**Despliegue de tienda web en Kubernetes**

Grado superior ASIR 2021/2023

Salesianos Atocha

Mario Dieguez Salamanca

**Índice**

[**Resumen** 2](#_Toc137107765)

[**Metodología** 3](#_Toc137107766)

[**Base de datos** 3](#_Toc137107767)

[**WEB** 4](#_Toc137107768)

[Introducción 4](#_Toc137107769)

[Navegar como cliente 4](#_Toc137107770)

[Cesta de la compra 7](#_Toc137107771)

[Métodos de pago 7](#_Toc137107772)

[Navegar como administrador 10](#_Toc137107773)

[**Kubernetes** 11](#_Toc137107774)

[Conceptos teóricos 11](#_Toc137107775)

[Creación del clúster 12](#_Toc137107776)

[Imágenes Docker 15](#_Toc137107777)

[Manifests 17](#_Toc137107778)

[Despliegue de la tienda en el clúster 20](#_Toc137107779)

# **Resumen**

Este proyecto está basado en una web que tiene como finalidad la venta de productos informáticos. Esta tiendaestá desplegada sobre en clúster de pruebas de Kubernetes local (KInD). Este va a estar compuesto por dos nodos; un nodo control plane (Master) y un nodo Worker simulados como contenedores Doker.

La web de la tiendase ha desarrollado con los lenguajes HTML, CSS, JavaScript y PHP para interaccionar con la base de datos.

La base de datos se ha desarrollado en el sistema de gestión de base de datos relacional MySQL.

# **Metodología**

A continuación, se desarrolla la realización del trabajo, detallando los aspectos no vistos en el módulo.

## **Base de datos**

Tras la recogida de productos informáticos a través de tiendas reales publicadas en internet, se realiza una base de datos llamada *“TWT\_First”* (nombre ficticio de la tienda). Dentro de la base de datos se pueden visualizar nueve tablas.

* Los datos de los productos se almacenan en las tablas “Componentes”, “TVs”, “Consolas”, “Portátiles”, “PCs” y “Móviles”.
* Los datos de los usuarios registrados se almacenan en la tabla “usuarios”.
* Los datos de las compras realizadas en la tienda se encuentran en tabla “compras”.
* Los registros del foro, dividido en opiniones y preguntas, se guardan en la tabla “foro”.

El motor de almacenamiento de las tablas de la base de datos es “InnoDB” (por defecto de MySQL).

El formato de codificación de caracteres de los registros de las tablas es “UTF-8” ya que empleo tildes y el carácter “ñ” en la sintaxis de algunos productos, de esta manera MySQL podrá traducir los datos correctamente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen de las tablas de la base de datos.

## **WEB**

### Introducción

Dentro de esta tienda web se pueden navegar de diferentes maneras, como cliente o como administrador.

Si el usuario es un cliente o un administrador se define en el inicio de sesión, no se permite que un usuario administrador navegue en el entorno de los clientes y viceversa.

### Navegar como cliente

Para poder visualizar los productos, aparte de tener un menú desplegable arriba a la izquierda (desde el cual se pueden visitar todas las páginas que resumen los productos) dispone de una barra de búsqueda, facilitando al usuario el poder encontrar un artículo desde cualquier página de la web.

Esta barra de búsqueda está desarrollada con Ajax y PHP, vamos a trasladar el texto escrito en el <input> por método POST desde la página en la que el usuario se encuentre a una página PHP ubicada en el directorio raíz de nuestro árbol de directorios (buscador.php), desde el cual se va a realizar una consulta a todas las tablas para mostrar todos los resultados coincidentes gracias al operador “UNION”.

El resultado de la consulta volverá a la página en la que se encuentra el usuario para mostrarse en un <div> definido.

<div class="buscador">

<input onkeyup="buscar\_prod($('#buscar\_1').val());" type="text" id="buscar\_1" name="buscar\_1" placeholder="Buscar...">

</div>

<input> donde se define la palabra/valor a buscar.

<script type="text/javascript">

    function buscar\_prod(buscar) {

       if (buscar.length > 0) {

            document.getElementById('buscar\_1').style = "background-color: white";

            document.getElementById('main').style = "opacity: 0.5";

            document.body.style = "background-color: gray";

            var parametros = {"buscar":buscar};

            $.ajax ({

                data:parametros,

                type:'Post',

                url:'buscador.php',

                success:function(data) {

                    document.getElementById('datos\_buscador').innerHTML = data;

                }

            });

        }

        else {

            document.getElementById('buscar\_1').style = "background-color: transparent";

            document.getElementById('main').style = "opacity: 1";

            document.body.style = "background-color: lightgray";

            document.getElementById('datos\_buscador').innerHTML = '';

        }

    }

</script>

Código Ajax por el cual se manda el texto y se recibe la consulta de PHP para mostrarla en el <div> *“datos\_buscador”*.

$query = "select nombre, empresa, precio, tabla, imagen from (select nombre, empresa, precio, tabla, imagen from Componentes union select nombre, empresa, precio, tabla, imagen from Consolas union select nombre, empresa, precio, tabla,imagen from TVs union select nombre, empresa, precio, tabla, imagen from PCs union select nombre, empresa, precio, tabla, imagen from Portátiles union select nombre, empresa, precio, tabla,imagen from Móviles) as resultado where nombre like '%".$\_POST['buscar']."%' or empresa like '%".$\_POST['buscar']."%' or tabla like '%".$\_POST['buscar']."%' group by nombre";

$consulta = mysqli\_query($conexion, $query);

$rows = mysqli\_num\_rows($consulta);

?>

<div class="buscadorResult">

  <h3>Resultados encontrados: <?php echo $rows ?></h3>

    <?php while($resultado = mysqli\_fetch\_assoc($consulta)) { ?>

        <a href="/PHP/ProductosHTML/<?php echo $resultado['tabla']."/".$resultado['nombre'].".php" ?>"><p><?php echo "<table><tr><td width='10%'><img class='imgbuscador' src='/PHP/Fotos\_Productos/".$resultado['tabla'].'/'.$resultado['imagen']."'.jpg ></td><td>".$resultado['nombre']." - ".$resultado['precio']."€</td></tr></table>";?></p></a>

    <?php } ?>

</div>

Consulta a la base de datos a través del código PHP, seguidamente los registros que coinciden se muestran en el <div class=”buscadorResult”>, este <div> se muestra a su vez en la página en la que se encuentra el cliente dentro del <div> definido en el código Ajax mostrado a continuación:

<div id="datos\_buscador"></div>

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen de la barra de búsqueda.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Imagen del menú desplegable.

Si el usuario navega a través del menú desplegable acabará en una página donde podrá visualizar los productos que ha solicitado, además, depende de la página en la que se encuentre, podrá aplicar una serie de filtros para facilitarle la búsqueda del artículo deseado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen de la página de los productos mostrando los filtros.

Una vez encontrado el producto deseado por el usuario, este encontrará su producto final y podrá definir las unidades, color u otras características del artículo antes de añadirlo a su cesta.

Si el usuario no ha iniciado sesión previamente al intentar añadir el artículo a la cesta, la página le reubica directamente a la página de inicio de sesión, si no se ha registrado con anterioridad podrá dirigirse mediante un enlace a la página de registro. A parte de esto, podrá acceder y modificar sus datos de usuario si se dirigeal icono del individuo ubicado arriba a la derecha de cualquier página.

A su vez, si el usuario ya ha iniciado sesión, podrá añadir a la cesta cualquier producto que desee. El contenido de su cesta lo podrá visualizar dirigiéndose al carrito ubicado arriba a la derecha de cualquier página.

### Cesta de la compra

Estuve debatiéndome entre hacer el carrito de tres maneras; tablas para cada usuario dentro de la base de datos, usar la variable $\_SESSION de PHP o usar cookies.

Finalmente me decanté por desarrollar el carrito mediante la variable $\_SESSION[‘carrito’] de PHP, esta variable es un array que permite persistir datos entre páginas, pero una vez cerrado el navegador no se conservaran los productos almacenados en el carrito. Llegué a esta conclusión valorando la cantidad de almacenamiento que se requiere si se crean tablas de cesta para cada usuario (pensando a gran escala) y no dispongo de un equipo con almacenamiento suficiente. También descarté las cookies, ya que hace más vulnerables a los usuarios para los ciberdelincuentes, estos no pueden atacar directamente a los clientes, pero sí que se podría hacer un seguimiento del navegador del usuario.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Imagen de la página del carrito de un usuario.

### Métodos de pago

Cuando el usuario desee finalizar su compra y pagar los productos que hay en su carrito, podrá escoger entre tres métodos de pago distintos. Podrá elegir entre pagar con una cuenta de PayPal, con SOFORT, o directamente con una tarjeta de crédito o débito. Esto es gracias a la *“sandbox”* (API creada para simular pagos) que proporciona PayPal Developer. La propia página proporciona el código de JavaScript a importar en la página web del desarrollador, además, viene con guías para implementar los distintos códigos dependiendo del nivel profesional que se le quiera dar. A continuación, explicaré como he ido desarrollando mi *“sandbox”* de PayPal.

En primer lugar, me tuve que crear una cuenta de PayPal como negocio, una vez hecho el registro, en la página <https://developer.paypal.com/dashboard/applications/sandbox> hay que crear una nueva aplicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen creando una aplicación en PayPal Developer

Una vez creada la aplicación (en mi caso la app se llama “Test”) se nos proporcionará un ID Cliente, un correo de empresa, y un correo de cliente que se va utilizar como usuario cliente de PayPal para realizar las pruebas.

Seguidamente, nos dirigimos a la siguiente dirección web donde nos va a venir documentada toda la información de cómo importar el código desde el cual vamos a desarrollar nuestro entorno de pago de PayPal: <https://developer.paypal.com/docs/checkout/standard/integrate/>. Yo a partir de la guía, acabé personalizando el código para adaptarlo a mi tienda web.

Primero tenemos que escribir el enlace por el cual se va a procesar la información que se vaya a cargar del carrito, en este enlace debemos declarar el ID Cliente que nos devolvió nuestra aplicación y seguidamente, gracias a la cadena “&currency=EUR” especificamos el tipo de moneda que se va a usar en nuestro método de pago.

<script src=”https://www.paypal.com/sdk/js?client-id=AXa8zonYcUJU0OjrX0e4hFX-y3sHfNRzpMxqKHyFi0wyJ4IMUZM9ZDP5M0uD0\_7pB-03cXK74eypfRFB&currency=EUR”></script>

Imagen del enlace a la aplicación de PayPal.

Antes de desarrollar el código de JavaScript con el que se crean los botones de pago, hay que especificar dónde se van a mostrar en la web, esto lo detallamos en un <div> con el id que se pase en el código de JS.

<div id="paypal-button-container"></div>

<div> donde se van a mostrar los botones de pago.

Dependiendo de si el usuario realiza la compra, esta se procesa en otra página llamada “compra.php”. En cambio, si se cancela el pago, aparece una ventana informativa diciendo que la compra de los productos se ha cancelado. Los diferentes estilos que se le pueden otorgar a los botones de pago están expuestos en la página citada anteriormente.

<script>

    /\* Mostrar los botones \*/

    paypal.Buttons({

        /\* Personalización visual de los botones \*/

        style: {

        color: 'blue',

        label: 'pay',

        shape: 'pill'

        },

        /\* Definir importe total a pagar \*/

        createOrder: function(data, actions) {

            return actions.order.create({

                purchase\_units: [{

                    amount: {

                        value: <?php echo $total; ?>

                    }

                }]

            })

        },

        /\* Se realiza la compra \*/

        onApprove: function(data, actions) {

            window.location.href="compra.php"

        },

        /\* Se cancela el pago \*/

        onCancel: function(data) {

            Swal.fire('Compra cancelada...','Se ha cerrado la operación de tu compra','error');

        }

    }).render('#paypal-button-container');

</script>

Código que procesan los botones de PayPal.

### Navegar como administrador

Los usuarios que sean administradores accederán a otra web privada donde podrán monitorizar las tablas y los registros pertenecientes a la base de datos de la tienda.

Nada más iniciar sesión se le redirigirá al usuario a otro archivo “index.php” ubicado en otro directorio, desde ahí, podrá moverse entre las distintas tablas para visualizar, agregar, editar o eliminar cualquier registro.

Tabla

Descripción generada automáticamente

Imagen de la página de inicio de un usuario administrador.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Imagen de los registros de la tabla Componentes.

## **Kubernetes**

Kubernetes ofrece un entorno de administración centrado en contenedores. Se encarga de orquestar la infraestructura de cómputo, redes y almacenamiento para que los usuarios no tengan que hacerlo.

### Conceptos teóricos

* ¿Qué es Kubernetes?: Es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Ofrece un entorno de administración centrado en contenedores. Se encarga de orquestar la infraestructura de cómputo, redes y almacenamiento para que los usuarios no tengan que hacerlo.
* Contenedor: A diferencia de una máquina virtual que proporciona virtualización de hardware, un contenedor proporciona virtualización ligera a nivel de sistema operativo mediante la abstracción del "espacio del usuario". Los contenedores comparten el núcleo del sistema host con otros contenedores. Un contenedor, que se ejecuta en el sistema operativo host, es una unidad de software estándar que empaqueta código y todas sus dependencias, para que las aplicaciones se puedan ejecutar de forma rápida y fiable de un entorno a otro. Los contenedores no son persistentes y se activan desde imágenes.
* Pods: Son los objetos más pequeños y básicos que se pueden implementar en Kubernetes. Un Pod representa una instancia única de un proceso en ejecución en un clúster. Los Pods contienen uno o más contenedores.
* Service: Es una abstracción que define un conjunto lógico de Pods y una política por la cual acceder a ellos.
* PV: Es una instancia de almacenamiento virtual que se añade como volumen al clúster.
* Control Plane (Master): Nodo desde el cual se va a gestionar el clúster, se encarga de desplegar los Pods, Services, PVC… de la *“tienda”* sobre los nodos Worker.
* Workers: Nos proporcionara un entorno de ejecución. La web y la base de datos se encuentran contenerizadas en Pods en estos nodos y son controladas por los agentes del Master.
* Kubelet: Es un agente que se encuentra corriendo en cada nodo Worker y se comunica con los componentes del nodo Master. También monitoriza la salud de los contenedores que se ejecutan en los Pods.
* Kube-Proxy: Es el agente de red que se ejecuta en cada nodo. Es el encargado de reenviar las solicitudes de conexión a los Pods.

### Creación del clúster

Para simular un clúster de Kubernetes de manera local he escogido el entorno de software KInD, este es capaz de simular los nodos como contenedores Docker. Para instalar KInD primero debemos preparar el entorno para su correcto funcionamiento, necesitaremos tener instalado:

* Git
* Go: Entorno desde donde va a correr KInD y desde donde lo vamos a descargar.
* Docker: Para poder simular el clúster.
* Kubectl: Interfaz de línea de comandos para comunicarse con el clúster de Kubernetes.
* KInD

Texto

Descripción generada automáticamente

Imagen del comando para instalar Git.

Texto

Descripción generada automáticamente

Descarga del directorio de GO.



Descomprimir el directorio de GO.



Definir variables en /home/user/.bashrc.

Texto

Descripción generada automáticamente

Instalación de Docker.



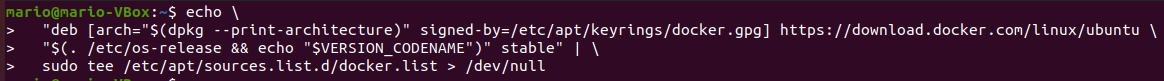
Instalación de Docker.



Instalación de Docker.



Instalación de Docker.



Instalación de Docker.

Texto

Descripción generada automáticamente

Instalación de Docker.

Texto

Descripción generada automáticamente

Instalación de Docker.



Añadir al usuario al grupo Docker para poder ejecutar comandos sin ser sudo.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza baja

Instalación del interfaz de línea de comandos Kubectl.



Habilitar autocompletado de Kubectl.

Texto

Descripción generada automáticamente

Descarga del software KInD v0.19.0 desde GO.

Texto

Descripción generada automáticamente

Creación personalizada del clúster.

Texto

Descripción generada automáticamente

Creación de clúster.

Imagen que contiene Texto

Descripción generada automáticamente

Información de clúster.

Una vez creado el clúster hay que clonar el repositorio donde tengo subido el proyecto.

Texto

Descripción generada automáticamente

Imagen de la descarga del repositorio del proyecto.

Texto

Descripción generada automáticamente

Contenido del repositorio.

### Imágenes Docker

Un Dockerfile es un fichero de texto desde el cual se ejecutan una serie de instrucciones con el fin de crear una imagen Docker personalizada, estas imágenes se crean a partir de otras imágenes Docker.

En este caso, he utilizado las imágenes oficiales de php:7.4-apache (apache2 + php v7.4) y mysql:latest (oraclelinux:8-slim).

Antes de crear imágenes hay que registrarse en <https://hub.docker.com/> para crear repositorios donde almacenar las imágenes o crear un repositorio local dentro de la máquina a partir de la imagen Docker *“Registry:2”*. Yo en particular opté por crearme una cuenta gratuita en Docker.hub ya que me parece más cómodo, se pueden manipular imágenes desde cualquier equipo y además no ocupa espacio de almacenamiento en el host.

Texto

Descripción generada automáticamente con confianza media

Imagen de inicio de sesión en Docker.hub.

Texto

Descripción generada automáticamente

Imagen del archivo dockerfile para la imagen *“proyecto:mysql-db”*.

Texto

Descripción generada automáticamente

Imagen del archivo dockerfile para la imagen *“proyecto:php-apache”*.

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

Construcción de la imagen de mysql.

Texto

Descripción generada automáticamente

Construcción de la imagen de php-apache.

Texto

Descripción generada automáticamente

Etiquetar o renombrar las imágenes creadas.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Subir las imágenes creadas al repositorio mariodsasir/proyecto en Docker.hub.

### Manifests

Dentro del directorio de manifest/ hay una serie de archivos, en estos ficheros está declarada la configuración específica del despliegue de la tienda en el clúster. A continuación, se describen brevemente dichos ficheros:

En el fichero llamado mysql-secret.yaml hay almacenados una serie de datos, la finalidad de este archivo de “secretos” es la de no mostrar contraseñas o nombres de bases de datos en este caso, pero se puede usar para esconder los datos que se deseen. Estas claves se muestran codificadas en base64 para más seguridad.

Texto

Descripción generada automáticamente

Otro fichero relacionado con la base de datos es el archivo mysql-pv.yaml. En este fichero vamos a desplegar un PersistentVolume y un PersistentVolumeClaim.

* Un PersistentVolume es por así decirlo un nexo entre un directorio el cual reside dentro del Pod y un directorio local ubicado en el nodo donde está desplegado ese Pod.
* Un PersistentVolumeClaim lo que hace es una solicitud de un recurso (modo de acceso y tamaño especificados) PersistentVolume.

De esta manera si el Pod muere o sufre daños, los datos no se perderán y a la hora de desplegar otro Pod en su lugar, este recogerá los datos guardados en el directorio /datos/ del nodo donde esté ubicado.

Texto

Descripción generada automáticamente

Para acabar de repasar los manifest relacionados con la base de datos, creamos un servicio y un deployment desde el mismo manifest.

Por un lado, definimos el servicio nombrado como mysql, a este servicio le tenemos que definir a qué grupo de Pods se refiere, esto lo hacemos dentro de spec.selector.app: twtfirst diferenciándose con la etiqueta tier: database. Además, se expone el puerto 3306 de los contenedores de los Pods pertenecientes a este servicio y se les niega una IP dentro del clúster.

Texto

Descripción generada automáticamente

Seguidamente se muestra la configuración establecida para el deployment de la base de datos de mysql, aquí se especifica y vincula todo lo descrito anteriormente. He utilizado kind: Deployment porque de esta manera se pueden actualizar los Pods sin necesidad de borrarlos o incluso realizar rollbacks a diferencia de kind: Pod.

Texto

Descripción generada automáticamente

El manifest del deployment de la web es similar al deployment de la base de datos, este se va a diferenciar, por ejemplo, en que el servicio expone el puerto 80, además, el servicio va a ser de tipo LoadBalancer (balanceo de carga) ya que la web va a estar soportada siempre por 2 Pods, esto se encuentra definido en el apartado del deployment replicas: 2.

Este servicio se refiere solo a los Pods que tengan la etiqueta tier: frontend dentro del grupo app: twtfirst.

Texto

Descripción generada automáticamente

### Despliegue de la tienda en el clúster

Ya que se han descrito los manifests, se lleva a cabo el despliegue en el clúster de KInD.

Texto

Descripción generada automáticamente

Despliegue de los manifest.

Imagen que contiene exterior, grande, cerca, tabla

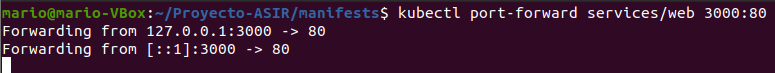
Descripción generada automáticamente

Imagen del clúster de Kubernetes.

Texto

Descripción generada automáticamente

Imagen de los datos persistentes en el nodo worker procedentes del directorio /var/lib/mysql/ del Pod de la base de datos.



Enlazar el puerto 3000 del host con el puerto 80 del servicio.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Búsqueda en el navegador a localhost puerto 3000 para visualizar la tienda web.