El testeo de aplicaciones ha sido tradicionalmente tedioso ya que la ejecución de nuestra aplicación requiere de la instalación de sus dependencias, y además puede necesitar componentes externos como bases de datos. Sabemos que Docker soluciona estos problemas, y que gracias a la herramienta *docker-compose*, es prácticamente trivial el proceso de ejecución de una aplicación en local.

Pues bien, estas ideas pueden ser aplicadas al proceso de testeo de una aplicación. Lo primero que tenemos que hacer es *dockerizar* nuestros scripts de tests, de tal manera que podamos ejecutarlos haciendo uso de *docker-compose* al igual que hacemos con cualquier otra imagen de Docker. El proceso de dockerizar nuestros tests equivale a crear un fichero `Dockerfile.test` que instalará las librerías necesarias para la ejecución de nuestros tests.

Una vez dockerizados los tests, podremos crear un fichero `docker-test.yml` donde haremos referencia a la imagen que ejecuta nuestros tests, y donde añadiremos el resto de servicios que necesitemos para levantar nuestro entorno de pruebas. En un caso sencillo, supongamos que la ejecución de nuestros tests sólo necesita tener levantada una base de datos *MySQL*. Nuestro fichero `docker-test.yml` podría ser así:

```
test:
    build: .
    dockerfile: Dockerfile.test
    link:
        - db
db:
    image: mysql
```

Por tanto, la ejecución de nuestros tests puede ser automatizada de la siguiente manera:

```
docker-compose -f ~/testing/docker-test.yml -p ci build
docker-compose -f ~/testing/docker-test.yml -p ci up -d
docker logs -f ci_test_1
docker wait ci_test_1
```

La primera línea construye la imagen que corre los tests, la segunda levanta el entorno de pruebas (`-p ci` sirve para prefijar los containers creados por *docker-compose* y `-f` para hacer referencia a un fichero *yml* que no sea `docker-compose.yml` , que es el valor por defecto), la tercera muestra la salida en *streaming* de nuestros tests, y finalmente la cuarta muestra el código de salida de nuestros tests, de tal manera que `0` indica que los tests han pasado, y cualquier otro valor, que los tests fallaron.

Este enfoque para la ejecución de tests basado en Docker y Docker Compose presenta las siguientes ventajas:

- Automatizable: esta manera de testear es independiente de la aplicación que queremos testear. Esto facilita enormemente la tarea del administrador que se dedique a automatizar el testeo de aplicaciones en el proceso de integración continua.

- Ligero: Docker es ligero, por lo que el fichero *docker-test.yml* puede contener tantos servicios como sea necesario y podrá ser ejecutado en una sola máquina. Esto permite la ejecución de entornos complejos para pruebas de integración y aceptación.

- Portable: gracias a que el proceso está basado en Docker y Docker es portable, la ejecución de pruebas puede realizarse en cualquier máquina, independientemente de su sistema operativo o el hardware interno.

- Inmutable: gracias a que Docker es inmutable, los procesos de pruebas también lo son, por lo que las pruebas que pasen en tu máquina local están garantizadas a pasar en la máquina utilizada para la integración continua.

Conclusión

Docker es una herramienta que también revoluciona los procesos de integración continua, al ofrecer cuatro ficheros que actúan como contrato entre el desarrollador y el personal de administración de sistema:

- **Dockerfile**: cómo construir la imagen de nuestra aplicación.
- **docker-compose.yml**: cómo ejecutar nuestra aplicación.
- **Dockerfile.test**: cómo construir la imagen que prueba nuestra aplicación.
- **docker-test.yml**: cómo testear nuestra aplicación.

Por tanto, el personal de sistemas puede añadir el proceso de *build* y de pruebas a su herramienta de integración continua independientemente de la aplicación en cuestión, o del sistema operativo o hardware que estén usando los desarrolladores de dicha aplicación.

Existen números herramientas para automatizar el *build* y las pruebas de aplicaciones, algunas puedes instalarlas en tu infraestructura, como:

- Jenkins
- Bamboo
- Drone.io

Y otras funcionan bajo el modelo SaaS, como:

- Travis
- Circleci

Docker ofrece sus propias herramienta de integración continua:

- Dockerhub: auto *builds* cuando hay un commit en Github.
- Tutum: testing cuando hay una PR en Github.