Universidad San Carlos de Guatemala

Centro Universitario de Occidente

Manejo e Implementación de Archivos

Ing. Christian López

GRAFILES

(Manual Técnico)

Javier Isaías Coyoy Marín

202030556

Quetzaltenango 31 de octubre de 2024

INDICE

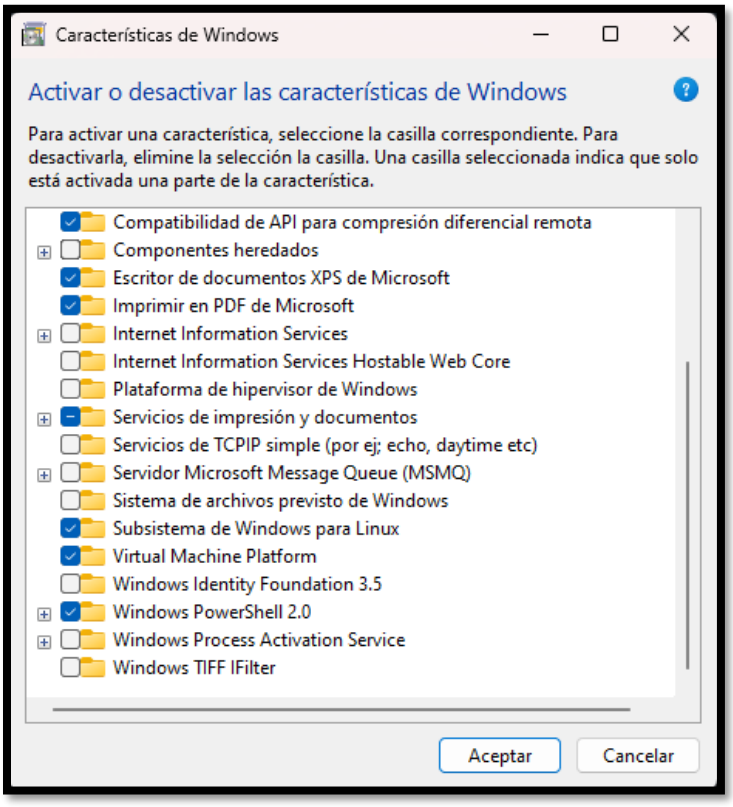
Tecnologías Utilizadas

Para la realización de este proyecto utilizamos las siguientes tecnologías:

Docker

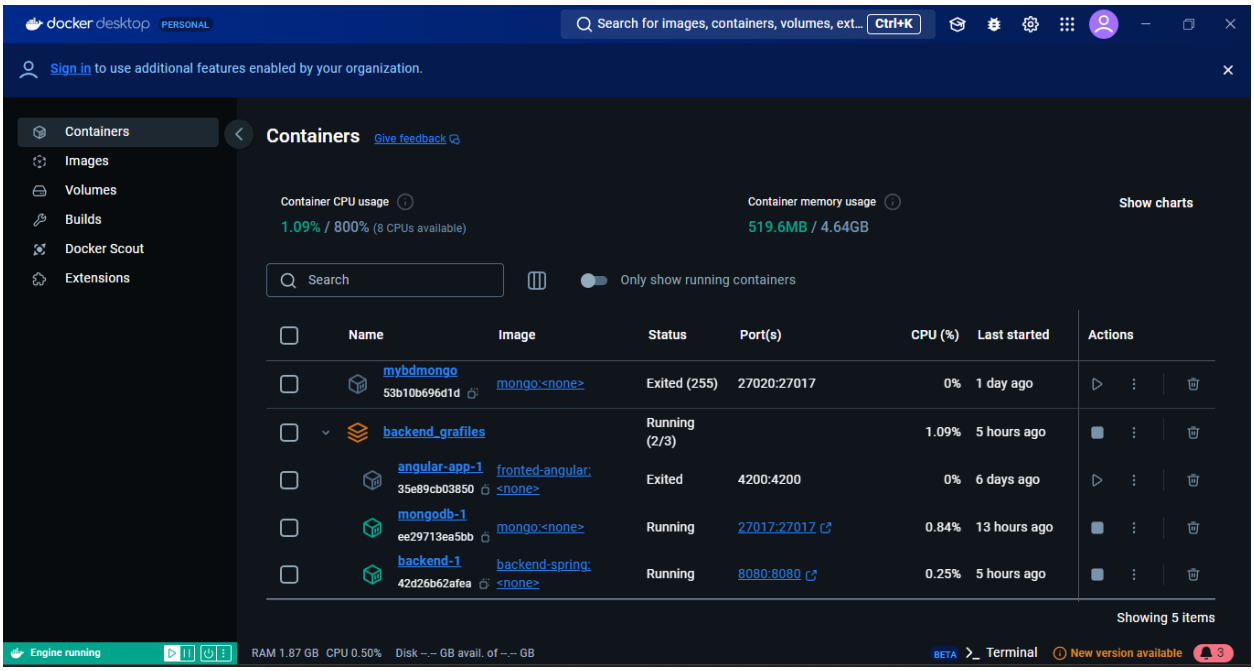
Es una plataforma de software que permite a los desarrolladores construir, empaquetar y ejecutar aplicaciones en contenedores. Los contenedores son entornos ligeros y aislados que incluyen todo lo necesario para que una aplicación funcione, como el código, las bibliotecas y las dependencias, sin necesidad de que estén instalados en el sistema operativo del host.

Para instalar Docker en Windows, primero debemos verificar que versión de Windows. Docker Desktop requiere el soporte de Hyper-V y el Subsistema de Windows para Linux (WSL 2). Para habilitar Hyper-V, dirígete al Panel de control, selecciona Programas y luego Activar o desactivar características de Windows. Busca y marca la casilla de Hyper-V y haz clic en Aceptar; es posible que debas reiniciar tu computadora.

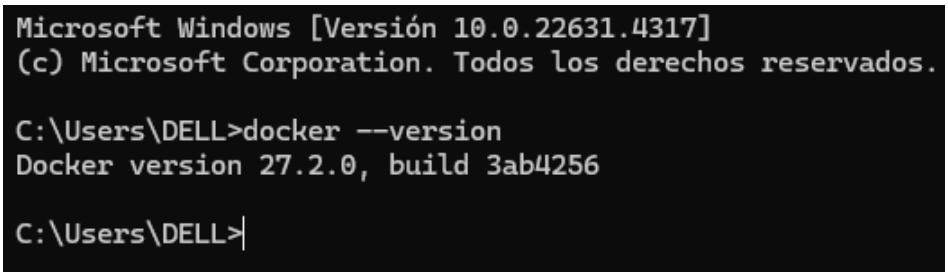


A continuación, habilita WSL 2. Abre PowerShell como administrador y ejecuta el comando wsl --install. Esto instalará el Subsistema de Windows para Linux, y nuevamente, es probable que necesites reiniciar tu máquina. Con estos requisitos previos cumplidos, puedes proceder a descargar Docker Desktop. Visita el sitio web de Docker y descarga el instalador correspondiente para Windows.

Una vez que hayas descargado el instalador (Docker Desktop Installer.exe), ejecútalo y sigue las instrucciones en pantalla. Durante la instalación, asegúrate de habilitar la opción para usar WSL 2. Al finalizar, busca Docker Desktop en el menú de inicio y ábrelo, termina la configuración y al finalizar obtendrás una vista como esta.



Finalmente, para verificar que Docker se ha instalado correctamente, abre una terminal como PowerShell y ejecuta el comando docker --version. Esto te mostrará la versión de Docker instalada, confirmando que todo ha ido bien. Ahora ya puedes empezar a crear y gestionar contenedores en tu entorno de desarrollo.



Angular

(Versión Node.js = v18.19.0)

(Versión npm = 10.2.3)

(Versión ng = 18.2.8)

Angular es un framework de desarrollo de aplicaciones web, que utiliza TypeScript como su principal lenguaje de programación. Para comenzar, es necesario asegurarse de tener instaladas algunas herramientas previas.

Primero, se debe instalar Node.js, que es un entorno de ejecución de JavaScript necesario para ejecutar Angular y su gestor de paquetes, npm (Node Package Manager). Se puede descargar Node.js desde su sitio oficial, donde se encuentra disponible tanto la versión LTS como la versión actual. Una vez completada la instalación, se puede verificar que Node.js y npm están correctamente instalados abriendo el símbolo del sistema (cmd) y ejecutando los comandos node -v y npm -v, que mostrarán las versiones instaladas.

Una vez que Node.js y npm están listos, se procede a instalar Angular CLI, que es la herramienta de línea de comandos para crear y gestionar aplicaciones Angular. Para ello, se abre el símbolo del sistema y se ejecuta el siguiente comando:

- **npm install -g @angular/cli**

Con Angular CLI instalado, se puede crear un nuevo proyecto Angular. Para ello, se ejecuta el comando:

- **ng new nombre-del-proyecto**

Este comando creará una nueva carpeta con el nombre especificado y generará la estructura básica de archivos necesarios para un proyecto Angular. Durante este proceso, se pueden elegir configuraciones adicionales, como el uso de CSS o SCSS, según las preferencias del desarrollador.

Una vez creado el proyecto, se debe navegar a la carpeta del proyecto con el comando:

- **cd nombre-del-proyecto**

Dado que vamos a trabajar con docker debemos definir un archivo docker file para que podamos crear nuestra imagen en docker para este caso nuestro archivo nos queda de la siguiente manera:

- El FROM nos define la imagen base que vamos a tomar como referencia para crear nuestra imagen.
- El WORKDIR establece la carpeta del proyecto dentro de docker.
- Los apartados de Copy, copian toda la información que este dentro de los archivos establecidos.
- Los RUN nos sirven para instalar las dependencias necesarias para nuestro proyecto.

Toma en cuenta que el archivo debe crearse en la carpeta src.

```
FROM node:lts

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install -g @angular/cli

RUN npm install

COPY . .

CMD ["ng", "serve", "--host", "0.0.0.0", "--disable-host-check"]
```

Spring Boot

(Versión Java = 17jdk)

(Versión IntelliJ IDEA = 21.0.3)

Spring Boot es un framework que simplifica el proceso de creación y configuración de aplicaciones basadas en Spring, permitiendo a los desarrolladores centrarse en la lógica del negocio sin preocuparse demasiado por la configuración del entorno. Para comenzar, es esencial contar con algunas herramientas previas.

En primer lugar, es necesario tener instalado Java Development Kit (JDK), preferiblemente la versión más reciente de Java 8 o superior. Se puede descargar el JDK desde el sitio oficial de Oracle o desde OpenJDK. Durante la instalación, se recomienda seguir las instrucciones del asistente y, al finalizar, agregar la variable de entorno JAVA_HOME al sistema, apuntando a la carpeta de instalación del JDK. Para verificar que Java está correctamente instalado, se puede abrir el símbolo del sistema (cmd) y ejecutar el comando java -version, que mostrará la versión instalada.

Una vez que el JDK está configurado, hay varias maneras de crear un proyecto Spring Boot, siendo una de las más populares el uso de Spring Initializar, un generador de proyectos web. Se puede acceder a Spring Initializr a través de su sitio web (https://start.spring.io/). En esta plataforma, se pueden seleccionar las configuraciones del proyecto, como el tipo de proyecto (Maven o Gradle), el lenguaje (Java), la versión de Spring Boot y las dependencias necesarias (como Spring Web, Spring Data JPA, etc.). Una vez configurado, se hace clic en "Generate" para descargar un archivo ZIP que contiene la estructura básica del proyecto.

Después de descargar el archivo, se debe descomprimir en una ubicación deseada. Para trabajar con el proyecto, es recomendable utilizar un IDE (Integrated Development Environment) como IntelliJ IDEA o Eclipse, que son populares entre los desarrolladores de Java. Estos IDEs permiten importar el proyecto descomprimido y ofrecen herramientas para facilitar el desarrollo.

Una vez que el proyecto está abierto en el IDE, se pueden ejecutar las clases principales de Spring Boot. Generalmente, la clase principal se encuentra en el paquete raíz y tiene la anotación @SpringBootApplication. Para ejecutar la aplicación, se puede utilizar la opción de ejecución del IDE o ejecutar el comando Maven o Gradle desde la terminal dentro del proyecto:

Nuevamente como Necesitamos trabajar con docker debemos definir un docker file dentro de nuestro proyecto para así poder cargar la imagen a docker y nuestro

- El FROM nos define la imagen base que vamos a tomar como referencia para crear nuestra imagen.
- El WORKDIR establece el director de trabajo dentro del contenedor.
- COPY establece el archivo .jar para copiar el proyecto.
- ENTRYPOINT Establece el comando que se ejecutará cuando se inicie el contenedor.

```
FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/Backend_grafiles-0.0.1-SNAPSHOT.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]
```

MongoDB

Debemos descargar la imagen de mongo utilizando el comando:

```
Docker pull mongo
```

Esto nos mandara la imagen oficial de MongoDB desde Docker Hub y esto descagará la ultima versión de de MongoDB que este disponible.

Docker Compose

Es una herramienta que permite definir y ejecutar aplicaciones multi-contenedor en Docker. Con Compose, puedes utilizar un archivo YML para configurar todos los servicios que necesita tu aplicación, lo que simplifica el proceso de gestión de contenedores.

Usando un solo archivo docker-compose.yml, puedes definir todos los servicios, redes y volúmenes que componen tu aplicación. Esto facilita la lectura y el mantenimiento de la configuración. Compose gestiona automáticamente la creación de redes y volúmenes para los contenedores, lo que permite una comunicación fluida entre ellos y la persistencia de datos.

Para nuestro caso utilizamos configuramos nuestro docker compose para que almacenara los contenedores de mongo, spring boot y angular y no queda de la siguiente forma:

```
version: '3.8'

services:
  backend:
    image: backend-spring
    build:
      context: ./Backend_grafiles
    ports:
      - "8080:8080"
    volumes:
      - C:/Users/DELL/OneDrive/Documentos/CUNOC/Sexto_Semestre_2024/Manejo de
Archivos (Laboratorio)/proyecto_grafiles/imagenes_backend:/app/images
      - backend_data:/app/data

  mongodb:
    image: mongo
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db

  fronted:
    image: fronted-angular
    build:
      context: ./Fronted_grafiles
    ports:
      - "4200:4200"

volumes:
  mongodb_data:
  backend_data:
```

Definimos nuestros servicios de la siguiente forma:

Primero definimos el nombre del contenedor, luego el nombre que queremos que tenga nuestra imagen, después definimos el contexto en con el cual se va a construir todo en este caso sería la dirección de nuestro proyecto en nuestra máquina definimos los puertos con los cuales se van a conectar tanto dentro de docker como en nuestra máquina local por ultimo definimos los volúmenes si necesitamos que los datos sean persistentes en el docker.

Base de Datos

Colección de los trabajadores y administradores:

```
db.trabajadores.insertMany([
  {
    _id: "JaviCM",
    nombres: "Javier Isaías",
    apellidos: "Coyoy Marín",
    correo: "javic4m@gmail.com",
    puesto: 1,
    clave: ""
  },
  {
    _id: "marfer",
    nombres: "María Fernanda",
    apellidos: "García Martínez",
    correo: "m_garcia@gmail.com",
    puesto: 2,
    clave: "12345"
  },
  {
    _id: "carlo22",
    nombres: "Carlos Alberto",
    apellidos: "López Ramírez",
    correo: "c_lopez@gmail.com",
    puesto: 2,
    clave: "12345"
  }
]);
```

Colección de ficheros del Trabajador:

```
db.ficheros.insertMany([
  {
    _id: 1,
    usuario: "JaviCM",
    padre: "_",
    nombre: "principal",
    activo: true,
    fecha: "28-10-2024 19:43:25"
  },
  {
    _id: 2,
    usuario: "marfer",
    padre: "_",
    nombre: "principal",
    activo: true,
    fecha: "29-10-2024 05:11:59"
  },
  {
    _id: 3,
    usuario: "carlo22",
    padre: "_",
    nombre: "principal",
    activo: true,
    fecha: "29-10-2024 05:12:44"
  }
]);
```

Colección de Archivos del Trabajador:

```
  db.archivos.insertMany([
    {
      "_id": 1,
      "usuario": "JaviCM",
      "padre": "principal",
      "nombre": "Horario",
      "contenido": "Horario de Clases:\n\nManejo e Implementación de
Archivos:             D…",
      "extension": "texto",
      "activo": true,
      "copia": false,
      "fecha": "31-10-2024 06:19:59"
    },
```

```
  {
    "_id": 2,
    "usuario": "JaviCM",
    "padre": "principal",
    "nombre": "Pensum",
    "contenido": "",
    "extension": "img_1",
    "activo": true,
    "copia": true,
    "fecha": "31-10-2024 07:35:00"
  },
  {
    "_id": 3,
    "usuario": "JaviCM",
    "padre": "principal",
    "nombre": "Pensum(COPIA)",
    "contenido": "",
    "extension": "img_1",
    "activo": false,
    "copia": false,
    "fecha": "31-10-2024 14:48:35"
  }
]);
```

Colección de Archivos Compartidos entre Trabajadores:

```
db.compartidos.insertOne({
  _id: 1,
  emisor: "JaviCM",
  receptor: "marfer",
  nombre: "Horario",
  contenido: "Horario de Clases:\n\nManejo e Implementación de Archivos: D…",
  extension: "texto",
  fecha: "31-10-2024 06:56:07",
  _class: "com.example.backend_grafiles.model.Compartido"
});
```