

## Sommaire

**Comprendre les microservices.**

**Comprendre les images**

**Créer et gérer les conteneurs**

**Créer ses images avec Dockerfile**

**Les microservices avec Docker-compose**

**La gestion des logs avec Elasticsearch, Kibana,  
logstash**

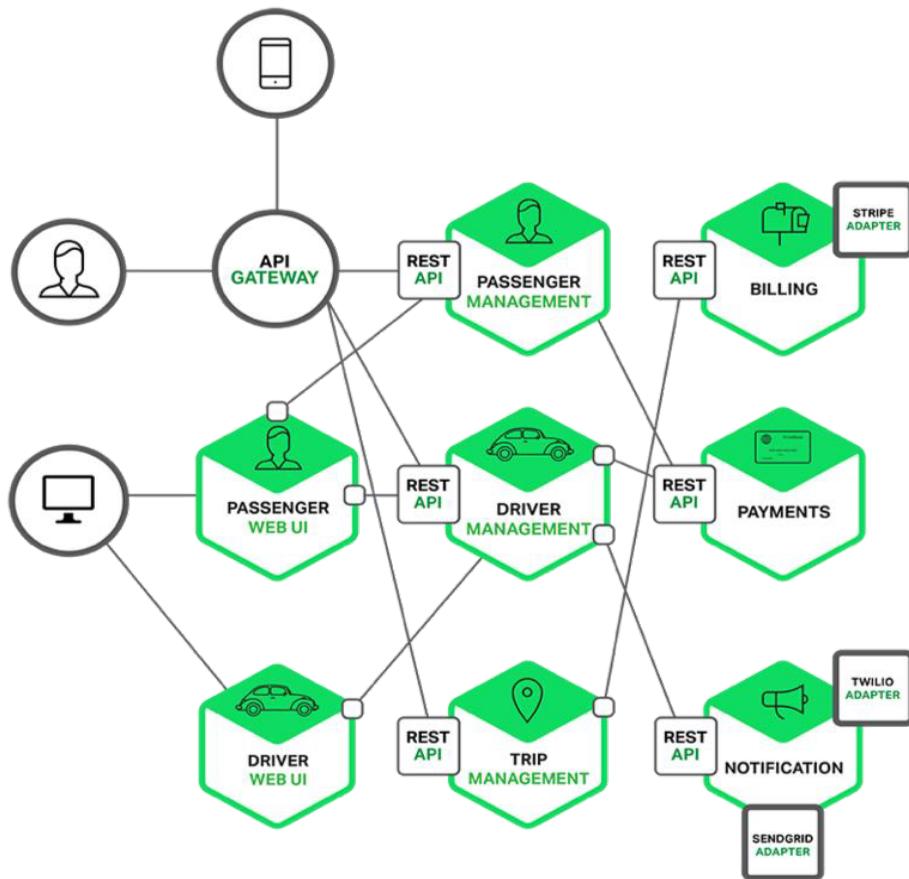
**Le Ci/CD avec Jenkins et gitlab**

**Orchestration des conteneurs**

## 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.

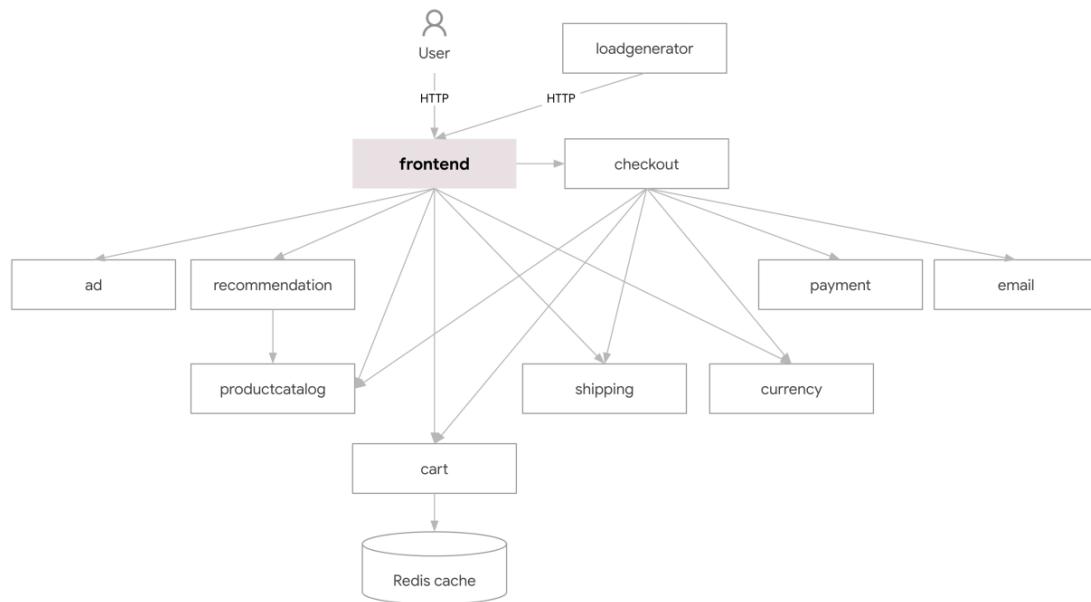


## Les microservices, qu'est-ce que c'est ?

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'en 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.

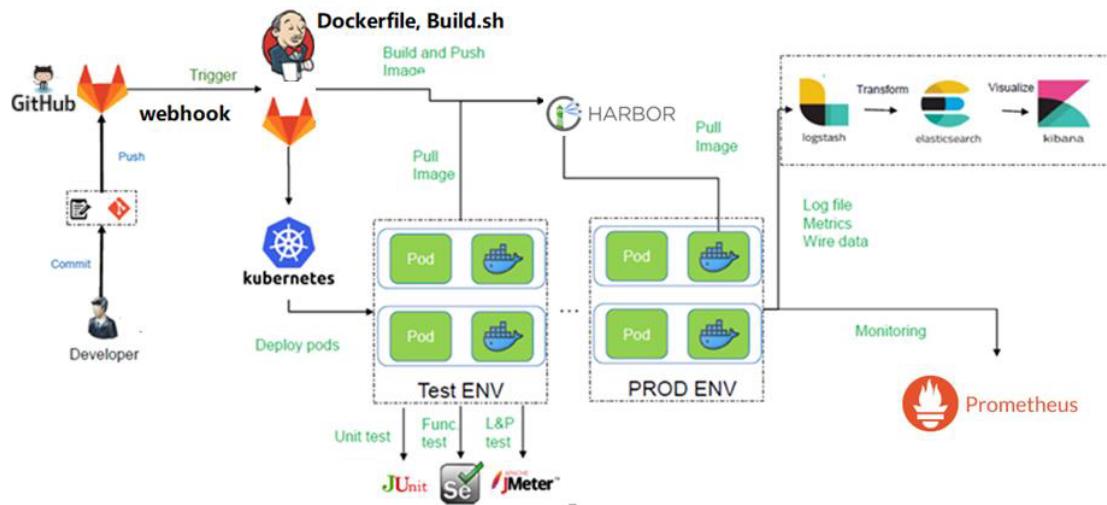
## Quel est l'impact des microservices sur l'intégration des applications ?

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 :

## Formation Docker

- 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 : 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 Personal 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

## Formation Docker

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 dépendances 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 claire et documentée 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)

## Formation Docker

- 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 le réseaux selon différents modes API REST, gRPC, job queues, GraphQL)

Les microservices permettent justement le DevOps car :

- Ils peuvent être déployés séparément

## **Formation Docker**

Une petite équipe gère chaque service ou groupe thématique de services

## Formation Docker

# Connexion sur le serveur de formation

Se connecter sur le serveur de formation

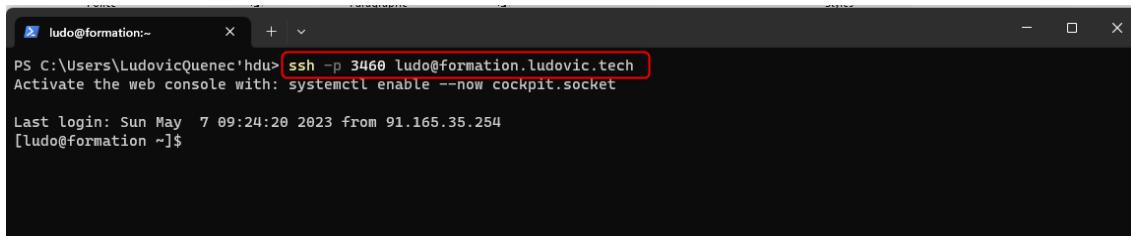
Serveur : formation.ludovic.tech

Port ssh : 3460

login : prénom (SANS ACCENT , NI MAJUSCULE)

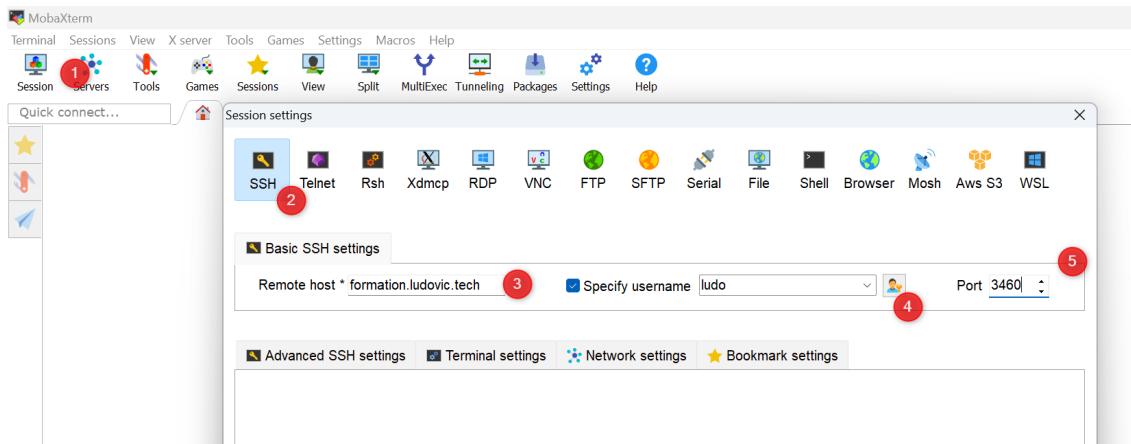
password: plb@2023

Avec un terminal Windows 11



```
ludo@formation:~> PS C:\Users\LudovicQuenec'hdhu> 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



## **Formation Docker**

Se connecter sur son hôte Docker

```
[ludo@formation:~$]ssh root@prenom  
password : root
```

Ajouter un utilisateur dans le groupe Docker

```
[root@dockerHost:~]# useradd -g docker Login_de_votre_choix  
[root@dockerHost:~]# su - votre_login  
[votre_login@dockerHost:~]$ docker version  
[votre_login@dockerHost:~]$ docker info
```

**Formation Docker**

## Les images

### L'image est un composant de base Docker.

Les images Docker sont des modèles de fichiers binaires qui contiennent tous les éléments nécessaires pour exécuter une application, y compris le code source, les bibliothèques, les dépendances, les fichiers de configuration, etc. Une image Docker est créée à partir d'un fichier appelé Dockerfile, qui contient les instructions pour construire l'image.

Les images Docker sont utilisées pour créer des conteneurs Docker, qui sont des instances en cours d'exécution d'une image Docker. Les conteneurs sont des environnements isolés et portables qui permettent d'exécuter des applications de manière cohérente sur différentes plates-formes et infrastructures.

Voici les principales caractéristiques des images Docker :

- **Légèreté** : les images Docker sont légères et comprennent uniquement les éléments nécessaires pour exécuter l'application, ce qui les rend plus rapides à télécharger et à exécuter que les machines virtuelles traditionnelles.
- **Portabilité** : les images Docker sont portables et peuvent être utilisées sur différentes plates-formes, telles que les ordinateurs de bureau, les serveurs locaux, les serveurs cloud et les conteneurs.
- **Versioning** : les images Docker sont versionnées, ce qui permet de suivre les modifications apportées à l'application au fil du temps.
- **Réutilisation** : les images Docker peuvent être réutilisées pour exécuter plusieurs instances d'une même application ou de différentes applications, ce qui permet de réduire les coûts de développement et de déploiement.
- **Sécurité** : les images Docker sont créées à partir d'un fichier Dockerfile, qui contient des instructions pour construire l'image en toute sécurité, en évitant les vulnérabilités et les failles de sécurité.

## Formation Docker

Les images Docker sont largement utilisées dans les pipelines DevOps pour faciliter l'intégration et le déploiement continus des applications.

Les images Docker sont construites à partir de couches. Chaque couche est un ensemble de modifications apportées à l'image de base pour créer une nouvelle image. Les couches permettent de créer des images de manière efficace et de réduire la taille des images en ne stockant que les modifications par rapport à l'image de base.

Chaque couche est une image Docker autonome, qui peut être réutilisée pour créer d'autres images. Les couches sont stockées dans un cache local sur l'hôte Docker, ce qui permet de réduire le temps de construction des images lorsqu'elles sont construites à nouveau.

Voici un exemple de couches dans une image Docker :

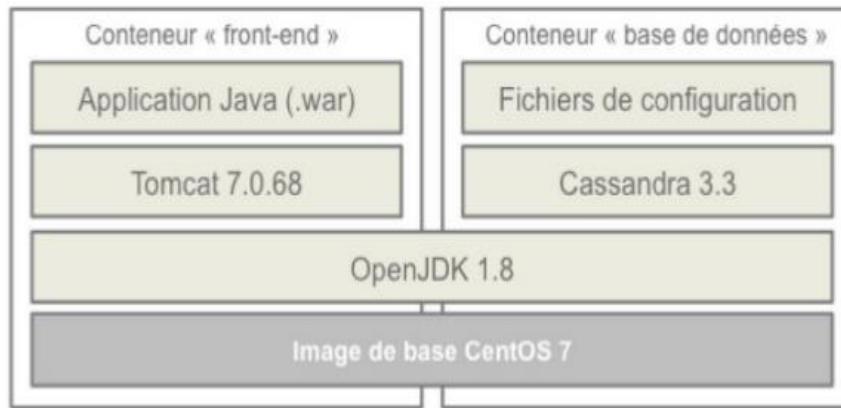
- Couche de base : cette couche contient l'image de base, qui peut être une distribution Linux, comme Alpine ou Ubuntu.
- Couche d'installation : cette couche contient les instructions pour installer les packages et les dépendances nécessaires à l'application.
- Couche de configuration : cette couche contient les fichiers de configuration de l'application, tels que les fichiers de configuration du serveur web ou de la base de données.
- Couche de code source : cette couche contient le code source de l'application.
- Couche d'exécution de l'application

Chaque fois que des modifications sont apportées à l'image Docker, une nouvelle couche est créée pour stocker ces modifications. Cela permet de minimiser le temps de construction en ne reconstruisant que les couches modifiées.

L'utilisation de couches dans les images Docker permet de réduire la taille des images et d'améliorer l'efficacité du processus de construction. Cela permet également de stocker et de réutiliser des couches pour créer d'autres images, ce qui contribue à améliorer la réproductibilité et la portabilité des applications.

## Formation Docker

Une image Docker est un modèle en lecture seule, utilisé pour créer des conteneurs Docker.  
Une image est constitué d'une série de couche



L'image est un composant de base Docker.

L'image contient une distribution, pas d'OS, un système d'exploitation est le noyau , pas les outils qui permettent d'utiliser la machine. GNU/LINUX.

On appelle cela l'image de base.

This screenshot shows the Docker Hub page for the official Ubuntu image. It features the Ubuntu logo, the text 'ubuntu DOCKER OFFICIAL IMAGE', and the message 'Updated 25 days ago'. On the right, it shows '1B+ Downloads' and '10K+ Stars'. Below this, a description states 'Ubuntu is a Debian-based Linux operating system based on free software.' and lists supported architectures: 'Linux ARM 64 PowerPC 64 LE riscv64 IBM Z 386 x86-64 ARM'.

L'image est un modèle en lecture seule qui contient une image de base (distribution minimal), le binaire de l'application, ses dépendances.

This screenshot shows the Docker Hub page for the official httpd image. It features the Apache logo, the text 'httpd DOCKER OFFICIAL IMAGE', and the message 'Updated 18 days ago'. On the right, it shows '1B+ Downloads' and '4.2K Stars'. Below this, a description states 'The Apache HTTP Server Project' and lists supported architectures: 'Linux ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z x86-64'.

L'image est la brique de base de Docker. On instancie une image afin de créer un conteneur. On dit que le **run** l'image. En fait, on exécute une commande lors de la création du conteneur. La commande qui se trouve l'image. La commande est exécuter dans le conteneur. Un conteneur = une application.

On crée ses propres, on dit que l'on construit, on **build** ses images. Nous étudierons plus tard dans cette formation le processus de **build** d'image et la construction automatique avec Jenkins et Gitlab. Dans un premier voyons comment administrer ses images au travers d'exemples.

## Manipuler ses images

Rechercher des images sur le Docker hub

```
[ludo@dockerHost:~]$ docker search centos
```

Télécharger une image Centos version 7 à partir du Docker hub

```
[ludo@dockerHost:~]$ docker image pull centos:7
```

Télécharger la dernière version de Centos

```
[ludo@dockerHost:~]$ docker image pull centos :latest
```

Récupérer l'image dockercloud/hello-world sur le Docker hub

```
[ludo@dockerHost:~]$ docker image pull dockercloud/hello-world
```

Récupérer l'image nginx en dernière version

```
[ludo@dockerHost:~]$ docker image pull nginx:latest
```

## **Formation Docker**

Lister les images sur l'hôte

```
[ludo@dockerHost:~]$ docker image ls
```

Lister les couches sur l'image nginx

```
[ludo@dockerHost:~]$ docker image history nginx
```

## Les images –Le Hub docker

Créer un compte sur le Docker hub a l'adresse : <https://hub.docker.com/>

### Get Started Today for Free

Already have an account? [Sign In](#)

Docker ID

---

Email

---

Password 

---

Send me occasional product updates and announcements.

Je ne suis pas un robot   
reCAPTCHA  
Confidentiality • Conditions

**Sign Up**

By creating an account, you agree to the [Terms of Service](#),  
[Privacy Policy](#), and [Data Processing Terms](#).

S'identifier sur le hub Docker

```
[ludo@dockerHost:~]$ docker login
username: quenec
password: *****
LoginSucceeded
```

Renommer une image et pousser une image sur le Hub

```
[ludo@dockerHost:~]$ docker image tag centos:latest quenec/centos:7.5
[ludo@dockerHost:~]$ docker image push quenec/centos:7.5
The push refers to repository [docker.io/quenec/centos]
```

## Formation Docker

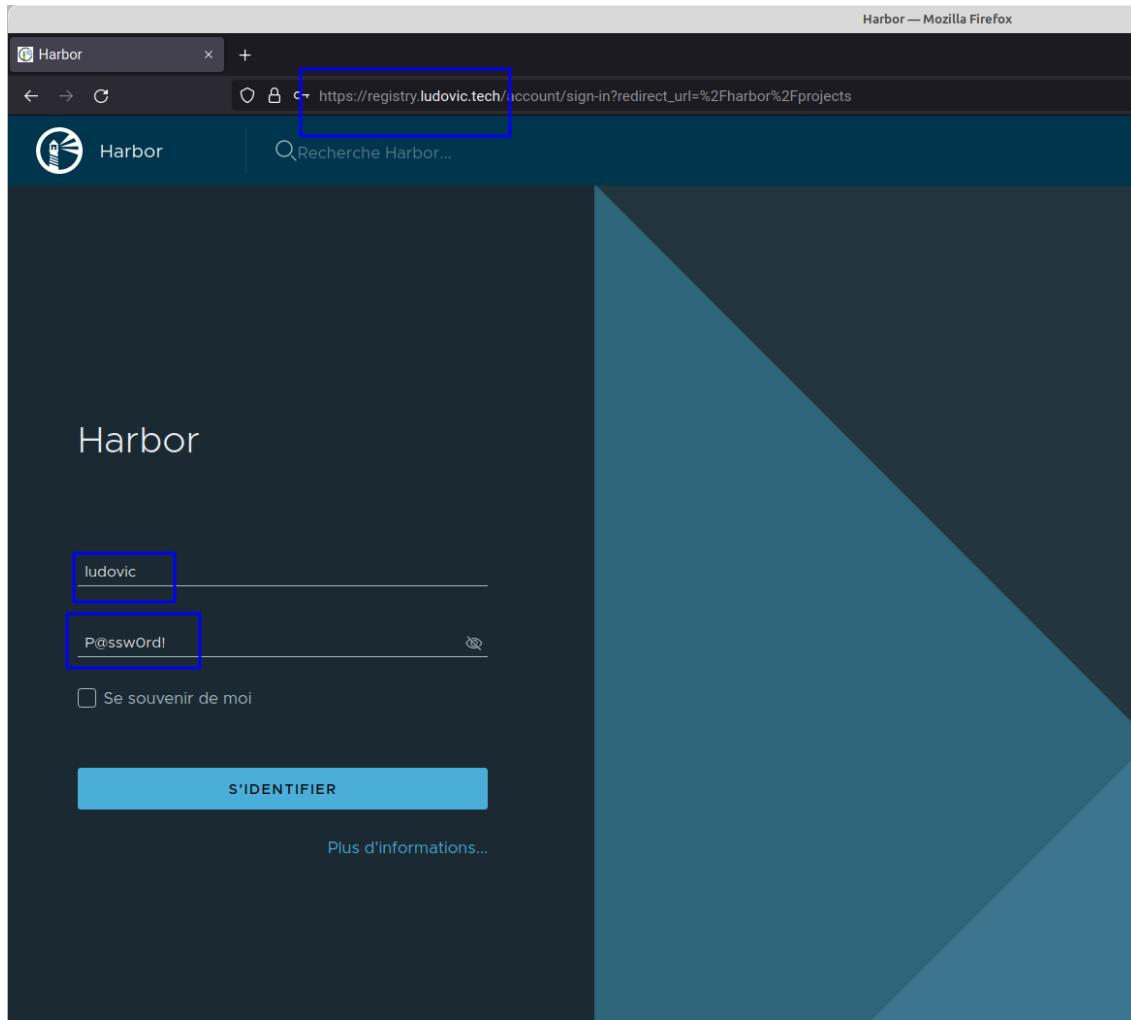
2653d992f4ef: Mounted from library/centos

7.5: digest: sha256:dbbacecc49b088458781c16f3775f2a2ec7521079034a7ba499c8b0bb7f86875  
size: 529



## Registre d'entreprise Harbor

Connexion sur notre dépôt d'image Harbor



S'identifier sur le hub Docker

```
ludo@dockerHost:~]$ docker login registry.ludovic.tech
username: user
password: P@sswOrd!
LoginSucceeded
```

## Formation Docker

Renommer une image et pousser une image sur le Hub

```
ludo@dockerHost:~]$ docker image tag nginx :latest registry.ludovic.tech/formation/monImage:latest
```

Poussez cette image dans le registre Harbor

```
ludo@dockerHost:~]$ docker image push registry.ludovic.tech/formation/monImage:latest
```

The screenshot shows the Harbor UI interface for the 'formation' repository. The 'Dépôts' tab is active. A table lists three images: 'formation/alpine', 'formation/web', and 'formation/apache'. The 'formation/web' row is selected and highlighted with a blue border around its entire row.

<input type="checkbox"/>	Nom	Artéfacts	Pulls
<input type="checkbox"/>	formation/alpine	1	0
<input type="checkbox"/>	formation/web	5	2
<input type="checkbox"/>	formation/apache	1	7

## Les conteneurs – Nos premiers conteneurs

Créer un conteneur nommé Helloworld à partir de l'image hello-world

```
[ludo@dockerHost:~]$ docker container run --name helloWorld hello-world
```

Lister tous les conteneurs

```
[ludo@dockerHost:~]$ docker container ls --all
```

Supprimer le conteneur créé précédemment

```
[ludo@dockerHost:~]$ docker container rm helloWorld
```

Créer un conteneur « éphémère »

```
[ludo@dockerHost:~]$ docker container run --rm hello-world
```

Créer un conteneur nommé apache avec les options terminal et interactive à partir d'une image centos :7

```
[ludo@dockerHost:~]$ docker run --interactive --tty --name apache centos:7
```

Sortir de la console du conteneur avec la combinaison de touche **ctrl+p** **ctrl+q**

```
[root@5deeb8a5d681 /]# CTRL+p+q
```

```
[ludo@dockerHost:~]$
```

Lister les conteneurs

```
[ludo@dockerHost:~]$ docker container ls
```

Attacher le terminal au conteneur

```
[ludo@dockerHost:~]$ docker container attach apache
```

```
[root@5deeb8a5d681 /]
```

## Formation Docker

Installer un serveur web apache dans un le conteneur

```
[root@5deeb8a5d681 /] yum install -y httpd
```

Démarrer le serveur Web

```
[root@5deeb8a5d681 /]httpd
```

Sortir et afficher l'adresse IP du container

```
[root@5deeb8a5d681 /]# CTRL+p+q
[ludo@dockerHost:~]$ docker container inspect apache | grep "IPAddress"
[ludo@dockerHost:~]$docker inspect -f "{{ .NetworkSettings.IPAddress }}" apache
```

Afficher la page index.html de votre serveur web

```
[ludo@dockerHost:~]$curl http://172.17.0.X
```

Modifier la page index.html de votre conteneur apache

```
[ludo@dockerHost:~]$docker container attach apache
[root@5deeb8a5d681 /]echo "<h1> Bienvenue formation Docker</h1>" \
/var/www/html/index.html
```

Afficher la page index.html de votre serveur web

```
[ludo@dockerHost:~]$curl http://172.17.0.X
```

Création d'une image Centos avec Apache installé

```
[ludo@docker-host-00$docker commit apache local/centos_httpd
```

Lister la nouvelle image

```
[ludo@dockerHost:~]$docker image ls
```

## Formation Docker

Suppression du conteneur apache

```
[ludo@dockerHost:~]$ docker container rm -f apache
```

Instancier la nouvelle image, dans le terminal récupérer l'adresse ip du conteneur et tester la connexion avec curl

```
[ludo@dockerHost:~]$ docker container run -d --name apache ... ..... local/centos_httpd  
[ludo@dockerHost:~]$ docker inspect -f "{{ .NetworkSettings.IPAddress }}" apache  
[ludo@dockerHost:~]$ curl 172.17.0.x
```

Que peut-on remarquer ?

Nettoyage de nos conteneurs

```
[ludo@dockerHost:~]$ docker container rm -f $(docker container ls -a -q )
```

## Les conteneurs – Le mode détaché

Exécuter un conteneur en mode détaché

```
[ludo@dockerHost:~]$ docker container run --name web --detach nginx:latest
```

Récupérer l'adresse ip du conteneur

```
[ludo@dockerHost:~]$ docker container inspect web | grep "IPAddress"  
[ludo@dockerHost:~]$ docker inspect -f "{{ .NetworkSettings.IPAddress }}" web
```

Afficher la page par default du conteneur

```
[ludo@dockerHost:~]$ curl 172.17.0.2  
.....
```

Ouvrir un Shell dans le conteneur et modifier le contenu du fichier index.html

```
[ludo@dockerHost:~]$ docker container exec -it web /bin/bash  
[root@bc70c9729c3/]# echo "Bienvenue formation Docker" > \  
/usr/share/nginx/html/index.html
```

Modifier le contenu du fichier index.html avec docker exec

```
[ludo@dockerHost:~]$ docker container exec web /bin/sh -c 'echo "Bienvenue formation \  
Docker" > /usr/share/nginx/html/index.html'
```

Supprimer le conteneur

```
[ludo@dockerHost:~]$ docker container rm --force web
```

## Les conteneurs – La redirection de ports

Exposer le port du conteneur accessible via l'hôte Docker

```
[ludo@dockerHost:~]$ docker container run --detach --name web --publish 8900:80 nginx
```

Lister les conteneurs en exécution, se connecter sur le conteneur

```
[ludo@dockerHost:~]$ docker container ls  
[ludo@dockerHost:~]$ curl localhost:8900
```

Exposer le port du conteneur avec un port aléatoire choisi par docker

```
[ludo@dockerHost:~]$ docker container run --detach --name web --publish-all nginx  
[ludo@dockerHost:~]$ docker container ls  
[ludo@dockerHost:~]$ curl localhost:45690
```

Exposer le port du conteneur avec un port aléatoire

```
[ludo@dockerHost:~]$ docker container run --detach --name web --publish-all nginx
```

## Les conteneurs – Les volumes Docker

Créer un répertoire \$HOME/www et associer le répertoire avec un répertoire dans le conteneur

```
[ludo@dockerHost:~]$ docker container run --detach --name web \
--volume $HOME/www:/usr/share/nginx/html --publish 8900:80 nginx
```

Afficher le contenu du fichier index.html

```
[ludo@dockerHost:~]$ curl localhost:8900
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
....
```

Modifier le fichier index.html localement et afficher de nouveau le contenu du serveur web

```
[ludo@dockerHost:~]$ echo «Vous êtes sur le serveur de $USER» >$HOME/www/index.html
[ludo@dockerHost:~]$ curl localhost:8900
« Vous êtes sur le serveur de ludo »
```

Créer un volume avec la commande docker volume et inspecter le volume

```
[ludo@dockerHost:~]$ docker volume create www
[ludo@dockerHost:~]$ docker volume inspect www
[
  {
    "CreatedAt": "2021-04-17T13:46:24+02:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/www/_data",
    "Name": "www"
```

## Formation Docker

Créer un conteneur nginx avec le nouveau volume

```
[ludo@dockerHost:~]$ docker container rm --force web
[ludo@dockerHost:~]$ docker container run --detach --name web \
--volume www:/usr/share/nginx/html --publish 8900:80 nginx
```

Afficher le contenu du fichier index.html

```
[ludo@dockerHost:~]$ curl localhost:8900
<!DOCTYPE html>
<head>
<title>Welcome to nginx!</title>
```

Modifier le fichier index.html

```
[ludo@dockerHost:~]$ echo « Vous êtes sur le serveur de $USER » > \
/var/lib/docker/volumes/www/_data/index.html
-bash: /var/lib/docker/volumes/www/_data/index.html: Permission denied
```

Résoudre les permissions ?

```
[ludo@dockerHost:~]$ mkdir www-data
[ludo@dockerHost:~]$ docker volume create --name www-data --opt type=none \
--opt device=$HOME/www-data --opt o=bind
```

Monter le nouveau volume www-data et modifier le fichier index.html

```
[ludo@dockerHost:~]$ docker container run --detach --name web \
--volume www-data:/usr/share/nginx/html --publish 8900:80 nginx
[ludo@dockerHost:~]$ echo « Vous êtes sur le serveur de $USER » > \
$HOME/www-data/index.html
$HOME/www-data/index.html: Permission denied
```

## Formation Docker

```
[ludo@dockerHost:~]$ ls -l $HOME/www-data/
-rw-r--r-- 1 root root 494 Apr 13 17:13 50x.html
-rw-r--r-- 1 root root 612 Apr 13 17:13 index.html
```

L'utilisateur est root dans le conteneur. !!

Modifier le contenu du fichier index.html avec docker container exec

```
[ludo@dockerHost:~]$ docker container exec web /bin/sh -c 'echo "Bienvenue \
formation Docker" > /usr/share/nginx/html/index.html'
```

Afficher le contenu du fichier index.html

```
[ludo@dockerHost:~]$ curl localhost:8900
Bienvenue formation Docker
.....
```

## Les conteneurs – Les variables d'environnements

Créer un conteneur de base de données avec une image mariadb :latest

```
[ludo@dockerHost:~]$ docker container run --detach --name my_db \
--volume db-data:/var/lib/mysql/ --publish 3306:3306 mariadb:latest
```

Installer le client mariadb et se connecter sur la base de données

```
[ludo@dockerHost:~]$ sudo yum install -y -q mariadb.x86_64
[ludo@dockerHost:~]$ mysql -u root -p
Enter password:
ERROR 2002 (HY000): Can't connect to local MySQL server through socket
'/var/lib/mysql/mysql.sock'
```

Lister les conteneurs

```
[ludo@dockerHost:~]$ docker container ls -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
205a33e05d55 mariadb:latest "docker-entrypoint.s..." 3 minutes ago Exited (1) 3 minutes ago
my_db
```

Afficher les logs du conteneur

```
[ludo@dockerHost:~]$ docker container logs my_db
You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and
MYSQL_RANDOM_ROOT_PASSWORD
```

Supprimer le conteneur et spécifier une variable d'environnement

```
[ludo@dockerHost:~]$ docker container rm -force my_db
[ludo@dockerHost:~]$ docker container run --detach --name my_db \
--volume db-data:/var/lib/mysql/ --publish 3306:3306 \
--env MYSQL_ROOT_PASSWORD=root mariadb:latest
```

## Formation Docker

```
[ludo@dockerHost:~]$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
d68bc1b993ab	mariadb:latest	"docker-entrypoint.s..."	3 seconds ago	Up 2 seconds	0.0.0.0:3306->3306/tcp, :::3306->3306/tcp my_db

Tester la connexion sur le conteneur my\_db, installer le client mysql avec le paquet mariadb

```
[ludo@dockerHost:~]$ mysql -u root -p -h localhost -P 3306 --protocol=tcp
```

Enter password:

```
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

Plusieurs variables dans un conteneur

```
[ludo@dockerHost:~]$ docker container run --detach --name multi_env \  
--volume db-data:/var/lib/mysql/ --publish 3306:3306 \  
--env MYSQL_ROOT_PASSWORD=root --env MYSQL_USER=root \  
--env MYSQL_HOME=/etc/ --env TZ=Europe mariadb:latest
```

Plusieurs variables dans un conteneur dans le fichier mysql.env

```
[ludo@dockerHost:~]$ cat mysql.env
```

```
MYSQL_ROOT_PASSWORD=root  
MYSQL_USER=root  
MYSQL_HOME=/etc/  
TZ=Europe
```

```
[ludo@dockerHost:~]$ docker container run --detach --name multi_env \  
--volume db-data:/var/lib/mysql/ --publish 3306:3306 \  
--env-file mysql.env mariadb:latest
```

## Le Dockerfile

Dans ce chapitre, nous allons apprendre à créer des images Docker. Nous verrons les différentes façons de ce faire.

Pour cela, nous allons créer un fichier nommé "**Dockerfile**". Dans ce fichier Dockerfile, vous allez trouver l'ensemble de la **recette** décrivant l'image Docker dont vous avez besoin pour votre projet.

Chaque **instruction** que nous allons donner dans notre Dockerfile va créer une nouvelle **Couche**, *layer* correspondant à chaque **étape** de la construction de l'image, ou de la recette.

Le fichier Dockerfile est donc un fichier texte qui contient les instructions nécessaires à la création d'une image de conteneur. Ces instructions incluent l'identification d'une image existante à utiliser comme base, les commandes à exécuter pendant le processus de création d'image et une commande qui s'exécute quand de nouvelles instances de l'image de conteneur sont déployées.

**Docker build** est la commande du moteur Docker qui utilise un fichier Dockerfile et déclenche le processus de création d'image.

## Rappel sur les images

Les images Docker sont construites à partir de couches. Chaque couche est un ensemble de modifications apportées à l'image de base pour créer une nouvelle image. Les couches permettent de créer des images de manière efficace et de réduire la taille des images en ne stockant que les modifications par rapport à l'image de base.

Chaque couche est une image Docker autonome, qui peut être réutilisée pour créer d'autres images. Les couches sont stockées dans un cache local sur l'hôte Docker, ce qui permet de réduire le temps de construction des images lorsqu'elles sont construites à nouveau.

Voici un exemple de couches dans une image Docker :

- Couche de base : cette couche contient l'image de base, qui peut être une distribution Linux, comme Alpine ou Ubuntu.
- Couche d'installation : cette couche contient les instructions pour installer les packages et les dépendances nécessaires à l'application.

## Formation Docker

- Couche de configuration : cette couche contient les fichiers de configuration de l'application, tels que les fichiers de configuration du serveur web ou de la base de données.
- Couche de code source : cette couche contient le code source de l'application.
- Couche d'exécution de l'application

Chaque fois que des modifications sont apportées à l'image Docker, une nouvelle couche est créée pour stocker ces modifications. Cela permet de minimiser le temps de construction en ne reconstruisant que les couches modifiées.

L'utilisation de couches dans les images Docker permet de réduire la taille des images et d'améliorer l'efficacité du processus de construction. Cela permet également de stocker et de réutiliser des couches pour créer d'autres images, ce qui contribue à améliorer la reproductibilité et la portabilité des applications.

## Comment construit-on des images ?

**Dockerfile** est un fichier de configuration qui définit les instructions nécessaires à la création d'une image Docker. Le Dockerfile est utilisé pour automatiser le processus de création d'images Docker et pour garantir la reproductibilité des images sur différentes plates-formes et environnements. Voici les principales instructions de Dockerfile :

**FROM** : Cette instruction spécifie l'image de base à utiliser pour créer une nouvelle image. Cette instruction doit être la première instruction dans le fichier Dockerfile.

**RUN** : Cette instruction permet d'exécuter une commande dans l'image Docker pendant le processus de construction. Elle est utilisée pour installer des packages, configurer l'environnement ou exécuter d'autres commandes nécessaires pour la création de l'image.

**COPY / ADD** : Ces instructions permettent de copier des fichiers depuis l'hôte Docker vers l'image Docker. COPY copie des fichiers depuis l'hôte Docker vers l'image Docker, tandis que ADD peut également télécharger des fichiers depuis une URL.

**WORKDIR** : Cette instruction permet de définir le répertoire de travail pour les commandes ultérieures dans le Dockerfile. Elle est souvent utilisée pour définir le répertoire de travail pour les commandes RUN, COPY et CMD.

**EXPOSE** : Cette instruction permet de déclarer le port sur lequel l'application écoute. Elle est utilisée pour informer les utilisateurs de l'image Docker du port sur lequel l'application peut être accessible.

## Formation Docker

**CMD** : Cette instruction spécifie la commande à exécuter par défaut lorsque le conteneur est démarré. Elle est souvent utilisée pour spécifier la commande principale de l'application à exécuter dans le conteneur.

Ces instructions sont les plus couramment utilisées dans les Dockerfiles, mais il existe d'autres instructions qui peuvent être utilisées pour personnaliser davantage le processus de construction des images Docker, telles que ENTRYPPOINT, ENV et LABEL, ARG. L'utilisation judicieuse de ces instructions permet de créer des images Docker efficaces, reproductibles et portables.

L'instruction **ENTRYPPOINT** dans Dockerfile est utilisée pour spécifier la commande qui doit être exécutée lorsque le conteneur est démarré. Contrairement à l'instruction CMD, qui peut être remplacée lors de l'exécution du conteneur en spécifiant une commande différente en ligne de commande, l'instruction ENTRYPPOINT est considérée comme une commande de base qui ne peut pas être remplacée par une autre commande.

L'instruction ENTRYPPOINT peut être spécifiée de deux manières différentes :

- ENTRYPPOINT ["command", "arg1", "arg2"] : cette méthode spécifie la commande et ses arguments en tant qu'argument JSON Array. Cette méthode est recommandée car elle est plus explicite.
- ENTRYPPOINT command arg1 arg2 : cette méthode spécifie la commande et ses arguments en tant que chaîne de caractères. Cette méthode est plus concise, mais elle peut être ambiguë si des espaces sont utilisés dans les arguments.

L'instruction **ENTRYPPOINT** est souvent utilisée en combinaison avec l'instruction **CMD**, qui spécifie les arguments à passer à la commande ENTRYPPOINT. Les arguments spécifiés dans l'instruction CMD seront passés à l'instruction ENTRYPPOINT comme des arguments supplémentaires.

Par exemple, supposons que nous ayons un conteneur qui exécute une application Python. Nous pouvons utiliser l'instruction ENTRYPPOINT pour spécifier la commande Python à exécuter et l'instruction CMD pour spécifier le script Python à exécuter.

```
FROM python:3.9-slim-buster
COPY . /app
WORKDIR /app
ENTRYPOINT ["python", "app.py"]
CMD ["--option1", "value1", "--option2", "value2"]
```

## **Formation Docker**

Dans cet exemple, l'instruction ENTRYPPOINT spécifie que la commande à exécuter est "python app.py", tandis que l'instruction CMD spécifie les options et valeurs à passer à la commande Python. Lorsque le conteneur est démarré, la commande "python app.py --option1 value1 --option2 value2" sera exécutée.

## Création d'image – le Dockerfile

Créer un répertoire pour notre projet

```
[ludo@dockerHost:~]$ mkdir $HOME/net-tools  
[ludo@dockerHost:~]$ cd $HOME/net-tools
```

Édition du dockerfile

```
[ludo@dockerHost:~]$ cat dockerfile  
FROM debian:buster-slim  
LABEL maintainer="ludo@plb.fr"  
  
RUN apt-get update -y  
RUN apt-get install -y iputils-ping dnsutils net-tools  
  
CMD ["ping" , "1.1.1.1" ]
```

Construction de l'image

```
[ludo@dockerHost:~]$ docker image build --tag net-tools:1.0 .  
Sending build context to Docker daemon 2.048kB  
Step 1/4 : FROM debian:buster-slim  
--> 48e774d3c4f5  
...  
Successfully built 056a58186026
```

Lister l'image

```
[ludo@dockerHost:~]$ docker image ls net-tools:1.0
```

Modifions le dockerfile

```
[ludo@dockerHost:~]$ cat dockerfile  
FROM debian:buster-slim
```

## Formation Docker

```
LABEL maintainer="ludo@plb.fr"  
RUN apt-get update -y \  
    && apt-get install -y iputils-ping dnsutils net-tools \  
    && apt-get remove --purge --auto-remove -y \  
    && rm -rf /var/lib/apt/lists/*  
  
CMD ["ping" , "1.1.1.1"]
```

Construction de l'image

```
[ludo@dockerHost:~]$ docker image build --tag net-tools:latest .  
Sending build context to Docker daemon 2.048kB  
Step 1/4 : FROM debian:buster-slim  
---> 48e774d3c4f5  
...  
Successfully built 056a58186026  
Successfully tagged net-tools:latest
```

Lister les images – Que remarque t'on?

```
[ludo@dockerHost:~]$ docker image ls
```

Instanciation de la nouvelle image

```
[ludo@dockerHost:~]$ docker run -it --rm net-tools:latest
```

**Travaux pratique non guidé**

A partir d'une image d'une image de base **centos:7**.

Créer une image personnalisée avec un dockerfile

Qui a l'instanciation de cette dernière démarre un serveur web apache avec son index.html personnalisé (cf: COPY, ADD)

## Sécurisation de ses images Docker

Édition du dockerfile

```
[ludo@dockerHost:~]$ cat dockerfile
FROM debian:buster-slim
LABEL maintainer="ludo@plb.fr"
RUN groupadd - 10001 noroot && \
    adduser -g 10001 -u 1001 ludo

RUN apt-get update -y \
    && apt-get install -y iputils-ping dnsutils net-tools \
    && apt-get remove --purge --auto-remove -y \
    && rm -rf /var/lib/apt/lists/*
USER ludo
CMD ["ping" , "1.1.1.1" ]
```

Construction de l'image

```
[ludo@dockerHost:~]$ docker image build --tag net-tools:latest .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM debian:buster-slim
--> 48e774d3c4f5
...
Successfully built 056a58186026
Successfully tagged net-tools:latest
```

Travaux Pratiques non guidé :

Reprenez votre image Docker qui contient un serveur Web apache personnalisé et l'exécution de votre serveur Apache. Le serveur doit être démarré avec un utilisateur autre que root.

## La construction multi stage

Créer un répertoire pour notre projet

```
[ludo@dockerHost:~$ mkdir $HOME/multiStage  
[ludo@dockerHost:~$ cd $HOME/multiStage
```

Édition de notre dockerfile

```
[ludo@dockerHost:~/multiStage$ vim dockerfile  
FROM golang:1.7.3  
WORKDIR /root/  
COPY app.go .  
  
RUN go get -d -v golang.org/x/net/html \  
&& CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .  
  
CMD ["./app"]
```

Construction de l'image

```
[ludo@dockerHost:~$ docker image build --tag app-go:1.0 .  
Sending build context to Docker daemon 2.048kB  
Step 1/5 : FROM golang:1.7.3  
--> 48e774d3c4f5  
...  
Successfully built 056a58186026
```

Lister l'image

```
[ludo@dockerHost:~$ docker image ls app-go:1.0
```

## Formation Docker

Instancions l'image

```
[ludo@dockerHost:~]$ docker container run --rm app-go:1.0
```

Édition de notre dockerfile

```
[ludo@dockerHost:~/multiStage$ vim dockerfile

FROM golang:1.7.3

WORKDIR /root/
COPY app.go .
RUN go get -d -v golang.org/x/net/html \
    && CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /root/app
CMD ["./app"]
```

Construction de l'image

```
[ludo@dockerHost:~$ docker image build --tag app-go:2.0 .
Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM golang:1.7.3
--> 48e774d3c4f5
Successfully built 056a58186026
```

Lister l'image

```
[ludo@dockerHost:~]$ docker image ls app-go:2.0
```

Instancions l'image

## Formation Docker

```
[ludo@dockerHost:~]$ docker container run --rm app-go:2.0
{"internal":0,"external":0}
```

Que remarque t'on ?

## Création d'un registre avec authentification

Nous avons besoin de créer pour notre registry :

- un compte (login/password) ;
- des certificats SSL auto-signés pour sécuriser l'API du registry avec HTTPS.

Pour ce faire, nous allons utiliser un conteneur qui va générer ces fichiers dans des volumes qu'il nous suffira ensuite de brancher sur notre registry. Pour créer le login/password, nous aurons besoin de l'utilitaire htpasswd qui permet de générer une entrée cryptée dans un fichier, qui sera ensuite utilisé pour l'authentification (en mode http basic). Cet utilitaire est disponible dans le repository EPEL https-tools. Pour la création des certificats, nous emploierons de nouveau l'utilitaire openssl.

Voici le Dockerfile de notre conteneur de configuration :

```
[ludo@dockerHost:~]$ cat dockerfile
FROM centos:7

#Installe le repository EPEL
RUN yum install -y --setopt=tsflags=nodocs epel-release && yum clean all

#Installe les deux utilitaires dont nous avons besoin
RUN yum install -y --setopt=tsflags=nodocs httpd-tools openssl && yum clean all

#Identifie deux volumes destinés à recevoir les fichiers
RUN mkdir -p /opt/certs
RUN mkdir -p /opt/auth

VOLUME /opt/certs
VOLUME /opt/auth

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT /entrypoint.sh
```

## Formation Docker

Il fait usage d'un script ENTRYPPOINT nommé entrypoint.sh que vous prendrez soin de disposer dans le même répertoire que le Dockerfile. Voici son contenu :

```
[ludo@dockerHost:~]$ cat entrypoint.sh
#!/bin/bash

#Création du premier compte echo "Creation d'un compte pour l'utilisateur $REGLOGIN"
htpasswd -Bbn $REGLOGIN $REGPASSWD > /opt/auth/htpasswd

#Création des certificats echo "Creation des certificats SSL"
openssl req -newkey rsa:4096 -nodes -sha256 -keyout /opt/certs/domain.key \
-x 509 -days 365 -out /opt/certs/domain.crt -subj '/CN=localhost'
```

Nous pouvons maintenant construire notre image :

```
[ludo@dockerHost:~]$ docker image build -t regconfig .
```

Nous allons maintenant créer deux volumes nommés qui vont recevoir les fichiers à créer :

- un volume certs\_volume qui va recevoir les clefs publique et privée SSL ;
- un volume auth\_volume qui recevra le fichier htpasswd contenant le login/password encrypté.

Voici les instructions requises :

```
[ludo@dockerHost:~]$ docker volume create certs_volume
[ludo@dockerHost:~]$ docker volume create auth_volume
```

Nous pouvons maintenant lancer notre conteneur avec la commande suivante :

```
[ludo@dockerHost:~]$ docker run --env REGLOGIN=admin --env REGPASSWD=admin1234 \
-v certs_volume:/opt/certs -v auth_volume:/opt/auth regconfig
```

Voici quelques explications relatives à la ligne de commande ci-dessus :

--env spécifie les valeurs pour les deux variables d'environnement (REGLOGIN et

## **Formation Docker**

REGPASSWD) contenant respectivement le login et le mot de passe de notre compte sur le registre.

## Formation Docker

Lancement du registry :

```
[ludo@dockerHost:~]$ docker run -d -p 5000:5000 --restart=always \
--name registry \
-v auth_volume:/auth \
-v certs_volume:/certs \
-e "REGISTRY_AUTH=hpasswd" \
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/hpasswd \
-e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
-e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
registry:2
```

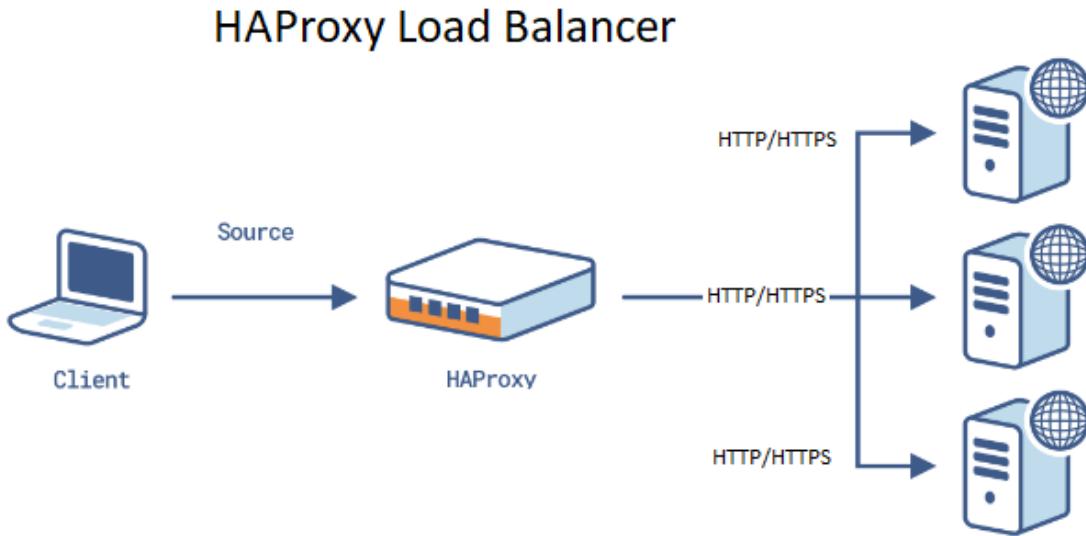
Une fois le registry lancé, vérifions que nous pouvons nous y connecter :

```
[ludo@dockerHost:~]$ docker login localhost:5000
Username (admin): admin
Password:
Login Succeeded
```

Taguer et pousser une image dans le registre :

```
[ludo@dockerHost:~]$ docker image tag mon-apache:latest localhost:5000/apache:latest
[ludo@dockerHost:~]$ docker image push localhost:5000/apache:latest
Login
....
```

## Projet haweb mult-conteneur - Serveur web Hautement disponible



Créer un répertoire pour notre projet

```
[ludo@dockerHost:~]$ mkdir $HOME/haweb  
[ludo@dockerHost:~]$ cd $HOME/haweb
```

Édition du dockerfile

```
[ludo@dockerHost:~]$ cat dockerfile  
FROM haproxy:2.3  
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg
```

## Formation Docker

Contenu du fichier haproxy.cfg

```
[ludo@dockerHost:~]$ cat haproxy.cfg
global

defaults

frontend public
    option forwardfor
    maxconn 800
    bind 0.0.0.0:80
    default_backend prive

backend prive
    balance roundrobin
    server web1 web1:80 check
    server web2 web2:80 check
    server web0 web0:80 check
```

Construction de notre image personnalisée

```
[ludo@dockerHost:~]$ docker image build --tag haweb:latest .
```

Création d'un réseau backend pour les serveurs web

```
[ludo@dockerHost:~]$ docker network create backend
```

Créer trois conteneur web sur le réseau backend

```
[ludo@dockerHost:~]$ docker container run --detach --name web1 --network backend \
--hostname web1 dockercloud/hello-world
```

Créer trois conteneurs web sur le réseau backend

## Formation Docker

```
[ludo@dockerHost:~]$ docker container run --detach --name web1 --network backend \
--hostname web2 dockercloud/hello-world
```

Créer trois conteneur web sur le réseau backend

```
[ludo@dockerHost:~]$ docker container run --detach --name web3 --network backend \
--hostname web3 dockercloud/hello-world
```

Création d'un réseau frontend pour le conteneur **haweb**

```
[ludo@dockerHost:~]$ docker network create frontend
```

Instancier le conteneur haweb

```
[ludo@dockerHost:~]$ docker container run --detach --name haweb --network backend \
--hostname haweb --publish 80:80 haweb :latest
```

Connecter le conteneur haweb sur le réseau frontend

```
[ludo@dockerHost:~]$ docker network frontend connect haweb
```

On test haproxy.

```
[ludo@dockerHost:~]$ curl localhost
```

## Docker Compose

Docker Compose est un outil qui permet de définir et de gérer des applications multi-conteneurs avec Docker. Il permet de décrire l'ensemble des services qui composent une application, leur configuration et les relations entre eux dans un fichier YAML.

Le fichier `docker-compose.yaml` permet de définir les services d'une application et de spécifier les dépendances et les relations entre eux. Chaque service peut être configuré avec des variables d'environnement, des volumes de données, des ports d'exposition, des liens vers d'autres services, etc.

Lorsque vous exécutez `docker-compose`, il crée automatiquement un réseau isolé pour votre application, où les conteneurs peuvent communiquer entre eux. Il peut créer également les volumes de données nécessaires pour stocker les données persistantes, comme les bases de données ou les fichiers de configuration.

Avec `docker-compose`, vous pouvez facilement déployer et gérer des applications multi-conteneurs sur un seul serveur ou sur plusieurs serveurs. Vous pouvez également utiliser `docker-compose` pour déployer votre application sur des plateformes cloud, comme AWS, Azure ou Google Cloud.

### Les services compose

Un service dans Docker Compose est une définition d'un conteneur Docker et de ses paramètres associés. Un service est généralement un conteneur qui exécute une application spécifique dans votre environnement Docker Compose.

Dans le fichier `docker-compose.yaml`, vous pouvez définir plusieurs services pour votre application, chacun avec ses propres paramètres. Par exemple, vous pouvez définir un service pour votre application web, un autre service pour votre base de données et un autre service pour un service tiers que votre application utilise.

Les services dans Docker Compose peuvent également être interconnectés à l'aide de réseaux définis dans le fichier `docker-compose.yaml`. Cela permet aux différents services de communiquer entre eux de manière transparente. De plus, vous pouvez spécifier des volumes de données pour chaque service, ce qui permet aux données d'être stockées de manière persistante entre les redémarrages des conteneurs.

En utilisant Docker Compose, vous pouvez facilement gérer les services de votre application et les déployer sur différents environnements, tels que des machines locales, des serveurs de test ou de production, des plateformes cloud, etc.

## Projet haweb mult-conteneur avec docker compose

Service web0, web1, web2 sur le réseau backend

```
[ludo@dockerHost:~]$ cat docker-compose.yml
version: '2.4'

services:
  web0:
    image: dockercloud/hello-world
    networks:
      - backend

  web1:
    image: dockercloud/hello-world
    networks:
      - backend

  web2:
    image: dockercloud/hello-world
    networks:
      - backend

networks:
  backend:
```

## Formation Docker

Service haproxy sur le réseau frontend, backend. Le service sera construit par docker-compose

```
[ludo@dockerHost:~]$ cat docker-compose.yml
version: '2.4'
services:
  ha:
    build: ./haproxy
    ports:
      - 80:80
    networks:
      - frontend
      - backend
    ...
  ...
```

Service web avec un volume

```
[ludo@dockerHost:~]$ cat docker-compose.yml
version: '2.4'
services:
  web1:
    image: dockercloud/hello-world
    networks:
      - backend
    volumes:
      - www-data:/www/
    ...
  ...
Volumes :
  www-data :
```

## Consolider les logs docker avec la suite ELK et Docker-compose

La multiplication des instances de conteneurs Docker peut vite rendre la consultation des logs des différentes applications un calvaire. Une solution est de consolider les logs de toute une infrastructure dans une seule application. Le choix le plus évident aujourd’hui est d’utiliser Elasticsearch Logstash et Kibana aka ELK.

- Elasticsearch : moteur d’indexation et de recherche
- Logstash : outil de pipeline de récupération de données opérant des transformations et poussant le résultat dans l’outil de persistence configuré. Un nombre important de connecteurs est disponible permettant de s’interfacer facilement avec les outils du marché ;
- Kibana : IHM de visualisation interagissant avec Elasticsearch pour mettre en forme les données via des graphiques, histogramme, carte géographique etc...

L’ensemble de ces composants sont disponibles sous forme d’image docker dans le docker hub, à savoir :

- Elasticsearch version 2.1.1
- Logstash version 2.1.1.-1
- Kibana version 4.3.1
- l’image elasticsearch en version 2.1.1 est utilisée
- le dossier /srv/elasticsearch/data de l’hôte est mappé sur le dossier /usr/share/elasticsearch/data du conteneur
- le port 9200 est exposé et est mappé tel quel sur l’hôte
- l’image logstash en version 2.1.1 est référencée
- le dossier conf/ du dossier courant est mappé sur le dossier /conf du conteneur
- le port 12201 en TCP et UDP est mappé sur l’hôte •le conteneur elasticsearch est lié à logstash, ce qui permet d’associer un nom d’hôte elasticsearch avec l’ip réelle du conteneur elasticsearch
- l’image kibana est démarrée en version 4.3, le port 5601 est mappé sur l’hôte et ce conteneur sera également lié à elasticsearch

Service ELK avec compose

## Formation Docker

```
[ludo@dockerHost:~]$ mkdir elk
[ludo@dockerHost:~]$ cd elk/

[ludo@dockerHost:~]$ cat docker-compose.yml
version: '2.4'

services:
  elasticsearch:
    image: elasticsearch:2.1.1
    volumes:
      - ./srv/elasticsearch/data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
  logstash:
    image: logstash:2.1.1
    environment:
      TZ: Europe/Paris
    ports:
      - "12201:12201"
      - "12201:12201/udp"
    volumes:
      - ./conf:/conf
    command: logstash -f /conf/gelf.conf
  kibana:
    image: kibana:4.3
    ports:
      - "5601:5601"
```

Le fichier de configuration de logstash référence le connecteur d'entrée gelf en écoutant sur le port 12201 et pousse sur notre elasticsearch :

```
[ludo@dockerHost:~]$ mkdir conf/
```

## Formation Docker

```
[ludo@dockerHost:~]$ cat conf/gelf.conf
input {
    gelf {
        type => docker
        port => 12201
    }
}
output {
    elasticsearch {
        hosts => elasticsearch
    }
}
```

L'arborescence de notre projet

```
[ludo@dockerHost:~]$ tree .
.
├── conf
│   └── gelf.conf
└── docker-compose.yaml
```

On injecte des données dans notre ELKx

```
[ludo@dockerHost:~]$ docker run --log-driver=gelf --log-opt \
gelf-address=udp://localhost:12201 debian bash -c 'seq 1 10'
```

## Formation Docker

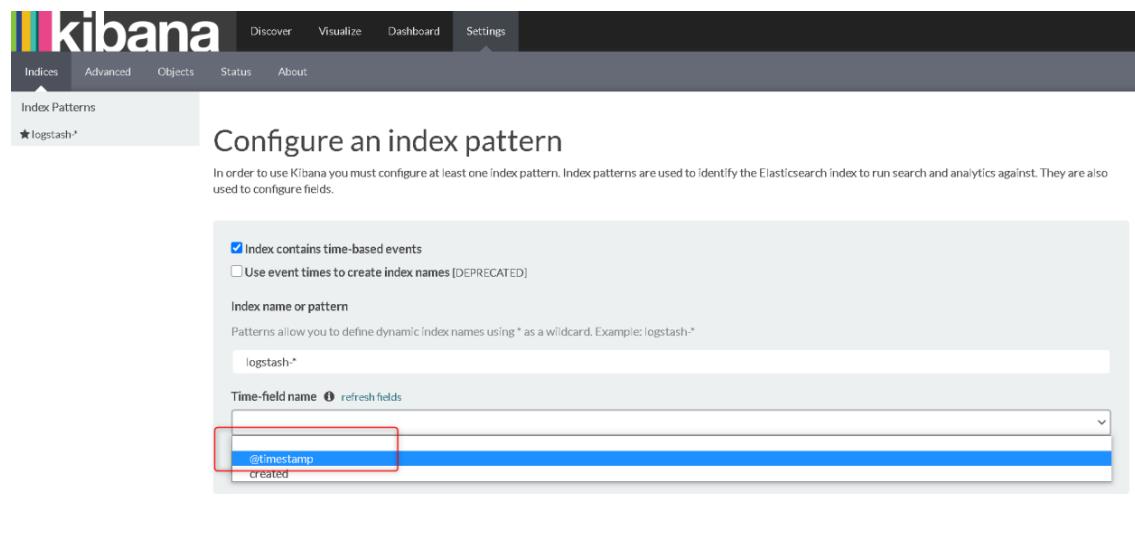
Se connecter sur l'HIM Kibana via le port 5601

[http://adresse IP:5601](http://adresse_IP:5601) Sur votre VM locale

Ou sur le serveur de formation :

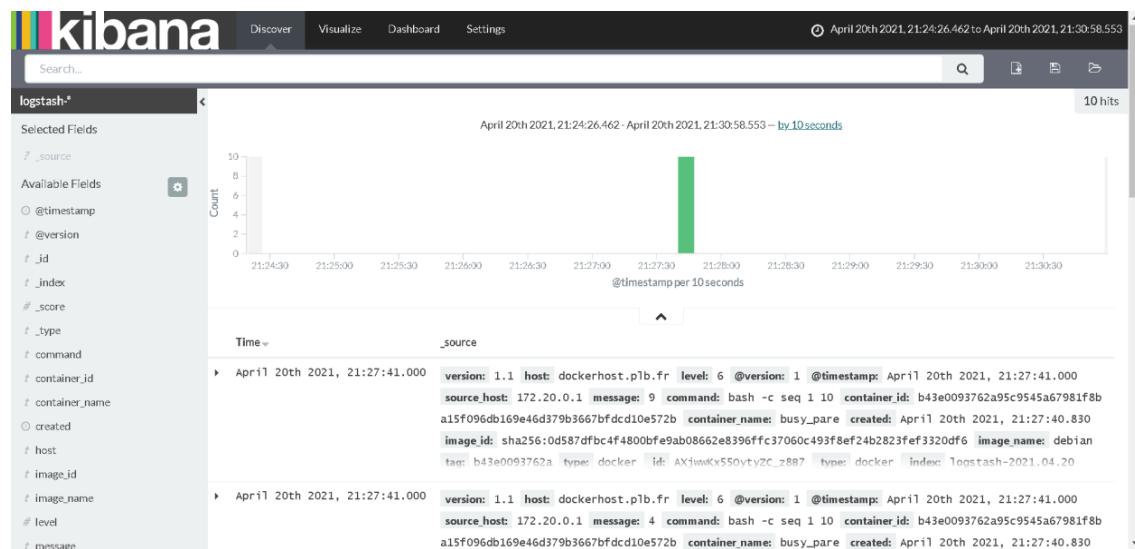
formation.ludovic.tech:56+IP (ex : ip = 10.0.0.80 port = 5680) formation.ludovic.tech5680

On crée un index avec Kibana. Un index est une base de données dans elasticsearchserach



The screenshot shows the Kibana interface for configuring an index pattern. The top navigation bar includes 'Discover', 'Visualize', 'Dashboard', and 'Settings'. The 'Indices' tab is selected. In the 'Index Patterns' section, there is one entry: 'logstash-\*'. Below this, under 'Time-field name', there is a dropdown menu containing '@timestamp' and 'created'. The '@timestamp' option is highlighted with a red box.

Voici nos journaux



The screenshot shows the Kibana 'Discover' page for the 'logstash-\*' index pattern. At the top, there is a search bar and a date range selector set to 'April 20th 2021, 21:24:26.462 to April 20th 2021, 21:30:58.553'. The results section shows '10 hits'. On the left, a sidebar lists 'Selected Fields' (including '\_source') and 'Available Fields' (including '@timestamp', '\_id', '\_index', '\_score', '\_type', 'command', 'container\_id', 'container\_name', 'host', 'image\_id', 'image\_name', 'level', and 'message'). The main area displays a histogram with a single bar at 21:27:30, labeled 'by 10 seconds'. Below the histogram is a table with two log entries:

Time	_source
April 20th 2021, 21:27:41.000	version: 1.1 host: dockerhost.plb.fr level: 6 @version: 1 @timestamp: April 20th 2021, 21:27:41.000 source_host: 172.20.0.1 message: 9 command: bash -c seq 1 10 container_id: b43e0093762a95c9545a67981f8ba15f096db169e46d379b3667bfdcd10e572b container_name: busy_pare created: April 20th 2021, 21:27:40.830 image_id: sha256:0d587dfbc4f4800bf9ab08662e8396ffcc37060c493f8ef24b2823fef3320df6 image_name: debian tag: b43e0093762a type: docker id: AX1mkx550ytyZC_z887 typer: docker index: logstash-2021.04.20
April 20th 2021, 21:27:41.000	version: 1.1 host: dockerhost.plb.fr level: 6 @version: 1 @timestamp: April 20th 2021, 21:27:41.000 source_host: 172.20.0.1 message: 4 command: bash -c seq 1 10 container_id: b43e0093762a95c9545a67981f8ba15f096db169e46d379b3667bfdcd10e572b container_name: busy_pare created: April 20th 2021, 21:27:40.830

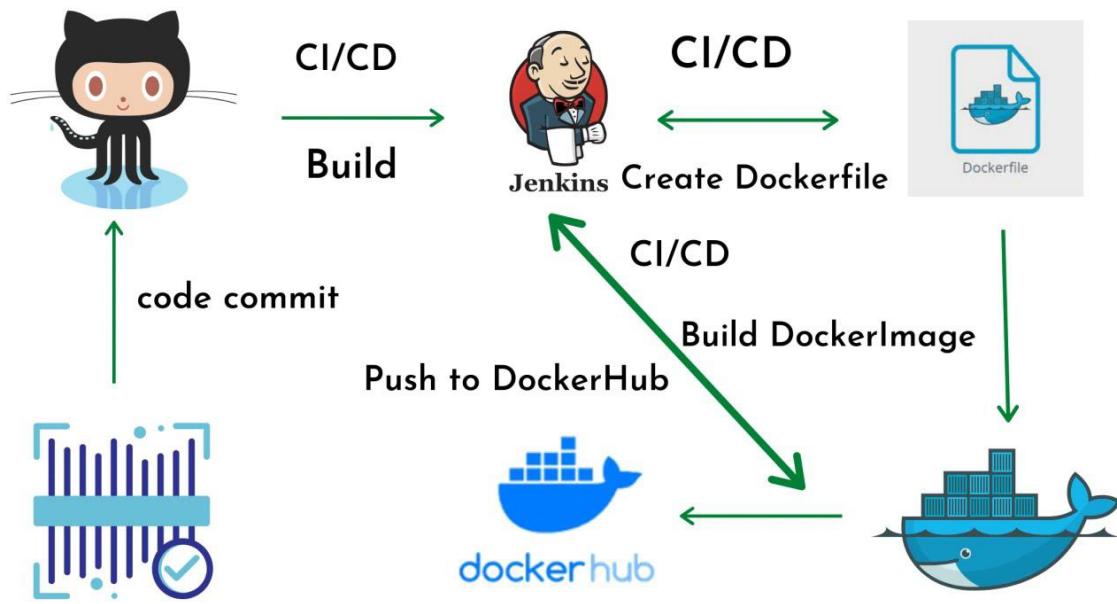
## Formation Docker

Ajout de l'envoie des logs dans nos applications

```
[ludo@dockerHost:~]$ cd haweb/  
  
[ludo@dockerHost:~]$ cat docker-compose.yml  
  
web0:  
  
  image: dockercloud/hello-world  
  
  networks:  
    - backend  
  
  logging:  
    driver: "gelf"  
  
  options:  
    gelf-address: "udp://localhost:12201"  
    tag: "Haweb-docker"
```



## CI/CD avec Docker, jenkins et gitlab



## Déploiements de Jenkins

```
[ludo@dockerHost:~]$ mkdir jenkins

[ludo@dockerHost:~]$ cat docker-compose.yml

version: '2.4'

services:

jenkinsci:

image: jenkinsci/blueocean:1.25.5

ports:

- 8080:8080

- 50000:50000

volumes:

- ./jenkins_home:/var/jenkins_home

- /var/run/docker.sock:/var/run/docker.sock

user: root
```

### Démarrer la configuration

Pour être sûr que que Jenkins soit configuré de façon sécurisée par un administrateur, un mot de passe a été généré dans le fichier de logs.

```
jenkinsci_1 | Jenkins initial setup is required. An admin user has been created and a password generated.
jenkinsci_1 | Please use the following password to proceed to installation:
jenkinsci_1 | ad1b795ce08c4e5ea9d413bf535ecd23
jenkinsci_1 | This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
jenkinsci_1 | ****
jenkinsci_1 | ****
jenkinsci_1 | ****
jenkinsci_1 |
root@docker-ludovic ~]#
[root@docker-ludovic ~]# docker-compose logs
```

## Débloquer Jenkins

Pour être sûr que que Jenkins soit configuré de façon sécurisée par un administrateur, un mot de passe a été généré dans le fichier de logs ([où le trouver](#)) ainsi que dans ce fichier sur le serveur :

`/var/jenkins_home/secrets/initialAdminPassword`

Veuillez copier le mot de passe depuis un des 2 endroits et le coller ci-dessous.

Mot de passe administrateur
.....



## Créer le 1er utilisateur Administrateur

Nom d'utilisateur:	<input type="text" value="admin"/>
Mot de passe:	<input type="password" value="....."/>
Confirmation du mot de passe:	<input type="password" value="....."/>
Nom complet:	<input type="text" value="ludovic"/>
Adresse courriel:	<input type="text" value="admin@ib.fr"/>

: 2.303.2

Continuer en tant qu'Administrateur

Sauver et continuer

## Formation Docker

### Plugins Docker et Docker pipelines

Voilà, nous avons déployé et configuré Jenkins. Passons maintenant à l'installation des plugins Docker et Docker pipelines.

The screenshot shows the Jenkins Administration interface. In the top navigation bar, the 'Administrer Jenkins' item is selected. On the left sidebar, the 'Administrer Jenkins' item is also highlighted. A search bar at the top contains the text 'docker'. Below the search bar, there are four tabs: 'Mises à jour' (Updates), 'Disponibles' (Available), 'Installés' (Installed), and 'Avancé' (Advanced). The 'Disponibles' tab is selected. A purple box highlights the search bar and the 'Disponibles' tab. The results list includes:

- Docker** (Selected):
  - Cloud Providers
  - Cluster Management and Distributed Build
  - docker

This plugin integrates Jenkins with Docker
- Docker Commons**:
  - api-plugin
  - docker
  - Library plugins (for use by other plugins)

Provides the common shared functionality for various Docker-related plugins.
- Docker Pipeline**:
  - Deployment
  - DevOps
  - docker
  - pipeline

Build and use Docker containers from pipelines.
- Docker API**:
  - api-plugin
  - docker

This plugin provides docker-java API for other plugins.

This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.

At the bottom of the page, there is a summary table of plugin status:

Plugin	Status
Mailer	Already Installed
Loading plugin extensions	Success
Docker Commons	Downloaded Successfully. Will be activated during the next boot
Docker API	Downloaded Successfully. Will be activated during the next boot
Docker	Downloaded Successfully. Will be activated during the next boot
Docker Pipeline	Downloaded Successfully. Will be activated during the next boot

Links at the bottom include 'Revenir en haut de la page' and 'Redémarrer Jenkins quand l'installation est terminée et qu'aucun job n'est en cours'.

On redémarre Jenkins, suite  
l'installation des plugins

## Formation Docker

### Notre premier job ou build

The screenshot shows the Jenkins dashboard with a new job creation dialog open. The job name is set to "MonPremierJob". The "Construire un projet free-style" option is selected, highlighted with a purple border. The "OK" button at the bottom left of the dialog is also highlighted with a purple border.

**Saisissez un nom**

MonPremierJob

» Champ obligatoire

**Construire un projet free-style**

Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour une version avec tous les systèmes de build.

**Pipeline**

Orchestrates long-running activities that can span multiple build agents. Suitable for and/or organizing complex activities that do not easily fit in free-style job type.

**Construire un projet multi-configuration**

Adapté aux projets qui nécessitent un grand nombre de configurations différentes binaires spécifiques à une plateforme, etc.

**Dossier**

Crée un conteneur qui stocke des objets imbriqués. Utile pour grouper ensemble de filtres, un dossier crée un espace de nommage distinct, de sorte que vous pouvez trouver dans des dossiers différents.

**Organization Folder**

Creates a set of multibranch project subfolders by scanning for repositories.

**Build**

Ajouter une étape au build ▾

- Add a new template to all docker clouds
- Appeler Ant
- Build / Publish Docker Image
- Exécuter un script shell**
- Exécuter une ligne de commande batch Windows
- Invoke Gradle script
- Invoquer les cibles Maven de haut niveau
- Run with timeout
- Set build status to 'pending' on GitHub commit
- Start/Stop Docker Containers

## Formation Docker

**Build**

Exécuter un script shell

Commande

```
docker image pull quenec/apache:latest
docker container run --rm quenec/apache:latest echo 'from jenkins'
```

Voir la liste des variables d'environnement disponibles

Ajouter une étape au build ▾

**Actions à la suite du build**

Ajouter une action après le build ▾

**Sauver** **Apply**

Tableau de bord > MonPremierJob >

Retour au tableau de bord

État

Modifications

Répertoire de travail

Lancer un build

Configurer

Supprimer Projet

Favoris

Open Blue Ocean

Rename

Historique des builds tendance ^

find x

Atom feed des builds Atom feed des échecs

Tableau de bord > MonPremierJob >

Retour au tableau de bord

État

Modifications

Répertoire de travail

Lancer un build

Configurer

Supprimer Projet

Favoris

Open Blue Ocean

Rename

Historique des builds tendance ^

find x

#4 12 oct. 2021 15:33

## Formation Docker

The screenshot shows the Jenkins interface for a job named "MonPremierJob". The build number is "#4". On the left, there's a sidebar with various links: "Retour au projet", "État", "Modifications", "Console Output" (which is highlighted with a red box), "View as plain text", "Informations de la construction", "Delete build '#4'", and "Open Blue Ocean". The main content area is titled "Sortie de la console" (Console Output) with a green checkmark icon. It displays the following log output:

```
Started by user ludovic
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/MonPremierJob
[MonPremierJob] $ /bin/sh -xe /tmp/jenkins10284921670976659918.sh
+ docker image pull quenec/apache:latest
latest: Pulling from quenec/apache
Digest: sha256:dead07b4d8ed7e29e98de0f4504d87e8880d4347859d839686a31da35a3b532f
Status: Image is up to date for quenec/apache:latest
docker.io/quenec/apache:latest
+ docker container run --rm quenec/apache:latest echo 'from jenkins'
from jenkins
Finished: SUCCESS
```

## Formation Docker

### Mon premier Pipeline Docker

Un pipeline de Build d'image Docker

The image shows two screenshots illustrating the setup of a GitHub repository and its configuration in Jenkins.

**GitHub Repository Setup:**

- Name:** BuidDocker (highlighted with a red box).
- Description:** Construire un projet free-style (highlighted with a red box).
  - Pipeline:** Organise des activités de longue durée qui (anciennement connues comme workflow type libre). (highlighted with a red box).
  - Construire un projet multi-configuration:** Adapté aux projets qui nécessitent un gra binaires spécifiques à une plateforme, etc.
- Dossier:** Crée un conteneur qui stocke des objets i un dossier crée un espace de nommage c dans des dossiers différents.

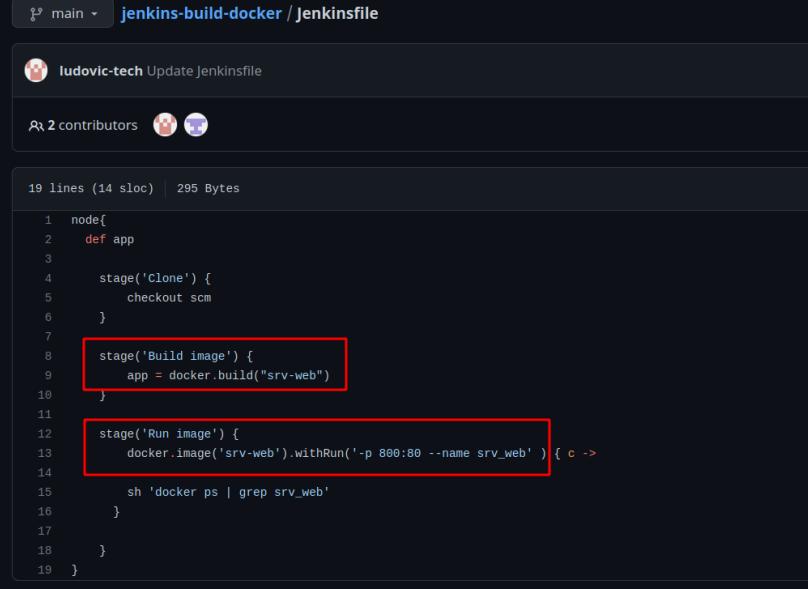
**Jenkins Pipeline Configuration:**

- Definition:** Pipeline script from SCM
  - SCM:** Git (highlighted with a red box).
  - Repository URL:** https://github.com/ludovic-tech/jenkins-build-docker.git (highlighted with a red box).
  - Credentials:** - aucun -
  - Branch Specifier (blank for 'any'):** \*/main (highlighted with a red box).
  - Script Path:** Jenkinsfile (highlighted with a red box).
  - Lightweight checkout:** checked (highlighted with a red box).
- Branches to build:** Branch Specifier (blank for 'any') \*/main (highlighted with a red box).

**Buttons:** OK, Sauver, Apply.

## Formation Docker

### Pipeline de Build



```
main ▾ jenkins-build-docker / Jenkinsfile

ludovic-tech Update Jenkinsfile
2 contributors

19 lines (14 sloc) | 295 Bytes

1 node{
2   def app
3
4   stage('Clone') {
5     checkout scm
6   }
7
8   stage('Build image') {
9     app = docker.build("srv-web")
10  }
11
12  stage('Run image') {
13    docker.image('srv-web').withRun('-p 800:80 --name srv_web') { c ->
14      sh 'docker ps | grep srv_web'
15    }
16  }
17
18 }
19 }
```

## Formation Docker

Tableau de bord > BuildDocker >

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Lancer un build](#) (highlighted with a red box)

[Configurer](#)

[Supprimer Pipeline](#)

[Full Stage View](#)

[Open Blue Ocean](#)

[Renommer](#)

[Pipeline Syntax](#)

[Historique des builds](#) (highlighted with a red box)

[tendance](#)

[Filter builds...](#)

[#1 Dec 16 07:13](#) (highlighted with a red box)

[No Changes](#)

[Atom feed des builds](#)

[Atom feed des échecs](#)

### Pipeline BuildDocker

#### Recent Changes

#### Stage View

Average stage times:  
(Average full run time: ~3s)

Clone	Build image	Run image
640ms	717ms	1s
640ms	717ms	1s

#### Liens permanents

The screenshot shows the Jenkins Pipeline BuildDocker dashboard. On the left, there's a sidebar with various options like Status, Changes, and a prominent 'Lancer un build' button. Below that is a 'Historique des builds' section with a dropdown for 'tendance'. A specific build entry for '#1' is highlighted with a red box, showing it was run on December 16 at 07:13 with 'No Changes'. At the bottom of this section are links for 'Atom feed des builds' and 'Atom feed des échecs'. The main area is titled 'Pipeline BuildDocker' and contains a 'Recent Changes' section with a pencil icon and a 'Stage View' section with a table showing average stage times for Clone (640ms), Build image (717ms), and Run image (1s). The 'Stage View' table has three rows, each corresponding to a build. The first row is labeled '#1' and shows the same timing data as the summary. The second and third rows also show the same timing data.

## Formation Docker

### Un pipeline de Build et de Push

1.



Nouveau Item

2.

**Saisissez un nom**

BuildAndPushDocker  
» Champ obligatoire

**Construire un projet free-style**  
Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pou

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for and/or organizing complex activities that do not easily fit in free-style job type.

**Construire un projet multi-configuration**  
Adapté aux projets qui nécessitent un grand nombre de configurations différentes, c binaires spécifiques à une plateforme, etc.

**Dossier**  
Crée un conteneur qui stocke des objets imbriqués. Utile pour grouper ensemble de filtre, un dossier crée un espace de nommage distinct, de sorte que vous pouvez avoir trouvent dans des dossiers différents.

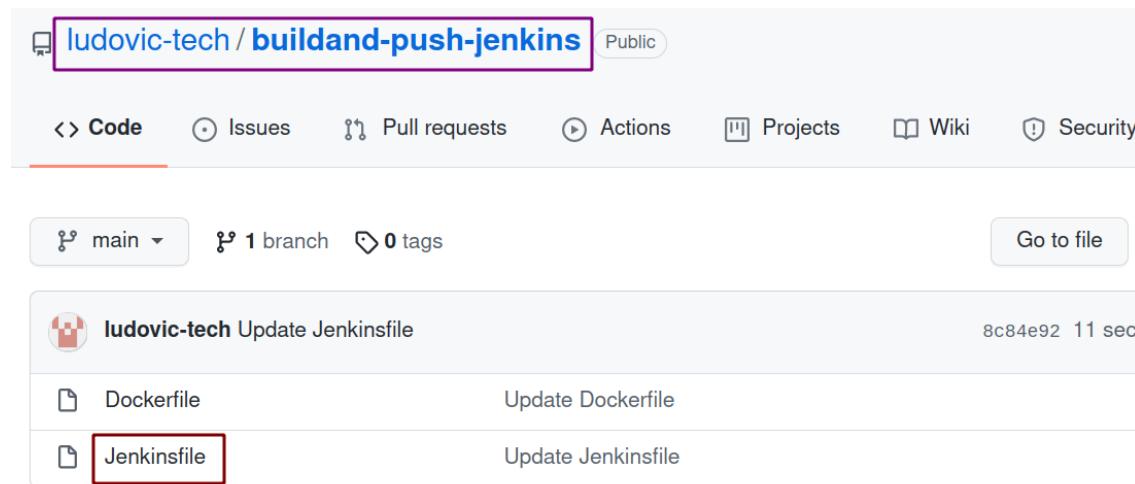
**Organization Folder**  
Creates a set of multibranch project subfolders by scanning for repositories.

**OK** Multibranches  
Créer un projet Pipeline en se basant sur les branches détectées dans

## Formation Docker

### A partir du dépôt git

<https://github.com/ludovic-tech/buildand-push-jenkins.git>



The screenshot shows a GitHub repository page for 'ludovic-tech / buildand-push-jenkins'. The repository is public. At the top, there are links for Code, Issues, Pull requests, Actions, Projects, Wiki, and Security. Below that, it shows 1 branch and 0 tags. A specific commit is highlighted with a red border, showing it updated the Jenkinsfile.

### 4. Jenkinsfile

```
node {  
  
    def registryProjet='localhost:5000/'  
    def IMAGE="${registryProjet}version-${env.BUILD_ID}"  
  
    stage('Clone') {  
        checkout scm  
    }  
  
    stage('Build') {  
        docker.build("$IMAGE", '.')  
    }  
  
    stage('Run') {  
        img.withRun("--name run-$BUILD_ID -p 22301:80") { c ->  
            }  
    }  
  
    stage('Push') {  
        docker.withRegistry('$registryProjet', 'registry_id') {  
            img.push()  
        }  
    }  
}
```

## Formation Docker

### 5. Configuration du Job

Pipeline

Definition

SCM

Git

Repositories

Repository URL

https://github.com/ludovic-tech/buildand-push-jenkins.git

Credentials

- aucun - Ajouter

Avancé... Add Repository

Branches to build

Branch Specifier (blank for 'any')

\*/main

This screenshot shows the Jenkins Pipeline configuration page. It's set to 'Pipeline script from SCM'. Under 'SCM', 'Git' is selected. The 'Repository URL' is set to 'https://github.com/ludovic-tech/buildand-push-jenkins.git'. There are no credentials yet. Under 'Branches to build', the 'Branch Specifier' is set to '\*/main'. Buttons for 'Avancé...' and 'Add Repository' are visible.

### 6. On créer un « credentials »

Credentials

- aucun - Ajouter Jenkins

Identifiants globaux (illimité)

Type

Nom d'utilisateur et mot de passe

Portée

Global (Jenkins, agents, items, etc...)

Nom d'utilisateur

admin

Treat username as secret

Mot de passe

\*\*\*\*\*

ID

registry\_id

Description

Login et password du registre local

Ajouter Annuler

This screenshot shows the Jenkins 'Credentials' dialog. It's set to 'Type: Nom d'utilisateur et mot de passe' (Global). The 'Nom d'utilisateur' is 'admin' and the 'Mot de passe' is masked as '\*\*\*\*\*'. The 'ID' is 'registry\_id' and the 'Description' is 'Login et password du registre local'. The 'Ajouter' button is highlighted with a red box.

7.

## Formation Docker

Repository URL  
https://github.com/ludovic-tech/buildand-push-jenkins.git

Credentials  
admin/\*\*\*\*\*\*\*\* (Login et password du registre local)

Branches to build

Branch Specifier (blank for 'any')  
\*/main

Navigateur de la base de code  
(Auto)

Additional Behaviours

Script Path  
Jenkinsfile

8.

Tableau de bord > BuildAndPushDocker

Lancer un build

Status

Changes

Recent Changes

Configure

Supprimer Pipeline

Full Stage View

Open Blue Ocean

Rename

Pipeline Syntax

Historique des builds

find

#1 12 oct. 2021 16:27

Pipeline BuildAndPushDocker

Stage View

Clone	Build	Run	Push
366ms	334ms	1s	1s

Average stage times:  
(Average full run time: ~3s)

#1 Oct 12 18:27 No Changes

Liens permanents

## CI/CD avec Gitlab

GitLab est une plateforme de développement collaboratif basée sur Git, un système de contrôle de version. Il permet aux développeurs de travailler ensemble sur un même projet, de partager du code, de le gérer, de le tester et de le déployer de manière efficace.

Dans le cadre de Docker, GitLab peut être utilisé pour gérer les images Docker, les registres d'images et les pipelines CI/CD. Les images Docker peuvent être stockées dans GitLab Container Registry, ce qui facilite leur gestion et leur partage entre les membres de l'équipe de développement.

GitLab permet également de créer des pipelines d'intégration continue (CI) et de déploiement continu (CD) pour les projets. Les pipelines sont des séquences d'actions automatisées qui permettent de tester, de compiler, de construire et de déployer des applications de manière continue. Les pipelines sont déclenchés automatiquement lorsqu'un développeur pousse une modification sur une branche spécifique du dépôt Git. Les pipelines permettent également d'automatiser le déploiement des applications sur différents environnements (par exemple, de l'environnement de développement à l'environnement de production).

L'utilisation de GitLab dans le cadre de Docker et du développement continu et de l'intégration continue permet aux équipes de développement de gagner du temps en automatisant les tâches répétitives, en réduisant les erreurs humaines et en améliorant la qualité du code. En outre, cela facilite la collaboration entre les membres de l'équipe, en permettant à chacun de travailler sur le même code, de partager des commentaires et de surveiller l'avancement du projet.

### Le Pipeline Gitlab

Les pipelines GitLab pour Docker fonctionnent en utilisant les fichiers de configuration de pipeline GitLab (`.gitlab-ci.yml`) et en combinant l'utilisation de GitLab Runner avec des images Docker.

## Formation Docker

Voici les étapes générales pour créer un pipeline GitLab pour Docker :

Créer un fichier ``.gitlab-ci.yml`` à la racine de votre projet GitLab. Ce fichier contiendra les instructions pour le pipeline de construction et de déploiement de votre application Docker.

Configurer les étapes du pipeline, qui peuvent inclure : la récupération du code source, la construction de l'image Docker, les tests unitaires, les tests d'intégration, la livraison de l'image Docker sur un registre d'images, le déploiement sur un environnement de test, etc.

Utiliser GitLab Runner pour exécuter les étapes du pipeline. GitLab Runner est un agent d'exécution de pipeline qui peut être exécuté localement sur un serveur ou dans le cloud.

Utiliser des images Docker dans le pipeline pour fournir un environnement d'exécution cohérent pour les étapes de construction, de test et de déploiement de l'application.

Utiliser un registre d'images Docker pour stocker les images Docker construites dans le pipeline. GitLab Container Registry est un registre d'images Docker intégré à GitLab qui peut être utilisé pour stocker et gérer des images Docker privées et publiques.

Configurer les variables d'environnement pour le pipeline, qui peuvent inclure des informations d'authentification pour le registre d'images Docker, des variables de configuration pour l'application, etc.

Une fois que le pipeline GitLab pour Docker est configuré, il peut être utilisé pour automatiser le processus de construction, de test et de déploiement de votre application Docker de manière continue. Le pipeline sera déclenché automatiquement chaque fois qu'un développeur pousse des modifications sur une branche spécifique de votre dépôt GitLab.

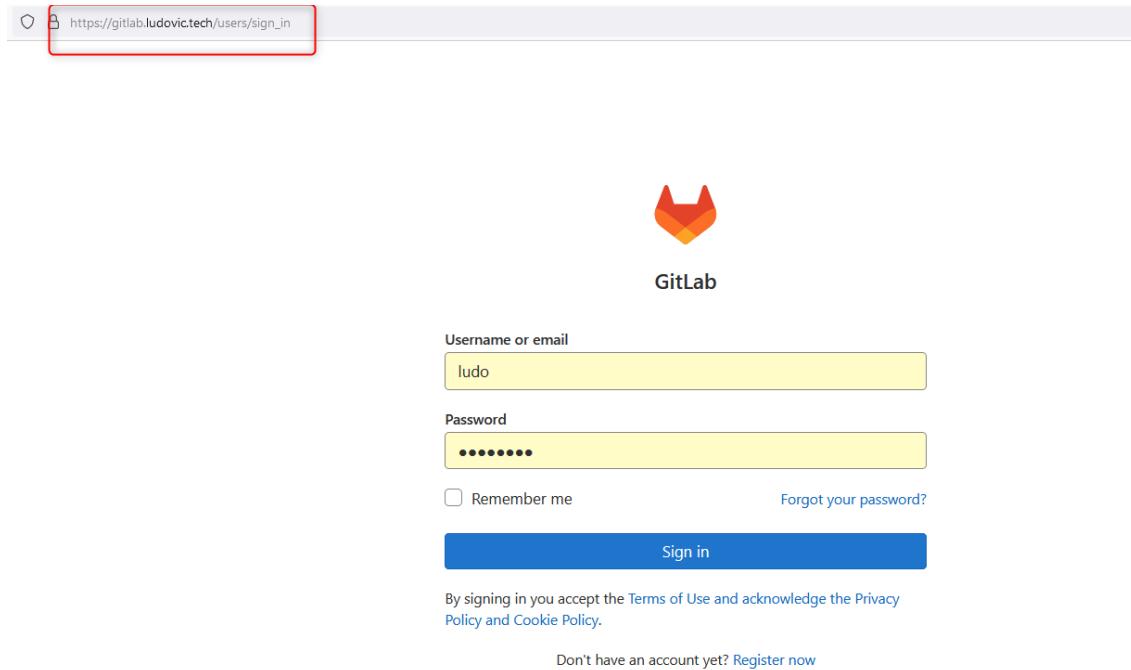
## Formation Docker

TP : Gitlab CI

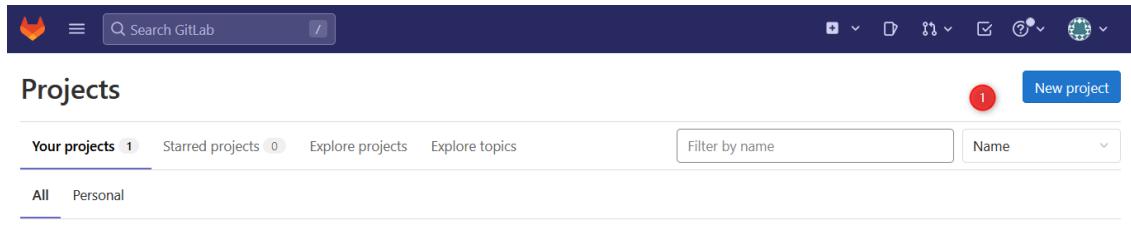
Se rendre sur le serveur Gitlab <https://gitlab.ludovic.tech>

Login : prénom

Mot de passe : rootroot (à changer à la première connexion)



Créer un nouveau projet pour Docker



## Formation Docker

The screenshot shows the 'Create blank project' page on GitLab. At the top, there's a navigation bar with a search field 'Search GitLab'. Below it, a large 'Create blank project' button with a plus sign is visible. The main form has the following fields:

- Project name:** A text input field containing 'c-cd-docker' with a red '1' badge above it.
- Project URL:** A text input field containing 'https://gitlab.ludovic.tech/ludo/' followed by a '/' character.
- Project slug:** A text input field containing 'c-cd-docker'.
- Visibility Level:** A section with three radio button options:
  - Private: Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.
  - Internal: The project can be accessed by any logged in user except external users.
  - Public: The project can be accessed without any authentication.
- Project Configuration:** A section with two checkboxes:
  - Initialize repository with a README: Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.
  - Enable Static Application Security Testing (SAST): Analyze your source code for known security vulnerabilities. [Learn more](#).
- Buttons:** A red '2' badge is above the 'Create project' button, which is highlighted in blue, and a 'Cancel' button.

## Formation Docker

### Cloner le projet :

The screenshot shows a GitLab project page for 'c-cd-docker'. At the top, a message says 'Project 'c-cd-docker' was successfully created.' Below it, the project details are shown: 1 Commit, 1 Branch, 0 Tags, and 20 KB Project Storage. A sidebar on the left contains various icons for repository management. At the top right, there are buttons for 'Find file', 'Web IDE', 'Download', and 'Clone'. A dropdown menu titled 'Clone with HTTPS' is open, showing the URL 'https://gitlab.ludovic.tech/ludo/c-cd-docker'. This URL is highlighted with a red box. Below the URL, there are options to 'Open in your IDE' with links for Visual Studio Code (SSH) and (HTTPS), and IntelliJ IDEA (SSH) and (HTTPS). A 'Copy URL' button is also present.

### Installer git au besoin :

```
[ludo@dockerhost-ludo ~]$ su - ①
Password:
Last login: Tue May 16 16:20:10 CEST 2023 on pts/0
[root@dockerhost-ludo ~]# dnf install -y -q git ②

Upgraded:
  git-2.31.1-3.el8_7.x86_64 git-core-2.31.1-3.el8_7.x86_64 git-core-doc-2.31.1-3.el8_7.noa
[root@dockerhost-ludo ~]# |
```

## Formation Docker

Cloner le projet :

```
[root@dockerhost-ludo ~]# git clone https://gitlab.ludovic.tech/ludo/c-cd-docker.git ①
Cloning into 'c-cd-docker'...
Username for 'https://gitlab.ludovic.tech': ludo
Password for 'https://ludo@gitlab.ludovic.tech':
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
[root@dockerhost-ludo ~]# cd c-cd-docker/ ②
[root@dockerhost-ludo c-cd-docker]# ls
README.md
[root@dockerhost-ludo c-cd-docker]# |
```

Créer un Dockerfile et un fichier index.html :

```
FROM httpd:2.4-alpine

#Custom changes
RUN apk update && apk upgrade
RUN apk -q add curl vim libcap
RUN addgroup toto && adduser -D -H -G toto toto
#Change access rights to conf, logs, bin from root to www-data
RUN chown -hR toto:toto /usr/local/apache2/

#setcap to bind to privileged ports as non-root
RUN setcap 'cap_net_bind_service=+ep' /usr/local/apache2/bin/httpd

#Run as a www-data
USER toto
```

Créer un fichier .gitlab-ci.yml :

```
image: docker:20-dind
services:
  - name: docker:20-dind
    command: ["--tls=false"]

stages:
- build

build:
  stage: build
  tags:
```

## Formation Docker

```
- formation
before_script:
- echo "build image docker sur le registry $HARBOR_HOST"
- docker login -u $HARBOR_USERNAME -p $HARBOR_PASSWORD $HARBOR_HOST
script:
- HARBOR_PROJECT=formation
- docker build -t $HARBOR_HOST/$HARBOR_PROJECT/from-gitlab:latest .
- docker push $HARBOR_HOST/$HARBOR_PROJECT/from-gitlab:latest
```

Ajout du registre Harbor :

The screenshot shows the GitLab interface for a project named 'c-cd-docker'. The left sidebar is open, showing various project management options like Project information, Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Packages and registries, Infrastructure, Monitor, Analytics, Wiki, Snippets, and Settings (with a red notification badge '1'). A dropdown menu is open over the 'Settings' option, with 'Integrations' selected (also indicated by a red '2'). The main content area displays basic project stats: 1 Commit, 1 Branch, 0 Tags, and 31 KB in size. Below this, there's a commit history with one entry from 'ludo' 16 minutes ago, titled 'Initial commit'. The repository structure shows a single file 'README.md'. At the bottom, there's a 'Getting started' section.

## Formation Docker

### Integrations

Integrations enable you to make third-party applications part of your GitLab workflow. If the available integrations don't meet your needs, consider using a webhook.

#### Active integrations

Integration	Description	Last updated
✓ Harbor	Use Harbor as this project's container registry.	17 minutes ago

#### Add an integration

After the Harbor integration is activated, global variables '\$HARBOR\_USERNAME', '\$HARBOR\_HOST', '\$HARBOR\_OCI', '\$HARBOR\_PASSWORD', '\$HARBOR\_URL' and '\$HARBOR\_PROJECT' will be created for CI/CD use.

#### Enable integration

Active

#### Harbor URL

`https://registry.ludovic.tech`

Base URL of the Harbor instance.

#### Harbor project name

formation

The name of the project in Harbor.

#### Harbor username

user

#### Enter new Harbor password

\*\*\*\*\*

Leave blank to use your current password.

**Save changes**

**Test settings**

**Cancel**

## Formation Docker

Ajout d'un token :

The screenshot shows the GitLab User Settings interface. The left sidebar is titled "User Settings" and includes options like Profile, Account, Applications, Chat, Access Tokens (highlighted with a red circle), Emails, Password, Notifications, SSH Keys, GPG Keys, Preferences, Active Sessions, Authentication log, and Usage Quotas. The main content area is titled "Personal Access Tokens" and contains instructions about generating tokens for applications. It has fields for "Token name" (with a red circle), "Expiration date" (set to 2023-06-16), and "Select scopes". The "Select scopes" section lists several options with descriptions: "api", "read\_api", "read\_user", "read\_repository", "write\_repository", "read\_registry", and "write\_registry". At the bottom are "Create personal access token" and "Active personal access tokens (1)". The top right shows a user profile for "ludo @ludo" with a notification count of 1, and navigation links for Set status, Edit profile, Preferences (with a red circle), and Sign out.

## Formation Docker

### Token name

 1

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

### Expiration date

### Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

**api**

Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

**read\_api**

Grants read access to the API, including all groups and projects, the container registry, and the package registry.

**read\_user**

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

**read\_repository**

Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.

**write\_repository**

Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

**read\_registry**

Grants read-only access to container registry images on private projects.

**write\_registry**

Grants write access to container registry images on private projects.

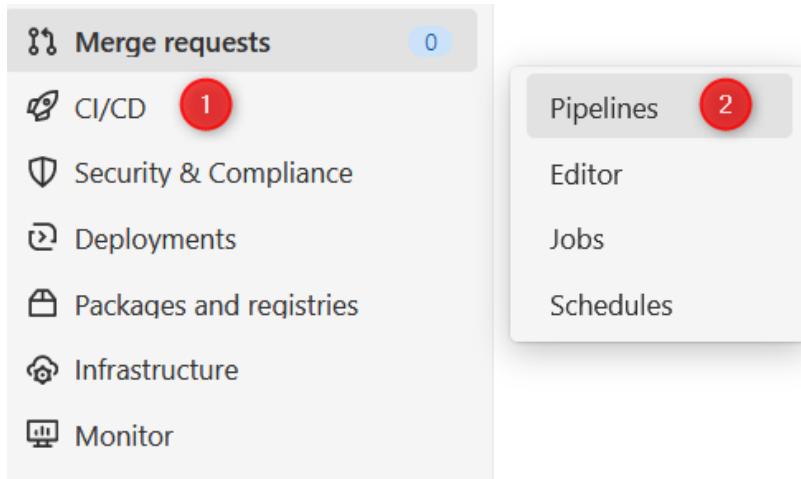
 2

### Your new personal access token

 1

Make sure you save it - you won't be able to access it again.

## Formation Docker



The screenshot shows the GitLab Pipelines interface for the project "c-cd-docker".

Left sidebar (Project Navigation):

- Project information
- Repository
- Issues (0)
- Merge requests (0)
- CI/CD
  - Pipelines (selected)
  - Editor
  - Jobs

Top navigation bar:

- Iudo > c-cd-docker > Pipelines
- All (2)
- Finished
- Branches
- Tags
- Clear runner caches
- CI lint
- Run pipeline

Search bar: Filter pipelines

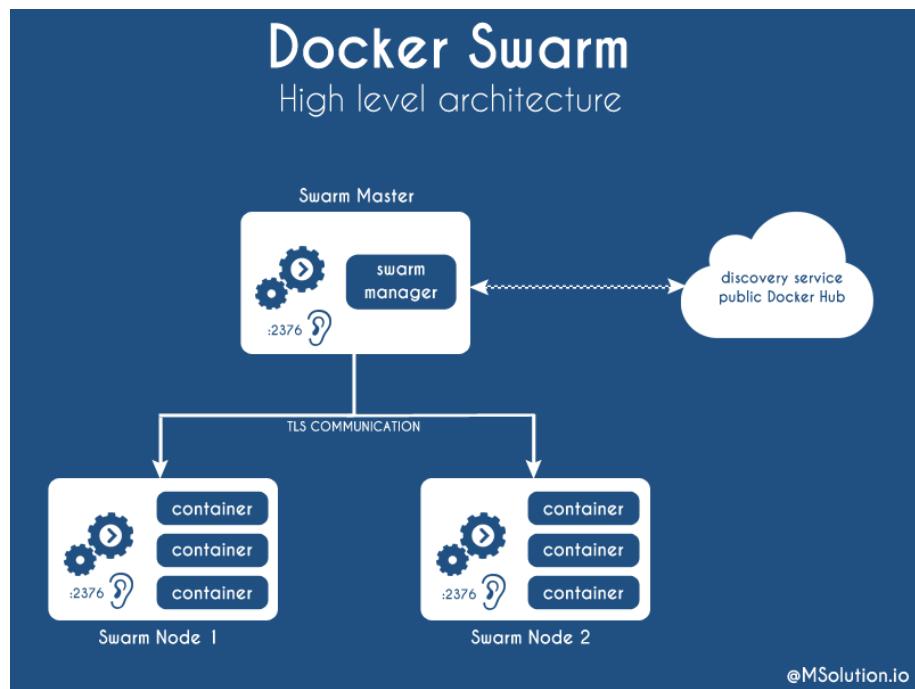
Table header:

Status	Pipeline	Triggerer	Stages	Actions
--------	----------	-----------	--------	---------

Table data:

<span>running</span>	Add new file #42 ➔ main ➔ 6659de77 [latest]	Git	CI	<span>Q</span> <span>↻</span> <span>▼</span>
----------------------	---	-----	----	--

# 1-Présentation de Swarm



## C'est quoi un Docker Swarm ? Manager Swarm ? Les nœuds ? Les workers ?

Un Swarm est un groupe de machines exécutant le moteur Docker et faisant partie du même cluster. Docker swarm vous permet de lancer des commandes Docker auxquelles vous êtes habitué sur un cluster depuis une machine maître nommée manager/leader Swarm. Quand des machines rejoignent un Swarm, elles sont appelés nœuds.

Les managers Swarm sont les seules machines du Swarm qui peuvent exécuter des commandes Docker ou autoriser d'autres machines à se joindre au Swarm en tant que workers. Les workers ne sont là que pour fournir de la capacité et n'ont pas le pouvoir d'ordonner à une autre machine ce qu'elle peut ou ne peut pas faire.

Jusqu'à présent, vous utilisez Docker en mode hôte unique sur votre ordinateur local. Mais Docker peut également être basculé en mode swarm permettant ainsi l'utilisation des commandes liées au Swarm. L'activation du mode Swarm sur hôte Docker fait instantanément de la machine actuelle un manager Swarm. À partir de ce moment, Docker exécute les commandes que vous exécutez sur le Swarm que vous gérez, plutôt que sur la seule machine en cours.

## C'est quoi un service ? une task (tâche) ?

Dans le vocabulaire Swarm nous ne parlons plus vraiment de conteneurs mais plutôt de services.

Un service n'est rien d'autre qu'une description de l'état souhaité pour vos conteneurs. Une fois le service lancé, une tâche est alors attribuée à chaque nœud afin d'effectuer le travail demandé par le service.

Nous verrons plus loin les détails de ces notions, mais histoire d'avoir une idée sur la différence entre une tâche et un service, nous allons alors imaginer l'exemple suivant :

Une entreprise vous demande de déployer des conteneurs d'applications web. Avant toute chose, vous allez commencer par définir les caractéristiques et les états de votre conteneur, comme par exemple :

- Trois conteneurs minimums par application afin de supporter des grandes charges de travail
- Une utilisation maximale de 100 Mo de mémoire pour chaque conteneur
- les conteneurs se baseront sur l'image httpd
- Le port 8080 sera mappé sur le port 80
- Redémarrer automatiquement les conteneurs s'ils se ferment suite à une erreur

Pour le moment vous avez défini l'état et les comportements de vos conteneurs dans votre service, par la suite quand vous exécuterez votre service, chaque nœud se verra attribuer alors une ou plusieurs tâches jusqu'à satisfaire les besoins définit par votre service.

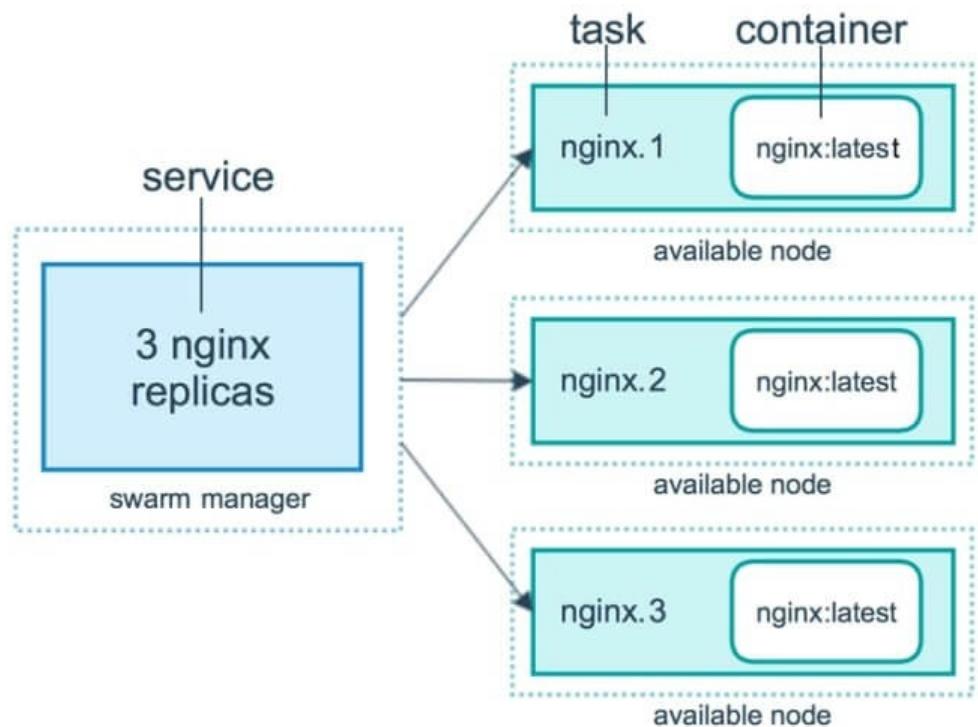
## Résumons les différents concepts de Docker Swarm

Il est important au préalable de bien comprendre la définition de chaque concept afin d'assimiler facilement le fonctionnement global de Docker Swarm.

Je vais donc vous résumer toutes ces notions à travers un seul exemple :

Imaginez que vous êtes embauché en tant qu'ingénieur système dans une start-up. Les développeurs de l'entreprise décident un jour de vous fournir les sources de leur application web tournant sous le serveur web nginx. Votre chef de projet vous demande alors de lui expliquer le process de déploiement de l'application web dans un Cluster Swarm.

- Un service sera créé dans lequel nous spécifierons qu'elle sera image utilisée et nous estimerons quelles seront les charges de travail suffisantes pour les conteneurs qui seront en cours d'exécution.
- La demande sera ensuite envoyée au manager Swarm (leader) qui planifie l'exécution du service sur des nœuds particuliers.
- Chaque nœud se voit assigné une ou plusieurs tâche(s) en fonction de l'échelle qui a été définie dans le service.
- Chaque tâche aura un cycle de vie, avec des états comme NEW , PENDING et COMPLÈTE.
- Un équilibrer de charge sera mis en place automatiquement par votre manager Swarm afin de supporter les grandes charges de travaux.



## Formation Docker avancée - PLB Consultant

### Travaux pratique – Docker Swarm. Les nodes

#### Initialisation du cluster

```
[root@docker-leader ~]# docker swarm init  
Swarm initialized: current node (rm1uucgx32n7b95vo2x2ux59v) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-4yw5c7j5n9us1xgkf3qbcynactr7afm84ytr65l1o6hdx1p9eb-df5ofa12wju99umkk5oehk6so 10.0.0.201:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

#### Ajout des nœuds dans le cluster

```
[root@docker-leader ~]# ssh worker10 docker swarm join --token SWMTKN-1-4yw5c7j5n9us1xgkf3qbcynactr7afm84ytr65l1o6hdx1p9eb-df5ofa12wju99umkk5oehk6so 10.0.0.201:2377  
  
[root@docker-leader ~]# ssh worker20 docker swarm join --token SWMTKN-1-4yw5c7j5n9us1xgkf3qbcynactr7afm84ytr65l1o6hdx1p9eb-df5ofa12wju99umkk5oehk6so 10.0.0.201:2377
```

#### Lister les nœuds du cluster

```
[root@docker-leader ~]# docker node ls  
ID          HOSTNAME      STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION  
rm1uucgx32n7b95vo2x2ux59v *  docker-leader  Ready   Active     Leader        20.10.8  
ks1306m24xlc0xhatoczmti0o  docker-worker10 Ready   Active           20.10.8  
pbmv1t9rgo4g6xgjj68or409  docker-worker20 Ready   Active           20.10.8
```

#### Promouvoir un nœud comme leader potentiel

```
[root@docker-leader ~]# docker node ls  
ID          HOSTNAME      STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION  
rm1uucgx32n7b95vo2x2ux59v *  docker-leader  Ready   Active     Leader        20.10.8  
ks1306m24xlc0xhatoczmti0o  docker-worker10 Ready   Active     Reachable    20.10.8
```

## Formation Docker avancée - PLB Consultant

### Travaux pratique – Docker Swarm. Les services

#### Notre premier service

```
[root@docker-leader ~]# docker service create --detach --publish published=8080,target=80 --name web --  
container-label web \ dockercloud/hello-world  
o9io7m2qq6cof8y3t00rx0sni
```

#### Lister le service

```
[root@docker-leader ~]# curl 127.0.0.1:8080
```

#### Mise à l'échelle du service

```
[root@docker-leader ~]# docker service scale web=3  
web scaled to 3  
overall progress: 3 out of 3 tasks  
1/3: running [=====>]  
2/3: running [=====>]  
3/3: running [=====>]  
verify: Waiting 2 seconds to verify that tasks are stable...  
verify: Waiting 2 seconds to verify that tasks are stable...  
verify: Service converged
```

```
[root@docker-leader ~]# curl 127.0.0.1:8080  
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome to nginx!</title>  
...
```

#### Mise jour du service. Déplacer sur les workers le service

```
[root@docker-leader ~]# docker service update --constraint-add node.role==worker web  
web  
overall progress: 3 out of 3 tasks  
1/3: running [=====>]
```

## Formation Docker avancée - PLB Consultant

```
2/3: running [=====>]
3/3: running [=====>]
verify: Service converged

[root@docker-leader ~]# docker service ps web
ID      NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE      ERROR      PORTS
uvofr7iwrrkh  web.1      nginx:latest  docker-worker20  Running      Running 19 seconds ago
6embn208ed8l  \_web.1    nginx:latest  docker-worker10  Shutdown     Shutdown 20 seconds ago
aoatuhumze7h  web.2      nginx:latest  docker-worker20  Running      Running 4 minutes ago
mm37p3vvcgkm  web.3      nginx:latest  docker-worker20  Running      Running 15 seconds ago
sz3mogg9h24w  \_web.3    nginx:latest  docker-leader    Shutdown     Shutdown 16 seconds ago

[root@docker-leader ~]# curl 127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Suppression du service

```
[root@docker-leader ~]# docker service rm web
ID      HOSTNAME      STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION
ks1306m24xlc0xhatoczmti0o  docker-worker10  Ready  Active  Reachable  20.10.8
```

## Formation Docker avancée - PLB Consultant

Filter les nœuds par label

```
[root@docker-leader ~]# mkdir web && cd web
```

```
[root@docker-leader ~]# vim nginx-stack.yml
```

```
version: "3.2"
```

```
services:
```

```
nginx:
```

```
    image: nginx
```

```
    ports:
```

```
        - 80:80
```

```
    deploy:
```

```
        mode: replicated
```

```
        replicas: 3
```

```
        restart_policy:
```

```
            condition: on-failure
```

```
            delay: 30s
```

```
            max_attempts: 3
```

```
            window: 120s
```

On applique le stack dans le swarm

```
[root@docker-leader ~]# docker stack deploy --compose-file nginx-stack.yml nginx
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
ks1306m24xlc0xhatoczmti0o	docker-worker10	Ready	Active	Reachable	20.10.8

Lister les stack

```
[root@docker-leader ~]# docker stack ls
```

NAME	SERVICES	ORCHESTRATOR
nginx-stack	1	Swarm

Plus d'information sur la stack nginx

```
[root@docker-leader ~]# docker stack ps nginx-stack
```

## Formation Docker avancée - PLB Consultant

ID PORTS	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
l6lsrdzq4pqp	nginx-stack_nginx.1	nginx:latest	docker-worker20	Running	Running 15 seconds ago	
k0nywar5aauw	nginx-stack_nginx.2	nginx:latest	docker-leader	Running	Running 15 seconds ago	
stji9dj5lx8s	nginx-stack_nginx.3	nginx:latest	docker-worker10	Running	Running 15 seconds ago	

Filter les nœuds par label

```
[root@docker-leader ~]#curl 127.0.0.1:800  
<!DOCTYPE html>  
...  
<h1>Welcome to nginx!</h1>
```

Mise à l'échelle de la stack

```
[root@docker-leader ~]#docker service scale nginx-stack_nginx=2  
nginx-stack_nginx scaled to 2  
overall progress: 2 out of 2 tasks  
1/2: running [=====>]  
2/2: running [=====>]
```

Filter les nœuds par label

```
[root@docker-leader ~]# docker stack ps nginx-stack  
  
ID      NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE      ERROR  
PORTS  
l6lsrdzq4pqp  nginx-stack_nginx.1  nginx:latest  docker-worker20  Running  Running 3 minutes ago  
k0nywar5aauw  nginx-stack_nginx.2  nginx:latest  docker-leader    Running  Running 3 minutes ago
```

Il n'y a plus de « task » en shutdown

Suppression de la stack nginx

```
[root@docker-leader ~]# docker stack rm nginx-stack  
  
ID      NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE      ERROR  
PORTS  
l6lsrdzq4pqp  nginx-stack_nginx.1  nginx:latest  docker-worker20  Running  Running 3 minutes ago
```

## Formation Docker avancée - PLB Consultant

```
k0nywar5aauw nginx-stack_nginx.2 nginx:latest docker-leader Running     Running 3 minutes ago
```

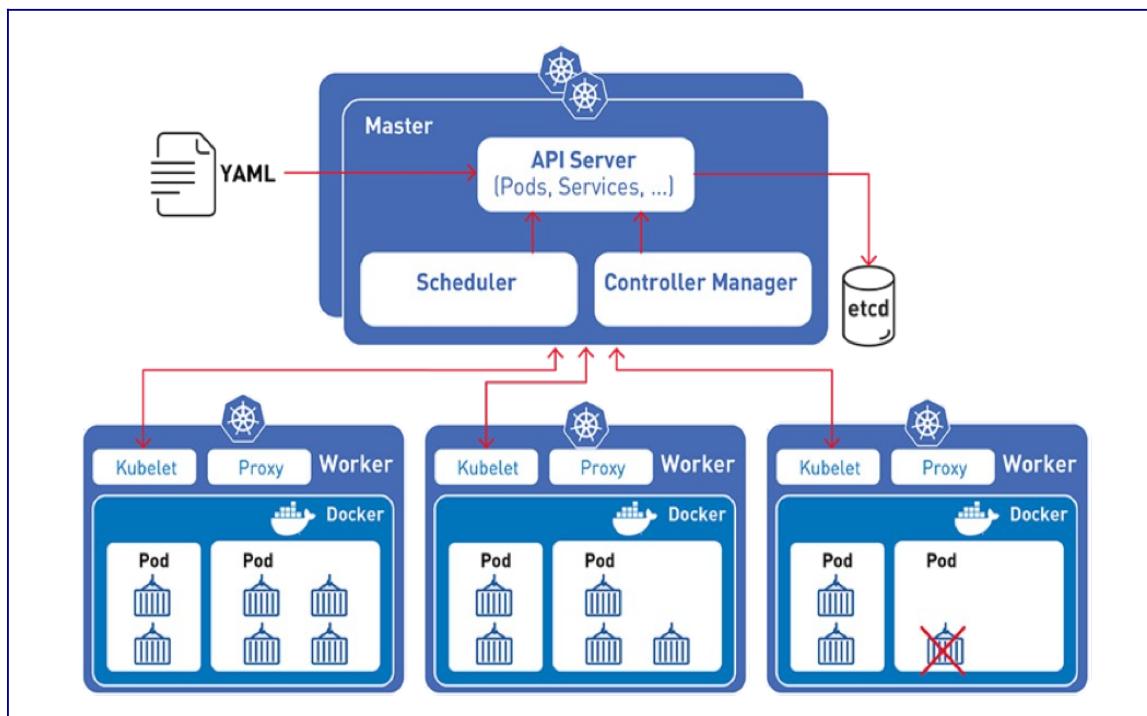
Il n'y a plus de « task » en shutdown

### Travaux pratiques Kubernetes

## 1-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 noeuds(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 déploiements logiciels 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.

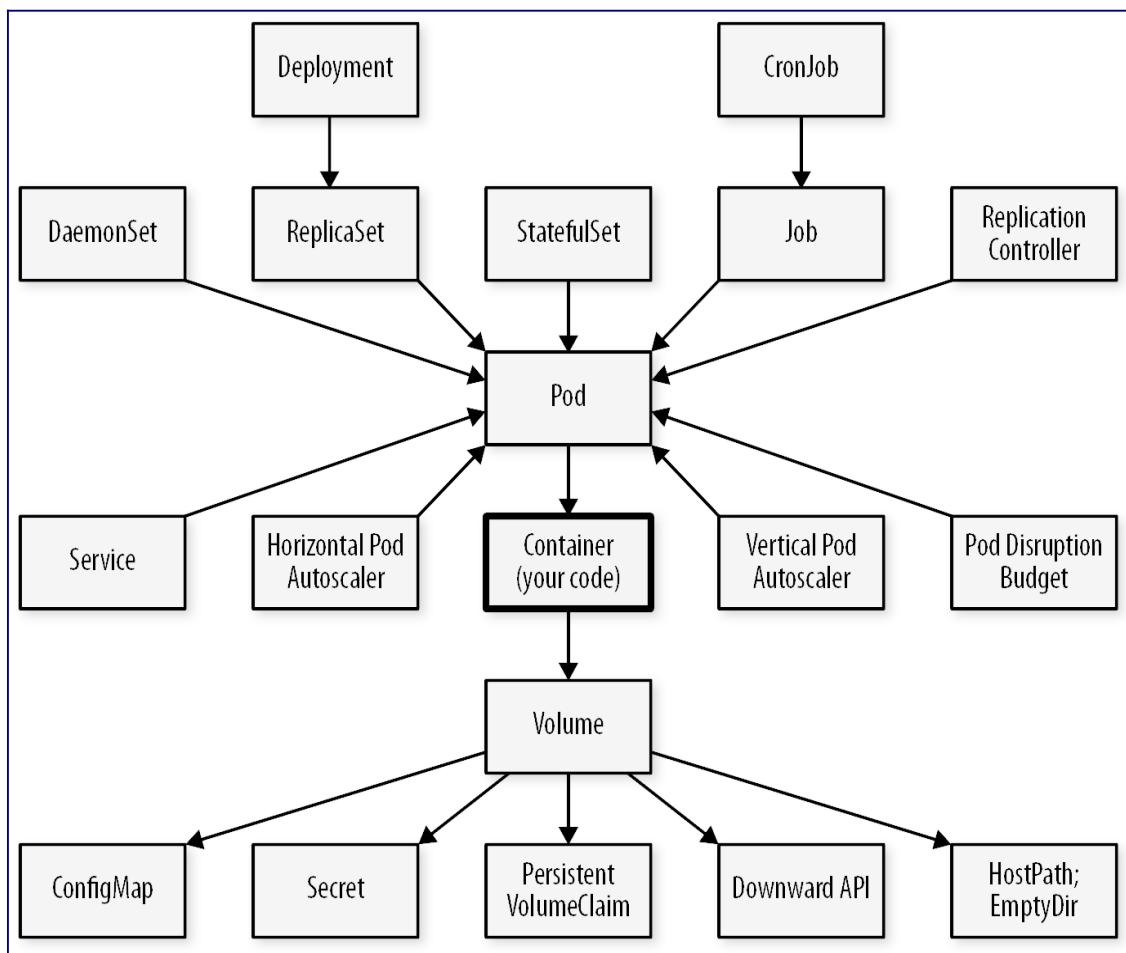
L'écosystème logiciel de Kubernetes s'est développé autour la Cloud Native Computing Foundation qui comprend des acteurs comme notamment : Google, Red Hat, Twitter, Huawei, Intel, Cisco, IBM, VMware, Amazon, Microsoft, etc... .

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 despace 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

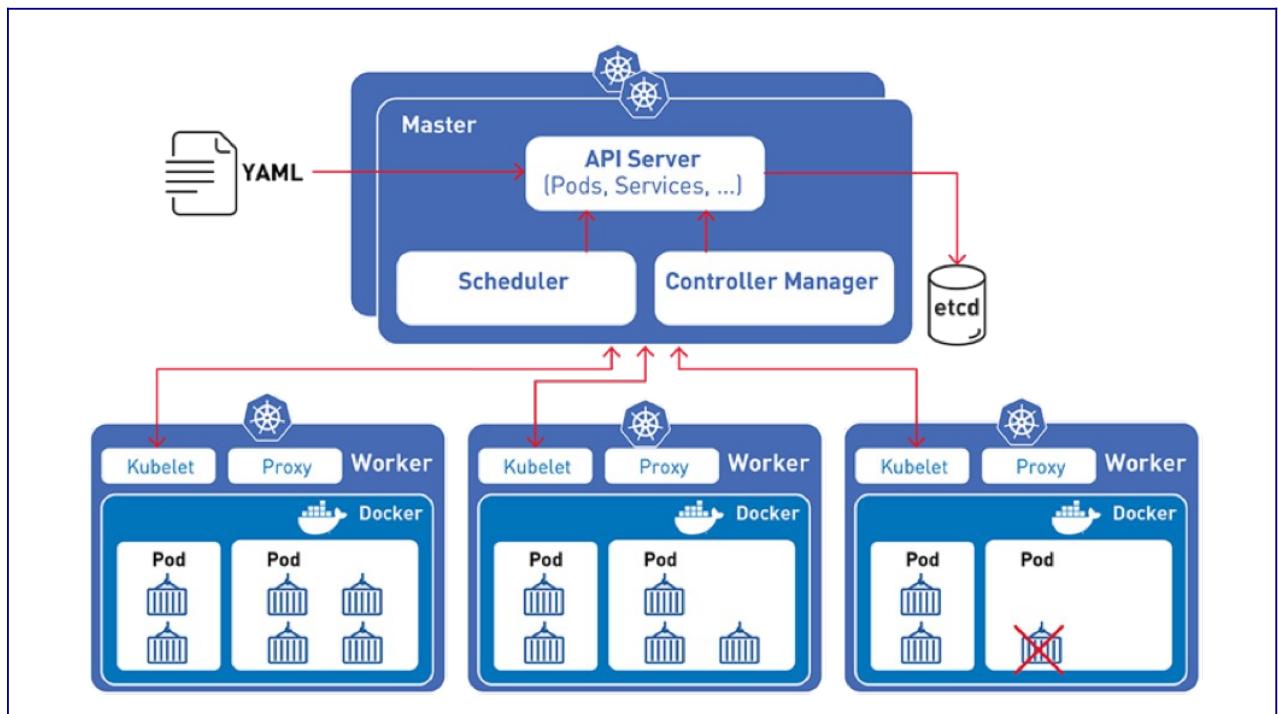
### Architecture de Kubernetes - Partie 1

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 noeud; 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: surveille les ressources des différents workers, décide et cartographie où doivent être ordonnancés les Workloads les conteneurs(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 Plateform 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 daemon 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 :

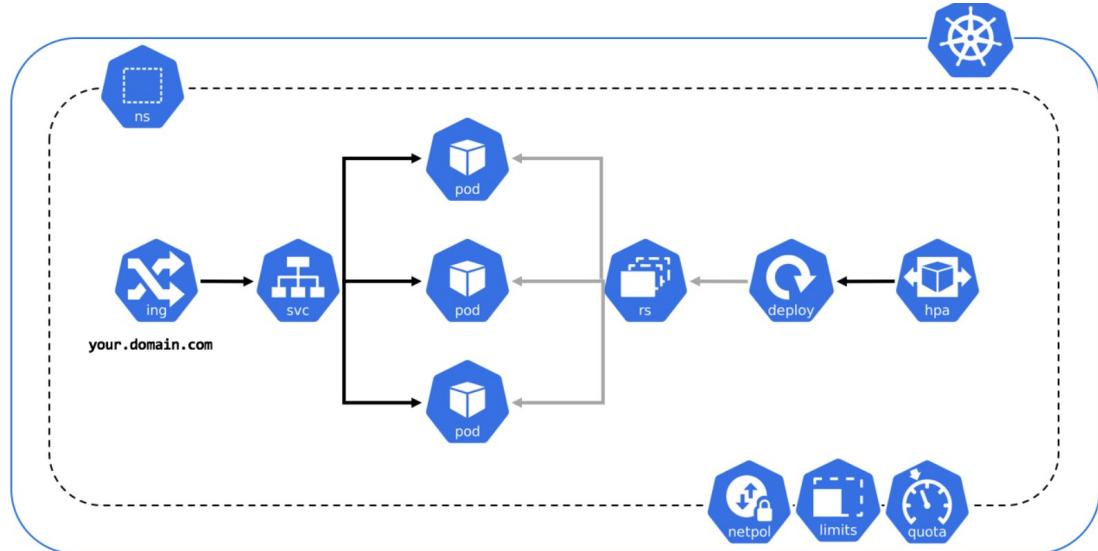
- Afficher les namespaces : kubectl get namespaces

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
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)
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP
deployment.apps/nginx	6/6	6	6	4m18s
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 PORT(S)
AGE
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP
7m4s
service/nginx NodePort 10.109.249.90 <none>
80:32401/TCP 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 67s	NodePort	10.109.249.90	<none>	80:32401/TCP

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-heptioprotip-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
```

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

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

## Les Pods

Un Pod est l'unité d'exécution de base d'une application Kubernetes que vous créez ou déployez. Un Pod représente des process en cours d'exécution dans votre Cluster.

Un Pod encapsule un conteneur (ou souvent plusieurs conteneurs), des ressources de stockage, une IP réseau unique, et des options qui contrôlent comment le ou les conteneurs doivent s'exécuter (ex: *restart policy*). Cette collection de conteneurs et volumes tournent dans le même environnement d'exécution mais les processus sont isolés.

Un Pod représente une unité de déploiement : un petit nombre de conteneurs qui sont étroitement liés et qui partagent :

- les mêmes ressources de calcul - RAM, CPU, ...
- des volumes communs
- la même IP donc le même nom de domaine
- peuvent se parler sur localhost
- peuvent se parler en IPC
- ont un nom différent et des logs différents

Chaque Pod est destiné à exécuter une instance unique d'un workload donné. Si vous désirez mettre à l'échelle votre workload, vous devez multiplier le nombre de Pods avec un déploiement.

Kubernetes fournit un ensemble de commandes pour débugger des conteneurs :

`kubectl logs <pod-name> -c <conteneur_name>` (le nom du conteneur est inutile si un seul)

```
kubectl exec -it <pod-name> -c <conteneur_name> -- bash
```

```
kubectl attach -it <pod-name>
```

Enfin, pour debugger la sortie réseau d'un programme on peut rapidement forwarder un port depuis un pods vers l'extérieur du cluster :

- kubectl port-forward <pod-name> <port\_interne>:<port\_externe>
- C'est une commande de debug seulement : pour exposer correctement des processus k8s, il faut créer un service, par exemple avec NodePort.

Pour copier un fichier dans un pod on peut utiliser: kubectl cp

```
<pod-name>:</path/to/remote/file> </path/to/local/file>
```

Pour monitorer rapidement les ressources consommées par un ensemble de processus il existe les commandes kubectl top nodes et kubectl top pods

## Un manifeste de Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: kuard-ludo
  labels:
    app: demo
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: kuard
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

# Rappel sur quelques concepts

## Haute disponibilité

- Faire en sorte qu'un service ait un "uptime" élevé.

On veut que le service soit tout le temps accessible même lorsque certaines ressources manquent :

- Elles tombent en panne
- Elles sont sorties du service pour mise à jour, maintenance ou modification

Pour cela on doit avoir des ressources multiples...

- Plusieurs serveurs
- Plusieurs versions des données
- Plusieurs accès réseau

Il faut que les ressources disponibles prennent automatiquement le relais des ressources indisponibles. Pour cela on utilise :

- des "load balancers" : aiguillages réseau intelligents
- des "healthchecks" : une vérification de la santé des applications

Nous allons voir que Kubernetes intègre automatiquement les principes de load balancing et de healthcheck dans l'orchestration de conteneurs

## Répartition de charge (load balancing)

- Un load balancer : une sorte d'"aiguillage" de trafic réseau, typiquement HTTP(S) ou TCP.
- Un aiguillage intelligent qui se renseigne sur plusieurs critères avant de choisir la direction.

Cas d'usage :

- Éviter la surcharge : les requêtes sont réparties sur différents backends pour éviter de les saturer.

L'objectif est de permettre la haute disponibilité : on veut que notre service soit toujours disponible, même en période de panne/maintenance.

- Donc on va dupliquer chaque partie de notre service et mettre les différentes instances derrière un load balancer.
- Le load balancer va vérifier pour chaque backend s'il est disponible (healthcheck) avant de rediriger le trafic.
- Répartition géographique : en fonction de la provenance des requêtes on va rediriger vers un datacenter adapté (+ proche).

## Healthchecks

Fournir à l'application une façon d'indiquer qu'elle est disponible, c'est-à-dire :

- Qu'elle est démarrée (*liveness*)
- Qu'elle peut répondre aux requêtes (*readiness*).

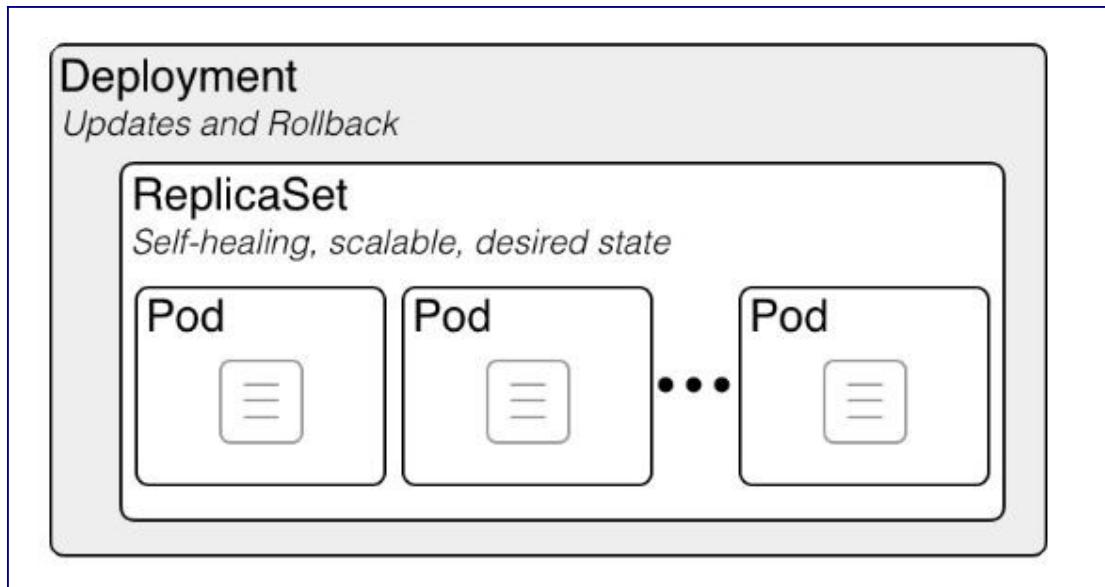
## Application microservices

- Une application composée de nombreux petits services communiquant via le réseau. Le calcul pour répondre à une requête est décomposé en différentes parties distribuées entre les services. Par exemple:
- un service est responsable de la gestion des clients et un autre de la gestion des commandes.
- Ce mode de développement implique souvent des architectures complexes pour être mis en œuvre et Kubernetes est pensé pour faciliter leur gestion à grande échelle.
- Imaginez devoir relancer manuellement des services vitaux pour une application en hébergeant des centaines d'instances : c'est en particulier à ce moment que Kubernetes devient indispensable.

2 exemples d'application microservices:

## Les Deployments (deploy)

Les Deployments sont un objet Kubernetes qui permet de gérer la mise à jour et le déploiement des applications sur un cluster Kubernetes de manière simple et automatisée. Les Deployments peuvent créer et gérer plusieurs répliques d'une application, ainsi que gérer la mise à jour des images et le contrôle des versions de l'application.



Les Deployments sont composés de plusieurs éléments :

- Le nombre de répliques souhaité de l'application
- L'image Docker à utiliser pour l'application
- La stratégie de mise à jour de l'application
- Les paramètres de santé pour la sonde Liveness et la sonde Readiness
- La stratégie de déploiement de l'application (rolling update, blue/green, canary)

Les Deployments permettent également de faire des rollbacks en cas de problèmes lors d'une mise à jour ou d'un déploiement. Les Deployments créent un objet ReplicaSet pour gérer les répliques de l'application et mettent à jour la configuration de ReplicaSet lors de la mise à jour de l'application.

Les Deployments sont une méthode pratique et efficace pour gérer le déploiement des applications sur un cluster Kubernetes. Ils permettent de s'assurer que les applications sont disponibles et à jour en tout temps, tout en évitant les temps d'arrêt pour les utilisateurs.

*Les poupées russes Kubernetes : un Deployment contient un ReplicaSet, qui contient des Pods, qui contiennent des conteneurs*

Exemple :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    kubernetes.io/change-cause: "mise a jour 1.10 vers 1.19"
  labels:
    app: web
    name: apache
spec:
  replicas: 10
  selector:
    matchLabels:
      app: web
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 5
    minReadySeconds: 5
    revisionHistoryLimit: 10
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - image: apache:latest
          name: apache
```

Les Deployments dans Kubernetes sont une façon de déclarer l'état souhaité d'une application et de laisser Kubernetes gérer la mise en place et le maintien de cet état.

Voici quelques détails supplémentaires sur les Deployments :

- Les Deployments gèrent les répliques de l'application à l'aide d'un objet ReplicaSet. Le ReplicaSet est un objet Kubernetes qui permet de créer et de gérer plusieurs répliques d'un pod. Le Deployment crée un ReplicaSet pour chaque version de l'application qui est déployée.
- Les Deployments prennent en charge différents types de stratégies de déploiement, notamment le rolling update, la stratégie blue/green et la stratégie canary. Le rolling update est la stratégie de déploiement par défaut et consiste à mettre à jour les répliques de l'application en continu. La stratégie blue/green consiste à déployer deux versions de l'application simultanément, puis à basculer le trafic vers la nouvelle version une fois qu'elle est prête. La stratégie canary consiste à déployer une nouvelle version de l'application sur un petit nombre de répliques, puis à augmenter progressivement le nombre de répliques pour tester la nouvelle version avant de la déployer complètement.
- Les Deployments peuvent être mis à jour de manière automatique ou manuelle.  
Les mises à jour automatiques sont gérées par Kubernetes, qui surveille en permanence les versions de l'application disponibles et met à jour les répliques de l'application en fonction de la stratégie de mise à jour définie. Les mises à jour manuelles sont effectuées par un administrateur Kubernetes, qui met à jour manuellement la configuration du Deployment pour déployer une nouvelle version de l'application.
- Les Deployments permettent de faire des rollbacks en cas de problème lors d'une mise à jour ou d'un déploiement. Les rollbacks sont effectués en

modifiant la configuration du Deployment pour revenir à la version précédente de l'application.

- Les Deployments peuvent être surveillés à l'aide de différentes métriques, telles que le nombre de répliques actives, le nombre de mises à jour en cours et le nombre de mises à jour réussies ou échouées.

En résumé, les Deployments dans Kubernetes permettent de gérer le déploiement et la mise à jour des applications de manière automatisée et contrôlée, tout en offrant des options flexibles pour la gestion des répliques et la gestion des mises à jour.

## Les ReplicaSets (rs)

Dans notre modèle, les ReplicaSet servent à gérer et sont responsables de :

- la réplication (avoir le bon nombre d'instances et le scaling)
- la santé et le redémarrage automatique des pods de l'application (Self-Healing)
- kubectl get rs pour afficher la liste des replicas.
- 

En général on ne les manipule pas directement , même s'il est possible de les modifier et de les créer avec un fichier de ressource. Pour créer des groupes de conteneurs on utilise soit un Deployment soit d'autres formes de workloads (DaemonSet, StatefulSet, Job) adaptés à d'autres cas. Le réseau dans Kubernetes

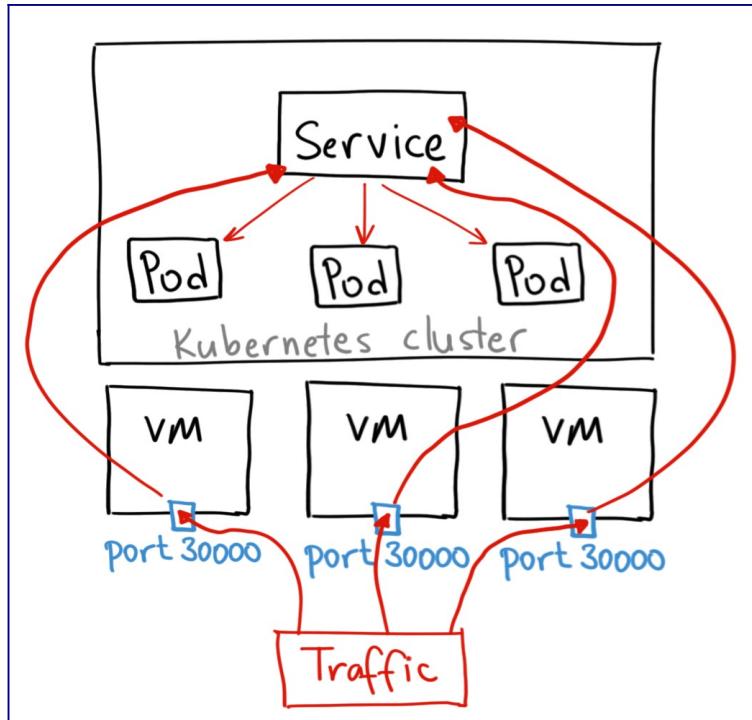
# Le réseau dans Kubernetes

Les solutions réseau dans Kubernetes ne sont pas standard. Il existe plusieurs façons d'implémenter le réseau.

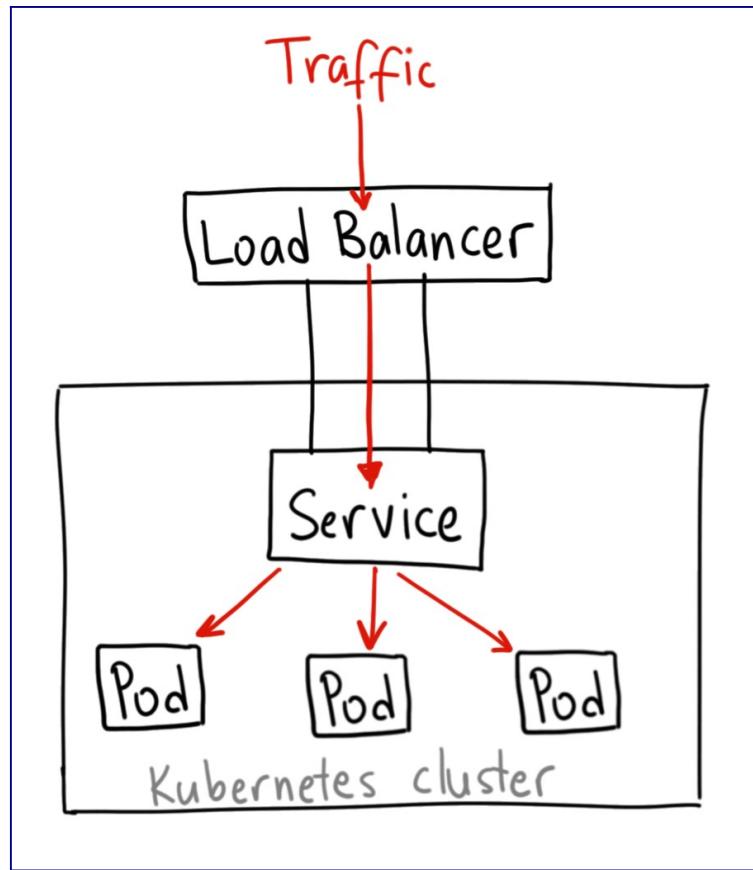
## les objets Services

Les Services sont de trois types principaux :

- ClusterIP: expose le service sur une IP interne au cluster appelée ClusterIP. Les autres pods peuvent alors accéder au service mais pas l'extérieur.
- NodePort: expose le service depuis l'IP publique de chacun des noeuds du cluster en ouvrant port directement sur le nœud, entre 30000 et 32767. Cela permet d'accéder aux pods internes répliqués. Comme l'IP est stable on peut faire pointer un DNS ou Loadbalancer classique dessus. Dans la pratique, on utilise très peu ce type de service.



- LoadBalancer: expose le service en externe à l'aide d'un Loadbalancer de fournisseur de cloud. Les services NodePort et ClusterIP, vers lesquels le Loadbalancer est dirigé sont automatiquement créés.
  - Dans la pratique, on utilise que ponctuellement ce type de service, pour du HTTP/s on ne va pas exposer notre service (ce sera un service de type ClusterIP) et on va utiliser à la place un objet Ingress (voir ci-dessous).

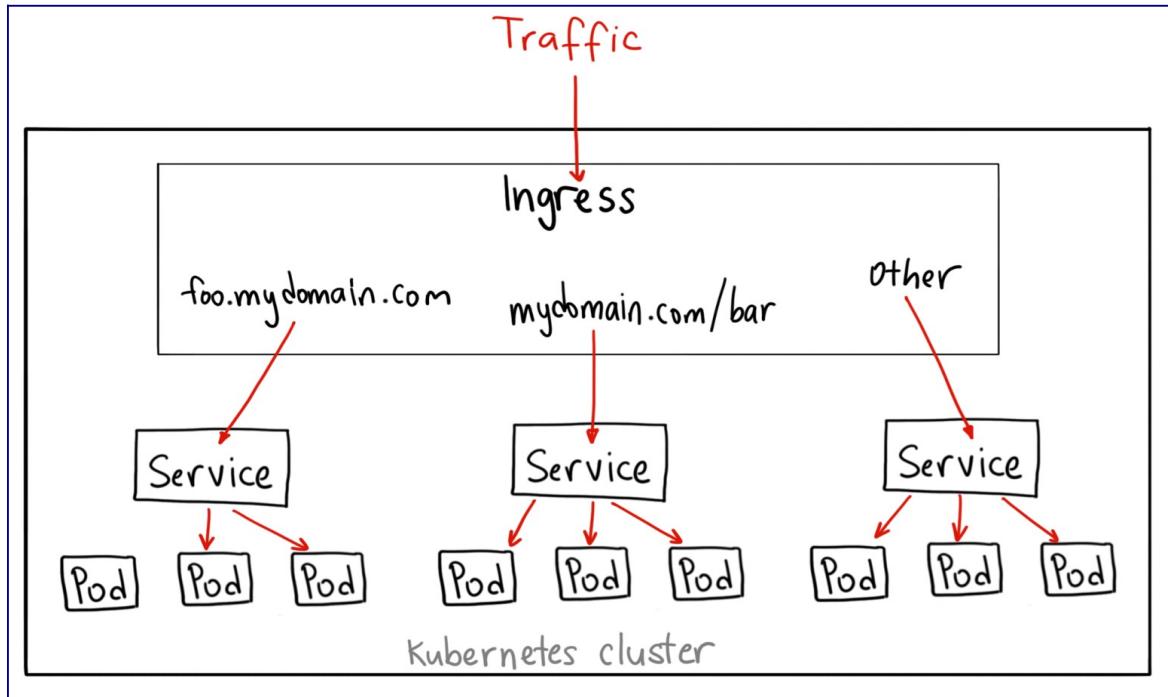


### Fournir des services LoadBalancer on premise avec MetalLB

Dans un cluster managé provenant d'un fournisseur de cloud, la création d'un objet Service Lodbalancer entraîne le provisionnement d'une nouvelle machine de loadbalancing à l'extérieur du cluster avec une IPv4 publique grâce à l'offre d'IaaS du provideur (impliquant des frais supplémentaires).

Cette intégration n'existe pas par défaut dans les clusters de dev comme minikube ou les cluster on premise (le service restera pending et fonctionnera comme un NodePort). Le projet [MetalLB](#) cherche à y remédier en vous permettant d'installer un loadbalancer directement dans votre cluster en utilisant une connexion IP classique ou BGP pour la haute disponibilité.

## Les objets Ingresses



Un Ingress est un objet pour gérer dynamiquement le reverse proxy  
HTTP/HTTPS dans Kubernetes

Exemple de syntaxe d'un ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-php
  namespace: ludo
spec:
  ingressClassName: nginx
  rules:
    - host: ludo-php.ludovic.tech
      http:
        paths:
          - backend:
              service:
                name: nginx
                port:
                  number: 80
            path: /
            pathType: Prefix
```

Ce manifeste Kubernetes définit une ressource Ingress, qui est utilisée pour gérer l'accès aux services HTTP/S d'un cluster Kubernetes depuis l'extérieur du cluster.

Voici une explication de chaque partie du manifeste :

- **apiVersion**: spécifie la version de l'API Kubernetes utilisée pour la ressource Ingress, dans ce cas, `networking.k8s.io/v1`.
- **kind**: spécifie le type de ressource, dans ce cas, `Ingress`.
- **metadata**: spécifie des métadonnées pour la ressource, telles que le nom et l'espace Kubernetes dans lequel la ressource est créée.
- **name**: le nom de la ressource, qui est défini sur `nginx-php`.
- **namespace**: l'espace Kubernetes dans lequel la ressource est créée, dans ce cas, `ludo`.
- **spec**: la spécification de l'Ingress qui définit les règles pour l'accès HTTP/S.
- **ingressClassName**: le nom de la classe Ingress que le contrôleur Ingress doit utiliser pour cette règle d'accès. Dans ce cas, la classe est `nginx`.
- **rules**: la liste des règles pour accéder aux services HTTP/S.
- **host**: l'hôte pour lequel cette règle s'applique, dans ce cas, `ludo-php.ludovic.tech`.
- **http**: les règles HTTP pour l'hôte spécifié.
- **paths**: les règles de chemin d'accès pour l'hôte et les règles HTTP.
- **backend**: le backend pour la règle de chemin d'accès.
- **service**: le service Kubernetes pour le backend, dans ce cas, `nginx`.
- **port**: le port du service pour le backend, dans ce cas, le port 80.

En résumé, ce manifeste Ingress définit une règle pour l'accès HTTP au service `nginx` dans l'espace de noms `ludo`, pour l'hôte `ludo-php.ludovic.tech`. Cela permet à des clients externes d'accéder au service `nginx` en utilisant l'URL `ludo-php.ludovic.tech`.

Pour pouvoir créer des objets ingress il est d'abord nécessaire d'installer un ingress controller dans le cluster:

- Il s'agit d'un déploiement conteneurisé d'un logiciel de reverse proxy (comme nginx, haproxy) et intégré avec l'API de kubernetes
- Le contrôleur agit donc au niveau du protocole HTTP et doit lui-même être exposé (port 80 et 443) à l'extérieur, généralement via un service de type LoadBalancer.
- Le contrôleur redirige ensuite vers différents services (généralement configurés en ClusterIP) qui à leur tour redirigent vers différents ports sur les pods selon l'URL de la requête.

Il existe plusieurs variantes d'ingress controller:

- Un ingress basé sur Nginx plus ou moins officiel à Kubernetes et très utilisé : <https://kubernetes.github.io/ingress-nginx/>
- Un ingress Traefik optimisé pour k8s.
- il en existe d'autres : celui de payant l'entreprise Nginx, Contour, HAProxy...

Chaque provider de cloud et flavour de kubernetes est légèrement différent au niveau de la configuration du contrôleur ce qui peut être déroutant au départ:

- kind permet d'activer l'ingress nginx simplement
- autre exemple: k3s est fourni avec traefik configuré par défaut
- On peut installer plusieurs ingress controllers correspondant à plusieurs IngressClasses

## Le mesh networking et les service meshes

Un service mesh est un type d'outil réseau pour connecter un ensemble de pods, généralement les parties d'une application microservices de façon encore plus intégrée que ne le permet Kubernetes.

En effet opérer une application composée de nombreux services fortement couplés discutant sur le réseau implique des besoins particuliers en termes de routage des requêtes, sécurité et monitoring qui nécessite l'installation d'outils fortement dynamique autour des nos conteneurs.

Un exemple de service mesh est <https://istio.io> qui, en ajoutant en conteneur "sidecar" à chacun des pods à supervisés, ajoute à notre application microservice un ensemble de fonctionnalités d'intégration très puissant.

## CNI (container network interface) : Les implémentations du réseau Kubernetes

Beaucoup de solutions de réseau qui se concurrencent, demandant un comparatif un peu fastidieux.

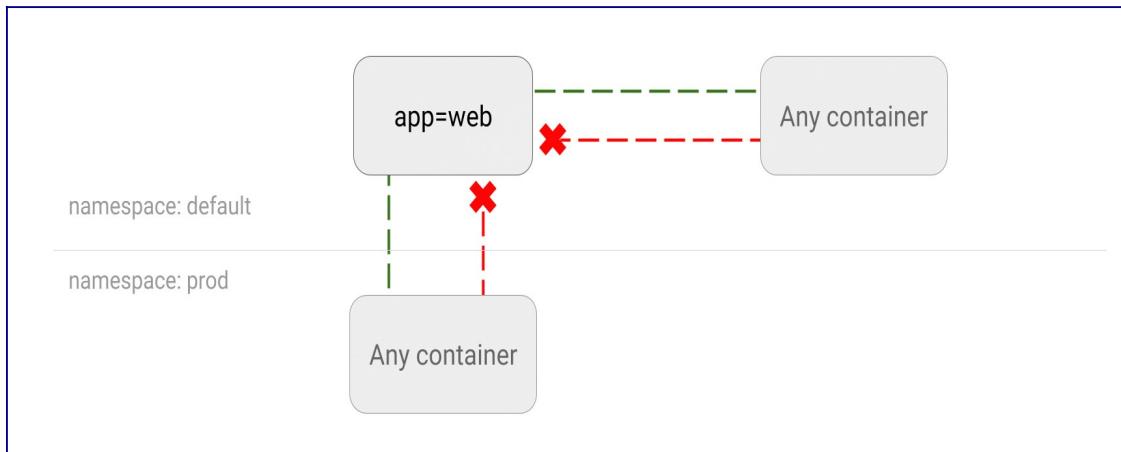
- Plusieurs solutions très robustes
- Diffèrent sur l'implémentation : BGP, réseau overlay ou non (encapsulation VXLAN, IPinIP, autre)
- Toutes ne permettent pas d'appliquer des NetworkPolicies : l'isolement et la sécurité réseau
- Peuvent parfois s'hybrider entre elles (Canal = Calico + Flannel)
- ces implémentations sont souvent concrètement des *DaemonSets* : des pods qui tournent dans chacun des nodes de Kubernetes
- Calico, Flannel, Weave ou Cilium sont très employées et souvent proposées en option par les fournisseurs de cloud
- Cilium a la particularité d'utiliser la technologie eBPF de Linux qui permet une sécurité et une rapidité accrue

Comparaisons :

- <https://www.objectif-libre.com/fr/blog/2018/07/05/comparatif-solutions-reseaux-kubernetes/>

- <https://rancher.com/blog/2019/2019-03-21-comparing-kubernetes-cni-providers-flannel-calico-canal-and-weave/>

## Les network policies : des firewalls dans le cluster



Par défaut, les pods ne sont pas isolés au niveau réseau : ils acceptent le trafic de n'importe quelle source.

Les pods deviennent isolés en ayant une NetworkPolicy qui les sélectionne. Une fois qu'une NetworkPolicy (dans un certain namespace) inclut un pod particulier, ce pod rejettera toutes les connexions qui ne sont pas autorisées par cette NetworkPolicy.

- Des exemples de Network Policies : [Kubernetes Network Policy Recipes](#)

## Ressources sur le réseau

- Documentation officielle :  
<https://kubernetes.io/fr/docs/concepts/services-networking/service/>
- [An introduction to service meshes - DigitalOcean](#)
- [Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?](#)
- [Determine best networking option - Project Calico](#)

- [Doc officielle sur les solutions de networking](#)

## Vidéos

Des vidéos assez complètes sur le réseau, faites par Calico :

- [Kubernetes Ingress networking](#)
- [Kubernetes Services networking](#)
- [Kubernetes networking on Azure](#)

Sur MetalLB, les autres vidéos de la chaîne sont très bien :

- [Why you need to use MetalLB - Adrian Goins](#)

# Le Stockage dans Kubernetes

Kubernetes offre plusieurs options pour la gestion du stockage. Voici un aperçu des principales méthodes de stockage dans Kubernetes :

## Persistance des données

1. **Stockage local** : Kubernetes permet de monter des volumes de stockage locaux sur les nœuds de votre cluster. Cela peut être utile pour stocker des fichiers temporaires ou des données qui ne nécessitent pas une haute disponibilité.
2. **Stockage persistant** : Kubernetes permet de créer des volumes de stockage persistants qui peuvent être utilisés pour stocker des données de manière fiable. Les volumes de stockage persistants sont liés à des pods, ce qui signifie que les données stockées seront disponibles même si le pod est déplacé sur un autre nœud.
3. **Stockage objet** : Kubernetes permet de monter des objets de stockage dans les conteneurs de votre application en utilisant des fournisseurs de stockage objet tels que Amazon S3, Google Cloud Storage ou Microsoft Azure Blob Storage, NFS, Cephfs.
4. **Stockage partagé** : Kubernetes permet également de partager des volumes de stockage entre plusieurs pods, ce qui peut être utile pour les applications qui nécessitent une synchronisation de données entre plusieurs instances.

Pour gérer le stockage dans Kubernetes, vous devez définir des ressources de stockage dans votre configuration de déploiement ou d'application. Vous pouvez utiliser des objets Kubernetes tels que **PersistentVolumeClaim** (PVC),

**PersistentVolume (PV), StorageClass et Volume** pour définir et gérer les ressources de stockage.

Une fois que vous avez défini les ressources de stockage, vous pouvez monter ces volumes dans vos conteneurs en utilisant des points de montage dans votre configuration de déploiement ou d'application

#### **Les PersistentVolumeClaims (PVC) :**

Sont des objets Kubernetes qui permettent aux applications de demander de l'espace de stockage persistant sur un cluster Kubernetes. Les PVC sont utilisés pour réserver un espace de stockage sur un PersistentVolume (PV) préexistant ou en créer un nouveau.

#### **Les PV :**

Sont des ressources de stockage dans le cluster Kubernetes qui représentent des volumes de stockage physiques ou virtuels.

L'utilisation de PVC permet aux développeurs de séparer la définition de l'espace de stockage de la configuration de l'application. Les développeurs peuvent demander un espace de stockage spécifique pour leur application en utilisant des PVC, sans avoir à se soucier de l'emplacement ou de la disponibilité physique du stockage. Les administrateurs du cluster peuvent ensuite assigner ou provisionner un PV pour répondre aux demandes de PVC.

Les PVC peuvent être configurés pour demander des ressources de stockage de différentes manières, notamment en spécifiant la taille de l'espace de stockage, le mode d'accès (lecture/écriture), le mode de réclamation (exclusif ou partagé) et d'autres paramètres avancés.

Voici un exemple simple de YAML pour un objet PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:  
  name: code  
  namespace: ludo  
spec:  
  accessModes:  
    - ReadWriteOnce  
  resources:  
    requests:  
      storage: 1Gi  
  storageClassName: nfs-storage
```

Dans cet exemple, nous créons un PVC appelé "code" avec un accès en lecture et écriture exclusif (ReadWriteOnce) et une demande de stockage de 1 gigaoctet (storage: 1Gi). Ce stockage appelle une classe de stockage : nfs-storage.

Une fois créé, le PVC peut être utilisé dans une configuration de déploiement ou d'application pour monter l'espace de stockage persistant demandé.

### **Une StorageClass :**

Est un objet Kubernetes qui fournit une interface abstraite pour provisionner un espace de stockage persistant dans un cluster Kubernetes. La classe de stockage permet aux administrateurs de cluster de définir des stratégies pour la provision de stockage persistant dans le cluster, telles que le type de stockage utilisé, le niveau de performance et les options de sauvegarde.

Une StorageClass est utilisée lorsqu'un développeur demande un espace de stockage persistant via un PersistentVolumeClaim (PVC). Lorsqu'un PVC est créé, il spécifie la classe de stockage à utiliser pour provisionner l'espace de stockage persistant. Si aucune classe de stockage n'est spécifiée, la classe de stockage par défaut sera utilisée.

La définition d'une StorageClass comprend plusieurs paramètres pour spécifier le type de stockage utilisé, le niveau de performance, les options de sauvegarde

et de restauration, ainsi que d'autres paramètres avancés. Ces paramètres peuvent varier selon le type de stockage utilisé.

Voici un exemple simple de YAML pour une classe de stockage:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: my-provisioner
parameters:
  type: ssd
```

Dans cet exemple, nous définissons une classe de stockage appelée "fast" avec un provisionneur nommé "my-provisioner" et le type de stockage "ssd". Les autres paramètres avancés peuvent être ajoutés selon les besoins.

Une fois la classe de stockage créée, elle peut être utilisée dans les PVC pour provisionner un espace de stockage persistant dans le cluster Kubernetes.

Un développeur peut voir les classes de stockage disponibles dans un cluster Kubernetes en utilisant la commande kubectl.

Pour afficher la liste des classes de stockage dans le cluster, le développeur peut exécuter la commande suivante:

```
kubectl get storageclasses
```

Cela affichera la liste des classes de stockage disponibles, ainsi que des informations telles que le nom, le provisionneur et les paramètres de chaque classe de stockage.

Le développeur peut également voir les détails de chaque classe de stockage en utilisant la commande suivante :

```
kubectl describe storageclass nom_de_la_classe_de_stockage
```

Cela affichera des informations plus détaillées sur la classe de stockage spécifiée, telles que les paramètres, les options de provisionnement et les limites de capacité.

Pour associer un PersistentVolumeClaim (PVC) à un Pod dans Kubernetes, il faut spécifier le PVC dans la configuration du volume du Pod. La configuration de volume est définie dans le fichier de déploiement ou de configuration YAML du Pod.

Voici un exemple simple de configuration de volume dans un fichier YAML pour un Pod qui utilise un PVC:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: my-image
      volumeMounts:
        - name: my-volume
          mountPath: /data
  volumes:
    - name: my-volume
      persistentVolumeClaim:
        claimName: my-pvc
```

Dans cet exemple, nous définissons un Pod avec un conteneur nommé "my-container" et une configuration de volume nommée "my-volume". Le volume est associé à un PVC nommé "my-pvc".

Le champ `volumeMounts` est utilisé pour spécifier comment monter le volume dans le conteneur. Dans cet exemple, nous montons le volume sur le chemin `/data` dans le conteneur.

Le champ `volumes` est utilisé pour définir le volume utilisé par le Pod. Dans cet exemple, nous définissons le volume comme étant un PVC avec le nom "my-pvc". Cela fait référence au PVC que nous avons créé précédemment.

Une fois que la configuration de volume est définie dans le fichier YAML, le Pod peut être créé en utilisant la commande

```
`kubectl create -f <nom_du_fichier_yaml>`.
```

Le Pod sera alors créé avec le volume associé à un PVC spécifique.

En utilisant ces commandes, les développeurs peuvent facilement voir les classes de stockage disponibles dans le cluster et choisir la meilleure classe de stockage pour répondre aux besoins de leur application.

## Les autres types stockage

Les **ConfigMaps** et les **Secrets** sont deux types de ressources Kubernetes utilisées pour stocker des données de configuration et des secrets, respectivement. Voici une brève explication de chaque type de ressource :

### ConfigMap :

Un ConfigMap est une ressource Kubernetes qui stocke des données de configuration sous forme de paires clé-valeur. Ces données peuvent être utilisées par des conteneurs dans un Pod, ou par des applications s'exécutant sur le cluster Kubernetes. Les ConfigMaps peuvent contenir des fichiers de configuration, des chaînes de caractères, des entiers, des adresses IP et d'autres types de données.

Les ConfigMaps peuvent être créés manuellement en utilisant la commande `kubectl create configmap`, ou en utilisant des fichiers YAML. Ils peuvent également être créés automatiquement par des applications s'exécutant sur le cluster Kubernetes. Les ConfigMaps peuvent être montés en tant que volumes dans des conteneurs dans un Pod, ou être utilisés pour définir des variables d'environnement.

### Secret :

Un Secret est une ressource Kubernetes qui stocke des données sensibles, telles que des clés d'API, des mots de passe et des certificats. Les Secrets sont similaires aux ConfigMaps, mais ils sont cryptés pour protéger les données qu'ils contiennent.

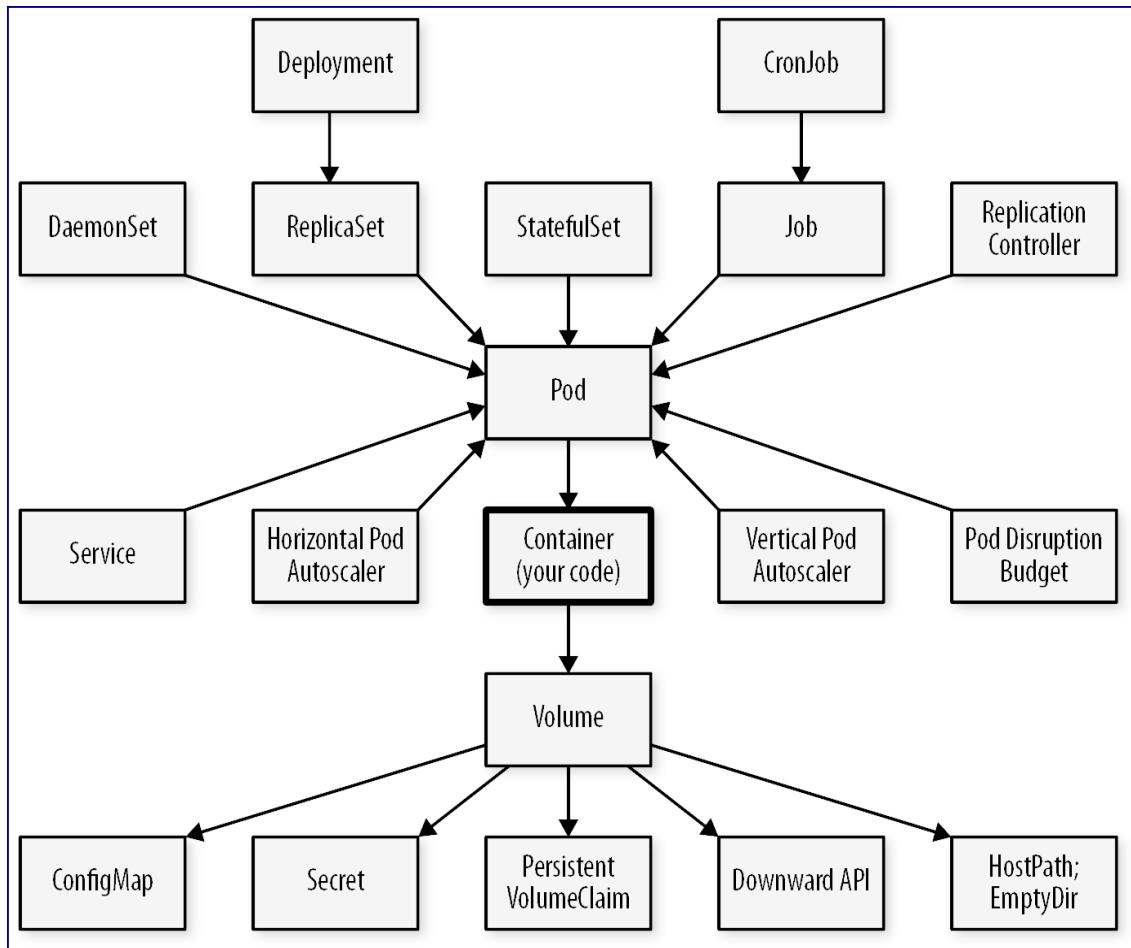
Comme les ConfigMaps, les Secrets peuvent être créés manuellement en utilisant la commande `kubectl create secret`, ou en utilisant des fichiers YAML. Les Secrets peuvent être montés en tant que volumes dans des conteneurs dans un Pod, ou être utilisés pour définir des variables d'environnement.

Les Secrets sont cryptés à l'aide d'un algorithme de chiffrement fort et sont stockés sous forme de fichiers dans un volume secret dans le système de fichiers du nœud hôte Kubernetes. Les Secrets peuvent être utilisés par des applications s'exécutant sur le cluster Kubernetes pour accéder à des ressources sécurisées, telles que des bases de données, des systèmes de fichiers partagés et des services tiers.

En ce qui concerne les Secrets dans Kubernetes, ils sont **chiffrés** plutôt que **cryptés**. Cela signifie que les données sont transformées en utilisant un algorithme de chiffrement pour les protéger contre l'accès non autorisé, mais les données restent accessibles aux applications s'exécutant sur le cluster Kubernetes qui ont l'autorisation appropriée pour y accéder. Les Secrets Kubernetes sont chiffrés à l'aide d'une clé de chiffrement forte, qui est stockée sur le cluster Kubernetes lui-même.

# Les autres types de Workloads

## Kubernetes



En plus du déploiement d'un application, Il existe pleins d'autre raisons de créer un ensemble de Pods:

- Le DaemonSet: Faire tourner un agent ou démon sur chaque nœud, par exemple pour des besoins de monitoring, ou pour configurer le réseau sur chacun des nœuds.

- Le Job : Effectuer une tache unique de durée limitée et ponctuelle, par exemple de nettoyage d'un volume ou la préparation initiale d'une application, etc.
- Le CronJob : Effectuer une tache unique de durée limitée et récurrente, par exemple de backup ou de régénération de certificat, etc.

De plus même pour faire tourner une application, les déploiements ne sont pas toujours suffisants. En effet ils sont peu adaptés à des applications statefull comme les bases de données de toutes sortes qui ont besoin de persister des données critiques. Pour cela on utilise un StatefulSet que nous verrons par la suite.

Étant donné les similitudes entre les DaemonSets, les StatefulSets et les Deployments, il est important de comprendre un peu précisément quand les utiliser.

Les Deployments (liés à des ReplicaSets) doivent être utilisés :

- Lorsque votre application est complètement découpée du nœud
- Que vous pouvez exécuter plusieurs copies sur un nœud donné sans considération particulière
- Que l'ordre de création des replicas et le nom des pods n'est pas important
- Lorsqu'on fait des opérations *stateless*

Les DaemonSets doivent être utilisés :

- lorsque au moins une copie de votre application doit être exécutée sur tous les nœuds du cluster (ou sur un sous-ensemble de ces nœuds).

Les StatefulSets doivent être utilisés :

- Lorsque l'ordre de création des replicas et le nom des pods est important
- Lorsqu'on fait des opérations *stateful* (écrire dans une base de données clustérisée)

## Jobs

Les jobs sont utiles pour les choses que vous ne voulez faire qu'une seule fois, comme les migrations de bases de données ou les travaux par lots. Si vous exécutez une migration en tant que Pod dans un deployment:

- Dès que la migration se finit le processus du pod s'arrête.
- Le replicaset qui détecte que l'"application" s'est arrêter va tenter de la redémarrer en créant le pod.
- Votre tâche de migration de base de données se déroulera donc en boucle, en repeuplant continuellement la base de données.

## CronJobs

Comme des jobs, mais se lancent à un intervalle régulier, comme les cron sur les systèmes unix.

## Déployer en une application

N'hésitez pas aussi à observer les derniers évènements arrivés à votre cluster avec

`kubectl get events --watch.`

- Commencez par supprimer les ressources présentes

`kubectl delete pod mes_pods`

`kubectl delete service mon_service`

- Changez de contexte avec :

`kubectl config set-context --current -namespace forma-ludo`

- Rendez-vous dans le répertoire `$HOME/formation/pod`

Nous allons d'abord déployer notre application comme un simple Pod (non recommandé mais montré ici pour l'exercice).

Ouvrir le fichier `nginx-pod.yaml`

- Modifier le nom du conteneur `nginx-prénom`
- Appliquez le fichier avec `kubectl apply -f nginx-pod.yaml`
- Modifiez le nom du pod dans la description précédente et réappliquez la configuration. Kubernetes mets à jour le nom.

Kubernetes fournit un ensemble de commande pour débugger des conteneurs :

- `kubectl logs <pod-name> -c <conteneur_name>` (le nom du conteneur est inutile si un seul)
- `kubectl exec -it <pod-name> -c <conteneur_name> -- bash`
- `kubectl attach -it <pod-name>`
- Explorez le pod avec la commande `kubectl exec -it <pod-name> -c <conteneur_name> -- bash` écrite plus haut.

- Supprimez le pod.

## Pod avec ressources et sondes

- Appliquez le fichier kuard-pod-full.yaml (apply -f )
- Ajoutons un service de type NodePort
- Créez un fichier demo-svc.yaml avec à l'intérieur le code suivant à compléter:

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport
spec:
  type: NodePort
  selector:
    app: demo
  ports:
    - port: 80
      targetPort: 80
      nodePort: 300XX
```

- Appliquez ce nouvel objet avec kubectl.
- Affichez votre service

kubectl get service

kubectl get all

=> Les services Kubernetes redirigent le trafic basé sur les étiquettes (labels) appliquées sur les pods du cluster. Il faut donc de même éviter d'utiliser deux fois le même label pour des parties différentes de l'application.

**Visitez votre application à l'adresse : formation.ludovic.tech:PORT**

Les sondes **Liveness** et **Readiness** sont deux mécanismes que Kubernetes utilise pour surveiller l'état de santé des conteneurs et pour s'assurer que les applications s'exécutent correctement sur un cluster.

La sonde Liveness est utilisée pour déterminer si un conteneur doit être redémarré en cas d'échec de l'application qu'il exécute. Si la sonde Liveness échoue, Kubernetes considère que le conteneur est en panne et le redémarre automatiquement. La sonde Liveness permet de s'assurer que les conteneurs sont toujours en cours d'exécution et que les applications qu'ils hébergent sont fonctionnelles.

La sonde Readiness est utilisée pour déterminer si un conteneur est prêt à répondre aux requêtes entrantes. Si la sonde Readiness échoue, Kubernetes considère que le conteneur n'est pas encore prêt à traiter les requêtes et ne le dirige pas vers les clients. Cela permet de s'assurer que les requêtes sont traitées uniquement par les conteneurs qui sont prêts à le faire, ce qui permet d'éviter des temps d'arrêt inutiles et des erreurs de connexion.

Les sondes Liveness et Readiness peuvent être configurées pour effectuer des vérifications de différents types, telles que des requêtes HTTP, des appels de méthode TCP ou des exécutions de commandes. Elles permettent d'assurer que les conteneurs sont fonctionnels et prêts à répondre aux requêtes, ce qui améliore la disponibilité et la fiabilité de l'application sur le cluster Kubernetes.

# Déployer une application de A à Z

Rendez-vous dans le répertoire :

```
formation/nginx-php-mariadb
```

Ce TP va consister à créer des objets Kubernetes afin de déployer une application microservices (plutôt simple) :Web, php, Base de données SQL

Commencez par supprimer les ressources présentes

```
kubectl delete pod mes_pods  
kubectl delete service mon_service  
kubectl delete all --all
```

Avant de pouvoir déployer nos applications, nous devons créer l'infrastructure pour notre appli. Le réseau, le stockage, de la configuration des services, des secrets, etc ...

- Création du stockage avec la ressource PersistentVolumeClaim (pvc)
- Création du réseau avec la ressource service de type ClusterIP
- Import des configurations avec la ressource ConfigMaps
- Import des données sensibles avec la ressource Secret
- Création d'url pour accéder à l'application

## TP 4 - Déployer la base de données

### Déployer MySQL avec du stockage et des Secrets

Ouvrir le fichier : mysql-deploy.yaml

Il nous faut :

- Du stockage PVC : Appliquer le fichier mysql-pvc.yaml
- Un service pour accéder a la DB : appliquer mysql-svc.yaml
- Vérifier que le stockage et le service sont présent.

Kubectl get all,pvc

### Observez la persistence

- Supprimez et recréer le pod DB. on constate que les données ont été conservées.