Implantación de aplicaciones web Documentation

Versión 2017.0

José Domingo Muñoz

02 de febrero de 2018

Índice general

El módulo profesional de **Implantación de aplicaciones web** se imparte durante el segundo curso del Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en Red (ASIR).

De acuerdo a la normativa reguladora del ciclo formativo, el módulo profesional de Implantación de aplicaciones web se imparte durante el segundo curso y tiene asignadas un total de 84 horas, a razón de 4 horas semanales.

El índice de contenidos que vamos a estudiar será:

CAPÍTULO 1

Unidades

1.1 Introducción a la implantación de aplicaciones web

1.1.1 Objetivos

- Aprender a trabajar con git y github
- Aprender lenguaje de marcas markdown
- Comprender que existe un integración continua y un despliegue continuo en el proceso de despliegue de la página

1.1.2 Enlaces

- Aspectos básicos de las aplicaciones Web
- Evolución de la web (Pablo E. Lozada Y.)
- Aplicaciones web en la Wikipedia
- Diferencias y beneficios entre páginas estáticas y dinámicas

1.2 Implantación de aplicaciones web PHP

1.2.1 Instalación local de un CMS PHP (Drupal)

Objetivos

- Aprender a instalar la pila LAMP en un servidor
- Identificar las características de los CMS más usado

- Descargar e instalar un CMS drupal
- Separar servidor web y servidor de base de datos en equipos distintos
- Instalación de otros CMS usando virtualhost
- Necesidad de otros servicios: por ejemplo, servidor de correo saliente

1.2.2 Despliegue tradicional de CMS PHP

- Seleccionar un hosting compartido e identificar sus características
- Subir ficheros al hosting compartido (FTP)
- Realizar la configuración de la base de datos de forma adecuada
- Modificación (instalación de plugin) en el CMS
- Migración a otro hosting compartido
- Estudiar cómo podemos realizar una copia de seguridad de nuestra aplicación
- Identificación de problemas: los cambios se realizan directamente en el servidor en producción
- Identificación de problemas: los hosting compartidos no son escalables ni elásticos
- Identificación de problemas: el uso de FTP no me permite control de versiones y no es el mecanismo más eficiente.
- Identificación de problemas: Este esquema no funciona si tenemos un equipo de desarrollo construyendo una aplicación web a medida.
- Identificación de problemas: Las copias de seguridad pueden ser complicadas de realizar y además es complicado la automatización.

1.2.3 Enlaces

- Opensource CMS
- How to Install a LAMP Server on Debian 9 Stretch Linux

1.3 Implantación de aplicaciones web Python

1.3.1 Entorno de desarrollo y producción con aplicaciones web python

Objetivos

- Identificar las características entre los distintos entornos (desarrollo, producción)
- Identificar la necesidad de utilizar sistemas de control de versión como git
- Identificar la problemática de que el entono de desarrollo se tiene asemejar lo máximo posible al de producción: infraestructura y dependencias (vagrant y entornos virtuales).
- Valorar el uso de los framework para el desarrollo de aplicaciones web
- Utilizar servidores web de desarrollo para probar la aplicación.
- Entender que los datos de la aplicación son distintos en desarrollo y producción (base de datos de desarrollo y base de datos en producción), incluso con motores de base de datos distintos.

- Instalar un CMS python
- Identificación de problemas: si estamos desarrollando una aplicación es necesario probarla, realizar test.
- Identificación de problemas: además de lo anterior el equipo de desarrollo necesita ir haciendo otros procesos: analizando el código generado, generar documentación,. . .
- Identificación de problemas: Nuestro equipo de desarrollo las componen varios miembros: es fundamental utilizar un repositorio común (git)
- Identificación de problemas: Si seguimos una metodología ágil es deseable que todos los cambios que vayan realizando los programadores se vayan probando, analizando, . . . de forma continúa
- Identificación de problemas: ¿Y si esas tareas las automatizamos? -> Integración continúa

1.3.2 Enlaces

- Crear una página web con Python
- Ficheros de ejemplo
- Lista de framework Python
- Lista de cms desarrollados en Django
- Lista de cms desarrollados en Python
- Entornos de desarrollo virtuales con Python 3
- WSGI
- Despliegue de aplicaciones flask
- Despliegue de aplicaiones django

1.4 Introducción a la integración continúa y despliegue continuo

1.4.1 Objetivos

- Conocer el software de integración continúa más utilizados
- Tener una primera experiencia con travis para realizar la integración continúa (en este caso simplemente realizar de forma automática que todos los cambios en nuestra página web estática es válida)
- Despliegue manual de nuestra página en el servicio surge.sh
- Entender el concepto de despliegue continúo y comprobar la función de despliegue de travis sobre surge.sh

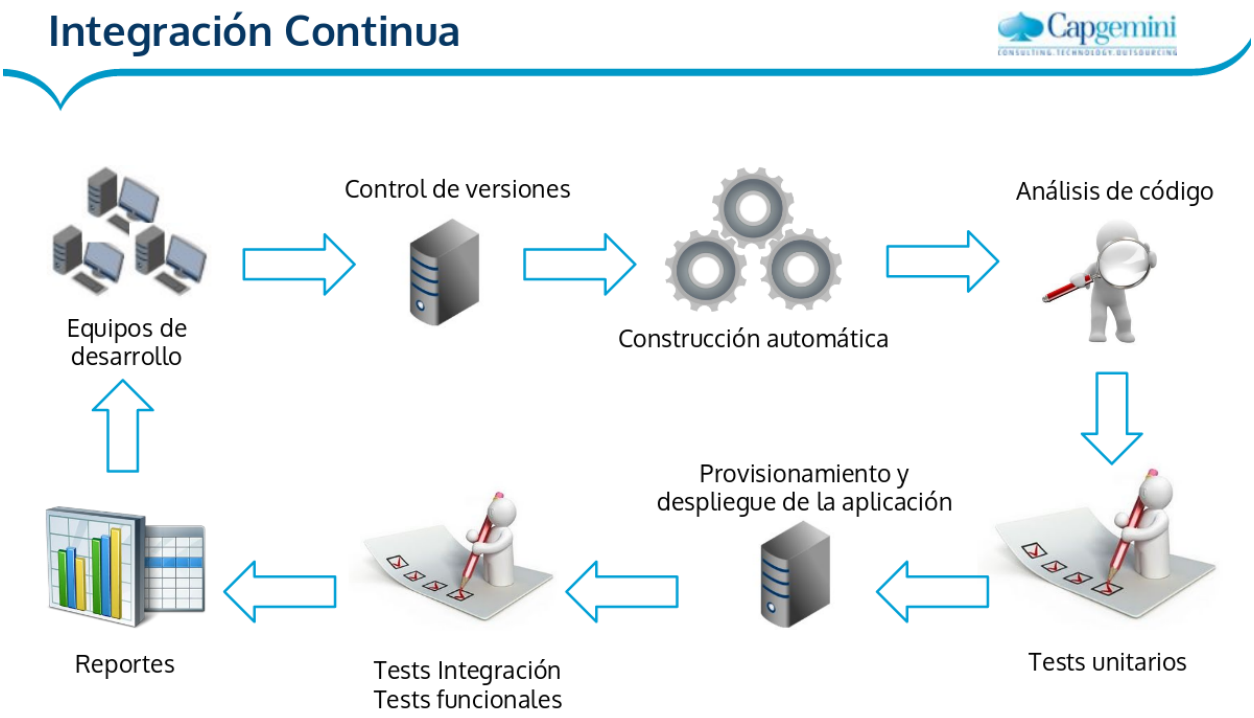
1.4.2 Contenidos

- Integración continua: La integración continua requiere que cada vez que alguien haga un commit se construya la aplicación entera y que se ejecuten una serie de tareas automatizados: Las pruebas y el despliegue se automatizan.
- Integración continua: Práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente, al menos una vez al día. Cada integración se verifica con un build automático (que incluye la ejecución de pruebas) para detectar errores de integración tan pronto como sea posible.

Procesos que se realizan en la IC:

- Se recoge el código fuente de un repositorio compartido.
- Se analiza el código
- Se compila o transforma el código necesario (build)
- Se ejecutan los test (unitarios, integrales y funcionales)
- Se generan informes y documentación

Es un cambio de paradigma. Es necesario la aceptación de los miembros del equipo, puesto que la integración continua es una práctica y no una herramienta.



- Entrega continua (EC): Es el siguiente paso de IC, y consiste en preparar la aplicación web para su puesta en producción. El paso a producción se hace de forma manual.
- Despliegue continuo (DC): Es similar a la anterior pero en este caso también se automatiza el despliegue final en producción.

Herramientas

- CloudBees (jenkins)
- Cloud Foundry
- CircleCi
- Semaphore
- Travis
- Codeship
- tddium
- Wercker
- Shippable

- Go-Ci
- snap-ci
- appveyor

1.4.3 Enlaces

- Introducción a la integración continua
- Instalación de Jenkins en debian
- QUÉ ES TDD - Test-driven development

1.5 Despliegue de aplicaciones web Java

1.5.1 Objetivos

- Conocer los elementos fundamentales en la creación de aplicaciones web con Java.
- Conocer los principales servidores de aplicación para desplegar aplicaciones JAva.
- Profundizar en la utilización del servidor de aplicación Tomcat.
- Desplegar aplicaciones web en Tomcat
- Integrar el servidor de aplicación Tomcat con un servidor web

1.5.2 Contenidos

- Introducción a tomcat8

1.5.3 Enlaces

- Estructura de directorio / ficheros en Tomcat
- Archivos WARS
- CMS escritos en Java
- Aplicaciones Java en Bitnami
- Conector Mysql
- Conector PostgreSQL

1.6 Despliegue de aplicaciones con contenedores

1.6.1 Objetivos

- Conocer los conceptos principales sobre el despliegue de aplicaciones web utilizando contenedores.
- Conocer los conceptos fundamentales de docker.
- Desplegar aplicaciones web sencillas en contenedores.

- Estudiar el ciclo de vida del desarrollo, pruebas y despliegue de aplicaciones utilizando contenedores.
- Creación de imágenes docker.
- Trabajo con almacenamiento en docker.

1.6.2 Contenidos

- Introducción a docker
- Instalación de docker en debian
- Nuestro primer contenedor «Hola Mundo»
- Ejecutando un servicio en docker
- Gestionando el registro Docker Hub
- Dockerfile: Creación de imágenes docker
- Ejemplos de ficheros Dockerfile, creando imágenes docker
- Gestionando el registro Docker Hub
- Enlazando contenedores docker
- Gestión del almacenamiento en docker
- Gestionando el almacenamiento docker con Dockerfile

1.6.3 Enlaces

- Docker
- rkt
- cri-io
- rkt us docker

Y las pácticas que vamos a realizar son:

CAPÍTULO 2

Prácticas

2.1 Implantación de una aplicación web estática en Github Pages

La forma tradicional de crear un sitio web estático sería de la siguiente forma:

1. Crear las distintas páginas html en el entorno de desarrollo.
2. Podemos añadir alguna funcionalidad añadiendo código javascript, que se ejecutará en el cliente.
3. El entorno de producción necesita sólo un servidor web.
4. El despliegue podemos hacer usando un servidor ftp para subir los ficheros al servidor.

Ventajas de las páginas estáticas:

- Portabilidad, funcionan en cualquier servidor.
- Tiempos de acceso óptimos, tardan muy poco en cargarse.
- Facilitan el posicionamiento.
- Costos de alojamiento menores.
- Mínimos requerimientos técnicos para su operación.

Desventajas de las páginas estáticas:

- Funcionalidad muy limitada
- No se pueden realizar búsquedas en la página.
- El visitante no tiene ninguna posibilidad de seleccionar, ordenar o modificar los contenidos o el diseño de la página a su gusto.
- El administrador web debe acceder al servidor donde está alojada la página para cambiar los contenidos de la página.
- El proceso de actualización es lento, tedioso y esencialmente manual.
- No se accede a bases de datos (esto puede ser también una ventaja)

2.1.1 Github Pages

Github Pages es un servicio que te ofrece Github para publicar de una manera muy sencilla páginas web. Disponemos de la opción de generar de forma automática las páginas utilizando una herramienta gráfica, o la creación y modificación de páginas web usando la línea de comandos con el comando git.

Cómo podemos construir nuestras páginas web

La forma más sencilla de construir nuestro sitio es subir a nuestro repositorio todos los ficheros necesarios: ficheros html, hojas de estilos, javascript, imágenes, etc. Si sólo tuviéramos esta opción de edición de páginas no tendríamos grandes ventajas para decidirnos a escoger este servicio de hosting.

Lo que realmente hace esta herramienta una opción muy potente es que Github Pages suporta Jekyll, herramienta escrita en Ruby que nos permite generar, de una forma muy sencilla, ficheros HTML estáticos. Aunque esta herramienta esta pensada para generar blogs, nosotros vamos a utilizar algunas de sus funcionalidades para crear páginas estáticas convencionales.

Usando Jekyll para crear páginas web

La principal característica de Jekylls es la generación de html estático a partir de dos recursos muy simples:

- Plantillas (templates): Ficheros que contienen el esqueleto de las página html que se van a generar. Estos ficheros normalmente se escriben siguiendo la sintaxis de Liquid.
- Ficheros de contenido: Normalmente escritos en sintaxis Markdown y que contienen el contenido de la página que se va a generar.

Por lo tanto una vez que tengo definidas mis plantillas, lo único que tengo que hacer es centrarme en el contenido escribiendo los diferentes de ficheros de contenido.

Usando Markdown para escribir el contenido de nuestras páginas

Los distintos contenidos de nuestras páginas serán definidos en ficheros Markdown con extensión md. La sintaxis de este lenguaje de marcas es muy sencilla y fácilmente convertible a html. Para practicar las distintas opciones puedes usar este editor online.

2.1.2 Mi experiencia

Si las cosas que aprendéis las escribís es probable que no se olviden y se entiendan mejor, por lo tanto tenéis más información en el artículo que escribí cuando aprendí a hacerlo (lo he modificado en estos días).

Nota: Crea un sitio web usando la herramienta github pages, está página te servirá durante el curso para documentar las distintas prácticas que vamos a realizar en el módulo. Tienes que tener en cuenta:

- Puedes crear una página de usuario, del tipo `http://nombre_de_usuario.github.io` o una página de repositorio, en este caso la url sería `http://nombredeusuaio.github.io/nombrederepositorio`
- Puedes crear una página utilizando el generador automático de plantillas, o subiendo directamente una plantilla propia.
- Tiene que tener al menos dos páginas: la principal (`index.md`) y una página que hablé de ti (`about.md`), y una imagen.

Comenta en la tarea los pasos relevantes.

2.2 Instalación local de un CMS PHP

Nota: Investiga las funcionalidades y características de la última versión del CMS drupal. (https://www.drupal.org)

Esta tarea consiste en instalar un CMS de tecnología PHP (drupal 7 o drupal 8) en un servidor local. Los pasos que tendrás que dar los siguientes:

2.2.1 Tarea 1: Instalación de un servidor LAMP

- Crea una instancia de vagrant basado en un box debian o ubuntu
- Instala en esa máquina virtual toda la pila LAMP

Nota: Entrega un documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (1 punto)

2.2.2 Tarea 2: Instalación de drupal en mi servidor local

- Configura el servidor web con virtual hosting para que el CMS sea accesible desde la dirección: `www.nombrealumno-drupal.org`.
- Crea un usuario en la base de datos para trabajar con la base de datos donde se van a guardar los datos del CMS.
- Descarga la versión que te parezca más oportuna de Drupal (7 o 8) y realiza la instalación.
- Realiza una configuración mínima de la aplicación (Cambia la plantilla, crea algún contenido, . . .)
- Instala un módulo para añadir alguna funcionalidad a drupal.

Nota: En este momento, muestra al profesor la aplicación funcionando en local. Entrega un documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (4 puntos)

2.2.3 Tarea 3: Configuración multinodo

- Realiza un copia de seguridad de la base de datos
- Crea otra máquina con vagrant, conectada con una red interna a la anterior y configura un servidor de base de datos.
- Crea un usuario en la base de datos para trabajar con la nueva base de datos.
- Restaura la copia de seguridad en el nuevo servidor de base datos.
- Desinstala el servidor de base de datos en el servidor principal.
- Realiza los cambios de configuración necesario en drupal para que la página funcione.

Nota: Entrega un documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. En este momento, muestra al profesor la aplicación funcionando en local. (2 puntos)

2.2.4 Tarea 4: Instalación de otro CMS PHP

- Elige otro CMS realizado en PHP y realiza la instalación en tu infraestructura.
- Configura otro virtualhost y elige otro nombre en el mismo dominio.

Nota: En este momento, muestra al profesor la aplicación funcionando en local. Y describe en redmine los pasos fundamentales para realizar la tarea. (2 puntos)

2.2.5 Tarea 5: Necesidad de otros servicios

- La mayoría de los CMS tienen la posibilidad de mandar correos electrónicos (por ejemplo para notificar una nueva versión, notificar un comentario,...)
- Instala un servidor de correo electrónico en tu servidor. debes configurar un servidor relay de correo, para ello en el fichero `/etc/postfix/main.cf`, debes poner la siguiente línea:

```
relayhost = babuino.gonzalonazareno.org
```

- Configura alguno de los CMS para utilizar tu servidor de correo y realiza una prueba de funcionamiento.

Nota: Muestra al profesor algún correo enviado por tu CMS. (1 punto)

2.3 Despliegue tradicional de CMS PHP

Advertencia: Tienes dos opciones para realizar esta práctica:

1. Consideramos que el primer hosting compartido es tu servidor dedicado, posteriormente tendrás que hacer la migración a un hosting externo.
2. Utilizar dos servicios de hosting distintos (tal cómo se explica en la práctica.)

2.3.1 Tarea 1: Elección del escenario que vas a montar

Nota: Indica que opción has elegido:

- Servidor dedicado y hosting externo (1 punto)
- Dos servicios de hosting distintos (2 puntos)

Esta tarea consiste en instalar un CMS de tecnología PHP (elige un CMS que no hayas usado en la práctica anterior) en un hosting compartido. Los pasos que tendrás que dar los siguientes:

2.3.2 Tarea 2: Elección de un hosting compartido

- Elige un servicio de hosting compartido con las caracterísitcas necesarias para instalar un CMS PHP (soporte PHP, base de datos,. . .)
- Date de alta en el servicio.

Nota: Indica que servicio de hosting has elegido. Indica las características del hosting que vas a utilizar. Indica la URL que tendrá tu página web. Indica el CMS que vas a instalar. (1 punto)

2.3.3 Tarea 3: Instalación del CMS PHP en el hosting compartido.

- Descarga en tu ordenado el CMS PHP que vas a instalar.
- Sube los ficheros al hosting compartido por FTP.
- Realizar la configuración de la base de datos de forma adecuada.
- Realiza la instalación.
- Realiza una configuración mínima de la aplicación (Cambia la plantilla, crea algún contenido, . . .)
- Instala un módulo para añadir alguna funcionalidad al CMS PHP.

Nota: En este momento, muestra al profesor la aplicación funcionando. Entrega un documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (2 puntos)

2.3.4 Tarea 4: Migración de la apliación web

- Elige otro servicio de hosting para PHP.
- Realiza el proceso de migración.

Nota: Entrega un documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. En este momento, muestra al profesor la aplicación funcionando en el otro hosting. (4 puntos)

2.3.5 Tarea 5: Copia de seguridad de tu aplicación

- ¿Cómo harías una copia de seguridad de tu aplicación? ¿Crees que se puede automatizar dicha tarea?

Nota: Entrega una documentación donde indiques los pasos para realizar una copia de seguridad. Si puedes realiza un pequeño script que automatice dicha tarea. (1 punto)

Advertencia:

- Identificación de problemas: los cambios se realizan directamente en el servidor en producción
- Identificación de problemas: los hosting compartidos no son escalables ni elásticos

- Identificación de problemas: el uso de FTP no me permite control de versiones y no es el mecanismo más eficiente.
- Identificación de problemas: Este esquema no funciona si tenemos un equipo de desarrollo construyendo una aplicación web a medida.
- Identificación de problemas: Las copias de seguridad pueden ser complicadas de realizar y además es complicado la automatización.

2.4 Introducción al despliegue de aplicaciones python

2.4.1 Tarea 1: Entorno de desarrollo

Vamos a desarrollar la aplicación del tutorial de django 1.10. Vamos a configurar tu equipo como entorno de desarrollo para trabajar con la aplicación, para ello:

- Clona el repositorio de GitHub: https://github.com/josedom24/django_tutorial.
- Crea un entorno virtual e instala las dependencias necesarias para que funcione el proyecto (fichero `requierements.txt`).
- Comprueba que vamos a trabajar con una base de datos sqlite (`django_tutorial/\settings.py`). ¿Cómo se llama la base de datos que vamos a crear?
- Crea la base de datos: `python manage.py migrate`. A partir del modelo de datos se crean las tablas de la base de datos.
- Crea un usuario administrador: `python manage.py createsuperuser`.
- Entra en la zona de administración (`\admin`) para comprobar que los datos se han añadido correctamente.
- Crea dos preguntas, con posibles respuestas. (Nota: El programa tiene problemas con caracteres unicode, no uses acentos, ni ñ,...)
- Ejecuta el servidor web de desarrollo y comprueba en el navegador que la aplicación está funcionando.

Nota: En este momento, muestra al profesor la aplicación funcionando. Entrega una documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (2 puntos)

2.4.2 Tarea 2: Entorno de producción

Vamos a realizar el despliegue de nuestra aplicación en un entorno de producción, para ello vamos a utilizar una instancia del cloud, para ello:

- Instala en el servidor los servicios necesarios (apache2). Instala el módulo de apache2 para ejecutar código python.
- Clona el repositorio en el `DocumentRoot` de tu virtualhost.
- En el entorno de producción no vamos acrear un entorno virtual, vamos a instala django en el sistema: `apt install python-django`.
- Configura un virtualhost en apache2 con la configuración adecuada para que funcione la aplicación. El punto de entrada de nuestro servidor será `django_tutorial//wsgi.py`. Es recomendable que sigas este manual: https://docs.djangoproject.com/en/1.10/howto/deployment/wsgi/modwsgi/

- Crea la base de datos.
- Crea un usuario administrador.
- Desactiva en la configuración (fichero `settings.py`) el modo debug a False. Para que los errores de ejecución no den información sensible de la aplicación.
- Muestra la página funcionando.

Nota: En este momento, muestra al profesor la aplicación funcionando. Entrega una documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (3 puntos)

2.5 Entorno de desarrollo y producción con aplicaciones web python

2.5.1 Tarea 1: Entrono de desarrollo

Formas parte del equipo de desarrollo de la aplicación «Gestión IESGN», aplicación web desarrollada con python, con el framework django. Vamos a configurar tu equipo como entorno de desarrollo para trabajar con la aplicación, para ello:

- Realiza un fork del repositorio de GitHub: https://github.com/jd-iesgn/iaw_gestionGN.
- Clona el repositorio en tu equipo.
- Crea un entorno virtual e instala las dependencias necesarias para que funcione el proyecto (fichero `requierements.txt`).
- Comprueba que vamos a trabajar con una base de datos sqlite (`gestion\settings.py`). ¿Cómo se llama la base de datos que vamos a crear?
- Crea la base de datos: `python manage.py migrate`. A partir del modelo de datos se crean las tablas de la base de datos.
- Añade los datos de prueba a la base de datos. Para más información: https://coderwall.com/p/mvsoyg/django-dumpdata-and-loaddata. Utiliza el fichero `datos.json`.
- Entra en la zona de administración para comprobar que los datos se han añadido correctamente. Usuario: `admin` ontraseña: `asdasd1234`).
- Ejecuta el servidor web de desarrollo y comprueba en el navegador que la aplicación está funcionando. Accede con el usuario `usuario` (contraseña: `asdasd1234`).

Nota: En este momento, muestra al profesor la aplicación funcionando. Entrega una documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (3 puntos)

2.5.2 Tarea 2: Desarrollando nuestra aplicación

Vamos a realizar un cambio en la aplicación y comprobar que los cambios se realizan correctamente.

- Modifica la página inicial de la aplicación para que aparezca tu nombre.
- Sube los cambios al repositorio

Nota: Muestra una captura de pantalla donde sea la modificación realizada. (1 punto)

2.5.3 Tarea 3: Entorno de producción

Vamos a realizar el despliegue de nuestra aplicación en un entorno de producción, para ello vamos a utilizar una instancia del cloud, para ello:

- Instala en el servidor los servicios necesarios (apache2, mysql, . . .). Instala el módulo de apache2 para ejecutar código python.
- Clona tu repositorio en el `DocumentRoot` de tu virtualhost.
- Crea un entorno virtual e instala las dependencias de tu aplicación.
- Instala el módulo que permite que python trabaje con mysql:

```
$ apt-get install python-mysqldb
```

  Y en el entorno virtual:

```
(env)$ pip install mysql-python
```

- Configura un virtualhost en apache2 con la configuración adecuada para que funcione la aplicación. El punto de entrada de nuestro servidor será `iaw_gestionGN/gestion/wsgi.py`.
- Crea una base de datos y un usuario en mysql.
- Configura la aplicación para trabajar con mysql, para ello modifica la configuración de la base de datos en el archivo `settings.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'myproject',
        'USER': 'myprojectuser',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

- Crea las tablas de la base de datos y carga los datos de pruebas. Accede a mysql y comprueba que se han creado de forma adecuada.
- Desactiva en la configuración (fichero `settings.py`) el modo debug a False. Para que los errores de ejecución no den información sensible de la aplicación.
- Muestra la página funcionando.

Nota: En este momento, muestra al profesor la aplicación funcionando. Entrega una documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. (4 puntos)

2.5.4 Tarea 4: Modificación de la aplicación en el entorno de producción

Vamos a realizar cambios en el entorno de desarrollo y posteriormente vamos a subirlas a producción.

- Modifica la página inicial para que muestre otra imagen. Despliega los cambios en el servidor de producción.
- Vamos a crear una nueva tabla en la base de datos, para ello sigue los siguientes pasos:

1. Añade un n uevo modelo al fichero `centro/models.py`:

```
class Modulos(models.Model):
    Abr = models.CharField(max_length=4)
    Nombre = models.CharField(max_length=50)
    Unidad = models.ForeignKey(Cursos,blank=True,null=True,on_delete=models.SET_
↪NULL)

    def __unicode__(self):
        return self.Abr+" - "+self.Nombre

    class Meta:
        verbose_name="Modulo"
        verbose_name_plural="Modulos"
```

2. Crea una nueva migración: `python manage.py makemigrations`.
3. Y realiza la migración: `python manage.py migrate`
4. Añade el nuevo modelo al sitio de administración de django:

Para ello cambia la siguiente línea en el fichero `centro/admin.py`:

```
from centro.models import Cursos,Alumnos,Departamentos,Profesores,Areas
```

Por esta otra:

```
from centro.models import Cursos,Alumnos,Departamentos,Profesores,Areas,Modulos
```

Y añade al final la siguiente línea:

```
admin.site.register(Modulos)
```

- Despliega el cambio producido al crear la nueva tabla en el entorno de producción.

Nota: Entrega una documentación resumida donde expliques los pasos fundamentales para realizar esta tarea. En este momento, muestra al profesor la aplicación funcionando en el otro hosting. (4 puntos)

2.5.5 Tarea 5: Despliegue de nuestra aplicación en un hosting python: pythonanywhere

- Siguiendo la documentación despliega nuestra aplicación django en pythonanwhere. Utiliza git para desplegar los ficheros y crea una base de datos en tu proyecto. Si con la documentación no es suficiente puede seguir mi documento: Despliegue de aplicación flask en hosting pythonanywhere.

Nota: Entrega una breve documentación donde expliques los pasos más importantes para el despliegue en pythonanywhere (3 puntos)

Advertencia:

- Identificación de problemas: si estamos desarrollando una aplicación es necesario probarla, realizar test.
- Identificación de problemas: además de lo anterior el equipo de desarrollo necesita ir haciendo otros procesos: analizando el código generado, generar documentación,...
- Identificación de problemas: Nuestro equipo de desarrollo las componen varios miembros: es fundamental utilizar un repositorio común (git)
- Identificación de problemas: Si seguimos una metodología ágil es deseable que todos los cambios que vayan realizando los programadores se vayan probando, analizando, ... de forma continúa
- Identificación de problemas: ¿Y si esas tareas las automatizamos? -> Integración continúa

2.6 Depliegue de CMS python: Mezzanine

En estas práctica vamos a desplegar un CMS python. Hemos elegido Mezzanine.

2.6.1 Tarea 1: Instalación de Mezzanine en el entorno de desarrollo

Instala localmente (usando un entrono virtual) el CMS Mezzazine. Realiza una modificación en la página web. Guarda los ficheros generados durante la instalación en un repositorio github.

Nota: Entrega una breve documentación de los pasos para realizar la instalación. Entrega una captura de pantalla donde se vea el acceso al servidor web de desarollo (2 punto)

2.6.2 Tarea 2: Migración a un entorno de producción

Prepara el entorno de producción con un entrono virtual. Instala el servidor web que vas a utilizar (apache o nginx). Instala el gestor de base de datos que vas a utilzar (mysql, postgress).

- Clona tu repositorio con tus ficheros del CMS.
- Realiza una copia de seguridad de los datos del gestor de contenido en desarrollo, y realiza la migración de forma adecuada.

Nota: Entrega una breve documentación de los pasos para realizar la instalación. Teniendo en cuenta las siguientes puntuaciones:

- Usar un entorno virtual con python2 (1 punto), utilizar python3 (2 puntos).
- Usar apache2 (1 punto), usar nginx (2 puntos).
- Usar mysql (1 punto), usar postgreSQL (2 puntos).
- Usar una URL del tipo `www.tupagina.com` (1 punto), usar una URL del tipo `www.tupagina.com/blog` (2 puntos)

  Entrega una captura de pantalla donde se vea el acceso al servidor web de producción.

2.7 Introducción a la integración continua

Antes de comenzar date de alta en el servicio web travis con tu cuenta de github. Travis nos permite hacer integración continúa en los proyectos que tenemos guardados en nuestros repositorios de GitHub.

2.7.1 Tarea 1: Corrector ortográfico de documentos markdown (test)

Imaginemos que estamos escribiendo documentos markdown y lo guardamos en un repositorio de github. Queremos que cada vez que hagamos una modificación (commit - push) queremos probar (test) si tienes alguna falta de ortografía. Ese proceso lo vamos a hacer de forma automática y continua con travis. Si tenemos activada la IC en travis sobre nuestro repositorio, cada vez que hagamos un push, travis va a crear una máquina (entorno de pruebas), va a clonar nuestro repositorio y va a realizar la prueba (test) que indiquemos en el fichero `.travis.yml`. Cuando termine la prueba nos va mandar un correo informándonos si la prueba ha tenido éxito o no.

Por lo tanto realiza los siguientes pasos:

- Realiza un fork del repositorio de GitHub: https://github.com/josedom24/ic-travis-diccionario.
- Activa la IC en travis de tu repositorio.
- Comprueba la prueba que vamos a realizar estudiando el fichero `.travis.yml`.
- Realiza cambios en los ficheros que están en el directorio `doc` y comprueba en travis como se van ejecutando las pruebas.

Nota: Entrega varias capturas de pantalla donde se vea una prueba que termina en éxito (sin faltas de ortografía) y otra que no termine en éxito (1 punto)

2.7.2 Tarea 2: Comprobación de html5 válido y despliegue en surge.sh (test y deploy)

En esta tarea queremos desplegar una página html5 en el servicio surge.sh, además queremos comprobar si el código html5 es válido. Estas dos operaciones: comprobar si el html5 es válido (test) y el despliegue en surge.sh (deploy) lo vamos a hacer con travis de forma automática (IC y DC).

Antes de empezar vamos a aprender a trabajar con surge.sh:

- Siguiendo las instrucciones de esta página instala NodeJS y npm.
- Instala surge.sh
- Despliega una pequeña página web en el dominio `tunombre.surge.sh`.

Nota: Entrega una descripción con los pasos fundamentales para la instalación y una captura de la página web que has realizado. (1 punto)

Vamos a añadir la funcionalidad de IC y DC con travis, para ello:

- Realiza un fork del repositorio de GitHub: https://github.com/josedom24/ic-travis-html5
- Activa la IC en travis de tu repositorio.
- Comprueba la prueba y el despliegue que vamos a realizar estudiando el fichero `.travis.yml`.
- Modifica el fichero `.travis.yml` para poner el nombre de dominio que vas a utilziar.

- Para que travis pueda hacer el despliegue en surge le tenemos que indicar un TOKEN. Genera el token:

  surge token

- Crea dos variables de entorno en *settings* del proyecto travis:
  - SURGE_LOGIN: Indica el correo electrónico que has utilizado como lógin en surge
  - SURGE_TOKEN: Indica el TOKEN que has obtenido en el paso anterior.
- Realiza cambios en el fichero index.html del directorio `_build` y comprueba, que si el código html5 es válido se despliega y puedes aceder a la página web. Si el código html5 no es válido no se realiza el despliegue y te mandan un correo informando de la incidencia.

Nota: Entrega una descripción con los pasos fundamentales que has realizado. Entrega varias capturas de pantalla donde se vea una prueba que termina en éxito (HTML5 válido) y otra que no termine en éxito (1 punto)

2.7.3 Tarea 3: Despliegue de un blog desarrollado con Pelican en GitHub Page (build, deploy)

En esta última tarea vamos a construir una página web (blog) con una herramienta escrita en python que se llama Pelican, y la vamos a desplegar en GitHub Page. Todo esto lo vamos a realizar co IC con Travis. Sigue los siguientes pasos:

- Crea un repositorio en GitHub con un README.md, crea también una rama `gh-pages` donde vamos a publicar el blog.
- Crea un entorno virtual e instala Pelican (Puedes seguir las instrucciones que encenstras en esta página: https://www.fullstackpython.com/blog/generating-static-websites-pelican-jinja2-markdown.html). (En el artículo se explica como crear un entrono virtual con python3, pero también funciona con un virtualenv de python2).
- Crea un proyecto, en tu repositorio local de github, con `pelican-quickstart`, aquí es muy importante que pongas la URL base (si suponemos que he creado un repositorio llamado `ic-blog` la URL base sería: `https://josedom24.github.io/ic-blog`).
- Con `make html` se genera el contenido estático, comprueba el resultado en el directorio `output`.
- Con `make devserver` se ejecuta un servidor web para desarrollo, con `make stopserver` se detiene dicho servidor.
- Siguiendo las instrucciones de la página anterior, crea un post en el blog.

Nota: Entrega una captura de pantalla donde se vea el blog con la nueva entrada en el servidor web de desarrollo. (1 punto)

Ahora queremos que este proceso de «build» se haga de manera automática con travis. Además tenemos la opción `make publish github` que realiza el despliegue en Github Page, esto también queremos hacerlos de forma automática. Vamos por pasos:

- Activa tu repositorio en Travis.
- Crea el fichero `.travis.yml` con el siguiente contenido:

  language: python branches: only: - master install:

  - pip install pelican ghp-import markdown script:
  - make publish github

- Sigue las instrucciones de este artículo: Publish your Pelican blog on Github pages via Travis-CI para crear un TOKEN en github, encriptarlo y añadirlo en el fichero `.travis.yml`.
- Por último, cómo indica en el artículo, modifica el fichero `Makefile` para poder desplegar en github usando tu TOKEN. (Sustituye la variable `$(TRAVIS_REPO_SLUG)` pon el nombre de tu repositorio.)

Nota: Describe los pasos más importantes para realzar dicha práctica, y entrega una captura de pantalla donde se vea el blog con la nueva entrada en el GitHub Page. (1 punto)

2.7.4 Tarea 4: Integración continúa de aplicación django (Test + Deploy)

Vamos a trabajar con el repositorio de la aplicación `django_tutorial`. Esta aplicación tiene definidas una serie de test, que podemos estudiar en el fichero `tests.py` del directorio `polls`.

Para ejecutar las pruebas unitarias, ejecutamos la instrucción `python manage.py test`.

Nota: Estudia las distintas pruebas que se han realizado, y modifica el código de la aplicación para que al menos una de ella no se ejecute de manera exitosa. (1 punto)

A continuación vamos a configurar la integración continúa para que cada vez que hagamos un commit se haga la ejecución de test en travis.

Nota: Crea un fichero `.travis.yml` para realizar de los tests en travis. Entrega el fichero `.travis.yml`, una captura de pantalla con un resltado existoso de la IC y otro con un error.(1 punto)

Siguiendo la guía de esta página: Continuous delivery of a Django app from Travis CI to PythonAnywhere. Para además de realizar los tests, se haga un despliegue al servicio **pythonanyhere**.

Nota: Entrega un breve descripción de los pasos más importantes para realizar el despliegue desde travis. (3 puntos)

2.8 Despliegue de CMS java

En esta práctica vamos a desplegar un CMS escrito en java. Puedes escoger la aplicación que vas a desplegar de CMS escritos en Java o de Aplicaciones Java en Bitnami.

Nota: Indica la aplicación que vas a instalar. Indica las características principales de la aplicación. Recuerda que tienes que hacer un despliegue en el servidor Tomcat que tienes instalado (no puedes hacer una instalación desde un fichero ejecutable.

- Se evaluará la originalidad en la elección de la aplicación (máximo 2 puntos).
- Se evaluará la complejidad de la instalación (máximo 3 puntos).

2.8.1 Tarea 1: Despliegue de la aplicacion en el servidor de aplicaciones Tomcat

Realiza el despliegue de la aplicación (fichero WAR) en tu servidor Tomcat.

Nota:

- Escribe una guía de los pasas fundamentales para realizar la instalación.
- ¿Has necesitado instalar alguna librería?¿Has necesitado instalar un conector de una base de datos?
- Entrega una captura de pantalla donde se vea la aplicación funcionando.

3 puntos

2.8.2 Tarea 2: Integración de apache2

Realiza la configuración necesaria en apache2 y tomcat para que la aplicación sea servida por el servidor web.

Nota:

- Escribe una guía de los pasas fundamentales para realizar la integración.
- Entrega una captura de pantalla donde se vea la aplicación funcionando.

2 puntos

2.9 Introducción a docker

En estas práctica vamos a empezar a trabajar con la tecnología de contenedores docker.

2.9.1 Tarea 1: Ejecutando un servicio en un contenedor Docker

- Elige una imagen docker como base para crear un contenedor interactivo. Accede a él y realiza la instalación de un servicio nginx.
- Crea una nueva imagen a partir del contenedor anterior.
- A partir de la nueva imagen crea un contenedor y comprueba que el servidor web esté funcionando.
- Suponiendo que has instalado docker en tu entorno de desarrollo/prueba, realiza un despliegue del contenedor a un entorno de producción (otra máquina con docker instalado). Para ello guarda la imagen generada en docker hub y bájala en el entorno de producción, para crear un nuevo contenedor. Este ejercicio lo podeís hacer entre compañeros.
- A continaución queremos cambiar la página web que sirve el servidor web, describe los pasos que deberíamos realizar para realizarlo. A coninuación realiza el cambio.

Nota:

- Entrega una breve documentación de los pasos que has dado para realizar el ejercicio. Entrega una captura de pantalla donde se vea el acceso al servidor web. (2 puntos).
- Repite el ejercicio con un servidor mysql (no es necesario hacer el último punto). Demuestra que puedes acceder al servidor mysql. (2 puntos)

2.9.2 Tarea 2: Creación de una imagen docker con Dockerfile

Realiza un `Dockerfile` que nos permita crear una imagen desde la que se puedan crear contenedores que sirvan una página web en un servidor web nginx. La página web que se sirve debe ser una plantilla html5.

Nota:

- Entrega el fichero *Dockerfile*.
- Documenta los pasos que has dado para crear un contenedor que sirva la página web.
- ¿Qué pasos tienes que dar para modificar el contenido dela página? Documenta brevemente los pasos que has dado.

(2 puntos)

2.9.3 Tarea 3: Configuración de contenedores por medio de variables de entorno

Queremos configurar el servidor nginx por medio del uso de variables de entorno, para ello:

- Crea en el fichero `Dockerfile` las variables de entorno siguiente:
  - `DOCUMENTROOT`: Indica el directorio raíz del servidor, por defecto será `/var/www/html`.
  - `SERVER_NAME`: Indica el nombre del servidor, (directiva `server_name`), por defecto es _.
- Tendrás que crear un script `run.sh` que será el encargado de configurar nginx antes de ejecuta el proceso, y será el proceso que ejecuta el contenedor.
- Recuerda que los ficheros tendrán que ser copiados al directorio donde se indica en la variable de entorno `DOCUMENTROOT`.

Nota:

- Crea un contenedor si modificar ninguna variable de entorno.
- A continuación crea otro contenedor pero modificando las dos variables de entorno.

(3 puntos)

2.9.4 Tarea 4: Creación automática de una imagen docker en Docker Hub

Realiza los siguientes pasos:

- Crea un repositorio en GitHub, y sube desde una máquina donde no tengas docker instalado el contexto (`Dockerfile` y ficheros necesarios) del ejercicio anterior.
- Crea desde Docker Hub un «Automated build» y da permisos a Docker Hub para que acceda a tu repositorio GitHub.
- Comprueba que se ha creado la nueva imagen.
- Descarga la nueva imagen en tu servidor Docker y crea un nuevo contenedor.

Nota:

- Documenta los pasos que has dado para realizar la tarea.

- ¿Qué pasos tienes que dar para modificar el contenido de la página? Documenta brevemente los pasos que has dado. Y muestra la nueva página con los cambios que has introducido.

(3 puntos)

2.10 Implantación de aplicaciones web PHP en docker

2.10.1 Tarea 1: Ejecución de una aplicación web PHP en docker (1)

- Queremos ejecutar en un contenedor docker la aplicación web escrita en PHP: bookMedik (https://github.com/evilnapsis/bookmedik).
- Es necesario tener un contenedor con mysql donde vamos a crear la base de datos y los datos de la aplicación. El script para generar la base de datos y los registros lo encuentras en el repositorio y se llama `schema.sql`. Debes crear un usuario con su contraseña en la base de datos. La base de datos se llama `bookmedik` y se crea al ejecutar el script.
- Ejecuta el contenedor mysql y carga los datos del script `schema.sql`. Para más información.
- El contenedor mysql debe tener un volumen para guardar la base de datos.
- Crea una imagen docker con la aplicación desde una imagen base de debian o ubuntu. Ten en cuenta que el fichero de configuración de la base de datos (`core\controller\Database.php`) lo tienes que configurar utilizando las variables de entorno del contenedor mysql.
- La imagen la tienes que crear en tu máquina con el comando `docker build`.
- Crea un contenedor a partir de la imagen anterior, enlazado con el contenedor mysql, y comprueba que está funcionando (Usuario: **admin**, contraseña: **admin**)
- El contenedor que creas debe tener un volumen para guardar los logs de apache2.

Nota:

- Creación del contenedor mysql y carga de los datos (1 punto).
- Creación de la imagen docker bookmedik (2 puntos).
- Creación del contenedor con la aplicación web. (1 punto).
- Comprueba que si borramos el contenedor mysql y volvemos a crear otro, los datos no se han perdido porque se han guardado en un volumen. (1 punto)

2.10.2 Tarea 2: Ejecución de una aplicación web PHP en docker (2)

- Realiza la imagen docker de la aplicación a partir de la imagen oficial PHP que encuentras en docker hub. Lee la documentación de la imagen para configurar una imagen con apache2 y php, además seguramente tengas que instalar alguna extensión de php.
- Crea esta imagen en docker hub.
- Crea un script con docker compose que levante el escenario con los dos contenedores.

Nota:

- Documenta como has creado el Dockerfile para crear la imagen, entrega la URL del repositorio github donde has guardado el contexto para la creación de la imagen (2 puntos).
- Docuementa el uso de docker compose, entrega el script que has desarrollado. (1 punto)

2.10.3 Tarea 3: Ejecución de una aplicación PHP en docker (3)

- En este caso queremos usar un contenedor que utilice nginx para servir la aplicación PHP. Puedes crear la imagen desde una imagen base debian o ubuntu o desde la imagen oficial de nginx.
- Vamos a crear otro contenedor que sirva php-fpm.
- Y finalmente nuestro contenedor con la aplicación.
- Crea un script con docker compose que levante el escenario con los tres contenedores.

A lo mejor te puede ayudar el siguiente enlace: Dockerise your PHP application with Nginx and PHP7-FPM

Nota:

- Documenta el proceso que has realizado para crear el escenario. Entrega el script de docker compose y realiza alguna prueba de funcionamiento. (3 puntos).

2.10.4 Tarea 4: Ejecución de un CMS en docker (1)

- A partir de una imagen base (que no sea una imagen con el CMS), genera una imagen que despliegue un CMS PHP (que no sea wordpress). El contenedor que se crea a partir de esta imagen se tendrá que enlazar con un contenedor mysql o postgreSQL.
- Crea los volúmenes necesarios para que la información que se guarda sea persistente.

Nota:

- Elige un CMS PHP y crea la imagen que despliega la aplicación. Crea los contenedores necesarios para servir el CMS. Entrega una prueba de funcionamiento.
- Elimina los contenedores, vuelve a crearlos y demuestra que la información no se ha perdido.

(2 puntos)

2.10.5 Tarea 5: Ejecución de un CMS en docker (2)

Busca una imagen oficial de un CMS PHP en docker hub (distinto al que has instalado en la tarea anterior, ni wordpress), y crea los contenedores necesarios para servir el CMS, siguiendo la documentación de docker hub.

Nota: Explica el proceso que has seguido para realizar la tarea, pon alguna prueba de funcionamiento. ¿Se ha creado algún volumen para que la información sea persistente? (2 puntos)