



ACTIVIDAD 1 SEMANA 3 – INTEGRACIÓN CONTINUA

Martha Yaneth Mera Velasco  
Erick Alexander Salamanca Borda  
Jesús David Mendivelso Lizarazo

Tutor  
Natalia Martínez

POLITÉCNICO GRANCOLOMBIANO  
FACULTAD INGENIERÍA - INGENIERÍA DE SOFTWARE  
BOGOTÁ D.C  
NOVIEMBRE 2024



## Tabla de contenido

INTRODUCCION.....	3
DESARROLLO ACTIVIDAD 1 .....	3
1. Contextualización.....	3
1.1 Github. ....	3
1.2 Contenedores.....	3
2. Creación de repositorio en GitHub .....	4
3. Creación de contenedores .....	5
4. Comunicación entre los contenedores .....	8
CONCLUSIONES.....	10
BIBLIOGRAFIA.....	10

## INTRODUCCION

En el desarrollo de software actual, la agilidad y la colaboración son claves para mantener la calidad y la velocidad en cada proyecto. La integración continua es una práctica que nos permite automatizar la entrega de código, permitiendo que las actualizaciones sean constantes, seguras y eficientes. Dos herramientas esenciales en este proceso son los controladores de versiones y los contenedores, que facilitan el control y la escalabilidad del desarrollo. En este documento, se explicará la forma de utilizar las herramientas de GitHub y el uso de los contenedores los cuales contribuyen a una integración continua eficiente, promoviendo un flujo de trabajo optimizado y minimizando errores a lo largo del ciclo de vida del software.

### DESARROLLO ACTIVIDAD 1

#### 1. Contextualización

##### 1.1 Github.

Es una plataforma de desarrollo colaborativo basada en la nube que permite a los desarrolladores gestionar, almacenar y colaborar en proyectos de software utilizando Git, un sistema de control de versiones. Sus principales funciones son:

<b>Control de versiones</b>	GitHub permite a los desarrolladores crear y gestionar versiones de su código, lo que facilita el seguimiento de cambios, la reversión a versiones anteriores y la colaboración sin conflictos.
<b>Repositorio centralizado</b>	Cada proyecto tiene un "repositorio" en el que se almacena el código. Desde este repositorio, los desarrolladores pueden clonar (copiar) el código a sus propias máquinas, trabajar en él y luego subir los cambios.
<b>Colaboración</b>	GitHub facilita la colaboración mediante herramientas como <i>pull requests</i> , que permiten a los colaboradores sugerir cambios que pueden ser revisados y aceptados por los administradores del proyecto. También cuenta con comentarios y revisiones de código, que fomentan el trabajo en equipo.
<b>Integraciones y automatización</b>	GitHub se integra con otras herramientas, como sistemas de integración continua y plataformas de despliegue, que permiten automatizar pruebas y lanzamientos de software.

##### 1.2 Contenedores.

Los contenedores son una tecnología que permite empaquetar una aplicación junto con todas sus dependencias y configuraciones necesarias en un entorno aislado y portátil. Esto significa que una aplicación en un contenedor puede ejecutarse de manera consistente en diferentes entornos, ya sea en la computadora del desarrollador, en servidores de pruebas o en producción. Sus características son:

<b>Aislamiento</b>	Los contenedores crean un entorno separado para cada aplicación, evitando conflictos entre dependencias o configuraciones de distintas aplicaciones en un mismo sistema
--------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>Portabilidad</b>	Como el contenedor incluye todas las dependencias y configuraciones necesarias, puede ejecutarse en cualquier sistema compatible, lo cual es ideal para mover aplicaciones entre entornos locales, servidores o la nube
<b>Eficiencia</b>	A diferencia de las máquinas virtuales, que emulan sistemas completos con su propio sistema operativo, los contenedores comparten el núcleo del sistema operativo del anfitrión, por lo que consumen menos recursos y son más rápidos de iniciar y detener
<b>Escalabilidad</b>	Es fácil crear múltiples instancias de un contenedor para adaptarse a la demanda. Esto hace que los contenedores sean ideales para aplicaciones que necesitan escalar rápidamente en entornos de producción

## 2. Creación de repositorio en GitHub

### 2.1. Iniciar sesión en GitHub:

Ingresa a [github.com](https://github.com/) y accede a la cuenta. Si no se cuenta con una se puede registrar gratuitamente.

### 2.2 Ir a la página de creación de repositorio:

Hacer clic en el ícono del perfil en la esquina superior derecha y seleccionar “Your repositories” y luego, hacer clic en el botón New o ir directamente a: <https://github.com/new>(<https://github.com/new>)

**2.3 Configurar el repositorio:** se debe diligenciar los siguientes campos:

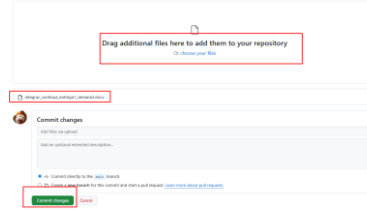
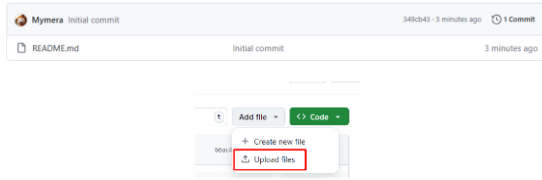
- ✓ **Nombre del repositorio:** Ingresar un nombre único para el repositorio.
- ✓ **Descripción (opcional):** Agregar una breve descripción del proyecto
- ✓ **Privacidad:**
  - **Público:** Cualquiera puede ver el repositorio.
  - **Privado:** Solo el propietario y las personas a las que se invite podrán verlo.
- ✓ **Inicializar con un README (opcional):** Archivo que describe el proyecto
- ✓ **Añadir un archivo .gitignore (opcional):** Se puede seleccionar una plantilla de .gitignore para evitar que ciertos archivos se incluyan en el repositorio.
- ✓ **Elegir una licencia (opcional):** Para definir los términos de uso del código.

### 2.4 Crear el repositorio

Una vez configurados los detalles, hacer clic en el botón “Create repository”.

### 2.5 Agregar archivos y realizar el primer commit:

Si se inicializó el repositorio con un README, ya se tendrá un archivo en el repositorio o si se desea se puede agregar archivos desde la computadora con el comando Git o directamente desde la interfaz de GitHub.



## 2.6 Clonar el repositorio (opcional)

Para trabajar en el repositorio localmente, se puede clonar en la computadora de la siguiente manera:

- Hacer clic en el botón Code en la página del repositorio.
- Copiar la URL del repositorio (HTTPS o SSH).
- Abrir una terminal en la computadora, navegar a la carpeta donde se quiere guardar el repositorio, y

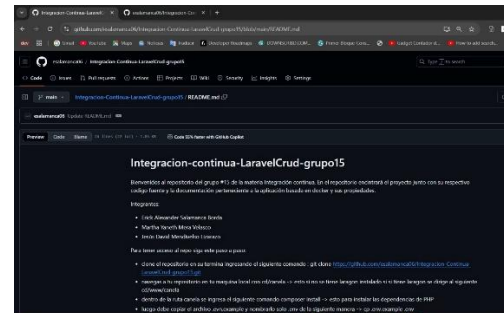
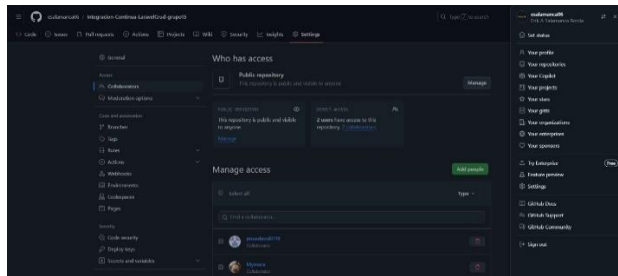
ejecutar: `git clone <URL_DEL_REPOSITORIO>`

De esta manera ya se tiene un repositorio en GitHub donde se puede almacenar y gestionar el código.

## Enlace del repositorio de trabajo:

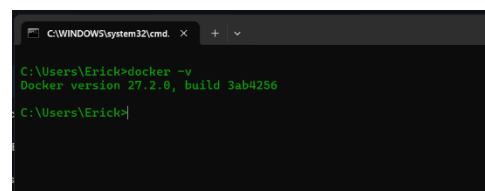
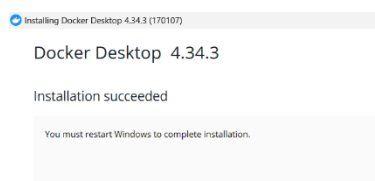
<https://github.com/esalamanca06/Integracion-Continua-LaravelCrud-grupo15.git>

Se asigna permiso a los integrantes del grupo y también se incluyen los datos en el archivo *Readme.md*:

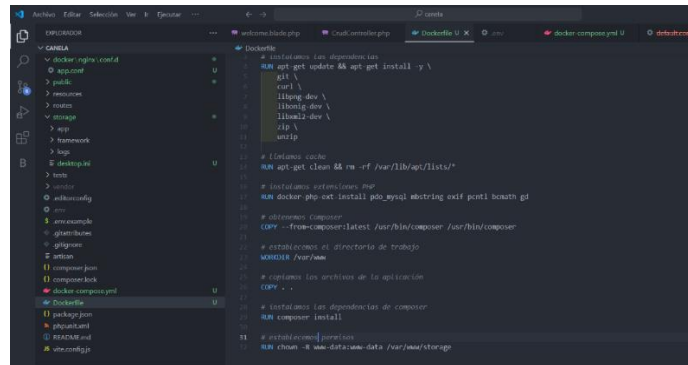


## 3. Creación de contenedores

3.1 Se descarga la aplicación desde la página de <https://www.docker.com/> y se procede a realizar la instalación de Docker validando al final su versión y conexión con Docker



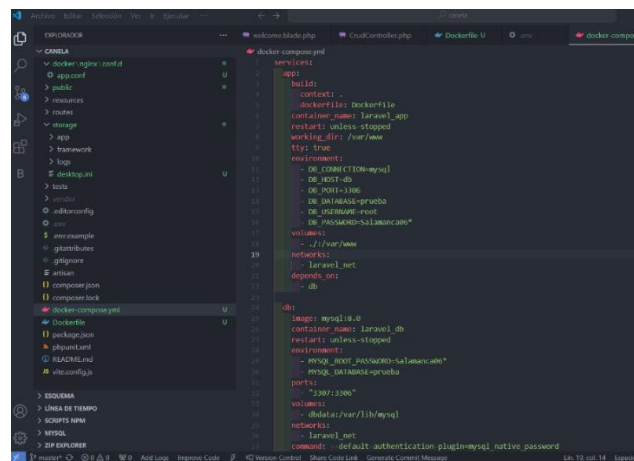
3.2 Posteriormente nos dirigimos a la carpeta raíz del proyecto de Laravel para crear y configurar el Dockerfile sobre el cual vamos a trabajar. Se estructuran las dependencias requeridas junto con el establecimiento de reglas como la limpieza de la cache. Adicionalmente, se instalan las extensiones de PHP junto con la obtención de composer y la estructura del directorio de trabajo. Finalmente generamos una copia de datos e instalación de composer y la estipulación de permisos.



```

FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
    git \
    curl \
    libpq-dev \
    libssl-dev \
    zip \
    unzip
# Instalar composer
RUN apt-get clean && rm -rf /var/lib/apt/lists/*
# Instalar composer
RUN curl -sS https://getcomposer.org/installer | php && mv composer.phar /usr/local/bin/composer
# Establecer el directorio de trabajo
WORKDIR /var/www
# Copiar los archivos de la aplicación
COPY . .
# Instalar las dependencias de composer
RUN composer install
# Crear el directorio de almacenamiento
RUN mkdir -p storage && chmod 777 storage
  
```

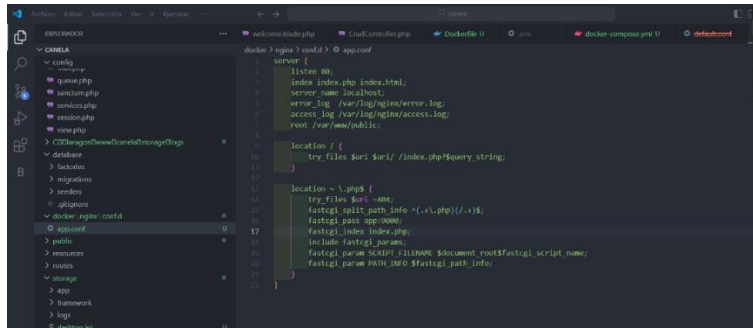
3.4 Se realiza la definición de dos contenedores mediante la creación del archivo docker-compose.yml. En cada uno de los contenedores se definen los nombres, rutas, puertos y dependencias a utilizar para la correcta conexión y junto a este mismo, se le brindan las respectivas reglas y parámetros que se requieren para la conexión a la base de datos con el fin de obtener la comunicación entre contenedores tanto el de aplicación, como el de base de datos.



```

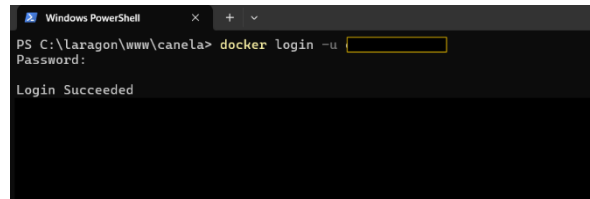
version: '3'
services:
  app:
    build: .
    context: .
    container_name: laravel_app
    restart: unless-stopped
    working_dir: /var/www
    tty: true
    environment:
      - DB_CONNECTION=mysql
      - DB_HOST=db
      - DB_PORT=3306
      - DB_DATABASE=prueba
      - DB_USERNAME=root
      - DB_PASSWORD=Salavarez123
    volumes:
      - ./var/www:/var/www
    networks:
      - laravel_net
    depends_on:
      - db
  db:
    image: mysql:8.0
    container_name: laravel_db
    restart: unless-stopped
    environment:
      - MYSQL_ROOT_PASSWORD=Salavarez123
      - MYSQL_DATABASE=prueba
      - TZ=UTC-5
    volumes:
      - dbdata:/var/lib/mysql
    networks:
      - laravel_net
networks:
  laravel_net:
    driver: bridge
  
```

3.5 Luego procedemos a la creación de carpetas en la raíz del proyecto definidas como **./docker/nginx/conf.d:/etc/nginx/conf.d** y dentro de esta, se procede a la creación de la arquitectura y posterior ruta para la definición de conexión del servidor nginx, el cual hace de intermediario para que sea utilizable la app de Laravel, estableciendo el puerto de escucha del servidor de levantamiento localhost, se estructura el control de errores y directorio raíz de archivos de Laravel y facilitador de manejo de archivos de php.

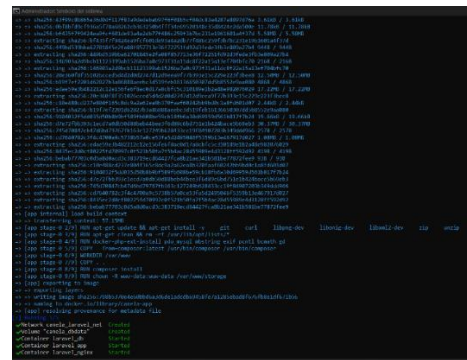
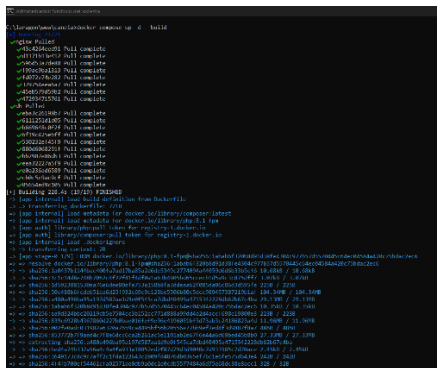


3.6 Seguido a ello se procede a construir y ejecutar los contenedores de la siguiente forma:

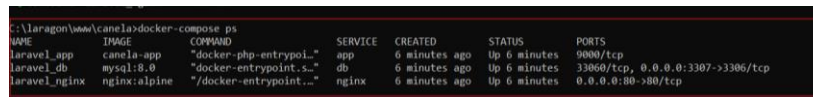
- Mediante powershell se debe navegar hasta el archivo raíz y nos logueamos en docker mediante el comando **login u-** junto con el password que previamente definimos al momento de la instalación de docker en windows.



- Luego de ello se procede a la construcción y levantamiento de los contenedores ejecutando el comando **docker-compose up -d --build**.



3.7 Ejecutado el proceso, procedemos a validar que los contenedores se ejecuten y estén corriendo correctamente mediante el comando **docker ps**.



3.8 Ahora el paso más importante, una vez validado que los contenedores estén corriendo, es ejecutar las migraciones con el fin de crear los 3 contenedores requeridos para este caso como lo son app (Laravel), db (mysql), y nxing (servidor web), donde el **comando docker-compose exec app php artisan**

**migrate** debe garantizar que los contenedores se comuniquen entre si y se pueda establecer los volúmenes persistentes para la base de datos y finalmente debe instalar el resto de dependencias para el correcto funcionamiento de Laravel. No sin antes limpiar la cache y configuraciones del entorno de implementación que puedan afectar el proceso.

```

[INFO] Preparing database...
[creating migration table ..... 73ms DONE]
[INFO] Running migrations...
2014-10-12 00:0000 create users table ..... 60ms DONE
2014-10-12 10:0000 create password_reset_tokens table ..... 21ms DONE
2019-08-19 00:0000 create failed_jobs table ..... 8ms DONE
2019-12-14 00:0000 create personal_access_tokens table ..... 47ms DONE
C:\laragon\www\canela\docker-compose ps
NAME                                COMMAND                                SERVICE    CREATED       STATUS       PORTS

```

Finalmente se puede evidenciar que se crearon correctamente cada uno de los tres contenedores.

## 4. Comunicación entre los contenedores

Para evidenciar la correcta comunicación entre los 3 contenedores debemos tener en cuenta:

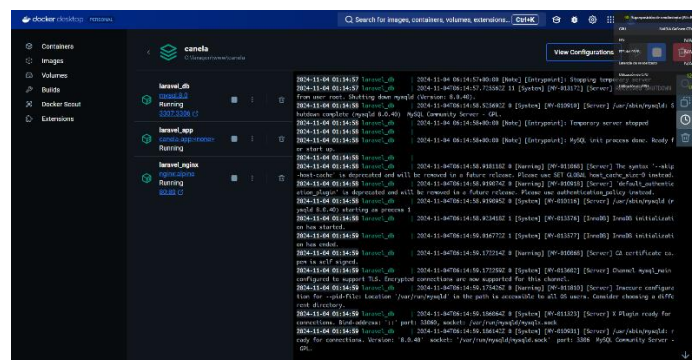
### 4.1 Validar que los contenedores se encuentran corriendo de la forma correcta mediante el comando

**docker ps**

### 4.2 Acceder al contenedor de Laravel mediante el comando **docker exec -it laravel\_db mysql -u root -p**

donde nos solicitará las credenciales respectivas junto con el **password** para la respetiva autenticación.

### 4.3 Una vez validada la contraseña se procede a generar una consulta a la base de datos mediante el comando **SHOW DATABASES;** en cuando lo ejecutamos nos aparecerá la lista de bases de datos junto con la que hemos definido llamada **prueba**, que aparece en el archivo .env del código fuente y con esto garantizamos la correcta comunicación entre contenedores.



```

C:\laragon\www\canela\docker-compose ps
NAME                                COMMAND                                SERVICE    CREATED       STATUS       PORTS
laravel_app                         docker-php-entrypoint                app        2 hours ago   Up 2 hours   0.0.0.0:80->80/tcp
laravel_db                           docker-entrypoint.sh                 db          2 hours ago   Up 2 hours   0.0.0.0:3306->3306/tcp
laravel_regis                        docker-entrypoint.sh                 redis       35 minutes ago   Up 35 minutes   0.0.0.0:6379->6379/tcp

C:\laragon\www\canela\docker exec -it laravel_db mysql -u root -p
Enter password:
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql> show databases;
+-----+
| database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| prueba |
| sys |
+-----+
mysql> exit;
Bye
C:\laragon\www\canela\

```



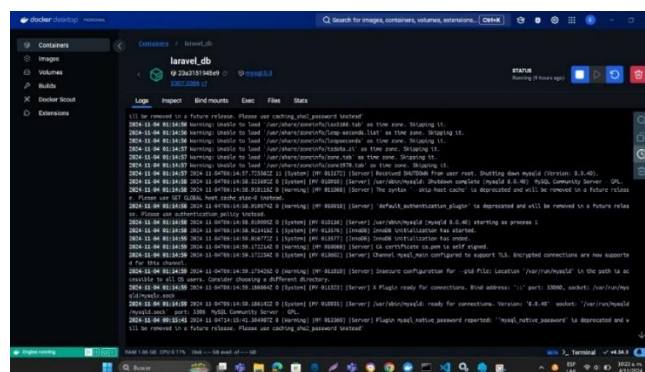
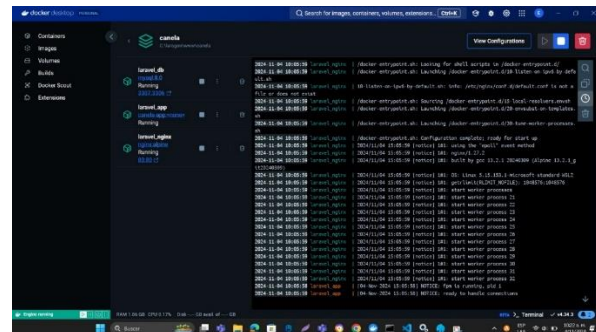
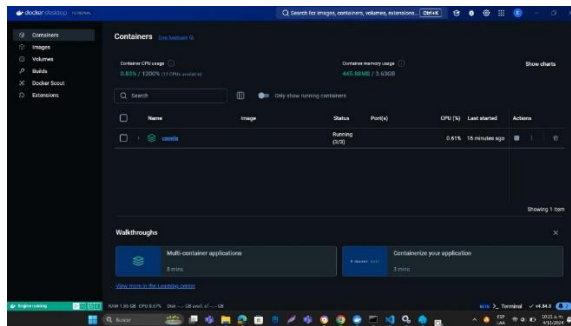
#### 4.4 Finalmente ingresamos al contenedor de la app de Laravel o a la aplicación, mediante el comando ***docker-compose exec app bash***

```
lavalab_jh mysql:8.0 "docker-entrypoint.sh." db 9 hours ago Up 9 hours 1386B/tip, 0.0.0.0:3307->3306/tcp  
lavalab_rginx nginx:alpine "/docker-entrypoint..." rginx 21 seconds ago Up 19 seconds 0.0.0.0:80->80/tcp
```

```
[lavalab@name:/aws/docker-compose exec app bash  
root@f8e6d6538ee:/var/www# php artisan tinker  
my shell v8.12.4 (env v1.0 - cli) by Justin Hileman  
DB_CONNECTION=mysql  
- DB_HOST=connection()-getpdo();  
- DB_PORT=3307;  
Informations: false,  
mysql:host=mysql,  
charset=utf8mb4,  
collation=utf8mb4_unicode_ci,  
encoding=utf8mb4,  
prefixIndex=';',  
prefixIndexType=NATURAL,  
strictMode=false,  
dateFormat='mysql',  
logger=null,  
logLevel=WARN,  
maxConnections=10000,  
maxIdleTime=1800,  
threads=2,  
questions=14,  
slowQueryLog=true,  
statements=184,  
flushTables=3,  
queryCacheSize=100,  
queriesPerSecondAvg=0.001,  
serverStatus='mysql:8.0.130',  
serverVersion='8.0.130',  
statisticsEnabled=[  
    'MODSTATS' => true,  
],  
uniqueConstraints=[],  
connectionStatistics=['db via TCP/IP',  
    'mysql_stats_mysql', BOTH,  
],  
},  
}
```

4.5 Una vez en el contenedor probamos la aplicación de la base de datos mediante el comando ***php artisan tinker*** y procedemos a ejecutar la consulta a la base de datos como si la hicieramos por elocuent de Laravel de la siguiente manera:

```
DB::connection()->getPdo();
```



## CONCLUSIONES

El uso de GitHub y los contenedores ha transformado el proceso de integración continua en el desarrollo de software, permitiendo a los equipos colaborar de manera más efectiva, segura y escalable. GitHub facilita la gestión del código fuente, el control de versiones y la colaboración entre desarrolladores, lo cual reduce los errores y permite un flujo de trabajo más ordenado y transparente. Los contenedores, por su parte, proporcionan un entorno consistente para ejecutar aplicaciones, asegurando que el software funcione de manera idéntica en cualquier entorno, desde desarrollo hasta producción.

## BIBLIOGRAFIA

Apple, L. F. (2020, 18 julio). *Cómo crear una cuenta de GitHub - lafactoriaapple*. La Factoría Apple.

Recuperado el 01 de noviembre de 2024 de

<https://www.lafactoriaapple.com/herramientas/github/crear-cuenta-github.php>

*Acerca de GitHub y Git - documentación de GitHub*. (s. f.). GitHub Docs. Recuperado el 01 de noviembre

de 2024 de <https://docs.github.com/es/get-started/start-your-journey/about-github-and-git>

*¿Qué son los contenedores?* (s. f.). Recuperado el 01 de noviembre de 2024 de

<https://www.netapp.com/>. <https://www.netapp.com/es/devops-solutions/what-are-containers/>