

Despliegue de aplicaciones web

Proyecto DAW

2º Evaluación

Licencia



Reconocimiento – NoComercial - Compartir Igual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

1.	Introducción	4
2.	Contenido	4
2.1	Backend	4
2.2	Frontend	6
2.3	Despliegue en AWS	8
2.4	HTACCESS	8
2.5	Web de mantenimiento	8
2.6	Despliegue en una plataforma Git-based Auto-Deployment	9
3.	Documentación	9
4.	Entrega	10
5.	Evaluación	10

1. INTRODUCCIÓN

Para esta segunda evaluación, realizaremos un proyecto que abarca el contenido de las últimas 2 unidades involucradas: U4. Servidores Web y U5. Servidor de aplicaciones.

En resumen, en este proyecto tendrás que preparar el entorno de desarrollo y de producción de una aplicación, para posteriormente desplegarla en **AWS** y en una plataforma **de Auto-Deployment Git-based** (*más abajo se explica qué significa este nombrado*). En este proceso, se irán viendo determinados aspectos de esas dos unidades pero no os preocupéis ni os asustéis por llamarlo proyecto, afrontarlo como si simplemente se hubieran juntado las dos supuestas prácticas del tema 5: las asociadas al tutorial de 5.2. *Docker compose* y al de 5.3. *Despliegue en AWS* y vais a poder realizarlo sin problemas siguiendo toda la teoría y tutoriales tanto de la unidad 4, como de la unidad 5.

IMPORTANTE: lee primero todo el contenido del proyecto antes de comenzarlo para tener una visión global de lo que se pide.

2. CONTENIDO

El proyecto consiste en preparar una aplicación FrontEnd y BackEnd con Docker Compose de la misma forma que habéis visto con los tutoriales de la U5 con la aplicación de productos. En este caso, os pasare otra aplicación y ésta será la que tenéis que preparar. No es más difícil de lo que ya habéis visto, puedes incluso partir del otro proyecto e ir adaptándolo todo a los requisitos que te iré pidiendo. Además, hay contenido de las prácticas de la unidad 4 que también os van a servir (si las habéis hecho).

Los recursos que necesitaris para ello, los podréis encontrar en el siguiente enlace, si no funcionara, por favor, enviad un email al profesor: [Recursos](#)

2.1 Backend

Prepara el proyecto BackEnd con Docker Compose, de forma que tengas el entorno de desarrollo y el entorno de producción preparado.

Requisitos:

- El proyecto Backend estará distribuido en **2 contenedores: Mysql y Apache**
- Crea la estructura de directorios que has visto en los tutoriales de la unidad 5 teniendo en cuenta que tienes que tener un entorno de desarrollo y un entorno de producción.

- **Contenedor Mysql:**
 - No será necesario construir ninguna **imagen**, simplemente podrás partir de la imagen de docker hub: “**mysql:8.0.35-debian**”.
 - Para el entorno de desarrollo, las **variables de entorno** se añaden directamente al docker-compose y para el entorno de producción, las variables se tienen que obtener del archivo **.env**.
 - Asigna una contraseña al usuario ROOT, pero utiliza un usuario adicional para trabajar con la base de datos.
 - El contendor debe inicializarse con la base de datos “**films**” siguiendo la estructura del dump que te doy en los recursos. Para el entorno de producción, la base de datos debe inicializarse vacía sin ningún dato en la tabla “**film**”.
 - En el caso de que el contenedor se eliminase, se debe mantener una **copia de seguridad** de la base de datos.
 - En el entorno de producción, este contenedor debe **iniciarse** automáticamente con la máquina.
 - Ponle un **nombre** a este contenedor, no dejes a Docker que te asigne uno automáticamente.
- **Contenedor Apache**
 - El código de la aplicación lo tienes en los recursos: “**backend.zip**”.
 - Será necesario construir la imagen, por lo tanto, crea los **Dockerfiles** correspondientes siguiendo las siguientes indicaciones:
 - Parte de la **imagen** base de docker hub: “**debian**”.
 - **Etiqueta** la imagen con tu email para especificar que eres el **autor**.
 - **Actualiza** los repositorios.
 - **Instala apache2**.
 - Instala **PHP** y las extensiones “**php-yaml**” y “**php-mysql**” ya que el proyecto lee archivos .yml y conecta con una base de datos Mysql.
 - Además, vamos a utilizar determinadas configuraciones de Apache, por lo tanto, se tendrán que habilitar los módulos “**rewrite**” y “**headers**” (echa un vistazo al apartado de HTACCESS y CORS de la unidad 4 y verás de lo que te estoy hablando).
 - **Docker Compose:**
 - El contenedor de la aplicación partirá de la **imagen** construida según los **Dockerfiles** definidos.
 - Ponle un **nombre** a este contenedor, no dejes a Docker que te asigne uno automáticamente.
 - Usa los **volúmenes** para compartir el **código** de la aplicación con el contenedor.

- Usa los **volumenes** para compartir los **virtualhosts** con el contendor.
- El **puerto** a mapear será el **8080**.
- En el entorno de producción, este contenedor debe **iniciarse** automáticamente con la máquina.
- Este contenedor **depende** del contenedor de mysql.
- Recuerda que tienes que tener dos docker-compose, uno para cada entorno, por lo tanto, haz los cambios que consideres según lo que has aprendido.

- **Virtualhosts:**

- Deberás tener dos **virtualhosts** uno para desarrollo y otro para producción, el dominio que especifiques en producción tiene que ser real, ya que posteriormente lo crearás en FreeDNS.
- Solo tendrás un **ServerName**, no es necesario ningún alias.
- Guarda los **logs** de error y access en un lugar que tú especifiques como ya vimos en la unidad 4.
- **CORS:** hay que configurar el CORS para que no de error cuando hagamos peticiones desde el frontend. En lugar de configurarlo en un archivo .htaccess, lo vamos a configurar directamente aquí en el virtualhost, de la misma forma que en el proyecto de productos de la unidad 5. Ten en cuenta que ahora estamos con apache, por lo tanto, revisa el apartado de CORS de la unidad 4 para que sepas cómo tienes que adaptarlo para ponerlo aquí.

- **gitignore:**

- Ignora los archivos .env y dbconfiguration.yml como ya hemos visto en el ejemplo de productos.
- **README:** crea un readme donde expliques cómo desplegar el proyecto en desarrollo y en producción (si quieres te puedes basar en el README del ejemplo de productos).
- **GIT:** crea un repositorio privado con la rama master y develop y sube el proyecto.

2.2 Frontend

Prepara el proyecto FrontEnd con Docker Compose, de forma que tengas el entorno de desarrollo y el entorno de producción preparado.

Requisitos:

- El proyecto Frontend estará distribuido en **1 contenedor: Apache**
- Crea la estructura de directorios que has visto en los tutoriales de la unidad 5 teniendo en cuenta que tienes que tener un entorno de desarrollo y un entorno de producción.
- **Contenedor Apache**

- El código de la aplicación lo tienes en los recursos: “**frontend.zip**”.
- Será necesario construir la imagen, por lo tanto, crea los **Dockerfiles** correspondientes siguiendo las siguientes indicaciones:
 - Parte de la **imagen** base de docker hub: “**debian**”.
 - **Etiqueta** la imagen con tu email para especificar que eres el **autor**.
 - **Actualiza** los repositorios.
 - **Instala apache2**.
 - Además, vamos a utilizar determinadas configuraciones de Apache, por lo tanto, se tendrá que habilitar el módulo “**rewrite**”.
- **Docker Compose:**
 - El contenedor de la aplicación partirá de la **imagen** construida según los **Dockerfiles** definidos.
 - Ponle un **nombre** a este contenedor, no dejes a Docker que te asigne uno automáticamente.
 - Usa los **volúmenes** para compartir el **código** de la aplicación con el contenedor.
 - Usa los **volúmenes** para compartir los **virtualhosts** con el contenedor.
 - Usa los **volúmenes** para compartir el fichero de los recursos “**urls.js**” de forma que se asigne dentro del contenedor en la ruta de la aplicación “**/config/urls.js**” (como habrás visto en el ejemplo de productos).
 - El **puerto** a mapear será el **80**.
 - En el entorno de producción, este contenedor debe **iniciarse** automáticamente con la máquina.
 - Recuerda que tienes que tener dos docker-compose, uno para cada entorno, por lo tanto, haz los cambios que consideres según lo que has aprendido.
- **Virtualhosts:**
 - Deberás tener dos **virtualhosts** uno para desarrollo y otro para producción, el dominio que especifiques en producción tiene que ser real, ya que posteriormente lo crearás en FreeDNS.
 - Para el entorno de producción, además del **ServerName**, tendrás también un **alias**, ese alias será el subdominio de FreeDNS “**mooo.com**”, por lo tanto, al frontend desde producción se deberá poder acceder desde “chickenkiller.com y mooo.com”.
 - **DirectoryIndex**, como el frontend no tiene un archivo index, vamos a especificar dentro del propio virtualhost (no hacerlo con .htaccess) que el index es “**pages/home.html**”.
 - Guarda los **logs** de error y access en un lugar que tú especifiques como ya vimos en la unidad 4.
 - **urls.js**: este archivo es el que se compartirá con el contenedor. Sirve para apuntar al backend, por lo tanto, modifica la URL dependiendo si estás en el entorno de desarrollo o producción.

- **gitignore:**
 - Ignora el archivo urls.js de la aplicación como vimos en el ejemplo de productos.
 - Ignora también los .htaccess (ya que posteriormente en el proyecto crearemos uno).
- **README:** crea un readme donde expliques cómo desplegar el proyecto en desarrollo y en producción.
- **GIT:** crea un repositorio privado con la rama master y develop y sube todo el proyecto.

2.3 Despliegue en AWS

- **Entorno desarrollo:** Una vez esté todo listo, prueba que todo funciona correctamente en el entorno de desarrollo.
- **Entorno producción:** siguiendo los pasos vistos en el tutorial de despliegue de la unidad 5, despliega la aplicación en una **EC2 de AWS** (acuérdate de crear los dominios que necesites en **FreeDNS**).
- **Cambios:** con el entorno de desarrollo funcionando, realiza algún cambio en el código y súbelo a master, después, píblicalo en producción y comprueba que se han aplicado los cambios.

2.4 HTACCESS

En el **entorno de producción** (es decir, desde AWS y estando conectado a la EC2) realiza lo siguiente (únicamente con el FrontEnd):

- Conéctate al contendor del frontend para configurar apache y habilitar la ejecución de los htaccess.
- Después, crea un **.htaccess** en el proyecto (esto puedes hacerlo sin estar conectado al contendor ya que el archivo está dentro del directorio “app/public_html” y lo estamos compartiendo con el volumen):
 - Impide que se **liste** el directorio del proyecto.
 - **Redirige** al dominio principal (es decir, si accedes con el dominio .mooo.com deberá redirigir al dominio de .chickenkiller.com).
 - Crea una página de **error 404** (créala desde el entorno de desarrollo y luego con git la podrás descargar en producción), una vez creada, haz que se redirija a ella cuando

se acceda a una ruta que no exista.

- Gracias a que hemos añadido a .gitignore los .htaccess, este archivo no se añadirá al repositorio.

2.5 Web de mantenimiento

Queremos poder hacer lo siguiente: modificar el .htaccess anterior para que cuando nosotros queramos, seamos capaces de redirigir el frontend a una web de mantenimiento. Por lo tanto, vamos a proceder a crear esta web de mantenimiento y a desplegarla:

- Crea un nuevo proyecto con docker compose, el cual también deberá estar preparado para tener un entorno de desarrollo y un entorno de producción.
- El proyecto partirá de un único contenedor con **Apache**.
- El código del proyecto puede ser simplemente un index.html que represente la web de mantenimiento.
- Este proyecto es muy similar al proyecto del frontend pero incluso más simple, por lo tanto, te puedes basar en él.
- Crea también un repositorio para guardar este proyecto en **GIT**.
- Prueba primero que funcione la web en el entorno de desarrollo.
- **Despliega la web en AWS**, pero esta vez haciendo uso de **EBS (Elastic Beanstalk)**.
- Una vez desplegada, prueba en producción a cambiar el .htaccess para redirigir temporalmente el frontend a esta web de mantenimiento.

2.6 Despliegue en una plataforma Git-based Auto-Deployment

Hoy en día, existen alternativas **PaaS (Platform as a Service)**, que facilitan el despliegue como es el caso de [Render](#), una plataforma de despliegue automático que permite publicar aplicaciones web directamente desde un repositorio Git, incluyendo proyectos basados en Docker, sin gestionar servidores. [Vercel](#), en cambio está especializada en despliegue de aplicaciones Front-End. También existe [Fly.io](#), etc.. Todas las mencionadas, incluyen un plan gratuito y aunque la recomendada es Render, por adaptarse más a las necesidades de este proyecto, se puede utilizar la que se quiera.

Así pues, además de en AWS, la aplicación deberá desplegarse en entorno de producción en una plataforma de *Cloud Hosting con auto-despliegue desde repositorio*, como Render. El alumnado deberá conectar su repositorio Git, configurar el despliegue mediante Docker y dejar la aplicación accesible públicamente, de forma que cualquier cambio en la rama principal se despliegue automáticamente. Para este apartado, **obligatorio para la corrección de la práctica**, deberá de investigar cómo hacerlo por él mismo (es sencillo) fomentando así la autonomía y familiarizándose con una alternativa de despliegue más sencilla, rápida y cómoda a AWS que podrá utilizar para su

TFG, por lo que el tiempo dedicado a este apartado será toda una inversión en el futuro.

3. DOCUMENTACIÓN

NO tenéis que documentar TODO el proceso del proyecto, simplemente hacer una documentación que incluya:

- Una portada (con vuestro nombre y apellidos, asignatura...).
- Un apartado donde mostréis capturas en funcionamiento de lo que os he pedido.
 - Aplicación funcionando en el entorno de desarrollo.
 - Aplicación funcionando en el entorno de producción.
 - Una captura mostrando las EC2 y la aplicación de EBS que has necesitado crear.
 - Documenta con capturas un cambio de código del frontend en el entorno de desarrollo y mostrar el proceso de cómo llevas el cambio a producción.
 - Mostrar en funcionamiento el apartado de htaccess.
 - Mostrar en funcionamiento la web de mantenimiento.
 - Aplicación desplegada en una plataforma auto-deploy

4. ENTREGA

El proyecto se entregará en Aules en la fecha especificada en la tarea que estará creada para ello (*el proyecto será evaluado el día 11 de febrero, lee el siguiente apartado para más información*).

Deberás entregar un fichero llamado “**proyecto_2ev_nombre_apellidos.zip**” que contenga:

- **frontend.zip** (El .zip de tu repositorio)
- **backend.zip** (El .zip de tu repositorio)
- **mantenimiento.zip** (El .zip de tu repositorio)
- **documentación.pdf**

(En el caso de que el tamaño del archivo sobrepase el límite de Aules y no te deje subirlo, súbelo a OneDrive y envía un .txt con el enlace).

5. EVALUACIÓN

El proyecto será evaluado in situ el día y hora del examen, el miércoles día 11 de febrero de 16:50 a 19:00 horas, de forma individual (aproximadamente cada uno tendrá unos 10 minutos).

En esta corrección, deberéis mostrar TODO el proyecto en funcionamiento, os iré evaluando según los apartados pedidos en este enunciado y os podré ir haciendo preguntas respecto a él y/o que hagáis alguna acción concreta. Básicamente quiero asegurarme que domináis vuestro proyecto y sabéis lo que estáis defendiendo y os desenvolvéis con soltura. Según estas respuestas o acciones, las valoraré de forma positiva o negativa.

Ya que en estas unidades no hay parte teórica (EX), todo el peso recae en la parte práctica (PR) distribuyéndose de la siguiente forma en cada unidad:

- **Unidad 4:**
 - El 70% se evaluará con las prácticas que ya habéis realizado (Práctica 4.1 Apache, Práctica 4.2 Docker y Práctica 5.1 Kubernetes (la meto en este RA)).
 - El 30% restante se evaluará con este proyecto.
- **Unidad 5:** el 100% de la UD será evaluado con este proyecto (80% AWS + 20% plataforma).