



Chapitre 4 : Introduction Kubernetes

Docker et Kubernetes

CCI Strasbourg

Chapitre 4 : Introduction Kubernetes

Objectifs

En suivant ce chapitre, nous allons aborder Kubernetes avec les modules suivants :

- 4.1 Historique et raison d'être de Kubernetes
- 4.2 Concepts fondamentaux : nœuds, pods, services, replica sets
- 4.3 kubectl : commandes de base

Chapitre 4 : Introduction Kubernetes

Module 4.1

Historique et raison d'être de Kubernetes

Chapitre 4 : Introduction Kubernetes

Module 4.1 – Historique et raison d'être de Kubernetes

Sommaire :

- Présentation de Kubernetes
- Historique de Kubernetes



Chapitre 4 : Introduction Kubernetes

Module 4.1 – Historique et raison d'être de Kubernetes

Présentation de Kubernetes



Chapitre 4 : Introduction Kubernetes

Module 4.1 – Présentation de Kubernetes

Kubernetes (grec pour « timonier » ou « pilote », abrégé en K8S) est un outil très puissant qui fournit un environnement de gestion **focalisé sur le conteneur**.

Pour simplifier, on peut dire que Kubernetes rajoute une couche sur une solution de conteneurisation comme Docker par exemple.

Kubernetes n'est pas lié à Docker, il peut fonctionner avec d'autres solutions de conteneurisation (lxc, containerd, CRI-O)



Chapitre 4 : Introduction Kubernetes

Module 4.1 – Présentation de Kubernetes

Pourquoi Kubernetes ?

La couche supplémentaire qu'apporte Kubernetes permet :

- d'orchestrer des conteneurs (comme docker swarm)
- de créer de l'abstraction avec la notion de services (beaucoup plus avancé que Docker Compose)
- d'apporter de la haute disponibilité avec un système déclaratif et non pas impératif.
- De fournir la possibilité de scaler (augmenter ou réduire le nombre d'instance en fonction de la demande) pour supporter la charge.

Globalement Kubernetes est une solution pour mettre en production (déployer, gérer, surveiller) des conteneurs.

Chapitre 4 : Introduction Kubernetes

Module 4.1 – Présentation de Kubernetes

L'orchestration de conteneurs ?

L'orchestration des conteneurs permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs.

Chapitre 4 : Introduction Kubernetes

Module 4.1 – Présentation de Kubernetes

Comment obtenir / utiliser Kubernetes ?

Kubernetes peut fonctionner en mode cluster ou en mode test :

- **cluster** : un master et des nœuds esclaves sur des machines distinctes
- **mode test** : tous les composants sur la même machine.

Pour utiliser Kubernetes, il y a plusieurs options :

- il faut avoir un fournisseur (Google Cloud, Amazon AWS, azure)
- avoir sa propre infrastructure pour héberger la solution sur des serveurs physiques
- en mode test sur son PC, avec minikube, Docker Desktop, etc...

Chapitre 4 : Introduction Kubernetes

Module 4.1 – Historique et raison d'être de Kubernetes

Historique de Kubernetes

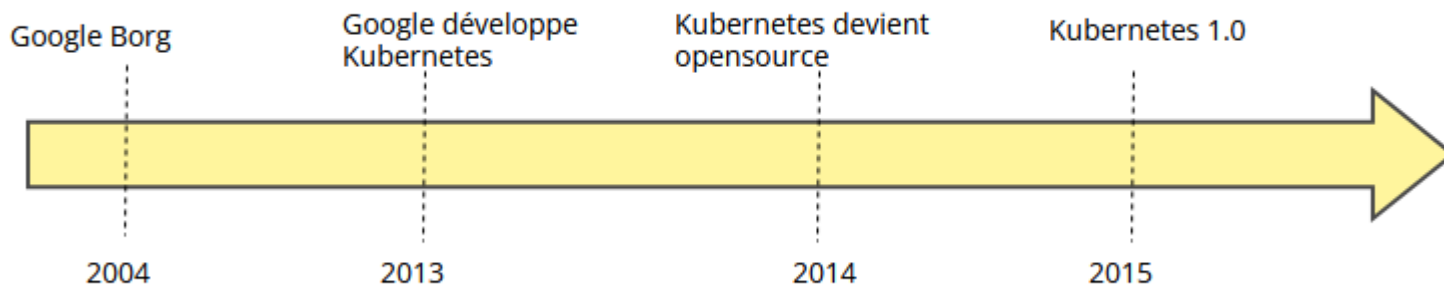


Chapitre 4 : Introduction Kubernetes

Module 4.1 – Historique de Kubernetes

Un peu d'histoire

Tout commence par un projet interne à Google pour la gestion des conteneurs Google Borg.



- Depuis la version 1.0, il y a environ 3 releases par an, et nous sommes actuellement à la version 1.28

Chapitre 4 : Introduction Kubernetes

Module 4.1 – Questions ?



Chapitre 4 : Introduction Kubernetes

Module 4.2

Concepts fondamentaux : nœuds, pods, service, replica sets

Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replica sets

Sommaire :

- Architecture de Kubernetes
- Les nœuds (ou nodes) et ses composants
- Pod – l'unité de base
- Les services
- Replica sets – contrôle des instances
- Les namespaces



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replica sets

Architecture de Kubernetes



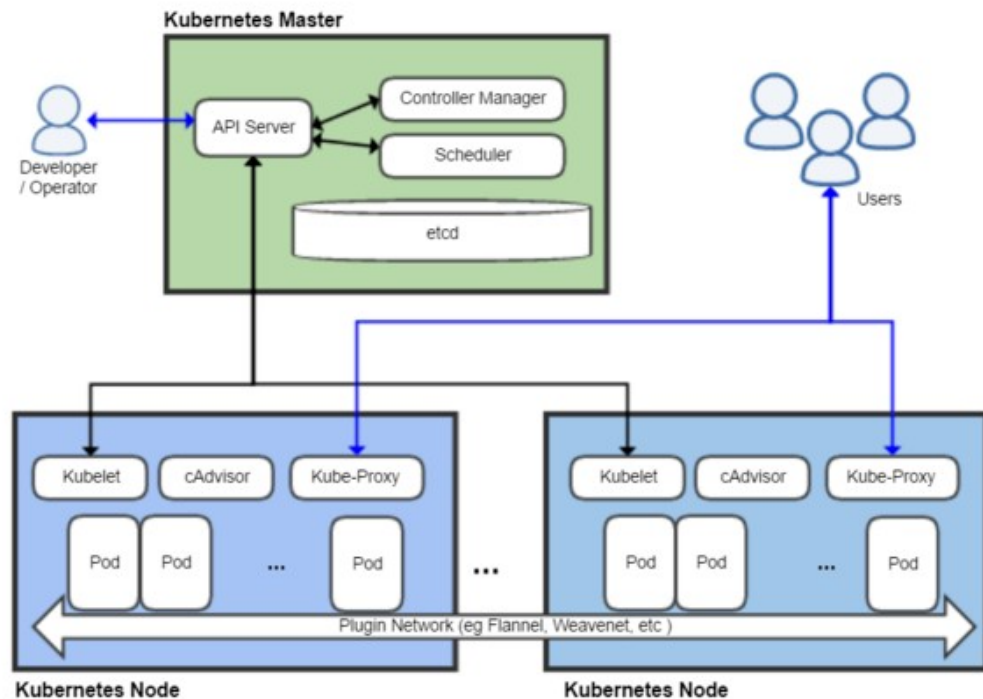
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Architecture de Kubernetes

Architecture de Kubernetes

L'architecture de Kubernetes est un cluster basé sur un (ou plusieurs) master et un (ou plusieurs nœuds).

- L'ensemble des composants du master est appelé **control plane (plan de contrôle)**
- Le plan de contrôle a pour objectif de
 - contrôler les nœuds et d'assigner les tâches
 - vérifier si les conteneurs peuvent fonctionner avec les ressources disponibles des nœuds et donc s'assurer que votre configuration est respectée



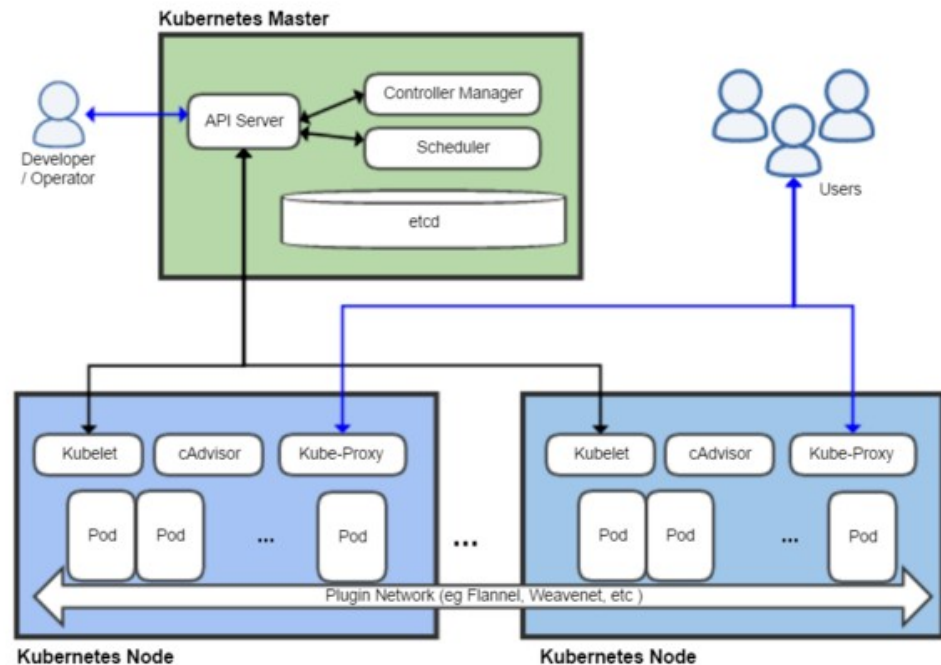
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Architecture de Kubernetes

Architecture de Kubernetes – Le plan de contrôle

Il est constitué des éléments suivants :

- **API Server** : il s'agit de la partie frontale du plan de contrôle, il permet de communiquer avec le cluster et configurer les différentes ressources.
- **etcd** : une base de données NoSQL clé-valeur qui permet de stocker les données du cluster
- **Scheduler** : Le processus qui permet d'affecter des pods à des nœuds (nodes) et qui vérifie l'intégrité du cluster
- **Controller Manager** : Processus qui effectue des boucles de contrôle et qui assure l'exécution des différents contrôleurs du cluster (node controller, replication Controller, Endpoints Controller et Service Account et Token Controllers).



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replica sets

Les nœuds (ou nodes) et ses composants



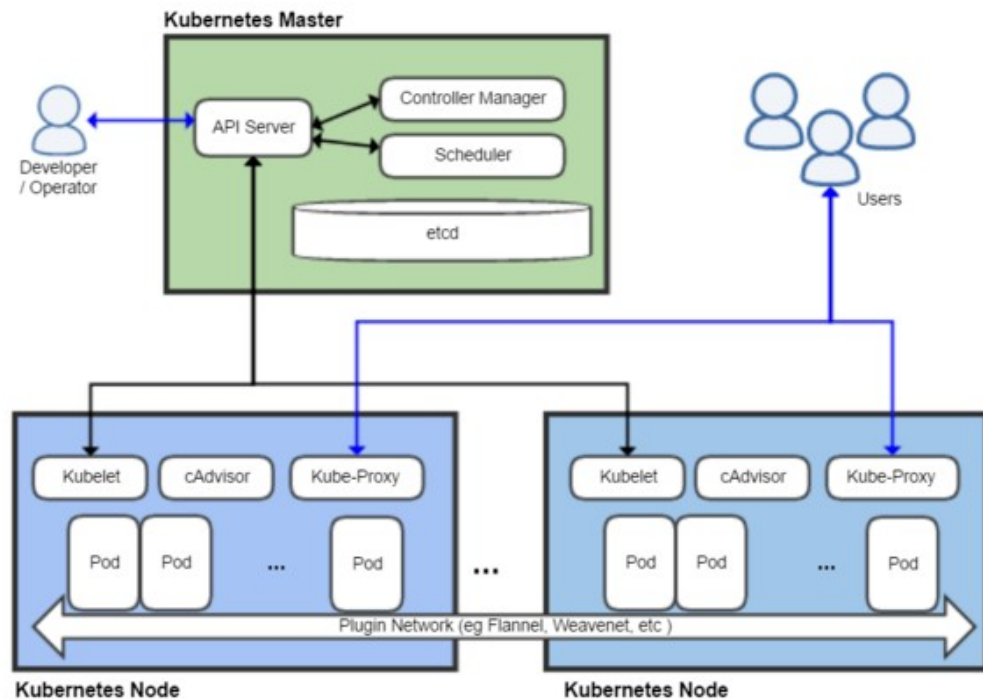
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les nœuds (ou nodes) et ses composants

Les nœuds dans Kubernetes

Un nœud est une machine physique ou virtuelle (même si une machine pourrait héberger plusieurs nœuds) qui fait partie du cluster Kubernetes.

Les nœuds sont des unités de calcul de base du cluster et ils fournissent l'environnement d'exécution pour des applications conteneurisées.

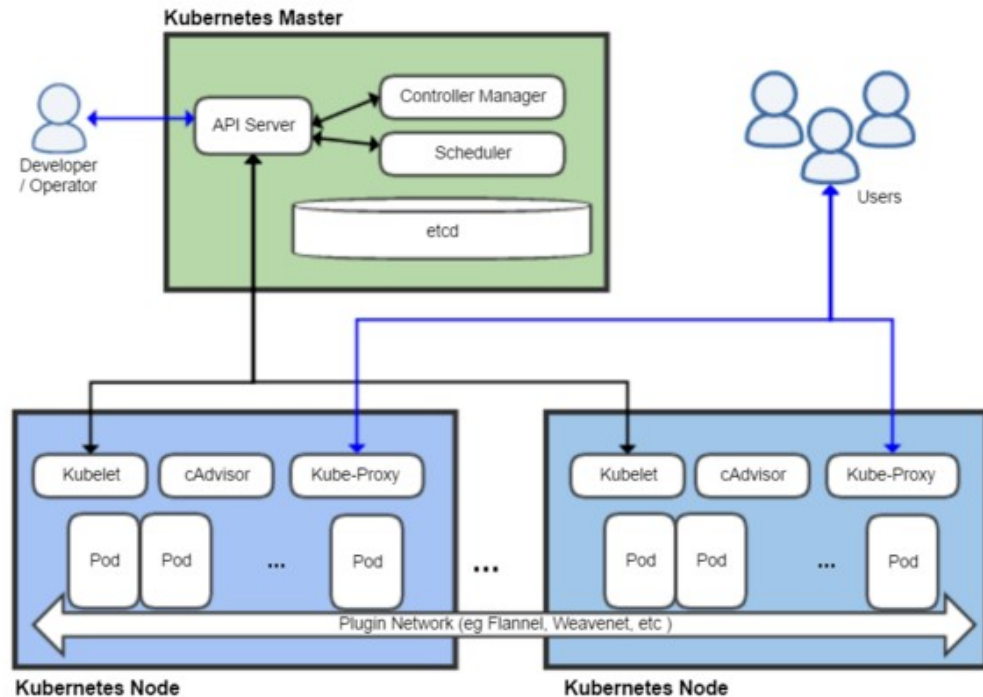


Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les nœuds (ou nodes) et ses composants

Les composants d'un nœud

- **Moteur d'exécution de conteneurs (runtime ou CRI) :**
Permet d'exécuter des conteneurs (Docker est un exemple possible de moteur qui est utilisé par k8s)
- **Kubelet :** c'est un agent qui s'exécute sur chaque nœud, il surveille l'état du nœud, il effectue la communication avec le master Kubernetes (via son API Server) et pilote le CRI du nœud.
- **cAdvisor :** ce composant collecte, agrège et expose les statistiques / informations des conteneurs qui sont exécutés dans le nœud.
- **Kube-Proxy :** avec d'autres composants (codeDNS, CNI), il gère **en partie** le réseau sur le nœud, ce qui permet aux Pods de communiquer entre eux et aussi avec des services externes.
- **Pod :** c'est le plus petit élément qui peut être déployé dans Kubernetes et il représente une application qui est constituée d'un ou plusieurs conteneurs qui partagent les mêmes ressources (réseau, stockage).



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replica sets

Pod – l'unité de base



Chapitre 4 : Introduction Kubernetes

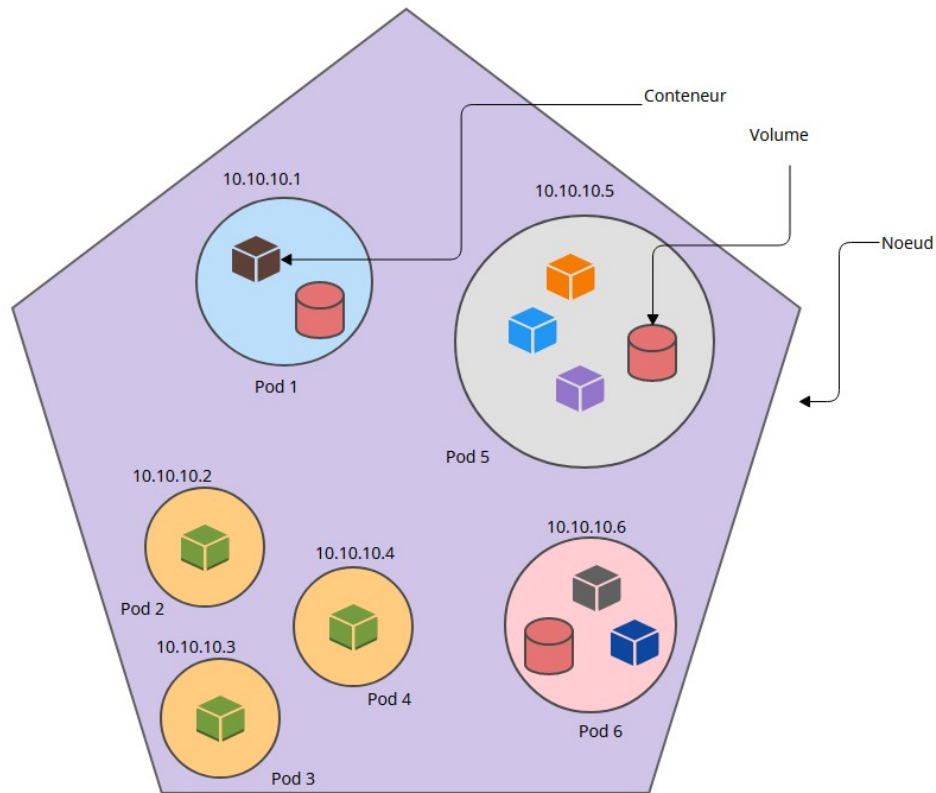
Module 4.2 – Les nœuds (ou nodes) et ses composants

Présentation du Pod

Un pod est un groupe d'un ou plusieurs conteneurs ayant du stockage / réseau partagé qui représente une application.

Il peut être vu comme une « machine hôte logique » spécifique à une application.

Les pods peuvent partager des volumes afin de persister les données des conteneurs.



Chapitre 4 : Introduction Kubernetes

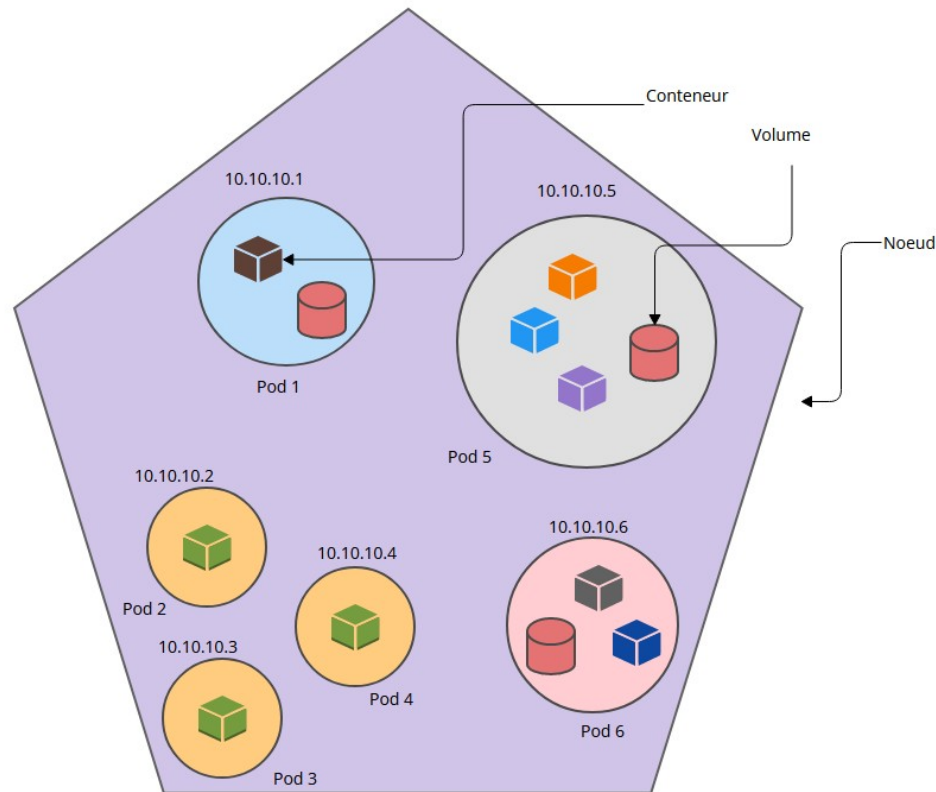
Module 4.2 – Les nœuds (ou nodes) et ses composants

Caractéristiques du Pod

Le **pod** dispose :

- d'un **ID unique** qui permet de l'identifier
- d'une adresse **IP propre** et ses conteneurs partagent cette adresse et un espace de ports (ils peuvent communiquer via « localhost »).
- D'un **cycle de vie**

Tout comme les conteneurs, les Pods sont plutôt considérés comme éphémères, si un pod doit être déplacé sur un nouveau nœud par exemple, il sera en réalité remplacé par un nouveau pod qui dispose d'un nouvel ID.



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les nœuds (ou nodes) et ses composants

Le cycle de vie d'un Pod

- Les Pods disposent d'un objet status (PodStatus) qui contient un champ « phase ».
- La phase signifie l'état dans lequel se trouve le Pod :
 - **Pending** : le pod est accepté par Kubernetes, mais il n'est pas encore créé (les conditions ne sont pas encore remplies pour la création, téléchargement des images en cours par exemple)
 - **Running** : le pod est affecté à un nœud et tous les conteneurs sont créés et au moins un conteneur est en cours d'exécution.
 - **Succeeded** : tous les conteneurs ont terminé avec succès leurs tâches et ne seront pas redémarrés
 - **Failed** : tous les conteneurs sont terminés et au moins un conteneur a terminé en échec (status!= 0).
 - **Unknown** : pour une raison inconnue, l'état du pod n'est pas obtenu.

Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replica sets

Les services



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

Présentation des services

Un service est une abstraction qui définit un ensemble logique de Pods.

Il fournit un moyen stable pour accéder aux applications des Pods à travers un **endpoint**, même si les Pods peuvent changer dynamiquement en raison de l'état du cluster ou des opérations de mise à l'échelle.

L'intérêt majeur des services est donc **la stabilité des applications**, en effet si un ensemble de Pods offrent une fonctionnalité, comme nous l'avons vu précédemment les Pods disposent d'un cycle de vie (ils naissent et meurent) et donc la fonctionnalité pourrait ne plus être disponible car la ressource ne l'est plus à cet endroit.

Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

Type de service

Il existe 4 types de services dans Kubernetes :

- **ClusterIP** : expose le service sur une IP interne du cluster, le service est alors uniquement accessible à partir du cluster. Il s'agit du type par défaut.
- **NodePort** : expose le service sur l'IP de chaque nœud sur un port statique (le NodePort). Le service est donc accessible depuis l'extérieur du cluster en demandant *nodeIp:nodePort*
- **LoadBalancer** : Expose le service en externe à l'aide d'un équilibreur de charge d'un fournisseur de cloud.
- **ExternalName** : Mappe le service sur un nom externe, par exemple *foo.service.com*

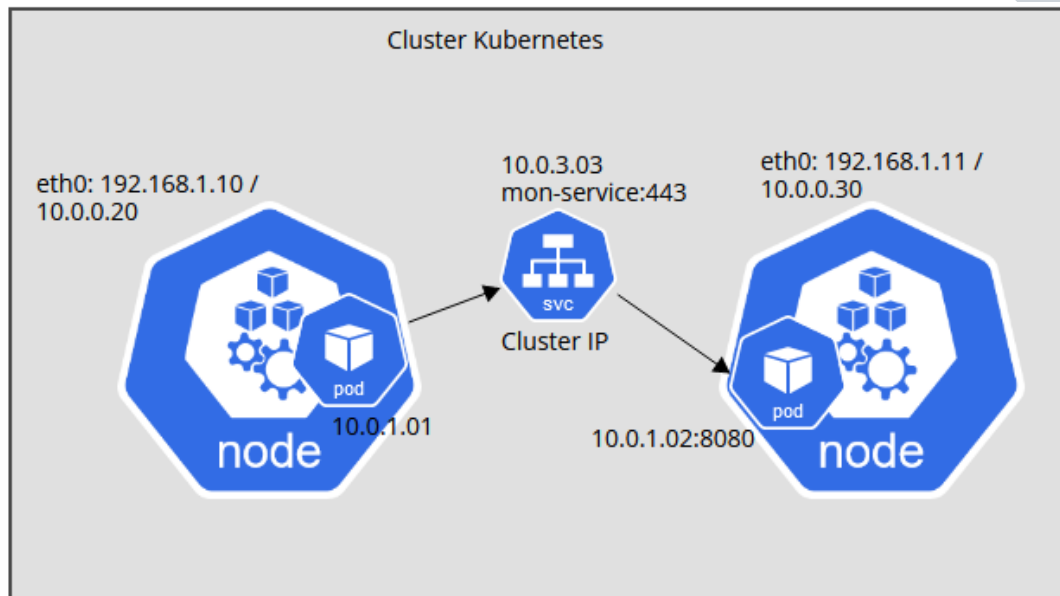
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

Le ClusterIP

Le clusterIP est le service par défaut dans Kubernetes, il donne un service à l'intérieur du cluster comme l'illustre le schéma ci-dessous :

- Dans cet exemple, le clusterIP va exposer l'application du pod de gauche à travers le service « mon-service ».



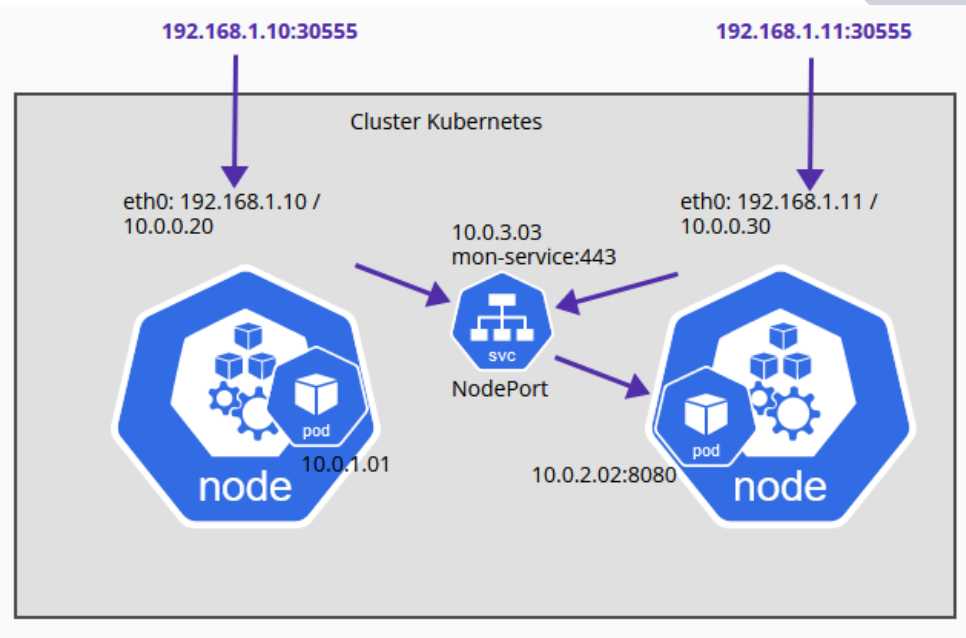
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

NodePort

Un NodePort permet de router un flux externe directement vers un Pod, il va ouvrir un port sur tous les nœuds pour ensuite aiguiller vers le pod :

- Dans cet exemple, on ouvre le port 30555 à l'extérieur du cluster pour l'ensemble des nœuds et ce port va rediriger vers le service « mon-service » qui va utiliser le Pod
- Pour information, le range par défaut des nodePort est **30000-32767**, ce qui signifie que Kubernetes attribue automatiquement un port de ce range par défaut.



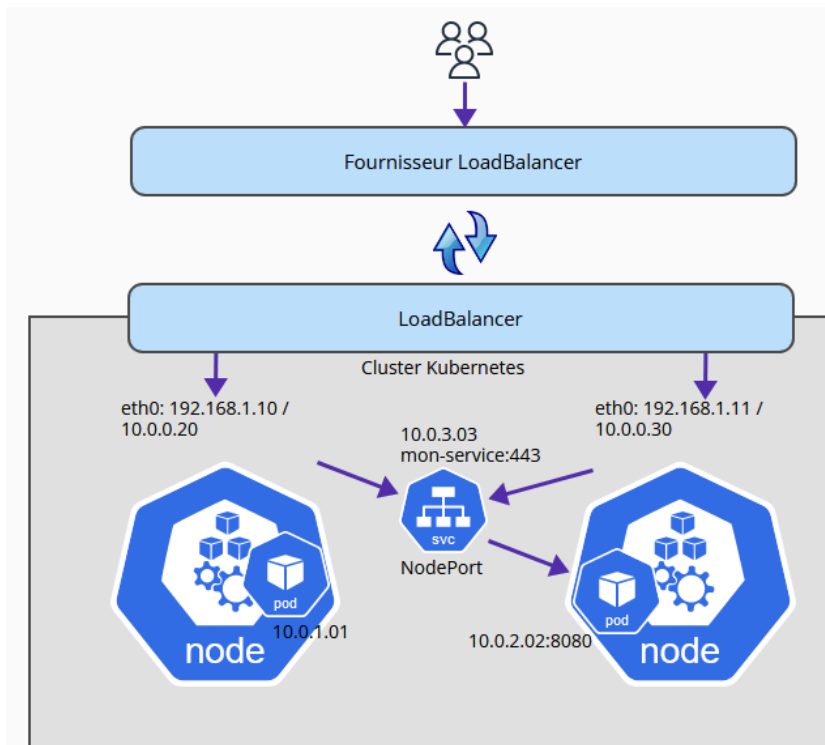
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

LoadBalancer

Un LoadBalancer va exposer un service à travers un répartiteur de charge qui va établir vers quel destination le flux sera acheminé.

- Le fournisseur LoadBalancer peut-être :
 - Azure
 - Google Load Balancing
 - Elastic Load Balancing (Amazon AWS)
- Une solution alternative pourrait d'exposer le service à travers un nodePort et de remplacer la partie LoadBalancer par un haproxy



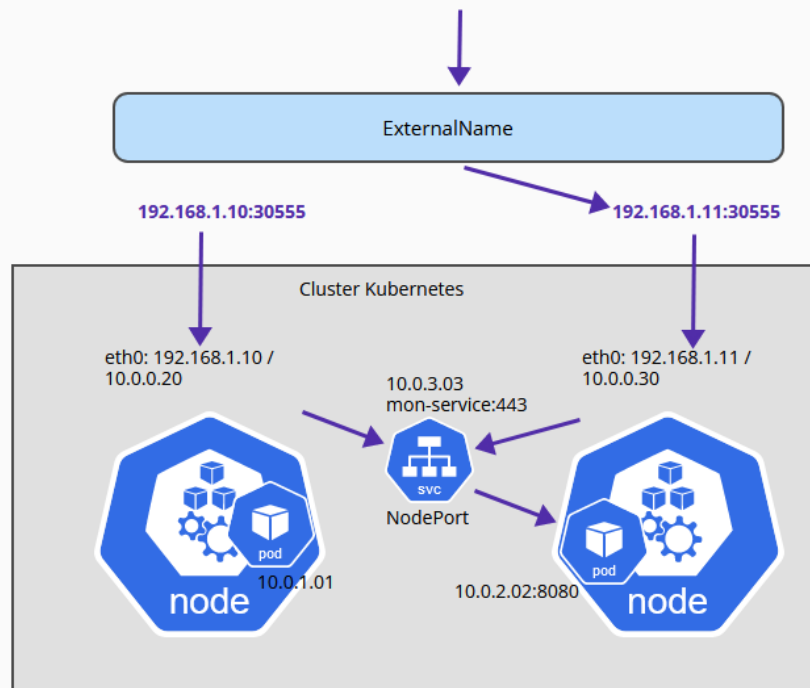
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

ExternalName

Un ExternalName va permettre d'accéder à un service à travers un alias DNS défini en dehors du cluster Kubernetes.

Ce type de service est recommandé lorsque qu'il faut par exemple migrer des applications existantes vers le cluster sans avoir de perte de service.

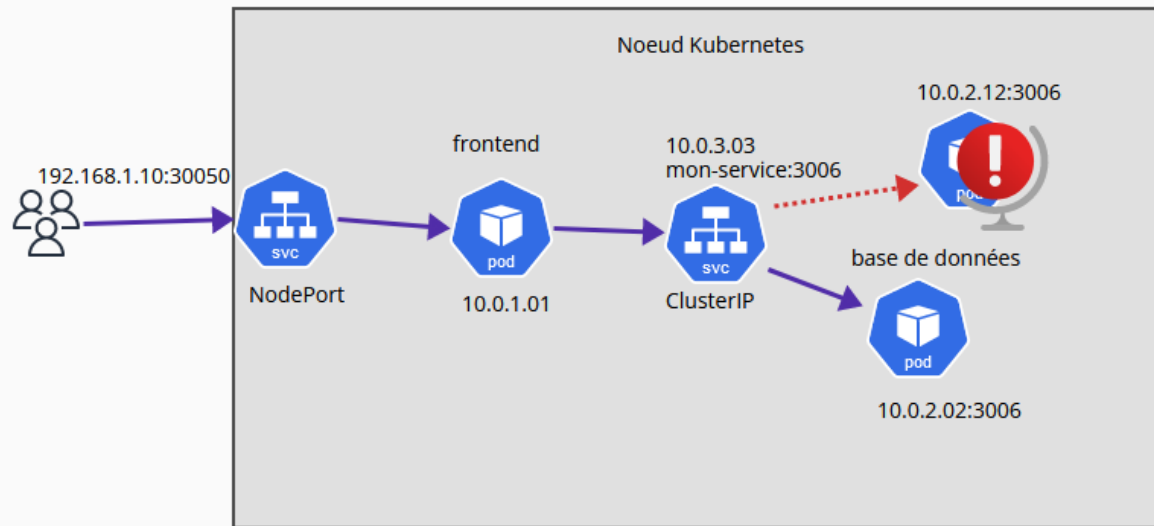


Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les services

Conclusion

Les services les plus utilisés sont les **cluster IP** et les **nodePort**, ils permettent d'exposer les applications avec des moyens stables.



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replicaset

Replicaset – contrôles des instances



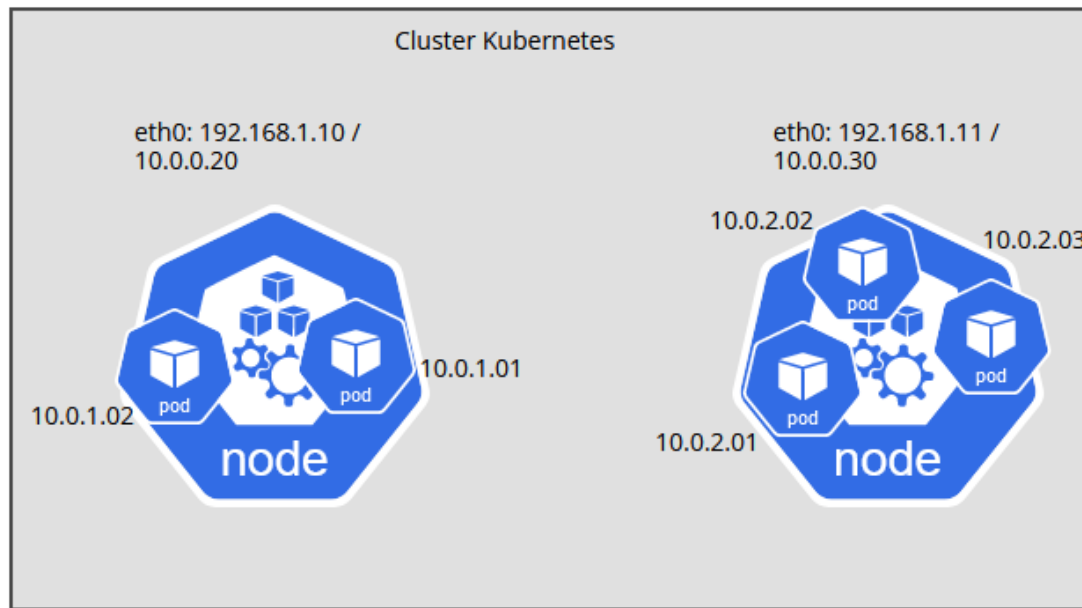
Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les replicaset

Présentation des replicaset

Un ReplicaSet est un moyen qui garantit un nombre de demandé de répliques (pods) d'une application au sein du cluster. L'objectif est de maintenir un **nombre stable de Pods identique** à un moment donné.

L'exemple suivant montre 5 replicas d'un même pod réparti sur 2 noeuds



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les replicaset

Fonctionnement des replicaset

Lorsqu'un Pod est supprimé, si il est géré à travers un replicaSet alors le contrôleur de replicas va alors automatiquement recréer le Pod pour atteindre le nombre déclaré.

Chapitre 4 : Introduction Kubernetes

Module 4.2 – Concepts fondamentaux : nœuds, pods, services, replicaset

Namespaces



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les namespaces

Présentation des namespaces

Un namespace (ou espace de noms) va permettre de définir un cadre pour définir nos ressources dans Kubernetes.

De cette manière, un pod avec un nom identique pourra être déclaré dans deux namespaces différents pour éviter les conflits.

Chapitre 4 : Introduction Kubernetes

Module 4.2 – Questions ?



Chapitre 4 : Introduction Kubernetes

Module 4.3

Kubectl : commandes de base

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl : commandes de base

Sommaire :

- Kubectl - présentation
- Kubectl – les commandes
- Manipulation avec Kubectl (TP)



Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl : commandes de base

Kubectl – présentation



Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl - présentation

Kubectl

Kubectl est un outil en ligne de commande officiel de Kubernetes, il permet d'interagir avec un ou plusieurs cluster(s) Kubernetes pour effectuer l'ensemble des opérations courantes : superviser, déployer et gérer les applications / pods.

Kubectl est disponible sur Windows, macOS et Linux. Lors de l'installation, il est recommandé d'utiliser une version compatible avec votre version de cluster (dans une version mineure du cluster cible). Par exemple, si votre kube-apiserver est 1.28, alors kubectl doit être utilisé en version 1.29, 1.28 et 1.27.

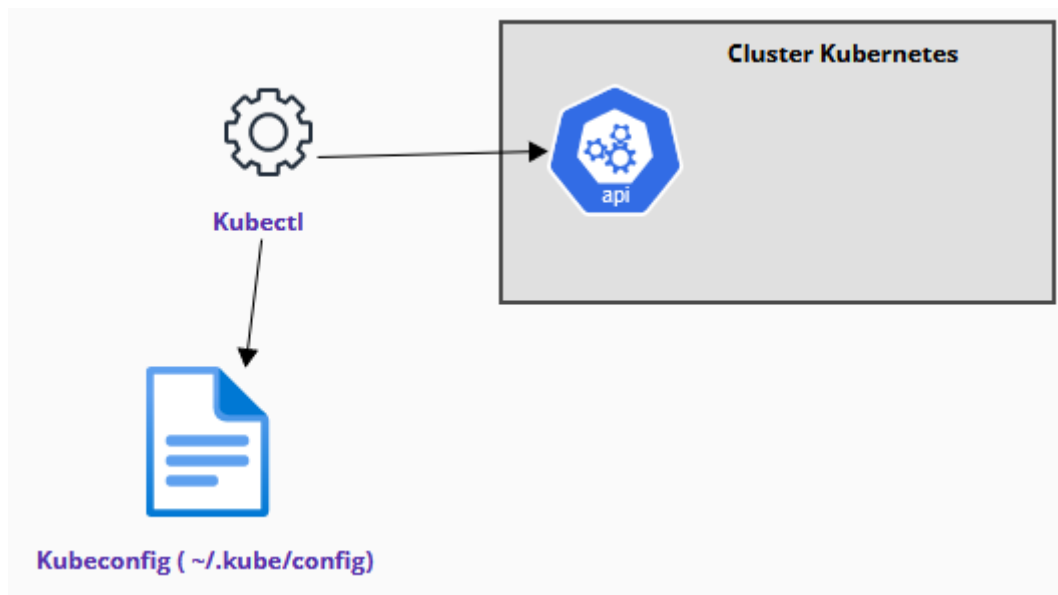
Pour fonctionner, kubectl nécessite l'utilisation d'un fichier de configuration **kubeconfig**. Dans ce fichier de configuration, nous allons notamment trouver la description des clusters, contextes et utilisateurs.

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl - présentation

Kubectl

Kubectl donne des instructions au cluster Kubernetes à travers le composant **API Server**.



Module 4.3 – Kubectl - présentation

Le fichier de configuration indique la définition du cluster et indique un nom pour y faire référence.

- le **context** qui est défini peut être vu comme un profil, il va définir une association entre :

- l'utilisateur qui va être utilisé
- Le cluster qui va être utilisé
- Le namespace qui va être utilisé

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTIiBDRVJUSUZjQ0FURS0tLS0tCk1JSm9kdmNOQVFFTEJRQXkGEVEVUTUjFR0ExvUUKQXhNS2EzVmlaWEp1W1hSbGN6QWVGdzB5TXpEApFekFSQmdOVKvJBTVRDBXQxwW1WewJtVjBawE13Z2dFau1BMEdDU3FHU01m0RRRUjBUV
    mFnHdBByNm1oVgpBvNtDHVRNEVnZmJLcLX3tExRRjQUpXHMorJZndzQkQKt1hPbwtwGZyL0
    LytST2t1eWUwSn1oanJBVQp7jNXRCtTNzUkcZRGw5bzFFMm10U205NTd1MGnQc2dTS0Y5an
    bu12NXVt23psY3U1YkY1a0NHZm1vTHiYzU5jQ3A4My9iCHNOwF7ja2x2RnpxMDdEQUpNyM
    dFNF1RV6K2FCN1NnK1pGb1dmQn13czRBVvkxHUnJnQpQc3pwV2ExL3NaTHF3R01yUj1ka0
    SUNwREFQcKJnT1ZIuK1CQWY4RUJjUQRBUUgVtIuIwR0ExVwREZ1FXQkJUN0JTQ2dBT01NR1
    SmxjbTVszEdwEk1BMEdDU3FHU01m0RRRUjDd1VBQTRJQkFRQTvJr3A5Rzb0MgpFd3UwcjZ
    R25VHwcnb1pFN0RxxSFkyN1picz15CjNRC1VBdk5vbXROwJhQU1F1R1VKS9UN1ZHQ1poYt
    L3R3YQ9ybnpVKYkFaaVNVtZfZzQ3RGdS9qZFVuVDBRY25uRVFYRZKZJUEt1RD30TjZ4dHZtQW
    Q1o4QjF1MFdTD09hbDkycHvtZT2k2EhtS5C9reUt1U1pVvFnfjC1FrYXJkaWR2d040MjQvRE
    VjdYUFlNSMudKZMKb3E3T1d0OFY4cW95Si0tLS0tRU5EIENFU1RJRklDQVRFLS0tLS0tCg==
    server: https://kubernetes.docker.internal:6443
  name: docker-desktop
contexts:
- cluster: docker-desktop
  user: docker-desktop
  name: docker-desktop
current-context: docker-desktop
kind: Config
preferences: {}
users:
- name: docker-desktop
  user:
    client-certificate-data: LS0tLS1CRUdJTIiBDRVJUSUZjQ0FURS0tLS0tCk1JSm9kdmNOQVFFTEJRQXkGEVEVUTUjFR0ExvUUKQXhNS2EzVmlaWEp1W1hSbGN6QWVGdzB5TXpE
    GekFWQmdOVKvJB1IREbk41YzNSbGJUCkRHRzWE4wW1hEk1Sc3dHUV1EV1FRREV4Smt1Mk5yW
    RRUjBUUVBQTRJQkR3QXdnZ0VlQW9jQkFRFRdX0XOEViTEQ1SHh2T1MKcXZ3Qi9PS1BRME1RZ
    jOEwNUBBzQ3RYcG5HZA0zQ35XhKOU9K5XfSRUpqc1ZtOUkzbvNSdnV2MUZWc1UwSXZuNmJNe
    mUFPpRnQrSeXfZ2Y5UGVpMHHfZ1pEM1U1Nkhpl3BLdFA3U1pEU31ZGdhU1R4bFf3S3Z4Z
    oUUVVkMXI3UmRmdUFJQ1RKbzBvSjddqR3dKVE1zTApPZThZnmFvXNC9PYnjvVDFvSzHmYmtLW
    xVktqCkcdNm1QQ2tWQmdNqkFBR2pkVEJ6TUE0R0ExVwREd0VCL3dRRUF3SUZvREFUQmdOV
    BakFBTUt1R0EXVWRJ3d1FZTUJhQUZUC0ZJS0FBNHd3WmRkQWZrc2dxOGsvVgpXQWNovTUIwR
    EQU5CZ2txaGtpRz13MEjBUXNGCkFBT0NBuUvBS0Q0a2pmMud0N1VoZVUuR0VdUWw0bT10N
    tM1Z0UuM3HRGZDbb0xekE2U0pRaTh4QWdMTzdzUzVBDZF15T3NiSGVWZU93S1YhRYfHBY25NM
    xWUtPd3C4Uk9kwdVUuM3hjT29rdzntDvHLOtGwLzNydmTICm9CSFBKEtNMNItKeFZ2a2RHR
    GOFNSNdS3RMKdQ0RjvJm3RKL2meKJZ2mEjY3ZGdwTVVRdFIT12PMXVjC9G6E7Z0SEZjakNKO
    1OGZSaEppazRyTjd1UXc9PQotLS0tLUVORCDBRVJUSUZjQ0FURS0tLS0tCg==
    client-key-data: LS0tLS1CRUdJTIiBSU0EGUgUjV3VkrFBR1RvktLS0tLS0tLQpNSU1Fb
    3S21cykEta31SEpDvhtdOUk1Tvr8c3NMVf1U1hYbY5Q9v3FHU1TMQ01SPS21Rc1Y2Hbhu
```

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl : commandes de base

Kubectl – commandes de base



Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Généralités sur les commandes

kubectl fonctionne avec cette syntaxe générale :

kubectl [command] [TYPE] [NAME] [flags]

- **command** : il s'agit de l'opération à exécuter : get, create, delete, ...
- **TYPE** : la ressource sur laquelle porte l'opération (pod, service, etc.)
- **NAME** : spécifie le nom de la ressource. Le nom est facultatif, si aucun nom est précisé alors la commande peut porter sur tous les objets, par exemple : kubectl get pods
- **Flags** : spécifie les options

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Gestion de la configuration

Pour gérer votre fichier de configuration, vous pouvez l'éditer manuellement ou vous pouvez le faire à travers kubectl.

- **kubectl config view** : permet de voir la configuration actuel
- **kubectl config set-cluster** <cluster-name> --server=<cluster-url> : définit un nouveau cluster
- **kubectl config set-context** <context-name> --cluster=<cluster-name> : permet de déclarer un nouveau contexte sur un cluster donné
- **kubectl config use-context** <context-name> : permet d'utiliser le contexte spécifié

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Vérification de kubectl

- **kubectl version** : permet de vérifier la version de kubectl

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Gestion des nœuds

- **kubectl get nodes** : Lister les nœuds du cluster
- **kubectl describe node <node-name>** : Afficher les détails d'un nœud du cluster

```
Name:          docker-desktop
Roles:         control-plane
Labels:        beta.kubernetes.io/arch=amd64
               beta.kubernetes.io/os=linux
               kubernetes.io/arch=amd64
               kubernetes.io/hostname=docker-desktop
               kubernetes.io/os=linux
               node-role.kubernetes.io/control-plane=
Annotations:   node.kubernetes.io/exclude-from-external-load-balancers=
               kubeadm.alpha.kubernetes.io/cri-socket: unix:///var/run/cni-dockerd.sock
               node.alpha.kubernetes.io/ttl: 0
               volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Sat, 09 Dec 2023 14:45:38 +0100
Taints:         <none>
Unschedulable:  false
Lease:
  HolderIdentity: docker-desktop
  AcquireTime:    <unset>
  RenewTime:      Sat, 09 Dec 2023 21:38:01 +0100
Conditions:
  Type           Status    LastHeartbeatTime           LastTransitionTime
  ----           -
MemoryPressure   False    Sat, 09 Dec 2023 21:37:48 +0100  Sat, 09 Dec 2023 14:45:37 +0100
DiskPressure     False    Sat, 09 Dec 2023 21:37:48 +0100  Sat, 09 Dec 2023 14:45:37 +0100
PIDPressure      False    Sat, 09 Dec 2023 21:37:48 +0100  Sat, 09 Dec 2023 14:45:37 +0100
Ready            True     Sat, 09 Dec 2023 21:37:48 +0100  Sat, 09 Dec 2023 14:45:38 +0100
Addresses:
  InternalIP: 192.168.65.3
  Hostname:   docker-desktop
Capacity:
  cpu: 12
```

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Gestion des pods

- **kubectl get pods** : Lister les pods du namespace courant
- **kubectl get pods --all-namespaces** : Lister les pods de tous les namespaces
- **kubectl run <pod-name> --image=<image-name>** : Lancer un pod à partir d'une image
- **kubectl exec -it <pod-name> -- /bin/bash** : Executer une commande à l'intérieur d'un pod
- **kubectl delete pod <pod-name>** : Supprimer un pod

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Gestion des services et des déploiements

- **kubectl get services** : Lister les services
- **kubectl expose deployment <deployment-name> --port=<port>** : Exposer un déploiement en tant que service
- **kubectl delete service <service-name>** : Supprimer un service
- **kubectl describe service <service-name>** : Afficher les détails d'un service
- **kubectl get deployments** : Lister les déploiements
- **kubectl scale deployment <deployment-name> --replicas=<num-replicas>** : Effectuer un scaling en ajoutant des réplicas
- **kubectl delete deployment <deployment-name>** : Supprimer un déploiement
- **kubectl set image deployment/<deployment-name> <container-name>=<new-image>** : Mettre à jour une image de déploiement

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Surveillance

- **kubectl get events** : Afficher les évènements du cluster
- **kubectl logs <pod-name>** : Vérifier les logs d'un pod
- **kubectl debug <pod-name> -it --image=<debugger-image>** : Debugger un pod

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Gestion des namespaces

- **kubectl get namespaces** : Affiche les namespaces disponibles
- **kubectl create namespace <NOM_NAMESPACE>** : Crée un nouveau namespace
- **kubectl config set-context --current --namespace=<NOM_NAMESPACE>** : Modifie le namespace par défaut
- **kubectl get pods --namespace=<NOM_NAMESPACE>** : Affiche les pods du namespace précisé (option -n possible également)

Gestion du contexte

- **kubectl config get-contexts** : Afficher la liste des contextes
- **kubectl config current-context** : Affiche le contexte courant
- **kubectl config set-context <NOM_CONTEXTE> --cluster=<NOM_DU_CLUSTER> --user=<UTILISATEUR> --namespace=<NOM_NAMESPACE>** : Crée un nouveau contexte
- **kubectl config use-context <NOM_CONTEXTE>** : Utilise un autre contexte
- **kubectl config rename-context <ANCIEN_NOM_CONTEXTE> <NOUVEAU_NOM_CONTEXTE>** : Renomme un contexte

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Exemple concret : lancer un pod nginx

- **kubectl run my-nginx --image=nginx --port=80** : ceci va créer un port nginx `pod/my-nginx created`
- **kubectl get pods** : Afficher les pods

NAME	READY	STATUS	RESTARTS	AGE
my-nginx	1/1	Running	0	19s

- **kubectl describe pod my-nginx** : Afficher les détails du pod

```
kubectl describe pod my-nginx
Name:          my-nginx
Namespace:     default
Priority:       0
Service Account: default
Node:          docker-desktop/192.168.65.3
Start Time:    Sat, 09 Dec 2023 21:52:17 +0100
Labels:        run=my-nginx
Annotations:   <none>
Status:        Running
IP:            10.1.0.8
IPs:
  IP: 10.1.0.8
Containers:
  my-nginx:
    Container ID:  docker://c4a1f1814895ba53189a0bc301
    Image:         nginx
    Image ID:      docker-pullable://nginx@sha256:10d1
    Port:         80/TCP
    Host Port:     0/TCP
    State:         Running
      Started:     Sat, 09 Dec 2023 21:52:25 +0100
    Ready:         True
    Restart Count: 0
    Environment:   <none>
```

Chapitre 4 : Introduction Kubernetes


Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

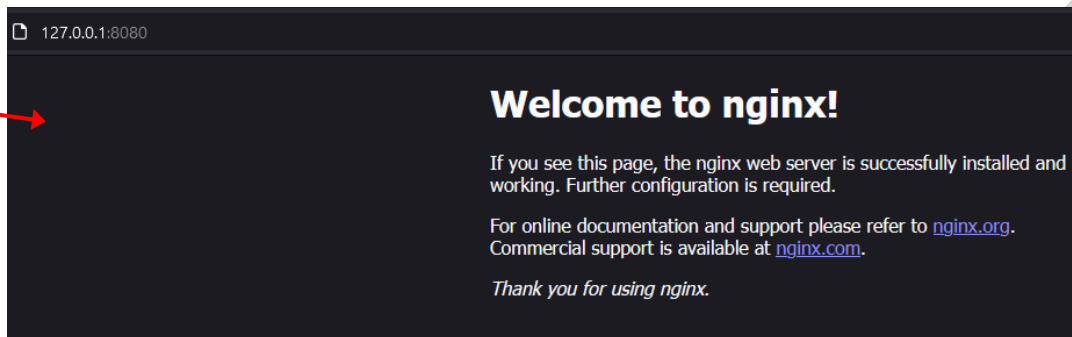
Exemple concret : lancer un pod nginx → port forwarding

- Sur le pod précédent, vous n'arrivez pas à vous connecter depuis votre navigateur (qui est en dehors du cluster), car le port est exposé mais il n'est pas lié avec le host.
- En réalisant un port forwarding, nous pourrions alors nous connecter à ce pod :
- **kubectl port-forward my-nginx 8080:80**

```
Forwarding from 127.0.0.1:8080 -> 80  
Forwarding from [::1]:8080 -> 80  
Handling connection for 8080
```



- **kubectl port-forward my-nginx 8080:80 &**
 - Lancement en arrière-plan



Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Exemple concret : lancer un pod nginx avec des replicas

Pour lancer des réplicas et donc réaliser une mise à l'échelle, il faut utiliser la notion de déploiement dans Kubernetes

- **kubectl create deployment my-nginxsss --image nginx --replicas 6** : création d'un déploiement avec 6 replicas
- **kubectl get pods** : Afficher les pods (on observe les 6 replicas)

NAME	READY	STATUS	RESTARTS	AGE
my-nginxsss-c874cbf47-gqws7	1/1	Running	0	39s
my-nginxsss-c874cbf47-h5rh1	1/1	Running	0	39s
my-nginxsss-c874cbf47-j7nb4	1/1	Running	0	39s
my-nginxsss-c874cbf47-kdjxx	1/1	Running	0	39s
my-nginxsss-c874cbf47-l9b9q	1/1	Running	0	39s
my-nginxsss-c874cbf47-sjcps	1/1	Running	0	39s

- **kubectl describe deployment my-nginxsss** : Afficher les détails du déploiement
- **kubectl delete deployment my-nginxsss** : Supprimer ce déploiement

Chapitre 4 : Introduction Kubernetes


Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Exemple concret : lancer un service nginx de type NodePort

Dans les exemples précédents, nous avons manipuler directement des pods (utile pour comprendre et tester), la manière adaptée pour exposer une application est d'utiliser un service :

- **kubectl create deployment nginx-deployment --image=nginx**
- **kubectl expose deployment nginx-deployment --port=80 --type=NodePort --name=nginx-service**
- **Kubectl get services**



NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	2d
nginx-service	NodePort	10.102.92.152	<none>	80:31191/TCP	17s

- On observe dans cet exemple que le service nginx-service a été crée avec comme port 31191

127.0.0.1:31191

Welcome to nginx!

If you see this page, the nginx web server is successfully running and working. Further configuration is required.

For online documentation and support please refer to the nginx documentation. Commercial support is available at nginx.com.

Thank you for using nginx.

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl – commandes de base

Kubectl – commandes de base

Mais où se trouve les images ?

Nous lançons des commandes via kubectl qui récupère et utilise des images mais d'où viennent-elles ?

N'oubliez pas que Kubernetes utilise un moteur de conteneurisation, dans notre cas il utilise Docker et donc les images proviennent du repository officiel de Docker !

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Kubectl : commandes de base

Kubectl – Manipulation (TP)



Chapitre 4 : Introduction Kubernetes

Module 4.2 – Les replicaset

Manipulation avec Kubectl (TP)

Suivre le TP fournit par le formateur.

Chapitre 4 : Introduction Kubernetes

Module 4.3 – Questions ?

