,postgresql} && systemctl disable {etcd,patroni,postgresql}
# rm -rf /var/lib/postgresql/14/main
```

▼ 2. CẤU HÌNH ETCD CLUSTER

▼ Có 2 cách dựng cụm etcd cluster

▼ **Dựng một lần (Static Bootstrapping)**

Mô tả:

- Tất cả các node trong cluster được cấu hình ngay từ đầu với danh sách đầy đủ các thành viên (nodes) của cụm.
- Mọi node phải biết trước **tất cả các node khác** trong cụm ngay từ lúc khởi động.

Thực hiện:

Bước 1: Tạo File Cấu Hình cho Node 1 (`/etc/etcd/etcd.conf.yml`)

```
yaml
CopyEdit
name: node1
data-dir: /var/lib/etcd
initial-cluster-token: "etcd-cluster-1"
initial-cluster: "node1=http://192.168.1.101:2380,node2=http://192.168.1.102:2380,node3=http://192.168.1.103:2380"
initial-cluster-state: "new"
initial-advertise-peer-urls: http://192.168.1.101:2380
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://0.0.0.0:2379
advertise-client-urls: http://192.168.1.101:2379
```

Bước 2: Tạo File Cấu Hình Cho Node 2 (`/etc/etcd/etcd.conf.yml`)

```
yaml
CopyEdit
name: node2
data-dir: /var/lib/etcd
initial-cluster-token: "etcd-cluster-1"
initial-cluster: "node1=http://192.168.1.101:2380,node2=http://192.168.1.102:2380,node3=http://192.168.1.103:2380"
initial-cluster-state: "new"
initial-advertise-peer-urls: http://192.168.1.102:2380
```

```
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://0.0.0.0:2379
advertise-client-urls: http://192.168.1.102:2379
```

Bước 3: Tạo File Cấu Hình Cho Node 3 (`/etc/etcd/etcd.conf.yml`)

```
yaml
CopyEdit
name: node3
data-dir: /var/lib/etcd
initial-cluster-token: "etcd-cluster-1"
initial-cluster: "node1=http://192.168.1.101:2380,node2=http://192.168.1.102:2380,node3=http://192.16
8.1.103:2380"
initial-cluster-state: "new"
initial-advertise-peer-urls: http://192.168.1.103:2380
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://0.0.0.0:2379
advertise-client-urls: http://192.168.1.103:2379
```

Bước 4: Khởi động etcd trên từng node

```
etcd --config-file=/etc/etcd/etcd.conf.yml
```

▼ **Dựng join lần lượt (Dynamic Bootstrapping - Member Add) - KHUYẾN KHÍCH DÙNG ĐỂ DỄ DÀNG SCALE UP)**

💡 **Mô tả:**

- Bắt đầu bằng một node **leader**, sau đó **dần dần thêm từng node một** vào cluster.
- Dùng lệnh `etcdctl member add` để thêm các node mới.

Thực hiện

Bước 1: Tạo và Khởi động Node Đầu Tiên (Leader)

File Cấu Hình (`/etc/etcd/etcd.conf.yml`) cho Node 1

```
yaml
CopyEdit
name: node1
data-dir: /var/lib/etcd
initial-cluster-token: "etcd-cluster-1"
initial-cluster-state: "new"
initial-advertise-peer-urls: http://192.168.1.101:2380
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://0.0.0.0:2379
advertise-client-urls: http://192.168.1.101:2379
```

Khởi động node 1

```
sh
CopyEdit
etcd --config-file=/etc/etcd/etcd.conf.yml
```

🔥 **Kết quả:** Node 1 khởi chạy và trở thành **leader** của cụm.

Bước 2: Thêm Node 2 Vào Cluster

Trên Node 1, chạy lệnh sau để thêm node 2

```
sh
CopyEdit
etcdctl --endpoints=http://192.168.1.101:2379 member add node2 --peer-urls=http://192.168.1.102:23
80
```

Lệnh này sẽ trả về thông tin cần thiết để cấu hình Node 2.

File Cấu Hình (`/etc/etcd/etcd.conf.yml`) cho Node 2

```
yaml
CopyEdit
name: node2
data-dir: /var/lib/etcd
initial-cluster-token: "etcd-cluster-1"
initial-cluster: "node1=http://192.168.1.101:2380,node2=http://192.168.1.102:2380"
initial-cluster-state: "existing"
initial-advertise-peer-urls: http://192.168.1.102:2380
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://0.0.0.0:2379
advertise-client-urls: http://192.168.1.102:2379
```

Khởi động Node 2

```
sh
CopyEdit
etcd --config-file=/etc/etcd/etcd.conf.yml
```

🔥 **Kết quả:** Node 2 tham gia vào cluster mà không cần dựng lại toàn bộ.

Bước 3: Thêm Node 3 Vào Cluster

Trên Node 1 hoặc Node 2, chạy lệnh sau để thêm node 3

```
sh
CopyEdit
etcdctl --endpoints=http://192.168.1.101:2379 member add node3 --peer-urls=http://192.168.1.103:23
80
```

File Cấu Hình (`/etc/etcd/etcd.conf.yml`) cho Node 3

```
yaml
CopyEdit
name: node3
data-dir: /var/lib/etcd
initial-cluster-token: "etcd-cluster-1"
initial-cluster: "node1=http://192.168.1.101:2380,node2=http://192.168.1.102:2380,node3=http://192.168.1.103:2380"
initial-cluster-state: "existing"
initial-advertise-peer-urls: http://192.168.1.103:2380
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://0.0.0.0:2379
advertise-client-urls: http://192.168.1.103:2379
```

Khởi động Node 3

```
sh
CopyEdit
etcd --config-file=/etc/etcd/etcd.conf.yml
```

🔥 **Kết quả:** Node 3 tham gia cluster mà không cần dựng lại từ đầu.

▼ So sánh file yaml

Thực hiện trên server 1

Cấu hình etcd cluster

Mở file cấu hình etcd

```
# vi /etc/etcd/etcd.conf.yaml
```

Với nội dung sau

```
name: 'server-1'
initial-cluster-token: postgresql_cluster_devopseduvn
initial-cluster-state: new
initial-cluster: server-1=http://192.168.213.11:2380
data-dir: /var/lib/etcd # nên tạo disk riêng để lưu trữ
initial-advertise-peer-urls: http://192.168.213.11:2380
listen-peer-urls: http://192.168.213.11:2380
advertise-client-urls: http://192.168.213.11:2379
listen-client-urls: http://192.168.213.11:2379
```

Khởi động dịch vụ etcd

```
# systemctl start etcd && systemctl enable etcd
```

Kiểm tra etcd cluster

```
# etcdctl --endpoints=http://192.168.213.11:2379 member list
```

Join etcd server 2 vào server 1

```
# etcdctl --endpoints=http://192.168.1.11:2379 member add server-2 --peer-urls=http://192.168.213.12:2380
```

Thực hiện trên server 2

Cấu hình etcd cluster

Mở file cấu hình etcd

```
# vi /etc/etcd/etcd.conf.yaml
```

Với nội dung sau

```
name: 'server-2'
initial-cluster-token: postgresql_cluster_devopseduvn
initial-cluster-state: existing
initial-cluster: server-1=http://192.168.213.11:2380,server-2=http://192.168.213.12:2380
data-dir: /var/lib/etcd # nên tạo disk riêng để lưu trữ
initial-advertise-peer-urls: http://192.168.213.12:2380
listen-peer-urls: http://192.168.213.12:2380
advertise-client-urls: http://192.168.213.12:2379
listen-client-urls: http://192.168.213.12:2379
```

Khởi động dịch vụ etcd

```
# systemctl start etcd && systemctl enable etcd
```

Thực hiện trên server 1

Cấu hình etcd cluster

Kiểm tra etcd cluster

```
# etcdctl --endpoints=http://192.168.213.11:2379 member list
```

Join etcd server 3 vào server 1

```
# etcdctl --endpoints=http://192.168.1.11:2379 member add server-3 --peer-urls=http://192.168.213.13:2380
```

Thực hiện trên server 3

Cấu hình etcd cluster

Mở file cấu hình etcd

```
# vi /etc/etcd/etcd.conf.yaml
```

Với nội dung sau

```
name: 'server-3'
initial-cluster-token: postgresql_cluster_devopseduvn
initial-cluster-state: existing
initial-cluster: server-1=http://192.168.213.11:2380,server-2=http://192.168.213.12:2380,server-3=http://192.16
8.213.13:2380
```

```
data-dir: /var/lib/etcd # nên tạo disk riêng để lưu trữ
initial-advertise-peer-urls: http://192.168.213.13:2380
listen-peer-urls: http://192.168.213.13:2380
advertise-client-urls: http://192.168.213.13:2379
listen-client-urls: http://192.168.213.13:2379
```

Khởi động dịch vụ etcd

```
# systemctl start etcd && systemctl enable etcd
```

Thực hiện trên server 1

Cấu hình etcd cluster

Kiểm tra etcd cluster

```
# etcdctl --endpoints=http://192.168.213.11:2379 member list
```

▼ 3. Cấu hình Patronic

Thực hiện trên cả 3 servers

Cấu hình patroni

Thêm các biến cần thiết

```
# export NODE_NAME=`hostname -f | cut -d. -f1`
# export NODE_IP=`ip addr show ens33 | awk '/inet / {print $2}' | cut -d/ -f1`
# DATA_DIR="/var/lib/postgresql/15/main"
# PG_BIN_DIR="/usr/lib/postgresql/15/bin"
# NAMESPACE="devopseduvn_percona"
# SCOPE="devopseduvn_cluster"
```

Tạo thư mục chứa password

```
# mkdir /opt/secretpg
# chown -R postgres. /opt/secretpg
# chmod -R 700 /opt/secretpg
```

Tạo file cấu hình patroni /etc/patroni/patroni.yml tham khảo: tại đây

Khởi động lại dịch vụ patroni

```
# systemctl restart patroni && systemctl enable patroni
```

Kiểm tra cluster

```
# patronictl -c /etc/patroni/patroni.yml list $SCOPE
```

▼ 4. CẤU HÌNH HAPROXY

Cấu hình HAproxy

```
# apt install -y haproxy
```

Backup và tạo file cấu hình mới

```
# mv /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.org
# vi /etc/haproxy/haproxy.cfg
```

Với cấu hình sau

```
global
    maxconn 100

defaults
    log global
    mode tcp
    retries 2
    timeout client 30m
    timeout connect 4s
    timeout server 30m
    timeout check 5s

listen stats
    mode http
    bind *:7000
    stats enable
    stats uri /

listen primary
    bind *:5000
    option httpchk /primary
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server node1 192.168.213.11:5432 maxconn 100 check port 8008
    server node2 192.168.213.12:5432 maxconn 100 check port 8008
    server node3 192.168.213.13:5432 maxconn 100 check port 8008

listen standbys
    balance roundrobin
    bind *:5001
    option httpchk /replica
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server node1 192.168.213.11:5432 maxconn 100 check port 8008
    server node2 192.168.213.12:5432 maxconn 100 check port 8008
    server node3 192.168.213.13:5432 maxconn 100 check port 8008
```

Khởi động lại HAproxy

```
# systemctl restart haproxy
```

▼ ELASTIC SEARCH

▼ 1. CÀI ĐẶT ELASTICSEARCH

▼ Lỗi khi start elasticsearch

- Lỗi không đủ ram để chạy ⇒ fix bằng cách giảm heap size (yêu cầu bộ nhớ của elasticsearch

Chỉnh sửa file `jvm.options`:

bash Copy Edit

```bash
vi /etc/elasticsearch/jvm.options
```

📌 **Tìm và chỉnh sửa dòng sau (giảm heap size nếu RAM thấp):**

bash Copy Edit

```bash
-Xms512m
-Xmx512m
```

✔ Nếu RAM dưới **4GB**, đặt `Xms512m` và `Xmx512m`.
✔ Nếu RAM từ **4GB - 8GB**, đặt `Xms2g` và `Xmx2g`.

Thực hiện trên cả 3 servers

Cài đặt ElasticSearch

Tải và lưu khóa GPG dùng để xác thực các gói phần mềm Elasticsearch

```
# wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
```

Cài đặt các gói công cụ cần thiết

```
# sudo apt-get install apt-transport-https net-tools
```

Thêm Elasticsearch 8.x vào danh sách các nguồn gói của hệ thống để cài bằng lệnh apt

```
# echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic-8.x.list
```

Cài đặt ElasticSearch

```
# sudo apt-get update && sudo apt-get install elasticsearch
```

▼ 2. Cấu hình ES server 1 thành master

Thực hiện trên server 1

Cấu hình ES server 1 thành master của ES cluster

Mở file cấu hình elasticsearch và chỉnh sửa

```
# vi /etc/elasticsearch/elasticsearch.yml
```

Chỉnh sửa những dòng sau đây (lưu ý phải xem video bài giảng)

```
cluster.name: es-cluster-devopseduvn
node.name: server-1
network.host: 192.168.213.11
```

```
http.port: 9200
transport.host: 0.0.0.0
```

Khởi động ES để tạo cụm

```
# systemctl start elasticsearch
```

Di chuyển đến thư mục làm việc bashscript của ES

```
# cd /usr/share/elasticsearch/bin/
```

Tiến hành reset lại mật khẩu

```
# ./elasticsearch-reset-password -i -u elastic
>> 123456
```

Kiểm tra trạng thái bằng câu lệnh

```
curl -k -u elastic:123456 https://192.168.213.11:9200/
# -k : Bỏ qua lỗi ssl
```

thông tin cluster được hiện ra

```
root@k8s-master-1:/usr/share/elasticsearch/bin# curl -k -u elastic:elastic https://192.168.189.111:9200/
{
  "name" : "k8s-master-1",
  "cluster_name" : "es-cluster-devopseduvn",
  "cluster_uuid" : "bwn8DaPbTbGEN-jYKI4SvQ",
  "version" : {
    "number" : "8.17.2",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "747663ddda3421467150de0e4301e8d4bc636b0c",
    "build_date" : "2025-02-05T22:10:57.067596412Z",
    "build_snapshot" : false,
    "lucene_version" : "9.12.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Kiểm tra trạng thái thông tin của ES cluster bằng lệnh

```
curl -k -u elastic:123456 https://192.168.213.11:9200/_cluster/health?pretty
# -k : Bỏ qua lỗi ssl
```

```
root@k8s-master-1:/usr/share/elasticsearch/bin# curl -k -u elastic:elastic https://192.168.189.111:9200/_cluster/health?pretty
{
  "cluster_name" : "es-cluster-devopseduvn",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 3,
  "active_shards" : 3,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "unassigned_primary_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

Khởi tạo token để cho server 2 và server 3 join vào ES cluster

```
./elasticsearch-create-enrollment-token -s node
>> <token>
```

▼ 3. **Join ES server 2 vào ES cluster (đã thiết lập ở server 1)**

Thực hiện trên server 2 và server 3

Di chuyển đến thư mục làm việc bashscript của ES

```
# cd /usr/share/elasticsearch/bin/
```

Xác nhận cấu hình token từ ES server 1

```
./elasticsearch-reconfigure-node --enrollment-token <token
```

/

▼ 4. **Thay đổi config discovery.seed_hosts**

Thực hiện trên server 1

Thay đổi phần discovery.seed_hosts

Mở lại file cấu hình ES và tìm đến phần Discovery

```
vi /etc/elasticsearch/elasticsearch.yml
```

Sửa cấu hình thành như sau (tương ứng địa chỉ IP của bạn)

```
discovery.seed_hosts:
  - 192.168.213.11:9300
  - 192.168.213.12:9300
  - 192.168.213.13:9300
```

Khởi động lại ES

```
# systemctl restart elasticsearch
```

▼ 5. Cấu hình thiết lập cụm cho server2 và server 3

Thực hiện trên server 2 và server 3

Cấu hình thiết lập cụm

Mở file cấu hình ES

```
# vi /etc/elasticsearch/elasticsearch.yml
```

Thay đổi các thông tin tương ứng server 2 và server 3 như sau

```
cluster.name: es-cluster-devopseduvn
node.name: server-2 hoặc server-3
network.host: 192.168.213.12 hoặc 192.168.213.13
http.port: 9200
discovery.seed_hosts:
```

```
  - 192.168.213.11:9300
  - 192.168.213.12:9300
  - 192.168.213.13:9300
```

Khởi động lại ES

```
# systemctl restart elasticsearch
```

▼ 6. Thiết lập Kibana

Thực hiện trên server 1

Thiết lập Kibana

Cài đặt Kibana

```
# sudo apt-get update && sudo apt-get install kibana
```

Tạo thư mục certs cho kibana

```
# mkdir /etc/kibana/certs
```

Tạo crt và key để truy cập kibana với https

```
# openssl req -newkey rsa:4096 -nodes -sha256 -keyout /etc/kibana/certs/domain.key -subj "/CN=192.168.213.11" -addext "subjectAltName = DNS:192.168.213.11,IP:192.168.213.11" -x509 -days 365 -out /etc/kibana/certs/domain.crt
```

Copy CA của ES sang Kibana

```
# cp /etc/elasticsearch/certs/http_ca.crt /etc/kibana/certs/
```

Mở file cấu hình Kibana

```
# vi /etc/kibana/kibana.yml
```

Sửa các nội dung sau

```
server.port: 5601
server.host: "0.0.0.0"
server.publicBaseUrl: "http://192.168.213.11:5601/"
server.ssl.enabled: true
server.ssl.certificate: /etc/kibana/certs/domain.crt
server.ssl.key: /etc/kibana/certs/domain.key
elasticsearch.hosts:
  - https://192.168.213.11:9200
  - https://192.168.213.12:9200
  - https://192.168.213.13:9200
elasticsearch.ssl.certificate: /etc/kibana/certs/domain.crt
elasticsearch.ssl.key: /etc/kibana/certs/domain.key
elasticsearch.ssl.certificateAuthorities: [ "/etc/kibana/certs/http_ca.crt" ]
elasticsearch.ssl.verificationMode: full
```

Tạo kibana token

```
curl -X POST -u elastic:123456 http://192.168.213.11:9200/_security/service/elastic/kibana/credential/token/kibana_token
```

Di chuyển đến thư mục làm việc bashscript của kibana

```
# cd /usr/share/kibana/bin/
```

Khởi chạy lệnh add elasticsearch.serviceAccountToken

```
# ./kibana-keystore add elasticsearch.serviceAccountToken
```

Thay đổi toàn bộ quyền thư mục /etc/kibana cho user kibana

```
# chown -R kibana:kibana /etc/kibana/
```

Tiến hành khởi chạy Kibana

```
# systemctl start kibana
```

▼ REDIS

▼ LƯU Ý KHI CÀI 1 CÔNG CỤ

1. Add repo của công cụ
2. Viết file repo riêng
3. Install trực tiếp trên OS nếu có sẵn

▼ 1. CÀI ĐẶT REDIS

Thực hiện trên cả 3 servers

Cài đặt Redis

Cài đặt redis và redis-sentinel

```
sudo apt install -y lsb-release curl gpg
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/deb $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/redis.list
sudo apt update -y
sudo apt install -y redis redis-sentinel
```

▼ 2. CẤU HÌNH REDIS CLUSTER

Thực hiện trên server 1

Cấu hình redis cluster

Mở file cấu hình

```
# vi /etc/redis/redis.conf
```

Chỉnh sửa các nội dung sau

```
bind 0.0.0.0 # Cho phép Redis chấp nhận kết nối từ tất cả các địa chỉ IP
masterauth devopseduvn  # Mật khẩu xác thực khi (Slave) kết nối đến Master
requirepass devopseduvn # Mật khẩu bắt buộc khi client kết nối đến Redis
```

Thực hiện trên server 2 và server 3

Cấu hình redis cluster

Mở file cấu hình

```
# vi /etc/redis/redis.conf
```

Chỉnh sửa các nội dung sau

```
bind 0.0.0.0
masterauth devopseduvn
requirepass devopseduvn
replicaof 192.168.213.11 6379  # Cấu hình Redis chạy ở chế độ Replica (Slav
#e), đồng bộ dữ liệu từ Master có IP 192.168.213.11 trên cổng 6379
```

Thực hiện trên cả 3 servers

Cấu hình redis cluster

Khởi động lại redis

```
# systemctl restart redis-server
```

▼ 3. KIỂM TRA HA

Thực hiện trên server 1

Kiểm tra HA

Truy cập vào môi trường redis

```
# redis-cli
```

Tiến hành thêm dữ liệu

```
>> auth devopseduvn
>> SET devopseduvn_popular_series '[{"name": "DevOps for Freshers"}, {"name": "Pipeline DevSecOps"},
{"name": "HA tools"}]'
>> GET devopseduvn_popular_series
```

Thực hiện trên server 2 và server 3

Kiểm tra HA

Truy cập vào môi trường redis

```
# redis-cli
```

Tiến hành kiểm tra dữ liệu

```
>> auth devopseduvn
>> GET devopseduvn_popular_series
```

▼ 4. CẤU HÌNH SENTINEL

Thực hiện trên cả 3 servers

Cấu hình sentinel

Mở file cấu hình

```
# vi /etc/redis/sentinel.conf
```

Chỉnh sửa các thông tin sau

```
protected-mode no  # Cho phép kết nối từ bên ngoài
sentinel monitor mymaster 192.168.213.11 6379 2 # Giám sát Redis Master, cần #2 Sentinel đồng ý để failove
sentinel auth-pass mymaster devopseduvn #Mật khẩu để Sentinel xác thực master
sentinel down-after-milliseconds mymaster 5000 # Đánh dấu Master là "down" nế#u không phản hồi trong
5 giây
sentinel failover-timeout mymaster 60000 # Thời gian tối đa (60 giây) để hoàn # thành failover
sentinel parallel-syncs mymaster 1 # Số Replica có thể đồng bộ cùng lúc khi f#ailover
```

Khởi động lại sentinel

```
# systemctl restart redis-sentinel
```

Thực hiện trên server 1

Kiểm tra failover

Truy cập vào môi trường redis

```
# redis-cli -p 26379 sentinel masters
# redis-cli -p 26379 sentinel slaves mymaster
# redis-cli -p 26379 sentinel sentinels mymaster
# systemctl stop redis-server
```