Kubernetes

# I. Understand the basics concepts

## 1. Tổng quan về Kubernetes

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.

### 1.1 Lợi ích của Kubernetes

Kubernetes cung cấp cho người dung:

* **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 khả năng tự động mount một hệ thống lưu trữ mà người dùng có thể chọn, như local storages, public cloud providers, v.v.
* **Tự động rollouts và rollbacks**  
  Người dùng 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 ta có thể tự động hoá Kubernetes để tạo mới các container cho việc triển khai của mình hoặc xoá các container hiện có và áp dụng tất cả các resource của chúng vào container mới.
* **Đóng gói tự động**  
  Một Kubernetes cluster gồm các node mà nó có thể sử dụng để chạy các tác vụ được đóng gói (containerized task). Khi đó Kubernetes sẽ được 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 tài nguyên hiện có.
* **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 ta lưu trữ và quản lý các thông tin nhạy cảm như: password, OAuth token và SSH key. Ta 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 mình.

### 1.2 Kubernetes Components

Một Kubernetes cluster bao gồm các thành phần đại diện cho control plane và một tập hợp các máy được gọi là các Node, hay còn gọi là các máy worker.

Các máy worker lưu trữ các Pods như là thành phần của application workload. Trong khi đó, nút control plaane quản lý các nút worker và các Pods trong Cluster. Trong môi trường production, control plane thường chạy trên nhiều máy tính và một cluster thường chạy nhiều nút mang lại khả năng chịu lỗi và tính sẵn sàng cao.  
  
  
Đây là sơ đồ của một cụm Kubernetes với tất cả các thành phần được gắn với nhau.

Trong sơ đồ trên, các Kubernetes Components có thể chia làm 3 phần chính bao gồm:

#### 1.2.1 Control Plane Components

kube-apiserver

etcd

kube-scheduler

kube-controller-manager

cloud-controller-manager

#### 1.2.2 Node Components

kubelet

kube-proxy

container runtime

#### 1.2.3 Addons

DNS

Web UI (Dashboard)

Container Resource

Monitoring

Cluster-level Logging

### 1.3 Kubernetes API

Kubernetes API cho phép bạn truy vấn và thao tác với trạng thái của các đối tượng trong Kubernetes. Cốt lõi của control plane trong Kubernetes là máy chủ API và API HTTP mà nó hiển thị. Người dùng, các thành phần khác nhau trong cluster và các thành phần bên ngoài đều giao tiếp với nhau thông qua máy chủ API.

### 1.4. Kubernetes Objects

Các đối tượng Kubernetes là các thực thể liên tục trong hệ thống Kubernetes. Kubernetes sử dụng các thực thể này để đại diện cho trạng thái cụm của bạn.

## 2. Kiến trúc của Cluster

### 2.1 Nodes

Kubernetes chạy workload bằng cách đặt các containers vào Pod để chạy trên Nodes. Một Node có thể là một máy ảo hoặc máy vật lý, tùy thuộc vào cluster. Mỗi node được quản lý bởi control plane và chứa các service cần thiết để chạy Pod

Các thành phần trên một Node bao gồm kubelet, container runtime và kube-proxy.

2.2 Control Plane-Node Communication

2.3 Controllers

2.4 Cloud Controller Manager

## 3. Containers

Mỗi container mà khi đã chạy đều có thể lặp lại nhờ tiêu chuẩn hóa từ việc bao gồm các dependencies, có nghĩa là ta sẽ có được behavior giống nhau ở bất kỳ nơi nào chạy nó.

Container tách các ứng dụng khỏi cơ sở hạ tầng máy chủ lưu trữ bên dưới. Điều này làm cho việc triển khai dễ dàng hơn trong các môi trường đám mây hoặc hệ điều hành khác nhau.

### 3.1 Container images

Container images là một gói phần mềm có thể chạy ở mọi nơi, nó chứa mọi thứ cần thiết để chạy một ứng dụng: code và bất kỳ runtime nào mà ứng dụng cần, các thư viện ứng dụng và hệ thống cũng như các giá trị mặc định cho bất kỳ cài đặt thiết yếu nào.

Theo thiết kế, container là bất biến: ta không thể thay đổi mã của container đang chạy. Nếu ta có một ứng dụng được chứa trong container và muốn thực hiện thay đổi, khi đó ta cần tạo một Image mới bao gồm các thay đổi đó, sau đó tạo lại container để bắt đầu từ container image đã cập nhật.

### 3.2 Container runtimes

Container runtimes là môi trường chịu trách nhiệm chạy các Containers. Kubernetes hỗ trợ một số container runtime như: Docker, containerd, CRI-O và bất kỳ triển khai nào của Kubernetes CRI (Container Runtime Interface).

## 4. Workloads

Workloads là một application chạy trên Kubernetes. Cho dù Workload là một thành phần đơn lẻ hay nhiều thành phần hoạt động cùng nhau thì trên Kubernetes đều sẽ chạy nó bên trong một tập hợp các Pods.

Các Pods trong Kubernetes có một vòng đời xác định. Khi một Pod đang chạy trong Cluster xảy ra lỗi trên Node nơi Pod đó đang chạy dẫn đến tất cả các Pod trên Node đó đều bị lỗi. Khi đó ra sẽ cần tạo một Pod mới để khôi phục lại trạng thái ok ban đầu và vì vậy việc quản lý Pods là cực kỳ quan trọng. Tuy nhiên ta không cần phải quản lý trực tiếp từng Pod. Thay vào đó có thể sử dụng Workload resources để quản lý một tập các Pods. Các Workload resources này định cấu hình controllers để đảm bảo một lượng phù hợp các Pods đang chạy và phù hợp với state được chỉ định từ trước.

## 4.1. Deployment và ReplicaSet

Mục đích của ReplicaSet là duy trì một tập hợp các bản sao của Pod có thể chạy ổn định tại bất kỳ thời điểm nào. Do đó nó thường được sử dụng để đảm bảo tính khả dụng của một số lượng Pods giống hệt nhau.

Một ReplicaSet được định nghĩa bằng các fields, bao gồm một bộ selector chỉ định cách xác định các Pods mà ReplicaSet có thể gom lại được, một số replicas cho biết nó sẽ duy trì bao nhiêu Pod và một pod template chỉ định dữ liệu của các Pod mới mà nó sẽ tạo để đáp ứng số lượng của tiêu chí nhân bản. Sau đó, một ReplicaSet thực hiện mục đích của nó bằng cách tạo và xóa các Pod nếu cần để đạt được số lượng mong muốn. Khi một ReplicaSet cần tạo các Pod mới, nó sẽ sử dụng Pod teplate của nó để tạo.

Một ReplicaSet được liên kết với Pods của nó thông qua field metadata.ownerRefferences của Pods, field này chỉ định tài nguyên mà đối tượng hiện tại sở hữu. Tất cả các Pods được ReplicaSet gom lại đều có thông tin nhận dạng của ReplicaSet lữu trữ trong field chủ sở hữu của chúng. Thông qua liên kết này mà ReplicaSet biết về state của các Pod mà nó đang duy trì và lập kế hoạch cho phù hợp.

Một ReplicaSet xác định các Pods mới bằng cách sử dụng selector của nó. Nếu có một Pod không có OwnerReference hoặc OwnerReference không phải là Contoller và nó khớp với selector của ReplicaSet, nó sẽ ngay lập tức được ReplicaSet đó gom lại

Một ReplicaSet đảm bảo rằng một số lượng bản sao các Pod được chỉ định đang chạy tại bất kỳ thời điểm nào. Tuy nhiên, Deployment là một khái niệm cấp cao hơn so với ReplicaSets và cung cấp các bản cập nhật khai báo cho Pods cùng với rất nhiều tính năng hữu ích khác. Do đó thông thường người ta sử dụng Deployments thay vì trực tiếp sử dụng ReplicaSets, trừ khi họ có nhu cầu tùy chỉnh và điều phối các cập nhật hoặc hoàn toàn không yêu cầu cập nhật.

Deployment cung cấp các bản cập nhật khai báo cho Pods và ReplicaSets. Ta mô tả một desired state trong Deployment và Deployment Controller thay đổi actual state thành desired state mà mình mong muốn với tốc độ kiểm soát được. Ta cũng có thể định nghĩa các Deployment để tạo các ReplicaSets mới hoặc để xóa các Deployment hiện có và sử dụng tất cả các tài nguyên của chúng cho Deployment mới.

Dưới đây là các trường hợp sử dụng điển hình cho Deployment:

* Tạo Deployment để rollout một ReplicaSet: ReplicaSet tạo các Pods trong background, ta có thể kiểm tra trạng thái rollout để xem liệu nó có thành công hay không.
* Khai báo new state của Pods bằng cách cập nhật PodTemplateSpec của Deployment: Một ReplicaSet mới được tạo và Deployment quản lý việc di chuyển các Pod từ ReplicaSet cũ sang ReplicaSet mới với tốc độ kiểm soát được. Mỗi ReplicaSet mới sẽ cập nhật lại bản sửa đổi của Deployment.
* Quay lại phiên bản Deployment trước đó nếu trạng thái hiện tại của Deployment không ổn định:
* Mở rộng quy mô Triển khai để tạo điều kiện tải nhiều hơn
* Tạm dừng Deployment: Để áp dụng nhiều bản sửa lỗi cho PodTemplateSpec của nó và sau đó tiếp tục triển khai để bắt đầu một đợt phát hành mới.
* Sử dụng status của Deployment như một indicator cho quá trình rollout nếu bị lỗi
* Dọn dẹp các ReplicaSets cũ không cần dùng đến nữa

## 4.2. DeamonSet

Một DaemonSet đảm bảo rằng tất cả (hoặc một số) Nodes chạy một bản sao của Pod. Các Pod sẽ được thêm đồng thời với khi các Node mới được thêm vào Cluster. Và khi các Node bị xóa khỏi Cluster, các Pod đó sẽ được theo bằng cách xóa DaemonSet sẽ xóa các Pod mà nó đã tạo.

Một số cách sử dụng điển hình của DaemonSet là:

* Chạy một daemon lưu trữ Cluster trên mọi Node
* Chạy daemon để ghi lại các logs trên mọi Node
* Chạy deamon nền giám sát mọi Node

Trong trường hợp đơn giản, một DaemonSet chứa tất cả các Node sẽ được sử dụng cho từng loại daemon. Một thiết lập phức tạp hơn có thể sử dụng nhiều DaemonSets cho một loại daemon, nhưng với các flags khác nhau hoặc các yêu cầu bộ nhớ và cpu khác nhau cho các loại phần cứng khác nhau.

## 4.3. StatefulSets

StatefulSet là workload API object dùng để quản lý state của các ứng dụng. Cụ thể StatefulSet sẽ quản lý việc triển khai và mở rộng quy mô của một tập hợp các Pods và cung cấp đảm bảo về thứ tự và tính duy nhất của các Nhóm này.

Giống như một Deployment, StatefulSet quản lý các Pod dựa trên một thông số kỹ thuật của vùng chứa giống hệt nhau. Điểm khác biệt so với Deployment là một StatefulSet duy trì một định danh cố định cho mỗi Pod của chúng. Các Pod này được tạo từ cùng một thông số kỹ thuật, nhưng không thể hoán đổi cho nhau: mỗi Pod có một số nhận dạng liên tục mà nó duy trì qua bất kỳ lần lên lịch nào.

Nếu người dùng muốn sử dụng khối lượng Volumes để cung cấp sự bền bỉ, tiên tục cho workloads của mình, họ có thể sử dụng StatefulSet mặc dù các Pod riêng lẻ trong StatefulSet dễ bị lỗi, nhưng các mã nhận dạng Pod liên tục giúp dễ dàng khớp các ổ hiện có với các Pod mới thay thế các ổ bị lỗi. Có thể liệt kê một số trường hợp cần sử dụng StatefulSets như sau:

Số lượng các Pod ổn định và network được đinh danh duy nhất. Lưu trữ ổn định, bền bỉ. Có thứ tự, triển khai nhỏ gọn và cps thể mở rộng. Cập nhật luân phiên theo thứ tự và cập nhật tự động.

## 4.4. Job và CronJob

Một Job tạo ra một hoặc nhiều Pod và sẽ tiếp tục thử thực hiện lại các Pod cho đến khi một số lượng cụ thể trong số đó kết thúc công việc, chức năng hay mục tiêu hoạt động của nó thành công. Khi các Pod hoàn thành chức năng của nó, Job sẽ theo dõi các Pods hoàn thành thành công này và ghi lại cho đến khi đạt đến một số lần hoàn thành thành công được chỉ định từ trước, nhiệm vụ (tức là Job) đã hoàn thành thì xóa Job sẽ xóa các Pod mà nó đã tạo. Tạm dừng một Job sẽ xóa các Pods đang hoạt động của nó cho đến khi Job được tiếp tục trở lại.

Một trường hợp đơn giản là tạo một đối tượng Job để chạy một Pod đến khi hoàn thành một chức năng của nó. Đối tượng Job sẽ bắt đầu một Pod mới nếu Pod đầu tiên bị lỗi hoặc bị xóa (ví dụ: do lỗi phần cứng của nút hoặc do nút khởi động lại). Ta cũng có thể sử dụng một Job để chạy nhiều Pods song song với nhau.

Một CronJob tạo các Jobs theo một lịch trình lặp đi lặp lại.

Một đối tượng CronJob giống như một dòng của tệp crontab. Nó chạy một Job định kỳ theo một lịch trình nhất định, được viết ở định dạng Cron.

CronJobs rất hữu ích để tạo các tác vụ định kỳ và lặp đi lặp lại như chạy sao lưu hoặc gửi email. CronJobs cũng có thể lên lịch cho các nhiệm vụ riêng lẻ trong một thời gian cụ thể, chẳng hạn như lên lịch cho một Job khi Cluster đang chạy có khả năng xảy ra lỗi, ngưng hoạt động.

## 4.5 Tổng kết

Deployment và ReplicaSet: Phù hợp để quản lý khối lượng công việc ứng dụng không cần trạng thái trên Cluster đang chạy, nơi bất kỳ Pod nào trong Deployment đều có thể hoán đổi cho nhau và có thể được thay thế nếu cần.

* StatefulSet: Cho phép chạy một hoặc nhiều Pod liên quan có trạng thái theo dõi. Ví dụ: nếu khối lượng công việc cần phải ghi lại dữ liệu liên tục, ta có thể chạy StatefulSet khớp với từng Pod với PersentlyVolume. Ứng dụng khi đó đang chạy trong các Pod với StatefulSet đó có thể sao chép dữ liệu sang các Pod khác trong cùng một StatefulSet để cải thiện khả năng phục hồi tổng thể cho ứng dụng.
* DaemonSet: Định nghĩa các Pod cung cấp các tiện ích node-local. Đây có thể là cơ sở cho hoạt động của Cluster chẳng hạn như công cụ trợ giúp mạng hoặc là một phần của tiện ích bổ sung. Mỗi khi thêm một node vào Cluster của mình phù hợp với đặc điểm kỹ thuật trong DaemonSet, node Master sẽ lên lịch một Pod cho DaemonSet đó vào Node mới này.
* Job và CronJob: Xác định các nhiệm vụ chạy đến khi hoàn thành và sau đó dừng lại. Job đại diện cho các nhiệm vụ một lần, trong khi CronJobs lặp lại theo lịch trình.

Trong Kubernetes còn rất nhiều kiểu workload resources với mỗi chức năng và cách sử dụng khác nhau, có thể xem chi tiết tại kubernetes/workloads

## 5. Service, Load Balancing và Networking

# II. Practice

## 1. Cài đặt Kubernetes locally với Minikube