



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Docker, K8s, Serverless

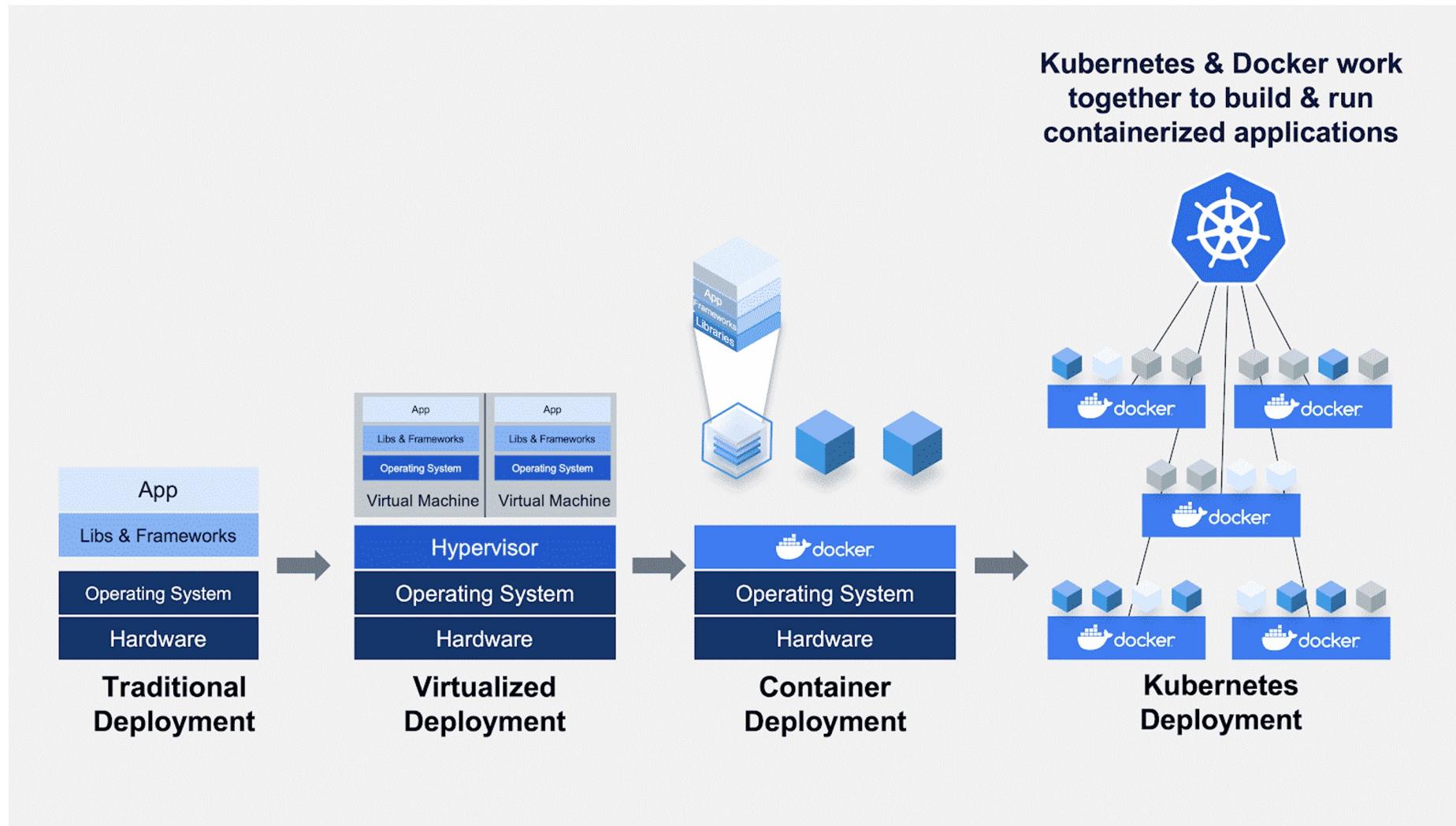
Nội dung

1. Triển khai ứng dụng web
2. Docker
3. K8s
4. Kiến trúc serverless

Nội dung

1. Triển khai ứng dụng web
2. Docker
3. K8s
4. Kiến trúc serverless

Triển khai ứng dụng web



Nội dung

1. Triển khai ứng dụng web
2. Docker
3. K8s
4. Kiến trúc serverless

Nội dung

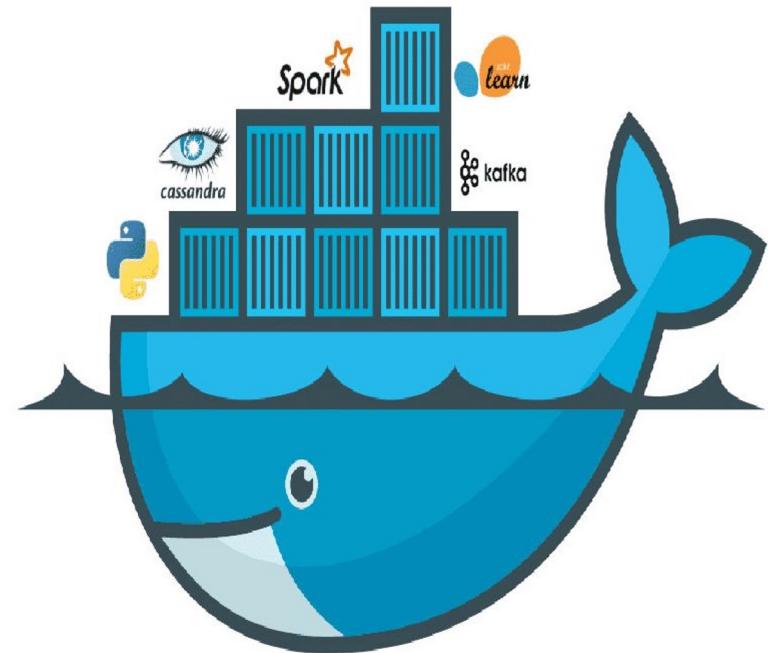
- 2.1. Giới thiệu chung
- 2.2. Các thao tác cơ bản
- 2.3. Volume
- 2.4. Network

Hypervisor vs Containerization

	Hypervisor	Containerization
Khái niệm	Công nghệ ảo hóa ở tầng phần cứng: Vmware, Bluestack,...	Công nghệ ảo hóa ở tầng hệ điều hành
Ưu điểm	Cài được nhiều hệ điều hành trên 1 máy, sử dụng dễ	<ul style="list-style-type: none">- Cài được nhiều hệ điều hành trên cùng 1 máy, dùng chung tài nguyên với máy host, dùng xong tự giải phóng tài nguyên- Thời gian khởi tạo nhanh
Nhược điểm	<ul style="list-style-type: none">- Lãng phí tài nguyên- Thời gian khởi tạo lâu	<ul style="list-style-type: none">- Bảo mật- Khó sử dụng

Docker

- Docker là một open platform cung cấp cho chúng ta các công cụ, service giúp chúng ta đóng gói và chạy chương trình của mình trên các môi trường khác nhau một cách nhanh, gọn, nhẹ.
- Các khái niệm cơ bản:
 - Dockerfile
 - Image
 - Container



Dockerfile

- Là một tập các câu lệnh để tạo ra docker image
- Một số lệnh cơ bản:

FROM <image gốc> : <phiên bản>: Đây là lệnh bắt buộc trong Dockerfile nhằm để chỉ image này được build từ đầu ra (từ image gốc nào).

RUN <câu lệnh>: Dùng để thực thi một câu lệnh nào đó trong quá trình build image, có thể có nhiều lệnh RUN trong một Dockerfile

EXPOSE: lệnh này để cho docker biết container sẽ lắng nghe trên các cổng mạng được chỉ định khi chạy

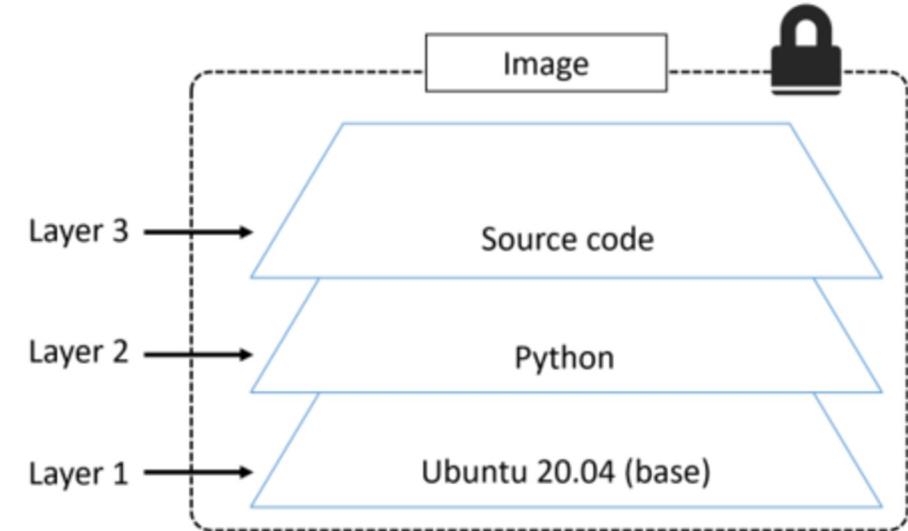
ADD <src> <dest>: dùng để copy một thư mục ở local hoặc một remote file nào đó vào một thư mục nào đó trong container

Dockerfile - myhttpd:0.1

```
1 # A simple web app served by httpd
2 FROM httpd:2.4
3
4 LABEL AUTHOR=user@example.com
5
6 LABEL VERSION=0.1
7
8 # COPY mypage.html /usr/local/apache2/htdocs/mypage.html
9 # WORKDIR /usr/local/apache2
10
11 COPY mypage.html htdocs/mypage.html
```

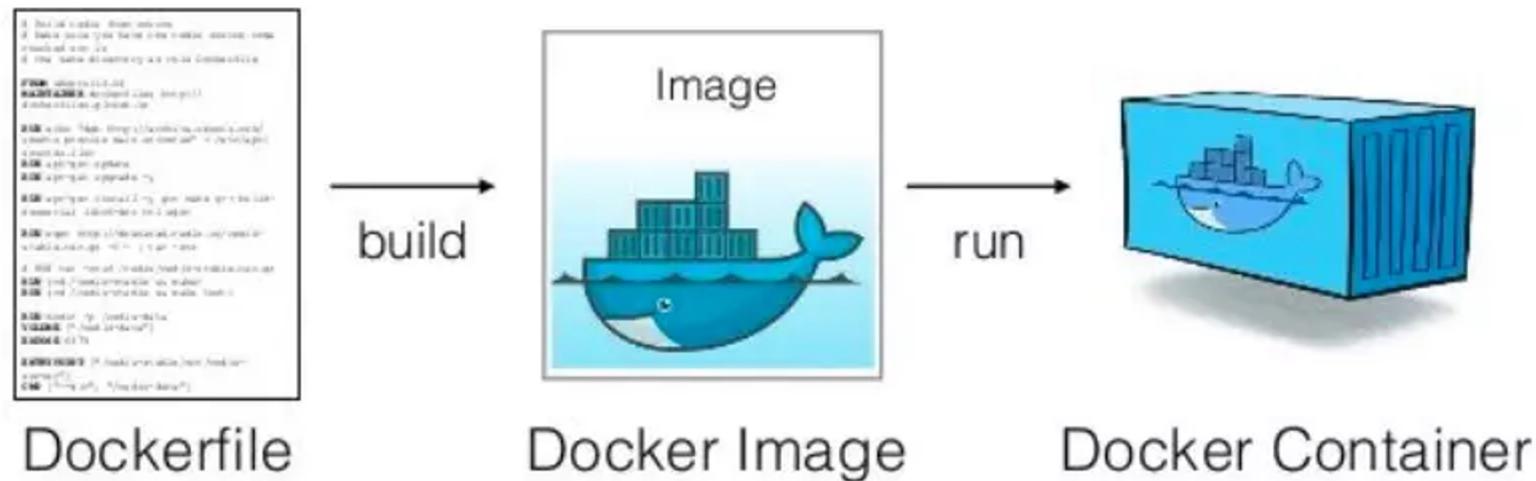
Docker Image

- Khuôn mẫu để tạo ra **container**, đóng gói mọi thứ cần thiết để 1 ứng dụng có thể chạy
- Có thể tự tạo hoặc có thể lấy từ **registry**
- Được tạo thành từ các lớp (layer) được định nghĩa trong Dockerfile



Docker container

- Là ứng dụng được tạo ra khi chạy image.
- Một image có thể tạo ra nhiều container
- Có thể coi image là class còn container là instance được tạo ra giống trong OOP



Nội dung

- 2.1. Giới thiệu chung
- 2.2. Các thao tác cơ bản
- 2.3. Hướng dẫn viết Dockerfile
- 2.4. Volume
- 2.5. Network

Thao tác cơ bản với Docker

Thao tác với image

- 1 IMAGE đặc trưng bởi tên + phiên bản hoặc ID ([name]:[tag] && [ID]). Mọi [name]:[tag] đều có thể thay thế bằng [ID].

LỆNH	CHỨC NĂNG
Docker pull <image name>:<tag>	Tải 1 image từ registry về máy (mặc định tag latest, registry: dockerhub)
Docker rmi <image name>:<tag>	Xóa image trên máy hiện tại
Docker push <image name>:<tag>	Đưa image lên registry (dockerhub, local registry, ...)
Docker build -t <image name>:<tag> -f <Dockerfile> .	Build image với dockerfile

Thao tác cơ bản với Docker

Thao tác với container

- Có thể thao tác theo tên hoặc theo id của container

LỆNH	CHỨC NĂNG
Docker run <image name>:<tag>	Chạy container từ image
Docker stop <container id>	Dừng chạy container
Docker exec <container id>	Chạy lệnh trong container
Docker attach <container id>	Trở lại container đang chạy

Nội dung

- 2.1. Giới thiệu chung
- 2.2. Các thao tác cơ bản
- 2.3. Hướng dẫn viết Dockerfile**
- 2.4. Volume
- 2.5. Network

Hướng dẫn viết Dockerfile

- Lấy base image node
- Thư mục trong container sẽ là app
- Copy file package.json để tiến hành cài đặt trong container
- Copy tất cả file code vào thư mục app
- Mở 1 port 8000 trong container để bên ngoài có thể gọi vào
- Chạy code trong container bằng npm start

```
📦 Dockerfile > ...
1  FROM node:16
2
3  WORKDIR /app
4
5  COPY package*.json .
6
7  RUN npm install
8
9  COPY . .
10
11 EXPOSE 8000
12
13 ENTRYPOINT [ "npm" ]
14
15 CMD [ "start" ]
```



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



Bài tập

- Tạo image cho project backend của mình với ngôn ngữ nhóm chọn (php, java, ...)

Nội dung

- 2.1. Giới thiệu chung
- 2.2. Các thao tác cơ bản
- 2.3. Hướng dẫn viết Dockerfile
- 2.4. Volume**
- 2.5. Network

Chia sẻ dữ liệu với container

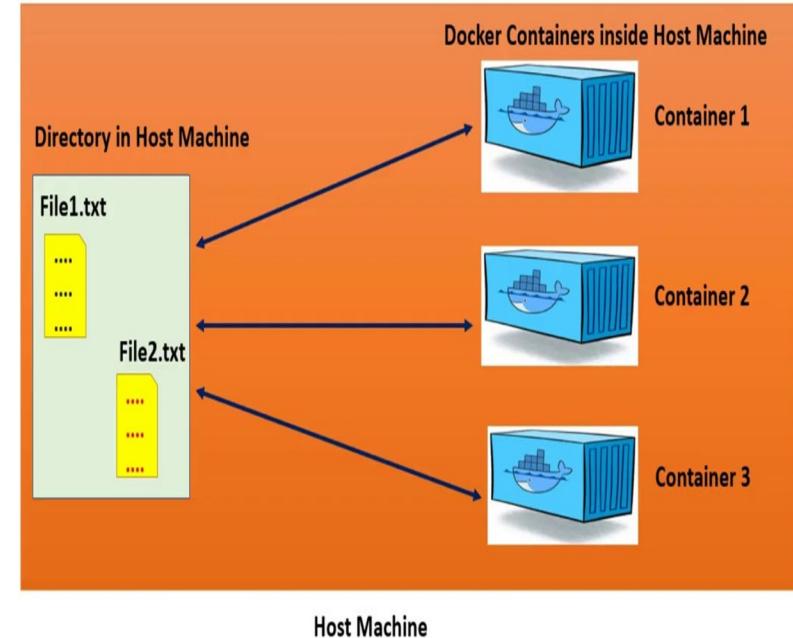
- Chia sẻ dữ liệu với máy host

```
docker run -i -v <path1>:<path2> [image_ID]
```

- Tham số `-v` : chỉ định tài nguyên
- `<path1>`: đường dẫn tài nguyên trong máy host
- `<path2>`: tài nguyên trên máy host được ánh xạ sang đường dẫn path2 trong container

-> Mọi dữ liệu được chỉnh sửa trên container hay trên máy host thì đều được thay đổi trên cả 2

Volumes



Chia sẻ dữ liệu với container

Tạo 1 container cùng được chia sẻ dữ liệu giống 1 container đang chạy

- Cú pháp: `docker run -it --name [container_2] --volumes-from [container_1] [image_ID]`
- Ví dụ: `docker run -it --name C2 --volumes-from C1 ubuntu:16.04`

-> Mọi dữ liệu được thay đổi thì đều thay đổi trên các bên

*Note: Tạo và quản lý ổ đĩa trong docker - <https://docs.docker.com/storage/volumes/#create-and-manage-volumes>

Nội dung

1. Triển khai ứng dụng web
2. Docker
3. K8s
4. Kiến trúc serverless

Nội dung

3.1. Giới thiệu

3.2. Kiến trúc

3.3. Các thành phần

3.4. Thực hành

K8s

APPLICATION



Lets say you have
created an application



DOCKER



And used **Docker containers**
to package the application



K8s

DEPLOYED



Say you have deployed on **3 different servers using Docker**



SCALING



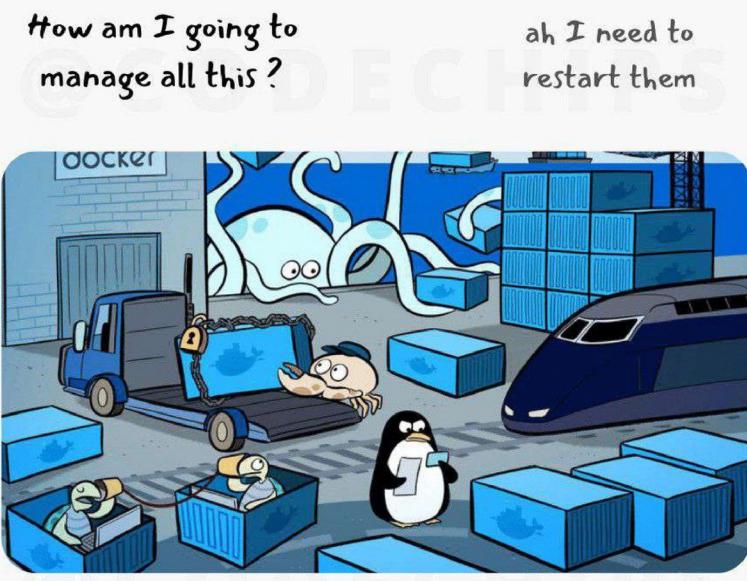
Now you need to scale up fast; how will you go from 3 servers to 40 servers that you may require?

How to decide which container should go where? Monitor all containers? & make sure they restart if they die?



K8s

OUT OF CONTROL



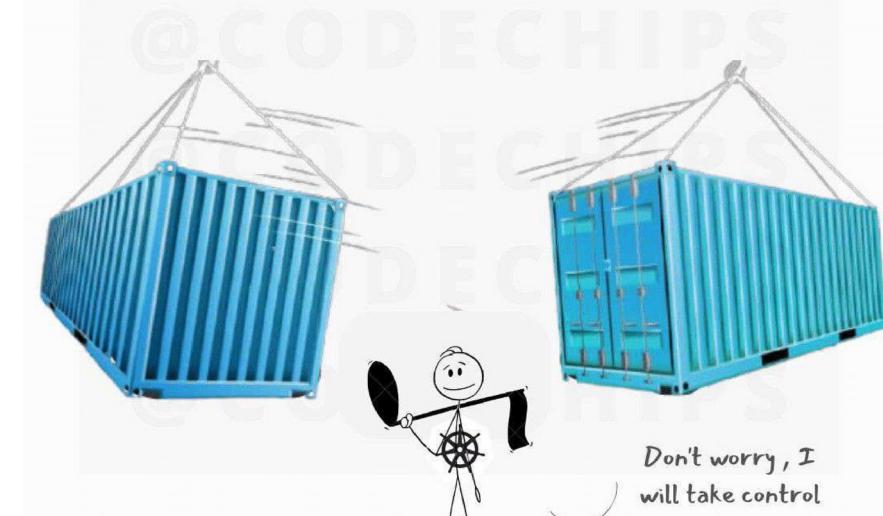
Wouldn't it be easier
if this behavior was
handled by a system?

KUBERNETES



This is where **Kubernetes** comes into play

Kubernetes (aka k8s or “kube”) is an open source **container orchestration** platform that automates deploying, managing, and scaling containerized applications.



@codechips

Cody Dev

codechipsig@gmail.com

Nội dung

3.1. Giới thiệu

3.2. Kiến trúc

3.3. Các thành phần

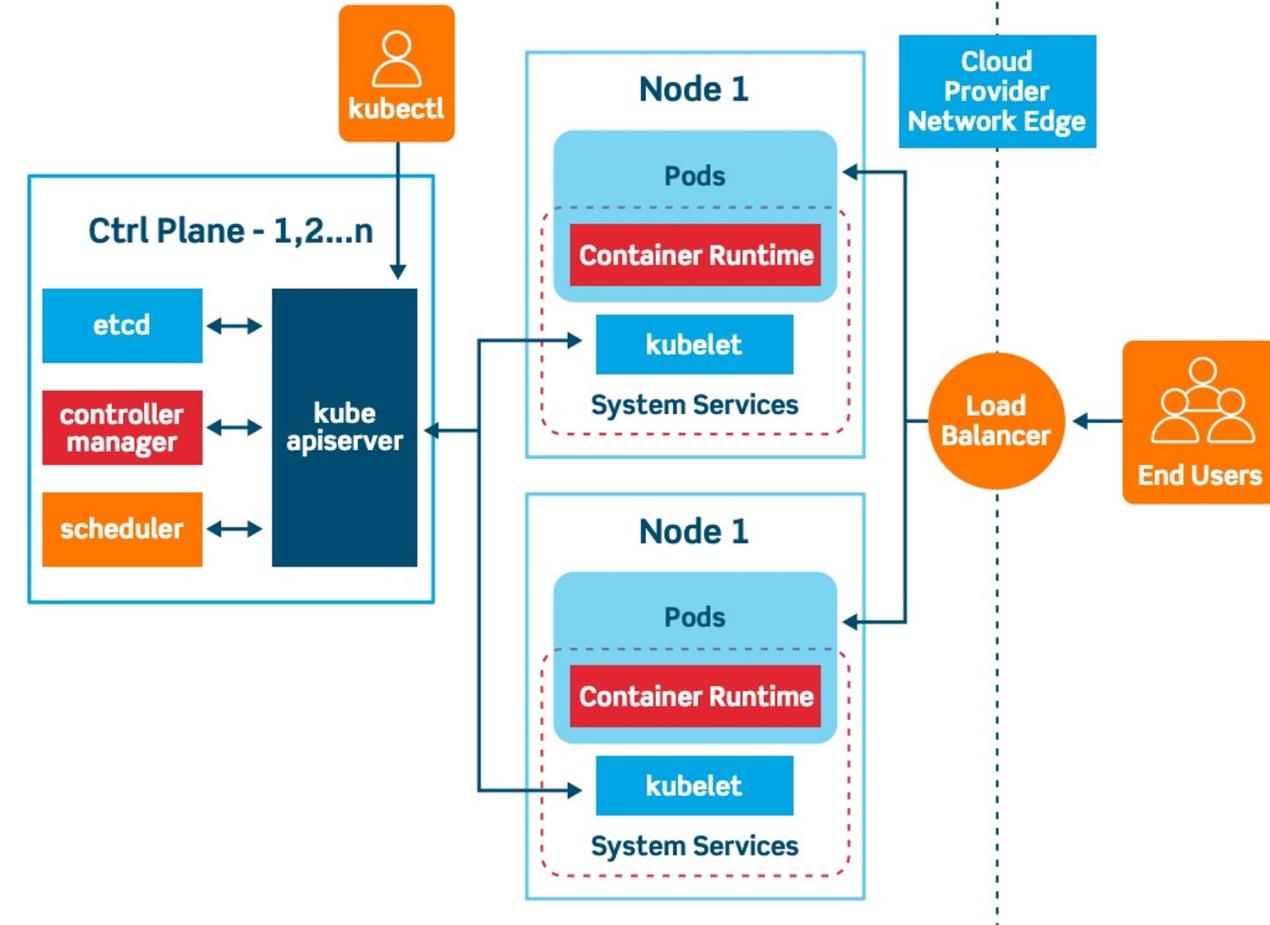
3.4. Thực hành

Kiến trúc

Kubernetes cluster (một cụm bao gồm một master và một hoặc nhiều worker) bao gồm 2 thành phần (component) chính:

Master node:

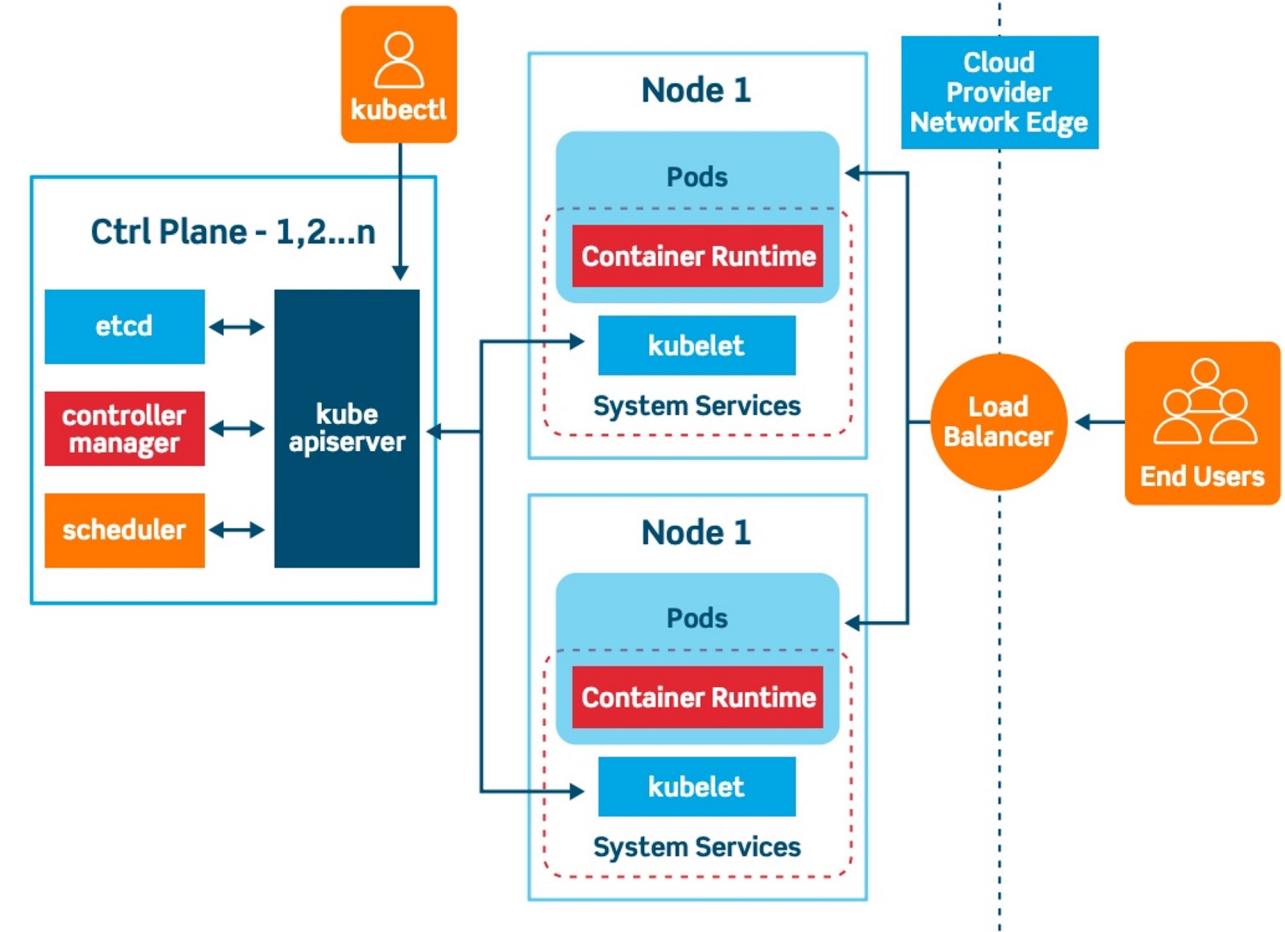
- Etcd
- API server
- Schedule
- Controller



Kiến trúc

Worker node:

- Container runtime (docker, rkt hoặc nền tảng khác): chạy container
- Kubelet: giao tiếp với API server và quản lý container trong một worker node
- Kubernetes Service Proxy (kube-proxy): quản lý network và traffic của các ứng dụng trong worker node



Nội dung

3.1. Giới thiệu

3.2. Kiến trúc

3.3. Các thành phần

3.4. Thực hành

Nội dung

- 3.3.1. Pod
- 3.3.2. Controller
- 3.3.3. Service
- 3.3.4. Deployment

Cài đặt môi trường

Trong phạm vi tìm hiểu chúng ta có thể sử dụng k3s

Cách cài đặt:

```
curl -sfL https://get.k3s.io | sh -
# Check for Ready node, takes ~30
seconds
k3s kubectl get node
```

Nội dung

- 3.3.1. Pod
- 3.3.2. Replicaset
- 3.3.3. Service
- 3.3.4. Deployment

Cách tạo pod trong k8s

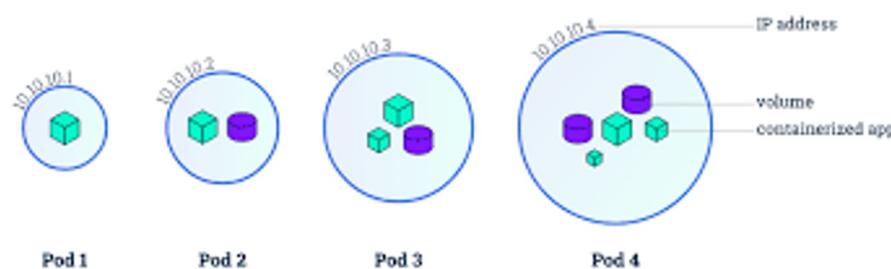
- Tạo 1 file yaml với nội dung như hình, có thể lấy luôn image vừa tạo phần docker
- Chạy lệnh

```
kubectl apply -f <filename>.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: <POD NAME>
spec:
  containers:
    - name: <container name>
      image: <image>
      ports:
        - containerPort: 80
```

Giới thiệu về k8s pod

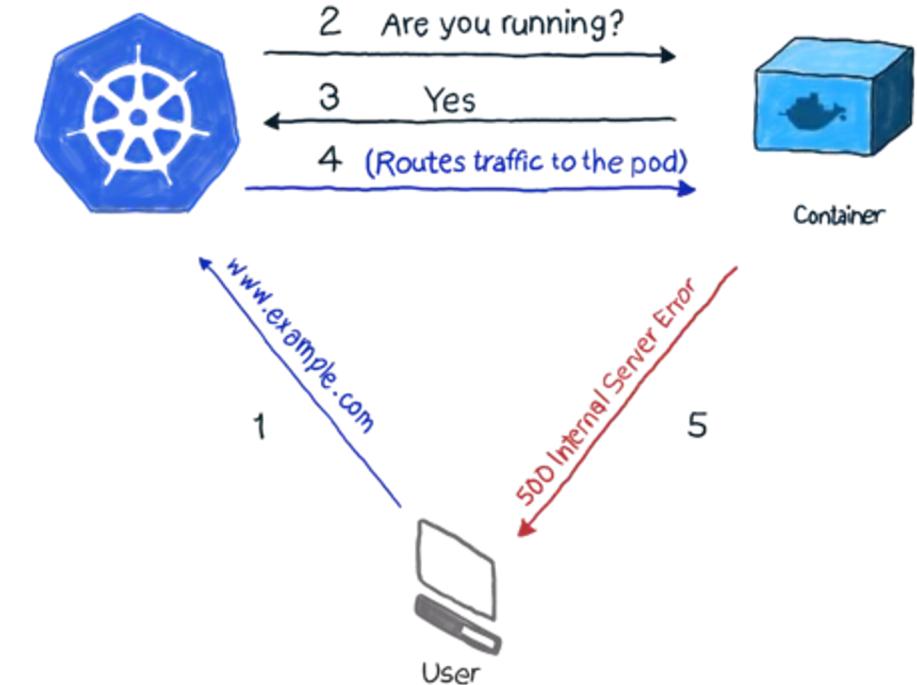
- Pod là thành phần cơ bản nhất để deploy và chạy một ứng dụng, được tạo và quản lý bởi kubernetes
- Pod được dùng để nhóm (group) và chạy một hoặc nhiều container lại với nhau trên cùng một worker node, những container trong một pod sẽ chia sẻ chung tài nguyên với nhau



K8s pod

Tại sao không chạy container trực tiếp mà phải qua pod ?

-> Pod giúp quản lý tài nguyên và trạng thái container tốt hơn



Cách tạo pod trong k8s

- Tạo 1 file yaml với nội dung như hình, có thể lấy luôn image vừa tạo phần docker
- Chạy lệnh

```
kubectl apply -f <filename>.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: <POD NAME>
spec:
  containers:
    - name: <container name>
      image: <image>
      ports:
        - containerPort: 80
```

Nội dung

3.3.1. Pod

3.3.2. Replicaset

3.3.3. Service

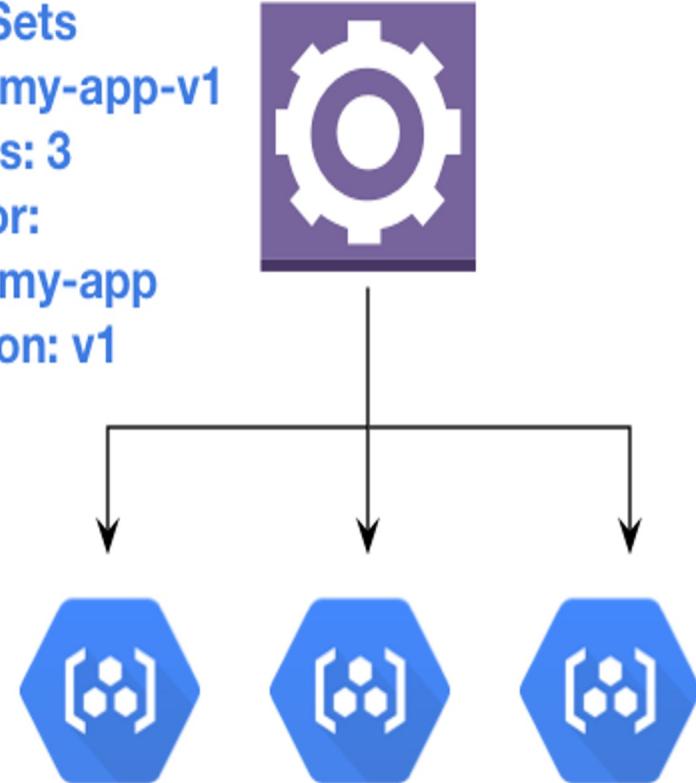
3.3.4. Deployment

Replica set

ReplicationSet là một resource mà sẽ tạo và quản lý pod, và chắc chắn là số lượng pod nó quản lý không thay đổi và tiếp tục chạy.

ReplicationControllers sẽ tạo số lượng pod bằng với số ta chỉ định ở thuộc tính replicas và quản lý pod thông qua labels của pod

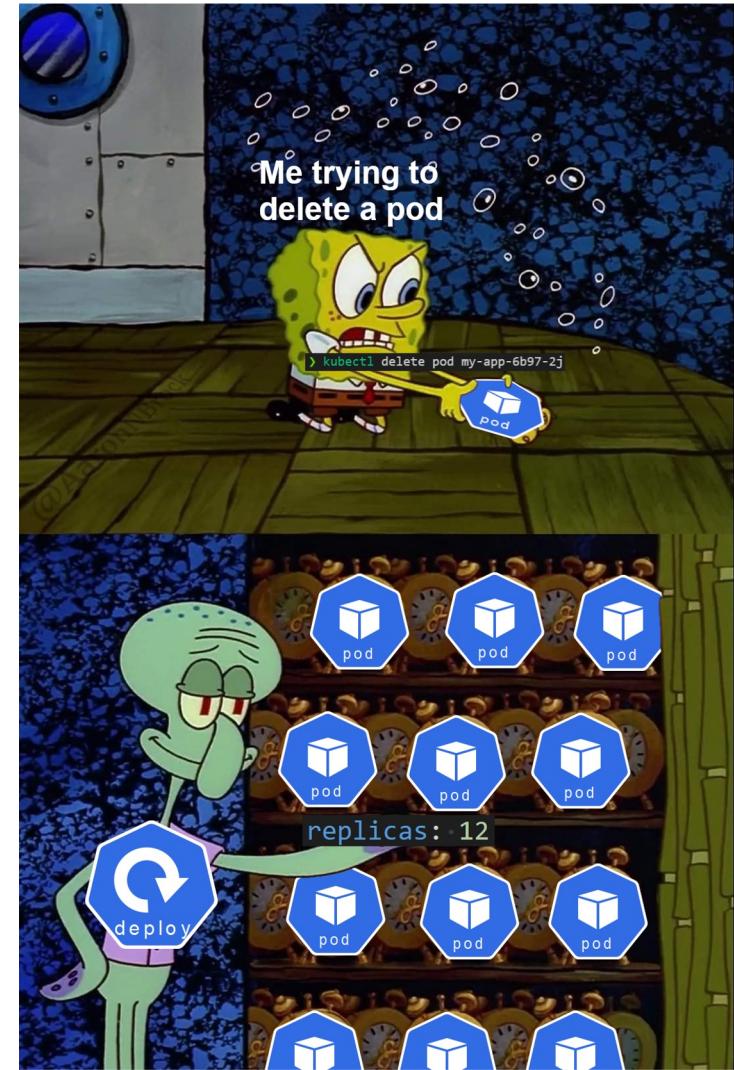
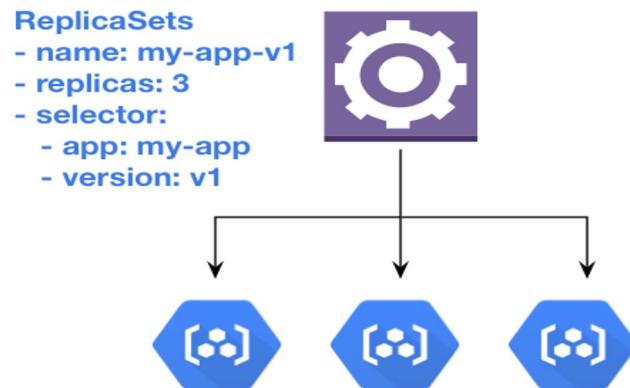
ReplicaSets
- name: my-app-v1
- replicas: 3
- selector:
 - app: my-app
 - version: v1



Replica set

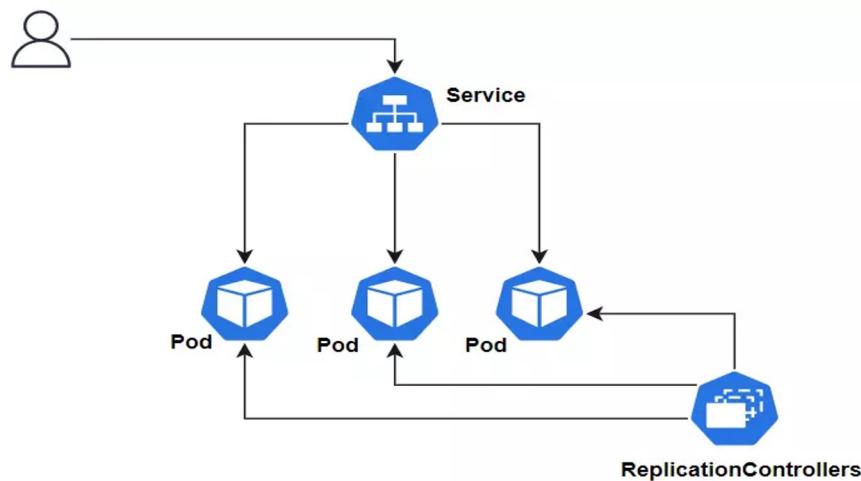
ReplicationSet là một resource mà sẽ tạo và quản lý pod, và chắc chắn là số lượng pod nó quản lý không thay đổi và tiếp tục chạy.

ReplicationControllers sẽ tạo số lượng pod bằng với số ta chỉ định ở thuộc tính replicas và quản lý pod thông qua labels của pod



Tại sao dùng Replica set

- Đảm bảo tính khả dụng cho ứng dụng:
-> Vì RC sẽ chắc chắn rằng số lượng pod mà nó tạo ra không thay đổi, nên ví dụ khi ta tạo một thằng RC với số lượng replicas = 1, RC sẽ tạo 1 pod và giám sát nó, khi một thằng worker node die, nếu pod của thằng RC quản lý có nằm trong worker node đó, thì lúc này thằng RC sẽ phát hiện ra là số lượng pod của nó bằng 0
- Tăng tốc độ của ứng dụng:



Tạo replicaset

- Chạy lệnh

```
kubectl apply -f <filename>.yaml
```

- Kiểm tra:

```
kubectl get rs
```

- Xem mô tả:

```
kubectl describe <rs name>
```

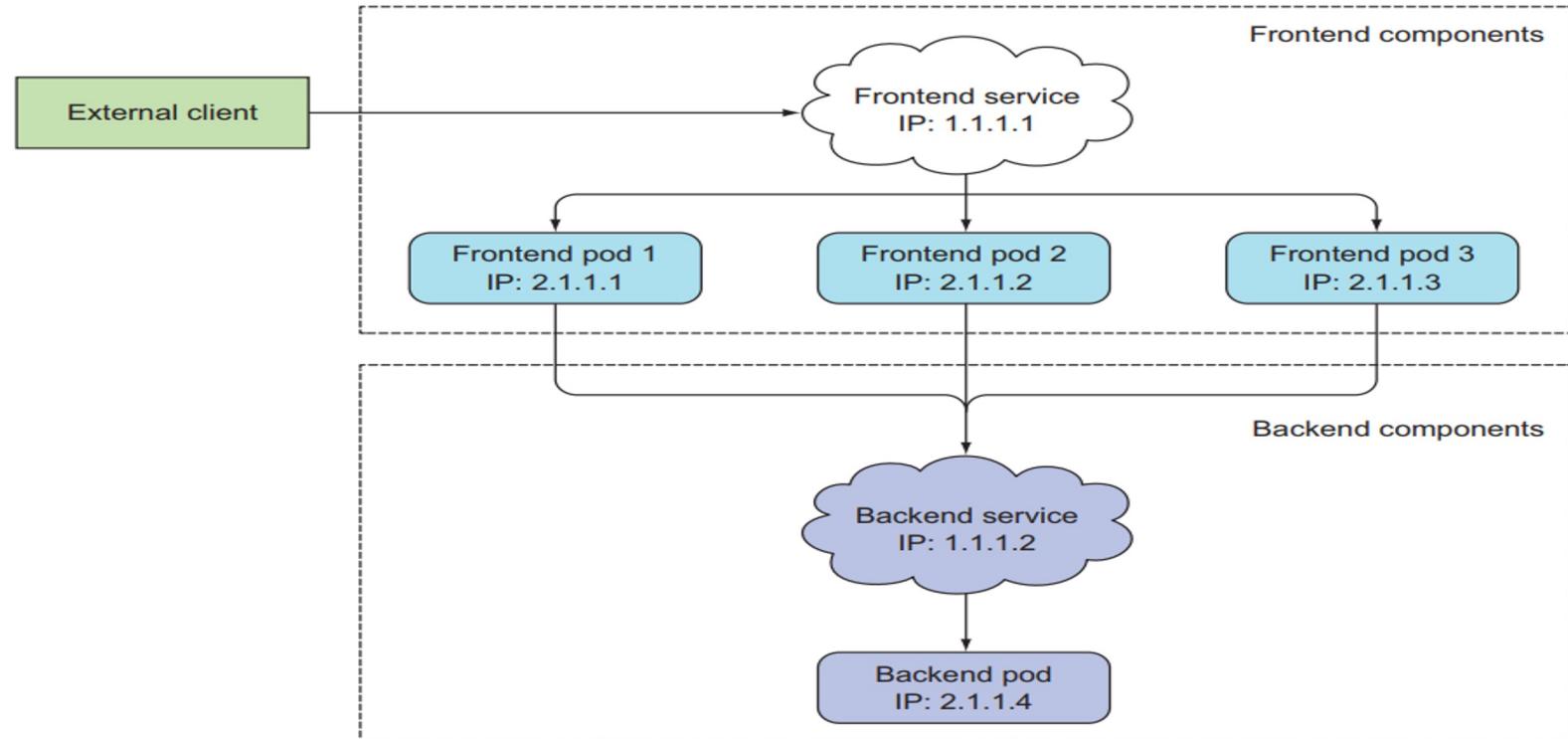
```
apiVersion: apps/v1 # change version API
kind: ReplicaSet # change resource name
metadata:
  name: <replica name>
spec:
  replicas: 2
  selector:
    matchLabels: # change here
      app: <pod labels>
  template:
    metadata:
      labels:
        app: <pod labels>
    spec:
      containers:
        - image: <image name>
          name: <container name>
      ports:
        - containerPort: 3000
```

Nội dung

- 3.3.1. Pod
- 3.3.2. Replicaset
- 3.3.3. Service**
- 3.3.4. Deployment

Service

Là một resource sẽ tạo ra một single, constant point của một nhóm Pod phía sau nó. Mỗi service sẽ có một địa chỉ IP và port không đổi, trừ khi ta xóa nó đi và tạo lại. Client sẽ mở connection tới service, và connection đó sẽ được dẫn tới một trong những Pod ở phía sau.



Service giải quyết vấn đề gì

- Mỗi pod có địa chỉ ip riêng tại sao không gọi luôn ip của pod
-> pod có thể bị thay đổi (kill, die,...)
- Ứng dụng chạy nhiều pod thì biết gọi vào pod nào ?
-> Service sẽ tạo ra một endpoint không đổi cho các Pod phía sau, client chỉ cần tương tác với endpoint này.

Các service

- ClusterIP: Đây là loại service sẽ tạo một IP và local DNS mà sẽ có thể truy cập ở bên trong cluster, không thể truy cập từ bên ngoài, được dùng chủ yếu cho các Pod ở bên trong cluster dễ dàng giao tiếp với nhau.
- Nodeport: Đây là cách đầu tiên để expose Pod cho client bên ngoài có thể truy cập vào được Pod bên trong cluster.
- LoadBaclancer: Khi bạn chạy kubernetes trên cloud, nó sẽ hỗ trợ LoadBalancer Service, nếu bạn chạy trên môi trường không có hỗ trợ LoadBalancer thì bạn không thể tạo được loại Service này
- ExternalName

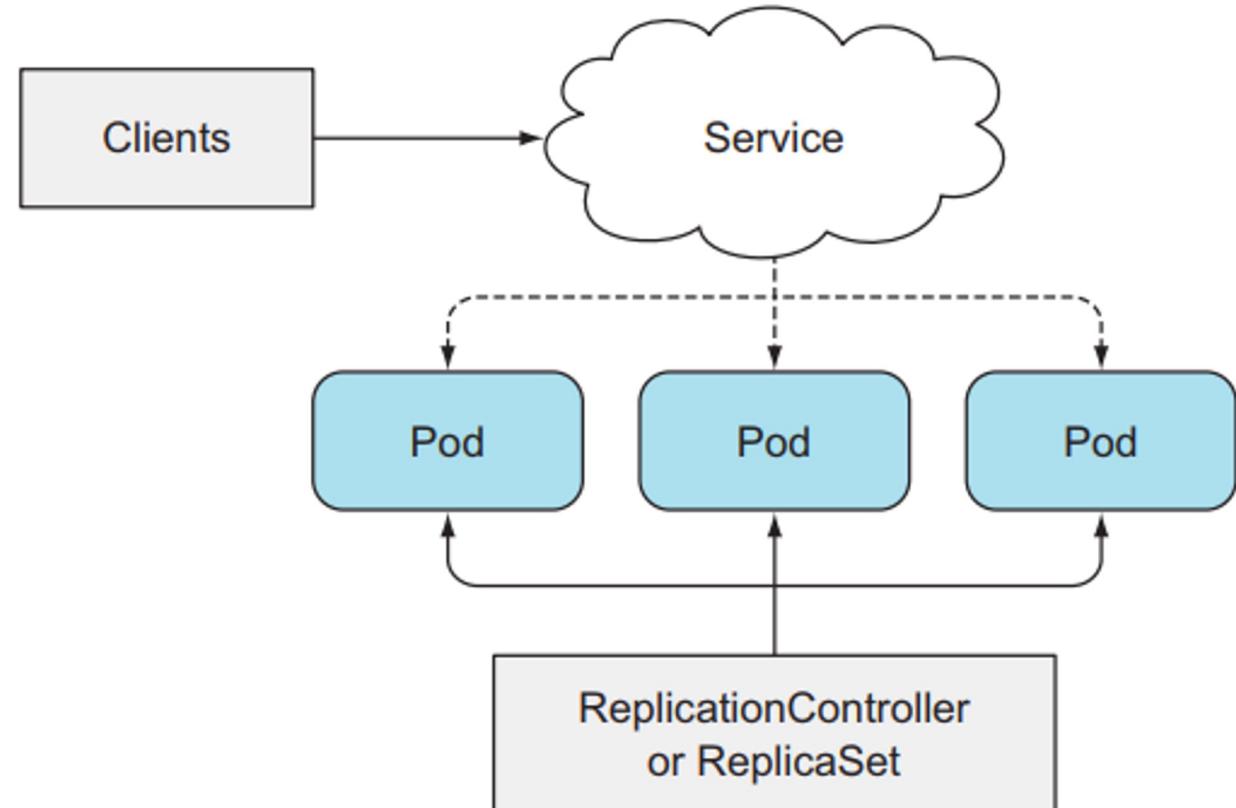
Nội dung

- 3.3.1. Pod
- 3.3.2. Replicaset
- 3.3.3. Service
- 3.3.4. Deployment

Deployment

Bây giờ các dev trong team đã viết xong tính năng mới, ta build lại image với code mới, và ta muốn update lại các Pod đang chạy này với image mới. Ta sẽ làm như thế nào?

- Rolling Update
- Recreate



Recreate

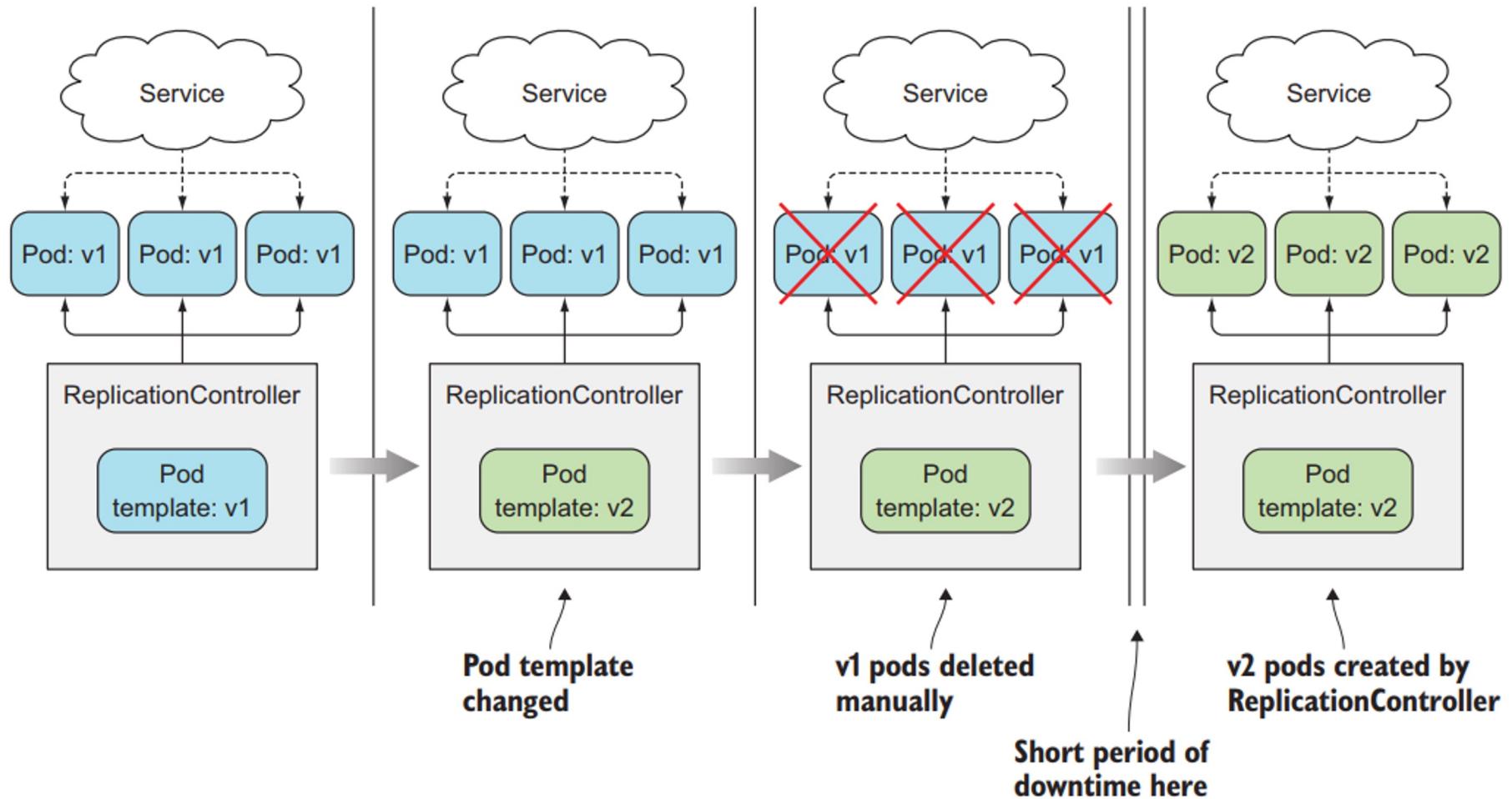


Figure 9.2 Updating pods by changing a ReplicationController's pod template and deleting old Pods

Rolling update

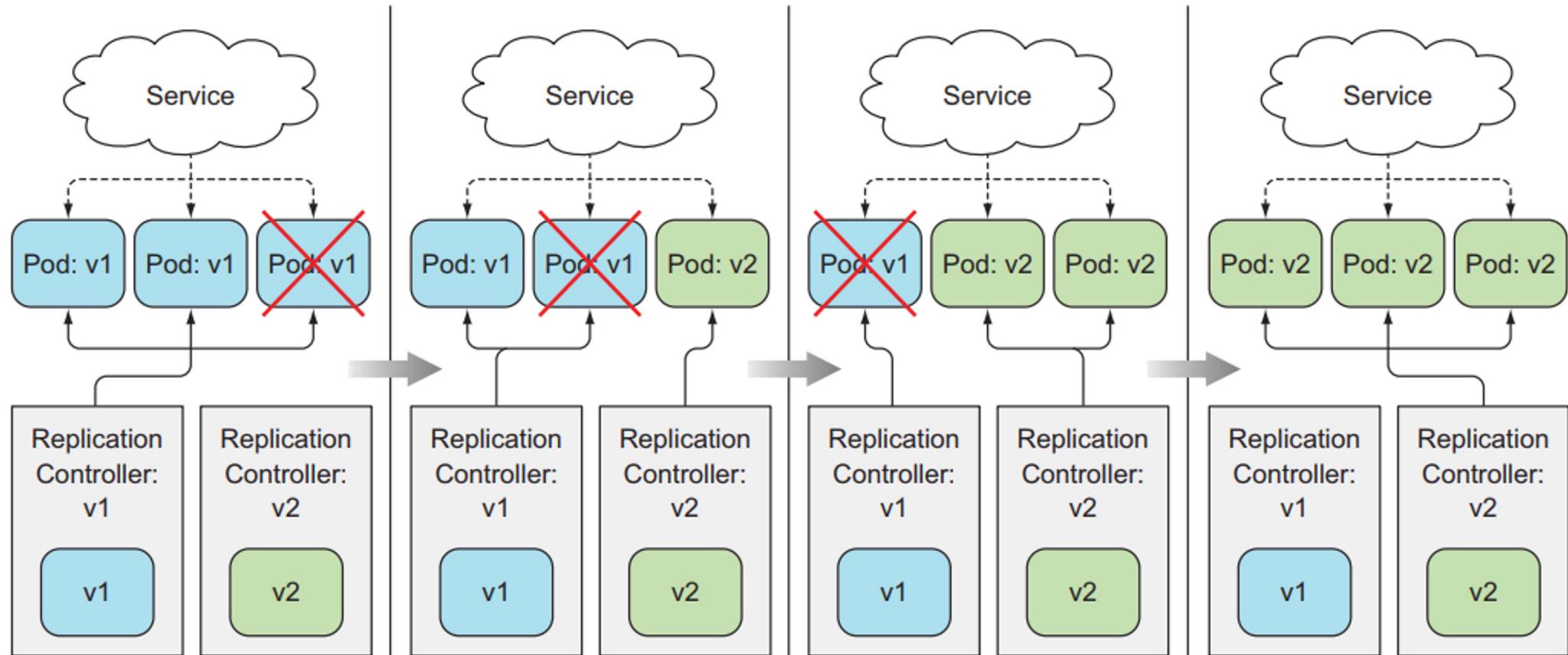


Figure 9.4 A rolling update of pods using two ReplicationControllers

Deployment giải quyết vấn đề gì

- Deployment là một resource của kubernetes giúp ta trong việc cập nhật một version mới của ứng dụng một cách dễ dàng, nó cung cấp sẵn 2 strategy để deploy là Recreate và RollingUpdate, tất cả đều được thực hiện tự động bên dưới, và các version được deploy sẽ có một histroy ở đằng sau, ta có thể rollback and rollout giữa các phiên bản bất cứ lúc nào mà không cần chạy lại CI/CD.



Nội dung

1. Triển khai ứng dụng web
2. Docker
3. K8s
4. Kiến trúc serverless

Serverless là gì

Serverless là môi trường, nền tảng để thực thi các ứng dụng và dịch vụ mà không cần quan tâm đến máy chủ. Đối với Serverless, bạn sẽ không phải quan tâm đến việc phân bổ, quản lý tài nguyên hệ điều hành hay những vấn đề về nâng cấp, bảo mật. Với môi trường Serverless, bạn chỉ cần tập trung phát triển sản phẩm còn vấn đề vận hành như thế nào sẽ do nền tảng này đảm nhiệm.

Ưu và nhược điểm

Ưu điểm	Nhược điểm
Không cần quản lý máy chủ	Độ trễ:
Thay đổi quy mô một cách linh hoạt	Giới hạn về bộ nhớ, thời gian
Độ sẵn sàng cao	Phụ thuộc nhà cung cấp
Tiết kiệm chi phí:	Chi phí ngầm

Nên sử dụng khi nào

Websites và APIs: hoàn toàn có thể xây dựng 1 website hoặc API, website có thể là động hoặc là bán tĩnh (bán tĩnh nghĩa là nguồn gốc file là tĩnh, nhưng dùng route động).

Xử lý đa phương tiện: các thao tác xử lý hình ảnh, video với yêu cầu không quá cao như cắt, nén, định dạng kích thước ảnh, tạo ảnh thumbnail, hoặc chuyển đổi bộ mã của video để phù hợp với thiết bị tương ứng.

Xử lý sự kiện: có thể đóng vai trò như 1 công tắc cầu giao để thực hiện một loạt các hành động khác khi được kích hoạt tùy theo sự kiện.

Xử lý dữ liệu:

Tài liệu tham khảo

Docker: <https://viblo.asia/s/thuc-hanh-docker-tu-can-ban-68Z00n0zZkG>

Docker network: <https://docker-ghichep.readthedocs.io/en/latest/docker-network/>

K8s: <https://viblo.asia/s/kubernetes-series-bq5QL8QGID8>

Serverless: <https://topdev.vn/blog/lam-quen-voi-kien-truc-serverless/>



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

