Kubernetes

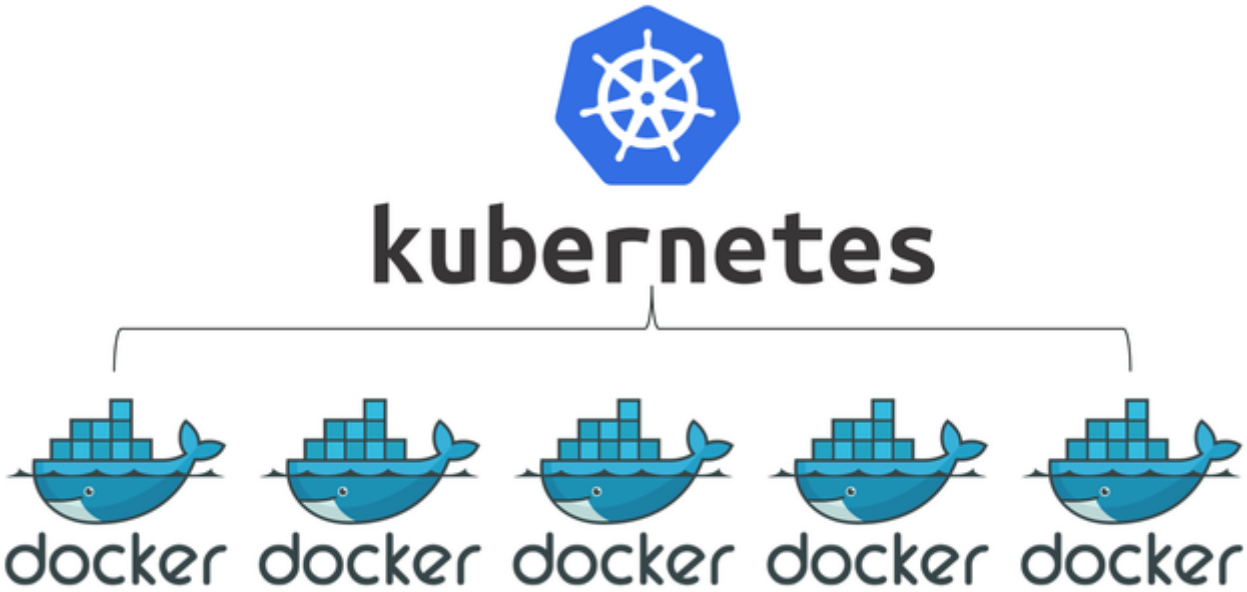
Kubernetes là một Platform được áp dụng để tự động hoá việc quản lý, scaling(mở rộng) và triển khai ứng dụng dưới dạng container hoá cụ thể là các docker, Kubernetes còn gọi là Container orchestration engine.

- Giới thiệu Kubernetes
- Kubernetes Workloads
- Kubernetes Pod
- Kubernetes Node
- Lệnh cơ bản trong Kubernetes
- Kubernetes Label - Nhãn đối tượng trong Kubernetes
- Kubernetes Deployment - Triển khai ứng dụng bằng Deployment
- Hướng dẫn cài đặt Sonatype Nexus 3 trên Kubernetes
- Kubernetes Service - Truy cập Service trong Kubernetes
- Hướng dẫn cài đặt Kubernetes trên Linux
- Kubernetes - Job
- Kubernetes - Labels & Selectors
- Kubernetes - Namespace
- Kubernetes - Service
- Kubernetes - Replication Controller
- Kubernetes - Replica Sets
- Kubernetes - Volume
- Kubernetes - Secret
- Kubernetes - Network Policy
- Kubernetes - Autoscaling

Giới thiệu Kubernetes

1. Kubernetes là gì?

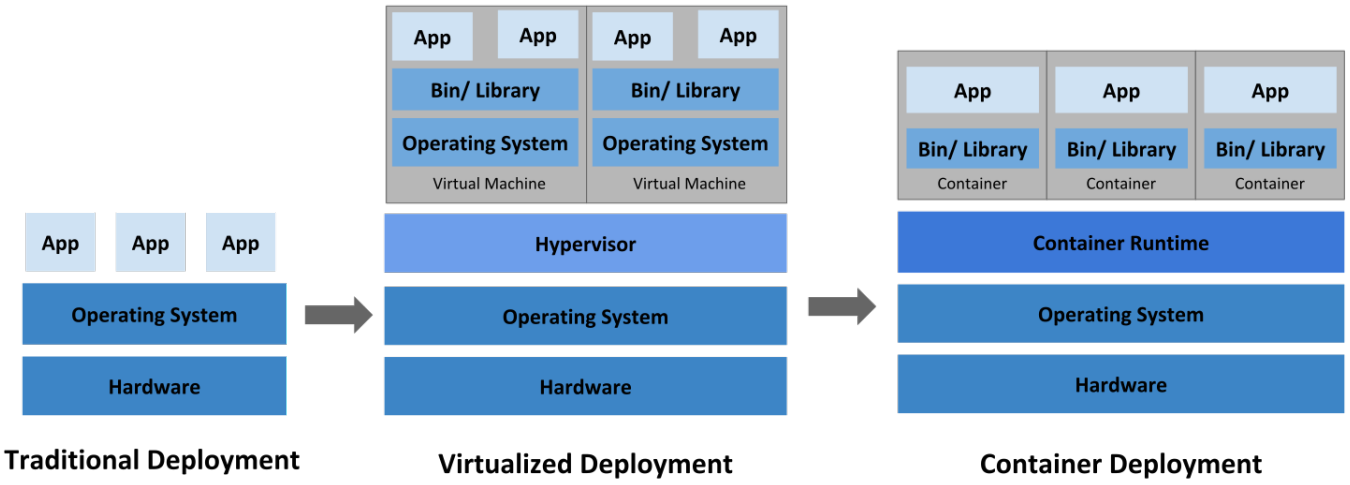
Kubernetes là một Platform được áp dụng để tự động hoá việc quản lý, scaling(mở rộng) và triển khai ứng dụng dưới dạng container hoá cụ thể là các docker, Kubernetes còn gọi là Container orchestration engine.



Các đặc điểm chính của Kubernetes bao gồm:

- Quản lý các container cluster
- Cung cấp các công cụ để triển khai các ứng dụng
- Mở rộng các ứng dụng khi cần thiết
- Quản lý các thay đổi đối với các ứng dụng được container hóa hiện có
- Giúp tối ưu hóa việc sử dụng phần cứng cơ bản bên dưới container
- Cho phép thành phần ứng dụng khởi động lại và di chuyển trên toàn hệ thống khi cần thiết

2. Các mô hình triển khai ứng dụng



Mô trình truyền thống: Ban đầu, các ứng dụng được chạy trên các máy chủ vật lý. Không có cách nào để xác định ranh giới tài nguyên cho các ứng dụng trong máy chủ vật lý và điều này gây ra sự cố phân bổ tài nguyên. Ví dụ, nếu nhiều ứng dụng cùng chạy trên một máy chủ vật lý, có thể có những trường hợp một ứng dụng sẽ chiếm phần lớn tài nguyên hơn và kết quả là các ứng dụng khác sẽ hoạt động kém đi. Một giải pháp cho điều này sẽ là chạy từng ứng dụng trên một máy chủ vật lý khác nhau. Nhưng giải pháp này không tối ưu vì tài nguyên không được sử dụng đúng mức và rất tốn kém cho các tổ chức để có thể duy trì nhiều máy chủ vật lý như vậy.

Mô hình ảo hóa: Như một giải pháp, ảo hóa đã được giới thiệu. Nó cho phép bạn chạy nhiều Máy ảo (VM) trên CPU của một máy chủ vật lý. Ảo hóa cho phép các ứng dụng được cô lập giữa các VM và cung cấp mức độ bảo mật vì thông tin của một ứng dụng không thể được truy cập tự do bởi một ứng dụng khác.

Ảo hóa cho phép sử dụng tốt hơn các tài nguyên trong một máy chủ vật lý và cho phép khả năng mở rộng tốt hơn vì một ứng dụng có thể được thêm hoặc cập nhật dễ dàng, giảm chi phí phần cứng và hơn thế nữa. Với ảo hóa, bạn có thể có một tập hợp các tài nguyên vật lý dưới dạng một cụm các máy ảo sẵn dùng.

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.

Môi trường Container: Các container tương tự như VM, nhưng chúng có tính cô lập để chia sẻ Hệ điều hành (HĐH) giữa các ứng dụng. Do đó, container được coi là nhẹ (lightweight). 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.

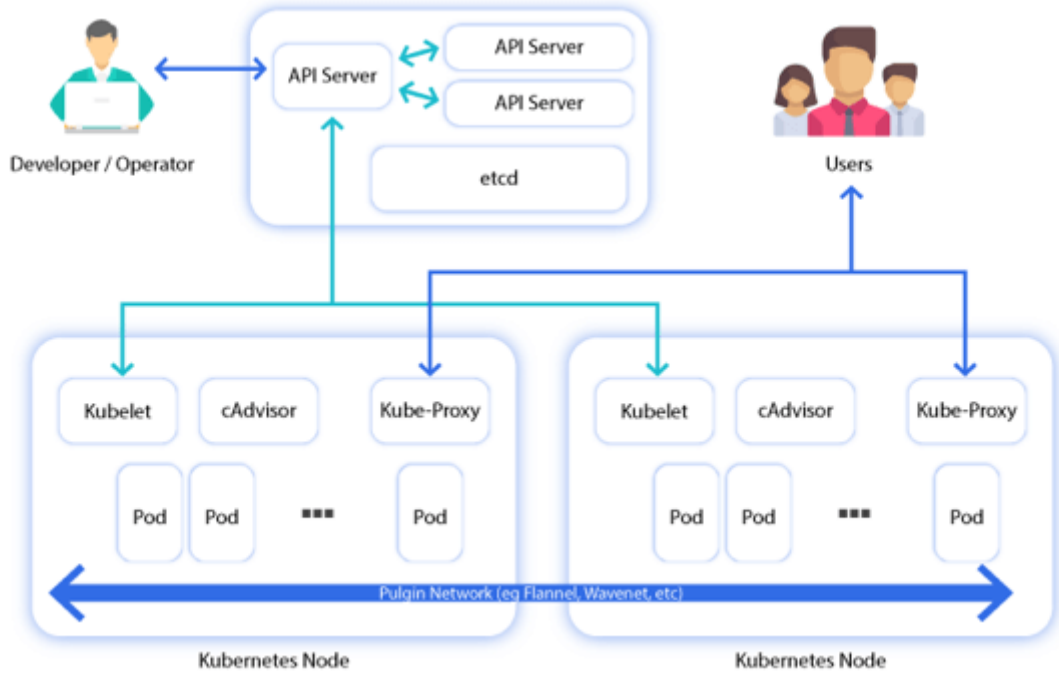
Các container đã trở nên phổ biến vì chúng có thêm nhiều lợi ích, chẳng hạn như:

- 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

3. Thành phần cơ bản của Kubernetes

Kubernetes cung cấp nhiều hơn ngoài framework cơ bản, cho phép người dùng chọn loại framework ứng dụng, ngôn ngữ, công cụ giám sát, logging và các công cụ khác mà họ chọn. Mặc dù nó không phải là Platform as a Service, nhưng nó có thể được sử dụng làm cơ sở cho một PaaS hoàn chỉnh.



Các thành phần cơ bản của Kubernetes bao gồm:

- **Kubernetes Master:** Đây là đơn vị điều khiển chính quản lý workloads và liên lạc trên toàn hệ thống. Mỗi thành phần của nó có một quy trình khác nhau có thể chạy trên một master node hoặc trên nhiều master node. Thành phần của nó là:
  - **Etcd Storage**: Đây là kho lưu trữ dữ liệu key-value, là nguồn mở được phát triển bởi nhóm CoreOS và có thể được truy cập bởi tất cả các node trong cluster. Kubernetes

sử dụng "Etcd" để lưu trữ dữ liệu cấu hình của cluster để thể hiện trạng thái chung của cluster bất cứ lúc nào.
    - **API-Server**: API server là thực thể quản lý trung tâm nhận các REST requests để cập nhật các thông tin cấu hình, đóng vai trò là front-end để điều khiển cluster. Hơn nữa, đây là thứ duy nhất giao tiếp với cluster Etcd, và đảm bảo rằng dữ liệu được lưu trữ trong Etcd.
    - **Scheduler**: Nó giúp lên lịch các pod trên các node khác nhau dựa trên việc sử dụng tài nguyên và quyết định nơi nào triển khai dịch vụ nào. Bộ lập lịch có các thông tin liên quan đến các tài nguyên có sẵn cho các thành viên cũng như tài nguyên còn lại để cấu hình dịch vụ để chạy.
    - **Controller Manager**: Thực hiện một số quy trình điều khiển riêng biệt trong nền để điều chỉnh trạng thái chia sẻ của cluster và thực hiện một tác vụ theo quy luật. Khi có bất kỳ thay đổi nào trong dịch vụ, bộ điều khiển sẽ phát hiện ra sự thay đổi và bắt đầu làm việc theo trạng thái mong muốn mới.

- **Worker Node:** Đây còn được gọi là Minion node, nó chứa thông tin để quản lý kết nối mạng giữa các container như Docker và liên lạc giữa master node khi gán tài nguyên cho các container theo lịch trình
    - **Kubelet**: Kubelet đảm bảo rằng tất cả các container trong node đang chạy và ở trạng thái healthy. Kubelet theo dõi trạng thái của một pod nếu nó không ở trạng thái mong muốn. Nếu một node thất bại, bộ điều khiển sao chép sẽ quan sát sự thay đổi này và khởi chạy các pod trên một pod healthy khác.
    - **Container**: Container là mức thấp nhất của microservice, được đặt bên trong pod và cần địa chỉ IP bên ngoài để xem xét quy trình bên ngoài.
    - **Kube Proxy**: Nó hoạt động như một proxy mạng và bộ cân bằng tải. Ngoài ra, nó chuyển tiếp yêu cầu tới các pod chính xác trên các mạng bị cô lập trong một cluster.
    - **cAdvisor**: Hoạt động như một trợ lý chịu trách nhiệm theo dõi và thu thập dữ liệu về việc sử dụng tài nguyên và số liệu hiệu suất trên mỗi node.

4. Ưu điểm của Kubernetes

- **Nguồn mở và di động:** Kubernetes có thể chạy các container trên một hoặc nhiều môi trường public cloud, máy ảo hoặc trên bare metal, điều đó có nghĩa là nó có thể được triển khai trên bất kỳ cơ sở hạ tầng nào. Hơn nữa, nó tương thích trên nhiều nền tảng, giúp cho chiến lược đa cloud trở nên linh hoạt và có thể sử dụng được.
- **Khả năng mở rộng workloads:** Kubernetes cung cấp một số tính năng hữu ích cho mục đích mở rộng:
    - **Horizontal Infrastructure Scaling:** Các thao tác được thực hiện ở cấp máy chủ riêng lẻ để thực hiện chia tỷ lệ theo chiều ngang. Máy chủ mới có thể được thêm hoặc gỡ bỏ dễ dàng.
    - **Auto-Scaling:** Dựa trên việc sử dụng tài nguyên CPU hoặc các số liệu ứng dụng khác, có thể sửa đổi số lượng container đang chạy.
    - **Manual Scaling:** Có thể thay đổi số lượng container đang chạy thông qua một lệnh hoặc giao diện.
    - **Replication Controller:** Bộ điều khiển sao chép đảm bảo rằng cluster có số lượng pod tương đương được chỉ định trong điều kiện đang chạy. Nếu có quá nhiều pod, bộ

điều khiển sao chép có thể loại bỏ các pod bổ sung hoặc ngược lại.
- **Tính sẵn sàng cao:** Kubernetes có thể xử lý sự sẵn có của cả ứng dụng và cơ sở hạ tầng. Nó đã khắc phục được:
  - **Health Checks:** Kubernetes đảm bảo rằng ứng dụng không bị lỗi bằng cách liên tục kiểm tra tình trạng của các mode và container. Kubernetes cung cấp khả năng tự phục hồi và tự động thay thế nếu pod bị hỏng do lỗi.
  - **Định tuyến lưu lượng và cân bằng tải:** Bộ cân bằng tải Kubernetes phân phối tải trên nhiều tải, cho phép cân bằng tài nguyên nhanh chóng trong lưu lượng ngẫu nhiên hoặc xử lý hàng loạt.

5. Tại sao bạn cần Kubernetes và nó có thể làm những gì?

Các container là một cách tốt để đóng gói và chạy các ứng dụng của bạn. Trong môi trường production, bạn cần quản lý các container chạy các ứng dụng và đảm bảo rằng không có khoảng thời gian downtime. Ví dụ, nếu một container bị tắt đi, một container khác cần phải khởi động lên. Điều này sẽ dễ dàng hơn nếu được xử lý bởi một hệ thống.

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 cho bạn:

- **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.

6. Được thiết kế để triển khai

Triển khai ứng dụng theo mô hình container hóa có khả năng tăng tốc quá trình xây dựng, thử nghiệm và phát hành phần mềm và tính năng hữu ích bao gồm:

- **Tự động triển khai và khôi phục:** Kubernetes xử lý phiên bản mới và cập nhật cho ứng dụng mà không có downtime, đồng thời theo dõi health trong quá trình triển khai. Nếu bất kỳ lỗi nào xảy ra trong quá trình, nó sẽ tự động khôi phục.
- **Triển khai Canary:** Kubernetes kiểm tra song song việc sản xuất triển khai mới và phiên bản trước đó, trước khi tăng quy mô triển khai mới và đồng thời giảm quy mô triển khai trước đó.
- **Hỗ trợ Framework và ngôn ngữ lập trình:** Kubernetes hỗ trợ hầu hết các ngôn ngữ và framework lập trình như Java, .NET, v.v. và cũng nhận được sự hỗ trợ lớn từ cộng đồng phát triển. Nếu một ứng dụng có khả năng chạy trong một container, nó cũng có thể chạy trong Kubernetes.
- **Và nhiều hơn nữa:** Kubernetes cung cấp quản lý DNS, giám sát tài nguyên, ghi nhật ký, sắp xếp lưu trữ và cũng giải quyết vấn đề bảo mật. Chẳng hạn, nó đảm bảo rằng thông tin như mật khẩu hoặc ssh keys được lưu trữ an toàn trong các Kubernetes secret. Các tính năng mới được phát hành liên tục và có thể có trên Kubernetes GitHub.

7. Kubernetes không phải là gì?

Kubernetes không phải là một hệ thống PaaS (Nền tảng như một Dịch vụ) truyền thống, toàn diện. Do Kubernetes hoạt động ở tầng container chứ không phải ở tầng phần cứng, nó cung cấp một số tính năng thường áp dụng chung cho các dịch vụ PaaS, như triển khai, nhân rộng, cân bằng tải, ghi nhật ký và giám sát. Tuy nhiên, Kubernetes không phải là cấu trúc nguyên khối và các giải pháp mặ định này là tùy chọn và có thể cắm được (pluggable).

Kubernetes:

- Không giới hạn các loại ứng dụng được hỗ trợ. Kubernetes nhằm mục đích hỗ trợ một khối lượng công việc cực kỳ đa dạng, bao gồm cả stateless, stateful và xử lý dữ liệu. Nếu một ứng dụng có thể chạy trong một container, nó sẽ chạy rất tốt trên Kubernetes.
- Không triển khai mã nguồn và không build ứng dụng của bạn. Quy trình CI/CD được xác định bởi tổ chức cũng như các yêu cầu kỹ thuật.
- Không cung cấp các service ở mức ứng dụng, như middleware (ví dụ, các message buses), các framework xử lý dữ liệu (ví dụ, Spark), cơ sở dữ liệu (ví dụ, MySQL), bộ nhớ cache, cũng như hệ thống lưu trữ của cluster (ví dụ, Ceph). Các thành phần như vậy có thể chạy trên Kubernetes và/hoặc có thể được truy cập bởi các ứng dụng chạy trên Kubernetes

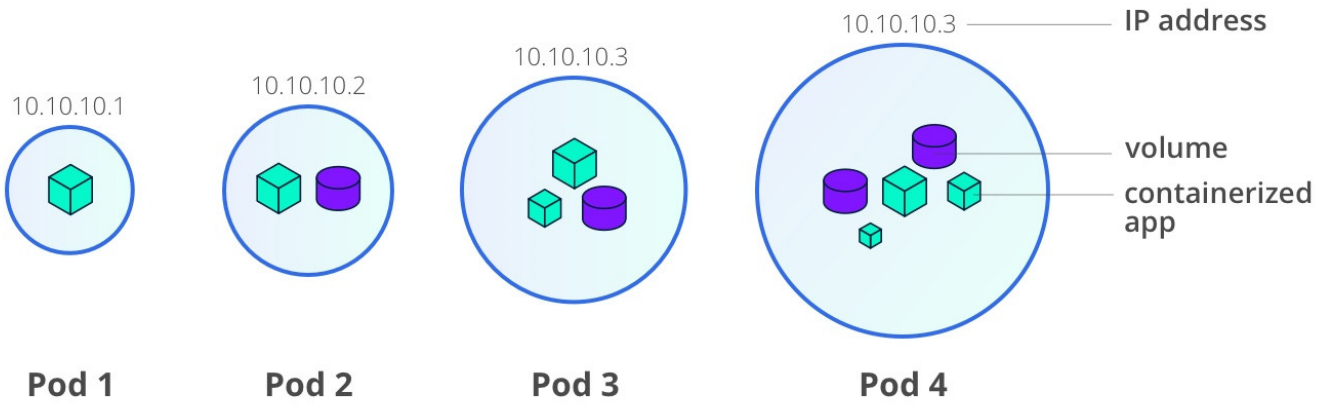
thông qua các cơ chế di động, chẳng hạn như Open Service Broker.

- Không bắt buộc các giải pháp ghi lại nhật ký (logging), giám sát (monitoring) hoặc cảnh báo (alerting). Nó cung cấp một số sự tích hợp như proof-of-concept, và cơ chế để thu thập và xuất các số liệu.
- Không cung cấp, không bắt buộc một cấu hình ngôn ngữ/hệ thống (ví dụ: Jsonnet). Nó cung cấp một API khai báo có thể được targeted bởi các hình thức khai báo tùy ý.
- Không cung cấp cũng như áp dụng bất kỳ cấu hình toàn diện, bảo trì, quản lý hoặc hệ thống tự phục hồi.
- Ngoài ra, Kubernetes không phải là một hệ thống điều phối đơn thuần. Trong thực tế, nó loại bỏ sự cần thiết của việc điều phối. Định nghĩa kỹ thuật của điều phối là việc thực thi một quy trình công việc được xác định: đầu tiên làm việc A, sau đó là B rồi sau chót là C. Ngược lại, Kubernetes bao gồm một tập các quy trình kiểm soát độc lập, có thể kết hợp, liên tục điều khiển trạng thái hiện tại theo trạng thái mong muốn đã cho. Nó không phải là vấn đề làm thế nào bạn có thể đi được từ A đến C. Kiểm soát tập trung cũng không bắt buộc. Điều này dẫn đến một hệ thống dễ sử dụng hơn, mạnh mẽ hơn, linh hoạt hơn và có thể mở rộng

Kubernetes Workloads

Kubernetes workloads được chia thành 2 nhóm chính bao gồm: **Pod** và **controllers** (Deployment, Relicaset, Stateful, Cronjob...)

Pod



- Pod là thành phần đặc trưng nhất của Kubernetes, Pod là container cho docker bao gồm cả tài nguyên lưu trữ, địa chỉ IP và các quy tắc (rule) về cách (các) container sẽ chạy.
- Thông thường, mỗi Pod sẽ triển khai 1 ứng dụng đơn lẻ (container docker), tuy nhiên, Pod hoàn toàn có thể triển khai nhiều container tùy theo mục đích triển khai.

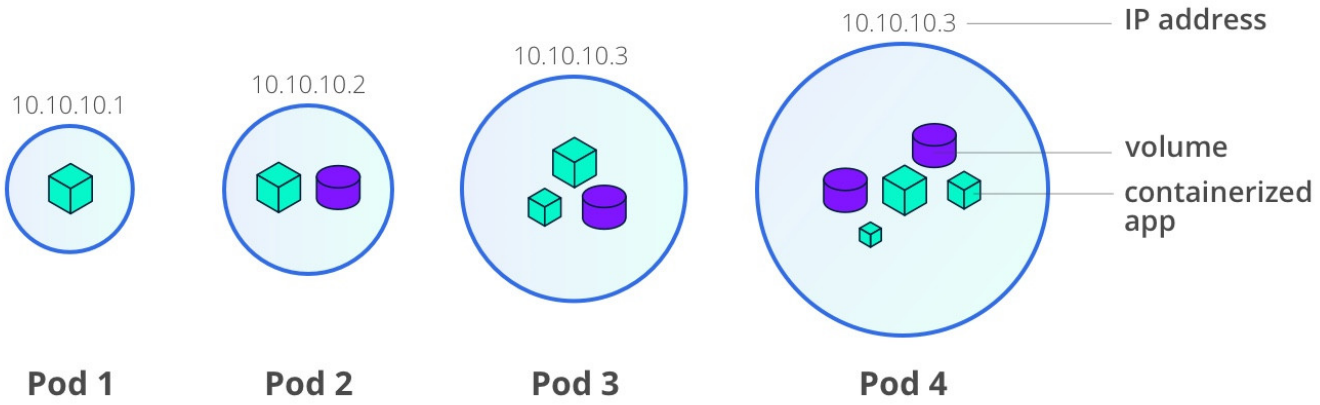
Workload Controllers

Các bộ điều khiển Controllers là các kịch bản triển khai Pod, bao gồm các loại sau:

- ReplicaSet: Là bộ điều khiển cho phép triển khai các bản sao cụ thể của Pod.
- Deployment: Là bộ điều khiển thường dùng nhất của Workload, Deployment cho phép cập nhật liên tục, khôi phục, mở rộng/scaling dễ dàng cho các Pod.
- StatefulSet: Là bộ điều khiển tương tự như Deployment nhưng dành cho kịch bản triển khai yêu cầu sử dụng network cụ thể và các vùng storage được chỉ định sẵn.
- Cronjob: Là bộ điều khiển quản lý các job cần được lập lịch.

Kubernetes Pod

`Pod` là thành phần đặc trưng nhất của Kubernetes, Pod là container cho docker bao gồm cả tài nguyên lưu trữ, địa chỉ IP và các quy tắc (rule) về cách (các) container sẽ thực thi và hoạt động.



Thông thường, mỗi Pod sẽ triển khai 1 ứng dụng đơn lẻ (container docker), tuy nhiên, Pod hoàn toàn có thể triển khai nhiều container tùy theo mục đích triển khai.

Vì Pod được hiểu là một đơn vị nguyên tử (atomic unit) trong Kubernetes, nên mỗi Pod thực sự phải chạy một phiên bản duy nhất của một ứng dụng nhất định. Vì vậy, nếu bạn cần chạy nhiều bản sao của một container, thì mỗi container phải chạy trong Pod duy nhất của riêng nó thay vì để tất cả các container đó nằm trong một Pod duy nhất. Tuy nhiên, đôi khi một Pod có thể triển khai nhiều container nếu chúng có liên quan chặt chẽ với nhau (ví dụ phổ biến là một số thành phần ghi log). Một điều quan trọng cần lưu ý là tất cả các container trong Pod sẽ chia sẻ cùng một môi trường: bộ nhớ, ổ đĩa, ngăn xếp mạng và quan trọng nhất là địa chỉ IP.

Trong quá trình triển khai thông thường, Pod không được lập trình viên trực tiếp tạo ra. Thay vào đó, các bộ điều khiển (controller) sẽ lên lịch cho Pod chạy trên Node. Một số điều quan trọng cần biết về Pod:

- Một Pod chỉ có thể tồn tại trên một Node duy nhất
- Một Pod không bao giờ có thể ở trạng thái được triển khai một phần. Nếu một phần của Pod không được triển khai thành công, nó được coi là `unhealthy` và bị lỗi.
- Một Pod vì lý do nào đó không được triển khai thành công hoặc gặp lỗi dẫn đến bị lỗi, nó được coi là một thành phần dùng một lần. Nói cách khác, nếu một Pod trở nên `unhealthy`, các phần tử Controller sẽ giết Pod đó và khởi động một Pod khác, thay thế thay vì chữa lành nó.

1. Tạo Pod từ command

Để tạo 1 từ container image `quay.io/openshiftlabs/simpleservice:0.5.0` và publish 1 HTTP API trên port 9876, thực hiện:

```
kubectl run sise --image=quay.io/openshiftlabs/simpleservice:0.5.0 --port=9876
```

Kiểm tra pod đang chạy:

```
kubectl get pods -A
```

Kết quả:

```
NAME READY STATUS RESTARTS AGE
sise 1/1 Running 0 1m
```

Xóa Pod vừa tạo:

```
kubectl delete pod sise
```

2. Tạo Pod từ file kịch bản (Manifest file)

Chúng ta có thể tạo Pod từ các file kịch bản, sử dụng lệnh kubectl apply:

```
kubectl apply -f https://raw.githubusercontent.com/openshift-
evangelists/kbe/main/specs/pods/pod.yaml
```

hoặc, có thể tạo file trên local với nội dung sau và thực thi lệnh kubectl apply:

```
apiVersion: v1
kind: Pod
metadata:
  name: twocontainers
spec:
  containers:
  - name: sise
    image: quay.io/openshiftlabs/simpleservice:0.5.0
    ports:
    - containerPort: 9876
  - name: shell
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
```

```
      - "sleep 10000"
```

Kiểm tra Pod vừa tạo:

```
kubectl get pods
```

Kết quả:

```
NAME                        READY   STATUS    RESTARTS   AGE
twocontainers               2/2     Running   0          36s
```

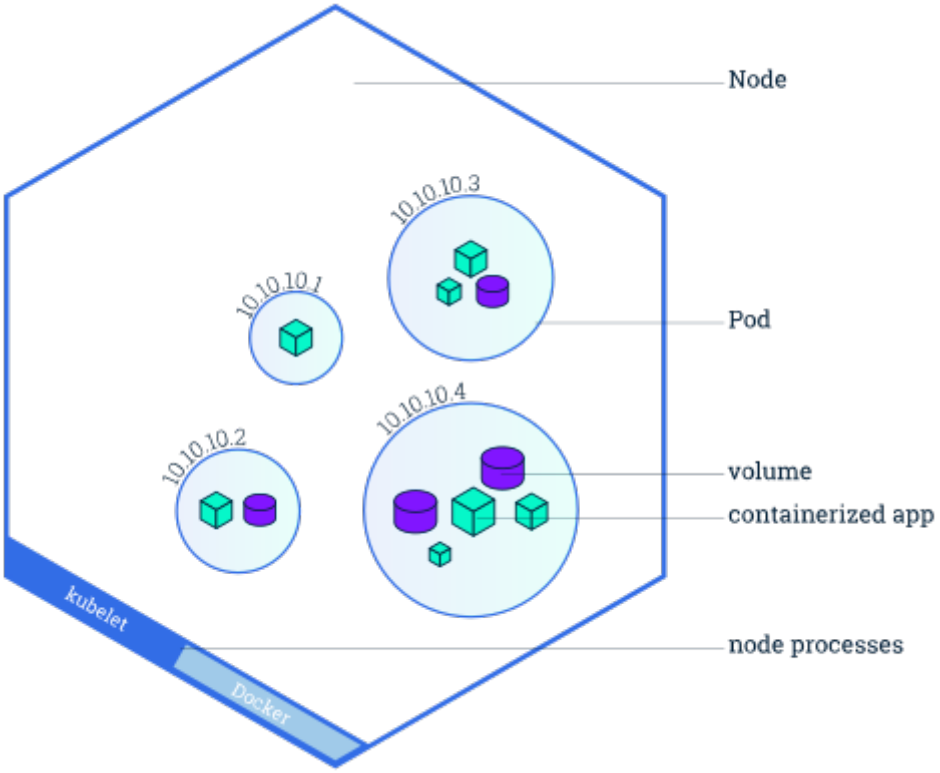
Xóa Pod vừa tạo:

```
kubectl delete pod twocontainers
```

Kubernetes Node

Trong Kubernetes, Node là đơn vị nhỏ nhất xét về phần cứng. Nó là một máy vật lý hay máy ảo (VPS) trong cụm máy (cluster). Xem danh sách các nút (node) trong cụm (cluster) chạy lệnh:

```
kubectl get nodes
```



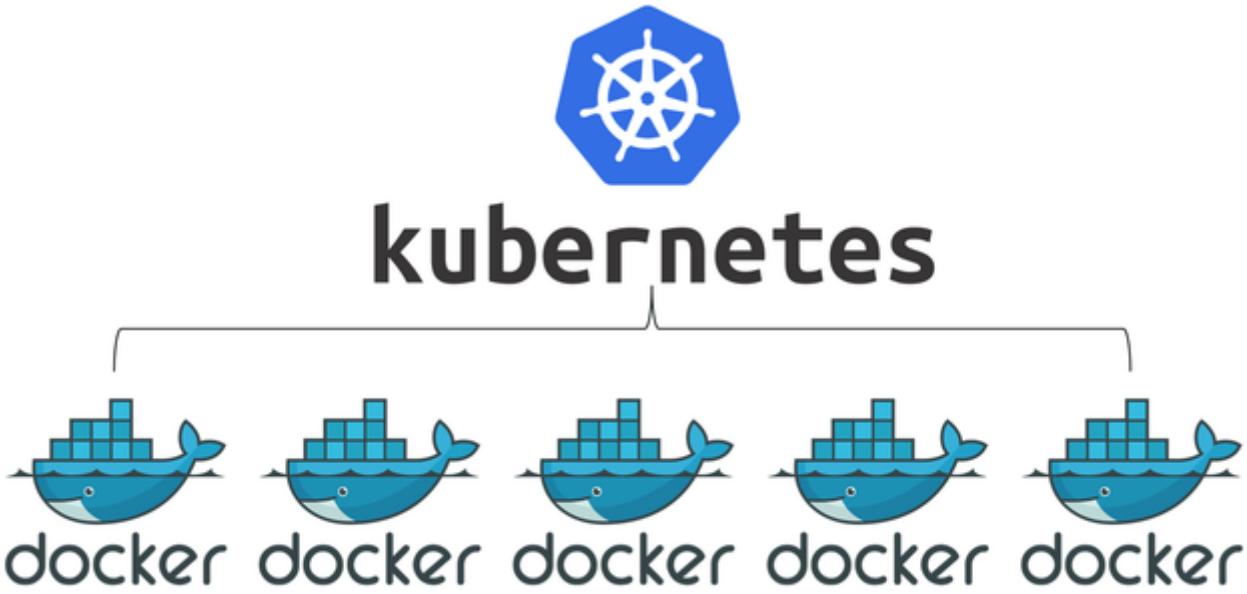
Một Node là một máy worker trong Kubernetes và có thể là máy ảo hoặc máy vật lý, tuỳ thuộc vào cluster. Mỗi Node được quản lí bởi Master. Một Node có thể chứa nhiều Pods và Kubernetes master tự động xử lí việc lên lịch trình các Pods thuộc các Nodes ở trong cluster. Việc lên lịch trình tự động của Master sẽ tính đến các tài nguyên có sẵn trên mỗi Node.

Mỗi Node ở Kubernetes chạy ít nhất:

- Kubelet, một quy trình chịu trách nhiệm liên lạc giữa Kubernetes Master và Node; quản lí các Pods và các containers đang chạy trên cùng một máy.
- Một container runtime (như Docker, rkt) chịu trách nhiệm lấy container image từ registry, giải nén container và chạy ứng dụng. Các containers chỉ nên được lên lịch trình cùng nhau trong một Pod duy nhất nếu chúng được liên kết chặt chẽ.

Lệnh cơ bản trong Kubernetes

Kubernetes là một Platform được áp dụng để tự động hoá việc quản lý, scaling(mở rộng) và triển khai ứng dụng dưới dạng container hoá cụ thể là các docker.



Dưới đây là các lệnh cơ bản trong Kubernetes.

1. Cấu hình kubectl

```
kubectl config view # Hiển thị các thiết lập kubeconfig đã được merged

# sử dụng nhiều tệp kubeconfig cùng một lúc và xem cấu hình hợp nhất
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# lấy mật khẩu cho người dùng e2e
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # hiển thị người dùng đầu tiên
kubectl config view -o jsonpath='{.users[*].name}'   # lấy danh sách người dùng
```

```
kubectl config get-contexts                          # hiển thị danh sách các ngữ cảnh
kubectl config current-context                       # hiển thị ngữ cảnh hiện tại
kubectl config use-context my-cluster-name           # thiết lập ngữ cảnh mặc định cho my-
cluster-name

# thêm một cụm mới vào kubeconf hỗ trợ xác thực cơ bản
kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser --
password=kubepassword

# lưu vĩnh viễn namespace cho tất cả các lệnh kubectl tiếp theo trong ngữ cảnh đó
kubectl config set-context --current --namespace=ggckad-s2

# thiết lập ngữ cảnh sử dụng tên người dùng và namespace cụ thể
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # xóa người dùng foo
```

2. Kubectl apply

`apply` quản lý các ứng dụng thông qua các tệp định nghĩa tài nguyên Kubernetes. Nó tạo và cập nhật các tài nguyên trong một cụm thông qua việc chạy kubectl apply. Đây là cách được đề xuất để quản lý các ứng dụng Kubernetes trong thực tế.

```
kubectl apply -f ./my-manifest.yaml            # tạo tài nguyên
kubectl apply -f ./my1.yaml -f ./my2.yaml      # tạo từ nhiều tệp
kubectl apply -f ./dir                         # tạo tài nguyên từ tất cả các tệp manifest trong
thư mục dir
kubectl apply -f https://git.io/vPieo          # tạo tài nguyên từ url
kubectl create deployment nginx --image=nginx  # tạo một deployment nginx
kubectl explain pods,svc                       # lấy thông tin pod và service manifest

# Tạo nhiều đối tượng YAML từ stdin
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
```

```
    image: busybox
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000"
EOF

# Tạo một secret với một số keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

3. Xem, tìm các tài nguyên Pod, Node

```
# Lệnh get với một số đầu ra cơ bản
kubectl get services                          # Liệt kê tất cả các services trong namespace
kubectl get pods --all-namespaces             # Liệt kê tất cả các pods trong tất cả các
namespaces
kubectl get pods -o wide                      # Liệt kê tất cả các pods namespace, với nhiều
thông tin hơn
kubectl get deployment my-dep                 # Liệt kê một deployment cụ thể
kubectl get pods                              # Liệt kê tất cả các pods trong namespace
```

```
kubectl get pod my-pod -o yaml                # Lấy thông tin của một pod ở dạng YAML
kubectl get pod my-pod -o yaml --export       # Lấy thông tin của một pod ở dạng YAML mà không c
thông tin cụ thể về cụm

# Lệnh describe
kubectl describe nodes my-node
kubectl describe pods my-pod

# Liệt kê các services được sắp xếp theo tên
kubectl get services --sort-by=.metadata.name

# Liệt kê các pods được sắp xếp theo số lần khởi động lại
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# Liệt kê các pods được sắp xếp theo dung lượng trong namespace có tên là test

kubectl get pods -n test --sort-by='.spec.capacity.storage'

# Lấy thông tin phiên bản của tất cả các pods có nhãn app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Liệt kê tất cả các worker nodes (sử dụng một selector để loại trừ kết quả có một nhãn
# có tên 'node-role.kubernetes.io/master'
kubectl get node --selector='!node-role.kubernetes.io/master'

# Liệt kê tất cả các pods đang chạy trong namespace
kubectl get pods --field-selector=status.phase=Running

# Liệt kê tất cả các ExternalIPs của tất cả các nodes
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

# Liệt kê tên của các pods thuộc về một RC nhất định
# Lệnh "jq" hữu ích cho các chuyển đổi quá mức phức tạp cho jsonpath, xem thêm tại
https://stedolan.github.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries | .[] |
"\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})

# Hiển thị nhãn của tất cả các pods (hoặc các đối tượng Kubernetes khác hỗ trợ gán nhãn)
```

```
kubectl get pods --show-labels

# Kiểm tra xem nodes nào đã sẵn sàng
JSONPATH='{range .items[*]}{@.metadata.name}:{range
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# Liệt kê tất cả cá Secrets hiện đang sử dụng bởi một pod
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name'
| grep -v null | sort | uniq

# Liệt kê tất cả các sự kiện được sắp xếp theo thời gian
kubectl get events --sort-by=.metadata.creationTimestamp
```

4. Cập nhật các tài nguyên

Từ phiên bản 1.11, rolling-update đã không còn được dùng nữa, sử dụng rollout thay thế.

```
kubectl set image deployment/frontend www=image:v2               # Cập nhận container "www" của
deployment "frontend", cập nhật image
kubectl rollout history deployment/frontend                      # Kiểm tra lịch sử deployment
bao gồm các sửa đổi
kubectl rollout undo deployment/frontend                         # Quay trở lại deployment
trước đó
kubectl rollout undo deployment/frontend --to-revision=2         # Quay trở lại một phiên bản cụ
thể
kubectl rollout status -w deployment/frontend                    # Xem trạng thái cập nhật của
deployment "frontend" cho đến khi hoàn thành



# không còn được sử dụng kể từ phiên bản 1.11
kubectl rolling-update frontend-v1 -f frontend-v2.json           # (không dùng nữa) Cập nhật
pods của frontend-v1
kubectl rolling-update frontend-v1 frontend-v2 --image=image:v2  # (không dùng nữa) Đổi tên tài
nguyên và cập nhật image
kubectl rolling-update frontend --image=image:v2                 # (không dùng nữa) Cập nhật
image của pod của frontend
kubectl rolling-update frontend-v1 frontend-v2 --rollback        # (không dùng nữa) Hủy bỏ tiến
trình cập nhật hiện tại
```

```
cat pod.json | kubectl replace -f -                                    # Thay thế một pod dựa trên
JSON được truyền vào std

# Buộc thay thế, xóa và sau đó tạo lại tài nguyên, sẽ gây ra sự cố ngưng services
kubectl replace --force -f ./pod.json

# Tạo một services cho nginx, phục vụ trên công 80 và kết nối đến các container trên cổng 8000
kubectl expose rc nginx --port=80 --target-port=8000

# Cập nhật phiên bản image của một container đơn lẻ lên v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | kubectl replace -f -

kubectl label pods my-pod new-label=awesome                            # Thêm một nhãn
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq             # Thêm một chú thích
kubectl autoscale deployment foo --min=2 --max=10                      # Tự động scale deployment
"foo"
```

5. Cập nhật các tài nguyên

```
# Cập nhật một phần một node
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# Cập nhật image của container; spec.containers[*].name là bắt buộc vì đó là khóa hợp nhất
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-
hostname","image":"new image"}]}}'

# Cập nhật image của container sử dụng một bản vá json với các mảng vị trí
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path":
"/spec/containers/0/image", "value":"new image"}]'

# Vô hiệu hóa một deployment livenessProbe sử dụng một bản vá json với các mảng vị trí
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "path":
"/spec/template/spec/containers/0/livenessProbe"}]'

# Thêm một phần tử mới vào một mảng vị trí
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "value":
{"name": "whatever" } }]'
```

6. Chỉnh sửa các tài nguyên

Chỉnh sửa bất kì API tài nguyên nào trong trình soạn thảo ưa thích của bạn.

```
kubectl edit svc/docker-registry # Ch□nh s□a services có tên docker-registry
KUBE_EDITOR="nano" kubectl edit svc/docker-registry # S□ dụng một trình soạn th□o thay th□
```

7. Scaling tài nguyên

```
kubectl scale --replicas=3 rs/foo # Scale một replicaset có tên 'foo' thành 3
kubectl scale --replicas=3 -f foo.yaml # Scale một tài nguyên được xác định trong "foo.yaml"
thành 3
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql # N□u kích thước hiện tại c□a
deployment mysql là 2, scale mysql thành 3
kubectl scale --replicas=5 rc/foo rc/bar rc/baz # Scale nhi□u replication controllers
```

8. Xóa tài nguyên

```
kubectl delete -f ./pod.json # Xóa một pod s□ dụng loại và tên được xác định trong pod.json
kubectl delete pod,service baz foo # Xóa pods và services có tên "baz" và "foo"
kubectl delete pods,services -l name=myLabel # Xóa pods và services có nhãn name=myLabel
kubectl -n my-ns delete pod,svc --all # Xóa t□t c□ pods và services trong namespace my-ns,
# Xóa t□t c□ pods matching với pattern1 hoặc pattern2
kubectl get pods -n mynamespace --no-headers=true | awk '/pattern1|pattern2/{print $1}' |
xargs kubectl delete -n mynamespace pod
```

9. Xem log với các pods đang chạy

```
kubectl logs my-pod # k□t xu□t logs c□a pod (stdout)
kubectl logs -l name=myLabel # k□t xu□t logs c□a pod có nhãn name=myLabel (stdout)
kubectl logs my-pod --previous # k□t xu□t logs c□a pod (stdout) cho kh□i tạo trước c□a một
container
kubectl logs my-pod -c my-container # k□t xu□t logs c□a container c□a pod (stdout, trường hợp có
nhi□u container)
kubectl logs -l name=myLabel -c my-container # k□t xu□t logs c□a container có tên my-container v
có nhãn name=myLabel (stdout)
kubectl logs my-pod -c my-container --previous # k□t xu□t logs c□a container my-container c□a po
my-pod (stdout, trường hợp có nhi□u container) cho kh□i tạo trước c□a một container
kubectl logs -f my-pod # l□y logs c□a pod my-pod (stdout)
kubectl logs -f my-pod -c my-container # l□y logs c□a container my-container trong pod my-pod
(stdout, trường hợp nhi□u container)
kubectl logs -f -l name=myLabel --all-containers # l□y logs c□a t□t c□ các container c□a pod có
nhãn name=myLabel (stdout)
```

10. Tương tác với các Pod đang chạy

```
kubectl run -i --tty busybox --image=busybox -- sh # Chạy pod trong một shell tương tác
kubectl run nginx --image=nginx --restart=Never -n
mynamespace # Chạy pod nginx trong một namespace cụ thể
kubectl run nginx --image=nginx --restart=Never # Chạy pod nginx và ghi spec của nó vào file có
tên pod.yaml
--dry-run -o yaml > pod.yaml

kubectl attach my-pod -i # Đính kèm với container đang chạy
kubectl port-forward my-pod 5000:6000 # Lắng nghe trên cổng 5000 của máy local và chuyển tiếp sa
cổng 6000 trên pod my-pod
kubectl exec my-pod -- ls / # Chạy lệnh trong một pod (trường hợp 1 container)
kubectl exec my-pod -c my-container -- ls / # Chạy lệnh trong pod (trường hợp nhiều container)
kubectl top pod POD_NAME --containers # Hiển thị số liệu của pod và container chạy trong nó
```

11. Tương tác với các node và cụm cluster

```
kubectl cordon my-node # Đánh dấu my-node là không thể lập lịch
kubectl drain my-node # Gỡ my-node ra khỏi cụm để chuẩn bị cho việc bảo trì
kubectl uncordon my-node # Đánh dấu my-node có thể lập lịch trở lại
kubectl top node my-node # Hiển thị số liệu của node
kubectl cluster-info # Hiển thị địa chỉ master và các services
kubectl cluster-info dump # Kết xuất trạng thái hiện tại của cụm ra ngoài stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-state # Kết xuất trạng thái hiện
tại của cụm vào /path/to/cluster-state

kubectl taint nodes foo dedicated=special-user:NoSchedule
```

12. Các loại tài nguyên

Liệt kê tất cả các loại tài nguyên được hỗ trợ cùng với tên viết tắt của chúng, API group, cho dù chúng là namespaced, và Kind:

```
kubectl api-resources
```

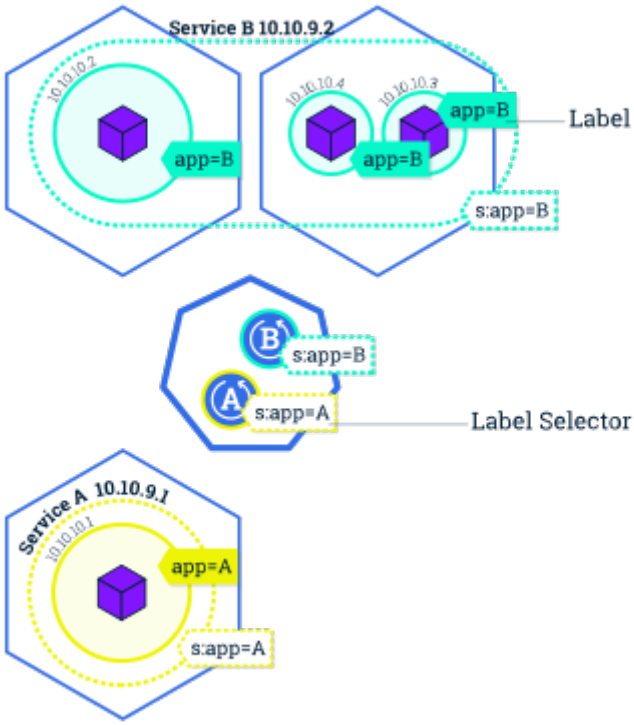
Các hoạt động khác để khám phá các tài nguyên API:

```
kubectl api-resources --namespaced=true # Tất cả các tài nguyên được đặt tên
kubectl api-resources --namespaced=false # Tất cả các tài nguyên không được đặt tên
kubectl api-resources -o name # Tất cả các tài nguyên với đầu ra đơn giản (chỉ gồm tên tài nguyê
```

```
kubectl api-resources -o wide # T□t c□ các tài nguyên với đ□u ra m□ r□ng
kubectl api-resources --verbs=list,get # T□t c□ các tài nguyên h□ tr□ yêu c□u "list" và "get"
kubectl api-resources --api-group=extensions # T□t c□ tài nguyên trong nhóm API "tiện ích m□
r□ng"
```

Kubernetes Label - Nhãn đối tượng trong Kubernetes

Label (nhãn) là cơ chế được sử dụng để tổ chức các đối tượng Kubernetes. Label là một cặp key - value với với một số hạn chế nhất định liên quan đến độ dài và các giá trị được phép nhưng không có bất kỳ ý nghĩa nào được xác định trước. Bạn có thể tự do chọn Label phù hợp hoặc để thể hiện tên, môi trường, quyền sở hữu...



- Tạo Pod với label sử dụng kubectl apply:

```
apiVersion: v1
kind: Pod
metadata:
  name: labelex
  labels:
    env: development
spec:
```

```
  containers:
  - name: sise
    image: quay.io/openshiftlabs/simpleservice:0.5.0
    ports:
    - containerPort: 9876
```

- Hiển thị label của pod vừa tạo:

```
kubectl get pods --show-labels
```

Kết quả:

```
NAME      READY   STATUS    RESTARTS   AGE   LABELS
labelex   1/1     Running   0          6s    env=development
```

- Thêm Label cho pod bằng cách:

```
kubectl label pods labelex owner=michael
```

Kết quả:

```
NAME      READY   STATUS    RESTARTS   AGE   LABELS
labelex   1/1     Running   0          65s   env=development,owner=michael
```

- Filter pod theo label:

```
kubectl get pods --selector owner=michael
```

- Xóa pod theo label:

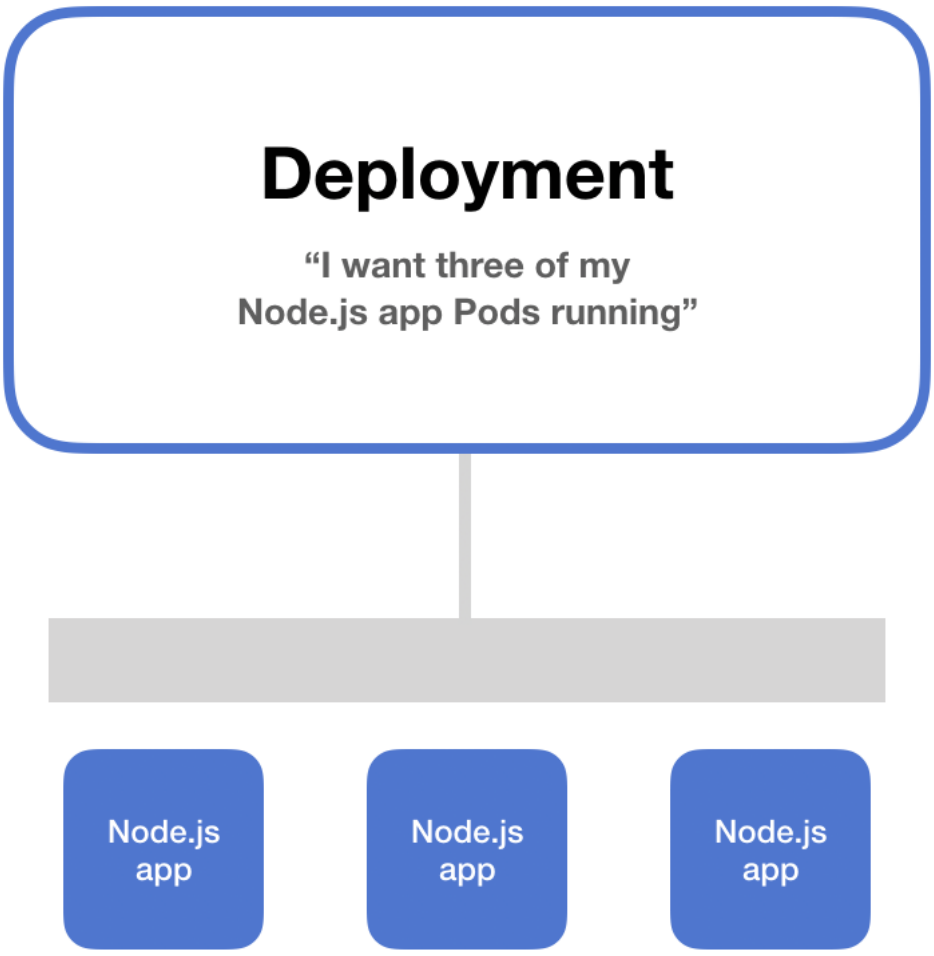
```
kubectl delete pods -l 'env in (production, development)'
```

- Xóa pod sử dụng kubectl delete:

```
kubectl delete pods labelex
kubectl delete pods labelexother
```

Kubernetes Deployment - Triển khai ứng dụng bằng Deployment

Deployment là một supervisor (kịch bản thực hiện) cho các Pod, cung cấp cho bạn quyền kiểm soát chi tiết về cách thức và thời điểm một phiên bản Pod mới được triển khai cũng như quay trở lại trạng thái trước đó.



1. Tạo Kubernetes Deployment

- File: `nginx-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- Giải thích: Trong ví dụ trên:

- Deployment được tạo với tên `nginx-deployment` đã được tạo, được chỉ định bởi trường `metadata.name`
- Deployment được tạo với 3 replicated Pod, được chỉ định bởi trường spec.replicas
- Trường `spec.selector` định nghĩa cách Deployment tìm thấy Pod cần quản lý. Trong trường hợp này, chúng ta sử dụng label của Pod (`app: nginx`). Tuy nhiên, có thể thực hiện các quy tắc lựa chọn phức tạp hơn, miễn là bản thân mẫu Pod thỏa mãn quy tắc.
- Trường template chứa các thông tin sau:
  - template.metadata: Label (nhãn) của Pod `app: nginx` được chỉ định bởi trường `metadata.labels`
  - template.spec: Đặc điểm kỹ thuật của Deployment
    - Trong ví dụ này, Pod có một container, chạy nginx image version 1.14.2 (từ docker hub)
    - name: Tên container
    - image: Docker image
    - ports: Các port của container tương ứng với port của image sau khi chạy

- Thực hiện:

```
kubectl apply -f nginx-deployment.yaml
```

- Kiểm tra:

```
kubectl get deployments
```

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   0/3     0            0           1s
```

Khi bạn kiểm tra Deployment, các trường sau được hiển thị:

- NAME: Liệt kê tên của các Triển khai trong không gian tên.
- READY: Hiển thị số lượng bản sao của ứng dụng có sẵn cho người dùng của bạn. Nó theo mẫu đã sẵn sàng / mong muốn.
- UP-TO-DATE: Hiển thị số lượng bản sao đã được cập nhật để đạt được trạng thái mong muốn.
- AVAILABLE: Hiển thị số lượng bản sao của ứng dụng có sẵn cho người dùng của bạn.
- AGE: Hiển thị lượng thời gian ứng dụng đã chạy.

- Để kiểm tra trạng thái thực hiện của Deployment, thực hiện `kubectl rollout status deployment/nginx-deployment`

```
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
deployment "nginx-deployment" successfully rolled out
```

- Kiểm tra số lượng ReplicaSet được tạo bởi Deployment, thực hiện:

```
kubectl get rs
```

```
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-75675f5897   3         3         3       18s
```

ReplicaSet hiển thị các trường sau:

- NAME: Liệt kê tên của ReplicaSets trong không gian tên.
- DESIRED: Hiển thị số lượng bản sao mong muốn của ứng dụng mà bạn xác định khi tạo Triển khai. Đây là trạng thái mong muốn.
- CURRENT: Hiển thị số lượng bản sao hiện đang chạy.
- READY: Hiển thị số lượng bản sao của ứng dụng có sẵn cho người dùng của bạn.
- AGE: Hiển thị lượng thời gian ứng dụng đã chạy.

- Để kiểm tra Pod được tạo, thực hiện:

```
kubectl get pods --show-labels
```

```
NAME                                READY     STATUS    RESTARTS   AGE       LABELS
nginx-deployment-75675f5897-7ci7o   1/1       Running   0          18s       app=nginx,pod-template-hash=3123191453
nginx-deployment-75675f5897-kzszj   1/1       Running   0          18s       app=nginx,pod-template-hash=3123191453
nginx-deployment-75675f5897-qqcnn   1/1       Running   0          18s       app=nginx,pod-template-hash=3123191453
```

2. Cập nhật Deployment

- Thực hiện cập nhật nginx Pod sử dụng image version `nginx:1.16.1` từ version `nginx:1.14.2`

```
kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.1
```

hoặc sử dụng:

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1
```

Kết quả:

```
deployment.apps/nginx-deployment image updated
```

Tuy nhiên, chúng ta có thể edit Deployment và thay đổi `.spec.template.spec.containers[0].image` từ `nginx:1.14.2` thành `nginx:1.16.1`:

```
kubectl edit deployment/nginx-deployment
```

Kết quả:

```
deployment.apps/nginx-deployment edited
```

- Kiểm tra trạng thái Deployment rollout, ReplicaSet, Pod theo các command ở trên để kiểm tra chi tiết.

3. Kiểm tra chi tiết Deployment

- Kiểm tra chi tiết Deployment, thực hiện:

```
kubectl describe deployments
```

- Kết quả:

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Thu, 30 Nov 2017 10:56:25 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
   Containers:
    nginx:
      Image:        nginx:1.16.1
      Port:         80/TCP
      Environment:  <none>
      Mounts:       <none>
    Volumes:        <none>
  Conditions:
    Type           Status  Reason
    ----           ------  ------
    Available      True    MinimumReplicasAvailable
    Progressing    True    NewReplicaSetAvailable
  OldReplicaSets:  <none>
  NewReplicaSet:   nginx-deployment-1564180365 (3/3 replicas created)
  Events:
    Type    Reason             Age   From                   Message
    ----    ------             ----  ----                   -------
    Normal  ScalingReplicaSet  2m    deployment-controller  Scaled up replica set nginx-
deployment-2035384211 to 3
    Normal  ScalingReplicaSet  24s   deployment-controller  Scaled up replica set nginx-
deployment-1564180365 to 1
    Normal  ScalingReplicaSet  22s   deployment-controller  Scaled down replica set nginx-
deployment-2035384211 to 2
    Normal  ScalingReplicaSet  22s   deployment-controller  Scaled up replica set nginx-
deployment-1564180365 to 2
    Normal  ScalingReplicaSet  19s   deployment-controller  Scaled down replica set nginx-
deployment-2035384211 to 1
    Normal  ScalingReplicaSet  19s   deployment-controller  Scaled up replica set nginx-
```

```
deployment-1564180365 to 3
    Normal  ScalingReplicaSet  14s   deployment-controller  Scaled down replica set nginx-
deployment-2035384211 to 0
```

4. Deployment Rollover

Mỗi khi Deployment controller quan sát thấy một kịch bản triển khai Deployment mới (cập nhật), một ReplicaSet mới được tạo để quản lý các Pod. Nếu Deployment được update, ReplicaSet đang tồn tại có label trùng `.spec.selector` nhưng không trùng `template.spec.selector` sẽ bị scaled down. Cuối cùng, một ReplicaSet mới sẽ được scaled tới .spec.replicas và tất cả các ReplicateSet cũ sẽ bị scaled về 0

Nếu chúng ta cập nhật Deployment trong khi Deployment rollout hiện tại đang trong quá trình thực thi, Deployment sẽ tạo một ReplicaSet mới theo bản cập nhật và bắt đầu scaling up, và ReplicaSet hiện tại sẽ được thêm vào danh sách ReplicaSet cũ để thực hiện scaling down

Ví dụ: Trong trường hợp chúng ta tạo Deployment để tạo 5 replicas của `nginx:1.14.2`, nhưng sau đó thực hiện cập nhật Deployment để tạo 5 replicas của `nginx:1.16.1`, khi mới chỉ 3 replicas của `nginx:1.14.2` đã được tạo. Trong trường hợp này, Deployment sẽ ngay lập tức killing 3 Pod `nginx:1.14.2` (mà không chờ đến lúc tạo đủ 5 pod `nginx:1.14.2`) đã được tạo này, sau đó thực hiện tạo pod `nginx:1.16.1` theo kịch bản Deployment mới.

5. Rolling Back a Deployment

continue..

Hướng dẫn cài đặt Sonatype Nexus 3 trên Kubernetes

Nexus Repository Manager là một công cụ dùng để lưu trữ các thư viện dependencies mà chúng ta cần để sử dụng trong các dự án.

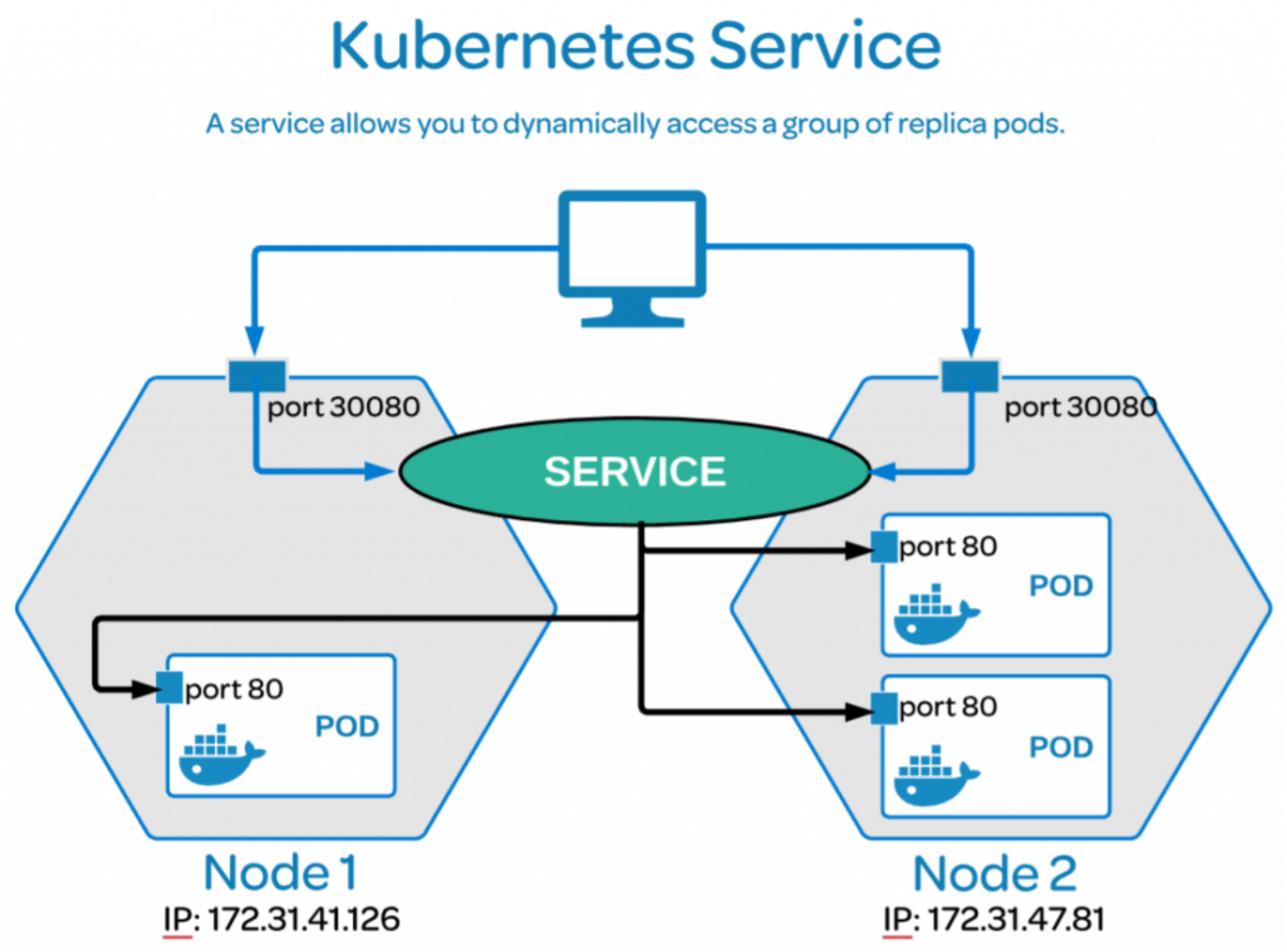
`Deployment.yaml`

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: sonatype-registry-data
spec:
  storageClassName: local-path
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: sonatype-service
```

```
spec:
  ports:
    - name: sonatype-registry
      port: 8081
      protocol: TCP
      targetPort: 8081
  selector:
    app: sonatype-registry
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sonatype-registry
  labels:
    app: sonatype-registry
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sonatype-registry
  template:
    metadata:
      labels:
        app: sonatype-registry
    spec:
      volumes:
        - name: registry-vol
          persistentVolumeClaim:
            claimName: sonatype-registry-data
      containers:
        - image: sonatype/nexus3
          name: nexus
          ports:
            - containerPort: 8081
          volumeMounts:
            - name: registry-vol
              mountPath: /nexus-data
              subPath: sonatype
```

Kubernetes Service - Truy cập Service trong Kubernetes

Service trong Kubernetes là một tài nguyên để quản lý Pod và cung cấp các chính sách truy cập đến các Pod đó.

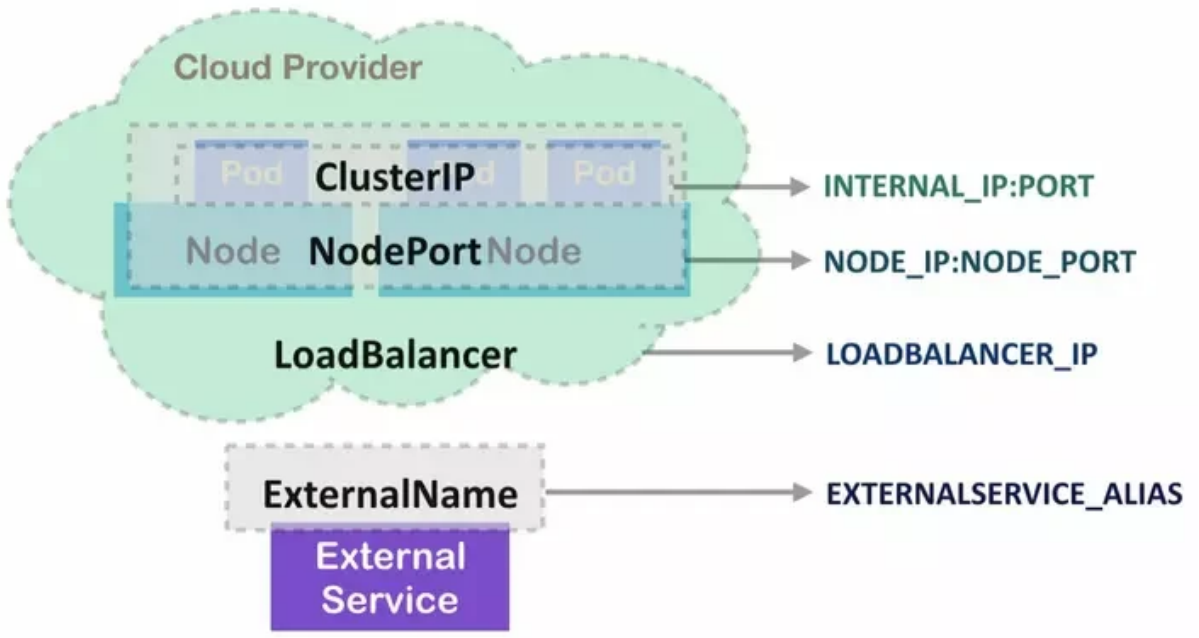


Hiểu một cách đơn giản, để có thể truy cập vào các Pod trên Kubernetes thì cần phải khai báo Service cho nó, các loại Service trong Kubernetes bao gồm:

- **ClusterIP**: Expose 1 port để giao tiếp với Pod trong cluster với internal IP. Loại này thường được sử dụng trong trường hợp Pod chỉ giao tiếp nội bộ với nhau trong cùng cluster (bên

ngoài/end user sẽ không truy cập được). Đây là loại **mặc định** của Service.

- **NodePort**: Expose 1 static public port (mặc định trong khoảng 30000 - 32767) cho Pod trên mỗi Node IP, được hiểu là Kubernetes sẽ NAT port để cho phép người sử dụng có thể truy cập vào các Pod đang được deploy trên Node đó từ bên ngoài bằng địa chỉ: <NodeIP>:<NodePort>. Loại này thường được sử dụng trong trường hợp phục vụ test/thử nghiệm.
- **LoadBalancer**: Expose 1 public port cho Pod trên 1 địa chỉ IP, cho phép người sử dụng truy cập dịch vụ từ bên ngoài. Điểm khác biệt so với NodePort, đó là lưu lượng này sẽ được phân tải cho tất cả các Pod trên các Node theo cơ chế load balancer (cân bằng tải). Loại này thường được sử dụng khi cần cung cấp truy cập đến dịch vụ cho người dùng.
- **ExternalName**: Đây là loại service có cơ chế tách biệt một chút so với 3 loại service phía trên. Loại service này không sử dụng selectors mà thay vào đó lại sử dụng tên DNS. Nó ánh xạ một service với một tên DNS là nội dung của trường externalName (Ví dụ: app.test.com). Khi bạn muốn truy cập vào tên service đó, thay vì trả về cluster-ip của service này, nó sẽ trả về bản ghi CNAME với giá trị được đề cập trong externalName



Cùng tìm hiểu cách khai báo các Service trong Kubernetes theo ví dụ sau:

- Tạo Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
```

```
spec:
  containers:
    - name: nginx-container
      image: nginx
```

- Khai báo Service ClusterIP:

```
apiVersion: v1
kind: Service
metadata:
  name: front-end-service
spec:
  type: ClusterIP
  selector:
    app: myapp
    type: front-end
  ports:
    - targetPort: 80
      port: 80
```

- Khai báo Service NodePort:

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  selector:
    app: myapp
    type: front-end
  ports:
    - targetPort: 80
      port: 80
      nodePort: 32593
```

- Khai báo Service LoadBalancer:

```
apiVersion: v1
kind: Service
metadata:
```

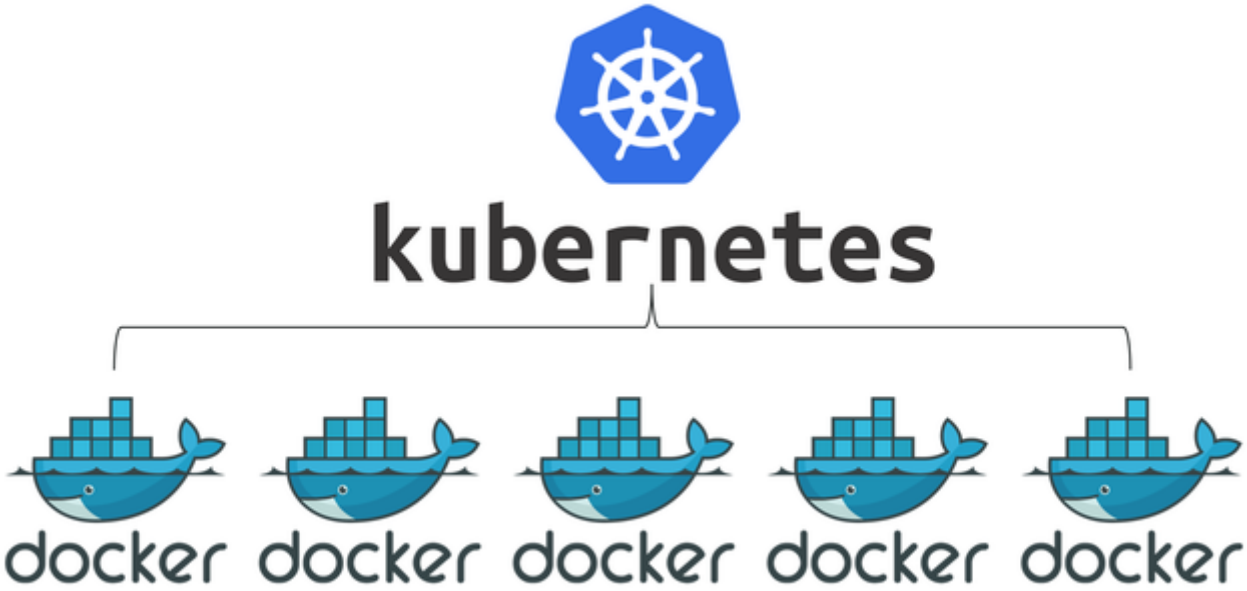
```
  name: my-service
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 9376
      nodePort: 31000
  selector:
    app: myapp
```

- Khai báo Service ExternalName:

```
apiVersion: v1
kind: Service
metadata:
  name: "google-service"
spec:
  ports:
    - port: 80
  type: ExternalName
  externalName: google.com
```

Tham khảo: https://kubernetes.io/docs/concepts/services-networking/service/

Hướng dẫn cài đặt Kubernetes trên Linux



Để cài đặt một Kubernetes cluster trên Linux, bạn cần thực hiện các bước sau:

Bước 1: Chuẩn bị môi trường

- Hệ điều hành: Ubuntu hoặc CentOS.
- Cài đặt Docker để có thể chạy các container trên các node của cluster.

Bước 2: Cài đặt Kubernetes Master Node

1. Thực hiện thành phần chính của Kubernetes trên Master Node.

- Cài đặt Kubernetes Control Plane (kubectl, kube-apiserver, kube-controller-manager, kube-scheduler):

- Truy cập vào Master Node và mở Terminal.
- Cài đặt các gói bằng lệnh apt (Ubuntu) hoặc yum (CentOS):

```
# Ubuntu
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

```
    curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
    echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
    sudo apt-get update
    sudo apt-get install -y kubelet kubeadm kubectl

    # CentOS
    sudo yum install -y yum-utils device-mapper-persistent-data lvm2
    sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
    sudo yum install -y docker-ce
    sudo systemctl start docker
    sudo systemctl enable docker
    sudo systemctl status docker
    sudo systemctl enable kubelet
    sudo systemctl start kubelet
```

- Thiết lập Kubernetes Master Node: Khởi tạo Master Node bằng lệnh `kubeadm init`:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

Lưu lại đầu ra cuối cùng của lệnh này, bởi vì nó chứa thông tin xác thực cần thiết để kết nối các Worker Node với Master Node.

- Thiết lập môi trường cho `kubectl`:

```
    mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- Cài đặt một mạng Pod (ví dụ: Flannel):

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

- Xác minh trạng thái của Master Node:

```
kubectl get nodes
```

Nếu mọi thứ đều đúng, Master Node sẽ được hiển thị làm "Ready".

Bước 3: Cài đặt Kubernetes Worker Nodes

1. Thực hiện cài đặt các thành phần của Kubernetes trên Worker Nodes.

- Cài đặt Kubernetes Worker Node:

- Truy cập vào Worker Node và mở Terminal.
- Cài đặt các gói giống như trên Master Node (bước 2.1).

- Tham gia vào Kubernetes Cluster: Sử dụng thông tin xác thực từ đầu ra của lệnh `kubeadm init` trên **Master Node** đã thực hiện để tham gia vào cluster:

```
sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --discovery-token-ca-cert-hash <hash>
```

Thay thế `<master-node-ip>:<master-node-port>`, `<token>`, và `<hash>` bằng các thông tin tương ứng từ đầu ra của lệnh `kubeadm init`.

- Xác minh trạng thái của các Worker Nodes: Trên Master Node, chạy lệnh:

```
kubectl get nodes
```

Các Worker Node đã tham gia vào cluster sẽ hiển thị "Ready".

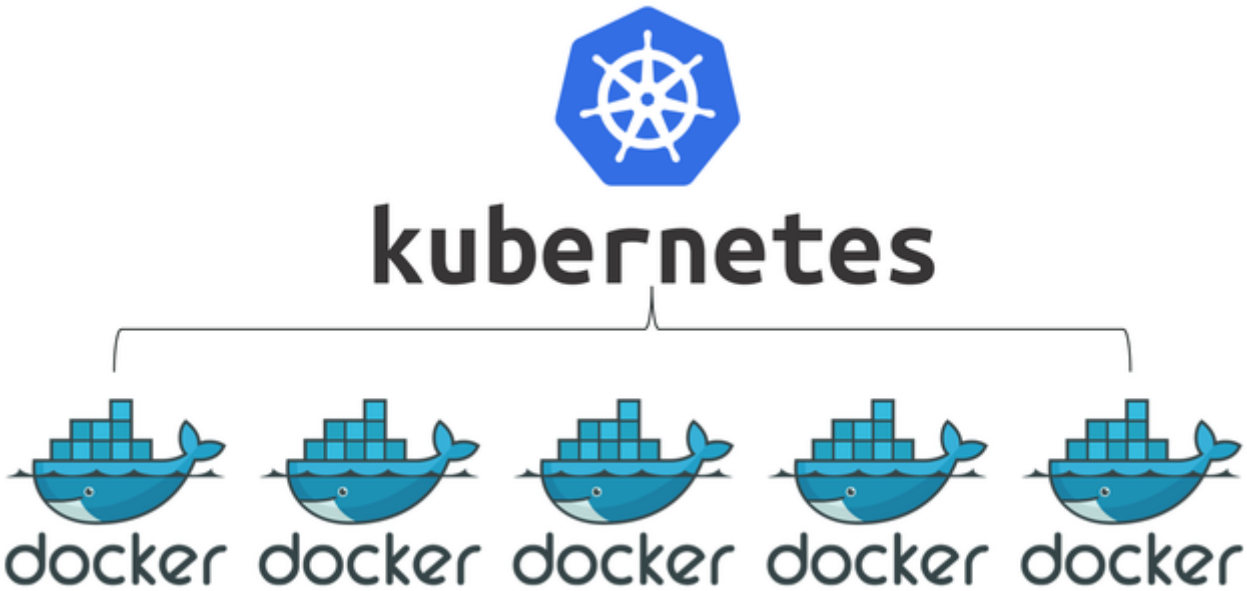
Bước 4: Kiểm tra cluster

- Bây giờ bạn đã có một Kubernetes cluster hoạt động trên Linux. Bạn có thể kiểm tra trạng thái của cluster bằng cách chạy các lệnh `kubectl`. Tham khảo các lệnh kubectl tại đây

Với các bước trên, bạn đã cài đặt thành công một Kubernetes cluster trên Linux. Lưu ý rằng đây chỉ là hướng dẫn cơ bản và có thể cần điều chỉnh tùy thuộc vào phiên bản Linux và các yêu cầu cụ thể của bạn.

Kubernetes - Job

Kubernetes Job là một đối tượng được sử dụng để chạy các tác vụ hoặc công việc một lần trong Kubernetes. Các công việc này được thiết lập để bắt đầu và kết thúc một lần duy nhất, không giống như các Pod thông thường, các Pod đóng vai trò để chạy các ứng dụng liên tục.



Trong một Job, Kubernetes sẽ tạo ra một hoặc nhiều Pod để chạy các công việc cần thiết. Mỗi Pod sẽ được tạo ra và chạy cho đến khi công việc được hoàn thành. Sau đó, Pod sẽ tự động bị xóa bỏ. Tuy nhiên, nếu Pod bị gián đoạn hoặc không thể hoàn thành công việc, Kubernetes sẽ tạo ra một Pod mới để tiếp tục công việc.

Các Job được sử dụng phổ biến trong các trường hợp sau:

- Tạo ra dữ liệu mẫu hoặc dữ liệu test
- Xử lý các tác vụ batch hoặc các tác vụ dữ liệu lớn
- Backup hoặc khôi phục dữ liệu
- Cập nhật dữ liệu

Để tạo một Kubernetes Job, bạn cần định nghĩa một file YAML chứa thông tin về Job, bao gồm cả các Pod cần được tạo ra. Sau đó, bạn chạy lệnh `kubectl apply -f <tên file YAML>` để triển khai Job.

Ví dụ dưới đây là một file YAML đơn giản để tạo một Job trong Kubernetes:

```
apiVersion: batch/v1
kind: Job
metadata:
```

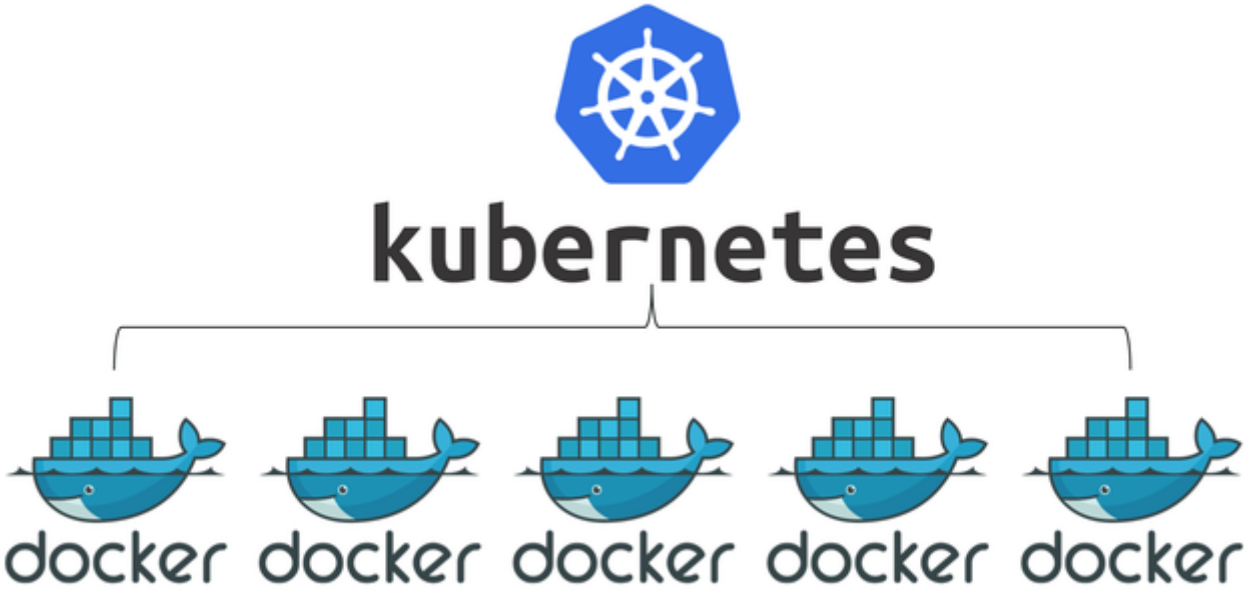
```
  name: my-job
spec:
  template:
    spec:
      containers:
      - name: my-container
        image: busybox
        command: ["echo", "Hello from the Kubernetes Job!"]
      restartPolicy: Never
  backoffLimit: 4
```

Trong ví dụ này, Job sẽ tạo một Pod chạy một container với hình ảnh BusyBox, và thực hiện lệnh `echo "Hello from the Kubernetes Job!"`. Sau khi công việc hoàn thành, Pod sẽ bị xóa bỏ và Job sẽ được đánh dấu là đã hoàn thành.

Bạn có thể kiểm tra trạng thái của các Job bằng lệnh `kubectl get jobs`. Nếu bạn muốn xem chi tiết về một Job cụ thể, bạn có thể sử dụng lệnh `kubectl describe job <tên job>`.

Đó là những chi tiết cơ bản về Kubernetes Job. Nếu bạn muốn tìm hiểu thêm về các đối tượng và tính năng khác của Kubernetes, hãy tham khảo tài liệu chính thức của Kubernetes.

Kubernetes - Labels & Selectors



Labels là các cặp key/value có thể được gắn vào các đối tượng Kubernetes, chẳng hạn như các Pod, Services và Deployments. Labels được sử dụng để tổ chức và chọn các đối tượng trong một cluster Kubernetes.

Selectors được sử dụng để lọc các đối tượng dựa trên các Labels của chúng. Selectors có thể được sử dụng để chọn các đối tượng cho các hành động như mở rộng hoặc thu nhỏ một Deployment, hoặc để định hướng lưu lượng mạng đến một Service cụ thể.

Labels và Selectors hoạt động cùng nhau để tạo ra một hệ thống mạnh mẽ để tổ chức và quản lý các đối tượng trong một cluster Kubernetes.

Cú pháp Labels

Labels bao gồm một key và một value, được phân tách bởi dấu bằng (=). Ví dụ:

```
app=nginx
```

Labels cũng có thể được gán cho các đối tượng Kubernetes bằng cú pháp YAML, như sau:

```
metadata:
  labels:
    app: nginx
```

Cú pháp Selectors

Selectors được sử dụng để lọc các đối tượng dựa trên các Labels của chúng. Selectors có thể được sử dụng để chọn các đối tượng cho các hành động như mở rộng hoặc thu nhỏ một Deployment, hoặc để định hướng lưu lượng mạng đến một Service cụ thể.

Selectors được chỉ định bằng cú pháp tương tự như Labels. Ví dụ, một selector chọn tất cả các đối tượng với Label "app=nginx" sẽ trông như sau:

```
app=nginx
```

Selectors cũng có thể được sử dụng để chọn các đối tượng dựa trên nhiều Labels. Ví dụ, một selector chọn tất cả các đối tượng với Label "app=nginx" và Label "environment=production" sẽ trông như sau:

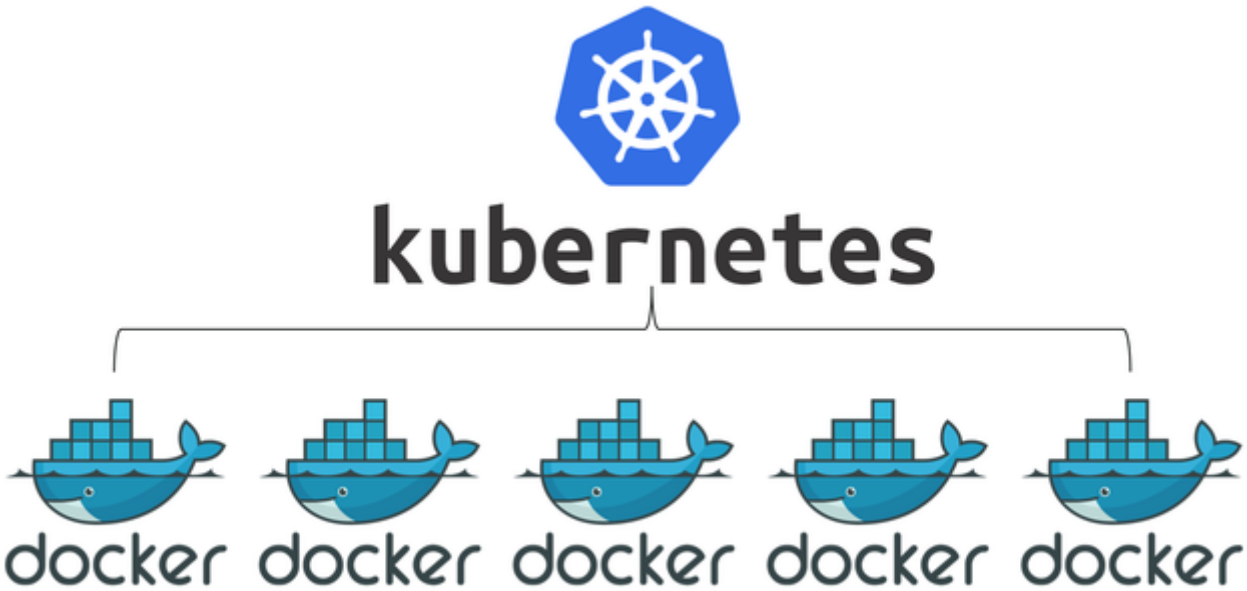
```
app=nginx,environment=production
```

Kết luận

Labels và Selectors là những công cụ mạnh mẽ để tổ chức và quản lý các đối tượng trong một cluster Kubernetes. Bằng cách sử dụng Labels và Selectors, bạn có thể dễ dàng nhóm các đối tượng liên quan với nhau và chọn chúng cho các hành động cụ thể.

Kubernetes - Namespace

Namespace trong Kubernetes được sử dụng để phân chia cụm Kubernetes thành các phần riêng biệt, giúp quản lý các đối tượng trong cụm trở nên dễ dàng hơn. Mỗi namespace chứa các đối tượng Kubernetes, chẳng hạn như Pod, Service và Deployment, được phân biệt với các đối tượng trong các namespace khác.



Kubernetes sử dụng một namespace mặc định, gọi là namespace mặc định. Nếu không chỉ định namespace cho một đối tượng, nó sẽ được tạo trong namespace mặc định.

Bạn có thể tạo namespace mới bằng lệnh `kubectl create namespace <tên namespace>`. Sau đó, bạn có thể chuyển đổi giữa các namespace bằng lệnh `kubectl config set-context --current --namespace=<tên namespace>`.

Ví dụ dưới đây là một file YAML đơn giản để tạo một namespace trong Kubernetes:

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
```

Trong ví dụ này, một namespace có tên là my-namespace sẽ được tạo.

Ví dụ: sau đây minh họa cách sử dụng namespace trong Kubernetes để phân chia ứng dụng thành các phần riêng biệt.

Giả sử bạn có một ứng dụng web đơn giản gồm hai thành phần: một Pod chạy Apache và một Service để định tuyến lưu lượng đến Pod.

Bạn muốn triển khai ứng dụng này trong hai môi trường: môi trường sản xuất và môi trường kiểm thử. Bạn muốn cả hai môi trường này hoạt động trên cùng một cụm Kubernetes, nhưng bạn muốn chúng được phân biệt với nhau bằng namespace.

Đầu tiên, bạn tạo hai namespace mới bằng lệnh `kubectl create namespace production` và `kubectl create namespace staging`.

Sau đó, bạn định nghĩa một file YAML để triển khai ứng dụng trong môi trường sản xuất:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: production
spec:
  selector:
    app: my-app
  ports:
  - name: http
    port: 80
    targetPort: 80
---
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: production
spec:
  containers:
  - name: my-container
    image: httpd
    ports:
    - containerPort: 80
    env:
    - name: ENVIRONMENT
      value: production
```

Trong file YAML này, một Service và một Pod sẽ được triển khai trong namespace production. Service được định nghĩa để định tuyến lưu lượng đến Pod, trong khi Pod chạy một container Apache

và định nghĩa biến môi trường ENVIRONMENT với giá trị production.

Nếu bạn muốn triển khai ứng dụng trong môi trường kiểm thử, bạn có thể sửa file YAML trên để sử dụng namespace staging thay vì production.

Kubernetes - Service

Kubernetes Service là một tài nguyên quan trọng để định tuyến lưu lượng trong cụm Kubernetes. Một Service trong Kubernetes cung cấp một địa chỉ IP ổn định cho một tập hợp các Pod, cho phép các Pod trong Service truy cập và tương tác với nhau bằng cách sử dụng địa chỉ IP này.

Một Service trong Kubernetes được định nghĩa bằng một file YAML, bao gồm các thông tin như tên Service, tập hợp các Pod được liên kết với Service, cổng và giao thức sử dụng để truy cập Service.

Bạn có thể tạo một Service mới bằng lệnh `kubectl create service <tên service>`. Sau đó, bạn có thể sửa file YAML của Service bằng lệnh `kubectl edit service <tên service>`.

Ví dụ: sau đây minh họa cách sử dụng Service trong Kubernetes để định tuyến lưu lượng đến các Pod trong cụm.

Giả sử bạn có một ứng dụng web đơn giản gồm hai thành phần: một Pod chạy Apache và một Pod chạy MySQL. Bạn muốn định tuyến lưu lượng đến Pod Apache từ bên ngoài cụm Kubernetes và định tuyến lưu lượng từ Pod Apache đến Pod MySQL. Bạn muốn các Pod này được triển khai trên cùng một Node trong cụm Kubernetes.

Đầu tiên, bạn tạo hai Pod mới trong cụm Kubernetes bằng lệnh `kubectl run <tên pod> --image=<tên image>`. Sau đó, bạn tạo một file YAML để triển khai Service cho các Pod này:

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

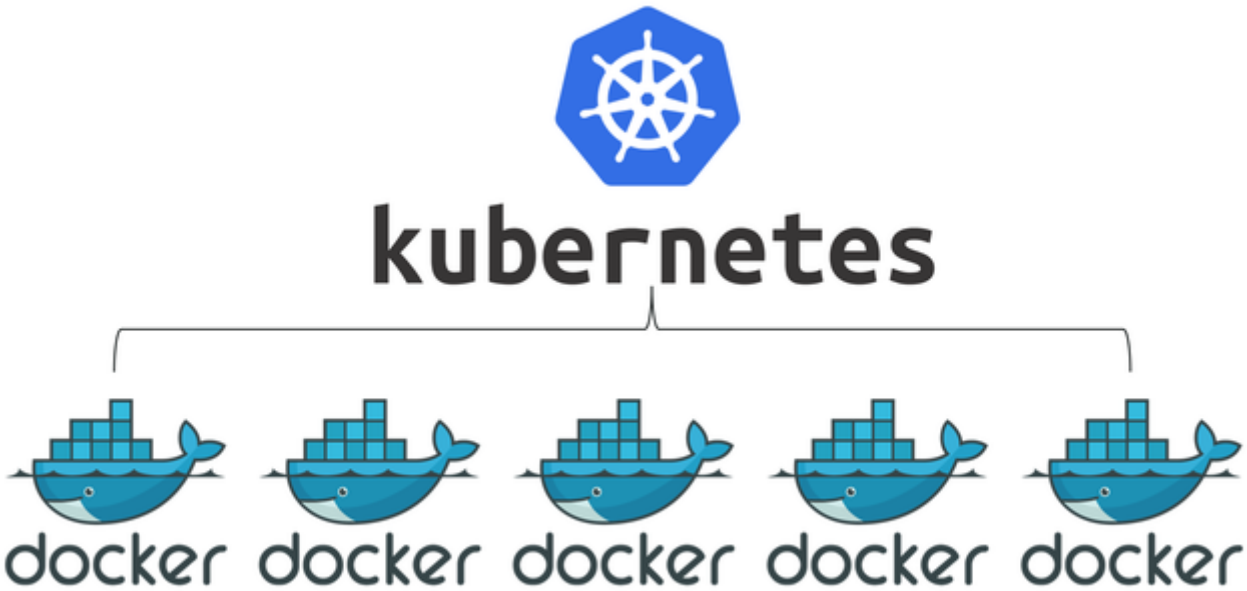
Trong file YAML này, một Service có tên là my-service sẽ được tạo. Service này sẽ định tuyến lưu lượng đến các Pod được liên kết với Service bằng cách sử dụng selector `app: my-app`. Cổng 80 sẽ được sử dụng để truy cập Service.

Sau đó, bạn triển khai file YAML này bằng lệnh `kubectl apply -f <tên file YAML>`.

Mong rằng thông tin này sẽ giúp bạn hiểu rõ hơn về Kubernetes Service. Hãy để lại câu hỏi nếu cần thêm thông tin.

Kubernetes - Replication Controller

Replication Controller trong Kubernetes là một tài nguyên quan trọng để quản lý số lượng Pod trong cụm Kubernetes. Nó đảm bảo rằng số lượng Pod đang chạy trên cụm Kubernetes luôn đúng với số lượng được chỉ định.



Một Replication Controller trong Kubernetes được định nghĩa bằng một file YAML, bao gồm các thông tin như tên Replication Controller, số lượng Pod được yêu cầu, template Pod, và các thông tin khác.

Bạn có thể tạo một Replication Controller mới bằng lệnh `kubectl create replicationcontroller <tên Replication Controller>`. Sau đó, bạn có thể sửa file YAML của Replication Controller bằng lệnh `kubectl edit replicationcontroller <tên Replication Controller>`.

Ví dụ: sau đây minh họa cách sử dụng Replication Controller trong Kubernetes để quản lý số lượng Pod trong cụm.

Giả sử bạn có một ứng dụng web đơn giản gồm một Pod chạy Apache. Bạn muốn triển khai ứng dụng này trên cụm Kubernetes và đảm bảo rằng luôn có hai Pod đang chạy trong cụm.

Đầu tiên, bạn tạo một file YAML để định nghĩa Replication Controller cho ứng dụng của mình:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: my-controller
spec:
  replicas: 2
  selector:
    app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: httpd
        ports:
        - containerPort: 80
        env:
        - name: ENVIRONMENT
          value: production
```

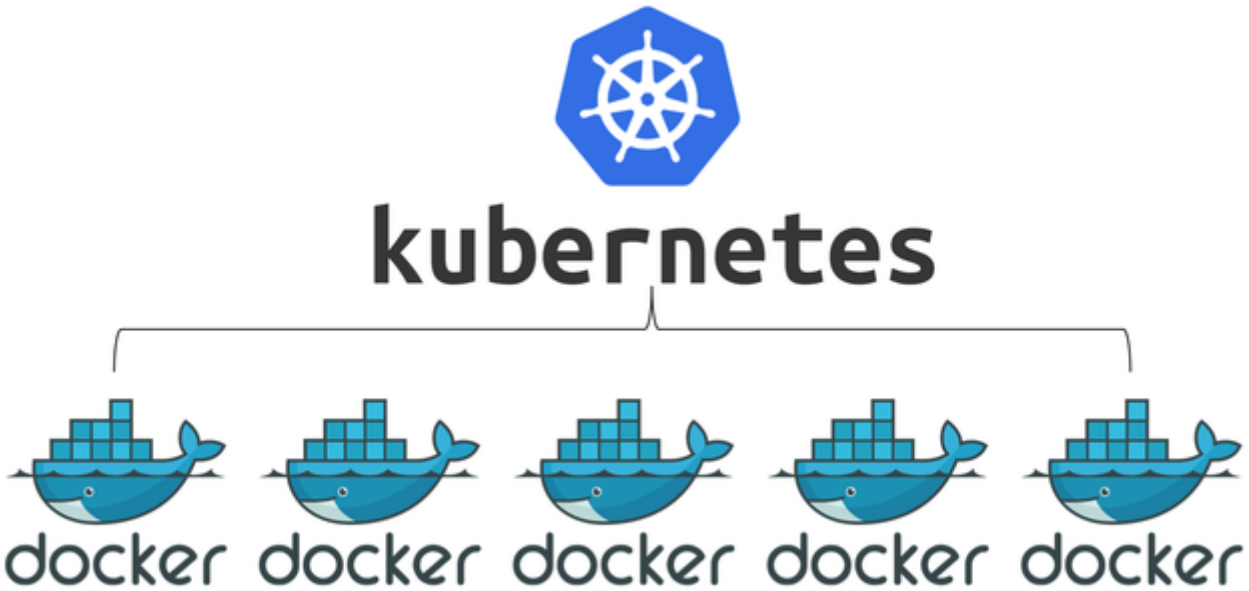
Trong file YAML này, một Replication Controller có tên là my-controller sẽ được tạo. Replication Controller này sẽ đảm bảo rằng luôn có hai Pod đang chạy trong cụm Kubernetes. Các Pod được liên kết với Replication Controller bằng cách sử dụng selector `app: my-app`. Mỗi Pod sẽ chạy một container Apache và định nghĩa biến môi trường ENVIRONMENT với giá trị production.

Sau đó, bạn triển khai file YAML này bằng lệnh `kubectl apply -f <tên file YAML>`.

Mong rằng thông tin này sẽ giúp bạn hiểu rõ hơn về Replication Controller trong Kubernetes. Hãy để lại câu hỏi nếu cần thêm thông tin.

Kubernetes - Replica Sets

Replica Sets trong Kubernetes là một tài nguyên quan trọng để quản lý số lượng Pod trong cụm Kubernetes. Nó đảm bảo rằng số lượng Pod đang chạy trên cụm Kubernetes luôn đúng với số lượng được chỉ định.



Một Replica Set trong Kubernetes được định nghĩa bằng một file YAML, bao gồm các thông tin như tên Replica Set, số lượng Pod được yêu cầu, template Pod, và các thông tin khác.

Bạn có thể tạo một Replica Set mới bằng lệnh `kubectl create replicasets <tên Replica Set>`. Sau đó, bạn có thể sửa file YAML của Replica Set bằng lệnh `kubectl edit replicasets <tên Replica Set>`.

Ví dụ: sau đây minh họa cách sử dụng Replica Sets trong Kubernetes để quản lý số lượng Pod trong cụm.

Giả sử bạn có một ứng dụng web đơn giản gồm một Pod chạy Apache. Bạn muốn triển khai ứng dụng này trên cụm Kubernetes và đảm bảo rằng luôn có hai Pod đang chạy trong cụm.

Đầu tiên, bạn tạo một file YAML để định nghĩa Replica Set cho ứng dụng của mình:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 2
```

```
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: httpd
        ports:
        - containerPort: 80
        env:
        - name: ENVIRONMENT
          value: production
```

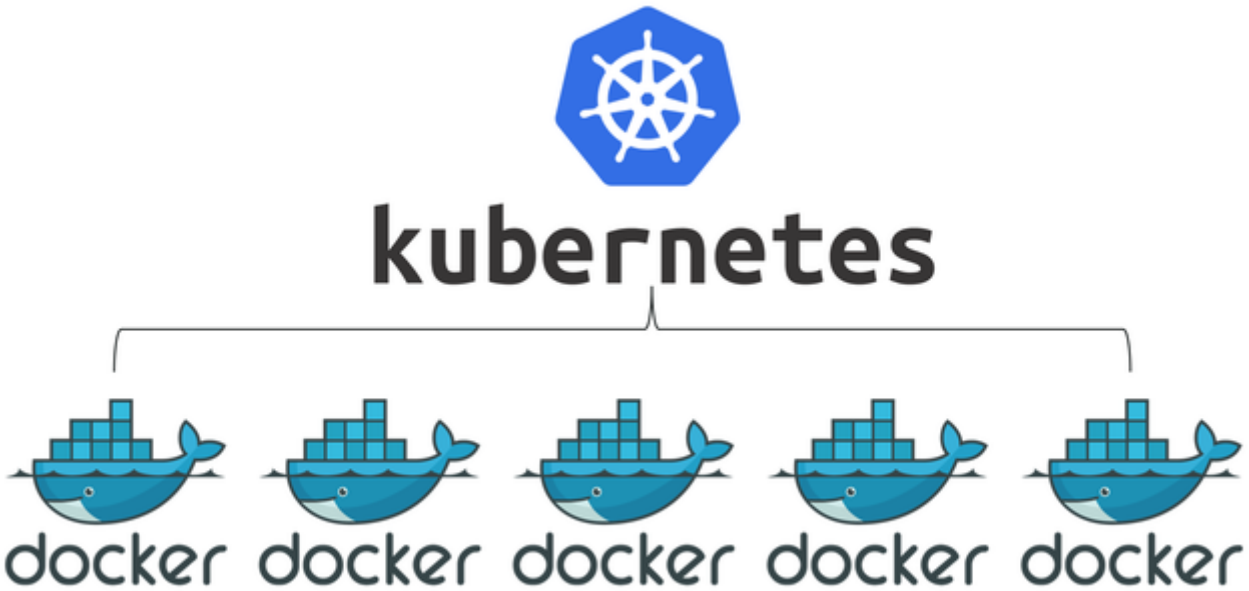
Trong file YAML này, một Replica Set có tên là my-replicaset sẽ được tạo. Replica Set này sẽ đảm bảo rằng luôn có hai Pod đang chạy trong cụm Kubernetes. Các Pod được liên kết với Replica Set bằng cách sử dụng selector `app: my-app`. Mỗi Pod sẽ chạy một container Apache và định nghĩa biến môi trường ENVIRONMENT với giá trị production.

Sau đó, bạn triển khai file YAML này bằng lệnh `kubectl apply -f <tên file YAML>`.

Mong rằng thông tin này sẽ giúp bạn hiểu rõ hơn về Replica Sets trong Kubernetes. Hãy để lại câu hỏi nếu cần thêm thông tin.

Kubernetes - Volume

Volume trong Kubernetes là một tài nguyên quan trọng để quản lý lưu trữ dữ liệu trong các Pod. Nó cho phép các Pod chia sẻ dữ liệu với nhau và lưu trữ dữ liệu khi Pod bị xóa và khởi chạy lại.



Một Volume trong Kubernetes được định nghĩa bằng một file YAML, bao gồm các thông tin như tên Volume, loại Volume, và các thông tin khác. Có nhiều loại Volume khác nhau trong Kubernetes, bao gồm:

- emptyDir: một Volume trống được tạo ra khi một Pod được khởi động và bị xóa khi Pod bị xóa.
- hostPath: một Volume được gắn kết với một đường dẫn trên máy chủ, cho phép các Pod truy cập dữ liệu trực tiếp trên máy chủ.
- persistentVolumeClaim: một Volume được yêu cầu từ cụm để sử dụng với một PersistentVolume.

Bạn có thể tạo một Volume mới bằng lệnh `kubectl create -f <tên file YAML>`. Sau đó, bạn có thể sửa file YAML của Volume bằng lệnh `kubectl edit <tên Volume>`.

Ví dụ: Sau đây minh họa cách sử dụng Volume trong Kubernetes để chia sẻ dữ liệu giữa hai Pod.

Giả sử bạn có hai Pod chạy trong cùng một cụm Kubernetes. Pod đầu tiên là một Pod Apache chứa tất cả các tệp tin cần thiết để chạy một trang web đơn giản. Pod thứ hai là một Pod MySQL chứa một cơ sở dữ liệu MySQL. Bạn muốn chia sẻ dữ liệu trang web giữa hai Pod này, để trang web có thể truy cập cơ sở dữ liệu.

Đầu tiên, bạn tạo một Volume và định nghĩa nó trong file YAML như sau:

```
apiVersion: v1
kind: Volume
metadata:
  name: my-volume
spec:
  emptyDir: {}
```

Sau đó, bạn sửa file YAML của Pod Apache để sử dụng Volume này bằng cách thêm đoạn mã sau:

```
spec:
  containers:
    - name: apache
      image: httpd
      volumeMounts:
        - name: my-volume
          mountPath: /var/www/html
  volumes:
    - name: my-volume
      emptyDir: {}
```

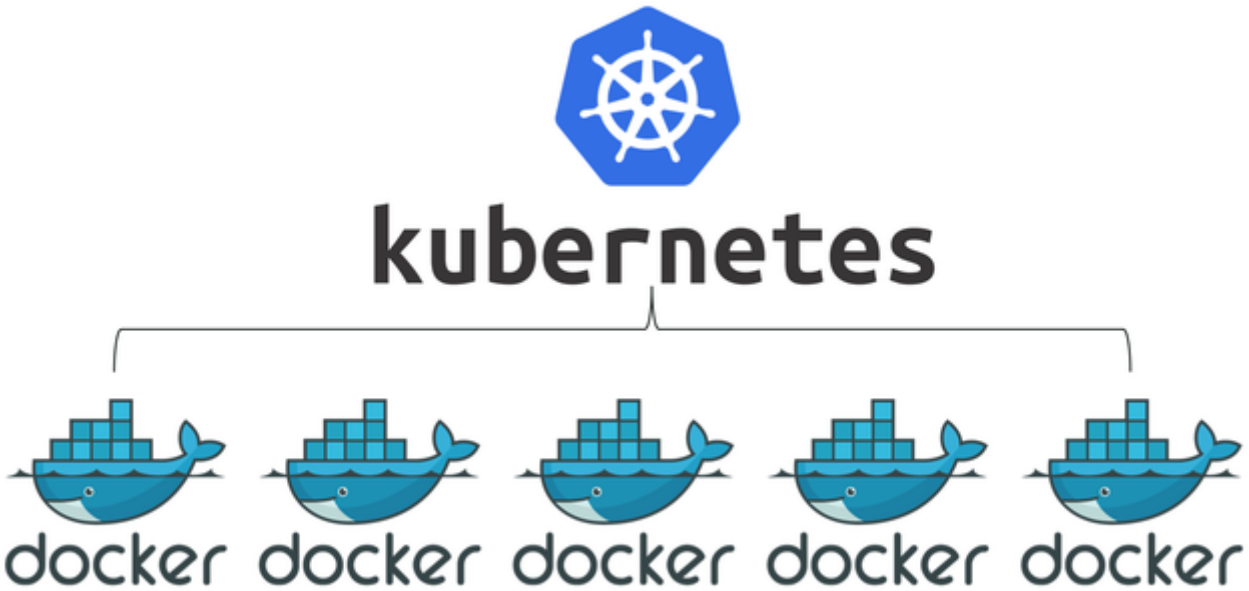
Cuối cùng, bạn sửa file YAML của Pod MySQL để sử dụng Volume này bằng cách thêm đoạn mã sau:

```
spec:
  containers:
    - name: mysql
      image: mysql
      volumeMounts:
        - name: my-volume
          mountPath: /var/lib/mysql
  volumes:
    - name: my-volume
      emptyDir: {}
```

Sau khi triển khai các file YAML này, Pod Apache và Pod MySQL sẽ có thể truy cập cùng một Volume và chia sẻ dữ liệu trang web.

Kubernetes - Secret

Secret trong Kubernetes là một tài nguyên quan trọng để quản lý các thông tin nhạy cảm như mật khẩu, chứng chỉ TLS, hoặc khóa SSH.



Một Secret trong Kubernetes được định nghĩa bằng một file YAML, bao gồm các thông tin như tên Secret, dữ liệu nhạy cảm được mã hóa, và các thông tin khác. Có nhiều loại Secret khác nhau trong Kubernetes, bao gồm:

- Opaque: loại Secret mặc định được sử dụng để lưu trữ các dữ liệu nhạy cảm không được mã hóa.
- kubernetes.io/tls: loại Secret được sử dụng để lưu trữ các chứng chỉ TLS.
- kubernetes.io/dockerconfigjson: loại Secret được sử dụng để lưu trữ thông tin đăng nhập Docker.

Bạn có thể tạo một Secret mới bằng lệnh `kubectl create secret <loại Secret> <tên Secret> --from-literal=<tên khóa>=<giá trị>` hoặc `kubectl create -f <tên file YAML>`. Sau đó, bạn có thể sửa file YAML của Secret bằng lệnh `kubectl edit secret <tên Secret>`.

Ví dụ: Sau đây minh họa cách sử dụng Secret trong Kubernetes để lưu trữ mật khẩu của cơ sở dữ liệu.

Giả sử bạn có một cơ sở dữ liệu MySQL chạy trên cụm Kubernetes và bạn muốn đặt mật khẩu của cơ sở dữ liệu này vào một Secret để tránh lộ thông tin nhạy cảm. Đầu tiên, bạn tạo một Secret mới bằng lệnh sau:

```
kubectl create secret generic mysql-pass --from-literal=password=MY_PASSWORD
```

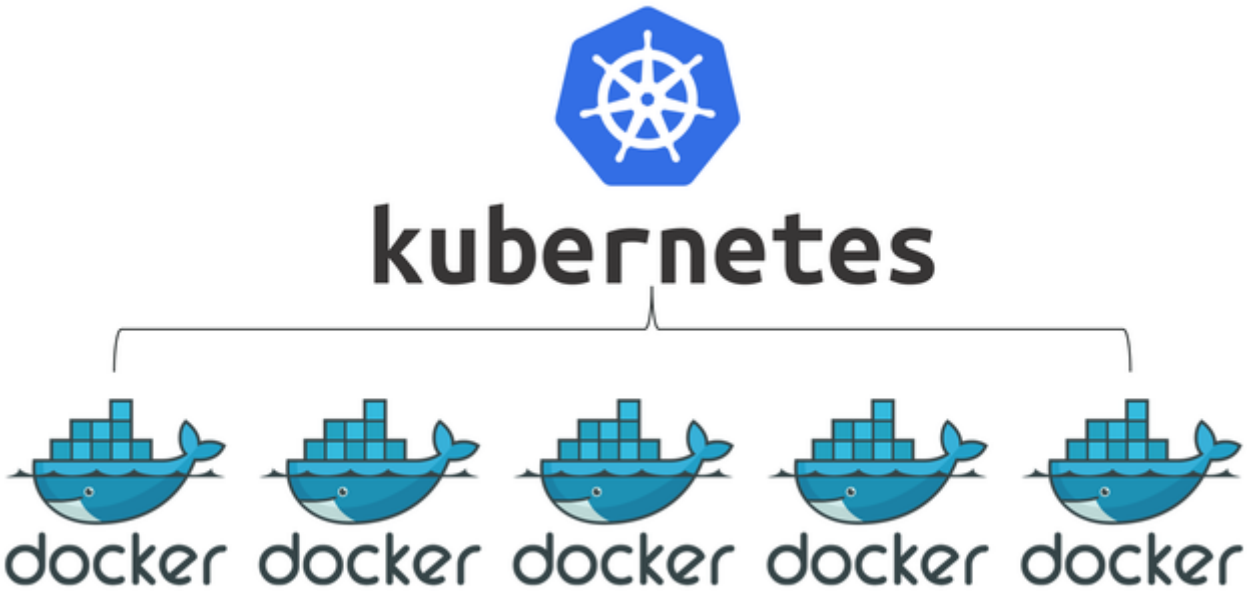
Trong đó `mysql-pass` là tên của Secret và `MY_PASSWORD` là mật khẩu của cơ sở dữ liệu. Sau đó, bạn sửa file YAML của Pod MySQL để sử dụng mật khẩu từ Secret bằng cách thêm đoạn mã sau:

```
spec:
  containers:
    - name: mysql
      image: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
```

Trong đó `mysql-pass` là tên của Secret và `password` là tên của khóa chứa mật khẩu. Sau khi triển khai file YAML này, Pod MySQL sẽ sử dụng mật khẩu từ Secret để đăng nhập vào cơ sở dữ liệu.

Kubernetes - Network Policy

Network Policy là một tài nguyên quan trọng trong Kubernetes để quản lý luồng dữ liệu giữa các Pod và Service. Nó cho phép bạn định nghĩa các quy tắc để kiểm soát truy cập từ các Pod khác nhau trong cùng một cụm Kubernetes.



Một Network Policy được định nghĩa bằng một file YAML, bao gồm các thông tin như tên Network Policy, các quy tắc truy cập và các thông tin khác. Các quy tắc truy cập sử dụng các kết hợp của các giá trị sau:

- `podSelector`: định nghĩa các Pod mà Network Policy này áp dụng.
- `policyTypes`: định nghĩa các loại truy cập được phép, bao gồm Ingress và Egress.
- `ingress`: định nghĩa các quy tắc truy cập Ingress.
- `egress`: định nghĩa các quy tắc truy cập Egress.

Bạn có thể tạo một Network Policy mới bằng lệnh `kubectl create -f <tên file YAML>`. Sau đó, bạn có thể sửa file YAML của Network Policy bằng lệnh `kubectl edit networkpolicy <tên Network Policy>`.

Ví dụ: Sau đây minh họa cách sử dụng Network Policy để kiểm soát truy cập giữa các Pod trong cùng một cụm Kubernetes.

Giả sử bạn có hai Pod trong cùng một cụm Kubernetes: Pod A và Pod B. Bạn muốn chỉ cho phép Pod A truy cập Pod B qua cổng 80. Để làm điều này, bạn tạo một Network Policy với các quy tắc truy cập Ingress như sau:

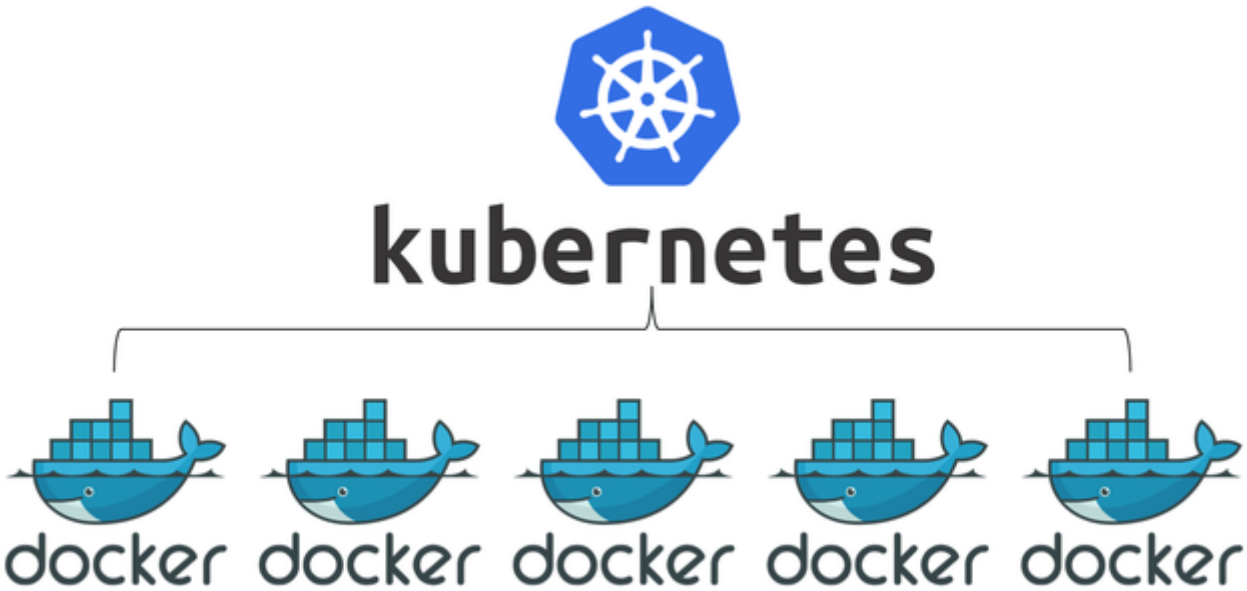
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-pod-a-to-pod-b
spec:
  podSelector:
    matchLabels:
      app: pod-b
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: pod-a
    ports:
    - protocol: TCP
      port: 80
```

Trong file YAML này, một Network Policy có tên là `allow-pod-a-to-pod-b` sẽ được tạo. Policy này chỉ cho phép Pod A truy cập Pod B qua cổng 80. Các Pod được liên kết với Network Policy bằng cách sử dụng selector `app: pod-a` và `app: pod-b`.

Sau khi triển khai file YAML này, Network Policy sẽ được áp dụng và chỉ cho phép Pod A truy cập Pod B qua cổng 80.

Kubernetes - Autoscaling

Autoscaling là một tính năng quan trọng trong Kubernetes cho phép tự động điều chỉnh số lượng Pod để đáp ứng với tải lưu lượng truy cập đến ứng dụng. Nó giúp tối ưu hóa sử dụng tài nguyên và đảm bảo rằng ứng dụng luôn sẵn sàng phục vụ người dùng.



Có hai loại autoscaling trong Kubernetes: Horizontal Pod Autoscaling (HPA) và Vertical Pod Autoscaling (VPA).

Horizontal Pod Autoscaling (HPA)

HPA được sử dụng để điều chỉnh số lượng Pod của một Deployment hoặc ReplicationController dựa trên tải lưu lượng truy cập đến ứng dụng. Nó hoạt động bằng cách sử dụng các metrics như CPU utilization hoặc custom metrics để quyết định số lượng Pod cần tạo ra hoặc xóa bỏ.

Để sử dụng HPA, bạn cần thực hiện các bước sau:

1. Cài đặt các metrics server để thu thập thông tin về tài nguyên và lưu lượng truy cập.
2. Định nghĩa một HPA cho Deployment hoặc ReplicationController của bạn bằng một file YAML.
3. Khi tải lưu lượng truy cập đến ứng dụng tăng, HPA sẽ tự động tạo thêm các Pod để đáp ứng với nhu cầu và giảm số lượng Pod khi tải lưu lượng truy cập giảm.

Horizontal Pod Autoscaling (HPA) là một trong những tính năng quan trọng nhất của Kubernetes. Nó cho phép tự động điều chỉnh số lượng Pod để đáp ứng với tải lượng truy cập đến ứng dụng. HPA hoạt động dựa trên các metrics như CPU utilization hoặc custom metrics, và sử dụng thông tin này để quyết định số lượng Pod cần tạo ra hoặc xóa bỏ. Khi tải lưu lượng truy cập đến ứng dụng tăng,

HPA sẽ tự động tạo thêm các Pod để đáp ứng với nhu cầu và giảm số lượng Pod khi tải lưu lượng truy cập giảm.

Ví dụ: Giả sử bạn có một Deployment trong Kubernetes chứa một ứng dụng web. Bạn muốn sử dụng HPA để tự động tạo thêm các Pod khi CPU utilization của các Pod hiện có vượt quá 80%.

Đầu tiên, bạn cần tạo một file YAML để định nghĩa HPA. Ví dụ:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80
```

Trong file YAML này, một HPA có tên là `my-app-hpa` sẽ được tạo. Nó liên kết với một Deployment có tên là `my-app`. HPA sẽ giữ số lượng Pod trong khoảng từ 1 đến 10 và sẽ tự động tạo thêm các Pod khi CPU utilization vượt quá 80%.

Sau khi triển khai file YAML này, HPA sẽ bắt đầu thu thập thông tin về CPU utilization và tự động điều chỉnh số lượng Pod để đáp ứng với tải lưu lượng truy cập đến ứng dụng.

Vertical Pod Autoscaling (VPA)

VPA được sử dụng để điều chỉnh tài nguyên của các container trong Pod. Nó hoạt động bằng cách thay đổi các giá trị tài nguyên được yêu cầu bởi container để giảm thiểu sự lãng phí và tăng khả năng sử dụng tài nguyên. VPA hỗ trợ các container chạy trên các node Kubernetes và sử dụng các metrics như CPU và memory để quyết định số lượng tài nguyên cần thiết cho mỗi container.

VPA hỗ trợ các container chạy trên các node Kubernetes và sử dụng các metrics như CPU và memory để quyết định số lượng tài nguyên cần thiết cho mỗi container. Nó có thể giúp giảm thiểu sự lãng phí và tăng khả năng sử dụng tài nguyên bằng cách thay đổi các giá trị tài nguyên được yêu cầu bởi container.

Ví dụ:

```
apiVersion: apps/v1
  kind: Deployment
```

```
  metadata:
    name: nginx-deployment
    labels:
      app: nginx
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
        - name: nginx
          image: nginx:1.7.8
          ports:
          - containerPort: 80
```

- VPA resource:

```
apiVersion: autoscaling.k8s.io/v1beta1
  kind: VerticalPodAutoscaler
  metadata:
    name: nginx-deployment-vpa
  spec:
    targetRef:
      apiVersion: "apps/v1"
      kind:       Deployment
      name:       nginx-deployment
    updatePolicy:
      updateMode: "Off"
```

- Recommended resource:

```
recommendation:
  containerRecommendations:
  - containerName: nginx
    lowerBound:
```

```
      cpu: 40m
      memory: 3100k
    target:
      cpu: 60m
      memory: 3500k
    upperBound:
      cpu: 831m
      memory: 8000k
```

Tổng kết

Autoscaling là một tính năng quan trọng trong Kubernetes giúp tăng khả năng sẵn sàng và tối ưu hóa sử dụng tài nguyên. Horizontal Pod Autoscaling (HPA) và Vertical Pod Autoscaling (VPA) là hai tính năng chính trong autoscaling của Kubernetes, cung cấp các cách tiếp cận khác nhau để điều chỉnh số lượng Pod và tài nguyên của chúng.