Server-side
Web Development

Unit 13c. Symfony deployment. Guided example.

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2024-25

Index

In this guided example we are going to deploy the Contacts app.

1 Setting up the server

There are several ways to deploy a Symfony app:

1. Dedicated server: a full server with the only purpose of hosting our app. The server can be a physical server (very expensive) or a virtual server (AWS, Azure, OVH, etc).
2. PaaS (platform as a service): using some of the PaaS services available: AWS Elastic Beanstalk, Azure App Service, etc.
3. In Docker containers, that actually run in virtual servers and PaaS services.

We will take the first approach and deploy our app in a Dedicated server in a virtual machine, because this way you can follow all the process more easily.

1.1 Installing the required dependencies

Our server operating system is a Ubuntu server 22.10, but you can use the operative system and server of your choice (IIS, Nginx, etc). Do not deploy the app to a XAMPP server because it's not used in real environments.

Let's install all the required packets:

Apache and PHP:

```
sudo apt update
sudo apt install apache2
sudo apt install php
```

PHP modules:

```
sudo apt install php8.1-cli php8.1-common php8.1-mysql php8.1-zip php8.1-gd
↪  php8.1-mbstring php8.1-curl php8.1-xml php8.1-bcmath php8.1-cgi
```

Check the status of the installed software:

```
sudo service apache2 status
php -v
```

MySQL:

```
sudo apt install mysql-server
```

Composer:

Follow the Composer installation guide: https://getcomposer.org/download/

Check the installation:

```
composer -V
```

Symfony CLI:

```
wget https://get.symfony.com/cli/installer -O - | bash
sudo mv .symfony5/bin/symfony /usr/local/bin/symfony
```

Check the installation:

```
symfony -V
```

Check the Symfony requirements and install the not installed ones (in this example, the `intl` extension):

```
symfony check:requirements
sudo apt-get install php-intl
```

After installing the requirements, run again `symfony check:requirements` until all the dependencies are satisfied.

Git:

If you plan to upload the app using another way, like scp or rsync, you don't need to do this step.

```
sudo apt install git
```

Configure the access to your GitHub account following the guide.

1.2 Configuring the services

Apache:

Enable the `rewrite` module:

```
sudo a2enmod rewrite
sudo service apache2 restart
```

MySQL:

First, create an admin user to manage the entire DBMS:

```
sudo mysql
```

```
mysql> create user 'admin'@'localhost' identified by 'yourpassword';
mysql> grant all privileges on *.* to 'admin'@'localhost' with grant
 ↪  option;
```

Then, create the database and the user for the Symfony app. Here you can use the same connection string of your local server (user, password and database name) or change it:

```
mysql> CREATE USER 'contactsymfony'@'localhost' IDENTIFIED BY
 ↪  'yourpassword';
mysql> CREATE DATABASE `contactsymfony`;
mysql> GRANT ALL PRIVILEGES ON `contactsymfony`.* TO
 ↪  'contactsymfony'@'localhost';
```

2 Deploying the App

2.1 Upload the code

Use git for downloading your repository code from GitHub (or from another git repo):

```
git clone your_repo_url
sudo mv contacts_symfony /var/www/html
```

Upload or write your `.env.local` file with the database connection string and other variables:

On your local computer:

```
cd your_contacts_app_folder
scp .env.local your_server_user@your_server_url_or_ip:
```

On the server:

```
sudo mv ~/.env.local /var/www/html/contacts_symfony/
```

2.2 Installing vendor dependencies

```
cd /var/www/html/contacts_symfony/
export APP_ENV=prod
composer install --no-dev --optimize-autoloader
```

The `--optimize-autoloader` flag improves Composer's autoloader performance by building a "class map". The `--no-dev` flag ensures that development packages are not installed in the production environment.

> If you get a **"class not found"** error during this step, you may need to run `export APP_ENV=prod` before running this command so that the `post-install-cmd` scripts run in the prod environment.

If everything is ok, clear your Symfony cache:

```
APP_ENV=prod APP_DEBUG=0 php bin/console cache:clear
```

2.3 Checking your env variables

Edit your `.env` and `.env.local` to change or update your variables.

In `.env`, change the `APP_ENV` variable from `dev` (development) to `prod` (production):

```
APP_ENV=prod
```

2.4 Migrate your database

Execute the Doctrine migration command:

```
symfony console doctrine:migration:migrate
```

Now, you can insert your database data.

2.5 Generating the JWT keys

We need to generate again the JWT public and private keys because they are not uploaded to our git repository:

```
symfony console lexik:jwt:generate-keypair
```

2.6 Configuring the Apache virtual server

In your app folder (typically `/var/www/html/your_app_name`) run:

```
cd /var/www/html/your_app_name
composer require symfony/apache-pack
```

This pack installs a `.htaccess` file in the `public/` directory that contains the rewrite rules needed to serve the Symfony application.

Copy the `.htaccess` contents inside the `<Directory>` configuration associated to the Symfony application `public/` directory, and replace `AllowOverride All` by `AllowOverride None`:

```
<VirtualHost *:80>
    # ...
    DocumentRoot /var/www/your_project/public

    <Directory /var/www/your_project/public>
        AllowOverride None

        # Copy .htaccess contents here
    </Directory>
</VirtualHost>
```

```
cat public/.htaccess # copy the content
sudo nano /etc/apache2/sites-available/000-default.conf
# Change DocumentRoot, <Directory> and AllowOverride
# and paste the content inside <Directory>
sudo service apache2 restart
```

Done! Now it's time to check that everything is ok, including the API.

But don't forget that your app sends and receives credentials in plain text via the Internet, so your next step should be to create an HTTPS secure server and to redirect all the requests sent to the 80 port to the 443 port.