Contenerizar una aplicación

Durante el resto de esta guía, trabajará con un administrador de tareas sencillo que se ejecuta en Node.js. Si no estás familiarizado con Node.js, no te preocupes. Esta guía no requiere ninguna experiencia previa con JavaScript.

Antes de poder ejecutar la aplicación, necesita obtener el código fuente de la aplicación en su máquina.

1. Clonar el repositorio de la aplicación usando el siguiente comando:

   $ git clone https://github.com/docker/getting-started-app.git

2. Ver el contenido del repositorio clonado. Debería ver los siguientes archivos y subdirectorios.

   ├── getting-started-app/

   | ├── package.json

   | ├── README.md

   | ├── spec/

   | ├── src/

   | └── yarn.lock

Para crear la imagen, necesitará utilizar un Dockerfile. Un Dockerfile es simplemente un archivo basado en texto sin extensión de archivo que contiene un script de instrucciones. Docker usa este script para crear una imagen de contenedor.

1. En el directorio getting-started-app, la misma ubicación que el archivo package.json, cree un archivo llamado Dockerfile. Puede utilizar los siguientes comandos para crear un Dockerfile.

En la terminal, ejecute los siguientes comandos.

Asegúrate de estar en el getting-started-appdirectorio. Reemplace /path/to/getting-started-app con la ruta a su getting-started-appdirectorio.

$ cd getting-started-app

Cree un archivo vacío llamado Dockerfile.

$ touch Dockerfile

2. Usando un editor de texto o editor de código, agregue el siguiente contenido al Dockerfile:

```
# syntax=docker/dockerfile:1


FROM node:18-alpine

WORKDIR /app

COPY . .

RUN yarn install --production

CMD ["node", "src/index.js"]

EXPOSE 3000
```

3. Construya la imagen usando los siguientes comandos:

En la terminal, asegúrate de estar en el directorio getting-started-app

$ cd getting-started-app

Construye la imagen.

$ docker build -t getting-started .

El docker build utiliza Dockerfile para crear una nueva imagen. Es posible que hayas notado que Docker descargó muchas "capas". Esto se debe a que le indicó al constructor que quería comenzar desde la node:18-alpine. Pero, como no lo tenía en su máquina, Docker necesitaba descargar la imagen.

Después de que Docker descargó la imagen, las instrucciones del Dockerfile se copiaron en su aplicación y se usaron yarn para instalar las dependencias de su aplicación. La CMD especifica el comando predeterminado que se ejecutará al iniciar un contenedor desde esta imagen.

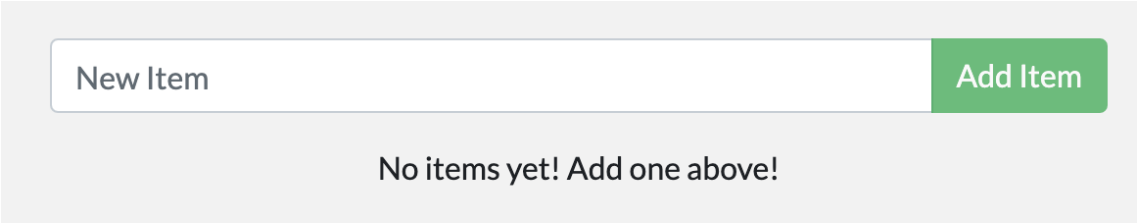
Finalmente, la -t etiqueta tu imagen. Piense en esto como un nombre legible por humanos para la imagen final. Dado que nombró la imagen getting-started, puede hacer referencia a esa imagen cuando ejecute un contenedor.

Ahora que tiene una imagen, puede ejecutar la aplicación en un contenedor usando el docker run.

1. Ejecute su contenedor usando el comando docker run y especifique el nombre de la imagen que acaba de crear:
2. $ docker run -p 3000:3000 getting-started

La -p crea una asignación de puertos entre el host y el contenedor. La -pbandera toma un valor de cadena en el formato de HOST:CONTAINER, donde HOST es la dirección en el host y CONTAINER es el puerto en el contenedor. Sin la asignación de puertos, no podrá acceder a la aplicación desde el host.

3. Después de unos segundos, abra su navegador web



4. Agregue uno o dos elementos y compruebe que funciona como espera. Puede marcar elementos como completos y eliminarlos. Su interfaz está almacenando elementos correctamente en el backend.

En este punto, tiene un administrador de lista de tareas pendientes en ejecución con algunos elementos.

Si echa un vistazo rápido a sus contenedores, debería ver al menos un contenedor ejecutándose que utiliza la getting-started y el puerto 3000.

Ejecute el siguiente docker ps en una terminal para enumerar sus contenedores.

$ docker ps

Debería aparecer un resultado similar al siguiente.

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

df784548666d getting-started "docker-entrypoint.s…" 2 minutes ago Up 2 minutes 127.0.0.1:3000->3000/tcp priceless_mcclintock

Actualizar la aplicación

En esta seccion, actualizarás la aplicación y la imagen. También aprenderás cómo detener y retirar un contenedor.

Actualizar el código fuente

En los siguientes pasos, cambiará el "texto vacío" cuando no tenga ningún elemento de la lista de tareas pendientes a "¡Aún no tiene elementos de tarea pendiente! ¡Agregue uno arriba!".

1. En el src/static/js/app.jsarchivo, actualice la línea 56 para usar el nuevo texto vacío.

   - No items yet! Add one above!

   + You have no todo items yet! Add one above!

2. Cree su versión actualizada de la imagen usando el comando docker build.

   $ docker build -t getting-started .

3. Inicie un nuevo contenedor usando el código actualizado.

   $ docker run -p 3000:3000

4. Probablemente hayas visto un error como este:

docker: Error response from daemon: driver failed programming external connectivity on endpoint laughing_burnell

(bb242b2ca4d67eba76e79474fb36bb5125708ebdabd7f45c8eaf16caaabde9dd): Bind for 127.0.0.1:3000 failed: port is already allocated.

El error se produjo porque no puede iniciar el nuevo contenedor mientras el contenedor anterior aún se está ejecutando. La razón es que el contenedor antiguo ya está usando el puerto 3000 del host y solo un proceso en la máquina (incluidos los contenedores) puede escuchar un puerto específico. Para solucionar este problema, debe quitar el contenedor antiguo.

Para retirar un contenedor, primero debe detenerlo. Una vez que se haya detenido, puedes eliminarlo. Puede eliminar el contenedor antiguo utilizando la CLI o la interfaz gráfica de Docker Desktop. Elija la opción con la que se sienta más cómodo.

1. Obtenga el ID del contenedor mediante el docker pscomando.

   $ docker ps

2. Utilice el comando docker stop para detener el contenedor. Reemplace <the-container-id>con la identificación de docker ps.

   $ docker stop <the-container-id>

3. Una vez que el contenedor se haya detenido, puede eliminarlo usando el comando docker rm.

   $ docker rm <the-container-id>

   Ahora, inicie su aplicación actualizada usando el docker runcomando.

   $ docker run -dp 127.0.0.1:3000:3000 getting-started

   Actualiza tu navegador y deberías ver el texto de ayuda actualizado.

Comparte la aplicación

Ahora que ha creado una imagen, puede compartirla. Para compartir imágenes de Docker, debe utilizar un registro de Docker. El registro predeterminado es Docker Hub y es de donde provienen todas las imágenes que ha utilizado.

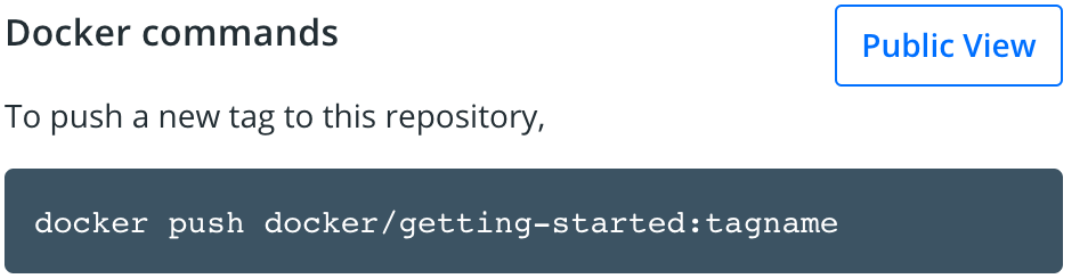
ID de ventana acoplable

Una ID de Docker le permite acceder a Docker Hub, que es la biblioteca y comunidad de imágenes de contenedores más grande del mundo.

Para enviar una imagen, primero debe crear un repositorio en Docker Hub.

1. Inicie sesión en Docker Hub .
2. Seleccione el botón **Crear repositorio** .
3. Para el nombre del repositorio, utilice getting-started. Asegúrese de que la **visibilidad** sea **pública** .
4. Seleccione **Crear** .

En la siguiente imagen, puede ver un comando Docker de ejemplo de Docker Hub. Este comando enviará a este repositorio.



1. En la línea de comando, ejecute el docker pushcomando que ve en Docker Hub. Tenga en cuenta que su comando tendrá su ID de Docker, no "docker".

   $ docker push docker/getting-started

   The push refers to repository [docker.io/docker/getting-started]

   An image does not exist locally with the tag: docker/getting-started

¿Por qué fracasó? El comando push estaba buscando una imagen llamada docker/getting-started, pero no encontró ninguna. Si corres docker image ls, tampoco verás ninguno.

Para solucionar este problema, debe etiquetar la imagen existente que creó para darle otro nombre.

2. Inicie sesión en Docker Hub usando el comando docker login -u YOUR-USER-NAME.
3. Utilice el comando docker tag para darle a la imagen getting-started un nuevo nombre. Reemplácelo YOUR-USER-NAME con su ID de Docker.

   $ docker tag getting-started YOUR-USER-NAME/getting-started

4. Ahora ejecute el comando docker push nuevamente. Si está copiando el valor de Docker Hub, puede descartar la parte tagname, ya que no agregó una etiqueta al nombre de la imagen. Si no especifica una etiqueta, Docker usa una etiqueta llamada latest.

   $ docker push YOUR-USER-NAME/getting-started

Persistir la base de datos

En caso de que no lo hayas notado, tu lista de tareas pendientes está vacía cada vez que inicias el contenedor. ¿Por qué es esto? En esta parte, profundizará en cómo funciona el contenedor.

Cuando se ejecuta un contenedor, utiliza las distintas capas de una imagen para su sistema de archivos. Cada contenedor también tiene su propio "espacio temporal" para crear/actualizar/eliminar archivos. Los cambios no se verán en otro contenedor, incluso si usan la misma imagen.

Para ver esto en acción, iniciará dos contenedores y creará un archivo en cada uno. Lo que verás es que los archivos creados en un contenedor no están disponibles en otro.

1. Inicie un contenedor Ubuntu que creará un archivo /data.txt con un nombre aleatorio entre 1 y 10000.

2. $ docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /data.txt && tail -f /dev/null"

En caso de que tenga curiosidad acerca del comando, está iniciando un shell bash e invocando dos comandos (por eso tiene el &&). La primera parte elige un único número aleatorio y lo escribe en /data.txt. El segundo comando consiste simplemente en observar un archivo para mantener el contenedor en funcionamiento.

3. Valide que puede ver el resultado accediendo a la terminal en el contenedor. Para hacerlo, puede utilizar la CLI o la interfaz gráfica de Docker Desktop.

En la línea de comando, use el comando docker exec para acceder al contenedor. Necesita obtener la identificación del contenedor (úsela docker ps para obtenerla).

$ docker exec <container-id> cat /data.txt

Deberías ver un número aleatorio.

4. Ahora, inicia otro Ubuntu contenedor (la misma imagen) y verás que no tienes el mismo archivo.

5. $ docker run -it ubuntu ls /

En este caso, el comando enumera los archivos en el directorio raíz del contenedor. ¡Mira, no hay ningún data.txtarchivo ahí! Esto se debe a que se escribió en el espacio temporal solo para el primer contenedor.

6. Continúe y elimine el primer contenedor usando el docker rm -f <container-id>comando.

Con el experimento anterior, viste que cada contenedor comienza desde la definición de la imagen cada vez que comienza. Si bien los contenedores pueden crear, actualizar y eliminar archivos, esos cambios se pierden cuando elimina el contenedor y Docker aísla todos los cambios en ese contenedor. Con los volúmenes, puedes cambiar todo esto.

Los volúmenes brindan la capacidad de conectar rutas específicas del sistema de archivos del contenedor a la máquina host. Si monta un directorio en el contenedor, los cambios en ese directorio también se ven en la máquina host. Si monta ese mismo directorio al reiniciar el contenedor, verá los mismos archivos.

Hay dos tipos principales de volúmenes. Eventualmente usarás ambos, pero comenzarás con montajes de volumen.

De forma predeterminada, la aplicación de tareas pendientes almacena sus datos en una base de datos SQLite en /etc/todos/todo.db sistema de archivos del contenedor. Si no estás familiarizado con SQLite, ¡no te preocupes! Es simplemente una base de datos relacional que almacena todos los datos en un solo archivo. Si bien esto no es lo mejor para aplicaciones a gran escala, funciona para demostraciones pequeñas. Más adelante aprenderá cómo cambiar esto a un motor de base de datos diferente.

Dado que la base de datos es un solo archivo, si puede conservar ese archivo en el host y ponerlo a disposición del siguiente contenedor, debería poder continuar donde lo dejó el último. Al crear un volumen y adjuntarlo (a menudo llamado "montarlo") al directorio donde almacenó los datos, puede conservar los datos. A medida que su contenedor escriba en el todo.db archivo, los datos persistirán en el host del volumen.

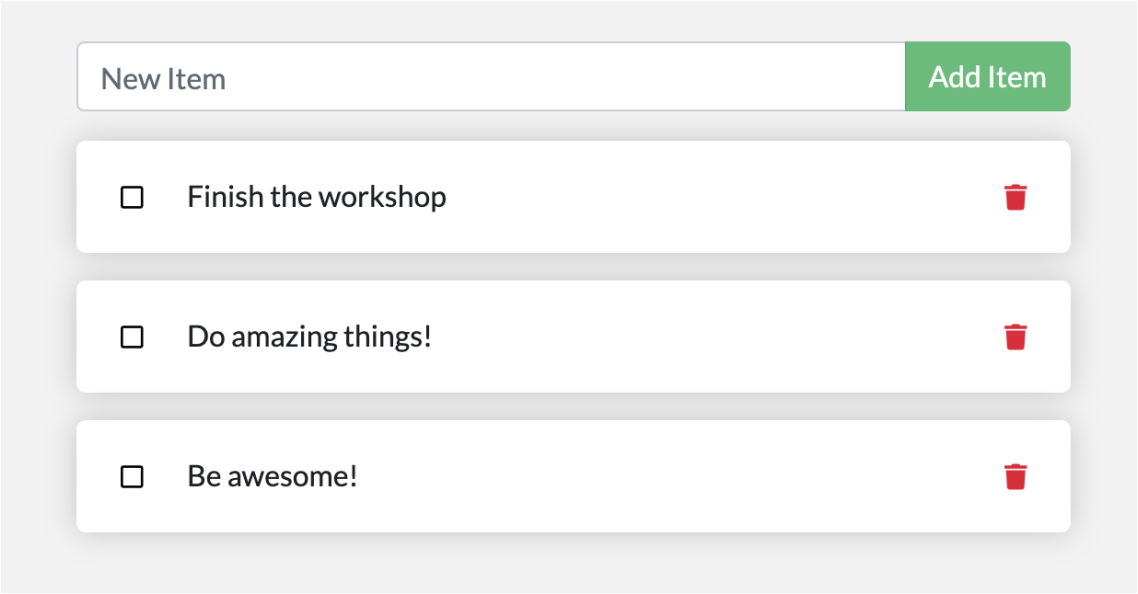
Como se mencionó, vas a utilizar un soporte de volumen. Piense en un montaje de volumen como un depósito opaco de datos. Docker administra completamente el volumen, incluida la ubicación de almacenamiento en el disco. Sólo necesitas recordar el nombre del volumen.

Puede crear el volumen e iniciar el contenedor utilizando la CLI o la interfaz gráfica de Docker Desktop.

Cree un volumen usando el comando docker volume créate.

1. $ docker volume create todo-db
2. Detenga y elimine el contenedor de la aplicación de tareas pendientes nuevamente con docker rm -f <id>, ya que todavía se está ejecutando sin usar el volumen persistente.
3. Inicie el contenedor de la aplicación de tareas pendientes, pero agregue la opción –mount para especificar un montaje de volumen. Asigne un nombre al volumen y móntelo /etc/todos en el contenedor, que captura todos los archivos creados en la ruta. En su terminal Mac o Linux, o en el símbolo del sistema de Windows o PowerShell, ejecute el siguiente comando:
4. $ docker run -p 3000:3000 --mount type=volume,src=todo-db,target=/etc/todos getting-started

   Una vez que se inicie el contenedor, abra la aplicación y agregue algunos elementos a su lista de tareas pendientes.



1. Detenga y elimine el contenedor de la aplicación de tareas pendientes. Utilice Docker Desktop o docker ps para obtener la identificación y luego docker rm -f <id>eliminarla.
2. Inicie un nuevo contenedor siguiendo los pasos anteriores.
3. Abre la aplicación. Deberías ver tus artículos todavía en tu lista.
4. Continúe y retire el contenedor cuando haya terminado de revisar su lista.

Ahora ha aprendido cómo conservar datos.

Mucha gente pregunta con frecuencia "¿Dónde almacena Docker mis datos cuando uso un volumen?" Si quieres saberlo, puedes usar docker volume inspect.

$ docker volume inspect todo-db

```
[
    {
        "CreatedAt": "2019-09-26T02:18:36Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/todo-db/_data",
        "Name": "todo-db",
        "Options": {},
        "Scope": "local"
    }
]
```

Es Mountpoint la ubicación real de los datos en el disco. Tenga en cuenta que en la mayoría de las máquinas necesitará tener acceso de root para acceder a este directorio desde el host.