Soluciones

Arturo Silvelo

Try New Roads

Crear, arrancar y validar la imagen Docker

1. Construir la imagen

Ubícate en la carpeta de los ejercicios y ejecuta:

```
docker build -t ejercicio-avanzado .
```

2. Arrancar el contenedor

Ejecuta el siguiente comando para levantar la app y exponer el puerto 3000:

```
docker run -p 3000:3000 ejercicio-avanzado
```

3. Validar la respuesta

Accede a la app desde tu navegador en `http://localhost:3000` o usa `curl`:

```
curl -s http://localhost:3000
```

La respuesta esperada es:

```
{
  "mensaje": "¡Hola desde Docker!",
  "entorno": "produccion",
  "puerto": "3000",
  "secreto": "***",
  "pista": "El secreto real nunca debe mostrarse en producción."
}
```

Solución Ejercicio 1: Multi-stage build básico

Resumen de los cambios realizados

Se ha creado el fichero `Dockerfile.multi-stage` usando la técnica de multi-stage build. Esto permite instalar dependencias en una etapa y copiar solo los archivos necesarios a la imagen final, reduciendo el tamaño y mejorando la seguridad. Además, se mantienen las variables de entorno por defecto para la app.

1. Creación de la imagen

```
# Construir solo hasta la fase de build
docker build --target build -f Dockerfile.multi-stage -t ejercicio-build .

docker build -f Dockerfile.multi-stage -t ejercicio-multistage .
```

2. Inicio del contenedor Docker

Ejecuta el siguiente comando para levantar la app y exponer el puerto 3000:

```
# Ejecutar la imagen build, como no tiene CMD hay que añadirlo
docker run -p 3000:3000 ejercicio-build node app.js

docker run -p 3000:3000 ejercicio-multistage
```

3. Validar la respuesta

Accede a la app desde tu navegador en `http://localhost:3000` o usa `curl`:

```
curl -s http://localhost:3000
```

La respuesta esperada es:

```
{
  "mensaje": "¡Hola desde Docker!",
  "entorno": "produccion", // ó development dependiendo de la imagen usada
  "puerto": "3000",
  "secreto": "***",
  "pista": "El secreto real nunca debe mostrarse en producción."
}
```

Solución Ejercicio 2: Optimización de capas

Resumen de los cambios realizados

Se ha optimizado el Dockerfile para aprovechar mejor la cache de Docker y reducir el número de capas innecesarias:

- Se instala primero solo las dependencias necesarias (`npm install --omit=dev`) antes de copiar el código fuente.
- Solo se copian los archivos requeridos para producción.
- Se usa una imagen final ligera y segura.

1. Construcción de la imagen optimizada

```
docker build -f Dockerfile.optimizado -t ejercicio-optimizado .
```

2. Arrancar el contenedor

```
docker run -p 3000:3000 ejercicio-optimizado
```

3. Validar la respuesta

Accede a la app desde tu navegador en `http://localhost:3000` o usa `curl`:

```
curl -s http://localhost:3000
```

La respuesta esperada es:

```
{
  "mensaje": "¡Hola desde Docker!",
  "entorno": "produccion",
  "puerto": "3000",
  "secreto": "***",
  "pista": "El secreto real nunca debe mostrarse en producción."
}
```

4. Comprobación de la optimización de capas

- Analiza las capas con:

```
docker history ejercicio-optimizado
```

- Haz un cambio en el código fuente y vuelve a construir la imagen para comprobar que Docker reutiliza las capas previas y el build es más rápido.

Solución Ejercicio 3: Uso de ARG y ENV

Resumen de los cambios realizados

Se ha modificado el Dockerfile para permitir la personalización de variables de entorno y argumentos de build:

- Se añaden instrucciones ARG y ENV para el puerto (`PORT`) y el entorno (`NODE_ENV`).
- Permite cambiar estos valores tanto en la construcción de la imagen como en la ejecución del contenedor.
- El secreto `DB_PASSWORD` debe pasarse solo en tiempo de ejecución, nunca en el Dockerfile.

1. Construcción de la imagen con argumentos personalizados

```
docker build --build-arg PORT=4000 --build-arg NODE_ENV=staging -f Dockerfile.arg-env -t ejercicio-arg-env .
```

2. Arrancar el contenedor con variables de entorno personalizadas

```
docker run -p 4000:4000 -e PORT=4000 -e NODE_ENV=staging -e DB_PASSWORD=valor_secreto ejercicio-arg-env
```

3. Validar la respuesta

```
curl -s http://localhost:4000
```

La respuesta esperada es:

```
{
  "mensaje": "¡Hola desde Docker!",
  "entorno": "staging",
  "puerto": "4000",
  "secreto": "***",
  "pista": "El secreto real nunca debe mostrarse en producción."
}
```

Solución Ejercicio 4: Gestión segura de secretos

Resumen de los cambios realizados

Se ha modificado el Dockerfile para no incluir el secreto en la imagen ni en el Dockerfile. El secreto debe pasarse solo en tiempo de ejecución.

1. Variable de entorno en tiempo de ejecución

Pasa el secreto directamente al arrancar el contenedor:

```
docker run -p 3000:3000 -e DB_PASSWORD=valor_secreto ejercicio-secreto
```

La aplicación lo leerá como `process.env.DB_PASSWORD`.

2. Archivo externo montado como volumen

Guarda el secreto en un archivo local, por ejemplo `db_password.txt`, y móntalo en el contenedor:

```
docker run -p 3000:3000 -v /ruta/local/db_password.txt:/run/secrets/db_password.txt ejercicio-secreto
```

La aplicación debe leer el secreto desde el archivo `/run/secrets/db_password.txt` dentro del contenedor.