



MASTER RESEAU SECURITE INFORMATIQUE

Projet DevOps

THEME

Docker et Apache2 en tant que proxy pour le déploiement et la gestion d'applications web

Université Nongo Conakry
Groupe 05

Table des matières

CHAPITRE 1 Docker	2
Historique de Docker.....	2
L'origine de Docker	2
Définition de Docker	2
Les trois concept clés de Docker.....	3
Quelques technologies qui l'ont précédées	4
CHAPITRE 2 Docker-Compose	6
Fonctionnalité de docker docker-compose	7
Utilité de changement.....	7
Sécurité : Docker et Docker-compose.....	8
Sécurité des Conteneurs docker	8
Inconvénients docker.....	9
CHAPITRE 3 Déploiement	11
Proxy apache2	13
Le site web construit avec l'image qui fonctionnera sur python :	18
Un déploiement de Mkdocs avec l'ensemble des éléments construits dans le dockerfile	22
Le site web WordPress	29
Schema illustratif	42
Conclusion :	44

CHAPITRE I : Docker

Historique de Docker

Le logiciel publié à l'origine sous le nom « Docker » fut développé sur la base de la technologie Linux Container (LXC). LXC fut ensuite remplacée par le libcontainer de Docker. De nouveaux composants logiciel ont été ajoutés alors que Docker a continué à croître et à devenir le standard pour la virtualisation basée sur les conteneurs. ContainerD a notamment émergé du développement de Docker en tant que moteur d'exécution de conteneur avec l'implémentation standard runC. Aujourd'hui, ces projets sont gérés par la Cloud Native Computing Foundation (CNCF) et l'Open Container Initiative (OCI).

Outre l'équipe consacrée à Docker, des entreprises leader dans le monde de la tech telles que Cisco, Google, Huawei, IBM, Microsoft, ou encore Red Hat sont impliquées dans le développement de Docker et de technologies connexes. Parmi les évolutions les plus récentes, on notera le fait que Windows est désormais également utilisé en tant qu'environnement natif pour les conteneurs Docker, en plus du noyau Linux.

L'origine de Docker

Docker a été créé en 2013 par Solomon Hykes. Cette plateforme permet de lancer des applications dans des conteneurs logiciels. Contrairement à la virtualisation traditionnelle, Docker ne nécessite pas de système d'exploitation séparé pour chaque conteneur. Au lieu de cela, il s'appuie sur les fonctionnalités du noyau et utilise l'isolation de ressources pour exécuter les processus de manière isolée. Ainsi, Docker offre une flexibilité et une portabilité accrues pour l'exécution d'applications sur différentes machines hôtes.

Définition :

Docker est une plateforme open source qui permet aux développeurs de créer, déployer, exécuter, mettre à jour et gérer des conteneurs, des composants exécutables standardisés qui combinent le code source d'une application avec les bibliothèques de système d'exploitation et les dépendances nécessaires pour exécuter ce code dans n'importe quel environnement.

Les trois concepts clés de Docker

Il existe trois concepts clés dans Docker : les conteneurs, les images et les fichiers Docker (Dockerfile). Dans cette partie, nous découvrirons ces trois concepts.

Ces trois concepts forment la chaîne de déploiement de conteneurs et possèdent deux sens de lecture. Dans notre cas, nous allons partir du résultat final, les conteneurs, pour remonter jusqu'à leur origine, les Dockerfile, en passant par les images qui se trouvent entre les deux.

Premier concept clé de Docker : les conteneurs

Un conteneur est donc un espace dans lequel une application tourne avec son propre environnement. Les applications qu'un conteneur peut faire tourner sont de tous types : site web, API, db, etc.

Chaque conteneur est une instance d'une image. Il possède son propre environnement d'exécution et donc ses propres répertoires. Nous verrons comment déployer un conteneur dans le second article.

La force des conteneurs Docker réside dans le grand panel de configurations qui nous est proposé. Nous pouvons ainsi changer le port d'un conteneur à l'intérieur du système Docker mais également les faire facilement communiquer entre eux.

Deuxième concept clé de Docker : les images

Nous avons abordé les images lorsque nous avons parlé des conteneurs. Mais d'ailleurs, qu'est-ce qu'une image ?

Les images représentent le contexte que plusieurs conteneurs peuvent exécuter. Elles sont aux conteneurs ce que les classes sont aux objets en Programmation Orientée Objet : un moule.

Par contexte, il faut entendre la disposition des dossiers, la disponibilité de certaines librairies, le bindage de certains ports en interne et en externe du conteneur mais également un ensemble de commandes à exécuter au lancement d'un conteneur.

Tout comme les conteneurs, nous regarderons les différentes manières d'obtenir des images dans le second article.

Troisième concept clé de Docker : le Dockerfile

Et voici la troisième et dernière pièce du puzzle : le Dockerfile.

Un Dockerfile est un fichier qui liste les instructions à exécuter pour build une image. Il est lu de haut en bas au cours du processus de build.

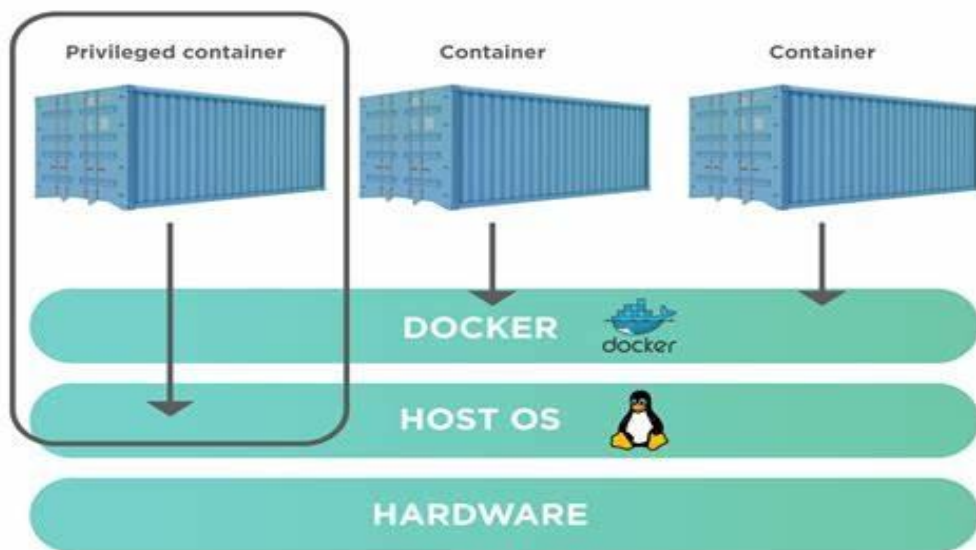
Quelques technologies qui l'ont précédées :

Les technologies qui ont précédé docker :

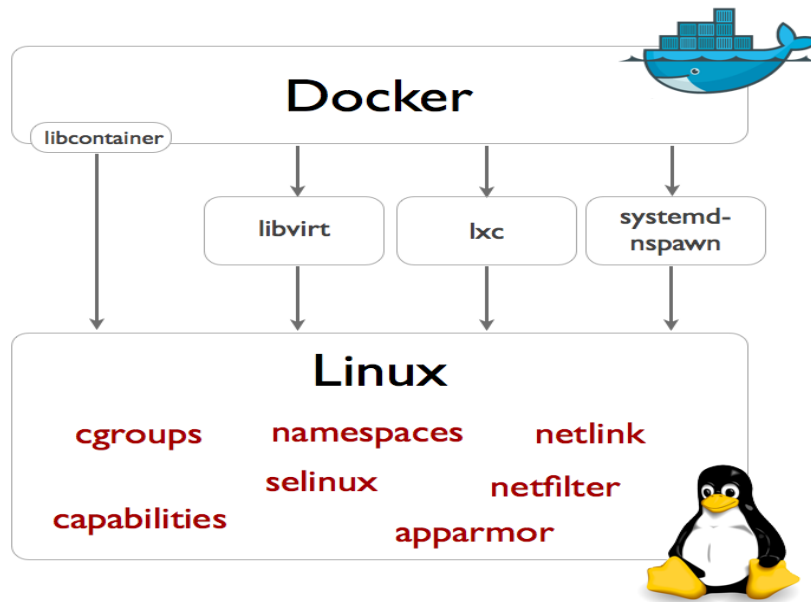
Avant Docker, les entreprises utilisaient souvent des machines virtuelles (VM) pour exécuter des applications. Ces machines virtuelles peuvent émuler des ordinateurs physiques, permettant aux développeurs de transformer un serveur en plusieurs serveurs. Cependant, cette approche présente certains inconvénients, notamment en termes de consommation de ressources et de lourdeur. Les VM nécessitent un système d'exploitation complet pour chaque instance, ce qui peut entraîner une surcharge en termes de stockage et de mémoire.

Docker, en revanche, a introduit une approche plus légère et plus efficace : la virtualisation basée sur des conteneurs. Voici un bref historique de l'évolution de Docker :

1. **LXC (Linux Containers)** : La première version de la technologie Docker est née sous dotCloud. À l'origine, elle était utilisée pour faire tourner la plate-forme en tant que service (PaaS). Par la suite, Solomon Hykes, le fondateur de Docker, a décidé de réécrire cette technologie en open source (en langage Go) et de la partager avec la communauté.



2. **Libcontainer** : Docker a remplacé LXC par libcontainer. Cette évolution a permis à Docker de devenir le standard pour la virtualisation basée sur les conteneurs. Libcontainer est devenu le moteur d'exécution de conteneur avec l'implémentation standard runC.



3. **ContainerD** : Issu du développement de Docker, ContainerD est un moteur d'exécution de conteneur qui a émergé. Aujourd'hui, des projets tels que ContainerD sont gérés par la Cloud Native Computing Foundation (CNCF) et l'Open Container Initiative (OCI).
4. **Windows** : Initialement basé sur le noyau Linux, Docker s'est étendu pour prendre en charge Windows en tant qu'environnement natif pour les conteneurs, en plus du noyau Linux.

En somme, Docker a révolutionné la manière dont les applications sont développées, distribuées et exécutées, en proposant une approche plus légère et plus flexible que les machines virtuelles traditionnelles

Architectures des technologies qui ont précédées docker

CHAPITRE II : Docker-compose

Docker Compose est un outil utilisé sur Docker, afin de pouvoir mieux gérer ses conteneurs. En effet, lorsqu'on a deux ou trois conteneurs à coordonner, ce n'est pas trop compliqué, mais plus le nombre de conteneurs augmente, et plus ça devient difficile. C'est là que Docker Compose intervient.

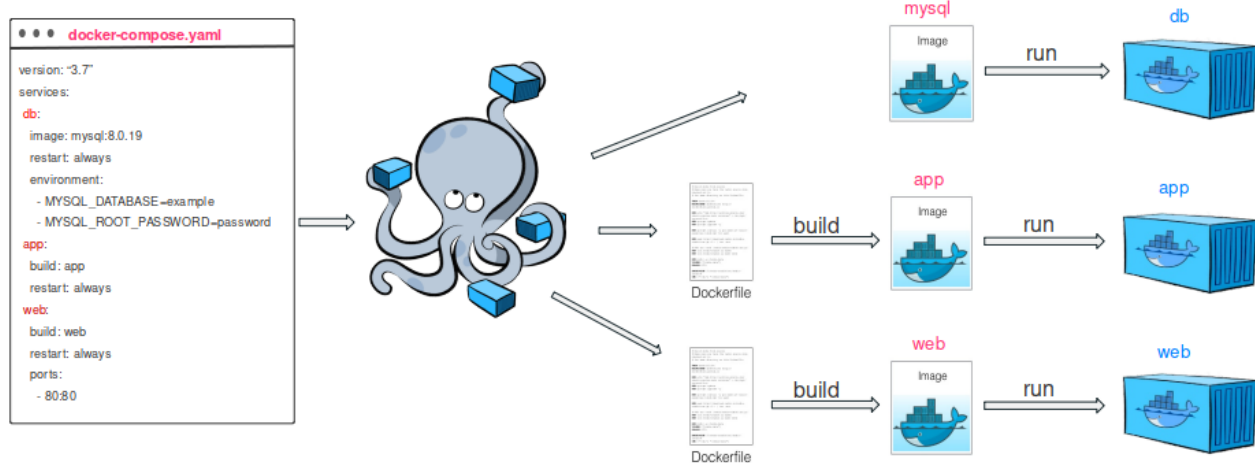
En effet, Docker Compose apporte cette coordination recherchée, et permet à ses utilisateurs d'être plus efficaces, à l'aide d'une meilleure gestion des dépendances (réseau, volumes), et cela aussi car Docker Compose intervient en matière de service.

Qu'est-ce qu'un service ? Un service comprend plusieurs conteneurs, par plusieurs on peut aussi dire vraiment beaucoup de conteneurs, car Docker Compose est un orchestrateur Docker. Docker compose orchestre une série de conteneurs, avec des actions qui auront des interactions entre les uns et les autres.

Comment reconnaît-on un fichier Docker Compose ? C'est un fichier, en YAML (d'extension .yaml), qui peut facilement être partagé soit par git ou par mail, etc. Un fichier Docker Compose se compose comme ceci :

```
services: //les éléments gérer par docker-compose
premierService: //nom du service, comprend un conteneur
image: mysql
container_name: MySQL //nom du conteneur
```

Dans ce fichier on peut y mettre tous les services souhaités. On déclare son nom, son image, et le nom du conteneur. On peut rajouter toute sorte d'options à notre fichier, comme restart : always par exemple, qui permet de systématiquement redémarrer le conteneur dès lors qu'il s'arrêtera tout seul. Même si docker stop est utilisé, le conteneur sera toujours existant. Docker Compose donc crée un conteneur à partir d'une image mysql, et permet une certaine garantie que le service va fonctionner.



Fonctionnalité de docker et docker-compose

La plateforme de conteneurisation repose sur sept composants principaux. Le Docker Engine est un outil client-serveur sur lequel repose la technologie de container pour prendre en charge les tâches de création d'applications basées container. Le moteur crée un processus daemon server-side permettant d'héberger les images, les containers, les réseaux et les volumes de stockage. Ce daemon fournit aussi une interface SLI client-side permettant aux utilisateurs d'interagir avec le daemon via l'API de la plateforme.

Les containers créés sont appelés Dockerfiles. Le composant Docker Compose permet de définir la composition des composants au sein d'un container dédié. Le Docker Hub est un outil SaaS permettant aux utilisateurs de publier et de partager des applications basées container via une bibliothèque commune.

Le mode Docker Swarm du Docker Engine prend en charge l'équilibrage des charges des clusters. Ainsi, les ressources de plusieurs hôtes peuvent être rassemblées pour agir comme un seul ensemble. Ainsi, les utilisateurs peuvent rapidement échelonner le déploiement de containers.

Utilité et changement :

C'est l'outil qui a démocratisé la conteneurisation et qui est le plus répandu. De plus, de nombreux outils open source se basent dessus, tels que Molecule (qui permet de tester du code Ansible), ou Kubernetes (un orchestrateur de conteneurs)

Elle permet de développer des applications de façon plus efficace, en utilisant moins de ressources, et de déployer ces applications plus rapidement.

Docker vous permet d'envoyer du code plus rapidement, de standardiser les opérations de vos applications, de migrer aisément du code et de faire des économies en améliorant l'utilisation des ressources

Docker permet d'encapsuler toutes les dépendances relatives au système d'information, et ceci tout en conservant

Sécurité : Docker et Docker-compose

La sécurité des conteneurs Docker et Docker Compose est essentielle pour garantir un environnement robuste et protégé. Voici quelques bonnes pratiques à suivre pour renforcer la sécurité de vos conteneurs :

1. **Utilisez des quotas de ressources** : Configurez des limites pour la consommation de mémoire et de CPU dans chaque conteneur. Cela améliore l'efficacité de l'environnement et limite les dénis de service en cas d'infection par du contenu malveillant¹.
2. **Évitez d'exécuter les conteneurs en tant que root** : Respectez le principe du moindre privilège en limitant l'accès des applications aux ressources requises. Exécuter des conteneurs en tant qu'utilisateur root peut augmenter la surface d'attaque¹.
3. **Sécurisez vos registres de conteneurs Docker** : Assurez-vous que vos registres de conteneurs sont solides et fiables. Utilisez des images de base officielles et signées, et accédez à la source du code¹.
4. **Mettez en œuvre la segmentation du réseau** : Concevez vos API et réseaux en gardant la sécurité à l'esprit. La segmentation du réseau limite la propagation des attaques².
5. **Surveillez et enregistrez l'activité des conteneurs** : Utilisez des outils de surveillance pour détecter toute activité suspecte et enregistrer les journaux. Cela vous permet de réagir rapidement en cas d'incident².

En suivant ces meilleures pratiques, vous pouvez renforcer la sécurité de vos conteneurs Docker et Docker Compose. N'oubliez pas de maintenir vos systèmes à jour et de rester vigilant face aux nouvelles menaces³. Si vous avez besoin d'informations plus détaillées, n'hésitez pas à me le faire savoir

Sécurité des conteneurs Docker

La sécurisation d'un conteneur Docker est identique à la sécurité d'autres conteneurs. Elle requiert une approche tout compris, qui sécurise tous les éléments de l'hôte au réseau. De nombreuses organisations ont des difficultés à assurer la sécurité des conteneurs en raison de leurs parties mobiles ; un niveau de vigilance rudimentaire ne suffit pas.

Éléments à prendre en compte

Voici quelques éléments à prendre en compte lorsque vous sécurisez vos conteneurs Docker :

- Utilisez des quotas de ressources
- Les conteneurs Docker ne doivent pas être exécutés en tant que root
- Assurez la sécurité de vos registres de conteneur Docker
- Utilisez une source fiable

- Accédez à la source du code
- Concevez des API et des réseaux en gardant la sécurité à l'esprit L'analyse de sécurité des images Docker est essentielle pour identifier les vulnérabilités connues dans les paquets répertoriés dans vos images Docker. Voici quelques points importants à considérer concernant les vulnérabilités de Docker et Docker Compose :
- **Logiciels obsolètes** : Les images Docker peuvent contenir des logiciels obsolètes, ce qui constitue l'une des vulnérabilités les plus courantes. Il est crucial de maintenir à jour les dépendances et les bibliothèques utilisées dans vos conteneurs.
- **Faillies Leaky Vessels** : Un ensemble de vulnérabilités appelé "Leaky Vessels" a été découvert dans l'outil CLI runc, utilisé notamment dans Docker et Kubernetes. Ces failles permettent à un attaquant de s'échapper du conteneur et d'accéder aux données de l'hôte physique.
- **Connexions réseau non sécurisées** : Assurez-vous que les connexions réseau entre les conteneurs et l'hôte sont sécurisées. Évitez d'exposer des ports non nécessaires et configurez les règles de pare-feu pour limiter l'accès.

Meilleures pratiques de sécurité : Suivez les meilleures pratiques pour garantir la sécurité de vos images Docker. Cela inclut la création d'une image Docker sécurisée, la réduction des vulnérabilités introduites par les images de base Docker et l'utilisation de Dockerfiles sécurisés.

N'oubliez pas de scanner vos images de conteneurs pour détecter les vulnérabilités et de les corriger avant de les pousser vers Docker Hub ou tout autre registre. Si vous utilisez Snyk, vous pouvez intégrer l'analyse de sécurité directement dans vos outils Docker pour une sécurité renforcée

Les inconvénients de Docker ?

Docker a des avantages certes, mais il a aussi son lot d'inconvénients.

On peut, par exemple, citer les problèmes de sécurité induits par le fait que tous les conteneurs, même isolés, tournent sur le même OS. De ce fait, si un conteneur est infecté d'une manière ou d'une autre par un virus, il peut propager le virus à l'OS qui fait tourner Docker et par conséquent faire tomber toute l'infrastructure. C'est un problème qu'on ne retrouve pas dans le cadre de la virtualisation où les VM possèdent chacune un OS isolé.

Il devient également très difficile de gérer un Docker lorsque le nombre de conteneurs utilisés devient grand. Pour permettre une gestion plus simplifiée des conteneurs, plusieurs applications peuvent être utilisées parmi lesquels Kubernetes.

Les conteneurs tournent sur un même OS, limitant ainsi les bibliothèques utilisables à celles dont cet OS dispose et donc cassant l'aspect portabilité de Docker, contrairement à la

virtualisation où chaque VM a son propre OS et donc rend disponibles ses librairies aux applications associées.

Il est très difficile de persister des données au sein de Docker. En effet, lorsqu'un conteneur est supprimé, l'ensemble de son contexte, et par extension ses répertoires, sont supprimés. Il existe deux méthodes qui permettent d'accéder à un répertoire extérieur à conteneur et d'y réaliser des opérations mais elles s'avèrent insuffisantes car difficiles et lourdes à mettre en place.

Vous l'aurez compris, la virtualisation propose parfois de meilleures solutions que la conteneurisation face à certaines problématiques. **Il faut ainsi voir Docker comme un moyen de déployer rapidement des applications qui auront pour but d'évoluer dans le temps.**

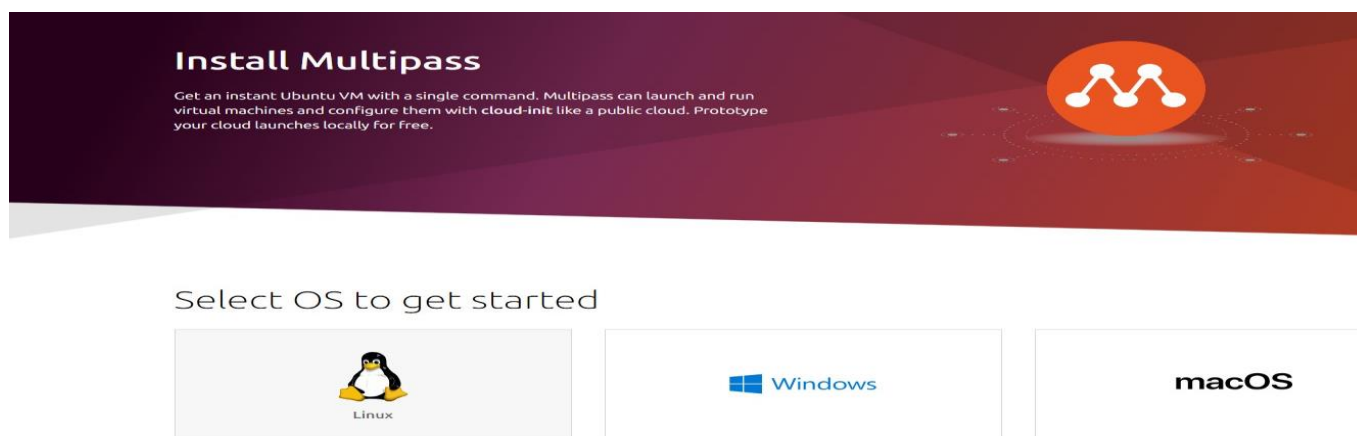
L'erreur à ne pas commettre serait de n'utiliser que la virtualisation ou la conteneurisation car on perdrait les avantages que proposent chacune des deux technologies. Vous pouvez donc sans souci profiter des avantages de la conteneurisation et de la virtualisation en les utilisant ensemble au sein d'un même projet.

Elle présente aussi plusieurs inconvénients. Il peut être difficile de gérer de façon efficiente un grand nombre de containers simultanément.

Déploiement et configuration

Environnement de travail

Nous allons faire une description de l'ensemble de la structure de travail dans laquelle nous retrouverons une hiérarchisation des étapes pour la réalisation du projet. Mais avant tout d'abord nous avons **installé** une infrastructure basée sur du multipass que vous pouvez trouver sur le lien <https://multipass.run> ou on a choisit l'image de **ubuntu 23.10** pour l'installer, puis on a installé **Docker** là-dessus à travers le lien [Installing Docker on Ubuntu 23.10: Step-by-Step Guide - Shapehost](#)



La creation de la machine du nom ubuntu-02 avec un système ubuntu 23.10

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\DELL> multipass launch 23.10 --name ubuntu-02
```

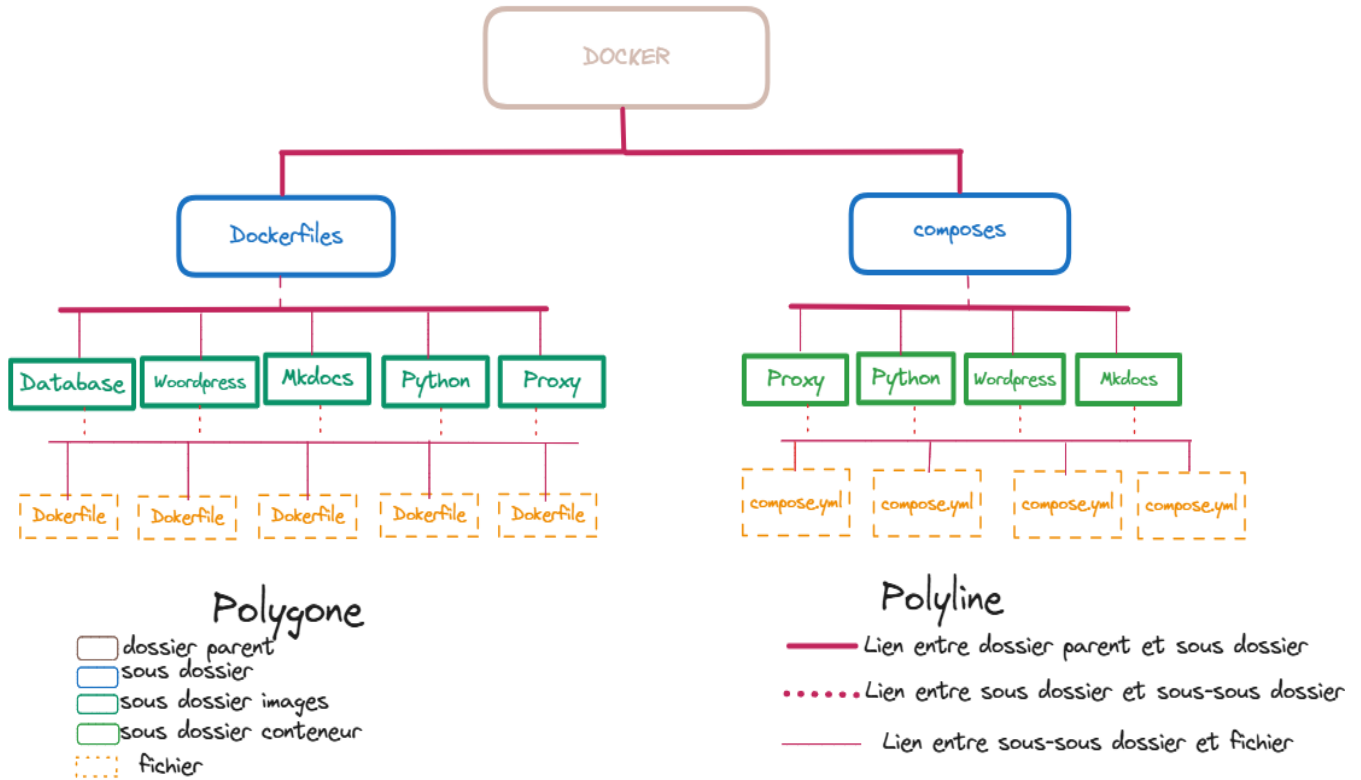
Le lancement de la machine qu'on vient de créer

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

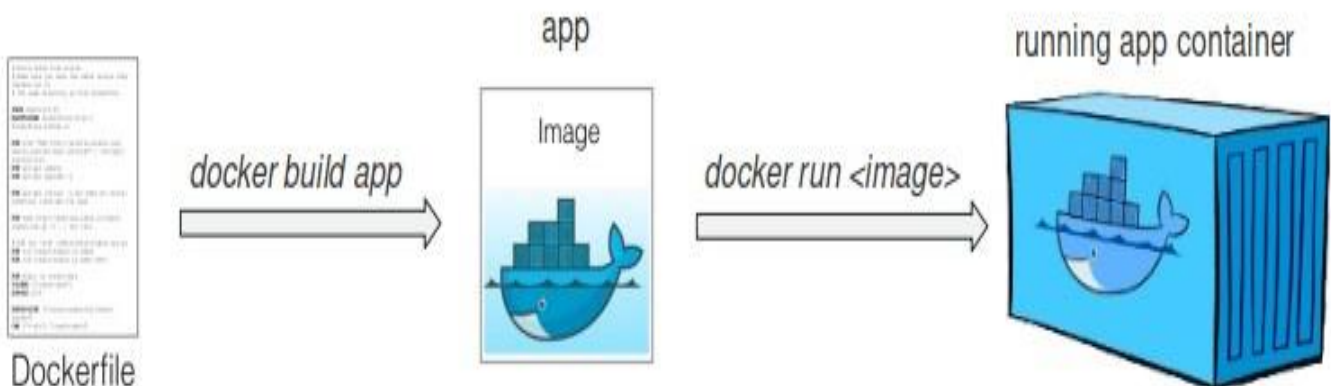
Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\DELL> multipass shell ubuntu-02
```

Hierarchisation des répertoires et fichiers.

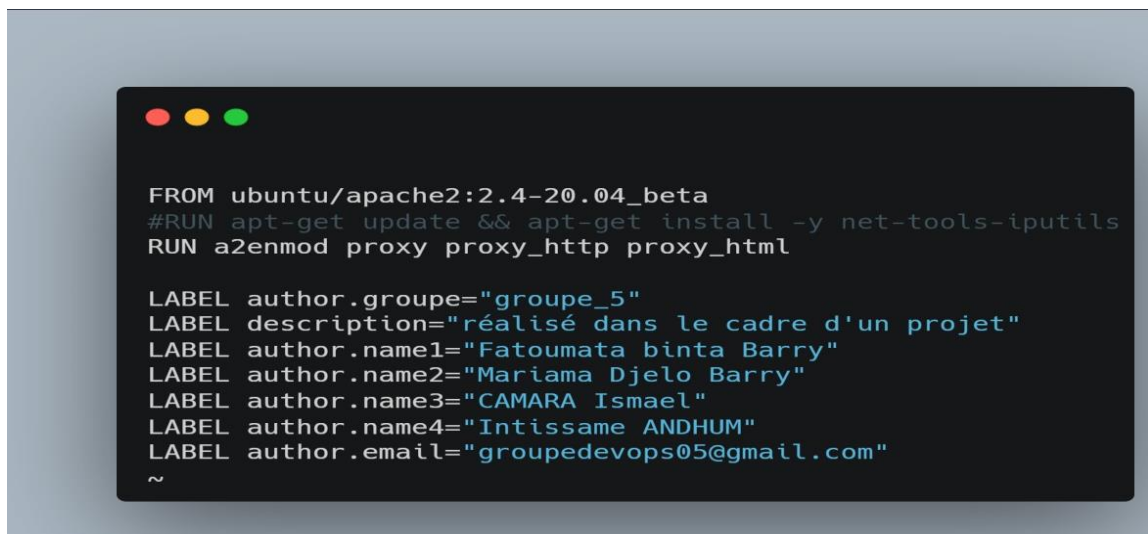


Il faut retenir que nous partons toujours d'un fichier pour créer une image et de l'image pour faire tourner le conteneur.



Le proxy apache2 :

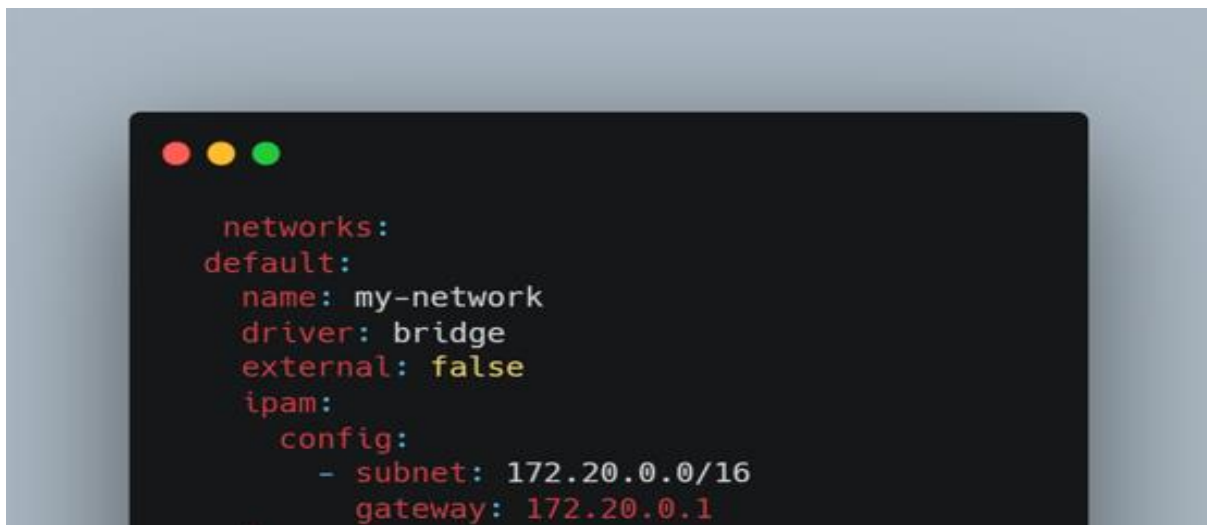
La première étape consiste la construction de l'image proxy, tout en procédant comme suit dans le terminal en tapant la commande docker build -t suivi de l'identifiant docker hub, puis nous l'avons attribué un nom plus la version souhaitée. Ensuite nous avons ajouté des différents Labels puis l'activation des modules pour le proxy.



```
FROM ubuntu/apache2:2.4-20.04_beta
#RUN apt-get update && apt-get install -y net-tools-iputils
RUN a2enmod proxy proxy_http proxy_html

LABEL author.groupe="groupe_5"
LABEL description="réalisé dans le cadre d'un projet"
LABEL author.name1="Fatoumata binta Barry"
LABEL author.name2="Mariama Djelo Barry"
LABEL author.name3="CAMARA Ismael"
LABEL author.name4="Intissame ANDHUM"
LABEL author.email="groupedevops05@gmail.com"
~
```

La deuxième étape constitue la création du fichier docker-compose.yml en commençant par la déclaration de la partie réseau en déclarant le nom, le type, le pont, ipam dans lequel nous retrouvons la déclaration de l'adresse réseau plus la passerelle.



```
networks:
default:
  name: my-network
  driver: bridge
  external: false
  ipam:
    config:
      - subnet: 172.20.0.0/16
        gateway: 172.20.0.1
```

La troisième étape concerne la déclaration du service dans lequel il y a le nom du service une déclaration de l'image construite dans la première étape, un port exposé qui est 80 plus un volume qui abrite une page web montée, une adresse ip fixe, pour terminer par une limitation des ressources utilisées.

```
services:
  proxy:
    image: ismaelttus1/proxy:0.0.1
    container_name: proxy
    volumes:
      - ./appli:/srv/www
      - ./apache2.conf:/etc/apache2/apache2.conf
    networks:
      default:
        ipv4_address: 172.20.0.2
    deploy:
      resources:
        limits:
          cpus: '0.1'
          memory: 100M
    ports:
      - 80:80
```

Ensuite nous avons créé un fichier apache2.conf qui abrite le chemin d'accès /srv/www vers le document root sur lequel nous avons monté un volume appli qui regorge la page web.

```
User ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}
ErrorLog ${APACHE_LOG_DIR}/error.log

HostnameLookups Off

# Include module configuration:
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf

# Include list of ports to listen on
Include ports.conf

<Directory />
    Options FollowSymLinks
    AllowOverride None
    Require all denied
</Directory>

<Directory /srv/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

<VirtualHost *:80>
    DocumentRoot /srv/www
</VirtualHost>
```


Puis nous allons exécuter la commande `docker-compose up -d` pour faire tourner le container.

Pour terminer nous exécuterons la commande `docker inspect proxy` qui est le nom du conteneur pour voir l'ensemble des configurations effectuées. Nous verrons l'identifiant

```
"Id": "38c3cee5ee3b0866302551e85f122e2427675694ecb62e9c450da9640e1baedb",
"Created": "2024-06-01T16:18:24.068298417Z",
"Path": "apache2-foreground",
"Args": [],
"State": {
  "Status": "running",
  "Running": true,
  "Paused": false,
  "Restarting": false,
  "OOMKilled": false,
  "Dead": false,
  "Pid": 2379,
  "ExitCode": 0,
  "Error": "",
  "StartedAt": "2024-06-01T16:18:41.906587485Z",
  "FinishedAt": "0001-01-01T00:00:00Z"
```

Dans notre processus d'inspection nous affichons l'ensemble des configurations effectuées en plus le port exposé qui est par défaut 80 sur le http.

```
"Config": {
  "Hostname": "38c3cee5ee3b",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": true,
  "AttachStderr": true,
  "ExposedPorts": {
    "80/tcp": {}
  }
},
```


Dans cette partie commande nous retrouverons l'image utilisée, les différents labels, le numero du container, le nom, le chemin d'accès au fichier de configuration, le chemin d'accès au dossier, le nom du service, sa version qui est le 2.27.0, la reference de l'image qui est ubuntu, et aussi sa version qui est

```

"Cmd": [
    "apache2-foreground"
],
"Image": "ismaelttus1/proxy:0.0.1",
"Volumes": null,
"WorkingDir": "",
"Entrypoint": null,
"OnBuild": null,
"Labels": {
    "author.email": "groupedevops05@gmail.com",
    "author.groupe": "groupe_5",
    "author.name1": "Fatoumata binta Barry",
    "author.name2": "Mariama Djelo Barry",
    "author.name3": "CAMARA Ismael",
    "author.name4": "Intissame ANDHUM",
    "com.docker.compose.config-hash":
"aa43b9dc6a04fab35f09ef385fa1baadf357b417082cb710f2d51005b84d207",
    "com.docker.compose.container-number": "1",
    "com.docker.compose.depends_on": "",
    "com.docker.compose.image":
"sha256:62daa0ee3721e698f4fff061990201aa7ea64f7939c4b971b6e773be18096cbc",
    "com.docker.compose.oneoff": "False",
    "com.docker.compose.project": "proxy",
    "com.docker.compose.project.config_files": "/root/docker/composes/proxy/docker-
compose.yml",
    "com.docker.compose.project.working_dir": "/root/docker/composes/proxy",
    "com.docker.compose.service": "proxy",
    "com.docker.compose.version": "2.27.0",
    "description": "réalisé dans le cadre d'un projet",
    "org.opencontainers.image.ref.name": "ubuntu",
    "org.opencontainers.image.version": "20.04"
},
"StopSignal": "SIGWINCH"
},

```

Dans cette autre partie nous verrons le nom du reseau créer, l'adresse ip, l'aliases, l' adresse mac, l'identifiant du container, la pacerelle par défaut, le préfixe et le nom du domaine.

```

"Networks": {
    "my-network": {
        "IPAMConfig": {
            "IPv4Address": "172.20.0.2"
        },
        "Links": null,
        "Aliases": [
            "proxy",
            "proxy"
        ],
        "MacAddress": "02:42:ac:14:00:02",
        "NetworkID": "3dbaaebadf1306d2e0aa2aad5109f0a518581e0bd0c6a08e47783a4e692785c",
        "EndpointID": "3577ff1351f749dec99403f7ad404dac3100bb9179617ae9b81c3fb05128f6e6",
        "Gateway": "172.20.0.1",
        "IPAddress": "172.20.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "DriverOpts": null,
        "DNSNames": [
            "proxy",
            "proxy"
        ]
    }
}

```

Pour terminer nous allons faire un docker stats pour afficher la limitation des ressources allouées c'est-à-dire la mémoire et le cpu

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
60b24b2b5cb6	proxy	0.07%	5.738MiB / 100MiB	5.74%	7.47kB / 0B	4.1kB / 8.19kB	56

Un aperçu du site proxy :

Celle-ci est notre page officielle

Pour plus de question consulter notre adresse gmail qui s'affiche juste en bas
 Nous serons à votre disposition pour des informations concernant le site
 l'ensemble des dispositifs prise pour la réalisation de ce projet
 Celle-ci est réalisée dans le cadre du projet DevOps
groupepdevops05@gmail.com

welcome to site proxy

Le site web construit avec l'image qui fonctionnera sur python :

La première étape consiste construction de l'image par la commande docker build -t suivi de l'identifiant docker hub, le nom de l'image, la version souhaitée plus des différent Labels avec un port exposé à l'intérieur du conteneur, une création du dossier qui sera le répertoire par défaut ensuite une copie du dossier appli_liste plus son contenu index.html et écrasement du cmd.

```
FROM python:3.12.3-alpine
RUN mkdir /site
COPY appli_liste /site
WORKDIR /site
EXPOSE 8080
CMD python -m http.server 8080

LABEL author.groupe="groupe_5"
LABEL description="réalisé dans le cadre d'un projet"
LABEL author.name1="Fatoumata binta Barry"
LABEL author.name2="Mariama Djelo Barry"
LABEL author.name3="CAMARA Ismael"
LABEL author.name4="Intissame ANDHUM"
LABEL author.email="groupedevops05@gmail.com"
~
```

La deuxième étape constitue la création du fichier docker-compose.yml en commençant par la déclaration de la partie réseau en débutant par le nom, le type, le pont, ipam dans lequel nous retrouvons la déclaration de l'adresse réseau plus la passerelle et le réseau du proxy qui lui est externe.

```
networks:
  default:
    name: net
    driver: bridge
    ipam:
      config:
        - subnet: 172.21.0.0/16
          gateway: 172.21.0.1
  my-network:
    external: true
```

La troisième étape concerne la déclaration du service dans lequel se trouve l'image construite dans la première étape, le nom du conteneur, cette fois-ci pas de port mappé puisque nous mettrons en place un proxypass pour atteindre le service et pas de volume aussi car le dossier abrite déjà la page web par défaut dans la construction de l'image, une adresse ip fixe pour le conteneur pour terminer une limitation des ressources utilisées.

```
services:
  python-1:
    image: ismaelttus1/python:0.0.1
    container_name: python
    #ports:
    #  #- 81:8080
    deploy:
      resources:
        limits:
          cpus: '0.01'
          memory: 80M
    networks:
      default:
        ipv4_address: 172.21.0.2
      my-network:
```

Pour la mise en place du proxypass dans le fichier apache2.conf nous allons ajouter le nom du container plus le port exposé à l'intérieur.

```
<VirtualHost *:80>
  ServerName python.local
  ProxyPass / http://python:8080/
  ProxyPassReverse / http://python:8080/
</VirtualHost>
```

Ensuite nous utiliserons la commande `docker-compose up -d` pour faire tourner le conteneur.

Pour terminer nous exécuterons la commande `docker inspect python` pour voir l'ensemble des configurations effectuées.

```
[
  {
    "Id": "e143ade63b98a939c625c9db3e6e4df169904a4e484be0af2ef82ffdc56a884c",
    "Created": "2024-05-30T13:25:48.58159549Z",
    "Path": "/bin/sh",
    "Args": [
      "-c",
      "python -m http.server 8080"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 22335,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2024-05-30T13:26:04.135666965Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    }
  },
]
```

Dans cette étape nous ressortons la commande qui à été changé, l'image utilisée, les différents labels respectifs, le numero, le nom, du container, le fichier de configuration, le chemin d'accès du repertoire, le nom et la version du service

```
"Cmd": [
  "/bin/sh",
  "-c",
  "python -m http.server 8080"
],
"Image": "ismaelttus1/python:0.0.1",
"Volumes": null,
"WorkingDir": "/site",
"Entrypoint": null,
"OnBuild": null,
"Labels": {
  "author.email": "groupepdevops05@gmail.com",
  "author.groupe": "groupe_5",
  "author.name1": "Fatoumata binta Barry",
  "author.name2": "Mariama Djelo Barry",
  "author.name3": "CAMARA Ismael",
  "author.name4": "Intissame ANDHUM",
  "com.docker.compose.config-hash":
f8619606c0363fd1ef3c756fcc71a366bc85923490cc03aef738f3c0164f0cc",
  "com.docker.compose.container-number": "1",
  "com.docker.compose.oneoff": "False",
  "com.docker.compose.project": "python",
  "com.docker.compose.project.config_files": "docker-compose.yml",
  "com.docker.compose.project.working_dir": "/root/docker/composes/python",
  "com.docker.compose.service": "python-1",
  "com.docker.compose.version": "1.29.2",
  "description": "réalisé dans le cadre d'un projet"
}
```


L'inspection de la partie réseau dans laquelle nous retrouverons l'alias, l'identifiant réseau, la pacerelle par défaut, le préfixe et l'adresse mac des deux réseaux créer à savoir my-network et net.

```
"my-network": {
  "IPAMConfig": null,
  "Links": null,
  "Aliases": [
    "e143ade63b98",
    "python-1"
  ],
  "NetworkID": "7235131f6bb150cd198ae621adc4e6b644492abebd10838f577cc88fcc4ab90a",
  "EndpointID": "043bcc14271e26710c9b4fd2e63bde19ec242e888ed95f52e3c72eca254f5a86",
  "Gateway": "172.20.0.1",
  "IPAddress": "172.20.0.3",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "MacAddress": "02:42:ac:14:00:03",
  "DriverOpts": null
},
"net": {
  "IPAMConfig": {
    "IPv4Address": "172.21.0.2"
  },
  "Links": null,
  "Aliases": [
    "e143ade63b98",
    "python-1"
  ],
  "NetworkID": "c75ca38deacf4d3df1b47a361ae3a6362170314b3109661b977f5a11d94ac848",
  "EndpointID": "572bddd01befd3d9f273c3e7281494555e6edb58bc93509262ebd0596f777090",
  "Gateway": "172.21.0.1",
  "IPAddress": "172.21.0.2",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "MacAddress": "02:42:ac:15:00:02",
  "DriverOpts": null
}
```

Pour terminer nous réaliserons un docker stats pour afficher les limitations effectuées à savoir le cpu et la mémoire

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
fb184a493ef4	python	0.02%	15.52MiB / 80MiB	19.39%	16.8kB / 10.4kB	5.17MB / 1.91MB	1

Un aperçu de la liste des membres



Un déploiement de Mkdocs avec l'ensemble des éléments construits dans le dockerfile.

Dans cette première phase nous effectuerons la construction de l'image par commande docker build -t suivi de l'identifiant docker hub, du nom de l'image plus la version voulue. Dans notre dockerfile nous retrouverons la provenance de l'image qui est le python, l'installation de Mkdocs, le changement du répertoire courant de travail, une copie du répertoire personnel, la construction de Mkdocs, le rechargement du répertoire de travail par /app/site qui va automatiquement générer le sous répertoire site, ensuite exposé le port à l'intérieur du container, en plus un écrasement de la commande, pour terminer les différents labels habituels.

```
FROM python:3.12.3-alpine
RUN pip install mkdocs
WORKDIR /app
COPY . .
RUN mkdocs build
WORKDIR /app/site
EXPOSE 8000
CMD python -m http.server 8000

LABEL author.groupe="groupe_5"
LABEL description="réalisé dans le cadre d'un projet"
LABEL author.name1="Fatoumata binta Barry"
LABEL author.name2="Mariama Djelo Barry"
LABEL author.name3="CAMARA Ismael"
LABEL author.name4="Intissame ANDHUM"
LABEL author.email="groupedevops05@gmail.com"
```

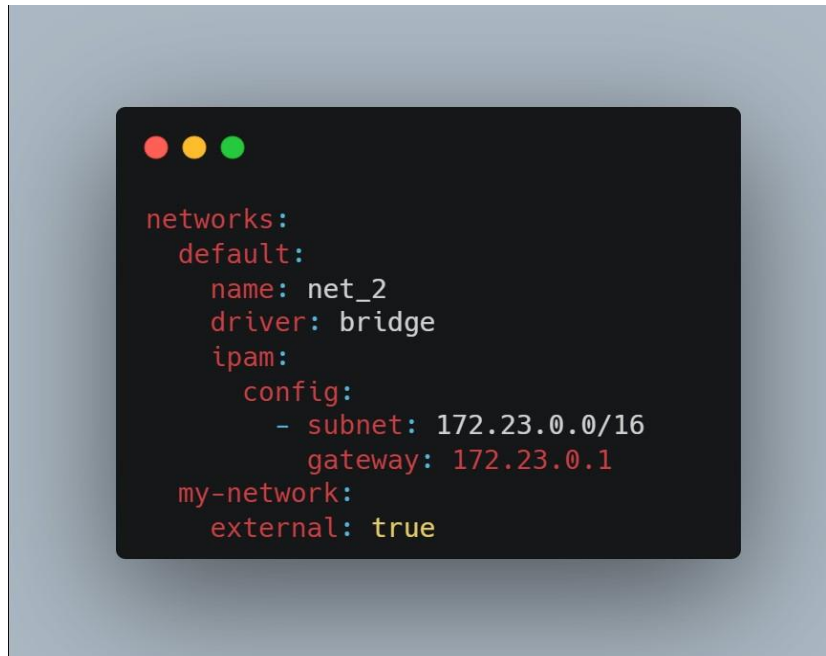
la création du fichier mkdocs.yml qui abritera le nom du site, ensuite un dossier docs qui a son tour regorgera un fichier aussi index.md dans lequel nous retrouverons le contenu du site.

```
site_name: My Docs
```

```
# Le groupe de projet DevOps numero
05

## La liste des membres
Barry Fatoumata Binta
Barry Djelo Mariama
CAMARA Ismael
ANDHUM Intissame
#Sous l'encadrement
DIALLO Aboubacar
```

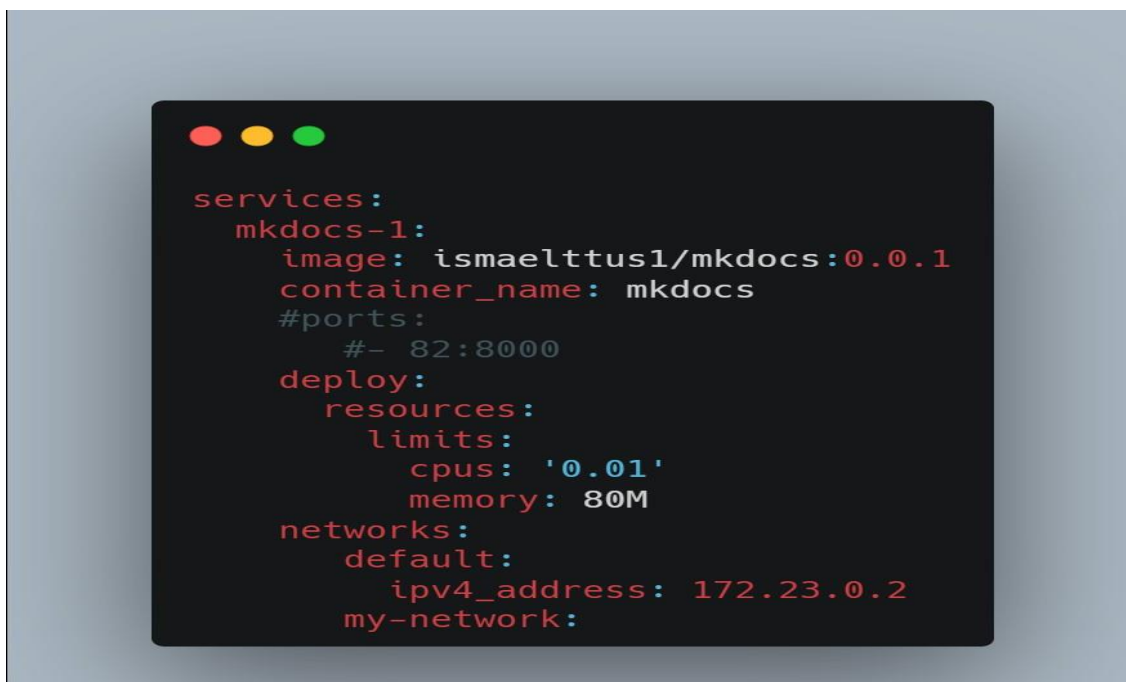

La deuxième phase constitue la création du fichier docker-compose.yml en commençant par la déclaration de la partie réseau en débutant par le nom, le type, le pont, ipam dans lequel nous retrouvons la déclaration de l'adresse réseau plus la passerelle et le réseau du proxy qui lui est externe.



```

networks:
  default:
    name: net_2
    driver: bridge
    ipam:
      config:
        - subnet: 172.23.0.0/16
          gateway: 172.23.0.1
  my-network:
    external: true
  
```

La troisième phase concerne la déclaration du service dans lequel se trouve l'image construite dans la première phase, le nom du conteneur, cette fois-ci pas de port mappé puisque nous mettrons en place un proxypass pour atteindre le conteneur, une adresse ip fixe pour le conteneur pour terminer une limitation des ressources utilisées.



```

services:
  mkdocs-1:
    image: ismaelttus1/mkdocs:0.0.1
    container_name: mkdocs
    #ports:
    #  - 82:8000
    deploy:
      resources:
        limits:
          cpus: '0.01'
          memory: 80M
    networks:
      default:
        ipv4_address: 172.23.0.2
      my-network:
  
```

Pour la mise en place du proxypass dans le fichier apache2.conf nous allons ajouter le nom du container plus le port exposé à l'intérieur.

```
<VirtualHost *:80>
  ServerName mkdocs.local
  ProxyPass / http://mkdocs:8000/
  ProxyPassReverse / http://mkdocs:8000/
</VirtualHost>
```

Ensuite nous utiliserons la commande docker-compose up -d pour faire tourner le conteneur. Pour terminer nous exécuterons la commande docker inspect Mkdoks pour voir l'ensemble des configurations effectuées.

```
[
  {
    "Id":
    "ed73064e59b10f4ba7bc66b9bacff158984a82b724007
    2e6eed7d8b68a0c999b",
    "Created": "2024-06-
    05T15:46:38.283931292Z",
    "Path": "/bin/sh",
    "Args": [
      "-c",
      "python -m http.server 8000"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 1814,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2024-06-
      05T15:46:46.561643594Z",
      "FinishedAt": "0001-01-
      01T00:00:00Z"
    },
  },
]
```

Dans cette partie commande nous retrouverons la nouvelle commande, l'image qui a servi à faire tourner le container, le repertoire courant, le numero du container, le nom du dossier, le nom du fichier de configuration, le chemin d'accès du dossier, le nom du service et sa version.

```

"Cmd": [
    "/bin/sh",
    "-c",
    "python -m http.server 8000"
],
"Image": "ismaelttus1/mkdocs:0.0.1",
"Volumes": null,
"WorkingDir": "/app/site",
"Entrypoint": null,
"OnBuild": null,
"Labels": {
    "com.docker.compose.config-hash":
"eadeb6b7e27eae4997e4a71e86c64f07013e123701f82ac6ef4768277671b3f",
    "com.docker.compose.container-number": "1",
    "com.docker.compose.oneoff": "False",
    "com.docker.compose.project": "mkdocs",
    "com.docker.compose.project.config_files": "docker-compose.yml",
    "com.docker.compose.project.working_dir": "/root/docker/composes/mkdocs",
    "com.docker.compose.service": "mkdocs-1",
    "com.docker.compose.version": "1.29.2"
}

```

Dans cette partie nous rencontrons aliases, l'identifiant du réseau, l'adresse ip et la passerelle du réseau afférant plus le préfixe. L'adresse mac et le nom.

```
"my-network": {
  "IPAMConfig": null,
  "Links": null,
  "Aliases": [
    "ed73064e59b1",
    "mkdocs-1"
  ],
  "NetworkID": "c6a5853862a6b42190eafa1d6803b1014163ddef8f49ca08cb9d0db09e7b253b",
  "EndpointID": "051ce1b86bd201d89fd251f4f4d6f1a39bcbe68716c235e7665320cc848b8ad9",
  "Gateway": "172.20.0.1",
  "IPAddress": "172.20.0.2",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "MacAddress": "02:42:ac:14:00:02",
  "DriverOpts": null
}
"my-network": {
  "IPAMConfig": null,
  "Links": null,
  "Aliases": [
    "ed73064e59b1",
    "mkdocs-1"
  ]
}
```

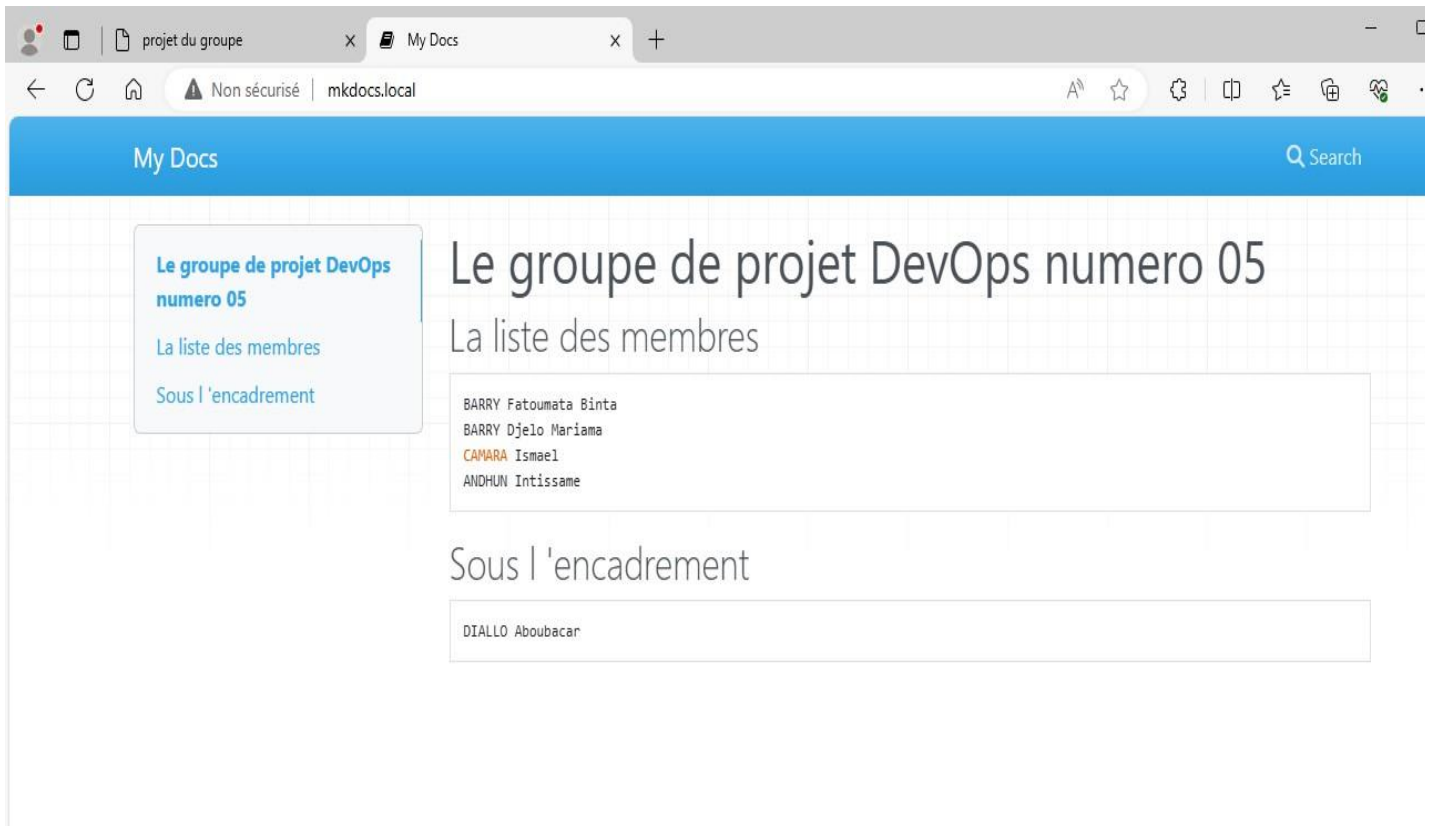
Dans cette partie nous rencontrons aliases, l'identifiant du réseau, l'adresse ip et la passerelle du réseau afférant plus le préfixe. L'adresse mac et le nom.

```
"net_2": {
  "IPAMConfig": {
    "IPv4Address": "172.23.0.2"
  },
  "Links": null,
  "Aliases": [
    "ed73064e59b1",
    "mkdocs-1"
  ],
  "NetworkID": "03f5377dc0b5c8c5b5dc3a4a66443d2eb2002b1ae839c08d5b81d604c5f0fb",
  "EndpointID": "895e872f618342dabee9090c0578cb34a9cdb3f8e8c63ecdcf6eaa8f39d2398b",
  "Gateway": "172.23.0.1",
  "IPAddress": "172.23.0.2",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "MacAddress": "02:42:ac:17:00:02",
  "DriverOpts": null
}
```

Pour terminer nous réaliserons un docker stats pour afficher les limitations effectuées à savoir le cpu et la mémoire

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
ed73064e59b1	mkdocs	0.02%	24.51MiB / 80MiB	30.64%	3.34kB / 0B	14.6MB / 57.3kB	1

Un aperçu de notre Mkdocs



Le site web WordPress s'appuyant sur une image construite mais référent sur l'image de wordpress et de la base de données avec une connexion à mysql qui sera préconfigurée.

Dans cette séance nous effectuerons la construction de l'image par commande docker build -t suivi de l'identifiant docker hub, du nom de l'image plus la version voulue. Pour l'ensemble des deux images wordpress et mysql. Dans notre dockerfile nous retrouverons les provenances des l'images qui sont wordpress et mysql, pour finir les différents labels habituels.

La construction de l'image wordpress

```
FROM wordpress:php8.3-apache

LABEL author.groupe="groupe_5"
LABEL description="réalisé dans le cadre d'un projet"
LABEL author.name1="Fatoumata binta Barry"
LABEL author.name2="Mariama Djelo Barry"
LABEL author.name3="CAMARA Ismael"
LABEL author.name4="Intissame ANDHUM"
LABEL author.email="groupedevops05@gmail.com"
```

La construction de l'image mysql

```
FROM mysql:8.3.0

LABEL author.groupe="groupe_5"
LABEL description="réalisé dans le cadre d'un projet"
LABEL author.name1="Fatoumata binta Barry"
LABEL author.name2="Mariama Djelo Barry"
LABEL author.name3="CAMARA Ismael"
LABEL author.name4="Intissame ANDHUM"
LABEL author.email="groupedevops05@gmail.com"
```


Cette deuxième séance correspond la création du fichier docker-compose.yml en commençant par la déclaration de la partie réseau en débutant par le nom, le type, le pont, ipam dans lequel nous retrouvons la déclaration de l'adresse réseau plus la passerelle et le réseau du proxy qui lui est externe.

```
networks:
  default:
    name: net_1
    driver: bridge
    ipam:
      config:
        - subnet: 172.22.0.0/16
          gateway: 172.22.0.1
  my-network:
    external: true
services:
```

La troisième séance concerne la déclaration du service dans lequel se trouve l'image construite dans la première séance, le nom du conteneur, cette fois-ci nous exposons le port puisque nous n'avons pas pu mettre en place un proxypass pour atteindre le service et un volume qui sera affecté au chemin d'accès par défaut, une adresse ip fixe pour le conteneur pour terminer une limitation des ressources utilisées.

```
wordpress:
  image: ismaelttus1/wordpress:0.0.1
  container_name: wordpress
  restart: always
  ports:
    - 8080:80
  environment:
    WORDPRESS_DB_HOST: db
    WORDPRESS_DB_USER: root
    WORDPRESS_DB_PASSWORD: password
    WORDPRESS_DB_NAME: wp_wordpress
  volumes:
    - wordpress:/var/www/html
  networks:
    default:
      ipv4_address: 172.22.0.2
    my-network:
  deploy:
    resources:
      limits:
        cpus: '0.1'
        memory: 512M
```


Dans cette quatrième séance nous allons déclarer un autre service qui à son tour va servir d' une banque de données, en commençant par l' image utilisée, le nom du container, les environnements à savoir le nom, le mot de passe administrateur. Ensuite un volume monté vers le chemin d'accès par défaut, un réseau qui regorge une adresse ip fixe plus le nom du réseau qui lui est externe. Une limitation des ressources à savoir le cpus et la memoire pour terminer la declaration des deux volumes de la troisième et quatrième séances

```

db:
  image: ismaelttus1/mysql:0.0.1
  container_name: db
  restart: always
  environment:
    MYSQL_DATABASE: wp_wordpress
    #MYSQL_USER: exampleuser
    MYSQL_ROOT_PASSWORD: password
    #MYSQL_RANDOM_ROOT_PASSWORD: '1'
  volumes:
    - db:/var/lib/mysql
  networks:
    default:
      ipv4_address: 172.22.0.3
    my-network:
  deploy:
    resources:
      limits:
        cpus: '0.1'
        memory: 1024M

volumes:
  wordpress:
  db:

```

Ensuite nous utiliserons la commande `docker-compose up -d` pour faire tourner le conteneur. Pour terminer nous exécuterons la commande `docker inspect wordpress` pour voir l'ensemble des configurations effectuées une première fois.

```

[
  {
    "Id": "c81144cc7bfa724de9b6d14bd09409456be4271a106d6dd9f5c4561c17247c66",
    "Created": "2024-06-07T04:50:34.397613976Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "apache2-foreground"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 34762,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2024-06-07T04:51:09.082335447Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    }
  },

```

Dans cette partie Mounts nous retrouverons le type monté qui est un volume, le nom, la source du volume, le chemin d'accès plus les droits qui lui est assigné

```

    },
    "Name": "overlay2"
  },
  "Mounts": [
    {
      "Type": "volume",
      "Name": "wordpress_wordpress",
      "Source": "/var/lib/docker/volumes/wordpress_wordpress/_data",
      "Destination": "/var/www/html",
      "Driver": "local",
      "Mode": "z",
      "RW": true,
      "Propagation": ""
    }
  ],

```

La partie configuration qui porte l'identifiant de la machine plus le port exposé

```

    "Config": {
      "Hostname": "c81144cc7bfa",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": true,
      "AttachStderr": true,
      "ExposedPorts": {
        "80/tcp": {}
      }
    }

```

Le paramétrage de wordpress et de la base de données à savoir le mot passe et le nom de la base de données. Le nom d'utilisateur.

```
"Env": [
    "WORDPRESS_DB_HOST=db",
    "WORDPRESS_DB_USER=root",
    "WORDPRESS_DB_PASSWORD=password",
    "WORDPRESS_DB_NAME=wp_wordpress",
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
```

La partie commande ou nous trouverons le nom de l'image utilisée, le chemin d'accès au fichier, le repertoire courant.

```
],
    "Cmd": [
        "apache2-foreground"
    ],
    "Image": "ismaelttus1/wordpress:0.0.1",
    "Volumes": {
        "/var/www/html": {}
    },
    "WorkingDir": "/var/www/html",
    "Entrypoint": [
        "docker-entrypoint.sh"
```

Le Labels respectifs, le numero du conteneur, le nom du repertoire, le chemin d'accès au fichier de configuration, le chemin d'accès au repertoire courant, le nom du service et la version.

```
"Labels": {
  "author.email": "groupedevops05@gmail.com",
  "author.groupe": "groupe_5",
  "author.name1": "Fatoumata binta Barry",
  "author.name2": "Mariama Djelo Barry",
  "author.name3": "CAMARA Ismael",
  "author.name4": "Intissame ANDHUM",
  "com.docker.compose.config-hash":
"313b6512f590e5e5d5bddb30d93398f3fd2525cbde0e169366b430ef6be11c36",
  "com.docker.compose.container-number": "1",
  "com.docker.compose.depends_on": "",
  "com.docker.compose.image":
"sha256:da49c028070dd2aa05bc6d532621661d6f0fc5c17a883d6d3cd50a7540b7c1cc",
  "com.docker.compose.oneoff": "False",
  "com.docker.compose.project": "wordpress",
  "com.docker.compose.project.config_files": "/root/docker/composes/wordpress/docker-
compose.yml",
  "com.docker.compose.project.working_dir": "/root/docker/composes/wordpress",
  "com.docker.compose.replace":
"d9107ba8522c4b59c84ealc1b4d2c6620291f8e424d9721f0174b8cbb10112ff",
  "com.docker.compose.service": "wordpress",
  "com.docker.compose.version": "2.27.0",
  "description": "réalisé dans le cadre d'un projet"
},
```

La partie réseau ou nous retrouverons aliases, l'adresse mac, l'identifiant réseau, l'adresse de la pacerelle et celle du conteneur plus le préfixe pour terminer le nom dns

```
"Networks": {
  "my-network": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
      "wordpress",
      "wordpress"
    ],
    "MacAddress": "02:42:ac:14:00:03",
    "NetworkID": "bc3ca9a52b63808d91b4137189a80606c090819cd084e405f9d4f1cec77138ce",
    "EndpointID": "e1a9b381f9efb4efa22512afe643779624483b57c3716fbe8706180d7e57c13d",
    "Gateway": "172.20.0.1",
    "IPAddress": "172.20.0.3",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "DriverOpts": null,
    "DNSNames": [
      "wordpress",
      "c81144cc7bfa"
    ]
  },
}
```

La deuxième partie ressort l'adresse ip, l'aliases, l'adresse mac, l'identifiant, la pacerelle, le préfixe et le dns

```

"net_1": {
  "IPAMConfig": {
    "IPv4Address": "172.22.0.2"
  },
  "Links": null,
  "Aliases": [
    "wordpress",
    "wordpress"
  ],
  "MacAddress": "02:42:ac:16:00:02",
  "NetworkID": "449441e92339b613d550f9893b99dec695de4c97fb84eaa21261fd03ae059954",
  "EndpointID": "3ec00cf76071830a8c189c3807db35b930e08cfa11123936203d80e838bcb808",
  "Gateway": "172.22.0.1",
  "IPAddress": "172.22.0.2",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "DriverOpts": null,
  "DNSNames": [
    "wordpress",
    "c81144cc7bfa"
  ]
}

```

Nous exécuterons la commande `docker inspect mysql` pour voir l'ensemble des configurations effectuées une seconde fois.

```

"Id": "43913be16785705ae184e5ed9404f1bec9e3aca57ecc3108949fb8fbf4775535",
"Created": "2024-06-07T04:50:34.359849161Z",
"Path": "docker-entrypoint.sh",
"Args": [
  "mysqld"
],
"State": {
  "Status": "running",
  "Running": true,
  "Paused": false,
  "Restarting": false,
  "OOMKilled": false,
  "Dead": false,
  "Pid": 46969,
  "ExitCode": 0,
  "Error": "",
  "StartedAt": "2024-06-07T05:12:04.4064349Z",
  "FinishedAt": "2024-06-07T05:11:55.219375016Z"
}

```

Dans cette partie Mounts nous retrouverons le type monté qui est un volume, le nom, la source du volume, le chemin d'accès plus les droits qui lui est assigné.

```
"Mounts": [
  {
    "Type": "volume",
    "Name": "wordpress_db",
    "Source": "/var/lib/docker/volumes/wordpress_db/_data",
    "Destination": "/var/lib/mysql",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  }
]
```

La partie configuration qui regorge le nom de la machine, le port exposé plus le paramétrage à savoir le mot de passe et le nom de la base de données

```
"Config": {
  "Hostname": "43913be16785",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": true,
  "AttachStderr": true,
  "ExposedPorts": {
    "3306/tcp": {},
    "33060/tcp": {}
  },
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
    "MYSQL_ROOT_PASSWORD=password",
    "MYSQL_DATABASE=wp_wordpress",
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "GOSU_VERSION=1.17",
    "MYSQL_MAJOR=innovation",
    "MYSQL_VERSION=8.3.0-1.el8",
    "MYSQL_SHELL_VERSION=8.3.0-1.el8"
  ]
}
```


Dans cette partie commande nous ressortons l'image utilisée, le chemin d'accès, les labels respectifs, le numero du conteneur, le nom du dossier de travail, le chemin d'accès du fichier de configuration, le chemin d'accès au repertoire courant, le nom et la version du service.

```

"Cmd": [
    "mysqld"
],
"Image": "ismaelttus1/mysql:0.0.1",
"Volumes": {
    "/var/lib/mysql": {}
},
"WorkingDir": "",
"Entrypoint": [
    "docker-entrypoint.sh"
],
"OnBuild": null,
"Labels": {
    "author.email": "groupeDEVops05@gmail.com",
    "author.groupe": "groupe_5",
    "author.name1": "Fatoumata binta Barry",
    "author.name2": "Mariama Djelo Barry",
    "author.name3": "CAMARA Ismael",
    "author.name4": "Intissime ANDHUM",
    "com.docker.compose.config-hash":
"2d6c4e5a55f80a03d90e72d3d2207d6612cdb2b3affef76ef76627922ae9b13a",
    "com.docker.compose.container-number": "1",
    "com.docker.compose.depends_on": "",
    "com.docker.compose.image":
"sha256:c2ff98026338a2d29227dc16453f1ce2f3b56a60fe85c802a147b6009225fb9b",
    "com.docker.compose.oneoff": "False",
    "com.docker.compose.project": "wordpress",
    "com.docker.compose.project.config_files": "/root/docker/composes/wordpress/docker-
compose.yml",
    "com.docker.compose.project.working_dir": "/root/docker/composes/wordpress",
    "com.docker.compose.replace":
"19c99124ce0e293c9f0631b5a8cc35ec9bdd8cf38c6a05165b0ff033fe464b",
    "com.docker.compose.service": "db",
    "com.docker.compose.version": "2.27.0",
    "description": "réalisé dans le cadre d'un projet"
}

```

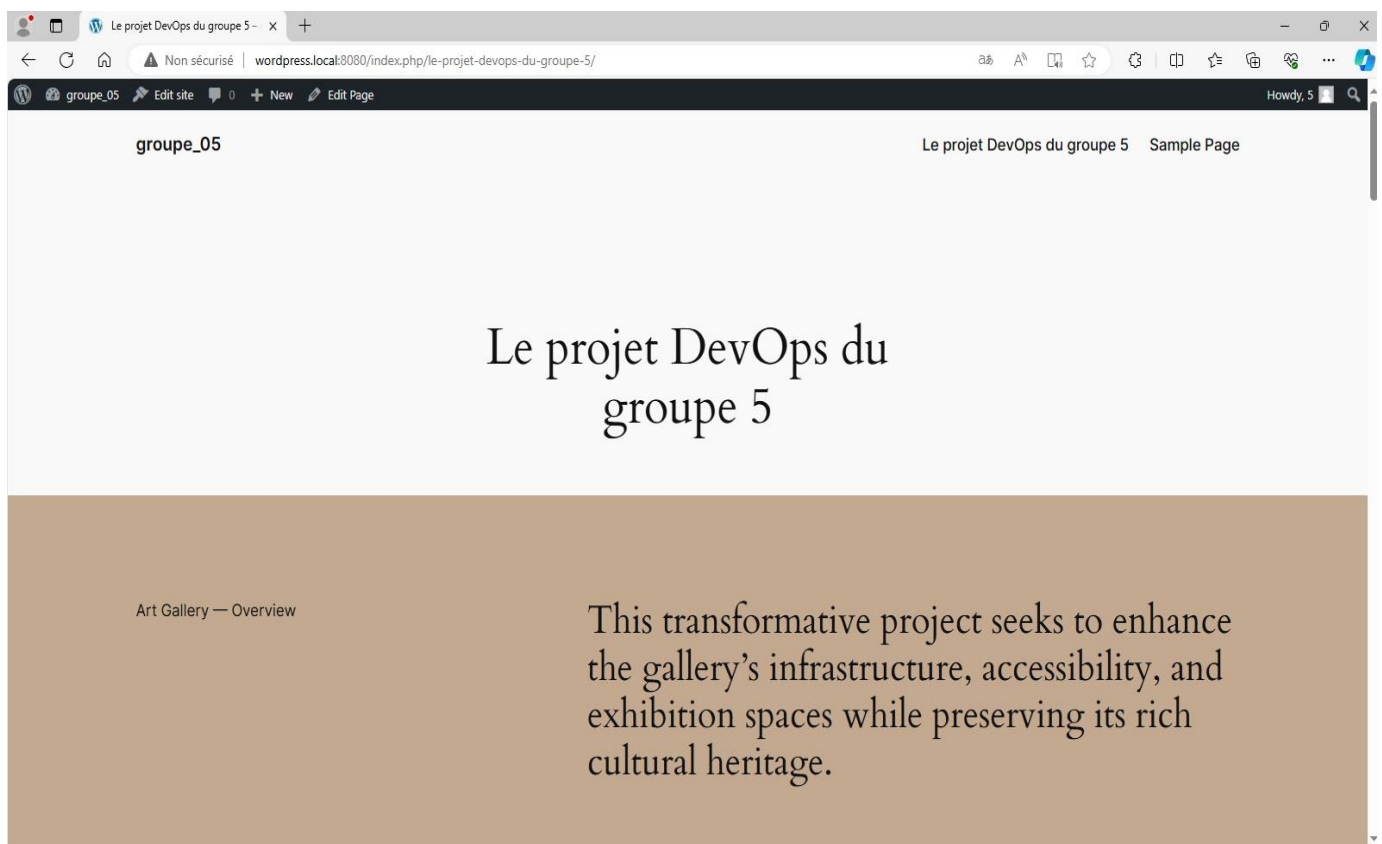
La partie réseau ou nous pouvons voir l'aliases, l'adresse mac, l'identifiant réseau, l'adresse ip de la pacerelle et du conteneur suivi du préfixe plus le nom dns

```
"Networks": {
  "my-network": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
      "db",
      "db"
    ],
    "MacAddress": "02:42:ac:14:00:04",
    "NetworkID": "bc3ca9a52b63808d91b4137189a80606c090819cd084e405f9d4f1cec77138ce",
    "EndpointID": "2654f3a6a2b84324656f27a6b5341d9bb86e0e9b351df4c175d929b24aef188c",
    "Gateway": "172.20.0.1",
    "IPAddress": "172.20.0.4",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "DriverOpts": null,
    "DNSNames": [
      "db",
      "43913be16785"
    ]
  }
}
```


La partie réseau ou nous pouvons voir l'aliases, l'adresse mac, l'identifiant réseau, l'adresse ip de la pacerelle et du conteneur suivi du préfixe plus le nom dns

```
"net_1": {
  "IPAMConfig": {
    "IPv4Address": "172.22.0.3"
  },
  "Links": null,
  "Aliases": [
    "db",
    "db"
  ],
  "MacAddress": "02:42:ac:16:00:03",
  "NetworkID": "449441e92339b613d550f9893b99dec695de4c97fb84eaa21261fd03ae059954",
  "EndpointID": "3e88a3da778e742cbbf81ab5aa75f75d20513ce5914c1ee686a72eb18046858e",
  "Gateway": "172.22.0.1",
  "IPAddress": "172.22.0.3",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "DriverOpts": null,
  "DNSNames": [
    "db",
    "43913be16785"
```

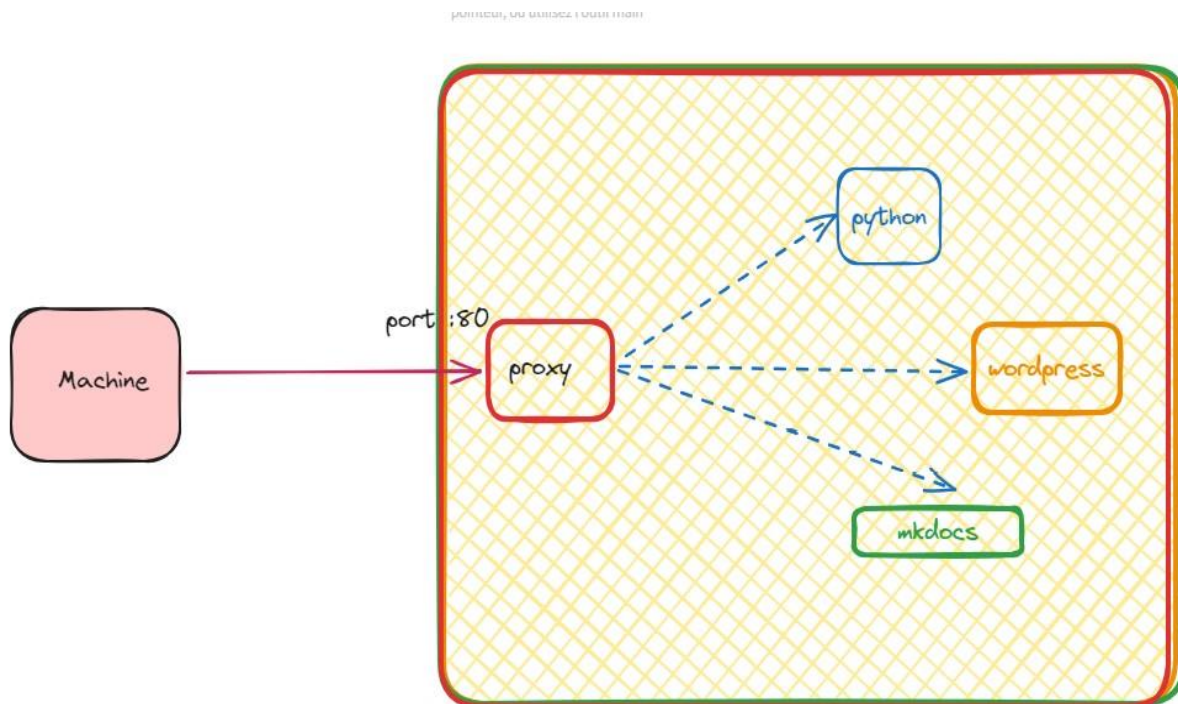
Un aperçu de notre page wordpress en passant par le port mappé puisque nous avons pas réaliser un proxypass au niveau du proxy



Schema illustratif

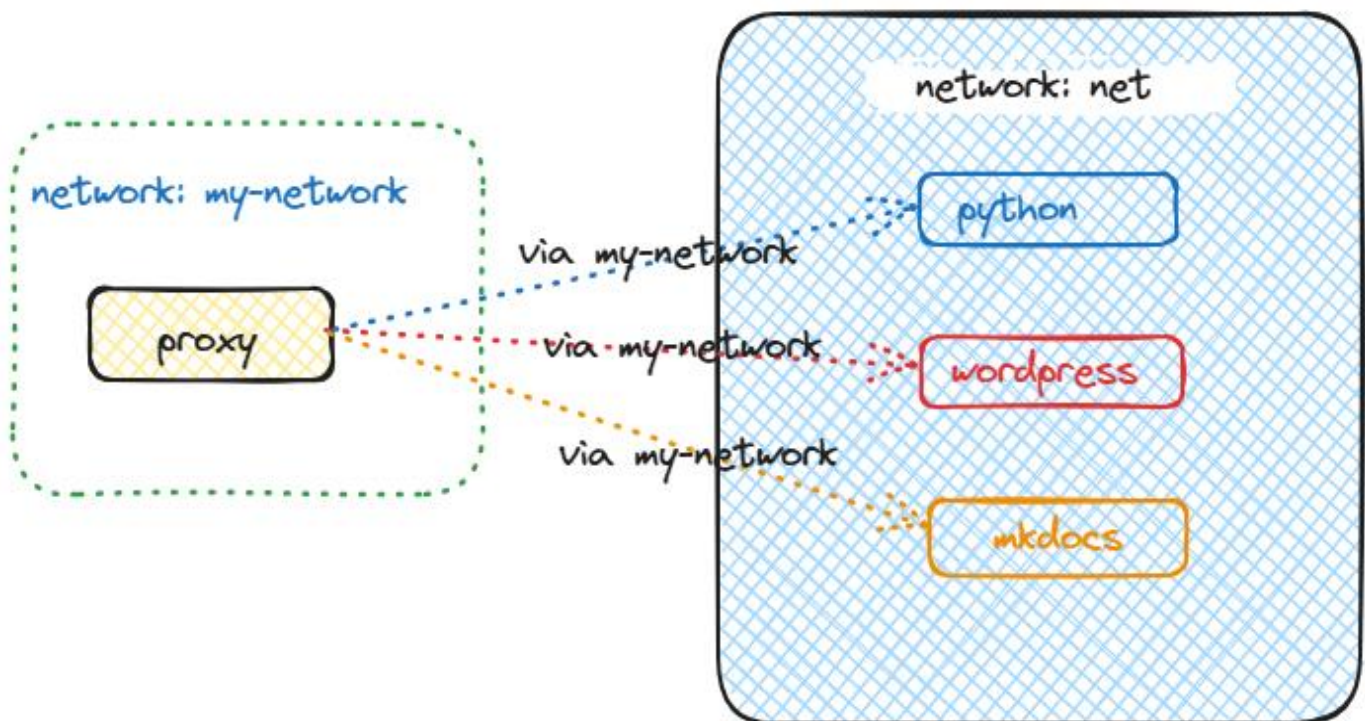
Nous venons de réaliser un proxy entrant dont l'utilisateur sollicitera des services telque decrit précédament dans la partie deploiement, le proxy qui sert de serveur mendataire c'est à dire toutes les requêtes passent par lui ayant un port exposé qui est le 80 par défaut, qui à son tour ira vers les différents conteneurs selon la spécification de la demande, en effectuant un proxypass pour la dissimulation des ports des conteneurs tout en faisant une mise en cache (proxy clair).

NB: Il faut signaler que nous avons pas pu réaliser le proxypass pour le site wordpress.



Schema illustratif

Un réseau my-network issue du conteneur proxy ou qui est le réseau du proxy dont les autres font partir accessible comme indiqué sur le schema. Un autre réseau net qui englobe les trois autres conteneurs c'est à dire le réseau des trois conteneurs.



Conclusion

Les solutions de Docker sont diverses et variées, couvrant un large éventail de besoins dans le domaine du développement, du déploiement et de la gestion des applications. Voici quelques-unes des solutions les plus courantes :

- Docker Engine: Le cœur de Docker, qui permet de créer et d'exécuter des conteneurs Docker sur une machine hôte.
- Docker Hub: Un registre cloud public de conteneurs Docker, où les utilisateurs peuvent partager, stocker et télécharger des conteneurs prêts à l'emploi.
- Docker Compose: Un outil pour définir et exécuter des applications multi-conteneurs à l'aide d'un fichier YAML pour configurer les services, les réseaux et les volumes.
- Docker Enterprise: Une plateforme complète pour créer, gérer et déployer des applications conteneurisées en production, comprenant des fonctionnalités avancées telles que la sécurité, la gestion des politiques, l'orchestration d'entreprise et le support.
- Docker Kubernetes Service (DKS): Un service Kubernetes intégré à Docker Desktop et Docker Enterprise, permettant aux développeurs de créer, exécuter et déboguer des applications Kubernetes localement.
- Docker Security Scanning: Un service de sécurité intégré à Docker Hub qui analyse les images de conteneurs pour détecter les vulnérabilités et les menaces de sécurité.
- Docker Trusted Registry (DTR): Une solution de registre privé sécurisé pour stocker, organiser et gérer les images de conteneurs dans un environnement d'entreprise.
- Ces solutions, parmi d'autres, offrent aux utilisateurs des outils puissants pour tirer parti de la technologie des conteneurs Docker dans divers scénarios d'utilisation, qu'il s'agisse de développement, de test, de déploiement ou de production.

Docker étant une plateforme logicielle pour créer, déployer et gérer des conteneurs d'application virtualisés sur un système d'exploitation commun. Il dispose d'un écosystème d'outils qui facilite son déploiement et sa gestion. Docker rend facile le déploiement d'applications avec des résultats prévisibles et reproductibles.

Pour information Docker est **utilisé par des millions de professionnels de l'informatique** dans le monde entier, et comprend la plus grande bibliothèque de contenu de conteneurs et de son écosystème, avec plus de 100 000 images de conteneurs provenant de grands fournisseurs de logiciels, de projets open source et de la communauté.