

e-application - 2019

2. Découverte de Symfony4

Guillaume Kulakowski

Architecte Technique en solutions open-source @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
cariou.loic.um2@gmail.com



<https://blog.kulakowski.fr>

CGI



UNIVERSITÉ
DE MONTPELLIER

Agenda

1. Symfony4, les outils
2. Pourquoi un framework
3. Pourquoi Symfony
4. Comprendre les concepts de bases de Symfony4
5. L'architecture logicielle
6. TD n°2 part 1
7. Le moteur de template de Sf4 : Twig
8. TD n°2 part 2
9. Outillage & industrialisation

1 – Symfony4, les outils

- Composer
- Symfony Installer (legacy)

Composer

- Composer est un gestionnaire de dépendances open source écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin. Le développement a débuté en avril 2011 et a donné lieu à une première version sortie le 1er mars 2012. Développé au début par Nils Adermann et Jordi Boggiano (qui continuent encore aujourd'hui à le maintenir), le projet est maintenant disponible sur la plateforme GitHub. Il est ainsi développé par toute une communauté.
- Composer est à l'initiative d'un portage en PHP de Libzypp satsolver d'Open Suse.
- Composer est fortement inspiré de npm pour Node.js et de bundler pour Ruby.
- [Installer Composer](#).
- Le référentiel des libs : [Packagist](#).



Le client composer

- Installer une *lib* au sein d'un projet :
`composer require symfony/yaml`
- Installer toutes les *libs* définies dans le ***composer.json*** :
`composer install`
- Mettre à jour les *libs* d'un projet :
`composer update`
- Installer globalement (*root* ou *user*) une *lib* dans son système :
`composer global require symfony/symfony-installer`
- [Aller plus loin avec la ligne de commande.](#)

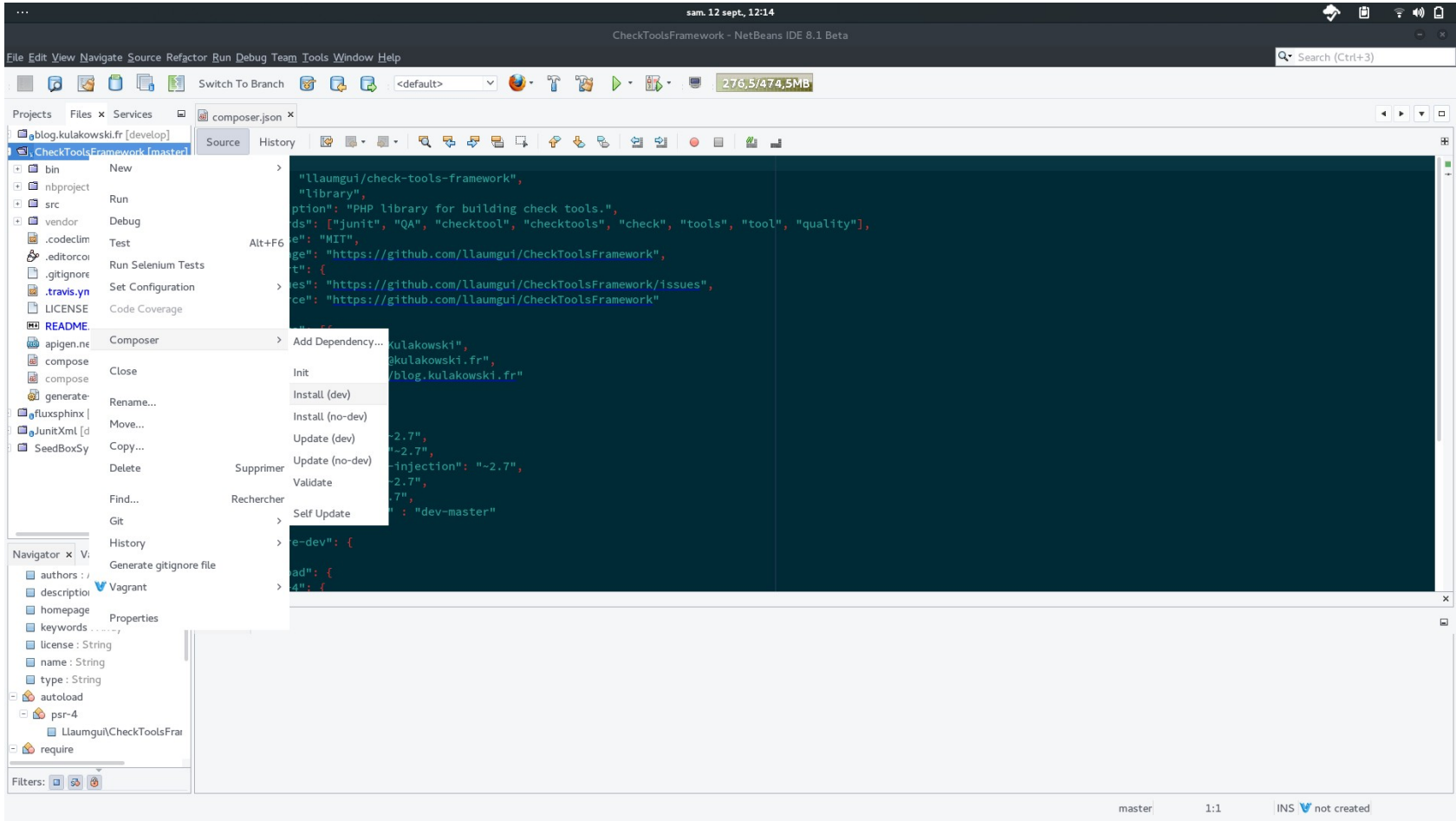
composer.json & composer.lock

- Le *composer.json* est un fichier au format JSON qui regroupe les dépendances du projet dans un **format particulier** :

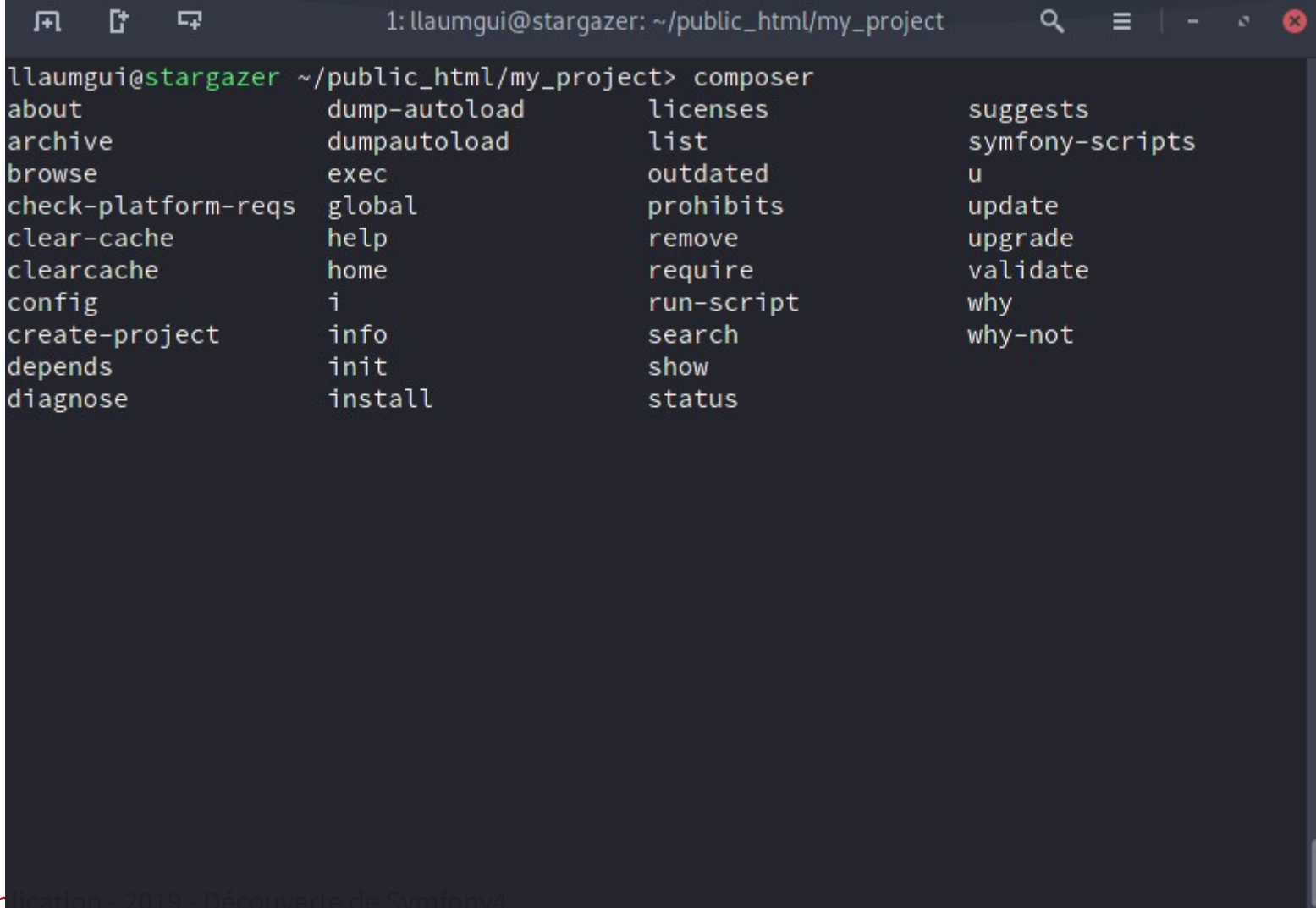
```
{  
    "require": {  
        "monolog/monolog": "1.0.*@beta",  
        "acme/foo": "@dev"  
    }  
}
```

- Le *composer.lock* est une sorte de cache des références exactes induites par le *composer.json* à un instant « t ».

7 Composer dans votre EDI (ex : Netbeans)



Composer dans votre shell (ex : Oh My ZSH)



A terminal window titled "1: llaumgui@stargazer: ~/public_html/my_project" displays the output of the 'composer' command. The output is a list of 20 Composer commands arranged in four columns. The window has a dark theme and standard window controls at the top.

```
llaumgui@stargazer ~/public_html/my_project> composer
about          dump-autoload  licenses      suggests
archive        dumpautoload   list          symfony-scripts
browse         exec           outdated      u
check-platform-reqs global         prohibits     update
clear-cache    help           remove        upgrade
clearcache     home          require       validate
config         i             run-script    why
create-project info          search        why-not
depends         init          show
diagnose       install      status
```


Symfony Installer (legacy)

- *Symfony Installer* est un client permettant d'installer Symfony en ligne de commande.
- Créer un nouveau projet :
`symfony new my_project`
- Créer un nouveau projet avec la version LTS :
`symfony new my_project lts`
- Créer un nouveau projet en Symfony 2.3 (ancienne lts) :
`symfony new my_project 2.3`
- [Démo de Symfony Installer.](#)
- [Documentation de Symfony Installer.](#)

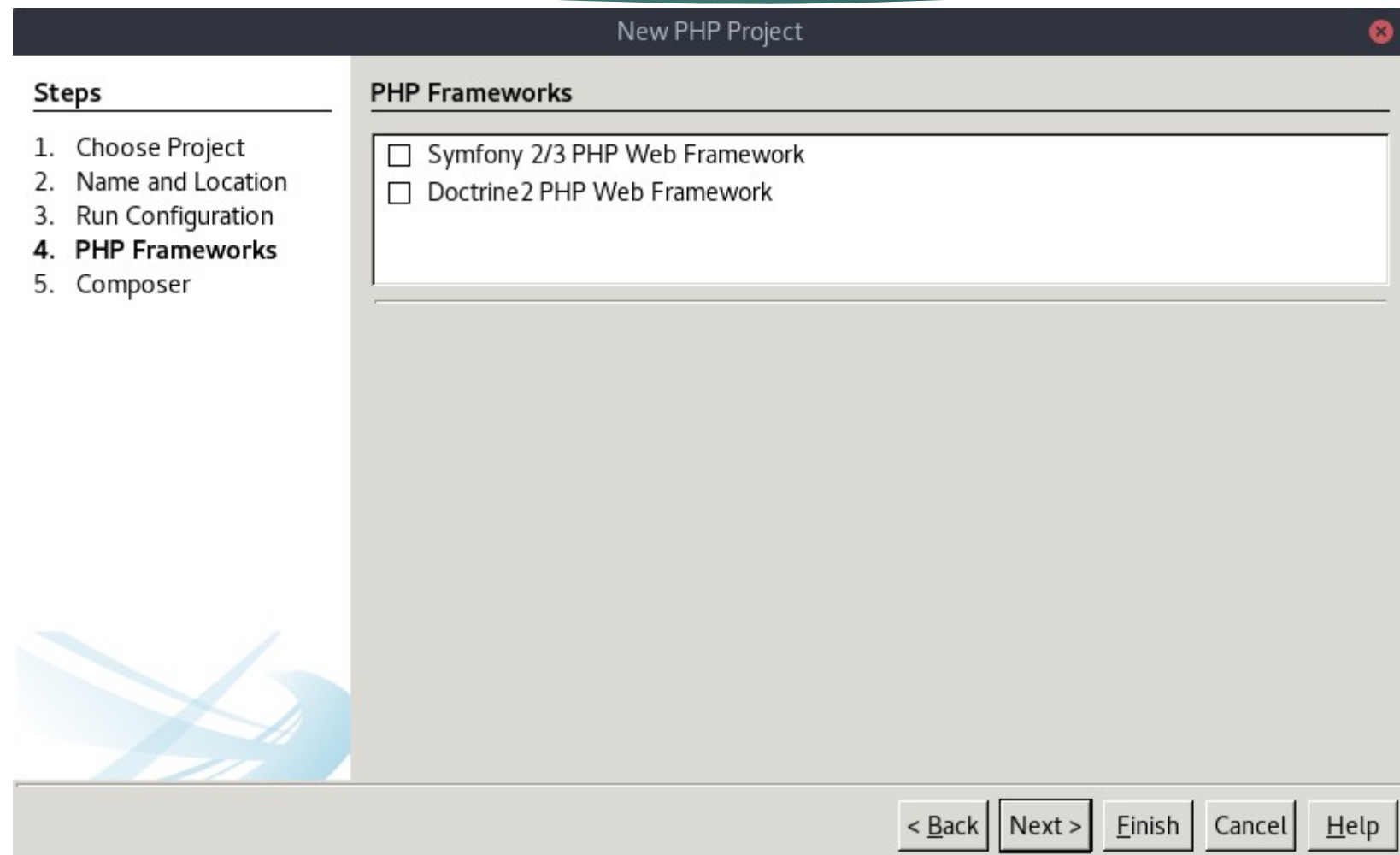


Déprécié
sous Sf4

Installer Symfony4 avec composer

- Créer un nouveau projet de site web:
`composer create-project symfony/website-skeleton my_project_name`
- Pour une version nude :
`composer create-project symfony/skeleton my_project_name`
- [Documentation d'installation de Symfony.](#)

Symfony4 dans votre EDI (ex : Netbeans)



2 – Pourquoi un framework ?

- Les avantages d'un framework
- Les similitudes entre framework

Un framework c'est quoi ?

En programmation informatique, un framework est un kit de composants logiciels structurels, qui servent à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). En programmation orientée objet un framework est typiquement composé de classes mères qui seront dérivées et étendues par héritage en fonction des besoins spécifiques à chaque logiciel qui utilise le framework.

Les framework sont utilisés pour modeler l'architecture des logiciels applicatifs, des applications web, des middleware et des composants logiciels. Les framework sont achetés par les ingénieurs, puis incorporés dans des logiciels applicatifs mis sur le marché, ils sont par conséquent rarement achetés et installés séparément par un utilisateur final.

Des tentatives de francisation du terme ont été faites. On trouve ainsi parfois les termes cadre d'applications, proposé par l'Office québécois de la langue française, canevas ou cadriciel – terme en usage depuis au moins 1997. (© Wikipedia)

Un framework pourquoi ?

- Développement haut niveau (la roue existe déjà), plus valorisant pour les équipes.
- Gain de temps = Gain de productivité = € £ \$.
- Maturité du code.
- Présence de tests.
- Une contrainte afin de cadrer les développements et les équipes.
- Communauté adhérant à la solution.
- Documentation.



Les différents types de frameworks 1/2

Comme abordé précédemment, il existe des frameworks pour tout :

- Frameworks CSS :
 - Bootstrap by Twitter
 - Pure
 - Etc...
- Frameworks JavaScript
 - jQuery
 - Prototype
 - MooTools
 - Etc...

Les différents types de frameworks 2/2

- Frameworks php :
 - Symfony4
 - Laravel
 - Yii
 - Etc...
- Frameworks Python
 - Django
 - Etc...

Des composants et « Design Pattern » proches 1/3

En informatique, et plus particulièrement en développement logiciel, un patron de conception (en anglais : « design pattern ») est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels.

Un patron de conception est issu de l'expérience des concepteurs de logiciels. Il décrit sous forme de diagrammes un arrangement récurrent de rôles et d'actions joués par des modules d'un logiciel, et le nom du patron sert de vocabulaire commun entre le concepteur et le programmeur. D'une manière analogue à un patron de couture, le patron de conception décrit les grandes lignes d'une solution, qui peuvent ensuite être modifiées et adaptées en fonction des besoins.

Les patrons de conception décrivent des procédés de conception généraux et permettent en conséquence de capitaliser l'expérience appliquée à la conception de logiciel. Ils ont une influence sur l'architecture logicielle d'un système informatique. (© Wikipedia)

Des composants et « Design Pattern » proches 2/3



Framework Ikéa,
Composant Billy



Framework Alinéa,
Composant Book'in

Des composants et « Design Pattern » proches 3/3

Des composants répondant à des besoins :

- Bâtir une architecture logicielle : **MVC**.
- Une application multilingues : **i18n** (fr) & **l10n** (fr_fr).
- L'utilisation d'une base de données : **ORM**.
- Une mise en cache : **cache**.
- L'utilisation de fichiers de configurations : **configuration**.
- L'enregistrement des événements : **log**.
- La séparation fond / forme : **templates**.
- Etc...

3 – Pourquoi un Symfony4 ?

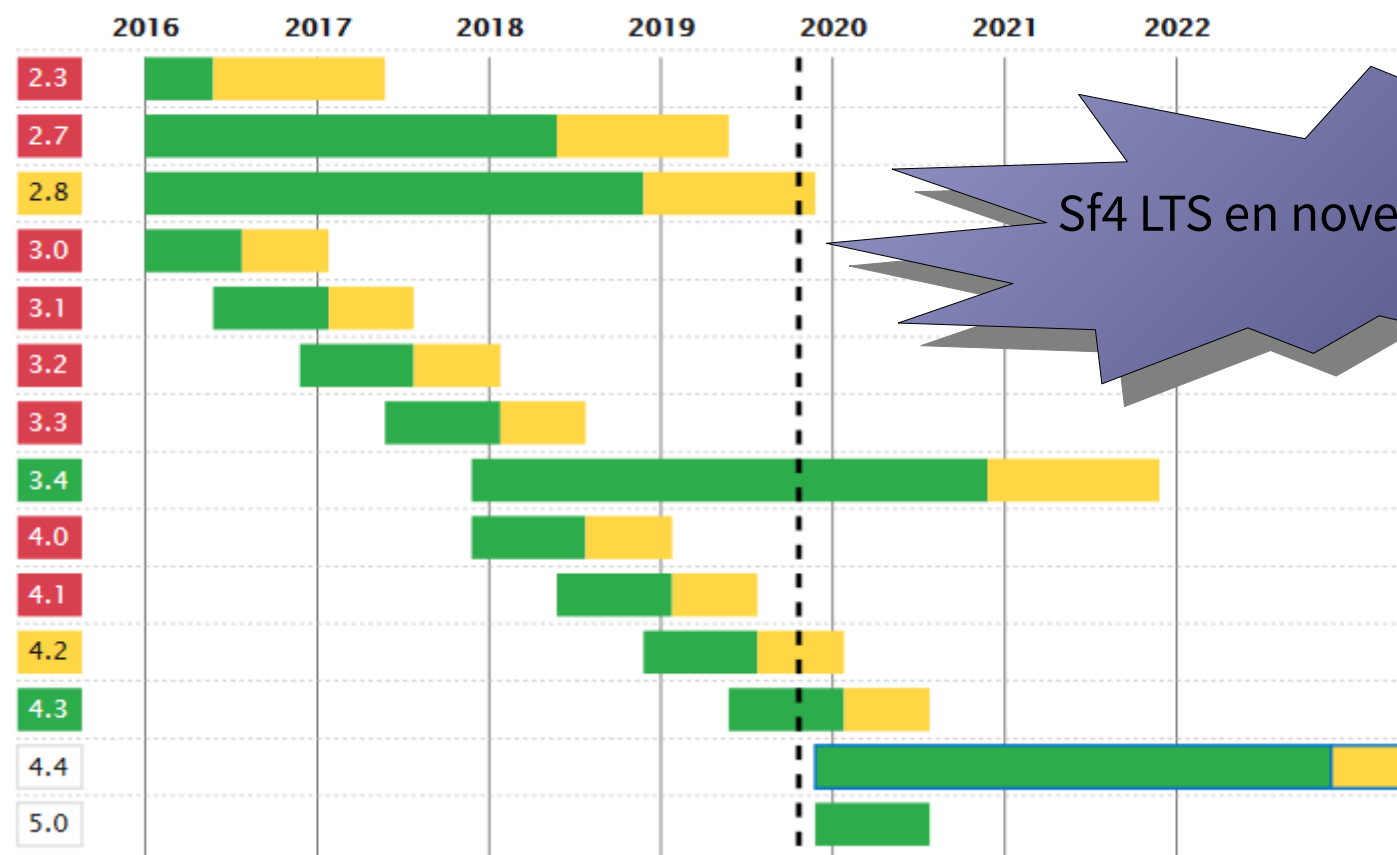
- Présentation
- Les forces
- Les faiblesses

Symfony4 en quelques mots & chiffres

- Le site du framework Symfony a été lancé en **octobre 2005** (maturité). À l'origine du projet, on trouve une agence web française, **Sensio**, qui a développé ce qui s'appelait à l'époque Sensio Framework pour ses propres besoins et a ensuite souhaité en partager le code avec la communauté des développeurs PHP. Le projet est alors devenu **symfony** (car le créateur voulait garder les initiales Sf comme "Sensio Framework"), puis **Symfony** à partir de la **version 2.0**.
- **La version 3.0** de Symfony est une évolution de la branche 2 qui casse la compatibilité PHP 5.3.3 pour requérir 5.5.9.
- **La version 4.0** de Symfony casse la compatibilité PHP 5 pour **requérir PHP 7.1**. Elle introduit une nouvelle architecture logicielle basée sur **Flex et les recipes**.
- Licence : **MIT**.
- Sources disponibles sur GitHub : <https://github.com/symfony> .
- Exemples de grosses utilisations : [Dailymotion](#), Yahoo ! ([Delicio.us](#)), [Mappy](#), etc...
- Intégré dans : [eZ Platform](#) (CMS en full stack Sf2), [Drupal 8](#) (CMS utilisant les composants), etc...
- ~1.200 bundles sur [Knp Bundles](#).

Symfony, une roadmap claire

Symfony Releases Calendar



Sf4 LTS en novembre 2019 !!!

3.4 en LTS, 4.3 dernière stable, 4.4 LTS pour Novembre 2019 !

[Home](#) / [What is Symfony](#) / [Symfony Releases](#) / [Symfony 4.4 Release](#)

Symfony 4.4 Release

Status:

In development

Release date:

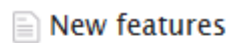
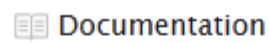
November 2019

End of bug fixes:

November 2022

End of security fixes:

November 2023



★ This will be a long-term support version.

💡 **TIP** : Get this information in JSON format: <https://symfony.com/releases/4.4.json>

Les points forts de Symfony4

- Une documentation faisant partie intégrante du framework et elle aussi sous licence libre : <http://symfony.com/doc/current/index.html>,
- Un bundle e-Commerce tiers : <http://www.sylius.com/>,
- Des bundles officiels & tiers pour vous aider à développer,
- Des bonnes pratiques et des règles ([Coding standard](#)),
- Une approche orientée services,
- Disponible en full stack ou en composants,
- Full MVC,
- Une intégration au sein de pas mal d'EDI (ex : Netbeans).

Les points faibles de Symfony4

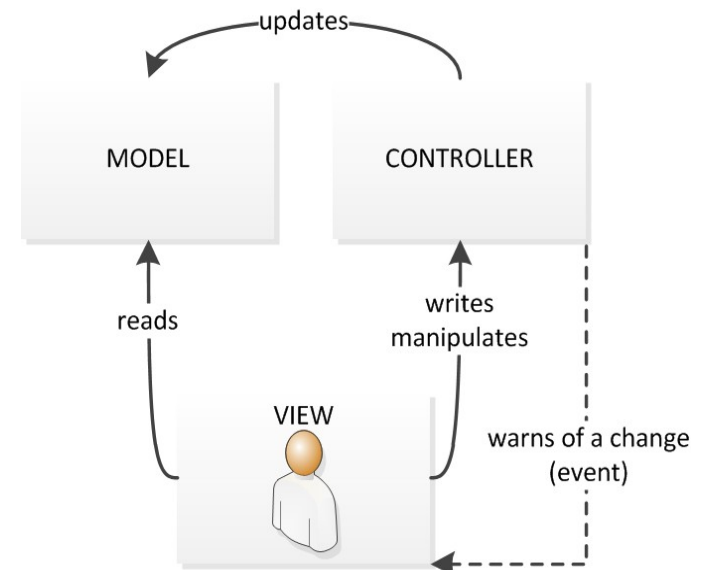
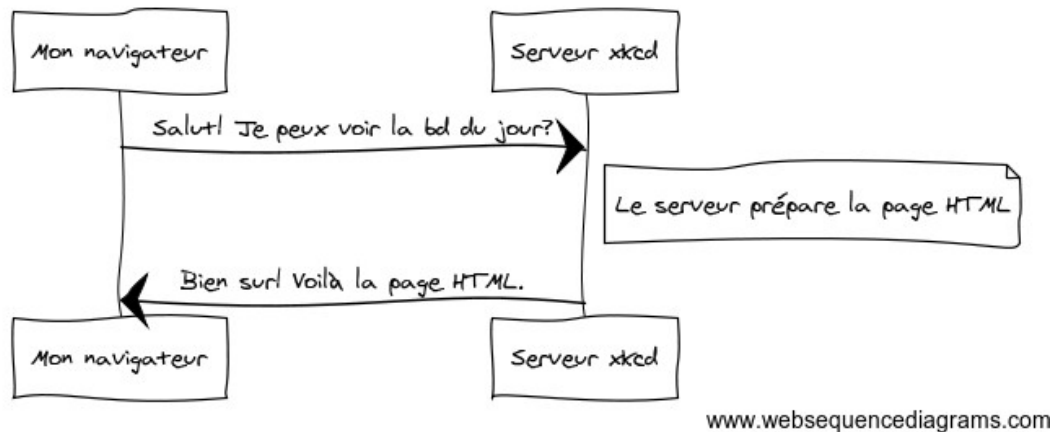
- Riche donc demande de bonnes compétences pour en maîtriser toutes les facettes,
- Lourd en environnement de développements,
- **Composer** qui peut ramener de façon presque invisible des bundles à la stabilité discutable.

4 – Comprendre les concepts de bases de Symfony4

- Les fondamentaux de Symfony4 et HTTP
- La requêtes & le contrôleur
- HttpFoundation

L'HTTP 1/3

- Il est important de comprendre le fonctionnement de l'HTTP, car c'est sur ce principe simple que repose la base de Symfony4 : router la bonne requête vers le bon contrôleur.
- En effet, Symfony4 est un framework full MVC.



L'HTTP 2/3

URL	Fichier
/index.php	exécute index.php
/index.php/contact	exécute index.php
/index.php/blog	exécute index.php

```
// index.php
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
$request = Request::createFromGlobals();
$path = $request->getPathInfo(); // Le chemin de l'URI demandée

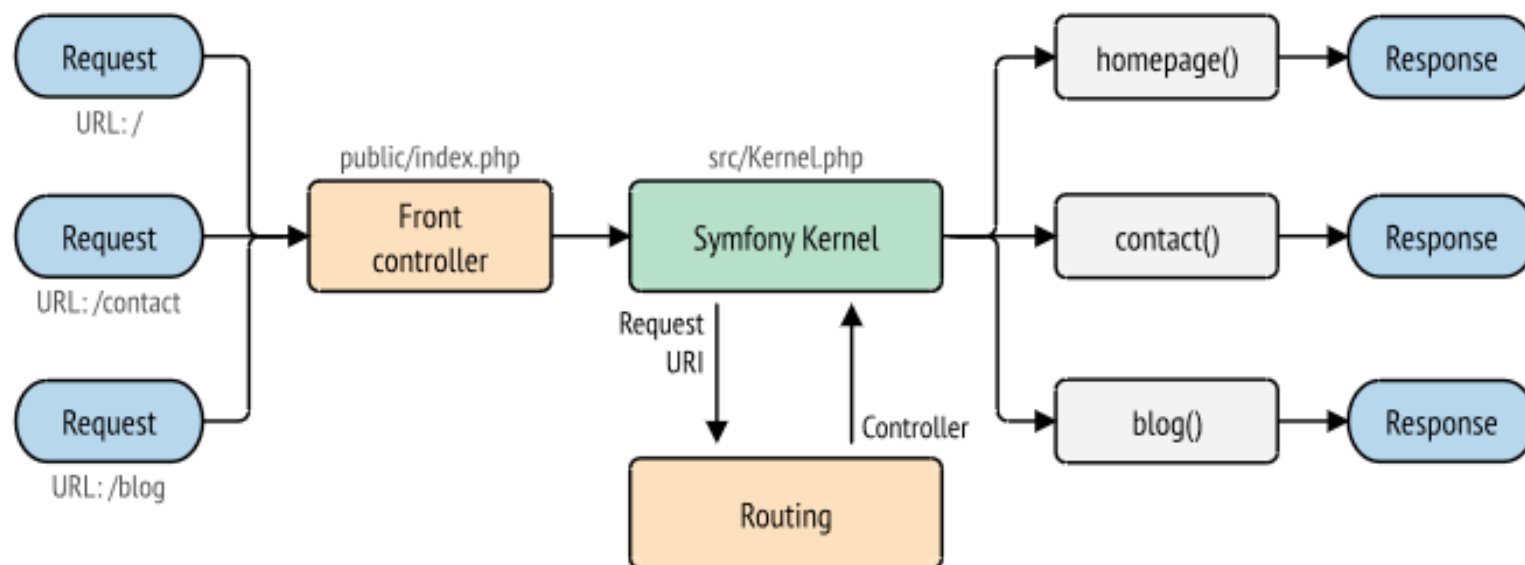
if (in_array($path, array('', '/'))){
    $response = new Response('Bienvenue sur le site.');
```

```
} elseif ($path == '/contact') {
    $response = new Response('Contactez nous');
```

```
} else {
    $response = new Response('Page non trouvée.', 404);
}
```

```
$response->send();
```


L'HTTP 3/3



Aller plus loin : [Symfony and HTTP Fundamentals](#).

Manipuler l'HTTP avec HttpFoundation

```
use Symfony\Component\HttpFoundation\Request;

$request = Request::createFromGlobals();

// l'URI demandée (par exemple: /about) sans aucun paramètre
$request->getPathInfo();

// obtenir respectivement des variables GET et POST
$request->query->get('foo');
$request->request->get('bar', 'valeur par défaut si bar est inexistant');

// obtenir les variables SERVER
$request->server->get('HTTP_HOST');

// obtenir une instance de UploadedFile identifiée par foo
$request->files->get('foo');

// obtenir la valeur d'un COOKIE value
$request->cookies->get('PHPSESSID');

// obtenir un entête de requête HTTP request header, normalisé en minuscules
$request->headers->get('host');
$request->headers->get('content_type');

$request->getMethod(); // GET, POST, PUT, DELETE, HEAD
$request->getLanguages(); // un tableau des langues que le client accepte
```

Router ma requête sur mon contrôleur 2/2

Nous avons vu, jusqu'à présent, des cas de routages orientés *Sf Components*. Voici un cas concret dans une application *Sf4 full stack* utilisant YAML et les *@nnotations*.

```
# config/routes/annotations.yml
app:
  resource:
    "../../src/Controller/"
  type:      annotation
```

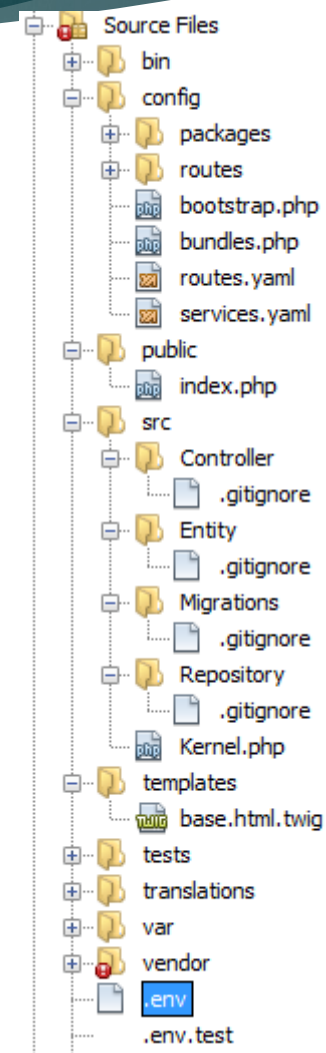
```
1  // src/Controller/LuckyController.php
2  namespace App\Controller;
3
4  use Symfony\Component\HttpFoundation\Response;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class LuckyController
8  {
9      /**
10       * @Route("/lucky/number/{max}", name="app_lucky_number")
11       */
12       public function number($max)
13       {
14           $number = random_int(0, $max);
15
16           return new Response(
17               '<html><body>Lucky number: '.$number.'</body></html>'
18           );
19       }
20  }
```

5 - L'architecture logicielle

- L'arborescence applicative
- Les fichiers importants
- Les environnements

L'arborescence d'un projet Sf4

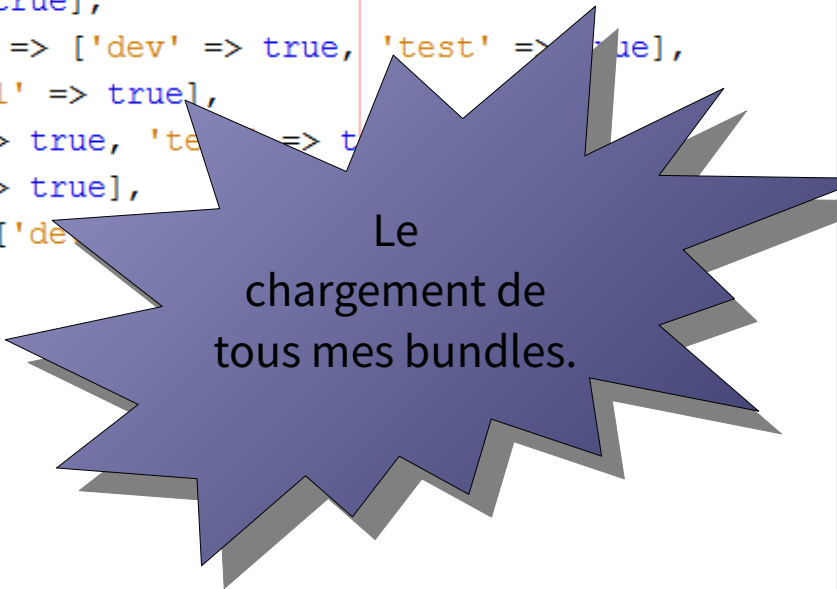
- **bin** : console de Sf4, phpUnit et binaire nécessaires au projet.
- **config** : configuration de mon application.
- **config/packages** : configuration dédiée aux packages. 1 fichier = 1 package.
- **config/routes** : configuration des routes particulières (par environnement).
- **config/bootstrap.php** : la gestion des autoloads.
- **config/bundles.php** : la gestion du chargement des bundles (anciennement alouée à AppKernel.php).
- **public** : Racine (DocumentRoot) du serveur web (ex Apache).
- **src** : code source de l'application.
- **templates** : templates de l'application.
- **var/cache** (app/cache sous Sf2) : cache de l'application par environnements.
- **var/log** (app/cache sous Sf2) : logs de l'application.
- **vendor** : les bundles third party.
- **.env*** : Simulateur de variables d'environnements.



Le bundles.php

```
<?php

return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
    Doctrine\Bundle\DoctrineCacheBundle\DoctrineCacheBundle::class => ['all' => true],
    Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class => ['all' => true],
    Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle::class => ['all' => true],
    Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],
    Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle::class => ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MonologBundle\MonologBundle::class => ['all' => true],
    Symfony\Bundle\DebugBundle\DebugBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
    Symfony\Bundle\WebServerBundle\WebServerBundle::class => ['dev' => true],
];
```



Le
chargement de
tous mes bundles.

L'index.php, là où tout commence

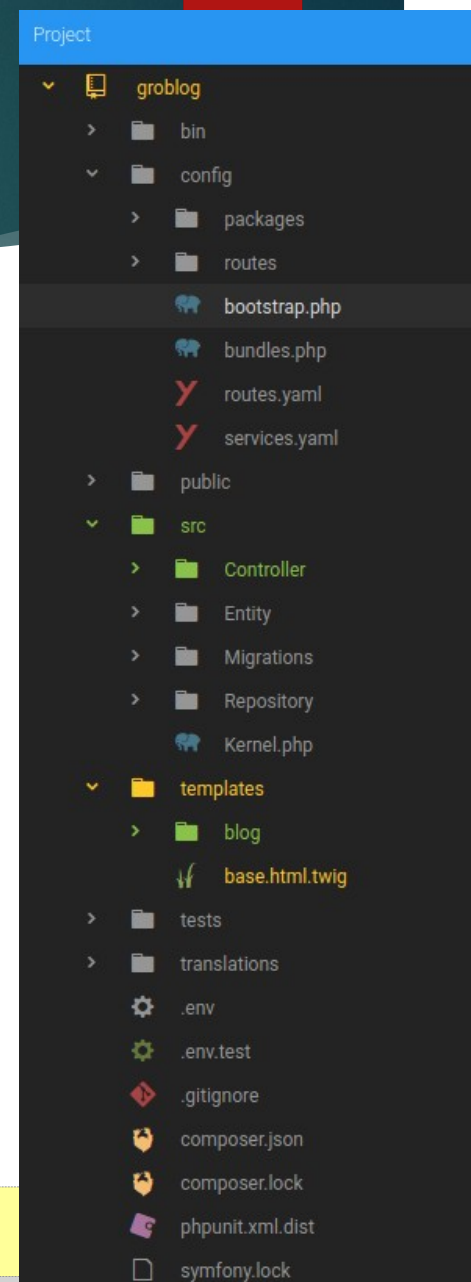
```
1  <?php
2
3  use App\Kernel;
4  use Symfony\Component\ErrorHandler\Debug;
5  use Symfony\Component\HttpFoundation\Request;
6
7  require dirname(__DIR__).'/config/bootstrap.php';
8
9  if ($_SERVER['APP_DEBUG']) {
10     umask(0000);
11
12     Debug::enable();
13 }
14
15 if ($trustedProxies = $_SERVER['TRUSTED_PROXIES'] ?? $_ENV['TRUSTED_PROXIES'] ?? false) {
16     Request::setTrustedProxies(explode(',', $trustedProxies), Request::HEADER_X_FORWARDED_ALL ^ Request::HEADER_X_FORWARDED_HOST);
17 }
18
19 if ($trustedHosts = $_SERVER['TRUSTED_HOSTS'] ?? $_ENV['TRUSTED_HOSTS'] ?? false) {
20     Request::setTrustedHosts([$trustedHosts]);
21 }
22
23 $kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
24 $request = Request::createFromGlobals();
25 $response = $kernel->handle($request);
26 $response->send();
27 $kernel->terminate($request, $response);
```

Les différentes configurations

Symfony4 permet des **configurations par environnement**. Ceci est idéal pour pouvoir modifier des paramètres entre les différents environnements que connaît une application professionnelle :

- Développement,
- Intégration,
- Recette,
- Préproduction,
- Production.

Aller plus loin : [Configuration Environments](#).



6 – Installer Symfony4 : TD n°2 (part 1)

- Installer Symfony4

Description du projet final

A rendre pour le :
03/01/2020

Dans le cadre de ce cours et afin de mettre en pratique l'utilisation du framework Symfony, 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 HTLM5 (**sémantique** et **valide**). Utilisant un **framework CSS** (Bootstrap, pure, etc).

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

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

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

NewPost => Ajout

EditPost => Edition

DeletePost => Suppression

En gris les sujet à traiter lors des prochains 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 ».

Installation de Symfony4 1/2

Il existe plusieurs façons d'installer Symfony4 :

- Via Composer,
- Via Symfony Installer.

La meilleure façon, et celle conseillée, est à partir de Composer.

Installation de Symfony4 2/2

- Installer votre projet :

```
composer create-project symfony/website-skeleton my_project_name
```

- Puis lancer le serveur (normalement il vous suffit de lire la sortie de la commande en haut pour savoir comment)

Remarque : pour développer plus facilement, nous allons utiliser le serveur embarqué dans Sf4 et non pas apache ou nginx.

Si vous voulez faire tourner Sf4 via un serveur d'application, installez-le à la racine de votre serveur (*DocumentRoot* sous apache).

Aller plus loin : [Installing & Setting up the Symfony Framework.](#)

7 - Le moteur de template de Sf4 : Twig

- La syntaxe
- Tags
- Filtres
- Fonctions
- Les assets

La syntaxe

- `{{ ... }}`: « Dit quelque chose »: écrit une variable ou le résultat d'une expression dans le template,
- `{% ... %}`: « Fait quelque chose »: un tag qui contrôle la logique du template ; il est utilisé pour exécuter des instructions comme la boucle for par exemple,
- `{# #}`: Commentaire, peut être utilisée sur plusieurs lignes comme la syntaxe PHP `/* commentaire */` qui est équivalente.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Bienvenue sur Symfony !</title>
5   </head>
6   <body>
7     <h1>{{ page_title }}</h1>
8
9     <ul id="navigation">
10       {% for item in navigation %}
11         <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
12       {% endfor %}
13     </ul>
14   </body>
15 </html>
```

Filtres et tags

- Twig contient également des filtres, qui modifient le contenu avant de le rendre. Le filtre suivant met la variable `title` en majuscule avant de la rendre :

```
{{ title | upper }}
```

- Twig est fourni avec une longue liste de [tags](#) et de [filtres](#) qui sont disponibles par défaut. Vous pouvez même ajouter [vos propres extensions](#) à Twig si besoin.

Fonctions

Comme vous le verrez tout au long de la documentation, Twig supporte aussi les [fonctions](#), et de nouvelles fonctions peuvent être ajoutées.

Par exemple, la fonction suivante utilise le tag standard *for* et la fonction *cycle* pour écrire dix balises *div* en alternant les classes *odd* et *even* :

```
1 {% for i in 0..10 %}  
2     <div class="{ cycle(['odd', 'even'], i) }">  
3         <!-- some HTML here -->  
4     </div>  
5 {% endfor %}
```

Twig et le cache

Twig est rapide. Chaque template Twig est compilé en une classe PHP natif qui est rendue à l'exécution.

Les classes compilées sont stockées dans le répertoire `var/cache/{environment}/twig` (où {environment} est l'environnement, par exemple dev ou prod) et elles peuvent être utiles dans certains cas pour déboguer

Pour vider votre cache :

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

L'héritage de template et le layout 1/2

L'une des forces Twig par rapport aux autres moteurs de templates et d'implémenter le concept d'héritage de templates.

- Tout d'abord définissons un layout ainsi que des blocs :

```
1  {# app/Resources/views/base.html.twig #}  
2  <!DOCTYPE html>  
3  <html>  
4      <head>  
5          <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
6          <title>{% block title %}Test Application{% endblock %}</title>  
7      </head>  
8      <body>  
9          <div id="sidebar">  
10             {% block sidebar %}  
11                 <ul>  
12                     <li><a href="/">Home</a></li>  
13                     <li><a href="/blog">Blog</a></li>  
14                 </ul>  
15             {% endblock %}  
16         </div>  
17  
18         <div id="content">  
19             {% block body %}{% endblock %}  
20         </div>  
21     </body>  
22 </html>
```


L'héritage de template et le layout 2/2

- Ensuite, déclarons que mon template hérite de mon layout et surchargeons les blocs :

```
1  {%# src/Acme/BlogBundle/Resources/views/Blog/index.html.twig #%}
2  {% extends '::base.html.twig' %}
3
4  {% block title %}My cool blog posts{% endblock %}
5
6  {% block body %}
7      {% for entry in blog_entries %}
8          <h2>{{ entry.title }}</h2>
9          <p>{{ entry.body }}</p>
10         {% endfor %}
11     {% endblock %}
```

Lier vos assets

- Des images :

```

```

- Des CSS :

```
<link href="{{ asset('css/blog.css') }}" rel="stylesheet" />
```

- Versionner vos assets :

```

```

- Dans la vie réelle :

```
<link href="{{ asset('bundles/acmedemo/css/contact.css') }}"  
rel="stylesheet" />
```

Less, SaaS avec Webpack Encore

```
1  // webpack.config.js
2  // ...
3
4  Encore
5      // ...
6
7      // enable just the one you want
8
9      // processes files ending in .scss or .sass
10     .enableSassLoader()
11
12     // processes files ending in .less
13     .enableLessLoader()
14
15     // processes files ending in .styl
16     .enableStylusLoader()
17 ;
```

Aller plus loin :

- [CSS Preprocessors: Sass, LESS, Stylus, etc.](#)
- [Managing CSS and JavaScript.](#)

8 – Installer Symfony4 : TD n°2 (part 2)

- Exploiter le dossier SRC

Ma première page

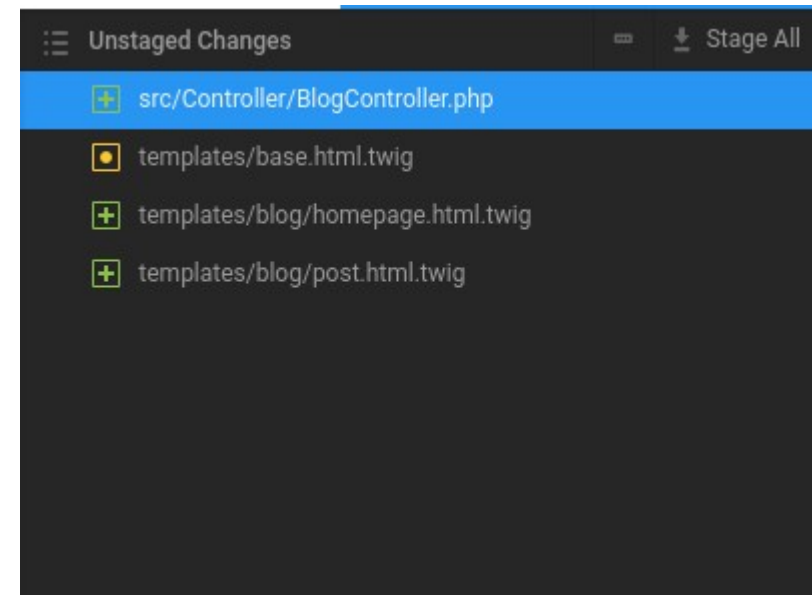
Pour les besoins du projet, nous allons réaliser un contrôleur (BlogController) et 2 routes :

- / => index
- /post/idudpost => vue de détail

Dans un premier temps, nous nous contenterons que l'index rende le texte « Page d'accueil » via un template Twig et que la page post rende juste le paramètre passé en argument à l'url.

Indices :

- Contrôleur,
- Rendre des templates.



Conclusion

- Maintenant nous avons une base solide pour commencer nos devs !

9 - Outillage & industrialisation :

9.1 Les VCS

- Définition.
- Les différents types de VCS.
- Focus sur GIT.
- Les workflows.

Définition

Un **logiciel de gestion de versions** (ou **VCS** en anglais, pour ***V**ersion **C**ontrol **S**ystem*) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.

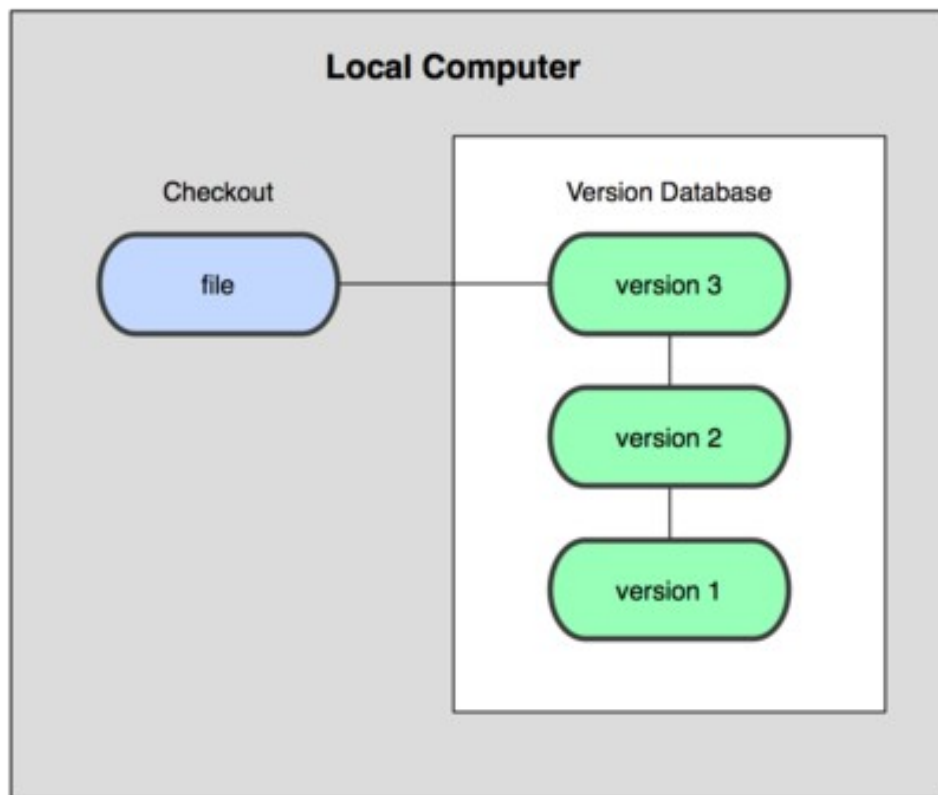
Il existe aussi des logiciels et services de gestion de versions **décentralisé** (distribué) (ou **DVCS** en anglais, pour ***D**istributed **V**ersion **C**ontrol **S**ystem*). Git et Mercurial sont deux exemples de logiciel de gestion de versions décentralisé et sont disponibles sur la plupart des systèmes Unix et Windows.

Les logiciels de gestion de versions sont utilisés notamment en développement logiciel pour conserver le code source relatif aux différentes versions d'un logiciel. (© *Wikipedia*)

Répond à des besoins

- **Partager** des sources en équipe.
- **Partager** des sources avec tout le monde (open source).
- **Historique** des sources.

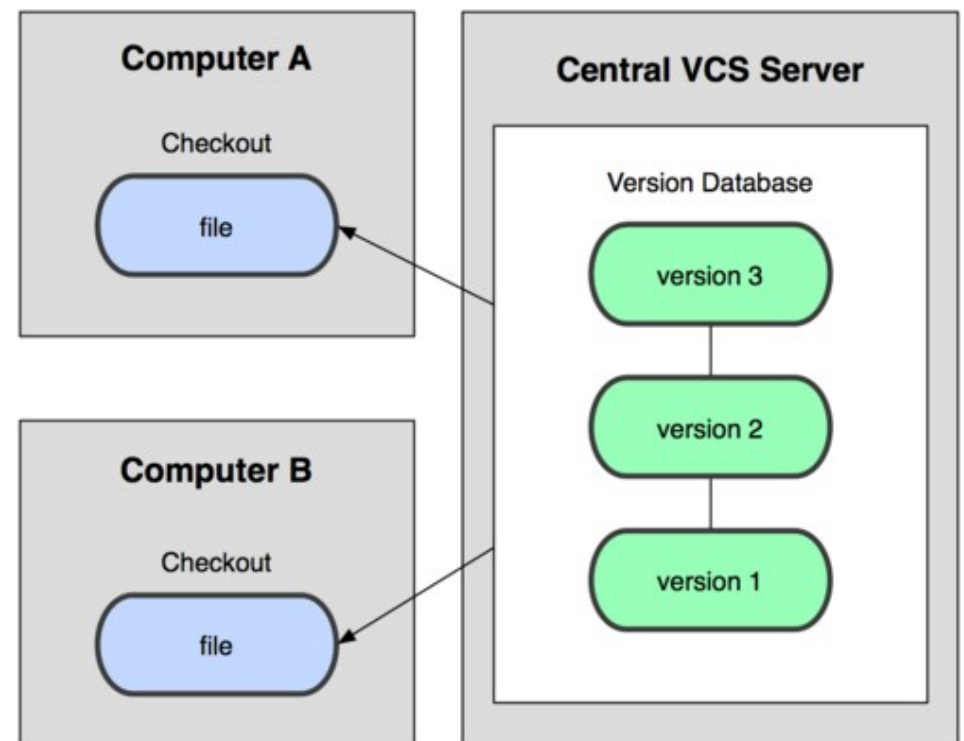
Type #1 : Gestion de versions locale



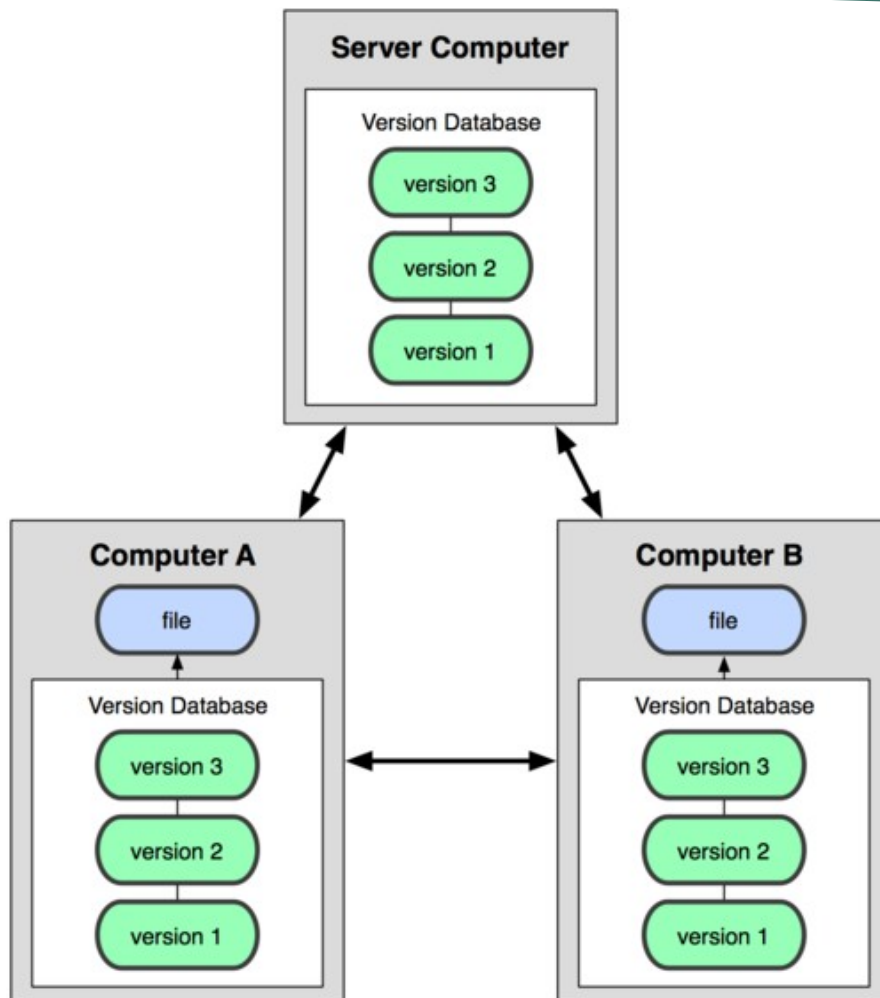
- Exemple : [RCS](#)
- Système plus répandu (heureusement) de nos jours.

Type #2 : Gestion de versions centralisée

- VSC.
- Exemple [SVN](#).
- Tout passe par le serveur central : [SPOF](#).
- Les versions locales (et donc poussées) sont anarchiques et incohérentes (possibilité d'*updater* une arborescence partielle).



Type #3 : Gestion de versions décentralisée



- DVSC.
- Exemple [GIT](#), [Hg](#), [Bz](#), etc.
- Serveur local clone du distant (rapidité).
- Les versions locales (et donc poussées) sont cohérentes car on ne peut pousser si le dépôt n'est pas en *phase*.
- Puissant donc complexe à utiliser.
- Vraie gestion de branche.

Focus sur GIT : Historique

- Une histoire liée au noyau **Linux** et à **Linus Torvalds**.
- En **2002**, le projet du noyau Linux commence à utiliser un DVCS propriétaire : BitKeeper.
- En **2005**, clash entre la communauté & BitKeeper qui passe sur un modèle payant.
- Suite à ça, toujours en **2005**, La réponse de la communauté & de Linus fut la **création de GIT**.

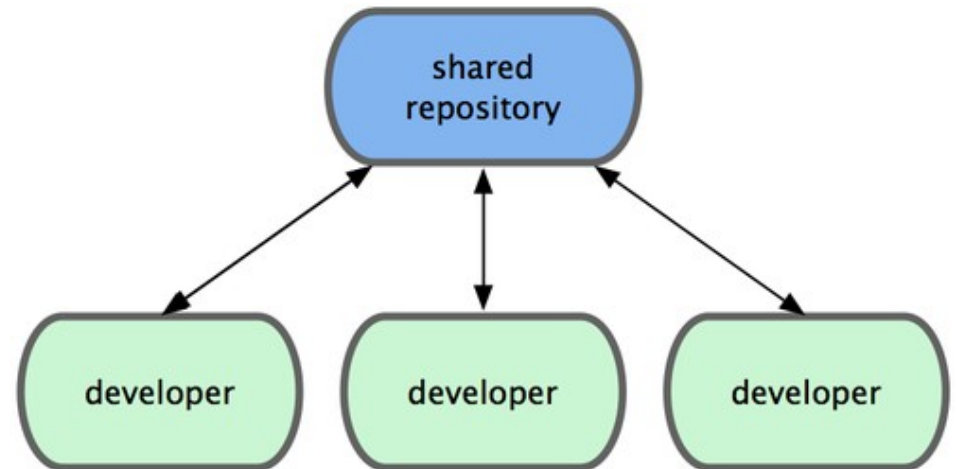
Focus sur GIT : ADN

Le cahier des charges :

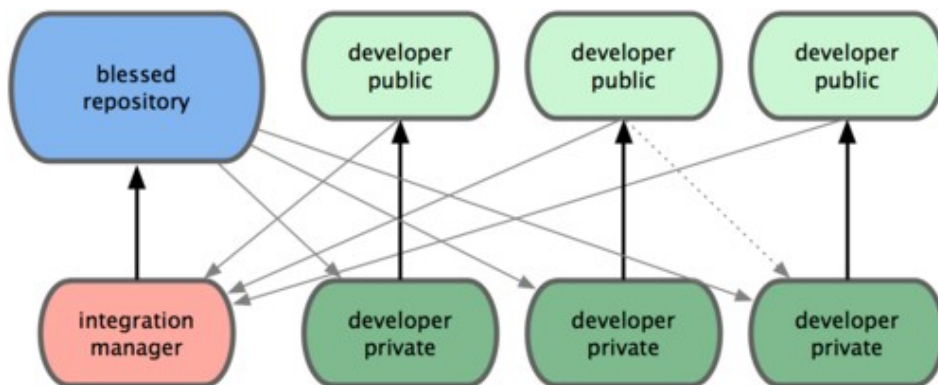
- **Vitesse.**
- Conception **simple.**
- Support pour les développements non linéaires (**milliers de branches** parallèles).
- Complètement **distribué.**
- Capacité à **gérer efficacement des projets d'envergure** tels que le noyau Linux (vitesse et compacité des données).

Workflow #1 : Mode distribué

- Le plus utilisé car le plus simple.
- Proche du VSC.
- Tous les développeurs *push* sur le même dépôt distant.



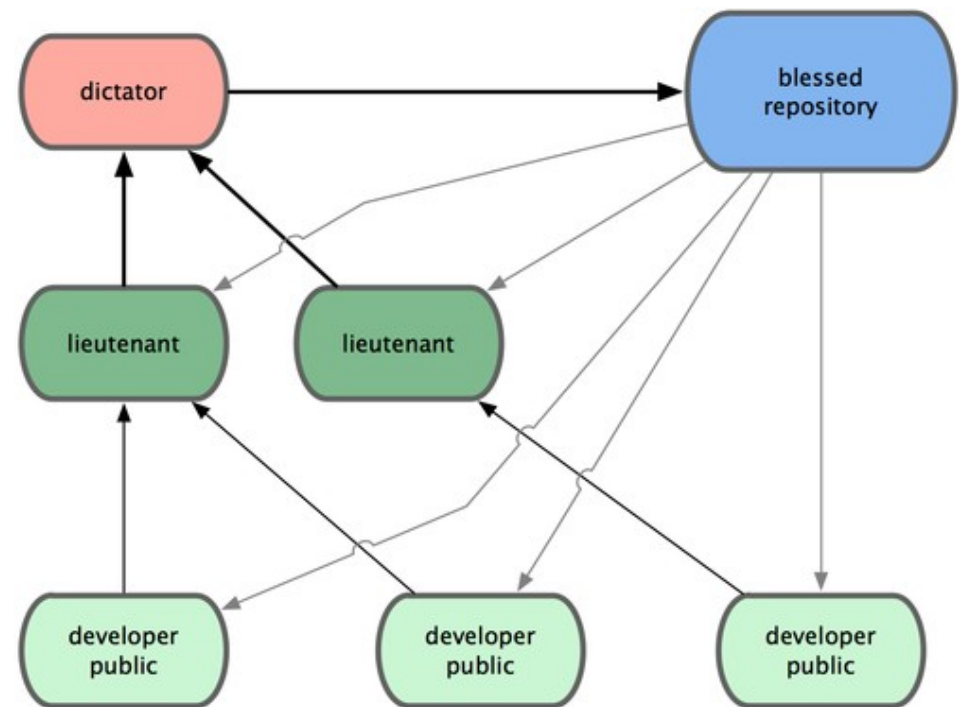
Workflow #2 : Gestionnaire d'intégration



- Approche proche du principe de *Pull Request* de [GitHub](#).
- 1 gardien du temple vérifie les *PR* avant de les *merger* dans le dépôt officiel.

Workflow #3 : Dictateur

- La méthode la plus efficace pour gérer les gros projets subdivisés en plusieurs sous-projets eux même subdivisés en d'autres projets...
- Exemple Linux.



Workflow #4 : GIT-Flow

- Plus une méthode de gestion de branches qu'un workflow.
- Utilisation de branches sémantiques :
 - **master** : code de production. On ne développe rien sur cette branche.
 - **develop** : code prêt pour le *merge* pour la nouvelle release. On peut également corriger, améliorer de petites fonctionnalités.
 - **feature/xxx** : développement des nouvelles fonctionnalités. Une fois terminé, on *merge* les changements dans *develop*.
 - **release/vx.y.z** : Préparation d'une nouvelle release. Un *merge* sera alors effectué dans *master*.
 - **hotfix** : permet de réparer tout de suite un bug critique en production.



Aller plus loin...

Git, GitHub & social coding

Par Guillaume KULAKOWSKI

Expert Technique en Solutions Open Source @ CGI.

Ambassadeur & Packageur @ FedoraProject.

Blogueur @ blog.kulakowski.fr.



9 - Outillage & industrialisation :

9.2 - Plateforme d'Intégration Continue

- Définition
- Exemple
- Outil de qualimétrie
- Métriques

Définition

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Le concept a pour la première fois été mentionné par Grady Booch et se réfère généralement à la pratique de l'extreme programming. Le principal but de cette pratique est de détecter les problèmes d'intégration au plus tôt lors du développement. De plus, elle permet d'automatiser l'exécution des suites de tests et de voir l'évolution du développement du logiciel.

L'intégration continue est de plus en plus utilisée en entreprise afin d'améliorer la qualité du code et du produit final.(© *Wikipedia*)

Concrètement ?

Un PIC ça sert à quoi ?

- A construire (build, compilation, etc.).
- A lancer des tests de qualité (QA, qualimétrie).
- A lancer des tests d'intégration (TI) automatisés.
- A lancer des tests de non-régression (TNR) automatisés.
- A construire un paquet (et même à le déployer).
- A lancer des outils divers (tests, sondes, etc.).
- Etc...

Le rôle central de l'ordonnanceur

Comme le montre le choix du logo de Jenkins, une PIC ne fait pas grande chose, elle se base sur un ordonnanceur pour faire le travail.

- [Phing](#) / [Composer](#) : ordonnanceur PHP.
- [ANT](#) / [Maven](#) : ordonnanceur JAVA.
- Etc...

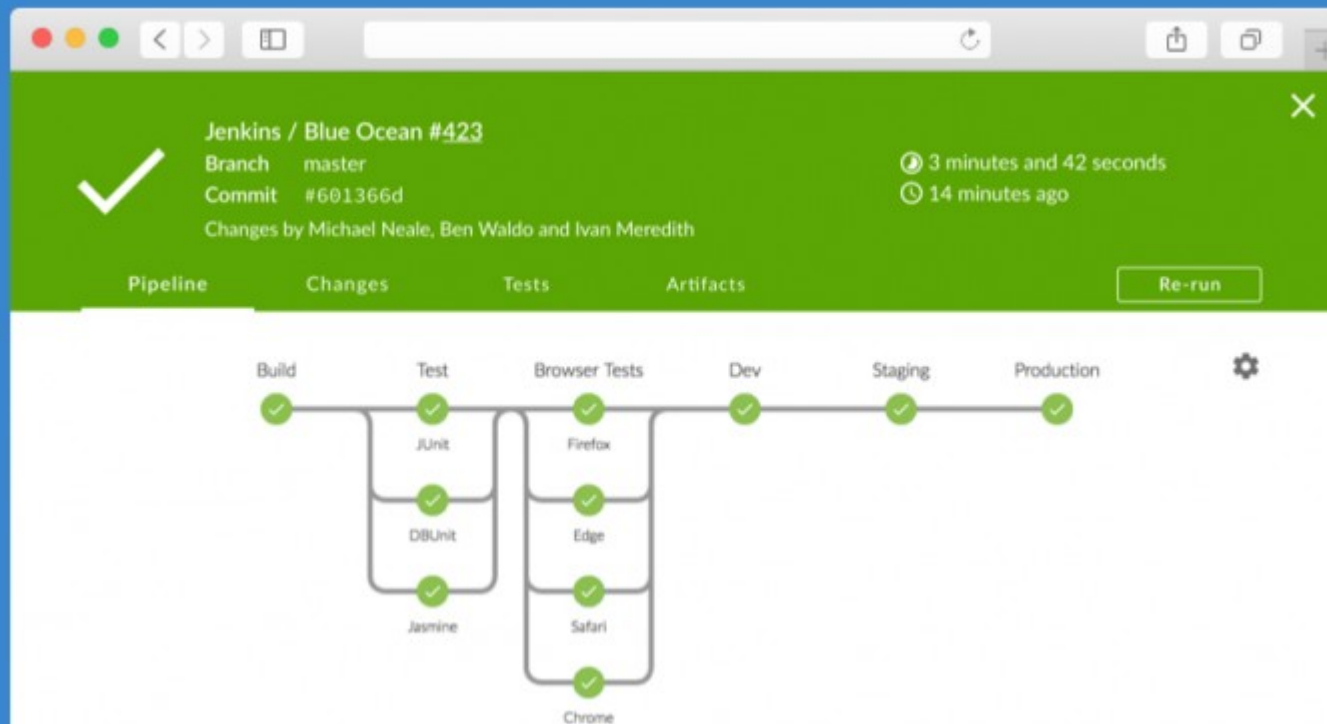


C'est eux qui au travers de leur langage de script vont faire les appels aux outils (tests, qualimétrie, etc.).

Approche pipeline (Jenkins 2 / Blue Ocean)



Steps parallèles des pipelines (Jenkins 2 / Blue Ocean)



Présentation de Blue Ocean

Autre exemple de PIC : Travis

Travis CI [Blog](#) [Status](#) [Help](#)

Guillaume Kulakowski



Search all repositories

My Repositories +

✓ llaumgui/seedboxsync #29

⌚ Duration: 14 sec
📅 Finished: 11 days ago

✓ llaumgui/CheckToolsFramework #19

⌚ Duration: 4 min 31 sec
📅 Finished: 3 months ago

✓ llaumgui/JunitXml #24

⌚ Duration: 4 min 23 sec
📅 Finished: 3 months ago

✓ llaumgui/jquery-async-gravatar #35

⌚ Duration: 34 sec
📅 Finished: 4 months ago

llaumgui / JunitXml build passing

[Current](#) [Branches](#) [Build History](#) [Pull Requests](#)

⚙ Settings

✓ develop Fix some comments errors

24 passed

👤 Commit e3ca67b
👤 Compare 3f73b02..e3ca67b
⌚ ran for 4 min 23 sec
📅 3 months ago

👤 Guillaume Kulakowski authored and committed

Build Jobs

✓ # 24.1	👤	</> PHP: 5.3	📦 no environment variables set	⌚ 38 sec
✓ # 24.2	👤	</> PHP: 5.4	📦 no environment variables set	⌚ 37 sec
✓ # 24.3	👤	</> PHP: 5.5	📦 no environment variables set	⌚ 39 sec
✓ # 24.4	👤	</> PHP: 5.6	📦 no environment variables set	⌚ 38 sec

Allowed Failures ?

✓ # 24.5	👤	</> PHP: 7	📦 no environment variables set	⌚ 35 sec
✓ # 24.6	👤	</> PHP: hhvm	📦 no environment variables set	⌚ 38 sec
✓ # 24.7	👤	</> PHP: nightly	📦 no environment variables set	⌚ 38 sec

.travis.yml

40 lines (34 sloc) 1.7 KB

Raw Blame History

```

1 sudo: false
2 language: php
3 php:
4   - 5.3
5   - 5.4
6   - 5.5
7   - 5.6
8   - 7.0
9   - hhvm
10  - nightly
11 matrix:
12   allow_failures:
13     - php: 7.0
14     - php: hhvm
15     - php: nightly
16
17
18 before_install:
19   # Update composer binary
20   - composer self-update
21
22 before_script:
23   # Install require & require-dev
24   - composer install --prefer-source
25   # Install build tools but no dev tools
26   - composer require squizlabs/php_codesniffer codeclimate/php-test-reporter:dev-master
27
28 script:
29   - phpunit --configuration phpunit.xml.dist --coverage-text --coverage-clover build/logs/clover.xml
30   - ./vendor/bin/phpcs --standard=PSR2 src/*
31
32 after_script:
33   # TestReporter seem desn't work with these version og PHP
34   - if [ $TRAVIS_PHP_VERSION = '5.6' ] && [ $TRAVIS_BRANCH = 'master' ] && [ $TRAVIS_PULL_REQUEST = 'false' ]; then ./venc
35   # ApiGen generation
36   - if [ $TRAVIS_PHP_VERSION = '5.6' ] && [ $TRAVIS_BRANCH = 'master' ] && [ $TRAVIS_PULL_REQUEST = 'false' ]; then sh ger
37
38 env:
39   global:
40     secure: mKk20bdLrNU56z9ocgiMBMgjsxeNZYVMwmaTrRhL1/2UKDHLL56PqW3DJSBcwehoZy6fWPVtpD5X74124FqgWK+q/XkquWVF9EY00ZKCE9d5A/

```


Métrique : outil de type « Checkstyle »

- Exemple : [Checkstyle](#) (JAVA), [PHPCS](#) (PHP), etc.
- Contrôle la cohérence du style d'un code source :
 - Choix des noms de classe et fonction,
 - Indentation,
 - Nom des fichiers,
 - Namespacing,
 - Etc...

Métrique : duplication de code

- Exemple [PHPCPD](#), [PMD](#)
- Permet de détecter les copier/coller et les manques de factorisation au sein d'un projet.

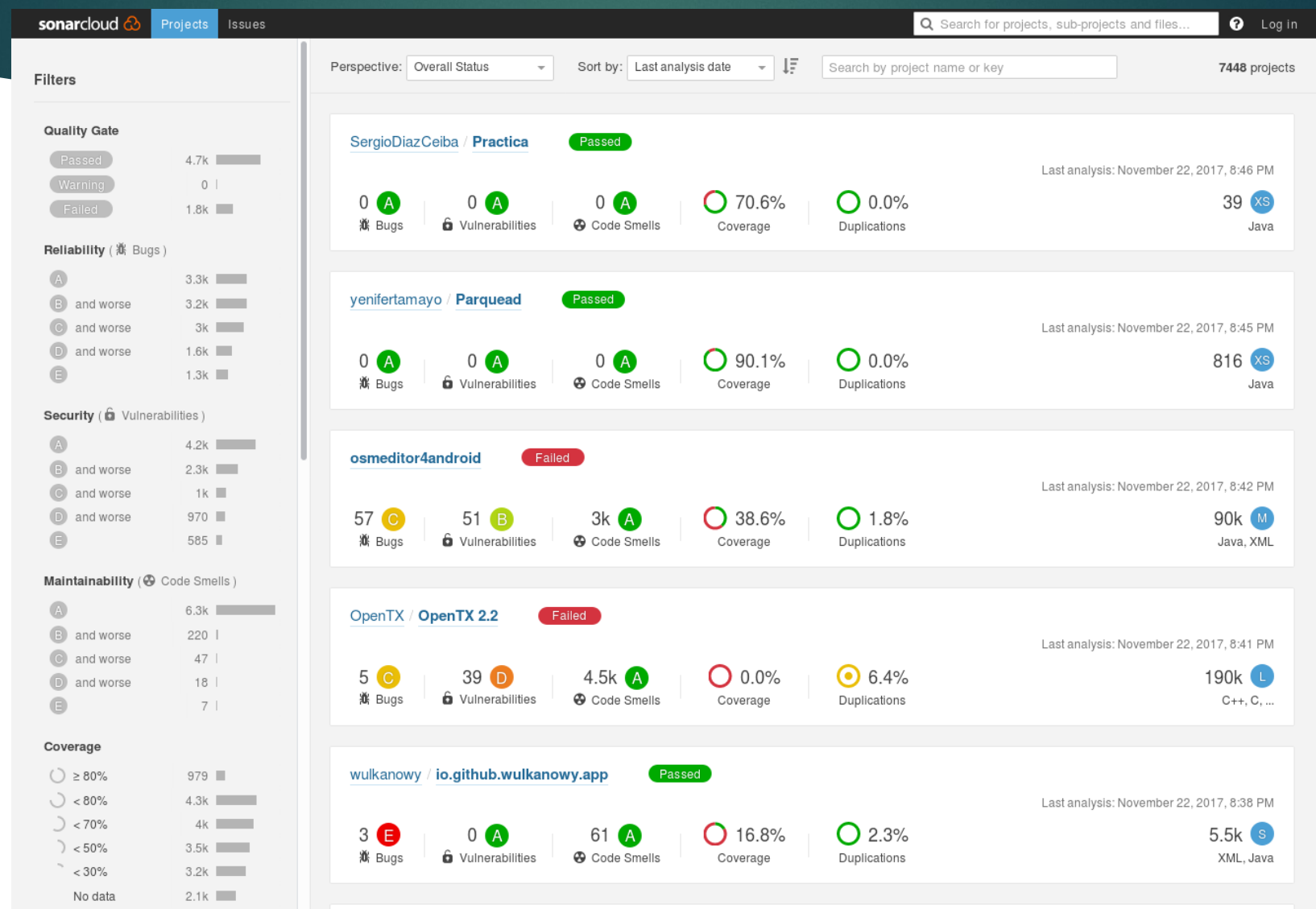
Métrique : complexité

- Exemple : [PMD](#), [Pdepend](#), etc.
- Détection de code mort, fonctions et/ou variable inutilisé.
- Notion de complexité cyclomatique
- Etc...

Métrique : divers

- Accessibilité (ex : [Asqatasun](#)).
- Ligne de code ([phploc](#)).
- Performance Web ([YSlow](#), [GTmetrix](#)).
- Bien sûr tous les outils de TU / TI / TNR ([CasperJS](#), [Selenium](#)).

Exemple d'outil de qualimétrie : Sonar



L'expert technique / Lead developer

- Son rôle est primordiale,
- Il doit suivre tous ces indicateurs,
- Il doit faire des revues de code.

9 Outillage & industrialisation :

9.3 Déploiement continu

- Définition
- L'approche DevOps

Définition

Le déploiement continu, prolongement de l'intégration continue, est une pratique visant à réduire le plus possible le temps de cycle, le temps écoulé entre l'écriture d'une nouvelle ligne de code et l'utilisation réelle de ce même code par des utilisateurs finaux.

L'équipe s'appuie sur une infrastructure qui automatise l'ensemble des étapes de déploiement (ou "mise en production"), de sorte qu'après chaque intégration qui se solde par des tests passant avec succès, l'application en production est mise à jour (*© institut-agile*).

Discours d'un autre temps...



Ne pousse pas en prod le vendredi, ça risque de planter très fort.

Don't push your apps live on Fridays, they might crash very hard.

Freitags nichts in der Produktion abgeben, es könnte schnell abstürzen.

Non spingere troppo il venerdì, la produzione rischia di piantarsi !

¡No pongas en línea un viernes, puede trabarse muy duro!

CommitStrip.com

Comment mon vendredi soir devait se passer



Comment mon vendredi soir s'est passé



CommitStrip

DevOps c'est quoi ?

- Devops est un mouvement visant à **réduire la friction organisationnelle** entre les "devs" (chargés de faire évoluer le système d'information) et les "ops" (chargés d'exploiter les applications existantes).
Ce que l'on pourrait résumer en **travailler ensemble pour produire de la valeur pour l'entreprise**. Dans la majorité des entreprises, la valeur sera économique mais pour d'autres elle sera sociale ou morale.
- Les initiateurs du mouvement sont principalement des **administrateurs systèmes férus de méthode agile**, évoluant en entreprise. Certains avaient même lancé l'**agile-infrastructure**.
Le principe de base de devops part du constat suivant : **si une équipe d'exploitation est primée sur la stabilité du système alors que l'équipe de développement est récompensée à chaque nouvelle fonctionnalité livrée, il est évident que ces deux équipes vont se retrouver en conflit perpétuel.**
- Plus généralement, organiser une entreprise comme un ensemble d'équipes que l'on va objectiver indépendamment les unes des autres avec des indicateurs spécifiques à chaque équipe, va générer des optimums locaux et des guerres entre équipes. Ce qui globalement pour l'entreprise n'est pas la meilleure chose.
- Promouvoir la coopération, industrialiser et motiver chacun sans noyer l'ensemble dans la masse n'est pas une mince affaire. Le mouvement devops en plus de réfléchir à une nouvelle organisation de l'entreprise, vise à trouver des techniques et des outils pour favoriser cette coopération.

L'outillage du DevOps

- Sans passer par des concepts de conteneurs (ex : [Docker](#)) qui permettent de pousser une infrastructure complète, des outils comme [Capistrano](#) ou [Mina](#) permettent de pousser du code en production.
- Pousse sur **différents serveurs en même temps** (plus de facteur humain).
- Effectue des tâches post-update (ex : vidage de cache, *warmup*, *updater*, etc.).
- Autorise le *rollback*.
- Autorise le *staging*.
- Peut-être connecté à une PIC (la boucle est bouclée).

9 Outillage & industrialisation :

9.4 Docker

- C'est quoi ?
- Pourquoi ?
- Docker VS VM

Docker c'est quoi ?

- **Docker c'est quoi ?**

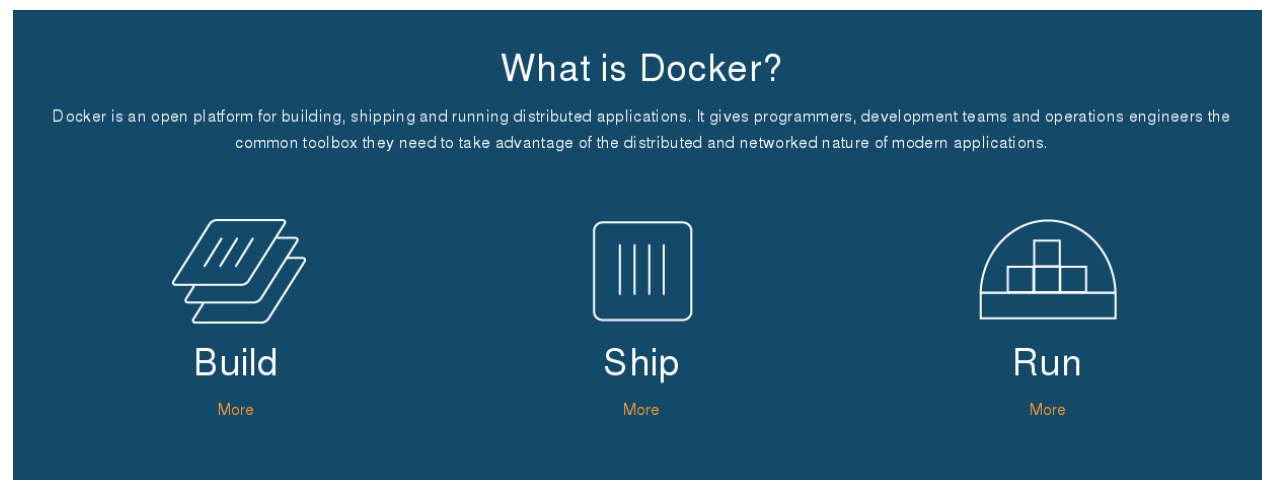
Docker est une solution permettant d'exécuter un ou plusieurs logiciels dans des environnements séparés (conteneurs) pouvant communiquer entre eux.

- **Un conteneur c'est quoi ?**

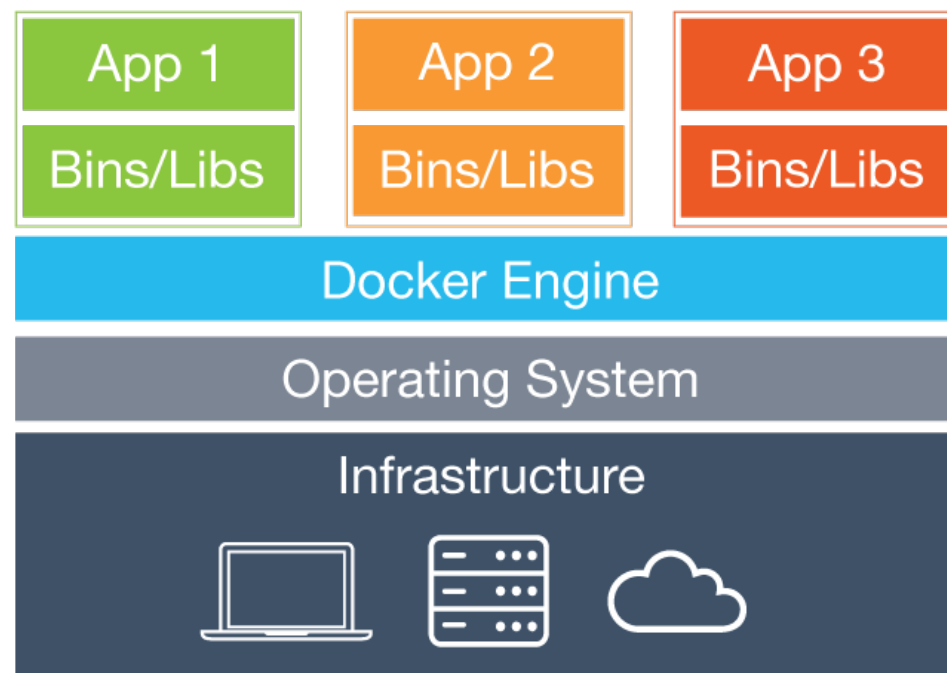
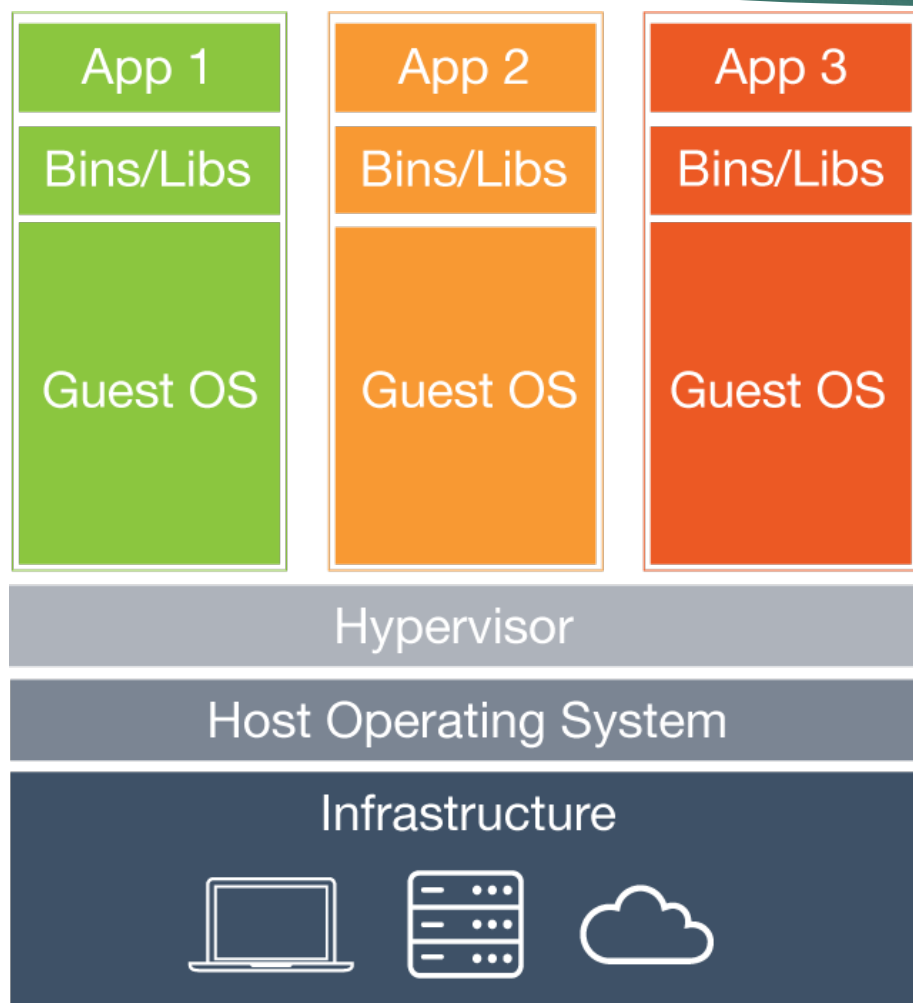
- Un espace isolé permettant d'exécuter des processus. Une bonne pratique est de faire en sorte qu'**1 conteneur corresponde à 1 application**.
- Ça comprend tout ce que comprend une VM : root, IP, etc...

Docker pourquoi ?

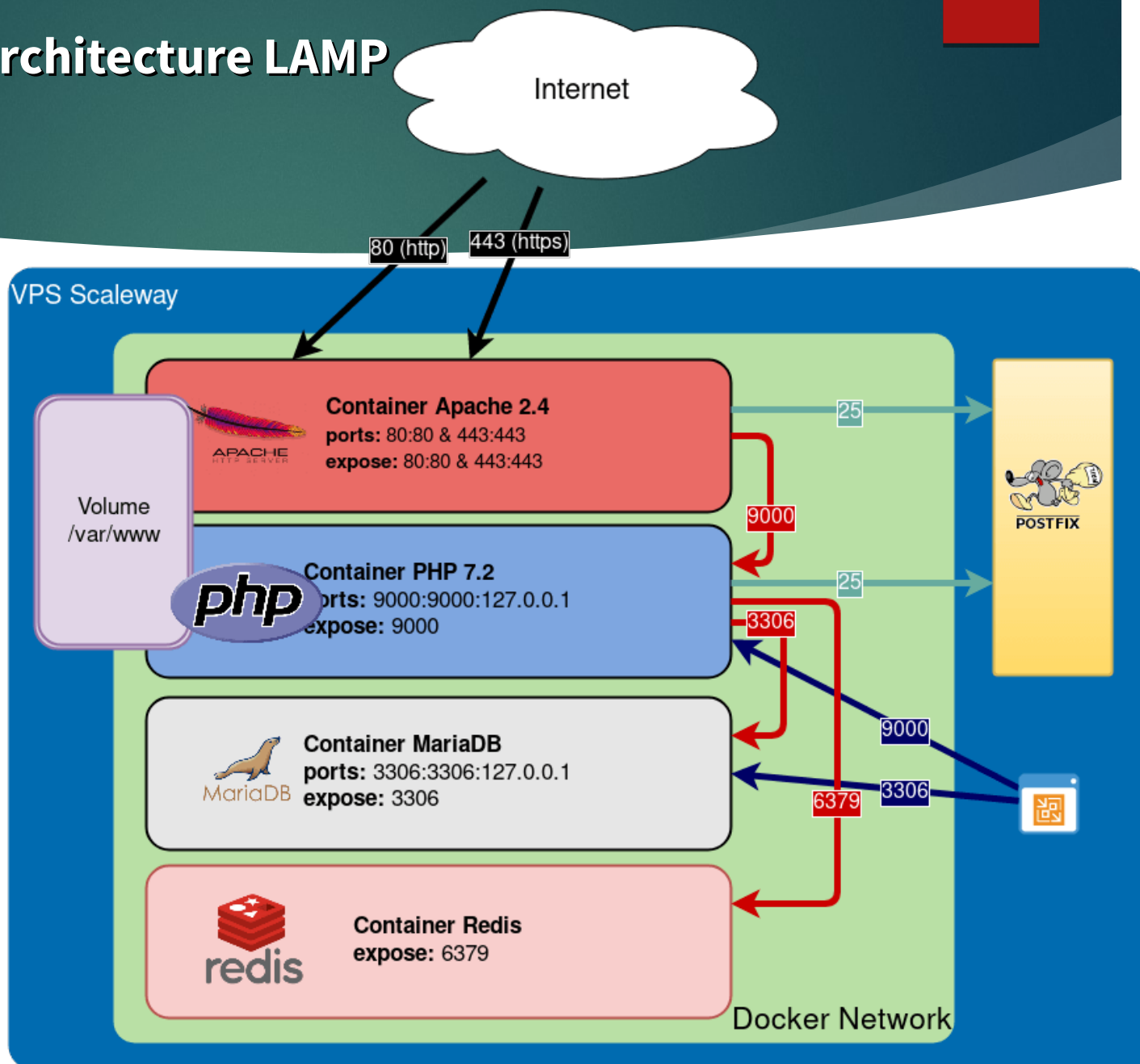
- Vous ne développez pas une application qui sera poussée en prod. Vous développez dans un container qui sera poussé (après resizing et test) en prod' !
- On gagne en productivité, en time-to-market.



Virtualization VS Container



Anatomie d'une architecture LAMP



En résumé

- Docker est une solution de virtualisation légère basée sur des conteneurs (containers).
- Un conteneur est un environnement d'exécution **isolé** (vraiment) avec ses **propres ressources**.
- Tous les conteneurs utilisent le même noyau qui est celui de la machine hôte.
- Docker se basait uniquement sur le noyau Linux et ses technologies LXC, namespaces et cgroups puis sur libcontainer depuis la version 0.9.

Les outils de Docker

- **Docker Hub** : un [hub d'échange](#) d'images Docker.
- **Docker Desktop** : un GUI pour Docker sous Mac & Windows
- **Docker Compose** : pour définir des compositions de plusieurs containers.
- Etc...

Aller plus loin...

Introduction à Docker

Par Guillaume KULAKOWSKI

Expert Technique en Solutions Open Source @ [CGI](#).

Ambassadeur & Packageur @ [FedoraProject](#).

Blogueur @ blog.kulakowski.fr.



Annexes

Guide de survit de bin/console

- Vider le cache :

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


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.