

Kubernetes (κυβερνήτης en grec - **timonier** en français) est une technologie de gestion de conteneurs développée dans Google lab pour gérer les applications conteneurisées dans différents types d'environnement tels que les infrastructures physiques, virtuelles et cloud. C'est un système open source qui aide à créer et à gérer la conteneurisation des applications. C'est un orchestrateur. Ses principaux concurrents sont : **Docker Swarm**, **Apache Mesos Marathon**, **Mesos DC/OS**, **OpenStack Magnum**, **Cattle (RancherOS)**, **Fleet (CoreOS)**, **Titus (Netflix)**, **Nomad (Hashicorp)**, **Deis**, etc. **L'orchestrateur Kubernetes** s'occupe des couches d'infrastructure sous-jacentes (OS, serveurs, stockage et réseaux).

Il permet de :

- Créer des services applicatifs (Frontend et Backend) sur plusieurs conteneurs.
- Planifier leur exécution dans un cluster.
- Garantir leur intégrité.
- Assurer le monitoring.

Le développeur s'occupera uniquement des applications et du contexte d'exécution (conteneurs). Mais un orchestrateur a besoin d'un environnement d'exécution (runtime) en support. Différents environnements d'exécution existent et sont compatibles avec Kubernetes comme ceux qui sont présentés ici :

CRI-O est une implémentation de l'interface **CRI (Container Runtime Interface)** de Kubernetes qui permet d'utiliser des environnements compatibles avec **OCI (Open Container Initiative)**. C'est une alternative légère à l'utilisation de Docker comme moteur d'exécution pour les kubernetes. Il permet à Kubernetes d'utiliser tout environnement d'exécution compatible OCI en tant qu'exécution de conteneur pour l'exécution des pods. Aujourd'hui, il prend en charge **runc** et **Clear Containers** en tant que runtime de conteneurs, mais tout runtime conforme à OCI peut être utilisé en principe.

Rkt (rocket) est un moteur de conteneur d'application développé pour les environnements de production natifs dans le cloud. Il présente une approche native du pod, un environnement d'exécution enfichable et une surface bien définie, ce qui le rend idéal pour l'intégration avec d'autres systèmes.

Docker est un logiciel libre qui automatise le déploiement d'applications dans des conteneurs logiciels. Il s'agit d'une plateforme logicielle open source permettant de créer, de déployer et de gérer des containers d'applications virtualisées sur un système d'exploitation. Les services ou fonctions de l'application et ses différentes bibliothèques, fichiers de configuration, dépendances et autres composants sont regroupés au sein du container. Chaque container exécuté partage les services du système d'exploitation.

Kata Containers est un projet open source et une communauté travaillant à la mise en œuvre d'une implémentation standard de machines virtuelles (VM) légères qui se sentent et fonctionnent comme des conteneurs, tout en offrant l'isolation de la charge de travail et les avantages de sécurité des VM. Le projet Kata Containers comprend six composants : **Agent**, **Runtime**, **Proxy**, **Shim**, **Kernel** et **packaging de QEMU 2.11**. Il est conçu pour être indépendant de l'architecture, et pour s'exécuter

sur plusieurs hyperviseurs en étant compatible avec la spécification OCI pour les conteneurs Docker et CRI pour Kubernetes. Kata Containers associe la technologie d'Intel® Clear Containers et Hyper runV. Le code est hébergé sur **Github** sous la licence Apache 2 et le projet est géré par **OpenStack Foundation**.

CNI (Container Network Interface) est conçu pour être une spécification minimale concernant uniquement la connectivité réseau des conteneurs et la suppression des ressources allouées lorsque le conteneur est supprimé.

CSI (Container Storage Interface - Interface de stockage de conteneurs) est une interface de stockage universelle (en fait une API) entre les orchestrateurs de conteneurs et les fournisseurs de stockage, ce qui facilite l'interopérabilité entre les deux. Les orchestrateurs de conteneurs qui adoptent CSI peuvent en gros tirer parti de tout fournisseur de stockage, cloud ou autre. De même, CSI permet aux fournisseurs de stockage de fournir des services de stockage à tout orchestrateur de conteneurs qui implémente la spécification.

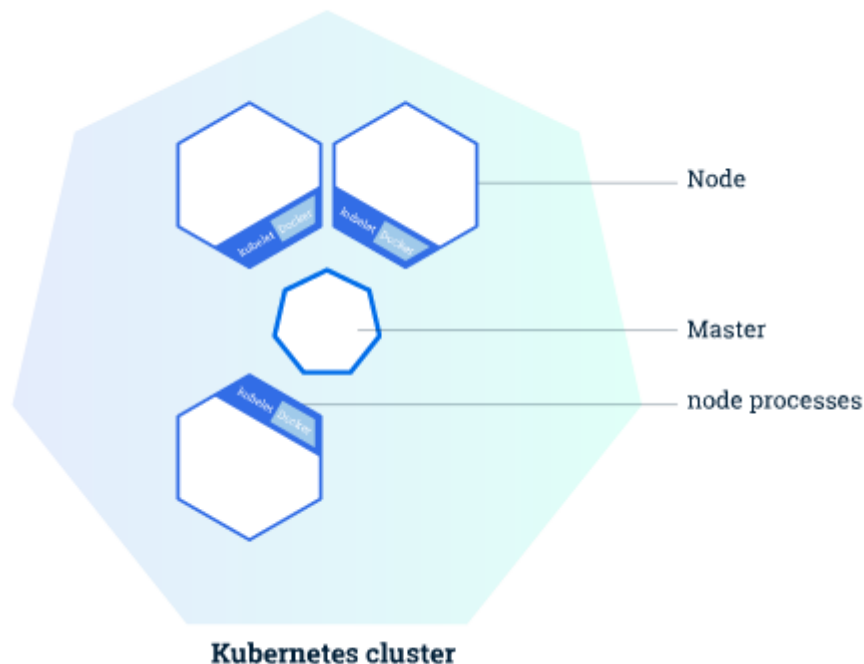
Par la suite, on choisira le **runtime Docker**.

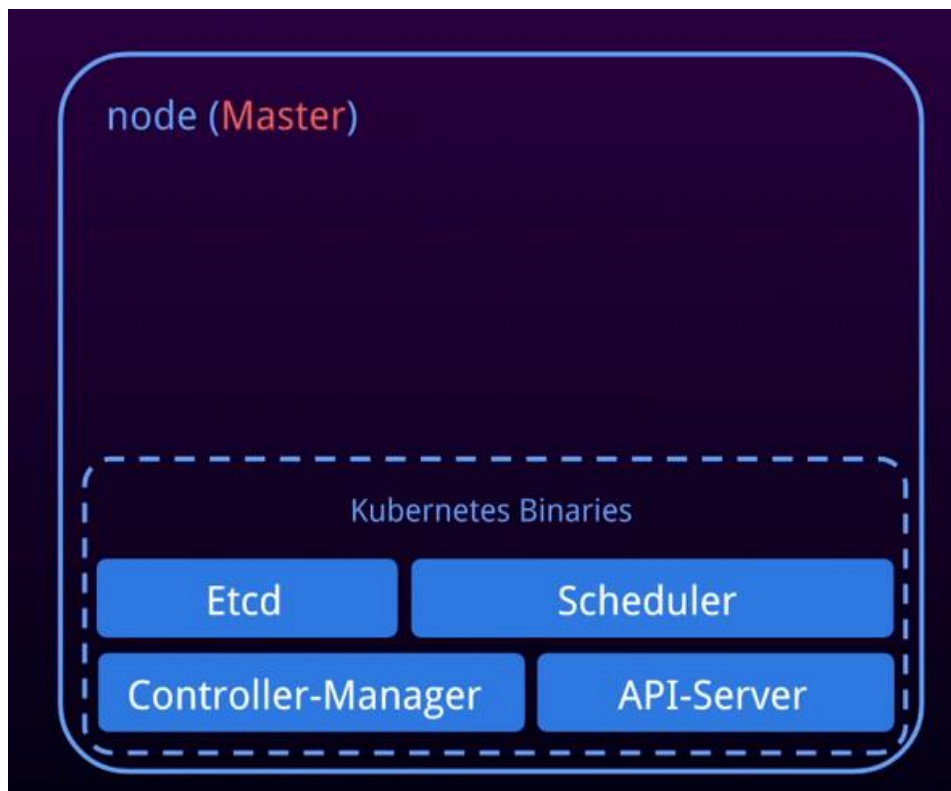
1) Les composants du Master (les processus de kubernetes)	3
1.1) Le Kube-apiserver	4
1.2) L'Etd	4
1.3) Le Kube-scheduler	4
1.4) Le Kube-controller-manager	5
1.5) Le Cloud-controller-manager	5
2) Les composants des nœuds (nodes, workers ou minions)	6
2.1) L'agent Kubelet	6
2.2) Le Kube-proxy	7
2.3) Le Container Runtime.....	8
3) Les Addons	8
3.1) Le DNS	8
3.2) L'interface utilisateur Web (tableau de bord)	8
3.3) Le Container Resource Monitoring	8
3.4) Le Cluster-level Logging	8
4) Les objets Kubernetes	9
4.1) Les pods	9
4.2) Les services	10
4.3) Les volumes	11
Les contrôleurs	11
4.4) Les ReplicaSets.....	11
4.5) Les déploiements (Deployments)	12

4.6) L'objet StatefulSet	13
4.7) L'objet DaemonSet	13
4.8) Les jobs	13
Les principales commandes (CLI) de kubernetes :.....	13
Webographie.....	16

1) Les composants du Master (les processus de kubernetes)

Les composants maîtres fournissent le plan de contrôle du cluster. Les composants principaux prennent des décisions globales sur le cluster (par exemple, la planification) et détectent et répondent aux événements du cluster (démarrage d'un nouveau pod lorsque le champ «**réplicas**» d'un replication controller n'est pas satisfait). Les composants maîtres peuvent être exécutés sur n'importe quelle machine du cluster. Toutefois, pour simplifier, les scripts d'installation démarrent généralement tous les composants principaux sur le même ordinateur et n'exécutent pas les conteneurs utilisateur sur cette machine.





1.1) Le Kube-apiserver

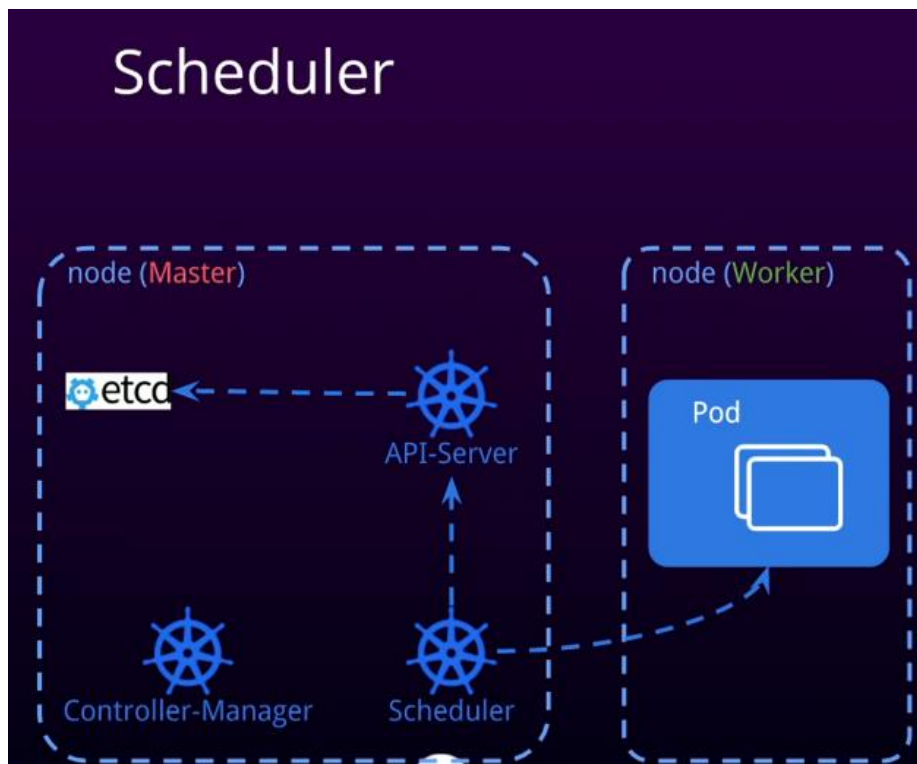
C'est un composant sur le maître qui expose l'API Kubernetes. C'est le front-end du plan de contrôle de Kubernetes. Il est conçu pour évoluer horizontalement, c'est-à-dire qu'il évolue en déployant davantage d'instances.

1.2) L'Etcd

C'est une base de données clés/valeurs cohérente et hautement disponible utilisée comme fichiers de sauvegarde de Kubernetes pour toutes les données du cluster. Il faut toujours avoir un plan de sauvegarde pour les données de etcd pour votre cluster Kubernetes.

1.3) Le Kube-scheduler

C'est un composant sur le master qui surveille les pods nouvellement créés qui n'ont pas de nœud affecté et sélectionne de ce fait un nœud sur lequel ils doivent être exécutés. Les facteurs pris en compte pour les décisions d'ordonnancement comprennent les besoins en ressources individuelles et collectives, les contraintes matérielles / logicielles / stratégiques, les spécifications d'affinité et d'anti-affinité, la localisation des données, l'interférence de charge de travail et les délais.



1.4) Le Kube-controller-manager

C'est un composant sur le master qui exécute les contrôleurs. Logiquement, chaque contrôleur est un processus distinct, mais pour réduire la complexité, ils sont tous compilés en un seul fichier binaire et exécutés en un seul processus. Ces contrôleurs incluent :

- **Le Node Controller** : responsable de la détection et de la réponse en cas de panne des nœuds.
- **Le Replication Controller** : responsable du maintien du nombre correct de pods pour chaque objet replication controller dans le système.
- **Le Contrôleur de points de terminaison** : remplit l'objet points de terminaison (c'est-à-dire rejoint les services et les pods).
- **Le Service Account & Token Controllers**: crée des comptes par défaut et des jetons d'accès API pour les nouveaux espaces de noms.

1.5) Le Cloud-controller-manager

Il exécute des contrôleurs qui interagissent avec les fournisseurs de cloud sous-jacents. Le binaire cloud-controller-manager est une fonctionnalité alpha introduite dans la version 1.6 de Kubernetes. **Cloud-controller-manager** exécute uniquement des boucles de contrôleur spécifiques au fournisseur de cloud. Vous devez désactiver ces boucles de contrôleur dans le kube-controller-manager. Vous pouvez désactiver les boucles du contrôleur en définissant l'indicateur **--cloud-provider** sur **external** lors du démarrage du **kube-controller-manager**.

Cloud-controller-manager permet au code des fournisseurs de cloud et au code de Kubernetes d'évoluer indépendamment l'un de l'autre. Dans les versions antérieures, le code Kubernetes principal dépendait du code spécifique du fournisseur de cloud pour les fonctionnalités. Dans les versions à

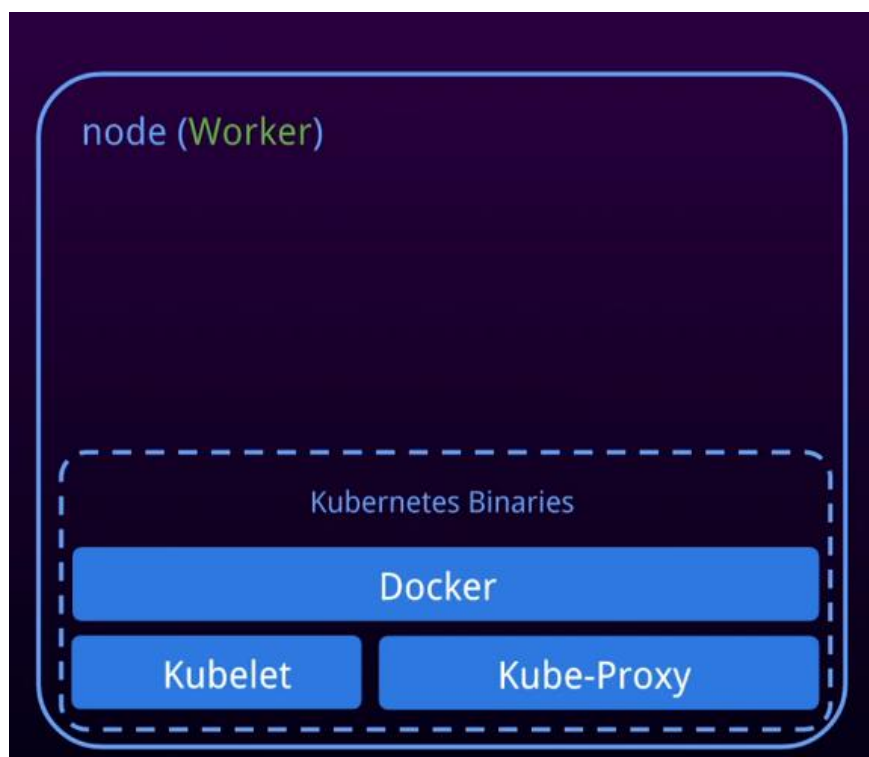
venir, le code propre aux fournisseurs de cloud devrait être géré par le fournisseur du cloud lui-même et lié au gestionnaire de contrôleur de cloud lors de l'exécution de Kubernetes.

Les contrôleurs suivants ont des dépendances de fournisseur de cloud :

- **Le Node Controller** : pour vérifier le fournisseur de cloud pour déterminer si un nœud a été supprimé dans le cloud après qu'il a cessé de répondre.
- **Le Route Controller** : pour configurer des routes dans l'infrastructure cloud sous-jacente.
- **Le Service Controller** : pour créer, mettre à jour et supprimer des équilibreurs de charge de fournisseur de cloud.
- **Le Volume Controller** : pour créer, attacher et monter des volumes, et interagir avec le fournisseur de cloud pour orchestrer les volumes.

2) Les composants des nœuds (nodes, workers ou minions)

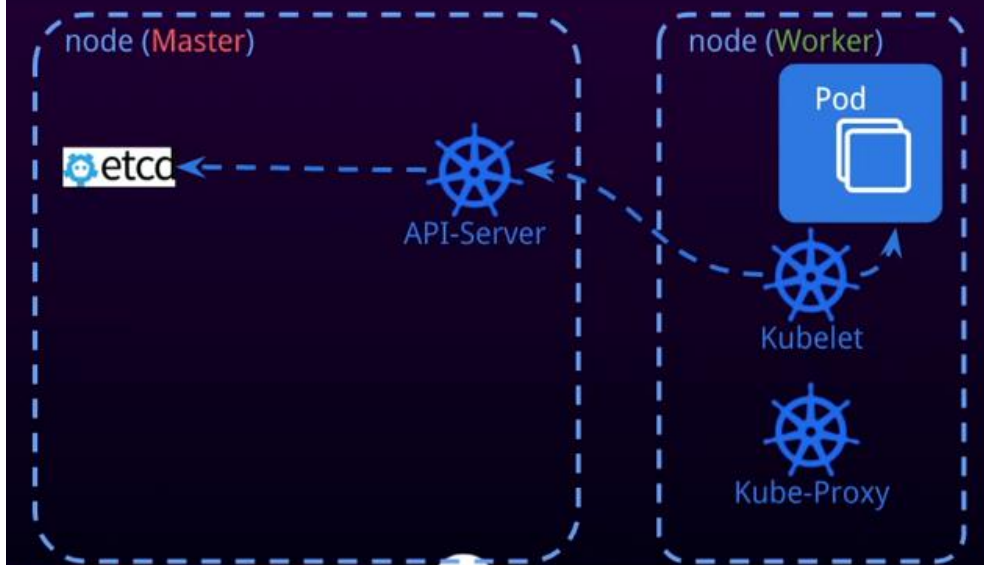
Les composants des nœuds s'exécutent sur chaque nœud, en maintenant les pods en cours d'exécution et en fournissant l'environnement d'exécution de Kubernetes.



2.1) L'agent Kubelet

Kubelet est un agent qui s'exécute sur chaque nœud du cluster. Il s'assure que les conteneurs fonctionnent dans un pod. Kubelet prend un ensemble de PodSpecs qui sont fournis par divers mécanismes et s'assure que les conteneurs décrits dans ces PodSpecs fonctionnent et sont en bonne santé. Kubelet ne gère pas les conteneurs qui n'ont pas été créés par Kubernetes.

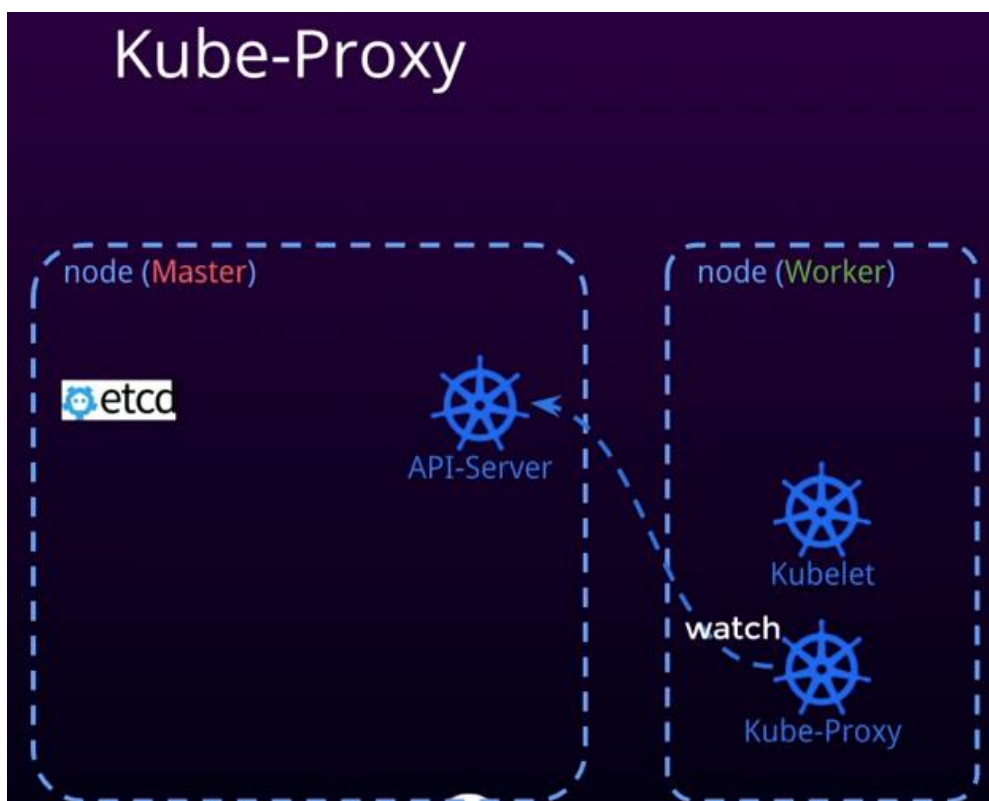
Kubelet



2.2) Le Kube-proxy

Kube-proxy active l'abstraction du service Kubernetes en maintenant les règles de réseau sur l'hôte et en effectuant le transfert de connexion.

Kube-Proxy



2.3) Le Container Runtime

Le **runtime** du conteneur est le logiciel responsable de l'exécution des conteneurs. Kubernetes prend en charge plusieurs runtimes : Docker, rkt, runc et toute implémentation OCI à l'exécution.

3) Les Addons

Les **addons** sont des pods et des services qui implémentent des fonctionnalités de cluster. Les pods peuvent être gérés par Deployments, ReplicationControllers, etc.

Les **objets addon namespaced** sont créés dans l'espace de noms kube-system.

3.1) Le DNS

Bien que les autres addons ne soient pas strictement requis, tous les clusters Kubernetes doivent avoir un DNS en cluster, car de nombreux exemples en dépendent. Un cluster DNS est un serveur DNS, en plus des autres serveurs DNS de votre environnement, qui sert les enregistrements DNS pour les services Kubernetes. Les conteneurs démarrés par Kubernetes incluent automatiquement ce serveur DNS dans leurs recherches DNS.

3.2) L'interface utilisateur Web (tableau de bord)

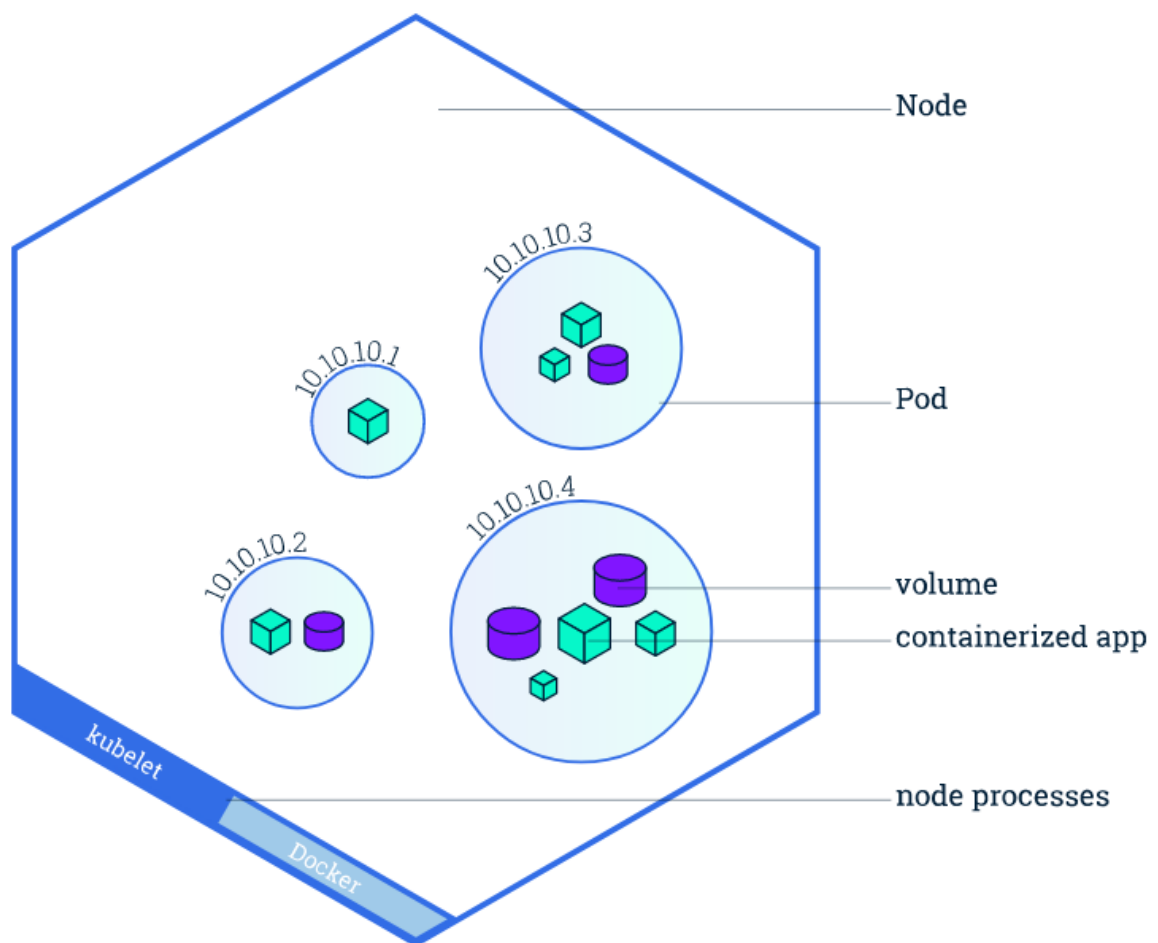
Dashboard est une interface utilisateur universelle pour les clusters Kubernetes. Il permet aux utilisateurs de gérer et de dépanner les applications s'exécutant dans le cluster, ainsi que le cluster lui-même.

3.3) Le Container Resource Monitoring

Container Resource Monitoring enregistre des métriques de séries chronologiques génériques sur les conteneurs dans une base de données centrale et fournit une interface utilisateur pour parcourir ces données.

3.4) Le Cluster-level Logging

Cluster-level Logging est responsable de l'enregistrement des journaux de conteneur dans un magasin de journaux central avec une interface de recherche / navigation.



4) Les objets Kubernetes

4.1) Les pods

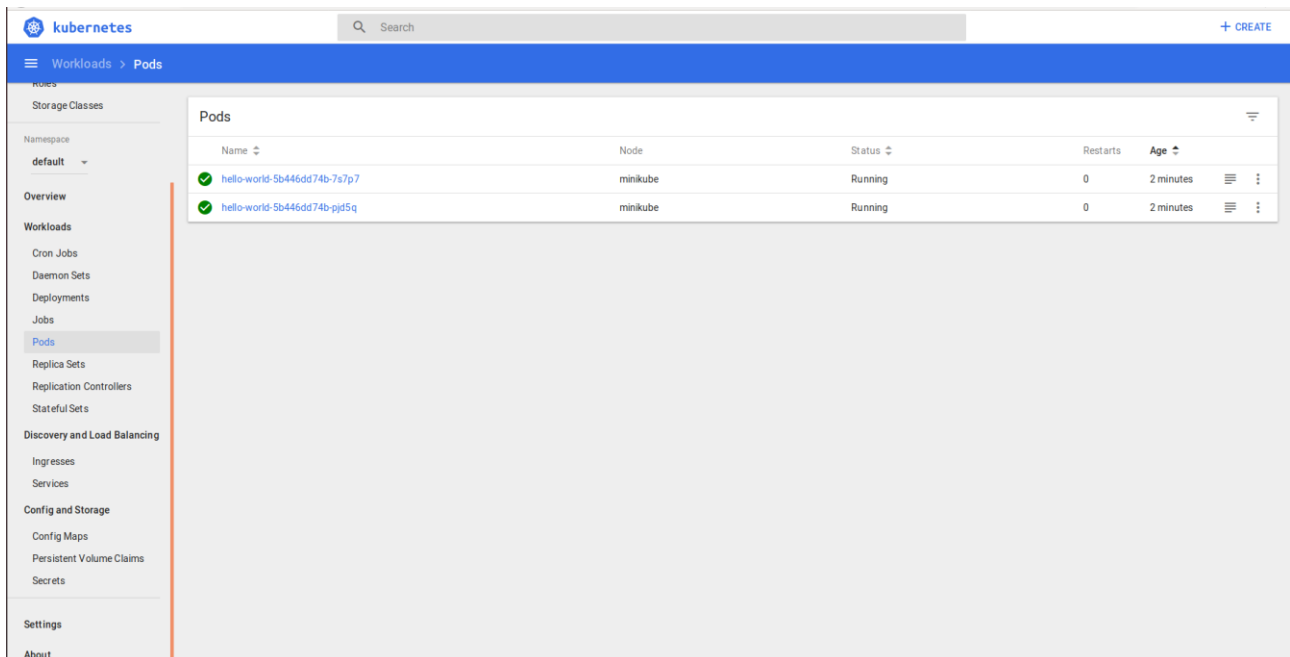
Un **pod** est le composant de base de Kubernetes, l'unité la plus petite et la plus simple du modèle d'objets Kubernetes que vous créez ou déployez. Un pod représente un processus en cours sur votre cluster.

Un **pod** encapsule un conteneur d'application (ou, dans certains cas, plusieurs conteneurs), des ressources de stockage, une adresse IP réseau unique et des options qui régissent l'exécution des conteneurs. Un pod représente une unité de déploiement, une instance unique d'une application dans Kubernetes, qui peut être constituée d'un seul conteneur ou d'un petit nombre de conteneurs étroitement couplés et partageant des ressources.

Les **pods** dans un cluster Kubernetes peuvent être utilisés de deux manières principales : soit ils exécutent un seul conteneur, ou bien ils exécutent plusieurs conteneurs mais ces derniers doivent fonctionner ensemble.

Les pods de Kubernetes sont mortels. Ils sont nés et quand ils meurent, ils ne sont pas ressuscités. Les réplicas en particulier créent et détruisent dynamiquement les pods (par exemple lors de la mise à

l'échelle ou de la réduction). Bien que chaque pod reçoive sa propre adresse IP, même ces adresses IP ne peuvent pas être considérées comme stables dans le temps. Cela pose un problème. Si un ensemble de pods (appelons-les backends) fournit des fonctionnalités à d'autres pods (appelons-les frontends) au sein du cluster Kubernetes, comment ces interfaces découvrent-elles et gardent-elles une trace des backends ? d'où la notion de services.



4.2) Les services

Un **service Kubernetes** est une abstraction qui définit un ensemble logique de pods et une politique permettant d'y accéder (parfois appelée micro-service). L'ensemble de pods ciblés par un service est généralement déterminé par un sélecteur d'étiquettes. À titre d'exemple, considérons un backend de traitement d'images qui fonctionne avec 3 répliques. Ces répliques sont fongibles, les interfaces ne se soucient pas du backend qu'elles utilisent. Bien que les pods qui composent le jeu de back-end puissent changer, les clients front-end ne devraient pas en avoir besoin ou garder une trace de la liste des backends eux-mêmes. L'abstraction du service permet ce découplage.

Pour les applications natives de Kubernetes, Kubernetes propose une simple API de points de terminaison qui est mise à jour chaque fois que le jeu de pods dans un service change. Pour les applications non natives, Kubernetes propose un pont basé sur une IP virtuelle aux services qui redirige vers les pods principaux.

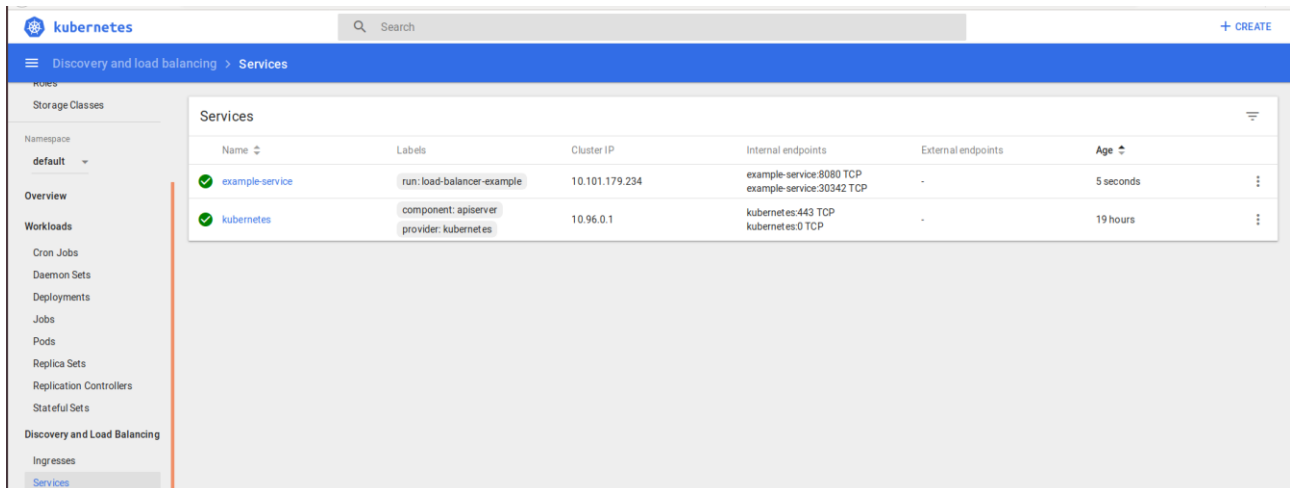
Un **service** dans Kubernetes est un objet **REST (Representational State Transfer)**, similaire à un pod. Comme tous les objets REST, une définition de service peut être envoyée à l'api-server pour créer une nouvelle instance. Par exemple, supposons que vous ayez un ensemble de pods qui exposent chacun le port 9376 et portent une étiquette "app = MyApp".

kind: Service

apiVersion: v1

metadata:

name: my-service
spec:
selector:
app: MyApp
ports:
- protocol: TCP
port: 80
targetPort: 9376



Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
example-service	run:load-balancer-example	10.101.179.234	example-service:8080 TCP example-service:30342 TCP	-	5 seconds
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	19 hours

4.3) Les volumes

Les fichiers sur disque dans un conteneur sont éphémères, ce qui pose certains problèmes pour les applications non triviales lors de l'exécution dans des conteneurs. Tout d'abord, lorsqu'un conteneur se bloque, kubelet le redémarre, mais les fichiers seront perdus. Le conteneur commence par un état propre. Deuxièmement, lors de l'ensemble de l'exécution de conteneurs dans un pod, il est souvent nécessaire de partager des fichiers entre ces conteneurs. L'abstraction Kubernetes Volume résout ces deux problèmes.

Kubernetes prend en charge plusieurs types de volumes comme **cephfs**, **configMap**, **emptyDir**, **fc (fibre channel)**, **flocker**, **hostPath**, **persistantVolumeClaim**, **secret** ainsi que les volumes des différents Cloud Service Provider.

Kubernetes prend en charge plusieurs clusters virtuels soutenus par le même cluster physique. Ces clusters virtuels sont appelés espaces de noms.

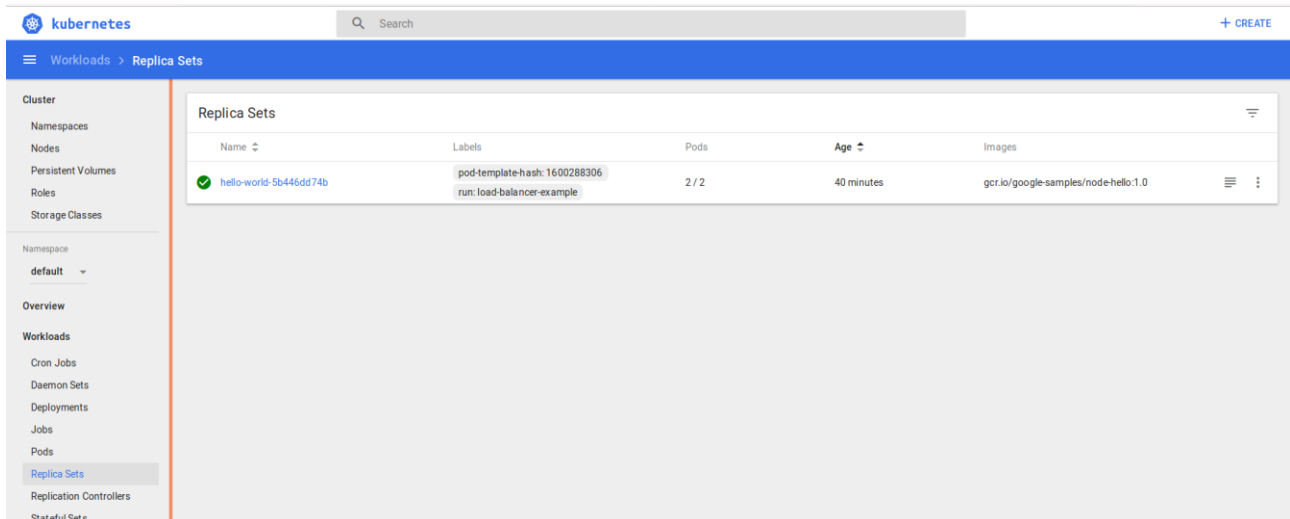
Les contrôleurs

En outre, Kubernetes contient un certain nombre d'abstractions de niveau supérieur appelées Contrôleurs. Les contrôleurs s'appuient sur les objets de base et fournissent des fonctionnalités supplémentaires et des fonctionnalités pratiques. Ils comprennent :

4.4) Les ReplicaSets

ReplicaSet est le contrôleur de réplication de nouvelle génération. La seule différence entre un ReplicaSet et un Replication Controller est la prise en charge du sélecteur. ReplicaSet prend en charge les nouvelles exigences de sélecteur basées sur des ensembles, telles que décrites dans le guide de

l'utilisateur des étiquettes, tandis qu'un contrôleur de réplication ne prend en charge que les exigences de sélecteur basées sur l'égalité.



4.5) Les déploiements (Deployments)

Un contrôleur de déploiement fournit des mises à jour déclaratives pour les **pods** et les **ReplicaSets**. Vous décrivez un état souhaité dans un objet **Deployment** et le contrôleur de déploiement modifie l'état actuel à l'état souhaité à un débit contrôlé. Vous pouvez définir des déploiements pour créer de nouveaux ReplicaSets ou pour supprimer des déploiements existants et adopter toutes leurs ressources avec de nouveaux déploiements.

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:

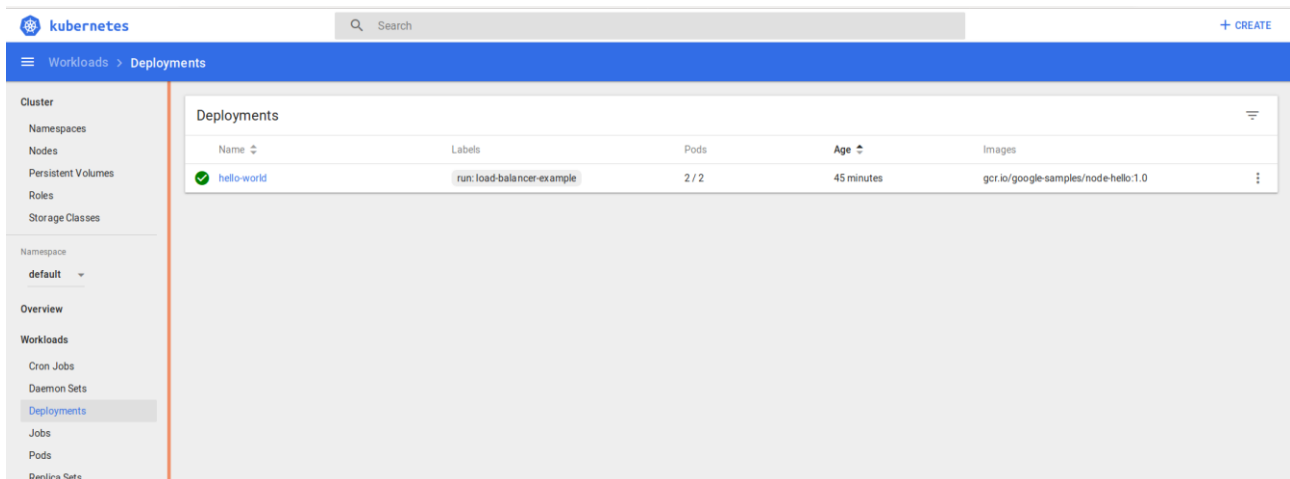
containers:

- **name:** nginx

image: nginx:1.7.9

ports:

- **containerPort:** 80



4.6) L'objet StatefulSet

Le **StatefulSet** est l'objet API de workload utilisé pour gérer les applications avec état.

4.7) L'objet DaemonSet

Un **DemonSet** garantit que tous les nœuds (ou certains) exécutent une copie d'un pod. Au fur et à mesure que des nœuds sont ajoutés au cluster, des pods leur sont ajoutés. Au fur et à mesure que les nœuds sont supprimés du cluster, ces pods sont supprimés. La suppression d'un DaemonSet nettoiera les pods créés.

Voici quelques utilisations typiques d'un **DemonSet**:

- Exécuter un démon de stockage de cluster, tel que glusterd, ceph, sur chaque nœud.
- Exécuter un démon de collecte de journaux sur chaque nœud, tel que fluentd ou logstash.
- Exécuter un démon de surveillance des nœuds sur chaque nœud, par exemple Prometheus Node Exporter, collectd, agent Datadog, agent New Relic ou Ganglia gmond.

Dans un cas simple, un DaemonSet, couvrant tous les nœuds, serait utilisé pour chaque type de démon. Une configuration plus complexe peut utiliser plusieurs DaemonSets pour un seul type de démon, mais avec des indicateurs différents et / ou des requêtes de mémoire et CPU différentes pour différents types de matériel.

4.8) Les jobs

Un **job** crée un ou plusieurs **pods** et garantit qu'un certain nombre d'entre eux se terminent avec succès. Une fois les pods terminés, le job suit les achèvements réussis. Lorsqu'un nombre spécifié d'achèvements réussis est atteint, le job lui-même est terminé. La suppression d'un job nettoie les pods créés. Un cas simple consiste à créer un objet Job afin d'exécuter de manière fiable un Pod à la fin. L'objet Tâche démarrera un nouveau pod si le premier pod échoue ou est supprimé (par exemple à la suite d'une défaillance matérielle d'un nœud ou à un redémarrage du nœud). Un Job peut également être utilisé pour exécuter plusieurs pods en parallèle.

Les principales commandes (CLI) de kubernetes :

```
`annotate` | `kubectl annotate (-f FILENAME \ TYPE NAME \ TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]`
```

Ajouter ou mettre à jour les annotations d'une ou plusieurs ressources.

`api-versions` | ``kubectl api-versions [flags]``

Lister les versions d'API disponibles.

`apply` | ``kubectl apply -f FILENAME [flags]``

Appliquer une modification de configuration à une ressource à partir d'un fichier ou à une entrée stdin

`attach` | ``kubectl attach POD -c CONTAINER [-i] [-t] [flags]``

Attacher à un conteneur en cours d'exécution soit pour afficher le flux de sortie ou interagir avec le conteneur (stdin).

`autoscale` | ``kubectl autoscale (-f FILENAME \ TYPE NAME \ TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [flags]``

Mettre à l'échelle automatiquement l'ensemble de pods gérés par un contrôleur de réplication.

`cluster-info` | ``kubectl cluster-info [flags]``

Afficher les informations de point de terminaison sur le master et les services du cluster.

`config` | ``kubectl config SUBCOMMAND [flags]``

Modifie les fichiers kubeconfig. Voir les sous-commandes individuelles pour plus de détails.

`create` | ``kubectl create -f FILENAME [flags]``

Créer une ou plusieurs ressources à partir d'un fichier ou d'un stdin.

`delete` | ``kubectl delete (-f FILENAME \ TYPE [NAME \ /NAME \ -l label \ --all]) [flags]``

Supprimer des ressources d'un fichier stdin ou spécifier des sélecteurs d'étiquette, des noms, des sélecteurs de ressources ou des ressources.

`describe` | ``kubectl describe (-f FILENAME \ TYPE [NAME_PREFIX \ /NAME \ -l label]) [flags]``

Afficher l'état détaillé d'une ou plusieurs ressources.

`edit` | ``kubectl edit (-f FILENAME \ TYPE NAME \ TYPE/NAME) [flags]``

Modifier et mettre à jour la définition d'une ou plusieurs ressources sur le serveur en utilisant l'éditeur par défaut.

`exec` | ``kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]``

Exécuter une commande sur un conteneur dans un pod.

`explain` | ``kubectl explain [--include-extended-apis=true] [--recursive=false] [flags]``

Obtenir la documentation de diverses ressources. Par exemple, des pods, des nœuds, des services, etc.

`expose` | ``kubectl expose (-f FILENAME \ TYPE NAME \ TYPE/NAME) [--port=port] [--protocol=TCP\UDP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [flags]``

Exposer un replication controller, un service ou un pod en tant que nouveau service Kubernetes.

`get` | ``kubectl get (-f FILENAME \ TYPE [NAME \ /NAME \ -l label]) [--watch] [--sort-by=FIELD] [--o \ --output]=OUTPUT_FORMAT [flags]``

Énumérer une ou plusieurs ressources.

`label` | ``kubectl label (-f FILENAME \ TYPE NAME \ TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]``

Ajouter ou mettre à jour les étiquettes d'une ou de plusieurs ressources.

`logs` | ``kubectl logs POD [-c CONTAINER] [--follow] [flags]``

Imprimer les logs d'un conteneur dans un pod.

`patch` | ``kubectl patch (-f FILENAME \ TYPE NAME \ TYPE/NAME) --patch PATCH [flags]``

Mettre à jour un ou plusieurs champs d'une ressource à l'aide du processus patch de fusion stratégique.

`port-forward` | ``kubectl port-forward POD [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N] [flags]``

Transférer un ou plusieurs ports locaux vers un pod.

`proxy` | ``kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [flags]``

Exécuter un proxy sur l'API-server de Kubernetes.

`replace` | ``kubectl replace -f FILENAME``

Remplacer une ressource d'un fichier ou de l'entrée stdin.

`rolling-update` | ``kubectl rolling-update OLD_CONTROLLER_NAME ([NEW_CONTROLLER_NAME] --image=NEW_CONTAINER_IMAGE \ -f NEW_CONTROLLER_SPEC) [flags]``

Effectuer une mise à jour continue en remplaçant progressivement le replication controller spécifié et ses pods.

`run` | ``kubectl run NAME --image=image [--env="key=value"] [--port=port] [--replicas=replicas] [--dry-run=bool] [--overrides=inline-json] [flags]``

Exécuter une image spécifiée sur le cluster.

`scale` | ``kubectl scale (-f FILENAME \ TYPE NAME \ TYPE/NAME) --replicas=COUNT [--resource-version=version] [--current-replicas=count] [flags]``

Mettre à jour la taille du replication controller spécifié.

`stop` | ``kubectl stop``

Obsolète : à la place, voir **`kubectl delete`**.

`version` | ``kubectl version [--client] [flags]``

Afficher la version de Kubernetes exécutée sur le client et le serveur.

Webographie

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

[https://fr.wikipedia.org/wiki/Docker_\(logiciel\)](https://fr.wikipedia.org/wiki/Docker_(logiciel))

<https://www.lebigdata.fr/docker-definition>

<https://coreos.com/rkt/>

<http://cri-o.io/>

<https://katacontainers.io/>

<https://kubernetes.io/docs/home/?path=users&persona=app-developer&level=foundational>

<https://kubernetes.io/docs/tutorials/>

<https://kubernetes.io/docs/tutorials/hello-minikube/>

<https://www.tutorialspoint.com/kubernetes/index.htm>

<https://www.youtube.com/watch?v=QDvxQWKY17s>