



Universidad de Guadalajara - CUCEI
Computación Tolerante a Fallas
Docker (Contenedores)

Contenido

1	Introducción	2
1.1	Ventajas de Docker	2
2	Desarrollo	3
2.1	Script (<i>dockerfile</i>)	3
3	Conclusión	6
4	Bibliografía	7

1 Introducción

Docker es una plataforma de software que te permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

La tecnología Docker utiliza el kernel de Linux y sus funciones, como los grupos de control y los espacios de nombre, para dividir los procesos y ejecutarlos de manera independiente. El propósito de los contenedores es ejecutar varios procesos y aplicaciones por separado para que se pueda aprovechar mejor la infraestructura y, al mismo tiempo, conservar la seguridad que se obtendría con los sistemas individuales.

Las herramientas de los contenedores, como Docker, proporcionan un modelo de implementación basado en imágenes. Esto permite compartir fácilmente una aplicación o un conjunto de servicios, con todas las dependencias en varios entornos. Docker también automatiza la implementación de las aplicaciones (o los conjuntos de procesos que las constituyen) en el entorno de contenedores.

Estas herramientas están diseñadas a partir de los contenedores de Linux, por eso la tecnología Docker es sencilla y única. Además, ofrecen a los usuarios acceso sin precedentes a las aplicaciones, la posibilidad de realizar implementaciones en poco tiempo y el control sobre las versiones y su distribución.

1.1 Ventajas de Docker

- Modularidad
 - El enfoque de Docker sobre la organización en contenedores se centra en la capacidad de separar una parte de la aplicación para actualizarla o repararla, sin necesidad de deshabilitarla por completo. Además de aprovechar este modelo basado en los microservicios, puede intercambiar procesos entre varias aplicaciones casi de la misma forma en que funciona la arquitectura orientada a los servicios (SOA).
- Capas y control de versiones de imágenes
 - Cada archivo de imagen Docker está compuesto por varias capas que conforman una sola imagen. Cuando un usuario especifica un comando, como ejecutar o copiar, la imagen cambia, y se crea una capa nueva.
 - Docker reutiliza las capas para agilizar el diseño de los contenedores nuevos. Los cambios intermedios se comparten entre las imágenes para mejorar aún más la agilidad, el tamaño y la eficiencia. El control de versiones también es propio de la creación de capas: el registro incorporado de los cambios le brinda el control total de las imágenes de contenedores cada vez que se produce una modificación.
- Restauración
 - Uno de los mayores beneficios de las capas es la capacidad de restauración. Todas las imágenes cuentan con capas. Si no le gusta la iteración actual de una imagen, puede restaurarla a una versión anterior. Esto respalda el enfoque de desarrollo ágil y permite lograr la integración e implementación continuas (CI/CD) desde la perspectiva de las herramientas.
- Implementación rápida
 - Antes, se necesitaban varios días para poner en marcha un sistema de hardware nuevo, implementarlo y ponerlo a disposición de los usuarios, lo cual implicaba un esfuerzo y un costo

abrumadores. Con los contenedores basados en Docker, la implementación se puede realizar en cuestión de segundos. Cada proceso se encuentra en un contenedor distinto, por lo que puede compartirlos con aplicaciones nuevas rápidamente. Además, ya que no es necesario iniciar el sistema operativo para agregar o trasladar un contenedor, los tiempos de implementación son mucho más cortos, y puede crear datos y eliminar aquellos que generen los contenedores de manera fácil y rentable, sin preocupaciones.

En definitiva, la tecnología Docker tiene un enfoque más detallado y controlable, que se basa en los microservicios y que prioriza la eficiencia.

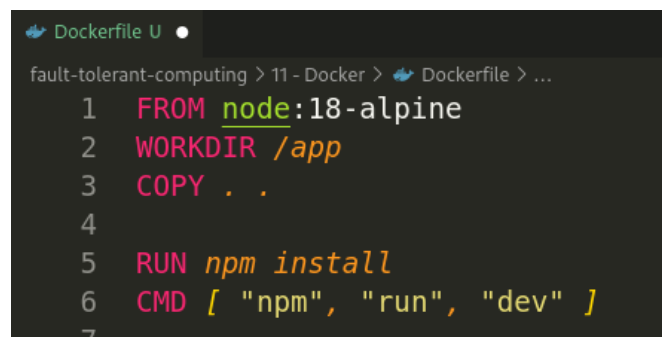
2 Desarrollo

2.1 Script (*dockerfile*)

Para el desarrollo de esta actividad, se desarrollo una aplicación simple que requiere de un servicio web activo para acceder a la misma, por lo que es un caso de estudio perfecto para implementar una solución con Docker.

El Dockerfile consta de 3 partes:

1. Imagen: En la primer línea definimos que imagen de Docker se utilizará para crear nuestro contenedor, para este caso partículas, debido a que nuestra app es utiliza NodeJs, utilizamos la imagen proporcionada por el mismo proyecto, utilizando la variación de *NodeJs 18* y *Alpine Linux*, que es la imagen de Linux más ligera.
2. Directorio de trabajo: En la 2da y 3ra línea definimos cual es el directorio principal que se utilizará como espacio de trabajo dentro de nuestro contenedor ('/app'), para después copiar todo nuestro proyecto (de nuestra maquina principal) a el espacio de trabajo dentro del contenedor.
3. Comandos: Las ultimas dos lineas son ejecutadas como comandos comunes de Linux, estás realizan la instalación de las dependencias del proyecto y finalmente inicializan el proyecto en modo desarrollo, de forma que nuestra app está, ahora, en ejecución.

A screenshot of a code editor showing a Dockerfile script. The editor has a dark background with syntax highlighting. The script consists of seven lines: 1. FROM node:18-alpine, 2. WORKDIR /app, 3. COPY . ., 4. (empty line), 5. RUN npm install, 6. CMD ["npm", "run", "dev"], 7. (empty line). The window title is 'Dockerfile U' and the breadcrumb is 'fault-tolerant-computing > 11 - Docker > Dockerfile > ...'.

```
1 FROM node:18-alpine
2 WORKDIR /app
3 COPY . .
4
5 RUN npm install
6 CMD [ "npm", "run", "dev" ]
7
```

Una vez se definió la estructura de nuestro contenedor, solo nos queda crear la imagen, iniciarla y utilizar nuestra app.

1. Construcción de imagen

```

felipe in ~/.../fault-tolerant-computing/11 - Docker on mainx λ docker build . -t docker-ftc:lastest
[+] Building 16.3s (9/9) FINISHED
=> [internal] load .dockerignore 1.2s
=> => transferring context: 2B 0.0s
=> [internal] load build definition from Dockerfile 1.6s
=> => transferring dockerfile: 182B 0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine 1.4s
=> [1/4] FROM docker.io/library/node:18-alpine@sha256:435dcad253bb5b7f347ebc6 0.0s
=> [internal] load build context 0.7s
=> => transferring context: 44.64kB 0.0s
=> CACHED [2/4] WORKDIR /app 0.0s
=> [3/4] COPY . . 1.7s
=> [4/4] RUN npm install 7.1s
=> exporting to image 1.7s
=> => exporting layers 1.6s
=> => writing image sha256:0b4315c2d9feabbde3c2c2ed7615dca4a26045802079865c41 0.0s
=> => naming to docker.io/library/docker-ftc:lastest 0.1s
felipe in ~/.../fault-tolerant-computing/11 - Docker on mainx λ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
docker-ftc latest 0b4315c2d9fe 8 seconds ago 183MB
<none> <none> 93aa0ad2a43a 21 minutes ago 182MB
felipe in ~/.../fault-tolerant-computing/11 - Docker on mainx λ

```

2. Inicialización y asignación de puerto a exponer

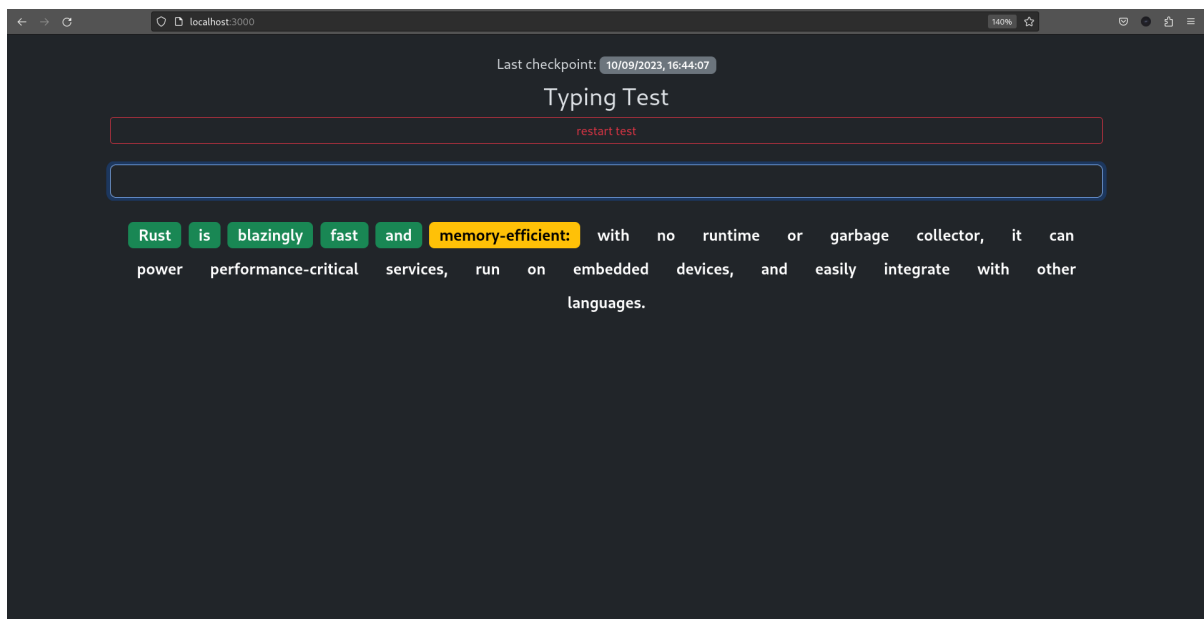
```

felipe in ~/.../fault-tolerant-computing/11 - Docker on mainx λ docker run -p 3000:3000 docker-ftc:lastest
> 06---application-checkpointing@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node src/index.js'

```

3. Revisión



3 Conclusión

Finalmente podemos denotar la importancia de Docker, ya que esta herramienta es importante muy importante, ya que revoluciona la forma en que desarrollamos, entregamos y ejecutamos aplicaciones. Proporciona un entorno más consistente, eficiente y seguro para el desarrollo y la implementación de software, lo que a su vez mejora la productividad, reduce los costos y permite la escalabilidad de las aplicaciones de manera efectiva.

4 Bilbiografia

1. What is docker? (n.d.). Redhat.com. Retrieved October 24, 2023, from <https://www.redhat.com/en/topics/containers/what-is-docker>