Dockerize su aplicación Java

Garrett Sweeney
6 de diciembre de 2018 · 6 min de lectura

A medida que todos y su hermano se mudan a Kubernetes, es bueno saber cómo Dockerize aplicaciones. Este es un pequeño tutorial sobre cómo crear un Dockerfile para un proyecto spring.io/guides, construir una imagen, enviar nuestra imagen a dockerhub y ejecutar nuestra aplicación en contenedores localmente.

Una palabra rápida sobre Docker

¿Por qué Docker? Resuelve el problema de "se ejecuta en mi máquina". En lugar de entregar frascos, guerras, lo que sea, entregas "imágenes".

Todo lo que alguien necesita para ejecutar su "imagen" en su máquina es Docker instalado.

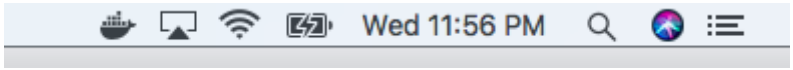
Esto es increíble porque el estándar anterior era entregar artefactos, lo que significa que había una dependencia en la configuración del entorno y las dependencias instaladas. ¡Pero ahora, todo eso se entrega a tu imagen! "Se ejecuta en mi máquina" ahora es mucho menos que nada.

Obtener configuración local

Estoy trabajando en una Mac, así que daré una breve descripción de cómo me configuré. Si estás trabajando en otra cosa, todo esto es bastante apto para Google.

Primero, cree una cuenta en https://cloud.docker.com/ . Navegue a la sección Repositorios. Por supuesto, estará vacío, pero tenga en cuenta que aquí es donde empujaremos nuestras imágenes.

Luego seguí las instrucciones en el sitio web de Docker para instalar Docker para Mac. Es bastante simple, pero tendrás una ballena disponible como a continuación:



¡Mira! Una ballena a la izquierda

De todos modos, haga clic en la ballena e inicie sesión con las mismas credenciales que creó al registrarse en cloud.docker.com.

A continuación, asegúrese de tener instalada la ventana acoplable ejecutándola con el siguiente comando en su terminal:

```
$ docker --version
Docker versión 18.09.0-ce, compilación 4d60db4
```

Impresionante, adelante!

Clonar una aplicación Java de muestra

Como dije en la introducción, voy a clonar una aplicación spring.io/guides de muestra. Luego construiré el Dockerfile para dockerizar esta aplicación de Java.

Continúe y clone el proyecto "Servir contenido web con Spring MVC":

```
git clone https://github.com/spring-guides/gs-serving-web-content.git
```

Las guías de primavera tienen varios proyectos enterrados en lo que clonas. Navegue hasta el proyecto completado y agregue un archivo vacío en este directorio llamado `Dockerfile`

```
cd ./complete
touch Dockerfile
```

También me gusta ver la estructura de lo que he clonado `tree` . Su directorio actual ahora debería verse así.

```
.
├── Dockerfile
├── README.md
├── mvnw
├── mvnw.cmd
├── pom.xml
└── src
    ├── principal
    │ ├── java
    │ │ └── hola
    │ │ ├── Application.java
    │ │ └── GreetingController.java
    │ └── recursos
    │ ├── estático
    │ │ └── index.html
    └── └── plantillas
    │ └── greeting.html
    └── prueba
        └── java
            └── hola
                └── ApplicationTest.java
```

porción procesar-mvc.sh alojada con ♡ por GitHub sinver la

Otra palabra en Docker

Antes de comenzar a agregar comandos al Dockerfile, déjame darte una idea general de lo que hace un Dockerfile.

Primero, un Dockerfile siempre comienza con otra imagen llamada "imagen base". Este es el componente básico de nuestra imagen, y los ejemplos incluyen cosas como una imagen de Red Hat o una imagen de Ubuntu. Esta imagen puede ser tan gruesa o delgada como desee, ya que podemos agregar cosas a esta imagen o construir "capas".

Entonces, si comienza con una imagen que ya tiene Java instalado, no tendremos que instalar Java más adelante en el Dockerfile. Si no comienza con una imagen que ya está instalada, tendremos que instalar Java. Al final, este Dockerfile se usará para construir otra imagen. Esta imagen final (con todas las dependencias y su aplicación agregada) se puede entregar a aquellos que desean ejecutar su aplicación.

Construyendo el Dockerfile

Como acabo de decir, tenemos que elegir una imagen para comenzar. Voy a comenzar con Ubuntu. Para hacer que nuestra imagen base sea Ubuntu, agregue esta línea:

```
DE ubuntu: último
```

¿Qué es lo último? Las imágenes tienen etiquetas. A menudo son versiones, pero "último" extraerá la última imagen de ubuntu publicada en DockerHub. ¿Y qué es DockerHub? ¡Un registro público de imágenes disponibles para su uso!

Ahora necesitamos instalar una dependencia, la jre (que se requiere en cualquier máquina para ejecutar una aplicación java). Los archivos Docker tienen un `RUN` comando que podemos utilizar como si estuviéramos instalando un jre en una máquina ubuntu desde la línea de comandos.

```
EJECUTAR \
# Actualizar
apt-get update -y && \
# Instalar Java
apt-get install default-jre -y
```

Impresionante, ahora nuestra imagen tiene Java instalado.

Ahora, ¿dónde está nuestro archivo jar? Si recuerda nuestra estructura de proyecto, tenemos una aplicación Java localmente. Vamos a construirlo con `mvn clean install`. Esto debería construir su aplicación Java y colocar el jar en su `./target/gs-serving-web-content-0.1.0.jar` directorio.

Y ahora que conocemos la ruta relativa de nuestro `jar` a nuestro Dockerfile, podemos utilizar el `ADD` comando Docker .

```
AGREGAR ./target/gs-serving-web-content-0.1.0.jar spring-mvc-example.jar
```

Este comando agrega el `./target/gs-serving-web-content-0.1.0.jar` artefacto local como `spring-mvc-example.jar`.

Cuando nuestra aplicación se está ejecutando, se está ejecutando en el puerto `8080`. Vamos a usar el `EXPOSE` comando docker para abrir ese puerto.

```
EXPOSE 8080
```

Y finalmente, ¡corramos el frasco! Utiliza el `CMD` comando docker para ejecutar nuestro jar.

```
CMD java -jar spring-mvc-example.jar
```

En conjunto, su Dockerfile debería verse así:

```
# Tire de la imagen base.
DE ubuntu: último

CORRER \
# Actualización
apt-get update -y && \
# Instalar Java
apt-get install default-jre -y

AGREGAR ./target/gs-serving-web-content-0.1.0.jar spring-mvc-example.jar

```

```
12  EXPOSE 8080
13
14  CMD java -jar spring-mvc-example.jar
```

archivo java-docker sin procesar alojado con ♡ por GitHub ver el

Una tercera palabra en Docker

En este punto, espero que se dé cuenta del beneficio de Docker, ya que podemos controlar la máquina donde se ejecuta nuestra aplicación.

Anteriormente, pasaríamos un `jar` archivo y esperamos que jar encuentre un lugar para vivir que esté configurado correctamente.

Ahora podemos usar esto `Dockerfile` para generar un `image` comportamiento que pueda ofrecer un comportamiento coherente en cualquier entorno que sea compatible con Docker.

Vamos a construir y ejecutar la imagen localmente

Reitero que necesitamos tener nuestro frasco construido en este punto. Si no ha creado el proyecto java, hágalo ahora:

```
java -version
mvn --version
mvn clean install
```

Para construir la imagen con el nombre `spring-mvc-sample-image` :

```
Docker build. -t spring-mvc-sample-image
```

Puede verificar la imagen construida correctamente con `docker images` :

```
$ docker images

REPOSITORY TAG ID DE IMAGEN TAMAÑO CREADO

spring-mvc-sample-image última 8fa27ad00edd Hace 34 minutos 540MB
```

Ahora, veamos un contenedor llamado `sample-mvc-sample-container` basado en su imagen `spring-mvc-sample-image` .

```
docker run -t -p 8080: 8080 --name sample-mvc-sample-container
spring-mvc-sample-image
```

Puede agregar `-d` a su Docker ejecutar comando para ejecutar el contenedor en segundo plano. Sin embargo, si ejecuta el comando anterior (con `-t`), se encontrará en el contenedor (y verá los registros de la aplicación Spring). Con -t, debería ver los registros que dicen que la aplicación se inicia correctamente. Con -d, tendrá el comando `docker ps -a` para encontrar su contenedor y `docker logs <container_id>` ver los registros de la aplicación Spring.

Use `ctrl + c` para salir del contenedor. Ahora manda `docker ps -a` .

```
$ docker ps -a
CONTENEDOR ID IMAGEN COMANDO ESTADO CREADO PUERTOS NOMBRES
736f9cdc1499 spring-mvc-sample-image "/ bin / sh -c 'java -j…" Hace
2 horas Arriba 2 horas 8080 / tcp sample-mvc-sample-container
```

Ahí está tu contenedor! Está funcionando. Justo allí tienes una máquina ubuntu que ejecuta tu aplicación java. Si desea volver al interior del contenedor que ejecuta su aplicación java, ordene `docker exec -it <container-id> /bin/bash` .

Finalmente, etiquete y envíe la imagen al público Dockerhub

Primero, enumera la imagen y encuentra la que construiste

```
$ docker images

REPOSITORY TAG ID DE IMAGEN TAMAÑO CREADO

spring-mvc-sample-image última 8fa27ad00edd Hace 34 minutos 540MB
```

A continuación, etiquete la imagen agarrando la identificación de la imagen `(8fa27ad00edd)` y etiquetándola con su nombre de usuario de Docker con el nombre que elija. Mi nombre de usuario de Docker es `gsweene2` y el nombre con el que lo etiqueto es

`gsweene2/spring-mvc-sample-image:0.1` . Una etiqueta acoplable consta de ay `name` a `version` , separadas por a `:` .

```
etiqueta del acoplador 8fa27ad00edd gsweene2 / spring-mvc-sample-image: 0.1
```

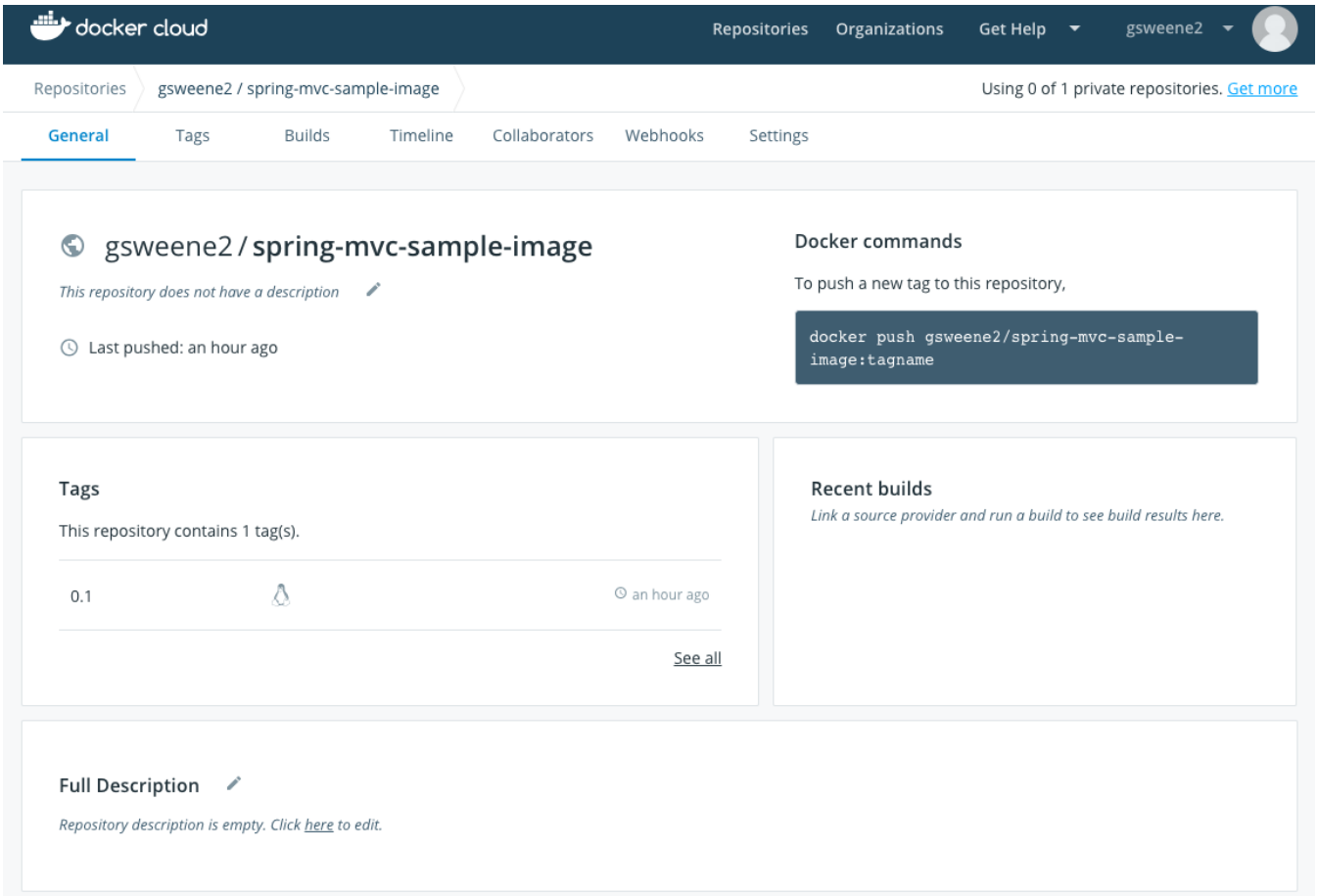
Finalmente, empuja! Use el mismo nombre con el que etiquetó la imagen, como `gsweene2/spring-mvc-sample-image:0.1` .

```
docker push gsweene2 / spring-mvc-sample-image: 0.1
```

Encuentra tu imagen en DockerHub

Debería poder encontrarlo en cloud.docker.com después de iniciar sesión, pero el patrón de URL parece ser:

```
https://cloud.docker.com/repository/docker/<username>/<image_name>
```



¿Alguna pregunta? Esto está bien documentado en Internet, ¡pero contácteme en garrett.d.sweeney@gmail.com!

Únete a nuestra comunidad Slack y lee nuestros temas semanales de Fauno ⬇

Si esta publicación fue útil, haga clic en el botón de aplaudir below a continuación para mostrar su apoyo al autor. ⬇

Estibador AWS Primavera Java Kubernetes

Sobre Ayuda Legal