Extensión de la arquitectura Docker para el despliegue automático de contenedores

Extension of the Docker architecture for automatic container deployment

***Luz Elena Gutiérrez López*[1]**
***Carlos Andrés Guerrero Alarcón*[2]**

DOI: https://doi.org/10.18041/1909-2458/ingeniare.29.7432

RESUMEN

Los contenedores se han convertido es una estrategia ideal para acelerar el proceso de desarrollo de plataformas. Su importancia radica en la capacidad que tienen de separar una aplicación e interactuar con sus partes sin que la totalidad de la aplicación tenga que ser afectada. Los contenedores pueden compartir procesos entre varias aplicaciones, de manera muy similar al esquema propuesto por la arquitectura orientada a servicios. El objetivo de esta investigación fue definir una arquitectura para el despliegue automático de contenedores en contextos académicos; la verificación y validación de la arquitectura se realizó mediante la construcción de una plataforma que adapta los conceptos de la arquitectura y permite visualizar nivel a nivel cada uno de sus componentes. Se realizó un análisis bibliográfico sobre las arquitecturas propuestas para la gestión de contenedores, con lo cual se evidenciaron fortalezas y debilidades. El resultado directo de esta investigación fue la propuesta arquitectónica para el despliegue de contenedores como una extensión de Docker. El resultado indirecto fue la plataforma web con miras a la verificación y la validación de la arquitectura.

***Palabras clave*:** A dministración web; Contenedor; Desarrollo de *software*; Docker.

ABSTRACT

Containers have become an ideal strategy to speed up the platform development process. The importance of containers is the ability they must separate an application and interact with its parts, without the entire application having to be affected. Containers can share processes between multiple applications, using the same strategy used by service-oriented architecture. The aim this paper is to present a novel architecture for the automatic deployment of containers in academic contexts. The verification and validation of the architecture was carried out through the construction of a platform that adapts the concepts of the architecture and allows to visualize each of its components level by level. In this study, a bibliographic analysis was carried out on the architectures that manage containers, showing strengths and weaknesses. The principal result of this research was the proposal of novel architecture for the deployment of containers as extension of Docker. The secondary result was the development of a web platform, this platform allows to present all characteristics of the novel approach. It was used for the verification and validation of the architecture.

***Keywords*:** Web management; Container; Software development; Docker.

Como citar este artículo: G. L. Luz Elena y G. A. Carlos Andrés, Extensión de la arquitectura Docker para el despliegue automático de contenedores, ingeniare, vol. 2, n.º 29, dic. 2020.

1. Magíster en Ingeniería Área Informática y Ciencias de la Computación, Ingeniera de Sistemas. Universidad del Norte, Colombia, Correo: egluz@uninorte.edu.co
2. Ph. D. en Ingeniería, Magíster en Informática, Ingeniero de Sistemas. Universidad Santo Tomás, Colombia, Correo: carlos.guerrero@usantoto.edu.co

Fecha de recepción: 5 de enero de 2020 • Fecha de aceptación: 25 de marzo de 2020

1. INTRODUCCIÓN

El diseño y la programación de algoritmos definen las bases y las competencias que debe adquirir un ingeniero de desarrollo, por tanto, los cursos de programación son de vital importancia en el proceso de formación [1]. Las habilidades en programación que un estudiante adquiere en su proceso de formación están calificadas como las competencias requeridas en el siglo XXI [2], lo que realza la importancia de aplicar un proceso de enseñanza aprendizaje que asegure la adquisición de competencias en el área de programación.

Facilitar la enseñanza de la programación es una actividad que no solo requiere conocimiento técnico por parte del docente, sino habilidades para motivar a los estudiantes a superar los obstáculos que se presenten en su formación [3]. Un estudiante frustrado, probablemente, incrementará las estadísticas de deserción [1].

El proceso de formación de un desarrollador de *software* requiere de tiempo y dedicación, y existen múltiples problemáticas que inciden en el éxito o el fracaso [4]. Entre estas problemáticas se encuentran la formación y la experticia del docente, los conocimientos previos que debe tener el estudiante, así como los recursos tecnológicos a los que tiene acceso el desarrollador, entre otros [5].

En [6]critical thinking and collaboration or recognized as Higher Order Thinking Skills (HOTS se plantea que los procesos actuales de enseñanza son ineficientes, lo cual crea en el estudiante una resistencia que, eventualmente, se transformará en miedo a la programación [7], [8]. Existe evidencia de trabajos —como, por ejemplo, [9]— en los que utilizan otras estrategias para enseñar programación (p. ej., el caso de los videojuegos).

Este documento presenta una arquitectura soportada en Docker [10] [11] para el despliegue automático de contenedores en procesos de desarrollo *software* orientados a la web. El enfoque de este proyecto busca proporcionar herramientas que permitan emplear estrategias diferentes dirigidas a que el estudiante en formación del área de sistemas esté en capacidad de construir aplicaciones.

El artículo está organizado de la siguiente forma: inicia con una revisión de literatura acerca de arquitecturas para el despliegue de contenedores, posteriormente, presenta la arquitectura utilizada como línea de base en la presente investigación. Continua con la descripción de la arquitectura propuesta para el despliegue automático de contenedores en contextos académicos. Finaliza con el planteamiento de las limitaciones de la presente investigación y las conclusiones del estudio.

2. REVISIÓN DE LITERATURA

La Tabla 1 presenta una descripción de arquitecturas para la gestión de contenedores y su comparación con la propuesta de arquitectura de este artículo.

Tabla 1. Comparación de estudios

Investigación	Descripción	Comparativo
"Automatic programming assignment checker" [5]	Este trabajo utiliza la arquitectura de Docker con el fin de encapsular el proceso de verificación de código en asignaturas, en el cual el estudiante debe entregar fuentes para verificación de calidad. Los contenedores propuestos son estáticos y utilizan un componente que se encarga de la lógica del negocio en el proceso de validación y compilación de código.	Tanto el trabajo de [5]lecturers should be exempted from routine tasks like source code compilation, testing and grading. Current computers are equipped with enough computational power to automate these routine tasks. This paper discusses the analysis and realization of such a system for user submitted automatic source code evaluation. The main system requirement was the safe runtime environment (sandbox como la presente investigación toman como referencia la arquitectura de Docker. No obstante, la solución propuesta en este artículo permite la instalación de múltiples contenedores en tiempo de ejecución. Además, no requiere conceptos técnicos para el usuario final. Incluso el componente de verificación de código propuesto en [5 podría ser utilizado en la presente investigación, de manera que se evita el montaje de todo el host de Docker.
"Docker cluster management for the cloud" [12]	Esta investigación utiliza un caso de estudio con la finalidad de presentar las ventajas de Docker en un ambiente de desarrollo. El valor agregado radica en el uso de BPEL (business process execution language) para la orquestación de todos los contenedores.	El trabajo en [12] y el propuesto en este artículo plantean el uso de Docker como una herramienta dirigida a facilitar y orquestar contenedores. La diferencia con la presente propuesta es el enfoque de la solución propuesta, puesto que buscan medir y analizar el impacto de la dockerización en la nube. Los autores indican la necesidad que tiene Docker de seguir en constante evolución, y plantean la importancia de realizar trabajos que permitan automatizar el uso de contenedores.
Self-hosted Kubernetes [13]	Los autores expresan la importancia de utilizar Docker para gestionar contenedores. Adicionalmente, plantean el dilema entre realizar múltiples virtualizaciones en una máquina o crear contendedores con acceso entre ellos. Con respecto a la gestión de contendedores proponen el uso de Kubernates, sin embargo, hacen énfasis en el nivel de conocimiento que debe tener el usuario para determinar las configuraciones precisas a la hora de establecer un ambiente de pruebas o de producción.	El aporte de nuestra investigación está enfocado en el no uso de Kubernates para gestionar contenedores; por el contrario, se propone una arquitectura con el fin de administrar contenedores sin necesidad que el usuario final tenga conocimientos técnicos.

Fuente: elaboración propia.

3. ARQUITECTURA DE REFERENCIA

El desarrollo de aplicaciones robustas es un requerimiento fundamental en la industria del *software* [14] geographically distributed development teams, new business models and diverse cultural interactions steer these tools. Software development supported by web-based services, built on top of Web 2.0 technologies, is emerging as a new paradigm for distributed software development. New generation software forges (web-based development environments. Una buena práctica para el desarrollo de *software* es planear y realizar seguimiento a cada fase del ciclo de vida del aplicativo. Por tal motivo, es importante el uso de herramientas que apoyen la producción y la puesta en marcha del *software*. La arquitectura de referencia utilizada en esta investigación fue Docker [10] Docker agiliza la configuración de los equi-

pos de desarrollo, de modo que permite la virtualización de un servidor en muchos espacios aislados de trabajo y compartir recursos en una máquina anfitriona. Adicionalmente, Docker permite encapsular entornos y aplicaciones que, posteriormente, pueden ser utilizadas en otros espacios de trabajo, lo que facilita el uso y la configuración de las herramientas.

La arquitectura cliente servidor de Docker se presenta en la Figura 1. La capa cliente puede ejecutar instrucciones directamente sobre el anfitrión de Docker para descargar y desplegar imágenes y contenedores. En el Docker Host existe un Docker Daemon; este se encarga de controlar las imágenes y los contenedores que un cliente puede generar [15]. Toda la comunicación entre cliente y servidor se realiza a través de *sockets* utilizando un API de tipo Restfull [11].

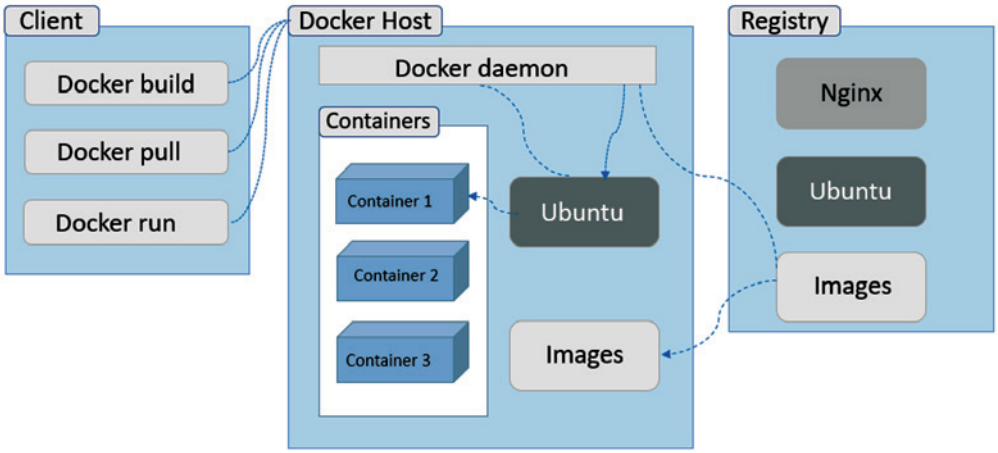


Figura 1. Arquitectura Docker

Fuente: Adaptada de [16]

El Docker Daemon controla las imágenes a través del Docker registry, el cual es el encargado de organizar y publicar las imágenes que un usuario puede llegar a utilizar. Cada contenedor se crea a partir de una imagen y es un entorno aislado y seguro en el que se ejecuta la aplicación [17].

Docker administra espacios de nombres, dispositivos de red, cortafuegos y, en general, los recursos del sistema operativo a través de *libcontainer*. La administración de los contenedores se realiza al dar cumplimiento al estándar Open Container Initiative-OCI [18], [19].

El despliegue de la arquitectura de Docker se puede realizar sobre un servidor Linux o sobre hipervisores tipo I y II. Los hipervisores tipo I son aquellos que se utilizan a través de *hardware*, y se caracterizan por ser rápidos y costosos. Los hipervisores tipo II no son tan rápidos como los hipervisores tipo I, sin embargo, son económicos e, incluso, algunos se pueden utilizar de forma gratuita [20].

4. ARQUITECTURA PROPUESTA

Utilizar Docker para construir una nueva arquitectura requiere tres aspectos fundamentales: 1) la definición de los elementos diferenciadores de las dos arquitecturas; 2) la construcción de los componentes que permiten que la arquitectura se despliegue en un entorno de desarrollo; y 3) la ejecución de pruebas con el fin de validar la implementación de la nueva arquitectura.

4.1 Diseño de la arquitectura

La solución tecnológica propuesta en esta investigación se compone de una arquitectura base y una plataforma para la gestión automática de imágenes y contenedores. La arquitectura base reutiliza la totalidad de la arquitectura Docker y añade dos artefactos que permiten la automatización en contextos académicos. La Figura 2 describe la arquitectura base propuesta; en color verde se presentan los elementos diferenciadores con relación a la arquitectura Docker: *middleware* y *primary storage of containers*.

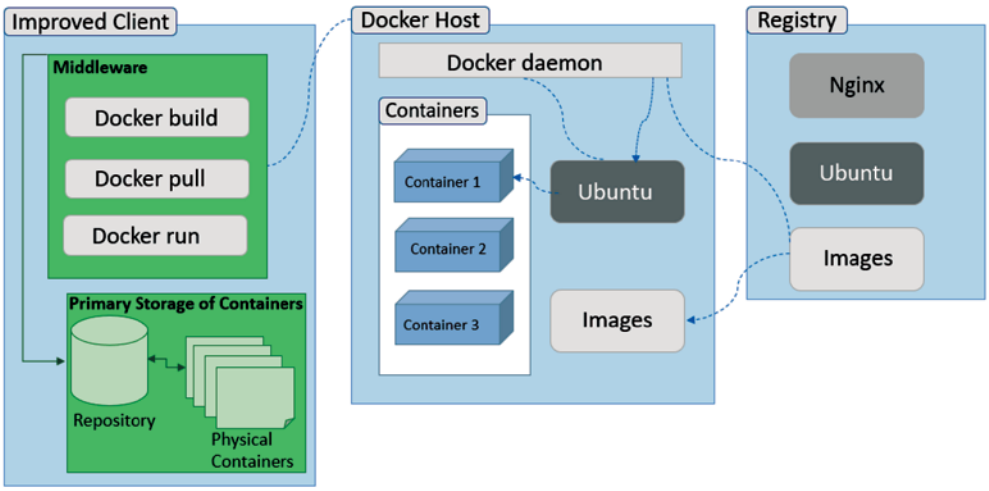


Figura 2. Arquitectura propuesta

Fuente: elaboración propia.

El cliente Docker permite la ejecución de tres instrucciones que se encargan de realizar la descarga de las imágenes. En la arquitectura base propuesta es posible apreciar una capa que cubre al cliente, la cual automatiza este proceso. Esta capa es un *middleware* [21] que gestiona una plataforma Web, la cual permite el uso y la administración de contenedores a usuarios que están en proceso de aprendizaje en contextos de desarrollo académico.

La siguiente capa de la arquitectura está ubicada en el cliente y se denomina “almacenamiento primario de contenedores”. La arquitectura define este almacenamiento como un espacio de nombres y un conjunto de repositorios que permiten la identificación, selección y búsqueda de los contenedores que

puedan ser definidos por los usuarios. La implementación de la arquitectura en la plataforma utiliza el concepto de identificadores únicos para el almacenamiento primario, con lo cual un usuario puede llegar a crear muchos contenedores con el mismo nombre.

La arquitectura propuesta encapsula los conceptos de Docker, de modo que se logra una abstracción total para el usuario. Como resultado de esta abstracción un usuario no necesita tener conocimientos técnicos sobre Docker para implementar la arquitectura desarrollada. A continuación, se presentan los artefactos que hacen parte de la arquitectura y de la plataforma desarrollada utilizada para su verificación y validación.

4.1.1 Artefactos de la solución

- *Primer artefacto.* Servidor Linux con distribución Ubuntu en su versión 19.04 [22]. El *hardware* para la implementación estaba soportado con un procesador Intel(R) Core (TM) i5-8250U, con 8 gigas de RAM y conexiones ethernet.

- *Segundo artefacto: Docker Community Edition CE*. Lo utilizan desarrolladores nuevos en el uso de la tecnología o grupos pequeños de desarrollo. El canal de actualización para la descarga fue “Stable”, ideal para para personas que necesitan trabajar con contenedores sin errores [16].

- *Tercer artefacto.* La plataforma web desarrollada con base en los lineamientos definidos en la arquitectura propuesta. La implementación se realizó con base en patrones arquitectónicos y de diseño, buscando simplificar la funcionalidad, mantenibilidad y escalabilidad del sistema [23]. En el desarrollo de la aplicación web se utilizaron los patrones creacionales: Factory Method, Prototype y Singleton. Los patrones estructurales: Decorator y Facade, y los patrones de comportamiento Iterator y Template [24].

- *Cuarto artefacto: Docker registry*. Fue utilizado con el fin de acceder a los índices de las imágenes y presentar a los usuarios un listado de componentes reutilizables susceptibles de ser seleccionados sin conocimientos técnicos. La arquitectura puede implementar cualquier tipo de contenedor almacenado en Docker, sin embargo, en la plataforma se personalizaron dos contenedores para facilitar la comprensión de la arquitectura: 1) contenedores para bases de datos MySql [25] y 2) contenedor para servidores Web apache.

4.2 Implementación de la arquitectura

La arquitectura requiere de una plataforma web para su despliegue y validación. A continuación, se presenta la configuración base requerida por la plataforma, la descarga de imágenes de prueba, la inclusión de las imágenes en el repositorio de la arquitectura y el ejemplo de funcionamiento de la plataforma Web.

4.2.1 Instalación y configuración de componentes básicos

1. Instalación de la máquina virtual Oracle VM Virtual Box.

2. Definir distribución de Linux. Es importante definir qué distribución de Linux utilizar. Según [26], las versiones más estables de Linux son: Redhat, Debian, Ubuntu, Linux Mint, Fedora y MX Linux. La versión que se utilizó para este caso de estudio fue Ubuntu (Server 19.04 Disco Dingo), por las siguientes razones:

- Distribución sencilla y con un repositorio de paquetes amplio.
- Docker tiene paquetes preconfigurados para Ubuntu.
- Sistema operativo actualizado cada dos meses y con una frecuencia de soporte a largo plazo (dos años).
- Es una distribución utilizada en contextos académicos.

3. Instalación de Docker.

4. Descarga y sincronización del repositorio base de la arquitectura. En este punto se realizó la descarga de imágenes de prueba tales como: MySql, PHP, Java y Python, entre otras.

5. Creación del contenedor MySql a partir de la imagen almacenada en el repositorio de la arquitectura. Este contenedor es el que permite el almacenamiento de usuarios desde la plataforma que gestiona la arquitectura. La plataforma provee una interfaz de usuario final.

6. Configuración del archivo Dockerfile para el contenedor que almacena los usuarios de la plataforma, tal como se presenta en la Figura 3. Los Dockerfile son útiles para usuarios que no tienen conocimiento en la configuración de archivos de Docker [27].

```
1   FROM mysql:latest
2   ENV MYSQL_ROOT_PASSWORD clavemysql
3   ENV MYSQL_USER usuariophp
4   ENV MYSQL_PASSWORD clave123
5   EXPOSE 3306
```

Figura 3. Contenido Dockerfile mysql

Fuente: elaboración propia.

La instrucción “FROM” especifica la imagen que se desea descargar, en este caso se solicita la última versión de MySql. La instrucción “ENV” permite inicializar una variable que puede estar definida dentro de la imagen, en este caso se inicializó la variable “MYSQL_ROOT_PASSWORD” con una contraseña de ejemplo. La instrucción “EXPOSE” permite definir el puerto sobre el cual los contenedores de esta imagen prestarán sus servicios.

7. Creación del contenedor Apache-PHP a partir de la imagen almacenada en el repositorio de la arquitectura. La plataforma desarrollada que soporta la arquitectura fue construida en el lenguaje de programación PHP.

8. En la Figura 4 se presenta la configuración del Dockerfile para el contenedor Apache-PHP.

```
1   FROM php:7.2-apache
2
3   RUN apt-get update && apt-get install -y wget zip unzip
4       libfreetype6-dev libzip-dev
5   RUN apt-get update && apt-get install -y zliblg-dev
6   RUN apt-get install && docker-php-ext-configure zip
7       --with-libzip && docker-php-ext-install zip
8   RUN apt-get update && docker-php-ext-install pdo_mysql
9   ENV APACHE_RUN_USER www-data
10  ENV APACHE_RUN_GROUP www-data
11  ENV APACHE_LOG_DIR /var/log/apache2
12  ENV APACHE_PID_FILE /var/run/apache2/apache2.pid
13  ENV APACHE_RUN_DIR /var/run/apache2
14  ENV APACHE_LOCK_DIR /var/lock/apache2
15  EXPOSE 8086:80
16  CMD ["apache2-foreground"]
```

Figura 4. Contenido Dockerfile PHP

Fuente: elaboración propia.

La instrucción “FROM” especifica la imagen que se desea descargar, en este caso se solicita la versión 7.2 de PHP con Apache. Adicionalmente, se incluyen las instrucciones de descargar las librerías necesarias para descomprimir [28]. Posteriormente, se definen las variables básicas para la instalación del apache y se presenta el puerto que será expuesto a fin de consumir los servicios del contenedor.

4.3 Modelo vista controlador de la plataforma

La Figura 5 presenta la estructura de ficheros utilizada en el componente de administración de contenedores de la plataforma desarrollada.

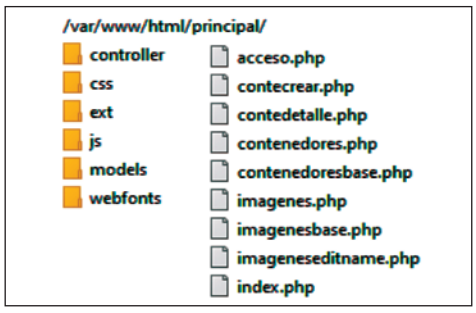


Figura 5. Modelo vista controlador del componente administrador

Fuente: elaboración propia.

A continuación, se describen tres ficheros de código que hacen parte de la plataforma desarrollada. La Figura 6 presenta el sistema de conexión de la base de datos a través de PDO [29].

```php
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
$direccionIP = "192.168.43.204";

try {
    $usuario = "user_php";
    $clave = "clave123";
    $conn = new PDO('mysql:host=localhost;dbname=bd_principal', $usuario,
                $clave);
} catch (PDOException $e) {
    print "<strong>Error:</strong><br />" . $e->getMessage();
    exit();
}
?>
```

Figura 6. Contenido conexion.php

Fuente: elaboración propia.

Como se describió, la arquitectura permite el uso de cualquier contenedor. En la plataforma desarrollada este proceso se automatiza mediante la ejecución de instrucciones parametrizadas que aíslan al usuario final de la problemática de instalación y configuración. La Figura 7 presenta las instrucciones que hacen parte de uno de los controladores para la creación de contenedores.

```php
$comando = "docker run -d -p {$extra}:80 -v /home/osboxes/proyecto/sitioweb/"
. "{$carpeta}/:/var/www/html --name {$nombreContenedor} "
. "{$nombreImagen}:{$versionImagen}";
$ejecutar = shell_exec($comando);

$contenedores = shell_exec('docker ps -a --format "table{{.ID}}@{{.Names}}@'
        . '{{.Image}}@{{.RunningFor}}@{{.Status}}@{{.Ports}}"');
$arrContenedores = explode("\n", $contenedores);
$limite = count($arrContenedores) - 1;
```

Figura 7. Contenido contecreacontroller.php

Fuente: elaboración propia.

A fin de visualizar los contenedores se sigue el mismo principio: aislar la complejidad al usuario final y ejecutar instrucciones para presentar los resultados finales. La Figura 8 presenta las instrucciones utilizadas en el propósito de visualizar los contenedores en las vistas de usuario final.

```php
<?php
$control = 0;
$arrIdImagen = [];
$contenedores = shell_exec('docker images --format "table{{.ID}}'
        . '@{{.Repository}}@{{.Tag}}@{{.Digest}}@{{.CreatedSince}}'
        . '@{{.CreatedAt}}"');
$arrContenedores = explode("\n", $contenedores);
foreach ($arrContenedores as $indice => $fila) {
    $temporal = explode("@", $fila);
    $arrIdImagen[$control]["ID"] = $temporal[0];
    if (isset($temporal[4])){
        $arrIdImagen[$control]["CREADO"] = $temporal[4];
    }else{
        $arrIdImagen[$control]["CREADO"] = "";
    }
    $control++;
}
```

Figura 8. Contenido imágenes base.php

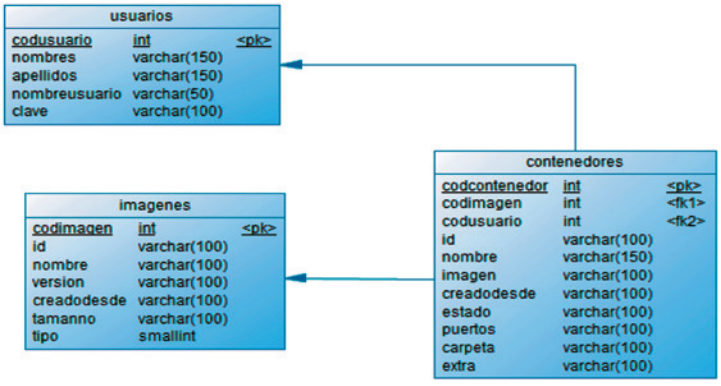
Fuente: elaboración propia.

La arquitectura propuesta fue verificada y validada por la plataforma desarrollada. A continuación, se presentan las funcionalidades de la plataforma, las cuales brindan una comprensión mayor de la arquitectura.

4.2.2 Funcionalidades plataforma

La plataforma desarrollada para la gestión de los contenedores cuenta con un sistema de autenticación y autorización [30]. Por defecto, un usuario requiere un navegador para hacer uso de la plataforma, un usuario y una contraseña que le permiten acceder al componente que administra y gestiona los contendedores. Si el usuario desea crear un contenedor nuevo lo puede realizar a partir de una imagen descargada o, en su defecto, tener acceso a la lista de imágenes que le permitirán acceder a más contenedores sin tener conocimiento técnico de Docker.

La Figura 9 presenta el modelo de clases implementado en la autenticación, autorización, gestión de contenedores y descarga de imágenes.



9. Modelo de clases plataforma

Fuente: elaboración propia.

La clase usuarios representa los posibles desarrolladores que tendrán permisos para gestionar imágenes y contenedores en la plataforma desarrollada. La clase imágenes representa las imágenes almacenadas en la plataforma Docker, las cuales han sido descargadas con anterioridad del sitio oficial. La clase contenedores es una abstracción de los posibles contenedores que puede llegar a crear un usuario a través de la plataforma.

La plataforma desarrollada cuenta con las funcionalidades descritas en la Tabla 2.

Tabla 2. Descripción de funcionalidades

Código	Funcionalidad	Descripción
f01	Manejo de usuarios	Sistema dirigido a administrar la autorización y autenticación de usuarios. Utiliza funciones de tipo Hash para la seguridad de la información.
f02	Lista de imágenes	Permite el despliegue de las imágenes que la nueva plataforma puede llegar a gestionar.
f03	Edición/eliminación de imágenes	Este componente permite eliminar o agregar imágenes a la biblioteca base de la arquitectura propuesta.
f04	Creación de contenedores	Permite instanciar las imágenes que existen en la biblioteca, así como manejar instancias con el mismo nombre, dado que el componente de autorización permite el aislamiento de la data a cada uno de los perfiles de usuario.
f05	Listado de contenedores	Permite visualizar los contenedores desde dos vistas: modo administrador y modo cliente. Asimismo, la información de cada usuario en su espacio de trabajo.
f06	Iniciar/detener contenedor	Este disparador permite visualizar, iniciar o detener un contendor que ha sido instanciado.
f07	Eliminar contenedor	Permite eliminar un contenedor de un usuario particular o, en su defecto, eliminar los contenedores de múltiples usuarios en modo administrador.
f08	Crear base de datos	Con el contenedor instanciado de una de las imágenes del repositorio el usuario puede crear bases de datos en cada uno de los contenedores que utilice.
f09	Interprete de código DDL	Este interprete se encarga de verificar la estructura de las sentencias de creación de tablas en un motor específico. El usuario puede crear tablas en una base de datos Postgres, MySQL o la que seleccione, sin necesidad de interactuar con el motor directamente.
f10	Ver tablas base de datos	Permite obtener un listado de las tablas de cualquier base de datos existente en cada uno de los contenedores.
f11	Subir sitio web	Permite integrar al contenedor web seleccionado, un sitio web externo, ya sea estático o dinámico. En el caso de los sitios web dinámicos permite la interconexión con otros contenedores que gestionen bases de datos.
f12	Desplegar sitio web	El despliegue del sitio se da por un puerto preconfigurado, sin embargo, esta configuración se puede cambiar por medio de un asistente.
f13	Interfaz web para la gestión de contenedores	Permite la administración de contenedores a través de un navegador web mediante el protocolo http.

Fuente: elaboración propia.

A fin de acceder a la plataforma un usuario debe tener credenciales de acceso (f01). La plataforma verifica las credenciales de un usuario, posteriormente, le permite visualizar las imágenes que tiene disponible para su uso, tal como se presenta en la Figura 10 (f02, f03, f05, f06, f07). En cada imagen brinda opciones para crear contenedores (f04) y ver detalles de las imágenes de acuerdo con la información suministrada por el Docker registry. Dependiendo de la imagen a utilizar, se presentan interfaces personalizadas, lo que permite la creación de contenedores con información mínima y sin conocimientos sobre los aspectos técnicos de la filosofía de contenedores.

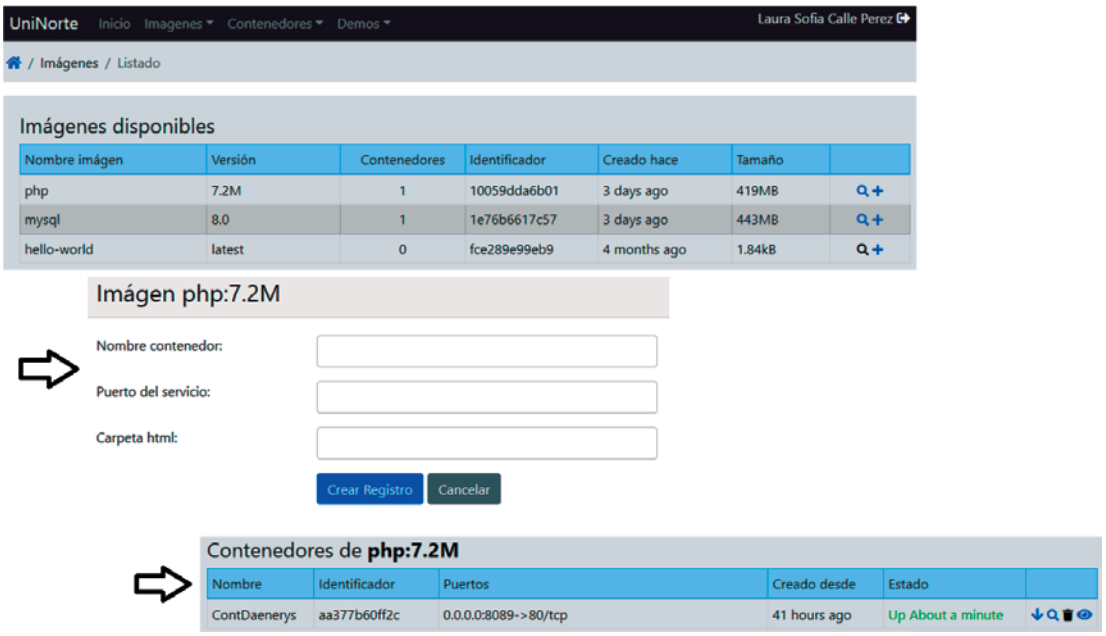


Figura 10. Creación contenedores

Fuente: elaboración propia.

La creación de un contenedor a través de la plataforma solicita tres valores: nombre del contenedor, puerto sobre el cual desea exponer los servicios y, por último, el nombre del espacio de nombres que el usuario desea manejar para el almacenamiento de sus archivos. Los contenedores creados son de propiedad del usuario en sesión, es decir, que al ingresar con otro usuario no aparecerán esos contenedores.

Como se describió, los contenedores tienen propietarios y estos usuarios pueden realizar múltiples funciones con sus contenedores. La plataforma provee opciones para iniciar o detener los servicios de los contenedores (f06).

La Figura 11 presenta la interfaz que permite manipular un contenedor de mysql. En cada contenedor se pueden crear bases de datos (f08). Para cada base de datos existe un intérprete (f09) que permite utilizar el lenguaje de definición de datos DDL.

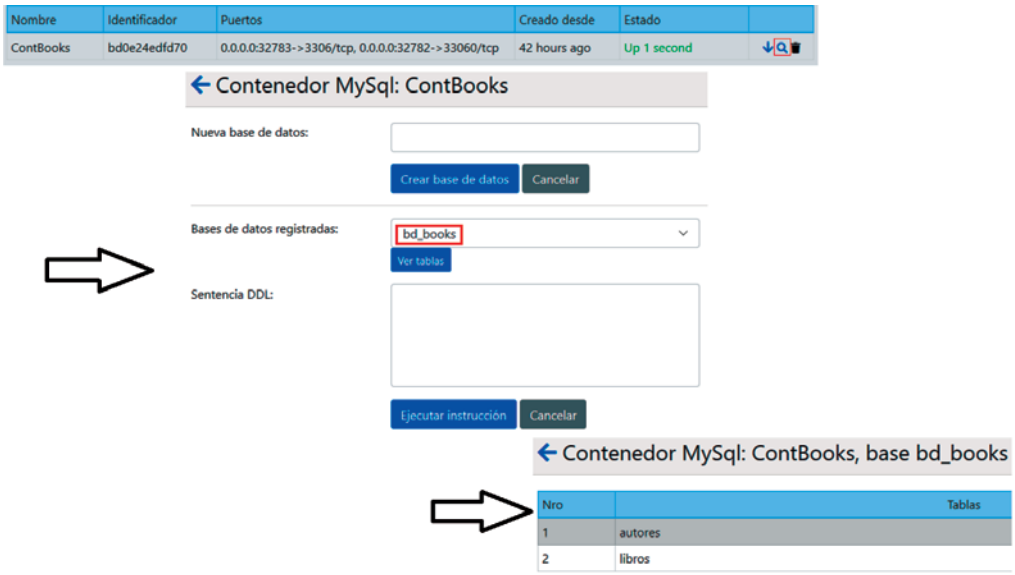


Figura 11. Administración contenedor mysql

Fuente: elaboración propia.

En la Figura 11 se presenta la forma como se pueden visualizar las tablas de cada base de datos (f10).

La Figura 12 describe la interfaz para subir sitios web a un contenedor (f11, f12). Los archivos deben estar en formato zip.

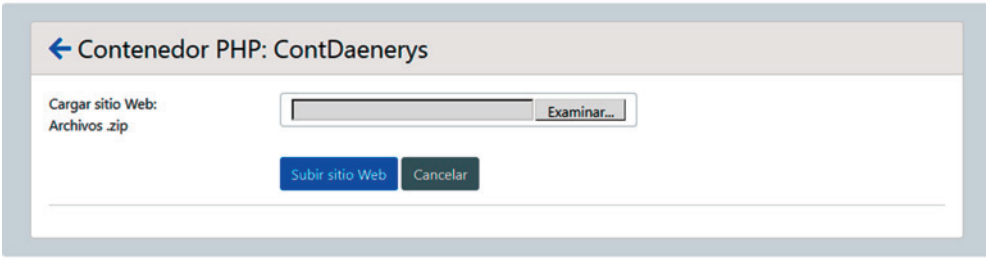


Figura 12. Formulario subir sitio

Fuente: elaboración propia.

De acuerdo con [12], la plataforma web desarrollada (f13) cumple con el 65 % de los requerimientos funcionales que debería tener un sistema de administración de contenedores. El 35 % de los requerimientos no desarrollados están orientados hacia el balanceo de carga y la transferencia de contenedores entre diferentes *hosts*, sin embargo, ese 35 % no hace parte del alcance de la presente investigación.

5. LIMITACIONES

- La plataforma desarrollada requiere la implementación de funcionalidades adicionales, como, por ejemplo, la personalización de otros tipos de contenedores.

- Está orientado solo a contextos académicos en los que las personas están en proceso de aprendizaje para entornos web.

- La degradación de un contenedor afecta los demás contenedores, por tanto, se debe utilizar un sistema de balanceo a fin de que la plataforma detecte la falla y la corrija de manera automática, tal como lo realiza Kubernates.

- La cantidad de contenedores y procesos que puede llegar a crear un usuario son ilimitados, por tanto, se debe gestionar una cuota para cada sesión de usuario.

6. CONCLUSIONES

- Docker no es una herramienta dirigida a orquestar contenedores, se utiliza para gestionarlos en un entorno particular. Es importante contar con una suite de administración de contenedores. Existen herramientas como Kubernates, un orquestador de contenedores construido por Google que tiene por objeto crear aplicaciones distribuidas soportadas en contenedores. El uso y la administración de Kubernates requiere de conocimiento técnico o personal con capacitación en el tema.

- Docker Swarm es otra herramienta para organizar clústeres de contendedores. La diferencia con Kubernates radica en su enfoque: Swarm busca extender el uso de la API de Docker a fin de que todos los contenedores se vean como una sola unidad. Existen otras herramientas con fines similares, sin embargo, ninguna de ellas está enfocada en procesos de formación de profesionales.

- Docker permite extender su arquitectura mediante la adición de componentes, los cuales componentes pueden estar encapsulados en un *middleware*. A través del *middleware* se puede realizar la ejecución de código desde una interfaz web hasta el sistema operativo base que contiene el *host* Docker.

- El proceso de verificación y validación de la arquitectura se llevó a cabo con la implementación de la plataforma, teniendo un marco controlado de usuarios que permita garantizar la aplicación de todos los conceptos de la arquitectura.

- La arquitectura propuesta puede utilizarse en entornos de capacitación para formación de desarrolladores junior, puesto que permite encapsular la complejidad de Docker y, al mismo tiempo, utilizar la potencia de Docker.

REFERENCIAS

[1] V. F. Martins, I. de Almeida Souza Concilio, y M. de Paiva Guimarães, "Problem based learning associated to the development of games for programming teaching", *Comput. Appl. Eng. Educ.*, vol. 26, n.º 5, pp. 1577-1589, sep. 2018.

[2] S. Popat and L. Starkey, "Learning to code or coding to learn? A systematic review", *Comput. Educ.*, vol. 128, pp. 365-376, en. 2019.

[3] S. Azmi, N. A. Iahad y N. Ahmad, "Attracting students' engagement in programming courses with gamification", in *2016 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)*, 2016, pp. 112-115.

[4] I. Hadar, "When intuition and logic clash: the case of the object-oriented paradigm", *Sci. Comput. Program.*, vol. 78, n.º 9, pp. 1407-1426, sep. 2013.

[5] F. Špaček, R. Sohlich y T. Dulík, "Docker as platform for assignments evaluation," *Procedia Eng.*, vol. 100, pp. 1665–1671, en. 2015.

[6] M. E. Ismail, N. Sa'adan, M. A. Samsudin, N. Hamzah, N. Razali y I. I. Mahazir, "Implementation of the gamification concept using Kahoot! among TVET students: an observation", *J. Phys. Conf. Ser.*, vol. 1140, n.º 1, p. 012013, dic. 2018.

[7] A. Draz, S. Abdennadher y Y. Abdelrahman, "Kodr: a customizable learning platform for computer science education," En *DM Review*, vol. 2, pp. 579-582, sep. 201.

[8] S. P. Sarkar, B. Sarker y S. K. A. Hossain, "Cross platform interactive programming learning environment for kids with edutainment and gamification", En *2016 19th International Conference on Computer and Information Technology (ICCIT)*, 2016, pp. 218-222.

[9] C. A. Guerrero Alarcon, L. E. Gutiérrez López y K. D. Cuervo Cely, "Los videojuegos como estrategia para incrementar la motivación y alcance de logros en procesos de aprendizaje," En *Memorias la formación de ingenieros: un compromiso para el desarrollo y la sostenibilidad*, 2020.

[10] H. Zhu y I. Bayley, "If Docker is the answer, what is the question?," En *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2018, pp. 152-163.

[11] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi y H. C. Gall, "An empirical analysis of the Docker container ecosystem on gitHub," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 323-333.

[12] R. Peinl, F. Holzschuher y F. Pfitzer, "Docker cluster management for the cloud-survey results and own solution", *J. Grid Comput.*, vol. 14, n.º 2, pp. 265-282, jun. 2016.

[13] R. Muddinagiri, S. Ambavane y S. Bayas, "Self-hosted Kubernetes: deploying Docker containers locally with Minikube," En *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*, 2019, pp. 239-243.

[14] R. Fernández, J. Soriano, X. Larrucea, A. L. Martínez y J. M. González-Barahona, "Towards the improvement of the software quality: an Enterprise 2.0 architecture for distributed software developments," En *2008 First International Conference on Distributed Framework and Applications*, 2008, pp. 52-59.

[15] Preeth E N, F. J. P. Mulerickal, B. Paul y Y. Sastri, "Evaluation of Docker containers based on hardware utilization," En *2015 International Conference on Control Communication & Computing India (ICCC)*, nov. 2015, pp. 697-700.

[16] I. Docker, "Overview of Docker editions," 2019. [En línea]. Disponible en: https://docs.docker.com/engine/docker-overview/.

[17] B. Kavitha y P. Varalakshmi, "Performance analysis of virtual machines and Docker containers," En *Communications in Computer and Information Science*, vol. 804, 2018, pp. 99-113.

[18] S. Fu, J. Liu, X. Chu y Y. Hu, "Toward a Standard interface for cloud providers: the container as the narrow waist," *IEEE Internet Comput.*, vol. 20, n.º 2, pp. 66–71, mar. 2016.

[19] T. L. F. Projects, "Open container initiative," 2016. [En línea]. Disponible en: https://www.opencontainers.org/.

[20] R. Morabito, J. Kjallman y M. Komu, "Hypervisors vs. Lightweight virtualization: a performance comparison," En *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386-393.

[21] U. Batra y S. Mukharjee, "Enterprise application integration (middleware): Integrating stovepipe applications of varied enterprises in distributed middleware with service oriented architecture," En *2011 3rd International Conference on Electronics Computer Technology*, 2011, vol. 5, pp. 226-230.

[22] Osboxes, "Ubuntu server," 2019. [En línea]. Disponible en: https://www.osboxes.org/ubuntu-server/

[23] A. Singh, P. Chawla, K. Singh y A. K. Singh, "Formulating an MVC framework for web development in JAVA," En *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2018, Icoei, pp. 926-929.

[24] C. A. Guerrero Alarcón, J. M. Suárez Pedrazay L. E. Gutiérrez López, "Patrones de diseño GOF (The Gang of Four) en el contexto de procesos de desarrollo de aplicaciones orientadas a la web", *Inf. tecnológica*, vol. 24, n.º 3, pp. 103-114, 2013.

[25] R. Senington, B. Pataki y X. V. Wang, "Using docker for factory system software management: Experience report", *Procedia CIRP*, vol. 72, pp. 659-664, 2018.

[26] Masgnulinux, "Distribuciones más estables: 5 versiones de GNU/Linux que recomendamos", 2018. [En línea]. Disponible en: https://masgnulinux.es/distribuciones-mas-estables-5-versiones-de-gnu-linux-que-recomendamos/.

[27] J. Cito y H. C. Gall, "Using docker containers to improve reproducibility in software engineering research", En *Proceedings of the 38th International Conference on Software Engineering Companion-ICSE '16*, 2016, pp. 906-907.

[28] R. Avellaneda, S. Cabrera, P. A. Martínez, y C. G. Donoso Albarracín, "Apoyo tecnológico para la fidelización y captación de nuevos clientes por medio de una aplicación móvil", *Inv. Inn. Ing.*, vol. 5, n.º 1, pp. 92-101, jul. 2017.

[29] T. P. Group, "PHP.NET," 2019. [En línea]. Disponible en: https://php.net/manual/es/intro.pdo.php.

[30] J. M. Suárez Pedraza y L. E. Gutiérrez López, "Tipificación de dominios de requerimientos para la aplicación de patrones arquitectónicos," *Inf. Tec.*, vol. 27, n.º 4, pp. 193-202, 2016.