

Universidad de Alicante

Contenedores en sistemas embebidos

Tanase Dragos



Sistemas Embebidos

dt23@alu.ua.es

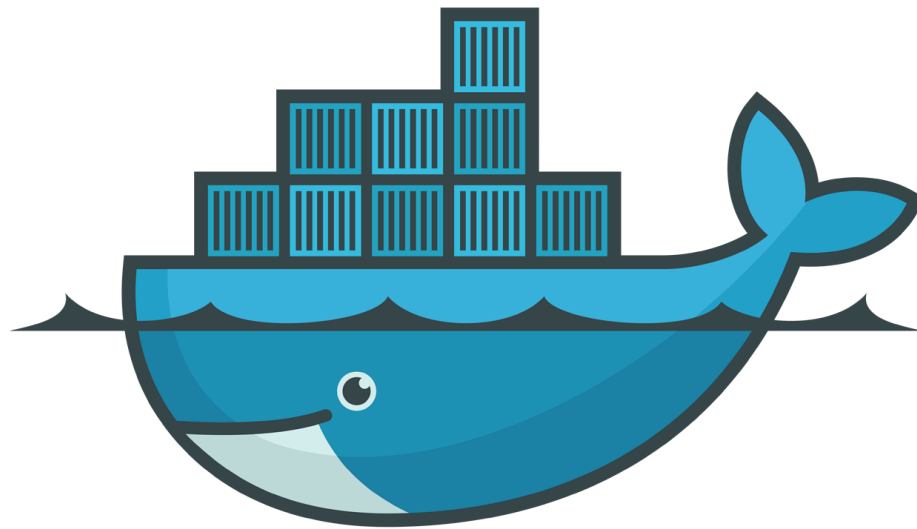
2021/04/28

Contenido

Introducción.....	3
Docker en Raspberry Pi.....	4
Docker en escritorio.....	5
Mi primera API.....	5
Dockerizar una API	7
Lanzamiento automático con Compose.....	9

Introducción

En esta práctica vamos a explorar como mediante el uso de Docker vamos a poder crear, probar e implementar aplicaciones de forma rápida. Los contenedores de Docker son software empaquetado con todo lo que necesita para trabajar correctamente, de forma que se pueda escalar de forma rápida con la seguridad de que todo funcionara correctamente.



docker

Docker en Raspberry Pi

Lo primero que haremos en esta práctica será instalar Docker en una Raspberry Pi 3B+ y ejecutar un contenedor para comprobar que funcione correctamente. Procedemos con la instalación ejecutando el siguiente comando:

```
curl -fsSL https://get.docker.com | sh
```

Una vez lo hayamos instalado, podremos comprobar con el comando “*docker --versión*” que este se haya instalado correctamente

```
usuarioSeguro@RPiPapito:~ $ docker --version
Docker version 20.10.6, build 370c289
usuarioSeguro@RPiPapito:~ $
```

Ahora vamos a ejecutar un contenedor de prueba para estar seguros del todo que funciona.

```
docker run armhf/hello-world
```

```
usuarioSeguro@RPiPapito:~ $ sudo docker run armhf/hello-world
sudo: unable to resolve host RPiPapito: Name or service not known
[sudo] password for usuarioSeguro:
Unable to find image 'armhf/hello-world:latest' locally
latest: Pulling from armhf/hello-world
a0691bf12e4e: Pull complete
Digest: sha256:9701edc932223a66e49dd6c894a11db8c2cf4eccd1414f1ec105a623bf16b426
Status: Downloaded newer image for armhf/hello-world:latest
WARNING: The requested image's platform (linux/arm64) does not match the detected host platform (linux/arm/v7) and no specific platform was requested

Hello from Docker on armhf!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

usuarioSeguro@RPiPapito:~ $
```

En la imagen podemos apreciar como la máquina se descarga desde el repositorio remoto, ejecuta correctamente su contenido mostrándonos información que sería útil en caso de desconocer completamente Docker.

Docker en escritorio

Docker se puede instalar en máquinas de escritorio también, para ello basta con ir a su página web y descargar los archivos binarios para la instalación. Otra forma de instalarlo más cómodo sería hacerlo por línea de comando. En Linux con llamar al gestor de paquetería e indicarle “install docker” debería ser suficiente. Personalmente he realizado la instalación utilizando el gestor de paquetes *Choco* en Windows, simplemente por curiosidad. El resto de la practica la he realizado utilizando un ordenador con Ubuntu Server y una máquina virtual con CentOS7.

En esta practica se nos propuso crear una API y que formemos un contenedor docker con ella. El segundo paso sería preparar un archivo *compose* para que se pueda automatizar el lanzamiento de la API dockerizada.

Mi primera API

Una API (Interfaz de programación de aplicaciones) es una herramienta que permite comunicaciones entre componentes software. Junto a densas capas de abstracción y documentaciones esenciales, hoy en día todo internet funciona a base de APIs.

Se nos propuso framework y lenguaje de programación libre para la realización de esta, y decidí hacerlo en NodeJS ya que he escuchado hablar mucho de el pero nunca he investigado de que trata ni para que sirve concretamente.

Con la magia de unos tutoriales en YouTube puestos a x2 de velocidad, unas guías y un poco de documentación, en poco tiempo ya estaba haciendo un esquema mental de la API que tenía pensada hacer. Decidí realizar un programa que me permita ver el precio tanto en dólares estadounidenses como en euros de las criptomonedas. La funcionalidad que deseaba era que simplemente devuelva en un formato *json* los datos solicitados, y al tener estos un formato específico, sería fácil de crear un código que utilice este API.

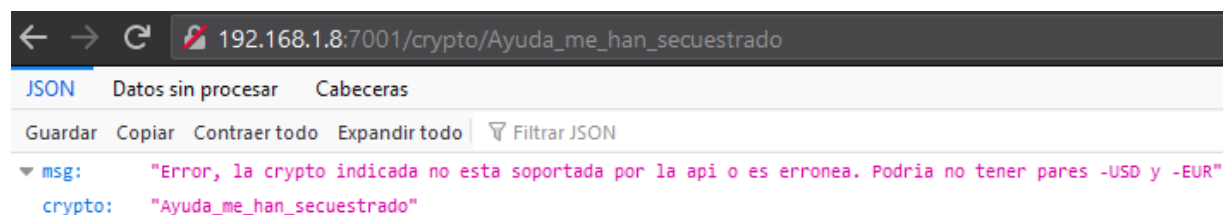


El proceso de crear la API fue bastante más difícil de lo que pensaba. Tenía claro que me adentraba en un lenguaje de programación distinto, uno que no había tocado nunca, pero lo que sí que me costó fue entender el estilo de programación que tiene. Estuve intentando durante varias horas realizar un código limpio y fácil de leer como tengo de costumbre, pero al final decidí hacer que simplemente funcione. No creo que haya entendido correctamente como funciona este lenguaje, ni como se debería utilizar, pero sí que he podido realizar una aplicación funcional con ella, a si que de momento es suficiente.

El directorio principal, /, muestra una pequeña descripción de cómo utilizar la API

```
message: "Buenas. Bienvenido a la mejor api del mundo."
usage:   ".../crypto/$var"
example: ".../crypto/BTC"
```

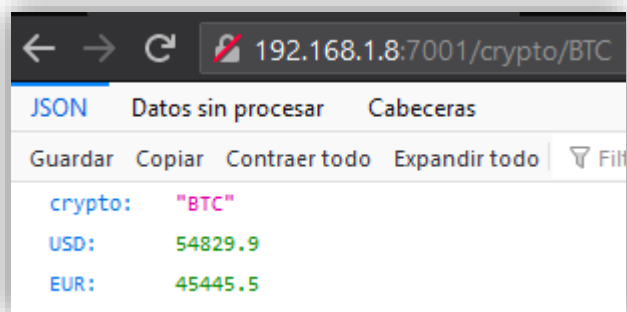
En caso de que intentes acceder a cualquier otro directorio que no este definido por el programa, este te responderá con el mismo mensaje que la página de inicio. En caso de introducir una criptomoneda no valida, o que no este soportada por la API de *blockchain* que estoy utilizando, se indicará el siguiente mensaje:



The screenshot shows a web browser at the URL `192.168.1.8:7001/crypto/Ayuda_me_han_secuestrado`. The page displays a JSON response with the following content:

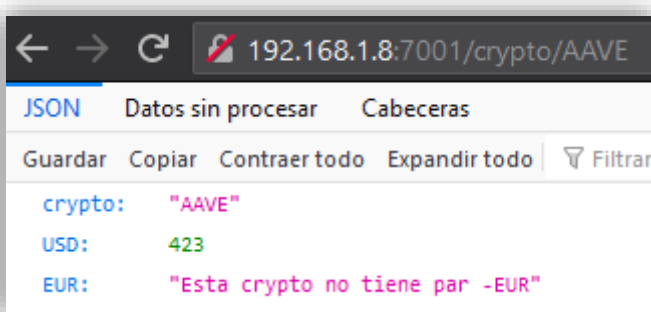
```
{
  "msg": "Error, la crypto indicada no esta soportada por la api o es erronea. Podria no tener pares -USD y -EUR",
  "crypto": "Ayuda_me_han_secuestrado"
}
```

Si la moneda indicada está soportada por la API de *blockchain*, se listará con los pares -USD y -EUR en caso de que existan.



The screenshot shows a web browser at the URL `192.168.1.8:7001/crypto/BTC`. The page displays a JSON response with the following content:

```
{
  "crypto": "BTC",
  "USD": 54829.9,
  "EUR": 45445.5
}
```



The screenshot shows a web browser at the URL `192.168.1.8:7001/crypto/AAVE`. The page displays a JSON response with the following content:

```
{
  "crypto": "AAVE",
  "USD": 423,
  "EUR": "Esta crypto no tiene par -EUR"
}
```

Dockerizar una API

Con *dockerizar* una API nos referimos a preparar un contenedor de Docker con la API dentro. Esto lo he realizado siguiendo el tutorial proporcionado por el profesor, el cual es un enlace a la documentación oficial de Docker.

El primer paso es preparar un archivo *Dockerfile* el cual tendrá las directrices que necesitará saber docker a la hora de preparar el contenedor.

```
VSSH > Node > node_test3 > Dockerfile
1  # syntax=docker/dockerfile:1
2  FROM node:12-alpine
3  RUN apk add --no-cache python g++ make
4  WORKDIR /app
5  COPY . .
6  RUN yarn install --production
7  CMD ["node", "src/server.js"]
```

En este caso, siguiendo el ejemplo de la documentación oficial, cambié el comando a realizar al iniciarse el contenedor por el nombre del archivo principal de mi código. El siguiente paso ya es construir el contenedor, así de sencillo.

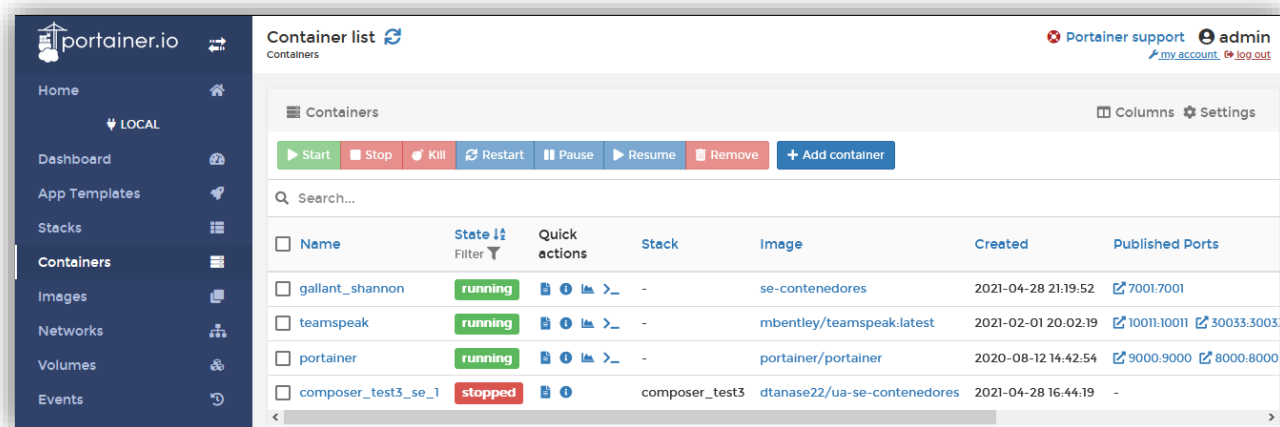
```
docker build -t ua-se-contenedores .
```

El comando indicado lo que hará es indicarle a docker que cree un contenedor con todo lo que haya en esta carpeta (el punto del final le indica que coja el directorio actual como referencia) y con el nombre de ua-se-contenedores. Una vez tenemos el contenedor creado, podemos ejecutarlo para ver si funciona.

```
docker run -d -p 7001:7001 se-contenedores
```

En este caso, el flag *-d* indica descabezado, para que se ejecute de fondo y nos quede libre la terminal. El flag *-p* hace referencia a los puertos que quiero abrir, siendo según la nomenclatura <puerto_real>:<puerto_contenedor>. Esto quiere decir que podemos encender otro contenedor indicándole *7002:7001* y este funcionará correctamente si saber que le hemos redirigido la salida a otro puerto. Para comprobar que el contenedor funcione, podemos observar Docker Desktop, o en mi caso, he instalado un contenedor que sirve como interfaz web para todo tu docker, Portainer.

Desde la vista de contenedores de Portainer se puede observar como el contenedor está encendido, y se le ha asignado el puerto 7001 real, conectado al puerto 7001 interno. La imagen muestra la configuración que tenía puesta cuando tome las capturas de pantalla del funcionamiento de la API.



Utilizando docker mismo, he subido el contenedor a mi cuenta personal, en un repositorio público. Todo el mundo puede conseguir la API indicando la imagen dtanase22/ua-se-contenedores.



Lanzamiento automático con Compose

Compose es una herramienta para manejar controlar aplicaciones que dependan de varios contenedores. Un programa puede tener una base de datos y un servidor web. Ya que utilizando Docker se puede aislar todo cómodamente, se puede preparar todo por separado en contenedores para ser juntados más adelante.

Al igual que para dockerizar necesitamos solamente un fichero, para preparar el encendido ordenado mediante Compose también nos sirve con solamente un fichero. Siguiendo la guía de la documentación oficial de Docker, apartado Compose, cree este fichero.

```
VSSH > Node > node_test3 > docker-compose.yml
1  version: "3.9"
2
3  services:
4    se:
5      image: dtanase22/ua-se-contenedores
6      ports:
7        - "7001:7001"
8      restart: always
```

La estructura es bastante sencilla. Siguiendo un espacial al estilo de código Python, cada dos espacios cambiamos de nivel. Cada servicio es un contenedor que se lanzará. En este caso le indico la imagen remota de mi contenedor, el cual esta subido a DockerHub. Le indico los puertos que deseo abrir y como ultimo un flag para que en caso de error o crash, el contenedor se reinicie.

Para ejecutar nuestro script el cual iniciará todos los contenedores que le hayamos indicado utilizaremos el siguiente comando:

```
docker-compose up
```

Una vez ejecutado el comando, se realizarán las ordenes indicadas, y la terminal se volverá la salida por consola de los contenedores indicados.

```
fxserver@fxserver_pls:/home/vscdragos/Code/VSSH/Node/composer_test3$ ls
docker-compose.yml
fxserver@fxserver_pls:/home/vscdragos/Code/VSSH/Node/composer_test3$ cat docker-compose.yml
version: "3.9"

services:
  se:
    image: dtanase22/ua-se-contenedores
    ports:
      - "7002:7001"
    restart: always

fxserver@fxserver_pls:/home/vscdragos/Code/VSSH/Node/composer_test3$ sudo docker-compose up
Starting composer_test3_se_1 ... done
Attaching to composer_test3_se_1
```