

# e-application - 2019

## 3. Symphony 4

**Guillaume Kulakowski**

Architecte Technique API Management @CGI

**Loïc Cariou**

Architecte Technique en solutions open-source @CGI



<https://twitter.com/llaumgui>



<https://www.linkedin.com/in/guillaumekulakowski>  
<https://www.linkedin.com/in/loïc-cariou-93b8418a>



[guillaume@kulakowski.fr](mailto:guillaume@kulakowski.fr)  
[cariou.loic.um2@gmail.com](mailto:cariou.loic.um2@gmail.com)



<https://blog.kulakowski.fr>



UNIVERSITÉ  
DE MONTPELLIER

# Agenda

1. Configuration
2. L'ORM de Sf4 : Doctrine
3. TP n°4
4. Le Service Container
5. La traduction dans Sf4
6. Déployer une application Sf4 dans le cloud
7. TP n°5
8. Installation d'un bundle tiers
9. TP n°6

# 1 – Configuration

- Les fichiers
- Créer ses entités
- parameters

# Les fichiers de configurations

- Comme évoqué précédemment il y a plusieurs façon d'écrire des fichiers de configurations :
  - **YAML,**
  - XML,
  - PHP,
  - Pas de fichier (**@notation**).
- Les fichiers de configurations se chargent de la sorte :
  - *Les fichier du dossier config/package*
  - *Eux-mêmes surcharger par les fichier du dossier config/package/{env}*

```
1 # app/config/config.yml
2 imports:
3   - { resource: parameters.yml }
4   - { resource: security.yml }
5   - { resource: services.yml }
6
7 framework:
8   secret:          '%secret%'
9   router:          { resource: '%kernel.root_dir%/config/routing.yml' }
10  # ...
11
12 # Twig Configuration
13 twig:
14   debug:           '%kernel.debug%'
15   strict_variables: '%kernel.debug%'
16
17 # ...
```

## La clé « *parameters* »

- Chaque bundle peut apporter son espace de nom ~ clé.
- Il existe un espace de nom spécial appelé « *parameters* » *il est défini dans le fichier config/service.yaml.*
- Il est utilisé pour définir des variables qui ne sont pas référencées dans d'autres fichiers de configurations.
- Ces variables sont récupérables par la suite.

```
1 # app/config/config.yml
2 # ...
3
4 parameters:
5     locale: en
6
7 framework:
8     # ...
9
10    # any string surrounded by {{ and }} is automatically value
11    default_locale: "%locale%"
12
13 # ...
```

« local » défini dans *parameters*.

« local » exploité ailleurs.



# Le fichier «.env»

- **Fichier ultra sensible !**
- Il ne doit **pas être versionné**.
- Il permet de simuler des variables d'environnements serveur
- Il contient toute les données sensibles (secret, accès base de données, etc.).
- En prod on utilise de vraies variables d'environnement

On exploite ailleurs.

On défini dans .env

```
framework:  
  secret: '%env(APP_SECRET) %'  
  #csrf_protection: true  
  #http_method_override: true
```

```
###> symfony/framework-bundle ###  
APP_ENV=dev  
APP_SECRET=905190bdd026685ce49ac36358b59192  
#TRUSTED_PROXIES=127.0.0.1,127.0.0.2  
#TRUSTED_HOSTS='^localhost|example\.com$'  
###< symfony/framework-bundle ###
```

## Aller plus loin : créer ça configuration

- Symfony permet de créer un espace de nom dédié à son bundle, pouvant utiliser un fichier spécifique.
- Il permet également de contrôler les données renseignées : nature, absence d'informations requises, valeurs par défauts, etc...
- Je vous invite à découvrir tout cela dans la doc.

Aller plus loin : [How to Create Friendly Configuration for a Bundle.](#)

# Débuguer

Une ligne de commande permet de récupérer tout les configurations disponibles :

```
php bin/console config:dump-reference [extension alias]
```

```
llaumgui@stargazer ~/public_html/gro_blog> php bin/console config:dump-reference
```

Available registered bundles with their extension alias if available

```
=====
```

Bundle name	Extension alias
DebugBundle	debug
DoctrineBundle	doctrine
FrameworkBundle	framework
GrdBlogBundle	
MonologBundle	monolog
SecurityBundle	security
SensioDistributionBundle	sensio_distribution
SensioFrameworkExtraBundle	sensio_framework_extra
SensioGeneratorBundle	
SwiftmailerBundle	swiftmailer
TwigBundle	twig
WebProfilerBundle	web_profiler

```
=====
```

// Provide the name of a bundle as the first argument of this command to dump its default configuration. (e.g.  
// config:dump-reference FrameworkBundle)

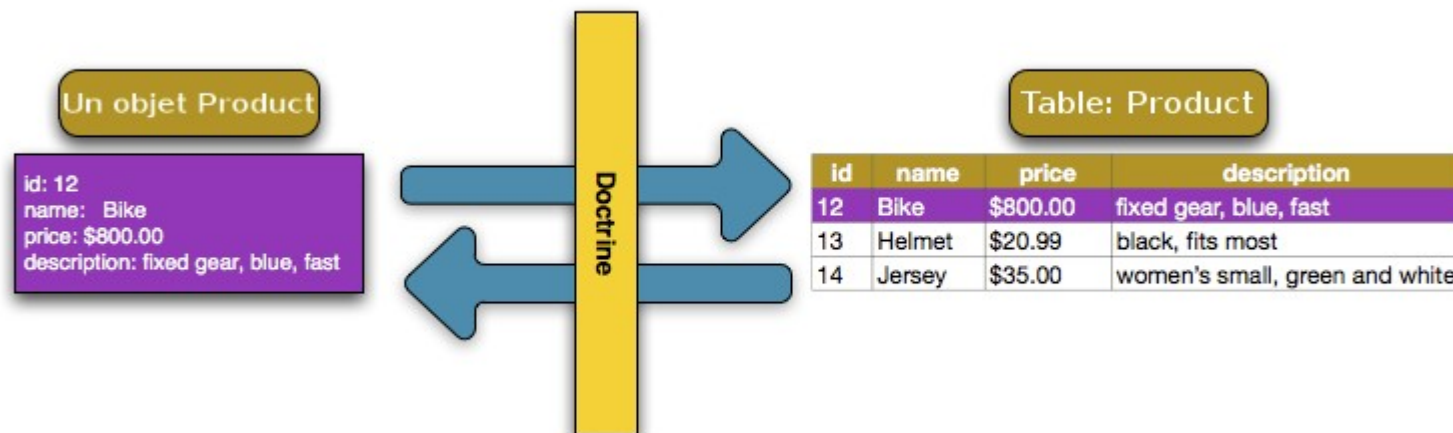


## 2 - L'ORM de Sf4 : Doctrine

- Le concept d'ORM
- Configuration de Doctrine
- L'entité
- Utiliser Doctrine
- Aller plus loin

## Qu'est ce que l'ORM

Un mapping objet-relationnel (en anglais object-relational mapping ou ORaM) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. On pourrait le désigner par « correspondance entre monde objet et monde relationnel » (© Wikipedia).



## Doctrine & ...

- A partir de Sf4, Symfony n'est compatible qu'avec 1 seul ORM : [Doctrine](#).
- [Propel](#) était l'ORM historique de Symfony mais il a laissé sa place à Doctrine, ORM par défaut de Symfony depuis la 1.2 Puis seul ORM de Sf4.

## Configuration de la base de données

- Jusqu'à présent nous avons installé Sf4 sans nous préoccuper de la persistance des données au sein d'une base de données.
- Tout d'abord, le prérequis pour travailler avec Doctrine est d'être sur une base de données utf8\_general\_ci.
- La configuration de l'accès à la base de données se fait via le fichier .env et le fichier config/package/doctrine.yaml :

# Configuration de la DB sqlite

- Pour faciliter nos dev et le portabilité de notre application, nous pouvons utiliser la base de données fichier [SQLite](#).
- Pour manipuler une base SQLite, on peut utiliser l'extension Firefox [SQLite Manager](#).
- Pour cela changer la variable DATABASE\_URL du fichier .env :
  - DATABASE\_URL=sqlite:///kernel.project\_dir%/var/blog.db
- Une fois configurer, créez votre base avec :  
`php bin/console doctrine:database:create`



## Création d'une classe entités 1/3

- La première étape est de déclarer une classe entité au sein de son application dans un espace de nom se terminant par Entity :

```
1 // src/Acme/StoreBundle/Entity/Product.php
2 namespace Acme\StoreBundle\Entity;
3
4 class Product
5 {
6     protected $name;
7
8     protected $price;
9
10    protected $description;
11 }
```

## Création d'une classe entités 2/3

- Ensuite on rajoute les information de mapping. Il existe plusieurs méthode pour cela (YAML, XML) mais ce que je vous conseil est d'utiliser les annotations
- La listes des types de champs est disponible [ici](#).

```
1  // src/Acme/StoreBundle/Entity/Product.php
2  namespace Acme\StoreBundle\Entity;
3
4  use Doctrine\ORM\Mapping as ORM;
5
6  /**
7   * @ORM\Entity
8   * @ORM\Table(name="product")
9   */
10 class Product
11 {
12     /**
13      * @ORM\Id
14      * @ORM\Column(type="integer")
15      * @ORM\GeneratedValue(strategy="AUTO")
16      */
17     protected $id;
18
19     /**
20      * @ORM\Column(type="string", length=100)
21      */
22     protected $name;
23
24     /**
25      * @ORM\Column(type="decimal", scale=2)
26      */
27     protected $price;
28
29     /**
30      * @ORM\Column(type="text")
31      */
32     protected $description;
33 }
```

## Création d'une classe entités 3/3

- Par défaut les attributs sont privés, il vous faudra donc des getter et des setters pour les manipuler. Rien de plus simple : pour générer vos getter et vos setter, c'est aussi simple

```
php bin/console make:entity App/Entity/Product
```

- Le déploiement de la base se fait alors simplement avec bin/console :

```
php bin/console doctrine:schema:update --force
```

- Les plus sportifs ou les plus fainéants pourront réaliser l'opération en 1 ligne de commande :

```
php bin/console make:entity \
--entity="AcmeStoreBundle:Product" \
--fields="name:string(255) price:float
description:text
```

# Manipuler des entités

- Créer ou manipuler des objets est relativement intuitif :

```
1 // src/Acme/StoreBundle/Controller/DefaultController.php
2
3 // ...
4 use Acme\StoreBundle\Entity\Product;
5 use Symfony\Component\HttpFoundation\Response;
6
7 public function createAction()
8 {
9     $product = new Product();
10    $product->setName('A Foo Bar');
11    $product->setPrice('19.99');
12    $product->setDescription('Lorem ipsum dolor');
13
14    $em = $this->getDoctrine()->getManager();
15    $em->persist($product);
16    $em->flush();
17
18    return new Response('Id du produit créé : '.$product->getId());
19 }
```

## Récupérer des objets via Doctrine

- La récupération d'un objet passe par la récupération d'un repository :

```
1 public function showAction($id)
2 {
3     $product = $this->getDoctrine()
4         ->getRepository('AcmeStoreBundle:Product')
5         ->find($id);
6
7     if (!$product) {
8         throw $this->createNotFoundException(
9             'Aucun produit trouvé pour cet id : '.$id
10        );
11    }
12
13    // ... faire quelque chose comme envoyer l'objet $product à un template
14 }
```



## Aller plus loin avec le DQL

- Bien sûr, Doctrine vous permet également d'écrire des requêtes plus complexes en utilisant le Doctrine Query Language (**DQL**). Le DQL est très ressemblant au SQL excepté que vous devez imaginer que vous requêtez un ou plusieurs objets d'une classe d'entité (ex: Product) au lieu de requêter des lignes dans une table (ex: product).

```
1 $em = $this->getDoctrine()->getEntityManager();
2 $query = $em->createQuery(
3     'SELECT p
4     FROM AcmeStoreBundle:Product p
5     WHERE p.price > :price
6     ORDER BY p.price ASC'
7 )->setParameter('price', '19.99');
8
9 $products = $query->getResult();
```

- Histoire de rendre la chose encore plus sympathique, vous pouvez enrichir le langage DQL en **créant vos fonctions**.

## Aller encore plus loin

- Il va de soit que doctrine permet encore plus de choses :
  - **C**reate,
  - **R**ead,
  - **U**ppdate,
  - **D**eleate.
- Jointure complexe,
- La documentation de Doctrine pour Sf4 est riche :  
<http://symfony.com/fr/doc/current/book/doctrine.html>
- Mais celle de doctrine l'est encore plus : <http://www.doctrine-project.org/>

## 3 – Doctrine : TP n°4

- Créer son modèle
- Créer ses entités
- Manipuler

## Description du projet final

A rendre pour le :

03/01/2020

**2 personnes max !**

Dans le cadre de ce cours et afin de mettre en pratique l'utilisation du framework Symfony4, nous allons réaliser un blog (par exemple celui de la communauté de communes du Groland : groblog). Celui-ci devra être héberger sur la plateforme Cloud Heroku.

Le blog devra être :

En HTML5 (**sémantique** et **valide**). Utilisant un **framework CSS** (Bootstrap, pure, etc).

Le blog comprend 2 contrôleurs (Blog & Crud) et 5 routes (Action) :

IndexAction => page d'accueil (/) listant les x derniers billets (page à page),

PostAction => page billet (/post/\$id) affichant le contenu d'un billet

NewAction => Ajout

EditAction => Edition

DeleteAction => Suppression

En gris les sujet à traiter lors du prochain cours

*Le blog implémente FosUser pour se connecter (la page de login **doit être habillée**) et les routes CRUD doivent être protégées.*

*Le code source et le blog seront stockés dans le « cloud ».*

## Modèle de données de mon blog

Notre blog sera simple. Il se composera uniquement d'une table « post » composée des champs suivants :

- titre : titre de l'article.
- url\_alias (ou slug ou etc.) : URL de l'article.
- content : contenu de l'article.
- published : date de publication.

A l'aide des connaissances acquises en cours, écrivez l'entité et déployez le modèle en base.



## Alimenter la base

Afin de partir avec de vraies données, renseignez quelques billets dans votre base directement depuis phpMyAdmin de SA.P.IENS (Voir le TP du cours 2 sur comment utiliser SA.P.IENS).

Autre solution, en ligne de commande (par exemple, comme ça):

```
php bin/console doctrine:query:sql "INSERT INTO post SET  
name='RDV à Val d'Aran by', content='RDV cet été à Val  
d'Aran by UTMB(c)', slug='val-aran', display=1"
```

## Exploitation des données

Maintenant que nous avons une classe entité, nous allons modifier les contrôleurs créés précédemment :

- Le contrôleur `indexAction` va récupérer les 10 derniers billets et les passer au templates.
- Le contrôleur `postAction` va récupérer le billet en fonction de l'`url_alias` passé en URL et le passer aux templates.

## 4 – Le Service Container

- Un service c'est quoi ?
- Créer un service
- Utiliser un service

## Introduction

- Une application PHP moderne est pleine d'objets.
- Un objet peut faciliter l'envoi des messages e-mails, tandis qu'un autre peut vous permettre de persister les informations dans une base de données.
- Dans votre application, vous pouvez créer un objet qui gère votre inventaire de produits, ou tout autre objet qui traite des données via une API tierce.
- Le fait est qu'une application moderne fait beaucoup de choses et est organisée entre de nombreux objets qui gèrent chaque tâche.

## Un service c'est quoi ?

- Plus simplement, un Service désigne tout objet PHP qui effectue une sorte de **tâche « globale »**.
- C'est un nom générique utilisé en informatique pour décrire un objet qui est créé dans un **but précis** (par ex. l'envoi des e-mails).
- Chaque service est utilisé tout au long de votre application lorsque vous avez besoin de la fonctionnalité spécifique qu'il fournit.
- Vous n'avez pas à faire quelque chose de spécial pour fabriquer un service : il suffit d'écrire une classe PHP avec un code qui accomplit une tâche spécifique.



## Créer un service 1/2

- Exposer une classe en tant que service est simple et passe par un fichier de configuration.
- On peut aussi lui passer des arguments au constructeur.

```
1  # app/config/config.yml
2  services:
3      my_mailer:
4          class:      Acme\HelloBundle\Mailer
5          arguments:  [sendmail]
```

Notez le paramètre passé au service.

**Conseil :** Encore mieux, utilisez un fichier *services.yml* dans votre bundle pour y exposer les services de votre bundle.

## Créer un service 2/2

- Depuis symfony 2.8 un nouveau système autowire/autoconfigure permet la déclaration implicite de service sans devoir spécifier les configuration dans le fichier services .yml.

```
5
6 services:
7   _defaults:
8     autowire: true
9     autoconfigure: true
10    public: false
11
12   AppBundle\:
13     resource: '../src/AppBundle/*'
14     exclude: '../src/AppBundle/{Entity,Repository,Tests}'
```

## Utiliser un service

- Utiliser le service est alors simple grâce à l'injection de dépendance ou du *get* :

```
class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(LoggerInterface $logger)
    {
        $logger->alert('Test');
        $router = $this->get('router');

        return $this->render('default/index.html.twig', []);
    }
}
```

Accessible depuis « *\$this* » car nous sommes dans un *controller*.

## Argument de services

- Il est possible de passer des arguments de type string.

```
1 # app/config/services.yml
2 parameters:
3     app.mailer.transport: sendmail
4
5 services:
6     app.mailer:
7         class:      AppBundle\Mailer
8         arguments:  ['%app.mailer.transport%']
```

- Mais la chose devient intéressante lorsque l'on passe des services et que l'on fait de l'injection de services !

```
1 // src/AppBundle/Newsletter/NewsletterManager.php
2 namespace AppBundle\Newsletter;
3
4 use AppBundle\Mailer;
5
6 class NewsletterManager
7 {
8     protected $mailer;
9
10    public function __construct(Mailer $mailer)
11    {
12        $this->mailer = $mailer;
13    }
14
15    // ...
16 }
```

```
1 # app/config/services.yml
2 services:
3     app.mailer:
4         # ...
5
6     app.newsletter_manager:
7         class:      AppBundle\Newsletter\NewsletterManager
8         arguments:  ['@app.mailer']
```

## Pour conclure

- Dans Symfony il faut utiliser au maximum les services et oublier les singletons.
- Vous verrez par la suite que nous utiliserons des composants natifs de Sf4 qui sont eux aussi des services.
- Une fois vos classes de service exposées vous pouvez les récupérer partout au sein de votre application.
- Différents type d'arguments sont passables aux services et vous pouvez également leur passer des services (@ en YAML) et ainsi faire de l'injection de service.
- Vous pouvez également appeler des fonction « call » lors de l'initialisation d'un service.

**Conseil :** Il est possible de passer le service container (@service\_container), l'objet qui contient (entre autre) tous les services. C'est facile, **mais c'est mal !**

Aller plus loin : [Service Container](#).

## 5 – La traduction dans Sf4

- Activation
- Les fichiers de traduction
- Utilisation



## Activer le service de traduction

- Les traductions sont traitées par le **service** *Translator* qui utilise la locale de l'utilisateur pour chercher et retourner les messages traduits. Avant de l'utiliser, activez le Translator dans votre configuration :

```
1 # app/config/config.yml
2 framework:
3     translator: { fallback: en }
```

- L'option `fallback` définit la locale de secours à utiliser quand une traduction n'existe pas dans la locale de l'utilisateur.

## Différents fichier de traduction

- Comme toujours :-(, Symfony4 propose plusieurs formats de fichiers. La traduction n'échappe pas à cela et propose 3 formats :
  - **XML / XLIFF**,
  - PHP,
  - YAML.
- La forme la plus simple étant le YAML, cependant **la documentation officielle recommande le XLIFF** qui est AMHA la forme la plus complexe (mais aussi la plus puissante).

# XLIFF VS YAML

```
1 <!-- messages.fr.xliff -->
2 <?xml version="1.0"?>
3 <xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
4   <file source-language="en" datatype="plaintext" original="file.ext">
5     <body>
6       <trans-unit id="1">
7         <source>Symfony2 is great</source>
8         <target>J'aime Symfony2</target>
9       </trans-unit>
10    </body>
11  </file>
12 </xliff>
```

```
1 # messages.fr.yml
2 Symfony2 is great: J'aime Symfony2
```

## Utilisation du service de traduction

- L'utilisation du service de traduction est alors simple :

```
class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(TranslatorInterface $translator)
    {
        $translationMessage = $translator->trans('Symfony is great');

        return $this->render('default/index.html.twig', [
            'translationMessage' => $translationMessage
        ]);
    }
}
```

## Aller plus loin

Nous venons de voir les bases de la traduction dans Sf4. Mais l'outil va bien plus loin :

- Paramètres de substitution,
- Utilisation de mots clef,
- Utilisation de domaines,
- Formes plurielles complexes (ex : Russe),
- Traduction au sein des templates Twig.

Aller plus loin : [Translation](#).

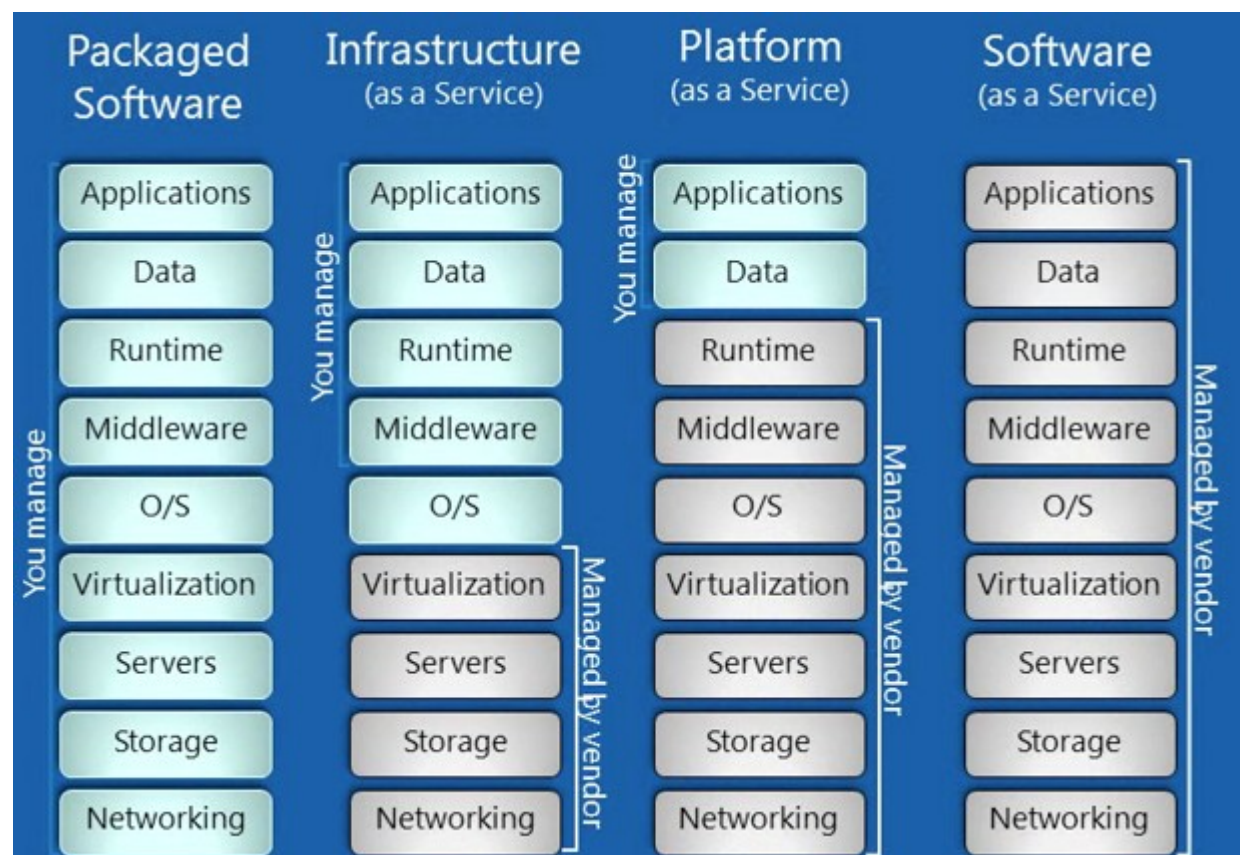
## 6 – Déployer une application Sf4 dans le cloud

- Le cloud
- Pourquoi utiliser le PaaS
- Les différents PaaS pour symfony
- Heroku



# Le cloud ?

« Le cloud computing c'est de pouvoir utiliser des ressources informatiques sans les posséder »



## Pourquoi utiliser le PaaS ?

- Approche no ops : le talent des développeurs 100 % consacré aux tâches à valeur ajoutée
- Une mise en production quasi instantanée
- Des capacités d'expérimentation illimitées
- Une fluidité dans l'organisation
- Une meilleur sécurité

## Iterate Faster

Write Code. We Run It.

## Les différents PaaS pour symfony

- Microsoft Azure Website Cloud
- Fortrabbit
- Platform.sh
- Heroku Cloud

## Heroku Cloud

- Heroku est un service de cloud computing de type plate-forme en tant que service. Créé en 2007, il était l'un des tout premiers services cloud, puis a été racheté par Salesforce.com.
- À l'origine dévolue aux applications web programmées en Ruby et utilisant Rack (typiquement, des applications Ruby on Rails ou Sinatra), l'offre s'est ensuite étendue à d'autres runtimes : node.js, Java, Spring et Play framework, Clojure, Python et Django, Scala, API Facebook, ainsi que PHP. Les différents runtimes coexistent aujourd'hui dans un même stack polyglotte, nommé Cedar, tournant sur une base d'Ubuntu. Le service permet le déploiement très rapide d'applications web dans le cloud, avec une gestion très souple du scaling horizontal au travers d'un modèle de gestion des processus emprunté à Unix et adapté au Web.

## 7 – TP n°5

- Déployer son application sur HEROKU

## Déployer son application sur Heroku 1/5 : Sur Heroku

- Aller sur <https://www.heroku.com/> et créez y un compte.
- Une fois connecté :
  - Cliquer sur « *Create a new App* »,
  - Pour la App-name veuillez saisir : nom1-nom2-blog (ex : kulakowski-cariou-blog),
  - Pour la région choisir « *Europe* ».
- Une fois l'application créée :
  - Cliquer sur « Settings » puis sur « Add buildpack »,
  - Choisir « *heroku/php* ».
- Ajout de la variable d'environnement :
  - Aller dans « settings » puis dans « Config Variables » ajoutez
  - APP\_ENV: prod



## Déployer son application sur Heroku 2/5 : Le client Heroku

- Installer le client Heroku (version standalone?) :  
<https://devcenter.heroku.com/articles/heroku-cli#standalone>
- Création de la ssh key :
  - Allez sur « account settings »,
  - Ajouter votre clé publique dans la section SSH Keys.
- Ajouter le support de PostgreSQL :  
`heroku addons:add heroku-postgresql`



## Déployer son application sur Heroku 3/5 : Configurez votre application

### Important

- Ajouter un fichier « Procfile » à la racine de votre projet :  
`web: vendor/bin/heroku-php-apache2 public/`
- Activer les logs dans Heroku : <https://devcenter.heroku.com/articles/deploying-symfony4#logging>
- Nettoyez le fichier **config/packages/doctrine.yaml** en supprimant cette partie :

```
# configure these for your database server
# use postgresql for PostgreSQL
# use sqlite for SQLite
driver: 'mysql'
server_version: '5.7'

# only needed for MySQL
charset: utf8mb4
default_table_options:
    charset: utf8mb4
    collate: utf8mb4_unicode_ci
```

- Ajouter la gestion de l'URL rewriting en répondant bien « yes » pour déployer la *recipe* :  
<https://devcenter.heroku.com/articles/deploying-symfony4#url-rewrites>
- Ajouter à composer la mise à jour automatique de la BDD post installation

## Déployer son application sur Heroku 4/5 :

### Push de l'application sur heroku

- Allez à la racine de votre application et lancer les commande suivante :

```
git init
heroku git:remote -a {nom1}-{nom2}-blog
git add .
git commit -am "Init project on Heroku"
git push heroku master
```

## Déployer son application sur Heroku 5/5

- L'application est déployer sur :  
<https://{nom1}-{nom2}-blog.herokuapp.com/>
- Pour finir aller dans Access et cliquer sur « Add collaborateur »
  - [cariou.loic.um2@gmail.com](mailto:cariou.loic.um2@gmail.com)
  - [guillaume@kulakowski.fr](mailto:guillaume@kulakowski.fr)

Buts :

- Avoir votre application qui fonctionne sur heroku

Personal &gt; loic-cariou-blog



Open app

More

GitHub loic-cariou/blog

[Overview](#) [Resources](#) [Deploy](#) [Metrics](#) [Activity](#) [Access](#) [Settings](#)

## Installed add-ons \$0.00/month

[Configure Add-ons](#)Heroku Postgres Hobby Dev  
postgresql-fitted-28339

## Dyno formation \$0.00/month

[Configure Dynos](#)

This app is using free dynos

web vendor/bin/heroku-php-apache2 public/

ON

## Collaborator activity

[Manage Access](#)

cariou.loic.um2@gmail.com

3 deploys

## Latest activity

[All Activity](#)cariou.loic.um2@gmail.com: Deployed 9996d05f  
Today at 5:58 PM · v10 · [Compare diff](#)cariou.loic.um2@gmail.com: Build succeeded  
Today at 5:58 PM · [View build log](#)cariou.loic.um2@gmail.com: Deployed 9996d05f  
Today at 5:57 PM · v9 · [Compare diff](#)cariou.loic.um2@gmail.com: Build succeeded  
Today at 5:57 PM · [View build log](#)cariou.loic.um2@gmail.com: Deployed 9996d05f  
Today at 5:56 PM · v8cariou.loic.um2@gmail.com: Build succeeded  
Today at 5:56 PM · [View build log](#)cariou.loic.um2@gmail.com: Remove SYMFONY\_ENV config var  
Today at 5:56 PM · v7cariou.loic.um2@gmail.com: Set APP\_ENV config var  
Today at 5:56 PM · v6cariou.loic.um2@gmail.com: Build failed  
Today at 5:55 PM · [View build log](#)

Personal > loic-cariou-blog

★ Open app More

GitHub loic-cariou/blog

Overview Resources Deploy Metrics Activity Access Settings

Free Dynos Change Dyno Type

web vendor/bin/heroku-php-apache2 public/

Toggle switch \$0.00 Edit icon

Add-ons

Find more add-ons

Quickly add add-ons from Elements

Heroku Postgres

Attached as DATABASE

Hobby Dev

Free

Estimated Monthly Cost \$0.00

## Name

Name management is not available

Only the owner, cariou.loic.um2@gmail.com, can rename this app.

## Config Vars

Config vars change the way your app behaves.  
In addition to creating your own, some add-ons come with their own.

## Config Vars

Hide Config Vars

APP\_ENV

prod



DATABASE\_URL

postgres://jxhmm1easovkfk:f46e7402eab89l



KEY

VALUE

Add

## Info

Region

 Europe

Stack

heroku-18


Framework

 PHP

Slug Size

23.5 MiB of 500 MiB

GitHub Repo

 loic-cariou/blog

Heroku Git URL

<https://git.heroku.com/loic-cariou-blog.git>

## Buildpacks

## 8 – Installation d'un bundle tiers

- Comment
- Où les trouver



## Installer un bundle tiers

Là encore, plusieurs façons de faire. Je vous conseille de passer par composer :

```
{
    ...,
    "require": {
        ...,
        "friendsofsymfony/user-bundle": "2.0.*@dev"
    }
}
```

2. `php composer.phar update`

## Trouver des bundles tiers

- [KnpBundle](#) propose un large éventail de bundle.
- Les Bundles que vous allez regretter de ne pas avoir connu plus tôt.

Les meilleurs bundles...

# LES BUNDLES

QUE VOUS ALLEZ REGRETTER DE  
NE PAS AVOIR CONNU PLUS TÔT

Une présentation par [Damien Alexandre](#) / JoliCode  
Symfony Live Paris 2013

## 9 – TP n°6

- Installation d'un bundle tiers.

## Installation de FOSUserBundle

- Lisez la [documentation de FosUserBundle](#),
- Appliquez la.

Buts :

- Avoir une connexion utilisateur via MySQL.
- Apprendre à lire / utiliser / appliquer une documentation.

# Annexes

## Guide de survit de bin/console

- Vider le cache :

```
bin/console cache:clear
```

- Générer mes getter et setter de mes entities :

```
php bin/console make:entity App/Entity/Product
```



## Webographie

- Site officiel de Symfony : <http://symfony.com/>
- Le manuel PHP : <http://php.net/manual/fr/index.php>
- La doc : <http://symfony.com/doc/current/book/index.html>

## Bibliographie

- Toutes la documentation et les « books » officiels sont librement téléchargeables  
<http://symfony.com/doc/current/index.html>

## Licence

La documentation Symfony4 étant bien faite et sous Licence libre (MIT), des illustrations tout comme des bouts de textes de ce cours en sont extraits.