

Formation Kubernetes avancée :

Introduction

Objectif opérationnel :

Au terme de ce cours, les participants auront une connaissance pratique de l'orchestration des conteneurs en production.

Objectifs pédagogiques :

Concrètement, à l'issue de cette formation les participants seront capables de :

- Savoir gérer les problématiques de sécurité des conteneurs, connaître les bonnes pratiques à adopter
- Les fonctions avancées de la construction d'image avec Dockerfile, Arguments, variables, sondes...
- Développer et déployer des applications avancées multi-containers avec Docker Compose
- Concepts d'orchestration de conteneurs
- Découverte de Swarm
- Comprendre le concept d'applications «Orchestrator-ready»
- Comprendre l'architecture de Kubernetes : Les différents types et rôles des nœuds
- Notion de pods, service, stockage et déploiements
- Gérer le cycle de ses déploiements
- Gérer les mises à jour des applications
- Déploiement et partage des éléments de configuration
- Comprendre les applications Stateless et Stateful
- Comprendre les outils de l'écosystème Kubernetes
- Déployer des applications complexes avec le manager Helm
- Gérer le réseau avec le loadbalancer et les ingress ...
- L'importance des services mesh avec Istio
- L'importance des services des solutions de stockage
- L'importance du monitoring de l'infrastructurelle et des applications

Le cours est découpé en plusieurs parties. Celles-ci pourront être réorganisées de manière différente en fonction du déroulement de la formation.

Sommaire

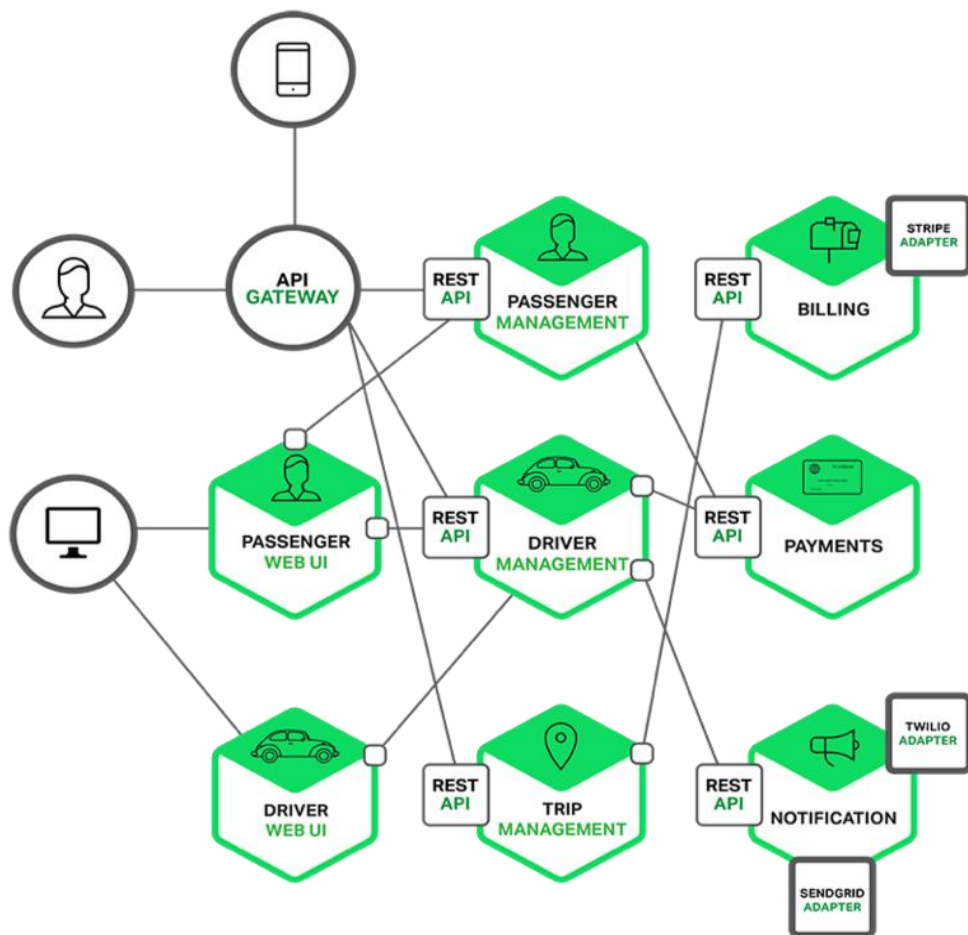
Comprendre les applications microservices

Les projets de l'écosystème Kubernetes

Comprendre les applications microservices

Les applications sont généralement créées de manière monolithique. Autrement dit, toutes les fonctionnalités de l'application qui peuvent être déployés résident dans cette seule application. L'inconvénient, c'est que plus celle-ci est volumineuse, plus il devient difficile de l'enrichir de fonctions et de traiter rapidement les problèmes qui surviennent.

Avec une approche basée sur des microservices, il est possible de résoudre ces problèmes, d'améliorer le développement et de gagner en réactivité.

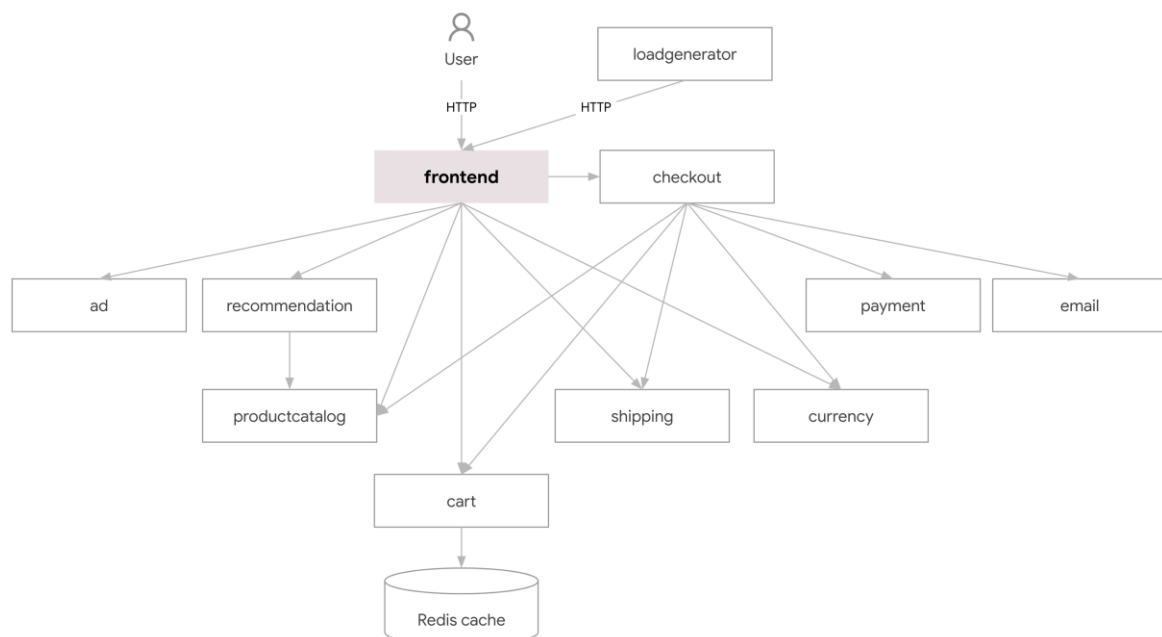


Les microservices, qu'est-ce que c'est ?

Les microservices désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Une fonctionnalité par service.

Contrairement à une approche monolithique classique, selon laquelle tous les composants forment une entité indissociable, les microservices fonctionnent en synergie pour accomplir les mêmes tâches, tout en étant séparés.

Chacun de ces composants ou processus est un microservice. Granulaire et léger, ce type de développement logiciel permet d'utiliser un processus similaire dans plusieurs applications. Il s'agit d'un élément essentiel pour optimiser le développement des applications en vue de l'adoption d'un modèle cloud-native.



Mais quel est l'intérêt d'une infrastructure basée sur des microservices ?

L'objectif, qui consiste tout simplement à proposer des logiciels de qualité en un temps record, devient atteignable grâce aux microservices. Pour autant, d'autres éléments entrent également en ligne de compte. La décomposition des applications en microservices ne suffit pas. Il faut aussi gérer ces microservices, les orchestrer et traiter les données qui sont générées et modifiées par les microservices.

Quels sont les avantages des microservices ?

Par rapport aux applications monolithiques, les microservices sont beaucoup plus faciles à créer, tester, déployer et mettre à jour et ainsi éviter un processus de développement interminable sur plusieurs années. Aujourd'hui, les différentes tâches de développement peuvent être réalisées simultanément et de façon agile pour apporter immédiatement de la valeur aux clients.

Les microservices ont-ils un rapport avec les conteneurs Linux?

Avec les conteneurs Linux, vos applications basées sur des microservices disposent d'une unité de déploiement et d'un environnement d'exécution parfaitement adaptés. Lorsque les microservices sont stockés dans des conteneurs, il est plus simple de tirer parti du matériel et d'orchestrer les services, notamment les services de stockage, de réseau et de sécurité.

C'est pour cette raison que la Cloud Native Computing Foundation affirme qu'ensemble, les microservices et les conteneurs constituent la base du développement d'applications cloud-native.

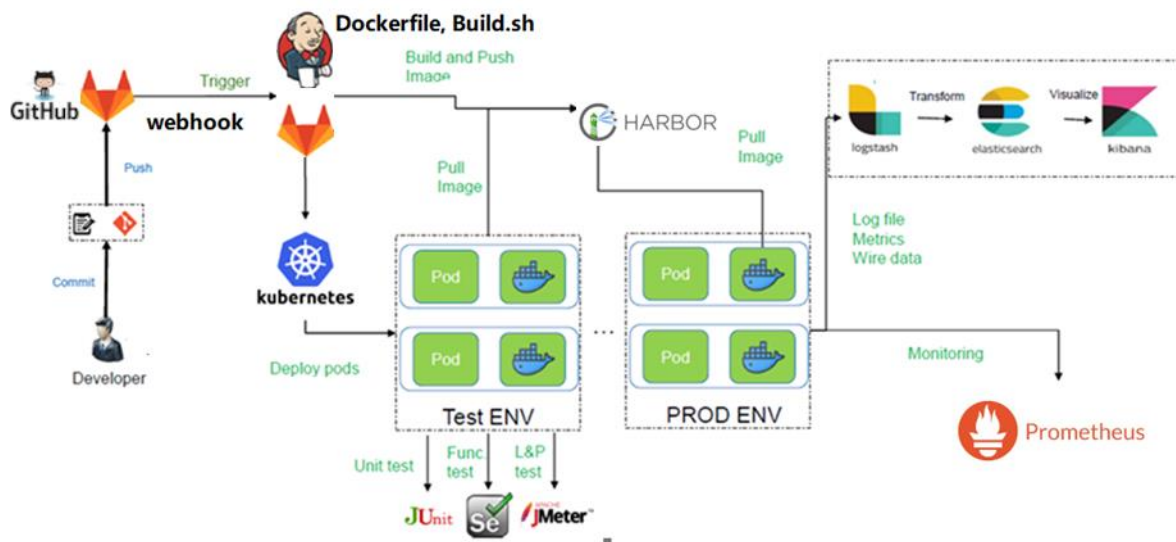
Ce modèle permet d'accélérer le processus de développement et facilite la transformation ainsi que l'optimisation des applications existantes, en commençant par le stockage des microservices dans des conteneurs.

Quel est l'impact des microservices sur l'intégration des applications ?

Pour qu'une architecture de microservices s'exécute comme une application cloud fonctionnelle, les services doivent en permanence demander des données à d'autres services via un système de messagerie.

Si la création d'une couche de Service Mesh (réseau maillé) directement dans l'application facilite la communication entre les services, l'architecture de microservices doit également intégrer les applications existantes et les autres sources de données. Base de données As Service, Monitoring As Service, etc ... L'intégration agile est une stratégie de connexion des ressources qui allie des technologies d'intégration, des techniques de distribution agile et des plateformes cloud-native dans le but d'accélérer la distribution des logiciels tout en renforçant la sécurité.

DevOps (Jenkins, Ansible, etc...)



Un pipeline DevOps est un ensemble d'étapes automatisées qui permettent d'assurer une intégration et un déploiement continu (CI/CD) de logiciels. Le pipeline est conçu pour permettre aux équipes de développement de livrer rapidement des logiciels de qualité supérieure tout en garantissant une expérience utilisateur optimale.

Voici les principales étapes d'un pipeline DevOps typique :

- **Intégration** : les modifications de code sont intégrées dans un référentiel centralisé (par exemple, Git) qui sert de source unique de vérité pour l'ensemble du projet.
- **Compilation** : le code source est compilé en un exécutable ou un package.
- **Test** : le code est soumis à une série de tests automatisés pour garantir qu'il fonctionne comme prévu et qu'il répond aux normes de qualité.
- **Intégration continue** : les modifications de code sont automatiquement intégrées dans le référentiel principal et testées à chaque modification de code.
- **Livraison continue** : le code compilé et testé est automatiquement livré à un environnement de développement, de test ou de production.
- **Déploiement continu** : le code livré est automatiquement déployé dans l'environnement de production.
- **Surveillance** : l'application est surveillée pour détecter les erreurs, les pannes et les problèmes de performance. Les rapports de surveillance sont utilisés pour améliorer la qualité du code et des processus DevOps.

L'automatisation de ces étapes garantit que les changements de code sont rapidement intégrés, testés et livrés, ce qui permet aux équipes de développement de se concentrer sur la création de fonctionnalités et d'améliorations, plutôt que de passer du temps à configurer et à déployer des logiciels manuellement. Le pipeline

DevOps permet également de réduire les erreurs humaines et d'améliorer la qualité du code grâce à une surveillance constante et à des tests automatisés.

Il existe une grande variété d'outils qui peuvent être utilisés pour mettre en place un pipeline DevOps. Le choix des outils dépendra des besoins spécifiques de l'équipe de développement et de l'entreprise, ainsi que des langages de programmation et des plates-formes utilisés.

Voici quelques exemples d'outils couramment utilisés pour mettre en place un pipeline DevOps :

- Outils de gestion de code source : Git, Bitbucket, SVN, Gitlab.
- Outils de compilation et de construction : Jenkins, Gitlab, Tekton, etc.
- Outils de tests : JUnit, NUnit, Selenium, etc.
- Outils de déploiement : Ansible, Puppet, ArgoCD, Jenkins.
- Outils de conteneurisation : Docker, Kubernetes, Openshift, etc.
- Outils de journalisation et de surveillance et: Nagios, ELK Stack (Elasticsearch, Logstash, Kibana), Prometheus, Grafana, etc.
- Outils de collaboration et de communication : Slack, Microsoft Teams, etc.

Il est important de noter que la liste des outils ci-dessus n'est pas exhaustive, et que de nouveaux outils sont régulièrement développés pour aider les équipes de développement à améliorer leur pipeline DevOps.

Trois transformations profondes de l'informatique

Kubernetes se trouve au cœur de trois transformations profondes techniques, humaines et économiques de l'informatique :

- Le cloud
- La conteneurisation logicielle
- Le mouvement DevOps

Il est un des projets qui symbolise et supporte techniquement ces transformations. D'où son omniprésence dans les discussions informatiques actuellement.

Le Cloud

- Au-delà du flou dans l'emploi de ce terme, le cloud est un mouvement de réorganisation technique et économique de l'informatique.
- On retourne à la consommation de "temps de calcul" et de services après une "aire du Personnel Computer".
- Pour organiser cela on définit trois niveaux à la fois techniques et économiques de l'informatique :

- Software as a Service : location de services à travers internet pour les usagers finaux
- Platform as a Service : location d'un environnement d'exécution logiciel flexible à destination des développeurs
- Infrastructure as a Service : location de ressources "matérielles" à la demande pour des VMs et de conteneurs sans avoir à maintenir un data center.

Conteneurisation

La conteneurisation est permise par l'isolation au niveau du noyau du système d'exploitation du serveur : les processus sont isolés dans des namespaces au niveau du noyau. Cette innovation permet de simuler l'isolation sans ajouter une couche de virtualisation comme pour les machines virtuelles.

Ainsi les conteneurs permettent d'avoir des performances d'une application traditionnelle tournant directement sur le système d'exploitation hôte et ainsi d'optimiser les ressources.

Les technologies de conteneurisation permettent donc d'exécuter des applications dans des « boîtes » isolées, ceci pour apporter l'uniformisation du déploiement :

- Une façon standard de packager un logiciel (basée sur l'image)
- Cela réduit la complexité grâce:
 - À l'intégration de toutes les dépendance déjà dans la boîte
 - Au principe d'immutabilité qui implique de jeter les boîtes (automatiser pour lutter contre la culture de prudence). Rend l'infra prédictible.

Le mouvement DevOps

- Dépasser l'opposition culturelle et de métier entre les développeurs et les administrateurs système.
- Intégrer tout le monde dans une seule équipe et ...
- Calquer les rythmes de travail sur l'organisation agile du développement logiciel
- Rapprocher techniquement la gestion de l'infrastructure du développement avec l'infrastructure as code.
 - Concrètement on écrit des fichiers de code pour gérer les éléments d'infra
 - L'état de l'infrastructure est plus clair et documenté par le code
 - La complexité est plus gérable car tout est déclaré et modifiable au fur et à mesure de façon centralisée

- L'usage de git et des branches/tags pour la gestion de l'évolution d'infrastructure

Objectifs du DevOps

- Rapidité (velocity) de déploiement logiciel (organisation agile du développement et livraison jusqu'à plusieurs fois par jour)
 - Implique l'automatisation du déploiement et ce qu'on appelle la CI/CD c'est à dire une infrastructure de déploiement continu à partir de code.
- Passage à l'échelle (horizontal scaling) des logiciels et des équipes de développement (nécessaire pour les entreprises du cloud qui doivent servir pleins d'utilisateurs)
- Meilleure organisation des équipes
 - Meilleure compréhension globale du logiciel et de son installation de production car le savoir est mieux partagé
 - Organisation des équipes par thématique métier plutôt que par spécialité technique (l'équipe scale mieux)

Apports techniques de Kubernetes pour le DevOps

- Abstraction et standardisation des infrastructures :
- Langage descriptif et incrémental : on décrit ce qu'on veut plutôt que la logique complexe pour l'atteindre
- Logique opérationnelle intégrée dans l'orchestrateur : la responsabilité de l'état du cluster est laissée au contrôleur k8s ce qui simplifie le travail

Architecture logicielle optimale pour Kubernetes

Kubernetes est très versatile et permet d'installer des logiciels traditionnels "monolithiques" (gros backends situés sur une seule machine).

Cependant aux vues des transformations humaines et techniques précédentes, l'organisation de Kubernetes prend vraiment sens pour le développement d'applications microservices ou Cloud ready :

- Des applications avec de nombreux de "petits" services.
- Chaque service a des problématiques très limitées (gestion des utilisateurs = une application qui ne fait que ça)
- Les services communiquent par les réseaux selon différents modes API REST, gRPC, job queues, GraphQL)

Les microservices permettent justement le DevOps car :

- Ils peuvent être déployés séparément

Une petite équipe gère chaque service ou groupe thématique de services

Connexion sur le serveur de formation

Se connecter sur le serveur de formation

Serveur : formation.ludovic.io

Port ssh : 3460

login : nom.prénom (SANS ACCENT , NI MAJUSCULE)

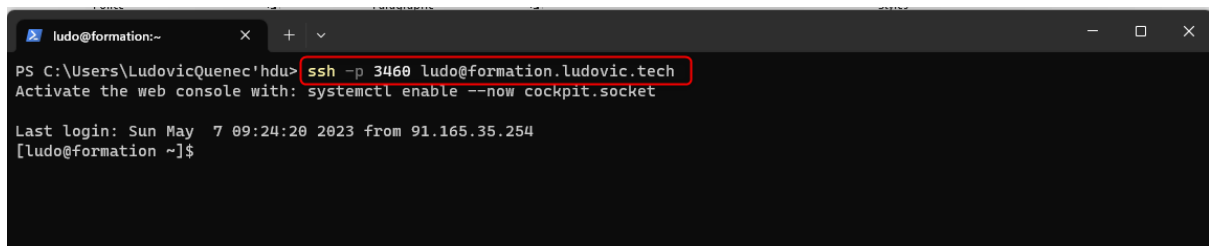
password: Thales@2023

Exemple :

Login : bonin.dylan

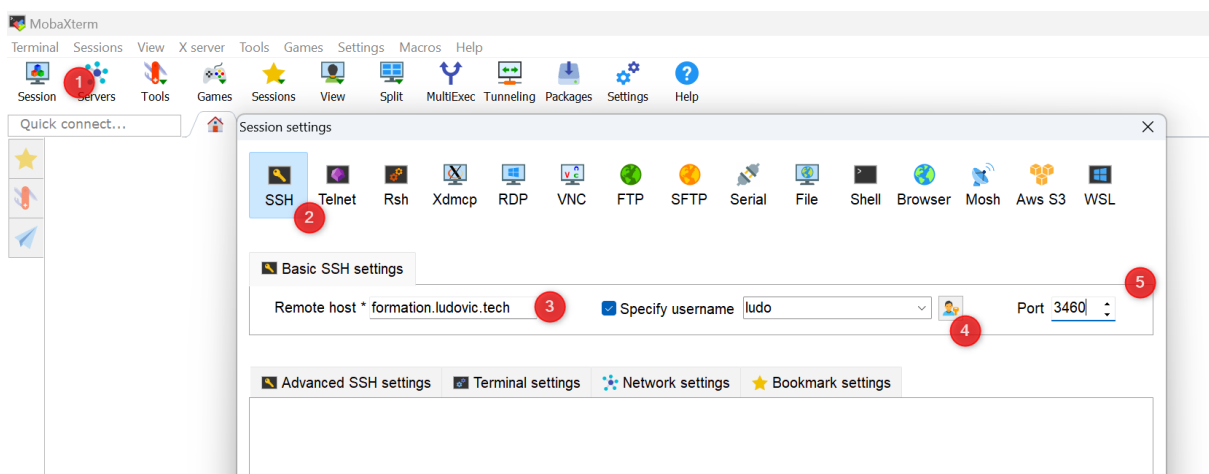
password: Thales@2023

Avec un terminal Windows 11



```
ludo@formation:~  
PS C:\Users\LudovicQuenec\hdu> ssh -p 3460 ludo@formation.ludovic.tech  
Activate the web console with: systemctl enable --now cockpit.socket  
  
Last login: Sun May 7 09:24:20 2023 from 91.165.35.254  
[ludo@formation ~]$
```

Avec MobaXterm



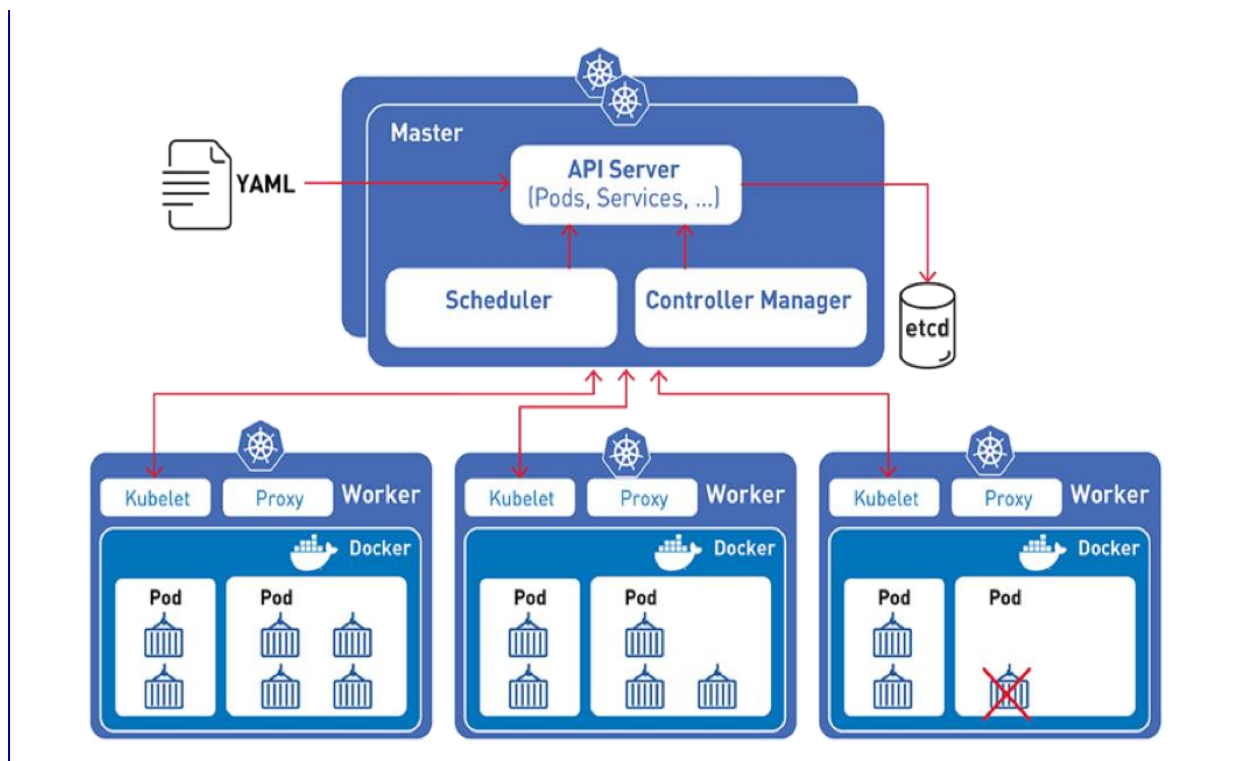
Dans son repertoire k8s

```
[ludo@formation:~$]ls k8s/  
apache-php      daemonSet      documents      hpa            liveness-  
probe namespace . . . .
```

Présentation de Kubernetes

- Kubernetes est une solution d'orchestration de conteneurs extrêmement populaire.
- Le projet est très ambitieux : une façon de considérer son ampleur est de voir Kubernetes comme un système d'exploitation (et un standard ouvert) pour les applications distribuées et le cloud.
- Le projet est développé en Open Source au sein de la Cloud Native Computing Foundation.

Concrètement : Architecture de Kubernetes



- Kubernetes rassemble en un cluster et fait coopérer un groupe de serveurs appelés nœuds(nodes).
- Kubernetes a une architecture Master/workers (cf. cours 2) composée d'un control plane et de nœuds de calculs (workers).
- Cette architecture permet essentiellement de rassembler les machines en un cluster unique sur lequel on peut faire tourner des "charges de calcul" (workloads) très diverses.

- Sur un tel cluster le déploiement d'un workload prend la forme de ressources (objets k8s) qu'on décrit sous forme de code et qu'on crée ensuite effectivement via l'API Kubernetes.
- Pour uniformiser les déploiement logiciel Kubernetes est basé sur le standard des conteneurs (défini aujourd'hui sous le nom Container Runtime Interface, Docker est l'implémentation la plus connue).
- Plutôt que de déployer directement des conteneurs, Kubernetes crée des agrégats de un ou plusieurs conteneurs appelés des Pods. Les pods sont donc l'unité de base de Kubernetes.

Philosophie derrière Kubernetes et le mouvement "Cloud Native"

Historique et popularité



Kubernetes est un logiciel développé originellement par Google et basé sur une dizaine d'années d'expérience de déploiement d'applications énormes (distribuées) sur des clusters de machines.

Dans la mythologie Cloud Native on raconte que son ancêtre est l'orchestrateur borg utilisé par Google dans les années 2000.

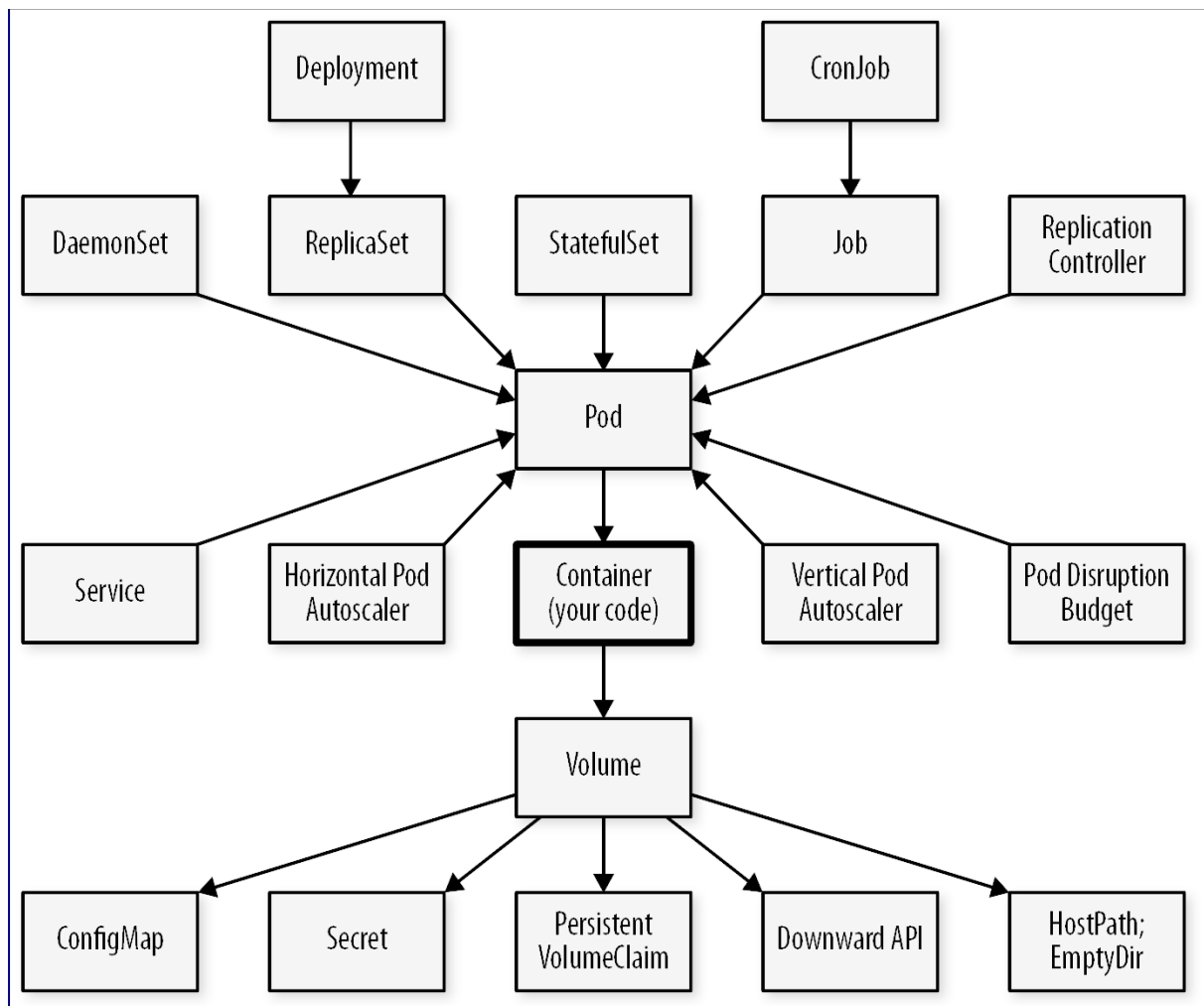
La première version est sortie en 2015 et k8s est devenu depuis l'un des projets open source les plus populaires du monde.

L'écosystème logiciel de Kubernetes s'est développé autour la Cloud Native Computing Foundation qui comprend des acteurs comme notamment : Google, Red Hat, Twitter, Huawei, Intel, Cisco, IBM, VMware, Amazon, Microsoft, etc.. . Cette fondation vise au pilotage et au financement collaboratif du développement de Kubernetes et de bien d'autre projets liés aux applications Cloud ready, Cloud native

Objets fondamentaux de Kubernetes

- Les pods Kubernetes servent à grouper des conteneurs fortement couplés en unités d'application. La fonctionnalité
- Les deployments sont une abstraction pour créer ou mettre à jour (ex : scaler) des groupes de pods. Via des ReplicaSet
- Les services sont des points d'accès réseau qui permettent aux différents workloads (deployments) de communiquer entre eux et avec l'extérieur.
- Les PersistentVolumes et les PersistentVolumeClaims, sont une abstraction d'espace de stockage (nfs, cephfs, cloud, etc...)

Au-delà de ces éléments, l'écosystème d'objets de Kubernetes est vaste et complexe



2 - Mettre en place un cluster

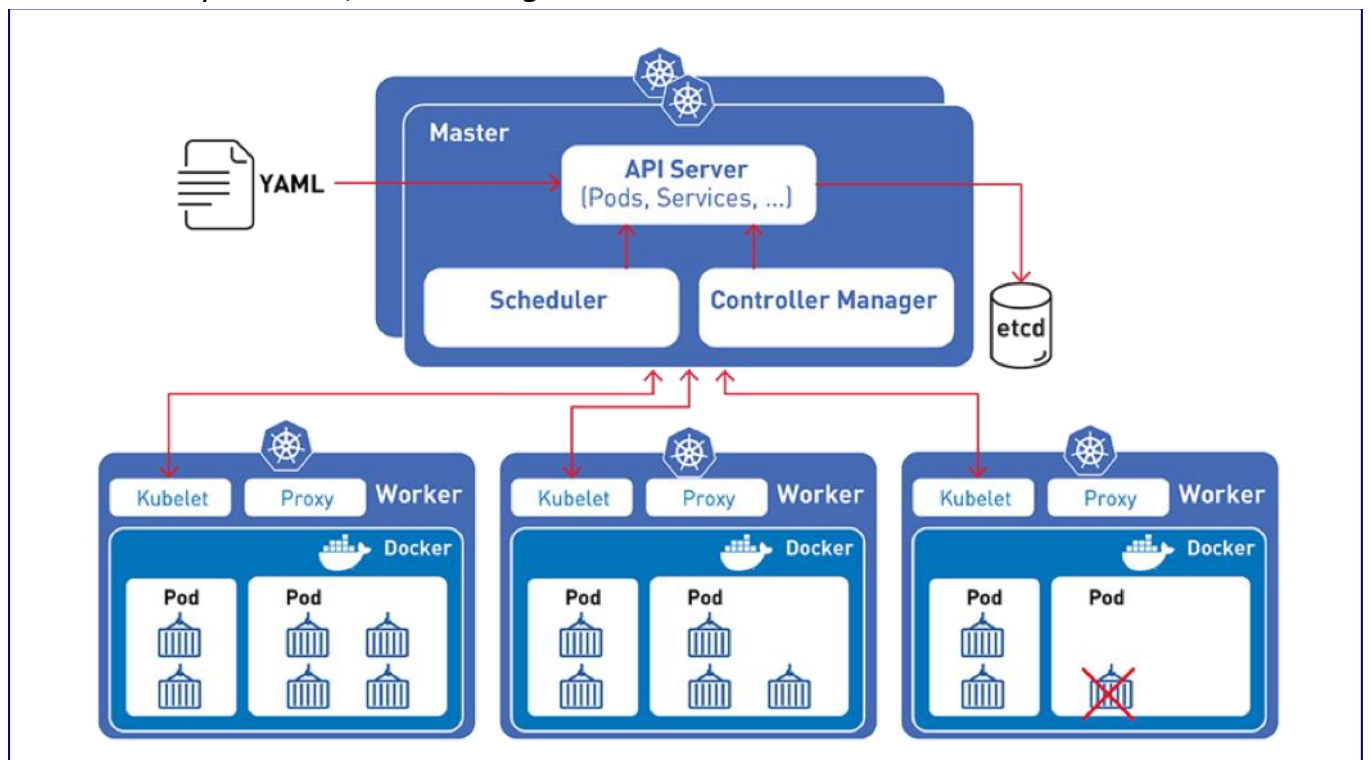
Architecture de Kubernetes - Partie 1

Kubernetes master

- Le master est responsable du maintien de l'état souhaité pour votre cluster. Lorsque vous interagissez avec Kubernetes, par exemple en utilisant l'interface en ligne de commande kubectl, vous communiquez avec le master Kubernetes de votre cluster.
- Le "master" fait référence à un ensemble de processus gérant l'état du cluster. Le master peut également être répliqué pour la disponibilité et la redondance.

Noeuds Kubernetes

Les nœuds d'un cluster sont les machines (serveurs physiques, machines virtuelles, etc.) qui exécutent vos applications et vos workflows. Le master node Kubernetes contrôle chaque nœud; vous interagirez rarement directement avec les nœuds.



- Pour utiliser Kubernetes, vous utilisez les objets de l'API Kubernetes pour décrire l'état souhaité de votre cluster: quelles applications ou autres processus que vous souhaitez exécuter, quelles images de conteneur elles

utilisent, le nombre de réplicas, les ressources réseau et disque que vous mettez à disposition, et plus encore.

- Vous définissez l'état souhaité en créant des objets à l'aide de l'API Kubernetes, généralement via l'interface en ligne de commande, `kubectl`. Vous pouvez également utiliser l'API Kubernetes directement pour interagir avec le cluster et définir ou modifier l'état souhaité.
- Une fois que vous avez défini l'état souhaité, le plan de contrôle Kubernetes (control plane) permet de faire en sorte que l'état actuel du cluster corresponde à l'état souhaité. Pour ce faire, Kubernetes effectue automatiquement diverses tâches, telles que le démarrage ou le redémarrage de conteneurs, la mise à jour du nombre de *replicas* d'une application donnée, etc.

Le Kubernetes Control Plane

- Le control plane Kubernetes comprend un ensemble de processus en cours d'exécution sur votre cluster:
 - Le master Kubernetes est un ensemble de trois processus qui s'exécutent sur un seul nœud de votre cluster, désigné comme nœud maître (*master node* en anglais). Ces processus sont:
 - `kube-apiserver`: expose l'API pour parler au cluster
 - `kube-controller-manager`: basé sur une boucle qui contrôle en permanence l'état des ressources et essaie de le corriger s'il n'est plus conforme.
 - `kube-scheduler`: monitore les ressources des différents workers, décide et cartographie ou doivent être ordonnancer les Workloads les conteneur(Pods)
 - Chaque nœud (master et worker) du cluster exécute deux processus : `kubelet`, qui communique avec le master et contrôle la création et l'état des pods sur son nœud. `kube-proxy`, un proxy réseau reflétant les services réseau Kubernetes sur chaque nœud.

Les différentes parties du control plane Kubernetes, telles que les processus `kube-controller-manager` et `kubelet`, déterminent la manière dont Kubernetes communique avec votre cluster.

Le control plane conserve un enregistrement de tous les objets Kubernetes du système et exécute des boucles de contrôle continues pour gérer l'état de ces objets. À tout moment, des boucles de contrôle du control plane répondent aux modifications du cluster et permettent de faire en sorte que l'état réel de tous les objets du système corresponde à l'état souhaité que vous avez fourni.

Par exemple, lorsque l'on utilise l'API Kubernetes pour créer un objet `Déploiement`, vous fournissez un nouvel état souhaité pour le système. Le control plane Kubernetes enregistre la création de cet objet et exécute vos instructions en lançant les applications requises et en les planifiant vers des nœuds de cluster, afin que l'état actuel du cluster corresponde à l'état souhaité.

Discuter avec Kubernetes avec kubectl

...Permet depuis sa machine de travail de contrôler le cluster avec une ligne de commande qui ressemble un peu à celle de Docker (cf. TP1 et TP2):

- Lister les ressources
- Créer et supprimer les ressources
- Gérer les droits d'accès
- etc.
- Ces informations sont fournies sous forme d'un fichier YAML appelé `kubeconfig`
- Comme nous le verrons en TP ces informations sont généralement fournies directement par le fournisseur d'un cluster k8s (provider ou k8s de dev)

Se connecter sur son serveur personnel Docker

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ...
    server: https://10.0.0.9:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    namespace: default
    user: kubernetes-admin
    name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: ==
    client-key-data:
```

Le fichier `kubeconfig` peut stocker plusieurs configurations dans un fichier YAML :

Déployer un orchestrateur Kubernetes

Pour installer un cluster de développement :

- Kind : kind est un outil permettant d'exécuter des clusters Kubernetes locaux en utilisant des conteneurs Docker "nodes". kind a été principalement conçu pour tester Kubernetes lui-même, mais peut être utilisé pour le développement local ou le CI. <https://kind.sigs.k8s.io/>
- Un cluster avec k3s, de Rancher (simple et utilisable en production/edge). <https://k3s.io/>

Commander un cluster en tant que service (*managed cluster*) dans le cloud

Tous les principaux providers de cloud fournissent depuis plus ou moins longtemps des solutions de cluster gérées par eux :

- Google Cloud Platform avec Google Kubernetes Engine (GKE) : très populaire car très flexible et l'implémentation de référence de Kubernetes.
- AWS avec EKS : Kubernetes assez standard mais à la sauce Amazon pour la gestion de l'accès, des loadbalancers ou du scaling.
- Azure avec AKS : Kubernetes assez standard mais à la sauce Amazon pour la gestion de l'accès, des loadbalancers ou du scaling.
- DigitalOcean ou Scaleway : un peu moins de fonctions mais plus simple à appréhender

Installer un cluster de production on premise : l'outil officiel kubeadm

kubeadm permet de créer un cluster Kubernetes, viable et conforme aux meilleures pratiques. Avec kubeadm, votre cluster doit passer les [tests de Conformité Kubernetes](#). Kubeadm prend également en charge d'autres fonctions du cycle de vie, telles que les mises à niveau, la rétrogradation et la gestion des [bootstrap tokens](#).

Comme vous pouvez installer kubeadm sur différents types de machines (par exemple, un ordinateur portable, un serveur, Raspberry Pi, etc.), il est parfaitement adapté à l'intégration avec des systèmes d'approvisionnement comme Terraform ou Ansible.

Pré-requis au déploiements via kubeadm

- Installer le démon `Kubelet` sur tous les noeuds
- Installer l'outil de gestion de cluster `kubeadm`
- Initialisé un cluster avec `kubeadm`
- Installer un réseau CNI k8s comme `flannel` (d'autres sont possible et le choix vous revient)
- Joindre les nœuds worker au cluster.

L'installation est décrite dans la [documentation officielle](#)

Opérer et maintenir un cluster de production Kubernetes "à la main" est très complexe et une tâche à ne pas prendre à la légère. De nombreux éléments doivent être installés et géré par les opérateurs.

- Mise à jour et passage de version de kubernetes qui doit être fait très régulièrement car une version n'est supportée que 2 ans.
- Choix d'une configuration réseau et de sécurité adaptée.
- Installation probable de système de stockage distribué comme Ceph à maintenir également dans le temps
- Etc.

3 – Découverte de Kubernetes

Installer le client CLI kubectl

kubectl est la commande pour administrer tous les type de ressources de kubernetes. C'est un client en ligne de commande qui communique en REST avec l'API d'un cluster.

Nous allons explorer kubectl au fur et à mesure des TPs. Cependant à noter que :

- kubectl peut gérer plusieurs clusters/configurations et switcher entre ces configurations

Afficher la version du client kubectl.

```
ludo@formation~$ kubectl version -short
Client Version: v1.25.0
Kustomize Version: v4.5.7
Server Version: v1.26.1
```

Explorons notre cluster k8s

Notre cluster k8s est plein d'objets divers, organisés entre eux de façon dynamique pour décrire nos applications.

Listez les noeuds (`kubectl get nodes`) puis affichez une description détaillée avec `kubectl describe node`

```
[root@formation ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	control-plane	90d	v1.26.1
worker00	Ready	<none>	90d	v1.26.1
worker01	Ready	<none>	90d	v1.26.1
worker02	Ready	<none>	90d	v1.26.1
worker03	Ready	<none>	90d	v1.26.1

La commande `get` est générique et peut être utilisée pour récupérer la liste de tous les types de ressources.

De même, la commande `describe` peut s'appliquer à tout objet k8s. On doit cependant préfixer le nom de l'objet par son type (ex : `node/minikube` ou `nodes minikube`) car k8s ne peut pas deviner ce que l'on cherche quand plusieurs ressources ont le même nom.

- Pour afficher tous les types de ressources à la fois que l'on utilise :

```
kubectl get all
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	2m34s

Il semble qu'il n'y a qu'une ressource dans notre cluster. Il s'agit du service d'API Kubernetes, pour qu'on puisse communiquer avec le cluster.

En réalité il y en a généralement d'autres cachés dans les autres `namespaces`. En effet les éléments internes de Kubernetes tournent eux-mêmes sous forme de services et de daemons Kubernetes. Les *namespaces* sont des groupes qui servent à isoler les ressources de façon logique et en termes de droits (avec le *Role-Based Access Control* (RBAC) de Kubernetes).

Pour vérifier cela on peut :

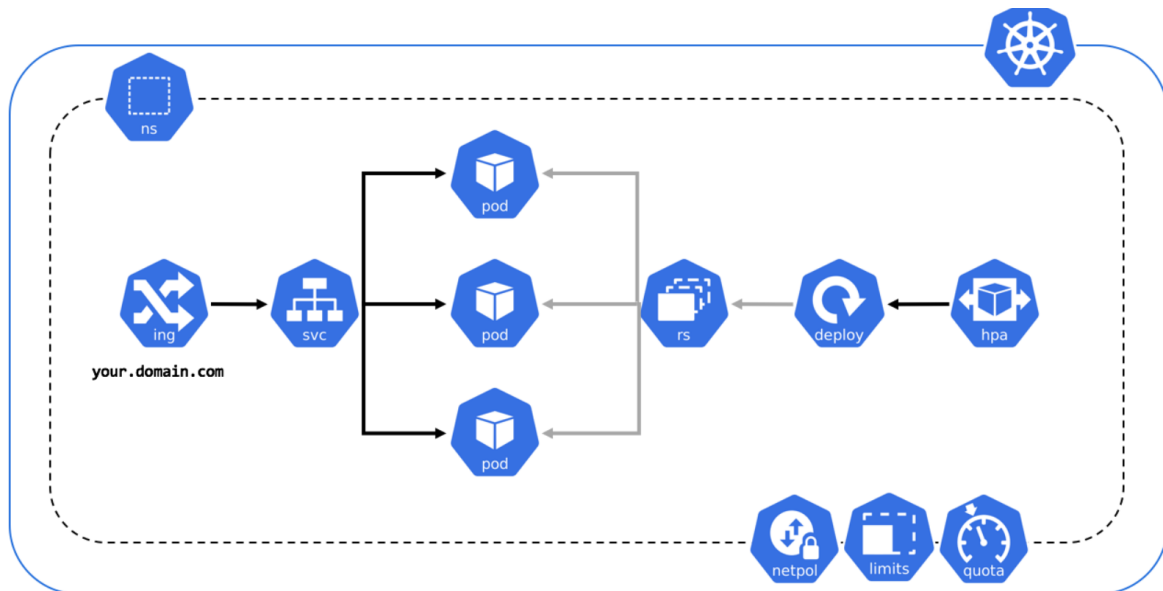
- Afficher les namespaces : `kubectl get namespaces`

Un cluster Kubernetes a généralement un namespace appelé `default` dans lequel les commandes sont lancées et les ressources créées si on ne précise rien. Il a également aussi un namespace `kube-system` dans lequel résident les processus et ressources système de k8s. Pour préciser le namespace on peut rajouter l'argument `-n` à la plupart des commandes k8s.

- Pour lister les ressources liées au `kubectl get all -n kube-system`
- Ou encore : `kubectl get all --all-namespaces` qui permet d'afficher le contenu de tous les namespaces en même temps.
- Pour avoir des informations sur un namespace :

`kubectl describe namespace/kube-system`

Déployer une application en CLI



Nous allons déployer une première application conteneurisée. Le déploiement est un peu plus complexe qu'avec Docker, en particulier car il est séparé en plusieurs objets et plus configurable.

- Pour créer un déploiement en ligne de commande (par opposition au mode déclaratif que nous verrons plus loin), on peut exécuter par exemple:

```
[root@formation ~]# kubectl create deployment --image=nginx nginx
deployment.apps/nginx created

[root@formation ~]# kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-748c667d99-mh5jr	1/1	Running	0	7s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	37s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nginx	1/1	1	1	7s

NAME	DESIRED	CURRENT	READY	AGE
• replicaset.apps/nginx-748c667d99	1	1	1	7s

Cette commande crée un objet de type `deployment`. Nous pouvons afficher les détails ce deployment avec la commande

```
[root@formation ~]# kubectl describe deployments.apps nginx
Name:          nginx
Namespace:     default
CreationTimestamp: Thu, 11 May 2023 18:41:55 +0200
Labels:        app=nginx
Annotations:   deployment.kubernetes.io/revision: 1
Selector:      app=nginx
Replicas:      1 desired | 1 updated | 1 total | 1 available | 0
unavailable
StrategyType:  RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
    nginx:
      Image:      nginx
```

Notez la liste des événements sur ce déploiement en bas de la description.
Mettons à l'échelle ce déploiement :

```
[root@formation ~]# kubectl scale deployments.apps nginx --replicas 6
deployment.apps/nginx scaled

[root@formation ~]# kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-748c667d99-hs8tt	1/1	Running	0	17s
pod/nginx-748c667d99-jn5kq	1/1	Running	0	17s
pod/nginx-748c667d99-jtxzp	1/1	Running	0	17s
pod/nginx-748c667d99-mh5jr	1/1	Running	0	4m18s
pod/nginx-748c667d99-tqkwc	1/1	Running	0	17s
pod/nginx-748c667d99-v9cpf	1/1	Running	0	17s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	4m48s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nginx	6/6	6	6	4m18s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/nginx-748c667d99	6	6	6	4m18s

Observez à nouveau la liste des évènements, le scaling y est enregistré...

A ce stade impossible d'afficher l'application : le déploiement n'est pas encore accessible de l'extérieur du cluster. Pour régler cela nous peut l'exposer avec l'aide d'un service :

```
[root@formation ~]# kubectl expose deployment nginx --type NodePort --port 80
service/nginx exposed
```

```
[root@formation ~]# kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-748c667d99-hs8tt	1/1	Running	0	2m33s
pod/nginx-748c667d99-jn5kq	1/1	Running	0	2m33s
pod/nginx-748c667d99-jtxzp	1/1	Running	0	2m33s
pod/nginx-748c667d99-mh5jr	1/1	Running	0	6m34s
pod/nginx-748c667d99-tqkwc	1/1	Running	0	2m33s
pod/nginx-748c667d99-v9cpf	1/1	Running	0	2m33s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	7m4s
service/nginx	NodePort	10.109.249.90	<none>	8s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nginx	6/6	6	6	6m34s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/nginx-748c667d99	6	6	6	6m34s

```
[root@formation ~]# kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx	NodePort	10.109.249.90	<none>	80:32401/TCP	67s

Un service permet de créer un point d'accès unique exposant notre déploiement. Ici nous utilisons le type de service Nodeport. Un port est maintenant mappé aux applications

Simplifier les lignes de commande

- Pour gagner du temps on dans les commandes Kubernetes on peut définir un alias: `alias kc='kubectl'` (à mettre dans votre `.bash_profile` en faisant `echo "alias kc='kubectl'" >> ~/.bash_profile`, puis en faisant `source ~/.bash_profile`).

- `services` **devient** `svc`
- `deployments` **devient** `deploy`
- etc.

La liste complète : <https://blog.heptio.com/kubectl-resource-short-names-heptiopro-tip-c8eff9fb7202>

- Essayez d'afficher les `serviceaccounts` (users) et les namespaces avec une commande courte.

4 Ressources - Objets Kubernetes

L'API et les Objets Kubernetes

Utiliser Kubernetes consiste à déclarer des objets grâce à l'API Kubernetes pour décrire l'état souhaité d'un cluster : quelles applications ou autres processus exécuter, quelles images elles utilisent, le nombre de replicas, les ressources réseau et disque que vous mettez à disposition, etc.

On définit des objets généralement via l'interface en ligne de commande et `kubectl` de deux façons avec :

- La commande `kubectl run <conteneur> ...`, `kubectl expose ...`
- Un objet dans un fichier YAML ou JSON et `kubectl apply -f monpod.yml`

Vous pouvez également écrire des programmes qui utilisent directement l'API Kubernetes pour interagir avec le cluster et définir ou modifier l'état souhaité. Kubernetes est complètement automatisable !

La commande `apply`

Kubernetes encourage le principe de l'infrastructure-as-code : il est recommandé d'utiliser une description YAML et versionnée des objets et configurations Kubernetes plutôt que la CLI.

Pour cela la commande de base est `kubectl apply -f object.yaml`.

La commande inverse `kubectl delete -f object.yaml` permet de détruire un objet précédemment appliqué dans le cluster à partir de sa description.

Lorsqu'on vient d'appliquer une description on peut l'afficher dans le terminal avec `kubectl apply -f myobj.yaml view-last-applied`

Globalement Kubernetes garde un historique de toutes les transformations des objets : on peut explorer, par exemple avec la commande `kubectl rollout history deployment`.

Objets de base

Les namespaces

Tous les objets Kubernetes sont rangés dans différents espaces de travail isolés appelés `namespaces`.

Cette isolation permet 3 choses :

- Ne voir que ce qui concerne une tâche particulière (ne réfléchir que sur une seule chose lorsqu'on opère sur un cluster)
- Créer des limites de ressources (CPU, RAM, etc.) pour le namespace
- Définir des rôles et permissions sur le namespace qui s'appliquent à toutes les ressources à l'intérieur.
- Positionner des quotas d'objets, nombre de Pods, nombre de service, ..
- Créer des règles d'accès réseaux vers les ressources de l'espace de nom

Lorsqu'on lit ou créé des objets sans préciser le namespace, ces objets sont liés au namespace `default`.

Pour utiliser un namespace autre que `default` avec `kubectl` il faut :

- le préciser avec l'option `-n`: `kubectl get pods -n kube-system`
- créer une nouvelle configuration dans la `kubeconfig` pour changer le namespace par défaut.

Kubernetes gère lui-même ses composants internes sous forme de pods et services.

Créer son espace de nom

- `kubectl create namespace forma-ludo`
- Se positionner dans son espace de nom
- `kubectl config set-context --current --namespace forma-ludo`

Nous allons préférer le mode déclaratif. La création de ressources avec des manifests. Dans le répertoire `$HOME/formation/`. Editons le fichier `namespace.yaml`

```
apiVersion: v1
kind: Namespace
metadata:
  name: forma-ludo
```

```
[ludo@formation k8s]$kubectl apply -f namespace.yaml
namespace/forma-ludo created
```

```
[ludo@formation k8s]$kubectl config set-context --current --namespace forma-
ludo
Context "kubernetes-admin@kubernetes" modified.
```

Gestion des quotas

Dans le repertoires k8s/namespace

```
[ludo@formation namespace]$ls
namespace.yaml pod-quota-mem-exceed.yaml pod-quota-mem.yaml quota-
count.yaml quota-limitrange.yaml quota-mem.yaml
```

Limiter le nombre de Pods dans un namespace

```
[ludo@formation namespace]$cat quota-count.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota-demo1
  namespace: forma-ludo
spec:
  hard:
    pods: "2"
    configmaps: "1"
```

On applique le mainfest

```
[ludo@formation namespace]$ kubectl apply -f quota-count.yaml
```

Et on créer des Pods

```
[ludo@formation namespace]$cat quota-count.yaml
[ludo@formation namespace]$ kubectl run --image nginx nginx
pod/nginx created

[ludo@formation namespace]$ kubectl run --image nginx nginx0
pod/nginx0 created

[ludo@formation namespace]$ kubectl run --image nginx nginx1
Error from server (Forbidden): pods "nginx1" is forbidden: exceeded quota:
quota-demo1, requested: pods=1, used: pods=2, limited: pods=2
```

Déploiement d'une application microservice

Dans le répertoire k8s/microservice-demo

Modifier l'espace de noms et déployer l'application

```
[ludo@formation microservice-demo]$ kubectl apply -f kubernetes-manifests.yaml

[ludo@formation microservice-demo]$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
adservice-fb7bcb498-kdpvx	1/1	Running	5 (14d ago)	42d
cartservice-5f4f74f58f-dj76l	1/1	Running	3 (14d ago)	42d
checkoutservice-85955686b6-wt9sc	1/1	Running	3 (14d ago)	42d
currencyservice-7d94dbd854-2qnm	1/1	Running	5 (14d ago)	42d
.				

Afficher toutes les ressources :

```
[ludo@formation microservice-demo]$ kubectl get all
```

service/recommendationservice	ClusterIP	10.99.27.30	<none>
8080/TCP	42d		
service/redis-cart	ClusterIP	10.97.31.15	<none>
6379/TCP	42d		
service/shippingservice	ClusterIP	10.106.94.48	<none>
50051/TCP	42d		

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/adservice	1/1	1	1	42d
deployment.apps/cartservice	1/1	1	1	42d
deployment.apps/checkoutservice	1/1	1	1	42d

Ajouter un ingress pour accéder a l'application :

```
[ludo@formation microservice-demo]$ cat ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: microservice
  namespace: forma-ludo
spec:
  ingressClassName: nginx
  rules:
  - host: shop-ludo.ludovic.io
    http:
```

```
paths:
- backend:
    service:
      name: frontend
      port:
        number: 80
    path: /
    pathType: Prefix
```

Sécuriser l'application :

```
[ludo@formation microservice-demo]$ cat ingress-tls.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: microservice
  namespace: forma-ludo
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - shop.ludovic.io
  rules:
  - host: shop-ludo.ludovic.io
    http:
```

Appliquer le manifest secret-cloudfare.yaml et issuer.yaml apres modification :

```
[ludo@formation microservice-demo]$ kubectl apply -f
```

Sécuriser l'application :

```
[ludo@formation microservice-demo]$ cat ingress-tls.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: microservice
  namespace: forma-ludo
  annotations:
    #nginx.ingress.kubernetes.io/ssl-redirect: "true"
    cert-manager.io/issuer: issuer
spec:
  ingressClassName: nginx
  tls:
```

```
- hosts:
  - shop.ludovic.io #changer le nom d'hote
  secretName: tls-secret-2 #changer le numéro
  - shop.ludovic.io
rules:
- host: shop-ludo.ludovic.io
  http:
```