Docker

Introduction & Installation

Les bases

Untitled.png

Architecture

Untitled 1.png

Réseau

Untitled 2.png

Docker Networking

Compose

Untitled 3.png

Swarm

Untitled 4.png

Installation de docker

Get Docker

```
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

Linux / WSL

Install Docker Engine on CentOS

Install Docker Engine on Debian

Install Docker Engine on Ubuntu

Ubuntu distros manual install

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

```
sudo apt-get update


sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

WSL & Localhost issue

Fixing WSL2 localhost access issue

Portainer

Untitled.png

Architecture

Untitled 1.png

Installation

Standalone

Install Portainer with Docker on Linux

```
docker volume create portainer_data
```

```
docker run -d -p 8000:8000 -p 9000:9000 -p 9443:9443 --name portainer \
    --restart=always \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v portainer_data:/data \
    portainer/portainer-ce:2.11.1
```

Swarm

> Pensez à activer docker swarm `docker swarm init`

Install Portainer with Docker Swarm on Linux

```
version: '3.2'


services:
  agent:
```

```
    image: portainer/agent:2.11.1
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/volumes:/var/lib/docker/volumes
    networks:
      - agent_network
    deploy:
      mode: global
      placement:
        constraints: [ node.platform.os == linux ]

  portainer:
    image: portainer/portainer-ce:2.11.1
    command: -H tcp://tasks.agent:9001 --tlsskipverify
    ports:
      - "9443:9443"
      - "9000:9000"
      - "8000:8000"
    volumes:
      - portainer_data:/data
    networks:
      - agent_network
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints: [ node.role == manager ]

networks:
  agent_network:
    driver: overlay
    attachable: true

volumes:
  portainer_data:
```

```
curl -L https://downloads.portainer.io/portainer-agent-stack.yml \
    -o portainer-agent-stack.yml
```

```
docker stack deploy -c portainer-agent-stack.yml portainer
```

```
docker ps

CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS
PORTS                                NAMES
59ee466f6b15   portainer/agent:2.11.1         "./agent"                About a minute ago   Up About a
minute                               portainer_agent.xbb8k6r7j1tk9gozjku7e43wr.5sa6b3e8cl6hyu0snlt387sgv
2db7dd4bfba0   portainer/portainer-ce:2.11.1  "/portainer -H tcp:/..."  About a minute ago   Up About a minute
8000/tcp, 9000/tcp, 9443/tcp         portainer_portainer.1.gpuvu3pqmt1m19zxfo44v7izx
```

Accès à portainer

On y accède ensuite via `http://<server_url>:9000` ou `https://<server_url>:9443`

Nginx proxy manager

> Faire du reverse proxy efficace avec Docker

Prérequis

> Dans le cadre de cette explication, nous utiliserons l'interface web Portainer afin de faire no manipulations sur Docker.

VPS / Serveur autohébergé

Nous aurons également besoin d'un VPS. Je vous en recommande un personnellement qui est très peu chère grâce à une offre étudiante et fera très bien le taf :

PulseHeberg

La première offre est à 15 €/an, ce n'est rien du tout !

Vous pouvez aussi utiliser un serveur autohébergé, à condition d'avoir la redirection de port active vers ce dernier sur votre box.

> Vous devez avoir les ports **80**, **81** et **443** de disponible sur votre serveur, désactiver donc toute application les utilisants tel que apache, nginx, caddy, ...

Domaine

Puisque nous allons faire du reverse proxy, il est également nécessaire d'avoir un domaine (même gratuit) pour faire nos manipulations

Explication

Les différents types de réseau Docker

Le réseau Bridge (équivalant overlay de swarm)

Un réseau Bridgé est le type de réseau le plus utilisé dans Docker. Les réseaux Bridges créés par les utilisateurs sont semblables au réseau bridge par défaut créé à l'installation de Docker. Cependant, de nouvelles fonctionnalités sont ajoutées, la gestion du DNS par exemple. Lors de la création de nouveau réseau, une nouvelle interface est créée...

Docker daemon exécute un serveur DNS intégré qui fournit une résolution de noms aux conteneurs connectés au réseau créé par les utilisateurs, de sorte que ces conteneurs peuvent résoudre les noms de d'hôtes en adresses IP.

Si le serveur DNS intégré est incapable de résoudre la demande, il sera transmis à tous les serveurs DNS externes configurés pour le conteneur. Pour faciliter cela, lorsque le conteneur est créé, seul le serveur DNS intégré 127.0.0.11 est renseigné dans le fichier **resolv.conf** du conteneur.

Untitled.png

Le driver overlay

Si vous souhaitez une mise en réseau multihôte native, vous devez utiliser un driver overlay. Il crée un réseau distribué entre plusieurs hôtes possédant le moteur Docker. Docker gère de manière transparente le routage de chaque paquet vers et depuis le bon hôte et le bon conteneur.

Untitled 1.png

Le driver host

Ce type de réseau permet aux conteneurs d'utiliser la même interface que l'hôte. Il supprime donc l'isolation réseau entre les conteneurs et seront par défaut accessible de l'extérieur. De ce fait, il prendra la même IP que votre machine hôte.

Untitled 2.png

Ajout du réseau

Dans le cadre de notre manipulation, nous allons utiliser le réseau Bridge.

Pour cela, rendez-vous sur Portainer dans la section **Network**

Untitled 3.png

Cliquez sur **Add a network**

Untitled 4.png

Nommez votre réseau (dans mon cas **proxy_network**), sélectionnez le type de réseau **Bridge** et cochez **Enable manual container attachment**.

Créez le réseau.

Untitled 5.png

Le réseau apparait alors dans la liste des réseaux.

Untitled 6.png

Installation Nginx Proxy Manager

Rendez-vous dans la section **Containers** et cliquez sur **Add container**

Untitled 7.png

Nommez votre instance, donc mon cas, **nginx_proxy_manager**, renseignez l'image docker `jc21/nginx-proxy-manager:latest` et ouvrez les ports :

- 443 ⇒ Port d'accès HTTPS
- 80 ⇒ Port d'accès HTTP
- 81 ⇒ Port du dashboard HTTP

Untitled 8.png

Au niveau des **Volumes**, il faudra connecter :

- /data ⇒ configuration de nginx proxy manager
- /etc/letsencrypt ⇒ stockage des certificats

Untitled 9.png

Dans la section **Network**, sélectionnez le réseau **Bridge** créé précédemment, dans mon cas **proxy_network**.
Pensez également à renseigner un **Hostname.**

Untitled 10.png

Mettez la **Restart policy** à **Always**

Untitled 11.png

Cliquez ensuite sur **Deploy the container**

Untitled 12.png

Notre container est correctement lancé avec les ports configurés ouverts

Untitled 13.png

Connexion au dashboard

Pour accéder au dashboard, il faut se rendre en HTTP sur le port 81 de notre server : `http://ip-du-serveur:81/`

Les identifiants par défaut sont :

- Email : **admin@example.com**
- Password: **changeme**

Untitled 14.png

> À la première connexion, il vous sera demandé de changer vos identifiants, faites le ! Et surtout, renseignez une adresse email **valide**, c'est important pour la suite.

Une fois connecté, nous pouvons voir les différentes configurations mises en place ou non. Par défaut il n'y a rien de présent (Dans la capture, j'avais déjà setup 2-3 trucs).

Untitled 15.png

Configuration du reverse proxy sur le dashboard

Actuellement, nous accédons au dashboard sur le port 81 avec une connexion HTTP. Ça n'est pas **ACCEPTABLE**. Le premier reverse proxy que nous allons créer sera destiné à accéder au dashboard lui-même via un domaine et une connexion HTTPS sécurisé.

Rendez-vous sur **Hosts → Proxy Hosts**, puis cliquez sur **Add Proxy Host**

Untitled 16.png

Renseignez les différents champs du premier onglet **Details** :

- **Domain Names** : Le nom du domain par lequel nous souhaiterons accéder à notre reverse proxy, dans le cas du dashboard je l'appellerais **proxy.nospy.fr**
- **Scheme** : Le protocole d'accès à notre service interne. En l'occurrence, ça sera du HTTP.
- **Forward Hostname / IP** : Il s'agit du nom d'hôte / adresse ip de ce que nous souhaitons atteindre ***au sein du réseau docker proxy_network***.
  Puisque nous cherchons à atteindre le dashboard se trouvant sur le même container, nous renseignerons l'ip local 127.0.0.1 (mais si vous avez bien suivi, on aurait également pu mettre le nom d'hôte que nous avions renseigné, à savoir **nginx_proxy**).
- **Forward Port** : Le port de destination. Le dashboard est accessible sur le port 81.
- Le reste des options parlent pour elles-mêmes, mais sont facultatives dans le cas du dashboard.

Untitled 17.png

Rendez-vous ensuite sur l'onglet **SSL**, où nous allons configurer le certificat SSL de notre site ainsi que la redirection HTTP → HTTPS.

- Sur le champ **SSL Certificate**, sélectionnez **Request a new SSL Certificate** pour générer un certificat Let's Encrypt et ainsi sécuriser notre page.
- Penser à cocher l'option **Force SSL**, c'est cette dernière qui forcera la redirection HTTP → HTTPS.
- **HTTP/2 Support** permet d'utiliser la version 2 de HTTP
  https://www.hosteur.com/ressources/articles/adopter-http-2
- **HSTS Enabled** améliore la sécurité de notre flux https://www.globalsign.com/fr/blog/qu-est-ce-que-le-hsts-comment-le-mettre-en-uvre
- Veillez à ce que l'adresse email renseigné pour **Let's Encrypt** soit correct ! Sinon ça peut ne pas fonctionner…

Untitled 18.png	**Untitled 19.png**

Cliquez sur **Save** et patientez le temps que Nginx Proxy Manager mette en place le reverse proxy et configure le certificat SSL (cette étape peux prendre un peu plus d'une minute la première fois et nécessite que vous ayez correctement configuré votre nom de domain pour qu'il pointe sur votre serveur).

Si tout s'est bien passé, nous devrions apercevoir dans la section **Hosts → Proxy Hosts** une ligne concernant notre reverse proxy mis en place.

La section **SSL certificates** devrait également contenir le certificat SSL Let's Encrypt généré pour notre nom de domaine. Le renouvellement de ce dernier est automatique, mais peut être déclenché manuellement.

Untitled 20.png

Untitled 21.png

Désormais, si nous essayons de nous connecter à notre domaine et qu'il est bien configuré (dans mon cas, proxy.nospy.fr) , nous devrions accéder à notre dashboard de manière sécurisée.

Untitled 22.png

Et voilà !

Une fois que vous avez bien accès à votre dashboard, de façon sécurisé cette fois-ci, nous allons modifier notre container pour fermer le port 81 au niveau du binding, car nous n'avons plus besoin qu'il soit accessible de l'extérieur grâce au reverse proxy interne.

Untitled 23.png

Untitled 24.png

Untitled 25.png

Reverse proxy sur un autre container

> Ayez au préalable configuré vos noms de domaines et sous-domaines que vous comptez utiliser pour qu'ils pointent correctement sur vos serveurs

Maintenant que nous avons un dashboard sécurisé, nous allons configurer un reverse proxy sur un autre container. Dans mon cas, j'utiliserais une image **nginx** toute simple.

N'importe qu'elle image ayant un service web dessus fonctionnerait, vous devez juste avoir la connaissance de cette image, notamment le port utilisé, s'il nécessite l'activation de web sockets...

Je vais donc ajouter un container, et le mettre sur mon réseau créé précédemment (**proxy_network**) tout en spécifiant un nom d'hôte spécifique (dans mon cas **nginx_container)**.

Untitled 26.png

Je ne ferai pas de redirection de port vers ma machine hôte, car tout sera géré en interne via notre reverse proxy.

Untitled 27.png

Retournons désormais sur **Nginx Proxy Manager** dans la section **Hosts → Proxy Hosts** et ajoutez un nouveau proxy.

- Au niveau du domaine, renseignez celui que vous souhaitez utiliser.

- Pour le **Scheme**, renseignez vous sur ce que votre image utilise en interne (dans le cas de mon image nginx ça sera du http).
- **Hostname** : le nom d'hôte du container cible (ici, je l'avais nommé **nginx_container**).
- **Forward Port** : même chose que le scheme (ici, 80).

Untitled 28.png

Dans l'onglet **SSL**, il faut aussi configurer correctement notre accès sécurisé, comme pour le dashboard.

Untitled 29.png

On sauvegarde et on va vérifier que tout s'est bien passé au niveau des **Proxy Hosts**, **SSL Certificates**.

Enfin, on va sur le domaine en question pour s'assure que notre site est bien en ligne.

Untitled 30.png

Untitled 31.png

Untitled 32.png

Redirection www.domain → domain

Dans certaines situations, on souhaiterais faire en sorte que notre site soit disponible sous le domaine www.domaine.com et également domaine.com. Dans cette situation, on aimerait bien que le domaine en www. redirige vers celui sans.

Pour cet exemple, je vais reprendre le site nginx que j'ai exposé dans l'exemple précédent.

Pour ce faire, nous allons nous rendre dans la section **Hosts → Redirection Hosts**

Cliquez sur **Add Redirection Host** et remplissez les champs comme il se doit.

- **Domain names** : Les différents domaines que l'on souhaite rediriger
- **Scheme** : le protocol HTTP sur lequel rediriger (on preferera HTTPS tout de même)
- **Forward domain** : le domain de destination sur lequel rediriger
- **HTTP Code** : le code HTTP de redirection que l'on souhaite transmettre
- **Preserve Path** : On garde le suffixe de notre requête et ne changeons que le domaine.

Untitled 33.png

Désormais, si je me rends sur le site www.nginx.nospy.fr, je serais automatiquement redirigé vers nginx.nospy.fr

Untitled 34.png

Wildcard DNS record

Celle-là, c'est cadeau !

Si vous avez un serveur sur lequel vous souhaitez rediriger plein de sous domain sans vous embêter à devoir les déclarer en permanence sur votre hébergeur DNS, vous pouvez déclarer un Wildcard DNS record.

Cela fonctionne un peu comme un sorte de **Regex** où tout les sous domaines concernés par le filtre irons automatiquement vers une adresse IP, sans avoir à les configurer à chaque fois.

Dans mon exemple, je possède un domaine chez **OVH**, mais l'opération est faisable chez d'autre.

Admettons que je veuille que tout les domaine finissant par **.app.nospy.fr** redirige vers le même serveur (proxy.app.nospy.fr, vecolo.app.nospy.fr, plex.app.nospy.fr, ...).

Il faut se rendre sur votre zone DNS et déclarer deux entrées.

- La première sera une entrée de type **A**, concernera le sous domaine **app.nospy.fr**, elle pointera vers l'ip de votre serveur.

Untitled 35.png

- La deuxième concernera tout les sous domaines qui découlerons de ce suffixe. Pour faire ça, on ajoute une nouvelle entrée de type **A**, mais qui concernera cette fois le sous domaine ***.app.nospy.fr**, et pointera également vers l'ip de votre serveur.

Untitled 36.png

Vous vous retrouverez alors ave ces deux entrées, et désormais tout vos sous domaine irons automatiquement dessus, sauf pour les exceptions que vous aurez ajoutées vous-même □□

Untitled 37.png

Ressources

Fonctionnement et manipulation du réseau dans Docker

Nginx Proxy Manager

Qu'est-ce que le HTTP/2 et pourquoi l'adopter ?

Qu'est-ce que le HSTS et comment le met-on en œuvre ? - Blog - GlobalSign

Wildcard DNS record - Wikipedia

OVH Autorise enfin les wildcards pour les DNS des noms de domaines!

Stacks

My favorites docker stacks

Utils

Services

Nginx proxy manager

Env

- TZ

Volumes

- /config

Shepherd

Automated service image updater

Env

- SHEPHERD_SLEEP_TIME
- SHEPHERD_IMAGE_AUTOCLEAN_LIMIT
- TZ

Ofelia

Docker cron scheduler

Env

- TZ

Volumes

- /var/run/docker.sock

Stack

```
version: '3'
services:
  nginx_proxy_manager:
    image: jlesage/nginx-proxy-manager:latest
    restart: always
    ports:
      - '80:8080' # Public HTTP Port
      - '443:4443' # Public HTTPS Port
      #- '81:8181' # Admin Web Port
      # Add any other Stream port you want to expose
      # - '21:21' # FTP
    environment:
      TZ: ${TZ}
      # Uncomment this if IPv6 is not enabled on your host
      DISABLE_IPV6: 'true'
    volumes:
      - nginx_proxy_data:/config
    networks:
      - proxy_network

  shepherd:
    image: mazzolino/shepherd:latest
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      SLEEP_TIME: ${SHEPHERD_SLEEP_TIME}
      IMAGE_AUTOCLEAN_LIMIT: ${SHEPHERD_IMAGE_AUTOCLEAN_LIMIT}
      TZ: ${TZ}
    deploy:
      placement:
        constraints:
          - node.role == manager

  scheduler:
```

```
    image: mcuadros/ofelia:latest
    command: daemon --docker
    environment:
      TZ: ${TZ}
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      #labels:
      #ofelia.job-local.my-test-job.schedule: "@every 5m"
      #ofelia.job-local.my-test-job.command: "date"



volumes:
  nginx_proxy_data:
    external: true

networks:
  proxy_network:
    external: true
```

Filebrowser

Services

Filebrowser

Filebrowser is a web file manager that allows you to manage your files in a web browser.

Env

- TZ
- PUID
- PGID

Volumes

- /config
- /database
- /srv

Stack

```
version: '3'

services:
  app:
    image: 'filebrowser/filebrowser:s6'
    volumes:
      - filebrowser_settings:/config
      - filebrowser_data:/database
      - /mnt/disk:/srv
    environment:
      TZ: ${TZ}
```

```
      PGID: ${PGID}
      PUID: ${PUID}
    networks:
      - proxy_network

volumes:
  filebrowser_settings:
  filebrowser_data:

networks:
  proxy_network:
    external: true
```

Nextcloud

Services

MariaDB

Env

- MYSQL_ROOT_PASSWORD

Volumes

- /var/lib/mysql

Command

- `--transaction-isolation=READ-COMMITTED --binlog-format=ROW`

Nextcloud

Env

- APACHE_DISABLE_REWRITE_IP
- OVERWRITEHOST
- OVERWRITEPROTOCOL
- OVERWRITEWEBROOT
- PHP_MEMORY_LIMIT
- PHP_UPLOAD_LIMIT
- TRUSTED_PROXIES
- PGID
- PUID
- TZ

Volumes

- /var/www/html
- /var/www/html/data

Stack

```
version: '3'

services:
  db:
    image: mariadb:10.5
    restart: always
    command: --transaction-isolation=READ-COMMITTED --binlog-format=ROW
    volumes:
      - mariadb:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
    networks:
      - nextcloud_network

  app:
    depends_on:
      - db
    image: nextcloud:latest
    restart: always
    volumes:
      - config:/var/www/html
      - ${NEXTCLOUD_DATA_PATH}:/var/www/html/data
    environment:
      APACHE_DISABLE_REWRITE_IP: 1
      OVERWRITEHOST: nextcloud.mydomain.fr
      OVERWRITEPROTOCOL: https
      OVERWRITEWEBROOT: /
      PHP_MEMORY_LIMIT: 4G
      PHP_UPLOAD_LIMIT: 10G
      TRUSTED_PROXIES: IPV4 Gateway - 10.0.3.1
      PGID: ${PGID}
      PUID: ${PUID}
      TZ: ${TZ}
```

```
    networks:
      - proxy_network
      - nextcloud_network

volumes:
  config:
  mariadb:

networks:
  proxy_network:
    external: true
  nextcloud_network:
```

JDownloader

Services

JDownloader

Env

- MYJD_DEVICE_NAME
- MYJD_PASSWORD
- MYJD_USER
- XDG_DOWNLOAD_DIR
- PUID
- PGID

Volumes

- /downloads
- /opt/JDownloader
- /opt/JDownloader/cfg

Stack

```
version: '3'


services:
  app:
    image: jaymoulin/jdownloader:latest
    ports:
      - '3129:3129'
    volumes:
      - ${DIRECT_DOWNLOAD_PATH}:/downloads
      - app:/opt/JDownloader
```

```
      - config:/opt/JDownloader/app/cfg
    environment:
      MYJD_DEVICE_NAME: ${MYJD_DEVICE_NAME}
      MYJD_PASSWORD: ${MYJD_PASSWORD}
      MYJD_USER: ${MYJD_USER}
      XDG_DOWNLOAD_DIR: /downloads
      PUID: ${PUID}
      PGID: ${PGID}

volumes:
  config:
  app:
```

Sonarqube

Services

Postgres

Env

- ALLOW_EMPTY_PASSWORD
- POSTGRESQL_USERNAME
- POSTGRESQL_DATABASE
- TZ

Volumes

- /bitnami/postgresql
- /docker-entrypoint-initdb.d
- /docker-entrypoint-preinitdb.d

Sonarqube

Env

- ALLOW_EMPTY_PASSWORD
- SONARQUBE_DATABASE_HOST
- SONARQUBE_DATABASE_PORT_NUMBER
- SONARQUBE_DATABASE_USER
- SONARQUBE_DATABASE_NAME
- SONARQUBE_USERNAME
- SONARQUBE_PASSWORD
- SONARQUBE_EMAIL
- TZ

Volumes

- /bitnami/sonarqube

Stack

```
version: '3'
services:
  postgresql:
    image: docker.io/bitnami/postgresql:13
    volumes:
      - 'postgresql_data:/bitnami/postgresql'
      - 'entry_point:/docker-entrypoint-preinitdb.d'
      - 'entry_point:/docker-entrypoint-initdb.d'
    environment:
      # ALLOW_EMPTY_PASSWORD is recommended only for development.
      ALLOW_EMPTY_PASSWORD: 'yes'
      POSTGRESQL_USERNAME: ${SONARQUBE_DB_USER}
      POSTGRESQL_DATABASE: ${SONARQUBE_DB_NAME}
      TZ: ${TZ}
    networks:
      - proxy_network
  sonarqube:
    image: bitnami/sonarqube:9.6.1
      #ports:
    # - '8080:8080'
    volumes:
      - 'sonarqube_data:/bitnami/sonarqube'
    depends_on:
      - postgresql
    environment:
      # ALLOW_EMPTY_PASSWORD is recommended only for development.
      ALLOW_EMPTY_PASSWORD: 'yes'
      SONARQUBE_DATABASE_HOST: postgresql
      SONARQUBE_DATABASE_PORT_NUMBER: 5432
      SONARQUBE_DATABASE_USER: ${SONARQUBE_DB_USER}
      SONARQUBE_DATABASE_NAME: ${SONARQUBE_DB_NAME}
      SONARQUBE_USERNAME: ${SONARQUBE_USER}
      SONARQUBE_PASSWORD: ${SONARQUBE_PASSWORD}
      SONARQUBE_EMAIL: ${SONARQUBE_EMAIL}
```

```
      TZ: ${TZ}
    networks:
      - proxy_network
volumes:
  postgresql_data:
  sonarqube_data:
  entry_point:

networks:
  proxy_network:
    external: true
```