Kubernetes

Kubernetes là dự án mà nguồn mở để quản lý các container: automating deployment, scaling, and management các ứng dụng trên container. (Tạo, sửa, xoá, xếp lịch(schedule), mở rộng (scale)...) trên nhiều máy).

Kubernetes viết tắt là k8s.

Kubernetes hỗ trợ các công nghệ container là docker và rkt.

Các tiện ích mà k8s mang lại cho chúng ta:

- Triển khai ứng dụng một cách nhanh chóng.
- Scale ứng dụng dễ dàng.
- Liên tục đưa ra các tính năng mới.
- Tối ưu hóa việc sử dụng tài nguyên.

1. Các khái niệm trong K8S.

1.1 Pod

- Pod là 1 nhóm (1 trở lên) các container chứa ứng dụng cùng chia sẽ các tài nguyên lưu trữ, địa chỉ ip...
- Pod có thể chạy theo 2 cách sau:
    - **Pods that run a single container.**: 1 container tương ứng với 1 pod.
    - **Pods that run multiple containers that need to work together.**: Một Pod có thể là một ứng dụng bao gồm nhiều container được kết nối chặt chẽ và cần phải chia sẻ tài nguyên với nhau giữa các container.
- Pods cung cấp hai loại tài nguyên chia sẻ cho các containers: networking và storage.
- **Networking**: Mỗi pod sẽ được cấp 1 địa chỉ ip. Các container trong cùng 1 Pod cùng chia sẽ network namespace (địa chỉ ip và port). Các container trong cùng pod có thể giao tiếp với nhau và có thể giao tiếp với các container ở pod khác (use the shared network resources).
- **Storage**: Pod có thể chỉ định một shared storage volumes. Các container trong pod có thể truy cập vào volume này.

1.2 Replication Controllers

- Replication controller đảm bảo rằng số lượng các pod replicas đã định nghĩa luôn luôn chạy đủ số lượng tại bất kì thời điểm nào.

- Thông qua Replication controller, Kubernetes sẽ quản lý vòng đời của các pod, bao gồm scaling up and down, rolling deployments, and monitoring.

1.3 Services

- Vì pod có tuổi thọ ngắn nên không đảm bảo về địa chỉ IP mà chúng được cung cấp.
- Service là khái niệm được thực hiện bởi : domain name, và port. Service sẽ tự động "tìm" các pod được đánh label phù hợp (trùng với label của service), rồi chuyển các connection tới đó.
- Nếu tìm được 5 pods thoả mã label, service sẽ thực hiện load-balancing: chia connection tới từng pod theo chiến lược được chọn (VD: round-robin: lần lượt vòng tròn).
- Mỗi service sẽ được gán 1 domain do người dùng lựa chọn, khi ứng dụng cần kết nối đến service, ta chỉ cần dùng domain là xong. Domain được quản lý bởi hệ thống name server SkyDNS nội bộ của k8s - một thành phần sẽ được cài khi ta cài k8s.
- Đây là nơi bạn có thể định cấu hình cân bằng tải cho nhiều pod và expose các pod đó.

1.4 Volumes

- Volumes thể hiện vị trí nơi mà các container có thể truy cập và lưu trữ thông tin.
- Volumes có thể là local filesystem,local storage, Ceph, Gluster, Elastic Block Storage,..
- Persistent volume (PV) là khái niệm để đưa ra một dung lượng lưu trữ THỰC TẾ 1GB, 10GB ...
- Persistent volume claim (PVC) là khái niệm ảo, đưa ra một dung lượng CẦN THIẾT, mà ứng dụng yêu cầu.

Khi 1 PV thoả mãn yêu cầu của 1 PVC thì chúng "match" nhau, rồi "bound" (buộc / kết nối) lại với nhạu.

1.5 Namespaces

- Namespace hoạt động như một cơ chế nhóm bên trong Kubernetes.
- Các Services, pods, replication controllers, và volumes có thể dễ dàng cộng tác trong cùng một namespace.
- Namespace cung cấp một mức độ cô lập với các phần khác của cluster.

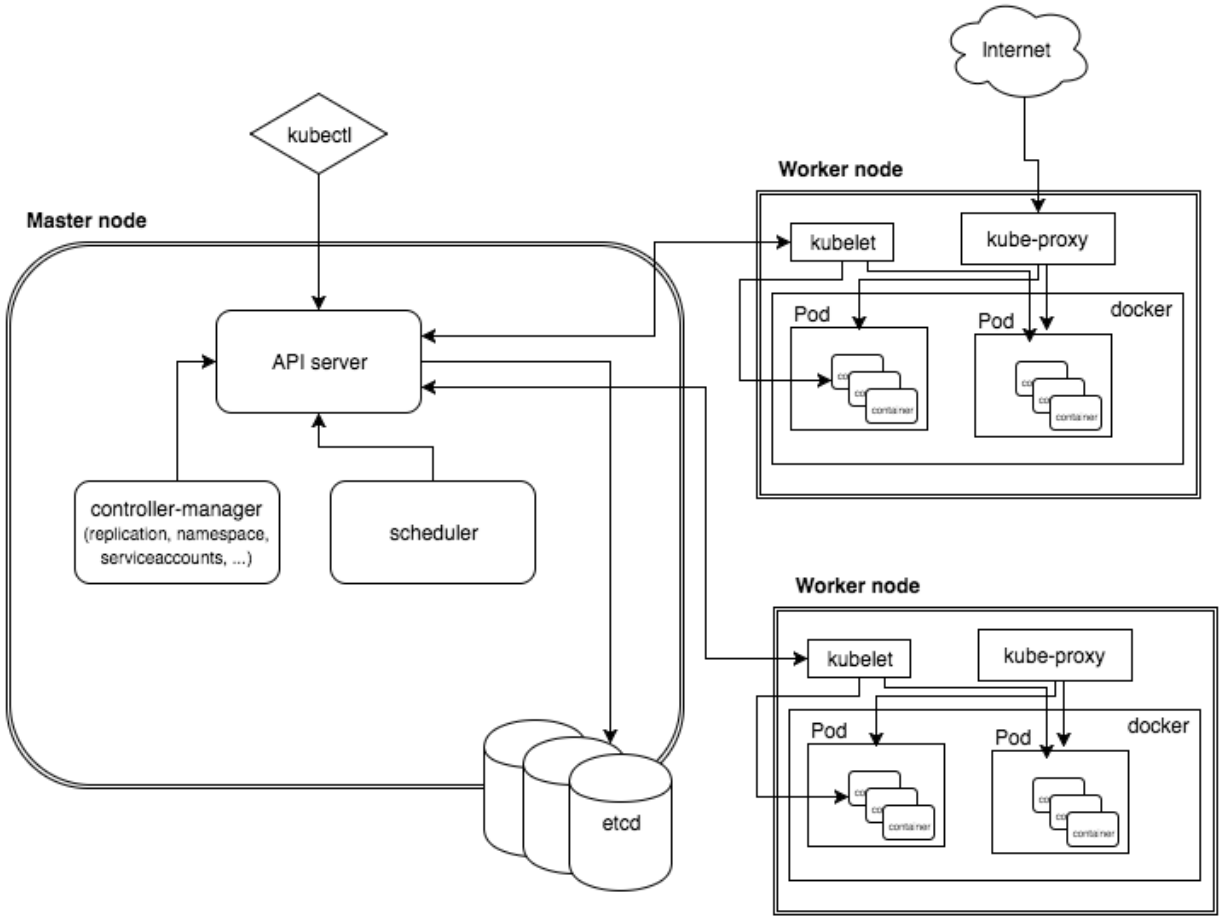
1.6 ConfigMap (cm) - Secret

- ConfigMap là giải pháp để nhét 1 file config / đặt các ENVironment var hay set các argument khi gọi câu lệnh. ConfigMap là một cục config, mà pod nào cần, thì chỉ định là nó cần - giúp dễ dàng chia sẻ file cấu hình.
- secret dùng để lưu trữ các mật khẩu, token, ... hay những gì cần giữ bí mật. Nó nằm bên trong container.

1.7 Labels - Annotations

- Labels: Là các cặp **key-value** được Kubernetes đính kèm vào pods, replication controllers,...
- Annotations: You can use Kubernetes annotations to attach arbitrary non-identifying metadata to objects. Clients such as tools and libraries can retrieve this metadata.
- Labels can be used to select objects and to find collections of objects that satisfy certain conditions. In contrast, annotations are not used to identify and select objects.

2. Thành phần



2.1 Master node

- Chịu trách nhiệm quản lý Kubernetes cluster.
- Đây là nơi mà sẽ cấu hình các nhiệm vụ sẽ thực hiện.
- Quản lý, điều hành các work node.

2.1.1 API server

- API server là nơi tiếp nhận các lệnh REST được sử dụng để kiểm soát cluster.
- Nó xử lý các yêu cầu, xác nhận chúng, thực hiện các ràng buộc.
- Trạng thái kết quả phải được duy trì ở một nơi nào đó, và điều đó đưa chúng ta đến thành phần tiếp theo của nút chính.

2.1.2 etcd storage

- etcd: một cơ sở dữ liệu key-value có tính khả dụng cao, phân phối và nhất quán sử dụng để tìm kiếm dịch vụ.
- Nó chủ yếu được sử dụng để chia sẻ các cấu hình và khám phá dịch vụ (service discovery).
- Ví dụ về dữ liệu được lưu trữ bởi Kubernetes trong etcd là các công việc được lên kế hoạch (jobs being scheduled), tạo và triển khai pod, services, namespaces, replication information,..

2.1.3 Scheduler

- Đảm nhiệm chức năng là triển khai các pods, services lên các nodes.
- Scheduler nắm các thông tin liên quan đến các tài nguyên có sẵn trên các thành viên của cluster, cũng như các yêu cầu cần thiết cho dịch vụ cấu hình để chạy và do đó có thể quyết định nơi triển khai một dịch vụ cụ thể.

2.1.4 controller-manager

- Sử dụng api server để có thể xem trạng thái của cluster và từ đó thực hiện các thay đổi chính xác cho trạng thái hiện tại để trở thành một trạng thái mong muốn.
- Ví dụ Replication controller có chức năng đảm bảo rằng số lượng các pod replicas đã định nghĩa luôn luôn chạy đủ số lượng tại bất kì thời điểm nào.

2.2 Worker node

- Là nơi mà các pod sẽ chạy.
- Chứa tất cả các dịch vụ cần thiết để quản lý kết nối mạng giữa các container, giao tiếp với master node, và gán các tài nguyên cho các container theo kế hoạch.

2.2.1 Docker

- Là môi trường để chạy các container.

2.2.2 kubelet

- kubelet lấy cấu hình thông tin pod từ api server và đảm bảo các containers up và running.
- kubelet chịu trách nhiệm liên lạc với master node.
- Nó cũng liên lạc với etcd, để có được thông tin về dịch vụ và viết chi tiết về những cái mới được tạo ra.

2.2.3 kube-proxy

- Kube-proxy hoạt động như một proxy mạng và cân bằng tải cho một dịch vụ trên một work node.
- Nó liên quan đến việc định tuyến mạng cho các gói TCP và UDP.

2.2.4 kubectl

- Giao diện dòng lệnh để giao tiếp với API service.
- Gửi lệnh đến master node.

3. Viết file cấu hình tạo Deployments, pod, services.

3.1 Ví dụ file Pods.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: rss-site
  labels:
    app: web
spec:
  containers:
  – name: rss-reader
    image: nginx:1.10
    ports:
      – containerPort: 80
    volumeMounts:
      - name: rss-volume
        mountPath: /var/www/html
  volumes:
  - name: rss-volume
    hostPath:
      path: /data
```

- apiVersion: v1: Chỉ ra phiên bản api đang sử dụng. Hiện tại k8s có pod ở phiên bản 1

- kind: Pod: : Chỉ loại file mà ta muốn tạo. Có thể là pod, Deployment, Job, Service,.... Tùy theo từng loại mà có các phiên bản api khác nhau. Xem thêm tại đây: https://kubernetes.io/docs/api-reference/v1.6/
- metadata:: Là các siêu dữ liệu khi tạo pod.
    - name: rss-site: Khai báo tên của pod.
    - app: web: Ta bổ sung thêm nhãn label với key là app và value tương ứng là web.
- spec:: Khai báo các đối tượng mà k8s sẽ xử lý như: containers, volume,...
    - containers: Khai báo đối tượng containers
        - name: rss-reader: Tên containers.
        - image: nginx:1.10: containers sẽ được build từ image nginx với tag là 1.10
        - ports: Liệt kê các port sẽ được containers expose ra. Ở đây là port 80.
        - volumeMounts: Chỉ định volume sẽ được mount vào containers.
            - name: rss-volume: Tên volume.
            - mountPath: /var/www/html: Volume sẽ được mount vào thư mục /var/www/html trên containers
    - volumes:: Khai báo đối tượng volume.
        - name: rss-volume: Tên volume
        - hostPath:: Volume sẽ sử dụng thư mục /data trên máy host.

Tài liệu được viết rất rõ ràng và chi tiết tại địa chỉ https://kubernetes.io/docs/api-reference/v1.6/

3.2 File Deployments

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

- apiVersion: apps/v1beta1: Chỉ ra phiên bản api sử dụng.
- kind: Deployment: Loại file, ở đây là Deployment.
- metadata:: Các thông tin bổ sung cho deployment này.
    - name: nginx-deployment: Tên deployment.
- spec: Khai báo các đối tượng mà k8s sẽ xử lý như: containers, volume,...

    - replicas: 3: 3Pods sẽ được tạo ra.
    - template: Template định nghĩ pod sẽ được tạo từ template này. Các thông tin trong mục template tương ứng với phần định nghĩa ở File pod ở trên. Tạo ra containers từ image nào, xuất ra port nào, có các labels nào,...

3.3 File Service

```
apiVersion: v1
kind: Service
metadata:
  name: dbfrontend
  labels:
    name: dbfrontend
  spec:
    # label keys and values that must match in order to receive traffic for this service
    selector:
      name: dbfrontend
    ports:
     # the port that this service should serve on
      - port: 5555
        targetPort: 3306
    type: NodePort
```

Ta chỉ tập trung vào các thông số:

- selector:
    - name: dbfrontend: Định tuyến traffic đến pod với cặp giá trị này (name->dbfrontend).
- port:
    - port: 5555: port sẽ được expose bởi service.
    - targetPort: 3306: Port mà truy cập vào pods. (Giá trị từ 1-65535). (port của service trên container).
- type: NodePort: type determines how the Service is exposed. Defaults to ClusterIP. NodePort sẽ expose port 5555 trên tất cả các node (cả cụm cluster).

Giải thích thêm các thông số trong type:

Trong type có 3 dạng đó là:

- ClusterIP: "ClusterIP" allocates a cluster-internal IP address for load-balancing to endpoints. Endpoints are determined by the selector or if that is not specified, by manual construction of an Endpoints object. If clusterIP is "None", no virtual IP is allocated and the endpoints are published as a set of endpoints rather than a stable IP.

- NodePort: on top of having a cluster-internal IP, expose the service on a port on each node of the cluster (the same port on each node). You'll be able to contact the service on any NodeIP:NodePort address.

- LoadBalancer: on top of having a cluster-internal IP and exposing service on a NodePort also, ask the cloud provider for a load balancer which forwards to the Service exposed as a NodeIP:NodePort for each Node.

Câu lệnh về kubectl

View cluster info

```
root@master:~# kubectl cluster-info
Kubernetes master is running at https://172.16.69.237:6443
Heapster is running at https://172.16.69.237:6443/api/v1/proxy/namespaces/kube-system/services/heapster
KubeDNS is running at https://172.16.69.237:6443/api/v1/proxy/namespaces/kube-system/services/kube-dns

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
root@master:~#
```

List the nodes in cluster:

```
root@master:~# kubectl get nodes
NAME      STATUS    AGE       VERSION
master    Ready     7d        v1.6.1
minion1   Ready     7d        v1.6.1
minion2   Ready     7d        v1.6.1
root@master:~#
```

List all Containers in all namespaces

```
kubectl get pods --all-namespaces
```

List Containers filtering by Pod namespace

```
kubectl get pods --namespace kube-system
```

Trong đó, kube-system là tên namespace.

List cluster events:

```
root@master:/opt/heapster/deploy# kubectl get events
LASTSEEN   FIRSTSEEN   COUNT     NAME                          KIND                  SUBOBJECT
TYPE      REASON                  SOURCE                  MESSAGE
3m         3m          1         php-apache-3580908300-x6tmd   Pod                                 Normal
Scheduled               default-scheduler       Successfully assigned php-apache-3580908300-x6tmd to minion2
3m         3m          1         php-apache-3580908300-x6tmd   Pod                 spec.containers{php-apache}
Normal    Pulling                 kubelet, minion2        pulling image "gcr.io/google_containers/hpa-example"
2m         2m          1         php-apache-3580908300-x6tmd   Pod                 spec.containers{php-apache}
Normal    Pulled                  kubelet, minion2        Successfully pulled image "gcr.io/google_containers/hpa-
example"
2m         2m          1         php-apache-3580908300-x6tmd   Pod                 spec.containers{php-apache}
Normal    Created                 kubelet, minion2        Created container with id
b5ff9a5fe3e47666b78835dd818ea70d989f48121db9806628dc9cb0e0bb0d6c
```

```
2m        2m        1        php-apache-3580908300-x6tmd      Pod                        spec.containers{php-apache}
Normal    Started                    kubelet, minion2           Started container with id
b5ff9a5fe3e47666b78835dd818ea70d989f48121db9806628dc9cb0e0bb0d6c
3m        3m        1        php-apache-3580908300            ReplicaSet                                 Normal
SuccessfulCreate             replicaset-controller      Created pod: php-apache-3580908300-x6tmd
```

List deployments, pods, svc

```
kubectl get deployments
kubectl get pods
kubectl get svc
```

Delete nodes

```
root@master:~# kubectl delete nodes minion1
node "minion1" deleted
root@master:~#
```

Delete deployments, pods, svc

```
root@master:~# kubectl delete svc monitoring-grafana --namespace=kube-system
service "monitoring-grafana" deleted
root@master:~#
```

Get describe deployments pods, svc,...

```
root@master:~# kubectl describe deployments php-apache
Name:                   php-apache
Namespace:              default
CreationTimestamp:      Wed, 26 Apr 2017 00:01:27 +0700
Labels:                 run=php-apache
Annotations:            deployment.kubernetes.io/revision=1
Selector:               run=php-apache
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:       run=php-apache
  Containers:
   php-apache:
    Image:      gcr.io/google_containers/hpa-example
    Port:       80/TCP
    Requests:
      cpu:              200m
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
OldReplicaSets: <none>
```

```
NewReplicaSet:  php-apache-3580908300 (1/1 replicas created)
Events:
  FirstSeen    LastSeen    Count   From                    SubObjectPath   Type          Reason                Message
  ---------    --------    -----   ----                    -------------   --------      ------                -------
  5m           5m          1       deployment-controller                   Normal        ScalingReplicaSet Scaled up replica
set php-apache-3580908300 to 1
root@master:~#
```

Get a shell to the running Container:

```
kubectl exec -it shell-demo /bin/bash
```

Trong đó shell-demo là tên container.

Scale

```
# Scale a replicaset named 'foo' to 3.
root@master:~# kubectl scale --replicas=3 rs/foo

# If the deployment named mysql's current size is 2, scale mysql to 3.
root@master:~# kubectl scale --current-replicas=2 --replicas=3 deployment/mysql
```

auto scale

```
# Auto scale a replication controller "foo", with the number of pods between 1
and 5, target CPU utilization at 80%:
root@master:~# kubectl autoscale rc foo --max=5 --cpu-percent=80
```

Port Forwarding: Forward one or more local ports to a pod.

```
# Listen on port 8888 locally, forwarding to 5000 in the pod
root@master:~# kubectl port-forward mypod 8888:5000
```

Expose: Expose a resource as a new Kubernetes service.

```
# Create a service for an nginx deployment, which serves on port 80 and connects to the containe
rs on port 8000.
root@master:~# kubectl expose deployment nginx --port=80 --target-port=8000
```

Rollout

kubectl rollout supports both Deployment and DaemonSet. It has the following subcommands:

- kubectl rollout undo works like rollback; it allows the users to rollback to a previous version of deployment.

```
# Rollback to the previous deployment
root@master:~# kubectl rollout undo deployment/abc
```

- kubectl rollout pause allows the users to pause a deployment. See pause deployments.
- kubectl rollout resume allows the users to resume a paused deployment.

- kubectl rollout status shows the status of a deployment.
- kubectl rollout history shows meaningful version information of all previous deployments. See development version.
- kubectl rollout retry retries a failed deployment. See perm-failed deployments.

Rolling-update

- Note that kubectl rolling-update only supports Replication Controllers.
- A rolling update works by:
    - a. Creating a new replication controller with the updated configuration.
    - b. Increasing/decreasing the replica count on the new and old controllers until the correct number of replicas is reached.
    - c. Deleting the original replication controller.

```
// Update pods of frontend-v1 using new replication controller data in frontend-v2.json.
$ kubectl rolling-update frontend-v1 -f frontend-v2.json

// Update the pods of frontend-v1 to frontend-v2
$ kubectl rolling-update frontend-v1 frontend-v2 --image=image:v2

// Update the pods of frontend, keeping the replication controller name
$ kubectl rolling-update frontend --image=image:v2
```