

Aplicaciones Multicontenedor en Docker

Aplicaciones Multicontenedor en Docker

Generalidades



Aplicaciones multicontenedor.pdf

54 kB

La idea original de Docker fue la de poder tener y correr aplicaciones aisladas del mundo exterior, incluso de otras aplicaciones corriendo también en docker. Para ese aislamiento, se crea lo que se llama un contenedor. Un contenedor es una aplicación compilada (y solo una) con todo lo que la soporta, incluyendo un árbol de directorios propio, aislado de otras aplicaciones y aparte del sistema operativo donde corre todo docker. En el diagrama de arriba App1, App2, y App3 son aplicaciones, cada una en un contenedor aparte. Ahora, a un contenedor se le puede asignar uno o varios directorios que donde trabaja, siendo en nuestro ejemplo /app1, /app2 y /app3. Muchas veces se necesita que lo que una aplicación trabaje se pueda ver fácilmente desde el exterior, desde el sistema operativo donde docker funciona, eso es lo que se muestra en el diagrama como volumen publicado. Estos son importantes, porque cuando se actualiza (y muchas veces cuando se reinicia) un contenedor, todos los datos dentro del mismo se pierden, porque son efímeros; sin embargo si un directorio que usa el contenedor es un volumen, este volumen es un directorio del exterior y en ese caso los datos no son efímeros. Adicionalmente a ello, cada contenedor tiene una propia dirección IP de una red "interna" de tres predeterminadas que maneja docker. Si una aplicación quisiera comunicarse con el exterior sería muy fácil, docker le permite esta salida de manera automática, pero si quisiera que una aplicación se comuniqué con otra aplicación en docker pero en diferente contenedor deberán estar en la misma red, y esto es más fácil si se tiene una red nombrada para eso (es decir que se define y se le asigna un nombre). Si desde el exterior se quisiera contactar con una aplicación que está en un contenedor no se va a poder a menos que se haya expuesto el puerto (puerto TCP o puerto UDP). Así como cuando se publica un volumen del exterior, cuyo nombre de directorio en el exterior puede ser uno mientras en el contenedor es otro, un puerto TCP (o UDP) publicado, puede tener un número en el exterior y uno en el interior.

Adicionalmente a eso, si uno quiere tener una aplicación, y la misma está compuesta por múltiples "motores" o servicios, de manera predeterminada no es fácil, porque la idea de

docker es que una aplicación está protegida, aislada y de manera predeterminada no se puede comunicar con otra aplicación que también esté en docker. Para lograr tener una aplicación de varios componentes o servicios se deben combinar, y para ello hay varias tecnologías, siendo la primera y más simple, pero menos escalable docker-compose. Esta tecnología se maneja a través de una definición declarativa (se especifica lo que se quiere, no el paso a paso para lograrlo) en unos archivos .yaml, donde se combinan:

- Las definiciones de cada contenedor como servicio
- Las definiciones de las redes nombradas
- Las definiciones de los volúmenes

Cuando se tiene una definición de esas se le dice a docker-compose que la cree y la active y la herramienta crea los contenedores, las redes, los volúmenes y las relaciones que hagan falta y las deja corriendo y funcionales. Si algo cambia docker-compose si se puede ajustar dinámicamente la ajusta en la próxima invocación, pero si dinámicamente no se puede, desactiva el componente, lo borra (lo desdefine), lo crea (lo define) y lo activa nuevamente.

Aplicación Entregable

Según todo lo anterior, en nuestro proyecto tenemos:

 Aplicaciones multicontenedor - Sema... 51 kB

A continuación describimos cómo se obtiene la funcionalidad de la aplicación multicomponente/multicontenedor

Se crea un archivo docker-compose.yaml (si se usa ese nombre la herramienta lo encuentra sin nombrarlo). Se muestra:

version: "3.7"

services:

node:

image: node-app-ci

container_name: node

hostname: node

build:

context: app

dockerfile: Dockerfile

command: node /app/index.js

ports:

- 2080:3000

networks:

- CI

links:

- mariadb

volumes:

- **type:** volume
source: app
target: /app

mariadb:

image: mariadb

container_name: mariadb

hostname: mariadb

ports:

- 4407:3306

networks:

- CI

volumes:

- **type:** volume
source: dbdata
target: /var/lib/mysql
volume:
nocopy: true
- **type:** bind
source: \$PWD/mariadb-server.cnf
target: /etc/mysql/mariadb.conf.d/50-server.cnf

environment:

MARIADB_ROOT_PASSWORD: P4ssw0rd

MARIADB_USER: mariadb

MARIABD_PASSWORD: P4ssw0rd

MARIADB_DATABASE: Mini

adminer:

image: adminer

container_name: adminer

```
hostname: adminer
restart: unless-stopped
ports:
  - 8081:8080
networks:
  - CI
links:
  - mariadb
jenkins:
  image: jenkins/jenkins
  container_name: jenkins
  hostname: jenkins
  ports:
    - 1080:8080
  networks:
    - CI
  volumes:
    - type: bind
      source: $PWD/jenkins_home
      target: /var/jenkins_home

volumes:
  dbdata:
  app:
  jenkins_home:
networks:
  CI:
    name: CI
    driver: bridge
```

Explicación

version: 3.7

Con esto se define que se necesita docker-compose que entienda al menos esta versión del archivo. Versiones anteriores no manejan todo lo que definimos aquí.

services:

Con esta sección se definen todos los servicios que nuestra aplicación necesita. Cada servicio es un contenedor aparte. Se usa como se describe:

services:

contenedor1:

definiciones del contenedor 1

contenedor2:

definiciones del contenedor 2

Así cada contenedor es aparte de los demás pero la aplicación está compuesta de todos los contenedores descritos.

volumes:

Con esta sección se declaran los volúmenes o directorios publicados, es decir los del equipo principal donde corre docker que usan cada uno de los contenedores. Se usa como se describe:

volumen1:

definición del volumen1 (si es que no se define en otra parte)

volumen2:

definición del volumen2 (si es que no se define en otra parte)

networks:

Con esta sección se definen redes si no se desea trabajar con las predeterminadas. Se usa como se describe:

networks:

red1:

name: el nombre con el que se quiere trabajar.

driver: el tipo, el más común es tipo bridge.

Cada contenedor tiene, que es cada nombre dentro de la sección **services**, tiene:

nombre: es el nombre en el archivo, a final no se reconoce ni en docker ni en el exterior o interior con este nombre.

image: nombre de un paquete de un catálogo local. Si se define que se use un paquete con la aplicación con un nombre que no está en el catálogo local docker buscará en nombre en el repositorio Docker preconfigurado, y el preconfigurado es, de manera predeterminada, hub.docker.io. Docker copiará del repositorio de internet al repositorio local si no lo encuentra y correrá usando los archivos de ese paquete.

container_name: es el nombre del contenedor para todas las herramientas, comandos e interfaces dentro de docker en la máquina principal donde se corra la aplicación.

hostname: es el nombre del sistema operativo como lo ve la aplicación si usa `gethostname()` en C (aunque sea indirectamente) o el comando `hostname` de linux.

command: es el comando con el que se inicia la aplicación dentro del contenedor. La mayoría de las imágenes ya traen uno predeterminado, pero los desarrollos propios necesitan que se especifique.

ports: son los puertos que se necesita que se expongan para recibir conexiones desde el exterior. La sintaxis es `exterior:interior`.

networks: si se da, se especifica que la dirección IP del contenedor y la conexión deben estar en la red nombrada dada. Como nuestra aplicación es multicontenedor lo natural es que todos los contenedores se conecten a la misma red.

volumes: los directorios compartidos desde el exterior como una ruta en el interior del contenedor.

links: si se da, el contenedor podrá conectarse a otro contenedor de esta aplicación usando el nombre (es decir que el nombre es resoluble con la función `gethostbyname()` de C o el comando `host` de Linux).

environment: si se da, docker crea las variables de ambiente con los valores dado para que el motor dentro del contenedor las pueda ver y usar.

Todo lo de arriba es válido cuando se usan contenedores comunes. Ahora, si un contenedor tiene código que hay que ajustar, interpretar o correr dentro, es decir que hay código modificable, lo común es usar "build", que con base en una imagen plantilla crea una nueva y la deposita en el repositorio local. Para ello dentro de la definición de un contenedor (de un servicio) se usa build. Se usa build así:

build:

dockerfile: archivo con las instrucciones para convertir una plantilla en una imagen en el repositorio local.

context: directorio temporal donde se pueden encontrar los contenidos para combinar con la plantilla.

Archivo Dockerfile.

El archivo Dockerfile se usa para crear una imagen a partir de una plantilla. Se muestra:

```
FROM bitnami/express
```

```
WORKDIR /app
COPY package.json package-lock.json
COPY package* /app/
COPY . /app/
RUN npm install
COPY . /app
ENTRYPOINT npm run start
```

Se tiene:

FROM: define de qué imagen plantilla basaremos la nueva imagen que estamos construyendo.

WORKDIR: define dónde trabajará la aplicación inicialmente cuando esté corriendo como contenedor a partir de la imagen que estamos construyendo.

COPY: copie unos archivos o directorios

RUN: antes de crear la imagen corra el comando dado, si da resultados esos resultados se incluyen en la nueva imagen

ENTRYPOINT: cuando la imagen esté creada, se usará ese comando para iniciar la aplicación.

Como nuestra imagen es basada en **Node.js NPM**, dentro de la imagen debe haber un archivo package.json que tenga una sección scripts, y dentro de la sección scripts una variable start para que npm run la encuentre y la ejecute.

Ejecución de la Aplicación

Como nuestra aplicación tiene código propio lo ideal es que se "compile" inicialmente, para eso se usa docker-compose build. Ubicados en el directorio donde están todas las definiciones de nuestra aplicación se usa docker build.

```
root@vubuntusrv:/home/e/ci/I-C_Proyecto# docker build -t node-app-ci -f app/Dockerfile app
```

```
Sending build context to Docker daemon 3.341MB
```

```
Step 1/8 : FROM bitnami/express
```

```
---> 0ee7e3e3fc17
```

```
Step 2/8 : WORKDIR /app
```

```
---> Running in e25a89d28879
```

```
Removing intermediate container e25a89d28879
```

```
---> a76d7a6d9ce8
```

```
Step 3/8 : COPY package.json package-lock.json
```

```

---> 0c3f83b71119
Step 4/8 : COPY package* /app/
---> fdd9268e1401
Step 5/8 : COPY . /app/
---> e31c230ecd02
Step 6/8 : RUN npm install
---> Running in 216d3c2959b9
npm WARN node-app@1.0.0 No repository field.
audited 57 packages in 1.207s
7 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
Removing intermediate container 216d3c2959b9
---> 61dc8cd8a98c
Step 7/8 : COPY . /app
---> dfc23dc77d8b
Step 8/8 : ENTRYPOINT npm run start
---> Running in e051d1d6653c
Removing intermediate container e051d1d6653c
---> 2dad421b3bff

```

Successfully built 2dad421b3bff

Successfully tagged node-app-ci:latest

Arriba se especifica que la imagen construida se va a identificar como node-app-ci y que va a usar un archivo app/Dockerfile y debe trabajar en el subdirectorio app del directorio donde estamos. El archivo Dockerfile le dice qué imágenes de internet y qué archivos locales va a usar. Luego de eso listamos y se encuentra:

```
oot@vubuntusrv:/home/e/ci/I-C_Proyecto# docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED
node-app-ci	latest	2dad421b3bff	About a minute ago
jenkins/jenkins	latest	51432d6486b3	2 days ago
node	16	993a4cf9c1e8	2 days ago

bitnami/express	latest	0ee7e3e3fc17	3 days ago
432MB			
mariadb	latest	1de5905a6164	7 days ago
410MB			
adminer	latest	ef63a68bb1a5	3 weeks ago
91.3MB			
node	9-slim	e20bb4abe4ee	4 years ago
182MB			

Con eso ya se puede crear la aplicación multicontenedor, usando docker-compose up. Note que crea lo que haga falta y activa lo que haga falta.

```
root@vubuntusrv:/home/e/ci/I-C_Proyecto# docker-compose up
```

```
Creating network "CI" with driver "bridge"
```

```
Creating jenkins ... done
```

```
Creating mariadb ... done
```

```
Creating adminer ... done
```

```
Creating node ... done
```

```
Attaching to jenkins, mariadb, node, adminer
```

```
jenkins | Running from: /usr/share/jenkins/jenkins.war
```

```
jenkins | webroot: /var/jenkins_home/war
```

```
jenkins | 2022-12-08 22:58:22.599+0000 [id=1] INFO
```

```
winstone.Logger#logInternal: Beginning extraction from war file
```

```
jenkins | 2022-12-08 22:58:22.721+0000 [id=1] WARNING
```

```
o.e.j.s.handler.ContextHandler#setContextPath: Empty contextPath
```

```
jenkins | 2022-12-08 22:58:22.935+0000 [id=1] INFO
```

```
org.eclipse.jetty.server.Server#doStart: jetty-10.0.12; built:
```

```
2022-09-14T01:54:40.076Z; git:
```

```
408d0139887e27a57b54ed52e2d92a36731a7e88; jvm 11.0.17+8
```

```
jenkins | 2022-12-08 22:58:24.005+0000 [id=1] INFO
```

```
o.e.j.w.StandardDescriptorProcessor#visitServlet: NO JSP Support
```

```
for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
```

```
jenkins | 2022-12-08 22:58:24.181+0000 [id=1] INFO
```

```
o.e.j.s.s.DefaultSessionIdManager#doStart: Session
```

```
workerName=node0
```

```
mariadb | 2022-12-08 22:58:21+00:00 [Note] [Entrypoint]:
```

```
Entrypoint script for MariaDB Server 1:10.10.2+maria~ubu2204  
started.
```

```
mariadb | 2022-12-08 22:58:21+00:00 [Note] [Entrypoint]:  
Switching to dedicated user 'mysql'  
mariadb | 2022-12-08 22:58:21+00:00 [Note] [Entrypoint]:  
Entrypoint script for MariaDB Server 1:10.10.2+maria~ubu2204  
started.  
mariadb | 2022-12-08 22:58:22+00:00 [Note] [Entrypoint]:  
MariaDB upgrade not required  
mariadb | 2022-12-08 22:58:22 0 [Note] mariadbd (server  
10.10.2-MariaDB-1:10.10.2+maria~ubu2204) starting as process 1 ...  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Compressed  
tables use zlib 1.2.11  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Number of  
transaction pools: 1  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Using crc32 +  
pclmulqdq instructions  
mariadb | 2022-12-08 22:58:23 0 [Note] mariadbd: O_TMPFILE is  
not supported on /tmp (disabling future attempts)  
mariadb | 2022-12-08 22:58:23 0 [Warning] mariadbd:  
io_uring_queue_init() failed with ENOMEM: try larger memory  
locked limit, ulimit -l, or  
https://mariadb.com/kb/en/systemd/#configuring-limitmemlock under  
systemd (262144 bytes required)  
mariadb | 2022-12-08 22:58:23 0 [Warning] InnoDB: liburing  
disabled: falling back to innodb_use_native_aio=OFF  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Initializing  
buffer pool, total size = 128.000MiB, chunk size = 2.000MiB  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Completed  
initialization of buffer pool  
adminer | [Thu Dec 8 22:58:25 2022] PHP 7.4.33 Development  
Server (http://\[::\]:8080) started  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: File system  
buffers for log disabled (block size=512 bytes)  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: 128 rollback  
segments are active.  
mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Setting file  
'./ibtmp1' size to 12.000MiB. Physically writing the file full;
```

Please wait ...

mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: File './ibtmp1' size is now 12.000MiB.

mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: log sequence number 46456; transaction id 14

mariadb | 2022-12-08 22:58:23 0 [Note] Plugin 'FEEDBACK' is disabled.

mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/ib_buffer_pool

mariadb | 2022-12-08 22:58:23 0 [Warning] You need to use --log-bin to make --expire-logs-days or --binlog-expire-logs-seconds work.

mariadb | 2022-12-08 22:58:23 0 [Note] Server socket created on IP: '[0.0.0.0](#)'.

mariadb | 2022-12-08 22:58:23 0 [Note] InnoDB: Buffer pool(s) load completed at 221208 22:58:23

mariadb | 2022-12-08 22:58:23 0 [Note] mariadbd: ready for connections.

mariadb | Version: '10.10.2-MariaDB-1:10.10.2+maria~ubu2204' socket: '/run/mysqld/mysqld.sock' port: 3306 [mariadb.org](#) binary distribution

node |

node | > node-app@1.0.0 start /app

node | > node ./index.js

node |

jenkins | 2022-12-08 22:58:26.459+0000 [id=1] INFO

hudson.WebAppMain#contextInitialized: Jenkins home directory: /var/jenkins_home found at:

EnvVars.masterEnvVars.get("JENKINS_HOME")

node | Server is listening on port 3000 ...

jenkins | 2022-12-08 22:58:26.909+0000 [id=1] INFO

o.e.j.s.handler.ContextHandler#doStart: Started

w.@779de014{Jenkins

v2.381,/,file:///var/jenkins_home/war/,AVAILABLE}

{/var/jenkins_home/war}

```
jenkins      | 2022-12-08 22:58:26.965+0000 [id=1] INFO
o.e.j.server.AbstractConnector#doStart: Started
ServerConnector@5b94b04d{HTTP/1.1, (http/1.1)}{0.0.0.0:8080}
jenkins      | 2022-12-08 22:58:27.033+0000 [id=1] INFO
org.eclipse.jetty.server.Server#doStart: Started
Server@757acd7b{STARTING}[10.0.12,sto=0] @5812ms
jenkins      | 2022-12-08 22:58:27.066+0000 [id=23]          INFO
winstone.Logger#logInternal: Winstone Servlet Engine running:
controlPort=disabled
jenkins      | 2022-12-08 22:58:27.829+0000 [id=29]          INFO
jenkins.InitReactorRunner$1#onAttained: Started initialization
jenkins      | 2022-12-08 22:58:28.180+0000 [id=31]          INFO
jenkins.InitReactorRunner$1#onAttained: Listed all plugins
jenkins      | 2022-12-08 22:58:36.465+0000 [id=29]          INFO
jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
jenkins      | 2022-12-08 22:58:36.555+0000 [id=31]          INFO
jenkins.InitReactorRunner$1#onAttained: Started all plugins
jenkins      | 2022-12-08 22:58:36.778+0000 [id=31]          INFO
jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
jenkins      | WARNING: An illegal reflective access operation has
occurred
jenkins      | WARNING: Illegal reflective access by
org.codehaus.groovy.vmplugin.v7.Java7$1
(file:/var/jenkins_home/war/WEB-INF/lib/groovy-all-2.4.21.jar) to
constructor
java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)
jenkins      | WARNING: Please consider reporting this to the
maintainers of org.codehaus.groovy.vmplugin.v7.Java7$1
jenkins      | WARNING: Use --illegal-access=warn to enable
warnings of further illegal reflective access operations
jenkins      | WARNING: All illegal access operations will be
denied in a future release
jenkins      | 2022-12-08 22:58:39.025+0000 [id=30]          INFO
h.p.b.g.GlobalTimeoutConfiguration#load: global timeout not set
jenkins      | 2022-12-08 22:58:40.130+0000 [id=28]          INFO
jenkins.InitReactorRunner$1#onAttained: System config loaded
```

```
jenkins | 2022-12-08 22:58:40.132+0000 [id=29] INFO
jenkins.InitReactorRunner$1#onAttained: System config adapted
jenkins | 2022-12-08 22:58:40.386+0000 [id=30] INFO
jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
jenkins | 2022-12-08 22:58:40.394+0000 [id=29] INFO
jenkins.InitReactorRunner$1#onAttained: Configuration for all
jobs updated
jenkins | 2022-12-08 22:58:40.604+0000 [id=28] INFO
jenkins.InitReactorRunner$1#onAttained: Completed initialization
jenkins | 2022-12-08 22:58:40.882+0000 [id=22] INFO
hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and
running
jenkins | 2022-12-08 22:58:41.394+0000 [id=44] WARNING
h.m.DownloadService$Downloadable#updateNow: No tool installer
metadata found for
```

[jenkins.plugins.nodejs.tools.MirrorNodeJSInstaller](#)

Como no se dio la opción -d todos los mensajes son visibles y no se puede cerrar la ventana donde se usó el comando, pero en otra ventana se puede observar:

```
e@vubuntusrv:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	PORTS
bafe3ae6c7768	adminer	entrypoint.sh docke...	About a minute ago Up About a minute 0.0.0.0:8081->8080/tcp, :::8081->8080/tcp
8a5a35cbf153	node-app-ci	"/bin/bash -o pipefa..."	About a minute ago Up About a minute 0.0.0.0:2080->3000/tcp, :::2080->3000/tcp
86f81a3ef943	jenkins/jenkins	"/usr/bin/tini -- /u..."	About a minute ago Up About a minute 50000/tcp, 0.0.0.0:1080->8080/tcp, :::1080->8080/tcp
92e38b9ff47d	mariadb	"docker-entrypoint.s..."	About a minute ago Up About a minute 0.0.0.0:4407->3306/tcp, :::4407->3306/tcp

Se tienen el contenedor de nuestra aplicación node-app-ci con nombre **node**, con el puerto externo **2080** apuntando al 3000 interno; el contenedor jenkins con nombre

jenkins y con el puerto **1080** externo apuntando al 8080 interno; y el contenedor **maradb**, con nombre **maradb** y con el puerto externo **4407** apuntando al 3306 internamente.

Si probamos una consulta a nuestra aplicación, apuntando al servidor docker y con el puerto 2080 debería contestar:

```
[0 e@Gunther:~/Documents/Other/Politécnico/Code/CI/I-C_Proyecto] curl
http://192.168.25.69:2080/customers/ 18:03:59
{"id":1,"name":"Ramón"}%
```

Con el cliente de URLs HTTP, curl, probamos y nos responde {"id":1,"name":"Ramón"}. El código de ejemplo se muestra:

```
[0 e@Gunther:~/Documents/Other/Politécnico/Code/CI/I-C_Proyecto] cat
app/index.js 18:04:47
const express = require('express');
const app = express();
const port = 3000;
app.use(express.json());
app.listen(port, ()=>{
    console.log('Server is listening on port',port,'...');
});
app.get('/customers', (req,res)=>{
    res.send(
        { id: 1, name: "Ramón", },
    );
});
```

Es un código muy simple hecho en **Node.js** con **Express.js** dentro de un contenedor.