PROYECTO II – Emotion Analysis in SOA

Mariano Muñoz Masís, Luis Daniel Prieto Sibaja
mmunoz@estudiantec.cr,prieto.luisdaniel@estudiantec.cr
Área Académica Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

I. Requerimientos Generales

El departamento de recursos humanos de su empresa, está interesado en construir un sistema de software que le permita analizar los sentimientos de los empleados de la empresa al recibir buenas noticias. La idea es eventualmente utilizar la herramienta para comunicar mejor los resultados mensuales de ganancias de la compañía.

Al final de cada día laboral, 10 imágenes de empleados deben ser analizadas, y el resumen del análisis debe ser agregado a un archivo local común, el cual se comparte entre los servicios, y es donde se guardan todos los estados.

El sistema debe correr en forma de contenedores, conteniendo un servicio cada uno (al menos cuatro servicios deben existir en la aplicación, por lo que se aconseja revisar los principios SOLID) y los mensajes entre ellos deben ser asíncronos, se aconseja utilizar un contenedor corriendo como Service Broker.

El sistema puede manejarse como contenedores individuales corriendo en docker en la computadora del desarrollador. Sin embargo, se aconseja utilizar algún orquestador de contenedores como Minikube o una versión local de Kubernetes.

El servicio que identifica la emoción o sentimiento de las imágenes puede ser tanto un servicio local corriendo en un contenedor, como un servicio de la nube utilizado as a service

II. Arquitectura Orientada a Servicios

Después de analizar los requerimientos del sistema, se desarrolla el diagrama presentado en la Figura 1, en el cual se distinguen 4 servicios básicos para el funcionamiento del programa y sus respectivas sub-funcionalidades, después de un análisis más detallado y con el paso del proyecto se simplifica a una arquitectura como la que se aprecia en la Figura 2.

Estos diagramas son la base para formar la estructura del proyecto, y asemejan el comportamiento de los módulos desarrollados. Los cuatro módulos siguen el principio de responsabilidad única, de manera que estos puedan ser instanciados de manera individual y que su ejecución sea específica para la tarea que se le asignó. De la misma manera, al ser servicios tan fundamentales y básicos, presentan la característica que pueden tener cierta escalabilidad, pero no pueden perder su funcionalidad básica dado que perdería la integridad del módulo. Así mismo, dada su modularidad e independencia entre servicios, estos podrían ser fácilmente reemplazados por versiones más eficientes o con rendimientos mejores. Dados los requerimientos específicos con los que se desarrollaron los servicios, las interfaces entre ellos son únicas y específicas para cada comunicación. Por último, no se puede asegurar el principio de Inversión de Dependencias, dado la dependencia a APIs externas.

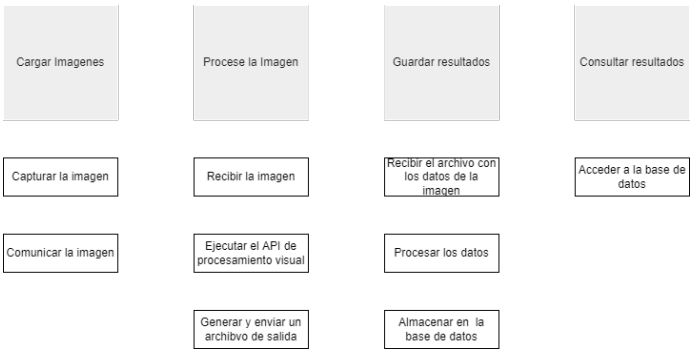

Fig. 1. Diagrama de Arquitectura Orientada a Servicios - Diagrama Inicial

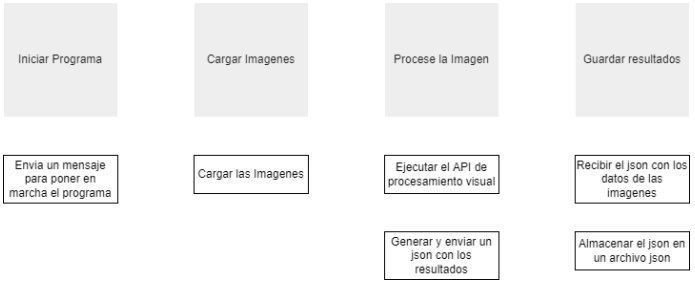

Fig. 2. Diagrama de Arquitectura Orientada a Servicios - Diagrama Final

III. Funcionalidad

Para realizar cada uno de los módulos o servicios presentes en la Figura 2 se utilizaron dos herramientas, la primera de ellas es Docker, que nos ayuda a crear instancias o contenedores en los cuales podamos ejecutar código de manera independiente y así, tener un control más cercano a servicios corriendo en localidades distintas y la segunda herramienta es RabbitMQ, el cual se utilizó para manejar el comportamiento y comunicación de dichos computadores. De igual manera, RabbitMQ se utilizó a través de la plataforma de Docker y del navegador.

- Iniciar Programa - Marca el inicio de la ejecución del sistema, de manera que solo cuando este módulo se ejecuta

los demás tienen la posibilidad de ser ejecutados, este módulo es un *publisher* de la herramienta de RabbitMQ.
- Cargar Imágenes - Este módulo es un *consumer* que realiza algunas funciones de *publisher*, dado que está a la espera que en la cola de mensajes se despliegue el mensaje *start* para iniciar a cargar las imágenes, una vez que se recibe dicho mensaje, este módulo publica un mensaje de éxito, que da paso al siguiente módulo.
- Procesar la Imagen - Este módulo es similar al anterior ante el comportamiento del flujo, se está a la espera del mensaje de éxito del módulo anterior para iniciar a cargar y analizar las imágenes, de manera que se consume el servicio de reconocimiento facial brindado por Google.
- Guardar resultados - Este módulo se encarga de guardar el resultado del análisis facial a un archivo JSON.

IV. Diagrama de Arquitectura - Orientada a Eventos

Para lograr la funcionalidad de los contenedores, se llegó a la conclusión debían tanto publicar, como consumir mensajes de la cola de Rabbit, por ello como se puede apreciar en la Figura 3, tanto autorizar el procesamiento, como el procesamiento en sí, consumen y producen mensajes que se guardan en el *message broker*

El flujo del programa comienza con el módulo que inicia el programa, el cual publica el mensaje que *start*, el módulo de autorización consume este mensaje y a su vez publica uno nuevo para dar paso al módulo que procesa las imágenes, este publica a la cola el resultado obtenido como un objeto Json y el último módulo consume este mensaje y lo convierte a un documento Json que es almacenado de manera local.

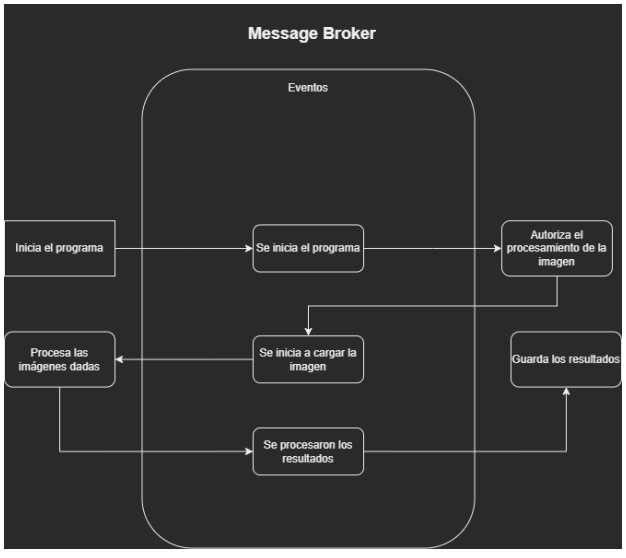

Fig. 3. Diagrama de Arquitectura Orientada a Objetos

Cada componente mostrado en la Figura 4 es en realidad un contenedor de Docker, que se ejecuta de manera independiente, estos contenedores facilitan la implementación de aplicaciones sobre la misma máquina, pero con entornos exclusivos, tanto así que cada contenedor tiene sus propias variables de ambiente.

El sistema desarrollado tiene 5 módulos, 4 corresponden a los servicios, 1 corresponde al *message broker* y también contiene una ”base de datos” representada por el archivo Json.

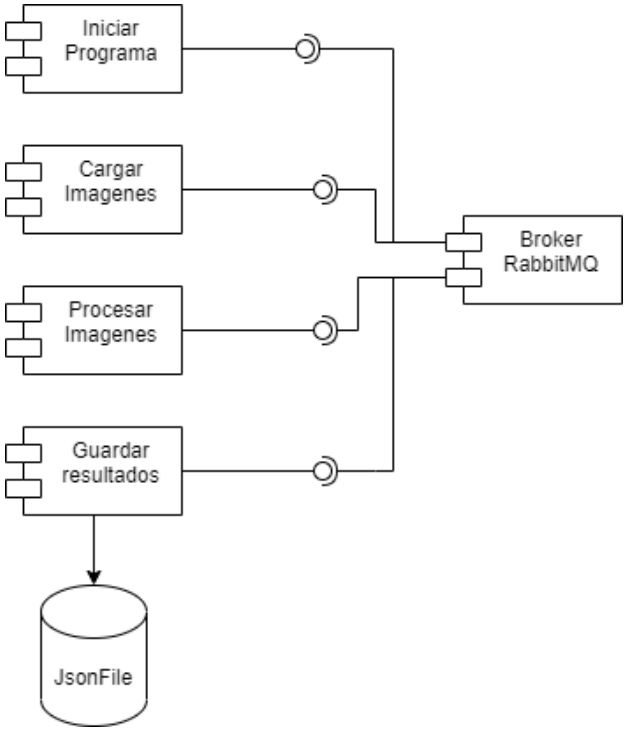

Fig. 4. Diagrama de Arquitectura: Vista de Componentes

References

[1] Marcel Dempers (agosto 5,2020) *RabbitMQ : Message Queues for beginners*.[Youtube] That DevOps Guy https://www.youtube.com/watch?v=hfUIWe1tK8E&ab_channel=ThatDevOpsGuy
[2] RabbitMQ (s.f.) *RabbitMQ : Python*.[Online] RabbitMQ https://www.rabbitmq.com/tutorials/tutorial-one-python.html
[3] Docker(s.f.) *Docker overview*.[Online] Docker Docs. https://docs.docker.com/get-started/overview/