Pontificia Universidad Católica de Chile
Escuela de Ingeniería
IIC2173 - Arquitectura de Sistemas de Software

Documentación

Entrega 1

Grupo 24

Felipe Carrasco
María Ignacia Gallardo
Catalina Musalem
Diego Nazal
Sebastian Mitjans

1. Introducción

Para esta entrega se nos pidió realizar la arquitectura de una plataforma de alertas con un sistema de usuarios y con un sistema de "Workers"que sean capaces de procesar información de forma independiente.

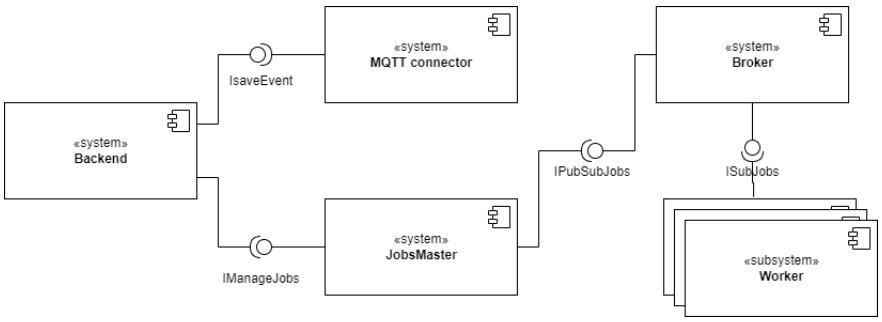

Figura 1: Diagrama de referencia, del enunciado de la entrega [1]

Como podemos ver en este diagrama referencial, las información de las alertas llegan a través de un broker administrado por un tercero. Luego, la lógica interna de la arquitectura se encarga de delegar el trabajo de procesamiento a una unidad de control que sigue el patrón de diseño "Master/Slave", en la que cada componente Worker realiza su tarea de forma asíncrona con el funcionamiento principal del programa.

[1] https://cursos.canvas.uc.cl/courses/46688/files/6599884

2. Diagrama UML

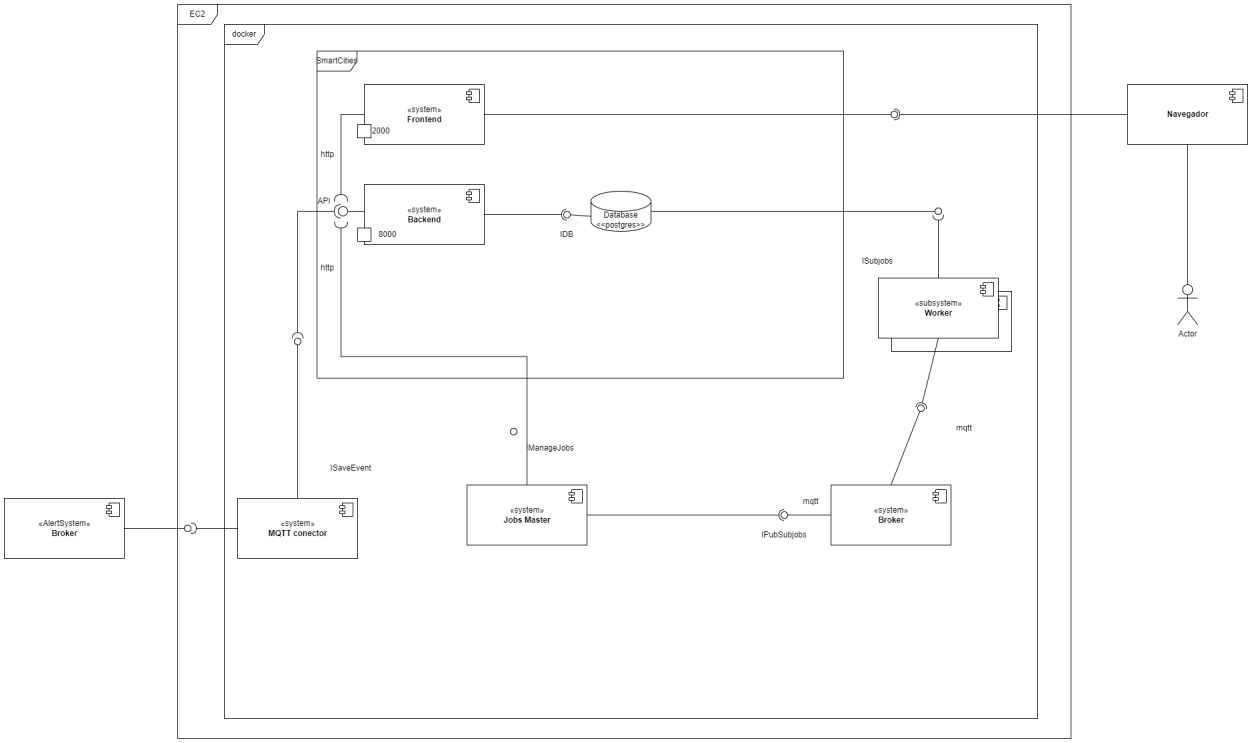

Figura 2: Diagrama UML

Nuestra Arquitectura consiste de tres partes fundamentales. El usuario interactua con el sistema a través del Frontend. El Backend se encarga de recibir las alertas del agente externo y de comunicar las distinta partes del sistema. El Job Master se encarga de recibir las solicitudes del Backend y delegar las tareas de procesamiento a los Workers.

Además, la base de datos guarda los datos de cada alerta según las instrucciones del Backend y actualiza los estados de dificultad de cada alerta luego de su cálculo por parte de los Workers.

Los directorios de nuestro repositorio son los siguientes:

- Frontend: FrontEnd
- Backend: SmartCities
- MQTT connector: broker
- Job Master: complejidad

3. CI

Para el proceso de Continuous Integration de nuestro proyecto usamos CirleCI. Cada vez que se sube un nuevo commit a un repositorio se corren los tests ubicados en el directorio /SmartCities/-tests y se actualiza el estado que se puede ver como un símbolo a la izquierda del nombre del commit.

Un check verde significa que se aprobaron los tests, una equis roja significa que no se aprobaron y un punto naranjo significa que se están corriendo. Si no se encuentra el archivo de configuración en el directorio /.circleci los tests fallarán.

Para replicar el pipeline, basta con apretar el símbolo a la izquierda del id del commit y luego apretar en "Detalles", lo cual nos redirigirá a la página de CircleCI donde podemos ver los detalles del test ejecutado y repetir el proceso de testeo si así lo deseamos.

4. Ambiente Local

Para correr la arquitectura de forma local, podemos usar docker-compose.

Para esto, necesitamos tener docker instalado y abrir una consola en la raíz del repositorio. Luego, podemos correr `sudo docker compose build` y `sudo docker compose up` para levantar los contenedores del sistema.

Luego de que los contenedores estén corriendo, es necesario correr las migraciones del contenedor SmartCities (con el nombre de web). Para esto, se debe correr el comando `pyhon manage.py migrate` en la consola de docker del contenedor respectivo.

Con respecto a los endpoints de la API complejidad se recomendaba que fuese `/job` el url para obtener la complejidad pero se ocupo `/complejidad`. Tienen la misma funcion pero utilizamos nombres distintos.

Los Url funcionales de la app son `/main` , `/main/signup` , `/main/login` y los Url de la API implementados son `/complejidad/:id` que entrega la complejidad de la alerta con ese id.

Se puede hacer un post a traves de `/complejidad` para recibir las alertas y tambien un GET a traves de `/heartbeat` que retorna True