Kubernetes

1. Introduction à Kubernetes

Kubernetes (k8s) est un orchestrateur de conteneurs open-source conçu pour automatiser :

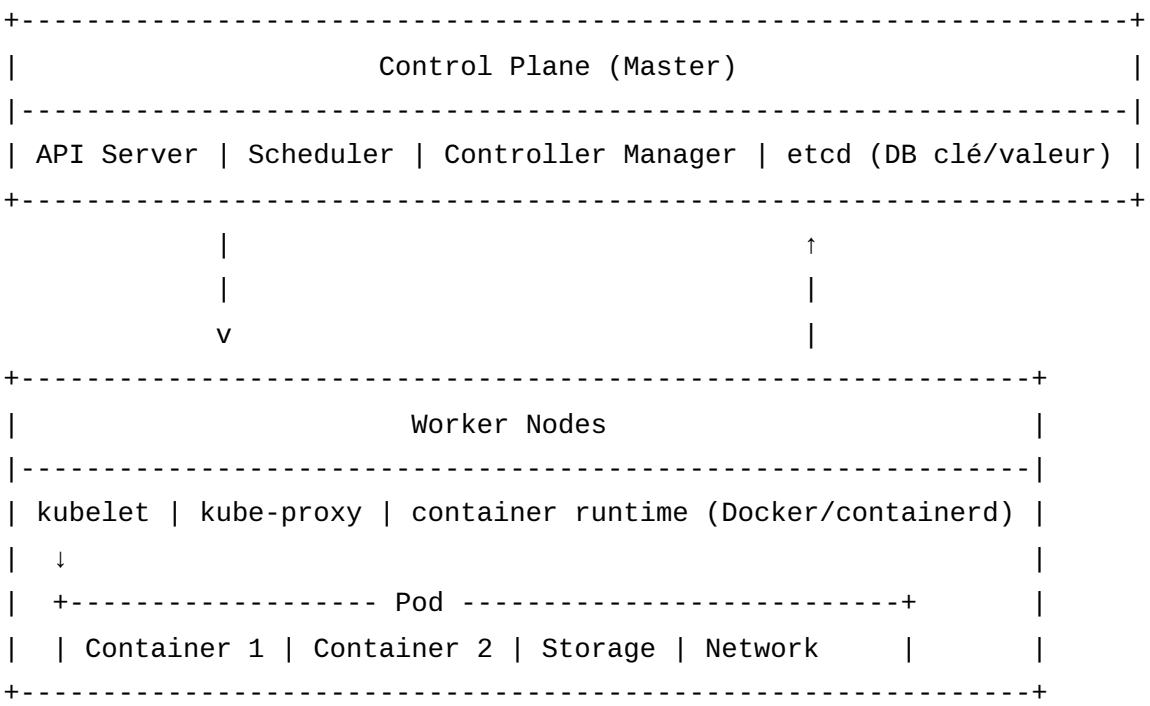
- le déploiement d'applications conteneurisées,
- la montée en charge automatique,
- la tolérance aux pannes,
- la mise à jour continue.

Architecture générale

Un **cluster** Kubernetes est l'ensemble complet qui fait tourner tes conteneurs. Il est composé de plusieurs machines (physiques ou virtuelles) appelées **nodes** reliées entre elles :

- un (ou plusieurs) **Control Plane / Master Node**: C'est le "cerveau" : il planifie, surveille et orchestre tout.
- des **Worker Nodes** : Ce sont les machines qui exécutent réellement les conteneurs.

> Avec k3s, le même nœud peut jouer les deux rôles.



Rôle des composants principaux

- **API Server** : point d'entrée du cluster, reçoit les commandes `kubectl` .
- **etcd** : base de données clé/valeur qui stocke tout l'état du cluster.
- **Scheduler** : assigne les pods aux nœuds en fonction des ressources disponibles.
- **Controller Manager** : surveille et corrige l'état des ressources (ReplicaSets, Deployments, etc.).
- **kubelet** : exécute les conteneurs sur chaque nœud.

- **kube-proxy** : gère la communication réseau entre services et pods.

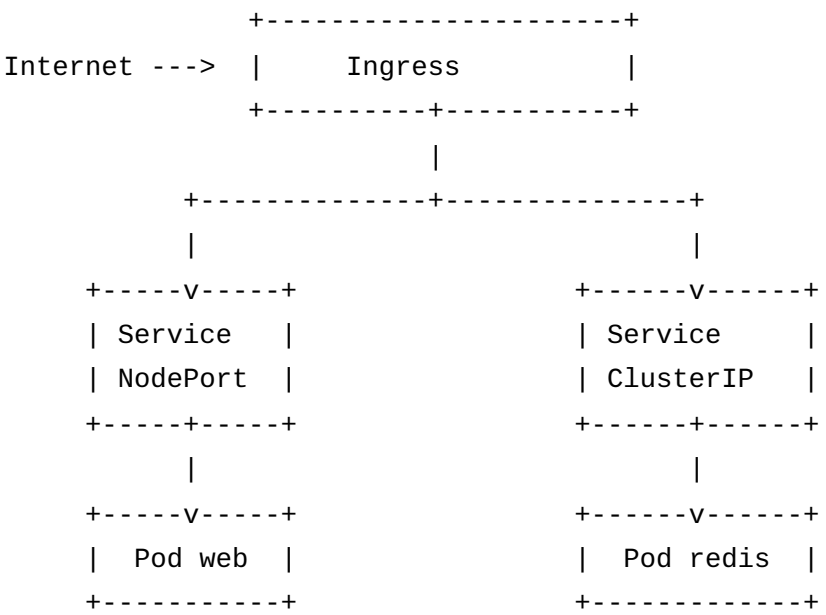
Commandes de base

```
kubectl version
kubectl cluster-info
kubectl get nodes -o wide
kubectl get ns
```

Fonctionnement réseau

- Chaque **Pod** a sa propre IP, **temporaire** et **interne** au cluster. Elle change si le pod est recréé.
- On utilise des **Services** pour exposer les pods et permettre la communication entre eux.
  - **ClusterIP** : service interne au cluster, communication entre pods.
  - **NodePort** : expose le service sur un port spécifique de chaque nœud. Accès via `http://<IP_Nœud>:<NodePort>` .
  - **LoadBalancer** : attribue une IP externe (nécessite un composant type MetalLB ou Traefik).
- **Ingress** : point d'entrée HTTP/HTTPS unique pour plusieurs services, avec routage basé sur les noms d'hôtes et les chemins. Il joue un rôle de **reverse proxy**.

> On peut imaginer les pods comme des **maisons**, les services comme des **rues** et l'ingress comme une **grande avenue** qui redirige vers les bonnes rues.



2. Fonctionnement des principales ressources

Namespace

Permet d’isoler des ressources dans un cluster.

- Utile pour séparer les environnements (dev, prod) ou les équipes.
- Chaque ressource (Pod, Service, etc.) appartient à un namespace.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: ns-vgauti01        # Nom du namespace (personnalisé)
  labels:
    name: ns-vgauti01      # Label pour faciliter la sélection
```

```
kubectl create namespace ns-demo
kubectl get namespaces
kubectl delete namespace ns-demo
```

Pod

Le Pod est l’unité de base de Kubernetes.

- Il encapsule **un ou plusieurs conteneurs** qui partagent : réseau, IP et stockage.
- Si un pod meurt, il est recréé par le ReplicaSet ou Deployment qui le contrôle.

```yaml
# nginx-pod.yaml
apiVersion: v1            # Version de l'API utilisée pour cette ressource
kind: Pod                 # Type de ressource
metadata:
  name: nginx-pod         # Nom unique du pod
  namespace: ns-demo      # Namespace d'exécution
  labels:
    app: nginx            # Label utilisé pour la sélection par d'autres objets
spec:
  containers:
  - name: nginx-container  # Nom du conteneur
    image: nginx:latest    # Image Docker utilisée
    ports:
    - containerPort: 80    # Port ouvert dans le conteneur
```

```
kubectl get pods -n ns-demo
kubectl describe pod nginx-pod -n ns-demo
kubectl logs nginx-pod -n ns-demo
kubectl exec -it nginx-pod -n ns-demo -- /bin/bash
kubectl delete pod nginx-pod -n ns-demo
```

ReplicaSet

Maintient un nombre défini de Pods identiques en fonctionnement.

- Surveille les Pods et en crée/supprime pour respecter `replicas` .

```
# rs-nginx.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-nginx
  namespace: ns-demo
spec:
  replicas: 3              # Nombre de pods souhaités
  selector:
    matchLabels:
      app: nginx           # Doit correspondre aux labels du template
  template:                # Modèle de pod à répliquer
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
```

```
kubectl get rs -n ns-demo
kubectl describe rs rs-nginx -n ns-demo
kubectl scale rs rs-nginx --replicas=4 -n ns-demo
kubectl delete rs rs-nginx -n ns-demo
```

Deployment

Surcouche du ReplicaSet permettant :

- les **mises à jour progressives (rolling updates)**,
- les **retours en arrière (rollback)**,
- le **scaling automatique**.
- Les différentes stratégies de déploiement:
  - RollingUpdate (par défaut) : mise à jour progressive des pods.
  - Recreate : suppression de tous les pods avant de créer les nouveaux.
  - Blue/Green : déploiement parallèle de deux versions.

```yaml
# dep-nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dep-nginx
  namespace: ns-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:                  # Gabarit du pod
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2  # Image initiale
        ports:
        - containerPort: 80
```

```
kubectl get deployments -n ns-demo
kubectl rollout status deployment/dep-nginx -n ns-demo
kubectl set image deployment/dep-nginx nginx=nginx:1.18 -n ns-demo
kubectl rollout undo deployment/dep-nginx -n ns-demo
kubectl delete deployment dep-nginx -n ns-demo
```

Mise à jour :

```
kubectl set image deployment/dep-nginx nginx=nginx:1.18 -n ns-demo
kubectl rollout status deployment/dep-nginx -n ns-demo
kubectl rollout undo deployment/dep-nginx -n ns-demo
```

Service

Expose un ensemble de Pods sur le réseau du cluster.

a) ClusterIP — Communication interne

```yaml
# svc-clusterip.yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-clusterip
  namespace: ns-demo
spec:
  type: ClusterIP            # Service interne uniquement
  selector:
    app: nginx
  ports:
  - port: 80                 # Port exposé
    targetPort: 80           # Port des conteneurs
```

b) NodePort — Expose sur un port des nœuds

```yaml
# svc-nodeport.yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-nodeport
  namespace: ns-demo
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31101          # Port externe (30000–32767)
```

Accès : `http://<IP_Nœud>:31101`

c) LoadBalancer — Avec IP publique

```yaml
# svc-lb.yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-lb
  namespace: ns-demo
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
```

> Nécessite un composant type **MetalLB** ou **Traefik** pour attribuer une IP externe.

```
kubectl get svc -n ns-demo
kubectl describe svc svc-nodeport -n ns-demo
kubectl port-forward svc/svc-nodeport 8080:80 -n ns-demo
kubectl delete svc svc-nodeport -n ns-demo
```

Ingress

Point d’entrée HTTP unique vers plusieurs services.

Il permet de définir des **règles de routage** basées sur les noms d’hôtes et les chemins.

```yaml
# ingress-demo.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-demo
  namespace: ns-demo
spec:
  rules:
  - host: demo.local           # Nom de domaine
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: svc-nodeport  # Service cible
            port:
              number: 80
```

3. k3s sur Raspberry Pi (Cluster léger)

Installation

```bash
curl -sfL https://get.k3s.io | sh -s - --write-kubeconfig-mode 644
```

Vérification

```bash
kubectl get nodes
kubectl get pods -A
kubectl get svc -A
```

Services intégrés

Des services internes qui tournent dans le namespace `kube-system` :

- **coredns** : résolution DNS interne.
- **traefik** : contrôleur Ingress par défaut.
- **metrics-server** : collecte des statistiques.
- **local-path-provisioner** : provisionnement de volumes.