

Práctica de Sistemas Distribuidos

Art With Drones



Integrantes:

- Alejandro Guillén Merino **DNI:** 48790456G
- Víctor Padrón García **DNI:** 55178804X

Universidad de Alicante
Escuela Politécnica Superior
Curso 2023-24

1. Índice

1. Índice.....	2
2. Componentes software:.....	3
2.1. AD_Drone.....	3
- Class AD_Drone:.....	3
- Class EscucharDestino():.....	3
2.2. AD_Registry.....	4
- Class AD_Registry:.....	4
2.3. AD_Engine.....	5
- Class AD_Engine:.....	5
- Class EscucharDrones.....	6
- Class EscucharDrone.py.....	7
- Class Map.....	7
2.4. AD_Weather.....	8
- Class AD_Weather:.....	8
3. Despliegue de aplicación:.....	9
4. Funcionamiento de las aplicaciones:.....	10

2. Componentes software:

2.1. AD_Drone

AD_Drone es el encargado de crear los drones que se van a utilizar y en él se implementa la clase de los drones y los métodos para realizar transferencias de datos que se van a detallar a continuación:

- Class AD_Drone:

La clase drone tiene los siguientes atributos:

- + **estado:** Indica si el drone ha llegado a la posición final (Verde) o no (Rojo).
- + **posicionActual:** Indica la posición en la que se encuentra el drone.
- + **posicionFin:** Indica la posición donde debe acabar el drone.
- + **alias:** Nombre del drone.
- + **id:** Id del drone. Se cambia más adelante para asignarle el que se muestra en el mapa.
- + **token:** Almacena el token de acceso después de haberse registrado, para autenticar el drone con AD_Engine.

Por otro lado, existen diferentes funciones para su funcionamiento. Antes de acceder a ellas, se comprueban los parámetros que deben contener las ip y puerto de engine, registry y de kafka, así como el alias del drone.

Una vez iniciado, se muestra un menú para elegir entre registrarse (conectar con Registry) y entrar al espectáculo (conectar con Engine).

- + **registrarse(ip, puerto):** Recibe la ip y el puerto del dispositivo donde se encuentra AD_Registry y donde escucha. Establece conexión mediante un socket con AD_Registry y envía, codificado con utf-8, su id actual y su alias. A continuación, si todo ha ido bien, recibe de vuelta un token para autenticarse más adelante y un id que lo identificará en el mapa.
- + **solicitar_inclusión(ip, puerto):** Recibe la ip y el puerto de escucha del AD_Engine. Establece conexión mediante socket para autenticarse. Envía mediante el socket el token y el id, y recibe de vuelta, en caso de que la autenticación funcione de manera satisfactoria, un mensaje de que todo ha ido bien.

Una vez el drone se ha autenticado, se comienza un nuevo proceso para escuchar y notificar posiciones, el mapa,... Para ello se crea un objeto de la clase EscucharDestino.

- Class EscucharDestino():

Esta clase se encarga de la comunicación entre el drone y engine mediante kafka. Gestiona la información que llega y se la proporciona al drone para moverse. Cuenta con los siguientes atributos:

- + **Broker:** La dirección ip de broker.
- + **id:** El id del drone para el espectáculo.
- + **posicionFin:** La posición dónde debe acabar el drone.
- + **posicionActual:** La posición dónde está el drone actualmente.

- + **estado:** Si el dron ha llegado o no a la posición final.
- + **mapa:** El mapa con todos los drones en tiempo real.

A esta clase no se accede hasta que se autentifica el dron, y se comienza por el método `run()`. A partir de este se explicarán el resto en orden de aparición:

- + **run():** En este método se crean los consumidores del Destino y el Mapa y el productor de la posición. A continuación, se crean Threads para paralelizar los procesos y que puedan funcionar simultáneamente de manera concurrente. Por último, se muestra un menú con opciones para mostrar o no el mapa y salir del espectáculo.
- + **escucharPorKafkaDestino(consumidor):** Se utiliza un topic llamado "destino" y, en caso de recibir un mensaje, se lee la posición final y se almacena. En tal caso, el dron comenzará a moverse hacia esa posición.
- + **escucharEstadoMapa(consumidor):** Se utiliza un topic llamado "mapa" y espera a recibir el mapa que se envía desde Engine. En caso de recibirlo, se almacena en "mapa" para que se pueda utilizar más adelante. El mapa, como se crea en Engine, contiene todas las posiciones de todos los drones implicados en el espectáculo.
- + **enviarPosicion(productor):** Se utiliza un topic "posiciones" y envía al Engine por kafka el id y la posición del dron.
- + **mover(productor):** Comprueba el estado del dron y si no ha llegado a la última posición se mueve a una adyacente más cercana a su posición final. Por último, se invoca a "enviarPosicion()" para transmitir la posición actualizada al Engine.

De esta forma funciona un dron de manera interna, transmitiendo y recibiendo datos con AD_Registry y AD_Engine. Todo este código se encuentra en el archivo AD_Drone.py.

2.2. AD_Registry

AD_Registry es el encargado de registrar drones en la BD y proporcionar tokens de verificación a estos. Funciona de manera independiente a Engine.

- Class AD_Registry:

La clase AD_Registry tiene los siguientes atributos:

- + **token:** Almacena el nuevo token generado para proporcionarles a un dron.
- + **id_nueva:** Id que representará al dron en la bd.
- + **token_dron_actual:** Variable con el token para escribirlo en el socket.

Además, se han implementado diferentes métodos para controlar todo lo que ocurre, así como para dirigir el funcionamiento y el envío de los drones.

Al iniciar AD_Registry comprueba el parámetro, que es el puerto de escucha dónde espera recibir una solicitud para iniciar un registro, y se procede a iniciar el socket y esperar una nueva conexión. Se han implementado los siguientes métodos:

- + **leer_socket(socket, datos):** Permite extraer los datos del socket para poder tratarlos y analizarlos.

- + **enviar_socket(socket, token):** Escribe datos en el socket para transmitirlos.
- + **existe_en_bd(id, alias):** Comprueba si el drone que se va a registrar ya se había registrado con anterioridad.
- + **escribir_bd():** Almacena los datos del drone que se acaba de registrar en la BD para poder acceder a ellos.
- + **generar_token():** Genera un token usado para autenticar los drones después de registrarse.
- + **borrar_fichero():** Borra la BD para volver a registrar nuevos drones.

Así conseguimos que AD_Registry establezca conexión con cada drone, obtenga sus datos, almacene en la BD y devuelva un token para poder verificarlos más adelante. Todo el código referente a este punto se encuentra en el archivo AD_Registry.py.

2.3. AD_Engine

En AD_Engine se almacenan las clases y métodos necesarios para que todo funcione y se coordine de manera correcta. Es el encargado de comunicarse con los drones, obtener la figura e informarse del clima, así como de verificar los drones.

- Class AD_Engine:

La clase AD_Engine tiene los siguientes atributos:

- + **drones:** Almacena las posiciones finales de los drones
- + **figuras:** Vector con las figuras leídas del archivo "Figuras.json".
- + **dronesActuales:** Los drones que se han autenticado con Engine.

A parte de estos atributos son necesarios algunos métodos para que funcione todo correctamente.

Iniciando desde el main, se van a explicar todos los métodos en orden de invocación según el flujo del programa. En el main se comprueban los parámetros por línea de comando y se crea un proceso que crea un objeto de la clase EscucharDrones para escuchar los drones y autenticarlos drones. Por otro lado, se espera hasta que se detecta el archivo Figuras.json. A partir de ahí, se crea una instancia de la clase AD_Engine, el productor de destinos que enviará las posiciones finales de los drones, el productor de mapa, que enviará el mapa, y el consumidor de posiciones, que recibirá las posiciones de los drones. Por último, se inician todos estos procesos.

- + **start(productor_destinos, productor_mapa, consumidor):** Inicia el espectáculo. Crea un thread para controlar el clima y lee las figuras del archivo Figuras.json y crea un thread para enviar las posiciones a los drones. Comprueba constantemente si la figura se ha terminado o no para volver a enviar repetidamente el mapa.
- + **stop():** Detiene la acción que realiza el engine y para el espectáculo.
- + **escribe_socket(p_datos):** Manda datos de verificación al socket.
- + **lee_socket(p_datos):** Recibe los datos de verificación del drone.

- + **figura_completada()**: Comprueba si todos los drones han alcanzado sus posiciones finales.
- + **leer_figuras()**: Lee las figuras que existen en el archivo Figuras.json y lo elimina, a la espera de que llegue uno nuevo con nuevas figuras para el espectáculo.
- + **escuchar_posición_drones(consumidor)**: Recibe por kafka las posiciones de los drones y las actualiza en el vector correspondiente. Si no existe, crea un nuevo drone en el vector (no un AD_Drone) con ese identificador.
- + **enviar_mapa(productor)**: Envía el mapa por kafka a los drones a través del topic “mapa”.
- + **enviar_por_kafka_destinos(productor)**: Envía a través del topic “destinos” el destino de cada drone para finalizar la figura.
- + **consumidor_posiciones(broker)**: Crea el consumidor de las posiciones de los drones.
- + **productor_mapa(broker)**: Crea el productor del mapa para enviarlos a los drones mediante kafka
- + **productor_destinos(broker)**: Crea el productor de los destinos que se enviarán a los drones mediante kafka.
- + **clear_terminal()**: Método para borrar la terminal y conseguir sobrescribir el mapa cada vez que se actualiza.

Además, como se ha comentado anteriormente, es necesaria una clase que se llama EscucharDrones.

- Class EscucharDrones

Esta clase se encarga de la comunicación con los drones mediante el socket; mantiene abierta la escucha para que si hay nuevos drones se puedan autenticar a pesar de que el espectáculo haya comenzado. Tiene los siguientes atributos:

- + **puerto**: Puerto del AD_Engine dónde se va a escuchar para recibir nuevos drones.
- + **detener_thread**: Se indica si se debe finalizar el thread.

Además, tiene los siguientes métodos:

- + **run()**: Obtiene la ip de la red actual y comienza el funcionamiento de todo. Abre el socket y se mantiene a la escucha de que algún drone se conecte. Si se conecta, se crea una instancia de la clase EscucharDrone para recibir los datos y se mantiene a la espera de recibir un nuevo drone.

El código de estas dos clases se puede encontrar en el archivo AD_Engine.py.

- Class EscucharDrone

Esta clase sirve para recibir los datos de los drones que desean incorporarse al espectáculo y verificar su id y token. Tiene los siguientes atributos:

- + **drone:** Información del drone que se ha conectado.

Además se tienen los siguientes métodos para el funcionamiento.

Los métodos son los siguientes:

- + **run():** Recibe el drone por socket y lee la información de este con `lee_socket()`. La separa en diferentes posiciones de un vector separándolos por los espacios. A continuación, se llama al método `autenticar()` para verificar la información que ha llegado. Por último se escribe el socket con la información según si ha sido aceptado o denegado.
- + **lee_socket(p_datos):** Recibe el drone, decodifica la información y la devuelve en un string.
- + **autenticar(token, id):** Va leyendo la BD de los drones y comprueba para el id enviado si el token recibido coincide con el almacenado y devuelve un valor booleano.
- + **escribe_socket(p_datos):** En función de lo recibido de `autenticar` devuelve "aceptado" con el id con el orden de inclusión en el espectáculo o "denegado" para indicar que no se ha podido verificar el drone.

El código de esta clase se puede encontrar en el archivo `EscucharDrones.py`.

- Class Map

La clase Map representa el mapa que se muestra en la terminal con todos los drones. Sus atributos son:

- + **filas:** Cantidad de filas que tiene el mapa.
- + **columnas:** Cantidad de columnas que tiene el mapa.
- + **mapa:** String que contendrá el mapa una vez generado para poder transmitirlo.

Además de esto, cuenta con varios métodos para su funcionamiento.

Los métodos son los siguientes:

- + **get_filas():** Devuelve el número de filas que tiene el mapa.
- + **get_columnas():** Devuelve el número de columnas que tiene el mapa.
- + **to_string(drones, dronesActuales):** Inicializa el String mapa a vacío e invoca al método `print_mapa()` para generarlo. Devuelve el string.
- + **print_mapa(drones, dronesActuales):** Recorre todas las posiciones del mapa y según si existe un drone en esa posición o no añade al string un hueco vacío o el id del drone del color que corresponda: Rojo si no ha llegado a la posición final o Verde si ha llegado. Almacena todo el mapa en el string "mapa".

Esto es lo necesario para generar el mapa que se muestra por pantalla. Este código se encuentra en el archivo `Map.py`.

2.4. AD_Weather

En AD_Weather se trata la obtención y transmisión del clima al engine.

- Class AD_Weather:

Cuenta con los siguientes atributos:

- + **engine:** El socket para comunicarse con el engine.
- + **ciudades:** Vector que almacena las ciudades en el orden en el que se leen de la BD.
- + **temperaturas:** Vector que almacena las temperaturas en el orden en el que se leen de la BD.

A continuación, se va a detallar el funcionamiento de AD_Weather.

Comienza por el main, donde se comprueba que se haya recibido por parámetro por línea de comandos el puerto de escucha, y obtiene la dirección ip local del dispositivo en el que está. A continuación, crea un socket para comunicarse con Engine y comienza a escuchar. Cuenta con los siguientes métodos:

- + **run():** Contiene el bucle principal de la ejecución. Se mantiene a la espera de recibir una petición de temperatura y tenerla actualizada.
- + **actualizar_temperaturas():** Abre el archivo climas.txt para leer las ciudades y las temperaturas y las almacena en orden en los vectores correspondientes, haciendo que la posición 0 del vector temperaturas corresponda con la ciudad almacenada en la posición 0 del vector ciudades, y así con todas.
- + **escribe_socket(sock, datos):** Escribe la temperatura a mandar al Engine en el socket.
- + **leer_socket(sock):** Recibe la ciudad de la que se quiere saber la temperatura.
- + **clima_actual(ciudad):** Obtiene la posición dónde está almacenada la ciudad que se ha recibido por el socket y utiliza esa misma posición para obtener la temperatura del otro vector.

De esta manera se consigue desde obtener la información del clima hasta transmitirla al Engine. Todo este código se encuentra en el archivo AD_Weather.py.

3. Despliegue de aplicación:

Para desplegar de manera correcta la aplicación se deben ejecutar varios comandos de una manera específica, ya que si se inicia un servicio en otro orden pueden haber fallos por que no se pueden conectar los drones con registry o engine, o que engine no pueda pedir el clima a weather. Se van a tratar 3 ordenadores, aunque podrían ser más. El "PC-1" será dónde esté alojado AD_Weather y Kafka, el "PC-2" será dónde se encuentren AD_Engine y AD_Registry y por último, en el "PC-3" (y hasta el "PC-n") se encontrarán los drones. El orden sería el siguiente:

1. **Iniciar Zookeeper:** Lo primero es iniciar zookeeper en el PC-1. Para ello se debe abrir una terminal en la carpeta /AD_Weather/kafka/bin y ejecutar el siguiente comando: "\$./zookeeper-server-start.sh ../config/zookeeper.properties".
2. **Iniciar Kafka-Server:** A continuación, también en el PC-1, se debe arrancar kafka-server. Para ello, desde la carpeta /AD_Weather/kafka/bin se debe ejecutar el siguiente comando: "\$./kafka-server-start.sh ../config/server.properties"
3. **Crear los topics:** Para terminar de iniciar todo lo relacionado con kafka se deben crear los topics para el envío de mensajes. De nuevo en el PC-1 se debe ejecutar el siguiente comando desde la carpeta /Ad_Weather/kafka/bin: "\$./kafka-topics.sh --create --bootstrap-server (ip_ordenador):9092 --topic (topic) --partitions 100 --replication-factor 1". Se debe hacer esto para cada topic que necesitemos, en nuestro caso los topics "destino", "mapa" y "posiciones".
4. **Iniciar el AD_Weather:** Para terminar con el PC-1, se debe iniciar el AD_Weather. Se utilizará el comando "\$ python3 AD_Weather.py 8085", dónde 8085 es el puerto de escucha que utilizará para recibir la petición del clima. Al ejecutar el comando queda a la espera de recibir una petición de clima.
5. **Iniciar el AD_Registry:** En el PC-2 se debe abrir una nueva terminal e iniciar el AD_Registry con el siguiente comando: "\$ python3 AD_Registry.py 8081". El 8081 es el puerto que escuchará para recibir peticiones de registro de los drones. Al ejecutar el comando se queda a la espera de recibir un drone para registrarse.
6. **Iniciar al AD_Engine:** En el mismo dispositivo que el anterior, PC-2, se inicia AD_Engine, que será el motor principal del espectáculo y con quien se comunicarán los drones a través de kafka. Se inicia con el siguiente comando: "\$ python3 AD_Engine.py 9090 (max_drones) (ip_brpoker):9092 (ip_clima):8085 (ciudad)". Los parámetros son los siguientes: "max_drones" es el número máximo de drones permitidos conectados simultáneamente, "ip_broker" es la dirección ip de kafka, "ip_clima" es la dirección ip dónde se encuentra AD_Weather y "ciudad" es la ciudad dónde se está desarrollando el espectáculo. Al ejecutar el comando AD_Engine entra en espera hasta leer alguna figura y tener drones para el espectáculo.
7. **Crear los AD_Drones:** Por último, desde el PC-3, queda crear los drones. Cada drone se crea con el siguiente comando: "\$ python3 AD_Drone.py (nombre) (ip_engine) 9090 (broker_kafka) 9092 (ip_registry) 8081". Los parámetros son los siguientes: "nombre" es el alias del drone, "ip_engine" es la dirección ip del dispositivo dónde se encuentra AD_Engine, "broker_kafka" es la dirección ip del dispositivo dónde se encuentra kafka y el parámetro "ip_registry" es la dirección ip del dispositivo dónde se encuentra el AD_Registry. Al ejecutar el comando se muestra un menú con las opciones "Registrarse", "Entrar al Espectáculo" y "Salir". Se debe registrar primero y después entrar al espectáculo.

4. Funcionamiento de las aplicaciones:

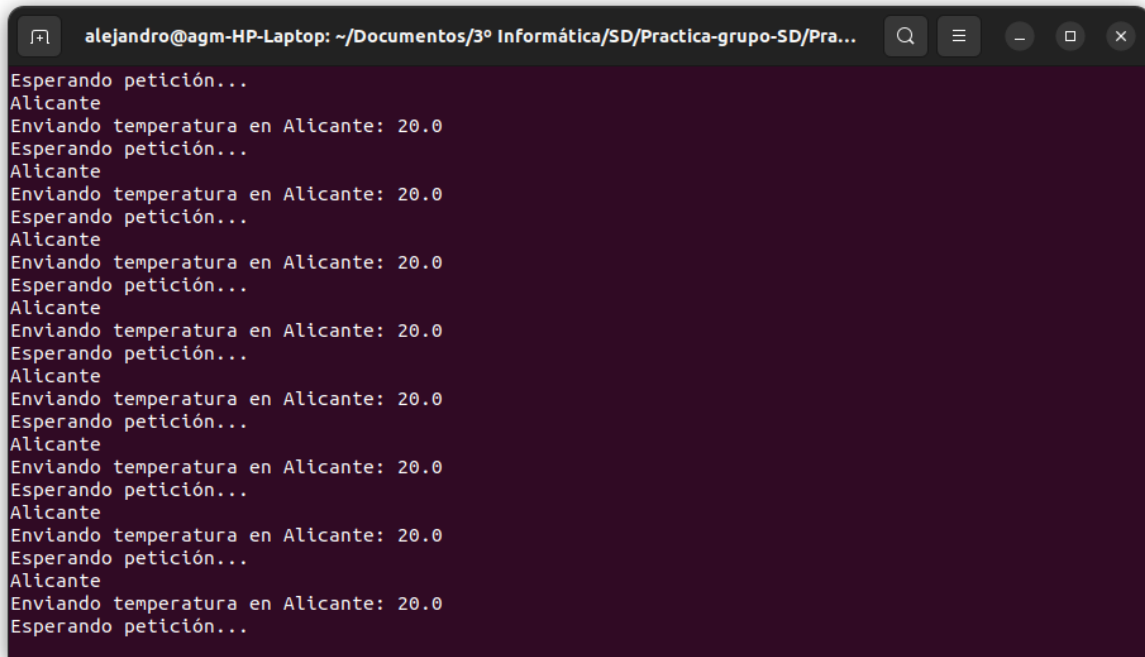
A continuación se va a explicar de manera más detallada cómo funcionan las aplicaciones en el conjunto del sistema. Partiremos desde el punto en el que se ha arrancado cada una de las aplicaciones y se va a explicar el flujo del sistema pasando por cada una de las aplicaciones.

1. **Registro de un Drone:** En el menú que se muestra en la terminal del drone se debe seleccionar la primera opción para registrar un drone. Éste establece conexión mediante socket con AD_Registry enviándole su alias y su id. Registry los almacena en la BD de drones (drones.txt) y le devuelve al drone un token, que también almacena en la BD, para autenticarse más adelante. El AD_Registry ya no tiene más interacción en el sistema.
2. **Verificación del Drone:** Ahora en el menú del drone se selecciona la segunda opción para autenticar el drone en el AD_Engine y tomarlo en cuenta para el espectáculo. El drone establece conexión con el engine y le envía su token y su id, que será verificado con el que hay almacenado en la BD. El AD_Engine le devuelve un id para referirse a él cuando lo muestre en el mapa (1, 2, 3, ...).
3. **Conseguir una Figura:** Simultáneamente a lo que ocurre en el punto 2, AD_Engine se mantiene a la espera de que se inserte un fichero llamado "Figuras.json". Cuando se recibe, se lee en orden, almacenando cada figura (nombre) con sus posiciones en un vector dentro del engine para poder compartirlo con los drones más adelante.
4. **Consultar el Clima:** Una vez hay drones y se tiene una figura, se empieza a comprobar el clima para saber si el espectáculo puede continuar. AD_Engine consulta cada pocos segundos a AD_Weather el clima de la ciudad dónde sucede el evento. AD_Weather, que lleva esperando una petición desde que se inició, ha leído el archivo "climas.txt" y ha almacenado las ciudades y sus temperaturas en dos vectores. Este proceso se realiza antes de cada petición, para mantener la información actualizada en el instante en el que se solicita la misma. Weather busca la temperatura de la ciudad solicitada y la devuelve al engine, volviendo a su posición de espera. En caso de que la ciudad no exista en la BD de climas, weather devuelve -1, que es una temperatura para apagar los drones y cancelar el espectáculo.
5. **Comunicar Posiciones:** Con la temperatura por encima de 0 °C se da comienzo al espectáculo. Engine envía por kafka a todos los drones que se han mantenido a la espera después de autenticarse las coordenadas a las que deben moverse como posición final para completar la primera figura. Éstos empiezan a moverse, respondiendo con la posición a la que se mueven constantemente al engine, que almacena estas posiciones en un vector según el id del drone que la envía.
6. **Imprimir Mapa:** Para imprimir el mapa se utiliza el vector antes mencionado, además de otro que contiene la posición final a la que debe llegar cada drone. Así se convierte en un string todo el mapa, que se puede imprimir en engine o enviarlo por kafka a todos los drones para que lo muestre el que quiera. Teniendo la posición actual y la final se puede establecer si el drone ha terminado su movimiento ("Verde") o no ("Rojo"). A continuación, se espera un par de segundos para poder ver y entender qué está ocurriendo.

7. **Final de Figura:** Una vez se finaliza una figura se espera a recibir una nueva. En caso de que no se reciba ninguna en un espacio de tiempo determinado, se termina el espectáculo.

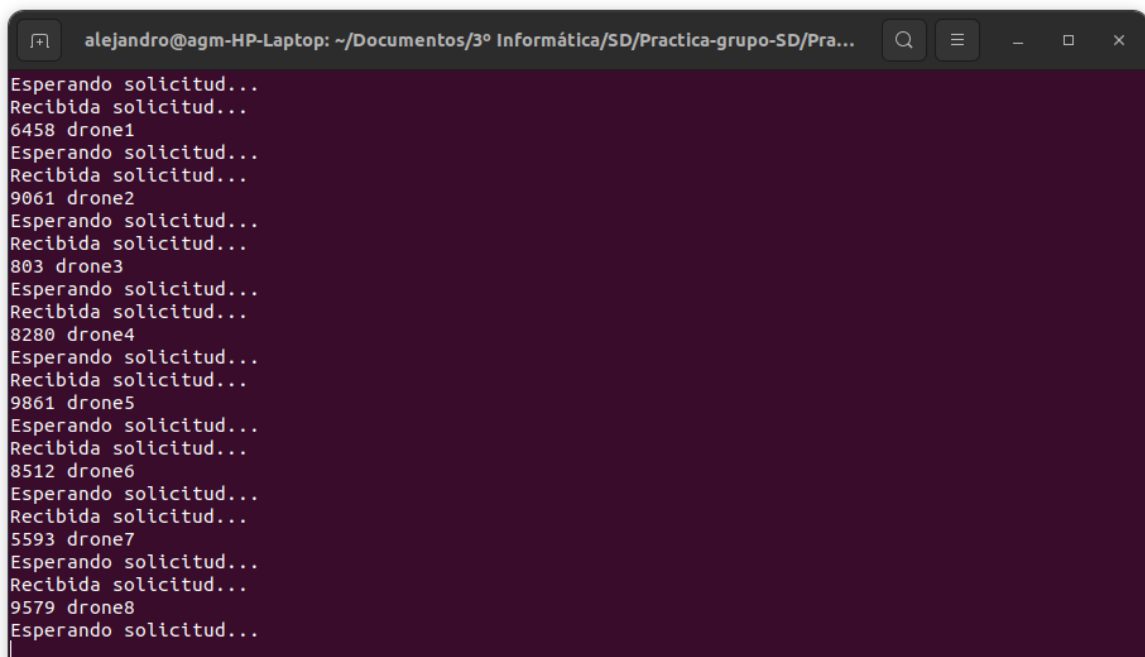
Ahora se van a mostrar unas capturas de las terminales dónde sucede todo lo que se ha dicho anteriormente.

- Ejemplo de ejecución normal:



```
alejandro@agm-HP-Laptop: ~/Documentos/3º Informática/SD/Practica-grupo-SD/Pra...
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
Alicante
Enviando temperatura en Alicante: 20.0
Esperando petición...
```

AD_Weather



```
alejandro@agm-HP-Laptop: ~/Documentos/3º Informática/SD/Practica-grupo-SD/Pra...
Esperando solicitud...
Recibida solicitud...
6458 drone1
Esperando solicitud...
Recibida solicitud...
9061 drone2
Esperando solicitud...
Recibida solicitud...
803 drone3
Esperando solicitud...
Recibida solicitud...
8280 drone4
Esperando solicitud...
Recibida solicitud...
9861 drone5
Esperando solicitud...
Recibida solicitud...
8512 drone6
Esperando solicitud...
Recibida solicitud...
5593 drone7
Esperando solicitud...
Recibida solicitud...
9579 drone8
Esperando solicitud...
```

AD_Registry

```
victor@victor-KLVL-WXX9: ~/Escritorio/SD/PracticaSD/PracticaSD/Nucleo$ python3 AD_Engine.py 9090 8 192.168.1.84:9092 192.168.1.84:8085 Alicante
Escuchando puerto 9090
Maximo de drones establecido en 8 drones
Esperando drone...
```

AD_Engine (Inicio)

```
victor@victor-KLVL-WXX9: ~/Escritorio/SD/PracticaSD/PracticaSD/Drone
victor@victor-KLVL-WXX9: ~/Escritorio/SD/PracticaSD/PracticaSD/Nucleo$ cd ..
victor@victor-KLVL-WXX9: ~/Escritorio/SD/PracticaSD/PracticaSD$ cd Drone
victor@victor-KLVL-WXX9: ~/Escritorio/SD/PracticaSD/PracticaSD/Drone$ python3 AD_Drone.py bebe 192.168.1.84 9090 192.168.1.84 9092 192.168.1.84 8081
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
1
---Drone registrado de manera satisfactoria---
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
2
---Drone unido de manera satisfactoria---
```

AD_Drone

```
victor@victor-KLVL-W
victor@vict... x victor@vict... x victor@vict... x victor@vict... x victor@vict...
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
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
1
---Drone registrado de manera satisfactoria---
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
2
---Drone unido de manera satisfactoria---
[1] Imprimir Mapa
[2] Salir del espectáculo
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
[1] Imprimir Mapa
[2] Salir del espectáculo
```

```
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
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
1
---Drone registrado de manera satisfactoria---
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
2
---Drone unido de manera satisfactoria---
[1] Imprimir Mapa
[2] Salir del espectáculo
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
[1] Imprimir Mapa
[2] Salir del espectáculo
```

(mapa en engine)

(mapa en drone)

```
alejandro@agm-HP-Laptop: ~/Documentos/3º Informática/SD/Practica-grupo-SD/PracticaSD/...
Espectaculo finalizado
```

AD_Engine (Fin)

- Si se recibe una temperatura menor o igual a 0.0 °C:

```
victor@victor-KLVL-WXX9: ~/Escritorio/
victor@vict... x victor@vict... x victor@vict... x victor@vict... x victor@vict... x victor@vict...
victor@victor-KLVL-WXX9:~/Escritorio/SD/PracticaSD/PracticaSD$ cd Weather
victor@victor-KLVL-WXX9:~/Escritorio/SD/PracticaSD/PracticaSD/Weather$ ls
AD_Weather.py climas.txt kafka_2.13-3.5.1
victor@victor-KLVL-WXX9:~/Escritorio/SD/PracticaSD/PracticaSD/Weather$ python3 AD_Weather.py 8085
Esperando petición...
Alicante
Enviando temperatura en Alicante: 0.0
CONDICIONES CLIMATICAS ADVERSAS.ESPECTACULO FINALIZADO
```

```
victor@victor-KLVL-WXX9: ~/Escritorio/SD/PracticaSD/PracticaSD/Nucleo
victor@vict... x victor@vict... x victor@vict... x victor@vict... x victor@vict... x victor@vict... x victor@vict... x victo
victor@victor-KLVL-WXX9:~/Escritorio/SD/PracticaSD/PracticaSD/Nucleo$ python3 AD_Engine.py 9090 5 192.168.1.84:9092 192.168.1.84:8085
5
ERROR: Los parámetros no son correctos
victor@victor-KLVL-WXX9:~/Escritorio/SD/PracticaSD/PracticaSD/Nucleo$ python3 AD_Engine.py 9090 5 192.168.1.84:9092 192.168.1.84:8085 Alicante
Escuchando puerto 9090
Maximo de drones establecido en 5 drones
Esperando drone...
^CTraceback (most recent call last):
  File "AD_Engine.py", line 350, in <module>
    time.sleep(1)
KeyboardInterrupt
^CException ignored in: <module 'threading' from '/usr/lib/python3.8/threading.py'>
Traceback (most recent call last):
  File "/usr/lib/python3.8/threading.py", line 1388, in _shutdown
    lock.acquire()
KeyboardInterrupt:
victor@victor-KLVL-WXX9:~/Escritorio/SD/PracticaSD/PracticaSD/Nucleo$ python3 AD_Engine.py 9090 8 192.168.1.84:9092 192.168.1.84:8085 Alicante
Escuchando puerto 9090
Maximo de drones establecido en 8 drones
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
Esperando drone...
Contactando drone...
CONDICIONES CLIMATICAS ADVERSAS.ESPECTACULO FINALIZADO
```