[Docker SecDevOps] Capítulo 2 : Dockerfile

- # comentarios o directivas
- `INSTRUCCIÓN argumentos` : por convención, instrucción en mayúsculas
- primera instrucción: FROM (o ARG)

build

- `docker build` o `docker image build`
  - `-t <nombre_imagen>[:tag]`
  - `-f <nombre_fichero_Dockerfile>`

directivas

- antes de la instrucción **FROM**
- no repeticiones
- formato concreto: `# directiva=valor` (respetando espacios) → si no, es tratado como un comentario
- directivas soportadas actualmente:
  - **escape** : caracter de escape en ficheros Dockerfile. Soporta \ y `

variables de entorno

- **ENV**
- `ENV var=valor`
- `ENV var=valor var2=valor2 var3=$var2`
- se referencian con el signo **$** o **${}**
- funcionalidades tipo bash:
  - ${var:-texto} : si var tiene valor propio (está inicializada) lo devuelve, si no, devuelve *texto*
  - ${var:+texto} : si var tiene valor propio, devuelve la cadena *texto*, si no, devuelve vacío
- se pueden usar en:
  - ADD
  - COPY
  - ENV
  - EXPOSE
  - FROM
  - ONBUILD
  - LABEL
  - STOPSIGNAL
  - USER
  - VOLUME
  - WORKDIR

.dockerignore

- se procesa al mismo tiempo que se procesa el contexto en el **build** de una imagen
- ignora todos los archivos / directorios que estén especificados
- uso de comodines: *****, **?**, **!**
- comentarios: **#**
- importancia del orden de criterio de exclusión:

```
*.md
!README.md
```

```
*.md
!README*.md
README-secret.md
```

```
*.md
README-secret.md
!README*.md
```

    - el primer ejemplo excluye todos los ficheros *.MD* excepto el *README.md*
    - el segundo excluye todos los ficheros *.MD* excepto los *README*.md*, aunque el *README-secret.md* también quedaría excluido
    - el tercer ejemplo es una mala construcción por el orden de las instrucciones, ya que el fichero *README-secret.md* quedaría incluido, cuando lo que pretendemos es excluirlo
- se puede excluir *Dockerfile* también, pero solo se ignorará en las instrucciones **COPY** y **ADD**

FROM

- `FROM <imagen>[:tag|@digest] [AS <nombre>]`
    - si no se especifica *tag* se usará **latest**
    - el digest es el SHA256 de la imagen: `FROM busybox@sha256:3e8…0e7`
- primera instrucción del *Dockerfile* (con excepción de **ARG**)

multistage

uso de más de una imagen Docker para realizar la tarea

- uso de 2 o más FROM en el Dockerfile
- la imagen del último FROM es la que prevalece, todas las anteriores son descartadas
- es posible «traspasar» ficheros de un fase a otra con un parámetro en el comando **COPY**
    - `COPY –from=0 ….`
    - 0 haría referencia a la primera imagen usada, también se puede hacer referencia a través del nombre asignado en **AS**

RUN

ejecución de comandos en la imagen que estamos construyendo

- `RUN <comando>` → comando es pasado como parámetro a la shell del sistema:
    - linux: /bin/sh -c
    - windows: cmd /s /c
- `RUN [«ejecutable»,«parámetro1»,«parámetro2»]`
    - no se ejecuta shell (o para cambiar la shell o entornos sin shell)
    - vector JSON (comillas obligatorias)
- cada RUN genera una layer(capa)
- uso de | (pipe) para redirigir la salida de un ejecutable a otro
    - tener en cuenta que si falla la ejecución del primero, pero no del segundo, la ejecución se dará por buena

- se puede usar `set -o pipefail` para evitar este comportamiento (aunque no todos los shell lo soportan)
- `RUN [«/bin/bash», «-c»,«set -o pipefail && wget …»]`

CMD