



# Práctica 2

20/12/2022

—

Sebastian Pasker

Sistemas Distribuidos

## Introducción

Un sistema distribuido es un conjunto de computadoras que trabajan juntas para ofrecer servicios comunes. En esta memoria se presentarán tres aspectos importantes en el diseño y despliegue de sistemas distribuidos: APIs, encriptación de sockets y Kafka.

## Guía de despliegue

Para poder desplegar el proyecto lo que tendremos que realizar desde la terminal es lo siguiente:

```
sebas-p ~> # kafka_* = kafka y la versión
sebas-p ~> cp ./properties/server.properties
./bin/kafka_*/config/server.properties
sebas-p ~> cp ./properties/zookeeper.properties
./bin/kafka_*/config/zookeeper.properties
sebas-p ~> ./bin/kafka_*/bin/zookeeper-server-start.sh
./bin/kafka_*/config/zookeeper.properties &> ./logs/zookeeper.log &
sebas-p ~> ./bin/kafka_*/bin/kafka-server-start.sh
./bin/kafka_*/config/server.properties &> ./logs/server.log &
```

También se podrá correr:

```
sebas-p ~> pip3 install -r dependencies.txt
```

Para la base de datos:

```
sebas-p ~> python3 scripts/create_tabler_player.py # Para el engine
sebas-p ~> python3 scripts/create_weather_table.py # Para el weather
```

En diferentes terminales, corremos:

```
sebas-p ~> python3 AA_Register.py <config.json>
sebas-p ~> python3 AA_Weather.py <config.json>
sebas-p ~> python3 AA_Engine.py <config.json>
sebas-p ~> python3 AA_Player.py <config.json>
sebas-p ~> python3 AA_NPC.py <config.json>
sebas-p ~> node api/index.js
```

## Funcionamiento

### API Weather

Para la api de weather lo hemos implementado en el archivo de AA\_Engine.py en el que cogemos cuatro ciudades al azar de un json, llamamos cuatro veces la api de open weather con su temperatura y lo metemos en un diccionario general que tenemos de las ciudades con su temperatura. Esto se encuentra en las funciones de weather\_socket() y read\_cities\_json.

### API Rest Registry

Para las api REST hemos utilizado node js. En el que utilizamos express para crear la api. Hemos creado las funciones de GET, PUT, POST y DELETE para el registry en que se añade, edita, crea y elimina a un usuario. Esta api devuelve un json y se puede observar en el archivo api/registry.js y api/index.js. Lo hemos implementado desde el AA\_Player con las funciones create\_user\_api, edit\_user\_api, get\_user\_api y delete\_user\_api.

```
Creación de usuario por api:
Introduzca alias:
Juan
Contraseña:
Usuario creado con éxito.
```

```
[2022-12-22 08:34:30.458 PM] info:      Request to POST registry/ IP: ::ffff:127.0.0.1, Alias: Juan, Password: 123
[2022-12-22 08:34:30.460 PM] info:      Process SUCCESS. Request to POST registry /. IP: ::ffff:127.0.0.1, alias: Juan, password: 123
```

## Logs

Para los logs hemos utilizado winston para node js y logging para python. Para ello, dentro de la carpeta logs guardamos los mensaje tipo info, error y warning. Con el formato especificado de hora, tipo de error, quien lo produce, acción y variables. Esto se puede encontrar en sockets\_registry.log y también en api\_registry.log.

```
3 [2022-12-18 09:10:28.201 PM] error:    !! error 404 !! PUT registry/:alias. ip: undefined, alias: Adios, password: 321, ERR: Alias not found.
2 [2022-12-18 09:11:01.356 PM] error:    !! ERROR 404 !! In POST registry/, IP: ::ffff:127.0.0.1, alias: Sebas, ERR: User already exists.
1 [2022-12-18 09:11:29.314 PM] error:    !! error 404 !! DELETE registry/:alias. ip: undefined, alias: Hola, ERR: Alias not found.
49 [2022-12-22 08:34:30.393 PM] error:    !! ERROR 404 !! In GET registry/:alias, IP: ::ffff:127.0.0.1, usersPath: ../json/users.json, ERR: Alias not found.
```

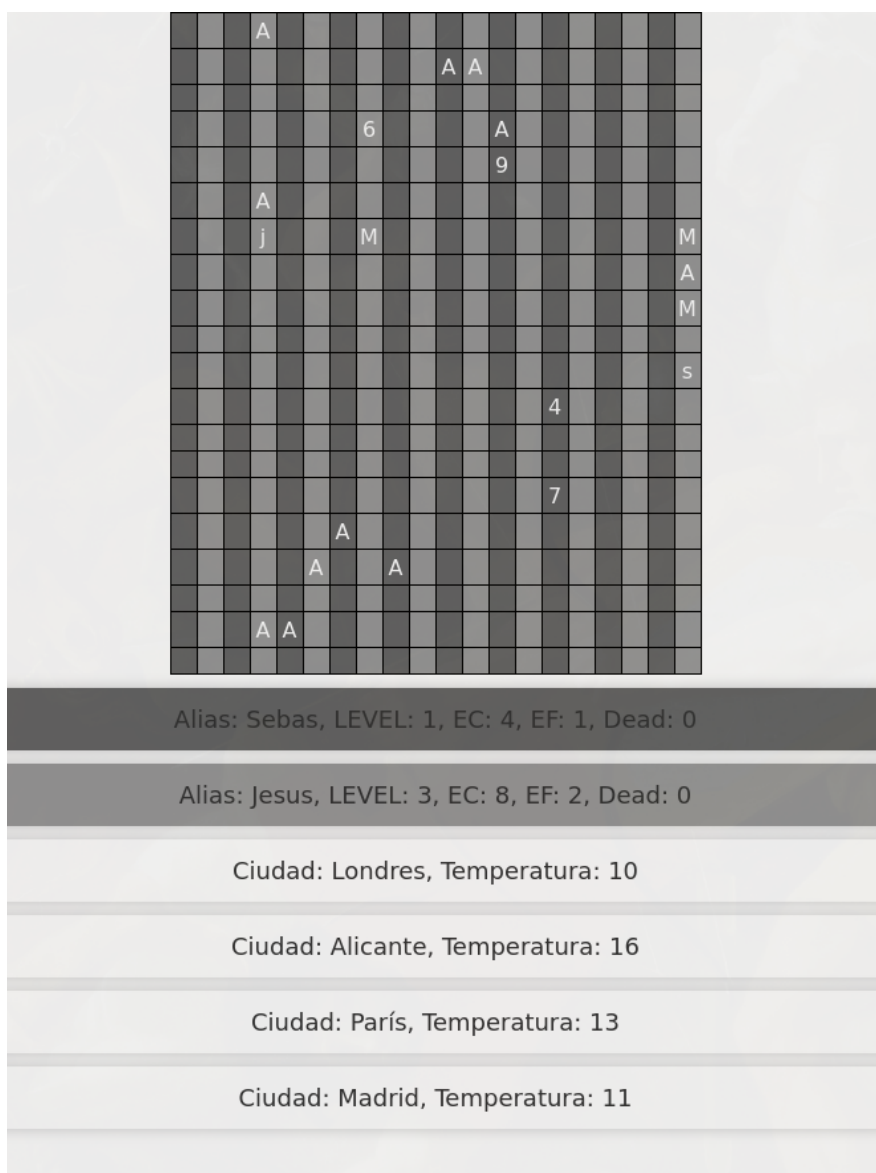
```
2022-12-19 21:00:42,365 - INFO - ('127.0.0.1', 42424) - Edición de perfil, alias: Sebas
1 2022-12-19 21:00:42,379 - INFO - ('127.0.0.1', 42424) - Perfil editado con éxito, alias: Sebas
2 2022-12-19 21:00:52,374 - INFO - ('127.0.0.1', 48426) - Edición de perfil, alias: Sebas
```

## API Engine

Para la api del engine, en node hemos creado bajo la dirección de map/ una api que devuelve un json del mapa, los usuario y el weather.

## Front end

Para el front utilizamos un node js para llamamos un html, en el llamamos con un XMLHttpRequest la api del engine map/. Vamos refrescando con setInterval() cada segundo llamando a la api. Los datos del mapa los metemos en una tabla y los jugadores con el weather lo ponemos debajo.



The screenshot displays a game interface. At the top is a 20x20 grid map. The map contains several characters represented by letters: 'A' appears in multiple locations, 'j' is at row 5, column 3, 'M' is at row 5, column 10, and 'S' is at row 10, column 19. Numbers are also present: '6' at row 3, column 10, '9' at row 4, column 13, '4' at row 10, column 15, and '7' at row 11, column 15. Below the map is a list of player statistics and city weather data.

Alias: Sebas, LEVEL: 1, EC: 4, EF: 1, Dead: 0
Alias: Jesus, LEVEL: 3, EC: 8, EF: 2, Dead: 0
Ciudad: Londres, Temperatura: 10
Ciudad: Alicante, Temperatura: 16
Ciudad: París, Temperatura: 13
Ciudad: Madrid, Temperatura: 11

## Seguridad

Para el tema de la seguridad de la web hemos utilizado https. Para ello hemos creado una clave pem y con node js simplemente con el propio módulo de *https* añadimos la clave y corremos el servidor.

Para la seguridad del kafka y los sockets, hemos utilizado un sistema asimétrico RSA. En el con sockets mandamos la clave pública del servidor al cliente y del cliente al servidor. Y a posteriori convertimos a bits el mensaje, lo encriptamos, enviamos y desencriptamos con la clave privada.

Además, hemos implementado un sistema de hash por md5, por lo que solamente se puede deducir la contraseña del usuario si se tiene la misma contraseña para comprobar. Mejor dicho, se puede verificar que la contraseña introducida es correcta.