Práctica Art With Drones

Alejandro Torres Montero, Pablo David Ortuño Ñíguez

Universidad de alicante

**Índice:**

**1.- Introducción**

**2.- Explicación de la estructura del proyecto**

**3.- Funciones y datos sobre Kafka**

**4.- Explicación de los módulos**

**4.1.- ad\_engine.py**

**4.2.- ad\_registry.py**

**4.3.- drone.py**

**5.- Despliegue**

**1.- Introducción**

El sistema Art with Drones se encarga de recibidos una serie de drones, realizar un espectáculo dibujando sobre un mapa que en este caso será de 20 x 20, un serie de figuras. Estos espectáculos se realizarán en base al tiempo de la ciudad en la que se estén realizando, y en caso de cumplir con las condiciones, se empezará la función. Cabe decir que hemos cifrado los datos y contraseñas de los drones para que estos sean seguros y no se pueda acceder a ellos sin saber las credenciales, aunque todos estos detalles los veremos más adelante.

**2.- Explicación de la estructura del proyecto:**

El proyecto consta de los siguientes archivos:

* Ad\_registry.py: Encargado del registro de drones y la creación de nuevos tokens de seguridad para los drones que lo soliciten. También cifrara las contraseñas de cara al acceso a la base de datos mediante funciones HASH
* Ad\_engine.py: Encargado de manejar la lógica del mapa y coordinar el movimiento de todos los drones. Se encargará de verificar que drones están preparados y son necesarios para el espectáculo así como de recibir la información del tiempo, y acceder a la base de datos para saber en que punto se encuentra cada dron en todo momento.
* Drone.py: Son cada uno de los drones que conforman los dibujos encargados por engine, cada uno manejará su propio movimiento de cara a la figura que en ese momento esté coordinando el engine
* Ad\_weather: Servicio encargado de recibir y enviar los datos del tiempo a ad\_engine.py.

La BBDD en la que se guardan los drones es engineDDBB.json. Hemos optado en este caso por guardar los archivos en un tipo JSON debido a que buscábamos una mayor facilidad de cara a trabajar con los datos pero en un futuro se podría estudiar la forma de integrar cualquier base de datos relacional.

También disponemos de un archivo FIGURA.json, el cual indicara el número de drones necesarios para cada figura, así como la posición final de cada dron para completar cada figura. Todas las figuras serán identificadas por un nombre.

En cuanto a la comunicación entre los diferentes archivos, ad\_registry y ad\_engine reciben los datos de la base de datos mediante las funciones apropiadas para lectura y escritura de los formatos JSON de Python. Ad\_registry manda y recibe los datos de los drones mediante sockets, mientras que los drones se comunicarán con ad\_engine mediante el uso de un sistema de gestión de colas como en este caso es Kafka.

**3.- Funciones y datos sobre Kafka.**

Antes de entrar en detalles sobre las implementaciones de cada archivo, veamos como hemos distribuido el uso de Kafka. Tanto el engine como los drones tienen momentos de escucha como de envío de mensajes dependiendo de la acción que estén realizando en ese momento. Para su correcto funcionamiento se han creado diferentes topics, que son los siguientes:

* Posiciones: El engine escucha y coordina todas las posiciones de los drones, que usan este topic para enviar su posición cada vez que realizan un movimiento, de forma que cada vez que un dron se mueve, lo recibe y actualiza el mapa teniendo en cuenta esta actualización.
* Volver: Topic utilizado por el engine para enviar a los drones un mensaje indicando si estos deben volver a la base o no.
* Siguiente: topic usado para indicar a los drones que va a comenzar a dibuarse la siguiente figura
* Figura: Topic usado para que el engine envíe a cada uno de los drones las posiciones finales de la siguiente figura, de forma que todos realicen la misma figura.
* Comenzar: El engine avisa a todos los drones que se va a comenzar a dibujar una nueva figura
* Engine: el engine envia datos a cada dron para informar de si se ha iniciado sesión correctamente
* Confirmar: el engine escucha a cada dron para saber cuales están preparados para dibujar alguna figura.

Se han creado diferentes funciones auxiliares las cuales se encuentran en el archivo AD\_aux.py para facilitar el envío y recepción de los mensajes de Kafka:

Texto

Descripción generada automáticamente

Create\_producer se usará para crear un productor con la ip y puerto que se le pase como parámetro, la función send\_kafka será la encargada de enviar los mensajes con el productor creado. Se ha creado estas funciones con el fin de que desde cualquier archivo que lo necesite, con una simple llamada a send\_kafka, indicando dirección, topic y mensaje a enviar, se cree automáticamente el productor y se envie el mensaje.

A continuación nos encontramos las funciones dedicadas a los consumidores de mensajes del gestor de colas. Vemos que en este caso disponemos de 4 funciones, siendo create\_consumer, receive\_kafka, receive\_kafka\_pos y create consumer\_pos. Se ha optado por esta estructura ya que a lo largo de la realización del proyecto nos hemos enfrentado a un problema a la hora de leer los mensaje de Kafka enviados a un topic al mismo tiempo. Ya que en caso de enviar varias iteraciones de mensajes varios drones a la vez, solo se leía un mensaje a la vez, descartando el resto de mensajes los cuales nunca llegaban a ser consumidos, pese a que si se encontraban dentro del topic. Por ejemplo el funcionamiento en el siguiente caso, al menos el esperado sería que Engine consumiese los mensajes 1, 2 y 3 (que los situamos en la misma columna porque llegan simultáneamente), en cualquier orden, para posteriormente consumir los mensajes 4, 5 y 6 (que también habrían llegado al mismo tiempo):

Diagrama

Descripción generada automáticamente

Sin embargo, cuando ejecutábamos leía cualquiera de los mensajes de la primera iteración de forma aleatoria (1, 2 o 3), posteriormente leía únicamente uno de los mensajes igualmente de forma aleatoria de la segunda iteración, por ejemplo:

Diagrama

Descripción generada automáticamente

Este era el funcionamiento que describía la ejecución del programa, consumiendo únicamente el mensaje 1 y 5 (que los seleccionaba de forma aleatoria), y quedando los mensajes 2, 3, 4 y 6 sin consumir en ningún momento. Este es el último es el comportamiento que siguen las funciones create\_consumer y receive\_kafka. Podríamos pensar que simplemente descartando estas funciones y empleando las otras 2 podríamos crear nuestro proyecto, pero hemos comprobado que con esta implementación los mensajes que había previamente en cola, no se leen, lo cual para determinados casos, puede ser beneficioso. Por ejemplo, imaginemos que queremos que el sistema escuche solo un determinado tiempo para saber que drones están listos y así indicarles que comiencen la figura, aunque falte alguno. Estas funciones podrían ayudarnos a ello, ya que en caso de haber mensajes en la cola sin leer como por ejemplo un dron que había mandado un mensaje de preparado, pero ya no esta preparado, en lugar de tenerlo en cuenta para la figura cuando éste ya no esta listo para la ejecución, no se le tiene en cuenta ya que se descarta su mensaje. Las funciones create\_consumer y receive Kafka son las siguientes:

Texto

Descripción generada automáticamente

Vemos que ambas funciones reciben como parámetros la dirección IP y puerto del servidor Kafka y el topic al que se van a suscribir. Con esto, con una simple llamada a receive\_kafka, se crearía el consumidor y comenzaría la escucha, al recibir un mensaje, este se devuelve directamente con su .value, que indica el mensaje en sí, pero codificado en bytes debido a la implementación del propio servicio Kafka.

Las funciones create\_consumer\_pos y receive\_kafka\_pos son las siguientes:

Texto

Descripción generada automáticamente

Vemos que la única diferencia con respecto a las otras 2 es la inclusión en este caso del parámetro group\_id, que es el encargado de leer todos los mensajes que llegan a la cola de forma simultanea. Por lo que es más indicado para el manejo de la posiciones de los drones, ya que si varios drones se mueven a la vez, con esta función recogerá todos los mensajes que lleguen a la cola sin dejar ninguno sin consumir.

**4.- Explicación de los módulos**

A continuación, veremos una explicación más detallada de los diferentes módulos. Explicaremos con más detalle las decisiones de diseño que se han elegido, así como el funcionamiento de cada dron. La realización se ha hecho de forma que funcione lo mas parecido posible a como nosotros plantearíamos este problema de cara a un ámbito real:

**4.1.- ad\_engine.py**

Este módulo consta de la lógica principal del proyecto, al comenzar la ejecución y entrar en el main, se nos pedirá elegir entre una de las siguiente opciones:

+Texto

Descripción generada automáticamente

En caso de escoger la primera, se mostrará la situación actual del mapa.

La segunda nos mostrará los datos de cada uno de los drones registrados.

La tercera, iniciará las figuras, cosas que veremos más adelante y la cuarta nos servirá para salir del programa.

Antes de pedirnos las opciones se crea un hilo con un socket que permanece a la espera, este socket sirve para escuchar a los drones que quieran autenticarse en el engine, y permanecerá siempre a la escucha, para comprobar las credenciales en caso de que quieran iniciar sesión. Toda esta gestión la realiza la siguiente función:

Captura de pantalla de un teléfono celular

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Aquí, obviando los registros de eventos, lo que se hace es recibir mediante el socket entre el dron y el registry un alias y una contraseña. Posteriormente se abre la base de datos con la información de los drones registrados y se busca alguna coincidencia de usuario con estos datos. Si se encuentra se envía de vuelta por el socket el id del dron en caso de que el token no sea nulo. Si el token es nulo se envía de vuelta None. Si no se encuentra un usuario con ese alias y esa contraseña se envía de vuelta false. Finalmente se cierra la conexión del socket.

En caso de escoger la tercera opción entraremos en la función iniciar:

Texto

Descripción generada automáticamente

Tras esto en caso de poder abrir el fichero de figuras, se comienza a iterar sobre todas las figuras del fichero y se almacena en num\_drones, el número máximo de drones que pueden participar en cada figura y se llama a la función hacer:

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Al entrar a esta función vemos que se le han pasado 4 parámetros, drones\_data, que contiene los datos del JSON de todos los drones, num\_drones y figura, que contiene los datos de la figura que se quiere dibujar en este momento. Se llama a la función join\_show y posteriormente, una vez que los drones que están preparados han confirmado que lo están, cada uno de ellos queda a la espera de recibir los datos de su posición final. La función join\_show es la siguiente:

Texto

Descripción generada automáticamente

Primero se nos pide la ciudad en la que vamos a hacer el espectáculo, tras esto, accedemos a la API de openWeather para obtener la temperatura de dicha ciudad:

Texto

Descripción generada automáticamente

En esta función, se hace una petición a la API REST de openweather para pedirle la información acerca de una ciudad. Para ello se le pasa en la url la ciudad y la api\_key que obtuve al registrarme en su plataforma. Después de realizar dicha petición se obtiene de la respuesta obtenida (response) el campo temp (temperatura ) del campo main (información principal de la ciudad requerida) y se devuelve dicha temperatura. Si no se obtiene una respuesta válida del servidor, se devuelve None.

En caso de ser la temperatura mayor que 0, se envía los datos de los drones en cada iteración por si acaso se prepara un dron en último momento durante la ejecución del join\_show. Se espera a que los drones confirmen y posteriormente se indica que drones están preparados. Cabe indicar que para el correcto funcionamiento del proyecto, para coordinar los drones se designa un capitán, esto con el fin de asegurarnos tanto que hay un dron activo, como de indicarles a todos que comiencen el espectáculo. Este dron se eligirá siguiendo el criterio de que su ID debe ser igual al número máximo de drones necesarios para pintar una figura, por lo que si son necesarios 8 drones para completar la figura, al menos el dron con id 8 debe estar preparado. Cualquiera de los demás puede no estar preparado o unirse posteriormente al espectáculo, pero el coordinador debe estarlo para cada una de las figuras.

Tras enviar a cada uno de los drones los datos de la figura a dibujar, los drones detectan su posición final, y entonces el engine usa el topic comenzar para indicarles que pueden comenzar el espectáculo. Una vez dada la señal para comenzar a dibujar la figura, se llama a la funcion update\_map:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Esta función es la encargada de recibir cada una de las posiciones a las que se mueve cada uno de los drones y actualizar en mapa en consecuencia a la nueva posición de cada uno de los drones.

La función llama a receive\_kafka\_pos, que como vimos en el apartado 3 se encarga de recibir por Kafka todos los mensajes que llegan simultáneamente. Esta funcion devolverá una cadena formada por {idDron, posX, posY}, posteriormente asigna a las variables idDron, X, e Y sus respectivos datos, y se actualiza la base de datos engineDDBB con la nueva posición del dron que haya mandado los datos. Para saber que dron tenemos que modificar usamos la variable idDron.

Posteriormente tenemos una condición que como puede verse compara las variables X e Y con 99, y en un primer momento podemos pensar que es descabellado, pues no cabe en el mapa, pero hemos hecho que los drones para indicar que han llegado a su posición final, una vez se encuentren en ella, envíen esa posición para indicar que han terminado, si el engine detecta esto, aumenta la variable NUM\_TERMINADOS en 1, que en el momento que esta sea igual a NUM\_PREPARADOS, que es el número de drones que han indicado que están preparados en join\_show, dará por finalizada la figura. En este momento se nos indicará que todos han llegado a la posición final, y nos preguntará que si queremos que los drones vuelvan a la base (posición: (0, 0)). En caso de indicarle que si que queremos que vuelva, los drones volverán a la base y esto se le indicará a los mismos mandando un mensaje por el topic volver con el dato “s”. En caso de que no queramos que vuelvan a la base se enviará el mensaje “n”, por el mismo topic.

En caso de que queramos dibujar la siguiente figura directamente, la función se detendrá, se volverá a la funcion hacer, y esta repetirá el proceso hasta acabar todas las figuras del fichero. En caso de que indiquemos que si que queremos que todos los drones vuelvan a la base, se seguirá escuchando las nuevas posiciones de los drones, y cada vez que uno de ellos llegue a la posición 0,0, se reducirá NUM\_TERMINADOS en 1, hasta llegar 0, momento en el que se indicará que todos los drones han llegado a la base, y mediante el topic siguiente, se les indicara que va a comenzar a dibujarse la siguiente figura, en este momento de la misma forma, se volverá a la función hacer y se volverá a iterar hasta dibujar todas las figuras del fichero.

Cada vez que se itera sobre update\_map, se llama a la funcion print\_map, que será la encargada de imprimir por consola el mapa:

Texto

Descripción generada automáticamente

**4.2.- ad\_registry.py**

Este módulo se encarga de registrar nuevos drones y de actualizar el tokens de los mismos. El main es el siguiente:

Texto

Descripción generada automáticamente

Aquí se comprueba que se le pase como argumento a la hora de ejecutar el módulo el puerto en el que queremos que se ejecute. El IP será la dirección de loopback 127.0.0.1.

Posteriormente se llamará a la función inicializar\_token\_expiry:

Texto

Descripción generada automáticamente

Esta función se encarga de añadir el campo token\_expiry a todos los drones de la BBDD que no dispongan de dicho campo. Este campo será el tiempo que, una vez alcanzado, hará que expire el token del dron. En un principio se inicializa a nulo.

Posteriormente se lanzan 2 hilos paralelos a la ejecución del hilo principal:

El primer hilo ejecutará la función iniciar\_api:

Texto

Descripción generada automáticamente

Aquí se lanza la API REST de flask de forma cifrada en localhost y el puerto siguiente al utilizado para lanzar el registry. Esta es la app que se lanza:



Esta API REST contiene la ruta /register, mediante la cual se actualiza el token de un dron que solicita la renovación de su token:

Texto

Descripción generada automáticamente

Aquí se reciben vía api un alias y una contraseña. Se vuelve a hashear la contraseña (ya que en la base de datos del núcleo se almacena doblemente hasheada) y se busca el dron de la base de datos que coincide con ese alias y contraseña. En caso de no existir, se devolverá None.

Si se ha encontrado un dron, se le genera un nuevo token mediante la librería uuid y se le asignan 20 segundos de vida al token. Posteriormente, se guarda en la base de datos y se registra el evento.

Si no se encuentra ningún dron con ese usuario y contraseña, se registra el evento de que se ha fallado en la autenticación y se devuelve un mensaje de error con su correspondiente status.

El segundo hilo lanza la función limpiar\_tokens\_expirados:

Texto

Descripción generada automáticamente

En esta función se abre la base de datos y se comprueba la hora actual. Se van recorriendo los drones comprobando que tengan el campo token\_expiry, que dicho campo no sea nulo y que sea menor o igual que el actual. Si el dron cumple estas tres condiciones, se registra el log de que se le ha expirado el token debido a que han pasado +20s desde que se le asignó y se le establece tanto el token como el token\_expiry a nulo. Se duerme al proceso durante 1 segundo y se repite todo. De este modo, cada segundo comprobamos en la base de datos que drones han superado el tiempo de expiración y se le borran los tokens.

A continuación se crea un contexto seguro cifrado por ssl utilizando el certificado y la clave almacenados en la carpeta certs (al tratarse del servidor, se necesitan tanto el certificado como la clave), y se lanza un servidor de sockets que se queda a la escucha indefinidamente a la espera de conexiones por parte de algún dron para registrarse. Cuando recibe algún intento de conexión, se lanza la función registrar\_dron sobre dicha conexión.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

En esta función se registra el evento de que se ha establecido conexión con una determinada IP del dron que ha lanzado la conexión. Después se reciben el alias y la contraseña del dron (previamente cifrada) por el socket. Se genera un id único mediante la función generar\_id\_unico, la cual recorre los drones de la base de datos, se queda con el ID más alto y continúa asignando ID’s a partir del más alto encontrado:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

También se generan un token aleatorio y único mediante la librería uuid, unas posiciones x e y que se inicializan a 0 y el token\_expiry que es la hora actual pero sumándole el tiempo de vida del token. Además, la contraseña se hashea de nuevo para aumentar la seguridad del sistema.

Posteriormente, se almacena el dron en la base de datos, se le envía de vuelta al dron por vía socket su id y alias y se registra el evento de que se ha registrado satisfactoriamente un nuevo dron.

En caso de que salte alguna excepción, esta será capturada y mostrada por pantalla además de quedar guardada en la auditoría de logs.

Finalmente se cierra el socket.

**4.3.- Drone.py:**

Cuando ejecutamos un dron veremos el siguiente menú:

Texto

Descripción generada automáticamente

En el si escogemos la primera de las opciones llamaremos a la funcion registrar\_dron\_via\_socket():

Texto

Descripción generada automáticamente

Aquí se crea un contexto seguro utilizando ssl y el certificado con el cual se certifica el servidor (certificado autogenerado por nosotros). Luego se pide un alias y una contraseña para registrar al dron en el servidor. La contraseña se hashea para evitar que cualquier hacker que esté realizando sniffing de paquetes de red no sea capaz de ver la contraseña sin hashear enviada por el socket. Esos dato se pasan a una diccionario y se envían por el socket al servidor. Luego se lee la respuesta del servidor y se muestra el id que se le ha generado al dron con el alias requerido.

En caso de escoger la segunda opción en el menú, la cual sirve para asignar un token valido durante 20 segundos al token, llamaremos a la funcion solicitar\_nuevo\_token\_via\_rest, la cual es la siguiente:

Texto

Descripción generada automáticamente

Aquí se pide un alias y una contraseña, que se hashea y se añade a un diccionario junto al alias y se manda por medio de una petición API al API REST programado en el registry. Luego se muestra por pantalla la respuesta del servidor y se vuelve al main.

Si por el contrario en el menú escogemos la tercera opción, accederemos a la funcion opt3(), que es la siguiente:

Texto

Descripción generada automáticamente

Lo primero que pedirá es que se autentique el dron, introduciendo su alias y password, tras esto se hashea la contraseña y se llama a la funcion getToken:

Texto

Descripción generada automáticamente

Esta función se encarga de autenticarse por socket contra el engine, y en caso de que los datos (alias y contraseña) sean correctos y el dron tenga asignado algún token, devuelve el id del dron.

Si consigue autenticarse, entra en la función preparado:

Texto

Descripción generada automáticamente

En ella los drones confirman al Engine que están preparados, dado que se ha creado el consumidor de Engine con create\_consumer visto en el punto 3, y como hemos visto, no puede recibir mensajes simultáneamente, cada dron enviará el mensaje de preparado por Kafka con el topic confirmar en un segundo diferente, el cual equivaldrá al id del Dron. Cuando hayan enviado el mensaje, llamarán a la función figura, la cual recibe el dron activo y su id:

Texto

Descripción generada automáticamente

En esta función el dron queda a la espera que el Kafka le envíe los datos de la figura que se va a dibujar, y busca su posición final en ellos, tras recibir los datos, queda a la espera que el Engine le avise por el topic comenzar de Kafka que debe iniciar su movimiento. Una vez reciba la confirmación, avisa del comienzo de su movimiento y llama a la función move\_dron, a la que le pasa el dron y las posiciones que debe alcanzar.

La función move\_dron es la encargada del movimiento de dron hasta la posición final de la figura, y es la siguiente:

Texto

Descripción generada automáticamente

En ella, se compara en la condición del bucle while la posición actual del dron con la posición final y se aumenta o se disminuye la X o la Y hasta llegar a la posición final. En cada iteración se envía por Kafka con el topic posiciones la posición a la que se ha movido el dron en cada paso al Engine para que actualice el mapa y la BBDD, en caso de salir del bucle, se avisa que ha llegado al destino y para ello envía las posiciones 99, 99 por Kafka al engine para avisar de su llegada como hemos visto en el 4.1, después queda a la espera de recibir un mensaje por el topic volver, el cual le indicará si debe volver a la base, o si por el contrario debe dibujar directamente la siguiente figura. En caso de dibujar la figura directamente volverá a llamar a la función preparado, en caso contrario, se llamará a la funcion move\_dron\_inicial:

Texto

Descripción generada automáticamente

El funcionamiento de esta función es el mismo que el de move\_Dron, con la salvedad de que en este caso una vez llegue a la posición final, que en este caso será 0,0 quedará a la espera de recibir un mensaje por el topic siguiente, el cual lo usará el Engine para indicar a todos los drones que el resto han llegado a a la base y que por tanto se procederá a realizar la siguiente figura. Una vez hecho esto, se vuelve a llamar a la funcion preparado.

**5.- Despliegue**

Para el despliegue, hemos seguido la siguiente estructura:

Diagrama

Descripción generada automáticamente

Hemos optado por ejecutar el servidor de KAFKA en el mismo PC que engine, ya que de esta manera el Kafka siempre estará activo para el engine, el cual es el encargado de coordinar a los drones, por lo que en este caso, en caso de caer el Kafka, al menos los drones podrían seguir comunicándose con el ad\_registry.