

docker

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

Trois transformations profondes de l'informatique

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

- Le cloud
- La conteneurisation logicielle
- Le mouvement DevOps

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

Le Cloud

- Au-delà du flou dans l'emploi de ce terme, le cloud est un mouvement de réorganisation technique et économique de l'informatique.
- On retourne à la consommation de "temps de calcul" et de services après une "aire du Personnel Computer".
- Pour organiser cela on définit trois niveaux à la fois techniques et économiques de l'informatique :
 - **Software as a Service** : location de services à travers internet pour les usagers finaux
 - **Platform as a Service** : location d'un environnement d'exécution logiciel flexible à destination des développeurs
 - **Infrastructure as a Service** : location de ressources "matérielles" à la demande pour des VMs et de conteneurs sans avoir à maintenir un data center.

Conteneurisation

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

Formation Docker

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'immuabilité qui implique de jeter les boîtes (automatiser pour lutter contre la culture de prudence). Rend l'infra prédictible.

Le mouvement DevOps

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

Objectifs du DevOps

- Rapidité (velocity) de déploiement logiciel (organisation agile du développement et livraison jusqu'à plusieurs fois par jour)
 - Implique l'automatisation du déploiement et ce qu'on appelle la CI/CD c'est à dire une infrastructure de déploiement continu à partir de code.

Formation Docker

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

Les microservices permettent justement le DevOps car :

- Ils peuvent être déployés séparément

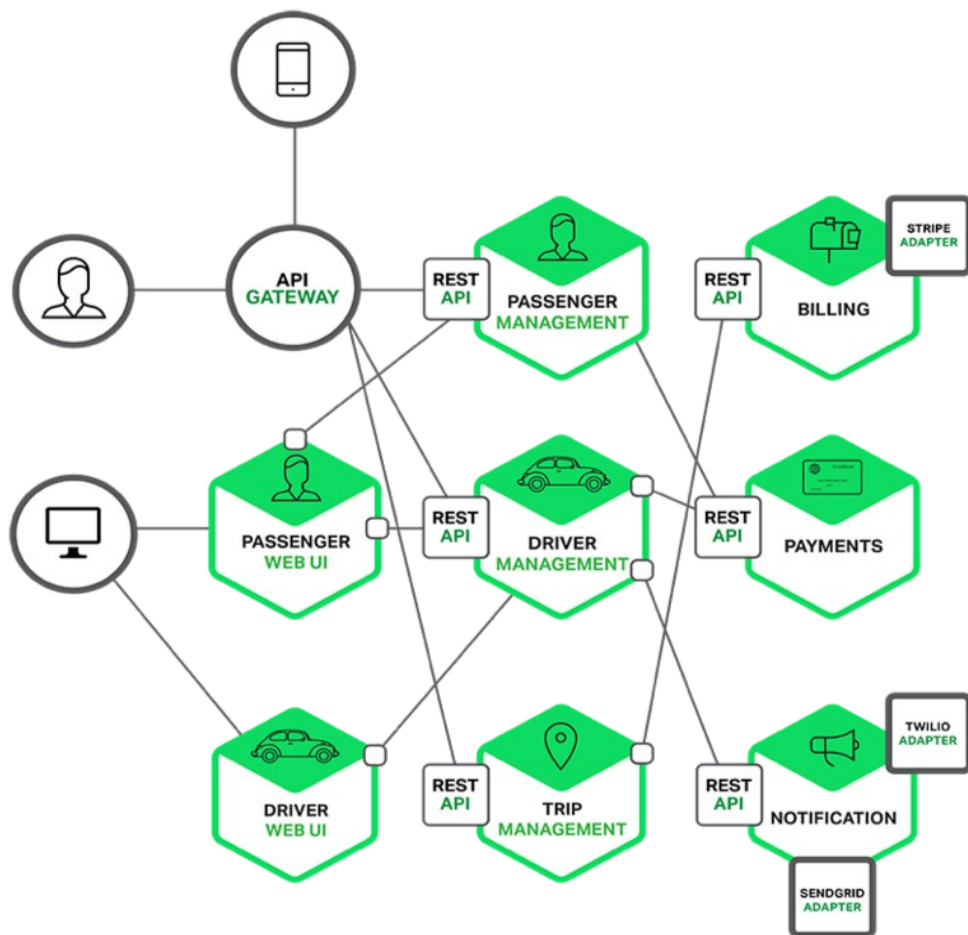
Une petite équipe gère chaque service ou groupe thématique de services

Comprendre les applications microservices

Introduction

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

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

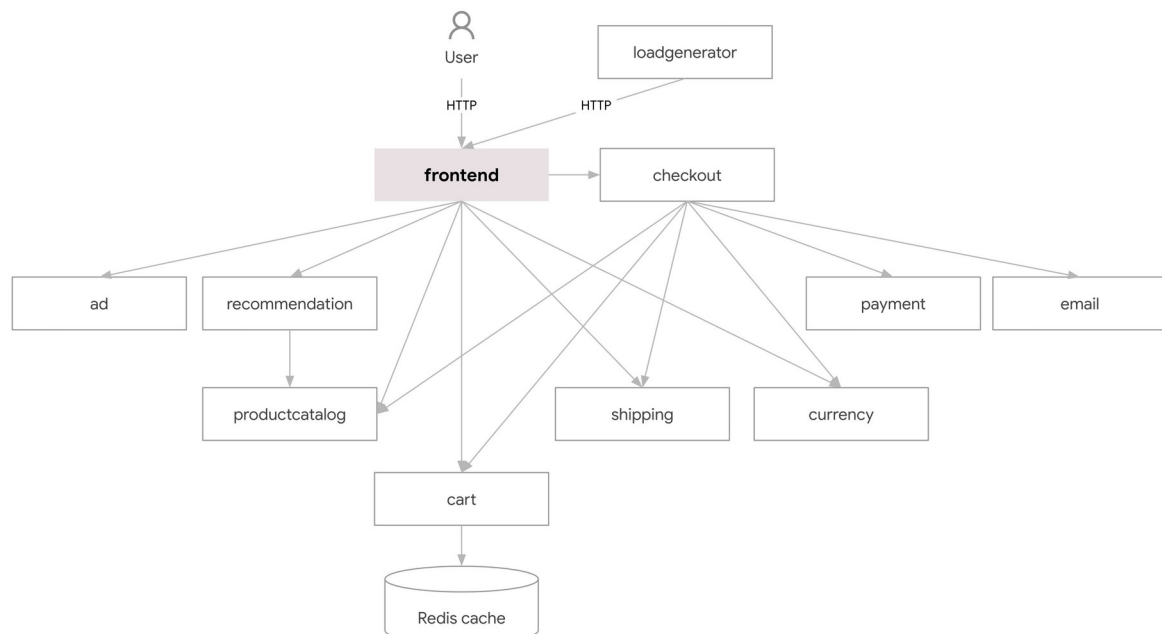


Les microservices, qu'est-ce que c'est ?

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

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

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



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

L'objectif, qui consiste tout simplement à proposer des logiciels de qualité en un temps record, devient atteignable grâce aux microservices. Pour autant, d'autres éléments entrent également en ligne de compte. La décomposition des applications en microservices ne suffit

pas. Il faut aussi gérer ces microservices, les orchestrer et traiter les données qui sont générées et modifiées par les microservices.

Quels sont les avantages des microservices ?

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

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

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

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

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

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

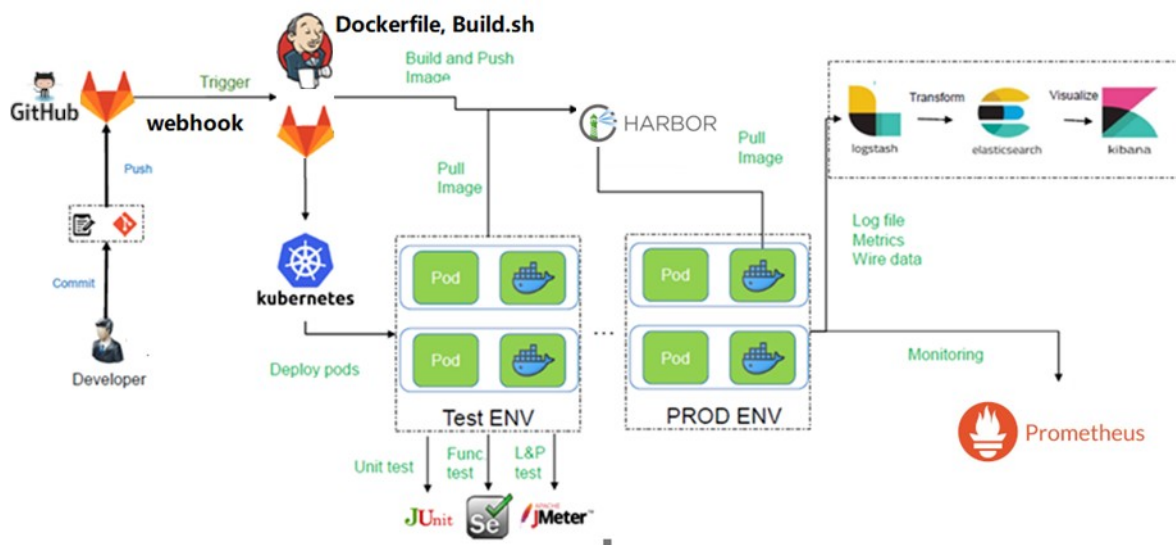
Pour qu'une architecture de microservices s'exécute comme une application cloud fonctionnelle, les services doivent en permanence demander des données à d'autres services via un système de messagerie.

Si la création d'une couche de Service Mesh (réseau maillé) directement dans l'application facilite la communication entre les services, l'architecture de microservices doit également

intégrer les applications existantes et les autres sources de données. Base de données As Service, Monitoring As Service, etc ...

L'intégration agile est une stratégie de connexion des ressources qui allie des technologies d'intégration, des techniques de distribution agile et des plateformes cloud-native dans le but d'accélérer la distribution des logiciels tout en renforçant la sécurité.

DevOps (Jenkins, Ansible, etc...)



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

Voici les principales étapes d'un pipeline DevOps typique :

- **Intégration** : les modifications de code sont intégrées dans un référentiel centralisé (par exemple, Git) qui sert de source unique de vérité pour l'ensemble du projet.
- **Compilation** : le code source est compilé en un exécutable ou un package.
- **Test** : le code est soumis à une série de tests automatisés pour garantir qu'il fonctionne comme prévu et qu'il répond aux normes de qualité.
- **Intégration continue** : les modifications de code sont automatiquement intégrées dans le référentiel principal et testées à chaque modification de code.
- **Livraison continue** : le code compilé et testé est automatiquement livré à un environnement de développement, de test ou de production.

Formation Docker

- Déploiement continu : le code livré est automatiquement déployé dans l'environnement de production.
- Surveillance : l'application est surveillée pour détecter les erreurs, les pannes et les problèmes de performance. Les rapports de surveillance sont utilisés pour améliorer la qualité du code et des processus DevOps.

L'automatisation de ces étapes garantit que les changements de code sont rapidement intégrés, testés et livrés, ce qui permet aux équipes de développement de se concentrer sur la création de fonctionnalités et d'améliorations, plutôt que de passer du temps à configurer et à déployer des logiciels manuellement. Le pipeline DevOps permet également de réduire les erreurs humaines et d'améliorer la qualité du code grâce à une surveillance constante et à des tests automatisés.

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

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

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

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

Connexion sur le serveur de formation

Se connecter sur le serveur de formation

Serveur : formation.ludovic.io

Port ssh : 3460

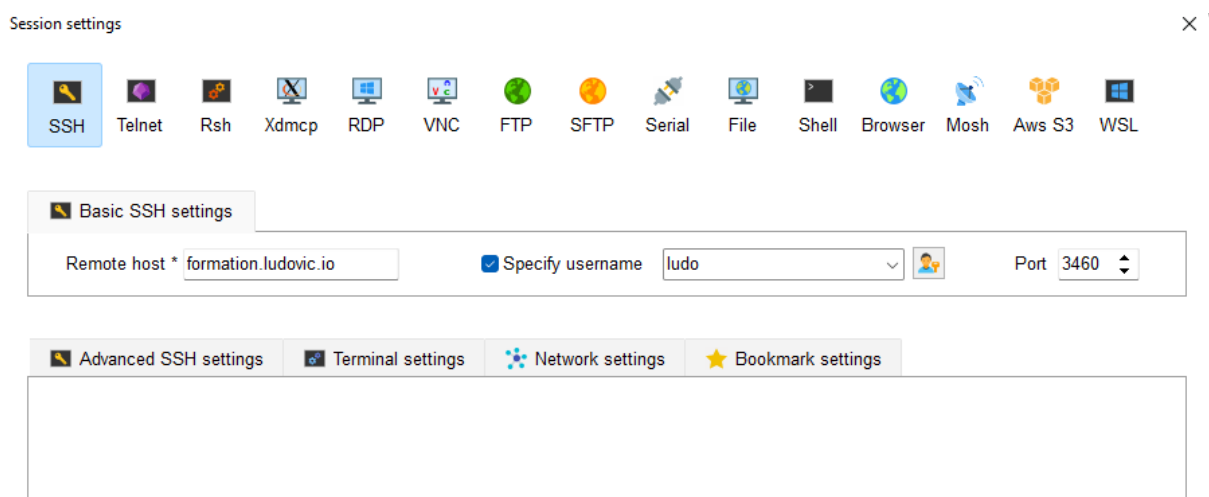
login : prénom (SANS ACCENT , NI MAJUSCULE)

password: Plb@2024

Avec un terminal Windows 11

```
ludo@formation:~  
PS C:\Users\ludo> ssh -p 3460 ludo@formation.ludovic.io  
ludo@formation.ludovic.io's password:  
Activate the web console with: systemctl enable --now cockpit.socket  
  
Last login: Mon Nov 27 08:40:51 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 :

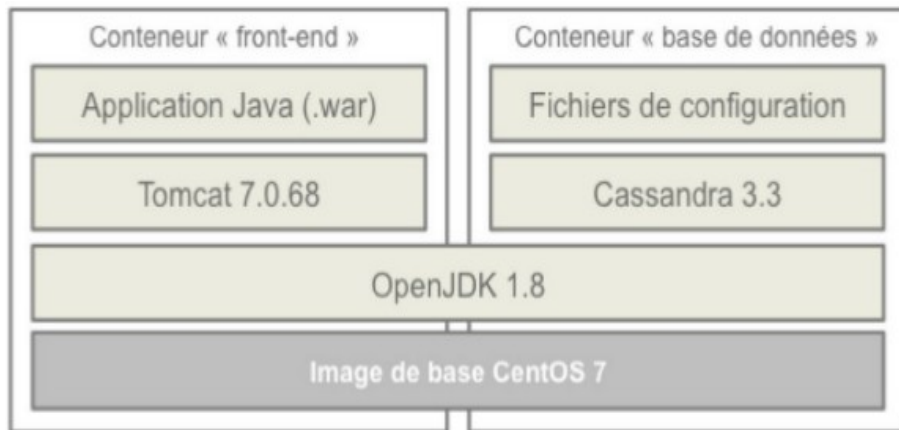
- Couche de base : cette couche contient l'image de base, qui peut être une distribution Linux, comme Alpine ou Ubuntu.
- Couche d'installation : cette couche contient les instructions pour installer les packages et les dépendances nécessaires à l'application.
- Couche de configuration : cette couche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

Formation Docker

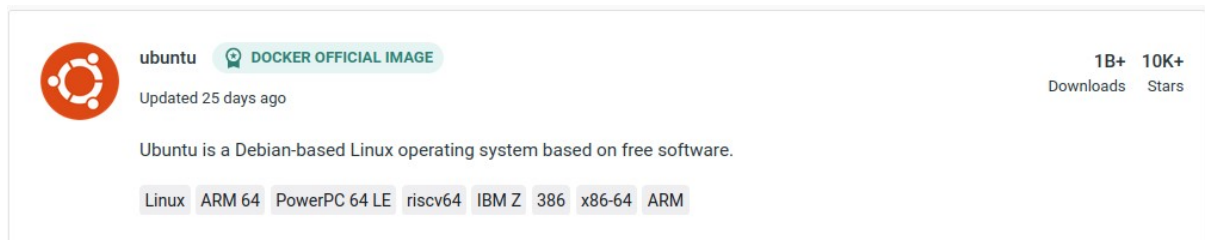
Une image Docker est un modèle en lecture seule, utilisé pour créer des conteneurs Docker.
Une image est constitué d'une série de couche



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.



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



Formation Docker

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 créer 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.

TP- Manipuler ses images

Télécharger une image Centos version 7 à partir du Docker hub

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

Télcharger la deniere version de debian

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

Récupérer l'image nginx en version 1.19

```
[ludo@dockerHost:~]$ docker image pull nginx:1.19
```

Lister les images sur l'hôte

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

Formation Docker

Lister les couches sur l'image nginx

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


Supprimer l'image.

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](#)



☐ Send me occasional product updates and announcements.



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: votre_login
password: *****
LoginSucceeded
```

Renommer une image et pousser une image sur le Hub

```
[ludo@dockerHost:~]$ docker image tag nginx:latest votre_login/web:latest

[ludo@dockerHost:~]$ docker image push votre_login/web:latest
The push refers to repository [docker.io/votre_login/web:latest]
2653d992f4ef: Mounted from library/nginx
7.5: digest: sha256:dbbacecc49b088458781c16f3775f2a2ec7521079034a7ba499c8b0bb7f86875
size: 529
```

Formation Docker

quenec

▼

Search by repository name

Create Repository

quenec / **centos**
Updated 6 minutes ago

Not Scanned

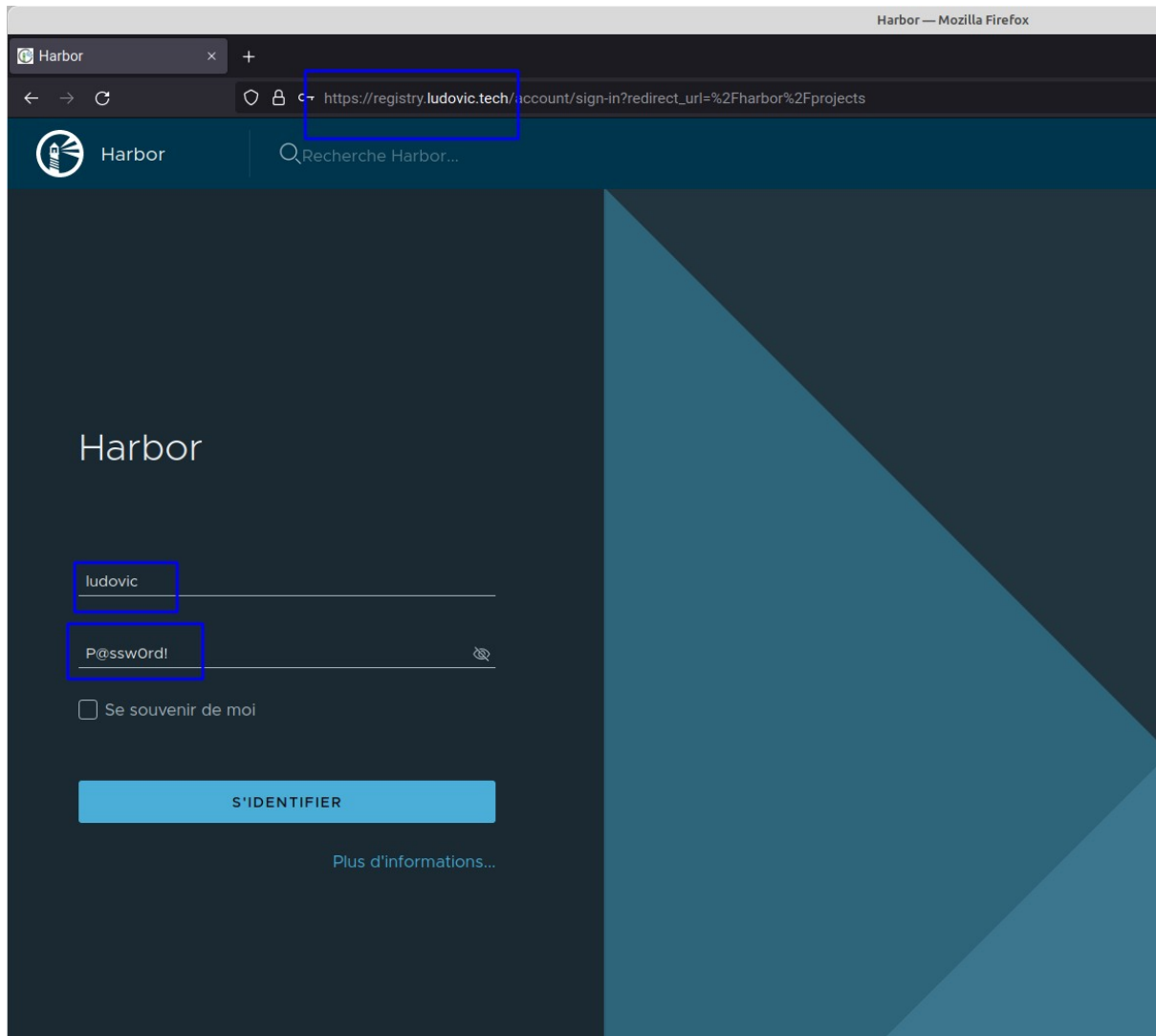
☆ 0

↓ 1

Public

Registre d'entreprise Harbor

Connexion sur notre dépôt d'image Harbor **registry.ludovic.io**



S'identifier sur le hub Docker

```
ludo@dockerHost:~]$ docker login registry.ludovic.io
```

```
username: user
```

```
password: P@ssw0rd!
```

```
LoginSucceeded
```

Formation Docker

Renommer une image et pousser une image sur le Hub

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

Poussez cette image dans le registre Harbor

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

formation

Administrateur Système

Résumé

Dépôts

Membres

Labels

Scanneur

Préchauffage P2P

Politique

Comptes Robot

Web

X

SUPPRIMER

<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 premier 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 créer 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 debian:latest
```

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 /]# apt install -y apache2
```

Démarrer le serveur Web

```
[root@5deeb8a5d681 /]# apachectl start
```

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 debian avec Apache installé

```
[ludo@docker-host-00]$ docker commit apache local/debian_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 --it --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@dockerHostdocker:~$ mkdir www-data
[ludo@dockerHostdocker:~$ 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
```

Formation Docker

```
$HOME/www-data/index.html: Permission denied
```

```
[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 -h localhost -P 3306 --protocol=tcp
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
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
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 image 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.

- 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.

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 Dockerfile s, mais il existe d'autres instructions qui peuvent être utilisées pour personnaliser davantage le processus de construction des images Docker, telles que ENTRYPOINT, ENV et LABEL, ARG. L'utilisation judicieuse de ces instructions permet de créer des images Docker efficaces, reproductibles et portables.

L'instruction **ENTRYPOINT** 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 ENTRYPOINT est considérée comme une commande de base qui ne peut pas être remplacée par une autre commande.

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

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

Par exemple, supposons que nous ayons un conteneur qui exécute une application Python. Nous pouvons utiliser l'instruction ENTRYPOINT 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"]
```

Dans cet exemple, l'instruction ENTRYPOINT 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

Formation Docker

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

Modifions le Dockerfile

```
[ludo@dockerHost:~]$ cat Dockerfile
FROM debian:buster-slim
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:1.21  
WORKDIR /src  
COPY main.go ./main.go  
RUN go build -o /bin/hello ./main.go  
  
CMD ["/bin/hello"]  
  
[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
```

Instancions l'image

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


Formation Docker

Édition de notre Dockerfile

```
[ludo@dockerHost:~/multiStage$ vim Dockerfile
FROM golang:1.21
WORKDIR /src
COPY main.go ./main.go
RUN go build -o /bin/hello ./main.go

FROM alpine
COPY --from=0 /bin/hello /bin/hello

CMD ["/bin/hello"]
```

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
```

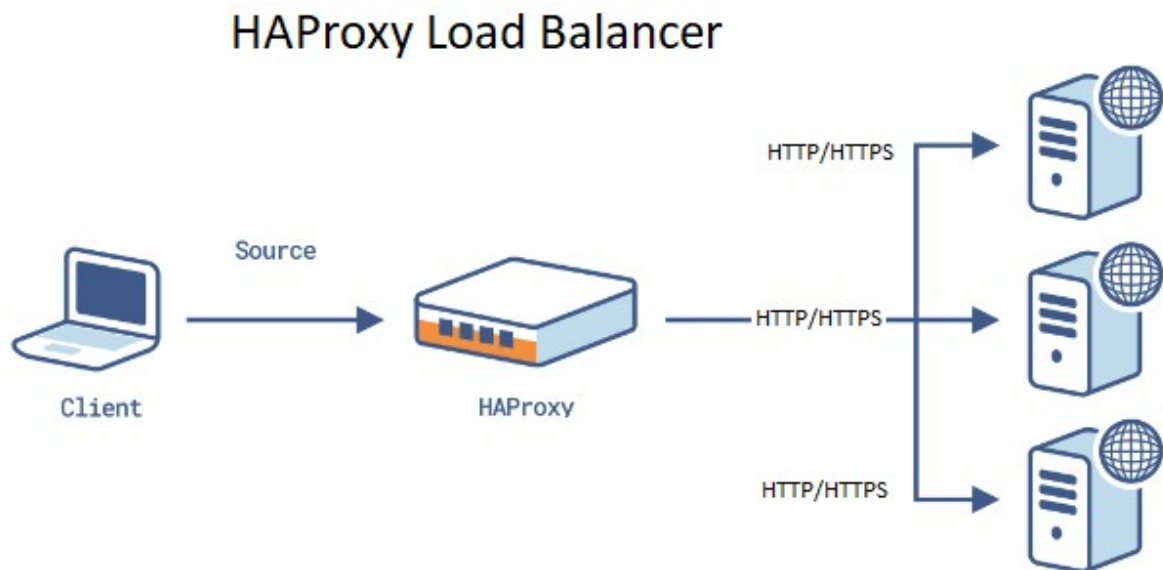
Lister l'image

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

Instancions l'image

```
[ludo@dockerHost:~]$ docker container run --rm app-go:2.0
hello, world, form Docker
```

Projet haweb mult-conteneur - Serveur web Hautement disponible



Créer un répertoire pour notre projet

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

Edition du Dockerfile

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

Formation Docker

Contenue 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:
```

Vérification de la configuration

```
[ludo@dockerHost:~]$ docker compose config
name: haweb
services:
  haweb:
    build:
....
```

Formation Docker

Démarrage de la stack docker-compose

```
[ludo@dockerHost:~]$ docker compose up -detach
```

```
:: Container web0      Started                                0.5s
:: Container web1      Started                                0.5s
....
```

Lister la stack docker-compose

```
[ludo@dockerHost:~]$ docker compose ps
```

Mise à jour avec le 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:
```

Formation Docker

```
image: dockercloud/hello-world
```

```
networks:
```

```
- backend
```

```
volumes:
```

```
- www-data:/www/
```

```
...
```

```
Volumes :
```

```
www-data :
```

Démarrage de la stack docker-compose

```
[ludo@dockerHost:~]$ docker compose up -detach
```

```
:: Container web0      Started                                0.5s
```

```
:: Container web1      Started                                0.5s
```

```
....
```

Lister la stack docker-compose

```
[ludo@dockerHost:~]$ docker compose ps
```


TP Docker compose application php.

Lister la stack docker-compose

Le service de développement d'application nous confit l'intégration leur application de paye, codé en php.

Vous trouverez le code dans le répertoire /root/db-data de votre machine.

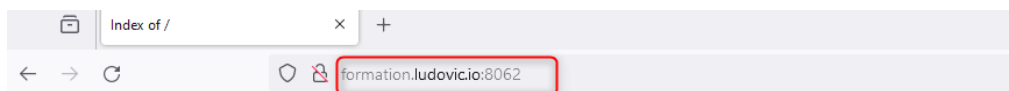
```
[root@ludo-dockerhost ~]# ls db-data/  
create_db.php create_table.php getting_data.php insert_data.php ...
```

Cette application stocke les données dans une base de données mariadb.





Cette application sera accessible via le serveur apache a l'url suivante :

formation.ludovic.io:80+IP (ex : ip = 10.0.0.50 port = 8050)

url : formation.ludovic.io :8050



Index of /

	<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
	create_db.php	2023-12-13 06:59	493	
	create_table.php	2023-12-13 06:59	770	
	getting_data.php	2023-12-13 06:59	761	
	insert_data.php	2023-12-13 06:59	2.8K	

Formation Docker

Nous devons créer une stack docker-compose. Avec des images quenec/apache-php et mariadb.

Configuration du service apache-php :

- Une image : quenec/apache-php
- Un volume pour stocker les fichiers php
- Un port 8080 pour exposer notre application
- Le service web apache-php dépend du service de base de données

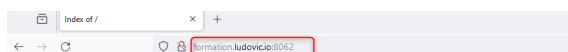
Configuration du service mysql

- Une image mariadb:latest
- Une variable d'environnement pour initialiser mariadb : MARIADB_ROOT_PASSWORD
- Un volume pour les données persistantes : /var/lib/mysql
- Le nom de votre service est le nom DNS de votre application de base de données





```
[root@ludo-dockerhost ~]# cat db-data/create_db.php
<?php
$dbhost = 'mysql';
$dbuser = 'root';
$dbpass = 'root';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
```

Notre application sera accessible a l'url

formation.ludovic.io:80+IP (ex : ip = 10.0.0.50 port = 8050) , url : formation.ludovic.io :8050



Index of /

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 create_db.php	2023-12-13 06:59	493	
 create_table.php	2023-12-13 06:59	770	
 getting_data.php	2023-12-13 06:59	761	
 insert_data.php	2023-12-13 06:59	2.8K	

Formation Docker

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 persistance 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

Formation Docker

Service ELK avec compose

```
[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 :

Formation Docker

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

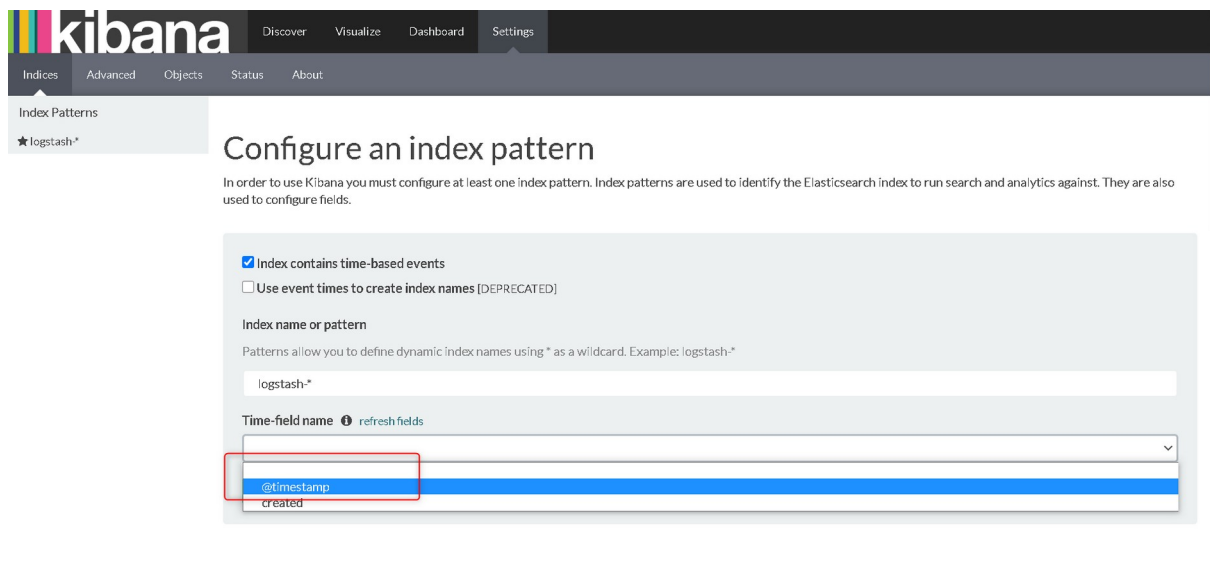
Se connecter sur l'HIM Kibana via le port 5601

http://adresse_IP:5601 Sur votre VM locale

Ou sur le serveur de formation :

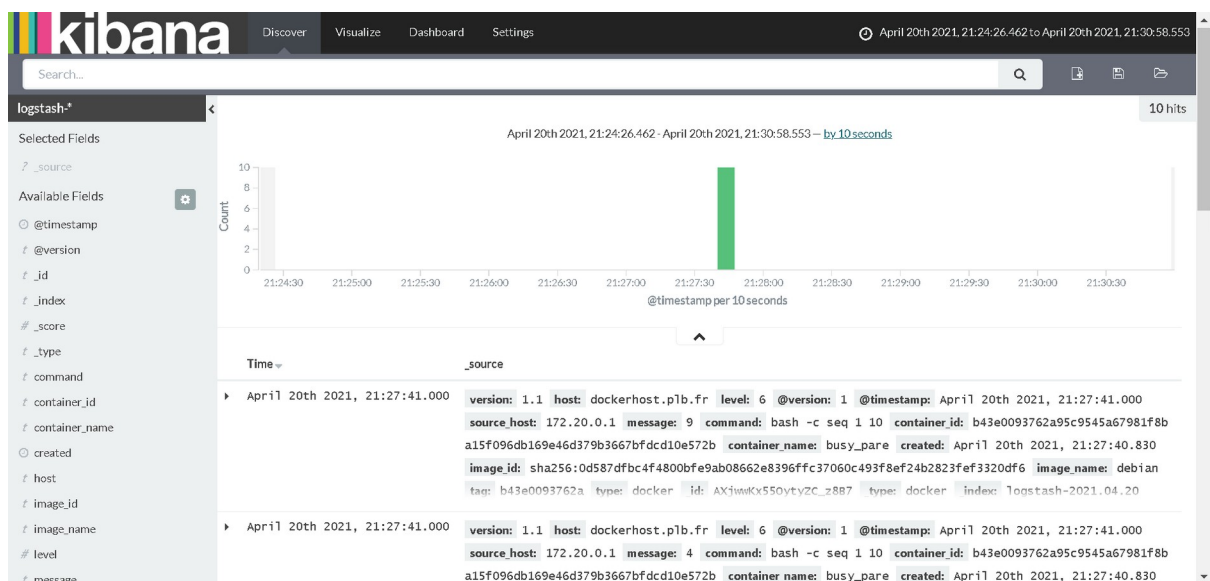
formation.ludovic.io:56+IP (ex : ip = 10.0.0.80 port = 5680) formation.ludovic.io5680

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



The screenshot shows the Kibana interface with the 'Settings' tab selected. Under 'Index Patterns', the 'logstash-*' pattern is chosen. The 'Configure an index pattern' page is displayed, showing options for time-based events and a text input for the index name or pattern, which contains 'logstash-*'. Below this, a dropdown menu for 'Time-field name' is open, showing '@timestamp' and 'created', with '@timestamp' selected and highlighted by a red box.

Voici nos journaux



Formation Docker

Ajout de l'envoi des logs dans nos applications

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

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

web0:

image: dockercloud/hello-world

networks:

- backend

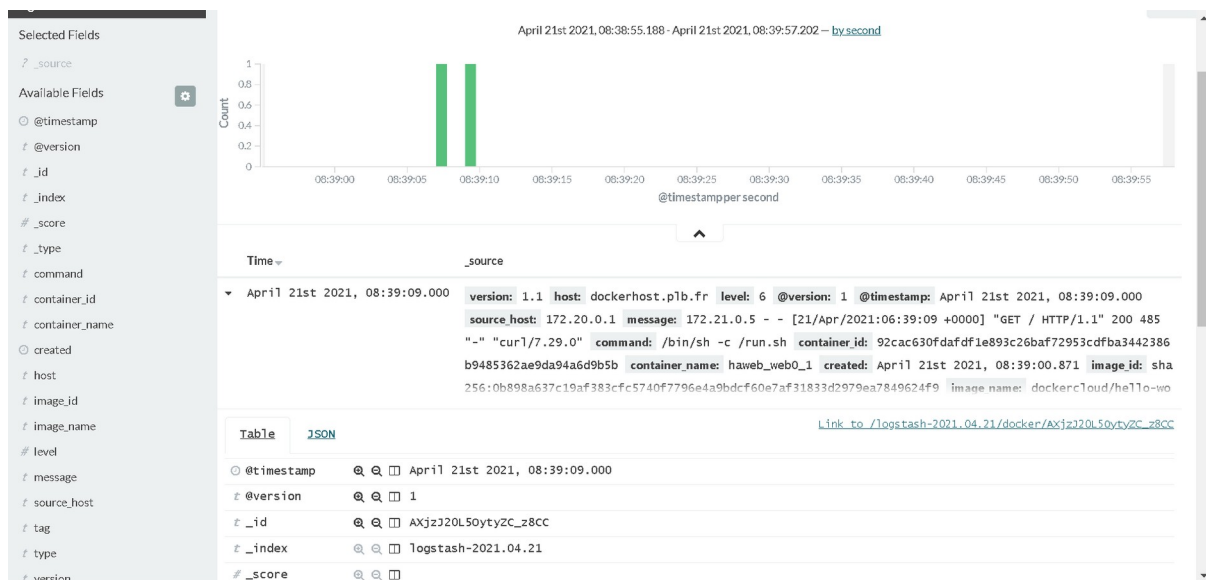
logging:

driver: "gelf"

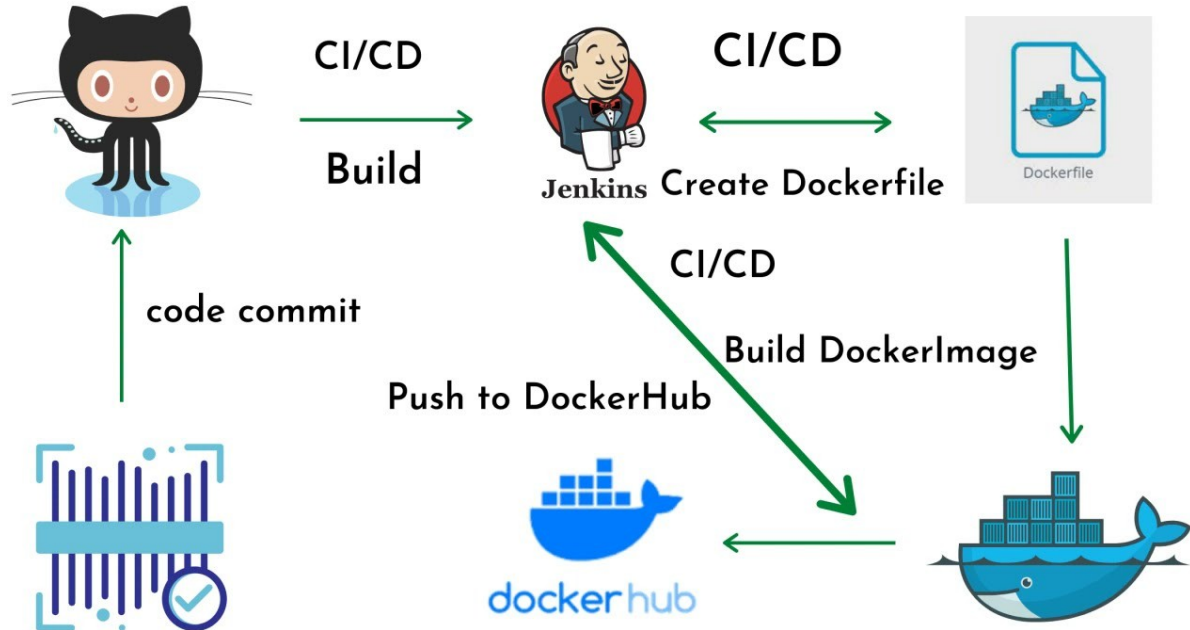
options:

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 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 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
```

Veillez 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"/>

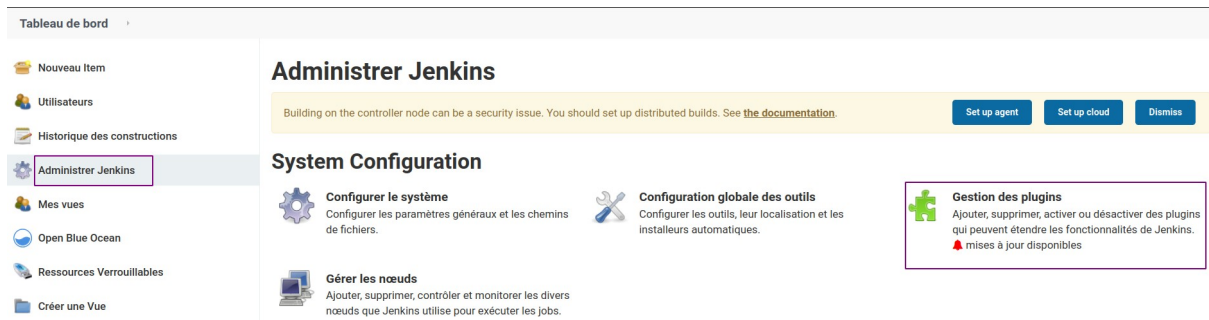
: 2.303.2

Continuer en tant qu'Administrateur

Sauver et continuer

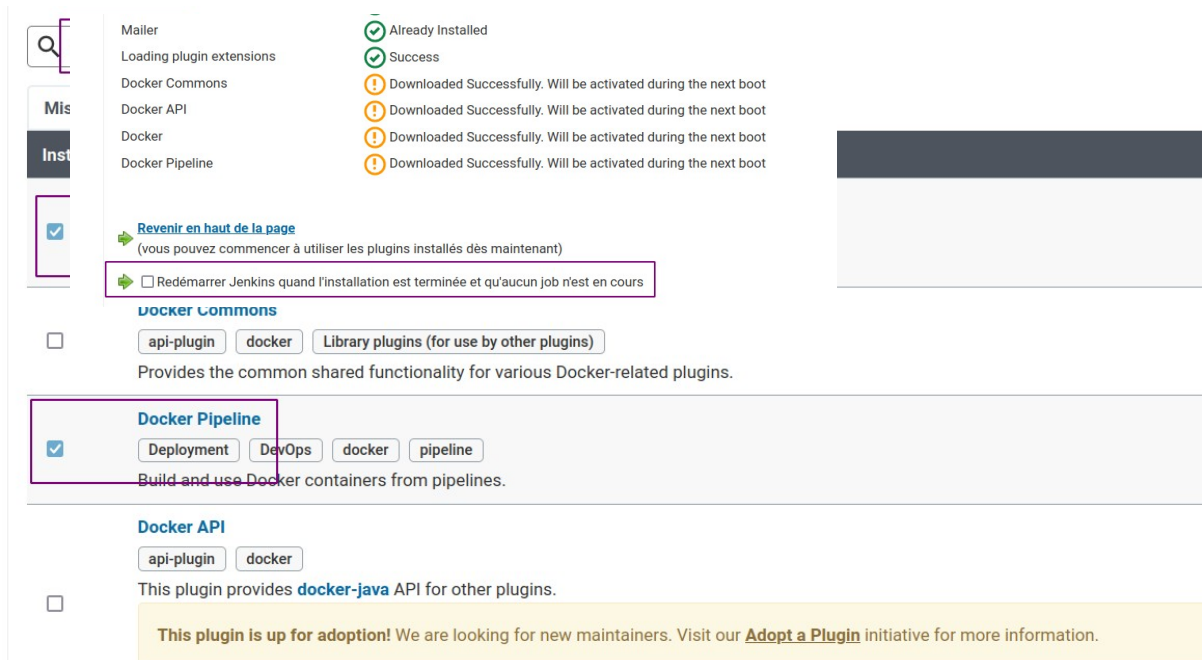
Plugins Docker et Docker pipelines

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



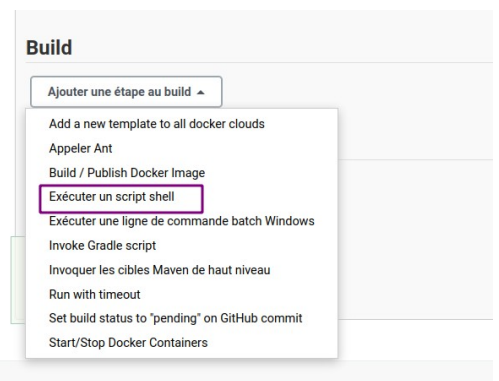
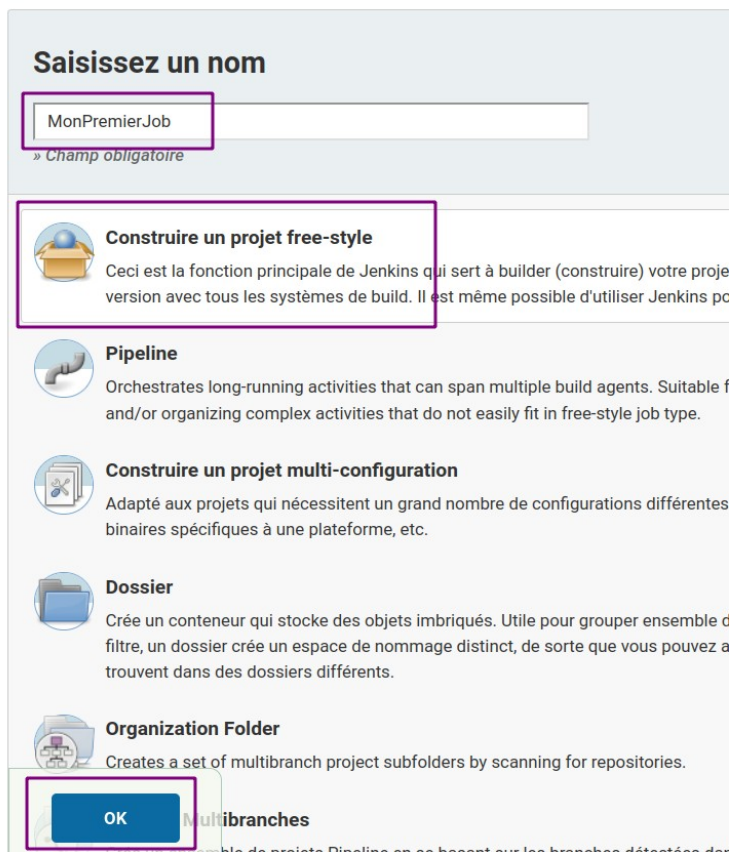
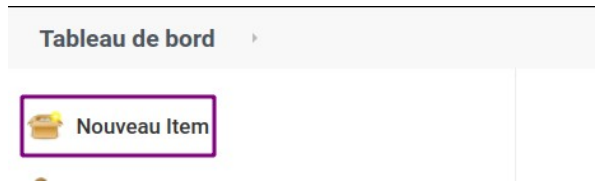
The screenshot shows the Jenkins Administration page. On the left is a sidebar with navigation links: 'Nouveau Item', 'Utilisateurs', 'Historique des constructions', 'Administrer Jenkins' (highlighted), 'Mes vues', 'Open Blue Ocean', 'Ressources Verrouillables', and 'Créer une Vue'. The main content area is titled 'Administrer Jenkins' and includes a warning banner about distributed builds. Below this is the 'System Configuration' section with three sub-sections: 'Configurer le système' (configure general parameters and file paths), 'Configuration globale des outils' (configure tools, their location, and automatic installers), and 'Gérer les nœuds' (add, delete, control, and monitor nodes). A 'Gestion des plugins' box on the right provides instructions on how to manage plugins.

On redémarre Jenkins, suite l'installation des plugins



The screenshot shows the Jenkins Plugin Manager. On the left is a sidebar with a search icon and a list of installed plugins: 'Mailier', 'Loading plugin extensions', 'Docker Commons', 'Docker API', 'Docker', and 'Docker Pipeline'. The main content area shows the status of these plugins: 'Mailier' is 'Already Installed', 'Loading plugin extensions' is 'Success', and the others are 'Downloaded Successfully. Will be activated during the next boot'. A 'Revenir en haut de la page' link is present. Below this is a checkbox to 'Redémarrer Jenkins quand l'installation est terminée et qu'aucun job n'est en cours'. The 'Docker Commons' section is expanded, showing 'api-plugin', 'docker', and 'Library plugins (for use by other plugins)'. The 'Docker Pipeline' section is also expanded, showing 'Deployment', 'DevOps', 'docker', and 'pipeline'. The 'Docker API' section is partially visible at the bottom, showing 'api-plugin' and 'docker'. A yellow banner at the bottom states: 'This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.'

Notre premier job ou build



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


Tableau de bord


MonPremierJob


#4


 Retour au projet


 État


 Modifications


 Console Output

 View as plain text

 Informations de la construction

 Delete build '#4'

 Open Blue Ocean

 **Sortie de la console**

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

Saisissez un nom

BuidDocker

» Champ obligatoire

Construire un projet free-style
Ceci est la fonction principale de Jenkins version avec tous les systèmes de build. I

Pipeline
Organise des activités de longue durée qu (anciennement connues comme workflow type libre.

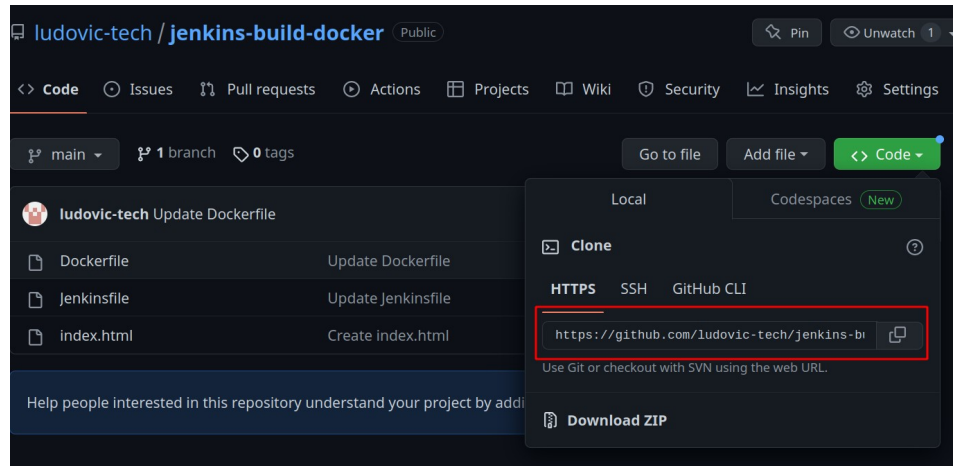
Construire un projet multi-configur
Adapté aux projets qui nécessitent un gra binaires spécifiques à une plateforme, etc

Dossier
Crée un conteneur qui stocke des objets i un dossier crée un espace de nommage c dans des dossiers différents.

OK

tion Folder

set of multibranch project subfi



Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/ludovic-tech/jenkins-build-docker.git

Credentials ?

- aucun -

+ Ajouter

Avancé...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Navigateur de la base de code ?

(Auto)

Additional Behaviours

Ajouter

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

Pipeline Syntax

Sauver Apply

Formation Docker

Pipeline de Build

```
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
15      sh 'docker ps | grep srv_web'
16    }
17  }
18 }
19 }
```

Tableau de bord > BuidDocker >

↑ Back to Dashboard

🔍 Status

📁 Changes

▶ Lancer un build

⚙️ Configurer

🗑️ Supprimer Pipeline

🔍 Full Stage View

🌐 Open Blue Ocean

✎ Renommer

🔍 Pipeline Syntax

Pipeline BuidDocker

Recent Changes

Stage View

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

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

Dec 16 07:13 No Changes

Liens permanents

Historique des builds **tendance** ▼

🔍 Filter builds...

✅ #1 16 déc. 2022 07:13

📡 Atom feed des builds 📡 Atom feed des échecs

Un pipeline de Build et de Push


1.





2.


Saisissez un nom


» 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 pour

**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 av trouvent dans des dossiers différents.

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

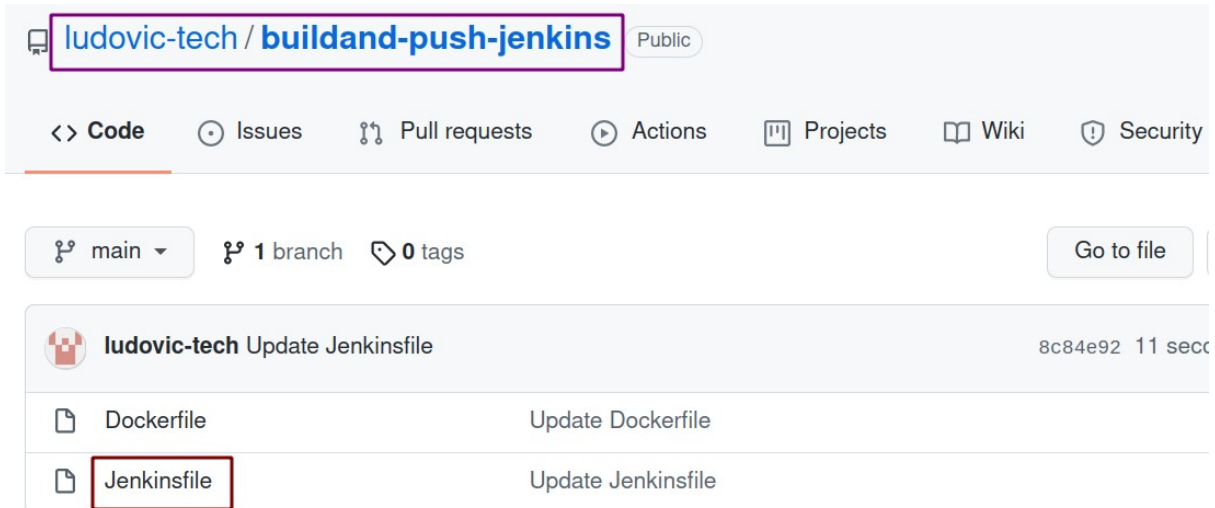
Multibranches

Crée un ensemble de projets 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>



ludovic-tech / buildand-push-jenkins Public

<> Code Issues Pull requests Actions Projects Wiki Security

main 1 branch 0 tags Go to file

	ludovic-tech Update Jenkinsfile	8c84e92 11 sec
	Dockerfile	Update Dockerfile
	Jenkinsfile	Update 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

Pipeline script from SCM

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

6. On créer un « credentials »

Credentials

- aucun - Ajouter

Jenkins

7.

Identifiants globaux (illimité)

Type

Nom d'utilisateur et mot de passe

Portée

Global (Jenkins, agents, items...)

Nom d'utilisateur

admin

☐ Treat username as secret

Mot de passe

ID

registry_id

Description

Login et password du registre local

Ajouter Annuler

Formation Docker

Repository URL

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

Credentials

admin/***** (Login et password du registre local) [Ajouter](#)

Branches to build

Branch Specifier (blank for 'any')

*/main

Navigateur de la base de code

(Auto)

Additional Behaviours

[Ajouter](#)

Script Path

Jenkinsfile

8.

Tableau de bord > BuildAndPushDocker >

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Lancer un build](#)

[Configurer](#)

[Supprimer Pipeline](#)

[Full Stage View](#)

[Open Blue Ocean](#)

[Rename](#)

[Pipeline Syntax](#)

[Historique des builds](#) **tendance**

find

[#1](#) 12 oct. 2021 16:27

Pipeline BuildAndPushDocker

[Recent Changes](#)

Stage View

	Clone	Build	Run	Push
Average stage times: (Average full run time: ~3s)	366ms	334ms	1s	1s
#1 Oct 12 18:27 No Changes	366ms	334ms	1s	1s

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.

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.

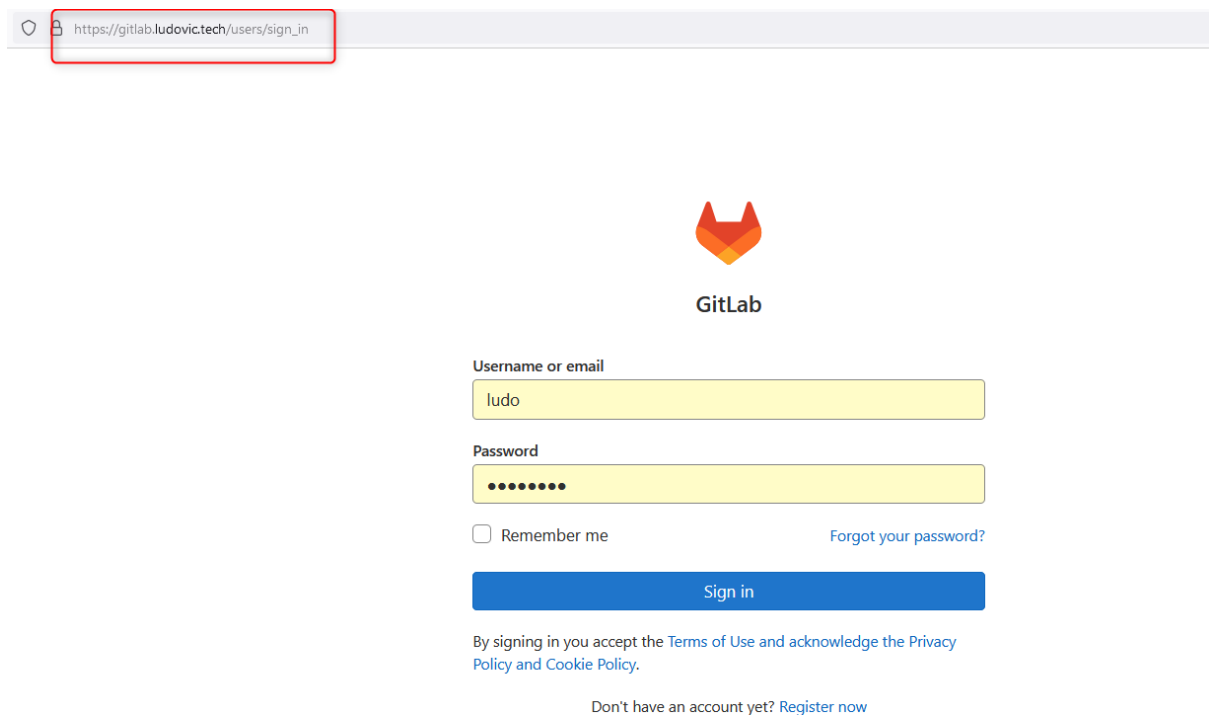
Formation Docker

TP : Gitlab CI


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

Login : prénom

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



https://gitlab.ludovic.tech/users/sign_in


GitLab

Username or email
ludo

Password
●●●●●●

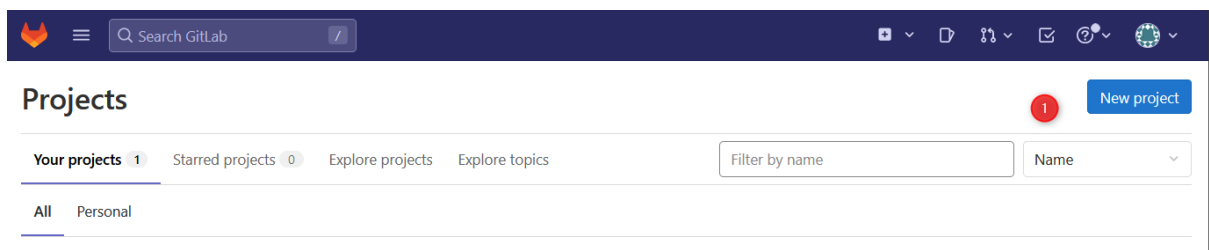
☐ Remember me [Forgot your password?](#)




[Sign in](#)

By signing in you accept the [Terms of Use](#) and acknowledge the [Privacy Policy](#) and [Cookie Policy](#).

Don't have an account yet? [Register now](#)


Créer un nouveau projet pour Docker





Projects 1 [New project](#)


[Your projects](#) 1 [Starred projects](#) 0 [Explore projects](#) [Explore topics](#)

Name 

[All](#) [Personal](#)

Formation Docker





Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

New project > Create blank project

Project name

c-cd-docker

1

Project URL

https://gitlab.ludovic.tech/ludo/


/

Project slug

c-cd-docker


Want to organize several dependent projects under the same namespace? [Create a group.](#)

Visibility Level




☒ 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

☒ 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.](#)

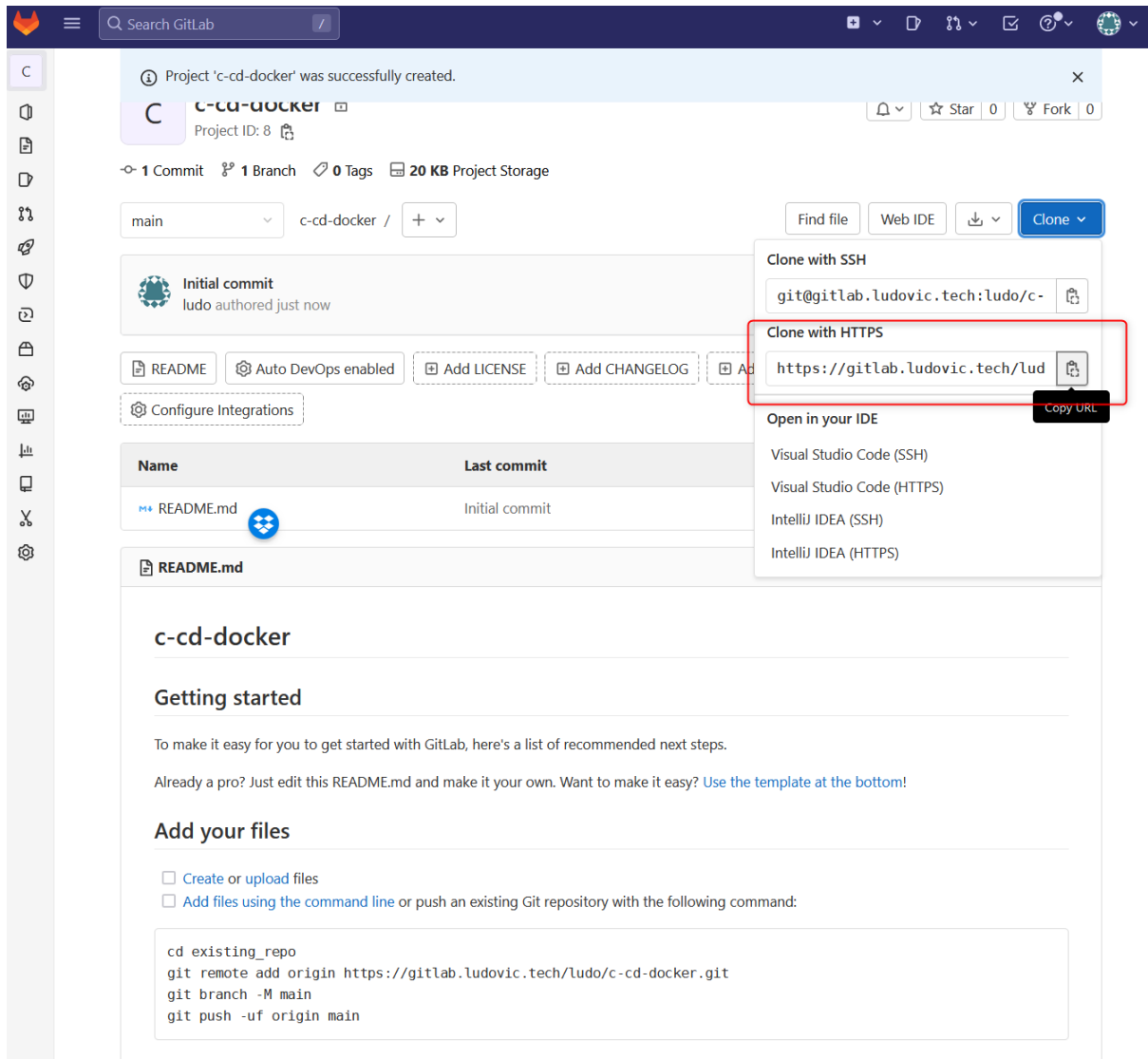
2

Create project

Cancel

Formation Docker

Cloner le projet :



Project 'c-cd-docker' was successfully created.

c-cd-docker
Project ID: 8

1 Commit 1 Branch 0 Tags 20 KB Project Storage

main c-cd-docker / +

Initial commit
ludo authored just now

README Auto DevOps enabled Add LICENSE Add CHANGELOG Add

Configure Integrations

Name	Last commit
README.md	Initial commit

c-cd-docker

Getting started

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? [Use the template at the bottom!](#)

Add your files

☐ Create or upload files

☐ Add files using the command line or push an existing Git repository with the following command:

```
cd existing_repo
git remote add origin https://gitlab.ludovic.tech/ludo/c-cd-docker.git
git branch -M main
git push -u origin main
```

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"]

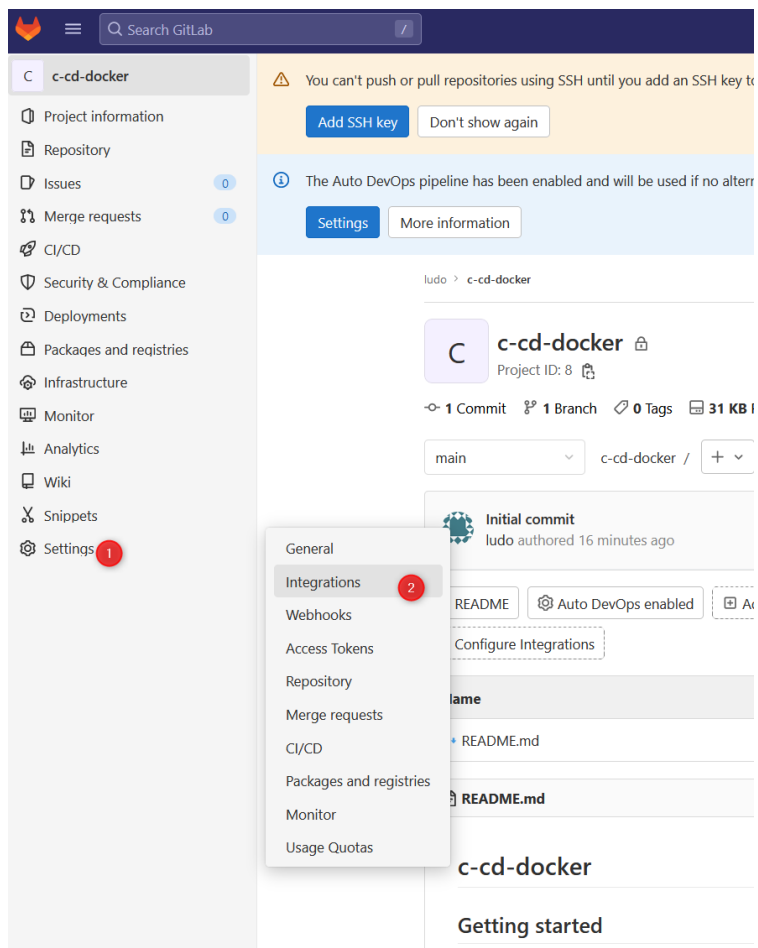
stages:
  - build

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 :



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

Base URL of the Harbor instance.

Harbor project name

The name of the project in Harbor.

Harbor username

Enter new Harbor password

Leave blank to use your current password.

Formation Docker

Ajout d'un token :

The screenshot displays the GitLab User Settings page for 'Access Tokens'. The left sidebar contains a list of settings: 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 includes a search bar. Below the title, there is a section 'Add a personal access token' with a red circle highlighting the 'Token name' input field. The 'Expiration date' is set to '2023-06-16'. Under 'Select scopes', there are several checkboxes for different permissions: 'api', 'read_api', 'read_user', 'read_repository', 'write_repository', 'read_registry', and 'write_registry'. A 'Create personal access token' button is located at the bottom of the form. On the right side, a user profile dropdown menu is visible, showing the user 'ludo' and options like 'Set status', 'Edit profile', 'Preferences' (highlighted with a red circle), and 'Sign out'.

User Settings > Access Tokens

Search page

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

Token name

Expiration date

2023-06-16

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.

Create personal access token

Active personal access tokens (1)

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.

Create personal access token

2

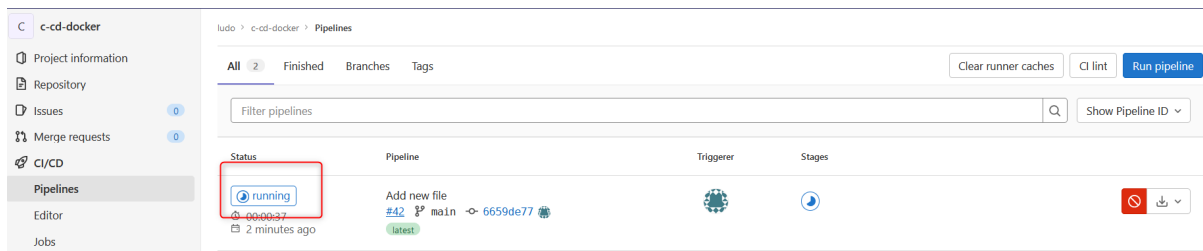
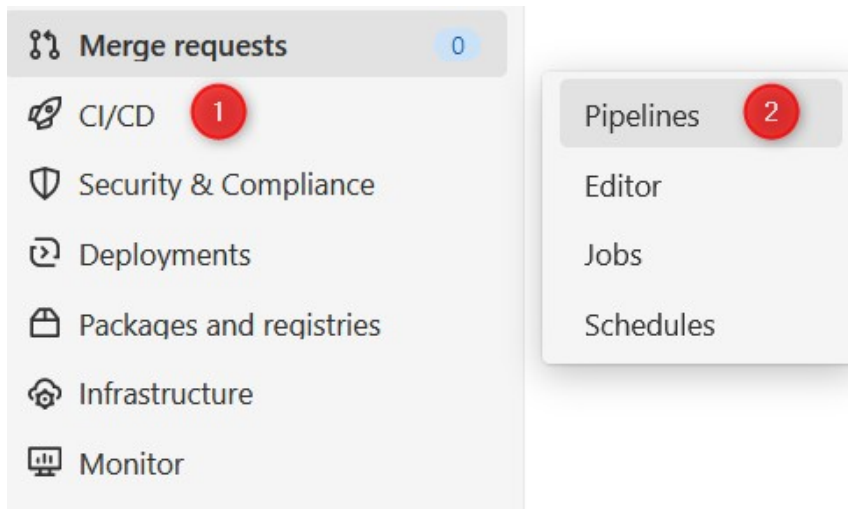
Your new personal access token



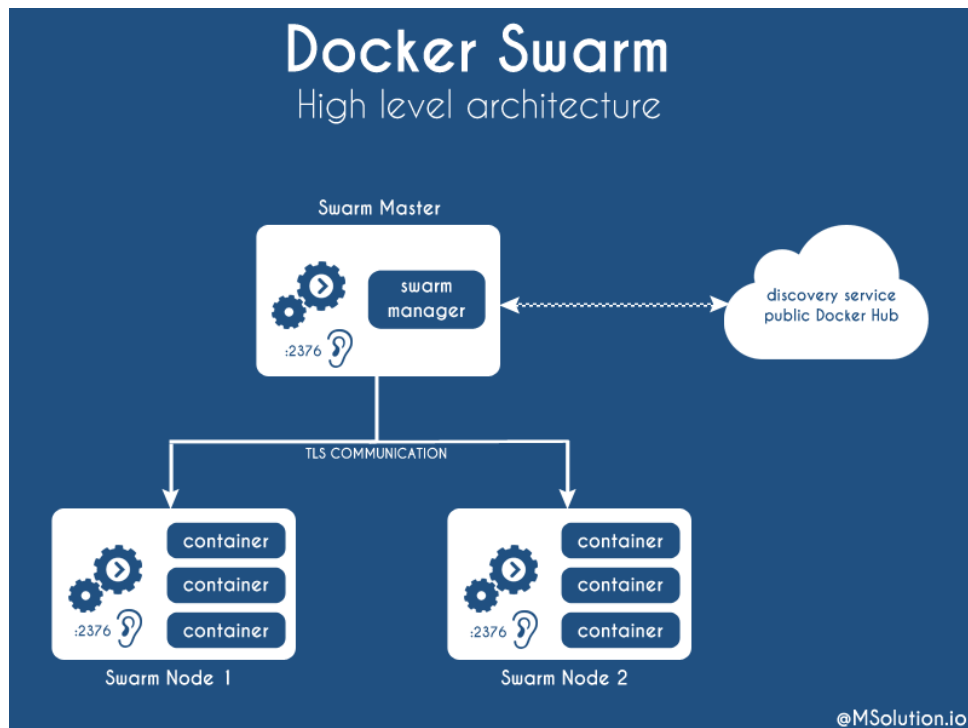
1

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

Formation Docker



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 maîtresse 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 utilisiez 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 définis par votre service.

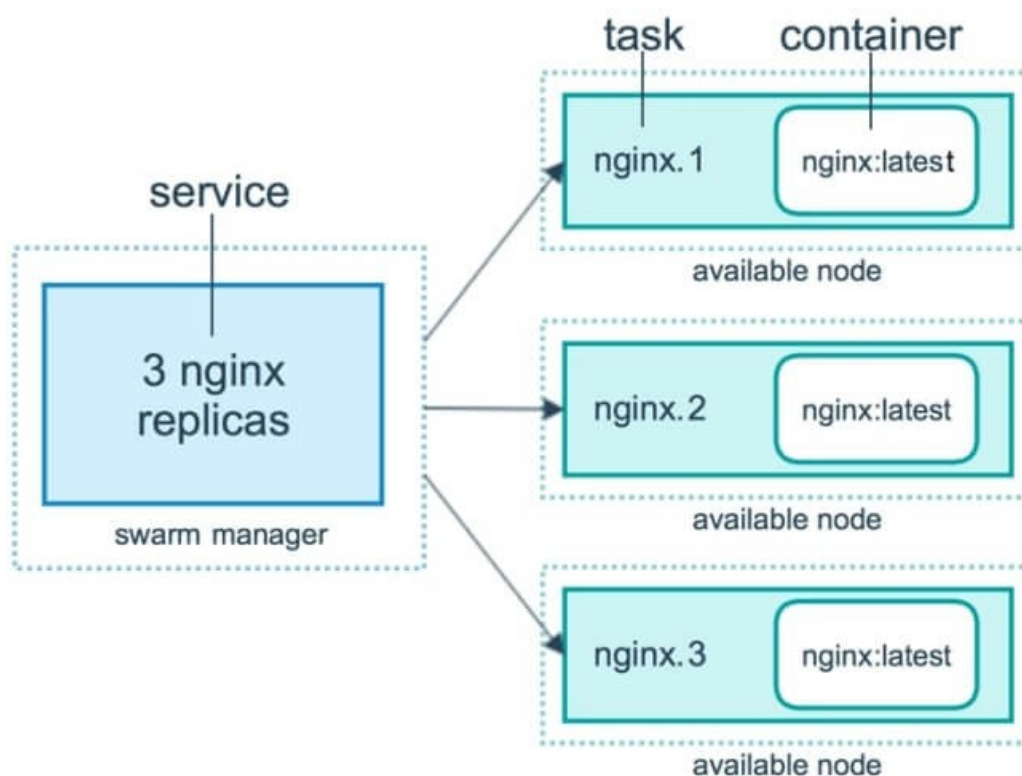
Résumons les différents concepts de Docker Swarm

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 équilibreur de charge sera mis en place automatiquement par votre manager Swarm afin de supporter les grandes charges de travaux.



Formation Docker

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
ks1306m24xlc0xhatoczm0o	docker-worker10	Ready	Active		20.10.8
pbmvt1t9go4g6xgjg68or409	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
ks1306m24xlc0xhatoczm0o	docker-worker10	Ready	Active	Reachable	20.10.8

TP -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 a 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
```

Formation Docker

```
1/3: running [=====>]
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

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

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
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	
k0nywar5aauw	nginx-stack_nginx.2	nginx:latest	docker-leader	Running	Running	3 minutes ago	

Il n'y a plus de « task » en shutdown

I. Introduction à Kubernetes

Qu'est-ce que Kubernetes ?

Kubernetes est un système open-source d'orchestration de conteneurs qui facilite le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Il a été développé à l'origine par Google et est désormais maintenu par **la Cloud Native Computing Foundation** (CNCF). Kubernetes permet de gérer efficacement des clusters de machines virtuelles ou physiques, en fournissant une plateforme cohérente pour exécuter et gérer des applications conteneurisées.

L'objectif principal de Kubernetes est de simplifier la gestion des applications distribuées, en fournissant des fonctionnalités telles que l'automatisation du déploiement, la gestion des ressources, la répartition de charge, la récupération automatique en cas de défaillance, la gestion des secrets, la mise à l'échelle automatique, la gestion des configurations, etc. Il offre également une interface déclarative pour spécifier l'état souhaité du système et laisse Kubernetes se charger de son exécution.

Pourquoi utiliser Kubernetes ?

Kubernetes présente de nombreux avantages et raisons pour lesquelles il est largement utilisé dans l'industrie :

Scalabilité : Kubernetes facilite la mise à l'échelle horizontale et verticale des applications, ce qui permet d'ajuster dynamiquement les ressources en fonction de la charge de travail.

Haute disponibilité : Kubernetes offre des mécanismes de récupération automatique en cas de défaillance des nœuds ou des pods, garantissant ainsi une disponibilité élevée des applications.

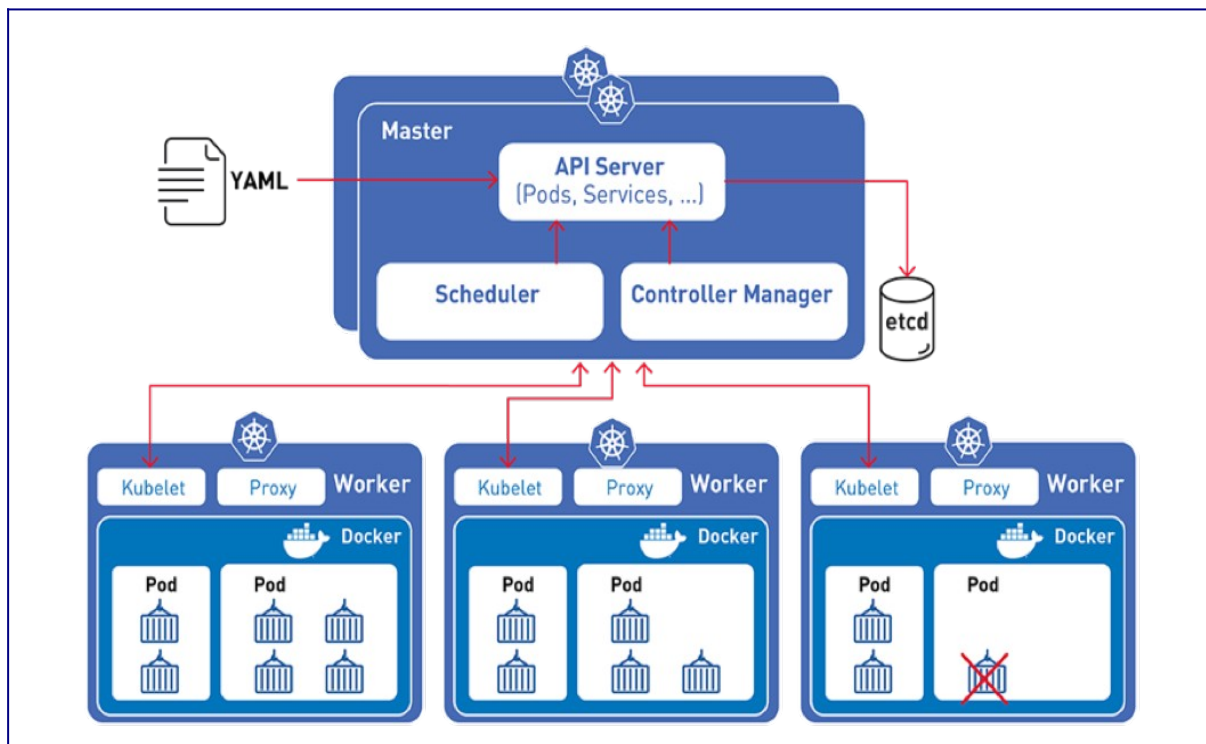
Déploiement et gestion facilités : Kubernetes permet de déployer et de gérer facilement des applications conteneurisées sur des clusters, ce qui réduit le temps et les efforts nécessaires pour la mise en production des applications.

Portabilité : Kubernetes est compatible avec différents fournisseurs de services cloud et peut être exécuté sur site ou dans le cloud, offrant ainsi une portabilité des charges de travail.

Écosystème riche : Kubernetes bénéficie d'une vaste communauté et d'un écosystème d'outils et de services complémentaires, ce qui facilite l'intégration avec d'autres technologies et permet d'étendre les fonctionnalités de base.

Architecture de base de Kubernetes

Kubernetes est basé sur une architecture maître-esclave qui comprend les composants suivants :



Cluster : Un cluster Kubernetes est composé de plusieurs nœuds, qui peuvent être des machines virtuelles ou physiques, sur lesquels les conteneurs seront exécutés.

Maître (Master) : Le maître est responsable de la gestion globale du cluster Kubernetes. Il comprend plusieurs composants clés :

API Server : Il expose l'API Kubernetes et permet aux utilisateurs et aux autres composants de communiquer avec le cluster.

Scheduler : Il est responsable de la planification des pods sur les nœuds du cluster en fonction des ressources disponibles et des contraintes spécifiées.

Controller Manager : Il surveille l'état global du cluster et prend des mesures pour le maintenir dans l'état souhaité.

etcd : Il est une base de données distribuée utilisée pour stocker la configuration du cluster et l'état souhaité.

II. Installation de Kubernetes

Configuration des prérequis

Avant d'installer Kubernetes, il est essentiel de s'assurer que votre environnement remplit les prérequis nécessaires. Voici quelques points importants à prendre en compte :

Système d'exploitation : Kubernetes est compatible avec divers systèmes d'exploitation, notamment Linux, Windows et macOS. Assurez-vous d'utiliser une version prise en charge du système d'exploitation sur vos nœuds.

Matériel et ressources : Vérifiez les spécifications matérielles recommandées pour les nœuds du cluster Kubernetes. Assurez-vous que vous disposez de suffisamment de ressources (CPU, mémoire, espace de stockage) pour exécuter les applications et les conteneurs.

Réseau : Assurez-vous que les nœuds du cluster peuvent communiquer entre eux via le réseau. Configurez des adresses IP statiques ou utilisez un service DNS pour résoudre les noms de domaine.

Docker : Kubernetes utilise Docker (ou un autre runtime de conteneurisation) pour exécuter les conteneurs. Assurez-vous d'installer Docker et de le configurer correctement sur chaque nœud du cluster.

Installation de Kubernetes sur différents systèmes d'exploitation

La procédure d'installation de Kubernetes peut varier en fonction du système d'exploitation. Voici quelques méthodes courantes pour l'installation :

1. Linux :

Utilisation de kubeadm : Kubeadm est un outil populaire pour l'installation de Kubernetes sur Linux. Il permet de configurer rapidement un cluster Kubernetes en gérant les tâches d'initialisation, de planification et de déploiement des composants principaux.

Utilisation de distributions spécifiques : Certains projets, tels que kops pour Kubernetes sur AWS ou Rancher pour la gestion de clusters Kubernetes, fournissent des outils spécifiques pour simplifier l'installation sur des plates-formes spécifiques.

Il est recommandé de consulter la documentation officielle de Kubernetes et les ressources spécifiques au système d'exploitation pour obtenir des instructions détaillées sur l'installation.

Utilisation de solutions de gestion de cluster

En plus des méthodes d'installation traditionnelles, il existe des solutions de gestion de cluster Kubernetes qui simplifient davantage le déploiement et la gestion de Kubernetes. Voici quelques exemples :

kubeadm : Kubeadm est un outil en ligne de commande fourni par Kubernetes lui-même. Il simplifie la création et la configuration d'un cluster Kubernetes en automatisant les tâches d'initialisation, de planification et de déploiement des composants principaux.

kops : Kops est un outil open-source qui permet de créer, mettre à l'échelle et gérer des clusters Kubernetes sur des infrastructures cloud telles que AWS (Amazon Web Services).

Kubernetes as a Service (KaaS)

Kubernetes as a Service (KaaS) est une offre de services cloud proposée par divers fournisseurs de cloud, tels que Google Cloud Platform (GCP), Amazon Web Services (AWS), Microsoft Azure, etc. Avec KaaS, vous n'avez pas à vous soucier de l'installation, de la configuration et de la gestion de l'infrastructure sous-jacente. Le fournisseur de cloud s'occupe de tout cela, vous permettant de vous concentrer sur le déploiement et la gestion de vos applications sur Kubernetes.

Les avantages de l'utilisation de KaaS sont les suivants :

Facilité de déploiement : Vous pouvez créer un cluster Kubernetes en quelques clics, sans avoir à configurer manuellement les nœuds ou les composants de Kubernetes. Le fournisseur de cloud gère toute l'infrastructure pour vous.

Évolutivité : Les fournisseurs de KaaS offrent des options d'évolutivité automatique pour les clusters, ce qui signifie que vous pouvez facilement augmenter ou diminuer la capacité de votre cluster en fonction de la demande.

Maintenance simplifiée : Les fournisseurs de KaaS se chargent des mises à jour et des correctifs de sécurité pour les composants de Kubernetes. Vous n'avez pas à vous soucier de maintenir et de mettre à jour l'infrastructure.

Intégration avec d'autres services cloud : Les fournisseurs de KaaS offrent souvent une intégration transparente avec d'autres services cloud, tels que les bases de données, les services de stockage, les services de messagerie, etc., ce qui facilite le développement et le déploiement d'applications cloud natives.

Concepts clés de Kubernetes

Pods :

Un pod est l'unité de base dans Kubernetes. C'est une abstraction logique qui représente un groupe d'un ou plusieurs conteneurs qui sont toujours déployés ensemble sur un même nœud. Les pods sont éphémères et peuvent être créés, mis à jour et supprimés dynamiquement par Kubernetes en fonction des besoins. Les conteneurs d'un pod partagent le même espace réseau et peuvent communiquer entre eux via localhost. Les pods sont généralement utilisés pour regrouper des conteneurs étroitement liés, tels que les composants d'une application.

Services :

Un service est une abstraction qui définit un ensemble de pods et une politique d'accès à ces pods. Il fournit une adresse IP stable et un nom DNS pour permettre aux autres composants ou services d'accéder aux pods de manière transparente. Les services peuvent être de différents types, tels que le service ClusterIP pour l'accès interne au cluster, le service NodePort pour l'accès externe via un port spécifique sur tous les nœuds, ou le service LoadBalancer pour l'équilibrage de charge externe via un équilibreur de charge externe.

Déploiements :

Un déploiement est une ressource Kubernetes qui définit l'état souhaité d'une application et gère le processus de déploiement, de mise à jour et de mise à l'échelle des pods. Il permet de spécifier le nombre de répliques (instances) de l'application à exécuter, la stratégie de mise à jour, les contraintes de ressources, etc. L'utilisation de déploiements permet de garantir que l'application est toujours en cours d'exécution selon l'état spécifié, et permet de réaliser des mises à jour sans interruption de service.

StatefulSet

Un StatefulSet est un contrôleur Kubernetes spécifique utilisé pour gérer les applications étatiques (Stateful Applications). Contrairement aux déploiements qui sont conçus pour les applications sans état, les StatefulSets sont utilisés lorsque les pods doivent conserver un état unique et stable, tels que les bases de données, les systèmes de fichiers distribués, etc.

Réplicas et mise à l'échelle :

Les réplicas désignent les instances des pods qui sont créées par un déploiement. En spécifiant le nombre de réplicas dans un déploiement, Kubernetes veillera à ce que le nombre spécifié de pods soit en cours d'exécution en tout temps. La mise à l'échelle est le processus d'ajustement dynamique du nombre de réplicas en fonction de la charge de travail. Kubernetes prend en charge la mise à l'échelle automatique, où il ajuste automatiquement le nombre de réplicas en fonction de métriques prédéfinies telles que l'utilisation du CPU ou la latence, ainsi que la mise à l'échelle manuelle où l'utilisateur peut modifier le nombre de réplicas manuellement.

Secrets et ConfigMaps :

Les secrets et les ConfigMaps sont utilisés pour gérer les données sensibles ou de configuration dans Kubernetes. Les secrets sont utilisés pour stocker des informations sensibles, telles que les clés d'API, les informations d'identification, les certificats, etc. Ils sont encodés en base64 pour la sécurité. Les ConfigMaps sont utilisées pour stocker des données de configuration qui peuvent être utilisées par les conteneurs, tels que les variables d'environnement, les fichiers de configuration, etc. Ils peuvent être montés en tant que volumes ou injectés directement dans les conteneurs.

Volumes et persistance des données :

Les volumes sont utilisés pour la persistance des données dans Kubernetes. Ils permettent aux conteneurs d'accéder et de stocker des données de manière durable, même lorsque les pods sont recréés ou redéployés. Kubernetes offre différentes options de volumes pour répondre aux besoins de persistance des données

Namespaces :

Les namespaces (espaces de noms) sont une fonctionnalité clé de Kubernetes qui permettent de diviser un cluster en plusieurs espaces logiques isolés. Chaque namespace agit comme un conteneur virtuel qui sépare les ressources et les objets Kubernetes en groupes distincts. Cela permet d'organiser et de gérer efficacement les applications, les équipes ou les environnements au sein d'un même cluster Kubernetes.

Découverte de Kubernetes

Découverte de la 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.

La commande Kubectl effectue une action sur une ou des ressources.

Formation Docker

Pour créer un pod :

```
ludo@formation~$ kubectl run --image=nginx nginx-ludo
pod/nginx-ludo created
```

Pour afficher des ressources :

```
ludo@formation~$ kubectl get RESSOURCES
```

```
ludo@formation~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-ludo	1/1	Running	0	21s

Pour modifier la sortie

```
ludo@formation~$ kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-ludo	1/1	Running	0	36s	192.168.30.126	worker02

Pour obtenir des détails sur la ressource

```
ludo@formation~$ kubectl describe pod nginx-ludo
```

```
Name:          nginx-ludo
Namespace:     default
Priority:      0
Service Account: default
Node:         worker02/10.0.0.12
...
...
```

Events:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	Scheduled	77s	default-scheduler	Successfully assigned default/nginx-ludo to worker02

Supprimer des ressources

```
ludo@formation~$ kubectl delete pod nginx-ludo
pod "nginx-ludo" deleted
```

Formation Docker

Lister les ressources de l'api des ressources

```
ludo@formation~$ kubectl api-resources
```

Création d'un déploiement

```
ludo@formation~$ kubectl create deployment prenom-nginx --image=nginx \
--replicas=2
```

Affichage des ressources créées

```
ludo@formation~$ kubectl get deployment prenom-nginx
```

Afficher toutes les ressources

```
ludo@formation~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/test-nginx-6b4d69f6f8-4jxm8	1/1	Running	0	5m14s
pod/test-nginx-6b4d69f6f8-pngr5	1/1	Running	0	5m14s

Suppression d'un pod

```
ludo@formation~$ kubectl delete pod test-nginx-6b4d69f6f8-4jxm8
```

Afficher de nouveau toutes les ressources

```
ludo@formation~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/test-nginx-6b4d69f6f8-pngr5	1/1	Running	0	6m26s
pod/test-nginx-6b4d69f6f8-qg6g8	1/1	Running	0	4s

Le pod a été recrée !

On supprime le déploiement

```
ludo@formation~$ kubectl delete deployment prenom-nginx
deployment.apps "test-nginx" deleted
```

Le pod a été recrée !

Le Pod

Un pod est la plus petite unité déployable dans Kubernetes. Il représente un ou plusieurs conteneurs étroitement liés qui partagent des ressources, un espace de stockage, une adresse IP et des options de gestion communes.

Les pods sont souvent utilisés pour exécuter des microservices et sont généralement éphémères, ce qui signifie qu'ils peuvent être créés, mis à jour, redémarrés ou supprimés à mesure que nécessaire.

Les conteneurs à l'intérieur d'un pod partagent le même cycle de vie et sont planifiés et déployés ensemble sur un nœud du cluster Kubernetes.

Les sondes

Les sondes sont des mécanismes utilisés pour vérifier l'état d'un conteneur à l'intérieur d'un pod et prendre des mesures en conséquence.

Il existe trois types de sondes couramment utilisés :

Vous avez raison, et je m'excuse pour la confusion précédente. Voici une reformulation précise de la sonde de démarrage (Startup Probe) dans Kubernetes :

La sonde de démarrage (Startup Probe) est utilisée pour vérifier si une application à l'intérieur d'un conteneur a suffisamment de temps pour démarrer correctement. Elle diffère des sondes de disponibilité et d'activité (Liveness Probe et Readiness Probe) qui sont exécutées périodiquement une fois que le conteneur est déjà en cours d'exécution. Lorsqu'un pod contenant des conteneurs est créé ou redéployé, la sonde de démarrage retarde le début de la vérification de disponibilité et d'activité jusqu'à ce que le délai spécifié se soit écoulé. Cela permet à l'application de démarrer complètement avant que Kubernetes ne commence à vérifier son état. Une fois que le délai de démarrage spécifié est atteint, la sonde de démarrage cesse de retarder les autres sondes et les sondes de disponibilité et d'activité (Liveness Probe et Readiness Probe) commencent à être exécutées périodiquement selon leur propre configuration. La sonde de démarrage est utile lorsque l'application a besoin d'un temps supplémentaire pour se préparer avant d'être considérée

comme prête à traiter les requêtes ou à être considérée comme active. Elle permet d'éviter les faux positifs lorsqu'une application est encore en cours de démarrage et n'est pas encore prête à gérer le trafic.

En résumé, la sonde de démarrage (Startup Probe) permet de retarder la vérification de disponibilité et d'activité jusqu'à ce que le délai de démarrage spécifié soit écoulé, donnant à l'application le temps nécessaire pour se préparer avant d'être considérée comme prête à traiter les requêtes.

Liveness Probe (Sonde de vérification d'activité) :

La Liveness Probe est utilisée pour vérifier si un conteneur à l'intérieur d'un pod est en cours d'exécution. Elle effectue des vérifications périodiques de l'état du conteneur, en envoyant des requêtes vers une URL spécifiée, un port TCP ou en exécutant une commande à l'intérieur du conteneur. Si la Liveness Probe échoue (c'est-à-dire si le conteneur ne répond pas ou renvoie une erreur), Kubernetes considère le conteneur comme en échec et tente de le redémarrer automatiquement dans le même pod.

L'objectif de la Liveness Probe est de maintenir la disponibilité de l'application en redémarrant les conteneurs en cas d'échec.

Readiness Probe (Sonde de disponibilité) :

La Readiness Probe est utilisée pour vérifier si un conteneur à l'intérieur d'un pod est prêt à recevoir du trafic. Elle effectue des vérifications périodiques de l'état du conteneur, en envoyant des requêtes vers une URL spécifiée, un port TCP ou en exécutant une commande à l'intérieur du conteneur. Si la Readiness Probe échoue, Kubernetes considère le conteneur comme non prêt et le retire de la rotation du trafic. Ainsi, le pod ne recevra pas de nouvelles requêtes jusqu'à ce que la sonde de disponibilité réussisse.

L'objectif de la Readiness Probe est de garantir que seuls les conteneurs prêts à servir le trafic reçoivent effectivement les requêtes, évitant ainsi les interruptions potentielles pour les utilisateurs finaux.

En résumé, la Liveness Probe vérifie l'activité et la santé d'un conteneur en redémarrant le pod en cas d'échec, tandis que la Readiness Probe vérifie la disponibilité d'un conteneur et le déconnecte du service si la sonde échoue, afin d'éviter d'envoyer du trafic à un conteneur non prêt. Ces deux sondes sont essentielles pour garantir la fiabilité et la disponibilité des applications dans Kubernetes.

La gestion des ressources :

Les ressources dans Kubernetes font référence aux limites de capacité et aux demandes de ressources des conteneurs à l'intérieur d'un pod.

Les requête de ressources indiquent les ressources minimales requises pour qu'un conteneur puisse s'exécuter correctement. Elles sont utilisées par le planificateur Kubernetes pour allouer les ressources aux nœuds appropriés.

Les limites de ressources définissent la quantité maximale de ressources qu'un conteneur peut consommer. Elles sont utilisées pour garantir un équilibre des ressources et empêcher qu'un conteneur ne monopolise excessivement les ressources du cluster.

Les ressources peuvent inclure le CPU, la mémoire, le stockage, la bande passante réseau, etc.

Ces concepts jouent un rôle crucial dans la gestion des applications dans Kubernetes. Les pods permettent de regrouper et de gérer les conteneurs, les sondes permettent de surveiller et de maintenir la santé des conteneurs, et les ressources garantissent l'utilisation efficace et équilibrée des ressources du cluster. En comprenant ces concepts, vous serez en mesure de déployer et de gérer des applications de manière plus efficace sur Kubernetes.