

PLATAFORMAS e INFRAESTRUCTURAS
PUNTO CONTROL: CONCEPTOS GENERALES DOCKER
(2021/10/18)

Al finalizar el punto de control, cada alumno deberá subir un documento (similar al de las prácticas) con los resultados obtenidos durante los pasos ejecutados en la prueba a la plataforma Moodle.

Para ello se ha creado una tarea dentro de la asignatura. No habrá posibilidad de subir trabajos a partir de las 12.00.

1. Contenedorizando una aplicación (10 puntos)

En este ejercicio, tendrás que poner en marcha una aplicación distribuida y basada en microservicios.

Aplicación Encuesta
=====

Aplicación distribuida que corre en múltiples contenedores Docker.

Una vez terminado todo el desarrollo, ejecuta:
docker-compose up

La aplicación debería de ejecutarse en <http://localhost:5000>: para hacer la votación y los resultados deberían de aparecer en <http://localhost:5001>.

Nota

La aplicación debe de aceptar un único voto por cliente. No debe de registrar votos que hayan sido introducido por el mismo cliente.

Tareas a realizar:

- Voting-app: escribir el Dockerfile para esta aplicación. **(2 puntos)**.
 - La imagen base será Python (tal y como se indica en el propio Dockerfile) y el directorio de trabajo /app.
[En la instrucción FROM utilizamos Python:latest y en la Instrucción WORKDIR utilizamos el directorio /app.](#)
 - Para que la instalación sea adecuada y tenga en cuenta los requisitos, hay que copiar el fichero de requisitos, en el directorio de trabajo.

Añadimos la línea `ADD requirements.txt` .

- Una vez instalado todas las librerías indicadas en los requisitos, se copiarán todos los ficheros de la aplicación, en el directorio de trabajo

`COPY . .`

- Exponer el puerto 80

`EXPOSE 80`

- El comando a ejecutar ya está escrito en el Dockerfile

```
0 Using official python runtime base image
1 FROM python:latest
2
3 # Set the application directory
4 WORKDIR /app
5
6 # Install our requirements.txt
7 ADD requirements.txt .
8 RUN pip install -r requirements.txt
9
10 # Copy our code from the current folder to /app inside the container
11 COPY . .
12
13 # Make port 80 available for links and/or publish
14 EXPOSE 80
15
16 # Define our command to be run when launching the container
17 CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-", "--access-logfile", "-", "--workers", "4", "--keep-alive", "0"]
```

```

markel in Azterketa_2021_Docker/vote/ > docker build -t vote .
+ ] Building 45.1s (11/11) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 658B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                    0.0s
=> [internal] load metadata for docker.io/library/python:latest                 2.3s
=> [auth] library/python:pull token for registry-1.docker.io                   0.0s
=> [1/5] FROM docker.io/library/python:latest@sha256:d0075399d5663b0d8619d0260dddccee7b2ee8e6c6 33.4s
=> => resolve docker.io/library/python:latest@sha256:d0075399d5663b0d8619d0260dddccee7b2ee8e6c66 0.0s
=> => sha256:d0075399d5663b0d8619d0260dddccee7b2ee8e6c66be441ba088607d3cc7e05 2.60kB / 2.60kB 0.0s
=> => sha256:c05c608cfa20e03bee68abe9d939be74b8b249bfc42f339650503a0a3fc05bb3 8.60kB / 8.60kB 0.0s
=> => sha256:f02b617c6a8c415a175f44d7e2c5d3b521059f2a6112c5f022e005a44a759f2d 5.15MB / 5.15MB 4.1s
=> => sha256:d32e17419b7ee61bbd89c2f0d2833a99cf45e594257d15cb567e4cf7771ce34a 10.87MB / 10.87MB 4.5s
=> => sha256:18b5a822a1ea3031be2ef990f017aaf7ed3330dfe776fe4402bd7f979ec91fc2 2.22kB / 2.22kB 0.0s
=> => sha256:bb7d5a84853b217ac05783963f12b034243070c1c9c8d2e60ada47444f3cce04 54.92MB / 54.92MB 4.4s
=> => sha256:c9d2d81226a43a97871acd5afb7e8aabfad4d6b62ae1709c870df3ee230bc3f5 54.57MB / 54.57MB 13.9s
=> => sha256:3c24ae8b66041c09dabc913b6f16fb914d5640b53b10747a343ddc5bb5bd67 196.50MB / 196.50MB 24.2s
=> => extracting sha256:bb7d5a84853b217ac05783963f12b034243070c1c9c8d2e60ada47444f3cce04 2.2s
=> => sha256:8a4322d1621dec2def676639847e546c07a1da4ba0c035edb1036bf5fb45063b 6.29MB / 6.29MB 5.3s
=> => sha256:2412fd0e5000b1a3f7d7431dc76261720335d72004eb74651bba6ac6bd351c59 18.70MB / 18.70MB 10.9s
=> => extracting sha256:f02b617c6a8c415a175f44d7e2c5d3b521059f2a6112c5f022e005a44a759f2d 0.2s
=> => extracting sha256:d32e17419b7ee61bbd89c2f0d2833a99cf45e594257d15cb567e4cf7771ce34a 0.3s
=> => sha256:0c700e8025dba3d52387fd8641d26b72e613899a110c0ad47912db24e35599ce 233B / 233B 11.6s
=> => sha256:6e1cb03c0153a976f04715aca3d54dcd7cc898276ad5a7f31ac2ee03a995e0a7 2.35MB / 2.35MB 12.5s
=> => extracting sha256:c9d2d81226a43a97871acd5afb7e8aabfad4d6b62ae1709c870df3ee230bc3f5 2.4s
=> => extracting sha256:3c24ae8b66041c09dabc913b6f16fb914d5640b53b10747a343ddc5bb5bd6769 7.1s
=> => extracting sha256:8a4322d1621dec2def676639847e546c07a1da4ba0c035edb1036bf5fb45063b 0.3s
=> => extracting sha256:2412fd0e5000b1a3f7d7431dc76261720335d72004eb74651bba6ac6bd351c59 0.7s
=> => extracting sha256:0c700e8025dba3d52387fd8641d26b72e613899a110c0ad47912db24e35599ce 0.0s
=> => extracting sha256:6e1cb03c0153a976f04715aca3d54dcd7cc898276ad5a7f31ac2ee03a995e0a7 0.2s
=> [internal] load build context                                                  0.0s
=> => transferring context: 10.36kB                                              0.0s
=> [2/5] WORKDIR /app                                                            0.1s
=> [3/5] ADD requirements.txt .                                                  0.0s
=> [4/5] RUN pip install -r requirements.txt                                    9.1s
=> [5/5] COPY . .                                                                0.0s
=> exporting to image                                                            0.2s
=> => exporting layers                                                            0.2s
=> => writing image sha256:f022fc52e6555c8eb0422d543da34b1dc4ccb67b0cb6331e83b99ab63534d177 0.0s
=> => naming to docker.io/library/vote                                          0.0s

```

se 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

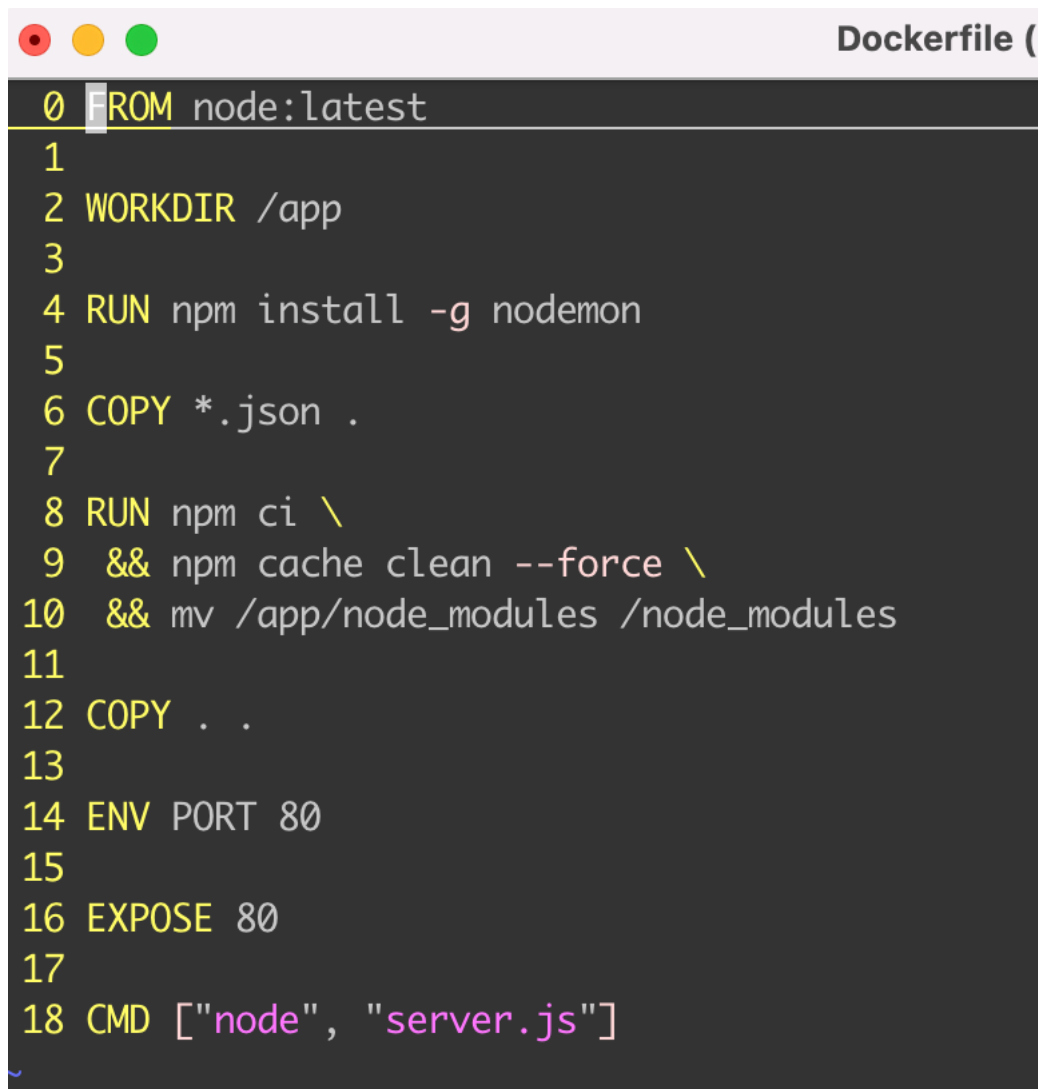
- Worker: escribir el Dockerfile para esta aplicación. **(2 puntos)**
 - La imagen base es de maven (ya incluida en el Dockerfile), y el directorio de trabajo está situado en /code.
 - Copiar el fichero pom.xml en el directorio de trabajo.
 - Ejecuta los comandos ya escritos en RUN (ya incluidos en el Dockerfile) y añade el contenido/programa/desarrollo que se encuentra en /src/main en la misma estructura pero dentro del directorio de trabajo dentro del contenedor.
 - Ejecuta el comando RUN (ya incluido en el Dockerfile) que generará el .jar.
 - La segunda etapa del Dockerfile se basa en la imagen de java (ya incluido en el Dockerfile), y copiar el jar generado en el stage anterior en el directorio indicado en el Dockerfile (ya incluido en el Dockerfile)
 - ¿Para qué hacemos esta segunda etapa? ¿Qué pasa con la primera etapa?

La segunda etapa la utilizamos para que dentro de la imagen no se copien todos los archivos creados al compilar el jar. La primera etapa instalamos todas las dependencia y todo lo necesario para crear el archivo .jar. Al ser el .jar un archivo que puede funcionar de manera independiente, todos los paquetes que hemos instalado no nos hacen falta. Para eso, se crea un segundo stage y se copia el .jar dentro.

- El comando a ejecutar ya está escrito en el Dockerfile

```
0 FROM maven:3.5-jdk-8-alpine AS build
1
2 WORKDIR /code
3 #Copy pom.xml to the working directory
4 COPY pom.xml .
5 RUN ["mvn", "dependency:resolve"]
6 RUN ["mvn", "verify"]
7
8 # Adding source, compile and package into a fat jar
9 ADD /src/main ./src/main
10 RUN ["mvn", "package"]
11
12 FROM openjdk:8-jre-alpine
13
14 # Copy from the previous stage the generated jar
15 COPY --from=build /code/target/worker-jar-with-dependencies.jar /
16
17 CMD ["java", "-XX:+UnlockExperimentalVMOptions", "-XX:+UseCGroupMemoryLimitForHeap", "-jar", "/worker-jar-with-dependencies.jar"]
~
~
~
```

- Result: escribir el Dockerfile para esta aplicación. (2 puntos)
 - La imagen base es node
 - Configura el directorio de trabajo en /app.
 - Ejecuta el comando RUN para instalar todo lo necesario para la aplicación node (ya incluido en el Dockerfile)
 - Copiar todos los ficheros .json que están en el directorio Result en el directorio de trabajo.
 - Ejecuta los comando RUN para generar la aplicación node (incluido en el Dockerfile).
 - Copiar todo lo generado en el fichero que tenemos en el directorio result del host en el directorio de trabajo.
 - Declarar como variable de entorno PORT con el valor 80 y exponer el puerto 80.
 - Ejecutar la aplicación server.js con el comando node (incluido en el dockerfile)

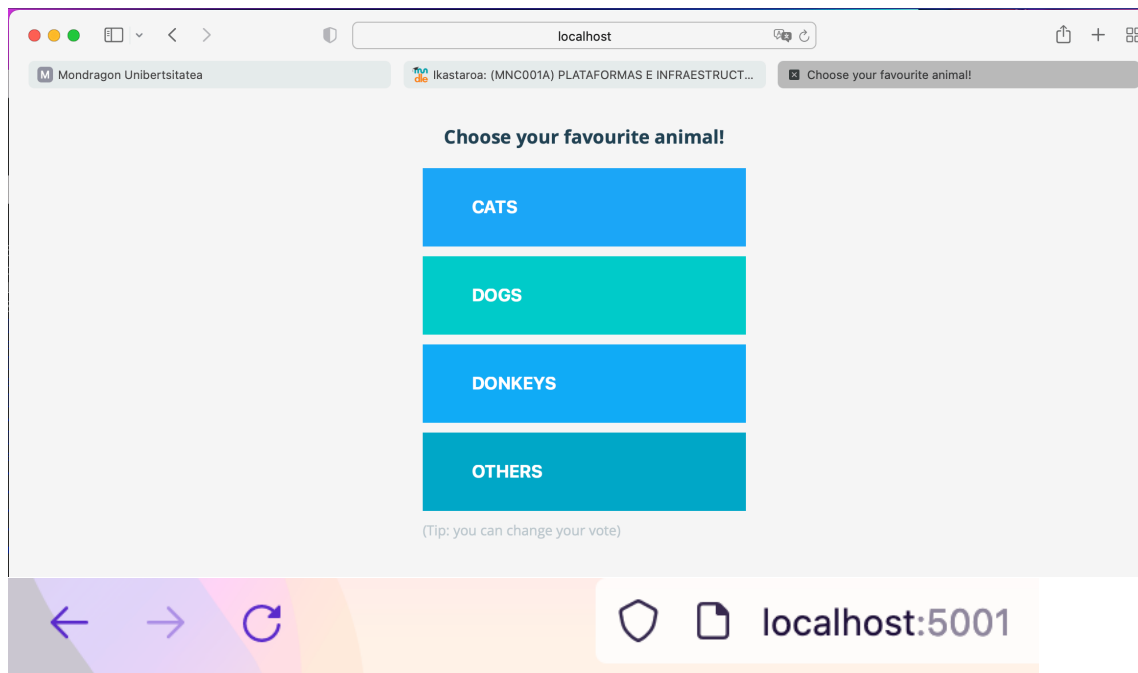


```
0 FROM node:latest
1
2 WORKDIR /app
3
4 RUN npm install -g nodemon
5
6 COPY *.json .
7
8 RUN npm ci \
9   && npm cache clean --force \
10  && mv /app/node_modules /node_modules
11
12 COPY . .
13
14 ENV PORT 80
15
16 EXPOSE 80
17
18 CMD ["node", "server.js"]
```

- Docker Compose: escribir el fichero docker-compose (simple) para lanzar toda la aplicación mediante el comando docker-compose up.(1.5 puntos)

```
0 version: "3"
1
2 services:
3   vote:
4     build: ./vote
5     command: python app.py
6     volumes:
7       - ./vote:/app
8     ports:
9       - "5000:80"
10
11  redis:
12    image: redis:alpine
13    container_name: redis
14    ports: ["6379"]
15
16  worker:
17    build: ./worker
18
19  db:
20    image: postgres:9.4
21    container_name: db
22    environment:
23      POSTGRES_USER: "postgres"
24      POSTGRES_PASSWORD: "postgres"
25
26  result:
27    build: ./result
28    command: nodemon server.js
29    volumes:
30      - ./result:/app
31    ports:
32      - "5001:80"
33      - "5858:5858"
```

En Mudle, encontrarás toda la infraestructura necesaria para poner en marcha la práctica



>

Cats

25.0%

Dogs

25.0%

Donkeys

25.0%

Others

25.0%

Arquitectura (2.5 puntos)

Analiza la infraestructura y código que has desarrollado y dibuja la arquitectura de la aplicación general.

¿Cuántos contenedores dispone la aplicación distribuida?

La aplicación contiene 5 contenedores. Son los siguientes:

1. Contenedor de la aplicación de voto
2. Redis
3. Worker
4. Base de datos postgres
5. Aplicación de resultados

¿Cuál crees que es el objetivo de cada contenedor? Explica la funcionalidad de cada contenedor/servicio.

1. Contenedor de la aplicación de voto: Sirve como una interfaz via web para votar.
2. Redis: Un sistema de colas para los votos
3. Worker: Se encarga de procesar los votos.
4. Base de datos postgres: Almacenar los votos
5. Aplicación de resultados: Sirve para mostrar los resultados de la votación.

¿Están los contenedores conectados entre sí? ¿Cómo?

Si, al levantar la aplicación distribuida con docker-compose se crea una red a la que se conectan los cinco contenedores.

```
✓ markel in exam_docker/Azterketa_2021_Docker/ > docker network ls
NETWORK ID          NAME                                DRIVER  SCOPE
f101c2ed6de8        azterketa_2021_docker_default      bridge  local
d8f18aa20ba3        bridge                             bridge  local
c44eabc6d42e        host                               host    local
5ad364c73089        none                               null    local
✓ markel in exam_docker/Azterketa_2021_Docker/ >
```

Si lanzamos un docker inspect de esa red vemos como los contenedores están conectados:


```

    "Containers": {
      "4a32be7b4f7480aaa0914d5c67240f680d86f5c2fcb8a5901294a67efec32b9d": {
        "Name": "redis",
        "EndpointID": "3542ef5b3a0883bacbccbda3bd84f990a052170c6d735807386b91d3ae5f9841",
        "MacAddress": "02:42:ac:14:00:04",
        "IPv4Address": "172.20.0.4/16",
        "IPv6Address": ""
      },
      "4d2c286c20ad7c2c7033bded19eab47173051a59068ec3a340c72758fdc77f3b": {
        "Name": "azterketa_2021_docker_vote_1",
        "EndpointID": "5b73201f8ffee0ced9e5558460757d18b7b3af83630c2ebdf9038099c0d614e2",
        "MacAddress": "02:42:ac:14:00:06",
        "IPv4Address": "172.20.0.6/16",
        "IPv6Address": ""
      },
      "6aaba05f8ffb2d69a8e0a24a2a0962738468d7b696eb07033a4c0199326de227": {
        "Name": "db",
        "EndpointID": "5b770be5cd8721ecc75bfe7284037a8f51dae23d7ff7e773e93ee90e50105270",
        "MacAddress": "02:42:ac:14:00:03",
        "IPv4Address": "172.20.0.3/16",
        "IPv6Address": ""
      },
      "aabc0a7b8649663b539639c8a40923e4c9e9e15b9dcb03d0589ce25b1b06bbe7": {
        "Name": "azterketa_2021_docker_result_1",
        "EndpointID": "5bd3fc19ec3e7bd2cf807d9f09757bee3dcd86377016e9288c955cd00623fe4a",
        "MacAddress": "02:42:ac:14:00:05",
        "IPv4Address": "172.20.0.5/16",
        "IPv6Address": ""
      },
      "ffd8e07a6b9764d3aa4f01c4988a5f7803de242f48630a7e80b1003b2d1c74e0": {
        "Name": "azterketa_2021_docker_worker_1",
        "EndpointID": "bd74f7a8aa3cce4189cc381b3cbffff9bc898bd18786e402174d2716fe9d2f66",
        "MacAddress": "02:42:ac:14:00:02",
        "IPv4Address": "172.20.0.2/16",
        "IPv6Address": ""
      }
    }
  },
  "Containers": {

```

¿Existe algún sistema donde se guardan los datos de forma persistente?

Tenemos una base de datos, aun asi los datos que se guardan en ella no se guardan de manera persistente. Al no utilizar volúmenes, si el contenedor de la base de datos se borra perderemos los datos.

A continuación, abre la versión “complex” del docker-compose y compara con la versión con la que has estado trabajando hasta ahora.

¿Cuáles son las diferencias principales? ¿Tiene alguna ventaja la nueva versión? ¿Por qué?

Las diferencias principales son que en esta nueva versión de docker compose se utilizan redes, volúmenes y dependencias.

Para el caso de las redes, vemos que cada uno de los contenedores solo tiene conectividad con los que necesita y no están todos los contenedores conectados a la misma red. De esta manera podemos hacer una separación entre el front-end y el back-end.

Para los volúmenes, vemos que la base de datos se configura con un volumen, de tal manera que si el contenedor de la base de datos se cae la información persiste en el host de docker y no perderíamos la información sobre las votaciones.

En el caso de las dependencias, vemos que algunos servicios necesitan que otros estén levantados para funcionar, de tal manera que si los contenedores de los que dependen otros contenedores no están levantados no se levantarán los contenedores. Así evitamos errores en al iniciar las aplicaciones.

FEEDBACK

¿Qué te ha parecido la primera parte de la asignatura? ¿Crees que ha sido útil para tu formación?

Prácticas útiles para entender el funcionamiento de docker.

¿Qué te ha parecido el ritmo de impartición? ¿Y el nivel de dificultad?

Nivel de dificultad fácil y ritmo alto.

¿Cambiarías o quitarías algún tema? ¿Plantearías la asignatura de otra forma?

No, lo dejaría como esta.

¿Alguna propuesta de mejora? No.