Tutoriel : Déploiement d'une application Symfony 7.3 sur Azure avec CI/CD

Objectifs

- Créer une VM Debian 13 sur Azure (< 8€/mois)
- Installer et configurer PHP 8.4, MariaDB 12.1.2, NGINX
- Configurer un self-hosted runner GitHub Actions sur la VM
- Automatiser le déploiement de l'application Symfony

---

1. Création de la machine virtuelle Debian 13 sur Azure

Prérequis

- Un compte Azure (gratuit pour les étudiants via Azure for Students)
- Une carte bancaire pour la vérification (non débitée si vous restez dans la limite gratuite)

Étapes

1.1. Connexion au portail Azure

- Rendez-vous sur portal.azure.com et connectez-vous.

1.2. Création de la VM

- Cliquez sur **"Créer une ressource"** > **"Machine virtuelle"**.
- Remplissez les informations :
  - **Abonnement** : Sélectionnez votre abonnement (gratuit si étudiant).
  - **Groupe de ressources** : Créez-en un nouveau (ex: `BTS-SIO-Symfony`).
  - **Nom de la machine virtuelle** : `symfony-vm`.
  - **Région** : Choisissez `France Centre` ou `Europe Ouest`.
  - **Options de disponibilité** : `Aucune infrastructure redondante requise`.
  - **Image** : Sélectionnez `Debian 13`.
  - **Taille** : Choisissez `B2ats-v2` (1 vCPU, 1 Go RAM, ~7€/mois).
  - **Type d'authentification** : `Clé publique SSH` (recommandé) ou `Mot de passe`.
  - **Nom d'utilisateur** : `azureuser`.
  - **Ports d'entrée publics** : Autorisez `SSH (22)` et `HTTP (80)`.
- Dans la sections "Disques":
  - **Sélectionner**: `SSD Standard`
- Dans la section "Mise en réseau":
  - **Sélectionner**: `Supprimer l'adresse IP publique et la carte réseau lors de la suppression de la machine virtuelle`

- Cliquez sur **"Vérifier + créer"**, puis **"Créer"**.

1.3. Connexion à la VM

- Une fois la VM créée, notez son **adresse IP publique**.
- Connectez-vous en SSH :

```
ssh ssh -i <nom de la clé SSH>.pem azureuser@<adresse-ip-publique>
```

---

2. Installation et configuration de l'environnement

2.1. Mise à jour du système

```
sudo apt update && sudo apt upgrade -y
```

2.2. Installation de PHP 8.4

Pour PHP 8.4

```
sudo apt update
sudo apt install -y lsb-release ca-certificates apt-transport-https curl
gnupg
curl -sSL https://packages.sury.org/php/README.txt | sudo bash -x
echo "deb https://packages.sury.org/php/ $(lsb_release -sc) main" | sudo
tee /etc/apt/sources.list.d/sury-php.list
sudo apt update
sudo apt install -y php8.4 php8.4-cli php8.4-fpm php8.4-mysql php8.4-xml
php8.4-curl php8.4-mbstring php8.4-intl php8.4-zip
```

- Vérifiez la version :

```
php -v
```

2.2.1 Installation de Composer

```
curl -sS https://getcomposer.org/installer -o composer-setup.php
HASH=`curl -sS https://composer.github.io/installer.sig`
php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo
'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-
setup.php'); } echo PHP_EOL;"
```

```
sudo php composer-setup.php --install-dir=/usr/local/bin --
filename=composer
```

2.3. Installation de MariaDB 12.1.2

- Ajoutez le dépôt officiel de MariaDB :

```
sudo curl -o /usr/share/keyrings/mariadb-keyring.gpg
https://mariadb.org/mariadb_release_signing_key.asc
echo "deb [signed-by=/usr/share/keyrings/mariadb-keyring.gpg
arch=amd64] https://mirror.netcologne.de/mariadb/repo/12.1/debian
trixie main" | sudo tee /etc/apt/sources.list.d/mariadb.list
sudo apt update
```

- Installez MariaDB :

```
sudo apt install -y mariadb-server mariadb-client
```

- Sécurisez l'installation :

```
sudo mysql_secure_installation
```

  - Suivez les instructions pour définir un mot de passe root et supprimer les utilisateurs anonymes.

- Créez une base de données pour Symfony :

```
sudo mysql -u root -p
```

```
CREATE DATABASE symfony_db;
CREATE USER 'symfony_user'@'localhost' IDENTIFIED BY
'votre_mot_de_passe';
GRANT ALL PRIVILEGES ON symfony_db.* TO 'symfony_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

2.4. Installation de NGINX

- Installez NGINX :

```

```

```
sudo apt install -y nginx
```

- Configurez un hôte virtuel pour Symfony :

```
sudo nano /etc/nginx/sites-available/symfony.conf
```

  - Collez la configuration suivante (adaptez le `server_name` et le `root`) :

```
server {
    server_name symfony.francecentral.cloudapp.azure.com;
    root /var/www/symfony/public;

    location / {
        # try to serve file directly, fallback to index.php
        try_files $uri /index.php$is_args$args;
    }

    # optionally disable falling back to PHP script for the asset
directories;
    # nginx will return a 404 error when files are not found
instead of passing the
    # request to Symfony (improves performance but Symfony's 404
page is not displayed)
    # location /bundles {
    #     try_files $uri =404;
    # }

    location ~ ^/index\.php(/|$) {
        fastcgi_pass unix:/var/run/php/php-fpm.sock;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;

        # optionally set the value of the environment variables
used in the application
        # fastcgi_param APP_ENV prod;
        # fastcgi_param APP_SECRET <app-secret-id>;
        # fastcgi_param DATABASE_URL
"mysql://db_user:db_pass@host:3306/db_name";

        # When you are using symlinks to link the document root
to the
        # current version of your application, you should pass
the real
        # application path instead of the path to the symlink to
PHP
        # FPM.
        # Otherwise, PHP's OPcache may not properly detect
changes to
        # your PHP files (see
```

```
https://github.com/zendtech/ZendOptimizerPlus/issues/126
        # for more information).
        # Caveat: When PHP-FPM is hosted on a different machine
from nginx
        #         $realpath_root may not resolve as you expect!
In this case try using
        #         $document_root instead.
        fastcgi_param SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
        fastcgi_param DOCUMENT_ROOT $realpath_root;
        # Prevents URIs that include the front controller. This
will 404:
        # http://domain.tld/index.php/some-path
        # Remove the internal directive to allow URIs like this
    }

    # return 404 for all other php files not matching the front
controller
    # this prevents access to other php files you don't want to
be accessible.
    location ~ \.php$ {
        return 404;
    }

    error_log /var/log/nginx/symfony_error.log;
    access_log /var/log/nginx/symfony_access.log;
}
```

- Activez la configuration :

```
sudo ln -s /etc/nginx/sites-available/symfony.conf /etc/nginx/sites-
enabled/
sudo nginx -t
sudo systemctl restart nginx
```

- Paramétrage des droits sur NGINX

```
sudo usermod -a -G www-data azureuser
sudo chown -R www-data:www-data /var/www/
sudo chmod 2775 /var/www/ && find /var/www/ -type d -exec sudo chmod 2775
{} \;
find /var/www/ -type f -exec sudo chmod 0664 {} \;
```

2.5. Clonage de l'application Symfony

- Installez Git :

```
sudo apt install -y git
```

- Authentication Github via SSH Pourn comprendre les commandes données, voir tuto: https://mgimond.github.io/Colby-summer-git-workshop-2021/authenticating-with-github.html#:~:text=Saving%20tokens%20in%20Linux,-To%20temporarily%20cache&text=The%20next%20time%20you%20are%20prompted%20for%20your%20GitHub%20user,this%20file%20is%20not%20encrypted.

```
ssh-keygen -t ed25519 -C "email de votre compte Github"
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
#mettre la clé SSH dans profil Github
cat ~/.ssh/id_ed25519.pub
#test de connexion
ssh -T git@github.com
```

- Clonez le dépôt de l'application (remplacez par l'URL de votre dépôt) :

```
cd /var/www
git clone git@github.com:votre compte Github/votre-depot-symfony.git
symfony
```

- Installez les dépendances avec Composer :

```
composer install
```

- Configurez les variables d'environnement (`.env`) :

```
nano .env.local
```

  - Mettez à jour `DATABASE_URL` avec les informations de MariaDB et votre environnement en mode production afin de ne pas laisser d'info sensible apparaitre sur internet :

```
APP_ENV=prod
DATABASE_URL="mysql://symfony_user:votre_mot_de_passe@localhost:3306/symfony_db"
```

- Application vos migrations pour établir le schéma de base de données :

```
php bin/console doctrine:migrations:migrate
```

---

3. Configuration du self-hosted runner GitHub Actions

3.1. Préparation de la VM

- Installez les dépendances nécessaires :

```
sudo apt install -y curl jq
```

3.2. Création du runner

- Sur GitHub, allez dans votre dépôt > **Settings** > **Actions** > **Runners** > **New self-hosted runner**.
- Suivez les instructions pour Linux (Debian).
- Téléchargez et configurez le runner :

```
mkdir actions-runner && cd actions-runner
curl -o actions-runner-linux-x64-2.311.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.311.0/actions-
runner-linux-x64-2.311.0.tar.gz
tar xzf ./actions-runner-linux-x64-2.311.0.tar.gz
./config.sh --url https://github.com/votre-utilisateur/votre-depot-
symfony --token VOTRE_TOKEN_RUNNER
```

- Lancez le runner en arrière-plan :

```
sudo ./svc.sh install
sudo ./svc.sh start
```

3.3. Configuration du workflow GitHub Actions

- Créez un fichier `.github/workflows/deploy.yml` dans votre dépôt :

```
name: Déploiement Symfony

on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: self-hosted
```

```
    steps:
    - uses: actions/checkout@v4

    - name: Installer les dépendances
      run: |
        cd /var/www/symfony
        composer install --no-dev --optimize-autoloader

    - name: Mettre à jour la base de données
      run: |
        cd /var/www/symfony
        php bin/console doctrine:migrations:migrate --no-interaction
        php bin/console cache:clear

    - name: Redémarrer NGINX et PHP-FPM
      run: |
        sudo systemctl restart nginx
        sudo systemctl restart php8.4-fpm
```

---

4. Vérification et tests

4.1. Accéder à l'application

- Ouvrez un navigateur et allez sur `http://<adresse-ip-publique>`.
- Vous devriez voir votre application Symfony.

4.2. Tester le pipeline CI/CD

- Faites un `git push` sur la branche `main`.
- Allez dans l'onglet **Actions** de votre dépôt GitHub pour suivre le déploiement.

---

6. Ressources supplémentaires

- GitHub Actions : Self-hosted runners
