Instalación del entorno de trabajo

Costasur

Capítulo 1
Requisistos previos

En este primer capítulo, vamos a ver que necesitamos tener ya instalado en nuestra máquina local para poder empezar a instalar el proyecto de Costasur.

1.1 Entorno LAMP

En primer lugar necesitaremos tener montando un entorno LAMP, para poder hacer pruebas de forma local en nuestra máquina. Bastará con tener la versión de PHP 5.3 y Apache2, puesto que el sistema de base de datos lo montamos en el servidor local de la oficina, el cual tendrá SQL para las bases de datos relacionales y MongoDB para las bases de datos no relacionales.

1.2 Control de versiones GIT

En segundo lugar necesitaremos tener instalado un control de versiones, en nuestro caso, GIT.

En el caso de que no lo tengamos instalado, ejecutamos la siguiente lína en nuestar terminal y posteriormente editamos el archivo .gitconfig en nuestro directorio home y lo configuramos con nuestro usuario.

```
$ apt-get install git
```

Fichero de configuración

```
[user]
    name = Usuario
    email = email@costasur.com
[alias]
    co = checkout
    st = status
    ci = commit
    br = branch
    undo = reset --hard HEAD
    last = log -1 HEAD
    ramas = remote show
    save  = stash
    back = stash pop
[color]
    ui = true
```

1.3 DNSMASQ

En tercer lugar necesitaremos tener instalado y configurado el servicio DNSMASQ. Si no lo tenemos instalado, lo instalamos con el siguiente comando y configuramos el fichero

/etc/dnsmasq.conf

```
$ apt-get install dnsmasq
```

Añadir al fichero de configuración

```
server = 127.0.0.1
server = 8.8.8.8
address=/ra/192.168.1.31
address=/tratewolocal.com/127.0.0.1
```

1.4 FLUENTD

En costasur vamos a generar logs directamente sobre base de datos, de tal forma que podamos monitorizar nuestras aplicaciones sin necesidad de tener acceso directo al sistema de ficheros. Hemos escogido fluent porque es rápido y guarda la información en formato JSON, al igual que nuestro gestor de base de datos.

En un servidor será interesante seguir los pasos del apéndice 1: Optimizando Fluent

1.4.1 Instalando fluentd (tdagent)

En debian, simplemente ejecuta esto:

```
$ curl -L http://toolbelt.treasure-data.com/sh/install-ubuntu-lucid.sh | sh
```

Ya debería estar instalado y ejecutándose.

1.4.2 Configurando para logs de Apache

Editamos el fichero en `/etc/td-agent/td-agent.conf`

```
<match mongo.*.*>
  type mongo
  database apache_quigon
  collection access
  host ra
  port 27017
  flush_interval 10s
</match>
```

```
<source>
  type tail
  format apache2
  path /var/log/apache2/costa.accesss.log --> (tu fichero de log)
  pos_file /var/log/td-agent/apache2.access_log.pos
  tag mongo.apache.access
</source>
```

1.4.2.1 Los permisos de Apache

Hay que dar permisos de lectura al usuario/grupo de td-agent (fluentd) para que pueda leer los logs de apache. Nos vamos al directorio /var/log. De paso nos podemos dar permisos a nosotros mismos para leerlos sin tener que hacer sudo...

```
$ sudo setfacl -dR -m g:td-agent:rwx
```

```
    -m u:td-agent:rwx -m u:`whoami`:rwx apache2
```

Reiniciamos el servicio con un elegante

```
$ sudo service td-agent restart</code></pre>
```

1.5 Resumen

Llegados a este punto, tenemos ya montando nuestro control de versiones GIT, con el servidor local LAMP, Fluentd y el servidor de DNS local.

A continuación en el siguiente capítulo, tendremos que clonar el repositorio de Costasur y posteriormente, configurar el proyecto en Symfony, donde tendremos que configurar apache, para que tratewolocal.com apunte a este proyecto, instalar el interpretador de mongo para php e instalar composer.

Capítulo 2
Instalación

En primer lugar, tenemos que tener pensado donde vamos a alojar nuestro proyecto, en toda la documentación, hemos optado por el directorio `/var/www/`

2.1 Clonar Repositorio

En primer lugar nos situamos en el directorio donde alojaremos el proyecto y clonamos el repositorio con la siguiente línea. Mencionar que actualmente el repositorio está en nuestro servidor local de la oficina, aunque si se variase de ubicación, lo único que cambiaría seria la dirección física de la que nos bajemos el repositorio.

```
cd /var/www/
sudo git  clone ssh://amon@ra/~/costasur.git
```

Tras haber clonado el repositorio, nos debería de haber quedado la siguiente estructura de directorios, donde más adelante explicaremos que contiene cada uno de ellos:

```
/var/www/costasur/
    app
    src
    vendor
    web
    (*)
```

2.2 Configurar sites-availables

El siguiente paso sería que http://www.tratewolocal.com apunte a nuestro proyecto local, por tanto para esto necesitamos crear un sites-available nuevo.

```
sudo touch /etc/apache2/sites-available/tratewolocal
```

Configuramos el sites-available de la siguiente forma.

```
<Directory /var/www/costasur/web>
```

```
    AllowOverride all
    Allow from all
    Options FollowSymlinks
</Directory>
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName tratewolocal.com
    ServerAlias documentation.tratewolocal.com
    ServerAlias documentacion.tratewolocal.com
    DirectoryIndex  index.html
    DocumentRoot  /var/www/costasur/web/bundles/costasurdocumentation
    LogLevel warn
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName tratewolocal.com
    DirectoryIndex  app.php
    ServerAlias *.tratewolocal.com
    DocumentRoot  /var/www/costasur/web
    ErrorLog /var/log/apache2/tratewolocal.log
    LogLevel warn
    CustomLog /var/log/apache2/tratewolocal.access.log combined
    RewriteEngine On
</VirtualHost>
```

Una vez que tenemos creado el *sites-availables* , lo añadimos y reiniciamos apache.

```
sudo a2ensite /etc/apache2/sites-available/tratewolocal
sudo service apache2 restart
```

Ya tenemos configurado nuestro servidor local, si probamos a introducir en el navegador www.tratewolocal.com debe de cargar el proyecto sin problemas, pero dará error 500, puesto que aún nos queda por configurar symfony.

2.3 Instalación de *Composer*

Composer es una herramienta para instalar dependencias de PHP, la cual vamos a necesitar para nuestro proyecto de *Symfony* . Mediante esta herramienta como veremos más adelante, instalaremos todos los vendors necesarios en nuestro proyecto.

Para instalarlo abre una consola de comandos, accede al directorio raíz de tu proyecto *Symfony2* (/var/www/costasur) y ejecuta el siguiente comando:

```
$ sudo curl -s https://getcomposer.org/installer | php
```

Si todo ha funcionado bien, en el directorio raíz de tu proyecto *Symfony2* verás un nuevo archivo llamado composer.phar. Para comprobar que se ha instalado correctamente, ejecuta el siguiente comando que muestra el menú de opciones de *Composer*:

```
$ php composer.phar
```

Instalar *Composer* de esta manera es correcto, pero te obliga a realizar una nueva instalación para cada proyecto *Symfony2*. Si tienes muchos proyectos, es mucho mejor que instales *Composer* de forma global en tu ordenador, para que todos los proyectos utilicen la misma versión de *Composer*.

Para instalar *Composer* globalmente, mueve el archivo composer.phar a algún directorio ejecutable del sistema, como por ejemplo:

```
$ sudo mv *Composer*.phar /usr/local/bin/composer
```

Comprueba que todo funciona bien ejecutando el comando *Composer* sin opciones desde cualquier directorio del sistema. La instalación global de *Composer* también facilita su mantenimiento, ya que sólo hay que ejecutar el siguiente comando para actualizar la versión de *Composer* de todos los proyectos:

```
$ sudo composer self-update
```

De esta forma ya tenemos instalado *Composer* correctamente en nuesta máquina.

2.4 Instalación del cliente de Mongo para PHP

Como ya sabemos, en el entorno local, trabajaremos con la base de datos documental Mongo, pero no la tendremos instalada en nuestras máquinas, sino que utilizaremos el servidor local de la oficina.

Debido a esta razón, no necesitamos instalar MongoDB en nuestros equipos, pero necesitaremos el interprete de php, para que php se pueda conectar con la base de datos del servidor local de la oficina.

Para instalarlo hacemos lo siguiente:

```
$ pecl install mongo
```

Por último quedaria añadir al archivo php.ini en /var/php5 para que PHP cargue el driver:

```
$ sudo gedit /etc/php5/apache2/php.ini
$ sudo gedit /etc/php5/cli/php.ini
```

Añadir la siguiente línea:

```
extension=mongo.so
```

Capítulo 3
Configuración proyecto Symfony

3.1 Primeros pasos: verificación

En primer lugar para ver que todo está correcto todo los pasos anteriores, al poner `www.tratewolocal.com/app_dev.php`, tiene que cargar correctamente y dar algun tipo de error, en mi caso, Forbidden, debido a que tal como lo bajamos del repositorio, tiene un permiso de grupo *costasurGroup* y nuestro usuario no está en ese grupo.

3.2 Permisos

En primer lugar cambiamos los permisos del directorio de forma recursiva al directorio costasur, en mi caso el usuario es *obiwan*

```
sudo chown obiwan costasur -R
sudo chgrp www-data costasur -R
```

3.3 Config parameters

En segundo lugar, abrimos nuestro proyecto con nuestro editor de texto o ide y vamos al directorio `/app/config` y veremos que hay un fichero llamado `parameters.yml.sample` lo copiamos a `parameters.yml`.

3.4 Instalar dependencias

En este paso instalaremos las dependencias de *Symfony*. En el proyecto, solo está nuestro código de Costasur, pero los vendors no están en nuestro repositorio, entonces la manera de descargarnos estos vendors es mediante *composer*, que obtiene los vendors que tiene que descargar del archivo composer.json.

```
php composer.phar install
```

Una vez que termine la instalación, ya tendremos todo el código fuente del proyecto al completo.

3.5 Cache de Symfony

Este framework, posee de un propio sistema de cache, que normalmente suele dar muchos quebraderos de cabeza, y el primero de ellos está en la instalación.

Una vez que hemos installado las dependencias mediante composer, el siguiente paso es volver a cargar www.tratewolocal.com/app_dev.php y ver que ya *Symfony* se ejecuta correctamente y nos sale algun tipo de alerta o error.

En este caso el primer error de ejecución es el siguiente.

```
RuntimeException: Failed to write cache file "/var/www/costasur/app/cache/dev/
classes.php".
```

Esto quiere decir que no tenemos permisos para escribir en la caché, por lo tanto vamos a hacer lo siguiente:

Le damos los permisos correctos

```
sudo setfacl -R -m u:www-data:rwx -m u:obiwan:rwx app/cache app/logs
sudo setfacl -dR -m u:www-data:rwx -m u:obiwan:rwx app/cache app/logs
```

Eliminamos el contenido que pudiese tener

```
sudo rm -fr app/cache/*
sudo rm -fr app/logs/*
```

3.6 Generación de assets

Assets son todos aquellos ficheros que tienen que ser accedidos desde el servidor web, en nuestro caso, en primer lugar sería los .css y .js, para esto ejecutamos la siguiente orden y ya debe de funcionar correctamente todos los estilos y las funcionalidades javascript.

```
php app/console assetic:dump --env=prod --no-debug
```

3.7 Easybook

Por ultimo faltaría modificar la siguiente línea en el siguiente fichero de Easybook, puesto que para organizar mejor la estructura del proyecto hemos hecho cambios en este vendor, y al tocar en un código que es externos a nosotros, cada vez que instalamos dependencias mediante composer, este archivo puede variar y tenemos que estar atentos de tenerlo correctamente.

`/var/www/costasur/vendor/easybook/easybook/book`

```
require __DIR__.'/../../autoload.php';
```

Capítulo 4
Configuración del sistema de emails

4.1 Introducción

Para mandar emails desde Symfony, hemos elegido el Bundle de Swiftmailer donde este funciona conectadonos a un servidor de email, nos logueamos con una cuenta que la hemos configurado previamente en dicho servidor, y mandamos el email a través de él.

Todo esto lo haremos con un servicio que hemos creado en el Bundle de Servicios, alojado en la carpeta Helper de dicho Bundle.

4.2 Configuración

La configuración la encontramos en dos ficheros, config.yml y parameters.yml

En el fichero parameters.yml, hemos configurado el servidor de correo. En este fichero, definimos que vamos a mandar los emails mediante smtp sin cifrar, mediante el servidor mail.selfmodif.com, con la cuenta de correo envios@costasur, que tiene la password envioscostasur y por último hemos creado un email para debug, el cual como veremos más adelante, entrará en acción cuando activemos el correo en modo debug.

```
mail_transport:     smtp
mail_encryption:    ~
mail_host:          mail.selfmodif.com
mail_user:          envios@costasur.com
mail_password:      envioscostasur
mail_debug:         envios@costasur.com
```

En el fichero config.yml, es donde realmente configuramos Swifmailer y aqui lo que hay que destacar, es que como podemos ver, la variable delivery_address está comentada. Esto es porque tenemos el correo en producción, si descomentamos esta línea, significa que

estamos en modo debug, y todos los correos se envían a la dirección que contenga mail_debug, que como vimos anteriomente es envios@costasur.com. Por otro lado, podemos desactivar el envío de emails poniendo a true la variable disable_delivery.

Si nos fijamos en la ultima linea, utilizamos spool para el envío de emails, es decir, los emails que enviamos están en una pila, la cual procesa Symfony y el ordena cuando se envía esos emails, de forma que es mucho mas rápido para la carga de la página. ya que no esperamos ningún tipo de respuesta por parte del servidor.

En el entorno de desarrollo, con este sistema no se envía los mails, ya que solo en producción es cuando symfony se encarga de ir vaciando esa pila de email, por lo tanto para enviar los emails desde producción, tendremos que ejecutar la siguiente línea desde la terminal.

```
app/console swiftmailer:spool:send
```

A continuación vemos el fichero de configuración

```
# Swiftmailer Configuration
swiftmailer:
    transport:  "%mail_transport%"
    encryption: "%mail_encryption%"
    auth_mode:  login
    host:       "%mail_host%"
    username:   "%mail_user%"
    password:   "%mail_password%"
    #delivery_address: "%mail_debug%"  # Si activamos este campo, se envia tod
os los mails a este correo para debuguear
    disable_delivery: false # Parametro para desactivar el envio de mails si e
stá a true
    spool:
        type: file
        path: %kernel.cache_dir%/swiftmailer/spool
```

4.3 Uso

Una vez que tenemos configurado el swiftmailer, ya tenemos configurado el sistema de mails en el servidor y ya en este punto funciona en cualquier entorno de desarrollo que tenga conexión a internet, ya que necesitamos conectarnos al servidor de mail.

Para mandar un email, hemos creado un servicio y solamente deberíamos de llamar a la siguiente función.

```
$bodyMail = "Esto es un correo de prueba";
```

```
$this->get('costasur.helper.emails')->sendMail("destinatario@costasur.com",
"Test",$bodyMail,true);
```

Capítulo 5

Configuración de KnpSnappyBungle como conversor html->pdf o html->Imagen

5.1 Introducción

Hemos elegido este Bundle para generar ficheros pdf desde nuestra aplicación. El bundle tira de "wkhtmltopdf". Y para que funcione correctamente debemos instalarla en nuestro sistema.

5.2 Instalación

Visistamos la web: http://code.google.com/p/wkhtmltopdf/

Aqui nos bajamos la versión binaria estable que sea acorde a nuestro sistema. En nuestro caso, para una plataforma en linux x64.

Estos son los ficheros a descargar:

```
wkhtmltopdf-0.9.9-static-amd64.tar.bz2
wkhtmltoimage-0.11.0_rc1-static-amd64.tar.bz2
```

Los descargamos y descomprimimos dentro de (con permisos de root):

```
/usb/local/bin
```

Una vez allí podemos renombrarlos para que queden así:

```
/usb/local/bin/wkhtmltopdf
/usb/local/bin/wkhtmltoimage
```

Es muy importante realizar la instalacion de esta forma. De hacerlo usando “apt-get” no nos funcionará correctamente al igual que tampoco funciona si utilizamos versiónes "RC".

5.3 Instalación en SF2.

Este paso solo se ejecuta una vez en el proyecto, por lo que no tendrás que volver a repetirlo si estás dentro del proyecto Costasur.

Agregar al Composer.json en require :

```
"knplabs/knp-snappy-bundle": "dev-master"
```

Agregar al Appkernel.php:

```
new Knp\Bundle\SnappyBundle\KnpSnappyBundle(),
```

En config.yml:

```
knp_snappy:
    pdf:
        enabled:    true
        binary:     /usr/local/bin/wkhtmltopdf
        options:    []
    image:
        enabled:    true
```

```
        binary:     /usr/local/bin/wkhtmltoimage
        options:    []
```

5.4 Ejemplos de uso dentro de un controller.

Para usarlo dentro de un controler podemos hacerlo de dos formas.

5.4.1 Generando pdf o imagen desde una Url

Con este metodo, basta con pasarle la url y la ruta destino donde guardaremos nuestro resultado.

```
$ruta = '/home/fran/Documentos/costa/app/cache/prod/tmp/tmp.pdf';
$url = 'http://www.telecinco.es';
$this->container->get('knp_snappy.pdf')->generate($url, $ruta);
```

Si bien no estaría de más acordar entre todos cual será el directorio "tmp" por defecto y colocar esta ruta en algún fichero de parámetros. Es importante tener en cuenta que la ruta que escojamos tenga permiso de escritura para www-data.

5.4.2 Generando pdf o imagen desde una plantilla de twig.

Si escogemos esa opción, el código de ejemplo seria el siguiente:

Use Symfony\Component\HttpFoundation\Response;

```
$html = $this->renderView('MyBundle:Foo:bar.html.twig', array(
    'some'  => $vars
));
```

```
return new Response(
    $this->get('knp_snappy.pdf')->getOutputFromHtml($html),
    200,
    array(
        'Content-Type'          => 'application/pdf',
        'Content-Disposition'   => 'attachment; filename="file.pdf"'
    )
);
```

Básicamente consiste en guardar en una variable lo que pensabamos enviar a twig, cambiando el método render por renderview, y pasamos esta variable a snappy que lo transforma en un fichero en pdf.

Hay que tener en cuenta que todo esto no ha pasado por twig, por lo que el "html" no tiene formato ni estilo CSS. Habrá que investigar pués sobre este último método, por si nos fuera necesario usarlo.

5.4.3 Notas de Uso.

- El comando acepta varias url para agruparlas en un solo fichero pdf.
- El comando en consola por defecto, machaca el destino si ya existe. Sin embarlgo el bundle, hace lo contrario.

Apéndice 1

Optimizando Fluentd

Para mejorar el comportamiento de fluentd en un servidor habrá que seguir los siguientes pasos, siempre que el administrador del servidor lo considere oportuno, es sólo una recomendación.

1.1 Instala NTPD

> Sincronicemos los relojes...

Para que los logs no tengan errores de timestamp, todos los nodos que vuelquen a la misma base de datos deben marcar la misma hora, para ello instalaremos NTP (http://www.cyberciti.biz/faq/debian-ubuntu-linux-install-ntpd/)

```
$ apt-get install ntp
```

1.2 Incrementa el nº máximo de descriptores de ficheros

Si ejecutas esto

```
$ ulimit -n
```

y te muestra 1024 o menos, es insuficiente: Añade lo siguiente a tu fichero `/etc/security/limits.conf` y "rebootea"

```
root soft nofile 65536
root hard nofile 65536
* soft nofile 65536
* soft nofile 65536
```

1.3 Optimizar el kernel

En el fichero /etc/sysctl.conf añade:

```
# net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240    65535
```

y reinicia (o ejecuta sysctl -w)...