

Centro Universitario de Ciencias Exactas e Ingenierías

CUCEI



Computación Tolerante a fallas

Ejemplo utilizando Docker

Michel Emanuel López Franco

Gutiérrez Galán Ruben Alejandro

Código: 214798315

Contenido

¿Qué es Docker?:	3
Ventajas y desventajas de usar Docker:.....	3
Programa desarrollado:.....	4
Archivo main.py:	4
Archivo requirements.txt:	5
Archivo Dockerfile:	5
Comandos de Docker:	6
Actualizar main:.....	7
Sin uso de Docker comandos:	8
Conclusión:	9
Bibliografía:	9

¿Qué es Docker?:

Docker es una plataforma de código abierto diseñada para ayudar en la creación, implementación y administración de aplicaciones en contenedores. Los contenedores son entornos ligeros y autónomos que encapsulan una aplicación y todas sus dependencias, lo que facilita la ejecución de aplicaciones de manera consistente en diferentes entornos, como máquinas locales, servidores en la nube o clústeres de contenedores.

Algunas características clave de Docker incluyen:

- **Contenedores:** Docker utiliza tecnologías de contenedorización, como Docker Engine, para crear y ejecutar contenedores. Estos contenedores son unidades aisladas que contienen aplicaciones y sus componentes, como bibliotecas y configuraciones, lo que garantiza que las aplicaciones se ejecuten de manera coherente en cualquier entorno que admita Docker.
- **Portabilidad:** Los contenedores Docker son portátiles y se pueden ejecutar en diferentes sistemas operativos (Linux, Windows, macOS) y plataformas en la nube (Amazon Web Services, Microsoft Azure, Google Cloud Platform, etc.).
- **Eficiencia:** Los contenedores son más livianos que las máquinas virtuales, lo que significa que se pueden iniciar y detener rápidamente, lo que ahorra recursos y acelera la implementación de aplicaciones.
- **Gestión de recursos:** Docker proporciona herramientas para administrar y orquestar contenedores en clústeres, lo que facilita la escalabilidad y la gestión de aplicaciones distribuidas.
- **Control de versiones:** Docker permite la creación de imágenes de contenedor, que son instantáneas de aplicaciones y su entorno. Estas imágenes se pueden versionar y compartir a través de Docker Hub u otros registros de contenedores.
- **Automatización:** Docker se integra bien con herramientas de automatización y orquestación, como Docker Compose, Kubernetes y otras, lo que facilita la administración de aplicaciones en entornos complejos.

Ventajas y desventajas de usar Docker:

Ventajas:

- **Portabilidad:** Las aplicaciones en contenedores de Docker son portátiles, lo que significa que puedes ejecutarlas en cualquier entorno que admita Docker, independientemente del sistema operativo o la infraestructura subyacente. Esto facilita la implementación en diferentes entornos de desarrollo, pruebas y producción.
- **Aislamiento:** Docker utiliza tecnologías de contenedorización para garantizar un alto grado de aislamiento entre las aplicaciones. Cada contenedor es independiente y comparte recursos con otros contenedores solo si se configura para hacerlo. Esto evita problemas de conflicto de dependencias y facilita la administración de aplicaciones.
- **Eficiencia:** Los contenedores son ligeros y comparten el mismo kernel del sistema operativo anfitrión. Esto hace que sean más eficientes en términos de recursos en comparación con las máquinas virtuales, lo que permite una implementación más rápida y una utilización más eficiente de los recursos del sistema.

- **Escalabilidad:** Docker facilita la escalabilidad de aplicaciones a través de la orquestación de contenedores, como Kubernetes o Docker Swarm. Esto permite agregar o quitar contenedores según la demanda, lo que es esencial para aplicaciones distribuidas y de alto tráfico.
- **Gestión de versiones:** Docker permite crear imágenes de contenedor que contienen aplicaciones y todas sus dependencias. Estas imágenes se pueden versionar y compartir, lo que simplifica la administración y el despliegue de aplicaciones en diferentes entornos.
- **Automatización:** Docker se integra con herramientas de automatización y orquestación, lo que permite la automatización de tareas de implementación, actualización y administración de contenedores, lo que ahorra tiempo y reduce errores.

Desventajas:

- **Aprendizaje inicial:** Para algunos usuarios, especialmente aquellos que no están familiarizados con la contenedorización, Docker puede tener una curva de aprendizaje inicial. La creación de imágenes de contenedor y la configuración pueden ser complejas.
- **Seguridad:** Si no se configuran adecuadamente, los contenedores pueden tener vulnerabilidades de seguridad. La seguridad en Docker es una preocupación, y es importante aplicar las mejores prácticas de seguridad para garantizar que los contenedores estén protegidos.
- **Tamaño de imágenes:** Las imágenes de contenedor pueden ser voluminosas si no se gestionan adecuadamente, lo que puede aumentar los requisitos de almacenamiento y transferencia de datos.
- **Compatibilidad limitada:** Aunque Docker es muy versátil, algunas aplicaciones pueden tener problemas de compatibilidad o requerir modificaciones para funcionar en un entorno de contenedor.
- **Recursos compartidos:** Si los recursos del sistema anfitrión no se administran adecuadamente, los contenedores pueden competir por recursos, lo que puede afectar el rendimiento de las aplicaciones.

Programa desarrollado:

Se desarrollo un programa sencillo el cual nos permitirá crear un servicio web (pagina) haciendo uso de Python y de una de sus librerías fastapi para el desarrollo, y el uso de Docker para tener un contenedor que nos permita realizar descarga de complementos, extensiones o bibliotecas necesarias para el programa, la idea es que pueda funcionar en cualquier equipo independientemente del sistema que maneje.

Archivo main.py:

El archivo main.py cuenta con la estructura de la pagina web, desde este archivo se podrán realizar los cambios estéticos, y agregado de elementos que sean necesarios para el propósito de la página (en este caso es solo de prueba por lo que solo se agregaran elementos sencillos, principalmente texto).

```

main.py X
main.py > read_root
1  from fastapi import FastAPI
2  from fastapi.responses import HTMLResponse
3
4  app = FastAPI()
5
6  @app.get("/", response_class=HTMLResponse)
7  async def read_root():
8      html_content = """
9      <!DOCTYPE html>
10     <html>
11     <head>
12         <title>Mi Aplicación FastAPI</title>
13     </head>
14     <body>
15         <header style="background-color: #007BFF; color: white; padding: 20px;">
16             <h1 style="text-align: center;">Bienvenido a mi aplicación FastAPI</h1>
17         </header>
18         <div style="margin: 20px;">
19             <p style="font-size: 18px; text-align: center;">Esta aplicación fue creada como una prueba de contenedores de
20         </div>
21
22         <div style="text-align: center;">
23             <button style="background-color: #007BFF; color: white; padding: 10px 20px; border: none; border-radius: 5px;">
24         </div>
25
26         <div style="text-align: center; margin-top: 20px;">
27             
28         </div>
29     </body>
30     </html>
31
32     """
33     return HTMLResponse(content=html_content)
34

```

Archivo requirements.txt:

Contiene los requerimientos del servidor y del framework web que se utiliza para la ejecución de la aplicación.

```

main.py requirements.txt X
requirements.txt
1  fastapi==0.59.0          # Framework web
2  uvicorn==0.11.5         # Servidor

```

Archivo Dockerfile:

Nos permitirá generar las especificaciones del programa en base a las necesidades, permitirá el llamado de ciertas instrucciones para la instalación con Docker ya sea de extensiones, servidores, aplicaciones etc.

```

main.py requirements.txt Dockerfile X
Dockerfile > FROM
1  FROM python:3.8.4-slim-buster
2  COPY . usr/src/app
3  WORKDIR /usr/src/app
4
5  RUN pip install -r requirements.txt
6
7  ENTRYPOINT uvicorn --host 0.0.0.0 main:app --reload

```

En este caso pide la instalación de requirements.txt haciendo uso de pip, esto como mencionaba era para el uso del servidor y el framework con el cual trabaja el programa, se especifica la versión de Python con la que trabajara y el host con el cual trabajara, se especifica el reload para cargar los cambios con forme se van realizando en el main.

Comandos de Docker:

Generamos primero el contenedor con el que manejaremos la aplicación.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMME
PS D:\Programacion\Docker> docker build -t simple_app .
```

Verificamos que se haya creado de manera correcta, lo que buscamos es que aparezca el nombre de simple_app.

En este caso se creó de manera correcta.

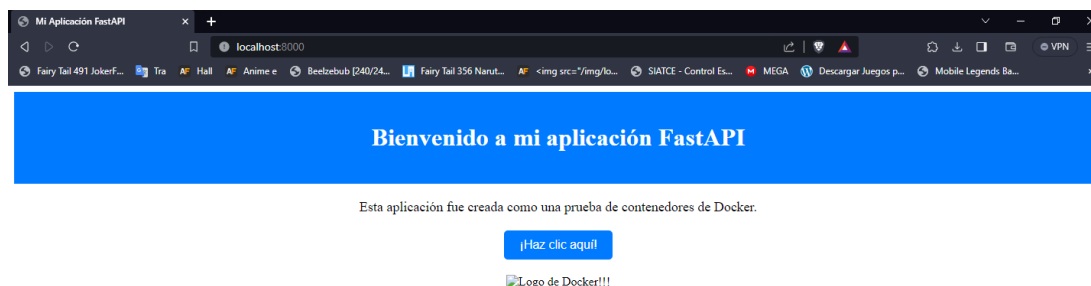
```
PS D:\Programacion\Docker> docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
simple_app            latest      f1914b13b1f1  About an hour ago  215MB
nginx                latest      bc649bab30d1  8 days ago     187MB
alpine               3.18.4     8ca4688f4f35  3 weeks ago    7.34MB
docker/welcome-to-docker latest      912b66cfd46e  4 months ago   13.4MB
nginx                1.23       a7be6198544f  5 months ago   142MB
PS D:\Programacion\Docker>
```

Una vez construido el contenedor vamos a hacer que la carpeta contenedora de nuestro proyecto sea tomada como volumen, para lo cual escribiremos el siguiente comando.

```
PS D:\Programacion\Docker> docker run -it -p 8000:8000 -v D:\Programacion\Docker:/usr/src/app simple_app
INFO:      Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [7] using statreload
INFO:      Started server process [9]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

Configuramos el puerto por el cual accederemos, la carpeta contenedora se convertirá en volumen del contenedor simple_app que habíamos creado e iniciaremos la ejecución de la pagina.

Como se muestra a continuación:



El puerto que pusimos es el 8000 por lo cual accedemos con localhost:8000 y nos muestra lo que generamos en el main, ya no es necesario instalar el requirements.txt ni lanzar el servidor de manera manual puesto con el archivo Docker lo podemos hacer de manera automática sin ningún problema, de igual manera si se requiriera instalar alguna otra extensión se podría automatizar, así pudiendo correr sin ningún problema el programa.

Actualizar main:

```

main.py > read_root
6 , response_class=HTMLResponse)
7 :ad_root():
8 :tent = ""
9 :E html>
10
11
12 :le>Mi Aplicación FastAPI</title>
13
14
15 :der style="background-color: #007BFF; color: white; padding: 20px;">
16 <h1 style="text-align: center;">He actualizado main y espero que se genere el cambio</h1>
17 :ader>
18 :r style="margin: 20px;">
19 <p style="font-size: 18px; text-align: center;">Esta aplicación fue creada como una prueba de contenedores de Docker.</p>
20 .v>
21
22 :r style="text-align: center;">
23 <button style="background-color: #007BFF; color: white; padding: 10px 20px; border: none; border-radius: 5px; font-size:
24 .v>
25

```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS	COMMENTS
alpine		3.18.4	8ca4688f4f35	3 weeks ago	7.34MB
docker/welcome-to-docker		latest	912b66cfd46e	4 months ago	13.4MB
nginx		1.23	a7be6198544f	5 months ago	142MB

```

PS D:\Programacion\Docker> docker run -it -p 8000:8000 -v D:\Programacion\Docker:/usr/src/app simple_app
INFO:      Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [7] using statreload
INFO:      Started server process [9]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      172.17.0.1:43872 - "GET / HTTP/1.1" 200 OK
INFO:      172.17.0.1:43872 - "GET /301_docker.jpg HTTP/1.1" 404 Not Found

```

Modifico un poco el texto, aun no realizo el cambio, una vez guarde se deberá volver a cargar y actualizar.

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS	COMMENTS
INFO:	Application startup complete.				
INFO:	172.17.0.1:43872 - "GET / HTTP/1.1" 200 OK				
INFO:	172.17.0.1:43872 - "GET /301_docker.jpg HTTP/1.1" 404 Not Found				
WARNING:	Detected file change in 'main.py'. Reloading...				
INFO:	Shutting down				
INFO:	Waiting for application shutdown.				
INFO:	Application shutdown complete.				
INFO:	Finished server process [9]				
INFO:	Started server process [11]				
INFO:	Waiting for application startup.				
INFO:	Application startup complete.				

Se realizo el cambio, se vuelve a iniciar la aplicación y reinicia con los cambios hechos.



En caso de requerir alguna instalación extra se puede hacer uso del Dockerfile para iniciar las instrucciones.

```

main.py requirements.txt Dockerfile X
Dockerfile > ...
1 FROM python:3.8.4-slim-buster
2 COPY . /usr/src/app
3 WORKDIR /usr/src/app
4
5 RUN pip install -r requirements.txt
6 RUN pip install request
7
8 ENTRYPOINT uvicorn --host 0.0.0.0 main:app --reload

```

Sin uso de Docker comandos:

En el caso de que no estuviéramos usando Docker sería necesario hacer las instalaciones de manera manual de los requerimientos para el arranque de la aplicación con el servidor.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS D:\Programacion\Docker> pip install -r requirements.txt
Requirement already satisfied: fastapi==0.59.0 in d:\anaconda\envs\pythondocker\lib\site-packages (from -r requirements.txt (line 1)) (0.59.0)
Requirement already satisfied: uvicorn==0.11.5 in d:\anaconda\envs\pythondocker\lib\site-packages (from -r requirements.txt (line 2)) (0.11.5)
Requirement already satisfied: starlette==0.13.4 in d:\anaconda\envs\pythondocker\lib\site-packages (from fastapi==0.59.0->-r requirements.txt (line 1)) (0.13.4)
Requirement already satisfied: pydantic<2.0.0,>=0.32.2 in d:\anaconda\envs\pythondocker\lib\site-packages (from fastapi==0.59.0->-r requirements.txt (line 1)) (1.10.13)
Requirement already satisfied: click==7.* in d:\anaconda\envs\pythondocker\lib\site-packages (from uvicorn==0.11.5->-r requirements.txt (line 2)) (7.1.2)
Requirement already satisfied: h11<0.10,>=0.8 in d:\anaconda\envs\pythondocker\lib\site-packages (from uvicorn==0.11.5->-r requirements.txt (line 2)) (0.9.0)
Requirement already satisfied: websockets==8.* in d:\anaconda\envs\pythondocker\lib\site-packages (from uvicorn==0.11.5->-r requirements.txt (line 2)) (8.1)
Requirement already satisfied: typing-extensions==4.2.0 in d:\anaconda\envs\pythondocker\lib\site-packages (from pydantic<2.0.0,>=0.32.2->fastapi==0.59.0->-r requirements.txt (line 1)) (4.2.0)
Ln 6, Col 24 Spaces: 4 UTF-8 CRLF Dockerfile
04:53 p.m. 20/10/2023

```

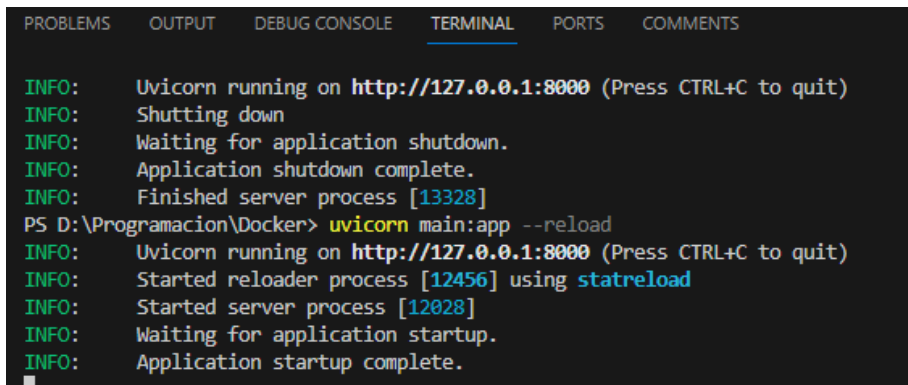
Y para el arranque se haría igual por medio de la siguiente instrucción cada que se realice un cambio.

```

PS D:\Programacion\Docker> uvicorn main:app
INFO: Started server process [13328]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

```

Esta instrucción no toma en cuenta los cambios que se hagan después de la ejecución.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Shutting down
INFO:      Waiting for application shutdown.
INFO:      Application shutdown complete.
INFO:      Finished server process [13328]
PS D:\Programacion\Docker> uvicorn main:app --reload
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [12456] using statreload
INFO:      Started server process [12028]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

En este caso si tomamos en cuenta cambios posteriores.

Conclusión:

Para este proyecto hice uso tanto de Docker como de fastapi para el desarrollo de la pagina web, realmente es una pagina muy sencilla la cual me permitió ver en mejor medida el funcionamiento de Docker y como logra facilitar muchas cosas, por ejemplo con el archivo Dockerfile puedo automatizar la ejecución de algunos comandos, tales como aquellos que me permiten ya sea instalar extensiones o actualizar estas mismas para un correcto funcionamiento de mi programa, también es posible descargar versiones de ciertos componentes de manera automática con lo cual se solucionan problemas de compatibilidad, es una herramienta realmente útil que me permitió desarrollar de manera más rápida y sin tanta complicación.

Bibliografía:

- Docker: Accelerated Container Application Development. (2023). *Docker*.

<https://www.docker.com/>