

Rapport TD Virtualisation et Containerisation

Ryan Korban, Baptiste Delaborde

1. Préparation de l'environnement

1.1 Vérification de l'installation Docker

Vérifiez que Docker et Docker Compose sont installés sur votre machine :

```
Microsoft Windows [version 10.0.22631.5335]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\ryank>docker --version
Docker version 28.1.1, build 4eba377

C:\Users\ryank>docker-compose --version
Docker Compose version v2.35.1-desktop.1
```

Je confirme il est bien installé ! (La réponse très pertinente à cette question vaut normalement 10 points dans le barème il me semble)

2. Partie 2 : Configuration de la base de données MariaDB

Quels sont les risques liés à l'utilisation de mots de passe en dur dans les fichiers de configuration ?

Il y a plusieurs risques à mettre en dur les mots de passe dans les fichiers de configuration. Premièrement, les utilisateurs qui ont accès aux fichiers de configuration pourront voir les mots de passes. De plus avec git ils peuvent facilement fuiter si quelqu'un le push par inadvertance.

Que signifie l'utilisation de '%' dans la création de l'utilisateur MySQL ?

Le « % » est ce qu'on appelle une wildcard en MySQL, ici elle permet à l'utilisateur de se connecter depuis n'importe quelle adresse IP.

3. Exercices pratiques

Exercice 1 : Analyse des logs

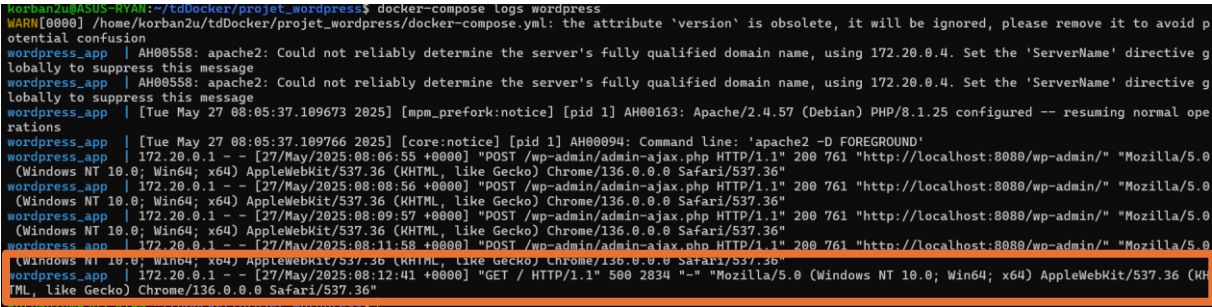
Question : Que se passe-t-il si WordPress démarre avant que MariaDB soit complètement initialisée ?

Il ne se passe rien dans les logs si on n'ouvre pas une page WordPress. Cependant une fois la page ouverte, on a une erreur de connexions. Et si après ça on repart voir les logs, il y a du nouveau. On identifie une erreur 500, ce qui est cohérent vu que la base de données n'est pas active.



Error establishing a database connection

Figure 1 Erreur dans le navigateur lorsque on ouvre WordPress avant MariaDB



```
korban2u@ASUS-RYAN:~/tdocker/projet_wordpress$ docker-compose logs wordpress
WARN[0000] /home/korban2u/tdocker/projet_wordpress/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
wordpress_app | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.20.0.4. Set the 'ServerName' directive globally to suppress this message
wordpress_app | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.20.0.4. Set the 'ServerName' directive globally to suppress this message
wordpress_app | [Tue May 27 08:05:37.109673 2025] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.57 (Debian) PHP/8.1.25 configured -- resuming normal operations
wordpress_app | [Tue May 27 08:05:37.109766 2025] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
wordpress_app | 172.20.0.1 - - [27/May/2025:08:06:55 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 761 "http://localhost:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36"
wordpress_app | 172.20.0.1 - - [27/May/2025:08:08:56 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 761 "http://localhost:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36"
wordpress_app | 172.20.0.1 - - [27/May/2025:08:09:57 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 761 "http://localhost:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36"
wordpress_app | 172.20.0.1 - - [27/May/2025:08:11:58 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 761 "http://localhost:8080/wp-admin/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36"
wordpress_app | 172.20.0.1 - - [27/May/2025:08:12:41 +0000] "GET / HTTP/1.1" 500 2834 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36"
```

Figure 2 Erreur dans la console lorsque on ouvre WordPress avant MariaDB

Exercice 2 : Tests de connectivité

1. **Connectez-vous au conteneur WordPress :**
2. **Testez la connexion à la base de données :**

```
root@9c6fde38992d:/var/www/html# mysql -h db -u wp_user -p wordpress_db
bash: mysql: command not found
root@9c6fde38992d:/var/www/html#
```

Figure 3 Message d'erreur à la suite de la commande

Étant donné que mySQL/mariaDB n'est pas installé dans le conteneur, on ne peut évidemment pas faire cette commande pour l'instant. Pour régler le problème, j'ai ajouté au Dockerfile « mariadb-client » pour pouvoir utiliser la commande mysql sans avoir à installer un serveur mySQL.

```
# Installation des extensions PHP nécessaires
RUN apt-get update && apt-get install -y \
    mariadb-client \
    libzip-dev \
    zip \
    unzip \
    less \
    && docker-php-ext-install zip
```

Figure 4 Ajout de mariadb-client

Une fois cela fait, la connexion est bien établie

On fait de même pour redis :

```
# Installation des extensions PHP nécessaires
RUN apt-get update && apt-get install -y \
    mariadb-client \
    libzip-dev \
    zip \
    unzip \
    less \
    redis-tools \
    && docker-php-ext-install zip
```

Figure 5 Ajout de redis -tools

Ça marche bel est bien :

```
root@efeac684f3c4:/var/www/html# redis-cli -h redis ping
PONG
```

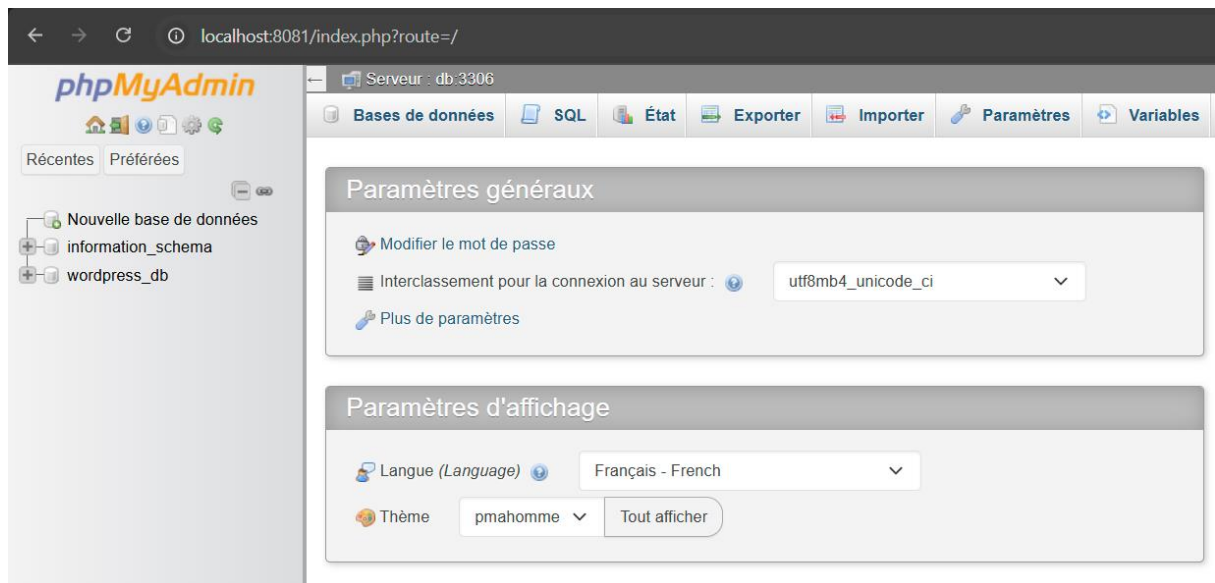


Figure 6 Bon fonctionnement de phpMyAdmin

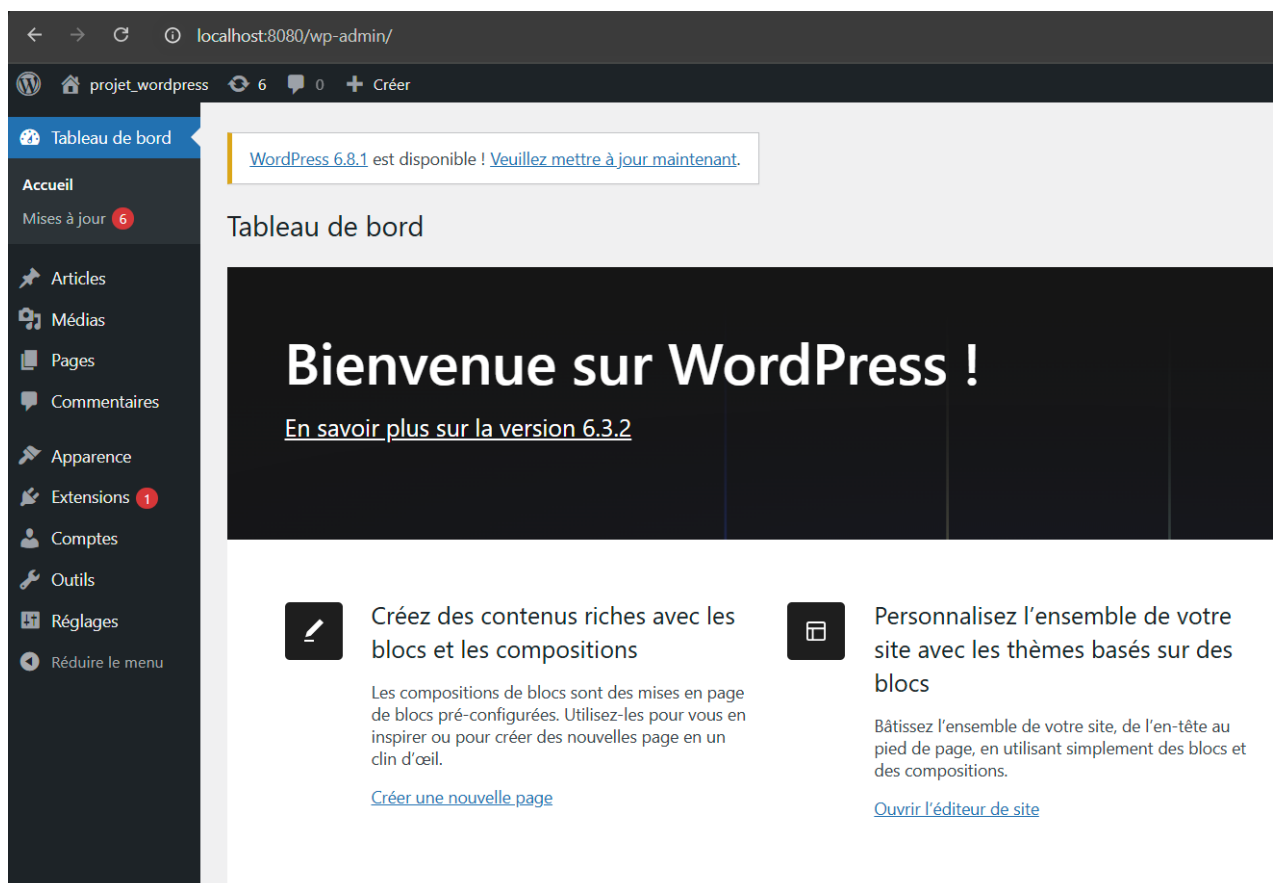


Figure 7 Bon fonctionnement de WordPress

Exercice 3 : Sauvegarde et restauration

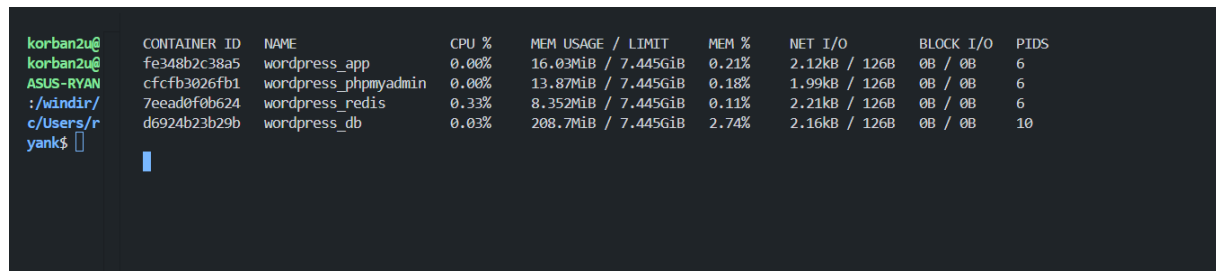
1. Créez une sauvegarde de la base de données :
2. Comment restaureriez-vous cette sauvegarde ?

Avec cette commande :

```
docker-compose exec -T db mysql -u root -p wordpress_db < backup.sql
```

Etant donné que mysqldum n'a pas de commandes pour importer une BD, on utilise la même commande mais avec mysql au lieu de mysqldum et on inverse le > qui sert à exporter en < qui sert à importer

Exercice 4 : Monitoring des ressources



	CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
korban2u@	fe348b2c38a5	wordpress_app	0.00%	16.03MiB / 7.445GiB	0.21%	2.12kB / 126B	0B / 0B	6
korban2u@	cfcfb3026fb1	wordpress_phpmyadmin	0.00%	13.87MiB / 7.445GiB	0.18%	1.99kB / 126B	0B / 0B	6
ASUS-RYAN	7eead0f0b624	wordpress_redis	0.33%	8.352MiB / 7.445GiB	0.11%	2.21kB / 126B	0B / 0B	6
:/windir/	d6924b23b29b	wordpress_db	0.03%	208.7MiB / 7.445GiB	2.74%	2.16kB / 126B	0B / 0B	10
c/Users/r								
yank\$								

Figure 8. Affichage après un docker stats

Quel conteneur consomme le plus de mémoire ?

Le conteneur qui consomme le plus de mémoire est le wordpress_db

Comment pourriez-vous limiter l'utilisation des ressources ?

En utilisant les paramètres « mem_limit » et « cpus » dans le service db :

```
db:
  image: mariadb:10.9
  environment:
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    MYSQL_DATABASE: ${MYSQL_DATABASE}
    MYSQL_USER: ${MYSQL_USER}
    MYSQL_PASSWORD: ${MYSQL_PASSWORD}

  container_name: wordpress_db
  restart: unless-stopped
  volumes:
    - ./volumes/mariadb/var/lib/mysql
    - ./mariadb/init.sql:/docker-entrypoint-initdb.d/init.sql
  networks:
    - wordpress_network
  mem_limit: 512m
  cpus: 0.5
```

Figure 9 Ajout de mem_limit et cpus

4. Questions de réflexion et d'approfondissement

1. Expliquez l'avantage d'utiliser des conteneurs séparés pour chaque service plutôt qu'un seul conteneur avec tous les services.

Ce qui est bien avec le fait d'utiliser des conteneurs séparés, c'est que chaque service (WordPress, MariaDB, Redis) a sa responsabilité. Donc 1 conteneur = une responsabilité unique.

Si un service crash par exemple, les autres eux, continuent de fonctionner (à part si le service dépend d'un autre évidemment, par exemple la BD). Donc ça facilite grandement les mises à jour étant donné qu'elles peuvent être effectuées service par service.

Par exemple on va pouvoir mettre à jour WordPress sans toucher à MariaDB. De plus, cet avantage nous permet aussi de pouvoir réutiliser les conteneurs dans d'autres projets ce qui peut nous éviter de perdre trop de temps à tout configurer au début

2. Quels sont les avantages et inconvénients du stockage des données dans des volumes Docker par rapport au stockage dans le conteneur ?

Avantages des volumes Docker :

Avec le stockage de donnée dans des volumes Docker (le dossier volumes), quand on fait un **docker-compose down** (commande qui arrête et supprime les conteneurs) et que on refait un **docker-compose up**, on récupère toutes les données des conteneurs qui se sont pourtant supprimé (la base de données ou WordPress par exemple). Dans le cas où on n'avait pas de volumes docker, après le **docker-compose up**, le conteneur sera juste neuf, sans les données qu'on avait avant. Tout est stocké dans le dossier /volumes sur notre machine.

ET donc grâce à ce dossier /volumes, il y a un partage des données entre les conteneurs vu qu'ils peuvent accéder aux mêmes données. La sauvegarde des données devient donc beaucoup plus facile car on n'a pas besoin d'extraire les informations de nos conteneurs un par un. Ainsi la sauvegarde des conteneurs elle est indépendante de la sauvegarde des données.

Inconvénients des volumes Docker :

Etant donné que les données sont sur notre machine, on peut facilement y accéder et cela pose des soucis de sécurité, on pourrait facilement ouvrir à un « .env » avec des mots de passe à l'intérieur si on ne sécurise pas bien le dossier.

3. Comment le réseau Docker permet-il la communication entre conteneurs ?

Dans le docker-compose.yml, on a créé un réseau wordpress_network. Et grâce à cela, tous les conteneurs qui sont sur ce réseau peuvent communiquer en utilisant leurs noms. Par exemple, WordPress il peut contacter la base de données en utilisant db:3306 au lieu d'une adresse IP étant donné que docker utilise un DNS interne qui traduit les noms en adresses IP.

Par exemple, WordPress pour communiquer avec MariaDB utilise

« WORDPRESS_DB_HOST: db:3306 »

(Normalement l'info est dans le «.env» mais on avait oublié au moment du screen de remplacer les variables en dur dans le docker-compose par les variables qui font appel au .env)

```
environment:
  WORDPRESS_DB_HOST: db:3306
  WORDPRESS_DB_NAME: wordpress_db
  WORDPRESS_DB_USER: wp_user
  WORDPRESS_DB_PASSWORD: wp_password
```

Et phpMyAdmin pour communiquer avec MariaDB utilise « PMA_HOST: db »

```
environment:
  PMA_HOST: db
  PMA_PORT: 3306
  MYSQL_ROOT_PASSWORD: root_password
```

4. Identifiez trois vulnérabilités potentielles dans cette configuration et proposez des solutions

En premier lieu dans le docker compose les mots de passe qui sont clairement visibles (on les voit notamment sur les images juste au-dessus) : De plus wp_password et root_password qui sont des mots de passe nuls ils n'ont pas de chiffre pas de caractère spéciaux. Donc on pourrait déjà utiliser des mots de passe plus complexe que on stockerait dans le fichier « .env ».

Ensuite, dans la config, on expose les ports 8080 et 8081 à tout le monde, il faudrait plutôt la limiter l'accès avec un firewall ou un reverse proxy.

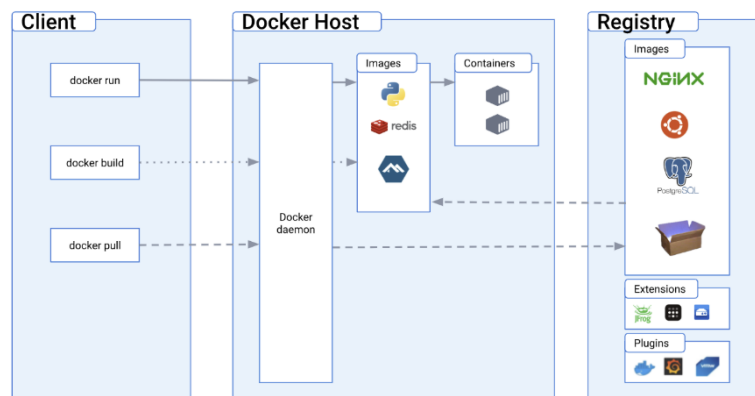
Enfin, les conteneurs sont en root étant donné que on n'a pas créé des utilisateurs. On doit donc en créer avec moins de droits que le root pour pallier à ce problème.

5. Comment pourriez-vous chiffrer les communications entre les conteneurs ?

En faisant des recherches j'ai trouvé que on peut utiliser des certificats SSL/TLS entre les conteneurs. C'est des certificats utilisés pour « chiffrer les flux de données sur les réseaux, qu'ils soient privés ou publics » et on peut apparemment les utiliser pour docker.

Les certificats SSL/TLS sont largement utilisés pour chiffrer les flux de données sur les réseaux, qu'ils soient privés ou publics. Et pour ce qui est des conteneurs Docker, les certificats SSL/TLS garantissent la sécurité des éléments suivants :

- ✔ Communication entre le démon Docker et le client
- ✔ Communication entre les conteneurs Docker
- ✔ Communication entre les hôtes Docker



Architecture de l'hôte Docker. Source de l'image : docs.docker.com

Figure 10 <https://www.globalsign.com/fr/blog/securiser-conteneurs-docker-ssl-tls>

De ce que j'ai compris, docker utilise SSL/TLS pour chiffrer les communications entre les conteneurs, entre le client et le daemon Docker, et pour sécuriser les apps web avec des certificats HTTPS.

De ce que j'ai vu le plus simple à faire c'est utiliser un « Reverse Proxy SSL/TLS ». Pour implémenter ça il faut ajouter un conteneur Nginx/Traefik qui gère le HTTPS.

6. Comment Redis améliore-t-il les performances de WordPress ?

Redis améliore les performances de WordPress grâce au cache. Au lieu que WordPress aille chercher les données dans MariaDB à chaque fois (ce qui est lent), il peut les stocker temporairement dans Redis qui est en RAM donc beaucoup plus rapide. Donc en réduisant le nombre de requêtes, il diminue la charge sur le serveur de base de données.

<https://www.wp-assistance.fr/definition-wordpress/redis/#:~:text=Comment%20Redis%20am%C3%A9liore%2Dt%2DiL,r%C3%A9duisant%20le%20nombre%20de%20requ%C3%AAtes.>

7. Proposez une stratégie pour gérer une montée en charge (scaling horizontal).

Pour gérer une montée en charge on pourrait mettre la base de données sur un serveur plus puissant qui lui pourra gérer la montée en trafic, utiliser le cache de Redis pour que quand il y a une forte demande pour une page, donner la page sauvegardée dans le cache au lieu de faire des requêtes inutiles sur le serveur. On pourrait aussi mettre une liste d'attente d'accès au site pour ne pas qu'il crash par exemple.

8. Quelles modifications apporteriez-vous pour un déploiement en production ?

Alors premièrement on change les mots de passe de test que on a mis par de vrais mots de passe sécurisés et on les déplace dans le fichier « .env » qu'on ne doit surtout pas push sur le git.

Ensuite on ajoute les fameux génère un certificat HTTPS et on met en place le « Reverse Proxy SSL/TLS ».

De plus, dans notre config docker compose, pour phpmyadmin on utilise une version « latest » vu que on n'a pas spécifié de version.

```
> Run Service
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  container_name: wordpress_phpmyadmin
```

Ce qui serait bien c'est de donner plutôt une version stable pour qu'il n'y a pas une version sur le serveur avec de potentielle vulnérabilité.

9. Comment intégreriez-vous cette stack dans un pipeline CI/CD ?

Alors j'ai fait quelques recherches sur le CI/CD parce que je n'avais jamais entendu parler de ça...

D'après ce que j'ai vu, CI/CD ça veut dire "Continuous Integration / Continuous Deployment". En gros, à chaque fois qu'on modifie notre code, ça lance automatiquement des tests et ça déploie si tout se passe bien.

Pour notre stack WordPress ça correspond donc à mettre tous nos fichiers de configuration (docker-compose.yml, Dockerfile, etc.) sur GitHub et qu'à chaque modification qui marche, on commit et push. Pour les tests automatiques, je ne sais pas du tout comment ça marche, j'ai vu que des outils tel que GitubAction existe pour cela, mais je n'ai pas trop compris comment on fait.

5. Défis supplémentaires

Défi 1 : Reverse Proxy

Ajoutez un conteneur Nginx comme reverse proxy devant WordPress.

```
events {
    worker_connections 1024;
}

http {
    upstream wordpress {
        server wordpress:80;
    }

    server {
        listen 80;
        server_name localhost;

        location / {
            proxy_pass http://wordpress;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

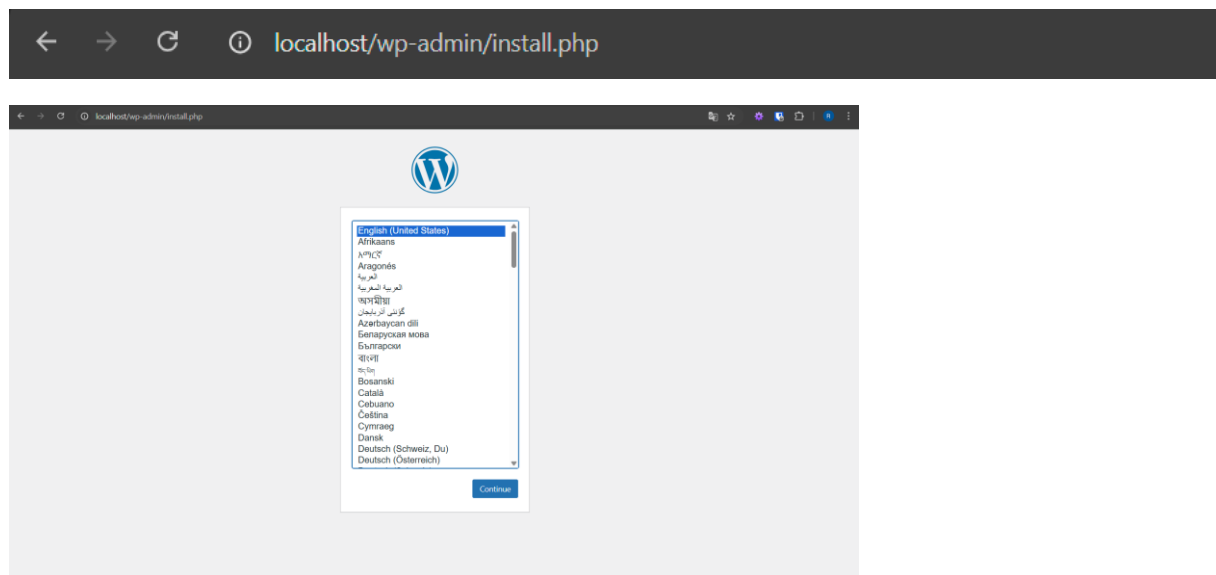
On configure nginx

```
Run Service
nginx:
  image: nginx:alpine
  container_name: wordpress_nginx
  restart: unless-stopped
  ports:
    - "80:80"
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
    - wordpress
  networks:
    - wordpress_network
```

Puis on l'ajoute dans le docker-compose

```
# WordPress
> Run Service
wordpress:
  build: ./wordpress
  container_name: wordpress_app
  restart: unless-stopped
  environment:
    WORDPRESS_DB_HOST: ${WORDPRESS_DB_HOST}
    WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}
    WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
    WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
  volumes:
    - ./volumes/wordpress:/var/www/html
  depends_on:
    - db
    - redis
  networks:
    - wordpress_network
```

On enlève le port dans WordPress



Et ça marche sans port !

Défi 2 : Monitoring

Ajoutez des conteneurs Prometheus et Grafana pour le monitoring.

```

global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node-exporter'
    static_configs:
      - targets: ['node-exporter:9100']

  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']

```

Configuration de Prometheus

```

▷ Run Service
cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: wordpress_cadvisor
  restart: unless-stopped
  ports:
    - "8080:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:ro
    - /sys:/sys:ro
    - /var/lib/docker:/var/lib/docker:ro
    - /dev/disk/:/dev/disk:ro
  privileged: true
  networks:
    - wordpress_network

```

```

▷ Run Service
node-exporter:
  image: prom/node-exporter:latest
  container_name: wordpress_node_exporter
  restart: unless-stopped
  ports:
    - "9100:9100"
  networks:
    - wordpress_network

```

On ajoute des services de monitoring, node-exporter pour la surveillance du système, et cadvisor pour la surveillance des conteneurs

```

> Run Service
prometheus:
  image: prom/prometheus:latest
  container_name: wordpress_prometheus
  restart: unless-stopped
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml:ro
    - ./volumes/prometheus:/prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
    - '--storage.tsdb.path=/prometheus'
  networks:
    - wordpress_network

```

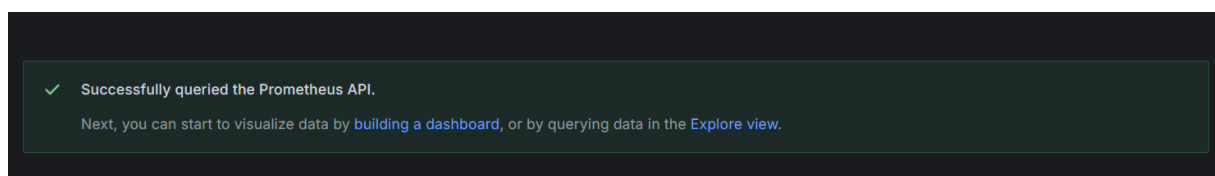
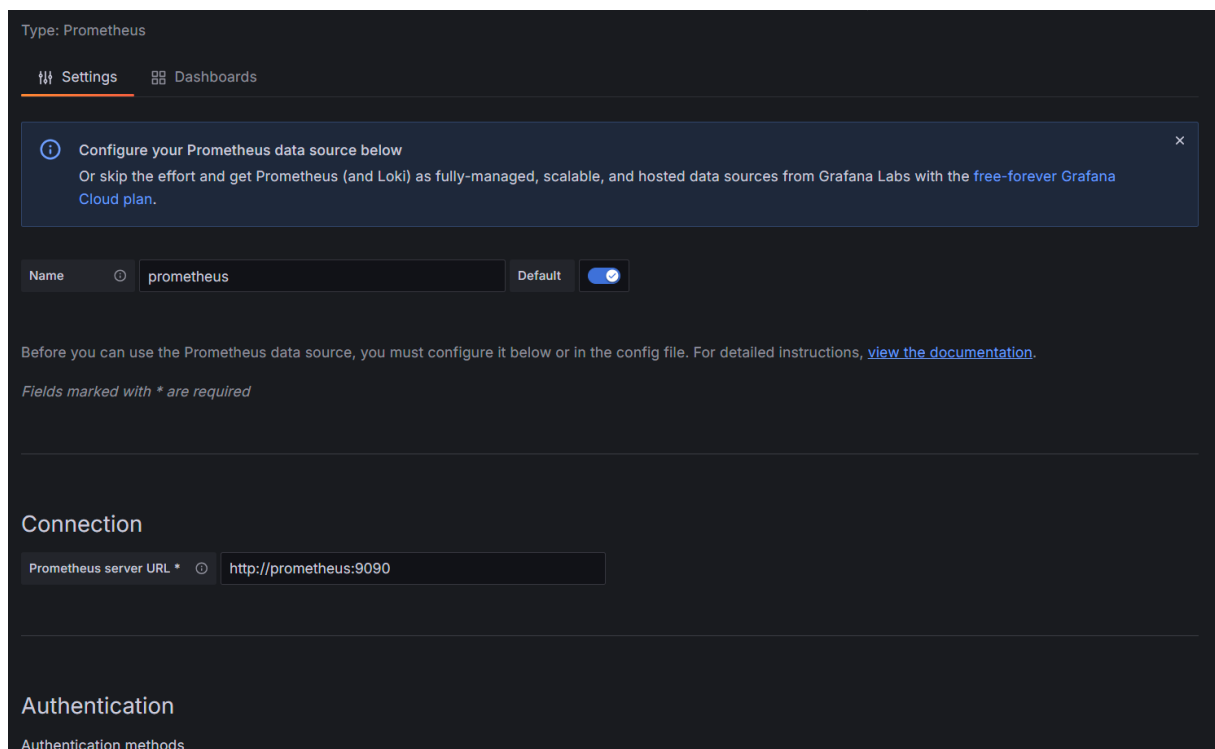
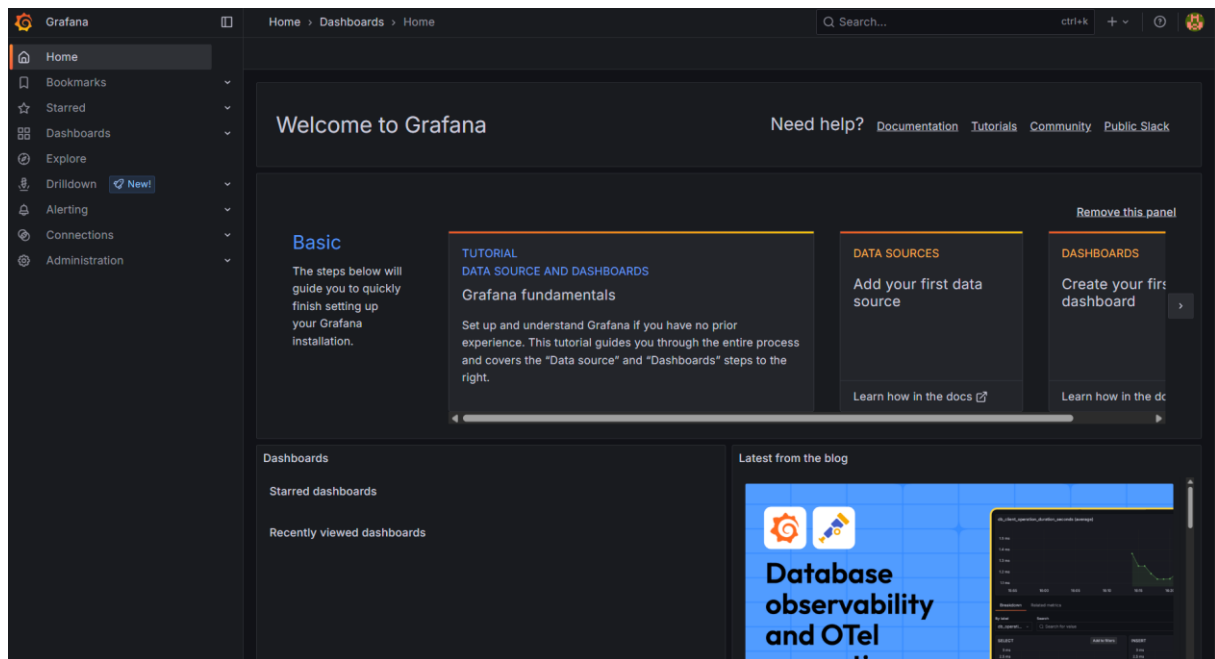
On ajoute prometheus qui est une bd pour les métriques

```

> Run Service
grafana:
  image: grafana/grafana:latest
  container_name: wordpress_grafana
  restart: unless-stopped
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin
  volumes:
    - ./volumes/grafana:/var/lib/grafana
  depends_on:
    - prometheus
  networks:
    - wordpress_network

```


Et grafana qui est une interface graphique à la quelle on va lier prometheus



On lie prometheus à grafana

Import dashboard

Import dashboard from file or Grafana.com



Upload dashboard JSON file

Drag and drop here or click to browse

Accepted file types: json, .txt

Find and import dashboards for common applications at grafana.com/dashboards

Load

Import via dashboard JSON model

```
{
  "title": "Example - Repeating Dictionary variables",
  "uid": "_0HnEoN4z",
  "panels": [...]
  ...
}
```

Load

Cancel

On importe un dashbord qui existe déjà, le « Node Exporter Full » qui affiche des métriques du système tel que le CPU ou la mémoire

Import dashboard

Import dashboard from file or Grafana.com


Importing dashboard from Grafana.com

Published by	phillicious
Updated on	2017-02-17 16:06:00

Options

Name

Folder

 Dashboards


▼

Unique identifier (UID)

The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

Change uid

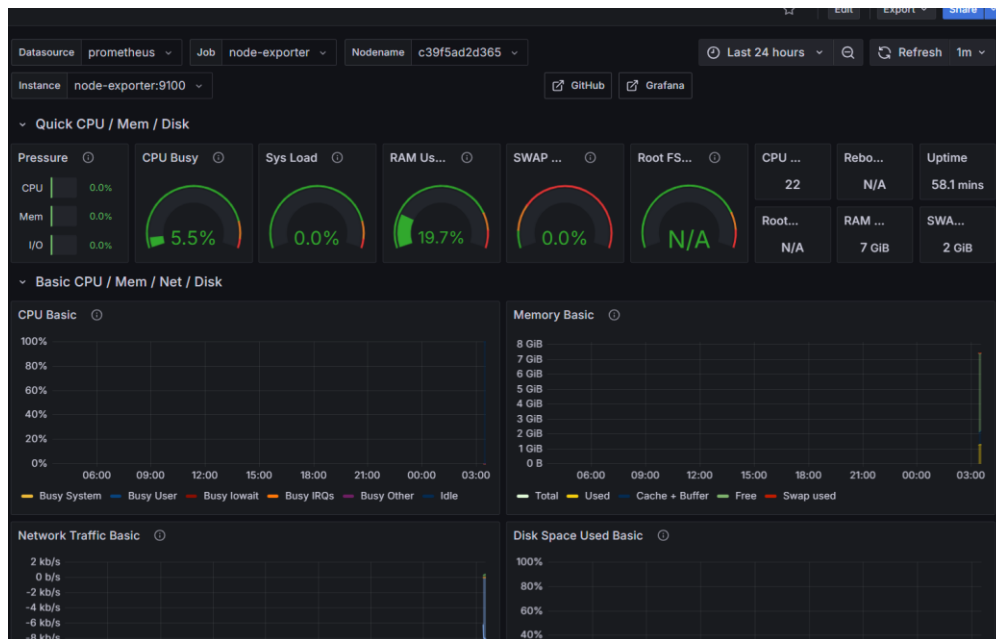
prometheus

 prometheus

▼

Import

Cancel



Ça marche !

Défi 3 : Clustering

Configurez plusieurs instances WordPress avec un load balancer.

```
# WordPress
wordpress1:
  build: ./wordpress
  container_name: wordpress_app1
  restart: unless-stopped
  environment:
    WORDPRESS_DB_HOST: ${WORDPRESS_DB_HOST}
    WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}
    WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
    WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
  volumes:
    - ./volumes/wordpress:/var/www/html
  depends_on:
    - db
    - redis
  networks:
    - wordpress_network

wordpress2:
  build: ./wordpress
  container_name: wordpress_app2
  restart: unless-stopped
  environment:
    WORDPRESS_DB_HOST: ${WORDPRESS_DB_HOST}
    WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}
    WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
    WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
  volumes:
    - ./volumes/wordpress:/var/www/html
  depends_on:
    - db
    - redis
  networks:
    - wordpress_network
```


Ajout de 2 wordpress

```
nginx:
  image: nginx:alpine
  container_name: wordpress_nginx
  restart: unless-stopped
  ports:
    - "80:80"
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
    - wordpress1
    - wordpress2
  networks:
    - wordpress_network
```

Ajout des depends_on vu que on a changer le nom du service wordpress et ajouté un deuxième

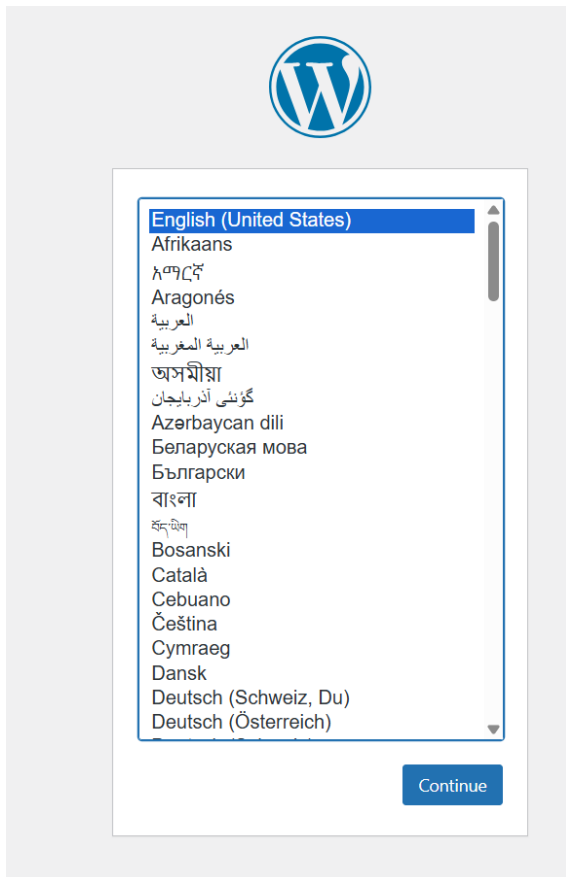
```
=> [wordpress2] resolving provenance for metadata file 0.0s
[+] Running 2/5
✓ wordpress1 Built 0.0s
✓ wordpress2 Built 0.0s
* Container wordpress_redis Creating 0.1s
* Container wordpress_grafana Creating 0.1s
* Container wordpress_db Creating 0.1s
Error response from daemon: Conflict. The container name "/wordpress_db" is already in use by container "a0cbd58caeb02a1d6a8536a1e40f416bdb8329e8395fa1a1426802fa2a9d4921". You have to remove (or rename) that container to be able to reuse that name.
root@ASUS-RYAN:/home/korban2u/tdocker/projet_wordpress# docker rm -f wordpress_db
wordpress_db
```

Problème de cache sûrement, j'ai supprimé wordpress_db et les autres qui posaient problème aussi.

```
Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint wordpress_cadvisor (1696106adb894a4828e0009ccc3843f4dc39e236baba2964a0627849bec037f5): Bind for 0.0.0.0:8080 failed: port is already allocated
root@ASUS-RYAN:/home/korban2u/tdocker/projet_wordpress# docker-compose up -d --build
WARN[0000] /home/korban2u/tdocker/projet_wordpress/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
```

Problème de port déjà utilisé, j'ai remplacé le port. C'est bizarre ça marchait avant sur ce port.

```
▶ Run Service
cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: wordpress_cadvisor
  restart: unless-stopped
  ports:
    - "8082:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:ro
    - /sys:/sys:ro
    - /var/lib/docker:/var/lib/docker:ro
    - /dev/disk:/dev/disk:ro
  privileged: true
  networks:
    - wordpress_network
```



Wordpresse marche

```
✓ Container wordpress_app1 Stopped  
root@ASUS-RYAN: /home/korban2u/tdocker/projet_wordpress#
```

Et il marche toujours si on stoppe le premier wordpress

