Máster Universitario en Tecnologías Web,
Computación en la Nube y
Aplicaciones Móviles

Asignatura:

Desarrollo basado en componentes distribuidos y servicios

29 de mayo de 2023

Trabajo Final

Andoni Salcedo Navarro

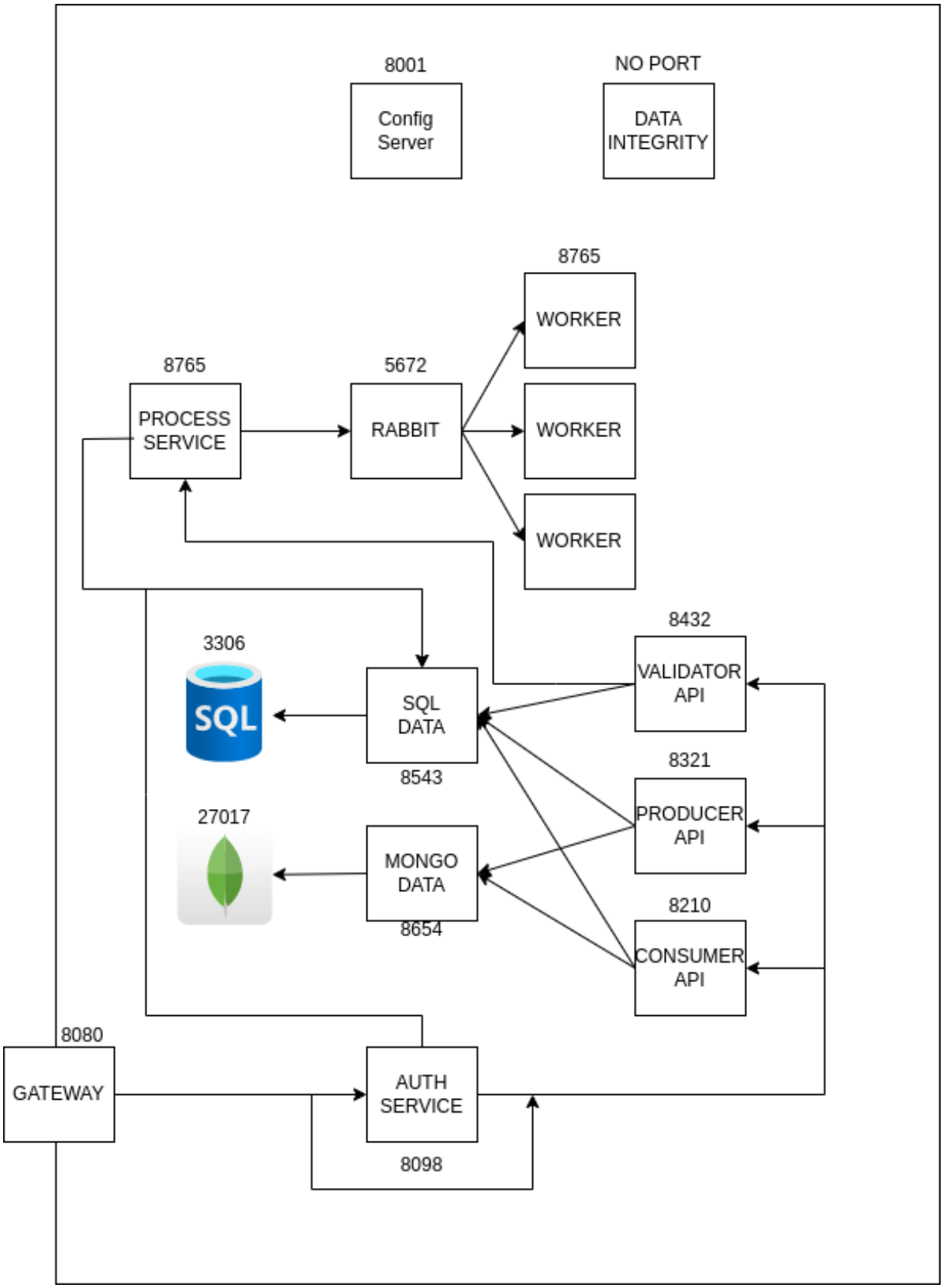

Figura 1: Diagrama de componentes de la app

Descripción de los componentes

A continuación se realiza una breve introducción a la función que realiza cada componente en el sistema:

- Gateway: es el punto de entrada de la aplicación donde se gestiona la seguridad de la aplicación, por defecto los endpoints de la aplicación no requieren estar loggeado, en este punto es donde se le añade esta seguridad, a partir de las rutas de la aplicación en el fichero config se decide si requiere autenticación y el **rol** que debe tener el usuario autenticado para acceder a dicho endpoint
- Auth Service: Es el componente donde están las rutas de login y autenticación interactúa con el componente de acceso a la base de datos sql para acceder al usuario y comprobar si existe en la base de datos y el rol que tiene.
- Consumer API: implementa los endpoints de los requisitos de la api de consumidores
- Producer API: implementa los endpoints de los requisitos de la api de productores
- Validator API: implementa los endpoints de los requisitos de la api de validadores
- Mongo data: elemento de conexión con la base de datos no relacional mongodb contiene la logica de almacenar, ficheros leer ficheros, etc.
- SQL data: elemento de conexión con la base de datos relacional contiene las operaciones CRUD para productores, validadores, usuarios y ficheros además de los repositorios de datos que implementan las consultas a la base de datos.
- Process Service: componente que gestiona el proceso de validación de un fichero, es el que se conecta con rabbitmq y manda y recibe el evento de validar fichero y fichero validado.
- Worker: es el encargado de simular la validación del fichero. Según el tamaño del fichero es el tiempo que está validando. Para simular carga durante ese tiempo se realizan operaciones aritméticas complejas.
- Config Server: es el que guarda la configuración de cada componente de la aplicación urls, puertos, etc.
- Data integrity: componente que se encarga de la integridad de los datos, al existir datos en bases de datos distinta este componente está hecho para ejecutarse periódicamente comprobando que los datos en una base y los de la otra mantienen su integridad.

Decisiones de diseño

A continuación se explica las decisiones de diseño tanto arquitecturales como de implementación que se han realizado en el sistema:

- Separación de la lógica del proceso de validación en un componente aparte

  Se ha decidido separar las rutas relacionadas con el proceso de validación de ficheros a un componente aparte dado que este se tiene que conectar a un servicio rabbitmq por lo que en caso de fallar este componente el resto de endopints de la api de validación seguirían disponibles y solo caería la parte de validación de ficheros. Además, la lógica es suficientemente distinta como para posicionarlo en un componente aparte.

- Worker no conecta directamente al componente de dato.

  El worker es un elemento que solo se debe encargar de validar el fichero por lo que no debería tener en consideración una conexión directa con el componente de datos por ello lo que se ha decidido es que cuando acabe de validar el fichero mande un evento al componente process service que es el que tiene la lógica de esta parte y es donde se implementa que hacer cuando se ha terminado de validar el fichero. De esta forma si se debe cambiar que hacer cuando se acaba de validar es el componente process service el que se tiene que cambiar.

- Securización centralizada

  Se ha decidido utilizar el componente gateway para centralizar la seguridad de la aplicación de esta forma internamente en el sistema se puede acceder a cualquier endpoint sin estar autenticado. Esta seguridad centralizada permite de una manera sencilla controlar que rutas requieres y cuáles no sin que haga falta que cada api rest internamente controle la autenticación, esto hace que el proyecto sea más mantenible y si se necesitan realizar cambios de control en las rutas se realice desde la propia configuración del sistema y no desde un componente que se debería recompilar a posteriori.

  Además, en este punto se ha añadido control de roles al componente gateway, especificando que roles pueden acceder en este caso a cada ruta. Esto se ha hecho también desde la propia configuración. Además se pasa el id del usuario que se ha validad por los headers de esta forma si se requiere su uso en los endpoints está accesible.

  Se ha añadido lógica al filtro para que compruebe el tipo de usuario que se ha intentado autenticar y compare con una lista de usuarios a los que se les permiten acceder a esa ruta, si está entre esos usuarios entonces deja hacer la petición. Para ello hay que implementar la clase de configuración con una lista en este caso la lista de roles que será accesible cuando llegue el filtro de autenticación. Para securizar la aplicación desde las properties sería de la siguiente forma:

```
# Todo lo que vaya a /validator se securiza
spring.cloud.gateway.routes[0].id=${validator.name}
spring.cloud.gateway.routes[0].uri=${url.validator.api}
spring.cloud.gateway.routes[0].predicates[0]=Path=${validator.path}/**
# Se añade para que solo los validadores puedan acceder a estas rutas
spring.cloud.gateway.routes[0].filters[0]=AuthFilter=Validator

# Todo lo que vaya a /consumer se deja paso
spring.cloud.gateway.routes[1].id=${consumer.name}
spring.cloud.gateway.routes[1].uri=${url.consumer.api}
spring.cloud.gateway.routes[1].predicates[0]=Path=${consumer.path}/**


# Todo lo que vaya a /producer se deja paso
spring.cloud.gateway.routes[2].id=${producer.name}-public
spring.cloud.gateway.routes[2].uri=${url.producer.api}
spring.cloud.gateway.routes[2].predicates[0]=Path=${producer.path}


# Todo lo que vaya a /producer/* se securiza
spring.cloud.gateway.routes[3].id=${producer.name}-private
spring.cloud.gateway.routes[3].uri=${url.producer.api}
spring.cloud.gateway.routes[3].predicates[0]=Path=${producer.path}/**
# Se añade para que solo los productores puedan acceder a estas rutas
spring.cloud.gateway.routes[3].filters[0]=AuthFilter=Producer
```

- Por defecto la aplicación está en modo producción

  Se han gestionado los perfiles de ejecución de la aplicación para que por defecto la aplicación se lance en modo de producción por ello si se quiere ejecutar la aplicación en local no se debe cambiar a ningún perfil. Además, se han creado unos scripts para lanzar los recursos de la base de datos, mongo y rabbitmq desde Docker en el puerto que se ha configurado en el sistema.

- Se han seguido algunos principios solid

  Por ejemplo se ha utilizado el principio de Sustitución de Liskov en los servicios de las apis que conectan con la base de datos, dado que para la finalidad de la aplicación se ha requerido utilizar un componente externo para almacenar los datos, pero si se tratase de una aplicación en un entorno realista tal vez se puede cambiar donde se han de almacenar los datos.

- Se ha utilizado el componente @EventListener de spring

  Con la finalidad de la creación de un componente que compruebe la integridad de los datos periódicamente. Se ha creado una aplicación de spring que lo que hace es comprobar si los datos de una base de datos y la otra son los mismos, para ello su única función es que cuando se arranque la aplicación se realice esta comprobación y luego se pare hasta el siguiente periodo. Se ha decidido utilizar este componente para que cuando se lance la aplicación realice el chequeo de la integridad de los datos.