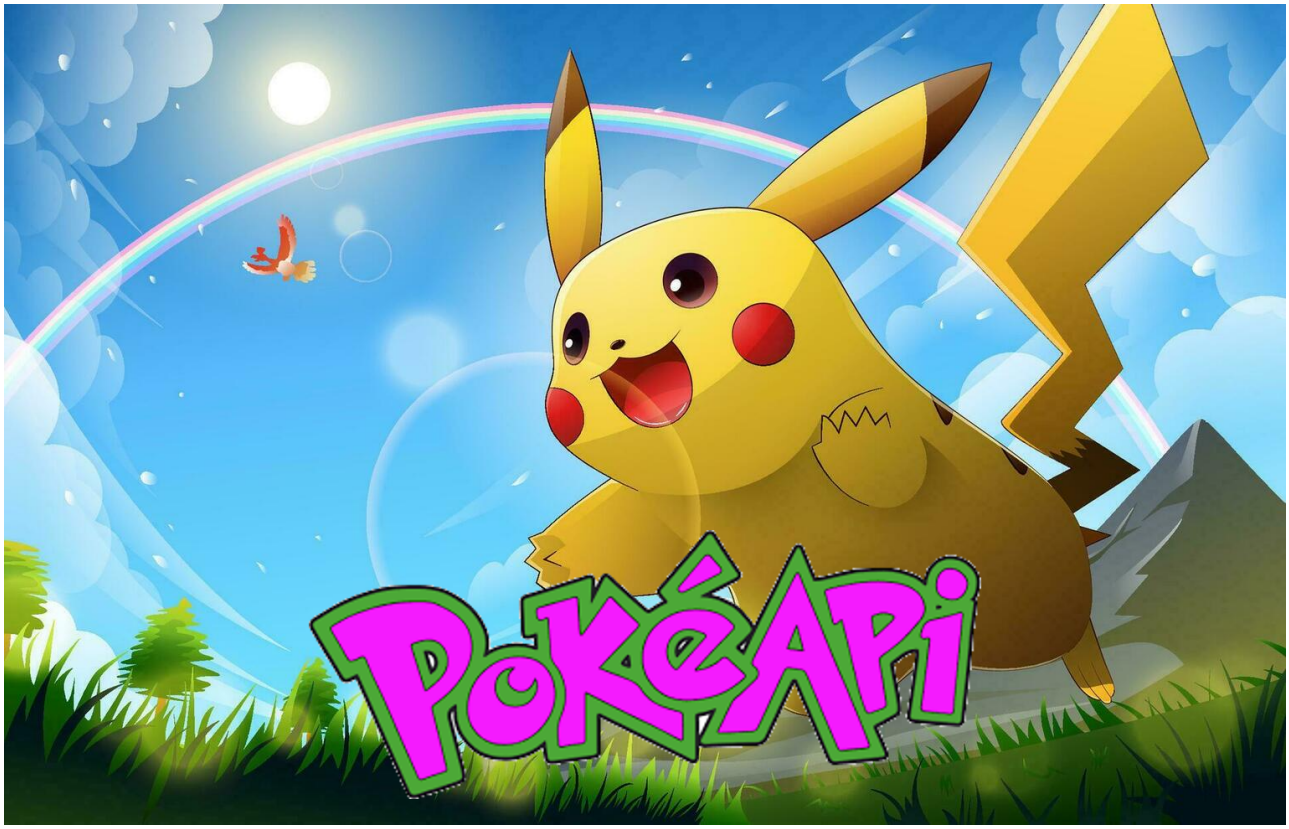

Qu'est ce que Docker ?

Avant toute chose, il est important de savoir que ce programme à définitivement changer notre manière de gérer les infrastructures informatique moderne. **Docker** permet d'automatiser toutes les opérations de déploiement d'une solution. Pour mieux comprendre son utilité, revenons quelques années en arrière. A l'époque ou la virtualisation n'existait pas. A cette époque là chaque service était attaché à une machine physique. C'est ce que l'ont appel des serveurs. Parmi ceux-là, les plus connues sont les serveurs mails, serveurs DNS, serveurs web, serveur base de données et bien d'autres. Aujourd'hui les besoins ont évolués et une seule solution (même petite) peut avoir besoin de 3 ou 4 serveurs pour fonctionner. Une vraie galère à configurer ! Et c'est là qu'intervient **Docker**. Ce que **Docker** permet est simple, il remplace les machines serveurs physique par des **machines virtuels standardisées**. Le système d'exploitation standard choisit est une version **ultra allégée de linux** appelé **alpine**. En une seule commande vous pouvez créer et lancer une machine linux minimale et configurer celle-ci à l'aide d'un seul fichier, le fichier `Dockerfile` !

Installation

1. Dans une console administrateur: `# wsl --install` puis installer Docker Desktop.
2. Les options suivantes doivent être activées `Use the WSL2 based engine` et `WSL Integration Ubuntu`.

Dockerfile API

Le fichier `Dockerfile` permet de décrire ce qu'une image **Docker** contient comme programme, librairies, configurations etc... Une fois le fichier configuré, il est possible de **construire** l'image avec `docker build` et de la lancer avec `docker run`. Le fichier `Dockerfile` ci-dessous est spécialement conçu pour faire fonctionner notre API. Pour cela nous partons d'une image de base `php:8.3.0-apache` qui sera modifier pour arriver à la configuration voulue:

```
# use PHP 8.2
FROM php:8.3-apache

# Install common php extension dependencies
RUN apt-get update && apt-get install -y \
    libfreetype-dev \
    libjpeg62-turbo-dev \
    libpng-dev \
    zlib1g-dev \
    libzip-dev \
    unzip \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install -j$(nproc) gd \
    && docker-php-ext-install zip \
    && docker-php-ext-install mysqli pdo pdo_mysql

# Set the working directory
COPY . /var/www/app
WORKDIR /var/www/app

COPY docker/.env /var/www/app/.env
COPY docker/vhost.conf /etc/apache2/sites-available/000-default.conf

RUN a2enmod rewrite

RUN chown -R www-data:www-data /var/www/app \
    && chmod -R 775 /var/www/app/storage

# install composer
COPY --from=composer:2.6.5 /usr/bin/composer /usr/local/bin/composer

# copy composer.json to workdir & install dependencies
COPY composer.json ./
RUN composer install
```

Un nouveau fichier d'environnements

Ce fichier doit se situé dans `docker/.env`. Copier et adapté votre `.env` à la nouvelle infrastructure:

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:YievxBp47SLZdWp7+HLSXC68NIoDPoioRdhlaBZ2+UM=
APP_DEBUG=true
APP_TIMEZONE=UTC
APP_URL=http://localhost

APP_LOCALE=en
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=en_US

APP_MAINTENANCE_DRIVER=file
# APP_MAINTENANCE_STORE=database

BCRYPT_ROUNDS=12

LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=laravel-mysql_db-1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=root

SESSION_DRIVER=database
SESSION_LIFETIME=120
SESSION_ENCRYPT=false
SESSION_PATH=/
SESSION_DOMAIN=null

BROADCAST_CONNECTION=log
FILESYSTEM_DISK=local
QUEUE_CONNECTION=database

CACHE_STORE=database
CACHE_PREFIX=
```

```
MEMCACHED_HOST=127.0.0.1

REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=log
MAIL_HOST=127.0.0.1
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false

VITE_APP_NAME="${APP_NAME}"

SESSION_DOMAIN=localhost
SANCTUM_STATEFUL_DOMAINS=localhost:8000
```

Le fichier vhost.conf

Ce fichier doit se situé dans `docker/vhost.conf`.

```
<VirtualHost *:80>
    DocumentRoot /var/www/app/public

    <Directory /var/www/app/public>
        AllowOverride all
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Dockerfile SPA

```
FROM node:lts-alpine as build-stage

WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# production stage
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Configuration des conteneurs

Bon c'est super tout ça mais jusqu'ici nous avons qu'une seule image docker, hors comme nous l'avons vu un site web a besoin de minimum 2 serveurs pour fonctionner:

- Un serveur web
- Un serveur de base de donnée

Dans notre cas il nous en faut même trois dû à l'architecture 3-tiers:

- Un serveur web API
- Un serveur web SPA
- Un serveur de base de donnée

Pour cela docker intègre `docker-compose`. Une solution qui permet de configurer le lancement de plusieurs machines virtuels d'un coup. Ici, pour nos besoins nous configurons `docker-compose` pour qu'il lance:

- Un serveur Web API
- Un serveur Web SPA
- Un serveur MySQL
- Un serveur PhpMyAdmin

Cette configuration s'effectue grâce au fichier `docker-compose.yml`. Ce fichier doit donc être situé au dessus de toutes les parties du projet:

```
services:
  laravel-docker:
```

```
    container_name: laravel-docker
    build: ./Cursus-api
    ports:
      - 8000:80

  vue-docker:
    container_name: vue-docker
    build: ./Cursus-spa
    ports:
      - 80:80

  mysql_db:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: laravel
    ports:
      - 3306:3306

  phpmyadmin:
    image: phpmyadmin:latest
    ports:
      - 9001:80
    environment:
      - PMA_ARBITRARY=1
```

Makefile

Voici en bonus un Makefile pour nous simplifier la vie:

```
setup:
        @make build
        @make up
        @make composer-update
build:
        docker-compose build --no-cache --force-rm
stop:
        docker-compose stop
up:
        docker-compose up -d
composer-update:
        docker exec laravel-docker bash -c "composer update"
data:
```

```
        docker exec laravel-docker bash -c "php artisan migrate"
        docker exec laravel-docker bash -c "php artisan db:seed"
```

Plus qu'a lancer le `make setup` et ont à accès à tous nos services comme avant mais cette fois-ci indépendant et pouvant être déployé en une seule commande.

Intégration continue avec Github

L'intégration continue est une technique qui permet de tester un projet tout au long de son développement et cela sans effort supplémentaire à fournir de la part du développeur. Biensurs, une pipeline de tests doit être configurée en amont mais une fois cela fait, les tests s'effectuerons automatiquement après chaque push sur la branche principale. Rassurez-vous cependant, le gros du travail a été réalisé plus haut grâce à Docker. Github va simplement lancer nos conteneurs, migrer la base de données, et enfin exécuter les tests. La configuration est donc finalement assez simple:

```yaml
name: pokedex-app

on: # specify the build to trigger the automated ci/cd
  push:
    branches:
      - "main"
jobs:
  docker:
    timeout-minutes: 10
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: Tests-api
    steps:
    - name: Checkout
      uses: actions/checkout@v1
    - name: Install Docker
      uses: hoverkraft-tech/compose-action@v2.0.1
    - name: Sleep for 10 seconds
      run: sleep 10s
      shell: bash
    - name: Database migrate
      run: docker exec laravel-docker bash -c "php artisan migrate"
    - name: Database seeding
      run: docker exec laravel-docker bash -c "php artisan db:seed"
    - name: Install node
      uses: actions/setup-node@v3
      with:
        node-version: 20.x
```

```
    - name: Install dependencies
      run: npm install
    - name: Run tests
      run: npm run test
```