Báo cáo thực tập về Kubernetes

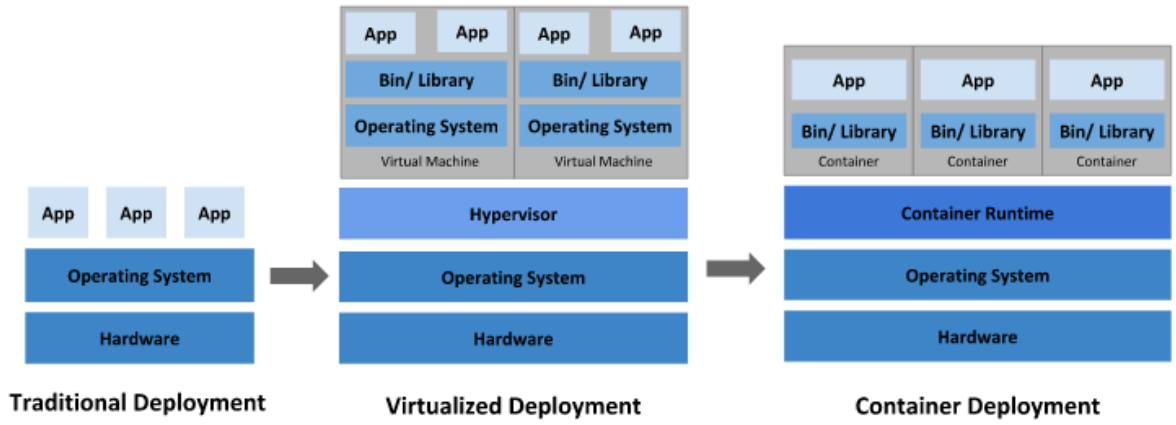
Mục lục

1. Tổng quan về kubernet

Kubernetes là một nền tảng nguồn mở, khả chuyển, có thể mở rộng để quản lý các ứng dụng được đóng gói và các service, giúp thuận lợi trong việc cấu hình và tự động hoá việc triển khai ứng dụng. Kubernetes là một hệ sinh thái lớn và phát triển nhanh chóng. Các dịch vụ, sự hỗ trợ và công cụ có sẵn rộng rãi.

Tên gọi Kubernetes có nguồn gốc từ tiếng Hy Lạp, có ý nghĩa là người lái tàu hoặc hoa tiêu. Mục đích của kubernetes là quản lí các container của docker một cách dễ dàng hơn

Chúng ta hãy xem tại sao Kubernetes rất hữu ích bằng cách quay ngược thời gian.



Traditional Deployment : Ban đầu, các ứng dụng chạy trên các máy chủ vật lí. Không có cách nào để xác định ranh giới phân bổ tài nguyên cho ứng dụng trong máy chủ vật lí do đó sẽ có những sự cố liên quan đến phân bổ tài nguyên xảy ra. Ví dụ ta chạy nhiều ứng dụng trên một máy chủ và trong đó có một máy chủ chiếm hết tài nguyên làm cho ứng dụng khác bị lỗi không sử dụng được nữa. Một giải pháp nữa là chay từng ứng dụng trên từng máy chủ vật lí khác nhau. Tuy nhiên như vậy sẽ rất tốn tiền mua máy chủ và không tối ưu được tài nguyên do các ứng dụng chưa chắc đã sử dụng hết được tài nguyên

Virtualized Deployment: giải pháp ảo hóa ra đời để giải quyết các vấn đề của traditional deployment. Nó cho phép chạy nhiều máy tính ảo trong cùng nền tảng máy chủ vật lí. Các máy tính này luôn sẵn sàng để sử dụng như những máy tính vật lí thông thường. Ảo hóa giúp tối ưu tài nguyên máy chủ hơn so với traditional deployment và ngoài ra còn giúp khả năng mở rộng và quản lí tài nguyên dễ dàng hơn. Mỗi VM là một máy tính chạy tất cả các thành phần, bao gồm cả hệ điều hành riêng của nó, bên trên phần cứng được ảo hóa.

Container Deployment : Các container tương tự VM nhưng chúng độc lập hơn và chia sẻ chung một hệ điều hành so với máy chủ vật lí. Cũng nhờ thế mà chúng nhẹ hơn so với VM. Tương tự như VM, một container có hệ thống tệp (filesystem), CPU, bộ nhớ, process space, v.v. Khi chúng được tách rời khỏi cơ sở hạ tầng bên dưới, chúng có thể khả chuyển

(portable) trên cloud hoặc các bản phân phối Hệ điều hành. Các container đã trở nên phổ biến vì chúng có thêm nhiều lợi ích, chẳng hạn như:

- Tạo mới và triển khai ứng dụng Agile: gia tăng tính dễ dàng và hiệu quả của việc tạo các container image so với việc sử dụng VM image.
- Phát triển, tích hợp và triển khai liên tục: cung cấp khả năng build và triển khai container image thường xuyên và đáng tin cậy với việc rollbacks dễ dàng, nhanh chóng.
- Phân biệt giữa Dev và Ops: tạo các images của các application container tại thời điểm build/release thay vì thời gian triển khai, do đó phân tách các ứng dụng khỏi hạ tầng.
- Khả năng quan sát không chỉ hiển thị thông tin và các metric ở mức Hệ điều hành, mà còn cả application health và các tín hiệu khác.
- Tính nhất quán về môi trường trong suốt quá trình phát triển, testing và trong production: Chạy tương tự trên laptop như trên cloud.
- Tính khả chuyển trên cloud và các bản phân phối HĐH: Chạy trên Ubuntu, RHEL, CoreOS, on-premises, Google Kubernetes Engine và bất kì nơi nào khác.
- Quản lý tập trung ứng dụng: Tăng mức độ trừu tượng từ việc chạy một Hệ điều hành trên phần cứng ảo hóa sang chạy một ứng dụng trên một HĐH bằng logical resources.
- Các micro-services phân tán, elastic: ứng dụng được phân tách thành các phần nhỏ hơn, độc lập và thể được triển khai và quản lý một cách linh hoạt - chứ không phải một app nguyên khối (monolithic).
- Cô lập các tài nguyên: dự đoán hiệu năng ứng dụng
- Sử dụng tài nguyên: hiệu quả

Vậy thì tại sao lại cần dùng thêm kubernetes. Trong container deploy, ứng dụng của chúng ta được đóng gói và chạy trên container. Tuy nhiên, trong môi trường production thì sẽ không dùng lại ở như vậy được. Các container phải được chạy và không có thời gian downtime. Ví dụ một container vừa bị chết thì phải có một cái mới được tạo ra ngay lập tức để thay thế. Điều này dễ dàng hơn khi có một hệ thống xử lí

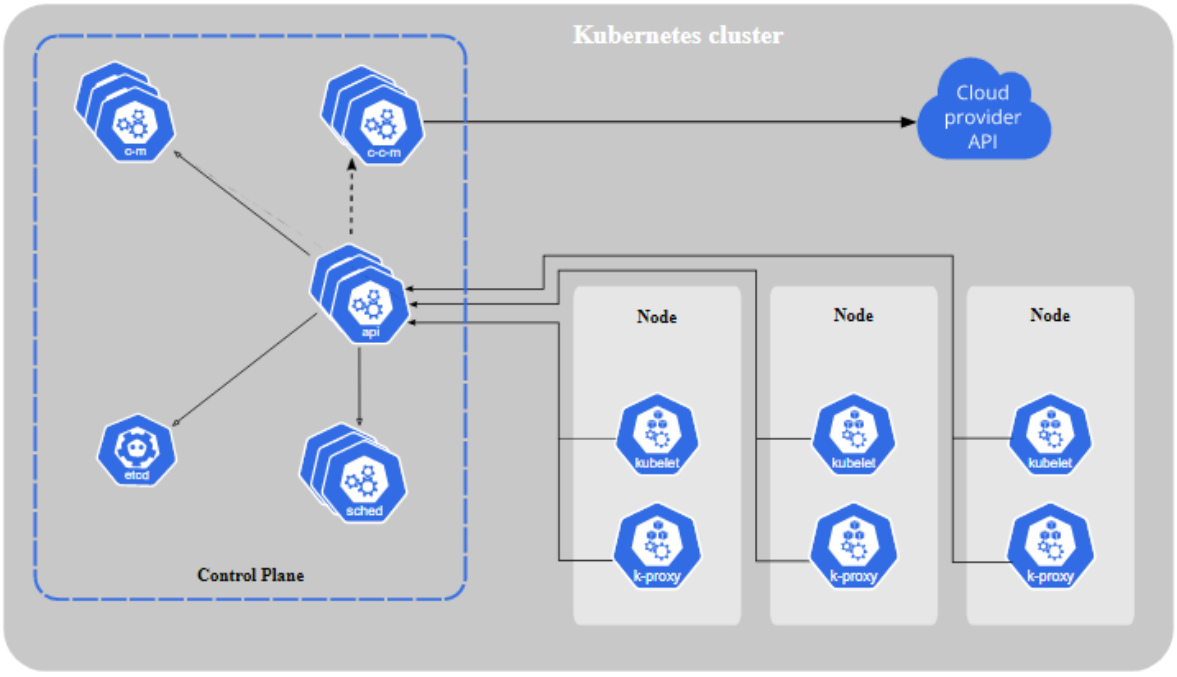
Đó là cách Kubernetes đến với chúng ta. Kubernetes cung cấp cho bạn một framework để chạy các hệ phân tán một cách mạnh mẽ. Nó đảm nhiệm việc nhân rộng và chuyển đổi dự phòng cho ứng dụng của bạn, cung cấp các mẫu deployment và hơn thế nữa. Ví dụ, Kubernetes có thể dễ dàng quản lý một triển khai canary cho hệ thống của bạn.

Kubernetes cung cấp một số tính năng như:

- **Service discovery và cân bằng tải** : Kubernetes có thể expose một container sử dụng DNS hoặc địa chỉ IP của riêng nó. Nếu lượng traffic truy cập đến một container cao, Kubernetes có thể cân bằng tải và phân phối lưu lượng mạng (network traffic) để việc triển khai được ổn định.
- **Điều phối bộ nhớ:** Kubernetes cho phép bạn tự động mount một hệ thống lưu trữ mà bạn chọn, như local storages, public cloud providers, v.v.
- **Tự động rollouts và rollbacks:** Bạn có thể mô tả trạng thái mong muốn cho các container được triển khai dùng Kubernetes và nó có thể thay đổi trạng thái thực tế sang trạng thái mong muốn với tần suất được kiểm soát. Ví dụ, bạn có thể tự động hoá Kubernetes để tạo mới các container cho việc triển khai của bạn, xoá các container hiện có và áp dụng tất cả các resource của chúng vào container mới.
- **Đóng gói tự động :** Bạn cung cấp cho Kubernetes một cluster gồm các node mà nó có thể sử dụng để chạy các tác vụ được đóng gói (containerized task). Bạn cho Kubernetes biết mỗi container cần bao nhiêu CPU và bộ nhớ (RAM). Kubernetes có thể điều phối các container đến các node để tận dụng tốt nhất các resource của bạn.
- **Tự phục hồi:** Kubernetes khởi động lại các containers bị lỗi, thay thế các container, xoá các container không phản hồi lại cấu hình health check do người dùng xác định và không cho các client biết đến chúng cho đến khi chúng sẵn sàng hoạt động.
- **Quản lý cấu hình và bảo mật:** Kubernetes cho phép bạn lưu trữ và quản lý các thông tin nhạy cảm như: password, OAuth token và SSH key. Bạn có thể triển khai và cập nhật lại secret và cấu hình ứng dụng mà không cần build lại các container image và không để lộ secret trong cấu hình stack của bạn

2. Các thành phần trong kubernetes

Khi bạn triển khai một kubernetes, bạn sẽ có một kubernetes cluster. Kubernetes cluster bao gồm các worker machine gọi là nodes chạy containerization. Trong nodes có các Pods – là thành phần chứa container của chúng ta. Control Plane sẽ điều khiển các node và pod trong một cluster



The components of a Kubernetes cluster

2.1. Control Plane

2.1.1. Kube-api-server

Đây giống như là trái tim của kubernetes do nó hoạt động cùng tất cả các thành phần có trong kubernetes. Nó cung cấp các API của kubernetes . Nó được coi là frontend của kubernetes control plane

2.1.2. Etcd

Là một hệ thống cơ sở dữ liệu dạng key-value của kubernetes lưu trữ dữ liệu trong k8s cluster. Các dữ liệu này là dữ liệu quản lý tài nguyên của k8s nhưng thông tin node, pod, service, deployment, configmap… Ta cần phải có kế hoạch backup dữ liệu etcd định kì

2.1.3. Kube – schedule

Đây là thành phần điều khiển có nhiệm vụ theo dõi các Pods mới được tạo mà chưa được gán vào node, và thực hiện tìm kiếm một node phù hợp trong cluster để chạy Pods đó lên. Có rất nhiều tham số ảnh hưởng tới việc điều phối một Pod vào một node như:

- Các yêu cầu về tài nguyên cho Pod như RAM/CPU..
- Các ràng buộc về phần cứng phần mềm
- Các chỉ định về affinity và anti-affinity.. và nhiều yếu tố khác nữa

2.1.4. Kube-controler manager

Là thành phần điều khiển có vai trò quản lý, trong đó nó là tổng hợp của nhiều controller khác nhưng chạy chung 1 process, bao gồm:

- Node controller: Có trách nhiệm phát hiện và cảnh báo khi node bị down
- Job controller: Theo dõi các object (đại diện cho một task cụ thể nào đó) được tạo ra sau đó tạo Pods để chạy các task đó tới khi hoàn thành.
- Endpoints controller: Quản lý các endpoints, là việc tạo sự kết nối giữa Service và Pods.
- Service Account và Token controllers: Tạo ra các account và token API mặc định cho namespace mới.

2.1.5. Cloud-controller-manager

Là thành phần điều khiển mà nhúng các logic điều khiển của nền tảng cloud cụ thể. Cloud-controller-manager cho phép người quản lý kết nối k8s cluster với API của nhà cung cấp cloud. Thành phần này là không bắt buộc nếu bạn không sử dụng cloud của cloud provider.

2.2. Thành phần của node

2.2.1. Kubelet

Là một agent chạy trên từng node trong cluster. Nó có vai trò đảm bảo các container ở trạng thái running ở trong Pod. Công việc của nó là lắng nghe thông tin các Pod được gán cho node đó (mà nó đang chạy) và dùng nhiều cơ chế để đảm bảo việc tạo ra các container đúng theo mô tả trong podspec được chạy (running) và không bị lỗi (healthy). Nó cũng đồng thời cập nhật trạng thái của các container trên node cũng như thông tin của node về control plane. Ví dụ đơn giản nhất là nếu kubelet bị stop thì node đó sẽ hiển thị trạng thái NotReady trong cluster.

2.2.2. Kube-proxy

Đây là network-proxy chạy trên từng node. Nó quản lý và duy trì các network rule trên các node. Những rule này đảm bảo kết nối tới các Pods từ bên trong hoặc bên ngoài cluster.

2.2.3. Container runtime

Đây là thành phần có trách nhiệm cho việc chạy các container. Một số container runtime điển hình mà k8s hỗ trợ gồm có Docker và Containerd.

2.3. Thành phần addon

2.3.1. DNS

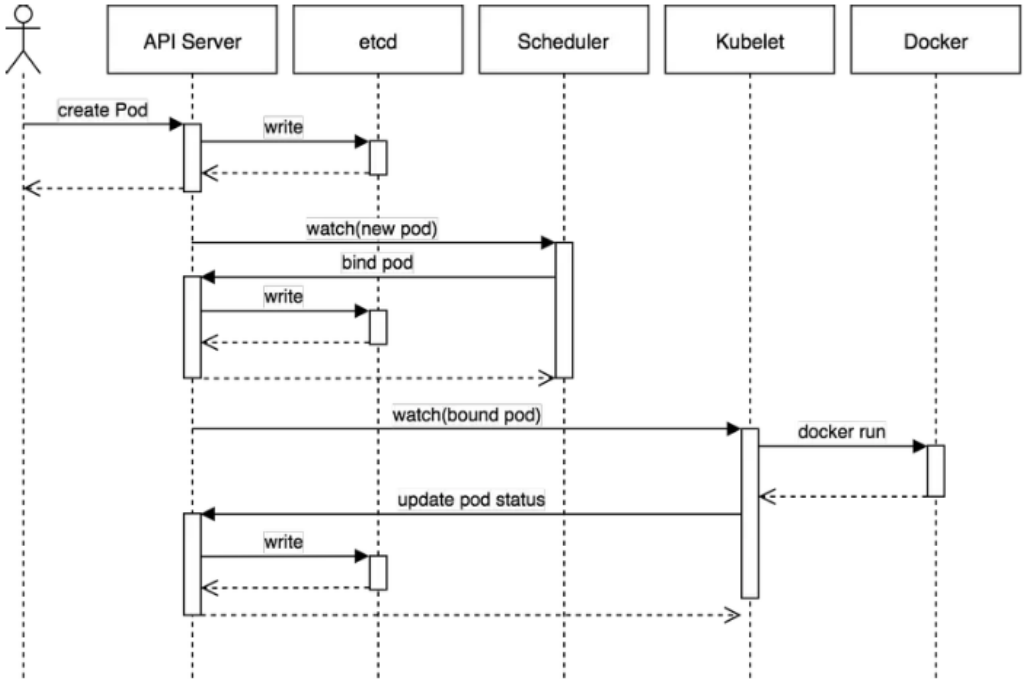
Dù addon thì thường không phải là bắt buộc nhưng mọi kubernetes cluster đều cần có nó. Nó bản chất làm một DNS server lưu trữ các bản ghi cho các service của kubernetes. Sau này khi làm việc với k8s các bạn sẽ thấy hầu hết chúng ta làm việc với các service qua service name, và đương nhiên thì để làm được việc đó thì k8s cần có DNS.

2.3.2. Web UI

Dashboard là một giao diện web-UI giúp ta theo dõi,quản lý cũng như troubleshoot các ứng dụng chạy trên cluster và chính tình trạng của cluster một cách trực quan hơn. Ngoài sử dụng dashboard, thì có nhiều giải pháp tương tự cũng khá phổ biến như sử dụng Rancher, Lens hay k9s.. để có giao diện quản trị với k8s cluster.

2.4. Life cycle của Pod

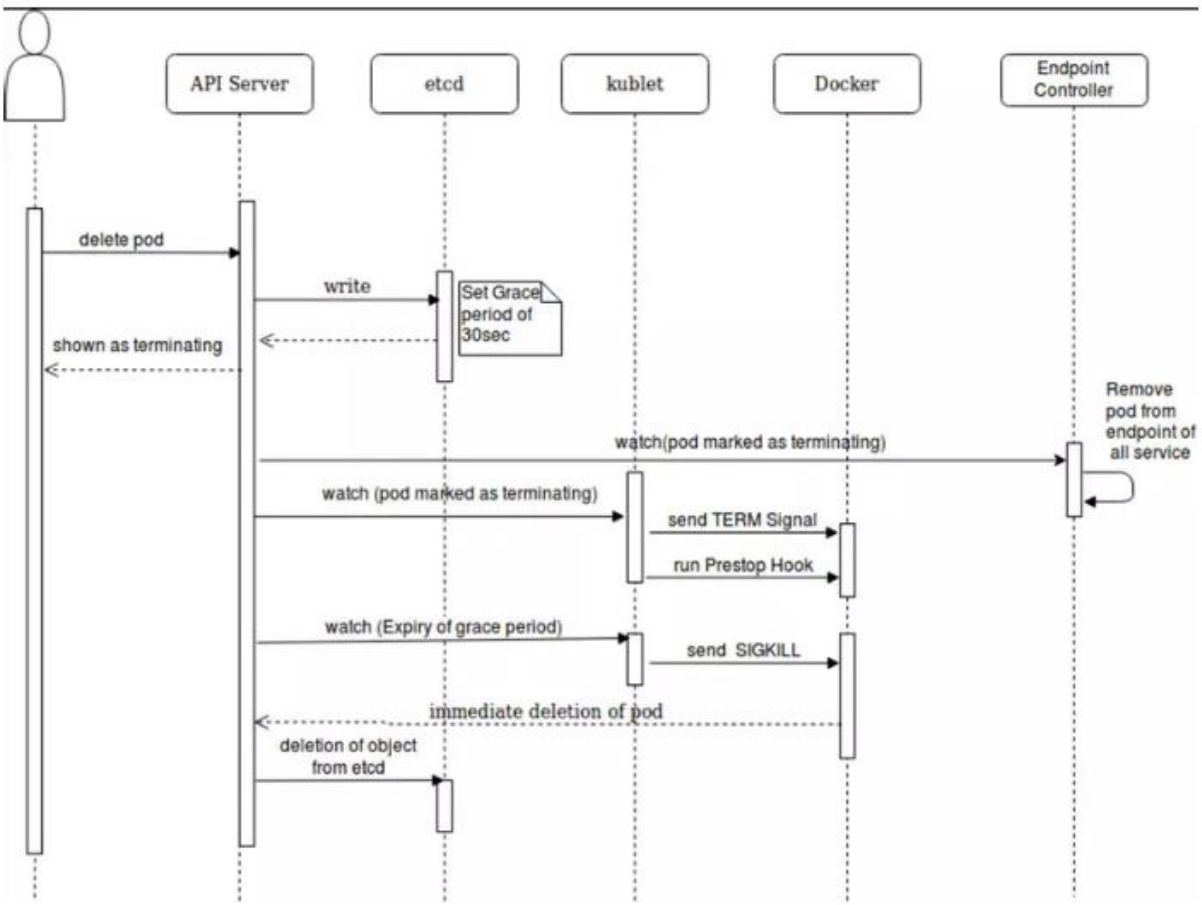
Khi tạo Pod mới, thì các sự kiện sẽ diện ra như sau:



1. Khi ta thực hiện tạo một Pod mới, thông tường là dùng lệnh kubectl để apply một file yaml là file mô tả chi thiết các thông tin cần thiết cho việc tạo Pod. Khi đó bản chất lệnh kubectl sẽ làm việc với api-server để gọi một api tương ứng cho việc tạo Pod.

2. API server xử lý yêu cầu trên bằng cách validate cú pháp của file yaml và nếu không có vấn đề gì thì sẽ thực hiện ghi dữ liệu này vào etcd - Là key-value db đã mô tả bên trên. Như vậy tại thời điểm này, trên hệ thống đã ghi nhận một Pod mới cần được tạo. Sau khi ghi xong vào etcd thì api-server phản hồi lại kết quả cho client là Pod đã được tạo.
3. Lúc này tới lượt scheduler tham gia vào. Như đã nói, nó sẽ theo dõi các Pod mới tạo trên hệ thống (bằng cách định kỳ kiểm tra api-server xem có thay đổi không) mà chưa được gán vào node để xử lý. Giờ nó phát hiện ra Pod mới này, nó sẽ lấy thông tin của Pod này và tìm một node thỏa mãn các yêu cầu, ví dụ là node1 và update vào thông tin của Pod là nó hãy chạy trên node1. Thông tin này được scheduler gửi cho api-server
4. API server nhận được thông tin Pod mới được gán vào node1 thì thực hiện update thông tin này và etcd. Lúc nào pod ở trạng thái bound.
5. Tiếp đến là kubelet cũng theo dõi các Pod ở trạng thái bound và được xếp lịch chạy trên node đó (bằng cách định kỳ lấy thông tin từ api-server). Ví dụ trong trường hợp này kubelet trên node1 phát hiện Pod mới được yêu cầu chạy trên node1 nên nó sẽ lấy thông tin cần thiết cho Pod và chạy Pod này thành các container trên node1. sau đó update lại trạng thái của Pod cho api-server.
6. API server nhận được thông tin cập nhật trạng thái Pod từ kubelet ở node1 thì nó thực hiện ghi thông tin này vào etcd và nhận phản hồi kết quả từ etcd. Sau đó nó gửi bản tin acknowledgement tới kubelete để báo rằng event này đã được chấp nhận.

Khi xóa Pod sẽ diễn ra như sau:



Chi tiết luồng xóa Pod:

- Người dùng gửi lệnh để xóa Pod
- Đối tượng Pod trên k8s được cập nhật trạng thái thành "dead" sau một khoảng thời gian gọi là grace-time
- Các hành động sau diễn ra song song: Pod sẽ hiện thị ở trạng thái "Terminating" khi được kiểm tra từ phía client. Kubelet thấy một Pod được đánh dấu là Terminating thì nó bắt đầu thực hiện dừng process của Pod. Endpoint controller theo dõi pod đã được xóa chưa để xóa thông tin pod đó khỏi các endpoint mà nó phục vụ
- Nếu pod có định nghĩa một **preStop hook**, thì nó được gọi tới bên trong pod. Nếu preStop hook vẫn đang chạy mà grace-time đã hết, thì bước (2) sẽ lại được gọi với thời gian grace-time nới thêm là 2 giây.
- Process bên trong Pod đã được gửi tín hiệu yêu cầu terminate (TERM signal)
- Sau khi grace-time kết thúc, thì mọi process bên trong Pod sẽ bị kill bởi SIGKILL.
- Kubelet hoàn thành xóa Pod bằng cách gọi API server và set grace-time bằng 0, nghĩa là yêu cầu xóa ngay lập tức. Lúc này Pod sẽ không còn và client sẽ không thể thấy được Pod này nữa.

3. Service, Load Balance trong kubernetes

3.1. Service trong kubernetes

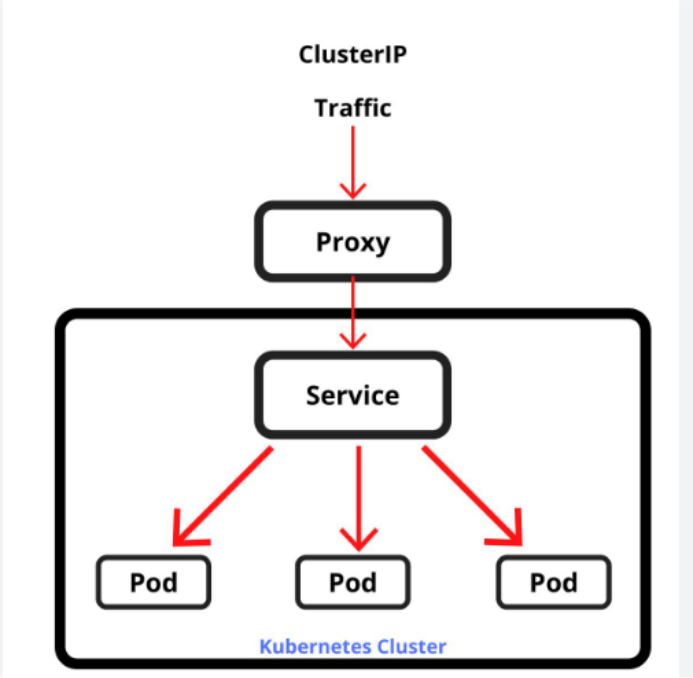
Trong Kubernetes, một **Service** là một đối tượng quan trọng được sử dụng để định tuyến và quản lý lưu lượng mạng giữa các pod trong một cluster. Đây là một khái niệm giúp bạn truy cập các ứng dụng chạy trong cluster một cách dễ dàng và đáng tin cậy.

Một số chức năng của Service có thể kể đến như:

- **Khả năng Truy cập Ổn định:** Service cung cấp một địa chỉ IP cố định và tên DNS cho các pod mà nó quản lý, đảm bảo rằng các ứng dụng có thể truy cập các pod thông qua cùng một địa chỉ ngay cả khi các pod đó có thể bị thay thế hoặc khởi động lại.
- **Cân Bằng Tải (Load Balancing):** Service có thể phân phối lưu lượng mạng đến các pod khác nhau để đảm bảo sự phân phối đều và cải thiện hiệu suất.
- **Cung Cấp Địa Chỉ và Cổng:** Service cung cấp một điểm cuối duy nhất mà bạn có thể sử dụng để giao tiếp với các pod, bất kể số lượng pod thay đổi.

Một số type của service:

- **ClusterIP**: Đây là kiểu mặc định của Service. Nó cung cấp một địa chỉ IP nội bộ trong cluster và chỉ có thể được truy cập từ bên trong cluster.
- **NodePort:** Kiểu này mở một cổng cố định trên tất cả các node trong cluster, cho phép bạn truy cập Service từ bên ngoài cluster thông qua IP của node và cổng đã mở.
- **LoadBalancer:** Kiểu này tạo ra một Load Balancer bên ngoài (thường là của nhà cung cấp dịch vụ đám mây) và ánh xạ nó tới Service của bạn, cung cấp một địa chỉ IP công cộng để truy cập từ bên ngoài.



Service tạo các internal IP cho các Pod có trong node của mình và lưu lại. Sau đó, nó mở một port ra ngoài môi trường ngoài và tiếp nhận traffic từ môi trường ngoài. Traffic đó sau đó được định tuyến đến các Pod theo các internal IP phù hợp. Trong quá trình định tuyến này nó sẽ sử dụng load balancing để cân bằng tải trong các Pod

Dưới đây là ví dụ về đoạn mã YAML cấu hình Service trong K8s

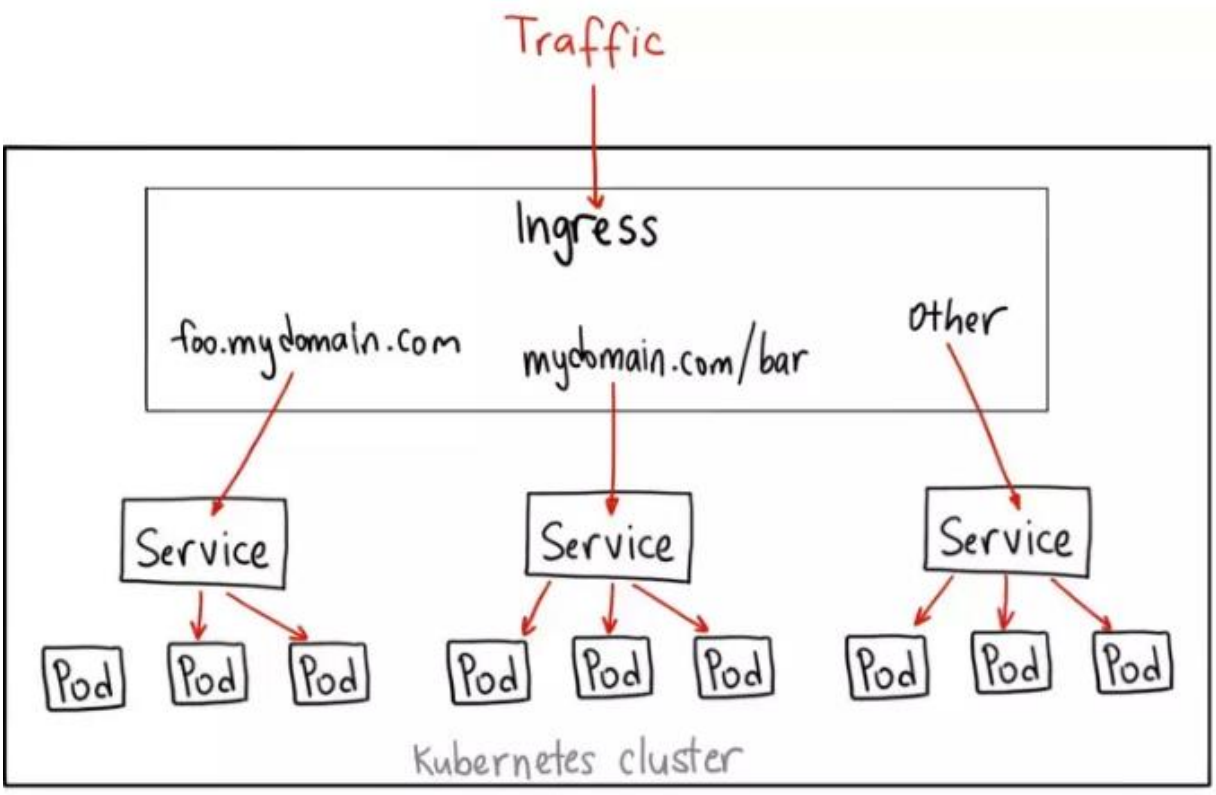
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

3.2. Ingress trong K8s

Trong Kubernetes, Ingress là một đối tượng cung cấp khả năng điều phối lưu lượng HTTP và HTTPS từ bên ngoài vào các dịch vụ (Services) bên trong cluster. Nó giúp bạn quản lý các yêu cầu HTTP/HTTPS và thực hiện các chức năng như điều hướng, cân bằng tải, và bảo mật.

Một số chức năng của Ingress có thể kể đến như:

- **Quản lý Điều Hướng:** Ingress cho phép bạn định nghĩa các quy tắc để điều hướng lưu lượng dựa trên URL hoặc tên miền, giúp bạn triển khai nhiều ứng dụng trong cùng một cluster và truy cập chúng qua cùng một địa chỉ IP công cộng.
- **Hỗ Trợ HTTPS:** Ingress có thể cấu hình với chứng chỉ TLS để mã hóa lưu lượng HTTPS và cung cấp bảo mật cho các kết nối giữa người dùng và ứng dụng của bạn.
- **Cân Bằng Tải:** Ingress controller có thể phân phối lưu lượng giữa nhiều dịch vụ hoặc phiên bản của dịch vụ, giúp cải thiện hiệu suất và độ tin cậy của ứng dụng.
- **Tối Ưu Hóa Lưu Lượng:** Ingress cho phép bạn tạo ra các quy tắc để tối ưu hóa lưu lượng mạng, chẳng hạn như chuyển hướng lưu lượng từ một URL đến một URL khác hoặc từ một tên miền đến một tên miền khác.



Trước khi Traffic được gửi đến các service, nó sẽ đi qua Ingress. Tại đây, ta thiết lập các rule để chuyển tiếp lưu lượng đến đúng service. Ví dụ như trong ảnh, nếu traffic có tên miền foo.mydomain.com sẽ chuyển đến service 1, tên miền mydomain.com/bar sẽ chuyển đến service 2 và lưu lượng còn lại sẽ chuyển đến service 3.

Dưới đây là ví dụ đơn giản về cấu hình YAML cho Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: example-service
                port:
                  number: 80
  tls:
    - hosts:
      - example.com
      secretName: example-tls
```

Để hoạt động, Ingress cần một Ingress controller. Ingress Controller là một ứng dụng quản lý các đối tượng Ingress và thực thi các quy tắc mà bạn đã định nghĩa. Một số Ingress Controller phổ biến bao gồm:

- **NGINX Ingress Controller**: Một Ingress Controller phổ biến sử dụng NGINX để thực hiện các quy tắc Ingress.
- **HAProxy Ingress**: Một Ingress Controller sử dụng HAProxy để xử lý lưu lượng mạng.
- **Traefik**: Một Ingress Controller và cân bằng tải hiện đại hỗ trợ nhiều tính năng và cấu hình dễ dàng.

4. Lưu trữ dữ liệu trong K8s

Trong Kubernetes, việc quản lý lưu trữ (storage) là một phần quan trọng của việc triển khai các ứng dụng, đặc biệt khi bạn cần lưu trữ dữ liệu bền vững hoặc chia sẻ dữ liệu giữa các pod. Có một số thành phần như sau:

- **Volume**: Là một khái niệm chung hơn trong Kubernetes, cho phép pod chia sẻ dữ liệu. Tuy nhiên, volume không bền vững và sẽ bị xóa nếu Pod bị xóa.
- **PersistentVolume (PV)**: Là một tài nguyên trong cluster cung cấp một phần lưu trữ bền vững. Được tạo ra bởi người quản trị hoặc tự động
- **PersistentVolumeClaim (PVC)**: Là một yêu cầu của người dùng để sử dụng một lượng lưu trữ nhất định. PVC định nghĩa yêu cầu về kích thước và các thuộc tính lưu trữ

4.1. PersistentVolume (PV)

PersistentVolume (PV) trong Kubernetes là một đối tượng giúp quản lý và trừu tượng hóa lưu trữ bền vững, cung cấp một cách để lưu trữ dữ liệu mà không bị mất khi các pod bị xóa hoặc khởi động lại. PV là một phần quan trọng của cơ chế lưu trữ trong Kubernetes, giúp bạn duy trì dữ liệu lâu dài và chia sẻ dữ liệu giữa các pod.

Nó có một số tính năng như sau:

- **Lưu Trữ Bền Vững**: PV cung cấp lưu trữ bền vững cho các pod, nghĩa là dữ liệu trên PV vẫn còn tồn tại ngay cả khi pod sử dụng nó bị xóa.

- **Trừu Tượng Hóa Lưu Trữ**: PV trừu tượng hóa việc quản lý lưu trữ, cho phép bạn làm việc với lưu trữ mà không cần biết chi tiết về cách lưu trữ đó được cung cấp (ví dụ: ổ đĩa cục bộ, ổ đĩa mạng, lưu trữ đám mây).
- **Kích Thước và Chế Độ Truy Cập**: Bạn có thể cấu hình kích thước lưu trữ và chế độ truy cập (Access Modes) của PV, như ReadWriteOnce (chỉ có một pod có thể ghi dữ liệu), ReadOnlyMany (nhiều pod có thể đọc dữ liệu nhưng không ghi), và ReadWriteMany (nhiều pod có thể đọc và ghi dữ liệu).
- **Tính Tương Thích với PVC**: PV được liên kết với PersistentVolumeClaim (PVC) bởi các thuộc tính như kích thước và chế độ truy cập. PVC là yêu cầu của người dùng về lưu trữ mà Kubernetes sẽ tự động liên kết với PV phù hợp.
- **Quản Lý Vị Trí Lưu Trữ**: PV có thể được cấu hình để sử dụng các loại lưu trữ khác nhau, bao gồm ổ đĩa cục bộ (Local Storage), mạng (Networked Storage), hoặc lưu trữ đám mây (Cloud Storage).

Dưới đây là ví dụ config YAML PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
  storageClassName: manual
  reclaimPolicy: Retain
```

4.2. PVC

PersistentVolumeClaim (PVC) trong Kubernetes là một đối tượng mà người dùng hoặc ứng dụng sử dụng để yêu cầu lưu trữ từ các PersistentVolume (PV) đã được định nghĩa trong cluster. PVC giúp trừu tượng hóa việc yêu cầu lưu trữ bền vững và liên kết với PV, mà không cần người dùng phải biết chi tiết về cách thức lưu trữ được cung cấp.

Nó bao gồm một số tính năng như:

- **Yêu Cầu Lưu Trữ**: PVC cho phép bạn yêu cầu lưu trữ với kích thước và các thuộc tính cụ thể mà ứng dụng của bạn cần, mà không cần biết chi tiết về loại hoặc cách thức lưu trữ cụ thể.
- **Tự Động Liên Kết với PV**: Kubernetes sẽ tự động liên kết PVC với một PV đáp ứng yêu cầu của PVC nếu có sẵn. Nếu không có PV phù hợp, PVC sẽ giữ trạng thái Pending cho đến khi có một PV tương thích.
- **Tính Linh Hoạt**: PVC có thể yêu cầu các thuộc tính như kích thước lưu trữ, chế độ truy cập, và có thể được sử dụng để yêu cầu lưu trữ từ các StorageClass, giúp định nghĩa loại lưu trữ và các tùy chọn bổ sung.
- **Quản Lý Tài Nguyên**: PVC giúp quản lý tài nguyên lưu trữ một cách hiệu quả, cho phép các pod dễ dàng truy cập lưu trữ bền vững mà không cần cấu hình lưu trữ trong từng pod.

Dưới đây là cấu hình PVC bằng YAML

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: standard
```

5. Configuration trong K8s

Configuration có một vai trò quan trọng trong việc triển khai và quản lí ứng dụng. Kubernetes cung cấp nhiều cách để quản lí và cấu hình ứng dụng, dịch vụ giúp điều chỉnh và định nghĩa các hành vi trong cluster

Có thể kể đến một số configuration như sau:

- ConfigMap
- Secret
- Environment Variables

5.1. ConfigMap

ConfigMap là một đối tượng dùng để lưu trữ các cặp khóa-giá trị, thường được sử dụng để cung cấp cấu hình cho các ứng dụng. ConfigMap cho phép bạn tách biệt cấu hình khỏi mã nguồn và dễ dàng cập nhật cấu hình mà không cần thay đổi hình ảnh container.

Ví dụ cấu hình ConfigMap như sau:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  app.config: |
    key1=value1
    key2=value2
  config-file.properties: |
    property1=value1
    property2=value2
```

sử dụng configMap trong Pod như sau:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
```

```
  containers:
    - name: my-container
      image: nginx
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: my-config
```

5.2. Secret

Cũng tương tự so với ConfigMap tuy nhiên thì Secret được dùng để lưu dữ liệu bảo mật

Ví dụ cấu hình trong Secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: dXNlcg==  # base64 encoding of 'user'
  password: cGFzc3dvcmQ=  # base64 encoding of 'password'
```

sử dụng secret trong Pod

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx
      env:
        - name: USERNAME
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: username
        - name: PASSWORD
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: password
```

5.3. Environment Variables

Các biến môi trường (Environment Variables) là một cách phổ biến để cấu hình các ứng dụng. Bạn có thể định nghĩa các biến môi trường trực tiếp trong cấu hình pod hoặc tham chiếu đến các ConfigMap hoặc Secret.

Ví dụ về environment

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
```

```
spec:
  containers:
    - name: my-container
      image: nginx
      env:
        - name: APP_ENV
          value: production
        - name: LOG_LEVEL
          value: debug
```