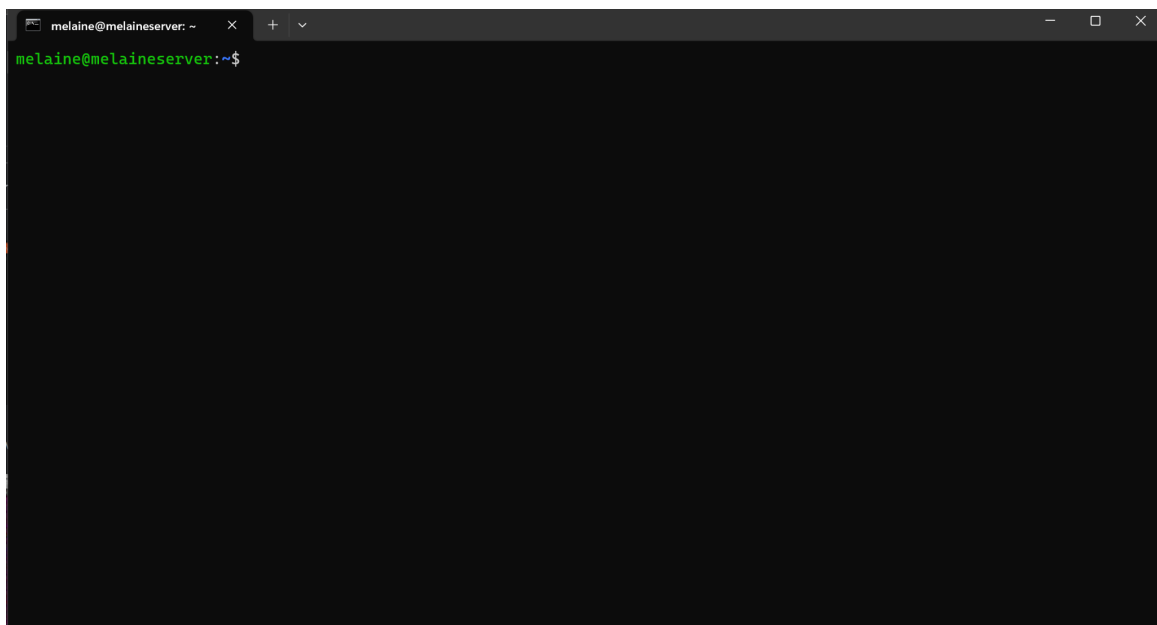


FAVENNEC Mélaine
GUENARD Juliette

Documentation d'exploitation

(outils et services mis en place)



SOMMAIRE

- I. Quelques informations
- II. SSH et DNS
- III. Redémarrage automatique
- IV. Serveur apache
- V. WordPress avec docker
- VI. Load Balancing
- VII. Certificats
- VIII. Reverse proxy

I. Quelques informations

Pour rappel nous utilisons un serveur ubuntu.

Tous nos sites WordPress seront stockés à la racine de notre serveur dans le dossier **srv/**, de même que pour notre serveur backend stocké dans le dossier **backend/**.

Nous avons mis en place deux utilisateurs afin de pouvoir travailler chacun de notre côté sur le même serveur :

```
melaine@melaineserver:~$ sudo cut -d: -f1 /etc/passwd
melaine
juliette
```

Puis on peut se connecter en SSH (cette partie sera expliquée dans le prochain chapitre).

Voici les ports ouvert du routeur (box internet) :

#	Nom	Protocole	Type	Ports externes	IP de destination	Ports de destination
1	SERVEUR_UBUNTU	TCP	Port	2222	192.168.1.53	22
2	SERVEUR_UBUNTU_	TCP	Port	80	192.168.1.53	80
3	PLAGE_UBUNTU	TCP	Plage	8081-8200	192.168.1.53	8081-8200
4	<input type="text"/>	TCP <input type="button" value="v"/>	Plage <input type="button" value="v"/>	<input type="text"/> - <input type="text"/>	192.168.1. <input type="button" value="v"/>	<input type="text"/> - <input type="text"/>

192.168.1.1 → accéder à la box

Pour gérer nos sites web nous avons choisi d'utiliser Docker car c'est une technologie de conteneurisation légère et portable qui permet de créer, déployer et exécuter des applications de manière efficace. Docker facilite la gestion des dépendances, l'isolation des environnements et la mise à l'échelle des applications.

II. SSH et DNS

La mise en place d'une connexion SSH est importante pour **sécuriser** les communications, accéder à distance à notre serveur, gérer l'administration système, transférer des fichiers en toute sécurité et contrôler les autorisations d'accès.

Dans un premier temps il faut installer ssh :

```
sudo apt-get install openssh-server
```

```
melaine@melaineserver:~$ sudo service ssh status
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2023-06-16 11:41:37 UTC; 1 day 10h ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 106436 (sshd)
      Tasks: 1 (limit: 9271)
     Memory: 15.8M
        CPU: 5min 5.118s
    CGroup: /system.slice/ssh.service
            └─106436 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

Puis Le configurer :

```
sudo ufw allow ssh
```

 (on le configurant sur le port 2222)

Enfin, ajouter les utilisateurs :

```
sudo adduser <username>
```


Et pour finir, pouvoir se connecter :

```
ssh -p <port> <username>@<server-address>
```

Par exemple dans notre cas il suffit d'écrire :

```
ssh -p 2222 melaine@93.2.21.129 ou ssh -p 2222 melaine@meljujuserver.duckdns.org
```

Mais pour que la commande au dessus fonctionne, il faut mettre en place notre DNS. Nous avons choisi DUCK DNS pour se faire :



Duck DNS

account melaine.favennec@gmail.com
type free
token fdf8ebac-f37a-471e-848a-d0bd61bf8319
token 1 week ago
generated 9 Jun 2023, 16:21:34
created date

domains 1/5

domain	current ip	ipv6	changed
meljuju	<input type="text" value="93.2.21.129"/> <input type="button" value="update ip"/>	<input type="text" value="ipv6 address"/> <input type="button" value="update ipv6"/>	1 week ago <input type="button" value="delete domain"/>

This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

Nous avons donc :

Une ip dns : 93.2.21.129

Et un nom de domaine: meljuju.duckdns.org

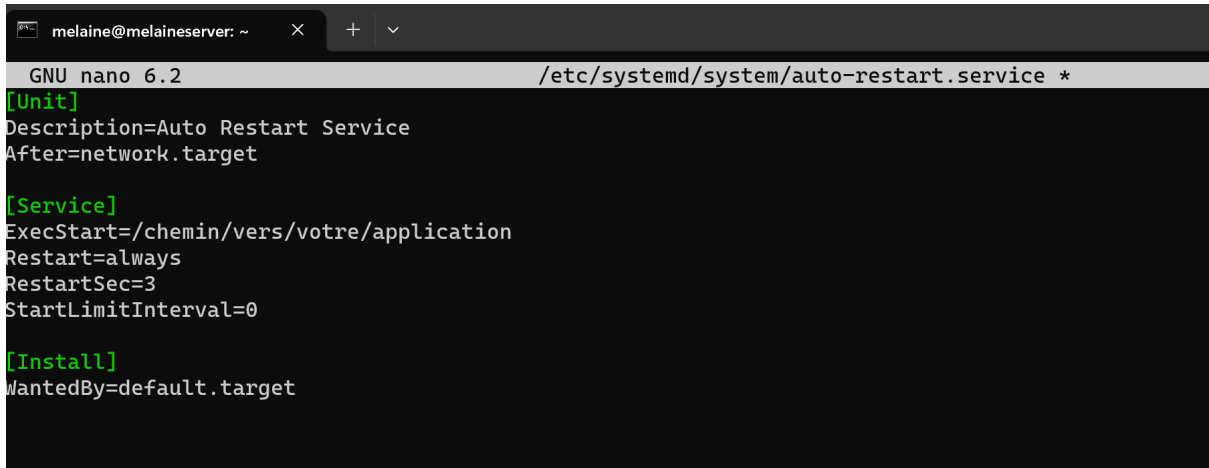
Mais pourquoi mettre en place un **DNS** ?

Mettre en place un DNS sur un serveur Ubuntu permet de traduire les noms de domaine en adresses IP, facilitant ainsi l'accès aux services hébergés en permettant aux utilisateurs d'accéder à ces services en utilisant des noms conviviaux plutôt que des adresses IP complexes.

III. Redémarrage automatique

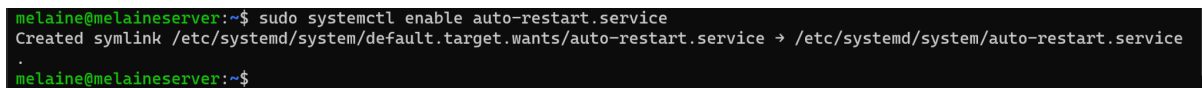
Il est important de mettre en place un redémarrage automatique en cas de crash de notre serveur, cela permet de redémarrer le serveur et ses services le plus rapidement possible.

On crée un fichier **auto_restart.service** qui va permettre de gérer automatiquement le redémarrage en cas de crash :



```
melaine@melaineserver: ~  
GNU nano 6.2 /etc/systemd/system/auto-restart.service *  
[Unit]  
Description=Auto Restart Service  
After=network.target  
  
[Service]  
ExecStart=/chemin/vers/votre/application  
Restart=always  
RestartSec=3  
StartLimitInterval=0  
  
[Install]  
WantedBy=default.target
```

Une fois le fichier enregistré on active le service :

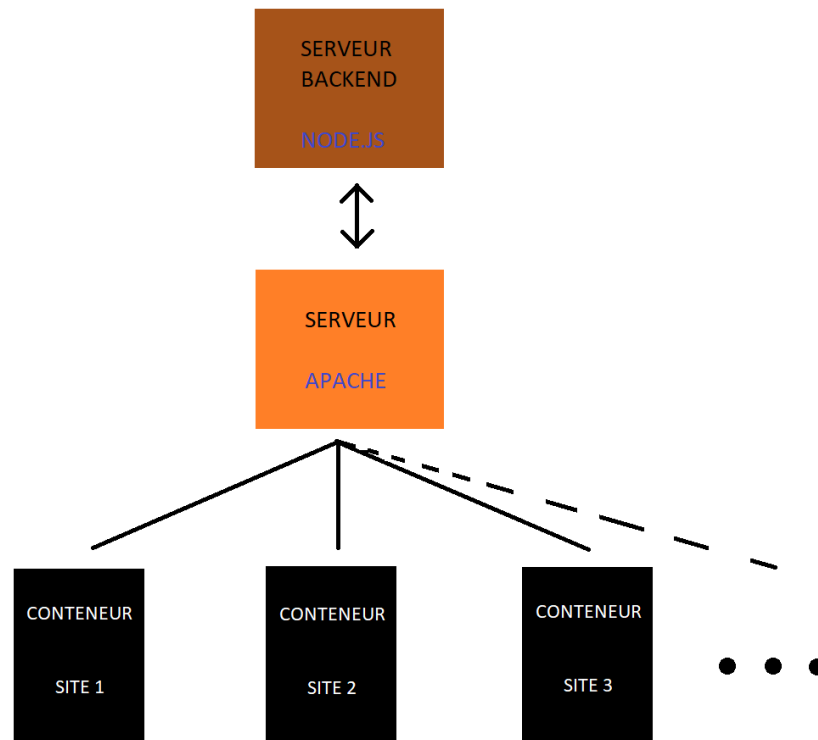


```
melaine@melaineserver:~$ sudo systemctl enable auto-restart.service  
Created symlink /etc/systemd/system/default.target.wants/auto-restart.service -> /etc/systemd/system/auto-restart.service  
.  
melaine@melaineserver:~$
```

Et voilà, cette fonctionnalité est mise en place !

IV. Serveur apache

Avant de commencer, voici un petit schéma pour expliquer comment va fonctionner notre serveur ubuntu :



Le serveur backend permet de gérer la gestion des données dans une application web. Il reçoit les requêtes des clients, traite les données, interagit avec la base de données, exécute la logique métier et renvoie les réponses aux clients.

Dans un premier temps il faut mettre en place notre serveur apache :

```
melaine@melaineserver:/srv$ sudo apt install apache2
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
apache2 est déjà la version la plus récente (2.4.52-1ubuntu4.5).
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  linux-headers-5.15.0-72 linux-headers-5.15.0-72-generic linux-image-5.15.0-72-generic
  linux-modules-5.15.0-72-generic linux-modules-extra-5.15.0-72-generic
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
melaine@melaineserver:/srv$
```

Une fois apache installé, on vérifie qu'il est actif :

```
melaine@melaineserver:/srv$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2023-06-16 12:28:24 UTC; 3s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 123310 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
  Main PID: 123315 (apache2)
    Tasks: 55 (limit: 9271)
   Memory: 5.0M
      CPU: 27ms
   CGroup: /system.slice/apache2.service
           └─123315 /usr/sbin/apache2 -k start
             └─123316 /usr/sbin/apache2 -k start
               └─123317 /usr/sbin/apache2 -k start

juin 16 12:28:24 melaineserver systemd[1]: Starting The Apache HTTP Server...
juin 16 12:28:24 melaineserver apachectl[123314]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name: [123314]
juin 16 12:28:24 melaineserver systemd[1]: Started The Apache HTTP Server.
lines 1-17/17 (END)
```

avec succès.

C'est bon il fonctionne !

Notre serveur apache va permettre de faire tourner les conteneurs avec nos sites web ! A présent mettons en place docker (chapitre suivant).

V. WordPress avec docker

Maintenant que nous avons notre serveur apache nous pouvons mettre en place nos conteneurs avec l'image de WordPress pour faire fonctionner nos sites.

Docker sera utile dans ce cas car il permet d'isoler chaque application web dans des conteneurs, garantissant ainsi une séparation des environnements et des dépendances. Cela facilite le déploiement, la gestion et la mise à l'échelle des applications. De plus, Docker permet de reproduire facilement l'environnement de développement sur différents serveurs, assurant ainsi la portabilité des applications.

Mettons docker en place :

```
melaine@melaineserver:~$ sudo docker version
Client: Docker Engine - Community
Version: 24.0.2
API version: 1.43
Go version: go1.20.4
Git commit: cb74dfc
Built: Thu May 25 21:51:00 2023
OS/Arch: linux/amd64
Context: default

Server: Docker Engine - Community
Engine:
Version: 24.0.2
API version: 1.43 (minimum version 1.12)
Go version: go1.20.4
Git commit: 659604f
Built: Thu May 25 21:51:00 2023
OS/Arch: linux/amd64
Experimental: false
```

Une fois que cela est fait, il nous faut créer un conteneur pour notre site web. On va faire en sorte de séparer le front du back, pour se faire on va créer un réseau docker pour pouvoir faire communiquer nos deux conteneurs (front et back) de notre site :

```
docker network create wordpress-network
```

```
melaine@melaineserver:/srv$ docker network create wordpress-network
71181c2a2863f678da0e173cb62735f8325d0b4e85b961762514ed01acce0f81
```

Avant de pouvoir continuer, il nous faut créer l'image docker de notre site WordPress : Grâce à un DockerFile qui contient une série d'instructions on va pouvoir générer notre image :

```
melaine@melaineserver: /srv/ × + ▾
GNU nano 6.2 Dockerfile *
# Utilisez une image de base avec Apache et PHP
FROM php:7.4-apache

# Copiez les fichiers de votre site WordPress dans le conteneur
COPY . /var/www/html

# Définissez le répertoire de travail
WORKDIR /var/www/html

# Exposez le port 80 pour permettre l'accès HTTP
EXPOSE 8081

# Installez les dépendances PHP nécessaires
RUN docker-php-ext-install mysqli && \
    docker-php-ext-enable mysqli

# Définissez les variables d'environnement pour la base de données
ENV WORDPRESS_DB_HOST=mysql \
    WORDPRESS_DB_NAME=wordpress \
    WORDPRESS_DB_USER=root \
    WORDPRESS_DB_PASSWORD=<root_password>

# Activez le module Apache mod_rewrite
RUN a2enmod rewrite

# Redirigez les erreurs de journalisation vers la sortie standard
```

On remarque qu'il faut mettre en place une base de données pour notre WordPress !
Faisons le :

Dans un premier temps on va installer mysql :

```
melaine@melaineserver: /srv/WordPress_container$ sudo apt-get install mysql-server
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  linux-headers-5.15.0-72 linux-headers-5.15.0-72-generic linux-image-5.15.0-72-generic
  linux-modules-5.15.0-72-generic linux-modules-extra-5.15.0-72-generic
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
Les paquets supplémentaires suivants seront installés :
  libcgi-fast-perl libcgi-pm-perl libclone-perl libencode-locale-perl libevent-pthreads-2.1-7 libfcgi-bin libfcgi-perl
  libfcgi0ldbl libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl libhttp-date-perl libhttp-message-perl
  libio-html-perl liblwp-mediatypes-perl libmecab2 libprotobuf-lite23 libtimedate-perl liburi-perl mecab-ipadic
  mecab-ipadic-utf8 mecab-utils mysql-client-8.0 mysql-client-core-8.0 mysql-common mysql-server-8.0
  mysql-server-core-8.0
Paquets suggérés :
  libdata-dump-perl libipc-sharedcache-perl libbusiness-isbn-perl libwww-perl mailx tinycat
Les NOUVEAUX paquets suivants seront installés :
  libcgi-fast-perl libcgi-pm-perl libclone-perl libencode-locale-perl libevent-pthreads-2.1-7 libfcgi-bin libfcgi-perl
  libfcgi0ldbl libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl libhttp-date-perl libhttp-message-perl
```

Et l'on va le configurer :

sudo mysql -u root -p : On rentre dans mySql

CREATE DATABASE wordpress; : On créer notre base de donnée

CREATE USER 'melaine'@'localhost' IDENTIFIED BY 'meljuju1234'; : Si l'on a besoin on créer un utilisateur

GRANT ALL PRIVILEGES ON wordpress.* TO 'melaine'@'localhost'; : On lui donne les droits sur la base de données

FLUSH PRIVILEGES;

La création d'un utilisateur dans MySQL permet de définir des identifiants de connexion pour accéder à la base de données. Cela permet de contrôler l'accès à la base de données et d'attribuer des privilèges spécifiques à cet utilisateur.

Maintenant remplaçons dans notre Dockerfile la partie sur la base de donnée pour y mettre nos valeurs :

```
GNU nano 6.2 Dockerfile *
# Utilisez une image de base avec Apache et PHP
FROM php:7.4-apache

# Copiez les fichiers de votre site WordPress dans le conteneur
COPY . /var/www/html

# Définissez le répertoire de travail
WORKDIR /var/www/html

# Exposez le port 8081 pour permettre l'accès HTTP
EXPOSE 8081

# Installez les dépendances PHP nécessaires
RUN docker-php-ext-install mysqli && \
    docker-php-ext-enable mysqli

# Définissez les variables d'environnement pour la base de données
ENV WORDPRESS_DB_HOST=mysql \
    WORDPRESS_DB_NAME=wordpress \
    WORDPRESS_DB_USER=root \
    WORDPRESS_DB_PASSWORD=meljuju1234

# Activez le module Apache mod_rewrite
RUN a2enmod rewrite

# Redirigez les erreurs de journalisation vers la sortie standard
```

Et voilà le Dockerfile est configuré, nous pouvons donc créer les deux conteneurs :

```
docker run -d --name wordpress-db --network wordpress-network -e
MYSQL_ROOT_PASSWORD=<root_password> -e MYSQL_DATABASE=wordpress mysql:latest
```

Ce premier conteneur permet de faire fonctionner le back du site web (la base de données). Il utilise l'image MySQL pour exécuter le serveur de base de données. Les options "-e" sont utilisées pour définir des variables d'environnement, telles que le mot de passe root de MySQL et le nom de la base de données.

On fait pareil pour faire fonctionner le serveur WordPress :

```
docker run -d --name wordpress --network wordpress-network -p 8082:80 -v
/srv/WordPress_container/wordpress_file:/var/www/html -e WORDPRESS_DB_HOST=wordpress-db -e
WORDPRESS_DB_NAME=wordpress -e WORDPRESS_DB_USER=root -e
WORDPRESS_DB_PASSWORD=meljuju1234 wordpress:latest
```

Le deuxième conteneur, nommé "wordpress", utilise l'image WordPress pour exécuter le serveur WordPress. Les options "-p" sont utilisées pour faire communiquer le port 8082 (dans notre cas, cela varie d'un site à l'autre car ils utilisent des ports différents) du conteneur et le port 80, permettant ainsi d'accéder au site WordPress depuis l'extérieur.

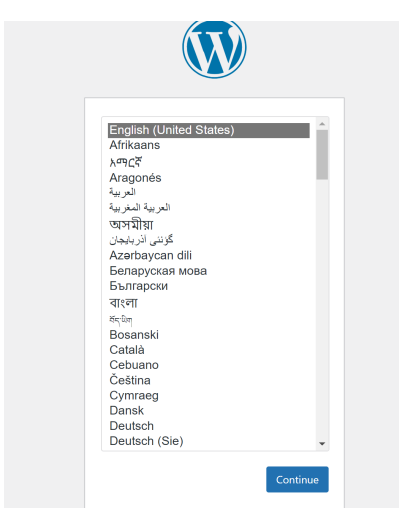
On vérifie qu'ils soient bien en cours d'exécution :

```
melaine@melaineserver:/srv/WordPress_container$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                                                                 NAMES
3ac711febe3c   wordpress:latest   "docker-entrypoint.s..." 23 seconds ago Up 15 seconds 0.0.0.0:8082->80/tcp, :::8082->80/tcp   wordpress-front
eb712bcb7035   mysql:latest      "docker-entrypoint.s..." 11 minutes ago Up 11 minutes 3306/tcp, 33060/tcp               wordpress-db
917315f0f71e   wordpress-image    "docker-php-entrypoi..." 17 minutes ago Up 17 minutes 80/tcp, 0.0.0.0:8081->8081/tcp, :::8081->8081/tcp   sad_dewdney
```

Cela fonctionne !

Vous pouvez à présent configurer votre WordPress en nous rendant à l'adresse web :
dns:port

(par exemple **93.2.21.129:8082**) :



Veuillez renseigner les informations suivantes. Ne vous inquiétez pas, vous pourrez les modifier plus tard.

Titre du site

Identifiant
Les identifiants ne peuvent utiliser que des caractères alphanumériques, des espaces, des tirets bas (" _ "), des traits d'union (" - "), des points et le symbole @.

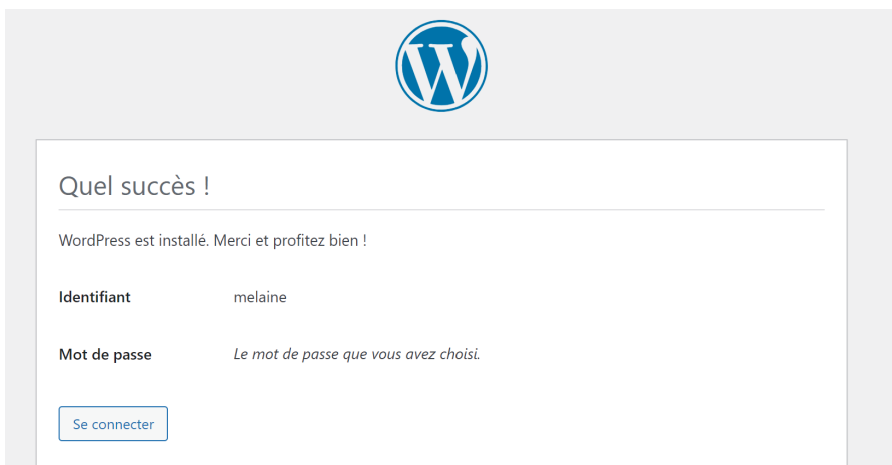
Mot de passe
Medium

Important : Vous aurez besoin de ce mot de passe pour vous connecter. Pensez à le stocker dans un lieu sûr.

Votre e-mail
Vérifiez bien cette adresse e-mail avant de continuer.

Visibilité par les moteurs de recherche ☒ Demander aux moteurs de recherche de ne pas indexer ce site
Certains moteurs de recherche peuvent décider de l'indexer malgré tout.

On met nos valeurs (nom de la bdd créer avec mysql précédemment) :



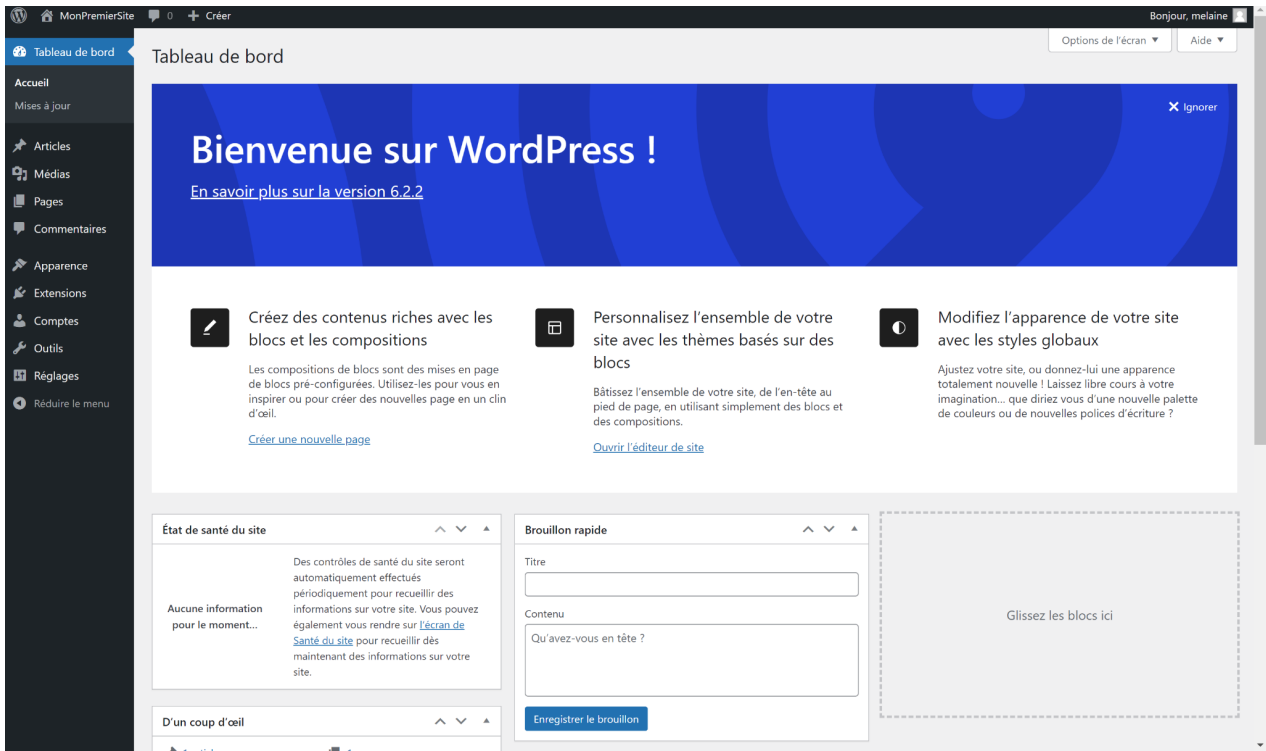
Quel succès !

WordPress est installé. Merci et profitez bien !

Identifiant melaine

Mot de passe Le mot de passe que vous avez choisi.

Vous pouvez à présent gérer votre site :



VI. Load Balancing

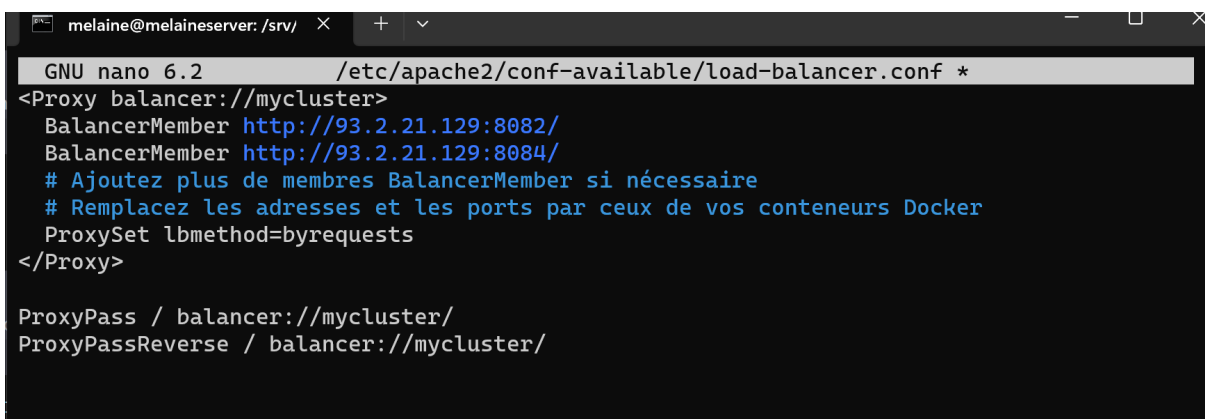
Notre serveur apache va contenir plusieurs conteneurs avec nos sites web, il faut donc mettre en place un load balancing.

Le load balancing permet de répartir le trafic entrant de manière équilibrée entre les différents conteneurs Docker. Cela améliore la performance du système en évitant la surcharge d'un seul conteneur et en assurant une meilleure utilisation des ressources disponibles.

Avant de pouvoir le faire fonctionner il nous faut mettre en place quelques extensions :

```
melaine@melaineserver:/srv/WordPress_container_3$ sudo a2enmod proxy
Enabling module proxy.
To activate the new configuration, you need to run:
  systemctl restart apache2
melaine@melaineserver:/srv/WordPress_container_3$ sudo a2enmod proxy_balancer
Considering dependency proxy for proxy_balancer:
Module proxy already enabled
Considering dependency alias for proxy_balancer:
Module alias already enabled
Considering dependency slotmem_shm for proxy_balancer:
Enabling module slotmem_shm.
Enabling module proxy_balancer.
To activate the new configuration, you need to run:
  systemctl restart apache2
melaine@melaineserver:/srv/WordPress_container_3$ sudo a2enmod lbmethod_byrequests
Considering dependency proxy_balancer for lbmethod_byrequests:
Considering dependency proxy for proxy_balancer:
Module proxy already enabled
Considering dependency alias for proxy_balancer:
Module alias already enabled
Considering dependency slotmem_shm for proxy_balancer:
Module slotmem_shm already enabled
Module proxy_balancer already enabled
Enabling module lbmethod_byrequests.
To activate the new configuration, you need to run:
  systemctl restart apache2
melaine@melaineserver:/srv/WordPress_container_3$
```

Pour faire fonctionner le load balancing nous allons créer un fichier de configuration qui va stocker tous les ports actifs (nos sites web) et d'autres options :



```
melaine@melaineserver: /srv/ x + v
GNU nano 6.2 /etc/apache2/conf-available/load-balancer.conf *
<Proxy balancer://mycluster>
  BalancerMember http://93.2.21.129:8082/
  BalancerMember http://93.2.21.129:8084/
  # Ajoutez plus de membres BalancerMember si nécessaire
  # Remplacez les adresses et les ports par ceux de vos conteneurs Docker
  ProxySet lbmethod=byrequests
</Proxy>

ProxyPass / balancer://mycluster/
ProxyPassReverse / balancer://mycluster/
```

On active le fichier de configuration :

```
melaine@melaineserver:/srv/WordPress_container_3$ sudo a2enconf load-balancer
Enabling conf load-balancer.
To activate the new configuration, you need to run:
    systemctl reload apache2
melaine@melaineserver:/srv/WordPress_container_3$ sudo systemctl restart apache2
melaine@melaineserver:/srv/WordPress_container_3$
```

Maintenant, Apache agit comme un équilibrage de charge pour nos conteneurs Docker exécutant les sites WordPress.

VII. Certificats

Il est utile d'avoir des certificats sur notre serveur Ubuntu pour sécuriser les communications et assurer l'authenticité des sites Web, en permettant le chiffrement des données. Pour ce faire nous allons utiliser Let's Encrypt qui permet de fournir à partir d'un domaine, un certificat.

Passons maintenant à la configuration des certificats grâce à Let's Encrypt :

```
melaine@melaineserver:/srv/WordPress_container_3$ sudo certbot certonly --apache
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Please enter the domain name(s) you would like on your certificate (comma and/or
space separated) (Enter 'c' to cancel): meljuju.duckdns.org
Requesting a certificate for meljuju.duckdns.org

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/meljuju.duckdns.org/fullchain.pem
Key is saved at: /etc/letsencrypt/live/meljuju.duckdns.org/privkey.pem
This certificate expires on 2023-09-15.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the backgr
ound.

-----
If you like Certbot, please consider supporting our work by:
* Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
* Donating to EFF: https://eff.org/donate-le
-----
melaine@melaineserver:/srv/WordPress_container_3$
```

Et voilà notre domaine à été certifié.

Grâce à cette certification on va pouvoir configurer le HTTPS :

Dans le fichier de configuration apache on met ce code :

```
<VirtualHost *:443>
    ServerName meljuju.duckdns.org

    SSLEngine on
    SSLCertificateFile /etc/letsencrypt/live/meljuju.duckdns.org/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/meljuju.duckdns.org/privkey.pem

    # Si vous avez un certificat de chaîne de confiance, incluez également le fichier de
    certificat intermédiaire :
    SSLCertificateChainFile /path/to/your/intermediate.crt

    # Autres directives spécifiques à votre configuration
    ...
</VirtualHost>
```

Et voilà ! Notre serveur est configuré pour le https.

VIII. Reverse proxy

Pour finir, nous allons mettre en place un reverse proxy.

Mettre en place un reverse proxy sur votre serveur Ubuntu va permettre de gérer le routage des requêtes HTTP vers les différents services Docker, offrant une meilleure flexibilité et une meilleure sécurité en centralisant les demandes vers un point d'entrée unique.

Nous avons choisi **Node.js** comme technologie pour notre serveur back-end.

A présent que tout est installé on met en place le fichier backend (server.js) :

```
GNU nano 6.2 server.js *
const express = require('express');
const app = express();

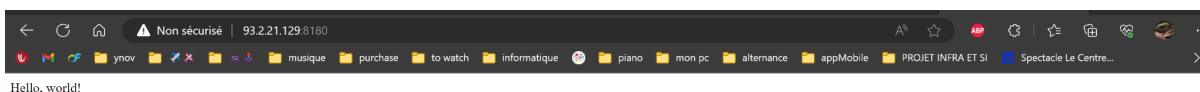
app.get('/', (req, res) => {
  res.send('Hello, world!');
});

const port = 8180;
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

A présent on peut démarrer le server :

```
melaine@melaineserver:/backend$ node server.js
Server is running on port 8180
```

Et hop notre serveur backend est en place sur le port 8180 :

A screenshot of a web browser window. The address bar shows a non-secure connection to 93.2.21.129:8180. The browser's bookmark bar contains various folders like 'ynov', 'musique', 'purchase', etc. The main content area of the browser displays the text 'Hello, world!'.

A présent configurons sur la partie apache dans le fichier de configuration :

```
ProxyPass / http://meljuju.duckdns.org:8180/
ProxyPassReverse / http://meljuju.duckdns.org:8180/
```

On démarre le serveur :

```
melaine@melaineserver:/backend$ node server.js
```

Notre serveur backend Node.js servira à **traiter les requêtes** provenant des clients et à gérer la logique métier de votre application.

En le configurant correctement, Il pourra effectuer des opérations telles que l'accès à une **base de données**, le **traitement des données**, l'**authentification** des utilisateurs, la gestion des sessions, etc. Il joue un **rôle central** dans le fonctionnement de votre application web

De cette manière, Apache agit comme une **interface** entre les utilisateurs et votre serveur backend Node.js. Il reçoit les requêtes, les redirige vers votre serveur backend, puis renvoie les réponses au client.

Voilà un type de fonctionnement que l'on pourra mettre en place avec notre reverse proxy entre notre serveur apache et notre backend.

L'avantage de mettre en place un serveur backend Node.js est sa capacité à gérer de manière efficace les connexions concurrentes et à développer des applications réactives et évolutives.