

SISTEMAS DISTRIBUIDOS

Práctica no guiada: Sockets, Streaming de Eventos, Colas y
modularidad.

Art With Drones

Joan Cerveto Serrano — 50387157E

5 de noviembre de 2023

Universitat d'Alacant
Escuela Politécnica Superior

Contents

1	Introducción.	3
2	Tecnologías.	4
2.1	Despligue de aplicación.	4
2.2	Lenguajes de programación.	4
2.3	Persistencia.	6
2.4	Automatización.	7
2.5	Control de versiones.	9
3	Ejecución.	10
3.1	AD_Kafka	10
3.2	AD_Weather	12
3.3	AD_Registry	14
3.4	AD_Database	15
3.5	AD_Frontend	16
3.6	AD_Engine	19
3.7	AD_Drone	23
4	Conclusiones	29

1 Introducción.

El objetivo de la práctica a desarrollar es un sistema distribuido que implemente una simulación de una solución para la creación de figuras mediante dispositivos autónomos (drones) manejados en tiempo real.

Se podrán lanzar drones que formen figuras específicas. Cada dron se podrá desplegar en máquinas diferentes dentro de una misma red local.

El uso de *docker*, *Apache Kafka* y *sockets* ha sido crucial para el desarrollo de esta práctica.

Se requiere, como mínimo, la implementación de los siguientes módulos:

- *AD_Registry*.
- *AD_Engine*.
- *AD_Weather*.
- *AD_Drone*.

Además de estos cuatro módulos, en la presente práctica se ha desarrollado dos servicios más:

- *AD_Database* para desplegar y manejar adecuadamente la base de datos.
- *AD_Frontend* para visualizar de una manera más amigable el movimiento de los drones.

En el proyecto adjuntado con la memoria, además de estos módulos se ha adjuntado una carpeta *docs/* para almacenar toda información relevante del proyecto. También otras *enunciado/* y *pruebas/*.

La práctica ha sido desarrollada con el control de versión Git y GitHub. En el proyecto se podrá consultar el historial de *commits*.

Querría aclarar que, como *TypeScript* se transpila a *JavaScript*, a lo largo de esta memoria se puede utilizar indistintamente las palabras *TypeScript* como *JavaScript* para referirnos al mismo código. Incluso también *Node*.

A la hora de levantar cada *docker-compose* se ha utilizado el parámetro *-build* para forzar a que se reconstruya la imagen, por si hubiera cambios en el código.

2 Tecnologías.

Durante el desarrollo de esta práctica se han utilizado diferentes tecnologías para conseguir el mejor desempeño posible.

2.1 Despliegue de aplicación.

Para facilitar el despliegue de los diferentes servicios en varias máquinas para tener un sistema realmente distribuido se ha hecho uso de *docker*. Se ha utilizado tanto imágenes y contenedores creados por separado como *docker-compose* para automatizar estos procesos. Se han expuesto los puertos y se han montado diferentes volúmenes para conseguir una persistencia adecuada en todas las aplicaciones.

La práctica tendrá el siguiente esquema de puertos:

Servicio	Puerto	Observación
AD_Engine	8888	Servidor <i>HTTP</i>
AD_Engine	8080	Servidor de <i>Sockets</i>
AD_Drone	Sin puerto	Actúa como cliente.
AD_Database	Sin puerto	Se usa una BBDD <i>sqlite</i> . Funciona con archivos, no puertos.
AD_Registry	6000	Servidor de <i>Sockets</i>
AD_Weather	5000	Servidor de <i>Sockets</i>
AD_Frontend	3000	Servidor de <i>HTTP</i>
Zookeeper	2181	-
Kafka	29092	-

Los mensajes enviados entre servicios, tendrán el siguiente formato:

2.2 Lenguajes de programación.

A lo largo de esta práctica se ha utilizado *TypeScript*, *Python* y *JavaScript*. Además de algunos scripts en *Bash*.

Más del 45% está escrito en *TypeScript*. Únicamente se ha utilizado este lenguaje para el desarrollo de *AD_Engine*. Para elegir qué tecnología utilizar para este servicio, lo primero que me planteé fue buscar una que tuviera una gran facilidad en la gestión de hilos concurrentes. El entorno de ejecución de *Node* permite esto sin ningún tipo de dudas. Una vez decidido que quería ejecutar el servidor en código de *JavaScript* tendría que decidir entre hacerlo con código nativo de *JavaScript* o si quería transpilar mi código de *TypeScript* a *JavaScript*. Como quería tener una aplicación bien modularizada y escalable, elegí hacerlo con *TypeScript*.

El 33% del código de está escrito en *Python*. Elegí este lenguaje por la cantidad de documentación que tiene a lo largo de todo Internet. También, sin duda,

por la facilidad de escritura y ejecución del mismo.

Para hacer el servicio *AD_Frontend* se ha utilizado el *framework* *React.js*. Además de *HTML* y *CSS*.

El resto del código del proyecto es *JavaScript* o *Shell*. Se han utilizado para tareas menores, de pruebas o de automatización de ejecución de procesos.

Para *Node* se ha utilizado la imagen: **node:18-alpine**. Para *python* se ha utilizado la imagen: **3.9-alpine**.

2.3 Persistencia.

En cuanto a la persistencia, se ha utilizado una base de datos *sqlite* para conectar los drones registrados entre *AD_Registry* y *AD_Engine*.

Dentro de los cuatro servicios también se han utilizado archivos para mantener una lógica de persistencia entre estos.

En *AD_Drone* se ha utilizado un archivo csv para almacenar los @id, @alias y @token de los drones registrados en *AD_Registry*.

En *AD_Weather* se ha guardado un archivo csv para que el servidor pueda leer la temperatura de las ciudades.

En *AD_Engine* se ha creado una tabla para almacenar la información relacionada con la figura actual presentada, junto a los drones que están actualmente creándola.

La base de datos se compone de dos tablas. Donde una de ellas tiene una clave ajena apuntando a la otra. La estructura de las tablas es la siguiente:

nombre	tipo	constraints	descripción
pk_registry_id	INTEGER	primary key	Identificador único de la tabla.
alias	TEXT	not null	Nombre del usuario.
token	TEXT	not null	Token de autenticación.

nombre	tipo	constraints	descripción
pk_fk_map_registry_id	INTEGER	foreign key references Registry	Identificador único de la tabla.
uk_map_figura	INTEGER	unique	ID en la figura del mapa.
row	INTEGER	not null	Fila objetivo del dron.
column	INTEGER	not null	Columna objetivo del dron.

Todos los archivos utilizados, tienen disponibles con variables de entorno sus rutas dentro de volúmenes montados en *docker* o como rutas como archivos locales físicos, la manera habitual.

2.4 Automatización.

El uso *docker-compose* ha supuesto una mejora enorme en la automatización. Sin embargo, no se entenedría utilizar *docker-compose* sin variables de entorno. Las variables de entorno son ficheros *text.env*. De estos ficheros hay que extraer su contenido usando las librerías correspondientes de cada lenguaje de programación. En python *dotenv-python* o en Node *dotenv*.

Un ejemplo de la obtención de estos datos es la siguiente: En un archivo *.env*:

```
# SECURITY
ENCODING=utf-8
MAX.CONTENTLENGTH=1024

# KAFKA
KAFKA.HOST=192.168.0.235
KAFKA.PORT=29092
```

En un archivo *python*:

```
import dotenv
import os

dotenv.load_dotenv()

def getBrokerHost() -> str:
    return os.getenv('KAFKA_HOST')

def getBrokerPort() -> str:
    return os.getenv('KAFKA_PORT')
```

Además, para levantar varios drones se han creado archivos Bash para poder crear muchos drones concurrentemente.

A continuación se muestra un script.

```
#!/bin/bash
```

```
# PARAMETROS:
```

```
NUM_INSTANCES=$1
```

```
FIRST_ID=$2
```

```
LAST_ID=$(( $FIRST_ID + $NUM_INSTANCES - 1 ))
```

```
# Construir la imagen de docker
```

```
docker-compose build ad_drone
```

```
echo "Ejecutando-$NUM_INSTANCES-instancias-de-engine-desde-ID=$FIRST_ID-hasta-ID
```

```
# Bucle para crear y ejecutar las instancias en paralelo
```

```
for i in $(seq $FIRST_ID $LAST_ID); do
```

```
echo "id:-$i"
```

```
CONTAINER_NAME="ad_drone__registry_id_$i"
```

```
docker-compose run --name $CONTAINER_NAME --rm ad_drone python app/src/runnerReg
```

```
&
```

```
if [ $? -ne 0 ]; then
```

```
echo "Error-en-la-ejecucion-de-la-instancia-$i.-Parando-el-script..."
```

```
exit 1
```

```
fi
```

```
done
```

```
echo "Esperando-a-que-terminen-las-instancias..."
```

```
exit 0
```

Se ejecuta:

```
./run-engine-instances.sh <NUM_INSTANCES> <FIRST_DRONE_ID>
```

```
./run-engine-instances.sh 8 400
```

En este último caso se crearán 8 drones desde el id 400 al id 407.
Este script solo es válido en ordenadores Unix, pero no Windows.

2.5 Control de versiones.

Para mantener, desarrollar y publicar el código fuente de este proyecto, se ha utilizado el sistema de control de versiones *Git* junto con *Github*. El repositorio está publicado, aunque de manera privada, en el siguiente enlace ”https://github.com/jcerveto/art_with_drones”.

3 Ejecución.

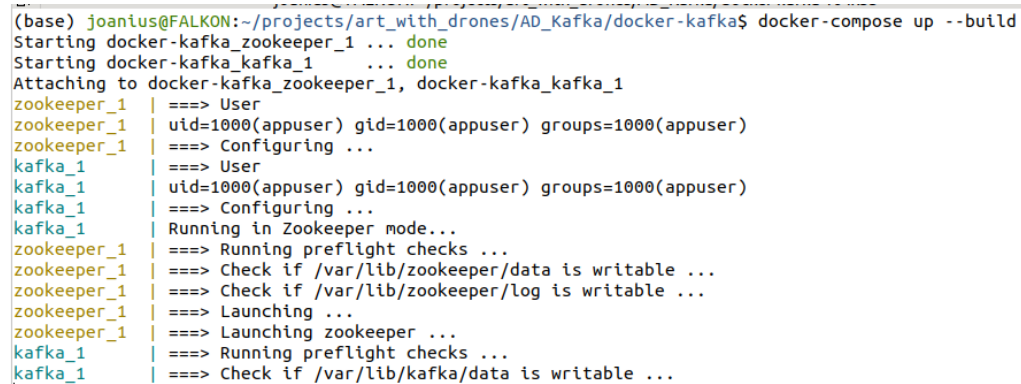
A continuación se presenta cómo se levantan todos los servicios utilizados, junto a capturas que demuestran que cada módulo funciona.

Se recomienda la ejecución de toda la aplicación en el orden que se presenta a continuación, aunque como es un sistema distribuido, funcionará igualmente sin importar el orden de ejecución.

3.1 AD_Kafka

Desde la carpeta `/docker-kafka/AD_Kafka` se ha utilizado un *docker-compose* para levantar *Apache Kafa*.

```
docker-compose up --build
```



```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Kafka/docker-kafka$ docker-compose up --build
Starting docker-kafka_zookeeper_1 ... done
Starting docker-kafka_kafka_1 ... done
Attaching to docker-kafka_zookeeper_1, docker-kafka_kafka_1
zookeeper_1 | ==> User
zookeeper_1 | uid=1000(appuser) gid=1000(appuser) groups=1000(appuser)
zookeeper_1 | ==> Configuring ...
kafka_1 | ==> User
kafka_1 | uid=1000(appuser) gid=1000(appuser) groups=1000(appuser)
kafka_1 | ==> Configuring ...
kafka_1 | Running in Zookeeper mode...
zookeeper_1 | ==> Running preflight checks ...
zookeeper_1 | ==> Check if /var/lib/zookeeper/data is writable ...
zookeeper_1 | ==> Check if /var/lib/zookeeper/log is writable ...
zookeeper_1 | ==> Launching ...
zookeeper_1 | ==> Launching zookeeper ...
kafka_1 | ==> Running preflight checks ...
kafka_1 | ==> Check if /var/lib/kafka/data is writable ...
```

Figure 1: Levantando Zookeeper y Kafka

Se paran el *Kafka* y el *Zookeeper* o bien. Parando individualmente sus contenedores de *docker* o bien ejecutando "Ctrl + C" desde su terminal, la opción más sencilla.

```
kafka_1 | [2023-11-05 20:16:50,005] INFO [GroupCoordinator 1]: Group current_position_ad_engine_cur
rent_position_1699215373518 with generation 2 is now empty (__consumer_offsets-5) (kafka.coordinator.gro
up.GroupCoordinator)
^CGracefully stopping... (press Ctrl+C again to force)
Stopping docker-kafka_kafka_1 ... done
Stopping docker-kafka_zookeeper_1 ... done
(base) joanius@FALKON:~/projects/art_with_drones/AD_Kafka/docker-kafka$ |
```

Figure 2: Apagando *Kafka* y *Zookeeper*.

3.2 AD_Weather

Desde la carpeta `/AD_Weather` se ha utilizado un *docker-compose* para levantar *AD_Weather*.

`docker-compose up --build`

```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Weather$ docker-compose up --build
Building ad_weather
[+] Building 0.3s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 253B                             0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load metadata for docker.io/library/python:3.9-alpine 0.0s
=> [1/6] FROM docker.io/library/python:3.9-alpine              0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 6.01kB                               0.0s
=> CACHED [2/6] WORKDIR /app                                    0.0s
=> CACHED [3/6] COPY requirements.txt .                          0.0s
=> CACHED [4/6] RUN pip install -r requirements.txt             0.0s
=> CACHED [5/6] COPY .env app/.env                             0.0s
=> CACHED [6/6] COPY src app/src                               0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:4da64d94b8bf5e4388be36acf6979bce98ceb4b4e41e84afa18708178bcd4fda 0.0s
=> => naming to docker.io/library/ad_weather                   0.0s
Starting ad_weather_container ... done
Attaching to ad_weather_container
ad_weather_container | Hola mundo
ad_weather_container | Server listening on 0.0.0.0:5000. Get pid: 1
```

Figure 3: Levantando AD_Weather

Tras una petición desde *AD_Engine* se muestre en el servidor:

```
ad_weather_container | Sent: b'{"city": "alacant", "temperature": 20}'
ad_weather_container | Closing connection...
ad_weather_container | -----
ad_weather_container | *****
ad_weather_container | Connection from ('172.31.0.1', 39190) has been established!
ad_weather_container | *****
ad_weather_container | Executing handle_request. Currents requests: 1 in a new thread.
ad_weather_container | Received raw data: b'alacant'
ad_weather_container | Processing request...
ad_weather_container | Clean request: alacant
ad_weather_container | City: alacant
ad_weather_container | Getting temperature for alacant...
ad_weather_container | Sent: b'{"city": "alacant", "temperature": 20}'
ad_weather_container | Closing connection...
ad_weather_container | -----
ad_weather_container |
```

Figure 4: Petición de sockets desde AD_Engine

3.3 AD_Registry

Desde la carpeta `/AD_Registry` se ha utilizado un *docker-compose* para levantar *AD_Registry*.

```
docker-compose up --build
```

```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Registry$ docker-compose up --build
Building ad_registry
[+] Building 0.2s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 311B                                0.0s
=> [internal] load .dockerignore                                  0.1s
=> => transferring context: 57B                                     0.0s
=> [internal] load metadata for docker.io/library/python:3.9-alpine 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 992B                                    0.0s
=> [1/7] FROM docker.io/library/python:3.9-alpine                0.0s
=> CACHED [2/7] WORKDIR /app                                       0.0s
=> CACHED [3/7] COPY requirements.txt .                            0.0s
=> CACHED [4/7] RUN pip install -r requirements.txt                0.0s
=> CACHED [5/7] COPY .env app/.env                                0.0s
=> CACHED [6/7] COPY main.py app/main.py                          0.0s
=> CACHED [7/7] COPY src/ app/src/                                0.0s
=> exporting to image                                              0.0s
=> => exporting layers                                             0.0s
=> => writing image sha256:a68009ccaf5c6239905192390d7165fd6915cfca57d35f3c22d0b301cf311bad 0.0s
=> => naming to docker.io/library/ad_registry_ad_registry         0.0s
Starting ad_registry_container ... done
Attaching to ad_registry_container
ad_registry_container | Files and directories in the current directory:
ad_registry_container | .env
ad_registry_container | src
ad_registry_container | main.py
ad_registry_container | Hello World!
ad_registry_container | Server listening on 0.0.0.0:6000. Get pid: 1
```

Figure 5: Levantando AD_Registry

3.4 AD_Database

Desde la carpeta `/AD_Database` se ejecuta:

```
npm install
```

Se descargan todas las dependencias.

```
node src/display-data.js
```

Para ver el contenido de la base de datos.

```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Database$ tree -I node_modules/
.
├── database.db
├── package.json
├── package-lock.json
├── README.md
└── src
    ├── app.js
    ├── create-db.js
    ├── delete-tables.js
    ├── display-data.js
    └── insert-fake-data.js

1 directory, 9 files
(base) joanius@FALKON:~/projects/art_with_drones/AD_Database$ node src/display-data.js |
```

Figure 6: Contenido de AD_Database

El nombre del *script* es suficientemente descriptivo para saber utilizarlo.

3.5 AD_Frontend

Se mostrarán las casillas vacías, en verde o en rojo, según su estado. Se conectará mediante conexión HTTP, usando *GET* para conectarse con el puerto de *AD_Engine* disponible.

Desde la carpeta */AD_Frontend* se ha utilizado un *docker-compose* para levantar *AD_Frontend*.

```
docker-compose up --build
```

```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Frontend/art_with_drones$ sudo docker-compose up --build
[sudo] contraseña para joanius:
Building ad_frontend
[+] Building 8.3s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 180B                                0.0s
=> [internal] load .dockerignore                                  0.1s
=> => transferring context: 34B                                     0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine  1.4s
=> [auth] library/node:pull token for registry-1.docker.io        0.0s
=> [1/5] FROM docker.io/library/node:18-alpine@sha256:435dcad253bb5b7f347ebc69c8cc52de7c912eb724 0.0s
=> [internal] load build context                                  6.7s
=> => transferring context: 3.47MB                                  6.2s
=> CACHED [2/5] WORKDIR /app                                       0.0s
=> CACHED [3/5] COPY package.json package-lock.json /app/        0.0s
=> CACHED [4/5] RUN npm install                                    0.0s
=> CACHED [5/5] COPY . /app                                        0.0s
=> exporting to image                                              0.0s
=> => exporting layers                                              0.0s
=> => writing image sha256:58c5224cade04fdef0a17dc2f1f892ea0e1ddfad1731eb1a042472617e367b2e 0.0s
=> => naming to docker.io/library/ad_frontend_image              0.0s
Starting ad_frontend_container ...
```

Figure 7: Levantando AD_Frontend

A continuación en el puerto **3000** se podrá visualizar la página web creada con *React.js*.

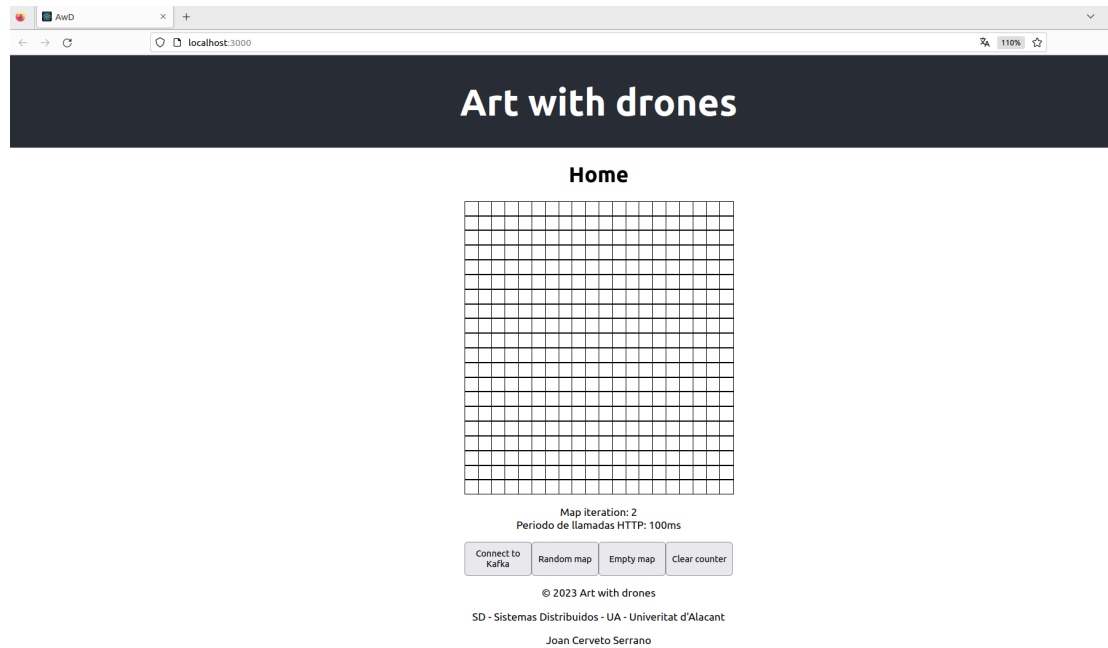


Figure 8: Levantando AD_Frontend vacío (si AD_Engine no funciona)

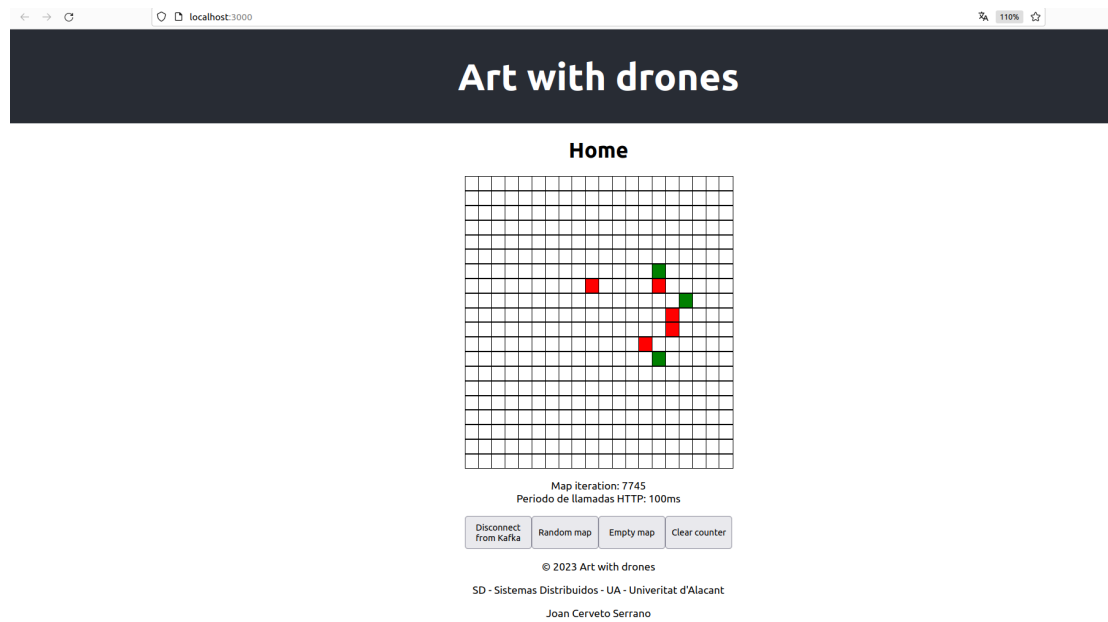


Figure 9: Levantando AD_Frontend vacío (si AD_Engine en funcionamiento)

3.6 **AD_Engine**

Cabe recalcar que realizar dos *expose* de este módulo. Uno para montar el servidor por *sockets* y otro para levantar el puerto *HTTP* con *express*.

Se marcan con caracteres ASCII es estado de la casilla, según los drones de la misma estén en una posición correcta, o no.

Desde la carpeta */AD_Engine* se ha utilizado un *docker-compose* para levantar *AD_Engine*.

```
docker-compose up --build
```

```

(base) joanius@FALKON:~/projects/art_with_drones/AD_Engine$ sudo docker-compose up --build
Building ad_engine
[+] Building 1.3s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 176B                                0.0s
=> [internal] load .dockerignore                                  0.1s
=> => transferring context: 2B                                       0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine  1.0s
=> [internal] load build context                                  0.1s
=> => transferring context: 1.70kB                                    0.1s
=> [1/7] FROM docker.io/library/node:18-alpine@sha256:435dcad253bb5b7f347ebc69c8cc52de7c912eb724 0.0s
=> CACHED [2/7] WORKDIR /app                                       0.0s
=> CACHED [3/7] COPY package*.json ./                             0.0s
=> CACHED [4/7] RUN npm install                                    0.0s
=> CACHED [5/7] COPY tsconfig.json tsconfig.json                 0.0s
=> CACHED [6/7] COPY src src                                       0.0s
=> CACHED [7/7] COPY .env .env                                     0.0s
=> exporting to image                                              0.0s
=> => exporting layers                                              0.0s
=> => writing image sha256:173a4c2f9440c8d0bb5bcb8f5a8b68a482f4c6e1ec81dd61970a9e08fbc525d9 0.0s
=> => naming to docker.io/library/ad_engine_image                 0.0s
Starting ad_engine_container ... done
Attaching to ad_engine_container
ad_engine_container | > ad_engine@1.0.0 build:start
ad_engine_container | > tsc && node dist/main.js
ad_engine_container |
ad_engine_container | MAIN_PORT: 8080
ad_engine_container | *****
ad_engine_container | Weather will be validate each 10000 .
ad_engine_container | *****
ad_engine_container | listening on 0.0.0.0:8080
ad_engine_container | Map publisher connected
ad_engine_container | Map Broker connected.
ad_engine_container | drones: 0 alive: 0 dead: 0
ad_engine_container | Map published.
ad_engine_container | Loading figures...
ad_engine_container | ERROR: Trying to create square. Out of Range. Moving to (1, 1) Invalid row number
ad_engine_container | (0)}
ad_engine_container | Figure Triangulo with 8 squares
ad_engine_container | 10-7 => 1
ad_engine_container | 9-8 => 2
ad_engine_container | 8-9 => 3
ad_engine_container | 9-9 => 4
ad_engine_container | 10-9 => 5
ad_engine_container | 11-9 => 6
ad_engine_container | 12-9 => 7
ad_engine_container | 11-8 => 8
ad_engine_container | Figure Cuadrado with 8 squares
ad_engine_container | 9-11 => 1
ad_engine_container | 10-11 => 2
ad_engine_container | 11-11 => 3
ad_engine_container | 9-12 => 4
ad_engine_container | 9-13 => 5

```

Figure 10: Levantando AD_Engine

```

ad_engine_container | Received message {"id_registry":1,"current_position":{"row":10,"col":7}}
ad_engine_container | droneId: 1 row: 10 col: 7
ad_engine_container | Show is active. Figure: Triangulo
ad_engine_container | New current position received: [1], 10-7
ad_engine_container | drones: 8 alive: 8 dead: 0
ad_engine_container | -----
ad_engine_container | currentFigure: Triangulo
ad_engine_container | -----
ad_engine_container | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
ad_engine_container | Waiting for drones to reach target position... 1699215074615
ad_engine_container | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
ad_engine_container | Sleeping for 1000 ms
ad_engine_container | ++++++
ad_engine_container | drones in map: [{"__status":0,"__target":{"__row":8,"__column":9,"__drones":{}},{"__id":6},{"__status":0,"__target":{"__row":9,"__column":8,"__drones":{}},{"__id":400},{"__status":0,"__target":{"__row":9,"__column":9,"__drones":{}},{"__id":5},{"__status":0,"__target":{"__row":10,"__column":7,"__drones":{}},{"__id":1},{"__status":0,"__target":{"__row":10,"__column":9,"__drones":{}},{"__id":7},{"__status":0,"__target":{"__row":11,"__column":8,"__drones":{}},{"__id":3},{"__status":0,"__target":{"__row":11,"__column":9,"__drones":{}},{"__id":2},{"__status":0,"__target":{"__row":12,"__column":9,"__drones":{}},{"__id":4}]
ad_engine_container | Received message {"id_registry":5,"current_position":{"row":9,"col":9}}
ad_engine_container | droneId: 5 row: 9 col: 9
ad_engine_container | Show is active. Figure: Triangulo
ad_engine_container | New current position received: [5], 9-9
ad_engine_container | drones: 8 alive: 8 dead: 0
ad_engine_container | -----
ad_engine_container | currentFigure: Triangulo
ad_engine_container | -----
ad_engine_container | #####
ad_engine_container | Weather task executed.
ad_engine_container | #####
ad_engine_container | Connected to weather server.
ad_engine_container | Handling good weather...
ad_engine_container | Connection closed
ad_engine_container | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
ad_engine_container | Waiting for drones to reach target position... 1699215075615
ad_engine_container | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
ad_engine_container | FIGURE Triangulo COMPLETED!
ad_engine_container | WAITING 10 SECONDS TO DRAW NEXT FIGURE...
ad_engine_container | Sleeping for 10000 ms

```

Figure 11: Ejecución normal de AD_Engine sin la *flag* de **map**

```
# MAIN PORT APP
MAIN_HOST=0.0.0.0
MAIN_PORT=8080
RECOVER=no # 'yes' or 'no'

# LOGGING: MAX Concurrent TCP socket connections
MAX_CONCURRENT_CONNECTIONS=2

# HTTP PORT
HTTP_PORT=8888

# DATABASE

# FÍSICO
#DATABASE_PATH=./AD_Database/database.db
#FIGURES_PATH=./data/AwD_figuras.json
# DOCKER
DATABASE_PATH=/app/AD_Database/database.db
FIGURES_PATH=/app/data/AwD_figuras.json

# KAFKA: Broker/Bootstrap-server del gestor de colas
KAFKA_HOST=192.168.0.235
KAFKA_PORT=29092

## TOPICS
KAFKA_TOPIC_MAP=map
KAFKA_TOPIC_START_FIGURE=start_figure
KAFKA_TOPIC_TARGET_POSITION=target_position
KAFKA_TOPIC_CURRENT_POSITION=current_position
KAFKA_TOPIC_KEEP_ALIVE=keep_alive
KAFKA_TOPIC_COMMUNICATION=communication
SHOW_MAP=no # 'yes' or 'no'

# Keep Alive

KEEP_ALIVE_INTERVAL=2000 # (ms) Cada cuánto tiempo se revisan los keep alive de todos los drones
KEEP_ALIVE_TIMEOUT=5000 # (ms) Cuánto tiempo tiene que pasar para que un drone se considere desconectado

# WEATHER API
WEATHER_VALIDATE=yes # 'yes' or 'no'
WEATHER_HOST=192.168.0.235
WEATHER_PORT=5000
WEATHER_TIMEOUT=10000 # ms

~
~
~
~
~
~
~
~
~
~
".env" 43L, 1115B
```

4,11 Todo

Figure 12: Fichero de configuración (.env) de AD_Engine

3.7 AD_Drone

Desde la carpeta `/AD_Drone` se han utilizado diferentes comandos.

```
docker-compose build ad_drone
```

```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Drone$ docker-compose build ad_drone
Building ad_drone
[+] Building 0.2s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 216B                                0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 57B                                     0.0s
=> [internal] load metadata for docker.io/library/python:3.9-alpine 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 1.51kB                                  0.0s
=> [1/6] FROM docker.io/library/python:3.9-alpine                0.0s
=> CACHED [2/6] WORKDIR /app                                       0.0s
=> CACHED [3/6] COPY requirements.txt .                            0.0s
=> CACHED [4/6] RUN pip install -r requirements.txt               0.0s
=> CACHED [5/6] COPY .env app/.env                                0.0s
=> CACHED [6/6] COPY src app/src                                  0.0s
=> exporting to image                                              0.0s
=> => exporting layers                                              0.0s
=> => writing image sha256:0a2f2ae0855b31b3d850f9884b8335de4a1eaa09423d93a2ccea6992d1d50cc9 0.0s
=> => naming to docker.io/library/ad_drone_image                  0.0s
```

Figure 13: Construyendo la imagen principal de AD_Drone

```
docker-compose build ad_drone
```

```

(base) joanius@FALKON:~/projects/art_with_drones/AD_Drone$ docker-compose run --name ad_drone_id_400 --rm
ad_drone python app/src/runnerEngine.py 400 -s
Creating ad_drone_ad_drone_run ... done
Starting drone=[400, alias_is_not_required, 9fd924b8-d]
Starting connection with AD_Engine for drone: [400, alias_is_not_required, 9fd924b8-d]...
Starting map consumer thread...
Starting map consumer...
Starting communication thread...
Starting communication consumer...
Starting authentication main thread...
Sending data. stage=auth_request: b'{"stage": "auth_request", "id_registry": 400, "token": "9fd924b8-d"}'
Received data: {
  "ok": true,
  "message": "Successful authentication. Subscribe to topics: \n
    'target_position',\n
    'start',\n
    'map'. Publish to topic: \n
    'current_
position\n
}
abriendo topics...
Starting targetPositionConsumer...
Topic name: target_position; Group id: target_position_400_1699214668.715179
Broker host: 192.168.0.235; Broker port: 29092
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 192.168.0.235:29092
Map consumer created: <kafka.consumer.group.KafkaConsumer object at 0x7f46169d28e0>
Communication consumer created: <kafka.consumer.group.KafkaConsumer object at 0x7f4615d5ae20>
Consumer created
Consumer open has been communicated.
Starting consuming target_position topic...
Subscribed to target_position...
Esperando unos segundos de cortesía para que se pueda abrir adecuadamente el consumidor de target_positio
n...
New map received. Time: 1699214672.880791
New message read from Kafka.
*****
0
length of message: 1
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
1:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
2:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
3:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
4:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
5:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
6:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
7:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
8:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
9:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
11: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
12: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
13: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
14: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
15: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
16: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
17: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
18: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
19: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
20: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

```

Figure 14: Levantando 1 instancia de AD_Drone

```
./run-engine-instances 7 1 &
```



```
(base) joanius@FALKON:~/projects/art_with_drones/AD_Drone$ ./run-engine-instances.sh 7 1 &
[1] 13745
(base) joanius@FALKON:~/projects/art_with_drones/AD_Drone$ Ejecutando 7 instancias de engine desde ID=1
hasta ID=7...
Building ad_drone
[+] Building 0.1s (2/3)                                docker:default
=> [internal] load .dockerignore                        0.0s
=> => transferring context: 57B                        0.0s
=> [internal] load build definition from Dockerfile    0.0s
=> => transferring dockerfile: 216B                    0.0s
=> [internal] load metadata for docker.io/library/python:3.9-alpine 0.0s
|
```

Figure 15: Levantando 7 instancias de AD_Drone

A continuación se muestra como se ve cuando hay una sola instancia en ejecución.

```
Drone is in target_position. Sending current position... 1699214727.6723912
Publishing current position...
Topic name: current_position
Publishing position: (10, 7)
New map received. Time: 1699214727.819806
New message read from Kafka.
*****
56
*****
length of message: 1
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
1:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
2:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
3:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
4:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
5:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
6:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
7:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
8:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
9:  □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
10: □ □ □ □ □ □ □ ■ □ □ □ □ □ □ □ □ □ □ □ □ □
11: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
12: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
13: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
14: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
15: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
16: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
17: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
18: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
19: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
20: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
```

Figure 16: Una sola instancia AD_Drone en el mapa

[illegible]

Figure 17: 8 instancias AD_Drone en el mapa

A continuación, se muestra como se "matan" o "paran" varias instancias de AD_Drone. Se ejecuta:

```
./run-engine-remove-instances.sh <NUM_INSTANCES> <FIRST_DRONE_ID> &
./run-engine-remove-instances.sh 7 1 &
```

```
run-engine-instances.sn          run-engine-remove-instances.sn
(base) joanius@FALKON:~/projects/art_with_drones/AD_Drone$ ./run-engine-remove-instances.sh 7 1
Ejecutando 7 instancias de engine desde ID=1 hasta ID=7...
id: ad_drone_engine_id_1
ad_drone_engine_id_1
Error response from daemon: No such container: ad_drone_engine_id_1
id: ad_drone_engine_id_2
ad_drone_engine_id_2
Error response from daemon: No such container: ad_drone_engine_id_2
id: ad_drone_engine_id_3
ad_drone_engine_id_3
Error response from daemon: No such container: ad_drone_engine_id_3
id: ad_drone_engine_id_4
ad_drone_engine_id_4
Error response from daemon: No such container: ad_drone_engine_id_4
id: ad_drone_engine_id_5
ad_drone_engine_id_5
Error response from daemon: No such container: ad_drone_engine_id_5
id: ad_drone_engine_id_6
ad_drone_engine_id_6
Error response from daemon: No such container: ad_drone_engine_id_6
id: ad_drone_engine_id_7
ad_drone_engine_id_7
Error response from daemon: No such container: ad_drone_engine_id_7
```

Figure 18: Matando 7 instancias AD_Drone.

4 Conclusiones

Para ser una práctica desarrollada en tan solo seis semanas ha supuesto un gran aprendizaje en la automatización de procesos, uso de *docker*, de *Apache Kafka*. Incluso de *python*, *JavaScript*, *TypeScript* y *Bash*. Aunque también de aspectos concretos de redes de computadores.

Por su puesto *docker* es una herramienta para cualquier ingeniero especificado en los sistemas distribuidos. Sin embargo, también cabe recalcar que lenguajes modernos como *JavaScript* o *Python* tienen gestores de paquete que simplifican la gestión de las versiones. Ya sea *npm* para *Node* o *pip* y los entornos virtuales (*.venv*) para *python*.