Installing and Configuring Symfony

Installing the Symfony Installer

requires PHP 5.4 or higher. If you use PHP 5.3, you cannot use it.

```
################## Linux and Mac OS X Systems
$ sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
$ sudo service apache2 restart
$ source ~/.bashrc
```

Creating the Symfony Application

```
################## Linux and Mac OS X Systems
$ symfony new my_project_name
# use the most recent version in any Symfony branch
$ symfony new my_project_name 2.6
# use a specific Symfony version
$ symfony new my_project_name 2.6.5
# Symfony LTS version
$ symfony new my_project_name lts
```

Running the Symfony Application

```
################## Linux and Mac OS X Systems
$ cd my_project_name/
$ php app/console server:run
# segundo plano
$ php app/console server:start
```

Open your browser and access the http://localhost:8000/

The server:run command is only suitable while developing the application.

Symfony applications on production servers, you'll have to configure your Apache or Nginx web server as explained in Configuring a Web Server.

When you are finished working on your Symfony application, you can stop the server with

```
# cuando segundo plano
$ php app/console server:stop
```

Checking Symfony Application Configuration and Setup

Access the following URL http://localhost:8000/config.php

```
################## Linux: Debian
$ sudo emacs -nw /etc/php5/cli/php.ini
date.timezone =America/Lima
$ sudo apt-get install php5-intl
$ sudo service apache2 restart
```

Permissions here.

Installing the Symfony Demo Application

```
################## Linux and Mac OS X Systems
$ symfony demo
```

Soluttions

```
$ sudo apt-get install sqlite3
$ sudo apt-get install php5-sqlite
```

Installing a Symfony Distribution

Symfony project packages "distributions", which are fully-functional applications that include the Symfony core libraries, a selection of useful bundles, a sensible directory structure and some default configuration. In fact, when you created a Symfony application in the previous sections, you actually downloaded the default distribution provided by Symfony, which is called Symfony Standard Edition.

- The Symfony CMF Standard Edition is the best distribution to get started with the Symfony CMF project, which is a project that makes it easier for developers to add CMS functionality to applications built with the Symfony Framework.

- The Symfony REST Edition shows how to build an application that provides a RESTful API using the FOSRestBundle and several other related bundles.

Beginning Development

Be sure to also check out the Cookbook, which contains a wide variety of articles about solving specific problems with Symfony.

Create your First Page in Symfony

it's an HTML page or a JSON endpoint - is a simple two-step process

1. *Create a route*: A route is the URL and points to a controller
2. *Create a controller:* A controller is the function

Symfony Response object, which can hold HTML content, a JSON string or anything else

Creating a Page: Route and Controller

```php
// src/AppBundle/Controller/LuckyController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;

class LuckyController extends Controller
{
    /**
     * @Route("/lucky/number")
     */
    public function numberAction()
    {
        $number = rand(0, 100);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

http://localhost:8000/app_dev.php/lucky/number

Note: If you setup a proper virtual host in *Apache or Nginx*

The `@Route` above `numberAction()` is called an *annotation* and it defines the URL

What's the `app_dev.php` in the URL? you're executing Symfony through a file - web/app_dev.php - that boots it in the dev environment. For production, you'll use clean URLs `http://localhost:8000/lucky/number` - that execute a different file – `app.php`

Creating a JSON Response

The `Response` object you return can contain HTML, JSON or even a binary file like an image or PDF

```
// src/AppBundle/Controller/LuckyController.php
// ...

class LuckyController extends Controller
{
    // ...

    /**
     * @Route("/api/lucky/number")
     */
    public function apiNumberAction()
    {
        $data = array(
            'lucky_number' => rand(0, 100),
        );

        return new Response(
            json_encode($data),
            200,
            array('Content-Type' => 'application/json')
        );
    }
}
```

http://localhost:8000/app_dev.php/api/lucky/number

You can even shorten this with the handy `JsonResponse`

```
// src/AppBundle/Controller/LuckyController.php
// ...

// --> don't forget this new use statement
use Symfony\Component\HttpFoundation\JsonResponse;

class LuckyController extends Controller
{
    // ...

    /**
     * @Route("/api/lucky/number")
     */
    public function apiNumberAction()
    {
        $data = array(
            'lucky_number' => rand(0, 100),
        );

        // calls json_encode and sets the Content-Type
header
        return new JsonResponse($data);
    }
}
```

Dynamic URL Patterns: /lucky/number/{count}

```
// src/AppBundle/Controller/LuckyController.php
// ...

class LuckyController extends Controller
{

    /**
     * @Route("/lucky/number/{count}")
     */
    public function numberAction($count)
    {
        $numbers = array();
        for ($i = 0; $i < $count; $i++) {
            $numbers[] = rand(0, 100);
        }
        $numbersList = implode(', ', $numbers);

        return new Response(
            '<html><body>Lucky numbers: '.$numbersList.'</body></html>'
        );
    }

    // ...
}
```

http://localhost:8000/app_dev.php/lucky/number/7

You can get the value of any `{placeholder}` in your route by adding a `$placeholder` argument to your controller.

Rendering a Template (with the Service Container)

Symfony comes with Twig: a templating language that's easy, powerful and actually quite fun.

```
// src/AppBundle/Controller/LuckyController.php
// ...

class LuckyController extends Controller
{
    /**
     * @Route("/lucky/number/{count}")
     */
    public function numberAction($count)
    {
        // ...
        $numbersList = implode(', ', $numbers);

        $html = $this->container->get('templating')->render(
            'lucky/number.html.twig',
            array('luckyNumberList' => $numbersList)
        );

        return new Response($html);
    }

    // ...
}
```

For now, you just need to know that it holds a lot of objects, and you can `get()` any object by using its nickname, like `templating` or `logger`. The `templating` service is an instance of [`TwigEngine`] and this has a `render()` method.

But this can get even easier! By extending the `Controller` class, you also get a lot of shortcut methods, like `render()`:

```
// src/AppBundle/Controller/LuckyController.php
// ...

/**
 * @Route("/lucky/number/{count}")
 */
public function numberAction($count)
{
    // ...

    /*
    $html = $this->container->get('templating')->render(
        'lucky/number.html.twig',
        array('luckyNumberList' => $numbersList)
    );

    return new Response($html);
    */

    // render: a shortcut that does the same as above
    return $this->render(
        'lucky/number.html.twig',
        array('luckyNumberList' => $numbersList)
    );
}
```

Create the Template

```
{# app/Resources/views/lucky/number.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}
    <h1>Lucky Numbers: {{ luckyNumberList }}</h1>
{% endblock %}
```

off the basics: like how the `{{ variableName }}` syntax is used to print something.

The `{% extends 'base.html.twig' %}` points to a layout file, The `{% block body %}` part uses Twig's *inheritance system* to put the content into the middle of the `base.html.twig` layout.

Exploring the Project

`app/,` Contains things like configuration and templates. Basically, anything that is *not* PHP code goes here.

`src/,` Your PHP code lives here.

The `app/` directory also holds a few other things, like the cache directory `app/cache/`, the logs directory `app/logs/` and `app/AppKernel.php`, which you'll use to enable new bundles

`src/AppBundle.` A bundle is like a "plugin" and you can find open source bundles and install them into your project.

`vendor/,` ibraries and bundles are downloaded here by the Composer package manager.

`web/,` This is the document root for the project and contains any publicly accessible files, like CSS, images and the Symfony front controllers that execute the app (`app_dev.php` and `app.php`).
How to Override Symfony's default Directory Structure.

Application Configuration

Symfony comes with several built-in bundles (open your `app/AppKernel.php` file) and you'll probably install more. The main configuration file for bundles is `app/config/config.yml`:
Or, to get a big example dump of all of the valid configuration under a key, use the handy `app/console` command

```
$ app/console config:dump-reference framework
```

What's Next?

- *Controller*
- *Routing*
- *Creating and Using Templates*

learn about the *service container*, the *form system*, using *Doctrine* (if you need to query a database) and more!

There's also a *Cookbook* *packed* with more advanced "how to" articles to solve *a lot* of problems.