Área Académica de Ingeniería en Computadores

Curso: CE5508

Arquitectura Orientada a Servicios

Proyecto 2

Realizado por:

Mario Araya Chacón 2018319178
Fabian Ramírez Arrieta 2018099536
Mariana Vargas Ramírez 2018086985

Profesor:

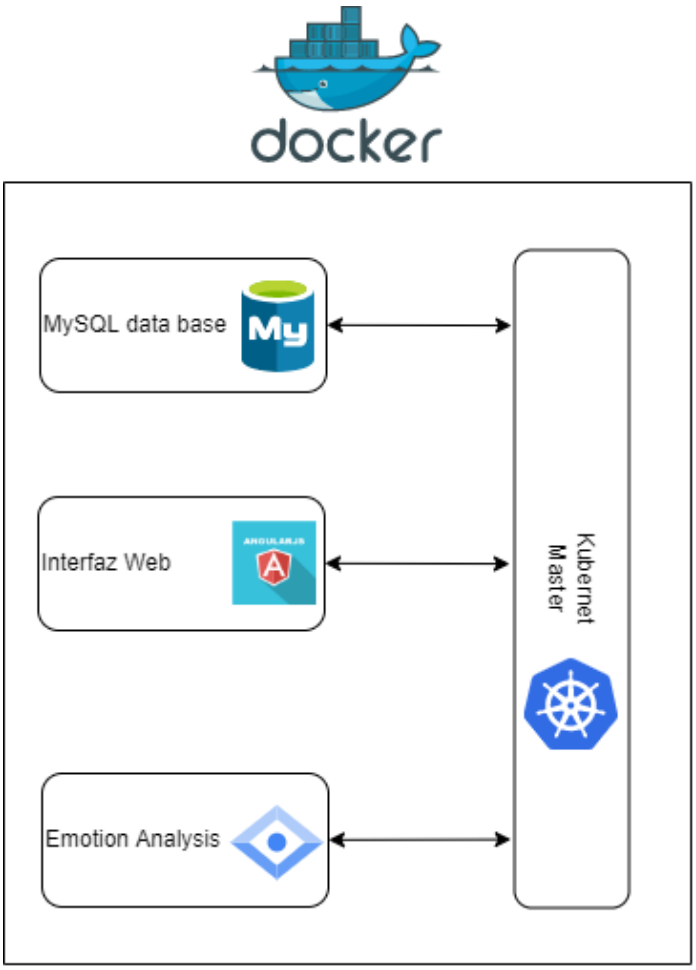
Alejandra Bolaños Murillo

Fecha: Cartago, octubre 18, 2022

índice

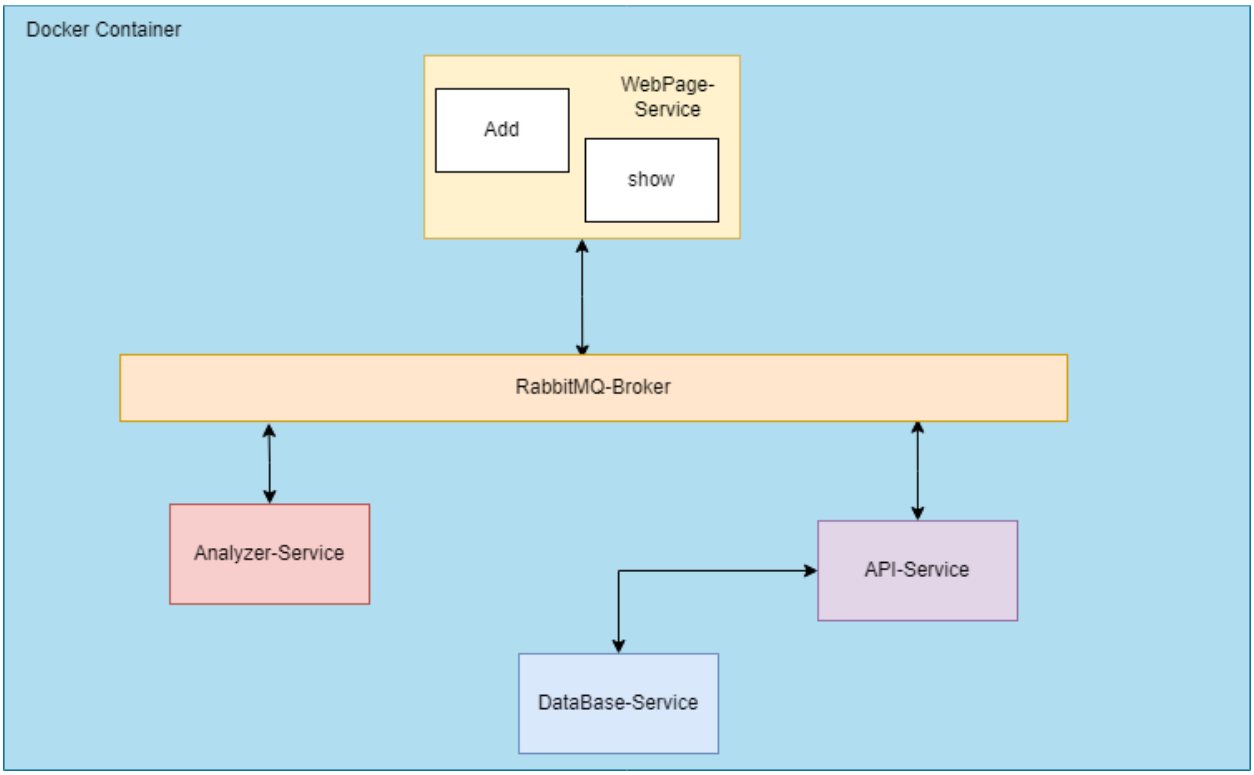
Arquitectura orientada a servicios

Diagrama Prescriptivo



En el diagrama prescriptivo se puede apreciar el diseño inicial que se tenía para el proyecto, donde se contaba con 4 “pods” con una función específica cada una, donde la base de datos en MySQL se encarga de almacenar las imágenes y los resultados del análisis de estos una vez que son procesadas por el “emotion analysis”, para finalmente desplegar estos datos en la pantalla del usuario por medio de una interfaz realizada con angular. Sin embargo, para toda esta comunicación entre los distintos servicios, se planeaba utilizar un Master o mediador, encargado de comunicar a todos los servicios según se necesitara. Sin embargo, para el diseño final se cambió la aproximación de un mediador a un “service brooker”.

Diagrama Descriptivo



En el diagrama anterior se muestra la arquitectura de servicios que es utilizada por la aplicación desarrollada. Como se puede observar cuenta con 4 servicios principales el primero es la página web la cual se encarga de la interacción con el cliente tanto de tomar las fotos ingresadas como mostrar las imágenes que se encuentran en la base de datos, por lo que se mencionara el segundo servicio el cual es la base de datos; en esta se puede almacenar los datos como las imágenes, fechas, nombres y emociones de las diferentes imágenes.

De igual forma se encuentra el API-services que se encuentra de transformar los datos que se encuentran en la cola de rabbit y lo traspasa a un formato para poder ser enviado a la base de datos. Por último, se encuentra analyzer-srvice el cual es un servicio que se va a encargar de analizar las emociones de la persona que se encuentra en la imagen con ayuda del API visión de Google.

Finalmente se utiliza un servicio de broker, en este caso se decidió utilizar Rabbit esto debido a que se comporta de una manera intuitiva con los diferentes servicios que se utilizaron. Aparte que con la utilización de json es de fácil acceso.

Principios de SOLID

Single Responsibility

Este principio se basa en que cada servicio debe de tener una única tarea para separar el comportamiento y debuggear más fácilmente. Por lo que es uno de los principios implementados en el proyecto a continuación se muestra como es implementado este principio en cada uno de los servicios.

- WebPage: se encarga únicamente de la interacción con el usuario.
- Analyzer: encargado únicamente de analizar la emoción de la persona por una imagen.
- DataBase: cumple la función de almacenar los datos de las imágenes y las imágenes.
- Api: su tarea es comunicar la base de datos con el broker y otros servicios.
- Broker: encargado de la comunicación de mensajes de manera asincrónica.

Open/closed

El objetivo de este principio es poder extender el comportamiento de los servicios o clases, sin necesidad de cambiar la clase para evitar crear errores. Para esto cualquier servicio se le puede agregar una función adicional siempre y cuando sea compatible con el lenguaje, aparte que si se desea crear una función general al sistema puede realizarse creando un nuevo servicio el cual se implementa de una manera sencilla y sin necesidad de modificar alguno de los otros servicios ya existentes.

Liskov Substituition

Este principio busca que siempre se cumpla la función que se solicita y no se pierda una tarea debido a que una clase deje de funcionar o sea eliminada, por lo que se desea que otra tome el cargo si algo llegara a pasar y realice esas tareas. En el proyecto realizado se puede decir que este principio se cumple en cierta medida ya que cuando se crean los servicios se puede decidir si realizar diferentes réplicas del mismo servicio, quizás no sea específicamente lo que busca el principio, pero con estas replicas se puede crear diferentes pods donde si uno falla el otro puede tomar el lugar.

Interface Segretation

El objetivo de este principio es dividir un conjunto de acciones en conjuntos más pequeños así la clase solo ejecuta lo necesario para la realización de la tarea. Para este principio se puede decir que todo el proyecto está basado en esto, debido a que al realizar diferentes servicios se divide y solo son utilizados los necesarios para la ejecución de la tarea, por ejemplo, a la hora de desplegar las fotos solo es necesario el api, la página web y la base de datos. Igualmente, cada servicio y pagina tienen sus propias importaciones ya que no todas necesitan de ellas.

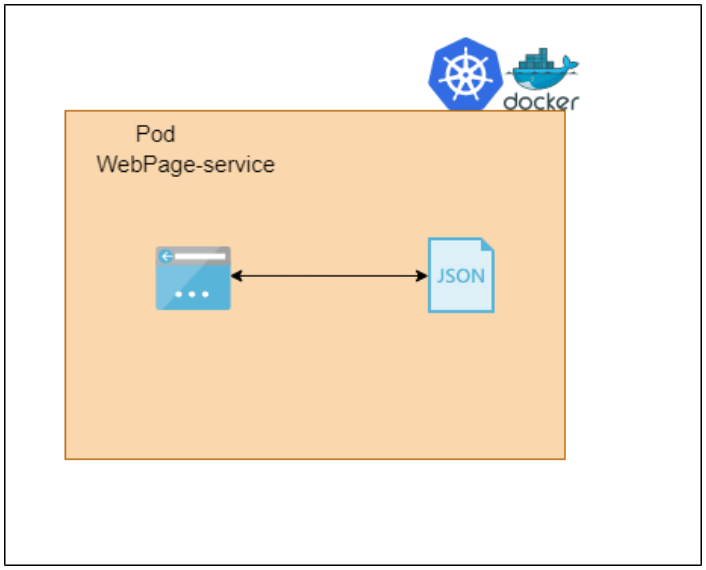
Dependecy Inversion

Por último, se busca que cada clase o servicio puedan realizar la tarea correspondiente incluso si le falta alguna de sus partes importantes que no dependa de ninguna abstracción. Las clases principales de cada servicio no deben depender de otras. Para este principio se buscó la forma de que no dependan un servicio con otro, sin embargo, esto no es posible ya que para que toda la función haga su tarea se necesitan de todos los servicios, pero cada servicio tiene su propia funcionalidad y puede realizar su propia tarea sin depender del resto.

Funcionalidad de los servicios

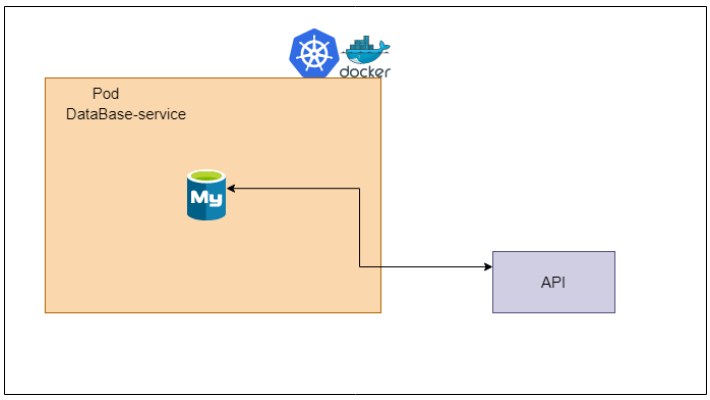
Página web

La funcionalidad principal de la página web es la interacción del sistema con el usuario, de una manera fácil y agradable para el cliente es por esto por lo que se utiliza angular para un mejor desarrollo de esta. Por otro lado, la forma de comunicarse con los otros servicios es realizando un file json que se le envía al api para que lo distribuya. Como bien se ha mencionado el proyecto es realizado en kubernetes por lo que este se encuentra dentro de un pod de kubernetes que se encuentra dentro de un container de Docker.



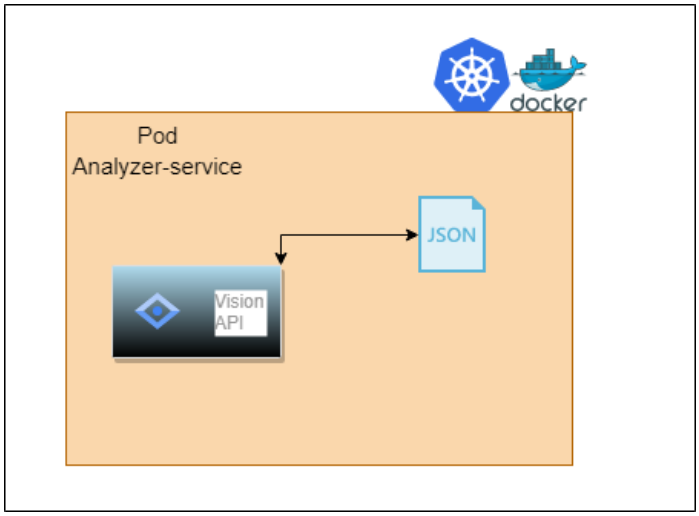
Base de datos

Al igual que el resto de los servicios es un pod de kubernetes que se encuentra dentro de un container de Docker, esta es un poco más sencillo ya que solo utiliza una base de datos que en este caso se decidió utilizar mysql ya que se acopla muy bien con el container y se puede comunicar con el resto de servicios como lo es el API sin problema alguno, aparte que es una de las bases que los desarrolladores tienen más conocimiento.



API analizador

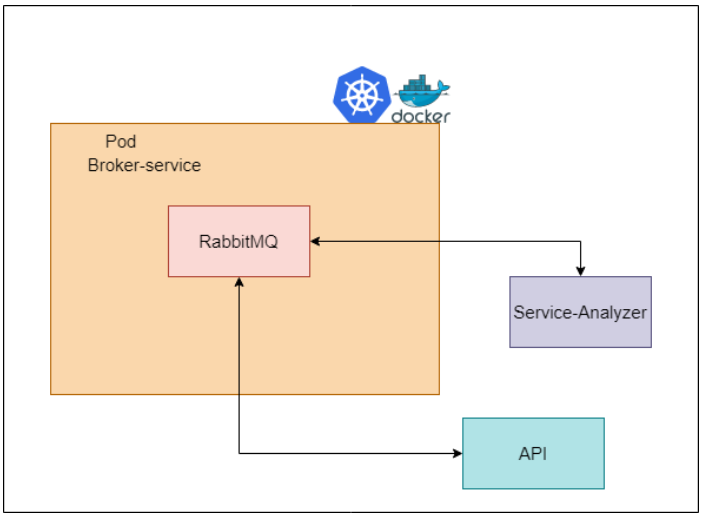
Para este servicio se utiliza un API de Google llamado visión, el cual se encarga de tomar una imagen y analizar el estado de la persona que se encuentra en la fotografía esto con ayuda de un script de Python para poder dar una respuesta un poco mas exacta. Los datos de la imagen los recibe de un json que es tomado de la cola de rabbitmq y la respuesta igual es enviada por otra cola de rabbitmq. Al igual que los demás servicios se encuentra dentro de un container de Docker utilizando kubernetes.



Broker

Broker es un servicio, sin embargo, este es un modelo que se utiliza para mensajería asíncrona. Por lo que es utilizado para manejar los mensajes entre la página web y el analizador, por lo que de esta forma si uno se encuentra ocupado el dato puede ser guardado en la cola sin ser perdido. Para la realización del broker se utilizó la ayuda

de rabbitMQ y al igual que los demás se encuentra en container usando kubernetes para un mejor desempeño.



Arquitectura orientada a eventos

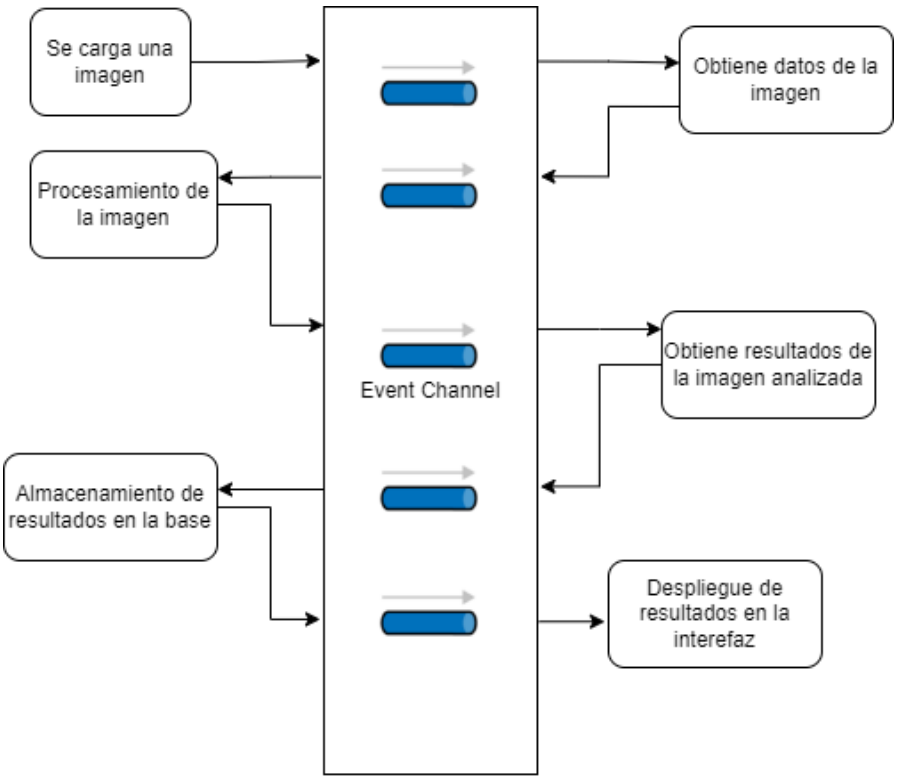


Diagrama de eventos

En el siguiente diagrama se muestra los pasos que debe de seguir la aplicación para poder realizar la tarea como es solicitado, por lo que se puede ver de una mejor manera y más dinámica los pasos y los servicios que lo ejecutan.

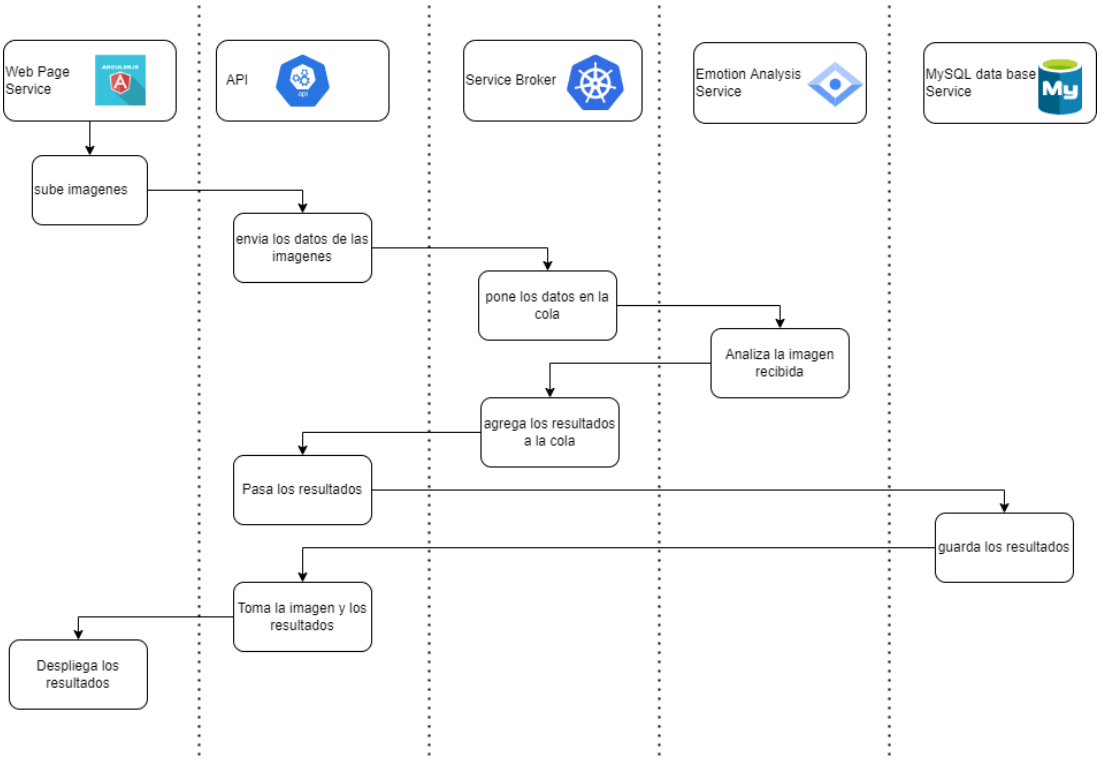


Diagrama de componentes

En el siguiente diagrama se muestra los componentes especificados en el proyecto realizado, como se observa cada componente es un servicio y el api se encuentra conectando el broker con la página principal ya que no fue posible conectarlo todo por rabbit.