UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster universitario en Ingeniería Informática

Proyecto de Servicios y Aplicaciones Distribuidas

Git Executor Service

Autores:
Iniesta Blasco, Gonzalo
Rodríguez Pérez, Raúl

Profesor:
Bernabeu Aubán, José Manuel

Arquitectura del servicio

El código del servicio se encuentra en el siguiente enlace de github. En dicho código, podemos observar el archivo docker-compose.yml, en donde se definen los componentes de la arquitectura del servicio. Estos componentes son:

1. **Zookeeper**

Kafka presenta una dependencia del proyecto **apache Zookeeper**. Este proyecto almacena metadata con el propósito de obtener persistencia en el conjunto de brokers que forman las réplicas de una partición de un topic, así como ser el encargado de la elección de una réplica líder (algoritmo de consenso) de entre dichos brokers para proporcionar escritura de sus datos y persistencia del subconjunto de nodos que se consideran elegibles para ser el siguiente líder (en caso de que el actual falle). Para este servicio hemos utilizado la imagen de apache Zookeeper: wurstmeister/zookeeper.

2. **Kafka**

Apache Kafka es un sistema de mensajería de publicación/suscripción descrito a menudo como un "registro de commit distribuido" o, más recientemente, como una "plataforma de distribución de streaming". Este componente junto con la funcionalidad de Zookeeper, nos proporcionará una **cola de eventos**, la cual será la encargada del aprovisionamiento de información, y de la forma de comunicación entre el frontend y los workers a través de los topics de dichos eventos. Para el servicio se crearán **2 topics**:

- **jobQueue**: cola donde se enviarán los trabajos junto con su información, y desde donde nuestro worker leerá y obtendrá dicha información para realizar sus funciones.
- **jobStatus**: cola donde el worker irá enviando el estado actual del trabajo, para que nuestro frontend pueda leer y actualizar dicho valor.

Para hacer posible la comunicación entre los contenedores implicados en el servicio (worker, frontend) con kafka, se han habilitado las **variables de los listeners** de Kafka, permitiendo dicha comunicación. Por último, para este servicio se ha utilizado la imagen de Kafka: wurstmeister/kafka.

3. **Frontend**

Una vez el componente de Keycloak nos ha dado autorización para utilizar el servicio, nos encontramos con el frontend. Dicho componente es el encargado de la **distribución de información entre el worker y el cliente** que realiza las peticiones a la API Rest. El frontend inicia un servidor con el uso del framework express (protegido por Keycloak), y define las siguiente funciones de una rest API:

- **app.post("/sendJob/")**: Función post que, en primer lugar, **decodifica el token** pasado para recaudar información útil como el username del cliente que está utilizando el servicio. Tras esto, **almacena la información** recopilada del body en un map (almacenamos dicha información en esta estructura de datos, ya que en posteriores funciones se harán consultas de búsqueda en dicho map, siendo esta estructura muy eficiente en este aspecto). Finalmente se **envía dicha información** a la

cola kafka con el topic “jobQueue”, y se envía un comprobante de la trabajo enviado al cliente. Sí este no fuera su primer trabajo, y hubieran trabajos enviados y finalizados previamente sin consultar, también se le enviará dicha información.

- **app.get("/status/:idJob")**: función get que dado el id de un trabajo, devuelve en formato JSON la **información de su estado actual**. Dicho estado puede ser: En cola, En proceso, Finalizado o el resultado del trabajo. El método comprueba la existencia del trabajo sin consumir excesivos recursos, incluso aunque dicho trabajo no se encuentre almacenado (se utilizan identificadores incrementales en hexadecimal para esto).
- **app.get("/result/:idJob")**: función que dado el id de un trabajo, devuelve el resultado del mismo. Sí el trabajo no ha finalizado devuelve un error 400.
- **app.get("/showSendJobs")**: función que devuelve todos los trabajos enviados por un usuario. Conseguimos el nombre del cliente gracias al token de autorización que decodificamos.

Finalmente quiero destacar que utilizamos, tanto para el worker como el frontend, la librería node-rdkafka. Esta librería es la que nos permite realizar la **comunicación y la escritura de los eventos** en los topics de la cola Kafka implicados. Hemos elegido esta librería ya que como se puede apreciar en el código, permite realizar la implementación de las funcionalidades dichas de una manera muy **simple y con pocas líneas de código**. El funcionamiento de todos los apartados donde utilizamos la librería es simple, está obliga al worker y al frontend a que estén permanentemente escuchando en la cola asignada, hasta que una vez alguien escriba, se reciba dicha información y se realice la funcionalidad deseada. La imagen empleada para el worker se basa en: raulrguez09/producer_sad_v4.

4. Worker

El worker representa el componente donde se realiza la funcionalidad del trabajo. Esta consiste en leer la información del trabajo de la cola, la cual contiene todo lo necesario para, **clonar un repositorio** github (público) que pasamos como parámetros, y **ejecutar el archivo** indicado en el body de la petición (sólo se ejecutarán archivos escritos en python), pudiendo incluso **añadirle parámetros o instalar dependencias** (dependencias del lenguaje python) para su correcta ejecución. Una vez ejecutado el archivo, se almacena y envía el resultado a la cola, donde el frontend será el encargado de hacerlo llegar al cliente. El último paso que realiza el worker siempre es la eliminación del repositorio clonado.La imagen empleada para el worker se basa en: raulrguez09/worker_sad_v4.

5. Postgres

Postgre es una base de datos relacional de código abierto, está es utilizada para **almacenar los datos** de la sesión admin de keycloak. Es decir, una vez hemos configurado Keycloak, creando el realm, el cliente, el usuario y asociando el rol adecuado al usuario creado, toda esta información se almacenará en la base de datos, permitiendo que sí paramos la orden docker-compose up, podamos volver a ejecutarla sin tener que volver a poner dicha información de configuración.

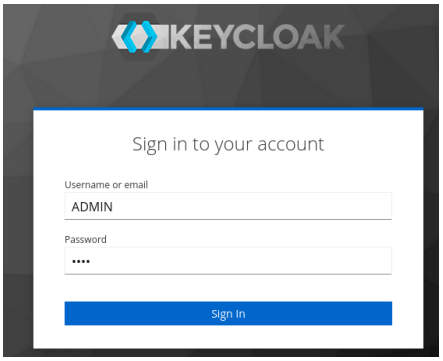
6. **Keycloak**

Keycloak es un componente que nos proporciona la **gestión de acceso e identificación** de nuestro servicio. En nuestro servicio, a parte de su llamada en el fichero de docker-compose.yml, este componente tiene su propio fichero de configuración (keycloak_config.js), en donde proporcionamos funciones para inicializar el servicio de keycloak con la información de las variables de entorno que hemos establecido en el docker compose. Esta inicialización la llamaremos desde nuestro frontend, el cual, una vez inicializada la sesión, y cargada la configuración de keycloak, este nos permitirá **restricción de acceso a las peticiones** de la API rest, únicamente a los usuarios que posean el access token necesario. Es decir, gracias a la configuración necesaria que debemos realizar en la consola de administración de keycloak (crear realm, cliente, usuario, role, etc), podemos realizar una petición donde nos devuelva un access token, el cual, colocándolo en el apartado de "Authorization" a la hora de hacer las peticiones a la API, nos entregará acceso al servicio. Finalmente, la imagen usada de este componente es quay.io/keycloak/keycloak:20.0.3.

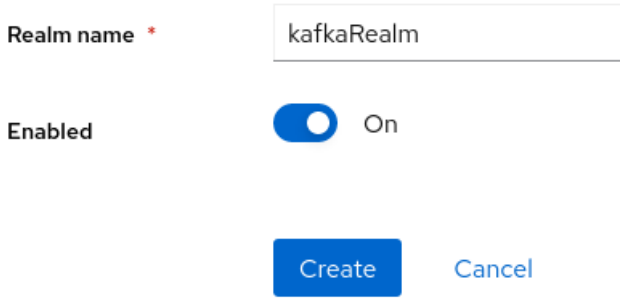
Funcionamiento del servicio

Para hacer uso del servicio, es necesario realizar los siguientes pasos:

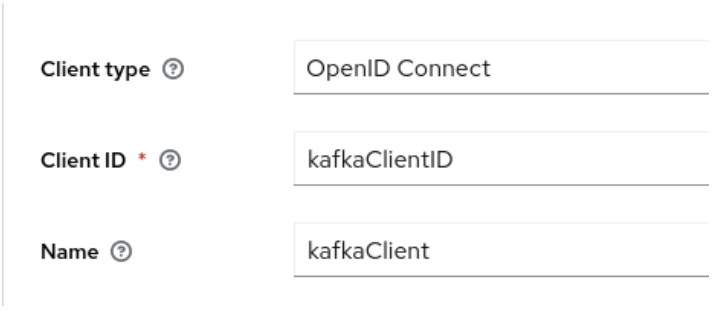
1. Descargar el repositorio del servicio que se encuentra en el siguiente enlace de github
    a. A partir de aquí se explica la última versión que posee dicho repositorio, pero también es posible descargarse las versiones anteriores, como por ejemplo la versión sin emplear el componente Keycloak (esto es un inciso pero ambas son completamente funcionales).
    b. También se encuentra disponible el repositorio de pruebas para el worker, el cual es el repositorio que he empleado para testear toda la funcionalidad del mismo. El enlace del repositorio lo puede visitar aquí.
2. Tras descargar el servicio, ejecutamos la orden: docker-compose up
    a. Esto provocará el despliegue de todos los componentes que hemos analizado en el apartado anterior
3. Sí es la primera vez que utiliza el servicio debemos realizar la configuración de Keycloak
    a. En primer lugar diríjase a la dirección https:/localhost/8080 y en la consola de administración utilice las credenciales que aparecen en el docker-compose



b. Tras esto creamos el realm de Kafka haciendo click en la parte superior izda

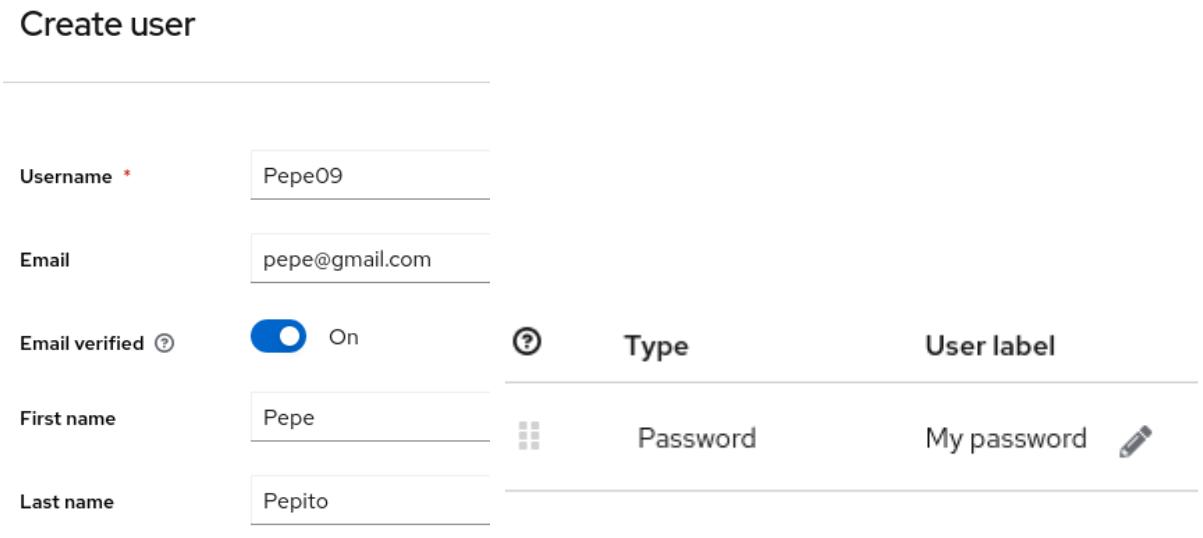


c. Creamos un nuevo cliente

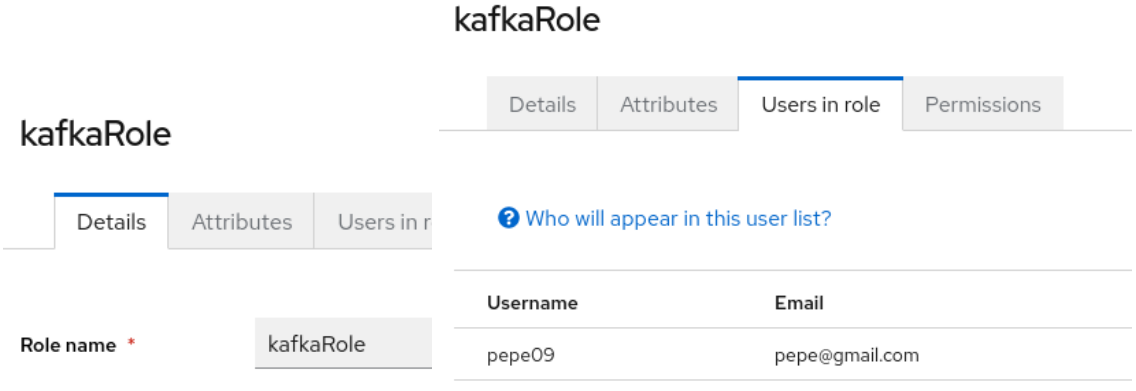


d. Copiamos el client secret del cliente y lo guardamos

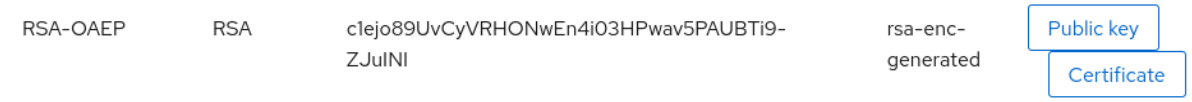
e. Creamos user y establecemos una password



f. Creamos y Asociamos un rol al usuario creado

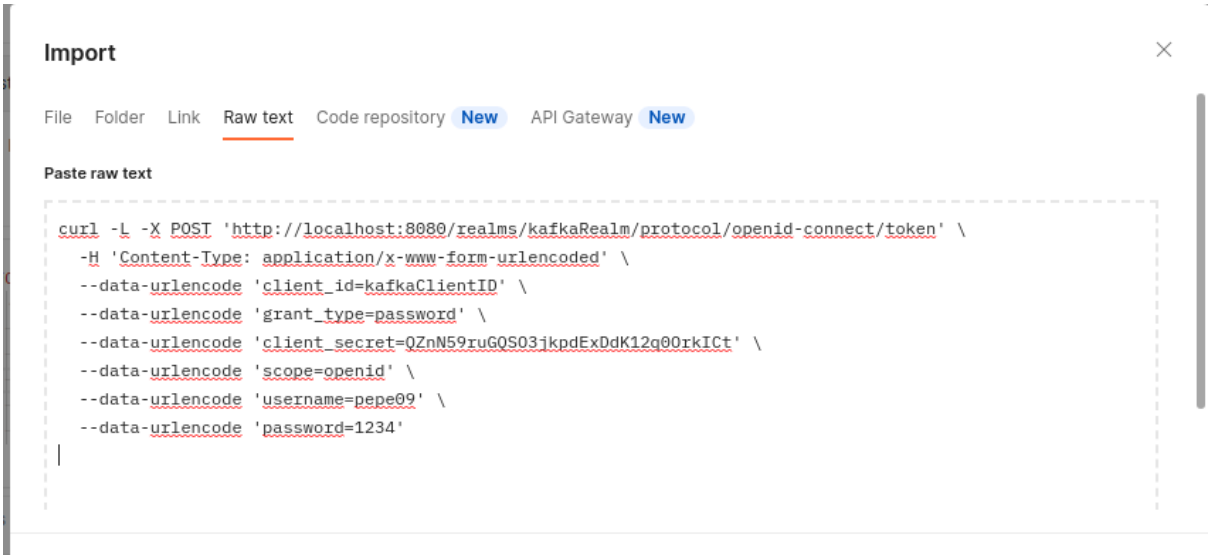


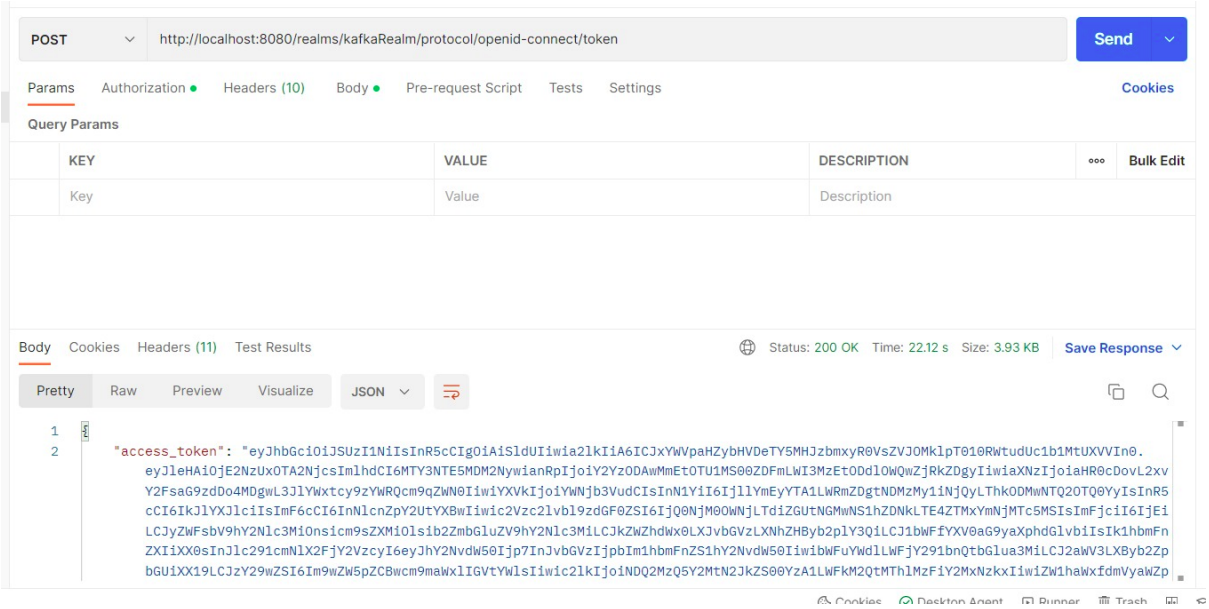
g. Copiamos el realm public key y lo copiamos en el docker-compose



h. Copiamos el token endpoint

i. Con toda la información recopilada en los apartados anteriores, elaboramos el siguiente curl, el cual sí ejecutamos nos proporcionará el access token necesario para utilizar el servicio





j. Finalmente colocamos al access token en el apartado de Authorization, y enviamos la petición al localhost:3000/sendJob. El body que debemos poner tienes el aspecto que vemos en la captura:

localhost:3000/sendJob