



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**título del TFG
Documentación Técnica**



Presentado por nombre alumno
en Universidad de Burgos — 12 de junio
de 2023

Tutor: nombre tutor

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	1
Apéndice B Especificación de Requisitos	3
B.1. Introducción	3
B.2. Objetivos generales	3
B.3. Catalogo de requisitos	3
B.4. Especificación de requisitos	3
Apéndice C Especificación de diseño	5
C.1. Introducción	5
C.2. Diseño de datos	5
C.3. Diseño procedimental	5
C.4. Diseño arquitectónico	5
Apéndice D Documentación técnica de programación	7
D.1. Introducción	7
D.2. Estructura de directorios	7
D.3. Manual del programador	7

D.4. Compilación, despliegue y ejecución del proyecto	7
D.5. Pruebas del sistema	13
Apéndice E Documentación de usuario	15
E.1. Introducción	15
E.2. Requisitos de usuarios	15
E.3. Instalación	15
E.4. Manual del usuario	15
Bibliografía	17

Índice de figuras

D.1.	Página de descarga de Docker Desktop	8
D.2.	Instalación de Docker Desktop en Mac	8
D.3.	Ventana principal de Docker Desktop	9
D.4.	Creación de las imágenes de los contenedores Docker	11
D.5.	Despliegue de los contenedores Docker	12
D.6.	Ejecución multi-contenedor de la aplicación	12
D.7.	Logs de la ejecución de un contenedor en Docker Desktop	13

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

A.2. Planificación temporal

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

Una muestra de cómo podría ser una tabla de casos de uso:

B.2. Objetivos generales

B.3. Catalogo de requisitos

B.4. Especificación de requisitos

CU-1	Ejemplo de caso de uso
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-xx, RF-xx
Descripción	La descripción del CU
Precondición	Precondiciones (podría haber más de una)
Acciones	<ul style="list-style-type: none"> 1. Pasos del CU 2. Pasos del CU (añadir tantos como sean necesarios)
Postcondición	Postcondiciones (podría haber más de una)
Excepciones	Excepciones
Importancia	Alta o Media o Baja...

Tabla B.1: CU-1 Nombre del caso de uso.

Apéndice C

Especificación de diseño

- C.1. Introducción
- C.2. Diseño de datos
- C.3. Diseño procedimental
- C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

D.4. Compilación, despliegue y ejecución del proyecto

Para realizar la compilación, despliegue y ejecución del proyecto es imprescindible disponer de **Docker CLI** y **Docker Compose** instalados en el equipo. La mejor forma de instalarlos es mediante **Docker Desktop**, ya que se encarga de instalar en el equipo el *daemon* de Docker, Docker CLI, Docker Compose, y demás dependencias y herramientas que no vamos a utilizar para este proyecto, pero que nos pueden ayudar en el futuro. Docker Desktop es compatible con Windows, macOS y Linux, y soporta tanto arquitecturas x86 como ARM64 (Apple Silicon).

Instalación de Docker Desktop

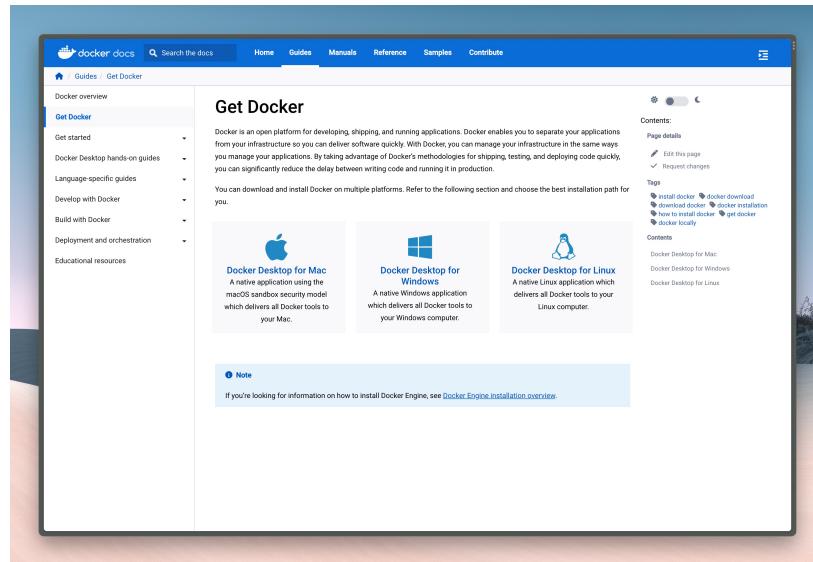


Figura D.1: Página de descarga de Docker Desktop

Para realizar la instalación de Docker Desktop tan sólo debemos dirigirnos a <https://docs.docker.com/get-docker/>, seleccionar la plataforma deseada, y descargar el instalador.

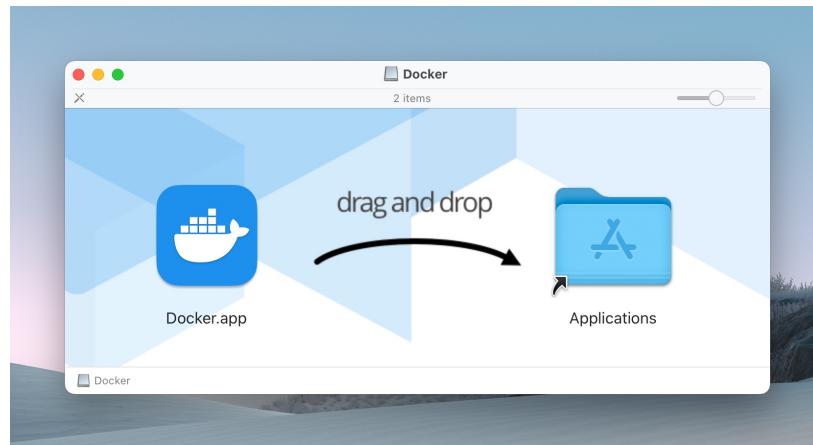


Figura D.2: Instalación de Docker Desktop en Mac

Una vez ejecutemos el instalador (en caso de macOS simplemente se debe arrastrar la aplicación a la carpeta de Aplicaciones), y hayamos seguido todos los pasos hasta finalizar la instalación, nos encontraremos con el panel principal de Docker Desktop.

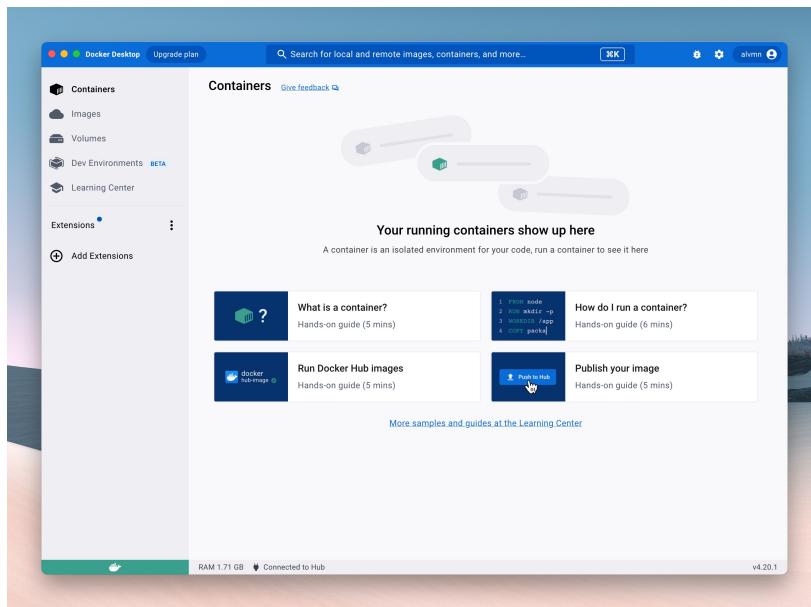


Figura D.3: Ventana principal de Docker Desktop

En esta ventana podemos ver todos los contenedores que están ejecutándose actualmente en el sistema, así como las imágenes descargadas y los volúmenes creados.

Preparación del entorno de desarrollo

Para hacer más sencillo el cambio de valores en distintos parámetros de la aplicación, como el mapeado de puertos o la gestión de secretos, y facilitar así el despliegue de nuevos entornos o la escalabilidad de los servicios, estos valores se han externalizado en un sólo archivo `.env`, que va a ser un archivo que simplemente va a contener las variables junto a sus valores.

Al ser un fichero que contiene secretos, siguiendo lo que dictan las buenas prácticas no se sincronizará con el repositorio, ya que si se tratase de un entorno de producción o el código se hiciera público, cualquier usuario podría obtener acceso a los datos de los usuarios de la aplicación.

Por lo tanto, antes de desplegar los contenedores, para que la aplicación funcione correctamente el archivo se ha de crear dentro del directorio `/docker-compose`, y se deben de definir las siguientes variables dentro de él:

```
# Variables del backend de la parte de los clientes
```

```

CLIENT_WEB_DOCKER_PORT=8081
CLIENT_WEB_LOCAL_PORT=8081
CLIENT_RELOAD_DOCKER_PORT=35730
CLIENT_RELOAD_LOCAL_PORT=35730
CLIENT_DEBUG_DOCKER_PORT=5006
CLIENT_DEBUG_LOCAL_PORT=5006

# Variables del backend de la parte de gestión
MANAGE_WEB_DOCKER_PORT=8080
MANAGE_WEB_LOCAL_PORT=8080
MANAGE_RELOAD_DOCKER_PORT=35729
MANAGE_RELOAD_LOCAL_PORT=35729
MANAGE_DEBUG_DOCKER_PORT=5005
MANAGE_DEBUG_LOCAL_PORT=5005

# Variables de la base de datos
DATABASE_HOST=mysql ldb
MYSQL_DATABASE=nutri_db
MYSQL_USER=<MySQL user> # Introduce el nombre del usuario de MySQL
MYSQL_PASSWORD=<MySQL password> # Introduce la pass escogida para
el usuario creado en el paso anterior
MYSQL_ROOT_PASSWORD=<MySQL root password> # Introduce la pass
escogida para el usuario root de MySQL
DB_DOCKER_PORT=3306
DB_LOCAL_PORT=3306

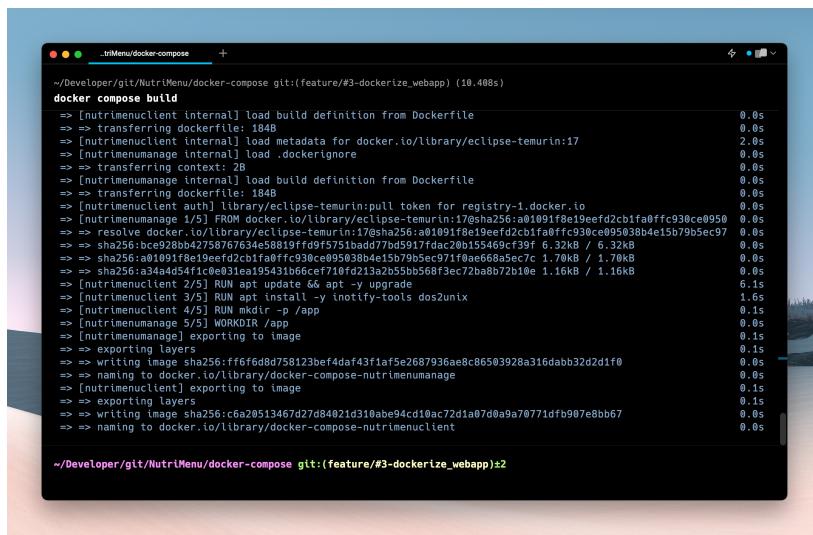
```

Los valores pueden ser modificados si es necesario, pero se recomienda usar los valores dados (indicando los valores escogidos en `MYSQL_USER`, `MYSQL_PASSWORD` y `MYSQL_ROOT_PASSWORD`). Con estos valores lo que estamos indicando es:

- Los puertos en los que se van a ejecutar cada una de las aplicaciones, que en este caso van a ser el **8080** para la aplicación de gestión y el **8081** para la aplicación de los clientes.
- Los puertos en los que va a estar escuchando el plugin *LiveReload* de Spring, que es el encargado de recargar el servidor en caliente cada vez que se produzcan cambios en el código, y no tener así que recompilar todo el código cada vez que cambiemos algo. En este caso los puertos van a ser el **35729** para la aplicación de gestión y el **35730** para la aplicación de los clientes.

- Los puertos que se van a usar para permitir conectar un debugger remoto a la aplicación, y poder así hacer debug desde el IDE que usemos para el desarrollo. Los puertos serán el **5005** para la aplicación de gestión y el **5006** para la aplicación de los clientes.
- Todas las variables correspondientes a la base de datos, como el **nombre de la instancia**, **nombre de la base de datos**, **credenciales de los usuarios de MySQL**, y el puerto escogido para la base de datos, que en este caso va a ser el usado por defecto, **3306**.

Despliegue de la aplicación



```
~/Developer/git/NutriMenu/docker-compose git:(feature/#3-dockerize_webapp) (10.408s)
docker compose build
>> [nutrimenclient internal] load build definition from Dockerfile
>> [nutrinenclient internal] load metadata for docker.io/library/eclipse-temurin:17
>> [nutrinenmanage internal] load .dockernignore
>> [nutrinenclient internal] load context: 28
>> [nutrinenmanage internal] load build definition from Dockerfile
>> [nutrinenclient internal] load Dockerfile: 184B
>> [nutrinenclient auth] library/eclipse-temurin:pull token for registry-1.docker.io
>> [nutrinenmanage 1/5] FROM docker.io/library/eclipse-temurin:17@sha256:a01091f8e19eef2cb1fa0ffc930ce0950
>> [nutrinenclient 1/5] RUN apt update && apt -y upgrade
>> [nutrinenclient 1/5] RUN apt install -y inotify-tools dos2unix
>> [nutrinenclient 1/5] RUN mkdir -p /app
>> [nutrinenmanage 5/5] WORKDIR /app
>> [nutrinenclient 5/5] exporting to image
>> exporting layers
>> writing image sha256:ffff6fdbd758123bef4dadf43f1af5e2687936ae8c86503928a316dabb32d2d1f0
>> naming to docker.io/library/docker-compose-nutrimenmanage
>> [nutrinenclient] exporting to image
>> exporting layers
>> writing image sha256:c6a20513467d27d84021d310abe94cd10ac72d1a07d0a9a70771dfb907e8bb67
>> naming to docker.io/library/docker-compose-nutrimenclient

~/Developer/git/NutriMenu/docker-compose git:(feature/#3-dockerize_webapp)±2
```

Figura D.4: Creación de las imágenes de los contenedores Docker

Una vez definidas las variables de entorno, podemos probar que todo funciona correctamente construyendo las imágenes de los contenedores de los servicios. Para ello, abriremos una consola y accederemos al directorio `/docker-compose`, donde ejecutaremos el siguiente comando:

```
docker-compose build
```

```

~/Developer/git/NutriMenu/docker-compose git:(feature/#3-dockerize_webapp) (10.408s)
docker compose build

~/Developer/git/NutriMenu/docker-compose git:(feature/#3-dockerize_webapp) (20.296s)
docker-compose up -d

[+] Running 12/12
  ✓ mysql 11 layers [██████████]    0B/0B    Pulled          19.2s
  ✓ 2ae143351a4f Pull complete      4.8s
  ✓ 093b9ff06d48 Pull complete      4.8s
  ✓ 1cf4f76b0363a Pull complete      4.9s
  ✓ a8e5c746438f Pull complete      5.0s
  ✓ 3b8559256f40 Pull complete      5.1s
  ✓ 5b3ed9226ef40 Pull complete      5.1s
  ✓ d1388739c015 Pull complete      5.1s
  ✓ 94a7da2e25e1 Pull complete      9.6s
  ✓ 17b0d06008a32 Pull complete     9.6s
  ✓ b4191a53e3a86 Pull complete     16.8s
  ✓ 0d3688abe7eb Pull complete     16.9s
  ✓ 0d3688abe7eb Pull complete     16.9s

[+] Building 0.0s (0/0)
[+] Running 5/5
  ✓ Network docker-compose_nutrimenu-net      Created          0.0s
  ✓ Volume "docker-compose_db_data"             Created          0.0s
  ✓ Container docker-compose-mysqldb-1        Started         0.7s
  ✓ Container docker-compose-nutrimenumanage-1 Started         0.5s
  ✓ Container docker-compose-nutrimenuclient-1 Started         0.5s

~/Developer/git/NutriMenu/docker-compose git:(feature/#3-dockerize_webapp)±3

```

Figura D.5: Despliegue de los contenedores Docker

Para desplegar los contenedores y levantar el servicio, ejecutaremos el siguiente comando:

```
docker-compose up -d
```

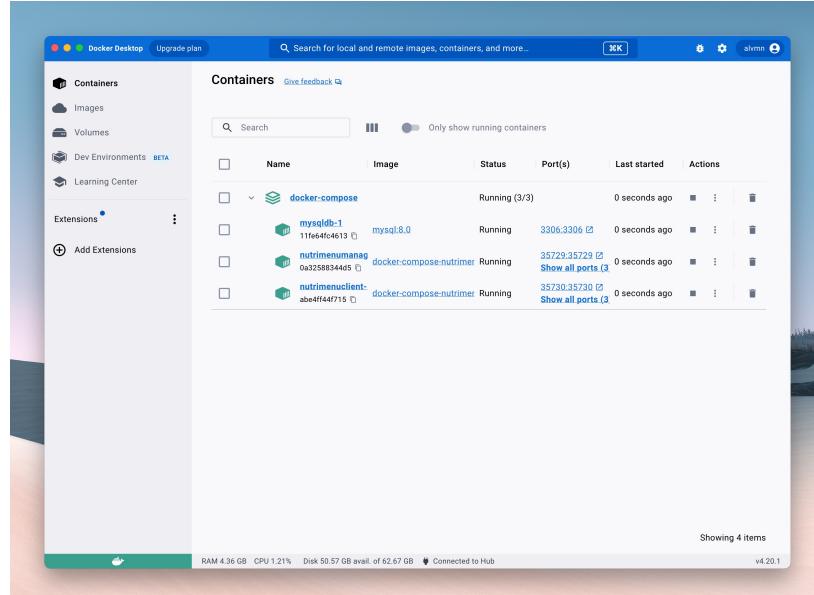


Figura D.6: Ejecución multi-contenedor de la aplicación

Si todo ha ido bien, podremos ver en Docker Desktop los contenedores corriendo con estado *Running*, y ya podremos acceder a la aplicación como haríamos con cualquier tipo de despliegue.

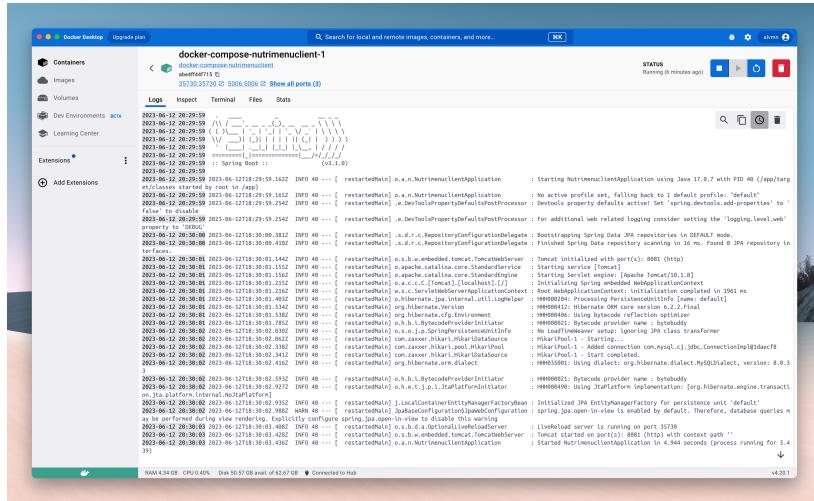


Figura D.7: Logs de la ejecución de un contenedor en Docker Desktop

Para asegurarnos de que no hay ningún tipo de problema, podemos seleccionar los 3 puntos verticales situados a la derecha de cada contenedor, y hacer click en *View details*, ya que esto nos permite poder ver el log de cada servicio en tiempo real.

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

- E.1. Introducción**
- E.2. Requisitos de usuarios**
- E.3. Instalación**
- E.4. Manual del usuario**

Bibliografía
