Taller #1 Docker

Leonardo Bustamante

Actividades

i. Introducción

Se debe crear un sistema de software distribuido, a partir de la integración de un conjunto de estrategias de diseño arquitectónico. De esta forma, se diseñará y construirá su primer **componente lógico** y su respectivo **componente** de **base de datos**.

- Microservicio **supermarket_ms**: Python.
- Base de Datos **supermarket_db**: MySQL.

ii. Entrega

- Se debe entregar un archivo pdf, con las evidencias del sistema en ejecución, salidas del comando docker y la aplicación postman.

- Explicar componentes y salidas.

- **Tiempo:** 2 hrs
  Medio: moodle

iii. Requisitos

a. Infraestructura

Un equipo con Docker instalado.

b. Postman

Descargar e instalar Postman según el sistema operativo.

Nota: esta herramienta puede ser reemplazada por cualquier **cliente HTTP** de preferencia.

iv. Componente 1: Base de Datos

1. Crear un directorio llamado **supermarket_db**.

2. Dentro del directorio creado, crear un archivo **Dockerfile**:

```
FROM mysql:5.7
ENV MYSQL_ROOT_PASSWORD=123
ENV MYSQL_DATABASE=supermarket_db
ENV MYSQL_USER=supermarket
ENV MYSQL_PASSWORD=2021
EXPOSE 3306
```

Puerto TCP a usar: **3306**.

3. Crear la imagen Docker, dentro del mismo directorio ejecutar el siguiente comando:

```
docker build -t supermarket_db .
```

4. Desplegar la base de datos, mediante el siguiente comando:

```
docker run -d -t -i -p 3306:3306 --name supermarket_db
supermarket_db
```

5. Desplegar el cliente web de MySQL **PhpMyAdmin**, mediante el siguiente comando:

```
docker run --name phpmyadmin -d --link supermarket_db:db -p
8081:80 phpmyadmin
```

6. Acceder a la base de datos, usando el cliente *PhpMyAdmin*, mediante el navegador web: **http://localhost:8081**.

7. Iniciar sesión usando las credenciales definidas en el *Dockerfile* de la imagen de la base de datos.

v. Componente 2: Microservicio

Puerto TCP a usar: **4000**.

a. Código Fuente

Descargar el código fuente. Este se puede encontrar en la carpeta de **Google Drive** del curso (sección **Archivos**).

b. Arquitectura del Microservicio

La arquitectura interna del microservicio está compuesta por las siguientes tres capas:

- **Models**: abstracción del modelo de datos.
- **Serializers**: conversión de tipos de datos.
- **APIViews**: gestión de peticiones HTTP basada en REST.

c. Dockerización

En la raíz del proyecto crear un archivo llamado **Dockerfile**:

```
FROM python:3
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
ARG URL=0.0.0.0:4000
CMD ["sh", "-c", "python manage.py makemigrations supermarket_ms && python
manage.py migrate && python manage.py runserver $URL"]
```

d. Creación de Imagen Docker

Ejecutar el siguiente comando:

```
docker build -t supermarket_ms .
```

e. Creación y Despliegue del Contenedor

```
docker run -p 4000:4000 -e DB_HOST=X -e DB_PORT=3306 -e DB_USER=Y
-e DB_PASSWORD=Z -e DB_NAME=supermarket_db -e URL=0.0.0.0:4000
supermarket_ms
```

X = **host.docker.internal** (alternativa #1).

X = **IP del contenedor (supermarket_db)** (alternativa #2). Para obtener la IP del contenedor, ejecutar el siguiente comando:

```
docker inspect CONTAINER_ID
```

La IP corresponde al atributo *Networks* > *bridge* > *IPAddress*.

Y = usuario de la base de datos.
Z = contraseña de la base de datos.

f. Verificar Contenedor

Ejecutar el siguiente comando en una pestaña de la terminal, diferente a la que se usó para el despliegue (paso inmediatamente anterior).

```
docker ps
```

vi. Conector 1: 'Django Client for MySQL'

Corresponde al cliente MySQL usado por el framework **Django** para conectarse a la base de datos.

vii. Conector 2: REST

a. Exposición de la API-REST

Analizar el código fuente del componente **Microservicio** e identificar la totalidad de operaciones expuestas mediante la API: operaciones CRUD para **Categories** y **Products** utilizando las operaciones HTTP: **GET**, **POST**, **PUT** y **DELETE**.

b. Consumo de la API-REST

Realizar peticiones a través de *Postman* para verificar la funcionalidad del microservicio.

1. Ejemplo: crear una categoría.

Method: **POST**
URL: localhost:4000/categories/
Body:

```
{

  "name": "Cat. Jeisson",
  "description": "Categoría creada para el curso de Desarrollo de
Aplicaciones Web 2"
}
```

Response: **201 Created**

2. Ejemplo: obtener una categoría por ID.

Method: **GET**

URL: localhost:4000/categories/1

Response: **200 Ok**

3. Ejemplo: obtener la lista de todas las categorías.

Method: **GET**
URL: localhost:4000/categories/
Response: **200 Ok**

4. Ejemplo: actualizar una categoría.

Method: **PUT**
URL: localhost:4000/categories/1
Body:

```
{
  "name": "Cat. AWRI",
  "description": "Categoría creada para el curso de Desarrollo de Aplicaciones Web"
}
```

Response: **200 Ok**

5. Ejemplo: eliminar una categoría.

Method: **DELETE**
URL: localhost:4000/categories/1
Response: **204 No Content**

c. Verificación de la Base de Datos

1. Conectarse a la base de datos desplegada, mediante el cliente PhpMyAdmin.

2. Verificar el estado de las tablas de la base de datos, tras la ejecución de cada una de las peticiones HTTP realizadas sobre la API-REST del microservicio.